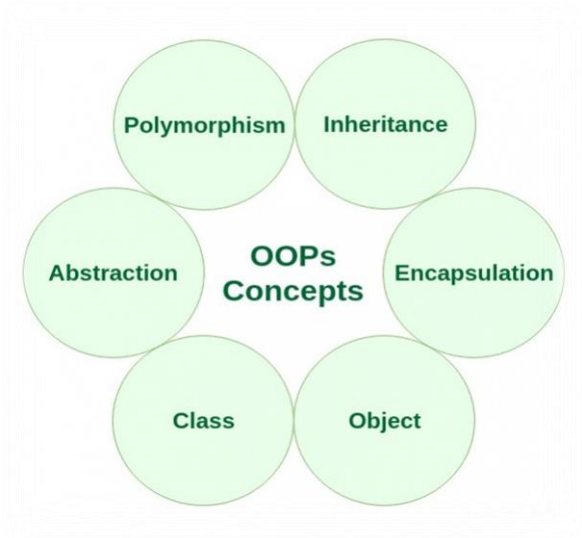
OOPS

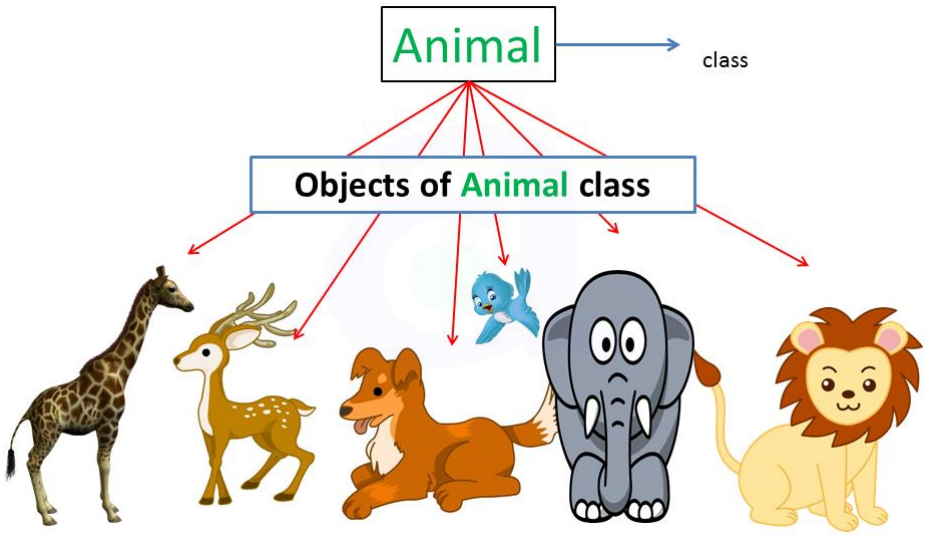
- Object oriented programming
- Used to implement real world entities into programming.
- Increasing readability and reusability of code

6 CONCEPTS OF OPPS



1. **CLASS**
   - A class is like a blueprint for an object.
   - It represents the set of objects which have common properties or methods.
   - It's a logical entity.
2. **OBJECT**
   - Member of class
   - Physical entity
   - An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (an object is created) memory is allocated.

EXAMPLE: If animal is a class, every animal like dog, lion, cat are the objects of animal class. All animals share common properties and methods like walking style, food habits and their behaviour.



SIMPLE SYNATX OF CLASS AND OBJECT CREATION:

```
class A:
    def sample(self):
        print('hello')

obj  = A()
obj.sample()
```

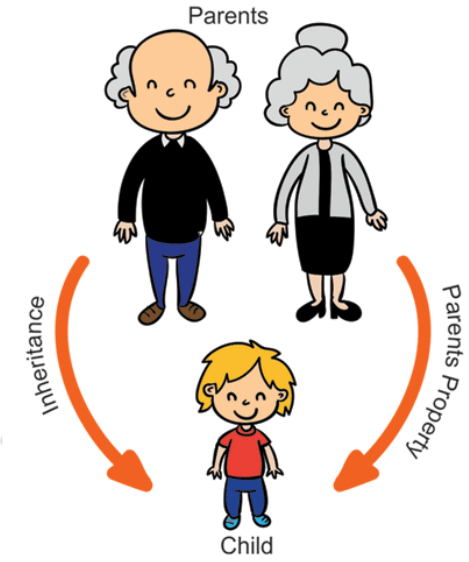
OUTPUT:

Hello

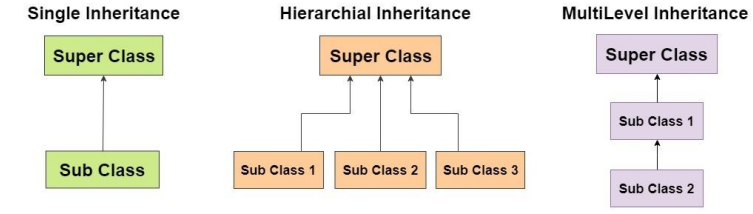
3. INHERETANCE

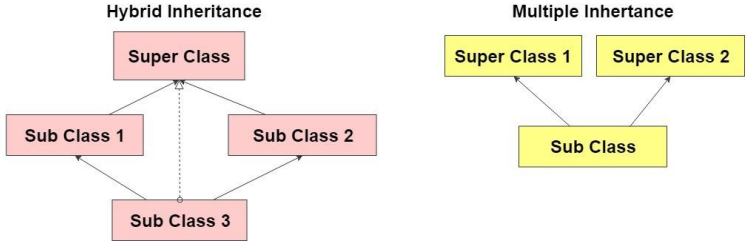
- It's a capability of a class to derive properties and characteristics from another class.
- The parent class called super class and the child class which inherit from parent class is called derived class or sub class.
- Derived class or child class have Its own properties and methods and also get all the properties and methods of parent class or super class.
- It increases reusability of code.

EXAMPLE:



TYPES OF INHERETANCE:





HIERARCHIAL INHERETANCE SYNTAX:

```
In [ ]: class Parent:
            def func1(self):
                print("this is function 1")

        class Child(Parent):
            def func2(self):
                print("this is function 2")

        class Child2(Parent):
            def func3(self):
                print("this is function 3")

        ob1 = Child()
        ob2 = Child2()
        ob1.func2()
        ob2.func3()
        ob1.func1()
        ob2.func1()
```

4. POLYMORPHISM

- means having many forms
- For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person shows different behaviour in different situations. This is called polymorphism.

TYPES OF POLYMORPHISM

- **Method overloading:**
  Multiple methods with the same name but with a different type of parameter or a different number of parameters is called Method overloading.

```
def product(a, b):
    p = a*b
    print(p)

def product(a, b, c):
    p = a*b*c
    print(p)

#Gives you an error saying one more argument is missing as it updated to the second function
#product(2, 3)
product(2, 3, 5)
```

Output:
30

- **Method overriding:**

  If a subclass method has the same name as in the superclass Method then it is called Method overriding.

  To achieve method overriding we must use inheritance.

```python
class A:
    def sayhi(self):
        print("I am in A")

class B(A):
    def sayhi(self):
        print("I am in B")

ob = B()
ob.sayhi()
```

OUTPUT:

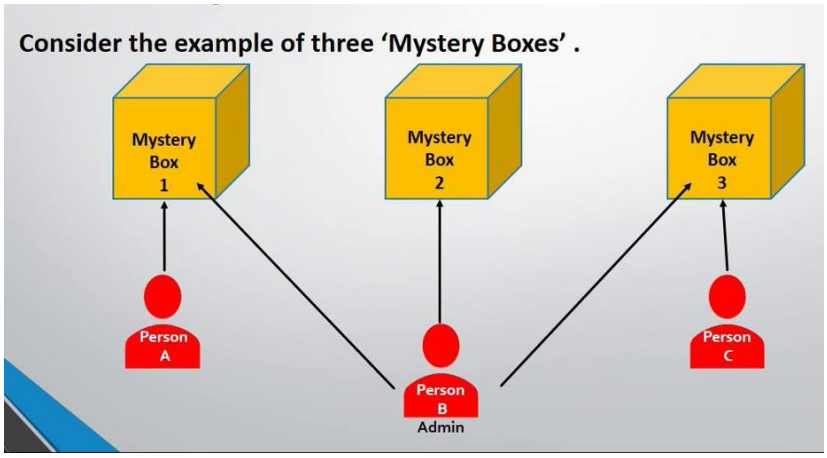
```
I am in B

Process finished with exit code 0
```

5. ABSTRACTION

- Show only required information to specific user and hid unnecessary information.

  EXAMPLE:

  Driver can see and use the steering, wheels, seats of a car but cannot see the behind mechanism of engine of a car.



Only admin have access to 3 boxes. Person A have only access to box 1, person C have access to box 3 only.

6. ENCAPSULATION (not satisfying this concept in python)

- This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.

- Compared to languages like Java that offer access modifiers (public or private) for variables and methods, Python provides access to all the variables and methods globally.

__INIT__ CONSTRUCTOR IN PYTHON

- It's a constructor of a class

- Python will automatically call the __init__ () method immediately after creating a new object, you can use the __init__ () method to initialize the object's attributes(variables or data members).
- The double underscores at both sides of the __init__ () method indicate that Python will use the method internally. So it only declare within a class.

Syntax

```
class GreetClass:
    def __init__(self):
        self.say_hello = "Hello!"
        self.say_hi = "Hi!"

hello_obj = GreetClass()
print(hello_obj.say_hello)

hi_obj = GreetClass()
print(hi_obj.say_hi)
```

```
C:\Users\ak111\PycharmProjects\pythonProject\venv\Scripts\py
Hello!
Hi!
```