Lecture-7

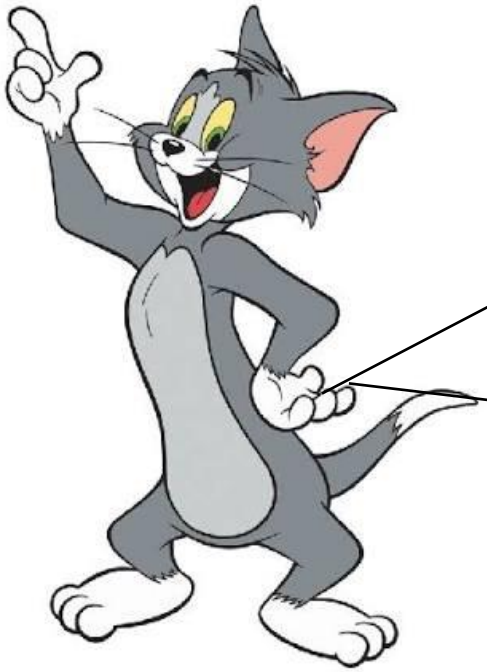
OOP Concepts with Python

Content

- Defines class and object
- Encapsulation and data hiding
- Controlling access to attributes
- Properties for data access

What is Object

Object have attributes (variables) and behaviour (Methods/Function)

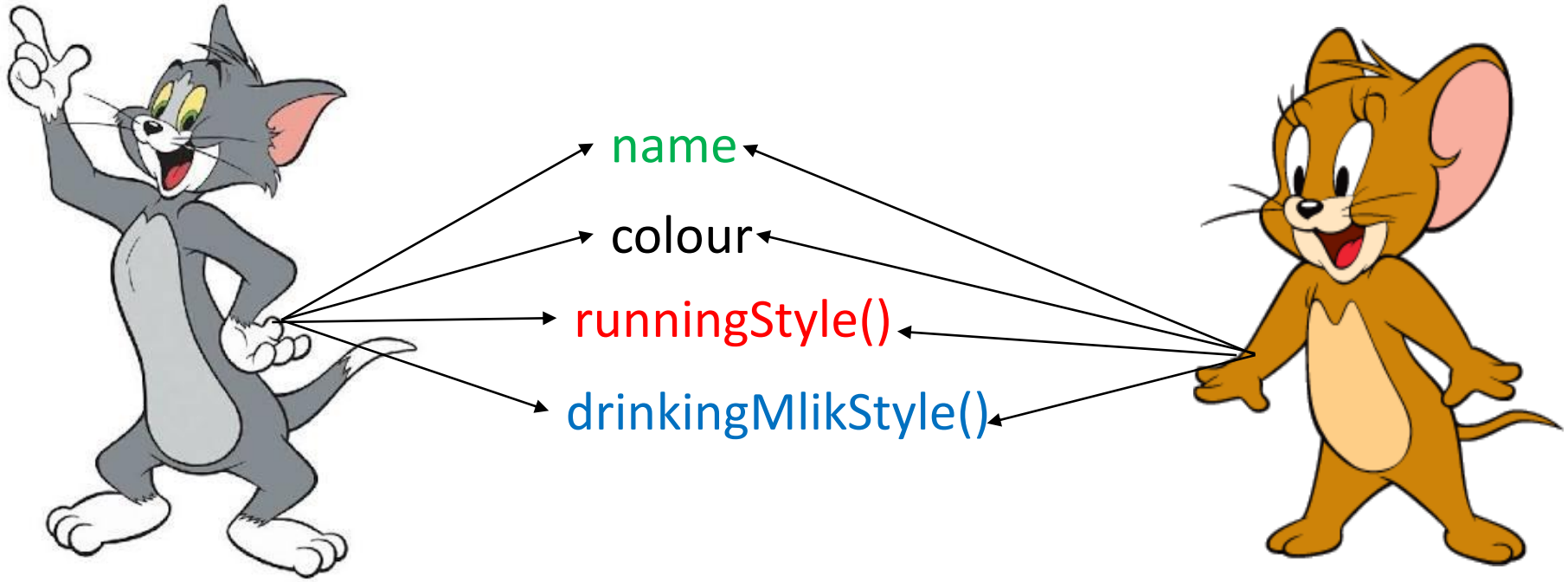


Attributes are: name , colour etc.

Behavior : running, drinking etc.

What is class?

A class is blue print form which individual object are created



Syntax of Creating Python Class

```
class NameOfClass():
  def __init__(self, parameter1, parameter2):
    self.parameter1 = parameter1
    self.parameter2 = parameter2

  def someMethod(self):
    #perfrom some action
    print(self.parameter1)
```

Create Person Class

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

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Output: John
36

The Self Parameter

- The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

- It does not have to be named self , you can call it whatever you like, but it has to be **the first parameter of any function in the class**. *[it work like get() method in java]*

The __init__ Function

- The __**init**()__ function is called automatically every time the class is being used to create a new object.

- All classes have a function called __**init**()__, which is always executed when the class is being initiated. *[it work like constructor in java]*

Create Object

```
tom=Animal('tom', 'gray')
jerry=Animal('jerry', 'browon')
```

Understanding Object Attribute and Behavior

```
class Animal:
  def __init__(self,name,color):
    self.name = name
    self.color = color

  def running(self):
    print(f'hey! i am {self.name}, now i am running!')
  def drinkingMilk(self):
    print(f'hey! i am {self.name}, now i am drinking mlik!')


tom=Animal('tom','gray')
jerry=Animal('jerry','brown')
tom.running()
jerry.running()
```

Encapsulation and Data Hiding

- Encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition.
- A class is an example of encapsulation as it encapsulates all the data that is member functions, variables etc.
- In Python, all data attributes are accessible. You use attribute naming conventions to indicate that attributes should not be accessed directly from client code.

Encapsulation (Public)

- Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method.
- All members in a Python class are **public** by default. Any member can be accessed from outside the class environment.

Encapsulation (Protected)

- Protected members of a class are accessible from within the class and are also available to its sub-classes. No other environment is permitted access to it. This enables specific resources of the parent class to be inherited by the child class.
- Python's convention to make an instance variable **protected** is to add a prefix _ (single underscore) to it. This effectively prevents it to be accessed, unless it is from within a sub-class.

Encapsulation (Private)

- Python doesn't have any mechanism that effectively restricts access to any instance variable or method. Python prescribes a convention of prefixing the name of the variable/method with single or double underscore to emulate the behavior of protected and private access specifiers.
- A double underscore __ prefixed to a variable makes it **private**.

Example of Protected and Private

```
class Encap:
    def __init__(self):
        # Protected member
        self._a = 2

        # Private member
        self.__b = 5
```

Thank You