Introduction to Python

Python

- Python is a high level, general purpose programming language.
- Python is a dynamic, type-checked and garbage collected.
- It supports object oriented programming and functional programming.
- Python is a easy to learn language as it emphasise code readability, and uses indentation to specify blocks.

Python

- Python 2.0 was released in October 2000.
- Python 3.0 was released in December 2008.
- Python 3.0 was not completely backward compatible.
- Python was proposed by Guido Van Rossum in the late 1980s.
- The name “Python” came from the word “Monty Python’s Circus”.

Variables and DataTypes

- Python is a dynamically typed language so it is not necessary to specify the datatypes of the variables.
- There are different types of datatypes they are:
  - Numbers : integers, floats and complex numbers.
  - List : List is a mutable collection of ordered items.
  - Tuple: Tuple is a immutable collection of ordered items.
  - Set: Set is an mutable unorder collection of items.
  - Dictionary: Dictionary is a mutable collection of key-value pairs.
  - Boolean: contains True of False.
  - String: String is a immutable collection of characters.

Operators

- Operators are used to perform operations on variables and values.
- There are different types of operators they are:
  - Assignment Operator
  - Arithmetic Operator
  - Comparison Operator
  - Logical Operator
  - Bitwise Operator
  - Membership Operator
  - Identity Operator

Comments

- Comments are ignored by the interpreter.
- Comments are used to explain the code and make it more readable
- Comments begin with ‘#’ in python or use “ ’’’ ” triple quote.

```
# This is a comment in python

‘‘‘
This is a
Multi line comment
’’’
```

Conditional Statement

- Conditional statements are used to change to flow of the program based on some conditons.
- There are different types of conditional statements they are:
  - If else
  - If elif else
  - Nested if else
  - Match statement

If Statements

```
a = 10
if (a == 10):
   print(“a is 10”)
elif (a == 20):
   print(“a is 20”)
else:
   print(“a is neither 10 nor 20”)

# Output:
# a is 10
```

Match Statements

```
a = 10
match a:
    case 10:
        print(“a is 10”)
    case 20:
        print(“a is 20”)
    case default:
        print(“a is neither 10 nor 20”)
# Output:
# a is 10
```

Loop Statement

- Looping statements are used to repeat a code for a particular number of times
- There are different types of Looping statements they are:
  - While Loop.
  - For Loop.
- There are loop control statements that alter the execution of looping statement execution.
  - Continue statement.
  - Break statement.

While Statement

```
a = 10
match a:
   case 10:
      print(“a is 10”)
   case 20:
      print(“a is 20”)
   case default:
      print(“a is neither 10 nor 20”)
# Output:
# a is 10
```

For Statement

```
for i in range(3):
  print(i, end=‘, ’)
print()
lst = [10,20,30]
for i in lst:
     print(i, end=‘, ’)
# Output:
# 0, 1, 2
#
# 10, 20, 30
```

Break Statement

```
a = 3
for i in range(5): # [0,1,2,3,4]
    if i == a:
        break
    print(i)

# Output:
# 0
# 1
# 2
```

Continue Statement

```
a = 2
for i in range(4): # [0,1,2,3]
    if i == a:
        break
    print(i)

# Output:
# 0
# 1
# 3
```

Introduction to OOP

- Object Oriented Programming is a programming paradigm, in which we use “objects” to model software structure.
- To understand OOP we need to basic concepts like:
  - Classes
  - Objects
  - Polymorphism
  - Encapsulation
  - Inheritance
  - Abstraction

Classes

- Classes are blueprint to create an object.

```
class myClass:
    def __init__(self, myVar):
        self.myVar = myVar
    def myMethod(self):
        print(self.myVar)
myClassObj = myClass(30) # Creates an instance
myClassObj.myMethod()    # 30
myClassObj.myVar = 20
myClassObj.myMethod()    # 20
```

Object

- Object is an instance of a class

```
myClassObj = myClass(30) # Creates an instance
myClassObj2 = myClass(30)# Creates a 2nd instance
myClassObj2.myMethod()   # print: 30

myClassObj.myVar = 10000 # Change myVar

myClassObj2.myMethod()   # print: 1000
myClassObj.myMethod()    # print: 30
```

Inheritance

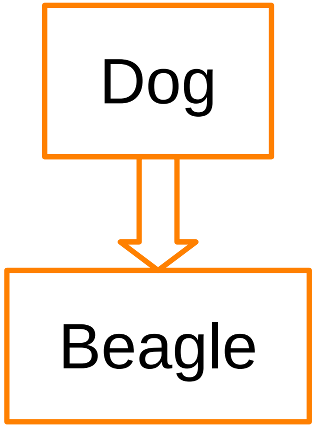
- Inheritance allows a class to acquire the properties and methods of the parent class to the child class, there are different types of inheritance they are:
  - Single level
  - Multi level
  - Multiple
  - Hierarchical
  - Hybrid

Types of inheritance

- **Single Inheritance:**
  - A class inherits from one parent class.
- **Multi-Level Inheritance:**
  - A class inherits from a parent class which in-turn again inherits from another parent class.
- **Multiple Inheritance:**
  - A class inherits from more than a single parent class, and inherits from two parent class.
- **Hierarchical Inheritance:**
  - Multiple child class inherits from a single parent class.
- **Hybrid Inheritance:**
  - A combination of one or two types of inheritance.

Single Inheritance

```
class Dog:              # Parent class
   def makeSound(self):
      print(“Woof!”)

class Beagle(Dog):  # Child class
   def __init__(self, name):
      self.name = name
   def makeSound(self):
      print(self.name, “says beagle Woff!”)
```



Single Inheritance

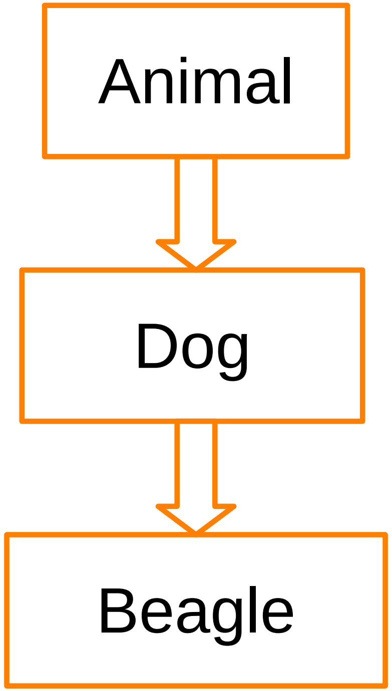
```
genericDog = Dog()
genericDog.makeSound()

jimmy = Beagle(“Jimmy”)
jimmy.makeSound()

# Output:
# Woof!
# Jimmy says beagle Woff!
```

Multi-Level Inheritance

```
class Animal:            # Grand Parent class
  def makeSound(self):
    print(“......”)
class Dog(Animal):       # Parent class
  def makeSound(self):
    print(“Woof!”)
class Beagle(Dog):       # Child class
  def __init__(self, name):
    self.name = name
  def makeSound(self):
    print(self.name, “says beagle Woff!”)
```

Multi-Level Inheritance

```
animal = Animal()
animal.makeSound()
genericDog = Dog()
genericDog.makeSound()
jimmy = Beagle(“Jimmy”)
Jimmy.makeSound()
# Output:
# ……
# Woof!
# Jimmy says beagle Woff!
```

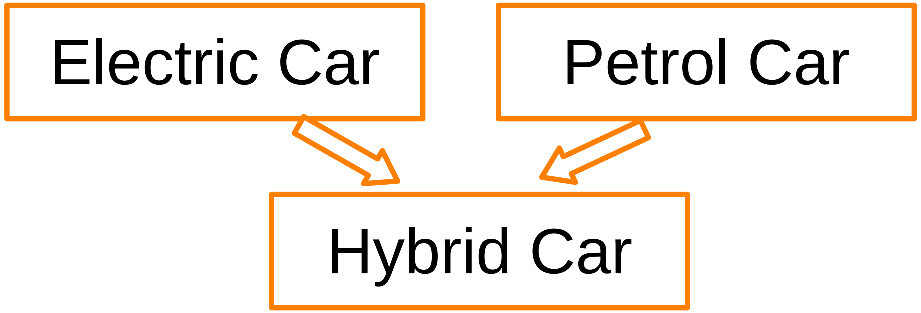


Animal

Dog

Beagle

Multiple Inheritance



```
class PetrolCar:
   fuelCapacity = 0;
class ElectricCar:
   chargeCapacity = 0;
class HybridCar(PetrolCar, ElectricCar):
   def __init__(self, fuelCap, chargeCap):
      self.fuelCapacity = fuelCap
      self.chargeCapacity = chargeCap
   def printCapacity(self):
      print(“fuelCapacity:”,self.fuelCapacity,“\n”
, ”\belectricCapacity:”, self.chargeCapacity)
```

Multiple Inheritance

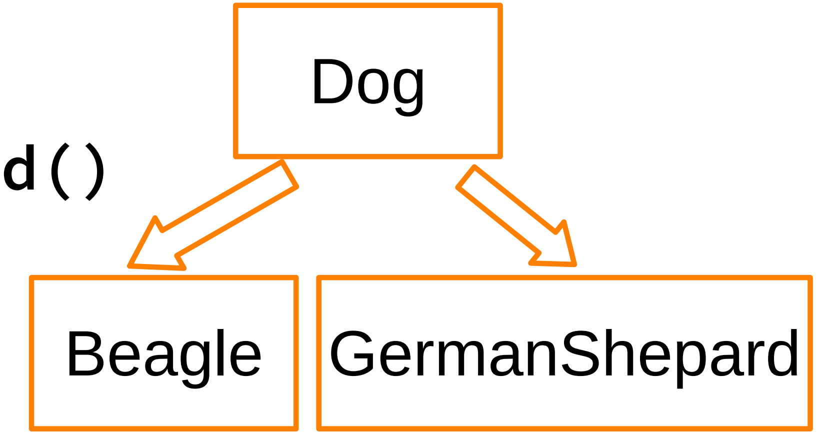
```
pc = PetrolCar()
ec = ElectricCar()
hc = HybridCar(90, 80)
print(“Fuel Capacity:”,pc.fuelCapacity)
print(“Charge Capacity:”,ec.chargeCapacity)
hc.printCapacity()
 # Output:
 # Fuel Capacity: 0
 # Charge Capacity: 0
 # fuelCapacity: 90
 # chargeCapacity: 80
```

Hierarchical Inheritance

```
class Dog:                 # Parent class
    def makeSound(self):
        print(“Woof!”)
class Beagle(Dog):         # Child 1 class
    def __init__(self, name):
        self.name = name
    def makeSound(self):
        print(self.name, “says beagle Woff!”)
class GermanShepard(Dog):  # Child 2 class
    def makeSound(self):
        print(“German Shepard Woff!”)
```

Hierarchical Inheritance

```
genericDog = Dog()
genericDog.makeSound()
jimmy = Beagle(“Jimmy”)
jimmy.makeSound()
germanShepard = GermanShepard()
germanShepard.makeSound()
# Output:
# Woof!
# Jimmy says beagle Woff!
# German Shepard Woff!
```



Dog

Beagle

GermanShepard

Encapsulation

- Encapsulation restricts the access of variables and methods to prevent unwanted change or modification of values.
- Encapsulation enables to reduce code coupling and lets us change the implementation without impacting code segments.
- The use encapsulation in python we can have two ways:
  - Private methods
  - Private variables
- To create private variables or methods we simply put two underscores in-front of the identifiers Eg: __name, __method().

Encapsulation

```
class Car:
   __name = “”
   def __init__(self):
      self.__name = “Toyota car”
   def printCar(self):
      print(self.__name)
   def setName(self, name):
      self.__name = name
   def __printPrivate(self):
      print(“private”)
```

Encapsulation

```
car1 = Car()
car1.printCar()
car1.setName(“Innova Car”)
car1.printCar()
# car1.__printPrivate() # Error cannot access
car1._Car__printPrivate()
# Output:
# Toyota Car
# Innova Car
# private
```

Polymorphism

- The ‘poly’ means **many** ‘morphism’ means **forms**.
- It is achieved in python by allowing same methods to have different functionality based on the object type.
- It uses dynamic typing and duck typing, there are two types:
- **Method Overriding:**
  - Same Method name but different parameters.
- **Method overloading:**
  - Same Method name and signature but different implementations based on class the method is called from.

Method Overriding

```
class Animal:            # Grand Parent class
    def makeSound(self):
        print(“……”)
class Dog(Animal):       # Parent class
    def makeSound(self):
        print(“Woof!”)
class Beagle(Dog):       # Child class
    def makeSound(self):
        print(“Beagle Woff!”)
```

Method Overriding

```
animal = Animal()
animal.makeSound()
genericDog = Dog()
genericDog.makeSound()
beagleDog = Beagle()
beagleDog.makeSound()
# Output:
# ……
# Woof!
# Beagle Woff!
```

Method Overloading

```python
class Example:
    def display(self, a=None, b=None):
        if a is not None and b is not None:
            print(a + b)
        elif a is not None:
            print(a)
        else:
            print("Nothing to display")
```

Method Overloading

```
obj = Example()
obj.display()
obj.display(10)
obj.display(10, 20)

# Output:
# Nothing to display
# 10
# 30
```

File Handling

- File Handling is the process of reading or writing data to a file stored in the hard disk.
- There are three main operations in file handling:
  - Opening a file
  - Reading/Writing to a file
  - Closing a file
- While opening a file we need to specify the modes to open the file:
- Eg: `r` – Read, `w` – Write, `rb` – Read Binary, `wb` – Write Binary

File Handling

```
f = open(“test.txt”, “w”)        # Open
f.writeLines([“Hello, World!”,])# Operation
f.close()                        # Close
f = open(“test.txt”, “r”)        # Open
for i in f.readlines():          # Operation
  print(i)
f.close()                        # Close

# Output:
# Hello, World!
```

NumPy

```
import numpy as np

arr = np.array([1,2,3,4])

print(arr)
print(type(arr))

# Output:
# [1,2,3,4]
# <class, ‘numpy.ndarray’>
```

Pandas

```
import pandas as pd
Mydata = {
  ‘cars’: [“Toyota”, “Tata”],
  ‘passengers’ [5, 4]
}
formatedData = pd.DataFrame(myData)
print(formatedData)
# Output:   cars    passengers
#         0 Toyota  5
#         1 Tata    4
```

Programs

- Given number is prime or not.
- Given number is palindrome or not.
- Given number is amstrong or not.
- Find the factorial of a number.
- Find the fibonacci sequence for the number.
- Find the GCD of two numbers.