

The background is a dark, textured surface resembling a chalkboard. It is covered with faint, light-colored chalk drawings of various scientific and mathematical symbols. These include a large 'V' in the top left, a globe in the top center, a microscope on the left side, a plus sign, a percentage sign, and other geometric shapes scattered throughout.

# **Introduction to Data Analysis and Basic Python Data Structure**

## **Unit-1**

# Outline

- What is Data Science?
- Why Data Science?
- Data Science Components
- Data Science Process
- Tools for Data Science
- Difference between Data Science with BI (Business Intelligence)
- Applications of Data Science

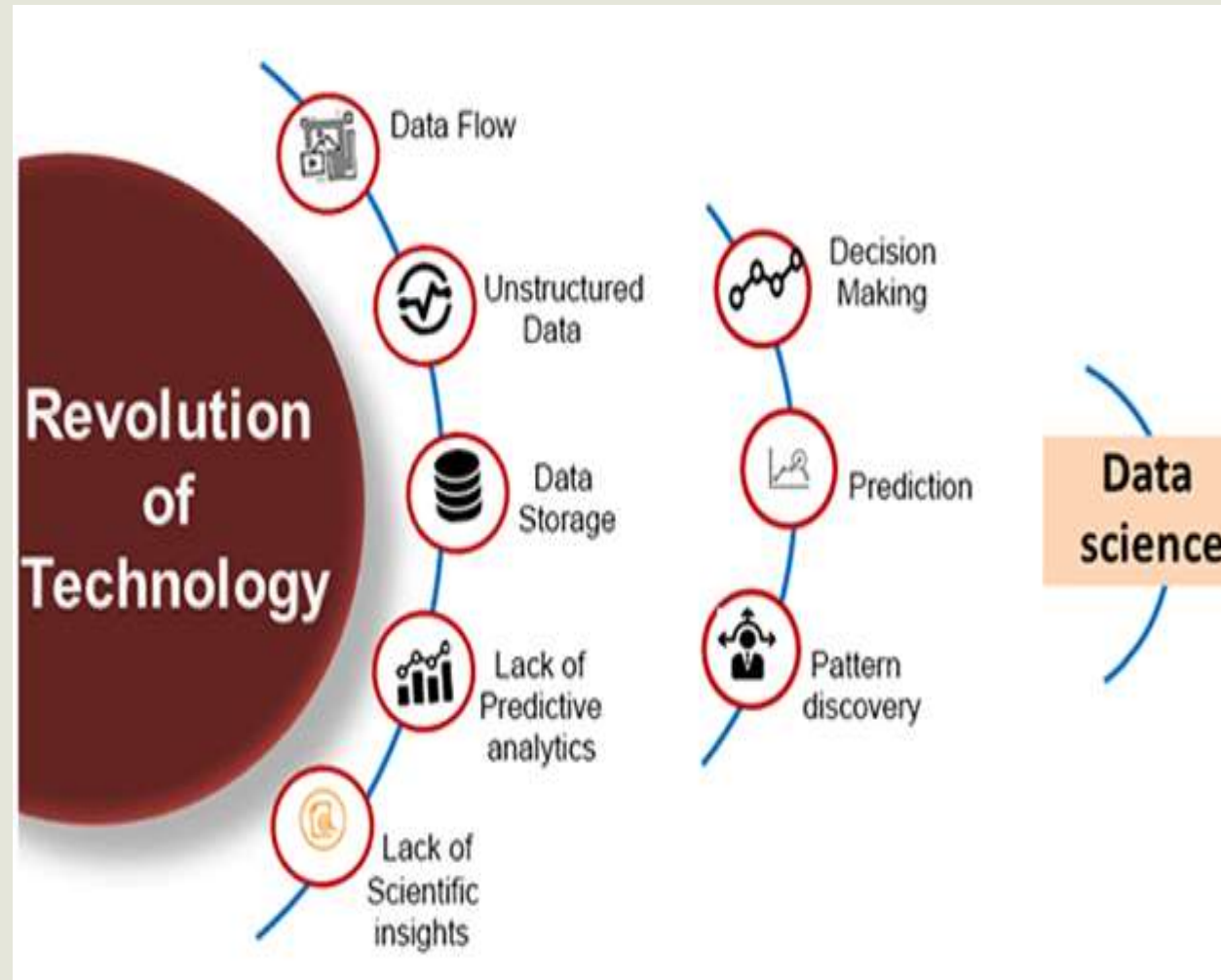
# What is Data Science?

- Data Science is the area of study which involves extracting insights from vast amounts of data using various scientific methods, algorithms, and processes.
- It helps you to discover hidden patterns from the raw data.
- The term Data Science has emerged because of the evolution of mathematical statistics, data analysis, and big data
- Data Science is an interdisciplinary field that allows you to extract knowledge from **structured or unstructured data**.
- Data science enables you to translate a business problem into a research project and then translate it back into a practical solution.

# Why Data Science?

- Here are significant advantages of using Data Analytics Technology:
  - Data is very **important** for today's world.
  - With the right tools, technologies, algorithms, we can use data and convert it into a **distinct business advantage**.
  - Data Science can help you to **detect fraud** using advanced machine learning algorithms
  - It helps you to prevent any **significant monetary losses**
  - Allows to build **intelligence ability in machines**
  - It enables you to take **better and faster decisions**

# Why Data Science?

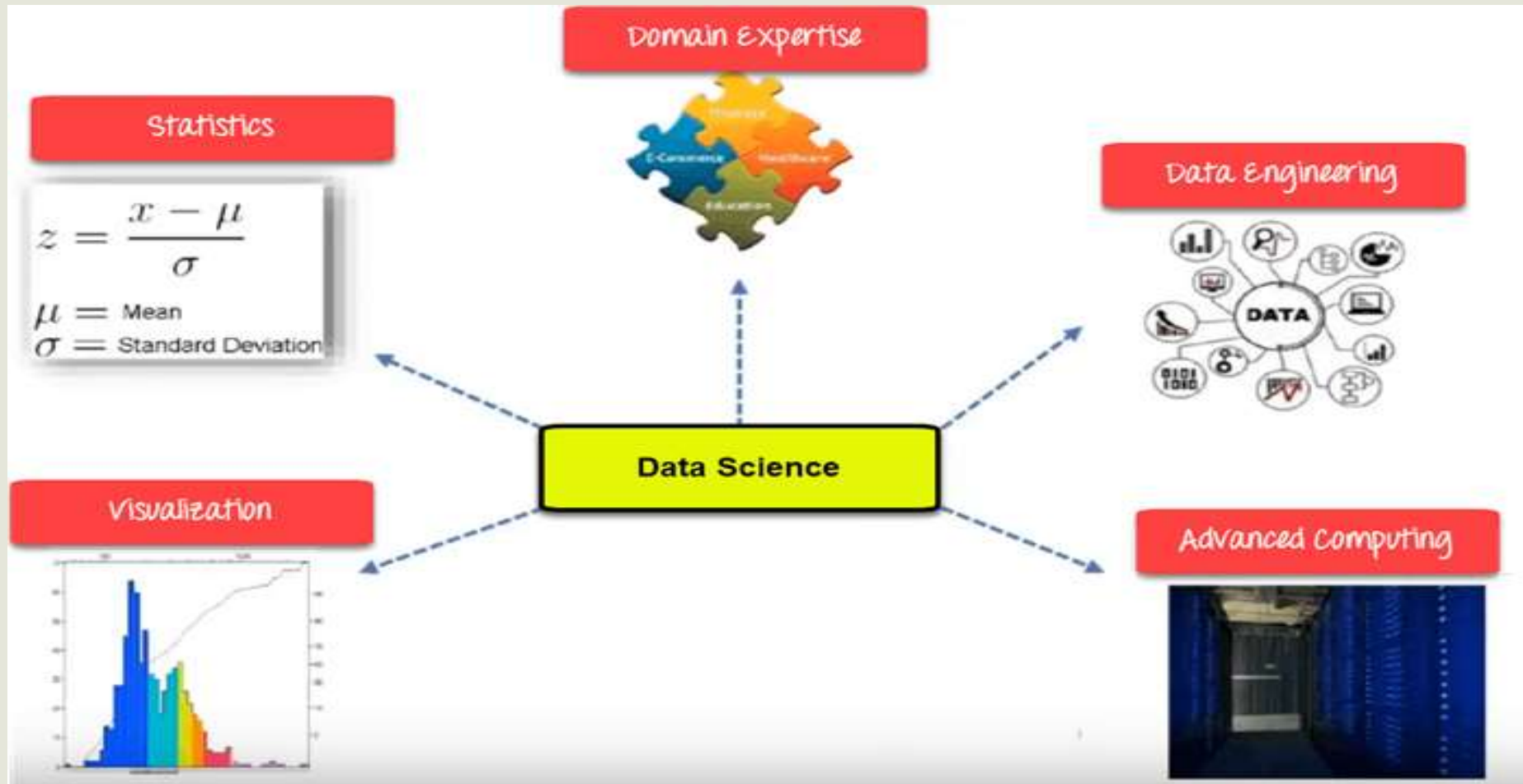




# Why?

- Following are some main reasons for using data science technology:
  - With the help of data science technology, we can convert the massive amount of **raw and unstructured data into meaningful insights**.
  - Data science technology is opting by various companies, whether it is a big brand or a startup.
  - Google, Amazon, Netflix, etc, which handle the huge amount of data, are using data science algorithms for better customer experience.
  - Data science is working for automating transportation such as creating a self-driving car, which is the future of transportation.
  - Data science can help in different predictions such as various survey, elections, flight ticket confirmation, etc.

# Data Science Components



# Data Science Components

- **Statistics:**

- Statistics is the most critical unit of Data Science basics, and it is the method or science of collecting and analyzing numerical data in large quantities to get useful insights.

- **Visualization:**

- Visualization technique helps you access huge amounts of data in easy to understand and digestible visuals.

- **Machine Learning:**

- Machine Learning explores the building and study of algorithms that learn to make predictions about unforeseen/future data.

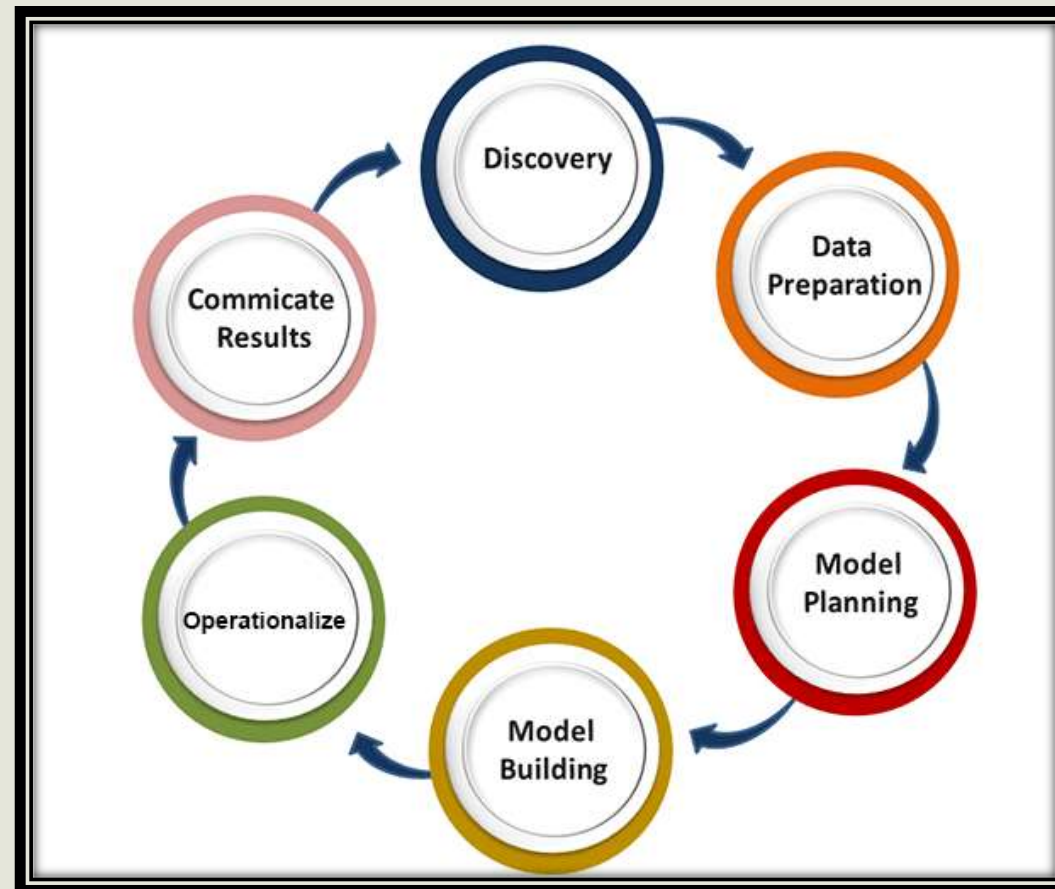
- **Deep Learning:**

- Deep Learning method is new machine learning research where the algorithm selects the analysis model to follow.



# Data Science Process

- Now in this Data Science , we will learn the Data Science Process:



# Data Science Process

## 1. Discovery:

- Discovery step involves acquiring data from all the identified internal & external sources, which helps you answer the business question.
- The data can be:
  - Logs from webserver
  - Data gathered from social media
  - Census datasets
  - Data streamed from online sources using APIs

# Data Science Process

## 2. Preparation:

- Data can have many inconsistencies like missing values, blank columns, an incorrect data format, which needs to be cleaned.
- You need to process, explore, and condition data before modelling.
- The cleaner your data, the better are your predictions.

## 3. Model Planning:

- In this stage, you need to determine the method and technique to draw the relation between input variables.
- Planning for a model is performed by using different statistical formulas and visualization tools.
- SQL analysis services, R, and SAS/access are some of the tools used for this purpose.

# Data Science Process

## 4. Model Building:

- In this step, the actual model building process starts. Here, Data scientist distributes datasets for training and testing.

## 5. Operationalize:

- You deliver the final baseline model with reports, code, and technical documents in this stage.
- Model is deployed into a real-time production environment.

## 6. Communicate Results

- In this stage, the key findings are communicated to all team members. This helps you decide if the project results are a success or a failure based on the inputs from the model.

# Tools for Data Science



Data Analysis	Data Warehousing	Data Visualization	Machine Learning
<b>R</b> , Spark, <b>Python</b> and <b>SAS</b>	<b>Hadoop</b> , SQL, <b>Hive</b>	R, <b>Tableau</b> , Raw	<b>Spark</b> , Azure ML studio



# Difference Between Data Science with BI (Business Intelligence)

Parameters	Business Intelligence	Data Science
<b>Perception</b>	Looking Backward	Looking Forward
<b>Data Sources</b>	Structured Data. Mostly SQL, but some time Data Warehouse	Structured and Unstructured data:Like logs, SQL, NoSQL, or text
<b>Approach</b>	Statistics & Visualization	Statistics, Machine Learning, and Graph
<b>Method</b>	Analytical: historical data	Scientific: goes deeper to know the reason for the data report
<b>Emphasis</b>	Past & Present	Analysis & Neuro-linguistic Programming
<b>Tools</b>	Pentaho, Microsoft BI, QlikView,	R, TensorFlow

# Applications of Data Science

- **Image recognition and speech recognition:**

- Data science is currently using for Image and speech recognition.
- When you upload an image on Facebook and start getting the suggestion to tag to your friends. This automatic tagging suggestion uses image recognition algorithm, which is part of data science.
- When you say something using, "**Ok Google/Siri**" and these devices respond as per voice control, so this is possible with speech recognition algorithm.

- **Gaming world:**

- In the gaming world, the use of Machine learning algorithms is increasing day by day.
- **EA Sports, Sony, Nintendo**, are widely using data science for enhancing user experience.

- **Internet search:**

- When we want to search for something on the internet, then we use different types of search engines such as **Google, Yahoo, Bing, Ask, etc.**
- All these search engines use the data science technology to make the search experience better, and you can get a search result with a fraction of seconds.

# Applications...

- **Transport:**

- Transport industries also using data science technology to create self-driving cars. With self-driving cars, it will be easy to reduce the number of road accidents.

- **Healthcare:**

- In the healthcare sector, data science is providing lots of benefits. Data science is being used for tumor detection, drug discovery, medical image analysis, virtual medical bots, etc.

- **Recommendation systems:**

- Most of the companies, such as Amazon, Netflix, Google Play, etc., are using data science technology for making a better user experience with personalized recommendations.

- **Risk detection:**

- Finance industries always had an issue of fraud and risk of losses, but with the help of data science, this can be rescued.
- Most of the finance companies are looking for the data scientist to avoid risk and any type of losses with an increase in customer satisfaction.

# Introduction of Python?

- **Python is a high-level, interpreted, interactive and object-oriented scripting language.**
- Python is designed to be **highly readable**. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

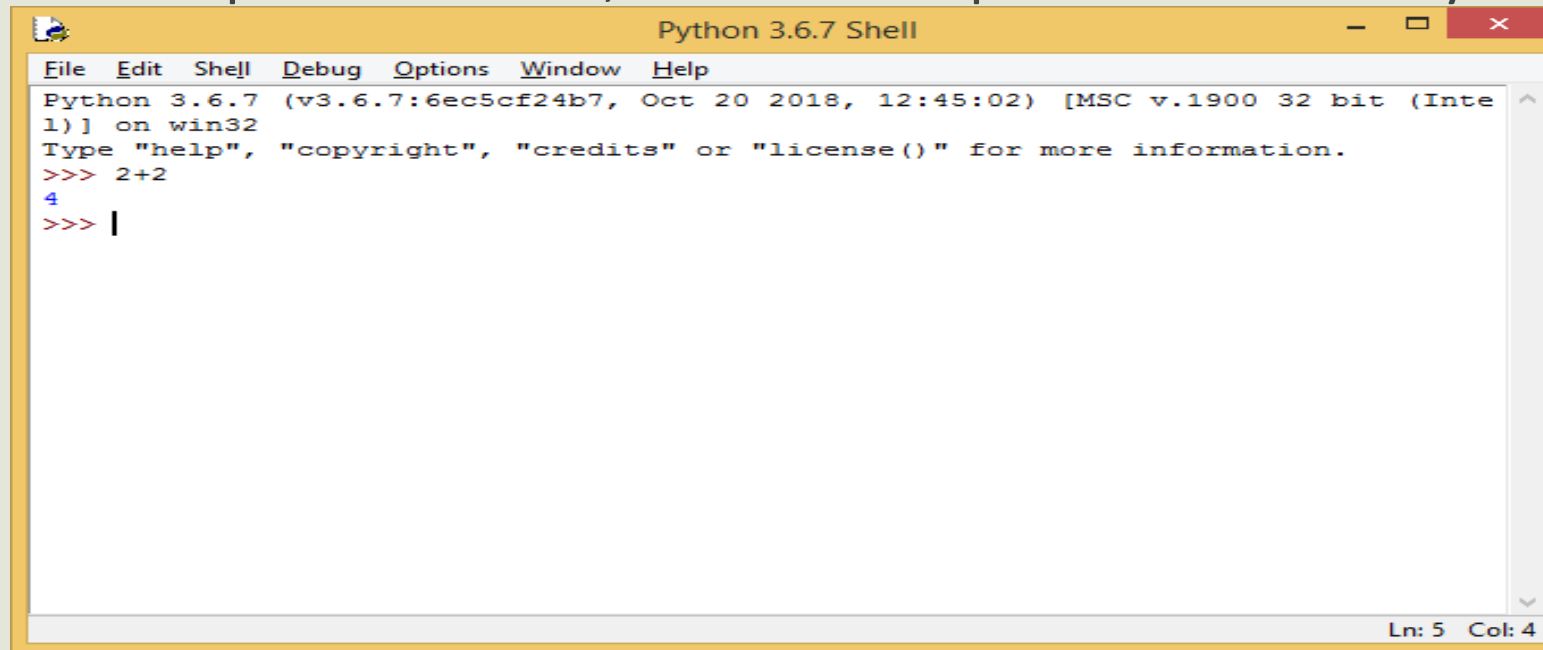
# Python...High Level Language

- Python is an example of a high-level language; other high-level languages you might have heard of are C++, PHP, Pascal, C#, and Java.
- Thus, programs written in a high-level language have to be translated into something more suitable before they can run.
- It is much easier to program in a high-level language so programs take less time to write, they are shorter and easier to read, and they are more likely to be correct.
- Portable, meaning that they can run on different kinds of computers with few or no modifications.



# Python...Interpreted Language

- The engine that translates and runs Python program is called the **Python Interpreter**: There are two ways to use it:
- immediate mode and script mode. In immediate mode, you type Python expressions into the Python Interpreter window, and the interpreter immediately shows the result:



```
Python 3.6.7 Shell
File Edit Shell Debug Options Window Help
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 12:45:02) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> |
```

# Pyhton...

- The `>>>` is called the Python prompt. The interpreter uses the prompt to indicate that it is ready for instructions.
- We typed `2 + 2`, and the interpreter evaluated our expression, and replied 4, and on the next line it gave a new prompt, indicating that it is ready for more input.
- Alternatively, you can write a program in a file and use the interpreter to execute the contents of the file. Such a file is called a script.
- **Files** have the advantage that they **can be saved to disk, printed, and so on.**
- Working **directly** in the interpreter is convenient for testing short bits of code because you get **immediate output.**

# Python History

- Invented at the **National Research Institute** for Mathematics and Computer Science in the Netherlands, early 90s by **Guido Van Rossum**
- Python was conceived in the late 1980s and its implementation was started in **December 1989**
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.
- **Open sourced** from beginning
- Considered a scripting language, but is much more Scalable, object oriented and functional from the beginning
- **Used by Google** from the beginning

# History Of Python

- Author is Guido Van Rossum
- Convinced in Late 1980
- Finally released in 1991
- Start implementation in December 1989 at CWI in Netherland.
- Successor of ABC Programming language.
- Name came from BBC's TV Show – 'Monty Python's Flying Circus'

# Python Version

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006

Python Version	Released Date
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016

Python 3.7

June 27, 2018



# Python Features:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

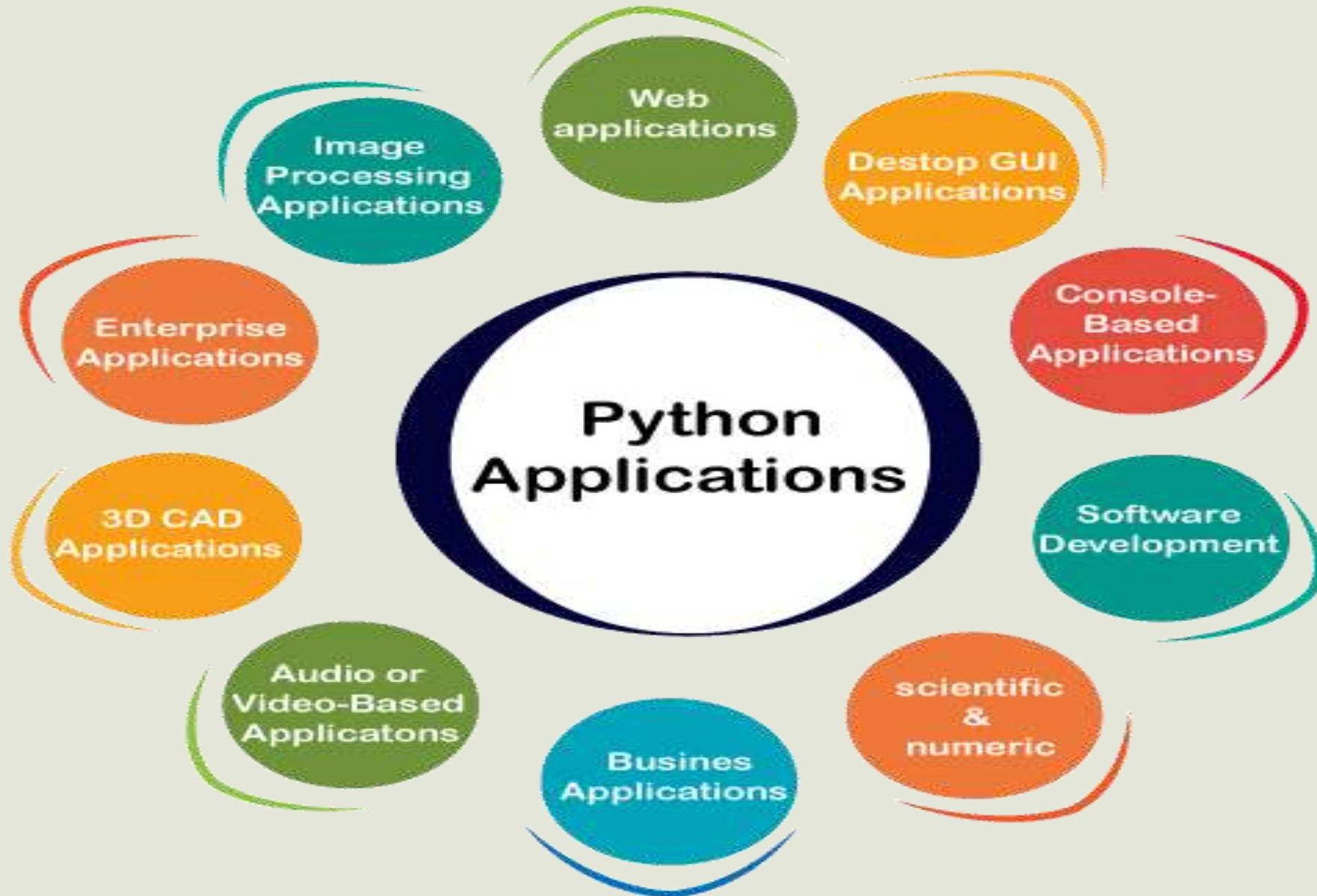
# Features...

- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

# Where Python is USED??

- **Google** makes extensive use of Python in its web search system.
- Intel, Cisco, HP, Seagate, Qualcomm, and IBM use Python for hardware testing.
- **ESRI(Environmental Systems Research Institute)** uses Python as an popular GIS mapping products.
- The **YouTube** video sharing service is largely written in Python.
- **Widely used** (Google, NASA, Yahoo, Electronic Arts, some UNIX scripts etc.)

# Applications of Python



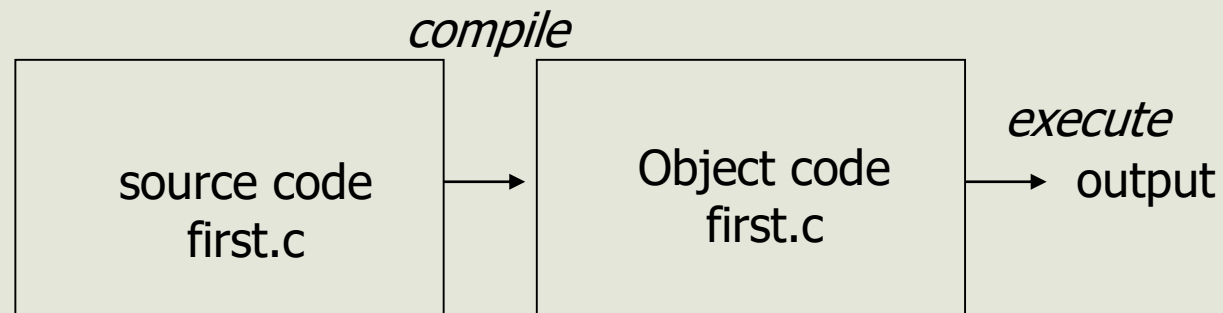
# Installing Python

- Python is pre-installed on most Unix systems, including Linux and MAC OS X
- But for Windows Operating Systems, user can download from the <https://www.python.org/downloads/>
- Form the above link download latest version of python IDE and install.

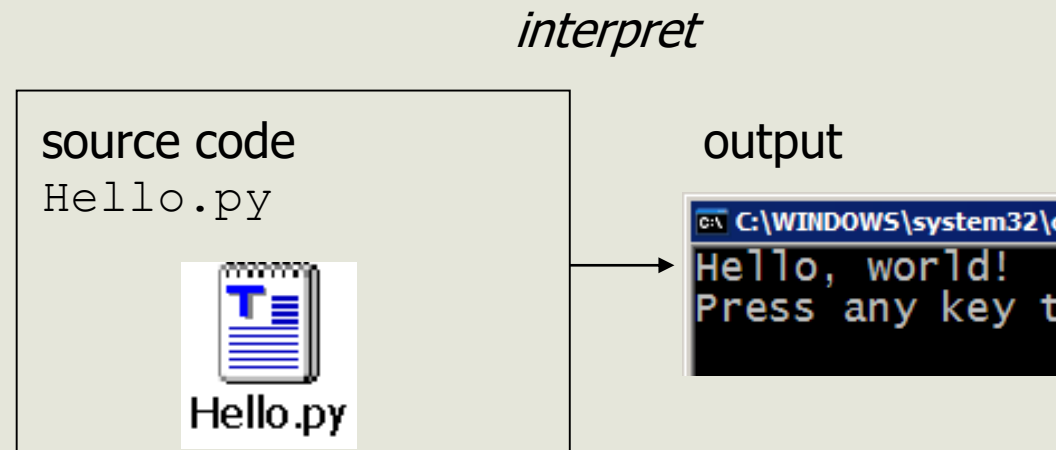


# Compiling and interpreting

- Many languages require you to compile (translate) your program into a form that the machine understands.



- Python is instead directly interpreted into machine instructions.



# Reserved Words

- The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters only.

<b>and</b>	<b>def</b>	<b>If</b>	<b>not</b>	<b>return</b>
<b>break</b>	<b>assert</b>	<b>exec</b>	<b>in</b>	<b>or</b>
<b>try</b>	<b>finally</b>	<b>del</b>	<b>import</b>	<b>pass</b>
<b>for</b>	<b>else</b>	<b>elif</b>	<b>with</b>	<b>print</b>
<b>is</b>	<b>continue</b>	<b>global</b>	<b>except</b>	<b>Raise</b>
<b>from</b>	<b>lambda</b>			

# Python Identifiers

- A Python identifier is a name used to identify a **variable, function, class, module or other object**.
- An identifier starts with a letter **A to Z or a to z or an underscore ( \_ )** followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as **@, \$, and % within identifiers**.
- Python is a **case sensitive programming** language

# Basic Syntax

```
# Python program to print  
print("Hello Student")
```

```
# Python program to declare variables  
myNumber = 3  
print(myNumber)
```

```
myNumber2 = 4.5  
print(myNumber2)
```

```
myNumber = "helloworld"  
print(myNumber)
```

# Python Variables

- Variables are containers for storing data values.

- Example

```
x = 10
```

```
y = "hello"
```

```
print(x)
```

```
print(y)
```

```
x, y, z = "Python", "Java", "Oracle"
```

```
x = y = z = "Python"
```



# Comments

## ■ Single Line Comments

- This comments starts with hash symbol(#) and are useful to mention that the entire line till the end should be treated as comments.

```
# Sinle Line Comments in Python  
a=10 #store 10 in variable a
```

## ■ Multi Line Comments

```
'''  
Multi Line Comments in Python  
Program to print Hello World in Python.  
'''  
print("Hello World")
```

```
"""  
Multi Line Comments in Python  
Program to print Hello World in Python.  
"""  
print("Hello World")
```



# Data Types and Operators

## Unit-1

# Indentation

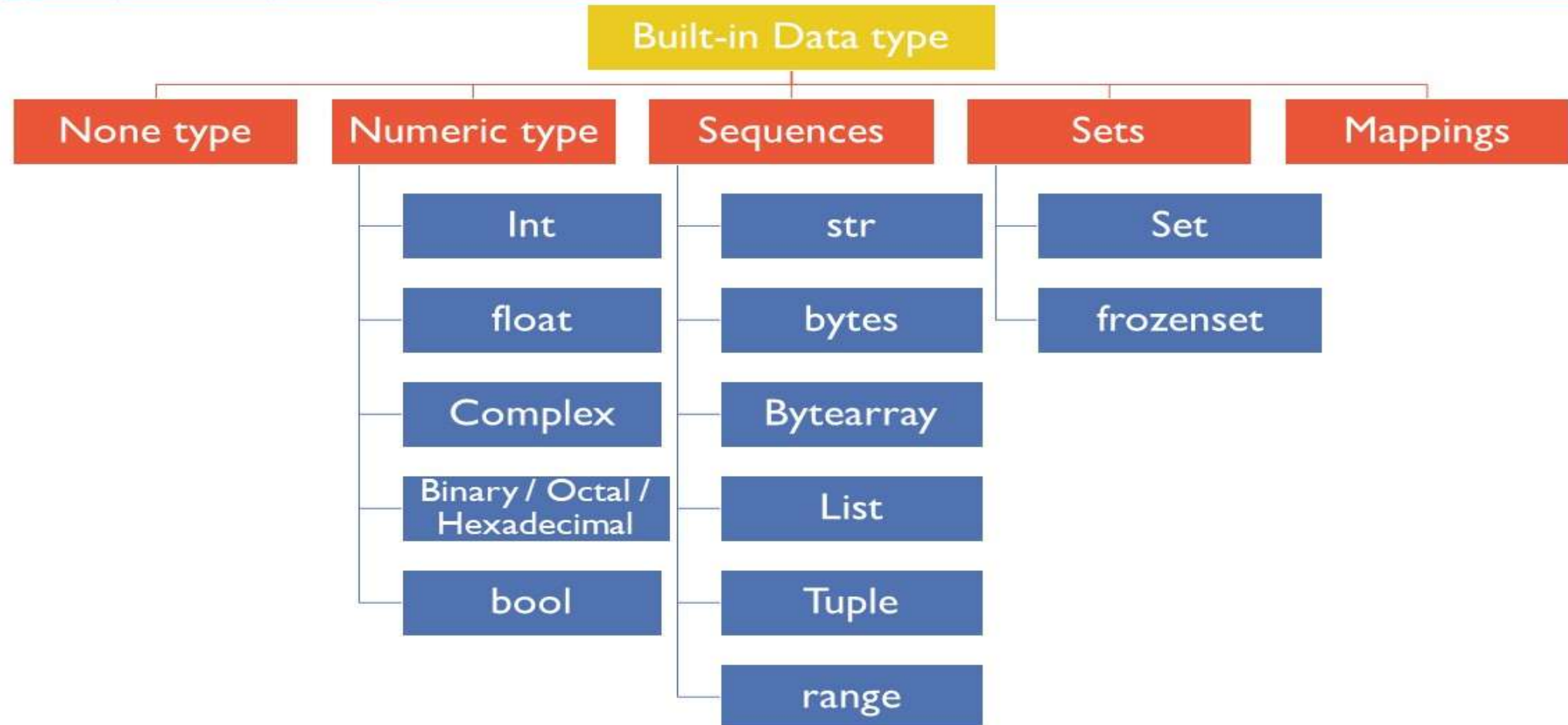
- Indentation means adding white space before a statement.
- Most of the programming languages like C, C++, Java use braces { } to define a block of code.
- Python use indentation to define a block of code.
- Whitespace is used for indentation. All statements with the same distance to the right belong to the same block of code.
- If a block has to be more deeply nested, it is simply indented further to the right.

# Data types of python

- `a=10`
- `b="Hi Python"`
- `c = 10.5`
- `print(type(a))` # int
- `print(type(b))` # str
- `print(type(c))` # float

# Python Data Types

## Datatypes (Built-in) in Python





# Data types in Python

- Numbers (int and float)
  - int (signed integers Ex. 10)
  - long (long integers, they can also be represented in octal and hexa Ex. 0122L)
  - float (floating point real values Ex. 15.20)
  - complex (complex numbers Ex. 3e+26J)
- Sequence Data Type : String (char and string) , List, Tuple
- Dictionary
- Set
- Boolean

# Integer Number

- positive or negative whole numbers without decimal point.
- no limit to how long an integer value.

# Float Number

- real numbers.
- decimal point dividing the integer and fractional parts.
- scientific notation
- E or e indicating the power of 10

# Complex

- Complex numbers are written with a "j" as the imaginary part:
- $x=5+7j$
- `print(x)`
- `print(x.real)`
- `print(x.imag)`
- **Output:**
- $(5+7j)$
- 5.0
- 7.0

# Literals in Python

- String literals :: "hello" , '12345'
- Int literals :: 0,1,2,-1,-2
- Long literals :: 89675L
- Float literals :: 3.14
- Complex literals :: 12j
- Boolean literals :: True or False
- Special literals :: None

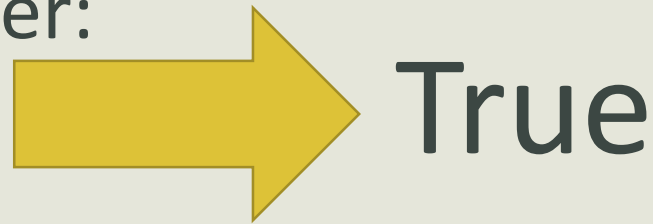


# Representing Binary, Octal and Hexadecimal Numbers

- A binary number should be written by prefixing **0b** or **0B** before the value.  
For example,
  - **0b100011**
  - **0B10100010**
- Hexadecimal numbers are written by prefixing **0x** or **0X** before the value.  
For example,
  - **0xA180**
  - **0X11fb**
- Similarly, Octal numbers are indicated by prefix **0o** or **0O** before the value.  
For example,
  - **0O145**
  - **0o773**

# Boolean

- Booleans represent one of two values: True or False.
- The `bool()` function allows you to evaluate any value, and give you True or False in return.
- Example
- Evaluate a string and a number:
- ```
print(bool("Hello"))  
print(bool(15))
```



# Sequence Type : **String**

- A string is a **sequence of characters**.
- A character is simply a symbol. For example, the English language has 26 characters.
- Computers do not deal with characters, **they deal with numbers (binary)**.
- This conversion of character to a number is called **encoding**, and the reverse process is **decoding**.
- **ASCII and Unicode** are some of the popular encoding used.
- In Python, string is a sequence of **Unicode characters**.
- Unicode was introduced to include every character in all languages and bring uniformity in encoding.

# Sequence Type (String, List, Tuple): String

- The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use **single, double, or triple quotes** to define a string.

```
# all of the following are equivalent
#Creating String using single quotes
my_string = 'Hello'
print(my_string)

#Creating String using double quotes
my_string = "Hello"
print(my_string)

#Creating String using tripple quotes
my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
the world of Python"""
print(my_string)
```

# How to access characters in a **String**?

- We can access **individual characters** using **indexing** and a **range of characters** using **slicing**.
- Index starts from **0**.
- Trying to access a character out of index range will raise an **IndexError**.
- The index must be an **integer**. We can't use float or other types, this will result into **TypeError**.

```
s="atmiya"
print('s = ', s)

#first character
print('s[0] = ', s[0])

#slicing 2nd to 5th character (tmiy)
print('s[1:5] = ', s[1:5])

# index must be in range
print(s[10])          #IndexError: string index out of range

# index must be an integer
print(s[1.5])         #TypeError: string indices must be integers
```



# How to access characters in a String?

- Python allows **negative indexing** for its sequences.
- The index of **-1** refers to the last item, **-2** to the second last item and so on.
- We can access a range of items in a string by using the **slicing operator (colon)**.

```
s="atmiya university"
print('s = ', s)

#last character
print('s[-1] = ', s[-1])

#slicing 7th to 2nd last character
print('s[7:-2] = ', s[7:-2])
```

# Iterating through a string

- Using a for loop we can iterate through a string.
- For example, count number of 'l' in to the string.

```
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count, 'letters found')
```

# String Formatting

- The **format()** method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contain **curly braces {}** as placeholders or replacement fields which get replaced.
- We can use positional arguments or keyword arguments to specify the order.

```
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

# Formatting....

- We can even format strings like the old style used in the C programming language. We use the **% operator** to accomplish this.

```
a=10
b=20.554545454

#Formatting single variable
print("The value of A is %d"%a)

#Formatting multiple variables
print("The value of A is %d and value of B is %f"%(a,b))

#Formatting fractional values
print('The value of x is %.3f' %x) #will display only 3 decimal values
```

# Reversing Strings Through Slicing

- To slice a string, you can use the following syntax:  
**`a_string[start:stop:step]`**
- All the offsets are optional, and they have the following default values:
- **Offset Default Value** **`start 0`** **`stop len(a_string)`** **`step 1`**
- Here, start represents the index of the first character in the slice, while stop holds the index that stops the slicing operation. The third offset, step, allows you to decide how many characters the slicing will jump through on each iteration.

## Example...

```
letters = "AaBbCcDd"
print(letters[::])    # 'AaBbCcDd'
print(letters[::-1])  # 'dDcCbBaA'
print(letters[:])     # 'AaBbCcDd'
print(letters[::2])   # 'ABCD'
print(letters[1::2])  # 'abcd'
```



## Sequence Type (String, List, Tuple): **String**

```
str1 = 'Hello Students' #string str1
```

```
str2 = ' how are you' #string str2
```

```
print (str1[0:2]) #printing first two character using slice operator
```

```
print (str1[4]) #printing 4th character of the string
```

```
print (str1*2) #printing the string twice
```

```
print (str1 + str2) #printing the concatenation of str1 and str2
```

# Sequence Type: List

- 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.
- In Python programming, a list is created by placing all the items (elements) inside a **square bracket [ ]**, separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).
- List items are **ordered, changeable, and allow duplicate values.**

# Creation of LIST

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]

#nested list
my_list = [1,"meera",995556642,[70,75,86,66,48],77.56]
print(my_list)
```

# How to access elements from a List? **Indexing**

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

```
# Output: p
```

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

```
# Output: o
```

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

```
# Output: e
```

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

```
my_list[4.0]      # Error! Only integer can be used for indexing
```

# Indexing with nested list...

```
# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing

# Output: a
print(n_list[0][1])

# Output: 5
print(n_list[1][3])
```

# Negative Indexing...

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

```
# Output: e
```

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

```
# Output: p
```

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



# How to slice lists in python?

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']  
# elements 3rd to 5th  
print(my_list[2:5])  
  
# elements beginning to 4th  
print(my_list[:4])  
  
# elements 6th to end  
print(my_list[5:])  
  
# elements beginning to end  
print(my_list[:])  
  
# reverse elements  
print(my_list[::-1])
```

# How to change or add elements to a List?

- Lists are **mutable**, meaning their elements **can be changed** unlike string or tuple.
- We can use **assignment operator (=)** to change an item or a range of items.

```
my_list = [2, 4, 6, 8]

# change the 1st item
my_list[0] = 1

# Output: [1, 4, 6, 8]
print(my_list)

# change 2nd to 4th items
my_list[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(my_list)
```

## Append and Extend method of list

- We can add one item to a list using the **append()** method or add several items using **extend()** method.

```
my_list = [1, 3, 5]
```

```
my_list.append(7)          # Output: [1, 3, 5, 7]
```

```
print(my_list)
```

```
my_list.extend([9, 11, 13]) # Output: [1, 3, 5, 7, 9, 11, 13]
```

```
print(my_list)
```

## + and \* operators with list

- We can also use + **operator** to combine two lists. This is also called concatenation.
- The \* **operator** repeats a list for the given number of times.

```
my_list = [1, 3, 5]
```

```
print(my_list + [9, 7, 5]) # Output: [1, 3, 5, 9, 7, 5]
```

```
print(["re"] * 3)          #Output: ["re", "re", "re"]
```

## insert( ) with List

- we can insert one item at a desired location by using the method **insert()** or insert multiple items by squeezing it into an empty slice of a list.

```
my_list = [1, 9]
#insert(position, element)
my_list.insert(1,3)      # Output: [1, 3, 9]
print(my_list)

my_list[2:2] = [5, 7]    # Output: [1, 3, 5, 7, 9]
print(my_list)
```

# How to delete or remove elements from a List?

- We can delete one or more items from a list using the keyword `del`. It can even delete the list entirely.

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

# delete one item
del my_list[2]
print(my_list)           # Output: ['p', 'r', 'b', 'l', 'e', 'm']

# delete multiple items
del my_list[1:5]
print(my_list)           # Output: ['p', 'm']

# delete entire list
del my_list
print(my_list)           # Error: List not defined
```



# remove, pop and clear methods

- We can use **remove()** method to remove the given item or **pop()** method to remove an item at the given index.
- The **pop()** method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).
- We can also use the **clear()** method to empty a list.

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

# Removing an element
my_list.remove('p')
print(my_list)          # Output: ['r', 'o', 'b', 'l', 'e', 'm']

# Remove element 'o' from list
print(my_list.pop(1))
print(my_list)          # Output: ['r', 'b', 'l', 'e', 'm']

# Remove last element from a list
print(my_list.pop())
print(my_list)          # Output: ['r', 'b', 'l', 'e']

# Clearing list
my_list.clear()
```

# List Length

- To determine how many items a list has, use the len() function:
- **Example**
- Print the number of items in the list:
  - `thislist = ["apple", "banana", "cherry"]`
  - `print(len(thislist))`
- **Output: 3**

# Iteration of List

```
#List Iteration
fruits = ['apple', 'banana', 'mango']:
for i in fruits:
    print("I like",i)
```

# List Methods

| Method          | Description                                                |
|-----------------|------------------------------------------------------------|
| <b>append()</b> | Add an element to the end of the list                      |
| <b>extend()</b> | Add all elements of a list to the another list             |
| <b>insert()</b> | Insert an item at the defined index                        |
| <b>remove()</b> | Removes an item from the list                              |
| <b>pop()</b>    | Removes and returns an element at the given index          |
| <b>clear()</b>  | Removes all items from the list                            |
| <b>index()</b>  | Returns the index of the first matched item                |
| <b>count()</b>  | Returns the count of number of items passed as an argument |

# Methods...

| Method           | Description                                                      |
|------------------|------------------------------------------------------------------|
| <b>sort()</b>    | Sort items in a list in ascending order                          |
| <b>reverse()</b> | Reverse the order of items in the list                           |
| <b>copy()</b>    | Returns a shallow copy of the list                               |
| <b>len()</b>     | Return the length (the number of items) in the list.             |
| <b>list()</b>    | Convert a collection (tuple, string, set, dictionary) to a list. |
| <b>max()</b>     | Return the largest item in the list.                             |
| <b>min()</b>     | Return the smallest item in the list                             |
| <b>sum()</b>     | Return the sum of all elements in the list.                      |

# Tuple

- A tuple is a collection which is **ordered, unchangeable and allows duplicate values**.
- Tuples are used to store multiple items of any type like int, float and string in a single variable.
- Tuples are written with round brackets.
- Example :Create a Tuple
  - `thistuple = ("apple", "banana", "cherry")`
  - `print(thistuple)`
  - Output :- ('apple', 'banana', 'cherry')



# Tuple...

```
# empty tuple
my_tuple = ()
print(my_tuple)

# tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)           # Output: (1, 2, 3)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)           # Output: (1, "Hello", 3.4)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)           # Output: ("mouse", [8, 4, 6], (1, 2, 3))
```

# Tuple...

```
# only parentheses is not enough
my_tuple = ("hello")
print(type(my_tuple))           # Output: <class 'str'>

# need a comma at the end
my_tuple = ("hello",)
print(type(my_tuple))           # Output: <class 'tuple'>

# parentheses is optional
my_tuple = "hello",
print(type(my_tuple))           # Output: <class 'tuple'>
```

# Tuple...

```
# nested tuple  
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
```

```
# nested index  
print(n_tuple[0][3])           # Output: 's'
```

```
# nested index  
print(n_tuple[1][1])           # Output: 4
```

```
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')
```

```
print(my_tuple[-1])            # Output: 't'
```

```
print(my_tuple[-6])            # Output: 'p'
```

# Indexing and Slicing

```
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')

print(my_tuple[0])          # Output: 'p'
print(my_tuple[5])          # Output: 't'

# index must be in range
print(my_tuple[6])          # IndexError: list index out of range
print(my_tuple[7:])          # Output: ('i', 'z')

# elements beginning to end
print(my_tuple[:])
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

# Iterating through Tuple

```
#Iterating Tuple  
for name in ('John', 'Kate'):  
    print("Hello", name)
```

# Tuples are **immutable**

- This means that elements of a tuple **cannot be changed** once it has been assigned.
- We can also assign a tuple to different values (reassignment).

```
my_tuple = (4, 2, 3, [6, 5])

# we cannot change an element
my_tuple[1] = 9 # TypeError: 'tuple' object does not support item assignment

# but item of mutable element can be changed
my_tuple[3][0] = 9
print(my_tuple)          # Output: (4, 2, 3, [9, 5])

# tuples can be reassigned
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple)
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```



# Deleting a tuple

- As discussed above, we cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
- But deleting a tuple entirely is possible using the keyword `del`.

```
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items
del my_tuple[3] # TypeError: 'tuple' object doesn't support item deletion

# can delete entire tuple
del my_tuple
print(my_tuple) # NameError: name 'my_tuple' is not defined
```

# Use of + and \* with tuple

- We can use + operator to combine two tuples (concatenation).
- We can also repeat the elements in a tuple for a given number of times using the \* operator.
- Both + and \* operations result into a new tuple.

```
# Concatenation
```

```
print((1, 2, 3) + (4, 5, 6))    # Output: (1, 2, 3, 4, 5, 6)
```

```
# Repeat
```

```
print(("Repeat",) * 3)          # Output: ('Repeat', 'Repeat', 'Repeat')
```

# Tuple Membership Test : IN and NOT IN

```
# Python program to show how to perform membership test for tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Ordered")
# In operator
print('Tuple' in tuple_)
print('Items' in tuple_)
# Not in operator
print('Immutable' not in tuple_)
print('Items' not in tuple_)
_
```

True

False

False

True

# Methods of Tuple

| Method          | Description                                                     |
|-----------------|-----------------------------------------------------------------|
| <b>index(x)</b> | Return index of first item that is equal to x                   |
| <b>count(x)</b> | Return the number of items that is equal to x                   |
| <b>len()</b>    | Return the length (the number of items) in the tuple.           |
| <b>tuple()</b>  | Convert an iterable (list, string, set, dictionary) to a tuple. |
| <b>max()</b>    | Return the largest item in the tuple.                           |
| <b>min()</b>    | Return the smallest item in the tuple                           |
| <b>sum()</b>    | Return the sum of all elements in the tuple.                    |

# Methods...

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)
# Count()
print(my_tuple.count('p'))           # Output: 2

# Index()
print(my_tuple.index('l'))           # Output: 3
```

# Range

- The range data type represents a sequence of numbers.
- The numbers in the range are not modifiable.
- Generally, range is used for repeating a for loop for a specific number of times.

```
#creating a range
```

```
r=range(10)
```

```
for i in r:
```

```
    print(i)
```

```
#Output:[0 1 2 3 4 5 6 7 8 9]
```



# Range...

- The range () has three arguments:

**range(start, stop, step)**

- **Start** : it is an optional argument that represents the starting value of range. Default value is 0
- **Stop** : represents the ending value of range ( excluding).
- **Stepsize** : it is an optional argument that represents the increment between each number in a range. Default value is 1.you can provide negative stepsize to reverse a range.

# Set

- A set is an **unordered collection of elements** much like sets in Mathematics.
- The order of elements is not maintained in the sets.
- It means the elements may not appear in the same order as they are entered into the set.
- Moreover, a set **does not accept duplicate elements**
- There are two subtypes in sets:
  - **set**
  - **frozenset**

# Set....

- To create a set, we should enter the elements separated by commas inside **curly braces { }**.

```
#creating a set
s={10,20,30,40,50,40}
print(s)           #Output : {50, 20, 40, 10, 30}
```

- Since sets are unordered, we cannot retrieve the elements using indexing and slicing operations.

```
s={10,20,30,40,50}
print(s[3])        #Output : TypeError: 'set' object is not subscriptable
print(s[0:3])      #Output : TypeError: 'set' object is not subscriptable
```

# Update and Remove method with Set

- The **update()** method is used to add elements to a set.
- The **remove()** method is used to remove any particular element from a set.

```
s = {10, 20, 30, 40, 50, 60}

#Adding new elements in a set
s.update([90, 100])
print(s)          #Output : {50, 20, 90, 100, 40, 10, 60, 30}

#removing elements from a set
s.remove(30)
print(s)          #Output : {50, 20, 90, 100, 40, 10, 60}
```

# frozenset Datatype

- The frozenset datatype is the same as the set datatype.
- The main difference is that the elements in the set can be modified; whereas, the elements of the frozenset **cannot be modified**.
- We can create a frozenset by passing a set to **frozenset()**.

```
s = {10, 20, 30, 40, 50, 60}
fs=frozenset(s)
print(fs)    #Output: frozenset({50, 20, 40, 10, 60, 30})
```

# Dictionary

- The data is stored as key-value pairs using a Python dictionary is called **Mapping**
  - This data structure is mutable
  - The components of dictionary were made using keys and values.
  - Keys must only have one component.
  - Values can be of any type, including integer, list, and tuple.
- The key and its value should be separated **by a colon (:) and every pair should be separated by a comma.**
- All the elements should be enclosed inside **curly brackets { }.**



# Dictionary

```
#Creating an empty dictionary
```

```
my_dict={}
```

```
#Creating dictionary with elements
```

```
my_dict={101:"Meera", 102:"Disha", 103:"Falguni", 104:"Vaishali"}
```

```
print(my_dict)
```

```
#Output : {101: 'Meera', 102: 'Disha', 103: 'Falguni', 104: 'Vaishali'}
```

```
#Retrieve values of key 102
```

```
print(my_dict[102])      #Output:Disha
```

# Add new element to dictionary

```
#Creating dictionary with elements
```

```
my_dict={101:"Meera", 102:"Disha", 103:"Falguni", 104:"Vaishali"}
```

```
#Adding New value to dictionary
```

```
my_dict[105]="Kamal"
```

```
print(my_dict)
```

```
#Output: {101: 'Meera', 102: 'Disha', 103: 'Falguni', 104: 'Vaishali', 105: 'Kamal'}
```

# Change the value in to the dictionary

```
#Creating dictionary with elements
my_dict={101:"Meera", 102:"Disha", 103:"Falguni", 104:"Vaishali"}

#Update the old value in dictionary
print("Before Updation:",my_dict)
#Output: {101: 'Meera', 102: 'Disha', 103: 'Falguni', 104: 'Vaishali'}

my_dict[101]="Khyati"

print("After Updation:",my_dict)
#Output: {101: 'Khyati', 102: 'Disha', 103: 'Falguni', 104: 'Vaishali'}
```

# Delete the element from Dictionary

```
#Creating dictionary with elements
my_dict={101:"Meera", 102:"Disha", 103:"Falguni", 104:"Vaishali"}

#Update the old value in dictionary
print("Before Deletion:",my_dict)
#Output: {101: 'Meera', 102: 'Disha', 103: 'Falguni', 104: 'Vaishali'}

del my_dict[101]

print("After Deletion:",my_dict)
#Output: {102: 'Disha', 103: 'Falguni', 104: 'Vaishali', 105: 'Kamal'}
```

# Setting Data Type

- `x = int(20)`
- `x = float(20.5)`
- `x = complex(1j)`
- `x = list(("apple", "banana", "cherry"))`
- `x = tuple(("apple", "banana", "cherry"))`
- `x = dict(name="John", age=36)`
- `x = set(("apple", "banana", "cherry"))`
- `x = bool(5)`

# Python Data Types

- Here are four collection data types in the Python programming language:
- **List** is a collection which is **ordered, changeable** and **allows duplicate members**.
- **Tuple** is a collection which is **ordered, unchangeable** and **allows duplicate members**.
- **Set** is a collection which is **unordered, unchangeable, unindexed** and **No duplicate members**.
- **Dictionary** is a collection which is **ordered and changeable** and **No duplicate members**.



# Simple Output – `print()`

- `Print()` -Output data to the standard output device (screen).
- Example:

```
print("Atmiya University-Rajkot")
```

Output:- Atmiya University-Rajkot

# print() Example

```
a = 5
```

```
print('The value of a is', a)      Output:-The value of a 5
```

```
x=12.34574
```

```
print('The value of x is %3.2f',x)      Output: The value of x is 12.35
```

```
print('The value of x is %3.4f',x)      Output: The value of x is 12.3457
```

# input function

- `input()`:- to allow take the input from the user at run-time.
- Syntax of `input()`: `input([prompt])`
- Example:
- `num = input('Enter a number: ')`
- `print("Number is:",num)`
- Output: **Enter a number:10**
- **Number is:10**

# input()....

- # two input 2 no and calculate sum
- a=int(input("enter first no..."))
- print(a)
- b=int(input("enter second no..."))
- print(b)
- c=a+b
- print("sum is",c)

# Operators

- An operator is a symbol that performs an operation.
- An Operator acts on some variables called **operands**.
- If an operator acts on a single variable it is called a **unary operator**.
- if an operator acts on two variables, it is called a **binary operator**.
- If an operator acts on three variables, then it is called **ternary operator**.

# Python Basic Operators

## ■ Types of Operator

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Logical Operators
- Boolean Operators
- Bitwise Operators
- Identity Operator
- Membership Operator



# Arithmetic Operators

- Assume variable a holds 13 and variable b holds 5, then

| Operator          | Description                                       | Example         |
|-------------------|---------------------------------------------------|-----------------|
| + Addition        | Addition of number                                | $a + b = 18$    |
| - Subtraction     | Subtracts of number                               | $a - b = 8$     |
| * Multiplication  | Multiplication of number                          | $a * b = 65$    |
| / Division        | Division of number                                | $b/a = 2.6$     |
| % Modulus         | Found the modulus                                 | $b \% a = 3$    |
| // Floor Division | Perform the division and return answer in integer | $b//a = 2$      |
| ** Exponent       | Calculates exponential power value.               | $a**b = 371293$ |

# Assignment Operators

| Operator | Example                                          |
|----------|--------------------------------------------------|
| =        | <code>c = a + b</code>                           |
| +=       | <code>c += a</code> ( <code>c = c + a</code> )   |
| -=       | <code>c -= a</code> ( <code>c = c - a</code> )   |
| *=       | <code>c *= a</code> ( <code>c = c * a</code> )   |
| /=       | <code>c /= a</code> ( <code>c = c / a</code> )   |
| **=      | <code>c **= a</code> ( <code>c = c ** a</code> ) |
| //=      | <code>c //= a</code> ( <code>c = c // a</code> ) |

# Relational Operators

- For ex: a=10 and b= 20

| Operator | Example              |
|----------|----------------------|
| ==       | (a == b) is not true |
| !=       | (a != b) is true     |
| >        | (a > b) is not true  |
| <        | (a < b) is true      |
| >=       | (a >= b) is not true |
| <=       | (a <= b) is true     |

# Logical Operators

- For ex: a=10 and b= 20

| Operator | Example                |
|----------|------------------------|
| and      | (a and b) is true.     |
| or       | (a or b) is true       |
| not      | not(a and b) is false. |

# Bitwise Operators

| Operator | Name                | Description                                                                                              | Syntax       |
|----------|---------------------|----------------------------------------------------------------------------------------------------------|--------------|
| &        | Bitwise AND         | Result bit 1,if both operand bits are 1;otherwise results bit 0.                                         | $x \& y$     |
|          | Bitwise OR          | Result bit 1,if any of the operand bit is 1; otherwise results bit 0.                                    | $x   y$      |
| ~        | Bitwise NOT         | inverts individual bits                                                                                  | $\sim x$     |
| ^        | Bitwise XOR         | Results bit 1,if any of the operand bit is 1 but not both, otherwise results bit 0.                      | $x \wedge y$ |
| >>       | Bitwise right shift | The left operand's value is moved toward the right by the number of bits specified by the right operand. | $x >>$       |
| <<       | Bitwise left shift  | The left operand's value is moved toward the left by the number of bits specified by the right operand.  | $x <<$       |

# Python Identity Operators

- Identity operators compare the memory locations of two objects. There are two Identity operators explained below

| Operator | Description                                                                                                     | Example                                                              |
|----------|-----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| is       | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here is results in 1 if id(x) equals id(y).                  |
| is not   | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise  | x is not y, here is not results in 1 if id(x) is not equal to id(y). |



# Python Membership Operators

- Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below

| Operator | Description                                                                                      | Example                                                                    |
|----------|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| in       | Evaluates to true if it finds a variable in the specified sequence and false otherwise..         | x in y, here in results in a 1 if x is a member of sequence y.             |
| not in   | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |