Result = 104.123456

Print(“The result was {r:1.2f}”.format(r=result))

Output: The result was 104.12

Print(“The result was {r:1.5f}”.format(r=result))

Output: The result was 104.12345

Print(“The result was {r:10.5f}”.format(r=result))

Output: The result was 104.12345

‘pwd’ in jupyter notebook returns location of jupyter notebook

%%writefile myfile.txt #This is to write a text file from jupyter notebook

**Hello this is a text file**

**This is a second line**

**This is a third line** #These 3 lines will be saved inside ‘myfile.txt’ file.

Myfile.seek(0) #Resets the curser to 0

Myfile.readlines() #Gives each lines as an element in a list

with open(‘myfile.txt’) as my\_new\_file:

Contents = my\_new\_file.read()

with open(‘myfile.txt’, mode = ‘r’) as f:

print(f.read())

with open(‘myfile.txt’, mode = ‘w’) as f:

f.write(‘This is a fourth line’) #This line will be re-write in to the file

with open(‘myfile.txt’, mode = ‘a’) as f:

f.write(‘This is a fourth line’) #This line will be appended in the end

mylist1 = [1,2,3, 4,5,6]

mylist2 = [‘a’, ‘b’, ‘c’]



mylist3 = [100,200,300]



for items in zip(mylist1, mylist2, mylist3):

print(items) #zips together as far as the list which is shortest doesn’t give error

list(zip(mylist1, mylist2, mylist3))

word = ‘abcde’

for index, item in enumerate(word):

print(index,item)

from random import shuffle

mylist = [1,2,3,4,5,6,7,8,9,10]

shuffle(mylist)

from random import randint

randint(0,100)

|  |  |
| --- | --- |
| Mystr = ‘hello’  Mylist = []  for letter in Mystr:  Mylist.append(letter) | Mystr = ‘hello’  Mylist = []  Mystr = [letter for letter in Mystr] |

Mylist = [num for num in range(0,11)]

Mylist = [num\*\*2 for num in range(0,11)] #output: [0,1,4,16,36,49,64,81,100]

Mylist = [x for x in range(0,11) if x%2==0] #output: [0,2,4,6,8,10]

Mylist = [x\*\*2 for x in range(0,11) if x%2==0] #output: [0,4,16,36,64,100]

Celcius = [0,10,20,34,5]

Fahrenheit = [((9/5)\*temp + 32) for temp in celcius] #output: [32.0, 50.0,68.0, 94.1]

Results = [x if x%2==0 else ‘odd’ for x in range(0,11)] #output: [0,’odd’,2,’odd’,4,’odd’,6,’odd’,8,’odd’]

Mylist = [x\*y for x in [2,4,6] for y in [1,10,1000]] #output: [2,20,2000,4,40,4000,6,60,6000]

|  |
| --- |
|  |



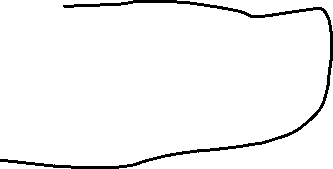
**Notes for oop python:**

class Dog():



def \_\_init\_\_(self, breed):

self.breed = breed ---- by convention we set same parameter name for all 3



my\_dog = Dog(breed = ‘Lab’)

my\_dog.breed

output: ‘Lab’

class Dog():

def \_\_init\_\_(self, mybreed):

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.breed = mybreed

my\_dog = Dog(mybreed = ‘Huskie)

my\_dog.breed

output: ‘Huskie’

class Dog():

def \_\_init\_\_(self, mybreed):

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.my\_attribute = mybreed

my\_dog = Dog(mybreed = ‘Huskie)

my\_dog.my\_attribute

output: ‘Huskie’

class Dog():



def \_\_init\_\_(self, breed, name, spots):

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.breed = breed

self.name = name

# Expect Boolean True/False

self.spots = spots

my\_dog = Dog(breed = ‘Lab’, name = ‘Sammy’, spots = False)

my\_dog.breed

my\_dog.name

my\_dog.spots

output: ‘Lab’, ‘Sammy’, False

class Dog():



# CLASS OBJECT ATTRIBUTE

# SAME FOR ANY INSTANCE OF A CLASS

species = ‘mammal’

def \_\_init\_\_(self, breed, name, spots):

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.breed = breed

self.name = name

# Expect Boolean True/False

self.spots = spots

my\_dog = Dog(breed = ‘Lab’, name = ‘Sam’, spots = False)

my\_dog.breed

my\_dog.name

my\_dog.spots

my\_dog.species

output: ‘Lab’, ‘Sam’, False, ’mammal’

class Dog():



# CLASS OBJECT ATTRIBUTE

# SAME FOR ANY INSTANCE OF A CLASS

species = ‘mammal’

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

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.breed = breed

self.name = name

# OPERATIONS/ACTIONS 🡪 Methods

Def bark(self):

Print(‘Woof’)

my\_dog = Dog(‘Lab’, ‘Frankie’)

my\_dog.breed

my\_dog.name

my\_dog.species

my\_dog.bark()

output: ‘Lab’, ‘Frankie’, ‘mammal’, ‘Woof’

class Dog():



# CLASS OBJECT ATTRIBUTE

# SAME FOR ANY INSTANCE OF A CLASS

species = ‘mammal’

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

# Attributed

# We take in the argument

# Assign it using self.attribute\_name

self.breed = breed

self.name = name

# OPERATIONS/ACTIONS 🡪 Methods

Def bark(self, number):

Print(‘Woof my name is {} and number is {}’.format(self.name, number))

my\_dog = Dog(‘Lab’, ‘Frankie’)

my\_dog.breed

my\_dog.name

my\_dog.species

my\_dog.bark(10)

output: ‘Lab’, ‘Frankie’, ‘mammal’, ‘Woof my name is Frankie and number is 10’

class Circle():

# CLASS OBJECT ATTRIBUTE

Pi = 3.14

Def \_\_init\_\_(self, radius=1):

Self.radius = radius

# METHOD

Def get circumference(self):

Return self.radius \* self.pi\*2

My\_circle = Circle(30)

My\_circle.pi

My\_circle.radius

My\_circle.get\_circumference()

Output: 3.14, 30, 188.4

class Circle():

# CLASS OBJECT ATTRIBUTE

Pi = 3.14

Def \_\_init\_\_(self, radius=1):

Self.radius = radius

Self.area = radius\*radius\*self.pi

# METHOD

Def get circumference(self):

Return self.radius \* self.pi\*2

My\_circle = Circle(30)

My\_circle.pi

My\_circle.radius

My\_circle.area

My\_circle.get\_circumference()

Output: 3.14, 30, 2826.0,188.4

class Circle():

# CLASS OBJECT ATTRIBUTE

Pi = 3.14

Def \_\_init\_\_(self, radius=1):

Self.radius = radius

Self.area = radius\*radius\*Circle.pi

# METHOD

Def get circumference(self):

Return self.radius \* Circle.pi\*2

My\_circle = Circle(30)

My\_circle.pi

My\_circle.radius

My\_circle.area

My\_circle.get\_circumference()

Output: 3.14, 30, 2826.0,188.4

Inheritance and polymorphism:

Inheritence:

class Animal():

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

print(“ANIMAL CREATED”)

def who\_am\_i(self):

print(“Im an animal”)

def eat(self):

print(“Im eating”)

myanimal.eat()

myanimal.who\_am\_i()

output: im eating, im an animal

class Animal():

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

print(“ANIMAL CREATED”)

def who\_am\_i(self):

print(“Im an animal”)

def eat(self):

print(“Im eating”)

class Dog(Animal):

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

Animal.\_\_init\_\_(self)

Print(“Dog Created”)

Def who\_am\_i(self):

Print(“im a dog”)

Def bark(self):

Print(“Woof”)

Mydog = Dog()

Mydog.who\_am\_i()

Mydog.eat()

Mydog.bark()

output: ANIMAL CREATED, Dog Created, im a dog, im eating, Woof

Polymorphism:

Class Dog():

Def \_\_inti\_\_(self, name):

Self.name = name

Def speak(self):

return self.name + “says woof!”

Class Dog():

Def \_\_inti\_\_(self, name):

Self.name = name

Def speak(self):

return self.name + “says meow!”

niko = Dog(“niko”)

felix = Cat(“felix”)

print(niko.speak())

print(felix.speak())

output: niko says woof!, felix says meow!

For pet\_class in [niko, felix]:

Print(type(pet))

Print(pet.speak())

Output: class main\_\_Dog, niko says woof!, class main\_\_Cat, felix says meow!

Def pet\_speak(pet):

Print(pet.speak())

Pet\_speak(niko)

Pet\_speak(felix)

Output: niko says woof!, felix says meow!

Class Animal():

Def \_\_init\_\_(self,name):

Self.name = name

Def speak(self):

Raise NotImplementedError(“Subclass nust implement this abstract method”)

Class Dog(Animal):

Def speak(self):

return self.name+ “ says woof!”

class Cat(Animal):

def speak(self):

return self.name+ “ says meow!”

fido = Dog(“Fido”)

isis = Cat(“Isis”)

print(fido.speak())

print(isis.speak())

output: Fido says woof!, Isis says meow!

Special objects:

Class Book():

Def \_\_inti\_\_(self, title, author, pages):

Self.title = title

Self.author = author

Self.pages = pages

Def \_\_str\_\_(self):

return f”{self.title} by {self.author}”

def \_\_len\_\_(self):

return self.pages

def \_\_del\_\_(self):

print(“A book object has been deleted”)

B = Book(“Python rocks”, ‘Jose’, 200)

Print(B)

Len()B

Output: “Python rocks by Jose”, 2009

Collection modules

From collection import Counter

Mylist = [1,1,1,1,1,2,2,2,2,3,3,3,3,3,3,3]

Counter(mylist)

Output: Counter({1:5, 2:4, 3:7})

Mylist = [“a”: 2, 10: 3]

Counter(mylist)

Output: Counter({‘a’: 2, 10: 3})

Counter(‘aaaabbbbshshsjs’)

Output: Counter({‘a’:4, ‘b’: 4, ‘h’: 2, ‘j’: 1, ‘s’: 4)

Sentence = ‘How many times does each word show up in this sentence with a word’

Counter(Sentence.lower().split())

Counter({‘a’: 1, ‘does’: 1, ‘each’: 1, ‘how’: 1, ‘in’: 1, ‘many’: 1, ‘sentence’: 1, ‘show’: 1, ‘this’: 1, ‘times’: 1, ‘up’: 1, ‘with’:1, ‘word’: 2})

Letters = ‘aaabbbbccccccccddddddddddd’

C = Counter(letter)

C

Output: Counter({‘a’: 3, ‘b’: 4, ‘c’: 8, ‘d’: 11})

c.most\_common()

output: [(‘d’, 11), (‘c’,8), (‘b’, 4), (‘a’, 3)]

c.most\_common(3)

output: [(‘d’, 11), (‘c’,8), (‘b’, 4)]

list(c)

output: [a,b,c,d]

from collections import defaultdict

d = {‘a’: 10}

d

output: {‘a’: 10}

d[‘a’]

output: 10

d = defaultdict(lambda:0)

d[‘correct’] = 100

d[‘correct’]

output: 100

d[‘wrong’]

output: 0

mytuple = (10,20,30)

from collections import namedtuple

Dog = namedtuple(‘Dog’, [‘age’, ‘breed’, ‘name’])

Sammy = Dog(age = 5, breed = ‘Husky’, name = ‘Sam’)

Sammy.age  
output: 5

Sammy.breed

Output: Husky

Sammy.name

Output: sam

Datetime Module

Import datetime

Mytime = datetime.time(2, 20)

Mytime.minute  
output: 20

Mytime.hour

Output: 2

Print(Mytime)

Output: 02:20:00

Mytime = datetime.time(2)

Print(Mytime)

Output: 02:00:00

Mytime = datetime.time(13, 20,1,20)

Print(Mytime)

Output: 13:20:01:000020

Mytime.microsecond

Output: 0

Type(mytime)

Output: datetime.time

Today = datetime.date.today()

Print(today)

Output: 2020-06-12

today.year

Output: 2020

today.month

output: 6

today.day

output: 12

today.ctime()

output: ‘Fri Jun 12 00:00:00 2020’

from datetime import datetime

mydatetime = datetime(2021, 10,3, 14, 20,1)

print(mydatetime)

output: 2021-10-03 14:20:01

mydatetime = mydatetime.replace(year = 2020)

print(mydatetime)

output: 2020-10-03 14:20:01

# DATE

from datetime import date

date1 = date(2021, 11, 3)

date2 = date(2020, 11,3)

result = date1 – date2

result.days

output: 365

datetime1 = datetime(2021,11,3,22,0)

datetime2 = datetime(2020,11,3,12,0)

datetime1 – datetime2

output: datetime.timedelta(365, 36000)

mydiff = datetime1-datetime2

mydiff.seconds

output: 36000

mydiff.total\_seconds()

output: 31572000.0

python math and random modules

import math

value = 4.35

math.ceil(value)

output: 5

math.floor(value)

output: 4

round(4.35)

output: 4

round(4.5)

output: 4

round(5.5)

output: 6

math.pi

output: 3.141592653589793

from math import pi

pi

output: 3.141592653589793

math.e

2.718281828459045

math.nan

Output: nan

math.inf

output: inf

math.log(math.e)

output: 1.0

math.log(100,10)

output: 2.0

math.sin(10)

output: -0.5440211108893698

Math.degrees(pi/2)

Output: 90.0

Math.radians(180)

Output: 3.141592653589793

Random modules

import random

random.randint(0, 100)

ouput: 22

random.seed(101)

random.int(0,100) # will get the same output every time I run it

output: 74

random.randint(0,100)

output: 24

random.seed(101)

print(random.randint(0,100))

print(random.randint(0,100))

output: 74, 24

mylist = list(range(0,20))

mylist

output: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]

random.choice(mylist) # chooses a random integer

output: 16

# SAMPLE WITH REPLACEMENT

Random.choices(population = mylist, k = 10)

output: [4,4,5,13,4,19,1,3,1,15]

# SAMPLE WITHOUT REPLACEMENT

Random.sample(population = mylist, k = 10)

Output: [11,6,15,10,7,16,12,18,13,3]

random.shuffle(mylist)

output: [12,7,19,11,0,3,17,8,15,4,5,18,16,10,1,6,9,14,13,2]

random.uniform(a=0, b=100) # will get random number, might even get floating number

output: 46.41054065279665

random.gauss(mu = 0, sigma = 1)

output: -0.8984857541998804

python debugger

notes wil be in jupyter