Course: Artificial Intelligence

Assistant Lecturer : Yasmin Alsakar

Information Technology Department

Faculty of Computer & Information Sciences - Mansoura University.

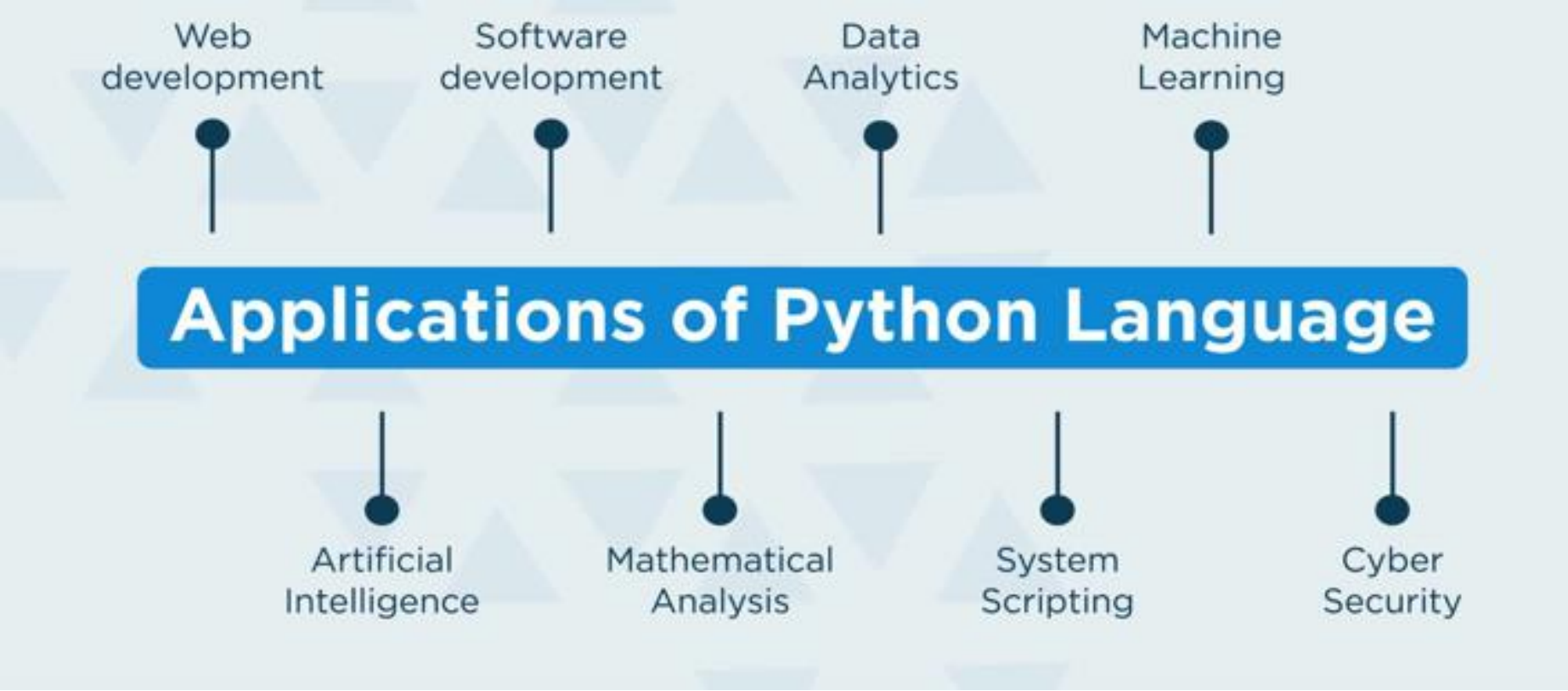
What is Python?

- **Python** is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

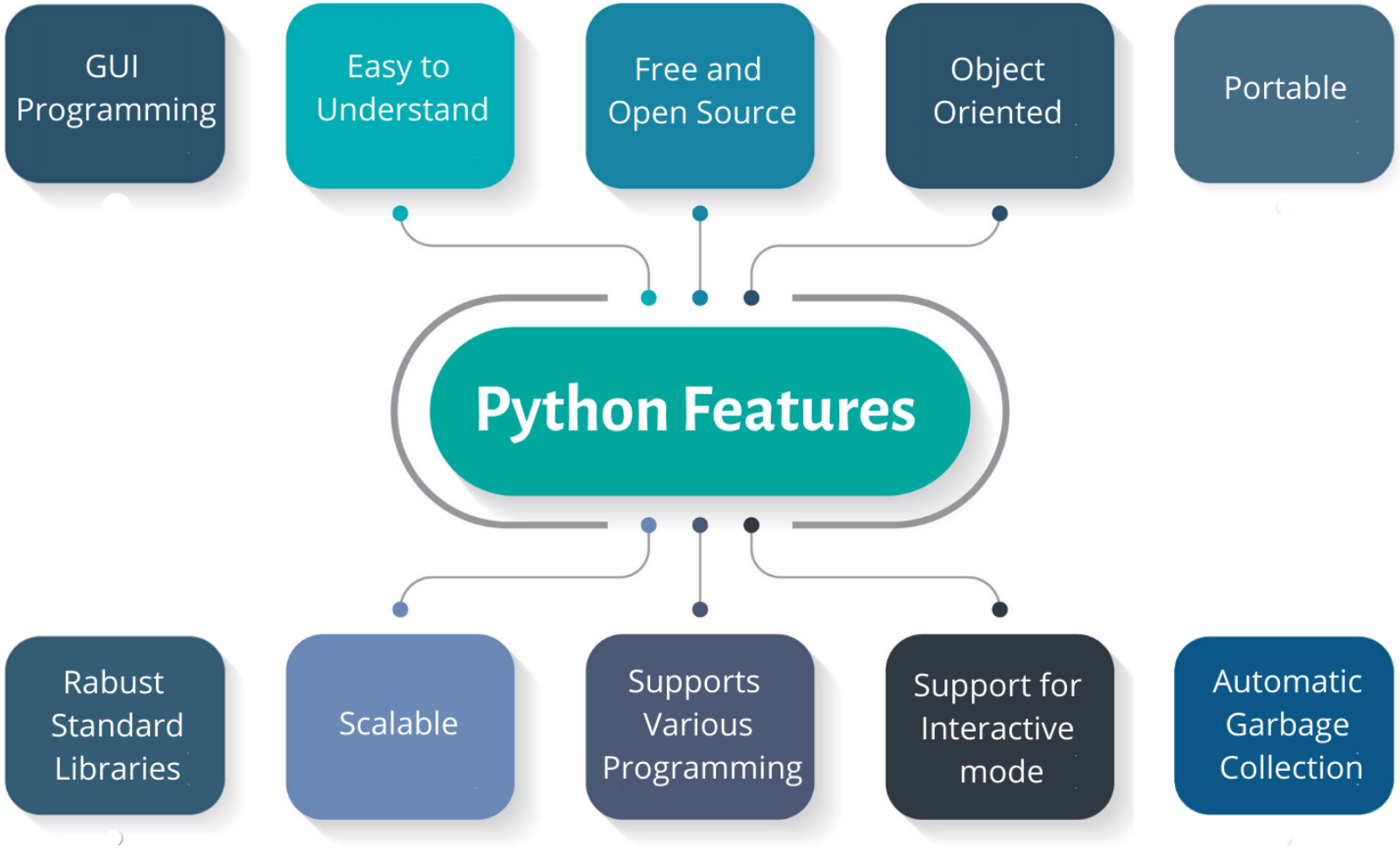
Why to Learn Python?

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

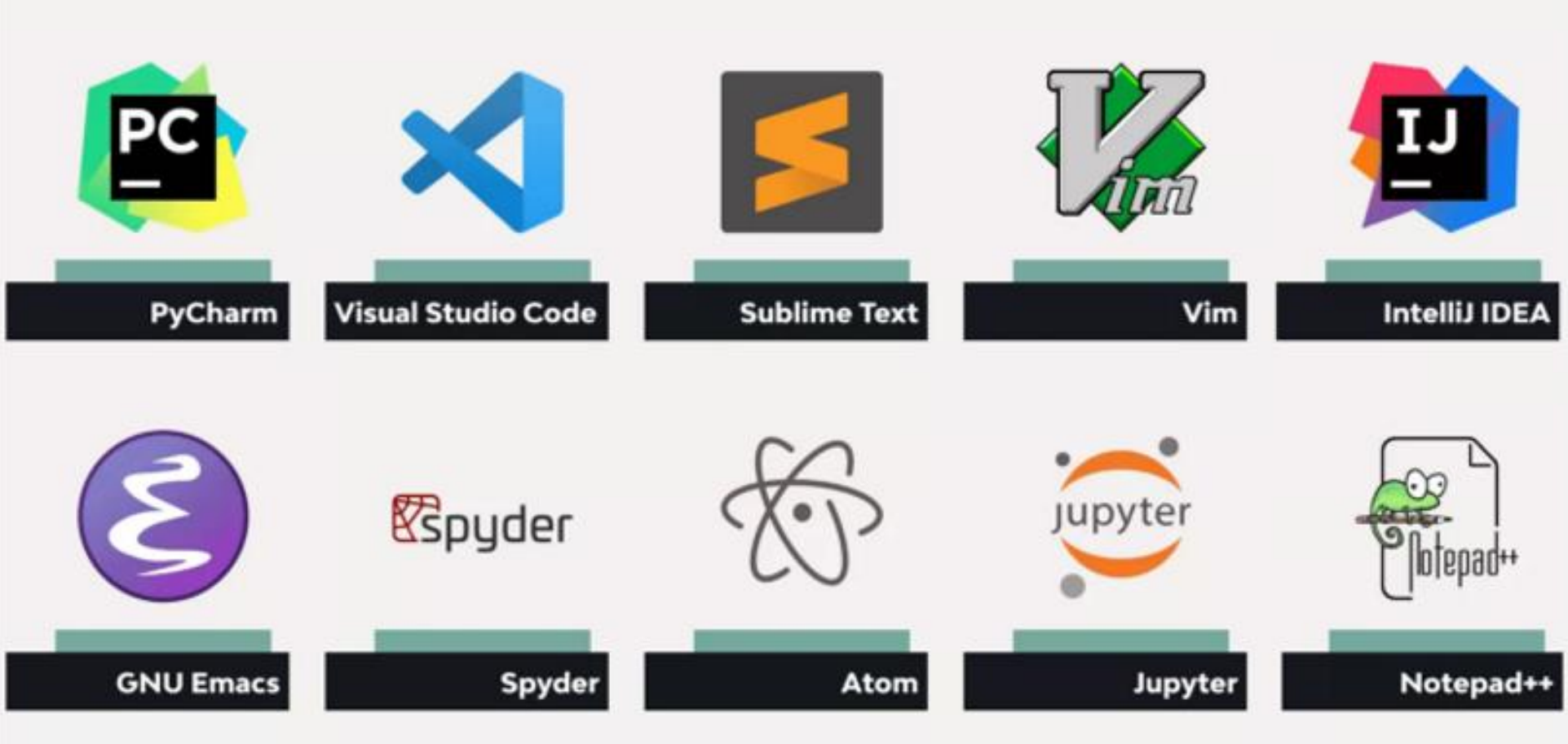
Python Applications



Python Features

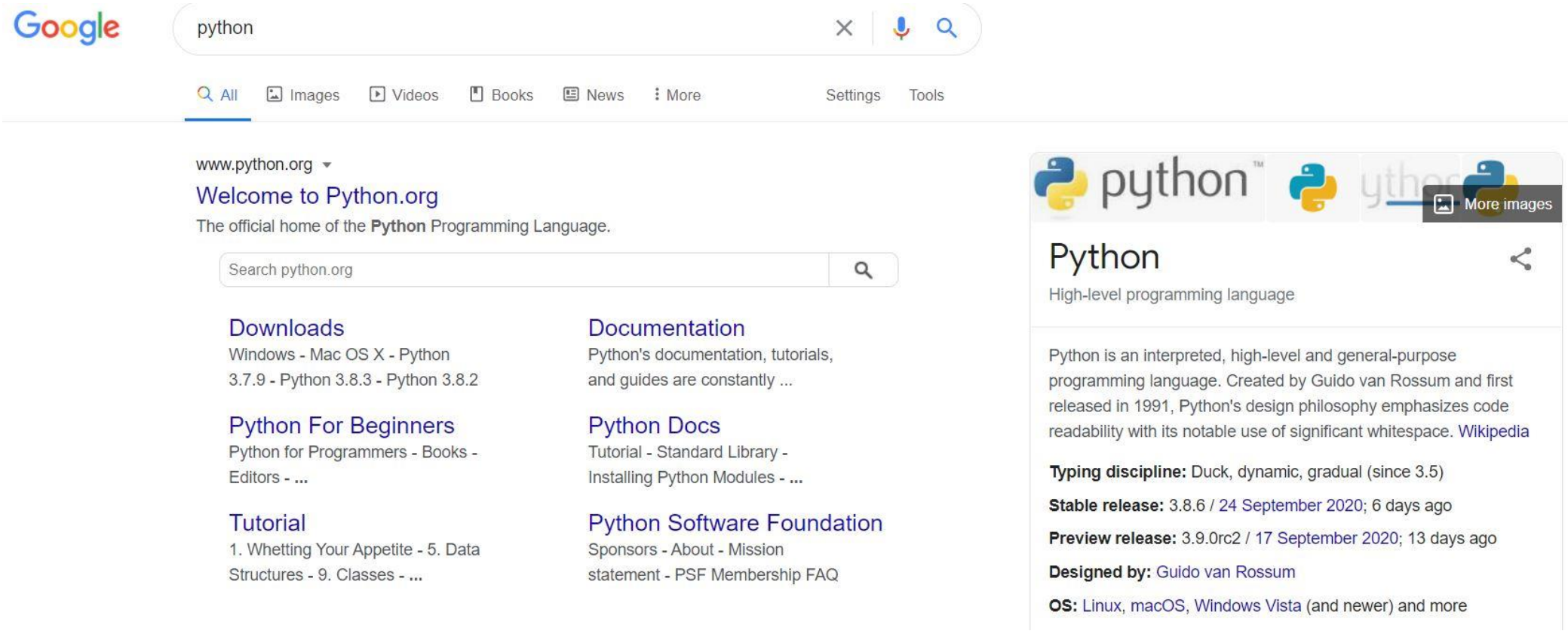


Python Editors

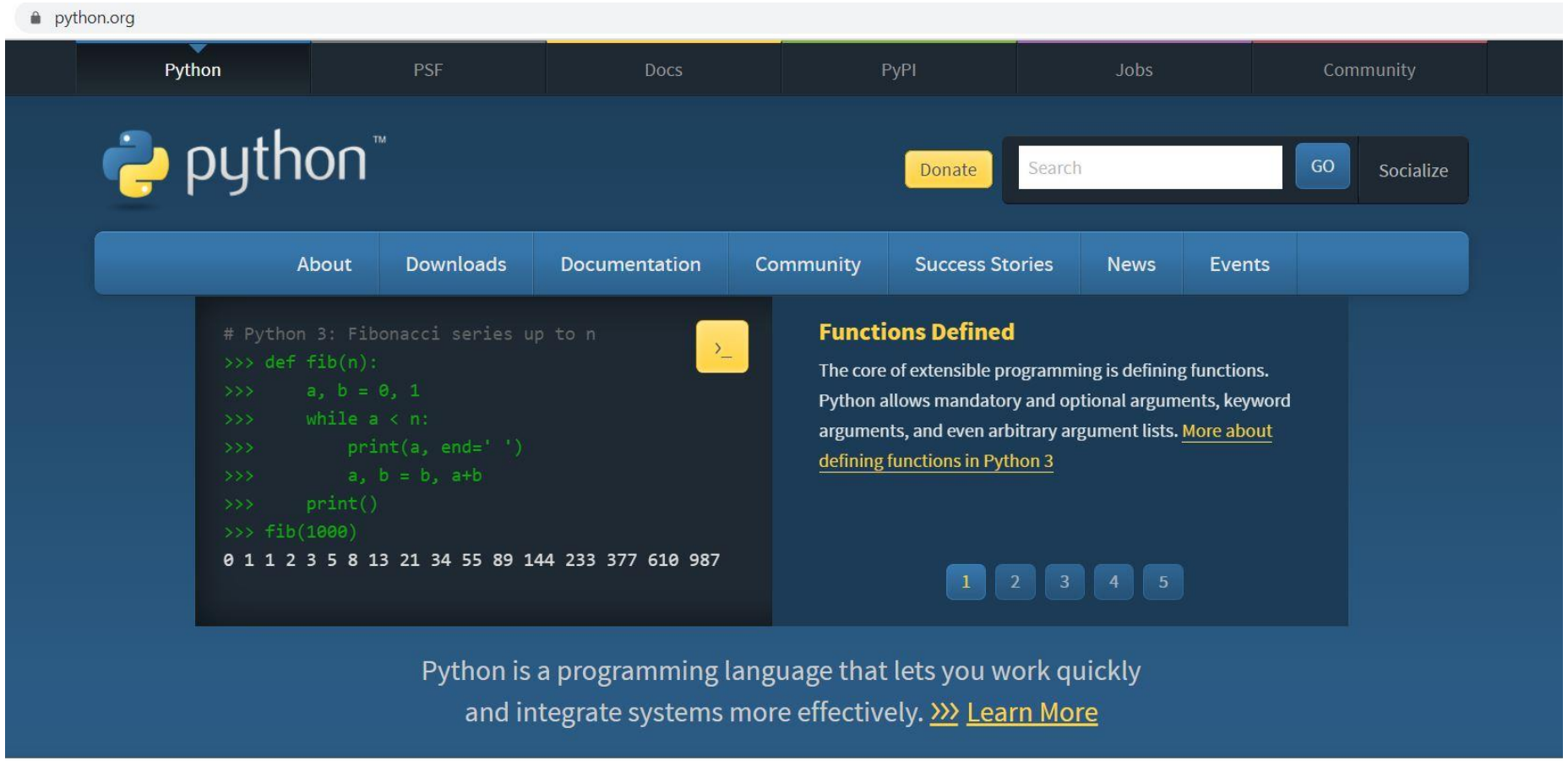


Python Installation

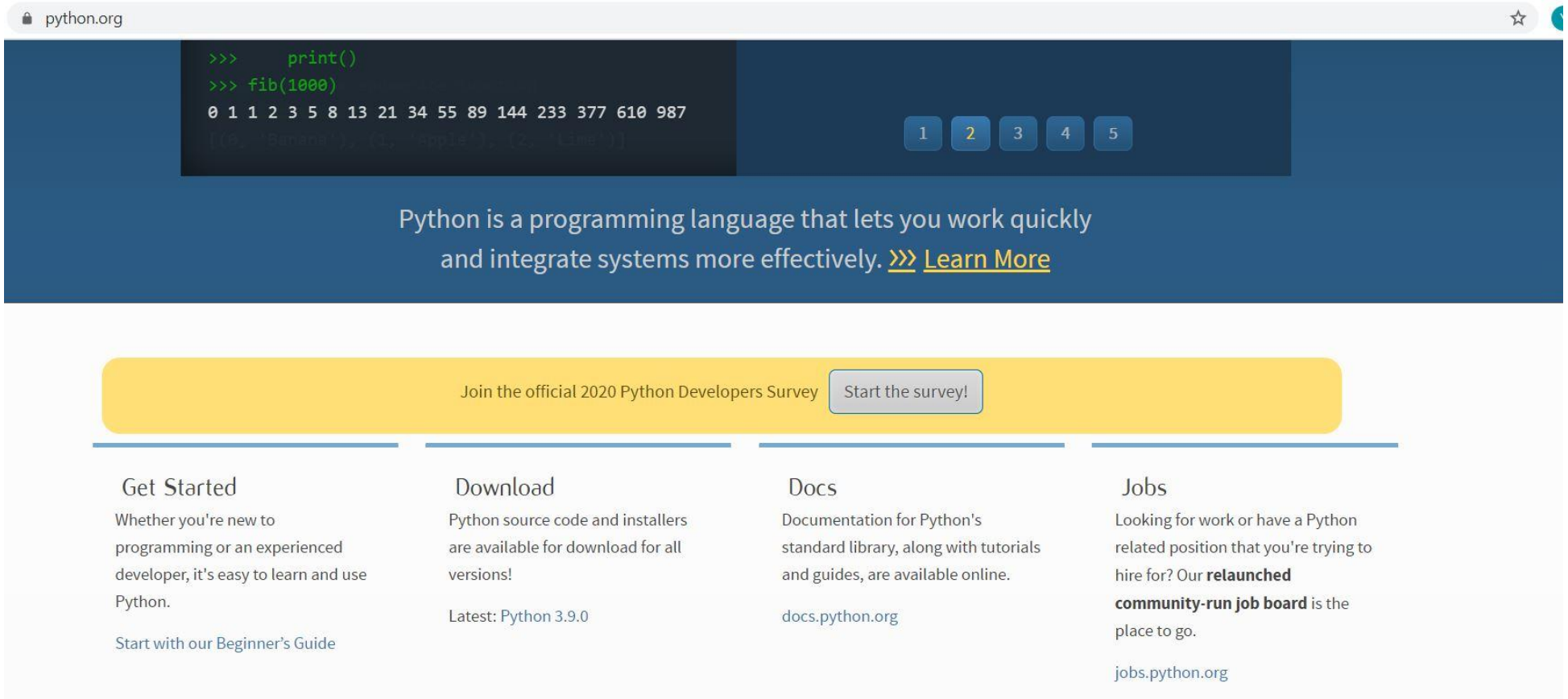
- Step1: login python website.



- Step2: scroll down in page.



- Step3: click on all downloads.



- Step4: choose all releases.

About
Applications
Quotes
Getting Started
Help
Python Brochure

Events
Python Events
User Group Events
Python Events Archive
User Group Events Archive
Submit an Event

Downloads
All releases
Source code
Windows
Mac OS X
Other Platforms
License
Alternative Implementations

Documentation
Docs
Audio/Visual Talks
Beginner's Guide
Developer's Guide
FAQ
Non-English Docs
PEP Index
Python Books
Python Essays

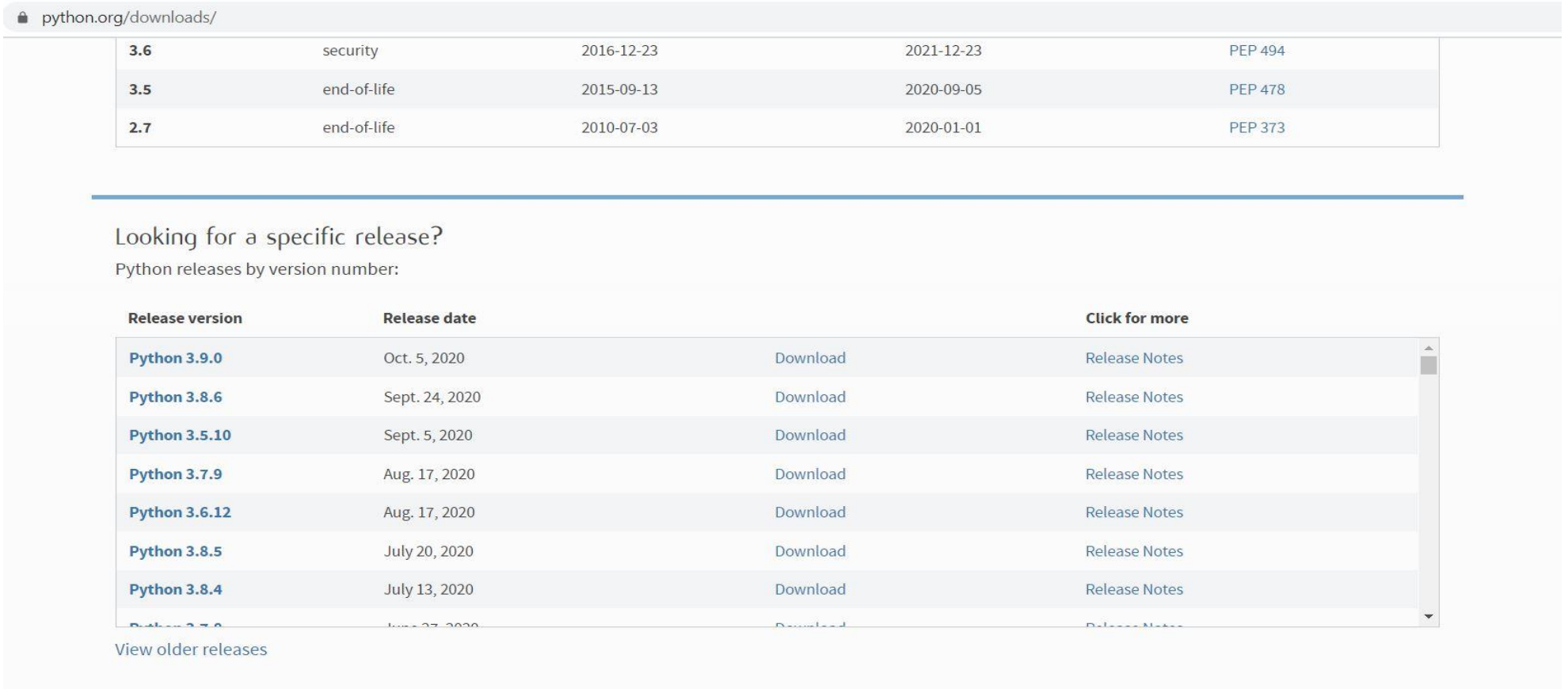
Community
Community Survey
Diversity
Mailing Lists
IRC
Forums
PSF Annual Impact Report
Python Conferences
Special Interest Groups
Python Logo
Python Wiki
Merchandise
Community Awards
Code of Conduct

Success Stories
Arts
Business
Education
Engineering
Government
Scientific
Software Development

News
Python News
PSF Newsletter
Community News
PSF News
PyCon News

Contributing
Developer's Guide
Issue Tracker
python-dev list
Core Mentorship
Report a Security Issue

- Step5: .choose python version (3.7.6)



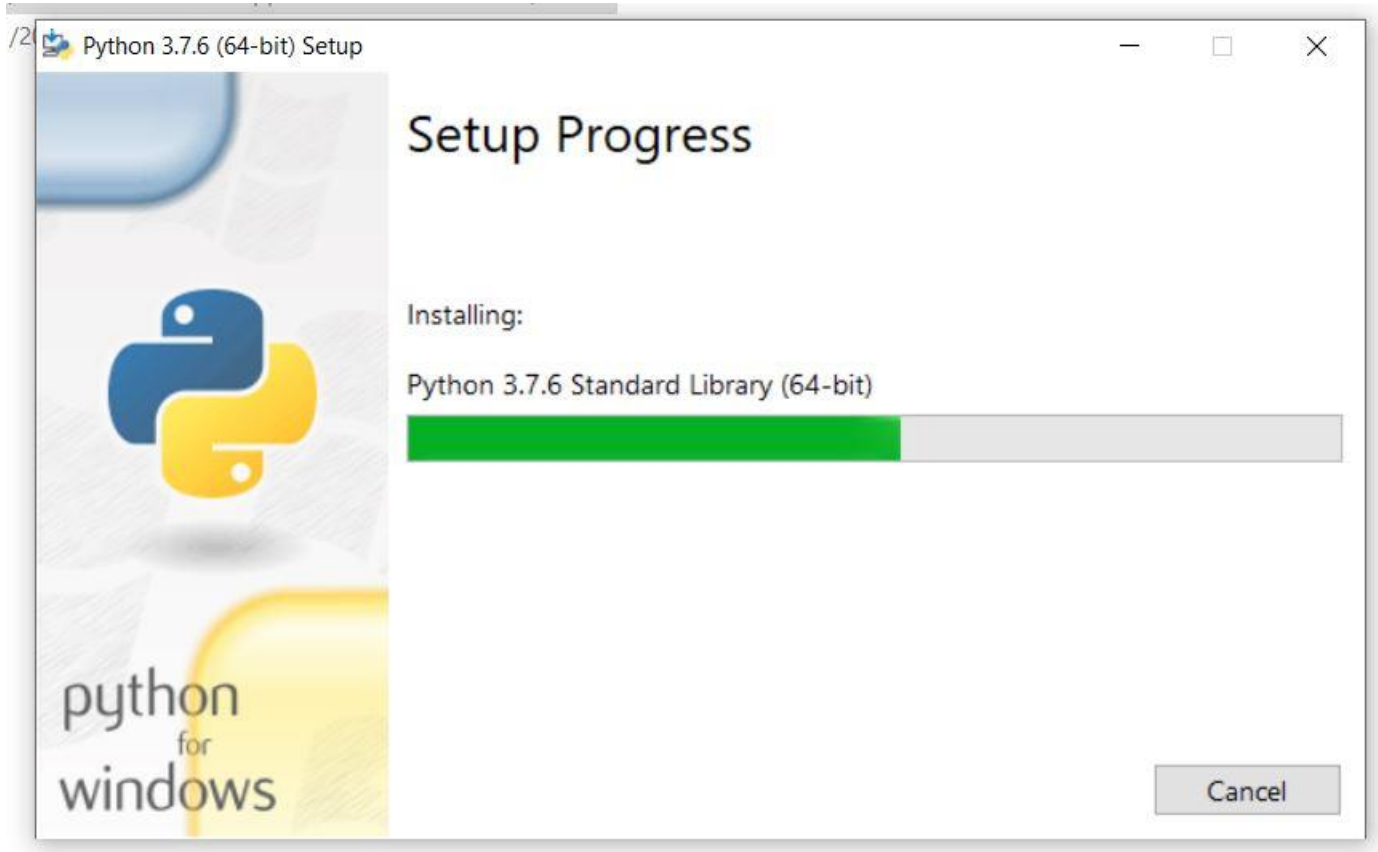
- Step6: choose your suitable installer for your computer.

python.org/downloads/release/python-376/

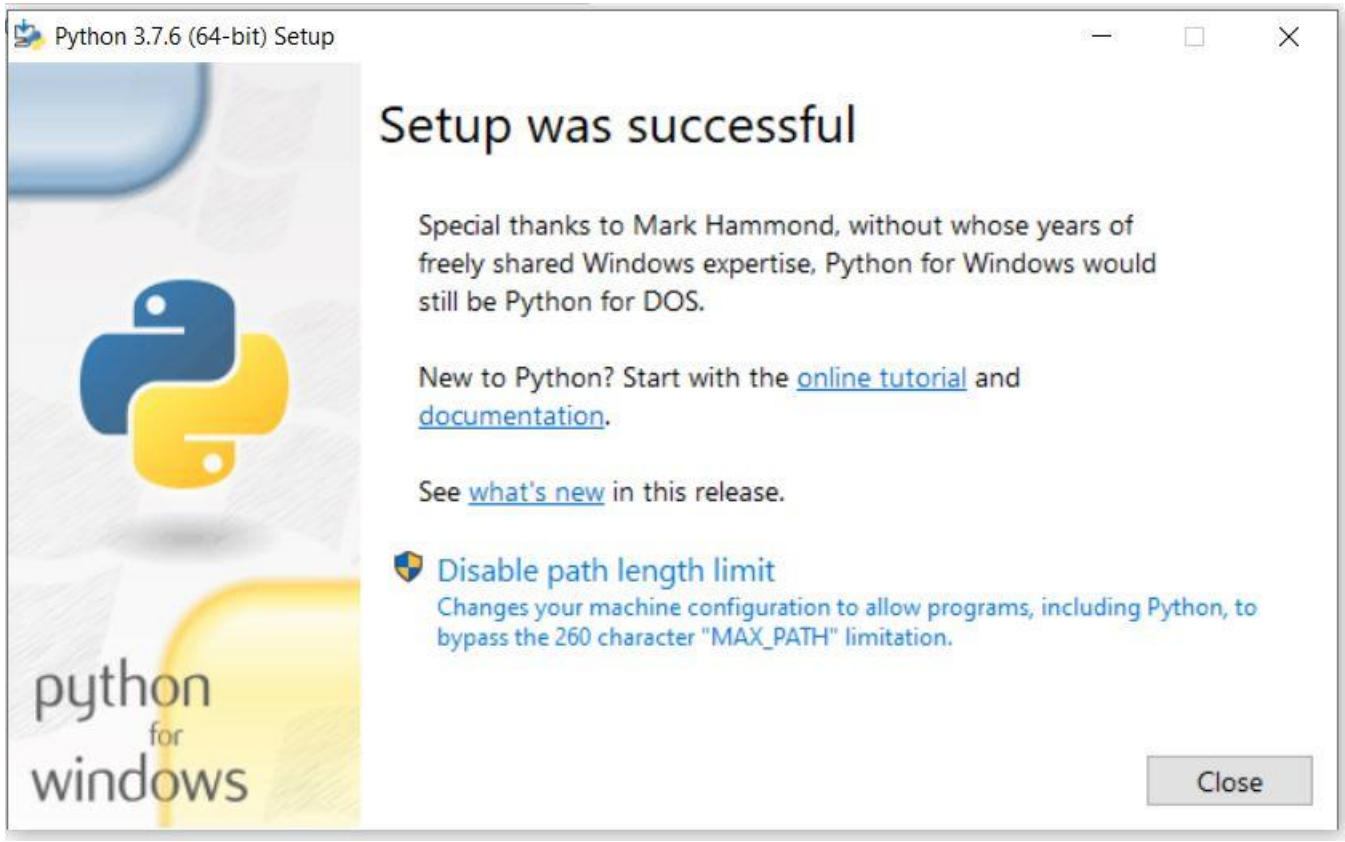
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		3ef90f064506dd85b4b4ab87a7a83d44	23148187	SIG
XZ compressed source tarball	Source release		c08fbee72ad5c2c95b0f4e44bf6fd72c	17246360	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	0dfc4cdd9404cf0f5274d063eca4ea71	35057307	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	57915a926caa15f03ddd638ce714dd3b	28235421	SIG
Windows help file	Windows		8b915434050b29f9124eb93e3e97605b	8158109	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	5f84f4f62a28d3003679dc693328f8fd	7503251	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	cc31a9a497a4ec8a5190edecc5cdd303	26802312	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	f9c11893329743d77801a7f49612ed87	1363000	SIG
Windows x86 embeddable zip file	Windows		accb8a137871ec632f581943c39cb566	6747070	SIG
Windows x86 executable installer	Windows		9e73a1b27bb894f87fdce430ef88b3d5	25792544	SIG
Windows x86 web-based installer	Windows		c7f474381b7a8b90b6f07116d4d725f0	1324840	SIG

- Step7: start install.

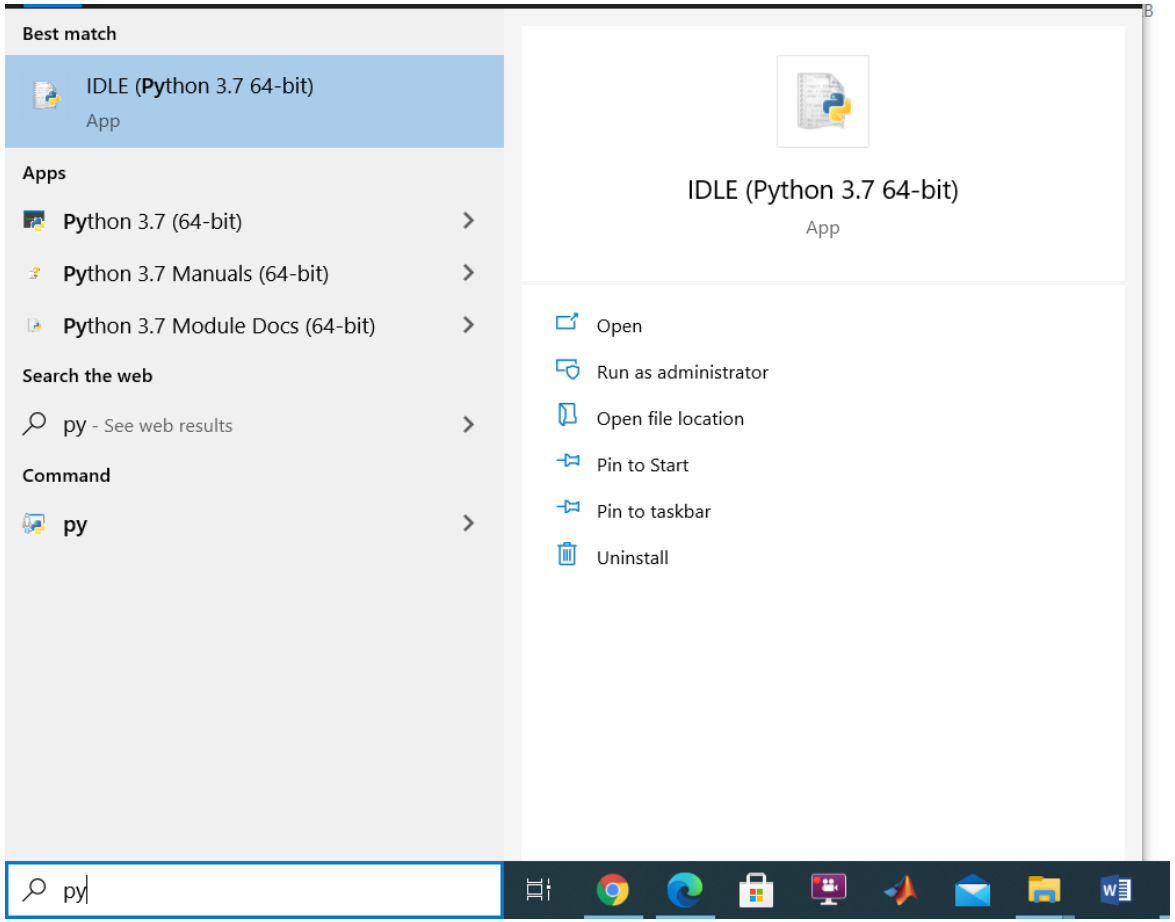
- Step8: .setup progress



- Step10: finish installation.

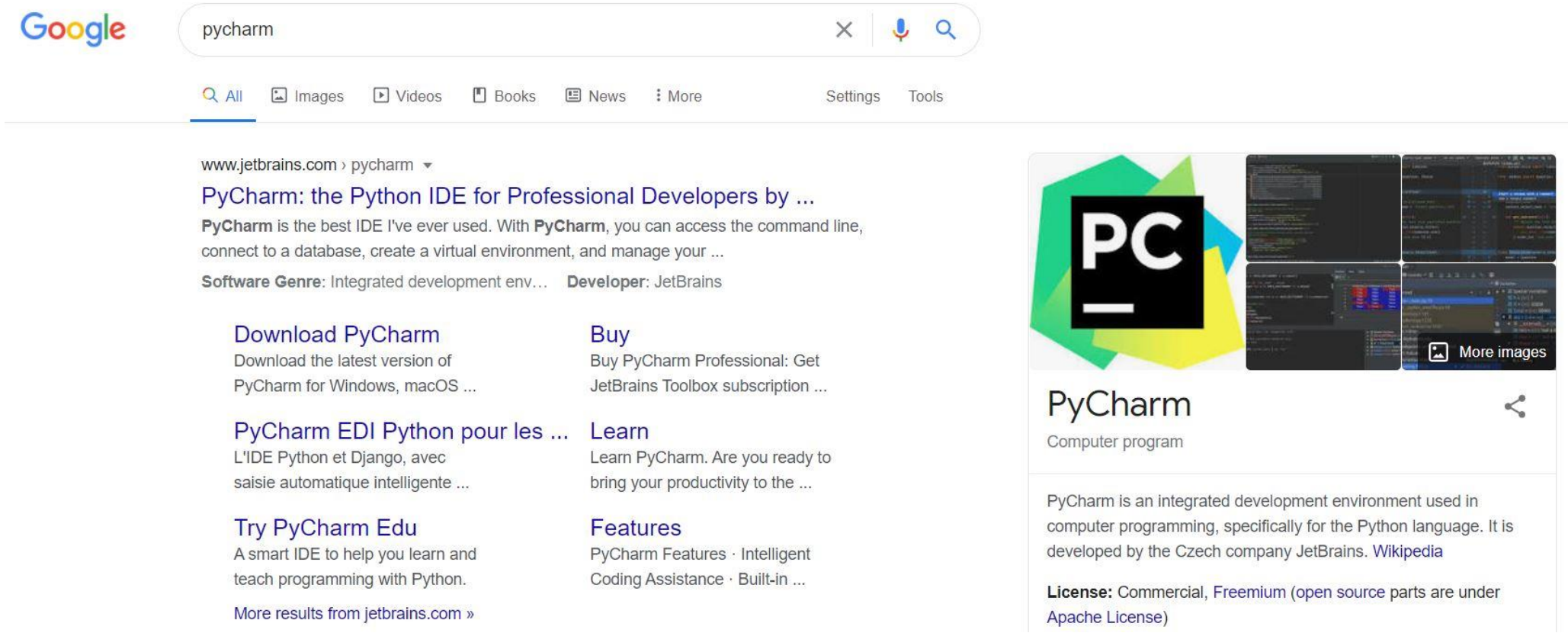


- Step11: try the IDLE python 3.7.6



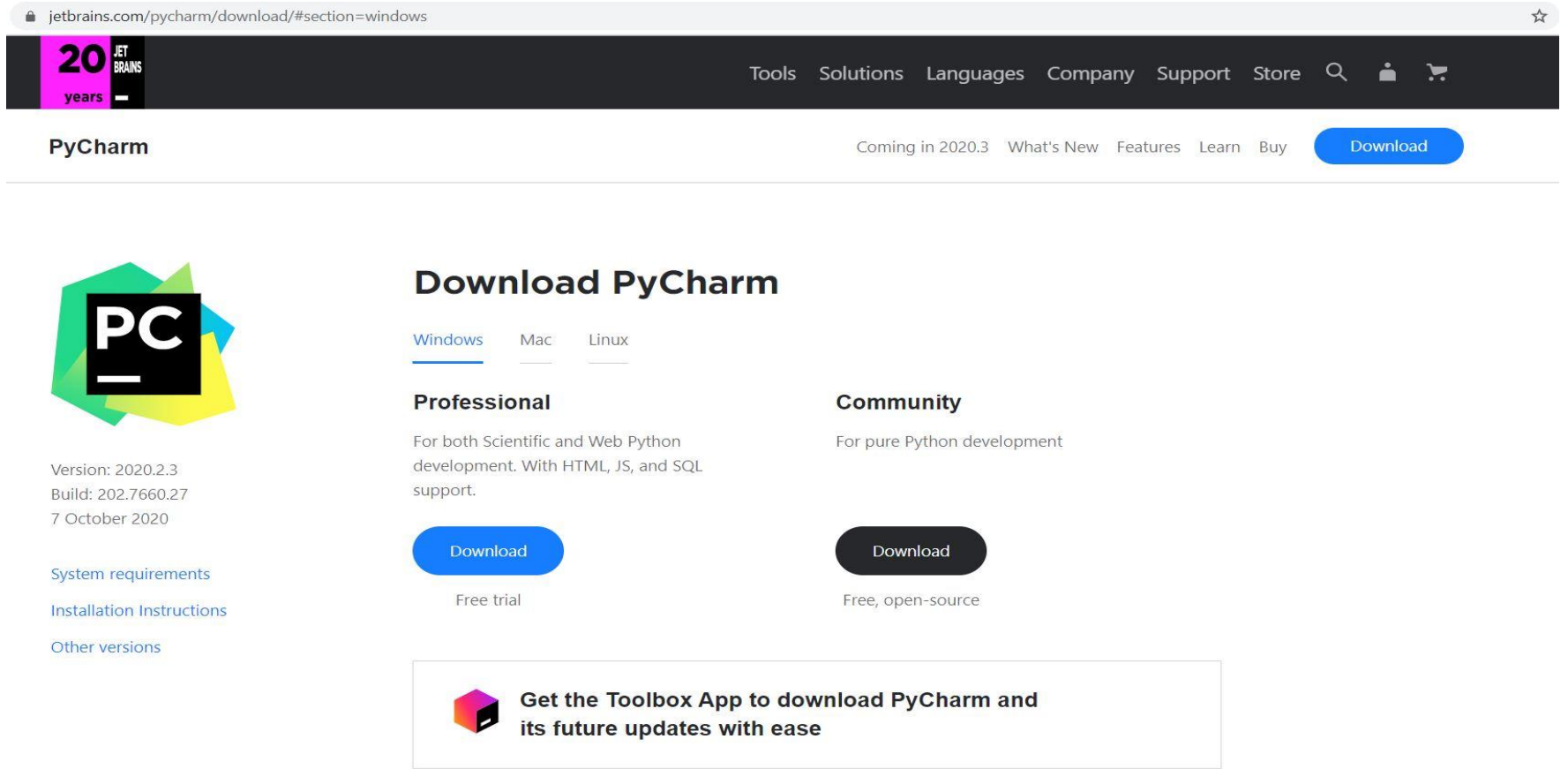
Pycharm Installation

- Step1: login in pycharm website.

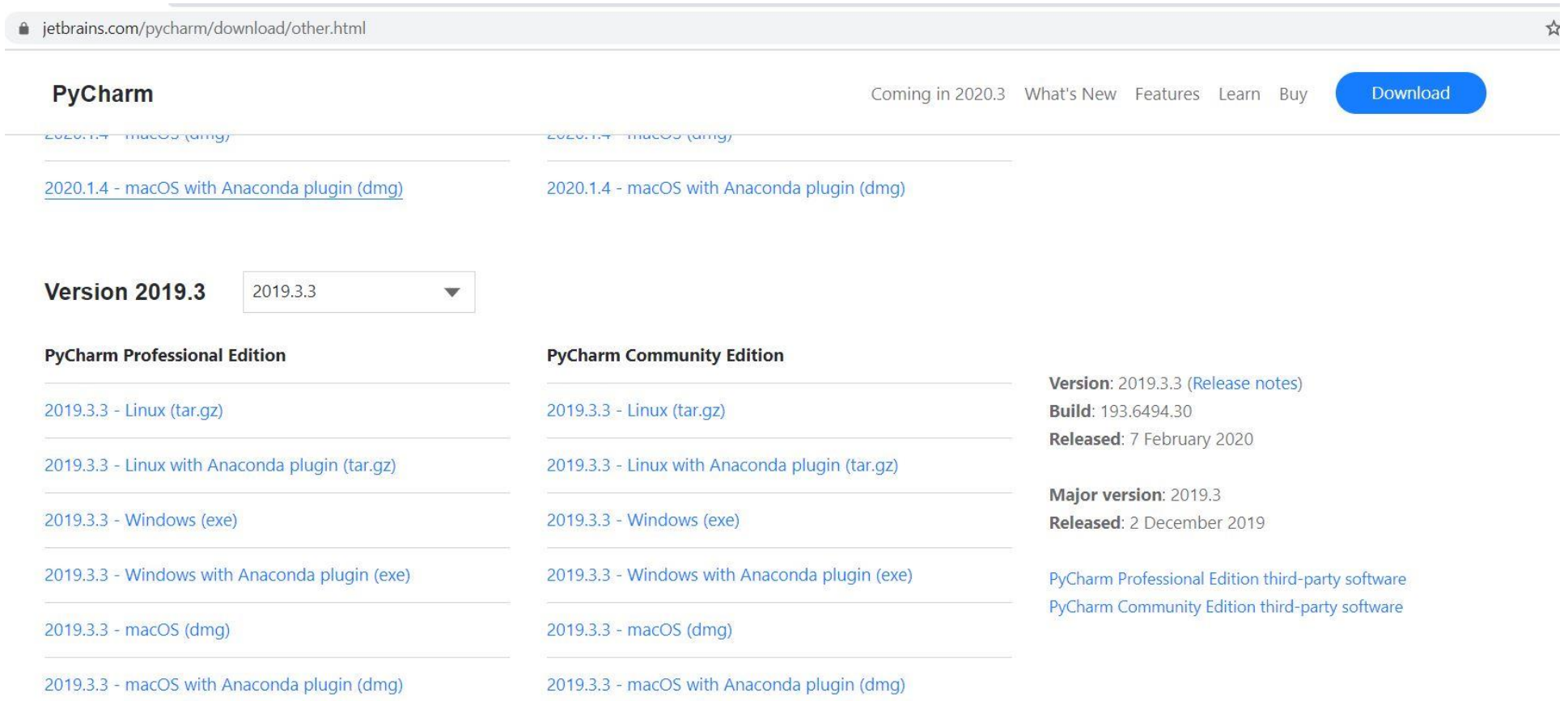


- Step2: click on download.

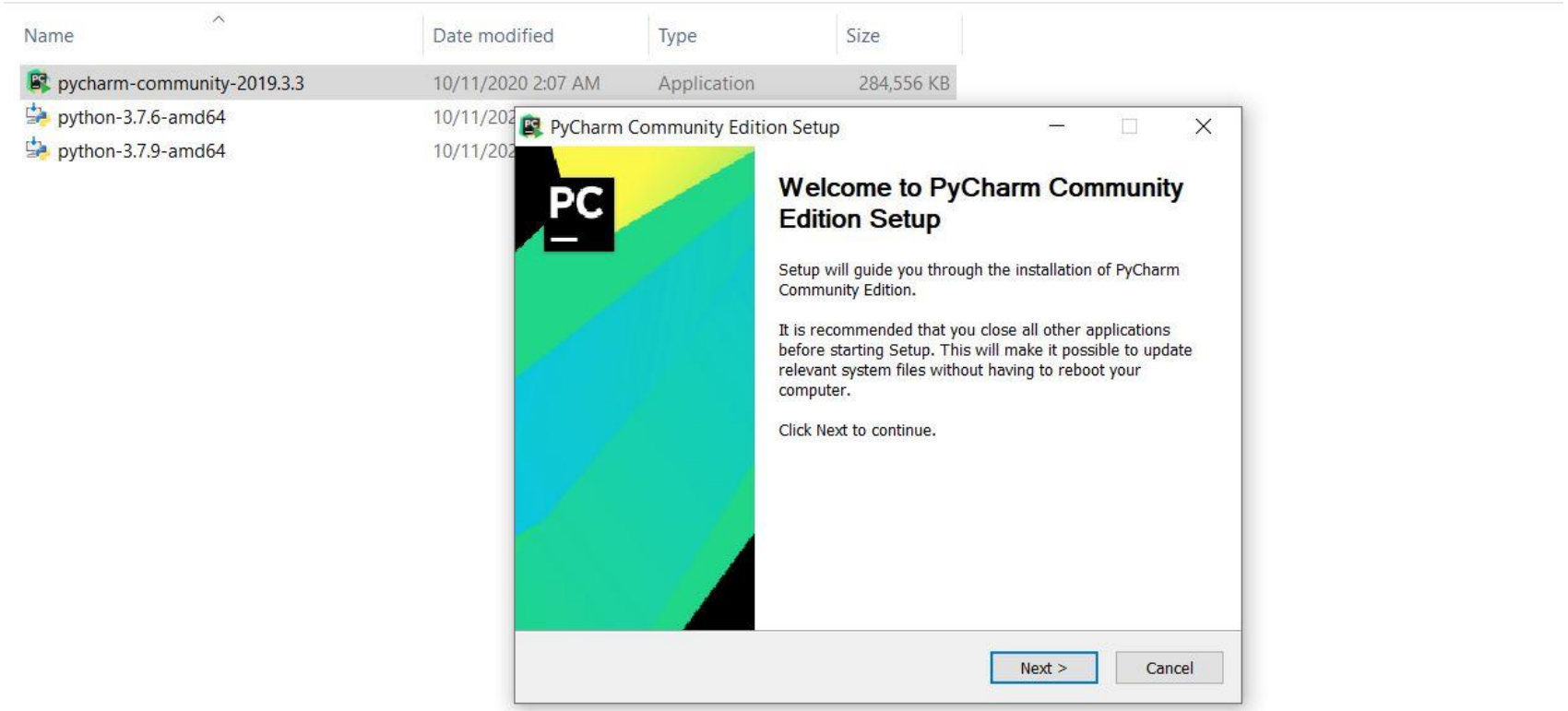
- Step3: click on other versions link.



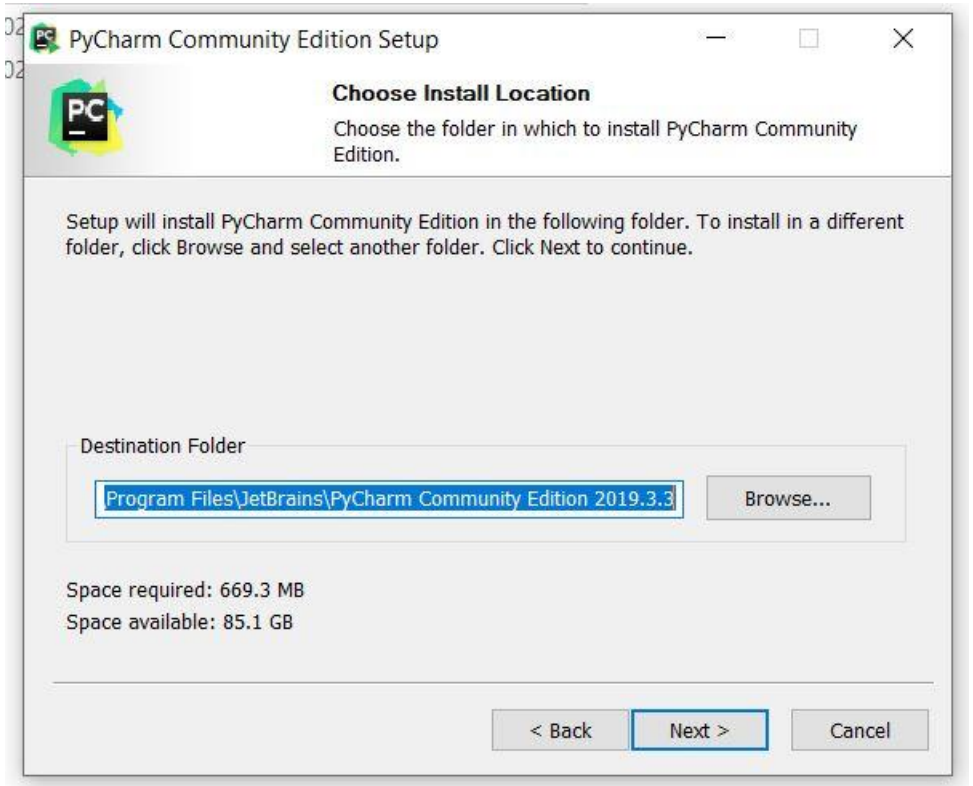
- Step4: choose pycharm version 2019.3.3.



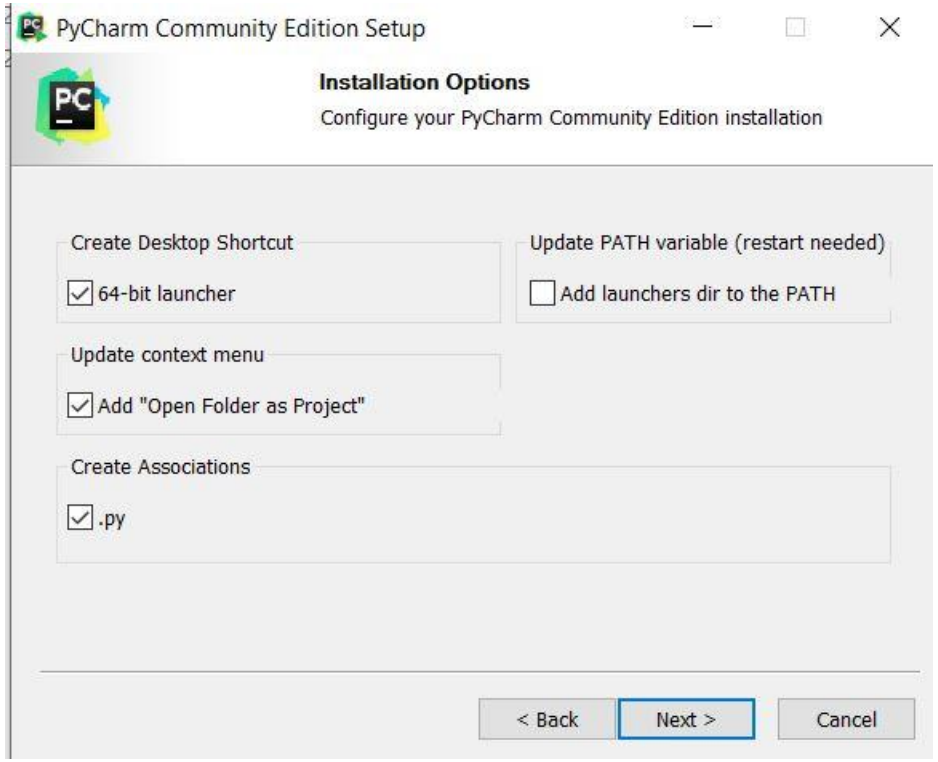
- Step5: start installation.



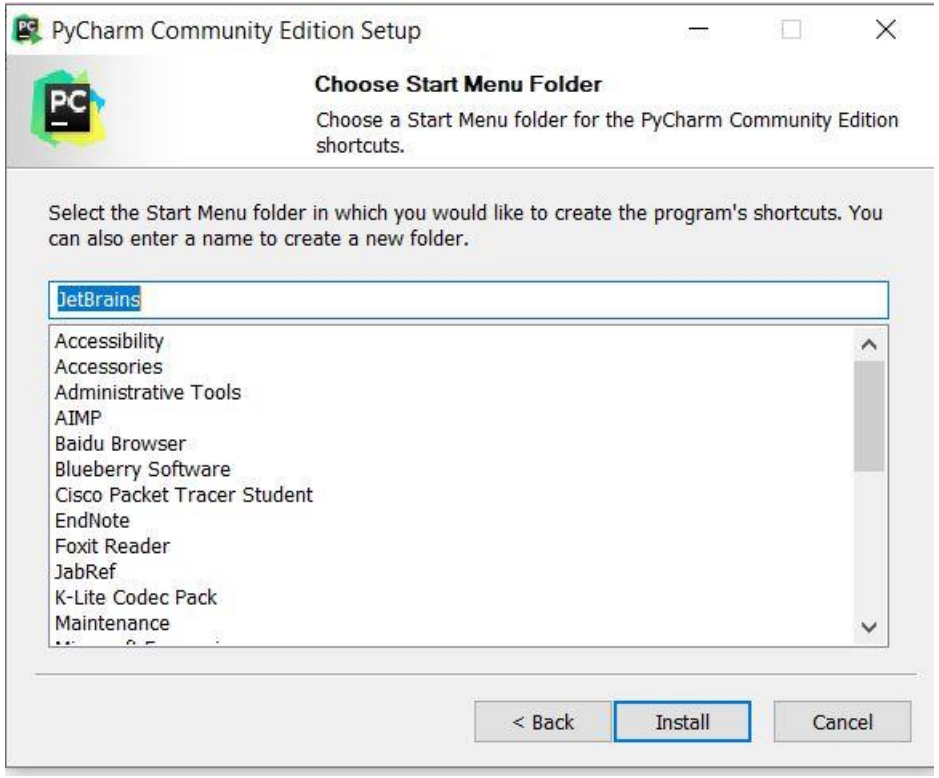
- Step6: complete the installation.



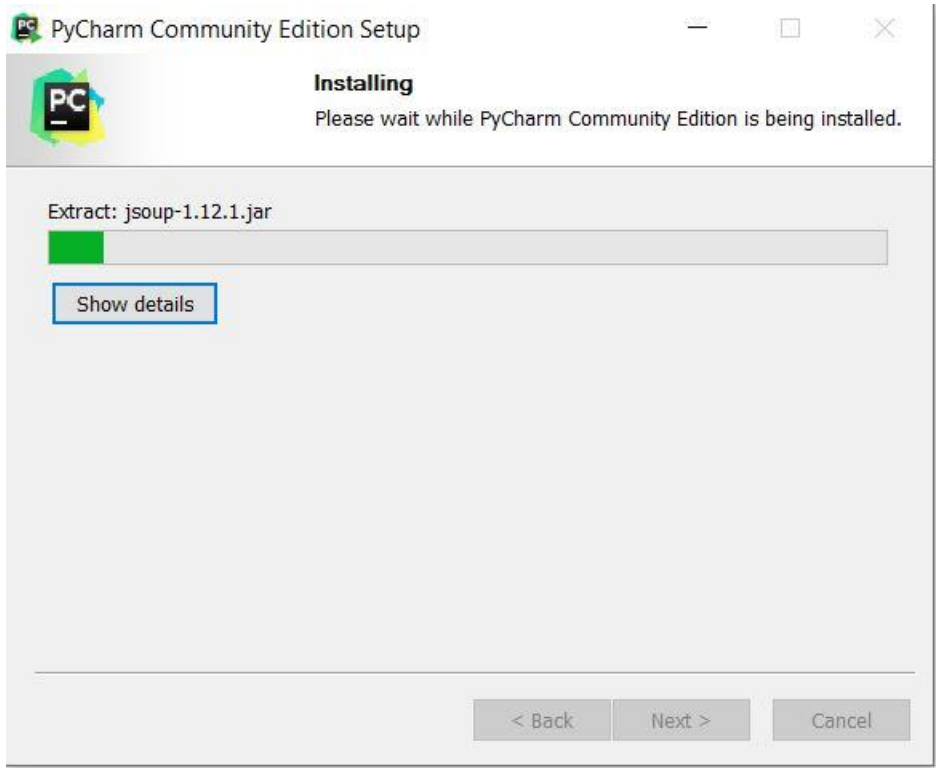
- Step7: complete the installation.



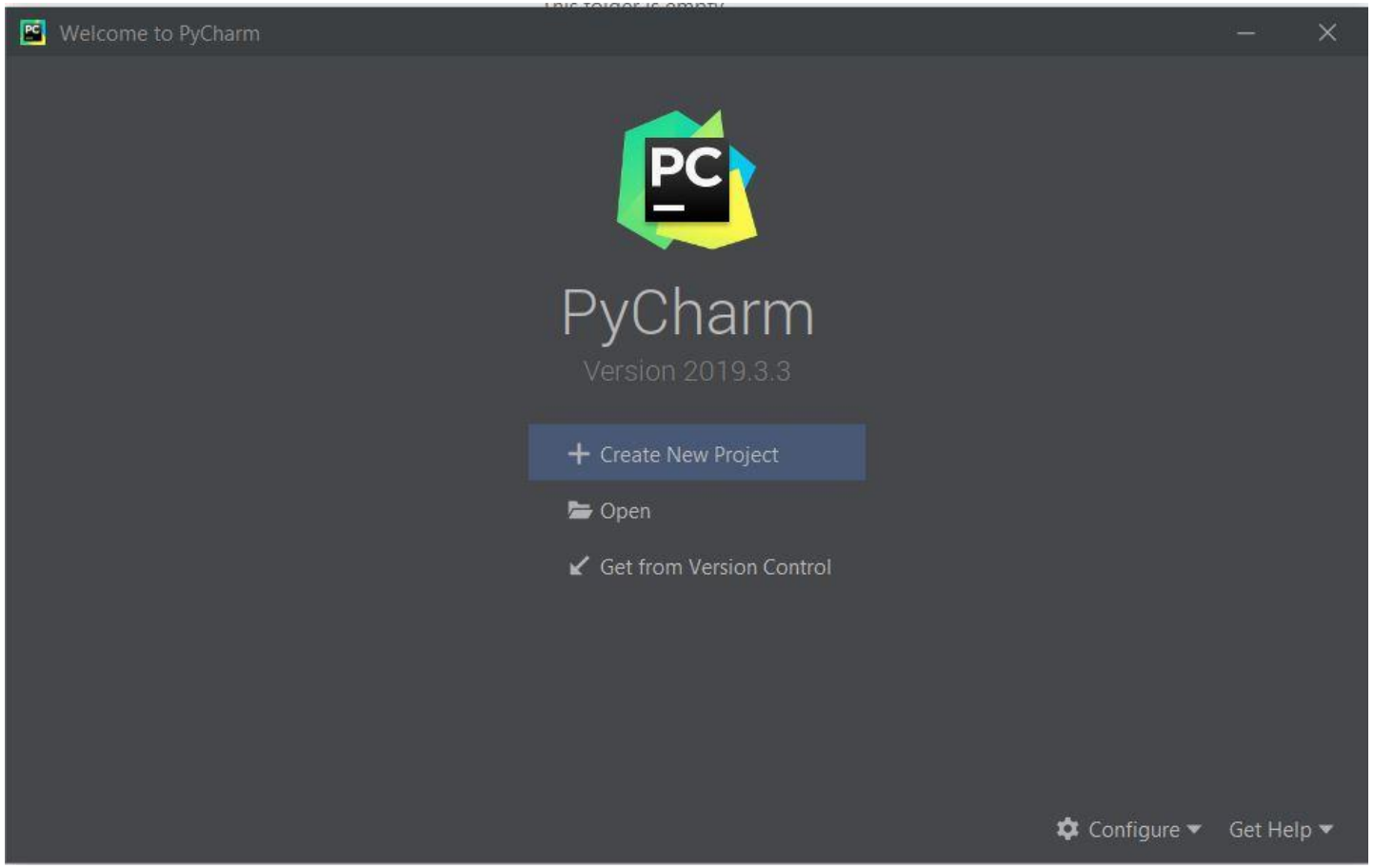
- Step8: complete the installation.



- Step9: complete the installation.

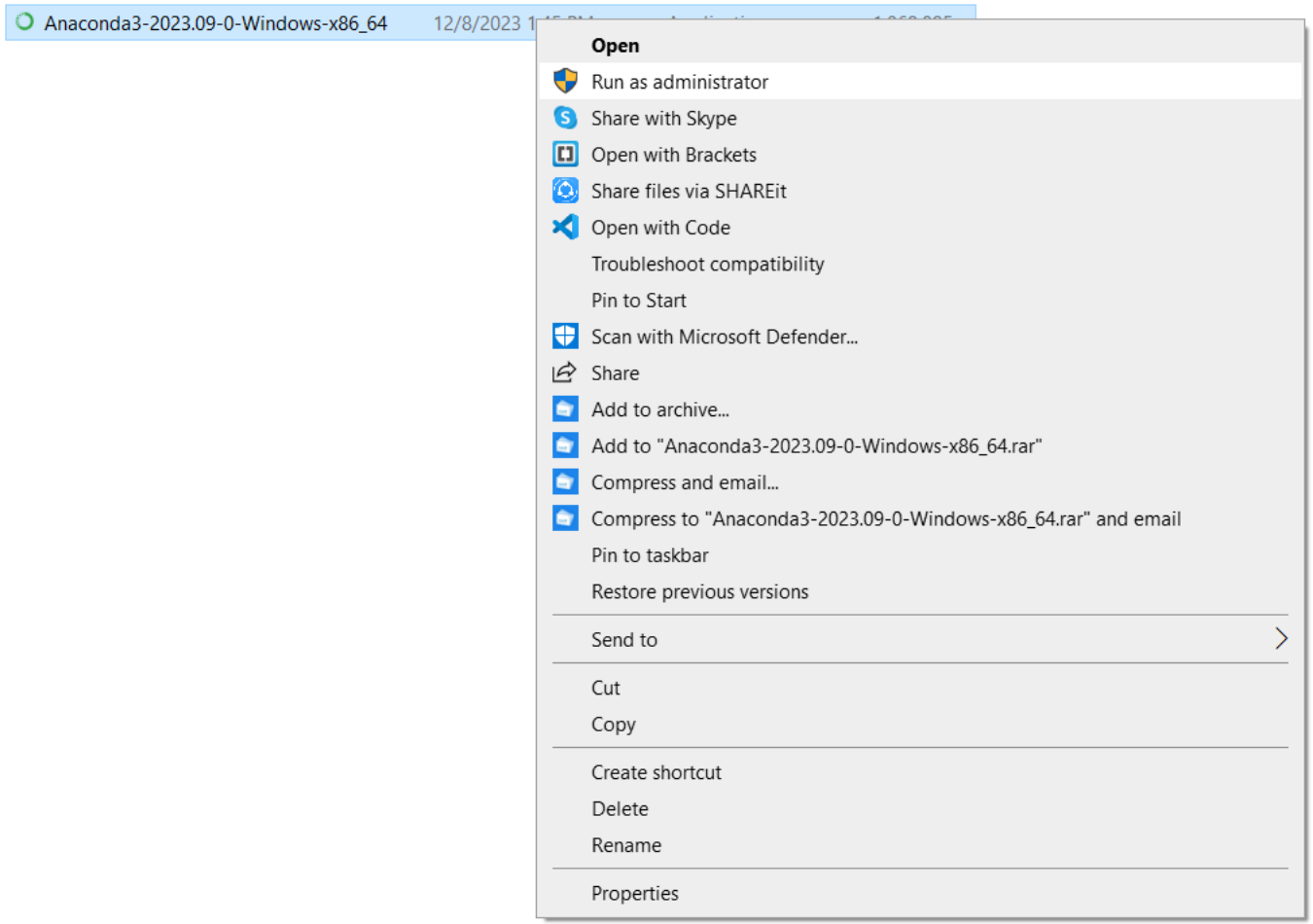


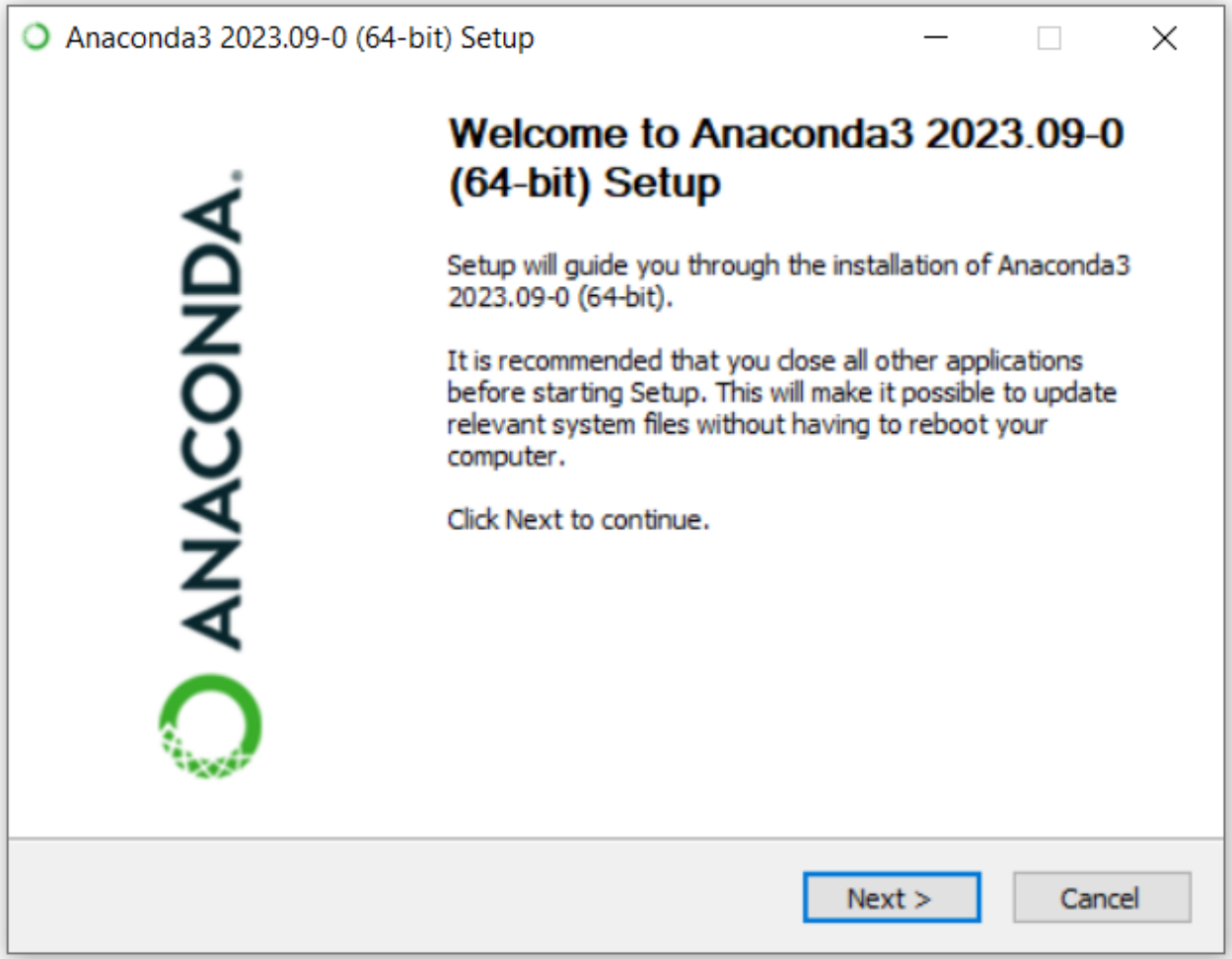
- Step10: finish the installation.

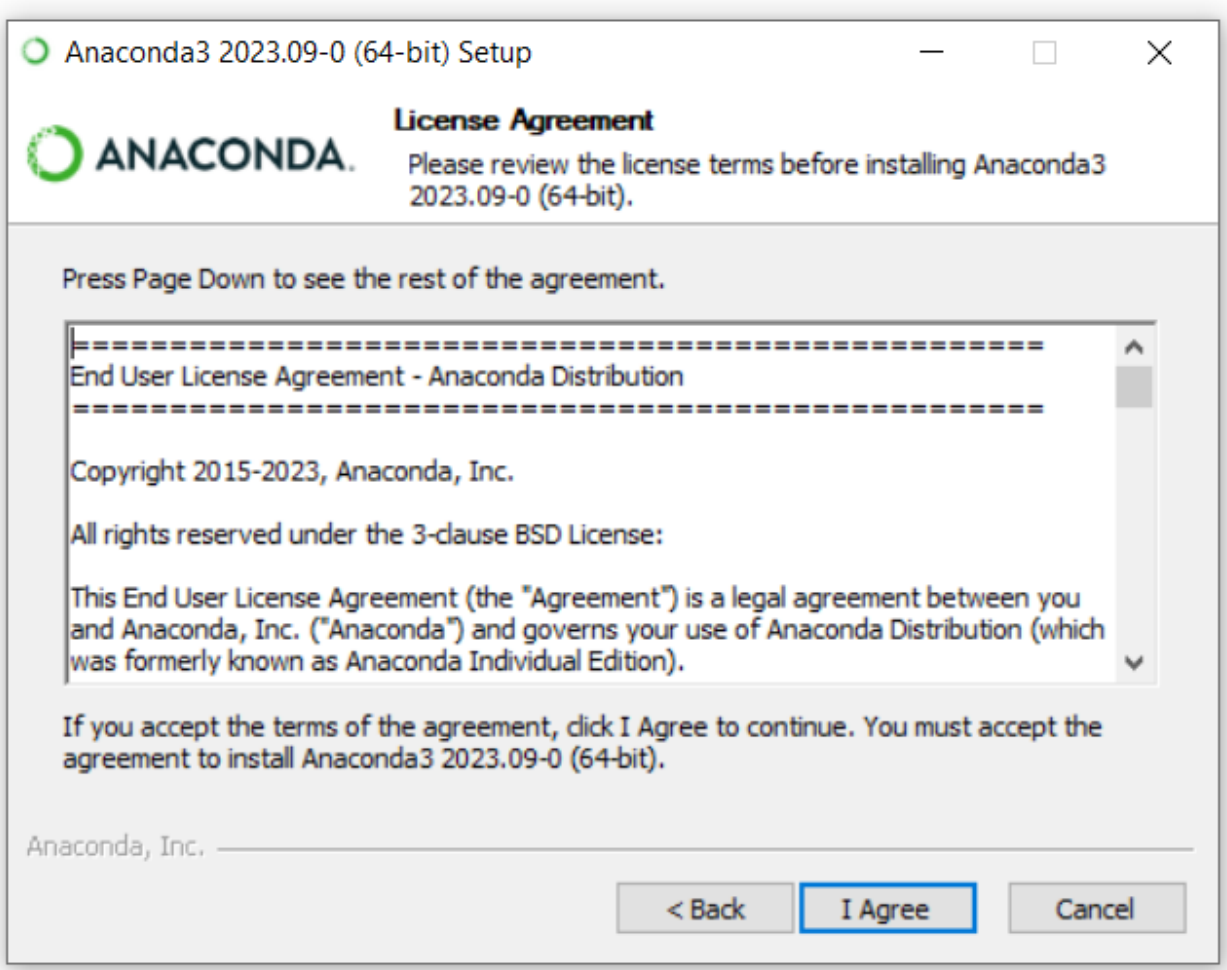


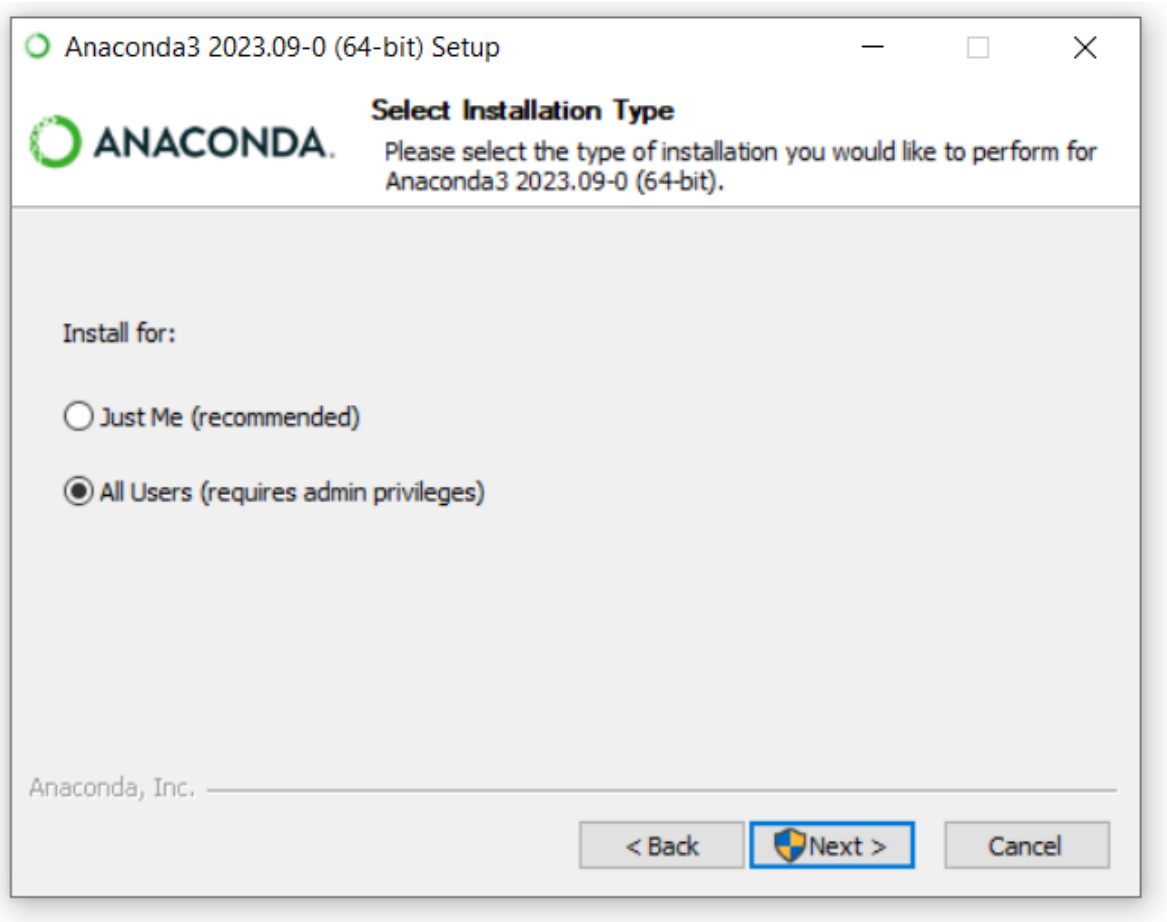
What Is Anaconda For Python?

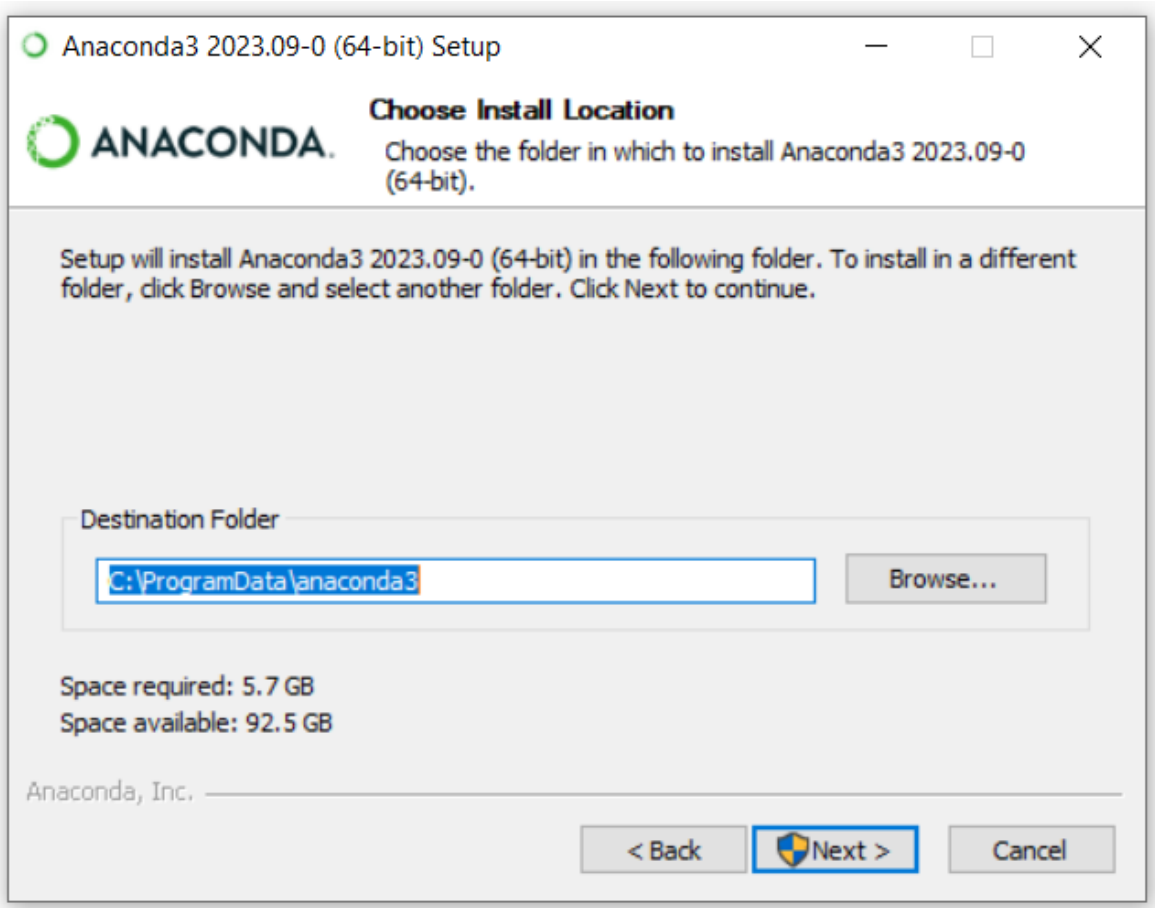
- **Anaconda** Python is a free, open-source platform that allows you to write and execute code in the programming language Python.
- It is by continuum.io, a company that specializes in Python development. The Anaconda platform is the most popular way to learn and use Python for scientific computing, data science, and machine learning.

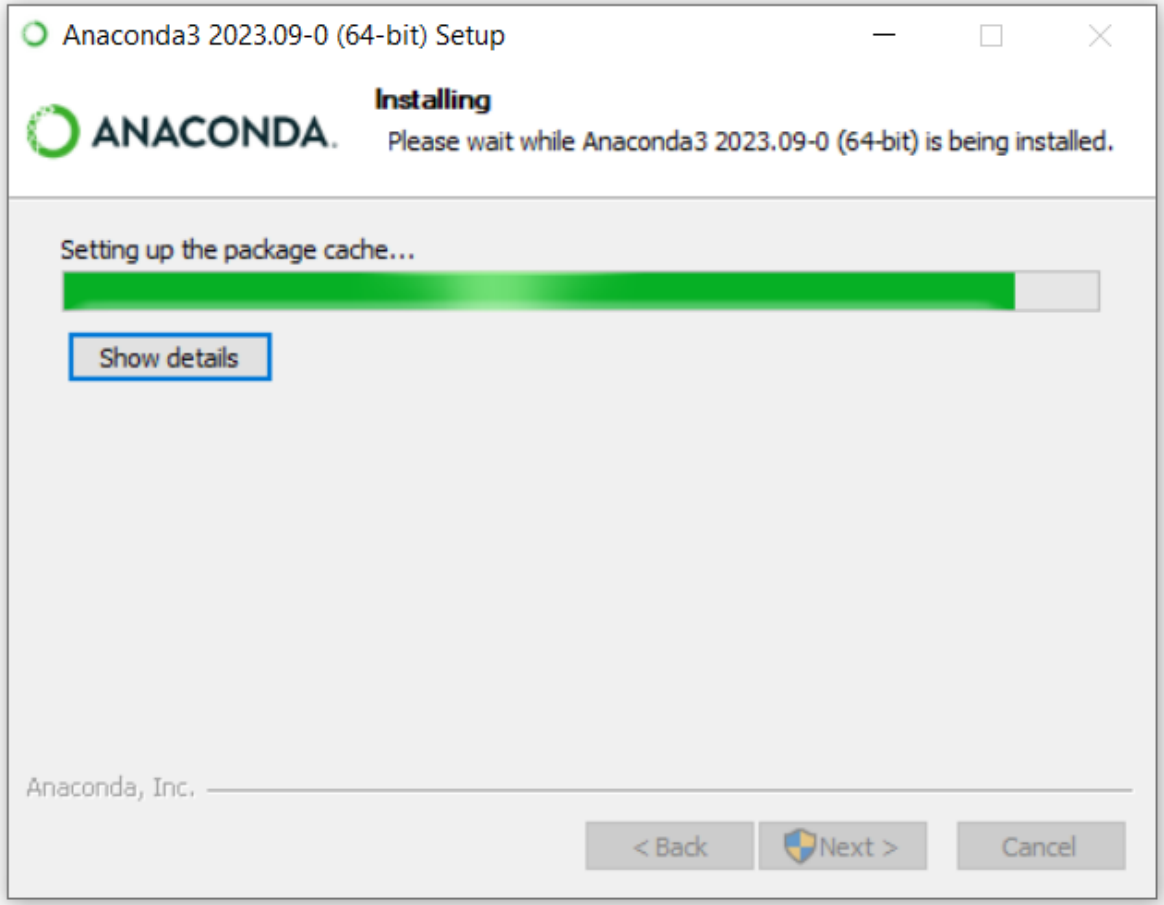

Anaconda3-2023.09-0-Windows-x86_64
Open
Run as administrator
Share with Skype
Open with Brackets
Share files via SHAREit
Open with Code
Troubleshoot compatibility
Pin to Start
Scan with Microsoft Defender...
Share
Add to archive...
Add to "Anaconda3-2023.09-0-Windows-x86_64.rar"
Compress and email...
Compress to "Anaconda3-2023.09-0-Windows-x86_64.rar" and email
Pin to taskbar
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

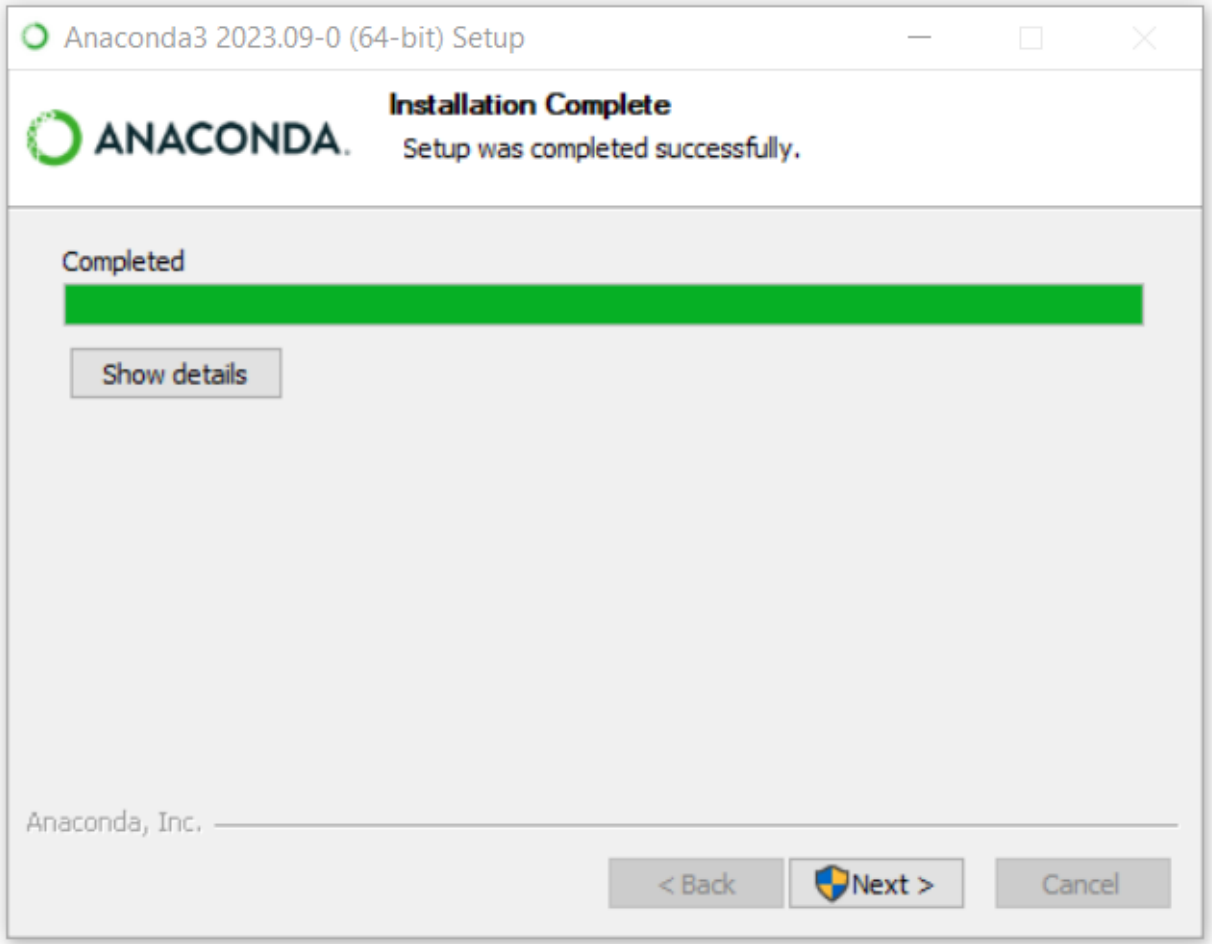

Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Welcome to Anaconda3 2023.09-0 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 2023.09-0 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel

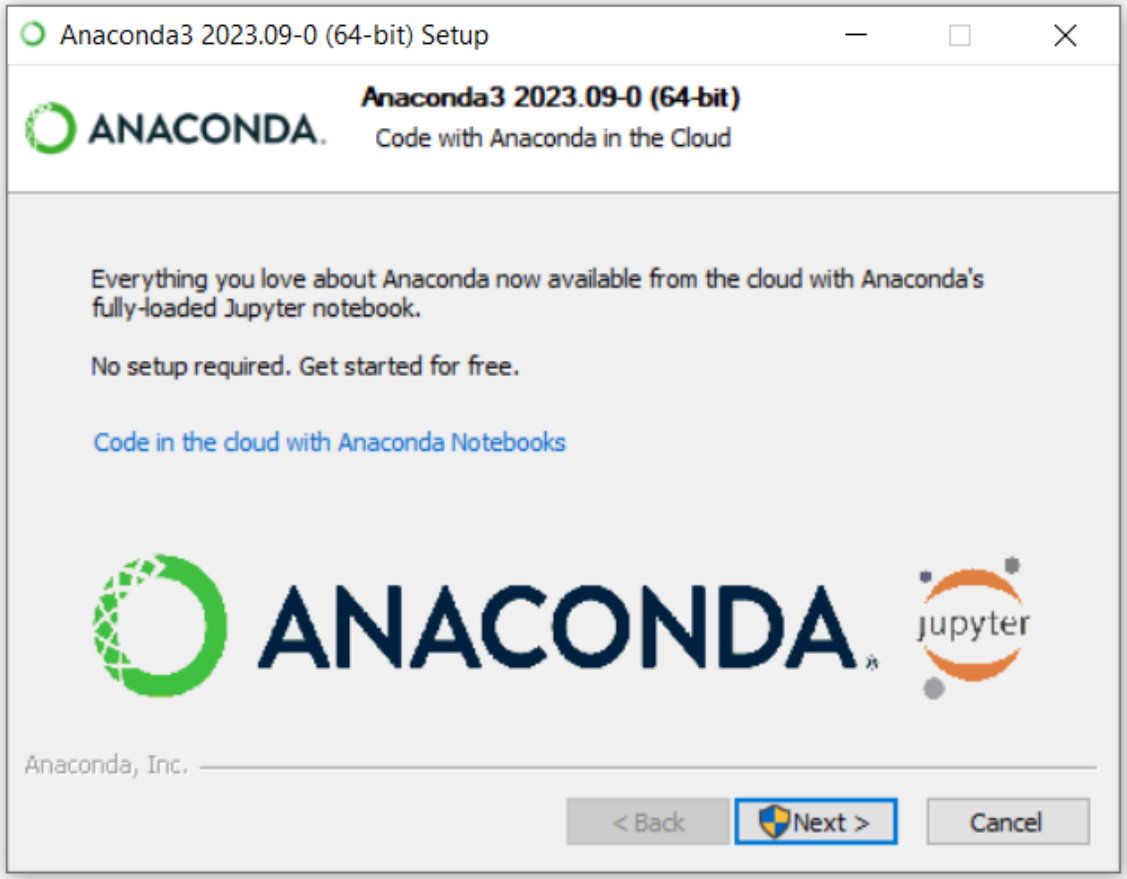

Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
License Agreement
Please review the license terms before installing Anaconda3 2023.09-0 (64-bit).
Press Page Down to see the rest of the agreement.
==========
End User License Agreement - Anaconda Distribution
==========
Copyright 2015-2023, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
This End User License Agreement (the "Agreement") is a legal agreement between you and Anaconda, Inc. ("Anaconda") and governs your use of Anaconda Distribution (which was formerly known as Anaconda Individual Edition).
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2023.09-0 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel

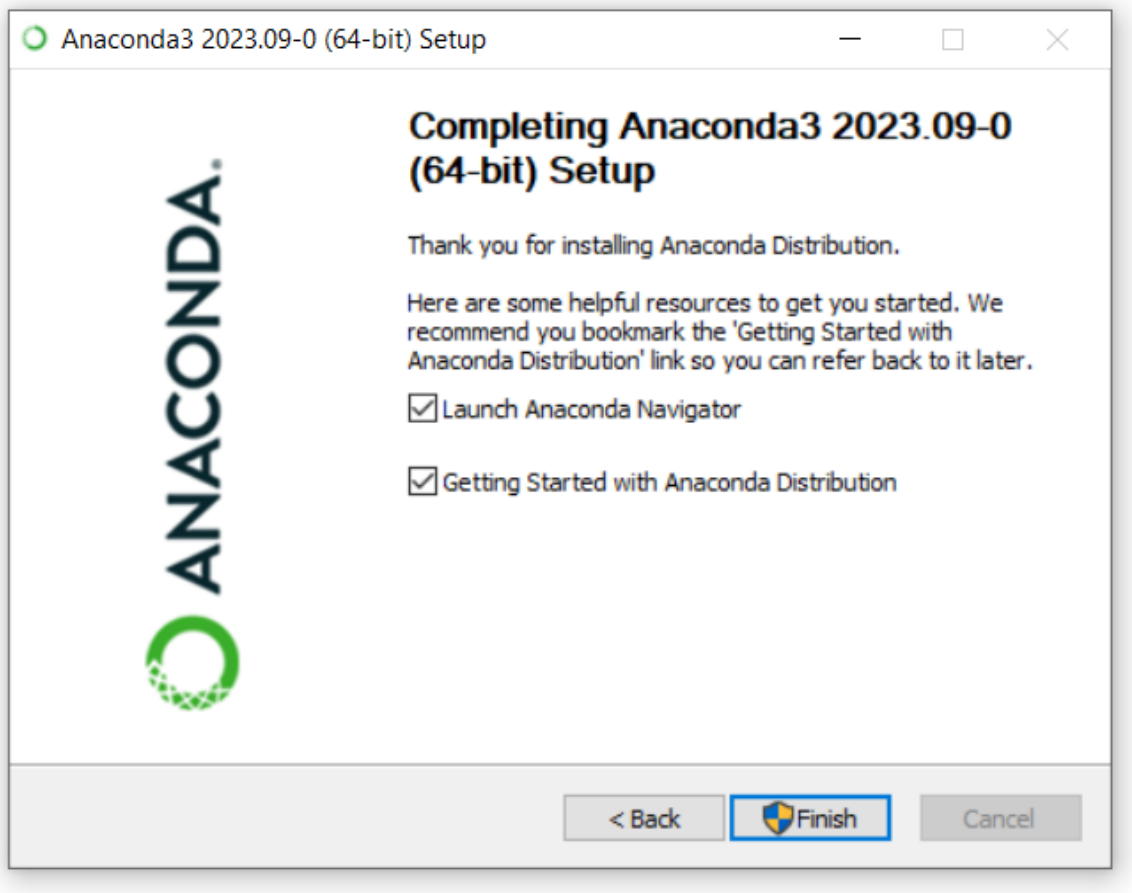

Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Select Installation Type
Please select the type of installation you would like to perform for Anaconda3 2023.09-0 (64-bit).
Install for:
Just Me (recommended)
All Users (requires admin privileges)
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Choose Install Location
Choose the folder in which to install Anaconda3 2023.09-0 (64-bit).
Setup will install Anaconda3 2023.09-0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\ProgramData\anaconda3
Browse...
Space required: 5.7 GB
Space available: 92.5 GB
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Installing
Please wait while Anaconda3 2023.09-0 (64-bit) is being installed.
Setting up the package cache...
Show details
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Installation Complete
Setup was completed successfully.
Completed
Show details
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Anaconda3 2023.09-0 (64-bit)
Code with Anaconda in the Cloud
Everything you love about Anaconda now available from the cloud with Anaconda's fully-loaded Jupyter notebook.
No setup required. Get started for free.
Code in the cloud with Anaconda Notebooks
ANACONDA.
jupyter
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2023.09-0 (64-bit) Setup
ANACONDA.
Completing Anaconda3 2023.09-0 (64-bit) Setup
Thank you for installing Anaconda Distribution.
Here are some helpful resources to get you started. We recommend you bookmark the 'Getting Started with Anaconda Distribution' link so you can refer back to it later.
Launch Anaconda Navigator
Getting Started with Anaconda Distribution
< Back
Finish
Cancel


Initializing...

Python Language

- Python comments
- Python variables
- Python collections
- Python condition
- Python loops
- Python functions

Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Creating a Comment
  - Comments starts with a #, and Python will ignore them:

```
#This is a comment
print("Hello, World!")
```

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

```
print("Hello, World!") #This is a comment
```

- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multiline Comments

- Python does not really have a syntax for multiline comments.
- To add a multiline comments you could insert a # for each line:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python Variables

- Variables are containers for storing data values.
- Creating Variables:
  - Python has no command for declaring a variable.
  - A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

- Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

- Casting: If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

- Get the Type: You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

- Single or Double Quotes?
  - String variables can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John'
```

- Case-Sensitive: Variable names are case-sensitive.

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Variable Names

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). 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)

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

- Multi Words Variable Names:
- Variable names with more than one word can be difficult to read.
- There are several techniques you can use to make them more readable:
  - **Camel Case**
    - Each word, except the first, starts with a capital letter:
      - myVariableName = "John“

- **Pascal Case**
  - Each word starts with a capital letter:
    - MyVariableName = "John"
- **Snake Case**
  - Each word is separated by an underscore character:
    - my_variable_name = "John"

- Many Values to Multiple Variables: Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- One Value to Multiple Variables: And you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Output Variables

- The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

- In the print() function, you output multiple variables, separated by a comma:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

- You can also use the + operator to output multiple variables:

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

- Notice the space character after "Python " and "is ", without them the result would be "Pythonisawesome".

- In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

```
x = 5
y = 10
print(x + y)
```

- The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x + y)
```

- The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x, y)
```

Global Variables

- Variables that are created outside of a function (as in all of the examples above) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

- If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

The global Keyword

- Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.
- To create a global variable inside a function, you can use the global keyword.

If you use the `global` keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

- Also, use the global keyword if you want to change a global variable inside a function.

To change the value of a global variable inside a function, refer to the variable by using the `global` keyword:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Python Data Types

- Built-in Data Types
  - In programming, data type is an important concept.
  - Variables can store data of different types, and different types can do different things.
  - Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

- Getting the Data Type: You can get the data type of any object by using the type() function.

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Setting the Data Type

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Setting the Specific Data Type

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Python Numbers

- There are three numeric types in Python:
  - int
  - float
  - complex
- Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

- To verify the type of any object in Python, use the type() function:

```
print(type(x))
print(type(y))
print(type(z))
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 356562225548887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

- Float can also be scientific numbers with an "e" to indicate the power of 10.

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

- You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Random Number

- Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

Python Lists

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets:

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Items

- List items are **ordered**, **changeable**, and **allow duplicate values**.
- List items are **indexed**, the first item has index [0], the second item has index [1] etc.
- **Ordered**
  - When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
  - If you add new items to a list, the new items will be placed at the end of the list.
- **Changeable**
  - The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
- **Allow Duplicates**
  - Since lists are indexed, lists can have items with the same value:

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

- List Length: To determine how many items a list has, use the len() function:

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

- List Items - Data Types:

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

- A list can contain different data types:

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

- type(): From Python's perspective, lists are defined as objects with the data type 'list’:

What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

- The list() Constructor: It is also possible to use the list() constructor when creating a new list.

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

- Python Collections (Arrays):
- There are four collection data types in the Python programming language:
  - **List** is a collection which is **ordered** and **changeable**. **Allows duplicate members**.
  - **Tuple** is a collection which is **ordered** and **unchangeable**. **Allows duplicate members**.
  - **Set** is a collection which is **unordered**, **unchangeable***, and **unindexed**. **No duplicate members**.
  - **Dictionary** is a collection which is **ordered**** and **changeable**. No **duplicate members.**

Python - Access List Items

- List items are indexed and you can access them by referring to the index number:

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

- Negative Indexing: Negative indexing means start from the end
- -1 refers to the last item, -2 refers to the second last item etc.

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

- Range of Indexes:
- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

This example returns the items from the beginning to, but NOT including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

This example returns the items from "cherry" to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

- Check if Item Exists: To determine if a specified item is present in a list use the in keyword

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

- **Change Item Value**: To change the value of a specific item, refer to the index number:

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- **Change a Range of Item Values**: To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

- **Insert Items**:
- To insert a new list item, without replacing any of the existing values, we can use the insert() method.
- The insert() method inserts an item at the specified index:

Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Python - Add List Items

- **Append Items**: To add an item to the end of the list, use the append() method:

Using the `append()` method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- **Insert Items**: To insert a list item at a specified index, use the insert() method.

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

- **Extend List**: To append elements from another list to the current list, use the extend() method.

Add the elements of `tropical` to `thislist`:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

- **Add Any Iterable**: The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Python - Remove List Items

- **Remove Specified Item**: The remove() method removes the specified item.

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- **Remove Specified Index**: The pop() method removes the specified index.

Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

- If you do not specify the index, the pop() method **removes the last item**.

Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

- The del keyword also **removes the specified index**:

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

- The **del keyword** can also delete the list completely.

Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

- **Clear the List**: The clear() method empties the list.

Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Python - Loop Lists

- Loop Through a List: You can loop through the list items by using a for loop:

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

- Loop Through the Index Numbers: You can also loop through the list items by referring to their index number.

Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.
- Example:
- Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
- Without list comprehension you will have to write a for statement with a conditional test inside:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

- With list comprehension you can do all that with only one line of code:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

Python - Sort Lists

- Sort List Alphanumerically: List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Sort the list alphabetically:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

- Sort Descending: To sort descending, use the keyword argument reverse = True:

Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Sort the list descending:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

Python - Copy Lists

- You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.
- There are ways to make a copy, one way is to use the built-in List method copy().

Make a copy of a list with the `copy()` method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

- Another way to make a copy is to use the built-in method list().

Make a copy of a list with the `list()` method:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Python - Join Lists

- Join Two Lists:
- There are several ways to join, or concatenate, two or more lists in Python.
- One of the easiest ways are by using the + operator.

Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

- Another way to join two lists is by appending all the items from list2 into list1, one by one:

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

- Or you can use the extend() method, which purpose is to add elements from one list to another list:

Use the `extend()` method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python Tuples

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

- Tuple Items: Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

- Tuple Length: To determine how many items a tuple has, use the len() function:

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

- Create Tuple With One Item: To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

- Tuple Items - Data Types:

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

- The tuple() Constructor: It is also possible to use the tuple() constructor to make a tuple.

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets:

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Python - Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.
- But there are some workarounds.
- Change Tuple Values:
- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

- Add Items:
- Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.
- 1. Convert into a list: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

- 2. Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

- Remove Items:
- **Note:** You cannot remove items in a tuple.
- Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

- Or you can delete the tuple completely:

The `del` keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Python - Loop Tuples

- Loop Through a Tuple: You can loop through the tuple items by using a for loop.

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Python - Join Tuples

- Join Two Tuples: To join two or more tuples you can use the + operator:

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

- Multiply Tuples: If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

Python - Tuple Methods

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Python Sets

- Sets are used to store multiple items in a single variable.
- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
- * **Note:** Set *items* are unchangeable, but you can remove items and add new items.

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the `len()` function.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

type()

From Python's perspective, sets are defined as objects with the data type 'set':

```
<class 'set'>
```

Example

What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

The set() Constructor

It is also possible to use the `set()` constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Python - Access Set Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Python - Add Set Items

- Add Items: To add one item to a set use the add() method.

Add an item to a set, using the `add()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

- Add Sets: To add items from another set into the current set, use the update() method.

Add elements from `tropical` into `thisset`:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

- Add Any Iterable: The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Add elements of a list to at set:

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

Python - Remove Set Items

- To remove an item in a set, use the remove(), or the discard() method.

Remove "banana" by using the `remove()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Note: If the item to remove does not exist, `remove()` will raise an error.

Example

Remove "banana" by using the `discard()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Note: If the item to remove does not exist, `discard()` will **NOT** raise an error.

You can also use the `pop()` method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.

The return value of the `pop()` method is the removed item.

Remove a random item by using the `pop()` method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

Note: Sets are *unordered*, so when using the `pop()` method, you do not know which item that gets removed.

Example

The `clear()` method empties the set:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Python - Loop Sets

- You can loop through the set items by using a for loop:

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Python - Join Sets

- There are several ways to join two or more sets in Python.
- You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

The `union()` method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

The `update()` method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

Python - Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Python If ... Else

- Python Conditions and If statements:

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- Indentation
- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

Elif

The `elif` keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

The `else` keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

You can also have an `else` without the `elif`:

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

And

The `and` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The `or` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Not

The `not` keyword is a logical operator, and is used to reverse the result of the conditional statement:

Example

Test if `a` is NOT greater than `b`:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

Nested If

You can have `if` statements inside `if` statements, this is called *nested* `if` statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Python While Loops

- Python has two primitive loop commands:
  - while loops
  - for loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

The else Statement

With the `else` statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

PYTHON FOR LOOPS

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The `for` loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when `x` is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Using the range() function:

```
for x in range(6):
  print(x)
```

Note that `range(6)` is not the values of 0 to 6, but the values 0 to 5.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 6)`, which means values from 2 to 6 (but not including 6):

Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: `range(2, 30, 3)`:

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Note: The `else` block will NOT be executed if the loop is stopped by a `break` statement.

Example

Break the loop when `x` is 3, and see what happens with the `else` block:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Thank you