#_ the Python Ultimate [Cheat Sheet]

```
• 📚 Fundamentals
     variables: x = 5
     o Print: print("Hello, World!")
     o Comments:
          ■ Single-line: # Comment
          ■ Multi-line: '''Comment'''
 🔤 Data Types
    o Primitive:
          ■ String: "Hello"
          ■ Integer: 42
          ■ Float: 3.14
          ■ Boolean: True
     o Collections:
          ■ List: [1, 2, 3]
          ■ Tuple: (1, 2, 3)
          ■ Set: {1, 2, 3}
          ■ Dictionary: {"key": "value"}
• | Operators
    o Arithmetic: +, -, *, /, //, %, **
    o Comparison: ==, !=, <, >, <=, >=
     o Logical: and, or, not
     o Membership: in, not in
     o Identity: is, is not
• 🔀 Conditionals
    \circ If: if x > y:
    o Elif: elif x < y:</pre>
    ∘ Else: else:
• 🔄 Loops
     o For: for x in range(5):
     ○ While: while x < 5:
     o Break: break
     Continue: continue
```

```
• | Functions
    o Defining: def my_function():
    Calling: my_function()
    o Default parameters: def func(x, y=0):
    variable-length arguments: def func(*args, **kwargs):
 m Classes & Objects
    o Class definition: class MyClass:
    o Constructor: def __init__(self):
    o Instance methods: def method(self):
    o Class variables: class_var = 0
    o Object instantiation: my_object = MyClass()
    Inheritance: class DerivedClass(BaseClass):
    o Method overriding: def method(self):
 Error Handling
    ∘ Try: try:
    o Except: except Exception as e:
    Raise: raise ValueError("Error message")
    ∘ Finally: finally:
 Importing Libraries
    o Import: import numpy
    Alias: import numpy as np

    Specific import: from math import pi

• File I/O
    o Open: with open("file.txt", "r") as file:
    o Read: file.read()
    o Write: with open("file.txt", "w") as file:
    o Append: with open("file.txt", "a") as file:
 List Comprehensions
    ∘ Syntax: [expression for item in iterable if condition]
• 📝 Lambda Functions
    ∘ Syntax: lambda arguments: expression
• 🔄 Iterators & Generators
    o Iterator: iter(obj)
```

Generator function: def my_generator(): yield value

o Next item: next(iterator)

• Generator expression: (expression for item in iterable if condition)

• 🔀 Context Managers

- o Defining: class MyContext:
- o Enter method: def __enter__(self):
- Exit method: def __exit__(self, exc_type, exc_value, traceback):
- o Using: with MyContext() as my_context:

★ Built-in Functions

- o len(obj) → Length of object
- o sum(iterable[, start]) → Sum of elements
- o max(iterable[, key]) → Maximum element
- $\circ \ \, \min(\texttt{iterable}[\,,\,\, \texttt{key}]) \, \rightarrow \, \texttt{Minimum element}$
- o sorted(iterable[, key][, reverse]) → Sorted list
- o range(stop[, start][, step]) → Sequence of numbers
- zip(*iterables) → Iterator of tuples
- map(function, iterable) → Apply function to αll items
- o filter(function, iterable) → Filter elements by function
- o isinstance(obj, classinfo) → Check object's class

String Methods

- o lower() → Lowercase
- o upper() → Uppercase
- o strip([chars]) → Remove leading/trailing characters
- split([sep][, maxsplit]) → Split by separator
- o replace(old, new[, count]) → Replace substring
- o find(sub[, start][, end]) → Find substring index
- o format(*args, **kwargs) → Format string

List Methods

- o append(item) → Add item to end
- o extend(iterable) → Add elements of iterable
- o insert(index, item) → Insert item at index
- o remove(item) → Remove first occurrence
- o pop([index]) → Remove & return item
- o index(item[, start][, end]) → Find item index
- o count(item) → Count occurrences
- o sort([key][, reverse]) → Sort list

o reverse() → Reverse list

• 📋 Dictionary Methods

- o keys() → View list of keys
- o values() → View list of values
- o items() → View key-value pairs
- get(key[, default]) → Get vαlue for key
- o update([other]) → Update dictionary
- pop(key[, default]) → Remove & return value
- o clear() → Remove αll items

32 Set Methods

- o add(item) → Add item
- o update(iterable) → Add elements of iterable
- o discard(item) → Remove item if present
- o remove(item) → Remove item or raise KeyError
- o pop() → Remove & return item
- clear() → Remove all items
- o union(*others) → Union of sets
- o intersection(*others) → Intersection of sets
- o difference(*others) → Difference of sets
- o issubset(other) → Check if subset
- o issuperset(other) → Check if superset

Regular Expressions

- ∘ import re
- o re.search(pattern, string)
- o re.match(pattern, string)
- o re.findall(pattern, string)
- re.sub(pattern, repl, string)
- Common patterns: \d, \w, \s, ., ^, \$, *, +, ?, {n}, {n,}, $\{, m\}, \{n, m\}$

Decorators

- o Defining: def my_decorator(func):
- o Applying: @my_decorator

Modules & Packages

- o Creating a module: Save as .py file
- ∘ Importing a module: import my_module
- Creating a package: Create directory with __init__.py

Importing from a package: from my_package import my_module

Virtual Environments

- o Creating: python -m venv myenv
- Activating:
 - Windows: myenv\Scripts\activate
 - Unix/Mac: source myenv/bin/activate
- ∘ Deactivating: deactivate

Package Management (pip)

- ∘ Install: pip install package_name
- ∘ Uninstall: pip uninstall package_name
- Upgrade: pip install --upgrade package_name
- List installed packages: pip list
- Show package details: pip show package_name

• Time

- import datetime
- Current date & time: datetime.datetime.now()
- ∘ Date object: datetime.date(year, month, day)
- Time object: datetime.time(hour, minute, second, microsecond)
- o Format: datetime.datetime.strftime(format)
- Parse: datetime.datetime.strptime(date_string, format)
- Common format codes: %Y, %m, %d, %H, %M, %S

JSON

- o import json
- JSON to Python: json.loads(json_string)
- Python to JSON: json.dumps(obj)
- Read from file: json.load(file)
- Write to file: json.dump(obj, file)

Threading

- o import threading
- o Create a thread: t = threading. Thread(target=function, args=(arg1, arg2))
- Start a thread: t.start()
- Wait for thread to finish: t.join()

Multiprocessing

import multiprocessing

- Create a process: p = multiprocessing.Process(target=function, args=(arg1, arg2))
- o Start a process: p.start()
- Wait for process to finish: p.join()

Working with Databases (SQLite)

- o import sqlite3
- o Connect to a database: conn = sqlite3.connect('mydb.sqlite')
- o Cursor object: cursor = conn.cursor()
- Execute SQL commands: cursor.execute("CREATE TABLE my_table (id INTEGER, name TEXT)")
- Commit changes: conn.commit()
- o Fetch results: cursor.fetchall()
- o Close the connection: conn.close()

- o from bs4 import BeautifulSoup
- o Create a BeautifulSoup object: soup = BeautifulSoup(html_content, 'html.parser')
- o Find elements by tag: soup.find_all('tag_name')
- Access element attributes: element['attribute_name']
- o Get element text: element.text

- o import requests
- o GET request: response = requests.get(url)
- o POST request: response = requests.post(url, data=payload)
- Response content: response.content
- o JSON response: response.json()
- ∘ Response status code: response.status_code

- from flask import Flask, render_template, request, redirect, url_for
- o Create α Flask αpp: app = Flask(__name__)
- Define α route: @app.route('/path', methods=['GET', 'POST'])
- Run the αpp: app.run(debug=True)
- ∘ Return a response: return "Hello, World!"
- Render α template: return render_template('template.html', variable=value)

- Access request data: request.form['input_name']
- Redirect to another route: return redirect(url_for('route_function'))

🔬 Data Science Libraries

- NumPy: import numpy as np
- o pandas: import pandas as pd
- Matplotlib: import matplotlib.pyplot as plt
- o seaborn: import seaborn as sns
- o scikit-learn: import sklearn
- o TensorFlow: import tensorflow as tf
- o Keras: from tensorflow import keras
- PyTorch: import torch

• Q Command Line Arguments (argparse)

- import argparse
- ∘ Create an ArgumentParser: parser = argparse.ArgumentParser(description='Description of your program')
- Add arguments: parser.add_argument('--arg_name', type=str, help='Description of the argument')
- ∘ Parse arguments: args = parser.parse_args()
- ∘ Access argument values: args.arg_name

- o import logging
- Basic configuration: logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
- Logging levels: logging.debug(), logging.info(), logging.warning(), logging.error(), logging.critical()

Environment Variables

- o import os
- Get an environment variable: os.environ.get('VAR_NAME')
- Set an environment variable: os.environ['VAR_NAME'] = 'value'

• Type Hints

- o from typing import List, Dict, Tuple, Optional, Union, Any
- Function type hints: def my_function(param: int, optional_param: Optional[str] = None) -> List[int]:
- variable type hints: my_variable: Dict[str, int] = {}