Unit 1 - Basics of Algorithms and Flow charts

Module 1 – Introduction to problem solving and algorithms

1.1 Introduction:

Intelligence is one of the key characteristics which differentiate a human being from other living creatures on the earth. Basic intelligence covers day to day problem solving and making strategies to handle different situations which keep arising in day to day life. One person goes Bank to withdraw money. After knowing the balance in his account, he/she decides to with draw the entire amount from his account but he/she has to leave minimum balance in his account. Here deciding about how much amount he/she may with draw from the account is one of the examples of the basic intelligence. During the process of solving any problem, one tries to find the necessary steps to be taken in a sequence. In this Unit you will develop your understanding about problem solving and approaches.

Problem Solving:

Can you think of a day in your life which goes without problem solving? Answer to this question is of course, No. In our life we are bound to solve problems. In our day to day activity such as purchasing something from a general store and making payments, depositing fee in school, or withdrawing money from bank account. All these activities involve some kind of problem solving. It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving. To make it clearer, let us see some other examples.

Example1: If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. move to that channel by pressing that channel number on your remote. This is a kind of problem solving.

Example 2: One Monday morning, a student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable. So here picking up books and copies as per timetable is a kind of problem solving.

Example 3: If someone asks to you, what is time now? Seeing time in your watch and telling is also a kind of problem solving.

Example 4: Some students in a class plan to go on picnic and decide to share the expenses among them. So calculating total expenses and the amount an individual have to give for picnic is also a kind of problem solving.

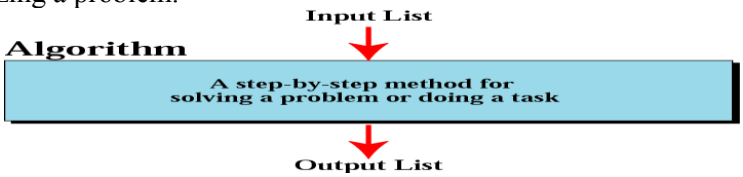
- Now, we can say that problem is a kind of barrier to achieve something and problem solving is a process to get that barrier removed by performing some sequence of activities.
- Here it is necessary to mention that all the problems in the world cannot be solved. There are some problems which have no solution and these problems are called Open Problems.
- If you can solve a given problem then you can also write an algorithm for it. In next section we will learn what is an algorithm?

1.2 Algorithm Introduction

Algorithm is a step-by-step process of solving a well-defined computational problem. In practice, in order to solve any complex real-life problems, first we have to define the problem and then, design algorithm to solve it. Writing and executing a simple program may be easy; however, for executing a bigger one, each part of the program must be well organized. In short, algorithms are used to simplify the program implementation. The study of algorithms is one of the key foundations of computer science. Algorithms are essential to the way computers process information, because a computer program is essentially an algorithm that tells the computer what specific steps to perform (in what specific order) in order to carry out a specified task, such as calculating employees' paychecks or printing students' report cards. Thus, an algorithm can be considered to be any sequence of operations that can be performed by a computing device such as a computer.

1.2.1 What is an Algorithm?

- Algorithm is a step by step procedure to solve a particular problem or "A sequence of activities to be processed for getting desired output from a given input.
- It's a easy way of analyzing a problem.



1.2.2 Structure of an algorithm:

START

- STEP 1
- SETP 2
- SETP 3
- SETP 4
- ……so on

STOP

Example 1: A simple algorithm to print „Good Morning".
Step 1: Start
Step 2: Print „Good Morning"
Step 3: Stop
Example 2: Let us take one simple day-to-day example by writing algorithm for making "Maggi Noodles" as a food.
Inputs to the algorithm: Maggie Noodles packet, Pan, Water, Gas
Expected output: Maggie Noodles
Algorithm:

Step 1: Start
Step 2: Take pan with water
Step 3: Put pan on the burner
Step 4: Switch on the gas/burner
Step 5: Put Maggie and masala
Step 6: Give two minutes to boil
Step 7: Take off the pan
Step 8: Take out the magi with the help of fork/spoon
Step 9: Put the Maggie on the plate and serve it
Step 10: Stop

Example3: Find the area of a Circle of radius r.
Inputs to the algorithm: Radius r of the Circle.
Expected output: Area of the Circle
Algorithm:

Step1: Start
Step2: Read\input Radios
Step3: Set 3.14 to PI
Step4: Set Area ←PI* Radios * Radios // calculation of area
Step5: Print Area
Step6: stop

1.3 Expressing Algorithms:
An algorithm is a set of steps designed to solve a problem or accomplish a task. Algorithms are usually written in pseudo code, or a combination of your speaking language and one or more programming languages, in advance of writing a program**.** An algorithm may be expressed in a number of ways, including:

- **Natural language:** usually verbose and ambiguous.
- **Flow charts:** avoid most (if not all) issues of ambiguity; difficult to modify w/o specialized tools; largely standardized.
- **Programming language:** Tend to require expressing low – level details that are not necessary for a high – level understanding.

1.4 Qualities of a good algorithm

1. Inputs and outputs should be defined precisely.
2. A good algorithm should produce correct and accurate results for any set of legal or correct inputs.
3. Each step in algorithm should be clear and unambiguous.
4. Algorithm should be most effective among many different ways to solve a problem.
5. An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that it can be used in similar programming languages Whenever we write an algorithm, we should make sure that all of these parameters given for a good algorithm are incorporated.

1.5 Advantage and disadvantages of algorithm
1.5.1 Advantages

1. It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
2. An algorithm uses a definite procedure.
3. It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
4. Every step in an algorithm has its own logical sequence so it is easy to debug.
5. By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program

1.5.2 Disadvantages

1. Writing algorithm takes a long time.
2. An Algorithm is not a computer program, it is rather a concept of how a program should be
3. Big tasks are difficult to put in Algorithms.
4. Difficult to show Branching and Looping in Algorithms.

1.6 Properties of algorithm:

1. **Finiteness:** An algorithm must always terminate after a finite number of steps. It means after every step one reach closer to solution of the problem and after a finite number of steps algorithm reaches to an end point.
2. **Definiteness:** Each step of an algorithm must be precisely defined. It is done by well thought actions to be performed at each step of the algorithm. Also the actions are defined unambiguously for each activity in the algorithm.
3. **Input:** An algorithm has zero or more inputs, taken from a specified set of objects.
4. **Output:** An algorithm has one or more outputs, which have a specified relation to the inputs.
5. **Effectiveness:** All operations to be performed must be sufficiently basic that they can be done exactly and in finite length

1.7 Algorithm Efficiency

Algorithmic efficiency is a property of an algorithm which relates to the number of computational resources used by the algorithm. An algorithm must be analyzed to determine its resource usage, and the efficiency of an algorithm can be measured based on usage of different resources.

- ❖ Speed - Method that takes the least time
- ❖ Space - Method that uses the least memory
- ❖ Code — Method that is shortest to describe

Speed is now the most important factor Example

A real world example: Travelling from Vempalli to Hyderabad in Vehicle

Case 1:

1. Take vehicle
2. Check vehicle condition if not get it repair
3. Drive from vempalli to Proddutur
4. Then Proddutur to Hyderabad
5. Travel is successfully completed.

Case 2:

1. Take vehicle
2. Check vehicle condition if not get it repair
3. Drive from Vempalli to kadapa
4. Then Kadapa tokurnool
5. Then Kurnool to Hyderabad
6. Travel is successfully completed

While comparing both cases Algorithm executed successfully with same output but in case 1 travelling time is lesser than the case 2 travelling time.

1.8 Tracing an Algorithm

Tracing of an Algorithm means showing how the value of a variable changes during the execution of an algorithm.

Steps in tracing:

1. Identify the variables in the algorithm that need to be traced.
2. Examine the value of the variables at each step in the execution of the algorithm.
3. Determine if the algorithm is giving the correct outputs for a set of legitimate/legal input. Values.
4. Analyze and determine what the purpose of the algorithm

In this module we will see examples of computational algorithms and the techniques used for tracing algorithms. A computational algorithm is a set of precise instructions expected to be expressed as a computer program that can execute on a computer.

Let us now get started with an example of adding two numbers.

Algorithm: Add two numbers

Step1: Start
Step2: Input x
Step3: Input y
Step4: Set z ←x+ y
Step5: Output z
Step6: Stop

If you notice the algorithm it takes two inputs, performs an addition operation and gives the output. Let us see how the algorithm executes step by step.

Input x/* Takes input from the user by using input devices like keyboard. After entering input from the key board, for ex: 5 the value is stored in a memory location. The name of this memory location is x in this particular case. Any time this variable x can store only one value. For ex: if you set x value 6 you will find the new value is referred by x is 6 but not 5 that means the old value which was stored in x values is lost. This is a very good example that shows a variable can store only one value at a time. Input value :5*/ Input y /* Similar to previous statement where new memory location is created for variable y and the user inputs to y is stored.Input value :2*/

Set z to x+ y /* in this statement actual operation that is performed on the input values which is to add the numbers referred to by the variables x and y. In this example you will notice after the addition operation is executed and the values 5 and 2 are added and set to new variable z. */

Output z //the result value i.e. z is displayed to an output device like monitor?

You will notice that algorithm also can be visualized as functions. A function performs a specific operation like addition and multiplication or it could be more complex operations like finding square roots or sorting list of numbers.

Variable:

- ❖ Variables are used to handle the data by the algorithm.
- ❖ Variables refer to a memory location where the actual value is stored.
- ❖ Variable can store one value at a time.
- ❖ Old values of the variable are lost when a new value is assigned.

1.9 Control structure:

Control Structures are just a way to specify flow of control in programs. Any algorithm or program can be clearer and more understood if they use self-contained modules called as logic or control structures. It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions.

19.1 Types of control structures:

There are three types of control structures

1. Sequence
2. Selection or Branching
3. Loop or Repetition

1.9.1Sequence Control Structure: This refers to the line – by – line execution, in which statements are executed sequentially, in the same order in which they appear in the script.They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.

Example problem:

Write algorithm to find the greater number between two numbers

Step1: Start

Step2: Read\input the Number1

Step3: Read\input the Number2.

Step4: Set Average ← (Number1+ Number2)/2

Step5: Print Average

Step6: End

1.9.2Selection or Decision Control Structure: The branch refers to a binary decision based on some condition. Depending on whether a condition is true or false. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken. This is usually represented by the 'if-then' construct in pseudo-codes and programs. This structure is also known as the selection structure

Example problem:

A person whose age is more than or equals to 18 years is eligible to vote. Now write an algorithm to check whether he is eligible to vote?

Step 1: Start

Step2: Input/Read age //Taking age value from the user

Step 3: Check if age >= 18

a) Print "Eligible to vote" //If Condition is true

Step 4: Otherwise/else

a) Print "Not eligible to vote" //if Condition is false

Step 5: Stop

1.9.3 Repetition or Loop Control Structure: This is a control structure that allows the execution of a block of statements multiple times until a specified condition is met. The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the 'while' and 'for' constructs in most programming languages, for unbounded loops and bounded loops respectively. (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied; bounded loops refer to those whose number of iterations is known before-hand.) In the flowcharts, a back arrow hints the presence of a loop. A trip around the loop is known as iteration. You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the repetition structure.

Example problem:

Algorithm to print all-natural numbers upto "n" (Here you need to accept a number from the user, and to print all the natural numbers till 'n' i.e. 1 to n)

Step 1: Start

Step2: Read/Input n

Step 3: Store 1 to "i" //initialization

Step 4: Do the following statements until i<=n//condition

a) print i

b) add 1 to i//increment

Step 5: Stop

Nested control structures: Combining the use of these control structures, for example, a loop within a loop (nested loops), a branch within another branch (nested if), a branch within a loop, a loop within a branch, and so forth, is not uncommon. Complex algorithms may have more complicated logic structure and deep level of nesting, in which case it is best to demarcate parts of the algorithm as separate smaller modules. Beginners must train themselves to be proficient in using and combining control structures appropriately, and go through the trouble of tracing through the algorithm before they convert it into code.

Module 2 - Introduction to Flow chart

1.10 Flowchart

The diagrammatic representation of an algorithm is called "Flowchart".Before you start coding a program it is necessary to plan the step by step solution to the task your program will carry out. Such a plan can be symbolically developed using a diagram. This diagram is then called a flowchart. Hence a flowchart is a symbolic representation of a solution to a given task. A flowchart can be developed for practically any job. Flowcharting is a tool that can help us to develop and represent graphically program logic sequence.

For example suppose you are going for a picnic with your friends then you plan for the activities you will do there. If you have a plan of activities then you know clearly when you will do what activity. Similarly when you have a problem to solve using computer or in other word you need to write a computer program for a problem then it will be good to draw a flowchart prior to writing a computer program. Flowchart is drawn according to defined rules.

Features of flowchart:

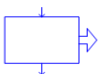
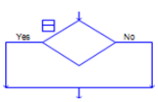
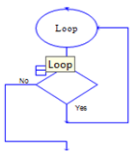
The features of a flowchart are:

1. It is an easy method of communication.
2. It is independent of a programming language.
3. It is the key to correct programming.
4. It clearly indicates the task to be performed at each level.

1.11 Flowchart Symbols

There are 6 basic symbols commonly used in flowcharting of assembly language Programs: Terminal, Process, and input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols; it is the ones used most often in the structure of Assembly language programming.

Purpose	Symbol	Name	Description
Start/Stop		Start & Stop	It is Oval symbol
INPUT		Input statement	Allow the user to enter data. Each data value is stored in a ***variable***.
OUTPUT		Output statement	Display (or save to a file) the value of a ***variable***.
PROCESSING		Assignment statement	An assignment statement is used to modify or replace the data stored at the memory location associated with a variable. .

Purpose	Symbol	Name	Description
PROCESSING (Function)		Procedure call	Execute a group of instructions defined in the named procedure. In some cases some of the procedure arguments (i.e., ***variables***) will be changed by the procedure's instructions.
Selection		Decision making statement	Allows you to make 'decision ' in boolean type
Loop		Iteration statements	Which allows you to execute repeatedly until certain condition satisfied. (An iteration control statement controls how many times a block of code is executed)

1. **Process Symbol:**
   - A process symbol is used to represent arithmetic and data movement instructions in the flowchart. All arithmetic processes of addition, subtraction, multiplication and division are indicated in the process symbol.
   - The logical process of data movement from one memory location to another is also represented in the process box. If there are more than one process
2. **Input/ Output Symbol:**
   - This symbol is used to denote any input/output function in the program. Thus, if there is any input to the program via an input device, like a keyboard, tape, card reader etc. it will be indicated in the flowchart with the help of the Input/output symbol.
   - Similarly, all output instructions, for output to devices like printers, plotters, magnetic tapes, disk, monitors etc. are indicated in the Input/ Output symbol.
3. **Decision Symbol:**
   - The decision symbol is used in a flowchart to indicate the point where a decision is to be made and branching done upon the result of the decision to one or more alternative paths. The criteria for decision making are written in the decision box.
   - All the possible paths should be accounted for. During execution, the appropriate path will be followed depending upon the result of the decision.
4. **Loops:** Looping is a problem-solving technique through which a group of statements is executed repeatedly, until certain specified condition is satisfied. Looping is also called a repetitive or an iterative control statement
5. **Connectors:**
   This symbol shows continuation of the flow chart from one page to another. When you reach the bottom of the page or need to jump to another page, draw flow chart connector symbol and connect it to the last item on the chart. Label the inside of the symbol with a letter, typically beginning with an "A".
6. **Predefined process (Function):**
   This symbol indicates a sequence of actions that perform a specific task embedded within a larger process.
7. **Terminal Symbol:**
   - Every flowchart has a unique starting point and an ending point.
   - The flowchart begins at the start terminator and ends at the stop terminator.
   - The Starting Point is indicated with the word START inside the terminator symbol. The Ending Point is indicated with the word STOP inside the terminator symbol. There can be only one START and one STOP terminator in your entire flowchart.

1.12 General Rules for flowcharting

1. All boxes of the flowchart are connected with Arrows. (Not lines)
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
4. Generally, a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
5. Connectors are used to connect breaks in the flowchart. Examples are:
   i. From one page to another page.
   ii. From the bottom of the page to the top of the same page.
6. Subroutines (Functions) and Interrupt programs have their own and independent flowcharts.
7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
8. All flowcharts end with a terminal or a contentious loop.

Flowcharting uses symbols that have been in use for a number of years to represent the type of operations and/or processes being performed. The standardized format provides a common method for people to visualize problems together in the same manner. The use of standardized symbols makes the flow charts easier to interpret, however, standardizing symbols is not as important as the sequence of activities that make up the process.

1.13 Advantages and Disadvantages of using flowchart

1.13.1 Advantages: As we discussed flow chart is used for representing algorithm in pictorial form. This pictorial representation of a solution/system is having many advantages. These advantages are as follows:

- **Communication:** A Flowchart can be used as a better way of communication of the logic of a system and steps involve in the solution, to all concerned particularly to the client of system.
- **Effective analysis:** A flowchart of a problem can be used for effective analysis of the problem.
- **Documentation of Program:** Program flowcharts are a vital part of good program documentation. Program document is used for various purposes like knowing the components in the program, complexity of the program etc.
- **Efficient Program Maintenance:** Once a program is developed and becomes operational it needs time to time maintenance. With help of flowchart maintenance become easier.

- **Coding of the Program:** Any design of solution of a problem is finally converted into computer program. Writing code referring the flowchart of the solution become easy.

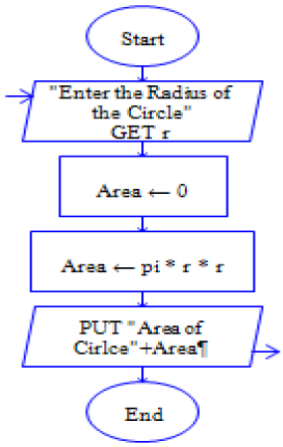
1.13.2Disadvantages:

- Drawing flowchart is a time-consuming task.
- Manual tracing is needed to check correctness of flowchart drawn on paper.
- Simple modification in problem logic may lead to complete redraw of flowchart.
- Showing many branches and looping in **flowchart** is difficult.
- In case of complex program/algorithm, **flowchart** becomes **very complex** and **clumpsy.**

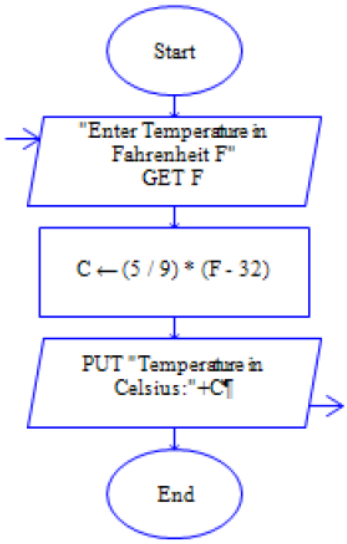
1.14 Some examples of Flowcharts

Now, we will discuss some examples on flowcharting. These examples will help in proper understanding of flowcharting technique. This will help you in program development process in next unit of this block.

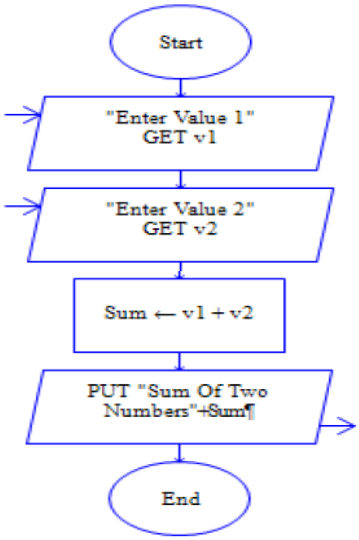
Problem1: Find the area of a circle of radius r



Problem 2: Convert temperature Fahrenheit to Celsius.



Problem3: Flowchart for an algorithm which gets two numbers and prints sum of their value.



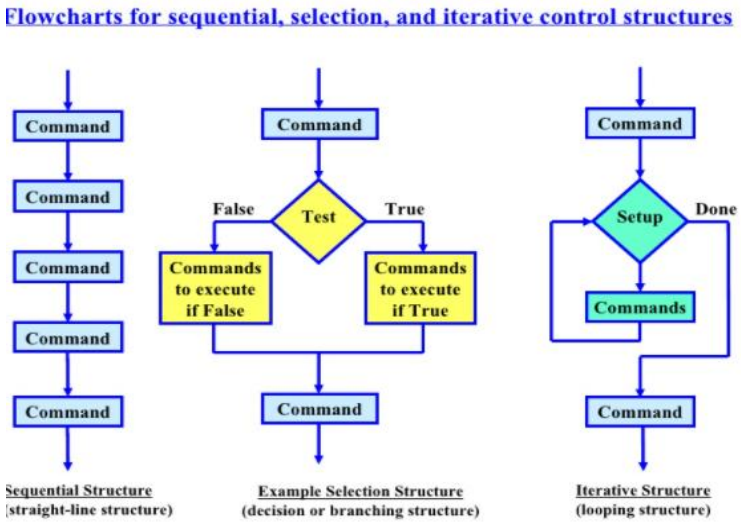
1.15 Types of control structures:

Definition

- The logic of a program may not always be a linear sequence of statements to be executed in that order.
- The logic of the program may require execution of a statement based on a decision.
- Control structures specify the statements to be executed and the order of execution of statements.

Where is the usage of control structure?

Flowchart and Pseudo Code use Control structures for representation



There are three kind of Control Structure:

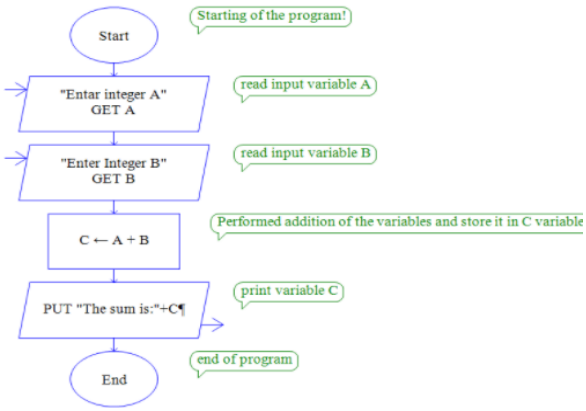
1. Sequential
2. Selection(Branch or conditional)
3. Iterative(loop)

1.15.1 Sequential:

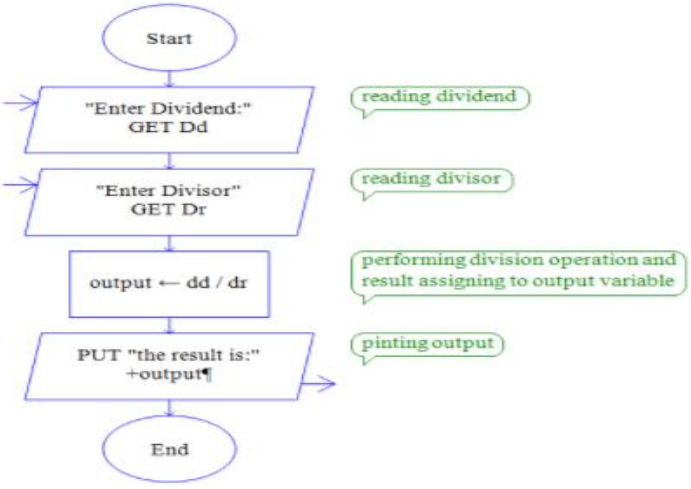
- Instruction is executed in linear order.
- Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.

Example problem:

1. Take two numbers from the user print their sum?



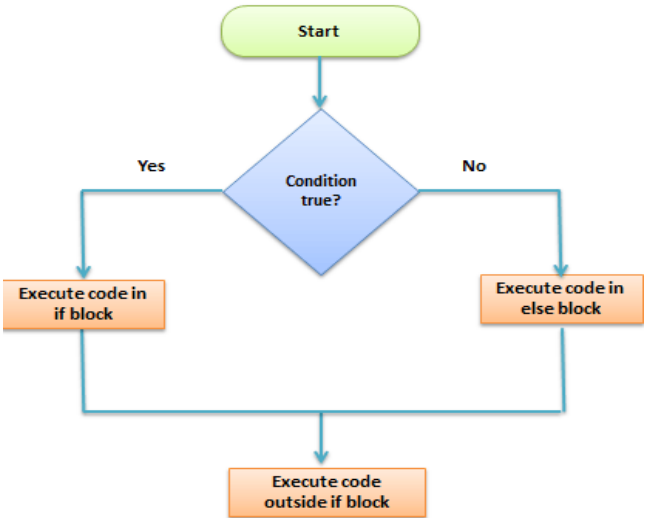
2. Take input of two numbers from the user and do the division of two numbers?



1.15.2 Selection (Branch or conditional):

- It's asked a true/false question THEN selects the next instruction based on the answer.
- It selects a statement to execute on the basis of **condition**. Statement is executed when the condition is true and ignored when it is false

Example: if, if else, switch structures.



Example problem:

3. Draw a flowchart to determine greatest among two numbers?

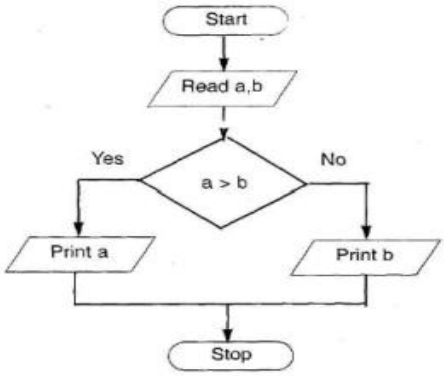
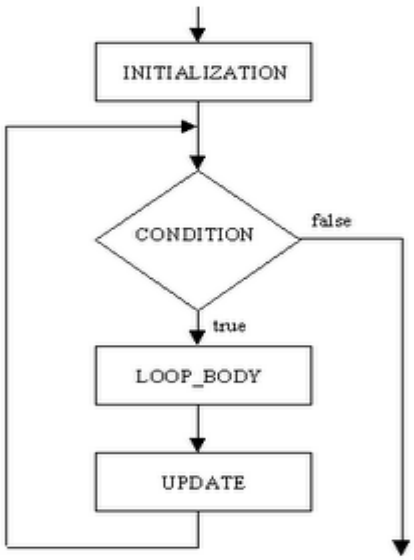


Fig. 2b) Flowchart to determine the greater of two numbers a and b

1.15.3 Iterative (loop):

- Its repeat the execution of a block of instruction.
- In this structure the statements are executed more than one time. It is also known as iteration or loop
- A loop structure is used to execute a certain set of actions for a predefined number of times or until a particular condition is satisfied

Example: while loop, for loop do-while loops etc.

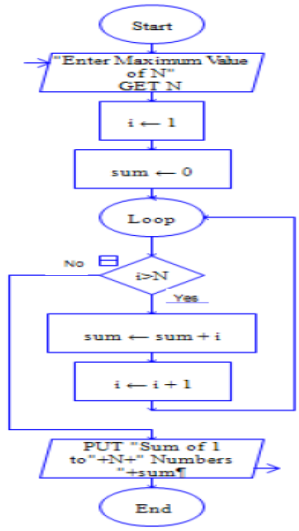


Iterative Structure

Example problem:

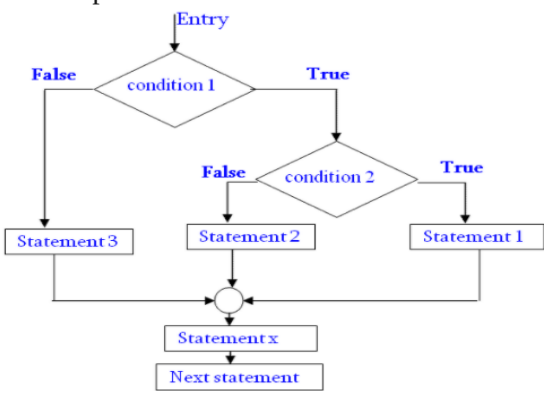
Write an algorithm and flowchart to find Total Sum from 1 to 100 natural numbers

Step 1: Start
Step 2: Read value of n
Step 3: Set i←1
Step 4: Set Total_Sum← 0
Step 5: Repeat the steps until i greater than n
a) Set Total_Sum←Total_Sum+i
b) increment i // (i.e. i=i+1)
Step 6: Display "1 to 100 Natural Numbers Sum:" Total_Sum
Step 7: End



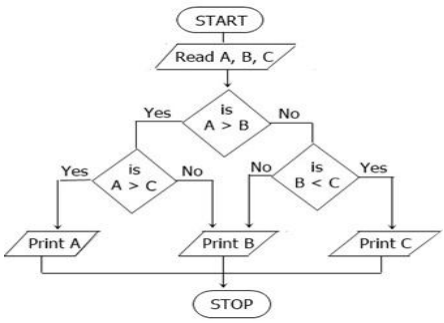
Nested Control Structures:

A nested control statement is a control statement that is contained within another control statement. You can do this to many levels. You can place control statements inside other control statements, for example an If...Then...Else block within a for. Next loop. A control statement placed inside another control statement is said to be ***nested*.**



Example problem:

1. Draw a flowchart to determine largest among three numbers?



1.16 Pseudo code

A pseudo code is an informal way of writing a program. However, it is not a computer program. It only represents the algorithm of the program in natural language and mathematical notations. Besides, there is no particular programming language to write a pseudo code. Unlike in regular programming languages, there is no syntax to follow when writing a pseudo code. Furthermore, it is possible to use pseudo codes using simple English language statements.

- Pseudo code is a kind of structured English for describing algorithm.
- It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.
- No syntax for Pseudo code
- Not executable program

1.17 Advantages of Pseudo code

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.

- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer

1.18 How to write a Pseudo-code?

- To begin the comment double forward slash are used "//".
- Matching braces "**{**and**}**" are used to present blocks where a compound statement (set of simple statements) can be illustrated as a block and terminated by a semicolon"**;**". The body of a procedure constructs a block as well.
- All the identifiers start with a letter and the data type of the variables are not declared explicitly.
- An assignment statement is used for the assigning values to the variables.
- To produce the Boolean values (i.e., true and false) the logical operators and, or and not and the relational operators <, ≤, =, =, ≥ and > are provided.
  Input and output are presented by read and write instructions.

Example:

1. A pseudo code to find the total of two numbers is as follows.

```
Sum Of Two Numbers ()
begin
Set sum =0;
Read: number 1, number 2;
Set sum = number1 + number 2;
Print sum;
End
```

2. A pseudo code to find the area of a triangle is as follows.

```
AreaofTrinagle()
Begin
Read: base,height;
Set area =0.5*base*height;
Print area;
End
```

1.19 Differences between algorithm and Flowchart:

Algorithm	Flow chart
Step by step instruction representing the process of any solution.	Block by block information diagram representing the data flow.
It is step wise analysis of the work to be done	It is a pictorial representation of a process
Solution is shown in non computer language like English.	Solution is shown in graphical format.
It is something difficult to understand.	Easy to understand as compared to algorithm.
Difficult to show branching and looping.	Easy to show branching and looping.
Algorithm can be written for any problem.	Flow chart for big problem is impractical.
Easy to debug errors.	Difficult to debug errors.
It is difficult to write algorithm as compared to flowchart.	It is easy to make flowchart.

1.20 Differences between algorithm and Pseudocode:

Basis for comparison	Algorithm	Pseudo code
Comprehensibility	Quite hard to understand	Easy to interpret
Uses	Complicated programming language	Combination of programming language and natural language.
Debugging	Moderate	simpler
Ease of construction	Complex	Easier

1.21 Difference between flow chart and pseudo code:

Flow chart	Pseudo code
A diagrammatic representation that illustrates a solution model to a given problem.	An informal high – level description of the operating principle of an algorithm.
Written using various symbols.	Written in natural language and mathematical notations help to write pseudo code.

Solved problems:

1. Write an algorithm to go for class picnic.

Algorithm:

Step 1: Start
Step 2: Decide the picnic venue, date and time
Step 3: Decide the picnic activities
Step 4: Hire a vehicle to reach to the venue and comeback
Step 5: Goto to the picnic venue on the decided date
Step 6: Do the activities planned for the picnic
Step 7: Come back to school in the hired vehicle
Step 8: Stop

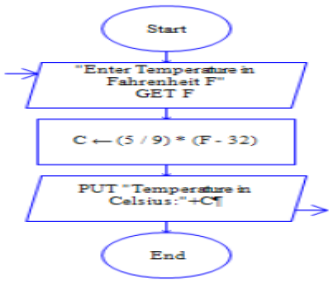
2. To celebrate Teachers' Day

Algorithm:
Step 1: Start
Step 2: Decide the activities for teachers' day like dance performances, plays, etc.
Step 3: Form groups of students and assign the decided activities from step 2 to each group.
Step 4: Decide the practice timings for each group.
Step 5: Each group to practice as per the timings decided in step 4.
Step 6: Invite the teachers to Teachers' Day celebrations.
Step 7: Perform the activities planned in step 2 on Teachers' Day
Step 8: Stop

3. To celebrate New Year

Algorithm:
Step 1: Start
Step 2: Prepare a guest list for New Year party
Step 3: Decide the venue, food menu, games and fun activities for the party
Step 4: Invite the guests for the party
Step 5: On New Year eve, get ready and enjoy the party
Step 6: Stop

4. Convert temperature Fahrenheit to Celsius with flowchart



Inputs to the algorithm: Temperature in Fahrenheit
Expected output: Temperature in Celsius

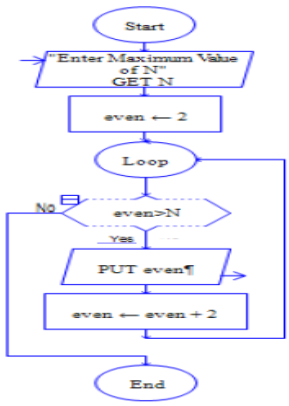
Algorithm:
Step1: Start
Step2: Read Temperature in Fahrenheit F
Step3: Set C ←5/9*(F-32)
Step4: Print "Temperature in Celsius: "+C
Step5: End

5. Write an algorithm to read two numbers and find their product?

Inputs to the algorithm: First num1. Second num2.
Expected output: Sum of the two numbers.

Algorithm:
Step1: Start
Step2: Read\input num1.
Step3: Read\input num2.
Step4: set product ← num1 * num2 // calculation of product
Step5: Print product.
Step6: End

6. An algorithm to display even numbers between 0 and 99 with flowchart



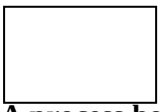
Step 1: Start
Step2: input/read value of n
Step3: Set even←0
Step4: Do the following until even < 100
a) Display even
b) Set even ←even+2
Step5: End

7. Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

Step 1: Start

Step2: input/read value of n
Step3: Set even 1000
Step4: Do the following until even < 2000
a) Display even
b) Set even even+2
Step5: End

8. Define the following symbol and use?



Answer: Process Box: A process box is used to represent all types of mathematical tasks like addition, subtraction, multiplication, division, etc.

9. Algorithm to find area of a rectangle?

Algorithm:

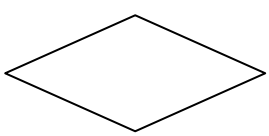
Step 1: Start
Step 2: Take length and breadth and store them as L and B? // Input
Step 4: Multiply L and B and store it in area //Area of Rectangle = L*B
Step 5: Print area //Output
Step 6: Stop.

10. Write an algorithm to find the largest among three different numbers entered by user with flowchart

Step 1: Start
Step 2: Read variables a, b and c.
Step 3: If a greater than b
a) If a greater than c
i) Display a is the largest number.
b) else
i) Display c is the largest number.
Step 4: else
a) If b greater than c
i) Display b is the largest number.
b) else
i) Display c is the greatest number.
Step 5: Stop

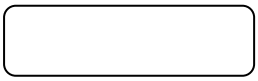
Multiple Choice Questions:

1) An Algorithm is expressed by
a) Flowchart b) common language c) pseudo code d) all
2) A good algorithm having finite steps?
a) True b) False
3) Algorithm efficiency is measured by?
a) Speed b) space c) code d) all of the above
4) How many control structures are there in algorithm?
a) 3 b) 2 c) 1
5) Which control structure used weather a condition is true or false?
a) Selection b) iteration c) sequence d) none
6) Which control structure used to check a condition more than one time?
a) Iteration b) loop c) a & b d) None of the above
7) In computer science, algorithm refers to a pictorial representation of a flowchart.
a) True b) False
8) The process of drawing a flowchart for an algorithm is called _____
a) Performance b) Evaluation c) Algorithmic Representation d) Flowcharting
9) Actual instructions in flowcharting are represented in _____
a) Circles b) Boxes c) Arrows d) Lines
10) A box that can represent two different conditions.
a) Rectangle b) Diamond c) Circle d) Parallelogram



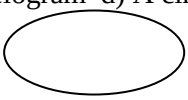
11) The following box denotes?
a) Decision b) Initiation c) Initialization d) I/O
12) There should be certain set standards on the amount of details that should be provided in a flowchart
a) True b) False
13) Which of the following is not an advantage of a flowchart?
a) Better communication b) Efficient coding c) Systematic testing d) Improper documentation

14) The symbol denotes _____



a) I/O n b) Flow c) Terminal d) Decision

15) Pseudo code is an informal high-level description of an algorithm.
    a) True b) false
16) In a flowchart a calculation(process) is represented by
    a) A rectangle b) A rhombus c) A parallelogram d) A circle
17) The symbol denotes_____



    a) I/O b) Flow c) Loop d) Decision
18) To repeat a task a number times
    a) Loop statement b) Input statement c) Conditional statement d) Output statement
19) In a flow chart how are the symbols connected?
    a) Symbols do not get connected together in a flowchart
    b) With lines and arrow to show the direction of flow
    c) With dashed lines and numbers
    d) With solid lines to link events
20) A flow chart does need to have a start?
    a) True b) False

Unsolved Problems:

1) Write an algorithm and flow chart to add four numbers?
2) Write an algorithm to make tea/coffee
3) Write 6 No's Algorithms for performing day to day activities.
4) Write an algorithm and flow chart to find large number among given two numbers.
5) Draw the flow chart to convert distance entered in Km to Meters.
6) Accept the age of a person and check whether he/she is eligible to vote or not. A person is eligible to vote only when he/she is 18 years or more.
7) Draw the flow chart to find the area of a rectangle whose length and breadth are given
8) Draw the flow chart to find the average of three numbers a, b and c.
9) Write an algorithm and flow chart Print 1 to 100?
10) Write an algorithm to find average of three numbers?

Unit 2 - Introduction to Python Programming

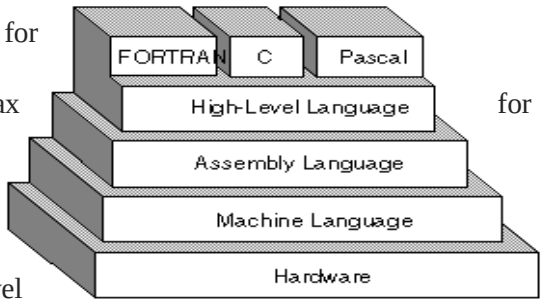
Module 1 - What is Python Programming Language and its Classifications

2.1 What is a Language?

- Language is the method of human communication, either spoken or written, It consisting of the use of words in a structured and conventional way.
- The Language is nothing but set of instructions we are used to communicate.
- To communicate a particular person, we are passing instruction using a particular language like English, Telugu, and Hindi…. etc.
- But while using a language we need to follow some of instructions, some rules are already they have given.
- What are the rules? If I want to speak in English, to form a sentence, first we should be good at grammatically, are else we cannot form a sentence, to speak in English language.
- Similarly, computer language is also for communication sake only

2.2 What is Programming Language?

- Programming Language is also like English, Telugu…etc.
- It should contain vocabulary and set of grammatical rules for instructing a computer to perform specific tasks.
- Each language has a unique set of keywords and a special syntax for organizing program instructions
- Programming languages are classified as:
- Machine language, Assembly language and High-level language
- The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal



2.3 Types of Programming Languages

2.3.1 Machine Language: The language of 0s and 1s is called as machine language. This is the only language which can be understood computers directly.

Merits:

- It is directly understood by the processor so has faster execution time since the programs written in this language need not to be translated.
- It doesn't need larger memory.

Demerits:

- It is very difficult to program using Machine Language since all the instructions are to be represented by 0s and 1s.
- Use of this language makes programming time consuming.
- It is difficult to find error and to debug.
- It can be used by experts only.

2.3.2 Assembly Languages: It is low level programming language in which the sequence of 0's and 1's are replaced by mnemonic (ni-monic) codes. Typical instructions for addition and subtraction.

Example: ADD for addition, SUB for subtraction etc.

Since our system only understands the language of 0s and 1s .therefore, a system program is known as assembler. Which is designed to translate an assembly language program into the machine language program?

Merits:

- It is makes programming easier than Machine Language since it uses mnemonics code for programming. Eg: ADD for addition, SUB for subtraction, DIV for division, etc.
- It makes programming process faster.
- Error can be identified much easily compared to Machine Language
- It is easier to debug than machine language.

Demerits:

- Programs written in this language is not directly understandable by computer so translators should be used.

- It is hardware dependent language so programmers are forced to think in terms of computer's architecture rather than to the problem being solved.
- Being machine dependent language, programs written in this language are very less or not portable.
- Programmers must know its mnemonics codes to perform any task.

2.3.3 High Level Language: High level languages are English like statements and programs Written in these languages are needed to be translated into machine language before execution using a system software such as compiler. Program written in high level languages are much easier to maintain and modified.

- High level language program is also called source code.
- Machine language program is also called object code

Merits:

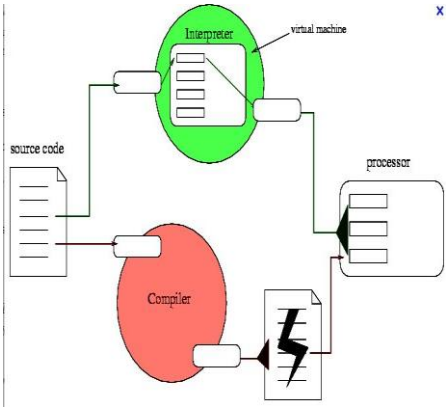
- Since it is most flexible, variety of problems could be solved easily
- Programmer does not need to think in term of computer architecture which makes them focused on the problem.
- Programs written in this language are portable.

Demerits:

- It is easier but needs higher processor and larger memory.
- It needs to be translated therefore its execution time is more.

2.4 Interpreter and Compiler

- We generally write a computer program using a high-level language. A high-level language is one which is understandable by humans.
- But a computer does not understand high-level language. It only understands program written in 0's and 1's in binary, called the machine code or object code.



A program written in high-level language is called a source code. We need to convert the source code into machine code and this is accomplished (actioned) by compilers and interpreters.

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

Module 2 – Introduction to Python Programming

2.5 Introduction

- Python is an interpreter, object-oriented, high-level programming language with dynamic semantics (substance).
- Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.
- Python supports modules and packages, which encourages program modularity and code reuse
- It consists of high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.
- Python is an open-source programming language made to both look good and be easy to read.

2.6 History of Python

- Python is an old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.
- Python is influenced by following programming languages:
    - ABC language.
    - Modula-3

2.7 Why Python was created?

- In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group.

- He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls.
- So, he decided to create a language that was extensible. This led to a design of new language which was later named Python.
- Python drew inspiration from other programming languages like C, C++, Java, Perl, and Lisp.

2.8 Why the name was Python?

- No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series Monty Python's Flying Circus in late seventies.
- The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Release Dates of Different Versions

Version	Release Data
Python1.0(firststandardrelease)	January1994
Python1.6 (Lastminorversion)	September5,2000
Python2.0(Introducedlistcomprehensions)	October 16,2000
Python2.7 (Last minorversion)	July3, 2010
Python3.0(Emphasis onremovingduplicativeconstructs and module)	December 3, 2008
Python3.5	September13,2015
Python 3.7	June 27, 2018
Python 3.8.0 (Last updated version)	Oct. 14, 2019

29Features of Python Programming

A simple language which is easier to learn Python has a very simple and elegant (graceful) syntax. It's much easier to read and write Python programs compared to other languages like: C++,Java, C#.

Free and open-source

- You can even make changes to the Python's source code and update.

Portability

- You can move Python programs from one platform to another, and run it without any changes. It runs smoothly on almost all platforms including Windows, Mac OS X and Linux

Extensible and Embeddable

- Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code and other languages may not provide out of the box

A high-level, interpreted language

- Unlike C/C++, you don't have to worry about daunting (Cause to lose courage) tasks like memory management, garbage collection and so on, likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations

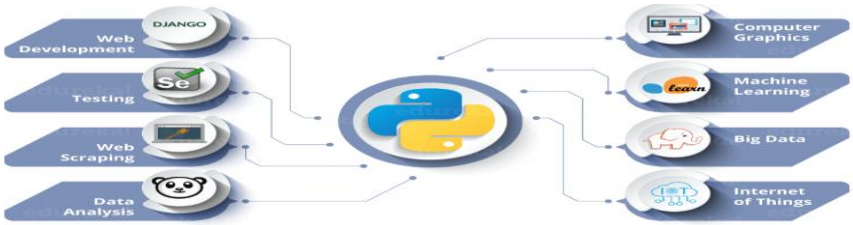
Large standard libraries to solve common tasks

- Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself
- For example: Need to connect **MySQL** database on a Web server? You can use **MySQLdb** library using **import MySQLdb**

Object-oriented

- Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.
- With OOP, you are able to divide these complex problems into smaller sets by creating objects.

Images For Applications of Python



Web Applications

- You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python.
- Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS.
- Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

Scientific and Numeric Computing

- There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: **SciPy** and **NumPy** that are used in general purpose computing. And,
- There are specific libraries like: **EarthPy** for earth science, **AstroPy** for Astronomy and so on.
- Also, the language is heavily used in machine learning, data mining and deep learning.

2.10 Why Python is very easy to learn?

- One big change with Python is the use of white space to define code: spaces or tabs are used to organize code by the amount of spaces or tabs.
- This means at the end of each line; a **semicolon** is not needed and curly braces ({}) are not used to group the code.
- Which are both common in C. The combined effect makes Python a very easy to read language.

4 Reasons to Choose Python as First Language

2.10.1 Simple Elegant (Graceful) Syntax

- It's easier to understand and write Python code.
- Why? Syntax feels Naturals with Example Code

```
A=12
B=23
sum=A+B
print(sum)
```

- Even if you have never programmed before, you can easily guess that this program adds two numbers and prints it.

2.10.2 Not overly strict

- You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.
- Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

2.10.3 Expressiveness of the language

- Python allows you to write programs having greater functionality with fewer (less) lines of code.
- We can build game **(Tic-tac-toe)**with Graphical interface in less than 500 lines of code
- This is just an example. You will be amazed how much you can do with Python once you learn the basics.

2.10.4 Great Community and Support

- Python has a large supporting community.
- There are numerous active forums online which can be handy if you are stuck

2.11 Install and Run Python

Ubuntu

1. Install the following dependencies:
   $sudo apt-get install **build-essential** check install
   $sudo apt-get install**libsqlite3-dev**
   $sudo apt-get install**libbz2-dev**
   (libreadline-gplv2-devlibncursesw5-dev**libssl-dev**tk-dev**libgdbm-dev**libc6-dev)
   Video Link: -https://www.youtube.com/watch?v=sKiDjO_0dCQ
2. Go to Download Python page on the official site and click **DownloadPython3.4.3**
3. In the terminal, go to the directory where the file is downloaded and run the command:
   i. $tar-xvfPython-3.4.3.tgz
   ii. This will extract your zipped file.

Note: The filename will be different if you've downloaded a different version. Use the appropriate filename

4. Go to the extracted directory.
   $ cd Python-3.4.3
5. Issue the following commands to compile Python source code on your Operating system
   $./configure
   $ make
   $ make install
6. Go to Terminal and type the following to run sample 'helloworld' Program
   ubuntu@~$python3.4.3
   >>>print('HellowWorld')
   Hello World

Windows

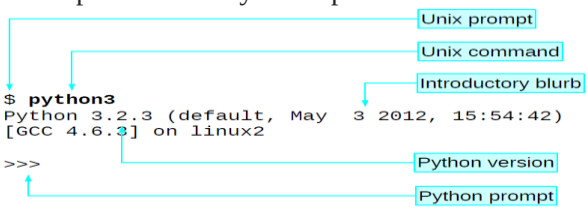
1. Go to Download Python page on the official site and click Download Python 3.4.3
2. When the download is completed, double-click the file and follow the instructions to install it
3. When Python is installed, a program called IDLE is also installed along with it. It provides graphical user interface to work with Python
4. Open IDLE, copy the following code below and press enter
   print ("Hello, World ")
5. To create a file in IDLE, go to File > New Window (Shortcut: Ctrl+N).
6. Write Python code (you can copy the code below for now) and save (Shortcut: Ctrl+S) with .py file extension like: hello.py or your-first-program.py
   print ("Hello, World ")
7. Go to Run > Run module (Shortcut: F5) and you can see the output. Congratulations, you've successfully run your first Python program

2.12 Modes of running

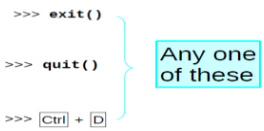
There are various ways to start Python

Immediate Mode or Interactive Mode

- Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output **(>>>)**
- Is the python prompt and it tells us interpreter is ready for input

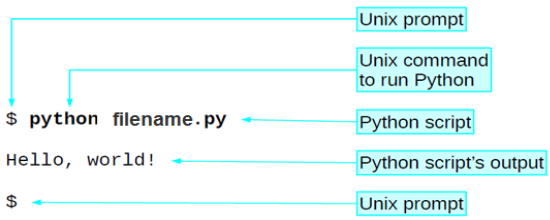


Quitting Python



Script Mode

- This mode is used to execute Python program written in a file. Such a file is called a script. Scripts can be saved to disk for future use. Python scripts should have the extension .py, it means that the filename ends with **.py**.
- For example: **helloWorld.py**
- To execute this file in script mode we simply write **python3 helloWorld.py** at the command prompt.



Integrated Development Environment (IDE)

- We can use any text editing software to write a Python script file. Like Notepad, Editplus, sublime…etc

2.13 Python Program to Print Helloworld!

- Type the following code in any text editor or an IDE and Save it as **helloworld.py**
- Now at the command window, go to the location of this file, use **cd** command to **change directory**
- To run the script, type **python3helloworld.py** in the command window then we get output like this: **HelloWorld**

- In this program we have used the built-in function **print(),** to print out a string to the screen
- String is the value inside the quotation marks **i.e. HelloWorld**

Let us execute programs in different modes of programming.

Interactive Mode Programming:

Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

```
$ python3
Python 3.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
```

Type the following text at the Python prompt and press the Enter:

```
>>> print "Hello, IIIT RK Valley, RGUKT-AP!";
```

If you are running new version of Python, then you need to use print statement with parenthesis as in print ("Hello, IIIT RK Valley");.However in Python version 2.7, this produces the following result:

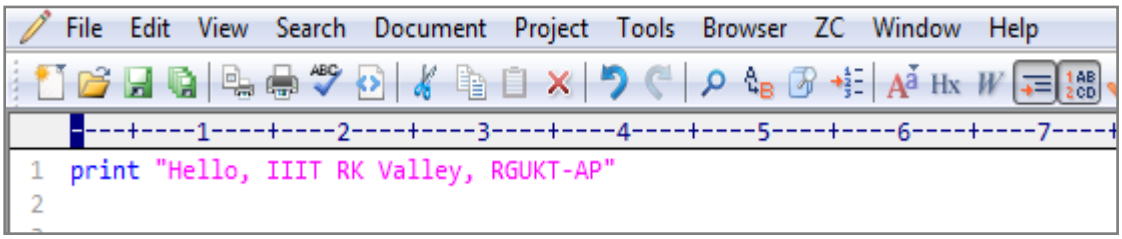
```
Hello, IIIT RK Valley, RGUKT-AP!
```

Script Mode Programming:

Python programs must be written with a structure. The syntax must be correct, or the interpreter will generate error messages and not execute the program. This section introduces Python by providing a simple example program.

To write Python programming in script mode we have to use editors. Let us write a simple Python program in a script mode using editors. Python files have extension .py. Type the following source code in a simple.py file.

Program 2.1 (simple.py) is one of the simplest Python programs that does something



We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows:

```
$ python3 simple.py
```

This produces the following result:

```
Hello, IIIT RK Valley, RGUKT-AP!
```

Module 3 - Reserved key words, Identifiers, Variables and Constant

2.14 Keywords

- Keywords are the reserved words in Python and 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 35 keywords in Python 3.7.3 This number keep on growing with the new features coming in python
- All the keywords except True, False and None are in lowercase and they must be written as it is.
- The list of all the keywords is given below

We can get the complete list of keywords using python interpreter help utility.

```
$ python3
>>>help()
help> keywords
```

2.15 Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python.

Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to9) or an underscore (_).
- Names like myClass, var_1and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit.1variable is invalid, but variable1is perfectly fine.
- Keywords cannot be used as identifiers.

```
>>> global=1File"<interactiveinput>",line1
global=1
^SyntaxError: invalid syntax
```

- We cannot use special symbols like!, @, #, $,% etc. in our identifier. >>>a@ =0

```
File"<interactive input>", line1a@= 0^
Syntax Error: invalid syntax
```

- Identifier can be of any length.

2.16 Things to care about

- Python is a case-sensitive language. This means, **Variable** and **variable** are not the same. Always give a valid name to identifiers so that it makes sense.
- While, c = 10is valid. Writing count =10would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.
- Multiple words can be separated using an underscore, this_is_a_long_variable.
- We can also use camel-case style of writing,
- i.e., capitalize every first letter of the word except the initial word without any spaces.
- For example: camelCase Example.

2.17 Variable

- A variable is a location in memory used to store some data.
- Variables are nothing but reserved memory locations to store values, this means that when we create a variable, we reserved some space in memory.
- They are given unique names to differentiate between different memory locations.
- The rules for writing a variable name are same as the rules for writing identifiers in Python.
- We don't need to declare a variable before using it.
- In Python, we simply assign a value to a variable and it will exist.
- We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable assignment: We use the assignment operator (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

```
a= 5
b = 3.2
c = "Hello"
```

Here, we have three assignment statements. 5 is an integer assigned to the variable a. Similarly, 3.2 is a floating-point number and "Hello" is a string (sequence of characters) assigned to the variables b and c respectively.

Multiple assignments:

- In Python, multiple assignments can be made in a single statement as follows: a, b, c = 5,3.2, "Hello"
- If we want to assign the same value to multiple variables at once, we can do this as x = y= z ="same"
- This assigns the "same" string to all the three variables.

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. You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag.

Assigning value to constant in Python: In Python, constants are usually declared and assigned in a module. Here, the module is a new file containing variables, functions, etc which is imported to the main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example 1: 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)
```

Output

```
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.

Note: In reality, we don't use constants in Python. Naming them in all capital letters is a convention to separate them from variables; however, it does not actually prevent reassignment.

2.18 Statements & Comments

Instructions that a Python interpreter can execute are called statements.

For example,

a = 1 is an assignment statement. If statement, for statement, while statement etc.

2.18.1 Multi-line statement

- ❖ In Python, end of a statement is marked by a new line character. But we can make a statement extend over multiple lines with the line continuation character (\).

For example:

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

This is explicit line continuation.

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

```
a = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)
```

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

For example: colors = ['red',
'blue',
'green']

- ❖ We could also put multiple statements in a single line using semicolons, as follows a= 1; b=2;c =3

2.18.2 Comments

- Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out.
- You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.
- In Python, we use the hash (#) symbol to start writing a comment. It extends up to the new line character.
- Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

```
#this is a comment
#print out Hello
print('Hello')
```

Multi-linecomments

- If we have comments that extend multiple lines, one way of doing it is to use hash(#) in the beginning of each line. For example:

```
#this is a long comment
#and it extends
#to multiple lines
```

- Another way of doing this is to use triple quotes, either '''or""".
- These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well. Unless they are not docstrings, they do not generate any extracode.

```
"""This is also a
perfect example of
multi-line comments"""
```

Module 4 - Python Operators

2.19 Get Started with Python Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example: >>> 2+3

- Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.
- Python has a number of operators which are classified below.
  - Arithmetic operators
  - Assignment operators
  - Comparison (Relational) operators
  - Logical (Boolean) operators
  - Bitwise operators
  - Assignment operators
  - Special operators

2.19.1 Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x – y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the	x // y

	left in the number line	
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Example Program for Arithmetic operators

```
x = 25
y = 15
print('x + y = ',x+y)
print('x - y = ',x-y)
print('x * y = ',x*y)
print('x / y = ',x/y)
print('x // y = ',x//y)
print('x ** y = ',x**y)
```

2.19.2 Comparison (Relational) Operators

Comparison operators are used to compare values. It either returns True or False according to the condition

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example: x = 10;y = 12

```
print('x > y  is',x>y)
```

2.19.3 Logical (Boolean) Operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Here is an example. x = True y = False; print('x and y is',x and y); print('x or y is',x or y) ; print('not x is',not x)

2.19.4 Bitwise Operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.
For example, 2 is 10 in binary and 7 is 111.
In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

2.19.5 Assignment Operators

Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable *a* on the left. There are various compound operators in Python
Example: a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

2.19.6 Membership Operators

in and **notin** are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Here is an example.

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)
print('hello' not in x)
print(1 in y)
print('a' in y)
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

2.20 Rules for solving equations in Python

Order of Operations Worksheets - BEDMAS or PEMDAS

Step 1: First, **perform** the operations within the brackets or parenthesis.

Step 2: Second, evaluate the exponents.

Step 3: Third, **perform** multiplication and division from left to right.

Step 4: Fourth, **perform** addition and subtraction from left to right.

Example: >>>(40+20)*30/10

Output: 180

2.21 Reading Input value from the User

The print() function enables a Python program to display textual information to the user. Python provides built-in functions to get input from the user. The function is input(). Generally input() function is used to retrieve string values from the user.

Program 3.6 (sampleinput.py) shows the type of user enter values

```
x = input("Enter Value : ")
print(type(x))

y = int(input("Enter Value : "))
print(type(y))
```

Program 3.6 (sampleinput.py) produce result as

```
Enter Value : 4
<class 'str'>
Enter Value : 4
<class 'int'>
```

We can use the above mentioned functions in *python 2.x*, but not on *python 3.x*. input() in python 3.x always return a string. Moreover, raw_input() has been deleted from python 3

python2.x		python3.x
raw_input()	----------	input()
input()	----------	eval(input())

We can simply say in *python 3.x* only use of the input () function to read value from the user and it assigns a string to a variable.

```
x = input()
```

The parentheses are empty because, the input function does not require any information to do its job.

Program 3.7 (usinginput.py) demonstrates that the input function produces a string value.

```
1  print('Please enter some text:')
2  x = input()
3  print('Text entered:', x)
4  print('Type:', type(x))
```

The following shows a sample run of Listing 2.11 (usinginput.py):

```
>>>
Please enter some text:
Hi RGUKT
('Text entered:', 'Hi RGUKT')
('Type:', <type 'str'>)
```

The second line shown in the output is entered by the user, and the program prints the first, third, and fourth lines. After the program prints the message *Please enter some text:*, the program's execution stops and waits for the user to type some text using the keyboard. The user can type, backspace to make changes, and type some more. The text the user types is not committed until the user presses the Enter (or return) key. In Python 3.X, input() function produces only strings, by using conversion functions we can change the *type* of the input value. Example as int(), str() and float().

Exercise

1) What is a programming language?
2) Explain different types of programming languages?
3) What is a compiler?
4) What is an interpreter?
5) How is compiled or interpreted code different from source code?
6) Difference between Interpreter and Compiler?
7) What is Python Programming Language?
8) How many ways can run the Python Programming?
9) What is print statement
10) What is prompt
11) Print "Welcome to Python Programming Language" from interactive Python.
12) Write above statement in exercise1.py to print the same text.
13) Run the modified exercise1.py script.
14) Write the below statements in exercise2.py to print the same below
15) What is a variable?
16) What is value? How can you define the type of value?
17) What is assignment statement?
18) What is comment?
19) What is expression?
20) Write a program that uses raw_input to prompt a user for their Name and then welcomes them.
21) Write a program to prompt the user for hours and rate per hour to compute gross pay.
22) Assume that we execute the following assignment statements: width = 17, height = 12.0

    For each of the following expressions, write the value of the expression and the type (of the value of the expression).

    (a) width/2
    (b) width/2.0
    (c) height/3
    (d) 4. 1 + 2 * 5

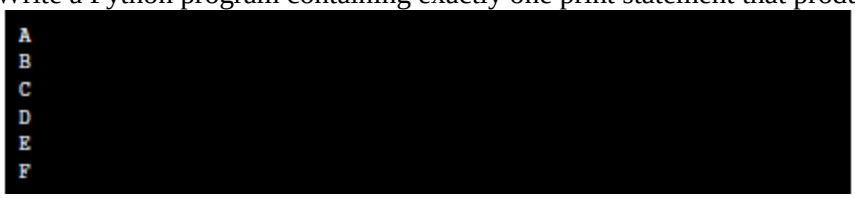
23) Write program and also Use the Python interpreter to check your answers.
24) Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.
25) Will the following lines of code print the same thing? Explain why or why not.

```
x = 6
print(6)
print("6")
```

26) What happens if you attempt to use a variable within a program, and that variable has not been assigned a value?
27) What is wrong with the following statement that attempts to assign the value ten to variable x? 10 = x
28) In Python can you assign more than one variable in a single statement?
29) Classify each of the following as either a legal or illegal Python identifier:

    a. Flag
    b. if
    c. 2x
    d. -4
    e. sum_total
    f. sum-total
    g. sum total
    h. sumtotal
    i. While
    j. x2
    k. $16
    l. _static
    m. wilma's

30) What can you do if a variable name you would like to use is the same as a reserved word?
31) What is the difference between the following two strings? 'n' and '\n'?
32) Write a Python program containing exactly one print statement that produces the following output



33) Explain type of operators
34) Write a python program to find type of the value of below expressions
    - ✓ 4 > 23
    - ✓ 5+=21
    - ✓ 2**=3

35) Find whether these expressions will evaluate to **True** or **False**. Then try them in interactive mode and script mode
   - ✓ 4 > 5
   - ✓ 12 != 20
   - ✓ 4 <= 6
   - ✓ 4 => 1
   - ✓ 'sparrow' > 'eagle'
   - ✓ 'dog' > 'Cat' or 45 % 3 == 0
   - ✓ 42+12 < 34//2
   - ✓ 60 - 45 / 5 + 10 == 1 and 32*2 < 12

36) What is compound assignment operator? Explain with examples?
37) Difference between '=' and '=='
38) Difference between '/' and '//'
39) Write a python program to prompts the user for x and y operands and find result for all arithmetic operators.

Multiple Choice Questions

1) Python is -------
   a) Objected Oriented Programming Language
   b) Powerful Programming Language
   c) Powerful scripting Language
   d) All the above
2) Python is developed by ----------
   a) Guido van Rossum. b) Balaguruswami c) James Gosling d) None of the above
3) Which of the following is not a keyword?
   a) eval b) assert c) nonlocal d) pass
4) All keywords in Python are in _____
   a) lower case b) UPPER CASE c) Capitalized d) None of the mentioned
5) Which of the following is true for variable names in Python?
   a) unlimited length
   b) all private members must have leading and trailing underscores
   c) underscore and ampersand are the only two special characters allowed
   d) none of the mentioned
6) Which of the following translates and executes program code line by line rather than the whole program in one step?
   a) Interpreter b) Translator c) Assembler d) Compiler
7) Which of the following languages uses codes such as MOVE, ADD and SUB?
   a) assembly language b) Java c) Fortarn d) Machine language
8) What is the output of the following assignment operator?

```
y = 10
x = y += 2
print(x)
```

   a) 12 b) 10 c) Syntax error
9) What is assignment operator?
   a) = = b)= c) += d) - =
10) What is a correct Python expression for checking to see if a number stored in a variable x is between 0 and 5?
   a) x>0 and <5 b) x>0 or x<5 c) x>0 and x<5

Descriptive Questions

1) Explain about Applications and Features of Python Programming Language?
2) Write short notes on types of operators in python with appropriate examples?
3) Explain about interpreter and compiler?
4) Explain about identifiers
5) Explain about types of modes
6) Explain briefly about:
   - Constant and variables
   - keywords and statement

Solved Problems

1) Evaluate the value of z=x+y*z/4%2-1

```
x=input("Enter the value of x= ")
y=input("Enter the value of y= ")
z=input("Enter the value of z= ")
a=y*z
```

```
b=a/4
c=b%2
t=x+c-1
print("the value of z=x+y*z/4%2-1 is",t)
```

Input: Enter the value of x=2
Enter the value of y=3
Enter the value of z=4

Output: the value of z=x+y*z/4%2-1 is 2

Note: while solving equations follow the BEDMAS or PADMAS Rule

2) Python program to show bitwise operators

```
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
```

Unsolved Problems

1) Write a program to find out the value of 9+3*9/3?
2) Write a program to find out the value of (50-5*6)/4?
3) Write a program to find out the value of 3*3.75/1.5?
4) 45-20+(10?5)%5*3=25 what is the operator in the place of "?" mark?
5) 3*6/22%2+7/2*(3.2-0.7)*2 in this problem which one is evaluated first? finally what is the answer?
6) Write a program to display **Arithmetic** operations for any two numbers?
7) Python Program to find the average of 3 numbers?
8) Write a python program to find simple and compound interest?

Unit 3 - Data types, I/O, Types of Errors and Conditional Constructs

Module 1 - Data types, I/O, Types of Errors

A data type is a data storage format that can contain a specific type or range of values. Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories.

Numeric Types:	int, float, complex
Sequence Types:	str,list, tuple, range
Mapping Type:	Dict
Set Types:	set, frozenset

3.1.1 Numeric Types

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them

Example: x = 1 # int, y = 2.8 # float , z = 1j # complex

Int: Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example: x = 1, y = 35656222554887711, z = -3255522

Float: Float or "floating point number" is a number, positive or negative, containing one or more decimals.

Example: x = 1.10, y = 1.0, z = -35.59

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example: x = 35e3, y = 12E4, z = -87.7e100

Complex: Complex numbers are written with a "j" as the imaginary part.

Example: x = 3+5j, y = 5j, z = -5j

3.1.2 Sequence Types

In Python, sequence is the generic term for an ordered set. There are several types of sequences in Python; the following are the most important.

String: Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello". You can display a string literal with the print() function:

Example: print ("Hello"); print('Hello')

Assign String to a Variable, Assigning a string to a variable is done with the variable name followed by an equal sign and the string. Example: a = "Hello"; print(a)

Multiline Strings: You can assign a multiline string to a variable by using three quotes, You can use three double quotes. Example: a = """Lorem ipsum dolor sit amet, consecteturadipiscingelit, sed do eiusmodemporincididuntutlabore et dolore magna aliqua."""

print(a)

Or three single quotes:

a = '''Lorem ipsum dolor sit amet, consecteturadipiscingelit, sed do eiusmodtemporincididuntutlabore et dolore magna aliqua.'''

print(a)

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Lists: List is an ordered sequence of items. All the items in a list do not need to be of the same type and lists are mutable - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

a = [1, 2.2, 'python']

Tuples: Tuples is an ordered sequences of items same as a list. The only difference is that Tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically. It is defined within parentheses () where items are separated by commas.

t = (5,'program', 1+3j)

range(): The range() type returns an immutable sequence of numbers between the given start integer to the stop integer. print(list(range(10))

3.1.3 Mapping type

Dictionary is an unordered collection of key-value pairs. In python there is mapping type called dictionary. It is mutable. In Python, dictionaries are defined within braces {} with each item being a pair in the form key-value. Key and value can be of any type. The values of the dictionary can be any type, but the key must be of any immutable data type such as strings, numbers, and Tuples.

>>>d = {1:'value','key':2} >>>type(d)

3.1.4 Set types

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.

a = {5,2,3,1,4}
print("a = ", a) # printing set variable
print(type(a)) # data type of variable a

We can use the set for some mathematical operations like set union, intersection, difference etc. We can also use set to remove duplicates from a collection.

3.2. Getting the Data type

We can use the type() function to know which class a variable or a value belongs to. Similarly, the isinstance() function is used to check if an object belongs to a particular type.

a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(a,complex))

Output

5 is of type <class 'int'>
2.0 is oftype <class 'float'>
(1+2j) is complexnumber? True

3.3. Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

3.3.1 Implicit Type Conversion

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement. Let's see an example where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

Example 1: Converting integer to float.

```
num_int = 123
num_flo = 1.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Output

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

In the above program,

- We add two variables num_int and num_flo, storing the value in num_new.
- We will look at the data type of all three objects respectively.
- In the output, we can see the data type of num_int is an integer while the data type of num_flo is a float.
- Also, we can see the num_new has a float data type because Python always converts smaller data types to larger data types to avoid the loss of data.

Now, let's try adding a string and an integer, and see how Python deals with it.

Example 2: Addition of string(higher) data type and integer(lower) datatype

num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))
print(num_int+num_str)

When we run the above program, the output will be:

Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>
Traceback (most recent call last):
  File "python", line 7, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'

In the above program,

- We add two variables num_int and num_str.
- As we can see from the output, we got TypeError. Python is not able to use Implicit Conversion in such conditions.
- However, Python has a solution for these types of situations which is known as Explicit Conversion.

3.3.2 Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion. This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

Syntax: <required_datatype>(expression)

Typecasting can be done by assigning the required data type function to the expression.

Example 3: Addition of string and integer using explicit conversion

```
num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

When we run the above program, the output will be:

```
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
```

In the above program,

- We add num_str and num_int variable.
- We converted num_str from string(higher) to integer(lower) type using int() function to perform the addition.
- After converting num_str to an integer value, Python is able to add these two variables.
- We got the num_sum value and data type to be an integer.

Module 2 - Mutable and Immutable types

A first fundamental distinction that Python makes on data is about whether the value changes are not. If the value can change, then is called mutable, while if the value cannot change, that is called immutable.

Mutable:

- list,
- dict
- set

Immutable:

- int,
- float,
- complex,
- string,
- tuple

3.5 Input and Output Operations and Formats

Python provides numerous built-in functions that are readily available at the Python prompt.

Some of the functions like input() and print() are widely used for standard input and output operations respectively.

3.5.1 Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

Example 1:

```
print('This sentence is output to the screen')
```

Output

```
This sentence is output to the screen
```

Example 2:

```
a = 5
print('The value of a is', a)
```

Output

```
The value of a is 5
```

In the second print() statement, we can notice that space was added between the string and the value of variable a. This is by default, but we can change it.
The actual syntax of the print() function is:
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

- Here, objects are the value(s) to be printed.
- The sep(separator) is used between the values. It defaults into a space character.
- After all values are printed, end is printed. It defaults into a new line.
- The file is the object where the values are printed and its default value is sys.stdout (screen). Here is an example to illustrate this.

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='#', end='&')
```

Output

```
1 2 3 4
1*2*3*4
1#2#3#4&
```

3.5.2 Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

```
>>>x = 5; y = 10
>>>print('The value of x is {} and y is {}'.format(x,y))
The value of x is5and y is10
```

Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers.

```
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

Output

```
I love bread and butter
I love butter and bread
```

We can even use keyword arguments to format the string.

```
>>>print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
Hello John, Goodmorning
```

We can also format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

```
>>>x = 12.3456789
>>>print('The value of x is %3.2f' %x)
The value of x is12.35
>>>print('The value of x is %3.4f' %x)
The value of x is12.3457
```

3.5.3 Python Input

Until now, our programs were static. The value of variables was defined or hard coded into the source code.
To allow flexibility, we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is:

```
input([prompt])
```

where prompt is the string we wish to display on the screen. It is optional.

```
>>>num = input('Enter a number: ')
Enter a number: 10
>>>num
'10'
```

Here, we can see that the entered value 10 is a string, not a number. To convert this into a number we can use int() or float() functions.

```
>>>int('10')
10
>>>float('10')
10.0
```

3.6 Types of Errors in python

No matter how smart or how careful you are, errors are your constant companion. With practice, you will get slightly better at not making errors, better at finding and correcting them.There are three kinds of errors: syntax errors, runtime errors, and logic errors.

3.6.1 Syntax errors: Syntax errors are produced by Python when it is translating the source code into byte code. They usually indicate that there is something wrong with the syntax of the program.
Example: Omitting the colon at the end of if statement yields the somewhat redundant message.
Syntax Error: invalid syntax.
Here are some ways to avoid the most common syntax errors:

- Make sure you are not using a Python keyword for a variable name.
- Check that you have a colon at the end of the header of every compound statement, including for, while, if, and def statements.
- Check that indentation is consistent. You may indent with either spaces or tabs but it's better not to mix them. Each level should be nested the same amount.
- Make sure that strings in the code have matching quotation marks.
- If you have multiline strings with triple quotes (single or double), make sure you have terminated the string properly. An un-terminated string may cause an invalid token error at the end of your program, or it may treat the following part of the program as a string until it comes to the next string. In the second case, it might not produce an error message at all!
- An unclosed bracket – (, {, or [– makes Python continue with the next line as part of the current statement. Generally, an error occurs almost immediately in the next line.
- Check for the classic = instead of == inside a conditional.

3.6.2 Run Time Error: Errors that occur after the code has been executed and the program is running. The error of this type will cause your program to behave unexpectedly or even crash. An example of a runtime error is the division by zero. Consider the following example:

```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))
z = x/y
print (x,'dividedby',y,'equals: ',z)
```

The program above runs fine until the user enters 0 as the second number:

```
>>>
Enter a number: 9
Enter a number: 2
9.0 divided by 2.0 equals: 4.5
>>>
Enter a number: 11
Enter a number: 3
11.0 divided by 3.0 equals: 3.6666666666666665
>>> Enter a number: 5
Enter a number: 0
Traceback (most recent call last):
File "C:/Python34/Scripts/error1.py", line 3, in <module>
z = x/y
ZeroDivisionError: float division by zero
```

It is also called semantic errors, logical errors cause the program to behave incorrectly, but they do not usually crash the program. Unlike a program with syntax errors, a program with logic errors can be run, but it does not operate as intended. Consider the following example of logical error:

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

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

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

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

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

Module 3 - Conditional Constructs

There come situations in real life when we need to make some decisions and based on these decisions, we decide what we should do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.

Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

- if statement
- if..else statements
- nested if statements
- if-elif ladder
- Short Hand if statement
- Short Hand if-else statement

3.7.1 If statement: if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if condition:
  # Statements to execute if
  # condition is true
```

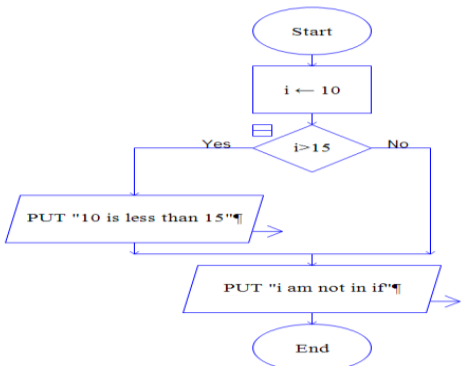
Here, condition after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not. We can use *condition* with bracket '(' ')' also.

As we know, python uses indentation to identify a block. So the block under an if statement will be identified as shown in the below example:

```
if condition:
  statement1
statement2
# Here if the condition is true, if block
# will consider only statement1 to be inside
# its block.
```

Flowchart:- # python program to illustrate If statement

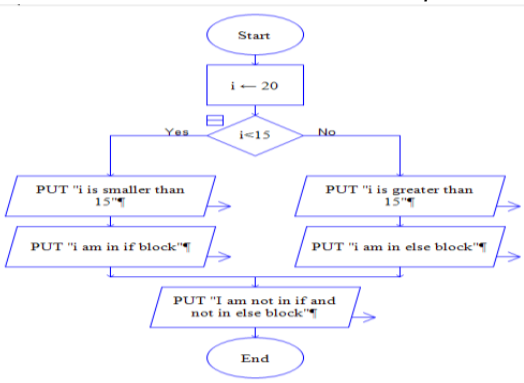
```
i = 10
if (i> 15):
  print ("10 is less than 15")
print ("I am Not in if")
```



Output:

```
I am Not in if
```

3.7.2 if- else: if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. Here comes the *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.



Syntax:

```
if (condition):
   # Executes this block if
   # condition is true
else:
   # Executes this block if
   # condition is false
```

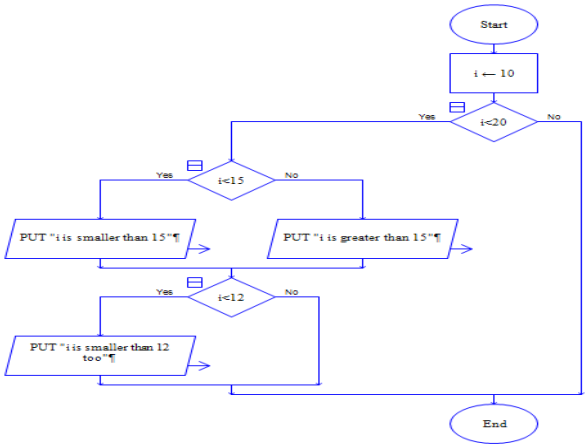
```
# Python program to illustrate If else statement
#!/usr/bin/python
i = 20;
if (i< 15):
   print ("i is smaller than 15")
   print ("i'm in if Block")
else:
   print ("i is greater than 15")
   print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

Output:

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

The block of code following the else statement is executed as the condition present in the if statement is false after call the statement which is not in block (without spaces).

3.7.3 nested-if: A nested if is if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.



Syntax:

```
if (condition1):
  # Executes when condition1 is true
  if (condition2):
    # Executes when condition2 is true
  # if Block is end here
# if Block is end here
```

```
# python program to illustrate nested If statement
#!/usr/bin/python
i = 10
if (i<20):
   #  First if statement
   if (i< 15):
      print ("i is smaller than 15")
   # Nested - if statement
   # Will only be executed if statement above
   # it is true
else:
      print ("i is greater than 15")
   if (i< 12):
      print ("i is smaller than 12 too")
```

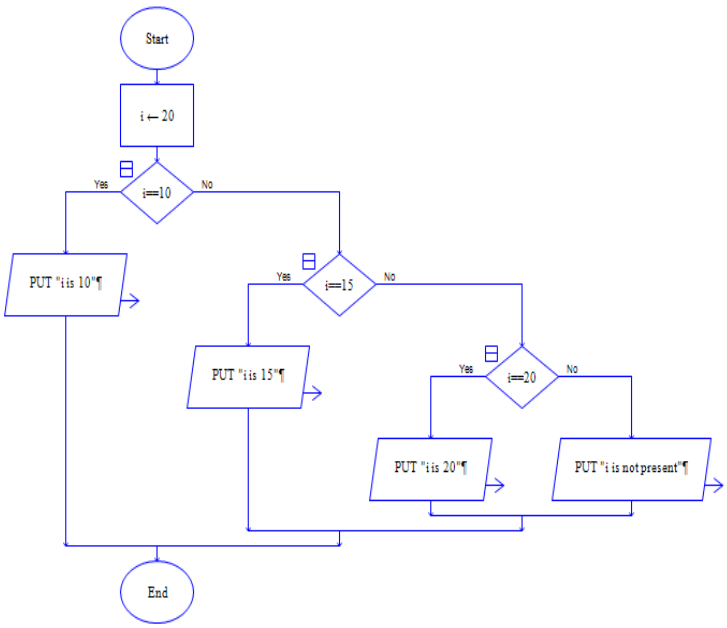
Output:

I. i is smaller than 15
II. i is smaller than 12 too

3.7.4 if-elif-else ladder: Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:-

```
if (condition):
   statement
elif (condition):
   statement
.
.
else:
   statement
```



Example:-

```
# Python program to illustrate if-elif-else ladder
#!/usr/bin/python
i = 20
if (i == 10):
   print ("i is 10")
elif (i == 15):
   print ("i is 15")
elif (i == 20):
   print ("i is 20")
else:
   print ("i is not present")
```

Output:

i is 20

3.8 Short Hand if statement

Whenever there is only a single statement to be executed inside if block then shorthand if can be used. The statement can be put on the same line as if statement.

Syntax:

if condition: statement

Example:

```
# Python program to illustrate short hand if
i = 10
if i< 15: print("i is less than 15")
```

Output:

i is less than 15

Short Hand if-else statement: This can be used to write the if-else statements in a single line where there is only one statement to be executed in both if and else block.

Syntax:

statement_when_True if condition else statement_when_False

Example: # Python program to illustrate short hand if-else

```
i = 10
print(True) if i< 15 else print(False)
```

Output:

True

Multiple Choice Questions

1) Which statement is correct?
   a) List is immutable && Tuple is mutable
   b) List is mutable && Tuple is immutable
   c) Both are Mutable.
   d) Both are Immutable.
2) Which one of the following is mutable data type?
   a) Set b) Int c) Str d) tupl
3) Which one of the following is immutable data type?
   a) List b) Set c) Int d) dict
4) Which of these is not a core data type?
   a) List b) Tuple c) Dictionary d) Class
5) Which one of the following is False regarding data types in Python?
   a) In python, explicit data type conversion is possible
   b) Mutable data types are those that can be changed.
   c) Immutable data types are those that cannot be changed.
   d) None of the above
6) Which of the following function is used to know the data type of a variable in Python?
   a) datatype() b) typeof() c) type() d) vartype()
7) Which one of the following is a valid Python if statement :
   a) if (a>=2): b) if (a >= 2) c) if (a => 22) d) if a >= 22
8) What keyword would you use to add an alternative condition to an if statement?
   a) else if b) elseif c) elif d) None of the above
9) Can we write if/else into one line in python?
   a) Yes b) No c) if/else not used in python d) None of the above
10) Which statement will check if a equal to b?
    a) if a = b: b) if a == b: c) if a === c: d)if a == b

Solved Problem Set

1) Write a Python program to find whether the given number is divisible by 7 and multiple of 5?

```
num=int(input("Enter a number: "))
if num%7==0 and num%5==0:
print(num, "is divisible by 7 and multiple of 5")
else:
print("the given number is either divisible by 7 or multiple of 5")
```

2) Write a program to input any number and check whether it is even or odd

```
num=int(input("Enter a number: "))
if num%2==0:
print("Even number")
else:
print("Odd number")
```

3) Write a program to input any number and check whether it is negative, positive or zero

```
num=int(input("Enter a number: "))
if num>0:
print("Positive number")
else if num<=0:
print("Negative number")
else:
   print("Zero")
```

4) A student will not be allowed to sit in exam if his/her attendance is less than 75%.
Take following input from user:
Number of classes held
Number of classes attended.
And print
percentage of class attended
Is student is allowed to sit in exam or not.

```
print "Number of classes held"
noh = input()
print "Number of classes attended"
noa = input()
atten = (noa/float(noh))*100
print "Attendence is",atten
ifatten>= 75:
  print "You are allowed to sit in exam"
else:
  print "Sorry, you are not allowed. Attend more classes from next time."
```

Un-Solved Problem Set

1) Take values of length and breadth of a rectangle from user and check if it is square or not.
2) Take two int values from user and print greatest among them
3) A shop will give discount of 10%, if the cost of purchased quantity is more than 1000.
Ask user for quantity, suppose, one unit will cost 100.
Ex: quantity=11, cost=11*100=1100>1000, so, he will get 10% discount, that is 110,
Final cost is 1100-110=990
4) A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years. Ask user for their salary and year of service and print the net bonus amount.
5) A school has following rules for grading system:

a. Below 25 – F	d. 50 to 60 - C
b. 25 to 45 – E	e. 60 to 80 - B
c. 45 to 50 – D	f. Above 80 - A

Ask user to enter marks and print the corresponding grade.
6) Take input of age of 3 people by user and determine oldest and youngest among them.
7) Write a program to print absolute value of a number entered by user.
E.g.- INPUT: 1 OUTPUT: 1, INPUT: -1 OUTPUT: 1
8) Modify the 4th question (in solved problem set) to allow student to sit if he/she has medical cause. Ask user if he/she has medical cause or not ('Y' or 'N') and print accordingly.
9) Ask user to enter age, sex (M or F), marital status (Y or N) and then using following rules print their place of service. if employee is female, then she will work only in urban areas.
if employee is a male and age is in between 20 to 40 then he may work in anywhere
if employee is male and age is in between 40 t0 60 then he will work in urban areas only.
And any other input of age should print "ERROR".

Descriptive Questions

1) What is data type? What are the different types of data?
2) What are the numeric data types? Explain with examples.
3) What are the sequence types of data?
4) Explain about type conversions.
5) What is mutable and immutable? What are mutable and immutable data types?
6) Explain about python output formatting.
7) Describe types of errors.
8) Explain about types of conditional constructs.
9) Give examples for all conditional constructs.
10) Write shorthand if and shorthand if-else statement.

Unit 4 - Loops/iterative statements

Module 1 – While Loop

Introduction

Looping is a powerful programming technique through which a group of statements is executed repeatedly, until certain specified condition is satisfied. Looping is also called repetitive or iterative control statements.

A loop in a program essentially consists of two parts, one is called the body of the loop and other is known as a control statement. The control statement performs a logical test whose result is either **True or False**. If the result of this logical test is true, then the statements contained in the body of the loop are executed. Otherwise, the loop is terminated.

There must be a proper logical test condition in the control statement, so that the statements are executed repeatedly and the loop terminates gracefully. If the logical test condition is carelessly designed, then there may be possibility of formation of an infinite loop which keeps executing the statements over and over again.

4.1 while loop

This is used to execute a set of statements repeatedly as long as the specified condition is true. It is an indefinite loop. In the while loop, the condition is tested before executing body of the statements. If the condition is True, only body of the block of statements will be executed otherwise if the condition is False, the control will jump to other statements which are outside of the while loop block.

Syntax:

```
while(logexp):
        block of Statements
```

Where,

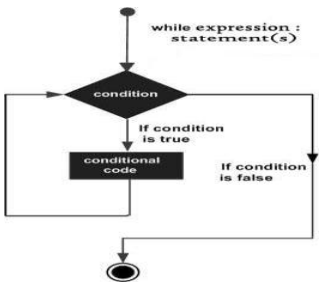
while → is a keyword

logexp →is a logical expression that results in either **True or Fasle**

Statement→ may be simple or a compound statement

Here, first of all the logical expressions is evaluated and if the result is true (non-zero) then the statement is repeatedly executed. If the result is false (zero), then control comes out of the loop and continues with the next executable statements.

Flowchart:



In while loop there are mainly it contains three parts, which are as follows

- Initialization: to set the initial value for the loop counter. The loop counter may be an increment counter or a decrement counter
- Decision: an appropriate test condition to determine whether the loop be executed or not
- Updation: incrementing or decrementing the counter value.

Example Programs:

Q1. Write a program to display your name up to 'n' times

```
n=int(input('Enter the number'))
#from the above line we take input from the user
i=1# assigning value to i variable
while(i<=n):# checking the condition
    print('RGUKT')#displaying output
    i=i+1#increasing i value here
print('good bye')
```

When you run the following program, then output will be display like

```
Enter the number
3
RGUKT
RGUKT
RGUKT
good bye
```

Q2. Write a program to display natural numbers up to n

```
n=int(input('Enter the number\n'))
i=1
print('Natural numbers are up to %d'%n)
while(i<=n):
    print(i)
    i=i+1
```

Output:

```
Enter the number
7
Natural numbers are up to 7
1
2
3
4
5
6
7
```

Q3.Write a program to display even numbers up to n

```
n=int(input('Enter the number: '))
i=1# assigning value 1 to i or intilization
print('Even numbers up to %d'%n)
while(i<=n):
    if(i%2==0):# here checking the condition whether the value is divisible or not
        print(i)# if it is divisible printing even numbers
    i=i+1# increment operator
```

Output:

```
Enter the number: 10
Even numbers up to 10
2
4
6
8
10
```

4.1.1 The infinite while loop: A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Program to demonstrate infinite loop:

```
while True:
    print('Hi')
print('Good bye')
```

```
x=1
while(x==1):
    print('Hi')
print('Good bye')
```

Above two programs that never stop, that executes until your Keyboard Interrupts (ctrl+c). Otherwise, it would have gone on unendingly. Many ‘Hi’ output lines will display when you executes above two programs and that never display ‘Good bye’

4.1.2 While loop with else clause/statement: Python allows an optional else clause at the end of a while loop. This is a unique feature of Python. If the else statement is used with a while loop, the else statement is executed when the condition becomes false. But, the while loop can be terminated with a break statement. In such cases, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is False

Syntax:

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

Example Program:

Q1. Write a program to demonstrate while loop with else block.

```
i=1
while(i<=3):
    print('RGUKT')
    i=i+1
else:
    print('Good bye')
```

Output:

```
RGUKT
RGUKT
RGUKT
Good bye
```

Q2. Write a program to demonstrate else block with break statement

```
i=1
while(i<=6):
    if(i==4):
        break
    print(i)
    i=i+1
else:
    print('Never executes thie else block')
```

Output:

```
1
2
3
```

Module 2 – For Loop

4.2 for loop

Like the while loop, for loop works, to repeat statements until certain condition is True. The for loop in python is used to iterate over a sequence (list, tuple, string and range() function) or other iterable types. Iterating over a s sequence is called traversal. Here, by sequence we mean just an ordered collection of items. for loop is usually known as definite loop because the programmer knows exactly how many times the loop will repeat.

Syntax:

for counter_variable in sequence:

    block of statements

Here counter_variable is the variable name that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for the loop is separated from the rest of the code using indentation.

Example programs:

Q7. Write a program to make sum of all numbers stored in a list.

```
#listof numbers
numbers=[5, 2, 7, 10, 14]
sum=0#assigning value to variable
for i in numbers:#iterate over the list
    sum=sum+i #making sum here
print('The sum is',sum)#displayiing output
```

When you run the program, Output will be:

```
The sum is 38
```

4.2.1 The range() function:

The range() function is a built-in function in python that is used to iterate over a sequence of numbers. The range() function contains three arguments/parameters like

range(start, end, step)

The range() generates a sequence of numbers starting with start(inclusive) and ending with one less than the number 'end'. The step argument is optional

By default, every number in the range is incremented by 1. Step can be either a positive or negative value but it cannot be equal to zero

Example Program:

Q8. Write a program to print first n natural numbers using a for loop

```
n=int(input('Enter the number: '))
for i in range(1,n,1):
    print(i,end=' ')
n=int(input('Enter the number: '))
for i in range(1,n):
    print(i,end=' ')
```

For both programs, output will be the same

```
Enter the number: 10
1 2 3 4 5 6 7 8 9
```

Q9. If you want to display including the given input number you can write a program like

```
n=int(input('Enter the number: '))
for i in range(1,n+1):
    print(i,end=' ')
```

Output:

```
Enter the number: 10
1 2 3 4 5 6 7 8 9 10
```

4.2.2 for loop with else:

A for loop can have an optional else block. The else part is executed when the loop has exhausted iterating the list. But, when the break statement use in for loop, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

Example:

Q. Program to demonstrate else block in for loop

```
for i in range(5):
    print(i)
else:
    print("No items left.")
```

Output:

```
0
1
2
3
4
No items left.
```

Here, the for loop prints numbers from the starting number to less than ending number. When the for loop exhausts, it executes the block of code in the else and prints No items left.

Q. Program to demonstrate else block in for loop with break statement:

```
for i in range(5):
    if(i==3):
        break
    print(i)
else:
    print('This else block not executes')
```

Output:

```
0
1
2
```

4.3 Nested loops

Python programming language allows using one loop inside another loop which is called a nested loop. Although this feature will work for both while loop as well as for loop. The syntax for a nested while loop statement in Python programming language is as follows

Syntax:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Note:

You can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

Example Program: Program to demonstrate nested for loop

```
for i in range(1,6):        # Outer loop
    for j in range(i):      # Inner loop
        print("*",end=' ')
    print()
```

Output:

If you run above two programs output will be the same

```
*
* *
* * *
* * * *
* * * * *
```

Nested while loop:

```
i=1
while(i<=5):                # Outer loop
    j=1
    while(j<=i):            # Inner loop
        print('*',end=' ')
        j=j+1
    print()
    i=i+1
```

Module 3 - Control Statements

4.4 Control statements

Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. These can be done by loop control statements. Loop control statements change execution from its normal sequence.

There are three different kinds of control statements which are as follows

1. break
2. continue
3. pass

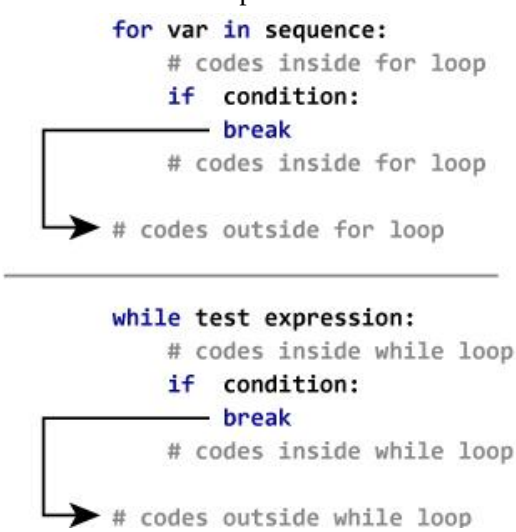
4.4.1Break Statement: The break statement in Python terminates the current loop and resumes execution at the next statement (out of the loop).

This break statement is used in both for and while loops.

Syntax:

break

Flowchart



Example Programs:

Write a program to demonstrate break statement by using for loop and while loop

for loop

```
for i in range(1, 11):
    if(i==5):
        break
    print(i)
print("Broke out of loop at i=",i)
```

when i equals 5 the loops break

while loop

```
i=1
while(i<11):
    if(i==5):
        break
    print(i)
    i=i+1
print("Broke out of loop at i=",i)
```

Output:

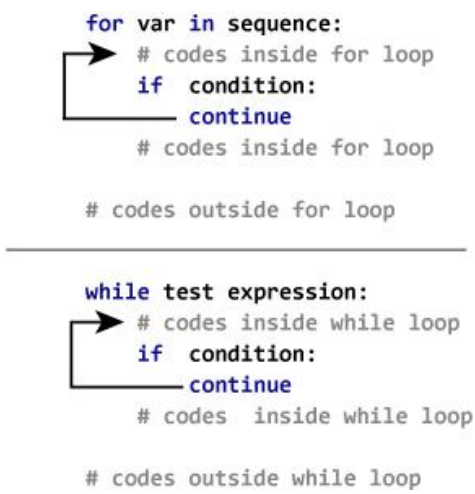
```
1
2
3
4
Broke out of loop at i= 5
```

4.4.2Continue Statement: The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration. Continue statement is opposite to break statement, instead of terminating the loop; it forces to execute the next iteration of the loop.

Syntax:

continue

Flowchart:



Example Programs:

Write a program to demonstrate continue statement by using for loop

```
for i in range(1, 10):
    if(i==5):
        continue
    print(i)
```

Output: 1 2 3 4 6 7 8 9

In output, 5 integer value is skipped based on if condition and continues flow of the loop until the condition satisfied.

4.4.3 Pass Statement: The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is a null statement. However, nothing happens when the pass statement is executed. It results in no operation (NOP). Pass statement can also be used for writing empty loops, functions and classes.

Syntax:

pass

Example Program:

Program to demonstrate pass statement using for loop

```
for i in range(1, 10):
    if(i==5):
        pass
    print(i, end=' ')
```

Output:

```
1 2 3 4 5 6 7 8 9
```

Solved Questions

1. **Write a program to display your name up to n times.**

Using while loop

```
n=int(input("Enter number of Times:"))
i=1
while(i<=n):
    print("RGUKT-RK Valley")
    i=i+1
```

Using for loop

```
n=int(input("Enter number of Times:"))
for i in range(n):
    print("RGUKT-RK Valley")
```

Output:

```
Enter number of Times:4
RGUKT-RK Valley
RGUKT-RK Valley
RGUKT-RK Valley
RGUKT-RK Valley
```

2. **Write a program to display sum of odd numbers up to n**

#Using while loop

```
n=int(input("Enter n Value:"))
i=1
s=0
while(i<=n):
   if(i%2!=0):
      s=s+1
   i=i+1
print("Sum of Odd Values", s)
```

#Using for loop

```
n=int(input("Enter n Value:"))
s=0
for i in range(1,n+1):
   if(i%2!=0):
      s=s+1
print("Sum of Odd Values", s)
```

3. **Write a program to display numbers of factors to the given number**

 # Using while loop

```
n=int(input("Enter n Value:"))
i=1
c=0
while(i<=n):
   if(n%i==0):
      c=c+1
   i=i+1
print("count of factors is ", c)
```

 # Using for loop

```
n=int(input("Enter n Value:"))
c=0
for i in range(1,n+1):
   if(n%i==0):
      c=c+1
print("count of factors is ", c)
```

4. **Write a program to find given number is prime number or not**

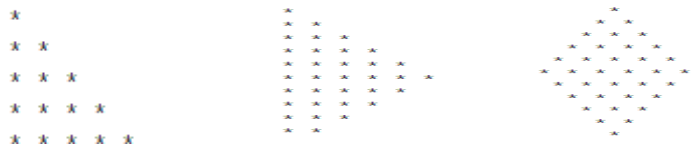
```
n=int(input("Enter n Value:"))
c=0
for i in range(1,n+1):
   if(n%i==0):
      c=c+1
if(c==2):
   print("given number is prime number")
else:
   print("given number is not prime number")
```

Descriptive Questions:

1) Explain about while loop and for loop?
2) Explain about loop control statements?
3) Explain about range function with different arguments?

Unsolved Questions:

1) Write a program to print all-natural numbers in reverse (from n to 1).
2) Write a program to print all even numbers up to n
3) Write a program to print sum of all odd numbers between two intravel given.
4) Write a program to print table of any number.
5) Write a program to enter any number and calculate sum of its digits.
6) Write a program to print all Prime numbers between 1 to n
7) Write a program to enter any number and display perfect numbers between 1 to n
8) Write a program to enter any number and find its first and last digit.
9) Write a program to display given number is a number palindrome or not
10) Write a program to print all alphabets from a to z.
11) Write a program to enter any number and check whether it is Armstrong number or not.
12) Write a program to print all Strong numbers between 1 to n.
13) Write a program to print Fibonacci series up to n terms.
14) Star pattern programs - Write a program to print the given star patterns.



15) Write a Python program to construct the following patterns, using a nested loop number

```
1                          P
2 3                        PY
4 5 6                      PYT
7 8 9 10                   PYTH
11 12 13 14 15             PYTHO
                           PYTHON
```

Multiple Choice Questions:

1) A while loop in Python is used for what type of iteration?
   a) Indefinite b) Discriminate c) Definite d) Indeterminate
2) When does the else statement written after loop executes?
   a) When break statement is executed in the loop
   b) When loop condition becomes false
   c) Else statement is always executed
   d) None of the above
3) What do we put at the last of for/while loop?
   a) Semicolon b) Colon c) Comma d) None of the above
4) Which of the following loop is work on the particular range in python?
   a) For loop b) While loop c) Do-while loop d) Recursion
5) How many times it will print the statement?, for i in range(100): print(i)
   a) 101 b) 99 c) 100 d) 0
6) What is the result of executing the following code?

```
n=5
while n<=5:
        if n<5:
                n=n+1
        print(n)
```

   a) The program will loop indefinitely
   b) The value of number will be printed exactly 1 time
   c) The while loop will never get executed
   d) The value of number will be printed exactly 5 times
7) Which of the following sequences would be generated by given line of the code?
   a) 5 4 3 2 1 0 -1 b) 5 4 3 2 1 0 c)5 3 1 d) Error
8) Which of the following is a valid for loop in Python?
   a) for (i=0;i<=n; i++) b) for i in range(5) c) for i in range (1, 5): d) for i in range(0,5,1)
9) Which statement is used to terminate the execution of the nearest enclosing loop in which it appears?
   a) pass b) break c) continue d) jump
10) Which statement indicates a NOP?
   a) Pass b) break c) continue d) jump

Unit 5 – Strings
Module 1

5.1 Definition

In python, consecutive sequence of characters is known as a string. Strings are immutable and amongst the most popular types in Python. In python, String literals are surrounded by single or double or triple quotation marks. 'RGUKT' is same as "RGUKT"

5.1.1 Declaring & Initializing StringVariables

Assigning a string to a variable is done with the variable name followed by an equal sign and the string

>>> s1 = "This is a stringone"
>>> s2 = "This is a string two"
>>>s3=" " #emptystring
>>>type(s1)<class'str'>

- Multi-line strings can be denoted using triple single or double quotes, ''' or""".
- Even triple quotes can be used in Python but generally used to represent **multiline strings** or**docstrings**.

>>> s4 = '''amultiline String'''

```
>>> v = ''' Python is a programming language.

Python can be used on a server to create web applications. '''
>>> print(v)
 Python is a programming language.

Python can be used on a server to create web applications.
```

5.1.2 How to access characters in String

- We can access individual characters using indexing and a range of characters using slicingoperator **[ ]**or **TheSubscriptOperator**
- Index starts from 0. Trying to access a character out of index range will raise an Index Error.
- The index must be an integer. We can't use float or other types;this will result into Type Error.
- Python allows negative indexing for itssequences.
- The index of -1 refers to the last item, -2 to the second last item and soon.
- We can access a range of items in a string by using the slicing operator withcolon**[:]**.

S="hello"

0	1	2	3	4
H	E	L	L	O
-5	-4	-3	-2	-1

Important points about accessing elements in the strings using index

- Positive index helps in accessing the string from the beginning
- Negative index helps in accessing the string from the end.
- Index 0 or –ve n(where n is length of the string) displays the first element.
  Example:A[0] or A[-5] will display "H"
- Index -1 or (n-1) displays the last element.
  Example: A[-1] or A[4] will display "O"

Note: Python does not support character data type. A string of size 1 can be treated as characters.
If we try to access index out of the range or use decimal number, we will get errors.

Accessing sub strings/Slicing Operator:

To access substrings, use the square brackets for slicing, using slice operator, along with the index or indices to obtain your substring.

Syntax:

stringname[start:end:step]
start – starting point of the index value (default value: 0)
end or stop – ending point of the index value (default value: len(stringname))
step- increment of the index values (default value: 1)

Examples:

```
>>> s = 'Andhra Pradesh'
>>> print(s[0:6])
Andhra
```

```
>>> s = 'Andhra Pradesh'
>>> print(s[0:6:2])
Adr
```

```
>>> s = 'Amaravathi'
>>> print(s[:4])
Amar
```

```
>>> s = 'Andhra Pradesh'
>>> print(s[-1:-4:-1])
hse
```

```
>>> s = 'Andhra Pradesh'
>>> print(s[-1::-1])
hsedarP arhdnA
```

5.1.3 How to change or delete a string?

Strings are immutable

Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example.We can simply reassign different strings to the samename.

```
my_string = 'Hello world'
my_string[5]='j'

Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    my_string[5]='j'
TypeError: 'str' object does not support item assignment
```

We cannot delete or remove characters from string,deleting the string is possible use the keyword **del**.

```
my_string = 'Hello world'
del my_string[5]

TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

5.2. Python String Operations

There are many operations that can be performed with string which makes it one of the most used data types in python

Operator	Description	Example
+ (Concatenation)	The + operator joins the text on both sides of the operator	>>> 'Save'+'Earth' 'Save Earth' To give a white space between the two words, insert a space before the closing single quote of the first literal.
* (Repetition)	The * operator repeats the string on the left hand side times the value on right hand side.	>>>3*'Save Earth ' 'Save Earth Save Earth Save Earth '
in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters.	>>>A='Save Earth' >>> 'S' in A True >>>'Save' in A True >>'SE' in A False
not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of **in** operator discussed above)	>>>'SE' not in 'Save Earth' True >>>'Save ' not in 'Save Earth' False

Comparing Strings: Python allows you to compare strings using relational (or comparison) operator such as

- = =,! =, >, <, < =, >=,etc.
- == if two strings are equal, it returns True
- != or <> if two strings are not equal, it returns True
  - ➢ if first string is greater than the second, it returns True
- < if second string is greater than the first ,it returns True
- >= if first string is greater than or equal to the second, it returns True
- <= if second string is greater than or equal to the first, it returns True

Note:

- *These operators compare the strings by using the lexicographical order i.e using ASCII values of the characters.*
- *The ASCII values of A-Z is 65 -90 and ASCII code for a-z is 97-122 .*

Logical Operators on String in Python: Python considers empty strings as having boolean value of 'false' and non-empty string as having boolean value of 'true'.

Let us consider the two strings namely str1 and str2 and try boolean operators on them:

```
str1 = 'IIIT'
str2 = ''

print(str1) #returns IIIT
print(repr(str1)) #returns 'IIIT'

#logical operators on strings

print(str1 or str2) #returns IIIT
print(str1 and str2) #returns empty string
print(repr(str1 and str2)) #returns empty string in quotes

print(not str1) #returns False
print(not str2) #returns True
```

Output:

```
IIIT
'IIIT'
IIIT

''
False
True
```

5.3 Escape sequence characters

- To insert characters that are illegal in a string, use an escape character.
- Escape sequence characters or non-printable characters that can be represented with backslash notation.
- An escape character gets interpreted; in a single quoted as well as double quoted strings

An example of an illegal character is a double quote inside a string that is surrounded by double quotes

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

#You will get an error if you use double quotes inside a string that are surrounded by double quotes:

```
    txt = "We are the so-called "Vikings" from the north."
                               ^
SyntaxError: invalid syntax
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

txt = "We are the so-called \"Vikings\" from the north."

print(txt)

```
We are the so-called "Vikings" from the north.
```

Other escape characters used in Python

Escape Character	Description	Example	Result
\'	Single Quote	txt = 'It\'s alright.' print(txt)	It's alright.
\\	Backslash	txt = "This will insert one \\ (backslash)." print(txt)	This will insert one \ (backslash).
\n	New Line	txt = "Hello\nWorld!" print(txt)	Hello World!
\r	Carriage Return	txt = "Hello\rWorld!" print(txt)	Hello World!
\t	Tab	txt = "Hello\tWorld!" print(txt)	Hello World!

\b	Backspace	#This example erases one character (backspace):
		txt = "Hello \bWorld!"
		print(txt)
			HelloWorld!

\ooo	Octal value	#A backslash followed by three integers will result in a octal value:
		txt = "\110\145\154\154\157"
		print(txt)
			Hello

\xhh	Hex value	#A backslash followed by an 'x' and a hex number represents a hex value:
		txt = "\x48\x65\x6c\x6c\x6f"
		print(txt)
			Hello

Module – 2

5.4 Iterating through String/Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

Example:

```
"""
Python Program:
 Using for loop to iterate over a string in Python
"""
string_to_iterate = "RGUKT"
for char in string_to_iterate:
    print(char)
```

Example:

```
"""
Python Program:
 Using for loop to iterate using index of a string in Python
"""
string_to_iterate = "RGUKT"
for i in range(len(string_to_iterate)):
    print(string_to_iterate[i])
```

Example:

```
"""
Python Program:
 Using while loop to iterate using index of a string in Python
"""
string_to_iterate = "RGUKT"
i=0
while i< len(string_to_iterate):
    print(string_to_iterate[i])
    i=i+1
```

Example:

Python program to print reverse string using for loop

```
s1='Hello' # Assign String here
s2='' #empty string
l=0 # length
for i in s1:
    l=l+1
#finaly legth of the string l=5
for i in range(1,l+1):
    s2+=s1[l-i]
print("The Reverse String of ",s1,"is",s2)
```

5.5 The format() Method for Formatting Strings

- The format() method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
- We can use positional arguments or keyword arguments to specify the order.

```
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

5.6 Built-in functions to Work with Python Strings

- Various built-in functions that work with sequence, works with string as well.
- The **enumerate()** function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.
- The **len()** function returns the length (number of characters) of the string.

```
s='RGUKT'
#enumerate()
list_enumerate=list(enumerate(s))
print('list(enumerate(s))=',list_enumerate)
#list(enumerate(s))= [(0,'R'),(1,'G'),(2,'U'),(3,'K'),(4,'T')]

#character count
print('length of the string =',len(s))
#length of the string = 5
```

- The method **center()** makes str centered by taking width parameter into account. Padding is specified by parameter ***fillchar***. Default filler is a space.

Syntax: str.center(width[, fillchar])

```
str = "RGUKT IIIT"
str1= str.center(30,'a')
str2= str.center(30)
print(str1)
print (str2 )
```

- The function**max()** returns the max character from string str according to ASCII value.in first print statement y is max character, because ASCII code of "y" is 121. In second print statement "s" is max character, ASCII code of "s" is 115.

Syntax:- max(str)

- The function**min()** returns the min character from string str according to ASCII value.

Syntax :- min(str)

```
s="RGUKT-RKV RGUKT-RKV"
print(s.split())
print(s.split('-',1))

s="RGUKT-RKVRGUKT-RKV"
print(max(s))

print(min(s))
```

- The **ord()** function returns ASCII code of the character.

```
ch='A'
print(ord(ch)) # 65
```

- The **chr()** function returns character represented by a ASCII number.

```
num=90
print(chr(num)) # Z
```

Module – 3

5.7 String Methods

- **count():**The method **count()** returns the number of occurrence of Python string *substr* in string *str*. By using parameter *start* and *end* you can give slice of *str*.This function takes 3 arguments, substring, beginning position (by default 0) and end position (by default string length).Return Int Value

Syntax: str.count(substr [, start [, end]])

```
str = "This is a count example"
sub = "i"
print(str.count(sub, 4, len(str)))
sub = "a"
print (str.count(sub) )
```

- **endswith("string", beg, end):** This function returns true if the string ends with mentioned string(suffix) else return false.Return Bool Value

The use of *start* and *end* to generate slice of Python string str.

Syntax: str.endswith(suffix[, start[, end]])

```
str = "RGUKT IIIT RKVALLEY"
sub="IIIT"
l=len(str)
print(l)
print(str.endswith(sub,2,6))
print(str.endswith(sub,10))
sub='sdfs'
print(str.endswith(sub))
```

- **startswith("string", beg, end):**This function returns true if the string begins with mentioned string(prefix) else return false.

```
s = 'Amaravathi, Tirupathi'
i = s.startswith("Amar") #checks from the beginning of the string and returns result
print(i) #displays True

i = s.startswith("Amar", 10, 20) #checks from 10th index and returns result
print(i) #displays False
```

- **find("string", beg, end):**This function is used to find the position of the substring within a string. It takes 3 arguments, substring , starting index(by default 0) and ending index(by default string length).
  - It returns "-1" if string is not found in given range.
  - It returns first occurrence of string if found.

Return the lowest index in S where substring sub is found

If given Python string is found, then the **find()** method returns its index. If Python string is not found then -1 would be returned.

Syntax : str.find(str, beg=0 end=len(string))

```
str = "RGUKT IIIT RKVALLEY"
sub1="IIIT"
sub2='RKV'
print(str.find(sub1))
print(str.find(sub1,20))
print(str.find(sub2))
```

- **rfind("string", beg, end):** This function is similar to find(), but it returns the position of the last occurrence of sub string.

```
s = "Amaravathi, Tirupathi"
i = s.rfind("thi") #checks entire string and returns the last occurrence of substring i.e., 18
print(i)
```

- **replace():** This function is used to replace the substring with a new substring in the string. This function has 3 arguments. The string to replace, new string which would replace and max value denoting the limit to replace action (by default unlimited).

```
#string.replace(oldstring, newstring)
st="RGUKT RKV"
n=st.replace('RKV','ONG')
print(n)
```

- The method **isalnum()** is used to determine whether the Python string consists of alphanumeric characters, false otherwise

  Syntax :-str.isalnum()
- The method **isalpha()** return true if the Python string contains only alphabetic character(s), false otherwise.

Syntax :- str.isalpha()

- The method **isdigit()** return true if the Python string contains only digit(s),false otherwise.

Syntax :-str.isdigit()

- The method **islower()** return true if the Python string contains only lower cased character(s), false otherwise

Syntax :- str.isdigit()

```
s='rgukt1234'
print(s.isalnum())
s1='#$@#%'
print(s1.isalnum())
s2='PinCode'
print(s2.isalpha())
s3='rgukt1234'
print(s3.isalnum())
s4='1234'
print(s3.isdigit())
s5='rgukt1234'
print(s5.isdigit())
```

```
s='rguktrkv'
print(s.islower())
s1='pin343223'
print(s1.islower())
s2='PinCode'
print(s2.islower())
```

- The method **isspace()** return true if the Python string contains only white space(s).

Syntax :- str.isspace()

- The method **istitle()** return true if the string is a titlecased

Syntax :- str.istitle()

- The method **isupper()** return true if the string ontains only upper cased character(s), false otherwise.

Sytax:-str.isupper()

- The method **ljust(),** it returns the string left justified. Total length of string is defined in first parameter of method *width*. Padding is done as defined in second parameter *fillchar* .(default is space)

Syntax : -str.ljust(width[, fillchar])

```
s=" "
print(s.isspace())

s1="pin 2342"
print(s1.isspace())

s2="rgukt rkv"
print(s2.istitle())

s3="Rgukt rkv"
print(s3.istitle())

s4="Rgukt Rkv"
print(s4.istitle())
```

```
s5="RGUKT"
print(s5.isupper())

s6="RGUKT123"
print(s6.isupper())

s7="rGUKT"
print(s7.isupper())

s="rgukt"
print(s.ljust(10,"k"))
print(s.ljust(10))
print(s.ljust(3,"k"))
```

- In above example you can see that if you don't define the *fillchar* then the method **ljust()** automatically take space as *fillchar*.
- The method **rjust(),** it returns the string right justified. Total length of string is defined in first parameter of method*width*. Padding is done as defined in second parameter *fillchar* .(default is space)

Syntax:-str.rjust(width[, fillchar])

- This function **capitalize()** first letter of string.

Syntax:-str.capitalize()

- The method **lower()** returns a copy of the string in which all case-based characters have been converted to lower case.

Syntax:-str.lower()

- The method **upper()** returns a copy of the string in which all case-based characters have been converted to upper case.

Syntax:-str.upper()

- The method **title()** returns a copy of the string in which first character of all words of string are capitalised.

Syntax:-str.title()

- The method **swapcase()** returns a copy of the string in which all cased based character swap their case

Syntax:-str.swapcase()

```
s = "APJ Kalam is GREAT Scientist"

l = s.lower() #returns a new string with all lower case letters in it
print(l)

u = s.upper() #returns a new string with all upper case letters in it
print(u)

sc = s.swapcase() #returns a new string 'apj kALAM IS great sCIENTIST'
print(sc)

t = s.title() #returns a new string 'Apj Kalam Is Great Scientist'
print(t)

c = s.capitalize() #returns a new string 'Apj kalam is great scientist'
print(c)
```

- This method **join()** returns a string which is the concatenation of given sequence and stringas shown in example.

seq = it contains the sequence of separated strings.
str = it is the string which is used to replace the separator of sequence

Syntax:-str.join(seq)

- The method **lstrip()** returns a copy of the string in which specified char(s) have been stripped from left side of string. If char is not specified then space is taken as default.

Syntax : - str.lstrip([chars])

- The method **rstrip()** returns a copy of the string in which specified char(s) have been stripped from right side of string. If char is not specified then space is taken as default.

Syntax : - str.rstrip([chars])

- The method **strip()** returns a copy of the string in which specified char(s) have been stripped from both side of string. If char is not specified then space is taken as default.

Syntax : - str.strip([chars])

- The method **split()** returns a list of all words in the string, delimiter separates the words. If delimiter is not specified then whitespace is taken as delimiter, parameter num

Syntax :- str.split("delimiter", num)

Multiple Choice Questions

```
str1="6/4"

print("str1")
```

1) What will be the output of above Python code?
   a) 1 b) 6/4 **c) 1.5** d) str1
2) Which of the following is False?
   a) String is immutable.
   b) capitalize() function in string is used to return a string by converting the whole given string into uppercase.
   c) lower() function in string is used to return a string by converting the whole given string into lowercase.
   d) None of these

```
str1="Information"

print(str1[2:8])
```

3) What will be the output of below Python code?
   a) **format** B. formation C. orma D. ormat
4) What will be the output of below Python code?
   a) Application b) Application c) **ApplicAtion d)** Application

```
str1="Aplication"

str2=str1.replace('a','A')

print(str2)
```

5) What will be the output of below Python code?
   a) **POWER b)** Power c) power d) power

```
str1="poWer"

str1.upper()

print(str1)
```

6) What will the below Python code will return?

```
list1=[0,2,5,1]

str1="7"

for i in list1:

        str1=str1+i

print(str1)
```

a) 70251 b) 7 c) 15 d) **Error**

7) Which of the following will give "Simon" as output?

```
If str1="John,Simon,Aryan"
```

a) print(str1[-7:-12]) b) print(str1[-11:-7]) c) **print(str1[-11:-6])** d) print(str1[-7:-11])

8) What will return following Python code?

```
str1="Stack of books"
print(len(str1))
```

a) 13 B. 14 **C.15** D. 16

Solved Problems

1. **Write a program to print Alphabet using ASCII Numbers ?**

Solutions:-

```
ch='A'
print(ord(ch)) # A=65,a=97

ch='Z'
print(ord(ch)) # Z=90,z=122

for i in range(65,91):
    print(chr(i),end=" ")
#A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
print()
for i in range(97,123):
    print(chr(i),end=" ")
#a b c d e f g h i j k l m n o p q r s t u v w x y z
```

2. **Write a Python program to calculate the length of a string.**

Solutions:-

```
def string_length(str1):
    count = 0
    for char in str1:
        count += 1
    return count
print(string_length('w3resource.com'))
```

3. **Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.**

Solutions:-

```
def string_both_ends(str):
  if len(str) < 2:
    return ''

  return str[0:2] + str[-2:]

print(string_both_ends('w3resource'))
print(string_both_ends('w3'))
print(string_both_ends('w'))
```

Unsolved Problems

1. Write a program to find number of digits ,alphabets and symbols ?
2. Write a program to convert lower case to upper case from given string?
3. Write a program to print the following output? image like

```
R
R G
R G U
R G U K
R G U K T
R G U K
R G U
R G
R
```

4. Write a program to check whether given string is palindrome or not?
5. Write a program to find no_ words, no_letters, no_digits and no_blanks in a line?
6. Write a program to sort list names in alphabetical order?
7. To find the first character from given string,count the number of times repeated and replaced with * except first character then print final string?
8. To find the strings in a list which are matched with first character equals to last character in a string?
9. Write a program that accepts a string from user and redisplays the same string after removing vowels from it?
10. This is a Python Program to take in two strings and display the larger string without using built-in functions?
11. Python Program to Read a List of Words and Return the Length of the Longest One?
12. Python Program to Calculate the Number of Upper-Case Letters and Lower-Case Letters in a String?

Unit 6 - List, Tuples and Dictionary

Module 1 - List

6.1 Definition

Python offers a range of compound data types often referred to as sequences. List is one of the most frequently used and very versatile data types used in Python. Like a string, a list is a sequence of values. In a string, the values are characters; in a list, they can be any type. The values in list are called elements or sometimes items.

There are several ways to create a new list; the simplest is to enclose the elements in square brackets and are separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.).

[10, 20, 30, 40]

['crunchy frog', 'ram bladder', 'lark vomit']

The first example is a list of four integers. The second is a list of three strings. The elements of a list don't have to be the same type. The following list contains a string, a float, an integer, and another list:

['spam', 2.0, 5, [10, 20]]

A list can also have another list as an item. This is called a nested list. It can have any number of items and they may be of different types (integer, float, string etc.).

6.2 How to create a list?

In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas.

 # empty list
my_list = []
 # list of integers
my_list = [1, 2, 3]
 # list with mixed data types
my_list = [1, "Hello", 3.4]

6.3 How to access elements from a list?

There are various ways in which we can access the elements of a list.

6.3.1 List Index: We can use the index operator [] to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4. Trying to access indexes other than these will raise an IndexError. The index must be an integer. We can't use float or other types, this will result in TypeError.

- Nested lists are accessed using nested indexing.

```
# List indexing

my_list = ['p', 'r', 'o', 'b', 'e']

# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0][1])

print(n_list[1][3])

# Error! Only integer can be used for indexing
print(my_list[4.0])
```

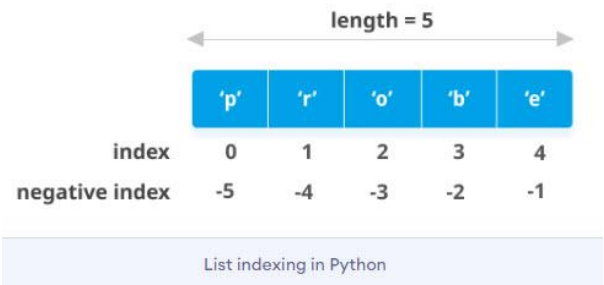
Output

```
p
o
e
a
5
Traceback (most recent call last):
  File "<string>", line 21, in <module>
TypeError: list indices must be integers or slices, not float
```

6.3.2 Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
# Negative indexing in lists
my_list = ['p','r','o','b','e']

print(my_list[-1])

print(my_list[-5])
```



List indexing in Python

When we run the above program, we will get the following output:

```
e
p
```

6.4 Change or add elements to a list

The syntax for accessing the elements of a list is the same as for accessing the characters of a string the bracket operator. The expression inside the brackets specifies the index. Remember that the indices start at 0:

Unlike strings, lists are mutable because you can change the order of items in a list or reassign an item in a list. When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

>>> numbers = [17, 123]
>>>numbers[1] = 5
>>> print numbers
[17, 5]

The one-eth element of numbers, which used to be 123, is now 5.

You can think of a list as a relationship between indices and elements. This relationship is called a mapping; each index "maps to" one of the elements.

List indices work the same way as string indices:

6.5 Traversing a list

The most common way to traverse the elements of a list is with a *for* loop. The syntax is the same as for strings:

```
cheeses = ['Cheddar', 'Edam', 'Gouda']
for cheese in cheeses:
        print (cheese)
```

This works well if you only need to read the elements of the list. But if you want to write or update the elements, you need the indices. A common way to do that is to combine the functions *range* and *len*:

```
numbers = [17, 123]
for i in range(len(numbers)):
numbers[i] = numbers[i] * 2
```

This loop traverses the list and updates each element.*len* returns the number of elements in the list. *range* returns a list of indices from 0 to *n*−1, where *n* is the length of the list. Each time through the loop, *i* gets the index of the next element. The assignment statement in the body uses *i* to read the old value of the element and to assign the new value.

A *for* loop over an empty list never executes the body:

```
for x in empty:
print ('This never happens.')
```

Although a list can contain another list, the nested list still counts as a single element. The length of this list is four:

['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]

6.6 List operations

The + operator concatenates lists:

>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print (c)
[1, 2, 3, 4, 5, 6]

Similarly, the * operator repeats a list a given number of times:

>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

The first example repeats [0] four times. The second example repeats the list [1, 2, 3] three times.

List slices: The slice operator also works on lists

>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>t[1:3]
['b', 'c']
>>>t[:4]
['a', 'b', 'c', 'd']
>>>t[3:]
['d', 'e', 'f']
If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. So if you omit both, the slice is a copy of the whole list.
>>>t[:]
['a', 'b', 'c', 'd', 'e', 'f']
Since lists are mutable, it is often useful to make a copy before performing operations that fold, spindle, or mutilate lists.
A slice operator on the left side of an assignment can update multiple elements:
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>t[1:3] = ['x', 'y']
>>> print (t)
['a', 'x', 'y', 'd', 'e', 'f']
Syntax: List[start:end:step]
start –starting point of the index value end
stop –ending point of the index value
step- increment of the index values
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>t[1:3:1]
['b', 'c']
6.7 List methods
Adding Elements into a list
6.7.1 list.append()
Python provides methods that operate on lists. For example, *append* adds a new element to the end of a list
>>> t = ['a', 'b', 'c']
>>>t.append('d')
>>> print (t)
['a', 'b', 'c', 'd']
6.7.2 list.extend()
extend takes a list as an argument and appends all of the elements
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print (t1)
['a', 'b', 'c', 'd', 'e']
This example leaves t2 unmodified.
sort arranges the elements of the list from low to high:
>>> t = ['d', 'c', 'e', 'b', 'a']
>>>t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
6.7.3 list.insert(*i, x*)
Insert an item at a given position. The first argument is the index of the element before which to insert,
so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
Example 1: Inserting an Element to the List
 # vowel list
vowel = ['a', 'e', 'i', 'u']
 # 'o' is inserted at index 3
 # the position of 'o' will be 4th
vowel.insert(3, 'o')
print('Updated List:', vowel)
Deleting elements
6.7.4 list.remove(*x*)
Remove the first item from the list whose value is equal to *x*. It raises a ValueError if there is no such item.

6.7.5 list.clear()
Remove all items from the list. Equivalent to del a[:].
Example 1: Working of clear() method

```
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]
# clearing the list
list.clear()
print('List:', list)
```

6.7.6 list.pop([*i*])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *I* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

```
>>> t = [‘a’, ‘b’, ‘c’]
>>> x = t.pop(1)
>>> print (t)
[‘a’, ‘c’]
>>> print (x)
b
```

If you don’t need the removed value, you can use the del operator:

```
>>> t = [‘a’, ‘b’, ‘c’]
>>> del t[1]
>>> print (t)
[‘a’, ‘c’]
```

If you know the element you want to remove (but not the index), you can use remove:

```
>>> t = [‘a’, ‘b’, ‘c’]
>>>t.remove(‘b’)
>>> print (t)
[‘a’, ‘c’]
```

The return value from remove is None.
To remove more than one element, you can use del with a slice index:

```
>>> t = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
>>> del t[1:5]
>>> print (t)
[‘a’, ‘f’]
```

As usual, the slice selects all the elements up to, but not including, the second index.
6.7.6 list.index(*x*[, *start*[, *end*]])
Return zero-based index in the list of the first item whose value is equal to *x*. Raises a ValueError if there is no such item.
The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.
Example:

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# index of 'e' in vowels
index = vowels.index('e')
print('The index of e:', index)
# element 'i' is searched
# index of the first 'i' is returned
index = vowels.index('i')
print('The index of i:', index)
```

Example: Working of index() With Start and End Parameters

```
# alphabets list
alphabets = ['a', 'e', 'i', 'o', 'g', 'l', 'i', 'u']
# index of 'i' in alphabets
index = alphabets.index('e')   # 2
print('The index of e:', index)
# 'i' after the 4th index is searched
index = alphabets.index('i', 4)   # 6
```

```
print('The index of i:', index)
# 'i' between 3rd and 5th index is searched
index = alphabets.index('i', 3, 5)   # Error!
print('The index of i:', index)
```

6.7.7 list.count(*x*)

Return the number of times *x* appears in the list.

Example:

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('p')
# print count
print('The count of p is:', count)
```

6.7.8 list.sort(*key=None, reverse=False*)

Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their explanation). Most list methods are void; they modify the list and return None. If you accidentally write list= list.sort(), you will be disappointed with the result.

Example : Sort a given list

```
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort()
# print vowels
print('Sorted list:', vowels)
```

Example : Sort the list in Descending order

```
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort(reverse=True)
# print vowels
print('Sorted list (in Descending):', vowels)
```

6.7.9 list.reverse()

Reverse the elements of the list in place.

```
# Operating System List
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)
# List Reverse
systems.reverse()
# updated list
print('Updated List:', systems)
```

6.7.10 list.copy()

Return a shallow copy of the list. Equivalent to a[:]

```
# mixed list
my_list = ['cat', 0, 6.7]
# copying a list
new_list = my_list.copy()
print('Copied List:', new_list)
```

6.8 List Comprehension

List comprehension is an elegant and concise way to create a new list from an existing list in Python. A list comprehension consists of an expression followed by for statement inside square brackets. Here is an example to make a list with each item being increasing power of 2.

```
pow2 = [2 ** x for x in range(10)]
print(pow2)
```

```
Output

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

List Membership Test:
We can test if an item exists in a list or not, using the keyword *in*.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

Output

```
True
False
True
```

6.9 Built-in functions
There are a number of built-in functions that can be used on lists that allow you toquickly look through a list without writing your own loops:
>>>nums = [3, 41, 12, 9, 74, 15]
>>> print len(nums)
6
>>> print max(nums)
74
>>> print min(nums)
3
>>> print sum(nums)
154
>>> print sum(nums)/len(nums)
25
The sum() function only works when the list elements are numbers. The other functions (max(), len(), etc.) work with lists of strings and other types that can be comparable.
Example:

```
#A PROGRAMME TO CREAT A LIST WITH N ELEMENTS and find its average, min & max value
n=int(input("Enter length of list: "))
l=[]
c=0
sum=0
for i in range(n):
    a=float(input("Enter a value: "))
    l.append(a)
    if(l[i]!=0):
        sum=sum+l[i]
        c=c+1
avg=float(sum/c)
print ("average of given elements is:", avg)
max=l[0]
for i in range(1,n):
    if(max<l[i]):
        max=l[i]
print ("Maximum value of given list is: ",max)
min=l[0]
for i in range(1,n):
    if(min>l[i]):
        min=l[i]
print ("Minimum value of given list is: ",min)
```

Output:

```
Enter length of list: 6

Enter a value: 2

Enter a value: 5

Enter a value: 8

Enter a value: 3

Enter a value: 9

Enter a value: 7
average of given elements is: 5.666666666666667
Maximum value of given list is:  9.0
Minimum value of given list is:  2.0
```

Example:2

```
#A program to remove duplicate elements from a given list
l=input("Enter list of elements without  comma between: ")
k=len(l)
l1=[]
for i in l:
    if i not in l1:
        l1.append(i)
print ("updated list: ",l1)
```

Output: 1

```
Enter list of elements without  comma between: 1234516276543
updated list:  ['1', '2', '3', '4', '5', '6', '7']
```

Output: 2

```
Enter list of elements without  comma between: RGUKT RK Valley
updated list:  ['R', 'G', 'U', 'K', 'T', ' ', 'V', 'a', 'l', 'e', 'y']
```

Example:3

```
#a program to find sum of elements of a given list
li=[2,3,4,5,8,1,7]
le=len(li)
i=0
sum=0
while(i<le):
    sum=sum+li[i]
    i=i+1
print("The sum of the elements:%d"%sum)
```

Output:

```
The sum of the elements:30
```

Module 2 – Tuples

We saw that lists and strings have many common properties, such as indexing and slicing operations. They are two examples of *sequence* data types (Sequence Types — list, tuple, range). Since Python is an evolving language, other sequence data types may be added. There is also another standard sequence data type: the *tuple*.

A tuple consists of a number of values separated by commas, for instance:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

As you see, on output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses, although often parentheses are necessary anyway (if the tuple is part of a larger expression). It is not possible to assign to the individual items of a tuple, however it is possible to create tuples which contain mutable objects, such as lists.

Though tuples may seem similar to lists, they are often used in different situations and for different purposes. Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing (or even by attribute in the case of namedtuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

A special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these. Empty tuples are constructed by an empty pair of parentheses; a tuple with one item is constructed by following a value with a comma. For example:

```
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

A tuple can also be created without using parentheses. This is known as tuple packing.

```
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog
```

```
Output

(3, 4.6, 'dog')
3
4.6
dog
```

6.10 Access Tuple Elements

There are various ways in which we can access the elements of a tuple.

Indexing:

We can use the index operator [] to access an item in a tuple, where the index starts from 0. So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range (6,7,... in this example) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

```
# Accessing tuple elements using indexing
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'
print(my_tuple[5])   # 't'

# IndexError: list index out of range
# print(my_tuple[6])

# Index must be an integer
# TypeError: list indices must be integers, not float
# my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])       # 's'
print(n_tuple[1][1])       # 4
```

```
Output

p
t
s
4
```

Negative Indexing:

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on.

```
# Negative indexing for accessing tuple elements
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')

# Output: 't'
print(my_tuple[-1])

# Output: 'p'
print(my_tuple[-6])
```

```
Output

t
p
```

Slicing:

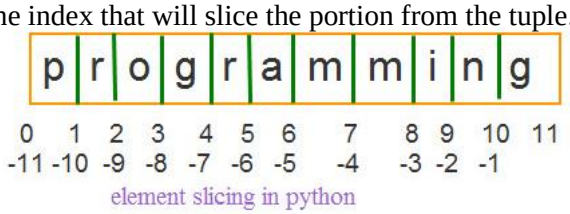
We can access a range of items in a tuple by using the slicing operator ***colon:***.

```
# Accessing tuple elements using slicing
my_tuple = ('p','r','o','g','r','a','m','m','i','n','g')

# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r','o', 'g')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('m','i', 'n','g')
print(my_tuple[7:])

# elements beginning to end
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'n','g')
print(my_tuple[:])
```

Output

```
('r', 'o', 'g')
('p', 'r', 'o', 'g')
('m', 'i', 'n', 'g')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g')
```

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.



element slicing in python

6.11 Changing a Tuple

Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once they have been assigned. But, if the element is itself a mutable data type like list, its nested items can be changed.

We can also assign a tuple to different values (reassignment).

```
>>> t=(5,46,34)
>>> t[0]=43
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    t[0]=43
TypeError: 'tuple' object does not support item assignment
```

Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. It means that we cannot delete or remove items from a tuple. Deleting a tuple entirely, however, is possible using the keyword *del*.

```
# Accessing tuple elements using slicing
my_tuple = ('p','r','o','g','r','a','m','m','i','n','g')

#deleting entire tuple using del keyword
del my_tuple

#trying to print the deleted tuple,
#it will cause an error because tuple has already deleted
print(my_tuple)
```

When you run the above code output will be:

```
Traceback (most recent call last):
  File "C:/Users/VJ/Desktop/jaffa.py", line 9, in <module>
    print(my_tuple)
NameError: name 'my_tuple' is not defined
```

6.12 Tuple operations

We can use + operator to combine two tuples. This is called **concatenation**.

We can also **repeat** the elements in a tuple for a given number of times using the * operator.

Both + and * operations result in a new tuple.

```
# Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))

# Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

Output

```
(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword *in*.

```
# Membership test in tuple
my_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation
print('a' in my_tuple)
print('b' in my_tuple)

# Not in operation
print('g' not in my_tuple)
```

Output

```
True
False
True
```

6.13 Iterating Through a Tuple

We can use a *for* loop to iterate through each item in a tuple.

```
# Using a for loop to iterate through a tuple
for name in ('John', 'Kate'):
    print("Hello", name)
```

Output

```
Hello John
Hello Kate
```

6.14 Built-in functions

len()- to determine how many elements in the tuple

tuple()-a function to make a tuple

count()-Returns the number of times a specified value occurs in a tuple

index()-Searches the tuple for a specified value and returns the position where it was found

min()-Returns the minimum value in a tuple

max()-Returns the maximum value in a tuple

6.15 Advantages of Tuple over List

- Since tuples are quite similar to lists, both of them are used in similar situations. However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:
- We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
- Since tuples are immutable, iterating through a tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Example: 1

```
#to find repeated elements in a tuple
t1=(1,3,5,2,1,6,5,4,8)
t2=()
t3=()
for i in t1:
    if i not in t2:
        t2=t2+(i,)
    else:
        t3=t3+(i,)
print("The repeated values are:",t3)
```

Output:

```
The repeated values are: (1, 5)
```

Example: 2

```
#Python Program to Create a List of Tuples with the
#First Element as the Number and Second Element as the Square of the Number
l_range=int(input("Enter the lower range:"))
u_range=int(input("Enter the upper range:"))
a=[(x,x**2) for x in range(l_range,u_range+1)]
print(a)
```

Output:

```
Enter the lower range:1

Enter the upper range:10
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100)]
```

Module3 - Dictionaries

A **dictionary** is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.

You can think of a dictionary as a mapping between a set of indices (which are called **keys**) and a set of values. Each key map to a value. The association of a key and a value is called a **key-value pair** or sometimes an **item**.

As an example, we'll build a dictionary that map from English to Spanish words, so the keys and the values are all strings.

The function **dict** creates a new dictionary with no items. Because **dict** is the name of a built-in function, you should avoid using it as a variable name.

>>> eng2sp = dict()
>>> print (eng2sp)
{}

The curly brackets, {}, represent an empty dictionary. To add items to the dictionary,you can use square brackets:

>>> eng2sp['one'] = 'uno'

This line creates an item that maps from the key 'one' to the value 'uno'. If weprint the dictionary again, we see a key-value pair with a colon between the keyand value:

>>> print (eng2sp)
{'one': 'uno'}

This output format is also an input format. For example, you can create a newdictionary with three items:

>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}

But if you print eng2sp, you might be surprised:

>>> print (eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}

The order of the key-value pairs is not the same. In fact, if you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.

But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

>>> print (eng2sp['two'])
'dos'

The key **'two'** always maps to the value 'dos' so the order of the items doesn't matter.

If the key isn't in the dictionary, you get an exception:

>>> print (eng2sp['four'])
KeyError: 'four'

6.16 Accessing Elements from Dictionary

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets **[]** or with the **get()** method.

If we use the square brackets **[], KeyError** is raised in case a key is not found in the dictionary. On the other hand, the **get()** method returns **None** if the key is not found.

```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# KeyError
print(my_dict['address'])
```

Output:

```
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

6.17 Changing and Adding Dictionary elements

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator. If the key is already present, then the existing value gets updated. In case the key is not present, a new **(key: value)** pair is added to the dictionary.

```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Output:

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

6.18 Looping Techniques

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the **items()** method.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the **enumerate()** function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

To loop over two or more sequences at the same time, the entries can be paired with the **zip()** function.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

6.19 Removing elements from Dictionary

We can remove a particular item in a dictionary by using the **pop()** method. This method removes an item with the provided **key** and returns the **value**.

The **popitem()** method can be used to remove and return an arbitrary **(key, value)** item pair from the dictionary. All the items can be removed at once, using the **clear()** method.

We can also use the **del** keyword to remove individual items or the entire dictionary itself.

```
# Removing elements from a dictionary

# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(squares)

# remove all items
squares.clear()

# Output: {}
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)
```

Output

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

6.20 Dictionary Comprehension

Dictionary comprehension is an elegant and concise way to create a new dictionary from an iterable in Python. Dictionary comprehension consists of an expression pair **(key: value)** followed by a **for** statement inside curly braces **{}**.

Here is an example to make a dictionary with each item being a pair of a number and its square.

```
# Dictionary Comprehension
squares = {x: x*x for x in range(6)}

print(squares)
```

Output

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

6.21 Dictionary Membership Test

We can test if a **key** is in a dictionary or not using the keyword **in**. Notice that the membership test is only for the **keys** and not for the **values**.

```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in squares)

# Output: True
print(2 not in squares)

# membership tests for key only not value
# Output: False
print(49 in squares)
```

Output

```
True
True
False
```

6.22 Dictionary Methods

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.

fromkeys(seq[, v])	Returns a new dictionary with keys from `seq` and value equal to `v` (defaults to `None`).
get(key[,d])	Returns the value of the `key`. If the `key` does not exist, returns `d` (defaults to `None`).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the `key` and returns its value or `d` if `key` is not found. If `d` is not provided and the `key` is not found, it raises
popitem()	Removes and returns an arbitrary item (**key, value**). Raises `KeyError` if the dictionary is empty.
setdefault(key[,d])	Returns the corresponding value if the `key` is in the dictionary. If not, inserts the `key` with a value of `d` and returns `d` (defaults to `None`).
update([other])	Updates the dictionary with the key/value pairs from `other`, overwriting existing keys.
values()	Returns a new object of the dictionary's values

6.23 Dictionary Built-in Functions

Built-in functions like **all(), any(), len(), cmp(), sorted(),** etc. are commonly used with dictionaries to perform different tasks.

Function	Description
all()	Return `True` if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return `True` if any key of the dictionary is true. If the dictionary is empty, return `False`.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries. (Not available in Python 3)
sorted()	Return a new sorted list of keys in the dictionary.

Example:1

```
#Python Program to Check if a Given Key Exists in a Dictionary or Not
d={'A':1,'B':2,'C':3}
key=input("Enter key to check:")
if key in d.keys():
      print("Key is present and value of the key is:")
      print(d[key])
else:
      print("Key isn't present!")
```

```
Enter key to check:6
Key isn't present!
```

Example:2

```
#Python Program to Remove the Given Key from a Dictionary
d = {'a':1,'b':2,'c':3,'d':4}
print("Initial dictionary")
print(d)
key=input("Enter the key to delete(a-d):")
if key in d:
    del d[key]
else:
    print("Key not found!")
    exit(0)
print("Updated dictionary")
print(d)
```

```
Initial dictionary
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

Enter the key to delete(a-d):c
Updated dictionary
{'a': 1, 'b': 2, 'd': 4}
```

Example:3

```
#Python Program to Count the Frequency of Words
#Appearing in a String Using a Dictionary
test_string=input("Enter string:")
l=[]
l=test_string.split()
wordfreq=[l.count(p) for p in l]
print(dict(zip(l,wordfreq)))
```

```
Enter string:Hi, How are you?
{'Hi,': 1, 'How': 1, 'are': 1, 'you?': 1}
```

Multiple Choice Questions

1) What will be the output of the following Python code?
   >>>names = ['Amir', 'Bear', 'Charlton', 'Daman']
   >>>print(names[-1][-1])
   a) A b) Daman c) Error **d) n**
2) Suppose list1 is [1, 3, 2], What is list1 * 2?
   a) [2, 6, 4] b) [1, 3, 2, 1, 3] **c) [1, 3, 2, 1, 3, 2]** d) [1, 3, 2, 3, 2, 1]
3) Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is:
   a) [0, 1, 2, 3] b) [0, 1, 2, 3, 4] **c) [0.0, 0.5, 1.0, 1.5]** d) [0.0, 0.5, 1.0, 1.5, 2.0]
4) To insert 5 to the third position in list1, we use which command?
   a) list1.insert(3, 5) **b) list1.insert(2, 5)** c) list1.add(3, 5) d) list1.append(3, 5)
5) Suppose t = (1, 2, 4, 3), which of the following is incorrect?
   a) print(t[3]) **b) t[3] = 45** c) print(max(t)) d) print(len(t))
6) What will be the output of the following Python code?
   >>>t = (1, 2)
   >>>2 * t
   a) (1, 2, 1, 2) b) [1, 2, 1, 2] c) (1, 1, 2, 2) d) [1, 1, 2, 2]
7) What will be the output of the following Python code?
   >>>my_tuple = (1, 2, 3, 4)
   >>>my_tuple.append((5, 6, 7))
   >>>print len(my_tuple)
   a) 1 b) 2 c) 5 d**) Error**
8) Which of these about a dictionary is false?
   a) The values of a dictionary can be accessed using keys
   b) The keys of a dictionary can be accessed using values
   c) Dictionaries aren't ordered
   d) Dictionaries are mutable
9) Which of the following is not a declaration of the dictionary?
   a) {1: 'A', 2: 'B'} b) dict([[1,"A"],[2,"B"]]) c) **{1,"A",2"B"}** d) { }
10) What will be the output of the following Python code snippet?
    a={1:"A",2:"B",3:"C"}
    print (a.get(1,4))
    a)1 **b)A** c)4 d) Invalid syntax forget method

Descriptive Questions

1) What is list in python; explain its merits and demerits.
2) Can you explain tuple methods with suitable examples?
3) How can we create a dictionary and remove the item from it?
4) Write a python program to find second largest number in the given list.
5) Python Program to Generate Random Numbers from 1 to 20 and Append Them to the List
6) Python program to Sort a List of Tuples in Increasing Order by the Last Element in Each Tuple
7) Write a python program to multiply all the items in a dictionary.
8) Python Program to Create a Dictionary with Key as First Character and Value as Words Starting with that Character

Unit 1 - Searching

Module 1 - Introduction to Searching

What is searching?

Searching is an operation or a technique that helps to finds the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not.

Definition:- Searching involves to deciding whether a *search key* is present in the data.

Clifford A. Shaffer [1997] defines searching as a process to determine whether an element is a member of a certain data set.

- The process of finding the location of an element with a specific value (key) within a collection of elements
- The process can also be seen as an attempt to search for a certain record in a file.
    - Each record contains data field and key field
    - Key field is a group of characters or numbers used as an identifier for each record
    - Searching can done based on the key field.

Example: Table of Employee Record.

Index	Emp_ID	Emp_Name	Designation
0	1111	Ashok	Programmer
1	122	Ravi	Clerk
2	211	Kishore	System Analyst

Searching can be done based on certain field: empID, or empl_IC, or empName.

To search empID = 122, give us the record value at index 1.

To search empID = 211, give us the record value at index 2.

To search an element in a given list or array, it can be done in two ways of followed data structure is listed below:

1. Linear Search **or** Sequential Search
2. Binary Search

Module 2 - Linear Search Technique

Linear Search

A linear search is the basic and simple technique of searching algorithm. A linear search searches an element or value from an array or list till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the value index else it return -1. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

Algorithm

- Start from the leftmost element of given list[] and one by one compare element x with each element of list[]
- If x matches with any of the element, return the index value.
- If x doesn't match with any of elements in list[] , return -1 or element not found.

Example with Implementation

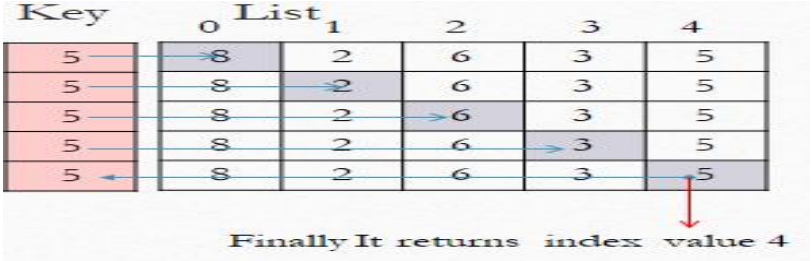
The list given below is the list of elements in an unsorted array. The array contains 5 elements. Suppose the element to be searched is '5', so 5 is compared with all the elements starting from the 0th element and searching process ends where 5 is found or the list ends. list_1=

8	2	6	3	5

The performance of the linear search can be measured by counting the comparisons done to find out an element.

Sample Tracing:-

- Here Searching Element is x=5 and Searching Start from the index 0
- First com, pare with 8, not equal, so next with 2 and so on, until the element is 5.
- Once 5 found stop the procedure.



Sequential Search Analysis:

- Searching time for sequential search is O(n).
- If the searched key is located at the end of the list or the key is not found, then the loop will be repeated based on the number of element in the list, O(n).
- If the list can be found at index 0, then searching time is, O(1).

- The number of comparison is O(n).
- The worst-case and average-case **time complexity** is O(n). The best-case is O(1).

Sample Program:-

```
# Searching an element in a list/array in python
# can be simply done using 'in' operator
# Example:
l1=[8,2,6,3,5]
x=5

if x in l1:
    print(l1.index(x))

# If you want to implement Linear Search in python
# Linearly search x in list1
# If x is present then return its location
# else return -1

def search(l1, x):

    for i in range(len(l1)):

        if l1[i] == x:
            return i

    return -1
print(search(l1,5))
```

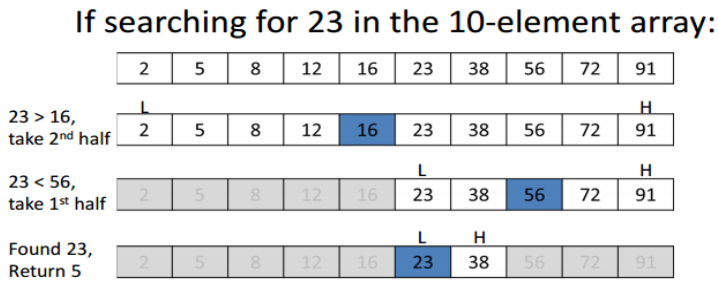
Module 3 - Binary Search Technique

Binary Search

Binary search is a very fast and efficient searching technique and it is applied on the sorted array or list. In a binary search, it first compares the value with the elements in the middle position of the array or list. If the value is matched, then it returns the value of index. If the value is less than the middle element, then it must lie in the lower **(left)** half of the array and if it is greater than the element then it must lie in the upper **(right)** half of the array or list. It repeats this procedure on the lower **(left)** or upper **(right)** half of the array until the value is found or the interval is empty. Binary Search is useful when there are large numbers of elements in an array or list

Example with Implementation

- As we can see the list is sorted in ascending order, to search an element 23.
- Binary Search is applied on sorted lists only, so that we can make the search fast, by breaking the list every time.
- Start with middle element 16, it its Equal to the number we are searching (23), then it returns value of the index
- If it is more than middle element 16, then it moves to the UPPER (right)
- If it is less than middle element 16, then it moves to the LOWER (left)and then, it REPEATS, till you find the number or end of the list



Program:

```
# Python code to implement iterative Binary Search.
# It returns location of x in given list
# if present, else returns -1
def binarySearch(list_1,start,end,key):
    while start <= end: #until list is empty or found
        mid = (start + end)//2;
        # Check if x is present at mid
        if list_1[mid] == key:
            return mid
        # If x is greater, ignore left half
        elif list_1[mid] < key:
            start = mid + 1
        # If x is smaller, ignore right half
        else:
            end = mid - 1
    # If we reach here, then the element
    # was not present
    return -1
# Test list_1
list_1 = [ 2, 3, 4, 10, 40 ]
key = 40
# Function call
result = binarySearch(list_1,0,len(list_1)-1, key)

if result != -1:
    print("Element is present at index %d" % result)
else:
    print("Element is not present in List")
```

Time Complexity:

The time complexity of Binary Search can be written as T(n) = T(n/2) + c

.

T(n)=T(n/2)+1

T(n/2)= T(n/4)+1+1

Put the value of The(n/2) in above so
T(n)=T(n/4)+1+1 . . . . T(n/2^k)+1+1+1.....+1

=T(2^k/2^k)+1+1....+1 up to k

=T(1)+k

As we taken 2^k=n

K = log n

So Time complexity is O(log n)

a binary search (half the elements) until you found it. In a formula this would be this:

$1 = N / 2^x$

multiply by 2^x:

$2^x = N$

now do the $\log_2$:

$\log_2(2^x) = \log_2 N$
$x * \log_2(2) = \log_2 N$
$x * 1 = \log_2 N$

this means you can divide log N times until you have everything divided. Which means you have to divide log N ("do the binary search step") until you found your element.

Multiple Choice Questions

1) The worst case occurs in linear search algorithm when …….
   a) Item is somewhere in the middle of the array
   b) Item is not in the array at all
   c) Item is the last element in the array
   d) Item is the last element in the array or item is not there at all
2) Searching is an operation or a technique that helps to finds the place of a given …… in the list
   a) element or value b) index c) length d) type
3) Linear Search is applied on the unsorted or unordered list
   a) True b) False c) True and False d) Both
4) Binary Search is applied on …… lists and starts with…… element?
   a) Unsorted ,middle b) Sorted, last c) Unsorted, first d) Sorted, middle
5) In which case, linear searching technique implemented?
   a) When the list has only a few elements
   b) When performing a single search in an unordered list
   c) Used all the time
   d) When the list has only a few elements and When performing a single search in an unordered list
6) Which of the following is a disadvantage of linear search?
   a) Requires more space
   b) Greater time complexities compared to other searching algorithms
   c) Not easy to understand
   d) Not easy to implement
7) The array is as follows: 1,2,3,6,8,10. At what time the element 6 is found? (By using linear search(recursive) algorithm)
   a) 4th call b) 3rd call c) 6th call d) 5th call
8) Given an input arr = {2,5,7,99,899}; key = 899; What is the level of recursion(mid value)?
   a) 5 b) 2 c) 3 d) 4
9) Given an array arr = {5,6,77,88,99} and key = 88; How many iterations are done until the element is found?
   a) 1 b) 3 c) 4 d) 2
10) What is the time complexity of binary search with iteration?
   a) O(nlogn) b) O(logn) c) O(n) d) $O(n^2)$

Descriptive Questions

1) Describe Searching technique with example
2) Explain different searching techniques
3) Write a python program to find a key values using linear search method?
   Key=5,18, L=[4,5,2,3,6,1,12,45,18,23]
4) Write a Binary Search Tracing from given list and key?
   Key=51, L= [41, 51, 53, 62, 73]
5) Define method Sequentialsearch() and pass two arguments as group of elements and key value. Return True if key is existed in the group otherwise returns False
   Example: Case1: SequentialSearch([34,2,9,10,4],10) ---------> True
   Case2: SequentialSearch([34,2,9,10,4],8) ----------> False

Unit 2 - Introduction to Sorting

Module 1 - Introduction to Sorting Techniques

Sorting is nothing but storage of data in sorted order; it can be in ascending or descending order. The term Sorting comes into the picture with the term Searching. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in the merit list, a particular telephone number, any particular page in a book etc. Sorting arranges data in a sequence, which makes searching easier and every record, which is going to be sorted, will contain one key. Based on the key the record will be sorted.

For example, suppose we have a record of the students, every record will have the following data:

Roll No **Name** **Age** **Class**

Here Student **Roll No**. Can be taken as key for sorting, the records in ascending or descending order. Now suppose we have to search a Student with **Roll No**. 15, we don't need to search the complete record we will simply search between the Students with **Roll No.** 10 to 20

Similarly, in a telephone dictionary, every record consists of the name of a person, address and telephone number. In this, the telephone number is a unique or key field. We can sort the data of the dictionary on this telephone field. Alternatively, we can also sort data either numerically or alphanumerically.

When we can adjust the data to be sorted in the main memory itself without any need of another auxiliary memory, then we call it as **Internal Sorting.**

On the other hand, when we need auxiliary memory for storing intermediate data during sorting, then we call the technique as **External Sorting.**

Sorting Efficiency

If you ask me, how I will arrange a deck of shuffled cards in order, I would say, I will start by checking every card, and making the deck as I move on. It can take me hours to arrange the deck in order, but that's how I will do it.

The two main criteria to judge which algorithm is better than the other have been:

1) The execution time of the program, (Time Complexity)
   - ✓ Time taken for execution of program (sort the given data)
2) The space taken by the program (Space Complexity
   - ✓ Memory Space required to do so.

Types of Sorting Techniques

There are many types of Sorting techniques, differentiated by their efficiency and space requirements.

Following are some sorting techniques

- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Heap Sort

Note: - At this level we are going to study Bubble Sort, Insertion Sort and Selection Sort

Bubble Sort

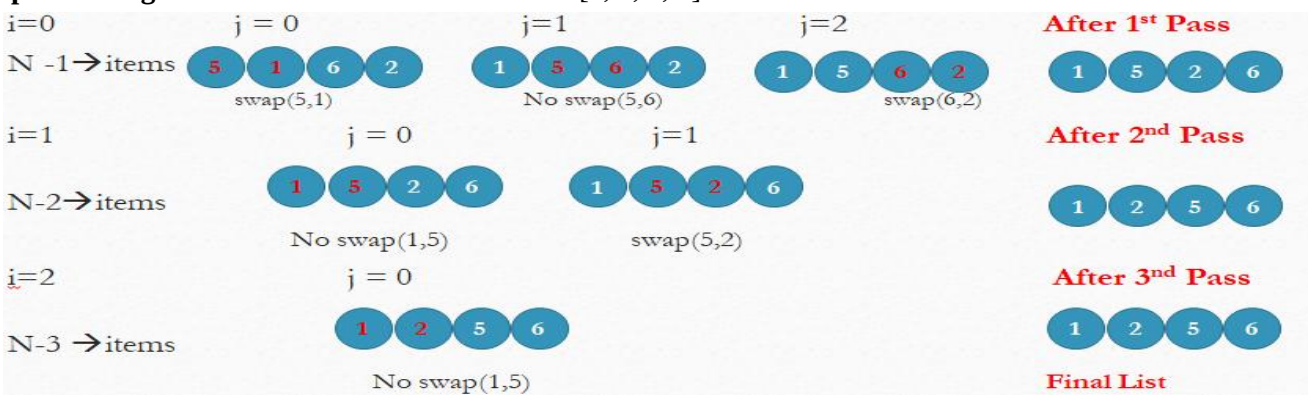
Bubble Sort is an algorithm which is used to sort **N** elements that are given in a memory for a List with **N** number of elements. Bubble Sort compares the entire element one by one and sort them based on their values. It is called Bubble sort. With each iteration the largest element in the list bubbles up towards the last place, just like water bubble rises up to the water surface. Sorting takes place by stepping through all the data items one-by-one in pairs and comparing adjacent data items and swapping each pair that is out of order.

In each **Case** it generate two parts of entire list, sorted and unsorted parts

OR **Bubble Sort** is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Note: - it is a slow sorting algorithm, it as good as selection sort but bubble and selection sorting are slow sorting algorithms.

Sample Tracing -1 Let's consider a List with values [5, 1, 6, 2]



Sample Tracing -2

First Pass:-

(5 **1** 4 2 8)→(1 **5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1**5 4** 2 8)→(1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) →(1 4 2 **5 8**),Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:-

(**1 4** 2 5 8)→ (**1 4** 2 5 8)

(1 **4 2** 5 8)→(1 **2 4** 5 8), Swap since 4 > 2

(1 2 **4 5** 8)→ (1 2 **4 5** 8)

(1 2 4 **5 8**)→ (1 2 4 **5 8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) →(1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

Source Code with Explanation

The program is to sort list using Bubble Sort. Although the program of the logic will sort an unsorted list, still the given program is not efficient because as per the logic of the given program or source code.

The for-loop will keep executing for five iterations, even if the list gets sorted after the second iteration.

```
# Python program for implementation of Bubble Sort
def bubbleSort(list_1,length):
        #Traverse through all list_1 elements
        for i in range(length):
                # Last i elements are already in place
                for j in range(length-i-1):
                        # traverse the list_1 from 0 to n-i-1
                        # Swap if the element found is greater
                        # than the next element
                        if list_1[j] > list_1[j+1] :
                                list_1[j], list_1[j+1] = list_1[j+1], list_1[j]
# Driver code to test above
list_1 = [64, 34, 25, 12, 22, 11, 90]
print ("UnSorted list_1 is:",list_1)
bubbleSort(list_1,len(list_1))
print ("Sorted list_1 is:",list_1)
```

How to overcome the Redundancy of bubble sort

Hence we can insert a flag and can keep checking whether swapping of elements is taking placed or not in the following iteration. If no swapping is taking placed, it means the array or list is sorted and we can jump out of the for loop, instead of executing all the iterations. In the given code, in a complete single cycle of **i**teration **(inner for loop)**, has no swapping takes places, then a flag value will remains**0** and then it will break, out of the for loop, because the array or list has been already sorted.

```
# Python program for implementation of Bubble Sort
def bubbleSort(list_1,length):
        #Traverse through all list_1 elements
        for i in range(length):
                flag=0
                for j in range(length-i-1):
                        if list_1[j] > list_1[j+1]:
                                list_1[j], list_1[j+1] = list_1[j+1], list_1[j]
                                flag=1
                if flag==0:
                        break
        #return list_1

# Driver code to test above
list_1 = [64, 34, 25, 12, 22, 11, 90]
print ("UnSorted list_1 is:",list_1)
bubbleSort(list_1,len(list_1))
print ("Sorted list_1 is:",list_1)
```

Complexity Analysis of Bubble Sorting

In Bubble Sort, n-1 comparisons will be done in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on.

So the total number of comparisons will be (n-1) + (n-2) + (n-3) +.....+3+2+1. Sum = n (n-1)/2. i.e. $O(n^2)$

Hence the **Worst and Average Case Time Complexity** of Bubble Sort is $O(n^2)$.

The main advantage of Bubble Sort is the simplicity of the algorithm.

Space complexity for Bubble Sort is O(1), because only single additional memory space is required

i.e. for temp variable, it is optional in python

Best-case Time Complexity will be O (n), it is when the list is already sorted

Module 2 - Insertion Sort & Selection Sort

Insertion Sort

Insertion sort is a **sorting** algorithm in which the elements are transferred one at a time to the right position.

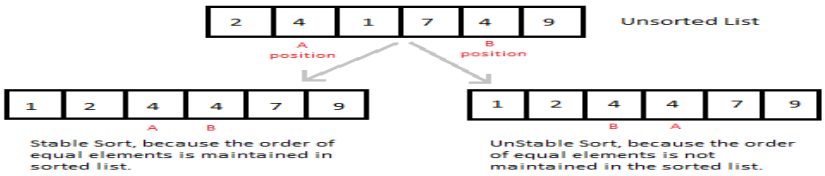
Example: - Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Insertion sort is a simple Sorting algorithm which sorts the array or list by shifting elements one by one. Following are some of the important characteristics of Insertion Sort.

- It has one of the simplest implementation
- It is efficient for smaller data sets, but very inefficient for larger lists.
- Insertion Sort is adaptive, that means it reduces its total number of steps in a given partially sorted list, hence it increases its efficiency.
- It is better than Selection Sort and Bubble Sort algorithms.
- Its space complexity is less. Like Bubble Sorting, insertion sort also requires a single additional memory space.
- It is a **Stable** sorting, as it does not change the relative order of elements with equal keys

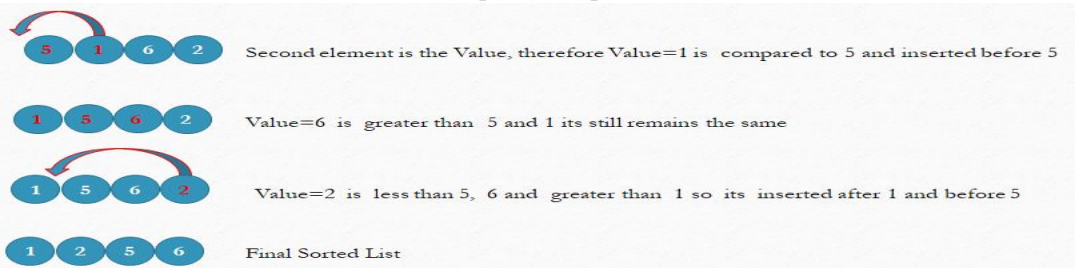
What is stable sorting?

It does not change the relative order of elements with equal keys.

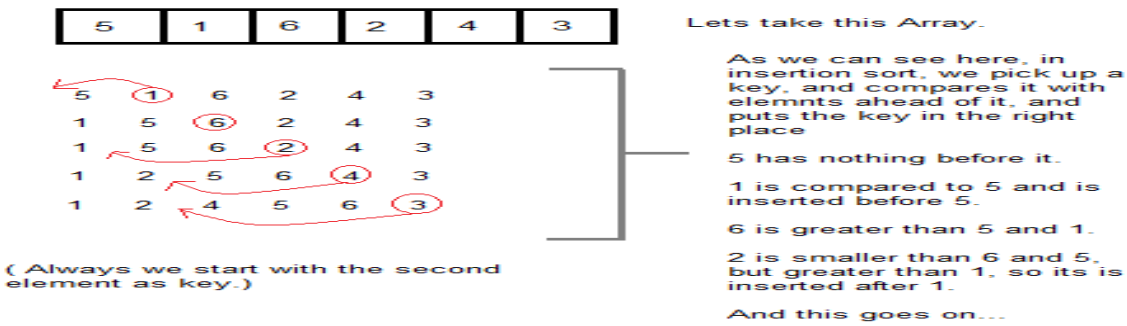


Sample Tracing of Insertion Sort - 1

Let's consider a List with values [5, 1, 6, 2]



Sample Tracing of Insertion Sort - 2



Source code of the Insertion Sort

- Now let's, understand the simple insertion sort algorithm. We took list with 6 integers. We took a variable **Value**, in which we put each element of the list, in each pass, starting from the second element that is **List [1]**.
- Then using the while loop, we iterate, until **gap** becomes greater than zero or we find an element which is greater than **Value**, and then we insert the key at that position.
- In the above list, first we pick 1 as Value, we compare it with 5(element before 1), 1 is smaller than 5, we shift 1 before 5.
- Then we pick 6, and compare it with 5 and 1, no shifting this time.
- Then 2 becomes the key and is compared with, 6 and 5, and then 2 is placed after 1 and this goes on, until complete list gets sorted

```
# Function to do insertion sort
def insertionSort(List,length):
    # Traverse through 1 to length
    for i in range(1, length):
        Value = List[i]
        # Move elements of List[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        gap = i
        while gap >0 and List[gap-1] > Value :
            List[gap] = List[gap-1]
            gap-=1
        List[gap] = Value

List=[5, 1, 6, 2, 4, 3]
print("UnSorted List:-",List)
insertionSort(List,len(List))
print("Sorted List:-",List)
```

Complexity Analysis of Insertion Sorting

Worst Case Time Complexity

T(n)=(c1+c3)(n-1)+{1+2+…….n-1}c2=(c1+c3)(n-1)+(n(n-1)/2)c2=a**(n^2)+bn+c**

i.e, O(n^2) when list is reverse order

Average Case Time Complexity also Big O of n^2 = O(n^2)

Best-case Time Complexity

T(n)=(c1+c3)(n-1)=c1n+c3n-(c1+c3)=an+c=**O(n)** When the list is already sorted.

Space complexity for Insertion Sort is **O(1)**,

Because only single additional memory space is required

Note:-

- It is better than Selection Sort and Bubble Sort algorithms. Its space complexity is less.
- Bubble Sorting, insertion sort also requires a single additional memory space.

Selection Sort

The **selection sort** algorithm sorts list by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm or logic will maintains two **sublists** in a given list.

1. The **sublist** which is already sorted.
2. Remaining **sublist** which is unsorted.

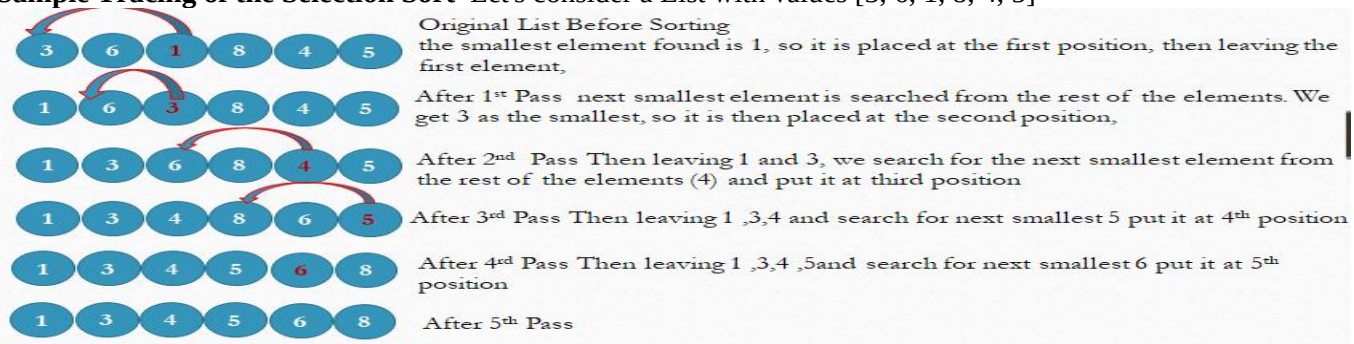
In each iteration of selection sort, the minimum element (considering ascending order) from the unsorted sublist is picked and moved to the sorted sublist.

Selection sorting is conceptually the simplest sorting algorithm. These algorithms first finds the smallest element in the list and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire list is sorted.

Note:

Selection sort is an unstable sort i.e. it might change the occurrence of two similar elements in the list while sorting. But it can also be a stable sort when it is implemented using linked list data structures.

Sample Tracing of the Selection Sort Let's consider a List with values [3, 6, 1, 8, 4, 5]



Source code of the Selection Sort

In the first pass, find the smallest element is 1, so it is placed at the first position, then leaving the first element. Next smallest element is searched from the rest of the elements. We get 3 as the smallest, so it is then placed at the

```
#Functon to do selectionSort
def selectionSort(List,length):
    # Traverse through all list elements
    for i in range(length):
        #Find the minimum element from unsorted List
        minIndex=i
        for j in range(i+1,length):
            if List[j]<List[minIndex]:
                minIndex=j
        List[i],List[minIndex]=List[minIndex],List[i]
    #return List

List =[3,6,1,8,4,5]
print("UnSorted List",List)
selectionSort(List,len(List))
print("Sorted List",List)
```

second position. Then leaving 1 and 3, we search for the next smallest element from the rest of the elements and put it at third position Keep doing this until list is sorted

Complexity Analysis of Selection Sorting

Worst Case Time Complexity: T(n)=c1(n-1)+(n(n-1)/2)c2+c3(n-1)=a(n^2)+bn+c; i.e, O(n^2) when list is reverse order; Average Case Time Complexity also Big O of n^2 = O(n^2), Best-case Time Complexity = O(n^2)

Multiple Choice Questions

1) Which of the following is not a stable sorting algorithm?
   a) Insertion sort b) Selection sort c) Bubble sort d) All of the above
2) The worst case occurs in linear search algorithm when …….
   a) Item is somewhere in the middle of the array b) Item is not in the array at all
   c) Item is the last element in the array d) Item is the last element in the array or item is not there at all
3) Searching is an operation or a technique that helps to finds the place of a given …… in the list
   a) element or value b) index c) length d) type

4) Linear Search is applied on the unsorted or unordered list
   a) True b) False c) True and False d) Both
5) Binary Search is applied on …… lists and starts with…… element?
   a) Unsorted ,middle b) Sorted, last c) Unsorted, first d) Sorted, middle
6) Which of the following Sort the simplest sorting method that works by repeatedly swapping the adjacent elements if they are in wrong order
   a) Selection b) Insertion c) Bubble d) All of the above
7) What is meant by stable sorting?
   a) It does not change the relative order of elements with equal keys
   b) It does change the relative order of elements with equal keys
   c) It does not change the relative order of elements with indexes
   d) It does change the relative order of elements with equal values
8) What is the Worst and Average Case Time Complexity of Bubble Sort?
   a) O(n) b. O(logn) c. $O(n^2)$ d. log n
9) Which of the following is the best sorting method?
   a) Insertion b) Selection c) Bubble d) Binary Search
10) Which of the following is the slow sorting method?
   a) Insertion b) Selection c) Bubble d) Binary Search
11) The number of interchanges required to sort 5, 1, 6, 2 4 in ascending order using Bubble Sort is?
   a) 5 b) 6 c)7 d) 8
12) Which of the following sorting methods would be most suitable for sorting a list which is almost sorted?
   a) Selection b) Insertion c) Bubble d) None of the above

Descriptive Questions

1) Write a python program to find a key values using linear search method? Key=5,18, L=[4,5,2,3,6,1,12,45,18,23]
2) Write a Binary Search Tracing from given list and key? Key=51, L= [41, 51, 53, 62, 73]
3) Write a python program to sort the given list using bubble sort? L=[4,5,2,3,6,1,12,45,18,23]
4) Write the implementation of selection sort using the given list? L=[4,5,2,3,6,1,12,45,18,23]
5) Write the sample tracing of insertion sort using the given list? L=[4,5,2,3,6,1,12,45,18,23]
6) What is linear search and write the implementation of the linear search program?
7) Write an example code for linear search and write the step by step process or passes or tracing using a list A= [5, 13, 81, 4, 16, 12, 1], search for an element (key =12) in the list using linear search?
8) Write the Definition of the binary search and implement binary search program?
9) Write an example code for implementation of binary search and write the step by step process or passes or tracing using a sorted list L=[20,34,45,67,74,89,99] ,search for an element (key =89) in the list using binary search?
10) Write the difference between linear, binary searching techniques and implement the code or programs of the linear and binary search?
11) Write the Definition of the sorting; explain the sorting efficiency, and importance of the sorting in python data structures with appropriate example program?
12) Define a bubble/selection/insertion sort technique and write the appropriate code for it,along with example program?
13) Write a sample programming code for bubble/selection/insertion sort and write the number of passes or tracing to the given list L= [3, 6, 1, 8, 4, 5]

Unit 3 – File and Exception handling

Module 1 – File handling

What is a file?

What if the data with which, we are working or producing as output is required for later use? Result processing done in Term Exam is again required for Annual Progress Report. Here if data is stored permanently, its processing would be faster. This can be done if we are able to store data in secondary storage media i.e. Hard Disk, which we know is permanent storage media. Data is stored using file(s) permanently on secondary storage media. You have already used the files to store your data permanently - when you were storing data in Word processing applications, Spreadsheets, Presentation applications, etc. All of them created data files and stored your data, so that you may use the same later on. Apart from this, you were permanently storing your python scripts (as .py extension) also.

Here is a basic definition of file handling in Python, "File is a named location on the system storage which records data for later access. It enables persistent storage in a non-volatile memory i.e. Hard disk."

In python files are simply stream of data, so the structure of data is not stored in the file, along with data. Basic operations performed on a data file are:

- Naming a file
- Opening a file
- Reading data from the file
- Writing data in the file
- Closing a file

Using these basic operations, we can process a file in many ways, such as
creating a file
traversing a file for displaying the data on screen
Appending data in file
Inserting data in file
Deleting data from file
create a copy of file
Updating data in the file, etc

File types

Python allows us to create and manage two types of file

Text:

A text file is usually considered as a sequence of lines. A line is a sequence of characters (ASCII), stored on permanent storage media. Although default character coding in python is ASCII but using constant u with string, supports Unicode as well. As we talk of lines in a text file, each line is terminated by a special character, known as End of Line (EOL). From strings, we know that \n is a newline character. So at the lowest level, a text file will be a collection of bytes. Text files are stored in human readable form and they can also be created using any text editor.

Binary:

A binary file contains arbitrary binary data i.e. numbers stored in the file, can be used for numerical operation(s). So when we work on a binary file, we have to interpret the raw bit pattern(s) read from the file into the correct type of data in our program. It is perfectly possible to interpret a stream of bytes originally written as a string, as a numeric value. But we know that will be an incorrect interpretation of data and we are not going to get desired output after the file processing activity. So in the case of a binary file, it is extremely important that we interpret the correct data type while reading the file. Python provides a special module(s) for encoding and decoding of data for the binary file.

Text File	Binary File
Its Bits represent character.	Its Bits represent a custom data.
Less prone to get corrupt as change reflects as soon as made and can be undone.	Can easily get corrupted, corrupt on even a single bit change
Store only plain text in a file.	Can store different types of data (audio, text, and image) in a single file.
Widely used file format and can be opened in any text editor.	Developed for an application and can be opened in that application only.
Mostly .txt and .rtf are used as extensions to text files.	Can have any application defined extension.

File Operations

Opening a file: How do you write data to a file and read the data back from a file? You need to first create a file object that is associated with a physical file. This is called opening a file. The syntax for opening a file is:

fileVariable = open(filename, mode) The open function returns a file object for filename. The mode parameter is a string that specifies how the file will be used (for reading or writing, Note: you can find the tables for more file modes)

For example, the following statement opens a file named Scores.txt in **the current directory** for reading:

file= open("Scores.txt", "r")
You can also use the absolute filename to open the file in Windows, as follows:
file = open(r"c:\pybook\Scores.txt", "r")
Example code for Opening a file:
file = open("Scores.txt", "r")
in a open(file_name,file_accessmode) here Scores.txt file name. ## r is the file access mode i.e read only
print(file.read()) ## read() method will read the data from file ## print() method will prints the data from file object
file.close() ## closing the file[will be explained in the closing file]

Absolute and relative paths

Absolute Path:

An absolute path is a path that contains the entire path to the file or directory that you need to access. This path will begin at the home directory of your computer and will end with the file or directory that you wish to access.

Example:

/home/your-username/earth-analytics/data/field-sites/california/colorado/streams.csv (in Unix/Linux OS)
G:\RGUKT\PUC\Sem2\index.html (in windows OS)

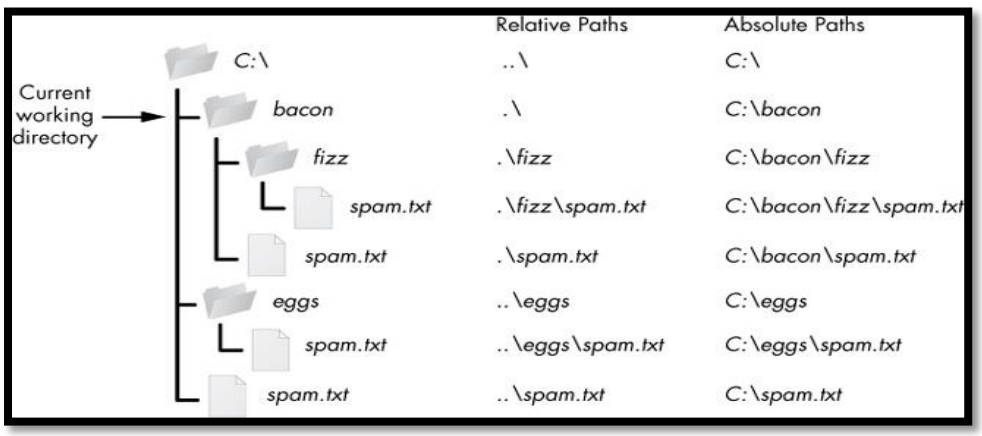
Relative Path:

A relative path is the path that (as the name sounds) is relative to the working directory location on your computer.

Example:

/data/field-sites/california/colorado/streams.csv (in Unix/Linux OS)
PUC\Sem2\index.html (in windows OS)

The following picture contains absolute and relative paths for folders and files in the working directory C:\bacon



File access modes

Sr.No.	Modes & Description
1	r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode
3	r+ Opens a file for both reading and writing. The file pointer placed at the beginning of the file
4	rb+ Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6	w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
7	wb Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
8	wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

9	a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
11	ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
12	ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

File methods: A file object contains the methods for reading and writing data are as follows

read([number.int): str ---> Returns the specified number of characters from the file. If the argument is omitted, the entire remaining contents in the file are read.

readline(): str ----> Returns the next line of the file as a string

readlines(): list ----> Returns a list of the remaining lines in the file.

write(s: str): None ---> Writes the string to the file.

close(): None ---> Closes the file

write(str): Write a string to the file. There is no return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

writelines(sequence): Write a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings. There is no return value. (The name is intended to match readlines(); writelines() does not add line separators.) After a file is opened for writing data, you can use the write method to write a string to the file

Here is writing demo into files

```
def main():
    # Open file for output
    outfile = open("student.txt", "w")
    # Write data to the file
    outfile.write("Bhuvan\n")## \n is a newline character in the files
    outfile.write("Sreekanth\n")
    outfile.write("Jagan\n")
    outfile.write("Sreenu\n")
    outfile.close()# Close the output file
main()# Call the main function
```

The program opens a file named student.txt using the w mode for writing data (line 3). If the file does not exist, the open function creates a new file. If the file already exists, the contents of the file will be overwritten with new data. You can now write data to the file. When a file is opened for writing or reading, a special marker called a file pointer is positioned internally in the file. A read or write operation takes place at the pointer's location. When a file is opened, the file pointer is set at the beginning of the file. When you read or write data to the file, the file pointer moves forward

Testing a File's Existence:

To prevent the data in an existing file from being erased by accident, you should test to see if the file exists before opening it for writing. The isfile function in the os.path module can be used to determine whether a file exists.

Testing a File's Existence Code:

```
import os.path
if os.path.isfile("student.txt"):
    print("student.txt exists")
else:
    print("student.txt DOES NOT exists")
```

Here isfile("students.txt.txt") returns True if the file students.txt exists in the current directory.

Reading Data:

After a file is opened for reading data, you can use the read method to read a specified number of characters or all characters from the file and return them as a string, the readline() method to read the next line, and the readlines() method to read all the lines into a list of strings.

Reading Data demo code:

```
def main():
    # Open file for input
    infile = open("student.txt", "r")
    print("(1) Using read(): ")
    print(infile.read())
```

```
    """ read() method reads all characters from the file and returns them as a string"""
    infile.close() # Close the input file
    # Open file for input
    infile = open("student.txt", "r")
    print("\n(2) Using read(number): ")
    s1=infile.read(4)
"""read(4) -- read(number) method to read the specified number of characters from the file. Invoking infile.read(4) reads 4 characters """
    print(s1)
    s2 = infile.read(10)
    print(repr(s2))
"""The repr(s)  - repr(s2)-function returns a raw string for s, which causes the escape sequence to be displayed as literals, as shown in the output. """
```

Output:

```
>>>
(1) Using read():
Bhuvan
Sreekanth
Sreenu

(2) Using read(number):
Bhuv
'an\nSreekan'

(3) Using readline():
'Bhuvan\n'
'Sreekanth\n'
'Sreenu\n'
''

(4) Using readlines():
['Bhuvan\n', 'Sreekanth\n', 'Sreenu\n']
```

```
    infile.close() # Close the input file
    # Open file for input
    infile = open("student.txt", "r")
    print("\n(3) Using readline(): ")
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()
    line4 = infile.readline()
    print(repr(line1))
    print(repr(line2))
    print(repr(line3))
    print(repr(line4))
    infile.close() # Close the input file
    # Open file for input
    infile = open("student.txt", "r")
    print("\n(4) Using readlines(): ")
    print(infile.readlines())
"""readlines() method to read all lines and return a list of strings"""
    infile.close() # Close the input file
    main() # Call the main function
```

Reading all data from the file

Programs often need to read all data from a file. Here are two common approaches to accomplishing this task:

1. Use the read() method to read all data from the file and return it as one string.
2. Use the readlines() method to read all data and return it as a list of strings.

These two approaches are simple and appropriate for small files, but what happens if the file is so large that its contents cannot be stored in the memory? You can write the following loop to read one line at a time, process it, and continue reading the next line until it reaches the end of the file

Using while loop

```
infile = open("student.txt", "r")
line = infile.readline() # Read a line
while line != '':
    print(line)
    # Process the line here ...
    # Read next line
    line = infile.readline()
infile.close()
```

Using for loop

```
infile = open("student.txt", "r")
for line in infile:
    print(line)
infile.close()
```

Source code for copying a file

Illustrates a program that copies data from a source file to a target file and counts the number of lines and characters in the file.

```
import os.path
import sys
def main():
    # Prompt the user to enter filenames
    f1 = input("Enter a source file: ").strip()
    f2 = input("Enter a target file: ").strip()
    # Check if target file exists
    if os.path.isfile(f2):
        print(f2 + " already exists")
        sys.exit()
    # Open files for input and output
    infile = open(f1, "r")
    outfile = open(f2, "w")
    # Copy from input file to output file
    countLines = countChars = 0
    ##initialize countLines and countChars
    for line in infile:
        ##increase countLines
        countLines += 1
        ##increase countChars
        countChars += len(line)
        outfile.write(line)
    print(countLines, "lines and", countChars, "chars copied")
    infile.close() # Close the input file
    outfile.close() # Close the output file
main() # Call the main function
```

Appending Data

You can use the mode to open a file for appending data to the end of an existing file.

```
def main():
    # Open file for appending data
    outfile = open("Info.txt", "a")
    outfile.write("\nPython is interpreted\n")
    outfile.close()# Close the file
main() # Call the main function
```

Writing and Reading Numeric Data

To write numbers to a file, you must first convert them into strings and then use the write method to write them to the file. In order to read the numbers back correctly, separate them with whitespace characters, such as " " or \n.

```
from random import randint.
def main():
    # Open file for writing data
    outfile = open("Numbers.txt", "w")
    for i in range(10):
        outfile.write(str(randint(0, 9)) + " ")
    outfile.close()
    # Open file for reading data
    infile = open("Numbers.txt", "r")
    s = infile.read()
    numbers = [eval(x) for x in s.split()]
    for number in numbers:
        print(number, end = " ")
    infile.close() # Close the file
main() # Call the main function
```

Closing a file:

When you're done working, you can use the ***file.close()*** command to end things. What this does is close the file completely, terminating resources in use, in turn freeing them up for the system to deploy elsewhere.

It's important to understand that when you use the ***file.close()*** method, any further attempts to use the file object will fail.

Syntax: *fileobject.close()*

Example:

```
f = open("data.txt", 'w')
print(f.closed)

f.close()
print(f.closed)
```

Output:

```
False
True
```

The tell() method:

Thismethod returns an integer giving the current position of object in the file. The integer returned specifies the number of bytes from the beginning of the file till the current position of file object.

Syntax: *fileobject.tell()*

Example:

```
fr = open('RGUKT.txt', 'r')
fr.readline()
print("current location: ", fr.tell())
fr.readline()
print("current location: ", fr.tell())
fr.close()
```

Output:

```
current location:  8
current location:  13
```

The seek() method:

This method can be used to position the file object at particular place in the file.

Syntax: *fileobject.seek(offset [, from_what])*

The offset argument indicates the number of bytes to be moved.

If from_what value is 0, the beginning of the file to seek. If it is set to1, the current position is used. If it is set to 2 then the end of the file would be taken as seek position.

Value	Reference point
0	Beginning of the file
1	Current position of file
2	End of file

Value	OS module member	Reference point
0	os.SEEK_SET	Beginning of the file
1	os.SEEK_CUR	Current position of file
2	os.SEEK_END	End of file

Default value of from_what is 0, i.e. beginning of the file. OS module has the following members.

In text files (those opened without a 'b' in the mode string), only seeks relative to the beginning of the file [os.SEEK_SET] are allowed i.e. values 1 and 2 do not work with text files in Python 3+ versions.

Example 1:

```
f = open('RGUKT.txt','r')
print("Current position: ", f.tell())
f.seek(8)
print("Current position: ", f.tell())
print(f.readline())
f.close()
```

Output:

```
Current position:  0
Current position:  8
RKV
```

Example 2:

```
f = open('RGUKT.txt','rb')
print("Current position: ", f.tell())
f.seek(-20, 2)
print("Current position: ", f.tell())
print(f.readline())
f.close()
```

(or)

```
import os
f = open('RGUKT.txt','rb')
print("Current position: ", f.tell())
f.seek(-20, os.SEEK_END)
print("Current position: ", f.tell())
print(f.readline())
f.close()
```

Output:

```
Current position:  0
Current position:  13
b'Srikakulam\r\n'
```

Pickle module:

We know that the methods provided in python for writing / reading a file works with string parameters. So when we want to work on binary file, conversion of data at the time of reading, as well as writing is required.

Pickle module can be used to store any kind of object in file as it allows us to store python objects with their structure in binary format. It provides two main methods for the purpose, dump and load.

The dump() method:
This method is used to write data/object in file which is opened in binary access mode.
Syntax:
import pickle
pickle.dump(object, fileobject)
Example:
```
import pickle
l = [1,2,3,4,5,6]
file = open('list.dat', 'wb')
pickle.dump(l,file)
file.close()
```
The load() method:
It is used to read data/object from binary file which was stored using dump() method.
Syntax:
import pickle
object = pickle.load(fileobject)
Example:
```
import pickle
f = open('list.dat', 'rb')
data = pickle.load(f)
print(data)
f.close()
```
Output:
```
[1, 2, 3, 4, 5, 6]
```
With statement:
The **with** statement creates a context manager and it will automatically close the file handler for you when you are done with it.
Syntax: *with open(filename, mode) as file object:*
Example 1:
```
with open("States.txt") as f:
    print(f.read())
```
Output:
```
Andhra Pradesh
Telangana
Delhi
Goa
Karnataka
Tamilnadu
Maharastra
```
Example 2: Opening multiple files using with statement
```
with open("States.txt") as f, open("Names.txt", "w") as n :
    line = f.readline()
    while line:
        n.write(line)
        line = f.readline()
    print("State names are copied to Names.txt file")
```
Output:
State names are copied to Names.txt file

Multiple Choice Questions

1) Which of the following function call can be used to read "n" number of characters from a file?
   a) fileobject.read(n) b) fileobject.readline(n) c) fileobject.readlines(n) d) none of the above
2) What does the <readlines()> method returns?
   a) A string b) A list of lines c) A list of single characters d) A list of integers
3) Does tell() function has parameters?
   a) Yes b) No
4) Which function is used to write a sequence of strings to the file?
   a) writeline() b) write() c) writelines() d) None of the above
5) What is the meaning of 'a' mode?
   a) append and write b) append and read c) append d) None of these
6) What does tell() function do?
   a) Returns the file's current position b) Returns the files previous position
   c) Returns the files end position d) none of these
7) Which of the following function reads the text from file by line by line?
   a) fileobject.readLines() b) fileobject.readLine() c) fileobject.readline() d) fileobject.readlines()
8) How many arguments can we pass to open function?
   a) 1 b) 2 c) 3 d) Any of the above
9) What is the last action that must be performed on a file?
   a) Save b) Close c) Write d) Append
10) What is the data type of data read from a file?
   a) Integer b) Real c) Boolean d) String

11) Which of the following statements correctly explain the function of seek() method?
   a) Tellthe current position within the file.
   b) Indicate that the next read or write occurs from that position in a file.
   c) Determine if you can move the file position or not.
   d) Move the current file position to a different location at a defined offset.

12) What is the use of 'x' mode?
   a) To create a new empty file b) To create a new file for reading
   c) To create a new files for writing d) to create a new file for appending

Solved Problems:

1) Write a Python program to compare two files to check whether they are identical or not.

```
import os
fn = input("Enter first file name: ")
sn = input("Enter second file name: ")

if os.path.exists(fn) and os.path.exists(sn):
   f = open(fn)
   s = open(sn)
   fline = f.readline()
   sline = s.readline()
   while fline and sline:
      if fline != sline:
         print("Files are not identical")
         break
      fline = f.readline()
      sline = s.readline()
   else:
      print("Both files are identical")
   f.close()
   s.close()
else:
   print("Error: Couldn't open file")
```

Output:

```
Enter first file name: ffile.txt
Enter second file name: sfile.txt
Files are not identical
```

2) Write a Python program to copy one file to another. Copy one character at a time.

```
import os
s = input("Enter source file name with complete path: ")
d = input("Enter destination file name with complete path: ")

if os.path.exists(s):
   sf = open(s, 'r')
   df = open(d, 'w')
   ch = sf.read(1)
   while ch:
      df.write(ch)
      ch = sf.read(1)
   sf.close()
   df.close()
   print("Data is copied to  ' %s ' "%d)
else:
   print("Error: Couldn't open file")
```

Output:

```
Enter source file name with complete path: ffile.txt
Enter destination file name with complete path: nfile.txt
Data is copied to  ' nfile.txt '
```

3) Write a program to append a record to the student's file.

```
with open("Students.txt", "a") as f:
   NAME = input("Enter student name: ")
   ID = input("Enter student ID: ")
   CGPA = float(input("Enter student CGPA: "))
   f.write("{0}\t{1}\t{2}\n".format(NAME,ID,CGPA))
   print("Student's record is successfully added to 'Students.txt' file")
```

Output:

```
Enter student name: Aaradhya
Enter student ID: N182020
Enter student CGPA: 8.5
Student's record is successfully added to 'Students.txt' file
```

4) Write a program to read the record of a particular student.

```
with open("Students.txt") as f:
    NAME = input("Enter student name: ")
    sdata = f.readline()
    while sdata:
        if sdata.find(NAME) != -1:
            print("NAME\tID\tCGPA\n%s"% sdata)
            break
        sdata = f.readline()
    else:
        print("Student name not found")
```

Output:

```
Enter student name: Akash
NAME    ID          CGPA
Akash   N182035 9.1
```

```
Enter student name: Usha
Student name not found
```

5) Write a Python program to store records of an employee in employee file. The data must be stored in a binary file.

```
import pickle

n = int(input("Enter no.of employees: "))
f = open("Employee.dat", "wb")
while n>0:
    l = []
    l.append(input("Enter employee name: "))
    l.append(input("Enter employee ID: "))
    l.append(input("Enter employee designation: "))
    pickle.dump(l, f)
    n = n-1
f.close()
print("Employee data is stored in Employee.dat file")
```

Output:

```
Enter no.of employees: 3
Enter employee name: Rishi
Enter employee ID: 12938
Enter employee designation: Manager
Enter employee name: Tarun
Enter employee ID: 14009
Enter employee designation: Asst.Manager
Enter employee name: Arnav
Enter employee ID: 17100
Enter employee designation: Clerk
Employee data is stored in Employee.dat file
```

6) Write a Program to read the records stored in 'Employee.dat' file.

```
import pickle
f = open('Employee.dat', 'rb')

'''
pickle.load(object) raises EOFError when file reaches its end.
In order to avoid that we are finding the file size and comparing it
with file's current position after every pickle.load(object) function
call. So that as soon as file reaches EOF our program terminates.
'''

#Finding file size with the help of seek() and tell()
f.seek(0,2) #setting file pointer at the end of file
f_size = f.tell() #storing file current position in 'f_size'
f.seek(0) #setting file pointer at the beginning of the file
while True:
    print(pickle.load(f))
    if f_size == f.tell(): #if file reaches its end terminate the loop
        break
f.close()
```

Output:

```
['Rishi', '12938', 'Manager']
['Tarun', '14009', 'Asst.Manager']
['Arnav', '17100', 'Clerk']
```

7) Write a menu-driven program to read, insert, append a record in a binary file.

```
fname = 'SData.bin'

def readData():
    with open(fname,'rb') as f:
        r = f.readline()
        while r:
            print(r)
            r = f.readline()

def appendRecord():
    with open(fname, 'ab') as f:
        id = input("Enter student ID: ")
        name = input("Enter student name: ")
        cgpa = float(input("Enter student CGPA: "))
        data = "{0}\t{1}\t{2}\n".format(id,name,cgpa)
        f.write(data.encode())

def insertRecord():
    with open(fname, 'rb+') as f:
        data = f.readlines()
        id = input("Enter student ID: ")
        name = input("Enter student name: ")
        cgpa = float(input("Enter student CGPA: "))
        sid = input("Enter student ID where this record should be inserted")
        for i in range(len(data)):
            t = data[i].decode()
            if sid in t:
                r = "{0}\t{1}\t{2}\n".format(id,name,cgpa)
                data.insert(i,r.encode())
                break
        f.seek(0)
        f.writelines(data)

while True:
    print('''Menu:
                1. Read student data
                2. Insert student data
                3. Append student data
                Exit''')
    c = int(input("Enter your choice: (Ex: 1)"))
    if c == 1:
        readData()
    elif c == 2:
        insertRecord()
    elif c == 3:
        appendRecord()
    elif c == 4:
        break
    else:
        print("Invalid choice")
```

Output:

```
Menu:
                1. Read student data
                2. Insert student data
                3. Append student data
                Exit
Enter your choice: (Ex: 1)1
b'\n'
b'145N\tRishi\t8.1\n'
b'N123\tJaya\t8.9\n'
Menu:
                1. Read student data
                2. Insert student data
                3. Append student data
                Exit
Enter your choice: (Ex: 1)3
Enter student ID: 228N
Enter student name: Nidhi
Enter student CGPA: 7.8
```

8) Write a program to create a file that stores only integer values. Append the sum of these integers at the end of the file.

```
n = int(input("Enter number of integers: "))
f = open("integers.txt", "w+")
s = 0
while n>0:
    i = int(input("Enter number: "))
    s += i
    f.write(str(i))
    f.write('\n')
    n = n-1
f.write(str(s))
f.close()
print("Data is stored in 'integers.txt'")
```

Output:

```
Enter number of integers: 5
Enter number: 1
Enter number: 2
Enter number: 3
Enter number: 4
Enter number: 5
Data is stored in 'integers.txt'
```

9) Write a program to store student names in a file and display those in reverse order without reopening file.

```
f = open("Names.txt", "w+")
n = int(input("Enter number of names: "))
while n>0:
    f.write(input("Enter name: "))
    f.write('\n')
    n = n-1
f.seek(0) #setting file pointer at the beginning of the file
for line in reversed(f.readlines()):
    print(line, end='')
f.close()
```

Output:

```
Enter number of names: 5
Enter name: Sugandhi
Enter name: Samanvitha
Enter name: Sravanika
Enter name: Srijanya
Enter name: Sreshta
Sreshta
Srijanya
Sravanika
Samanvitha
Sugandhi
```

10) Write a Python program to count the upper case and lower case letters in a text file.

```
with open('data.txt') as f:
    l = 0
    u = 0
    c = f.read(1)
    while c:
        if c>='a' and c<='z':
            l +=1
        if c>='A' and c<='Z':
            u +=1
        c = f.read(1)
    print("No.of lower case letters: ", l)
    print("No.of upper case letters: ", u)
```

Output:

```
No.of lower case letters:  183
No.of upper case letters:  2
```

Unsolved Problems

1) Write a Python program to count the frequency of words in a file.
2) Write a Python program to merge contents of two text files and store it in a new text file.
3) Write a Python program to read a text file and perform following operations.
    a) Count of alphabets
    b) Count of Vowels
    c) Count of consonants
    d) Count of digits
    e) Count of special characters
4) Write a program to accept a filename from the user and display all the lines from the file which contain python comment character '#'.
5) Write a Python program to replace a line in the given text file.
6) Write a function that writes a structure to disk (binary file) and then reads it back and display on screen.
7) Write a function to create a text file containing following data
   Neither apple nor pine is in pineapple. Boxing rings are square.
   Writers write, but fingers don't fing. Overlook and oversee are opposites. A house can burn up as it burns down. An alarm goes off by going on.
    a) Read back the entire file content using read() or readlines () and display on screen.
    b) Append more text of your choice in the file and display the content of file with line numbers prefixed to line.
    c) Display last line of file.
    d) Display first line from 10th character onwards
    e) Read and display a line from the file. Ask user to provide the line number to be read.
8) Create a dictionary having decimal equivalent of roman numerals. Store it in a binary file. Write a function to convert roman number to decimal equivalent using the binary file data.
9) Write a program to read a file 'Story.txt' and create another file, storing an index of Story.txt telling which line of the file each word appears in. If word appears more than once, then index should show all the line numbers containing the word.
   Hint: Dictionary with key as word(s) can be used to solve this.
10) Write a program to read data from text file and store it in a binary file. Also read the data stored in the binary file and display it on the screen.

Descriptive Questions

1) Compare & contrast read(), readline() and readlines().
2) How is write() different from writelines()?
3) In how many ways can end of file be detected?
4) How many file modes can be used with the open() function to open a file? State the function of eachmode.
5) Explain about dump() and load().
6) Explain about seek() and tell() with suitable examples.
7) What are file attributes? Explain.
8) What happens if you do not close a file once it is used?

Module 2 – OS and Shutil modules- File handling

Python Directory and Files Management

The OS module in python provides functions for interacting with the operating system. OS module, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. It has many functions to interact with the file system. In order to use those functions in our program we need to **import os** module.

Following are some functions in OS module, which can be used in file handling:

The exists() method:

It is used to check whether a path exists or not. It returns true if the path exists and returns false otherwise.

Syntax:

os.path.exists(path)

Example:

```
import os
print(os.path.exists('RGUKT.txt'))
print(os.path.exists('data.txt'))
```

Output:

```
True
False
```

The isfile() method:

It returns true if the given path is an existing file's path and returns false otherwise.

Syntax: *os.path.isfile(path)*

Example:

```
import os
print(os.path.isfile('RGUKT.txt'))
print(os.path.isfile('Data'))
```

Output:

```
True
False
```

What is Directory in Python?

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and subdirectories. Python has the os module, which provides us with many useful methods to work with directories (and files as well).

Get Current DirectoryWe can get the present working directory using the getcwd() method.

This method returns the current working directory in the form of a string. We can also use the getcwdb() method to get it as bytes object.

Ex: *>>> import os*
>>> os.getcwd()
>>> os.getcwdb()

Changing Directory:

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements. It is safer to use escape sequence when using the backward slash.

>>> os.chdir('C:\\Python33')
>>> print(os.getcwd())
C:\Python33

List Directories and Files

All files and sub-directories inside a directory can be known using the listdir() method. This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

Making a New Directory:

We can make a new directory using the mkdir() method. This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

Ex: *>>> os.mkdir('test')*
>>> os.listdir()
['test']

Renaming a Directory or a File:

The rename() method can rename a directory or a file. The first argument is the old name and the new name must be supplies as the second argument.

Ex: *>>> os.listdir()*
['test']
>>> os.rename('test','new_one')
>>> os.listdir()
['new_one']

Removing Directory or File:

A file can be removed (deleted) using the remove() method. Similarly, the rmdir() method removes an empty directory.

Ex: *>>> os.listdir()*
['new_one', 'old.txt']
>>> os.remove('old.txt')
>>> os.listdir()
['new_one']
>>> os.rmdir('new_one')
>>> os.listdir()

Shutil module

The **shutil module** offers a number of high-level operations on files and collections of files. In particular, functions are provided which support file copying and removal. Though there are many functions in shutil module, we are going to discuss about only a few in our course. Let us have a look at those.

The copy() method:

It is used to copy file from one location to another location. It takes two string arguments src, dst. If *dst* specifies a directory, the file will be copied into *dst* using the base filename from *src*. Returns the path to the newly created file.

Syntax: *import shutil*
*shutil.copy(*src, dst*)*

Example:

```
>>> import shutil
>>> shutil.copy('D:\RGUKT PUC\PUC-2 2019\SEM2\p2sem2offline\index.html', 'G:\RGUKT\PUC\Sem2')
'G:\\RGUKT\\PUC\\Sem2\\index.html'
>>> import os
>>> os.listdir('G:\RGUKT\PUC\Sem2')
['index.html']
```

The rmtree() method:

It takes a directory's path as argument and deletes an entire directory tree.

Syntax: *import shutil*
*shutil.rmtree(*path*)*

Example:

```
>>> os.listdir("OnlineClasses")
['Guidelines.txt']
>>> os.rmdir("OnlineClasses")
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    os.rmdir("OnlineClasses")
OSError: [WinError 145] The directory is not empty: 'OnlineClasses'
>>> import shutil
>>> shutil.rmtree("OnlineClasses")
>>> os.listdir("OnlineClasses")
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    os.listdir("OnlineClasses")
FileNotFoundError: [WinError 3] The system cannot find the path specified: 'OnlineClasses'
```

The move() method:

Recursively move a file or directory (src) to another location (dst) and return the destination.

Syntax: *import shutil*
*shutil.move(*src, dst)

Example:

```
>>> import shutil
>>> from os import listdir
>>> listdir('D:\Temp')
[]
>>> listdir('G:\RGUKT')
['PUC']
>>> shutil.move('G:\RGUKT', 'D:\Temp')
'D:\\Temp\\RGUKT'
>>> listdir('D:\Temp')
['RGUKT']
```

Multiple Choice Questions

1) Which of the following can be used to create a directory?
   a) os.mkdir() b) os.makedir() c) os.mk_dir() d) os.make_dir()
2) Why do we use chdir() method?
   a) To change directory name b) To change the current working directory
   c) To change directory permissions d) none of the above
3) Which of the following functions is used to copy a file from one location to another?
   a) os.copy() b) b. shutil.copy() c) os.copycontents() d) shutil.copycontents()
4) What is the use of listdir() method?
   a) To get a list of directories b) To get a list of files

c) To get the list of files and directories d) none of the above

5) What is the use of rmtree() ?
   a) It is used to delete non-empty directory.
   b) It is used to delete an empty dictionary.
   c) Both a and b.
   d) None of the above
6) What is the use of getcwd()?
   a) It is used to get the current working directory b) It is used to get the correct working directory
   c) It is used to get the changed working directory d) None of the above
7) Which of the following modules has move() built-in function?
   a) Os b) shutil c) sys d) None of the above
8) What is the use of exists()?
   a) It is used to check whether a file exists or not b) It is used to check whether a directory exists or not
   c) It is used to check whether a file/directory exists or not d) None of the above
9) What is the use of remove ()?
   a) It is used to delete a file b) It is used to delete a directory
   c) It is used to delete a file/directory d) none of the above
10) What is the use of rename()?
   a) It is used to change the name of an existing file
   b) It is used to change the name of an existing directory
   c) Both a & b
   d) None of the above

Descriptive Questions

1) How do you check whether a directory exists or not? Explain with example.
2) What is the use of mkdir() and mkdirs() methods?
3) What is the function that we use to move a directory/file? Explain with example.
4) Is it possible to delete a non-empty directory? If yes, explain how?
5) How can one rename a file/directory? Explain.
6) What is the difference between getcwd() and getcwdb()?
7) What is the use of listdir()?
8) When do we need to use chdir()?

Module 3 – Exception Handling

Exception:

An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

Why we use Exceptions:

Exceptions are convenient in many ways for handling errors and special conditions in a program. When you think that you have a code which can produce an error then you can use exception handling.

Some example exceptions(errors):

IOError: If the file cannot be opened.

ImportError: If python cannot find the module

ValueError: Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

KeyboardInterrupt: Raised when the user hits the interrupt key (normally Control-C or Delete)

EOFError: Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

ZeroDivisionError: Raised when division or modulo by zero takes place for all numeric types.

NameError: Raised when an identifier is not found in the local or global namespace.

TypeError: Raised when an operation or function is attempted that is invalid for the specified data type.

Handling Exceptions:

We can handle exceptions in our program by using try block and except block. A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block. The syntax for try–except block can be given as,

Syntax:

try:

You do your operations here;

.....

except Exception:

If there is Exception1, then execute this block.

Example

```
num = int(input("Enter the numerator : "))
deno = int(input("Enter the denominator : "))
try:
  quo = num/deno
  print("QUOTIENT : ", quo)
except ZeroDivisionError:
  print("Denominator cannot be zero")
```

OUTPUT

```
Enter the numerator : 10
Enter the denominator : 0
Denominator cannot be zero
```

Note:

1) A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
2) After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.

try block with multiple except statements:

try:

You do your operations here;

.....

except Exception1:

If there is Exception1, then execute this block.

except Exception2:

If there is Exception2, then execute this block.

.....

```
try:
    num = int(input("Enter the number : "))
    print(num**2)
except (KeyboardInterrupt):
    print("You should have enterd a number..... Program Terminating...")
except (ValueError):
    print("Please check before you enter..... Program Terminating...")
print("Bye")
```

OUTPUT

```
Enter the number : abc
Please check before you enter..... Program Terminating...
Bye
```

The else clause:

The try ... except block can optionally have an else clause, which, when present, must follow all except blocks. The statement(s) in the else block is executed only if the try clause does not raise an exception.

Syntax:

try:

You do your operations here;

.....

except:

If there is any exception, then execute this block.

.....

else:

If there is no exception then execute this block.

Example:

```
try:
    file = open('File1.txt')
    str = file.readline()
    print(str)
except IOError:
    print("Error occurred during Input
..... Program Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

```
Hello
Program Terminating Successfully.....
```

```
try:
    file = open('File1.txt')
    str = f.readline()
    print(str)
except:
    print("Error occurred ..... Program
Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

```
Error occurred.....Program
Terminating...
```

Except block without exception name:

You can even specify an except block without mentioning any exception (i.e., except:). This type of except block if present should be at the end of the try-except block.

try:

You do your operations here;

.....

except exception1:
If there is any exception, then execute this block.
.....
except exception2:
If there is any exception, then execute this block.
.....
except:
If there is any exception, then execute this block.
.....

This kind of a try-except statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

The except Clause with Multiple Exceptions

We can also use the same except statement to handle multiple exceptions as follows

try:
You do your operations here;
except(Exception1[, Exception2[,...ExceptionN]]]):
If there is any exception from the given exception list, then execute this block.
.....
else:
If there is no exception then execute this block.

Raising an exception:

We can deliberately raise an exception using the raise keyword. The general syntax for the raise statement is, raise Exception-Name

Example:

```
try:
    num = 10
    print(num)
    raise ValueError
except:
    print("Exception occurred .... Program Terminating...")

OUTPUT
10
Exception occurred .... Program Terminating...
```

The try-finally clause:

We can use a finally along with a try block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this,

Syntax:

try:
You do your operations here;
Due to any exception, this may be skipped.
finally:
This would always be executed.

Example:

```
try:
    print("Raising Exception.....")
    raise ValueError
finally:
    print("Performing clean up in Finally......")

OUTPUT
Raising Exception.....
Performing clean up in Finally......
```

```
try:
    file = pen('File1.txt')
    str = f.readline()
    print(str)
except IOError:
    print("Error occured during Input ..... Program Terminating...")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error.... Program Terminating...")

OUTPUT
Unexpected error.... Program Terminating...
```

Programming Tip: When an exception occurs, it may have an associated value, also known as the exception's *argument*.

User Defined Exceptions:

Python has many built-in exceptions. If we want to create user-defined exception, we need to use Exception class. Programs may name their own exceptions by creating a new exception class.

```
class UnderAge(Exception):
  pass
def verify_age(age):
  if int(age) < 18:
    raise UnderAge
  else:
    print('Age: '+str(age))
verify_age(23)  # won't raise exception
verify_age(17)  # will raise exception
```

Example:

```
#creating custom exception
class AgeException (Exception): #AgeException - derived class, Exception - base class
   pass

try:
   name = input("Enter name: ")
   age = int(input("Enter your age: "))
   if age <= 18:
      #print("You are not eligible for voter registration")
      raise AgeException("You are not eligible for voter registration")
except AgeException as e:
   print(e)
```

Output:

```
Enter name: Aadya
Enter your age: 15
You are not eligible for voter registration
```

Multiple Choice Questions

1) How many except statements can a try-except block have?
   a) zero b) one c) more than one d) at least one
2) When will the else part of try-except-else be executed?
   a) always b) when an exception occurs
   c) when no exception occurs d) when an exception occurs in to except block
3) When is the finally block executed?
   a) when there is no exception b) when there is an exception
   c) only if some condition that has been specified is satisfied d) always
4) What is the output of the following code?

```
def foo():
   try:
      return 1
   finally:
      return 2
print(foo())
```

   a) 1 b) 2 c) 3 d) error, there is more than one return statement in a single try-finally block
5) What is the output of the following code?

```
def foo():
   try:
      print(1)
   finally:
      print(2)
foo()
```

   a) 1 2 b) 1 c) 2 d) none of the mentioned
6) What is the output of the following program?

```
data = 50
try:
   data = data/0
except ZeroDivisionError:
   print('Cannot divide by 0 ', end = ' ')
else:
   print('Division successful ', end = ' ')

try:
   data = data/5
except:
   print('Inside except block ', end = ' ')
else:
   print('RGUKT', end = ' ')
```

   a) Cannot divide by 0 RGUKT b) Cannot divide by 0
   c) Cannot divide by 0 Inside except block RGUKT d) Cannot divide by 0 Inside except block
7) What is the output of the following program?

```
data = 50
try:
   data = data/10
except ZeroDivisionError:
   print('Cannot divide by 0 ', end = ' ')
finally:
   print('PYTHON ', end = ' ')
else:
   print('Division successful ', end = ' ')
```

   a) Runtime error b) Cannot divide by 0 PYTHON c) PYTHON Division successful d) PYTHON

8) What is the output of the following code snippet?

```
valid = False
while not valid:
  try:
     n=int(input("Enter a number"))
     while n%2==0:
        print("Bye")
        valid = True
  except ValueError:
     print("Invalid")
```

a) PYTHON b) RGUKT c) RGUKT PYTHON d) Compilation error

9) What is the output of the following code?

```
value = [1, 2, 3, 4, 5]
try:
  value = value[5]/0
except (IndexError, ZeroDivisionError):
  print('PYTHON', end = ' ')
else:
  print('RGUKT ', end = ' ')
finally:
  print('IIIT ', end = ' ')
```

a) Compilation error b) Runtime error c) PYTHON RGUKT IIIT **d**) PYTHON IIIT

10) What is the output of the following code if the input is 6?

a) Bye (printed once) b) No output

c) Invalid (printed once) d) Bye (printed infinite number of times)

Solved problems

1) Write a python program to handle AttributeError.

```
try:
    import math
    math.hello()
except AttributeError as e:
    print("Error: " , e)
```

Output: **Error: module 'math' has no attribute 'hello'**

2) Write a Python program that throws EOFError and caught it with except.

```
try:
    while True:
        d = input()
        print(d)
except EOFError as e:
    print(e)#Error message will be displayed
```

Output:

```
D:\RGUKT PUC\PUC-2 2019\SEM2\Practise\ExceptionHandling>echo 'Hello' | EOFErrorDemo.py
'Hello'
EOF when reading a line
```

NOTE: Do not execute this program in Python Shell. Follow the below steps

step 1: Open command prompt/terminal

step 2: set your Python file path in it using "cd" command

step 3: enter the following in prompt and hit Enter key

echo <'some message'> | <Python file name>

echo - used to display text in command prompt or terminal

| - used to redirect output of one command as input to another command

example: echo 'hello' | EOFErrorDemo.py

3) Write a Python program that throws FileNotFoundError and handles it.

```
try:
    f = open("NewFile.txt") #this statement generates FileNotFoundError
    f.close()
except FileNotFoundError as m:
    print("Error: ", m)
```

Output: **Error: [Errno 2] No such file or directory: 'NewFile.txt'**

4) Write a Python program to handle ImportError.

```
try:
    import MyModule #this statement raises ImportError
except ImportError as e:
    print("Error: " , e)
```

Output: **Error: No module named 'MyModule'**

5) Write a Python program to handle IndexError.

```
try:
    l = [1,2,3,4]
    print(l[8])
except IndexError as m:
    print("Error: ", m)
```

Output:ERROR: List Index out of Range

6) Write a Python program to handle IOError.

```
try:
    f = open("Dat.txt")
    print(f.readline())
    print(f.readline())
    print(f.readline())
    f.close()
except IOError as e:
    print(e)
```

Output:

[Errno 2] No such file or directory: 'Dat.txt'

7) Write a python program that handles KeyboardInterrupt.

```
try:
    print("Press ctrl+c")
    x = input()
except KeyboardInterrupt:
    print("You stopped execution")
```

Output:

```
Press ctrl+c
You stopped execution
```

8) Write a python program that handles NameError.

```
try:
    print(n) #this statement generates NameError
except NameError as m:
    print("Error: ", m)
```

Output:

```
Error:  name 'n' is not defined
```

9) Write a Python program that handles TypeError.

```
try:
    print("2" + 3)
except TypeError as e:
    print("Error: ", e)
```

Output:

```
Error:  can only concatenate str (not "int") to str
```

10) Write a Python program that creates a custom exception which is to be raised when a given element is not found the in the list.

```
class NNFError(Exception):
    pass

try:
    l = [18, 23, 52, 83]
    n = int(input("Enter number: "))
    if n not in l:
        raise NNFError("Number not found in the given list")
except NNFError as e:
    print(e)
```

Output:

```
Enter number: 57
Number not found in the given list
```

Problem Sets

1) Write a program to handle simple runtime error.
2) Write a program to handle multiple errors with one except statement.
3) Write a program to take a number from keyboard, raise an exception if the input value is not a number.
4) Write a program to read a list from the keyboard. Ask the user to enter an index and print the index value as output. Raise an exception if the index is invalid index?
5) Write a function to find average of a list of numbers. Your function should be able to handle an empty list and also list containing string.
6) Write a program, to accept a date as day, month & year from user and raise appropriate error(s), if legal value(s) is not supplied. Display appropriate message till user inputs correct value(s).
7) Write a Python program that takes an employee's basic salary, TA, DA and computes gross salary and raises an custom exception if user inputs negative values.
8) Write a Python program for course registration. It should raise a custom exception NotEligibleError incase if student's CGPA is less than 6.0.
9) Write a Python program that handles multiple exceptions. Program should display custom message whenever an exception is caught.
10) Write a Python program to display week day when user enters a number. Your program should raise an error if use enters a value < 1 and value > 7.

Descriptive Questions

1) List the situation(s) in which the code may result in IOError?
2) When do we get TypeError?
3) Explain about finally with a suitable example.
4) How do you create a custom exception? Explain with a suitable example.
5) Why do we need exception handling? What happens if do not handle exceptions properly?
6) How do you throw an exception with a custom message?
7) Can we place an else block after finally block? Justify your answer.

Unit IV - Classes and Objects

Module - I: Introduction to Classes and Objects

Introduction

Object oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes) which are used to create individual instances of objects. Because OOP is a programming paradigm, there are many object-oriented programming languages including: C++, Java, and Python.

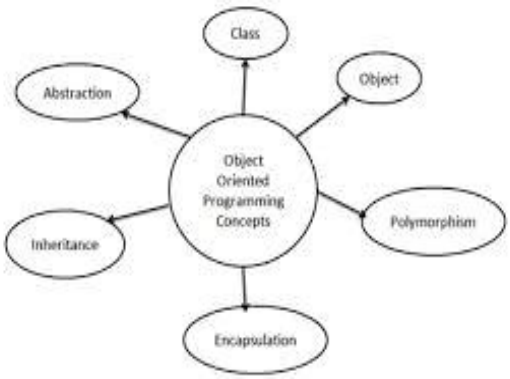
OOP makes code organized, reusable, and easy to maintain. OOP also prevents unwanted access to data, or exposing proprietary code through encapsulation and abstraction.

One of the features of Python programming language is, it supports Object Oriented Programming (OOP). To implement object oriented programming, we need classes and objects.

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods)

Following are the main concepts (features or building blocks) of object oriented programming.

- Classes
- Objects
- Data Abstraction
- Data Encapsulation
- Inheritance
- Polymorphism



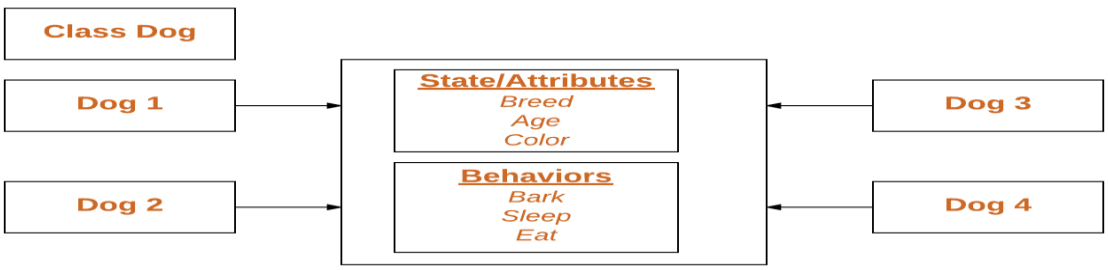
As part of the syllabus, you will learn the first two concepts in details in this unit and remaining you can learn in the advanced studies.

Class

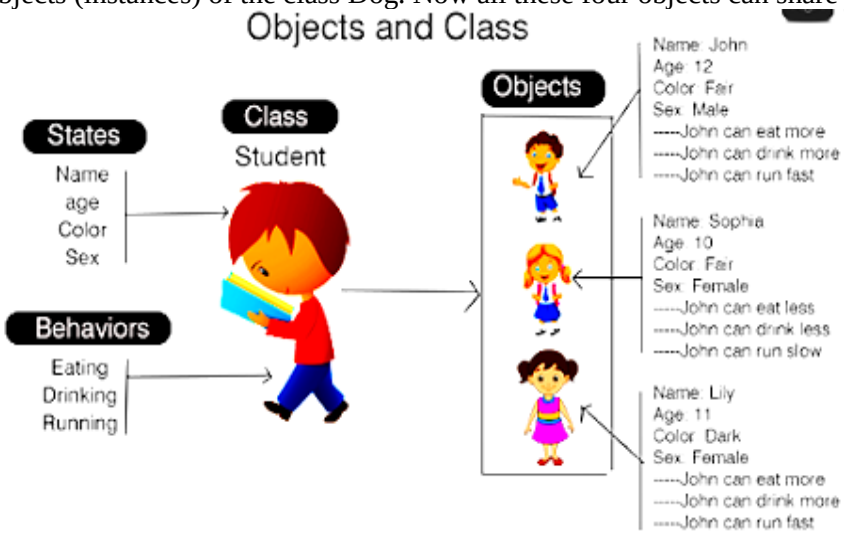
A class is a blueprint or a set of instructions to build a specific data type of object. Simply class is collection of behaviors and attributes. Behaviors is nothing but methods (functions) and attributes are variables. We can also say class is collection of methods and attributes.

Object

Object is an instance of a class. If we want to access a class, we have to create object for that class. Object is used to access the methods and attributes of a class. We can create any number of objects to a class. All class objects share the methods and attributes of that class.



In the above diagram, Dog is a class, which contains Breed, Age and Color as attributes; Bark, Sleep and Eat as methods. Now the class Dog can have any number of objects (instances). Let us consider Dog 1, Dog 2, Dog 3 and Dog 4 is the objects (instances) of the class Dog. Now all these four objects can share class attributes and methods.



Creating classes and objects

We can create a class in python by using the keyword class and followed by an indented block of code. This indented code can be a set of attributes and methods. These attributes and methods are also called as class members.

Syntax: A class needs to be instantiated if we want to use the class attributes in another class or method. Creating an object to a class is known as class instantiation. Following is the syntax to create an object to a class.

<Object_Name> = <Class_Name>(arguments)

```
class <Classname>:
    data1 = value1
    ...
    dataM = valueM

    def <function1>(self,arg1,...,argK):
        <block>
    ...
    def <functionN>(self,arg1,...,argK):
        <block>
```

Example 1

Output: Name is Ram
Location is India

Note

The **self** parameter is a reference to the current object of the class, and is used to access attributes and methods that belong to the class. We can use any name instead of **self**, but the first parameter should be the reference of the object

Example 2

```
class sample:
    name="Ram"
    location="India"
    def display(self):
        print("Name is ",self.name)
        print("Location is ",self.location)
s1=sample()
s2=sample()
s1.display()
s2.display()
```

Output:

Name is Ram
Location is India
Name is Ram
Location is India

In the above example we have created two objects for the class and called the display method with each object but we got same output. The reason is the attributes of the class are initialized with fixed values. If we are able to initialize different values for different objects then we will get different outputs. We can do this by using __init__() method.

Example 3:

```
class sample:
    name="Ram"
    location="India"
    def __init__(self,name,loc):
        self.name=name
        self.location=loc
    def display(self):
        print("Name is ",self.name)
        print("Location is ",self.location)
s1=sample("John","USA")
s2=sample("Ching","China")
s1.display()
s2.display()
```

Output

Name is John
Location is USA
Name is Ching
Location is China

Data Abstraction

Abstraction means hiding the complexity and only showing the essential features of the object. So in a way, Abstraction means hiding the real implementation and we, as a user, knowing only how to use it.

Real world example would be a vehicle which we drive without caring or knowing what all is going underneath. A TV set where we enjoy programs without knowing the inner details of how TV works.

Data Encapsulation

Data Encapsulation means wrapping(combining) data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variable.

Inheritance

Inheritance is a way of creating a new class for using details of an existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

Polymorphism

Polymorphism is a very important concept in programming. It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.

Access Modifiers (public, private and protected)

Object-oriented languages like C++, Java, Python supports access modifiers, which are used to restrict access to the attributes and methods of the class. Most programming languages has three forms of access modifiers, which are **Public, Protected** and **Private**

Python uses '_' symbol to determine the access control for a specific data member or a member function of a class. Access modifiers in Python have an important role to play in securing data from unauthorized access and in preventing it from being exploited.

A Class in Python has three types of access modifiers,

- Public Access Modifier
- Private Access Modifier
- Protected Access Modifier

Public Access Modifier

The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default. Any member can be accessed from outside the class environment. Accountant

Example:

```
class Employee:
    def __init__(self,name,desg,sal):
        self.name=name
        self.designation=desg
        self.salary=sal
    def display(self):
        print(self.name,"\t",self.designation,"\t",self.salary)
e1=Employee("Ram","Manager",60000)
e2=Employee("Ravi","Accountant",35000)
print(e1.name,e1.salary)
print(e2.name,e2.salary)
e1.display()
e2.display()
```

Output:

```
Ram 60000
Ravi 35000
Ram     Manager     60000
Ravi    Accountant  35000
```

In the above example, the attributes (name, designation, salary) and method (display()) are declared as public. So we can access those class members outside the class also.

Note: if attributes or methods are not declared with single or double underscore (_) then they are public members.

Private Access Modifier

The members of a class that are declared private are accessible within the class only, private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore '__' symbol before the data member of that class. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError:

Example

```
class Employee:
    __name=None
    __designation=None
    __salary=None
    def __init__(self,name,desg,sal):
        self.__name=name
        self.__designation=desg
        self.__salary=sal
    def __display(self):
        print(self.__name,"\t",self.__designation,"\t",self.__salary)
    def method_display(self):
        self.__display()
e1=Employee("Ram","Manager",60000)
e1.method_display()
#print(e1.__name)
#e1.__display()
#The above two statements will give error because
#__name and __display() are private members
```

Output:

```
Ram     Manager     60000
```

Protected Access Modifier

The members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '_' symbol before the data member of that class.

Example

```
class Employee:
    _name=None
    _designation=None
    _salary=None
    def __init__(self,name,desg,sal):
        self._name=name
        self._designation=desg
        self._salary=sal
    def _display(self):
        print(self._name,"\t",self._designation,"\t",self._salary)

class Clerk(Employee):
    def __init__(self,name,desg,sal):
        Employee.__init__(self,name,desg,sal)
    def display_salary(self):
        print("salary is: ",self._salary)
        self._display()
c=Clerk("Vas","Clerk",20000)
c.display_salary()
```

Output:
salary is: 20000
Vas Clerk 20000

Accessing class members

Naming	Type	Meaning
Name	Public	These attributes can be freely used inside or outside of a class definition
_name	Protected	Protected attributes should not be used outside of the class definition, unless inside of a subclass definition
__name	Private	This kind of attribute is inaccessible and invisible. It's neither possible to read nor to write those attributes, except inside of the class definition itself

We can access the members (attributes and methods) of a class by using the object of the class. The access depends on the access modifiers also. If class members are private or protected, we cannot easily access those members outside the class. Following are the syntaxes to access attributes and methods

Attribute accessing syntax: *Object_Name.Attribute_Name*

Method accessing syntax: *Object_Name.Method_Name()*

```
class Animal:
    name="Dog"
    color="White"
    def __init__(self,na,col):
        self.name=na
        self.color=col
    def display(self):
        print("Animal name: ",self.name)
        print("Animal color: ",self.color)
d=Animal("Duck","Brown")
c=Animal("Cow","Black")
print(Animal.name,Animal.color)
d.display()
c.display()
print(d.name,c.name)
```

Output:
Dog White
Animal name: Duck
Animal color: Brown
Animal name: Cow
Animal color: Black
Duck Cow

Multiple Choice Questions

1) Which of the following is/are OOP language?
   a) CPP b) Java c) Python d) All of the above
2) Which of the following statement is false about classes in python?
   a) Class is a blue print of an object
   b) Class contains attributes and methods
   c) Only one object is allowed to create for one class
   d) We can access members of class by using class name and object
3) Which of the following built-in method is used to initialize values to attributes of an object?
   a) __init__() b) __assign__() c) __create__() d) All of the above
4) Which of the following access modifier is declared with single underscore (_) symbol?
   a) Public b) protected c) private d) both b and c
5) Which type of access modifier members are not accessible outside the class?
   a) Private b) public c) protected d) Both and c
6) Which of the following statements is most accurate for the declaration x = Circle()?
   a) x contains an int value b) x is an object of circle class
   c) x is an attribute of circle class d) You can assign an int value to x.
7) _____ is used to create an object.
   a) Class b) constructor c) User-defined functions d) In-built functions
8) _____ represents an entity in the real world with its identity and behavior.
   A method b) An operator c) A class d) An object

9) What is the output of the following code?

```
class test:
    def __init__(self,a="Hello World"):
        self.a=a
    def display(self):
        print(self.a)
obj=test()
obj.display()
```

a) Hello World
b) None
c) Error
d) No output

10) What is the output of the following code?

```
class Sales:
    def __init__(self, id):
        self.id = id
        id = 100

val = Sales(123)
print (val.id)
```

a) Error
b) 100
c) 123
d) 0

Descriptive Questions

1) Explain class and object with an example?
2) Explain the structure of class in python?
3) Explain access modifiers in python?
4) What are the difference between private and protected access modifiers?
5) What are the features of OOP language?
6) How to access members of a class?

Solved Problems

1) Write a program to swap values of two objects by using classes and objects concepts?

```
class sample:
    def __init__(self,n):
        self.n=n
    def swap(self,second):
        t=self.n
        self.n=second.n
        second.n=t
n=int(input("Enter a value"))
s1=sample(n)
n=int(input("Enter a value"))
s2=sample(n)
print("Before swap")
print("Value of s1 is: ",s1.n)
print("Value of s2 is: ",s2.n)
s1.swap(s2)
print("After Swap")
print("Value of s1 is: ",s1.n)
print("Value of s2 is: ",s2.n)
```

2) Write a program to convert given integer to binary, octal and hexadecimal format?

```
class sample:
    def __init__(self,n):
        self.n=n
    def conversion(self):
        print("Decimal format: ",self.n)
        print("Binary format: ",bin(self.n))
        print("Octal format: ",oct(self.n))
        print("Hexadecimal format: ",hex(self.n))
n=int(input("Enter a value"))
s1=sample(n)
s1.conversion()
#s1=sample(8)
#s1.conversion()
```

3) Write a program to find GCD?

```
class sample:
    def __init__(self,n):
        self.n=n
    def gcd(self,second):
        a=self.n
        b=second.n
        r=a%b
        while(r!=0):
            a=b
            b=r
            r=a%b
        print("GCD of ",self.n," and ", second.n," is ",b)
n=int(input("Enter a value"))
s1=sample(n)
m=int(input("Enter a value"))
s2=sample(m)
s1.gcd(s2)
```

4) Write a program to find biggest of three numbers?

```
class sample:
      def __init__(self,n):
            self.n=n
      def big(first,second,third):
            a=first.n
            b=second.n
            c=third.n
            if a>=b and a>=c:
                  print("Big is ",a)
            elif b>=a and b>=c:
                  print("Big is ",b)
            else:
                  print("Big is",c)
n=int(input("Enter a value"))
s1=sample(n)
m=int(input("Enter a value"))
s2=sample(m)
p=int(input("Enter a value"))
s3=sample(p)
s1.big(s2,s3)
```

5) Write a program to find factors of the given number?

```
class sample:
      def __init__(self,n):
            self.n=n
      def factors(self):
            print("Factors are")
            for i in range(1,self.n+1):
                  if self.n%i==0:
                        print(i)
n=int(input("Enter a value"))
s1=sample(n)
s1.factors()
```

6) Create class Book and store n book details (title, cost, isbn) and print in a tabular format?

```
class Book:
    def __init__(self,name,cost,isbn):
        self.bname=name
        self.cost=cost
        self.isbn=isbn
    def display(bd):
        print("BookName\tCost\tISBN")
        for i in bd:
            print(i.bname,"\t",i.cost,"\t",i.isbn)

n=int(input("Enter number of books"))
Book_Details=[]
for i in range(n):
    name=input("Enter Book Name: ")
    cost=int(input("Enter Book Cost: "))
    isbn=input("Enter Book ISBN: ")
    b=Book(name,cost,isbn)
    Book_Details.append(b)
Book.display(Book_Details)
```

Unsolved Questions (Use classes and object concepts for every program)

1) Write a program to demonstrate private and public access modifiers?
2) Define a class Student and store N students details (sname, three subject marks) and find the sum. Display all student details.
3) Write a program to calculate simple interest? Use P,T,R as attributes and SI() method to calculate simple interest?
4) Create class Emp with attributes eid,ename,designation,sal,exp? Read N employee details and find bonus for every employee and display all employee details. If exp>10 years then bonus is 30% of salary, if exp>5 then bonus is 20% of salary, if exp<=5 then bonus is 10% of salary.
5) Write a program to find area and perimeter of the triangle?
6) Write a program to print the area of two rectangles having sides (4,5) and (5,8) respectively by creating a class named 'Rectangle' with a method named 'Area' which returns the area and length and breadth passed as parameters to its constructor.
7) Print the average of three numbers entered by user by creating a class named 'Average' having a method to calculate and print the average.
8) Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user.
9) Create class Movie with attributes name, director and rating. Create two methods setDetails() and getDetails() to read and display the movie details?
10) Create a class Triangle with three sides as attributes and print what type of triangle it is?

Module - II: Built-in Class Methods and Class Attributes

Introduction:

The class built-in methods in Python are the special methods which add "magic" to your class. These methods are also referred as magic methods. Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action. They're always surrounded by double underscores (e.g. __init__ or __lt__).

Construction and Initialization Methods

1. **__new__(cls)**

__new__() is the first method to get called in an object's instantiation. It takes the class, then any other arguments that it will pass along to __init__(). __new__() magic method is implicitly called before the __init__() method. The __new__() method returns a new object, which is then initialized by __init__().

2. **__init__(self)**

This method is called when an object is created from a class and it allows the class to initialize the attributes of the class. It is called as a constructor in object oriented terminology.

When you create an instance (object) x of a class A with the statement "x = A()", Python will do the necessary calls to __new__ and __init__.

Example:

```
class employee:
    def __new__(cls):
        print ("__new__ magic method is called")
        inst = object.__new__(cls)
        return inst
    def __init__(self):
        print ("__init__ magic method is called")
        self.name='RGUKT'
e=employee()
```

Output:

__new__ magic method is called
__init__ magic method is called

3. **__del__(self)**

The __del__ method is a special method of a class. It is also called the destructor method and it is called (invoked) when the instance (object) of the class is about to get destroyed.

```
class employee:
    def __init__(self):
        print("Object created")
    def __del__(self):
        print("Object deleted")
e=employee()
del e
```

Output:

Object created
Object deleted

Comparison Methods

- __eq__(self, other)
  Defines behavior for the equality operator, ==.
- __ne__(self, other)
  Defines behavior for the inequality operator, !=.
- __lt__(self, other)
  Defines behavior for the less-than operator, <.
- __gt__(self, other)
  Defines behavior for the greater-than operator, >.
- __le__(self, other)
  Defines behavior for the less-than-or-equal-to operator, <=.
- __ge__(self, other)
  Defines behavior for the greater-than-or-equal-to operator, >=.

```
>>> a=10
>>> b=20
>>> a.__eq__(b)
False
>>> a.__ne__(b)
True
>>> a.__lt__(b)
True
>>> a.__gt__(b)
False
>>> a.__ge__(b)
False
>>> a.__le__(b)
True
```

Unary Operators and Functions

- __pos__(self)
  Implements behavior for unary positive (e.g. +some_object)
- __neg__(self)
  Implements behavior for negation (e.g. -some_object)
- __abs__(self)
  Implements behavior for the built in abs() function.
- __invert__(self)
  Implements behavior for inversion using the ~ operator.
- __round__(self, n)
  Implements behavior for the built in round() function. n is the number of decimal places to round to.
- __floor__(self)
  Implements behavior for math.floor(), i.e., rounding down to the nearest integer.
- __ceil__(self)
  Implements behavior for math.ceil(), i.e., rounding up to the nearest integer.

```
>>> a=4.7
>>> a.__pos__()
4.7
>>> a.__neg__()
-4.7
>>> a.__abs__()
4.7
>>> a.__round__()
5
```

Arithmetic Operators

- __add__(self, other)
  Implements addition.
- __sub__(self, other)
  Implements subtraction.
- __mul__(self, other)
  Implements multiplication.
- __mod__(self, other)
  Implements modulo using the % operator.
- __pow__
  Implements behavior for exponents using the ** operator.
- __lshift__(self, other)
  Implements left bitwise shift using the << operator.
- __rshift__(self, other)
  Implements right bitwise shift using the >> operator.
- __and__(self, other)
  Implements bitwise and using the & operator.
- __or__(self, other)
  Implements bitwise or using the | operator.
- __xor__(self, other)
  Implements bitwise xor using the ^ operator.

```
>>> a=10
>>> b=20
>>> a.__add__(b)
30
>>> a.__mod__(3)
1
>>> a.__pow__(2)
100
>>> a.__lshift__(2)
40
>>> a.__and__(b)
0
```

Other Useful Methods

__str__() Method This method returns the string representation of the object. This method is called when print() or str() function is invoked on an object.

__repr__() Method This method returns the object representation. It could be any valid python expression such as tuple, dictionary, string etc.

```
>>> class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        print('inside str')
        return "Person: {}, Age: {}".format(self.name, self.age)
    def __repr__(self):
        print('inside repr')
        return " Person: {}, Age: {}".format(self.name, self.age)
```

Output

```
>>> per=Person("Ram",30)
>>> print(p)
inside str
Person: dsf, Age: 34
>>> p
inside repr
  Person: dsf, Age: 34
```

__len__() Method

__len__() is a special method that is internally called by len(s) method to return the length of an object. So, when we call len(s) method, s.__len__() is what actually happening behind the scenes to calculate the length.

Example1:

```
class Example:
        def __len__(self):
                return 6
obj = Example()
print(obj.__len__())
print(len(obj))
```

Output:

6
6

Example 2:

```
>>>l=[1,2,3]
>>>s="RGUKT"
>>>s.__len__()
5
>>>l.__len__()
3
```

Built-in Methods to access attributes of class

Attributes of a class can also be accessed using the following built-in methods. Following are the different methods.

- getattr() – This function is used to access the attribute of object.
  Syntax: getattr(Obj_Name, "Attribute_Name")

- hasattr() – This function is used to check if an attribute exist or not.
  Syntax: hasattr(Obj_Name, "Attribute_Name")
- setattr() – This function is used to set an attribute. If the attribute does not exist, then it would be created.
  Syntax: setattr(Obj_Name, "Attribute_Name",Value)
- delattr() – This function is used to delete an attribute. If you are accessing the attribute after deleting it raises error “class has no attribute”.
  Syntax: delattr(Class_name, Attribute_name)

Example:

```
# Python code for accessing attributes of class
class emp:
     name='Harsha'
     salary='25000'
     def show(self):
          print( self.name )
          print( self.salary )
e1 = emp()
# Use getattr instead of e1.name
print( getattr(e1,'name') )
# returns true if object has attribute
print (hasattr(e1,'salary'))
# sets an  attribute
setattr(e1,'height',152)
# returns the value of attribute name height
print( getattr(e1,'height') )
# delete the attribute
delattr(emp,'salary')
print (hasattr(e1,'salary'))
```

Output:
Harsha
True
152
False

Built-in Class Attributes Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute . Following are built-in class attributes.

- __dict__ : Dictionary containing the class's namespace.
- __doc__ : Class documentation string or none, if undefined.
- __name__ : Class name.
- __module__ : Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- __bases__ : A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Example

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount)
   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary)
print ("Employee.__doc__:", Employee.__doc__)
print ("Employee.__name__:", Employee.__name__)
print ("Employee.__module__:", Employee.__module__)
print ("Employee.__bases__:", Employee.__bases__)
print ("Employee.__dict__:", Employee.__dict__)
```

Output:
Employee.__doc__: Common base class for all employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: (<class 'object'>,)
Employee.__dict__: {'__module__': '__main__', '__doc__': 'Common base class for all employees', 'empCount': 0, '__init__': <function Employee.__init__ at 0x02CFBDB0>, 'displayCount': <function Employee.displayCount at 0x02CFBDF8>, 'displayEmployee': <function Employee.displayEmployee at 0x02CFBE40>, '__dict__': <attribute '__dict__' of 'Employee' objects>, '__weakref__': <attribute '__weakref__' of 'Employee' objects>}

Class variables and Object(Instance) variables

Object-oriented programming allows for variables to be used at the class level or the instance(object) level. At the class level, variables are referred to as class variables, whereas variables at the instance level are called instance(object) variables.

Class Variables

Class variables are defined within the class construction. Because they are owned by the class itself, class variables are shared by all instances of the class. They therefore will generally have the same value for every instance unless you are using the class variable to initialize a variable.

Class variables typically placed right below the class header and before the constructor method and other methods. A class variable can be declared as follows

```
class Shark:
    animal_type = "fish"
```

Here, the variable animal_type is assigned the value "fish". We can create an instance of the Shark class and print the variable by using dot notation.

```
class Shark:
        animal_type = "fish"
new_shark = Shark()
print(new_shark.animal_type
```

Like the above, we can add any number of class variables.

```
class Shark:
    animal_type = "fish"
    location = "ocean"
    followers = 5
new_shark1 = Shark()
new_shark2 = Shark()
print(new_shark1.animal_type,new_shark1.location)
print(new_shark2.animal_type,new_shark2.location)
```

Output:

fish ocean
fish ocean

animal_type and location are class variables so that the output is same for both objects.

Instance (Object) Variables

Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different. Unlike class variables, instance variables are defined within methods. In the Shark class example below, name and age are instance variables.

```
class Shark:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

When we create a Shark object, we will have to define these variables, which are passed as parameters within the constructor method or another method.

```
class Shark:
    def __init__(self, name, age):
        self.name = name
        self.age = age
new_shark1 = Shark("Sammy", 5)
new_shark2 = Shark("Bunny", 7)
print(new_shark1.name,new_shark1.age)
print(new_shark2.name,new_shark2.age)
```

Output:

Sammy 5
Bunny 7

Working with Class Variables and Instance(Object) Variables together

Python is an object-oriented programming language that allows programmers to define objects which can contain data. The object structure makes it easy to reduce repetition in your code; making it more efficient and easier to read. There are two types of variables you may encounter—instance variables and class variables.

Class Variables:

Class variables are declared when a class is being constructed. This means that the class owns the variable, it is contained within the class, so all instances of the class will be able to access that variable. Class variables are shared by all instances that access the class.

```
class Laptop:
    model="Acer"
print(Laptop.model)
l1=Laptop()
print(l1.model)
l2=Laptop()
l2.model="Dell"
print(l2.model)
print(Laptop.model)
```

output:

Acer
Acer
Dell
Acer

Instance Variables

Instance variables, unlike class variables, are owned by an instance of a class. This means that the value of each instance variable can be different, whereas, in a class variable, the variable can only have one value that you assign when you declare the variable. Instance variables are declared inside a class method.

```
class Laptop:
    def __init__(self,mod):
        self.model=mod
#print(Laptop.model),It is Error
l1=Laptop("Acer")
l2=Laptop("Dell")
print(l1.model)
print(l2.model)
```

output:

Acer
Dell

Class variables and instance variables will often be utilized at the same time, so let's look at an example of this using the Shark class we created.

```
class Shark:
    # Class variables
    type = "fish"
    loc = "ocean"
    # Constructor method with instance variables name and age
    def __init__(self, name, age):
        self.name = name
        self.age = age
new_shark1=Shark("sammy",5)
new_shark2=Shark("Banny",7)
print(new_shark1.name,new_shark1.age,new_shark1.type,new_shark1.loc)
print(new_shark2.name,new_shark2.age,new_shark2.type,new_shark2.loc)
```

Output:

sammy 5 fish ocean
Banny 7 fish ocean

Multiple Choice Questions

1) Which of the following built-in method will automatically invoked when we delete an object?
   a) __init__() b) __new__() c)__del__() d) Both a and b
2) Which of the following is not a built-in comparison method?
   a) __ne__() b) __le__() c) __gt__() d) None of the above
3) What is the output of the following statement?
   >>>k=-5.9 >>>k.__abs__()
   a) 5.9 b) 6 c) -6 d) -5
4) What is the output of the following code?
   >>>p=10 >>>q=28 >>>p.__xor__(q)
   a) 30 b) 22 c) 8 d) 38
5) What is the output of the following code?
   >>> l=[[4,6,[6,75],5,2],"abcd",3.4] >>> l[0].__len__()
   a) 3 b) 4 c) 5 d) 6
6) Which of the following built-in method is used add new attribute to the class?
   a) __new__() b) __setattr__() c) setattr() d) hasattr()
7) Which of the following is not a built-in class attribute?
   a) __dict__ b) __doc__ c) __bases__ d) None of the above
8) Which variable value is common to all objects?
   a) class variable b) instance variable c) Both a and b d) None of the above
9) What is the output of the following code?

```
class sample:
        name="Hello"
        def __init__(self,name):
            self.name=name
s=sample("Hai")
print(s.name)
```

   a) Hai b) Hello c) Hailo d) Error
10) Which of the following class built-in attribute is used to print class documentation string?
   a) __name__ b) __doc__ c) __bases__ d) __title__

Descriptive Questions

1) Explain construction and initialization methods with syntax?
2) Explain built-in unary methods with syntax?
3) Explain __str__ and __len__ method with example?
4) What are the different built-in methods to access attributes of a class?
5) Explain built-in class attributes?
6) What is the difference between class variable and instance variable?

Solved Problems

1) Write a program to add new attribute to the existing class object?

```
class car:
    Model="Aura"
    Company="Hyundai"
c=car()
print(c.Model)
print(c.Company)
#print(c.color), it is error
setattr(c,"Color","Choclate Brown")
print(c.Color)
```

2) Define a class with a private method check_prime() and check the given number is prime or not?

```
class sample:
    def __init__(self,no):
        self.no=no
        self.__check_prime()
    def __check_prime(self):
        f=0
        for i in range(1,self.no+1):
            if self.no%i==0:
                f+=1
        if(f==2):
            print("It is prime")
        else:
            print("It is not prime")
n=int(input("Enter a number"))
s=sample(n)
```

Unsolved Problems (Use classes and object concepts)

1) Write program to find the length of list?
2) Create a class with two and attributes and one method? Add new attribute to the existing class?
3) Create a class with a method compare() and compare the values of two objects without using built-in methods?
4) Write a program to demonstrate built-in numerical methods?
5) Write a program to convert given decimal value into binary, octal and hexadecimal format?
6) Write a program to check whether class contains user given attribute or not?

Module - III: Types of Methods and accessing methods from different classes

Introduction

A class contains methods and attributes, methods are used to perform actions. Methods can be accessed in the same class as well as different classes. A method is declared just like a function with def keyword and some parameters. In this module you will learn different types of methods and how to access methods from outside the class.

Instance, Class and Static Methods

There are three types of methods in Python: instance methods, static methods, and class methods.

Instance Methods

They are most widely used methods. Instance method receives the instance of the class as the first argument, which by convention is called **self**, and points to the instance of a class. However it can take any number of arguments. Using the self parameter, we can access the other attributes and methods on the same object.

```
class Student:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def avg(self):
        return (self.a + self.b) / 2
s1 = Student(10, 20)
print( s1.avg() )
```

Output:

15.0

In the above program, a and b are instance variables and these get initialized when we create an object for the Student class. If we want to call avg() function which is an instance method.

Class Methods

A class method accepts the class as an argument to it which by convention is called **cls**. It take the **cls** parameter, which points to the class instead of the object of it. It is declared with the @classmethod decorator. Class methods are bound to the class and not to the object of the class. A class method can be called either using the class or using an instance of that class.

Class methods don't need self as an argument, but they do need a parameter called cls. This stands for class, and like self, gets automatically passed in by Python.

```
class Student:
    name="Hello"
    def __init__(self, a, b):
        self.a = a
        self.b = b
    @classmethod
    def info(cls):
        return cls.name
print(Student.info())
```

Output:

Hello

In the above example, name is a class variable. If we want to create a class method we must use @classmethod decorator and cls as a parameter for that function.

Static Methods

Static method is marked with a @staticmethod decorator to flag it as static. It does not receive an implicit first argument (neither self nor cls). A static method can be called without an object for that class, using the class name directly.

```
class Student:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    @staticmethod
    def info():
        print("This static method")
Student.info()
```

Output:

This static method

Calling a method from another method in the same class

The following example demonstrates how to call a method from another method in the same class.

```
class Main:
    # constructor of Main class
    def __init__(self):
        # Initialization of the Strings
        self.String1 ="Hello"
        self.String2 ="World"
    def Function1(self):
        # calling Function2 Method
        self.Function2()
        print("Function1 : ", self.String2)
        return
    def Function2(self):
        print("Function2 : ", self.String1)
        return
# Instance of Class Main
Object = Main()
# Calling Function1
Object.Function1()
```

Output:

Function2 : Hello
Function1 : World

Calling one class method from another class

The following example demonstrates how to call one class method from another class.

```
class Parent:
    # constructor of Parent class
    def __init__(self):
        # Initialization of the Strings
        self.String1 ="Hello"
        self.String2 ="World"
    def Function2(self):
        print("Function2 : ", self.String1)
# Child class is inheriting from Parent class
class Child(Parent):
    def Function1(self):
        # calling Function2 Method in parent class
        self.Function2()
        print("Function1 : ", self.String2)
Object1 = Parent()
Object2 = Child()
# Calling Function1 using Child class instance
Object2.Function1()
```

Output:

Function2 : Hello
Function1 : World

```
class parent:
    def sum(self,a,b):
        return a+b
class your_class:
    def Fun(self,a,b):
        self.a=a
        self.b=b
        x=parent.sum(self,a,b)
        print("sum=",x)
#class object of child class
ob=your_class()
x=int(input("enter 1st no."))
y=int(input("enter 2nd no."))
#fuction call of your class
ob.Fun(x,y)
```

Output:

enter 1st no.10
enter 2nd no.8
sum= 18

Garbage Collection

Python deletes unwanted objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically frees and reclaims blocks of memory that no longer are in use is called Garbage Collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. There are two aspects to memory management and garbage collection.

- Reference counting
- Generational garbage collection

Reference counting:

An object's reference count changes as the number of aliases that point to it changes. An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with del, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

You can use the sys module from the Python standard library to check reference counts for a particular object. There are a few ways to increase the reference count for an object, such as

- Assigning an object to a variable.
- Adding an object to a data structure, such as appending to a list or adding as a property on a class instance.
- Passing the object as an argument to a function.

Let's see the following example, how the reference count increases and decreases for an object.

```
a = 40      # Create object <40>
b = a       # Increase ref. count  of <40>
c = [b]     # Increase ref. count  of <40>
del a       # Decrease ref. count  of <40>
b = 100     # Decrease ref. count  of <40>
c[0] = -1   # Decrease ref. count  of <40>
```

```
>>> import sys
>>> a=40
>>> print("Reference Count: ",sys.getrefcount(a))
Reference Count:  26
>>> b=a
>>> print("Reference Count: ",sys.getrefcount(a))
Reference Count:  27
>>> c=[b]
>>> print("Reference Count: ",sys.getrefcount(a))
Reference Count:  28
>>> b=100
>>> print("Reference Count: ",sys.getrefcount(a))
Reference Count:  27
>>> c[0]=-1
>>> print("Reference Count: ",sys.getrefcount(a))
Reference Count:  26
```

Generational Garbage Collection

In addition to the reference counting strategy, Python also uses a method called a generational garbage collector. There are two key concepts to understand with the generational garbage collector. The first concept is that of a generation and second concept is threshold.

The garbage collector is keeping track of all objects in memory. A new object starts its life in the first generation of the garbage collector. If Python executes a garbage collection process on a generation and an object survives, it moves up into a second, older generation. The Python garbage collector has three generations in total, and an object moves into an older generation whenever it survives a garbage collection process on its current generation.

For each generation, the garbage collector module has a threshold number of objects. If the number of objects exceeds that threshold, the garbage collector will trigger a collection process. For any objects that survive that process, they're moved into an older generation.

Unlike the reference counting mechanism, you may change the behavior of the generational garbage collector in your Python program. This includes changing the thresholds for triggering a garbage collection process in your code, manually triggering a garbage collection process, or disabling the garbage collection process altogether.

Let's see how you can use the gc module to check garbage collection statistics or change the behavior of the garbage collector. Import the gc module then check the configured thresholds of your garbage collector with the get_threshold() method.

```
>>> import gc
>>> gc.get_threshold()
(700, 10, 10)
```

By default, Python has a threshold of 700 for the youngest generation and 10 for each of the two older generations. You can check the number of objects in each of your generations with the get_count() method.

```
>>> import gc
>>> gc.get_count()
(596, 2, 1)
```

In this example, we have 596 objects in our youngest generation, two objects in the next generation, and one object in the oldest generation. As you can see, Python creates a number of objects by default before you even start executing your program. You can trigger a manual garbage collection process by using the gc.collect() method.

```
>>> gc.get_count()
(595, 2, 1)
>>> gc.collect()
57
>>> gc.get_count()
(18, 0, 0)
```

Running a garbage collection process cleans up a huge amount of objects 577 in the first generation and three more in the older generations. You can alter the thresholds for triggering garbage collection by using the set_threshold() method in the gc module.

```
>>> import gc
>>> gc.get_threshold()
(700, 10, 10)
>>> gc.set_threshold(1000, 15, 15)
>>> gc.get_threshold()  (1000, 15, 15)
```

Multiple Choice Questions

1) Which of the following is used as first parameter in the instance method?
   a) self(reference of the object) b) any integer value c) cls d) None of the above
2) Which of the following decorator is used to declare a class method?
   a) @decor b) @classdecor c) @classmethod d) @clsdecor
3) Which of the following type of method does not receive any parameter?
   a) class b) static c) instance d) None of the above
4) In which of the following situations, reference count value will be increased?
   a) Assigning an object to a variable b) Adding an object to a data structure
   c) Passing an object as an argument to a function d) All of the above
5) What is the output of the following code? Assume that the initial reference count of the object is 20?

```
import sys
a=5
b=a
c=b
print(sys.getrefcount(a))
```

   a) 20
   b) 21
   c) 22
   d) Error
6) What are the default threshold values of get_threshold() method in generalization garbage collection?
   a) (600,10,20) b) (700,10,10) c) (700,10,20) d) (600,10,10)
7) Which of the following method is used to change the default values of thresholds?
   a) gc.threshold() b) gc.putthreshold() c) gc.set_threshold() d) gc.get_threshold()
8) What is the output of the following code?

```
class one:
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def display(self):
        self.a=self.a+self.b
        self.b=self.a-self.b
        self.a=self.a-self.b
        print(self.a,self.b)
class two:
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def display(self):
        one.display(self)
        print(self.a,self.b)
obj=two(10,20)
obj.display()
```

   a) 20 10 and 10 20
   b) 10 20 and 10 20
   c) 20 10 and 20 10
   d) 10 20 and 20 10
9) What is the output of the following code?

```
class sample:
    a=100
    b=200
    def __init__(self,a,b):
        self.a=a
        self.b=b
    @classmethod
    def sum(cls):
        print(cls.a+cls.b)
obj=sample(50,60)
obj.sum()
```

   a) 110
   b) 300
   c) Error
   d) None of the above
10) Technique of freeing the memory of objects when they are no longer useful is known as?
   a) Memory management b) Free space management c) Garbage collection d) none of the above.

Descriptive Questions

1) What are the differences between class method and static method?
2) Explain reference counting?
3) Explain generalization garbage collection?
4) Explain instance method?
5) Explain about getrefcount() method?
6) What is garbage collector?

Solved problems

1) Write a program to demonstrate how to create nested methods in a class?

```
class NestedMethods:
      # define class variable
      x = 10
      def outer_method(self, c, d):
            # define inner method
            def operations(a, b):
                  print("Sum of numbers", a+b)
                  print("Product of numbers", a * b)
                  # access instance variable
                  print("Instance variable is", self.x)
            # call inner method
            operations(10, 20)

# create class instance
obj = NestedMethods()
# call outer method
o = obj.outer_method(2,3)
```

2) Create a class person with name and db as attributes. Create a new class DOB with dd,mm,yy as attributes inside person class. Create inner class object with help of outer class object and access inner class membe

```
class person:
    def __init__(self):
        self.name = 'AKASH'
        self.db = self.Dob()

    def display(self):
        print('NAME = ', self.name)
    # this is inner class
    class Dob:
        def __init__(self):
            self.dd = 10
            self.mm = 3
            self.yy = 2000
        def display(self):
            print('DOB = {}/{}/{}'.format(self.dd, self.mm, self.yy))
# creating person class object
p = person()
p.display()
# create inner class object
x = p.db
x.display()
```

Unsolved Problems

1) Write a program to take time in seconds as input and display time HH:MM:SS format?
2) Write a program that defines a shape class with a constructor that gives value to width and height. Define two classes' triangle and rectangle that calculate the area of the shape area (). Define two variables a triangle and a rectangle and then call the area () function in this two variables.
3) Create a class called Numbers, which has a single class attribute called MULTIPLIER, and a constructor which takes the parameters x and y (these should all be numbers).
    a) Write a method called add which returns the sum of the attributes x and y.
    b) Write a class method called multiply, which takes a single number parameter a' and returns the product of a' and MULTIPLIER.
    c) Write a static method called subtract, which takes two number parameters, b and c, and returns b - c.
    d) Write a method called value which returns a tuple containing the values of x and y. Make this method into a property, and write a setter and a deleter for manipulating the values of x and y.
4) Create a Python class called BankAccount which represents a bank account, having as attributes: accountNumber (numeric type), name (name of the account owner as string type), and balance.
    a) Create a constructor with parameters: accountNumber, name, balance.
    b) Create a Deposit() method which manages the deposit actions.
    c) Create a Withdrawal() method which manages withdrawals actions.
    d) Create an bankFees() method to apply the bank fees with a percentage of 5% of the balance account.
    e) Create a display() method to display account details.
5) Create a Coputation class to perform various calculations on integers numbers.
    a) Create a method called Factorial() which allows to calculate the factorial of an integer. Test the method by instantiating the class.
    b) Create a method called Sum() allowing to calculate the sum of the first n integers 1 + 2 + 3 + .. + n. Test this method.
    c) Create a method called testPrim() in the Calculation class to test the primality of a given integer. Test this method.
    d) Create sa method called testPrims() allowing to test if two numbers are prime between them.
    e) Create a tableMult() method which creates and displays the multiplication table of a given integer.

Unit-V&VI
Mini Project (Google search engine –web crawler)
Unit V - Crawling and indexing
Module 0 - Introduction to (Google) search engine

Every time you search, there are thousands, sometimes millions, of WebPages with helpful information. How Google figures out which results to show starts long before you even type, and is guided by a commitment to you to provide the best information.

Crawling and indexing

How Search organizes information

Before you search, **web crawlers** gather information from across hundreds of billions of WebPages and organize it in the Search index.

The fundamentals of Search

The crawling process begins with a list of web addresses from past crawls and sitemaps (A *sitemap* is a file where you provide information about the pages, videos, and other files on your site, and the relationships between them. Search engines like Google read this file to more intelligently crawl your site. A sitemap tells Google which pages and files you think are important in your site, and also provides valuable information about these files: for example, for pages, when the page was last updated, how often the page is changed and any alternate language versions of a page.) provided by website owners. As our crawlers visit these websites, they use links on those sites to discover other pages. The software pays special attention to new sites, changes to existing sites and dead links. Computer programs determine which sites to crawl, how often and how many pages to fetch from each site.

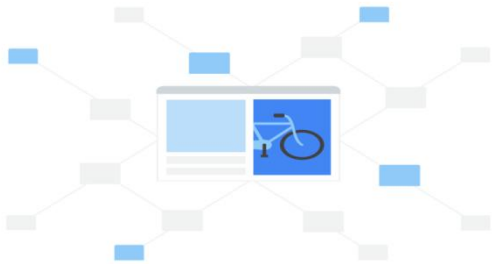
Finding information by crawling

The web is like an ever-growing library with billions of books and no central filing system. We use software known as **web crawlers** to discover publicly available web pages. Crawlers look at webpages and follow links on those pages, much like you would if you were browsing content on the web. They go from link to link and bring data about those webpages back to Google's servers.

Organizing information by indexing

When crawlers find a webpage, our systems render the content of the page, just as a browser does. We take note of key signals — from keywords to website freshness — and we keep track of it all in the Search index.

The Google Search index contains hundreds of billions of WebPages and is well over 100,000,000 gigabytes in size.



It's like the index in the back of a book — with an entry for every word seen on every webpage we index. When we index a webpage, we add it to the entries for all of the words it contains.

Instantly matching your search

In a fraction of a second, Google's Search algorithms sort through hundreds of billions of webpages in our Search index to find the most relevant, useful results for what you're looking for.

How Search algorithms work

With the amount of information available on the web, finding what you need would be nearly impossible without some help sorting through it. Google ranking systems are designed to do just that: sort through hundreds of billions of web pages in our Search index to find the most relevant, useful results in a fraction of a second, and present them in a way that helps you find what you're looking for.

These ranking systems are made up of not one, but a whole series of algorithms. To give you the most useful information, Search algorithms look at many factors, including the words of your query, relevance and usability of

pages, expertise of sources, and your location and settings. The weight applied to each factor varies depending on the nature of your query—for example, the freshness of the content plays a bigger role in answering queries about current news topics than it does about dictionary definitions.

Presenting results in helpful ways

To help you find what you're looking for quickly, Google provides results in many useful formats. Whether presented as a map with directions, images, videos or stories, we're constantly evolving with new ways to present information.

Larry Page once described the perfect search engine as understanding exactly what you mean and giving you back exactly what you want. Over time, our testing has consistently showed that people want quick answers to their queries. We have made a lot of progress on delivering you the most relevant answers, faster and in formats that are most helpful to the type of information you are seeking.

If you are searching for the weather, you most likely want the weather forecast on the results page, not just links to weather sites. Or directions: if your query is "Directions to San Francisco airport", you want a map with directions, not just links to other sites. This is especially important on mobile devices where bandwidth is limited and clicking between sites can be slow.

Dear Students,

Here we are implementing our own search engine similar like Google search engine.

Module 1 - Web Crawler

Introducing the Web Crawler

A web crawler is a program that collects content from the web. A web crawler finds web pages by starting from a seed page and following links to find other pages, and following links from the other pages it finds, and continuing to follow links until it has found many web pages.

Here is the process that a web crawler follows:

Start from one preselected page. We call the starting page the "seed" page.

Extract all the links on that page.

Follow each of those links to find new pages.

Extract all the links from all of the new pages found.

Follow each of those links to find new pages.

Extract all the links from all of the new pages found.

This keeps going as long as there are new pages to find, or until it is stopped.

Extracting Links

A web page is really just a long string of characters. Your browser renders the web page in a way that looks more attractive than just the string of characters. You can view the string of characters for any web page in your browser. In Chrome and Firefox, right-click anywhere on the pages that is not a link and select "View Page Source". For Internet Explorer, right-click and select "View Source".

Select View Page Source from the menu. The raw string for the web page pops up in a new window:

For our web crawler, the important thing is to find the links to other web pages in the page. We can find those links by looking for the anchor tags that match this structure:

<a href="<url>">

For example, here is a link to the News/Blag page:

<ahref="http://blag.xkcd.com"/>

To build our crawler, for each web page we want to find all the link target URLs on the page. We want to keep track of them and follow them to find more content on the web. we will do the first step which is to extract the first target URL from the page.

We will see how to keep track of them to be able to crawl the target pages. For now, our goal is to take the text from a web request and find the first link target in that text. We can do this by finding the anchor tag, <a href=", and then extract from that tag the URL that is found between the double quotes. We will assume that with the page's contents in a variable, page.

Example of Extracting Links

Write Python code that initializes the variable start_link to be the value of the position where the first 'http://udacity.com'.

page = '<contents of a web page>'
start_link = page.find('<a href=')
[your code here]"
printurl
http://udacity.com
Answer :-start_link = page.find('<a href=')

Example of print the URL

To extract the URL, we need to find the starting and ending double quotes and then extract the subsequence of characters between them.

Step 1: Initialize the variable start_quote, which is where the double quote that starts the URL is:

start_quote = page.find('"', start_link)

Note that it is important to pass in start_link as the second input to find, so we only find a string inside the anchor tag. Otherwise, there might be other strings on the page that are not link targets that would be found.

Step 2: Find the end quote:

end_quote = page.find ('"', start_quote + 1)

We need to pass in start_quote + 1 to find the next " after the start quote.

Step 3: Select the subsequence of characters from page between the start quote and end_quote, not including the quotes, and assign that to the variable url:

url = page[start_quote +1 : end_quote]

Final Code:

start_link = page.find('<a href=')
start_quote=page.find(' " ',start_link)
end_quote=page.find('"',start_quote+1)
url=page[start_quote+1:end_quote]

Remaining String or page

page=page[end_quote:]

In module 1, you wrote a program to extract the first link from a web page. The next step towards building your search engine is to extract all of the links from a web page. In order to write a program to extract all of the links, you need to know these two key concepts:

Procedures - a way to package code so it can be reused with different inputs.

Control - a way to have the computer execute different instructions depending on the data (instead of just executing instructions one after the other).

Recall this code from then end of module 1:

```
s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a>'
start_link=s.find('<a href=')
start_quote=s.find('"',start_link)
end_quote=s.find('"',start_quote+1)
url=s[start_quote+1:end_quote]
print(url)
```

This finds and prints the first link on the page. To keep going, we could update the value of page to be the characters from the **end_quote**, and repeat the same code again:

```
s=s[end_quote+1:]
s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a>'
start_link=s.find('<a href=')
start_quote=s.find('"',start_link)
end_quote=s.find('"',start_quote+1)
url=s[start_quote+1:end_quote]
print(url)
s=s[end_quote+1:]
s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a>'
start_link=s.find('<a href=')
start_quote=s.find('"',start_link)
end_quote=s.find('"',start_quote+1)
url=s[start_quote+1:end_quote]
print(url)
```

This code will print out the next two links on the web page. Clearly, this is tedious work. The reason for computers is to avoid having to do tedious, mechanical work! In addition to being tedious, repeating the same code over and over again like this will not work well because some pages only have a few links while other pages have more links than the number of repetitions.

Motivating Procedures or Functions

Procedural abstraction is a way to write code once that works on any number of different data values. By turning our code into a procedure, we can use that code over and over again with different inputs to get different behaviours.

Introducing Procedures

A procedure takes in inputs, does some processing, and produces outputs.

For example,

The + operator is a procedure where the inputs are two numbers and the output is the sum of those two numbers. The + operator looks a little different from the procedures we will define since it is built-in to Python with a special operator syntax. In this unit you will learn how to write and use your own procedures.

Here is the Python grammar for writing a procedure:

```
def name_of_the_Procedure(parameters):
    "block or body of the procedure"
```

The keyword **def** is short for "define".

<***name***> is the name of a procedure. Just like the name of a variable, it can be any string that starts with a letter and followed by letters, number and underscores.

<***parameters***> are the inputs to the procedure. A parameter is a list of zero or more names separated by commas: <***name***>, <***name***>,... Remember that when you name your parameters, it is more beneficial to use descriptive names that remind you of what they mean. Procedures can have any number of inputs. If there are no inputs, the parameter list is an empty set of closed parentheses: **( )**.

After the parameter list, there is a : (colon) to end the definition header.

The body of the procedure is a <***block***>, which is the code that implements the procedure. The block is indented inside the definition. Proper indentation tells the interpreter when it has reached the end of the procedure definition.

Consider how to turn the code for finding the first link into a **get_next_target** procedure that finds the next link target in the page contents. Here is the original code:

```
start_link=s.find('<a href=')
start_quote=s.find('"',start_link)
end_quote=s.find('"',start_quote+1)
url=s[start_quote+1:end_quote]
```

Next, to make this a procedure, we need to determine what the inputs and outputs are.

What are the inputs for the procedure, **get_next_target**?

A string giving contents of the rest of the web page.

What should the outputs be for **get_next_target**?

Best answer is url and end_quote

Create a procedure string 's' as parameter and return url

To make the **get_next_target** procedure, we first add a procedure header and indent our existing code in a block

To finish the procedure, we need to produce the outputs. To do this, introduce a new Python statement called return.

The syntax for return is:

return<***expression***>**,** <***expression***>**, ...**

A ***return*** statement can have any number of expressions. The values of these expressions are the outputs of the procedure.

A ***return*** statement can also have no expressions at all, which means the procedure produces no output. This may seem silly, but in fact it is quite useful. Often, you want to define procedures for their side-effects, not just for their outputs. Side-effects are visible, such as the printing done by a ***print*** statement, but are not the outputs of the procedure.

Complete the **get_next_target** procedure by filling in the **return** statement that produces the desired outputs.

```
# Create a procedure string 's' as parameter and return url
def get_next_target(s):
    start_link=s.find('<a href=')
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url
```

Create Procedure for rest of String

In order to use a procedure, you need the name of the procedure, followed by a left parenthesis, a list of the procedure's inputs (sometimes called operands or arguments), closed by right parenthesis:

<***procedure***>***(***<***input***>**,**<***input***>**, ...)**

For example, consider the **rest_of_string** procedure defined as:

```
s1='Hello RGUKT'

def rest_of_string(string):
    return string[4:]

print(rest_of_string(s1))
```

Module 2 – Getting Links

So far we have defined a procedure to eliminate writing tedious code:

```
def get_next_target(s):
    start_link=s.find('<a href=')
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote
```

Although you have not yet used a procedure that returns two things, it is pretty simple to do. You can do this by having two values on the left side of an assignment statement.
Assigning multiple values on the left side of an assignment statement is called **multiple assignment**. To write a multiple assignment you can put any number of names separated by commas, an equal sign and then any number of expressions separated by commas. The number of names and the number of expressions has to match so that the value of the first expression can be assigned to the first name and so on.
<name1>, <name2>, ... = <expression1>, <expression2>, ...

```
a, b = 1, 2
```

Therefore, in order to get the two values, (url, end_quote) to return from the procedure above, you will need to have the

```
url,endpos=get_next_target(s)
print(url,endpos)
```

two variables on the left side and the procedure on the right. Here is the syntax to do that:
What does mean of t,s =s,t ? it is mean Swaps the values of s and t
No Links
There is one concern with the **get_next_target** procedure that has to be fixed before tackling the problem of outputting all of the links, which is:
What should happen if the input does not have another link?
Test the code in the interpreter using a test link to print **get_next_target**:

```
def get_next_target(s):
    start_link=s.find('<a href=')
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote
```

When you run this code you get both outputs as a tuple, that is, the link followed by the position of the end quote. A tuple is an immutable list, meaning it cannot be modified. In the same way a list is enumerated with brackets, a tuple definition is bounded by parenthesises.
Or you can write this using a double assignment to return just the url:

```
def get_next_target(s):
    start_link=s.find('<a href=')
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote

s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a>'
url,endpos=get_next_target(s)
print(url,endpos)
```

```
def get_next_target(page):
    start_link = page.find('<a href=')
    start_quote = page.find('"', start_link)
    end_quote = page.find('"', start_quote + 1)
    url = page[start_quote + 1:end_quote]
    return url, end_quote

url, endpos = get_next_target('good')

print url
goo
```

What happens if we pass in a page that doesn't have a link at all?
The program returns, "goo" because when the find operation does not find what it is looking for it returns -1. When -1 is used as an index, it eliminates the last character of the string.
NOTE: using -1 for starting position in .find procedure makes the procedure to start searching at the end of string. (-1 = last index- on string)
Compare with a string that includes double quotes:

```
def get_next_target(s):
    start_link=s.find('<a href=')
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote

s='Hello "RGUKT" NUZVID'
url,endpos=get_next_target(s)
print(url,endpos)
#(Hello  6)
```

Think about making **get_next_target** more useful in the case where the input does not contain any link. This is something you can do! Here is a hint:
Write an **if** statement that will return *None* for the **url** when no hyperlink is found in the page.

```
def get_next_target(s):
    start_link=s.find('<a href=')
    if start_link==-1:
        return None,0
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote

s='Not "good" at all!'
url,endpos=get_next_target(s)
print(url,endpos)
#(Hello   6)

if url:
    print('Here')
else:
    print('Not Here')
```

Print All Links

At this point you have a piece of code, **get_next_target**, to replace a formerly tedious program. Here is where you are

```
s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a>'
url,endpos=get_next_target(s)

print(url)
s=s[endpos:]
url,endpos=get_next_target(s)
print(url,endpos)
```

so far, with a few modifications:
This code will have to repeat and keep going until the url that's returned is None.<
So far, you have seen a way to keep going, which is a while loop, you have seen a way to test the url, and now you

```
def get_next_target(s):
    start_link=s.find('<a href=')
    if start_link==-1:
        return None,0
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote

def print_all_links(s):
    while True:
        url,endpos=get_next_target(s)
        if url:
            print(url)
            s=s[endpos:]
        else:
            break

s='<a href="www.rguktn.ac.in">RGUKT NUZVID</a> Hello <a href="www.rgukt.in">RGUKT Kadapa</a>'
print_all_links(s)
```

have everything you need to print all the links on the page!

Module 3 - Presenting Results

Introduction

The main new topic for module 1 is structured data. By the end of this module, you will have finished building a simple web crawler.
The closest thing you have seen to structured data so far is the string type introduced in unit-4 and used in many of the procedures in previous unit modules. A string is considered a kind of structured data because you can break it down into its characters and you can operate on sub-sequences of a string. This unit introduces lists, a more powerful and general type of structured data. Compared to a string where all of the elements must be characters, in a list the elements can be anything you want such as characters, strings, numbers or even other lists!

Collecting Links

Now we are ready to finish our web crawler!
You need to start by finding all the links on the seed page, but instead of just printing them like you did in Unit 2, you need to store them in a list so you can use them to keep going. Go through all the links in that list to continue our crawl, and keep going as long as there are more pages to crawl.
The first step to define a procedure **get_all_links** that takes as input a string that represents the text on a web page and produces as output a list containing all the URLs that are targets of link tags on that page.

Get All Links Here is a recap of the code from Unit 4:

```
def print_all_links(page):
    while True:
        url, endpos = get_next_target(page)
        if url:
            print url
            page = page[endpos:]
        else:
            break
```

We defined a procedure, **get_next_target**, that would take a page, search for the first link on that page, return that as the value of **url** and also return the position at the end of the quote is so we know where to continue.
Then, we defined the procedure, **print_all_links**, that keeps going as long as there are more links on the page. It will repeatedly find the next target, print it out, and advance the page past the end position.
What we want to do to change this is instead of printing out the URL each time we find one, we want to collect the URLs so we may use them to keep crawling and find new pages. To do this, we will create a list of all of the links we find. We change the **print_all_links** procedure into **get_all_links** so that we can use the output, which will be a list of links, which will correspond to the links we were originally printing out.

Links

As an example of how this should work, there is a test page at test.html.(**Single webpage**) It contains four link tags that point to pages about crawling, walking, and flying (you can check them out for yourself by clicking on links on the test page in your web browser).
Here is how **get_all_links** should behave:

- *links = get_all_links(get_page('//test..html')*
- print links ['http://www.rgukt.in/', 'http://www.rguktn.ac.in', 'http://www.rguktrkv.ac.in', 'http://www.rgukts.ac.in', 'http://www.rgukto.ac.in']

Because the result is a list, we can use it to continue crawling pages. Think on your own how to define **get_all_links**, but if you get stuck, use the following quizzes to step through the changes we need to make.
What should the initial value of **links** be? Remember, your goal for **get_all_links** is to return a list of all the links found on a page. You will use the links variable to refer to a list that contains all the links we have found.

```
def get_all_links(page):
    links = []
    while True:
        url, endpos = get_next_target(page)
        if url:
            links.append(url)
            page = page[endpos:]
        else:
            break
    print(links)
```

Finishing the Web Crawler

At this point we are ready to finish the web crawler. The web crawler is meant to be able to find links on a seed page, make them into a list and then follow those links to new pages where there may be more links, which you want your web crawler to follow.
In order to do this the web crawler needs to keep track of all the pages. Use the variable **tocrawl** as a list of pages left to crawl. Use the variable crawled to store the list of pages **crawled**.

Crawling Process - First Attempt

Here is a description of the crawling process. We call this **pseudocode** since it is more precise than English and structured sort of like Python code, but is not actual Python code. As we develop more complex algorithms, it is useful to describe them in pseudocode before attempting to write the Python code to implement them. (In this case, it is also done to give you an opportunity to write the Python code yourself!)

- start with **tocrawl** = [seed]
- **crawled = []**
- while there are more pages **tocrawl**:
    - pick a page from **tocrawl** add that page to **crawled** add all the link targets on this page to **tocrawl** return **crawled**

Crawling Process- First Attempt

#!wiki What would happen if we follow this process on the test site, starting with the seed page http://www.udacity.com/cs101x/index.html ?

1. It will return a list of all the urls reachable from the seed page.
2. It will return a list of some of the urls reachable from the seed page.
3. It will never return.

Implement your web crawling procedure, **crawl_web**, that takes as input a seed page url, and outputs a list of all the urls that can be reached by following links starting from the seed page.
To start the **crawl_web** procedure, provide the initial values of tocrawl and crawl:

```
def crawl_web(seed):
    tocrawl = _____ - initialize this variable
    crawl = _______ - initialize this variable
```

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
```

Crawl the Web Loop

The next step is to write a loop to do the crawling, where you keep going as long as there are pages to crawl. To do this, you will use a **while** loop, with **tocrawl** as your test condition. You could use **len(tocraw) == 0** to test if the list is empty. There is an easier way to write this using just **tocrawl**. An empty list (a list with no elements) is interpreted as false, and every non-empty list is interpreted as true.

Inside the loop, we need to choose a page to crawl. For this quiz, your goal is to figure out a good way to do this. There are many ways to do this, but using things we have learned in this unit you can do it using one line of code that both initializes **page** to the next page we want to crawl and removes that page from the **tocrawl** list.

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    while tocrawl:
        page = _____________
```

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    while tocrawl:
        page = tocrawl.pop()
```

Crawl If

The next step is to manage the problem of cycles in the links. We do not want to crawl pages that we've already crawled, so what we need is some way of testing whether the page was crawled.

To make a decision like this, we use if. We need a test condition for if that will only do the stuff we do to crawl a page if it has not been crawled before.

Finishing Crawl Web

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    while tocrawl:
        page = tocrawl.pop()
        if _________________
```

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
```

Now you're ready to finish writing our crawler. Write two lines of code to update the value of tocrawl to reflect all of the new links found on page and update the value of crawled to keep track of the pages that have been crawled.

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            union(tocrawl, get_all_links(get_page(page)))
            crawled.append(page)
    return crawled
```

Unit 6 - How to structure data (Responding to Search Queries)

Module 1 - Building a Search Engine

Introduction (Building a Search Engine)

In unit 5 you are going to learn how to finish the code for your search engine and how to respond to a query when someone wants the given web pages that correspond to a given keyword. You will also learn about how networks and the World Wide Web work to understand more about how you can build up your search index.

The main new computer science idea you will learn is how to build complex data structures. You will learn how to design a structure that you can use so that you can respond to queries without needing to rescan all the web pages every time you want to respond to a query. The structure you will build for this is called an index. The goal of an index is to map a keyword and where that keyword is found. For example, in the index of a book you can see a page number which serves as a map to where a term or concept can be found. The key ideas in index will allow us to find references to what we want. With a search engine the index gives you a way for a keyword to map to a list of web pages, which are the urls where those particular keywords appear. Once you have done the work of building an index, then the look-ups are really fast.

Deciding on data structures is one of the most important parts of building software. As long as you pick the right data structure, the rest of the code will be a lot easier to write.

Which of these data structures would be a good way to represent the index for your search engine?

[[<keyword1>, <url1, 1="">, <url1, 2="">], [<keyword2>, <url2, 1="">, ...]]

Add to Index

Define a procedure, **add_to_index**, that takes three inputs:

- an index [<keyword>, [<url>, ..], ...]
- a keyword string
- a url string

If the keyword is already in the index, add the url to the list of urls associated with that keyword.

If the keyword is not in the index, add an entry to to the index: **[keyword, [url]]**

For example:

```
index=[]
add_to_index(index,'RGUKT','http://www.rgukt.in')
add_to_index(index,'NUZ', 'http://www.rguktn.ac.in')
add_to_index(index,'RGUKT','http://www.rguktongole.ac.in')
add_to_index(index,'SKLM','http://www.rgukts.ac.in')
print(index)
def add_to_index(index, keyword, url):
    for i in index:
        if i[0] == keyword:
            if url not in i[1]: #if url found in the existing key then not appen
                i[1].append(url)
                return
            else:
                return
    # not found, add new keyword to index
    index.append([keyword, [url]])
```

This code starts with the empty list index. After the two lines of code the empty list will contain two lists beginning with the keywords, RGUKT and NUZ.

```
index=[]
add_to_index(index,'RGUKT','http://www.rgukt.in')
add_to_index(index,'NUZ', 'http://www.rguktn.ac.in')
add_to_index(index,'RGUKT','http://www.rguktongole.ac.in')
add_to_index(index,'SKLM','http://www.rgukts.ac.in')
print(index)
```

In this code, **RGUKT** is already in the index and you don't want to add a new entry to the index itself. Since **RGUKT** is already in the index, what you want to do is add the new url to the list already associated with that keyword.

Lookup Procedure

Define a procedure, lookup that takes two inputs:

- An index: A list where each element of the list is a list containing a keyword and a list as its second element. The second list element is a list of urls where that keyword appears.
- The keyword to lookup

The output should be a list of the urls associated with the keyword. If the keyword is not in the index, the output should be an empty list.

For example:

lookup('RGUKT') → ['http: //www.rgukt.in', 'http://www.rguktn.ac.in']

```
#To Create lookup(index,key)
def lookup(index,key):
    for i in index:
        if i[0]==key:
            return i
    return None
```

Building the Web Index

To build your web index, you want to find a way to separate all the words on a web page. It is possible to use the concepts you've already seen to build a procedure to do this, however, Python has a built-in operation that will make this much simpler.

Split. When you invoke the split operation on a string, the output is a list of the words in the string.

<"string">.split()

[<"word">, <"word">, ...]

For Example:-

```
>>> string='When you invoke the split operation, on a string'
>>> string.split()
['When', 'you', 'invoke', 'the', 'split', 'operation,', 'on', 'a', 'string']
>>>
```

This operation does a pretty good job of separating out the words in the list so that they will be useful. However, in the case of '**operation,**' which was followed by a comma in the quote, for the keyword you would not want to include the comma. While this isn't perfect, it is going to good enough for now.

Here is another example of how **split** works, using triple quotes ("""). Using the triple quotes you can define one string over several lines:

```
>>> string='''When you invoke the (split operation), on a string'''
>>> string
'When you invoke the (split operation), on a string'
>>> string.split()
['When', 'you', 'invoke', 'the', '(split', 'operation),', 'on', 'a', 'string']
>>>
```

This still has similar problems to the first example, where the parentheses are included in the word '**(split**'

Add Page to Index

Define a procedure, **add_page_to_index** that takes three inputs:

- index
- url (string)
- content (string)

It should update the index to include all of the word occurrences found in the page content by adding the url to the word's associated url list.

For example:

```
#To Create add_page_to_index(index,url,content)
def add_page_to_index(index,url,content):
    for k in content.split():
        add_to_index(index,k,url)
    return index
#Call to add_page_index()
add_page_to_index(index,'http://www.rgukt.in','RGUKT SKLM RKV')
print(index)
[['RGUKT', ['http://www.rgukt.in']],
 ['SKLM', ['http://www.rgukt.in']],
 ['RKV', ['http://www.rgukt.in']]]
print(index[0])
['RGUKT', ['http://www.rgukt.in']]
print(index[1])
['SKLM', ['http://www.rgukt.in']]
```

Now, add a page called **http://www.rguktn.ac.in**

```
index=[]
add_page_to_index(index,'http://www.rgukt.in','RGUKT SKLM RKV')
add_page_to_index(index,'http://www.rguktn.in','RGUKT NUZ RKV ongole')

print(index)
[['RGUKT', ['http://www.rgukt.in', 'http://www.rguktn.in']],
 ['SKLM', ['http://www.rgukt.in']],
 ['RKV', ['http://www.rgukt.in', 'http://www.rguktn.in']],
 ['NUZ', ['http://www.rguktn.in']], ['ongole', ['http://www.rguktn.in']]]
```

Have a look at the entries when you index [1]

```
print(index[1])
['SKLM', ['http://www.rgukt.in']]
```

When you have already created list than you go for check the url with keyword using lookup ()

```
index=[]
add_page_to_index(index,'http://www.rgukt.in','RGUKT SKLM RKV')
add_page_to_index(index,'http://www.rguktn.in','RGUKT NUZ RKV ongole')
print(index)
#call to lookup
print("\n",lookup(index,'RGUKT'))

['RGUKT', ['http://www.rgukt.in', 'http://www.rguktn.in']]
```

Call with different keyword

```
#call to lookup
print("\n",lookup(index,'RKV'))

['RKV', ['http://www.rgukt.in', 'http://www.rguktn.in']]
```

Finishing the Web Crawler Returning to the code you wrote before for crawling the web; make some modifications to include the code you've just written.

First, a quick recap on how the **crawl_web** code below works before incorporating the indexing:

```
def crawl_web(urls):
    tocrawl = [urls]
    crawled = []
    while tocrawl:
        url = tocrawl.pop()
        if url not in crawled:
            union(tocrawl,get_all_links(get_page(url)))
            crawled.append(url)
    return crawled
```

First, you defined two variables **tocrawl** and **crawled**. Starting with the **seed** page, **tocrawl** keeps track of the pages left to crawl, whereas **crawled** keeps track of the pages that have already been crawled. When there are still pages left to crawl, remove the last page from **tocrawl** using the **pop** method. If that page has not been crawled yet, get all the links from the page and add the new ones to **tocrawl**. Then, add the page that was just crawled to the list of crawled links. When there are no more pages to crawl, return the list of **crawled** pages.

Adapt the code so that you can use the information found on the pages crawled. The changed code is below. First, add the variable **index**, to keep track of the content on the pages along with their associated urls. Since you are really interested in the **index**, this is what we will return. It is possible to return both **crawled** and **index**, but to keep it simple just **return index**. Next, add a variable, **content** to replace **get_page(url)**. This variable will be used twice, once in the code already there and once in the code to be filled in for the quiz. The procedure **get_page(url)** is expensive as it requires a web call, so we don't want to call it more often than is necessary. Using the variable **content** means that the call to the procedure **get_page(url)** only needs to be performed once and then the result is stored and can be used over and over without having to go through the expensive call again.

Finishing the Web Crawler with index List

```
def crawl_web(urls):
    tocrawl = [urls]
    crawled = []
    index=[]
    while tocrawl:
        url = tocrawl.pop()
        if url not in crawled:
            content=get_page(url)

            #Write Code Here


            union(tocrawl,get_all_links(content))
            crawled.append(url)
    return index
```

Module 2 – Ranking Web Pages

Startup

You now have a functioning web crawler! From a seed you can find a set of pages; for each of these pages you can add the content from that page to an index, and return that index. Additionally, since you have already written the code you can do the look-up that will return the pages for that keyword.

But you're not completely done yet. In next module, you will see how to make a search engine faster and in another module you will learn how to find the best page for a given query rather than returning all the pages.

Before then, you need to understand more about how the Internet works and what happens when you request a page on the World Wide Web.

The Internet→get_page(url)

So far, you've been using the get_page function where you pass in a url and it passes out the content of the page. This is the python code that does that:

The protocol used on the web is called Hypertext Transfer Protocol which is abbreviated as HTTP. When you look in your browser, almost all the urls that you use start with http. That indicates that when the page is requested, the protocol to be used to talk to the server is this Hypertext Transfer Protocol. It's a very simple protocol, and only has two main messages. One of those messages is **GET**.

The client can send a message to the server which says **GET** followed by the name of the object you want to get, **GET <object>**. That's all the client does. The python code for **get_page**:

def get_page(url):
try:
importurllib
returnurllib.urlopen(url).read()
except:
return ""

calls a library function **urllib.urlopen(url)** which actually does this. **Or**

```
def get_page(url):
    s=""
    try:
        import urllib.request
        f=urllib.request.urlopen(url).read()
        s=str(f.decode("utf8"))
        f.close()
        return s
    except:
        return s

url='file:///E:/Teach/2017-18/PUC2/Python/WebCrawler/Unit3/test.html'
print(get_page(url))
```

After the client sends the message, the server will receive it. The server runs some code on it, finds the file that was requested, perhaps runs some more code, and then sends back a response with the contents of the requested **<object>**. That's the whole protocol.

Ranking Web Pages

Having survived the bunny uprising, you're ready to move on to the main goal of the class, which is to return the best page that matches the search query rather than returning all the pages. It's important to do this well. This is something that really distinguished Google from earlier search engines. They had a much smarter way of ranking pages. Often the first or second item in the returned search was what the user was looking for.

To recap from earlier units; first, you learned to build a crawler. The crawler followed all the links in the web pages and built an index. After, you had an index, which was a hash table, where you could look up a keyword. You could find the entry where the keyword appears, and find the list of all the urls of all the pages that contain that keyword.

The order in which the pages appear in the list of urls associated with a keyword is the order the pages were crawled.

This process says nothing about which pages are best. In the early days of the web, when there weren't many pages, this maybe wasn't too much of a problem since only a few pages might match a given keyword. Those days are long gone and now there could be thousands, if not millions of pages containing a given keyword. A good search engine ranks the pages so that the one at the front of the list is the one the user most likely wants.

The problem of deciding how to rank the pages leads to the question of how to decide popularity, which is the topic of the next section.

Popularity

Consider a typical group of friends in middle school. One way to decide popularity is to look at friendship links. Friendship links go in one direction. Just because Bob is friends with Alice does not mean Alice is friends with Bob.

Is having a lot of friends enough to make you popular? No, it's not. You have to have the right sort of friends. It's no good to have lots of friends with no friends, you have to have friends who are popular.

Popularity is about having lots of friends who have lots of friends.

Initially, you can define popularity as the number of friends a person has. From the diagram above, it's the number of arrows pointing towards a person.

. *"popularity(p)" = "# No of people who are friends with p"*

This isn't quite right though because it doesn't take into consideration the number of friends of those friends. To do this, you could sum all the popularities of all the friends of a person. In mathematical notation:

. "popularity(p) = sigma over f E friends of p popularity(f)"

The notation σ , is the summation sign, which tells you to sum up the '"popularity(f)"'. The text under the symbol tells you what values of "f" to include in the sum. It tells you to sum the popularity of each friend, "f" of "p". If you're unfamiliar with the mathematical notation, here's the same thing in Python pseudocode.

Pseudocode is an outline of the code which is written for human readability rather than for a computer.

In the code below, you know that friends p means the friends of p, but it's not actually defined anywhere so the computer will not be able to run it.

{{{#!highlight python

```
def popularity(p):
score = 0
 for f in friends(p):
score = score + popularity(f)
 return score
```

Circular Definitions

How can this problem to be fixed? With all the other recursive definitions you had a base case - a way to stop.

For the recursive factorial definition, you predefined the value at 0 to be 1, that is, **factorial(0) -->1**.

For the palindromes, you defined an empty string to be a palindrome, that is, **palindrome(' ') --> True**.

For the Fibonacci sequence, you had two base cases.

For all of these definitions you had a starting point that was not defined in terms of the thing you're defining. That is why they were good recursive definitions. You had a base case. Maybe inventing a base case will solve the popularity problem.

Assume sai has popularity 1, and try that as a base case. For the mathematical definition, this is:

- *popularity('sai')* = 1
- *popularity(p)* = sigma over friends E friends of p *ppopularity(f)*.

For the code, you need to add the base case, which is an if statement, to see if the person you're checking the popularity of is **sai**.

```
def popularity(p):
    if p =="sai":
        return 1
    score =0
    for f in friends(p):
        score = score + popularity(f)
    return score
```

Relaxation

There was no sensible base case that provides a good recursive definition. Instead, an algorithm called the **relaxation algorithm** can be used. The basic idea is simple. Start with a guess and then loop where you do something to improve the guess. There isn't a good stopping place yet, or a clear starting place, like setting the popularity of sai. Each time you go through the loop, the guess will be refined, and at some point you'll stop and take that to be the result you want.
In summary:

 # start with a guess
while not done:
make the guess better

The procedure will have an extra parameter, which is a time step:
popularity(<time step>,<person>)--> score
The base case will be to set the popularity for everyone at time 0 to 1. For the recursive step, the popularity of each of their friends at the previous time step, t-1is summed. In mathematical terms this is:
Base case: popularity(0, p) = 1
Recursive step: popularity(t, p) = sigma over f E friends(p) popularity(t-1, p)for t> 0.

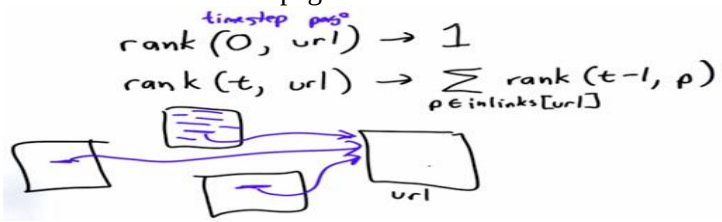
```
def popularity(t,p):
    if t == 0:  # base case, at time step 0
        return 1 # the score is always 1
    else:
        score = 0
        for f in friends(p): # summing over the friends
            score = score + popularity(t-1,f) # adding the popularity at the time step before
        return score
```

So, now you have a new definition written in both mathematical notation and in Python code.

Module 3 - Page Rank

Page Rank

Ranking web pages is the same as measuring popularity for people. Links on the web are analogous to friendships in that model. And links from some pages count for more than links from others.
This model is a random web surfer who starts at a random page and then follows the links at random. The popularity of a page is the probability that the random surfer reaches a particular page.
The rank function is defined recursively over time. At time 0, the rank of a url is 1. At time t, the rank of a url is the sum of the ranks for all pages that link to that url.

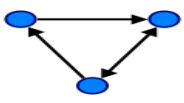


$$\sum_{p \in inlinks[url]} rank(t-1, p) / outlinks[p]$$

In addition, the rank contributed by each page is inversely weighted by the number of outlinks from that page. So, divide each rank in the sum by the number of outlinks from that page.

AltaVista Search Engine

On this model, pages with no links have a rank of 0, which makes it very hard to start a new page. So, instead each page will have some starting rank greater than 0.
Since the model represents the probability that a random surfer reached a given page, the ranks should be a probability distribution. This means that the ranks for all pages will sum up to 1. And so at time 0, instead of 1, the rank is 1/N for each page, where N is the number of pages.
A **damping constant** is used to diminish the raw values of the ranking algorithm. In the model, this represents the probability that a page was reached via following a link. Set the damping constant, d, to 0.8 for now.
At time t, add (1-d)/N to each rank that will represent the probability that the page was not reached via following a link. Multiply the first term, the sum, by d. This will make the ranks a probability distribution at any given time step.
Urank
Since PageRank is a registered trademark of Google, the algorithm will be called URank instead.
URank needs to keep track of which pages link to which pages, so you'll need a data structure to keep track of which pages link to which other pages. You'll use a directed graph. A **directed graph** is a data structure where nodes are linked to other nodes, and the links only go one way.
A **directed graph** (or **digraph**) is a graph that is a set of vertices connected by edges, where the edges have a direction associated with them.



So **crawl_web** will produce a graph in addition to an index, where the graph gives a mapping from each page to all the pages it links to. The graph will be a dictionary, since it is a mapping from individual URLs to lists of URLs.

Also, add the variable **outlinks** which stores the return value of **get_all_links(content)** so it can be used for both **tocrawl** and **graph**. Adding the line to update the graph will be a quiz.

Implementing Urank (pageRank)

To add one line of code which will update the graph for each page crawled.

```
def crawl_web(seed): # returns index, graph of inlinks
    tocrawl = [seed]
    crawled = []
    graph = {}   # <url>, [list of pages it links to]
    index = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            outlinks = get_all_links(content)

            #Insert Code Here
            union(tocrawl, outlinks)
            crawled.append(page)
    return index, graph
```

Note: - Insert the following code in missing place

graph [page] = outlinks

Computing Page Rank

Much like the Fibonacci solution before, you can use an iterative solution to implement the recursive definition, instead of a recursive solution. Use a loop to update the page ranks for each time step.

The output of **compute_ranks** is a dictionary mapping each URL to its rank, which is a number.

For the homework, the function **lookup_best** will find the highest-ranking page for a given keyword, using both the index and the page ranks.

Formal Calculations

Remember how the ranking function was defined; **npages** refers to the number of pages:

rank(0, url) = 1/npages

rank(t, url) = (1-d)/npages +

*sum([for each page 'p' that links to URL, d*rank(t-1, p)/(number of outlinks from p)])*

Since the ranks should not depend on the order that the pages were examined by the algorithm, you need to keep track of the ranks at the last time step. Keep two separate dictionaries, **ranks** and **newranks**, where **newranks** is the working space for each time step. This is similar to the trick used earlier for the iterative Fibonacci solution.

Computer Rank

d is the damping factor.

numloops is the number of times to do our "relaxation". Changing the number of loops can give different results.

npages is the number of pages in the graph, which is given by **len(graph)**.

Initially, all ranks are set to **1.0/npages** (remember: the decimal point to use floating point arithmetic), matching the base case of the recursive definition.

Then, the algorithm loops through **numloops** times, updating the rank for each page in the graph. **newrank** is initialized to **(1-d)/npages**, and then it will be to update **newrank** with the sum of the inlink ranks. Then **newrank** is stored in the **newranks** dictionary.

When the **for** loop is finished, assign **newranks** to **ranks**, since the calculations are finished for that time step. At the end of the function, return **ranks**.

Finishing Urank (PageRank)

Update **newrank** based on the values of the previous iteration, **ranks**, and the incoming links from **graph**.

```
def compute_ranks(graph):
    d = 0.8 # damping factor
    numloops = 10

    ranks = {}
    npages = len(graph)
    for page in graph:
        ranks[page] = 1.0 / npages

    for i in range(0, numloops):
        newranks = {}
        for page in graph:
            newrank = (1 - d) / npages

            #Insert Code Here

            newranks[page] = newrank
        ranks = newranks
    return ranks
```

Add Final Urank Code:

```
def compute_ranks(graph):
    d = 0.8 # damping factor
    numloops = 10
    ranks = {}
    npages = len(graph)
    for page in graph:
        ranks[page] = 1.0 / npages

    for i in range(0, numloops):
        newranks = {}
        for page in graph:
            newrank = (1 - d) / npages
            for node in graph:
                if page in graph[node]:
                    newrank=newrank+d*(ranks[node]/len(grahp[node]))
            newranks[page] = newrank
        ranks = newranks
    return ranks
```

Final Project - CODE

```
def get_next_target(s):
    start_link=s.find('<a href=')
    if start_link==-1:
        return None,0
    start_quote=s.find('"',start_link)
    end_quote=s.find('"',start_quote+1)
    url=s[start_quote+1:end_quote]
    return url,end_quote

def get_all_links(page):
    links = []
    while True:
        url, endpos = get_next_target(page)
        if url:
            links.append(url)
            page = page[endpos:]
        else:
            break
    return links
#add_to_index( index,key,url)
def add_to_index(index, keyword, url):
    for i,j in index.items():
        if i== keyword:
            if url not in j:#if url not found in the existing key then append
                j.append(url)
                return
            else:
                return
    # not found, add new keyword to index
    index[keyword]= [url]

#To Create lookup(index,key)
def lookup(index,key):
    for i,j in index.items():
        if i==key:
            return i,j
    return None

#To Create add_page_to_index(index,url,content)
def add_page_to_index(index,url,content):
    for k in content.split():
        add_to_index(index,k,url)
    return index

def union(l1,l2):
    for i in l2:
        if i not in l2:
            l1.append(i)
    return l1

def get_page(url):
    s=""
```

```
    try:
        import urllib.request
        f=urllib.request.urlopen(url)
        mybytes=f.read()
        s=str(mybytes.decode("utf8"))
        f.close()
        return s
    except:
        return s

def compute_ranks(graph):
    d = 0.8 # damping factor
    numloops = 10
    ranks = {}
    npages = len(graph)
    for page in graph:
        ranks[page] = 1.0 / npages

    for i in range(0, numloops):
        newranks = {}
        for page in graph:
            newrank = (1 - d) / npages
            for node in graph:
                if page in graph[node]:
                    newrank=newrank+d*(ranks[node]/len(grahp[node]))
            newranks[page] = newrank
        ranks = newranks
    return ranks

def crawl_web(seed): # returns index, graph of inlinks
    tocrawl = [seed]
    crawled = []
    graph = {}  # <url>, [list of pages it links to]
    index = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            outlinks = get_all_links(content)
            graph[page]=outlinks
            union(tocrawl, outlinks)
            crawled.append(page)
    return index, graph,crawled

seed='http://google.co.in/'
print(crawl_web(seed)[0])
print("************************************************")
print(crawl_web(seed)[1])
print("************************************************")
print(crawl_web(seed)[2])
print("************************************************")
print(compute_ranks(crawl_web(seed)[1]))
```