

Welcome To Thinking Programming



Workshop

About Me

- I'm Dr.S.Gowrishankar, working as an Professor in the Department of Computer Science and Engineering at Dr.Ambedkar Institute of Technology, Bengaluru – 56.



- 3.6 Years of Industry Experience and 8+ Years of Teaching Experience.
 - Research interests include Data Science with emphasis on Machine Learning, Deep Learning, and Big Data Analytics.
-

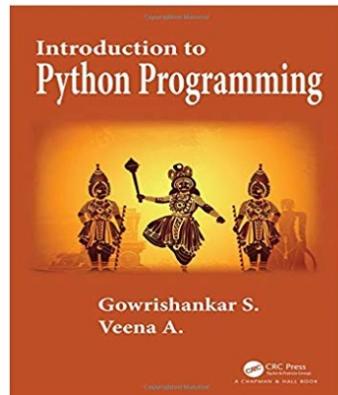
About Co-Speaker

- Mr.Srinivasa A. H., is currently working as an Associate Professor in the Department of Computer Science and Engineering at Dr.Ambedkar Institute of Technology, Bengaluru – 56.



- 18+ Years of Teaching Experience.
- Research interests include Internet of Things with emphasis on architecture, optimizing routing protocols, energy efficiency and sustainability.

Authored Book



"Introduction to Python Programming", authored by *Gowrishankar S.* and *Veena A.* and published by Taylor and Francis/CRC Press, USA. (ISBN = 0815394373)

Contact Dr.S.Gowrishankar

| | |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
|  | https://www.gowrishankarnath.com |
|  | gowrishankarnath@acm.org |
|  | @g_s_nath |
|  | https://in.linkedin.com/in/gowrishankarnath |
|  | https://github.com/gowrishankarnath |

Contact Mr.Srinivasa A. H.

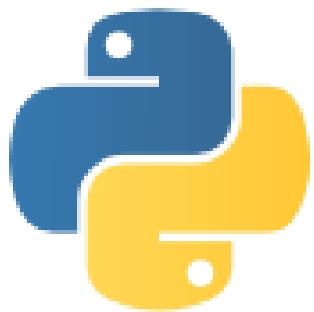
| | |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
|  | 9449611659 |
|  | srinivasaah.cs@drait.edu.in |

Prerequisites for this Workshop

No prior experience with any other programming language is required.

Of course, any familiarity with any other programming will be helpful, but is not required.

python



powered

Objectives of this Workshop

The objectives of the workshop is to provide hands-on experience in python programming and explore the features of the Python programming language.

Flow of the Workshop

| Thinking Programming | | |
|-----------------------------|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Training | Resource Persons | Topics to be covered |
| Day 1 20/12/2021 | Dr. Gowrishankar S. Mr. Srinivasa A. H. | Installation of Anaconda Python Distribution Using Jupyter Notebook Parts of Python Programming Language Control Flow Statements + Hands-on Session |
| Day 2 21/12/2021 | Ms. Asha Rani K. P. Ms. Veena A. | Functions Strings Lists + Hands-on Session |
| Day 3 22/12/2021 | Ms. Veena A. Ms. Asha Rani K. P. | Dictionaries Tuples and Sets Files + Hands-on Session |
| Day 4 23/12/2021 | Dr. Gowrishankar S. Mr. Srinivasa A. H. | Functional Programming Numpy Pandas + Hands-on Session |

To the Audience

- Audience are encouraged to make a note of all the concepts while discussing
- Based on your understanding of the concepts, you need to write the code to solve the questionnaires during Hands-on session
- Go to the following URL if you want to execute the code in parallel while discussing
Replace This <https://tinyurl.com/UVCE-Workshop>
- Use Audience_Test_the_Code.ipynb file to execute code samples

PYPL PopularitY of Programming Language

The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.

Worldwide, Dec 2021 compared to a year ago:

| Rank | Change | Language | Share | Trend |
|------|--------|-------------|---------|--------|
| 1 | | Python | 30.21 % | -0.5 % |
| 2 | | Java | 17.82 % | +1.3 % |
| 3 | | JavaScript | 9.16 % | +0.6 % |
| 4 | | C# | 7.53 % | +1.0 % |
| 5 | | C/C++ | 6.82 % | +0.6 % |
| 6 | | PHP | 5.84 % | -0.2 % |
| 7 | | R | 3.81 % | -0.0 % |
| 8 | ↑ | Swift | 2.03 % | -0.2 % |
| 9 | ↓ | Objective-C | 2.02 % | -1.6 % |
| 10 | ↑ | Matlab | 1.73 % | -0.1 % |

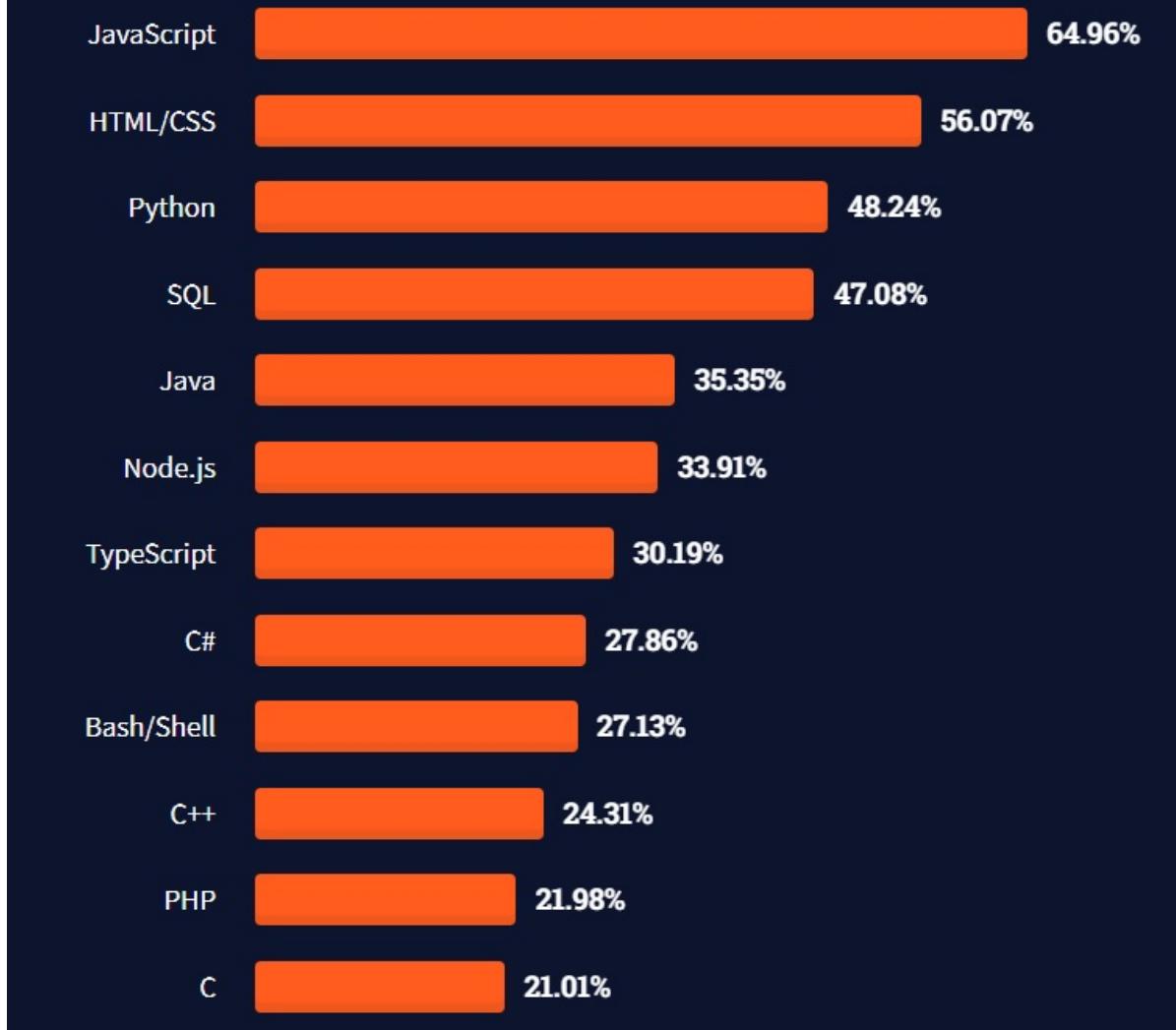
IEEE Top Programming Languages 2021

| Rank | Language | Type | Score |
|------|-------------|-------|-------|
| 1 | Python▼ | 🌐💻⚙️ | 100.0 |
| 2 | Java▼ | 🌐📱💻 | 95.4 |
| 3 | C▼ | 📱💻⚙️ | 94.7 |
| 4 | C++▼ | 📱💻⚙️ | 92.4 |
| 5 | JavaScript▼ | 🌐 | 88.1 |
| 6 | C#▼ | 🌐📱💻⚙️ | 82.4 |
| 7 | R▼ | 💻 | 81.7 |
| 8 | Go▼ | 🌐💻 | 77.7 |
| 9 | HTML▼ | 🌐 | 75.4 |
| 10 | Swift▼ | 📱💻 | 70.4 |

Top 10 programming languages (Stack Overflow's 2021 Developer Survey)

According to Stack Overflow's 2021 Developer Survey, these are the most commonly used programming languages in 2021 (among the >83K people surveyed)

Professional Developers



Python Domain

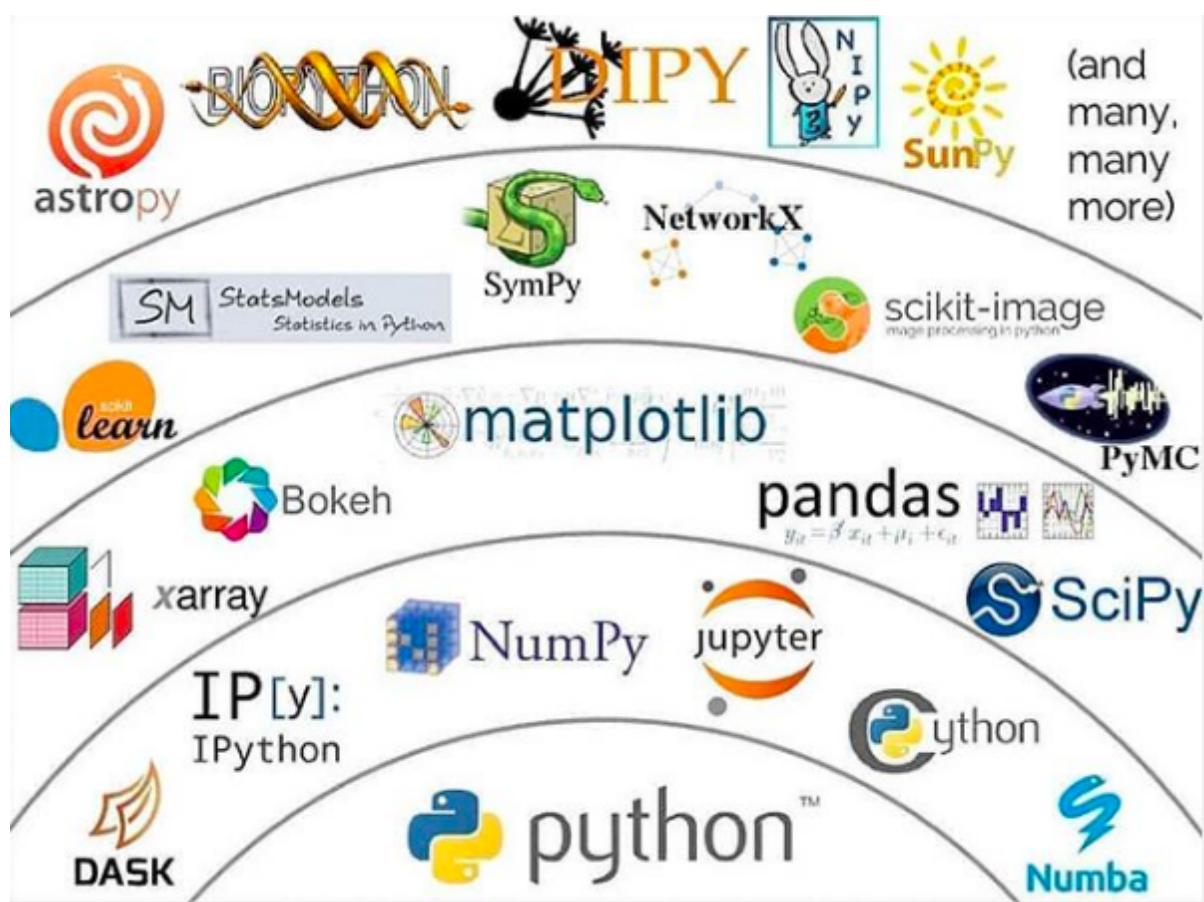


Top Companies Using Python

Top Companies Using Python



Python Ecosystem



How to obtain Python

Anaconda | The World's Most Po| X +

← → X https://www.anaconda.com

 ANACONDA.

Anaconda Individual Edition

[Download !\[\]\(cb320afc207b08dcff2bf04000ce280a_img.jpg\)](#)

For Windows
Python 3.9 • 64-Bit Graphical Installer • 510 MB

Get Additional Installers

 |  | 



Welcome to Anaconda3 5.2.0 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 5.2.0 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Next >

Cancel



ANACONDA

License Agreement

Please review the license terms before installing Anaconda3 5.2.0 (64-bit).

Press Page Down to see the rest of the agreement.

```
=====
Anaconda End User License Agreement
=====
```

Copyright 2015, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 5.2.0 (64-bit).

Anaconda, Inc.

< Back

I Agree

Cancel



ANACONDA

Select Installation Type

Please select the type of installation you would like to perform for Anaconda3 5.2.0 (64-bit).

Install for:

- Just Me (recommended)
- All Users (requires admin privileges)

Anaconda, Inc.

< Back

Next >

Cancel



ANACONDA

Choose Install Location

Choose the folder in which to install Anaconda3 5.2.0 (64-bit).

Setup will install Anaconda3 5.2.0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Users\Admin\Anaconda3

Browse...

Space required: 3.0GB

Space available: 331.3GB

Anaconda, Inc.

< Back

Next >

Cancel

**Advanced Installation Options**

Customize how Anaconda integrates with Windows

Advanced Options

- Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

- Register Anaconda as my default Python 3.6

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.

Anaconda, Inc. —

< Back

Install

Cancel

**Installing**

Please wait while Anaconda3 5.2.0 (64-bit) is being installed.

Extract: vs2015_runtime-14.0.25123-3.tar.bz2

[Show details](#)

Anaconda, Inc. —

< Back

Next >

Cancel



Installation Complete

Setup was completed successfully.

Completed

Show details

Anaconda, Inc.

< Back

Next >

Cancel



ANACONDA

Anaconda3 5.2.0 (64-bit)

Microsoft Visual Studio Code Installation

Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.

To install Visual Studio Code, you will need Administrator Privileges and Internet connectivity.

[Visual Studio Code License](#)

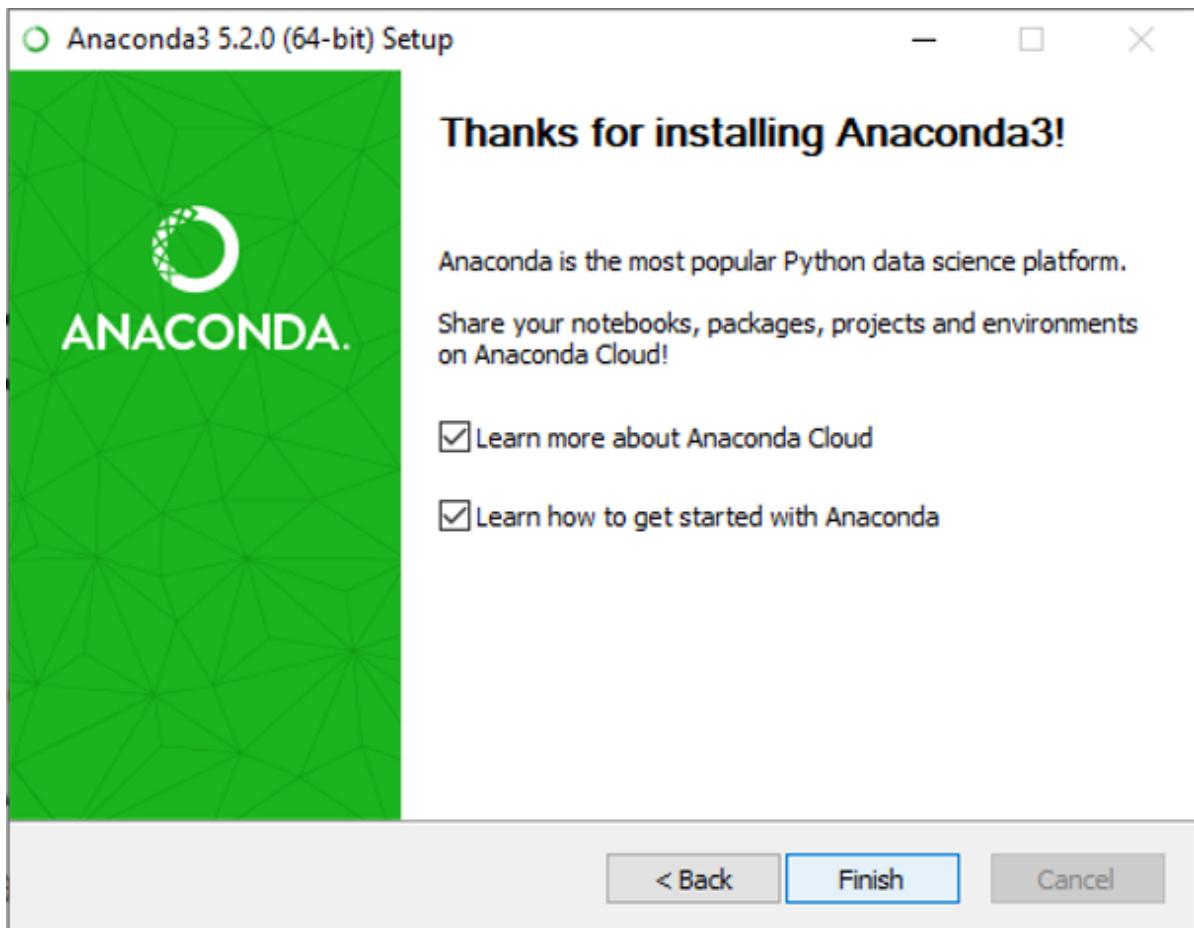
[Install Microsoft VSCode](#)

Anaconda, Inc.

< Back

Skip

Cancel

A screenshot of a Command Prompt window titled "Command Prompt - python". It shows the following output:

```
Microsoft Windows [Version 10.0.19044.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Gowrishankarnath>python
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

What is Jupyter Notebook?

- The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects.
- A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media.

Jupyter Notebook

- Jupyter Notebook is part of installed Anaconda distribution package. You need not install it separately.
- Create a directory in `C:\` drive. Let's assume we have created a directory **Workshop**.
- Open Command prompt using **cmd** command.

- Navigate to the directory **Workshop** using cd command. For example,
C:\cd Workshop
 - Next, issue the command
C:jupyter notebook
-

Jupyter Dashboard

The screenshot shows the Jupyter Notebook dashboard. At the top, there's a logo and a "Logout" button. Below that is a navigation bar with "Files", "Running", and "Clusters" tabs. A message says "Select items to perform actions on them." On the right, there are "Upload" and "New" buttons, and a dropdown for "Name" and "Last Modified". The main area shows a list of files and folders:

| | Name | Last Modified |
|---|------------|---------------|
| 0 | 3D Objects | 11 days ago |
| | Contacts | 11 days ago |
| | Desktop | 11 days ago |
| | Documents | 5 days ago |
| | Downloads | 2 days ago |
| | Favorites | 11 days ago |

New Notebook Creation

A red speech bubble with the text "Click here" points to the "New" button in the top right corner of the dashboard. A dropdown menu is open, listing options: "Notebook: Python 3", "Other: Text File", "Folder", and "Terminals Unavailable".

A red speech bubble with the text "Change file name here" points to the file name "Untitled" in the title bar. The title bar also shows "Last Checkpoint: Last Saturday at 12:36 PM (unsaved changes)". The dashboard includes a "Logout" button and a "Kernel starting, please wait..." message. The main area shows a code editor with an empty cell labeled "In []: 1 |".

- Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell. Enter your code in the cell.

- Click on Run button or else press Shift + Enter to execute your code in the cell.
 - Jupyter Notebooks are saved with **.ipynb** extension. You can create any number of notebooks.
-

Guido Van Rossum – Creator Python Programming Language



About Python Programming Language



- Often people assume that the name Python was written after a snake.
 - Even the logo of Python programming language depicts the picture of two snakes, blue and yellow.
 - Guido Van Rossum named it after the television show Monty Python's Flying Circus.
-

In []: `print("Hello World!!")`

Hello World!!

Parts of Python Programming Language

After completing this session, you should be able to

- Define identifiers, keywords, operators and expressions.
- Use different operators, expressions and variables available in Python.

- Build complex expressions using operators.
 - Determine the data types of values.
 - Use indentation and comments in writing Python programs.
-

Identifiers

An identifier is a name given to a variable, function, class or module.

Identifiers may be one or more characters in the following format:

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). Names like myCountry, other_1 and goodmorning, all are valid examples. A Python identifier can begin with an alphabet (A – Z and a – z and).
 - An identifier cannot start with a digit but is allowed everywhere else. 1plus is invalid, but plus1 is perfectly fine.
 - Keywords cannot be used as identifiers.
 - One cannot use spaces and special symbols like !, @, #, \$, % etc. as identifiers.
 - Identifier can be of any length.
-

Keywords

- Keywords are a list of reserved words that have predefined meaning.
- Keywords are special vocabulary and cannot be used by programmers as identifiers for variables, functions, constants or with any identifier name.
- Attempting to use a keyword as an identifier name will cause an error.

| Keywords in Python programming language | | | | |
|-----------------------------------------|----------|---------|----------|--------|
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

Statements

A statement is an instruction that the Python interpreter can execute.

For example, `z = 1` is an assignment statement.

Expressions

Expression is an arrangement of values and operators which are evaluated to make a new value.

Expressions are statements as well.

For example,

```
In [ ]:
```

```
8+2
```

```
Out[ ]:
```

```
10
```

Variables

- Variable is a named placeholder to hold any type of data which the program can use to assign and modify during the course of execution.
- In Python, there is no need to declare a variable explicitly by specifying whether the variable is an integer or a float or any other type.
- To define a new variable in Python, we simply assign a value to a name.

Legal Variable Names

Follow the below-mentioned rules for creating legal variable names in Python.

- Variable names can consist of any number of letters, underscores and digits.
- Variable should not start with a number.
- Python Keywords are not allowed as variable names.
- Variable names are case-sensitive. For example, `computer` and `Computer` are different variables.

Assigning Values to Variables

The general format for assigning values to variables is as follows:

variable_name = expression

```
In [ ]:
```

```
number =100
print(number)
miles =1000.0
print(miles)
name ="Python"
print(name)
```

```
100
1000.0
Python
```

```
In [ ]: century = 100  
print(century)  
  
century = "hundred"  
print(century)
```

```
100  
hundred
```

Operators

Python language supports a wide range of operators. They are

1. Arithmetic Operators
 2. Assignment Operators
 3. Comparison Operators
 4. Logical Operators
 5. Bitwise Operators
-

Arithmetic Operators

Arithmetic operators are used to execute arithmetic operations such as addition, subtraction, division, multiplication etc.

```
In [ ]: print("Addition Operator")  
print(10 + 35)  
print("Subtraction Operator")  
print(-10 + 35)  
print("Multiplication Operator")  
print(4*2)  
print("Exponent Operator")  
print(4**2)  
print("Division Operator")  
print(45/10)  
print("Modulus Operator --> Returns a remainder")  
print(2025%10)  
print("Floor division Operator")  
print("Returns the integral part of the quotient")  
print(45//10.0)  
print(2025//10)
```

```
Addition Operator  
45  
Subtraction Operator  
25  
Multiplication Operator  
8  
Exponent Operator  
16  
Division Operator  
4.5  
Modulus Operator --> Returns a remainder  
5  
Floor division Operator  
Returns the integral part of the quotient
```

Assignment Operators

In []:

```
x = 5
x = x + 1
print(x)
```

6

If you try to update a variable which doesn't contain any value, you get an error.

```
z = z + 1
```

```
NameError: name 'z' is not defined
```

Assign 0 to variable z to remove the error

In []:

```
z = 0
x = z + 1
print(x)
```

1

In []:

```
## Assignment Operators

print("Assignment")
p = 10
q = 12

print("Addition Assignment")
q += p
print(q)
```

Assignment

Addition Assignment

22

In []:

```
print("Floor Division Assignment")
p = 10
q = 12
q //= p
print(q)
```

Floor Division Assignment

1

In []:

```
print("Exponentiation Assignment")
p = 5
q = 2
q **= p
print(q)
```

Exponentiation Assignment

32

In []:

```
print("Reminder Assignment")
```

```
p = 10  
q = 12  
q %= p  
print(q)
```

Reminder Assignment
2

```
In [ ]: print("Division Assignment")  
p = 10  
q = 12  
q /= p  
print(q)
```

Division Assignment
1.2

```
In [ ]: print("Subtraction Assignment")  
p = 10  
q = 12  
q -= p  
print(q)
```

Subtraction Assignment
2

```
In [ ]: print("Multiplication Assignment")  
p = 10  
q = 12  
q *= p  
print(q)
```

Multiplication Assignment
120

Comparison Operators

When the values of two operands are to be compared then comparison operators are used.

The output of these comparison operators is always a Boolean value, either **True** or **False**.

```
In [ ]: print("Equal Operator")  
print(10 == 12)  
  
print("Not Equal To Operator")  
print(10 != 12)  
  
print("Less Than Operator")  
print(10 < 12)  
  
print("Greater Than Operator")  
print(10 > 12)  
  
print("Lesser than or Equal to")  
print(10 <= 12)  
  
print("Greater than or equal to")  
print(10 >= 12)
```

```
print("P" < "Q")  
print("Aston" > "Asher")  
print(True == True)
```

```
Equal Operator  
False  
Not Equal To Operator  
True  
Less Than Operator  
True  
Greater Than Operator  
False  
Lesser than or Equal to  
True  
Greater than or equal to  
False  
True  
True  
True
```

Logical Operators

The result of the logical operator is always a Boolean value, **True** or **False**. The logical operators are **and** , **or** and **not**

```
In [ ]:  
print(True and False) # False  
print(True or False) # True  
print(not(True) and False) # False  
print(not(True and False)) # True  
print((10 < 0) and (10 > 2)) # False  
print((10 < 0) or (10 > 2)) # True  
print(not(10 < 0) or (10 > 2)) # True  
print(not(10 < 0 or 10 > 2)) # False
```

```
False  
True  
False  
True  
False  
True  
True  
False
```

Bitwise operators

In Python, bitwise operators are used to performing bitwise calculations on integers.

The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators.

| OPERATOR | DESCRIPTION | SYNTAX |
|----------|---------------------|----------|
| & | Bitwise AND | $x \& y$ |
| | Bitwise OR | $x y$ |
| ~ | Bitwise NOT | $\sim x$ |
| ^ | Bitwise XOR | $x ^ y$ |
| >> | Bitwise right shift | $x >>$ |
| << | Bitwise left shift | $x <<$ |

Bitwise and operator: Returns 1 if both the bits are 1 else 0. Example:

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a & b = 1010

&

0100

= 0000

= 0 (Decimal)

Bitwise or operator: Returns 1 if either of the bit is 1 else 0. Example:

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a | b = 1010

|

0100

= 1110

= 14 (Decimal)

Bitwise not operator: Returns one's complement of the number. Example:

`a = 10 = 1010 (Binary)`

`~a = ~1010`
`= -(1010 + 1)`
`= -(1011)`
`= -11 (Decimal)`

Bitwise xor operator: Returns 1 if one of the bits is 1 and the other is 0 else returns false.

Example:

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a & b = 1010

^

0100

= 1110

= 14 (Decimal)

Shift Operators

These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.

They can be used when we have to multiply or divide a number by two.

Bitwise right shift

Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result.

Similar effect as of dividing the number with some power of two.

Example 1:

a = 10 = 0000 1010 (Binary)

a >> 1 = 0000 0101 = 5

Example 2:

a = -10 = 1111 0110 (Binary)

a >> 1 = 1111 1011 = -5

Bitwise left shift

Shifts the bits of the number to the left and fills 0 on voids right as a result.

Similar effect as of multiplying the number with some power of two.

Example 1:

`a = 5 = 0000 0101 (Binary)`

`a << 1 = 0000 1010 = 10`

`a << 2 = 0001 0100 = 20`

Example 2:

`b = -10 = 1111 0110 (Binary)`

`b << 1 = 1110 1100 = -20`

`b << 2 = 1101 1000 = -40`

Operator Precedence and Associativity

Operator precedence determines the way in which operators are parsed with respect to each other.

Almost all the operators have left-to-right associativity.

Operator Precedence in Python

| Operators | Meaning |
|-------------------------------------------------|---------------------------------------------------|
| () | Parentheses |
| ** | Exponent |
| +x, -x, ~x | Unary plus, Unary minus, Bitwise NOT |
| *, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, - | Addition, Subtraction |
| <<, >> | Bitwise shift operators |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| | Bitwise OR |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisons, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |

In []:

```
print(2 + 3 * 6)

print((2 + 3) * 6)

print(6 * 4 / 2)
```

```
20
30
12.0
```

Indentation

In Python, Programs get structured through indentation.

Usually, four whitespaces are used for indentation and are preferred over tabs.

→ Indicates 1 Space Indentation

```
Statement 1  
Statement 2  
Statement 3  
    Statement 4  
Statement 5  
Statement 6  
Statement 7
```

How the interpreter visualises



Here:

Statements 1, 2, 7 belong to code block 1 as they are at the same distance to the right.
Statements 3, 5, 6 belong to code block 2
Statement 4 belongs to code block 3

Execution happens in the same order.

Comments

Comments are an important part of any program.

A comment is a text that describes what the program or a particular part of the program is trying to do and is ignored by the Python interpreter.

In Python, use the **hash (#) symbol** to start writing a comment.

Single Line Comment

This is single line Python comment

Multiline Comments

This is

multiline comments

in Python

Another way of doing this is to use triple quotes, either "" or """.
""This is multiline comment in Python using triple quotes""

Reading Input

In Python, **input()** function is used to gather data from the user.

The syntax for input function is,

```
variable_name = input([prompt])
```

prompt is a string written inside the parenthesis that is printed on the screen.

```
In [ ]: person = input("What is your name?")
print(person)
```

Carrey

Print Output

The print() function allows a program to display text onto the console.

f-strings

- Formatted strings or f-strings were introduced in Python 3.6.
</br>
- A f-string is a string literal that is prefixed with "f".
</br>
- You need to specify the name of the variable or the expression itself inside the curly braces {} to display its value.

```
In [ ]: # Code to Demonstrate the Use of f-strings with print() Function

country = input("Which country do you live in?")
print(f"I live in {country}")
```

I live in India

```
In [ ]: # Given the Radius, Write Python Program to Find the Area and
# Circumference of a Circle

radius = int(input("Enter the radius of a circle"))
area_of_a_circle = 3.1415 * radius * radius
circumference_of_a_circle = 2 * 3.1415 * radius
print(
    f"Area = {area_of_a_circle} and Circumference = {circumference_of_a_circle}")
```

Area = 78.53750000000001 and Circumference = 31.415000000000003

```
In [ ]: # Write Python Program to Convert the Given Number of Days
# to a Measure of Time Given in Years, Weeks and Days. For Example,
# 375 Days Is Equal to 1 Year, 1 Week and 3 Days (Ignore Leap Year).

number_of_days = int(input("Enter number of days"))
number_0f_years = int(number_of_days/365)
number_of_weeks = int(number_of_days % 365 / 7)
remaining_number_of_days = int(number_of_days % 365 % 7)
print(
    f"Years = {number_0f_years}, Weeks = {number_of_weeks}, Days = {remaining_number}
```

Years = 1, Weeks = 26, Days = 3

In []:

```
# Program to Demonstrate int() Casting Function

float_to_int = int(3.5)
string_to_int = int("1")
print(f"After Float to Integer Casting the result is {float_to_int}")
print(f"After String to Integer Casting the result is {string_to_int}")
```

After Float to Integer Casting the result is 3
After String to Integer Casting the result is 1

In []:

```
# Program to Demonstrate str() Casting Function

int_to_string = str(8)
float_to_string = str(3.5)
print(f"After Integer to String Casting the result is {int_to_string}")
print(f"After Float to String Casting the result is {float_to_string}")
```

After Integer to String Casting the result is 8
After Float to String Casting the result is 3.5

In []:

```
# Program to Demonstrate chr() Casting Function

ascii_to_char = chr(100)
print(f'Equivalent Character for ASCII value of 100 is {ascii_to_char}')
```

Equivalent Character for ASCII value of 100 is d

In []:

```
# Program to Demonstrate complex() Casting Function

complex_with_string = complex("1")
complex_with_number = complex(5, 8)
print(f"Result after using string in real part {complex_with_string}")
print(
    f"Result after using numbers in real and imaginary part {complex_with_number}")
```

Result after using string in real part (1+0j)
Result after using numbers in real and imaginary part (5+8j)

In []:

```
# Program to Demonstrate ord() Casting Function

unicode_for_integer = ord('4')
unicode_for_alphabet = ord("Z")
unicode_for_character = ord("#")
print(f"Unicode code point for integer value of 4 is {unicode_for_integer}")
print(f"Unicode code point for alphabet 'A' is {unicode_for_alphabet}")
print(f"Unicode code point for character '#' is {unicode_for_character}")
```

Unicode code point for integer value of 4 is 52
Unicode code point for alphabet 'A' is 90
Unicode code point for character '#' is 35

In []:

```
# Program to Demonstrate hex() Casting Function
```

```
int_to_hex = hex(255)
print(f"After Integer to Hex Casting the result is {int_to_hex}")
```

After Integer to Hex Casting the result is 0xff

```
In [ ]: # Program to Demonstrate oct() Casting Function
```

```
int_to_oct = oct(255)
print(f"After Integer to Hex Casting the result is {int_to_oct}")
```

After Integer to Hex Casting the result is 0o377

The type() function

The syntax for type() function is,

type(object)

The type() function returns the data type of the given object.

```
In [ ]:
```

```
print(type(1))
print(type(6.4))
print(type("A"))
print(type(True))
```



```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

is and is not Identity Operators

The operators is and is not are identity operators.

Operator is evaluates to True if the values of operands on either side of the operator point to the same object and False otherwise.

The operator is not evaluates to False if the values of operands on either side of the operator point to the same object and True otherwise.

```
In [ ]:
```

```
x = "Seattle"
y = "Seattle"
x is y
```

```
Out[ ]: True
```

```
In [ ]:
```

```
place = "paris"
place is not "USA"
```

<>2: SyntaxWarning: "is not" with a literal. Did you mean "!="?

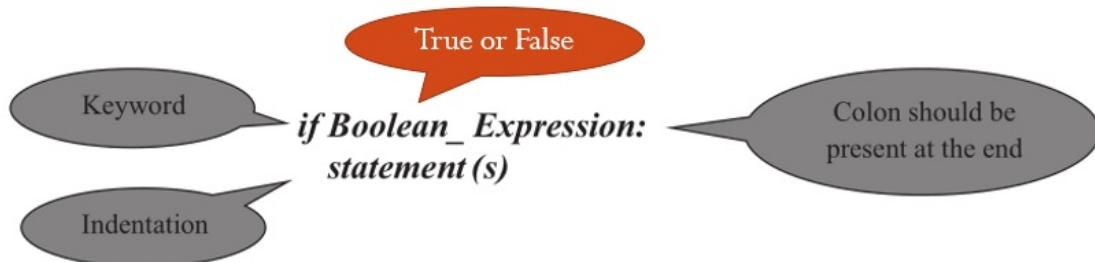
```
<>:2: SyntaxWarning: "is not" with a literal. Did you mean "!="?
C:\Users\GOWRIS~1\AppData\Local\Temp\ipykernel_17208/1247822337.py:2: SyntaxWarning:
"is not" with a literal. Did you mean "!="?
    place is not "USA"
Out[ ]: True
```

Control Flow Statements

After completing this session, you should be able to

- Use if, if...else and if...elif...else statements to transfer the control from one part of the program to another.
- Write while and for loops to run one or more statements repeatedly.
- Control the flow of execution using break and continue statements.
- Improve the reliability of code by incorporating exception handling mechanisms through try-except blocks.

The if Decision Control Statement



```
In [ ]: # Program Reads a Number and Prints a Message If It Is Positive

number = int(input("Enter a number"))
if number >= 0:
    print(f"The number entered by user is a positive number")
```

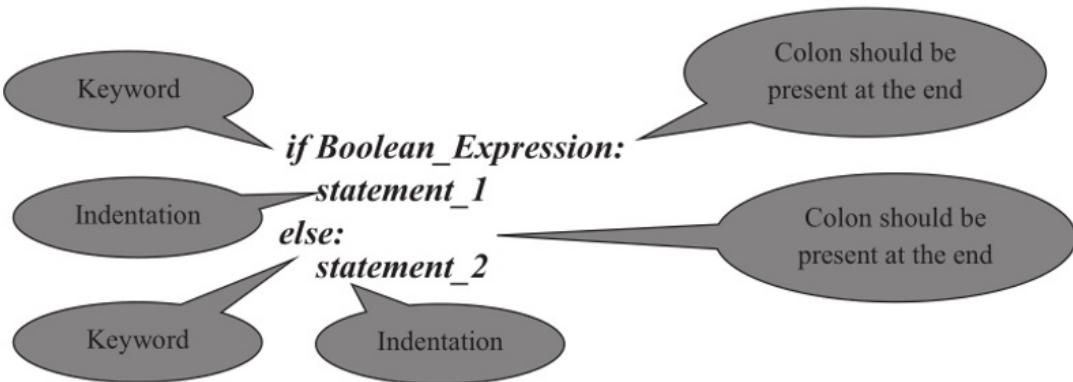
The number entered by user is a positive number

```
In [ ]: # Program to Read Luggage Weight and Charge the Tax Accordingly

weight = int(input("How many pounds does your suitcase weigh?"))
if weight > 50:
    print(f"There is a $25 surcharge for heavy luggage")
    print(f"Thank you")
```

There is a \$25 surcharge for heavy luggage
Thank you

The if...else Decision Control Statement



In []:

```
# Program to Find If a Given Number Is Odd or Even

number = int(input("Enter a number"))
if number % 2 == 0:
    print(f"{number} is Even number")
else:
    print(f"{number} is Odd number")
```

9 is Odd number

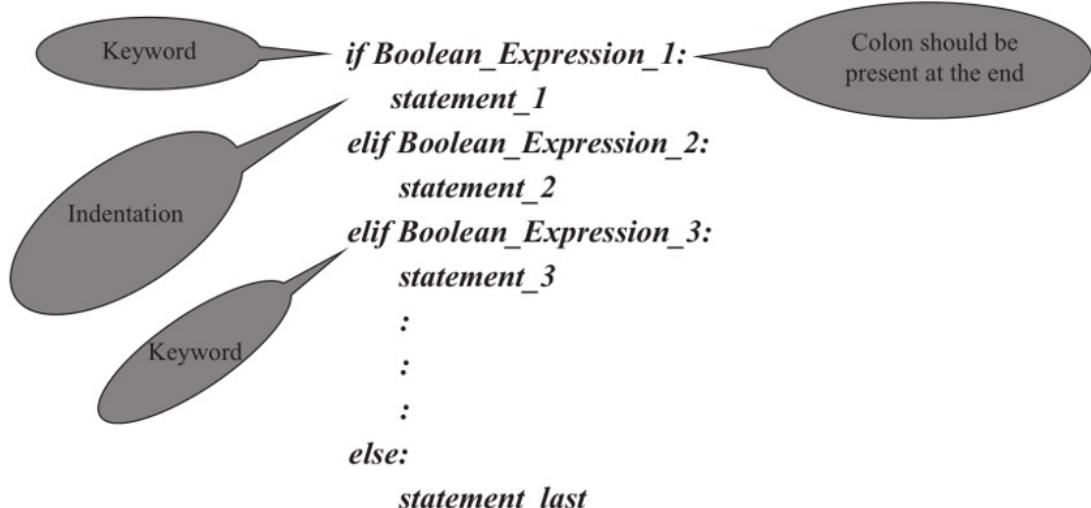
In []:

```
# Program to Find the Greater of Two Numbers

number_1 = int(input("Enter the first number"))
number_2 = int(input("Enter the second number"))
if number_1 > number_2:
    print(f"{number_1} is greater than {number_2}")
else:
    print(f"{number_2} is greater than {number_1}")
```

9 is greater than 5

if...elif...else Decision Control Statement



In []:

```
# Write a Program to Prompt for a Score between 0.0 and 1.0. If the
# Score Is Out of Range, Print an Error. If the Score Is between 0.0
# and 1.0, Print a Grade Using the Following Table
# Score  Grade
# >= 0.9  A
```

```

# >= 0.8    B
# >= 0.7    C
# >= 0.6    D
# < 0.6     F

score = float(input("Enter your score"))
if score < 0 or score > 1:
    print('Wrong Input')
elif score >= 0.9:
    print('Your Grade is "A" ')
elif score >= 0.8:
    print('Your Grade is "B" ')
elif score >= 0.7:
    print('Your Grade is "C" ')
elif score >= 0.6:
    print('Your Grade is "D" ')
else:
    print('Your Grade is "F" ')

```

Your Grade is "B"

In []:

```

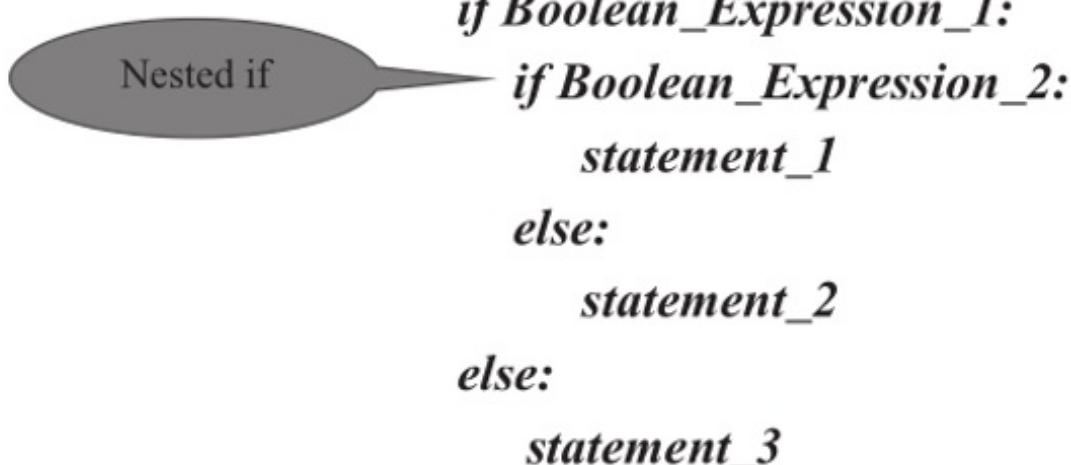
# Program to Display the Cost of Each Type of Fruit

fruit_type = input("Enter the Fruit Type:")
if fruit_type == "Oranges":
    print('Oranges are $0.65 a pound')
elif fruit_type == "Apples":
    print('Apples are $0.38 a pound')
elif fruit_type == "Bananas":
    print('Bananas are $0.42 a pound')
elif fruit_type == "Cherries":
    print('Cherries are $2.00 a pound')
else:
    print(f'Sorry, we are out of {fruit_type}')

```

Apples are \$0.38 a pound

Nested if Statement



In []:

```

# Program to Check If a Given Year Is a Leap Year

year = int(input('Enter a year'))

```

```

if year % 4 == 0:
    if year % 100 == 0:
        if year % 400 == 0:
            print(f'{year} is a Leap Year')
        else:
            print(f'{year} is not a Leap Year')
    else:
        print(f'{year} is a Leap Year')
else:
    print(f'{year} is not a Leap Year')

```

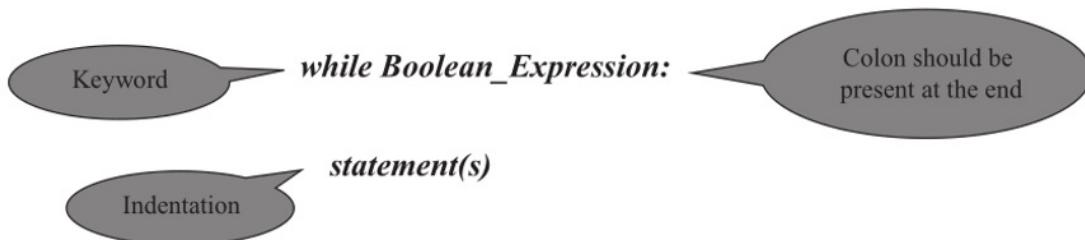
1600 is a Leap Year

All years which are perfectly divisible by 4 are leap years except for century years (years ending with 00) which is a leap year only if it is perfectly divisible by both 100 and 400.

For example, years like 2012, 2004, 1968 are leap years but 1971, 2006 are not leap years.

Similarly, 1200, 1600, 2000, 2400 are leap years but 1700, 1800, 1900 are not.

The while Loop



```

In [ ]:
# Write Python Program to Display First 10 Numbers Using
# while Loop Starting from 0

i = 0
while i < 10:
    print(f"Current value of i is {i}")
    i = i + 1

```

Current value of i is 0
 Current value of i is 1
 Current value of i is 2
 Current value of i is 3
 Current value of i is 4
 Current value of i is 5
 Current value of i is 6
 Current value of i is 7
 Current value of i is 8
 Current value of i is 9

```

In [ ]:
# Write a Program to Find the Average of n Natural Numbers
# Where n Is the Input from the User

number = int(input("Enter a number up to which you want to find the average"))
i = 0
sum = 0
count = 0
while i < number:
    i = i + 1 # it keeps track of count and increments as well
    sum = sum + i

```

```
average = sum/i
print(f"The average of {number} natural numbers is {average}")
```

The average of 5 natural numbers is 3.0

```
In [ ]: # Program to Find the GCD of Two Positive Numbers

m = int(input("Enter first positive number"))
n = int(input("Enter second positive number"))

if m == 0 and n == 0:
    print("Invalid Input")

if m == 0:
    print(f"GCD is {n}")

if n == 0:
    print(f"GCD is {m}")

while m != n:
    if m > n:
        m = m - n
    if n > m:
        n = n - m

print(f"GCD of two numbers is {m}")
```

GCD of two numbers is 4

```
In [ ]: # Write Python Program to Find the Sum of Digits in a Number

number = int(input('Enter a number'))

result = 0
remainder = 0

while number != 0:
    remainder = number % 10
    result = result + remainder
    number = int(number / 10)

print(f"The sum of all digits is {result}")
```

The sum of all digits is 6

```
In [ ]: # Write a Program to Display the Fibonacci Sequences up to nth
# Term Where n is Provided by the User

nterms = int(input('How many terms?'))
current = 0
previous = 1
count = 0
next_term = 0
if nterms <= 0:
    print('Please enter a positive number')
elif nterms == 1:
    print('Fibonacci sequence')
    print('0')
else:
    print("Fibonacci sequence")
    while count < nterms:
```

```

print(next_term)
current = next_term
next_term = previous + current
previous = current
count += 1

```

```

Fibonacci sequence
0
1
1
2
3

```

In []:

```

# Program to Repeatedly Check for the Largest Number Until the
# User Enters "done"

largest_number = int(input("Enter the largest number initially"))
check_number = input("Enter a number to check whether it is largest or not")

while check_number != "done":
    if largest_number > int(check_number):
        print(f"Largest Number is {largest_number}")
    else:
        largest_number = int(check_number)
        print(f"Largest Number is {largest_number}")

    check_number = input(
        "Enter a number to check whether it is largest or not")

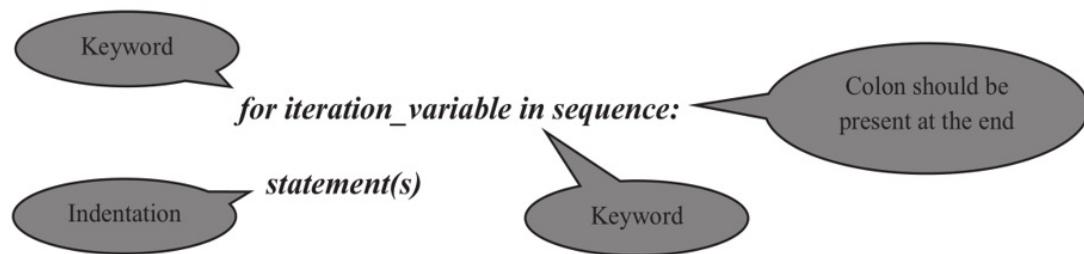
```

```

Largest Number is 5
Largest Number is 5
Largest Number is 8
Largest Number is 8
Largest Number is 9

```

The for Loop



The range() function

The range() function generates a sequence of numbers which can be iterated through using for loop.

The syntax for range() function is,

```
range([start ,] stop [, step])
```

Both start and step arguments are optional and the range argument value should always be an integer.

start → value indicates the beginning of the sequence. If the start argument is not specified, then the sequence of numbers start from zero by default.

stop → Generates numbers up to this value but not including the number itself.

step → indicates the difference between every two consecutive numbers in the sequence. The step value can be both negative and positive but not zero.

In []:

```
# Demonstrate for Loop Using range() Function
print("Only 'stop' argument value specified in range function")
for i in range(3):
    print(f"{i}")
print("Both 'start' and 'stop' argument values specified in range function")
for i in range(2, 5):
    print(f"{i}")
print("All three arguments 'start', 'stop' and 'step' specified in range function")
for i in range(1, 6, 3):
    print(f"{i}")
```

```
Only 'stop' argument value specified in range function
0
1
2
Both 'start' and 'stop' argument values specified in range function
2
3
4
All three arguments 'start', 'stop' and 'step' specified in range function
1
4
```

In []:

```
# Program to Iterate through Each Character in the String
# Using for Loop

for each_character in "Blue":
    print(f"Iterate through character {each_character} in the string 'Blue'")
```

```
Iterate through character B in the string 'Blue'
Iterate through character l in the string 'Blue'
Iterate through character u in the string 'Blue'
Iterate through character e in the string 'Blue'
```

In []:

```
# Write a Program to Find the Sum of All Odd and Even Numbers
# up to a Number Specified by the User.
```

```
number = int(input("Enter a number"))
even = 0
odd = 0
for i in range(number):
    if i % 2 == 0:
        even = even + i
    else:
        odd = odd + i

print(f"Sum of Even numbers are {even} and Odd numbers are {odd}")
```

Sum of Even numbers are 20 and Odd numbers are 25

```
In [ ]: # Write a Program to Find the Factorial of a Number

number = int(input('Enter a number'))
factorial = 1
if number < 0:
    print("Factorial doesn't exist for negative numbers")
elif number == 0:
    print('The factorial of 0 is 1')
else:
    for i in range(1, number + 1):
        factorial = factorial * i
print(f"The factorial of number {number} is {factorial}")
```

The factorial of number 5 is 120

The continue and break statements

The **break** and **continue** statements provide greater control over the execution of code in a loop.

Whenever the **break** statement is encountered, the execution control immediately jumps to the first instruction following the loop.

To pass control to the next iteration without exiting the loop, use the **continue** statement.

Both continue and break statements can be used in while and for loops.

```
In [ ]: # Program to Demonstrate Infinite while Loop and break

#n = 0
# while True:
#     print(f"The latest value of n is {n}")
#     n = n + 1

n = 0
while True:
    print(f"The latest value of n is {n}")
    n = n + 1
    if n > 5:
        print(f"The value of n is greater than 5")
        break
```

The latest value of n is 0
 The latest value of n is 1
 The latest value of n is 2
 The latest value of n is 3
 The latest value of n is 4
 The latest value of n is 5
 The value of n is greater than 5

```
In [ ]: # Write a Program to Check Whether a Number Is Prime or Not

number = int(input('Enter a number > 1: '))
prime = True
for i in range(2, number):
    if number % i == 0:
        prime = False
```

```
        break
if prime:
    print(f"{number} is a prime number")
else:
    print(f"{number} is not a prime number")
```

5 is a prime number

```
In [ ]: # Program to Demonstrate continue Statement
```

```
n = 10
while n > 0:
    print(f"The current value of number is {n}")
    if n == 5:
        print(f"Breaking at {n}")
        n = 10
        continue
    n = n - 1
```


KeyboardInterrupt

Traceback (most recent call last)

C:\Users\GOWRI\AppData\Local\Temp\ipykernel_17208\1483219944.py in <module>

$$3 \cdot n = 10$$

```
4 while n > 0:
```

```

----> 5     print(f"The current value of number is {n}")
       6     if n == 5:
       7         print(f"Breaking at {n}")

D:\anaconda3\lib\site-packages\ipykernel\iostream.py in write(self, string)
      527             is_child = (not self._is_master_process())
      528             # only touch the buffer in the IO thread to avoid races
--> 529             self.pub_thread.schedule(lambda: self._buffer.write(string))
      530             if is_child:
      531                 # mp.Pool cannot be trusted to flush promptly (or ever),

D:\anaconda3\lib\site-packages\ipykernel\iostream.py in schedule(self, f)
      212             self._events.append(f)
      213             # wake event thread (message content is ignored)
--> 214             self._event_pipe.send(b'')
      215         else:
      216             f()

D:\anaconda3\lib\site-packages\zmq\sugar\socket.py in send(self, data, flags, copy,
track, routing_id, group)
      489                     copy_threshold=self.copy_threshold)
      490             data.group = group
--> 491             return super(Socket, self).send(data, flags=flags, copy=copy, track=
track)
      492
      493     def send_multipart(self, msg_parts, flags=0, copy=True, track=False, **k
wargs):

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._send_copy()

D:\anaconda3\lib\site-packages\zmq\backend\cython\checkrc.pxd in zmq.backend.cython.
checkrc._check_rc()

KeyboardInterrupt:

```

Catching Exceptions Using try and except Statement

There are at least two distinguishable kinds of errors:

1. Syntax Errors
2. Exceptions

Syntax Errors

Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python. For example

In []:

```
while(True)
    print("Hello World!!)
```

```
File "C:\Users\GOWRIS~1\AppData\Local\Temp\ipykernel_17208/1028337131.py", line 1
```

```
  while(True)
^
```

```
SyntaxError: invalid syntax
```

Exceptions

Errors detected during execution are called exceptions.

By handling the exceptions, we can provide a meaningful message to the user about the issue rather than a system-generated message, which may not be understandable to the user.

Exceptions can be either built-in exceptions or user-defined exceptions.

In []:

```
10 * (1/0)
```

```
ZeroDivisionError Traceback (most recent call last)
C:\Users\GOWRIS~1\AppData\Local\Temp\ipykernel_17208/734848216.py in <module>
----> 1 10 * (1/0)
```

ZeroDivisionError: division by zero

In []:

```
4 + spam * 3
```

```
NameError Traceback (most recent call last)
C:\Users\GOWRIS~1\AppData\Local\Temp\ipykernel_17208/49873061.py in <module>
----> 1 4 + spam * 3
```

NameError: name 'spam' is not defined

In []:

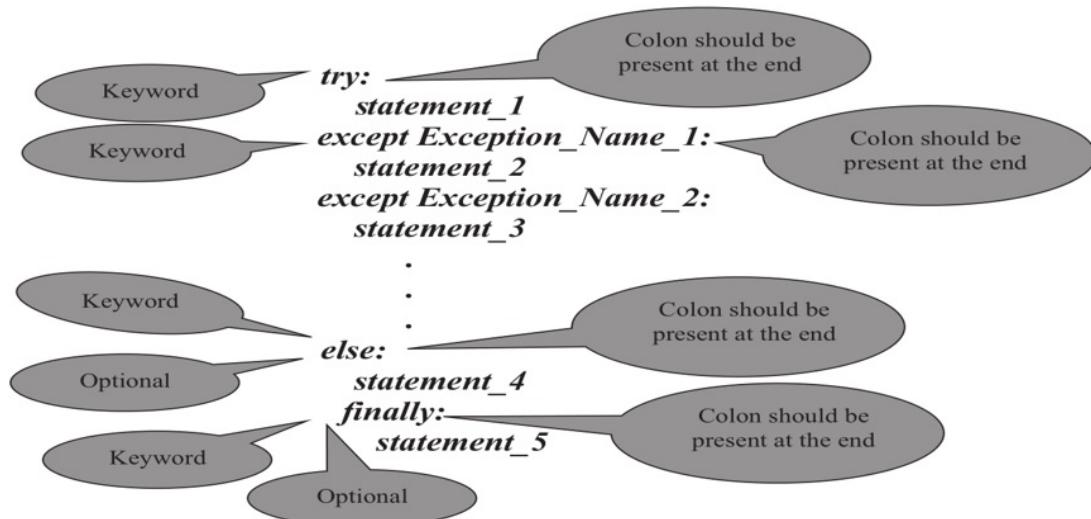
```
'2' + 2
```

```
TypeError Traceback (most recent call last)
C:\Users\GOWRIS~1\AppData\Local\Temp\ipykernel_17208/209420021.py in <module>
----> 1 '2' + 2
```

TypeError: can only concatenate str (not "int") to str

Handling of exception ensures that the flow of the program does not get interrupted when an exception occurs which is done by trapping run-time errors.

try...except...finally



```
In [ ]: # Program to Check for ValueError Exception
```

```
while True:  
    try:  
        number = int(input("Please enter a number: "))  
        print(f"The number you have entered is {number}")  
        break  
    except ValueError:  
        print("Oops! That was no valid number. Try again...")
```

Oops! That was no valid number. Try again...
The number you have entered is 5

```
In [ ]: # Program to Check for ZeroDivisionError Exception
```

```
x = input("Enter value for x: ")  
y = input("Enter value for y: ")  
try:  
    result = int(x) / int(y)  
except ZeroDivisionError:  
    print("Division by zero!")  
else:  
    print(f"Result is {result}")  
finally:  
    print("Executing finally clause")
```

Division by zero!
Executing finally clause

```
In [ ]:
```

```
# Write a Program Which Repeatedly Reads Numbers Until the User  
# Enters 'done'. Once 'done' Is Entered, Print Out the Total, Count, and Average of  
# Numbers. If the User Enters Anything Other Than a Number, Detect Their Mistake  
# Using try and except and Print an Error Message and Skip to the Next Number
```

```
total = 0  
count = 0  
while True:  
    num = input("Enter a number: ")  
    if num == 'done':  
        print(f"Sum of all the entered numbers is {total}")  
        print(f"Count of total numbers entered {count}")  
        print(f"Average is {total / count}")  
        break  
    else:  
        try:  
            total += float(num)  
        except:  
            print("Invalid input")  
            continue  
    count += 1
```

Sum of all the entered numbers is 15.0
Count of total numbers entered 5
Average is 3.0

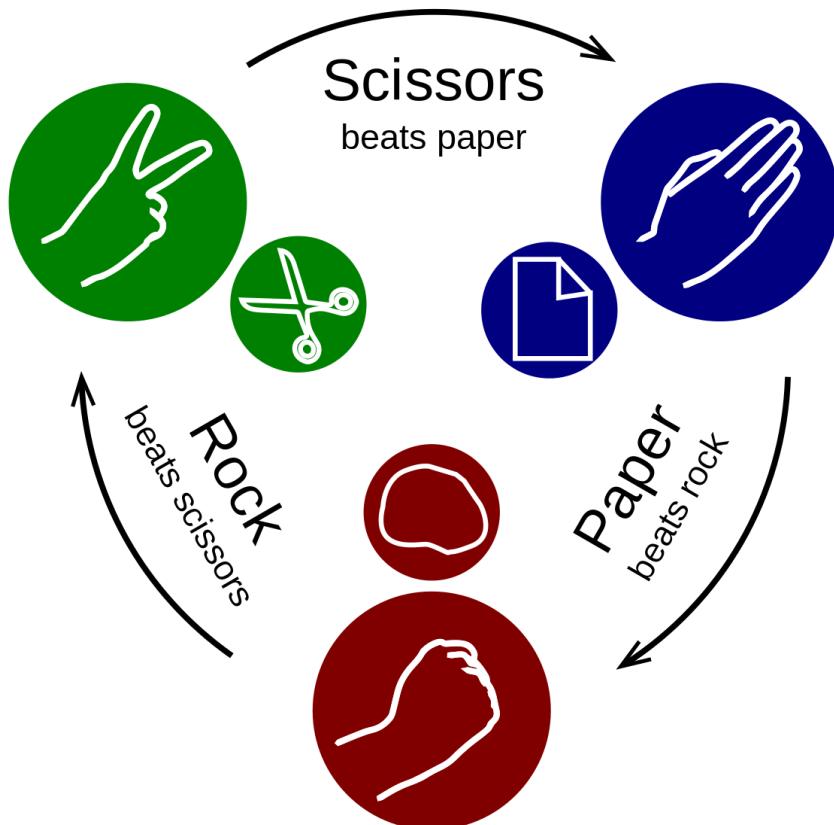
Case Study

Devise a Python program to implement the Rock-Paper-Scissor game.

rock-paper-scissors (also known as scissors-rock-paper or other variants) is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand.

These shapes are "rock" (a closed fist), "paper" (a flat hand), and "scissors" (a fist with the index finger and middle finger extended, forming a V).

"Scissors" is identical to the two-fingered V sign (also indicating "victory" or "peace") except that it is pointed horizontally instead of being held upright in the air.



A simultaneous, zero-sum game, it has only two possible outcomes: a draw, or a win for one player and a loss for the other.

A player who decides to play rock will beat another player who has chosen scissors ("rock crushes scissors" or sometimes "blunts scissors"), but will lose to one who has played paper ("paper covers rock").

A play of paper will lose to a play of scissors ("scissors cuts paper").

If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie.

In the program, the user gets the first chance to pick the option among rock, paper and scissor. After that the computer selects from the remaining two choices (randomly), then winner is decided as per the rules.

Winning Rules as follows :

```
rock vs paper-> paper wins
rock vs scissor-> rock wins
paper vs scissor-> scissor wins
```

In this game, the `randint()` inbuilt function is used for generating random integer value within the given range.

In []:

```
# import random module
import random

# Print multiLine instruction perform string concatenation of string
print(
    "Winning Rules of the Rock paper scissor game as follows: \n"
    + "rock vs paper -> paper wins \n"
    + "rock vs scissor -> rock wins \n"
    + "paper vs scissor -> scissor wins \n"
)

while True:
    print("Enter choice \n 1. rock \n 2. paper \n 3. scissor \n")
    # take the input from user
    choice = int(input("User turn: "))

    # OR is the short-circuit operator
    # if any one of the condition is true
    # then it return True value

    # Looping until user enter invalid input
    while choice > 3 or choice < 1:
        choice = int(input("Enter valid input: "))

    # initialize value of choice_name variable
    # corresponding to the choice value
    if choice == 1:
        choice_name = "rock"
    elif choice == 2:
        choice_name = "paper"
    else:
        choice_name = "scissor"

    # print user choice
    print(f"User choice is: {choice_name}")
    print("\nNow it is Computer's turn.....")

    # Computer chooses randomly any number among 1 , 2 and 3. Using randint method o
    comp_choice = random.randint(1, 3)

    # Looping until comp_choice value is equal to the choice value
    while comp_choice == choice:
        comp_choice = random.randint(1, 3)

    # initialize value of comp_choice_name variable corresponding to the choice val
    if comp_choice == 1:
        comp_choice_name = "rock"
    elif comp_choice == 2:
        comp_choice_name = "paper"
    else:
        comp_choice_name = "scissor"

    print(f"Computer choice is: {comp_choice_name}")
```

```

print(f"{choice_name} V/s {comp_choice_name}")

# condition for winning
if (choice == 1 and comp_choice == 2) or (choice == 2 and comp_choice == 1):
    print("paper wins => ", end="")
    result = "paper"
elif (choice == 1 and comp_choice == 3) or (choice == 3 and comp_choice == 1):
    print("rock wins => ", end="")
    result = "rock"
else:
    print("scissor wins =>", end="")
    result = "scissor"

# Print as either user or computer wins
if result == choice_name:
    print("<== So User wins ==>")
else:
    print("<== So Computer wins ==>")

print("Do you want to play again? (Y/N)")
ans = input()

# if user input n or N then condition is True
if ans == "n" or ans == "N":
    break

# After coming out of the while loop we print thanks for playing
print("\nThanks for playing")

```

Winning Rules of the Rock paper scissor game as follows:

rock vs paper -> paper wins
 rock vs scissor -> rock wins
 paper vs scissor -> scissor wins

Enter choice

1. rock
2. paper
3. scissor

User choice is: rock

Now it is Computer's turn.....

Computer choice is: scissor

rock V/s scissor

rock wins =><== So User wins ==>

Do you want to play again? (Y/N)

Enter choice

1. rock
2. paper
3. scissor

User choice is: paper

Now it is Computer's turn.....

Computer choice is: rock

paper V/s rock

paper wins => <== So User wins ==>

Do you want to play again? (Y/N)

Enter choice

1. rock
2. paper
3. scissor

User choice is: scissor

Now it is Computer's turn.....
Computer choice is: paper
scissor V/s paper
scissor wins =><== So User wins ==>
Do you want to play again? (Y/N)
Enter choice
1. rock
2. paper
3. scissor

User choice is: paper

Now it is Computer's turn.....
Computer choice is: rock
paper V/s rock
paper wins => <== So User wins ==>
Do you want to play again? (Y/N)

Thanks for playing