Python Programming(2101CS405)

Darshan
UNIVERSITY
योग: कर्मसु कौशलम्

Unit-01

Introduction to Python, Object and Data Structure

Prof. Jayesh D. Vagadiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot

✉ Jayesh.vagadiya@darshan.ac.in
☎ 9537133260

Outline

- ✔ Introduction to python
- ✔ Advantages of python
- ✔ Installing python
- ✔ Hello World program using python
- ✔ Indentations
- ✔ Print() function
- ✔ Input() function
- ✔ Data types
- ✔ Variables
- ✔ Expressions
- ✔ Functions
- ✔ String
- ✔ List
- ✔ Tuple
- ✔ Set
- ✔ Dictionary
- ✔ List Comprehension

Syllabus

Sr. No.	Unit
1	Introduction to Python, Object and Data Structure
2	Python Operators, Conditional and Looping Statements, Functions in Python
3	File IO in Python, Exception handling
4	Modules, Matplotlib
5	Object Oriented Programming with Python

Reference Book:

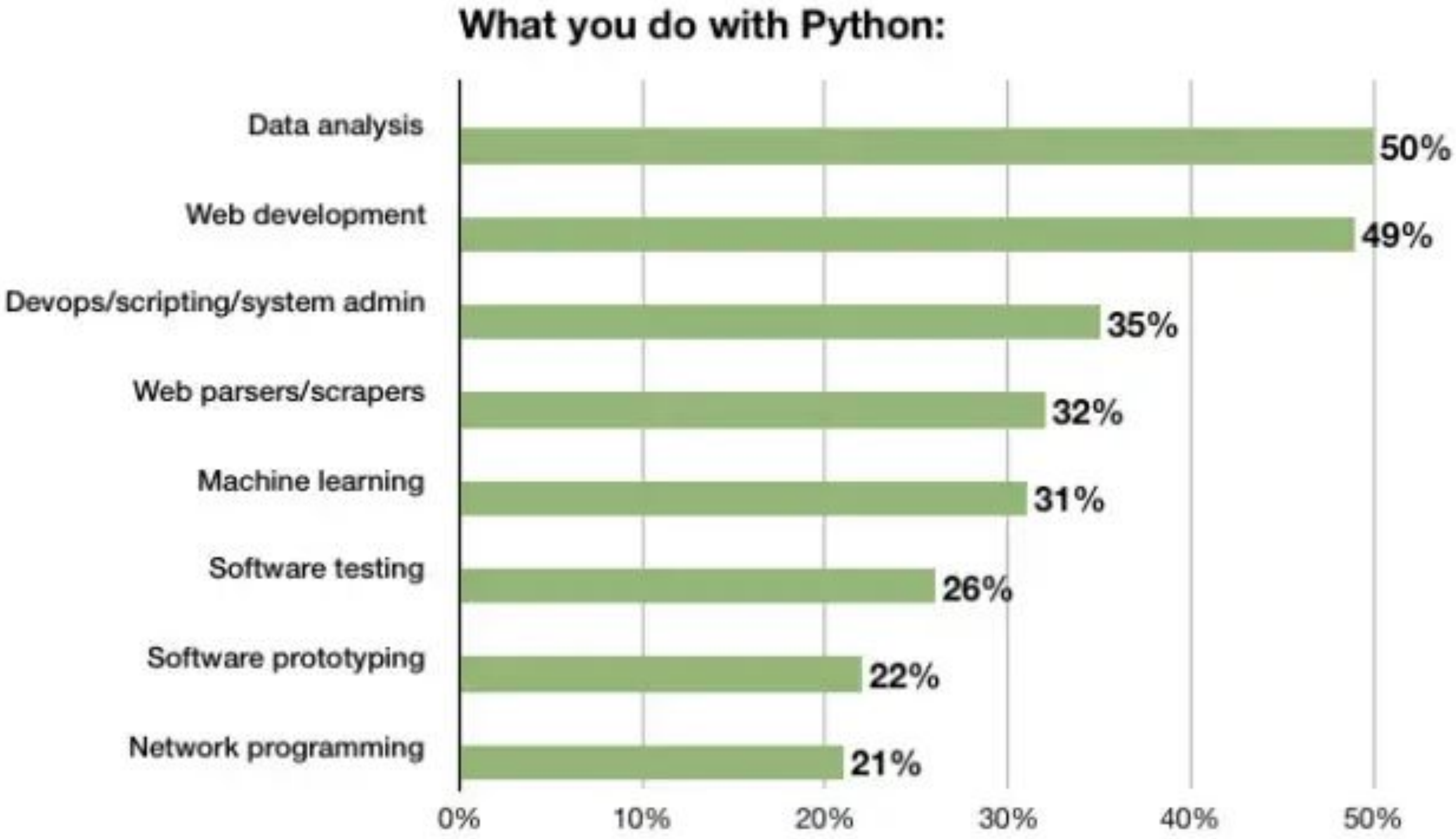
1) Programming in Python 3 : A Complete Introduction to the Python Language
2) Introduction to Computation and Programming Using Python
3) Core Python Programming

Introduction to Python

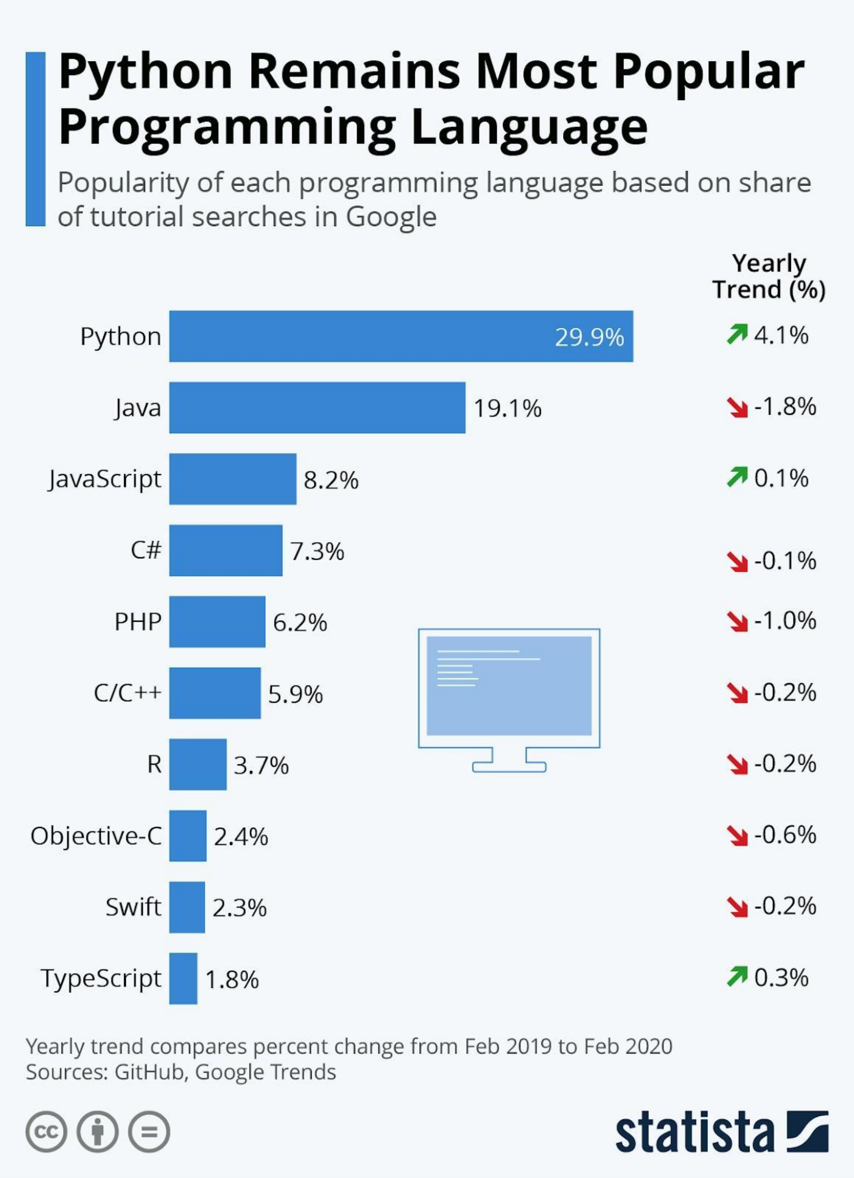
- Python is an **open source, interpreted, high-level, general-purpose** programming language.
- Python's design philosophy emphasizes **code readability** with its notable use of significant **whitespace**.
- Python is **dynamically typed** and **garbage-collected** language.
- Python was conceived in the late **1980s** as a successor to the **ABC language**.
- Python was Created by **Guido van Rossum** and first released in **1991**.

- **Python 2.0**, released in **2000**,
  - introduced features like list comprehensions and a garbage collection system with reference counting.
- **Python 3.0** released in **2008** and current version of python is **3.11.0** (as of Nov 2022).
  - The Python 2 language was officially discontinued in 2020

Why Python ?


What you do with Python:
Data analysis
50%
Web development
49%
Devops/scripting/system admin
35%
Web parsers/scrapers
32%
Machine learning
31%
Software testing
26%
Software prototyping
22%
Network programming
21%
0%
10%
20%
30%
40%
50%

Why Python ?



Python Remains Most Popular Programming Language

Popularity of each programming language based on share of tutorial searches in Google

Yearly trend compares percent change from Feb 2019 to Feb 2020
Sources: GitHub, Google Trends

Why Python?

Python has many advantages

- Easy to Read, Learn and Write
  - Python is a **high-level programming language** that has English-like syntax. This makes it easier to read and understand the code.
- Improved Productivity
  - Python is a very **productive language**. Due to the simplicity of Python, developers can focus on solving the problem.
- Interpreted Language
  - Python is an interpreted language which means that Python directly **executes the code** line by line. In case of any error, it stops further execution and reports back the error which has occurred.
- Dynamically Typed
  - python automatically assigns the data type during **execution**. The programmer doesn’t need to worry about declaring variables and their data types.

Why Python? (cont.)

- Free and Open-Source
  - Python comes under the OSI approved open-source license. This makes it free to use and distribute.
  - The Python Software Foundation distributes pre-made binaries that are freely available for use on all major operating systems called CPython.
  - You can get CPython's source-code, too. Plus, you can modify the source code and distribute it as allowed by CPython's license.
- Vast Libraries Support
  - The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don't have to depend on external libraries.
- Portability
  - In many languages like C/C++, you need to change your **code** to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere.
  - Python runs on all major operating systems like Microsoft Windows, Linux, and Mac OS X.

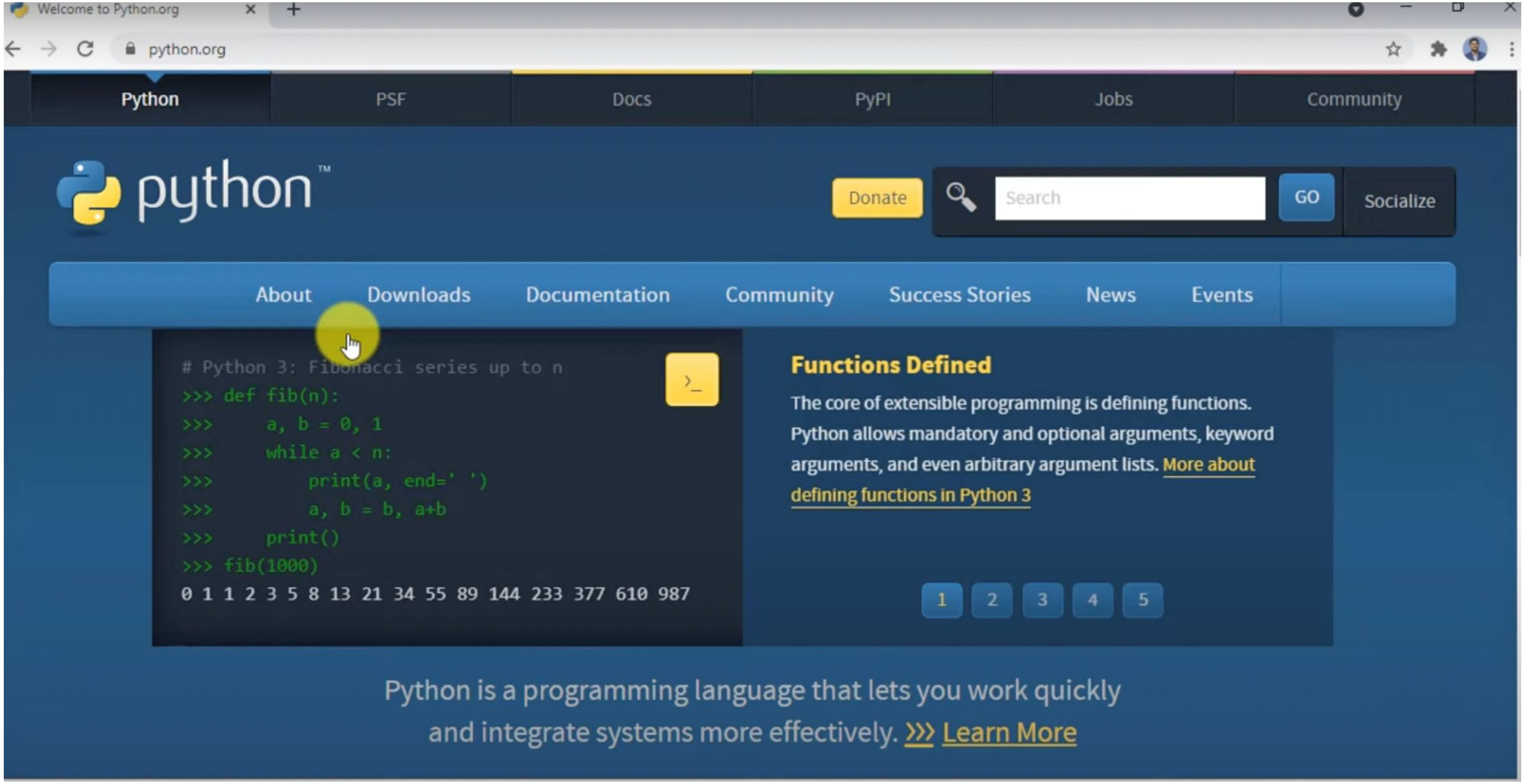
Why Python? (cont.)

- It's Safe
  - Python doesn't have pointers like other C-based languages, making it much more reliable.
  - Along with that, errors never pass silently unless they're explicitly silenced.
  - This allows you to see and read why the program crashed and where to correct your error.
- High-Level Language
  - Python looks more like a readable, human language than like a low-level language.
  - This gives you the ability to program at a faster rate.

Installing Python

- For Windows & Mac:
  - To install python in windows you need to download installable file from https://www.python.org/downloads/
  - After downloading the installable file you need to execute the file.
- For Linux :
  - For ubuntu 16.10 or newer
    - sudo apt-get update
    - sudo apt-get install python3.8
- To verify the installation
  - Windows :
    - python --version
  - Linux :
    - **python3** --version (linux might have python2 already installed, you can check python 2 using **python --version**)
- Alternatively we can use anaconda distribution for the python installation
  - http://anaconda.com/downloads
  - Anaconda comes with many useful inbuilt libraries.

Step – 1 open python.org



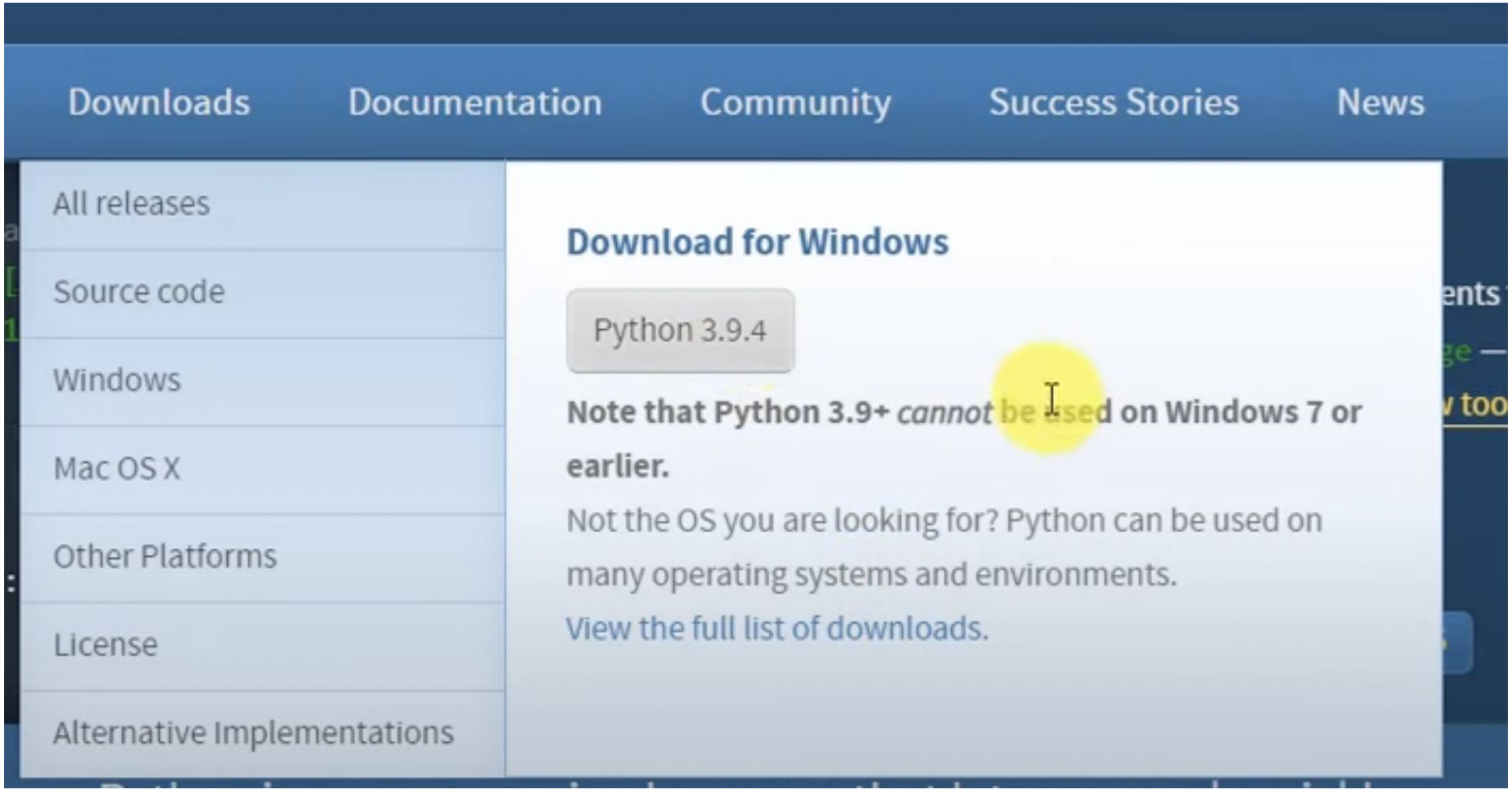
```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Step-2 goto download the python version (exe file)



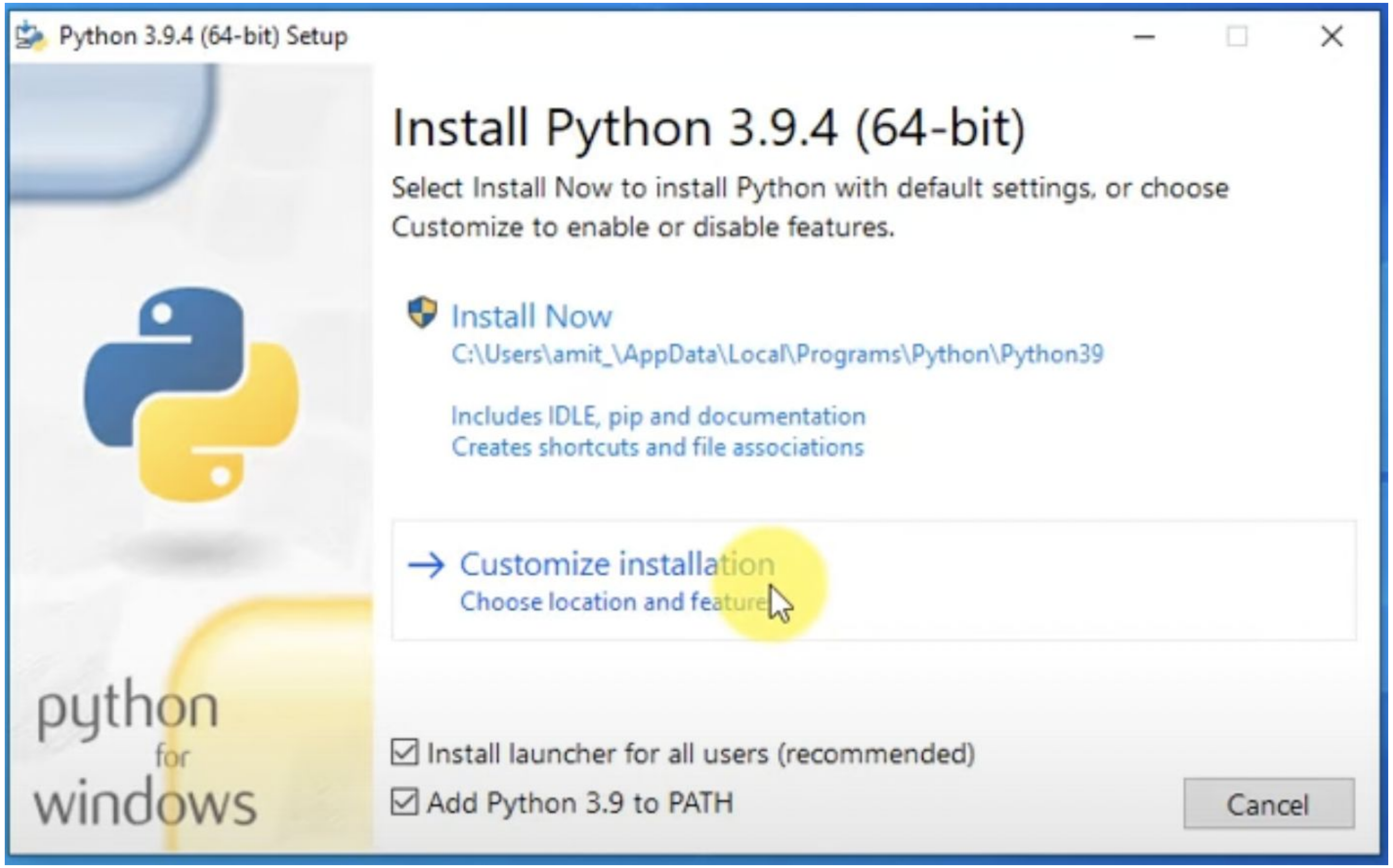
Download for Windows

Note that Python 3.9+ *cannot* be used on Windows 7 or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

Step – 3 run exe and select Customize Installation



Python 3.9.4 (64-bit) Setup

Install Python 3.9.4 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

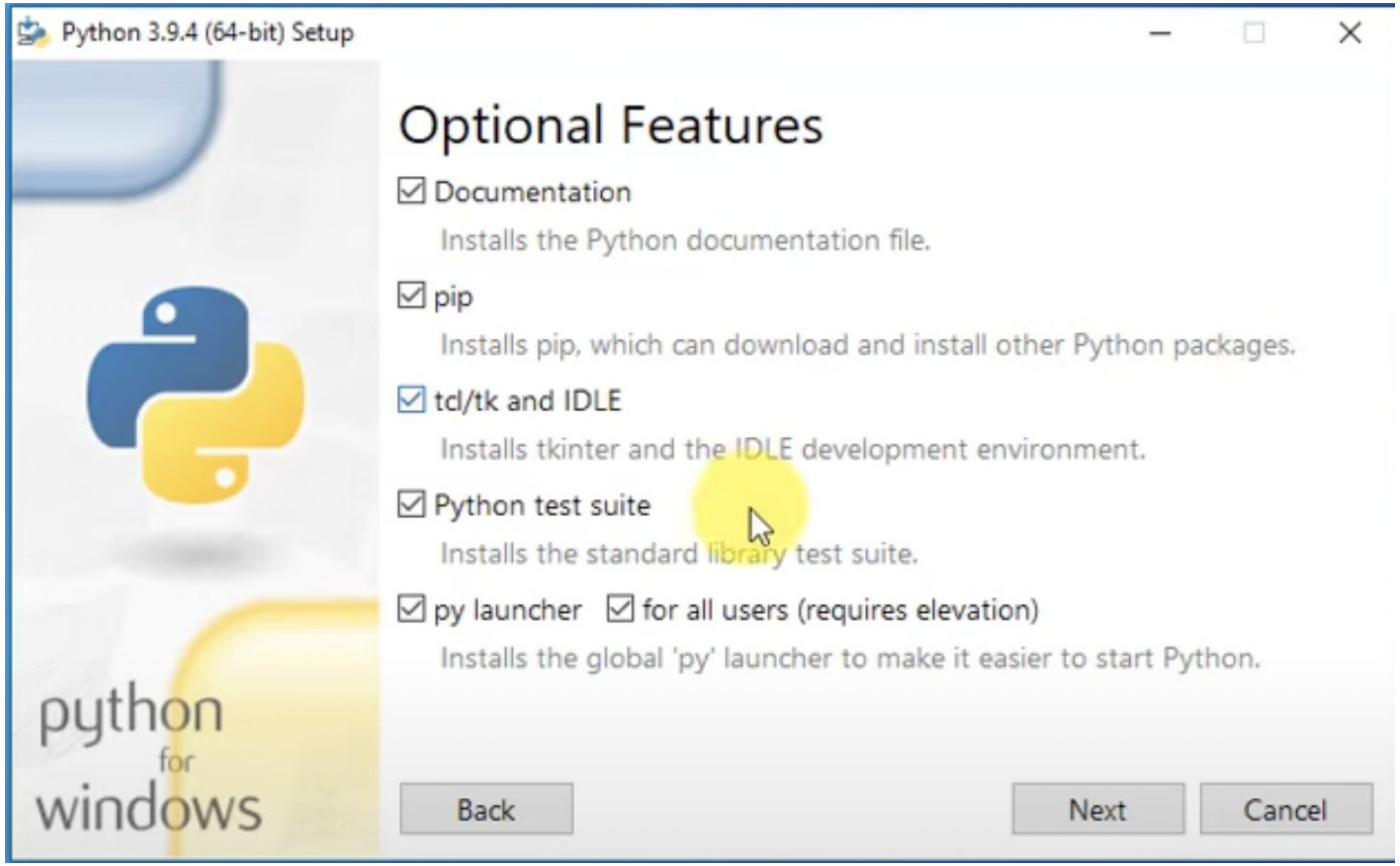
Install Now
C:\Users\amit_\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

Step – 4 select all option features



Python 3.9.4 (64-bit) Setup

Optional Features

☑ Documentation

Installs the Python documentation file.

☑ pip

Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE

Installs tkinter and the IDLE development environment.

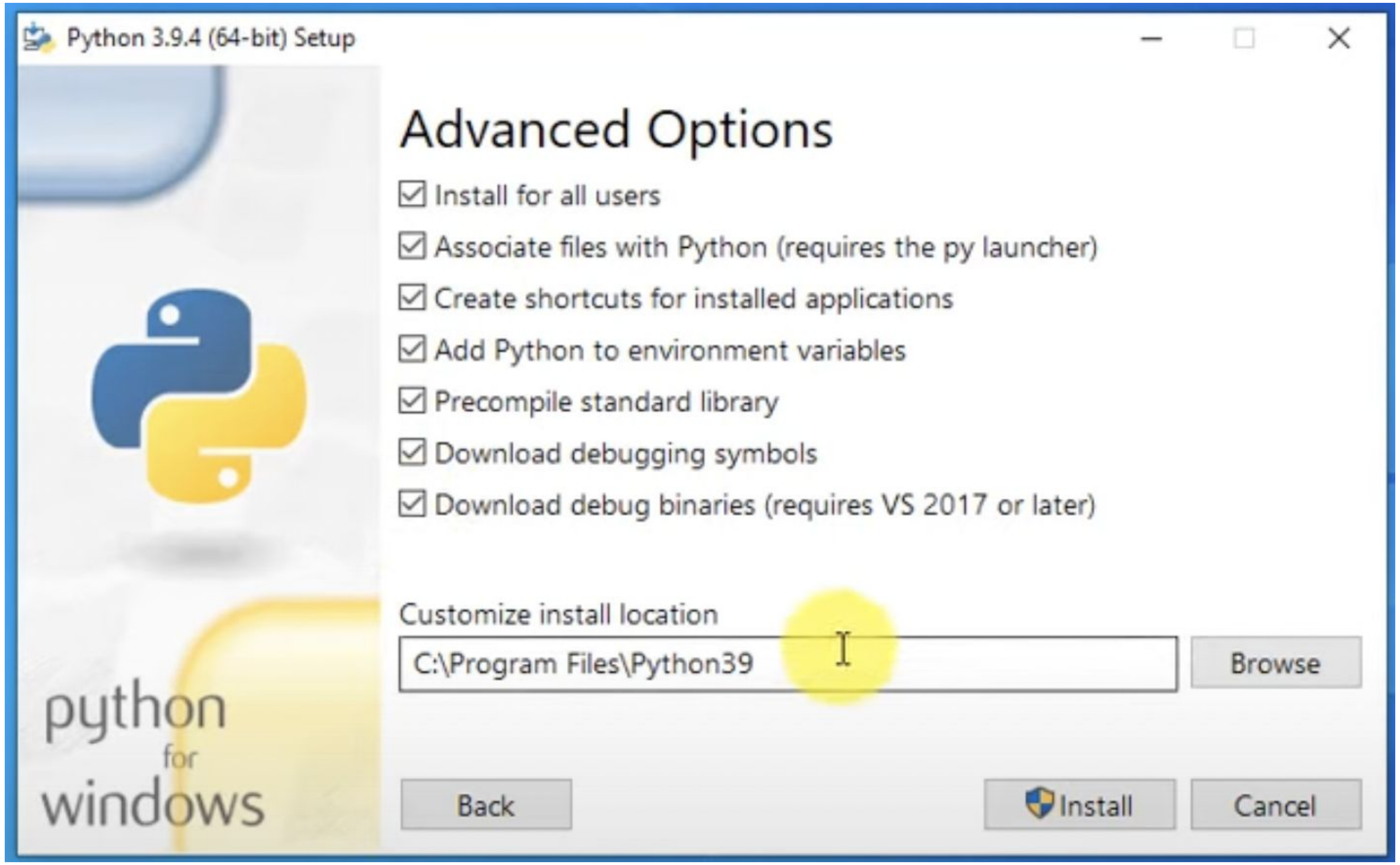
☑ Python test suite

Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)

Installs the global 'py' launcher to make it easier to start Python.

Step – 5 Select advanced options


Python 3.9.4 (64-bit) Setup

Advanced Options

☑ Install for all users

☑ Associate files with Python (requires the py launcher)

☑ Create shortcuts for installed applications

☑ Add Python to environment variables

☑ Precompile standard library

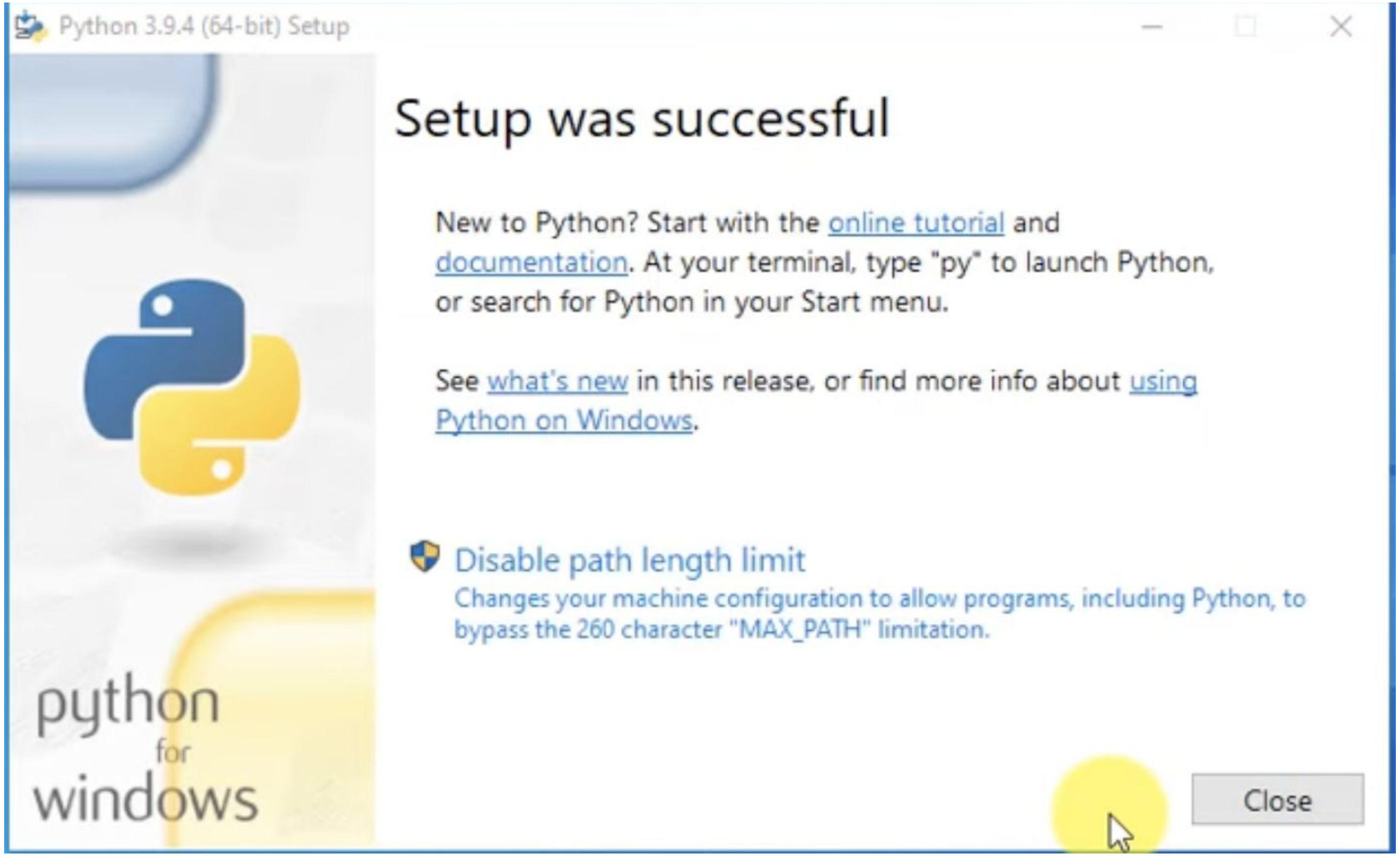
☑ Download debugging symbols

☑ Download debug binaries (requires VS 2017 or later)

Customize install location

C:\Program Files\Python39

Step – 5 installation Completed



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

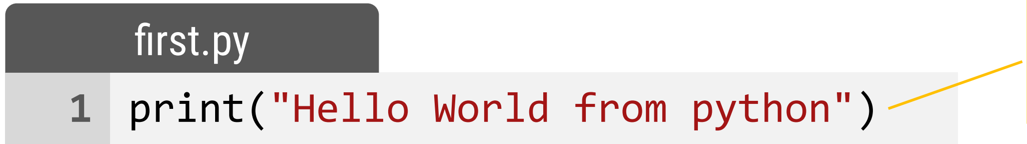
See what's new in this release, or find more info about using Python on Windows.

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Hello World using Python

- To write python programs, we can use any text editors or IDE (Integrated Development Environment), Initially we are going to use Visual Studio Code.
- Create new file in editor, save it as **first.py** (Extensions for python programs will be .py)

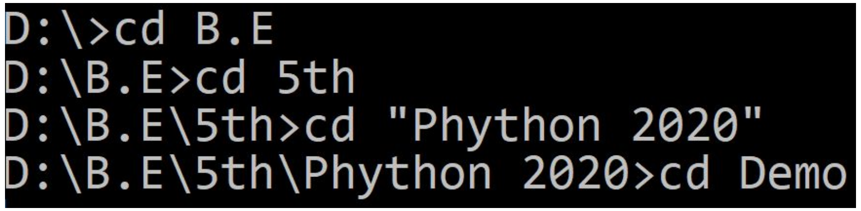


first.py

```python
print("Hello World from python")
```

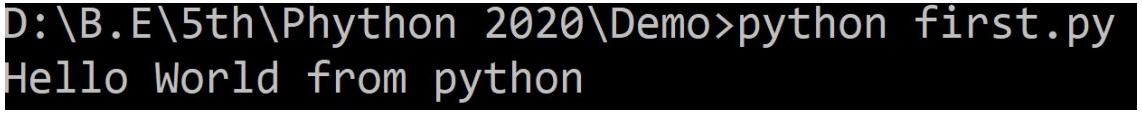
Python line does not end with ;

- To run the python file open command prompt and change directory to where your python file is



```
D:\>cd B.E
D:\B.E>cd 5th
D:\B.E\5th>cd "Phython 2020"
D:\B.E\5th\Phython 2020>cd Demo
```

- Next, run python command (python filename.py)



```
D:\B.E\5th\Phython 2020\Demo>python first.py
Hello World from python
```

Indentation in python

- Indentation in general means indenting words or spaces or lines in the document to follow the styling rule for documentation
- Indentation in Python refers to the (spaces and tabs) that are used at the beginning of a statement.
- The statements with the same indentation belong to the same group called a suite.
- In many different programming languages like C, C++, Java, etc. use flower brackets or braces {} to define or to identify a block of code in the program, whereas in Python, it is done using the spaces or tabs, which is known as indentation and also it is generally known as 4 space rule in Pep8 documentation of rules for styling and designing the code for Python.

Example.py

```
if Condition:
    print("True")
 else:
    print("False")

```

IndentationError: unindent does not match any outer indentation level

print() Function

- The print() function prints the specified message to the screen, or other standard output device.
- the object will be converted into a string before written to the screen.

Syntax

```
print(object(s), sep=separator, end=end)
```

Parameter Name	Description
Object(s)	Any number of objects. Will be converted into string before printing.
sep	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end	Optional. Specify what to print at the end. Default is '\n'

input() Function

- The input() function allows user to enter values.
- Whatever you enter as input, the input function converts it into a string. If you enter an integer value still input() function convert it into a string.

Syntax

```
input(prompt)
```

Parameter Name	Description
prompt(s)	A String, representing a default message before the input.

Example

Example.py

```python
a = 4
str = input("Enter your name = ")
num = int(input("Enter Number = "))

print(4,5)
print("Hello","World")
print("Hello","World",sep="-")
print("Hello","DU",sep="-",end=" ")
print("Hello " + "DU ")
print("a = ", a,end=", ")
print("num = ", num)
print("your Name ="+ str)
print("Your Name ",str,sep="=")
```

Output

```
Enter your name = ABC
Enter Number = 10
4 5
Hello World
Hello-World
Hello-DU Hello DU
a =  4, num =  10
your Name =ABC
Your Name =ABC
```

Program 1

- WAP to develop simple calculator in python

Example1.py

```python
number1 = int(input("Enter first number: "))
number2 = int(input("Enter second number: "))
print(number1, "+", number2, "=", number1+number2)
print(number1, "*", number2, "=", number1*number2)
print(number1, "-", number2, "=", number1-number2)
print(number1, "/", number2, "=", number1/number2)
```

Output

```
Enter first number: 3
Enter second number: 3
3 + 3 = 6
3 * 3 = 9
3 - 3 = 0
3 / 3 = 1.0
```

Program 2

- WAP to calculate simple interest

Example2.py

```
P = int(input("Enter P: "))
R = int(input("Enter R: "))
T = int(input("Enter N: "))
Simple_interest = (P * R * T) / 100
print("The simple interest is:", Simple_interest)
```

Output

```
Enter P: 10000
Enter R: 5
Enter N: 1
The simple interest is: 500.0
```

Program 3

- WAP to calculate area of circle.

Example3.py

```python
PI = 3.14
r = float(input("Enter the radius of a circle:"))
area = PI * r * r
print("Area of a circle = %.2f" %area)
```

Output

```
Enter the radius of a circle:100
Area of a circle = 31400.00
```

Data types in Python

Name	Type	Description
Data Types		
Integer	int	Whole number such as 0,1,5, -5 etc..
Float	float	Numbers with decimal points such as 1.5, 7.9, -8.2 etc..
String	str	Sequence of character (Ordered) such as “darshan”, ‘college’, “રાજકોટ” etc..
Boolean	bool	Logical values indicating **T**ure or **F**alse (T and F here are capital in python)

Data Structures		
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, “darshan”, True, 102.3]
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “darshan”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “darshan”, True, 102.3 }
		Unordered key : value pair of objects , will be represented with **curly**

Variables in Python

- A Python variable is a reserved memory location to store values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Python uses Dynamic Typing so,
  - We need not to specify the data types to the variable as it will internally assign the data type to the variable according to the value assigned.
  - we can also reassign the different data type to the same variable, variable data type will change to new data type automatically.
  - We can check the current data type of the variable with **type(variablename)** in-built function.
- Rules for variable name
  - Name can not start with digit
  - Space not allowed
  - Can not contain special character
  - Python keywords not allowed
  - **Should** be in lower case

Example of Python variable

- Example :

demo.py

```python
x = 10
print(x)
print(type(x))

y = 123.456
print(y)

x = "darshan insitute of engneering and technology"
print(x)
print(type(x))
```

Reassign same variable to hold different data type

Run in terminal

```
python demo.py
```

Output

```
10
int
123.456
Darshan institute of engineering and technology
str
```

String in python

- String is Ordered Sequence of character such as “darshan”, ‘college’, “રાજકોટ” etc..
- Strings are arrays of bytes representing Unicode characters.
- String can be represented as single, double or triple quotes.
- String with triple Quotes allows multiple lines.
- String in python is immutable.
- Square brackets can be used to access elements of the string, Ex. “Darshan”[1] = a, characters can also be accessed with reverse index like “Darshan”[-1] = n.

String index

```
            x =  "  D  a  r  s  h  a  n  "
        index =     0  1  2  3  4  5  6
Reverse index =    -7 -6 -5 -4 -3 -2 -1
```

String functions in python

- Python has lots of built-in methods that you can use on strings, we are going to cover some frequently used methods for string like
  - **len()**
  - count()
  - title(), lower(), upper()
  - istitle(), islower(), isupper()
  - find(), rfind(), replace()
  - index(), rindex()
  - Methods for validations like
    - isalpha(), isalnum(), isdecimal(), isdigit()
  - strip(), lstrip(), rstrip()
  - split
  - Etc..
- **Note** : len() is not the method of the string but can be used to get the length of the string

lendemo.py

```python
x = "Darshan"
print(len(x))
```

Output : 7 (length of “Darshan”)

String methods (cont.)

- **count**() method will returns the number of times a specified value occurs in a string.

countdemo.py

```python
x = "Darshan"
ca = x.count('a')
print(ca)
```

Output : 2 (occurrence of ‘a’ in “Darshan”)

- **title**(), **lower**(), **upper**() will returns capitalized, lower case and upper case string respectively.

changecase.py

```python
x = "darshan Institute, rajkot"
c = x.title()
l = x.lower()
u = x.upper()
print(c)
print(l)
print(u)
```

Output : Darshan Institute, Rajkot

Output : darshan institute, rajkot

Output : DARSHAN INSTITUTE, RAJKOT

String methods (cont.)

- **istitle**(), **islower**(), **isupper**() will returns True if the given string is capitalized, lower case and upper case respectively.

checkcase.py

```python
x = 'darshan institute, rajkot'
c = x.istitle()
l = x.islower()
u = x.isupper()
print(c)
print(l)
print(u)
```

Output : *False*

Output : *True*

Output : *False*

- **strip**() method will remove whitespaces from both side of the string and returns the string.

stripdemo.py

```python
x = '     Darshan  '
f = x.strip()
print(f)
```

Output : *Darshan* (without space)

- **rstrip**() and **lstrip**() will remove whitespaces from right and left side respectively.

String methods (cont.)

- **find**() method will search the string and returns the index at which they find the specified value

finddemo.py

```python
x = 'darshan institute, rajkot, india'
f = x.find('in')
print(f)
```

Output : 8 (occurrence of ‘in’ in x)

- **rfind**() will search the string and returns the last index at which they find the specified value

rfinddemo.py

```python
x = 'darshan institute, rajkot, india'
r = x.rfind('in')
print(r)
```

Output : 27 (last occurrence of ‘in’ in x)

- **Note** : **find**() and **rfind**() will **return -1** if they are unable to find the given string.
- **replace**() will replace str1 with str2 from our string and return the updated string

replacedemo.py

```python
x = 'darshan institute, rajkot, india'
r = x.replace('india','INDIA')
print(r)
```

Output :
“darshan institute, rajkot, INDIA”

String methods (cont.)

- **index**() method will search the string and returns the index at which they find the specified value, but if they are unable to find the string it will raise an exception.

indexdemo.py

```python
x = 'darshan institute, rajkot, india'
f = x.index('in')
print(f)
```

Output : 8 (occurrence of ‘in’ in x)

- **rindex**() will search the string and returns the last index at which they find the specified value , but if they are unable to find the string it will raise an exception.

rindexdemo.py

```python
x = 'darshan institute, rajkot, india'
r = x.rindex('in')
print(r)
```

Output : 27 (last occurrence of ‘in’ in x)

- Note : **find**() and **index**() are almost same, the only difference is if **find**() is unable to find the string it will return -1 and if **index**() is unable to find the string it will raise an exception.

String methods (cont.)

- **isalnum**() method will return true if all the characters in the string are alphanumeric (i.e either alphabets or numeric).

isalnumdemo.py

```python
x = 'darshan123'
f = x.isalnum()
print(f)
```

Output : *True*

- **isalpha**() and **isnumeric**() will return true if all the characters in the string are only alphabets and numeric respectively.
- **isdecimal**() will return true is all the characters in the string are decimal.

isdecimaldemo.py

```python
x = '123.5'
r = x.isdecimal()
print(r)
```

Output : *True*

- **Note** : **isnuberic**() and **isdigit**() are almost same, you suppose to find the difference as Home work assignment for the string methods.

String Slicing

- We can get the substring in python using string slicing, we can specify start index, end index and steps (colon separated) to slice the string.

syntax

```
x = 'darshan institute of engineering and technology, rajkot, gujarat, INDIA'
subx = x[startindex:endindex:steps]
```

endindex will not be included in the substring

strslicedemo.py

```
x = 'darshan institute of engineering and technology, rajkot, gujarat, INDIA'
subx1 = x[0:7]
subx2 = x[49:55]
subx3 = x[66:]
subx4 = x[::2]
subx5 = x[::-1]
print(subx1)
print(subx2)
print(subx3)
print(subx4)
print(subx5)
```

Output : *darshan*

Output : *rajkot*

Output : *INDIA*

Output : *drhnisiueo niern n ehooy akt uaa,IDA*

Output : *AIDNI ,tarajug ,tokjar ,ygolonhcet dna gnireenigne fo etutitsni nahsrad*

String print format

- **str.format()** is one of the *string formatting methods* in Python3, which allows multiple substitutions and value formatting.
- This method lets us concatenate elements within a string through positional formatting.

strformat.py

```python
x = '{} institute, rajkot'
y = x.format('darshan')
print(y)
print(x.format('ABCD'))
```

Output : *darshan institute, rajkot*

Inline function call
Output : *ABCD institute, rajkot*

- We can specify multiple parameters to the function

strformat.py

```python
x = '{} institute, {}'
y = x.format('darshan','rajkot')
print(y)
print(x.format('ABCD','XYZ'))
```

Output : *darshan institute, rajkot*

Inline function call
Output : *ABCD institute, XYZ*

String print format (cont.)

- We can specify the order of parameters in the string

strformat.py

```python
x = '{1} institute, {0}'
y = x.format('darshan','rajkot')
print(y)
print(x.format('ABCD','XYZ'))
```

Output : *rajkot institute, darshan*

Inline function call
Output : *XYZ institute, ABCD*

- We can also specify alias within the string to specify the order

strformat.py

```python
x = '{collegename} institute, {cityname}'
print(x.format(collegename='darshan',cityname='rajkot'))
```

Output : *darshan institute, rajkot*

- We can format the decimal values using format method

strformat.py

```python
per = (438 / 500) * 100
x = 'result = {r:3.2f} %'.format(r=per)
print(x)
```

Output : *result = 87.60 %*

width

precision

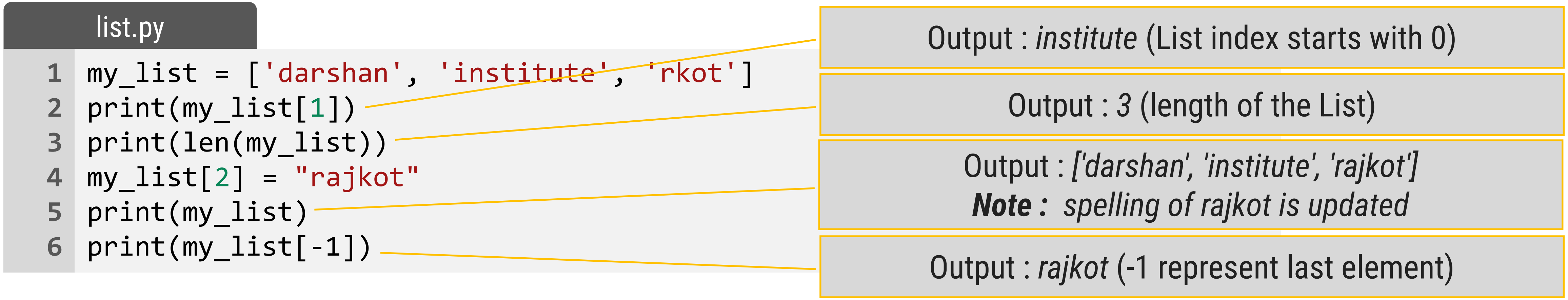
Data structures in python

- There are four built-in data structures in Python - *list, dictionary, tuple and set*.

Name	Type	Description
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, “darshan”, True, 102.3]
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { “college”: “darshan”, “code”: “054” }
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “darshan”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “darshan”, True, 102.3 }

List

- List is a mutable ordered sequence of objects, duplicate values are allowed inside list.
- List will be represented by square brackets [ ].
- Python does not have array, List can be used similar to Array.



list.py

```python
my_list = ['darshan', 'institute', 'rkot']
print(my_list[1])
print(len(my_list))
my_list[2] = "rajkot"
print(my_list)
print(my_list[-1])
```

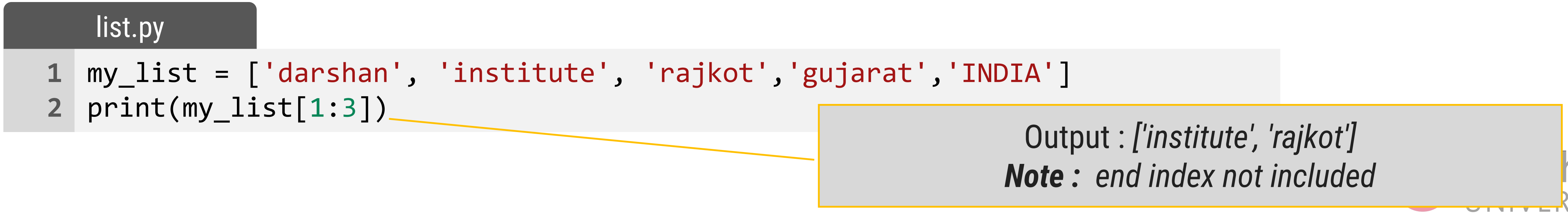
Output : *institute* (List index starts with 0)

Output : *3* (length of the List)

Output : *['darshan', 'institute', 'rajkot']*
Note : *spelling of rajkot is updated*

Output : *rajkot* (-1 represent last element)

- We can use slicing similar to string in order to get the sub list from the list.



list.py

```python
my_list = ['darshan', 'institute', 'rajkot','gujarat','INDIA']
print(my_list[1:3])
```

Output : *['institute', 'rajkot']*
Note : *end index not included*

List methods

- append() method will add element at the end of the list.

appendlistdemo.py

```python
my_list = ['darshan', 'institute', 'rajkot']
my_list.append('gujarat')
print(my_list)
```

Output : ['darshan', 'institute', 'rajkot', 'gujarat']

- insert() method will add element at the specified index in the list

insertlistdemo.py

```python
my_list = ['darshan', 'institute', 'rajkot']
my_list.insert(2,'of')
my_list.insert(3,'engineering')
print(my_list)
```

Output : ['darshan', 'institute', 'of', 'engineering', 'rajkot']

- extend() method will add one data structure (List or any) to current List

extendlistdemo.py

```python
my_list1 = ['darshan', 'institute']
my_list2 = ['rajkot','gujarat']
my_list1.extend(my_list2)
print(my_list1)
```

Output : ['darshan', 'institute', 'rajkot', 'gujarat']

List methods (cont.)

- pop() method will remove the last element from the list and return it.

poplistdemo.py

```python
my_list = ['darshan', 'institute','rajkot']
temp = my_list.pop()
print(temp)
print(my_list)
```

Output : *rajkot*

Output : *['darshan', 'institute']*

- remove() method will remove first occurrence of specified element

removelistdemo.py

```python
my_list = ['darshan', 'institute', 'darshan','rajkot']
my_list.remove('darshan')
print(my_list)
```

Output : *['institute', 'darshan', 'rajkot']*

- clear() method will remove all the elements from the List

clearlistdemo.py

```python
my_list = ['darshan', 'institute', 'darshan','rajkot']
my_list.clear()
print(my_list)
```

Output : *[]*

- index() method will return first index of the specified element.

List methods (cont.)

- count() method will return the number of occurrence of the specified element.

countlistdemo.py

```python
my_list = ['darshan', 'institute', 'darshan'
c = my_list.count('darshan')
print(c)
```

Output : *2*

- reverse() method will reverse the elements of the List

reverselistdemo.py

```python
my_list = ['darshan', 'institute','rajkot']
my_list.reverse()
print(my_list)
```

Output : *['rajkot', ‘institute','darshan']*

- sort() method will sort the elements in the List

sortlistdemo.py

```python
my_list = ['darshan', 'college','of','enginnering','rajkot']
my_list.sort()
print(my_list)
my_list.sort(reverse=True)
print(my_list)
```

Output : *['college', 'darshan', 'enginnering', 'of', 'rajkot']*

Output : *['rajkot', 'of', 'enginnering', 'darshan', 'college']*

Set

- Set is a unordered collection of unique objects.
- Set will be represented by curly brackets { }.

setdemo.py

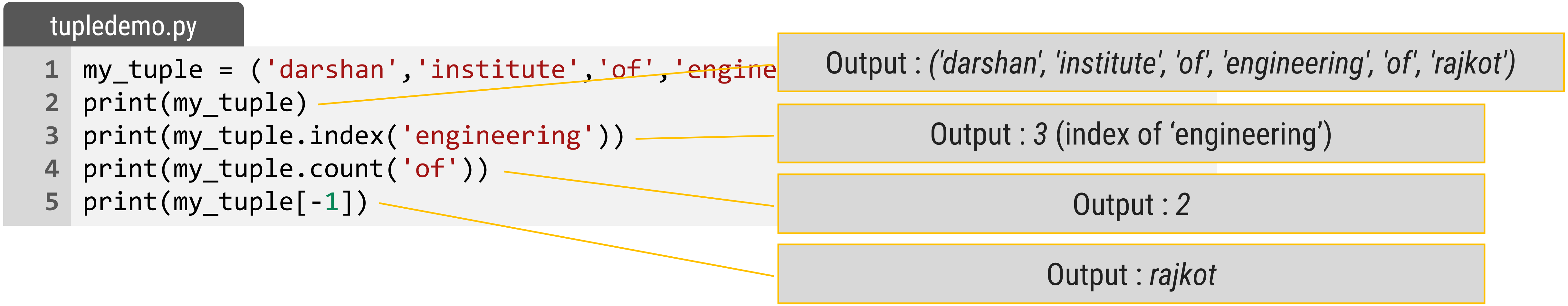
```python
my_set = {1,1,1,2,2,5,3,9}
print(my_set)
```

Output : *{1, 2, 3, 5, 9}*

- Set has many in-built methods such as add(), clear(), copy(), pop(), remove() etc.. which are similar to methods we have previously seen.
- Only difference between Set and List is that Set will have only unique elements and List can have duplicate elements.

Tuple

- Tuple is a immutable ordered sequence of objects, duplicate values are allowed inside list.
- Tuple will be represented by round brackets ( ).
- Tuple is similar to List but List is mutable whereas Tuple is immutable.



tupledemo.py

```python
my_tuple = ('darshan','institute','of','engine
print(my_tuple)
print(my_tuple.index('engineering'))
print(my_tuple.count('of'))
print(my_tuple[-1])
```

Output : *('darshan', 'institute', 'of', 'engineering', 'of', 'rajkot')*

Output : *3* (index of ‘engineering’)

Output : *2*

Output : *rajkot*

Dictionary

- Dictionary is a unordered collection of key value pairs.
- Dictionary will be represented by curly brackets { }.
- Dictionary is mutable.

syntax

```
my_dict = { 'key1':'value1', 'key2':'value2' }
```

Key value is seperated by :

Key value pairs is separated by ,

dictdemo.py

```
my_dict = {'college':"darshan", 'city':"rajkot",'type':"engineering"}
print(my_dict['college'])
print(my_dict.get('city'))
```

values can be accessed using key inside square brackets as well as using get() method

Output : *darshan*
rajkot

Dictionary methods

- keys() method will return list of all the keys associated with the Dictionary.

keydemo.py

```python
my_dict = {'college':"darshan", 'city':"rajkot",'type':"engineering"}
print(my_dict.keys())
```

Output : *['college', 'city', 'type']*

- values() method will return list of all the values associated with the Dictionary.

valuedemo.py

```python
my_dict = {'college':"darshan", 'city':"rajkot",'type':"engineering"}
print(my_dict.values())
```

Output : *['darshan', 'rajkot', 'engineering']*

- items() method will return list of tuples for each key value pair associated with the Dictionary.

itemsdemo.py

```python
my_dict = {'college':"darshan", 'city':"rajkot",'type':"engineering"}
print(my_dict.items())
```

Output : *[('college', 'darshan'), ('city', 'rajkot'), ('type', 'engineering')]*

List vs. tuple vs. set vs. dictionary

List	Tuple	Sets	**Dictionaries**
Ordered Data	Ordered Data	Unordered Data	Unordered Data
Mutable	immutable	mutable	mutable
Square braces.[]	Parenthesis ()	Curly brackets {}	Curly brackets {key : value}
Duplicate elements allowed	Duplicate elements allowed	No Duplicate elements	No Duplicate key
append()	Element cannot be added	add()	update()
We can create a list using the list() function.	We can create a tuple using the tuple() function.	We can create a set using the set()function.	We can create a dictionary using the dict() function.
my_list = [5,6,7,8]	my_tuple = (5,6,7,8)	my_sets = {3,4,5,6}	my_dictionaries = {"name" : "XYZ", "rollno." : "101"}

List Comprehension

- List comprehensions offer a way to create lists based on existing iterable. When using list comprehensions, lists can be built by using any iterable, including strings, list, tuples.
- For example, if we want to create a list of characters from the string, we can use for loop like below example,

ForLoop.py

```python
mystr = 'darshan'
mylist = []
for c in mystr:
    mylist.append(c)
print(mylist)
```

Output

```
['d', 'a', 'r', 's', 'h', 'a', 'n']
```

Syntax

```
[ expression for item in iterable ]
OR
[ expression for item in iterable if condition ]
```

Example1.py

```python
mylist = [c for c in 'darshan']
print(mylist)
```

List Comprehension (cont.)

- Similarly, we can use list comprehensions in many cases where we want to create a list out of other iterable, let’s see another example of the use of List Comprehension.
- Example (Using for loop):

ForLoop1.py

```python
#list of square from 1 to 10
mylist = []
for i in range(1,11):
    mylist.append(i**2)
print(mylist)
```

Output

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Example2.py

```python
# list of square from 1 to 10
mylist = [ i**2 for i in range(1,11) ]
print(mylist)
```

Magic Command

- IPython’s special commands (which are not built into Python itself) are known as “magic” commands.
- These are designed to facilitate common tasks and enable you to easily control the behavior of the IPython system.
- magic command is any command prefixed by the percent symbol %
- For example, you can check the execution time of any Python statement, such as a matrix multiplication, using the %timeit magic function (which will be discussed in more detail later):
- Magic commands come in two type:
  - line magics, which are denoted by a single % prefix and operate on a single line of input,
  - cell magics, which are denoted by a double %% prefix and operate on multiple lines of input.

Magic Command

Command	Description
%quickref	Display the IPython Quick Reference Card
%magic	Display detailed documentation for all of the available magic commands
%who, %whos, %who_ls	Print all information related to interactive variables.
%hist	Print command input (and optionally output) history
%cd	Change the current working directory.
%paste	Execute preformatted Python code from clipboard
%cpaste	Open a special prompt for manually pasting Python code to be executed
%reset	Delete all variables/names defined in interactive namespace
%page *OBJECT*	Pretty-print the object and display it through a pager
%run *script.py*	Run a Python script inside IPython
%time *statement*	Report the execution time of a single statement
%timeit *statement*	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time

Introspection

- Using a question mark (?) before or after a variable will display some general infor- mation about the object:

Introspection

```
In[1]:  b=[1,2,3]
In[2]:  b?

        Type: list
        String Form:[1, 2, 3]
        Length: 3
        Docstring:
        list() -> new empty list
        list(iterable) -> new list
        initialized from iterable's items
```

- Then using ? shows us the docstring if defined.
- Using ?? will also show the function’s source code if possible:

Introspection

```
In[1]:  add_numbers??

        Signature: add_numbers(a, b) Source:
        def add_numbers(a, b):
            """
            Add two numbers together Returns
            -------
            the_sum : type of arguments
            """
            return a+b
        File: <ipython-input-9-6a548a216e27> Type:
        function
```

The %run Command

- You can run any file as a Python program inside the environment of your IPython session using the %run command.

demo.py

```python
def f(x, y, z):
        return(x+y)/z

a=5
b=6
c=7.5

result = f(a, b, c)
```

IPython

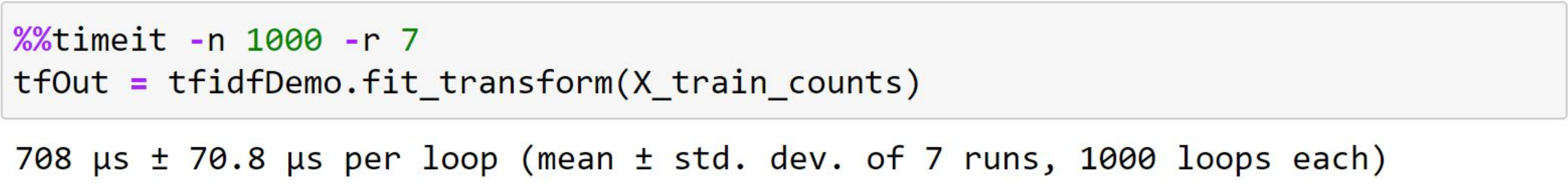
```
  In[1]:  %run demo.py

  In[2]:  c
 Out[2]:  7.5

  In[3]:  result
 Out[3]:  1.4666666666666666
```

- All of the variables (imports, functions, and globals) defined in the file (up until an exception, if any, is raised) will then be accessible in the IPython shell:

timeit (Magic Command in Jupyter Notebook)

- We can find the time taken to execute a statement or a cell in a Jupyter Notebook with the help of timeit magic command.
- This command can be used both as a line and cell magic:
  - In line mode you can time a single-line statement (though multiple ones can be chained with using semicolons).
  - In cell mode, the statement in the first line is used as setup code (executed but not timed) and the body of the cell is timed. The cell body has access to any variables created in the setup code.
- Syntax :
  - Line : %timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o]
  - Cell : %%timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o]

  Here, -n flag represents the number of loops and –r flag represents the number of repeats
- Example :



```
%%timeit -n 1000 -r 7
tfOut = tfidfDemo.fit_transform(X_train_counts)
```

```
708 µs ± 70.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```