# AIChampionsHub : Advanced Python Course

## Lesson on : Object Oriented Programming

OOPs is a important Programming Paradigm. In this you will learn about the basic building blocks of OOPs based programming using Python

- Classes
- Attributes
- Methods
- Objects and Instantiation
- Workout Examples have playing with Objects and Classes

## Example: Imagine you are Manager of a Zoo. There are various types of Animals. We will use this example to illustrate OOPs for an Application for the Zoo

## Step 1 : # Define a Animal Class

- Class has a name
- use the 'class' keyword to define a class.
- End with ':'

```
class Animal:
      # Attributes
      # Methods or Functions
```

## Step 2 : Add Details to the class

- Attributes: Say the Zoo manager wants to track name, age and few details of the Animals
- Methods: Second, as the animal changes, he will want to model the behavior of the animal. example as animal grows we want to update his age. Implement this as a method of the class.

```
# TEMPLATE OF AN ANIMAL CLASS as EXAMPLE
class Animal:
    def __init__(self, name, type, age):
        self.name = name
        self.type = type  # Type of Animal
        self.age = age
        print(f"Animal Instantiated with a name of : {self.name}")

    def makesound(self):
        print("Hello this is my sound")

    def updateAge(self, age):
        self.age = self.age + 2 # Change this to illustrate concept
        print(f"{self.name} is {self.age} years old")
```

## Step 4 : Objects

Objects are instances of the Class. They are specific Instances of concept called "Animal".

- In this case we create a specific Cat and a Dog that got admitted to the Zoo

```
# Instantiate Objects
mycat = Animal("shiro","cat","7")
```

```
Animal Instantiated with a name of : shiro
```

```
# Instantiate Objects
mydog = Animal("pyro","dog","5")
```

```
Animal Instantiated with a name of : pyro
```

```
print(mycat) # Look at the Address or type etc.
print(mydog)
```

```
<__main__.Animal object at 0x7dec10155dd0>
<__main__.Animal object at 0x7dec1010d490>
```

```
mycat == mydog # They are different Objects
```

```
False
```

## Step 5 : Object Manipulation

- You can access Details of an Object based on its attributes i.e. the current value of the Attributes.
- Second you can invoke (or call) the methods of the class to perform actions

```
mydog.name, mydog.age
```

```
('pyro', '5')
```

- Before we instantiate, to perform actions like Updating the Age we want to make sure the Age Attribute is of proper datatype (Integer). This is one example. So we update the class with those details.
- When you change the definition of the Class (like UpdateAge or Age Datatype) then you need to instantiate the objects again

```
class Animal:

    def __init__(self, name, type, age):
        self.name = name
        self.type = type  # Type of Animal
        self.age = int(age)                         # Force the DataType to be Integer
        print(f"Animal Instantiated with a name of : {self.name}")

    def makesound(self):
        print("Hello this is my sound")

    def updateAge(self):
        self.age = self.age + 2                    # Update Rules or Logic
        print(f"Hi ! {self.name} is {self.age} years old")
```

Double-click (or enter) to edit

```
# Instantiate Objects
mycat = Animal("Bobby","cat","7")
```

```
Animal Instantiated with a name of : Bobby
```

```
mycat.updateAge()
```

```
Hi ! Bobby is 9 years old
```

```
mycat.name
```

```
'Bobby'
```

## ABSTRACT BASE CLASS : INHERITANCE

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def updateAge(self):
        pass
```

```
    def defineColor(self):
        print("Message from Parent Class : Specificy Color")


class Bird(Animal):
  def __init__(self, name, type, age):
        self.name = name
        self.type = type  # Type of Animal
        self.age = age
        print(f"Bird Instantiated with a name of : {self.name}")

  def updateAge(self, age):
        self.age = self.age + 1 # Change this to illustrate concept
        print(f"{self.name} is {self.age} years old")
```

## Observe that you cannot instantiate without implementing Abstract Method

```
b1 = Bird("Vanakilli","parrot","1")
```

```
Bird Instantiated with a name of : Vanakilli
```

## Now Implement Methods in Parent Class and Use both Inherited Class as well as Parent Class

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def updateAge(self):
        pass
    def setColor(self):
        print("Message from Parent Class : Specificy Color")
```

## Note the use of super() to access method of the Parent class from which we are inheriting

```
class Bird(Animal):
  def __init__(self, name, type, age):
        self.name = name
        self.type = type  # Type of Animal
        self.age = age
        print(f"Bird Instantiated with a name of : {self.name}")

  def updateAge(self, age):
        self.age = self.age + 1 # Change this to illustrate concept
        print(f"{self.name} is {self.age} years old")

  def setColor(self, color):
        self.color = color
        super().setColor()            #Observe that Method of the Parent Class is also called
        print(f"Message from Bird Class ")
        print(f"Color of Bird is  {self.name} is {self.color}")


class Dog(Animal):
  pass


b2 = Bird("New Sparrow","sparrow","1")
```

```
Bird Instantiated with a name of : New Sparrow
```

```
b2.setColor("Blue")
```

```
Message from Parent Class : Specificy Color
Message from Bird Class
Color of Bird is  New Sparrow is Blue
```