Day 22: Python Classes

- 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.

```
# Syntax
class MyClass:
  x = 5
# Class created with the name MyClass with property named x.
```

- Now we can use the class named MyClass to create objects.

```
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.
- 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.

```
# Create a class named Person, use the __init__() function to assign values for na
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

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

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

The `__str__()` Function

- The `__str__()` function controls what should be returned when the class object is represented as a string.
- If the `__str__()` function is not set, the string representation of the object is returned.

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

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

Object Methods

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

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def __str__(self):
    return f"{self.name}({self.age})"

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

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

The self Parameter

- The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong 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.

```
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()
```

Modify Object Properties

```
p1.age = 40
```

Delete Object Properties

```
del p1.age
```

Delete Objects

```
del p1
```

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
```