

Programming language

A programming language is a formal notation used for writing instructions to carry out some task on a computer. The sequence of instructions to carry out some specific task on a computer system, is called a program. A computer can understand a very limited and basic instructions, in each of its instruction. However, we can combine a number of instructions together to make a computer to perform complex tasks. A program is formed by combining many instructions of a computer.

Generations of Programming languages:

1. First Generation - The language understood by a computer system without the need of any translation is called first generation programming language (1GL). The set of instructions understood by a computer depends upon the CPU inside the computer. The machine instructions are in the format of binary codes.

Advantages of 1GL:

1. It requires no translation.
2. Programs are very efficient.

Disadvantages of 1GL:

1. Very difficult to write and debug programs.
2. Programs written in machine language are not portable to other computers with different CPU type.

2 Second Generation languages : (2GL)

The second generation of programming languages are called Assembly language. Assembly language make use of mnemonics for machine instructions and names for memory locations & CPU registers. The names and mnemonics are in simple english words and hence are easy to remember. A program written in assembly language must be translated into the machine language so that it can be executed by the computer. An assembler is a system software that translates an assembly language program to machine language.

Advantages of 2GL :

1. Easy to write and debug programs as compared to machine language.
2. Program written in assembly language are very efficient.

Disadvantages of 2GL :

1. It requires translator (assembler).
2. Each CPU type has its own assembly language, hence programs are not portable.

21 Sep 2021

3 Third generation language [3GL] [High level language]

Third generation programming language are also known as high-level languages (HLL). Because they are independent of the CPU used in a computer system and are portable across different hardware platforms. 3GLs use natural language like notations to write programs and popular mathematical notations to write programs. Hence a program written in HLLs are easy to write and understand. A program written in 3GL must be

translated into machine language before execution.

There are two methods of translating a HLL program into machine language:

- (a) Using Interpreter: An interpreter is a system software that translates the source code of a program into machine language in a statement-by-statement mode and executes each statement before going to the next statement of the program.

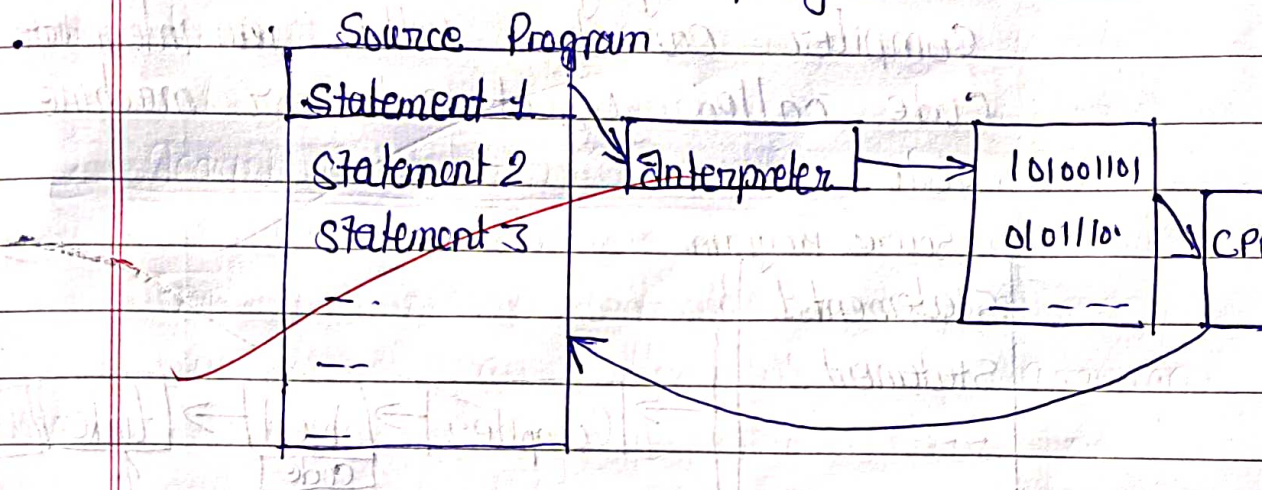


fig- WORKING OF INTERPRETER

Advantages of using Interpreter:-

- (i) Statements are executed immediately and we need not to write entire program before executing a statement.
- (ii) Interpreted languages are very secure as compared to compiled program.
- (iii) Interpreted languages are highly portable.

Disadvantages of interpreter:

- (i) The programs written in interpreted languages are not very efficient.
- (ii) A program may fail at a statement i.e. not interpreted yet.

(b) Using Compiler: A compiler is a system software that translates the whole HLL program into machine language. The output of a compiler can be either an intermediate code called object code or machine code.

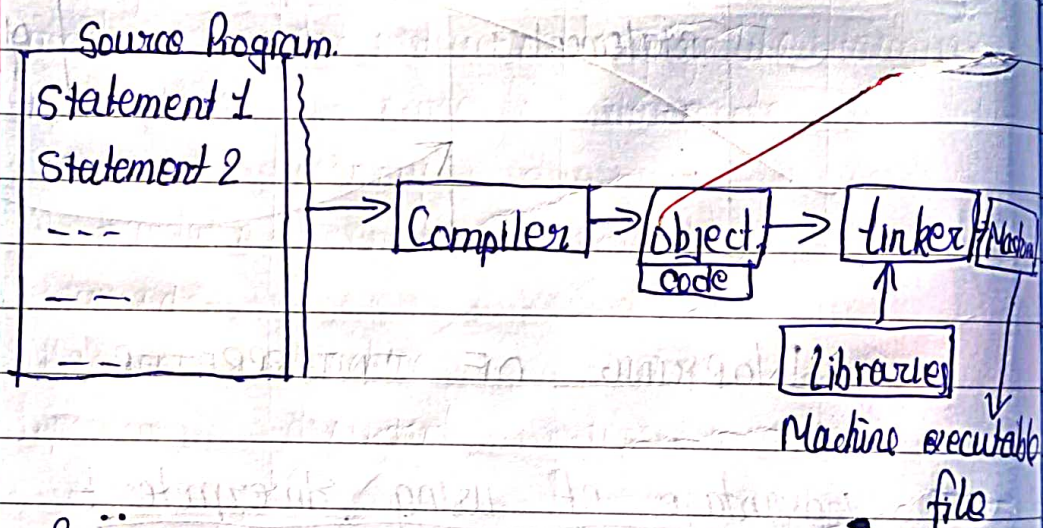


fig: WORKING OF COMPILER

Advantages of Compiler:

1. Entire program is checked before translation to machine code.
2. The machine code once produced

can be executed many times without the requirement of source code.

- (iii) Compiled programs are more efficient to execute as compared to interpreted programs.

Disadvantages of Compiler:

(i) X [The source program is required to be compiled into the machine code of each hardware platform for ex]

1. A compiler is required for each hardware platform to convert the source program to the machine code of that hardware platform.

2. The machine code file produced by the compiler can be easily infected by viruses.

4 Fourth Generation Languages (4GL)

The fourth generation programming languages are focused on "what to do?" rather than "How to do?" A 4GL does not require a detailed description of how to carry out some task but it focused on what is actually required to be achieved.

Example of 4GL is SQL structured

Query language] In SQL we only write commands to achieve the required output and detail of producing the required result is carried out by the interpreter.

22 Sep.

Generation of Programming languages

1 st Generation 1GL	2 nd Generation 2GL	3 rd Generation 3GL	4 th Generation 4GL
→ Machine language	Assembly language	high-level language	High level language
→ Binary coded machine instructions	Mnemonics for instructions and names for memory.	Natural language like statements	Natural language like statements.
→ No translation required	Assembler	Compiler/Interpreter	Interpreter
→ Machine dependent.	Machine dependent	Machine independent Example - C, C++, COBOL, Python	Machine independent SQL

SYNTAX

The syntax the set of huge rules used to write programs in a programming

language.

Or

Syntax is the set of rules that must be followed to write programs in a programming language.

✓ Syntax Error: Any violation of syntax in a program is called syntax error. If a program contains a syntax error, the compiler / interpreter will show an error message and stop translation when there is a syntax error in a program.

✓ Semantic Error [Bug]: A program is said to be containing semantic error if it is not producing the expected output. A semantic error is the result of careless programming. The translator does not detect the semantic error in a program. It is difficult to locate and fix semantic errors in a program. The systematic approach of locating and fixing semantic error [bug] is called debugging.

Debugging: A program is said to be containing bugs if it is not producing the expected result. A systematic approach of locating and fixing bugs

in a program is called debugging. Source code tracing and using of break points are two common methods of locating bugs in a program.

Documentation

Documentation is written text and illustration of a software product that describes the design / architecture working or implementation details of that software. Various types of software documentation are:

1. End User documentation: This type of documentation contains the detailed description of working / operation of software product. It contains the type of input and the output produced by the software. User manual is an example of end-user documentation. A user manual is provided with a software product either as a hard copy or soft copy. It also contains the instruction for installation and configuration of the software.

2. Technical documentation: Technical documentation of a software contains

the detailed description of architecture / design and implementation of a software product, it also contains the application programming interface (API) details of the software. Technical documentation can be embedded inside the source code of the software in the form of documentation comments.

Algorithm

An algorithm is a step-by-step description for solving a problem. An algorithm is written in simple English or other familiar notations and is independent of any programming language. Each step of an algorithm must carry out a very basic operation and should be very precise.

An algorithm is a sequence of steps to solve a problem. The steps of an algorithm are simple and written in English or well known mathematical notations. Each step of an algorithm is then written as statements in a program. computer program. The algorithm is independent of any programming language. So when an algorithm is written, it can be implemented as a program in any programming language.

Example of an algorithm :

Algorithm sum

Step 1 : Read first number as a

Step 2 : Read second number as b

Step 3 : Add a and b and store the result in $c \leftarrow a + b$

Step 4 : Print the sum as C

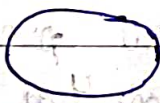
Step 5 : Stop

Flowchart

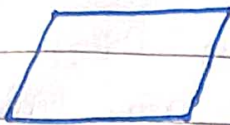
A flowchart is a diagram showing the sequence of steps to solve a problem.

Or, a flowchart is a diagram representing an algorithm. A flowchart is easy to understand as compared to an algorithm. A flowchart uses standard symbols to represent different types of operations. These symbols are then connected using arrows to represent the flow of execution of steps.

Various symbols used in flowchart are:

1.  : Start / stop : [Ellipse/oval]
Each flowchart must begin with a start symbol and end with a stop symbol.

2.



: Read [input / output] [parallelogram]
The symbol used to read input or write output.

3.



: Process [simple rectangle]
Symbol used to represent processing like calculations.

4.



: Decision : [Diamond]
A decision is used to check a condition if the condition is true then the arrow labeled ~~is~~ true is selected for execution and if the condition is false then the arrow labelled ~~is~~ false is selected for execution.

5.

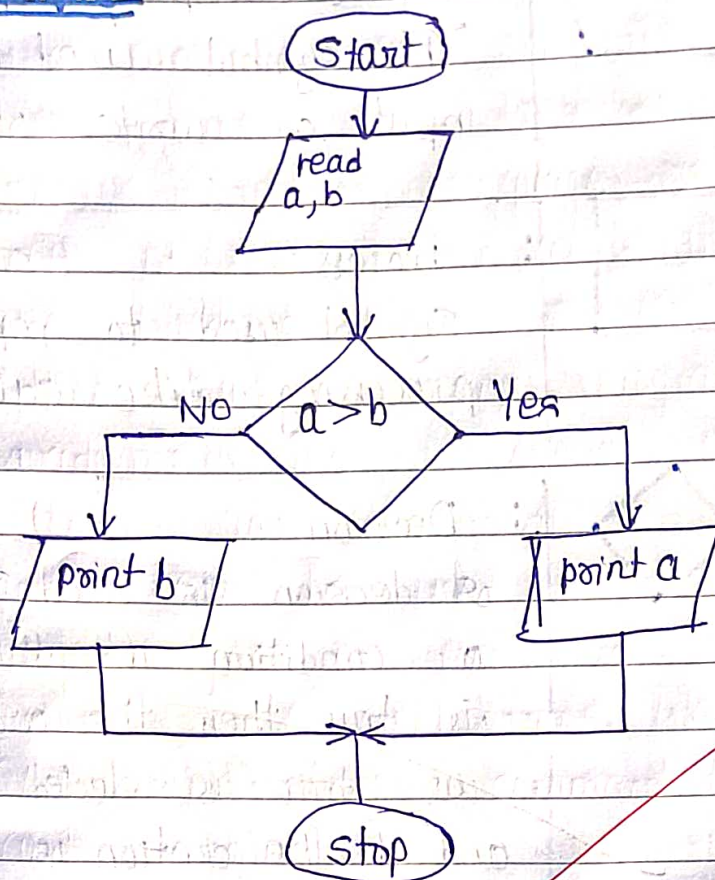


Connector : [Circle]
A big flowchart may not fit on a single page to continue the flowchart on the next page connectors are used. Each connector is labelled and the label the flowchart continues matching connectors on the next page.

6.



flow of execution : Arrows are used to represent the flow of execution of steps.

Example :

flowchart : Greatest of two numbers.

UNIT - 2

Getting started with Python.

Python

Python is a high-level computer programming language developed by Guido van Rossum in 1991.

⇒ Main features of Python language are:

- (i) Python is an interpreted language. The statements in python are executed immediately when they are written.
- (ii) Python Programs written in python are easy more readable.
- (iii) The syntax of python is easy as compare to other languages like C, C++ and JAVA.
- (iv) Complex problems can be solved in Python using fewer lines of code.
- (v) Python provides a very rich standard library to carry out routine operations.
- (vi) Python is available free of cost for all popular operating systems like Windows, Linux, MacOS.
- (vii) Python has a very strong community support.
- (viii) Python supports many programming paradigms procedural, object-oriented and functional.

Need of Python languages:

1. Python language has simple syntax and

fewer lines of code are required to write programs as compared to other programming languages.

(ii) Python interpreter language is available free of cost for all platforms like Windows, Linux, MacOS

(iii) Python support many programming paradigms like procedural, object oriented and functional.

(iv) Python can be used to develop many types of applications like: Desktop, GUI applications, Web applications, CLI applications, [command line interface] and scripting.

(v) Python has a very rich standard library and very third party libraries.

(vi) Python is a very suitable language for data analytics, machine learning, artificial intelligence and embedded programming.

(vii) Python is most popular programming language with very strong community of developers.

Advantages of Python language:

1. Simple Syntax: Python has a very simple syntax as compared to other programming languages.

like C and Java.

(ii) Fewer lines of Code : A program written in python has fewer lines of code as compared to the same program written in other languages.

(iii) Readability : Python makes use of code indentation to improve readability of programs. Python programs are easy to understand as compared to other languages.

(iv) Portability : A Python program can be ported to any platforms like Windows, Linux, macOS.

(v) Free of cost : The language Python language is available for free of cost.

(vi) One language for all programming needs : Python can be used to develop terminal applications, Desktop applications, Web applications, scripting, and embedded applications.

(vii) Rich libraries : Python has a rich set of standard libraries and

supported by strong developer communities.

~~29/9/21~~



Install and Configure Python and IDLE

Python can be downloaded from www.python.org/downloads and after downloading the correct version, on the correct operating system, Python can be installed by launching the downloaded installer.

IDLE [Integrating development & learning environment] is a development environment for Python. IDLE is also bundled with the Python installer and is installed with Python.

✓ IDLE [Integrating development and learning environment]

IDLE is a Python integrated development environment that provides a graphical interface to carry out different program activities like coding, formatting, debugging and executing. Various features of IDLE are:

- ① IDLE supports all programme development related activities like coding, formatting, debugging and running using a graphical user interface.
- ② IDLE is fully developed using Python language.
- ③ IDLE is available for all platforms like Windows, Linux, macOS.

- ④ IDLE provides two modes: interactive shell and editor. Interactive shell is used to execute Python statements as soon as they are typed. Editor provides the facility to write Python program.

Menu Structure:

IDLE has a menu-driven interface in which different functions are provided as menu options. Various menu options in IDLE are:

⇒ File menu: Various options under file menu are:

- ① New file: Create a new file in editor window.
- ② Open: ... : Open an existing Python file in editor window.
- ③ Recent files: provides a list of recently edited files. We can click on one of these files to open.
- ④ Open Module: Open an existing module.
- ⑤ Save: To save a file on storage device.

- (vi) Save as : Save the currently opened file in different file name
- (vii) Close : Close the current window
- (viii) Exit : Exit from IDE

Edit Menu

Various options under Edit menu are:

- (i) Undo : Revert the changes made to the current window. A maximum of 1000 changes can be undone.
- (ii) Redo : Revert the change after applying undo.
- (iii) Cut : Copy the selected text on the clipboard & delete the selected text.
- (iv) Copy : Copies the selected text to the clipboard.
- (v) Paste : Copy the text from the clipboard at the cursor location in the window.
- (vi) Select all : Select the all text in the window.

(vii) Find: Find a given text in the window.

(viii) Replace: Find a given text in the window and replace it with other text.

(ix) Go to line: Scroll the window and move the cursor to a given line number.

(x) Show surrounding Parenthesis: Highlight the matching parentheses.

3 Oct.

Format [Editor window]

(i) Indent Region: Shift the selected statements towards right by 4 spaces.

(ii) Redent Region: Shift the selected statements towards left by 4 spaces.

(iii) Comment Out region: Insert # # in front of selected lines.

(iv) Uncomment Region: Remove # or # #

... from the beginning of selected lines.

(v) Tabify Region : Replace the space character from the beginning of selected lines to tabs [1 tab. for 4 spaces]

(vi) Untabify : Replace tab character from the beginning of selected lines with spaces (4 spaces for 1 tab)

(vii) New indent width : Change width of indentation [default 4 spaces]

(viii) Format paragraph : Reset the selected paragraph to 72 character width

(ix) Run [Editor Window] : ~~Run~~ no Run Module : Check the module for errors and if no error is found the module is executed. The execution environment is cleaned before execution.

2- Run... Customized : Check the module for errors
We can: page command line

arguments and prevent restarting of shell before execution of the module

③ Check Module : Check the syntax of the module

④ Python Shell : Open the interactive python shell

Shell Menu [shell window]

(i) View last restart : Scroll the shell window to the line where the shell was recently restarted.

(ii) Restart shell : Reset the shell to clear the environment

(iii) Previous History : Show the previous command executed in the shell.

(iv) Next History : Show the next command in the shell history.

(v) Interrupt Execution : Stop executing

in a program.

(vi) Debug [shell window]:

(i) Go to File / line : Open the file and move the cursor to the line as specified in the syntax error.

(ii) Debugger: When debugger is opened the commands executed in the shell or the module executed from editor window were run using debugger. We can set breakpoints and view various variables using debug windows.

(iii) Stack viewer: View the stack of Function execution

Options [shell and editor]:

(i) Configure IDE : Using this option, we can customize the IDE and change preferences of: Font, indentation, keybindings, text colors and themes, window size etc.

(ii) Config Show/Hide Line Numbers
[Editor]: Show line numbers of statements in editor window or hide them.

(iii) Zoom / restore Height ÷ Toggle
the height of the window b/w normal size and maximum size.

(iv) Window + lists the opened windows. We can select window from the list to bring it to foreground.

Help Menu

(i) About IDLE: Display version, copyright and credits of IDLE.

(ii) IDLE help: Display the IDLE menu options and other tips.

(iii) Python Docs: Access local python documentation.

(iv) Turtle demo: Run the turtle

demo with turtle drawing

Format [Editor]

- indent
- Dedent
- Comment out region
- Uncomment
- Tabify
- untabify
- New indent width
- Format paragraph

Run [Editor]

- Run Module
- Run -- Customized run
- Check Module
- Python Shell

~~Shell menu~~

- View last restart
- Restart shell
- Previous history
- Next history
- Interrupt execution

Debug

- Debugger
- Stack viewer

Options

- Configure IDLE
- Show/hide navbar
- Zoom/restore height
- Window

Help menu

- About IDLE
- IDLE help
- Python Docs
- Turtle Demo

window

- list of opened window.

5 Oct

Python Interactive shell

Python interactive shell provides a command line interface that prompts the user to enter type python command / statement. The statement is interpreted and executed immediately when user press the "Enter" key.

Python interactive shell can be invoked using the command python. at the os command prompt. When the interactive shell is opened, it displays the python version information in the first line.

On the second line some example commands are listed.

The third line displays the python prompt where a user can type python code.

Example of using python interactive shell:

```
>>> "Hello Python." ↵
```

```
'Hello Python.'
```

↵ No command entered

The output produced by the user by python.

```
>>> 1000 ↵
```

```
1000
```



```
>>> 1000 * 5 + 7 * 3
5021
```

If an expression is typed on the shell prompt, it is evaluated by the interpreter and the result is displayed.

```
>>> 5 * (2 + 7) * 3 + 5 * 4
155
```

When a python statement is typed at the prompt it is executed by the python interpreter.

```
>>> print("Hello python!")
Hello python!
```

```
>>> for i in range(1, 10):
    print(i)
```

```
>>> for i in range(1, 11):
    print("7 x {} = {}".format(i, 7*i))
```

To exit from interactive shell type `exit()` or press ~~ctrl~~ + d

```
>>> exit()
```

Python Tokens

A python program consists of a sequence of statements. The statements consist of a valid combination of

tokens. A token is a lexical unit of a statement.

Various types of tokens in Python are: Identifier, keywords, Operators, constants etc.

Python keywords: A keyword (reserved word) has a predefined meaning in a language and cannot be used for an identifier.

Various keywords in python are: import, if, elif, else, for, break, continue, pass, while, with, try, except, finally, raise, class, del, lambda (λ), in, is, return, from etc.

Python Identifier: An identifier is a name given to some program element e.g. variable, function, class, label etc. The rules to create an identifier in python are:

- (1) The first character of an identifier must be a letter (a-z, A-Z) or an underscore (_).
- (2) The remaining characters of an identifier must be either from letters (a-z, A-Z), underscore (_), or digits (0-9).

③ The name of an identifier must be unique within its scope.

④ Python reserved words can not be used as identifiers.

Examples of identifiers in Python:

x: valid

x-y: Invalid [- is not allowed]

x_y: Valid

123: Invalid [An identifier must begin with a letter or underscore]

A23: Valid

for: Invalid [for is a python keyword]

invalid: Valid

Valid id: Invalid [space character is

: Invalid not allowed in identifier]

/*#* space invalid [special characters are not valid]

name: valid

Arithmetic Operators

An operator is a special symbol or keyword that represents some standard operation.

The values on which an operator is applied are called operands.

Various types of operators in Python are:

⇒ An operator always returns a single value after performing.

Arithmetic Operators

Arithmetic operations are used to perform basic mathematical operations. The operands of arithmetic operators are always numeric. The value returned by an arithmetic operator is also numeric.

Operator	Operation	Description	Example
+	Addition	+ operator returns the sum obtained by adding the value of its two operands	$\ggg 10 + 3$ 13 $\ggg 5.9 + 3.7$ 9.6 $\ggg -10 + 5$ -5
-	Subtraction	- operator returns a value obtained after subtracting the value of second operand from the value of its first operand	$\ggg 10 - 3$ 7 $\ggg 3 - 10$ -7 $\ggg 10.5 - 5.2$ 5.3
*	Multiplication	* operator returns the product obtained by multiplying the values	$\ggg 5 * 3$ 15

		of its first or two operands	$\ggg -10 * 3.2$ -32
/	Division	/ operator returns the value obtained by dividing the value of its operand by the value of its second operand	$\ggg 5/2$ 2.5 $\ggg 6.68/3$ 2.21 $\ggg .9/3$ -3
//	floor Division	// operator returns the value obtained by dividing its first operand and truncating the fractional part	$\ggg 10//2$ 5 $\ggg 10//3$ 3 $\ggg 10/9$ 1
%	Modules	% operator returns the remainder obtained after dividing its first operand by its second operand	$\ggg 10 \times 9$ 1 $\ggg 9 \times 10$
**	Exponentiation [raise to power]	** returns a value obtained by raising the value of its first operand to the power of the value of its second operand	$\ggg 2 ** 10$ 1024 $\ggg 25 ** 0.5$ 5

Python Operators

- 1) Arithmetic operators (+, -, *, /, //, %, **)
- 2) Comparison (Relational) operators :

Comparison operators are used to compare two values. If the comparison holds true is returned otherwise false is returned. Comparison operators in Python are :

Operator	Operation	Description	Example
<	less than	< operator returns true if the value of its first operand is less than the second operand, otherwise it returns false	<pre>>>> 5 < 7 True >>> 7 < 5 False >>> 5 < 5 False</pre>
>	Greater than	> operator returns true if the value of its first operand is greater than its second operand, otherwise it returns false	<pre>>>> 5 > 7 False >>> 7 > 5 True >>> 5 > 5 False</pre>

<= less than
or
equal to

<= operator returns true if the value of its first operand is either less than or equal to the value of its second operand, otherwise it returns false.

>>> 5 <= 7
True

>>> 7 <= 5
false

>>> 5 <= 5
True

>= Greater than
or
Equal to

>= operator returns ^{true} if the value of its first operand is either less than or equal to the value of its second operand, otherwise it returns false.

>>> 5 >= 7
false

>>> 7 >= 5
True

>>> 5 >= 5
True

>>> 5 >= 5
True

>>> 5 >= 5
True

>>> 5 >= 5
True

== Equal To

== operator returns true if the value of its both operands is equal.

>>> 5 == 7
false

>>> 7 == 5
false

>>> 5 == 5
True

>>> 5 == 5
True

!= Not Equal

!= operator returns true if the value of its operands are different otherwise it returns false.

>>> 5 != 7
True

>>> 7 != 5
True

>>> 5 != 5
false

>>> 5 != 5
false

>>> 5 != 5
false

(ii) logical operators $\Rightarrow$ logical operators are used to perform logical operations on boolean values [True/False]. logical operators in python are:

operator	Operation	Description	Example															
and	logical AND	and operator returns true if the value of its both operands is true otherwise it returns false. Truth table	$\ggg 5 < 7$ and $\ggg 3 > 4$ True False False															
		<table><tr><th>a</th><th>b</th><th>a and b</th></tr><tr><td>false</td><td>false</td><td>false</td></tr><tr><td>false</td><td>True</td><td>false</td></tr><tr><td>True</td><td>false</td><td>false</td></tr><tr><td>True</td><td>True</td><td>True</td></tr></table>	a	b	a and b	false	false	false	false	True	false	True	false	false	True	True	True	
a	b	a and b																
false	false	false																
false	True	false																
True	false	false																
True	True	True																
or	logical OR	or operator returns false if the value of its both operands is false otherwise it returns True. Truth table for an operator	$\lll 5 < 7$ or $\ggg 3 > 4$ True False True															
		<table><tr><th>a</th><th>b</th><th>a or b</th></tr><tr><td>false</td><td>false</td><td>false</td></tr><tr><td>false</td><td>True</td><td>True</td></tr><tr><td>True</td><td>false</td><td>True</td></tr><tr><td>True</td><td>True</td><td>True</td></tr></table>	a	b	a or b	false	false	false	false	True	True	True	false	True	True	True	True	
a	b	a or b																
false	false	false																
false	True	True																
True	false	True																
True	True	True																

		true	false	false	
		true	true	true	
not	logical Not	not operators return the opposite truth value of its operand. truth table of not operators is			not True False
		a	not a		
		false	true		
		true	false		

(iv) Bitwise Operators: Bitwise operators are used to manipulate the bits of its operands. Bitwise operators can be applied to integer operands only. Various bitwise operators in python are:

Operator	Operation	Description	Example
&	Bitwise AND	& operator sets a bit of its resultant integer to 1 if corresponding bits in both of its operands are 1 otherwise the bit is set to 0.	>>> a = 15 >>> b = 9 >>> a & b 9 >>> 50 & 60 48 >>> 15 & 16 0

|

Bitwise
OR

| operator set a bit of its resultant integer to 1 if corresponding bit in both of its operands are 1, otherwise the bit is set to 0

>>> ~~15~~ a = 15

>>> b = 9

>>> a | b

15

>>> ~~a~~ = 50

>>> b = 60

62

>>> a = 15

>>> b = 16

>>> a | b

31

^

Bitwise
EXOR

^ operator sets a bit of its resultant integer to 1 if corresponding bits in both of its operands are same otherwise the bit is set to 0

>>> a = 15

>>> b = 9

>>> a ^ b

6

>>> ~~a~~ = 16

>>> 16 ^ 15

31

~

Bitwise
NOT

~ operator returns an integer obtained by complementing the bits of its operand.

>>> ~ 10

-11

>>> ~ 8

-9

>>> ~ 25

-26

<<	Bitwise left shift	<< operator returns an integer obtained by shifting the bits of its operand towards left by a specified number of positions. The op left shift by one position is equivalent to multiplication by 2.	>>> 10 << 2 40 90 << 2 360
>>	Bitwise right shift	>> operator returns an integer obtained by shifting the bits of its operand towards right by a specified no. of position. Right shift by one position is equivalent to division by 2. The bits created at left are filled with the sign bit.	>>> 10 >> 2 2 90 >> 2 22 >>> -90 >> 2 -23 -73 >> 2 -19

~~14/10~~

→ Arithmetic (+, -, *, /, %, **, *)

Comparison (<, <=, >, >=, ==, !=)

Logical (and, or, not)

Bitwise (&, ^, |, ~, <<, >>)

Assignment operators:

Assignment operator is used to set the value of a variable to a given value. The expression on the right hand side of assignment operator is evaluated first and the result obtained is stored in the variables specified to the left hand side of the operator. The first operand of assignment operator is always a variable. There are two type of assignment operators in Python.

⇒ Simple Assignment (=)

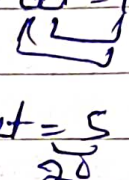
Component Assignment Operator =

= +, -, *, /, %, **, &, ^, |, ~, <<, >>

(ii) Simple Assignment (=) ⇒ operator evaluates the expression at its right hand side and stores the obtained value in the variables at its left hand side for example.

Assignment operators:

⇒

operator	operation	Description	Example
=	Simple Assignment	Assignment operator evaluates the expression on its right hand side and stores the obtained value in the variable on its left hand side	$\Rightarrow a = 10$ $\Rightarrow a$ 10
+=	Add & assign	+= operator is compound assignment operator that adds the value of its second operand to the current value of the first operand and then assigns the result in the variable at its left hand side.	$\Rightarrow a = 5$ $\Rightarrow a += 10$  $\Rightarrow a = 15$ 20

→ Assignment operators ($=, +=, -=, *=, /=, \%, \&, \&\&, \&\&\&$)

⇒ Identity operators (is) : Id identity operator returns true if both of its operand refer to same object, otherwise it returns false.

Example!

```
>>> a
```

```
>>> b = a
```

```
>>> a is b
```

True

```
>>> c = 10
```

```
>>> a is c
```

false

→ Identity operator (is)

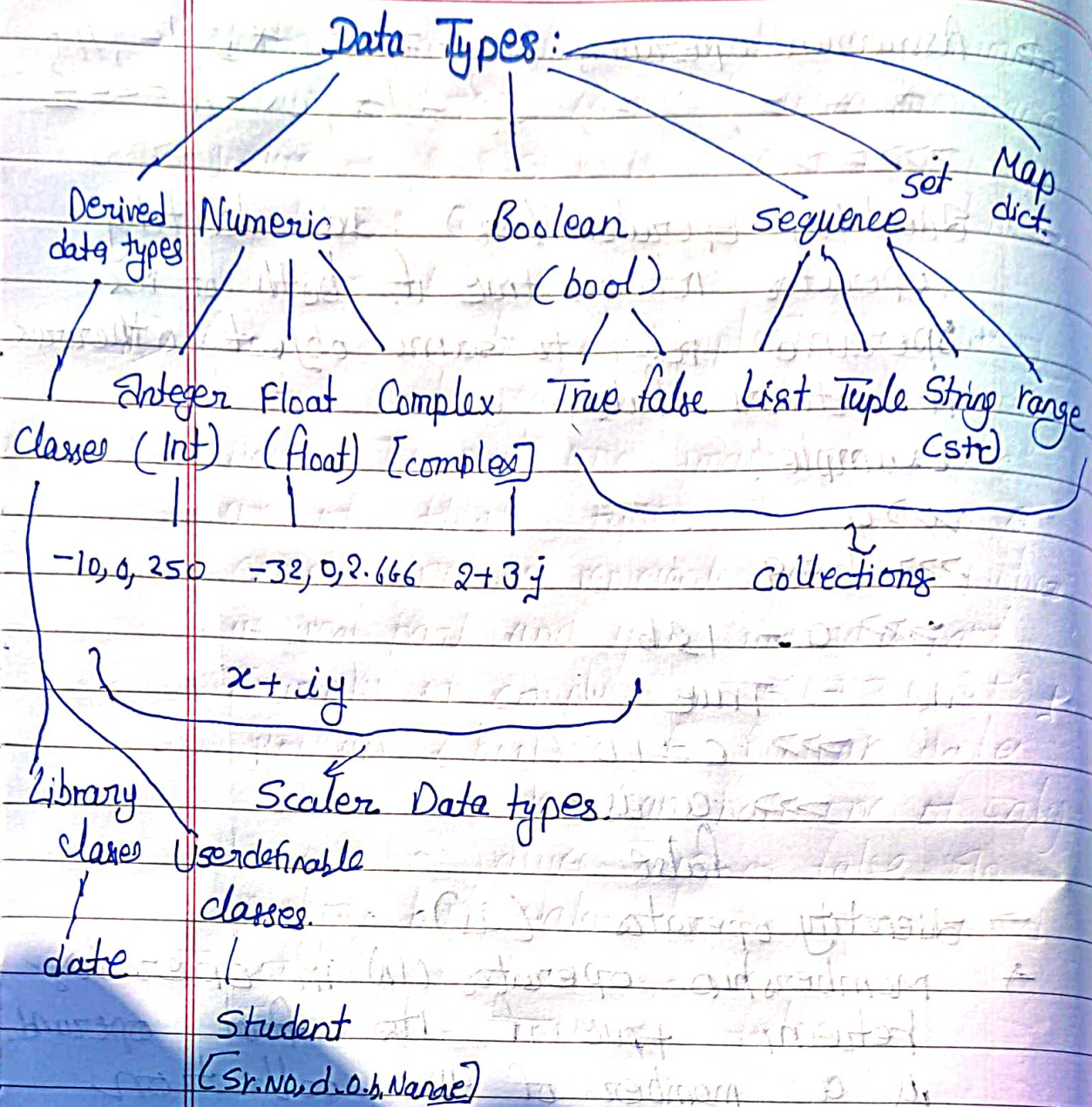
→ membership operators (in) in operator returns true if its left operand is a member of the collection at its right hand side

```
>>> 5 in [1, 2, 3, 4, 5]
```

True

```
>>> 10 in [1, 2, 3, 4, 5]
```

false



Data Types

1. Numeric [int, float, complex]
2. Boolean [bool]
3. Sequence
 - (i) List [List]
 - (ii) Tuple [tuple]
 - (iii) String [str]
 - (iv) Range [range]

~~A. Set: A set is an unordered collection of data items [elements]. The~~

Data type:- In python, every value that we use in a program has a data type. A data type determines the types of values that can be stored in a variable, the range of values and the operations that can be performed on these data items. We need not declare a variable before using it in a python program. Python automatically determines the data type of a variable depending upon the value assigned to it. Various data types in python are:

① Numeric Data type: Python supports three numeric data types, - int, float & complex.

i) Integer (int):- An integer is a ^{no.} numeric with out any fractional part. Integers can be +ve, zero or -ve. The int. data type in python is associated with integer values. There is no limit on the minimum & maximum value of int. data type in python. Examples of integers are:- -5167, 0, +7210

ii) Floating point [Float]:- These no. is also called real no. A floating point no. has a whole part as well as a fractional part. A float can either be +ve or -ve or zero. The float data type in python is associated with floating point numeric values.

Floating point numbers can have a maximum value of 1.797×10^{308} and a minimum value of 8.2250×10^{-308} and a precision of 15 digits. Examples of floating point no. are 10, 725, 0.0, 0.00009.

ii) Complex numbers [complex]: It consists of two parts real part and imaginary part. The real and imaginary parts are of float types. In python a complex no. is represented as $x + jy$ where x is real part and y is imaginary part. Examples of complex no: $5.0 + 3.0j$; $0 + 2j$

2) Boolean (bool): A Boolean data type can have one of only two possible values - True or False. In python bool data type is associated with Boolean values. Examples: True or False

⇒ sequence: A sequence is an ordered collection of elements. A sequence may also contain duplicate elements. The elements of a sequence can be accessed using their position within the sequence. In python, there are following built-in sequences:

Data type:

- Numeric (int, float, complex)
- Boolean

- Sequence ① List ② Tuple ③ Range.

String (str) A sequence is an ordered collection of elements. A sequence may also contain duplicate elements. The elements of a sequence can be accessed using their position within the sequence. In python, there are following built-in sequences:

① List: A list is an ordered linear collection of elements. A list may contain duplicate elements. The elements of a list are generally homogeneous. The elements of a list can be accessed using their positions [index] within the list. The index of first element of a list is 0. A list is a mutable sequence i.e. we can change the content of a list after it is created.

Example:- A list is created using comma separated elements enclosed inside square brackets.

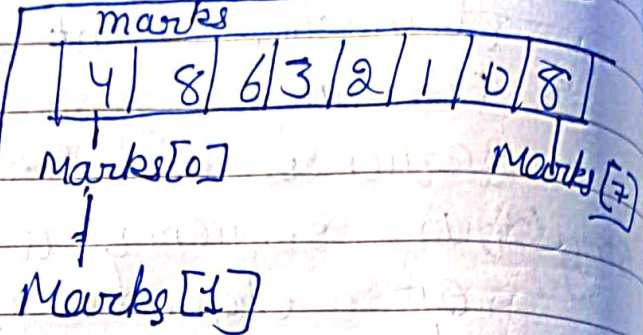
For example: ~~no~~

```
>>> marks = [4, 8, 6, 3, 2, 1, 0, 8]
```

```
>>> [4] = 10
```

```
>>> marks
```

```
[4, 8, 6, 3, 10, 10, 8]
```



2. Tuple: A tuple is an ordered linear collection of element. The elements of a tuple may be duplicate. The elements of a tuple can be accessed using their position within the tuple. A tuple is an immutable sequence i.e. we cannot make changes to the content of a tuple after it is created.

A tuple is created in Python by enclosing comma separated elements in round brackets.

Example :-

```
>>> marks = (5, 3, 1, 3, 7, 8, 0)
```

```
>>> marks [0]
```

```
>>> marks [2] = 15
```


3. **Range:** A Range is an ordered collection of ~~etc~~ integers. The elements of a range are specified using a starting value and end value and a step value. We can access the elements of a range using the position (index) of the element within the range.

A range is an ~~an~~ immutable sequence it cannot be changed after it is created. A range is created in Python using range keyword.

Example:

```
>>> odd = range(1, 100, 2)
```

```
>>> even = range(0, 100, 2)
```

```
# 0, 2, 4, ..., 98
```

A start value is part of the range whereas the end value is not included in the range.

```
>>> x = range(0, 100, -1)
```

4. **String (str):** A string is an ordered collection of characters. The elements ~~character~~ of a string may contain duplicate character. A character within

a string may be accessed using its position [index] within the string. A string is an immutable sequence i.e. we cannot make any changes to a string after it is created. A string is created in python using either double quotation or single quotations.

Example of strings are:

```
>>> name = "Mahima Thakur"
```

```
>>> name[5]
```

a

4. **Set:** A set is an unordered item collection of data items [elements]

The elements of a set are not indexed. A set cannot contain duplicate elements. A set is immutable data type i.e. the elements of a set cannot be changed after the set is created. A set can be created by enclosing the list of its elements in curly brackets.

e.g. >>> x = {2, 4, 3, 5, 0}

The duplicate elements are automatically

ignored by ^{while} creating a set.
 e.g. `>>> a = {1, 0, 4, 3, 1, 2, 0, 2}`
`>>> a`
`{1, 0, 4, 3, 2}`

5. **Map [dict]**: A dictionary is an unordered collection of elements where each element can be accessed using its unique key. The dictionary elements consist of key : value pairs. A dictionary is a mutable data type i.e. the elements in a dictionary can be changed after the creation of dictionary. The keys of a dictionary must be unique.

A dictionary can be created by enclosing the list of key : value pairs of elements in the curly brackets.

Examples:

`>>> marks = {"Python": 30, "OS": 50, "MT": 40}`

^{key}
 value

The elements of a dictionary can be accessed by specifying their keys in the square bracket except set after the name of the dictionary.


```
>>> marks["os"]  
50
```

mutable

```
>>> marks["os"] = 40
```

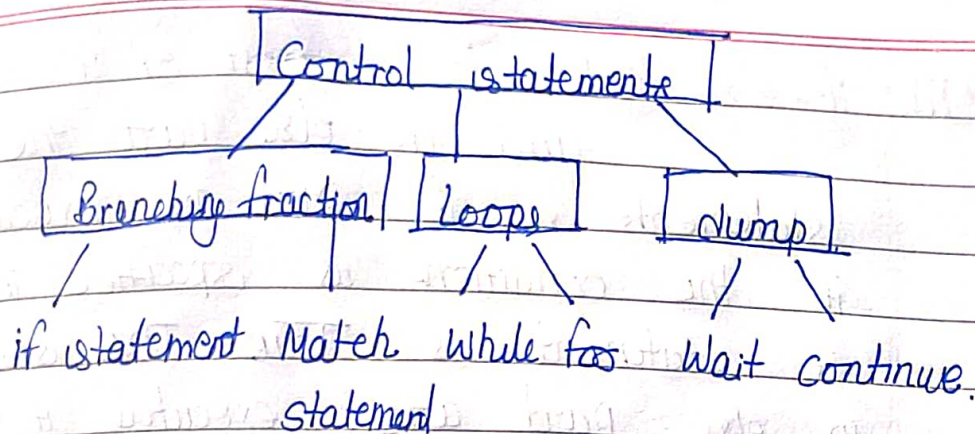
```
>>> marks["MM"] = 30
```

6. Derived Data types:

Python provides many other data types derived from built in data types. The standard python library contain many search derived data types. We can also create our own data type [user defined data types] as per our own requirements using python classes.

8 Nov. Python control statements

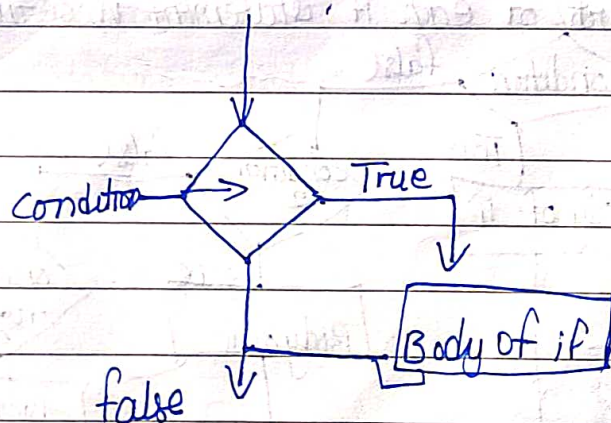
The statements of a program are executed sequentially i.e. in order in which they are written this is called a normal flow of execution. The control statements are used to change the normal flow of execution of a program. Various control statements in python are:



1. Selection / Conditional Control Statement:

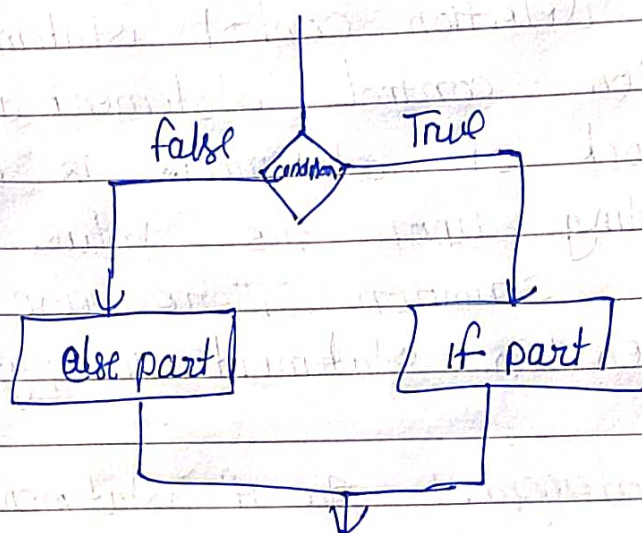
An selection control statement in selection control statement a statement a block of statements is executed depending upon the value of a given condition. There are two types of selection statements in python:

(1) if statement: An if statement the body of if is executed when the condition is specified in the if statement is True. otherwise the statements in the body of if are not executed.

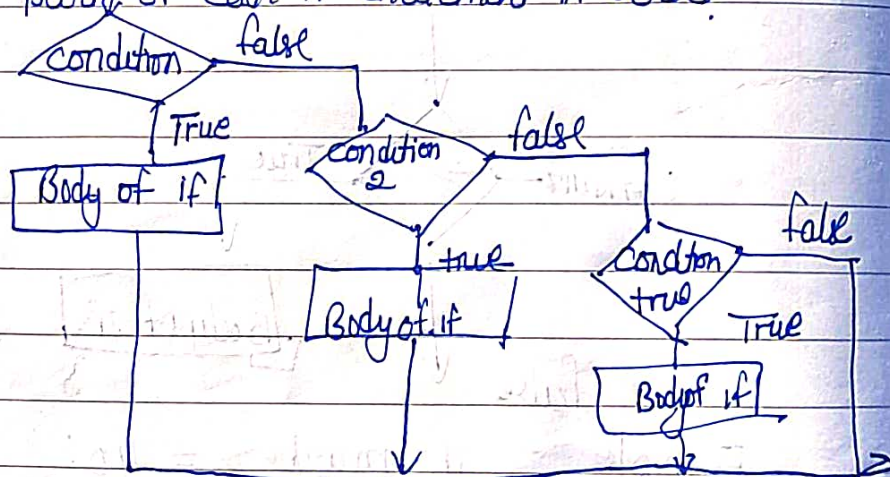


Example = if marks $\geq$ 40:
 $\rightarrow$ print("You have passed")

(ii) if-else statements - consist of if part and as else part the statements in if part are executed if the condition is specified in if statement is True; The statements in else part are executed if the condition specified in if statements is false.



(iii) if-elif ladder: An elif ladder is formed when another if statement is included in the else part of the if-else statement. We can check a no. of successive conditions in the else parts of each if statement in series.



Flow chart if-elif ladder

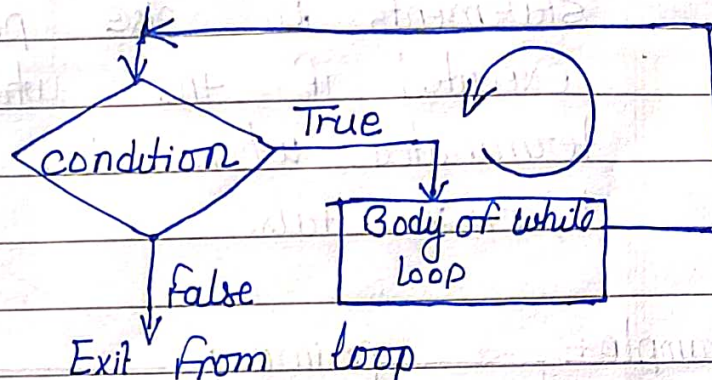
9 Nov 2021

Python Control statements.2. Loops / Iterations control statements:

Loops are used to execute a block of statements again & again until a given condition is satisfied.

In python there are two loop control statements :

- (i) while statement : while statement executes the statements in its body until the condition given in while statement becomes false.

Example:

```

mtable.py
"""
Multiplication Table
Mahima Thakur
"""
num = int(input("Enter Number:"))
i = 1
while i < 10 :
    print(num, "x", i, "=", num * i)
    i = i + 1
  
```


output

Enter Number : 6

$$6 \times 1 = 6$$

$$6 \times 2 = 12$$

$$6 \times 3 = 18$$

$$6 \times 4 = 24$$

$$6 \times 5 = 30$$

=

$$6 \times 10 = 60$$

Python also supports else part of while statement. The else part statements in else part are executed if the while loop is terminated when the loop condition becomes false.

Example ↴

prime.py

» » »

Check for a prime number

» » »

num = int(input("Enter Number:"))

i = 2

while i < num:

if num % i == 0:

print("Not a prime number")

break

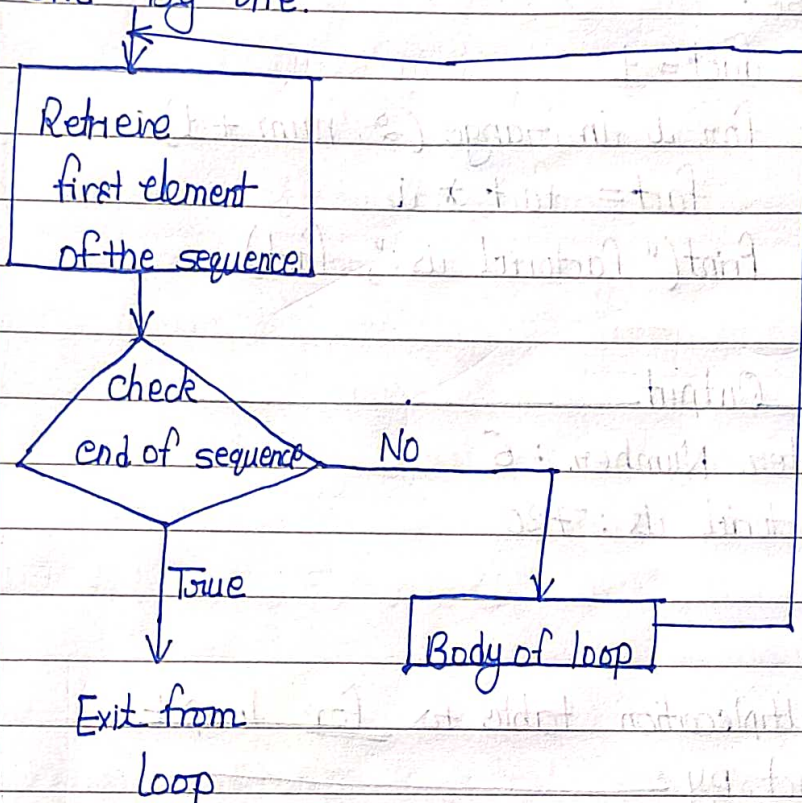
else:

print("Number is prime")

output	output
Enter Number : 49	Enter number : 5
Not a prime number	Number is prime.
	" " "

10 Nov.

(ii) For loop : for is a looping control statement in Python used to process elements of a sequence/collection one by one.



flowchart of for statement

The loop retrieves the next elements of a sequence and execute the body of the loop until the end of sequence is reached.

Example:-

fact.py

"""

To compute factorial of a number

"""

num = int(input("Enter number:"))

if num < 0:

print("Negative number not accepted")

else:

fact = 1

for i in range(2, num + 1):

fact = fact * i

Print("factorial is:", fact)

Output

Enter Number : 6

factorial is : 720

Multiplication table for for loop:-

fact.py

"""

To compute factorial of a no.

"""

num = int(input("Enter number:"))

for i in range(1, 11):

print(num, "x", i, "=", num * i)

fact.py

"""

To compute factorial of a number.

"""

num = int(input("Enter number:"))

lterm = 1

lterm = 1

for i in range(1, num+1):

if i <= 2:

print(1)

continue

term = lterm + lterm

print(term)

lterm = lterm

lterm = term

output

num=8

1, 1, 2, 3, 5, 8, 13, 21

15 Nov,

3. Jump Statement Control statements :

① **break statement** : break statement can be used inside loops only. When a break statement is executed, the loop is terminated immediately without checking the actual loop terminating condition.

Example :-

prime.py

» » »

To check whether a given no. is a prime no.

» » »

```
n = int(input("Enter a number:"))
```

```
for i in range(2, n):
```

```
    if n % i == 0:
```

```
        break
```

```
    print("Number is not prime")
```

```
    break
```

```
else
```

```
    print("Number is prime")
```

Output

Enter a number: 43

Number is prime.

Enter a number: 2

Number is prime.

- ② Continue statement : continue statement is can be used inside loops only. It terminates the current iteration of the loop and executes the next iteration (if any).

Example + divide.py

```
" " " print the numbers b/w 1&100 which  
are not divisible by 5 " " "  
for i in range(1, 100):  
    if i % 5 == 0:  
        continue  
    print(i)
```

Output

1
2
3
4
6
7
8
9
=
99

③ **return statement**:- It is used inside functions. The return statement return a value and the control back to the statement from which the function is called.

Example :-
def sum(a, b):
 return a + b
Sum = print(sum(10, 20))

return . py

```
def greatest (a, b):
```

```
    if a > b:
```

```
        return a
```

```
    else
```

```
        return b
```

```
greatest (10, 5)
```

```
    a = 10
```

```
greatest (10, 20)
```

```
    b = 20
```

④ pass statement : pass statement does not perform any action. pass statement is used when no action is required but a statement has to be written.

example

```
def x():
```

```
    pass
```

16 Nov

Variable

A variable is a named memory location that contains a value of a specific type. The memory location is accessed by the variable name.

in the program. The value of a variable can be changed during program execution. The name of a variable in Python must follow the rules of identifiers:

1. A variable name must begin with either an underscore (`_`) or a alphabetical letter (`a-z`), (`A-Z`)
2. Variable names are case-sensitive (lower case and upper case letters are considered to be different)
3. The remaining characters of a variable name may consist of underscore (`_`) letters (`A-Z`, `a-z`) and digits (`0-9`)
4. Python keywords are not allowed as variable names.
5. A variable is created whenever we assign some value to it.

A variable defined outside a function is called a global variable. And a variable defined inside a function is called a local variable. A global variable can be accessed from anywhere within the program. A local variable can be accessed only within the function in which it is defined. To change the value of a global variable within a function, we must

declare the glo: variable as global inside the function.

Examples:

```
a = 10
```

```
a = 5.4
```

```
a = Hello
```

```
print(a)
```

Output

Hello

```
a = 10 # global variable
```

```
def func(): # function definition
```

```
print(a) # 10
```

```
a = 5
```

```
func() # function call
```

```
print(a) # 10
```

```
a = 10
```

```
def fun():
```

```
    global a
```

```
    print(a)
```

```
    a = 5
```

```
fun() # 10
```

```
print(a) # 5
```


Strings

A string is a sequence of characters denoting some text data. In python, str data type is associated with strings. Strings are immutable in python i.e. the strings once created cannot be ~~m~~ changed. A string literal can be enclosed inside single quotations or double quotations in python. A multiline literal is enclosed inside triple quotations.

Example :-

```
name = "Mahima"
```

```
branch = 'Information Technology'
```

```
sem = '3rd'
```

```
Address = """ Vill. Tikkar P.O. Aalu Teh. Aut  
Distt - Mandi H.P. 175121
```

```
"""
```

17 Nov.

Python String Methods :-

Python provides many methods to manipulate strings. Some important string methods in python are

string method	Description	Example
1. capitalize()	capitalize method returns	>>> "hello"

is string after changing the first letter of its object string to uppercase.

world", capitalize()
"Hello world"
>>> "123".capitalize()
"123"
>>> "HELLO".capitalize()
"HELLO"

2. upper()

upper() method returning string after changing converting all the letters of its object string to uppercase (capital) letters.

>>> "Hello".upper()
"HELLO"

3. lower()

lower() method returning a string after converting all the letters of its object string to lowercase (small) letters.

>>> "Hello".lower()
"hello"

4. title()

title() method returns a string after converting the first letter of each word of the object string to capital letters and other letters to small letters.

>>> "hello world".title()
"Hello World"

5. Count()

Count() method returning the no. of times a given substring occurs in the string.

>>> name = "Machi Thakur"
>>> name.count("a")
>>> name.count("a")

22 Nov

classmate

Date _____

Page _____

			3
			>>> name.count("Sharma")
			0
6.	Find()	It returns the position of a given substring in a string. If the substring does not exist in the string then find returns -1.	>>> name.find("Kumar") ("Kumar") 8 >>> name.find("Sharma") -1
7.	Replace()	Replace method finds a substring in a given string and replaces it with another substring.	>>> name.replace("Kumar", "Sharma") 'Krishan Sharma' >>> name.replace("Sharma", "Kumar") Krishan Kumar
1	Format()	Format method substitutes the formatted values in place of placeholders given in the string. The substituted values are formatted as per the given format specifiers. The place holders are written in curly brackets within the string. The values to be substituted	>>> marks = 10 >>> "You scored {}" format(marks) You scored 10 >>> name = "Mahi" >>> marks = 0.45 >>> print("{} score {} marks") format(name, marks)

are passed in the arguments of format method. format

of a place holder is:

{field: format specifier}

print("{} scored {:.%}.marks,

format(name, marks))

Maahi scored 45.000%marks

The field can be left empty,

can be position of the

parameter in the format()

method or can be the name

of parameter of format()

method.

The format specifier contains

[fill][align][sign][width]

[.precision][format character]

where fill is the character

to be filled in the value

to make its width equal to

the specified width. [align]

can be <: left, >: right, ^: centre

or =; placement of sign can

be either + or -. width is

minimum width of the

substituted value. [precision]

is the maximum no. of

digits after decimal points

in floating point numbers

format character is the type

Maahi scored
0.45 marks

>>> name="Mahima
Thakur"

>>> mark=25

>>> print("{}: *25%

scored {1:5d}

marks." format

(name, marks))

***Mahima

Thakur secured

25 marks

of conversion to be applied,
it can be: d: decimal
x: hexadecimal, o: octal,
b: binary, s: string, f: floating
point, g: floating point
e: floating point, c: character,
%: percentage.

```
for i in range
    print ("{} scored {} marks
    ({} : 5.2 %)".format(name,
    marks, marks/50))
Aay Kumar scored 25 marks
(50.00%)
```

len()
function

len() function returns the
len of a string in characters

len("Aman
Kumar")
10

23 Nov + Operations on sequences

Various sequence data types in python are:-

→ List → Tuple → string → Range.

Different operations that can be performed
on sequence are:

① Indexing - The element of a sequence
can be accessed using their
positions within the sequence. By

using index of an element. The position of an element within a sequence is called its index. The index of first element is 0. This process of accessing the elements of a sequence using their index is called indexing. Python also supports negative indexes if the index is -ve, the elements are accessed relative to the last element of the sequence.

Example ÷

$x[0] \ x[1] \ x[2] \ x[3] \ x[4] \ \dots \ x[7] \rightarrow$ +ve index

x	5	4	3	1	6	7	2	0
---	---	---	---	---	---	---	---	---

 $x[-8] \ \dots \ x[-2] \ x[-1] \rightarrow$ negative index

Examples:

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> a[3]
```

4

```
>>> b = "Hello World!"
```

```
>>> b[-5]
```

o

```
>>> c = range(10, 50, 4)
```

```
>>> c[6]
```

34

(ii) Slicing $\leftarrow$ It is used to extract a portion of a sequence. The value

returned by slicing is of same type as that of the sequence on which it is applied. The syntax of slicing a sequence is: `[start: end: step]`.

sequence name `[start: end: step]`.

The elements contain in the resultant sequence are at position start, start + step, start + 2 step ---

The element at position end is not included in the resultant sequence.

Example $\rightarrow$

```
>>> a = [1, 3, 4, 2, 6, 7, 3, 1]
```

```
>>> a[2:5]
```

```
4, 2, 6
```

```
>>> a[-2:-5]
```

```
[3, 7, 6]
```

```
>>> b = "Manan Kumar"
```

```
>>> b[::-1]
```

```
ramuk namA
```

```
>>> b[2:2]
```

```
"Aa aa"
```

(iii) Concatenation $\div$ It is used to join two sequences together to perform a new sequence. The `+` operator is used to perform concatenation.

Ranges do not support concatenation.

Example ↴

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

```
>>> first name = "Ramesh"
```

```
>>> last name = "Kumar"
```

```
>>> Full name = [first name + " " + last name]
Ramesh Kumar,
```

(iv) Repetition — This operation is used to repeat a sequence multiple times to form a new sequence. Ranges do not support repetition.

Example * operator is used for sequence repetition.

eg :-

```
>>> [1, 2] * 4
```

```
[1, 2, 1, 2, 1, 2, 1, 2]
```

```
>>> "-" * 10
```

```
-----
```

1. Dec (v) Membership Testing (in): Membership testing operator is represented by 'in'. The 'in' operator returns true if a given numt item exist in the sequence, otherwise it returns False.

eg:-

`a = [4, 7, 6, 10, 4]``5 in a # Returns False``4 in a``# returns True``b = "Hello World!"``"xyz" in b # returns false``"LL" in b # true``c = range(15, 50, 4)``21 in c # False``23 in c # true`

Operations on Lists

Various operations that can be carried out on list are :

- (i) Append () : Append () method is used to add a new element at the end of a list

example :-

`a = [5, 7, 4, 3]``a.append(10) # Now a will contain
[5, 7, 4, 3, 10]`

- (ii) insert () : insert method is used to add a new element at a given index position in a list. The

~~elements~~ existing elements after the insert location are shifted to right by one position.

For example ÷

$a = [5, 7, 4, 3]$

$a.insert(2, 5)$ # Now a will contain $[5, 7, 5, 4, 3]$

(iii) $remove()$: $remove()$ method deletes the first element of the list having a given value.

For example ÷

$a = [2, 3, 4, 2, 5, 6]$

$a.remove(4)$ # Now a will contain $[2, 3, 2, 5, 6]$

(iv) $pop()$: pop method deletes an element of a list at a given index and returns the deleted element. If index is not specified then the last list element is deleted.

For example ÷

$a = [5, 7, 3, 4, 6]$

$a.pop()$ # Now a will contain $[5, 7, 3, 4]$

$a.pop(2)$ # Now a will contain $[5, 7, 4, 6]$

(v) $Clear()$: $Clear()$ method removes or deletes all the elements of a list.

eg:- a [1, 2, 3, 4]

a.clear() # Now a will contain []

(vi) reverse(): reverse() method reverses the positions of all the elements of a list.

Example :-

a = [5, 3, 4, 3, 2]

a.reverse() # Now a will contain [2, 3, 4, 3, 5]

3 Dec.

(vii) Sort(): sort() method is used to arrange the elements of a list in ascending or descending order.

The syntax of sort() method is:

list.sort(key, reverse)

where key is the name of the function which is applied to each item

to find the order key. reverse can

either be True or False. If reverse

is true the list is sorted in

descending order. Otherwise, it is

sorted in ascending order.

Example :-

a = [1, 6, 2, 4, 3, 0, -1]

a.sort() # a will contain [-1, 0, 1, 2, 3, 4, 6]

a.sort(reverse = True)

[6, 4, 3, 2, 1, 0, -1]

(viii) `Count()` ÷ `Count` method returns the no. of occurrences of a given element in the list:

For example ÷

`a = [1, 2, 1, 0, 7, 6, 1]`

`a.Count(1)` # it will return 3

`a.Count(10)` # 0

(ix) `index()` ÷ `index()` method return the index of the first occurrence of a given element in the list. We can specify start and end index of the list in which the element is to be searched.

For example ÷

`a = [5, 3, 1, 6, 0, 4]`

`a.index(0)` # it will return 4

if element is not found then an error is displayed.

CT-II

Q1: Fill in the blanks:

- (i) A string s can be reversed in python with `s[::-1]`
- (ii) The `title()` string method returns a string after capitalizing the first letter of each word in the object string.
- (iii) A multi-line string literal in python is enclosed inside Triple quotes (`""" / """`)
- (iv) The `pass` statement is null statement in python and does not perform any operation.
- (v) The `input()` python function is used to read data from the standard input device.

Que2: Write a short note on slicing operation on python sequences.

Ans- It is used to extract parts of a sequence in python.

The syntax of a slicing operation is:
sequence [start : end : step]

In this operation, elements of a sequence at index start, start + step, start + 2 step --- upto end are accessed.

The element at index end is not included in the slicing.

Example!

`a = [2, 3, 5, 6, 8, 10]`

`a[2:5:2]` will return `[5, 8]`

The start and end step values are optional. If start is ignored it is treated as zero (0). If end is ignored it is treated as length of a sequence. If step is ignored, it is treated as 1. We can also use -ve values for start, stop and step.

Que 3: What do you understand by a sequence? Explain various types of built-in sequences in python.

Answer: A sequence is an ordered collection of similar items. In python, there are four types of inbuilt sequences:

1. List.

2. Tuple

3. Range

4. String

A sequence may contain duplicate elements.

1. List: A list is an ordered collection of elements. A list is a mutable sequence (the elements in a list can be changed). The element of the list can be accessed by specifying its index within square brackets after the list name. The index of

first element of a list is 0. A list can be created by listing (enclosing) the list of items in square brackets. Example of a list: `marks = [10, 6, 3, 4, 2, 10]`

2. String (str): A string is an ordered collection of characters used to represent some text data. A string is an immutable. The elements in a string cannot be changed. The index of a string is within square brackets after the string name. The index of first element of a list is 0. A string can be created by enclosing the sequence of characters inside quotation (single / double).

```
>>> name = "Aman Kumar"
```

```
>>> name[5]
```

3. Tuple: A tuple is an ordered collection of elements. A tuple is an immutable sequence. The elements of a tuple can not be changed. Each element of a tuple can be accessed by specifying its index within square brackets after the tuple name. The index of a first element of a tuple is 0. A tuple can be created by listing

by enclosing the list of its elements in round brackets.

Examples:

[0, 1, 2, 3, 4] $a = (1, 6, 3, 2, 0)$

4. Range: A range is an ordered collection of integers forming an arithmetic progression (AP) series. A range is an immutable series sequence and the elements of a range can be accessed by specifying the index of its turn within the square brackets after the range name. A range can be created using `range()` class as follows:

`range name = range(start, stop, step)`

It will create a range `start, start + step, start + 2 x step, ...` upto stop.

The value of stop is not included in the range. The start and step

are optional arguments in the `range()`.

If start is ignored it is treated

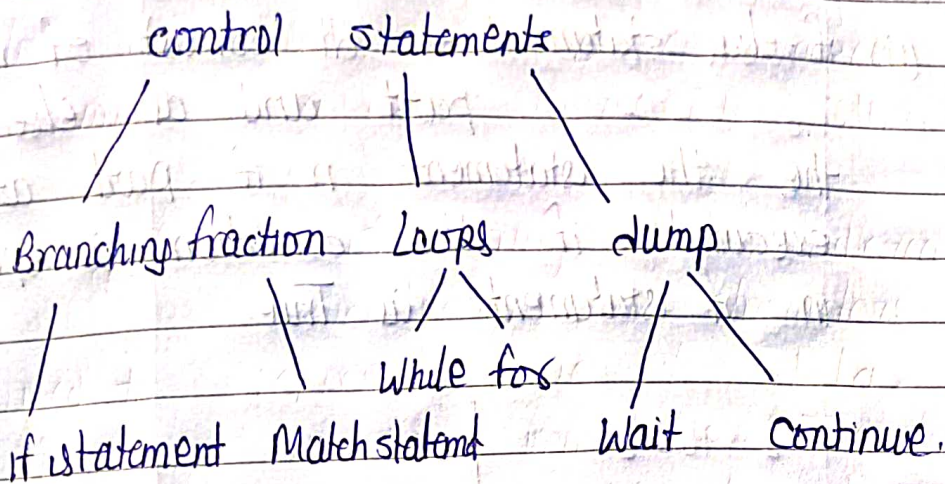
as 0 and if step is ignored

it is treated as 1.

Question 3: Explain various control structure in python.

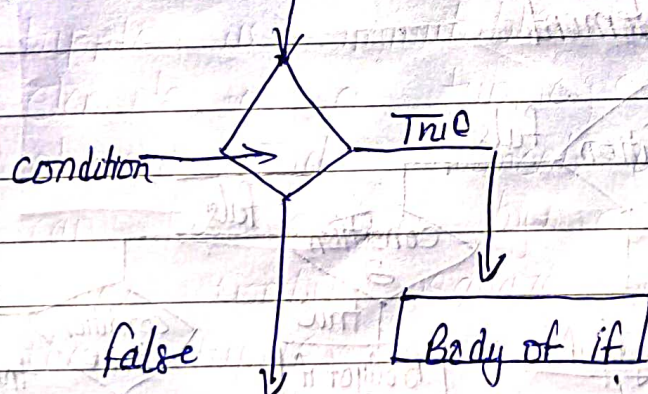
Answer: The statements of a program are executed sequentially i.e. in order in which they are written is

called a normal flow of execution.



1. selection / conditional control statement :-
 An selection control statement in selection control statement a block of statements is executed depending upon the value of a given condition.

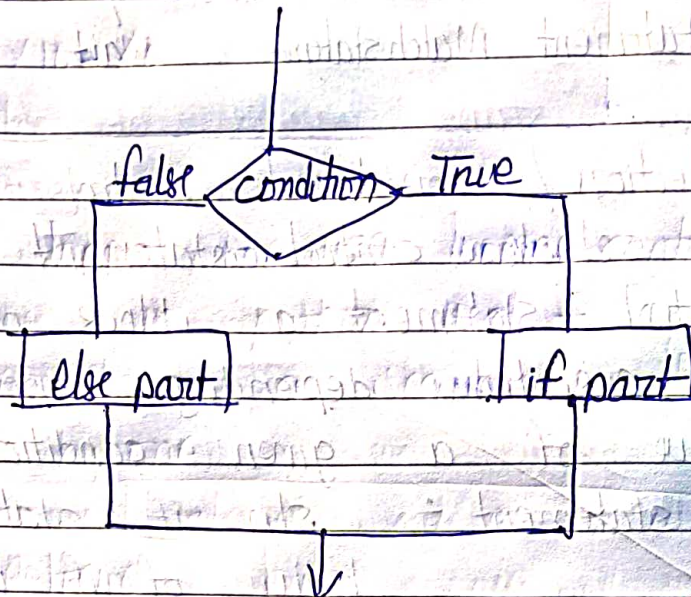
(i) if statement :- An if statement the body of if is executed when the condition is specified in the if statement is True. otherwise the statements in the body of if are not executed.



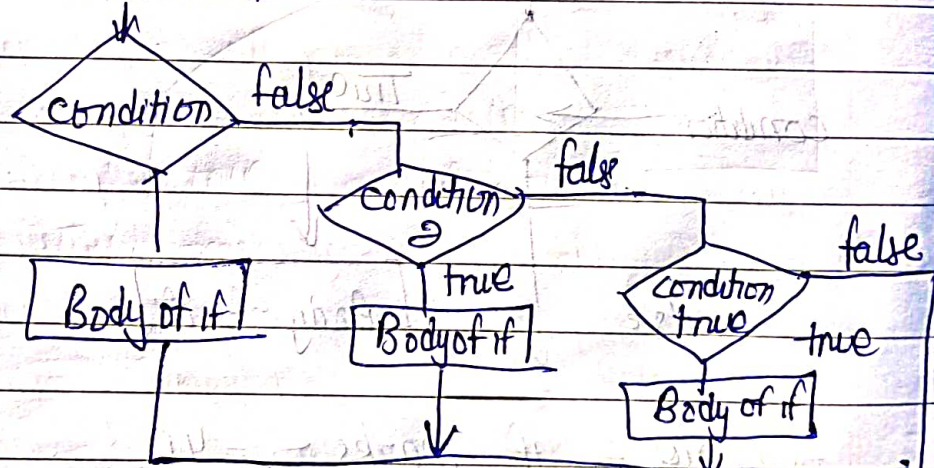
Example = if marks >= 40

>> print ("You have passed")

(ii) if-else statement - It consist of if part and an else part the the statement in if part are executed if the condition is specified in if statement is True.



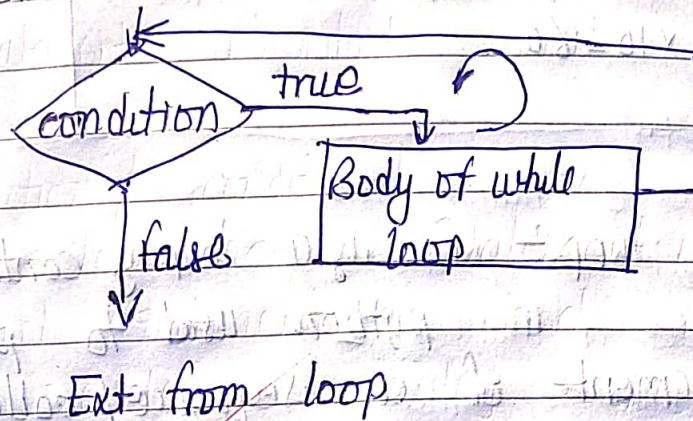
(iii) if-elif ladder - an elif ladder is formed when another if statement is included in the else part of the if else statement



flow chart of elif ladder

2 loops / Iterations control statements;
 loops are used to execute a block of statements again & again until a given condition is satisfied.

① While statement - It executes the statements in its body until the condition given in while statement becomes false.



Mtable.py

```
""" """
```

Multiplication Table

Mahima Thakur

```
""" """ """
```

```
num = int(input("Enter Number:"))
```

```
i = 1
```

```
while i <= 10:
```

```
    print(num, "x", i, "=", num * i)
```

```
    i = i + 1
```


output

Enter Number: 6

$$6 \times 1 = 6$$

$$6 \times 2 = 12$$

$$6 \times 3 = 18$$

$$6 \times 4 = 24$$

$$6 \times 5 = 30$$

$$6 \times 6 = 36$$

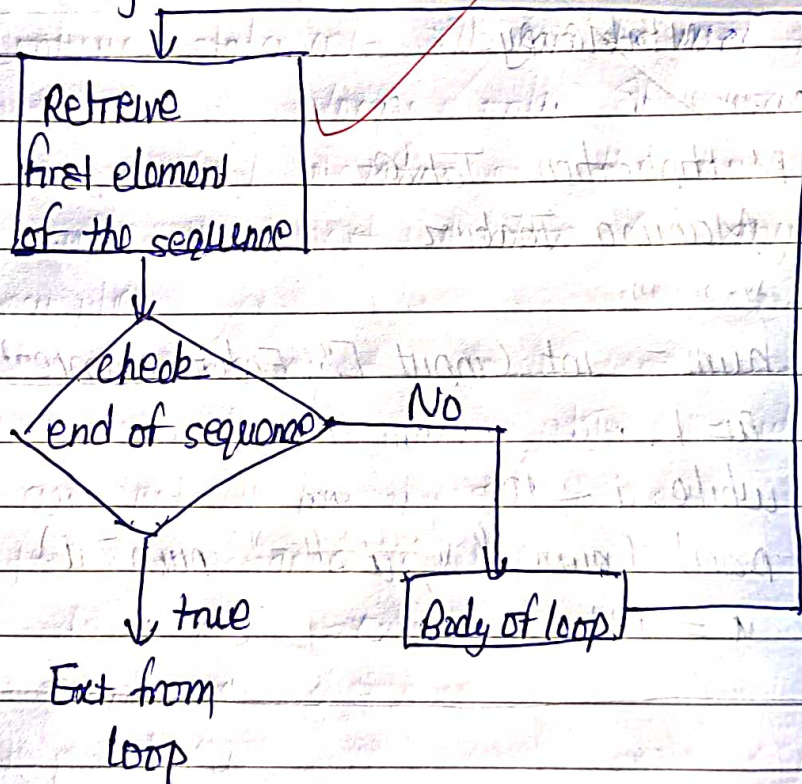
$$6 \times 7 = 42$$

$$6 \times 8 = 48$$

$$6 \times 9 = 54$$

$$6 \times 10 = 60$$

- (ii) For loop :- For is a looping control statement in python used to process elements of a sequence/ collection one by one



Example: fact.py

" " "

To compute factorial of a number

" " "

```
num = int(input("Enter number:"))
```

```
if num < 0:
```

```
    print("Negative number not accepted")
```

```
else:
```

```
    fact = 1
```

```
    for i in range(2, num + 1):
```

```
        fact = fact * i
```

```
    print("factorial is:", fact)
```

Output

Enter number : 6

factorial is : 720

3. Jump Control statements:

- ① break statement :- It can be used inside loops only. When a break statement is executed, the loop is terminated immediately without checking the actual loop-terminating condition.

Example:-

prime.py

" " "

To check whether a given no. is a prime no.


```

n = int(input("Enter a number:"))
for i in range(2, n):
    if n % i == 0:
        print("Number is not prime")
        break
    else:
        print("Number is prime")

```

Output

Enter a number: 43

Number is prime

Enter a number: 2

Number is prime

(2)

Continue statement: It can be used inside loops only. It terminates the current iteration of the loop and executes the next iteration (if any).

Example:

```

# divisible by 5
print the numbers b/w 1 to 100 which
are not divisible by 5
for i in range(1, 100):
    if i % 5 == 0:
        continue
    print(i)

```


Output

1

2

3

4

5

6

7

8

9

=

99

③ Return statement + It is used inside functions. The return statement returns a value and the control back to the statement from which the function is called.

```

return py
def greater(a, b):
    if a > b:
        return a
    else:
        return b
greater(10, 5)
a = 10
greater(10, 20)
b = 20
  
```


- ④ Pass statement: It does not perform any action. Pass statement is used when no action is required but a statement has to be written.

example

```
def x():
```

```
    pass
```

~~Does~~
04/12

6 Dec LIST COMPREHENSION

List comprehension is an easy way to create lists by applying some operation to the elements of a sequence. List comprehension can also be used to create a new list by applying a filter condition to the elements of a sequence.

Syntax to create a list using list comprehension is:

[x for x in some sequence]
mapping [fun(x) for x in sequence]

filter [x for x in sequence if condition]

example: " " "

create a list of squares of first 10 natural numbers.

" " "

$[x**2 \text{ for } x \text{ in range}(1,11)]$

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

" " "

create a list by selecting positive numbers from another list

" " "

$l = [1, 6, 0, -2, 6, 2, -1]$

$[x \text{ for } x \text{ in } l \text{ if } x \geq 0]$

7 Dec

Python Set

[] brackets

{ } braces

() parenthesis

A set is an unordered collection of elements. A set may not contain duplicate elements. A set is a mutable data type in python.

A set can be created in python using curly brackets or by using set() function.

Example of sets:

$s = \{1, 3, 4, 5, 0, 1\}$

print(s) # output shall be {0, 1, 3, 4, 5}

Various operations that can be performed on sets are: (1) add (2) remove (3) discard (4) ~~pop~~ ^{clear()}

(1) Add(): add() method is used to

add a new element in the set

Example: $s = \{5, 3, 4\}$

$s.add(6)$ # $\{5, 3, 4, 6\}$

- (2) Remove () ÷ Remove method deletes the given element from the set. If the element does not exist an error is raised.

Example: $s.remove(6)$ # $\{5, 3, 4\}$

- (3) Discard () ÷ discard method deletes a given element from the set if it exists.

Example:

$s.discard(6)$ # $\{5, 3, 4\}$

- (4) Clear () ÷ Clear method deletes all the elements of a set.

Example:

- (5) Membership (in) ÷ in operator is used to check whether a given element is a member of the set. It returns True if the element exists in the set otherwise it returns False.

Eg- $s = \{4, 3, 3, 2, 6, 1\}$ $10 \text{ in } s$ # returns False

13 Dec.

Set Operations

1. Union $\rightarrow$ Union operation can be applied on the two sets union returns a set that contain an elements of both the sets.

Example:

$$a = \{1, 2, 3, 4\}$$

$$b = \{1, 2, 3, 4, 5\}$$

$$a \cup b$$

$$a \mid b$$



union operation

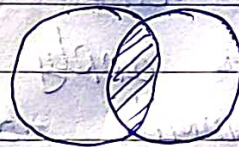
Union operators either can be applied using ~~two~~ union method or by | this symbol.

2. Intersection $()$: Intersection operation requires two sets intersection returns the element which are common in both sets.

Intersection operation can be applied using intersection $()$ method or $(\cap)$ operators.

$$a \cap b$$

$$a \& b$$



can be applied using this

3. difference: difference $()$ the difference operators of two sets is a

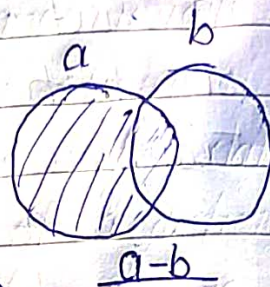
set that contains all the elements of the first set which are not in the second set.

Difference operation can be applied using either difference methods

Example:

$$a = \{1, 2, 3, 4\}$$

$$b = \{3, 4, 5, 6\}$$



method 1 $\rightarrow$ difference (b)

method 2 $\rightarrow$ $a - b$

4. Copy () : This method creates a copy of the set.

$$a = \{1, 2, 3, 4\}$$

a . copy ()

$$b = \text{add}(5)$$

print (a) " {1, 2, 3, 4}

print (b) " {1, 2, 3, 4, 5}

14/12

Set Comprehension

Set comprehension is an easy method to create a set from the elements of some other sequence / set. We can apply some operation on the elements of iterable to construct (iterable) a set using comprehension.

The syntax to create a set using comprehension is:

{ < expression > for < var > in < iterable >
if < condition > }

for example:

{ x * x for x in range(1, 11) }

x * x 2

for example:

{ x for x in range(2, 101) (for y in
(2, x) if
x % y != 0) }

DICTIONARY :-

Collection of items

⇒ key : value pairs

⇒ keys must be unique (not duplicate)

⇒ Mutable

⇒ { }

⇒ { key : value, key : value, --- }

⇒ m ⇒ { 0 : "zero", 1 : "one", 2 : "two" }

⇒ m[0] will be "zero"

m = dict(1, one) ⇒ m[3] will raise an error

(2, "Two") ⇒ m[3] = "Three" # { 0 : "zero",

1 : "one", 2 : "Two", 3 : "Three" }

DICTIONARY

A dictionary is a collection of items.

The items in a dictionary are stored using key : value pairs. The keys of dictionary are used to access the values of the item. The keys of a dictionary are unique (duplicate keys are not allowed). We can use any immutable data type value as key of a dictionary. A dictionary is also known as associative array in other programming languages.

A dictionary can be created by listing all key : value pairs inside curly brackets or using dict() class.

Example:

```
num = {1: "one", 2: "Two", 3: "Three", 4: "four"}
```

```
nums = dict(("one", 1), ("Two", 2), ("Three", 3),  
            ("four", 4))
```

```
print (nums)
```

```
{'one': 1, 'Two': 2, 'Three': 3, 'four': 4}
```

Elements of a dictionary can be accessed

by specifying the key within square brackets after dictionary name.

nums [3] will return 3

nums ["THREE"] will raise an error
if we assign some value to a
dictionary item and the key does
not exist in the dictionary then
a new item is added to the
dictionary

nums ["five"] = 5

Operations on dictionary:

1. **get () method**: get () method returns
the value of the item
with given key.

E.g:-

nums.get ("one") will return 1

2. **keys () Method**: keys () method returns
the list of keys of
the dictionary.

E.g:-

print (list (Num.keys ()))

["one", "Two", "Three", "four"]

3. **values () method**: values () method
returns the list
of values in the dictionary.

eg+

```
print (list (nums . value))
```

```
[1, 2, 3, 4]
```

4. **items()** method - items() method returns the list of (key, value) pairs in the dictionary.

eg+

```
print (list (nums . items))
```

```
[("one", 1), ("Two", 2), ("Three", 3),  
 ("four", 4)]
```

5. **Update()**

update() method is used to insert new items in the dictionary.

```
nums . update ({"five": 5, "six": 6})
```

```
print (nums)
```

```
{"one": 1, "Two": 2, "Three": 3,  
 "four": 4, "five": 5, "six": 6}
```

6. **Clear()**

Clear() method deletes all the items in the dictionary.

7. **Pop()**

pop() method deletes an item with specified key from the dictionary.


```
num1.pop("six")
```

8. **Membership (in):** in operator is used to check if an item with given key exists in the dictionary.

eg -

"One" in num1

True

"Ten" in num1

False

17 Dec

DICTIONARY COMPREHENSION

Dictionary comprehension is an easy way to create a dictionary from the values of other iterables. (sequence, set, generator etc.)

Syntax of dictionary comprehension is?

{ < expression > : < expression > for
< var > in < Iterable > if < condition > }

where < expression > for the keys of the dictionary and < expression > are the values.

Example -

{ x : x * x for x in range(5) }

will create

{ 0:0, 1:1, 2:4, 3:9, 4:16 }


```
* key = [1, 2, 3, 4, 5]
  value = ["one", "Two", "Three", "four",
           "five"]
```

```
{k:v for k, v in zip(keys, values)}
```

```
{1: "one", 2: "Two", 3: "Three",
```

17 Dec

Dictionary comprehension

Dictionary comprehension is an easy way to create a dictionary from the values of an iterable (sequence, set, generator).

Dictionary comprehension has following syntax:

```
{key: value for <var> in <iterable>
 if <condition>}
```

For example:

```
fig = [1, 2, 3, 4, 5]
```

```
words = ["One", "Two", "Three", "Four", "Five"]
```

Dictionary of squares of numbers in a range

```
{x: x**2 for x in range(5)}
```

will create a dictionary

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
{x: y for x, y in zip(fig, words)}
```

will create a dictionary

```
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
```