[Python Basics] [cheatsheet]

1. Basic Syntax and Operations

- Print Statements: `print("Hello, World!")`
- Comments: `# This is a single-line comment`
- Multi-line Comments: `'''This is a multi-line comment'''` or `"""This is also a multi-line comment"""`
- Variable Assignment: `x = 10`
- Data Types: `int`, `float`, `str`, `bool`
- Type Checking: `type(variable)`
- Type Conversion: `int("5")`, `str(20)`
- Arithmetic Operations: `+`, `-`, `*`, `/`, `//` (integer division), `%` (modulus), `**` (exponentiation)
- Increment/Decrement: `x += 1`, `x -= 1`
- String Concatenation: `'Hello' + ' ' + 'World'`
- String Multiplication: `'Python' * 3`
- Input from User: `input("Enter your name: ")`

2. Data Structures

- List Creation: `my_list = [1, 2, 3]`
- Append to List: `my_list.append(4)`
- Insert into List: `my_list.insert(1, 'a')`
- Remove from List: `my_list.remove('a')`
- List Slicing: `my_list[1:3]`
- Sort List: `sorted(my_list)`, `my_list.sort()`
- Dictionary Creation: `my_dict = {'key': 'value'}`
- Access Dictionary Items: `my_dict['key']`
- Add/Update Dictionary Item: `my_dict['new_key'] = 'new_value'`
- Remove Dictionary Item: `del my_dict['key']`
- Tuple Creation: `my_tuple = (1, 2, 3)`
- Set Creation: `my_set = {1, 2, 3}`
- Add to Set: `my_set.add(4)`
- Remove from Set: `my_set.remove(1)`

3. Control Flow

- If Statement: `if condition:`

By: Waleed Mousa

- If-Else Statement: `if condition: else:`
- If-Elif-Else Statement: `if condition: elif condition: else:`
- For Loop: `for item in iterable:`
- While Loop: `while condition:`
- Break: `break`
- Continue: `continue`
- Pass: `pass`
- List Comprehension: `[expression for item in list]`
- Dictionary Comprehension: `{key: value for item in list}`

4. Functions and Modules

- Defining a Function: `def my_function():`
- Function with Parameters: `def my_function(param1, param2):`
- Return Statement: `return x`
- Default Parameter Value: `def my_function(param1, param2=5):`
- Variable Length Arguments: `def my_function(*args):`, `def my_function(**kwargs):`
- Importing a Module: `import math`
- Importing Specific Functions: `from math import sqrt`
- Creating a Module: Save your functions in a `.py` file
- Using Pip: `pip install package_name`

5. Error Handling

- Try-Except: `try: except Exception:`
- Multiple Except Blocks: `try: except TypeError: except ValueError:`
- Finally Block: `try: except Exception: finally:`
- Raise an Error: `raise ValueError("A value error occurred")`

6. File Handling

- Open a File: `file = open('file.txt', 'r')`
- Read a File: `file.read()`
- Write to a File: `file = open('file.txt', 'w'); file.write("Hello, World!")`
- Append to a File: `file = open('file.txt', 'a'); file.write("Hello, again!")`
- Close a File: `file.close()`
- With Statement (Context Manager): `with open('file.txt', 'r') as file:`

By: Waleed Mousa

- Read Lines from a File: `file.readlines()`

7. Object-Oriented Programming

- Defining a Class: `class MyClass:`
- Creating an Object: `my_object = MyClass()`
- Constructor Method: `def __init__(self, param1):`
- Instance Methods: `def my_method(self):`
- Class Variables: `class MyClass: variable = "value"`
- Inheritance: `class DerivedClass(BaseClass):`
- Method Overriding: `def my_method(self): in derived class`
- Access Modifiers: `public`, `_protected`, `__private`

8. Advanced Topics

- Lambda Functions: `lambda arguments: expression`
- Map Function: `map(lambda x: x*2, [1, 2, 3, 4])`
- Filter Function: `filter(lambda x: x%2 == 0, [1, 2, 3, 4])`
- Reduce Function: `from functools import reduce; reduce(lambda x, y: x+y, [1, 2, 3, 4])`
- Generators: `def my_generator(): yield x`
- Decorators: `def my_decorator(func):`
- Context Managers: `class MyContextManager: def __enter__(): def __exit__():`
- Regular Expressions: `import re; re.search('pattern', 'string')`
- Multithreading: `from threading import Thread; my_thread = Thread(target=my_function)`
- Multiprocessing: `from multiprocessing import Process; my_process = Process(target=my_function)`

9. Libraries and Frameworks

- Using NumPy: `import numpy as np; np.array([1, 2, 3])`
- Using Pandas: `import pandas as pd; pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})`
- Using Matplotlib: `import matplotlib.pyplot as plt; plt.plot([1, 2, 3], [4, 5, 6])`
- Using SciPy: `from scipy.optimize import minimize`
- Using Scikit-learn: `from sklearn.linear_model import LinearRegression`

By: Waleed Mousa

10. Date and Time

- Current Date and Time: `from datetime import datetime; datetime.now()`
- Specific Date: `from datetime import date; date(2020, 1, 1)`
- Formatting Date and Time: `datetime.now().strftime('%Y-%m-%d %H:%M:%S')`
- Parsing Date and Time Strings: `datetime.strptime('2020-01-01', '%Y-%m-%d')`
- Timedelta Objects: `from datetime import timedelta; timedelta(days=1)`

11. File Paths and Directories

- List Files in Directory: `import os; os.listdir('path/to/directory')`
- Check if File Exists: `os.path.exists('file.txt')`
- Create a Directory: `os.makedirs('new_directory')`
- Change Current Working Directory: `os.chdir('path/to/directory')`
- Path Join: `os.path.join('directory', 'file.txt')`

12. Environment and System

- System Commands: `import os; os.system('command')`
- Environment Variables: `os.environ.get('VARIABLE_NAME')`
- Current Working Directory: `os.getcwd()`
- Executing Shell Commands: `import subprocess; subprocess.run(['ls', '-l'])`

13. Networking

- Simple HTTP Requests: `import requests; response = requests.get('http://example.com')`
- Parsing HTML: `from bs4 import BeautifulSoup; soup = BeautifulSoup(response.text, 'html.parser')`
- Socket Client: `import socket; s = socket.socket(); s.connect(('hostname', port))`

14. Debugging and Profiling

- Print Debugging: `print("Debug: ", variable)`
- Using pdb: `import pdb; pdb.set_trace()`
- Profiling Python Code: `import cProfile; cProfile.run('function()')`

15. Testing

By: Waleed Mousa

- Writing Unit Tests: `import unittest; class TestMyFunction(unittest.TestCase):`
- Running a Test Case: `if __name__ == '__main__': unittest.main()`
- Assertions in Tests: `self.assertEqual(function_to_test(input), expected_output)`

16. Virtual Environments

- Creating a Virtual Environment: `python -m venv myenv`
- Activating a Virtual Environment: `source myenv/bin/activate` (Linux/Mac), `myenv\Scripts\activate` (Windows)
- Deactivating a Virtual Environment: `deactivate`

17. Packaging and Distribution

- Creating a Package: Organize your code in a directory with an `__init__.py` file.
- Setup Script: Writing a `setup.py` for package distribution.
- Installing Your Package: `pip install .` in your package directory.
- Uploading to PyPI: `python setup.py sdist upload`

18. Advanced String Formatting

- Formatted String Literals (f-strings): `name = "World"; f"Hello, {name}!"`
- String Format Method: `"{0}, {1}".format('Hello', 'World')`
- Percent (%) Formatting: `"%s, %s" % ('Hello', 'World')`

19. Comprehensions Beyond Lists

- Set Comprehensions: `{x for x in 'hello world' if x not in 'aeiou'}`
- Dictionary Comprehensions: `{k: v for k, v in [('key1', 1), ('key2', 2)]}`

20. Itertools and More Functional Tools

- Itertools for Combining Data: `import itertools; itertools.chain([1, 2], [3, 4])`
- Functional Programming Tools: `map`, `filter`, `functools.reduce`

21. Lambda Functions

By: Waleed Mousa

- **Simple Lambda Function:** `square = lambda x: x**2; square(5)`
- **Lambda with Multiple Arguments:** `add = lambda x, y: x + y; add(2, 3)`
- **Lambda in Sorted:** `sorted([(1, 'b'), (2, 'a')], key=lambda x: x[1])`

22. Error and Exception Handling

- **Basic Try-Except:** `try: 1/0; except ZeroDivisionError: print("Cannot divide by zero")`
- **Try-Except-Else:** `try: result = x / y; except ZeroDivisionError: print("Error"); else: print("Result is", result)`
- **Try-Except-Finally:** `try: f = open('file.txt'); except IOError: print("Error opening file"); finally: f.close()`
- **Raising Exceptions:** `if x < 0: raise ValueError("x must be non-negative")`

23. Iterators and Generators

- **Creating an Iterator:** `iter_obj = iter([1, 2, 3]); next(iter_obj)`
- **Building Generators:** `def my_gen(): yield 'a'; yield 'b'; for letter in my_gen(): print(letter)`
- **Generator Expression:** `(x*x for x in range(3))`

24. Decorators

- **Simple Decorator:** `def my_decorator(func): def wrapper(): print("Something"); func(); return wrapper`
- **Using Decorators:** `@my_decorator; def say_hello(): print("Hello")`
- **Decorator with Arguments:** `def repeat(n): def decorator(func): def wrapper(*args, **kwargs): for _ in range(n): func(*args, **kwargs); return wrapper; return decorator`

25. Context Managers

- **Using with Statement:** `with open('file.txt', 'r') as f: print(f.read())`
- **Creating a Context Manager:** `from contextlib import contextmanager; @contextmanager; def my_context(): print('Enter'); yield 'example'; print('Exit')`

26. Modules and Packages

- **Importing a Module:** `import math; math.sqrt(4)`
- **Selective Import:** `from math import sqrt; sqrt(4)`

By: Waleed Mousa

- Importing with Alias: `import numpy as np; np.array([1, 2, 3])`
- Creating a Package: Create a directory with an `__init__.py` file and other module files.

27. Regular Expressions

- Matching Strings: `import re; re.match('p', 'python')`
- Search Within Strings: `re.search('n', 'python')`
- Replacing Strings: `re.sub('python', 'cython', 'I love python')`
- Compiling Patterns: `pattern = re.compile('python'); pattern.findall('I love python')`

28. Data Serialization

- JSON Serialization: `import json; json.dumps({'name': 'John', 'age': 30})`
- JSON Deserialization: `json.loads('{"name": "John", "age": 30}')`
- Pickle Serialization: `import pickle; pickle.dumps(obj)`
- Pickle Deserialization: `pickle.loads(pickled_data)`

29. Multithreading and Multiprocessing

- Creating Threads: `from threading import Thread; t = Thread(target=function_name); t.start()`
- Creating Processes: `from multiprocessing import Process; p = Process(target=function_name); p.start()`
- Joining Threads: `t.join()`
- Joining Processes: `p.join()`

30. Asynchronous Programming

- Basic Coroutine: `import asyncio; async def main(): await asyncio.sleep(1); print('done')`
- Running Async Code: `asyncio.run(main())`
- Async/Await with Functions: `async def fetch_data(): data = await some_async_operation(); return data`

By: Waleed Mousa