Introduction to Python

Department of Applied Mathematics and Scientific Computing

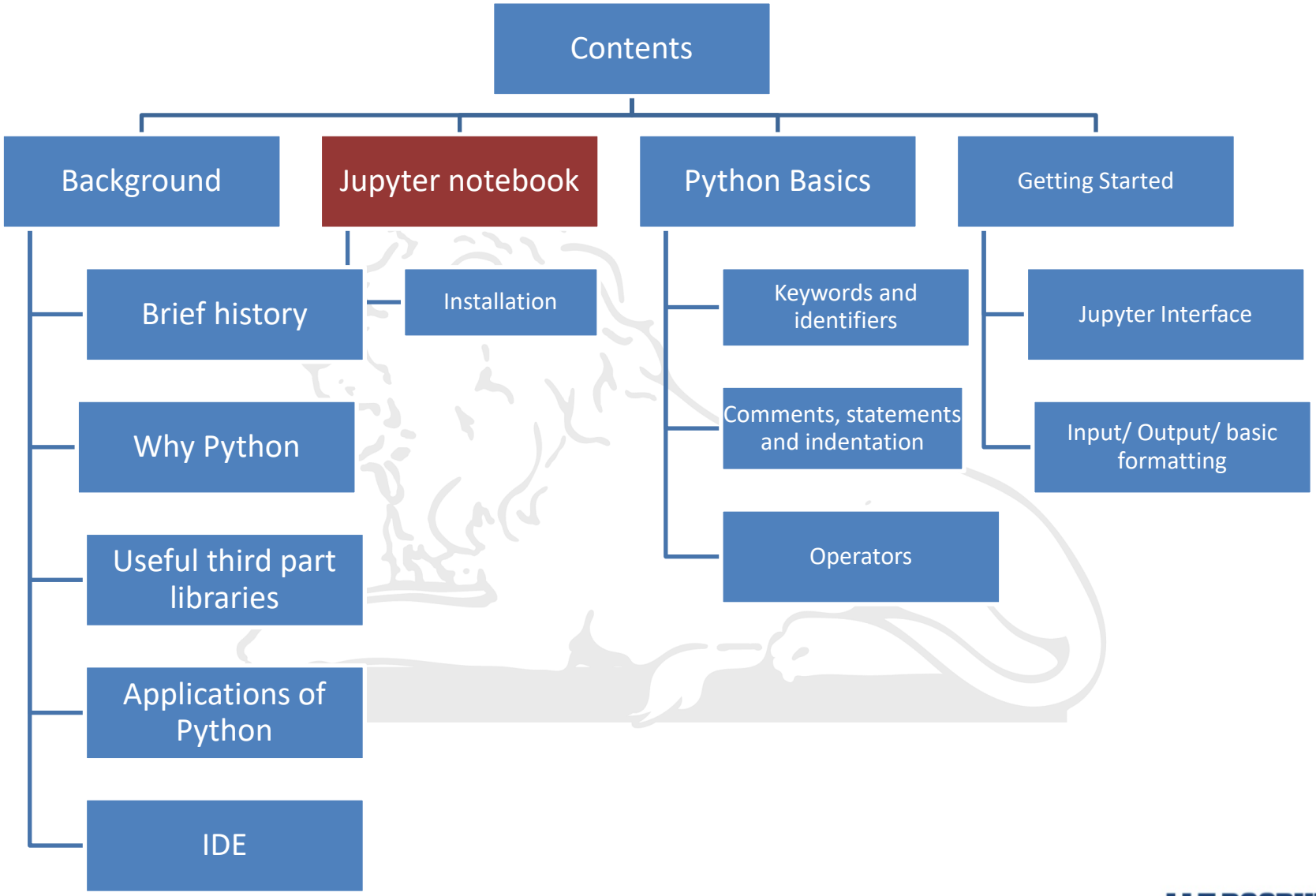
And

MFSDSAI

Millie Pant

pant.milli@as.iitr.ac.in

Content


Contents
Background
Jupyter notebook
Python Basics
Getting Started
Brief history
Installation
Why Python
Useful third part libraries
Applications of Python
IDE
Keywords and identifiers
Comments, statements and indentation
Operators
Jupyter Interface
Input/ Output/ basic formatting

Background: A very brief history

- Developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science, Netherlands.

- 1994 – Python 1.0
- 2000 -Python 2.0
- 2008 – Python 3.0

- Programming languages that influenced Python:
  - ABC language.
  - Modula-3
  - C
  - C++
  - ….
  - ….

Python: Why should I use it ?

- Beginners language
- Open source
- Interpreted
- Smaller code
- Syntactically simple
- Dynamically typed
- Extensive Library

Useful third party libraries

Python has several well defined libraries that assists in Scientific Computing, Data Analysis, Machine Learning, Statistical Computing, Visualization and many more

- NumPy
- Pandas
- Matplotlib
- Scikit Learn
- Tensor Flow

Core Python means Python without any additional libraries

Usage of Python

- Desktop and Web Applications
- Data Science Machine Learning and Artificial Intelligence
- Scientific Computing
- Robotics
- Gaming

Python in Action

Web Development Data Analytics Scientific Computing Games
Desktop Apps ……

IDE (Integrated Development Environment)

- An IDE is a software that provides programmers with an interphase combined with all the tools at hand.
- Selection of the right IDE influences the productivity and effectiveness of Python Programming.

IDEs for Python-

- IDLE
- Spyder
- PyCharm
- Atom
- Microsoft Visual Studio Code
- PyDev
- **Jupyter Notebook**
- Kite

And many more…

Jupyter Notebook as IDE

- Jupyter is an open-source, interactive development environment for Python including an editor and comes in package with **Anaconda**.
- It is **one of the best Python IDE** that supports for **Numerical simulation, data cleaning, machine learning, data visualization, and statistical modeling**.
- Combine code, text, and images.
- Support for many programming languages.
- Integrated data science libraries (NumPy, Scipy, Pandas, matplotlib).

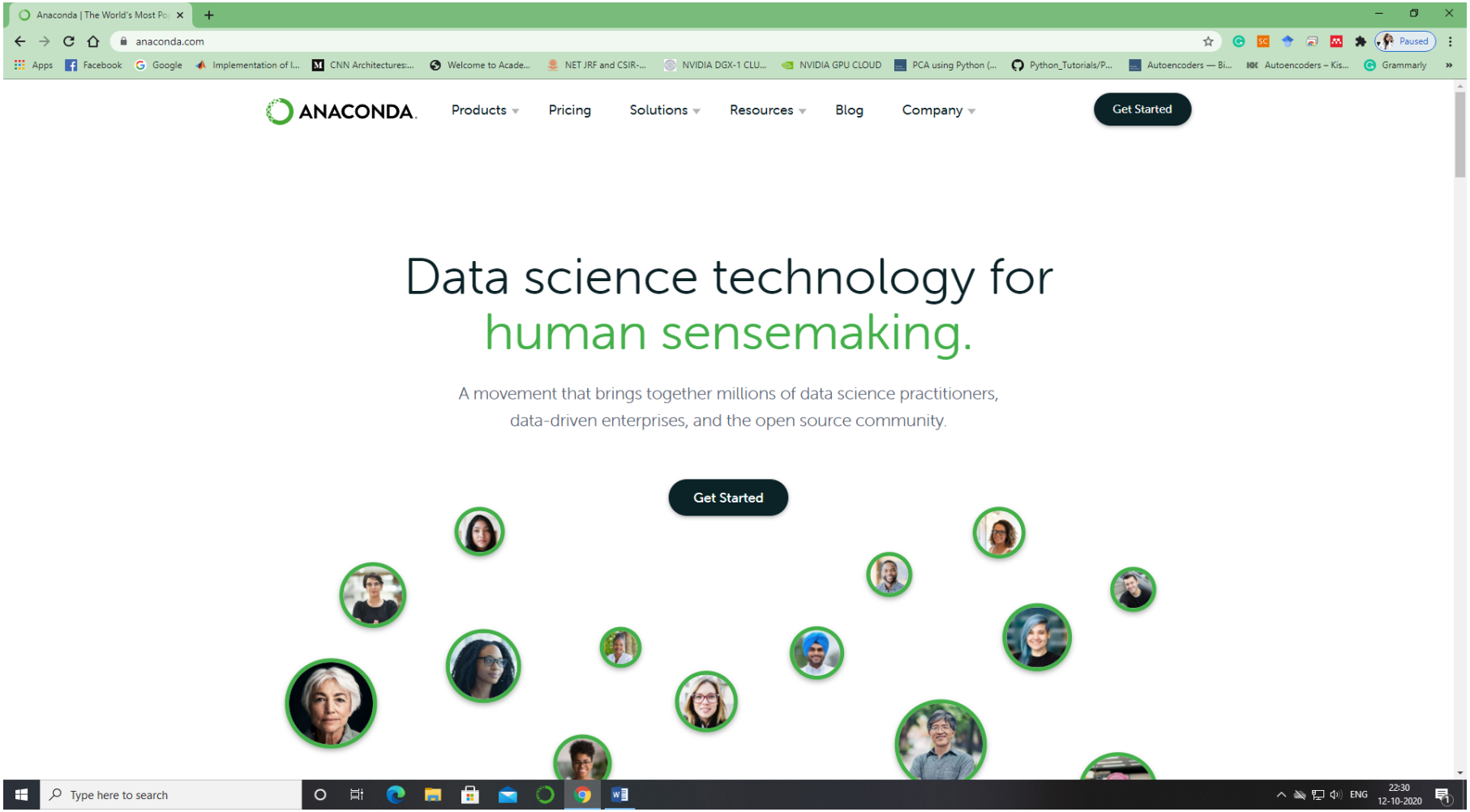
Installing Python through Anaconda

Installation Process

- Go to https://www.anaconda.com/
- Download the latest version
- Install Anaconda
- Install necessary libraries
  - Install NumPy
  - Install SciPy
  - Install Pandas
  - Install Matplotlib
  - Install scikit-learn
  - Install Tensorflow

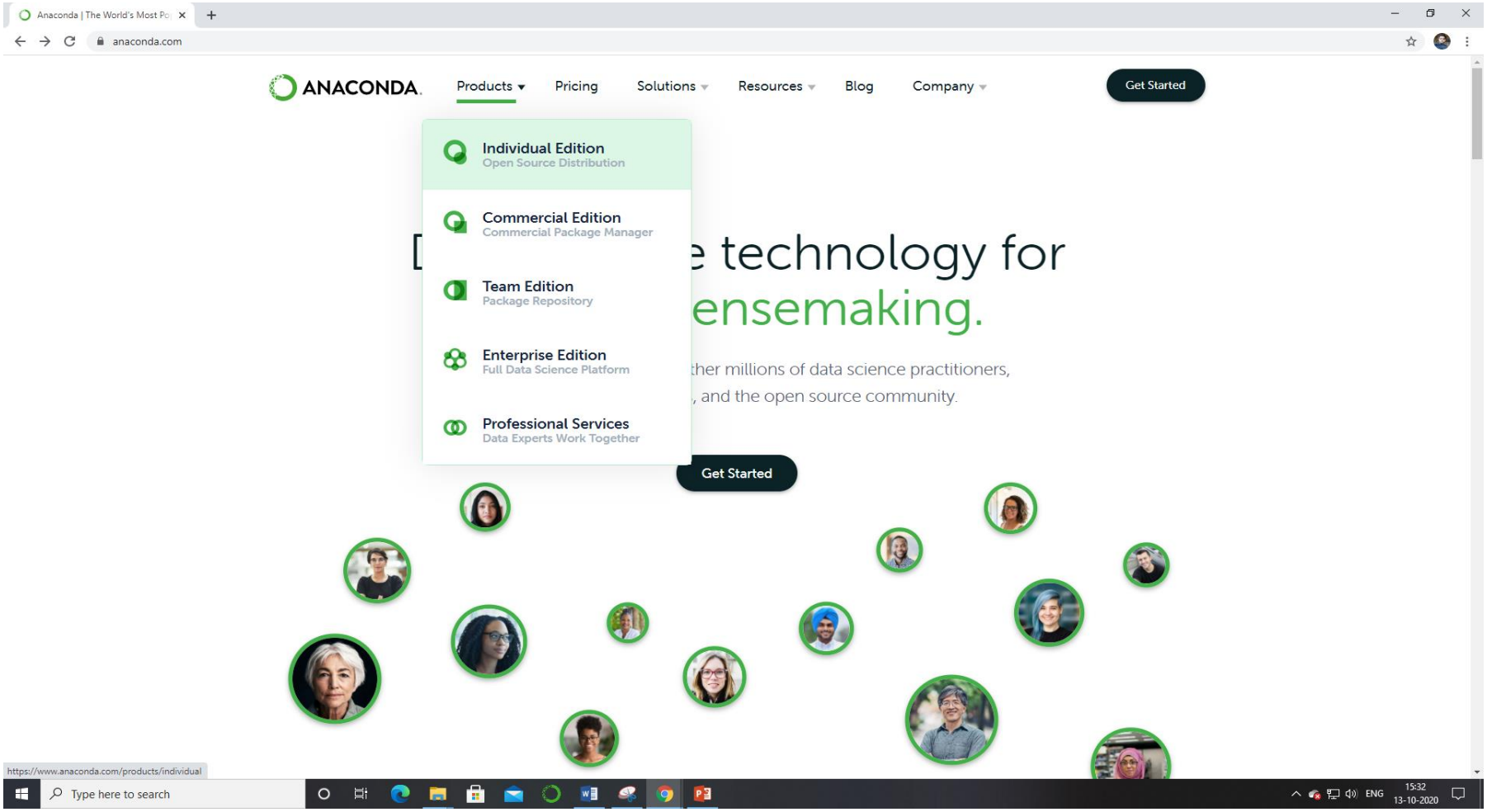
STEP 1

- Go to https://www.anaconda.com/



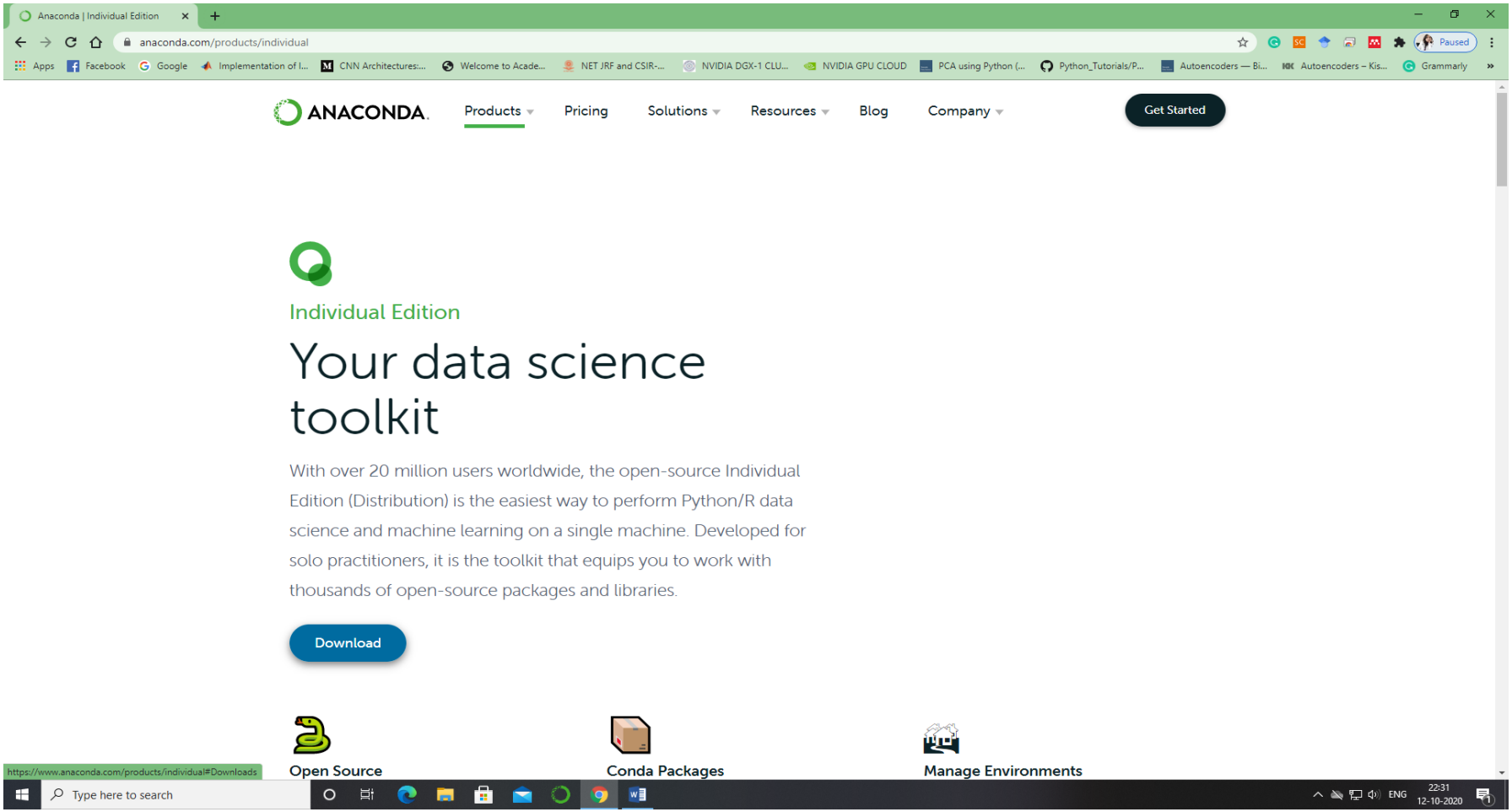
STEP 2

- Go to Products → Individual Edition



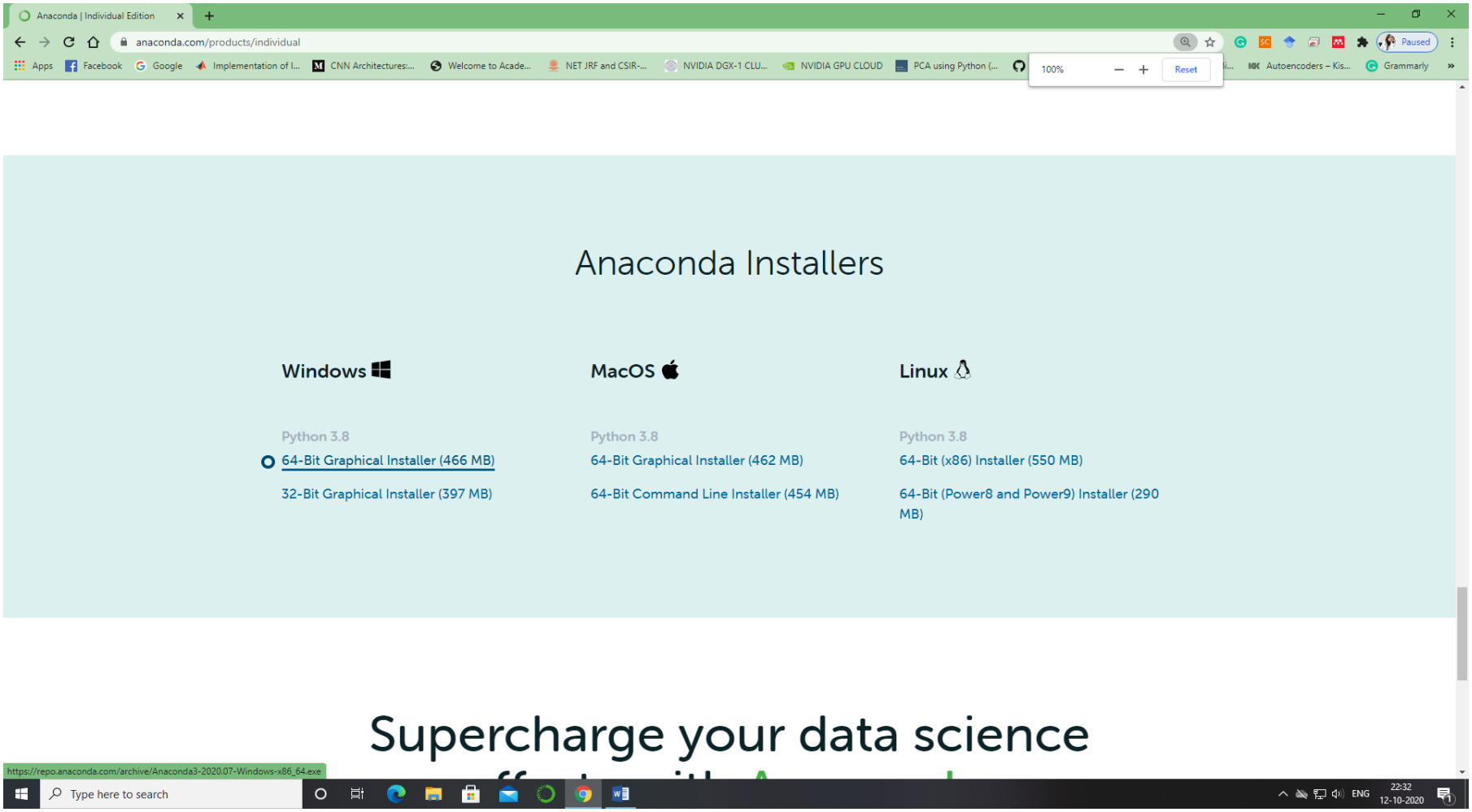
STEP 3

- Click on the Download button



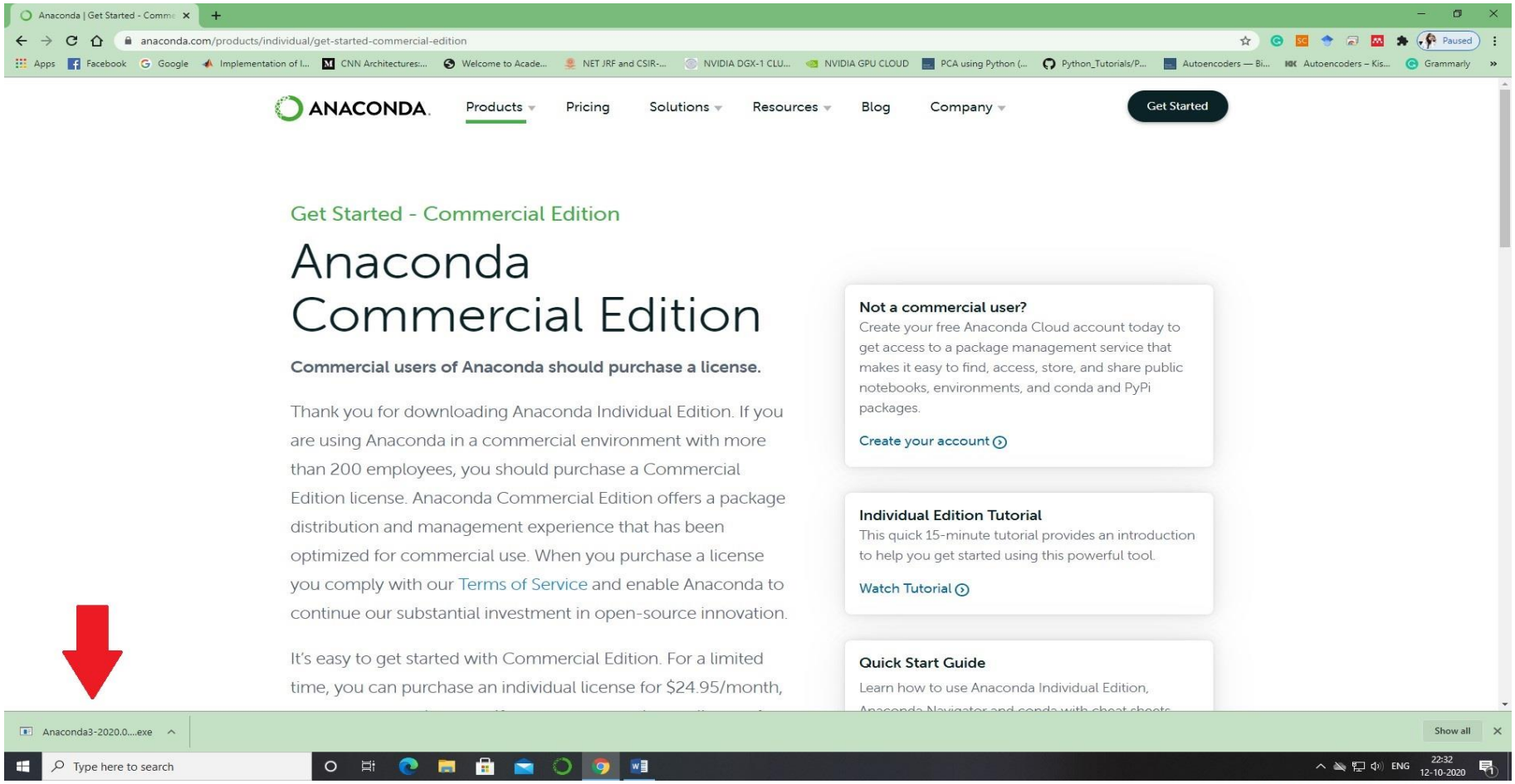
STEP 4

- Download the Installer according to the system requirements



STEP 5

- After downloading, click on the exe file in the bottom left corner

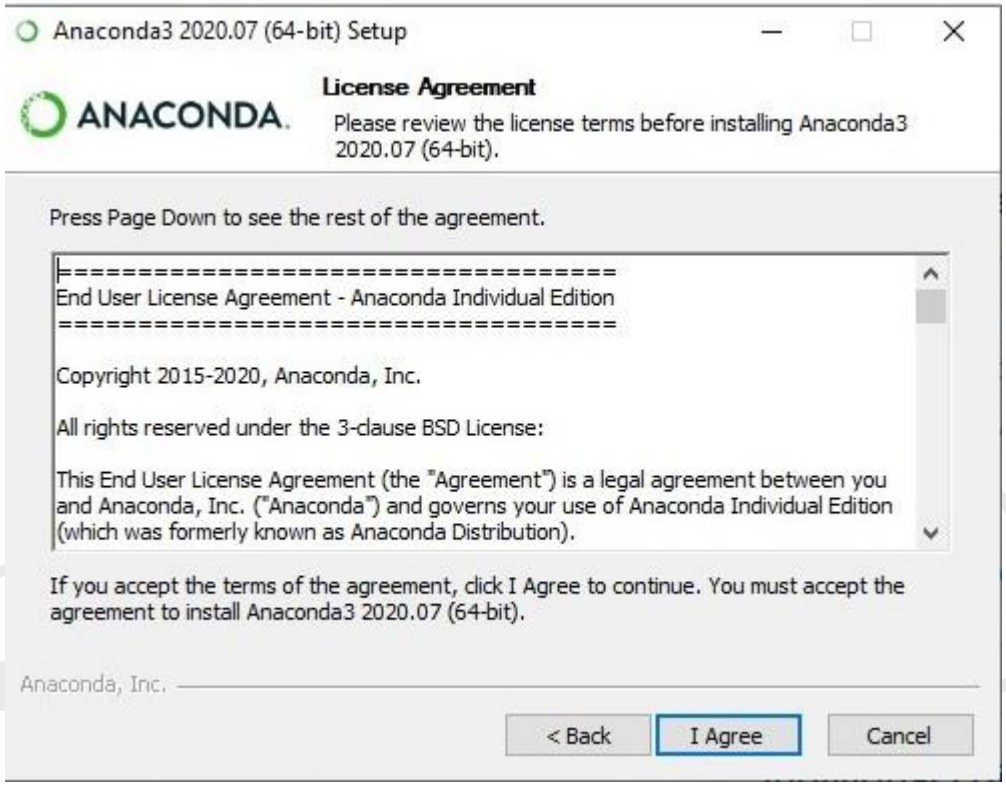


STEP 6

- Click “Next” on the popped up installation setup window

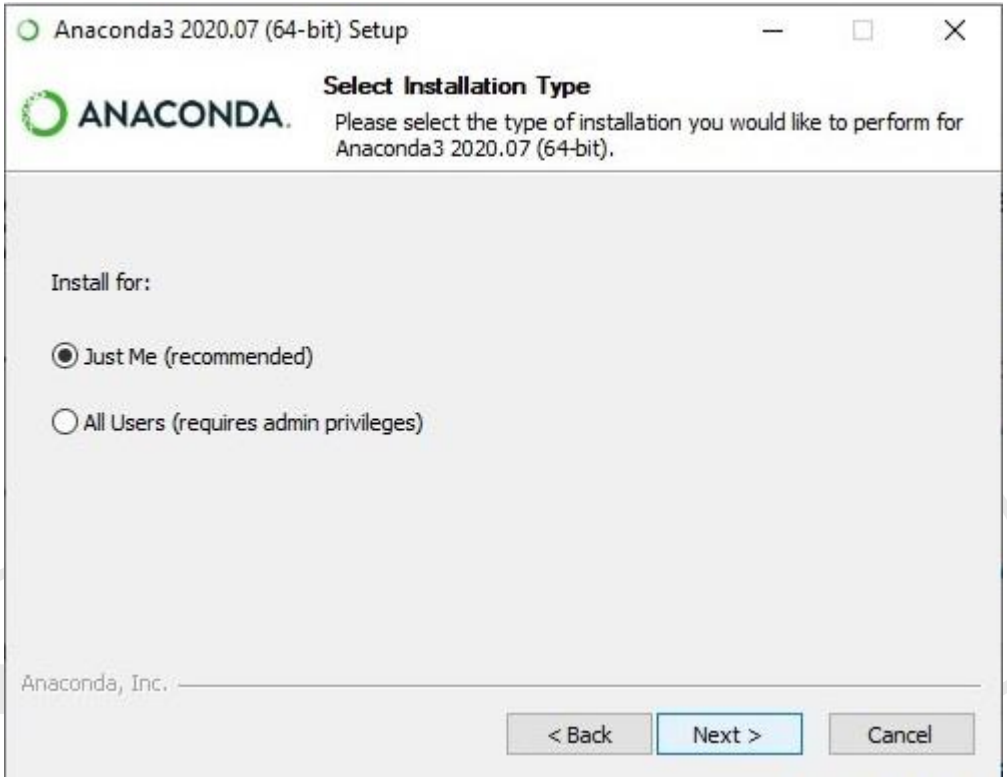
STEP 7

- Click “I Agree” on License Agreement



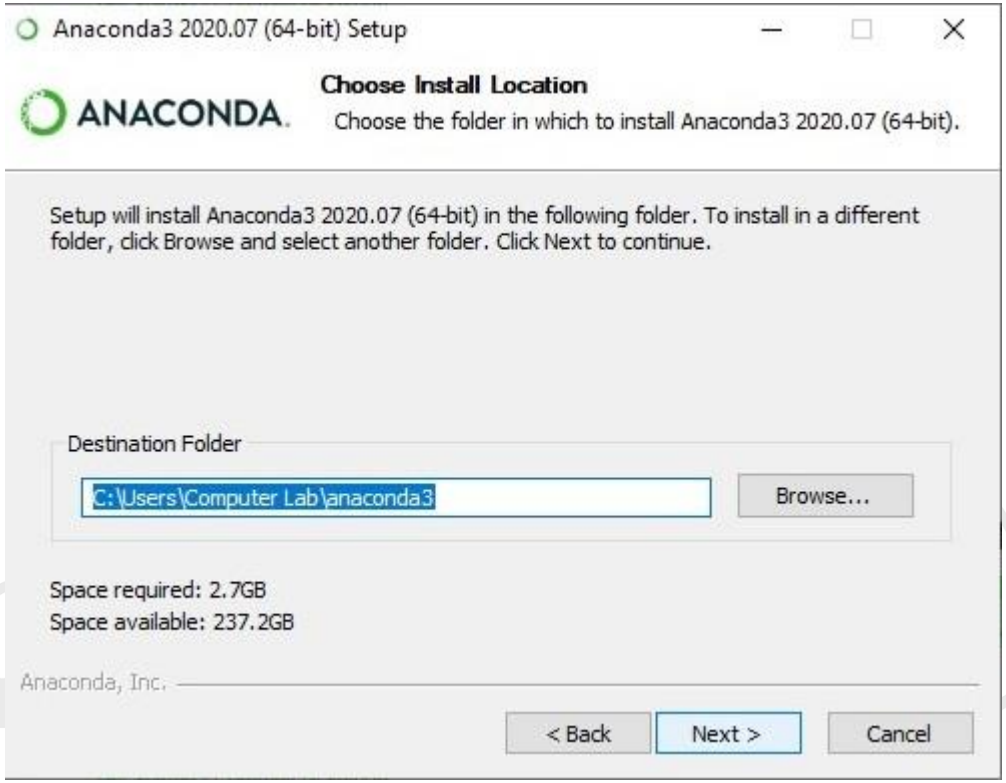
STEP 8

- Choose “Just Me (recommended)” and click “Next”



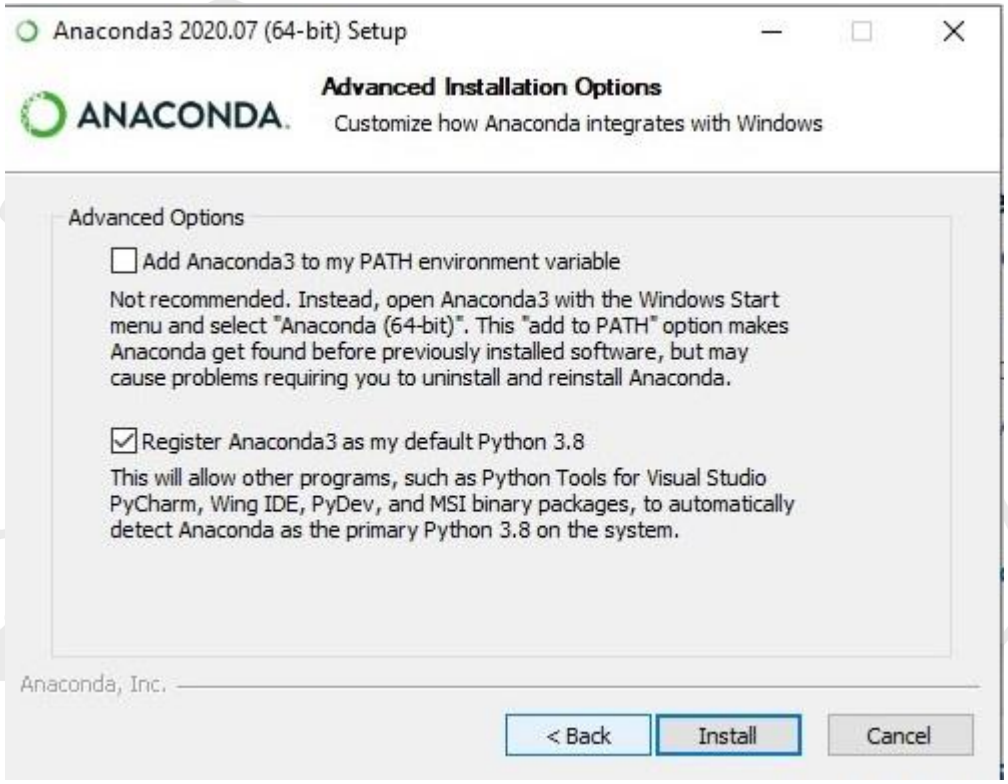
STEP 9

- Choose installation directory and click “Next”



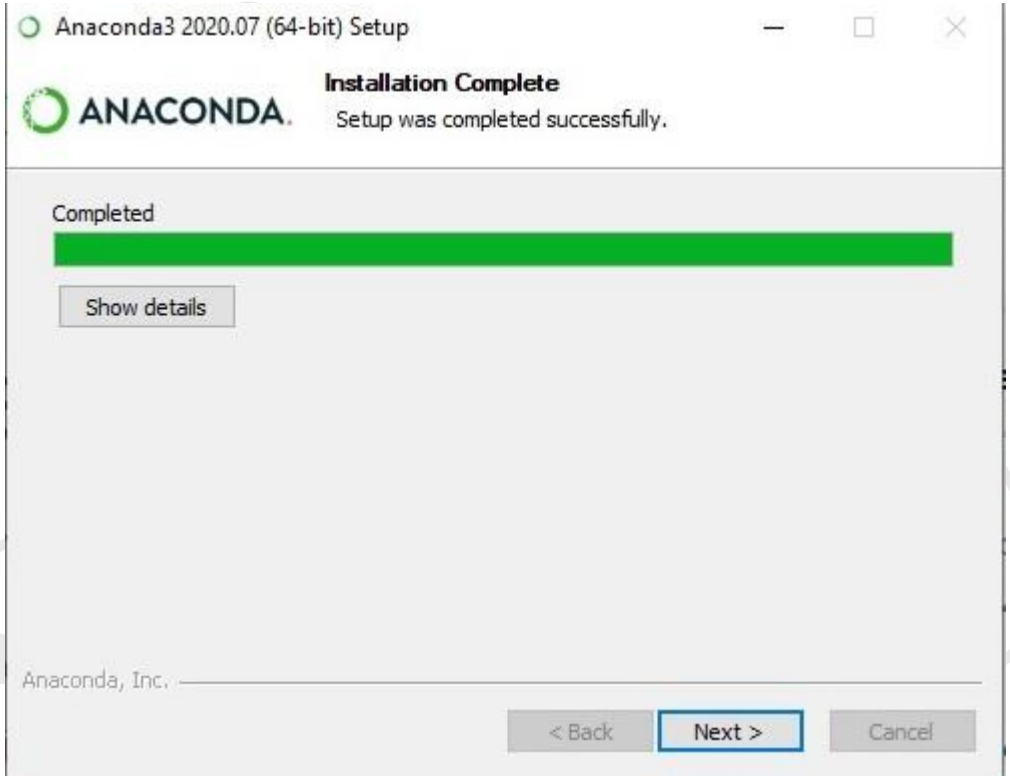
STEP 10

- Tick the checkbox for “Register Anaconda3 as my default Python 3.8 and click “Install”



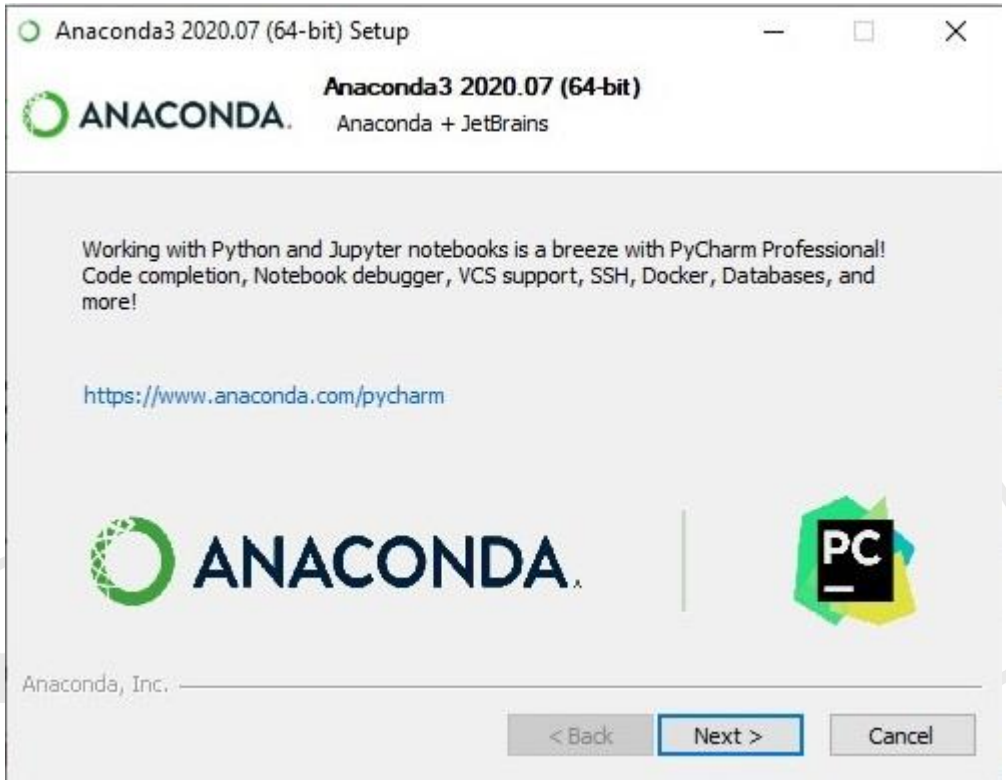
STEP 11

- After installation is completed, click “Next”



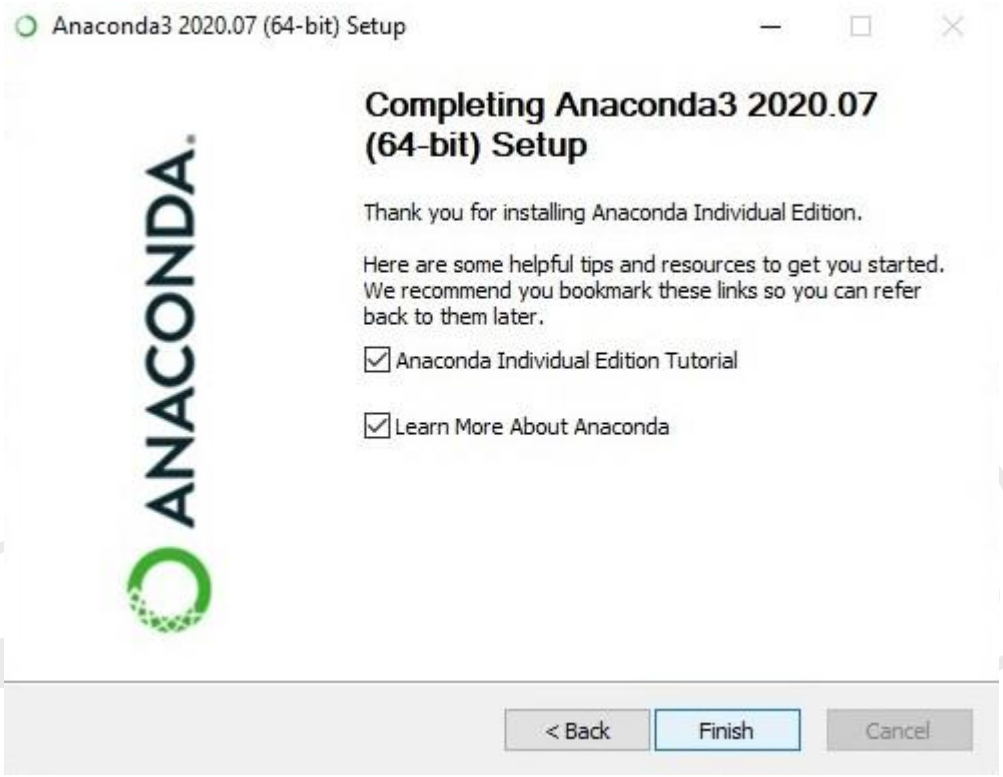
STEP 12

- Click “Next”



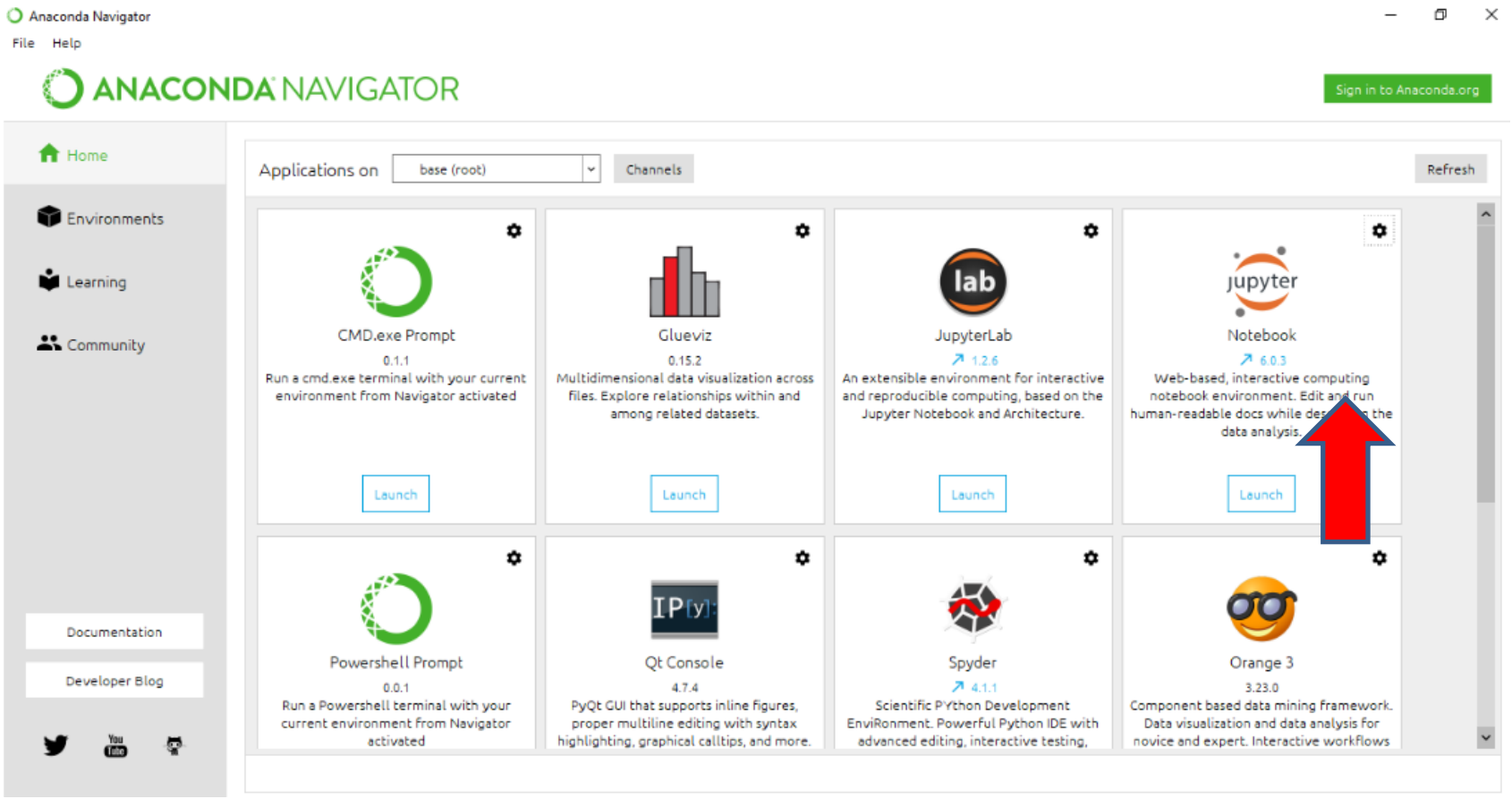
STEP 13

- Click “Finish”

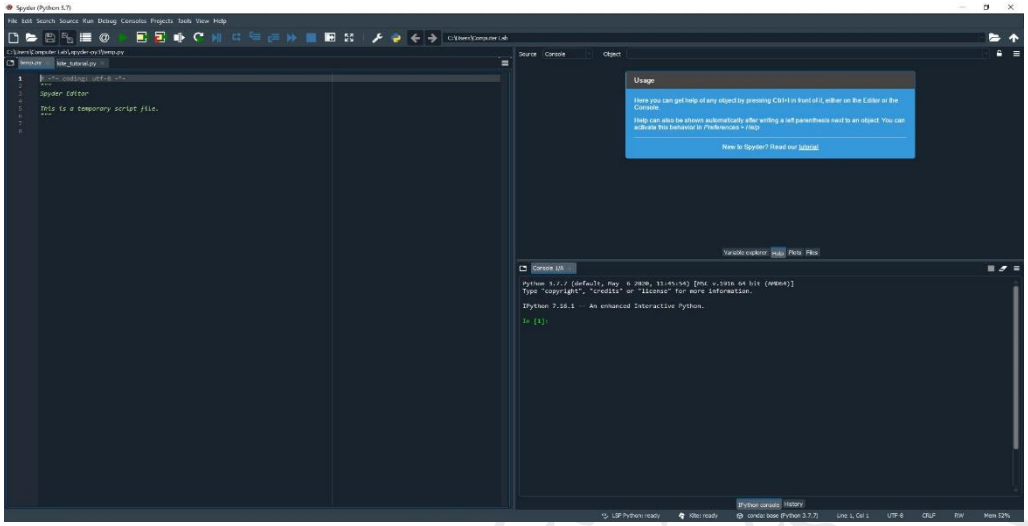


STEP 14

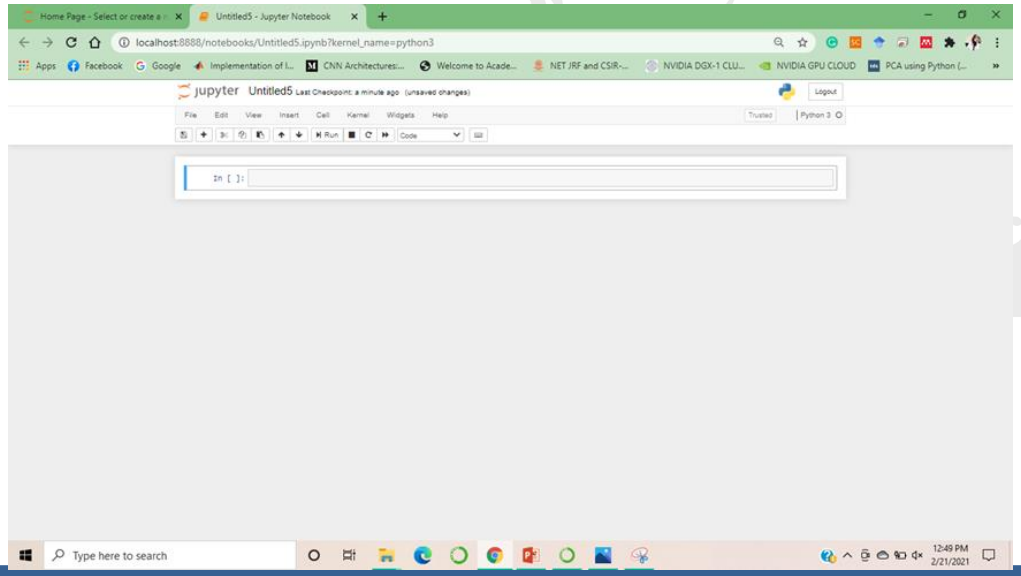
- Open the “Anaconda Navigator” and launch “Jupyter Notebook”



A view of IDEs Spyder and Jupyter



IDE - Spyder



IDE - Jupyter

Installing necessary libraries

- Open “Anaconda Prompt”

Installing necessary libraries

- Python libraries can be installed through either “conda” or “pip” package managers.

- Command to install through “conda”

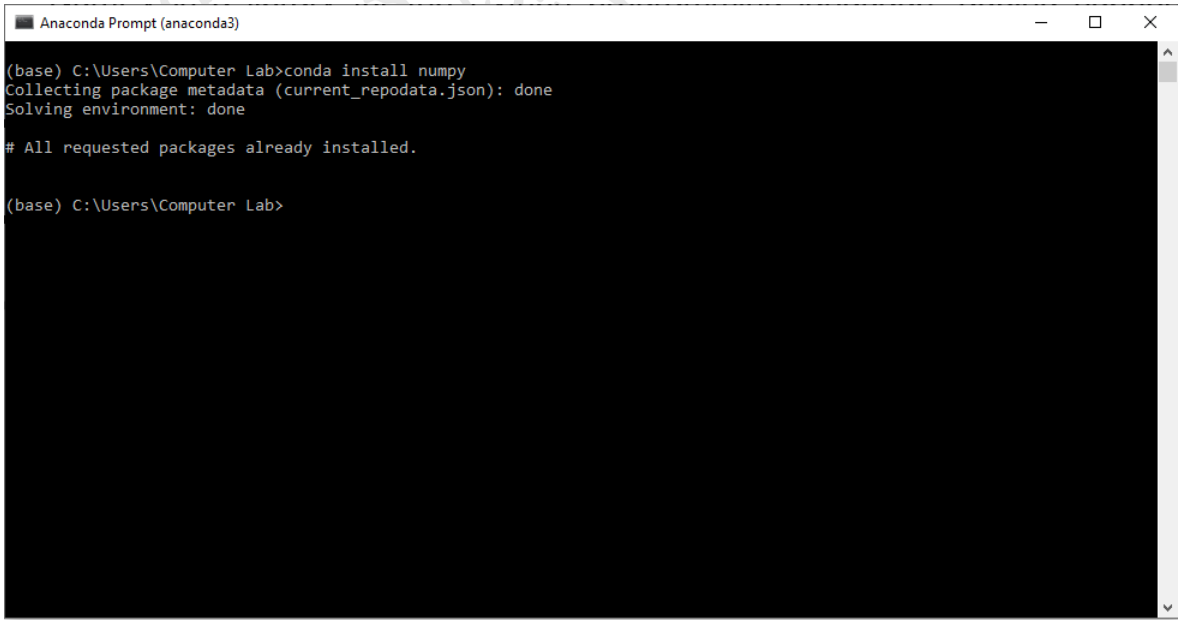
  – ***conda install <library name>***

- Command to install through “pip”

  – ***pip install <library name>***

Installing NumPy

- NumPy is a library for the Python programming language, **adding support for large, multi-dimensional arrays and matrices,** along with a large collection of high-level mathematical functions to operate on these arrays.
- **Write on Anaconda Prompt**
  - ***conda install numpy*** OR ***pip install numpy***
- This will install NumPy if it is not installed, otherwise it will show the following message



Installing SciPy

- SciPy is a free and open-source Python library used for **scientific computing** and technical computing. SciPy **contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering**.

- Write on Anaconda Prompt
  - ***conda install scipy*** OR ***pip install scipy***

Installing Pandas

- Pandas is a software library written for the Python programming language for **data manipulation and analysis which offers data structures and operations for manipulating numerical tables and time series**.

- Write on Anaconda Prompt
  - ***conda install pandas*** OR ***pip install pandas***

Installing Matplotlib

- **Matplotlib** is a **visualization and plotting** library for the Python programming language.

- Write on Anaconda Prompt
  - ***conda install matplotlib*** OR ***pip install matplotlib***

Installing scikit-learn

- **Scikit-learn** (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various **classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN**, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- Write on Anaconda Prompt
  - ***conda install scikit-learn*** OR ***pip install scikit-learn***

Installing Tensorflow

- TensorFlow is a Python library for fast and high performance numerical computing created and released by Google. It is a foundation library that can be used to create **Machine Learning and Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow**.

- Write on Anaconda Prompt
  - ***conda install tensorflow*** OR ***pip install tensorflow***

Keywords and Identifiers in Python

Keywords

- These are the reserved words in Python
- Keywords cannot be used for defining a variable, function, class etc.
- Keywords are case sensitive

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

All keywords except **True, False and None** are in lowercase

Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

- Identifiers are **user defined** and can be in lowercase, uppercase or a combination of lowercase and uppercase.
- An identifier cannot start with a digit (but can end up with a digit).
  - **1var – invalid**
  - **var1 – valid**
- Keywords cannot be used as identifiers.
- Except underscore (_), all special symbols like **!**, **@**, **#**, **$**, **%** etc. cannot be used in the identifier.
  - _var and var_ both are valid identifiers
  - @var or var@ etc. are invalid statements
- Identifiers can be of any length
- **It_is_my_var is a valid identifier**

Python is case sensitive therefore var is different from Var or VAR
Its good to name identifiers such that they properly represent an entity

Python Comments, Statements and Indentation

Python Comments

- Comments are very important while writing a program.
- Comments makes the understanding of the program easy.
- The hash (#) symbol to start writing a comment. It extends up to the newline character.

```
#comment to mark the beginning of the program
# can be used anywhere in the program
# It is a good programming practice to write comments at the
#beginning of the program
```

Good for single line comments

```
'''Triple quotation marks are another way of writing
comments'''
"""We can also use the double quation marks three times as
triple single quotation marks"""
```

Good for multiple line comments

Python Interpreter ignores comments.

Statements in Python

Instructions/ commands that a Python interpreter can execute are called statements.

For example,

a = 1 # is an assignment statement.

Explicit use of line continuation character (\) for multiple lines

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

line continuation is **implicit** parentheses ( ), brackets [ ], and braces { }.

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

Python Indentation

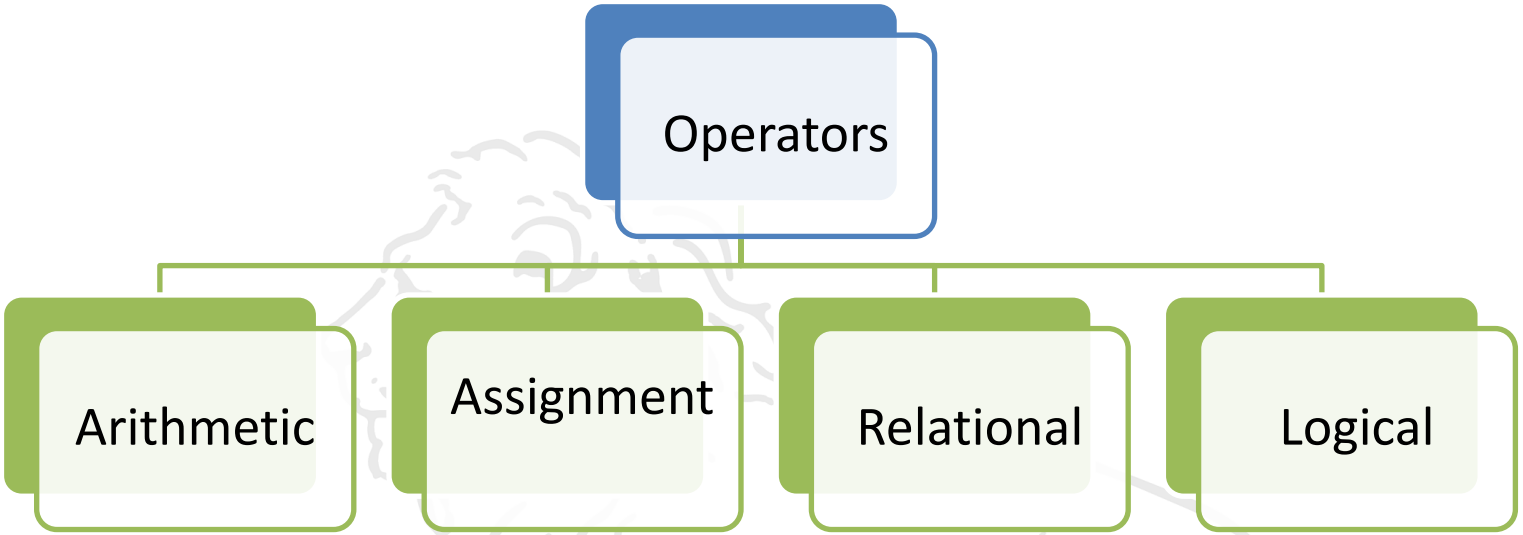
- Most of the programming languages like C, C++, and Java use braces *{ }* to define a block of code.
- Python

  uses

  Indentation
- A code block (body of a function, loop, etc.) starts with indentation and ends with the first unindented line.
- The amount of indentation is up to you, but it must be consistent throughout that block.
- Generally, **four whitespaces** are used for indentation and are preferred over tabs.

Python Operators

Types of operators



Basic Arithmetic Operators

Operator	Description	Example a=10, b=20
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//Floor Division	Division of operands. Output is quotient with digits after decimal point removed. **If one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)**	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Basic assignment operators

Assignment operators are used in Python to assign values to variables

Operator	Example	Equivalent to
=	x = 10	x = 10
+=	x += 8	x = x + 8
-=	x -= 15	x = x – 15
*=	x *= 7	x = x * 7
/=	x /= 22	x = x / 22
%=	x %= 9	x = x % 9
//=	x //= 6	x = x // 6
**=	x **= 2	x = x ** 2
&=	x &= 50	x = x & 50
|=	x |= 12	x = x | 12
^=	x ^= 4	x = x ^ 4

Comparison (relational) & Logical operators

Relational operators

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical operators

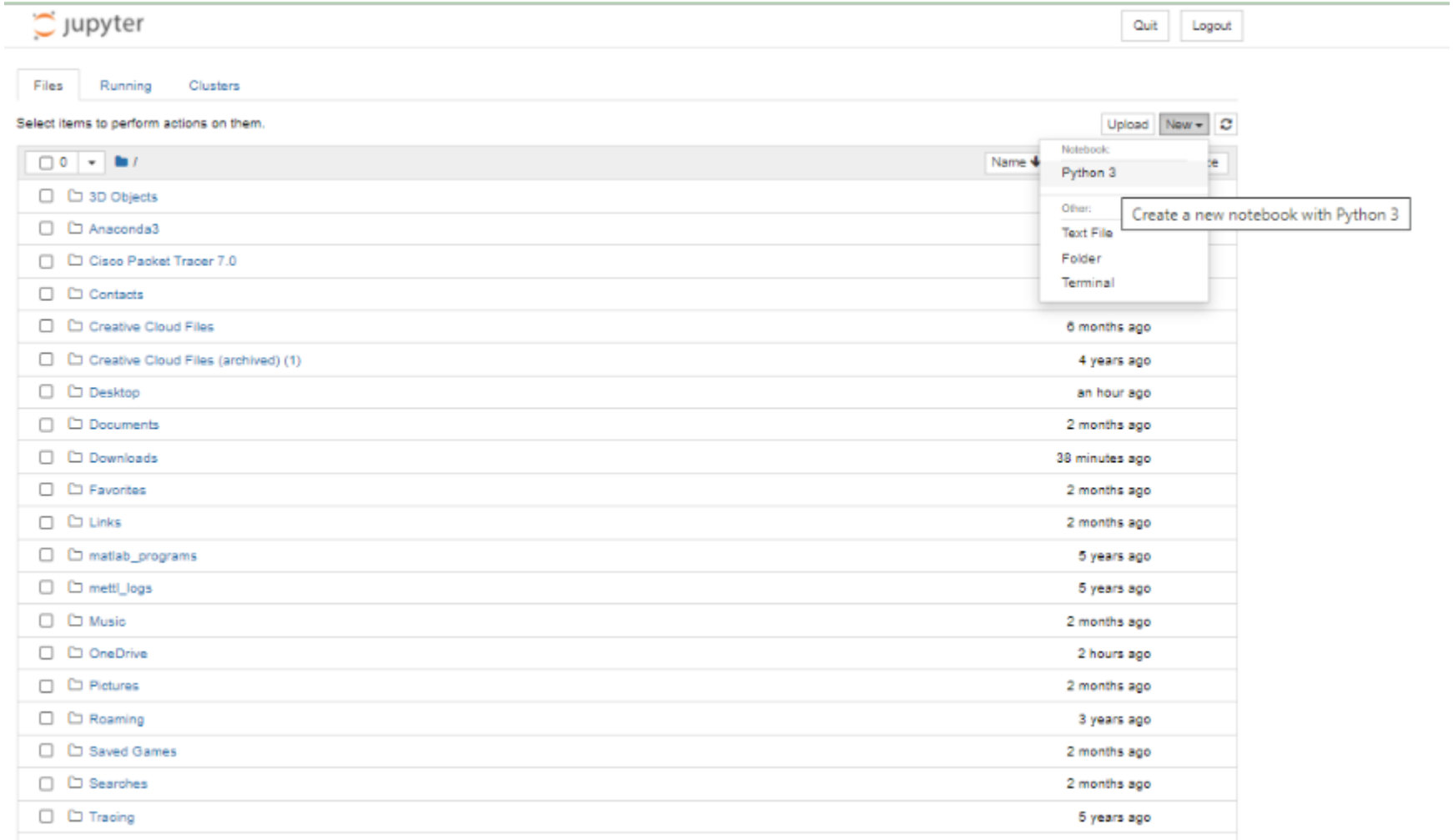
Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Order of precedence

Operator & Description
** Exponentiation (raise to the power)
~ + - Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % // Multiply, divide, modulo and floor division
+ - Addition and subtraction
>> << Right and left bitwise shift
& Bitwise 'AND'
^ | Bitwise exclusive `OR' and regular `OR'
<= < > >= Comparison operators
<> == != Equality operators
= %= /= //= -= += *= **= Assignment operators
is is not Identity operators
in not in Membership operators
not or and Logical operators

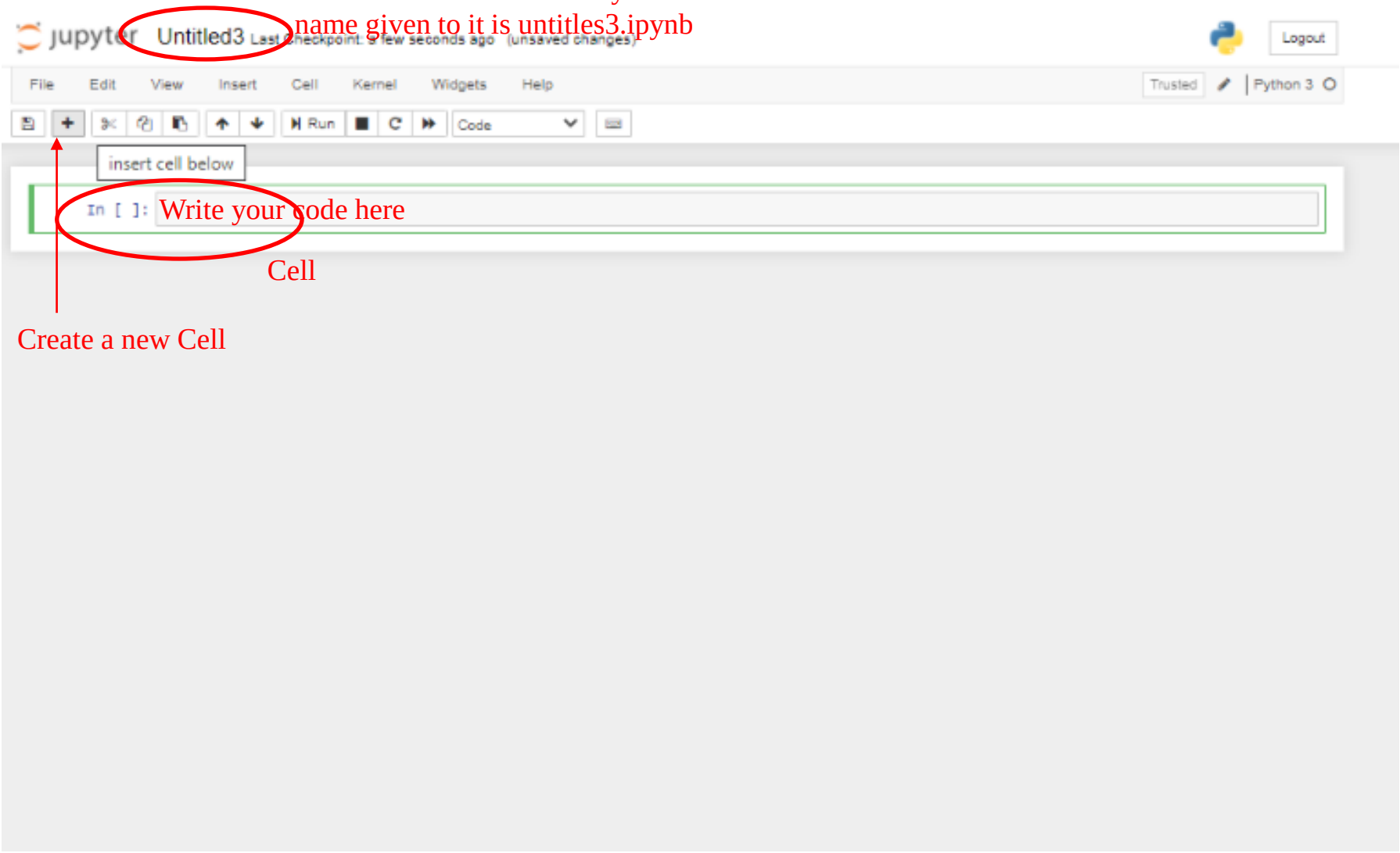
Order of precedence from top to bottom

A look at the Jupyter User Interface: Create a new notebook



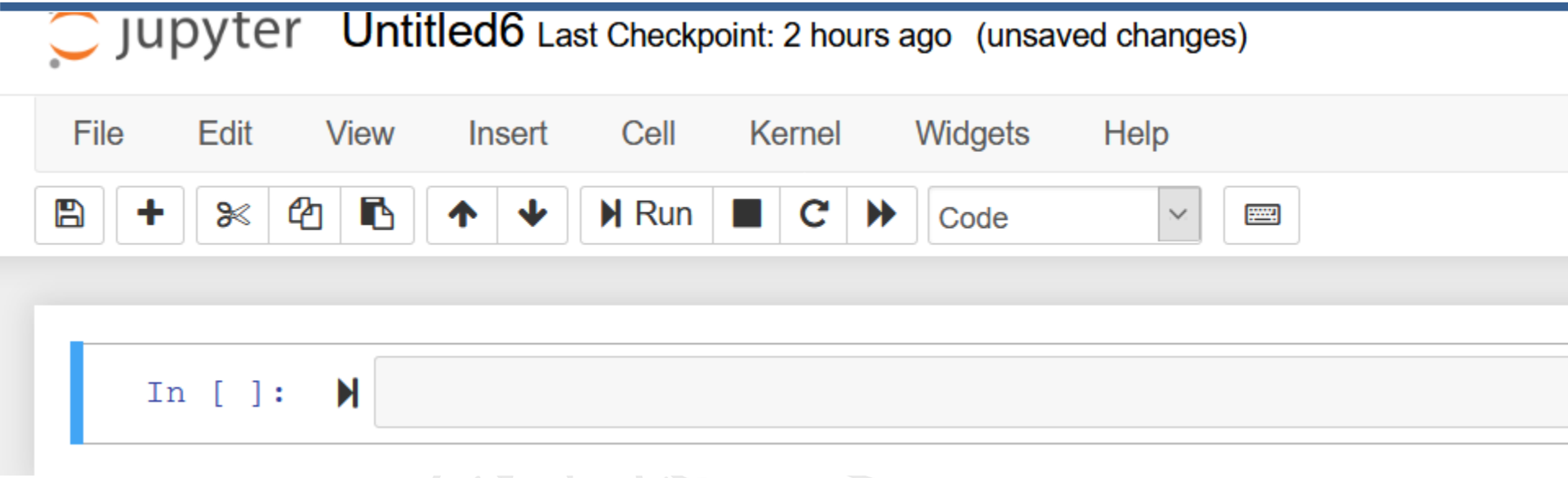
A look at the Jupyter User Interface

We have not saved the file yet so the default name given to it is untitles3.ipynb



Cell

Create a new Cell


jupyter
Untitled6 Last Checkpoint: 2 hours ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code
In [ ]:

- a = 30
- b = 10
- c = a+b
- d = a*b

Variables

```
print(a,b,c,d)
Output
30 10 40 300
print(a,b)
Output
30 10
print(c,d)
Output
40 300
```

Statically typed languages

Type of variable is known at compile time

Type of variables declared upfront

Eg – Java, C, C++

Dynamically typed languages

Type of variable known at runtime

Variable type need not be declared

Eg. **Python**, PHP

Your first Python Program: Doing some simple calculations

jupyter Basic_arithmetic_operations Last Checkpoint: a few seconds ago (autosaved)

Logout

We have saved the file as “Basic_arithmetic_operations”

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
In [1]: '''@author MP
        Arithmetic operation in Python'''

        x = 15
        y = 4
```

Statement written inside triple quotes “““” is non executable

```
In [5]: #Output x+y = 19
        print("x+y:", x+y)

        x+y: 19

In [6]: #Output x-y = 11
        print("x-y:", x-y)

        x-y: 11

In [7]: #Output x*y = 60
        print("x*y:", x*y)

        x*y: 60

In [8]: #Output x/y = 3.75
        print("x/y:", x/y)

        x/y: 3.75

In [9]: #Output x//y = 3
        print("x//y:", x//y)

        x//y: 3

In [10]: #Output x**y = 50625
         print("x**y:", x**y)

         x**y: 50625
```

Basic arithmetic operations

Relational Operators

jupyter Relational_operators Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:
```
'''Relational Operators
@author MP'''

x = 15
y = 10
```

In [3]:
```
#Output: x > y is True
print('x > y is', x > y)
```

```
x > y is True
```

In [4]:
```
#Output: x < y is False
print('x < y is', x < y)
```

```
x < y is False
```

In [5]:
```
#Output: x == y is False
print('x == y is', x == y)
```

```
x == y is False
```

In [6]:
```
#Output: x != y is True
print('x != y is', x != y)
```

```
x != y is True
```

In [7]:
```
#Output: x >= y is True
print('x >= y is', x >= y)
```

```
x >= y is True
```

In [8]:
```
#Output: x <= y is False
print('x <= y is', x <= y)
```

```
x <= y is False
```

Logical Operators

jupyter Logical_operators (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
'''Logical Operators
@author: MP'''

x = True
y = False
```

In [2]:
```
#Output: x and y is False
print('x and y is', x and y)
```

```
x and y is False
```

In [3]:
```
#Output: x or y is True
print('x or y is', x or y)
```

```
x or y is True
```

In [4]:
```
#Output: not x False
print('not x is', not x)
```

```
not x is False
```

Input/ Output and Formatting in Python

Reading Input From the Keyboard

Use of keyword **input()**

For example

lang = input("what language are you using")

```
In [8]:  lang=input("what language are you using?")

         what language are you using?Python
```

Shift+enter

```
In [9]:  lang

Out[9]:  'Python'
```

Point to remember:

input() always returns a string.

For numeric type, we need to do the type conversion like int(), float(), or complex() built-in functions:

```
In [12]:  n = input('Enter a number: ')
          y = n + 100
          y
```

```
Enter a number: 50

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-7668fc9822da> in <module>
      1 n = input('Enter a number: ')
----> 2 y = n + 100
      3 y

TypeError: can only concatenate str (not "int") to str
```

Type casting

```
In [14]:  n = int(input('Enter a number: '))
          y= n + 100
          y
```

Output

```
Enter a number: 50

Out[14]: 150
```

Displaying the Output

Use of keyword **print()**

For example

print("I am using python")

```
In [15]: print("i am using Python")

         i am using Python

In [16]: print(5+4)

         9

In [17]: print("sum of two numbers 5 and 4 is:", 5+4)

         sum of two numbers 5 and 4 is: 9
```

Double click on the output will hide it

Basic formatting in Python

In [19]:
```
for n in range(10):
    print(n, end=(' '))
```

```
0 1 2 3 4 5 6 7 8 9
```

In [20]:
```
for n in range(10):
    print(n, end=('\n'))
```

```
0
1
2
3
4
5
6
7
8
9
```

In [21]:

```
for n in range(10):
    print(n, end=('\t'))
```

```
0	1	2	3	4	5	6	7	8	9
```

Modulo operator (%) for formatting

```
In [24]:  print('%d %s cost Rs %.3f' % (6, 'bananas', 30))

          6 bananas cost Rs 30.000
```

```
In [25]:  print('%d %s cost Rs %.5f' % (6, 'bananas', 30))

          6 bananas cost Rs 30.00000
```

The '%' character denotes the conversion specifiers in the format string in the above example '%d', '%s', and '%.f'

In the output, each item from the tuple of values is converted to a string value and inserted into the format string in place of the corresponding conversion specifier.

```python
# Python Program for Strings

# String with single Quotes
S1 = 'Starting with Python'
print("String with Single Quote: ", S1)

# String with double Quotes
S2 = "Using double quotes in Python"
print("String with Double Quotes: ", S2)

# String with triple Quotes
S3 = '''Coming to triple quotes in Python"'''
print("String with Triple Quotes: ", S3)

# Feature of triple Quotes
S4 = '''Statement
            in
            three
            lines'''
print("Statement in multiple lines: ", S4)
```

Strings and basic operations on Strings

'Python'

"Python"

'''Python'''

Single quotes, double quotes or triple quotes…..
All are strings

In [2]:

```python
#Accessing the characters of the String
S1 = "Python"
print("Initial String is: ",S1)

# Accessing the First character
print("First character of String is: ", S1[0])

# Accissing the Last character
print("Last character of String is: ", S1[-1])
```

Concatenation of Strings

```
In [1]:  string1 = 'My'

In [2]:  string2='Python'

In [3]:  string1+string2

Out[3]: 'MyPython'
```

Using + operator to join the two strings

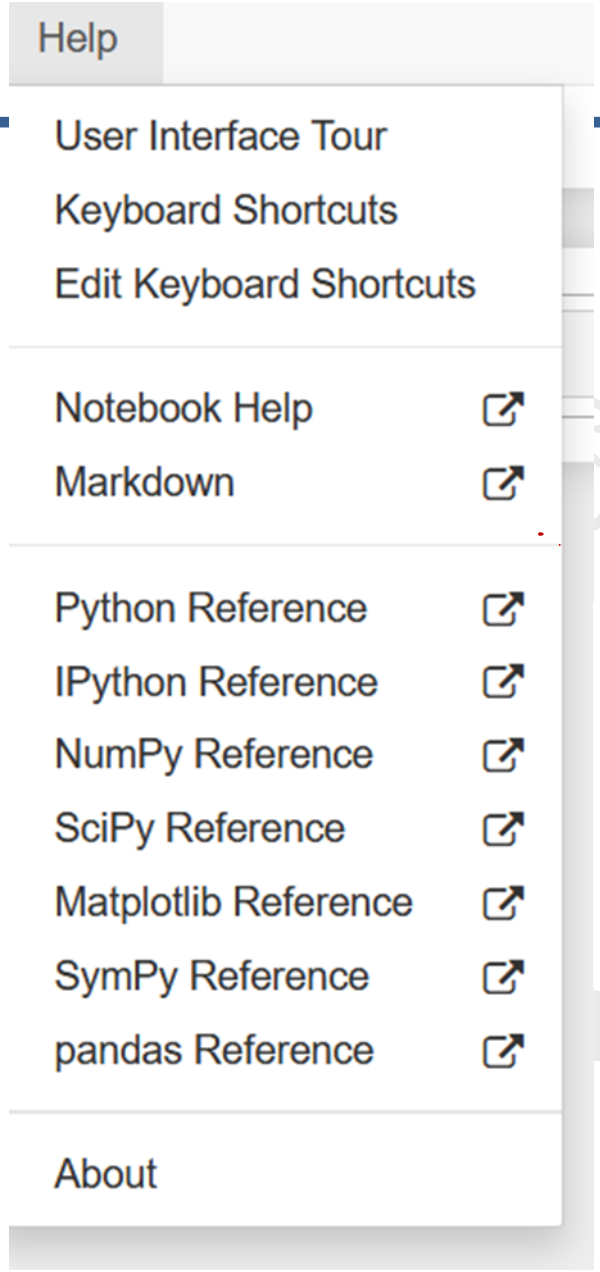
```
In [4]:  print(string1+string2)

         MyPython
```

```
In [5]:  string1

Out[5]:  'My'
```

What will be the output?

```
In [6]:  string1
         string2

Out[6]:  'Python'
```

Jupyter Notebooks Shortcuts



Useful Shortcuts :

Shift + Enter run the current cell, select below

Ctrl + Enter run selected cells

Alt + Enter run the current cell, insert below

Ctrl + S save and checkpoint

A : insert cell above

B : insert cell below

X : cut selected cells

C : copy selected cells

Know the version in which you are working

```
In [1]:  import sys

In [2]:  print(sys.version)

         3.7.7 (default, May  6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)]
```

Know the time of your system

```
In [43]: %system time

Out[43]: ['The current time is:  2:26:09.12', 'Enter the new time: ']
```

Know the variable type on which you are working

%whos displays the variable type plus some extra info: size, contents, etc.
%who_ls only displays the variables name

```
In [5]:  x, y = "hello", 5

In [6]:  %whos

Variable   Type      Data/Info
------------------------------
sys        module    <module 'sys' (built-in)>
x          str       hello
y          int       5
```

Interpreter vs. Compiler

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
Interpreters usually take less amount of time to analyze the source code. However, the **overall execution time is comparatively slower than compilers**.	Compilers usually take a large amount of time to analyze the source code. However, **the overall execution time is comparatively faster than interpreters**.
No Object Code is generated, hence are memory efficient.	Generates Object Code which further requires linking, hence requires more memory.
Programming languages like **JavaScript, Python, Ruby** use interpreters.	Programming languages like **C, C++, Java** use compilers.

https://www.programiz.com/article/difference-compiler-interpreter

Dynamically typed languages

A language is **dynamically-typed** if the type of a variable is checked during **run-time**. Common examples of dynamically-typed languages includes JavaScript, Objective-C, PHP, Python, Ruby, Lisp, and Tcl.

Now… unfold your hands

- **Learn the Basic Syntax**
- **Write Code by Own**
- **Keep Practicing**
- **Make Notes as Needed (Jupyter notebook is good for preparing notes)**
- **Discuss Concepts with Other**
- **Do small Projects**
- **Teach Others**
- **Explore Libraries and Frameworks**
- **Read latest blogs**

Practice problems – Day 1

#Precedence of operators

```
a = 2
b = 3
c = 4
d = 5
e = 0
f=0
e = (a + b) * c / d
print ("Value of (a + b) * c / d is ",  e)
e = ((a + b) * c) / d
print ("Value of ((a + b) * c) / d is ",  e)
e = (a + b) * (c / d)
print ("Value of (a + b) * (c / d) is ",  e)
e = a + (b * c) / d
print ("Value of a + (b * c) / d is ",  e)
f = a**b+c*d-a/b+c//d
print("using so many operators !!! What can be the answer ???")
print (f)
```

```
Value of (a + b) * c / d is  4.0
Value of ((a + b) * c) / d is  4.0
Value of (a + b) * (c / d) is  4.0
Value of a + (b * c) / d is  4.4
using so many operators !!! What can be
the answer ???
27.333333333333332
```

Refresher Questions

Que 1 Which one of the following is the correct extension of the Python Jupyter file?

a) .py
b) .python
c) .ipynb
d) None of these

Que 2 Which one of the following is correct way of declaring and initialising a variable, x with value 5?

a) int x
   x=5
a) int x=5
b) x=5
c) declare x=5x

Que 3 What is the maximum length of an identifier in python?

a) 32
b) 31
c) 63
d) None of the above

Que.7 What is the output of the following code:

```
print(9//2)
```

a) 4.5
b) 4.0
c) 4
d) Error

Que.8 What is the output of print(2 * 3 ** 3 * 4)

a) 216
b) 864

Que.9 What is the answer to this expression, 22 % 3 is?

a) 7
b) 1
c) 0
d) 5

Que.10 What is the output of print(2%6) ?

a) ValueError
b) 0.33
c) 2

Que.11 What is the output of print(2 ** 3 ** 2) ?

a) 64
b) 512

Que.12 What is the output of the following code?

```
x = 100
y = 50
print(x and y)
```

a) True
b) 100
c) False
d) 50

Que 4 Which of the following is a valid variable?

a) var@
b) 32var
c) Class
d) abc_a_c

Que 5 Which of these are keyword?

a) In
b) Is
c) Assert
d) All of the above

Que 6 Which is the correct operator for power(x^y)?

a) x^y
b) x**y
c) x^^y
d) None of the mentioned

Thanks…