Akilesh K

k.akilesh123@gmail.com

Data engineering - Batch 1

Date: 30-01-24

DAY 7-PYTHON

Variables

```
In [1]: x = 5
        y = "John"
        print(x)
        print(y)
```

```
5
John
```

```
: a = int(3)
  z = float(3)
```

Datatypes

```
n = ["apple", "banana", "cherry"]
m = {"apple", "banana", "cherry"}
```

```
print(type(z))
print(10 > 9)
```

```
<class 'float'>
True
```

Operators

In [5]:
```
i = 5
j = 3
print(i * j)
print(i - j)
print(i / j)
print(i % j)
print(i ** j)
print(i // j)
```

```
15
2
1.6666666666666667
2
125
1
```

Dictionary

In [6]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

If Else

In [7]:
```
h = 200
k = 33
if h > k:
  print("h is greater than k")
elif h == k:
  print("h and k are equal")
else:
  print("h is greater than k")
```

```
h is greater than k
```

Break

In [8]:
```
l = 1
while l < 6:
  print(l)
  if l == 3:
    break
  l += 1
```

```
1
2
3
```

For loop

```
for f in range(2, 6):
  print(f)
```

```
2
3
4
5
```

Class

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

```
<__main__.Person object at 0x0000015DA5820990>
```

Inheritance

```
In [11]: class Student(Person):
           def __init__(self, fname, lname):
             super().__init__(fname, lname)
             self.graduationyear = 2019
```

Datetime

```
In [12]: import datetime

         k = datetime.datetime.now()

         print(k.year)
         print(k.strftime("%A"))
```

```
2024
Tuesday
```

Polymorphism

In [13]:
```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")
boat1 = Boat("Ibiza", "Touring 20")
plane1 = Plane("Boeing", "747")

for x in (car1, boat1, plane1):
  x.move()
```

```
Drive!
Sail!
Fly!
```

In [21]:
```
import sys

# importing sys.path
print(sys.path)
```

```
['C:\\Users\\Akilesh K', 'C:\\Users\\Akilesh K\\anaconda3\\python311.zip', 'C:\\Users\\Akilesh K\\anaconda3\\DLLs', 'C:\\Users
\\Akilesh K\\anaconda3\\Lib', 'C:\\Users\\Akilesh K\\anaconda3', '', 'C:\\Users\\Akilesh K\\anaconda3\\Lib\\site-packages',
'C:\\Users\\Akilesh K\\anaconda3\\Lib\\site-packages\\win32', 'C:\\Users\\Akilesh K\\anaconda3\\Lib\\site-packages\\win32\\li
b', 'C:\\Users\\Akilesh K\\anaconda3\\Lib\\site-packages\\Pythonwin']
```

Functions(If else)

In [15]:
```
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the function
evenOdd(2)
evenOdd(3)
```

```
even
odd
```

Keyword functions

In [16]:
```
def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='Hexa', lastname='Practice')
student(lastname='Practice', firstname='Hexa')
```

```
Hexa Practice
Hexa Practice
```

Arbitrary functions

n [17]:
```
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


print("Case-1:")
nameAge("Suraj", 27)

print("\nCase-2:")
nameAge(27, "Suraj")
```

```
Case-1:
Hi, I am Suraj
My age is  27

Case-2:
Hi, I am 27
My age is  Suraj
```

Lambda functions

In [18]:
```
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]

result = map(lambda x, y: x + y, numbers1, numbers2)
print(list(result))
```

```
[5, 7, 9]
```

Mapping

In [20]:
```
l = ['sat', 'bat', 'cat', 'mat']

# map() can listify the list of strings individually
test = list(map(list, l))
print(test)
```

```
[['s', 'a', 't'], ['b', 'a', 't'], ['c', 'a', 't'], ['m', 'a', 't']]
```

Operations in sets

In [13]:
```
# set of letters
s = {'g', 'e', 'k', 's'}

# adding 's'
s.add('f')
print('Set after updating:', s)

# Discarding element from the set
s.discard('g')
print('\nSet after updating:', s)

# Removing element from the set
s.remove('e')
print('\nSet after updating:', s)

# Popping elements from the set
print('\nPopped element', s.pop())
print('Set after updating:', s)

s.clear()
print('\nSet after updating:', s)
```

```
Set after updating: {'e', 's', 'g', 'f', 'k'}

Set after updating: {'e', 's', 'f', 'k'}

Set after updating: {'s', 'f', 'k'}

Popped element s
Set after updating: {'f', 'k'}

Set after updating: set()
```

Lambda

In [1]:
```
def cube(y):
    return y*y*y

lambda_cube = lambda y: y*y*y
print("Using function defined with `def` keyword, cube:", cube(5))
print("Using lambda function, cube:", lambda_cube(5))
```

```
Using function defined with `def` keyword, cube: 125
Using lambda function, cube: 125
```

In [2]:
```
format_numeric = lambda num: f"{num:e}" if isinstance(num, int) else f"{num:,.2f}"

print("Int formatting:", format_numeric(1000000))
print("float formatting:", format_numeric(999999.789541235))
```

```
Int formatting: 1.000000e+06
float formatting: 999,999.79
```

In [3]:
```
List = [[2,3,4],[1, 4, 16, 64],[3, 6, 9, 12]]

sortList = lambda x: (sorted(i) for i in x)
secondLargest = lambda x, f : [y[len(y)-2] for y in f(x)]
res = secondLargest(List, sortList)

print(res)
```

```
[3, 16, 9]
```

Class and calling object

In [4]:
```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")


print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))


print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

In [5]:
```
class Dog:


    attr1 = "mammal"


    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My name is {}".format(self.name))

# Driver code

Rodger = Dog("Rodger")
Tommy = Dog("Tommy")


Rodger.speak()
Tommy.speak()
```

encapsulation

```
    def display(self):
        print(self.name)
        print(self.idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post


        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))



a = Employee('Rahul', 886012, 200000, "Intern")


a.display()
a.details()
```

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

Polymorphism

In [7]:
```
class Bird:

    def intro(self):
        print("There are many types of birds.")

    def flight(self):
        print("Most of the birds can fly but some cannot.")

class sparrow(Bird):

    def flight(self):
        print("Sparrows can fly.")

class ostrich(Bird):

    def flight(self):
        print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

Function override

In [12]:
```
class Bird:
# constructor
    def __init__(self,name):
        self.name = name
    def print_info(self):
        print('This bird is :',self.name)
    def fly(self):
        print(' bird can fly')

class Shalik (Bird):
    def __init__(self, name, color, charater):
        super().__init__(name)
        self.color = color
        self.charater = charater
# override method
    def print_info(self):
        super().print_info()
        print('color of bird is :', self.color)
        print('Character of bird is :', self.charater)

# Override method
    def fly(self):
        print(' bird can fly')

obj_Shalik = Shalik ('Shalik', 'black', 'not good')
obj_Shalik.fly()
obj_Shalik.print_info()
```

```
 bird can fly
This bird is : Shalik
color of bird is : black
Character of bird is : not good
```

Exception handling

```
In [22]: try:
             k = 5//0
             print(k)

         except ZeroDivisionError:
             print("Can't divide by zero")

         finally:
             print('This is always executed')
```

```
Can't divide by zero
This is always executed
```

Module

```
In [14]: from math import sqrt, factorial

         print(sqrt(16))
         print(factorial(6))
```

```
4.0
720
```

Module function

In [23]:

```
import math


print(math.sqrt(25))

e
print(math.pi)


print(math.degrees(2))


print(math.radians(60))

# Sine of 2 radians
print(math.sin(2))


print(math.cos(0.5))


print(math.tan(0.23))


print(math.factorial(4))


import random


print(random.randint(0, 5))


print(random.random())

print(random.random() * 100)
```

```
5.0
3.141592653589793
114.59155902616465
1.0471975511965976
0.9092974268256817
0.8775825618903728
0.23414336235146527
24
0
0.9314989204182499
52.16738956990261
True
1706520934.6687574
1970-01-06
```