Classes and 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.

Create a Class

To create a class, use the keyword class:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
====================

5
```

The __init__() Function

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

To understand the meaning of classes we have to understand the built-in **__init__( )** function.

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:

Example

Create a class named Person, use the **__init__( )** function to assign values for name and age:

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

p1 = Person("John", 36)

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

```
=======================

John
36
```

Note: *The **__init__( )** function is called automatically every time the class is being used to create a new object.*

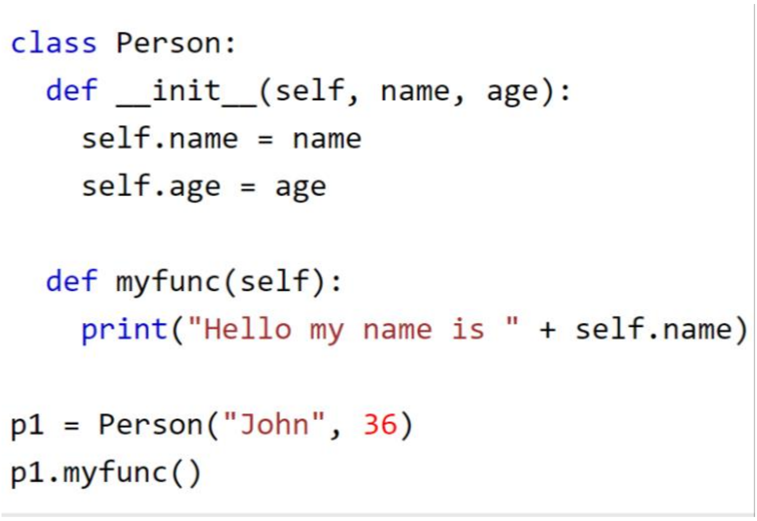
Object Methods

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

Let us create a method in the Person class:

Example

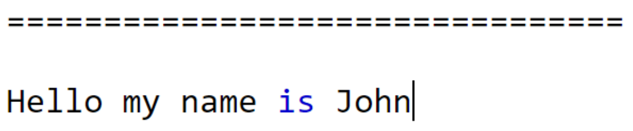
Insert a function that prints a greeting, and execute it on the p1 object:



```python
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()
```



```
===================================

Hello my name is John
```

The self Parameter

The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named **self** , you can call it whatever you like, but it has to be the first parameter of any function in the class:

It does not have to be named **self** , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the word *mysillyobject* and *abc* instead of *self*:

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

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

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

```
============
Hello my name is John
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

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

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

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
==============
40
```

Delete Object Properties

You can delete properties on objects by using the `del` keyword:

Example

Delete the age property from the p1 object:

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

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

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
AttributeError: 'Person' object has no attribute 'age'
```

Delete Objects

You can delete objects by using the **del** keyword:

Example

Delete the p1 object:

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

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

p1 = Person("John", 36)

del p1

print(p1)
================
NameError: 'p1' is not defined
```

The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the **pass** statement to avoid getting an error.

```
class Person:
  pass

# having an empty class definition
#like this, would raise an
#error without
#the pass statement
```