Inheritance

Inheritance is the capability of one class to derive or inherit the properties from some another class. The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides **reusability** of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

What is object class?
Like Java Object class, in Python (from version 3.x), object is root of all classes.
In Python 3.x, "class Test(object)" and "class Test" are same.

Subclassing (Calling constructor of parent class)
A child class needs to identify which class is its parent class. This can be done by mentioning the parent class name in the definition of the child class.
Eg: class **subclass_name (superclass_name)**:

```
# Python code to demonstrate how parent constructors are called.

# parent class
class Person(object):  #Note: same as class Person():

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)
```

```
# creation of an object variable or an instance
a = Person('Ana', 81178)

# calling a function of the class Person using its instance
a.display()
```

Output

```
Ana
81178
```

'a' is the instance created for the class Person. It invokes the __init__() of the referred class. You can see 'object' written in the declaration of the class Person. In Python, every class inherits from a built-in basic class called as 'object'. The constructor i.e. the '__init__' function of a class is invoked when we create an object variable or an instance of the class.

The variables defined within __init__() are called as the instance variables or objects. Hence, 'name' and 'idnumber' are the objects of the class Person. Similarly, 'salary' and 'post' are the objects of the class Employee. Since the class Employee inherits from class Person, 'name' and 'idnumber' are also the objects of class Employee.

If you forget to invoke the __init__() of the parent class then its instance variables would not be available to the child class. The following code produces an error for the same reason.

```
# Python program to demonstrate error if we
# forget to invoke __init__() of parent.

class A:
    def __init__(self, n = 'Rahul'):
        self.name = n
class B(A):
    def __init__(self, roll):
        self.roll = roll

object = B(23)
print (object.name)
```

Output:

```
Traceback (most recent call last):
  File "/home/de4570cca20263ac2c4149f435dba22c.py", line 12, in
    print (object.name)
AttributeError: 'B' object has no attribute 'name'
```

Overloading and Overriding

Overloading

Demonstrates the Overloading and the use of the "__str__" method. By using the __str__ method, we gain a lot, especially a leaner design. We have a string casting for our classes and we can simply print out instances.

```
class Person:
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last

    def __str__(self):
        return self.firstname + " " + self.lastname

class Employee(Person):
    def __init__(self, first, last, staffnum):
        super().__init__(first, last)
        self.staffnumber = staffnum


x = Person("Jane", "Doe")
y = Employee("John", "Doe", "1001")

print(x)
print(y)
```

Output:

```
Jane Doe
John Doe
```

We can see that if we print an instance of the Employee class, the __str__ method of Person is used. This is due to inheritance.

Overriding

```
class Person:

    def __init__(self, first, last, age):
        self.firstname = first
        self.lastname = last
        self.age = age

    def __str__(self):
        return self.firstname + " " + self.lastname + ", " + str(self.age)

class Employee(Person):
    def __init__(self, first, last, age, staffnum):
        super().__init__(first, last, age)
        self.staffnumber = staffnum

    def __str__(self):
        return super().__str__() + ", " +  self.staffnumber


x = Person("Jane", "Doe", 36)
y = Employee("John", "Doe", 28, "1001")

print(x)
print(y)
```

Output:

```
Jane Doe 36
John Doe 28 1001
```

Another Inheritance Example

Let's take Vehicle as a parent class from which we will derive a class Category. Category class will inherit the features of parent class Vehicle and also invoke the function from the parent class.

```
class Vehicle:   #parent class
  "Parent Class"
  def __init__(self, price):
    self.price = price
  def display(self):
    print ('Price = $',self.price)
```

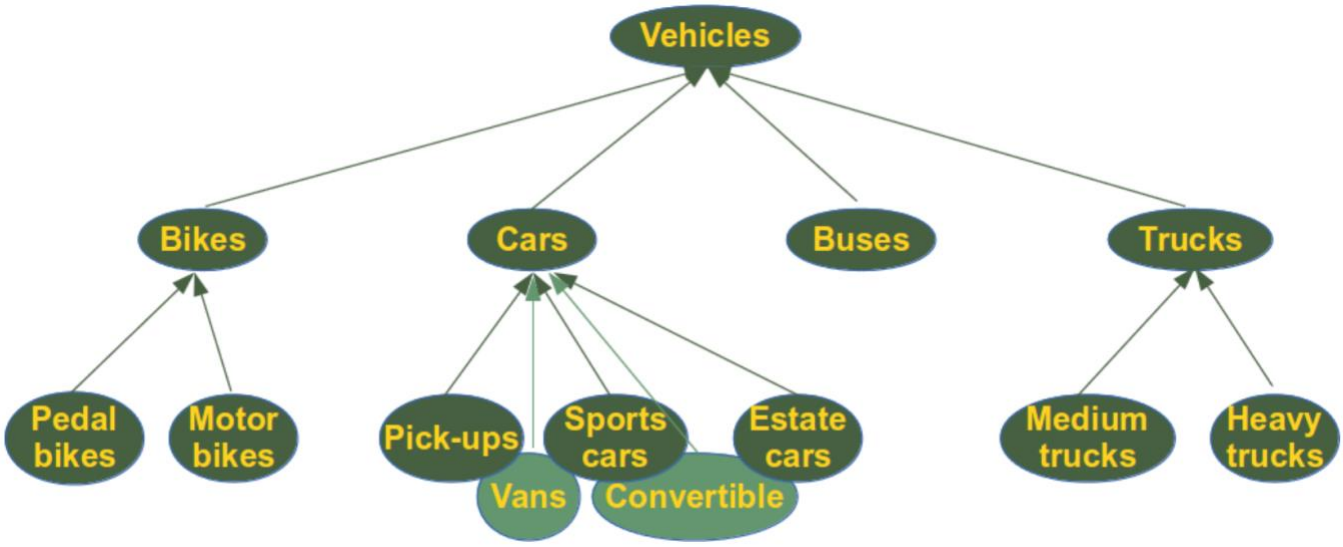
```
class Category(Vehicle):   #derived class
   "Child/Derived class"
   def __init__(self, price, name):
     Vehicle.__init__(self, price)
     self.name = name

   def disp_name(self):
     print ('Vehicle = ',self.name)

obj = Category(1200, 'BMW')
obj.disp_name()
obj.display()
```

Output:
Vehicle = BMW
Price = $1200

Inheritance example



Multiple Inheritance

```
class Person:
    def __init__(self, name, surname, number):
        self.name = name
        self.surname = surname
        self.number = number


class LearnerMixin:
    def __init__(self):
        self.classes = []

    def enrol(self, course):
        self.classes.append(course)


class TeacherMixin:
    def __init__(self):
        self.courses_taught = []

    def assign_teaching(self, course):
        self.courses_taught.append(course)


class Tutor(Person, LearnerMixin, TeacherMixin):
    def __init__(self, *args, **kwargs):
        super(Tutor, self).__init__(*args, **kwargs)

jane = Tutor("Jane", "Smith", "SMTJNX045")
jane.enrol(a_postgrad_course)
jane.assign_teaching(an_undergrad_course)
```