MARK GRAHM - PROGRAMMING LANGUAGES ACADEMY

THE BIGGEST PYTHON PROGRAMMING CRASH COURSE FOR BEGINNERS

Learn Python Coding Like a PRO in 7 Days! + Hands-On Workbook

PYTHON

< FOR BEGINNERS >

2023 EDITION

Python for Beginners

The Biggest Python Programming Crash Course for Beginners | Learn Python Coding Like a PRO in 7 Days! + Hands-On Workbook

PROGRAMMING LANGUAGES ACADEMY

Python for Beginners

Table of Contents

Python Programming for Beginners

The Ultimate Beginner’s Guide to Learning the Basics of Python in a Great Crash Course Full of Notions, Tips, and Tricks

PROGRAMMING LANGUAGES ACADEMY

Introduction

Programming has come a long way. The world of programming may have started quite some time ago; it was only a couple of decades ago that it gained attention from computer experts from across the globe. This sudden shift saw some great minds who contributed to the entire age of programming far more significant than most. We saw the great GNU project take shape during this era. We came across the rather brilliant Linux. New programming languages were born as well, and people certainly enjoyed these to the utmost.

While most of these programming languages worked, there was something that was missing. Surely, something could be done to make coding a less tedious task to do and carry out. That is precisely what a revolutionary new language, named after Monty Python's Flying Circus, did for the world. Immediately, coding became so much easier for programmers. The use of this language started gaining momentum, and today, it is set to overtake the only language that stands before it to claim the prestigious spot of being the world's most favored language.

This language was the brainchild of Guido Van Rossum. Created in the year 1991, Python has become a byword for efficient and user-friendly programming. This language is what connected the dots and gave programmers the much-needed ease of coding that they have since been yearning for. Naturally, the language was received well by the programming community. Today, it is one of the most critical languages for both professionals and students who aim to excel in fields like Machine Learning, automation, artificial intelligence, and so much more.

With real-life examples showing a wide variety of use, Python is now living and breathing in almost every major social platform, web application, and website. All of this sounds interesting and exciting at the same time, but what if you have no prior knowledge about programming? What if you have no understanding of basic concepts and you wish to learn Python?

I am happy to report that this book will provide you with every possible chance to learn Python and allow you to jump-start your journey into the world of programming. This book is ideally meant for people who have zero understanding of programming and/or may have never coded a single line of programs before.

I will walk you through all the basic steps from installation to application. We will look into various aspects of the language and hopefully provide you with real-life examples to further explain the importance of such aspects. The idea of this book is to prepare you as you learn the core concepts of Python. After this book, you should have no problem choosing your path ahead. The basics will always remain the same, and this book ensures that each one of those basic elements is covered in the most productive way possible. I will try to keep the learning process as fun as I can without deviating from the learning itself.

Things You Need!

“Wait. Did you not say I don’t need to know anything about programming?” Well, yes! You do not have to worry about programming or their concepts at the moment, and when the time comes, I will do my best to explain those. What is needed of you is something a little more obvious.

- Computer: Like I said, obvious! You need a machine of your own to download and practice the material and matter you learn from here. To make the most out of the book, practice as you read. This greatly increases your confidence and allows you to keep a steady pace. The specifications do not matter much. Most modern machines (2012 and above) should be able to run each of the components without posing any problem.
- An internet connection: You will be required to download a few files from the internet.
- An Integrated Development Environment (IDE): If, for some reason, you felt intimidated by this terminology, relax! I will be guiding you through each and every step to ensure you have all of

these and know what they are all about. For now, just imagine this as a text editor.

- A fresh mind: There is no point in learning if your mind is not there with you. Be fresh, be comfortable. This may take a little practice and a little time, but it will all be worth it.

That is quite literally all that you need. Before we go on into our very first chapter of the book and start learning the essentials, there is but one more thing I would like to clarify right away.

This Book Only Covers the Basics

Suppose you picked up a copy of this book or are considering it, under the impression that the book will teach you all the basics about Python, a good choice! However, if you are of the idea that by the end of the book, you will turn out to be a fully trained professional with an understanding of things like machine learning and other advanced Python fields, please understand that this would fall outside the scope of this book.

This book is to serve as a guide, a crash course of a sort. To learn more advanced methods and skills, you will first need to establish command over all the basic elements and components of the language. Once done, it is highly recommended to seek out books that are for advanced learning.

What I can recommend you to do is to continue practicing your codes after you have completed the book. Unlike driving and swimming, which you will remember for the rest of your life, even if you stop doing them, Python continues to update itself. It is essential that you keep yourself in practice and continue to code small programs like simple calculators, number predictors, and so on. There are quite a few exercises you can come across online.

For advanced courses, refer to Udemy.com. It is one of the finest sources to gain access to some exceptional courses and learn new dimensions of programming, amongst many other fields.

Phew! Now that this is out of the way, I shall give you a minute to flex your muscles, adjust your seat, and have a glass of water; we are ready to begin our journey into the world of Python.

Chapter 1

Python - The First Impressions

So, you have heard about a language that everyone is going mad about. They say it is the language of the future and how incredible it is. You sit with your friends, and all they have to talk about is essentially gibberish to you, and yet it seems interesting to the rest of them. Perhaps you plan to lead a business, and a little research into things reveals that a specific language is quite a lot in demand these days. Sure enough, you can hire someone to do the job for you, but how would you know if the job is being done the way you want it to be, top-notch in quality and original in nature?

Whether you aim to pursue a career out of this journey, you are about to embark on or set up your own business to serve hundreds of thousands of clients who are looking for someone like you; you need to learn Python.

When it comes to Python, there are so many videos and tutorials that you can find online. The problem is that each seems to be heading in a different direction. There is no way to tell what structure you need to follow, where you should begin, and where it should end. There is a good possibility you might come across a video that seemingly answers your call, only to find out that the narrator is not explaining much, and pretty much all you see, you have to guess what it does.

I have seen quite a few tutorials like that myself. They can be annoying and some even misleading. There are programmers who will tell you that you are already too late to learn Python and that you will not garner the kind of success you seek out for yourself. Let me put such rumors and ill messages to rest.

- **Age** – It is just a number. What truly matters are the desire you have to learn. You do not need to be X years old to learn this effectively. Similarly, there is no upper limit of Y years for the learning process. You can be 60 and still be able to learn the language and execute brilliant commands. All it requires is a mind that is ready to learn and a piece of good knowledge on how to operate a computer, open and close programs, and download stuff from the internet. That's it!

- **Language** – Whether you are a native English speaker or a non-native one, the language is open to all. As long as you can form basic sentences and make sense of them, you should easily be able to understand the language of Python itself. It follows something called the "clean-code" concept, which effectively promotes the readability of codes. We will look into that later on, I promise.
- **Python is two decades old already** – If you are worried that you are two decades late, let me remind you that Python is a progressive language in nature. That means, every year, we find new additions to the language of Python, and some obsolete components are removed as well. Therefore, the concept of "being too late" already stands void. You can learn today, and you will already be familiar with every command by the end of a year's time. Whatever has existed so far, you will already know. What would follow then, you will eventually pick up. There is no such thing as being too late to learn Python.
- **Difficulty in understanding** – Remember how hard it was for us to learn the numbers, the alphabet, sentences, and grammar? Everything needs practice, and so does Python. However, if you think Python is rocket science, you are in for a massive surprise. Children are being taught Python in schools, and I am not kidding here just to make you feel at ease. Two rather famous books have existed to support my claim. Search for ***Python For Kids****: A Playful Introduction to Programming* by Jason R. Briggs, and ***Teach Your Kids To Code:*** *A Parent-Friendly Guide to Python Programming* by Bryson Payne. See my point? If children, who have limited exposure to things and a developing mind, can learn the so-called complicated language, why can't you?

Of course, there are people who have been successful, and then there are those who haven't been that. Everything boils down to how effectively and creatively you use the language to execute problems and solutions. The more original your program is, the better you fare off.

“I vow that I will give my best to learn the language of Python and master the basics. I also promise to practice writing codes and programs after I am done with this book.”

Bravo! You just took the first step. Now, we are ready to turn the clock back a little and see exactly where Python came from. If you went through the introduction, I gave you a brief on how Python came into existence, but I left out quite a few parts. Let us look into those and see why Python was the need of the hour.

Python: The Need of the Hour

Before the inception of Python and the famous language that it has gone on to become, things were quite different. Imagine a world where programmers gathered from across the globe in a huge computer lab. You have some of the finest minds on the planet working together towards a common goal, whatever that might be. Naturally, even the finest intellectuals can end up making mistakes.

Suppose one such programmer ended up creating a program, and he is not too sure of what went wrong. The room is full of other programmers, and sure enough, approaching someone for assistance would be the first thought of the day. The programmer approaches another busy person who gladly decides to help out a fellow intellectual programmer. Within that brief walk from one station to the other, the programmer quickly exchanges the information, which seems to be a common error. It is only when the programmer views the code that they are caught off-guard. This fellow member has no idea what any of the code does. The variables are labeled with what can only be defined as encryptions. The words do not make any sense, nor is there any way to find out where the error lies.

The compiler continues to throw in error after error. Remember, this was well before 1991 when people did not have IDEs, which would help them see where the error was and what needed to be done. The entire exercise would end up wasting hours upon hours just to figure out that a semi-colon was missing. Embarrassing and absolutely time-wasting!

This was just a small example; imagine the entire thing but on a global scale. The programming community struggled to find ways to write codes that could be understood easily by others. Some languages supported some syntaxes, while others did not. These languages would not necessarily work in harmony with each other, either. The world of programming was a mess. Had Python not come at the opportune moment that it did, things would have been so much more difficult for us to handle.

Guido Van Rossum, a Dutch programmer, decided to work on a pet project. Yes, you read that, right? Mr. Van Rossum wanted to keep himself occupied during the holiday season and, hence, decided to write a new interpreter for a language he had been thinking of lately. He decided to call the language Python, and contrary to popular belief, it has nothing to do with the reptile itself. Tracing its root from its predecessor, the ABC, Python came into existence just when it was needed.

For our non-programming friends, ABC is the name of an old programming language. Funny as it may sound, naming conventions weren't exactly the strongest here.

Python was quickly accepted by the programming community, albeit there is the fact that programmers were a lot less numerous back then. Its revolutionary user-friendliness, responsive nature, and adaptability immediately caught the attention of everyone around. The more people vested their time into this new language, the more Mr. Van Rossum started investing his resources and knowledge to enhance the experience further. Within a short span of time, Python was competing against the then leading languages of the world. It soon went on to outlive quite a few of them owing to the core concept brought to the table: ease of readability. Unlike any other programming language of that time, Python delivered codes that were phenomenally easy to read and understand right away.

Remember our friend, the programmer, who asked for assistance? If he were to do that now, the other fellow would immediately understand what was going on.

Python also acquired fame for being a language that had an object-oriented approach. This opened more usability of the language to the programmers who required an effective way to manipulate objects. Think of a simple game. Anything you see within it is an object that behaves in a certain way. Giving that object that ‘sense’ is object-oriented programming (OOP). Python was able to pull that off rather easily. Python is considered a multi-paradigm language, with OOP being a part of that as well.

Fast forward to the world we live in, and Python continues to dominate some of the cutting-edge technologies in existence. With real-world applications and a goliath of a contribution to aspects like machine learning, data sciences, and analytics, Python is leading the charge with full force.

An entire community of programmers has dedicated their careers to maintaining Python and developing it as time goes by. As for the founder, Mr. Van Rossum initially accepted the title of Benevolent Dictator for Life (BDFL) and retired on 12 July 2018. This title was bestowed upon Mr. Van Rossum by the Python community.

Today, Python 3 is the leading version of the language alongside Python 2, which has its days numbered. You do not need to learn both of these in order to succeed. We will begin with the latest version of Python as almost everything that was involved in the previous version was carried forward, with the exception of components that were either dull or useless.

I know, right about now, you are rather eager to dive into the concepts and get done with history. It is vital for us to learn a few things about the language and why it came into existence in the first place. This information might be useful at some point in time, especially if you were to look at various codes and identify which one of those was written in Python and which one was not.

For anyone who may have used languages like C, C++, C#, or JavaScript, you might find quite a few similarities within Python and some major improvements too. Unlike in most of these languages, where you need to use a semicolon to let the compiler know that the line has ended, Python needs none of that. Just press enter, and the program immediately understands that the line has ended.

Before we do jump ahead, remember how some skeptics would have you believe it is too late to learn Python? It is because of Python that self-driving cars are coming into existence. Has the world seen too much of them already? When was the last time you saw one of these vehicles on the road? This is just one of a gazillion possibilities that lay ahead for us to conquer. All it needs is for us to learn the language, brush up on our skills, and get started.

"A journey to a thousand miles begins with the first step. After that, you are already one step closer to your destination."

Okay, I added that last sentence by myself, but it is only to provide you with all the confidence you need to learn this language. Most of the things we will visit within Python would make sense right away as Python main uses English sentences and allows for greater ease when it comes to readability. However, there is no reason why you should rush into things. Take your time and practice as much as you can using the exercise book, only after you have learned something through this book. While you can always read through the book from end to end, it is advisable to practice these codes as you learn them.

Last-Minute Tips

You have made it thus far and have learned all there is to, or at least all the important history bits. The time has come for us to start our journey and start typing some lines which, initially, might not make sense, but soon will start to, and you will certainly enjoy the journey. To make the most of the journey, here are a few things you need to ensure:

- While it is encouraged to practice as you read, it is highly unlikely that you will need to type in exactly the same code every time. It is, therefore, important to start playing with the code a little once you get the hang of things. This also increases your confidence and allows you to create your own unique programs. If you see me using a variable called name and I have assigned it a value of

‘Sam,’ feel free to change that to your own name. Change the numbers where and when possible to see how it changes the results.

- It is very much important for you to remain up to date with the latest version of Python. We will be looking into a bit about how to find the latest version of Python, and once we do, keep checking the official Python website to see if a new version has arrived.
- Do not be bogged down with problems as you will encounter quite a lot of them. I will deliberately place a few hurdles to test your knowledge as well, and I will solve those too. Learning how to overcome errors and understand them is a significant way to learn things. Should you encounter an error other than the ones covered within this book, feel free to seek help from the ever-active community of Python.
- Practice a few hours a day if you really wish to become a programmer and pursue a career in Python. It takes a significant time to master all the codes, the functions, the libraries, and much more.
- Do not skip chapters – that is a bad idea unless you genuinely know your way around.
- Do not compare yourself with others. Everyone is born with a unique mind, and so are you. If one is learning quicker than you, let them. Learn at a pace you can keep up with.
- Learn from one source at a time. Learning from various sources at the same time would get you into a state of confusion.
- If you do not understand something, try revisiting the basics or the previous chapters. You may have missed something back there.

With that said, the time is upon us to commence our journey. Ladies and gentlemen, start your engines! We are about to embark on a ride of a lifetime!

Chapter 2

Commencing the Digital Journey

Well, well, well! You have made it this far, and I am rather excited to welcome you to the beginning of your journey into the world of Python. There is every reason for you to be as excited as I am because learning a new language is always full of intriguing moments, challenges, and thought-provoking puzzles that we learn about and then solve.

Unlike most languages, Python is far too easy to make sense of, thanks to our friend Mr. Van Rossum. Let us not waste his efforts and get straight to business.

I am assuming you already have your laptop/computer ready to begin the process and install Python. For the purpose of demonstration, I will be using a Windows 10 operating system. This should work equally well with Windows 7 and Windows 8 users. For users who are using Mac or Linux, here’s the first piece of news. You already have Python installed within your systems as default.

“Wait. What?” If that is your reaction, you are probably wondering why you never came across an icon that has the magical word ‘Python’ written underneath it.

For Mac and Linux users, only

Go ahead and open up your terminal/console app from the app drawer. It looks like a black window with a > sign. For Windows users, although you will receive an error since curiosity is already creeping up, hit the start key and type in ‘cmd’ to bring up a command prompt.

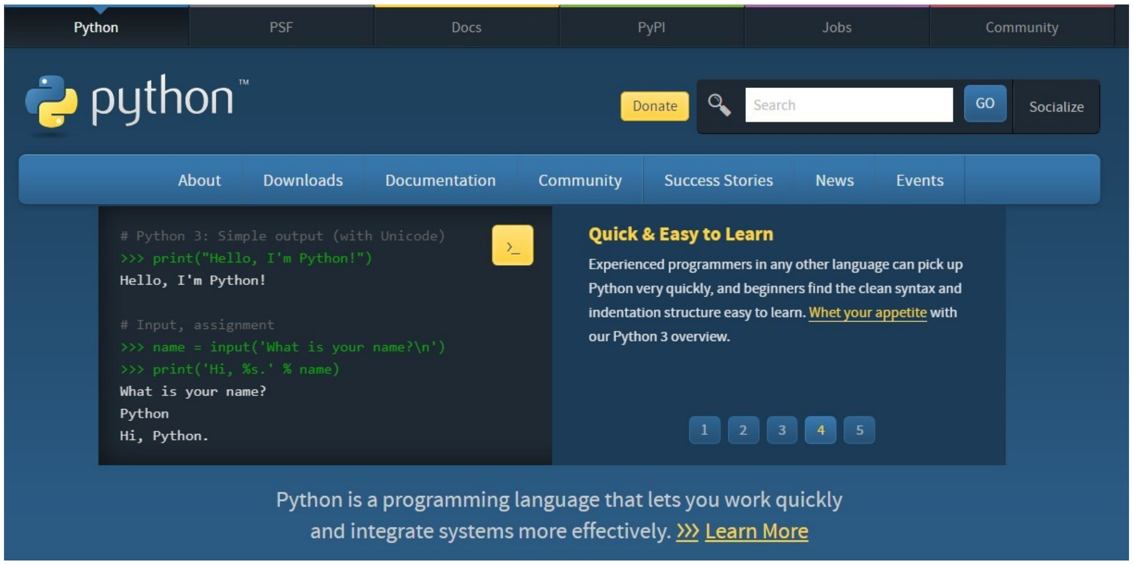
Once in, simply type ‘python.’ Mac and Linux users will immediately be presented with the version number. Usually, you will find Python 2 installed. In my case, since I already have a recent version, this is the result I get:

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.
```

Installing Python

The first order of business is to download the latest version of Python. The version that is available at the time of writing this book is Python 3.8.0, so we will be using that. For any future versions, there is a good chance all of the commands and features will work exactly the same unless the community decides to change the way the language works radically.

Let us begin by opening up our web browser and going straight to the source. In the address bar, type in www.python.org, and you will be greeted by a simplistic website as shown here:



```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!

# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Hover the mouse cursor over ‘Downloads’, and the website should be able to detect your platform and present the corresponding version accordingly automatically. Click on the button to commence the download.

If it doesn’t, simply click on the name of the platform you are using to be taken to the downloads page. Here, click on the first option that says “Latest Python 3 Release – Python 3.8.0” and download that.

Once the file has been downloaded, simply run it to install Python within your system. This should not take more than a minute or so. Now run the command prompt/terminal once again and type in ‘python’ to see the version of Python installed within your machine. This, therefore, confirms that the installation went well.

For Linux users, depending on the kind of flavor you are using, you will need to execute certain commands. Here’s a step-by-step process on how you can get Python 3.8.0 on your system.

Step 1: Enter the following commands on your terminal.

```
sudo apt-get install build-essential checkinstall
```

```
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev libffi-dev zlib1g-dev
```

This essentially installs the libraries which are needed to ensure the successful installation of Python 3.8. If asked to choose Yes or No, choose Yes and proceed.

Step 2: Once sorted, you will now need to enter the following commands:

```
Sudo wget https://www.python.org/ftp/python/3.8.0/Python-3.8.0.tgz
```

This will download the necessary file you need directly to your current working directory. Once the file is downloaded, you will need to extract it, and to do so, enter the following command:

```
sudo tax xzf Python-3.8.0.tgz
```

Step 3: Now, we will run some codes to install Python using ‘altinstall’, which is a more secure way of doing things. First, enter the Python-3.8.0

folder by entering the following command:

```
cd Python-3.8.0
```

Next, you will need to run these two commands. Please note, that this will take some considerable time, so do grab a bit of a break if you like.

```
sudo ./configure –enable-optimization
```

```
sudo make altinstall
```

Step 4: The long, arduous, and extensive installation process has finally ended. It is time to check the version of Python to verify we have the right one installed.

```
Python3.8
```

And voila! You have yourself the latest version of Python installed on your machine. Now, we can resume the preparations.

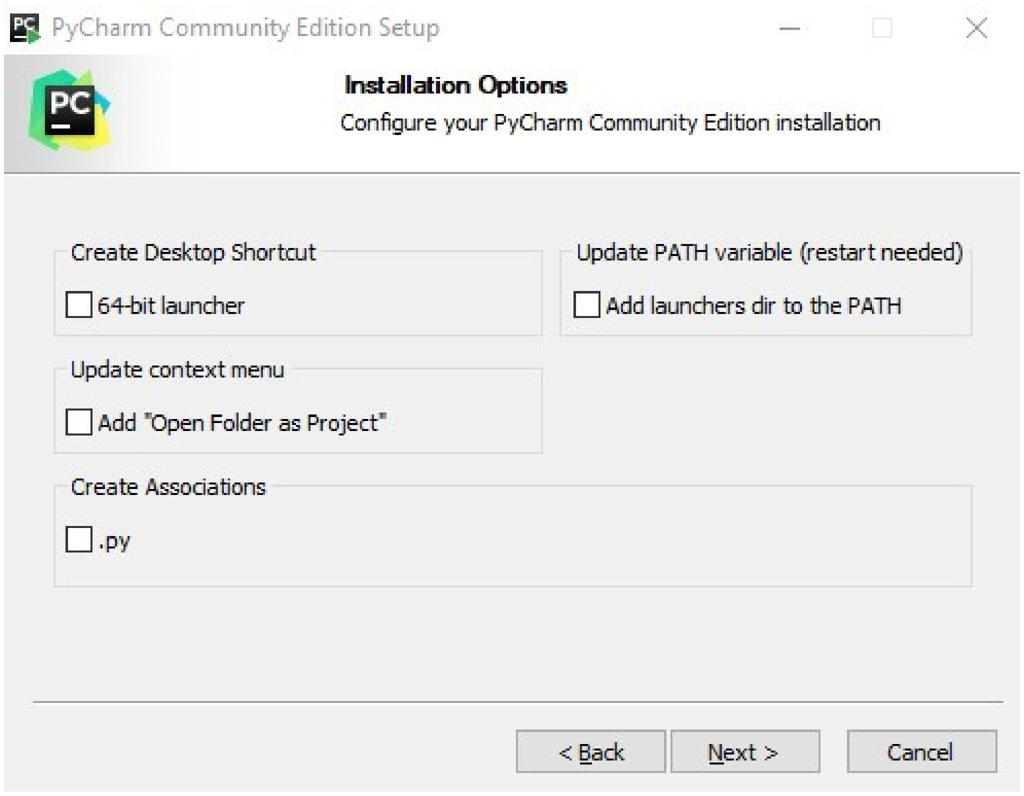
All computer users, regardless of their operating system, are now in line and have Python up and running. The problem is, no one can find an application for that. There is a reason for that. We haven't yet installed that "Text Editor" we spoke about earlier.

In order for us to run Python and be able to use it to write programs, we need an integrated development environment (IDE). For Python, we will be using one of the finest, if not the best, in the business. It is called the PyCharm, and it is all that you need to get started.

To download PyCharm, browse to https://www.jetbrains.com/pycharm/, and click on download. On the next screen, click on the community edition, as it is free, and we aren't exactly using it for large-scale operation, at least not yet. The download should start automatically. Once done, open the file to start the installation process.

Important!

For Windows users, at the start of the installation, you will come across a screen that gives you multiple options, as shown below. Be sure to check the "Add Launchers dir to the PATH." If you do not check this, you may run into issues later on.


PyCharm Community Edition Setup

Installation Options

Configure your PyCharm Community Edition installation

Create Desktop Shortcut

☐ 64-bit launcher

Update PATH variable (restart needed)

☐ Add launchers dir to the PATH

Update context menu

☐ Add "Open Folder as Project"

Create Associations

☐ .py

Continue with the installation to complete the process. You might be required to restart your system after the installation is done. If prompted, it is best to do so. Once the installation is complete, it is time to open up PyCharm for the first time.

Go ahead and click on the freshly added icon of PyCharm on your desktop to access the program. The program might have a delayed start as it is the first time it will run. Once started, you will be presented with a few quick settings. You can modify the looks and the feel and check out other components while you are at it. Do not roam around options that you may have no idea about. It is best to avoid doing something that may end up causing issues later on.

If you come across any option about the Python Interpreter, leave it be!

Finally, you have a window that will ask you to begin a new project. Begin by choosing the directory/folder where you would like to save this project and give it a name. Traditionally, we stick to the name of the first program we create, which for us programmers, is the ‘Hello World’ program. But I did promise this book to be fun, didn’t I? Go ahead and use any name you like and proceed.

Getting Ourselves Acquainted with the IDE

Right now, the fresh new feel of PyCharm is only adding to your curiosity. You know you are just a few keystrokes away from commencing your journey to become a programmer. But before you do that, it is vital for us to address the elephant in the room, or at least this, in this case, the options and a few faded buttons.

If you were expecting something futuristic, preferably with a massive number of buttons and icons, sorry!

We will not be going through each one of these as that would take quite a lot of time, and frankly, you would not even need to know all of these functions at this point in time. Once you are a well-versed Python programmer, you might want to refer to a manual or a book that provides you with all the references and resources you need to learn PyCharm’s complete interface, uses, and functions. It is best to stick to the concept of “This is just a text editor where we write codes, and it does something for us.”

I do not know what color theme you opt for, but I prefer mine to look dark. That way, code pops out and makes it easier for me to read. Yours might be white, and that is okay as well. The point here is to make yourself comfortable with the IDE. The sooner that happens, the quicker you learn.

For us to be acquainted, we only need to learn a few functions and buttons. The rest can gather digital dust as we will rarely ever use those. Let us focus on a few things.

The left pane, or the sidebar, is where your projects are. Right underneath the word ‘project,’ you should be able to see the name you gave to the project as a folder. Underneath that are the external libraries you may be using. For now, don’t worry about what these are; we will be covering them in detail near the end. At the very top, you should see the regular options, followed by some new ones. Should you wish to change the theme, add components, check other settings, click on the file, and you should find everything you are looking for in Settings. Apart from this, the only other menu we are interested in is ‘Run.’ This is where the program we write is executed. There are keyboard shortcuts, and it might even be wise to remember them, starting now.

“Great! So, where do I write the code? I don’t see any space.”

Right-click on the project folder with the name you chose earlier on. Hover your mouse to ‘New’ and choose ‘Python File’ and name your first-ever program. Let’s call this **Test1.py** for the sake of learning.

.py is an extension that shows the file is written in Python and must be executed by Python, just like .docx, .tar, and so on.

The second you hit enter, you will immediately have that large chunk of the screen in the middle vacant. This is where you will be writing all your codes for programs that will go on to make history. And, to your surprise, there’s one in the making right now. How? Go ahead and type in the following code:

```
print(“I made it!”)
```

Hmm! You pressed enter, and nothing happened. Well, that’s no fun, is it? Remember the ‘Run’ menu? Go ahead and click on it and choose, well, ‘Run’ to initiate the program. Another window would pop-up from the bottom and show this:

```
"C:\Users\Programmer \AppData\Local\Programs\Python\Python37-32\python.exe" "C:/Users/Programmer /PycharmProjects/PFB/Test1.py"

I made it!
```

```
Process finished with exit code 0
```

Two things just happened here. First, you may not have realized this, but you just made your first-ever program. Congratulations are in order! You just commanded your computer to print out a message of your choice.

Second, this little box that appeared from the bottom is called the console, and you have just discovered the last piece in the puzzle. The console is where the information, the program, and the result gets printed.

Now, you have seen everything you need to know to get started. You do not need to roam around any other options. Even the ‘Run’ menu will no longer be used, at least not all the time anyway. You now should be able to see the green play button on the top right corner, next to the little dialog box that has your program’s name on it. Next time, simply click that, and you should be able to execute a program.

Craving to type a little more? Go ahead, type the following lines, and have a blast running these.

```
print("I made it!")

print("======")

print("If I only knew it would be this easy")

print("======")

print("I would have taken up Python ages ago")

print("======")

```

```
Output:

I made it!

======

If I only knew it would be this easy
```

======

I would have taken up Python ages ago

======

Two programs already? Nice pace! It certainly seems like we are ready to dive deeper into the world of Python and start learning what exactly things like the ‘print’ command do, why we use quotation marks, and all that goodness.

You have installed Python, and you have grabbed your copy of PyCharm as well. You just created two simple programs that are able to print out information to the console accordingly to what you type in. We are now fully operational, ready, and open for business.

Remember, you cannot expect to become an overnight programming sensation. Programming of any kind takes time to master and develop. You cannot expect to learn anything new and become a master within a short span of time. Squeeze some time out from your daily routine. Maybe stop playing too many games, spending time pointlessly scrolling up and down on social media sites, or even going out with friends every day. If you put your heart and soul here for the next year, you will be thanking me for ages to come.

With that said, let’s take our first step on that journey of a thousand miles. Scratch that; you already took a few steps. Let’s just continue the momentum and hope to add a skill to our name that will serve us for times to come. Let us learn Python.

Chapter 3

Learning Python from Scratch

Even before buying this book, you had some idea as to how important and in-demand Python is. We also read a little about it earlier on and saw how it is overtaking many mainstream programming languages. We walked through a step-by-step guide to download Python 3, the integrated development environment, or the text/code editor and set everything up. Lastly, you created your first program: well done!

Now, it is time to stop scratching the surface and dive deep into the world of Python. There are far too many components and aspects to learn about Python, but we will only be focusing on what is essential for anyone to know and learn as a beginner.

Consider this as grammar for any other language. Without grammar, the language sounds broken, and so does Python.

Python at First Glance

Let us begin with the program that we just created in the last chapter. To remind ourselves what it was, here's a quick look at the commands we wrote:

```
print("I made it!")

print("======")

print("If I only knew it would be this easy")

print("======")

print("I would have taken up Python ages ago")

print("======")
```

We used a print command to have our message printed at the console box as our output of the program. Calling ‘print’ a command is technically wrong; it is a function. While we will be covering functions and methods in detail later on, for now, just remember that functions are names of commands which are followed by the parenthesis “()” where the brackets will either be empty or contain some type of data. There are set parameters that are pre-defined, meaning that certain functions will only be able to accept a specific type of data.

In the above example, we used nothing more than text. A few letters to create a message, and that is it. In Python, things work differently. Text is not identified as text. We need to tell Python that we want this to be printed as text. How do we do that? We use single or double quotation marks, which allows Python to understand that anything within the quotes is text, and it needs to print it the way it is.

I bet most of you may not have noticed how all of the lines start with a lowercase ‘p’ instead of the opposite. Ah, yes! Now that you noticed it let me tell you why we did that.

Python is a case-sensitive language. It considers everything as a character, not a letter or text. This means that the lowercase ‘p’ will not be the same character as the uppercase ‘P’ and so on, and so forth.

```
Print
```

```
PRINT
```

```
print
```

```
PrinT
```

```
pRINt
```

All of these will be treated differently by Python, and for printing purposes, these will not work at all except for ‘print’ as that is the standard way of outputting things.

To name anything in Python, we normally use lower cases for one-word commands. This isn’t something that is exclusive to Python, as every language uses some way as a standard to write codes. What makes Python different is the sheer amount of thought that was put into the naming convention to make code easier to read. Remember this, anything with more than the word, you can use a few ways to do so as shown here:

```
last_name
```

```
LastName
```

```
lastname
```

```
LASTNAME
```

In most of the cases, we will be using the first approach, where each letter begins with a lowercase. For components with more than one word, we will be using underscores to separate them. The next in line is generally used only in cases of classes. At this point in time, you do not have to worry about what classes are. Just remember those words with the first letter as capital and that having no underscores is an example of Camel Case and used for classes.

Next down the line is the way we use to name packages. Here, all the words begin and end with lowercase letters and have no underscores between them. On the polar opposite, we have our last entry, which is used to define constants. Here, all the letters are in uppercase and have no underscores separating the words.

Boring, wasn’t it? I know! But it is something you may want to remember as we will be doing quite a lot of these. You should know when to use which convention, as this greatly improves the code readability. The entire point of Python is to promote code readability, and if we go against that, there’s not much point in learning Python.

Now that we have covered this let us start by discussing data types that are at work within Python. Without these, no programming language would operate or work. They are what we use as inputs, and these are what direct the program per our desire accordingly.

What Are Data Types?

Every program has certain data that allows it to function and operate in the way we want. The data can be a text, a number, or any other thing in between. Whether complex in nature or as simple as you like, these data types are the cogs in a machine that allow the rest of the mechanism to connect and work.

Python is a host to a few data types, and, unlike its competitors, it does not deal with an extensive range of things. That is good because we have less to worry about and yet achieve accurate results despite the lapse. Python was created to make our lives, as programmers, a lot easier.

Strings

In Python and other programming languages, any text values that we may use, such as names, places, and sentences, are all referred to as strings. A string is a collection of characters, not words or letters, which is marked by the use of single or double quotation marks.

To display a string, use the print command, open up a parenthesis, put in a quotation mark, and write anything. Once done, we generally end the quotation marks and close the bracket.

Since we are using PyCharm, IntelliSense detects what we are about to do and delivers the rest for us immediately. You may have noticed how it jumped to the rescue when you only typed in the opening bracket. It will automatically provide you with a closing one. Similarly, for the quotation marks, one or two, it will provide the closing ones for you. See why we are using PyCharm? It greatly helps us out.

“I do have a question. Why do we use either single or double quotation marks if both provide the same result?”

Ah! Quite the eye. There is a reason we use these; let me explain by using the example below:

```
print(‘I’m afraid I won’t be able to make it’)
print(“He said “Why do you care?””)
```

Try and run this through PyCharm. Remember, to run, simply click on the green play-like button on the top right side of the interface.

```
"C:\Users\Programmer\AppData\Local\Programs\Python\Python37-32\python.exe" "C:/Users/Programmer/PycharmProjects/PFB/Test1.py"

  File "C:/Users/Programmer/PycharmProjects/PFB/Test1.py", line 1

print('I'm afraid I won't be able to make it')

^

SyntaxError: invalid syntax



Process finished with exit code 1
```

Here’s a hint: That’s an error!

So what happened here? Try and revisit the inputs. See how we started the first print statement with a single quote? Immediately, we ended the quote using another quotation mark. The program only accepted the letter ‘I’ as a string. You may have noticed how the color may have changed for every other character from ‘m’ until ‘won’ after which the program detects yet another quotation mark and accepts the rest as another string. Quite confusing, to be honest.

Similarly, in the second statement, the same thing happened. The program saw double quotes and understood it as a string, right until the point the second instance of double quotation marks arrived. That’s where it did not bother checking whether it is a sentence or that it may have still been going on. Computers do not understand English; they understand binary communications. The compiler is what runs when we press the run button. It compiles our code and interprets the same into a series of ones and zeros so that the computer may understand what we are asking it to do.

This is exactly why the second it spots the first quotation mark, it considers it as a start of a string and ends it immediately when it spots a second quotation mark, even if the sentence was carrying onwards.

To overcome this obstacle, we use a mixture of single and double quotes when we know we need to use one of these within the sentence. Try and replace the opening and closing quotation marks in the first state as double quotation marks on both ends. Likewise, change the quotation marks for the second statement to single quotation marks as shown here:

```
print("I'm afraid I won't be able to make it")

print('He said "Why do you care?"')
```

Now the output should look like this:

```
I'm afraid I won't be able to make it

He said "Why do you care?"
```

Lastly, for strings, the naming convention does not apply to the text of the string itself. You can use regular English writing methods and conventions without worries, as long as that is within the quotation marks. Anything outside it will not be a string in the first place and will or may not work if you change the cases.

Did you know that strings also use triple quotes? Never heard that before, have you? We will cover that shortly!

Numeric Data type

Just as the number suggests, Python is able to recognize numbers rather well. The numbers are divided into two pairs:

- **Integer** – A positive and/or negative whole numbers that are represented without any decimal points.
- **Float** – A real number that has a decimal point representation.

This means, if you were to use 100 and 100.00, one would be identified as an integer while the other will be deemed as a float. So why do we need to use two various number representations?

If you are designing a program, suppose a small game that has a character's life of 10, you might wish to keep the program in a way that whenever a said character takes a hit, his life reduces by one or two points. However, to make things a little more precise, you may need to use float numbers. Now, each hit might vary and may take 1.5, 2.1, or 1.8 points away from the life total.

Using floats allows us to use greater precision, especially when calculations are on the cards. If you aren't too troubled about the accuracy, or your programming involves whole numbers only, stick to integers.

Booleans

Ah! The one with the funny name. Boolean (or bool) is a data type that can only operate on and return two values: True or False. Booleans are a vital part of any program, except the ones where you may never need them, such as our first program. These are what allow programs to take various paths if the result is true or false.

Here's a little example. Suppose you are traveling to a country you have never been to. There are two choices you are most likely to face.

If it is cold, you will be packing your winter clothes. If it is warm, you will be packing clothes which are appropriate for warm weather. Simple, right? That is exactly how the Booleans work. We will look into the coding aspect of it as well. For now, just remember, when it comes to true and false, you are dealing with a bool value.

List

While this is slightly more advanced for someone at this stage of learning, the list is a data type that does exactly what it sounds like. It lists objects, values, or stores data within square brackets ([]). Here's what a list would look like:

```
month = ['Jan', 'Feb', 'March', 'And so on!']
```

We will be looking into this separately, where we will discuss lists, tuples, and dictionaries.

We have briefly discussed these data types. Surely, they are used within Python, but how? If you think you can type in the numbers and true and false, all on their own, it will never work.

Variables

You have the passengers, but you do not have a mode of commuting; they will have nowhere to go. These passengers would just be folks standing around, waiting for some kind of transportation to pick them up. Similarly, data types cannot function alone. They need to be 'stored' in these vehicles, which can take them places. These special vehicles, or as we programmers refer to as containers, are called 'variables,' and they are exactly what perform the magic for us.

Variables are specialized containers that store a specific value in them and can then be accessed, called, modified, or even removed when the need arises. Every variable that you may create will hold a specific type of data in them. You cannot add more than one type of data within a variable.

In other programming languages, you will find that in order to create a variable, you need to use the keyword 'var' followed by an equals mark '=' and then the value. In Python, it is a lot easier, as shown below:

```
name = "John"

age = 33

weight = 131.50

is_married = True
```

In the above, we have created a variable named ‘name’ and given it a value of characters. If you recall strings, we have used double quotation marks to let the program know that this is a string.

We then created a variable called age. Here, we simply wrote 33, which is an integer as there are no decimal figures following that. You do not need to use quotation marks here at all.

Next, we created a variable ‘weight’ and assigned it a float value.

Finally, we created a variable called ‘is_married’ and assigned it a ‘True’ bool value. If you were to change the ‘T’ to ‘t’ the system will not recognize it as a bool and will end up giving an error.

Focus on how we used the naming convention for the last variable. We will be ensuring that our variables follow the same naming convention.

You can even create blank variables in case you feel like you may need these at a later point in time or wish to initiate them at no value at the start of the application. For variables with numeric values, you can create a variable with a name of your choosing and assign it a value of zero. Alternatively, you can create an empty string as well by using opening and closing quotation marks only.

```
empty_variable1 = 0

empty_variable2 = ""
```

You do not have to name them like this necessarily; you can come up with more meaningful names so that you and any other programmer who may read your code would understand. I have given them these names to ensure anyone can immediately understand their purpose.

Now we have learned how to create variables, let’s learn how to call them. What’s the point of having these variables if we are never going to use them, right?

Let’s create a new set of variables. Have a look here:

```
name = "Jonah"
age = 47
height_in_cm = 170
occupation = "Programmer"
```

I do encourage you to use your own values and play around with variables if you like.

In order for us to call the name variable, we simply need to type the name of the variable. In order to print that to the console, we will do this:

```
print(name)

Output
Jonah
```

The same goes for the age, the height variable, and occupation. But what if we wanted to print them together and not separately?

Try running the code below and see what happens:

```
print(name age height_in_cm occupation)
```

Surprised? Did you end up with this?

```
print(name age height_in_cm occupation)
^
SyntaxError: invalid syntax

Process finished with exit code 1
```

Here is the reason why that happened. When you were using a single variable, the program knew what variable that was. The minute you added a second, a third, and a fourth variable, it tried to look for something that was written in that manner. Since there wasn't any, it returned with an error that otherwise says:

"Umm… Are you sure, Sir? I tried looking everywhere, but I couldn't find this 'name age height_in_cm occupation' element anywhere."

All you need to do is add a comma to act as a separator like so:

```
print(name, age, height_in_cm, occupation)

Output:
Jonah 47 170 Programmer
```

"Your variables, Sir!"

And now, it knew what we were talking about. The system recalled these variables and was successfully able to show us what their values were. But what happens if you try to add two strings together? What if you wish to merge two separate strings and create a third-string as a result?

```
first_name = "John"
last_name = "Wick"
```

To join these two strings into one, we can use the '+' sign. The resulting string will now be called a String Object, and since this is Python we are dealing with, everything within this language is considered as an object, thus the Object-Oriented Programming nature that we discussed somewhere in the start.

```
first_name = "John"
last_name = "Wick"
first_name + last_name
```

Here, we did not ask the program to print the two strings. If you wish to print these two instead, simply add the print function and type in the string variables with a + sign in the middle within parentheses. Sounds good, but the result will not be quite what you expect:

```
first_name = "John"
last_name = "Wick"
print(first_name + last_name)

Output:
JohnWick
```

Hmm. Why do you think that happened? Certainly, we did use a space between the two variables. The problem is that the two strings have combined together, quite literally here, and we did not provide a white space (blank space) after John or before Wick; it will not include that. Even the white space can be a part of a string. To test it out, add one character of space within the first line of code by tapping on the friendly spacebar after John. Now try running the same command again, and you should see “John Wick” as your result.

The process of merging two strings is called concatenation. While you can concatenate as many strings as you like, you cannot concatenate a string and an integer together. If you really need to do that, you will need to use another technique first to convert the integer into a string and then concatenate the same. To convert an integer, we use the *str()* function.

```
text1 = "Zero is equal to "
text2 = 0
print(text1 + str(text2))

Output:
Zero is equal to 0
```

Python reads the codes in a line-by-line method. First, it will read the first line, then the second, then third, and so on. This means we can do a few things beforehand as well, to save some time for ourselves.

```
text1 = "Zero is still equal to "
text2 = str(0)
print(text1 + text2)

Output:
Zero is still equal to 0
```

You may wish to remember this as we will be visiting the conversion of values into strings a lot sooner than you might expect.

There is one more way through which you can print out both string variables and numeric variables, all at the same time, without the need for ‘+’ signs or conversion. This way is called String Formatting. To create a formatted string, we follow a simple process, as shown here:

print(f“ This is where {var 1} will be. Then {var 2}, then {var 3} and so on”)

Var 1, 2, and 3 are variables. You can have as many as you like here. Notice the importance of whitespace. Try not to use the spacebar as much. You might struggle at the start but will eventually get the hang of it.

When we start the string, we place the character ‘f’ to let Python know that this is a formatted string. Here, the curly brackets are performing a part of placeholders. Within these curly brackets, you can recall your variables. One set of curly brackets will be a placeholder for each variable that you would like to call upon. To put this in practical terms, let’s look at an example:

```
show = "GOT"

name1 = "Daenerys"

name2 = "Jon"

name3 = "Tyrion"

seasons = 8

print(f"The show called {show} had characters like {name1}, {name2} and {name3} in all {seasons} seasons. ")


Output:

The show called GOT had characters like Daenerys, Jon, and Tyrion in all 8 seasons.
```

While there are other variations to convert integers into strings and concatenate strings together, it is best to learn those which are used throughout the industry as standard.

Remember the triple quotes that I mentioned earlier? I believe you are in a good position now to begin using those.

Have a look at this result, and keep in mind that I did not use any variable here at all.

Now, you have seen how to create a variable, recall it, and concatenate the same. Everything sounds perfect, except for one thing; These are predefined values. What if we need an input directly from the end-user? How can we possibly know that? Even if we do, where do we store them?

User-Input Values

Suppose we are trying to create an online form. This form will contain simple questions like asking for the user’s name, age, city, email address, and so on. There must be some way through which we can allow users to input these values on his/her own and for us to get those back. We can use the same to print out a message that thanks to the user for using the form and that they will be contacted on their email address for further steps.

To do that, we will use the *input()* function. The input function can accept any kind of input. In order to use this function, we will need to provide it with some reference so that the end-user is able to know what he/she is about to fill out.

Let us look at a typical example and see how such a form can be created:

```
print("Hello and welcome to my interactive tutorial.")

name = input("Your Name: ")

age = int(input("Your age: "))

city = input("Where do you live? ")

email = input("Please enter your email address: ")


print(f"Thank you very much {name}, you will be contacted at {email}.")

```

```
Output:

Hello, and welcome to my interactive tutorial.

Your Name: Sam
Your age: 28
Where do you live? London
Please enter your email address: sam@abcxyz.com
Thank you very much, Sam, you will be contacted at sam@abcxyz.com.
```

In the above, we began by printing a greeting to the user and welcoming them to the tutorial. Next, we created a variable named 'name' and assigned it a value that our user will generously provide us with. In the age, you may have noticed I changed the input to *int(),* just as we changed integer to string earlier on. This is because our message within the input parameters is a string value by default, as it is within quotation marks. You will always need to ensure you know what type of value you are after and do the needful, as shown above.

Next, we asked for the name of the city and the email address. Now, using a formatted string, we printed out our final message.

"Wait! How can we print out something we have yet to receive or know?"

I did mention that Python works line by line. The program will start with a greeting, as shown in the output. Then, it will move to the next line and realize that it must wait for the user to input something and hit enter. This is why the input value has been highlighted by bold and italic fonts here. The program then moves to the next line and waits yet again for the user to put something in and press enter, and this goes on until the final input command is sorted. Now the program has the values stored; it immediately recalls these values and prints them out for the viewer to see in the end.

The result was rather pleasing as it gave a personalized message to the user, and we received the information we need. Everybody walks away, happy!

Storing information directly from the user is both essential and, at times, necessary. Imagine a game that is based on Python. The game is rather simple, where a ball will jump when you tap the screen. The problem is, your screen isn't responding to the touch at all for some reason. While that happens, the program will either keep the ball running until input is detected, or it will just not work at all.

We also use input functions to gather information such as login ID and passwords to match with the database, but that is a point that we shall discuss later when we will talk about statements. It is a little more complicated than it sounds at the moment, but once you understand how to use statements, you will be one step closer than ever before from becoming a programmer.

Chapter 4

Introduction to Statements and Loops

I would like to confess something! I actually lost count of the programs you have successfully created so far! Bravo!

Never knew you had it in you until you decided to pick up this book and give it a shot, did you? That is exactly how most of us programmers learned. We were not born as a predefined variable that already knew if it was an engineer, a rocket scientist, a programmer, or a doctor. We were blanks; how we carved our path was our individual choices.

Python offers so much flexibility and ease of understanding that it leaves little room to question just why this language is becoming so popular. Already, you have created a form, a way to create the personalized greeting, and you have learned how to take inputs, direct from the users, and store those in their specific 'containers' and that is no ordinary feat for someone who had no idea about programming just a while ago.

Now, things will get interesting. This is where the part of the crash course starts, where we will be forced to use our heads and think the solutions through. This is where all our previous knowledge will be put to the test. If you have been practicing exercises and following tips and explanations, expect to enjoy your time from this point forward.

Statements: What Are They?

Before I begin explaining what a statement is, let me pose you a simple question. When was the last time you had to choose between two things, depending on the elements like what you prefer, what you can afford, what is near, and what isn't? Whenever we make decisions, we take into account quite a few components and elements which will eventually influence our decision accordingly. Similarly, to help us with such issues, we use statements, and that is exactly what we will be looking into.

In the simplest definition, Statements nothing more than instructions that Python interpreter understands and executes. We have been writing some ourselves when we set values to variables.

Statements, where we assign values to variables, are called assignment statements. However, as long as Python is being discussed, generally, statements refer to ‘if’ statements.

The ‘if’ statement is what provides Python with a situation and allows Python to take appropriate action ‘if’ a given situation is true. Otherwise, it takes another route. Sounds easy, and it is actually interesting too. Let us see how we can create our very first ‘if’ statement.

Here’s the situation. A user wishes to sign in using their account. The prompt asks for the passcode only. If the user inputs the right, case-sensitive, passcode, he should be allowed access. If the user enters the wrong password, it should not go through and inform the user that the entered password was incorrect.

To do that, we first need to establish a password. You can either come up with your own pre-defined one or ask the user to create a new passcode and then re-enter it. I leave the choice up to you.

```
password = input("Create a password: ")

print("Welcome to the portal")
```

So far, I have only asked the user to enter a password of their choice. If you wish, you can set any string or numbers as a password. Next, I created a little welcome greeting. Now, we shall ask the user to enter their password:

```
password_check = input("Please enter your password: ")
```

The only thing worth noting here is that I changed the name of the variable. If you are wondering why that is because had I used the same variable name; it would have updated the password, instead of comparing it. Since we wish to verify the password, we will need to use a different variable.

Now, the customer has given us two pieces of information. Here, we tell Python what to do if the password matches.

```
if password_check == password:

print("Successful! Welcome back!")
```

There are two things to notice here. Whenever you type in ‘if’ as your first word, PyCharm will detect that you wish to create an ‘if’ statement. The color of ‘if’ will change to denote the same. After ‘if,’ we need to define our condition. To do that, you may have observed that I used “==” instead of a single equals sign. These signs are called Operators, which we will discuss later. All you need to know here is this:

‘=’ is used to assign a value

‘==’ is used to either equate two variables or compare to see if the two are exactly the same.

In the above instance, we will use this comparison operator. Here is the most interesting bit; unlike all the codes we wrote so far, this line ends with a colon ‘:’.

Every conditional statement, such as the ‘if’ statement, ends with a colon in Python to create a block of code that will execute under that colon. The next line will begin with an indentation. Do not remove that indent as that would cause confusion. Since I had already set the condition, which quite literally reads as “If password_check is exactly the same as password” and now I added the command that it needs to carry out if the condition is met. When you execute this program, you will begin with the prompt asking you to choose a password. That would be stored as a variable named password.

Next, the prompt will ask us to type in the password once again for verification or for login purposes. Whatever we type here will be stored in a variable called password_check. Now, Python will compare the two values and see if the two are exactly alike. If so, it will print out a success message.

I am quite sure that you have just tried to enter the wrong password deliberately. It ended the program altogether without any warning, right? There is a reason for that. We have only defined the ‘if’ condition. We never got to the part to define the ‘else’ condition.

The ‘else’ condition is the final condition, and it usually comes into play when the ‘if’ condition or others are not true and are not fulfilled. To do that, we will add two lines of code beneath the first one. Now, the entire program should look like this:

```
password = input("Create a password: ")
print("Welcome to the portal")

password_check = input("Please enter your password: ")
if password_check == password:
print("Successful! Welcome back!")
else:
print("Sorry buddy! That's a Nay!")
```

Notice how ‘else’ statement needs no indentation here, and it does not require you to provide additional conditions either.

Now, I will run the code twice. Once correct and the other incorrect. Let us see how it works:

```
Correct password
Create a password: 123
```

```
Welcome to the portal
Please enter your password: 123
Successful! Welcome back!

Incorrect password
Create a password: 123
Welcome to the portal
Please enter your password: 122
Sorry buddy! That's a Nay!
```

Here’s a question, what if there is more than one condition to a statement? Suppose you are to choose a number between one to three and then give an appropriate message, depending on the number the user chooses, how would we do that?

```
print("Welcome to my little game")
number = int(input("Choose a number between 1-3: "))
if number == 1:
print("You love to consider yourself a leader, don't you?")
elif number == 2:
print("You hate being alone, right?")
elif number == 3:
print("The more, the merrier, is it?")
else:
print("Really? You can't follow simple instructions, can you?")
```

Quite a familiar way to put things, but the only thing to note here is the ‘elif’ statement. The ‘elif’ sits right between ‘if’ and ‘else’ where ‘if’ is the first condition, and ‘else’ is when no conditions are met.

Yes, I know! It should’ve been named as ‘ifel’, but then again, it is what it is!

Try it out yourself, check each of these with various numbers as your picks. For a little fun, use any number greater than three and see what happens.

This is how Python handles conditional statements. If you are a bit of a gamer, you may have seen various games where decisions can influence the outcome of the game itself. Now you know the culprit!

There is no limit to the number of ‘elif’ statements. You can create as many as you like. With that said, let’s make this a little more interesting.

Nested Conditional (‘if’) statements

Let us assume that we use the same numbers as above, but this time, we wish to add an ‘if’ statement within an ‘if’ statement. Let’s imagine that we want out user to select another numeric value, this time in decimal numbers, only if the user decides to choose the first value as the number.

Have a look at the code below and try to find out how the code will be executed.

```
print("Welcome to my little game")

number = int(input("Choose a number between 1-3: "))

if number == 1:

print("You love to consider yourself a leader, don't you?")

number2 = float(input("Enter a number with a decimal figure between 1 and 2: " ))

if number2 == 2.00:
```

```
print("Okay! I meant a little lesser than that!")
elif number < 1.50:
print("Oh, come on! You can go higher!")

else:
print("You know what, forget it!")
elif number == 2:
print("You hate being alone, right?")
elif number == 3:
print("The more, the merrier, is it?")
else:
print("Really? You can't follow simple instructions, can you?")
```

We created another variable within the first condition. If the user decides to settle for one, the prompt will ask the user to enter another number. We used the conversion here to convert the incoming number to afloat, as it will have a decimal figure.

We then created another condition which defines the upper limit and the lower limit. To add a little fun to it, there is no correct number to choose from here. Regardless of what the user may choose, they will either receive a message to state that they went a little too high, or one that will encourage them to go higher. The rest will always leave the user in a bit of a puzzled state.

This kind of conditional statement within a conditional statement is called a nested statement. This entire block of code can be avoided if the user decides to go for any other number than the triggering point.

These can sometimes be highly useful. You may have already been using these on Facebook, Netflix, and other major platforms. These are designed to refine the results further.

While we did all this, there was something I was hoping you would ask.

“Why do we have to put in a value and restart the program every time to test out another value?”

It does make sense. It will be highly annoying if your program restarts from the beginning every time it reaches some result, whether positive or not. It would be just like the old days where games on Nintendo would be fun, right until you run out of lives, and the screen says “Game over” and restart from the beginning. Frustrating days, weren’t they?

To address such an issue, we use what are called ‘Loops’, and these are equally as important as the conditional statements, as we saw above.

Loops – The Never-Ending Cycle

Imagine you are creating a program that asks the user to guess a number. The code should ideally run for three times before it could let the user know that they consumed their three chances and failed. Similarly, the program should be smart enough to know if the user guessed the right number, in which case, it would end the execution of the program by displaying “You guessed the right number!”

We use loops to address such situations. Loops are when an entire block of code continues to run over and over again until the condition set is no longer valid. If you forget to set a condition, or if a condition is not properly defined, you may start an endless loop that will never cease, causing the program to crash completely.

Do not worry; your system will not crash. You can end the program by using the red/pink stop button that always magically appears after you hit the green run button.

There are essentially two types of loops we use in Python. The first one is the ‘while’ loop, and the second one is the ‘for’ loop.

The ‘While’ Loop

This type of loop runs a specific block of code for as long as the given condition remains true. Once the given condition is no longer valid, or turns to false, the block of code will end right away.

This is quite a useful feature as there may be codes that you may need to rely on to process information quickly. To give you an idea, suppose, you are to guess a number. You have three tries. You want the prompt to ask the user to guess the number. Once the user guesses the wrong number, it will reduce the maximum number of tries from three to two, inform the user that the number is wrong and then ask to guess another time. This will continue until either the user guesses the right number or the set number of guesses are utilized, and the user fails to identify the number.

Imagine just how many times you would have to write the code over and over again. Now, thanks to Python, we just type it once underneath the ‘while’ loop, and the rest is done for us.

Here’s how the syntax for the ‘while’ loop looks like:

```
while condition:
code
        code
                          …
```

You begin by typing in the word ‘while’ followed by the condition. We then add a colon, just like we did for the ‘if’ statement. This means, whatever will follow next, it will be indented to show that the same is working underneath the loop or the statement.

Let us create a simple example from this. We start by creating a variable. Let’s give this variable a name and a value like so:

x = 0

Nothing fun here, so let us add something to make it more exciting. Now, we will create a condition for a while loop. The condition would state that as long as x is equal to or less than 10, the prompt will continue to print the value of *x*. Here's how you would do that:

```
x = 0

while x <= 10:

print(x)
```

Now try and run that to see what happens!

Your console is now bombarded with a never-ending loop of zeros. Why did that happen? If you look close enough at the code, we only assigned one value to our variable. There is no code to change the value or increase it by one or two or any of that.

In order for us to create a variable that continues to change variable after it has printed the initial value, we need to add one more line to the code. Call it as the increment code, where *x* will increase by one after printing out a value. The loop will then restart, this time with a higher value, print that, and then add one more. The loop will continue until *x* is equal to 10. The second it hits the value of 11, the interpreter will know that the condition no longer remains true or valid, and hence we will jump out of the loop.

```
x = 0

while x <= 10:

print(x)

x = x + 1
```

The last line will execute and recall the current value of *x*, and then it will add one to the value. The result would look like this.

```
1

2
```

```
3
4
5
6
7
8
9
10
```

If you do not like things to add just like that, add a little print statement to say “The End” and that should do the trick.

I almost forgot! If you intend to add a print statement at the end, make sure you hit the backspace key to delete the indentation first.

Let’s make things a little more fun now, and to do that, we will be creating our very first basic game.

Let me paint the scenario first. If you like, pick up a pen and paper, or just open notepad on your computer. Try and write down what you think is the possible solution for this.

The game has a secret number that the end-user cannot see. Let’s assume that the number is set to 19. We will allow the user to have three attempts to guess the number correctly. The game completes in a few possible ways:

1. The user guesses the number correctly before running out of lives.
2. The user runs out of the three chances and is unable to guess the number.
3. The user guesses the number on the final attempt.

Use your imagination and think about what can be the possible code. Once ready, let us proceed to the actual coding for this game and see how this works out to be.

Hint: *Use both a ‘while’ loop and an ‘if’ statement!*

Well done for those who tried. There is no shame in failing to pull this off. I failed to do the same myself until I saw the solution, and I practically kicked myself!

```
my_number = 19
guess = 0
max_guess = 3
while guess < max_guess:
number = int(input("Guess the number: "))
guess += 1
if number == my_number:
print("Wow! Look at you, genius!")
break
else:
print("Nope! Not in a million years! Try again!")

else:
print("You ran out of chances")
```

“Wait! Why did you use an ‘else’ with the ‘while’ loop? I didn’t know if you can do that!”

Now you do! The ‘else’ is not just limited to ‘if’ statements; you can use it with a while as well.

Here’s what the end result looks like:

All incorrect guesses

```
Guess the number: 1
Nope! Not in a million years! Try again!
Guess the number: 2
Nope! Not in a million years! Try again!
Guess the number: 3
Nope! Not in a million years! Try again!
You ran out of chances
```

Correct guess

```
Guess the number: 17
Nope! Not in a million years! Try again!
Guess the number: 18
Nope! Not in a million years! Try again!
Guess the number: 19
Wow! Look at you, genius!
```

Remember nested conditional statements? This is exactly that. The program begins with the first understanding of certain variables. See how I have named them to make it a little easier to read.

We gave ‘guess’ a value of zero to begin with. That is exactly what you need to do as the first attempt has not yet been registered by the system. Always begin such guesses/attempts from zero and then add increments. We then followed by setting an upper limit. We could have just written the same in this way:

```
while guess <= 3:
```

The problem with this would have been that the digit ‘3’ was only recognizable by us. For any other programmer, this would not make any sense. Therefore, we replaced that with a variable so that it literally improves readability. Now, it reads like this:

“While guess is less than or equal to three:”

This is how you should always aim your codes to be. They should be readable and easy to understand by everyone.

“<=” is yet another operator. Here, the values are either less than or equal to whatever the value of the variable is on the other side.

We started by asking the user to guess, and that’s what we need as an input. However, since it will be a whole number, we also converted the same into an integer. After the user guessed the number, whether right or wrong, we immediately need the program to add a value of ‘1’ to the number of guesses. This is where we used an increment. But, unlike what we did earlier, I changed a little and used the ‘+=’ operator. It basically means to increase the value by whatever the digit you choose to write on the other side. If you are more comfortable using the previous method, it will work flawlessly as well.

Now, here’s the twist. We used an ‘if’ statement to let the program know that if the user guesses the exact number, it should print out a message that is appropriate for the occasion. Otherwise, the ‘else’ condition will take place, as long as this is not the third and final guess.

Should the final guess wrong, the count will increase for the number of guesses, and the while statement will no longer be true, in which case, the ‘else’ part of it will come into play and end the game.

The thing to notice here is the word ‘break’ that I used within the code. Go ahead and see what happens when you remove this. If you guess your numbers wrong, the code will work fine. And, if you end up inputting the right value, instead of ending the game, it will still go on until the third attempt is made.

To avoid that from happening, we use the ‘break’ statement to let the program know what to do if the condition above the break statement is met.

Now, there is almost nothing left about the ‘while’ loop, let us move to the ‘for’ loops. Slightly different from what you might expect, but interesting nonetheless.

The ‘For’ Loop

The ‘while’ loop executes whatever the code block that is written within multiple times until the condition is no longer met or invalid. The ‘for’ loop is designed to “iterate over items of collections,” and right away, that is causing some confusion.

Do not be intimidated by fancy words or technical language; once you see the loop in action, it will automatically start making sense.

To give it a little clear meaning, let us look at the example below:

```
for char in "Loops":

print(char)
```

To create a ‘for’ loop, we begin by using the keyword here. The word ‘char’ is just a variable we created. Notice how we did not define this variable before. Whenever we use ‘for’ loops, we create what are called as loop variables. These exist only within the loop itself, to carry out the loop and its operations. Here, I used ‘char’ to represent ‘characters’ since Python does not identify letters as letters.

What this means is “for every character in the word ‘Loops’”, print out the characters. Surely enough, if you execute this code, you will end up with this:

```
L
o
o
p
s
```

The system iterates over each of the components and then uses those according to what the program says. Here, we only asked it to print characters. It started with ‘L’ and then moved on to ‘o’ and continued until there were no characters left.

It isn’t necessary that you use a string, you can use what is termed as lists. These are a collection of values, either strings or numbers, stored within a list. The lists are represented by a square bracket ‘[]’ and can hold as many items as you like.

Let’s try that and see what happens:

```
for char in ["I", "Love", "Programming"]:
print(char)

Output:
I
Love
Programming
```

See how that differed? That is because when we used a single string, every character was a different object. Here, the list holds multiple objects, instead of printing them separately, it printed out whatever the value was within each component of the list.

Here's one more example, and this one has a new function for us to dive into. Suppose you wish to print out the numbers from one to 20. Instead of typing the entire numbers, we use a built-in function called *range()**:***

```
for the number in range(20):

print(number)
```

Here, we pass the higher end of the range as a parameter. Now, Python will execute this for us, and the results will be exactly how you might imagine:

```
0

1

2

3

4

5

6

7

8

9

10

11

12

13

14
```

```
15
16
17
18
19
```

See how it printed the numbers to 19 and not 20? That is because, for Python, the first position is always considered as zero. If you scroll up, you will see that the count started from zero. For now, do not get bogged down there. We will discuss that when we discuss index numbers.

If you wish to set a specific starting point, you can do so by adding value, followed by a comma, just before the 20:

```
for the number in range(10, 20):
```

Now the count will begin from 10 and end at 19. Let’s take that up a notch. Suppose I want to print out numbers from 10 to 20, and I want 20 to be printed, but I do not want all the numbers. I want the program to print out every second number, like 12, 14, 16, and so on. You can actually do that as this range function comes with what is termed as a ‘step’ for this function.

```
for number in range(10, 21, 2):
print(number)

Output:
10
12
14
```

```
16

18

20
```

Now, the program executes, starts with the first number, and knows that it needs to jump two steps and print that number. This will carry on until the final number, or the last possible number of iteration is printed. Notice, in order to print 20, I had to change the value to 21 within the range.

E-commerce and e-shops use these quite a lot to iterate over the cart items and deliver you the total price of your potential purchase. In case you wish to see how that happens, here’s one more example for a ‘for’ loop.

Scenario: I have five items in my imaginary cart. They are $5, $10, $15, $20, and $25 in prices, respectively. I want the program to let me know what my total is. While I can use the calculator myself, or pause for a few seconds and calculate the price myself, I want a quicker solution. You, as a programmer, will need to create something like this:

```
prices = [5, 10, 15, 20, 25]

total = 0

for the item in prices:

total += item

print(f"Your total price is: ${total}")


Output:

Your total price is: $75
```

Let’s be honest. This was much more fun to do than using a simple calculator, wasn’t it? Programming can be tough at times, frustrating too.

Sometimes, you might arrive at a point where you would spend the rest of your day wondering what could possibly be causing you to have such a nightmarish time with a program that seemed too simple to execute.

Relax! Every one of us faces that. It comes with the kind of work we do. Programming can be quite deceptive and will take quite a lot of time for you to master. What’s important is that you never give up. Should you feel frustrated, grab a drink, have some fresh air, and calm your mind. The solution would be more obvious than you might think.

Now that we have calmed down a little. Let’s get back to learning Python. It is time to put an end to the loops by learning one more type of loop called ‘nested’ loop. If you recall, we have already seen a nested conditional statement; an ‘if’ statement within an ‘if’ statement. Similarly, we use a ‘for’ loop within a ‘for’ loop to get things that we wish to acquire.

The ‘Nested’ Loop

Let us start this one off by trying to type in some values for ‘a,’ ‘b,’ and ‘c’ We wish to have values from zero to two for each one is somewhat a similar fashion like we type coordinates:

(a, b, c)

(0, 0, 0)

(0, 0, 1)

And so it goes on until ‘c’ is two, after which the counter is (1, 0, 0) and starts again. That would be quite a lot of work if we were to write these on our own. Fortunately, we have Python to help us out by using nested loops. How? Let’s take a look.

```
for an in range(3):
for b in range(3):
for c in range(3):
print(f"({a}, {b}, {c})")
```

Wow! Look at that! A ‘for’ loop within a ‘for’ loop within another ‘for’ loop. That are a lot of loops right there. But, that is exactly how this will work. What happens now is that the program initiates with the first position of our loop variable ‘c’ while the remaining variables hold a value of zero. Then, the loop starts again; this time, only ‘c’ jumps to a value of one while others remain the same. This will continue right until ‘c’ reaches the end of the range, after which ‘b’ will gain value of one. Hopefully, you see how this is going. The result is as under:

```
(0, 0, 0)

(0, 0, 1)

(0, 0, 2)

(0, 1, 0)

(0, 1, 1)

(0, 1, 2)

(0, 2, 0)

(0, 2, 1)

(0, 2, 2)

(1, 0, 0)

(1, 0, 1)

(1, 0, 2)

(1, 1, 0)

(1, 1, 1)

(1, 1, 2)

(1, 2, 0)

(1, 2, 1)
```

```
(1, 2, 2)
(2, 0, 0)
(2, 0, 1)
(2, 0, 2)
(2, 1, 0)
(2, 1, 1)
(2, 1, 2)
(2, 2, 0)
(2, 2, 1)
(2, 2, 2)
```

Phew! That would have taken us quite some time to write. However, some clever trickery of nested loops and just a few keystrokes later, we have it right now we want it. That is how effective nested loops are. When you are to deal with big chunks of data, you will want to rely quite a bit on nested loops. These get the job done and are mighty effective too.

Now, since that is out of the way, let us focus on operators. Those pesky little signs that keep on changing every now and then remember? We will be looking into these to see how they work for us.

Chapter 5

Operators - The Types and Their Uses

Operators are pretty much how they sound like. They operate as per our needs and connect two dots together.

That was the simplest way I can explain these. However, there are quite a few operators available when it comes to Python. They are used for various purposes and are seemingly being used in every program that will be created, apart from the ones where you are only relying on print statements.

I shall not waste a lot of time here, so let us get straight to business and see the types first and then move a little towards their uses, both including quite a bit of arithmetic as well.

Not a fan of arithmetic myself, but then again, it is necessary!

The Types

Straight away, we begin by seeing some basic ones. When we talk about arithmetic, the first few things to pop-up are the addition, subtraction, multiplication, and division signs. Python is no stranger to these, either. There are a lot of applications and programs designed using these. We will be looking into those, too, I promise.

+, -, /, *

The above signs, not including the comma marks, are universal in nature. Whether you speak English, Japanese, or Mandarin, you know you are dealing with some basic operators. These operators are in use throughout the world, at least within a calculator. Using these within Python at this point in time, hopefully, should not be a problem for you. However, these are not the only operators we use.

The ‘=’ sign, if you may recall, is not an ‘equal to’ sign in Python. It is an operator that assigns a value to a variable. To equate something, we use the ‘==’ sign.

I am sure; you had already figured that out. What about these then?

!=

>=

<=

%

//

**

+=

Saving the last one, which is technically called an augmented assignment operator, the remainder might seem a little different. Don’t worry; I am here to explain these.

!=

This is the “Not equal to” operator. You will normally find this in use within statements to compare scenarios and create conditions that are not equal to something. How does that work?

Here’s a simple version. We will look into a more complex one after this.

```
x = 28

y = 19

print(x != y)


Output:
```

```
True
```

The output value here will not be a number, but a Boolean value of ‘True.’

Here’s a more complex-natured one. Suppose you have a list of questions for the user to check for loan approval. One of these questions will have “Do you have any criminal convictions or recorded history?” to which the banks will automatically refuse. In a programming language, we will create an ‘if’ statement and use the same details as under:

```
age = 28
is_working = True
is_married = False
has_criminal_record = False
if has_criminal_record != True:
print("You are eligible for a loan!")
else:
print("Sorry, but you are ineligible for a loan")
```

Assuming that we fill these out, the application will automatically print out that the person is eligible for a loan. However, change has_criminal_record to True and then run the program. Now the program will execute the ‘else’ statement as has_criminal_record now matches ‘True’ and, therefore, renders the ‘if’ condition void.

>= and <=

The “greater and equal to” and the “lesser and equal to” operators were already used by us at least once. These are used in a similar fashion to the ‘!=’ operator. However, owing to the nature of these operators, we use these for numbers, either integers or floats. These are also called as comparison operators, and so is ‘!=’ and ‘==’ operators.

Let us put these to some use and see how they function. Mind you; this one will take a little closer look. I have designed it deliberately to intimidate you. If you read closely, you will immediately realize how easy it is to understand what’s going on here.

```
print("Welcome to our online eligibility checker!")
age = int(input("Enter your age: "))
has_license = input("Do you have a license? [Y/N]: ")
if has_license.lower() == "y":
has_license = True
else:
has_license = False
salary = int(input("Your monthly Salary: $"))
if age <= 35:
print("The age is right!")
if has_license == True:
print("You have a valid license.")
if salary >= 3500:
print("Perfect! You are eligible.")
else:
print("I am sorry, but you are below our minimum requirement")
else:
print("Sorry, but you need to have a valid license")
else:
print("You are above our maximum age limit.")
```

I have created a little eligibility checker here to see whether a person is eligible for service. Here, you will find most of the comparative operators at work. I have also given this code a bit of freedom. Now, we do not have to put in the values manually any longer. The user can put in the required values, and the program will run through instantly to deliver results accordingly. Go ahead, try out the program, and see how it works for you.

Once again, I highly encourage you to change the values, modify the program, and see how it works. The more you experiment, the quicker you will learn.

For the above, as long as all conditions are met, this will be the output:

```
Welcome to our online eligibility, checker!

Enter your age: 28

Do you have a license? [Y/N]: y

Your monthly Salary: $4000

The age is right!

You have a valid license.

Perfect! You are eligible.
```

While everything seems to be going perfectly so far, did you happen to notice something new within the code? Look again at the first ‘if’ statement. Right at the start of the condition, I used has_license.lower(), what do you think that is?

The variable I created was assigned some built-in methods, and I accessed one of those named *lower()*. They are much like functions, but instead of having massive blocks of codes within them, they only do one purpose. In this case, I wanted to ensure that whatever the user inputs, it gets converted to lowercase, to match with the condition. Since Python is a case-sensitive language, had I left the value as ‘Y,’ the condition would have never met.

To access the list of methods available, after the name of a variable, type a period ‘.’ and the list of methods would open up.

We will discuss the methods when we start talking about functions in the next chapter.

Let us move on to the remaining three so that we can then discuss something equally important; operator precedence.

The ‘%’ is called the modulus operator. This basically returns the remainder value from the equation. Let’s try this out:

```
alpha = 20

beta = 11

print(alpha % beta)


Output:

9
```

This operator first divides the two numbers and then brings forth the remainder as there is no further division possible. You can try and change the value of alpha to 200, and the remainder will be two.

The ‘/’ is a straightforward division operator. Replace the modulus sign above (alpha at 20) with the division sign, and you get the following result:

```
1.8181818181818181
```

Now, the point to note here is that we entered two integers, and the return value was afloat. What if we just wish to settle for an integer instead? That’s where we use the ‘//’ sign. Now, use this sign instead of the division sign. The result should be ‘1’ only. The result will not be rounded off to the nearest tenth. This is the floor division operator, which always returns an integer value.

If, however, you wish to round off the value, you will need to use a function called *round()*, and this is how you will do so:

```
alpha = 20

beta = 11

print(round(alpha / beta))


Output:

2
```

Finally, we have the '**' signs. These are exponential operators. Now, replace the division operator with the exponential operator, remove the round function, and you will get the following result:

```
204800000000000
```

Ah! If only we had that many digits in our accounts!

Well, now that we have seen the operators and their types, let us discuss a little about the operator precedence.

The Operator Precedence

What do you think would be the outcome of this simple calculation?

```
print(10 + 20 - 5 * 4 / 2 ** 2)
```

400? 20? No! The answer would be 25. Why? Because there is specific precedence that operators follow. There are some who take higher priority and are calculated first. Here is a simple way to explain things:

Exponentiate > Divide > Multiply > Add/Subtract

That is always the case in Python. And with that, we can say goodbye to some basic operators and go to a little more advanced once.

The Logical Operators

For the purpose of explaining this a little more clearly, let us create a scenario. You are asked to create a program that checks for someone's eligibility for a mortgage based on certain values and inputs. You are told that there are questions like asking the name, age, and then there are two major factors that will influence the decision. These pertain to the applicant having a good or bad credit history, and having a high income of at least $5,000 per month.

To create this, we can certainly do what I did earlier on, by using nested conditional statements, but at times, they may not be the best option to go for. Instead, we use logical operators to do the hard work for us. How? Let's find out, shall we?

```
print("Eligibility Checker 101")

name = input("Please enter your name: ")

age = int(input("Please enter your age: "))

salary = int(input("What is your monthly salary? "))

min_salary = 5000

has_good_credit = True

if salary >= min_salary and has_good_credit:

print(f"Congratulations {name}, You are eligible for a mortgage.")

else:

print(f"{name}, it appears you may not be eligible at this time")
```

The first logical operator we have here is the 'and' operator. What that does is it creates a condition where both the former and the latter must be met. If not, the entire block of code will not be considered, and the 'else' statement will be executed instead.

Note: See how I did not assign any comparison operator for the has_good_credit? That’s because the value for that is a bool value, which is currently set to true!

If you read this code in plain English, it literally reads:

“If the salary is equal to or higher than the minimum salary, and has good credit.”

Now, let us type in the values to see if this works:

```
Eligibility Checker 101

Please enter your name: Smith

Please enter your age: 34

What is your monthly salary? 5000

Congratulations Smith, You are eligible for a mortgage.
```

Now, let’s try to change the bool value to ‘False’ and see what happens:

```
Eligibility Checker 101

Please enter your name: Snow

Please enter your age: 30

What is your monthly salary? 6000

Snow, it appears you may not be eligible at this time
```

This happened because the first condition is met, but the second condition was not satisfied.

So, you submitted the program, and a few days later, the client returns and says, “Well, I would like you to change the program a little. This time, we want our applicants to either have a good salary or good credit. Anyone would do!”

Now, let us look at the code again. How can we do that without changing much? That is where the second logical operator comes in.

The ‘or’ operator is used in such situations where one or the other condition is true, in which case the program will execute the ‘if’ statement block. Let us try it out by changing the ‘and’ to ‘or’ and keeping the has_good_credit value to false.

```
print("Eligibility Checker 101")

name = input("Please enter your name: ")

age = int(input("Please enter your age: "))

salary = int(input("What is your monthly salary? "))

min_salary = 5000

has_good_credit = False

if salary >= min_salary or has_good_credit:

print(f"Congratulations {name}, You are eligible for a mortgage.")

else:

print(f"{name}, it appears you may not be eligible at this time")
```

Output:

```
Eligibility Checker 101

Please enter your name: Nathan

Please enter your age: 28

What is your monthly salary? 6000

Congratulations Nathan, You are eligible for a mortgage.
```

Now, the code successfully executed and gave Nathan the good news. This is because the ‘or’ operator informed Python, “Hey! Even if one of these conditions is applicable, go ahead!”

Lo and behold! The world of logical operators. They make our lives so much easier, don’t they?

Lastly, we also have one more logical operator called ‘not’ operator. This is slightly tricky to understand, but bear with me on this one.

Suppose the same client returns and asks, “Hey! Good job, but I need another chance. This time, I want you to code your program so that it only works if the applicant has a good salary and not good credit.”

Sure, and to do just that, this is what we will do:

```
print("Eligibility Checker 101")

name = input("Please enter your name: ")

age = int(input("Please enter your age: "))

salary = int(input("What is your monthly salary? "))

min_salary = 5000

has_good_credit = False

if salary >= min_salary and not has_good_credit:

print(f"Congratulations {name}, You are eligible for a mortgage.")

else:

print(f"{name}, it appears you may not be eligible at this time")
```

What the ‘not’ operator does is it changes the value of the second variable from True to False, or from False to True. In this case, since the applicant has no credit history, the condition fits. The interpreter will see this as

“Applicant has a salary that is greater than the minimum and does not have a good credit history, which is true, then let’s go with it!”

The result will be as follows:

```
Eligibility Checker 101

Please enter your name: Nicole

Please enter your age: 29

What is your monthly salary? 8000

Congratulations, Nicole, You are eligible for a mortgage.
```

If you were to change the bool value to true, Nicole here would be left a little sad.

And that about wraps up our trip to the world of operators. We saw the types, and we saw their usages as well. These do take a little while to be understood fully, but rest assured, they are super helpful. Continue practicing these codes and use your own imagination to come up with situations and scenarios where you may be able to use these effectively.

Chapter 6

Lists, Tuples, and Dictionaries

This chapter will not exactly be that long at all. Most of these will make sense since you have come this far. You already know how to store values in variables, but when you have more than one value to store, you will need something else to rely on.

This is where Lists, Tuples, and Dictionaries come in. While they may seem similar in nature, they are quite different. We will look into these closely now, starting with lists, to understand how each of these work and, hopefully, know when to use these.

Lists

They are exactly as they sound and function pretty much the same. A list, in Python, is represented by square brackets '[]' and it can hold multiple items within it. You can store as many items or values as you like within a list and recall each component easily.

Let us look at a simple list first to see how exactly it works. For that, we have six imaginary volunteers; Joey, Chandler, Ross, Phoebe, Rachel, and Monica. Let's also assume that we have no idea of the obvious connection to these names. Time to create our first list:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
```

And we have our list created. Since we are using string values, we will need to use quotation marks to let Python know that these are string values.

Suppose you do not know what's on the list. You do not even know how long the list is. Our target is to find out:

- The number of components within this list
- Value of individual components

To do that, we will first need to see how long the list is, and we can do just that by using the *len()* function. The *len()* function basically displays the length of characters, components or items within a variable or a list.

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]

print(len(friends))


Output:

6
```

Now, we have obtained one piece of information. Moving to the next one, let us find out what is at the start of this list. To do that, we will call up the first element, and this is where the concept of index position comes in.

An index is the position of a component. Here, the first component is ‘Joey’ and to find out that, we will do this:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]

print(friends[0])
```

Here, we will use the square brackets and use the value of zero. Why zero and not one? In Python, and in quite a few languages as well, the first position is always a zero. Here, “friends[0]” essentially tells the program to print the component with the first index position. The output, obviously, is:

```
Joey
```

Similarly, let’s print the rest out accordingly!

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
print(friends[0])
print(friends[1])
print(friends[2])
print(friends[3])
print(friends[4])
print(friends[5])

Output:
Joey
Chandler
Ross
Phoebe
Rachel
Monica
```

There is another way to do this. Suppose you do not know the length of the list, and you wish to print out the last recorded entry of the same, you can do that by using the following method:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
print(friends[-1])

Output:
Monica
```

The ‘-1’ will always fetch you the last entry. If you use ‘-2’ instead, it will print out the second to last entry as shown here:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
print(friends[-2])

Output:
Rachel
```

There are other variations involved here, as well. You can call the items from a specific starting point. Using the same list above, let’s assume we wish the prompt to print out the last three entries only. We can do that easily by using the starting index number of the value we wish to print. In this case, it would be the index number ‘3’:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
print(friends[3:])
Output:
['Phoebe', 'Rachel', 'Monica']
```

You can also limit what you wish to see on the screen further by setting a range of index numbers. The first number, the one before the colon, represents the starting point. The number that you input after the colon is the endpoint. In our list of friends, we have a range from zero to five, let us narrow our results down a little:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]
print(friends[2:5])

Output:
['Ross', 'Phoebe', 'Rachel']
```

Remember, the last index number will not be printed; otherwise, the result would have also shown the last entry.

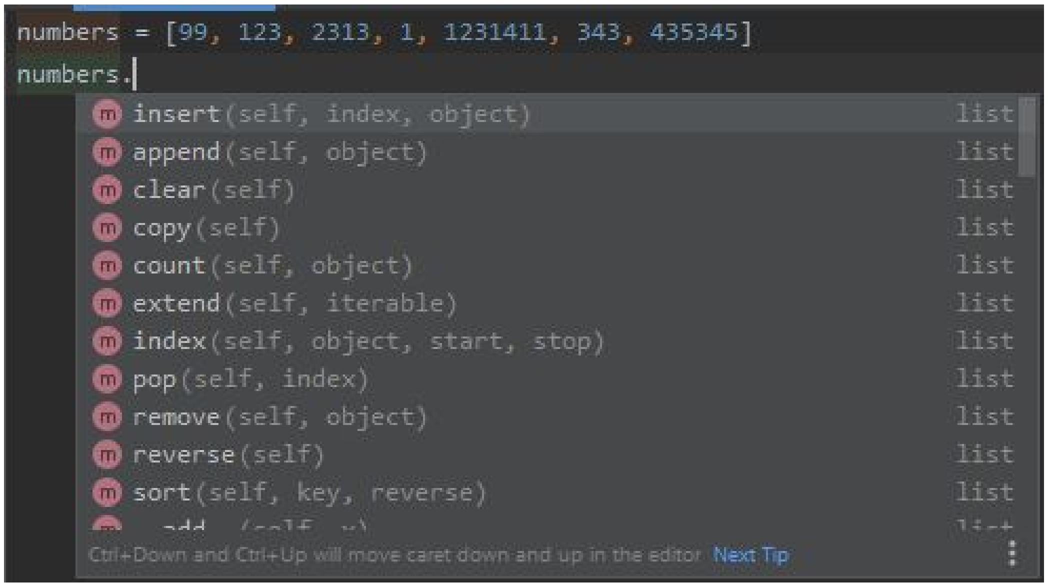
You can modify the values of a list quite easily. Suppose you wish to change the entry at index number five of the above list, and you wish to change the entry from ‘Monica’ to ‘Geller,’ this is how you would do so:

```
friends = ["Joey", "Chandler", "Ross", "Phoebe", "Rachel", "Monica"]

friends[5] = "Geller"

print(friends)


Output:

['Joey', 'Chandler', 'Ross', 'Phoebe', 'Rachel', 'Geller']
```

It is that easy! You can use lists with loops and conditional statements to iterate over random elements and use the ones which are most suitable to the situation. Practice a little, and you should soon get the hang of them.

What about if you wish to add numbers or values to the existing lists? Do we have to scroll all the way up and continue adding numbers manually? No! There are things called methods, which you can access at any given time to carry out various operations.

Here’s a screengrab to show just how many options you have available to you once you press the ‘.’ Key:



```
numbers = [99, 123, 2313, 1, 1231411, 343, 435345]
numbers.
```

m insert(self, index, object) list
m append(self, object) list
m clear(self) list
m copy(self) list
m count(self, object) list
m extend(self, iterable) list
m index(self, object, start, stop) list
m pop(self, index) list
m remove(self, object) list
m reverse(self) list
m sort(self, key, reverse) list
Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip

We will not be talking about all of these, but we will briefly look at some basic methods that every programmer should know.

Straight away, the ‘append’ method is what we use to add values. Simply type in the name of the list you wish to recall, followed by “.append” to let the program know you wish to add value. Type in the value, and that is it!

The problem with using the append method is that it adds the item randomly. What if you wish to add value to a specific index number? To do that, you will need to use the insert method.

Using an insert method, you will need to do this:

```
numbers = [99, 123, 2313, 1, 1231411, 343, 435345]
numbers.insert(2, 999)
print(numbers)

Output:
[99, 123, 999, 2313, 1, 1231411, 343, 435345]
```

The number was added right where I wanted. Remember to use an index position that is valid. If you are unsure, use the *len()* function to recall how many components are within a list. That should then allow you to know the index positions available.

You can also remove items from a list as well. Simply use the remove() method and input the number/value you wish to remove. Please note that if your list has more than one value that is exactly the same, this command will only remove the first instance only.

Let us assume you are presented with a list of mixed entries. There is no order that they follow. The numbers are just everywhere, disregarding the order. If you wish, you can sort the entire list to look more appealing by using the *sort()* method.

```
numbers = [99, 123, 2313, 1, 1231411, 99, 435345]

numbers.sort()

print(numbers)


Output:

[1, 99, 99, 123, 2313, 435345, 1231411]
```

You know, you can also have it the other way around by using the reverse() method. Try it!

To completely empty a list, you can use the clear() method. This specific method will not require you to pass any argument as a parameter. There are other methods such as pop() (which takes away the last item on the list only) that you should experiment with. Do not worry; it will not crash your system down or expose it to threats. The IDE is like a safe zone for programmers to test out various methods, programs, and scripts. Feel free and feel at ease when charting new waters.

Tuples

As funny as the name may be, tuples are pretty much like lists. The only major difference is that these are used when you do not wish for certain specialized values to change throughout the program. Once you create a tuple, it cannot be modified or changed later on.

Tuples are represented by parenthesis (). If you try and access the methods, you will no longer have access to the methods that you did when you were using lists. These are secure and used only in situations where you are certain you do not wish to change, modify, add, or remove items. Normally, we will be using lists, but it is great to know we have a safe way to do things as well.

Dictionaries

Unlike tuples and lists, dictionaries are different. To begin with, they work with “key-value pairs,” which sounds confusing, I know. However, let us look at what exactly a dictionary is and how we can call, create, and modify the same.

To help us with the explanation, we have our imaginary friend here named James, who has graciously accepted to volunteer for the exercise. We then took some information from him such as his name, email, age, the car he drives, and we ended up with this information:

```
Name – James

Age – 58

Email – james@domain.com

Car – Tesla T1
```

What we have here are called key pairs. To represent the same within a dictionary, all we need is to create one. How do we do that? Let’s have a look.

```
friend = {
"name": "James",
"age": 30,
"email": "james@domain.com",
"car": "Tesla T1"
}
```

We define a dictionary using {} braces. Add each pair as shown above with a colon in the middle. Use a comma to separate items from one another. Now, you have a dictionary called ‘friend’ and you can access the information easily.

Now, to call up the email, we will use square brackets as shown here:

```
friend = {
"name": "James",
"age": 30,
"email": "james@domain.com",
"car": "Tesla T1"
}
print(friend["email"])

Output:
james@domain.com
```

Similarly, try recalling the other elements to try it out yourself. Once again, I remind you that Python is case sensitive. If you recall ‘age’ as ‘Age’, it will not work at all.

Suppose you wish to recall an item without knowing the key pairs within a dictionary. If you type in a key named ‘dob’, the program is going to return an error like this:

```
Traceback (most recent call last):

  File "C:/Users/Programmer/PycharmProjects/PFB/Lists2.py", line 7, in <module>

print(friend["dob"])

KeyError: 'dob'
```

There is a way you can check for values without the program screaming back at you with red/pink fonts. Use the *.get()* method instead, and the program will simply say ‘None,’ which represents the absence of value.

You can also give any keypair, that may not have existed before, a default value as well.

```
friend = {

"name": "James",

"age": 30,

"email": "james@domain.com",

"car": "Tesla T1"

}

print(friend.get("dob", "1, 1, 1900"))


Output:
```

```
1, 1, 1900
```

Unlike tuples, you can add, modify, or change values within a dictionary. I have already shown you how to do that with lists, but just for demonstration purposes, here’s one way you can do that.

```
friend["age"] = 60
print(friend["age"])

Output:
60
```

Now, that wasn’t so bad, was it? This then ends our trip to the world of lists, tuples, and dictionaries. You must pay close attention to these as you will need to use a few of them, if not all, at once, more often than you might imagine. The more you practice and familiarize yourself with lists, tuples, and dictionaries, the easier it will be to create some incredible programs and code inefficient codes at the same time.

Chapter 7

Functions

We began with almost no prior knowledge about Python except for a clue that it was some kind of programming language that is in great demand these days. Now, look at you, creating simple programs, executing codes, and fixing small-scale problems on your own. Not bad at all! However, learning always comes to a point where things can get rather trickier.

In quite a similar fashion, Functions are docile looking things; you call them when you need to get something done. But did you know that these functions have so much going on at the back? Imagine every function as a mini-program. It is also written by programmers like us to carry out specific things without having us write lines and lines of codes. You only do it once, save it as a function and then just call the function where it is applicable or needed.

The time has come for us to dive into a complex world of functions where we don't just learn how to use them effectively, but we also look into what goes on behind these functions, and how we can come up with our very own personalized function. This will be slightly challenging, but I promise there are more references that you will enjoy to keep the momentum going.

Understanding Functions Better

Functions are like containers that store lines and lines of codes within themselves, just like a variable that contains one specific value. There are two types of functions we get to deal with within Python. The first ones are built-in or predefined; the others are custom-made or user-created functions.

Either way, each function has a specific task that it can carry out. The code that is written before creating any function is what gives that function identity and a task. Now, the function knows what it needs to do whenever it is called in.

When we began our journey, we wrote, “I made it!” on the console as our first program? We used our first function there as well: the print() function. Functions are generally identified by parentheses that follow the name of the function. Within these parentheses, we pass arguments called parameters. Some functions accept a certain kind of parenthesis, while others accept different ones.

Let us look a little deeper and see how functions greatly help us reduce our work and better organize our codes. Imagine, we have a program that runs during live streaming of an event. The purpose of the program is to provide our users with a customized greeting. Imagine just how many times you would need to write the same code again and again if there were quite a few users who decide to join your stream. With functions, you can cut down on your own work easily.

For us to create a function, we first need to ‘define’ the same. That is where a keyword called ‘def’ comes along. When you start typing ‘def’, Python immediately knows you are about to define a function. You will see the color of the three letters change to orange (if using PyCharm as your IDE). That is another sign of confirmation that Python knows what you are about to do.

```
def say_hi():
```

Here, say_hi is the name I have decided to go with; you can choose any that you prefer. Remember, keep your name descriptive so that it is understandable and easy to read for anyone. After you have named your function, follow it up with parentheses. Lastly, add the friendly old colon to let Python know we are about to add a block of code. Press enter to start a new indented line.

Now, we shall print out two statements for every user who will join the stream.

```
print("Hello there!")
print('Welcome to My Live Stream!')
```

After this, give two lines of space to take away those wiggly lines that appear the minute you start typing something else. Now, to have this printed out easily, just call the function by typing its name and run the program. In our case, it would be:

```
say_hi()
Output:
Hello there!
Welcome to My Live Stream!
```

See how easily this can work for us in the future? We do not have to repeat this over and over again. Let's make this function a little more interesting by giving it a parameter. Right at the top line, where it says "def say_hi()"? Let us add a parameter here. Type in the word 'name' as a parameter within the parenthesis. Now, the word should be greyed out to confirm that Python has understood the same as a parameter.

Now, you can use this to your advantage and further personalize the greetings to something like this:

```
def say_hi(name):
    print(f"Hello there, {user}!")
    print('Welcome to My Live Stream!')

user = input("Please enter your name to begin: ")
say_hi(user)
```

The output would now ask the user regarding their name. This will then be stored into a variable called user. Since this is a string value, say_hi() should be able to accept this easily. Bypassing ‘user’ as an argument, we get this as an output:

```
Please enter your name to begin: Johnny
Hello there, Johnny!
Welcome to My Live Stream!
```

Now that’s more like it! Personalized to perfection. We can add as many lines as we want, the function will continue to update itself and provide greetings to various users with different names.

There may be times where you may need more than just the user’s first name. You might want to inquire about the last name of the user as well. To add to that, add this to the first line and follow the same accordingly:

```
def say_hi(first_name, last_name):
    print(f"Hello there, {first_name} {last_name}!")
    print('Welcome to My Live Stream!')


first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
say_hi(first_name, last_name)
```

Now, the program will begin by asking the user for their first name, followed by the last name. Once that is sorted, the program will provide a personalized greeting with both the first and last names.

However, these are positional arguments, meaning that each value you input is in order. If you were to change the positions of the names for John Doe, Doe would become the first name, and John will become the last name. You may wish to remain a little careful about that.

Hopefully, now you have a good idea of what functions are and how you can access and create them. Now, we will jump towards a more complex front of ‘return’ statements.

“Wait! There’s more?” Well, I could have explained this earlier, but back then, when we were discussing statements, you may not have understood it completely. Since we have covered all the bases, it is appropriate enough for us to see exactly what these are and how these gel along with functions.

Return Statement

Return statements are useful when you wish to create functions whose sole job is to return some values. These could be for users or programmers alike. It is a lot easier if we do this instead of talk about theories, so let’s jump back to our PyCharm and create another function.

Let us start by defining a function called ‘cube’ which will multiply the number by itself three times. However, since we want Python to return a value, we will use the following code:

```
def cube(number):

return number * number * number
```

By typing ‘return’ you are informing Python that you wish for it to return a value to you that can later be stored in a variable or used elsewhere. It is pretty much like the input() function where a user enters something, and it gets returned to us.

```
def cube(number):

return number * number * number
```

```
number = int(input("Enter the number: "))
print(cube(number))
```

Go ahead and try out the code to see how it works. We don't need to define functions such as these. You can create your complex functions that convert kilos into pounds, miles into kilometers, or even carry out far greater and more complex jobs. The only limit is your imagination. The more you practice, the more you explore.

With that said, it is time to say goodbye to the world of functions and head into the advanced territories of Python. By now, you already have all you need to know to start writing your codes. What you learn from the next chapter will help you achieve greater understanding and access to some amazing things which are not possible otherwise.

Chapter 8

Classes and Exception Handling

Straight away, by classes, I do not mean the regular classes that you would expect at schools, colleges, and universities, nor do I mean that there are qualities of Python in any way; they are completely different things.

Classes are not exclusive to Python, but they are as important as anything else for any programmers across the globe. These are found in almost all known computer programming languages.

In the simplest definition, classes are what we use to define new types of data that we use. I did say there were three in the start, strings, numbers, and Booleans. Then we came across a little more complex things called lists, tuples, and dictionaries. But what if you are still unable to get the desired outcome from the program you have been working on for such a long time? What if you feel like there must be something else apart from these types which can help you achieve greater results? Fortunately, classes are your answer.

A class can hold various functions and methods within itself. It does not need parentheses like functions and methods, nor do we create these by using the ‘def’ keyword. These are created using the word ‘class’, and they can be super helpful, especially for programmers with a keen interest in object-oriented programming.

Creating Our First Class

Before you even begin to create a class, or function, or any other kind of component, always visualize what you want to get out of it. It makes things a lot easier for you as a programmer.

At this point, you may be blank and might be struggling to come up with a class to create. Let me help you out with one. Let us create a class to which we want specific functions and methods attached to. We want this class to do things other data types were unable to do so.

I will not be creating anything that may fall outside the scope of this book, so what I have here is easily understandable. However, there are a few things that might take you by surprise, but those are deliberate so that you have every chance of understanding what they are.

```
class Instructor:

def __init__(self, name):

self.name = name


def talk(self):

print("talk")


me = Instructor("Keanu Reeves")

print(me.name)

me.talk()
```

The first thing to notice here is the naming convention I have used to name the class. Always use an uppercase for the first letter for every word that you may type when naming a class. You do not need to use underscores to separate words either. If you were to name this class as your first class, it would look like this:

```
class MyFirstClass
```

Next, we have the familiar ‘def’ keyword. But what about the double underscores and init? You may have already noticed these when calling methods. These are called constructors. For now, all you need to know is that we call upon these to initialize something.

Then, we have a parameter that says ‘self’, and I did not put it there. It is something that will come up automatically. It is referencing itself. We have only added another parameter called ‘name’ to allow us to use strings as names to display. Next, we gave the object an attribute called ‘name’, as seen above. Attributes are required to provide your functions a greater detail.

The next function is rather simple. We just created a function called to talk and asked Python to print out the same on the prompt.

Moving forward, or downwards by two spaces, we created a variable called ‘me’ with an assigned value of the class we just created. Notice how I have used class as a function (with parentheses). You might be wondering that I just said moments ago that classes do not need parentheses, and yet, I am using them here. When you are defining classes, you do not need these; however, when you are using them, you will need to rely on them to pass additional information.

Now, with the print command ready, I used my newly created class to call upon an attribute of ‘.name’, which I created within this class. This then allows the prompt to print out the name, followed by the last function, which was again another print statement as defined above.

Classes are generally created so that other objects can be created using these classes. Imagine two characters, Tony and Steve. We want each of these to be objects carrying different attributes like name, age, and suit color. To do that, we will first need to create a class. Let us go ahead and do so.

```
class Heroes:

def __init__(self, name, age, color):

self.name = name

self.age = age
self.color = color

def introduce_self(self):
print("My name is " + self.name)
print(f"I am about {self.age} years old")
print("My costume color is " + self.color)


hero1 = Heroes("Steve", 40, "Blue")
hero2 = Heroes("Tony", 38, "Red")

hero1.introduce_self()
hero2.introduce_self()

Output:
My name is Steve
I am about 40 years old
My costume color is Blue
My name is Tony
I am about 38 years old
My costume color is Red
```

We began by naming our classes appropriately. We then created a constructor to help us create various attributes that we can call upon later. Before doing so, ensure that you pass those as parameters after ‘self’ so that they are recognizable by the program interpreter. After defining attributes and assigning them values, we created a function called ‘introduce_self’ where we had three statements printed. Notice how the second one is a formatted string. That is because the age is an integer, and it will not work if you try to merge a string and an integer on their own.

Once sorted and happy, we moved on to create objects called ‘hero1’ and ‘hero2’ from the same class. Now, instead of typing this information separately, we just passed the information as arguments in the ‘Heroes()’ class. Next, we just ran a function we created earlier on, and the rest was just plain history.

I know this might be a little complicated at first. Classes are a subject that is normally apt for advanced students of Python. Still, it is essential that you, as a programmer, do not just stop thinking you know everything and miss out on some of the more advanced learning opportunities. Mastering classes will take quite a bit of time. Just a chapter or even a book might not suffice to give you the perfect command of classes and some other aspects of programming. I intended to introduce you to this vast world of advanced topics. How you practice, and research on this is your call.

Exception Handling

What exactly are exceptions? Why haven’t we come across this so far? Well, we have, or at least you may have quite a lot of time but never noticed before. Create any program to make it crash deliberately.

Here’s one that I created:

```
name = "Bruce Wayne"

age = 45

print(name + age)


Output:
```

```
Traceback (most recent call last):
  File "C:/Users/Programmer/PycharmProjects/PFB/exception.py", line 3, in <module>
print(name + age)
TypeError: can only concatenate str (not "int") to str


Process finished with exit code 1
```

See the error? Not the one in the middle, I am referring to the very last line. It says the program ended with an exit code followed by the number one. That ‘one’ is informing us that something went wrong, and the program ended in a crash or abruptly ended. If the code were zero, it would have meant that our code went through and got executed beautifully!

We, as Python programmers, are bound to know when such errors are about to come. It’s called anticipating, and it is something you should have already done when you say the first three lines of my latest code.

The problem is, we programmers would know what this code means. For any ordinary user, they would have no clue what this means and would end up searching YouTube libraries, just to find a video that explains what the error code 1 means. There is a way we can address this situation, and that is called exception handling.

Before we begin, just remember the bold text that says ‘TypeError’ on our error that occurred just a moment ago. We will need to recall that a little later.

Exception handling is where we tell a program to *try* a block of code and see if the same works fine. If not, anticipate the type of error you will get. *Except* for showing a console that is lit up with gibberish, we then tell Python to print out a user-friendly text that means something. Don’t worry about the words ‘try’ and ‘except’ because that is exactly what I will be showing you now.

```
try:
name = "Bruce Wayne:"
age = 45
print(name + age)
except TypeError:
print("Please use a formatted string or convert age to a string")
```

We have just asked Python first to try a situation out. If the code is executed without returning any errors, it’s fine. If not, it will type in a friendly message instead of letting the users know what to do to avoid this. This will prevent the application from crashing and keep users informed of the errors they may have made. Now, let’s try and run this through to see what we get.

```
Please use a formatted string or convert age to a string
Process finished with exit code 0
```

Since we knew we were encountering a ‘TypeError’, we have just rectified the situation and now look; the program ended with an exit code of zero. Yay!

Here’s something for you to know! The exception we created here only deals with a specific type of error. If this error were ValueError, the code would not execute, and the program will still crash.

Just like ‘elif’, which we used back in the ‘if’ statement chapter, you can add as many ‘except’ codes as you like to catch such issues and address them accordingly.

Let us look at how that works as well, shall we?

```
try:
```

```
age = 45
age1 = 0
average = age / age1
print(average)
except TypeError:
print("Please use a formatted string or convert age to a string")
```

What do you think will happen here? Would the program go through? Would this program be able to catch the exception that might be caused, in case the program decides to crash? Let’s find out.

```
Traceback (most recent call last):
File "C:/Users/Programmer/PycharmProjects/PFB/exception.py", line 4, in <module>
average = age / age1
ZeroDivisionError: division by zero
Process finished with exit code 1
```

That was expected. Since the exception caused here is different from the one we have created earlier on, it just went through and crashed. Time to put our thinking caps on and come up with an exception to handle this situation.

```
try:
age = 45
age1 = 0
average = age / age1
print(average)
```

```
except ZeroDivisionError:
print("For God's sake! Who divides a number with 0?")
except TypeError:
print("Please use a formatted string or convert age to a string")
```

What do you think now? Will this work, or will we still end up with an exit code of 1?

```
For God's sake! Who divides a number with 0?

Process finished with exit code 0
```

And that's how you do this! Now, everyone knows what went wrong and how they can correct the error they may have unintentionally caused.

It is something of a trait to have to be able to anticipate errors coming your way beforehand. It comes with practice, but it is certainly a trait to have for any programmer from any corner of the world.

The better you are at handling exceptions, the easier your users will find the program/application to use. They would know what needs to be done, how to sort matters out, and how to continue having a great experience while using your written programs.

This is almost everything that you would need to know about exception handling. Now, you will no longer be bound to the silly errors that keep on coming. Well, at least your users will not have to face those anyway. Create messages for such exceptions which hold a precise meaning and deliver information to the end-users so that they do not have to rely on someone else to figure out what just went wrong here. For programmers, you can also add some notes within your code. These are not seen as a part of code when they are being executed. These are called **comments** and are represented with a # sign. To give you an example, here's one:

```
try:
age = 45
age1 = 0  #change this age to something else
average = age / age1  #to find out the average age
print(average)
except ZeroDivisionError: 
print("For God's sake! Who divides a number with 0?")
except TypeError:
print("Please use a formatted string or convert age to a string")
```

If you execute the program, it will still function normally. These are notes which you will find in quite a few programs and may have seen some already. It’s a good idea to use these if you wish to share your program and code with other programmers or users.

With that said, it is time to wrap this up and move on to our final frontier. I shall wait for you in Chapter 9.

Chapter 9:

Inheritance, Modules, and Packages

The journey so far has been full of ups and downs. There were things you may have anticipated, and then there were things which you didn't anticipate at all. You have learned everything that a beginner should know of, and you did quite a brilliant job by carrying out the exercises on your own, modifying certain aspects to experiment with PyCharm and Python, bravo!

#I am assuming you did that, right?

We have gone past the point where most beginners end and have stepped up into advanced levels of Python. Already, we encountered the confusing world of classes and exception handling. After having somewhat a vague concept of that, it is time for us to move to our final frontier and see what Inheritance, Modules, and Packages are all about, why we need them, and how we can use them in Python to make our programming more effective and easier.

Inheritance – Almost as it sounds!

Yes, that is true. They are indeed almost the same in meaning. Inheritance is nothing more than a mechanism to reuse code over and over again.

"Umm… Wasn't that functions?"

You may wish to hold on to that thought for now. Once we are formally introduced with inheritance, the answer will be obvious enough to see how this differs from functions or methods or loops.

Inheritance is a way to avoid repeating yourself, typing the same things again and again. How? Let us assume we wish to create a class called Cars. We give it some function named speed, whose primary objective is to print the word "fast" at the console.

In a real-life situation, you will not be dealing with one or two lines of code to define a function. You will be handling multiple lines just for one function alone.

Now, we wish to create another class called Bikes. We also wish to give it the same function of speed and print the same message saying the same thing. Now, if you re-write the code, you have failed as a programmer. As a programmer, it is our job to make things easier for us. Instead of repeating, we simply inherit this function from the other class. Let us see how this works.

```
class Cars:
    def speed(self):
        print("Fast")

class Bikes:
    def speed(self):
        print("Fast")
```

Sure enough, that does look simple, but imagine how many lines would you have to re-write when in a real-life situation? This poses more issues than you might think up of. Imagine you had to copy and paste the same line of code for more than 20 classes you defined further into your program. In the end, you find out you had some values wrong at the start. This means the rest will now need to be changed individually as well. Why go through such hard work? Instead, let's see how inheritance can help us out here.

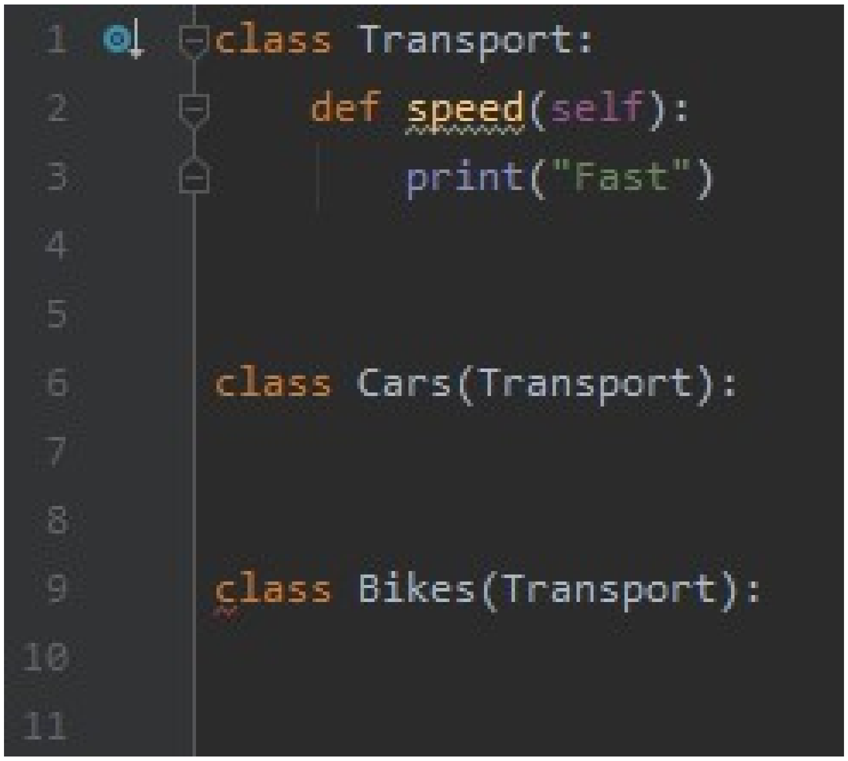
We begin by first creating a parent class. Since cars and bikes are modes for transportation, we will create a class called 'Transport' to begin the process. Once done, we simply move the entire block of code we were interested in, right inside this new class. It should now look like this:

```
class Transport:
    def speed(self):
        print("Fast")
```

Easy, right? It gets a lot easier now. To inherit these qualities, all you need to do is this:

```
class Transport:
    def speed(self):
        print("Fast")
class Cars(Transport):


class Bikes(Transport):
```

By passing the class name ‘Transport’ into parentheses of the new classes, we are telling Python, “Hey! They will inherit qualities from their parent class.” But here’s a problem. To show the problem, it’s best to look into this screenshot.



```
1  class Transport:
2      def speed(self):
3          print("Fast")
4
5
6  class Cars(Transport):
7
8
9  class Bikes(Transport):
10
11
```

See how at line ‘9’ the letter ‘c’ has a red line underneath? That’s Python’s way of saying that it is not happy with us. To avoid that, all we need is to pass a word called ‘pass’ within the sub-classes. This way, Python knows it does not need to worry about anything within the classes.

```
class Transport:
    def speed(self):
        print("Fast")

class Cars(Transport):
    pass

class Bikes(Transport):
    pass
```

Now, you can easily create objects as we did earlier on with our two heroes ‘Steve’ and ‘Tony.’ You can also call upon their specific attributes as well. So far, we have only set one attribute, which is speed.

```
class Transport:
    def speed(self):
        print("Fast")
  class Cars(Transport):
   pass

class Bikes(Transport):
    pass

ferrari = Cars()
ferrari.speed()
yamaha = Bikes()
yamaha.speed()
```

The result will print out the word ‘fast’ for both. However, since bikes and cars are two separate things, you can add exclusive attributes to each. Maybe the number of wheels, the engine position, the passenger capacity, and so on. But, to do that, you will need to remove the word ‘pass’ as we no longer want Python to skip through the block of code within a specific class. I designed some of the above. Sorry that this is long, but I have tried my best to keep it as readable as possible.

```
class Transport:
   def speed(self):
        print("It is really fast")
    def tyres(self, number):
        print(f"It has {number} tyres.")
    def engine(self, name):
        print(f"It has a massive {name} engine.")

class Cars(Transport):
    def make(self, name):
        print(f"It's a {name}")

class Bikes(Transport):
    def wheelie(self):
        print("It can wheelie like crazy")

ferrari = Cars()
ferrari.make("Ferrari")
ferrari.engine("V12")
ferrari.tyres(4)
ferrari.speed()
yamaha = Bikes()
yamaha.engine("Twin-V")
yamaha.speed()
yamaha.tyres(2)
yamaha.wheelie()
```

The results are rather easy to understand. But, for the sake of knowledge and information, here are the results:

```
It's a Ferrari

It has a massive V12 engine.

It has 4 tires.

It is really fast

It has a massive Twin-V engine.

It is really fast

It has 2 tires.

It can wheelie like crazy
```

This is how inheritance makes our lives so much easier. Instead of typing all those codes, we were able to make this easy. This is the magic of inheritance. Remember this, and you will no longer be writing hundreds of lines of code over and over again.

With that said, let us now move on to our last two components: modules and packages.

Modules and Packages

Do not be alarmed by these two words. They are far easier to understand than they sound like. We shall first begin by having a look into what modules are and then proceed towards packages.

The module is essentially a file in which there is some Python code that has been written by either you or someone else. We normally use modules for organizing our codes in an organized fashion. Think of a supermarket with multiple aisles where each one is labeled clearly to let us know what we can expect within these aisles.

Similarly, modules are files that have codes within them performing a specific task. These can then be saved and later imported to other Python programs for use. Yes, you read that right. Create a good function now and use it for eternities to come.

So how does that work? Suppose you have a simple distance converter. This converter converts kilometers to miles and vice versa. Let us first begin by defining these two functions:

```
def kms_to_miles(distance):

return distance * 0.621


def miles_to_kms(distance):

return distance * 1.609
```

Now we have our functions; it is time to save them as a separate file. To do that, go to the project browser window (the one on the left of the typing area), right-click on the project name, and choose the new>file. Name the file as distance_converter, or any other name that you fancy and add a .py extension at the end. Once done, a new blank page will open. Simply cut the entire code from the previous file into this one, and that's it.

Now, our main file is empty. It is time to 'import' these converters within our file. To do that, all you need to do is to type in the following:

```
import distance_converter
```

As soon as you press enter, this will be greyed out. That is perfectly fine. Now, the converters are imported and, hence, usable for us within this program. If you are not happy with the name, which may be a useful thing, later on, you can always rename these by doing the following:

```
import distance_converter as a converter
```

This will then import the converter and then give it the name you just assigned. To test, simply type the name you chose and press the ‘.’ Key to access the functions available. You should now be able to see both the kms_to_miles and miles_to_kms functions ready to use.

```
import distance_converter as converter

print(converter.kms_to_miles(160))

```

```
Output:

99.36
```

And there you have it. The module is working elegantly, and you were able to convert the figures easily.

Sometimes, you might not wish to import the complete module with all the functions. It is a possible scenario that you may just want to use a specific function from the said module and wish to import only that. Makes sense, right? Fortunately, that too can be done. For this, let us assume I only wish to import the miles_to_kms function from the converter module. Here’s how I can do that:

```
from distance_converter import miles_to_kms
```

Right now, we know there were only two functions, but what if there were more than 10? To check, simply type the first few bites, and right after the space following the word ‘import’ press ctrl+space to access the list of functions available within the module. Choose the one you are interested in, and Bob is your uncle!

```
from distance_converter import miles_to_kms

print(miles_to_kms(100))
```

```
Output:
160.9
```

Let us do things a little differently here. Let us create a little function first whose job is to identify the largest number within a set of random numbers (like we did before) and store it as a separate module. We will then call the same and provide it with a list of numbers to identify the largest entry and print out the same on the console.

```
def high_number(numbers):
max = numbers[0]
for the number in numbers:
if number > max:
max = number
return max                    #We need a return value
```

Save this in a separate module called max_number.py and then later import the same.

```
from max_number import high_number

numbers = [879, 7564654, 65654, 654853, 6676]
max = high_number(numbers)
print(max)
```

```
Output:
7564654
```

Let’s see what happened here. We initially defined a function where we first began by declaring a variable with a numbers[0] value. We then initialized the ‘for’ loop to iterate over individual components of the list. The ‘number’ is a loop variable, followed by the ‘if’ statement that defines what condition needs to be met for ‘max’ to change value. In the end, we asked Python to return us with a value. Simple, right?

We then imported this into our new file, provided it with a set of list numbers. Here, we declared a variable called max (since this program has never seen max before) and gave it a value that will be driven through the function using the list of numbers. Eventually, we printed out the result, which, in our case, was the largest number of the list.

This is how we use modules. Now, it is time for the last bit of the learning process. This is where we meet packages and find out what they are.

Packages

Modules are files with codes in them, and similarly, packages are directories, or folders, that contain multiple files within them. We use packages to ensure that we organize our files accordingly. If you were to create hundreds of files and a good chunk of them belonged to calculations, you can create a package directory within PyCharm and Python with an appropriate name and move all such files within this new folder.

To create a package file, right-click on the project name and choose the Python package. Instantly, you will notice that it has one special file within it by default. It is the __init__ file that initializes the package and allows Python to know what this is.

Now, let us add a module to this newly created package. We shall call this ‘test1’ and give it a function within it. Remember, this file needs to be created in the new package we created.

```
def test():

print("This is just a test")
```

We gave this as a function for the module. Now, let’s browse into one of our old files within the previous folder. If you like, you can create an empty file as well. Now, since we are in a different folder, and we need to import this, things will be slightly different. If you try and use the previous method of “import test1”, it will not work as the directories are different. Instead, we will do this:

```
import test.test1
```

This lets Python know that you are importing a module called test1 from a Python package called test. Now, you will be able to use the function easily.

There are far too many packages and modules available online, directly from their sources, which you can use for quite a lot more than you might imagine. You can browse around the internet and find ones that may be more suited to your type of programming. However, the principles remain the same.

Now, we have effectively learned everything that a beginner should know about. You are all set to set sail and seek out your true calling and create programs that are ready to take the world by storm. All it needs is practice and patience.

Chapter 10

Your Next Step of the Journey!

Right away! Congratulations on arriving all the way here. You began as someone who had no idea of what Python was, what it was all about, and was probably afraid to dive into the world of programming. Now, you have all you need to create a perfect understanding of things. Now, you should be able to look at the code and know what is going on.

The journey gets more and more amazing from here on out. While it is sad that we will not be able to continue to journey ahead, I can at least rest assured that you will no longer be struggling with concepts and that you will be able to make sense of quite a lot of coding that you will now encounter.

Here is a quick look back into some of the incredible things we learned so far.

Let’s Revisit What We Learned!

- We downloaded and installed Python and PyCharm.
- We created our first-ever program. Yes, it was simple, but it was exciting!
- We learned about data types – strings, numbers, and Booleans.
- We learned how to create variables and recall them.
- We visited the logical operators.
- The confusing ‘if’ and ‘else’ statements.
- We certainly cannot forget the never-ending ‘for’ loops.
- The operators, the types, and their uses.
- Functions – how they enable us to do so much more.
- Lists, tuples, and dictionaries – where and why to use them.
- Classes

- Exception handling
- Inheritance
- Modules and packages

That is quite a bit of an achievement for any programmer who started with practically nothing in mind just a while ago. Now, you have all the tools and knowledge you need to get out there and do some serious programming.

While the learning never stops, it is recommended that you continue practicing as much as you can. I have written a workbook to ensure that you keep your knowledge to good use and remain confident in using your code whenever and wherever possible.

There are hundreds of fields that are waiting for you from this point out. These include cutting-edge frontiers like artificial intelligence, self-driving cars, machine learning, ethical hacking, website development, data science, the list is never-ending.

Remember

There are a few things you should always remember and be habitual of when seeking out a career in programming.

You should stick to the clean-code practice and write code that is easy to understand and read for any programmer. The cleaner you organize your code, the better and more effective you will be as a programmer. Remember to use modules and packages to organize your files and modules further accordingly. You never know when you may need them again. It is always a good idea to have the codes and functions you wrote stick around.

You can find various exercises online and even have some great tutorials from places like Udemy and Coursera to fine-tune your knowledge further. If you are stuck on a specific part and are unable to wrap your head around it, take a break and think it through. Most likely, you will find a solution on your own. If not, the internet awaits to answer all your distress calls.

Python is a well-documented language. There is every possibility that you may find your answers within the documents provided on Python’s website. Mr. Van Rossum certainly did think this one through. Make use of these documents where possible and get further details about things that otherwise might sound confusing to you.

As a programmer, think out of the box. Think about how you can change things and make them more simple. Take everyday examples where you think you can use your knowledge to create programs that can ease your life. You never know, you might just come across a commonly faced problem and fix it; one good thought is all that stands between you being you and you being the next big name for the industry.

With that said, I bid you all farewell and the best of luck. May your future be as bright as the sun!

References

Briggs, J, R. (2013): Python For Kids. San Francisco, CA. No Starch Press

Matthes, E. (2016): Python Crash Course. San Francisco, CA: No Starch Press

Payne, B. (2015): Teach Your Kids To Code. No Starch Press

Python Workbook

Learn How to Quickly and Effectively Program with Exercises, Projects, and Solutions

PROGRAMMING LANGUAGES ACADEMY

Introduction

This workbook has been created for practice and to allow readers of the crash course to further enhance their knowledge, understanding, and usage of Python as a programming language.

If you have already gone through “Python Programming For Beginners: The Ultimate Beginner’s Guide to Learning the Basics of Python in a Great Crash Course Full of Notions, Tips, and Tricks,” then this book is designed to further assist you in practicing all that you have learned so far.

Keeping true to the nature of the previous book, we will be looking into fun, intuitive, and challenging exercises. These will test your ability and knowledge as a programmer and ensure that you are always prepared to tackle situations, questions, and can identify errors. The exercises compiled here are taken from various sources, the links to which will be provided at the end of the book for your convenience. Should you like, you can visit these links for further exercises and test your programming skills to the max.

A great programmer is one who constantly practices his coding skills, can solve technical issues, and can identify the kind of solution required to resolve a situation. Our aim with this book is in-line with this concept, which is why you can expect various types of exercises, projects, and tests to learn from.

For your convenience, all solutions to questions and problems are provided in the last chapter. Refer to these when you feel you are unable to figure the problem out yourself. You may also refer to the first book from time to time to refresh your concepts and further clarify any ambiguities you may have during the process of learning and applying Python as a programming language.

There is no harm in admitting defeat. I assure you, I have been there a thousand times. If an exercise or question seems to be too much, remember to take it a little bit at a time. The better your state of mind, the clearer things will be for you.

Why Do I Need Exercises and Projects?

Ask any successful programmer in the world, and they will confirm the same: practice does make a programmer perfect. In the beginning, you might have been struggling with using PyCharm to write your code. Eventually, with a bit of practice, your pace started to increase. This is because you’ve now adapted to the IDE and the overall environment and feel of Python. The more you hone your skills, the more fluently you will type out your programs.

These exercises are designed to ensure you always have something to practice on. Once done with practicing the exercises as shown here, modify them at will to further create complex programs on your own. It is a perfect way to move into the intermediate and advanced levels of being a programmer. Every programmer goes through thousands of such miniature projects to gain perfect command of any language. While Python is comparatively easier to understand, do not underestimate the language. It can get tricky rather quickly, and with poor knowledge, you might end up going in circles.

The previous book has indeed taught you quite a few things. I have ensured that I created quizzes, exercises, and questions to test every bit of that knowledge. This way, not only do you get to recall what you learned, but you also get to see the code in action by applying the solutions you think would be right. I do not claim to be the greatest programmer, nor that I am anywhere near being one, but what I can guarantee is that these exercises will certainly keep you on your toes and get you a step closer to becoming a great programmer yourself.

The projects are designed for your independent exploration. You have the entire internet to help you out with inspiration and ideas. Use the knowledge you gathered in the previous book and the experience that you will gain here to come up with more complex, more interactive programs that you can write for the projects I have chosen for you. These are projects which can make their way into the market as well, if you think out of the box and apply slightly more advanced logic and knowledge. Nothing is impossible, and programming is no different.

How Much Time Should You Spare?

While there are no specific limits that I would like to set here, I would, however, recommend that you squeeze out about one hour a day, if not more, to practice these exercises. Some of these might be quite easy in the beginning, but the purpose of such questions or exercises isn't to test your know-how alone; it is to push you further to come up with practical usage of the knowledge.

Let me also clarify that by one hour a day, I do not mean that you spend 60 minutes of your time pacing through these exercises alone. Copy these exercises to your IDE and work with them. By the end of the book, you should have enough exposure to Python that you will be able to develop your own code by first analyzing the situation, roughly noting down the logic that would help you and then writing the actual code itself.

Once you gain the habit of writing codes daily, you will progress at a phenomenal rate and will hopefully be well on your way to becoming a full-fledged programmer within no time.

With that said, it is time for us to begin the second phase of the journey. This is where we find out just how much you have learned thus far and whether you have picked up the concepts correctly to solve some complex issues and answer questions based on the technicalities of Python.

Here's a tip: Keep your PyCharm open as you read through the book to carry out the exercises as we move along. Do not jump to the last chapter just because the problem is seemingly impossible to solve. Take your time and analyze the situation carefully. The answer is far more obvious than you might think!

Chapter 1: Warm-Up Time

Well, first of all, my heartiest congratulations to you for picking up this book. If you have already completed the previous book, where I explained Python programming, double the wishes for yourself. The journey to Python is indeed one that is riddled with lines and lines of codes, waiting to be explored, understood, and executed correctly.

We have gone through various chapters in the previous book and discovered so much about Python, starting from its history, all the way to the modern-day interpretation of automation, artificial intelligence, and how things so advanced use Python.

We went through individual aspects of the language, such as the syntax, the variables, the data types, loops, and functions, to count a few. All that is good, but the problem is we still don't know if we are ready to take on more advanced courses and learn things far beyond the scope of the previous book.

Fortunately, I already gave this some thought, which is why I will be providing you with various methods to fine-tune your skills. Further, I am eager to get started, are you? Then let us dive into the exercises straight away and find out just where we stand.

Solutions to all exercises and questions are within the last chapter. Only consult them when you have tried everything possible to come up with the answer and somehow failed to do so.

Sea of Questions!

These questions, while they may sound easy, are designed to revisit some basic elements of python. Some of these may have options, while others may not. Do not be intimidated by these questions. Try and answer as many of these as possible.

Q-1: From the options given below, identify which of these is written in Python?

Code-1:

```
using system;
var username = console.readline("Please enter your name: ");
console.write("Hello " + username);
```

Code-2:

```
<html>
   <head>
      <title>"Why write Python?"</title>
   </head>
   <body>
      <p>
         print("Hello World!")
      </p>
   </body>
</html>
```

Code-3:

```
import turtle
def my_function(name):
   print(f"Hello {name}")
my_function("Sam")
```

Code-4:

```
var name = "Mr. Marvel";

console.log("My name is " + name);
```

I am sure you see some familiar things here. Take your time and analyze each of these closely. The answer is right there, all it needs is a keen eye to pick it out! It’s quite interesting to see that most of these seem to be using a familiar setup. It does take time for one to be fully familiar with the syntax, but once you are familiar, you should have no trouble figuring this out.

Visit any of the old exercises we did in the previous book. Try and match the way the code was written to the ones presented here. You should have your answer shortly!

Moving on to our next question, I promise it will be more questions and less of me, but I need to ensure I provide some explanations were needed to help those who may have picked up the book after a while, just to refresh their concepts.

Let us now begin a series of questions to test your understanding.

Q-2: How can you check if you have Python 3.8.x installed on your system?

A. Check if you have PyCharm installed on your system. If so, you have the latest Python version installed.

B. Run the command *python --version* in PyCharm to check for the version.

C. Run the command *python* in the command prompt for Windows to check the version. Run *python -v* on Mac and *python3* on Linux to get the version.

D. Visit the Python website to see if it can identify your version of Python.

Q-3: What is the language named after?

A. Monty Python’s flying circus
B. The reptile Python
C. To honor an endangered species of Python
D. Just a random name that caught on

Q-4: How does each line in Python end?

A. With a colon ‘:’
B. With a semicolon ‘;’
C. With a full stop/period ‘.’
D. None of the above

Q-5: What does the acronym IDE stand for?

A. International Day for Electronics
B. Integrated Developing Environment
C. Integrated Developer Environment
D. Integrated Developing Engineering

Q-6: How is a string represented in Python?

A. With a single quotation mark ‘’
B. With a double quotation mark “”
C. With either of the above.
D. None of the above.

Q-7: What is a variable?

A. It’s a function in Python.
B. It’s a method in Python.

C. It’s a user-created container holding immutable values.

D. It’s a user-created container holding values that can be modified.

Q-8: How would you print a string that says He said, “Yes!”?

A. print(“He said, “Yes!”)

B. print(f“He said, “Yes!”)

C. print(‘He said, “Yes!”’)

D. print(He said, Yes!)

Q-9: Running the code as shown, what will the output be?

num = '5' * '5'

print(num)

A. 25

B. 5, 5, 5, 5, 5

C. ‘5’ * ‘5’

D. TypeError: Can’t multiply sequence by non-int of type ‘str’

Q-10: If you run a code that ends up with an error, it will cause PyCharm to crash.

A. True

B. False

C. Depends on the type of code written

D. None of the above

Q-11: Which is the correct method to set the value for a bool ‘is_married’?

A. “True”

B. True

C. ‘True’

D. true

Q-12: Which of the following is a formatted string?

Code-1:

```
name = "Jiovanni"

age = 41

print("Hi, I am name and I am age years old")
```

Code-2:

```
name = "Jiovanni"

age = 41

print(f"Hi, I am {name} and I am {age} years old")
```

Code-3:

```
name = "Jiovanni"

age = 41

print("Hi, I am " + name + " and I am " + age + " years old")
```

Code-4:

```
name = "Jiovanni"

age = 41

print("Hi, I am [name] and I am [age] years old")
```

Q-13: To name a variable called first name, which method is correct? You can choose more than one answer to the following question.

A. firstname

B. FirstName

C. first.name

D. first_name

Q-14: Choose one or more answers which apply. Clean-code practice is:

A. To keep our workstations clean.
B. To name our variables and functions appropriately.
C. To improve readability.
D. To ensure the code is not breaking any laws.

Q-15: Which of the following is the correct way to create a variable named ‘test’:

A. def test():
B. test.create
C. test = “”
D. print(test)

Q-16: What is a concatenation of strings?

A. To merge two or more strings into a new string object
B. To separate two strings
C. To convert an integer into a string
D. None of the above

Q-17: Choose the correct answer(s). Python is:

A. The successor of ABC language
B. Only operable through PyCharm IDE
C. Used for automation and Machine Learning
D. All of the above

Q-18: What is OOP?

A. Object-Oriented Python
B. Object-Oriented PyCharm
C. Object-Oriented Programming
D. Only On Python

Q-19: How can you acquire user input and store it in a variable called income, for calculation purposes?

A. income = bool(“Enter your income:”)
B. income = int(“Enter your income:”)
C. income = input(“Enter your income:”)
D. income = int(input(“Enter your income:”)

Q-20: Which of these is/are true regarding Python?

A. There are two data types.
B. Variables can be called, modified, or removed.
C. Python can work without PyCharm.
D. Python is a case-sensitive language.

That was quite a little warm-up, wasn’t it? We have only just begun. You may wish to check how many questions you got right by referring to the last chapter and seeing where you stand. These 20 questions were random, and most of them did not involve many technicalities.

Let us reflect on where you stand as a beginner:

If you achieved:

20 – Bravo! You answered each one of these brilliantly. That goes to show you were paying close attention to the book and the questions above. This is exactly the kind of result you should expect if you know your basics.

15 – 19 – So close to perfection, but do not let that bring you down. You did a fabulous job at answering these questions. Just revisit the ones you got wrong, and you should soon be polishing your skills to make it through with the finest results.

10 – 14 – There is room to improve. You did answer some of these questions rather well, but you have missed out on a few critical topics. It is best to revisit the book and focus solely on the ones you got wrong. Go back to the concepts you missed and try to understand how the knowledge was applied here.

Below 10 – I have no other way to put it besides saying you need to work on your skills. It is quite likely that you were distracted during the test or when you were reading the book. Do not be disappointed, though; failure is a part of the learning curve. As long as you have the will to learn, and a passion for pursuing, you will soon understand the issues and be writing your programs like anyone else.

I cannot stress enough that it is only through practice that you will be able to become the kind of programmer you aim to be one fine day. Just going through the book and answering every question right, only to stop practicing right after, will not serve you with any purpose or advantage.

There are a few things you should always remember as a programmer:

- Programming languages are constantly being updated. If you are out of action for a little while, you will soon be holding on to outdated knowledge.
- You are only as good as your coding skills. Anyone better will replace you instantly.
- Accuracy and speed of coding will differentiate between a good programmer and a struggling programmer, even if the latter knows everything.

Be sure to check out Python's official website for any updates that may arrive in the future. Currently, these codes were developed using Python 3.8.0, and they might have already been updated by the time you read this book. Always stay current with the latest version of Python and PyCharm.

Now that we have had a little stretch, it is time to dive into some technicalities and immerse ourselves deeper into the world of Python.

Is This Correct? - Part 1

This section will test out your skills in understanding the scenario and identifying whether the code will work or if it needs to be modified. I will be presenting you with various scenarios in each chapter. They will grow in complexity and length, which is why it is a must that you read through each of the lines properly. Try not to copy and paste the code before you have concluded the program's ability to work or fail. Let your brain do the work first. Train yourself and your mind to think, analyze, and resolve issues efficiently and effectively. Relying too much on the IDE will never allow you to explore your potential talent and problem-solving skills truly

Q-1: The program shown below was created to display a concatenated string. Do you think that the following will work? Will this deliver the required output of "This will be added With this!" or would it produce a completely different output? If so, why?

```
string1 = "This will be added "

string2 = "With this!"

print("string1 + string2")
```

Q-2: The following is a program written by a skilled programmer for a local business called "Pete's Garage" which should provide the business with a more reliable way of dealing with things. The program is written, as shown below:

```
print('Welcome to Pete's garage)

name = input('Please enter your name: ')

job_number = int(input('Please enter your job number: '))
```

```
repair_cost = 100
discount = 15
total = repair_cost - discount
print("{name}, the total for {job_number} is ${total}")
print('Thank you for your business')
```

Will this program work? If not, can you identify the error that might cause this program to crash or stop responding?

Q-3: A university student decided to piece together a program that will allow potential online students from abroad to fill out the form and seek out further information regarding courses they're interested in. The form looks like this:

```
#Online Registration Form
print("Welcome to ABC Uni!")
print("Please enter the required information to begin.")
s_f_n = input("Enter your name: ")
phone = int(input("Enter your phone number: ")
em = "Enter your email: "
crs = "Choose your course:
```

The student has asked you to review the program and find out if there are any issues that need addressing. Find out what is wrong with the code and correct the issues.

Q-4: A student has created a program for a login page at a library for the new batch of users that have just joined. After a brief introduction, all users are asked to create their username and password pairs. After the users have entered their passwords, each password is compared to ensure that it matches. The program looks like this:

```
username = input("Username: ")

password = input("Password: ")

print("You have entered the following:")

print(username.lower())

print({password.lower()})

print(password==password.lower)
```

What seems to be the problem here? Why do you think the code will not work? What can be done to ensure that the program starts functioning?

Q-5: A programmer, with intermediate experience and knowledge, was asked to type out a program that would print out Boolean values for every grade a student acquires. For grades from A to B, the prompt was to print out True, while others would be considered False. Have a look at this code and see if this will work:

```
grades = ["A", "A", "B"", "U", "F", "E", "D"]

for x in grades:

    if x == A or x == B:

        x = True

        if x == True:

            print("Pass")

    else:

        x = False

        print("Fail")
```

The student has claimed that the code worked exceptionally. It is now up to you to analyze the code without testing it first to deliver your first impressions.

Bonus: Try and modify the same code with your own values and come up with something even better to practice.

Q-6: A string named initial_message contains the following message:

"*Hi, I have just taken part in the course. I hope that I will be a programmer one day.***"**

Without the quotation marks, a programmer was asked to find out the length of the string. She used the following method to do so:

String.length(initial_message)

Is this the correct method to check the length of a string? If not, what is the correct method applicable here?

Once you are done with these exercises, I do believe you will be in a better position to understand and analyze your understanding of the basic data types, the syntax, and the way variables work. I did add a few situations which were not exactly dealing with data types and variables; however, expect such questions and problems to come up in the future as well. Now, you should cross-check your answers with the last chapter to find out how much you were able to get and if you solved them correctly.

For those who have managed to score more than 70%, good job! For those who are struggling, I would recommend going through the first book and revisiting the data types, variables, and input methods to refresh what you may have forgotten. Once again, there is no shame if you were not able to grasp the concept the first time.

Programming does take time, and sometimes, even the best programmers spend days trying to figure out what is causing their program to crash, just to find out they may have missed out on a single quotation mark, a comma, or misspelled a variable or a function. This is genuinely the case and happens almost every day.

It is with the introduction of PyCharm that our lives have been made so much easier. Imagine having to write entire programs, as long as 1000 lines, on a notepad, which does not even come with the capability to check the spelling, let alone identify errors for any language after typing.

Use the power of IntelliSense, a technology used within PyCharm, that allows us to complete our codes with the click of a button. While that is time-saving, I would still recommend that you type out the complete code. If you make a habit of using PyCharm’s IntelliSense, you might find yourself struggling, should you join a firm that relies on Microsoft’s VS Code IDE. It is always best to write the codes completely and proof-read the same where possible before you decide to execute them.

Now, we shall move on to our second chapter. We have revised and revisited the most basic concepts and principles of Python within this chapter. It is time to take the next step and start practicing with user-inputs, storing values, and recalling them. We will also look into matching information, and lastly, we will come across the first project that I would recommend every reader to go through and complete on their own.

All projects within the book have no definitive solutions. Apply the knowledge that you have gained so far to add to these projects and make them more complex and interactive.

Chapter 2: Recording Information

Programmers and their programs cannot exist without information. When I say information, I mean everything that has to do with data that is used for input, calculations, predictions, decision-making, and eventual output. Getting the right information but storing the same in the wrong variable or having it tied up with elements that simply do not belong to each other can confuse, and at times, massive problems.

As a programmer, we need to ensure we know the kind of information we need for a specific program to run successfully. We need to ensure and ascertain that the information is stored accordingly and only called upon, modified, or utilized when the situation calls for it.

We have covered quite many examples in the previous book, but here, we will go through certain scenarios that will further test your skills and critical thinking. Read the codes thoroughly, and you should be able to solve these with ease.

Storing/Recalling Information

While this section will mostly talk about the aspects of information itself, we will also be ensuring that we maintain our practice of clean-code to help programmers better understand the program itself. Expect a few issues where the program might be right, but the naming could use some help. In real-life situations, you will often encounter such issues and hence would need to re-address the names to make the program more meaningful and improve the overall readability of the code. With that said, let us begin!

Task-1: A YouTube streamer decided to conduct a survey where users were asked to provide feedback on what they would like to watch in the next stream. Your job is to create a program that uses the following information and prints out the result of what the user chose, along with a thank you message.

What shall I stream next?

a) Days Gone
b) Resident Evil 2
c) Fortnite
d) Apex Legends
e) Death Stranding
f) Surprise Us!

The ending message should be:

You have chosen (option). I appreciate your time and hope to see you in the next one!

The exercise is fairly simple. Most of these will require you to print out information and then store the user-input value. Design the program so that it can understand what 'a' or 'b' or any character that the user chooses is, and then print the same out in the end greetings. You do not have to worry about printing out the name in the end. Just the letter of the selection will do for now.

Remember, the case-sensitive nature of the program still haunts us. Make use of methods like *.lower()* to ensure it matches our requirements. I will be providing my version of the program in the last chapter as a solution. For now, keep thinking, keep coding.

Hint: While you can use an 'if' statement, I would recommend bypassing that for now. Try and think of more basic means to store and recall values. This can easily be done without the use of logic and if/else statements. We will revisit this exercise in the future to make it a little more complex and appealing at the same time.

Task-2: A dentist wishes to have a program created for his website where the customers will be presented with multiple services. The customer will choose the option and will be presented with a total for the service that is payable by the customer. The services are given, as shown below:

a) Root Canal Therapy - $250

b) **Oral Hygiene Check - $50**
c) **Emergency Injury Treatment - $100**
d) **Post-Procedure Check-up - $150**
e) **Routine Check-ups and Consultation - $75**

For advanced payments, customers get a 50% discount.

Design a program that provides the customer with all the necessary information and gives a total according to what the customer chooses.

Confused? Let me give you a hint. You cannot store two values within a single variable unless you intend to create a list here. We are not aiming for that. You can either create two separate variables, service_a and price_a, or you can simply use conditional statements to enhance the program further. I leave the decision up to you.

I would prefer to use the latter, though. It is a lot easier and less messy.

So far, we have seen two simple exercises. Now, you will have probably realized that our programs can grow increasingly big and lengthy, the more complex we try to make them. The more information you have with you, the more lines of code will be used up. In typical situations, you will encounter programs that span well over 400 lines on average, and these are simple scripts used by programmers for various purposes.

I honestly do not mean to scare or intimidate you at all, but here's a little fact for you to digest.

The Mac OS X is believed to be the largest program ever written. It contains well over… wait for it… 85 million lines of code!

If you were to print that out and lay the regular A-4 sized papers in a straight line, you would cover a considerably large distance.

However, we are not here to set a world record, at least not yet. Our motive is rather simpler and kinder to our fingers and mind.

These exercises will test or have already tested your knowledge of creating variables, storing user-input information, modifying them (perhaps), and recalling them for simple calculations as well. At the heart of every program lies good and solid information. Without this, programs will have no reason to function.

So now that we have catered to these two exercises, let us move on ahead and see what else we can do.

Task-3: A college campus has decided to create a program that will determine the eligibility of an applicant based on a few questions and conditions. The college in question has asked you to create a program to record the following pieces of information:

- **a) First name**
- **b) Last name**
- **c) Age**
- **d) Overall score on their latest test result (out of 600)**
- **e) If seeking scholarship**

Based on the following conditions, the eligibility for admission and the scholarship will be decided:

For Admission:

- **The student should have achieved at least a 60% overall score or above for admission.**

For Scholarship:

- **The student must have at least a test score of 80% to be eligible for the scholarship.**

Create a program with data from three different students who have acquired a 471, 354, and 502 accordingly. Print out their results based on the above conditions.

This exercise will be fairly long and will require you to use the information and conditional statements to execute it to perfection. It might be wise to point out that such projects are actually in place, and a loosely similar module is being used for Canada’s Competitive Ranking System (CRS) score calculation for immigration purposes as well.

When solving complex programs and situations, anticipate where you should set variables and use them. If you miss out on key positions, your program will have information, but it will not function or be utilized properly.

It is essential to point out that you will need to have a basic understanding of mathematical operations like multiplication, division, and so on, as these greatly help you create better programs.

Python allows you to get as creative as you can. If you believe you are bound by restrictions, you will be surprised to learn that there aren't any. You can apply Python to practically everything to come up with programs that can tell you if you have the right ingredients for a recipe or the right amount to pay off your debts. You can use it to make a predictive program that can predict possible outcomes so that you may prepare for all of the eventualities and so on.

Major business organizations use such programs, which may seem simple but have quite a lot going on in the background. As a programmer, it is our job to realize what needs to be done, how it will be executed, and how the desired result can be achieved. Always use a blank paper to draw out the flow chart. There is no specific flowchart for you to follow, which means that you can come up with your ideas on your own and map them out. It greatly helps with the programming part of the entire exercise. Go ahead and try it for the exercise above (if you haven't already done the exercise); otherwise, use one for any of the exercises you will encounter later on.

Is This Correct? - Part 2

Once again, we will be diving into some programs which I have created and/or compiled from various sources. Your goal is to see and analyze, *without using PyCharm or any IDE*, if they could or would work.

Q-1: A programmer came up with a program that would find the highest number from a given set of numbers. The numbers provided were stored as a list in a list variable called 'number_data' and the program that he designed looked like this:

```
number_data = [323, 209, 5900, 31092, 3402, 39803, 78341, 79843740,
895, 6749, 2870984]

for number in number_data:

        if num < number:

         num = number

print(num)
```

Will the above code work? What’s wrong with the code?

Do not worry about the loops and if statements, try and analyze the error here. Take a guess and then check your answer. You can then try it on PyCharm to see whether things work or crash.

Q-2: A freelance programmer was tasked with creating a simple program to determine the eligibility of a profile for an auto-loan. Based on some specific information and conditions, such as the candidate should be less than 45 years of age, must have a minimum of a certain number as income and should not have any criminal records, the program was to determine if the same person was eligible for a loan or not. The programmer wrote the following program:

```
print("Your doorway to auto-loan eligibility check!")

print("Please provide complete information for best results")

name = input("Please enter your full name: ")

age = int(input("Enter your age: "))

income = int(input("Please enter your income per month: "))

nature_of_job = input("Do you work full-time, part-time or as a
freelancer?: ")

has_license = input("Do you have a valid license? [y/n]: ")

if has_license.lower() == "y":
    has_license = True
else:
    has_license = False
has_criminal_record = input("In the last 5 years, do you have any criminal
records? [y/n]: ")
if age > 45 and income >= 8000 and has_license == True and
has_criminal_record == False:
    print("You are eligible for a loan")
elif age < 45 and income >= 5000 and has_license == True and
has_criminal_record == False:
    print("You are eligible to apply for a loan")
elif has_criminal_record:
    print("You are not eligible for a loan")
elif income < 5000:
    print("You are not eligible at this time")
else:
    print("Please be patient as one of our specialists will be in touch!")
```

Upon executing a sample, the result was as follows:

```
Your doorway to auto-loan eligibility check!
Please provide complete information for best results
Please enter your full name: John Smith
Enter your age: 38
Please enter your income per month: 8300
Do you work full-time, part-time, or as a freelancer?: Full-time
Do you have a valid license? [y/n]: y

In the last 5 years, do you have any criminal records? [y/n]: n

You are not eligible for a loan

Process finished with exit code 0
```

Do you think the program executed correctly? If not, what do you think the issue is?

Sometimes, the solution is rather obvious. You should be able to recognize the error straight away if you can link the dots and see which value is being recalled and what value/status the variable is printing. I cannot give you any more hints than that. Try and think this one through. Once sorted, let us move on to another one.

Q-3: As a school project, every student was asked to come up with a program that is no longer than 10 lines and is able to do some basic mathematics to produce answers. The student came up with a simple program that asks the user to type in a number and will let them know if the number is even or odd. The program is as shown below:

```
print("Setting the ODDS, EVEN!")

num = input("Enter a number: ")

if (num % 2) = 0:

        print("{0} is Even")

else:

        print("{0} is Odd")
```

Do you think the program will work? What errors, if any, do you think, would cause problems for the student?

Simple games can sometimes be quite entertaining. While checking the code myself, I spent over an hour working on it for no reason. I modified the values, the conditions, and had quite the time.

Here is one more situation, see if you can figure out the issue on your own.

Q-4: As a side project, a programmer decided to create a simple program that can let users know if the year, mentioned by the user, is a leap year or not. The leap year is calculated by determining if the year is exactly divisible by the number '4' and in the case of a century year, as the year 2000, it must be exactly divisible by 400.

Using the above concept, the programmer wrote this code:

```
print("My Brilliant Little Leap Year Calculator!")

year = int(input("Please enter the year: "))

if (year / 4) == 0:

  if (year / 100) == 0:

        if (year / 400) == 0:

         print(f"{year} is a leap year")

        else:

         print(f"{year} is not a leap year")

  else:

        print(f"{year} is a leap year")

else:

  print(f"{year} is not a leap year")
```

When the code was run with the year 2020, the following was the response:

```
My Brilliant Little Leap Year Calculator!

Please enter the year: 2020

2020 is not a leap year

Process finished with exit code 0
```

Why do you think that is?

(Programiz: https://www.programiz.com/python-programming/examples/leap-year)

The above is a brilliant idea. Although we are a little too late, you can come up with more interesting ways to use such inspirations and programs to come up with something more meaningful.

Programmers across the world have been keenly looking into ways to simplify further things we normally overlook. Think good and think hard about matters in life which require some work and can be improved. Come up with your genuine ideas or try and improve programs that already exist out there.

The internet is full of such programs. You will find these on a great many platforms, forums, and social websites. Just browse through these, check out the source code, and see if you have the right to modify the code to come up with something vibrant and improved.

I believe it is time for us to look into something new, something fresh, and something that is genuinely challenging.

Project - 1

Before I provide you with your first project, let me quickly shed some light on what you can expect from these projects.

Every project will be unique, as each one of us will have different ideas about how to carry out the task and execute the same. The projects will be designed to provide you with seemingly simple tasks, only to find out that you may have to do a little more than just copying and pasting blocks of code from one file to another.

Use your coding knowledge from all sources, as these will not be bound to individual chapters. Projects are where you will encounter all kinds of problems, situations, and scenarios. To solve these, or finish them successfully, you will need to use various methods right from the beginning, all the way to the end of complex matters like functions, classes, and modules.

I will be providing you with links from which you can download specific modules, libraries, or classes to help further make the process easier. You already know how to import them into your PyCharm using the “from x import y” or “import xyz” method. Try and make a simple-looking scenario complex and interesting. Continue developing these projects with advanced knowledge that you will hopefully gain after this book. A program is never truly complete. Even the best programs and software continue to be updated with newer knowledge, modules, and variations.

Keep on practicing and adding more to these projects. Who knows, you might end up with something far superior and more useful than just a message that says “Hello World” at the end.

Task:

Create a simple game of “Rock, Paper, Scissors,” where the computer randomly generates value and asks the user to input their selection. The result should show whether the user wins or loses, or if it is a draw.

Requirements:

To complete this project, you will need to use the following:

Packages:

From random import randint – This will be your first line of code. Random comes pre-installed and allows you to force the computer to randomize the selection. This will help you in ensuring that every turn is unique and unpredictable.

There are quite a few ways you can complete this project. As a reference, I will share my solution for this project at the end of the book as well.

Please note that I wish to encourage you to explore the world of Python and use your genuine approaches, communicate with the community, and learn better ways to code. For this reason, I will not share the details on the projects moving forward. I will gladly share the answers to questions and solutions to other problems. The rest, I invite you to use your power of deduction and programming to learn better.

So far, we have gone through some exercises, questioned what was right and what was not. We even initiated our very first project, which is quite challenging in all fairness. However, everything hinges on how well you understand your basics. The better you know them, the easier it will be to move from a beginner to an intermediate programmer and eventually to a skilled programmer. If you are unsure about certain aspects, it is always a good habit to revisit the concepts and revise what you have learned.

Time to say goodbye to variables and storing values and move on to our friends, the statements, and loops.

Chapter 3

Running Around in Circles - Literally!

The world of loops and conditional statements is one that involves quite a lot of thinking. These will test your analytical and critical thinking, your problem-solving ability and will put you in rather uneasy spots. The trick behind each one of these is to understand how they work carefully.

As always, I would personally recommend using a pen and a paper, or your choice of text editor on your computer, to first draw out the scenario using flow charts and diagrams. To give you an idea, here is one:

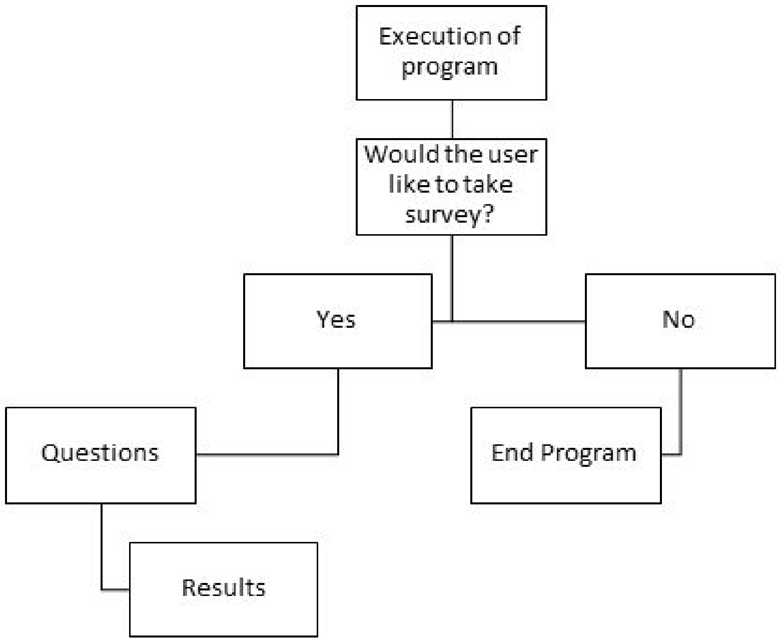


Table showing a simple 'if' conditional statement flow of a process.

Using tables or flow charts greatly helps us ease matters and develop a better understanding of the nature of the task at hand. Many great programmers first jot their ideas down into smaller parts to ensure that these are looked upon individually, where possible. By doing so, they can then focus on such charts to fully understand how the program needs to function and what the outcomes will be based on the path the user takes.

Similarly, it is highly recommended that you use such tables or charts to help you fully develop a command of ‘if’ and ‘else’ statements and ultimately master them. There is no other way to say this, so I will just say it: If you wish to create intelligent programs, you can never do so without developing a thorough understanding of these conditional statements and loops.

With that said, let us continue our quest to perfect our understanding of conditional statements and loops.

To ‘if’ or ‘for’ - That Is the Question!

Quite a lot of times, even I would spend days trying to figure out if we need to use the ‘if’ and ‘else’ conditions or opt to settle for a ‘for’ and ‘while’ combination. This is one of the trickiest aspects of programming, but it is one that can certainly deliver the program much-needed quality.

I will be posing you with various questions and scenarios. Where possible, I will also let you know of the desired outcome that you should achieve. Your job, as a programmer, is to figure out whether you need to use conditional statements, loops, or a combination of both to achieve a said outcome.

This will be tricky, which is why you must use your favorite IDE and have a go at these. Come up with your possible solutions and match those with the ones provided at the end of the book to see if you were successfully able to crack these open.

Task-1: A programmer has been asked to create a simple program where he is to map out digits from zero to nine in words. The program will ask a user to enter his/her number, and the program will print the same out in text instead. The desired result is as shown below:

Please, enter your number: ***415602397***

Output: ***Four One Five Six Zero Two Three Nine Seven***

How do you suppose this can be achieved? Would we need to use a loop here or a set of conditional statements?

Not as easy as it sounds, is it? I can start you off with a hint; use a dictionary to create key-value pairs for numbers and words.

What you need to do next is your call. Take your time as there is no time limit imposed on you. Right now, you are practicing and learning how to be a better programmer. You can utilize your time well and without any deadlines to meet. In real-world scenarios, you may need to know all this beforehand so that you do not end up wasting time. If you are not able to do this, someone else will.

Here is our second task to think about and solve.

Task-2: A student carried out a program that calculated the shipping cost for an online retailer for the customer. The program would base the shipping cost on the total of the cart and the country of residence of the customer in question.

The chart below shows the shipping cost details:

Country	**Total**	**Shipping cost**
US	<$50 $50 - $99 $100 - $249 >$250	Free $10 $25 $50
AU	<$50 $50 - $99 $100 - $249 >$250	$10 $20 $50 $100
CA	<$50 $50 - $99 $100 - $249 >$250	$5 $15 $30 $75

UK	<$50 $50 - $99 $100 - $249 >$250	$20 $25 $55 $110

Using the information above, the student was successfully able to create the program.

What do you think the student did?

This might be simple, but think this through and try to make this one as pleasing as possible. If you wish to take on a bit of a challenge, add lists and tuples to the mix to test yourself further.

Every program that you come across here can be done in hundreds of ways, if not thousands. To me, the glass may seem half empty, and to you, it may seem half full. It is completely based on how we analyze things and look at them.

Put your thinking cap on, and for the next one, do not copy and paste the code onto your PyCharm just yet. See if you can spot the issue with this one.

Task-3: You are a programmer who has been tasked with creating a simple yet intelligent game that stores a name that the users will have to guess. Upon providing the wrong name, the program will provide hints. You have created the following program; however, there seems to be something wrong here.

```
name = 'James'

guess = input("I have a name. Can you try to guess it?: ")

guess_num = 0

max_guess = 5

while guess != name and guess_num == max_guess:

    print(f"I am afraid, that's not quite right! Hint: letter {guess_num +1} ")
    print(guess_num + 1, "is", name[guess_num] + ". ")
    guess = input("Have another go: ")
    guess_num = guess_num + 1
if guess_num == max_guess and name != guess:
    print("Alas! You failed. The name was", name + ".")
else:
    print("Great, you got it in", guess_num + 1, "guesses!")
```

Try not to jump to your IDE to figure this one out. First, take a moment or two and analyze what is causing this program to end almost immediately upon providing an incorrect name. Surely, there must be something that is not right.

Read carefully between the code lines, and you should soon be able to figure the matter out. Try and resolve the issue and then try the possible solution on your IDE to see if it works.

Once again, I do encourage you to make things better by modifying the code to your liking. There are hundreds of games you can come up with, which are simple yet purely entertaining for others. As programmers, we take pride in knowing that we were able to execute codes with efficiency and ease. Programs will get more complex as you progress along in your programming journey. For those interested in deep learning and machine learning, expect hundreds of lines of code to train the machine. To do that, you will need to be well-versed with almost every method in existence, all the functions, and modules that are available across the internet to make the most out of the experience.

Project - 2

Time for yet another project. Since we are discussing games, create a Python program that lets the user know their astrological sign from the given date of birth. The program may seem rather easy, but once you look into the smaller details, you will soon realize that this will require you to think a little out of the box.

For this project, I will not be providing hints nor a model to follow. You already know, and you should be able to execute this one with ease and a bit of finesse as well. You do not need any special modules or packages to get this project done. All you need is a quick search on the internet to see which star sign starts when to get you going.

Through trial and error, you should be able to create a program that can work easily and exceptionally. Should you encounter issues, try and resolve them on your own instead of looking for a solution on the internet.

If such projects interest you, you can find many more by searching for “Python projects for beginners” and get started. The more projects you work on, the better you will learn. Keep an eye out for what is in demand these days and set your target to one day be able to carry out programming of a level that will get you paid handsomely.

Questions and Answers

I wish this were a game where we could see who leads with how many points and know who answered the questions correctly. However, we will not let that get in the way of learning.

Answer as many questions as you can. To make this interesting, time yourself and try to answer all the questions within 10 minutes. At the end of the 10-minute mark, stop and review your answers to see where you stand.

Q-1: What does the == operator do?

A. It assigns a value

B. It recalls and matches the value of variables before and after it

C. It lets Python know not to equate variables

D. None of the above

Q-2: What is wrong with the code below?

```
x = 20

y = 30

z = 40

if x > y:

print("Something's wrong here")
```

A. Since x is less than y, the program will crash and return an error
B. z is not called; hence the program will not function
C. The condition is not followed by an indentation
D. There is nothing wrong with the program

Q-3: What will the result of the following program be?

```
alpha = 'Bravo'

bravo = 'Charlie'

charlie = 'Alpha'

for char in alpha:

    if char != 'a':

        print(char)
```

A. Bravo
B. Charlie
C. Alpha
D. None of the above

Q-4: What is the difference between 100 / 30 and 100 // 30?

A. It is just a typing mistake.

B. Both will deliver the same results.

C. The / will show a floating figure while the // will show an integer remainder.

D. The / will show an integer remainder while the // a float remainder.

Q-5: What will the code shown below print as a result if a car is traveling at 75 miles per hour?

```
car_speed = int(input("Enter Car's current speed: "))

acceleration = 20 #per second

top_speed = 100

time = 0 #in seconds

if car_speed == 0:

    time = top_speed // acceleration

    print(f"It should take {time} second(s) for the car to reach its top speed")

    #For a stationary vehicle

else:

    time = (top_speed - car_speed) // acceleration

    print(f"it would take {time} second(s) to hit max speed.")

    #For a vehicle in motion
```

A. 5 second(s)

B. 3 second(s)

C. 1 second(s)

D. The program will return an error

Q-6: When should you use a 'for' loop?

A. When we need one specific output
B. When we need to iterate over a range of elements
C. When we wish to set a certain condition to be either true or false
D. None of the above

Q-7: What does the following error indicate?

```
Traceback (most recent call last):
  File "C:/Users/Programmer/PycharmProjects/PFB/PFB-2/Project-2.py", line 1, in <module>
car_speed = int(input("Enter Car's current speed: "))
ValueError: invalid literal for int() with base 10: 'abc'
Process finished with exit code 1
```

A. The program crashed due to an invalid value entry
B. The program crashed as ‘abc’ was not entered as a string
C. Used single quotes instead of double quotes
D. None of the above

Q-8: The following program uses the log10 module from the ‘math’ package. The program was designed to carry out a few arithmetic operations to test the values and functionality of the program. What seems to be wrong here?

```
from math import log10
a = input("Enter 1st value: ")
b = input("Enter 2nd value: ")
print(a, "+", b, "is", a + b)
print(a, "-", b, "is", a - b)
print(a, "*", b, "is", a * b)
```

print(a, "/", b, "is", a / b)

print(a, "%", b, "is", a % b)

print(f"The base 10 logarithm of {a} is {log10(a)}")

*print(a, "^", b, "is", a**b)*

A. The strings are not formatted properly

B. The input values are stored as strings instead of integers

C. The log10 function will not work within a formatted string

D. All of the above

Q-9: What does an ‘elif’ statement do that ‘else’ can’t?

A. Elif conditions are secondary conditions that are executed when the main condition is false

B. Elif statements do not require conditions whereas else statements do

C. Elif statements are exactly the same as else statements

D. None of the above

Q-10: What does the following produce as a result?

```
def high_number(numbers):
    max = numbers[0]
    for number in numbers:
    if number < max:
    max = number
    return max
list = [21, 200, 31, 1, 39]
print(high_number(list))
```

A. 39

B. 5

C. 200

D. 1

Q-11: It is necessary to use ‘if’ statements every time you use loops. Is this statement true or false?

And stop! Hopefully, you were able to do all of these within the time limit of 10 minutes. I do not expect that this was an easy task as 10 minutes is not exactly much, but then again, it was a challenge.

Well done for scoring as many as you did. The challenge wasn’t to see who could get the most answers correct. It was actually designed to ensure you continue practicing. Even if you were able to get one of these correct, using your knowledge and not just a wild guess, it shows that you were paying attention and trying to understand the problem to come up with a solution. That is exactly the kind of attitude and commitment, which will take us closer to success.

Take a break if you have been practicing for a while. It is said that a human mind needs a quick minute or two to relax after every 45 minutes. You have earned it. Once you are back, let us move ahead and try to analyze more programs and see where people have gone wrong.

Is This Correct? - Part 3

This is honestly becoming my favorite section already. We get to see many programs and get to point the errors out, and in the process, we get to learn so much more. In fact, we were even inspired by some of these programs to do something similar. Keep an eye out for programs that may seem interesting as you can always modify the way they work. However, this isn’t always allowed by the law.

It is best to ensure you first know your rights regarding copying or editing said code. For this book, you do not have to worry about it much.

Q-1: A programmer decided to create a simple program, just to practice basic ‘if’ and ‘else’ conditions. He wrote the following program:

```
name = "John"

age = 33

is_married = True

is_happy = input("Are you happy?: ")

if is_happy.lower() == "yes":

    print("Well done!")

else:

    print('Sorry to hear that')
```

While the program runs fine, there is something that is wrong. Can you figure out what it is? You should be able to remove it, and the program will still continue to function properly.

Q-2: A student defined a function, as shown below:

```
def kms_to_miles(distance):

    distance * 0.621
```

When trying to use it, the program returned a value of ‘None’ as a result. Why do you think that happened?

A. The student must not have passed the appropriate parameter.

B. The distance used would have been in miles, hence the error.

C. The student forgot to use ‘return’ before the calculation while defining the function.

D. I have no idea why this failed. It should have worked.

Q-3: Do you think the following program should work? If not, why?

```
prices = [5, 10, 15, 20, 25]

total = 0

for item in prices:

     total += item

print(f"Your total price is: ${total}")
```

Do not try and copy the code to your IDE. Try and analyze the situation first to see if you can spot an error.

Q-4: In our previous book, we went through an example program, as shown:

```
for an in range(3):

    for b in range(3):

        for c in range(3):

            print(f"({a}, {b}, {c})")
```

If you were to change the values of the ranges from top to bottom to 3, 2, 1, respectively, would the program work? What will the outcome be?

Q-5: According to a student, indentation is unnecessary and should not cause any problems when executing a program. The other student is of the idea that Python pays attention to whitespace, and hence the indentation is quite important to maintain the code and arrange it accordingly. Which of the two do you think is right?

Q-6: Look at the code snippet below. It was taken from a program that was designed to iterate over key pairs of a dictionary.

```
output = ""

for char in number:

    output += words.get(char) + " "

print(output)
```

What does the += operator do here?

Q-7: A teenager wanted to print out a simple design on python as a result. The design shown below was the output:

```
*

**

***

****

*****

******

*******

********

*********

**********

I did this!

**********

*********

********

*******

******

*****

****

***

**
```

```
*
```

Do you think this can be done using loops? If so, can you code the program?

For those who may remember the days when DOS was considered a new thing, we surely created a lot of these patterns and designs. Now, Python can carry that out, but instead of just relying on print statements, we use loops to do extensive work for us instead.

There are more modules and packages such as turtle, which helps you create some incredible designs. Turtle is a built-in, pre-installed package, and you should be able to import it easily. You can easily find tutorials online to see what the entire thing is all about and possibly use the same to see just how effortlessly Python can do our work for us.

Python is far more than just a coding language. You can get so much done by just typing a few lines. With the help of your favorite IDE, things are sure to enhance the experience and keep the learning curve high. With python, you always get to learn something new.

Speaking of new, it is time for us to look into our next chapter and come across another aspect of programming which we covered in the last book: Functions!

Chapter 4

Using the Right Functions

We have already seen how functions can greatly help us with carrying out and organizing massive blocks of code into a simple, word, or two long recallable functions. If that isn't enough, Python brings forth some of the finest pre-defined functions to the table to further assist us in carrying out so many other tasks flawlessly.

While we will not be diving into functions that are far above the scope of a beginner, we will still be looking into problems where we will see whether given functions would help us out or not.

Take some time to ensure you go through the previous book in case you need to refresh your memory and revisit the concepts behind what functions are and how they operate. Once sorted, let us proceed with our exercises, questions, and scenarios.

Getting Functioning Programs to Work

The objective of this section of the book is to reinforce the concepts related to functions. To make this a little more interesting than others, I will be providing you with scenarios to carry out on your own and convert into fully functioning programs. You will be provided with a solution at the end. All you need is to ensure you read the scenario and visualize it to sort out what needs to go where, and then form a fully functional program.

From this point forward, I will be providing you with various scenarios, each consisting of programs that you will need to create. Some of these may require you to do a bit of research as well. However, if any special method, library, module, or package is used, I will provide those within the instructions.

Q-1: You are to create a function that calculates a taxi fare. The taxi fare is comprised of a base fare of $3.00 and then $0.10 for every 100 meters traveled. Create a function that takes distance as its only parameter (in km) and returns the value of the total applicable fare. Follow up with a program to show the functioning nature of the function.

The situation is rather easy. All you need is to work on the function and work out the details accordingly. The rest will just fall into place automatically. Take your time and process the information.

I know it is easy to be overwhelmed. A quick look at the internet will only leave you confused. Take small steps and start somewhere. With time and a bit of understanding, you will soon grasp the concept and be able to figure out such situations easily.

The next one to come is even tougher. We have entered the phase where we can say goodbye to the five to 10 line programs. It is best to let them go and practice on bigger, more complex programs to polish your skills and make yourself step out of your comfort zone and properly start exploring.

Q-2: A client has asked you to create a reusable program containing reusable functions. The first situation is to create a function that creates a virtual deck of playing cards.

Since there are 52 cards, the cards with numeric values from two to nine will be represented by their respective numbers. For 10, Jack, Queen, King, and Ace, you are required to use T, J, Q, K, and A.

Following the numeric/alphabetic value will be another character to represent the suit. Use h for hearts, c for clubs, d for diamonds, and s for spades.

You should create a function that does not take any parameter and uses loops to iterate through all the cards and store them with a two-character abbreviation in a list. The function should only return this list as a result.

Hint: You will need to use the following as your beginning line.

from random import randrange

For this, expect quite a few lines of code. Your complete knowledge about Python will be tested and tried in this one and the ones to come ahead. There is no need to rush into things. If you are not able to do it the first time around, you can do a bit of research and get some suggestions. I would recommend not to jump to the solutions right away. Try and push your brain to think a little outside the box. There is much to be learned, and the only way it can be learned is to try.

Q-3: You have been asked by a colleague to help out with a Python project. The assignment is to create a function that can generate a random password for a user, which would contain between 6 to 8 characters at most.

For this, you will need to use the following:

from random import randint

shortest_pass = 6

max_pass = 8

min_ASCII = 33

max_ASCII = 126

The function should generate a random value from positions 33 to 126 of the ASCII table. This function will not take any parameters.

The above exercise will test you further and might even require a little research on the ASCII table if you haven't seen one before. It is always nice to carry out a bit of research as it greatly helps us as a programmer to further excel at what we do.

One of the toughest calls to make when creating a function is knowing the right parameter you need to use to make the function work. You can use almost any argument you pass through the parentheses to make it into a parameter. Sometimes, you may not need a parameter at all, as we saw in some of the exercises above. In such cases, it is best to leave the parameter blank.

When in doubt, always consult the Python documentation to learn more about the various parameters you can make use of when defining functions. For recalling functions, hover your mouse over the parentheses, and a little prompt should display the kind of parameters you can use with said function.

Is This Correct? - Part 4

Q-1: Below is a user-made function that is designed to iterate through a given range and look for the highest number. Will the function work when it is called?

```
def high_number(numbers):
        max = numbers[0]
        for number in numbers:
         if number < max:
          max = number
        return max
list = [21, 200, 31, 1, 39]
high_number(list)
```

Q-2: What seems to be the issue with the following?

```
def this_function():
   print("Hello From This Function!")
this_function_with_args(name, greeting):
   print(f"Hello {name}, From This Function!, I wish you {greeting}")
this_function()
```

```
this_function_with_args()
```

Q-3: What would this function do?

```
def plus(a,b):
  sum = a + b
(sum, a)
sum, a = plus(3,4)
print(sum)
```

Q-4: Can you place a loop within a function, as shown below?

```
def plus(*args):
  total = 0
  for i in args:
        total += i
  return total
print(plus(20,30,40,50))
```

Now that we have revisited quite a few concepts and methods, and functions as well, let us head over to our final project and see what it is all about.

Final Project – Hangman

Remember the old game called Hangman? The one that involved blanks and a limited number of guesses to guess a movie, a name, a person, a city, or something else? For your final project, I decided to come up with a tough one. A project that will use almost everything you have learned.

To make it even better, if you execute the program correctly, you can store it as a recallable function or create a separate package so that you can use this over and over again. While there are hundreds of variations for this game online, use your unique approach. Feel free to browse the internet to get some inspiration.

Requirements:

You will need to use the following as your first line:

import time

Let this be a project that you are proud of once it is finished. By the end of this project, rest assured you are ready to take on the challenges and offer some exquisite programming skills to those who require programmers such as yourself.

The journey of a programmer does not end here. There are far too many things that lay ahead, which you will need to keep pace with. Learn about various libraries, modules, and packages to see how they can refine your projects.

For those interested in Machine Learning, Automation, Artificial intelligence, and Deep Learning, you will come across some names like Scikit-Learn, Turtle, and a few more. While it is still too early to jump into these, it is a good idea to look at them and see how they perform in action.

Chapter 5

The Solutions!

Finally, a chapter where you do not have to worry about solving exercises. You have gone through every curveball, every scenario, and every exercise provided within this workbook, and for that, you deserve the heartiest congratulations. I am glad that you took your best shot at solving these exercises.

For every programmer, the beginning is always the biggest hurdle. Once you set your mind to things and start creating a program, things automatically start aligning. Your brain automatically omits the needless information through its cognitive powers and understanding of the subject matter. All that remains then is a grey area that we discover further through various trials and errors.

There is no shortcut to learn to program in a way that will let you type codes 100% correctly, without a hint of an error, at any given time. Errors and exceptions appear even for the best programmers on earth. There is no programmer that I know of personally who can write programs without running into errors. These errors may be as simple as forgetting to close quotation marks, misplacing a comma, passing the wrong value, and so on. Expect yourself to be accompanied by these errors and try to learn how to avoid them in the long run. It takes practice, but there is a good chance you will end up being a programmer who runs into these issues only rarely.

With that said, it is time to shift our focus back to the main part of this chapter, the solutions to the never-ending exercises. Some of these were incredibly simple, while others were not as simple as they sounded. Regardless of how many you get right, you should never be afraid of nor let down by failure. It is a part of our learning cycle and should be accepted, understood, and then corrected. The more you learn from your mistakes, the easier things will be in the future.

Chapter by Chapter Solutions

Chapter 1 Solutions

Below are the answers, and the correct answer applicable to the question is highlighted in bold text.

Q-1: The question asked to identify which of the codes mentioned used Python as its programming language.

- Code 1: Unity C# - We do not use the statement ‘using’ in Python, nor do we use semicolons at the end.
- Code 2: HTML – The second language in the list used HTML, which is used for developing websites.
- **Code 3: Python – Evident from the import statement and the missing semicolon.**
- Code 4: JavaScript – Similar to C# but with a few differences.

Q-2: How can you check if your system has Python 3.8.x installed on your system? Let’s see what the correct method is to check for the same:

A. PyCharm – PyCharm is an IDE and can be installed without having Python onboard.
B. Command on PyCharm – The given command will not work on PyCharm.
C. Command on console/terminal – This is the correct method to locate the version of python.
D. The website will not indicate the version you have installed.

Q-3: What is the language named after? This was related to the history that we covered in the previous book. If you got this wrong, it is okay. The correct answer is A. The language was named after the famous Monty Python’s flying circus.

Q-4: How does each line in Python end? To be honest, this was one of the easiest questions. But surprisingly, people still tend to forget.

A. With a colon ‘:’
B. With a semicolon ‘;’
C. With a full-stop/period ‘.’
D. **None of the above** – **Because every line in Python ends without any special character except when defining conditions or functions.**

Q-5: What does the acronym IDE stand for? We have discussed this in the previous book.

A. International Day for Electronics
B. Integrated Developing Environment
C. Integrated Developer Environment
D. Integrated Developing Engineering

Q-6: How is a string represented in Python?

A. With a single quotation mark ‘’
B. With a double quotation mark “”
C. With either of the above. – We can create strings using single or double quotation marks.
D. None of the above.

Q-7: What is a variable? Once again, an easy question.

A. It’s a function in Python.
B. It’s a method in Python.
C. A user-created container holding immutable values is close but not exactly the right answer as variables can change.
D. It’s a user-created container holding values that can be modified.

Q-8: How would you print a string that says He said, “Yes!”?

A. print(“He said, “Yes!”)
B. print(f“He said, “Yes!”)

C. **print(‘He said, “Yes!”’) – Using double quotation marks would end the string at the start of “Yes”**
D. print(He said, Yes!)

Q-9: Running the code as shown, what will the output be?

num = '5' * '5'

print(num)

A. 25
B. 5, 5, 5, 5, 5
C. ‘5’ * ‘5’
D. **TypeError: Can’t multiply sequence by non-int of type ‘str’ – A string cannot be multiplied by another string.**

Q-10: If you run a code that ends up with an error, it will cause PyCharm to crash.

A. True
B. **False – PyCharm is a safe working environment that is designed to test codes. If the code will crash, the IDE returns the error code to inform users that this will crash when used outside PyCharm.**
C. Depends on the type of code written
D. None of the above

Q-11: Which is the correct method to set the value for a bool ‘is_married’?

A. “True”
B. **True**
C. ‘True’
D. true

Q-12: Which of the following is a formatted string?

<u>Code-1</u>:

name = "Jiovanni"

age = 41

print("Hi, I am naming, and I am age years old")

<u>Code-2</u>**:**

name = "Jiovanni"

age = 41

print(f"Hi, I am {name} and I am {age} years old")

This is the correct way to format a string. There are other ways you can format strings, but I find this a lot easier.

<u>Code-3:</u>

name = "Jiovanni"

age = 41

print("Hi, I am " + name + " and I am " + age + " years old")

<u>Code-4</u>:

name = "Jiovanni"

age = 41

print("Hi, I am [name] and I am [age] years old")

Q-13: To name a variable called the first name, which method is correct? You can choose more than one answer to the following question.

- **A. firstname**
- **B. FirstName**
- C. first.name
- **D. first_name**

If you are surprised about the second entry here, remember that it is recommended not to use this format to name variables. While it is not recommended, it does not mean that it is wrong.

Q-14: Choose one or more answers which apply. Clean-code practice is:

A. To keep our workstations clean.
B. To name our variables and functions appropriately
C. To improve readability.
D. To ensure the code is not breaking any laws.

Q-15: Which of the following is the correct way to create a variable named 'test':

A. def test():
B. test.create
C. test = "" – You can always create variables as blanks
D. print(test)

Q-16: What is a concatenation of strings?

A. To merge two or more strings into a new string object
B. To separate two strings
C. To convert an integer into a string
D. None of the above

Q-17: Choose the correct answer(s). Python is:

A. The successor of the ABC language – This was discussed in the previous book.
B. Only operable through PyCharm IDE
C. Used for automation and Machine Learning – These are two of the most in-demand fields of modern times.
D. All of the above

Q-18: What is OOP?

A. Object-Oriented Python
B. Object-Oriented PyCharm
C. Object-Oriented Programming – Remember, everything in Python is considered an object.
D. Only On Python

Q-19: How can you acquire user input and store it in a variable called income, for calculation purposes?

A. income = bool(“Enter your income:”)
B. income = int(“Enter your income:”)
C. income = input(“Enter your income:”)
D. income = int(input(“Enter your income:”) – The user will input a value that will be then converted into an integer.

Q-20: Which of these is/are true regarding Python?

A. There are two data types
B. Variables can be called, modified, or removed.
C. Python can work without PyCharm
D. Python is a case-sensitive language

Yes! Python can work without PyCharm as PyCharm is just one of the many ways to operate Python and write codes.

Is This Correct? – Part 1 Solutions

This was indeed a section that caused quite a bit of confusion but provided the perfect opportunity to practice our coding skills and keep an eye out for errors. Let us look at the solutions to the questions posed within the first part of the “Is This Correct?” series.

Q-1: The program shown below was created to display a concatenated string. Do you think that the following will work? Will this deliver the required output of “This will be added With this!” or would it produce a completely different output? If so, why?

string1 = "This will be added "

string2 = "With this!"

print("string1 + string2")

Ans: The actual answer would be as shown below if you were to run this code as it is:

```
string1+string2
```

This happened because the print statement was given a string that contained the characters string1 and string2. Python took them quite literally as a string object. To make them work properly, you will need to use a formatted string. The solution would be as shown here:

```
string1 = "This will be added "

string2 = "With this!"

print(f"{string1}{string2}")
```

Output:

```
This will be added With this!
```

Q-2: Following is a program written by a skilled programmer for a local business called “Pete’s Garage” which should provide the business with a more reliable way of dealing with things. The program is written, as shown below:

```
print('Welcome to Pete's garage)

name = input('Please enter your name: ')

job_number = int(input('Please enter your job number: '))

repair_cost = 100

discount = 15

total = repair_cost - discount

print("{name}, the total for {job_number} is ${total}")

print('Thank you for your business')
```

Will this program work? If not, can you identify the error that might cause this program to crash or stop responding?

Ans: To begin with, there are a few errors that need addressing. Beginning at the very top, the first print statement is not correctly written. The string ends at the apostrophe after Pete, and hence the program will not function nor identify the rest.

Moving forward, the print statement declaring the total is not formatted. It is missing the ‘f’ key character. Once these two errors are sorted, the program should function rather well.

If you try the program now, it should display this:

```
Welcome to Pete's garage

Please enter your name: Emma

Please enter your job number: 91829

Emma, the total for 91829 is $85

Thank you for your business

Process finished with exit code 0
```

Q-3: A university student decided to piece together a program that will allow potential online students from abroad to fill out the form and seek further information regarding courses they’re interested in. The form looks like this:

```
#Online Registration Form

print("Welcome to ABC Uni!")

print("Please enter the required information to begin.")

s_f_n = input("Enter your name: ")

phone = int(input("Enter your phone number: ")

em = "Enter your email: "

crs = "Choose your course:
```

The student has asked to review the program and determine if any issues need addressing. Find out what is wrong with the code and correct the issues.

Ans: Once again, we see that the phone variable has two opening parentheses but only one closing parenthesis. Fix that first to ensure that this line of code is functional. Next, the ‘em’ variable has a fixed string value. But we are trying to get input from the user. Use the input function here. Lastly, the ‘crs’ has an opening quotation mark but no closing quotation mark. It is also missing the input function.

Lastly, if needed, a print statement can be added to confirm the selection. The overall program should look like this:

```
#Online Registration Form

print("Welcome to ABC Uni!")

print("Please enter the required information to begin.")

s_f_n = input("Enter your name: ")

phone = int(input("Enter your phone number: "))

em = input("Enter your email: ")

crs = input("Choose your course: ")

print(f"{s_f_n}, you have chosen {crs} as your course.")

print(f"Details of the {crs} course will be emailed to you at {em}")

print(f"We may also be in touch via a phone call at {phone}")
```

Output:

```
Welcome to ABC Uni!

Please enter the required information to begin.

Enter your name: Joel

Enter your phone number: 915789654

Enter your email: joel@abcxyz.com

Choose your course: Game Development
```

Joel, you have chosen Game Development as your course.

Details of the Game Development course will be emailed to you at joel@abcxyz.com

We may also be in touch via a phone call at 915789654

Q-4: A student has created a program for a login page at a library for the new batch of users that have just joined. After a brief introduction, all users are asked to create their username and password pairs. After the users have entered their passwords, each password is compared to ensure that it matches. The program looks like this:

```
username = input("Username: ")

password = input("Password: ")

print("You have entered the following:")

print(username.lower())

print({password.lower()})

print(password==password.lower)
```

What seems to be the problem here? Why do you think the code will not work? What can be done to ensure that the program starts functioning?

Ans: The program would not have worked as it would have printed out a case-sensitive password in all lower-case. A comparison between the two would then obviously return ‘false’ as they would not be the same.

Remember, Python is a case-sensitive language. Password is not the same as PASSWORD or password. To make this a fully functioning program, you will need to do a little more than just tweak parentheses or fill out missing values.

Here is my take on the same program.

```
username = input("Username: ")
```

```
password = input("Password: ")
print("You have entered the following:")
print(username)
print(password)
login = False
login_id = input("Please enter your username: ")
login_pwd = input("Enter your password: ")
while not login:
    if login_id == username and login_pwd == password:
        print("You have logged in successfully")
        login = True
    elif login_id == username and login_pwd != password:
        print("You have entered the wrong password")
        break
    elif login_id != username and login_pwd == password:
        print("You have entered the wrong ID!")
        break
    else:
        print("You have entered an invalid ID/Password")
        break
```

Q-5: A programmer, with intermediate experience and knowledge, was asked to type out a program that would print out Boolean values for every grade a student acquires. For grades from A to B, the prompt

was to print out True, while others would be considered False. Have a look at this code and see if this will work:

```
grades = ["A", "A", "B"", "U", "F", "E", "D"]

for x in grades:

    if x == A or x == B:

        x = True

        if x == True:

           print("Pass")

    else:

        x = False

        print("Fail")
```

The student has claimed that the code worked exceptionally. It is now up to you to analyze the code without testing it first to deliver your first impressions.

Ans: Once again, we see some misplaced quotation marks in the first line. The first order of the day is to correct those right away. Then we have our 'if' statements. See how the statement says if x == A. Here, we need to add quotation marks for A to be a string object instead of a separate variable. Here is the result:

```
grades = ["A", "A", "B", "U", "F", "E", "D"]

for x in grades:

          if x == 'A' or x == 'B':

           x = True

           if x == True:
```

```
 print("Pass")
else:
 x = False
 print("Fail")
```

Output:

```
Pass
Pass
Pass
Fail
Fail
Fail
Fail
```

Q-6: A string named initial_message contains the following message:

“*Hi, I have just taken part in the course. I hope that I will be a programmer one day.***”**

Excluding the quotation marks, a programmer was asked to find out the length of the string. She used the following method to do so:

String.length(initial_message)

Is this the correct method to check the length of a string? If not, what is the correct method applicable here?

Ans: No. That is not the correct way to check the length of the string. The correct way to do so is using the *len()* function, as shown here:

print(len(initial_message))

This will then count the characters and print out the value of characters in return.

Chapter 2 Solutions

Task-1: A YouTube streamer decided to conduct a survey where users were asked to provide feedback on what they would like to watch in the next stream. Your job is to create a program that uses the following information and prints out the result of what the user chose, along with a thank you message.

What shall I stream next?

- **a) Days Gone**
- **b) Resident Evil 2**
- **c) Fortnite**
- **d) Apex Legends**
- **e) Death Stranding**
- **f) Surprise Us!**

The ending message should be:

You have chosen (option). I appreciate your time and hope to see you in the next one!

Ans: This is true quite an easy one to do. Here is the solution for the survey that this streamer is trying to create.

```
print("Welcome to my survey, where I ensure I deliver what you want!")

print("Please, take some time to fill this out and help me decide what to play next!")

a = "Days gone"

b = "Resident Evil 2"

c = "Fortnite"

d = "Apex Legends"

e = "Death Stranding"

f = "Surprise Us"
```

```
print(f"""Here are your options. Remember, select one:
a){a},
b){b},
c){c},
d){d},
e){e},
f){f}""")
selection = input("Please make a selection: ")
print(f"You have chosen {selection}. I appreciate your time and hope to see you in the next one!")
```

And the corresponding output would look like this:

```
Welcome to my survey, where I ensure I deliver what you want!
Please, take some time to fill this out and help me decide what to play next!
Here are your options. Remember, select one:
a)Days gone,
b)Resident Evil 2,
c)Fortnite,
d)Apex Legends,
e)Death Stranding,
f)Surprise Us
Please make a selection: b
You have chosen b. I appreciate your time and hope to see you in the next one!
```

And that would be as simple as that. If you were able to do this quickly, well done. If not, now you have every idea of how it works. Do remember that this is an elementary level of programming. You can make this as interactive as you wish. Now, you can try using various methods to make it even better.

Task-2: A dentist wishes to have a program created for his website where the customers will be presented with multiple services. The customer will choose the option and will be presented with a total for the service that is payable by the customer. The services are given, as shown below:

- **a) Root Canal Therapy - $250**
- **b) Oral Hygiene Check - $50**
- **c) Emergency Injury Treatment - $100**
- **d) Post-Procedure Check-up - $150**
- **e) Routine Check-ups and Consultation - $75**

For advanced payments, customers get a 50% discount.

Design a program that provides the customer with all the necessary information and gives a total according to what the customer chooses.

Ans: I already gave you a hint, but now, let us see how this would work.

```
print("The Patient's Portal")

print("Please select the service you would like to come in for.")

a = "Root Canal Therapy"

print(f"A){a}")

b = "Oral Hygiene Check"

print(f"B){b}")

c = "Emergency Injury Treatment"

print(f"C){c}")

d = "Post-procedure checkup"
```

```
print(f"D){d}")
e = "Routine Checkups and consultation"
print(f"E){e}")
selection = input("Please choose one: ")
print(f"You chose {selection}")
total = 0
if selection.lower() == "a":
    total = 250
    print(f"Your total is ${total}")
elif selection.lower() == "b":
    total = 50
    print(f"Your total is ${total}")
elif selection.lower() == "c":
    total = 100
    print(f"Your total is ${total}")
elif selection.lower() == "d":
    total = 150
    print(f"Your total is ${total}")
else:
    total = 75
    print(f"Your total is ${total}")
print("Did you know? Book in advance and get 50% off!")
payment_time = input("Would you like to pay today? [y/n]: ")
```

```
if payment_time.lower() == "y":
    total = total * 0.50
    print(f"Your total payable is ${total}")
else:
    print(f"You will need to pay ${total} at the counter.")
print("Have a smiling day!")
```

The output should show you the following:

```
The Patient's Portal
Please select the service you would like to come in for.
A)Root Canal Therapy
B)Oral Hygiene Check
C)Emergency Injury Treatment
D)Post-procedure checkup
E)Routine Checkups and consultation
Please choose one: a
You chose a
Your total is $250
Did you know? Book in advance and get 50% off!
Would you like to pay today? [y/n]: y
Your total payable is $125.0
Have a smiling day!
```

The program is a little long, but it does keep you engaged and keeps you on your toes to type in the correct code to make a successful program. If you were able to figure this one out on your own, brilliant. If not, it is perfectly okay. There is plenty of time to learn what you need to do to carry out various tasks to come up with your desired results.

Task-3: A college campus has decided to create a program that will determine an applicant's eligibility based on a few questions and conditions. The college in question has asked you to create a program to record the following pieces of information:

- **a) First name**
- **b) Last name**
- **c) Age**
- **d) Overall marks in their latest test result (out of 600)**
- **e) If seeking scholarship**

Based on the following conditions, the eligibility for admission and the scholarship will be decided:

For Admission:

- **The student should have achieved at least a 60% overall score or above for admission.**

For Scholarship:

- **The student must have at least a test score of 80% to be eligible for the scholarship.**

Create a program with data from three different students where they have acquired 471, 354 and 502 accordingly. Print out their results based on the above conditions.

Ans: This program also required some critical thinking and posed a bit of a challenge with three different situations. To address these, we need our friendly ‘if’ statements to help us out and create an intelligent program.

```
print("Welcome to the Applicant’s eligibility checker")

first_name = input("Please enter your first name: ")

last_name = input("Please enter your last name: ")
```

```
age = input("Please enter your age: ")

marks = float(input("Please enter your overall marks (out of 600): "))

total_marks = 600.0

passing_marks = total_marks * 0.60

marks_for_scholarship = total_marks * 0.80

if marks >= passing_marks:

    print(f"Congratulations {last_name}! You are eligible for admission
to the college!")

    scholarship = input("Are you seeking a scholarship? [Y/N]: ")

    if scholarship.lower() == "y" and marks >= marks_for_scholarship:

     print(f"Congratulations {last_name}! You are eligible for a
scholarship!")

    elif scholarship.lower() != "y":

     pass

    else:

     print(f"{last_name}, you are not eligible for a scholarship at this
time!")

else:

    print(f"Unfortunately {last_name}, you are not eligible for
admission.")

print("Thank you for your input and we wish you good luck!")
```

The output for the above would be as shown below:

```
Welcome to the Applicant’s eligibility checker

Please enter your first name: John
Please enter your last name: Doe

Please enter your age: 21

Please enter your overall marks (out of 600): 402

Congratulations Doe! You are eligible for admission to the college!

Are you seeking a scholarship? [Y/N]: y

Doe, you are not eligible for a scholarship at this time!

Thank you for your input, and we wish you good luck!
```

Is This Correct? – Part 2 Solutions

Q-1: A programmer came up with a program that would find the highest number from a given set of numbers.

The numbers provided were stored as a list in a list variable called 'number_data' and the program that he designed looked like this:

```
number_data = [323, 209, 5900, 31092, 3402, 39803, 78341, 79843740, 895, 6749, 2870984]

for number in number_data:

    if num < number:

     num = number

print(num)
```

Will the above code work? What's wrong with the code?

Ans: The above code will not work as the variable 'num' is not defined. While some may have been unnecessarily worried, the solution was rather simple. Just declare a variable called num with a starting value of zero.

```
number_data = [323, 209, 5900, 31092, 3402, 39803, 78341, 79843740, 895, 6749, 2870984]

num = 0
```

```
for number in number_data:
    if num < number:
     num = number
print(num)
```

Remember to declare variables before they are used as Python reads the program line by line.

Q-2: A freelance programmer was tasked with creating a simple program to determine the eligibility of a profile for an auto-loan. Based on some specific information and conditions, such as the candidate should be less than 45 years of age, must have a minimum of a certain amount as income and should not have any criminal records, the program was to determine if the same person was eligible for a loan or not. The programmer wrote the following program:

```
print("Your doorway to auto-loan eligibility check!")
print("Please provide complete information for best results")
name = input("Please enter your full name: ")
age = int(input("Enter your age: "))
income = int(input("Please enter your income per month: "))
nature_of_job = input("Do you work full-time, part-time or as a
freelancer?: ")
has_license = input("Do you have a valid license? [y/n]: ")
if has_license.lower() == "y":
    has_license = True
else:
    has_license = False
```

```
has_criminal_record = input("In the last 5 years, do you have any criminal
records? [y/n]: ")

if age > 45 and income >= 8000 and has_license == True and
has_criminal_record == False:

    print("You are eligible for a loan")

elif age < 45 and income >= 5000 and has_license == True and
has_criminal_record == False:

    print("You are eligible to apply for a loan")

elif has_criminal_record:

    print("You are not eligible for a loan")

elif income < 5000:

    print("You are not eligible at this time")

else:

    print("Please be patient as one of our specialists will be in touch!")
```

Upon executing a sample, the result was as follows:

```
Your doorway to auto-loan eligibility check!

Please provide complete information for best results

Please enter your full name: John Smith

Enter your age: 38

Please enter your income per month: 8300

Do you work full-time, part-time, or as a freelancer?: Full-time

Do you have a valid license? [y/n]: y

In the last 5 years, do you have any criminal records? [y/n]: n

You are not eligible for a loan
```

```
Process finished with exit code 0
```

Do you think the program executed correctly? If not, what do you think the issue is?

Ans: The above program has one major issue which will cause the program to always push out not eligible as a result. The has_criminal_record has not yet been defined as True or False. For that, we need to create a separate condition. Here is the fully operable program with a sample test.

```
print("Your doorway to auto-loan eligibility check!")

print("Please provide complete information for best results")

name = input("Please enter your full name: ")

age = int(input("Enter your age: "))

income = int(input("Please enter your income per month: "))

nature_of_job = input("Do you work full-time, part-time or as a
freelancer?: ")

has_license = input("Do you have a valid license? [y/n]: ")

if has_license.lower() == "y":

    has_license = True

else:

    has_license = False

has_criminal_record = input("In the last 5 years, do you have any criminal
records? [y/n]: ")

if has_criminal_record.lower() != "y":

    has_criminal_record = False

else:

    has_criminal_record = True
```

```
if age > 45 and income >= 8000 and has_license == True and has_criminal_record == False:
    print("You are eligible for a loan")
elif age < 45 and income >= 5000 and has_license == True and has_criminal_record == False:
    print("You are eligible to apply for a loan")
elif has_criminal_record:
    print("You are not eligible for a loan")
elif income < 5000:
    print("You are not eligible at this time")
else:
    print("Please be patient as one of our specialists will be in touch!")
```

Output:

```
Your doorway to auto-loan eligibility check!
Please provide complete information for best results
Please enter your full name: Elliot Charrington
Enter your age: 41
Please enter your income per month: 6900
Do you work full-time, part-time, or as a freelancer?: full
Do you have a valid license? [y/n]: y
In the last 5 years, do you have any criminal records? [y/n]: n
You are eligible to apply for a loan
```

Q-3: As a school project, every student was asked to come up with a program that is no longer than 10 lines and can do some basic

mathematics to produce answers. The student came up with a simple program that asks the user to type in a number and will let them know if the number is even or odd. The program is as shown below:

```
print("Setting the ODDS, EVEN!")

num = input("Enter a number: ")

if (num % 2) = 0:

    print("{0} is Even")

else:

    print("{0} is Odd")
```

Do you think the program will work? What errors do you think, if any, would cause problems for the student?

Ans: This program has a few issues. Firstly, notice that the print statements are not formatted. Secondly, the condition set is not using the right comparison operator. Instead of ‘=’, we need to use the ‘==’ operator to provide the condition with a comparison point.

Moreover, the input will initially be stored as a string. Be sure to use the *int* converter. Lastly, replace the ‘0’ in the print statements with the variable ‘num’, and that should do it. If you run the program now, it should be fully functional and should return the correct results.

```
print("Setting the ODDS, EVEN!")

num = int(input("Enter a number: "))

if (num % 2) == 0:

    print(f"{num} is Even")

else:

    print(f"{num} is Odd")
```

Q-4: As a side project, a programmer decided to create a simple program that can let users know if the year, mentioned by the user, is a leap year or not. The leap year is calculated by determining if the year is exactly divisible by the number '4' and in the case of a century year, as the year 2000, it must be exactly divisible by 400.

Using the above concept, the programmer wrote this code:

```
print("My Brilliant Little Leap Year Calculator!")

year = int(input("Please enter the year: "))

if (year / 4) == 0:

  if (year / 100) == 0:

        if (year / 400) == 0:

         print(f"{year} is a leap year")

        else:

         print(f"{year} is not a leap year")

  else:

        print(f"{year} is a leap year")

else:

  print(f"{year} is not a leap year")
```

When the code was run with the year 2020, the following was the response:

```
My Brilliant Little Leap Year Calculator!

Please enter the year: 2020

2020 is not a leap year

Process finished with exit code 0
```

Why do you think that is?

Ans: This one involved a little more mathematics than actual programming. Using the ‘/’ operator, we would not have received the exact values. Instead, we will need to change all the ‘/’ operators to ‘%’ and run the program again. Now, the year 2020 will be shown as a leap year, and any other leap years will be correctly calculated as well.

```
print("My Brilliant Little Leap Year Calculator!")

year = int(input("Please enter the year: "))

if (year % 4) == 0:

   if (year % 100) == 0:

          if (year % 400) == 0:

           print(f"{year} is a leap year")

          else:

           print(f"{year} is not a leap year")

   else:

          print(f"{year} is a leap year")

else:

   print(f"{year} is not a leap year")
```

Output:

```
My Brilliant Little Leap Year Calculator!

Please enter the year: 2020

2020 is a leap year
```

Project – 1 Solution

(Exercise 118: Stephenson, B. (2014) The Python Workbook)

As promised, here is how I have created a simple game of “Rock, Paper, Scissor” using Python.

```
from random import randint
t = ["Rock", "Paper", "Scissors"]
computer = t[randint(0, 2)]
tries = 0
player = " "
while tries <= 9:
        player = input("Rock, Paper, Scissors?")
        tries += 1
        if player == computer:
         print("Tie!")
        elif player == "Rock":
         if computer == "Paper":
         print("You lose!", computer, "covers", player)
         else:
          print("You win!", player, "smashes", computer)
        elif player == "Paper":
         if computer == "Scissors":
          print("You lose!", computer, "cut", player)
         else:
          print("You win!", player, "covers", computer)
        elif player == "Scissors":
         if computer == "Rock":
          print("You lose...", computer, "smashes", player)
        else:
         print("You win!", player, "cut", computer)
        else:
         print("That's not a valid play. Check your spelling!")
        computer = t[randint(0, 2)]
```

You may copy this and try it out yourself. Change the values where possible to see how it affects the program. This is one terrific way to spend some quality time with a game while learning the dynamics as well.

Chapter 3 Solutions

Task-1: A programmer has been asked to create a simple program where he is to map out digits from zero to nine in words. The program will ask a user to enter his/her number, and the program will print the same out in text instead. The desired result is as shown below:

Please, enter your number: ***415602397***

Output: ***Four One Five Six Zero Two Three Nine Seven***

How do you suppose this can be achieved? Would we need to use a loop here or a set of conditional statements?

Ans: To achieve this simple goal, we will use a dictionary to create key-value pairs. Here is the solution, and honestly it is worth a try.

```
number = input("Phone number: ")
words = {
        "1": "One",
        "2": "Two",
        "3": "Three",
        "4": "Four",
        "5": "Five",
        "6": "Six",
        "7": "Seven",
        "8": "Eight",
        "9": "Nine",
        "0": "Zero"
}
output = ""
for char in number:
        output += words.get(char) + " "
print(output)
```

This should now print out words instead of numbers.

Task-2: A student carried out a program that calculated the shipping cost for an online retailer for the customer. The program would base the shipping cost on the total of the cart and the country of residence of the customer in question.

What do you think the student did?

Ans: Refer to the original chart in the question for reference purposes. A successful version of the program would be as shown here:

```
total = int(input("Please enter the total amount: "))
country = input("Country [US/AU/CA/UK]: ")
if country.upper() == "US":
    if total <= 99 and not total <= 49:
     print("Shipping Cost is  $10")
```

```
          elif total >= 100 and not total >= 250:
           print("Shipping Cost is $25")
          elif total >= 250:
           print("Shipping Costs $50")
          else:
           print("FREE")
if country.upper() == "AU":
          if total <= 99 and not total <= 49:
           print("Shipping Cost is  $20")
          elif total >= 100 and not total >= 250:
           print("Shipping Cost is $50")
          elif total >= 250:
           print("Shipping Costs $100")
          else:
           print("Shipping Cost is $10")
if country.upper() == "CA":
          if total <= 99 and not total <= 49:
           print("Shipping Cost is  $15")
          elif total >= 100 and not total >= 250:
           print("Shipping Cost is $30")
          elif total >= 250:
           print("Shipping Costs $75")
          else:
```

```
         print("Shipping Cost is $5")
if country.upper() == "UK":
        if total <= 99 and not total <= 49:
         print("Shipping Cost is  $25")
        elif total >= 100 and not total >= 250:
         print("Shipping Cost is $55")
        elif total >= 250:
         print("Shipping Costs $110")
        else:
         print("Shipping Cost is $20")
```

The output would provide you the correct answers according to the input.

Task-3: You are a programmer who has been tasked with creating a simple yet intelligent game that stores a name that the users will have to guess. Upon providing the wrong name, the program will provide hints. You have created the following program, however, there seems to be something wrong here.

```
name = 'James'
guess = input("I have a name. Can you try to guess it?: ")
guess_num = 0
max_guess = 5
while guess != name and guess_num == max_guess:
        print(f"I am afraid, that's not quite right! Hint: letter {guess_num +1} ")
        print(guess_num + 1, "is", name[guess_num] + ". ")
        guess = input("Have another go: ")
```

```
    guess_num = guess_num + 1
if guess_num == max_guess and name != guess:
  print("Alas! You failed. The name was", name + ".")
else:
    print("Great, you got it in", guess_num + 1, "guesses!")
```

Ans: Believe it or not, there was only one slight error. The program worked, but the problem was that it would accept even an incorrect entry as correct. If you pay close attention to the 'while' condition, you will notice that guess_num == max_guess is never met, hence this block of code never gets executed. When that happens, there is nothing to add an increment to the number of guesses. The program would then move on to the 'if' statement, and hence choose the 'else' part as the output. To correct this, all you needed to do was to replace the 'while' condition to this:

```
while guess != name and guess_num != max_guess:
```

Now, the program should function properly and make this into an interactive little game.

Question and Answers

Q-1: What does the == operator do?

A. It assigns a value
B. It recalls and matches the value of variables before and after it
C. It lets Python know not to equate variables
D. None of the above

Q-2: What is wrong with the code below?

```
x = 20
y = 30
z = 40
if x > y:
```

print("Something's wrong here")

A. Since x is less than y, the program will crash and return an error
B. z is not called; hence the program will not function
C. The condition is not followed by an indentation
D. There is nothing wrong with the program

Q-3: What will the result of the following program be?

alpha = 'Bravo'

bravo = 'Charlie'

charlie = 'Alpha'

for char in alpha:

if char != 'a':

print(char)

A. Bravo
B. Charlie
C. Alpha
D. None of the above – The result would print "B r a v o"

Q-4: What is the difference between 100 / 30 and 100 // 30?

A. It is just a typing mistake
B. Both will deliver the same results
C. The / will show a float figure while the // will show an integer remainder
D. The / will show an integer remainder while the // a float remainder

Q-5: What will the code shown below print as a result if a car is traveling at 75 miles per hour?

car_speed = int(input("Enter Car's current speed: "))

acceleration = 20 #per second

top_speed = 100

```
time = 0 #in seconds

if car_speed == 0:

    time = top_speed // acceleration

    print(f"It should take {time} second(s) for the car to reach its top speed")

    #For a stationary vehicle

else:

    time = (top_speed - car_speed) // acceleration

    print(f"it would take {time} second(s) to hit max speed.")

    #For a vehicle in motion
```

A. 5 second(s)
B. 3 second(s)
C. 1 second(s) – We are using the // operator, which will always return an integer value
D. Program will return an error

Q-6: When should you use a ‘for’ loop?

A. When we need one specific output
B. When we need to iterate over a range of elements
C. When we wish to set a certain condition to be either true or false
D. None of the above

Q-7: What does the following error indicate?

```
Traceback (most recent call last):

  File "C:/Users/Programmer/PycharmProjects/PFB/PFB-2/Project-2.py", line 1, in <module>

    car_speed = int(input("Enter Car's current speed: "))

ValueError: invalid literal for int() with base 10: 'abc'
```

```
Process finished with exit code 1
```

- **A. The program crashed due to an invalid value entry**
- B. The program crashed as ‘abc’ was not entered as a string
- C. Used single quotes instead of double quotes
- D. None of the above

Q-8: The following program uses the log10 module from the ‘math’ package. The program was designed to carry out a few arithmetic operations to test the values and functionality of the program. What seems to be wrong here?

```
from math import log10

a = input("Enter 1st value: ")

b = input("Enter 2nd value: ")

print(a, "+", b, "is", a + b)

print(a, "-", b, "is", a - b)

print(a, "*", b, "is", a * b)

print(a, "/", b, "is", a / b)

print(a, "%", b, "is", a % b)

print(f"The base 10 logarithm of {a} is {log10(a)}")

print(a, "^", b, "is", a**b)
```

- A. The strings are not formatted properly
- **B. The input values are stored as strings instead of integers**
- C. The log10 function will not work within a formatted string
- D. All of the above

Q-9: What does an ‘elif’ statement do that ‘else’ can’t?

- **A. Elif conditions are secondary conditions that are executed when the main condition is false.**

B. Elif statements do not require conditions, whereas else statements do.
C. Elif statements are exactly the same as else statements.
D. None of the above.

Q-10: What does the following products as a result?

```
def high_number(numbers):
    max = numbers[0]
    for number in numbers:
     if number < max:
      max = number
    return max
list = [21, 200, 31, 1, 39]
print(high_number(list))
```

A. 39
B. 5
C. 200
D. 1 – If you chose 200 because I named the function high_number, you missed out on the fact that I used a < operator instead of >, hence the result would always be the smallest number in the list.

Q-11: It is necessary to use ‘if’ statements every time you use loops. Is this statement true or false?

Ans: False – There have been many cases where we did not use ‘if’ statements when we used a loop.

Is This Correct? – Part 3 Solutions

Q-1: A programmer decided to create a simple program, just to practice basic ‘if’ and ‘else’ conditions. He wrote the following program:

```
name = "John"

age = 33

is_married = True

is_happy = input("Are you happy?: ")

if is_happy.lower() == "yes":

    print("Well done!")

else:

        print('Sorry to hear that')
```

While the program runs fine, there is something that is wrong. Can you figure out what it is? You should be able to remove it, and the program will still continue to function properly.

Ans: The is_married condition is neither needed nor being used anywhere. This is only making the program a little more confusing. It is best to remove anything from the program that will never be used. The program should still be able to run properly.

Q-2: A student defined a function, as shown below:

```
def kms_to_miles(distance):

        distance * 0.621
```

When trying to use it, the program returned a value of ‘None’ as a result. Why do you think that happened?

A. The student must not have passed the appropriate parameter.
B. The distance used would have been in miles, hence the error.
C. The student forgot to use ‘return’ before the calculation while defining the function.
D. I have no idea why this failed. It should have worked.

Q-3: Do you think the following program should work? If not, why?

```
prices = [5, 10, 15, 20, 25]

total = 0

for item in prices:

        total += item

print(f"Your total price is: ${total}")
```

Ans: The program is actually error-free. This program should run fine on its own.

Q-4: In our previous book, we went through an example program, as shown:

```
for a in range(3):

   for b in range(3):

      for c in range(3):

         print(f"({a}, {b}, {c})")
```

If you were to change the values of the ranges from top to bottom to 3, 2, 1, respectively, would the program work? What will the outcome be?

Ans: The program should continue to work, and the outcome will be the following:

```
(0, 0, 0)

(0, 1, 0)

(1, 0, 0)

(1, 1, 0)

(2, 0, 0)

(2, 1, 0)
```

Q-5: According to a student, the indentation is unnecessary and should not cause any problems when executing a program. The other student

is of the idea that Python pays attention to whitespace, and hence the indentation is quite important to maintain the code and arrange it accordingly. Which of the two do you think is right?

Ans: The latter has the correct answer as Python pays significant attention to whitespaces. This is exactly why Python will fail to function if you forget to use an indentation where it is necessary.

Q-6: Look at the code snippet below. It was taken from a program that was designed to iterate over key pairs of a dictionary.

```
output = ""

for char in number:

    output += words.get(char) + " "

print(output)
```

What does the += operator do here?

Ans: Although the program written above will fail to work, owing to multiple issues, the actual question is highlighting what the += operator would do. This operator adds as an increment the value of words.get(char) to the output on every loop.

Q-7: A teenager wanted to print out a simple design on python as a result. The design shown below was the output:

```
*

**

***

****

*****

******

*******
```

```
********

*********

**********

I did this!

**********

*********

********

*******

******

*****

****

***

**

*
```

Do you think this can be done using loops? If so, can you code the program?

Ans: There are two ways you can carry out this program. The first one involves a lot of lines, each using a print statement. That is far too basic, and we will simply ignore that. The second one involves the use of loops, and that is where things get interesting. Here is how you can achieve this:

```
character = '*'

for char in range(0, 11):

    output = character * char

    print(output)
print("I did this!")
for char in range(10, 0, -1):
    output = character * char
    print(output)
```

You have first informed Python to iterate through the range in ascending order. In the second loop, you have used a -1 step in the end and reversed the order of the range. Now, it will start from 10 and end at the last number of the range.

Chapter 4 Solutions

Q-1: Taxi Fare Calculator

Ans: This was a little tough. There were quite a few components that needed to be readdressed a few times before arriving at the final phase. Here is my defined function to carry out the taxi fare calculations.

```
def calculate_fare():
    distance = float(input("Enter the distance in kms: "))
    distance = distance * 1000  #Convert kms into meters
    fare = 3.0 + ((distance / 100) * 0.1) # extra fare charged at every 100 meters
    return fare
print(f"Your total taxi fare is ${calculate_fare()}")
```

Remember, you will need to use a float value to get the correct answers.

Q-2: Card Deck Shuffler

Ans: This one was hard. I am sure you had to do a bit of research, and if you did, it is perfectly okay. I was not expecting you get this one right immediately. Using functions can sometimes be tricky, especially if it involves quite a few elements. Here is how this is done:

```
from random import randrange

def createDeck():

        cards = []

        for suit in ["s", "h", "d", "c"]:

         for value in ["2", "3", "4", "5", "6", "7", "8", "9", "T", "J", "Q", "K",
"A"]:

          cards.append(value + suit)

        return cards

def shuffle(cards):

        for i in range(0, len(cards)):

         other_pos = randrange(0, len(cards))

         temp = cards[i]

         cards[i] = cards[other_pos]

         cards[other_pos] = temp

def main():

        cards = createDeck()

        print("This is the original deck: ")

        print(cards)

        print()

        shuffle(cards)

        print("This is the shuffled version: ")

   print(cards)
```

```
main()
```

Now, the result should show you both the original cards and the shuffled version as well.

Q-3: Random Password Generator

Ans: The solution to this is as shown below. Most of it is pretty much self-explanatory.

```
from random import randint

shortest_pass = 6

max_pass = 8

min_ASCII = 33

max_ASCII = 126

def randomPass():

    randomLength = randint(shortest_pass, max_pass)

    result = ""

    for i in range(randomLength):

     randomChar = chr(randint(min_ASCII, max_ASCII))

     result = result + randomChar

     return result

def main():

    print("Your randomly generated password: ", randomPass())

main()
```

I encourage you to try and change the values per your liking to change the outcome and see how the overall program is affected.

Is this correct? – Part 4 solutions

Q-1: Below is a user-made function that is designed to iterate through a given range and look for the highest number. Will the function work when it is called?

```
def high_number(numbers):
        max = numbers[0]
        for number in numbers:
         if number < max:
          max = number
        return max
list = [21, 200, 31, 1, 39]
high_number(list)
```

Ans: The above is a code you have already visited in this workbook. Currently, it will not print out anything as there is no print command used. To print the result, use the print function, and place the function with the parameter within the parentheses of the print function. Before executing the code, be sure to change the < operator to > to view the highest number on the list.

Q-2: What seems to be the issue with the following?

```
def this_function():
   print("Hello From This Function!")
this_function_with_args(name, greeting):
   print(f"Hello {name}, From This Function!, I wish you {greeting}")
this_function()
this_function_with_args()
```

Ans: In the above code, the function “this_function_with_args()” is not defined as the key element of ‘def’ is missing. Without it, Python will not be able to find anything called this_function_with_args(), and hence the program will fail to execute.

Once that is sorted, you will need to pass two arguments for this function to work properly. Here is an example:

```
this_function_with_args("John", "Happy Birthday")
```

Now, the function will work as per the requirements.

Q-3: What would this function do?

```
def plus(a,b):

  sum = a + b

(sum, a)

sum, a = plus(3,4)

print(sum)
```

Ans: This function will not work as it is not defined properly and is honestly a mess. Remember, we aim to write clean code so that everyone can understand what is going on. Here is the solution to make this function work properly:

```
def plus(a, b):

        return a + b

sum = plus(3, 4)

print(sum)
```

Now, this function will calculate the two numbers that are passed as arguments.

Q-4: Can you place a loop within a function, as shown below?

```
def plus(*args):

  total = 0
  for i in args:
        total += i
  return total
print(plus(20,30,40,50))
```

Ans: Yes, the function should work perfectly fine. When defining a function, you can write hundreds of lines using ‘if’ and ‘else’ conditions as well as ‘for’ and ‘while’ loops. You can also use constructors, other predefined functions, methods, and values to make the function more meaningful. There is no limitation to how much code a function can hold.

Where to Head Next

The world is now your digital playground. You have all the essentials you need to get started. You are already fluent with your basics and should have no problem tackling the bigger challenges that lay ahead of you.

To make the most of the journey, utilize great resources such as Udemy, Coursera, YouTube, and many other major names that offer more advanced learning opportunities. There are countless books out there that offer the same learning experience with great detail. Be sure not to miss out on programming practice as the more you wait, the more you lose track of the concepts you worked so hard to learn.

Continue making great and simple programs to keep yourself in touch with the basics. Remember, the stronger the basics, the easier it is for you to advance.

Work on various projects which you can find on many websites and freelance platforms. For those willing to seek a career, consider the option of working in Artificial Intelligence, Deep Learning, Machine Learning, and other related fields. These are advancing every day and require more talented programmers, such as yourself, to go out there and make a difference.

Conclusion

We were excited when we began this workbook. Then came some arduously long tasks which quickly turned into irritating little chores that nagged us as programmers and made us think more than we normally would. There were times where some of us even felt like dropping the whole idea of being a programmer in the first place. But, every one of us who made it to this page, made it through with success.

Speaking of success, always know that your true success is never measured properly nor realized until you have hit a few failures along the road. It is a natural way of learning things. Every programmer, expert, or beginner, is bound to make mistakes. The difference between a good programmer and a bad one is that the former would learn and develop the skills while the latter would just resort to Google and locate an answer.

If you have chosen to be a successful Python programmer, know that there will be some extremely trying times ahead. The life of a programmer is rarely socially active, either unless your friend circle is made up of programmers only. You will struggle to manage your time at the start, but once you get the hang of things, you will start to perform exceptionally well. Everything will then start aligning, and you will begin to lead a more relaxed lifestyle as a programmer and as a human being.

Until that time comes, keep your spirits high and always be ready to encounter failures and mistakes. There is nothing to be ashamed of when going through such things. Instead, look back at your mistakes and learn from them to ensure they are not repeated in the future. You might be able to make programs even better or update the ones which are already functioning well enough.

Lastly, let me say it has been a pleasure to guide you through both these books and to be able to see you convert from a person who had no idea about Python to a programmer who now can code, understand and execute matters at will. Congratulations are in order. Here are digital cheers for you!

print(“Bravo, my friend!”)

I wish you the best of luck for your future and hope that one day, you will look back on this book and this experience as a life-changing event that led to a superior success for you as a professional programmer. Do keep an eye out for updates and ensure you visit the forums and other Python communities to gain the finest learning experience and knowledge to serve you even better when stepping into the more advanced parts of Python.

References

ggs, J, R. (2013): Python For Kids. San Francisco, CA: No Starch Press

tthes, E. (2016): Python Crash Course. San Francisco, CA: No Starch Press

yne, B. (2015): Teach Your Kids to Code. No Starch Press

ogramiz. (n.d.). Python Program to Check Leap Year. Retrieved December 10, 2019, from https://www.programiz.com/python-programming/examples/leap-year.

phenson, B. (2014): The Python Workbook. Springer International Publishing