Python-Refresher-Notes

1. {} Format

```
name = input("Enter your name: ")
color = input("Enter your favorite colour: ")
hobby = input("Enter your favorite hobby: ")

print("Hello {}! Nice to meet you.".format(name))
print("")
print("I see that your favourite colour is {}, and your favourite hobby is
{}. It's great to learn more about you.".format(color, hobby))
```

Output

```
Enter your name: Jack
Enter your favorite colour: Teal
Enter your favorite hobby: Painting
```

```
Hello Jack! Nice to meet you.

I see that your favourite colour is Teal, and your favourite hobby is
Painting. It's great to learn more about you.
```

2. {} Format 2

```

pat_name = input("Enter the patient name: ")
illness = input("Illness: ")
admission_fee = float(input("Admission fee: "))
doctor_fee = float(input("Doctor fee: "))
nursing_fee = float(input("Nursing fee: "))
medicine_cost = float(input("Cost for medicines: "))

print("AUMA Hospital\n")
print("Patient Name: {}\n".format(pat_name))
print("Illness: {}\n".format(illness))
print("Admission fee: ${:.2f}\n".format(admission_fee))
print("Doctor fee: ${:.2f}\n".format(doctor_fee))
print("Nursing fee: ${:.2f}\n".format(nursing_fee))
print("Cost for medicines: ${:.2f}\n".format(medicine_cost))
print("Total amount: ${:.2f}\n".format(admission_fee + doctor_fee +
nursing_fee + medicine_cost))
```

- {:.2f} — Formatted Placeholder
  - This is used to format floating-point numbers to a specific number of decimal places.
  - :.2f means:
    - : → start of format spec
    - .2 → two digits after the decimal point
    - f → fixed-point notation (i.e., decimal number)

3. Lambda function

Before: function

```
def get_percentage(sub1, sub2, sub3):
     return ((sub1 + sub2 + sub3) / 300) * 100
```

How to change to lambda

- To turn it into lamba, do the following
  - Step 1:
    - Remove def
    - Remove brackets of parameters
    - remove return
    - We get
      - get_percentage sub1,sub2,sub3 : ((sub1,sub2,sub3)/300) * 100
  - Step 2:

- Add `=` and lambda after function name
- get_percentage = lambda sub1,sub2,sub3 : ((sub1 + sub2 + sub3) / 300) * 100

Lambda function

```
get_percentage = lambda sub1,sub2,sub3 : ((sub1 + sub2 + sub3) / 300) * 100
```

4. Text Files

Basic Overview of File Handling in Python

Opening a File

Python uses the built-in `open()` function:

```
file = open("filename.txt", "mode")
```

- `"filename.txt"`: Name of the file
- `"mode"`: Specifies the operation (read, write, etc.)

File Modes

Mode	Description
`'r'`	Read (default). Error if file doesn't exist.
`'w'`	Write. Creates file if not exists, overwrites if it does.
`'a'`	Append. Adds content to the end of the file.
`'x'`	Create. Error if file already exists.
`'b'`	Binary mode (e.g., `'rb'`, `'wb'`)
`'t'`	Text mode (default)
`'+'`	Read and write (e.g., `'r+'`, `'w+'`)

Common File Handling Functions

Once a file is opened, you can use:

Reading

```
file.read()          # Reads entire file
file.readline()      # Reads one line
file.readlines()     # Reads all lines into a list
```

Writing

```
file.write("text")  # Writes text to file
file.writelines([...])  # Writes list of strings
```

Other Operations

```
file.seek(offset)   # Moves cursor to specified position
file.tell()         # Returns current cursor position
file.close()        # Closes the file
```

Using `with` Statement (Best Practice)

```
with open("filename.txt", "r") as file:
    content = file.read()
```

Example: Writing and Reading a File

```
# Writing
with open("example.txt", "w") as f:
    f.write("Hello, world!")

# Reading
with open("example.txt", "r") as f:
    print(f.read())
```

5. CSV Files

What Does a CSV File Look Like?

```
Name,Age,City
Alice,30,New York
Bob,25,Los Angeles
Charlie,35,Chicago
```

Each line is a **record**, and each value is separated by a **comma**.

Working with CSV in Python

Python provides a built-in module called `csv` to handle CSV files.

Reading a CSV File

```python
import csv

with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Writing to a CSV File

```python
import csv

with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Name', 'Age', 'City'])
    writer.writerow(['Alice', 30, 'New York'])
```

Using `DictReader` and `DictWriter`

These allow you to work with CSV data as dictionaries:

Reading:

```
with open('data.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row['Name'], row['Age'])
```

Writing:

```
with open('output.csv', 'w', newline='') as file:
    fieldnames = ['Name', 'Age', 'City']
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'Name': 'Bob', 'Age': 25, 'City': 'LA'})
```

6. JSON files

What is JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format that's easy for humans to read and write, and easy for machines to parse and generate. It's commonly used for APIs, configuration files, and data storage.

🔹 Importing the Module

```
import json
```

Reading JSON

1. From a JSON string

```
json_string = '{"name": "Alice", "age": 30}'
data = json.loads(json_string)
print(data["name"])  # Output: Alice
```

2. From a JSON file

```
with open('data.json', 'r') as file:
    data = json.load(file)
```

🔹 **Writing JSON**

1. To a JSON string

```
person = {"name": "Bob", "age": 25}
json_string = json.dumps(person)
print(json_string)
```

2. To a JSON file

```
with open('output.json', 'w') as file:
    json.dump(person, file)
```

Formatting Options with `dumps()`

```
json.dumps(person, indent=4)       # Pretty print with indentation
json.dumps(person, sort_keys=True) # Sort keys alphabetically
```

Common Use Cases

- Parsing API responses
- Saving configuration settings
- Logging structured data
- Data exchange between systems

7. Classes in python - Example 1

Here, classes serve as blueprints to define the structure and behaviour of vehicles, while objects, instantiated from these classes, encapsulate specific vehicle data and operations, facilitating the management of vehicle information and premium calculations.

```
class Vehicle:
    def __init__(self, cost, vehicle_type):
        self.__cost = cost
        self.__type = vehicle_type
        self.__premium = 0

    def calculate_premium(self):
        if self.__type == 1:
            premium_amount = (self.__cost) * 0.02
            self.__premium = premium_amount
        elif self.__type == 2:
            premium_amount = (self.__cost) * 0.06
            self.__premium = premium_amount
        elif self.__type == 3:
            premium_amount = (self.__cost) * 0.08
            self.__premium = premium_amount

    def get_premium(self):
        return self.__premium


vehicle_cost = float(input("Enter the vehicle cost: "))
vehicle_type = int(input("Enter the type of the vehicle (1 for 2 wheeler, 2
for 4 wheeler, 3 for other types): "))

vehicle_obj = Vehicle(vehicle_cost, vehicle_type)
vehicle_obj.calculate_premium()

vehicle_premium_amount = vehicle_obj.get_premium()
print("The premium amount is: {}".format(vehicle_premium_amount))
```

8. Classes in Python - Example 2

```
class Addition:
    def __init__(self):
        self.__real = 0
        self.__img = 0

    def get_real(self):
        return self.__real
    def set_real(self, real):
        self.__real = real

    def get_img(self):
        return self.__img

    def set_img(self, img):
        self.__img = img

    def addRealPart(self, obj1, obj2):
        real_sum = obj1.get_real() + obj2.get_real()
        self.set_real(real_sum)

    def addImaginaryPart(self, obj1, obj2):
        img_sum = obj1.get_img() + obj2.get_img()
        self.set_img(img_sum)

    def display(self):
        print(f"Sum: {self.__real} + {self.__img}i")


# Example usage
obj1 = Addition()
obj1.set_real(3)
obj1.set_img(4)

obj2 = Addition()
obj2.set_real(5)
obj2.set_img(6)

result = Addition()
result.addRealPart(obj1, obj2)
result.addImaginaryPart(obj1, obj2)
result.display()
```

9. Sets in Python

A set is a built-in data type in Python that stores unique items. It's unordered, meaning the items don't have a fixed position.

```
my_set = {1, 2, 3, 4}
```

Methods of creating set

```python
# Using curly braces
fruits = {"apple", "banana", "cherry"}

# Using the set() function
numbers = set([1, 2, 3, 4])
```

No Duplicates

```python
colors = {"red", "blue", "red"}
print(colors)  # Output: {'red', 'blue'}
```

Adding item

```python
fruits.add("orange")
```

Remove Item

```python
fruits.remove("banana")  # Raises error if not found
fruits.discard("banana") # No error if not found
```

Check membership

```python
"apple" in fruits  # True
```

Set Union

```python
a = {1, 2}
b = {2, 3}
print(a | b)  # {1, 2, 3}
```

Set Intersection

```
print(a & b)  # {2}
```

Set Difference

```
print(a - b)  # {1}
```

Symmetric difference

```
print(a ^ b)  # {1, 3}
```

10. Dir() In Python

The dir() function in Python is used to list the names of all the attributes and methods of an object. It's super helpful for exploring what you can do with a variable, module, or class.

Example

```
x = [1, 2, 3]
print(dir(x))
```

```
['__add__', '__class__', ..., 'append', 'clear', 'copy', 'count', ...]
```

11. help()

The help() function in Python is a built-in tool that provides documentation about Python objects, modules, functions, classes, and more. It's like having a mini user manual right inside your Python environment.

12. Reference based assignment in python

In Python, assignment is reference-based for mutable objects like lists, dictionaries, sets, and custom objects. This means when you assign one variable to another, both variables point to the same object in memory.

Example With List

```python
a = [1, 2, 3]
b = a  # b references the same list as a

b.append(4)
print(a)  # Output: [1, 2, 3, 4]
```

- a and b are pointing to the same list
- Changing b also changes a

Example with Integer

```python
x = 10
y = x  # y gets a copy of x's value

y += 5
print(x)  # Output: 10
```

- x and y are independent
- Integers are immutable, so changes to y don’t affect x

13. OS Functions

1. Working with Directories

```python
import os

os.getcwd()          # Get current working directory
os.chdir('path')     # Change current directory
os.listdir()         # List files and folders in current directory
```

2. Creating and Removing Folders

```
os.mkdir('new_folder')       # Create a single folder
os.makedirs('a/b/c')         # Create nested folders
os.rmdir('folder')           # Remove a folder (must be empty)
os.removedirs('a/b/c')       # Remove nested folders
```

3. File Operations

```
os.remove('file.txt')        # Delete a file
os.rename('old.txt', 'new.txt')  # Rename a file or folder
```

4. Path Operations

```
os.path.exists('file.txt')       # Check if file/folder exists
os.path.isfile('file.txt')       # Check if it's a file
os.path.isdir('folder')          # Check if it's a directory
os.path.join('folder', 'file.txt')  # Join paths safely
```

5. Environment Variables

~~~
os.environ['HOME']               # Get environment variable
os.environ['NEW_VAR'] = 'value'  # Set environment variable

```python
os.system('echo Hello')  # Run a shell command
~~~

14. Function named arguments

```
def create_account(username, password, email=None, is_admin=False):
    print(f"Username: {username}")
    print(f"Password: {password}")
    print(f"Email: {email}")
    print(f"Admin Access: {is_admin}")

# Using named arguments
```

```
create_account(username="john_doe", password="secure123", is_admin=True)
```

15. Private properties in class

```
class BankAccount:
    def __init__(self, owner, balance):
        self.owner = owner
        self.__balance = balance  # Private property

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance

# Create an account
account = BankAccount("Alice", 1000)

# Accessing public property
print(account.owner)  # Alice

# Accessing private property directly (will raise AttributeError)
# print(account.__balance)  # ❌ Not allowed

# Accessing private property via method
print(account.get_balance())  # ✅ 1000
```

16. Deleting a class

```
class MyClass:
    def __init__(self):
        print("Object created")

obj = MyClass()
del obj  # Deletes the object
```

17. Map function

The map() function in Python is a built-in function used to apply a given function to each item in an iterable (like a list, tuple, etc.) and return a new iterable (specifically, a map object).

Example 1: Using map() with a built-in function

```python
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x**2, numbers)

print(list(squared))  # Output: [1, 4, 9, 16, 25]
```

Example 2: Using map() with a custom function

```python
def capitalize(word):
    return word.capitalize()

words = ['python', 'java', 'c++']
capitalized_words = map(capitalize, words)

print(list(capitalized_words))  # Output: ['Python', 'Java', 'C++']
```

Example 3: Multiple Iterables

```python
a = [1, 2, 3]
b = [4, 5, 6]

result = map(lambda x, y: x + y, a, b)
print(list(result))  # Output: [5, 7, 9]
```

18. filter function

Example 1: Filtering even numbers

```python
numbers = [1, 2, 3, 4, 5, 6]

even_numbers = filter(lambda x: x % 2 == 0, numbers)
```

```python
print(list(even_numbers))  # Output: [2, 4, 6]
```

Example 2: Filtering strings by length

```python
words = ["apple", "bat", "carrot", "dog"]

long_words = filter(lambda word: len(word) > 3, words)

print(list(long_words))  # Output: ['apple', 'carrot']
```

Example 3: Using a named function

```python
def is_positive(n):
    return n > 0

numbers = [-5, 3, 0, -2, 8]

positive_numbers = filter(is_positive, numbers)

print(list(positive_numbers))  # Output: [3, 8]
```

19.Reduce function

```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]
total = reduce(lambda x, y: x + y, numbers)
print(total)  # Output: 15
```

20. Map and Reduce Example

```python
from functools import reduce

numbers = [1, 2, 3, 4]
```

```python
squares = map(lambda x: x**2, numbers)
sum_of_squares = reduce(lambda x, y: x + y, squares)

print(sum_of_squares)  # Output: 30
```

21. Convert array to numpy

```python
import numpy as np

# Python list
my_list = [1, 2, 3, 4, 5]

# Convert to NumPy array
np_array = np.array(my_list)

print(np_array)
print(type(np_array))  # <class 'numpy.ndarray'>
```

22. Sort

```python
numbers = [5, 2, 9, 1]
numbers.sort()
print(numbers)  # Output: [1, 2, 5, 9]
```

23. re module

A regular expression (regex) is a sequence of characters that defines a search pattern. It's commonly used for:

- Searching text
- Validating input (like emails or phone numbers)
- Replacing substrings
- Splitting strings based on pattern

Example 1: re.search

```python
import re

text = "My phone number is 123-456-7890"
match = re.search(r"\d{3}-\d{3}-\d{4}", text)

if match:
    print("Phone number found:", match.group())
```

Example 2: re.findall()

```python
text = "Emails: alice@example.com, bob@mail.com"
emails = re.findall(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", text)
print(emails)
```

Common Regex Patterns

Pattern	Meaning
`.`	Any character except newline
`\d`	Digit (0–9)
`\w`	Word character (a-z, A-Z, 0-9, _)
`\s`	Whitespace
`*`	0 or more repetitions
`+`	1 or more repetitions
`?`	0 or 1 repetition
`{n}`	Exactly n repetitions
`[]`	Set of characters
`^`	Start of string
`$`	End of string

24. What are Decorators?

A **decorator** is a special function in Python that is used to **add extra features or behavior to another function** — without changing that function's actual code.

Decorator

```python
def my_decorator(func):
    def wrapper():
        print("Before the function runs.")
        func()
        print("After the function runs.")
    return wrapper
```

```python
@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

Output

```
Before the function runs.
Hello!
After the function runs.
```