**Day 46: Python**

**Python Notes**

**Python:** Python is a popular programming language. It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

**Applications:**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

**Why Python?**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written.
* The most recent major version of Python is Python 3.

**Note:** Python Syntax compared to other programming languages

* Python was designed for readability and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**ipython:** it is interactive python terminal where commands can be executed. once session is closed data will be lost.

**Install ipython:**

**apt install python3-pip**

**pip install ipython**

**Data Types:**

* integer
* float
* string
* List
* Dict

**Examples:**

1. 1 + 11 = 12
2. 1 + “11” -----🡪 throws an error
3. ‘1’ + ‘1’ = 11
4. 1.5 + 1.5 = 3.0

We can also find they type of values

1. Sum = 1

print(type(sum))

1. Sum = “Hello”

print(type(sum))

**Print Formatting:** It is the process of combining variables and strings together to display the formatted output.

1. dot format method

2. f method

1. **dot format method:** format method is used for string formatting. pair of {} brackets indicates a place holder which is a position where variable comes in position.

ex:

|  |
| --- |
| username = "lohit"  userid = 111  print("hello, My name is {}. My user id is {}.".format(username,userid)) |

O/P:

hello, My name is lohit. My user id is 111.

1. **f method:**

|  |
| --- |
| print(f"Hello. My name is {username}. My user id is {userid}") |

O/P:

hello, My name is lohit. My user id is 111

**Input Function:** input() is a function which is used to read the values from the user.

**ex:** username = input("Whats ur username\n")

**Python Script: to read the values from used and print**

|  |
| --- |
| username = input("whats the username?\n")  userid = input("whats the UID?\n")  print(f"Hello. My name is {username}. My user id is {userid}") |

Hello. My name is {username}. My user id is {userid}")

**string:** group of characters.

name = “Hello World”

here indexing allows the user to grab partial values from any data. Indexing starts from 0 to n-1

In above example name[0] points to **H**, name[1] points to **e** and so on

**Ex:**   
print(name) ---------- Hello World

print(name[0]) ------- H

print(name[4]) ------ o

print(name[0:4]) ------ display from 0th to 3rd character i.e, Hell

Python also supports reverse indexing. it will start display from last character. for last character index is -1.

print(name[-1]) --------d

print(name-3]) --------- r

print(len(name)) ---- print length of the string

**Day 47: Python**

**list:** it is a group of elements. It contain multiple values with different types or same type. list is created using a pair of square brackets. Using index, elements of list can be accessed.

**Note:** List can contain different data types like int, string, float, list.

We can add list to a list(With append function between the lists)

**List: heterogeneous**

**Array:homogeneous**  
Homogeneous means something is made up of similar or identical parts, while heterogeneous means something is made up of different parts

name = [] -------- empty list

name = [10, 20, 23, 25]

name = [“lohit”, 10, 15, 15.5]

name = [“lohit”, “vinay”, “raju”]

print(name[0]) ----- lohit

print(name[1]) --- vinay

print(name) ------ [“lohit”, “vinay”, “raju”]

name[2] = 10 --- in list, element can be replaced by another.

print(name) ----- [“lohit”, “vinay”, 10]

mylist=[1,2,”Phani”]

print(mylist) ----- [1, 2, 'Phani']

type(mylist) ------ list

print(mylist[0]) ------1

print(mylist[2]) ------ Phani

print(type(mylist[2])) ------- <class 'str'>

print(type(mylist[1])) -------- <class 'int'>

print(len(mylist)) ----- 3

mylist=[1, 10, 'Phani', 'NPR', 'NPR1']

print(len(mylist)) ----- 5

print(len(mylist[2])) ------ 5

print(len(mylist[3])) ------- 3

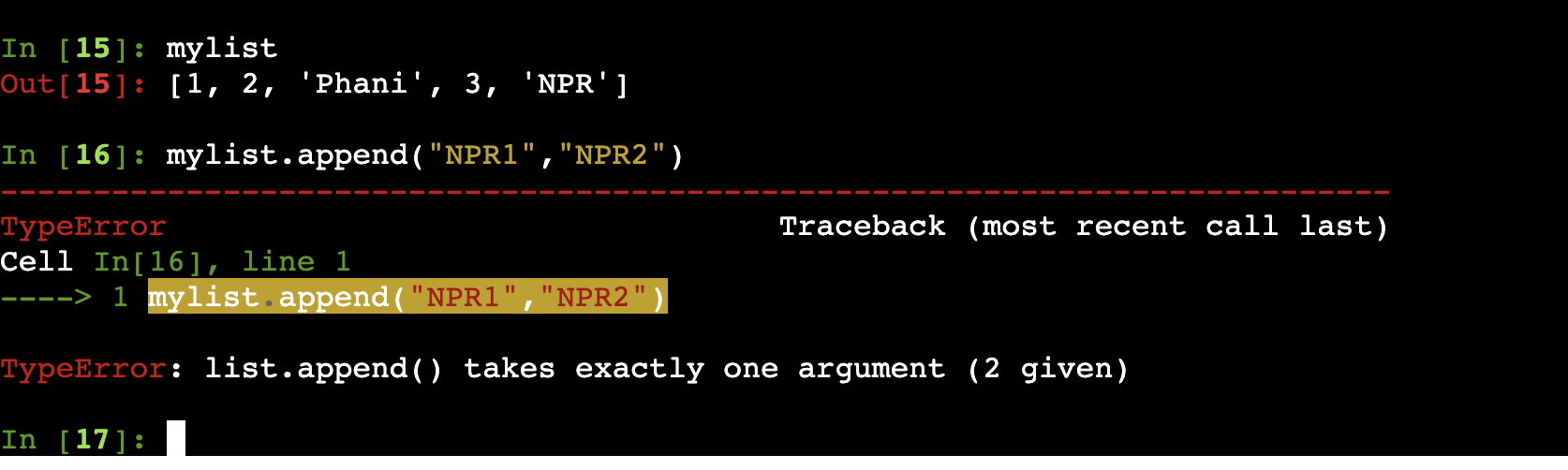
similarly, elements can be added and deleted from the list.

name.append(“welcome”)

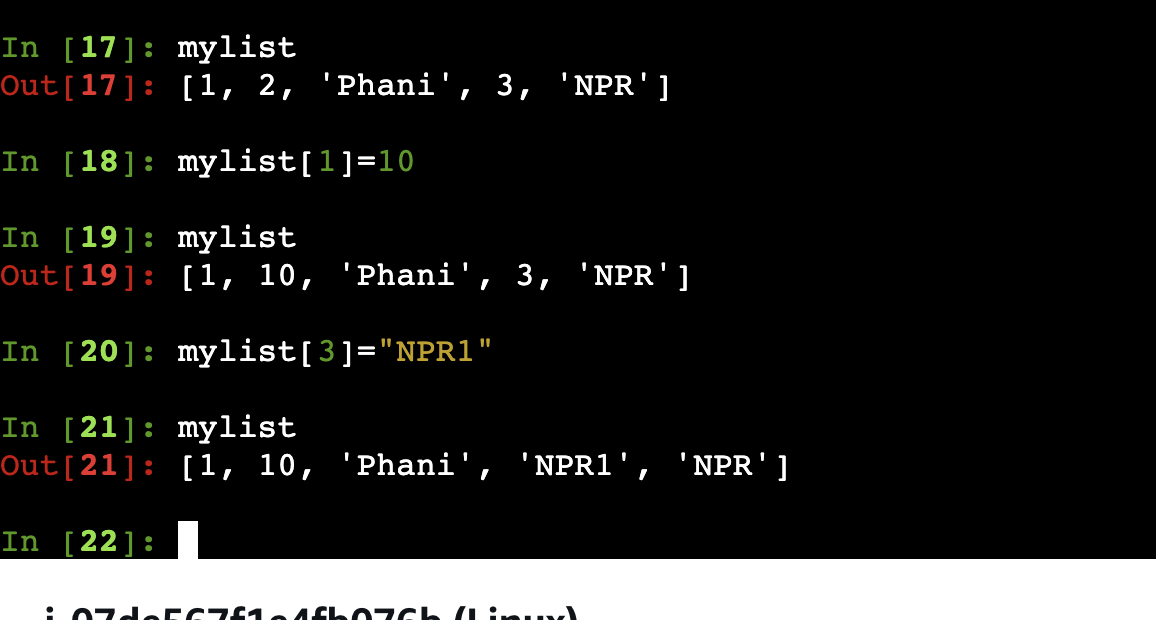
name.append(25)

print(name) ---- [“lohit”, “vinay”, 10, “welcome”, 25]

**Note: we cannot append multiple values at one time.**

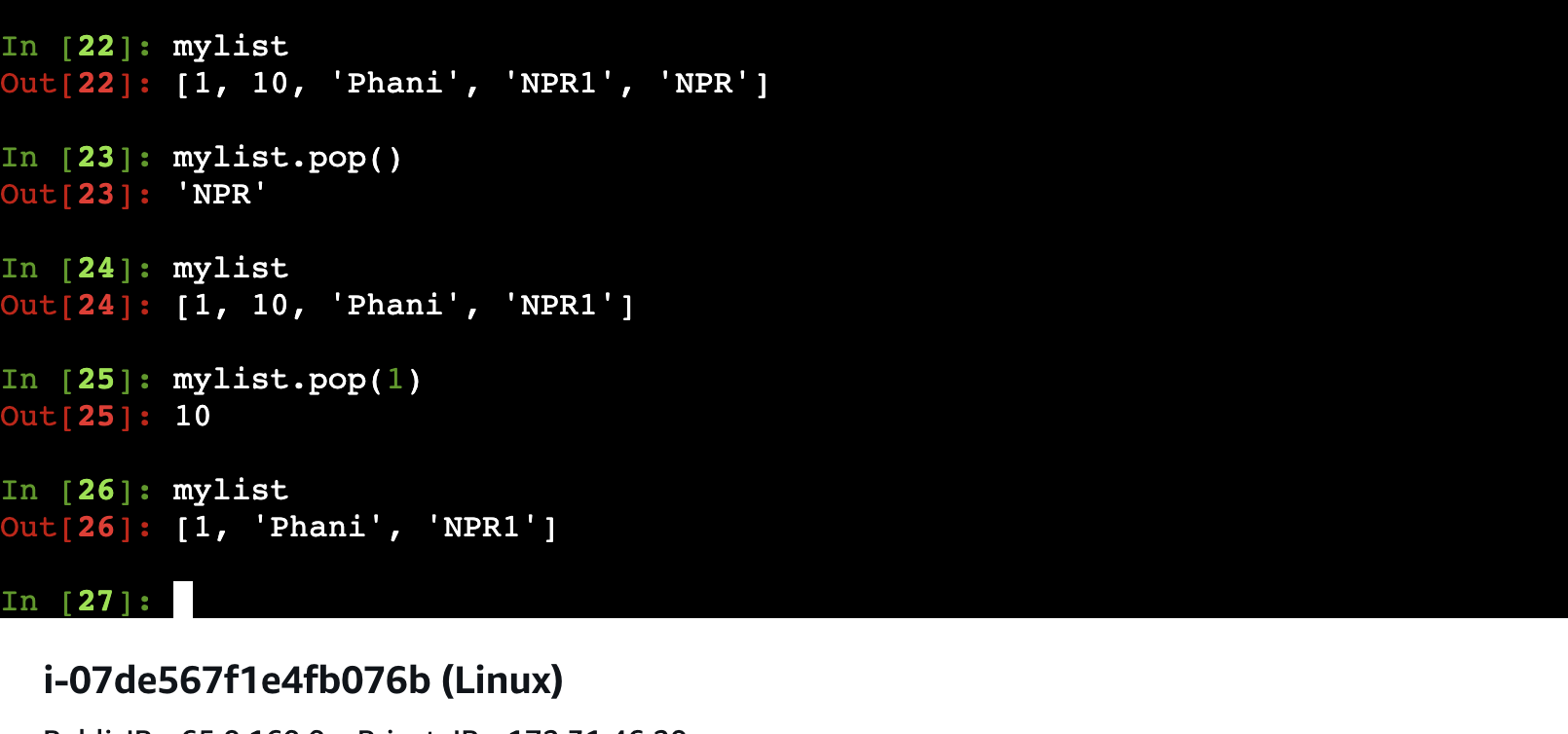


To replace:



name.pop() – remove the last element from the list

name.pop(3) --- removes the 4th element from the list since 3 points to 4the element in list.



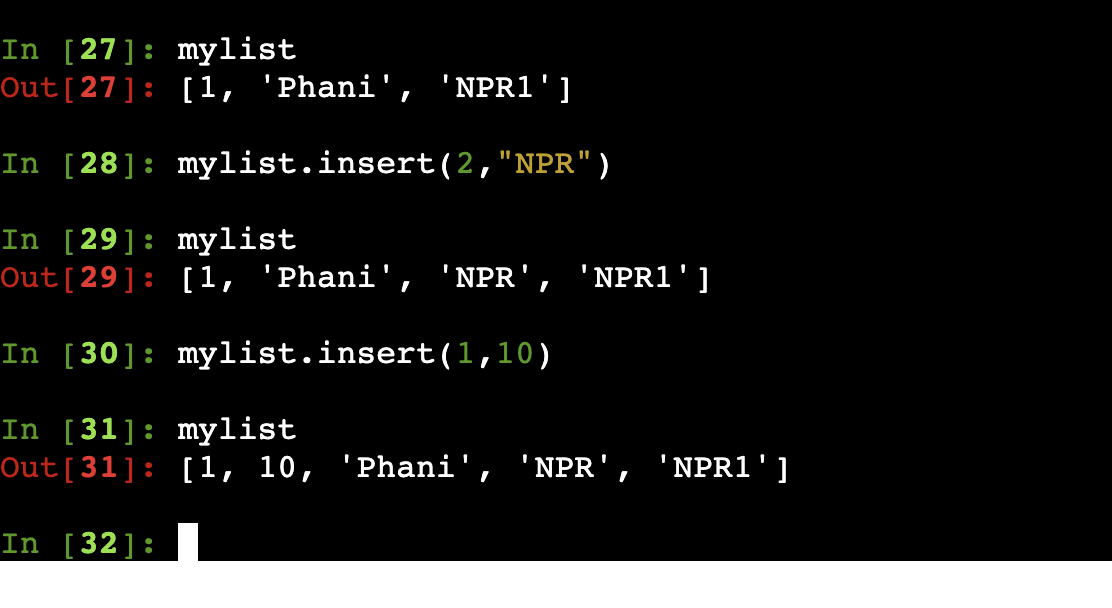
Insert function:

**Insert:**

* It’s a method used to add the element for a specific position to the list
* Syntax

Syntax  
mylist.insert(index,element)  
Example

mylist.insert(1,10)  
This will insert the element 10 to the index 1 of list

****

**Extend Function:**

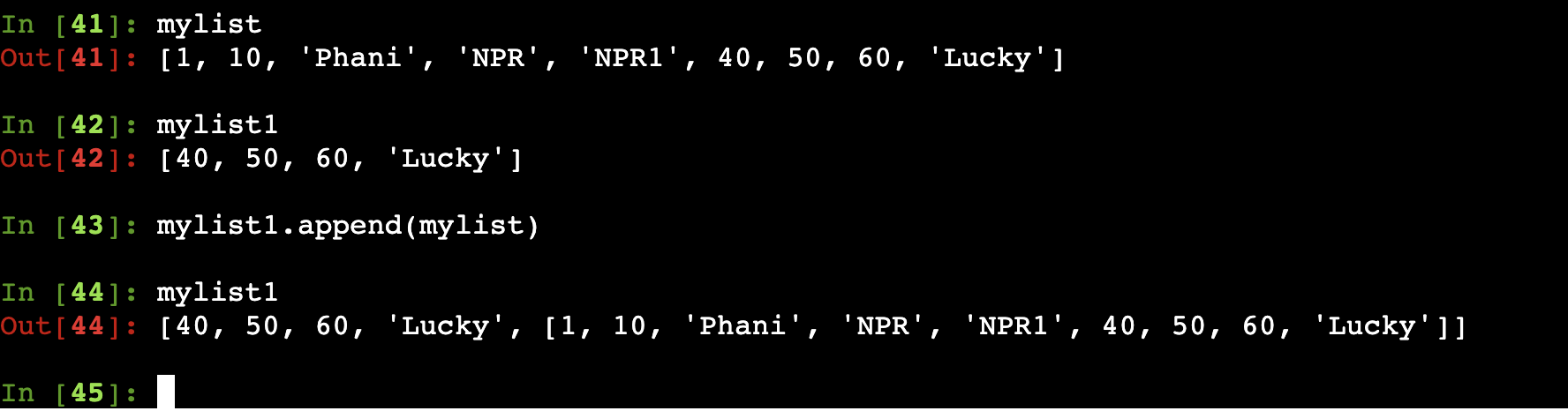
* It’s used to add only list to another list
* Syntax  
  mylist.extend(mylist1)  
  here mylist1 will be added to the mylist

****

observe the output below carefully.

When we try to append the list, the values of the list will be considered as single element

We shouldn’t use append, when we are trying to add one list to another list

****

**Note: for more function on list, go throgh the below doc**

<https://docs.python.org/3/tutorial/datastructures.html>

**Dictionary**: Dictionary is key value pairs. it is created using {} brackets (pair of curly braces). Using key, values of the dictionary can be accessed.

**Ex:**

mydict = {"username":"aaaaaa","userid":111,"permission":["read","write"]}

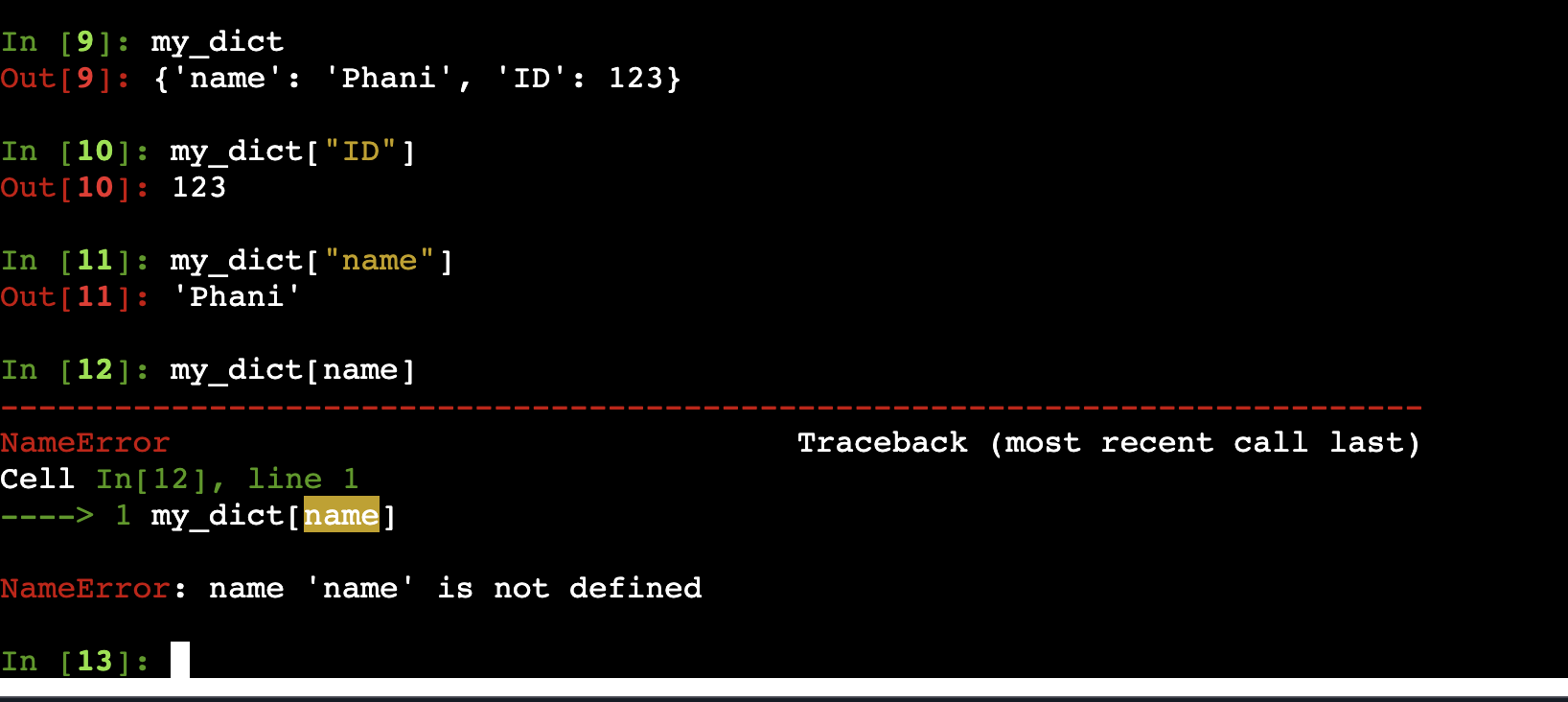
print(mydict[0]) does not work:: since its not list.. in dictionary we need to use key to print value

print(mydict["username"]) --- prints aaaaaa

mydict["permission"] ---- ["read","write"]

mydict["permission"][1] --- write

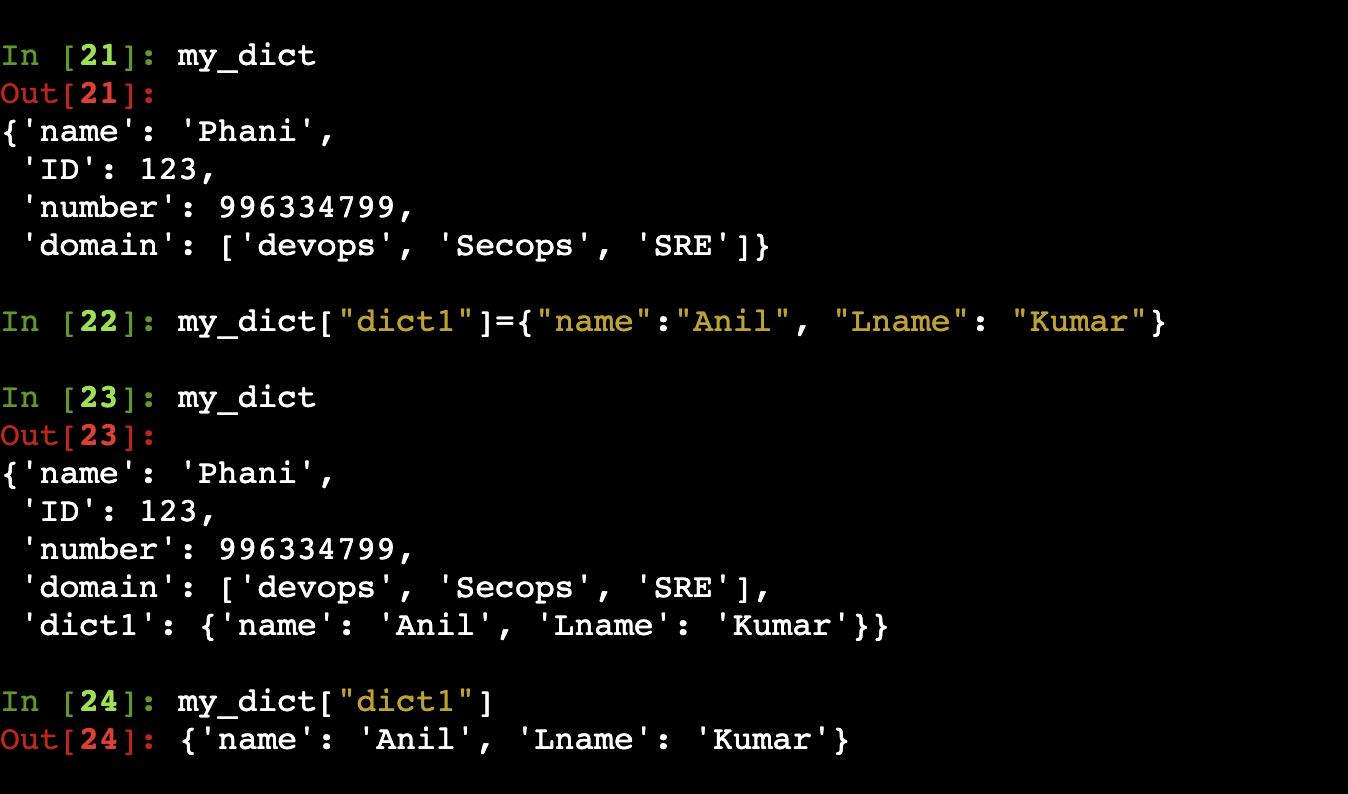
mydict["permission"][0] --- read

****

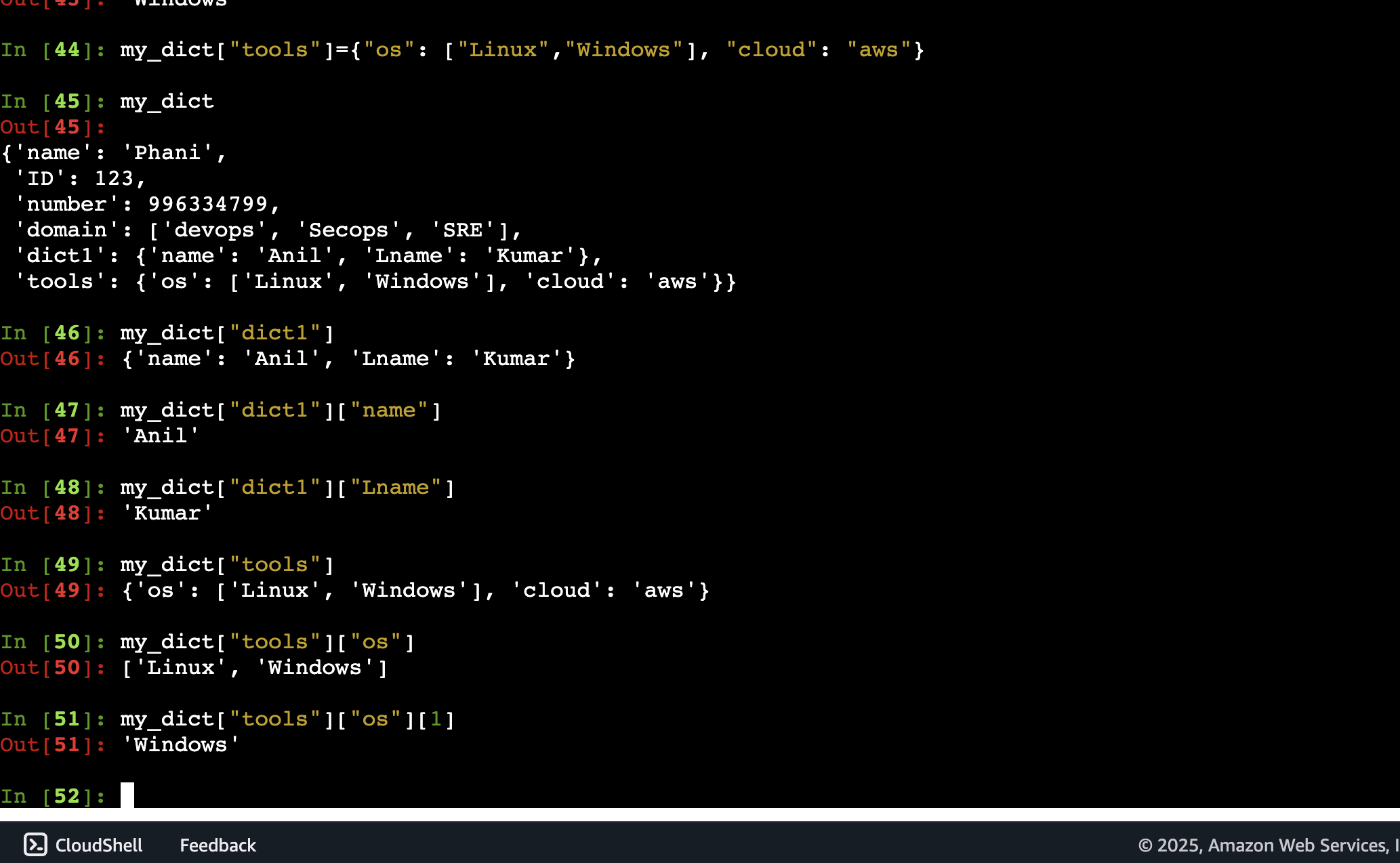
Adding elements and the list to the existing dictionary

****

Adding dictionary to the existing dictionary

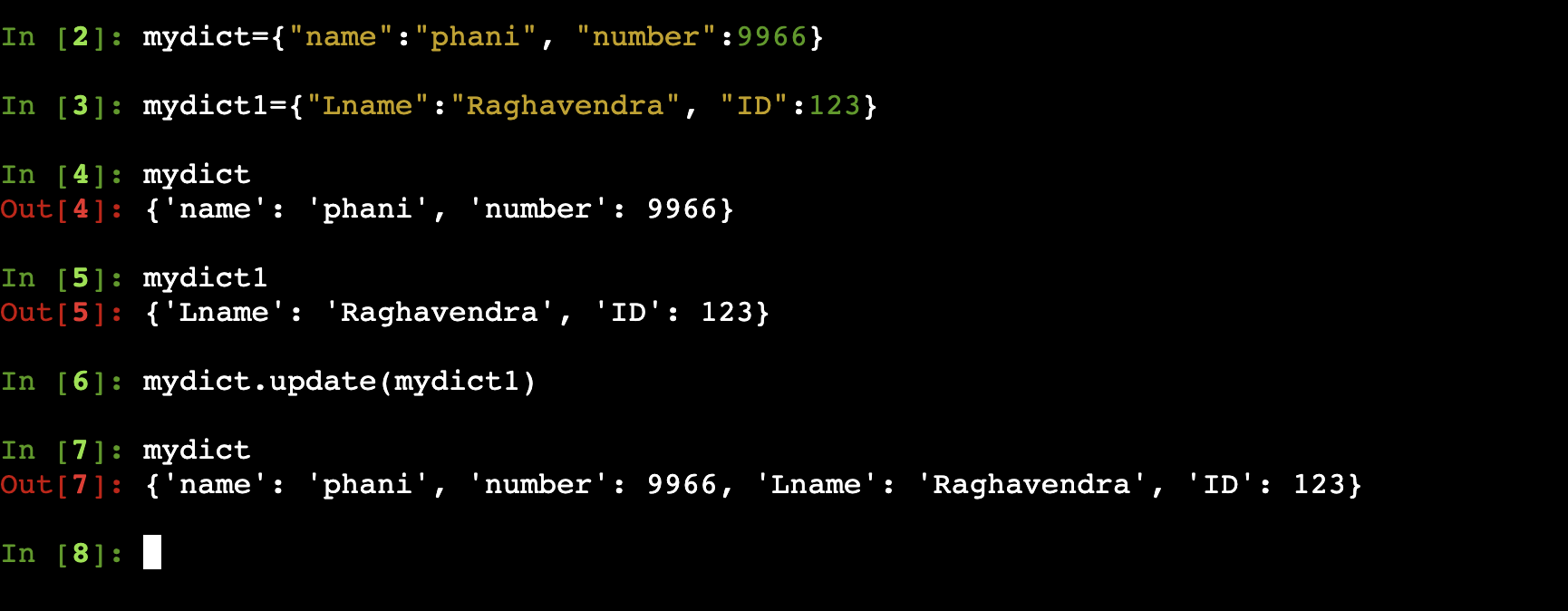


**Accessing the dictionary or the list**

****

**Update function**

* This is used to add on dictionary to another dictionary
* Syntax  
  mydict.update(mydict1)  
  here mydict1 is added to the dictionary mydict



**Script:**

**Script to read dictionary and display values**

aws\_policy = {

“Version”: “2012–10–17”,

“Statement”: {

“Effect”: “Allow”,

“Action”: [

“iam:AddUserToGroup”,

“iam:RemoveUserFromGroup”,

“iam:GetGroup”

],

“Resource”: [

“arn:aws:iam::609103258633:group/Developers”,

“arn:aws:iam::609103258633:group/Operators”

]

}

}

print(aws\_policy["version"])

print(aws\_policy["statement"])

print(aws\_policy["statement"]["Action"])

print(aws\_policy["statement"]["Action"][1])

**output:**

print(aws\_policy["version"]) ------- 2012–10–17

print(aws\_policy["statement"]) ------ {

“Effect”: “Allow”,

“Action”: [“iam:AddUserToGroup”,“iam:RemoveUserFromGroup”,“iam:GetGroup”],

“Resource”: [“arn:aws:iam::609103258633:group/Developers”,“arn:aws:iam::609103258633:group/Operators”]

}

print(aws\_policy["statement"]["Action"]) ------[“iam:AddUserToGroup”,“iam:RemoveUserFromGroup”,“iam:GetGroup”]

print(aws\_policy["statement"]["Action"][1]) ------ iam:RemoveUserFromGroup

**Assignment:**

**Day 48: Python**

**looping statements:**

**for loop:** It is used whenever we need to execute statements repeatedly

**syntax:**

**for val in variable:**

**print(val)**

**example:**

**list:**

1. fruits = [“apple”, “cherry”, “orange”]

for fruit in fruits:

print(fruit)

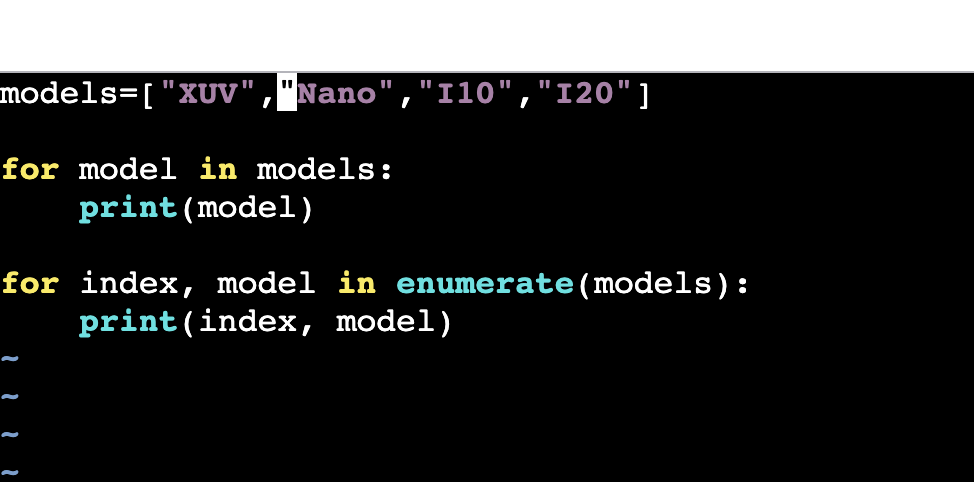
1. fruits = [“apple”, “cherry”, “orange”]

for index, fruit in enumerate(fruits):

print(f”Index: {index}, Fruits: {fruit})

1. for num in range(1,5):

print(num)



A black screen with white text

AI-generated content may be incorrect.

Dictionary:

emp\_data = {

“id”: 1111,

“name”: “aaaa”

“company name”: “sapiens”,

“domain”: [“developer”, “devops”, “tester”]

}

To get only keys elements

**for key in emp\_data.keys():**

**print(key)**

to get values from dictionay

**for val in emp\_data.values():**

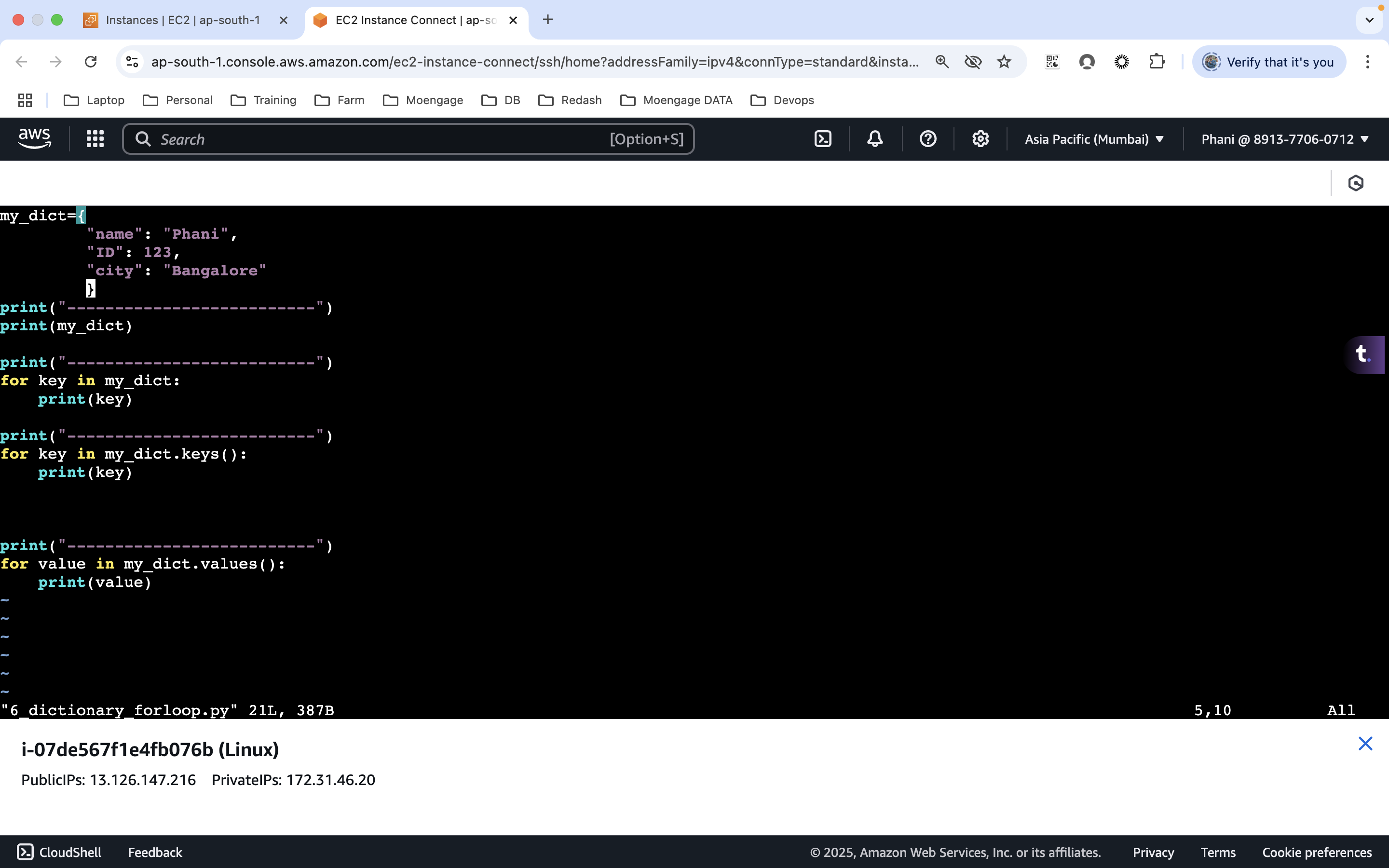
**print(val)**

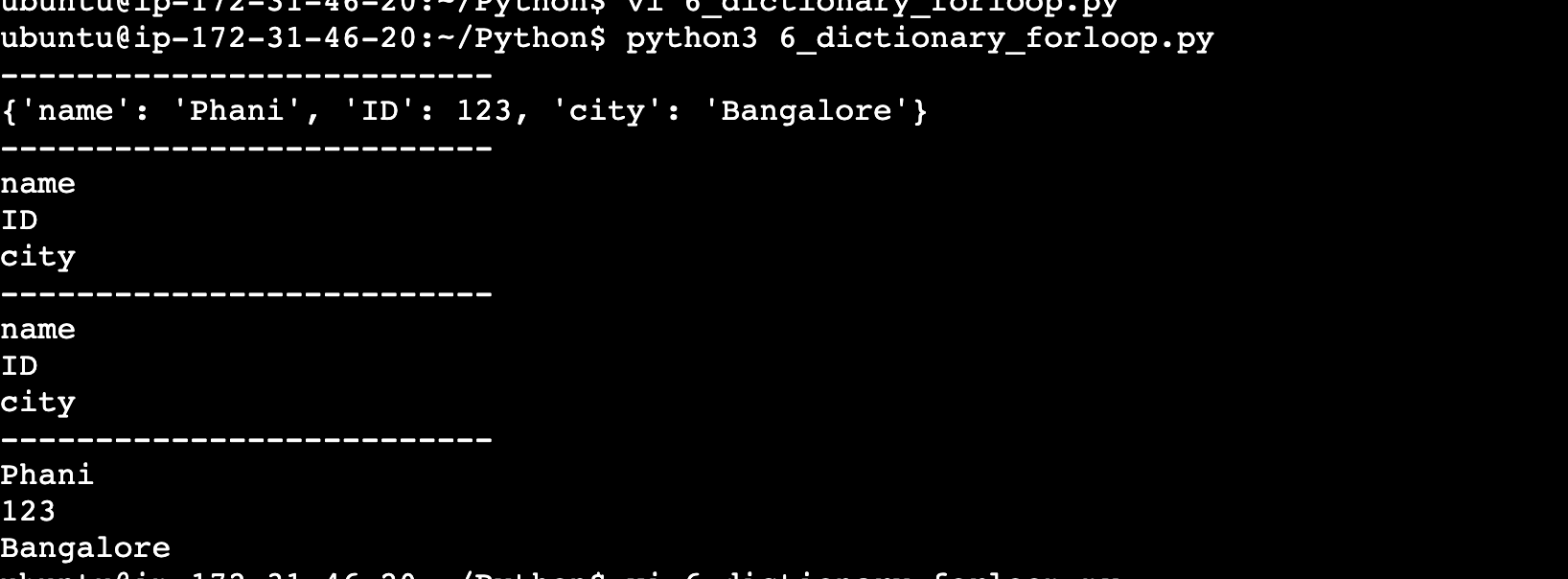
To get both key and value from dictionary

**for key, val in emp\_data.items():**

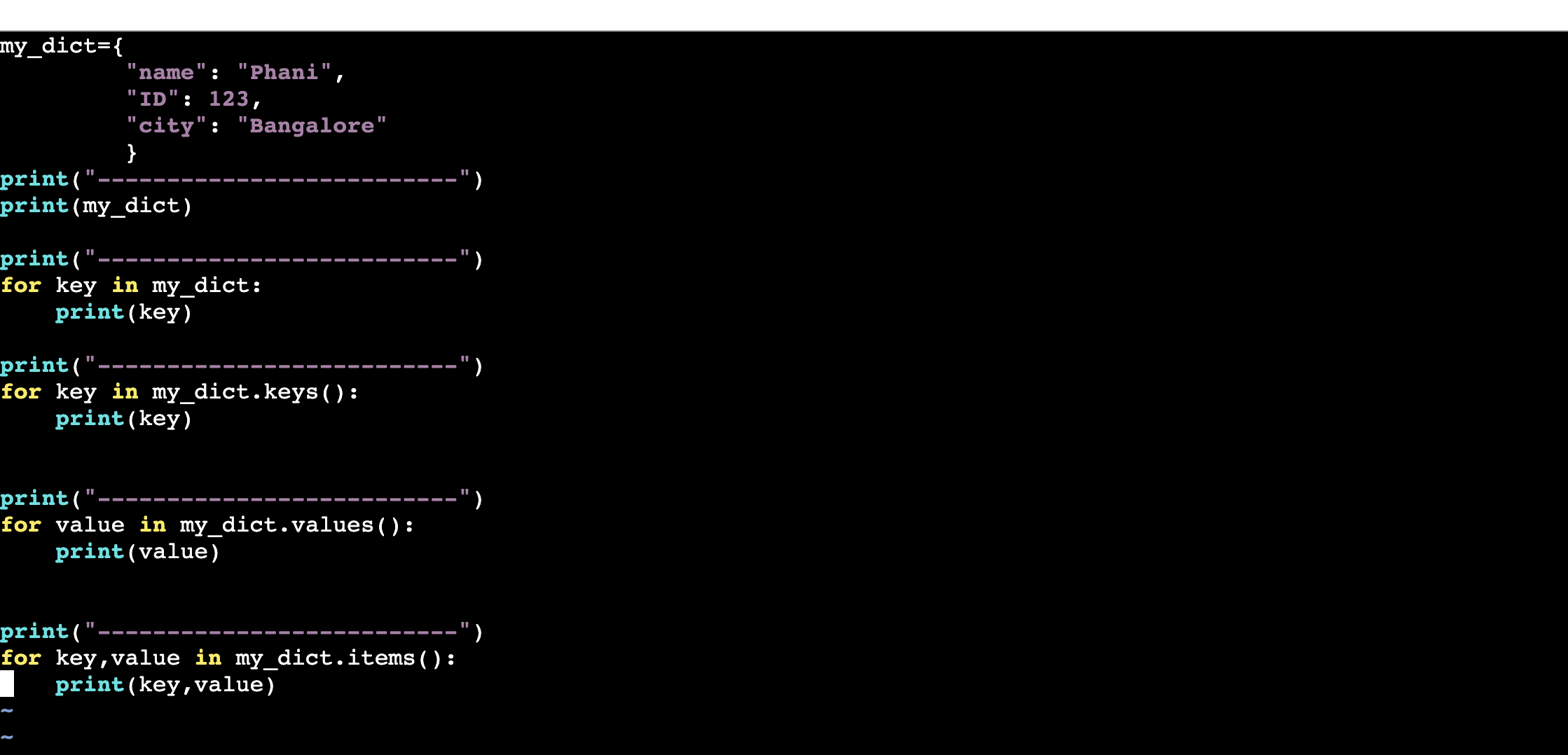
**print(f“{key}:{val}”)**

**Accessing key and values separately**





**Accessing both key and values in one shot**

****

**A screen shot of a computer

AI-generated content may be incorrect.**

**Conditional Statements:**

1. **If statements:**

**Syntax:** if condition:

**print()**

1. **If-else**

if condition:

print()

else:

print()

1. **If-elif**

if condition:

print()

elif condition:

print()

else:

print()

**Example:**

1. To check true or false: found = True

if found:

Statement

else:

statement

1. Fruits = [“apple”, “banana”, ‘cherry”]

**if “apple” in fruits:**

**print(“found”)**

**else:**

**print(“Not Found”)**

or

**if “apple” not in fruits:**

**print(“not Found”)**

1. If not found:

Statement

Above will check whether found is false

1. To check whether list or dictionay is empty or not

mylist = [10, 20, 23, 24]

**if mylist:**

**print(“List contain values”)**

**else:**

**print(“List empty”)**

1. To compare use =, >, < , !=, >=, <=
2. To check key is there or not in dictionary

**emp\_data = {**

**“id”: 1111,**

**“name”: “aaaa”**

**“company name”: “sapiens”,**

**“domain”: [“developer”, “devops”, “tester”]**

**}**

**if “name” in emp\_data:**

**print(“Key found”)**

**else:**

**print(“Key not found”)**

To check value in dictionary

**if “sapiens” in emp\_data.values():**

**print(“Value Found”)**

**else:**

**print(“Value not Found”)**

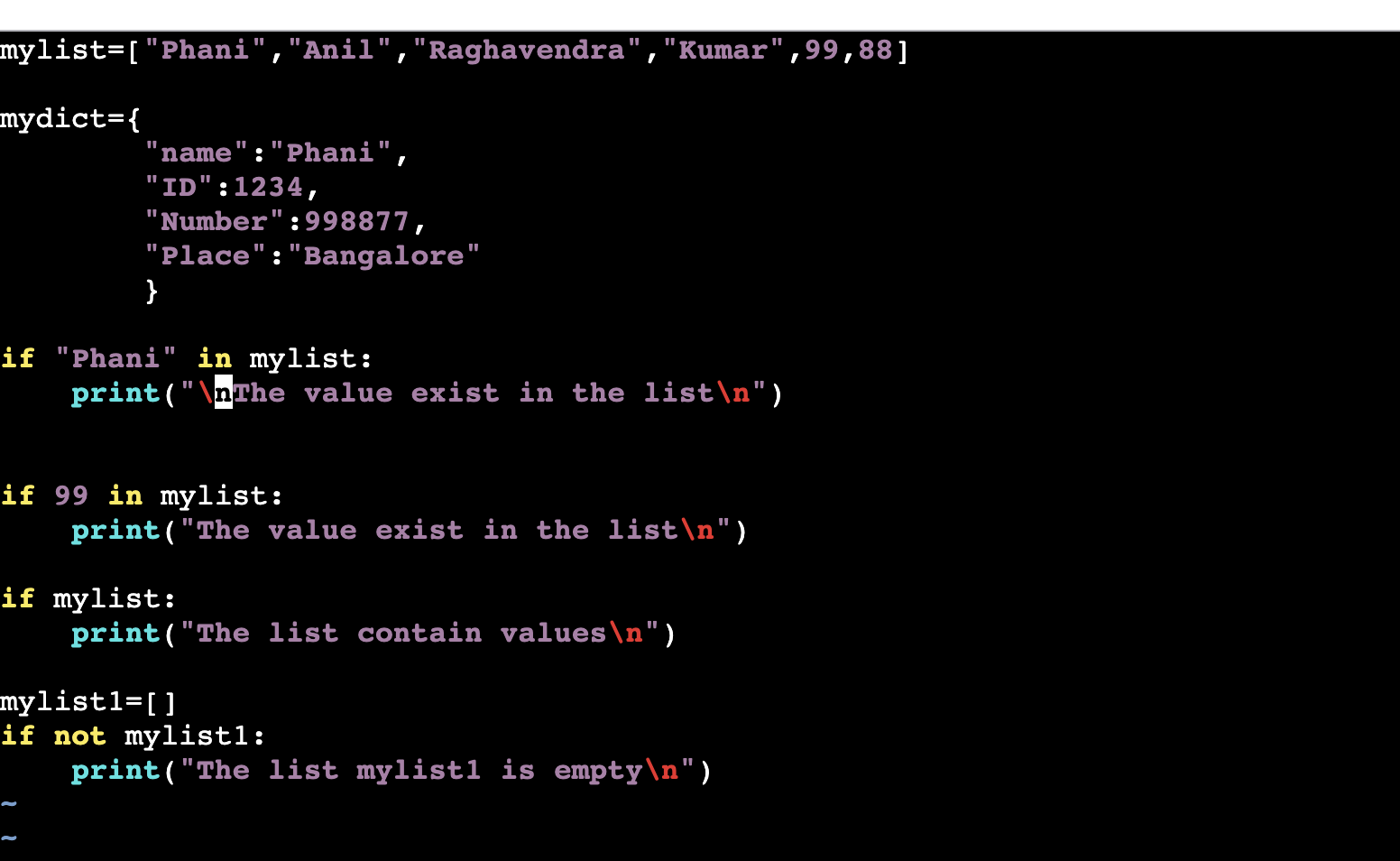
To check speck key contain value

**if emp\_data[“name”] == “aaa”:**

**print(“Value found”)**

**else:**

**print(“Not Found”)**



A screen shot of a computer

AI-generated content may be incorrect.

**Dictionaries:**

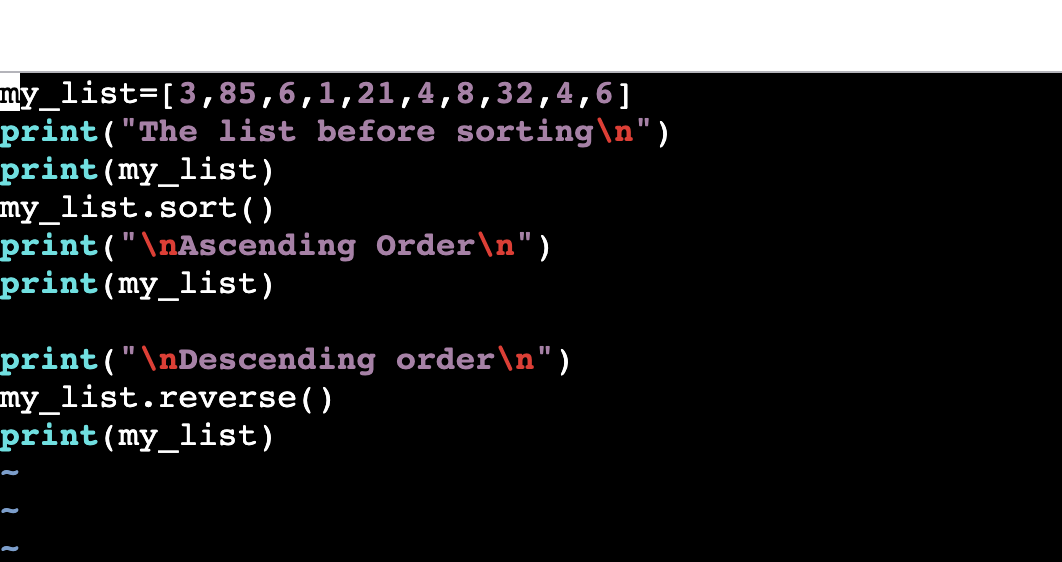


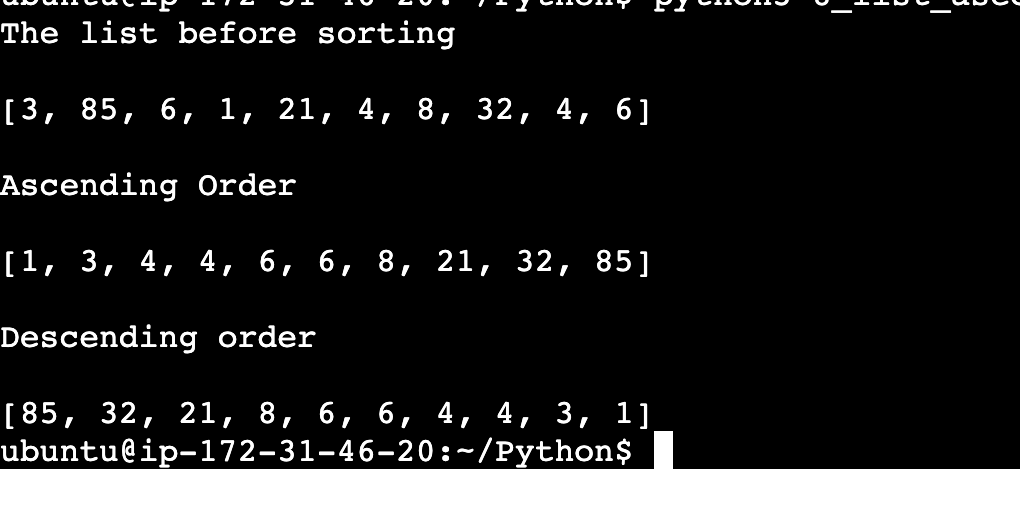
A black screen with white text

AI-generated content may be incorrect.

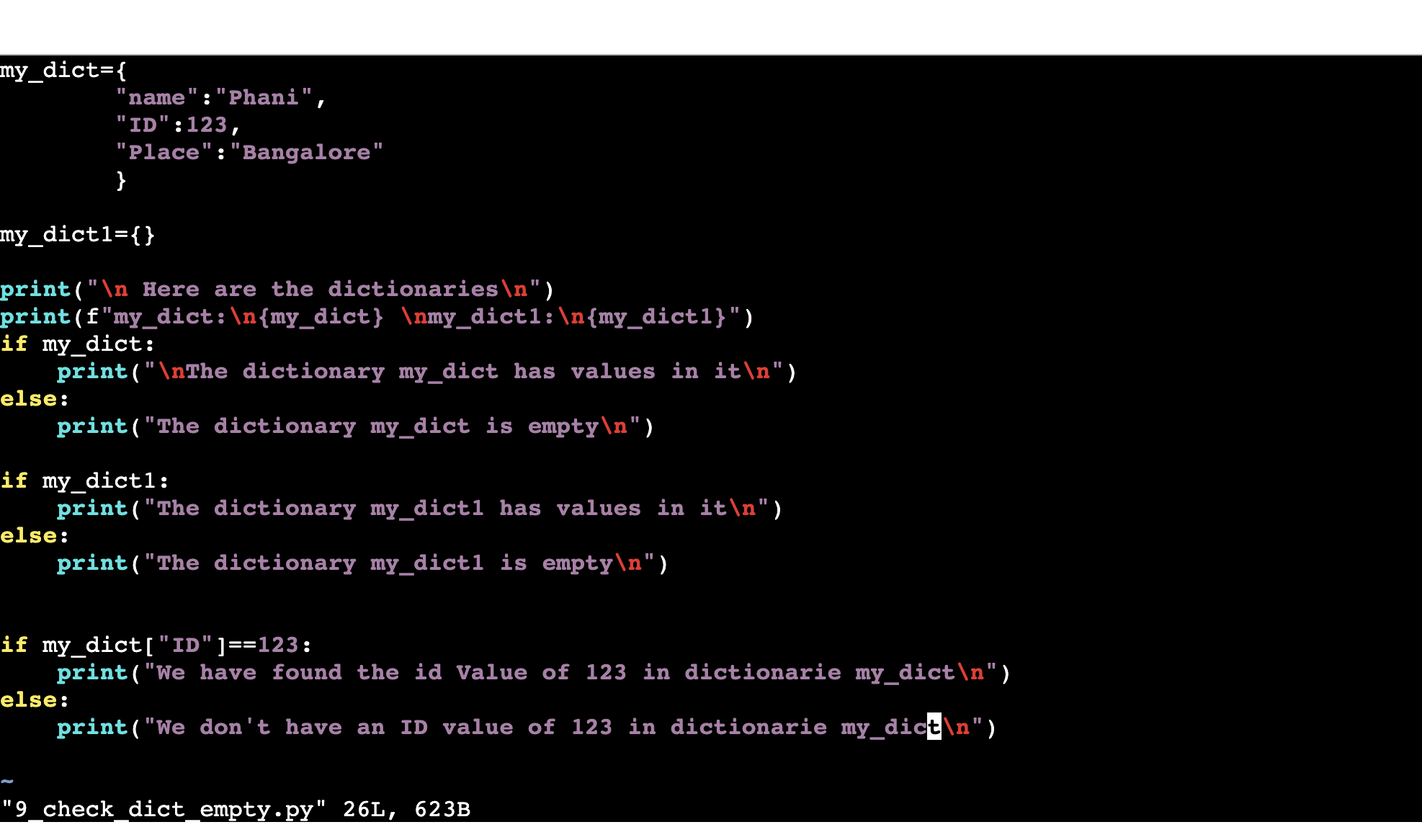
**Assignment:**

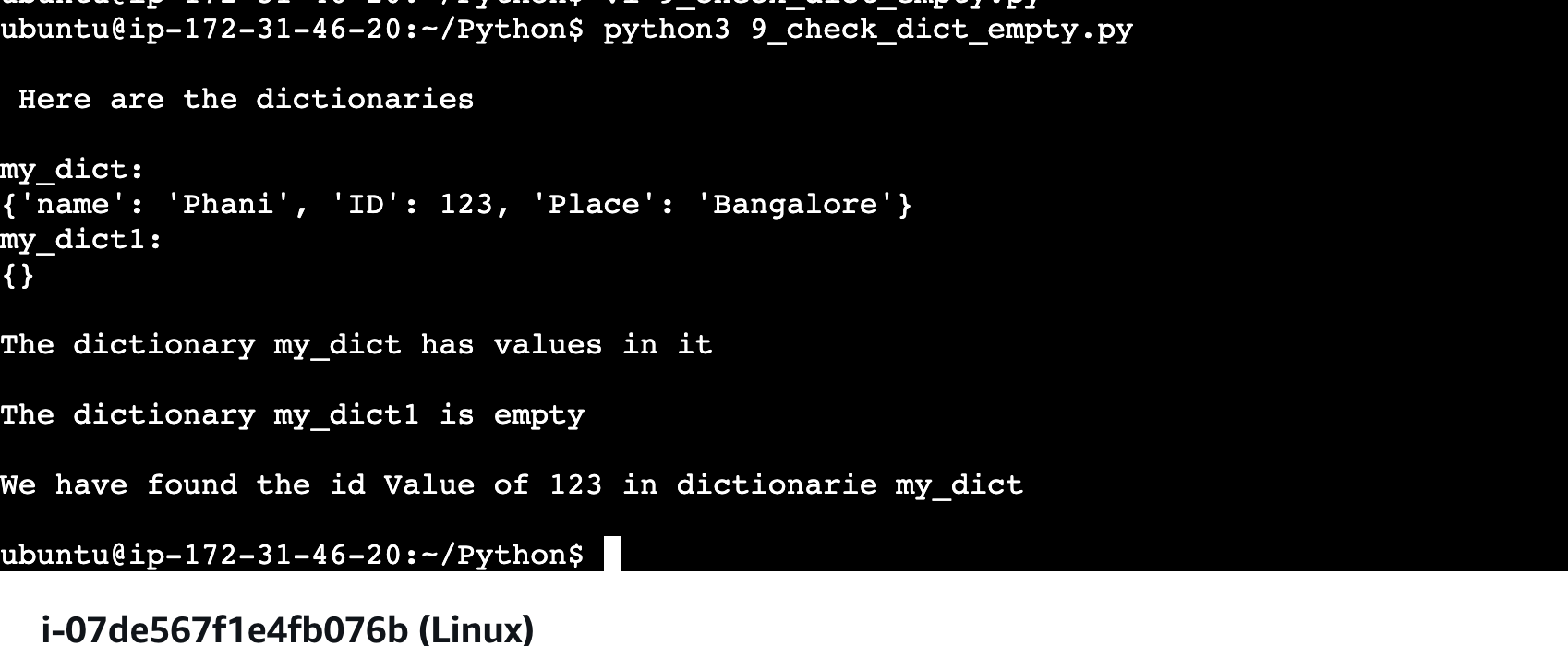
Write a script to display the elements in ascending order?



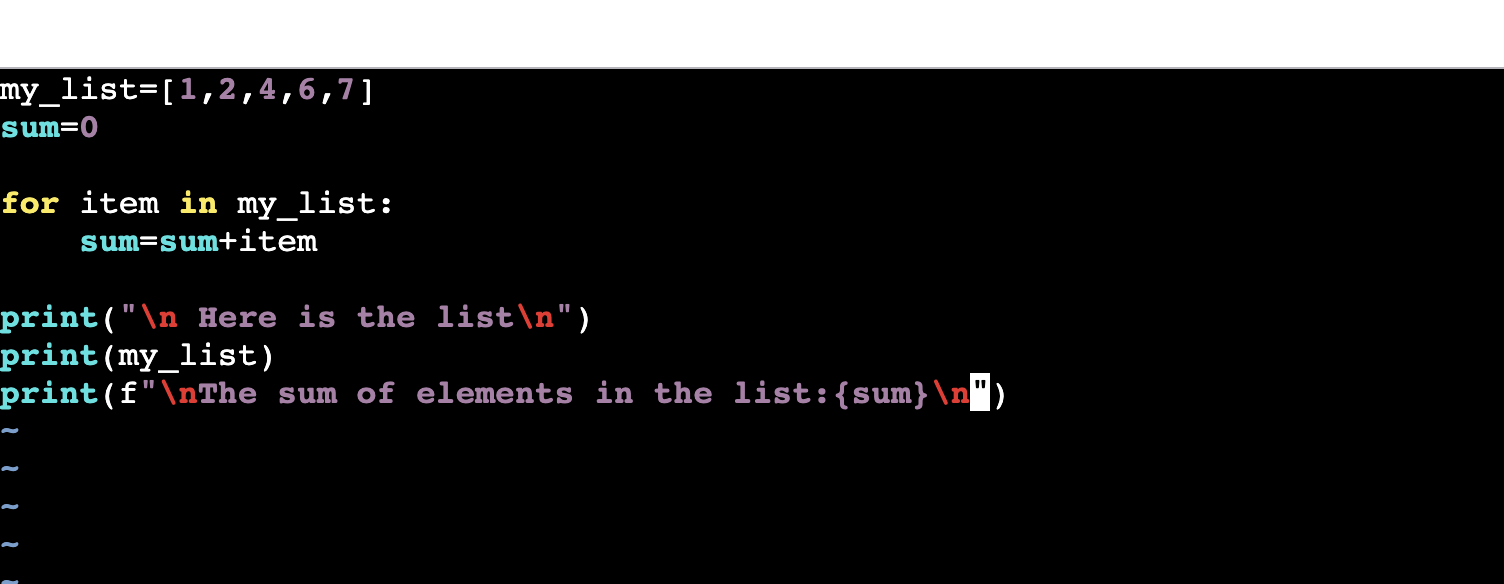


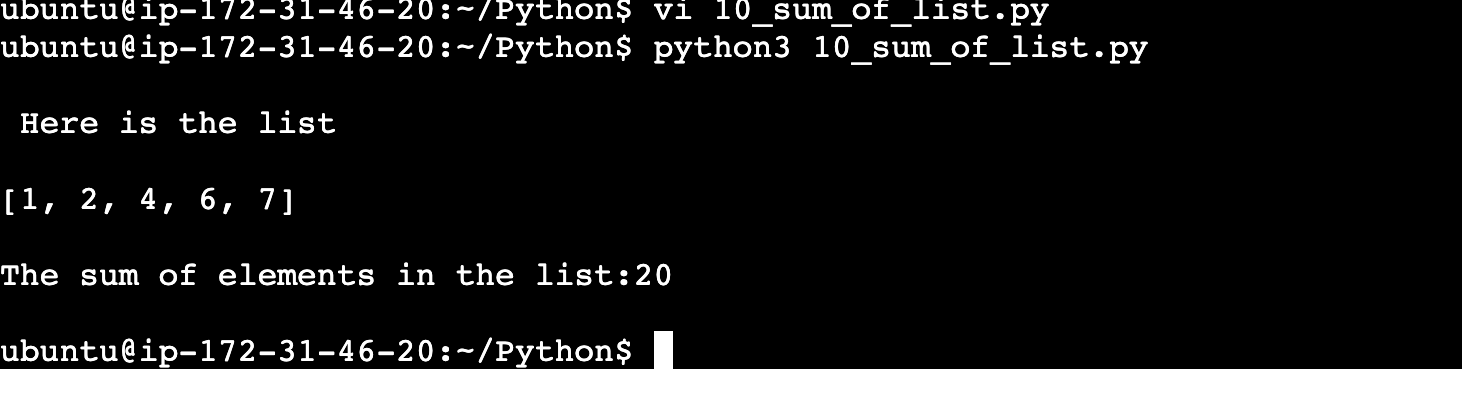
Write a script to check if the value is present in the dictionary or not?



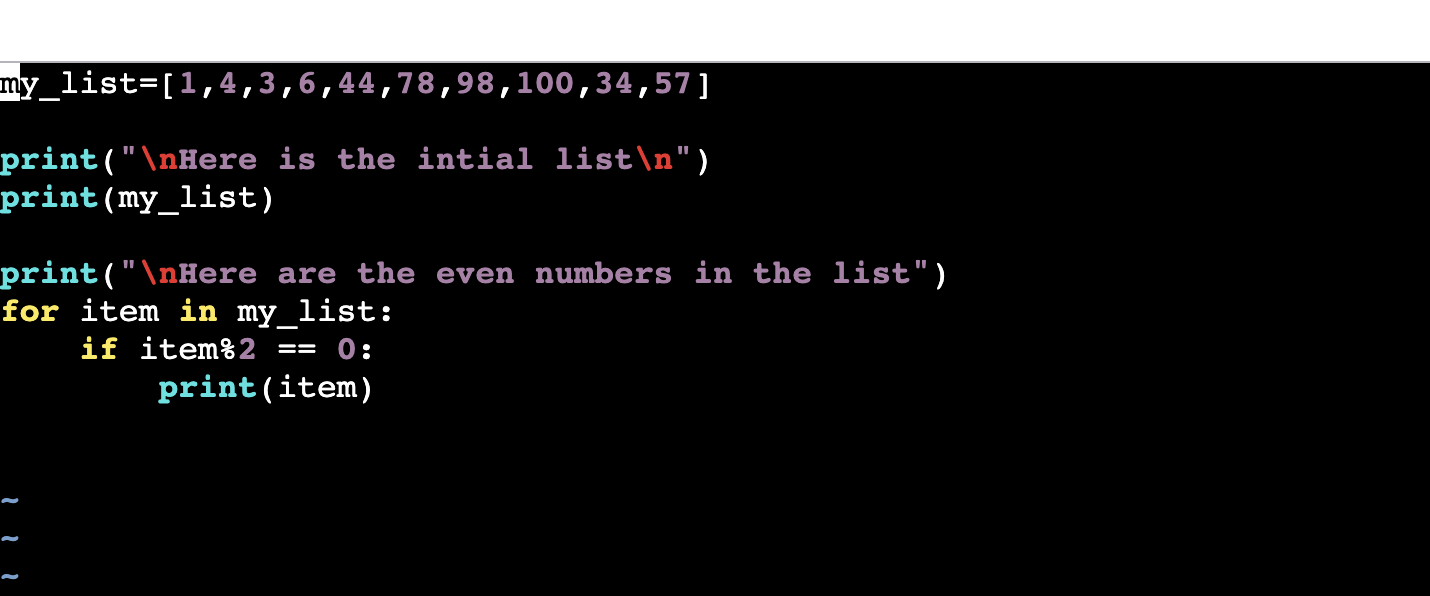


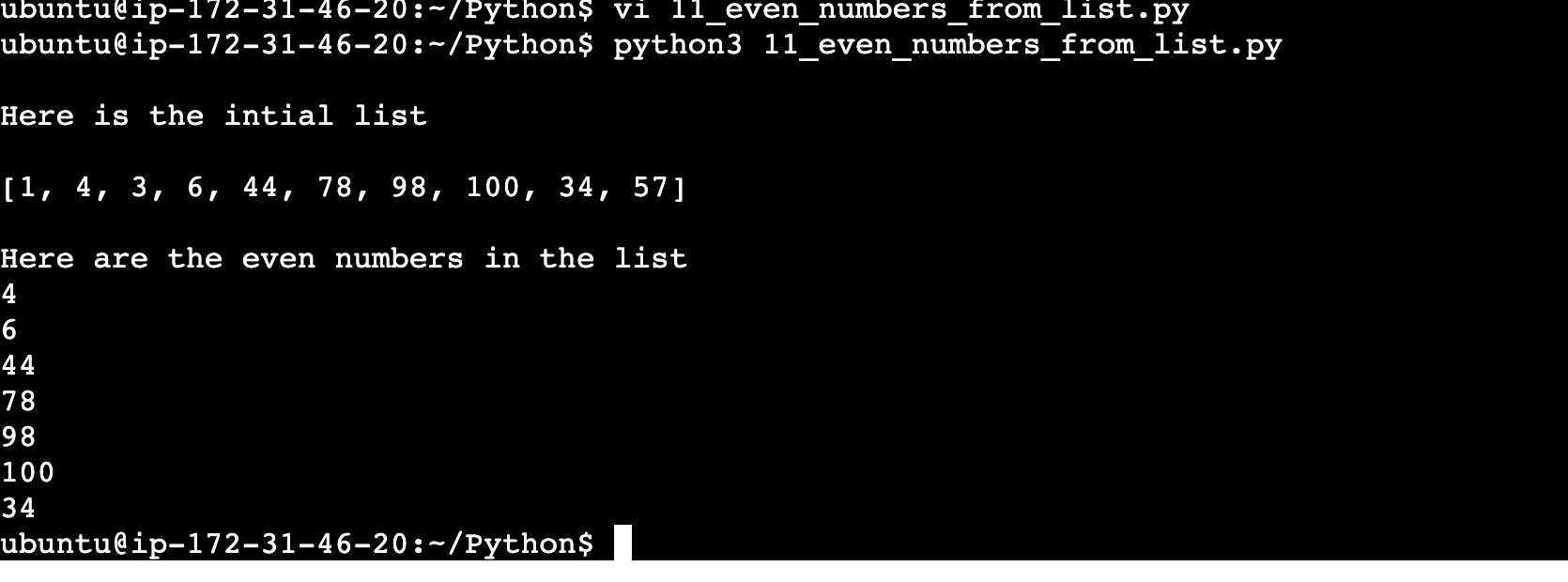
Write a script to find sum of all elements of the list?





Write a script to display only even numbers in the list?





**Day 49: Python**

**Basic build-in methods:**

1. lower(): It is used to convert string into lowercase

var = “Hello.World”)

var1 = var.lower()

print(var1)

1. upper(): Used to convert string into uppercase

var = “Hello.World”)

print(var.upper())

1. startswith(): used to check whether string is starts with pattern

var = “Hello.World”)

if var.startswith(“He”):

print(“starts with pattern”)

else:

print(“starts with different pattern”)

1. endswith(): used to check whether string ends with specific pattern or not
2. var = “Hello.World”)

if var.endswith(“He”):

print(“ends with pattern”)

else:

print(“sends with different pattern”)

1. split(): used to split the string based on delimiter

var = “hello.wor.ld”

val, val1, val2 = var.split(‘.’)

above string is divided into 3 parts “hello”, “wor”, “ld”

val, val1= var.split(‘.’, 1)

above is splitted only one time since in function, 1 is passed. (Hello, wor.ld)

val, val1 = var.rsplit(‘.’, 1)

splitted from reverse order (Hello.wor, ld)

1. replace: Used to replace the one character by another character

var = “hello.wor.ld”

var1 – var.replace(‘.’, ’/’)

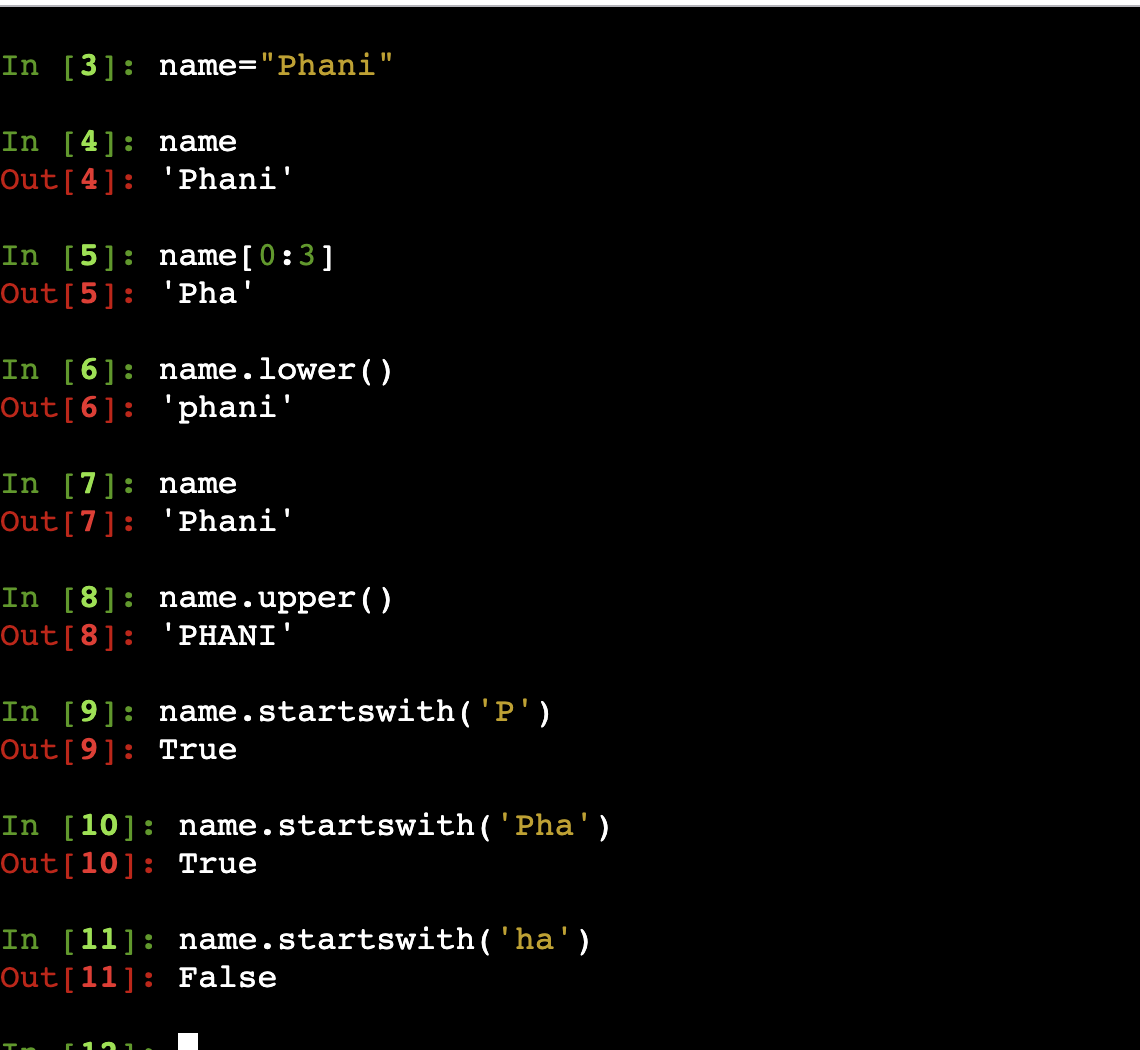
print(var1) ------- Hello/wor/ld

1. range: used to print the range between values

print(range(1,5))

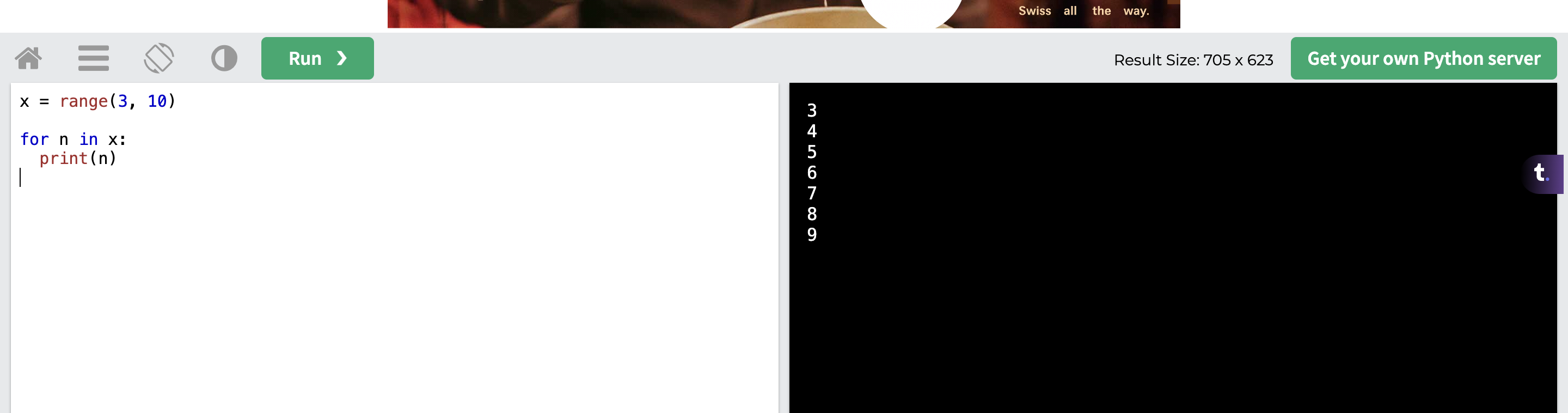
1. random: used to print random numbers

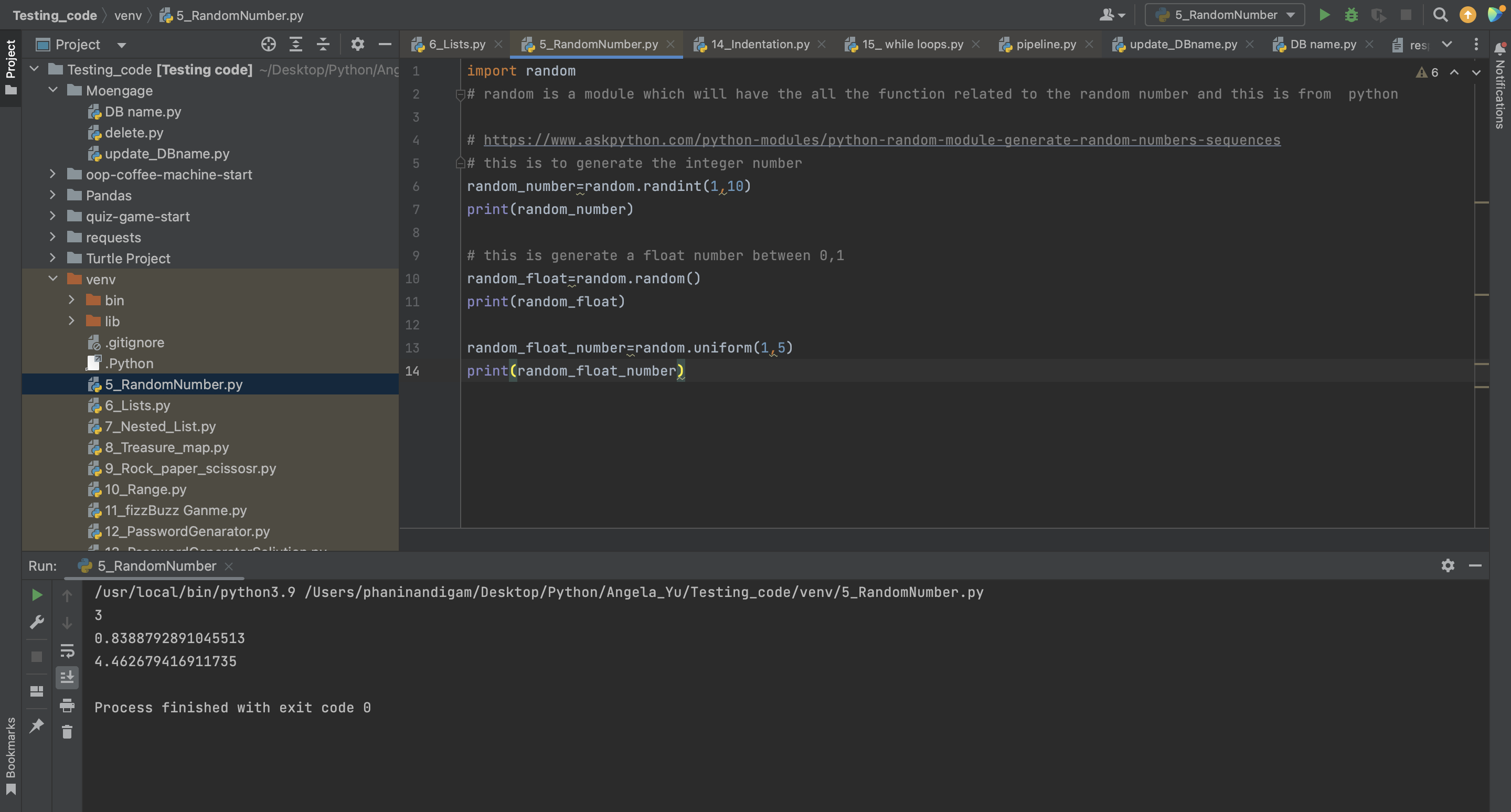
print(random(2,8))



****

****

****

****

**Sys module:** This module provides access to some variables used or maintained by the Python interpreter and functions that interact strongly with the interpreter. To use this module, need to import sys module.

**Syntax: import sys**

**Ex:**

1. **sys.argv:** It’s a listin Python, which contains the command-line arguments passed to the script. The first element, **sys.argv[0]**, is the script name itself.

**sys.argv[1],** first argument passed via command line

1. **sys.path:** print the current system path

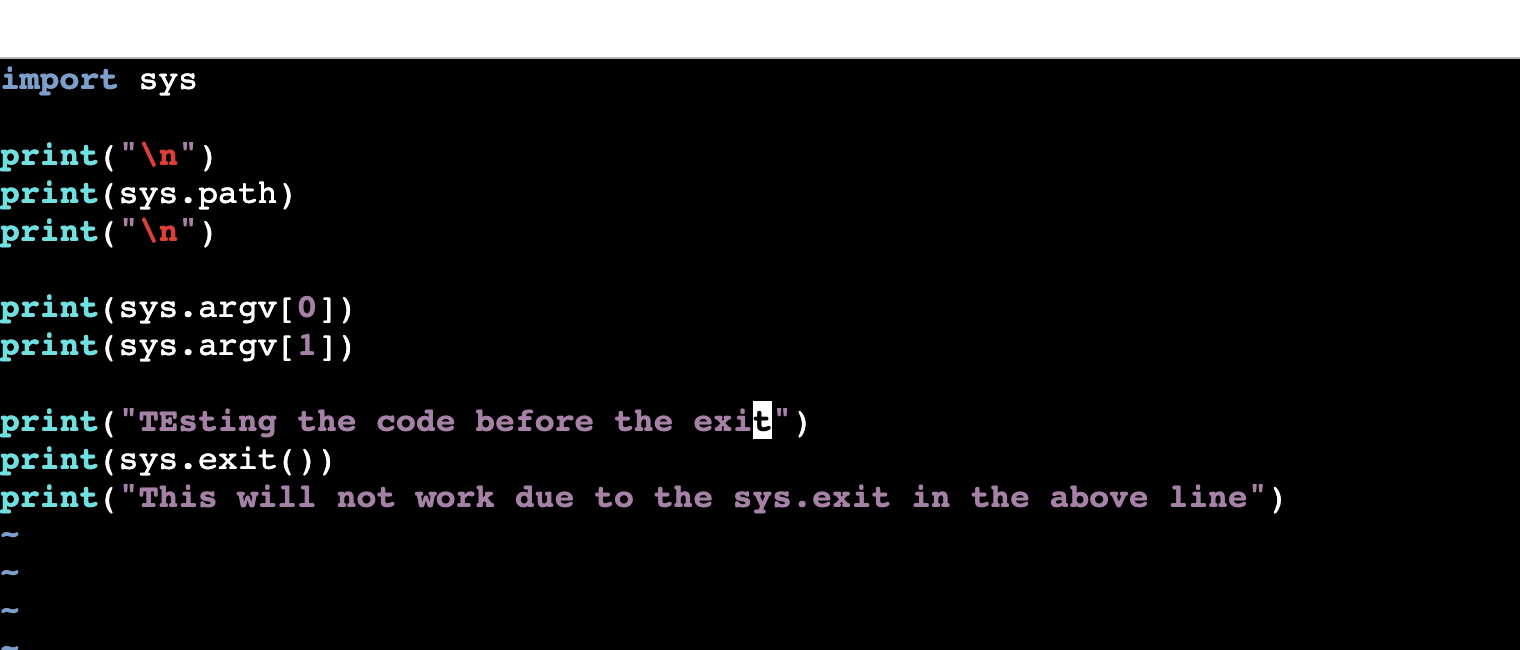
print(sys.path)

1. **sys.exit:** used to exit from the python interpreter by raising the systemexit exception.

sys.exit(0)

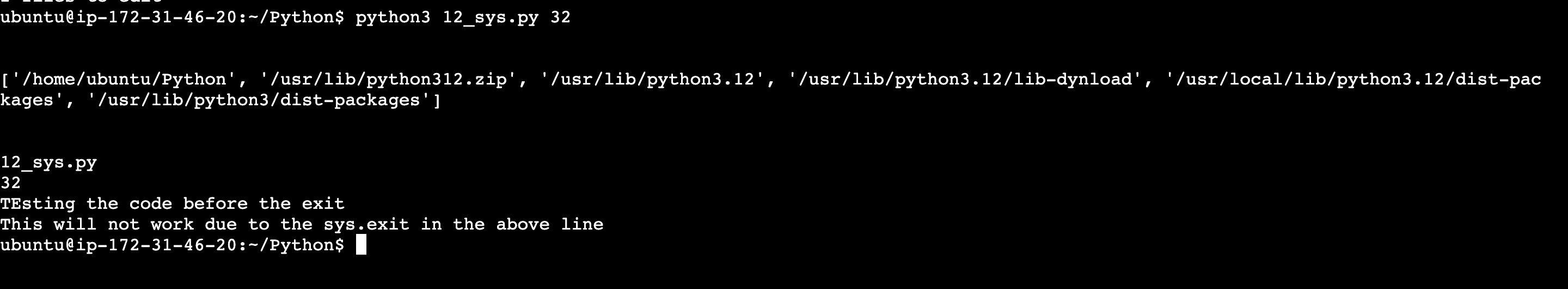
sys.exit(“Aborted”)

1. **sys.stdin, sys.stdout, sys.stderr:** provide access to standard input, output and error streams respectively





If we comment the exit function:



**OS Module:** This module provides the way for interacting with the operating system.

1. os.getcwd(): used to display current directory
2. os.chdir(): it will change the directory just like cd in linux

os.chdir(“/home/ubuntu/dir1”)

1. os.makedirs(): used to create directory

os.makedirs(“dir1”, exist\_ok=True)

1. os.path.join(): It will joins the path

os.path.join(‘/home/ubuntu’, ‘file1.txt’) ---------- /home/ubuntu/file1.txt

1. os.path.exists(): It will check whether the path exists or not

os.path.exists(‘/home/ubuntu/1.txt’) --- return true if it exists otherwise false

1. os.path.isfile(“1.txt”): it will check it’s a file or not and return true if its exists
2. os.path.basename(): it will return basename

os.path.basename(‘/home/ubuntu/1.txt’) ------- returns 1.txt

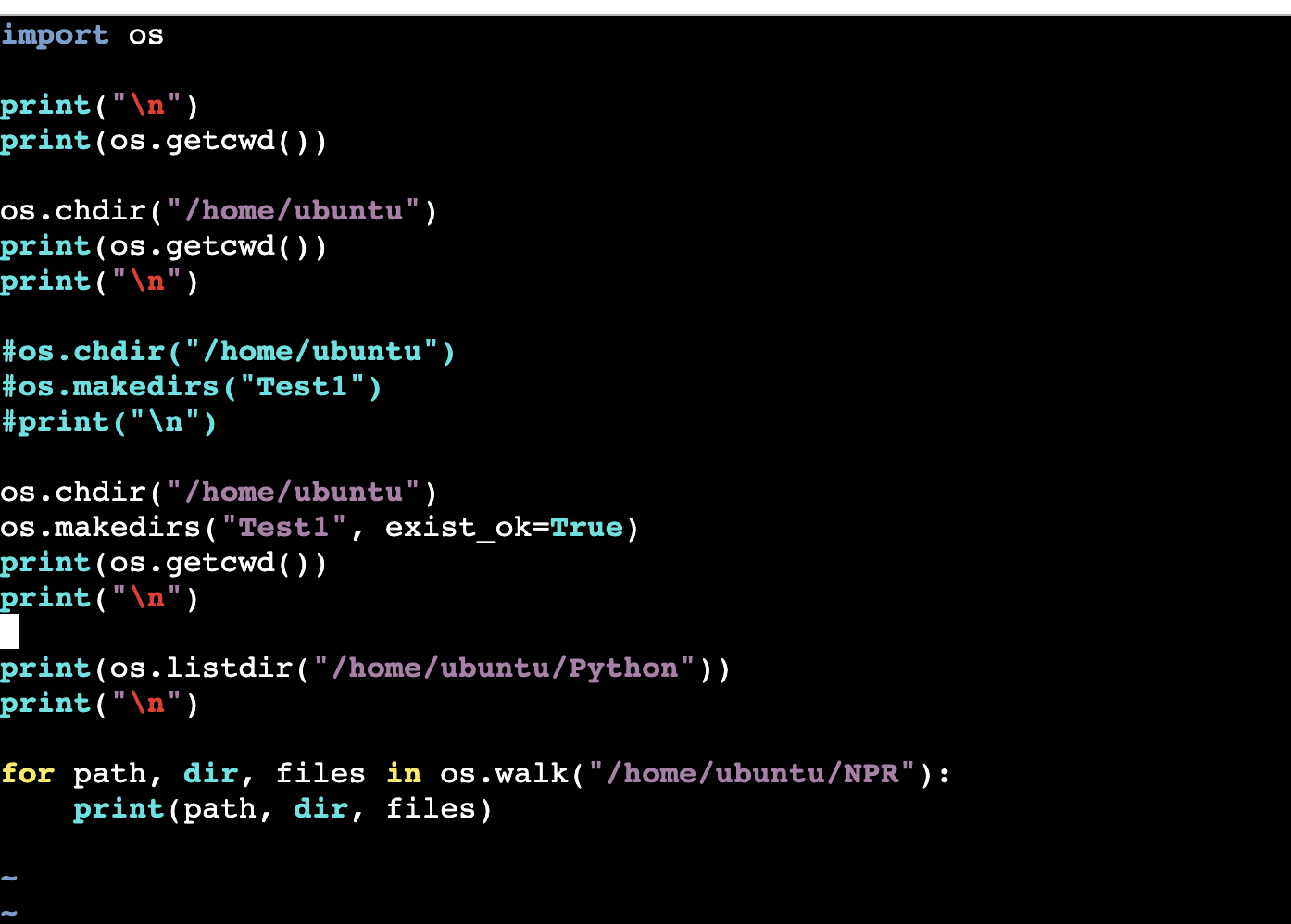
1. os.listdir() : It is used to list all the directory and files from the specified path.

os.listdir(/home/ubuntu) ------ list all files and directory from this path

1. os.walk(): It is used to walk through all the directory and sub directory and in each iteration returns current path, all the files and directories

ex: for root, dir, files in os.walk(“/home/ubuntu”)”

print(root, dir, files)



A computer screen with white text

AI-generated content may be incorrect.

Write a script to list the text files in the path /home/ubuntu?

**Day 50: Python**

**Day 51: Python**

**Files Handling:** File handling is an important part of any web application. Python has several functions for creating, reading, updating, and deleting files. The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode. There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist and if it exists it will overwrite.

"x" - Create - Creates the specified file, returns an error if the file exists.

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Note: To work with files, three different modes are available in python

1. read mode: If the file is opened in read mode then file can be used for only reading purpose. The character used is ‘r’.
2. Write mode: If the file is opened in write mode then file can be used to write the contents to the file. If the file is exists then it will overwrite otherwise it will create a new file. The character used is ‘w’.
3. Append: If the file is opened in append mode then file can be used to append the contents to the file. If the filed doesn’t exists then new file will create otherwise it will append at the end of the file

**Ex:**

with open(“1.txt”, ‘r’) as fh:

content = fh.read()

print(content)

with open(“1.txt”, ‘w’) as fh:

fh.write(“Hello Good Morning”)

with open(“1.txt”, ‘a’) as fh:

fh.write(“Hello Good Morning”)

mylist = [“Hello”, “Good”, “Morning”]

with open(“1.txt”, ‘w’) as fh:

for data in mylist:

fh.write(f”{data}\n”)

we can also write content from one file to another

with open(“1.txt”, ‘r’) as fh:

content = fh.read()

with open(“2.txt”, ‘w’) as fh:

fh.write(f“{content}”)

**Functions:** Set of instruction which is used to define specific task. To use function, user need to include two things

1. Function definition: Set of instruction to achieve specific task. The syntax is as follows

**def func\_name(arg1, arg2, ..):**

**statement1**

**statement2**

**return val1, val2, ..**

1. Function Call: Once function is define, need to call that function. The syntax is as follows

**val1, val2, .. = func\_name(arg1, arg2, ..)**

**Note:** Need to practice all scripts which was in github repository.

**Pip:** It is the python package manager which is used to install python modules.

**Apt install python3-pip**

**Pip3 install modulename**

**Classes:** Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. To understand the meaning of classes we have to understand the built-in \_init\_() function. All classes have a function called \_init\_(), which is always executed when the class is being initiated.

\_\_init\_\_ method is used as a constructor for classes. Whenever a new instance of a class is created, \_\_init\_\_ is called first. Constructors are usually used to set up class attributes. Use the \_init\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

class Person:

def \_init\_(self, name, age):

self.name = name

self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)

**Note:** The \_init\_() function is called automatically every time the class is being used to create a new object.

**Day 53: Python**

Classes and objects

**Day 54: Python**

Classes and objects

**Day 55: Python**

**Day 56: Python**

Custom commands