INHERITANCE

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class

---

Now lets say want to create different types of employees

- manager
- developers

What we can do is create two different types of class one for manager and another for developer. but that is not what oop is about.
OOP is about code reuse.

So what we do is break the entire problem into 3 separate classes.
One class would have all the attributes common to both the employee and developer class.
and the next 2 classes for manager and developer with the things that are not common

---

-- This is called as subclassing.

-- The subclasses inherit from the base class.

-- So in our case the the **base class** is the **employee class**

-- And the **subclasses** are the **manager and developers class**.

In [12]:
```
# creating parent class employee

class Employee():

    raise_amount = 1.04

    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'
```

```
    def full_name(self):
        return self.first_name+' '+self.last_name

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)
        print(self.salary)
```

In [8]:
```
# creating devleoper class by inheritance method

class Employee():

    raise_amount = 1.04

    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'

    def full_name(self):
        return self.first_name+' '+self.last_name

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)

class Developer(Employee):
    pass
```

In [10]:
```
dev1 =Developer("test1","user",50000)
```

In [11]:
```
print(dev1.salary)
```

```
50000
```

In [14]:
```
print(dev1.raise_amount)
```

```
1.04
```

In [13]:
```
# creating developer class with different raise amount
```

In [15]:
```
class Developer(Employee):
    raise_amount =2.0
```

In [16]:
```
dev1 =Developer("test1","user",50000)
```

In [17]:
```
print(dev1.raise_amount)
```

```
2.0
```

-- What the above cell means is that the changes made to the subclass **doesnot chnage anything in the base class**.

-- In the above code we initiated the developer class with the attributes that were defined in the parent class.

-- **lets say we want to initialise the subclass with much more attributes**.

In [18]:
```
class Employee():

    raise_amount = 1.04

    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'

    def full_name(self):
        return self.first_name+' '+self.last_name

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)

class Developer(Employee):
    raise_amount = 1.1

    def __init__(self,first,last,pay,lang):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'
        self.lang = lang

dev1 = Developer('test1','user',3000,'python')
```

In [19]:
```
dev1.apply_raise()

print(dev1.salary)
```

```
3300
```

---

problem : The above coed works fine but the problem being we just repeated ourselfs which is against OOPs

Solution :

- Use of `super()` key word.
- using super() we can direcetly pass atributes without passing the self keyword.
- but when using the class name to initialise we must pass the self keyword.

In [21]:

```
class Employee():

    raise_amount = 1.04

    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'

    def full_name(self):
        return self.first_name+' '+self.last_name

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)

class Developer(Employee):
    raise_amount = 1.1

    def __init__(self,first,last,pay,lang):
        super().__init__(first,last,pay)
        self.lang = lang

dev1 = Developer('test1','user',3000,'python')
```

In [23]:

```
# creating manager class in same way

class Employee():

    raise_amount = 1.04

    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'

    def full_name(self):
        return self.first_name+' '+self.last_name

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)

class Developer(Employee):
    raise_amount = 1.1

    def __init__(self,first,last,pay,lang):
        super().__init__(first,last,pay)
        self.lang = lang

class Manager(Employee):

    def __init__(self,first,last,pay,designation):
        super().__init__(first,last,pay)
        self.designation = designation


dev1 = Developer('test1','user',3000,'python')
dev2 = Developer('test2','user',6000,'java')
```

```
mang1 = Manager('mang1','super1',30000,'sales')
```

In [25]:
```
print(mang1.salary)
```

```
30000
```

In [26]:
```
print(dev1.lang)
```

```
python
```

In [27]:
```
# Task : Add more methods into the devleoper and manager class
```

POLYMORPHISM

- Polymorphism in python refers to how different object classes can share the same method name

In [28]:
```
# Creating separate class for dog with the method speak

class Dog():
    def __init__(self,name):
        self.name = name

    def speak(self):
        return '{} says woff'.format(self.name)
```

In [29]:
```
# Creating separate class for cat with the method speak
class Cat():
    def __init__(self,name):
        self.name = name

    def speak(self):
        return '{} says meow'.format(self.name)
```

In [30]:
```
# creating objects

dog1 = Dog('richard')
cat1 = Cat('isis')

print(dog1.speak())
print(cat1.speak())
```

```
richard says woff
isis says meow
```

In [31]:
```
# calling speak metho for pet using for loop in both classes

for pet in [dog1,cat1]:
    print(pet.speak())
```

```
richard says woff
isis says meow
```

In [34]:
```
def talk(pet):
    print(pet.speak())
```

In [35]:
```
talk(dog1)
```

```
richard says woff
```

In [36]:
```
talk(cat1)
```

```
isis says meow
```

meaing of above code :

In [ ]:
```

```

REVISION

- 1) Understanding : Inheritance
- 2) creating sub class: developer
- 3) creating sub class: manager
- 4) Understanting : Polymorphism
- 5) creating class for dog and cat to understand polymorphism

---

HOMEWORK

1. Create a vehicle class which is empty. Also create a vehicle object using this class.

In [ ]:
```

```

1. To the Above vehicle class add the following attributes
   - Name
   - Top speed

In [ ]:
```

```

1. Inherit from the vehicle class and create 2 base classes
   - Car
   - Bike

In [ ]:
```

```