Dhirubhai Ambani University

(Formerly known as DA-IICT)

Topic: Object Oriented Programming in Python

Course: Programming Lab

Course Code- PC503

Dr. Ankit Vijayvargiya

Assistant Professor

Room No. 4205, Faculty Block 4

Email: ankit_Vijayvargiya[at]dau.ac.in

Phone: 079-68261628(O), 7877709590(M)

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

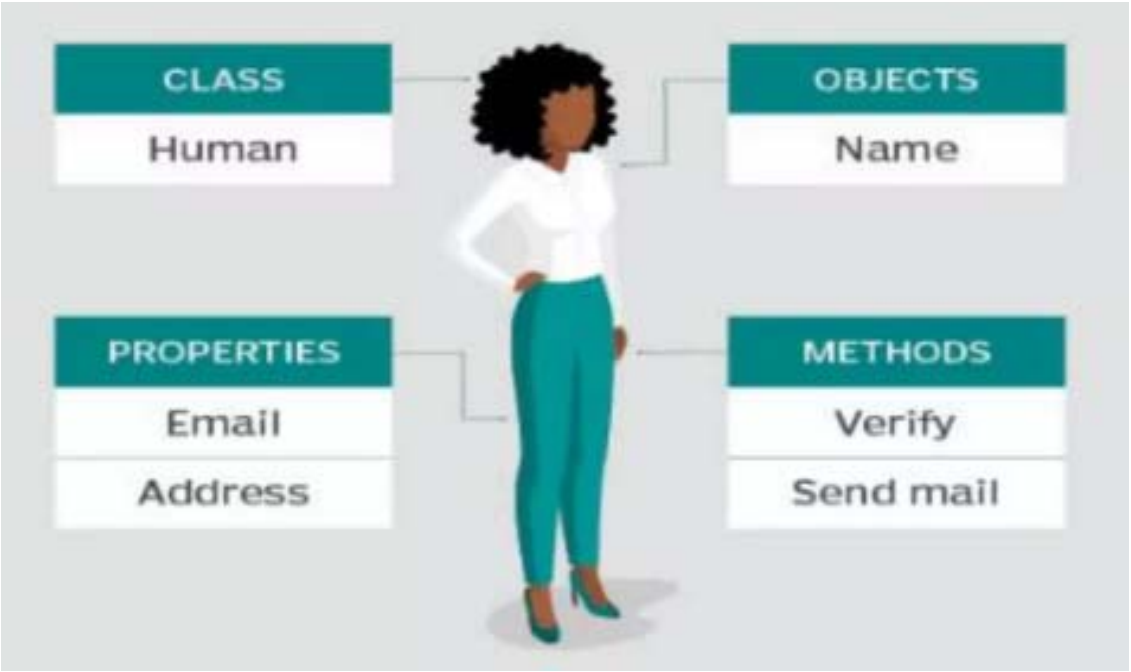
- The important task during programming:
  - Bug free
  - Easy to understand
  - Easy to change/update
- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.
- An object can be defined as a data field that has unique attributes and behavior.
- The first step in OOP is to collect all of the objects a programmers want to manipulate and identify how they relate to each other, which is called data modelling.

- **Library Management System**
  - Objects:
    - Book
    - Member:
    - Librarian
    - Loan

Object Oriented Programming

Paradigm of OOPS

A class is a blueprint for individual objects, attributes, and methods.



Objects are instances of a class created with specifically defined data.

Attributes are defined in the class template and represent the state of an object.

Methods are functions that are defined inside a class that describe the behaviour of the object

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Is Python being OOPS?

- Python is an object-oriented language. This means everything in Python is an object.

a=2
print(type(a))

```
a=2
print(type(a))
<class 'int'>
```

a="Python_Programming"
print(type(a))

```
a="Python_Programming"
print(type(a))
<class 'str'>
```

a=[1,2,3,4]
print(type(a)

```
a=[1,2,3]
print(type(a))
<class 'list'>
```

a is the object of the integer class/string class/ list class → **Built-in datatypes**

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Python Class:

- A class is a **user-defined** data structure that holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- A class is like a blueprint for an object.

Syntax:

```
class ClassName:
        # Statements

obj=ClassName()
print(obj.attr)
```

- Classes are created by the keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be assessed using the dot(.) operator.

Example

```
class vehicle:
    color= "Black"
    price=100000

v1=vehicle()
```

```
v1.color
```

```
'Black'
```

```
v1.price
```

```
100000
```

Empty Class

```
class vehicle:
    pass
v1=vehicle()
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Python class methods:

Once we have defined the class attributes, we can define some functions inside the particular class that can access the class attributes and add more functionality to the existing code.

These defined functions inside a class are known as methods.

```
class vehicle:
    def vehicle_info(self):
        self.color="Black"
        self.price=100000
v1=vehicle()
```

```
v1.vehicle_info()
```

```
v1.color
```

```
'Black'
```

```
v1.price
```

```
100000
```

Self in Python:

- self represents the instance of the class.
- By using the self, we can access the attributes and methods of the class in Python.

```
class vehicle:
    def vehicle_info(self,color,price):
        self.color=color
        self.price=price
v1=vehicle()
```

```
v1.vehicle_info("Red", 100000)
```

```
v1.color
```

```
'Red'
```

```
v1.price
```

```
100000
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Constructor in Python:

- A constructor in Python is a special method named **__init__** that is automatically called when a new object of a class is created.
- It is used to initialize the object's attribute with specific values at the time of creation.

```
class vehicle:
    def __init__(self, color, price):
        self.color = color
        self.price = price
    def show_details(self):
        print(f"Vehicle Color: {self.color}")
        print(f"Vehicle Price: {self.price}")

v1=vehicle("Black", 100000)
v2=vehicle("Red",50000)
```

```
v1.show_details()
```

```
Vehicle Color: Black
Vehicle Price: 100000
```

```
v2.show_details()
```

```
Vehicle Color: Red
Vehicle Price: 50000
```

Exercise

Write a Python program to define a class BankAccount that models a simple bank account. The class should:

- Initialize the account with an optional starting balance (default is 0).
- Have a method deposit(amount) that adds the given amount to the balance.
- Have a method to withdraw (amount) that deducts the given amount from the balance.

Create an instance of BankAccount, deposit 1000 Rs into the account, and then print the current balance, then withdraw 200 Rs and print the balance again.

```
class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance = self.balance + amount

    def withdraw(self, amount):
        if self.balance>0 and self.balance>amount:
            self.balance = self.balance-amount
        else:
            print("No sufficient balance")

account = BankAccount()
```

```
account.deposit(1000)
print("The remaining balance is:",account.balance)
```

```
The remaining balance is: 1000
```

```
account.withdraw(1200)
print("The remaining balance is:",account.balance)
```

```
No sufficient balance
The remaining balance is: 1000
```

```
account.withdraw(200)
print("The remaining balance is:",account.balance)
```

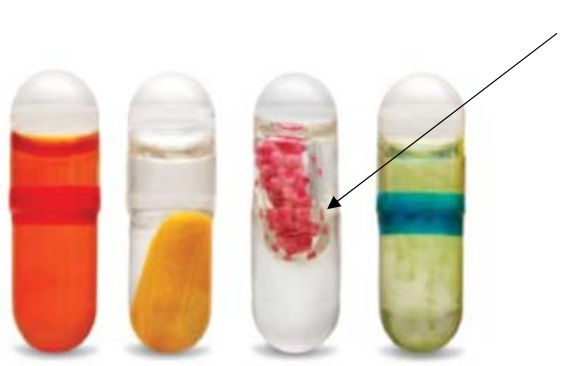
```
The remaining balance is: 800
```

Three key principles of the OOPS:

1. Encapsulation,
2. Inheritance,
3. Polymorphism,

Encapsulation:

Encapsulation is about bundling data (variables) and methods (functions) into a single unit (class) and restricting direct access to some of the object's components.



Similarly, in programming, you hide the inner details of how data is stored or changed, and you give controlled access through methods

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance = self.__balance + amount

    def withdraw(self, amount):
        if self.__balance>0 and self.__balance>amount:
            self.__balance = self.__balance-amount
        else:
            print("No sufficient balance")
account = BankAccount()
```

```
account.deposit(1000)
print("The remaining balance is:",account.__balance)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[22], line 2
      1 account.deposit(1000)
----> 2 print("The remaining balance is:",account.__balance)

AttributeError: 'BankAccount' object has no attribute '__balance'
```


```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance = self.__balance + amount

    def withdraw(self, amount):
        if self.__balance>0 and self.__balance>amount:
            self.__balance = self.__balance-amount
        else:
            print("No sufficient balance")
    def get_balance(self):
        return self.__balance

account = BankAccount()
```

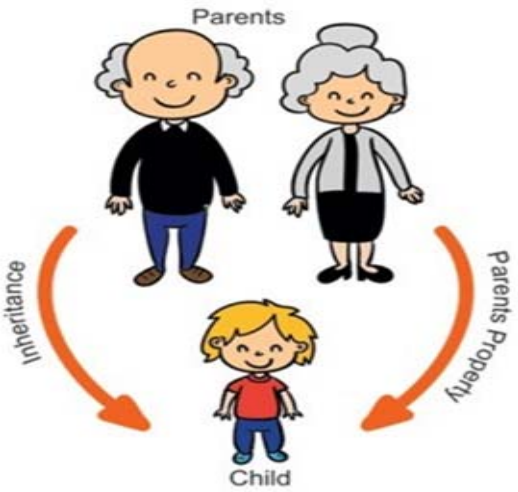

```
account.deposit(1000)
print("The remaining balance is:",account.get_balance())
```

```
The remaining balance is: 1000
```

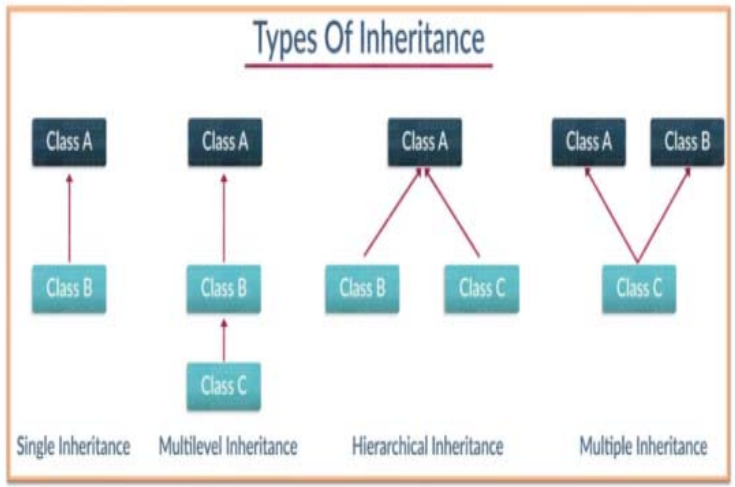
Dhirubhai Ambani University
Formerly known as DA-IICT

Inheritance:

Ever heard relatives say, *"You look exactly like your father/mother"*?
That's **inheritance** in real life — passing characteristics from parents to children.



- In programming, inheritance means transferring attributes and behaviors (properties and methods) from a parent (or base) class to a child (or derived) class without rewriting them.
- The child class automatically has access to everything the parent class has, unless it chooses to override or modify it.



- Single: One child class inherits from one parent class.
- Multiple: One child class inherits from two or more parent classes.
- Multilevel: A class inherits from another class, which itself inherits from another class (like a chain).
- Hierarchical: Multiple child classes inherit from the same parent class.

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if self.__balance>0 and self.__balance>amount:
            self.__balance = self.__balance-amount
        else:
            print("No sufficient balance")

    def get_balance(self):
        return self.__balance


class SavingsAccount(BankAccount):
    def __init__(self, balance=0, interest_rate=0.03):
        BankAccount.__init__(self, balance)  # directly calling parent init
        #super().__init__(balance)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest = self.get_balance() * self.interest_rate
        self.deposit(interest)
```

```
account=SavingsAccount(1000)
```

```
account.add_interest()
```

```
print(account.get_balance())
```

```
1030.0
```

```
account.withdraw(100)
print(account.get_balance())
```

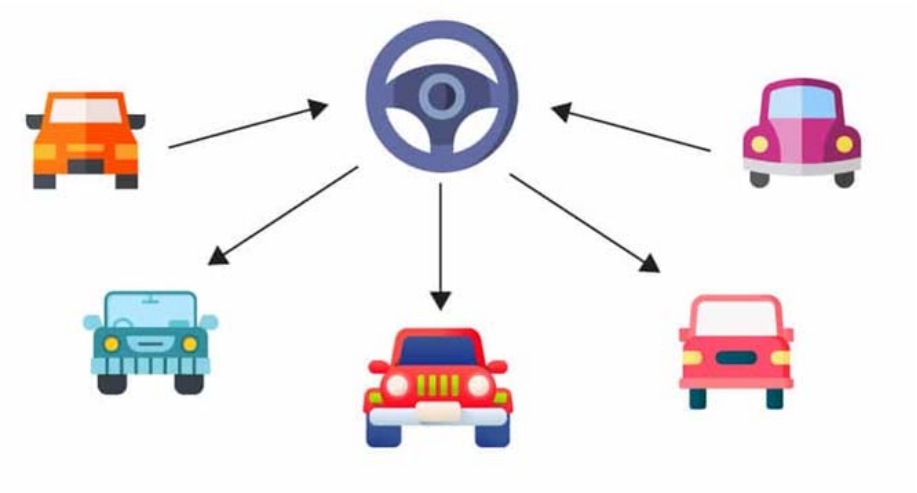
```
930.0
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Polymorphism:

The same word can have different meanings in different contexts — for example, “run” can mean jogging, running a program, or running a business.



In programming, polymorphism means the same method name can behave differently depending on the object calling it.

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if self.__balance>0 and self.__balance>amount:
            self.__balance = self.__balance-amount
        else:
            print("No sufficient balance")

    def get_balance(self):
        return self.__balance


class SavingsAccount(BankAccount):
    def __init__(self, balance=0, interest_rate=0.03):
        BankAccount.__init__(self, balance)  # directly calling parent init
        #super().__init__(balance)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest = self.get_balance() * self.interest_rate
        self.deposit(interest)

class FDAccount(BankAccount):
    def __init__(self, balance=0, interest_rate=0.07):
        BankAccount.__init__(self, balance)  # directly calling parent init
        #super().__init__(balance)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest = self.get_balance() * self.interest_rate
        self.deposit(interest)
```

```
account=SavingsAccount(1000)
```

```
account.add_interest()
```

```
print(account.get_balance())
```

```
1020.0
```

```
account.withdraw(100)
print(account.get_balance())
```

```
920.0
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Object Oriented Programming

Difference

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of Access modifiers 'public', private', protected'	Doesn't use Access modifiers
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance