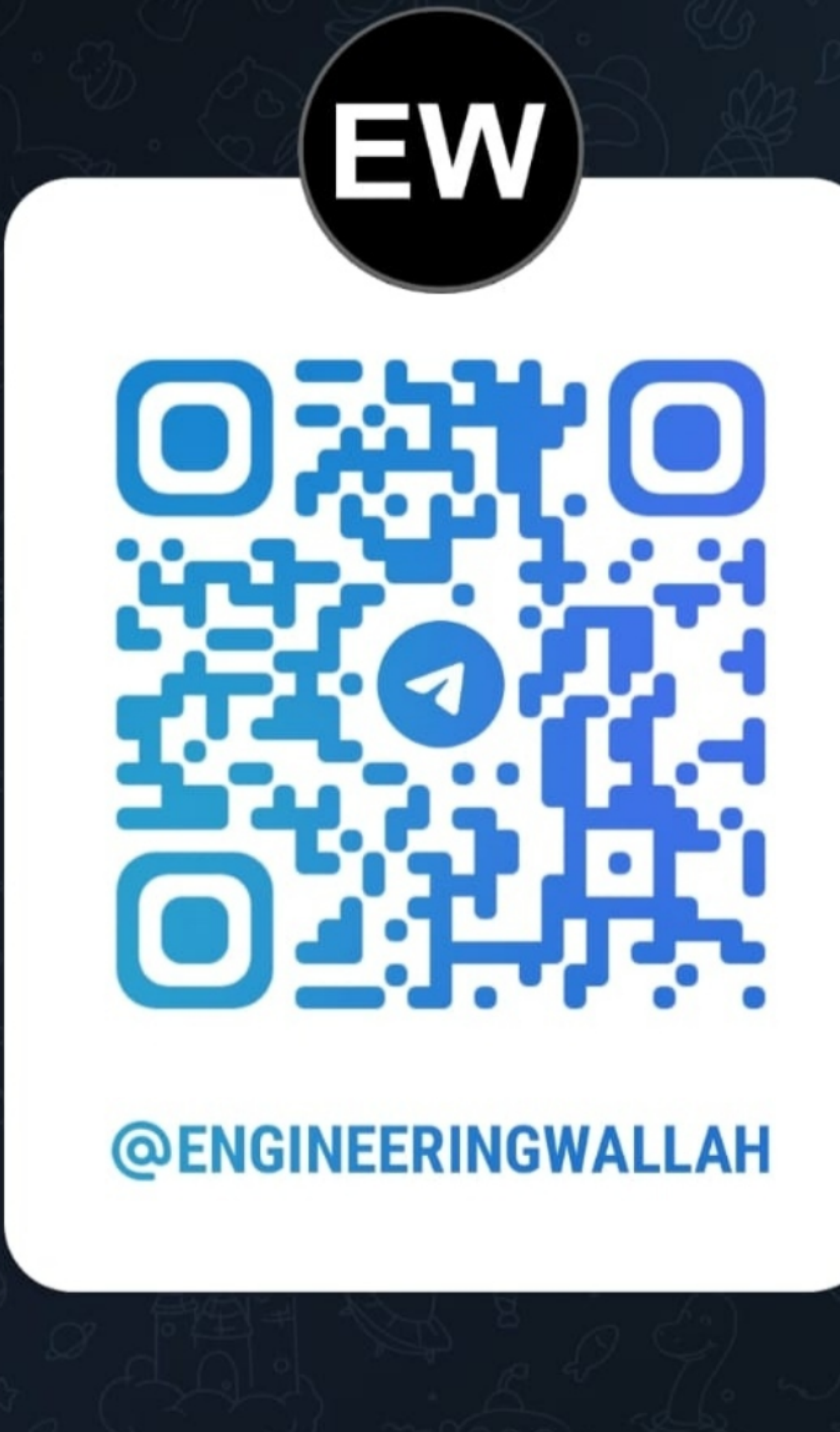

EW
@ENGINEERINGWALLAH

Programming and Problem Solving

Prof. M.R.Deore

DEPARTMENT OF E&TC, SCOE, VADGAON (BK), PUNE

UNIT: I	Problem Solving, Programming and Python Programming	Lectures: 07 Hrs

- **General problem solving concepts :**
- Problem solving in everyday life,
- types of problems, problem solving with computers,
- difficulties with problem solving
- problem solving aspects, top down design. Problem Solving Strategies
- **Program Design Tools:** Algorithms, Flowcharts and Pseudo-codes, implementation of algorithms.
- **Basics of Python Programming:** Features of Python, History and Future of Python, Writing and executing Python program, Literal constants, variables and identifiers, Data Types, Input operation, Comments, Reserved words, Indentation, Operators and expressions, Expressions in Python.

A problem is simply the difference between what you have and what you want. It may be a matter of getting something, of getting rid of something, of avoiding something, or of getting to know what you want.

"How you think about a problem is more important than the problem itself – so always → think positively."

~ Norman Vincent Peale

?

"If I have one hour to save the world I would spend fifty-five minutes defining the problem and only five minutes finding the solution.

-Einstein

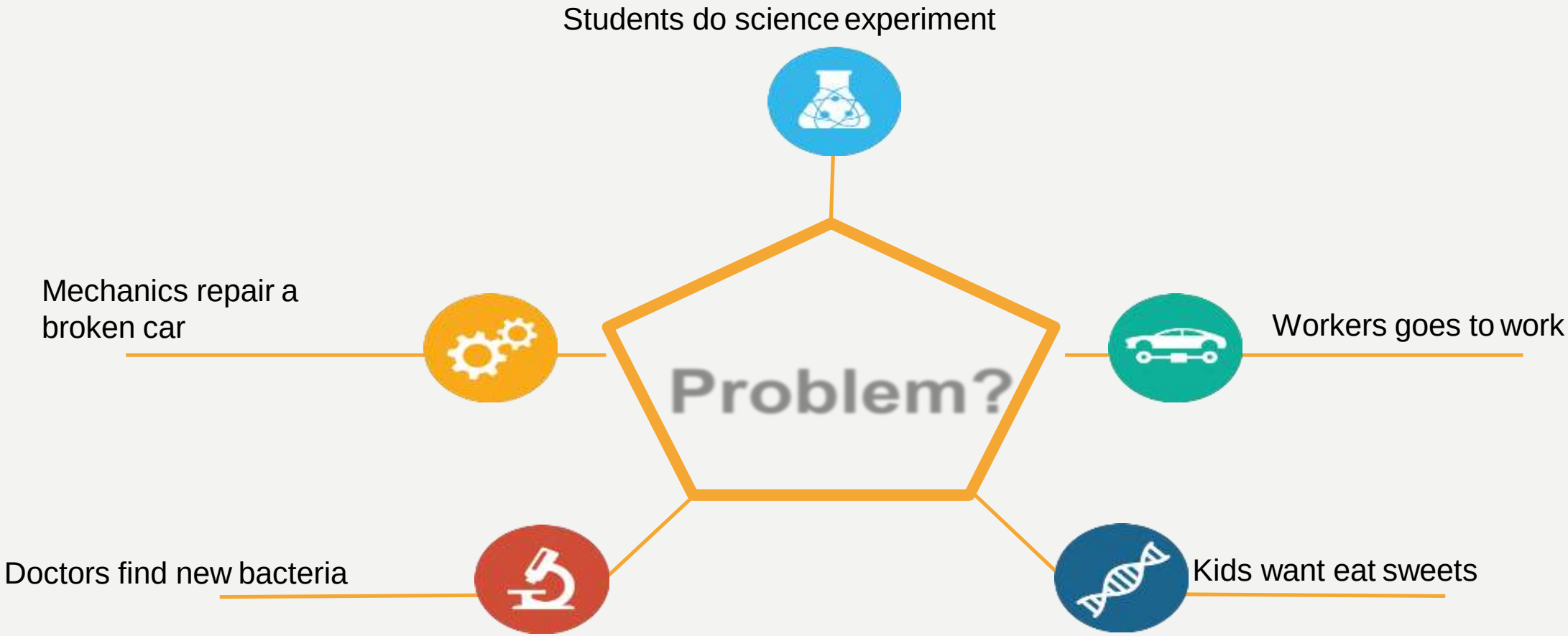
WHAT IS PAID?

A logical problem solving process can be done through some steps called

"PAID"

- **P**roblem Statement
- **A**nalyze the problem in detail
- **I**dentify likely causes
- **D**efine actual cause(s)

EXAMPLE PROBLEM IN DAILY LIFE



HOW TO SOLVE THE PROBLEM?

STRATEGIES

Ask questions!

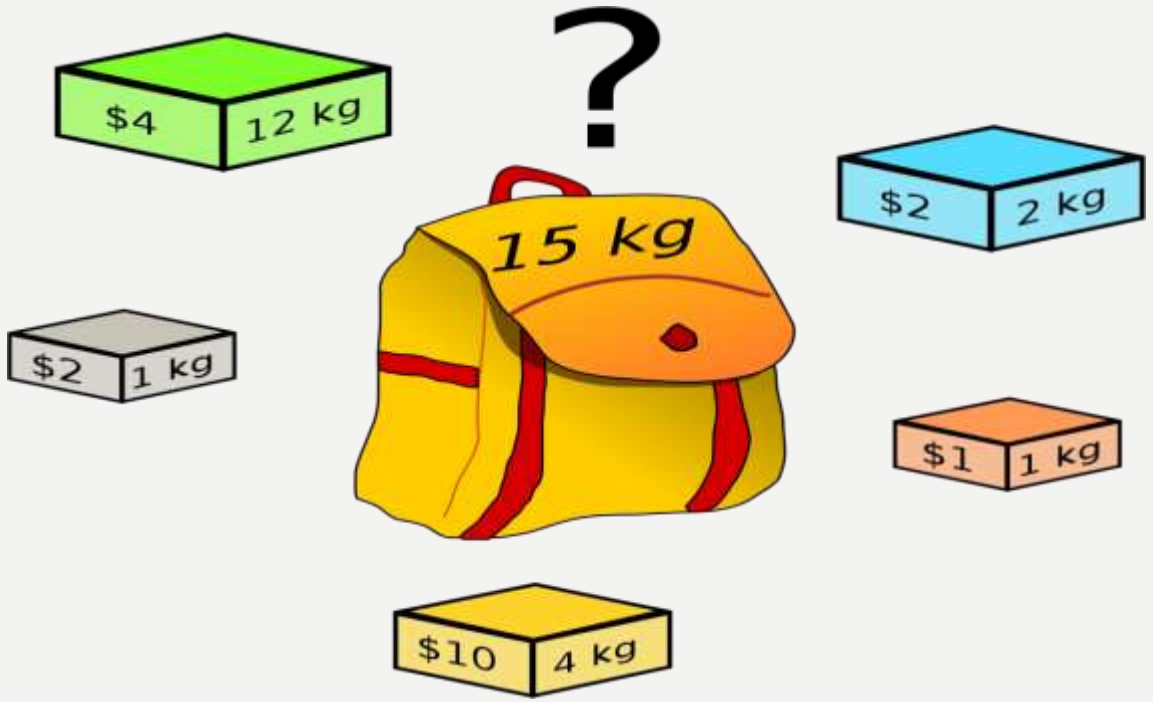
- *What do I know about the problem?*
- *What is the information that I have to process in order the find the solution?*
- *What does the solution look like?*
- *What sort of special cases exist?*
- *How will I recognize that I have found the solution?*

PROBLEM SOLVING DEFINITION

A Systematic approach to defining problem(question or situation that presents uncertainty or difficulty) and creating a vast number of possible solutions without judging these solutions.

PROBLEM SOLVING DEFINITION

A Systematic approach to defining problem and creating a vast number of possible solutions without which will gives maximum accuracy.



Problem Solving in Everyday Life

There are six step in problem solving:

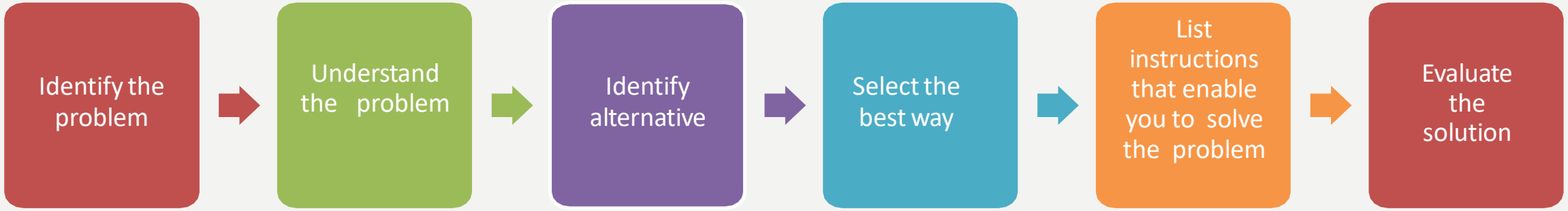
PROBLEM SOLVING IN EVERYDAY LIFE

Can I skip the step?

According to Sprankle and Hubbard (2012), if the six step not completed well, the result may be less than desired

EXAMPLE PROBLEMS…

Baking a **cake** according to certain specifications, input available are the ingredients **(such as eggs, flour, milk …etc.),** then followed by activities or procedures that should be done **sequentially**, taking into consideration that any mistake happens by doing any procedure before the other, results in an unsuitable and undesirable cake.



GROUP ACTIVITIES

In a group of 4 persons, think a problems in your daily life at home, office, park, school, shopping mall or wherever your go. Please explain how you solve the problem using the six steps of problem solving.

- Key Point: There is always a benefit to solving problems. Remember that if you solve a problem, even a difficult one, it is one less thing to worry about, and one less problem on your problem list!

TYPES OF PROBLEMS

- Problems do not always have straightforward solutions

1. ALGORITHMIC SOLUTIONS

   - Problems that can be solved with a series of actions
   - Ex: Baking a cake
   - The solution will be the same each time the algorithm is followed
   - The solution of a problem can be reached by completing the actions in steps. These steps are called the algorithm.
   - So once we eliminate the alternatives and have chosen one best methods, these steps are called algorithm.

TYPES OF PROBLEMS

2. HEURISTIC SOLUTIONS

- Problem which couldn't be solved through a direct set of steps.
  - Example: Baking a delicious cake, expanding a company
- These Problem with solutions require reasoning built on knowledge and experience, and a process of trial and error.
- The results may not produce the same results each time the algorithm is executed
- Usually use in the Artificial Intelligent software or Expert system software.

COMBINATION OF BOTH

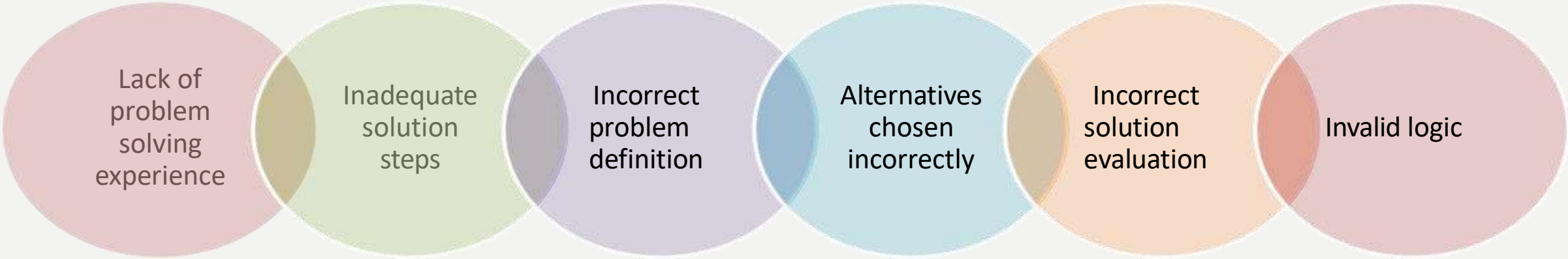
- Most problems require a combination of algorithmic and heuristic solutions
  - –Example:
  - –Repairing a car
  - –Driving a car
  - –To win in a computer game

PROBLEM SOLVING WITH COMPUTERS

Definitions by Sprankle & Hubbard (2012):

SOLUTION	RESULT	PROGRAM
instructions followed to produce best result	Outcome OR completed computer- assisted answer	Set of instructions for solution using computer language

DIFFICULTIES WITH PROBLEM SOLVING



PROBLEM SOLVING WITH COMPUTERS

- Computers are built to deal with algorithmic solutions, which are often difficult or very time consuming for humans.
- People are better than computers at developing heuristic solutions.

- Number of problems in our daily life.
- Suppose we have to calculate Simple Interest.
- Suppose we have to prepare a mark sheet.

- A computer is a DUMB machine.
- A computer cannot do anything alone without
- software i.e. Program

PROBLEM SOLVING WITH COMPUTERS

- A software is a set of programs written to solve a particular problem
- Software is a set of instructions on the basis of which computer gives output/result.
- If the instructions are not correct, the computer gives wrong result.

Never Ever Forget

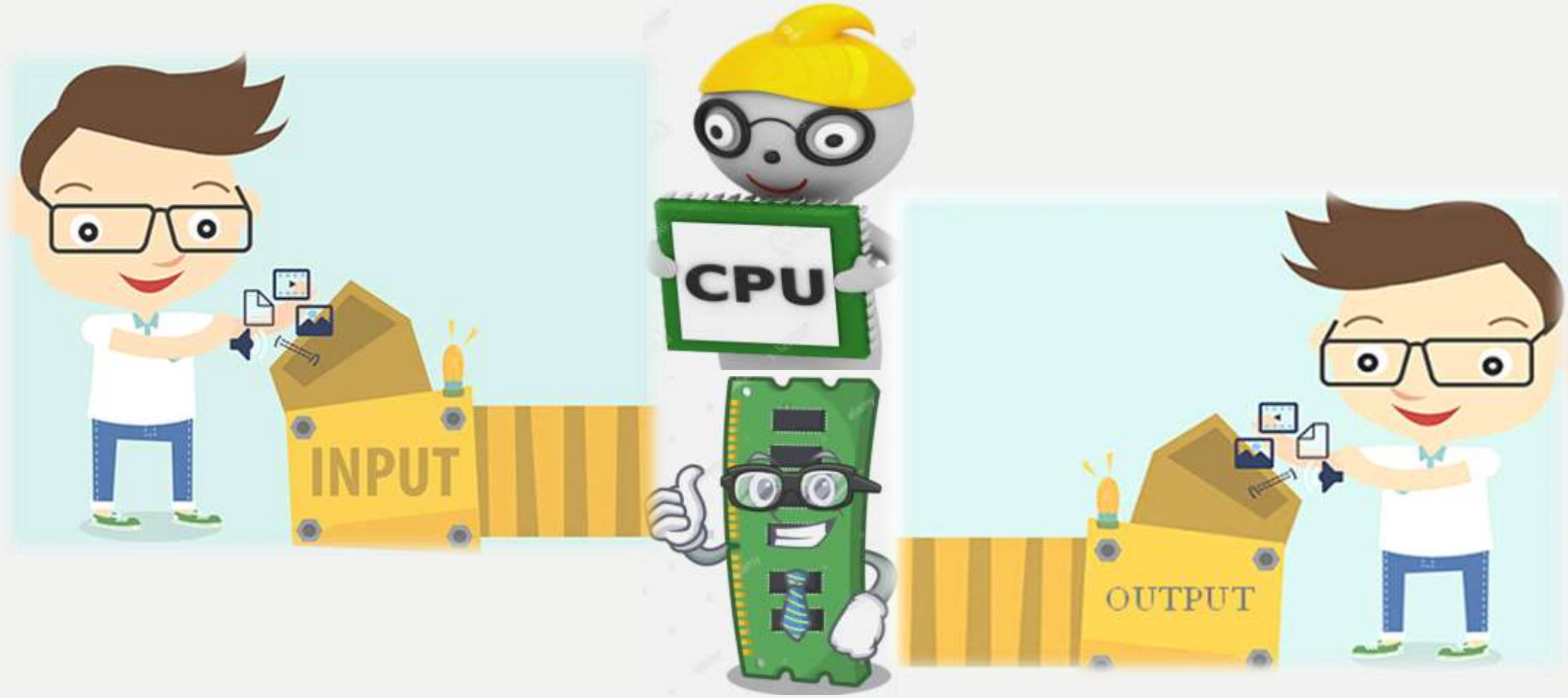
- Just writing code is not sufficient to solve a Problem.
- Program must be planned before coding in any computer language.
- There are many activities to be done before and after writing code

PROBLEM SOLVING WITH COMPUTERS

- Basic Definitions

1. Solution ⇔ instructions followed to produce best result
2. Result ⇔ outcome, computer-assisted answer
3. Program ⇔ The set of instructions for solution using computer language

CONCEPT WITH COMPUTER



CONCEPT WITH COMPUTER

- Problem that can be solve with computers generally consist of three:
  - Computational – problem with mathematical processing
  - Logical – problem involving with relational or logical processing. This is kind of processing involve in decision making.
  - Repetitive – problem involving repeating a set of mathematical or logical instructions.

STAGES WHILE SOLVING A PROBLEM USING COMPUTER

1. Problem Analysis
2. Algorithm Development
3. Flowcharting
4. Coding
5. Compilation and Execution
6. Debugging and Testing
7. Documentation

DIFFICULTIES WITH PROBLEM SOLVING

- People have many problems with problem solving
- Incorrect problem definition
- Lack of problem solving experience
- Afraid to make decisions
- People go through one or more steps inadequately
- Alternatives chosen incorrectly
- Invalid logic
- Problem solving on computer
  - Difficult task of writing instructions
  - Computer has specific system of communication

PROBLEM SOLVING ASPECTS

- There is no universal method for problem solving
- Phases
  - Problem Definition Phase
  - Getting started on a problem
  - The use of Specific Examples
  - Similarities among problems
  - Working backwards from the solution
  - General Problem Solving Strategies

PROBLEM SOLVING ASPECTS

- Problem Definition Phase
  - What must be done?
  - We must try to extract set of precisely defined tasks.
  - E.g. Finding Square root, Greatest Common Divisor
- Getting Started on a Problem
  - Sometimes, even after problem definition people do not know where to start?
  - What can we do?
- The use of Specific Examples
  - Best approach when we are stuck is to make a start on a problem is to pick a specific example
  - Geometrical and graphical representing certain aspect of a problem can be useful

PROBLEM SOLVING ASPECTS

- Similarities among problems
  - Bring as much as past experiences as possible
  - New problem cannot be completely different
  - Sometimes it blocks us from discovering a new thing
  - In the first instance try to independently solve the problem
- Working Backwards from the solution
  - In some cases we assume that we already have solution and then try to work backwards
  - Whatever attempts we make that we make to get started on a problem we should write it down

PROBLEM SOLVING ASPECTS

- General Problem Solving Strategies
  - Divide and Conquer
  - Dynamic Programming
  - Trial and Error

TOP-DOWN DESIGN

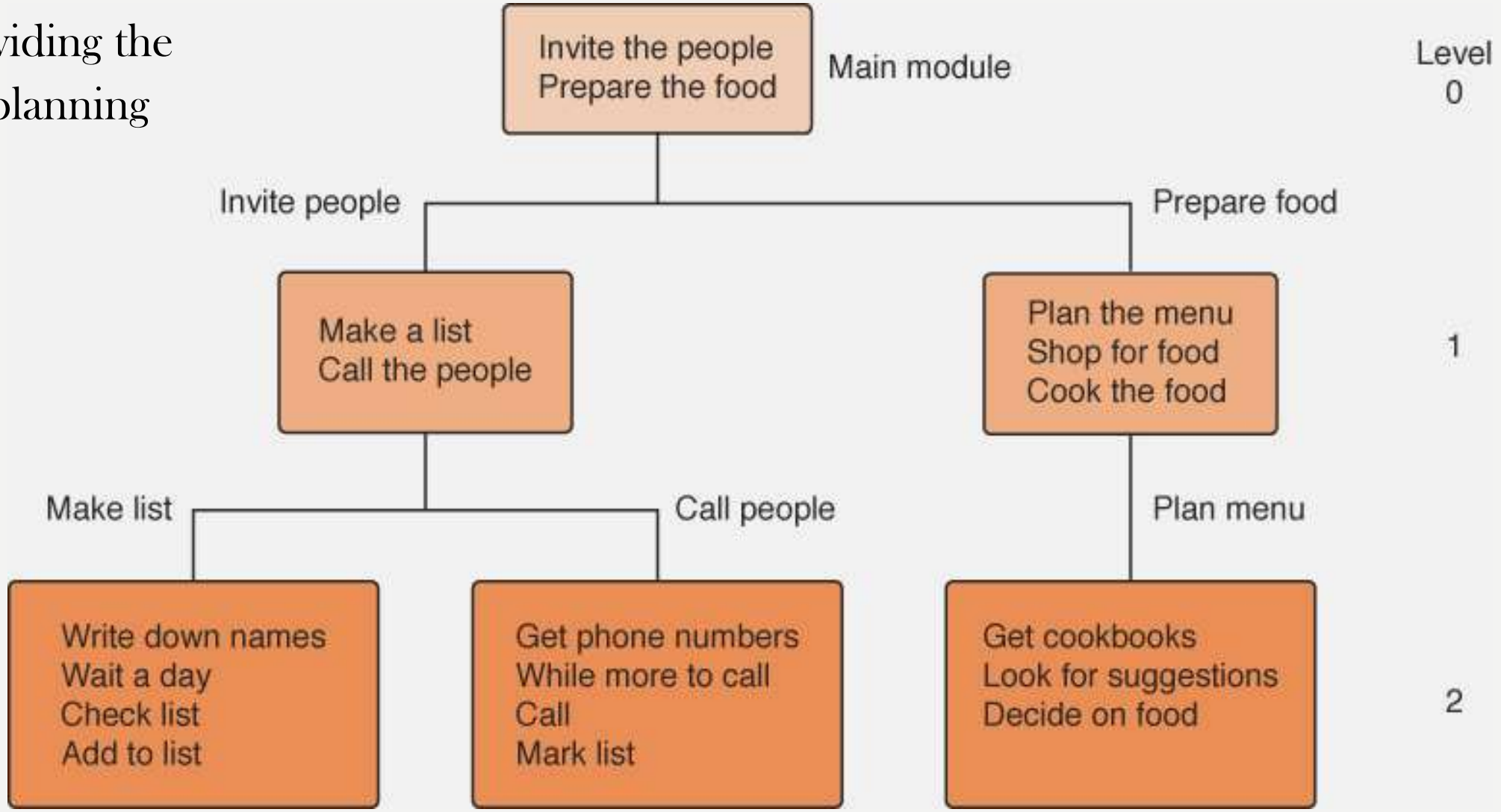
- Top-Down Design
  - Problem-solving technique in which the problem is divided into Sub problems; the process is applied to each sub problem.
- Modules
  - Self-contained collection of steps, that solve a problem or sub problem.
- Abstract Step
  - An algorithmic step containing unspecified details.
- Concrete Step
  - An algorithm step in which all details are specified.

TOP-DOWN DESIGN

- Planning a large party

TOP-DOWN DESIGN

- Subdividing the party planning

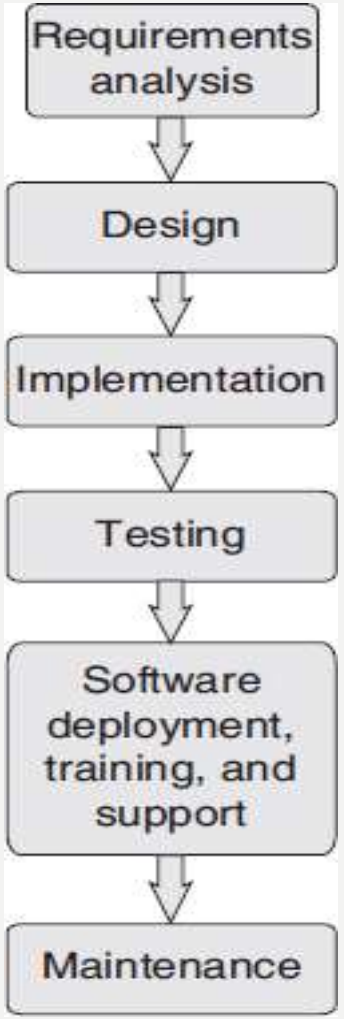


ADVANTAGES OF TOP-DOWN DESIGN

- Breaking the problem into parts helps us to clarify what needs to be done.
- At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work on the solution.

PROBLEM SOLVING STRATEGIES

- In requirements analysis, user‟s expectations are gathered to know why the software has to be built.
- In the design phase, a plan of actions is made before the actual development process can start.
- In implementation phase, the designed algorithms are converted into program code using any of the high-level languages.
- During testing, all the modules are tested together to ensure that the overall system works well as a whole product.
- In software deployment, training, and support phase, the software is installed or deployed in the production environment.
- Maintenance and enhancements are ongoing activities that are done to cope with newly discovered problems or new requirements



PROBLEM DESIGN TOOLS

- Program Design Tools are the tools used to develop a program.
- During designing a problem/program different tools are required to solve problem
- **There are three program tools**
  1. Algorithm
  2. Flowchart
  3. Pseudo code

1. ALGORITHM

- **Algorithm**

  A set of sequential steps usually written in Ordinary Language to solve a given problem is called Algorithm.

- **Steps involved in algorithm development**

  An algorithm can be defined as "a complete, unambiguous, finite number of logical steps for solving a specific problem "

  Step1. Identification of input:

  For an algorithm, there are quantities to be supplied called input and these are fed externally. The input is to be identified first for any specified problem.

STEPS INVOLVED IN ALGORITHM DEVELOPMENT (CONT.)

Step2: Identification of output:

From an algorithm, at least one quantity is produced, called for any specified problem.

Step3 : Identification the processing operations:

All the calculations to be performed in order to lead to output from the input are to be identified in an orderly manner.

Step4 : Processing Definiteness :

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

STEPS INVOLVED IN ALGORITHM DEVELOPMENT (CONT.)

Step5 : Processing Finiteness :

If we go through the algorithm, then for all cases, the algorithm should terminate after a finite number of steps.

Step6 : Possessing Effectiveness :

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

EXAMPLE

- Suppose we want to find the average of three numbers, the algorithm is as follows

Step 1 Read the numbers a, b, c
Step 2 Compute the sum of a, b and c
Step 3 Divide the sum by 3
Step 4 Store the result in variable d
Step 5 Print the value of d
Step 6 End of the program

APPROACHES TO DESIGNING AN ALGORITHM OR WAY TO SOLVE PROBLEM

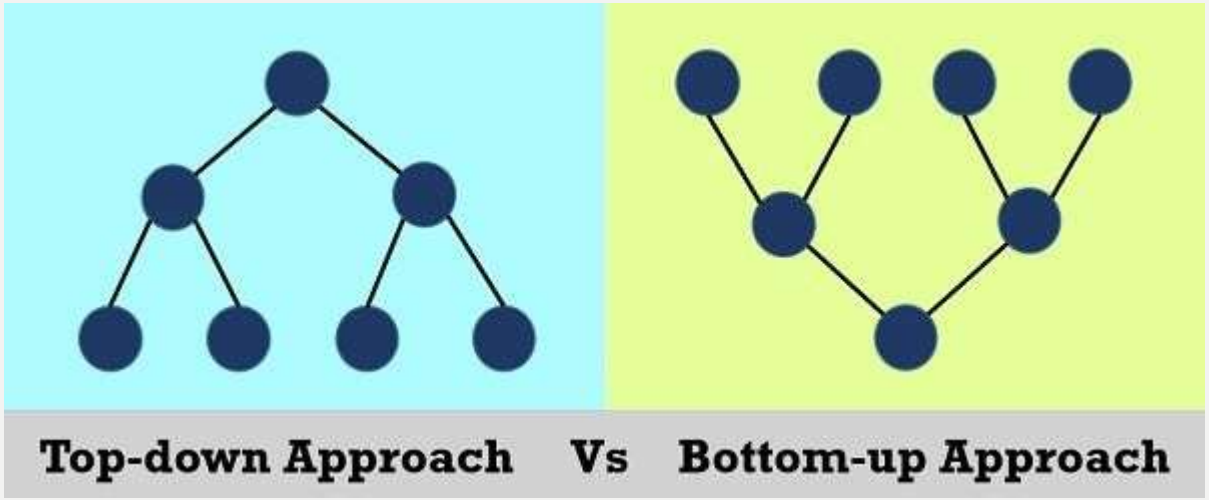
- There are typically two primary ways to approach problem-solving –

1. Top-down :

   Break Problem into smaller problems

2. Bottom-up :

   Solve smaller problem and then add features



CONTROL STRUCTURE USED IN ALGORITHM

- Sequence means that each step of the algorithm is executed in the specified order.
- Decision statements are used when the outcome of the process depends on some condition
- Repetition, which involves executing one or more steps for a number of times, can be implemented using constructs such as the while, do-while, and for loops.
- Recursion is a technique of solving a problem by breaking it down into smaller and smaller sub-problems until you get to a small enough problem that it can be easily solved. Usually, recursion involves a function calling itself until a specified condition is met.

```
Step 1 : Input first number as A
Step 2 : Input second number as B
Step 3 : Set Sum = A + B
Step 4 : Print Sum
Step 5 : End
```

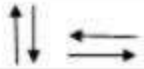
```
Step 1 : Input first number as A
Step 2 : Input second number as B
Step 3 : IF A = B
            Print "Equal"
         ELSE
            Print "Not equal"
         [END of IF]
Step 4 : End
```

```
Step 1 : [initialize] Set I = 1, N = 10
Step 2 : Repeat Steps 3 and 4 while I <= N
Step 3 : Print I
Step 4 : SET I = I + 1
         [END OF LOOP]
Step 5 : End
```

2. FLOWCHART

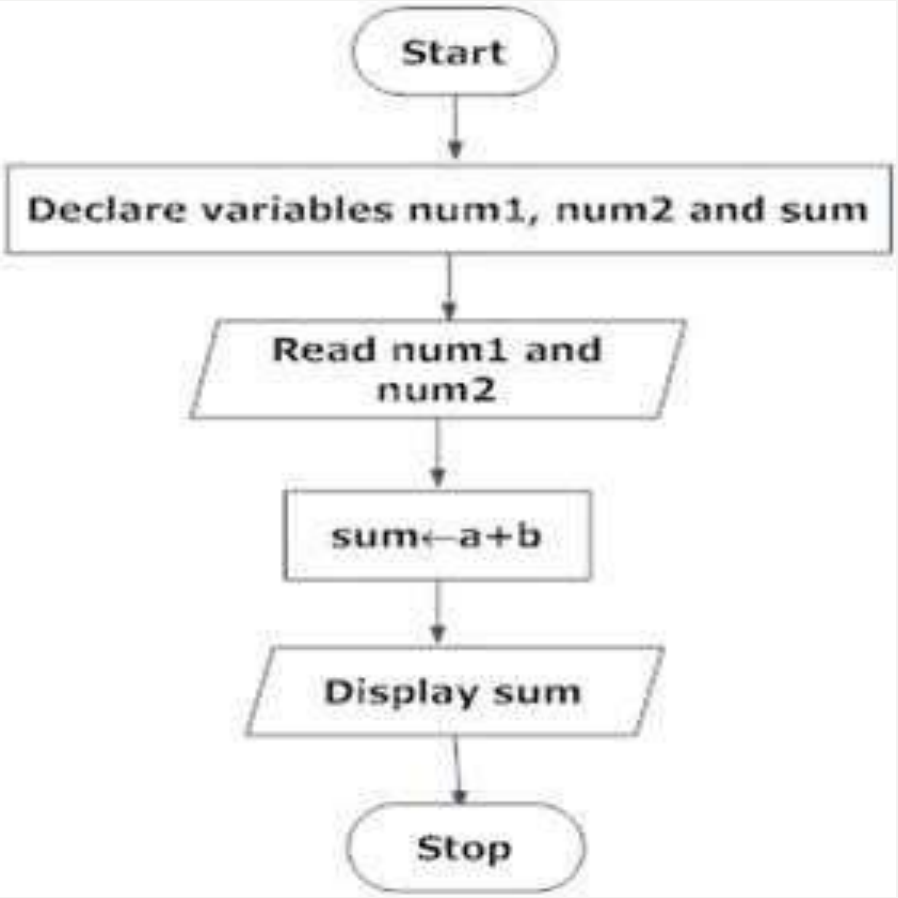
- A flow chart is a step by step diagrammatic representation of the logic paths to solve a given problem.
- A flowchart is visual or graphical representation of an algorithm.
- The flowcharts are pictorial representation of the methods to be used to solve a given problem and help a great deal to analyze the problem and plan its solution in a systematic and orderly manner.
- A flowchart when translated in to a proper computer language, results in a complete program.

SYMBOLS USED IN FLOW-CHARTS

Symbol	Name	Function
	**Process**	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

EXAMPLE:

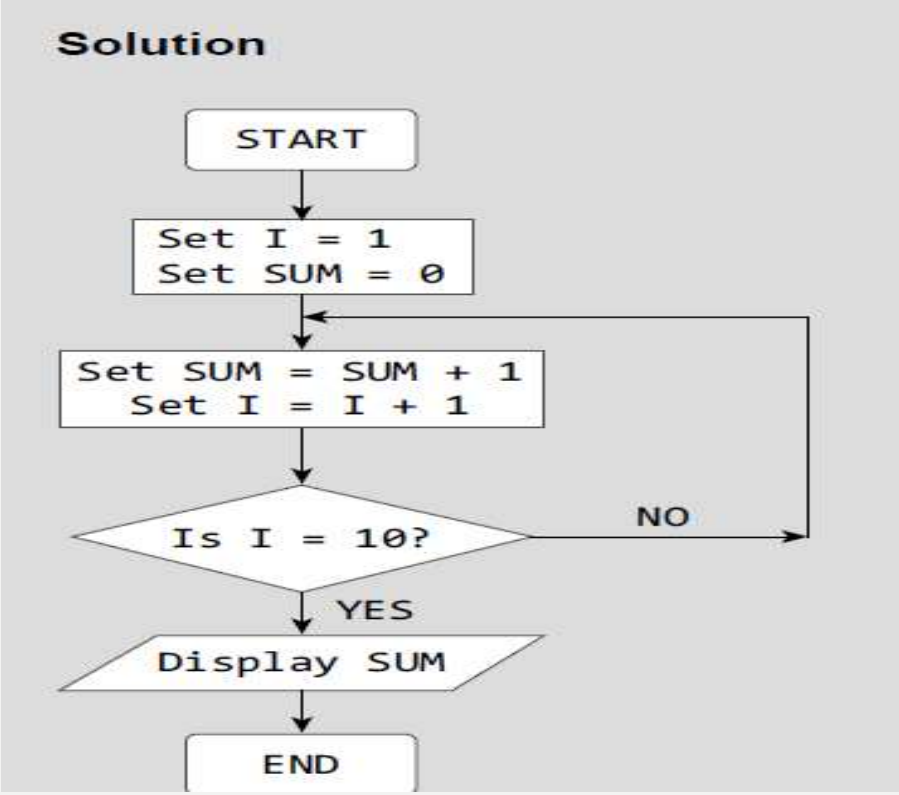
- Suppose we want to find the sum of two numbers, the flowchart is as follows



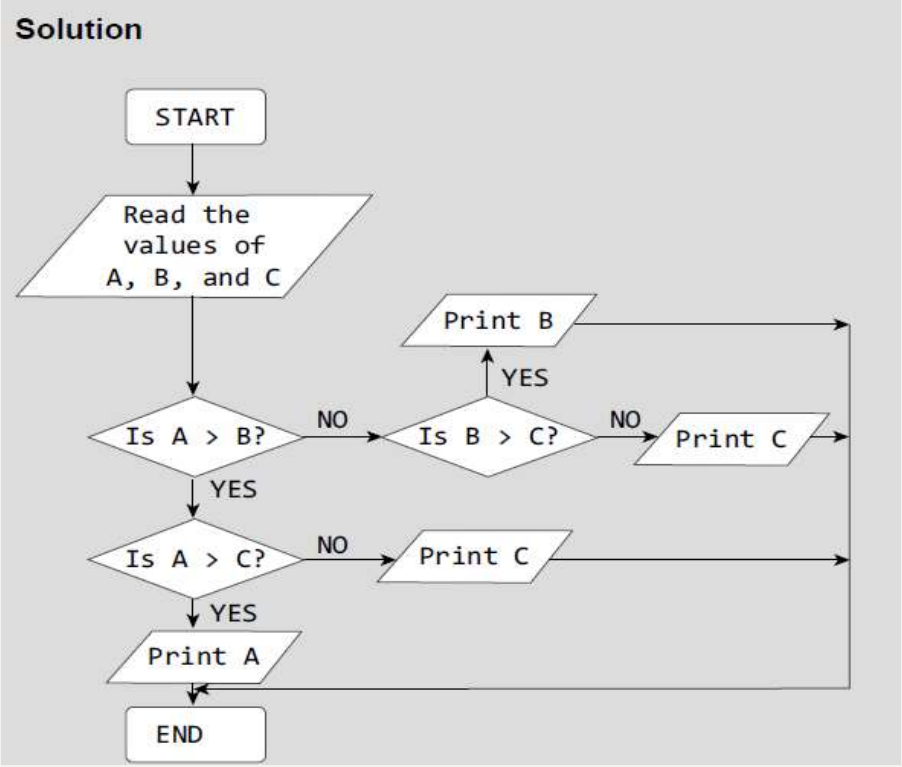
EXAMPLE OF FLOWCHART

Sinhgad Institutes

Example Draw a flowchart to calculate the sum of the first 10 natural numbers.



Example: Draw a flowchart to determine the largest of three numbers.



ADVANTAGES OF FLOWCHARTS

1. The flowchart shows the logic of a problem displayed in pictorial fashion which felicitates easier checking of an algorithm.
2. The Flowchart is good means of communication to other users. It is also a compact means of recording an algorithm solution to a problem.
3. The flowchart allows the problem solver to break the problem into parts. These parts can be connected to make master chart.
4. The flowchart is a permanent record of the solution which can be consulted at a later time.

3. PSEUDO CODE

- The Pseudo code is neither an algorithm nor a program. It is an abstract form of a program.
- An ideal pseudocode must be complete, describing the entire logic of the algorithm, so that it can be translated straightaway into a programming language.
- It consists of English like statements which perform the specific operations. It is defined for an algorithm. It does not use any graphical representation.
- In pseudo code, the program is represented in terms of words and phrases, but the syntax of program is not strictly followed.

- Easy to read
- Easy to understand
- Easy to modify

EXAMPLE:

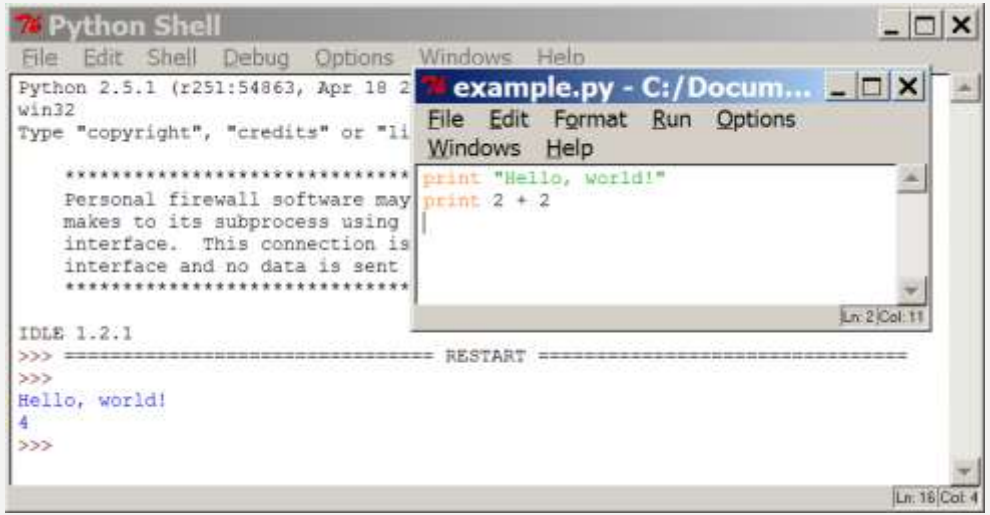
- **Write a pseudo code to perform the basic arithmetic operations.**

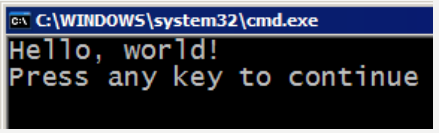
1. Read n1, n2
2. Sum = n1 + n2
3. Diff = n1 – n2
4. Mult = n1 * n2
5. Quot = n1/n2
6. Print sum, diff, mult, quot
7. End.


python
TM

PROGRAMMING BASICS

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.



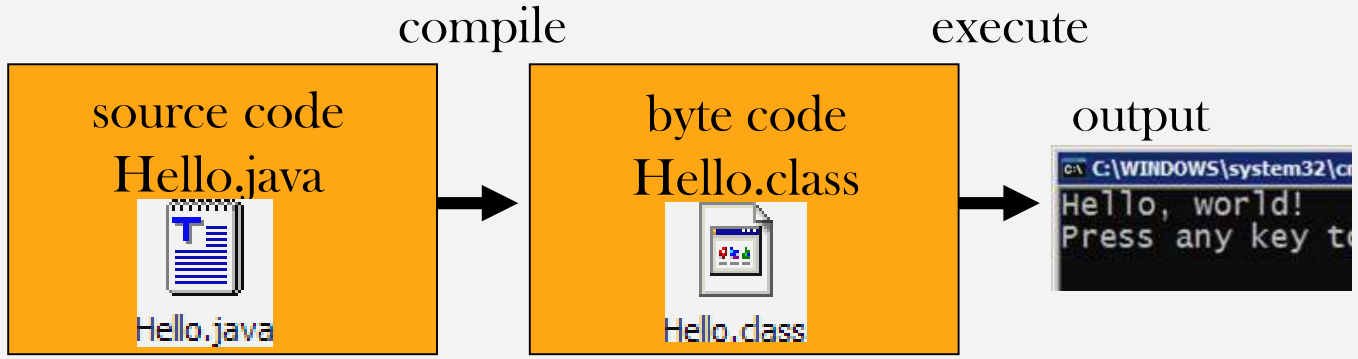


WHAT IS PYTHON

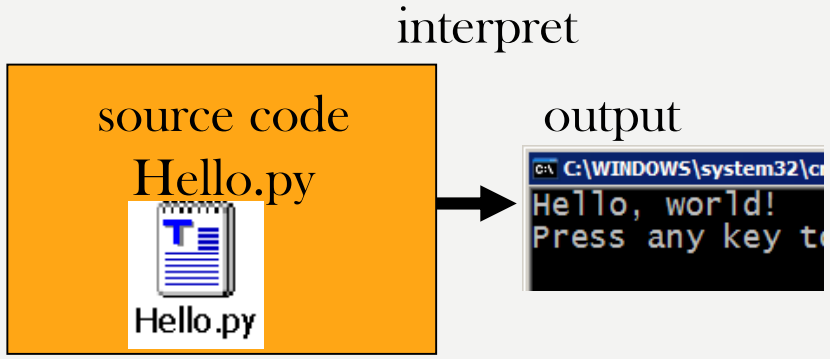
- Python is a **high-level, interpreted**, interactive and object-oriented scripting language.
- Multi-purpose (Web, GUI, Scripting, etc.)
- Python is designed to be highly readable.
- Strongly typed and Dynamically typed
- It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
- Languages that inspired Python – C, Perl, Algol 60, Pascal, Algol 68, ABC

COMPILING AND INTERPRETING

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly interpreted into machine instructions.



HISTORY OF PYTHON

Monty Python's *Flying Circus*

"At the time when he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus" (a BBC comedy series from the seventies, in the unlikely case you didn't know). It occurred to him that he needed a name that was **short**, **unique**, and **slightly mysterious**, so he decided to call the language Python."

-Charlie Cheever

HISTORY OF PYTHON

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Rossum published the first version of Python code (0.9.0) in February 1991 at the CWI (Centrum Wiskunde & Informatics) in the Netherlands , Amsterdam
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

PYTHON TIMELINE/HISTORY

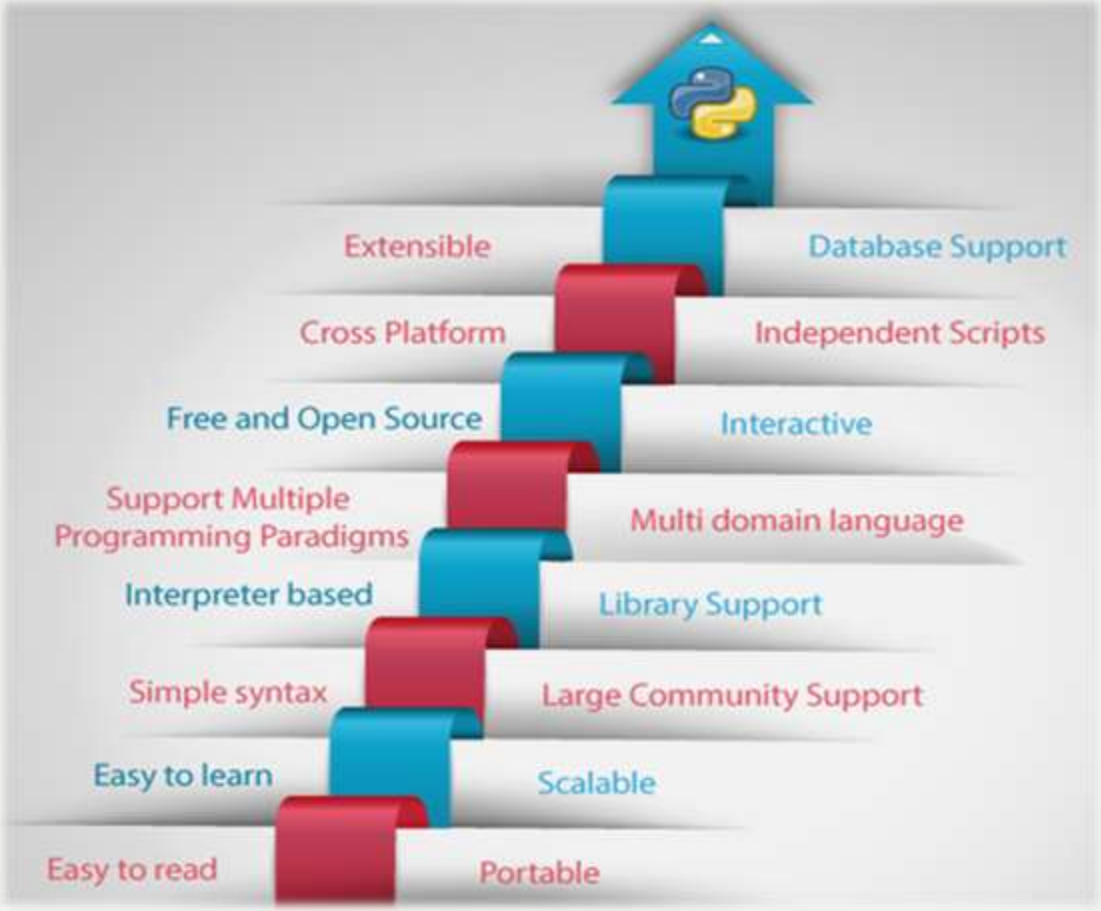
- In 1991 python 0.9.0 was released.
- In January of 1994 python 1.0 was released.
- In 1995, python 1.2 was released.
- In 2000, Python 2.0 was released.
- In 2001, Python 2.2 was released.
- Python 3.0 released in 2008

- Release dates for the major and minor versions:
- **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

- Release dates for the major and minor versions:
- **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 FEATURES

- Easy-to-learn
- Easy-to-read
- Easy-to-maintain
- A broad standard library
- Interactive Mode
- Portable
- Extendable
- Databases
- GUI Programming
- Scalable



FUTURE OF PYTHON

- Python Language is extensively used for web development, application development, system administration, developing games.
- Future of python:

    1. Artificial Intelligence
    2. Big data
    3. Networking

APPLICATIONS OF PYTHON

- https://www.menti.com/c7zbzvxti2

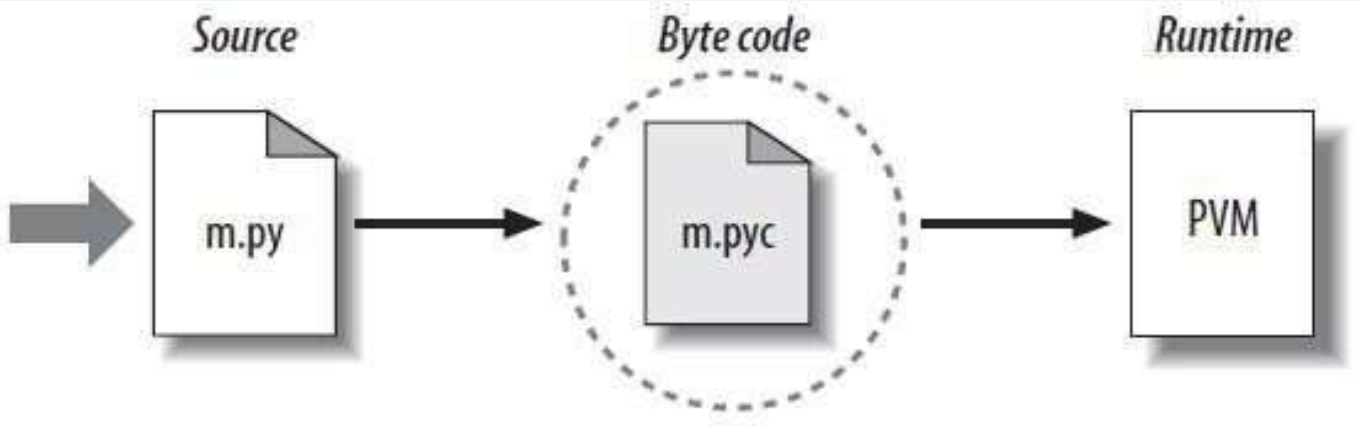
Applications of python

WHO USES PYTHON

- Google
- NASA
- IBM
- NOKIA
- Amazon
- ...the list goes on...

PYTHON CODE EXECUTION

Python's traditional runtime execution model: source code you type is translated to **byte code,** which is then run by the **Python Virtual Machine**. Your code is automatically compiled, but then it is interpreted.



Source code extension is **.py**

Byte code extension is **.pyc (compiled python code)**

INSTALLATION OF PYTHON

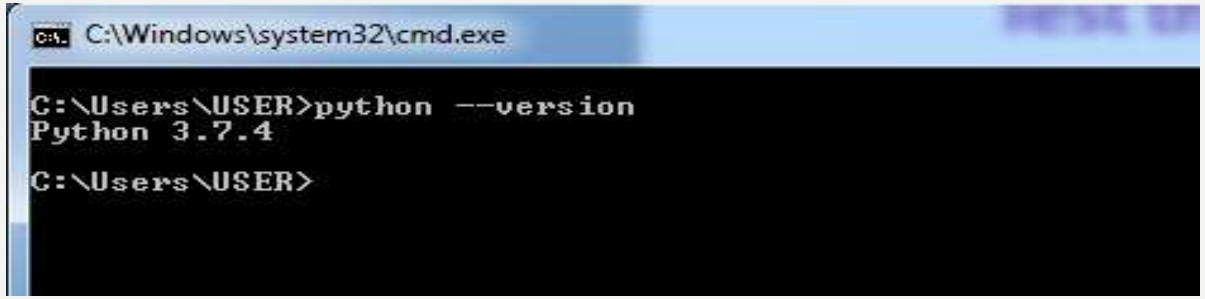
STEPS TO INSTALL PYTHON :

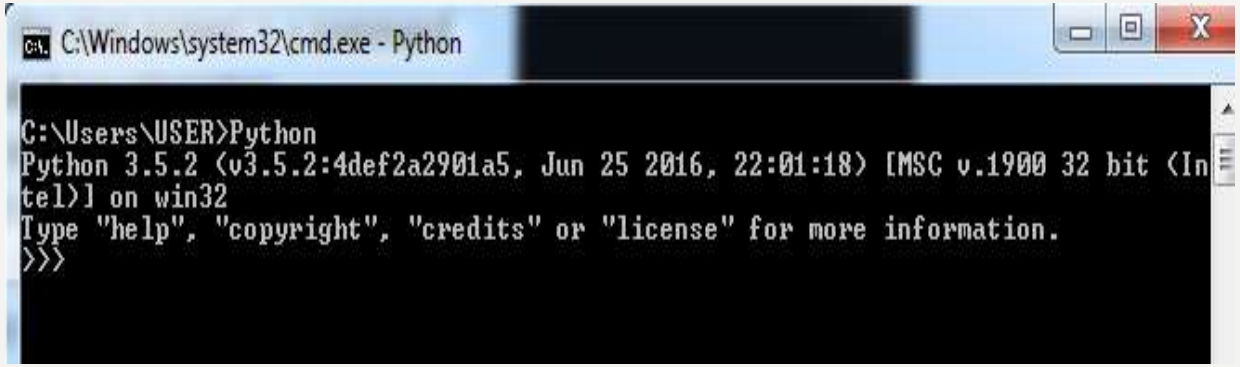
1. Download the latest version of python from official website
    1. URL to download the Python: https://www.python.org/downloads/
    2. Down load as per requirement like:
        - Operating System –Windows/Linux, etc.
        - Operating System – 32-bit or 64-bit.

INSTALLATION OF PYTHON (CONT..)

Steps to install and execute python program:

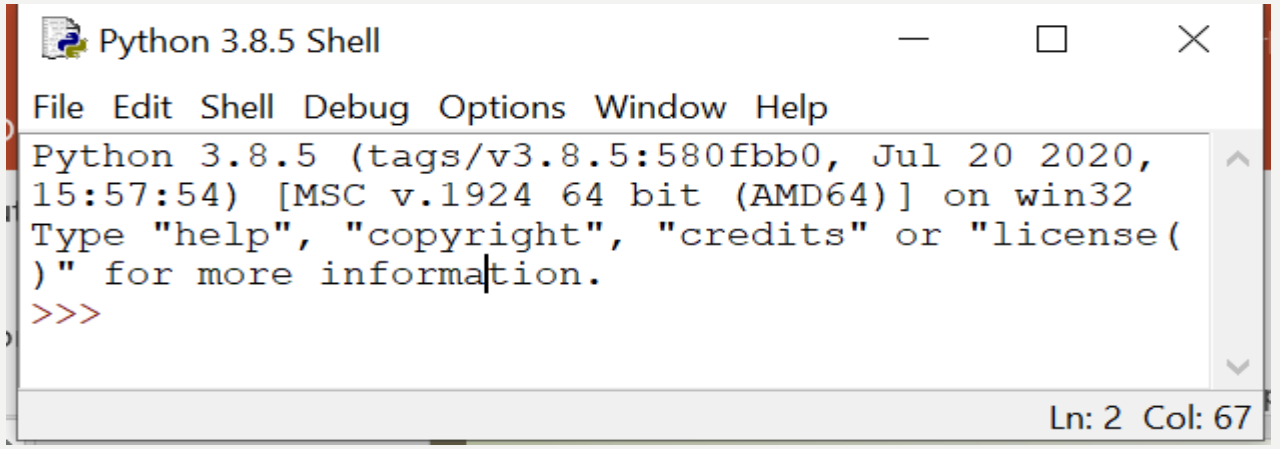
4. Install Python : Double Click on installable file. After successfully installation of setup ,check the version of python on your pc using following steps
    1. open command prompt
    2. type command : **python --version**
5. Test python Installation using **python** command

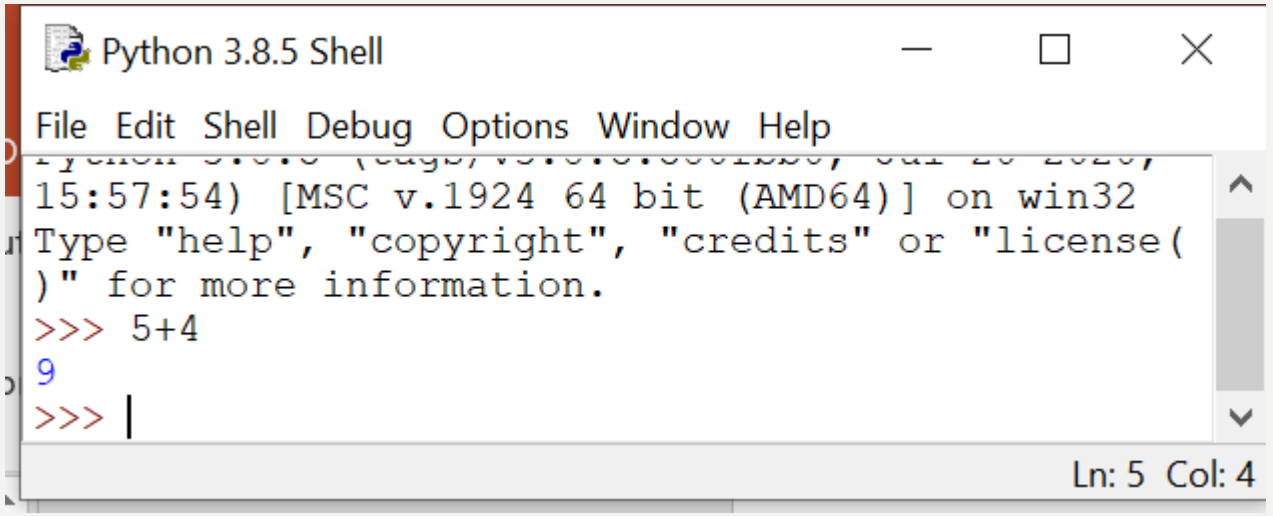




WRITING AND EXECUTING PYTHON PROGRAM

- Steps for writing in interactive mode:
  - **Open an editor (click on python or IDLE)**
  - **Write the instruction**
  - **It give result.**

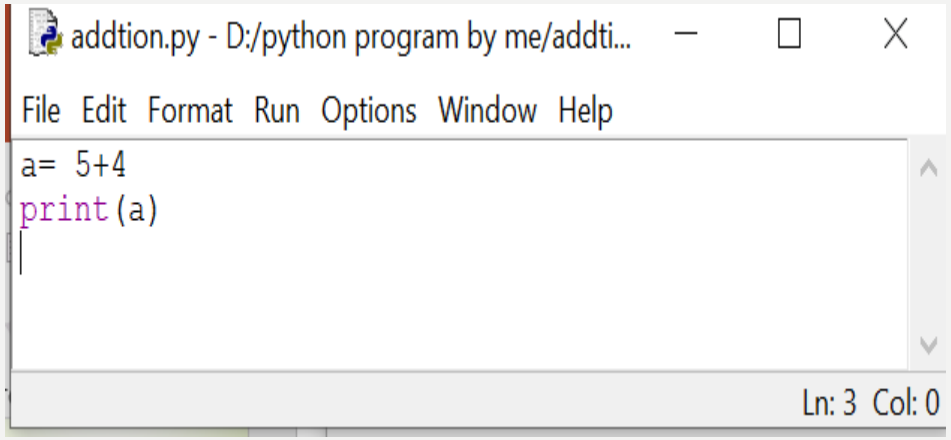


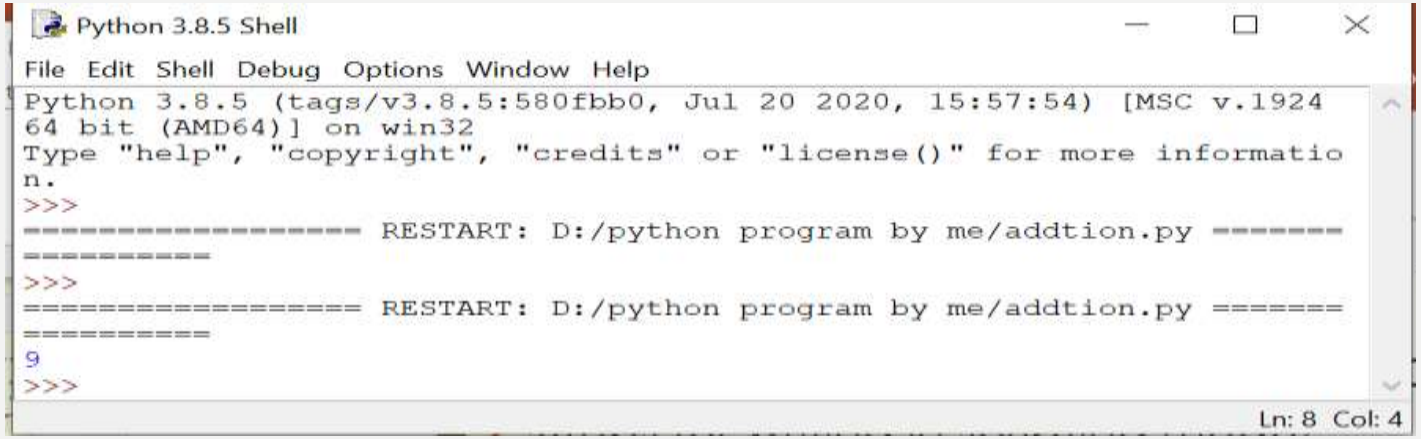


WRITING AND EXECUTING PYTHON PROGRAM

➢ Steps for writing in scripting mode:

1. Open an editor (click on python or IDLE)
2. click on new file
3. Write the instruction
4. Save it as a file with the filename having the extension .py (addition.py)
5. Run the interpreter with the command python program_name.py or use IDLE to run the programs.
6. If you want to execute the program in Python shell, then just press F5 key or click on Run Menu and then select Run Module.





Python IDLE

- Introduction
  - **IDLE** stands for Integrated Development and Learning Environment. The story behind the name IDLE is similar to Python. Guido Van Rossum named Python after the British comedy group Monty Python while the name IDLE was chosen to pay tribute to Eric Idle, who was one of the Monty Python's founding members. IDLE comes bundled with the default implementation of the Python language since the 01.5.2b1 release. It is packaged as an optional part of the Python packaging with many Linux, Windows, and Mac distributions.
  - IDLE, is a very simple and sophisticated IDE developed primarily for beginners, and because of its simplicity, it is highly considered and recommended for educational purposes.

PYTHON VARIABLES, CONSTANTS AND LITERALS

➢ PYTHON VARIABLES

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

```
number = 10
```

Here, we have created a named number.We have assigned value 10 to the variable.

PYTHON VARIABLES (CONT.)

- You can think variable as a bag to store books in it and those books can be replaced at any time.

```
number = 10
number = 1.1
```

Initially, the value of number was 10. Later it's changed to 1.1.

ASSIGNING A VALUE TO A VARIABLE IN PYTHON

- As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

- Example 1: Declaring and assigning a value to a variable

```
website = "SCOE.com"
    print(website)
```

When you run the program, the output will be:

```
SCOE.com
```

ASSIGNING A VALUE TO A VARIABLE IN PYTHON

- Example 2: Changing the value of a variable

```
website = “SCOE.com“ print(website)

        # assigning a new variable to Website

        website = “SKN.com“
        print(website)
```

- When you run the program, the output will be:

```
SCOE.com
SKN.com
```

ASSIGNING A VALUE TO A VARIABLE IN PYTHON

- In the above program, we have assigned apple.com to the website variable initially. Then, it's value is changed to programiz.com.

- Example 3: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, "Hello“

print (a)
print (b)
print (c)
```

Output:

```
5
3.2
Hello
```

CONSTANTS

- A constant is a type of variable whose value cannot be changed.
- It is helpful to think of constants as containers that hold information which cannot be changed later.
- Constants are written in all capital letters and underscore separating words
- Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag

ASSIGNING VALUE TO A CONSTANT IN PYTHON

- In Python, constants are usually declared and assigned on a module.
- The module means a new file containing variables, functions etc which is imported to main file.
- Inside the module, constants are written in all capital letters and underscores separating the words.

Example : Declaring and assigning value to a constant

- Create a constant.py

```
PI = 3.14
GRAVITY = 9.8
```

- Create a main.py

```
import constant
print(constant.PI)
print(constant.GRAVITY)
```

ASSIGNING VALUE TO A CONSTANT IN PYTHON

- When you run the program, the output will be:

  3.14

  9.8

- In the above program, we create a constant.py module file.

- Then, we assign the constant value to PI and GRAVITY.

- After that, we create a main.py file and import the constant module.

- Finally, we print the constant value.

ASSIGNING VALUE TO A CONSTANT IN PYTHON

- Create a name that makes sense. Suppose, vowel makes more sense than v.
- Use camelCase notation to declare a variable. It starts with lowercase letter. For example:

  myName

  myAge

  myAddress
- Use capital letters where possible to declare a constant. For example:

  PI

  G

  MASS

  TEMP

ASSIGNING VALUE TO A CONSTANT IN PYTHON

- Never use special symbols like !, @, # , $, %, etc.
- Don't start name with a digit.
- Constants are put into Python modules and meant not be changed.
- Constant and variable names should have combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example:

  Snake_case

  e.g."hello_world"

  MACRO_CASE

  camelCase

  e.g.iPhone

  CapWords

LITERALS

- The data that is provided in the variable are known as literals
- Literals can be defined as raw data that is given in variable or constant
- Literal is a constant value that is stored into a variable in a program.
- Example a=15, a – variable in which 15 constant value is stored , 15- Literal
- Python support the following literals:-
    1. String Literals
    2. Numeric Literals
    3. Boolean Literals
    4. Literal Collections such as List, Tuples, Dictionary
    5. None Literal

1. STRING LITERAL

- Single quotes or double quotes are used to defined by String literals.
- There are two kinds of strings supported in Python, single line and multiline string literals.

```
# Single Line Strings
name = "John Doe"
friend = "Ricky Ponting"

# Multiline String by black slash at the end of each line
hello = "Hello\
world"

# Using triple quotation marks
welcome = """welcome
to Programming
World of Python"""
```

2. NUMERIC LITERALS

- A number can be directly assigned to variables in Python.
- They are immutable (unchangeable).
- Numeric literals can belong to following four different numerical types.

Int: numbers can be both positive and negative with no fractional part. e.g. 432

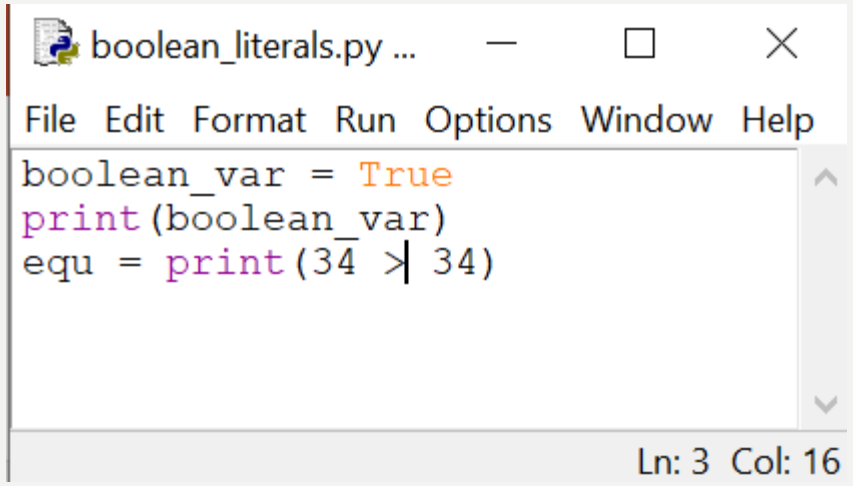
Long: Integers of unlimited size followed by lowercase or uppercase L e.g. 1422L

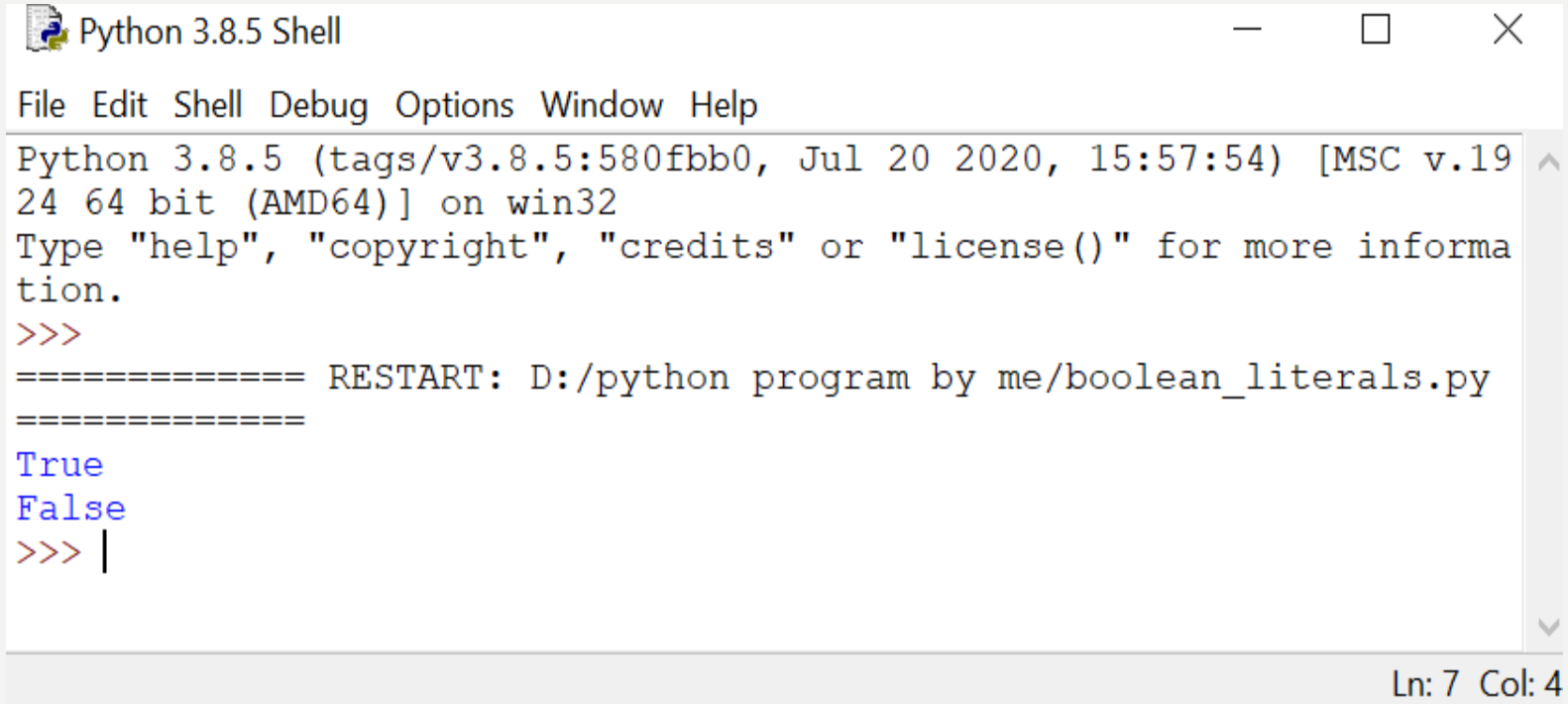
Float: Real numbers with both integer and fractional part eg: -26.2

Complex: In the form of a+bj where a forms the real part and b forms the imaginary part of a complex number. eg: 1+2j

3. BOOLEAN LITERALS

- A Boolean Literal can have any of the two values .
- True or False value.
- Boolean literals used for relational operators
- Example:





4.LITERAL COLLECTIONS

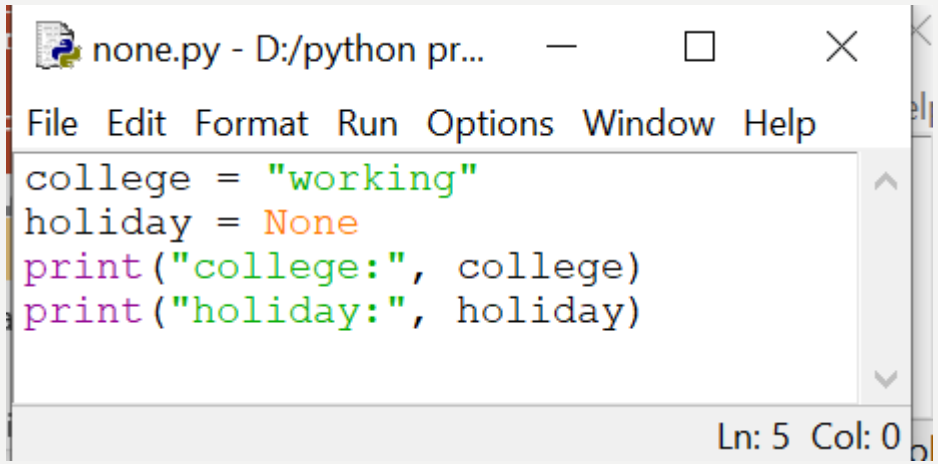
- There are 4 different literal collections List Literals, Tuple Literals, Dict Literals, and Set Literals. They represent more complex data and helps to provide extendibility to Python programs.
- Let us use an example to see how these Literals function:-

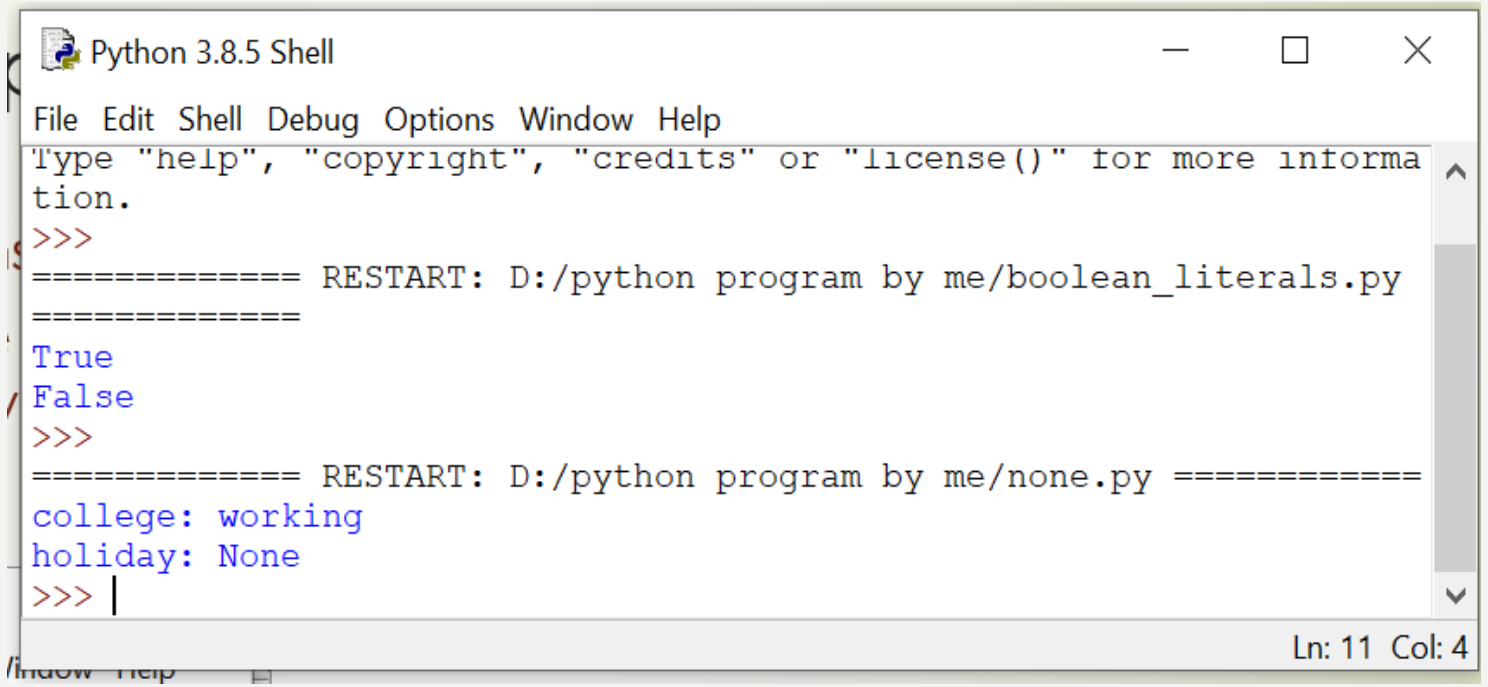
```
colors = ["red", "green", "yellow"] #list
numbers = (101, 202, 304) #tuple
student = {'name':'John Doe', 'address':'California', 'email':'john@doe.com'} #dictionary
vowels = {'a', 'e', 'i' , 'o', 'u'} #set

print(colors)
print(numbers)
print(student)
print(vowels)
```

5. SPECIAL (NONE) LITERAL

- Python contains a special type of literal known as None.
- It specifies the field that is not created. and also can indicate the end of lists In Python.
- Example:





IDENTIFIERS

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- Identifier helps to differentiate one entity from another.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language.
- Example: Manpower and manpower are two different identifiers in Python.

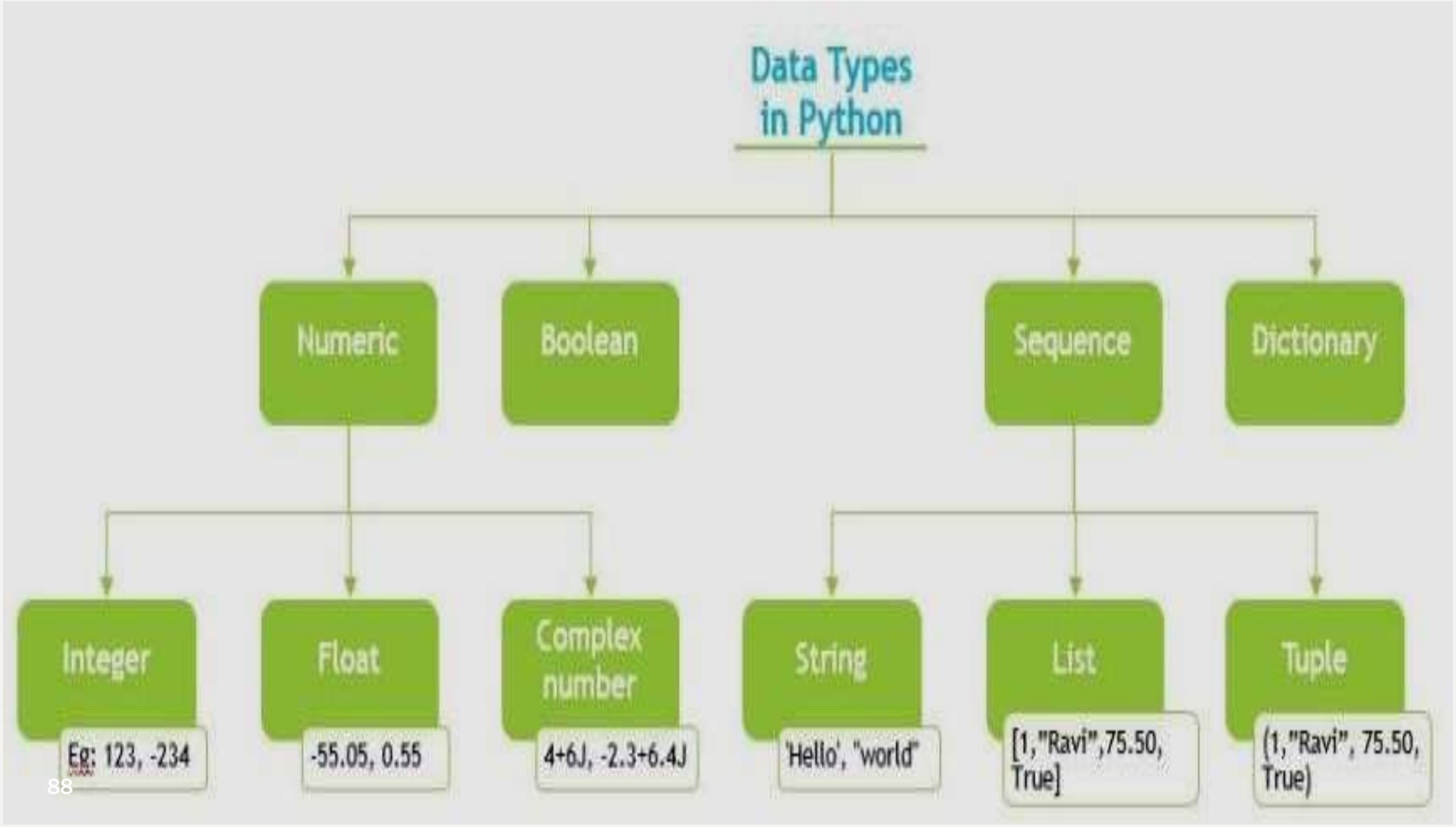
IDENTIFIERS (Cont.)

- Here are naming conventions for Python identifiers –

1. Class names start with an uppercase letter.
2. All other identifiers start with a lowercase letter.
3. Starting an identifier with a single leading underscore indicates that the identifier is private.e.g def__
4. Starting an identifier with two leading underscores indicates a strongly private identifier.
5. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

DATA TYPES

- Data type represents the type of data(literal) stored into a variable or memory.
- Python provides various standard data types .
- Python is dynamically typed language hence we need not define the type of the variable while declaring it.
- The interpreter implicitly binds the value with it.
  - 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. **Sets**
    6. **Dictionary**


Data Types in Python
Numeric
Boolean
Sequence
Dictionary
Integer
Eg: 123, -234
Float
-55.05, 0.55
Complex number
4+6J, -2.3+6.4J
String
'Hello', "world"
List
[1,"Ravi",75.50, True]
Tuple
(1,"Ravi", 75.50, True)

1. NUMBERS

- Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;
- a = 3 , b = 5 #a and b are number objects

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

2. 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.
- We can use slice [:] operators to access the substring.
- 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 "PythonPython ".

STRING (Cont.)

- The following example illustrates the string handling in python.

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

- **Output:**

```
he
o
hello javatpointhello javatpoint
hello javatpoint how are you
```

3. LIST

- Lists are similar to arrays in C. Order collection of data item.
- The list can contain data item 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.
- Syntax:

  L[Initial : End : IndexJump]
- If L is a list, then the above expression returns the portion of the list from index Initial to index End, at a step size Index Jump(by default 1).
- Consider the following example.

L = [1, "hi", "python", 2]

```
print(L[1:3:1])
print (L[3:])
print (L[0:2])
print (L)
print (L + L)
print (L * 3)
```

output: [hi, python]

```
[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]
```

MY_LIST =
['P','R','O','B','E']

4. 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(immutable) 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.

```
t = ("hi", "python", 2)
print (t[1:])
print (t[0:1])
print (t)
print (t + t)
print (t * 3)
print (type(t))
t[2] = "hi"
```

```
('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
```

5. SETS

- **Python Set** : Set is an unordered collection of unique items.
- Set is defined by values separated by comma inside braces { }.
- Items in a set are not ordered.
- Sets have unique values , they eliminate duplicates in sets.
- Slicing operator does not work or indexing has no meaning in sets

➢ e. g
```
s = {5,2,3,1,9, 4,10,9}
print("sets =", s)
print(s[1])
```
Output: sets = {5,2,3,1,4,10,9}

gives error

6. DICTIONARY

- Dictionary is an unordered 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.

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

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

TYPE FUNCTION :

Python provides us the type() function which returns the type of the variable passed.

- # Program to demonstrate type() function

```
num_int=10
num_float= 10.6
list = [10,20,30,'ten','twenty','thirty']
tuple = (1,2,'tuple', 'list', 2+3j)
str = 'python programming ,,
print("Type of num_int=",type(num_int))
print("Type of num_float=",type(num_float))
print("Type of list=",type(list))
print("Type of tuple=" ,type(tuple))
print("Type of str=" ,type(str))
```

- Output :

```
Type of num_int= <class 'int'>
Type of num_float= <class 'float'>
Type of list= <class 'list'>
Type of tuple= <class 'tuple'>
Type of str= <class 'str>
```

INPUT OPERATION

- Developers often have a need to interact with users, either to get data or to provide some sort of result.
- Most programs today use a dialog box as a way of asking the user to provide some type of input.
- While Python provides us with two inbuilt functions to read the input from the keyboard.
  - raw_input (prompt)
  - input (prompt)
- raw_input ( ) : This function works in older version (like Python 2.x). This function takes exactly what is typed from the keyboard, convert it to string and then return it to the variable in which we want to store.

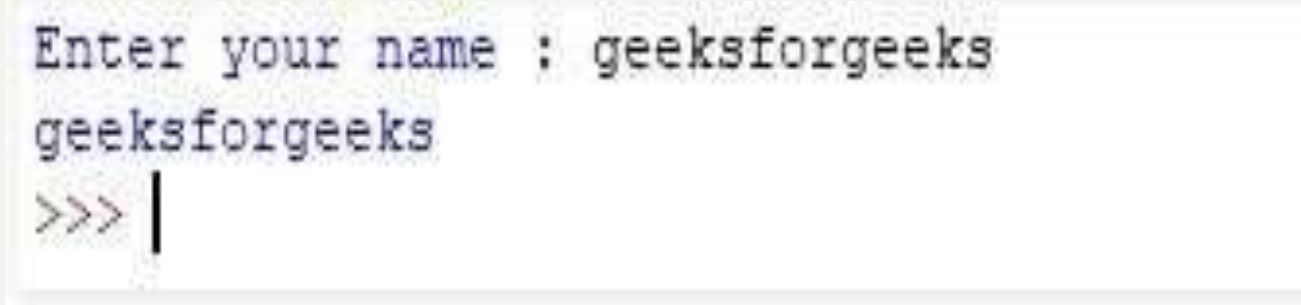
INPUT OPERATION (Cont.)

FOR EXAMPLE –

```
# Python program showing
# a use of raw_input()

g = raw_input("Enter your name : ")
print g
```

- Output :



INPUT OPERATION (Cont.)

- input ( ) : This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list.

- If the input provided is not correct then either syntax error or exception is raised by python.

```
# Python program showing
# a use of input()
val = input("Enter your value: ")
print(val)
```

```
Enter your value: 123
123
>>>
```

INPUT OPERATION WORKS (Cont.)

- When input() function executes program flow will be stopped until the user has given an input.
- The text or message display on the output screen to ask a user to enter input value is optional i.e. the prompt, will be printed on the screen is optional.
- Whatever you enter as input, input function convert it into a string
- If you enter an integer value still input() function convert it into a string
- You need to explicitly convert it into an integer in your code using **typecasting**.
- Example: str1 = "11"

```
print("String to int:",int(str1))
```

Output: String to int: 11

RESERVED WORDS

- The following list shows the Python keywords in python3.7 total 33 keywords.
- These are reserved words and you cannot use them as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters only but for ”True,False,None” use first letter as capital

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

COMMENTS

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
- You can type a comment on the same line after a statement or expression
- You can comment multiple lines as follows –

```
# This is a comment.
# This is a comment, too.
```

- For multiple line comment use “ “ “ multiline comment “ “ “ or single quote ,, ,, ,, multiline comment ,,,,,,

COMMENTS (Cont.)

comment.py - D:/python program by ...

File Edit Format Run Options Window Help

```
# This is a single line comment
print("Hello in Python*****")#First Line
#This is multiline comments
"""This is a multiline
   comment """
print("Welcome Scoe")
```

Ln: 8 Col: 0

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020,
15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license(
)" for more information.
>>>
================== RESTART: D:/python program by
 me/comment.py ==================
Hello in Python*****
Welcome Scoe
>>>
```

Ln: 7 Col: 4

INDENTATION

WITNESS

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- Generally four whitespaces are used for indention and preferred over tabs.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

```
if True:
    print "True"
else:
    print "False"
```

INDENTATION (Cont.)

- However, the following block generates an error −

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

- Thus, in Python all the continuous lines indented with same number of spaces would form a block.
- A code of block(body of function / loop) starts with indentation and ends with 1st indented line.
- Correct block format:

```
if True:
    print "True"
else:
    print "False"
```

OPERATORS AND EXPRESSIONS

- Operators are used to perform operations on variables and values.
- Operators can be symbol which is responsible for particular operation between operands.
- Python divides the operators in the following groups:
  1. **Arithmetic operators**
  2. **Assignment operators**
  3. **Comparison operators**
  4. **Logical operators**
  5. **Identity operators**
  6. **Membership operators**
  7. **Bitwise operators**

1. ARITHMETIC OPERATORS

- Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

2. ASSIGNMENT OPERATORS

- Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

3. COMPARISON OPERATORS

- Comparison operators are used to compare two values: x=3 y=4

Operator	Name	Example	output
==	Equal	print(x == y)	False
!=	Not equal	print(x != y)	True
>	Greater than	print(x > y)	False
<	Less than	print(x < y)	True
>=	Greater than or equal to	print(x >= y)	False
<=	Less than or equal to	print(x <= y)	True

4. LOGICAL OPERATORS

- Logical operators are used to combine conditional statements:

Operator	Description	Example	Output
and	Returns True if both statements are true	x=5 print(x > 3 and x < 10)	True
or	Returns True if one of the statements is true	x=5 print(x > 3 or x < 10)	True
not	Reverse the result, returns False if the result is true	x=5 print(not(x < 5 and x < 10))	True

5. IDENTITY OPERATORS

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location: For Example x=2 y=3

Operator	Description	Example	output
is	Returns true if both variables are the same object	print(x is y)	False
is not	Returns true if both variables are not the same object	print(x is not y)	True

6. MEMBERSHIP OPERATORS

- Membership operators are used to test if a sequence is presented in an object: For Example x=1 y=[3,5,7,1]

Operator	Description	Example	Output
in	Returns True if a sequence with the specified value is present in the object	print(x in y)	True
not in	Returns True if a sequence with the specified value is not present in the object	print(x not in y)	False

7. BITWISE OPERATORS

- Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits

Bitwise Operator Example

- **Bitwise AND operator:** Returns 1 if both the bits are 1 else 0.

  Example:

  a = 10 = 1010 (Binary) b = 4 = 0100 (Binary) a & b = 1010 &0100 = 0000 = 0 (Decimal)

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

  Example:

  a = 10 = 1010 (Binary) b = 4 = 0100 (Binary) a | b = 1010 | 0100 = 1110 = 14 (Decimal)

- **Bitwise not operator:** Returns one‘‘s complement of the number.

  Example:

  a = 10 = 1010 (Binary) ~a = ~1010 = -(1010 + 1) = -(1011) = -11 (Decimal)

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

  Example:

  a = 10 = 1010 (Binary) b = 4 = 0100 (Binary) a ^ b = 1010 ^ 0100 = 1110 = 14 (Decimal)

Bitwise Operator Program

Program:

```
a = 10
b = 4
# Print bitwise AND operation
print("a & b =", a & b)
# Print bitwise OR operation
print("a | b =", a | b)
# Print bitwise NOT operation
print("~a =", ~a)
# print bitwise XOR operation
print("a ^ b =", a ^ b)
```

Output:

```
a & b = 0
a | b = 14
~a = -11
a ^ b = 14
```

EXPRESSIONS

- Expressions is any legal combination of symbols (like variable, constants and operators.
- Expression are representations of value. They are different from statement in the fact that statements do something while expressions are representation of value.
- For example any string is also an expressions since it represents the value of the string as well.
- On evaluating an expression we get a value opened is the value which operator is applied.
- Python has some advanced constructs through which you can represent values and hence these constructs are also called expressions.

HOW TO CREATE AN EXPRESSIONS

- Python expressions only contain identifiers, literals, and operators. So, what are these?
- **Identifiers**: Any name that is used to define a class, function, variable module, or object is an identifier.
- **Literals**: These are language-independent terms in Python and should exist independently in any programming language.
- In Python, there are the string literals, byte literals, integer literals,floating point literals, and imaginary literals.


THANK
YOU !


THE FIRST STEP IS
ALWAYS THE HARDEST
@successpictures
SUCCESS
STEP 7
STEP 6
STEP 5
STEP 4
STEP 3
STEP 2
STEP 1