Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

In [1]:
```
# sample class example
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def sit(self):
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        print(f"{self.name} rolled over!")
```

In [ ]:
```
type("a")
```

In [ ]:
```
dir("a")
```

In [ ]:
```
"a".__class__
```

In [9]:
```
# to create a class, use the `class` keyword
# class names should be capitalized


class MyClass:
    x = 5
```

In [ ]:
```
# now we can use the class to create objects
# objects are instances of a class
p1 = MyClass()
print(p1.x)
```

The `__init__()` Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

All classes have a function called `__init__()` , which is always executed when the class is being initiated.

Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created:

In [ ]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


p1 = Person("John", 36)
p2 = Person("Jane", 25)
```

```
print(p1.name)
print(p1.age)
print(p2.name)
print(p2.age)
```

In [ ]:

```
id("a")
```

In [12]:

```
# classess can have common attributes


class Employee:
    retirement_age = 50

    def __init__(self, employee_name, employee_id):
        self.name = employee_name
        self.id = employee_id
        self.salary = None
        self.remaining_years = None
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

In [ ]:

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

    def myfunc(self):
        print("Hello my name is " + self.name)


p1 = Person("John", 36)
p1.myfunc()
```

In [ ]:

```
# the self parameter is a reference to the current instance of the class


class Person:
    def __init__(myobject, name, age):
        myobject.name = name
        myobject.age = age

    def myfunc(abc):
        print("Hello my name is " + abc.name)


p1 = Person("John", 36)
p1.myfunc()
```

In [ ]:

```
# modify object properties
p1.name = "James"
p1.myfunc()
```

In [ ]:

```
# delete object properties
del p1.name
p1.myfunc()
```

In [ ]:

```
# delete objects
del p1
```

Python 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.

In [ ]:

```
# example parent class
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)


p = Person("John", "Doe")
p.printname()
```

In [9]:

```
# example child class


class Student(Person):
    pass
```

In [ ]:

```
s = Student("Mike", "Olsen")
s.printname()
```

In [ ]:

```
# add the __init__() function to the child class


class Student(Person):
    def __init__(self, fname, lname):
        pass
```

When you add the `__init__()` function, the child class will no longer inherit the parent's `__init__()` function. To keep the inheritance of the parent's `__init__()` function, add a call to the parent's `__init__()` function:

In [ ]:

```
class Student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)
```

In [ ]:

```
# use the super() function
# super() function will make the child class inherit all the methods and properties f
class Student(Person):
    def __init__(self, fname, lname):
        super().__init__(fname, lname)
```

In [11]:

```
# add properties
class Student(Person):
    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year

    def welcome(self):
        print(
            "Welcome",
            self.firstname,
            self.lastname,
            "to the class of",
            self.graduationyear,
        )
```

In [ ]:
```
s = Student("Mike", "Olsen", 2019)
s.welcome()
```

In [ ]:
```
# hidden properties
# can be accessed inside the class
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def myfunc(self):
        print("Hello my name is " + self.name, "and I am", self.__age)

    def get_age(self):
        return self.__age

    def set_age(self, age):
        self.__age = age


p1 = Person("John", 36)
p1.myfunc()
# print(p1.__age) # this will raise an error
# p1.get_age()
```

In [ ]:
```
# hidden methods
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def __myfunc(self):
        print("Hello my name is " + self.name, "and I am", self.__age)

    def myfunc(self):
        self.__myfunc()


p1 = Person("John", 36)
# p1.__myfunc()  # this will raise an error
p1.myfunc()
```

In [ ]:
```
# Why we need classes in Python?
# Without using classes


def calculateGPA(gradeDict):
    return sum(gradeDict.values()) / len(gradeDict)


# defining students is not efficient without classes
john = {"age": 12, "gender": "male", "level": 6, "grades": {"math": 3.3}}
jane = {"age": 12, "gender": "female", "level": 6, "grades": {"math": 3.5}}
students = {"john": john, "jane": jane}

print(calculateGPA(students["john"]["grades"]))
print(calculateGPA(students["jane"]["grades"]))
```

```
# adding new grade is not easy without classes
# to add a grade we need to update the student dictionary
john.update({"grades": {"math": 3.3, "science": 3.5}})
```

In [ ]:

```
# with using classes


class Student(object):
    def __init__(self, name, age, gender, level, grades=None):
        self.name = name
        self.age = age
        self.gender = gender
        self.level = level
        self.grades = grades or {}

    def setGrade(self, course, grade):
        self.grades[course] = grade

    def getGrade(self, course):
        return self.grades[course]

    def getGPA(self):
        return sum(self.grades.values()) / len(self.grades)


# Define some students
john = Student("John", 12, "male", 6, {"math": 3.3})
jane = Student("Jane", 12, "female", 6, {"math": 3.5})

# Now we can get to the grades easily
print(john.getGPA())
print(jane.getGPA())

# We can add new courses for John very easily
john.setGrade("science", 3.2)
```

In [ ]:

```
# a class example of a basic calculator
class Calculator:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b

    def subtract(self):
        return self.a - self.b

    def multiply(self):
        return self.a * self.b

    def divide(self):
        return self.a / self.b


calc = Calculator(10, 5)
print(f"Add: {calc.add()}")
print(f"Subtract: {calc.subtract()}")
print(f"Multiply: {calc.multiply()}")
print(f"Divide: {calc.divide()}")
```

In [ ]:

```
# extending the calculator class
# to add more functions to the calculator
# with inheritance
class AdvancedCalculator(Calculator):
    def power(self):
        return self.a**self.b

    def square_root(self):
        return self.a**0.5


calc = AdvancedCalculator(10, 5)
print(f"Add: {calc.add()}")
print(f"Subtract: {calc.subtract()}")
print(f"Multiply: {calc.multiply()}")
print(f"Divide: {calc.divide()}")
print(f"Power: {calc.power()}")
print(f"Square Root: {calc.square_root()}")
```

In [7]:

```
# a least sqauares polynomial regression class example
import numpy as np
import matplotlib.pyplot as plt


class PolynomialRegression:
    def __init__(self, x, y, degree, num_plot_points=100):
        self.x = x
        self.y = y
        self.x_plot = np.linspace(min(self.x), max(self.x), num_plot_points)
        self.degree = degree
        self.coefficients = np.polyfit(self.x, self.y, self.degree)
        self.poly = np.poly1d(self.coefficients)

    def plot(self):
        fig = plt.figure(figsize=(10, 6))
        plt.scatter(self.x, self.y, label="Data", color="blue")
        plt.plot(self.x_plot, self.poly(self.x_plot), color="red", label="Polynomial"
        plt.xlabel("x")
        plt.ylabel("y")
        plt.title("Polynomial Regression")
        plt.legend()
        plt.show()
        plt.close(fig)
```

In [ ]:

```
x = np.linspace(0, 5, 20)
y = 2.9 * x**3 - 4.8 * x**2 + 3.5 * x + 2.1
degree = 3
poly_reg = PolynomialRegression(x, y, degree)
print(f"Coefficients:\n{poly_reg.coefficients}")
poly_reg.plot()
```

In [ ]:

```
x2 = np.linspace(0, 5, 20)
y2 = np.sin(x2 * np.pi / 2) + np.random.normal(0, 0.1, 20)
degree = 6
poly_reg2 = PolynomialRegression(x2, y2, degree)
print(f"Coefficients:\n{poly_reg2.coefficients}")
poly_reg2.plot()
```