Kernel: Python 3 (system-wide)

85%

Rowan Forgot To generate pdf file: -10 -5

Homework 2: Intro to Python

In this lab you will be familiarizing yourself with the programming language called Python.

Feel free to execute and modify the code cell. You should experiment with **Python** statements to understand the language.

Python is a so-called high level programming language. Which means it is very abstracted from machine languages and are easily readable by humans.

Python is one of the most popular programming languages nowadays, and is used in various spheres from research and Data Science to Machine Learning and Artificial Intelligence.

In this worksheet you will also learn some of the most popular **Python** libraries such as `Pandas`, `Numpy`, and `Matplotlib`. Libraries are used to make **Python** more efficient. By itself (solely with built-in functions and methods), Python is not a very useful language, hence people keep building libraries (sets of functions and methods) that can be used together with **Python** in order to make it more useful.

Python's Data Types

Numerals

One of the data types that is commonly used in **Python** are numerals (aka numbers). With numerals, **Python** can be used as a calculator. Run the cells below (and modify them as you wish) to see it for yourself.

In [1]:
```
3+5
```

Out[1]: 8

In [2]:
```
140-20
```

Out[2]: 120

In [3]:
```
40/2
```

Out[3]: 20.0

In [0]:
```
2*2
```

Python follows the same arithmetic rules as we do. So guess what the output of the next cell will be:

In [4]:
```
2*5+1
```
Out[4]: 11

What about this cell:

In [5]:
```
2*(5+1)
```
Out[5]: 12

Guess what a double star notation does? Run the cell below to see if you had the right intuition about it.

In [6]:
```
3**2
```
Out[6]: 9

There are a few types of numerals: `integers` and `floats`.

They are not compatible, so it is important to remember which one you are using in order to avoid bugs. Floats use decimal points, so it will be easy for you to spot them. See how the output changes when we print the same number as an `integer` and as a `float`.

In [7]:
```
int(1776)
```
Out[7]: 1776

In [8]:
```
float(1776)
```
Out[8]: 1776.0

Variables

To store the values in **Python** (and in many other programming languages) we need to use variables. Variables can store all kinds of information. This is useful if you plan to use the same values later on in your project. It will save you time and make your code more readable. To see how we store values in variables run the cell below.

In [9]:
```
burr = "Wait for it!"
burr
```
Out[9]: 'Wait for it!'

In [10]:
```
burr
```

Out[10]:
```
'Wait for it!'
```

If you don't assign any value to the variable and call it later, it will give you an error. We can see it in the cell below. (It will give you an error, but it was planned that way)

In [11]:
```
y = 8
z = x+y
```

Out[11]:
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_428/2809545446.py in <cell line: 2>()
      1 y = 8
----> 2 z = x+y

NameError: name 'x' is not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-33c5b6ca9af3> in <module>()
      1 y = 8
----> 2 z = x+y

NameError: name 'x' is not defined
```

Here, we did not assign x to anything, and since x was not defined, we cannot add its value to y, so this produces an error.

Strings

The phrases and words in Python are always stored inbetween double or single quotation marks. They are formally called strings.

In [0]:
```
"History has its eyes on you"
```

Also, you can use both single and double quotes for the strings, but sometimes you need to use single quotes inside of the string and that might cause a problem (it will "exit it out").

You can just always use the double quotation marks on the outside to avoid such problem. Or, you can use a backslash before the single quotation mark. It will tell Python to ignore it and not to exit it out. You can see an example of it below:

In [12]:
```
print("What'd I miss")
print('You\'ll be back')
```

Out[12]:
```
What'd I miss
You'll be back
```

The strings can also be "added" onto one another. Run the cell below to see how it works.

In [13]:
```
"York"+"town"
```

Out[13]:
```
'Yorktown'
```

When we are using **Jupyter Notebooks**, the last line is printed out automatically (unless it is performing some other operation, eg. getting assigned to a specific value).

But if you would like to print more than just the last line, you can use the `print( )` method. If you just run the next cell, it will not give you any output.

In [14]:
```
aaron = "Ev'ryone give it up for America's favorite fighting
Frenchman"
everyone = "Lafayette!"
mdl = "I'm takin this horse by the reins"
```

QUESTION 1

Try using print( ) in the cell below to output all the three lines of lyrics of "Guns and Ships".

Hint: you'll need to use it three times.

In [16]:
```
## Write your answer to the question in this cell
print(aaron)
print(everyone)
print(mdl)
```

Out[16]:
```
Ev'ryone give it up for America's favorite fighting Frenchman
Lafayette!
I'm takin this horse by the reins
```

A string is a sequence of characters. If you want to see how many characters your string has, use the len() method. Do you think spaces are considered characters in Python? Check if your intuition was correct by running the cell below.

In [17]:
```
len(burr)
```

Out[17]:
```
12
```

QUESTION 2

Now create a variable name in which you will save your full name. Note that you need to call the variable once again in order to get an output.

In [21]:
```
## Write your answer to the question in this cell
name = "Rowan Flynn"
name
```

Out[21]:
```
'Rowan Flynn'
```

QUESTION 3

Sometimes you need to switch the values in two variables. In the cell below we have the last names mixed up. The variable theodosia should have an output "burr" and philip should return "hamilton". How would you approach this problem without manually retyping the outputs?

Hint Use a temporary variable in which you can temporarily save the value of one of the other variables.

In [23]:
```
## Answer the question here by completing the code below
theodosia = "hamilton"
philip = "burr"

##fill these in, Version 1:
theodosia, philip = ..

# Version 2
temp = theodosia
theodosia = philip
philip = temp
```

Out[23]:
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_428/737499921.py in <cell line: 6>()
      4
      5 ##fill these in, Version 1:
----> 6 theodosia, philip = "test"
      7
      8 # Version 2
ValueError: too many values to unpack (expected 2)
```

Lists

You have learnt about the strings and numerals, it's time to take it up a notch. Another useful and very popular data type is a **Python** list. Lists can take various data types (both numerals, strings, arrays, other lists, etc).

To create an empty list just make your variable equal to " ", like so:

In [0]:
```
washington = []
washington
```

To add the values into your list you can either use lst.append( ), lst.insert( ), or lst.extend( ).

In [0]:
```
washington.extend(["Right Hand Man"])
washington
```

In [0]:
```
washington.insert(0, "I need my")
washington
```

In [0]:
```
washington.append("Back")
washington
```

Notice, that the `insert` method takes two positional values. You need to specify the index in the list at which it will be inserted, whilst append just adds the value to the end of the list. `extend` takes another list as its input. If you try to use a string or a number by itself, it will give you an error.

When you need to access an element at a specific index, you can use the following code. The code below returns the second element of the list.

In [0]:
```
washington[1]
```

Note: Python starts counting at 0, not 1.

Sometimes you just need to use part of the list. In this case you can use ":" to modify your list. Let's make a new list:

In [0]:
```
lst = [1,2,3,4,5]
```

Sometimes you just need to use part of the list or copy the whole list into a new variable. In this case you can use ":" to modify your list. The notation is:

a = b[start:stop:step]

start: the index you want to start with. (the default is 0)

stop: the index you want to end with. (the default is the last number)

step: use it, if you wish to skip some indeces (eg. use only every other value). For example, if you start with index 1 and step is 2, you will iterate through the 1st index, then 3rd, then 5th, etc. until you get to the stopping index. (the default is 1)

Note that python will stop iterating at the value before the last index. See the following example:

In [0]:
```
lst[0:5:1]
```

Note: if you wish to start iterating from the end of a list backwards, we use a negative step:

In [25]:
```
lst[::-1]
```

Out[25]:
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_428/3894335210.py in <cell line: 1>()
----> 1 lst[::-1]

NameError: name 'lst' is not defined
```

In [26]:
```
ham_songs = ["Alexander Hamilton", "Aaron Burr, Sir", "My Shot", \
             "The Story of Tonight", "The Schuyler Sisters", "Farmer Refuted",
             "You'll be Back", "Right Hand Man", "A Winter's Ball", \
             "Helpless", "Satisfied"]

every_other = ham_songs[::2]
every_other
```

Out[26]:
```
['Alexander Hamilton',
 'My Shot',
 'The Schuyler Sisters',
 "You'll be Back",
 "A Winter's Ball",
 'Satisfied']
```

In [0]:
```
lyrics = ["I", "am", "not", "throwing", "away", "my", "shot"]
lyrics
```

Let's say you want to save only the two last words from your old list into your new list. That's what you will need to do then:

In [0]:
```
alex = lyrics[-2:]
alex

# alternatively you can use lyrics[5:]
```

Notice also that if you want to learn the length of your list, you can use the same method "len( )", but it will not count all the characters anymore. Instead, it will count all the elements in it. Like so:

In [0]:
```
len(washington)
```

QUESTION 4)

Create a list with the values 1, 2, 3, 4, and 5 and only get the values 2 and 4

In [28]:
```
## Write your answer to the question in this cell
test = [1,2,3,4,5]
test[1::2]
```

Out[28]:
```
[2, 4]
```

Built-in Functions

Although Python doesn't have a lot of functions and methods by itself, it is not like it doesn't have any. Let's go over some of the most valuable built-in funsctions in Python.

In [0]:
```
min(1, 3)
```

In [0]:
```
max(15, 25, 70)
```

You can also use these functions with strings. Can you guess what the output will be?

In [29]:
```
max("Hamilton", "Washington")
```

Out[29]: 'Washington'

So does Python know that George Washington was older than Alexander Hamilton? Of course not, when comparing strings, Python goes of off the letters of alphabet. Since "W" goes after "H" in the alphabet, "Washington" is a value that is "bigger" than "Hamilton".

QUESTION 5)

We have created two lists with different integers. Let's now find the biggest number among the two smallest numbers in two lists. In other words, let's find the max( ) of the two min( ). You can achieve it with either 3 or 1 lines of code.

In [34]:
```
## Answer the question here by completing the code below
dob = [1757, 1756, 1732, 1737, 1754]
dod = [1804, 1836, 1799, 1793, 1782]

##fill in
min_dob = min(dob)
min_dod = min(dod)
max_of_mins = max(min_dob,min_dod)


# you can do it in one line

max_mins = max(min(dob), min(dod))

print(max_of_mins)
print(max_mins)
```

Out[34]:
```
1782
1782
```

Another valuable function you can use with your numerals is "round". It will round your floats to the nearest integer. Like so:

In [35]:
```
round(8.7)
```

Out[35]: 9

Another useful function we can use with numbers is "abs". It outputs an absolute value of a number:

In [36]:
```
abs(-3.5)
```

Out[36]: 3.5

Conditionals and For-loops

Conditionals

Conditionals are also known as **booleans**. With conditional statements we can let the computer know when (aka under which condition) we want a specific operation to be executed. Under the hood the computer evaluates if the condition is True and performs a specific operation (like print( ), return( ), etc). Some of the most popular conditional statements are:

A == B: True if A equals B

A != B: True if A is not equal to B

A > B: True if A is greater than B. Same syntax for "<"

A >= B: True if A is greater than or equal to B. Same syntax for "<="

We have three types of statements in the **Python** conditionals: `if` , `elif` , and `else` . We always start with "if", we always end with "else". The "elif" statement is optional. It literally means "else if". While both "if" and "else" can only be used once, "elif" can be used multiple times, and let you add many more conditions to your statement.

If the condition in the `if` case is False, then we move to the `elif` case. If the `elif` case is false, then we will move to the else case and perform whatever is inside the else condition. We move into the condition that has a True conditional value first and evaluate that.

In [37]:
```
playwright = "Lin-Manuel Miranda"

if playwright == "Lin-Manuel Miranda":
    print (True)
else:
    print (False)
```

Out[37]: True

Note: "=" and "==" are not the same operations in **Python**. A single equals sign (=) assignes the value to the right of it to the variable name to the left of it. While the double equals sign (==) compares if the value on the right is equal to the value to the left of it.

In the next line, let's ask **Python** to print the name of the school two of the Founding Fathers went based on their name. Feel free to change the string our "name" variable is assigned to.

In [39]:
```
name = "Burr"

# Feel free to change the value of the variable "name"

if name == "Hamilton":
    print("Arcadia")
elif name == "Burr":
    print("Princeton")
else:
    print("I don't know")
```

Out[39]: Princeton

In the cell below guess what the output will be, before you run it. Feel free to change the numeral under the variable "year".

In [40]:
```
year = 1789

# Who's the president

if year>=1789 and year<1797:
    print("George Washington")
elif year>=1797:
    print("John Adams")
else:
    print("King George")
```

Out[40]: George Washington

We can also search through lists of values in our conditional statements to see if a word is in our list using the keyword 'in':

In [42]:
```
founding_fathers = ["Jefferson", "Hancock", "Hamilton", "Adams",
"Washington", "Burr", "Madison"]

# change the name in "any_name"
if "Jefferson" in founding_fathers:
    print("You are not throwing away your shot!")
else:
    print("You've got to be in the room where it happens!")
```

Out[42]: You are not throwing away your shot!

Loops

Loops (aka for-loops) allow for some code to be executed repeatedly. For example, if you wish to print the numbers from 0 to 10, you can do that with the for-loop.

A for-loop iterates through a sequence of elements (list, string, array, etc.) and reassigns an element (which can have any name, most common are "x", "i", "elem", the name in itself doesn't matter) to each element of the sequence sequentially.

In [43]:
```
for elem in ['a', 'b', 'c']:
    print(elem)

elem
```

Out[43]:
```
a
b
c
'c'
```

That is why in the previous code cell we can see that when an "elem" is called at the end of the loop, it is equal to "c" which is the last element of the sequence.

In the cell below we will be using the built-in function called range( ). It enumerates the numbers from 0 up to a number you put in (exclusive of that last number).

Note: remember that Python starts to count from 0, hence the last number you specify is not going to be included.

In [44]:
```
for i in range(11):
    print(i)
```

Out[44]:
```
0
1
2
3
4
5
6
7
8
9
10
```

QUESTION 6)

Let's combine what you know so far about lists, conditionals, and for loops. Create a list with values 1, 2, 3, 4, 5. Iterate through the list with a for loop, and print only the values that are greater than 2. Otherwise, print the statement "This value is not greater than 2."

In [49]:
```
## Write your answer to the question in this cell
lst = [1,2,3,4,5]
for elem in lst:
    if elem > 2:
```

```
        print(elem)
    else:
        print('This value is not greater than 2.')
```

Out[49]:
```
This value is not greater than 2.
This value is not greater than 2.
3
4
5
```

Working with Libraries

It is time for us to introduce a concept of **Python** libraries.

As we mentioned before, **Python** is not a very powerful programming language by itself. What makes it powerful are the libraries people wrote for it. Libraries are the sets of different functions that you can import and use. The two most popular libraries are numpy and pandas.

Since we are using **Cocalc**, a lot of libraries have been preloaded for you. You just need to "call" them. Just run the cell below. It will make our notebook know that we will be using these preinstalled libraries from that point on.

In [50]:
```
import pandas as pd
import numpy as np
```

With the help of numpy we can introduce a new data type - arrays.

Arrays are commonly used with Data Frames (basically, tables of values). There are a lot of functions and methods that can be used with arrays when one is analysing their data. The main difference between the lists and arrays is that an array can take only one type of data (eg. only numerals, or strings, but never both).

To create an array with values, just make your variable equal to np.array("your value")

In [0]:
```
arr = np.array(10)
arr
```

A very useful method that can be used with an array of numbers is np.arange( ). It takes at least 2, sometimes 3 positional arguments. The first number will be the starting point of your array, the last one will identify up to which number your array will go (not inclusive). The third positional argument is optional, it shows how many steps an array should skip through. (This is very similar to lists)

For example, if you need an array with numbers from 0 to 20, but you only want for it to include every other number, you will need add a third positional argument.

In [51]:
```
every_other = np.arange(0, 21, 2)
every_other
```

Out[51]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

You can also convert a list of numbers in Python into an array:

In [52]:
```
num_lst = [1, 2, 3, 4, 5]
num_arr = np.array(num_lst)
num_arr
```

Out[52]:
```
array([1, 2, 3, 4, 5])
```

You can do some arithmetic with arrays. Guess what will be the output of this line of code before you run it:

In [53]:
```
num_arr*2
```

Out[53]:
```
array([ 2,  4,  6,  8, 10])
```

As you can see, all the values in our array got mutliplied by 2.

Try performing the same operation with the initial list of numbers instead. What do you think the output will be in the cell below?

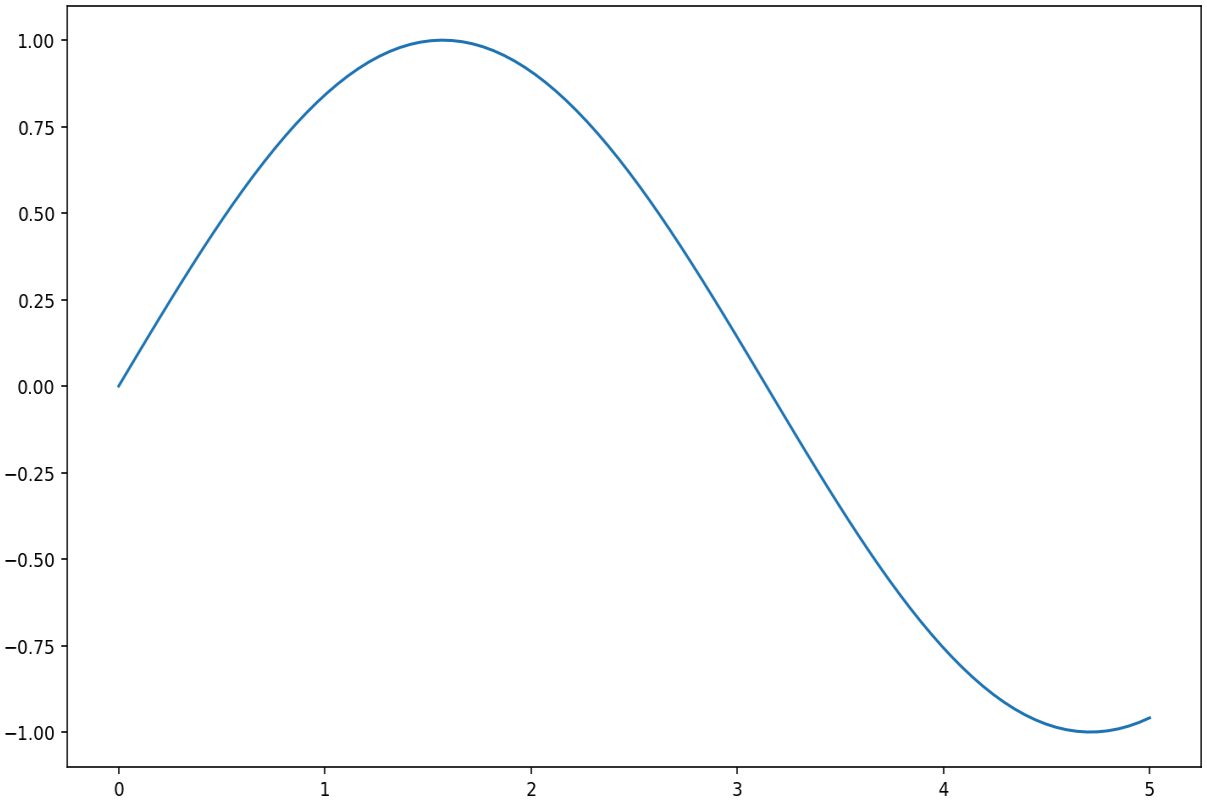
As you can see, lists and arrays not only can be used differently, but they also give different outputs when we are using the same operations on them.

Plots

To do visualizations, we will need to import yet another library.

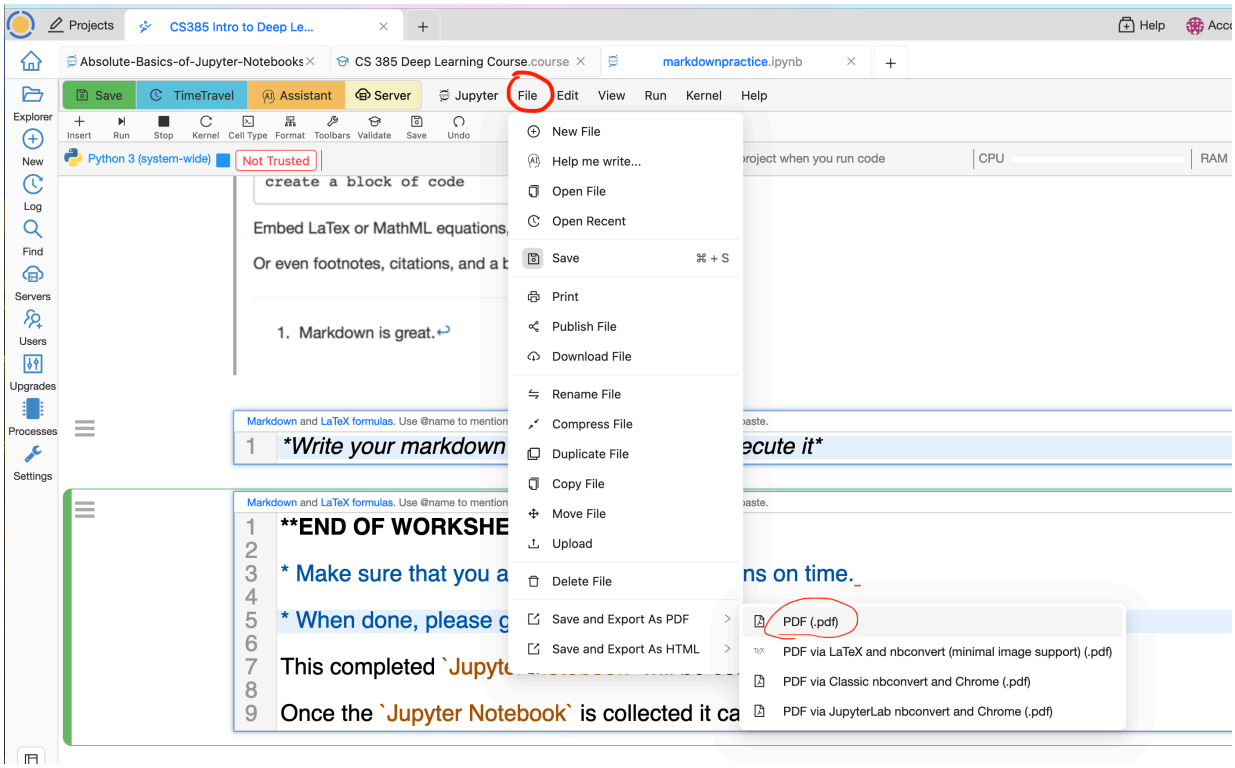
In [54]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 5, 100)  # Adjust range and number of points as
needed
y = np.sin(x)

plt.plot(x, y)
plt.show()
```

Out[54]:



END OF WORKSHEET

- Make sure that you answered all the questions.
- When done, please go to the `File` option for Cocalc and select export as a pdf:



RECALL: This completed `Jupyter Notebook Assigment` will be collected and graded.

Once the `Jupyter Notebook Assigment` is collected it can not be modified.

In [0]: