UNIT-1 Introduction: Introduction to Python, Program Development Cycle, Input, Processing, and Output, Displaying Output with the Print Function, Comments, Variables, Reading Input from the Keyboard, Performing Calculations, Operators, Type conversions, Expressions, More about Data Output. Data Types, and Expression: Strings Assignment, and Comment, Numeric Data Types and Character Sets, Using Functions and Modules. Decision Structures and Boolean Logic: If, If-else, If-elif-else Statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables. Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.

Introduction to Python:

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions: Python 2 and Python 3. Both are quite different.

Beginning with Python programming:

1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. http://ideone.com/ or http://codepad.org/ that can be used to run Python programs without installing an interpreter.

Windows: There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from http://python.org/.

Linux: Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check which version of Python you're running, type "python" in the terminal emulator. The interpreter should start and print the version number.

macOS: Generally, Python 2.7 comes bundled with macOS. You'll have to manually install Python 3 from http://python.org/.

2) Writing our first program:

Just type in the following code after you start the interpreter.

```
# Script Begins

print("welcome to Technical World")

# Scripts Ends
```

Output: welcome to Technical World

Let's analyze the script line by line.

Line 1: [# Script Begins] In Python, comments begin with a #. This statement is ignored by the interpreter and serves as documentation for our code.

Line 2: [print("welcome to Technical World")]To print something on the console, print() function is used. This function also adds a newline after our message is printed(unlike in C). Note that in Python 2, "print" is not a function but a keyword and therefore can be used without parentheses. However, in Python 3, it is a function and must be invoked with parentheses.

Line 3: [# Script Ends] This is just another comment like in Line 1.

Applications:

1. **Web development** – Web framework like Django and Flask are based on Python. They help you write server side code which helps you manage database, write backend programming logic, mapping urls etc.

2. **Machine learning** – There are many machine learning applications written in Python. Machine learning is a way to write a logic so that a machine can learn and solve a particular problem on its own. For example, products recommendation in websites like Amazon, Flipkart, eBay etc. is a machine learning algorithm that recognises user's interest. Face recognition and Voice recognition in your phone is another example of machine learning.

3. **Data Analysis** – Data analysis and data visualisation in form of charts can also be developed using Python.

4. **Scripting** – Scripting is writing small programs to automate simple tasks such as sending automated response emails etc. Such type of applications can also be written in Python programming language.

5. **Game development** – You can develop games using Python.

6. You can **develop Embedded applications** in Python.

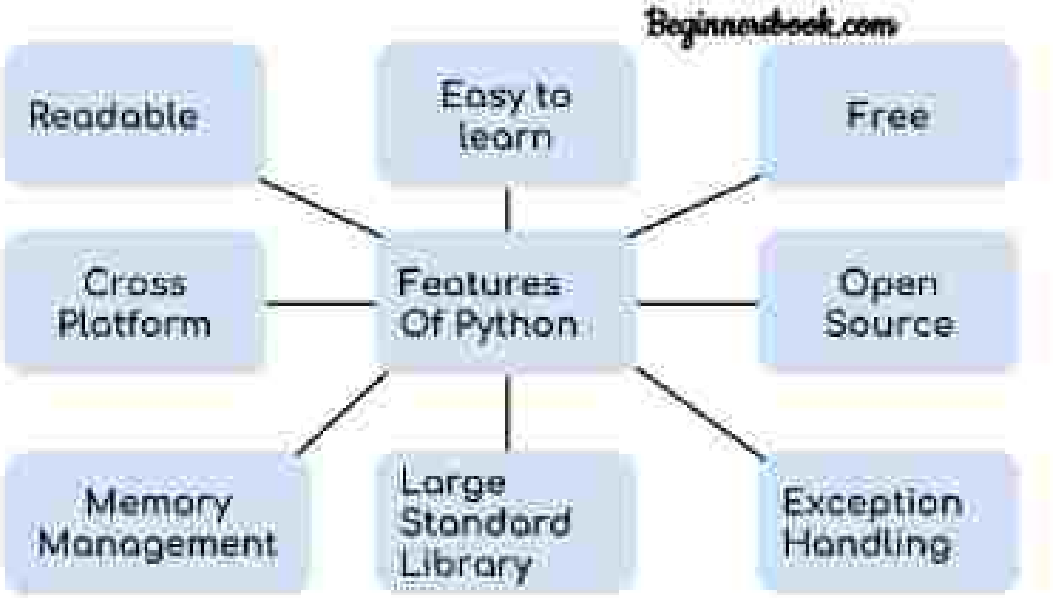
7. **Desktop applications** – You can develop desktop application in Python using library like TKinter or QT.

Organizations using Python :

1) Google(Components of Google spider and Search Engine)
2) Yahoo(Maps)
3) YouTube
4) Mozilla
5) Dropbox
6) Microsoft
7) Cisco
8) Spotify
9) Quora

Features of Python:

1. Simple and easy to learn.
2. Free ware and Open source.
3. High level programming language.
4. Python id platform independent.
5. Portability.
6. Dynamically typed.
7. Both procedure oriented and object oriented.
8. Interpreted programming language.
9. Extensible.
10. Embedded.
11. Extensive library.



1. **Readable:** Python is a very readable language.

2. **Easy to Learn:** Learning python is easy as this is a expressive and high level programming language, which means it is easy to understand the language and thus easy to learn.

3. **Cross platform:** Python is available and can run on various operating systems such as Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.

4. **Open Source:** Python is a open source programming language.

5. **Large standard library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in Python.

6. **Free:** Python is free to download and use. This means you can download it for free and use it in your application. Python is an example of a FLOSS (Free/Libre Open Source Software), which means you can freely distribute copies of this software, read its source code and modify it.

7. **Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program exception and can disrupt the normal flow of program. Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.

8. **Advanced features:** Supports generators and list comprehensions.

9. **Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

Limitations of Python:

1. Performance is not up to the mark.
2. For mobile applications it is not up to the mark.

Flavours of python:

1. Cpython
2. Jython (or) Jpython.
3. Ironpython.
4. Pypy.
5. Rubypython.
6. Anaconda python.
7. Stackless

How to install Python:

Python installation is pretty simple, you can install it on any operating system such as Windows, Mac OS X, Ubuntu etc. Just follow the steps

> **Local Environment Setup:** Open a terminal window and type "python" to find out if it is already installed and which version is installed.
>
> **Getting Python for Windows platform:** Binaries of latest version of Python 3 (Python 3.6.5) are available in https://www.python.org

The following different installation options are available.

- Windows x86-64 embeddable zip file
- Windows x86-64 executable installer
- Windows x86-64 web-based installer
- Windows x86 embeddable zip file
- Windows x86 executable installer
- Windows x86 web-based installer

Download the software and save to hard drive. Double click on the install file and install software as per instructions. By default, IDLE will install in to the system as default IDE (Integrated development Environment). After installing software it is very important to set the path if path is not set by default. Above 3 versions are not necessary to set the path. It will set automatically by default.

Setting Path at Windows:

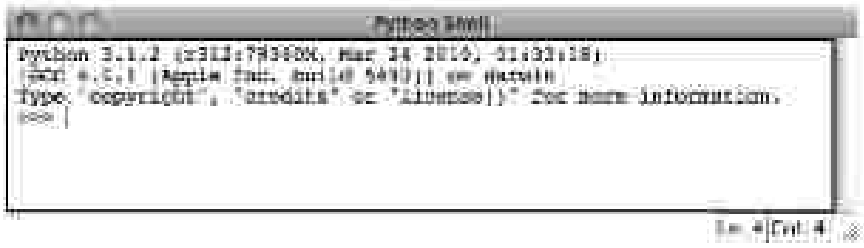
To add the Python directory to the path for a particular session in Windows-

At the command prompt: type path **%path%; C:\Python** and press Enter.

Note: C:\Python is the path of the Python directory.

Running Code in the Interactive Shell

Python is an interpreted language, and you can run simple Python expressions and statements in an interactive programming environment called the shell. The easiest way to open a Python shell is to launch the **IDLE**(Integrated Development and Learning Environment) comes with the Python installation. When you do this, a window named Python Shell opens. Figure 1.6 shows a shell window on Mac OS X. A shell window running on a Windows system or a Linux system should look similar, if not identical, to this one.



A shell window contains an opening message followed by the special **symbol>>>**, called a **shell prompt.** The cursor at the shell prompt waits for you to enter a Python command. Note that you can get immediate help by entering help at the shell prompt or selecting Help from the window's drop-down menu. When you enter an expression or statement, Python evaluates it and displays its result, if there is one, followed by a new prompt.

To quit the Python shell, you can either select the window's close box or press the **Control + D** key combination.

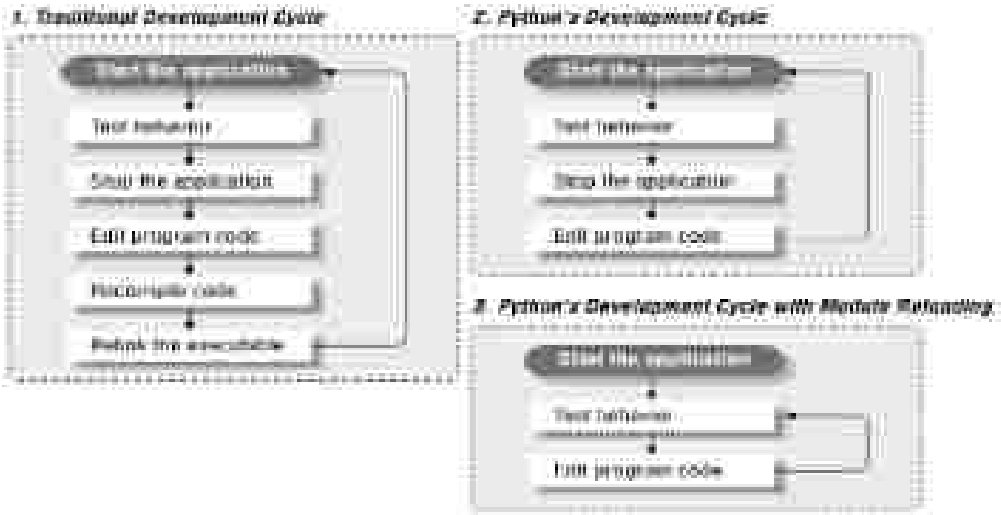
Command Interpreter vs IDLE:

Most of the professionals are use default IDE i.e IDLE to work with python environment. There are Some of the key features of IDLE it offers are:

- Python shell with syntax highlighting.
- Multi-window text editor.
- Code auto completion.
- Intelligent indenting.
- Program animation and stepping which allows one line of code to run at a time helpful for debugging.
- Persistent breakpoints.
- Finally, Call stack visibility.

Program Development Life cycle

- Program Development Life Cycle (PDLC) is a systematic way of developing quality software. It provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase.
- Python's development cycle is dramatically shorter than that of traditional tools as shown in figure



- In Python, there are no compile or link steps -- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made and in cases where **dynamic** module reloading can be used, it's even possible to change and reload parts of a running program without stopping it at all.

Variables

- Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

Assigning Values to Variables

- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example :

```
counter =100          # An integer assignment
miles  =1000.0        # A floating point
name   ="John"        # A string

print(counter)
print(miles)
print(name)
```

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result –

Output:

```
100
1000.0
'John'
```

Multiple Assignments: Python allows you to assign a single value to several variables simultaneously.

For example: a = b = c = 1

Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

For example: a, b, c = 1, 2, "john"

Here, two integer objects with values 1 and 2 are assigned to the variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Python Comments

Comments are descriptions that help programmers better understand the intent and functionality of the program. It is completely ignored by the Python interpreter.

Single-Line Comments in Python: In Python, we use the hash symbol # to write a single-line comment.

Example 1: Writing Single-Line Comments

```
# printing a string
print('Hello world')
```

Output: Hello world

Here, the comment is: # printing a string. This line is ignored by the Python interpreter. Everything that comes after # is ignored. So, we can also write the above program in a single line as:

```
print('Hello world')   #printing a string
```

The output of this program will be the same as in Example 1. The interpreter ignores all the text after #.

Multi-Line Comments in Python: We can use # at the beginning of each line of comment on multiple lines.

Example 2: Using multiple #

```
# it is a multiline
# comment
```

Here, each line is treated as a single comment and all of them are ignored.

In a similar way, we can use multiline strings (triple quotes) to write multiline comments as shown below

The quotation character can either be ' or ".

```
'''
I am a
multiline comment!
'''
print("Hello World")
```

Input, Processing, and Output

- Most useful programs accept inputs from some source, process these inputs, and then finally output results to some destination.
- In terminal-based interactive programs, the input source is the keyboard, and the output destination is the terminal display.
- The Python shell itself takes inputs as Python expressions or statements. Its processing evaluates these items. Its outputs are the results displayed in the shell.
- Python provides numerous built-in functions that are readily available to us at the Python prompt.

- Some of the functions like input() and print() are widely used for standard input and output operations respectively.

Let us see the output section first. : Python Output Using print () function

- We use the print() function to output data to the standard output device (screen). We can also output data to a file.

print() function: The print() function prints the given object to the standard output device (screen) or to the text stream file.

syntax of print() is:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- **objects** - object to the printed. * indicates that there may be more than one object
- **sep** - objects are separated by sep. **Default value** : ' '
- **end** - end is printed at last by **default it has \n**
- **file** - must be an object with write(string) method. If omitted it, sys.stdout will be used which prints objects on the screen.
- **flush** - A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Example 1:

```
print('This sentence is output to the screen')
```

Output: This sentence is output to the screen

Example 2:

```
a = 5
print('The value of a is', a)
```

Output: The value of a is 5

Example 3:

```
print("Python is fun.")
a = 5
# Two objects are passed
print("a =", a)
b = a
# Three objects are passed
print('a =', a, '= b')
```

Output:

Python is fun.
a= 5
a = 5 = b

Case 1: print without any arguments

```
print('Tirumala Engineering College')
print () # we have not passed any argument by default it takes new line and create one line blank space
print('Computer Science Engineering')
```

Output:

Tirumala Engineering College

Computer Science Engineering

Case 2: print function with string operations (with arguments)

1.Print('helloworld') **Output:** helloworld

2.print('hello\nworld') **Output:** hello
world

3.print('hello\tworld') **Output:** hello world

4. # string concatenation both objects are string only

print('tec' + 'cse') **Output:** teccse

print('tec' , 'cse') **Output:** tec cse

5.# repeat string into number of times

print(5*'cse') **Output:** csecsecsecsecse

print(5*'cse\n')

Output:
cse
cse
cse
cse
cse

cse

print(5*'cse\t') **Output:** cse cse cse cse cse

Case 3 print function with any number of arguments

1. (a) print('values are :',10,20,30) **Output:** values are : 10 20 30

   (b) a,b,c=10,20,30

   print('values are:',a,b,c)

Output: values are : 10 20 30

2. **print function with "sep" attribute: This is used to separate objects**

   By default, value of sep is empty space(sep='')

   >>>print('values are:',10,20,30,sep=":")

   Output: values are::10:20:30

   >>>print('values are:',10,20,30,sep="--")

   Output: values are:--10--20--30

Case 4: print statement with end attribute

- The **end** key of print function will set the string that needs to be appended when printing is done.
- By default, the **end** key is set by newline character (By default, attribute end ='\n' in print function). So after finishing printing all the variables, a newline character is appended. Hence, we get the output of each print statement in different line. But we will now overwrite the newline character by any character at the end of the print statement.

Example-1:

- print('Tirumala') # in this statement by default end ='\n' so it takes new line
- print('rajani') # in this statement by default end ='\n' so it takes new line
- print('devansh') # in this statement by default end ='\n' so it takes new line

Output:

Tirumala

rajani

devansh

Note: when you observe output 1st print statement prints output Tirumala and immediately takes new line character and execute 2nd print statement and followed.

Example-2:

- print ('Tirumala', end='$')
- print ('rajani' ,end='*')
- print('devansh')

Output:

Tirumala$rajani*devansh

Case 5: print function with sep and end attribute

Example

- print(19,20,30,sep=':',end='$$$')
- print(40,50,sep=':')
- print(70,80,sep=':',end='&&&')
- print(90,100)

Output: 19:20:30$$$40:50

70:80&&&90 100

Python | Output Formatting

- There are several ways to present the output of a program, data can be printed in a human-readable form, or written to a file for future use. Sometimes user often wants more control the formatting of output than simply printing space-separated values. There are several ways to format output.

1. **Formatting output using String modulo operator (%):**

Syntax: print ('formatted string' %(variable list)

The % operator can also be used for string formatting. string modulo operator (%) is still available in Python(3.x) and user is using it widely.

Example 1

```
# Python program showing how to use string modulo operator (%) to print fancier output

# print integer and float value

        print ("CSE: % 2d, Portal: % 5.2f"%(1, 05.333))

# print integer value

        print ("Total students: % 3d, Boys: % 2d"%(240, 120))

# print octal value

        print ("% 7.3o"%(25))

# print exponential value

        print ("% 10.3E"%(356.08977))
```

Output :

```
CSE:  1, Portal: 5.33
Total students: 240, Boys: 120
031
3.561E+02
```

Example 2

(a) a=6

```
print('a value is =%i' %a)
```

Output: a value is =6

(b) a=6;b=7;c=8

```
print('a value is =%i and b=%f and c=%i' %(a,b,c))
```

Output: a value is =6 and b=7.000000 and c=8

2. print function with replacement operator {}or format function

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.

Example:1

```
Name= 'John'
Salary='1000'
print('hello my name is {} and my salary is {}'.format(Name,Salary))
```

Output: hello my name is John and my salary is 1000

Example:2

```
name='John'
salary=1000
print('hello my name is "{}" and my salary is "{}"'.format(name,salary))
```

Output: : hello my name is "John" and my salary is "1000"

***********we can also use index values and print the output***********

Example: 3

```
name='John'
salary=1000
print('hello my name is "{0}" and my salary is "{1}"'.format(name,salary))
```

Output: hello my name is "John" and my salary is "1000"

Example: 4

```
print('hello my name is "{0}" and my salary is "{1}"'.format(salary,name))
```

Output: hello my name is "1000" and my salary is "John"

******* **** we can also use variables in the reference operator***********

Example: 5

```
print ('hello my name is "{n}" and my salary is "{s}"'. format (n=name, s=salary))
```

Output: hello my name is "John" and my salary is "1000"

Example: 6

```
print ('hello my name is "{n}"and my salary is { s}'.format(s=salary,n=name))
```

Output: hello my name is "John" and my salary is "1000"

Reading Input from the Keyboard: In python input() is used to read the input from the keyboard dynamically. By default, the value of the input function will be stored as a string.

Syntax: Variable name =input('prompt')

Example:

```
name=input("Enter Employee Name ")
salary =input("Enter salary ")
company =input("Enter Company name ")
print("\n")
print("Printing Employee Details")
print("Name","Salary","Company")
print(name, salary, company)
```

Output:

```
Enter Employee Name Jon
Enter salary 12000
Enter Company name Google

Printing Employee Details
Name Salary Company
Jon 12000 Google
```

Accept an numeric input from User: To accept an integer value from a user in Python. We need to convert an input string value into an integer using a int() function.

Example: first_number = int(input("Enter first number "))

We need to convert an input string value into an integer using a float() function.

Example: first_number = float(input("Enter first number "))

```
# program to calculate addition of two input numbers

first_number=int(input("Enter first number "))
second_number=int(input("Enter second number "))
print("First Number:",first_number)
print("Second Number:",second_number)
sum1 =first_number+second_number
print("Addition of two number is: ", sum1)
```

Output:

```
Enter first number 20
Enter second number 40
First Number: 20
Second Number: 40
Addition of two number is:  60
```

Get multiple input values from a user in one line: In Python, we can accept two or three values from the user in one input() call.

Example: In a single execution of the input() function, we can ask the user hi/her name, age, and phone number and store it in three different variables.

```
name, age, phone =input("Enter your name, Age, Percentage separated by space ").split()
print("\n")
print("User Details: ", name, age, phone)
```

Output:

```
Enter your name, Age, Percentage separated by space John 26 75.50
User Details:  John 26 75.50
```

Performing Calculations

Computers are great at math problems! How can we tell Python to solve a math problem for us? In this we use numbers in Python and the special symbols we use to tell it what kind of **calculation** to do.**Example-1:**

```
1. print(2+2)
2. print("2"+"2")
```

Output:

```
4
22
```

Question: Why does line two give the wrong Answer?

Answer: When we do math in Python, we can't use strings. We have to use numbers. The first line uses two numbers. Both of them are **integers** (called int in Python).

Example-2:

```
Subtraction:
print(2 - 2)
Multiplication:
print(2 * 2)
```

Division:
print(2 / 2)

Question: Why did the last statement output 1.0?

Answer: When Python does division, it uses a different kind of number called a **float**. Floats always have a decimal point. Integers are always whole numbers and do not have decimal points.

Run:
print(7/2)

Calculations in Python follow the Order of Operations, which is sometimes called **PEMDAS**.

Run:
print((6 - 2) * 5)
print(6 - 2 * 5)

Subtraction:
print(2 - 2)
Multiplication:
print(2 * 2)
Division:
print(2 / 2)

Example-3:

The first statement is **evaluated** by Python like this:

1. (6 – 2) * 5 *Parentheses first*
2. 4 * 5
3. 20

The second statement is evaluated like this:

1. 6 – 2 * 5 *Multiplication first*
2. 6 – 10
3. -4

Example-4:
The **modulo** operator (%) finds the remainder of the first number divided by the second number.

Run:
print(12 % 10)
12 / 10 = 1 with a remainder of 2.

Integer division (//) is like normal division, but it rounds down if there is a decimal point.

Run:
print(5 // 2)
5 / 2 = 2.5, which is rounded down to 2

Exponentiation (**) raises the first number to the power of the second number.

Run:
print(3 ** 2)
This is the same as 3^2

Example-5:

Remember, input() **returns** a string, and we can't do math with strings. Fortunately, we can change strings into ints like so:

two = "2"
two = int(two)
print(two + two)

If you need to work with a decimal point, you can change it to a float instead:

two = float(two)

Example-6:

Activity 1:

Do you know anyone who tends to one-up you in conversation? In this activity, we'll make a simple chatbot that asks a series of questions, explaining to you why it's superior after each one. The robot will have a variable level of one-upmanship.

We'll use print, input, and math operators and variables to accomplish this:

Example-7:one_up_level = 1
a1 = input("How many seconds does it take you to run the 100 meter dash?")
a1 = int(a1)
print("That's cool. I can do it in", a1 - one_up_level, "seconds though. And I don't even have legs, sooooo...")

a2 = input("But what about your GPA? I'm sure that's pretty good, eh? (Enter your GPA)")
a2 = float(a2)
print("Alright. Mine was", a2 + one_up_level)
print("Not that it matters, lol")

Explanation: No matter what number you tell the chatbot, in his calculations it'll always inform you that he's somehow better. It does this by adding or subtracting, depending on which operation flatters it. Many students delete this program after the exercise.

Example-8:

Activity 2: Make a simple program that tells you if a given number is a multiple of another given number.

A Correct Answer:

```
print("Is _ a multiple of _?")
num1 = input("Write the first number: ")
num2 = input("Write the second number: ")
print(int(num1) % int(num2))
print("If the number above is zero, then", num1, "is a multiple of", num2)
```

Explanation:

Remember, the modulo finds the remainder of the first number divided by the second number. If it gives us zero, then we know that the second number divides evenly into the first number.

Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

1. *Python Anthmetic Operators*
2. *Python Comparison/relational Operators*
3. *Python Logical Operators*
4. *Python Assignment Operators*
5. *Python Identity Operators*
6. *Python Membership Operators*
7. *Python Bitwise Operators*

- **Arithmetic operators**

Arithmetic Operators

Operator	Meaning	Example
+	Addition	4 + 7 ⟶ 11
-	Subtraction	12 - 5 ⟶ 7
*	Multiplication	6 * 6 ⟶ 36
/	Division	30 / 5 ⟶ 6
%	Modulus	10 % 4 ⟶ 2
//	Quotient	18 // 5 ⟶ 3
**	Exponent	3 ** 5 ⟶ 243

Example:

x ,y= 15,4

```
# Output: x + y = 19
 print('x + y =',x+y)

# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3  Floor Division Operator
print('x // y =',x//y)

# Output: x ** y = 50625  Power Operator
print('x ** y =',x**y)
```

Python Mixed-Mode Arithmetic

The calculation which done both integer and floating-point number is called mixed-mode arithmetic. When each operand is of a different type.

Example:

9/2.0 —> 4.5

- **Relational operators**

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Example:

```
x = 5
y = 2

# Output: x > y is False
print('x > y is',x>y)

# Output: x < y is True
print('x < y is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)

# Output: x <= y is True
rint('x <= y is',x<=y)
```

- **Logical operator :**

Logical operators

```
>>> a, b, c = 10, 20, 30

>>> (a > b) and (b < c)
False

>>> (a < b) and (b < c)
True

>>> (a > b) or (b < c)
True
```

Operator	Description
and	logical AND: If both operands are True then it returns True
or	logical OR: If one of the operands is true then it returns True
not	logical NOT

Example :

```
x = True y = False

# Output: x and y is False
print('x and y is',x and y)

# Output: x or y is True
print('x or y is',x or y)

# Output: not x is False
print('not x is',not x)
```

- **Assignment operator :**

Operator	Description
=	x=y , y is assigned to x
+=	x+=y is equivalent to x=x+y
-=	x-=y is equivalent to x=x-y
*=	x*=y is equivalent to x=x*y
/=	x/=y is equivalent to x=x/y
**=	x**=y is equivalent to x=x**y

Assignment operators in Python

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

- **Identity operators**

Operator	Meaning
is	True if the operands are identical (refer to the same object)
is not	True if the operands are not identical (do not refer to the same object)

```
x1 , y1 = 5 ,5
x2 , y2 = "cse","cse"
x3 , y3= [1,2,3],[1,2,3]

print(id(x1))          #20935504
print(id(y1))          #20935504
print(x1 is y1)
print(x1 is not y1)
print(id(x2))          #21527296
print(id(y2))          #21527296
print(x2 is y2)
print(x2 is not y2)
print(id(x3))          #45082264
print(id(y3))          #45061624
print(x3 is y3)
a=10
b=15
x=a
y=b
z=a
print(x is y)
print(x is a)
print(y is b)
print(x is not y)
print(x is not a)
print(x is z)
```

- **Membership operators:**

in: "in" operator return true if a character or the entire substring is present in the specified string, otherwise false.

not in: "not in" operator return true if a character or entire substring does not exist in the specified string, otherwise false.

Example:

```
str1="ramuit"
str2="tirumalacse"
str3="ramu"
str4="tirumala"
print(str3 in str1)                    # True
print(str4 in str2)                    # True
print(str3 in str2)                    # False
print("ratan" in "ratanit")            #true
print("ratan" in "durgasoft")          #False
print(str3 not in str1)                # False
print(str4 not in str2)                # False
print(str3 not in str2)                # True
print("ratan" not in "ratanit")        # false
print("ratan" not in "anu")            # true
```

- **Bitwise operator: -**

Operator	Description
|	Perform binary OR operation
&	Perform binary AND operation
^	Perform binary XOR operation
~	Perform binary one's Complement operation
<<	Left shift operator; left side operand bit is moved left by numeric number specified in right side
>>	Right shift operator; left side operand bit is moved right by numeric number specified in right side

& Operator:

```
print(3&7)
0011
0111
0011
print(9&6)
1001
0101
0000
print(15&15)
1111
1111
1111
print(0&0)
0000
0000
0000
```

| Operator:

```
print(3|7)
0011
0111
0111
print(9|6)
1001
0101
1111
print(15|15)
1111
1111
1111
print(0|0)
0000
0000
0000
```

Data types in Python

Numbers

- Int / float/complex : type
- Describes the numeric value & decimal value
- These are immutable modifications are not allowed.

Boolean

- bool : type
- represent True/False values.
- 0 =False & 1 =True
- Logical operators and or not return value is Boolean

Strings

- str : type
- Represent group of characters
- Declared with in single or double or triple quotes
- It is immutable modifications are not allowed.

Lists

- list : type
- group of heterogeneous objects in sequence.
- This is mutable modifications are allowed
- Declared with in the square brackets [ ]

Tuples :

- tuple : type:
- group of heterogeneous objects in sequence
- this is immutable modifications are not allowed.
- Declared within the parenthesis ( )

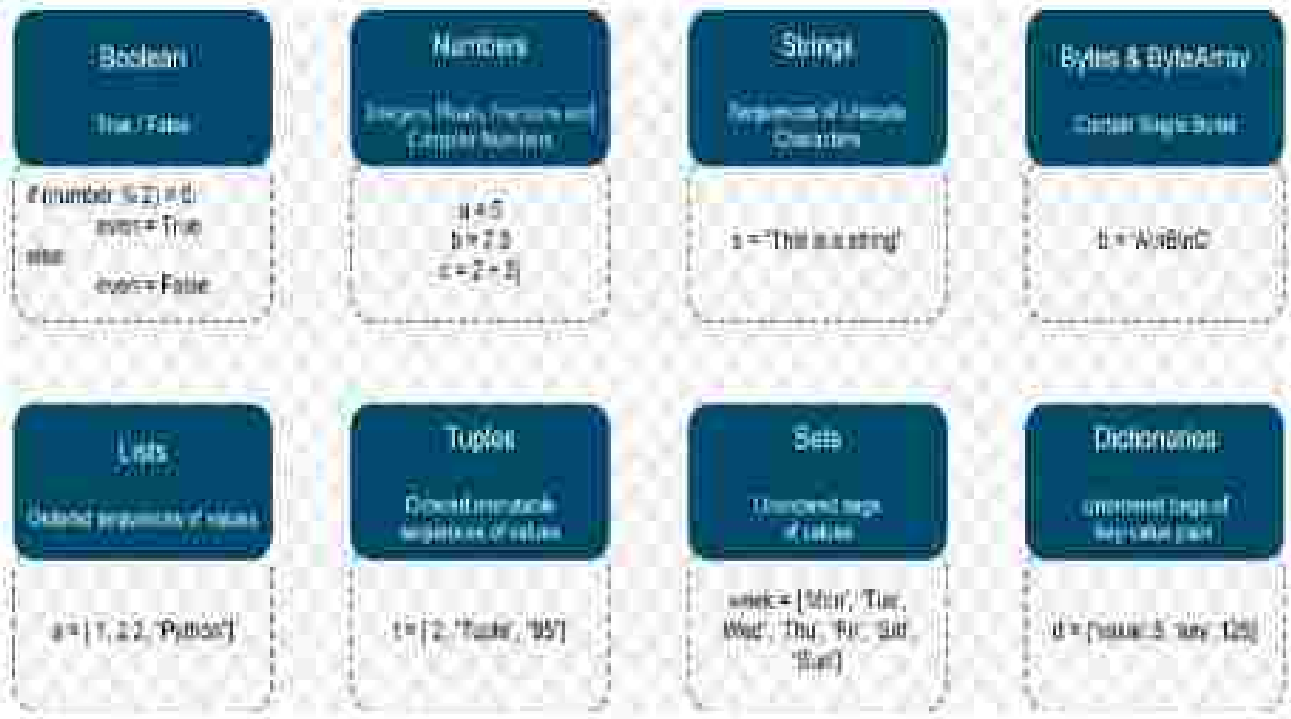
Sets

- set : type
- group of heterogeneous objects in unordered
- this is mutable modifications are allowed
- declared within brasses { }

Dictionaries

- dict : type
- It stores the data in key value pairs format.
- Keys must be unique & value
- It is mutable modifications are allowed.
- Declared within the curly brasses {key:value}

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	



Boolean data type : (bool)

- true& false are result values of comparison operation or logical operation in python.
- true & false in python is same as 1 & 0 1=true 0=false
- except zero it is always True.
- while writing true & false first letter should be capital otherwise error message will be generated.
- Comparison operations are return Boolean values.

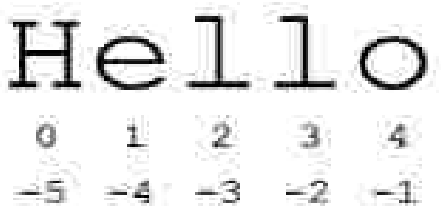
Example:

```
print (1 == 1)          #true
print (5 > 3)           #true
print (True or False)   #true
print (3> 7)            #false
print (True and False)  #false
```

Strings Data type : (str)

- A string is a list of characters in order enclosed by single quote or double quote.
- Python string is immutable modifications are not allowed once it is created.
- In Java String data combine with int data it will become String but not in python.

- String index starts from 0, trying to access character out of index range will generate IndexError.
- In python it is not possible to add any two different data types, possible to add only same data type data.

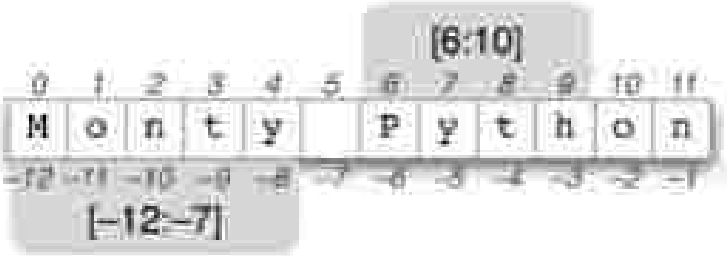


Slice Notation

- <string_name>[startIndex:endIndex],
- <string_name>[:endIndex],
- <string_name>[startIndex:]

- s[1:4] is 'ell' — chars starting at index 1 and extending up to but not including index 4 s[1:] is "ello" — omitting either index defaults to the start or end of the string
- s[:] is 'Hello' — omitting both always gives us a copy of the whole thing
- s[1:100] is 'ello' -- an index that is too big is truncated down to the string length s[-1] is 'o' — last char (1st from the end)

- s[-4] is 'e' — 4th from the end
- s[:-3] is 'He' — going up to but not including the last 3 chars.
- s[-3:] is 'llo' — starting with the 3rd char from the end and extending to the end of the string.



Practice Examples:

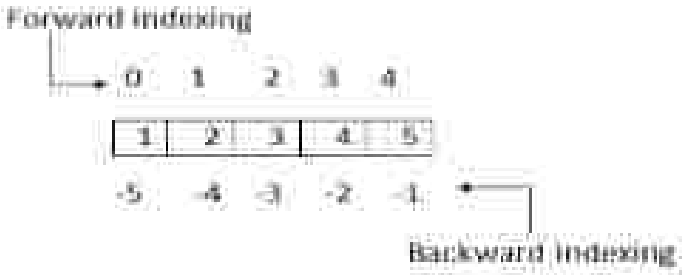
```
str="ratanit"
 print(str[3])          #a
print(str[1:3])         #at
print(str[3:])          #anit
print(str[:4])          #ratan
print(str[:])           #ratanit
print(str[-3])          #n
```

```
print(str[-6:-4])       #at
print(str[-3:])         #nit
print(str[:-5])         #ra
print(str[:])           #ratanit
```

List data type : (list)

- List is used to store the group of values & we can manipulate them, in list the values are stores in index format starts with 0;
- List is mutable object so we can do the manipulations.
- A python list is enclosed between square([]) brackets.
- In list insertion order is preserved it means in which order we inserted element same order output is printed.
- A list can be composed by storing a sequence of different type of values separated by commas.

  <list_name>=[value1,value2,value3,....valuen];
- The list contians forward indexing & backword indexing.



Example: List data

```
data1=[1,2,3,4]              # list of integers
data2=['x','y','z']          # list of String
data3=[12.5,11.6]            # list of floats
data4=[]                     # empty list
data5=['ramu',10.56,4,'a']   # list with mixed data types
```

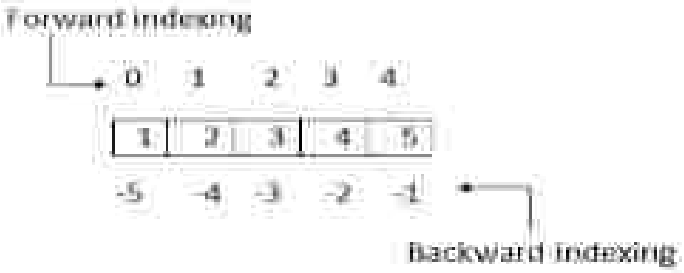
Practice Examples:

Accessing List data

```
data1=[1,2,3,4]
data2=['tirumala','anu','deva']
print (data1[0])
print (data1[0:3])
print (data2[-3:-1])
print (data1[0:])
print (data2[:2])
print (data2[:])
```

Tuple data type : (tuple)

- tuple : type
- group of heterogeneous objects in sequence
- this is immutable modifications are not allowed.
- Declared within the parenthesis ( )
- Insertion order is preserved it means in which order we inserted the objects same order output is printed.
- The tuple contains forward indexing & backward indexing.



Example:

```
tup1 = ['ratan', 'anu', 'durga']
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"                    # valid not recommended
tup4=()
tup5 = (10)
tup6 = (10,)
tup7=(1,2,3,"ratan",10.5)
print(tup1)
print(tup2)
print(tup3)
print(tup4)
print(type(tup5)) #<class 'int'>
print(type(tup6)) #<class 'tuple'>
print(tup7)
```

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses to help us quickly identify

tuples when we look at Python code:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

set:

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets.

Note: to create an empty set you have to use set(), not {}

Example:

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
print(type(my_set))
#creates empty set
s=set()
print(s)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)
```

Frozenset:

- It is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.
- Frozensets can be created using the function frozenset().
- This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(),issuperset(), symmetric_difference() and union(). Being immutable it does not have method that add or remove elements.

Dictionary data type:

- List,tupple ,set data types are used to represent individual objects as a single entity.
- To store the group of objects as a key-value pairs use dictionary.
- A dictionary is a data type similar to arrays, but works with keys and values instead of indexes.
- Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it.
- The keys must be unique keys but values can be duplicated.

Example:

```
phonebook = {}
phonebook["ramu"] = 935577566
phonebook["anu"] = 936677884
phonebook["devi"] = 9476655551
print(phonebook)
```

Example:

Alternatively, a dictionary can be initialized with the same values in the following notation:

```
phonebook = { "ramu" : 935577566, "anu" : 936677884, "devi" : 9476655551}
print(phonebook)
```

- Dictionaries can be created using pair of curly braces { }. Each item in the dictionary consist of key, followed by a colon, which is followed by value. And each item is separated using commas (,) .
- An item has a key and the corresponding value expressed as a pair, key: value.
- In dictionary values can be of any data type and can repeat.
- keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

Example:

```
# empty dictionary
        my_dict = {}
        print(my_dict)
# dictionary with integer keys
        my_dict = {1: 'apple', 2: 'ball'}
        print(my_dict)
# dictionary with mixed keys
        my_dict = {'name': 'John', 1: [2, 4, 3]}
        print(my_dict)
```

Python control flow statements

If-else statement:

- if the condition true if block executed.
- if the condition false else block executed.

Syntax : if(condition): Statement(s)
else:
statement(s)

Example-1:

```
    a=10
if(a>10):
    print("if body")
else:
    print("else body")
```

Example-2:

```
    In python 0=false 1=true
if(1):
    print("hi ramu")
else:
    print("hi anu")
```

Example-3:

```
    In python Boolean constants start with uppercase character.
if(False):
    print ("true body")
else:
    print ("false body")
```

Example-4:

```
        year = 2000

if year % 4 == 0:
print("Year is Leap")
else:
print("Year is not Leap")
```

elif statement :

- The keyword 'elif' is short for 'else if'.
- An if _ elif _ elif _ sequence is a substitute for the switch or case statements found in other languages

Syntax:

```
if expression:
        statement(s)
elif expression:
        statement(s)
elif expression:
        statement(s)
—
else:
        statement(s)
```

Example-1

```
number = 23
guess = int(input("Enter an integer : "))
if guess == number:
        print("Congratulations, you guessed it.")
elif guess < number:
        print("No, it is a little higher number")
else:
        print("No, it is a little lower number")
print("rest of the app")
```

Example:

```
x = int(input("Please enter an integer: "))
if x > 0:
        print ("Positive")
elif x == 0:
        print ("Zero")
```

for loop :

- Used to print the data n number of times based on condition.
- If you do need to iterate over a sequence of numbers, use the built-in function range().

syntax : for <temp-variable> in <sequence-data>:
statement(s)

range() function :

range(10)	1-10
range(5, 10)	5 through 9
range(0, 10, 3)	0, 3, 6, 9
range(-10, -100, -30)	-10, -40, -70

Syntax:

```
for iterator_name in range(10):
        ...statements...
for iterator_name in range(start,end):
        ...statements...
for iterator_name in range(start,stop,increment):
        ...statements...
```

Example:

```
for x in range(10):
    print("Tirumala World",x)
for x in range(8,10):
    print("CSE World",x)
for x in range(3,10,3):
    print("Technical world",x)
for x in range(-20,-10):
    print("Python",x)
for x in range(-28,-10,3):
    print("Running Trendy",x)
for i in range(-10,-100,-15):
    print(i)
for i in range(10, 0, -2):
    print(i)
```

Loops with else block:

ex: else is always executed if the loop executed normally termination.

```
for i in range(1, 5):
        print(i)
else:
        print('The for loop is over')
```

else block is not executed in two cases

case 1: if the exception raised in loop else block not executed.

```
for x in range(10):
        print("Python world",x)
        print(10/0)
else:
        print("else block")
```

case 2: In loop when we use break statement the else block not executed.

```
for x in range(10):
        print("Python Technical Page",x)
        if(x==4):
                break
        else:
                print("else block")
```

Example:

(a)

```
words = ["cat", "apple", "rat","four"]
for w in words[1:3]:
        print(w, len(w))
```

(b)

```
words = ["cat", "apple", "rat","four"]
for w in words:
        print(w, len(w))
```

While loop:

```
while <expression>:
        Body
```

Example :

```
a=0
while(a<10):
        print ("Python New Tech")
        a=a+1
```

Example: **else** is always executed after the while loop is over unless a break statement is encountered.

```
a=0
while(a<10):
        print ("Python New Tech ",a)
        a=a+1
else:
        print("else block after while");
        print("process done")
```

Example: In below example **else not executed.**

```
a=0
while(a<10):
        print ("hi Students", a)
        a=a+1
        if(a==2):
         break
else:
         print("else block after while");
 print("process done")
```

Break & continue:

- Break is used to stop the execution.
- Continue used to skip the particular iteration.

Example:

```
for i in range(1,10):
 if(i==4):
        break
 print(i)
```

Example:

```
for i in range(1,10):
 if(i==4):
         continue
 print(i)
```

Example:

```
while 1:
 n = input("Please enter 'hello':")
 if n.strip() == 'hello':
        break
 else:
```

```
            print("u entered wrong input")
```

Example:

```
while True:
    n = input["enter some name"]
    if(n=='exit'):
        break
    elif(len(n)<3):
        print("name is very small...")
    print["you entered good name..."]
```

Built-in functions and Modules in Python

- The Python interpreter has a number of built-in functions. They are loaded automatically as the interpreter starts and are always available.
- For example, **print()** and **input()** for I/O.
- Number conversion functions **int(), float(), complex()**; data type conversions **list(), tuple(), set()**, etc.
- In addition to built-in functions, a large number of **pre-defined functions** are also available as a part of libraries bundled with Python distributions. These functions are defined as **modules**.
- A module is a file containing definitions of **functions, classes, variables, constants or any other Python objects**. Contents of this file can be made available to any other program.
- Built-in modules are written in C and integrated with the Python interpreter.
- Each built-in module contains resources for certain system-specific functionalities such as OS management, disk IO, etc. The standard library also contains many Python scripts (with the .py extension) containing useful utilities.
- To display a list of all available modules, use the following command in the Python console:

```
>>> help('modules')
```

Which displays all modules that are supported in the python 3

(i) Python - Math Module:

Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

✓ Pie (π) is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793.

```
>>> import math
>>>math.pi
3.141592653589793
```

- ✓ Another well-known mathematical constant defined in the math module is e. It is called **Euler's number** and it is a base of the natural logarithm. Its value is 2.718281828459045.

```
>>>math.e
2.718281828459045
```

- ✓ The math module contains functions for calculating various trigonometric ratios for a given angle.
- ✓ The functions (sin, cos, tan, etc.) need the angle in radians as an argument. We, on the other hand, are used to express the angle in degrees.
- ✓ The math module presents two angle conversion functions: degrees() and radians(), to convert the angle from degrees to radians and vice versa.
- ✓ For example, the following statements convert the angle of 30 degrees to radians and back (Note: π radians is equivalent to 180 degrees).

```
>>>math.radians(30)
0.5235987755982988
>>>math.degrees(math.pi/6)
29.999999999999996
```

- ✓ The following statements show **sin, cos and tan** ratios for the angle of 30 degrees (0.5235987755982988 radians):

```
>>math.sin(0.5235987755982988)
0.49999999999999994
>>>math.cos(0.5235987755982988)
0.8660254037844387
>>>math.tan(0.5235987755982988)
0.5773502691896257
```

- ✓ You may recall that sin(30)=0.5, cos(30)=32 (which is 0.8660254037844387) and tan(30)= 13 (which is 0.5773502691896257).

math.log():

- ✓ The math.log() method returns the natural logarithm of a given number. The natural logarithm is calculated to the base e.

```
>>>math.log(10)
2.302585092994046
```

math.log10():

- ✓ The math.log10() method returns the base-10 logarithm of the given number. It is called the standard logarithm.

```
>>>math.log10(10)
1.0
```

math.exp():

- ✓ The math.exp() method returns a float number after raising e (math.e) to given number. In other words, exp(x) gives e**x.

```
>>>math.exp(10)
1.0
```

This can be verified by the exponent operator:

```
>>>math.e**10
22026.465794806703
```

math.pow():

- ✓ The math.pow() method receives two float arguments, raises the first to the second and returns the result. In other words, pow(4,4) is equivalent to 4**4.

```
>>>math.pow(2,4)
16.0
>>>2**4
16
```

math.sqrt():

- ✓ The math.sqrt() method returns the square root of a given number.

```
>>>math.sqrt(100)
10.0
>>>math.sqrt(3)
1.7320508075688772
```

Representation functions:

- ✓ The **ceil()** function approximates the given number to the smallest integer, greater than or equal to the given floating point number. The **floor()** function returns the largest integer less than or equal to the given number.

```
>>>math.ceil(4.5867)
5
>>>math.floor(4.5687)
4
```

(ii) Python - Statistics Module:

- ✓ The statistics module provides functions to mathematical statistics of numeric data. The following popular statistical functions are defined in this module.
- ✓ The **mean()** method calculates the arithmetic mean of the numbers in a list.

```
>>> import statistics
>>>statistics.mean([2,5,6,9])
5.5
```

- ✓ The **median()** method returns the middle value of numeric data in a list.

```
>>> import statistics
>>>statistics.median([1,2,3,8,9])
3
>>>statistics.median([1,2,3,7,8,9])
5.0
```

- ✓ The **mode()** method returns the most common data point in the list.

```
>>> import statistics
>>>statistics.mode([2,5,3,2,8,3,9,4,2,5,6])
2
```

- ✓ The **stdev()** method calculates the standard deviation on a given sample in the form of a list.

```
>>> import statistics
>>>statistics.stdev([1,1.5,2,2.5,3,3.5,4,4.5,5])
1.3693063937629153
```

(iii) Python - Random Module

- ✓ Functions in the random module depend on a pseudo-random number generator function random(), which generates a random float number between 0.0 and 1.0.
- ✓ **random.random():** Generates a random float number between 0.0 to 1.0. The function doesn't need any arguments.

```
>>>import random
>>>random.random()
0.6451736848075 33
```

- ✓ **random.randint():** Returns a random integer between the specified integers.

```
>>>import random
>>>random.randint(1,100)
95
>>>random.randint(1,100)
49
```

- ✓ **random.randrange():** Returns a randomly selected element from the range created by the start, stop and step arguments. The value of start is 0 by default. Similarly, the value of step is 1 by default.

```
>>>random.randrange(1,10)
2
>>>random.randrange(1,10,2)
5
>>>random.randrange(0,101,10)
80
```

- ✓ **random.choice():** Returns a randomly selected element from a non-empty sequence. An empty sequence as argument raises an IndexError.

```
>>>import random
>>>random.choice('computer')
't'
>>>random.choice([12,23,45,67,65,43])
45
>>>random.choice((12,23,45,67,65,43))
67
```

- ✓ **random.shuffle():** This functions randomly reorders the elements in a list.

```
>>>numbers=[12,23,45,67,65,43]
>>>random.shuffle(numbers)
>>>numbers
[23, 12, 43, 65, 67, 45]
>>>random.shuffle(numbers)
>>>numbers
[23, 43, 65, 45, 12, 67]
```

Functions ---- Def keyword

Step 1: Declare the function with the keyword def followed by the function name.

Step 2: Write the arguments inside the opening and closing parentheses of the function, and end the declaration with a colon.

Step 3: Add the program statements to be executed

Step 4: End the function with/without return statement.

Syntax :

```
def function_name(parameters):
"""doc string"""
statement(s)
```

Example:

```
def userDefFunction (arg1, arg2, arg3 ...):
 program statement1
 program statement3
 program statement3
 ---
 return
```

There are two types of functions

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Advantages of functions :

- User-defined functions are reusable code blocks; they only need to be written once, then they can be used multiple times.
- These functions are very useful, from writing common utilities to specific business logic.
- The code is usually well organized, easy to maintain, and developer-friendly.
- A function can have dif types of arguments & return value.

Example : 1

```
def disp():
        print("hi srividya")
        print("hi students")
disp()          # function calling
```

Example : 2

To specify no body of the function use pass statement.

```
def disp():
 pass
disp()
```

Example : 3

one function is able to call more than one function.

```
>>>def happyBirthday(person):
        print("Happy Birthday dear ",person)

>>>def mohan():
        happyBirthday('CSE')
       happyBirthday('IT')
>>>mohan()
```

Inner functions: A function can be created as an inner function in order to protect it from everything that is happening outside of the function. In that case, the function will be hidden from the global scope.

Example: 1

```
def function1(): # outer function
   print ("Hello from outer function")
   def function2(): # inner function
      print ("Hello from inner function")
   function2()

>>>function1()
```

Output:

Hello from outer function
Hello from inner function

Explanation:

In the above example, function2() has been defined inside function1(), making it an inner function. To call function2(), we must first call function1(). The function1() will then go ahead and call function2() as it has been defined inside it.

It is important to mention that the outer function has to be called in order for the inner function to execute. If the outer function is not called, the inner function will never execute.

Example:

- Inside the inner function to represent outer function variable use **nonlocal keyword.**

```
def outer():
 var_outer = 'TEC'
 print (var_outer)
 def inner():
        nonlocal var_outer
        var_outer="CSE&IT"
 inner()                          #calling of inner function
 print (var_outer)

>>>outer()
```

Output:

```
TEC
CSE&IT
```

Example:

- Inside the function to represent the global value use global keyword.

```
name='Tirumala'
def outer():
 var_outer = 'Eamcet Code'


 def inner():
        nonlocal var_outer
        var_outer="Eamcet Code:TMLN"

 global name
 print(name)
 name="TEC"
 print(name)

 print (var_outer)
 inner()        #calling of inner function
 print (var_outer)
```

```
>>>outer()
```

Output:

```
Tirumala
TEC
Eamcet Code
Eamcet Code:TMLN
```

Function vs arguments:-

1. default arg
2. required arg
3. keyword argument
4. variable argument

1 .Default arguments:

When we call the function if we are not passing any argument the default value is assigned.

Example:-1

```
def empdetails(eid=1,ename="anu",esal=10000):
    print ("Emp id =", eid)
    print ("Emp name = ", ename)
    print ("Empsal=", esal)
    print("**********")
empdetails()
empdetails(222)
empdetails(333,"bhavani")
empdetails(111,"srividya",10.5)
```

Output:

```
Emp id = 1
Emp name =  anu
Empsal= 10000
**********
Emp id = 222
Emp name =  anu
Empsal= 10000
**********
Emp id = 333
Emp name =  bhavani
Empsal= 10000
```

```
*********
Emp id = 111
Emp name = srividya
Empsal= 10.5
*********
```

2. Required arguments: Required arguments are the mandatory arguments of a function. These argument values must be passed in correct number and order during function call.

Example:-1

```
def add(a,b):
 print(a+b)
```

```
>>> add(10,20)                    Outout: 30
add(10.5,20.4)                    Output: 30.9
add("Tirumala","CSE")             Output: TirumalaCSE
add(10,10.5)                      Output:20.5
```

```
add("BALL",10)
Output: Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    add("BALL",10)
  File "<pyshell#58>", line 2, in add
    print(a+b)
TypeError: must be str, not int
```

3. Keyword arguments / named arguments:

- The keywords are mentioned during the function call along with their corresponding values.
- These keywords are mapped with the function arguments so the function can easily identify the corresponding values even if the order is not maintained during the function call.
- Using the Keyword Argument, the argument passed in function call is matched with function definition on the basis of the name of the parameter.

Example-1:

```
def msg(id,name):
   print(id)
   print(name)

msg(id=111,name='CSE')
msg(name='Students',id=222)
```

Output:

```
111
cse
222
Students
```

4. Variable number of arguments:

- This is very useful when we do not know the exact number of arguments that will be passed to a function.
- Or we can have a design where any number of arguments can be passed based on the requirement.

Example:

```
def disp(*var):
 for i in var:
    print("var arg=",i)

disp()
disp(10,20,30)
disp(10,20.3,"ratan")
```

Output:

```
var arg= 10
var arg= 20
var arg= 30
var arg= 10
var arg= 20.3
var arg= ratan
```