

# ALGORITHMIC PROBLEM SOLVING

# 1

## Computer

- A computer is a machine that manipulates data according to a set of instructions.
- A computer is a fast operating electronic device that receives data (input) and process the data, stores data and produces the resultant data (output)
- A computer is an electronic device used for storing data and to process to give the desired results. They play an important role in our daily life. It reduces the manual task, performs complex tasks easily in a fraction of seconds in nanoseconds speed.
- Computers are used for variety of purposes, starting from simple arithmetic calculations to very complex data analysis as weather forecasting, satellite launching.

## Classification of computers

- Computers are available in different shape, size and weights
- Due to these different shapes and sizes they perform different sorts of jobs from one another.
- They can be classified in different ways.

i) Based on Hardware design	a) Analog b) Digital c) Hybrid
ii) Based on Utility	a) General purpose computers b) Special purpose computers
iii) Based on Size and Capacity	a) Micro computer b) Mini computer c) Mainframe computer d) Super computer
iv) Based on Mode of use	a) Palmtop PCs b) Laptop PCs c) Personal computer d) Work station e) Mainframe system f) Clients and service

### 2.1. Based on Hardware Design.

Based on the Hardware Design computers are classified as  
Analog computers (b) Digital Computer (c) Hybrid Computer

(a)

### (a) Analog Computer

- **Analog signals:** An analog signal is a continuous variable electromagnetic wave.
- The analog computers operate by measuring instead of counting.
- In Analog Computer, the input data is continuously changing electrical or non-electrical information.
- Computations are carried out with physical quantities such as Voltage, Length, Current, Temperature etc.

### Characteristics

- It operates by measuring.
- It functions on continuously varying quantity.
- The output is usually represented in the form of graphs.
- The calculations are first converted into equations and later converted into electrical signals.
- The accuracy of the output is poor.
- It has limited memory space.
- It is not versatile, i.e. it has limited applications.
- Speed is very low.

Eg. Electronic weighing scale, patient's heart beat, blood pressure, temperature.

### b) Digital Computer

- As the name implies, the digital computer with quantities represented as digits.
- It is represented by binary notation in the form of 0's and 1's.
- Digital computers are much faster than analog computers and are more accurate.
- The basic operation performed by a digital computer is addition.
- Hence, the other operations such as multiplication, division, subtraction and exponentiation are first changed into "Addition and then compute".
- A digital signal consists of discrete voltage levels, a light pulses, that represent either binary 1 (ON) or a binary 0 (OFF).

### Characteristics

- It operates by counting.
- It functions on discrete numbers.
- The calculations are converted into binary numbers.
- Its accuracy is good.
- It has large memory space.
- It is versatile in nature.
- It is suitable for a number of applications.
- Its processing speed is high.

### c) Hybrid Computer

- In Hybrid Computer, an attempt is made to combine the qualities of both Analog and Digital computers.
- In a hybrid computer, the measuring functions are performed by the analog way while control and logic functions are digital in nature.

## 2.2. Based on Utility

Based on utility, the computers can be classified into

- General purpose computer b) Special purpose computer

### a) General purpose Computer

- These are design and constructed for almost all the needs of the society.
- They can perform various operations.
- They are able to perform according to the programs created to meet different needs.
- These can be used for a variety of tasks from financial accounting to mathematical calculations, from designing textile prints to controlling machinery.
- They are also flexible and can be used to work on business and scientific problems.

### b) Special purpose Computer

- Special purpose computer can be used designed to perform specific functions.
- In such devices, the instructions are permanently pre-programmed.
- Some of the special purpose computers are aircraft control system, electronic voting machines etc.

## 2.3. Based on Size and Capacity

Based on size and capacity, computers are classified into the following ;

(a)Microcomputer (b) Minicomputer (c) Mainframe computer (d)Super computer.

### a) Micro Computer

- Because of its small size and use of Micro processer, this computer is called Micro Computer.
- A micro processer is a processor whose components namely Input, Output and CPU are on a single integrated- circuit chip.
- It is a low -cost
- the word length of a micro computer lies in the range of 8to 32 bits.
- These mainly used in offices, homes, schools, shops, stores etc.
- IBM PCs, Apple Mac, IBM PS/2 are some popular computers of this range.

**Advantages:**

- They are small and portable.
- They are relatively inexpensive.
- They work as soon as they are switched on.
- They do not occupy much space.
- They do not consume much power.

**Disadvantage:****b) Mini Computer**

- Mini computers are larger than the micro computers and are more powerful in terms of processing power.
- Mini computers are mainly multi processor systems where many users simultaneously work on the systems.
- These are even capable of handling more input-output device.
- Their word length is 32 bits.
- The processing speed lies in the range 10 to 30 MIPS, memory (RAM) capacity lies in the range 8MB to 256MB.
- The hard disk capacity lies in the range 380to 3G,it can support up to 64 or even hundred terminals
- It is used for payroll preparation, accounting and scientific computation.
- High performance workstations with graphics input/output capability use minicomputer.

**Advantages:**

- It is a general purpose computer.
- Its storage capacity is about 2Mega words.
- It can support all high level language.
- It can support many terminals i.e. more than 20 terminals.
- It performs almost all the task that a mainframe computer do.
- It is relatively inexpensive.

**Disadvantage:**

- They are slower when compared with main frame computers.
- The memory of mini computers are smaller than mainframes.

**c) Mainframe computer**

- Mainframe computer are larger ,
- Faster and more expensive than other general purpose computers .
- These are used to handle huge volumes of data.
- Their word length may be 48 to 64 bit memory capacity

- the hard disk capacity 100MB to 10 GB or more and processing speed 30 to 100MIPS.
- These computer work with more than one processor at the same time.
- Thus one can see these as multi-user, multiprocessor systems.
- They are used where large amount of data are to be processed or very complex calculations are to be made
- It used in research organizations, large industries, large business and government organizations, banks and airline reservation where large database is required.

#### **Advantages:**

- They are capable of handling all tasks.
- They require large room space.
- Consumption of electricity is very high.
- Maintenance cost is also very high.

#### **d) Super computer:**

- Super computer is the most powerful of all computers.
- They have a high processing speed.
- Their processing speed lies in the range of 400 to 10,000 MIPS,
- The word length 64 to 96 bit or more.
- Memory capacity 256 MB or more,
- Hard disk designed to maximize the number of FLOPS.
- Their FLOPS rating is usually more than 1 gigaflops per second.
- Supercomputers are mainly used for purposes requiring enormous amounts of data to be processed within a very short time,
- Whether Forecasting, Space Research, Weapon Research, Atomic Research, Design of Aircrafts etc., are some of the applications.
- The best known super computers are PARAM series (developed in India)
- Super computer can perform billions of instructions per second.

#### **Advantages :**

- They use several processors working simultaneously.
- The process at a rapid speed.
- They have more main memory.
- They have operations done in parallel, rather than sequentially.

### **2.4. Based on Mode of Use**

Based on mode of use computers are classified as ,

#### **a) Palmtop computer**

- The palmtop computers accept handwritten inputs using an electronic pen, which can be used to write on a Palmtop's screen (besides a tiny keyboard).

- These have small disk storage and can be connected to a wireless network.
- One has to train the system on the user's handwriting before it can be used as a mobile phone, Fax and E-mail machine.
- A version of Microsoft Operating System called Windows-CE is available for palmtops.

#### **b) Laptop Computer**

- Laptop PC's are portable computers with less weight and small enough to rest it on the lap.
- They use a keyboard, flat screen liquid crystal display, and a Pentium processor.
- Laptop computer generally smaller than a briefcase that can easily be transported.
- Laptops come with hard disk, CD ROM and floppy disk drives.
- It will run in batteries. Many Laptops can be connected to a network.
- As Laptops use miniature components which consume low power and have to be packaged in small volume.

#### **c) Personal computers**

- The personal computers were mainly designed to meet the personal computing requirements of users at working place or at home.
- It is a non-portable and general purpose computer which can easily fit in on a normal size office table.
- The personal computers are also almost used by the children and adults for education and entertainments and hence they are now common everywhere.
- The personal computers configuration varies from one another based on their use. It consists a CPU, monitor, a keyboard and a mouse.
- It is usually designed the following two models.
  - Desktop model
  - Tower model

#### **d) Work Stations**

- It is a powerful desktop computer designed to meet the computing needs of users or clients, with better processing, high storage capacity and with efficient and effective graphics display facility.
- The workstation looks similar to a personal computer and can be used by only one person at a time through the Local Area Network.

#### **e) Mainframe system**

- The mainframe system is commonly used for handling voluminous data and in an environment where a large number of users need to share a common computing facility.
- It is housed in a central location with several user terminals and peripherals connected to it. The user terminals are connected with the host processor for accessing as and when required.

#### f) Clients and Servers

- With the increased gain of computer networks, it is possible to interconnect several computer for communicating over the networks to share the several resources or services among the multiplier user.
- **i)Server:** It is a large computer that manages a shared resources and provides a set of shared user services to the client.

**ii)Client:** It is a single user personal computer or workstation and supports a user friendly interface with the end user. It sends request to the server and then gain access with the server.

## Generation of Computer:

Sl. No.	Parameters	1 <sup>st</sup> Generation	2 <sup>nd</sup> Generation	3 <sup>rd</sup> Generation	4 <sup>th</sup> Generation
		1940-1956	1956-1963	1964-Early 1970s	1970-Till date
1	<b>Technology</b>	Vacuum Tubes	Transistors	Integrated Circuits	Micro Processors
2	<b>Input Device</b>	Punched cards paper tapes	Punched cards	Keyboard	Keyboard, mouse , GUI
3.	<b>Output device</b>	Printouts	Printouts	Monitors	monitors
4.	<b>Memory device</b>	Magnetic drum	<u>Magneticcore</u> as primary memory Magnetic disk as secondary memory	Magnetic core as primary memory Magnetic disk as secondary memory	Random Access Memory, Hard Disks
5.	<b>Language</b>	Binary coded language	Assembly Language	High level Language	High level Language
6.	<b>Computation Time</b>	<u>Milli</u> Seconds	Micro seconds	<u>Nano</u> Seconds	Pico seconds
7.	<b>Size</b>	Large	Smaller than 1 <sup>st</sup> Generation	Smaller	Smaller
8.	<b>Air conditioning</b>	Required	Required	Required	Required
9.	<b>Portability</b>	Non Portable	Portable	Portable	Portable
10.	<b>Reliability</b>	Un Reliable	Reliable	Reliable	Reliable
11.	<b>Cost</b>	Very Expensive	Expensive	Less Expensive than previous	Cheaper
12.	<b>Examples</b>	ENIAC, EDVAC, UNIV AC	PDP- 8, IBM1401, IBM7 090	NCR 395, B6500	<u>AppleII</u> , Altair 8800 and CRAY- I

# Algorithm

An algorithm (pronounced AL-go-rith-um) is a *procedure* or formula for solving a problem, based on conducting a *sequence of specified actions*.

*Logical thinking* scientifically exposes something in an abstract. Once the logic of the problem is identified, the next step is to form the correct sequence and procedure of instructions to carry out the task.

- Algorithm is correct a *sequence of procedural* instruction required to carry out the task.
- The Algorithm is often used to *refer the logic* of a program.
- It is one of the *basic tool* used to develop the *problem solving*
- A computer program can be viewed as an elaborate algorithm.
- To multiply single-digit numbers, you probably memorized the multiplication table. In effect, you memorized 100 specific solutions. That kind of knowledge is not algorithmic.
- But if you were “lazy”, you might have learned a few tricks. For example, to find the product of  $n$  and 9, you can write  $n * 1$  as the first digit and  $10 * n$  as the second digit. This trick is a general solution for multiplying any single-digit number by 9. That’s an algorithm!
- Similarly, the techniques you learned for addition with carrying, subtraction with borrowing, and long division are all algorithms.
- One of the characteristics of algorithms is that they do not require any intelligence to carry out the task. They are mechanical processes where each step follows from the last according to a simple set of rules.
- Some of the things that people do naturally, without difficulty or conscious thought, are the hardest to express algorithmically. Understanding natural language is a good example.

## Characteristics of Algorithm

- In Algorithms each and every instruction *should be precise* and *clear-cut*.
- The instruction in an algorithm should *not be repeated* infinitely.
- The algorithm makes sure that it *will ultimately be terminated*.
- Algorithm should be *written in sequence*.
- It looks like *normal English*.
- The desired result should be obtained only after the algorithm terminated.



## Quality of a good Algorithm

- There are many methods or logic available for problem solving. All of those method and logic may not be good, for given problem.
- The following are the *primary factors* that are often used to judge the quality of the algorithm.

<b>Time</b>	The computer system takes some amount of time to execute a program. The <i>lesser the time</i> required, the better is the algorithm
<b>Memory</b>	To execute a program, the computer system uses some amount of memory storage. The <i>lesser the memory</i> used, the better is the algorithm
<b>Accuracy</b>	Multiple algorithm may provide correct solution to given problem, but few of these may be more <i>accurate result</i> than other, such algorithm may be suitable
<b>Sequence</b>	The procedure of an algorithm must form in a <i>sequence</i> and some of the instructions of an algorithm may be repeated in number of times or until a particular <i>condition is met</i>
<b>Generality</b>	The designed algorithm must solve a single problem and more often algorithm are designed to handle a <i>range of input data</i> to meet this criteria, so the algorithms must be generalized.

## Representation of Algorithm

- The algorithm can be represented in several ways. Generally the programmer follows one of the following ways.

<b>Normal English</b>	The Algorithm can be easily represented in <i>step by step sequence</i> order in normal English, such algorithm are <i>easy</i> to understand write and read.
<b>Flow Chart</b>	The flowchart is a <i>pictorial representation</i> of an algorithm. i.e. algorithm can be represented as a flow chart using standard symbols.
<b>Pseudo code</b>	Pseudo code is also a formal design tool and utilized very well with the rules of <i>structured design and programming</i> .
<b>Decision table</b>	A decision table helps a lot in designing a specific segment of a design. It provides another way to look at a <i>complex, nested selection</i> to help <i>clarify the conditions</i> to be tested and how those conditions should be nested to arrive at the proper actions.
<b>Program</b>	The algorithm represented as a program using any <i>high level language</i> that becomes a program.

## Algorithm to get a valid email from the user:

### Analysis of Algorithms

- The practical goal of algorithm analysis is to predict the performance of different algorithms in order to guide design decisions.
- The goal of algorithm analysis is to make meaningful comparisons between algorithms, and bring out the best one.

*The relative performance of the algorithms might depend on characteristics of the hardware:*

- So one algorithm might be faster on Machine A, another on Machine B.
- Solution: specify a *machine model* and analyze the number of steps, or operations, an algorithm requires under a given model.

*Relative performance might depend on the details of the dataset:*

- For example, some sorting algorithms run faster if the data are already partially sorted; other algorithms run slower in this case.
- Solution: analyze the *worst case* scenario.

*Relative performance also depends on the size of the problem.*

- A sorting algorithm that is fast for small lists might be slow for long lists.
- Solution: express *run time* (or number of operations) *as a function* of problem size.

### Order of growth

- For algorithmic analysis, functions with the same leading term are considered equivalent, even if they have different coefficients.
- An **order of growth** is a set of functions whose growth behavior is considered equivalent.
- For example,  $2n$ ,  $100n$  and  $n+1$  belong to the same order of growth, which is written  $O(n)$  in **Big-Oh notation** and often called **linear** because every function in the set grows linearly with  $n$ .

Order of growth	Name
$O(1)$	constant
$O(\log_b n)$	logarithmic (for any $b$ )
$O(n)$	linear
$O(n \log_b n)$	linearithmic
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(c^n)$	exponential (for any $c$ )

- All functions with the leading term  $n^2$  belong to  $O(n^2)$ ; they are called *quadratic*.
- The following table tells about the various order of growth in the algorithms and their names.

## Building Blocks of Algorithms

- It has been proven that any algorithm can be constructed from just three basic building blocks. These three building blocks are

Building Block	Common name
Sequence	Action
Selection	Decision
Iteration	Repetition or Loop

- An **Action** is one or more instructions that the computer performs in sequential order (from first to last). Example: Compute 17 plus 31.
- A **Decision** is making a choice among several actions. Example: If you listen to KMEL radio, print "Friend of the Camel", otherwise print "Don't like Camels".
- A **Loop** is one or more instructions that the computer performs repeatedly. Example: Print "GOOD GRIEF, CHARLIE BROWN" 77 times.

## Sequential Control

- The algorithm and its steps are to be carried out in a sequential manner.
- Each step is executed exactly once.
- It does not have a branch or control flow.
- The final steps displays the desired output.

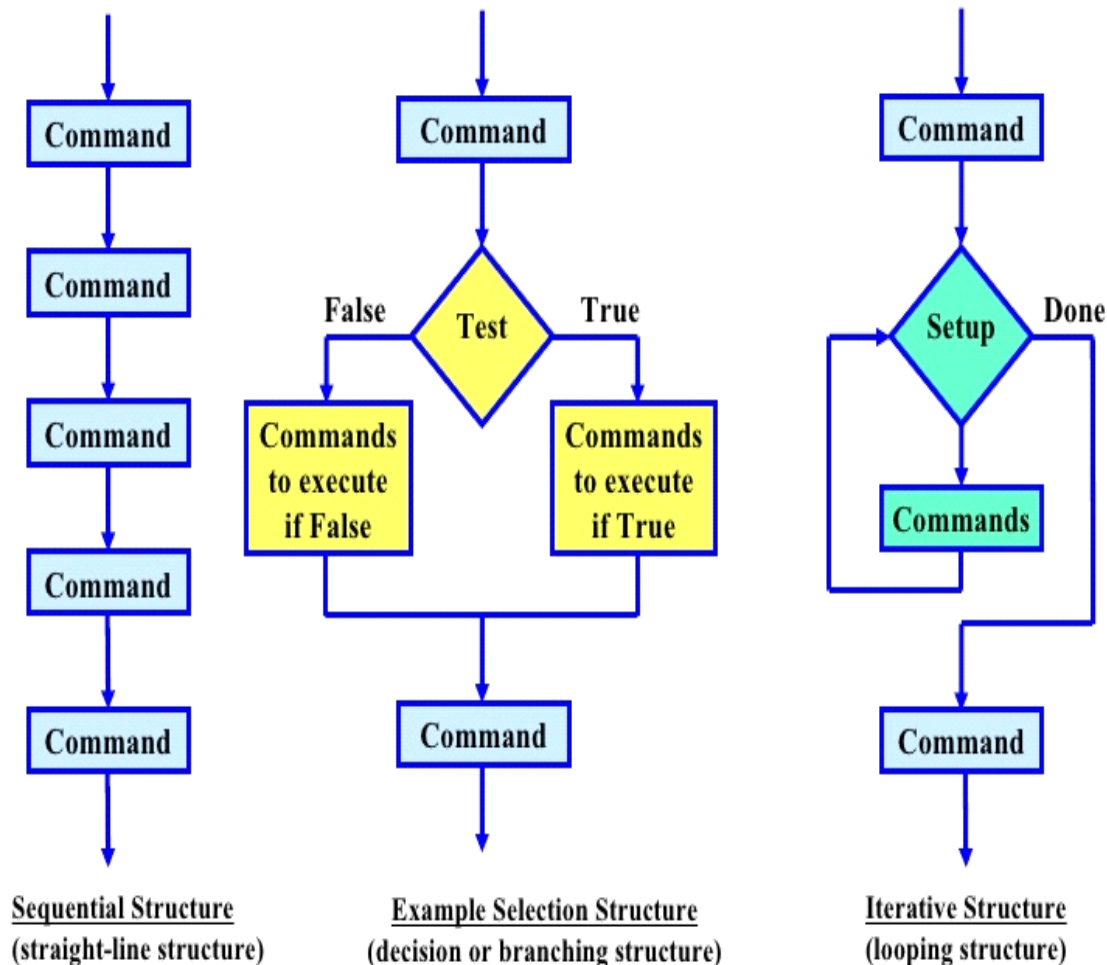
## Selection or Conditional Control

- The algorithm and its steps are carried out on selection basics.
- The selection is done based on the condition.
- One of the number of alternative steps are executes based on a condition.

## Repetition or Control Flow

- The algorithmic steps are conducted repeatedly.
- These are called the loops in the program logics.
- One or more instructions are executed repeatedly depending on a condition.

### Flowcharts for sequential, selection, and iterative control structures



## Statements

➔ Statements in Python are made up of:

<b>Reserved words</b>	words that have <i>pre-defined</i> meanings
<b>Identifiers</b>	words that are <i>created by programmers</i> for names of variables, functions, classes, etc.
<b>Literals</b>	Literal values written in code, like strings or numbers. The few literals are
	<b>Numeric Literal:</b> <i>Integer literal</i> - an actual integer number written in code (4, -10, 18) <i>Float literal</i> - an actual decimal number written in code (4.5, -12.9, 5.0)
	<b>Boolean literal</b> - has two values (True or False)
	<b>String literal</b> - a string in double quotes: ("Hello", "Bye", "Wow!\n")
<b>Operators</b>	Special <i>symbols</i> that perform certain actions on their operands. They are:
	A <i>unary</i> operator has one operand
	A <i>binary</i> operator has two operands
	A <i>ternary</i> operator has three operands (there's only one of these)
<b>Methods</b>	Calls to methods (functions)

## Reserved Words

- *Keywords* are the reserved words in Python.
- We *cannot use* a keyword as [variable name](#), [function](#) name or any other identifier.
- They are used to *define the syntax and structure* of the Python language.
- In Python, keywords are *case sensitive*.
- There are **33 keywords** in Python 3.3. This number can vary slightly in course of time.
- All the keywords *except True, False and None* are in *lowercase* and they must be written as it is.
- The lists of all the keywords are given below.

Keywords in Python programming language				
False	class	finally	is	return
None	continue	for	lambda	try

True	def	from	nonlocal	while
and	del	global	not	with

as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

## Identifiers

- Identifier is the *name* given to entities like *class, functions, variables* etc., in Python.
- It helps *differentiating* one entity from another.
- Identifiers are the names given to the fundamental building blocks in a program.
- Some identifiers are *built-in*, and others can be *created* by the programmer.
- *User-defined* identifiers can consist of *letters, digits, underscores*, and *the dollar-sign* \$.

## Rules for writing identifiers

- An identifier is a long sequence of *characters and numbers*. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_).
- Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
- *No special character* except underscore ( \_ ) can be used as an identifier.
- *First character* of an identifier can be *character, underscore* ( \_ ).
- Must start with a *non-digit*.
- Identifiers are *case sensitive* (count and Count are different variables) .
- We *cannot use special symbols* like !, @, #, \$, % etc., in our identifier.
- If we use special symbols which are not allowed we get a syntax error.
- Reserved words (*keywords*) *cannot* be used as identifiers.
- An identifier can be *any length*.

## Literals

- Literals can be defined as a data that is given in a variable or constant. Python support the following literals:

### String literals:

- String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

### Types of Strings:

There are two types of Strings supported in Python:

- Single *line String*- Strings that are terminated within a single line are known as Single line Strings.
- Multi line String*- A piece of text that is spread along multiple lines is known as Multiple line String. There are two ways to create Multiline Strings:

**Adding back slash at the end of each line .**

**\* Using triple quotations.**

### Numeric literals:

➔ Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers (can be both positive and negative) with no fractional part. Example: 100	Integers of unlimited size followed by lowercase or uppercase L Example: 87032845L	Real numbers with both integer and fractional part Example: -26.2	In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. Example: 3.14j

### Boolean literals:

- A Boolean literal can have any of the two values: True or False

## Special literals:

- Python contains one special literal i.e., none.
- None is used to specify to that field that is not created.
- It is also used for end of lists in Python.

## Literal Collections:

- Collections such as tuples, lists and Dictionary are used in Python.
- We will see them in the following chapters.

## List:

- List contains items of different data types.
- Lists are mutable i.e., modifiable.
- The values stored in List are separated by commas(,)
- Each value is enclosed within a square brackets([]).
- We can store different type of data in a List.
- Value stored in a List can be retrieved using the slice operator([] and [:]).
- The plus sign (+) is the list concatenation
- Asterisk(\*) is the repetition operator.

## Operators

- Operators are particular symbols which operate on some values and produce an output.
- The values are known as Operands.

## Python supports the following operators:

- Arithmetic Operators.
- Relational Operators.
- Assignment Operators.
- Logical Operators.
- Membership Operators.
- Identity Operators.
- Bitwise Operators.

## Arithmetic Operators:

Operators	Description
//	Perform Floor division(gives integer value after division)
+	To perform addition
-	To perform subtraction



*	To perform multiplication
/	To perform division
%	To return remainder after division(Modulus)
**	Perform exponent(raise to power)

### Relational Operators:

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
<>	Not equal to(similar to !=)

### Assignment Operators:

Operators	Description
=	Assignment
/=	Divide and Assign
+=	Add and assign
-=	Subtract and Assign
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign
//=	Floor division and assign

### Logical Operators:

Operators	Description
and	Logical AND(When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true)
not	Logical NOT(Compliment the condition i.e., reverse)

## Membership Operators:

Operators	Description
in	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

## Identity Operators:

Operators	Description
is	Returns true if identity of two operands are same, else false
is not	Returns true if identity of two operands are not same, else false.

## State

- An algorithm describes a *computation*.
- When *executed*, proceeds through a finite number of well-defined *successive states*.
- States are the temporary *intermediate* stages in the flow of execution.
- *Partial output* like intermediate output obtained called states.
- These outputs are then eventually produces the output terminating at a final ending state.
- The transition from one state to the next is not necessarily *deterministic*.

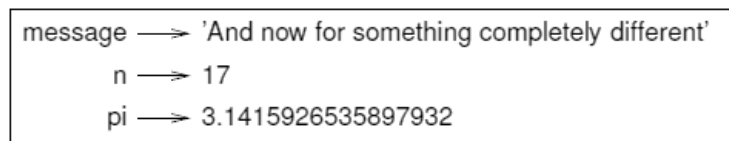


Figure:stated  
iagram

- **State diagram:** A graphical representation of a set of variables and the values they refer to.

## Reassignment

- It is legal to make more than one assignment to the same variable.
- A new assignment makes an existing variable refer to a new value (and stop referring to the old value).
- This is called as *reassignment*.
- The first time we display x, its value is 5; the second time, its value is 7.

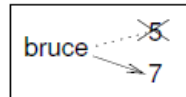


Figure shows what **reassignment** looks like in a state diagram.

- The third line changes the value of 'a', but does not change the value of b.
- So they are no longer equal.
- Reassigning variables is often useful, but you should use it with caution.
- If the values of variables change frequently, it can make the code difficult to read and debug.

## Updating variables

- A common kind of reassignment is an **update**, where the new value of the variable depends on the old.
- This means “get the current value of x, add one, and then update x with the new value”.
- If you try to update a variable that doesn't exist, you get an error, because Python evaluates the right side before it assigns a value to x.
- Before you can update a variable, you have to **initialize** it, usually with a simple assignment.
- Updating a variable by adding 1 is called an **increment**; subtracting 1 is called a **decrement**.

## Control Flow

- **(if/elif/else)**
  - Controls the order in which the code is executed.

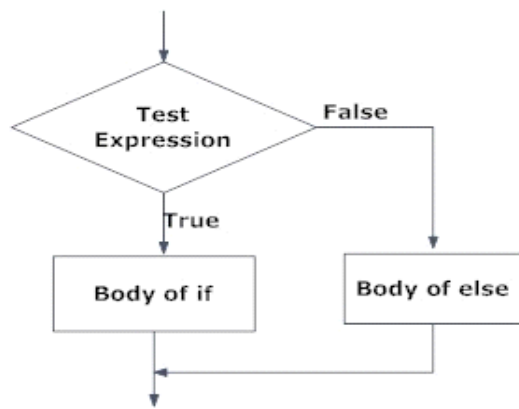


Fig: Operation of if...else statement

- Blocks are delimited by indentation.
- The Python shell automatically increases the indentation depth after a colon(:); to decrease the indentation depth, go four spaces to the left with the Backspace key.
- Press the Enter key twice to leave the logical block.
- Indentation is compulsory in scripts as well.

## for/range

- Iterating with an index.

But most often, it is more readable to iterate over values:

## while/break/continue

- Typical C-style while loop (Mandelbrot problem):

## More advanced features

- break out of enclosing for/while loop:
- continue the next iteration of a loop.:

## Conditional Expressions

- *if object*
- **Evaluates to True:**
  - any non-zero value
  - any sequence with a length > 0
- **Evaluates to False:**
  - any zero value
  - any empty sequence

- *a == b*, Tests equality, with logics:
- *a is b*, Tests identity: both objects are the same.
- *a in b*, For any collection *b*: *b* contains *a*
- If *b* is a dictionary, this tests that *a* is a key of *b*.

# Introduction For Python

In all programming and scripting language, a function is a block of program statements which can be used repetitively in a program. It saves the time of a developer. In Python concept of function is same as in other languages. There are some built-in functions which are part of Python. Besides that, we can defines functions according to our need.

## User defined function

- In Python, a user-defined function's declaration begins with the keyword `def` and followed by the function name.
- The function may take arguments(s) as input within the opening and closing parentheses, just after the function name followed by a colon.
- After defining the function name and arguments(s) a block of program statement(s) start at the next line and these statement(s) must be indented.

Here is the syntax of a user defined function.

### Syntax

```
def function_name(argument1, argument2, ...):
    statement_1
    statement_2
    ....
```

## Call a function

Calling a function in Python is similar to other programming languages, using the function name, parenthesis (opening and closing) and parameter(s). See the syntax, followed by an example.

## Syntax

```
function_name(arg1, arg2)
```

### Example :

```
def avg_number(x, y):  
    print("Average of ",x," and ",y, " is ",(x+y)/2)  
avg_number(3, 4)
```

### Output :

Average of 3 and 4 is 3.5

### Explanation :

1. Lines 1-2 : Details (definition) of the function.
2. Line 3 : Call the function.
3. Line 1 : Pass parameters : x = 3, y = 4
4. Line 2 : Print the value of two parameters as well as their average value.

## Function without arguments :

The following function has no arguments.

```
def function_name():  
    statement_1  
    statement_2  
    ....
```

### Example :

```
def printt():  
    print("This is Python 3.2 Tutorial")  
    print("This is Python 3.2 Tutorial")  
    print("This is Python 3.2 Tutorial")  
printt()
```

### Output :

This is Python 3.2 Tutorial  
This is Python 3.2 Tutorial  
This is Python 3.2 Tutorial

### Explanation :

1. Lines 1-4 : Details (definition) of the function.
2. Line 5 : Call the function.
3. Line 1 : No parameter passes.
4. Line 2-4 : Execute three print statements.

## The Return statement in function

In Python the return statement (the word return followed by an expression.) is used to return a value from a function, return statement without an expression argument returns none. See the syntax.

```
def function_name(argument1, argument2, ...):  
    statement_1  
    statement_2  
    ....  
    return expression  
function_name(arg1, arg2)
```

### Example :

The following function returns the square of the sum of two numbers.

```
def nsquare(x, y):  
    return (x*x + 2*x*y + y*y)  
print("The square of the sum of 2 and 3 is : ", nsquare(2,3))
```

### Output :

The square of the sum of 2 and 3 is : 25

### Explanation :

1. Lines 1-2 : Details (definition) of the function.
2. Line 3 : Call the function within a print statement.
3. Line 1 : Pass the parameters  $x = 2$ ,  $y = 3$
4. Line 2 : Calculate and return the value of  $(x + y)^2$

## Default Values:

In function's parameters list we can specify a default value(s) for one or more arguments. A default value can be written in the format "argument1 = value", therefore

we will have the option to declare or not declare a value for those arguments. See the following example.

### Example :

The following function returns the square of the sum of two numbers, where default value of the second argument is 2.

```
def nsquare(x, y = 2):  
    return (x*x + 2*x*y + y*y)  
print("The square of the sum of 2 and 3 is : ", nsquare(2))  
print("The square of the sum of 2 and 3 is : ", nsquare(2,4))
```

### Output :

The square of the sum of 2 and 2 is : 16

The square of the sum of 2 and 4 is : 36

### Explanation :

Lines 1-2 : Details (definition) of the function.

For first print statement [ Line no 3]

Line 3 : Call the function without a second argument, within a print statement.

Line 1 : Pass the parameters x = 2, default value.

Line 2 : Calculate and return the value of  $(x + y)^2$

For second print statement [ Line no 4]

Line 3 : Call the function with all arguments, within a print statement.

Line 1 : Pass the parameters x = 2, y = 4.

Line 2 : Calculate and return the value of  $(x + y)^2$

## Keyword Arguments

We have already learned how to use default arguments values, functions can also be called using keyword arguments. Arguments which are preceded with a variable name followed by a '=' sign (e.g. var\_name="") are called keyword arguments.

All the keyword arguments passed must match one of the arguments accepted by the function. You may change the order of appearance of the keyword. See the following example.



### Example :

```
def marks(english, math = 85, science = 80):  
    print('Marks in : English is - ', english, ', Math - ',  
          math, ', Science - ', science)  
marks(71, 77)  
marks(65, science = 74)  
marks(science = 70, math = 90, english = 75)
```

### Output :

Marks in : English is - 71 , Math - 77 , Science - 80  
Marks in : English is - 65 , Math - 85 , Science - 74  
Marks in : English is - 75 , Math - 90 , Science - 70

### Explanation :

Line 1 : The function named marks has three parameters, there is no default value in the first parameter (english) but remaining parameters have default values (math = 85, science = 80).

Line 3 : The parameter english gets the value of 71, math gets the value 77 and science gets the default value of 80.

Line 4 : The parameter english gets the value of 65, math gets the default value of 85 and science gets the value of 74 due to keyword arguments.

Line 5 : Here we use three keyword arguments and the parameter english gets the value 75, math gets the value 90 and science gets the value 70.

### Arbitrary Argument Lists

The arbitrary argument list is another way to pass arguments to a function. In the function body, these arguments will be wrapped in a tuple and it can be defined with \*args construct. Before this variable, you can define a number of arguments or no argument.

### Example :

```
def sum(*numbers):  
    s = 0  
    for n in numbers:  
        s += n  
    return s  
print(sum(1,2,3,4))
```

**Output :**

10

## Lambda Forms

In Python, small anonymous (unnamed) functions can be created with lambda keyword. Lambda forms can be used as an argument to other function where function objects are required but syntactically they are restricted to a single expression. A function like this:

```
def average(x, y):  
    return (x + y)/2  
print(average(4, 3))
```

may also be defined using lambda

```
print((lambda x, y: (x + y)/2)(4, 3))
```

**Output :**

3.5

## Methods

- Methods are simply another kind of function that resides in classes.
- You create and work with methods in Python in precisely the same way that you do functions, except that methods are always associated with a class.
- You can create two kinds of methods:
  - Those associated with the class itself .
  - Those associated with an instance of a class.

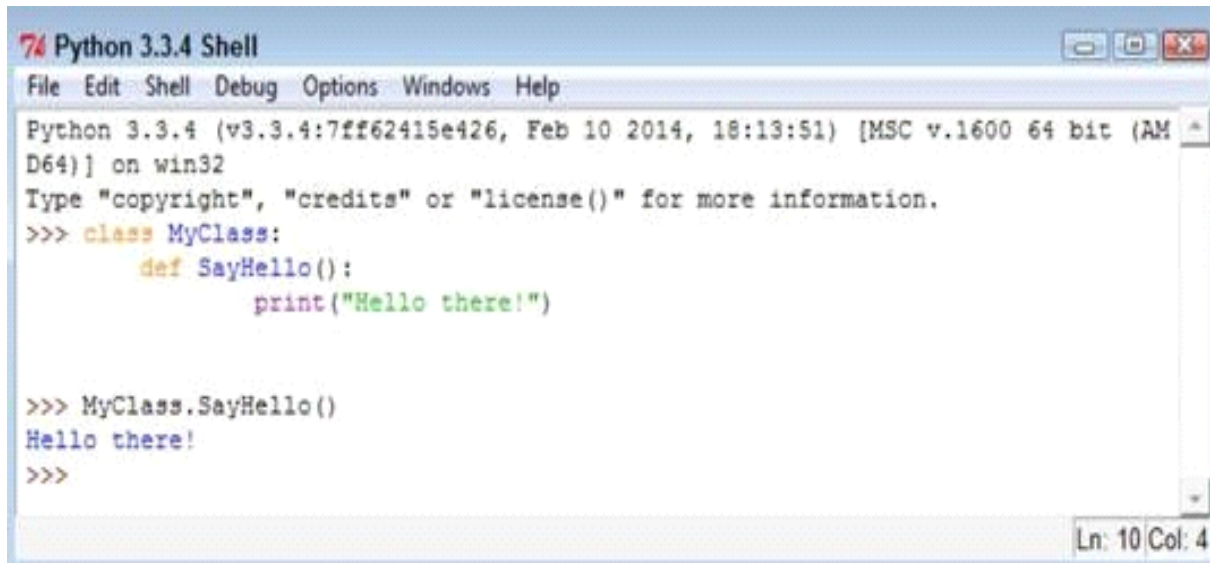
## Creating class methods

- A *class method* is one that you execute directly from the class without creating an instance of the class.
- Sometimes you need to create methods that execute from the class.
- The following steps demonstrate how to create and use a class method.
  - Open a Python Shell window
  - Type the following code

- The example class contains a single defined attribute, SayHello().
- This method doesn't accept any arguments and doesn't return any values.
- It simply prints a message as output.

**Type MyClass.SayHello() and press Enter.**

- The example outputs the expected string. Notice that you didn't need to create an instance of the class — the method is available immediately for use.



```

Python 3.3.4 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.4 (v3.3.4:7ff62415e426, Feb 10 2014, 18:13:51) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> class MyClass:
    def SayHello():
        print("Hello there!")

>>> MyClass.SayHello()
Hello there!
>>>
Ln: 10 Col: 4

```

**Close the Python Shell window.**

- A class method can work only with class data. It doesn't know about any data associated with an instance of the class.
- You can pass it data as an argument, and the method can return information as needed, but it can't access the instance data.
- As a consequence, you need to exercise care when creating class methods to ensure that they're essentially self-contained.

## Creating instance methods

- An *instance method* is one that is part of the individual instances.
- You use instance methods to manipulate the data that the class manages.
- As a consequence, you can't use instance methods until you instantiate an object from the class.
- All instance methods accept a single argument as a minimum, self.

- The self argument points at the particular instance that the application is using to manipulate data.
- Without the self argument, the method wouldn't know which instance data to use.
- However, self isn't considered an accessible argument — the value for self is supplied by Python, and you can't change it as part of calling the method.
- The following steps demonstrate how to create and use instance methods in Python.

### Open a Python Shell window.

- You see the familiar Python prompt.

**Type the following code** (pressing Enter after each line and pressing Enter twice after the last line):

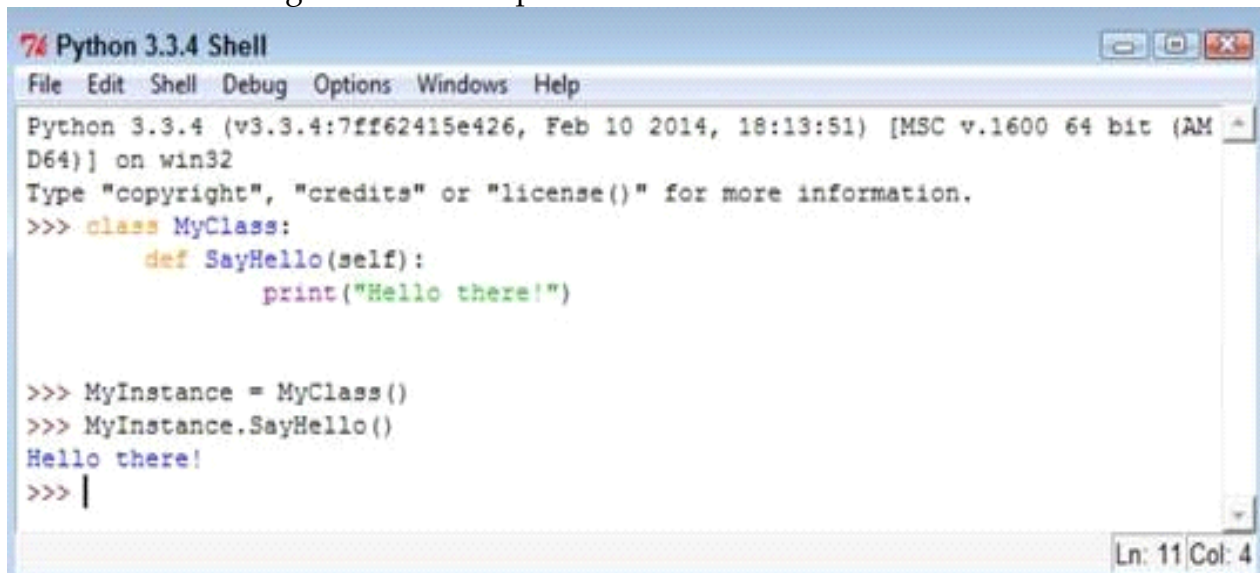
- The example class contains a single defined attribute, SayHello().
- This method doesn't accept any special arguments and doesn't return any values.
- It simply prints a message as output. However, the method works just fine for demonstration purposes.

Type `MyInstance = MyClass()` and press Enter.

- Python creates an instance of MyClass named MyInstance.

Type `MyInstance.SayHello()` and press Enter.

You see this message. After this step close the window.



```
Python 3.3.4 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.4 (v3.3.4:7ff62415e426, Feb 10 2014, 18:13:51) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> class MyClass:
    def SayHello(self):
        print("Hello there!")

>>> MyInstance = MyClass()
>>> MyInstance.SayHello()
Hello there!
>>> |
```

Ln: 11 Col: 4

# Notations

## Pseudo code

- Pseudo code is an *ordered version of your code* written for human understanding rather than machine understanding.
- Pseudo code is a *formal design tool*, developed with the structured programming.
- It is a programming analysis tool that is commonly used for planning the program logic.
- The name itself specifies "*Pseudo*" means *imitation*, "*Code*" means *the set of statements or instructions* written in a programming language.
- Pseudo code is also called "*Program Design Language (PDL)*".
- The pseudo code is written in *normal English* and cannot be understood by the computer.
- During the design of pseudo code, the designer or programmer can only concentrate the logic of the program *without worrying the syntax* of the instructions.
- There is *no one set way* to write pseudo code.

## Rules for writing Pseudocode

- The pseudo code is very clear unlike flowcharts. It is more individualistic; hence use the following rules for writing pseudo code.
- **Write one statement per line**
  - Break each task into smallest level of subtasks or subtasks, then convert each of those detail elements into one line in our pseudocode.
  - Each pseudocode line will represent one "action".
  - Each statement is written on a separate line.
- **Capitalize Initial Keywords**
  - The keywords in the pseudo code should be written in all capital letters.
  - In the statement  $\text{Total} = m_1 + m_2 + m_3$ , we do not put any word in all capitals because, we do not have a keyword.
- Primarily READ, WRITE, IF, WHILE and UNTIL are the keywords that must be written in capital letters.

- **Indent to show hierarchy**
  - Indentation is the process of showing the “boundaries” of the structure.
  - In case of sequence structure, we would not have any indentation.
  - But in the loop or selection structure, we must indicate the statement which is fall in the loop or selection.
- **End Multiline Structure**
  - The indentation is used to indicate what statements are executed within the structure.
  - When there is more than one line in the intention, it must be ended.
  - Example: ENDIF for IF statement.
  - It also helps to make sure that you know, where the loop or selection is completed or ended.
- **Keep statements language independent**
  - This rule refers to programming language.
  - Some programming languages have special capabilities that are not available in other languages.
  - So, we should design our program in such a way that it is free of those special requirements.

#### **Advantages:**

- It can be done easily in any word processor.
- It can be easily modified as compared to flowchart.
- It's implementation is very useful in structured design elements.
- It can be written easily.
- It can be read and understood easily.
- Converting a pseudo code to programming language is very easy to compare with converting a flowchart to programming language.

#### **Disadvantages:**

- It is not visual.
- We do not get a picture of the design.
- There is no standardized style or format, so one pseudo code may be different from another.
- For a beginner, it is more difficult to follow the logic or write pseudo code as compared to flowchart.

### Characteristics of a good pseudo code should:

- not be in a specific coding language
- draft the structure of your code
- be understandable to humans

## Flowchart

The flowchart is a *diagrammatic representation* that illustrates the sequence of operation to be performed to arrive at the solution. The operating instructions are placed in *symbols which are connected by arrows* to indicate the order of execution




### Importance for Flowchart

- A flow chart uses *different shapes box* to specify the several types of instructions.
- The program logic is *made very easy* through the flowchart that has *standardized meaning*.
- Thus every programmer uses the *same basic shapes*, so others can easily read and interpret the logic of the program.


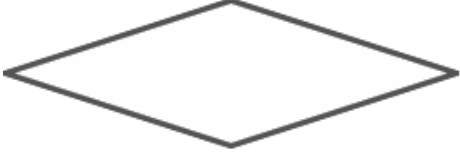
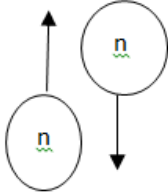
### Aim of Flowchart

- Program preparation can be *simplified* using the flowchart.
- Flowcharts are *easier* to understand at a glance.
- Flowchart are easy to *analyze and compare* various methods.
- Flowchart assist in *reviewing and debugging* of a program.
- Flowchart provide effective *programming documentation*

### Flow Chart Symbols:


Symbol	Name	What it does in the code
	Flow Lines Symbol	Arrows show connections between different parts of the code
	Terminal Symbol	Ovals show a start point or end point in the code
	Input / Output Symbol	Parallelograms show inputs and outputs (remember print is normally

		an input)
	Process	Rectangles show processes

	symbol	e.g. calculations (most things the computer does that does not involve an input, output or decision)
	Decision Symbol	Diamonds show a decision/conditional (this is normally if, else if/elif, while and for)
	Connectors Symbol	connector symbol is represented by a circle and a letter or digit is placed in the circle to specify the link.  Whenever a complex flowchart is to be drawn with the number of direction of flowlines is confusing or it can long as over the more than one page, in such situation, the connector symbol are used to connect the flowchart
	Functions	they represent a completed block of code that can be called form other parts of a program.

## Rules for Flowchart:

In order to produce a flowchart the designer must follow the rules given below.

- The *standard* symbols only be used.
- The arrowhead in the flowchart represents the *flow of direction* of control in the program.
- The usual direction of flow of procedure is from *top to bottom or left to right*.
-  Only *one flow line* come *to a process* symbol and only one flow line should out *from the process* symbol.



- Only *one flow line* should enter a *decision symbol* but *two or three* flow line one for each possible answer, should *leave the decision* symbol.



- Only one flow line is used in conjunction with terminal symbol.
- Flow line *should not cross* each other.
- Be *consistent* in using name and variable in the flowchart.
- Words in the flowchart symbol should be *common statement* and easy to understand.
- Keep flow chart as *simple* as possible.

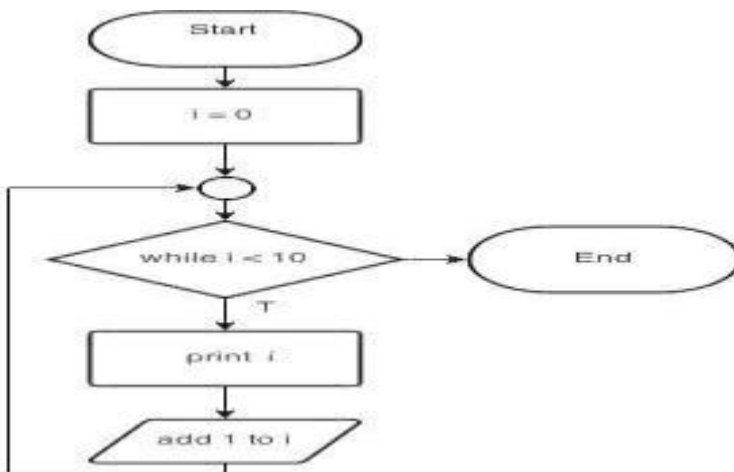
## Basic Design Structure in Flowchart

- Sequence Structure
- Selection Structure
- Loop Structure
  - Top Tested Loop
  - Bottom Tested Loop

## Tips to draw complex Flowcharts

- **For loops** normally repeat a nested block of code a set number of times or for a set range of data.
- **while loops** repeat a set block of code while a condition is true.
- The diamond symbol is used with **loops** to show the way the computer repeats a nested block of code then moves on to the next step.
- One of the easiest ways to think about it is that there are normally two paths the computer can go down when it reaches a loop:
  - the code in the loop
  - the code after the loop

For example:



**Note:** In this example there is no code after the loop so we represent this with the **end** symbol.

## Functions in flowcharts

- Another type of symbol that is important for more advanced programs is the symbol used for **functions**.
- **Functions** are sometimes also known as **modules** or **pre-defined processes**; they represent a completed block of code that can be called from other parts of a program.

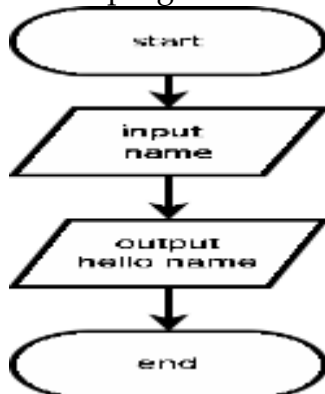
### Representing the contents of functions in flowcharts.

- A common way to represent the contents of functions is with separate unconnected flowcharts.
- The name of the function is normally added to the start symbol in these unconnected flowcharts.
- There may be times when you do not need to show the contents of every function in your flowchart.
- The example below shows a rough diagram of a program that makes use of functions that convert Fahrenheit to Celcius and compare the current temperature to the monthly average.
- This is a rough example only to show the idea of how you can represent functions in flowcharts; often the functions will be far more complex than the examples below.

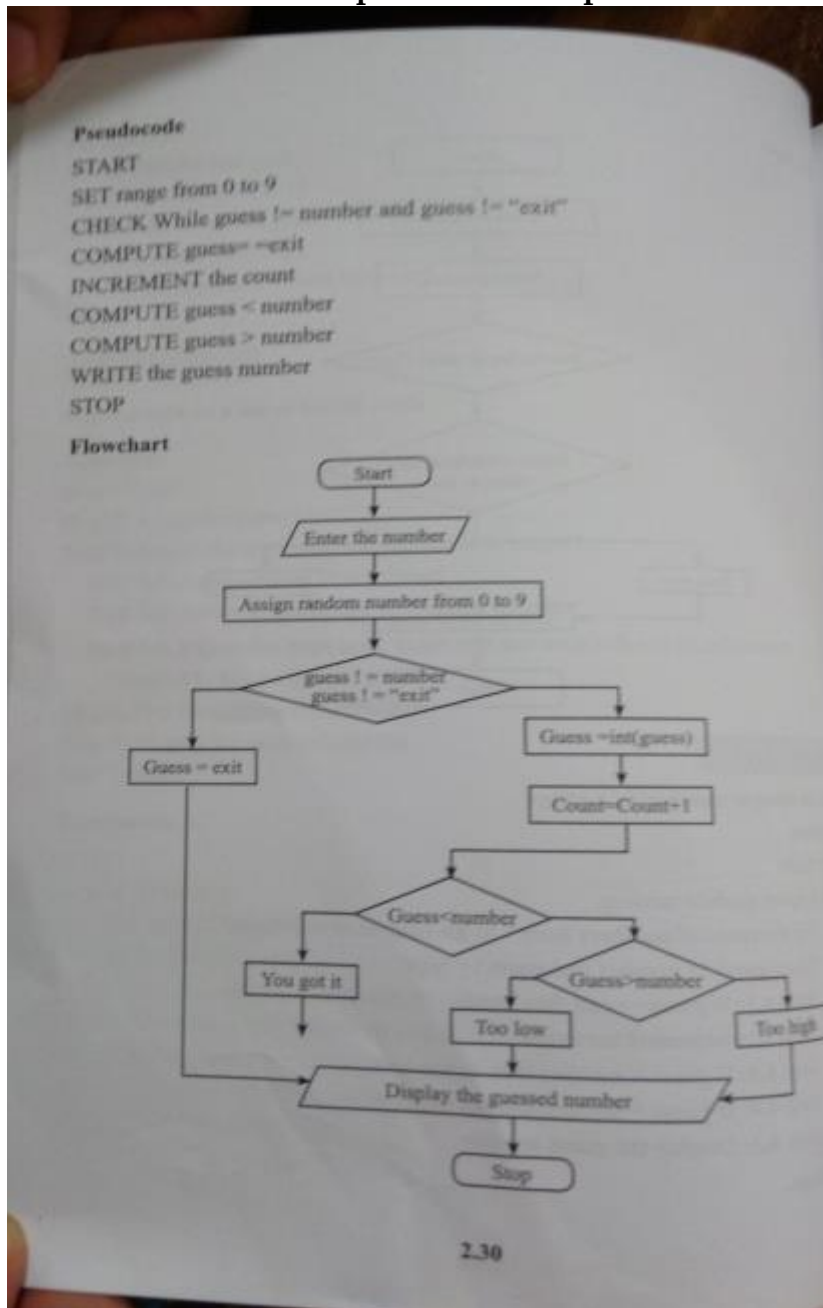
## Sample Flowcharts, Pseudo code and Python Code

### Example 1:

This program asks the user their name then says "Hello [Name]":



**Note:** This flow chart only shows ovals and parallelograms because the code only has start and end and one input and one output.



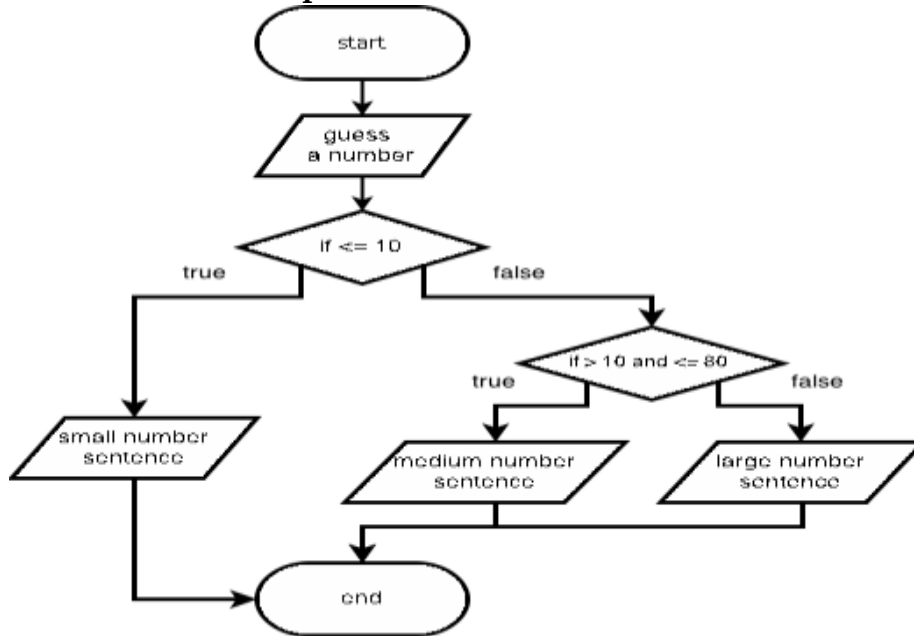
Pseudo code for Example 1:

Python code for Example 1:

## Example 2

This program asks the user to "Pick a number:" then prints 1 of 3 different outputs based on how big the number is.

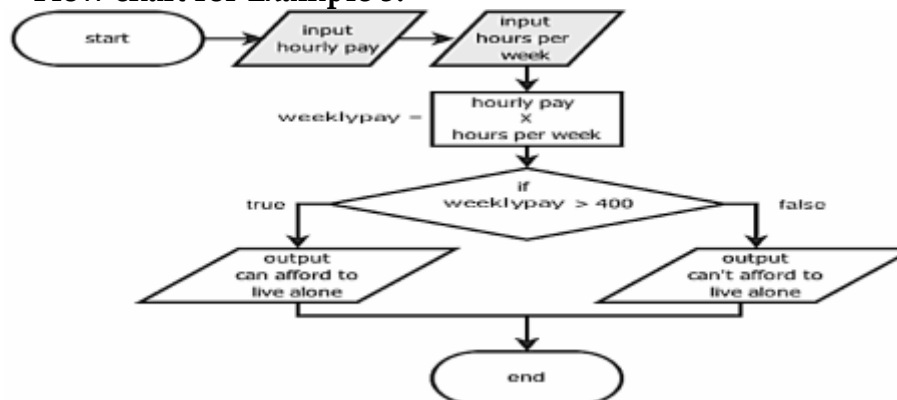
### Flow chart for Example 2



## Example 3

- This program asks the user how many hours a week they work and how much they earn per hour as two separate inputs. It then calculates how much money they earn a week and outputs two different sentences based on how much they earn.

### Flow chart for Example 3:



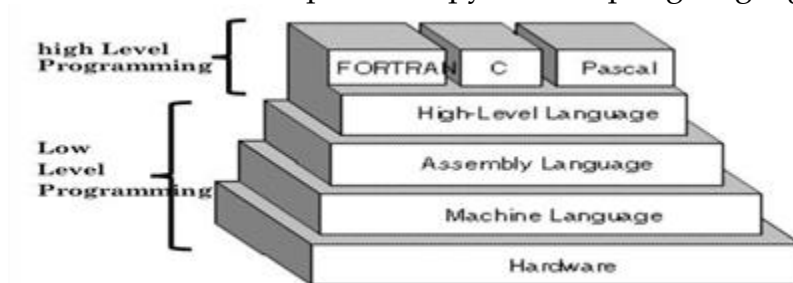
# Programming Language

- One can communicate with the computer through computer programming language.
- The programming languages are based on some syntactic and semantic rules.
- A vocabulary and set of *grammatical rules for instructing a [computer](#)* to perform specific tasks.
- The term *programming language* usually refers to [high-level languages](#), such as [BASIC](#), [C](#), [C++](#), [COBOL](#), FORTRAN, [Ada](#), and [Pascal](#).
- Each [language](#) has a *unique set of [keywords](#)* (words that it understands) and a *special [syntax](#)* for organizing [program](#) instructions.
- High-level programming languages, while simple compared to human languages.
- But are more complex than the languages the computer actually understands, called *machine languages*.
- Each different type of [CPU](#) has its own unique machine language.
- Lying between machine languages and high-level languages are languages called [assembly languages](#).
- Assembly languages are similar to machine languages, but they are much easier to program in because they allow a [programmer](#) to substitute names for numbers.
- Machine languages consist of numbers only.
- Regardless of what language you use, you eventually need to convert your program into machine language so that the computer can understand it.
- There are two ways to do this:
  - compile the program
  - interpret the program
- Every language has its strengths and weaknesses.
- For example, FORTRAN is a particularly good language for processing numerical data, but it does not lend itself very well to organizing large programs.
- Pascal is very good for writing well-structured and readable programs, but it is not as flexible as the C programming language.
- C++ embodies powerful object-oriented features, but it is complex and difficult to learn.

- The choice of which language to use depends on the type of computer the program is to run on.

### The programming languages are categorized as

- Integrated Programming language
- Functional programming language
- Compiled programming language
- Procedural programming language
- Scripting programming language
- Markup programming language
- Logic based programming language
- Concurrent programming language
- Object Oriented programming language.
- This book emphasis on python scripting language.



## What is python ?

- **Python** is an object-oriented, high level language.
- **It is** interpreted, dynamic and multipurpose programming language.
- Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.
- Python's syntax and *dynamic typing* with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas.
- Python supports *multiple programming pattern*, including object oriented programming, imperative and functional programming or procedural styles.
- Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.
- In python we don't need to use data types to declare variable because it is *dynamically typed* so we can write `a=10` to declare an integer value in a variable.

- Python makes the development and debugging *fast* because there is no compilation step included in python development.
- edit-test-debug cycle is very fast in python.

## Python Features:

- There are a lot of features provided by python programming language.

Features	Explanation
Easy to Use:	Python is easy to very easy to use and high level language. Thus it is programmer-friendly language.
Expressive Language:	Python language is more expressive. The sense of expressive is the code is easily understandable.
3) Interpreted Language:	Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform language:	Python can run equally on different platforms such as Windows, Linux, Unix , Macintosh etc. Thus, Python is a portable language.
5) Free and Open Source:	Python language is freely available( <a href="http://www.python.org">www.python.org</a> ).The source-code is also available. Therefore it is open source.
6) Object-Oriented language:	Python supports object oriented language. Concept of classes and objects comes into existence.
7) Extensible:	It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.
8) Large Standard Library	Python has a large and broad library.
9) GUI Programming:	Graphical user interfaces can be developed using Python.
10) Integrated:	It can be easily integrated with languages like C, C++, JAVA etc.

## Python History:

- 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.

- *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 programming languages like ABC language, Modula-3.

## Python Version:

Python programming language is being updated regularly with new features and support. There are a lot of updation in python versions, started from 1994 to current date. 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 Applications

- Python as a whole can be used in any sphere of development.
- There are several such applications which can be developed using Python.
- Let us see what are the major regions where Python proves to be handy.



Application:	Features
Console Based Application	Python can be used to develop console based applications. For example: <b>IPython</b> .
Audio or Video based Applications	Python proves handy in multimedia section. Some of real applications are: TimPlayer, cplay etc.
3D CAD Applications	Fandango is a real application which provides full features of CAD.
Web Applications	Python can also be used to develop web based application. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.
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.
Applications for Images	Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

### Python Example

- Python code is simple and easy to run.
- Here is a simple Python code that will print "Welcome to Python"
- A simple python example is given below.

### Explanation:

- Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.
- A variable is defined named "a" which holds "Welcome To Python".
- "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable.
- Therefore, the output "Welcome To Python" is produced.

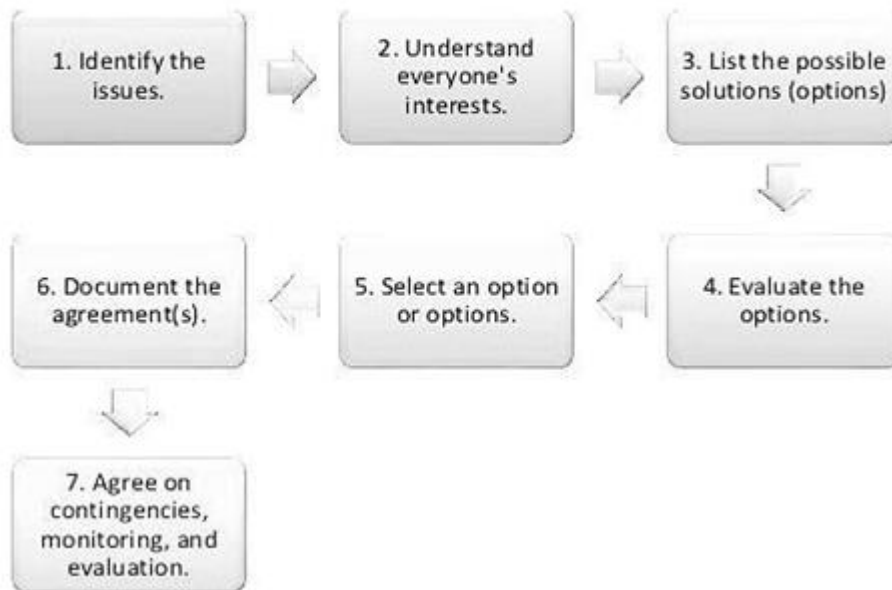
In python 3.4 version, you need to add parenthesis () in a string code to print it.

# Algorithmic problem solving

- Algorithmic problems are problems where the solution involves the design of an algorithm.
- Algorithmic problem solving is about the formulation and solution of such problems.
- “algorithmic-problem solving”; this means solving problems that require the formulation of an algorithm for their solution.
- On the other hand, it can be read as “algorithmic problem-solving”; this means exploiting what has been learned from the experience of developing computer software over the last 50 years that has helped us to hone our problem-solving skills.
- The formulation of algorithms has always been an important element of problem-solving.
- Algorithms are the end-product of the problem-solving process and it is imperative that they are made explicit and are carefully studied.
- A particular focus has been on so-called “correct-by-construction” design techniques.
- These techniques involve first formulating a mathematical specification of what the algorithm is required to compute and then developing the algorithm in a way that guarantees that it meets its specification.
- The process of formulating a specification involves abstraction from so-called “real- world” problems; it helps to eliminate ambiguity and clarify the true nature of the problem being tackled.
- Correct-by-construction is a systematic way of designing algorithms using the specification as a blueprint;
- Roland Backhouse’s book on Algorithmic Problem Solving introduces the principles underlying correct-by-construction design techniques.
- Problem solving is not easy.
- There are a number of mental processes at work during problem-solving. These include:
  - Perceptually recognizing a problem
  - Representing the problem in memory
  - Considering relevant information that applies to the current problem
  - Identify different aspects of the problem

- Labeling and describing the problem
- According to by [Tim Hicks](#) published in *The Business Journal of Sonoma/Marin*. There are seven-steps for an effective problem-solving process.
- **Identify the issues.**
  - Be clear about what the problem is.
  - Remember that different people might have different views of what the issues are.
  - *Separate the listing of issues from the identification of interests (that's the next step!).*
- **Understand everyone's interests.**
  - This is a critical step that is usually missing.
  - Interests are the needs that you want satisfied by any given solution. We often ignore our true interests as we become attached to one particular solution.
  - The best solution is the one that satisfies everyone's interests.
  - This is the time for active listening. Put down your differences for awhile and listen to each other with the intention to understand.
  - *Separate the naming of interests from the listing of solutions.*

## Seven Steps for Effective Problem Solving in the Workplace



- **List the possible solutions (options)**
  - This is the time to do some brainstorming. There may be lots of room for creativity.
  - *Separate the listing of options from the evaluation of the options.*
- **Evaluate the options.**
  - What are the pluses and minuses? Honestly!

- *Separate the evaluation of options from the selection of options.*
- **Select an option or options.**
  - What's the best option, in the balance?
  - Is there a way to "bundle" a number of options together for a more satisfactory solution?
- **Document the agreement(s).**
  - Don't rely on memory.
  - Writing it down will help you think through all the details and implications.
- **Agree on contingencies, monitoring, and evaluation.**
  - Conditions may change. Make contingency agreements about foreseeable future circumstances (If-then!).
  - How will you monitor compliance and follow-through?
  - Create opportunities to evaluate the agreements and their implementation.

## Simple strategies for developing algorithms

- The strategies for developing an algorithm is depended one the problem.
- The algorithm development consists of many steps.
  - Steps in development of Algorithms.
  - Problem definition
  - Development of a model
  - Specification of Algorithm
  - Designing an Algorithm
  - Checking the correctness of Algorithm
  - Analysis of Algorithm
  - Implementation of Algorithm
  - Program testing
  - Documentation Preparation
- The most common strategies that are used for algorithm developments are
  - 1 .Iteration
  - 2. Recursion.

## Iteration.

- Computers are great at performing some task a million times very fast.
- Iteration is the fancy term for repeating some programming commands multiple times.
- Performing a task n number of times refers to iteration.  
Python provides the **while** construct to iterate over commands:
  - As a simple example, suppose we want to print the numbers 0 through 4. Without using iteration, we could type the following in a Python program.
  - This will not be an efficient solution if we wanted to print the first thousand or million numbers.
  - Hence we go for Iteration.
  - We set up a **counter variable** to tell us how many times we've done something.
  - In this case, we start the variable at 0, and add one to it until it gets to 5.
  - Note that you must put a colon after the condition, and you must indent the commands that belong within the while loop.
  - The above code will execute the print command
  - The 'number=number+1' statement iterates over and over until the while the condition 'number<5' remains true.

## Recursion

- **Recursion** means “defining something in terms of itself” usually at some smaller scale, perhaps multiple times, to achieve your objective.
- Programming languages generally support **recursion**, which means that, in order to solve a problem, functions can *call themselves* to solve smaller sub problems.
- **Recursion** is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get to a small enough problem that it can be solved trivially.
- In programming recursion involves a function calling itself.
- Recursion allows us to write elegant solutions to problems that may otherwise be very difficult to program.

## Calculating the Sum of a List of Numbers

```
def listsum(numList):  
    theSum = 0  
    for i in numList:  
        theSum = theSum  
        + i
```

```
    return theSum
print(listsum([1,3,5,7,9]
))
```

- In the above example the function listsum calls itself until the condition is true

## The Three Laws of Recursion

- Like the robots of Asimov, all recursive algorithms must obey three important laws:
  - A recursive algorithm must have a **base case**.
  - A recursive algorithm must change its state and move toward the base case.
  - A recursive algorithm must call itself, recursively.

### Law 1:

- First, a base case is the condition that allows the algorithm to stop recursing.
- A base case is typically a problem that is small enough to solve directly.
- In the listsum algorithm the base case is a list of length 1.

### Law 2:

- To obey the second law, we must arrange for a change of state that moves the algorithm toward the base case.
- A change of state means that some data that the algorithm is using is modified.
- In the listsum algorithm our primary data structure is a list, so we must focus our state-changing efforts on the list.
- Since the base case is a list of length 1, a natural progression toward the base case is to shorten the list. This is exactly what actually done in the above example.

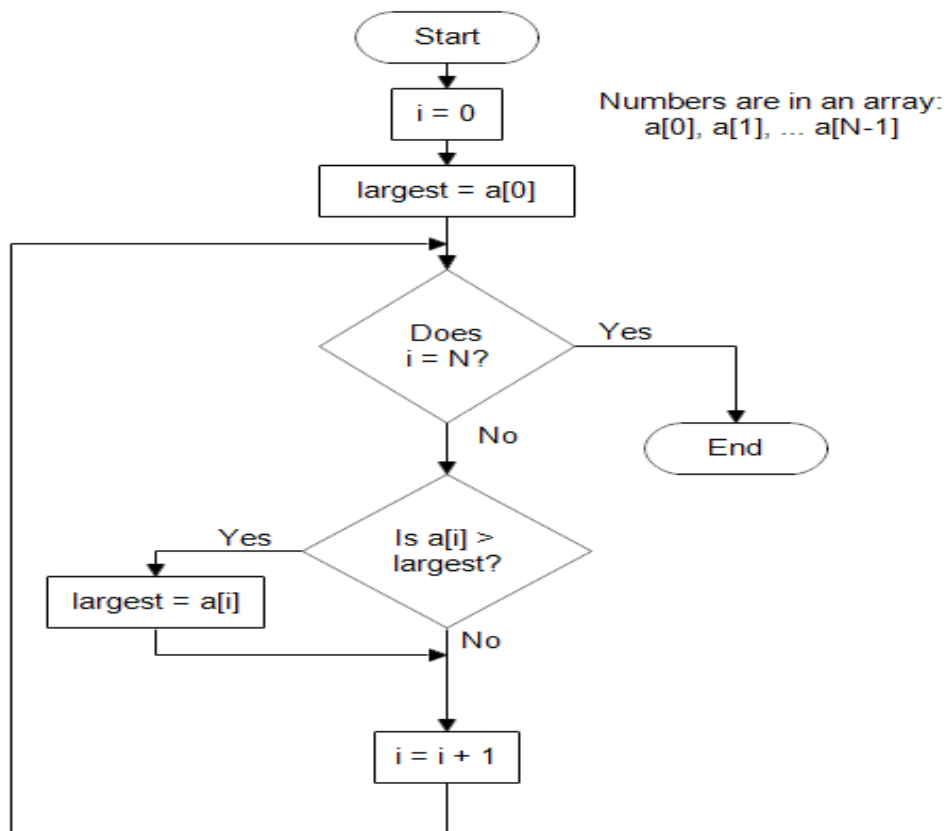
### Law 3:

- The final law is that the algorithm must call itself.
- This is the very definition of recursion.

# Illustrative example

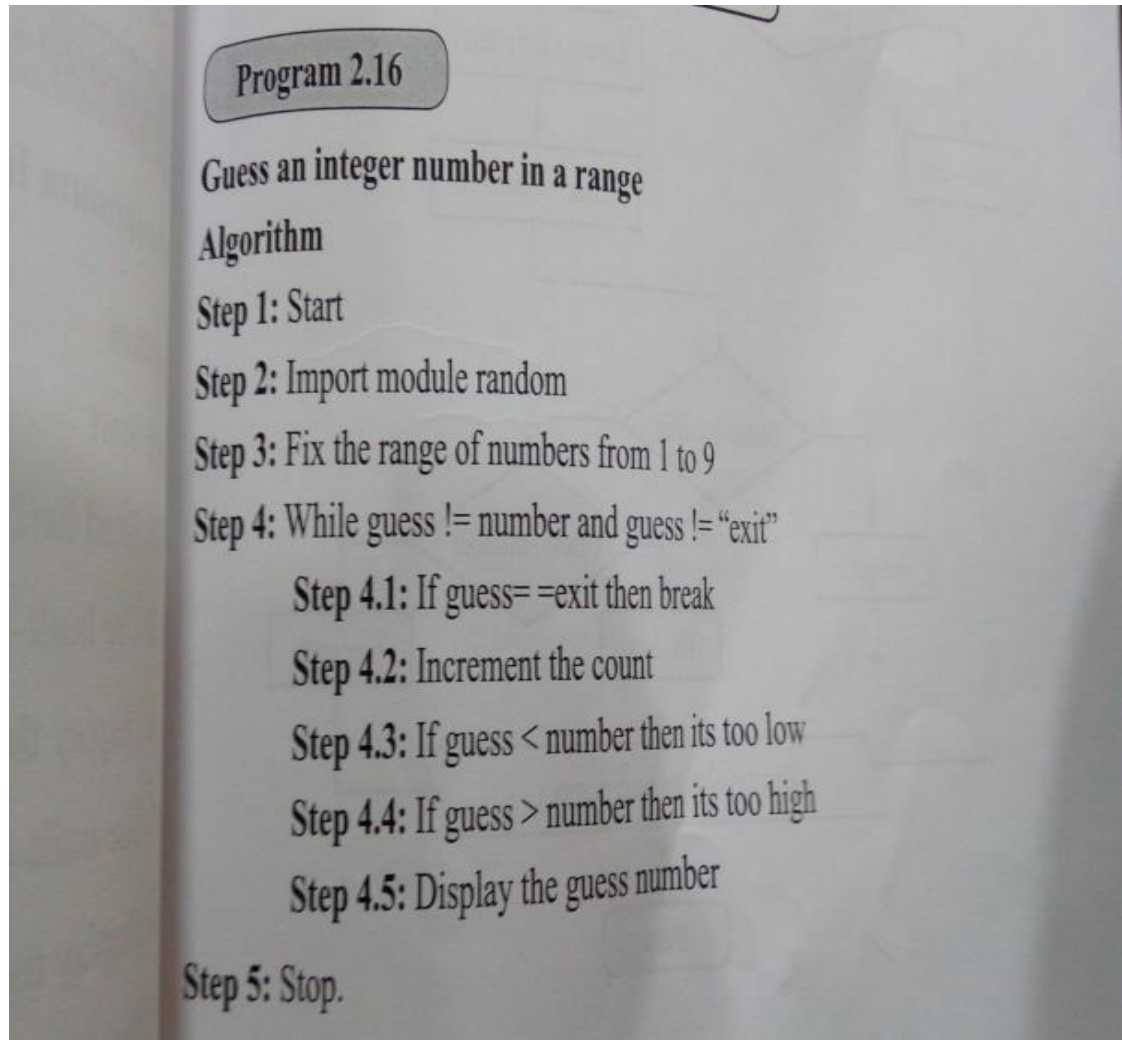
Find minimum and maximum in a list

## Finding the Largest Number in a List of Numbers



## Insertion Sort

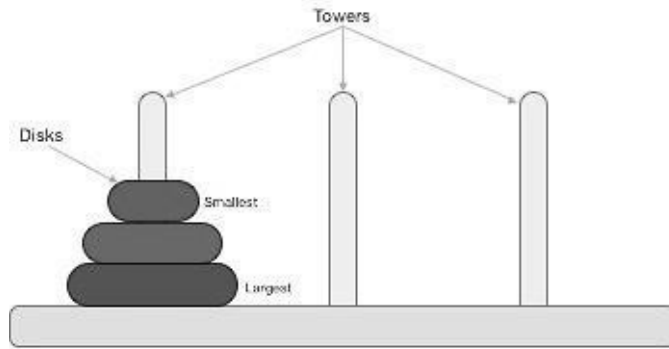
### Guess an integer number in a range



## Tower of Hanoi

- Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted –





- These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one.

### Rules

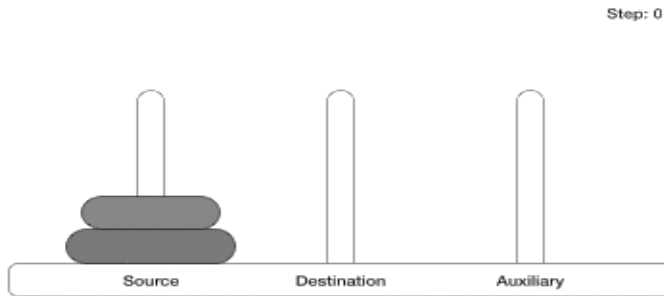
- The mission is to move all the disks to some another tower without violating the sequence of arrangement.
- A few rules to be followed for Tower of Hanoi are –
  - Only one disk can be moved among the towers at any given time.
  - Only the "top" disk can be removed.
  - No large disk can sit over a small disk.
- Tower of Hanoi puzzle with  $n$  disks can be solved in minimum  $2^n - 1$  steps.

### Algorithm

- To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say  $\rightarrow 1$  or  $2$ .
- We mark three towers with name, **source**, **destination** and **aux** (only to help moving the disks).
- If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks –

- First, we move the smaller (top) disk to aux peg.
- Then, we move the larger (bottom) disk to destination peg.
- And finally, we move the smaller disk from aux to destination peg.



- So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks.
- We divide the stack of disks in two parts. The largest disk ( $n^{\text{th}}$  disk) is in one part and all other ( $n-1$ ) disks are in the second part.
- Our ultimate aim is to move disk **n** from source to destination and then put all other ( $n-1$ ) disks onto it.
- We can imagine to apply the same in a recursive way for all given set of disks. The steps to follow are –

A recursive algorithm for Tower of Hanoi can be driven as follows –

## Algorithm

### Part A

#### • Define Algorithm.

An algorithm (pronounced AL-go-rith-um) is a *procedure* or formula for solving a problem, based on conducting a *sequence of specified actions*.

#### • What are the characteristics of Algorithm?

- In Algorithms each and every instruction *should be precise* and *clear-cut*.
- The instruction in an algorithm should *not be repeated* infinitely
- The algorithm make sure that it *will ultimately be terminated*
- Algorithm should be *written in sequence*
- It looks like *normal English*
- The desired result should be obtained only after the algorithm terminated

#### • What are the primary factors needed to judge the quality of the algorithm?

- Time

- Memory
- Accuracy
- Sequence
- Generality
- **What are the ways to represent an algorithm?**
  - Normal English
  - Flowchart
  - Pseudo code
  - Decision table
  - Program
- **On which the relative performance of the algorithm depends?**
  - The relative performance of the algorithms might depend on characteristics of the hardware
  - Relative performance might depend on the details of the dataset.
  - Relative performance also depends on the size of the problem
- **What are the building blocks of algorithm?**
  - Sequence
  - Selection
  - Iteration

- **Define Identifiers.**

Identifier is the *name* given to entities like *class, functions, variables* etc. in Python. It helps *differentiating* one entity from another. Identifiers are the names given to the

fundamental building blocks in a program. Some identifiers are *built-in*, and others can be *created* by the programmer. An identifier is a long sequence of *characters and numbers*. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_).

- **Define Literals. What are its types?**

- Literals can be defined as a data that is given in a variable or constant. Python support the following literals:
  - String literals

- Numeric literals
- Boolean literals
- Special literals

- **Define List.**

List contain items of different data types. Lists are mutable i.e., modifiable. The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List. Value stored in a List can be retrieved using the slice operator([] and [:]). The plus sign (+) is the list concatenation and asterisk(\*) is the repetition operator.

- **Define Pseudocode.**

Pseudo code is an *ordered version of your code* written for human understanding rather than machine understanding. Pseudo code is a *formal design tool*, developed with the structured programming. The name itself specifies "*Pseudo*" means *imitation*, "*Code*" means *the set of statements or instructions* written in a programming language. Pseudo code is also called "*Program Design Language (PDL)*".

- **Mention the rules for writing Pseudocode?**

- Write one statement per line
- Capitalize Initial Keywords
- Indent to show hierarchy
- End Multiline Structure
- Keep statements language independent

- **What are the advantages of Pseudocode**

- It can be done easily in any word processor.
- It can be easily modified as compared to flowchart.
- It's implementation is very useful in structured design elements.
- It can be written easily.
- It can be read and understood easily.
- Converting a pseudo code to programming language is very easy to compared with converting a flowchart to programming language.

- **What are the disadvantages of Pseudocode**

- It is not visual.
- We do not get a picture of the design.

- There is no standardized style or format, so one pseudo code may be different from another.
- For a beginner, it is more difficult to follow the logic or write pseudo code as compared to flowchart.

- **Define Flowchart.**

The flowchart is a *diagrammatic representation* that illustrates the sequence of operation to be performed to arrive at the solution. The operating instructions are pleased in *symbols which are connected by arrows* to indicate the order of execution.

- **Mention the aims of flowchart.**

- Program preparation can be *simplified* using the flowchart
- Flowchart are *easier* to understand at a glance
- Flowchart are easy to *analyze and compare* various methods
- Flowchart assist in *reviewing and debugging* of a program
- Flowchart provide effective *programming documentation*

- **Mention the basic design structure in Flowchart.**

- Sequence Structure
- Selection Structure
- Loop Structure
  - Top Tested Loop
  - Bottom Tested Loop

- **What are the categories of programming language?**

- Integrated Programming language
- Functional programming language
- Compiled programming language
- Procedural programming language
- Scripting programming language
- Markup programming language
- Logic based programming language
- Concurrent programming language
- Object Oriented programming language.

- **Define Python.**

**Python** is an object-oriented, high level language. **It is** interpreted, dynamic and multipurpose programming language. Python is *easy to learn* yet powerful and versatile

scripting language which makes it attractive for Application Development. Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.

- **What are the features of python?**

- Easy to Use
- Expressive Language
- Interpreted Language
- Cross-platform language
- Free and Open Source
- Object-Oriented language
- Extensible
- Large Standard Library
- GUI Programming
- Integrated

- **What are the applications of python?**

- Console Based Application
- Audio or Video based Applications
- 3D CAD Applications
- Web Applications
- Enterprise Applications
- Applications for Images

- **What are the steps for effective problem solving?**

- Identify the issues.
- Understand everyone's interests
- List the possible solutions (options)
- Evaluate the options.
- Select an option or options.
- Document the agreement(s).

- Agree on contingencies, monitoring, and evaluation.
- **Mention the simple strategies for developing algorithms?**

- Problem definition
- Development of a model
- Specification of Algorithm
- Designing an Algorithm
- Checking the correctness of Algorithm
- Analysis of Algorithm
- Implementation of Algorithm
- Program testing
- Documentation Preparation

- **Define Iteration.**

Computers are great at performing some task a million times very fast. Iteration is the fancy term for repeating some programming commands multiple times. Performing a task n number of times refers to iteration

- **Define Recursion.**

**Recursion** means “defining something in terms of itself” usually at some smaller scale, perhaps multiple times, to achieve your objective. **Recursion** is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get to a small enough problem that it can be solved trivially. In programming recursion involves a function calling itself.

- **What are the three laws of Recursion?**

- A recursive algorithm must have a **base case**.
- A recursive algorithm must change its state and move toward the base case.
- A recursive algorithm must call itself, recursively

## **Part B.**

- Explain Algorithms in details.
- What are the building blocks of algorithms. Explain each in detail.
- Define notation . what are the notations used in programming.
- Explain pseudo code in detail
- Explain flow chart in detail

- What is meant by programming language
- Discuss algorithmic problem solving
- Discuss the simple strategies for developing algorithms
- Define iteration. Explain in detail.
- Define recursion. Explain in detail.
- Illustrative problems: Write the algorithm, pseudo code and flowchart for find minimum in a list
- Illustrative problems: Write the algorithm, pseudo code and flowchart for insert a card in a list of sorted cards
- Illustrative problems: Write the algorithm, pseudo code and flowchart for guess an integer number in a range,
- Illustrative problems: Write the algorithm, pseudo code and flowchart for Towers of Hanoi.



### Program 2.16

Guess an integer number in a range

Algorithm

Step 1: Start

Step 2: Import module random

Step 3: Fix the range of numbers from 1 to 9

Step 4: While guess  $\neq$  number and guess  $\neq$  "exit"

Step 4.1: If guess == exit then break

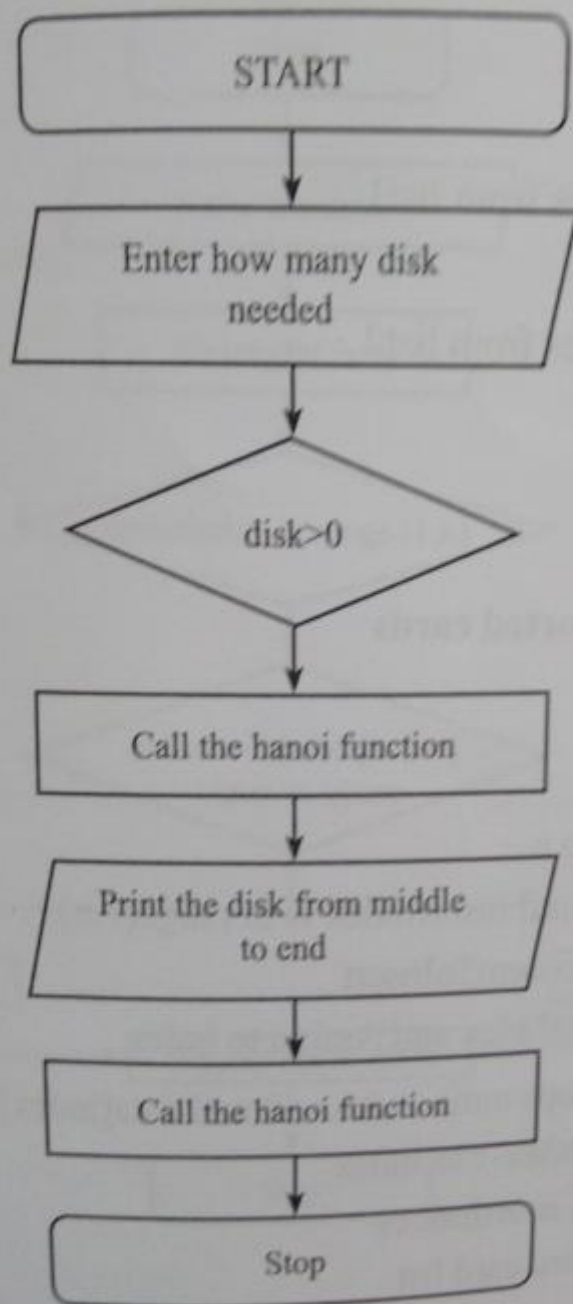
Step 4.2: Increment the count

Step 4.3: If guess < number then its too low

Step 4.4: If guess > number then its too high

Step 4.5: Display the guess number

Step 5: Stop.



### Pseudocode

START  
SET range from 0 to 9  
CHECK While guess  $\neq$  number and guess  $\neq$  "exit"  
COMPUTE guess = -exit  
INCREMENT the count  
COMPUTE guess < number  
COMPUTE guess > number  
WRITE the guess number  
STOP

### Flowchart

