#_ Important [Python Built-in Methods] {CheatSheet}

1. String Methods

- **strip()**: Removes whitespaces from the start and end. `s = ' hello '.strip()`
- **split()**: Splits a string into a list. `words = 'one,two,three'.split(',')`
- **join()**: Joins elements of an iterable. `s = '-'.join(['one', 'two', 'three'])`
- **replace()**: Replaces substrings. `s = 'hello'.replace('l', 'r')`
- **upper()**: Converts to uppercase. `s = 'hello'.upper()`
- **lower()**: Converts to lowercase. `s = 'Hello'.lower()`
- **startswith()**: Checks prefix. `b = s.startswith('He')`
- **endswith()**: Checks suffix. `b = s.endswith('lo')`
- **find()**: Finds substring index. `idx = s.find('l')`
- **isdigit()**: Checks if all characters are digits. `b = '123'.isdigit()`
- **isalpha()**: Checks if all characters are alphabetic. `b = 'abc'.isalpha()`

2. List Methods

- **append()**: Adds an element. `lst.append('new')`
- **extend()**: Appends iterable elements. `lst.extend([4, 5])`
- **insert()**: Inserts at index. `lst.insert(1, 'inserted')`
- **remove()**: Removes first occurrence. `lst.remove('item')`
- **pop()**: Removes and returns an element. `item = lst.pop()`
- **index()**: Returns first index of value. `idx = lst.index('item')`
- **count()**: Counts occurrences. `cnt = lst.count('item')`
- **sort()**: Sorts the list. `lst.sort()`
- **reverse()**: Reverses the list. `lst.reverse()`

3. Dictionary Methods

- **get()**: Retrieves value for key. `value = dct.get('key')`
- **keys()**: Returns dictionary keys. `keys = dct.keys()`
- **values()**: Returns dictionary values. `values = dct.values()`
- **items()**: Returns key-value pairs. `items = dct.items()`
- **update()**: Updates dictionary. `dct.update({'new_key': 'new_value'})`
- **pop()**: Removes key and returns value. `value = dct.pop('key')`
- **popitem()**: Removes last inserted key-value pair. `item = dct.popitem()`

- **clear()**: Clears the dictionary. `dct.clear()`

4. Set Methods

- **add()**: Adds an element. `st.add('item')`
- **remove()**: Removes an element. `st.remove('item')`
- **discard()**: Removes an element if present. `st.discard('item')`
- **pop()**: Removes and returns an element. `item = st.pop()`
- **clear()**: Removes all elements. `st.clear()`
- **union()**: Returns the union of sets. `union_set = st1.union(st2)`
- **intersection()**: Returns the intersection. `intersect_set = st1.intersection(st2)`
- **difference()**: Returns the difference. `difference_set = st1.difference(st2)`

5. File I/O Methods

- **open()**: Opens a file. `file = open('file.txt', 'r')`
- **read()**: Reads the entire file. `content = file.read()`
- **readline()**: Reads one line. `line = file.readline()`
- **readlines()**: Reads lines into a list. `lines = file.readlines()`
- **write()**: Writes a string. `file.write('hello')`
- **writelines()**: Writes a list of strings. `file.writelines(['hello\n', 'world'])`
- **close()**: Closes the file. `file.close()`

6. General Purpose

- **len()**: Returns the length. `length = len(iterable)`
- **range()**: Generates a range of numbers. `for i in range(10):`
- **print()**: Prints to the console. `print('Hello, world!')`
- **type()**: Returns the type. `t = type(obj)`
- **id()**: Returns the unique identifier. `identifier = id(obj)`
- **sorted()**: Returns sorted list. `sorted_lst = sorted(iterable)`
- **enumerate()**: Adds counter to an iterable. `for index, value in enumerate(lst):`
- **zip()**: Aggregates elements from iterables. `for a, b in zip(lst1, lst2):`

7. Conversion Functions

- **int()**: Converts to an integer. `i = int('123')`
- **float()**: Converts to a float. `f = float('123.45')`
- **str()**: Converts to a string. `s = str(123)`
- **list()**: Converts to a list. `lst = list('abc')`
- **dict()**: Converts to a dictionary. `dct = dict([(1, 'one'), (2, 'two')])`
- **set()**: Converts to a set. `st = set([1, 2, 3])`
- **tuple()**: Converts to a tuple. `t = tuple([1, 2, 3])`

8. Mathematical Functions

- **abs()**: Returns the absolute value. `absolute = abs(-5)`
- **sum()**: Sums the items. `total = sum([1, 2, 3])`
- **min()**: Returns the minimum. `minimum = min([1, 2, 3])`
- **max()**: Returns the maximum. `maximum = max([1, 2, 3])`
- **pow()**: Raises a number to a power. `result = pow(2, 3)`
- **round()**: Rounds a number. `rounded = round(3.14)`

9. Functional Programming Tools

- **filter()**: Filters elements. `evens = filter(lambda x: x % 2 == 0, lst)`
- **map()**: Applies a function. `squares = map(lambda x: x**2, lst)`
- **reduce()**: Reduces to a single value. `from functools import reduce; total = reduce(lambda a, b: a + b, lst)`

10. Input and Output

- **input()**: Reads a string from input. `s = input('Enter something: ')`
- **format()**: Formats a string. `formatted = format(123.4567, '.2f')`

11. Class and Object Related

- **getattr()**: Gets an attribute value. `attr = getattr(obj, 'attr_name')`
- **setattr()**: Sets an attribute value. `setattr(obj, 'attr_name', value)`
- **hasattr()**: Checks if attribute exists. `has_attr = hasattr(obj, 'attr_name')`
- **delattr()**: Deletes an attribute. `delattr(obj, 'attr_name')`
- **isinstance()**: Checks instance type. `is_instance = isinstance(obj, Class)`
- **issubclass()**: Checks subclass. `is_subclass = issubclass(Derived, Base)`

12. Miscellaneous

- **globals()**: Returns the global symbol table. `global_symbols = globals()`
- **locals()**: Returns the local symbol table. `local_symbols = locals()`
- **callable()**: Checks if an object is callable. `is_callable = callable(obj)`
- **eval()**: Evaluates a Python expression. `result = eval('1 + 2')`
- **exec()**: Executes Python code dynamically. `exec('print(42)')`

13. Exception Handling

- **try/except/finally**: Handles exceptions. `try: risky_operation() except Exception as e: handle_exception() finally: cleanup()`

14. Memory and Object Management

- **del()**: Deletes an object. `del obj`
- **gc.collect()**: Forces garbage collection. `import gc; gc.collect()`

15. Working with Iterables

- **next()**: Retrieves the next item from an iterator. `item = next(iterator)`
- **iter()**: Returns an iterator. `iterator = iter(iterable)`

16. Decorators and Metaprogramming

- **staticmethod()**: Converts a method to a static method. `@staticmethod def func():`
- **classmethod()**: Converts a method to a class method. `@classmethod def func(cls):`

17. Context Managers

- **with/as**: Manages resource with context. `with open('file.txt') as file:`

18. Comprehensions

- **List Comprehension**: Creates a new list. `[x * x for x in range(10)]`
- **Dict Comprehension**: Creates a new dictionary. `{k: v for k, v in zip(keys, values)}`
- **Set Comprehension**: Creates a new set. `{x for x in iterable if x > 0}`
- **Generator Expression**: Generates items on-the-fly. `(x*xforxin range(10))`

19. Advanced Python Features

- **lambda**: Creates an anonymous function. `f = lambda x: x * x`
- **globals()**: Accesses global variables. `global_vars = globals()`
- **locals()**: Accesses local variables. `local_vars = locals()`
- **dir()**: Lists properties and methods. `properties = dir(obj)`

20. Serialization

- **pickle.dump()**: Serializes an object. `import pickle; pickle.dump(obj, file)`
- **pickle.load()**: Deserializes data. `obj = pickle.load(file)`

21. Python Runtime Services

- **exec()**: Executes Python code dynamically. `exec('print("Hello")')`
- **eval()**: Evaluates a Python expression. `result = eval('2 + 3')`

22. Python Attribute and Method Resolution

- **getattr()**: Retrieves an attribute's value. `value = getattr(obj, 'attribute')`
- **setattr()**: Sets an attribute's value. `setattr(obj, 'attribute', value)`
- **hasattr()**: Checks if an attribute exists. `exists = hasattr(obj, 'attribute')`
- **delattr()**: Deletes an attribute. `delattr(obj, 'attribute')`

23. Class and Instance Utilities

- **isinstance()**: Checks if an object is an instance of a class. `if isinstance(obj, MyClass):`
- **issubclass()**: Checks if a class is a subclass of another. `if issubclass(MyClass, ParentClass):`

24. Import and Module Management

- **import()**: Imports a module dynamically. `module = __import__('module_name')`

25. File and Directory Management

- **open()**: Opens a file. `file = open('file.txt', 'r')`
- **os.path.exists()**: Checks if a path exists. `import os; exists = os.path.exists('file.txt')`

26. Exception Handling and Debugging

- **try/except/finally**: Manages exceptions. `try: risky_operation() except Exception: handle_exception() finally: cleanup()`

27. Iterators and Generators

- **iter()**: Returns an iterator. `it = iter(collection)`
- **next()**: Retrieves the next item from an iterator. `item = next(it)`

28. Built-in Constants

- **True/False**: Boolean constants. `a = True; b = False`
- **None**: Represents the absence of a value. `value = None`

29. Command Line Arguments

- **sys.argv**: Retrieves command-line arguments. `import sys; args = sys.argv`

30. Memory Management and Optimization

- **gc.collect()**: Triggers garbage collection. `import gc; gc.collect()`
- **sys.getrefcount()**: Gets the reference count of an object. `import sys; ref_count = sys.getrefcount(obj)`