Inheritance

In this notebook we introduce the concept of inheritance - a very important principle in object oriented programming. Inheritance helps to build a new class using an existing class thereby promoting reuse and enabling polymorphism. Using inheritance, a new class (called the child class or the sub class) can be created by extending an old class (called the base class, the parent class or the super class). The child class is a specialization of the base class and the base class is a generalization of the child class. The is-a relationship is used to depict the association between a base class and a child class. In other words, we say that a child class object 'is' (also) a base class object. The child class implicitly inherits all the attributes and methods of the base class and in addition can have its own attributes and methods.

In Python, all classes (except the class `object`) inherit from the `object` class

We will study inheritance via a simple example. We will begin by defining the parent class - Employee.
The Employee class has just two instance variables; first name and last name. It has the appropriate accessor and mutator methods in addition to the __init__ and the __str__ methods.

In [3]:

```
'''
Defines a blueprint of an Employee
'''
class Employee():

    def __init__(self, fn, ln):
        self.__first_name = fn
        self.__last_name = ln

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def set_first_name(self, fn):
        self.__first_name = fn

    def set_last_name(self, ln):
        self.__last_name = ln

    def __str__(self):
        return "The employee's name is {} {}".format(self.__first_name, self.__last_name)
```

We will next define a child class that has the Employee class as a parent - Full_Time_Employee. Note that a Full_Time_Employee is also an Employee and hence has all the methods and attributes of the Employee class. We therefore need not define them again.

The Full_Time_Employee has two instance variables in addition to first name and last name - salary and job title. We will define the accessor and mutator methods for the additional attributes and the __init__ and __str__ methods.

We will begin with the class statement for the Full_Time_Employee class. To indicate that one class is the child of another class, we use the syntax shown below.

In [7]:

```
'''
Defines a blueprint of a full-time employee
'''
class Full_Time_Employee(Employee):
    pass
```

We will next write the __init__ method. Since the child class has all the attributes of the parent class, we need to initialize those attributes as well as the additional attribute of the child. To initialize the parent class attributes, we reuse the __init__ method for the parent class. This is done by calling the parent class's __init__ method using the following syntax.

super().__init__(param1, param2.....) -- where param1, param2 etc. are the parameters that need to be passed to the parent class's __init__ method.

After initializing the parent class attributes, we can initialize the child class as usual.

In [ ]:

```
'''
Defines a blueprint of a full-time employee
'''
class Full_Time_Employee(Employee):
    def __init__(self, fn, ln, jt, sal):
        super().__init__(fn, ln) #initializing the instance variables of the base class
        self.__salary = sal
        self.__job_title = jt
```

We next create the accessor and mutator methods for the additional attributes of the child class. Note that the accessor and mutator methods for the parent attributes are already defined in the parent class.

In [ ]:

```
'''
Defines a blueprint of a full-time employee
'''
class Full_Time_Employee(Employee):
    def __init__(self, fn, ln, jt, sal):
        super().__init__(fn, ln)
        self.__salary = sal
        self.__job_title = jt

    def get_title(self):
        return self.__job_title

    def get_salary(self):
        return self.__salary

    def set_title(self, jt):
        self.__job_title = jb

    def set_salary(self, sal):
        self.__salary = sal
```

Finally, we will write the __str__ method for the child class. We do this by first calling the parent class's __str__ method and then adding to it.

In [8]:

```
'''
Defines a blueprint of a full-time employee
'''
class Full_Time_Employee(Employee):
    def __init__(self, fn, ln, jt, sal):
        super().__init__(fn, ln)
        self.__salary = sal
        self.__job_title = jt

    def get_title(self):
        return self.__job_title

    def get_salary(self):
        return self.__salary

    def set_title(self, jt):
        self.__job_title = jb

    def set_salary(self, sal):
        self.__salary = sal

    def __str__(self):
        op = super().__str__() #we call the parent class's __str__method first.
        op += ". The salary is: {} and the job title is: {}".format(self.__salary,
```

```
self.__job_title)
        return op
```

Test your child class by creating a child object and printing out the details.

In [10]:

```
'''
Test your child class
'''
child = Full_Time_Employee('Jane', 'Doe', 'Teacher', '45000')
print(child)
```

```
The employee's name is Jane Doe. The salary is: 45000 and the job title is: Teacher
```

The `isinstance(a, B)` method returns True if object a is an instance of class B. It returns False otherwise.

The `issubclass(A, B)` method returns True if class A is a subclass of class B

In [14]:

```
'''
Test class and inheritance relationships
'''
child = Full_Time_Employee('Jane', 'Doe', 'Teacher', '45000')
print(child)
emp = Employee('Jack', 'Ryan')
print(emp)
#The isinstance() method returns True if the object is an indirect or direct child of the class
print('isinstance(emp, Full_Time_Employee): ', isinstance(emp, Full_Time_Employee))
print('isinstance(child, Employee):', isinstance(child, Employee))

print('issubclass(Full_Time_Employee, Employee):', issubclass(Full_Time_Employee, Employee))
print('issubclass(Employee, Full_Time_Employee):', issubclass(Employee, Full_Time_Employee))
# A class is considered a subclass of itself
print('issubclass(Full_Time_Employee, Full_Time_Employee):', issubclass(Full_Time_Employee,
Full_Time_Employee))
```

```
The employee's name is Jane Doe. The salary is: 45000 and the job title is: Teacher
The employee's name is Jack Ryan
isinstance(emp, Full_Time_Employee):  False
isinstance(child, Employee): True
issubclass(Full_Time_Employee, Employee): True
issubclass(Employee, Full_Time_Employee): False
issubclass(Full_Time_Employee, Full_Time_Employee): True
```

Note that the child class has access to all the methods in the parent class. However, it does not have access to the hidden attributes in the parent class. These can only be accessed using the appropriate get and set methods.

This will return an error. However, we can get around it (not recommended) by prefixing the variable name with an underscore and the parent class name.

In [15]:

```
'''
Test class properties and attributes
'''
print(child.get_first_name())
print(child.__first_name)
```

```
Jane

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-15-fc583c453d08> in <module>
      3 '''
      4 print(child.get_first_name())
----> 5 print(child.__first_name)
```

```
AttributeError: 'Full_Time_Employee' object has no attribute '__first_name'
```