Introduction to Python

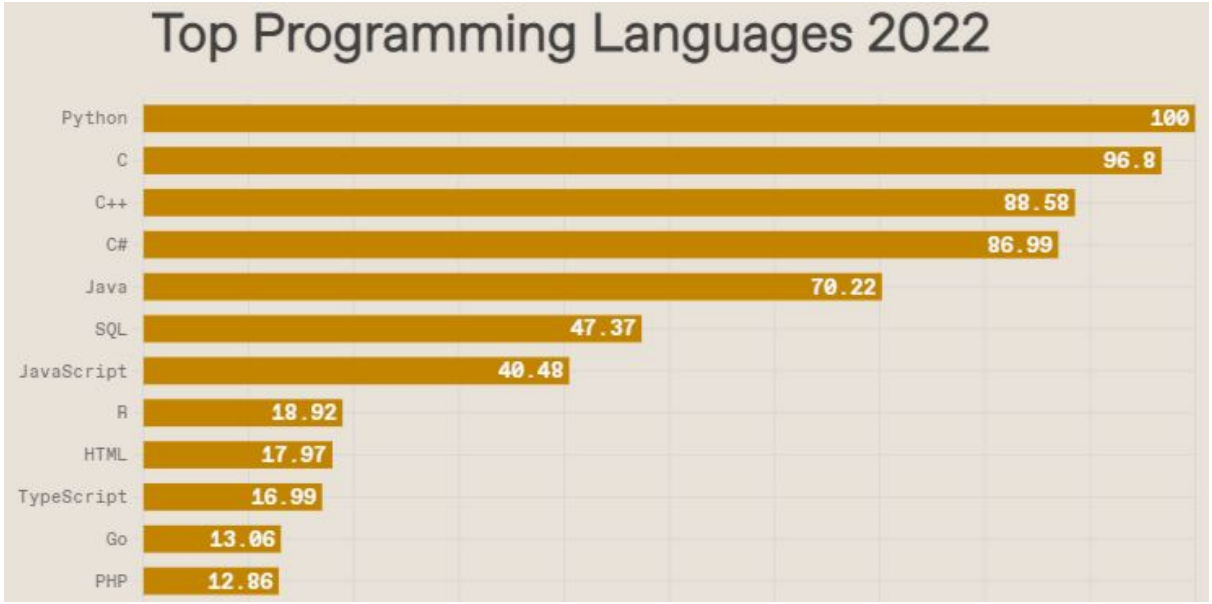
What is Python:

Python is a high-level object oriented programming language created by Guido Van Rossum. It is used in most of the domains and hence called as general purpose language too. The domains where it is used:

- Artificial Intelligence
- Machine Learning
- Deep Learning
- Data Analytics
- Game Development
- Software Development
- Web Development

Why Python?

- Python is very simple to understand
- Python is scalable making development speed fast
- Simple English like syntax
- Allow developers to write programs with fewer lines of code
- It is open-source
- Many libraries are available to make work easy
- Has power to handle large datasets
- Python has been listed as top programming language in IEEE spectrum list in 2021 and 2022. Python is open-source and its source-code is available publically at GNU general public license.



Advantages of Python

- Python is interpreted.
- Python is interactive.
- Python is a beginner’s language.
- Python supports structural, functional and object-oriented programming methods.
- Provides high level dynamic datatypes and supports type-checking dynamically.
- It can easily be integrated with C, C++, Matlab, Java etc.
- It has advantage of automatic garbage collection

Applications of Python

- Python has few keywords, clearly defined syntax and simple structure. This makes Python an easy-to-learn language.
- Python is written in very simple way making its code more readable and visible to eyes.
- The source code of Python is easy to maintain.
- Python comes with bulk of libraries which are portable and compatible with most platforms like Windows, Linux, Macintosh etc.
- Python supports interactive mode to perform interactive testing and debugging.
- Provides interfaces to almost all major commercial databases.
- It supports GUI applications.

Getting started with Python

- It is important to understand the documentation of Python which is a good starting point. The official website is www.python.org. You can start coding with many options available these days. You can use any of the following IDEs to start with Python coding:
- Google Colab: https://colab.research.google.com/
- Spyder: https://www.spyder-ide.org/
- Pycharm: https://www.jetbrains.com/pycharm/
- Jupyter notebook: https://jupyter.org/
- Anaconda: https://www.anaconda.com/products/distribution
- From the above list, I would suggest you download anaconda in your system. It includes Jupyter notebook, Jupyter Lab, Spyder, R studio etc. all in the same setup.

ANACONDA.

Products Pricing Solutions Resources Partners Blog Company

Contact Sales

Individual Edition is now

ANACONDA DISTRIBUTION

The world's most popular open-source Python distribution platform

Anaconda Distribution

Download

For Windows

Python 3.9 • 64-Bit Graphical Installer • 594 MB

Get Additional Installers

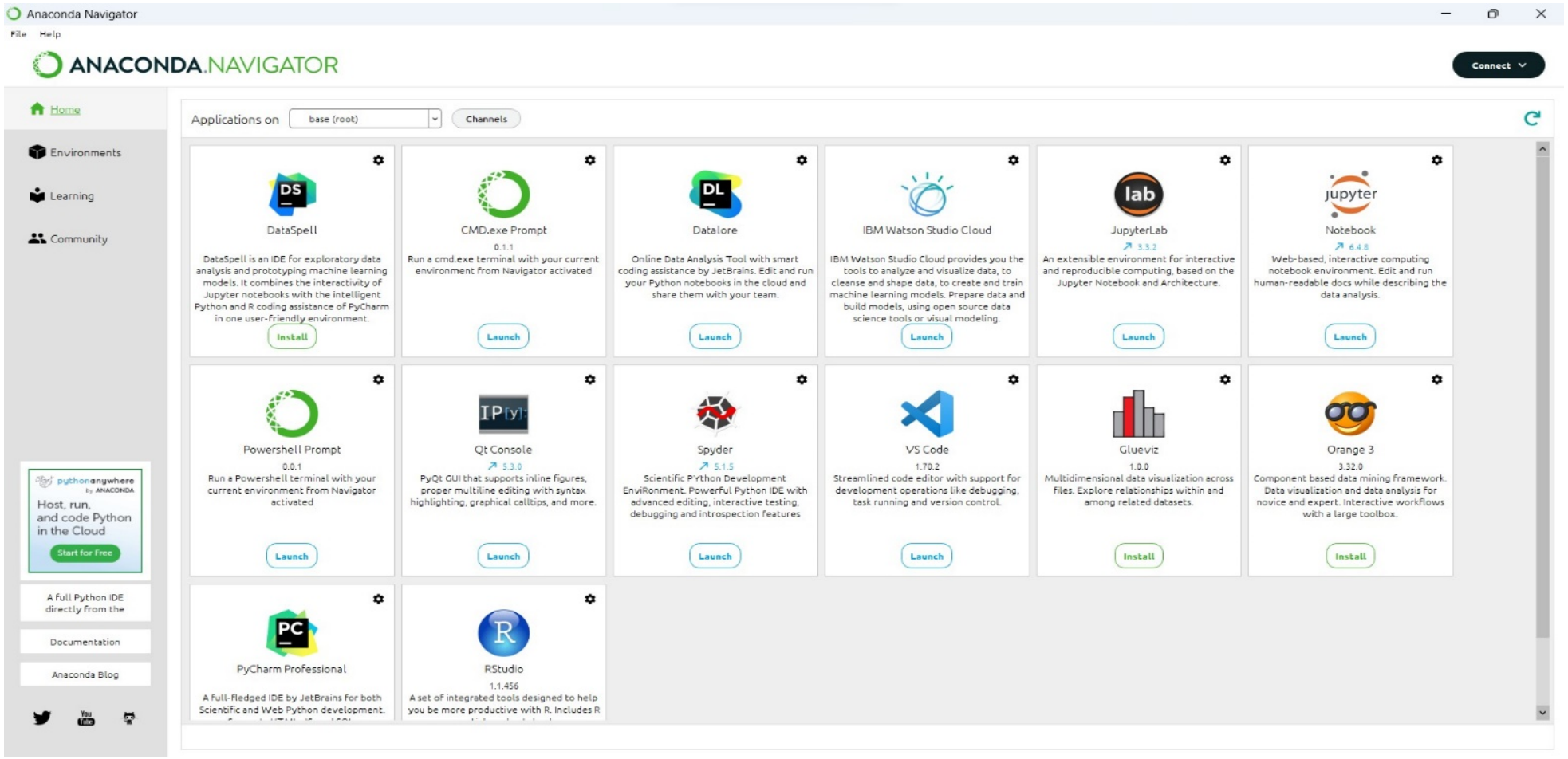
Open Source

User-friendly

Trusted

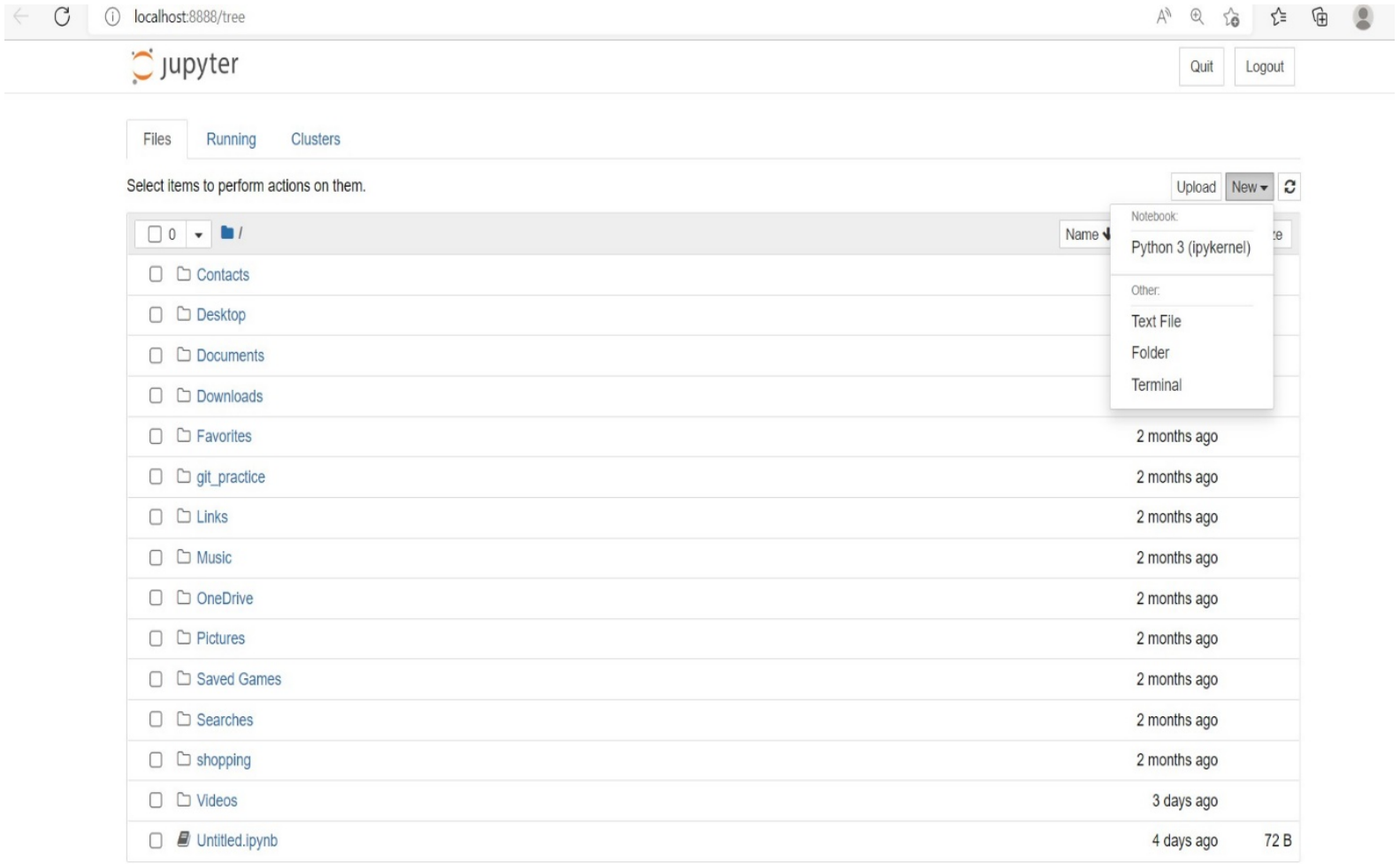
Features of Anaconda

Anaconda Navigator

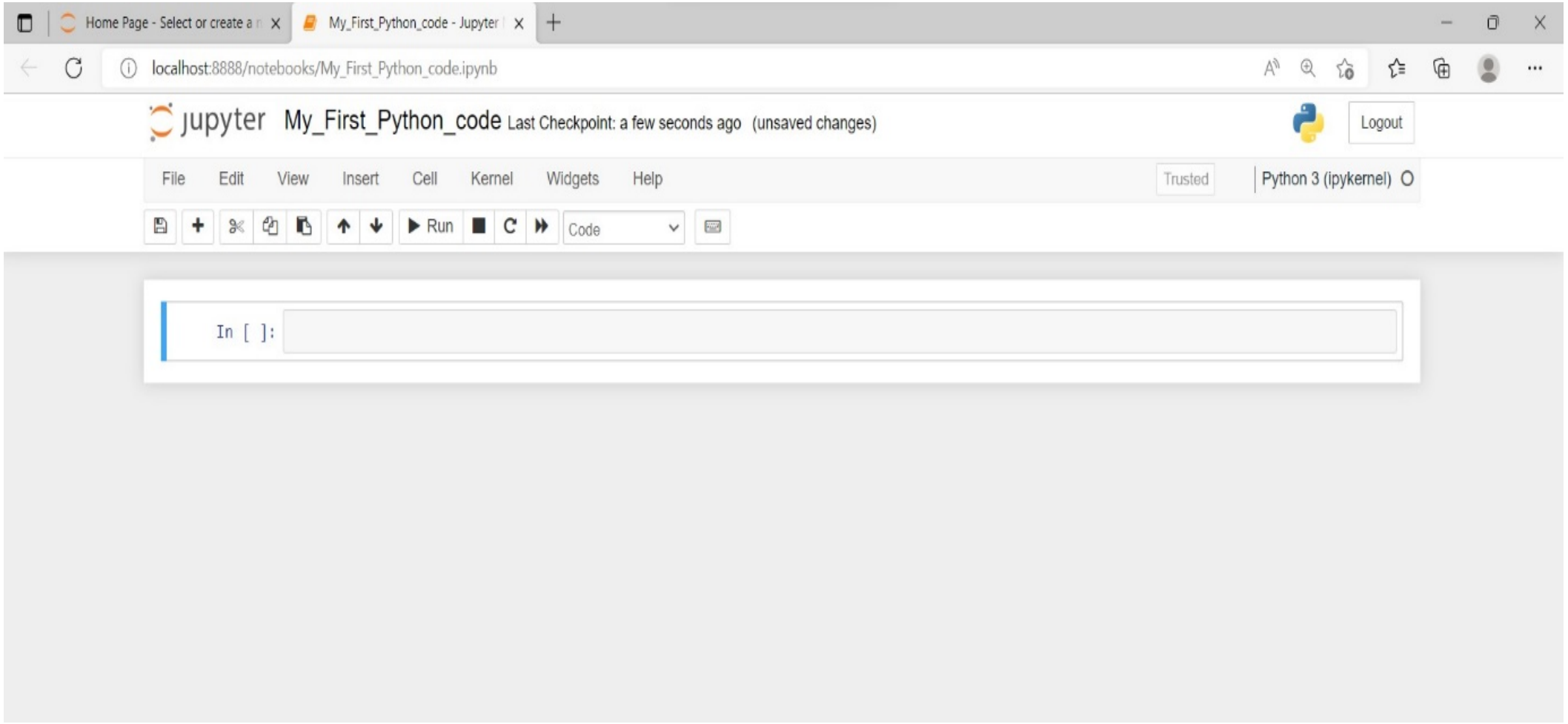


Jupyter Notebook

Once you launch the Jupyter notebook from anaconda, you will be able to see your system folders. If you have any code saved there, you can simply open it up by navigating to the code location. Else you can start by making a new Python 3 notebook as shown.



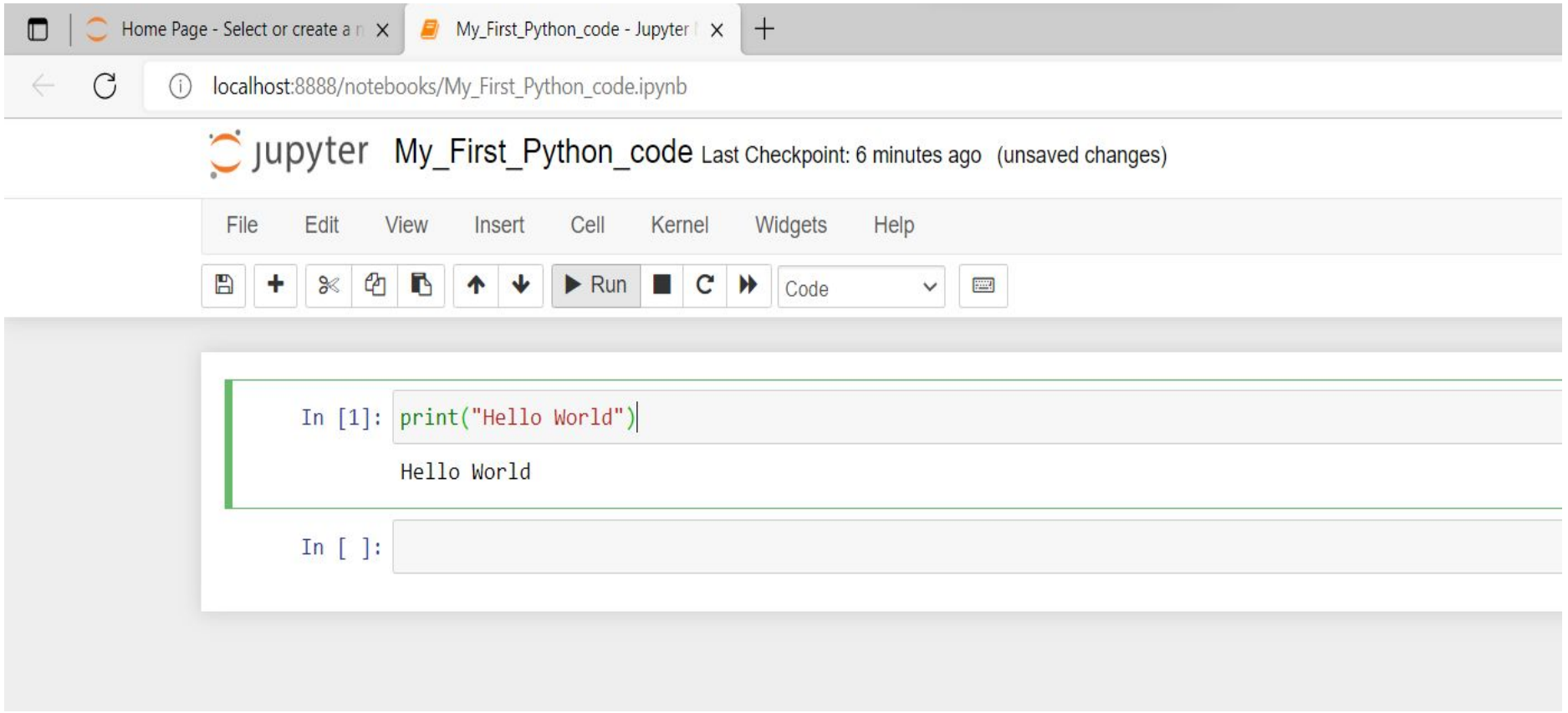
Create Python3 notebook:
Looks like a line by line interpreter



Writing “Hello World”

- Before we start writing our code, we must know that Python is case-sensitive and the use of spaces and tabs in a proper way is really important.
- The next step is to write your first hello world program. You simply have to use the print() function and write in double quotes the statement you want to print. In our case, it is simply print (“Hello World”). Then you have to run the code by clicking on “Run” button Figure 7. The “+” button is used to insert new cell.

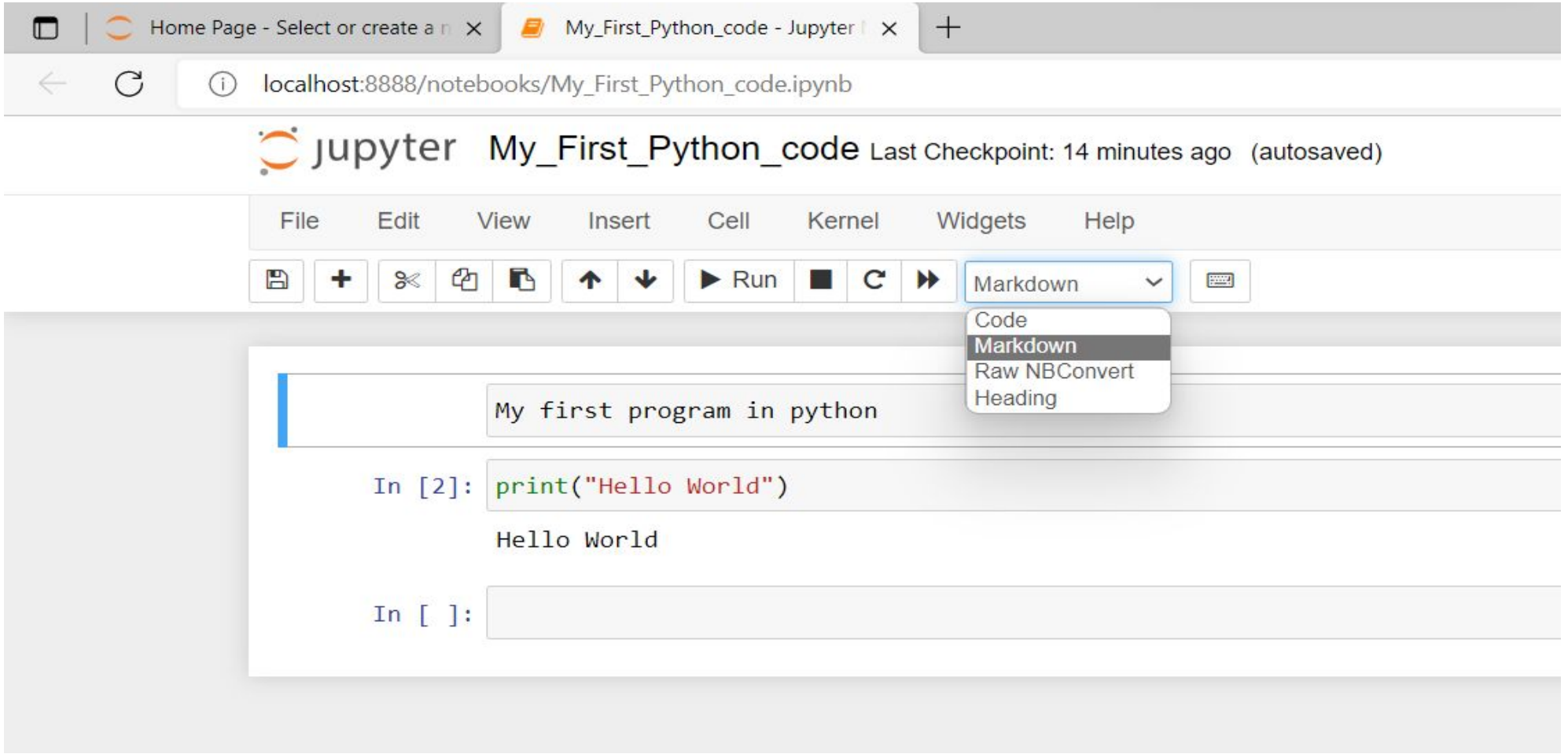
Writing “Hello World”



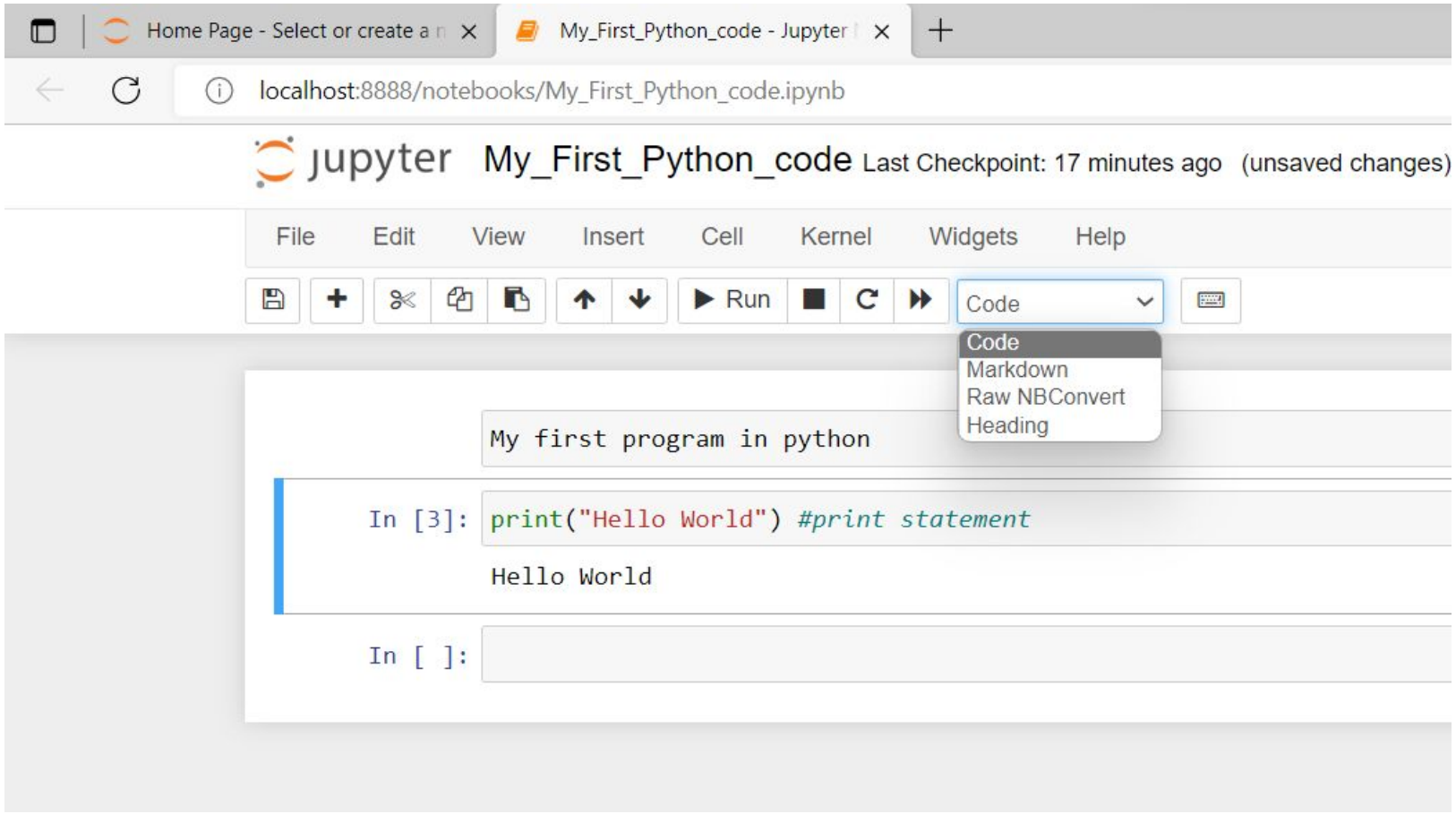
Comments and Headings

- If you want to write any comment, you can simply write your comment and just run it as markdown (not as code)
- or you can use # followed by comment within the code.
- If you use double ## followed by statement and run it as markdown or simply write the statement and run it as heading, it will show you comment as heading.

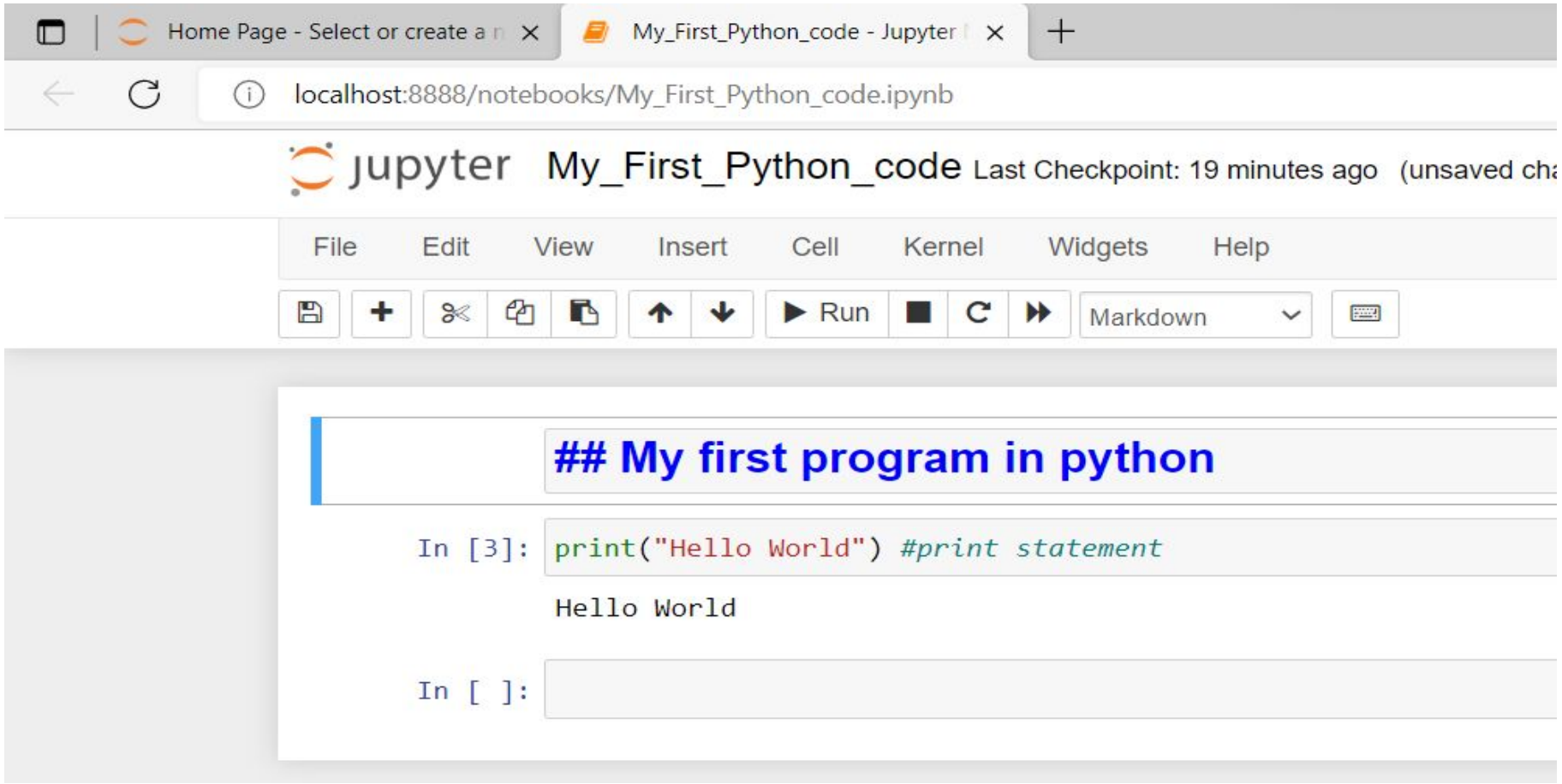
Comments and Headings



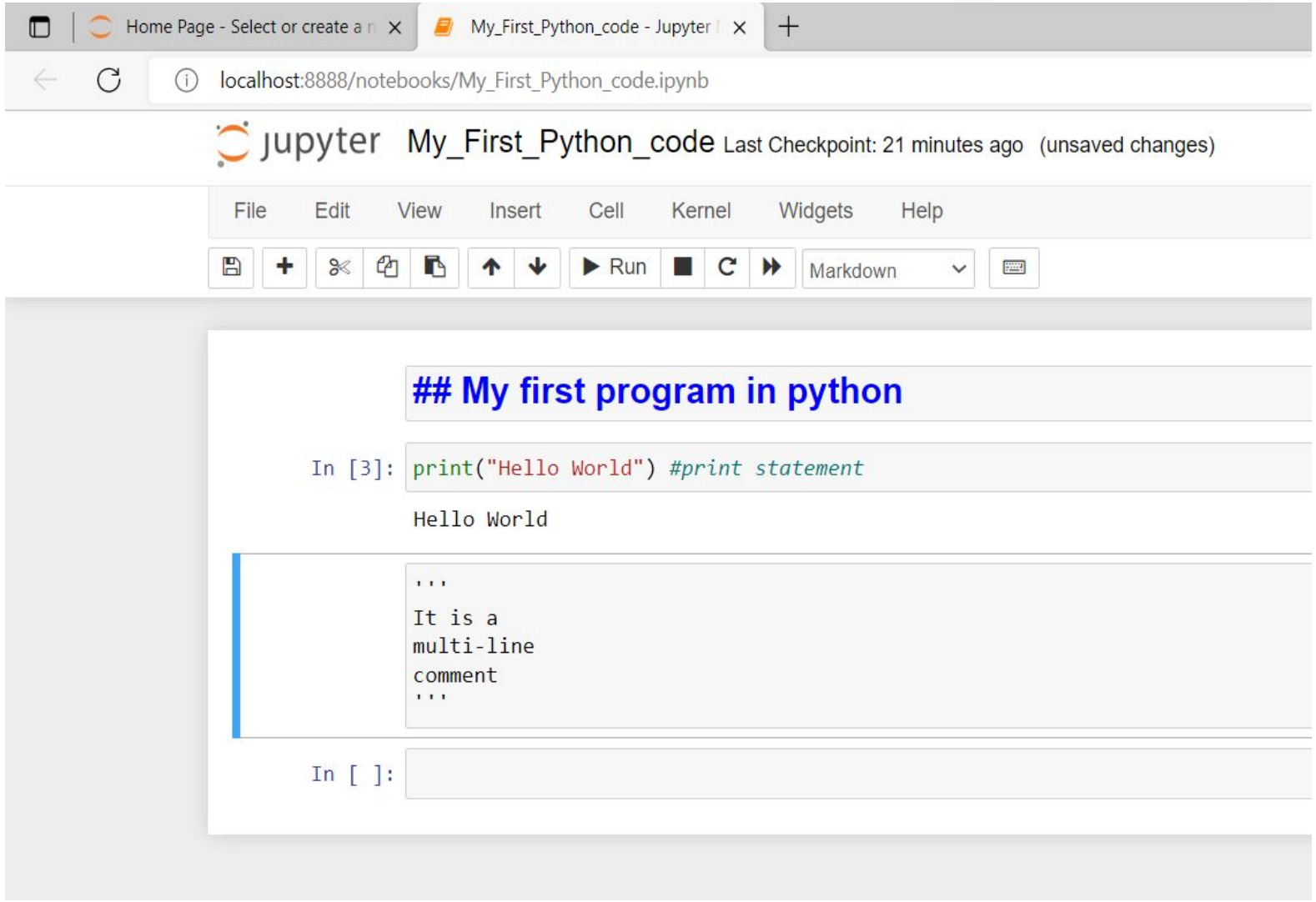
Comments and Headings



Comments and Headings



Comments and Headings



Python as a calculator

- For numbers:
- It is worth noting here that the number should be type compatible otherwise it will throw error. For e.g., if you divide two numbers, it will be a float value which can't be used with int operands for performing calculations. You will have to convert it to compatible type first.

Python as a calculator

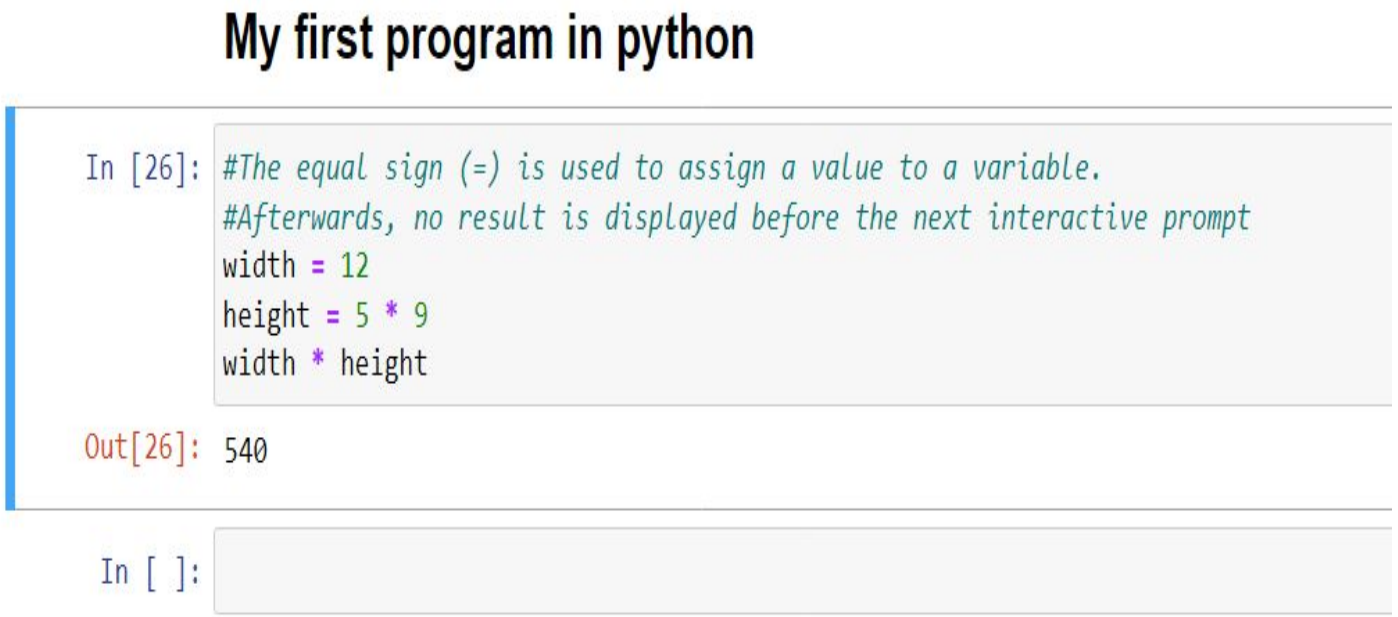
- The Figure show the difference between simple division which operates with single slash and returns a floating point number. The floor division using double slash discards the decimal points in the result of division.

My first program in python ¶

```
In [20]: 5 ** 2  # 5 square
Out[20]: 25

In [21]: 17 / 3  # classic division returns a float
Out[21]: 5.666666666666667

In [22]: 17 // 3  # floor division discards the fractional part
Out[22]: 5

In [23]: 17 % 3  # the % operator returns the remainder of the division
Out[23]: 2

In [24]: 5 * 3 + 2  # floored quotient * divisor + remainder
Out[24]: 17

In [25]: #With Python, it is possible to use the ** operator to calculate powers:
         5 ** 2   # 5 squared
Out[25]: 25
```

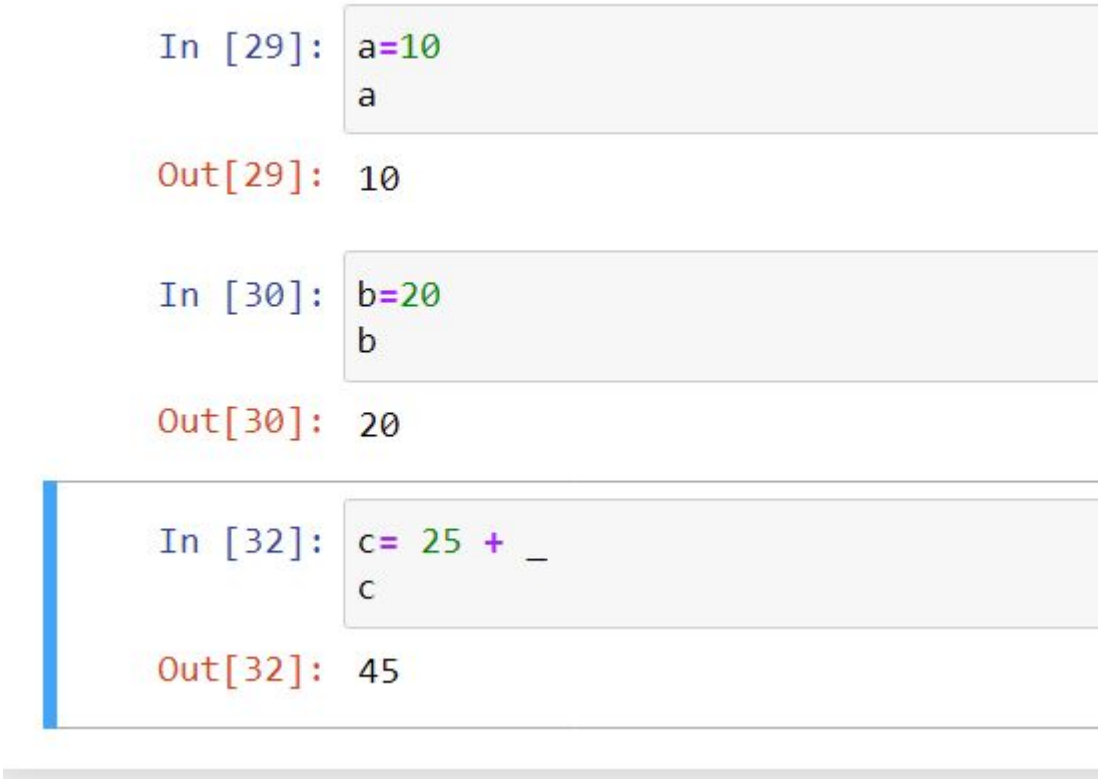
Python as a calculator

The use of assigning any value to a variable that can be used later on.



Use of underscore

If a variable is being used and it is not defined earlier, it will simply throw an error. There is an option to use the last printed value in python as it becomes so interactive. The last printed value is stored in an underscore variable Figure 15 and can be used further.



Python as a calculator

- For strings:
- Strings can be stored in either single quotes or double quotes and a forward slash “\” can be used to escape quotes. Figure shows some use cases for strings. For splitting the strings into multiple lines, the new line operator “\n” can be used. The result varies if you don’t use the print function than if you use the print.

```
In [33]: 'This is my first string'   #single quotes
Out[33]: 'This is my first string'

In [34]: "This is my first string"   #double quotes
Out[34]: 'This is my first string'

In [35]: 'This isn\'t my first string'   #use \ to escape single slash
Out[35]: "This isn't my first string"

In [36]: "This isn't my first string"     #use double quotes
Out[36]: "This isn't my first string"

In [38]: "Is it your first string? \"Yes\", it is."
Out[38]: 'Is it your first string? "Yes", it is.'
```

```
In [39]: 'This is my first string \n This is my second string'
Out[39]: 'This is my first string \n This is my second string'

In [41]: print('This is my first string \nThis is my second string')
         This is my first string
         This is my second string
```

Python as a calculator

- Strings can be concatenated with “+” operator and repeated with “*”. Two or more strings literals which are next to each other are automatically concatenated. This is useful in breaking down long strings. But it can only be used with literals not with variables. For that we will have to use “+” as shown in Figure 18

```
In [42]: # 3 times 'ab', followed by 'gcd'
         3 * 'ab' + 'gcd'

Out[42]: 'abababgcd'

In [44]: 'In' 'dia'

Out[44]: 'India'

In [45]: pre = 'In'
         pre 'dia'

           Input In [45]
             pre 'dia'
                 ^
         SyntaxError: invalid syntax

In [46]: pre = 'In'
         pre + 'dia'

Out[46]: 'India'
```

Python as a calculator

- String can be indexed staring from 0. Indices may be negative numbers also to start counting from the right. Since, -0 is same as 0, the negative indexing starts from -1 Figure 19. In addition to indexing, slicing is also supported. Indexing is used to fetch the individual character at the given index while slicing is used to find the substrings. Slice index have default values too. The omitted first index results in starting from 0 and omitted second index results in default to size of string Figure 21.

```
In [47]: word = 'MachineLearning'

In [48]: word[0] # character in position 0
Out[48]: 'M'

In [49]: word[6] # character in position 6
Out[49]: 'e'

In [50]: word[-1] # last character
Out[50]: 'g'

In [51]: word[-2] # second last character
Out[51]: 'n'

In [52]: word[0:2]  # characters from position 0 (included) to 2 (excluded)
Out[52]: 'Ma'

In [53]: word[2:5]  # characters from position 2 (included) to 5 (excluded)
Out[53]: 'chi'
```

Python as a calculator

- Slicing

```
In [54]: word[:2] # character from the beginning to position 2 (excluded)
Out[54]: 'Ma'

In [55]: word[4:]   # characters from position 4 (included) to the end
Out[55]: 'ineLearning'

In [56]: word[-2:]  # characters from the second-last (included) to the end
Out[56]: 'ng'

In [57]: word[:2] + word[2:]
Out[57]: 'MachineLearning'

In [58]: word[:4] + word[4:]
Out[58]: 'MachineLearning'

In [59]: word[42] # the word only has 15 characters
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [59], in <cell line: 1>()
----> 1 word[42]

IndexError: string index out of range

In [60]: word[4:42]
Out[60]: 'ineLearning'

In [61]: word[42:]
Out[61]: ''
```

Python as a calculator

- Changing String

Python strings cannot be changed — they are immutable. Therefore, assigning to an indexed position in the string results in an error:

```
In [62]: word[0]='J'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [62], in <cell line: 1>()
----> 1 word[0]='J'

TypeError: 'str' object does not support item assignment
```

```
In [63]: 'J'+ word[1:]
```

```
Out[63]: 'JachineLearning'
```

```
In [64]: #The built-in function len() returns the length of a string
         len(word)
```

```
Out[64]: 15
```

LOOPS

Loops are among the most basic and powerful concepts of programming. A loop in a computer program is an instruction that repeats until a specified condition is reached.

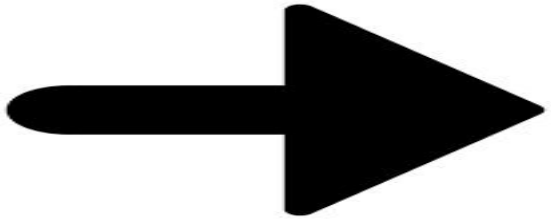
- In a loop structure, the loop asks a question. If the answer requires action, it is executed.
- Why to use loop?

  Reason:A computer programmer who needs to use the same lines of code many times in a program can use a loop to save time.

While loop

- A while loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

```
# Python program to illustrate
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```



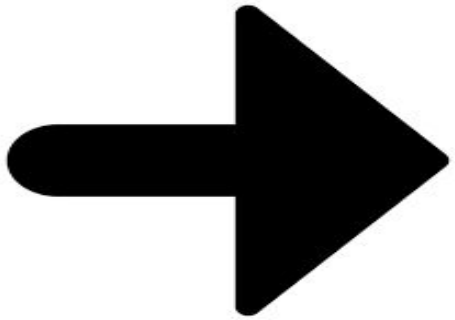
Output:

```
Hello Geek
Hello Geek
Hello Geek
```

For loop

- For Loops are used for sequential traversal.
- For example: traversing a list or string or array etc.
- In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to for each loop in other languages.

```
# Python program to illustrate
# Iterating over range 0 to n-1

n = 4
for i in range(0, n):
    print(i)
```



Output :

```
0
1
2
3
```

For loop example:

```
# A simple for loop example

fruits = ["apple", "orange", "kiwi"]

for fruit in fruits:

 print(fruit)
```

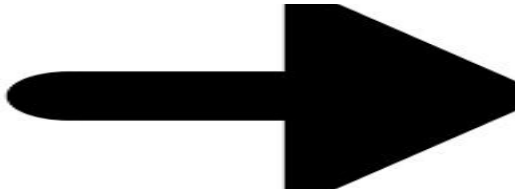
Output

```
apple
orange
kiwi
```

Nested loops

- Python programming language allows to use one loop inside another loop.

```
# Python program to illustrate
# nested for loops in Python
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

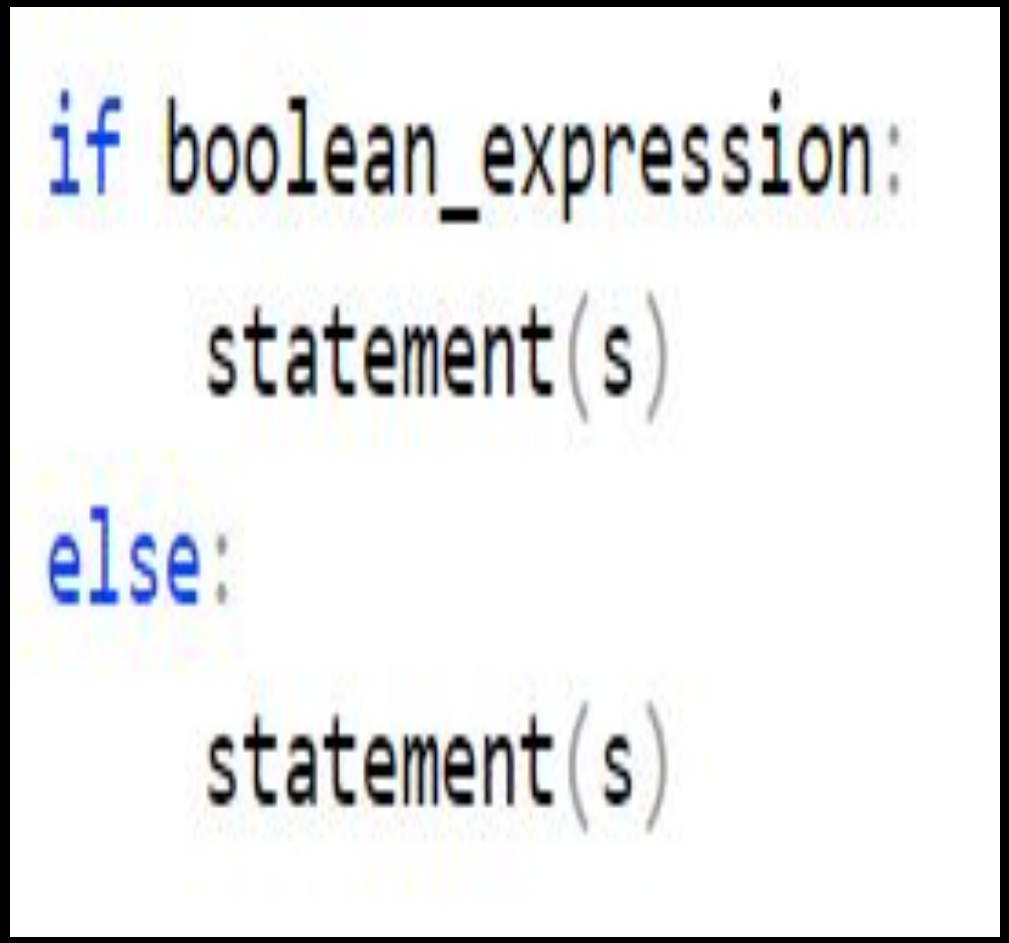


Output:

```
1
2 2
3 3 3
4 4 4 4
```

If else statement

Python If-Else is an extension of Python If statement where we have an else block that executes when the condition is false.

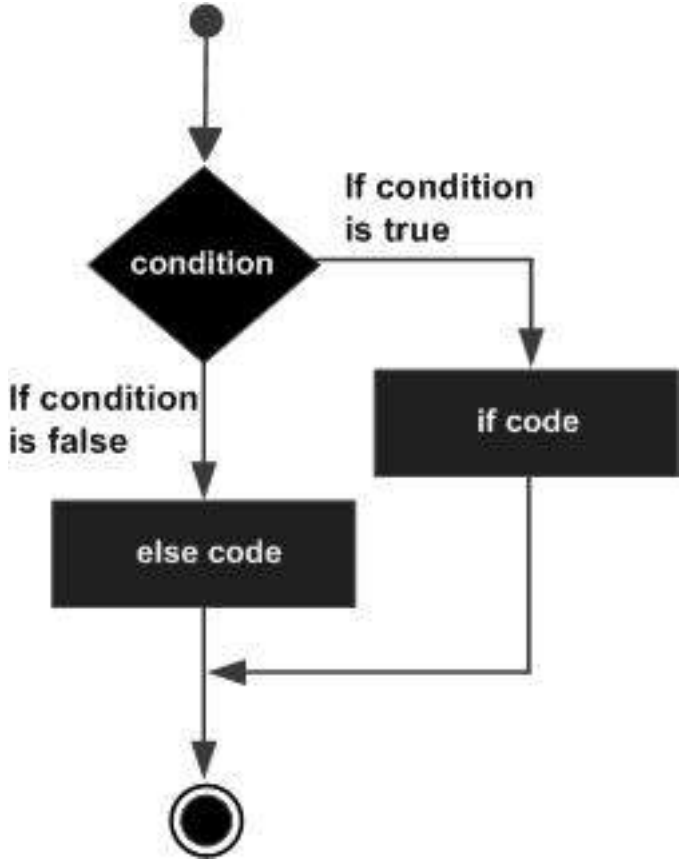


Example

```
a = 2
b = 4

if a<b:
    print(a, 'is less than', b)
else:
    print(a, 'is not less than', b)
```

Output :

```
2 is less than 4
```



Elif statement

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

Example:

```
var = 100
if var == 200:
   print "1 - Got a true expression value"
   print var
elif var == 150:
   print "2 - Got a true expression value"
   print var
elif var == 100:
   print "3 - Got a true expression value"
   print var
else:
   print "4 - Got a false expression value"
   print var

print "Good bye!"
```

```
3 - Got a true expression value
100
Good bye!
```

Lists

1. Multiple items in single variable.
2. Created using square brackets.
3. Items are ordered,changeable and allow duplicates.

Create a List: List is created by placing elements inside square brackets [], separated by commas.

```
[1] my_list = [1, "Hello", 3.4]
    print(my_list)
```

```
[1, 'Hello', 3.4]
```

Delete List Elements We can delete one or more items from a list using the Python del statement. It can even delete the list entirely.

```
[3] my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
    del my_list[2]
    print(my_list)
```

```
['p', 'r', 'b', 'l', 'e', 'm']
```

```
    del my_list[1:5]
    print(my_list)
```

```
['p']
```

Access List Items:

```
[10] my_list = ['p', 'r', 'o', 'b', 'e']

     print(my_list[0])
     print(my_list[2])
     print(my_list[4])
```

```
p
o
e
```

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

```
[11] print(len(my_list))
```

```
5
```

Negative indexing

```
print(my_list[-1])
print(my_list[-5])
```

```
e
p
```

Methods	Descriptions
append()	adds an element to the end of the list
extend()	adds all elements of a list to another list
insert()	inserts an item at the defined index
remove()	removes an item from the list
pop()	returns and removes an element at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the number of items passed as an argument
sort()	sort items in a list in ascending order
reverse()	reverse the order of items in the list
copy()	returns a shallow copy of the list

List Slicing in Python

```
[13] print(my_list[2:4])
     print(my_list[3:])
     print(my_list[:])
```

```
['o', 'b']
['b', 'e']
['p', 'r', 'o', 'b', 'e']
```

Examples

```
[14] # Example on Python list methods

     my_list = [3, 8, 1, 6, 8, 8, 4]

     # Add 'a' to the end
     my_list.append('a')
     print(my_list)

     # Index of first occurrence of 8
     print(my_list.index(8))

     # Count of 8 in the list
     print(my_list.count(8))
```

```
[3, 8, 1, 6, 8, 8, 4, 'a']
1
3
```

0s

```
#reverse
my_list = [3, 8, 1, 6, 8, 8, 4]
my_list.reverse()
print(my_list)

#insert operation
my_list.insert(3,12)
print(my_list)
```

```
[4, 8, 8, 6, 1, 8, 3]
[4, 8, 8, 12, 6, 1, 8, 3]
```

Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered ,unchangeable and allow duplicate values.
- Tuples are written with round brackets.

Creating Python Tuples : To create a tuple we will use () operators.

```
[19] var = ("Geeks", "for", "Geeks")
     print(var)
```

```
('Geeks', 'for', 'Geeks')
```

Concatenation of Python Tuples :To concatenate the Python tuple we will use plus operators(+).

```
[20]
     tuple1 = (0, 1, 2, 3)
     tuple2 = ('python', 'geek')
     # Concatenating above two
     print(tuple1 + tuple2)
```

```
(0, 1, 2, 3, 'python', 'geek')
```

Accessing Values in Python Tuples

```
#Method 1: Using Positive Index
var = ("Geeks", "for", "Geeks")
print("Value in Var[0] = ", var[0])
print("Value in Var[1] = ", var[1])
```

```
Value in Var[0] =  Geeks
Value in Var[1] =  for
```

```
[22] #Method 2: Using Negative Index
     var = ("Geeks", "for", "Geeks")
     print("Value in Var[-3] = ", var[-3])
     print("Value in Var[-2] = ", var[-2])
```

```
Value in Var[-3] =  Geeks
Value in Var[-2] =  for
```

Converting list to a Tuple

```
[23] list1 = [0, 1, 2]
     print(tuple(list1))
     print(tuple('python'))
```

```
(0, 1, 2)
('p', 'y', 't', 'h', 'o', 'n')
```

Slicing Python Tuples

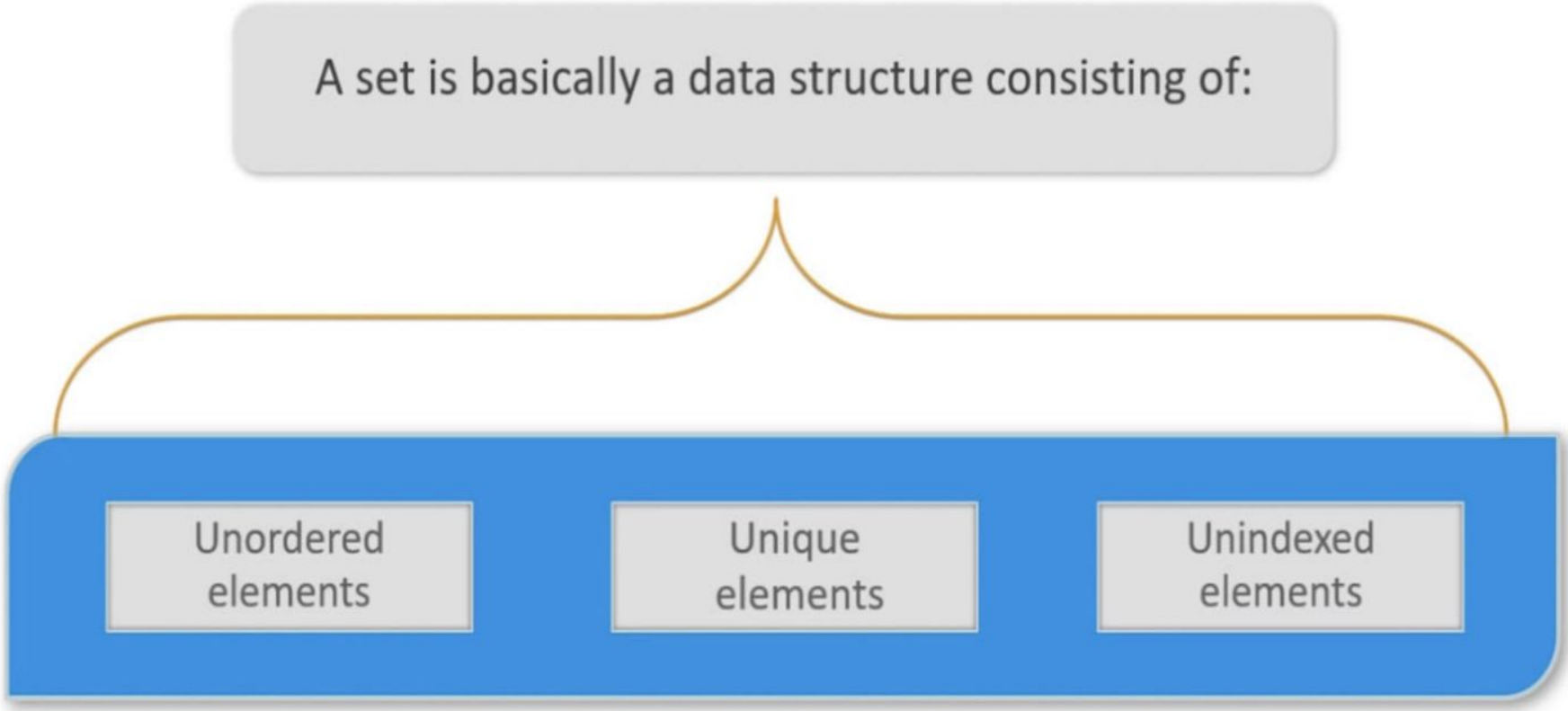
```
# code to test slicing

tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

Slicing Python Tuples

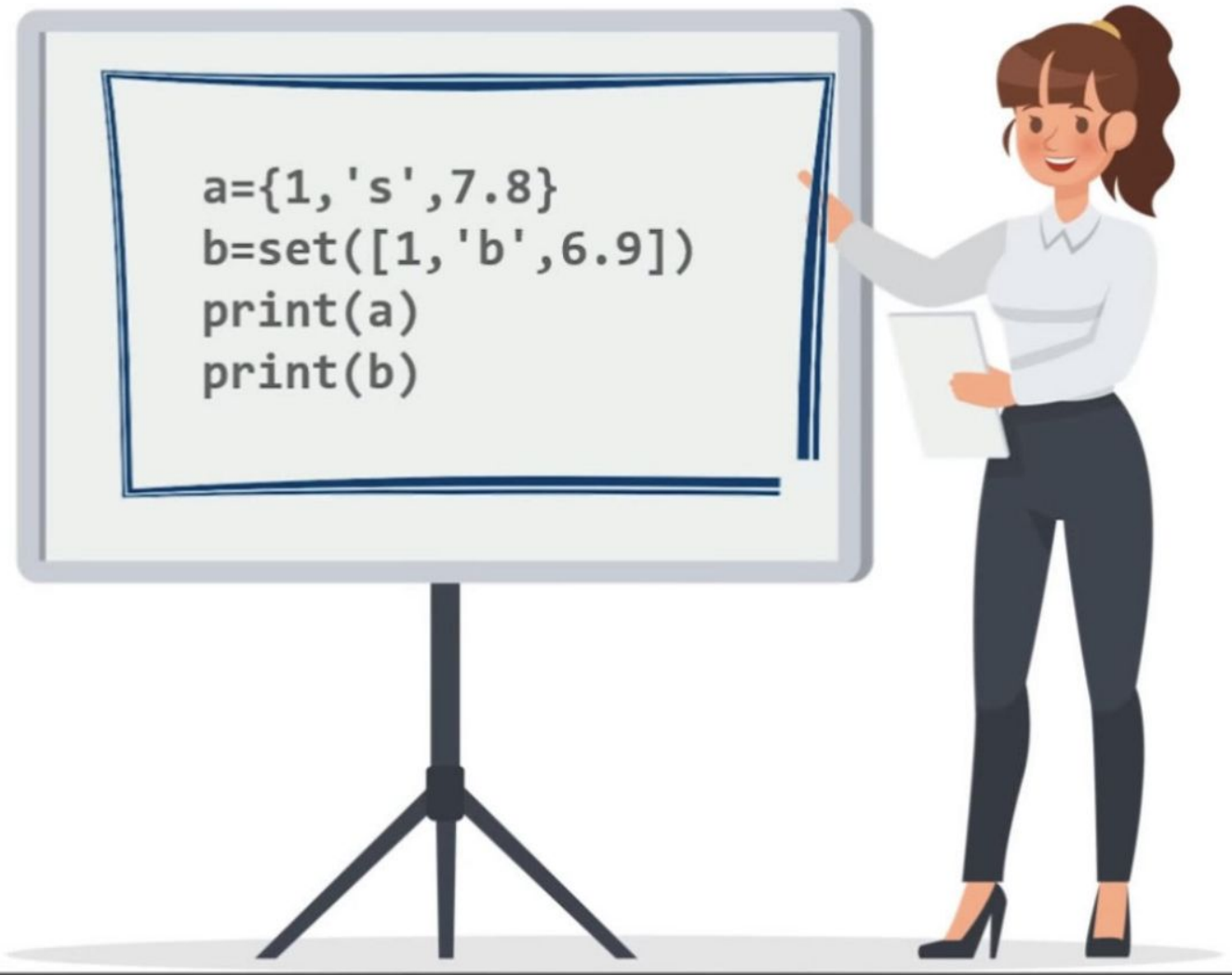
```
# code to test slicing

tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

```
(1, 2, 3)
(3, 2, 1, 0)
(2, 3)
```

What is a Set in Python?



1. CURLY BRACKETS
2. SET CONSTRUCTOR



How to create Sets in Python?

jupyter Sets in Python Last Checkpoint: 3 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

In [1]:
```
a={1,4,2.4,'aaa','abc',9,2}
b=set([1,4,3,'abc',87])
print(a)
print(b)
```

```
{1, 2.4, 2, 4, 9, 'aaa', 'abc'}
{1, 3, 4, 'abc', 87}
```

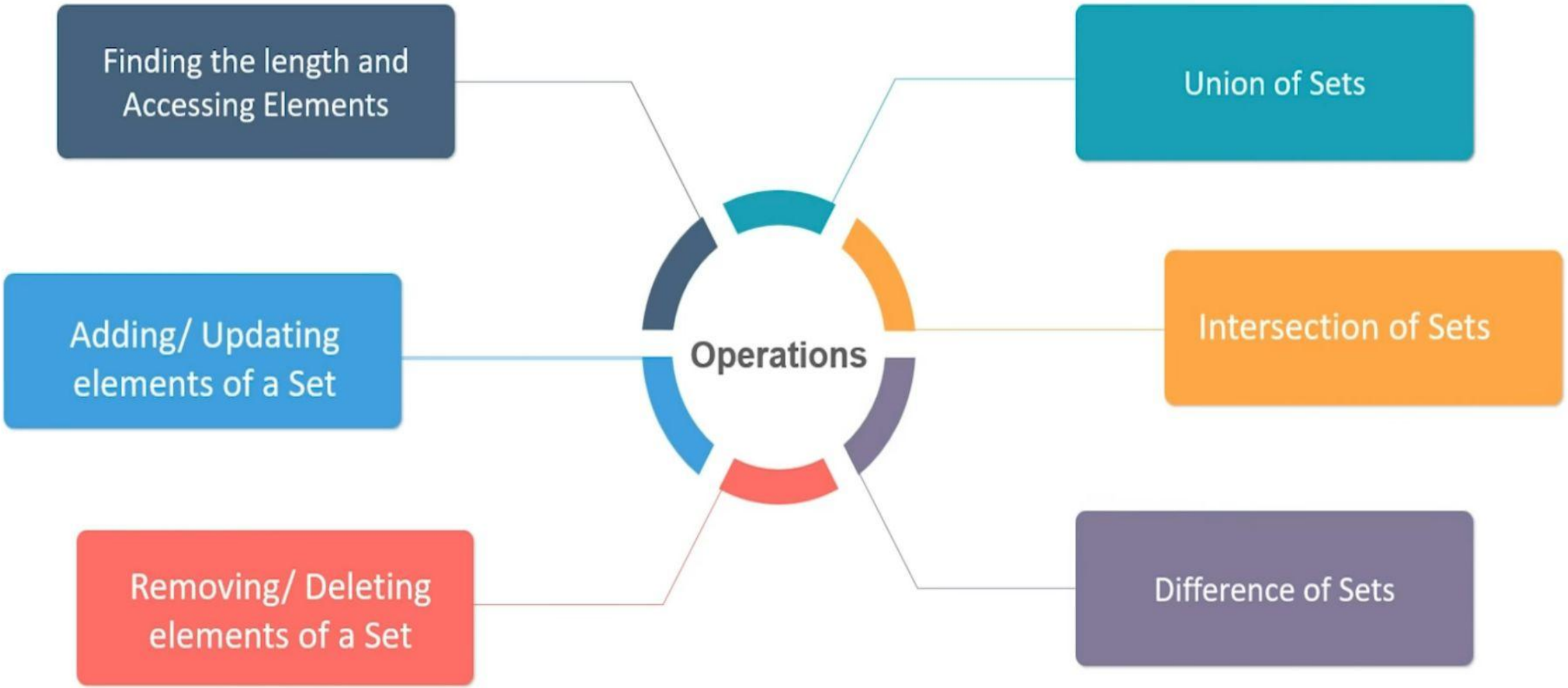
In [2]:
```
c=set()
```

In [3]:
```
print(c)
```

```
set()
```

In [ ]:

Python Set Operations



Finding the length of a Set

- Length of a set is the number of elements that are actually present in that set.
- You can make use of **len()** function to achieve this.

Example-

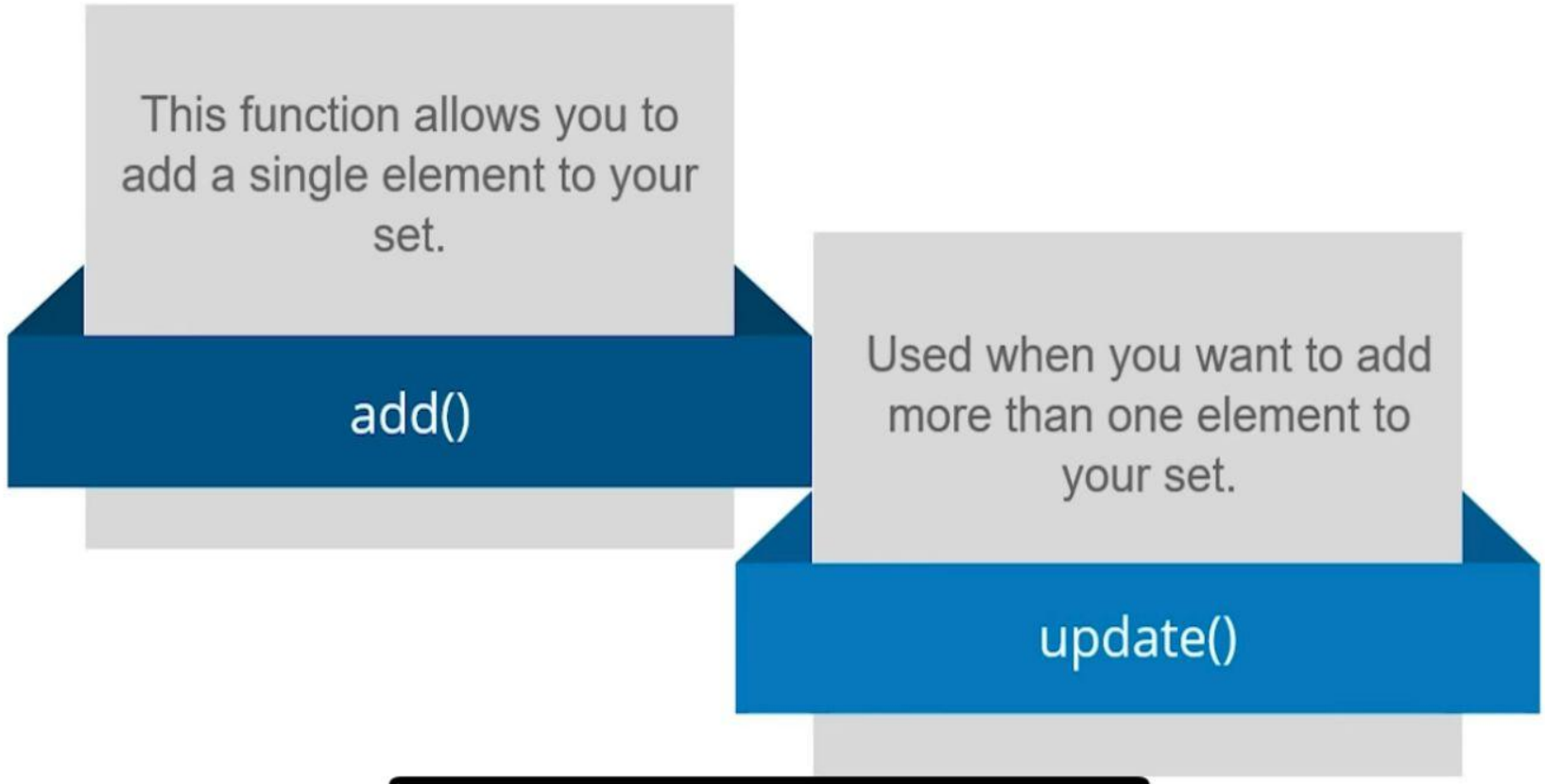
```
My_Set={1,'s',7.8}
len(My_Set)
```

LENGTH OF A SET EXAMPLE

```
In [5]: print(a)
        {1, 2.4, 2, 4, 9, 'aaa', 'abc'}

In [6]: len(a)
Out[6]: 7

In [ ]:
```

Adding Elements to a Set

Elements can be added to a set using these two functions-



In [9]:
```
c.add(3)
c
```

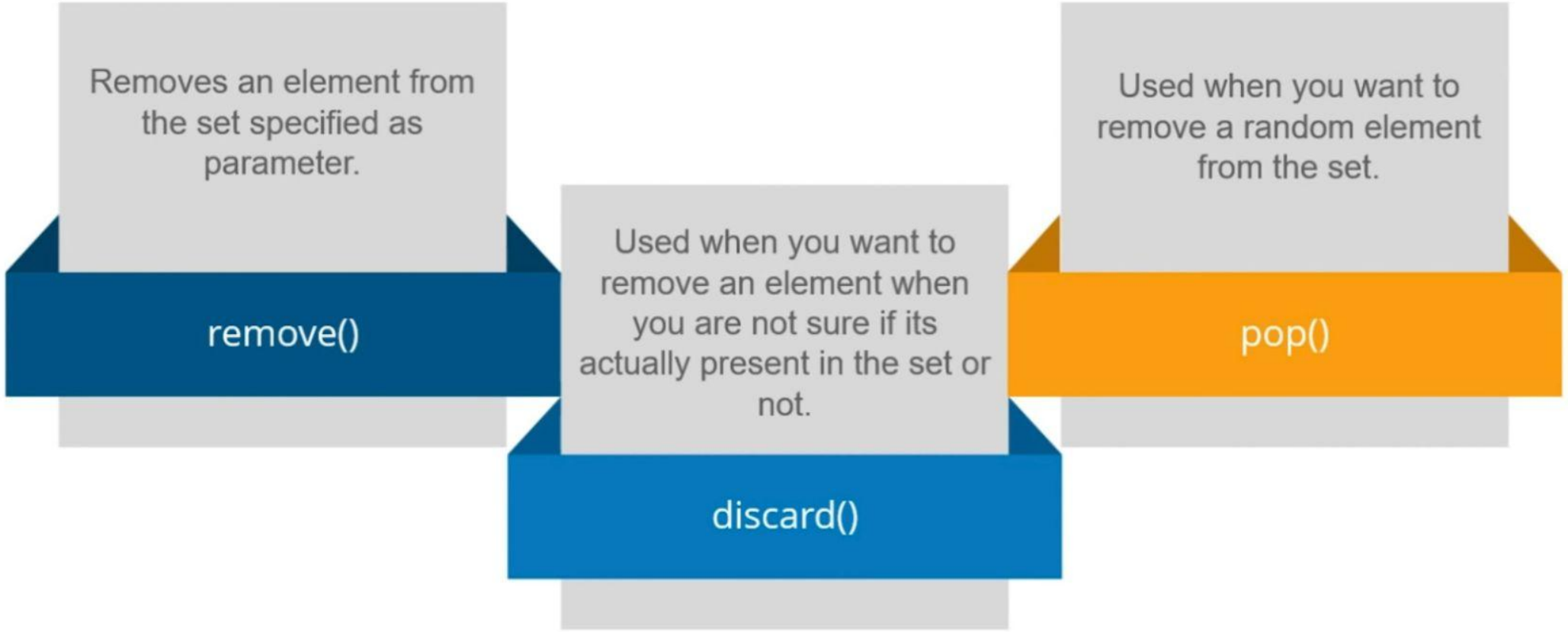
Out[9]: {3}

In [10]:
```
c.update([2,'abc',3])
c
```

Out[10]: {2, 3, 'abc'}

In [ ]:

Elements can be removed from a set using these functions-



Removing Elements from a Set

In [11]:
```
print(a)
print(b)
print(c)
```

```
{1, 2.4, 2, 4, 9, 'aaa', 'abc'}
{1, 3, 4, 'abc', 87}
{2, 3, 'abc'}
```

In [13]:
```
a.remove(2.4)
```

In [14]:
```
a
```

Out[14]:
```
{1, 2, 4, 9, 'abc'}
```

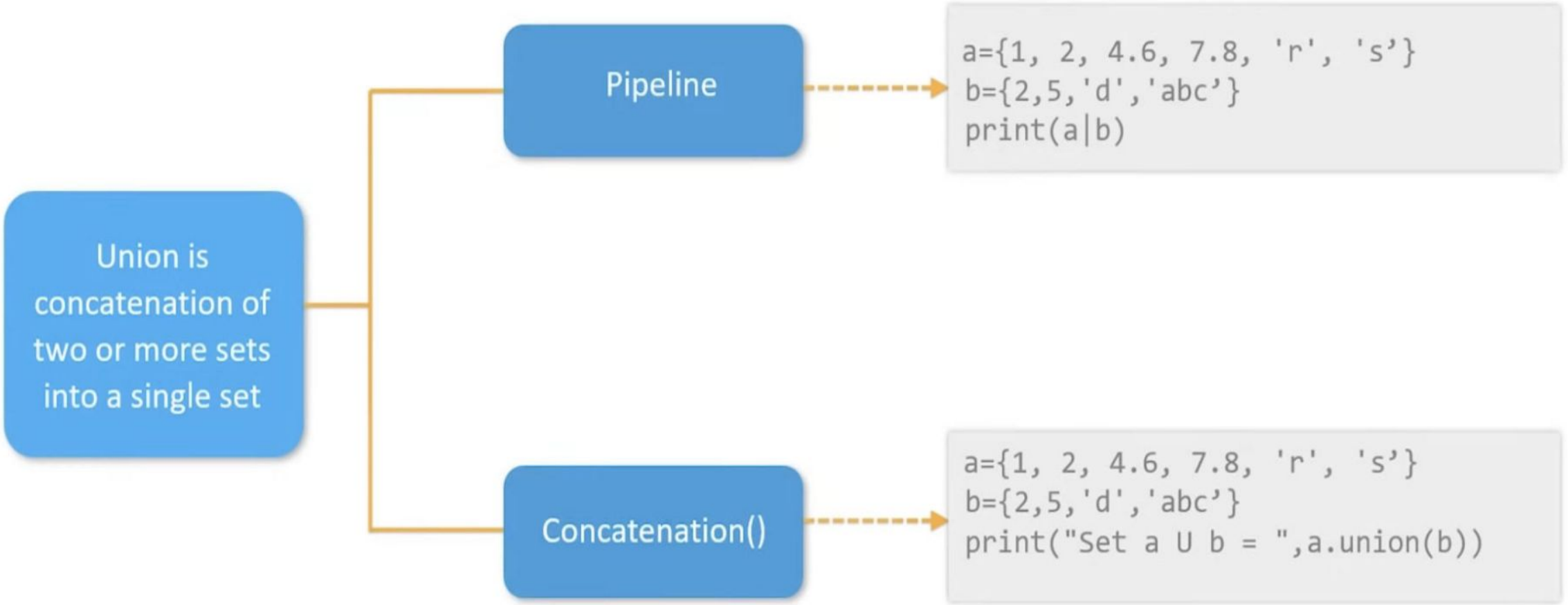
In [15]:
```
a.remove(6)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-15-b5608789f21a> in <module>
----> 1 a.remove(6)

KeyError: 6
```

```
In [16]: print(b)
         {1, 3, 4, 'abc', 87}

In [17]: b.discard(3)

In [18]: b
Out[18]: {1, 4, 87, 'abc'}

In [20]: b.discard(90)

In [ ]:
```

```
In [21]: a.pop()
Out[21]: 1

In [22]: a
Out[22]: {2, 4, 9, 'abc'}
```

Union of Sets



In [23]:
```
print(a)
print(b)
print(c)
```

```
{2, 4, 9, 'abc'}
{1, 4, 'abc', 87}
{2, 3, 'abc'}
```

In [24]:
```
a|b
```

Out[24]:
```
{1, 2, 4, 87, 9, 'abc'}
```

In [25]:
```
a.union(b)
```

Out[25]:
```
{1, 2, 4, 87, 9, 'abc'}
```

In [26]:
```
print(a|b|c)
print(a.union(b,c))
```

```
{1, 2, 3, 4, 9, 'abc', 87}
{1, 2, 3, 4, 9, 'abc', 87}
```

Intersection of two or more sets forms a new set consisting of only the common elements present in those sets.

- Using '&' symbol
- Using intersection()

Example-

```
a={1, 2,5, 4.6, 7.8, 'r', 's'}
b={2,5,'d','abc'}
print(a&b)
print("Set a intersection b =    ",a.intersection(b))
```

Intersection of Sets

```
In [27]: print(a)
         print(b)
         print(c)

         {2, 4, 9, 'abc'}
         {1, 4, 'abc', 87}
         {2, 3, 'abc'}

In [28]: a&b

Out[28]: {4, 'abc'}

In [29]: a.intersection(b,c)

Out[29]: {'abc'}
```

Difference of Sets

The difference of sets produces a new set consisting of elements that are present only in one of those sets.

- ✓ Using '-' symbol
- ✓ Using difference()

Example-

```
a={1, 2,5, 4.6, 7.8, 'r', 's’}
b={2,5,'d','abc’}
Print(a-b)
print("Set a - b = ",a.difference(b))
```

```
In [30]: a
Out[30]: {2, 4, 9, 'abc'}

In [31]: b
Out[31]: {1, 4, 87, 'abc'}

In [32]: a-b
Out[32]: {2, 9}
```

Frozen Sets

- A frozen set in Python is a set whose values cannot be modified
- Frozen sets can be created using the frozenset() method

Example-

```
a={1, 2,5, 4.6, 7.8, 'r', 's’}
b=frozenset(a)
print(b)
```

```
In [38]: d=frozenset(a)
         print(d)

frozenset({9, 2, 'abc', 4})

In [39]: d.add(3)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-39-346ad8e550cb> in <module>
----> 1 d.add(3)

AttributeError: 'frozenset' object has no attribute 'add'
```

HELPS TO SERVE AS KEY IN DICTIONARY

Dictionaries in python

Python Dictionary:

- A dictionary is mutable and is another container type that can store any number of Python objects
- Dictionaries consist of pairs (called items) of keys and their corresponding values.
- Python dictionaries are also known as associative arrays or hash tables.
- The general syntax of a dictionary is as follows:
- dict = {'A': '2341', 'B': '9102', 'C': '3258'}

Python Dictionary Features:

- Each key is separated from its value by a colon (:).
- The items are separated by commas, and the whole thing is enclosed in curly braces.
- An empty dictionary without any items is written with just two curly braces, like this: {}.
- Keys are unique within a dictionary while values may not be
- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary:

- To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.
- For example:
- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

  print "dict['Name']: ", dict['Name'];
- Output:

  dict['Name']: Zara

Updating Dictionary:

•You can update a dictionary by adding a new entry or item (i.e., a key-value pair)
I.e:
•modifying an existing entry, or deleting an existing entry as shown below in the simple example

•dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

```
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry


print "dict['Age']: ", dict['Age'];
print "dict['School']: ", dict['School'];
```

•Output:

```
dict['Age']:  8
```

•dict['School']: DPS School

Delete Dictionary Elements:

You can either remove individual dictionary elements or clear the entire contents of a dictionary

- To explicitly remove an entire dictionary, just use the del statement.

  i.e:

  dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

- del dict['Name']; # remove entry with key 'Name'

  dict.clear(); # remove all entries in dict

  del dict ; # delete entire dictionary

- print "dict['Age']: ", dict['Age'];

- print "dict['School']: ", dict['School'];

output:dict['Age']:

```
Traceback (most recent call last):
 File "test.py", line 8, in <module>
  print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

STRINGS

- A string is a sequence of characters.

P1:

```
[8]  fruit = 'banana'
     letter = fruit[1] # Indices can only be integers
     print(letter)
```

```
a
```

P2:

```
[10] i = 1
     print(fruit[i])
     print(fruit[i+1])
```

```
a
n
```

Concatenation and Repetition

P3:

```
[11] # Concatenation can be done with '+' operator
     print('un' + 'ium')
```

```
unium
```

P4:

```
[12] # Repetition can be done with '*' operator
     print(3 * 'un')
```

```
ununun
```

Sample Program

P5:

```
# Sample Program
prefixes = 'JKLMNOPQ'
suffix = 'ack'

for letter in prefixes:
  print(letter + suffix)
```

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

String Slicing

- A segment of a string is called a slice.
- [n:m] returns the part of the string from the nth character to the mth character, including the first but excluding the last.

P6:

```
[15] s = 'Monty Python'
     print(s[0:5])
     print(s[6:12])

     Monty
     Python
```

String Slicing

P

```
[16] fruit = 'banana'
     print(fruit[:3])
     print(fruit[3:])
```

```
ban
ana
```

```
[21] fruit = 'banana'
     print(fruit[3:3]) # It will give empty string
```

P

```
fruit = 'banana'
print(fruit[:])   # It will return the whole string
```

```
banana
```

Immutable Strings

- You cannot change an existing string.
- String objects does not support the item assignment.

```
greeting = ‘Hello, World!’

greeting[0] = ‘J’ #This will give TypeError
```

Immutable String

- The best way you can do is create a new string that is a variation on the original string.

P10:

```
greeting = 'Hello, World!'
new_greeting = 'J' + greeting[1:]
new_greeting
```

```
'Jello, World!'
```

String Methods

- A method is similar to a function - it takes arguments and returns a value - but the syntax is different.

P11:

```
word = 'banana'
new_word = word.upper() # upper() is a method
print(new_word)
```

```
'BANANA'
```

P12:

```
word = 'banana'
index = word.find('a') # find() is used to find the index
print(index)
```

```
1
```

String Methods

P13:

```
word = 'banana'
index = word.find('na',3)
print(index)
```

```
4
```

P14:

```
# String is not found
name = 'bob'
f = name.find('b', 1, 2)
print(f)
```

```
-1
```

In Operator

- The word **in** is a boolean operator that takes two strings and return True if first appears as a substring in the second.

P15

```
word = 'banana'
'a' in 'banana'
```

```
True
```

P16

```
word = 'banana'
'seed' in 'banana'
```

```
False
```

Question:

* What do we use to define a block of code in Python language?

a. Key
b. Brackets
c. Indentation
d. None of these

Another one →

*Which character is used in Python to make a single line comment?

a. /
b. //
c. #
d. !

*What will be the value of x when code executes successfully

```
#code:
x = 0
a = 5
b = 5
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 2
    print(x)
```

a) 0

a) 2

a) 5

a) 3