KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

DEPARTMENT OF CS, CA & IT

17CSU504B PROGRAMMING IN PYTHON 3H – 3C

Instruction Hours / week: L: 3 T: 0 P: 0 **Marks:** Int : **40** Ext : **60** Total: **100**

SCOPE

This programming language is versatile, robust and comprehensive programming language. It has true portability features and can be used across a multitude of platforms.

OBJECTIVES

- Master the principles of object-oriented programming and the interplay of algorithms and data structures in well-written modular code;
- Solve problems requiring the writing of well-documented programs in the Python language, including use of the logical constructs of that language;
- Demonstrate significant experience with the Python program development environment.

UNIT-I

Planning the Computer Program: Concept of problem solving-Problem definition- Program design-Debugging-Types of errors in programming-Documentation.

UNIT-II

Techniques of Problem Solving: Flowcharting-decision table-algorithms-Structured programming concepts-Programming methodologies: top-down and bottom-up Programming.

UNIT-III

Overview of Programming: Structure of a Python Program-Elements of Python.

UNIT-IV

Introduction to Python: Python Interpreter-Using Python as calculator-Python shell-Indentation. Atoms-Identifiers and keywords-Literals-Strings-Operators(Arithmetic operator, Relational operator, Logical or Boolean operator, Assignment, Operator, Ternary operator, Bit wise operator, Increment or Decrement operator).

UNIT-V

Creating Python Programs: Input and Output Statements-Control statements(Branching, Looping, Conditional Statement, Exit function, Difference between break, continue and pass.). Defining Functions-Default arguments.

Suggested Readings

1. Budd, T. (2011). Exploring Python(1st ed.). New Delhi: TMH.
2. Python Tutorial/Documentation www.python.org 2015.
3. Allen Downey., Jeffrey Elkner., & Chris Meyers. (2012). How to think like a computer scientist : learning with Python. Freely available online.
4. http://docs.python.org/3/tutorial/index.html.
5. http://interactivepython.org/courselib/static/pythonds.
6. http://www.ibiblio.org/g2swap/byteofpython/read/.

KARPAGAM ACADEMY OF HIGHER EDUCATION			
Department of CS,CA & IT			
III B.Sc CS - V Semester			
PROGRAMMING IN PYTHON - 17CSU504B			
UNIT I			
S.NO	**Lecture Duration (Hours)**	**Topics To Be Covered**	**Support Materials/ Pg.No**
		Planning the Computer Program	
1	1	Concept of problem solving	W1
2	1	Problem definition	
		Program design	
3	1	Debugging	T2 : 3,4
		Types of errors in programming	T1: 52-56
4	1	Documentation	W5
5	1	**Recapitulation and Discussuin of Important Questions**	
		Total No of Hours Planned for Unit I	**5**
		UNIT II	
S.NO	**Lecture Duration (Hours)**	**Topics To Be Covered**	**Support Materials/ Pg.No**
		Techniques of Problem Solving	
1	1	Flowcharting	W3
2	1	decision table	W4
3	1	algorithms	W5
4	1	Structured programming concepts	W5
		Programming methodologies	
5	1	top-down Programming	W5
		bottom-up Programming	
6	1	**Recapitulation and Discussuin of Important Questions**	
		Total No of Hours Planned for Unit II	**6**
		UNIT III	
S.NO	**Lecture Duration (Hours)**	**Topics To Be Covered**	**Support Materials/ Pg.No**
		Overview of Programming	
1	1	Python Inroduction	T1:1-4
		Python Features	W5
2	1	Structure of a Python Program	T1:6
		Elements of Python	
3	1	Variables	T1:17

		Data Tyes	
4	1	Arrays	T2:87-92
		Python List	
5	1	**Recapitulation and Discussuin of Important Questions**	
		Total No of Hours Planned for Unit III	**5**
		UNIT IV	
S.NO	**Lecture Duration (Hours)**	**Topics To Be Covered**	**Support Materials/ Pg.No**
		Introduction to Python	
		Python Interpreter	
1	1	Using Python as calculator	W5
2	1	Python shell Indentation	
3	1	Atoms	W4
4	1	Identifiers and keywords	T1:24-30
		Literals	
5	1	Strings	T2:71-75
		Operators	
6	1	Arithmetic operator	T3:40-50
		Relational operator	
7	1	Logical or Boolean operator	
		Assignment Operator,	
8	1	Ternary operator	
		Bit wise operator	
		Increment or Decrement operator	
9	1	Recapitulation and Discussuin of Important Questions	
		Total No of Hours Planned for Unit IV	**9**
		UNIT V	
S.NO	**Lecture Duration (Hours)**	**Topics To Be Covered**	**Support Materials/ Pg.No**
		Creating Python Programs	
1	1	Input and Output Statements	T3:51-55
		Control statements	
2	1	Branching	W5
3	1	Looping	T1:113-137
4	1	Conditional Statement	T1:67-83
		Exit function	W5
5	1	Difference between break continue and pass	T1:113-137
6	1	Defining Functions	T2:19-24

7	1	Default arguments	
8	1	Recapitulation and Discussuin of Important Questions	
9	1	Discussion of previous ESE question papers	
10	1	Discussion of previous ESE question papers	
11	1	Discussion of previous ESE question papers	
		Total No of Hours Planned for Unit V	**11**
Total Hours			**36**

S.NO	**TEXT BOOKS**
T1	Richard L. Halterman,"FUNDAMENTS OF PYTHON PROGRAMMIN" 2013
T2	Allen Downey,"Think Python How to Think Like a Computer Scientist",2012,Green Tea Press
T3	Andrew Johansen,"Python The Ultimate Beginner's Guide!",2016
S.NO	**WEB SITES**
W1	https://www.slideshare.net
W2	net-informations.com
W3	www.programiz.com
W4	www.guru99.com
W5	https://www.tutorialspoint.com

UNIT – I

SYLLABUS

Planning the Computer Program: Concept of problem solving-Problem definition-Program design-Debugging-Types of errors in programming-Documentation.

PLANNING THE COMPUTER PROGRAM:

Introduction to Computer Science

A **computer** is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format.

Computer science is the study of the theoretical foundations of information and computation, and of practical techniques for their implementation and application in computer systems.

A **program** is a sequence of instructions that can be executed by a computer to solve some problem or perform a specified task.

Programming language is an artificial language designed to automate the task of organizing and manipulating information, and to express problem solutions precisely.

A programming language "boils down to" a set of words, rules and tools that are used to explain (or define) what you are trying to accomplish. There are many different programming languages just as there are many different "spoken" languages.

Traditional programming languages were known as structural programming languages (e.g., C, Fortran, Pascal, Cobol, Basic). Since the late 80's however, object-oriented programming languages have become more popular (e.g., JAVA, C++, C#)

There are also other types of programming languages such as functional programming languages and logic programming languages. According to the Tiobe index (i.e., a good site for ranking the popularity of programming languages), as of February 2011 the 10 most actively used programming languages were (in order of popularity):

Java, C, C++, PHP, Python, C#, VisualBasic, Objective-C, Perl, Ruby

The **Computer Sciences Accreditation Board (CSAB)** identifies four general areas that it considers crucial to the discipline of computer science:

- theory of computation - investigates how specific computational problems can be solved efficiently
- **algorithms and data structures** - investigates efficient ways of storing, organizing and using data
- **programming methodology and languages** - investigates different approaches to

describing and expressing problem solutions

• **computer elements and architecture** - investigates the design and operation of computer systems

CONCEPT OF PROBLEM SOLVING

PROBLEM SOLVING

Regardless of the area of study, computer science is all about solving problems with computers. The problems that we want to solve can come from any real-world problem or perhaps even from the abstract world. We need to have a standard systematic approach to solving problems.

Problem Solving is the sequential process of analyzing information related to a given situation and generating appropriate response options.

There are 6 steps that you should follow in order to solve a problem:

1. Understand the Problem
2. Formulate a Model
3. Develop an Algorithm
4. Write the Program
5. Test the Program
6. Evaluate the Solution

In order to solve a problem by the computer, one has to pass though certain stages or steps. They are

1. Understanding theproblem
2. Analyzing theproblem
3. Developing the solution
4. Coding andimplementation.

Understanding the problem: Here we try to understand the problem to be solved in totally. Before withthenextstage or step, we should be absolutely sure about the objectives of the given problem.

- What input data/information is available?
- What does it represent? What format is it in?
- Is anything missing? Do I have everything that I need ?
- What output information am I trying to produce?
- What do I want the result to look like … text, a picture, a graph …?
- What am I going to have to compute?

Analyzing the problem: After understanding thoroughly the problem to be solved, we look different ways of solving the problem and evaluate each of these methods. The idea here is to search an appropriate solution to the problem under consideration. The end result of this stage is a broad overview of the sequence of operations that are to be carries out to solve the given problem.

Developing the solution: Here the overview of the sequence of operations that was the result of analysis stage is expanded to form a detailed step by step solution to the problem under

consideration.

Coding and implementation: The last stage of the problem solving is the conversion of the detailed sequence of operations in to a language that the computer can understand. Here each step is converted to its equivalent instruction or instructions in the computer language that has been chosen for the implantation.

Program Development Steps

The various steps involved in Program Development are:

- Defining or Analyzing the problem
- Design (Algorithm)
- Coding
- Documenting the program
- Compiling and Running the Program
- Testing and Debugging
- Maintenance

Analyzing or Defining the Problem

The problem is defined by doing a preliminary investigation. Defining a problem helps us to understand the problem clear. It is also known as Program Analysis.

Tasks in defining a problem:

- Specifying the input requirements
- Specifying the output requirements
- Specifying the processing requirements

Specifying the input requirements

Determine the inputs required and source of the data. The input specification is obtained by answering the following questions:

- What specific values will be provided as input to the program?
- What format will the values be?
- For each input item, what is the valid range of values that it may assume?
- What restrictions are placed on the use of these values?

Specifying the output requirements

Describe in detail the output that will be produced. The output specification is obtained by answering the following questions:

- What values will be produced?
- What is the format of these values?
- What specific annotation, headings, or titles are required in the report?
- What is the amount of output that will be produced?

Specifying the Processing Requirements

Determine the processing requirements for converting the input data to output. The

processing requirement specification is obtained by answering the following questions:

- What is the method (technique) required in producing the desired output?
- What calculations are needed?
- What are the validation checks that need to be applied to the input data?

Example 1.1 Find the factorial of a given number
Input: Positive valued integer number
Output: Factorial of that number
Process: Solution technique which transforms input into output. Factorial of a number can be calculated by the formula n! = 1*2*3*4….*n

Problem definition

The art of compiling logic in the form of general flow charts and logic diagrams which clearly explain and present the problem to the programmer in such a way that all requirements involved in the run are presented.

A computational problem is a mathematical object representing a collection of questions that computers might be able to solve. For example, the problem of factoring
"Given a positive integer n, find a nontrivial prime factor of n."

What is the mathematical object in the example above?

Commonly encountered mathematical objects include numbers, permutations, partitions, matrices, sets, functions, and relations.

So how can you 'represent' a collection of questions with numbers or permutations, matrices etc.? What is meant here is probably the following collection of sentences:

'find a nontrivial prime factor of 1', 'find a nontrivial prime factor of 2' and so on...

But the thing is - these sentences are not mathematical objects.

A little further in the article it reads:

A computational problem can be viewed as an infinite collection of instances together with a solution for every instance. Which makes perfect sense, but I don't quite see the relationship with the first definition.

Program design

Steps to Design
There are three fundamental steps you should perform when you have a program to write:

- Define the output and data flows.
- Develop the logic to get to that output.
- Write the program.

Program designing begins with deciding the output and framing the program logic. The

design is then broken down into modules to facilitate programming. All computer languages have a vocabulary of their own. If a programmer does not strictly follow the syntax of a programming language, the computer will not understand the commands given in the program.

DEBUGGING

Definition: Debugging is the process of detecting and removing of existing and potential errors (also called as 'bugs') in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

Description: To debug a program, user has to start with a problem, isolate the source code of the problem, and then fix it. A user of a program must know how to fix the problem as knowledge about problem analysis is expected. When the bug is fixed, then the software is ready to use. Debugging tools (called debuggers) are used to identify coding errors at various development stages. They are used to reproduce the conditions in which error has occurred, then examine the program state at that time and locate the cause. Programmers can trace the program execution step-by-step by evaluating the value of variables and stop the execution wherever required to get the value of variables or reset the program variables. Some programming language packages provide a debugger for checking the code for errors while it is being written at run time.

Here's thed ebugging process:

1. Reproduce the problem.

2. Describe the bug. Try to get as much input from the user to get the exact reason.

3. Capture the program snapshot when the bug appears. Try to get all the variable values and states of the program at that time.

4. Analyse the snapshot based on the state and action. Based on that try to find the cause of the bug.

5. Fix the existing bug, but also check that any new bug does not occur.

Code Debugging

Programming code might contain syntax errors, or logical errors.

Many of these errors are difficult to diagnose.

Often, when programming code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.

Searching for (and fixing) errors in programming code is called code debugging.

JavaScript Debuggers

Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.

Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.

Normally, otherwise follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

The console.log() Method

If your browser supports debugging, you can use console.log() to display JavaScript values in the debugger window:

```
<!DOCTYPE html>
<html>
<body>
<h1>MyFirstWebPage</h1>
<script>
a= 5;
b= 6;
c=a+b;
console.log(c);
</script>
</body>
</html>
```

Setting Breakpoints

In the debugger window, you can set breakpoints in the JavaScript code.

At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.

After examining values, you can resume the execution of code (typically with a play button).

The debugger Keyword

The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function.

This has the same function as setting a breakpoint in the debugger.

If no debugging is available, the debugger statement has no effect.

With the debugger turned on, this code will stop executing before it executes the third line.

Example

```
var x= 15 * 5;
debugger;
document.getElementById("demo").innerHTML = x;
```

Testing and Debugging

"Program testing can be used to show the presence of bugs, but never to show their absence!" (Dijkstra)

"If debugging is the process of removing software bugs, then programming must be the process of putting them in." (Anon)

"There are two ways to write error-free programs. Only the third one works." (Anon)
One of the first things you will discover in writing programs is that a program rarely runs correctly first time. In fact, errors are so common in programming that they have their own special name: **Bugs**. The process of correcting, or getting rid of bugs is called **Debugging.**

Testing

Testing is the process of executing programs with the intention of finding errors.
"A good test is one that has a high probability of finding an, as yet, undiscovered error."
In fact, a good program tester will try to make programs fail.

When considering the kind of tests to apply, a **systematic approach** should be used and **test cases** should be developed which will uncover common classes of errors. Never rely on intuition. The data collected during the testing should be kept as a means of demonstrating the correct operation of the program. This is important for future maintenance.

TYPES OF PROGRAM ERRORS

We distinguish between the following types of errors:

1. **Syntax errors**: errors due to the fact that the syntax of the language is not respected.
2. **Semantic errors**: errors due to an improper use of program statements.
3. **Logical errors**: errors due to the fact that the specification is not respected.

From the point of view of when errors are detected, we distinguish:

1. **Compile time errors**: syntax errors and static semantic errors indicated by the compiler.
2. **Runtime errors**: dynamic semantic errors, and logical errors, that cannot be detected by the compiler (debugging).

There are three basic categories of program errors:
1. Syntax Errors
2. Run-time Errors
3. Logic Flaws

In the first two cases when an error occurs, the computer displays an **'Error Message'**, which describes the error, and its cause. Unfortunately, error messages are often **difficult to interpret**, and are sometimes **misleading**. In the final case, the program will not show an error message but it will not do what the programmer wanted it to do.

Syntax Errors

These errors are the easiest to find because they are highlighted by the compiler. Error messages are given. This type of error is caused by the failure of the programmer to use the correct **grammatical rules** of the language.

Syntax errors are detected, and displayed, by the **compiler** as it attempts to translate your program, ie: the **Source** code into the **Object** code. If a program has a syntax error it cannot be translated, and the program will not be executed.

The compiler tries to highlight syntax errors where there seems to be a problem, however, it is not perfect and sometimes the compiler will indicate the next line of code as having the problem rather than the line of code where the problem actually exists.

run-time errors are detected by the computer and displayed **during** execution of a program. They will halt the program when they occur but they often do not show up for some time. Ex: When you forgot to type a semicolon (;) after the statement, the compiler shows the syntax error and it would point out where the problem occurred.

Run-time Errors

A run-time error occurs when the user directs the computer to perform an **illegal operation**, eg:

- Dividing a number by zero
- Assigning a variable to the wrong type of variable
- Using a variable in a program before assigning a value to it.

When a run-time error occurs, the computer stops executing your program, and displays a **diagnostic message** that indicates the line where the error occurred.

A runtime error is a program error that occurs while the program is running. The term is often used in contrast to other types of program errors, such as syntax errors and compile time errors.

There are many different types of runtime errors. One example is a logic error, which produces the wrong output. For example, a miscalculation in the source code or a spreadsheet program may produce the wrong result when a user enters a formula into a cell.

Another type of runtime error is a memory leak. This type of error causes a program to continually use up more RAM while the program is running. A memory leak may be due to an infinite loop, not deallocating unused memory, or other reasons.

A program crash is the most noticeable type of runtime error, since the program unexpectedly quits while running. Crashes can be caused by memory leaks or other

programming errors. Common examples include dividing by zero, referencing missing files, calling invalid functions, or not handling certain input correctly.

NOTE: Runtime errors are commonly called referred to as "bugs," and are often found during the debugging process, before the software is released. When runtime errors are found after a program has been distributed to the public, developers often release patches, or small updates, designed fix the errors.

Logic Flaws

These are the hardest errors to find as they do not halt the program. They arise from faulty thinking on behalf of the programmer. They can be very troublesome.
These are mistakes in a program's logic.

Programs with logic errors will often compile, execute, and output results. However, at least some of the time the output will be incorrect. Error messages will generally not appear if a logic error occurs, this makes logic errors very difficult to locate and correct.

The Differences between Compile-Time and Run-Time Error are:

COMPILE-TIME ERRORS	RUNTIME-ERRORS
These are the syntax errors which are detected by the compiler.	These are the errors which are not detected by the compiler and produce wrong results.
They prevent the code from running as it detects some syntax errors.	They prevent the code from complete execution.
It includes syntax errors such as missing of semicolon(;), misspelling of keywords and identifiers etc.	It includes errors such as dividing a number by zero, finding square root of a negative number etc.

Syntax and logical errors in Python

Two types of errors can occur in Python:
1. Syntax errors – usually the easiest to spot, syntax errors occur when you make a typo. Not ending an **if** statement with the colon is an example of an syntax error, as is misspelling a Python keyword (e.g. using **whille** instead of **while**). Syntax error usually appear at compile time and are reported by the interpreter. Here is an example of a syntax error:

```
x = int(input('Enter a number: '))

whille x%2 == 0:
   print('You have entered an even number.')
else:
   print ('You have entered an odd number.')
```

```
C:Python34Scripts>python error.py
  File "error.py", line 3
    whille x%2 == 0:
         ^
SyntaxError: invalid syntax
```

```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))

z = x+y/2
print ('The average of the two numbers you have entered is:',z)
```

The example above should calcuate the average of the two numbers the user enters. But, because of the order of operations in arithmetic (the division is evaluated before addition) the program will not give the right answer:

```
>>>
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is: 5.0
>>>
```

To rectify this problem, we will simply add the parentheses: **z = (x+y)/2**
Now we will get the right result:

```
>>>
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is: 3.5
>>>
```

PROGRAMDOCUMENTATION

Any written text, illustrations or video that describe a software or program to its users is called **program or software document**. User can be anyone from a programmer, system

analyst and administrator to end user. At various stages of development multiple documents may be created for different users. In fact, **software documentation** is a critical process in the overall software development process.

In modular programming documentation becomes even more important because different modules of the software are developed by different teams. If anyone other than the development team wants to or needs to understand a module, good and detailed documentation will make the task easier.

These are some guidelines for creating the documents −

- Documentation should be from the point of view of the reader
- Document should be unambiguous
- There should be no repetition
- Industry standards should be used
- Documents should always be updated
- Any outdated document should be phased out after due recording of the phase out

Advantages of Documentation

These are some of the advantages of providing program documentation −

- Keeps track of all parts of a software or program
- Maintenance is easier
- Programmers other than the developer can understand all aspects of software
- Improves overall quality of the software
- Assists in user training
- Ensures knowledge de-centralization, cutting costs and effort if people leave the system abruptly

Example Documents

Software can have many types of documents associated with it. Some of the important ones include −

- **User manual** − It describes instructions and procedures for end users to use the different features of the software.
- **Operational manual** − It lists and describes all the operations being carried out and their inter-dependencies.
- **Design Document** − It gives an overview of the software and describes design

elements in detail. It documents details like **data flow diagrams, entity relationship diagrams**, etc.

- **Requirements Document** − It has a list of all the requirements of the system as well as an analysis of viability of the requirements. It can have user cases, reallife scenarios, etc.
- **Technical Documentation** − It is a documentation of actual programming components like algorithms, flowcharts, program codes, functional modules, etc.
- **Testing Document** − It records test plan, test cases, validation plan, verification plan, test results, etc. Testing is one phase of software development that needs intensive documentation.
- **List of Known Bugs** − Every software has bugs or errors that cannot be removed because either they were discovered very late or are harmless or will take more effort and time than necessary to rectify. These bugs are listed with program documentation so that they may be removed at a later date. Also they help the users, implementers and maintenance people if the bug is activated.

Possible Questions:

2 Mark Questions

1. Mention the types of programming language?
2. Assess problem solving method.
3. What is mean by problem solving?
4. What are the properties of algorithm?
5. Define programming language

6 Mark Questions

1. Analyze the notations used in algorithmic problem solving
2. Summarize advantage and disadvantage of flow chart
3. How to prepare documentation in programming language?
4. Explain in detail about problem solving techniques?
5. Mention the types of errors in programming. Explain it.
6. Discover the steps of simple strategies for developing algorithms
7. Illustrate with example Programming methodologies in python.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

III B.Sc(CS) (BATCH 2017-2020) V SEMESTER

PROGRAMMING IN PYTHON (17CSU504B)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT I

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	Last step in process of problem solving is to	design a solution	define a problem	practicing the solution	organizing the data	practicing the solution
2	Second step in problem solving process is to	practicing the solution	organizing the data	design a solution	define a problem	design a solution
3	Thing to keep in mind while solving a	input data	output data	stored data	all of above	all of above
4	First step in process of problem solving is to	design a solution	define a problem	practicing the solution	organizing the data	define a problem
5	Stock maintaining control system in which both hardware and software is present to contact and manage suppliers is called	power system	control system	information processing system	operating system	information processing system
6	All components that work together in the form of unit is called	system	bound	baud	operator	system
7	The main task of a problem-solving agent is	Solve the given problem and reach to goal	To find out which sequence of action will get it to the goal state	Both a and b	Neither a nor b	Both a and b
8	A problem in a search space Is defined by	Initial state	Goal test	Intermediate states	Both a and b	Both a and b

9	The process of removing detail from a given state representation is called_____.	Extraction	Abstraction	Information Retrieval	Mining of data	Abstraction
10	Web Crawler is a/an	Intelligent goal-based agent	Problem-solving agent	Simple reflex agent	Both a and b	Both a and b
11	Error in a program is called	bug	debug	virus	noise	bug
12	Error which occurs when program tried to read from file without opening it is classified as	execution error messages	built in messages	user-defined messages	half messages	execution error messages
13	Error which occurs when user tried to use a device which is not switched ON is classified as	user-defined message	half message	execution error message	built in message	execution error message
14	Types of software programs are	Application programs	Replicate programs	Logical programs	both A and B	both A and B
15	Specialized program that allows the user to utilize in specific application is classified as	relative programs	application programs	relative programs	replicate programs	application programs
16	Program which is used to produce pictures, text and to organize it in newspaper is classified as	text publishing package	desktop publishing package	experimental package	organizing publishing package	desktop publishing package
17	Application program example includes	payroll program	desktop program	publishing program	editing program	payroll program
18	Application program used with all the documentation is considered	applications package	Replicate programs	Logical programs	systems programs	applications package
19	Set of programs which consist of full set of documentations is termed as	database packages	file packages	bus packages	software packages	software packages
20	Program which is readily available for computer users as a part of software package is classified as	library program	program library	software library	directory library	library program

21	Set of software authorized to a specific users is considered as	library program	program library	software library	directory library	program library
22	Programs are fully tested and documented properly before including it into	library	directory	package	database	library
23	To write a program function i.e. program for the sum of four integers, the program refinement first level includes	input four numbers	calculate sum	print the values	display the values	input four numbers
24	Data which is used to test each feature of the program and is carefully selected is classified as	program output	program input	test data	test program	test data
25	Function definition and first level refinement are part of	program design	program statement	program calculation	printing the program	program design
26	There are _____steps to solve the problem	7	4	6	2	6
27	_____is the first step in solving the problem	Understanding the Problem	Identify the Problem	Evaluate the Solution	None of these	Identify the Problem
28	_____is the last step in solving the problem	Understanding the Problem	Identify the Problem	Evaluate the Solution	None of these	Evaluate the Solution
29	Following is true for understanding of a problem	Knowing the knowledgebase	Understanding the subject on which the problem is based	Communication with the client	All of the above	All of the above

30	The six-step solution for the problem can be applied to I. Problems with Algorithmic Solution II. Problems with Heuristic Solution	Only I	Only II	Neither I nor II	Both I and II	Both I and II
31	While solving the problem with computer the most difficult step is _____.	describing the problem	finding out the cost of the software	writing the computer instructions	testing the solution	writing the computer instructions
32	The help menus or user manuals are the part of _____.	Program	Algorithm	Internal Documentation	External Documentation	External Documentation
33	The branch of computer that deals with heuristic types of problem is called _____.	system software	real time software	artificial intelligence	none of these	artificial intelligence
34	The correctness and appropriateness of _____solution can be checked very easily.	algorithmic solution	heuristic solution	random solution	none of these	algorithmic solution
35	Access in which records are accessed from and inserted into file, is classified as	direct access	sequential access	random access	duplicate access	sequential access
36	Distinct parts of documentation are called	technical documentation	documentation for user	program planning	both a and b	both a and b
37	Hardware and software specifications are part of	computing requirements	statement requirements	system flowchart	decision statement	computing requirements
38	Set of diagrams and notes that accompany program implementation are known as	program execution	program planning	program documentation	program existence	program documentation

39	Program documentation is used by the	programmers	system analyst	modifying the program	all of above	all of above
40	Program background, program functions and the computing requirements are part of	operations detail	predefined programs	decision box	Statement box	operations detail

UNIT – II

SYLLABUS

Techniques of Problem Solving: Flowcharting-decision table-algorithms-Structured programming concepts-Programming methodologies: top-down and bottom-upProgramming.

Techniques of Problem Solving

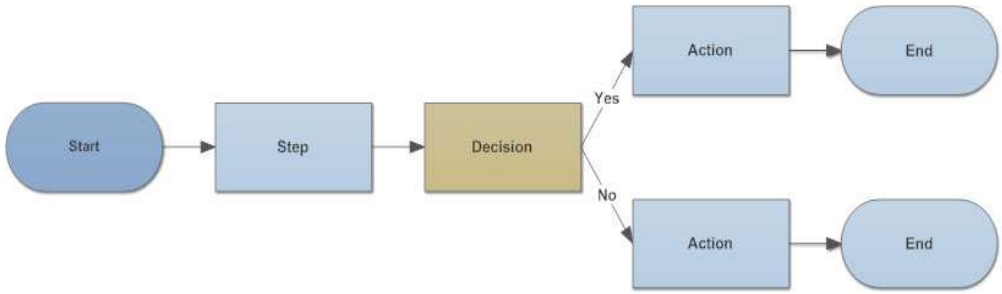
Flowcharting

A flowchart is simply a graphical representation of steps. It shows steps in a sequential order, and is widely used in presenting flow of algorithms, workflow or processes. Typically, flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows.

What is a Flowchart?

Flowcharts are graphical representation of steps. It was originated from computer science as a tool for representing algorithms and programming logic, but had extended to use in all other kinds of processes. Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning. They help us visualize complex processes, or make explicit the structure of problems and tasks. A flowchart can also be used to define a process or project to be implemented.

A Basic Flowchart



When to Draw Flowchart?

Using a flowchart has a variety of benefits:

- It helps to clarify complex processes.
- It identifies steps that do not add value to the internal or external customer, including: delays; needless storage and transportation; unnecessary work, duplication, and added expense; breakdowns in communication.
- It helps team members gain a shared understanding of the process and use this knowledge to collect data, identify problems, focus discussions, and identify resources.
- It serves as a basis for designing new processes.

Symbols used in Flow-Charts

The symbols that we make use while drawing flowcharts as given below are as per conventions followed by International Standard Organization (ISO).Different flowchart shapes have different conventional meanings. The meanings of some of the more common shapes are as follows:

Terminator

The terminator symbol represents the starting or ending point of the system.

Process

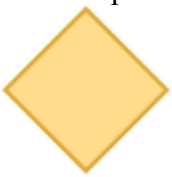
A box indicates some particular operation.

Document

This represents a printout, such as a document or a report.

Decision

A diamond represents a decision or branching point. A line coming out from the diamond indicates different possible situations, leading to different sub-processes.



Data

It represents information entering or leaving the system. An input might be an order from a customer. An output can be a product to be delivered.

On-Page Reference

This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on the same page.

Off-Page Reference

This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on a different page.

Delay or Bottleneck

Identifies a delay or a bottleneck.

Flow

Lines represent flow of the sequence and direction of a process.

Rules for drawing a flowchart

1. The flowchart should be clear, neat and easy tofollow.
2. The flowchart must have a logical start andfinish.
3. Only one flow line should come out from a processsymbol.
4. Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decisionsymbol.
5. Only one flow line is used with a terminalsymbol.
6. Within standard symbols, write briefly andprecisely.
7. Intersection of flow lines should beavoided.

Advantages of flowchart:

1. **Communication:** - Flowcharts are better way of communicating the logic of a system to all concerned.
2. **Effective analysis:** - With the help of flowchart, problem can be analyzed in more effective way
3. Proper documentation: - Program flowcharts serve as a good program documentation, which is needed for variouspurposes.
4. **Efficient Coding: -** The flowcharts act as a guide or blueprint during the systems analysis and program developmentphase.
5. **Proper Debugging: -** The flowchart helps in debuggingprocess.

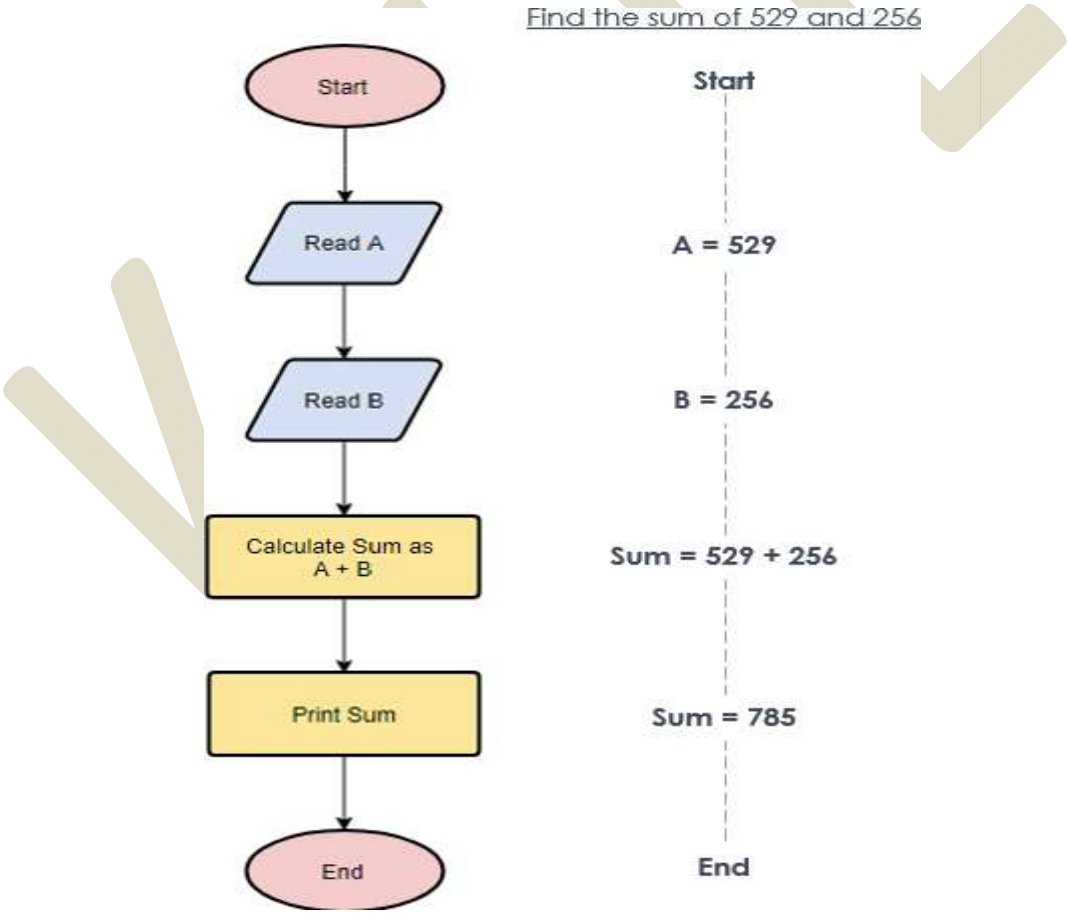
6. **Efficient Program Maintenance: -** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

1. **Complex logic: -** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex andclumsy.
2. **Alterations and Modifications: -** If alterations are required the flowchart may require re-drawingcompletely.
3. **Reproduction: -** As the flowchart symbols cannot be typed, reproduction of flowchart becomes aproblem.
4. **Cost:** For large application the time and cost of flowchart drawing becomescostly.

Flowchart Examples

1. Flowchart can also be used in visualizing an algorithm, regardless of its complexity. Here is an example that shows how flowchart can be used in showing a simple summation process.



2. Draw the Flowchart to find Roots of Quadratic equation $ax^2 + bx + c = 0$. The coefficients a, b, c are the input data

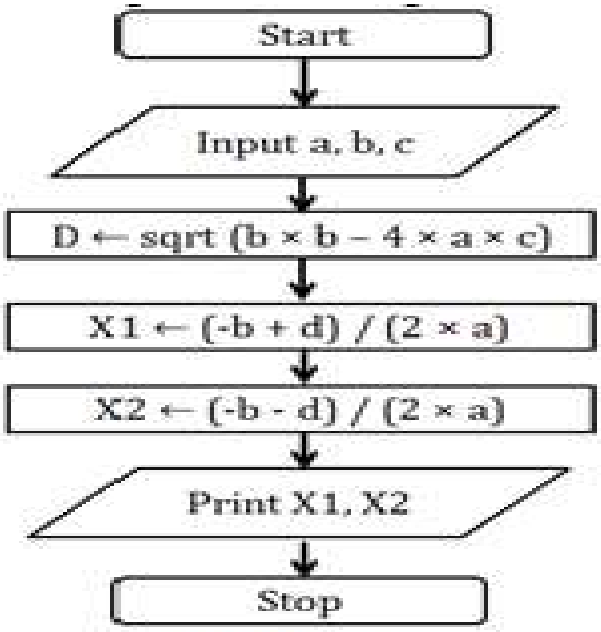
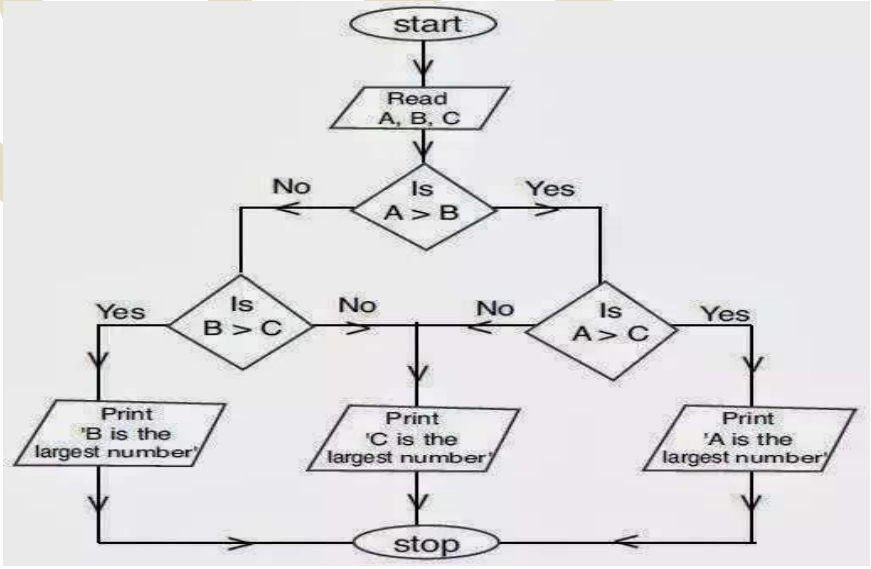


Figure 3

3. Draw a flowchart to find out the biggest of the three unequal positive numbers.



Decision Table:

A decision table is an excellent tool to use in both testing and requirements management. Essentially it is a structured exercise to formulate requirements when dealing with complex business rules. Decision tables are used to model complicated logic.

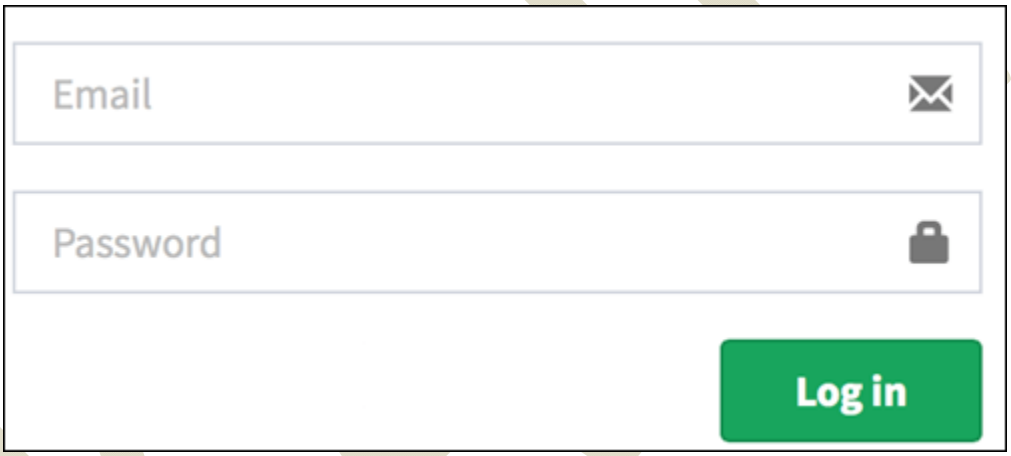
What is Decision Table Testing?

Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.

A Decision Table is a tabular representation of inputs versus rules/cases/test conditions. Let's learn with an example.

Example 1: How to make Decision Base Table for Login Screen

Let's create a decision table for a login screen.



The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend:

- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Interpretation:

- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

While converting this to test case, we can create 2 scenarios,

- Enter correct username and correct password and click on login, and the expected result will be the user should be navigated to homepage

And one from the below scenario

- Enter wrong username and wrong password and click on login, and the expected result will be the user should get an error message
- Enter correct username and wrong password and click on login, and the expected result will be the user should get an error message
- Enter wrong username and correct password and click on login, and the expected result will be the user should get an error message

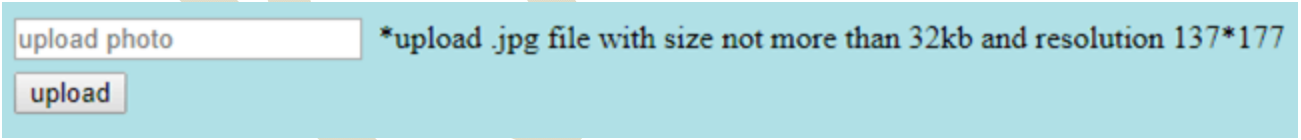
As they essentially test the same rule.

Example 2: How to make Decision Table for Upload Screen

Now consider a dialogue box which will ask the user to upload photo with certain conditions like –

1. You can upload only '.jpg' format image
2. file size less than 32kb
3. resolution 137*177.

If any of the conditions fails the system will throw corresponding error message stating the issue and if all conditions are met photo will be updated successfully



Let's create the decision table for this case.

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb

resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

For this condition, we can create 8 different test cases and ensure complete coverage based on the above table.

1. Upload a photo with format '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Photo should upload successfully
2. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message resolution mismatch should be displayed
3. Upload a photo with format '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message size mismatch should be displayed
4. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message size and resolution mismatch should be displayed
5. Upload a photo with format other than '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format mismatch should be displayed
6. Upload a photo with format other than '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message format and resolution mismatch should be displayed
7. Upload a photo with format other than '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format and size mismatch should be displayed
8. Upload a photo with format other than '.jpg', size more than 32kb and resolution not 137*177 and click on upload. Expected result is Error message for format, size and resolution mismatch should be displayed

Why is Decision Table Testing is important?

This testing technique becomes important when it is required to test different combination. It also helps in better test coverage for complex business logic.

In Software Engineering, boundary value and equivalent partition are other similar techniques used to ensure better coverage. They are used if the system shows the **same** behavior for a large set of inputs. However, in a system where for each set of input values the system behavior is **different**; boundary value and equivalent partitioning technique are not effective in ensuring good test coverage.

In this case, decision table testing is a good option. This technique can make sure of good coverage, and the representation is simple so that it is easy to interpret and use.

This table can be used as the reference for the requirement and for the functionality development since it is easy to understand and cover all the combinations.

The significance of this technique becomes immediately clear as the number of inputs increases. Number of possible Combinations is given by 2 ^ n , where n is the number of Inputs. For n = 10, which is very common in the web based testing, having big input forms, the number of combinations will be 1024. Obviously, you cannot test all but you will choose a rich sub-set of the possible combinations using decision based testing technique.

Advantages of Decision Table Testing

- Any complex business flow can be easily converted into the test scenarios & test cases using this technique.
- Decision tables work iteratively that means the table created at the first iteration is used as input table for next tables. The iteration is done only if the initial table is not satisfactory.
- Simple to understand and everyone can use this method design the test scenarios & test cases.
- It provides complete coverage of test cases which help to reduce the rework on writing test scenarios & test cases.
- These tables guarantee that we consider every possible combination of condition values. This is known as its completeness property.
- When the system behavior is different for different input and not same for a range of inputs, both equivalent partitioning, and boundary value analysis won't help, but decision table can be used.
- The representation is simple so that it can be easily interpreted and is used for development and business as well.
- This table will help to make effective combinations and can ensure a better coverage for testing
- Any complex business conditions can be easily turned into decision tables
- In a case we are going for 100% coverage typically when the input combinations are low, this technique can ensure the coverage.

Disadvantages of Decision Table Testing

The main disadvantage is that when the number of input increases the table will become more complex

ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.

Properties of Algorithms

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.
- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Steps involved in algorithm development

Analgorithmcanbedefinedas**"acomplete,unambiguous,finitenumberoflogicalsteps forsolving a specific problem"**

Step1. Identification of input: For an algorithm, there are quantities to be supplied called input and thesearefed externally. Theinputistobe identifiedfirst foranyspecifiedproblem.

Step2: Identification of output: From an algorithm, at least one quantity is produced, called for any specifiedproblem.

Step3: Identification the processing operations: All the calculations to be performed in order to leadto output fromtheinput are to be identified in anorderlymanner.

Step4: Processing Definiteness: The instructions composing the algorithm must be clear and there should not be any ambiguity in them.

Step5:ProcessingFiniteness:Ifwegothroughthealgorithm,thenforallcases,thealgorithmshouldterminatea fterafinitenumberofsteps.

Step6: Possessing Effectiveness: The instructions in the algorithm must be sufficiently basic and in practice they can be carries out easily.

An algorithm must possess the following properties

1. **Finiteness**:Analgorithmmustterminateinafinitenumberofsteps
2. **Definiteness**: Each step of the algorithm must be precisely and unambiguouslystated
3. **Effectiveness**: Each step must be effective, in the sense that it should be primitive easily convert ableintoprogramstatement)canbeperformedexactlyinafiniteamountoftime.
4. **Generality**:Thealgorithmmustbecompleteinitselfsothat itcanbeusedtosolveproblemsofa

specifictypefor any inputdata.

5. **Input/output**:Eachalgorithmmusttakezero,oneormorequantitiesasinputdataproduceoneor moreoutputvalues.Analgorithmcanbewritteninenglishlikesentencesorinanystandard representation sometimes, algorithm written in English like languages are called Pseudo Code

Example

Write an algorithm to check whether he is eligible to vote? Step 1: Start
Step 2: Get age
Step 3: if age >= 18 print "Eligible to vote"
Step 4: else print "Not eligible to vote" Step 6: Stop
Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example:

Write an algorithm to print all natural numbers up to n Step

1: Start
Step 2: get n value.
Step 3: initializei=1
Step 4: if (i<=n) go to step 5 else go to step 7
Step 5: Print i value and increment i value by 1 Step 6: go to step4
Step 7: Stop

Pseudo code

Writing a program is often called "writing code" or "implementing an algorithm". So the code (or source code) is actually the program itself. **Pseudocode**is a simple and concise sequence of English-like instructions to solve a problem

The **Pseudo code** is neither an algorithm nor a program. It is an abstract form of a program. It consists of English like statements which perform the specific operations. It is defined for analgorithm. It does not use any graphical representation. In pseudo code, the program is represented in termsofwords and phrases, but thesyntax of program is not strictly followed.

Advantages: * Easy to read, * Easy to understand, * Easy to modify.

Example*:* Writeapseudocodetoperformthebasicarithmetic operations.

Read n1, n2
Sum = n1 + n2

Diff = n1 – n2

Mult = n1 * n2

Quot = n1/n2

Print sum, diff, mult, quotEnd.

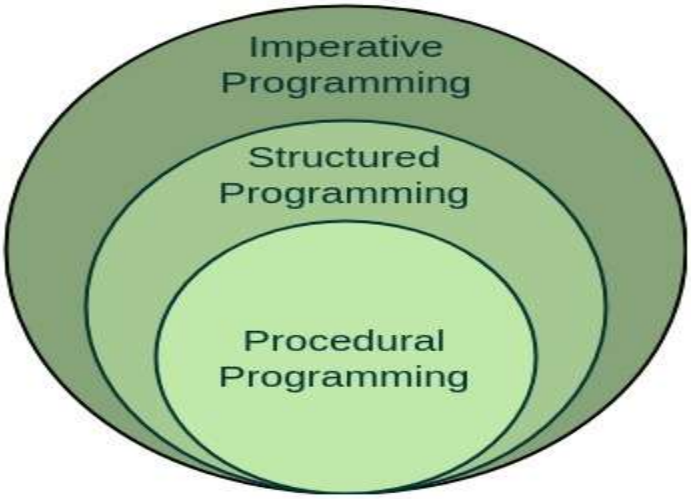
Compiling is the process of converting a program into instructions that can be understood by the computer.

Structured Programming Concepts:

Structured programming(sometimes known as Modular Programming) is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:

- C
- C++
- Java
- C# ..etc



On the contrary, in the Assembly languages like Microprocessor 8085, etc, the statements do not get executed in a structured manner. It allows jump statements like GOTO. So the program flow might be random.

The structured program mainly consists of three types of elements:

- Selection Statements
- Sequence Statements
- Iteration Statements

The structured program consists of well-structured and separated modules. But the entry and exit in a structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach:

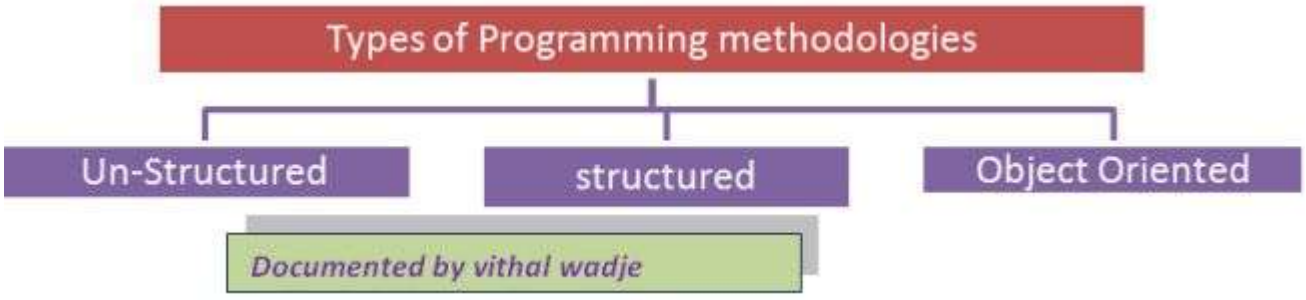
1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Programming Methodology

Programming Methodology is the approach to analyzing such complex problems by planning the software development and controlling the development process.
The following are some of the common types of programming methodologies.



Unstructured Programming

A programming language in which the entire logic of the program is written as a single continuous (nonstop or unbroken) block is called *unstructured programming*.

The following are the key points of unstructured programming languages:

- It is a type of problem-solving technique, in which we solve the problem in terms of lots of code. If the requirements increase then the code is also increased.
- There is no re-usability of existing code.
- Finding an error in a program is very difficult.
- Syntax and structure is very difficult to understand and remember.
- Modification is very difficult.
- Non-structured languages allow only basic data types, such as numbers, strings and arrays.
- Statements are usually one in each line.

For example:

JOSS, FOCAL, MUMPS, TELCOMP, COBOL, FORTRAN and so on.

2**. Structured/Procedure/Functional/Logical Programming Language**

A programming language in which the entire logic of the program is written by dividing it into smaller units or modules is called a *Structured Programming Language*.

The following are the key points of Structured Programming languages:

- Finding an error in a program is easily.
- Syntax and structure is very simple to understand and remember.
- Modification is easy.
- There is no re-usability of existing code as much as expected.

For example,

C and so on.

3**. Object Oriented Programming**

Object Oriented Programming (OOP) is the programming methodology in which each entity is an **object**.

This provides the following key advantages.

- Re-usability of existing code.
- Extensibility of existing code.
- Security.
- Flexibility.
- Maintainability.

The following are the key points of OOP languages:

- Classes
- Objects
- Methods
- abstraction
- Encapsulation
- Inheritance
- Polymorphism

And so on.

For example:

C++, Java, C#, F#, VB.net .and so on.

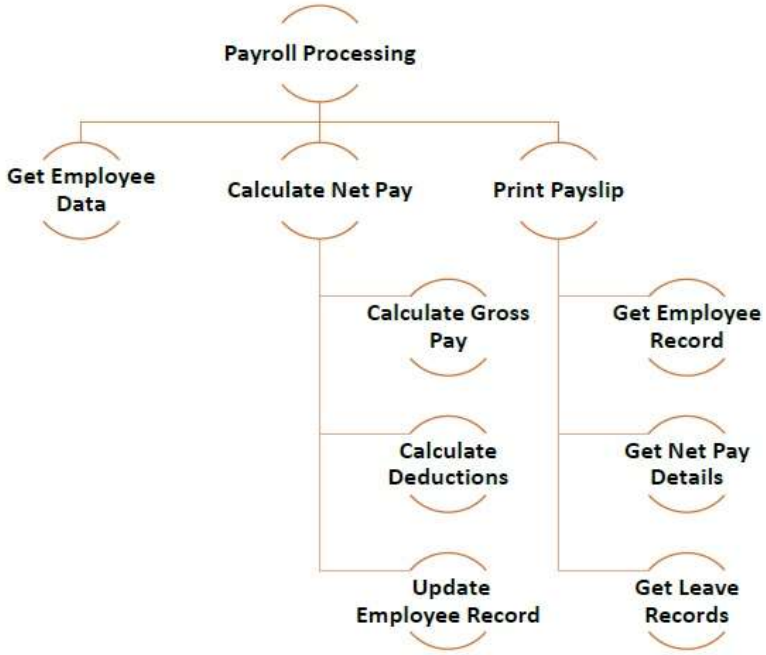
Software developers may choose one or a combination of more than one of these methodologies to develop software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches −

- Top-down approach
- Bottom-up approach

Top-down or Modular Approach

The problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a **module**. Each module is a self-sufficient unit that has everything necessary to perform its task.

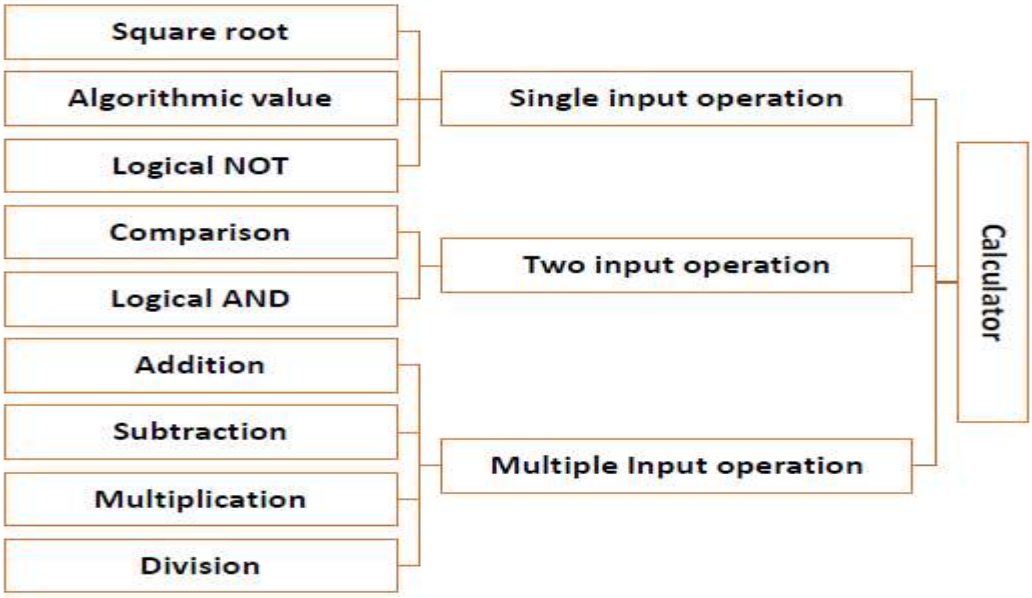
The following illustration shows an example of how you can follow modular approach to create different modules while developing a payroll processing program.



Bottom-up Approach

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the outset. So bottoms up approach are used only for very simple problems.

Let us look at the components of a calculator program.



What is the Similarity between Structured and Unstructured Programming?

- Both are programming paradigms.

What is the Difference between Structured and Unstructured Programming?

Structured vs Unstructured Programming	
Structured Programming is a programming paradigm which divides the code into modules or function.	Unstructured Programming is the paradigm in which the code is considered as one single block.
Readability	
Structured Programming based programs are easy to read.	Unstructured Programming based programs are hard to read.
Purpose	
Structured Programming is to make the code more efficient and easier to understand.	Unstructured programming is just to program to solve the problem. It does not create a logical structure.
Complexity	
Structured Programming is easier because of modules.	Unstructured programming is harder when comparing with the structured programming.
Application	
Structured programming can be used for small and medium scale	Unstructured programming is not applicable for medium and complex

projects.	projects.
Modification	
It is easy to do changes in Structured Programming.	It is hard to do modifications in Unstructured Programming.
Data Types	
Structured programming uses many data types.	Unstructured programming has a limited number of data types.
Code Duplication	
Structured programming avoids code duplication.	Unstructured programming can have code duplication.
Testing and Debug	
It is easy to do testing and debugging in Structured Programming.	It is hard to do testing and debugging in Unstructured programming.

Possible Questions:

2 Mark Questions

1. Distinguish between pseudo code and flowchart.
2. Define an algorithm
3. Define control flow statement with an eg:
4. Draw a flow chart for factorial given number(3*3)
5. Draw the flow chart sum of n numbers
6. What is pseudo code?

6 Mark Questions

1. Explain the basic symbols for constructing a flowchart & discuss about the basic guidelines for preparing flow chart.
2. Develop algorithm for i) Prime number or not (ii) odd or even
3. Develop algorithm for Celsius to Fahrenheit and vice versa
4. Discuss building blocks of algorithm
5. Summarize the symbol used in flowchart
6. Develop algorithm and flow chart for Celsius to Fahrenheit and vice versa.
7. Summarize decision table with neat sketch.
8. Write algorithm, pseudo code and flow chart for any example?

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

III B.Sc(CS) (BATCH 2017-2020) V SEMESTER

PROGRAMMING IN PYTHON (17CSU504B)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT II

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	A search algorithm takes _____ as an input and returns _____ as an output.	Input, output	Problem, solution	Solution, problem	Parameters, sequence of actions	Parameters, sequence of actions
2	In flow chart, diamond shaped symbol is used to represent	decision box	statement box	error box	if-statement box	decision box
3	The repetition of statement in the Jackson method is represented by	drawing sign of equal	drawing sign of subtraction	drawing an asterisk	drawing an arrow	drawing an asterisk
4	Symbol used in flowchart such as rectangle with the horizontal lines on two sides is used for	defined statement	predefined process	error fix	variables defined	predefined process
5	Program link with other parts of the program or connectors in flowchart are represented by	rhombus	parallelogram	circle	trapezoid	circle
6	Part of algorithm which is repeated for the fixed number of times is classified as	iteration	selection	sequence	reverse action	iteration
7	Defining data requirements such as input and output is classified as	process definition	function definition	print definition	writing purpose	function definition
8	Method used in writing and designing of a program is termed as	bottom-up method	top-down method	split method	binary states method	top-down method
9	The PAC stands for	Program Analysis Chart	Problem Algorithm Code	Problem Access Code	Problem Analysis Chart	Problem Analysis Chart

10	In interactivity chart the darkened circle indicates _____.	duplicate module	loop	decision	no special meaning	loop
11	. In interactivity chart the diamond indicates _____.	duplicate module	loop	decision	no special meaning	decision
12	The interactivity chart is also known as _____.	IPO Chart	Problem Analysis Chart	flow chart	structure chart	structure chart
13	The IPO stands for	Input Programming Option	Input Programming Output	Input Processing Output	Input Operating Operation	Input Processing Output
14	The main measure for efficiency algorithm are-	Processor and Memory	Complexity and Capacity	Data and Space	Time and space	Time and space
15	The time factor when determining the efficiency of algorithm is measured by	Counting microseconds	Counting the number of key operations	Counting the number of statements	Counting the kilobytes of algorithm	Counting the number of key operations
16	For retail shopping software which table would be example of Decision Table?	A table containing rules of discount.	A table containing rules for interfaces between components.	A table containing rule of employee behavior.	A table containing rules for combination of input.	A table containing rules of discount.
17	When different combination of input requires different combination of actions,Which of the following technique is used in such situation?	Boundary Value Analysis	Equivalence Partition	Decision Table	Decision Coverage	Decision Table
18	Which of the following Use Cases are useful?	Performance Testing	Business Scenarios	Static Testing	Unit Testing	Business Scenarios

19	A decision table is	a truth table	a table which facilitates taking decisions	a table listing conditions and actions to be taken based on the testing of conditions	a table in a Decision Support System	a table listing conditions and actions to be taken based on the testing of conditions
20	A decision table	has a structured English equivalent representation	cannot be represented using structured English	does not have an equivalent algorithmic representation	cannot be used to represent processes in a DFD	has a structured English equivalent representation
21	A decision table is preferable when the number of	conditions to be checked in a procedure is small	conditions to be checked in a procedure is large	actions to be carried out are large	actions to be carried out are small	conditions to be checked in a procedure is large
22	Decision table description of data processing is	non-procedural specification	procedural specification	purely descriptive specification	very imprecise specification	non-procedural specification
23	A rule in a limited entry decision table is a	row of the table consisting of condition entries	row of the table consisting of action entries	column of the table consisting of condition entries and the corresponding action entries	columns of the tables consisting of conditions of the stub	column of the table consisting of condition entries and the corresponding action entries
24	In a limited entry decision table the condition entries may be	Y or N only	Y, N or –	A binary digit	Any integer	Y, N or –

25	In a limited entry decision table the action entries	list X or – corresponding to actions to be executed	list the conditions to be tested	have Y or N or – entries	list the actions to be taken	list X or – corresponding to actions to be executed
26	Decision Tables are preferred when	Too many conditions need to be tested	Sequencing of testing conditions is important	When there are many loops to be performed	When too many actions are to be taken	Too many conditions need to be tested
27	The data type for numbers such as 3.14159 is:	double	int	real	string	double
28	To write a program function i.e. program for the sum of four integers, the program refinement first level includes	input four numbers	calculate sum	print the values	display the values	input four numbers
29	Data which is used to test each feature of the program and is carefully selected is classified as	program output	program input	test data	test program	test data
30	Object oriented programming employs_____ programming approach.	top-down	procedural	bottom-up	all of these.	bottom-up
31	_____are used for generic programming.	Inheritance	Virtual Functions	Templates	None of these	Templates
32	Higher-order functions are not built into the	structural language	object oriented programming	JAVA	C++	structural language
33	In structural language, we can't add a new sort of	loop	function	variable	constant	loop
34	No matter how well the structured programming is implemented, large programs become excessively	complex	simpler	formal	network	complex

35	Unrelated functions and data in procedural paradigm provides real world	poor model	simplest model	formal model	good model	poor model
36	Function has unrestricted access to	global data	local data	local variable	global name	global data
37	The _____ provides pictorial representation of given problem.	Algorithm	Flowchart	Pseudocode	All of these	Flowchart
38	_____ is a procedure or step by step process for solving a problem.	Algorithm	Flowchart	Pseudocode	All of these	Algorithm
39	The _____ symbol is used at the beginning of a flow chart.	Circle	Rectangle	Diamond	None of these	Circle
40	The _____ symbol is used to represent decision in flowchart.	Circle	Rectangle	Diamond	None of these	Diamond
41	The _____ symbol is used to represent process in flowchart.	Circle	Rectangle	Diamond	None of these	Rectangle
42	_____ symbol is used to represent input and output operation in flowchart.	Circle	Rectangle	Diamond	Parallelogram	Parallelogram
43	_____ is a symbol used connects two symbols of flowchart.	Circle	Rectangle	Diamond	Arrow	Arrow

UNIT – III

SYLLABUS

Overview of Programming: Structure of a Python Program-Element of Python

Programming Language:

A **programming language** is a vocabulary and set of grammatical rules for instructing a **computer** or computing device to perform specific tasks. The term **programming language** usually refers to high-level**languages**, such as BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal.

Meaning of Programming Languages

Normal Language is a communication between two people. We are using English, Tamil, Hindi and so on using communication between two people.

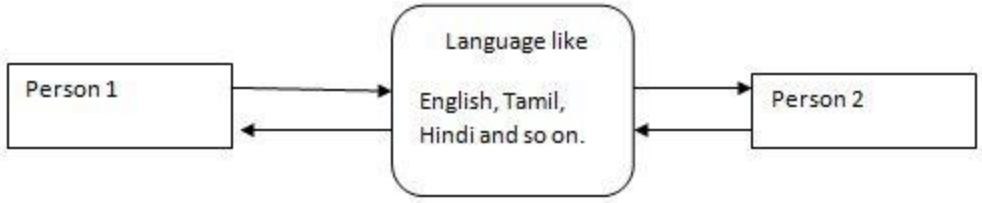


Figure 1: Intermediate for communicate

Programming Language is one kind of language. It is intermediate between human and system. Programming is one type of Language. There are different typs of programming Languages we are using. Each programming language has some different syntax. Syntax is a some set of rules and regulations.

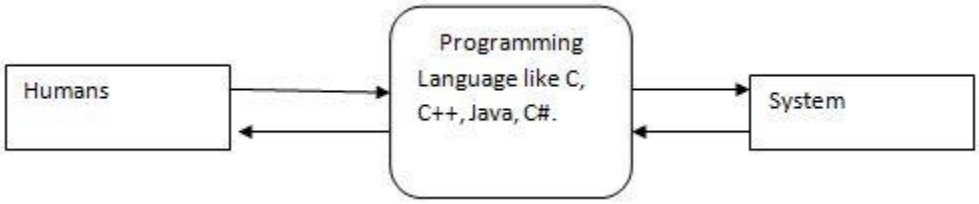


Figure 2: Intermediate for communicate with System

Important of Programming Languages

Programming language is the heart of software. Without programming we cannot make many applications and software. Programming Language is a key factor of software as well as embedded systems. Without programming language we cannot communicate with machines or systems. Systems only know machine code. Machine codes mean some set of series of numbers. Machine code we can call bits.

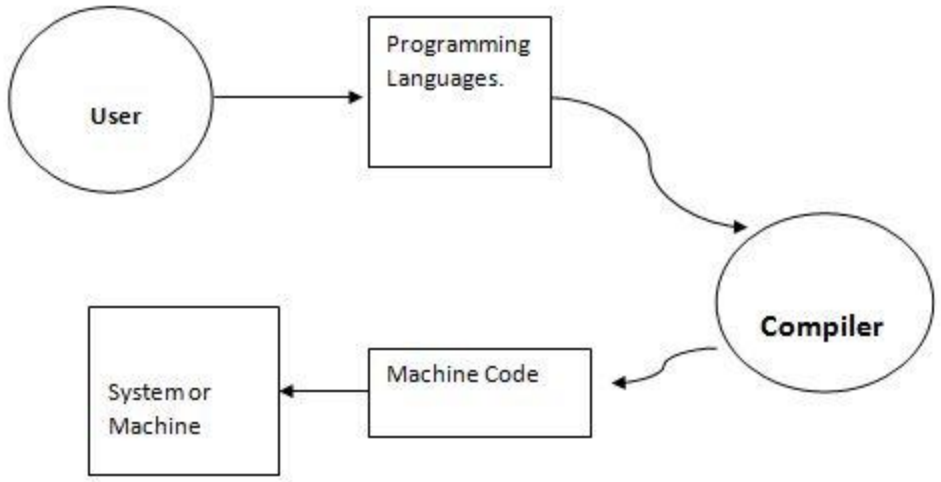
Humans only know high level languages but machines do not know high level languages. Humans and machine could not communicate directly. We need one intermediate because humans

could not understand machine languages like machines could not understand high level languages.

Compiler

Compiler is an intermediary to understand each language. Compiler converts High level languages to low level languages as well as low level languages convert to high level languages. Each language needs at least one compiler. Without compiler we could not understand low level language.

Input



Output

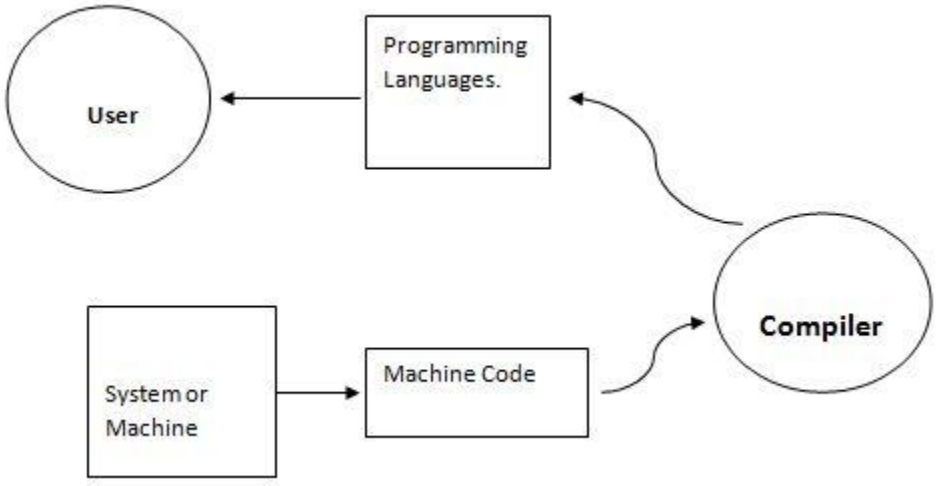


Figure 3: Flow of Programming Languages

Above diagram explains about flow of programming languages. Write code flow for any one of the programming languages. After writing programming we need to compile. Compiler should check syntax of programming language at the same time converting high level to low or machine level. If we

have any syntax errors do not convert machine language, instead of converting to inform us regarding error.

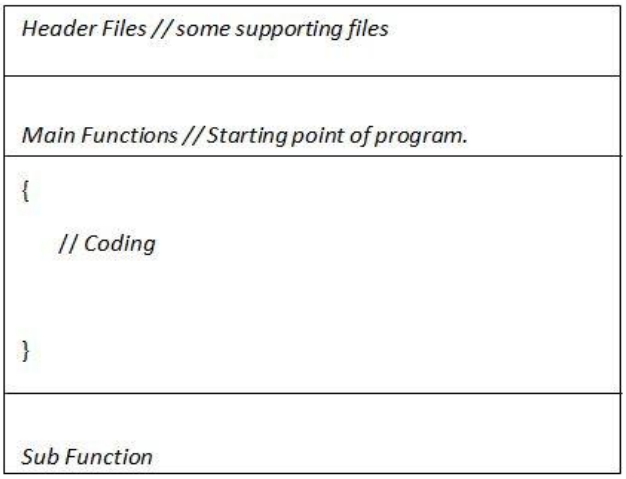
Figure 4: Compiler task

above diagram explains about conversion way of high level language to low level languages. Same way again low level to high level.

Structure of Programming Languages

Each programming Language has separate structure but a little bit changes in each programming change. Syntax wise we only have changes of each programming language otherwise it's the same structure.

Structure



Header file is some supporting files. It is located at the top of program. Header file is the head of program. We call header file a different name in different languages. Like bellow.

- Header File -> C Language
- Header File -> C++ Language
- Package -> Java Language
- Namespace -> C# Language

Main Function is important part of programming languages. Main function is like our body, each and every function happens in main function section. Main function is starting point of programming languages. Sub function is optional one. If need it we can use, otherwise leave it.

All programming language is syntax wise different apart from others, these are same. For example if need to print one line using any program using below.

printf("Welcome To C# Corner")	C Language
cout<<"Welcome To C# Corner";	C++ Language
System.out.print("Welcome To C# Corner");	Java Language
Console.WritLine("Welcome To C# Corner");	C# language

Types of Programming Languages

There are different types of programming languages available. We can see below.

- C
- C++
- Java
- C#
- Python
- Ruby

These are mainly using programming languages in current trends. We can use whichever language we feel is better. C, C++, Java and C# are having different syntax only but concept wise all are same.

Introduction

Python is a widely used general-purpose, high level **programming** language. It was initially designed by Guido van Rossum in 1991 and developed by **Python** Software Foundation.

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

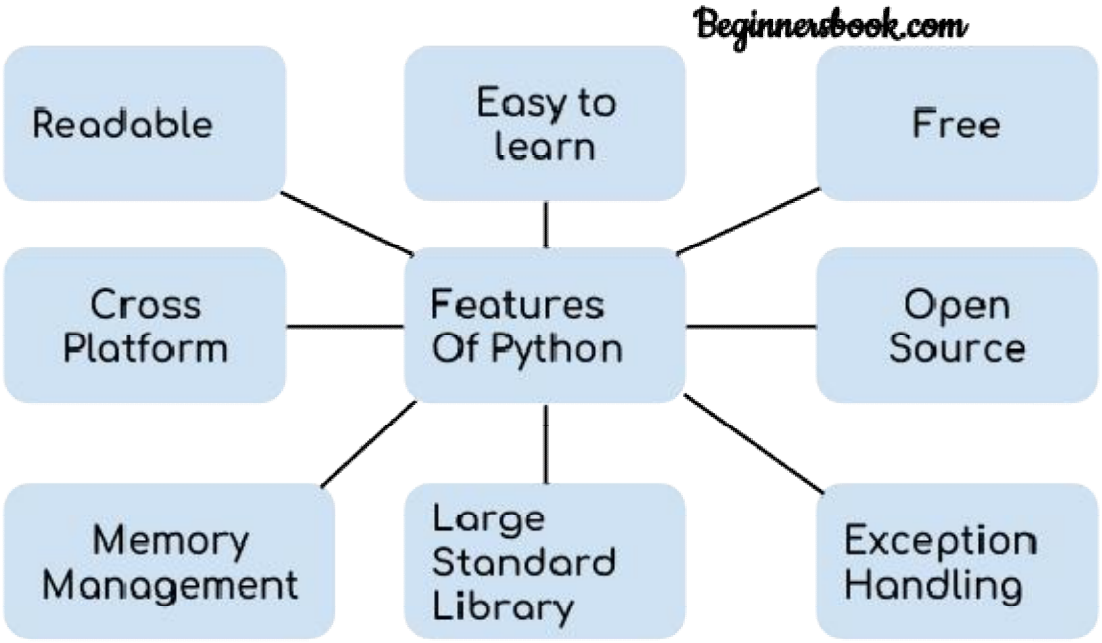
What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Features of Python programming language



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

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

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

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

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

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

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

8. **Advanced features:** Supports generators and list comprehensions. We will cover these features later.

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

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum**at CWI in Netherland.
- In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages:
    - ABC language.
    - Modula-3

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.

A list of python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018

Python Applications

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifying applications areas where python can be applied.

1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

2) Desktop GUI Applications

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

3) Software Development

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

4) Scientific and Numeric

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications

Python is used to build Business applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application

We can use Python to develop console based applications. For example: **IPython**.

7) Audio or Video based Applications

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications

To create CAD application Fandango is a real application which provides full features of CAD.

9) Enterprise Applications

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

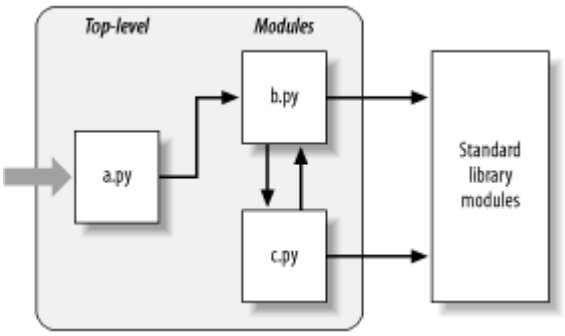
10) Applications for Images

Using Python several applications can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python

Programming Structure of python

In General Python Program Consists of so many text files, which contains python statements. Program is designed as single main, high file with one or more supplement files.



The structure Python Program consists of three files such as: a.py, b.py and c.py. The file model a.py is chosen for high level file. It is known as a simple text file of statements. And it can be executed from bottom to top when it is launched. Files b.py and c.py are modules. They are calculated as better text files of statements as well. But they are generally not started directly. Identically this attributes define Programming structure of Python.

Introduction:

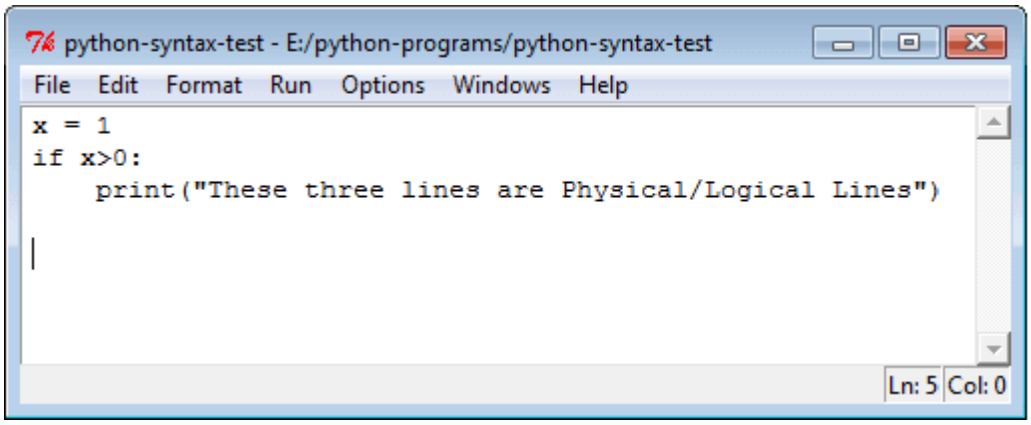
A Python program is read by a parser. Python was designed to be a highly readable language. The syntax of the Python programming language is the set of rules which defines how a Python program will be written.

Python Line Structure:

A Python program is divided into a number of logical lines and every logical line is terminated by the token NEWLINE. A logical line is created from one or more physical lines.

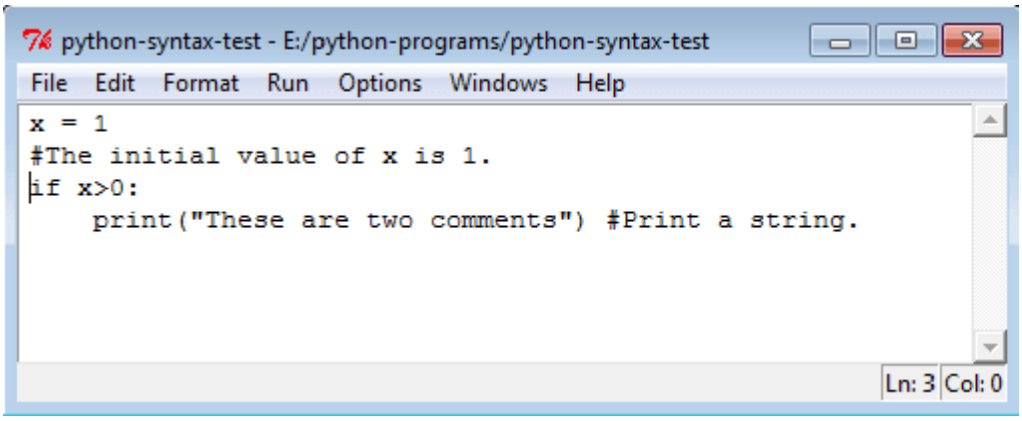
A line contains only spaces, tabs, formfeeds possibly a comment, is known as a blank line, and Python interpreter ignores it.

A physical line is a sequence of characters terminated by an end-of-line sequence (in windows it is called CR LF or return followed by a linefeed and in Unix, it is called LF or linefeed). See the following example.



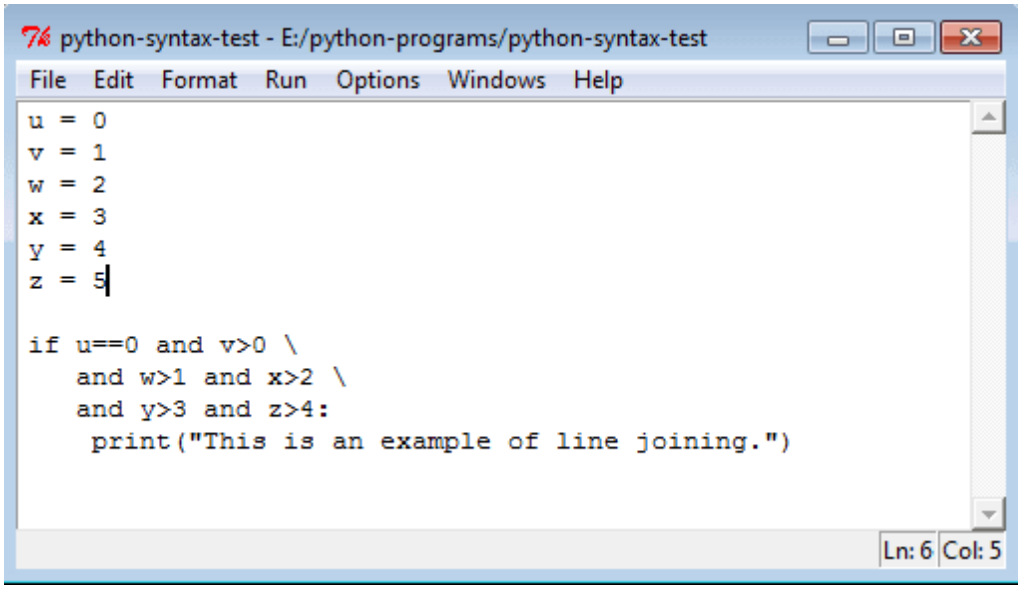
Comments in Python:

A comment begins with a hash character (#) which is not a part of the string literal and ends at the end of the physical line. All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them. See the following example. It should be noted that Python has no multi-lines or block comments facility.



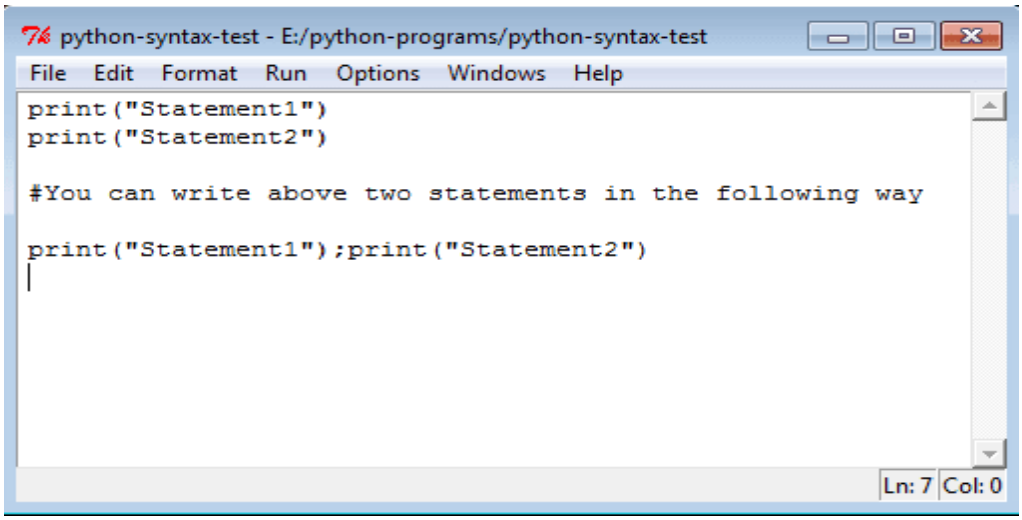
Joining two lines:

When you want to write a long code in a single line you can break the logical line in two or more physical lines using backslash character (\). Therefore when a physical line ends with a backslash characters (\) and not a part of a string literal or comment then it can join another physical line. See the following example.



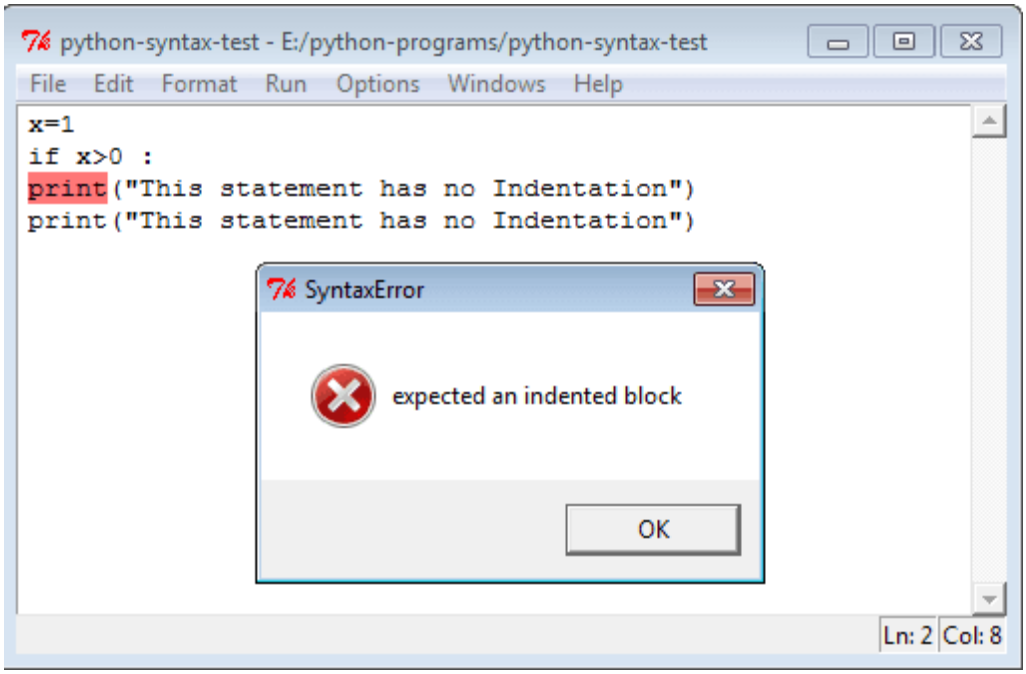
Multiple Statements on a Single Line:

You can write two separate statements into a single line using a semicolon (;) character between two lines.

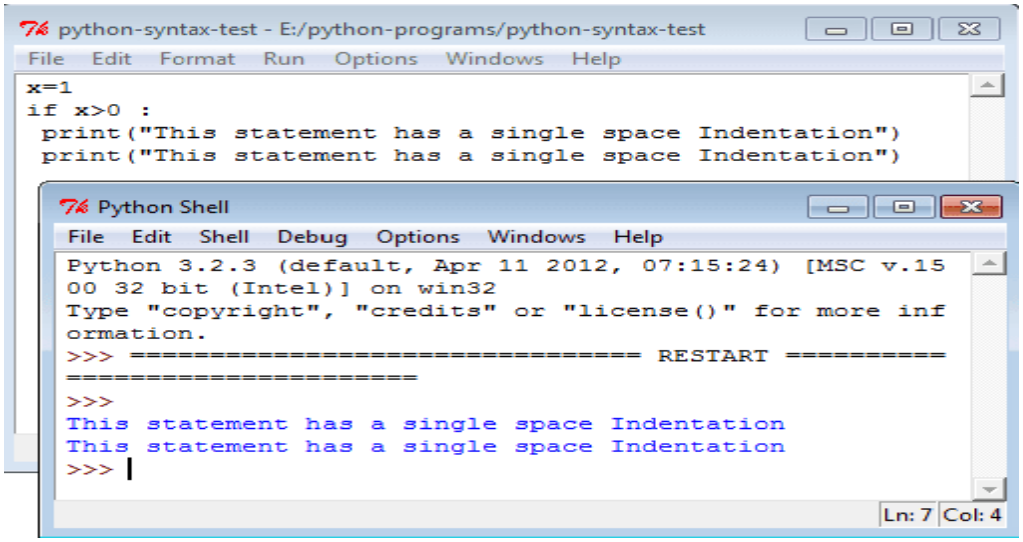


Indentation:

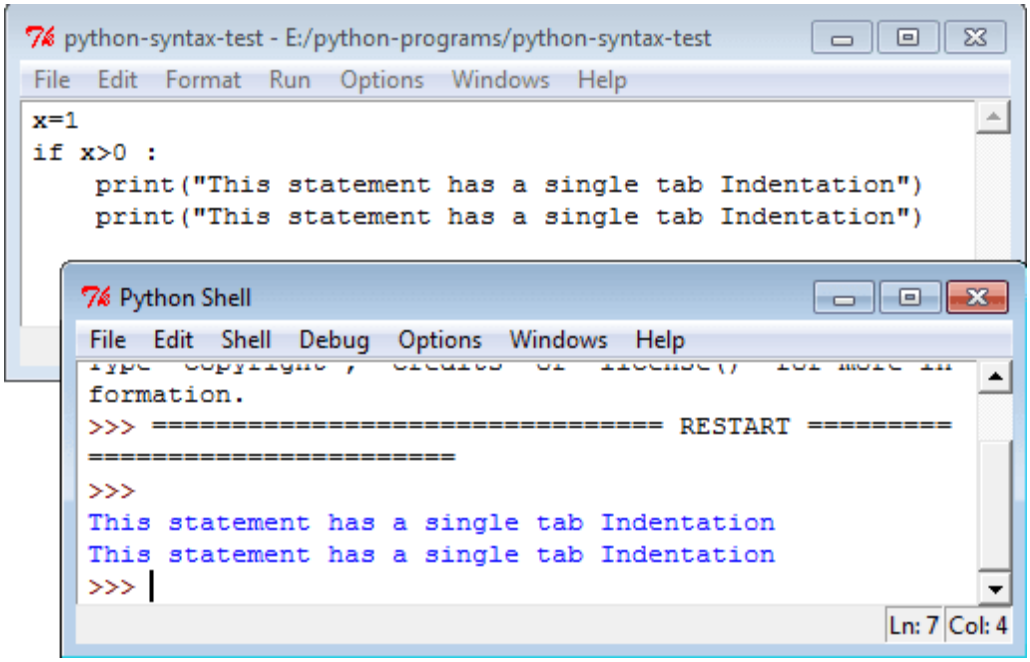
Python uses whitespace (spaces and tabs) to define program blocks whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control. The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount. In the following program, the block statements have no indentation.



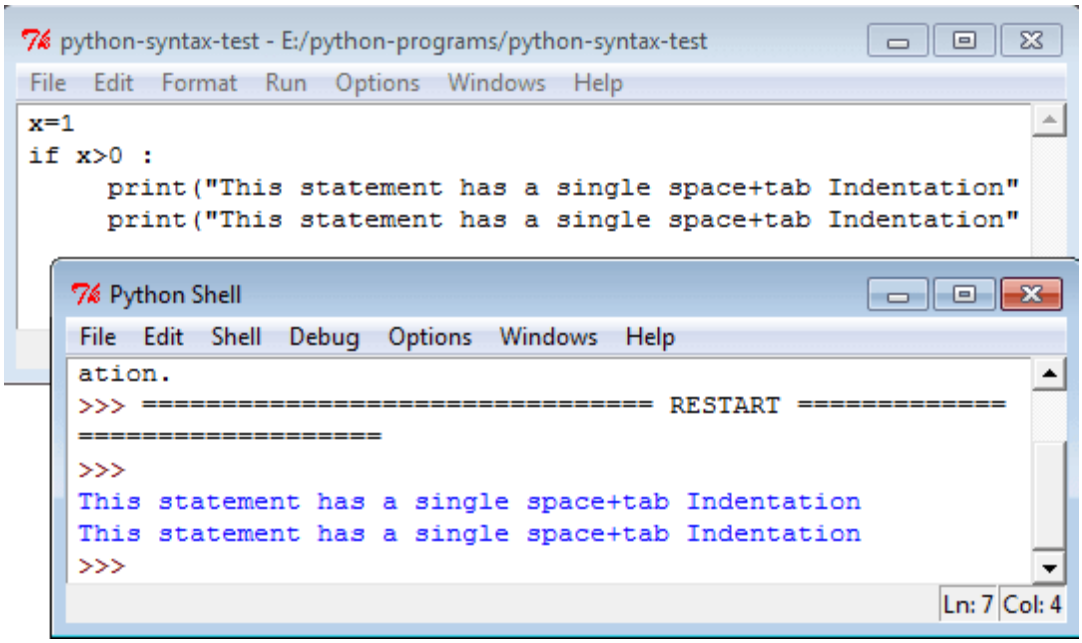
This is a program with single space indentation.



This is a program with single tab indentation.



Here is another program with an indentation of a single space + a single tab.



Python Coding Style:

- Use 4 spaces per indentation and no tabs.
- Do not mix tabs and spaces. Tabs create confusion and it is recommended to use only spaces.
- Maximum line length: 79 characters which help users with a small display.
- Use blank lines to separate top-level function and class definitions and single blank line to separate methods definitions inside a class and larger blocks of code inside functions.
- When possible, put inline comments (should be complete sentences).
- Use spaces around expressions and statements.

First Python Program

In this Section, we will discuss the basic syntax of python by using which, we will run a simple program to print hello world on the console.

Python provides us the two ways to run a program:

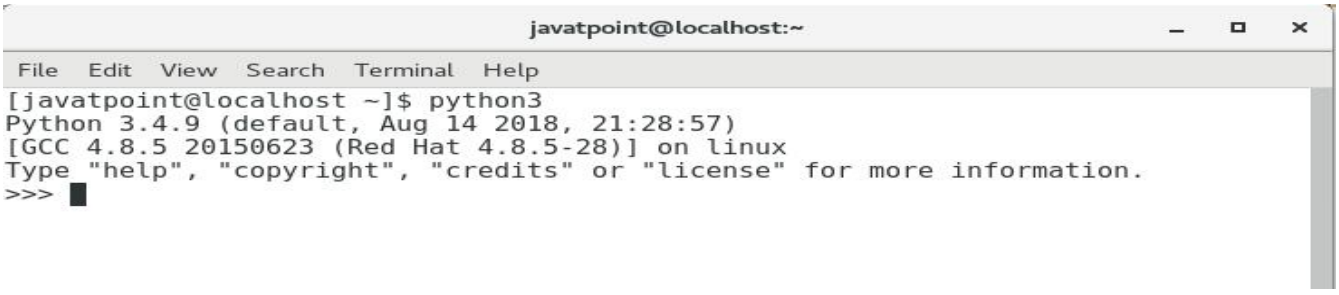
- Using Interactive interpreter prompt
- Using a script file

Interactive interpreter prompt

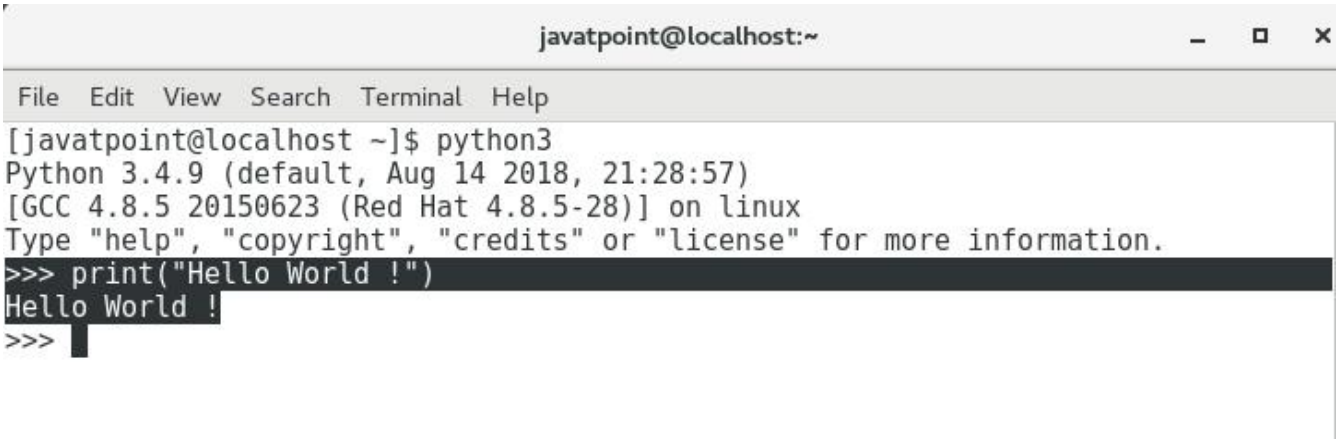
Python provides us the feature to execute the python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our python program.

To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case if you have python2 and python3 both installed on your system).

It will open the following prompt where we can execute the python statement and check their impact on the console.



Let's run a python statement to print the traditional hello world on the console. Python3 provides print() function to print some message on the console. We can pass the message as a string into this function. Consider the following image.



Here, we get the message **"Hello World !"** printed on the console.

Using a script file

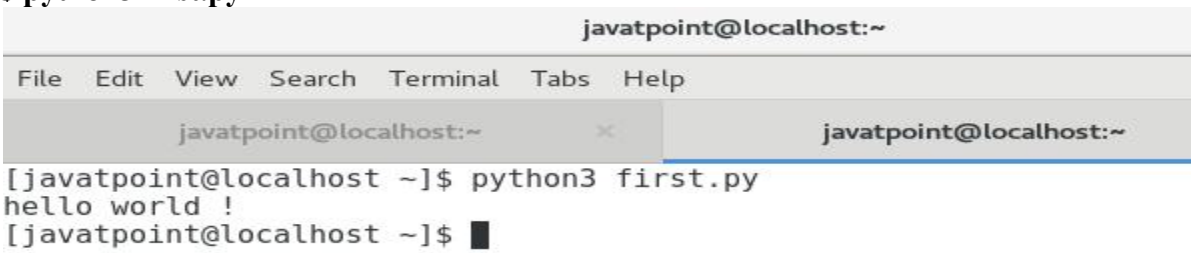
Interpreter prompt is good to run the individual statements of the code. However, we can not write the code every-time on the terminal.

We need to write our code into a file which can be executed later. For this purpose, open an editor like notepad, create a file named first.py (python used .py extension) and write the following code in it.

```
Print ("hello world"); #here, we have used print() function to print the message on the console.
```

To run this file named as first.py, we need to run the following command on the terminal.

$ python3 first.py



Hence, we get our output as the message **Hello World!** is printed on the console.

Python Variables

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive for example my name, and MyName is not the same.
- Examples of valid identifiers: a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

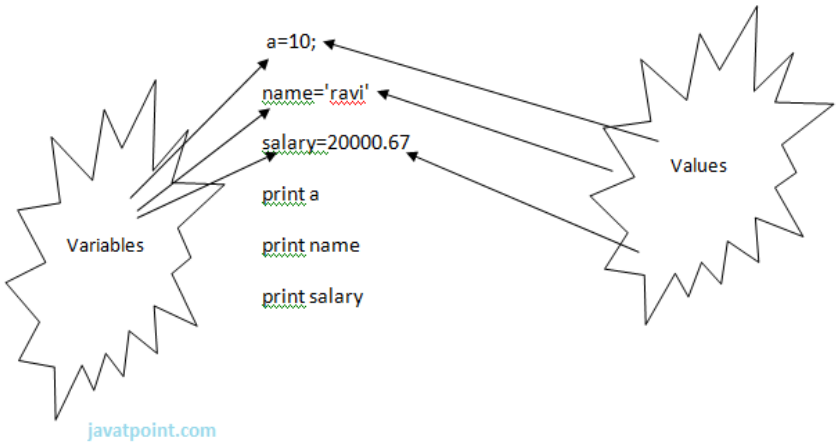
Declaring Variable and Assigning Values

Python does not bind us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Eg:



Output:

```
>>>
10
ravi
20000.67
>>>
```

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignments.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print x
print y
print z
```

Output:

```
>>>
50
50
```

```
4. 50
5. >>>
```

2. Assigning multiple values to multiple variables:

Eg:

```
1. a,b,c=5,10,15
2. print a
3. print b
4. print c
```

Output:

```
1. >>>
2. 5
3. 10
4. 15
5. >>>
```

The values will be assigned in the order in which variables appears.

Basic Fundamentals:

This section contains the basic fundamentals of Python like:

i)Tokens and their types.

ii) Comments

a)Tokens:

- Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
- Token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.

Tuples:

- Tuple is another form of collection where different type of data can be stored.
- It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
- Tuples are enclosed in parenthesis and cannot be changed.

Eg:

```
>>> tuple=('rahul',100,60.4,'deepak')
>>> tuple1=('sanjay',10)
>>> tuple
('rahul', 100, 60.4, 'deepak')
>>> tuple[2:]
(60.4, 'deepak')
>>> tuple1[0]
'sanjay'
>>> tuple+tuple1
('rahul', 100, 60.4, 'deepak', 'sanjay', 10)
>>>
```

Dictionary:

- Dictionary is a collection which works on a key-value pair.
- It works like an associated array where no two keys can be same.
- Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

Eg:

```
>>> dictionary={'name':'charlie','id':100,'dept':'it'}
>>> dictionary
{'dept': 'it', 'name': 'charlie', 'id': 100}
>>> dictionary.keys()
['dept', 'name', 'id']
>>> dictionary.values()
['it', 'charlie', 100]
>>>
```

Python Data Types

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
1. A=10
2. b="Hi Python"
3. c = 10.5
4. print(type(a));
5. print(type(b));
6. print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

```
1. a = 3 , b = 5  #a and b are number objects
```

Python supports 4 types of numeric data.

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

String

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

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
1. str1 = 'hello javatpoint' #string str1
2. str2 = ' how are you' #string str2
3. print (str1[0:2]) #printing first two character using slice operator
4. print (str1[4]) #printing 4th character of the string
5. print (str1*2) #printing the string twice
6. print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hellojavatpointhellojavatpoint
hellojavatpoint how are you
```

List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
1. l = [1, "hi", "python", 2]
```

```
2. print (l[3:]);
3. print (l[0:2]);
4. print (l);
5. print (l + l);
6. print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
1. t  = ("hi", "python", 2)
2. print (t[1:]);
3. print (t[0:1]);
4. print (t);
5. print (t + t);
6. print (t * 3);
7. print (type(t))
8. t[2] = "hi";
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<type 'tuple'>
Traceback (most recent call last):
  File "main.py", line 8, in <module>
t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
1. d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
2. print("1st name is "+d[1]);
3. print("2nd name is "+ d[4]);
4. print (d);
5. print (d.keys());
6. print (d.values());
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
[1, 2, 3, 4]
['Jimmy', 'Alex', 'john', 'mike']
```

Python List/Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list

copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

1) append() Method

The append() method appends an element to the end of the list.

Syntax

list.append(*elmnt*)

Parameter Values

Parameter	Description
elmnt	Required. An element of any type (string, number, object etc.)

Example:1

Add an element to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
```

Example:2

Add a list to a list:

```
a = ["apple", "banana", "cherry"]
b = ["Ford", "BMW", "Volvo"]
a.append(b)
```

2) clear() Method

The clear() method removes all the elements from a list. There is no parameter for clear() method.

Syntax

list.clear()

Example

Remove all elements from the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']

fruits.clear()
```

3) copy() Method

The copy() method returns a copy of the specified list.There is no parameter for copy() method.

Syntax

list.copy()

Example

Copy the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']

x = fruits.copy()
```

4) count() Method

The count() method returns the number of elements with the specified value.

Syntax

list.count(*value*)

Parameter Values

Parameter	Description
value	Required. Any type (string, number, list, tuple, etc.). The value to search for.

Example:1

Return the number of times the value "cherry" appears int the fruits list:

```
fruits = ['apple', 'banana', 'cherry']

x = fruits.count("cherry")
```

Example:2

Return the number of times the value 9 appears in the list:

```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = points.count(9)
```

5) extend() Method

The extend() method adds the specified list elements (or any iterable) to the end of the current list.

Syntax

list.extend(*iterable*)

Parameter Values

Parameter	Description
iterable	Required. Any iterable (list, set, tuple, etc.)

Example:1

Add the elements of cars to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)
```

Example:2

Add a tuple to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
points = (1, 4, 5, 9)
fruits.extend(points)
```

6) index() Method

The index() method returns the position at the first occurrence of the specified value.

Syntax

list.index(*elmnt*)

Parameter Values

Parameter	Description
elmnt	Required. Any type (string, number, list, etc.). The element to search for

Example:1

What is the position of the value "cherry":

```
fruits = ['apple', 'banana', 'cherry']

x = fruits.index("cherry")
```

Example:2

What is the position of the value 32:

```
fruits = [4, 55, 64, 32, 16, 32]

x = fruits.index(32)
```

Note: The index() method only returns the *first* occurrence of the value.

7) insert() Method

The insert() method inserts the specified value at the specified position.

Syntax

list.insert(*pos*, *elmnt*)

Parameter Values

Parameter	Description

pos	Required. A number specifying in which position to insert the value
elmnt	Required. An element of any type (string, number, object etc.)

Example

Insert the value "orange" as the second element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")
```

8) pop() Method

The pop() method removes the element at the specified position.

Syntax

list.pop(*pos*)

Parameter Values

Parameter	Description
pos	Optional. A number specifying the position of the element you want to remove, default value is -1, which returns the last item

Example:1

Remove the second element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.pop(1)
```

Example:2

Return the removed element:

```
fruits = ['apple', 'banana', 'cherry']

x = fruits.pop(1)
```

Note: The pop() method returns removed value.

9) remove() Method

The remove() method removes the first occurrence of the element with the specified value.

Syntax

list.remove(*elmnt*)

Parameter Values

Parameter	Description
elmnt	Required. Any type (string, number, list etc.) The element you want to remove

Example

Remove the "banana" element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")
```

10) reverse() Method

The reverse() method reverses the sorting order of the elements. There is no parameter for reverse() method.

Syntax

list.reverse()

Example:1

Reverse the order of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.reverse()
```

11) sort() Method

The sort() method sorts the list ascending by default.You can also make a function to decide the sorting criteria(s).

Syntax

list.sort(reverse=True|False, key=myFunc)

Parameter Values

Parameter	Description
reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
key	Optional. A function to specify the sorting criteria(s)

Example:1

Sort the list alphabetically:

```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort()
```

Example:2

Sort the list descending:

```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort(reverse=True)
```

Example:3

Sort the list by the length of the values:

```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)
```

```
cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)
```

Example:4

Sort a list of dictionaries based on the "year" value of the dictionaries:

```
# A function that returns the 'year' value:
def myFunc(e):
  return e['year']

cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]

cars.sort(key=myFunc)
```

Example:5

Sort the list by the length of the values *and* reversed:

```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(reverse=True, key=myFunc)
```

Common Dictionary Operations in Python

A8 dictionary constant consists of a series of key-value pairs enclosed by curlybraces { }

Below is a list of common dictionary operations:

i) create an empty dictionary

```
x = {}
```

ii) create a three items dictionary

```
x = {"one":1, "two":2, "three":3}
```

iii) access an element

```
x['two']
```

iv) get a list of all the keys
x.keys()

v) get a list of all the values
x.values()

add an entry
x["four"]=4

vi) change an entry
x["one"] = "uno"

vii) delete an entry
del x["four"]

viii) make a copy
y = x.copy()

remove all items
x.clear()

ix) number of items
z = len(x)

x) test if has key
z = x.has_key("one")

xi) looping over keys
for item in x.keys(): print item

xii) looping over values
for item in x.values(): print item

xiii) using the if statement to get the values
if "one" in x:
print x['one']

if "two" not in x:
print "Two not found"

if "three" in x:
del x['three']

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

1) clear() Method

The clear() method removes all the elements from a dictionary. There is no parameter for clear() method.

Syntax

dictionary.clear()

Example

Remove all elements from the car list:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

car.clear()

print(car)
```

2) copy() Method

The copy() method returns a copy of the specified dictionary. There is no parameter for copy() method.

Syntax

dictionary.copy()

Example

Copy the car dictionary:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
```

```
}

x = car.copy()

print(x)
```

3) fromkeys() Method

The fromkeys() method returns a dictionary with the specified keys and values.

Syntax

dict.fromkeys(*keys, value*)

Parameter Values

Parameter	Description
keys	Required. An iterable specifying the keys of the new dictionary
value	Optional. The value for all keys. Default value is None

Example:1

Create a dictionary with 3 keys, all with the value 0:

```
x = ('key1', 'key2', 'key3')
y = 0

thisdict = dict.fromkeys(x, y)

print(thisdict)
```

Example:2

Same example as above, but without specifying the value:

```
x = ('key1', 'key2', 'key3')

thisdict = dict.fromkeys(x)
```

```
print(thisdict)
```

4) get() Method

The get() method returns the value of the item with the specified key.

Syntax

dictionary.get(*keyname, value*)

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to return the value from
value	Optional. A value to return if the specified key does not exist. Default value None

Example:1

Get the value of the "model" item:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.get("model")
print(x)
```

Example:2

Try to return the value of an item that do not exist:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
x = car.get("price", 15000)
print(x)
```

5) items() Method

The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.

The view object will reflect any changes done to the dictionary, see example below.

Syntax

dictionary.items()

Example:1

Return the dictionary's key-value pairs:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.items()
print(x)
```

Example:2

When an item in the dictionary changes value, the view object also gets updated:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.items()
car["year"] = 2018
print(x)
```

6) keys() Method

The keys() method returns a view object. The view object contains the keys of the dictionary, as a list.

The view object will reflect any changes done to the dictionary, see example below.

Syntax

dictionary.keys()

Example:1

Return the keys:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.keys()
print(x)
```

Example:2

When an item is added in the dictionary, the view object also gets updated:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.keys()
car["color"] = "white"
print(x)
```

7) pop() Method

The pop() method removes the specified item from the dictionary. The value of the removed item is the return value of the pop() method, see example below.

Syntax

dictionary.pop(*keyname, defaultvalue*)

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to remove

defaultvalue	Optional. A value to return if the specified key do not exist. If this parameter is not specified, and the no item with the specified key is found, an error is raised

Example:1

Remove "model" from the dictionary:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.pop("model")
print(car)
```

Example:2

The value of the removed item is the return value of the pop() method:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.pop("model")
print(x)
```

8) popitem() Method

The popitem() method removes the item that was last inserted into the dictionary. In versions before 3.7, the popitem() method removes a random item.

The removed item is the return value of the popitem() method, as a tuple, see example below.

Syntax

dictionary.popitem(*keyname, defaultvalue*)

Example:1

Remove the last item from the dictionary:

```
car = {
```

```
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.popitem()
print(car)
```

Example:2

The removed item is the return value of the pop() method:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.popitem()
print(x)
```

9) setdefault() Method

The setdefault() method returns the value of the item with the specified key.If the key does not exist, insert the key, with the specified value, see example below

Syntax

dictionary.setdefault(*keyname, value*)

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to return the value from
value	Optional. If the key exist, this parameter has no effect. If the key does not exist, this value becomes the key's value Default value None

Example:1

Get the value of the "model" item:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.setdefault("model", "Bronco")
print(x)
```

Example:2

Get the value of the "color" item, if the "color" item does not exist, insert "color" with the value "white":

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.setdefault("color", "white")
print(x)
```

10) update() Method

The update() method inserts the specified items to the dictionary. The specified items can be a dictionary, or an iterable object.

Syntax

dictionary.update(*iterable*)

Parameter Values

Parameter	Description
iterable	A dictionary or an iterable object with key value pairs, that will be inserted to the dictionary

Example:1

Insert an item to the dictionary:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.update({"color": "White"})
print(car)
```

11) values() Method

The values() method returns a view object. The view object contains the values of the dictionary, as a list.The view object will reflect any changes done to the dictionary, see example below.

Syntax

dictionary.values()

Example:1

Return the values:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.values()
print(x)
```

Example:2

When a value is changed in the dictionary, the view object also gets updated:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.values()
car["year"] = 2018
print(x)
```

Python Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

count() Method

The count() method returns the number of times a specified value appears in the tuple.

Syntax

tuple.count(*value*)

Parameter Values

Parameter	Description
value	Required. The item to search for

Example

Return the number of times the value 5 appears in the tuple:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

index() Method

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

Syntax

tuple.index(*value*)
Parameter Values

Parameter	Description
value	Required. The item to search for

Example

Search for the first occurrence of the value 8, and return its position:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

Python Sets

Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Example

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

Example

Add an item to a set, using the add() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Example

Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

Example

Remove "banana" by using the remove() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Note: If the item to remove does not exist, remove() will raise an error.

Example

Remove "banana" by using the discard() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Note: If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use the pop(), method to remove an item, but this method will remove the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

Example

Remove the last item by using the pop() method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

Note: Sets are *unordered*, so when using the pop() method, you will not know which item that gets removed.

Example

The clear() method empties the set:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Example

The del keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

Example

The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
```

```
print(set3)
```

Example

The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

Note: Both union() and update() will exclude any duplicate items.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page.

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Possible Questions:

2 Mark Questions

1. What isPython?
2. Define Namespace & mention itstype.
3. How does a computer run a pythonprogram?
4. What is Pythonlist?
5. Definearray.
6. List the features ofpython.
7. What is necessary to execute a Pythonprogram?
8. What is a statement in a Pythonprogram?

6 Mark Questions

1. What is Python list? Explain the basic list operations with suitableexamples.
2. Briefly discuss about the fundamental ofPython.
3. Discuss the structure of python program with suitable exampleprogram.
4. What is Python list? Explain pythonelements.
5. Discuss the methods to manipulate the arrays inpython
6. Write a Python program to multiply twomatrices.
7. Explain the following i) Tuple ii) Dictionary iii) List

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

III B.Sc(CS) (BATCH 2017-2020) V SEMESTER

PROGRAMMING IN PYTHON (17CSU504B)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT III

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	Python is said to be easily	readable language	writable language	bug-able language	script-able language	readable language
2	Extensible programming language that can be extended through classes and programming interfaces is	Python	Perl	PHP	Ada	Python
3	Python was released publicly in	1941	1971	1981	1991	1991
4	What is the output when following code is executed ? print r"\nhello" The output is	a new line and hello	\nhello	the letter r and then hello	Error	\nhello
5	What is the output of the following code ? example = "snow world" example[3] = 's' print example	snow	snow world	Error	snos world	Error
6	Is Python case sensitive when dealing with identifiers?	yes	no	machine dependent	none	yes
7	What is the maximum possible length of an identifier?	31 characters	63 characters	79 characters	none of the mentioned	none of the mentioned
8	Which of the following is not a keyword?	eval	assert	nonlocal	pass	eval
9	All keywords in Python are in	lower case	UPPER CASE	Capitalized	None of the mentioned	None of the mentioned

10	Which of the following is true for variable names in Python?	unlimited length	all private members must have leading and trailing underscores	underscore and ampersand are the only two special characters allowed	none of the mentioned	unlimited length
11	Which of the following is an invalid statement?	abc = 1,000,000	a b c = 1000 2000 3000	a,b,c = 1000, 2000, 3000	a_b_c = 1,000,000	a b c = 1000 2000 3000
12	Which of the following cannot be a variable?	__init__	in	it	on	in
13	What is the output of print 0.1 + 0.2 == 0.3?	TRUE	FALSE	machine dependent	Error	FALSE
14	Which of the following is not a complex number?	k = 2 + 3j	k = complex(2, 3)	k = 2 + 3l	k = 2 + 3J	k = 2 + 3l
15	Which of the following data types is not supported in python?	number	list	string	slice	slice
16	Which of these is not a core data type?	Lists	Dictionary	Tuples	Class	Class
17	What data type is the object below ? L = [1, 23, 'hello', 1]	Lists	Dictionary	Tuples	Array	Lists
18	Which of the following function convert a string to a float in python?	int(x [,base])	long(x [,base])	float(x)	str(x)	float(x)
19	The sequence \n does what?	Makes a link	Prints a backslash followed by a n	Adds 5 spaces	Starts a new line	Starts a new line
20	Python is a _____ programming language	higher-level	lowe level	mid level	first level	higher-level

21	An _____translates a source file into machine language as the program executes	complier	interpreter	editor	none	interpreter
22	A _____ translates a source file into an executable file	compiler	interpreter	editor	none	compiler
23	Messages can be printed in the output window by using Python's _____function.	scan	edit	print	none	print
24	Python is a _____ language	case insensitive	case sensitive	character	none	case sensitive
25	The _____ function enables a Python program to display textual information to the user	scan	edit	print	input	print
26	Programs may use the _____ function to obtain information from the user.	scan	edit	print	input	input
27	Python does not permit _____ to be used when expressing numeric literals	commas	quote	questionmark	arrow	commas
28	The statement a = b copies the value stored in _____	variable a into variable b	variable b into variable a	error	none	variable b into variable a
29	The _____ function accepts an optional prompt string.	scan	edit	print	input	input
30	The _____ function can be used to convert a string representing a numeric expression into its evaluated numeric value.	scan	eval	print	input	eval
31	Python was created by__	James Gosling	Bill Gates	Steve Jobs	Guido van Rossum	Guido van Rossum

32	To start Python from the command prompt, use the command _____.	execute python	run proram	pyhton	go pyhton	pyhton
33	To run python script file named t.py, use the command _____.	execute python t.py	run python t.py	python t.py	go python t.py	python t.py
34	A Python line comment begins with _____.	//	/*	#	@	#
35	A Python paragraph comment uses the style _____.	// comments //	/* comments */	'' comments '''	/# comments #/	'' comments '''
36	In Python, a syntax error is detected by the _____ at _____.	compiler/at compile time	interpreter/at runtime	compiler/at runtime	interpreter/at compile time	interpreter/at runtime
37	_____ is the code in natural language mixed with some program code.	Python program	A Python statement	Pseudocode	A flowchart diagram	Pseudocode
38	2 ** 3 evaluates to _____.	9	8	9.1	8.1	8
39	The _____ function immediately terminates the program.	sys.terminate()	sys.halt()	sys.exit()	sys.stop()	sys.exit()
40	The following is NOT an example of a data type.	int	public	double	void	public

UNIT-IV

SYLLABUS

Introduction to Python: Python Interpreter-Using Python as calculator-Python shell- Indentation. Atoms-Identifiers and keywords-Literals-Strings-Operators(Arithmetic operator, Relational operator, Logical or Boolean operator, Assignment, Operator, Ternary operator, Bit wise operator, Increment or Decrement operator).

Interpreter:

An **interpreter** is a program that reads and executes code. This includes source code, pre-compiled code, and scripts. Common **interpreters** include Perl, **Python**, and Ruby **interpreters**, which execute Perl, **Python**, and Ruby code respectively.

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby uses interpreters.	Programming language like C, C++ use compilers.

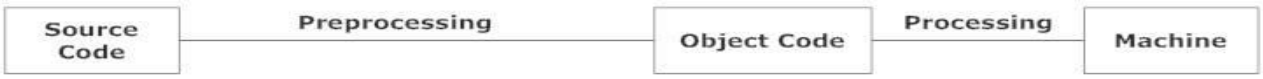


Figure: Compiler

Figure: Interpreter

Python Interpreter & its Environment (Source Code Encoding)

The default encoding for a Python source file is UTF-8. This is a *Unicode Standard variable-width character encoding;* it can encode 1,112,064 valid code points in Unicode using up to four 8-bit bytes. Using this encoding, we can use characters of most languages – we can use these in string literals, **comments**, and **identifiers**. Since the standard library makes use of ASCII characters only, you must declare the use of this encoding to your editor. This is to ensure that all such characters display without a problem. The font should be such that supports all characters in the file. We add this comment as the first line of the file we want to use it in-

```
# -*- coding: encoding -*-
```

In this, encoding is a valid codec that Python supports. Similarly, when you want to use the Windows-1252 encoding, you can use this as the first line of code:

```
# -*- coding: cp1252 -*-
```

However, when you want to begin code with a UNIX shebang line, you can put the comment for encoding second-

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

How to Invoke the Python Interpreter?

On your machine, you can find your interpreter at an address like:
C:\Python36
Or it may reside on the location you selected at the time of installation. Add path using this command:

```
set path=%path%;C:\python36
```

Features of Python Interpreter

Python interpreter offers some pretty cool features:

- Interactive editing
- History substitution
- Code completion on systems with support for read line

In the first Python prompt, try pressing the following keys:

Ctrl+P

this tells you if your interpreter supports command-line editing. A beep indicates that it does support command-line editing. Otherwise, it will either perform a no-operation or echo ^*p* to indicate it isn't available.

Passing Arguments

When you pass a script name and additional arguments to the shell when invoking the Python interpreter, it turns these into a list of strings. Then, it assigns these to the variable *argv* in the sys module. The following command will give us a list of this-

```
import sys
```

Without a script or arguments, sys.argv[0] denotes an empty string. A script name of '-' means that it sets sys.argv[0] to '-', and with '-*c*', it is set to '-*c*'. A value of '-*m*' sets sys.argv[0] to the module's full name. The command/ module handles the options after '-*c*' or '-*m*'.

Python Collections Module

Interactive Mode

Python interpreter is in an interactive mode when it reads commands from a try. The primary prompt is the following:

```
>>>
```

When it shows this prompt, it means it prompts the developer for the next command. This is the REPL(A **read–eval–print loop** (**REPL**), also termed an **interactive top-level** or **language shell**, is a simple, interactive computer programming environment that takes single user inputs (i.e., single expressions), evaluates (executes) them, and returns the result to the user; a program written in a REPL environment is executed piecewise). Before it prints the first prompt, Python interpreter prints a welcome message that also states its version number and a copyright notice.

This is the secondary prompt:

```
…
```

This prompt denotes continuation lines.

```
$ python3.7
Python 3.7(default, Jul 162018, 04:38:07)
[GCC 4.8.2] on Windows
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You will find continuation lines when working with a multi-line construct:

```
>>>it_rains =True
>>>ifit_rains:
print("The produce will be good")
```

Output:

The produce will be good

You can also use the Python interpreter as a calculator:

```
1. >>>2*7
2. 14
3. >>>4/2
4. 2.0
```

How Does Python Interpreter Works?

Well, internally, four things happen in a REPL:

i. Lexing - The lexer breaks the line of code into tokens.

ii. Parsing - The parser uses these tokens to generate a structure, here, an Abstract Syntax Tree, to depict the relationship between these tokens.

iii. Compiling- The compiler turns this AST(Abstract Syntax Tree) into code object(s).

iv. Interpreting- The interpreter executes each code object.

What is python shell?

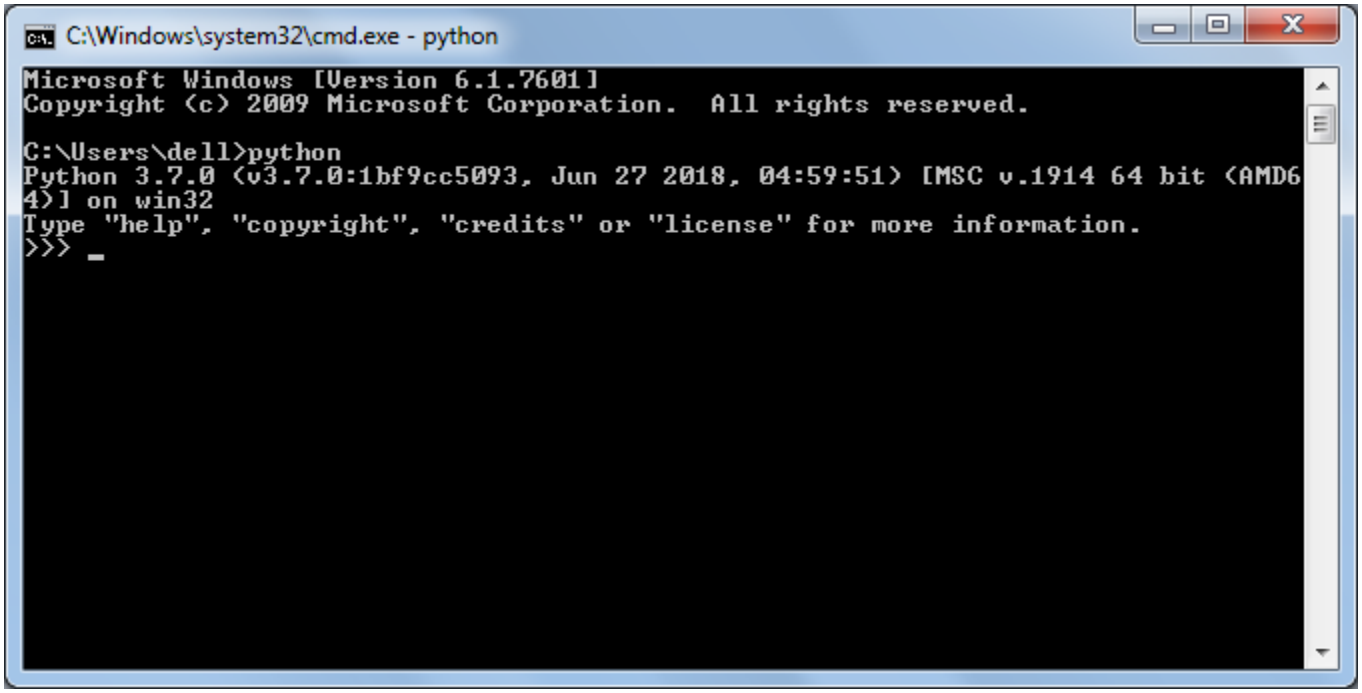
Python is an **interpreter** language. It means it executes the code line by line. **Python** provides a **Python Shell** (also known as **Python** Interactive **Shell**) which is used to execute a single **Python** command and get the result. **Python Shell** waits for the input command from the user.

Python - Shell (Interpreter)

Python is an interpreter language. It means it executes the code line by line. Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single Python command and get the result.

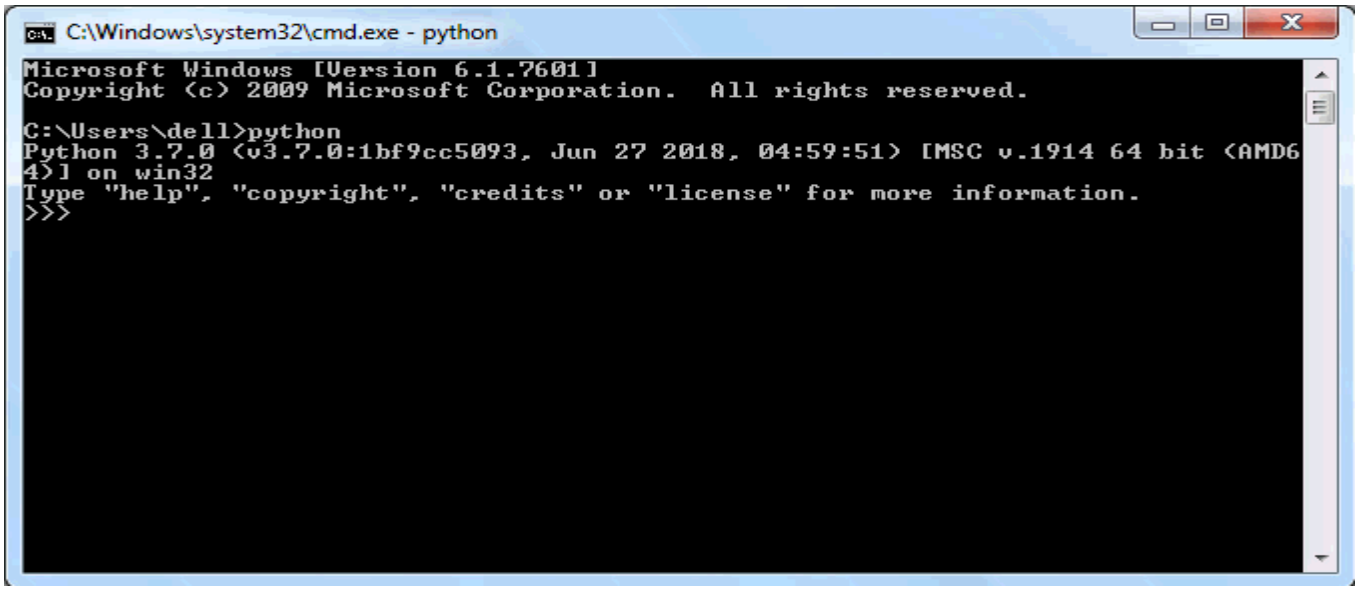
Python Shell waits for the input command from the user. As soon as the user enters the command, it executes it and displays the result.

To open the Python Shell on Windows, open the command prompt, write python and press **enter**.



Python Shell

As you can see, a Python Prompt comprising of three Greater Than symbols (>>>) appears. Now, you can enter a single statement and get the result. For example, enter a simple expression like 3 + 2, presses enter and it will display the result in the next line, as shown below.



Command Execution on Python Shell

Execute Python Script

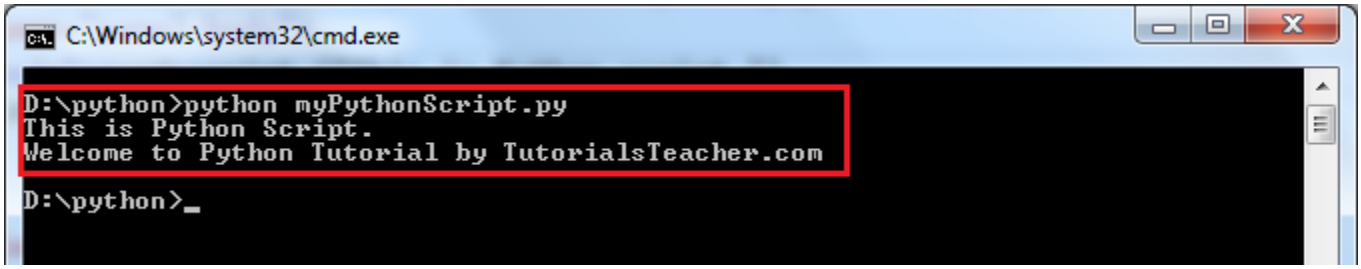
As you have seen above, Python Shell executes a single statement. To execute multiple statements, create a Python file with extension .py, and write Python scripts (multiple statements).

For example, enter the following statement in a text editor such as Notepad.

Example: myPythonScript.py

```
print ("This is Python Script.")
print ("Welcome to Python Tutorial by TutorialsTeacher.com")
```

Save it as myPythonScript.py, navigate command prompt to the folder where you have saved this file and execute the python myPythonScript.py command, as shown below.



Python Shell

Thus, you can execute Python expressions and commands using Python Shell.

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, `a = 1` is an assignment statement. `if` statement, `for` statement, `while` statement etc. are other kinds of statements which will be discussed later.

Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
1. a =1+2+3+ \
2. 4+5+6+ \
3. 7+8+9
```

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

```
1. a =(1+2+3+
2. 4+5+6+
```

```
3. 7+8+9)
```

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
1. colors =['red',
2. 'blue',
3. 'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
1. a =1; b =2; c =3
```

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
for i in range(1,11):
print(i)
ifi==5:
break
```

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

```
ifTrue:
print('Hello')
    a =5
```

and

```
ifTrue:print('Hello'); a =5
```

Both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError.

Atoms

Atoms are the most basic elements of expressions. The simplest atoms are identifiers or literals. Forms enclosed in reverse quotes or in parentheses, brackets or braces are also categorized syntactically as atoms.

Python Keywords

Well simply, python keywords are the words that are reserved. That means you can't use them as name of any entities like variables, classes and functions.

So you might be thinking what these keywords are for. They are for defining the syntax and structures of Python language.

You should know there are 33 keywords in Python programming language as of writing this tutorial. Although the number can vary in course of time. Also keywords in Python are case sensitive. So they are to be written as it is. Here is a list of all keywords in python programming.

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

If you look at all the keywords and try to figure out all at once, you will be overwhelmed. So for now just know these are the keywords. We will learn their uses respectively. You can get the list of python keywords through python shell help.

>>>help()

Welcome to Python 3.6's help utility!

help> keywords

Here is a list of the Python keywords. Enter any keyword to get more help.

False def if raise None del import return

True elif in try and else is while

as except lambda with assert finally nonlocal yield

breakfor not class from or continue global pass

help>

Below is a simple example showing usage of if-else in python program.

```
var = 1;
if(var==1):
print("odd")
else:
print("even")
```

Program, python understands the if-else block because of fixed keywords and syntax and then does the further processing.

Python Identifiers

Python Identifier is the name we give to identify a variable, function, class, module or other object. That means whenever we want to give an entity a name, that's called identifier.

Sometimes variable and identifier are often misunderstood as same but they are not. Well for clarity, let's see what is a variable?

Variable in Python

A variable, as the name indicates is something whose value is changeable over time. In fact a variable is a memory location where a value can be stored. Later we can retrieve the value to use. But for doing it we need to give a nickname to that memory location so that we can refer to it. That's identifier, the nickname.

Rules for writing Identifiers

There are some rules for writing Identifiers. But first you must know Python is case sensitive. That means **Name** and **name** are two different identifiers in Python. Here are some rules for writing Identifiers in python.

1. Identifiers can be combination of uppercase and lowercase letters, digits or an underscore(_). So **myVariable**, **variable_1**, **variable_for_print** all are valid python identifiers.
2. An Identifier cannot start with digit. So while **variable1** is valid, **1variable** is not valid.
3. We can't use special symbols like !,#,@,%,$ etc in our Identifier.

4. Identifier can be of any length.

Though these are hard rules for writing identifiers, also there are some naming conventions which are not mandatory but rather good practices to follow.

1. Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
2. Starting an identifier with a single leading underscore indicates the identifier is private.
3. If the identifier starts and ends with two underscores, than means the identifier is language-defined special name.
4. While **c = 10** is valid, writing **count = 10** would make more sense and it would be easier to figure out what it does even when you look at your code after a long time.
5. Multiple words can be separated using an underscore, for example **this_is_a_variable**.

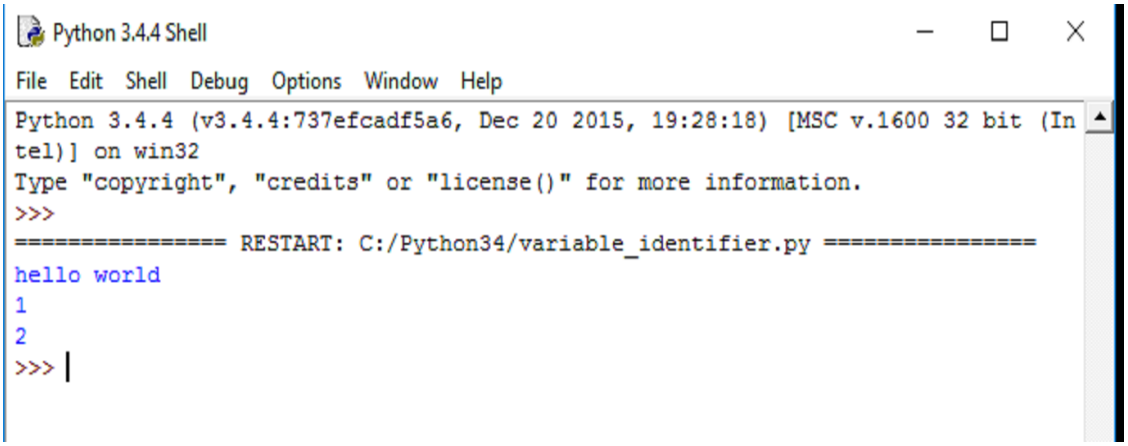
Here's a sample program for python variables.

```
myVariable="hello world"
print(myVariable)

var1=1
print(var1)

var2=2
print(var2)
```

If you run the program, the output will be like below image.



Data Types

A data type defines the type of data, for example 123 is an integer data while "hello" is a String type of data. The data types in Python are divided in two categories:

1. Immutable data types – Values cannot be changed.
2. Mutable data types – Values can be changed

Immutable data types in Python are:
1. Numbers
2. String
3. Tuple

Mutable data types in Python are:
1. List
2. Dictionaries
3. Sets

String literals

Definition:

A string literal is where you specify the contents of a string in a program.

```
>>>a='A string'
```

Here 'A string' is a string literal. The variable *a* is a string variable, or, better put in Python, a variable that points to a string.

A string is usually a bit of text (sequence of characters). In Python we use " (double quotes) or ' (single quotes) to represent a string.

1. How to create a String in Python

There are several ways to create strings in Python.
1. We can use ' (single quotes), see the string str in the following code.
2. We can use " (double quotes), see the string str2 in the source code below.
3. Triple double quotes """ and triple single quotes ''' are used for creating multi-line strings in Python. See the strings str3 and str4 in the following example.

```
# lets see the ways to create strings in Python
str='beginnersbook'
print(str)

str2 ="Chaitanya"
print(str2)

# multi-line string
str3 ="""Welcome to
  Beginnersbook.com"""
print(str3)

str4 ='''This is a tech
blog'''
```

```
print(str4)
```

Output:

```
beginnersbook
Chaitanya
Welcome to
Beginnersbook.com
This is a tech
blog
```

2. How to access strings in Python

A string is nothing but an array of characters so we can use the indexes to access the characters of a it. Just like arrays, the indexes start from 0 to the length-1.

You will get **IndexError** if you try to access the character which is not in the range. For example, if a string is of length 6 and you try to access the 8th char of it then you will get this error.

You will get **TypeError** if you do not use integers as indexes, for example if you use a float as an index then you will get this error.

```
str="Kevin"
# displaying whole string
print(str)

# displaying first character of string
print(str[0])

# displaying third character of string
print(str[2])

# displaying the last character of the string
print(str[-1])

# displaying the second last char of string
print(str[-2])
```

Output:

```
Kevin
K
v
n
i
```

3. Python String Operations

3.1. Getting a substring in Python – Slicing operation

We can slice a string to get a **substring** out of it. To understand the concept of **slicing** we must understand the positive and negative indexes in Python (see the example above to understand this).

Let's take a look at the few examples of slicing.

```
str="Beginnersbook"

# displaying whole string
print("The original string is: ",str)

# slicing 10th to the last character
print("str[9:]: ",str[9:])

# slicing 3rd to 6th character
print("str[2:6]: ",str[2:6])

# slicing from start to the 9th character
print("str[:9]: ",str[:9])

# slicing from 10th to second last character
print("str[9:-1]: ",str[9:-1])
```

Output:

```
The original stringis:Beginnersbook
str[9:]:  book
str[2:6]:ginn
str[:9]:Beginners
str[9:-1]:  boo
```

3.2 Concatenation of strings in Python

The + operator is used for **string concatenation in Python**. Let's take an example to understand this:

```
str1 ="One"
str2 ="Two"
str3 ="Three"

# Concatenation of three strings
print(str1 + str2 + str3)
```

Output:

```
OneTwoThree
```

Note: When **+ operator** is used on numbers it adds them but when it used on strings it concatenates them. However if you try to use this between string and number then it will throw TypeError.
For example:

```
s ="one"
n =2
print(s+n)
```

Output: TypeError: must be str,notint

3.3 Repetition of string – Replication operator

We can use * operator to repeat a string by specified number of times.

```
str="ABC"

# repeating the string str by 3 times
print(str*3)
```

Output:
ABCABCABC

3.4 Python Membership Operators in Strings

in: This checks whether a string is present in another string or not. It returns true if the entire string is found else it returns false.
not in: It works just opposite to what "in" operator does. It returns true if the string is not found in the specified string else it returns false.

```
str="Welcome to beginnersbook.com"
str2 ="Welcome"
str3 ="Chaitanya"
str4 ="XYZ"

# str2 is in str? True
print(str2 instr)

# str3 is in str? False
print(str3 instr)

# str4 not in str? True
print(str4 notinstr)
```

Output:
True
False
True

3.5 Python – Relational Operators on Strings

The **relational operator's** works on strings based on the ASCII values of characters.
The ASCII value of a is 97, b is 98 and so on.
The ASCII value of A is 65, B is 66 and so on.

```
str="ABC"
str2 ="aBC"
str3 ="XYZ"
str4 ="XYz"

# ASCII value of str2 is >str? True
print(str2 >str)

# ASCII value of str3 is > str4? False
print(str3 > str4)
```

Output:

True
False

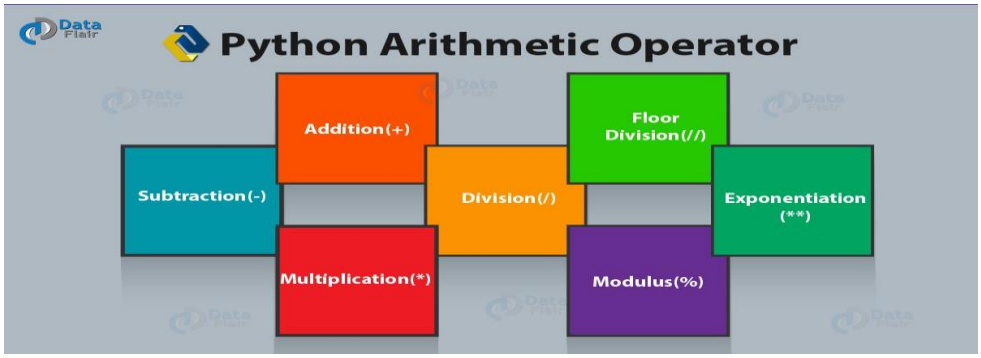
Introduction to Python Operator

Python Operator falls into 7 categories:

- Python Arithmetic Operator
- Python Relational Operator
- Python Assignment Operator
- Python Logical Operator
- Python Membership Operator
- Python Identity Operator
- Python Bitwise Operator

Python Arithmetic Operator

These Python arithmetic operators include Python operators for basic mathematical operations.



Arithmetic Operators in Python

a. Addition(+)

Adds the values on either side of the operator.

1. >>>3+4

Output: 7

b. Subtraction(-)

Subtracts the value on the right from the one on the left.

1. >>>3-4

Output: -1

c. Multiplication(*)

Multiplies the values on either side of the operator.

1. >>>3*4

Output: 12

d. Division(/)

Divides the value on the left by the one on the right. Notice that division results in a floating-point value.

1. >>>3/4

Output: 0.75

e. Exponentiation(**)

Raises the first number to the power of the second.

1. >>>3**4

Output: 81

f. Floor Division(//)

Divides and returns the integer value of the quotient. It dumps the digits after the decimal.

1. >>>3//4
2. >>>4//3

Output: 1

1. >>>10//3

Output: 3

g. Modulus(%)

Divides and returns the value of the remainder.

1. >>>3%4

Output: 3

1. >>>4%3

Output: 1

1. >>>10%3

Output: 1

1. >>>10.5%3

Output: 1.5

Python Relational Operator

Relational Operators in Python

Relational Python Operator carries out the comparison between operands. They tell us whether an operand is greater than the other, lesser, equal, or a combination of those.

a. Less than(<)

This operator checks if the value on the left of the operator is lesser than the one on the right.

1. >>>3<4

Output: True

b. Greater than(>)

It checks if the value on the left of the operator is greater than the one on the right.

1. >>>3>4

Output: False

c. Less than or equal to(<=)

It checks if the value on the left of the operator is lesser than or equal to the one on the right.

1. >>>7<=7

Output: True

d. Greater than or equal to(>=)

It checks if the value on the left of the operator is greater than or equal to the one on the right.

1. >>>0>=0

Output: True

e. Equal to(= =)

This operator checks if the value on the left of the operator is equal to the one on the right. 1 is equal to the Boolean value True, but 2 isn't. Also, 0 is equal to False.

1. >>>3==3.0

Output: True

1. >>>1==True

Output: True

1. >>>7==True

Output: False

1. >>>0==False

Output: True

1. >>>0.5==True

Output: False

f. Not equal to (!=)

It checks if the value on the left of the operator is not equal to the one on the right. The Python operator <> does the same job, but has been abandoned in Python 3.

When the condition for a relative operator is fulfilled, it returns True. Otherwise, it returns False. You can use this return value in a further statement or expression.

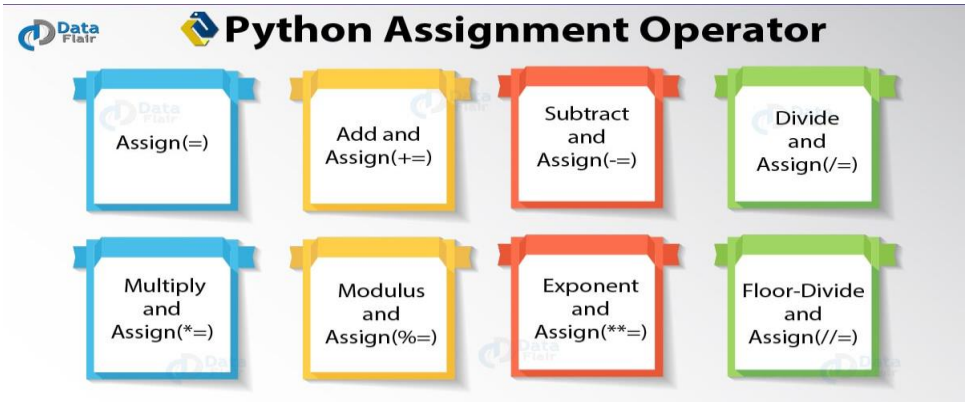
1. >>>1!=-1.0

Output: False

1. >>> -1<>-1.0

#This causes a syntax error

Python Assignment Operator



Assignment Operators in Python

Assignment Python Operator

An assignment operator assigns a value to a variable. It may manipulate the value by a factor before assigning it. We have 8 assignment operators- one plain, and seven for the 7 arithmetic python operators.

a. Assign(=)

Assigns a value to the expression on the left. Notice that = = is used for comparing, but = is used for assigning.

```
1. >>> a=7
2. >>>print(a)
```

Output: 7

b. Add and Assign(+=)

Adds the values on either side and assigns it to the expression on the left. a+=10 is the same as a=a+10.

The same goes for all the next assignment operators.

```
1. >>> a+=2
2. >>>print(a)
```

Output: 9

c. Subtract and Assign(-=)

Subtracts the value on the right from the value on the left. Then it assigns it to the expression on the left.

```
1. >>> a-=2
2. >>>print(a)
```

Output: 7

d. Divide and Assign(/=)

Divides the value on the left by the one on the right. Then it assigns it to the expression on the left.

```
1. >>> a/=7
2. >>>print(a)
```

Output: 1.0

e. Multiply and Assign(*=)

Multiplies the values on either side. Then it assigns it to the expression on the left.

```
1. >>> a*=8
2. >>>print(a)
```

Output: 8.0

f. Modulus and Assign(%=)

Performs modulus on the values on either side. Then it assigns it to the expression on the left.

```
1. >>> a%=3
2. >>>print(a)
```
Output: 2.0

g. Exponent and Assign(=)**

Performs exponentiation on the values on either side.Then assigns it to the expression on the left.

```
1. >>> a**=5
2. >>>print(a)
```
Output: 32.0

h. Floor-Divide and Assign(//=)

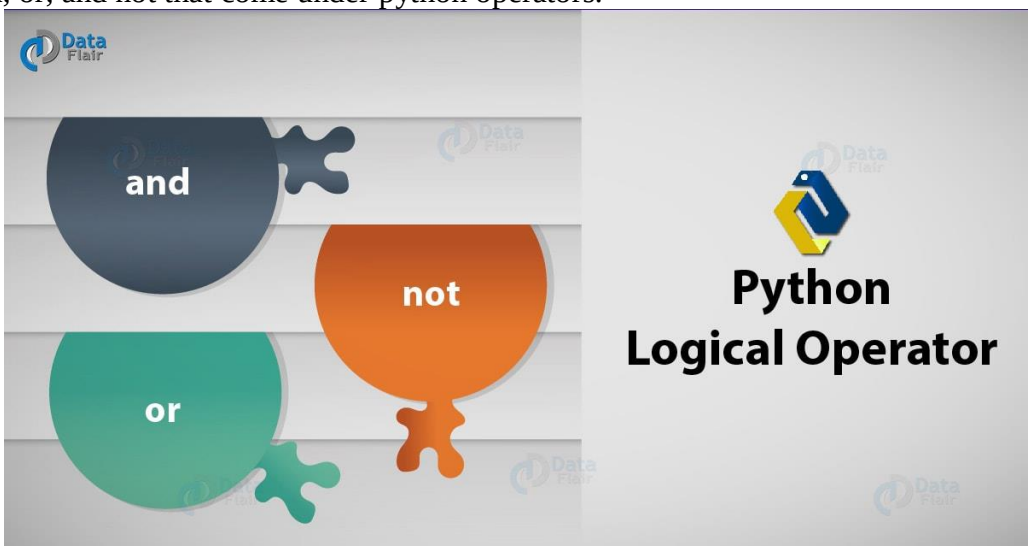
Performs floor-division on the values on either side.Then assigns it to the expression on the left.

```
1. >>>a//=3
2. >>>print(a)
```
Output: 10.0

This is one of the important Python Operator.

Python Logical Operator

These are conjunctions that you can use to combine more than one condition. We have three Python logical operator – and, or, and not that come under python operators.



Logical Operators in Python

a. and

If the conditions on both the sides of the operator are true, then the expression as a whole is true.

```
1. >>> a=7>7 and 2>-1
2. >>>print(a)
```
Output: False

b. or

The expression is false only if both the statements around the operator are false. Otherwise, it is true.

```
1. >>> a=7>7 or 2>-1
2. >>>print(a)
```

Output: True

'and' returns the first False value or the last value; 'or' returns the first True value or the last value

```
1. >>>7 and 0 or 5
```

Output: 5

c. not

This inverts the **Boolean value** of an expression. It converts True to False, and False to True. As you can see below, the Boolean value for 0 is False. So, not inverts it to True.

```
1. >>> a=not(0)
2. >>>print(a)
```

Output: True

Membership Python Operator

These operators test whether a value is a member of a **sequence**. The sequence may be a **list**, a **string**, or a **tuple**. We have two membership python operators- 'in' and 'not in'.

a. in

This checks if a value is a member of a sequence. In our example, we see that the string 'fox' does not belong to the list pets. But the string 'cat' belongs to it, so it returns true. Also, the string 'me' is a substring to the string 'disappointment'. Therefore, it returns true.

```
1. >>> pets=['dog','cat','ferret']
2. >>> 'fox' in pets
```

Output: False

```
1. >>> 'cat' in pets
```

Output: True

```
1. >>> 'me' in 'disappointment'
```

Output: True

b. not in

Unlike 'in', 'not in' checks if a value is not a member of a sequence.

```
1. >>> 'pot' not in 'disappointment'
```

Output: True

Python Identity Operator

These operators test if the two operands share an identity. We have two identity operators- 'is' and 'is not'.

a. is

If two operands have the same identity, it returns True. Otherwise, it returns False. Here, 2 is not the same as 20, so it returns False. Also, '2' and "2" are the same. The difference in quotes does not make them different. So, it returns True.

1. >>>2 is 20

Output: False

1. >>> '2' is "2"

Output: True

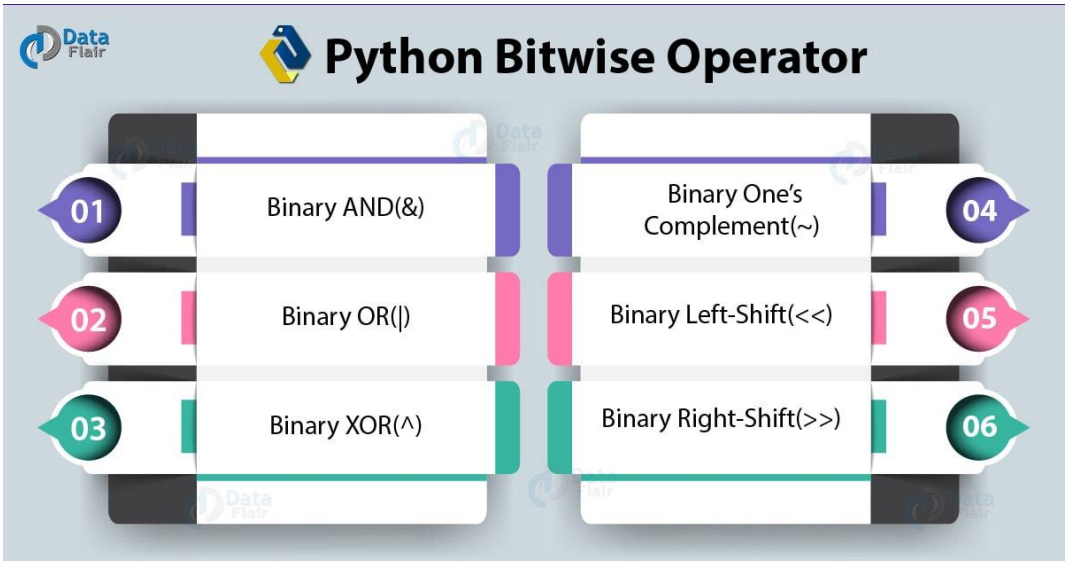
b. is not

2 is a number, and '2' is a string. So, it returns a True to that.

1. >>>2 is not '2'

Output: True

Python Bitwise Operator



Bitwise Operators in Python

On the operands, these operate bit by bit.

a. Binary AND(&)

It performs bit by bit AND operation on the two values. Here, binary for 2 is 10, and that for 3 is 11. &-ing them results in 10, which is binary for 2.Similarly, &-ing 011(3) and 100(4) results in 000(0).

1. >>>2&3

Output: 2

1. >>>3&4

Output: 0

b. Binary OR(|)

It performs bit by bit OR on the two values. Here, OR-ing 10(2) and 11(3) results in 11(3).

1. >>>2|3

Output: 3

c. Binary XOR(^)

It performs bit by bit XOR(exclusive-OR) on the two values. Here, XOR-ing 10(2) and 11(3) results in 01(1).

1. >>>2^3

Output: 1

d. Binary One's Complement(~)

It returns the one's complement of a number's binary. It flips the bits. Binary for 2 is 00000010. Its one's complement is 11111101. This is binary for -3. So, this results in -3. Similarly, ~1 results in -2.

1. >>>~-3

Output: 2

Again, one's complement of -3 is 2.

e. Binary Left-Shift(<<)

It shifts the value of the left operand the number of places to the left that the right operand specifies. Here, binary of 2 is 10. 2<<2 shifts it two places to the left. This results in 1000, which is binary for 8.

1. >>>2<<2

Output: 8

f. Binary Right-Shift(>>)

It shifts the value of the left operand the number of places to the right that the right operand specifies. Here, binary of 3 is 11. 3>>2 shifts it two places to the right. This results in 00, which is binary for 0. Similarly, 3>>1 shifts it one place to the right. This results in 01, which is binary for 1.

1. >>>3>>2
2. >>>3>>1

Output: 1

Increment and Decrement Operators in Python

If you're familiar with Python, you would have known Increment and Decrement operators (both pre and post) are not allowed in it.

Python is designed to be consistent and readable. One common error by a novice programmer in languages with ++ and -- operators is mixing up the differences (both in precedence and in return value) between pre and post increment/decrement operators. Simple increment and decrement operators aren't needed as much as in other languages.

You don't write things like :
for (int i = 0; i < 5; ++i)

In Python, instead we write it like

```
# A Sample Python program to show loop (unlike many
# other languages, it doesn't use ++)
for i in range(0, 5):
   print(i)
```

Output:

```
0
1
2
3
4
```

We can almost always avoid use of ++ and --. For example, x++ can be written as x += 1 and x-- can be written as x -= 1.

Possible Questions

2 Mark Questions

1. Define string. How to get a string at run time?
2. List the features of python.
3. What is interpreter?
4. Define operator and operand?
5. Differentiate for loop and while loop.

6 Mark Questions

1. Explain the different types of operators in python. Explain any two with example.
2. Define string. How to get a string at run time? Explain with example.
3. Demonstrate the various operators in python with suitable examples
4. How to make a calculator program in python? Explain it.
5. Define String. Create a program to reverse a string without using recursion.
6. Explain Ternary operator and Bit wise operator with suitable program.
7. Define methods in a string with an example program using at least five methods.
8. Illustrate with example. I) Relational operator ii) Logical operator

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

III B.Sc(CS) (BATCH 2017-2020) V SEMESTER

PROGRAMMING IN PYTHON (17CSU504B)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT IV

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	The following is NOT an example of a data type.	int	public	double	void	public
2	Identifiers must contain at least ____character	one	two	three	four	one
3	A _____ word cannot be used as an identifier	variable	reserved	string	token	reserved
4	_____ are identifiers that have predefined meanings in Python	Keywords	string	variable	token	Keywords
5	A _____ operator performs an operation using one operand	binary	unary	trinary	none	unary
6	_____ expression, sometimes called a predicate	bitwise	assignemnt	Boolean	arithmetic	Boolean
7	In Python, a string literal is enclosed in _____.	parentheses	brackets	single-quotes	braces	single-quotes
8	Suppose x is a char variable with a value 'b'. What will be displayed by the statement print(chr(ord(x) + 1))?	a	b	c	d	c
9	What is chr(ord('B')))?	A	B	C	D	B

10	Which of the following statement prints smith\exam1\test.txt?	print("smith\exam1\test.txt")	print("smith\\exam1\\test.txt")	print("smith\"exam1\"test.txt")	print("smith"\exam1"\test.txt")	print("smith\\exam1\\test.txt")
11	The Unicode of 'a' is 97. What is the Unicode for 'c'?	96	97	98	99	99
12	Suppose s = "Welcome", what is type(s)?	int	float	string	str	str
13	The format function returns _____.	an int	a float	a str	a chat	a str
14	Which of the following operators are right-associative.	=	*	+	_	=
15	Assume x = 4 and y = 5, Which of the following is true?	not (x == 4)	x != 4	x == 5	x != 5	x != 5
16	Which operator is overloaded by the or() function?	||	|	//	/	|
17	What is the output of the following program : i = 0 while i < 3: print i i++ print i+1	0 2 1 3 2 4	0 1 2 3 4 5	Error	1 0 2 4 3 5	Error
18	Which function overloads the >> operator?	more()	gt()	ge()	None of the above	None of the above
19	_____ creates a list.	list1 = list()	list1 = []	list1 = list([12, 4, 4])	list1 = [12, 4, 4]	All
20	Which of the following statements is used to create an empty set?	{ }	set()	[ ]	( )	set()

21	What is the output of the following piece of code when executed in the python shell? a={1,2,3} a.intersection_update({2,3,4,5}) a	{2,3}	Error, duplicate item present in list	Error, no method called intersection_update for set data type	{1,4,5}	{2,3}
22	Which of the following lines of code will result in an error?	s={abs}	s={4, 'abc', (1,2)}	s={2, 2.2, 3, 'xyz'}	s={san}	s={san}
23	hat is the output of the line of code shown below, if s1= {1, 2, 3}? s1.issubset(s1)	TRUE	Error	No output	FALSE	TRUE
24	What is the output of the code shown below? s=set([1, 2, 3]) s.union([4, 5]) s|([4, 5])	{1, 2, 3, 4, 5}{1, 2, 3, 4, 5}	Error{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5}Error	ErrorError	{1, 2, 3, 4, 5}Error
25	Which of the following function capitalizes first letter of string?	shuffle(lst)	capitalize()	isalnum()	isdigit()	capitalize()
26	Which of the following function checks in a string that all characters are digits?	shuffle(lst)	capitalize()	isalnum()	isdigit()	isdigit()
27	Which of the following function convert an integer to octal string in python?	unichr(x)	ord(x)	hex()	oct(x)	oct(x)
28	What is the name of data type for character in python ?	char	python do not have any data type for characters	charcter	chr	python do not have any data type for characters
29	In python 3 what does // operator do ?	Float division	Integer division	returns remainder	same as a**b	Integer division

30	What is "Programming is fun"[4: 6]?	ram	ra	r	pr	ra
31	What is "Programming is fun"[-1]?	pr	ram	ra	n	n
32	What is "Programming is fun"[1:1]?	pr	p	r	' '	' '
33	Given a string s = "Welcome", which of the following code is incorrect?	print(s[0])	print(s.lower())	s[1] = 'r'	print(s.strip())	s[1] = 'r'
34	Given a string s = "Programming is fun", what is s.find('ram')?	1	2	3	4	4
35	Given a string s = "Programming is fun", what is s.startswith('Program')?	0	1	TRUE	FALSE	TRUE
36	A ____ is an associative array of key-value pairs	dictionary	list	tuple	sequence	dictionary
37	A _____ is though similar to a list, but it's immutable.	dictionary	list	tuple	sequence	tuple
38	A ___, in Python, stores a sequence of objects in a defined order.	dictionary	list	tuple	sequence	list
39	What Will Be The Output Of The Following Code Snippet? a=[1,2,3,4,5,6,7,8,9] print(a[::2])	[1,2]	[8,9]	[1,3,5,7,9]	[1,2,3]	[1,3,5,7,9]
40	What is the output of the expression? round(4.5676,2)?	4.5	4.6	4.57	4.56	4.57