History of Programming:

Programming Language is the ***set of commands*** and ***instructions*** that we give to the machines ***to perform a particular task***.

1. Machine Language:
   - Machine language is only ***understand by the computers***.
   - In machine language data only represented with the help of ***binary format***(0s and 1s), ***hexadecimal*** and ***octadecimal***.
2. Assembly Language:
   - Assembly language is only ***understand by human beings*** not by the computers.
   - In assembly language data can be represented with the help of mnemonics such as ***MOV***, ***ADD***, ***SUB***, **END** etc.
3. Procedural Programming:
   - ***Procedures*** (functions) that ***manipulate data*** which broken down into a series of steps (procedures) that ***execute sequentially***.
   - ***Data*** and ***procedures*** are often ***separate***, with ***procedures operating on global*** or ***shared data***.
   - Examples: ***C***, ***Pascal***, ***Fortran***.
4. OOP Languages:
   - (**OOP**) is a programming paradigm that uses "***objects***" to ***model real-world entities*** and ***their interactions***.
   - It focuses on creating ***reusable*** and ***maintainable*** code by organizing programs around ***objects***, which combine data (***attributes***) and the functions that operate on that data (methods).
   - Examples: ***JAVA***, ***C++***, etc.

Object-Oriented Programming (OOP) vs Procedural Programming (POP):

- Data and Functions:
  In ***OOP***, ***data*** and ***functions*** are ***tightly coupled*** within ***objects***, while in ***POP***, functions operate on data that ***may be passed separately***.
- Modularity:
  OOP offers ***better modularity*** and ***reusability*** through ***objects*** and ***classes***, which can be ***reused*** across ***different parts of the program***.
- Data Hiding:
  OOP provides mechanisms for ***hiding data within objects***, ***enhancing security***, while ***POP*** may ***lack explicit data hiding*** capabilities.
- Inheritance and Polymorphism:
  OOP supports ***inheritance*** (reusing code) and ***polymorphism*** (multiple forms of a method), which are ***not*** features of ***POP***.

Why OOP is a useful paradigm:

- Real-world modelling:

  OOP allows developers to ***model real-world entities*** and ***relationships*** in their code, making it more intuitive and easier to understand.

- Code organization:

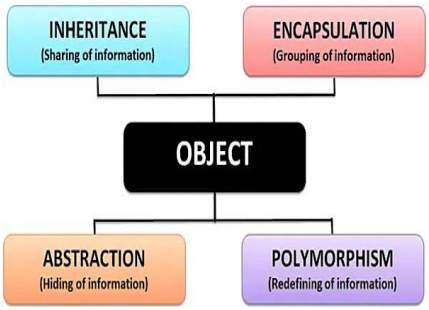
  OOP helps structure code in a ***logical*** and ***organized way***, making it easier to ***develop***, ***maintain***, and ***collaborate on large projects***.

- Collaboration:

  OOP's modular nature and ***clear object-oriented design*** facilitate ***collaboration*** among developers.

The four pillars of Object-Oriented Programming (OOP):

1) Encapsulation
2) Abstraction
3) Inheritance
4) Polymorphism



Abstraction:

- ***abstraction*** is the process of ***displaying only essential information*** to the user, while ***hiding the complex implementation details***.
- This allows developers to work with simplified representations of objects, making it ***easier to design***, ***maintain***, and ***reuse code***.

Sample code:

```java
abstract class Animal{
    Animal(){
     System.out.println("Animal created.");
    }

    abstract void walk();

    public void eats(){
     System.out.println("Animal Eats.");
    }
}
```

```java
class Cow extends Animal{
    Cow(){
        System.out.println("Cow created.");
    }

    @Override
    void walk(){
        System.out.println("Walks on 4 legs.");
    }
}
```

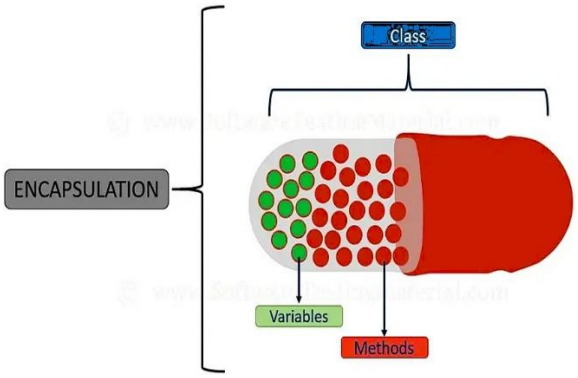
```java
public class Basics {
    public static void main(String[] args){
        Animal c = new Cow();
        c.walk();
        c.eats();
    }
}
```

Output:

```
Animal created.
Cow created.
Walks on 4 legs.
Animal Eats.
```

Encapsulation:

Encapsulation is the ***bundling*** of ***data*** (attributes) and the ***methods*** (functions) that ***operate on that data*** into a ***single unit***, typically a ***class***. This bundling also involves ***restricting direct access*** to some ***components*** of the ***class***, effectively ***hiding the internal workings*** of the object.



- Bundling Data and Methods:
  Encapsulation ***groups related data*** and the ***functions*** that manipulate that data within a single class.

- Data Hiding (Information Hiding):
  It allows you to ***control which parts of the class*** are ***accessible*** from ***outside***. You can make certain attributes ***private***, meaning they can only be ***accessed within the class itself***, or ***public***, meaning they are ***accessible from anywhere***.

- Controlled Access:
  Instead of directly accessing private attributes, you typically use ***getter*** and ***setter*** methods (also called ***accessor*** and ***mutator*** methods) to ***retrieve*** and ***modify*** the ***data***.

Sample code:

```java
class BankUser{
    private int balance;

    // Getter
    public Integer checkBalance(){
        return this.balance;
    }

    // Setters
    public void deposit(int amount){
        if (amount <= 0) return;
        this.balance += amount;
    }

    public void withdraw(int amount){
        if (amount > this.balance) return;
        this.balance -= amount;
    }
}
```

```java
public class Basics {
    public static void main(String[] args){
        BankUser siba = new BankUser();
        siba.deposit(5000);
        System.out.println(siba.checkBalance());
        siba.withdraw(3000);
        System.out.println(siba.checkBalance());
    }
}
```

Output:

```
5000
2000
```