Pip install flask

print(‘’’……..’’’)

from playsound import playsound

keywords – def, class, etc.

strings 🡪 a= {‘…’, “…”, ’’’…’’’}

print(type(a))

In python everything is an object of classes.

Variable are like containers

and, or, not

a = int(a) //Typecasting

a = input(“Enter your name”) //input() function->takes input as string

name[3] = “d” 🡪does not work //not support item assignment

print(name[0:3])

print(name[:3])

print(name[0:])

<|Name|>

name = input(“Enter your name”)

letter = letter.replace(<|Name|>, name)

Can change the element in the list.

print(friends[0:4]) 🡪List slicing

L1.append() 🡪 Adds the element at the end of the list

L1.insert(3, 8) 🡪 adds 8 at index 3

Python Docs 🡪 List Methods

List meth 🡪 print(sum(a))

Make list using []

Create a tuple using ()

You cannot update the values of tuple

print(myDict[‘Fast’])

print(myDict[‘anotherDict’][‘piyush’]) 🡪Nested Dictionary

Dictionaries are mutable.

#IMP{Diff b/w .get & [] syntax in Dictionaries}

print(myDict.get(‘piyush’)) 🡪 Returns none

print(myDict[‘piyush’]) 🡪 Throws an error

No Repetitive items in sets.

#This syntax will create an empty dictionary and not an empty set,

a = {}

#Empty set syntax

a = set()

a.add 🡪adds element to set

PracticeSet 🡺cannot store list in a set.

Indentation🡪space🡪to show next line is inside the last line.

If-elif-else

Multiple if statements.

--------------------------

a = NONE

if( a is NONE):

print(“Yes”)

else:

print(“No”)

---------------------------

a = [45 , 3 , 56]

print(455 in a)

---------------------------

pass 🡪here I’ll write after sometime till then don’t produce error. DO NOTHING

i = 0

while i<10:

print(“Yes”)

i = i + 1/ i +=1

----------------------------

k = [1, 2, 3]

for item in k:

print(item)

----------------------------

For with else

Else will only execute with a successful execution of loop. If loop stops due to the break statement, else block will not be executed.

if i == 5:

continue 🡪 skipped the loop at i = 5

Continue is kinda cousin of break.

-------------------------------

Function starts with a keyword def. Eg: def piyush():

f string🡪 print(f”{num}X{i}={num\*i}”)

end = “” 🡪 it’ll not print the by default new line.

Default arguments in function.

def greet(name = “Stranger”)

string concatenate with str….

print(“Hello ”, end = “ ”) 🡪 To not get a new line, by default its \n.

import random

random.randint(1, 3) 🡪 it’ll chose either 1, 2 or 3.

f.write() takes value in a string.

Content = f.read().lower()

os.remove(sample2.txt)

OOPS

Functions also implement Dry principles.

Class is a blueprint for creating an object.(No memory taken by class)

class Number:

company = “Google”

def sum(self):

return self.a + self.b

num = number()

self.a = 4

self.b = 6

s = num.sum()

print(s)

Number.company = “Youtube” //Class Attribute

----------------------------------

Encapsulation 🡪Suppose we have two class, then both of them wouldn’t know the methods and data of other class.\

Abstraction 🡪 After creating an object, we only call functions. Therefore, all the unwanted data or implementation (from the class) is not shown or can say abstracted.

INSTANCE ATTRIBUTE 🡪 piyush.name = “Piyush”

🡪 piyush.salary = “300k”

SELF🡪

class Employee:

company = “Google”

def getSalary(self, signature):

print(“Salary for the employee working in {self.company} is {self.salary}”\n{signature})

piyush= Employee()

piyush.salary = 1000000

// piyush.getSalary (“Thanks!)”

piyush.getSalary() // 🡪coverts into🡪 Employee.getSalary(piyush)

🡪SELF is a parameter which automatically passes when you calls an object.

🡪With the help of SELF we can print both instance and class attributes.

-------------------------------------------------------------------------------------

Static method decoration🡪to implement method when it’s not considering class instance.

@staticmethod

def greet()

print(“Good Morning”)

piyush = Employee()

piyush.greet()

\_\_init\_\_ constructor🡪

def \_\_init\_\_(self, name, salary, subunit):

(Read file 08\_constructor.py to understand this)

super().takeBreadth()

super().\_\_init\_\_()

CLASS METHOD DECORATION

#we don’t want to add instance attribute but want to change class attribute.

/\*def changeSalary(self, sal):

self.\_\_class\_\_.salary = sal\*/ 🡪Not gonna do this way

@classmethod

def changeSalary(cls, sal):

cls.salary = sal

PROPERTY DECORATOR/ “getter METHOD”

@property

def totalSalary(self):

return self.salary + self.bonus

e = Employee()

print(e.totalSalary) // Calling it as a property not as a function🡪 ()

setter METHOD

@totalSalary.setter

def totalSalary(self, val):

self.salaryrbonus = val – self.salary

e = Employee()

e.totalSalary() //6000

e.totalSalary = 5800

print(e.salary) //5600

print(e.salarybonus) //200

Operator overloading… study

Dunder Methods.

except Exception as e:

//print(\_\_name\_\_)

if \_\_name\_\_ = "\_\_main\_\_":

global a = 🡪starts changing the global a.

enumeration:

list1 = [1, 10, 10.28, “Piyush”, true ]

//PRINTS ITEMS OF THE LIST WITH INDEX!

for index, item in enumerate(list1):

print(item, index)

List Comprehensions.

b = [i for i in a if a%2 == 0]

print(b)