Short python tutorial

Within this tutorial, we'll go over some python code.

We're currently working in a Jupyter notebook, a place for running code easily. All of the code we type will go in a "cell" just like the "code" we're typing right now. There are few different types of cells but the most important for us will be a "code cell" and a "markdown cell". A code cell will contain and run python code. A markdown cell will contain text and information.

In [7]:
```
# comment - always comment your so when you go back you can remember what
# code does. A comment always starts with a "#" just like this line.
```

In [ ]:
```
# printing - helpful for finding mistakes in your code and seeing a value
# you just computed

print("Hello, world!")
```

In [2]:
```
# variables - way to store data in memory

a = 10
#print(a)

name = "Jesse"
#print(name)

# python is case sensitive - meaning that "name" is not the same as "Name"
# print(Name) # will yell at you (give you an error)
```

Data types - different kinds of data that can be stored

In [5]:
```
# numbers

a = 5
b = 1.5

print("a is of type", a)
print("b is of type", b)
```

```
a is of type 5
b is of type 1.5
```

In [12]:
```
# list - ordered sequence of items

l1 = [1, 5.2, "Mouad"]  #lists can have differeny types of data in them

l2 = [4, 10, 6, -1, 0, 5]

#accessing elements of a list
print(l1[1])


#accessing multiple elements of a list
print("accessing elements 3 onwards from l2:", l2[3:])
```

```
5.2
accessing elements 3 onwards from l2: [-1, 0, 5]
```

In [13]:
```
# strings - sequence of characters (letter or other symbols)

s1 = "Hi my name is Afia"

printStatement = "Hello world!"
print(printStatement)
```

```
Hello world!
```

In [16]:
```
# Set - unordered collection of unique items

set1 = {5, 4, 3, 10}
set1 = {5, 4, 3, 10, 5}
print(set1) # will print {10, 3, 4, 5} because the elements of the set
            # must be unique
```

```
{10, 3, 4, 5}
```

In [25]:
```
# Dictionary - unordered collection colletion of key-value pairs

dict1 = {
    "ominous": "creepy or spooky",
    "pristine": "prefectly clean",
    "evade": "to avoid"
}

#print(dict1["pristine"])


ages = {
    "Gk": 20,
    "Mouad": 19,
    "Jesse": 19
}

print("Jesse is", ages["Jesse"])
print("Jesse is", ages["Jesse"], "years old")
```

```
Jesse is 19
Jesse is 19 years old
```

In [33]:
```
# Booleans - scary word, but just means variables that are only true or false

Monica = True
Dominic = False

print(Monica)
```

```
True
```

Math in coding

In [26]:
```
# elementary operation

x = 2 + 3
y = 4 - 3
z = 2 / 3
h = 2 ** 3 # raised to a power
print(x, y, z, h)
```

```
5 1 0.6666666666666666 8
```

In [34]:
```
# Comparison operator (greater than, less than, greater than or equal,
# equal to, not equal to, ...)

print(1 > 2)    # greater than
print(1 >= 2)   # greater than or equal to
print(1 < 2)    # less than
print(1 <= 2)   # less than or equal to
print(1 == 2)   # equal to
print(1 != 2)   # not equal to

# The "type" of each of these operations are boolean
```

```
False
False
True
True
False
True
False
```

In [37]:
```
# Useful way to use comparison operators

a = (1 == 2)
print(a)
b = (1 != 1)
print(b)
```

```
False
False
```

In [43]:
```
# Bitwise operators (and, or, not) - allows us to combine comparison operators
# or combine booleans

tf = True or False
print(tf)


isThursday = True
isJune = True

# if you want to check if you are in a Thursday during June,
tf = isThursday and isJune
print(tf)
```

```
True
True
```

Cheatsheet for bitwise operators:

`and` : returns True if statement1 and statement2 are True, returns False otherwise
syntax: statement1 `and` statement2

`or` : returns True if statement1 or statement2 are True, returns False otherwise
syntax: statement1 `or` statement2

`not` : returns opposite of statement
syntax: `not` statement

In [4]:
```
# Example usage of bitwise operators

tf = True and False
print("True and False outputs:", tf)

tf = True or False
print("True or False outputs:", tf)

tf = not True
print("not True outputs:", tf)
```

```
True and False outputs: False
True or False outputs: True
not True outputs: False
```

More on bitwise operators in the next lecture