


ABDELGHAFOR'S VIRTUAL INTERNSHIP

PYTHON PROGRAM

SESSION (4)

PREPARED BY : MARK KOSTANTINE



AGENDA OVERVIEW

03

CLASSES AND OBJECTS

21

QUESTIONS

09

INHERITANCE

15

POLYMORPHISM

18

TASKS



CLASSES & OBJECTS

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

CREATE A CLASS AND OBJECT

To create a class, use the keyword `class`

```
class MyClass:  
    x = 5
```

Now we can use the class named MyClass to create objects

```
p1 = MyClass()  
print(p1.x)
```

THE `__init__()` FUNCTION

- To understand the meaning of classes we have to understand the built-in `__init__()` function.
- All classes have a function called `__init__()`, which is always executed when the class is being initiated.
- Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created

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

p1 = Person("John", 36)

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

Note: The `__init__()` function is called automatically every time the class is being used to create a new object.

THE `__STR__`() FUNCTION

- The `__str__()` function controls what should be returned when the class object is represented as a string.
- If the `__str__()` function is not set, the string representation of the object is returned

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

    def __str__(self):
        return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

OBJECT METHODS

Objects can also contain methods. Methods in objects are functions that belong to the object.

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

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

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

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

```
    def myfunc(abc):  
        print("Hello my name is " + abc.name)
```

```
p1 = Person("John", 36)  
p1.myfunc()
```

MODIFY OBJECT PROPERTIES

You can modify properties on objects like this

```
p1.age = 40
```

DELETE OBJECT PROPERTIES

You can delete properties on objects by using the `del` keyword

```
del p1.age
```

DELETE OBJECTS

You can delete objects by using the `del` keyword

```
del p1
```




INHERITANCE

Inheritance allows us to define a class that inherits all the methods and properties from another class.

- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.

CREATE A PARENT CLASS

Any class can be a parent class, so the syntax is the same as creating **any other class**



CREATE A CHILD CLASS

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class

```
class Student(Person):  
    pass
```

Note: Use the `pass` keyword when you do not want to add any other properties or methods to the class. Now the Student class has the same properties and methods as the Person class.

```
x = Student("Mike", "Olsen")  
x.printname()
```

ADD THE `__init__()` FUNCTION

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the `__init__()` function to the child class (**instead of the `pass` keyword**).

Note : The `__init__()` function is called automatically every time the class is being used to create a new object.

```
class Student(Person):
    def __init__(self, fname, lname):
        #add properties etc.
```

Note: The child's `__init__()` function **overrides** the inheritance of the parent's `__init__()` function.

To keep the inheritance of the parent's `__init__()` function, add **a call to the parent's `__init__()` function**

```
class Student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)
```

USE THE SUPER() FUNCTION

Python also has a `super()` function that will make the child class inherit all the methods and properties from its parent

```
class Student(Person):  
    def __init__(self, fname, lname):  
        super().__init__(fname, lname)
```

By using the `super()` function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

ADD PROPERTIES

Add a property called `graduationyear` to the Student class

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

Add a `year` parameter, and pass the correct year when creating objects

```
class Student(Person):  
    def __init__(self, fname, lname, year):  
        super().__init__(fname, lname)  
        self.graduationyear = year
```

```
x = Student("Mike", "Olsen", 2019)
```



ADD METHODS

Add a method called `welcome` to the Student class

```
class Student(Person):
    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year

    def welcome(self):
        print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)
```



POLYMORPHISM

The word "**polymorphism**" means "**many forms**", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

FUNCTION POLYMORPHISM

An example of a Python function that can be used on different objects is the `len()` function.

```
x = "Hello World!"
```

```
print(len(x))
```

```
mytuple = ("apple", "banana", "cherry")
```

```
print(len(mytuple))
```

CLASS POLYMORPHISM

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name

```
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")      #Create a Car class
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat class
plane1 = Plane("Boeing", "747")    #Create a Plane class

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


INHERITANCE CLASS POLYMORPHISM

What about classes with child classes with the same name?
Can we use polymorphism there?

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

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

class Car(Vehicle):
    pass

class Boat(Vehicle):
    def move(self):
        print("Sail!")

class Plane(Vehicle):
    def move(self):
        print("Fly!")

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

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

TASKS

Beginner Level :

- **Class and Object Relationships**

- Create a class Animal with attributes species and sound. Define a method make_sound to print the sound. Then, create subclasses Dog and Cat that inherit from Animal and set different sounds. Instantiate these objects and call their methods.

- **Using Inheritance and the Super() Function**

- Create a base class Person with attributes name and age, and a method display_info. Create a subclass Employee that inherits from Person and adds an attribute salary. Use super() in the Employee class to initialize the base class's attributes. Create objects and display their information.

- **Basic Polymorphism with Classes**

- Create a class Vehicle with a method start_engine that prints "Engine started." Create two subclasses Car and Boat, each with its own version of start_engine. Demonstrate polymorphism by calling start_engine from objects of Car and Boat.



TASKS

Intermediate Level :

- **Class Polymorphism in Action**

- Create a base class Media with a method play. Then, create subclasses Audio and Video, each with a different implementation of the play method. Write a function that accepts an object of type Media and calls its play method, demonstrating polymorphism.

- **Inheritance and Method Overriding with Super**

- Create a class Building with a method build that prints "Building foundation." Create a subclass House that overrides build to print "Building a house," but also calls the parent class's build method using super(). Instantiate a House object and call its build method.

- **Polymorphism with Inheritance**

- Create a base class Shape with an abstract method area. Create two subclasses Rectangle and Circle, overriding the area method to return the area of a rectangle and a circle, respectively. Demonstrate polymorphism by creating objects of Rectangle and Circle and calling their area methods.



TASKS

Advanced Level :

- **Polymorphism through Function Overloading**
 - Create a class Calculator with methods to add two integers, two floating-point numbers, and two strings. Use method overloading by checking the types of inputs and defining separate logic for each type within the same add method. Test it by creating a Calculator object and using the add method with different types.
- **Inheritance with Multiple Levels of Polymorphism**
 - Create a base class Device with a method start. Then, create subclasses Computer and Phone, each with its own start method. Further, create subclasses Laptop (from Computer) and Smartphone (from Phone). Demonstrate polymorphism by calling the start method on objects of Laptop and Smartphone.
- **Class Polymorphism with Real-World Example**
 - Create a class PaymentMethod with a method process_payment. Then, create subclasses CreditCardPayment and PaypalPayment that override process_payment with their specific implementations. Write a function that accepts any PaymentMethod object and processes the payment, demonstrating polymorphism.



ANY QUESTIONS ?



PYTHON PROGRAM

THANK YOU

UPCOMING NEXT WEEK : SESSION (5)