U-Tech®

U-TECH DIGITAL EDUCATION

Python Classes

Objects and Attributes

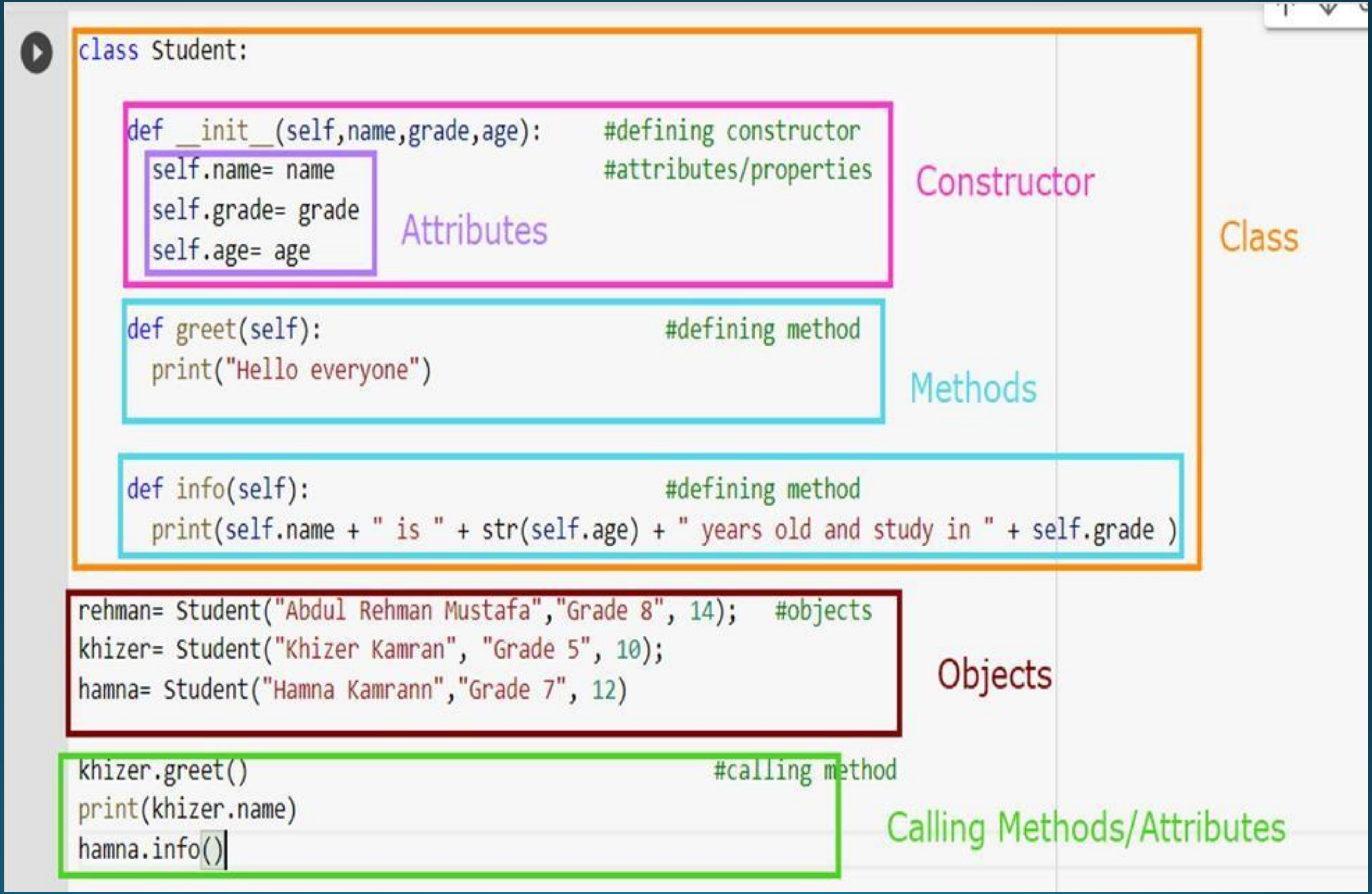
Topic 6

Object Oriented Programming - OOPs

- Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stresses on objects.
- An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.
- We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.
- As many houses can be made from a house's blueprint, we can create many objects from a class.
- An object is also called an instance of a class and the process of creating this object is called instantiation.

DESCRIPTION:

- Class
- Objects
- Methods/Functions
- Constructor (objects initializer method)
- Attributes



```python
class Student:

    def __init__(self,name,grade,age):      #defining constructor
      self.name= name                       #attributes/properties
      self.grade= grade
      self.age= age

    def greet(self):                                  #defining method
      print("Hello everyone")

    def info(self):                                   #defining method
      print(self.name + " is " + str(self.age) + " years old and study in " + self.grade )

rehman= Student("Abdul Rehman Mustafa","Grade 8", 14);    #objects
khizer= Student("Khizer Kamran", "Grade 5", 10);
hamna= Student("Hamna Kamrann","Grade 7", 12)

khizer.greet()                                         #calling method
print(khizer.name)
hamna.info()
```

DESCRIPTION:

- Class
- Objects
- Methods/Functions
- Constructor (objects initializer method)
- Attributes

CLASS

- Like function definitions begin with the def keyword in Python, class definitions begin with a class keyword.
- The first string inside the class is called docstring and has a brief description about the class. Although not mandatory, this is highly recommended.

SYNTAX:

```
class ClassName:
    '''This is a docstring. I have created a new class'''
    pass
```

DESCRIPTION:

- Class
- Objects
- Methods/Functions
- Constructor (objects initializer method)
- Attributes

OBJECT

- Object could be used to access different attributes/properties.
- It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.

SYNTAX:

```
>>> objectnamme = ClassName()
```

DESCRIPTION:

- Class
- Objects
- Methods/Functions
- Constructor (objects initializer method)
- Attributes

ATTRIBUTE:

- Any variable that is stored/ bound in a class is a class attribute.
- As an object-oriented language, Python provides two scopes for attributes: class attributes and instance attributes.

❑ An instance attribute is a Python variable and is defined inside the constructor function, __init__(self,..) of the class.

❑ A class attribute is a Python variable and is shared between all the objects of this class and it is defined outside the constructor function, __init__(self,...), of the class.

SYNTAX:

```
class ExampleClass(object):
    class_attr = 0

    def __init__(self, instance_attr):
        self.instance_attr = instance_attr
```

def_init_(self) # body of the constructor

DESCRIPTION:

- Class
- Objects
- Methods/Functions
- Constructor (objects initializer method)
- Attributes

CONSTRUCTOR:

- Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created.
- In Python the__init__() method is called the constructor and is always called when an object is created.

SYNTAX:

```
class ExampleClass(object):
    def__init__(self, attribute1, attribute2):
            # body of the constructor
```

ADDING ATTRIBUTES TO A CLASS

- Inside the class, an __init__ function has to be defined with def.
- __init__ must always be present! It takes one argument: self, which refers to the object itself.
- Inside the function, the pass keyword is used as of now, because Python expects you to type something there. Remember to use correct indentation!
- Inside the function, when you type some attributes/properties, the pass keyword is not used. Remember to use correct indentation!
- The attributes you make inside the function is also added with the arguments of function.
- To access an object's attributes in Python, you can use the dot notation. This is done by typing the name of the object, followed by a dot and the attribute's name.

Syntax:

```
Class ClassName:

  def__init__(self, attribute1, attribute2):         #defining__init__function
    self.attribute1 = attribute1                       #initilaize
     self.attribute2 = attribute2                      attributes/properties

#creating objects with arguments
Object1 = ClassName(value for attribute1, value for attribute2)
Object2 = ClassName(value for attribute1, value for attribute2)
```

METHODS:

- Functions inside the class are called Methods.
- These methods can be defined using “def” keyword with “self” as the necessary argument.
- The mandatory method should be named with__init__and is called “Constructor”. It is used to initialize the objects.
- You can create as many methods inside a class as you want.

Syntax:

```
class ClassName:

   #defining constructor
   def__init__(self,attribute1,attribute2,attribute3):
      self.attribute1=attribute1
      self.attribute2=attribute2
      self.attribute3=attribute3

   def MethodName(self):     #defining method
      ---body of method----

#creating objects
Object1= ClassName(value for attribute1, value for attribute2, value for attribute3);


 Object1.MethodName()        #calling method
```

PASSING ARGUMENTS TO METHODS

```
class ClassName:

   #defining constructor
   def__init__(self,attribute1,attribute2,attribute3):
        self.attribute1= attribute1
         self.attribute2= attribute2
         self.attribute3= attribute3

   def MethodName(self, argument1):    #defining method
        ---body of method----

    def MethodName(self, argument1, argument2, argument3): #defining method
         ---body of method----

#creating objects
Object1= ClassName(value for attribute1, value for attribute2, value for attribute3);


Object1.MethodName(value of argument1)       #calling method
Object1.MethodName(value of argument1, value of argument2, value of argument3)
```