Cheatsheets / **Introduction to Python Data Structures and Loops**

Classes

Python repr method

The Python `__repr__()` method is used to tell Python what the *string representation* of the class should be. It can only have one parameter, `self`, and it should return a string.

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

  def __repr__(self):
    return self.name

john = Employee('John')
print(john) # John
```

Python class methods

In Python, *methods* are functions that are defined as part of a class. It is common practice that the first argument of any method that is part of a class is the actual object calling the method. This argument is usually called **self**.

```
# Dog class
class Dog:
  # Method of the class
  def bark(self):
    print("Ham-Ham")

# Create a new instance
charlie = Dog()

# Call the method
charlie.bark()
# This will output "Ham-Ham"
```

Instantiate Python Class

In Python, a class needs to be instantiated before use.
As an analogy, a class can be thought of as a blueprint (Car), and an instance is an actual implementation of the blueprint (Ferrari).

```
class Car:
  "This is an empty class"
  pass

# Class Instantiation
ferrari = Car()
```

codecademy

Python Class Variables

In Python, class variables are defined outside of all methods and have the same value for every instance of the class.
Class variables are accessed with the `instance.variable` or `class_name.variable` syntaxes.

```
class my_class:
  class_variable = "I am a Class Variable!"

x = my_class()
y = my_class()

print(x.class_variable) #I am a Class Variable!
print(y.class_variable) #I am a Class Variable!
```

Python init method

In Python, the `.__init__()` method is used to initialize a newly created object. It is called every time the class is instantiated.

```
class Animal:
  def __init__(self, voice):
    self.voice = voice

# When a class instance is created, the instance variable
# 'voice' is created and set to the input value.
cat = Animal('Meow')
print(cat.voice) # Output: Meow

dog = Animal('Woof')
print(dog.voice) # Output: Woof
```

Python type() function

The Python `type()` function returns the data type of the argument passed to it.

```
a = 1
print(type(a)) # <class 'int'>

a = 1.1
print(type(a)) # <class 'float'>

a = 'b'
print(type(a)) # <class 'str'>

a = None
print(type(a)) # <class 'NoneType'>
```

codecademy

Python class

In Python, a class is a template for a data type.
A class can be defined using the `class` keyword.

```
# Defining a class
class Animal:
  def __init__(self, name,
number_of_legs):
    self.name = name
    self.number_of_legs = number_of_legs
```

Python dir() function

In Python, the built-in `dir()` function, without any argument, returns a list of all the attributes in the current scope.
With an object as argument, `dir()` tries to return all valid object attributes.

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

  def print_name(self):
    print("Hi, I'm " + self.name)


print(dir())
# ['Employee', '__builtins__',
'__doc__', '__file__', '__name__',
'__package__', 'new_employee']

print(dir(Employee))
# ['__doc__', '__init__', '__module__',
'print_name']
```

__main__ in Python

In Python, `__main__` is an identifier used to reference the current file context. When a module is read from standard input, a script, or from an interactive prompt, its `__name__` is set equal to `__main__`.
Suppose we create an instance of a class called `CoolClass`. Printing the `type()` of the instance will result in:

```
<class '__main__.CoolClass'>
```

This means that the class `CoolClass` was defined in the current script file.

Print Share

codecademy