Python Ultimate Guide

# Fundamentals

┣  Variables:x = 5

┣  Print: print("Hello, World!")

┣ Comments:

┣ Single-line: # Comment

┗ Multi-line: '''Comment'''  **Data Types**

┣ Primitive:

┣ String: "Hello"

┣ Integer: 42

┣ Float: 3.14 ┗ Boolean: True

┣ Collections:

┣ List: [1, 2, 3]

┣ Tuple: (1, 2, 3)

┣ Set: {1, 2, 3}

┗ Dictionary: {"key": "value"}

# Operators

┣  Arithmetic: +, -, \*, /, //, %, \*\*

┣  Comparison: ==, !=, <, >, <=, >=

┣  Logical: and, or, not

┣  Membership: in, not in

┗  Identity: is, is not

# Conditionals

┣  If: if x > y:

┣  Elif: elif x < y:

┗ Else: else:  **Loops**

┣  For: for x in range(5):

┣  While: while x < 5:

┣ Break: break

┗  Continue: continue

**Functions**

┣  Defining: def my\_function():

┣  Calling: my\_function()

┣  Default parameters: def func(x, y=0):

┗  Variable-length arguments: def func(\*args, \*\*kwargs):

**Classes & Objects**

┣  Class definition: class MyClass:

┣  Constructor: def \_\_init\_\_(self):

┣  Instance methods: def method(self):

┣  Class variables: class\_var = 0

┣  Object instantiation: my\_object = MyClass()

┣  Inheritance: class DerivedClass(BaseClass):

┗  Method overriding: def method(self):

**Error Handling**

┣ Try: try:

┣  Except: except Exception as e:

┣  Raise: raise ValueError("Error message")

┗  Finally: finally:

# Importing Libraries

Import: import numpy

Alias: import numpy as np

┗ Specific import: from math import pi

**File I/O**

┣  Open: with open("file.txt", "r") as file:

┣  Read: file.read()

┣  Write: with open("file.txt", "w") as file:

┣  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

┣  Iterator: iter(obj)

┣  Next item: next(iterator)

┣  Generator function: def my\_generator(): yield value

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

**Context Managers**

┣  Defining: class MyContext:

┣  Enter method: def \_\_enter\_\_(self):

┣  Exit method: def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

┗  Using: with MyContext() as my\_context:

# Built-in Functions

┣  len(obj) → Length of object

┣  sum(iterable[, start]) → Sum of elements

┣  max(iterable[, key]) → Maximum element

┣  min(iterable[, key]) → Minimum element

┣  sorted(iterable[, key][, reverse]) → Sorted list

┣  range(stop[, start][, step]) → Sequence of numbers

┣  zip(\*iterables) → Iterator of tuples

┣  map(function, iterable) → Apply function to all items

┣  filter(function, iterable) → Filter elements by function

┗  isinstance(obj, classinfo) → Check object's class

# String Methods

┣  lower() → Lowercase

┣  upper() → Uppercase

┣ 🪞 strip([chars]) → Remove leading/trailing characters

┣  split([sep][, maxsplit]) → Split by separator

┣ ↔️ replace(old, new[, count]) → Replace substring

┣  find(sub[, start][, end]) → Find substring index

┗  format(\*args, \*\*kwargs) → Format string

**List Methods**

┣  append(item) → Add item to end

┣  extend(iterable) → Add elements of iterable

┣  insert(index, item) → Insert item at index

┣  remove(item) → Remove first occurrence

┣  pop([index]) → Remove & return item

┣  index(item[, start][, end]) → Find item index

┣  count(item) → Count occurrences

┣  sort([key][, reverse]) → Sort list

┗  reverse() → Reverse list

# Dictionary Methods

keys() → View list of keys values() → View list of values items() → View key-value pairs

┣  get(key[, default]) → Get value for key

┣  update([other]) → Update dictionary

┣  pop(key[, default]) → Remove & return value

┗  clear() → Remove all items

**Set Methods**

┣  add(item) → Add item

┣  update(iterable) → Add elements of iterable

┣  discard(item) → Remove item if present

┣  remove(item) → Remove item or raise KeyError

┣  pop() → Remove & return item

┣  clear() → Remove all items

┣  union(\*others) → Union of sets

┣  intersection(\*others) → Intersection of sets

┣  difference(\*others) → Difference of sets

┣  issubset(other) → Check if subset

┗  issuperset(other) → Check if superset

# Regular Expressions

┣ import re

┣  re.search(pattern, string)

┣  re.match(pattern, string)

┣  re.findall(pattern, string)

┣  re.sub(pattern, repl, string)

┗  Common patterns:

┣ \d: Digit

┣ \w: Word character

┣ \s: Whitespace

┣ .: Any character (except newline)

┣ ^: Start of string

┣ $: End of string

┣ \*: Zero or more repetitions

┣ +: One or more repetitions

┣ ?: Zero or one repetition

┣ {n}: Exactly n repetitions

┣ {n,}: At least n repetitions

┣ {,m}: At most m repetitions

┣ {n,m}: Between n and m repetitions (inclusive)

**Decorators**

┣  Defining: def my\_decorator(func):

┗  Applying: @my\_decorator

## Modules & Packages

┣  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

┣  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

# Date & Time

┣  import datetime

┣  Current date & time: datetime.datetime.now()

┣  Date object: datetime.date(year, month, day)

┣  Time object: datetime.time(hour, minute, second, microsecond)

┣  Format: datetime.datetime.strftime(format)

┣  Parse: datetime.datetime.strptime(date\_string, format)

┗  Common format codes: %Y, %m, %d, %H, %M, %S

# JSON

┣ 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

┣  import threading

┣  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))

┣  Start a process: p.start()

┗  Wait for process to finish: p.join()

# Working with Databases (SQLite)

┣  import sqlite3

┣  Connect to a database: conn = sqlite3.connect('mydb.sqlite')

┣  Cursor object: cursor = conn.cursor()

┣  Execute SQL commands: cursor.execute("CREATE TABLE my\_table (id INTEGER, name TEXT)")

┣  Commit changes: conn.commit()

┣  Fetch results: cursor.fetchall() ┗  Close the connection: conn.close()

**Web Scraping (BeautifulSoup)**

┣  from bs4 import BeautifulSoup

┣  Create a BeautifulSoup object: soup = BeautifulSoup(html\_content, 'html.parser')

┣  Find elements by tag: soup.find\_all('tag\_name')

┣  Access element attributes: element['attribute\_name']

┗  Get element text: element.text

# Web Requests (Requests)

import requests

GET request: response = requests.get(url)

POST request: response = requests.post(url, data=payload)

┣  Response content: response.content

┣  JSON response: response.json()

┗  Response status code: response.status\_code

# Web Development (Flask)

┣  from flask import Flask, render\_template, request, redirect, url\_for

┣  Create a Flask app: app = Flask(\_\_name\_\_)

┣  Define a route: @app.route('/path', methods=['GET', 'POST'])

┣  Run the app: app.run(debug=True)

┣  Return a response: return "Hello, World!"

┣  Render a 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

┣  pandas: import pandas as pd

┣  Matplotlib: import matplotlib.pyplot as plt

┣  seaborn: import seaborn as sns

┣  scikit-learn: import sklearn

┣  TensorFlow: import tensorflow as tf

┣  Keras: from tensorflow import keras

┣  PyTorch: import torch

# 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

Logging

┣  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**

┣ import os

┣  Get an environment variable: os.environ.get('VAR\_NAME')

┗  Set an environment variable: os.environ['VAR\_NAME'] = 'value'

**Type Hints**

┣  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] = {}