Course Title:

Data Analysis

Course DA-L2-021

Introduction to Python Programming (for Data Science projects)

Session Topics

1. Introduction to Python Programming
2. Working with data frames in Python

Introduction to Python Programming

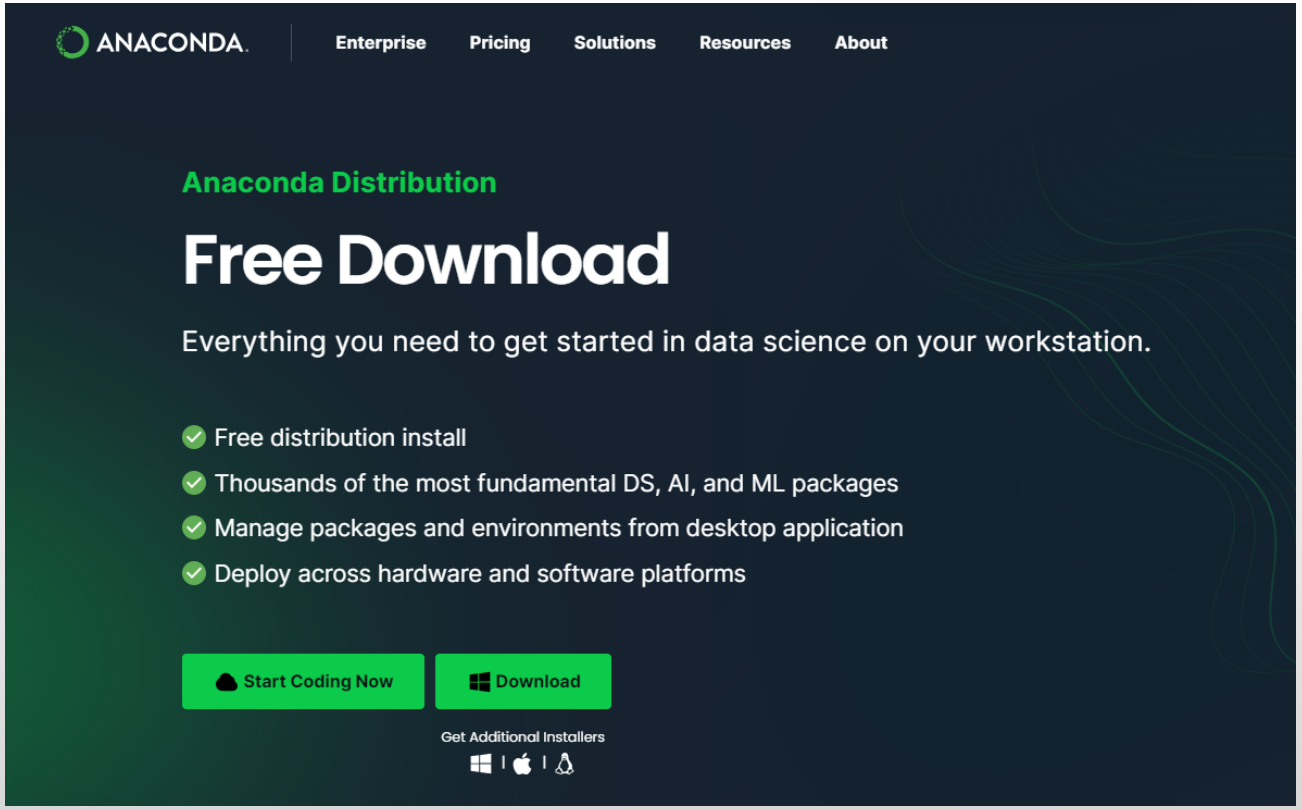
What is Python ?

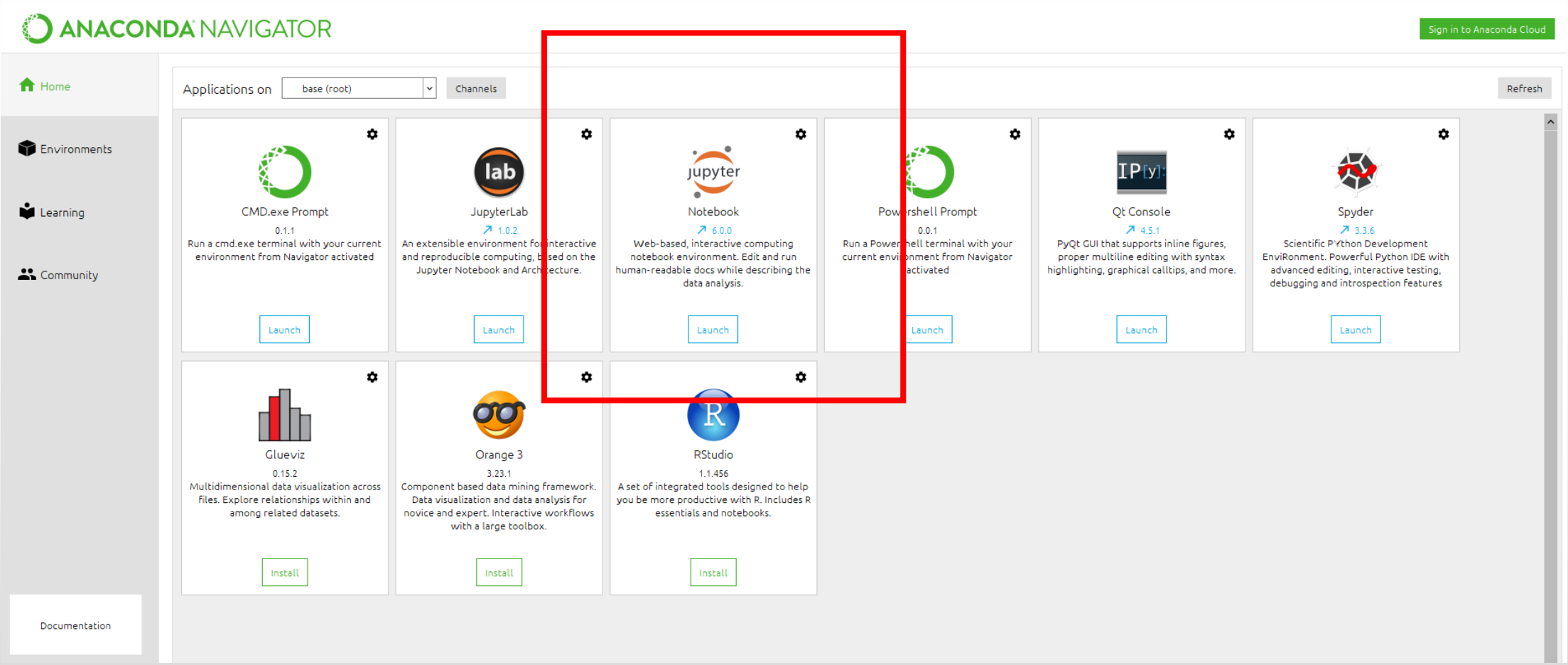
Python is a high-level, object-oriented programming language. Most beginners in the development field prefer Python as one of the first languages to learn because of its simplicity and versatility. It is also well supported by the community and keeps up with its increasing popularity.

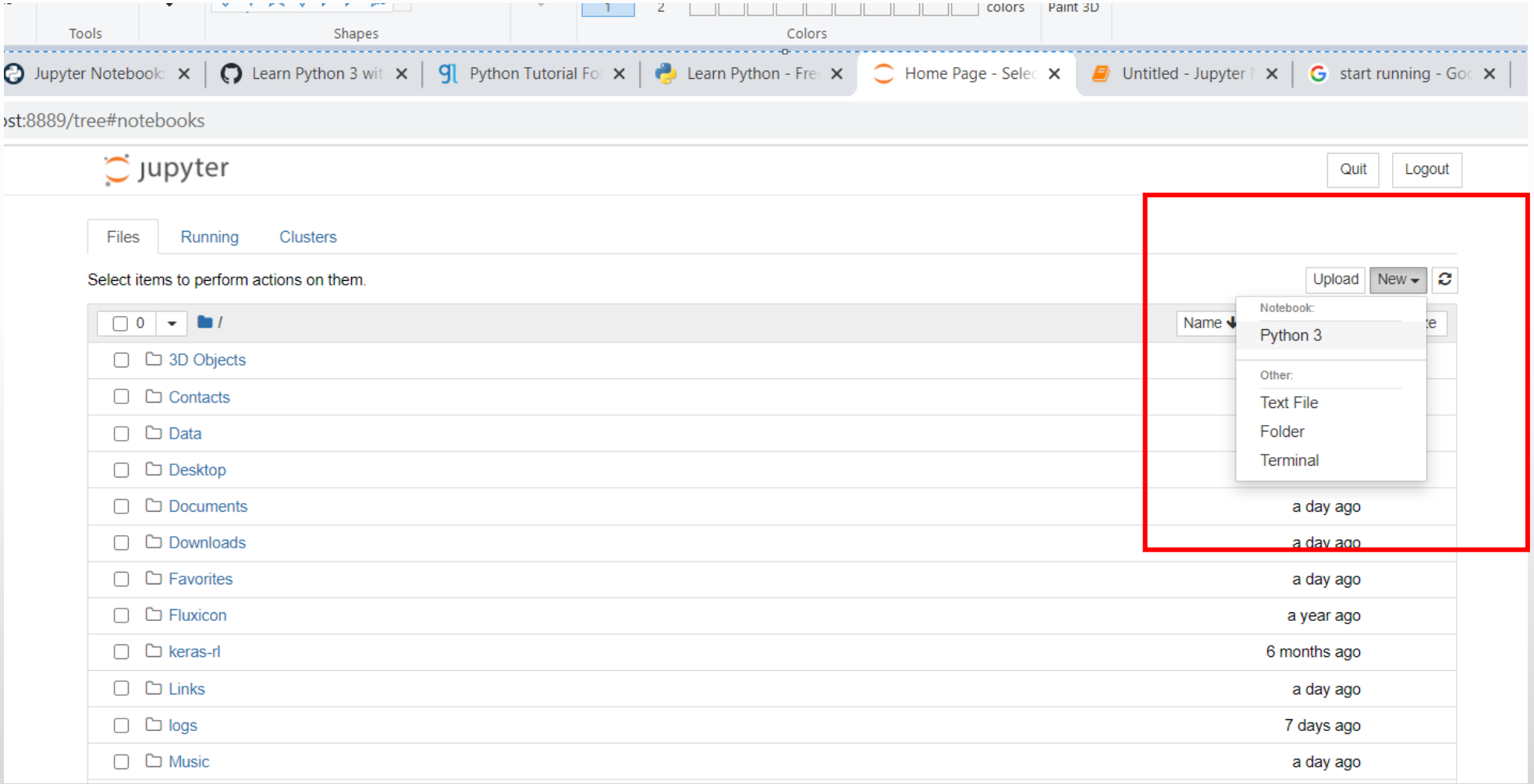
7 Reasons Why You Should Use Python

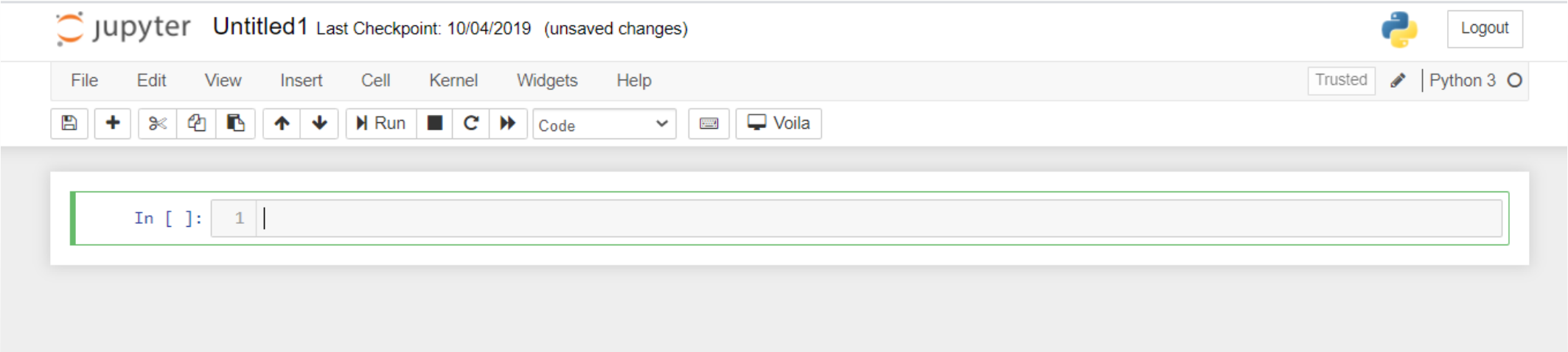
What are the top Python IDEs?

How to install Anaconda? https://www.anaconda.com/download







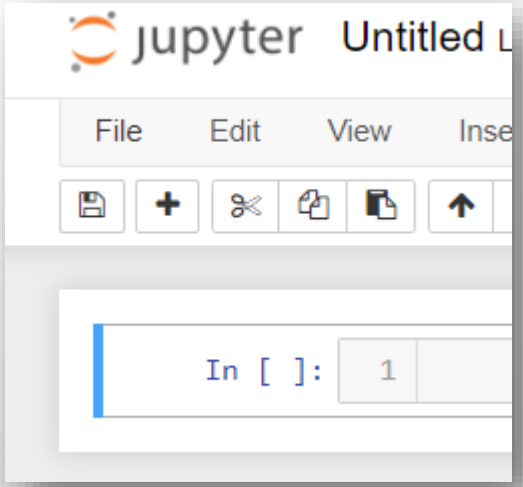


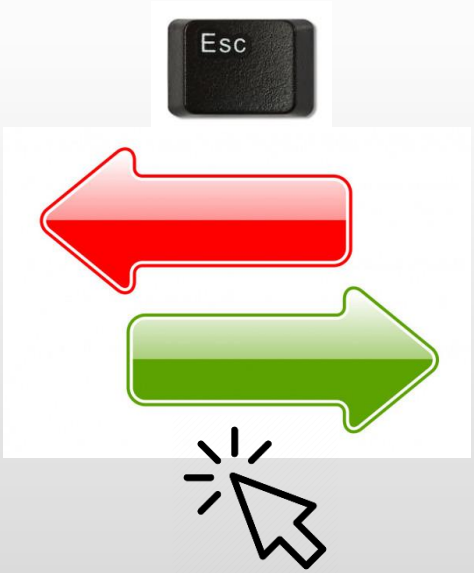
Jupyter Notebooks have two different keyboard input modes:

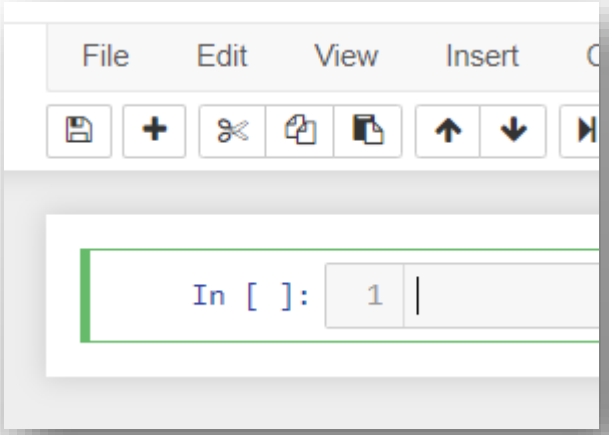
Command mode vs. **Edit mode**

binds the keyboard to notebook level actions. Indicated by a grey cell border with a blue left margin.

when you're typing in a cell. Indicated by a green cell border







Short keys [Command Mode] :

- `shift` + `enter` run cell, select below
- `ctrl` + `enter` run cell
- `option` + `enter` run cell, insert below
- `A` insert cell above
- `B` insert cell below
- `C` copy cell
- `V` paste cell
- `D` , `D` delete selected cell
- `shift` + `M` merge selected cells, or current cell with cell below if only one cell selected
- `I` , `I` interrupt kernel
- `0` , `0` restart kernel (with dialog)
- `Y` change cell to `code` mode
- `M` change cell to `markdown` mode (good for documentation)

View all keyboard shortcuts

`H` (in Command mode)

Format Text In Jupyter Notebook With Markdown:

- `M` change cell to `markdown` mode

Section Headers

You can create a heading using the pound (#) sign. For the headers to render properly, there must be a space between the (#) and the header text.

```
## Heading Two

### Heading Three

#### Heading Four
```

Lists

You can also use `Markdown` to create lists using the following syntax:

```
* This is a bullet list
* This is a bullet list
* This is a bullet list
```

Bold and Italicize

You can also use `**` to bold or `*` to italicize words. To bold and italicize words, the symbols have to be touching the word and have to be repeated before and after the word using the following syntax:

```
*These are italicized words, not a bullet list*
**These are bold words, not a bullet list**

* **This is a bullet item with bold words**
* *This is a bullet item with italicized words*
```

You can find more details here: https://www.earthdatascience.org/courses/intro-to-earth-data-science/file-formats/use-text-files/format-text-with-markdown-jupyter-notebook/

Your first program: “Hello World”

```
In [1]: 1 print("Hello world")
        Hello world
```

Lab Activity #1

- Open the Academy – Python Intro file
- Add an introduction using Markdown with following info:
  - Your full name
  - Current Date
  - Current Semester
- Write a Line of python code to print “Hello World”

Variables:

Python variables are containers to hold data. These assigned data value to a variable can be changed at a later stage.

The first assignment of value to a variable creates the variable. There is no explicit declaration of variable.

```
In [3]:
numOfBoxes = 12
ownerName = "Sofia"
print("numOfBoxes= ", numOfBoxes)
print("ownerName= ", ownerName)
```

```
numOfBoxes=  12
ownerName=  Sofia
```

Variable names follow these conventions:

- Variable name begins with letter or underscore character
- Alpha-numeric and underscores are allowed in the rest of name
- Variable names are case-sensitive

Numerical Variables:

In [5]:
```
#To define an integer, use the following syntax:

myint = 7
print(myint)

#To define a floating point number, you may use one of the following notations:

myfloat = 8.0
print(myfloat)
myfloat = float(8)
print(myfloat)
```

```
7
8.0
8.0
```

String Variables:

In [6]:
```
#Strings are defined either with a single quote or a double quotes.

mystring = 'hello'
print(mystring)
mystring = "hello"
print(mystring)
```

```
hello
hello
```

Variable assignment:

In [10]:
```
a=6
b=7

print("a:" , a , " b:" , b)
```

```
a: 6  b: 7
```

In [11]:
```
#Assignments can be done on more than one variable "simultaneously" on the same line like this

a, b = 9,10

print("a:" , a , " b:" , b)
```

```
a: 9  b: 10
```

Lists:

Lists are very similar to arrays. They can contain any type of variable, and they can contain as many variables as you wish. Lists can also be iterated over in a very simple manner. Here is an example of how to build a list.

In [13]:

```
# Lists

mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0]) # prints 1
print(mylist[1]) # prints 2
print(mylist[2]) # prints 3

print ("=============")

# prints out 1,2,3
for x in mylist:
    print(x)
```

```
1
2
3
=============
1
2
3
```

Lab Activity #2

- In this exercise, you will need to add numbers and strings to the correct lists using the "append" list method. You must add the numbers 1,2, and 3 to the "numbers" list, and the words 'hello' and 'world' to the string variable.
- You will also have to fill in the variable second_name with the second name in the names list, using the brackets operator []. Note that the index is zero-based, so if you want to access the second item in the list, its index will be 1.

In [ ]:

```python
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]

# write your code here
second_name = None
a

# this code should write out the filled arrays and the second name in the names list (Eric).
print(numbers)
print(strings)
print("The second name on the names list is %s" % second_name)
```

Basic Operators:

```python
#Arithmetic Operators

number = 1 + 2 * 3 / 4.0
print(number)
```

```
2.5
```

```python
remainder = 11 % 3
print(remainder)

```

```
2
```

```python
#Using two multiplication symbols makes a power relationship.

squared = 7 ** 2
cubed = 2 ** 3
print(squared)
print(cubed)
```

```
49
8
```

Basic Operators:

```python
# Python also supports multiplying strings to form a string with a repeating sequence:

lotsofhellos = "hello " * 10
print(lotsofhellos)
```

```
hello hello hello hello hello hello hello hello hello hello
```

```python
# Using Operators with Lists

even_numbers = [2,4,6,8]
odd_numbers = [1,3,5,7]
all_numbers = odd_numbers + even_numbers
print(all_numbers)
```

```
[1, 3, 5, 7, 2, 4, 6, 8]
```

```python
# Just as in strings, Python supports forming new lists with a repeating sequence using the multiplication operator:

print([1,2,3] * 3)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Conditions:

Python uses boolean logic to evaluate conditions. The boolean values True and False are returned when an expression is compared or evaluated. For example:

Notice that:

Variable assignment is done using a single equals operator "=", whereas

comparison between two variables is done using the double equals operator "==".

The "not equals" operator is marked as "!=".

Conditions

```
x = 2
print(x == 2) # prints out True
print(x == 3) # prints out False
print(x < 3) # prints out True
```

```
True
False
True
```

Conditions:

Boolean operators

The **"and"** and **"or"** boolean operators allow building complex boolean expressions, for example:

```python
name = "John"
age = 23
if name == "John" and age == 23:
    print("Your name is John, and you are also 23 years old.")

if name == "John" or name == "Rick":
    print("Your name is either John or Rick.")
```

```
Your name is John, and you are also 23 years old.
Your name is either John or Rick.
```

The "in" operator

could be used to check if a specified object exists within an iterable object container, such as a list:

```python
# the "in" opeartor

name = "John"
if name in ["John", "Rick"]:
    print("Your name is either John or Rick.")
```

```
Your name is either John or Rick.
```

Conditions:

If ... elif

Python uses indentation to define code blocks, instead of brackets. The standard Python indentation is 4 spaces, although tabs and any other space size will work, as long as it is consistent. Notice that code blocks do not need any termination.

```python
statement = False
another_statement = True

if statement is True:
    # do something

    print("Statement is TRUE!")
    pass
elif another_statement is True: # else if
    # do something else
    print("Another Statement is TRUE!")
    pass
else:
    # do another thing
    pass
```

Conditions:

The "not" operator
Using "not" before a boolean expression inverts it:

```
print(not False) # Prints out True
print((not False) == (False)) # Prints out False
```

```
True
False
```

Lab Activity #3

- Change the variables in the first section, so that each if statement resolves as True.

```python
# change this code
number = 10
second_number = 10
first_array = []
second_array = [1,2,3]

if number > 15:
    print("1")

if first_array:
    print("2")

if len(second_array) == 2:
    print("3")

if len(first_array) + len(second_array) == 5:
    print("4")

if first_array and first_array[0] == 1:
    print("5")
```

Loops:

The **"for"** loop
For loops iterate over a given sequence. Here is an example:

```python
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

```
2
3
5
7
```

"while" loops
While loops repeat as long as a certain boolean condition is met. For example:

```python
# Prints out 0,1,2,3,4

count = 0
while count < 5:
    print(count)
    count += 1  # This is the same as count = count + 1
```

```
0
1
2
3
4
```

Loops:

"break" and "continue" statements

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement. A few examples:

```python
# Prints out 0,1,2,3,4

count = 0
while True:
    print(count)
    count += 1
    if count >= 5:
        break

# Prints out only odd numbers - 1,3,5,7,9
for x in range(10):
    # Check if x is even
    if x % 2 == 0:
        continue
    print(x)
```

```
0
1
2
3
4
1
3
5
7
9
```

Lab Activity #4

- Write a loop to calculate the factorial for the value stored in variable inputVar, and print the result at the end.

Working with data frames in Python

How to use Python libraries?

Most of the power of a programming language is in its (software) libraries.

•A *(software) library* is a collection of files (called *modules*) that contains functions for use by other programs.

- May also contain data values (e.g., numerical constants) and other things.
- Library's contents are supposed to be related, but there's no way to enforce that.

•The Python standard library is an extensive suite of modules that comes with Python itself.

How to use Python libraries?

A program must import a library module before using it.

Use import to load a library module into a program's memory.

You can also import specific items from a library module to shorten programs.

Use *from ... import ...* to load only specific items from a library module.
Then refer to them directly without library name as prefix.

How to use Python libraries?

Some examples:

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt

In [2]: import math
        print("The value of pi is", math.pi)

        The value of pi is 3.141592653589793
```

15 Python Libraries for Data Science You Should Know

Data Mining

1. Scrapy
2. BeautifulSoup

Data Processing and Modeling

3. NumPy
4. SciPy
5. Pandas
6. Keras
7. SciKit-Learn
8. PyTorch
9. TensorFlow
10. XGBoost

Data Visualization

11. Matplotlib
12. Seaborn
13. Bokeh
14. Plotly
15. pydot

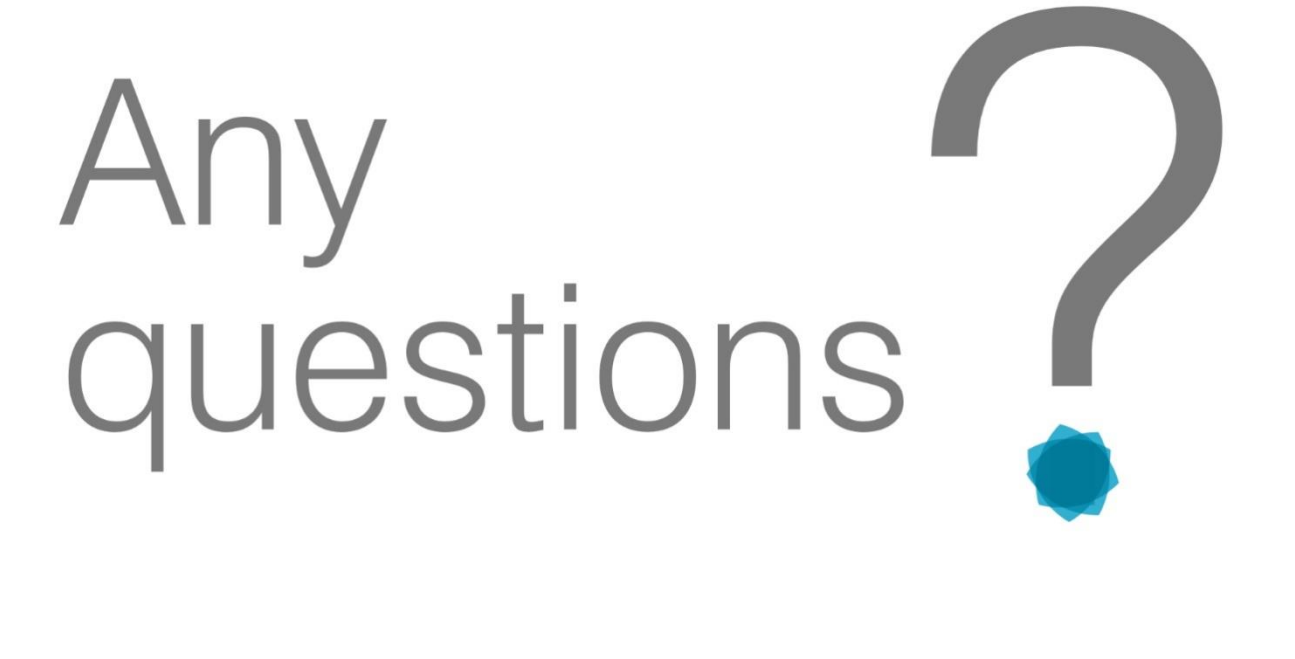
https://www.dataquest.io/blog/15-python-libraries-for-data-science/

Lab Activity #5

- Load the “Loans_DataSet.csv” dataset into Python and do a simple data exploration

References

1. https://www.learnpython.org/
2. https://www.mygreatlearning.com/blog/python-tutorial-for-beginners-a-complete-guide/
3. http://maxmelnick.com/2016/04/19/python-beginner-tips-and-tricks.html
4. https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/



vala@data-corner.com