Introduction to Python

Goals:

- Learn basic Python operations
- Understand differences in data structures
- Get familiarized with conditional statements and loops
- Learn to write custom functions and import python modules

Main Reference: McKinney, Wes. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media. Kindle Edition

Indentation

Python code is structured by indentation (tabs or spaces) instead of braces which is what other languages normally use. In addition, a colon (:) is used to define the start of an indented code block.

In [ ]:
```
for x in list(range(7)):
    print("The number per loop...")
    print(x)
    if x > 2:
        print("The number is greater than 2")
        print("----------------------")
```

Everything is an Object

- Everything in Python is considered an object.
- A string, a list, a function and even a number is an object.
- For example, you can define a variable to reference a string and then access the methods available for the string object.
- If you press the tab key after the variable name and period, you will see the methods available for it.

In [ ]:
```
a = "pedro"
```

In [ ]:
```
a.capitalize()
```

Variables

In Python, when you define/create a variable, you are basically creating a reference to an object (i.e string,list,etc). If you want to define/create a new variable from the existing variable, you will be creating another reference to the original object rather than copying the contents of the first variable to the second one.

In [ ]:
```
a = [1,2,3]
b = a
b
```

Therefore, if you update the original variable (a), the new variable (b) will automatically reference the updated object.

In [ ]:
```
a.append(4)
b
```

A variable can have a short name (like x and y) or a more descriptive name (age, dog, owner). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Reference:https://www.w3schools.com/python/python_variables.asp

In [ ]:
```
dog_name = 'Pedro'
age = 2
is_vaccinated = True
birth_year = 2015
```

In [ ]:
```
is_vaccinated
```

In [ ]:
```
dog_name
```

Data Types

As any other object, you can get information about its type via the built-in function type().

In [ ]:
```
type(age)
```

In [ ]:
```
type(dog_name)
```

In [ ]:
```
type(is_vaccinated)
```

Combining variables and operations

In [ ]:
```
x = 6
y = 10
```

In [ ]:
```
x-y
```

In [ ]:
```
x*y
```

In [ ]:
```
y/x
```

In [ ]:
```
y**x
```

In [ ]:
```
x-y
```

In [ ]:
```
x==y
```

In [ ]:
```
y>=x
```

Binary Operators and Comparisons

In [ ]:
```
2<4
```

In [ ]:
```
5>6
```

In [ ]:
```
5>3
```

The Respective Print statement

In [ ]:
```
print("Hello Haikt")
```

Control Flows

References:

- https://docs.python.org/3/tutorial/controlflow.html
- https://docs.python.org/3/reference/compound_stmts.html#the-if-statement

If,elif,else statements

- The if statement is used for conditional execution
- It selects exactly one of the suites by evaluating the expressions one by one until one is found to be true; then that suite is executed.
- If all expressions are false, the suite of the else clause, if present, is executed.

In [ ]:
```
print("x = " + str(x))
print("y = " + str(y))
```

In [ ]:
```
if x==y:
    print('yes')
else:
    print('no')
```

- An if statement can be optionally followed by one or more elif blocks and a *catch-all* else block if all of the conditions are False :

In [ ]:
```
if x==y:
    print("They are equal")
elif x > y:
    print("It is grater than")
else:
    print("None of the conditionals were true")
```

Loops

For

The for statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object.

In [ ]:
```
my_dog_list=['Pedro',2,True,2015]
```

In [ ]:
```
for i in range(0,100):
    print(i*10)
```

While

A while loop allows you to execute a block of code until a condition evaluates to false or the loop is ended with a break command.

In [ ]:
```
i = 1
while i <= 5:
    print(i ** 2)
    i += 1
```

In [ ]:
```
i = 1
while i > 2:
    if i > 10:
        break
    print(i ** 2)
    i += 1
```

Data structures

References:

- https://docs.python.org/3/tutorial/datastructures.html
- https://python.swaroopch.com/data_structures.html

Lists

- Lists are data structures that allow you to define an ordered collection of items.
- Lists are constructed with square brackets, separating items with commas: [a, b, c].
- Lists are mutable objects which means that you can modify the values contained in them.
- The elements of a list can be of different types (string, integer, etc)

In [ ]:
```
my_dog_list=['Pedro',2,True,2015]
```

In [ ]:
```
my_dog_list[0]
```

In [ ]:
```
my_dog_list[2:4]
```

In [ ]:
```
print("My dog's name is " + str(my_dog_list[0]) + " and he is " + str(my_dog_list[1]) + " years old.")
```

- The list data type has some more methods an you can find them here.
- One in particular is the `list.append()` which allows you to add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

In [ ]:
```
my_dog_list.append("tennis balls")
```

In [ ]:
```
my_dog_list
```

- You can modify the list values too:

In [ ]:
```
my_dog_list[1] = 4
my_dog_list
```

Dictionaries

- Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays".
- Dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.
- It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary).
- A pair of braces creates an empty dictionary: {}.
- Remember that key-value pairs in a dictionary are not ordered in any manner. If you want a particular order, then you will have to sort them yourself before using it.

In [ ]:
```
my_dog_dict={'name':'Pedro','age':3,'is_vaccinated':True,'birth_year':2015}
```

In [ ]:
```
my_dog_dict
```

In [ ]:
```
my_dog_dict['age']
```

In [ ]:
```
my_dog_dict.keys()
```

In [ ]:
```
my_dog_dict.values()
```

Tuples

- A tuple consists of a number of values separated by commas
- On output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses, although often parentheses are necessary anyway (if the tuple is part of a larger expression).

In [ ]:
```
my_dog_tuple=('Pedro',2,True,2015)
```

In [ ]:
```
my_dog_tuple
```

- Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing.
- Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

In [ ]:
```
my_dog_tuple[1]
```

Slicing

You can select sections of most sequence types by using slice notation, which in its basic form consists of start:stop passed to the indexing operator []

In [ ]:
```
seq = [ 7 , 2 , 3 , 7 , 5 , 6 , 0 , 1 ]
seq[ 1 : 5 ]
```

Functions

Functions allow you to organize and reuse code blocks. If you repeat the same code across several conditions, you could make that code block a function and re-use it. Functions are declared with the **def** keyword and returned from with the **return** keyword:

In [ ]:
```
def square(n):
    return n ** 2
```

In [ ]:
```
print("Square root of 2 is " + str(square(2)))
```

In [ ]:
```
number_list = [2,2,3,4,5]
```

In [ ]:
```
for number in number_list:
    sn = square(number)
    print("Square root of " + str(number) + " is " + str(sn))
```

Modules

References:

- https://docs.python.org/3/tutorial/modules.html#modules

- If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost.
- Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead.
- Let's say we define two functions:

In [ ]:
```
def square(n):
    return n ** 2

def cube(n):
    return n ** 3
```

- You can save the code above in a file named math_ops.py. I created the file for you already in the current folder.
- All you have to do is import the **math_ops.py** file

In [ ]:
```
import math_ops
```

In [ ]:
```
for number in number_list:
    sn = square(number)
    cn = cube(number)
    print("Square root of " + str(number) + " is " + str(sn))
    print("Cube root of " + str(number) + " is " + str(cn))
    print("----------------------")
```

- You can get a list of the current installed modules too

In [ ]:
```
help('modules')
```

- Let's import the **datetime** module.

In [ ]:
```
import datetime
```

- Explore the **datetime** available methods. You can do that by typing the module name, a period after that and pressing the **tab** key or by using the the built-in function **dir()** as shown below:

In [ ]:
```
dir(datetime)
```

- You can also import a module with a custom name

In [ ]:
```
import datetime as dt
```

In [ ]:
```
dir(dt)
```

Out[ ]:
```
['MAXYEAR',
'MINYEAR',
'__builtins__',
'__cached__',
'__doc__',
'__file__',
'__loader__',
'__name__',
'__package__',
'__spec__',
'date',
'datetime',
'datetime_CAPI',
'time',
'timedelta',
'timezone',
'tzinfo']
```

In [ ]:
```

```