1. Encapsulation - It is a practice of bundling related data into a structured unit along with the methods used to work with the data. It provides a way to restrict direct access to attributes of an object which prevents accidental modification of data.

```
class Employee:
  def __init__(self,name,age):
    self.__name=name # Private variable
    self.__age=age
  def getname(self): # method to access private variable
    return self.__name
  def getage(self):
    return self.__age
emp=Employee("Dheepa",24)
print(emp.getname())
print(emp.getage())
```

```
Dheepa
24
```

2. Polymorphism - It is an ability to represent objects of different types using a single interface.It provides a way to substitute classes that have common functionality in sense of methods and data.

```
class Shape:
  def intro(self):
    print("I am a shape")
class Square(Shape):
  def intro(self):
    print("I am a square")
class Rectangle(Shape):
  def intro(self):
    print("I am a rectangle")
obj= Shape()
sq= Square()
rec= Rectangle()
obj.intro()
sq.intro()
rec.intro()
```

```
I am a shape
I am a square
I am a rectangle
```

3. Single Level inheritance -It is an ability to define a single child class that inherits all the methods and properties from a single parent class.

```
class Employee: # Parent class
  def __init__(self,name,id):
    self.__name=name
    self.__id=id
  def employee_details(self):
    return self.__name,self.__id
class employee2(Employee): #child class
  def intro(self):
    print("Inside Employee2")

emp=Employee("A",123)
emp1=employee2("B",456)
print(emp.employee_details())
print(emp1.employee_details())
emp1.intro()
```

```
('A', 123)
('B', 456)
Inside Employee2
```

4. Multiple inheritance - A type of inheritance in which the child class is derived from more than one super class

```
class Father:
  def father(self):
    return "Father"


class Mother:
 def mother(self):
   return "Mother"

class child(Father,Mother):
  def __init__(self,name):
    self.name=name
c=child("xxx")

print(c.father())
print(c.mother())
```

```
Father
Mother
```

5. Multilevel Inheritance - A type of inheritance in which the child class is derived from a parent class which itself a derived from another class.

```
class Shape:
    def area(self):
       print("Area")


class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        print(f"Rectangle = {self.length*self.width}")


class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)

    def area(self):
        print(f"Square = {self.length ** 2}")
rec=Rectangle(2,3)
sq=Square(4)
rec.area()
sq.area()
```

```
Rectangle = 6
Square = 16
```

6. Conditional statements - Statements that provide a choice for the control flow based on a condition.

```
number=50

if number >=50 :
  print(f"Number greater than 50")
else:
  print(f"Number less than 50")
```

```
Number greater than 50
```

7. Decision making statement- decision-making statements let programs decide what to do and run distinct code blocks according to predefined conditions.The if, elif and if-else statements controls the flow of the code in python.

```
name="Polymorphism"
if(len(name)>10):
  print("Bigger string")
elif(len(name)>5):
  print("big string")
else:
  print("short string")
```

```
Bigger string
```

8. Factorial

```
def factorial(n):
  if n==0 or n==1:
    return 1
  else:
    return n*factorial(n-1)
print(factorial(5) )
```

```
120
```

9. Functions - a block of organized, reusable code that performs a specific task.

```
def addition(x,y):
  return x+y
print(3,4)
print(5,7)
print(7,29999)
```

```
3 4
5 7
7 29999
```

10. Pillars of OOPS - There are 4 pillars of OOPS
    - Encalsulation
    - Abstraction
    - Polymorphism
    - Inheritance

Start coding or generate with AI.

Start coding or generate with AI.