Classes and Objects

Object Oriented Programming (OOP)

- Python is an object-oriented programming language
  - A programming language based on objects.
- OOP allows to write *classes* that represent real-world things
  - Based on these classes we can make objects
- Classes allow us to group data and functions on that data in one place.
  - Makes our program more organized and easier to use

Classes and Objects

- Class is a blueprint/template to define objects of a certain type. It defines common attributes/properties and functions/behavior of objects.
  - Variables of a class are called attributes.
  - Functions defined in a class are called methods.
  - For example, Cars can have common attributes such as manufacturer, model, production year, type, mileage, color, etc. Cars can have common methods such as start, stop, accelerate, turn right/left, etc. Therefore, we can make a class called Car.

- Object is an instance of a class.
  - For example, an object is car1 with man_name =“Kia”, model =“Rio”, year=2020, etc.
  - car2 is another object of class Car with name=“Toyota”, model =“Corolla, year = 2018, etc.

Classes and Objects

```
car1_manuf = “Kia”
car1_year = 2015
car1_color =“red”
…
def move(car1_mileage,):
    car1_mileage = car1_mileaxge + x
```

- If we need to define another car, e.g., car2, then we need to do the same thing again.
- Or we can use a dictionary such as {“car1”:[“Kia”,2015,”red”,..], “car2”:[“Toyota”,2016,”white”,..],..}

- Instead we can create a class called Car and put all these data and methods that can process this data in that class
  - Makes our program more organized and easier to use

How to define a class

- We use the class keyword to define a class.

```
class ClassName:     # Start each word with a capital letter. Do not separate words with
                     # underscores. This style is called camel case
    Statement1
    Statement2
    …
    StatementN
```

For example,

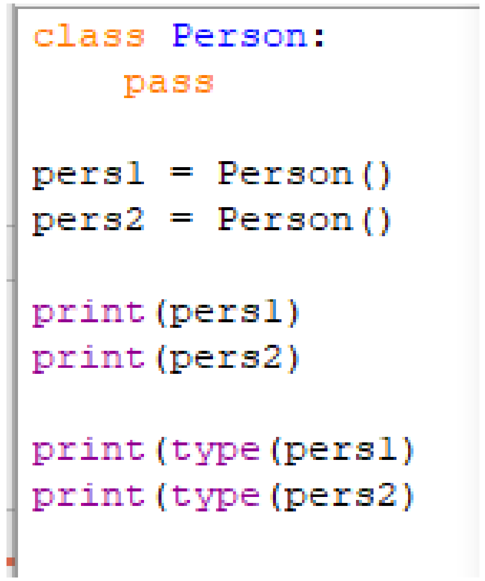
```
class Person:    #  we made a new type called Person
    pass         # means - leave the body empty. Otherwise it shows an error
```

Making instances

pers1 = Person() # Instantiation – making an object from a class

pers2 = Person()   # To instantiate we use function notation - ()

- pers1 and pers2 are different instances of the Person class

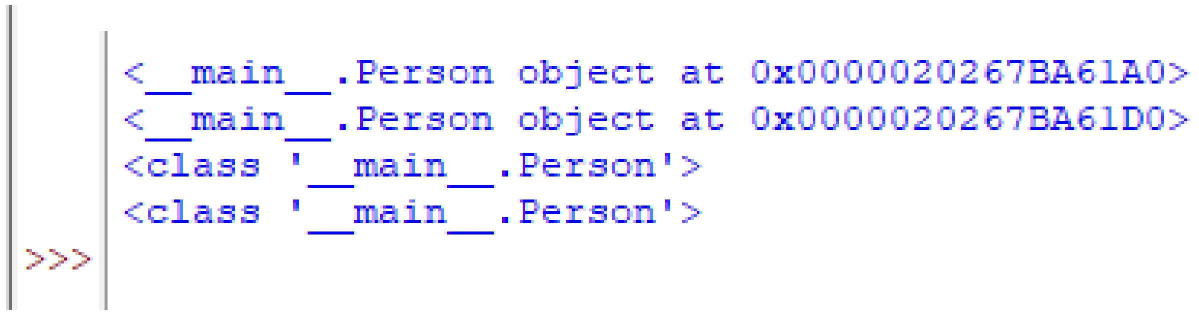


```
class Person:
    pass

pers1 = Person()
pers2 = Person()

print(pers1)
print(pers2)

print(type(pers1))
print(type(pers2))
```



```
<__main__.Person object at 0x0000020267BA61A0>
<__main__.Person object at 0x0000020267BA61D0>
<class '__main__.Person'>
<class '__main__.Person'>
>>>
```

Making instances

```
pers1 =  Person()
pers2 = Person()
```

One of the ways of making/accessing *attributes* is as follows:

```
pers1.nid = 123123
pers1.name = “Abdef”
pers1.age = 23
```

Like in using module variables, we use the dot notation.

__init__ method

- Every class has a special method called __init__(), which is used to initialize objects. If defined, class instantiation calls __init__ method.

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

Example

```
pers1 =  Person(“Abdef”, 20, 123123) # when creating we do not pass
pers2 = Person(“Adiya”, 15, 234234)   # self argument
```

```
print(pers1.id, pers1.name)
print(pers2.id, pers2.name)
```

```
>>> 123123 Abdef
>>> 234234 Adiya
```

self

- Reference to the current instance of the class. Can take a different name, but by convention it is called *self*.

  def __init__(*self*, a, b, c):

- Using this instance we can access attributes and methods that belong to the class.

  self.a = 12

  self.b = “asd”

- Self should be the first argument of a method.

  🚫 def processIt(a, b, self, c)

Methods

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

	def getInfo(self):
		return str(self.id)+ “ name: “+ self.name + “ age: “ + str(self.age)

pers = Person(“A”,1,111)
print(pers.getInfo())        # call methods using the dot notation
```

Modifying Attribute Values

- Attribute values can be modified directly by using the dot notation

```
pers = Person(“A”,1, 111)
pers.age = 2
```

- They can also be modified through a method

```
class Person:
    …
    def setAge(self, x):
        if x < 0:
            print(“Age cannot be negative”)
```