Unit II Object Oriented Programming

• Programming Paradigms :-

- fundamental style of programming defines how the structure and basic elements of a computer program will be built.
- Style of writing programs
- set of capabilities and limitations depends on programming paradigm it supports.

① MONOLITHIC Programming :- importance on finding a solution.

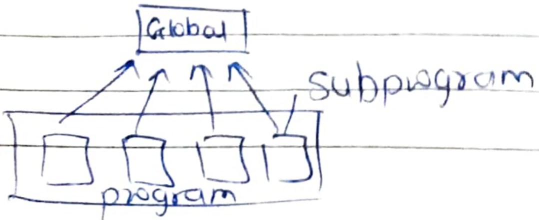
- complete program is written as a sequence.
- no modules or functions are used.
- BASIC consist of global data and sequental code.
- global data can be modified.
- used for very small and simple applications
- Advantages :- simplest way of programming
- Disadvantage :- lot of repetation of code.

② Procedural Programming :- lays stress on algorithms

- program is divided into functions
- function is a small unit of programing logic.

① FORTRAN ② COBOL.

- Advantages :- To write correct programs.
- easier to write programs than monolithic.
- function used multiple times.
- reduces redundancy in program
- makes program readable
- Disadvantages :- Data can be modified.
- Data not protected properly

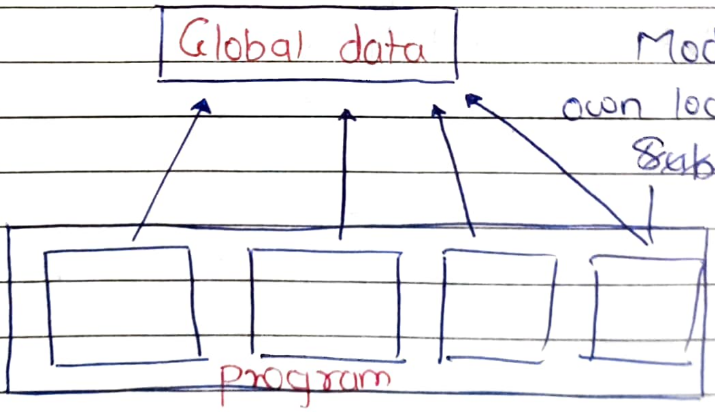


③ Structured programming (Paradigm) :- Focuses on modules.

- program is written in structured format compulsorily.
- most of languages support structured programming.
  C, Pascal

- Advantages :- readable program
- easy to find errors.
- write correct programs that are easy to understand
- takes less time
- each module performs specific task.
- each module has its own local data
- Disadvantages :- Data can be modified
- not data-centered.
- Global data is shared
- main focus on functions.

~~④~~



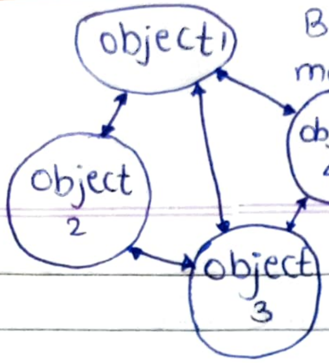
Modules have their own local data ~~Subprograms~~ and also share ~~local~~ ~~&~~ global data

④ Object Oriented programming: importance on classes and objects. • Data is major ~~factor~~ focus

- Data and functions put together to avoid misuse
- "Class" put together data and functions.
- functions belonging to that class can modify data.
- OOP supports features

Advantages :- OOP features enables reuse of programs
- Data is protected

Disadvantage :- ineffiecient programs.



By sending message to each othe

+ Features of Object Oriented Programming

① Class :- • A blue print to create objects
- model used to generate exact same object
- variables and functions are put together.

Advantages :-
- allows creating user defined data structure.
- putting variables and function together make data protection easy
- Helps in simulating real world scenario.

Example :-

```
    class sample :
        def instance_method (self):
            print ("In instance method")----  first argument 'self' (instance method)
        def class_method ():  ......  without 'self' parameter (class method)
            print ("In class method")
s= Sample()
s. instance -method ()  ........  instance method is accessed using object
Sample. class - method () ......  class method is accessed using class name
O/P→ In instance method
     In class method
```

② Objects :- object is a instance of a class
- contains all variables and methods together
class is like plan of a building and object is real building.
- One plan can be used to generate multiple buildings

Example :-

```
object_s_1 = Sample (10)   Object of class sample
object-s- 2= Sample(20)    object of class sample
object-s- 3 = Sample (30)  object of class sample
```

③ Methods :- defined inside a classs.

- can work on the data or variables inside the class.
- can be used to modify or read different variables of any object.
- use values of variables which are stored inside calling object.
- same method called from different objects will give different results.

Example :

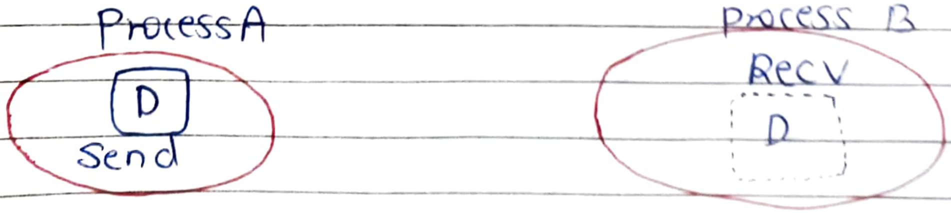
__init__ (self, ..., ...) :: ⇒ constructor method
displayCount() :: This is a class method
displayEmployee (self) :: Instance method

- Message Passing :-

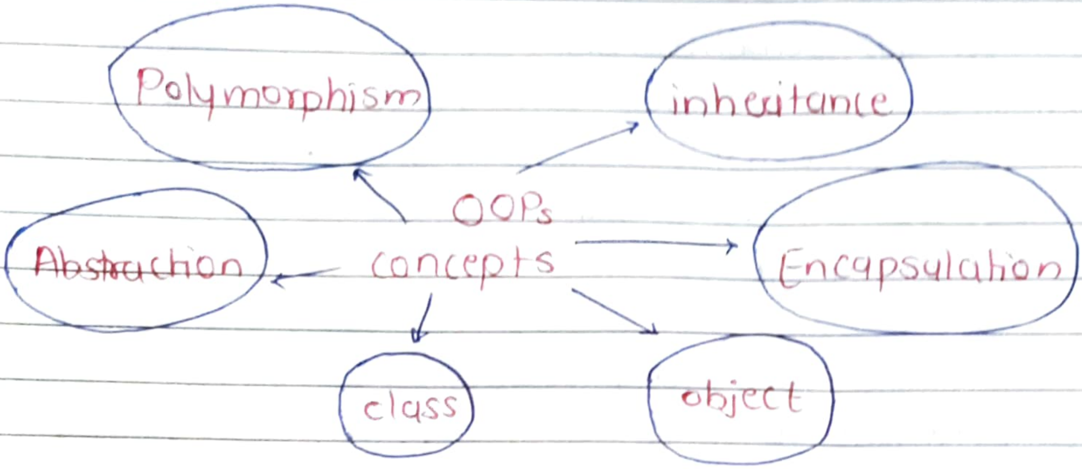
send and receive calls operate.

- process A deceides a message needs to be sent to process B.
- Process A then packs up all of its necessary data into a buffer for process B.
- Process A indicates that the data should be sent to B by calling send function.
- Before process B can receive the data, it needs to acknowledge that it wants to receive it

Process B does this by calling Recv function.



* Features of OOP :-



① Inheritance :- • child class inherits properties from a parent or base class.

- Methods of parent class can be directly accessed in child class.
- All public variables of parent class are accessible used to re-use methods of parent class easily
- "supe()" keyword is used to access parent class constructor.

② Polymorphism :- • same method behaves differently for different inputs.

- Advantages:
- code reusability
- operators behave according to input so readability of program increases.

③ Containership :- object of a class can be part of another class.

- can be class within class.
- like department which contains objects of classes & labs.

Advantage :-

Real world scenario can be better model.

④ Re-usability :-

- most important application.
- A code in class or function can be reused by inheritanre or by overiding.
- makes use of libraries very easy.
- increases reliability of code.

⑤ Delegation :- code is highly modular.

- each class has well defined functionalities.
- it becomes easy to just access the functionality anywhere.
- No other class needs to rework again for same functionality.
- allowing ease of access to different modules.
- each module has fixed small functionality.
- helps in generating redable code.

⑥ Data Abstraction :- User can access data only according to his access level.

- helps create standard classes.
- force implementation in future child classes.
- give abstract overview of required standard methods.

⑦ Encapsulation :- increases data security allows access and modification of private data ONLY by relevant functions

- **Classes :-** user defined blue-print or prototype (model) from which objects are created
- Classes provide a means of building data and functionality together.
- Creating new class creates new types of object
- class instances can also have methods.

Syntax :-

```
class ClassName:
      #statement
```

Classes :-

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

Class creates a user-defined data-structure, which holds its own data members and member functions, which can be accessed.

* **Objects :-** An object is an instance of a class.

State: It is represented by attributes of an objects.
reflects properties of an objects.

Behaviour: represented by methods of objects

Identity: gives a unique name to an object.

```
Class Dog:
    attr1="mammal"
    attr2= "dog"
  def fun(self):
      print ("Im a", self.attr1)
      print ("Im a", self.attr2)
  print (Rodger.attr1)
      Rodger.fun()
```

```
• class Student:
    mark=0

def compute_marks (self, obtained-marks):
     marks=obtained_marks
      print ('Obtained Marks:', marks)

 Student. print_marks =classmethod(Student.compute_marks)

 student. print-marks (88)

O/P ⇒  Obtained Marks: 88
```

- **Self object :-** 'self' is an important keyword in python
- "self" current object calling method.
- "self" gives access to all values stored in object.
- "self" can be used ONLY within an instance method within a class.
- "self" is also used as first default parameter to every function in a class

- **Class variable / Static Variable :-**
- variables which are created once for a class.
- common to all subjects of a class.
- declared outside any function in a class
- accessed using name of class.
- do not require object of class to access them.
- ALL class variables are PUBLIC.

```
class sample:
        class variable =0
        def --init--(self, local-var):
                self. instance-var =local-var
    S=Sample (10)
    print("Value of instance variable is", s.instance-var)
    print("Value of class variable is",sample.classvariable)
```

Object Variable	Class Variable
i) value is instance-specific and now shared among instances	i) defines a specific attributes or property for a class.
ii) cannot shared between classes	ii) can be shared between class.
iii) reserves memory for data that class needs.	iii) maintains a single shared value for all instances.
iv) created when an instance of the class is created	iv) created when the program begins to execute.
v) retains values as long as the object exists.	v) retains value until the program terminates
vi) every object has its own personal copy	vi) only has one copy shared among different objects.
vii) can be accessed directly	vii) accessed by calling class name
viii) variables declared without using Keyword	viii) variables are declared using keyword.
ix) changes that made to variables, one object will not reflect in another object	ix) changes that made to variables, one object will reflect in another object.

• Public Members :- accessible from outside the class.
All members in a python class are public by default.
Any member can accessed from outside the class environment.

```
class Student:
    SchoolName = 'XYZ School'

    def --init--(self, name, age):
        self.name = name
        self.age = age

    >>> std = Student("Steve", 25)
    >>> std.schoolName
        'XYZ School'
    >>> std.name
        'steve'
    >>> std.age = 20
    >>> std.age
        20
```

* **Private Members** :- Python prescribes a convention of prefixing the name of the variable/method with a single or double underscore to emulate the behavior of protected and private access specifiers.

- double underscore ("__") prefixed to a variable makes it private.
- gives a strong suggestion not to touch it from outside the class.

```
class student:
    _schoolName = 'XYZ School'
    def__init__ (self, name, age):
        self._name = name
        self._salary = age
    def_display (self):
        print ('This is private method'.)

>>> std = Student ("Bill", 25)
>>> std._schoolName
AttributeError: 'Student' object has no attribute '_Schoolna
>>> std._name
AttributeError: 'student' object has no attribute '_name'
>>> std._display()
AttributeError: 'Student' object has no attribute '_display'
```

- performs name mangling of private variables.
-

*

* Class Method:-

```
class person :

    def--init--(self, name, age):
        Self.name =name
        self.age = age
            decorator
@class method
    def Birthyear (cls, name, ~~age~~ year)
    return cls (name, 2623-Birthyear)

    def show(self):
        print ("name", self.nome)
        print ("age", self.age)

person1= person ("Rohan", 36)

person 2= person. Birthyear (2004)
```