

802
Lecture No. 01 Computational Thinking & Problem Solving

Topic(s) to be covered	Fundamentals of Computing - Generations of a Computers - Components of a Computer
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
Lo1	Understand the fundamentals	Understand
Lo2	Know the Generations	Understand
Lo3	Describe the components	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Fundamentals of Computing

- * Computer is an electronic device that can be programmed to accept data (Input), process it & generate result (Output).
- * A computer along with additional hardware & software together is called a computer system.
- * Computers do a lot of different tasks such as storing data, weather forecasting, booking airlines

Railway or Movie tickets & even Playing games.

Data: Data is defined as an un-processed Collection of raw facts, suitable for communication, interpretation or processing.

Information: Information is a Collection of facts that is processed to give meaningful, ordered or Structured Information.

Generations of a Computer

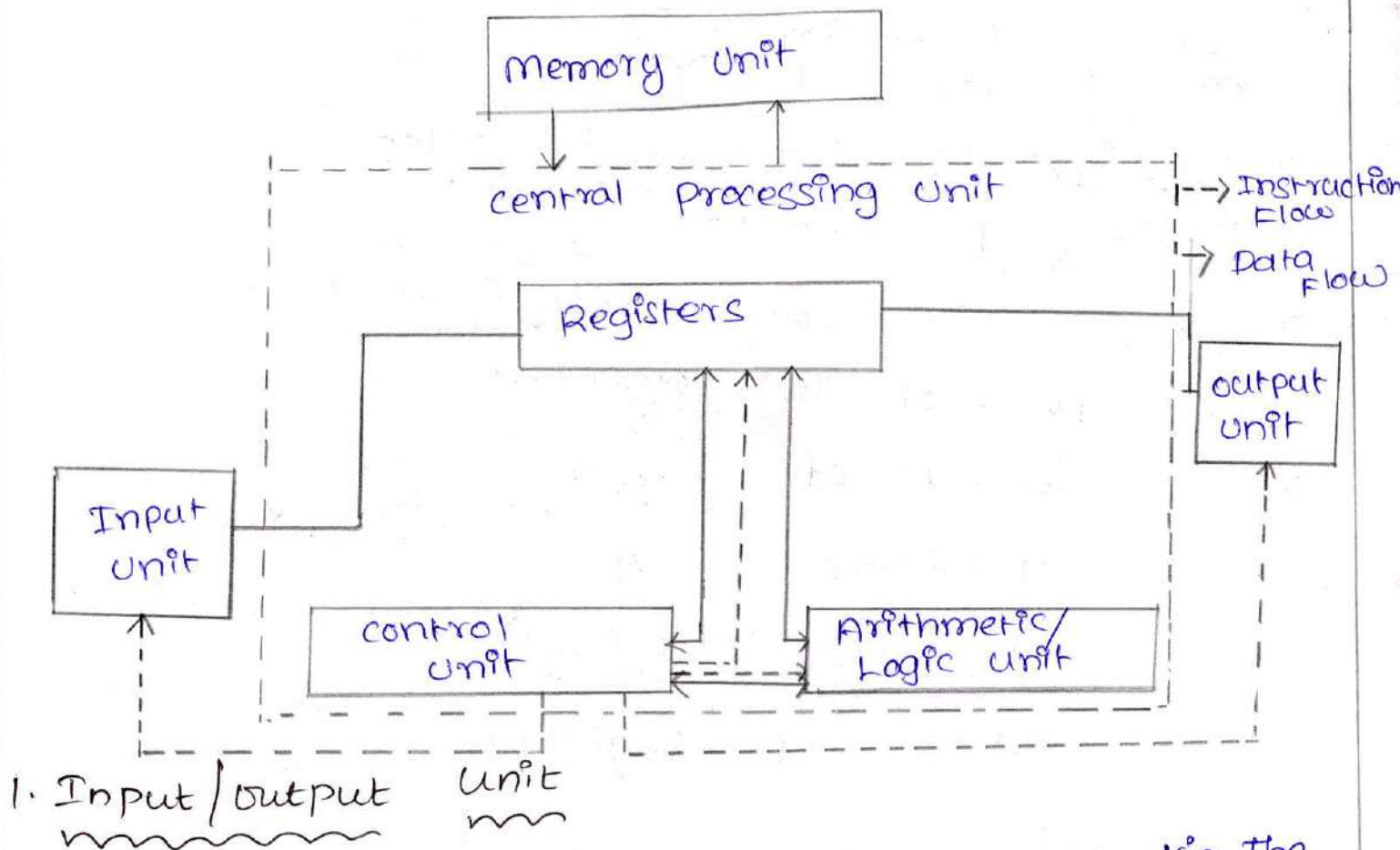
Generation	Period	Main Component used	Merits/Demerits
First Generation	1940 - 1956	Vacuum Tubes	→ Big in size → Consumed more Power → Malfunction due to Overheat → Machine language was used (Eg) ENIAC, EDVAC, UNIVAC
Second Generation	1956 - 1964	Transistor	→ Smaller Compared to First Generation → Generated less heat → Consumed less Power Compared to first Generation → Punched Cards were used. → First OS developed. Batch Processing & Multiprogramming OS

			Machine language as well as Assembly language was used (Eg) IBM 1401, IBM 1620
Third Generation	1964 - 1971	Integrated Circuits	→ Computers were smaller, faster & more reliable → Consumed less power → High level languages were used (Eg) IBM 360 Series Honeywell 6000 Series
Fourth Generation	1971 - 1980	Micro Processors	→ Smaller & Faster → Microcomputer Series such as IBM & APPLE were developed → Portable Computers were introduced.
Fifth Generation	1980 - Till Date	Ultra-large Scale Integration	→ Parallel Processing → Super Conductors → Computer's size was reduced → Can recognise images & Graphics → Introduction of Artificial Intelligence & Expert Systems. → Able to solve high complex problems including decision making & logical reasoning.

Sixth Generation	In future	→ Parallel & Distributed Computing → Computers have become Smarter, faster & Smaller → Development of robotics → Natural language Processing → Development of Voice recognition Software
------------------	-----------	--

Components of a Computer

- * The Computer is the combination of hardware & software.
- * Hardware is the physical component of a computer like motherboard, memory devices, monitor, keyboard etc.,
- * Software is the set of programs or instructions. Both hardware & software together make the computer system to function.
- * The computer system hardware comprises of 3 main components.
 - 1) Input / Output (I/O) unit,
 - 2) Central Processing Unit (CPU)
 - 3) Memory unit



1. Input/output unit

* The user interacts with the computer via the I/O unit

* The Input unit

→ Accept the data from user

→ It converts the data into a form that is understandable by the computer.

→ The input is provided to the computer using devices like keyboard, trackball & mouse.

* The Output unit

→ Provides the processed data i.e.) the information to the user.

→ Provides output in a form that is understandable by the user

→ Some of the output devices are monitor & printer.

ii) Central Processing Unit

- CPU (or) the Processor is often called the Brain of Computer
- CPU Controls, Coordinates & supervises the Operations of the Computer
- CPU Consist of
 - 1) Arithmetic logic unit (ALU)
 - 2) Control Unit (CU)
 - 3) Set of Registers

i) Arithmetic logic unit (ALU)

- ALU Consist of 2 units
 - 1) Arithmetic unit
 - 2) Logic unit
- Arithmetic unit Performs arithmetic operations on the data. (Eg) addition, Subtraction, Multiplication & division.
- Logic unit Performs logic operations. (Eg) Comparison of numbers, letters & special Characters. Testing for greater than, less than or equal to condition.
- ALU performs Arithmetic & logic operations, & uses registers to hold the data that is being processed.

Registers

- * Registers are high-speed storage areas within the CPU, but have the least storage capacity.
- * Registers store data, instructions, addresses & intermediate results of processing. So registers are often called as "CPU's working memory".
- * Some of the important registers in CPU

i) ACC → Accumulator

ii) IR → Instruction Register

iii) PC → Program Counter

iv) DR → Data Register

Control Unit (CU)

- * The Control unit acts as a supervisor, controls & coordinates the activity of the other units of computer.

Memory Unit

- * The memory unit consists of cache memory, primary & secondary memory.
- * Memory unit stores the data & instructions, intermediate results & output temporarily during processing of data. This memory is called main memory / primary memory.
- * Example: RAM, ROM.

RAM

- RAM is Volatile
- It Stores data when The Computer is on
- Information in RAM gets erased when the Computer is turned off.
- RAM Provides temporary storage for data & instructions.
- It Provides a limited Storage Capacity, due to its high cost.

ROM

- ROM is non-volatile memory & is a read only memory
- Storage in ROM is Permanent in nature
- Information will not be erased even if the Computer is turned off.
- ROM Comes programmed by The manufacturer.

Secondary Memory

- It Provides backup storage for data & instructions
(eg) Hard disk drive, floppy drive & optical disk drive
- The data & instructions that are currently not being processed, but may be required later by CPU are stored in Secondary memory
- Secondary memory is Cheaper & high storage capacity than Primary memory.
- But it takes longer time to access the data & instructions.

Cache memory

- * The data & instructions that are required during the processing of data are brought from the Secondary Storage devices & stored in the RAM.
- * Next for processing they are taken from RAM to registers.
- * Time taken to move the data between RAM & CPU registers is large. This affects the speed of processing of computer & results in decreasing the performance of CPU.
- * Cache memory (high speed memory) is placed in between RAM & CPU.
- * Cache memory is a storage buffer that stores the data that is used more often.
- * So during processing, CPU first checks cache memory for required data. If data is not found in cache, then it looks in RAM for that data.
- * To access the cache memory, CPU does n't have to use the mother board's system bus for data transfer.
- * Cache is twice as fast as RAM.

Identification of Computational Problems

- * Identification of Computational Problem is part of scientific method, as it serves as the first step in a systematic process to identify, evaluate a problem & explore potential solutions.

Computational Problem Identification Consist of two Steps

1. Identifying & acknowledging That there is a Problem.

2. Developing a Problem Identification Statement.

→ Understanding Computational Thinking will give a foundation for solving Problems.

→ Computational Thinking is part of a problem solving process that ends with writing a Program.

→ Four Steps related to the Identification of Problem & its Solution i) Decomposition ii) Pattern recognition iii) abstraction & algorithm design.

Decomposition → Breaking the Problem down into smaller more familiar components so they can be managed easier.

Pattern Recognition → Similarities & trends are identified within the Problem.

Abstraction → Identification of key components of the Solution. Ignoring any unnecessary details.

Algorithm Design → Detailed step by step set of Instructions are created which explain how to solve the Problem.

→ The measure of good algorithm is one that can be passed to someone else to follow without the need of any extra explanation.

Suggested Questions / Assignments / Home works / any other

1. Describe in detail about the components of Computer system.
2. Explain the generations of Computer.
3. Elaborate the Identification of Computational thinking
4. Define Abstraction.
5. what is meant by Decomposition?

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 03 Algorithms

Topic(s) to be covered	Algorithms - Guidelines for writing algorithms - Building blocks of algorithms
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	understand the algorithm	understand
LO2	know the guidelines for writing algorithm, building blocks	understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Algorithms

- * Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task.
- * It is a Step-by-Step Procedure for solving any problem.
- * Algorithm is an English like representation, which is used to solve the problem.
- * To accomplish a particular task, different algorithms can be written.

*The algorithm can be implemented in many different languages

*The algorithm is independent of any programming language.

Guidelines for writing algorithm

1) An algorithm should be clear, precise & well defined.

2) It should always begin with word 'Start' & end with word 'Stop'

3) Each step should be written in separate line

4) Steps should be numbered as Step 1, Step 2 and so on.

(Example)

write an algorithm for addition of two numbers

Step 1: Start

Step 2: Read the two numbers a, b

Step 3: Add the two numbers a, b & Store the result in C

$$C = a + b$$

Step 4: Print the result in C

Step 5: Stop

(Example)

Write an algorithm to find the greatest among three numbers

Step 1: Start

Step 2: Read the three numbers A, B, C

Step 3: Compare A and B. If A is greatest
Perform Step 4 else Perform Step 5.

Step 4: Compare A and C. If A is greatest,
Output "A is the greatest" else
Output "C is the greatest".

Step 5: Compare B and C. If B is greatest,
Output "B is the greatest" else
Output "C is the greatest".

Step 6: Stop

Properties of an Algorithm

- 1) Finiteness: An algorithm must be terminated after a finite number of steps.
- 2) Definiteness: Each step of an algorithm must be precisely defined.
- 3) Input: The data must be present before any operations can be performed on it.
The initial data is supplied by a Read instruction.

(Eg) Read a, b

4. output: After executing all the steps, at least one output must be obtained.

The write statement is used to print the output.

5. Effectiveness: The operations to be performed in the algorithm can be carried out manually in finite intervals of time.

Advantages of algorithm

- 1) It is simple to understand step by step solution of the problem
- 2) It is easy to debug
- 3) It is independent of programming languages
- 4) It is compatible to computers, because each step of an algorithm can be easily coded into its equivalent high-level language.

Building Blocks of Algorithm

The algorithms can be constructed from basic building blocks. The building blocks are

- i) Statements ii) State
- iii) Control flow iv) Functions

Statement

- * The algorithm consist of finite number of state-ments. It must be an ordered form.
- * The time taken to execute all the statements of the algorithm should be finite & within a reasonable limit.

State:

- * Computational Processes in the real-world have State. As a process evolves, the state changes.
- * An algorithm starts from the initial state with some input values. As actions are performed its state changes, it ends with a final state.

Control Flow

- * An algorithm is a sequence of statements. However, after executing a statement, the next statement to be executed need not be the next statement in the algorithm.
- * The next statement to be executed may depend on the state of the process.
- * The order in which the statements are executed may differ from the order in which written in the algorithm.
- * The order of execution of statements is known as the "Control flow"

There are 3 Control flow Statements

- 1) Sequence Control flow
- 2) Selection Control flow
- 3) Iteration (looping) Control flow

These Control flows allow The program to make Choices, change direction or repeat actions.

Sequence Control flow

*In Sequential control flow, a sequence of Statements is executed one after another in the same order as they are written from top to bottom.

*The instructions in Sequence Control flow are executed exactly once.

(Example) Algorithm to find the sum of two numbers.

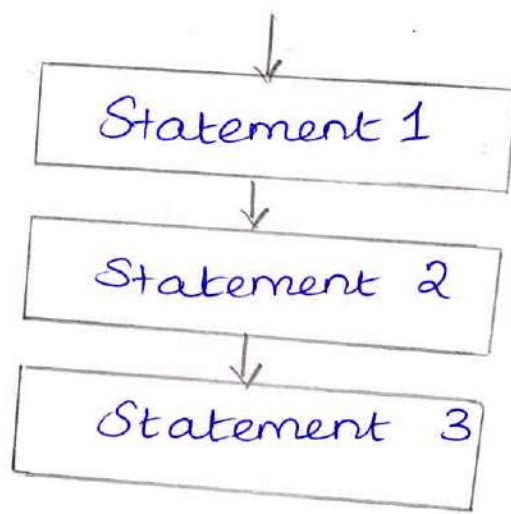
Step 1: Start

Step 2: Read two numbers A & B

Step 3: Calculate $Sum = A + B$

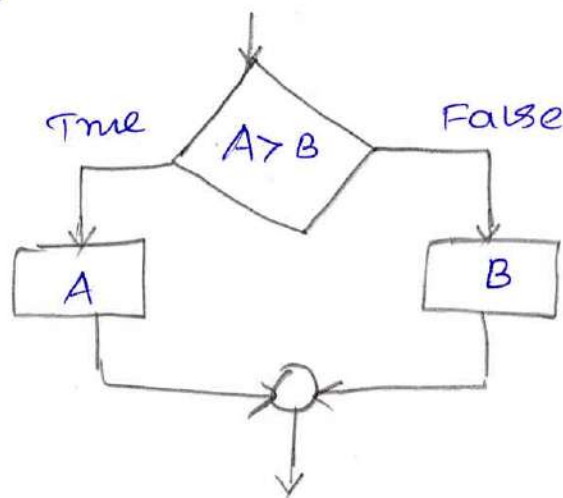
Step 4: Print the sum value

Step 5: Stop.



Selection Control Flow

* In Selection Control flow, a Condition is tested, & if the condition is true, one statement is executed. If the condition is false, an alternative statement is executed.



(Example) Algorithm to find greatest among two numbers

- Step 1: Start
- Step 2: Read the two numbers A, B
- Step 3: Compare A & B
- Step 4: If A is greatest print A is big
- Step 5: Else print B is big
- Step 6: Stop

Iteration (looping) Control flow

- * In Iteration Control flow, a set of statements are repetitively executed based upon a condition.
- * If Condition is true, the set of statements are executed again & again.
- * As soon as the condition becomes false, the repetition stops. This is known as looping statement or Iteration statement.

(Example) Algorithm to find the sum of first 100 integers

Step 1: Start

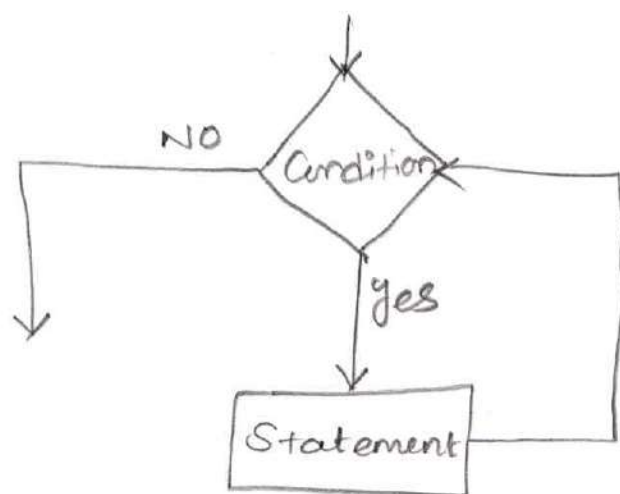
Step 2: Assign $sum = 0$, $i = 0$

Step 3: Calculate $i = i + 1$ & $sum = sum + i$

Step 4: Check whether $i \geq 100$, if no repeat Step 3. Otherwise go to next step.

Step 5: Print the value of sum

Step 6: Stop



Functions

- * A function is a group of statements that is used to perform a specific task.
 - * If a program is large, it is difficult to understand. Hence it is subdivided into smaller programs called subprograms.
 - * Such subprograms are called as functions.
 - * Functions avoid the repetition of some codes over & over. It helps easy debugging, testing & understanding of the program.
 - * Functions reduce program size & the program development time.
- (Example) Add the two numbers using function

def add(x, y)

z = x + y

Print(z)

(21)

```
x = int (input ("Enter a number"))  
y = int (input ("Enter a number"))  
add (x, y)
```

Output

Enter a number 15

Enter a number 35

50

Algorithm

Step 1 : Start

Step 2 : Read the two numbers x, y

Step 3 : Call the function add

Step 3.1 : In function add, add these two numbers & store the result in z .

$$z = x + y$$

Step 3.2 : Print the result

Step 4 : Stop

Suggested Questions / Assignments / Home works / any other

1. Write an algorithm to accept two numbers, compute the sum & print the result. (AU-Dec 22)
2. Define an Iterative Statement (AU-Dec 22)
3. What are the factors used to judge the quality of the algorithm? (AU-Dec 20)
4. What is control flow? List & define the ways of execution of control. (AU-Dec 20)
5. Discuss about building blocks of algorithm (AU-Jan 19)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. ⁰⁵ 048 Notations

Topic(s) to be covered	Notations: Pseudocode - Flowchart - Programming Language
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	understand the Pseudocode, ^{flow} chart	Understand
LO2	know what are guidelines to follow in Pseudocode, flowchart	Remember

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Notations

- * A notation is a system of characters, expressions, graphics or symbols used in problem solving technical facts.
- * It provides best result for a problem.

Example:

Pseudocode, Flowchart

Pseudocode

- * Pseudo means imitates (-false) & code means instruction.
- * Pseudocode is a short, readable & formally styled English language used for explaining an algorithm.
- * It is used to give the structure of the program, before the actual coding.
- * It is not machine readable
- * Pseudocode cannot be compiled or executed. There is no standard for the syntax of pseudocode.

Guidelines for writing Pseudocode

1. Pseudocode is written using structured English
2. The programmer may use their own style for writing the pseudocode, which can be easily understood.
3. It never use the syntax of any programming language.
4. Pseudocode uses some keywords to denote programming process.

Some of them are as follows.

Input: INPUT, GET, READ and PROMPT

Output: OUTPUT, PRINT, DISPLAY and SHOW

Processing : COMPUTE, CALCULATE, DETERMINE,
ADD, SUBTRACT, MULTIPLY and
DIVIDE

Initialize : SET, INITIALISE and ASSIGN

Incrementing : INCREMENT

5. The keywords should be capitalized

6. There are 3 control structures used in
Pseudocode

1) Sequence Control Structure

2) Selection Control Structure

3) Iteration Control Structure

Sequence Control Structures

* In Sequence Control Structure, the statements are executed one after another in the same order as they are written.

* The statements in sequence control structure are executed exactly once.

Example : Find the product of any two numbers

START.

READ values of a & b

COMPUTE c by multiplying a & b

PRINT the result c

STOP.

Selection Control Structure

* In Selection Control Structure, The Condition is tested, and If the Condition is true, one Set of Statements are executed.

* If The Condition is false, an alternative Set of Statements are executed.

There are 2 main Selection Structures

1) IF-THEN-ELSE Statement

2) CASE Statement

IF-THEN-ELSE Statement

* In IF-THEN-ELSE, If The Condition is true, The THEN Part is executed. Otherwise, The ELSE Part is executed.

Syntax

IF Condition THEN

Statement 1

ELSE

Statement 2

ENDIF

Here, The IF-THEN-ELSE Structure Specifies that, If the Condition is true, It executes Statement 1. If the Condition is false, It executes Statement 2.

Example: Find positive or negative number

START

READ a number x

IF x is greater than zero THEN

PRINT x is Positive

ELSE

PRINT x is Negative

ENDIF

Stop

CASE Statement

*The CASE Statement is used, when many numbers of conditions to be checked.

*In CASE Statement, depending on the expression, one of the condition is true.

*Based on the value, the corresponding statements are executed.

*If no match for the expression, then the default option is executed.

Syntax

CASE expression

Condition 1: Statement 1

Condition 2: Statement 2

:

Condition N: Statement N

default

Statement
END CASE

Example: Perform arithmetic operation

START

READ a, b

READ choice

CASE choice

+ : PRINT a+b

- : PRINT a-b

* : PRINT a*b

/ : PRINT a/b

default

PRINT a, b

END CASE

STOP

Iterative Control Structures

- * In Iterative Control Structure, a set of statements are repetitively executed based on a condition.
- * If a condition is true, the set of statements are executed again and again.
- * As soon as the condition becomes false, the repetition stops.
- * This is also known as looping or iterative statement.
- * There are 3 iterative control structures

i) WHILE ii) DO-WHILE iii) FOR

WHILE

*In WHILE loop, The condition is executed at the beginning of the loop.

*If the condition is false then the loop will not be executed.

Syntax

```
WHILE Condition
    Statements
ENDWHILE
```

Example

```
START
INITIALISE n=1
WHILE (n is less than
or equal to 100)
    PRINT n
    INCREMENT n by 1
ENDWHILE
STOP
```

DO-WHILE

*In DO-WHILE loop, The condition is executed at the end of the loop, so the loop is executed at least once irrespective of the loop.

Syntax

```
DO
    Statements
WHILE Condition
END DO
```

Example

```
START
INITIALISE n=1
DO
    PRINT n
WHILE n <= 100
    INCREMENT n by 1
END DO
STOP
```

FOR

* In FOR loop, repeat the execution of statements for specified number of times.

Syntax

```
For iteration bounds
  Statements
ENDFOR
```

Example

```
START
FOR n = 1 to 100
  PRINT n
ENDFOR
STOP
```

Advantages of Pseudocode

- * A Pseudocode is closer to the programming code. Thus it can be easily converted into actual program.
- * Writing Pseudocode is much easier in comparison with drawing a flowchart.

Limitations of Pseudocode

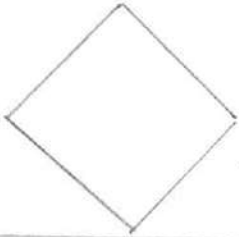
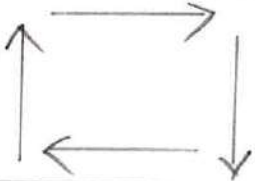
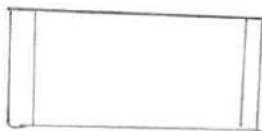
- * It is difficult to understand the program logic as compared to algorithm & flow chart.
- * As there are no specific standards for writing Pseudocode. It becomes hard to maintain consistency.

Flowchart

* A flowchart is a diagrammatic / pictorial representation of the program logic.

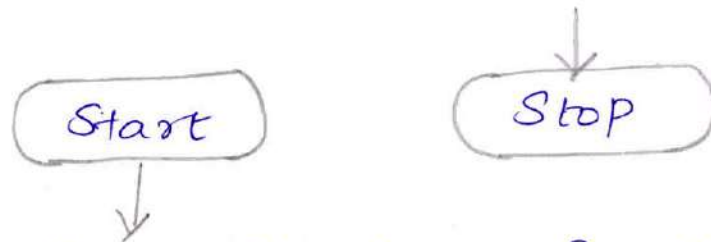
Flowchart Symbols

* A flowchart is drawn using different kinds of symbols. Every symbol used in a flowchart is specific purpose.

Symbol	Symbol Name	Description
	Terminator	Used for Start & Stop
	Data	Used for Input & output
	Process	Operation (or) Action
	Decision	Decision (or) branch
	Flow lines	Indicates the direction of flow
	Connector	Join flow lines
	Predefined Process	Process previously specified

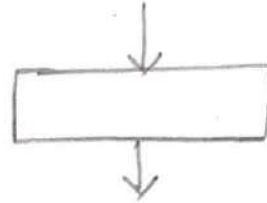
Guidelines for drawing a flowchart

- 1) There can be only one Start & Stop Symbol
- 2) only one flow line can be used with The Start & Stop Symbol.

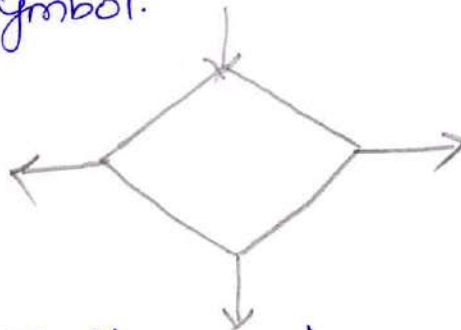


- 3) Direction of flow of information in a flow Chart must be from top to bottom or from left to right

- 4) only one flow line should come out from a Process Symbol



- 5) only one flow line should enter a decision Symbol, but 2 or 3 flow lines leave the decision Symbol.

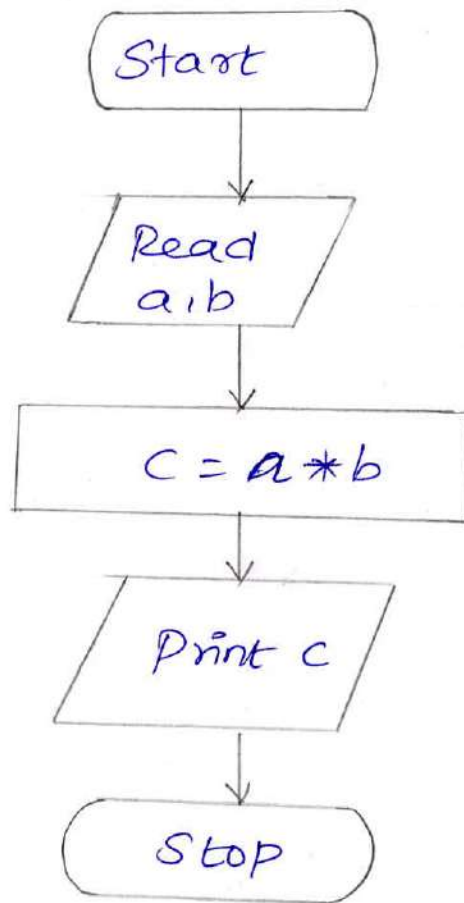


- 6) If the flow Chart becomes complex, it is better to use Connector Symbols to reduce the number of flow lines

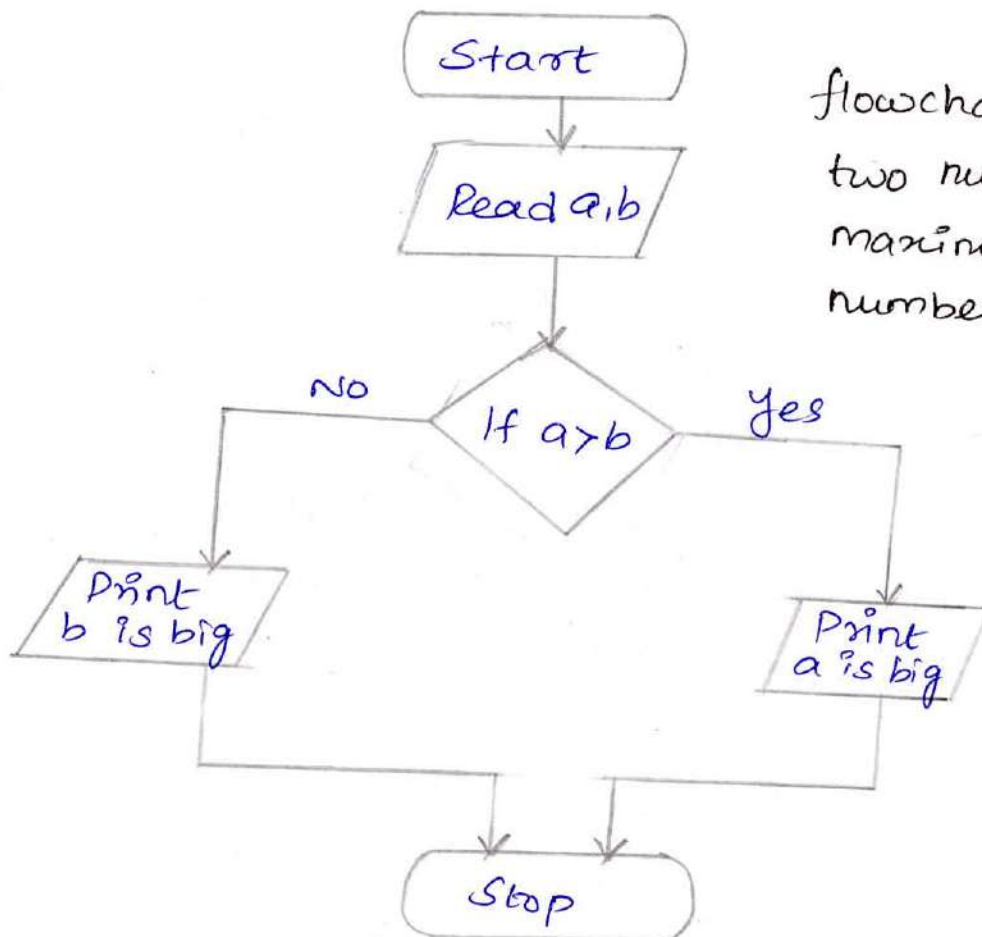
- 7) The Flow lines should not cross each other.

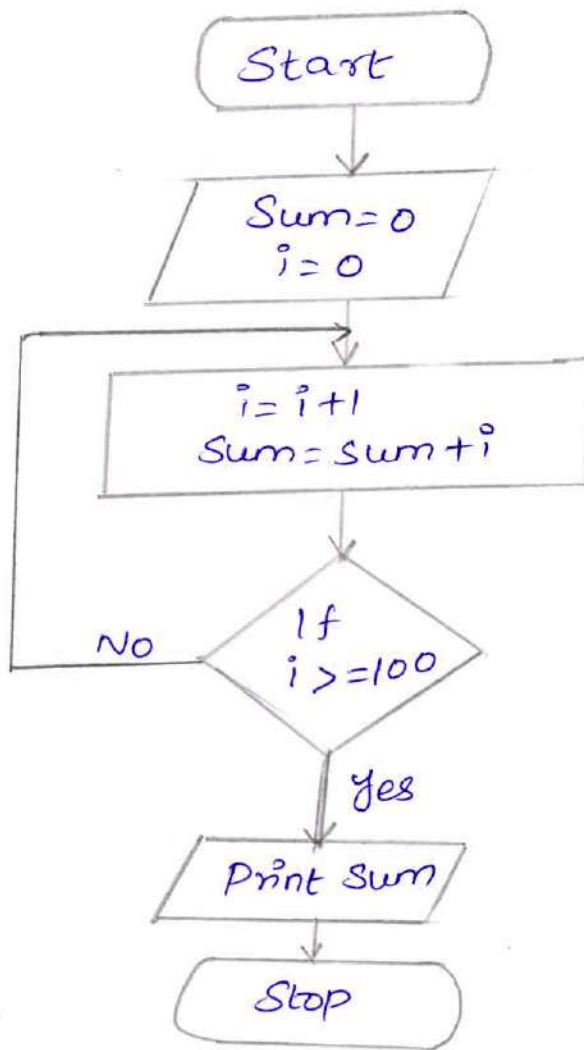
Sequence Control Structure

flowchart computes the product of any two numbers & gives the result.

Selection Control Structure

flowchart compares two numbers & finds maximum of two numbers.



Iterative Control Structure

flowchart finds The
Sum of first 100
Integers (1+2+3...+100)

Advantages of flowchart

- 1) The symbols used in flowchart are self-explanatory. This makes a flowchart easier to understand.
- 2) A Problem can be analyzed in an effective way.
- 3) flowchart better communicate the problem solving logic to the users.
- 4) Helps in monitoring, data collection & identifies the areas for improvement or increase in efficiency.

Limitations of flowchart

- 1) In case of large programs, the flowchart may continue too many pages. This makes difficult to understand.
- 2) It takes a lot of time to represent a program diagrammatically.

Programming Language

* A Programming language Consist of Vocabulary Containing a Set of grammatical rules intended to Convey instructions to a Computer or Computing device to Perform specific task.

* Each programming language has unique set of keywords along with a special Syntax to organise the Software instructions.

* There are 2 types of Programming languages

i) low level language

ii) High level language

Low level language

* Low level language include assembly & machine languages.

→ Machine-level language

* It is a language that consist of Set of instructions that are in the binary form 0 or 1.

* It Cannot easily understood by humans. It doesn't require any translator as the machine code is directly executed by the Computer.

* A machine level language is not portable as each Computer has its machine instructions so if we write a program in one Computer will not longer be valid in another Computer.

* It is the first generation programming language.

Assembly language

- * The assembly language contains some human readable commands such as MOV, ADD, SUB etc.
- * So it is easier to understand & write.
- * In assembly language, the assembler is used to convert the assembly code into machine code.
- * It is the second generation programming language.

High-level language

- * The high level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer.
- * The high-level languages are considered as high level because they are closer to human languages than machine-level languages.
- * A compiler is required to translate a high level language into a low level language.
- * BASIC, COBOL, FORTRAN, JAVA are examples of high level language.

Suggested Questions / Assignments / Home works / any other

1. Draw a flow chart to accept 3 distinct numbers, find the greatest & print the result.
2. Draw a flow chart to find the sum of the Series. $1+2+3+4+5+\dots+100$ (AU- Nov 2022)
3. What is Notations?
4. Define Pseudocode.
5. Difference between while & Do while.
6. Differences between algorithm, flowchart & Pseudocode.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 06 Algorithmic Problem Solving

Topic(s) to be covered	Algorithmic Problem Solving-obtain the description of Problem, Analyze the Problem, Develop & Refine & Review algorithm.
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Describe the techniques & steps for solving problem using algorithm	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Algorithmic Problem Solving

- * Computers are used for solving various day-to-day problems.
- * It is important that computers themselves cannot solve a problem.
- * Precise step-by-step instructions should be given by us to solve the problem.
- * The success of a computer in solving a problem depends on how correctly & precisely define

The problem, design a solution (algorithm) & implement the solution (program) using a programming language.

* Key steps required for solving a problem using a computer are,

Step 1: Obtain a description of the problem

Step 2: Analyze the problem

Step 3: Develop a high level algorithm

Step 4: Refine the algorithm by adding more details

Step 5: Review the algorithm

Step 1: Obtain a description of the problem

* To solve a problem, first we must state the problem clearly & precisely.

* A problem is specified by the given input and the desired output.

* To design an algorithm for solving a problem, we should know the properties of the given input & output.

* A problem description suffers from one or more following types of defects

- * The description may rely on unstated assumptions

- * The description may be ambiguous

- * The description may be incomplete

- * The description may have internal contradictions.

Step 2: Analyze the Problem

- * We need to read & analyse the problem statement carefully in order to list the principal components of the problem.

- * By analyzing a problem, we would be able to figure out what are the inputs that our program accept & output that it should produce.

- * Asking the following questions often helps to determine the starting point.

- * what data are available?

- * where is that data from?

- * what are the rules exist for working with the data?

- * what are the relationship exist among the data values?

Step 3: Develop high-level algorithm

(41)

- * It is essential part before writing a program for given problem.
- * The solution is represented in natural language & is called an algorithm.
- * we start with a tentative solution plan & keeps on refining the algorithm until the algorithm is able to capture all aspects of the desired solution.
- * For a given problem, more than one algorithm is possible & we have to select the most suitable solution.

Step 4: Refine the algorithm by adding more detail

- * A high level algorithm shows the major steps that need to be solve a problem
- * our goal is to develop algorithms that will lead to computer programs.
- * Consider the capabilities of the computer & provide enough details so that someone else use the algorithm to write the computer program

For larger, more complex problems, it is common to go through this process several times.

- * Each time more details are added to the previous algorithm, for further refinement.
- * Step wise refinement is a process of developing a detailed algorithm by gradually adding details to a high-level algorithm.

Step 5: Review the algorithm

- * The final step is to review the algorithm
- * First work through the algorithm step by step to determine, it will solve the problem
- * Asking these questions & seeking their answer is a good way to develop skills that can be applied to the next problem.
- * Does this algorithm solve a very specific problem or does it solve a more general problem?
- * If it solves a very specific problem, should it be generalized?

* For example, an algorithm that computes the area of circle having radius 5.2 meters (formula $\pi * 5.2^2$) solves a very specific problem.

* But an algorithm that computes the area of any circle (formula $\pi * r^2$) solves a more general problem.

* one formula for computing the Perimeter of a rectangle is

$$\text{length} + \text{width} + \text{length} + \text{width}$$

A simple formula would be

$$2(\text{length} + \text{width})$$

Once an algorithm is developed, translate it into a computer program in some programming language.

Suggested Questions / Assignments / Home works / any other

1. Explain The Process of algorithm design & analysis. (AU- Dec 2020)
2. How will you analyze the efficiency of an algorithm? (AU- Dec 2019)

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 07 Strategies

Topic(s) to be covered	Simple Strategies for developing algorithms - Iteration, Recursion
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know what are the strategies for developing algorithm	Understand
LO2	Understand Iteration, Recursion	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Simple Strategies for Developing Algorithms

- * There are different strategies for developing algorithms. Some of the simple strategies are
- i) Iteration
 - ii) Recursion

Iteration

- * In Iteration Control flow a set of statements are repetitively executed based upon a condition.
- * If a condition is true, the set of statements is executed again & again.

* As soon as the Condition becomes false, the repetition stops. This is known as looping Statement or Iteration Statement.

* In general, loops are classified as

i) Counter - Controlled loops

ii) Sentinel - Controlled loops

Counter - Controlled loops

* Counter - Controlled looping is a form of looping in which the number of iterations to be performed is known in advance.

* Counter Controlled loops use a control variable known as the loop Counter, to keep track of loop iterations.

* It is also known as definite repetition loops.

Example Find the Sum of first 100 integers

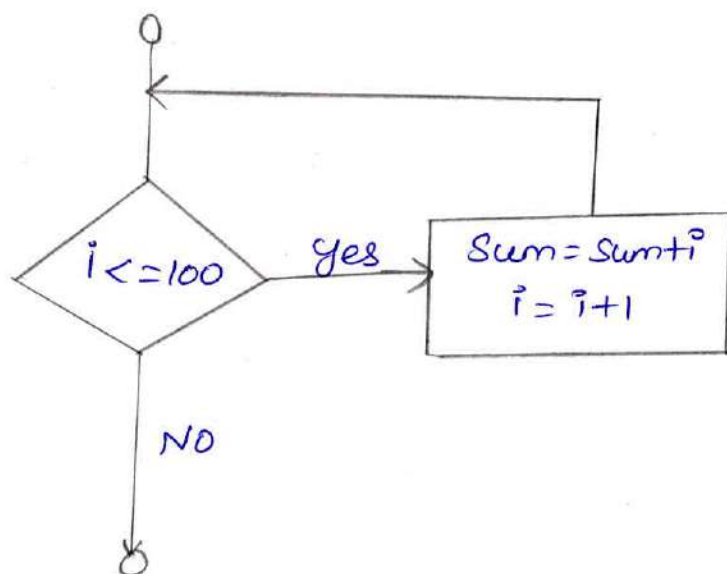
Step 1: Start

Step 2: Assign $i = 0$, $Sum = 0$

Step 3: Check whether $i \leq 100$, If true
Calculate $i = i + 1$ & $Sum = Sum + i$
Otherwise goto next Step

Step 4: Print the value of Sum

Step 5: Stop



Sentinel - Controlled loops

* In Sentinel Controlled looping, The number of times the iteration is to be performed is not known beforehand.

* The execution or termination of the loop depends upon a special value called the Sentinel Value.

* It is also known as indefinite repetition loop

Example Find the Sum of first N integers

Step 1: Start

Step 2: Read N

Step 3: Assign $Sum = 0$, $i = 0$

Step 4: Check whether $i \leq N$, If true

Calculate $i = i + 1$ & $Sum = Sum + i$

Otherwise goto next step

Step 5: Print The Value of Sum

Step 6: Stop.

Recursion

* In recursive technique, a function (or) procedure is called by itself again & again until a given condition is satisfied.

Example Recursive algorithm for finding the factorial of a number

Normally $n! = 1 \times 2 \times 3 \times 4 \dots \times n$ (where 'n' is an integer)

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

Step 1: Start

Step 2: Read number n

Step 3: Call factorial(n)

Step 4: Print factorial f

Step 5: Stop

Factorial(n)

Step 1: If $n == 1$ then
return 1

Step 2: Else

$$f = n * \text{factorial}(n-1)$$

Step 3: Return f

Suggested Questions / Assignments / Home works / any other

1. what is recursive function? what are its advantages & disadvantages? Compare it with iterative function. (AU-Jan 2019)
2. Identify the simple strategies for developing an algorithm. (AU-Jan 2019)
3. Write an algorithm to compute the factorial of a number n . (AU-Jan 2012)
4. Define an Iterative Statement.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 08 Illustrative Problems

Topic(s) to be covered	Illustrative Problems- Finding minimum Number in a list- Inserting a Card in a List of Sorted Cards.
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand how to find minimum number, inserting card in a list of sorted cards	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Finding Minimum Number in a List

Problem Statement: Given a list of numbers, return the minimum number on the list.

Algorithm

Step 1: Start The Program

Step 2: Read the list of numbers x.

Step 3: Assign min with the first element of the list $min = x[0]$

Step 4: for $i = 1$ to x

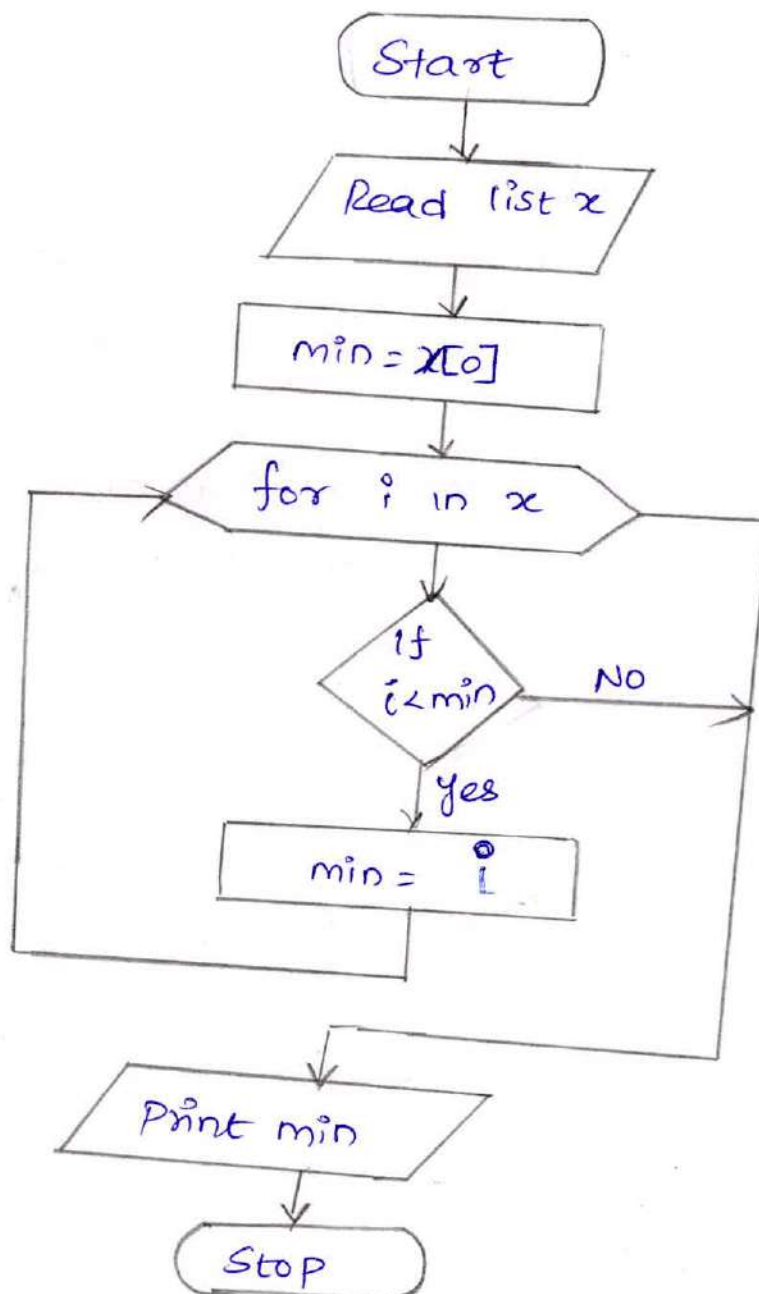
Step 5: If $x[i] < \min$ then
 $\min = x[i]$

Step 6: Repeat the Step 4 & Step 5 until loop
 reached x .

Step 7: Print $\min$

Step 8: Stop

Flow Chart



Pseudo code

START

READ The list of numbers

INITIALISE min with the first element of the list

FOR each element in the list

IF element is less than min

Copy element to min

ENDIF

ENDFOR

PRINT min as the minimum value

STOP

Insert a card in a list of sorted cardsProblem Statement

Imagine that you are playing a card game. You are holding the cards in your hand & these cards are sorted. You are given exactly one new card. You have to put it into the correct place so that the cards you are holding are still sorted.

Algorithm

Step 1: Start

Step 2: Read list of cards A, n Step 3: Read new card k

Step 4: Compare newCard with each element in the list

Step 5: If $\text{newCard} < A[i]$ Then
assign i to Pos, return Pos.

Step 6: From $n-1$ to Pos move the elements one position down

Step 7: Now store the new Card in Pos location.

Step 8: Stop.

Pseudocode

START

READ list of cards, new Card, n

COMPARE newCard with each element in the list

IF $\text{newCard} < A[i]$ THEN

ASSIGN i to Pos

return Pos

ENDIF

FOR $i = n-1; i \geq \text{Pos}; i--$

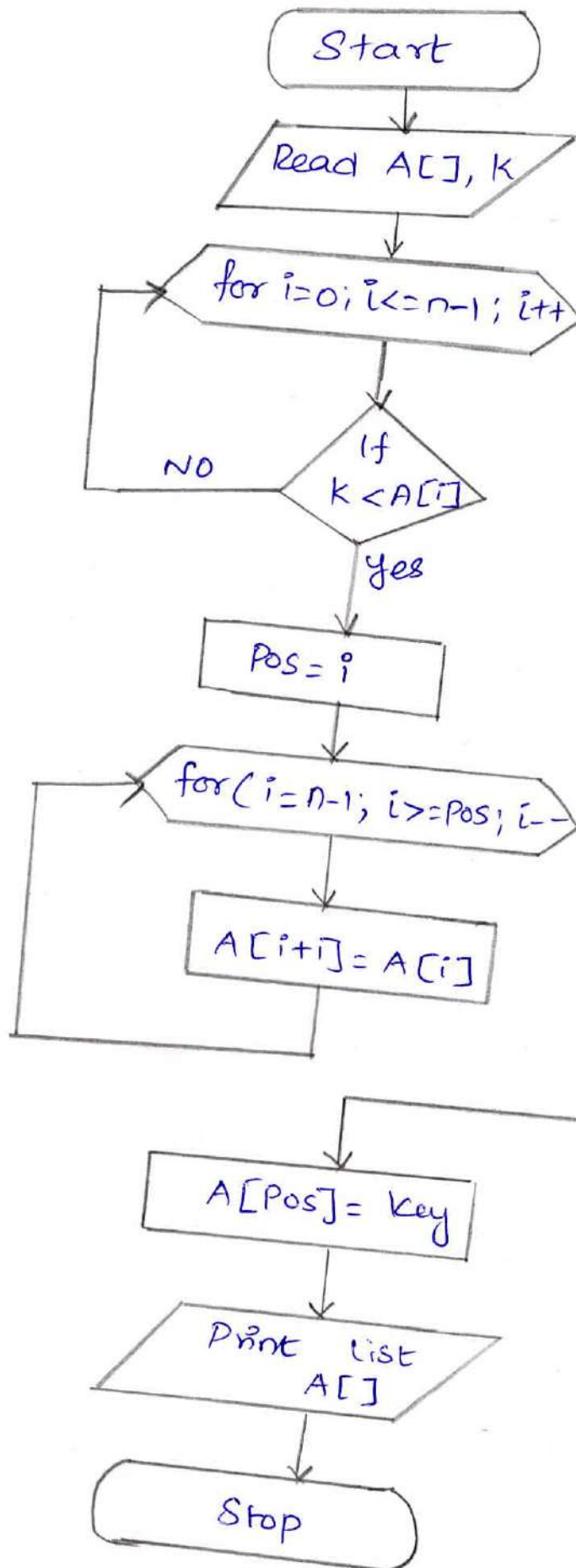
ASSIGN $A[i+1] = A[i]$

ENDFOR

ASSIGN $A[\text{Pos}] = \text{newCard}$

STOP

Flow Chart



0 1 2 3 4
 $A[] = 4 \quad 5 \quad 6 \quad 8 \quad 11$
 $K = 7$

$i=0 \quad 7 < 4$ false

$i=1 \quad 7 < 5$ false

$i=2 \quad 7 < 6$ false

$i=3 \quad 7 < 8$ true

$Pos = 3$

$i=4 \quad 4 >= 3$

$A[5] = A[4]$

$i=3 \quad 3 >= 3$
 $A[4] = A[3]$

$i=2 \quad 2 >= 3$ false

$A[3] = 7$

4 5 6 7 8 11

Suggested Questions / Assignments / Home works / any other

1. Write an algorithm to insert a card into a list of sorted cards (AU-Jan 19)
2. Draw a flowchart for finding minimum number in a list of numbers.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 09 Illustrative Problems

Topic(s) to be covered	Illustrative Problems - Guessing an Integer Number in a Range - Tower of Hanoi Problem
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	understand guessing an integer number, Tower of Hanoi problem	understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Guessing an Integer Number in a Range

Problem Statement

- * The objective is to randomly generate integer number from 0 to n.
- * Then the Player has to guess the number. If the Player guesses the number correctly output an appropriate message.
- * If the guessed number is less than the random number generated, output is "Your guess is lower than the number. Guess again."

(57)

Otherwise output is "your guess is higher than the number. Guess again."

*Then the player guesses another number. This process is repeated until the player guesses the correct number.

Algorithm

Step 1: Start

Step 2: Generate a random number &
read num

Step 3: Read the number to guess

Step 3.1: If guess is equal to num

Print "you guess the correct number"

Step 3.2: If guess is less than num

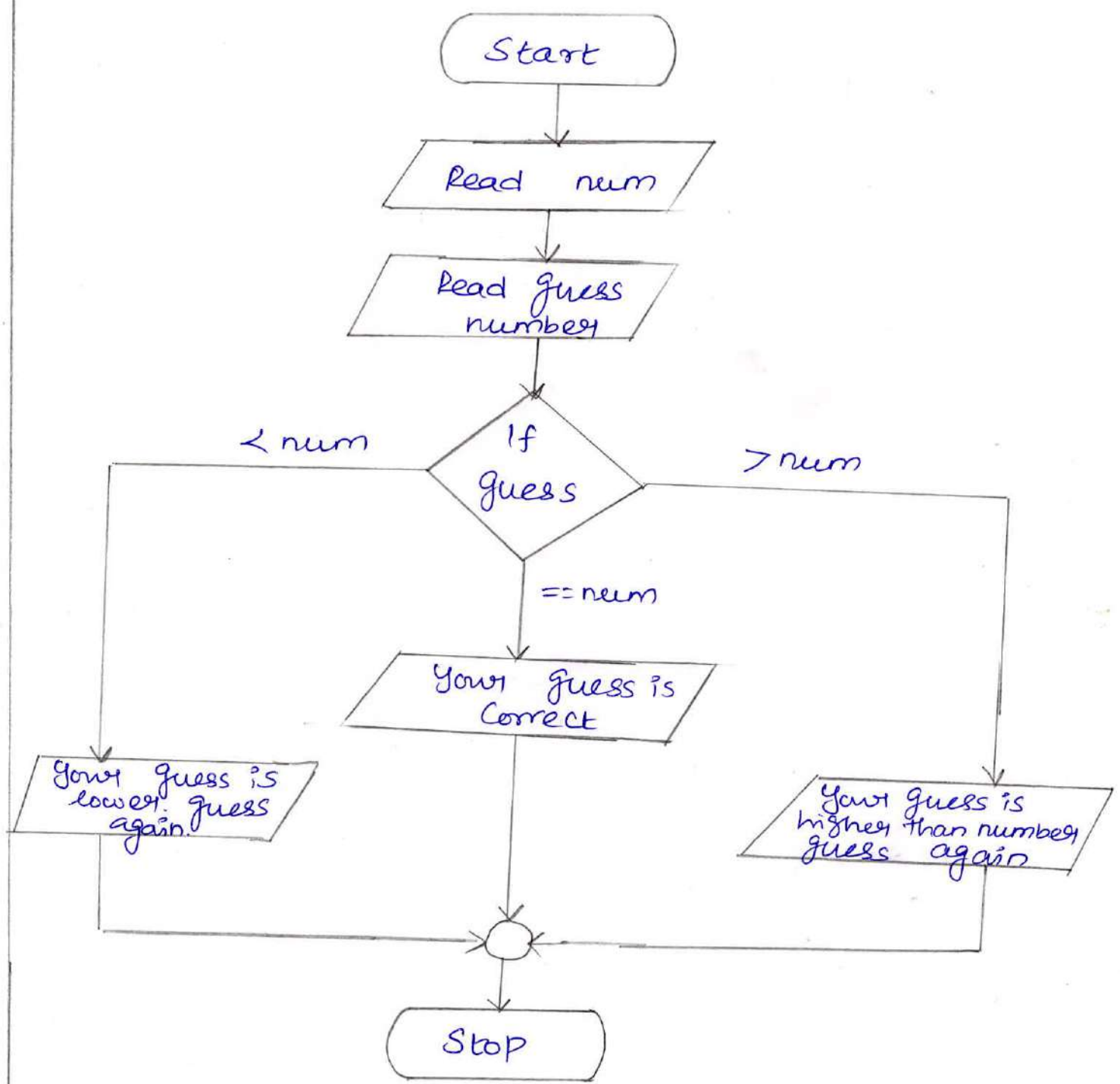
Print "your guess is lower than the number. Guess again."

Step 3.3: If guess is higher than num

Print "your guess is higher than the number. Guess again!"

Step 4: Stop

Flow Chart



Pseudocode

START
Generate random number and
READ num
READ the number to guess

IF $guess == num$

PRINT "You guess the correct number"

ELIF $guess < num$

PRINT "Your guess is lower than the number". Guess again"

ELSE

PRINT "Your guess is higher than the number. Guess again"

ENDIF

STOP

Tower of Hanoi Problem

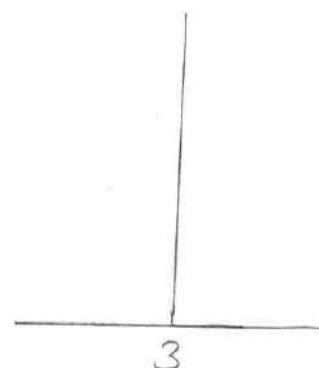
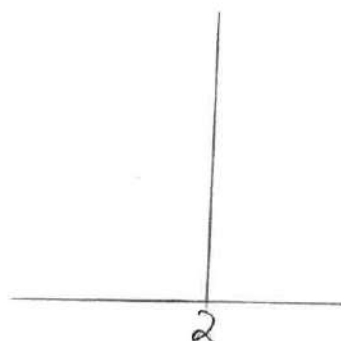
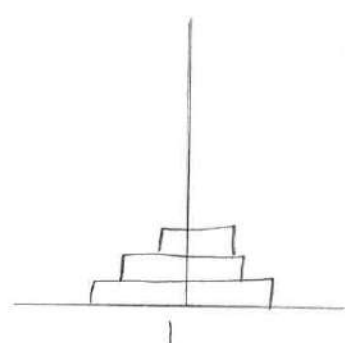
Problem Statement

Tower of Hanoi is one of the classical problem in computer science. The problem states that,

1) There are 3 stands (stand 1, 2 & 3) on which a set of disks, each with a different diameter, are placed.

2) Initially, the disks are placed on stand 1, in order of size, with the largest disk at the bottom.

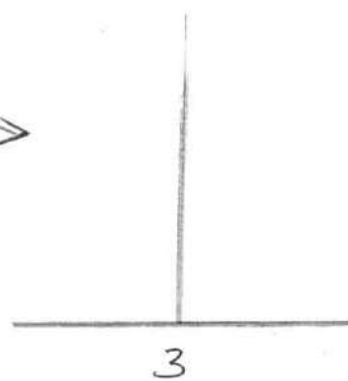
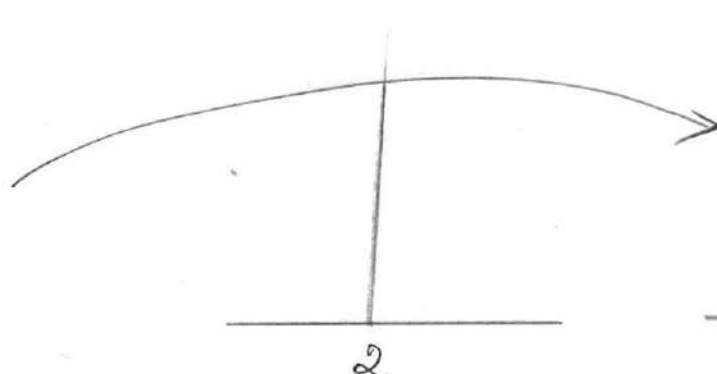
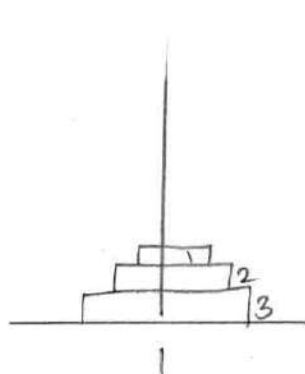
The Initial Structure of Tower of Hanoi with three disks



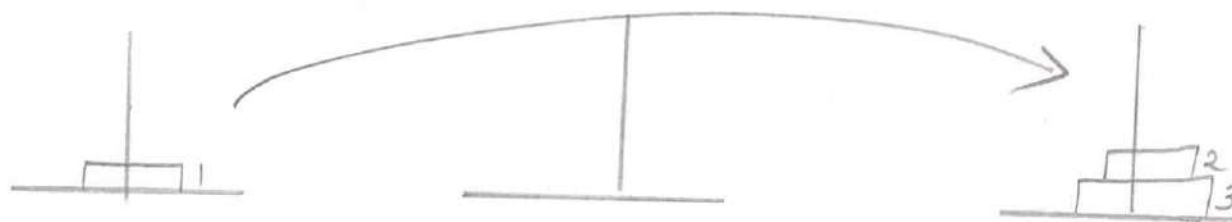
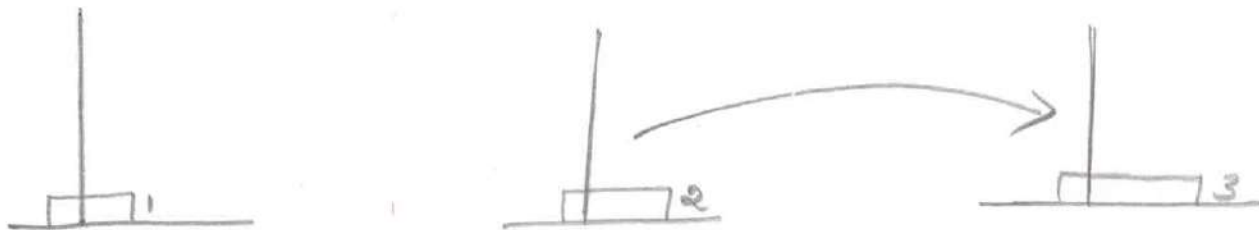
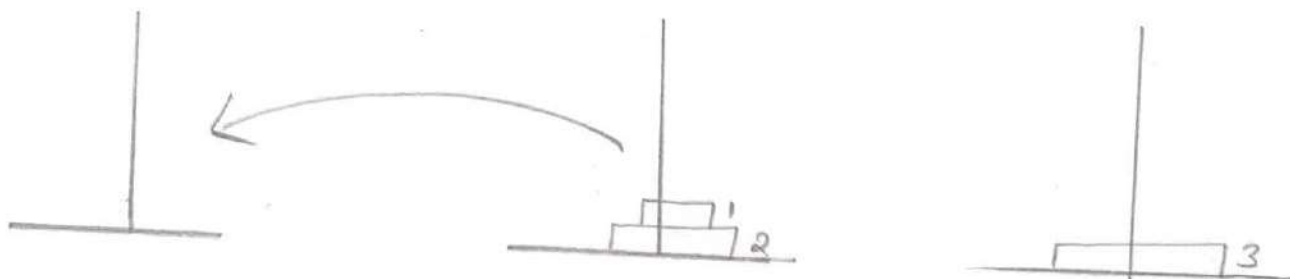
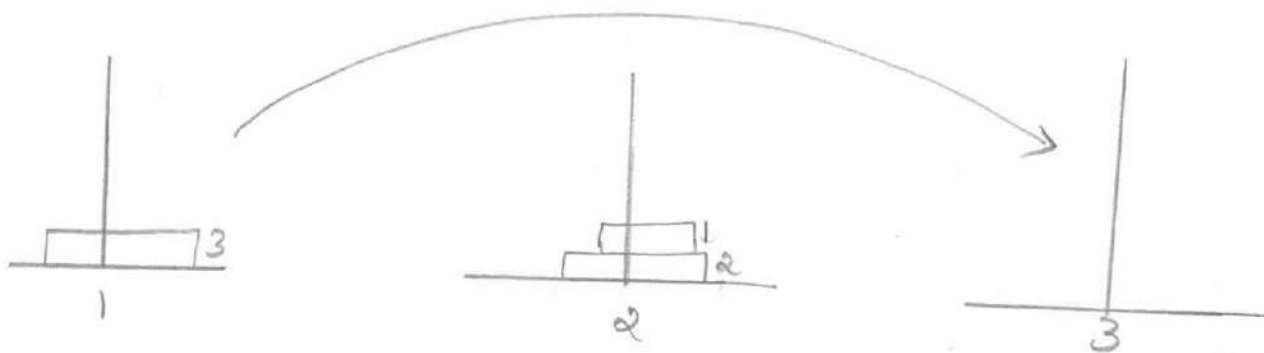
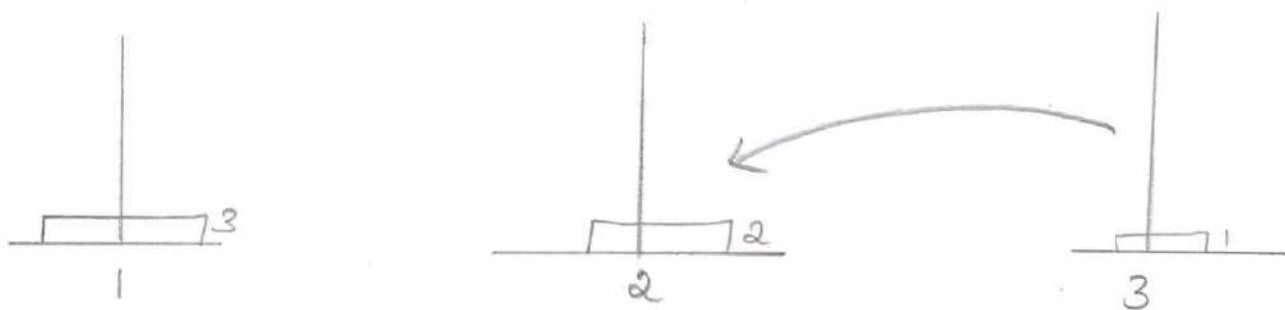
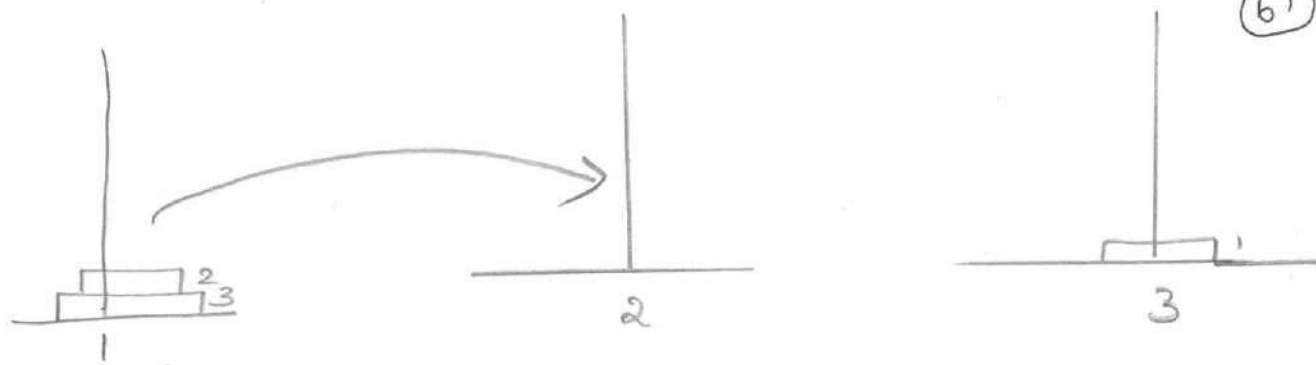
The "Tower of Hanoi" Problem is to find a sequence of disk moves so that all the disks moved from Stand-1 to Stand-3, using the following rules.

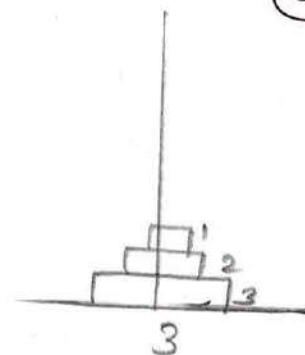
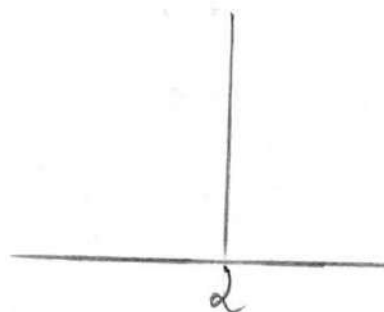
- i) Move only one disk at a time
- ii) A larger disk cannot be placed on top of a smaller disk.
- iii) All disks except the one being moved should be on a Stand.

The General Strategy for Solving the Tower of Hanoi Problem with n disks



(61)





Algorithm

Step 1: Start

Step 2: Procedure Hanoi(disk, source, dest, aux)

Step 3: If only one disk is there then move disk from source to dest

Step 4: If more than one disk is there then

Step 4.1: Recursively call Hanoi(disk-1, source, aux, dest)

Step 4.2: Move disk from source to dest

Step 4.3: Recursively call Hanoi(disk-1, aux, dest, source)

Step 5: Stop.

Pseudo Code

START

Procedure Hanoi(disk, source, dest, aux)

IF disk == 1 THEN

 move disk from source to dest

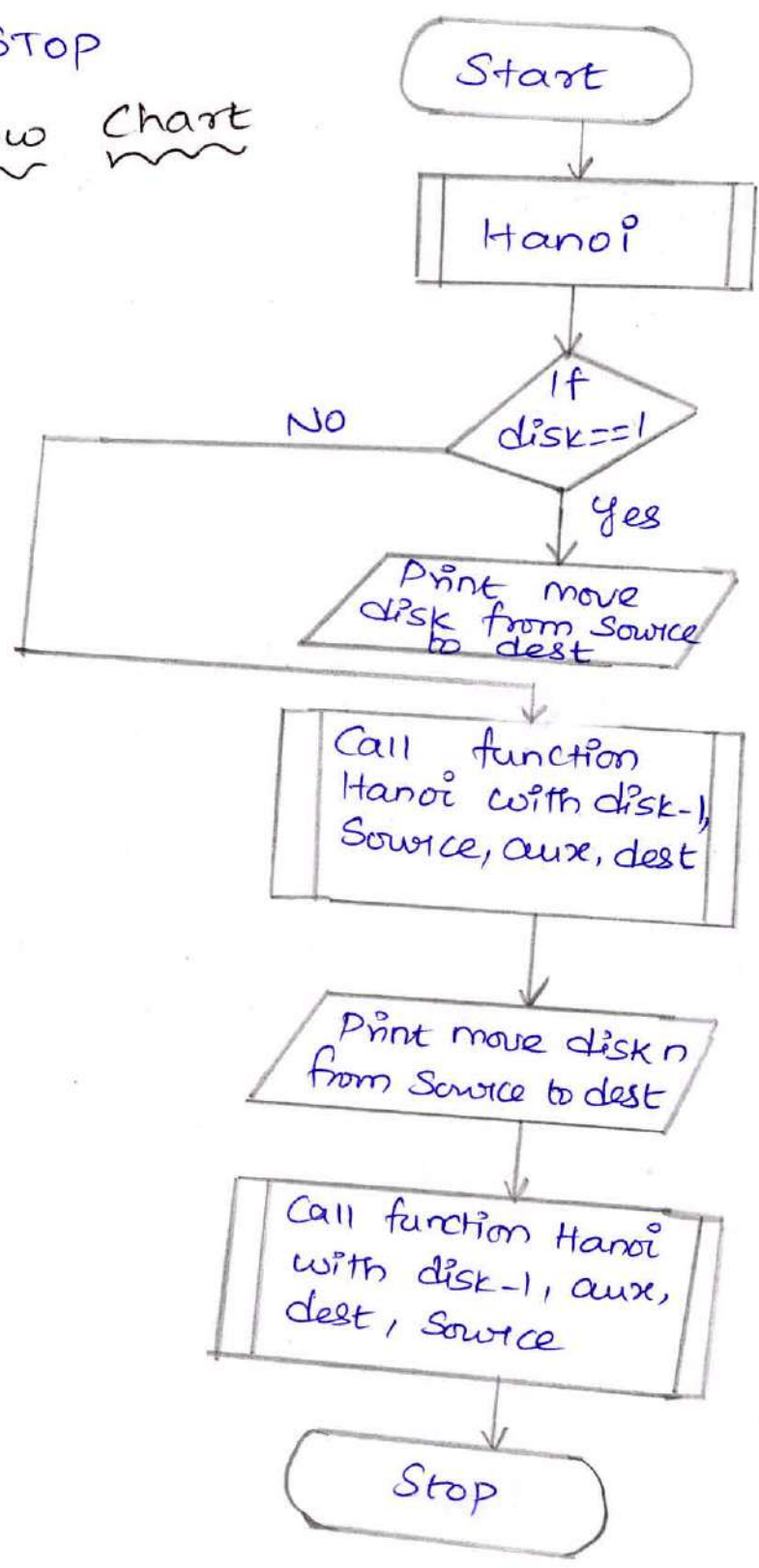
ELSE

Hanoi (disk-1, Source, aux, dest)
move disk from Source to dest
Hanoi (disk-1, aux, dest, Source)
ENDIF

END PROCEDURE

STOP

Flow Chart



Suggested Questions / Assignments / Home works / any other

1. State The Tower of Hanoi Problem.
Outline a Solution to the Tower of Hanoi Problem with relevant diagrams.
(AU-Dec 2022)
2. Write algorithm for Towers of Hanoi Problem. (AU-Dec 2020) (AU-Dec 2019)
3. Write an algorithm & Pseudocode for Guessing an Integer number in a range & also draw a flowchart.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 10 Interpreter & Interactive Mode

Topic(s) to be covered	Python Interpreter & Interactive mode. Compiler Vs Interpreter Values - Debugging
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand interpreter & interactive mode, compiler	Understand
LO2	Describe value & debugging	Understand

Teaching Learning Material	Student Activity
Chalk talk	Listen

Lecture Notes

Python Interpreter & Interactive Mode

Python has 2 basic modes: Interpreter and Interactive modes

Interpreter mode is the mode where the scripted & finished ".py" files are run in the Python Interpreter.

Interactive mode is a Command line shell which gives immediate feedback for each statement.

(2)

Python Interpreter Mode

- * In the Interpreter (or) Script Mode, a Python Program (Script) is written in a file where file name has extension ".py".
- * The Python Script is executed by the Python Interpreter.
- * To execute a script, type the file name along with the path at the prompt.

Python Interactive Mode

- * This allows the user to work in an interactive mode. It is a way of using the Python interpreter by executing a Python Commands from the command line with no script.
- * This allows the user to type an expression, and immediately the expression is executed & the result is printed.
- * The ">>>" is the prompt used in the Python Interactive mode which indicates that the prompt is waiting for the Python Command & produces the output immediately.

(Example)

```
>>> 5+7  
12
```

```
>>> print("Hello world")  
Hello world
```


Compiler and Interpreter

Compiler & Interpreter are very similar in Structure. The main difference is that an Interpreter directly executes the instruction in the source programming while Compiler translates those instructions into efficient machine code.

Compiler

- A Compiler takes entire program & converts it into object code which is typically stored in a file. The object code is also referred as binary code & can be directly executed by machine after linking.
- Examples of compiled programming languages are C and C++

Interpreter

- An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.
- Examples of interpreted languages are Perl, Python and matlab.

Differences between Compiler & Interpreter

④

S.No	Compiler	Interpreter
1.	The Compiler takes a Program as a whole & translates it.	Interpreter translates a program Statement by Statement.
2.	Intermediate Code or target Code is generated In Case of a Compiler	Interpreter doesn't Create Intermediate Code.
3.	A Compiler is comparatively faster than Interpreter as the Compiler take the whole program at one go.	Interpreters compile each line of code after the other.
4.	In a Compiler, the Process requires two Steps, in which first Source Code is translated then executed.	Interpreter is one Step Process in which Source Code is compiled & executed at the Same time.
5.	The Compiler is used in Programming languages like C, C++, C#, Scala	Interpreter is used in PHP, Ruby, Python

Values

Values are the basic units of data, like a number or a String that a Program manipulates.

Debugging

- * A Programmer can make mistakes while writing a program & hence the program may not execute or may generate wrong output.
- * The process of identifying & removing such mistakes also known as bugs or errors from a program is called debugging.
- * Errors categorized as
 - i) Syntax errors
 - ii) Logical errors
 - iii) Runtime errors.

Syntax Errors

- * Like other programming languages, Python has its own rules that determine its syntax.
- * The Interpreter, interprets the statements only if it is syntactically correct.
- * If any syntax error is present, the interpreter shows error message & stops the execution.

(Example)

Parentheses must be in pairs, so the expression $(20 + 22)$ is syntactically correct. whereas $(17 + 35$ is not correct, due to absence of right parenthesis.

→ Such errors need to be removed before the execution of the program

Logical Errors

- * The logical error is a bug in the program that causes it to behave incorrectly.
- * The logical error produces an undesired output but without abrupt termination of the execution of the program.

(Example)

- To find the average of 2 numbers 10 & 12. We write $10 + 12 / 2$ it produces the result 16. Surely, 16 is not the average of 10 & 12.
- The correct code is $(10 + 12) / 2$ to give the correct output as 11.
- Logical errors are also called semantic errors.

Runtime Errors

- A runtime error causes abnormal termination of the program while it is executing.
- Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it.

(Example)

In a statement having division operation in the program, by mistake, if the denominator is zero then it will give a runtime error like "division by zero".

⑦

Suggested Questions / Assignments / Home works / any other

1. Compare Interpreter & Compiler. what type of translator is used for Python?
(AU-dec 19)
2. Sketch the Structure of Interpreter & Compiler. Detail the differences between them. Explain how Python works in Interactive mode & Script mode with examples. (AU-may 2019)

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 11 & 12 Data types

Topic(s) to be covered	Python Data types - Numbers, String, List, Tuple, Dictionary, Boolean, Sets, None
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	know what are the data types supported by Python	Understand
LO2	Describe each type of data	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Python Data types

- * Data type is the type of data, that are going to access within the program.
- * Python program support different data type, each data type may have predefined memory & storage representation.
- * In Python, the standard datatypes available are
 - 1) Numbers 2) String 3) List 4) Tuple
 - 5) Dictionary 6) Boolean 7) Sets 8) None

i) Number Data type

Number data types stores numeric values. It Support 3 number types

i) Integer (10, 20, -20)

ii) Float (3.14, 2.5)

iii) Complex ($7.32 + 2j$)

Integer : It represents signed whole numbers.

⇒ It represents both positive & negative numbers

* To write an integer in decimal (base 10), The first digit must not be zero.

(Eg) 25000

* To write an integer in Octal (base 8), Precede it with O_0 (or) OO & use the digits 0 to 7.

(Eg) O_0177 (Zero followed by either lowercase/uppercase 'o')

* To write an integer in hexadecimal (base 16), Precede with $0x$ or $0X$

(Eg) $0x77$

* To write an integer in binary (base 2), Precede it with $0b$ or $0B$

(Eg) $0b101$

Floating Point

⇒ It represents number with fractional part

⇒ It represents both positive & negative numbers.

(eg) $x = 2.2$ $y = -3.56$

Complex Numbers

Complex numbers are pairs of real & imaginary numbers.

(eg) $a + bj$ where a & b are real numbers
& j is imaginary part

>>> $Z = 5 + 14j$

>>> Z .real

5.0

>>> Z .imag

14.0

>>> print (Z)

(5 + 14j)

ii) String Data type

⇒ String type represents sequence of characters within quotation marks.

⇒ These characters may be alphabets, digits or special characters including spaces.

⇒ It allows either pair of single quotes, double quotes or triple quotes

⇒ When using triple quotes, strings can specify multi line string without using escape character.

(11)

⇒ The quotes are not part of the string, used to mark the beginning & end of the string.

(eg)

Using single quotes: 'HELLO'

Using double quotes: "HELLO"

Using Triple quotes: """ Hello Every one.
welcome to python. """

List Data type

⇒ List is a sequence data type.

⇒ Sequence of items separated by comma & enclosed in square bracket [].

⇒ The sequence of values can be any type.

⇒ The values in list are called elements or items.

(eg)

Address = [231, "KK Nagar", "Trichy", 620020]

⇒ The built-in Python function type can be used to find out type of data

(eg)

>>> type(2)

<class 'int'>

>>> type(3.0)

<class 'float'>

>>> type("Hello")

<class 'str'>

>>> type([1, 1.5, H])

<class 'list'>

Tuples data type

- * Tuple is another Sequence data type similar to list.
- * Tuple consist of Sequence of Items Separated by Comma & enclosed in Parenthesis ().
- * The main differences between list & tuples are
 - 1) Enclosed in brackets [] → list
Enclosed in Parenthesis () → Tuple
 - 2) Elements & size can be changed → list
Elements & size cannot be changed → Tuple

(Sg)

T = ('sun', 1, 'Dec')

Print(T)

('Sun', 1, 'Dec')

Tuples with single Character

T = ('P', 'Y', 'T', 'H', 'O', 'N')

Print(T)

('P', 'Y', 'T', 'H', 'O', 'N')

Dictionary data type

- * Dictionary data type is a kind of hash table
- * It contains key value Pairs.
- * It enclosed by curly braces {}

(eg)

$$d = \{ \text{'name': 'python', 'code': 3171, 'dept': 'CSE'} \}$$

Boolean data type

⇒ A Boolean type represents special values 'True' & False.

⇒ They are represented as 1 & 0.

⇒ The most common way to produce a Boolean value is with a relational operator

(Eg) $2 < 3$ is True

$2 > 3$ is False

Sets

⇒ unordered collection of items separated by commas & items are enclosed in curly brackets $\{ \}$.

⇒ A Set is similar to list, except that it cannot have duplicate entries (every element is unique)

⇒ Once created, elements of a Set cannot be changed.

(Eg) $x = \text{Set} \{ \text{"python programming"} \}$

Print(x)

{ 'p', 'y', 't', 'h', 'o', 'n', 'x', 'g', 'a', 'm', 'i' }

```
>>> type(x)
```

```
<class 'set'>
```

(Eg)

```
a = { 5, 2, 3, 1, 4 }
```

```
Print ("a =", a)
```

```
Print (type(a))
```

Output

```
a = { 1, 2, 3, 4, 5 }
```

```
<class 'set'>
```

None type

- * None is a special data type with a single value
- * None datatype means non-existent, not known or empty
- * None supports no special operations & it is neither false nor zero.

(Eg)

```
drink = "available"
```

```
food = None
```

```
def menu(x):
```

```
    if x == drink:
```

```
        print(drink)
```

```
    else:
```

```
        print(food)
```

```
menu(drink)
```

```
menu(food)
```

Output:

Available

None

In the above program, inside the menu function
If the parameter is drink then, it displays
available.

If the parameter is food, it displays None.

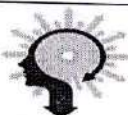
Suggested Questions / Assignments / Home works / any other

1. Outline The data types Supported by Python with an example. (AU-Dec 2022)
2. what are the types used in Python?
3. what is boolean type in Python?
4. How can you Create a Complex literal in Python?

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 13 & 14 Variables, Expressions & Statements

Topic(s) to be covered	Variables, Expressions - Types of Expression, Statements - Tokens
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
Lo1	Understand Variables & Rules	Understand
Lo2	Describe the Expressions & Types	Understand
Lo3	Know the Statements & Tokens	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Variables

- * A variable is an Identifier, which holds a value. In programming, a value is assigned to a variable.
- * Technically, a variable is the name given to the memory location to store values, so that there is no need to remember the address of the location.
- * Variable hold different kind of data & the same variable hold different values during execution of program.

Rules for naming Variables

(18)

- 1) Variable names can be uppercase letters or lowercase letters.
- 2) Variable names can be at any length
- 3) Variable contain both letters & numbers but they can't begin with a number.
- 4) The underscore(-) character can appear in a name. It is often used in names with multiple words
- 5) No blank spaces are allowed between variable names
- 6) The variable name cannot be any keywords or special characters
- 7) The variable names are case sensitive.

(Example)

a, Salary, GrossPay, Roll-no

n=10

name = "Python"

PI = 3.14

7wonders = 'tajmahal' // It is illegal because
it begin with number

Student_name = 'Mano'

a=b=c=d=10

Expressions

- * In Python, most of the lines or statements are written in the form of expressions.
- * Expressions are made up of operators & operands. Expressions are evaluated according to operators.
- * Expression is defined as combination of values, variables & operators.

(Example)

$$A * B + C$$

Where

$*$, $+$ operators

A, B & C are operands

Types of expression

Based on the position of operators in an expression.

- 1) Infix expression: The operator is placed in between the operands. Example: $a = b + c$
- 2) Prefix expression: The operator is placed before the operands. Example $a = +bc$
- 3) Postfix expression: The operator is placed after the operands. Example $a = bc +$

Based on The type of the result obtained on evaluating an expression

i) Arithmetic Expression

* This type of expression just do mathematical operations like $+$, $-$, $*$, $/$ etc

(eg)

num1 = 20

num2 = 30

Sum = num1 + num2

Print (Sum)

Output

The value of Sum is : 50

2) Relational / Conditional Expression

* This expression compares two statements using relational operators like $>$, $<$, $>=$, $<=$ etc.

* The relational expression is also called as Conditional expression.

* This expression used with branching (if, if-else etc) & looping (while) statement

* Expression produce results either True or False depending on Condition.

(Example)

```
Age = Int (Input ("Enter your age"))
```

```
If (Age >= 18):
```

```
    Print "You can vote"
```

```
else:
```

```
    Print "You can't vote"
```

Output

Enter your age 25

You can vote.

3) Logical Expression

*The logical expression uses the logical operators like and, or, not.

*The logical expression also produces a Boolean result like either True or False.

(Example)

```
a = 20
```

```
b = 30
```

```
c = 23
```

```
d = 21
```

```
Flag = ((a > b) and (c > d))
```

```
Print ("The result is", Flag)
```

Output

The result is False

Statements

- * A Statement is an instruction that the Python Interpreter can execute.
- * There are two kinds of statements

Assignment (=)

Print

Assignment Statement

- * Assignment Statement associates the name to the left of the '=' symbol with the value to the right of the '=' symbol.

(eg) Name = 'Jovita'

Age = 9

Simultaneous Assignment Statement

- * Python permits any number of variables to appear on the left side, separated by commas.
- * The same number of expressions must then appear on the right side, again separated by commas.

(Example)

```
>>> x = 10
```

```
>>> y = 5
```

```
>>> Sum, diff, Prod = x + y, x - y, x * y
```

```
>>> Sum
```

```
15
```

```
>>> diff
```

```
5
```

```
>>> Prod
```

```
50
```


Print Statement

- *The Print Statement recognizes series of values separated by commas.
- *Each value is converted into a string (by implicitly invoking the str function) & then printed.

(Example)

```
>>> name = input("Enter your name:")
```

```
Enter your name: Python
```

```
>>> print('Hello', name)
```

```
Hello Python
```

The Print Statement recognizes a few commands used to format input. These are termed 'escape characters' and are written as a character following backslash.

- *The most common escape character is the newline character '\n'. This moves the output to the new line.

- *The tab character, '\t' moves the output to the next tab.

(example)

```
>>> print "One\n two\t three"
```

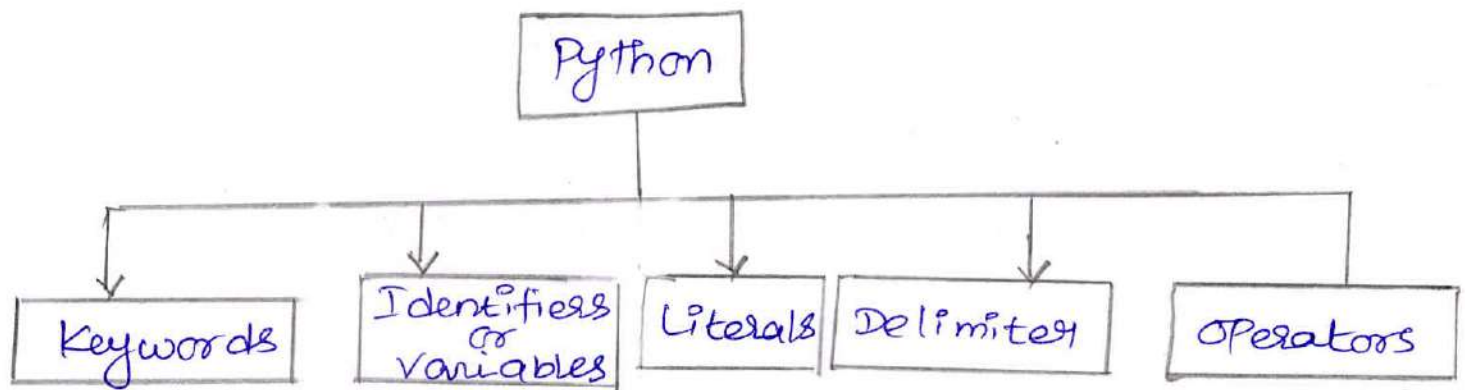
```
One
```

```
two three
```

Tokens

→ Python breaks each logical line to a sequence of elementary lexical components known as "tokens".

→ A token is the smallest unit of the program.



"Tokens in Python"

Keywords

* keywords are reserved words they have predefined meanings in Python.

* They cannot be used as ordinary identifiers

* Python is case sensitive so keywords must be spelled exactly as they are written.

* Some keywords are

if, for, import, break, while, continue
and, print, return, else, def, or, not

Identifiers

- * A Python Identifier is the name given to the Variable, function, class, module or other object.
- * An Identifier can begin with an alphabet (A-Z or a-z) or an underscore (-) and can include any number of digits, letters or underscores.
- * Spaces are not allowed.
- * Python will not accept @, \$ & % as Identifiers.
- * Python is a case-sensitive language. Thus 'Hello' and 'hello' are different Identifiers.

Escape Sequence

- * Escape Sequence are non-Printable Characters. It consists of backslash followed by a character.
- * Some recognized escape sequences are

' → single quote

" → double quote

\n → moves next line

\t → moves next horizontal tab

\v → moves next vertical tab

\b → Backspace

Delimiters

* Delimiters are symbols that perform 3 special roles in Python like grouping, punctuation & assignment

* Example delimiters are

(), { }, [] → grouping

., :, @ → punctuation

= += -= *= /= //= % = → Assignment

Literals

* A literal is a sequence of one or more characters that stands for itself.

* There are 2 important literals

1) Numeric literals 2) String literals

Numeric literals: It contains only the digits 0-9
Optional sign (+ or -) & possible decimal point.

→ If a numeric literal does not contain decimal

point, then it denotes "integer" (10)

→ If a numeric literal contains decimal point, then it denotes "floating point" (12.35)

String literals

→ String literals (or) strings represent a sequence of characters surrounded by quotes.

(Example) 'Hello'

"Python, Programming"

'''231, KK Nagar, 620001'''

Suggested Questions / Assignments / Home works / any other

1. What are the keywords? Give example.
2. Define Escape Sequence.
3. What is Variable?
4. List out the rules to be followed in naming Variable.
5. Define Expression.
6. What are the types of Expressions?

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 15 & 16 Operators

Topic(s) to be covered	Operators - operands - Classification of operators
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand the operators?	
LO2	Know the types of operators	Understand
LO3	Understand the working of operators	

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Operators

*An Operator is a Symbol that represents an operation that may be performed on one or more operands.

Operands: The value that the operator operates on is called the operand.

Example: $x = a + b$

where $x, a, b \rightarrow$ operands

$=, + \rightarrow$ operators

Classification of Operators

(29)

Unary Operators

* The unary operators operate on only one operand

Operator	meaning
+	unary plus
-	unary minus
~	Bitwise Inverse

* The unary Plus & minus operators are used to provide sign to the operands.

(eg) +6, -9

* The bitwise Inverse operator changes every bit
ie) 0 becomes 1 & 1 becomes 0.

Binary Operators

* The binary operators operate on two operands.

* The binary operators in Python are grouped into

i) Arithmetic operators

ii) Relational operators

iii) Logical operators

iv) Bitwise operators

v) Assignment Operators

vi) Special Operators

a) Identity operators

b) Membership operators

Arithmetic operators

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

Operator	Meaning	Example
+	Add two operands	$x + y$
-	Subtract right operand from the left	$x - y$
*	Multiply two operands	$x * y$
/	Divide left operand by the right one (results into float)	x / y
%	Modulus: remainder of the division	$x \% y$
//	Floor division (or) Truncating division: results into whole number	$x // y$
**	Exponent: right operand raised to the Power	$x ** y$

Example

Program

$x = 25$

$y = 5$

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)

Print('x**y=', x**y)

Output

x+y=30

x-y=20

x*y=125

x/y=5.0

x//y=5

x%.y=0

x**y=9765625

Relational Operators

- * The relational operators perform the comparison operations.
- * The result always is a boolean value (True or False).
- * The relational operators are also called comparison operators.
- * There is no $=<$ or $=>$ operators in Python.

Operator	Meaning	Example
$>$	Greater than - True if the left operand is greater than the right	$x > y$
$<$	less than - True if the 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 Program

$x = 10$

$y = 12$

Print (' $x > y$ is', $x > y$)

Print (' $x < y$ is', $x < y$)

Print('x == y is', x == y)

Print('x != y is', x != y)

Print('x >= y is', x >= y)

Print('x <= y is', x <= y)

Output

x > y is False

x < y is True

x == y is False

x != y is True

x >= y is False

x <= y is True

Logical Operators

- * The logical operators are used to logically relate the sub expressions.
- * The logical operators operate according to the truth tables.

Operator	Meaning	Example
and	True. If both operands are true	x and y
or	True. If either of the operands is true	x or y
not	True. If operand is false	not x

Truth table

34

OP1	OP2	OP1 and OP2	OP1 or OP2	not OP1
True	True	True	True	False
True	false	false	True	False
False	True	false	True	True
False	False	false	False	True

Example Program

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)

Output

x and y is false

x or y is True

not x is false

Bitwise operators

* Bitwise operators work with bits of binary number. It operates bit by bit

Operator	Meaning	Example
&	Bitwise AND	x & y
	Bitwise OR	x y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x >> 2
<<	Bitwise left shift	x << 2

(Example program)

a=8

b=64

Print ("Bitwise AND operator on 8 and 64 is =", a & b)

Print ("Bitwise OR operator on 8 and 64 is =", a | b)

Print ("Bitwise NOT operator on 8 is =", ~a)

Print ("Bitwise Exclusive OR operator on 8 and 64 is =", a ^ b)

Print ("Bitwise left shift operator on 8 is =", a << 1)

Print ("Bitwise right shift operator on 64 is =", b >> 1)

Output

Bitwise AND operator on 8 and 64 is = 0

Bitwise OR operator on 8 and 64 is = 72

Bitwise NOT operator on 8 is = -9

Bitwise Exclusive OR operator on 8 and 64 is = 72

Bitwise left shift operator on 8 is = 16

Bitwise right shift operator on 64 is = 32

	128	64	32	16	8	4	2	1
AND								
a → 8	0	0	0	0	1	0	0	0
b → 64	0	1	0	0	0	0	0	0
AND	0	0	0	0	0	0	0	0
OR	0	1	0	0	1	0	0	0
8 << 1	0	0	0	0	0	0	0	0
64 >> 1	0	0	1	0	0	0	0	0

0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0
XOR	0	1	0	0	1	0	0

↑ same → 0
diff → 1

$$\begin{array}{lcl}
 8 \rightarrow 01000 & & 1's \text{ Complement } 1000 \\
 \sim 8 \rightarrow 10111 & \rightarrow & 2's \text{ Complement } 10001 \\
 & & \underline{\underline{1001}}
 \end{array}$$

$\sim 8 = -9$

Assignment Operators

* Assignment operators are used to assign values to the variables.

(Eg) $a = 15$

* There are various Compound assignment operator in Python

Operator	Meaning	Equivalent to
=	$x = 5$	$x = 5$
+=	$x += 5$	$x = x + 5$
-=	$x -= 5$	$x = x - 5$
*=	$x *= 5$	$x = x * 5$
/=	$x /= 5$	$x = x / 5$
%=	$x \% = 5$	$x = x \% 5$
//=	$x //= 5$	$x = x // 5$
**=	$x ** = 5$	$x = x ** 5$
&=	$x \& = 5$	$x = x \& 5$
=	$x = 5$	$x = x 5$
^=	$x \wedge = 5$	$x = x \wedge 5$
>>=	$x >> = 5$	$x = x >> 5$
<<=	$x << = 5$	$x = x << 5$

Special Operators

37

Python language offers some special types of operators

1) Identity operators

2) Membership operators.

1) Identity operators

* Compare the memory location of two objects

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

(Example program)

x1 = 5

y1 = 5

x2 = 'hello'

y2 = 'hello'

x3 = [1, 2, 3]

y3 = [1, 2, 3]

Print (x1 is not y1)

Print (x1 is y1)

Print (x2 is y2)

Print (x3 is y3)

Output

False

True

True

False

Membership operators

(38)

- * Membership operators test for membership in a sequence (String, list, tuple, set & dictionary)
- * In a dictionary, test for the presence of key, not the value.

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

Example Program

```
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)
```

Output

True

false

True

false

Suggested Questions / Assignments / Home works / any other

1. Identify the operand(s) & operator(s) in the following expression
Sum = a + b (AU-Dec 22)
2. Name the types of operators supported by Python & outline any two with an example. (AU-Dec 22)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 17 operator Precedence

Topic(s) to be covered	Precedence of operators (order of operators) - Comments
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know the operator Precedence	understand
LO2	Understand how to represent the Comments in python	understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Precedence of Operators

- * Evaluation of Expression is based on Precedence of operators
- * When an expression contains different kinds of operators, Precedence determines which operator should be applied first.
- * Higher Precedence operator is evaluated before the lower Precedence operator.

* For mathematical operators, Python follows mathematical Convention. The acronym PEMDAS is Useful way to remember the rules.

- i) Parentheses have the highest Precedence & Can be Used to force an expression to evaluate in the order.

Example: $5 * (9 - 3) = 30$

- ii) Exponentiation has the next highest Precedence

Example: $1 + 2 ** 3 = 9$, not 27.

$2 * 3 ** 2 = 18$, not 36.

- iii) Multiplication & Division have higher Precedence than Addition & Subtraction

Example: $2 * 3 - 1 = 5$, not 4

$6 + 4 / 2 = 8$, not 5.

* Operators with the Same Precedence are evaluated from left to right (except exponentiation)

Operator	Description
$**$	Exponentiation (raised to Power)
$\sim + -$	Complement, unary plus & minus
$* / \% //$	Multiply, divide, modulus & floor division
$+ -$	Addition & Subtraction
$>> <<$	Right & left bitwise shift

&	Bitwise AND
^	Bitwise exclusive OR, Bitwise OR
<= <> >=	Comparison Operators
= = !=	Equality Operators
+= -= *= //= %=	Assignment Operators
is, is not	Identity Operators
in, not in	Membership Operators
and, or, not	Logical Operators

Comments

- * A Comment Statement contains information for persons reading the program
- * Comments make the program easily readable & understandable by the programmer and non-programmer who are seeing the code.
- * Comments are non-executable, so ignored during program execution.
- * In Python, a comment starts with # (hash) Everything following the # till end of that line is treated as a comment.

Example

```
# swap.py
# Swapping the values of 2 variable
# Written by D Section First Year
```


Suggested Questions / Assignments / Home works / any other

1. Are Comments executable Statements in a Python Program? How Comments are Included in a Python Program? (AU-Dec 22)
2. How does the order of evaluation of operations happen when an expression contains more than one operator? Explain (AU-Dec 20)
3. Evaluate the following expression.
 i) $24 // 6 \% 3$ ii) $\text{float}(4 + \text{int}(2.39) \% 2)$
 iii) $2 ** 2 ** 3$

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 18 & 19 Illustrative Programs

Topic(s) to be covered	Illustrative Programs - Exchange the values of two variables - Circulate the values of n variables - Find distance
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	write Python program for Exchange the values, Circulate the values, distance finding	Apply

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Exchange the values of two variable

Exchange the values of two variable using temporary variable

```
x = int(input("Enter value of x"))
y = int(input("Enter value of y"))
print("Before exchange of x, y")
print("x =", x)
print("y =", y)
```



```
temp = x
```

```
x = y
```

```
y = temp
```

```
Print("After Exchange of x, y")
```

```
Print("x=", x)
```

```
Print("y=", y)
```

Output

Enter value of x 67

Enter value of y 56

Before Exchange of x, y

x=67

y=56

After exchange of x, y

x=56

y=67

Exchange the values of two variables using function (Using two variables)

```
def exchange(x, y):
```

```
    x, y = y, x
```

```
    Print("After exchange of x, y")
```

```
    Print("x=", x)
```

```
    Print("y=", y)
```

```
x = int(input("Enter value of x"))
```

```
y = int(input("Enter value of y"))
```

Print ("Before exchange of x, y")

Print ("x = ", x)

Print (" y = ", y)

exchange (x, y)

Circulate the values of n variables using Slicing technique

```
def circulate (A, N):
```

```
    for i in range(1, N+1):
```

```
        B = A[i:] + A[:i]
```

```
        Print ("Circulation ", i, "=", B)
```

```
    return
```

```
A = [91, 92, 93, 94, 95]
```

```
N = int(input("Enter N:"))
```

```
circulate (A, N)
```

Circulate the values of n variables without using Slicing technique.

```
def circulate (A, N):
```

```
    for i in range(1, N+1):
```

```
        B = A.pop(0)
```

```
        A.append(B)
```

```
        Print ("Circulation ", i, "=", A)
```

```
    return
```

```
A = [91, 92, 93, 94, 95]
```

```
N = int(input("Enter N:"))
```

```
circulate (A, N)
```


Circulate the values of n variable without using list properties:

```
def circulate(A, n):
    for i in range(1, n+1)
        j = len(A) - 1
        while j > 0:
            tmp = A[j]
            A[j] = A[j-1]
            A[j-1] = tmp
            j = j - 1
        Print ("Circulation", i, "=", A)
```

A = [91, 92, 93, 94, 95]

N = int(input("Enter N:"))

circulate(A, N)

Output

Enter N: 5

Circulation 1 = [92, 93, 94, 95, 91]

Circulation 2 = [93, 94, 95, 91, 92]

Circulation 3 = [94, 95, 91, 92, 93]

Circulation 4 = [95, 91, 92, 93, 94]

Circulation 5 = [91, 92, 93, 94, 95]

Finding distance between two points:

```
import math
```

```
x1 = int(input("Enter x1"))
```

```
y1 = int(input("Enter y1"))
```

```
x2 = int(input("Enter x2"))
```

```
y2 = int(input("Enter y2"))
```

```
distance = math.sqrt((x2 - x1)**2 +  
                      (y2 - y1)**2)
```

```
print("Distance = ", distance)
```

Output

Enter x1 : 3

Enter y1 : 2

Enter x2 : 7

Enter y2 : 8

Distance = 7.21116255

Suggested Questions / Assignments / Home works / any other

1. Python Program to exchange the values of two variable without using a temporary variable. (AU-Dec 2020)
2. write a Python Program to Circulate the values of n variables
3. write a Python Program to find the distance between two points.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 20 Conditionals

Topic(s) to be covered	Conditional Execution - Alternative Execution (if-else) - Chained Conditional Execution (if-elif-else) - Nested Conditional
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Describe the Conditional Statements	Understand
LO2	Understand the types of conditionals	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Conditional Execution

* Python language supports different types of Conditional branching Statements. They are

- Conditional Execution (if)
- Alternative Execution (if-else)
- Chained Conditional Execution (if-elif-else)
- Nested Conditionals.

Conditional Execution (if)

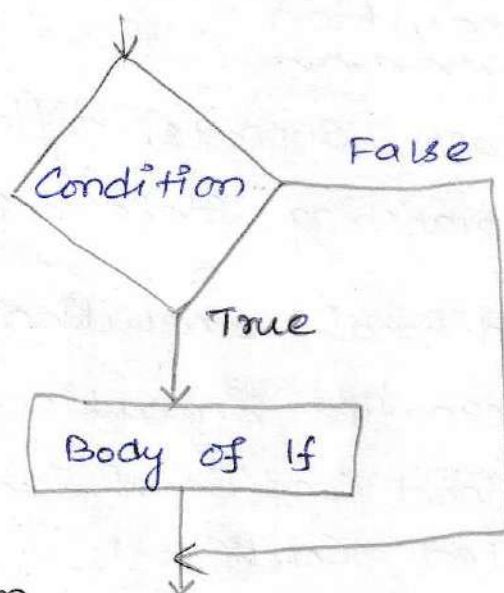
- * If Statement, execute the statement only if the condition is true.
- * If the condition is false then the statements are not executed.

Syntax

```
if Condition :
    Statement(s)
```

- * The Colon (:) is required at the end of the condition.
- * The body of the If Statement is indicated by the indentation.

Flowchart



Example Program

```
num = int(input("Enter number"))
if num > 0:
    print(num, "is a positive number")
```

Output

Enter number 5

5 is a positive number

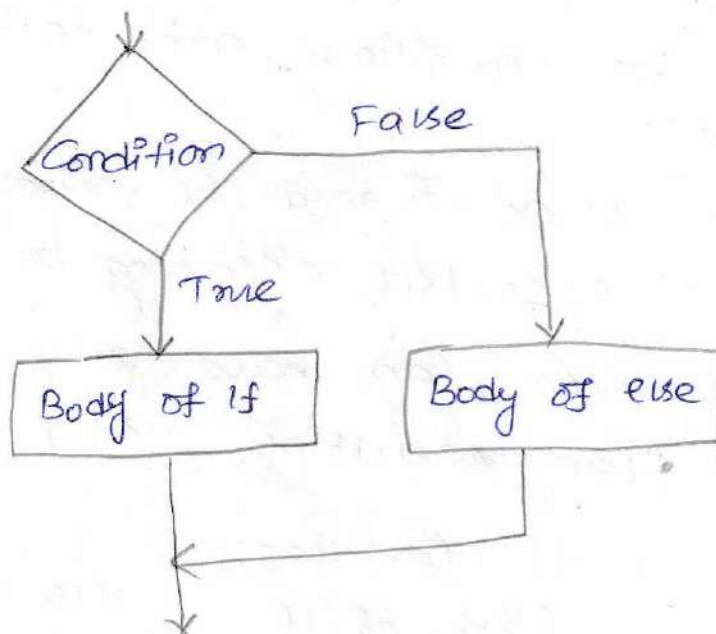
Alternative Execution (if-else)

- * The if-else Statement evaluates the condition & executes the true statement only when the condition is true.
- * If the condition is false, false statement is executed. Indentation is used to separate the blocks.

Syntax

```
If Condition :  
    Body of If  
else :  
    Body of else
```

Flowchart



Example Program

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

```
if num > 0:
```

```
    print(num, "is a positive number")
```

```
else:
```

```
    print(num, "is a negative number")
```

Output

```
Enter number -5
```

```
-5 is a negative number.
```

Chained Conditional Execution (if-elif-else)

- * The elif is short for else if. It allows checking for multiple expressions.
- * If the condition for 'if' is false, it checks the condition of the next elif block & so on.
- * If all the conditions are false, body of else is executed.
- * Only one block among the several if..elif..else block is executed according to the condition.
- * The 'if' block can have only one 'else' block. but it can have multiple elif blocks.

Syntax

```
if Condition:
```

```
    Body of if
```

```
elif Condition:
```

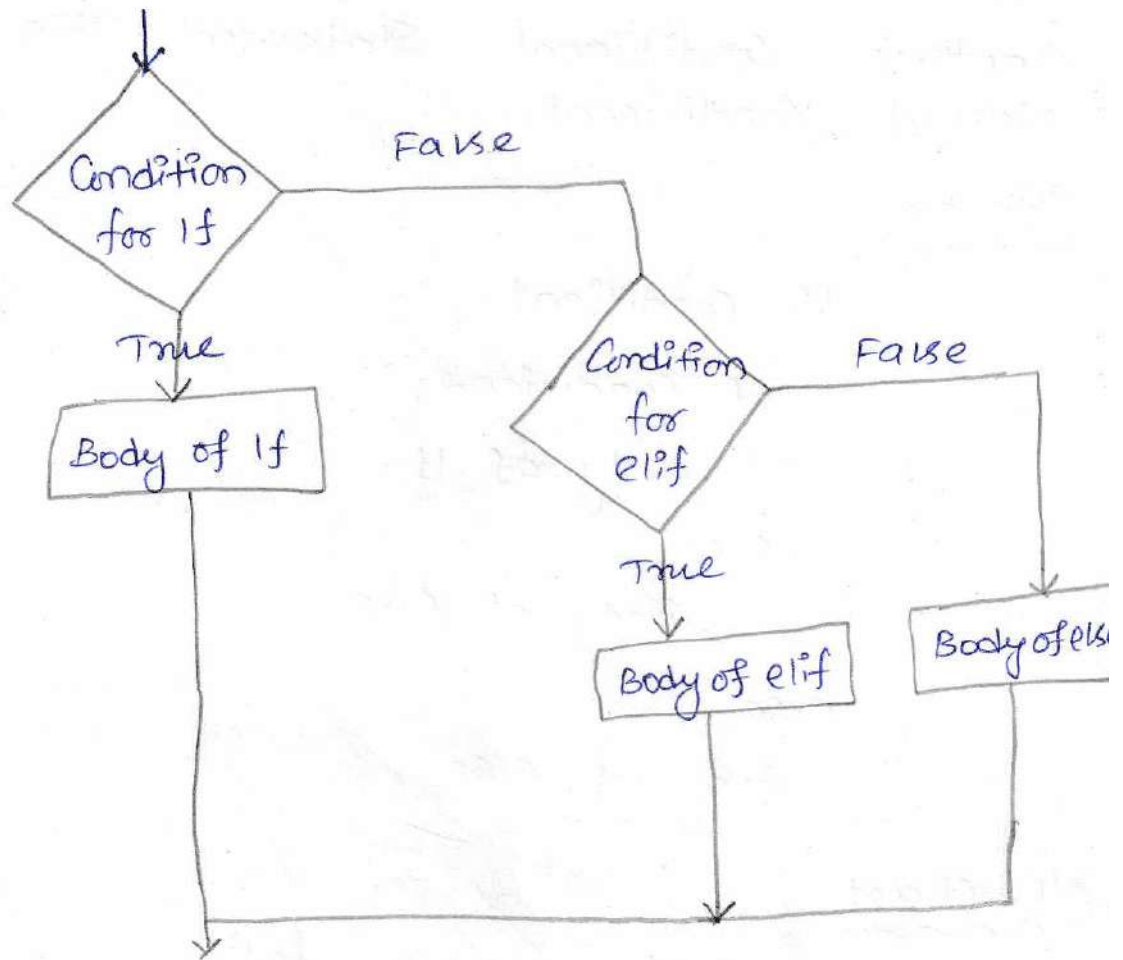
```
    Body of elif
```

```
else:
```

```
    Body of else
```

Flow chart

(5)



Example program

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

```
if num > 0:
```

```
    print(num, "is positive number")
```

```
elif num == 0:
```

```
    print(num, "is zero")
```

```
else:
```

```
    print(num, "is negative number")
```

Output

```
Enter any number -10
-10 is negative number
```


⑥

Nested If Statement

* The Conditional Statement is written within another Conditional Statement. This is called Nested Conditional.

Syntax

If Condition1:

 If Condition2:

 Body of If

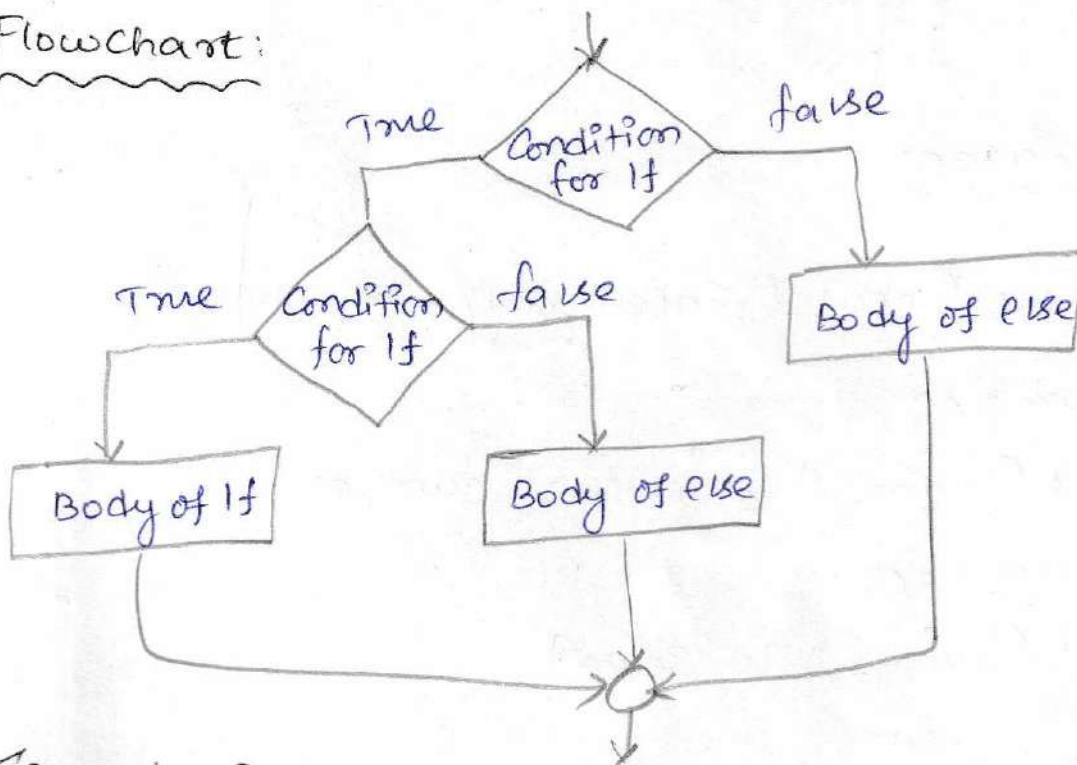
 else:

 Body of else

else:

 Body of else

Flowchart:



Example program

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

```
if num >= 0:
```

```
    if num == 0:
```

Print (num, "is zero")

⑦

else:

Print (num, "is positive number")

else:

Print (num, "is negative number")

Output

Enter any number: 5

5 is positive number.

Suggested Questions / Assignments / Home works / any other

1. Outline the conditional branching statements in Python with an example. (AU-Dec 2022)
2. List the three types of Conditional Statements & explain them. (AU-Dec 2019)
3. Write a Python program to find the greatest among three numbers using nested Conditional.

Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 21 Iteration (or) looping

Topic(s) to be covered	Looping - State of variable, while loop - For loop - looping with else break - Continue - Pass
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand The looping Statement	Understand
LO2	Describe the loop control Statements & working	Apply

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Iteration (or) Looping

State of a variable

- * It is possible to have more than one assignment for the same variable
- * The value which is assigned at the last is given to the variable.
- * The new assignment makes an existing variable assigned with new value by replacing the old value.

<u>For example</u>	<u>output</u>
$x = 5$	
$y = 3$	4
$x = 4$	3
Print(x)	
Print(y)	

Here,

The variable x is first assigned with 5. Then it is assigned with 4, the last assignment Statement $x = 4$ replaces the old value of x ($x = 5$).

Consider the following Program

	<u>output</u>
$x = 5$	
$y = x$	
$x = 3$	3
Print(x)	5
Print(y)	

Here,

The variable x is assigned with the value 5. The variable y is assigned with the value of x . Finally the value of x is updated to 3. So the value of x is 3. This is called State of the Variable.

Looping Statement

*It is used to execute a sequence of steps in a number of time or until a particular condition is met.

*Python has 2 types of looping statements

i) While loop ii) For loop

while loop

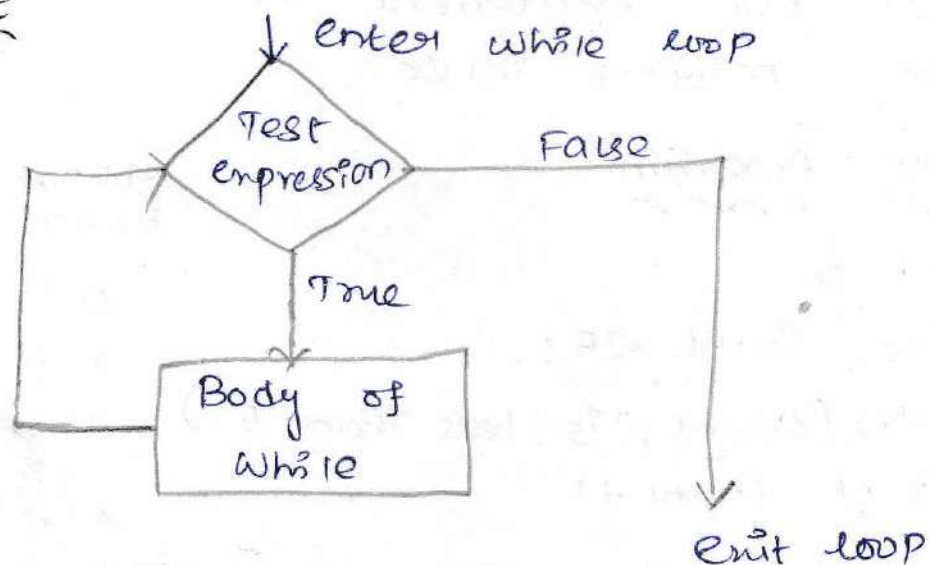
*The while loop is used to execute the statements within the body, until the condition becomes false.

*In each iteration, before executing the program code, the test expression will be evaluated. It stops the execution when the test expression is false.

Syntax

```
while test expression:
    Body of while
```

Flow chart



Example Program to add natural numbers upto n.

```
n = int(input("enter number"))
```

```
Sum = 0
```

```
i = 1
```

```
while i <= n:
```

```
    Sum = Sum + i
```

```
    i = i + 1
```

```
Print("The sum is", Sum)
```

Output

enter number 5

The sum is 15

Example

```
Count = 0
```

```
while Count < 6:
```

```
    print(Count)
```

```
    Count = Count + 1
```

Output

0

1

2

3

4

5

While loop with else

*If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

Example Program

```
Count = 0
```

```
while Count < 5:
```

```
    print(Count, "is less than 5")
```

```
    Count = Count + 1
```

```
else:
```

```
    print(Count, "is not less than 5")
```

Output

0 is less than 5

1 is less than 5

2 is less than 5

3 is less than 5

4 is less than 5

5 is not less than 5

For loop

* For loop is used to execute a set of instructions repeatedly, until the condition becomes false.

Syntax

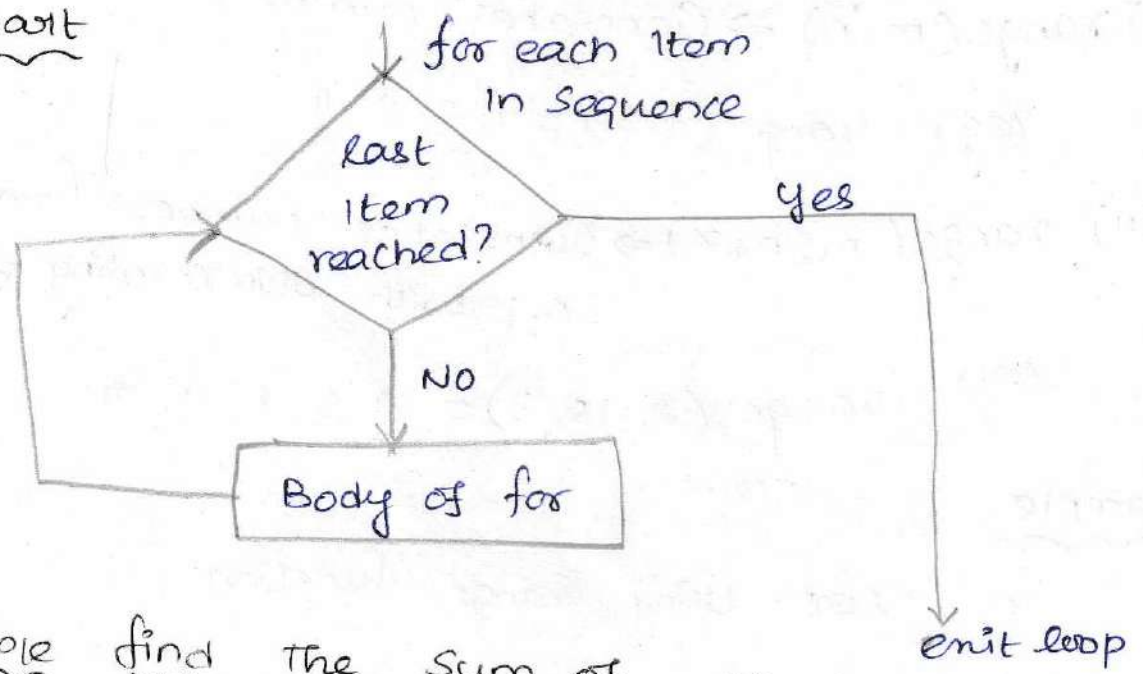
Method 1 General form of for loop

```
for variablename in sequence:  
    Statement
```

Method 2 For loop with range function

```
for variablename in range(Start, end, Step):  
    Statement
```

Flowchart



Example find the sum of all numbers stored in a list

```
number = [6, 5, 3, 8, 4]  
sum = 0  
for i in number:
```

Sum = Sum + i

Print("The sum is", Sum)

Output

The sum is 26.

For loop with range function

* To generate the sequence of numbers, range() function is used.

* The range function can be used in 3 different ways

i) range(n) $\Rightarrow$ Generates numbers from 0 to n-1

(eg) range(5) = 0, 1, 2, 3, 4

ii) range(m, n) $\Rightarrow$ Generates numbers from m to n-1

(eg) range(1, 5) = 1, 2, 3, 4

iii) range(m, n, x) $\Rightarrow$ Generates numbers from m to n-1 with skip counting of x.

(eg) range(2, 15, 3) = 2, 5, 8, 11, 14

Example

for loop using range function

Sum = 0

for i in range(6):

Sum = Sum + i

Print("The sum is", Sum)

output

The sum is 15

For loop with else

* If the else Statement is used with a for loop, The else Statement is executed when The loop has exhausted Iterating The list.

(Example)

```
digits = [0, 1, 5]
```

```
for i in digits:
```

```
    Print(i)
```

```
else:
```

```
    Print("no of Items left")
```

Output

0

1

5

no Items left

Loop Control Statements

* Loop Control Statements change execution from its normal sequence

* Python Supports The following Control Statements

break Statement

Continue Statement

Pass Statement

break Statement

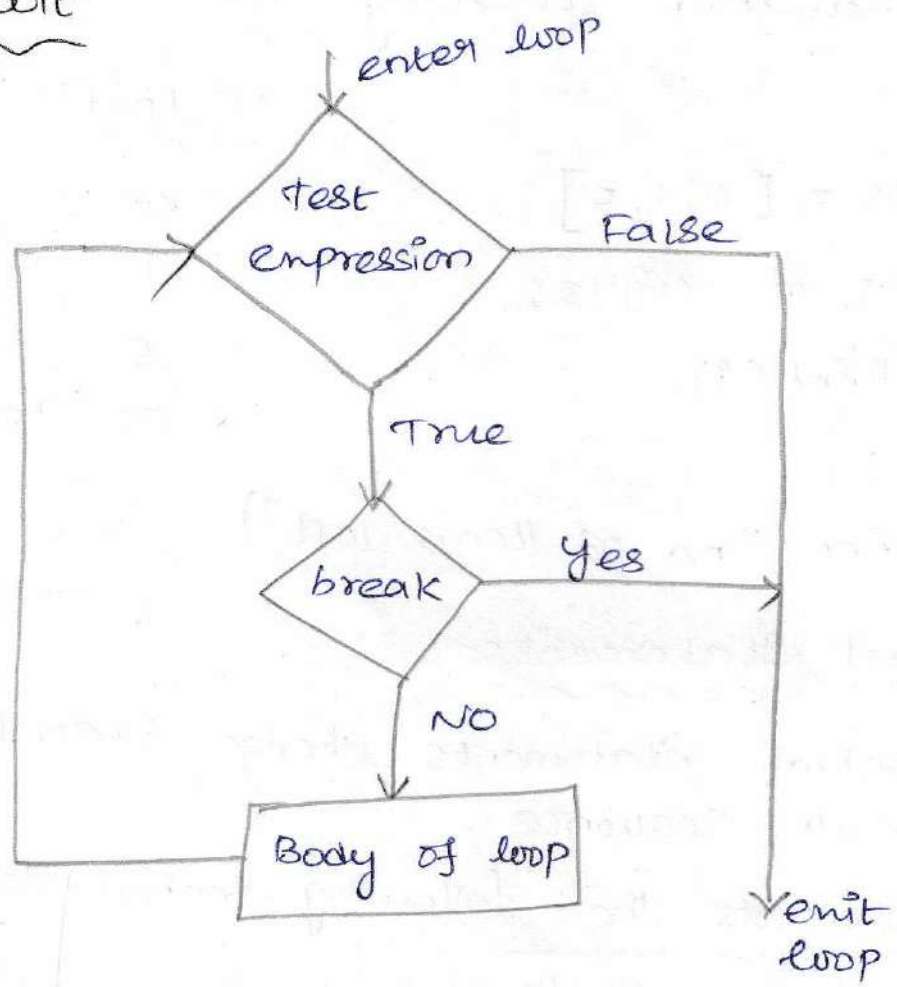
* The break Statement is used to terminate the loop.

* When the keyword break is used inside any loop, control is automatically transferred to the first Statement after the loop

Syntax

break

Flowchart



(Example)

```

n = "python"
for val in n:
    if val == 'h':
        break
    else:
        Print(val)
  
```

Output

p
y
t

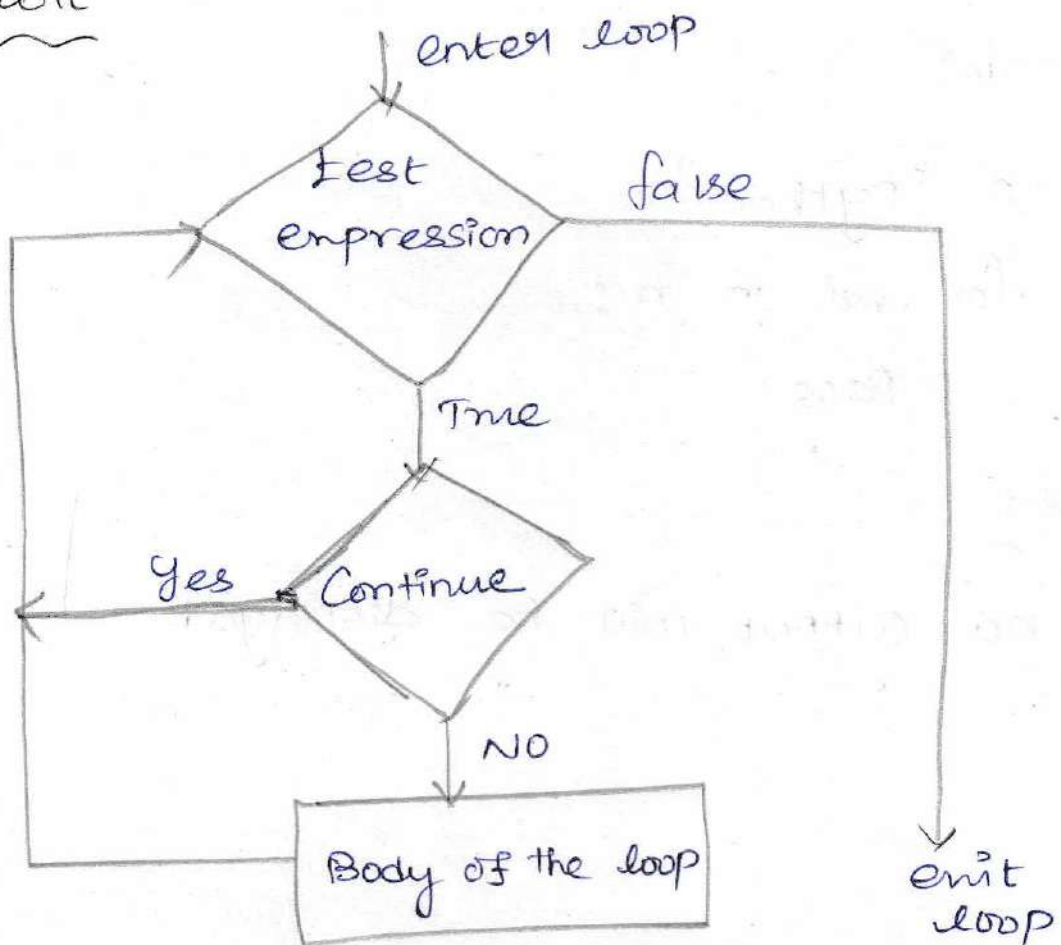
Continue Statement

* Continue Statement is used inside any python loop, Control is automatically transferred to beginning of the loop.

Syntax

Continue

Flowchart



Example

```

n="python"
for val in n:
    if val == 'h':
        Continue
    else:

```

Output

p
y
t
o
n

Pass Statement

- * Pass Statement is a null operation, nothing happens when it executes.
- * we generally use it as a placeholder

Syntax

Pass

(Example)

```
n = "python"  
for val in n:  
    pass
```

Output

no output will be displayed.

Suggested Questions / Assignments / Home works / any other

1. Name the two types of Iterative State-ments supported by Python. (AU-Dec 2022)
2. write a Python Program to Print Sum of Cubes of the values of n variables. (AU-Dec 2019)
3. Do loop Statements have else clause? when will it be executed. (AU-Dec 2019)
4. Present the flow of execution for a while Statement. (AU-Dec 2019)

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 22 & 23 Function

Topic(s) to be covered	Function - Function types - Parameters - Function Prototypes - Fruitful function - return values - local scope, global scope - function composition
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand the function concept	Understand
LO2	Know the function prototypes, Function Composition, Parameters	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

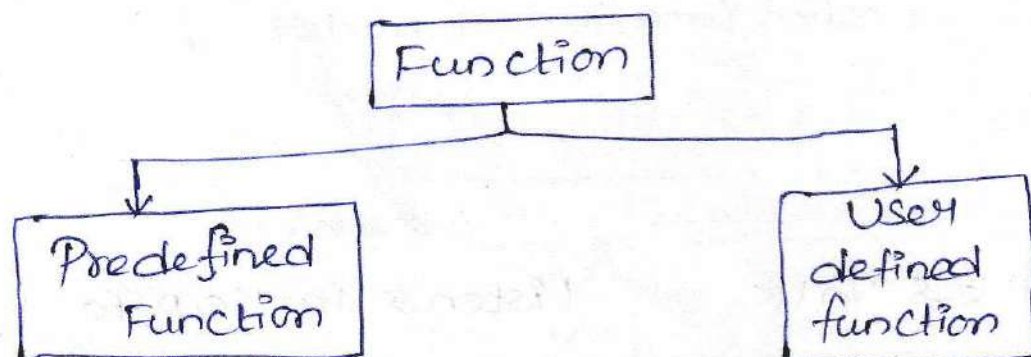
Functions

- * A function is a group of statements that perform a specific task
- * If a program is large, it is difficult to understand the steps involved in it.
- * Hence it is subdivided into a number of small programs called subprograms
- * Such subprograms called as function.

Advantages of Function

- * Decomposing larger Program into smaller function makes Program easy to understand.
- * Reduce Program development time & cost
- * It is easy to locate & isolate faulty function.

Types of Functions



Predefined / Built-In Function

- * Built in function are functions already built into Python interpreter & are readily available for use.

Example: `Print()`, `Input()`

Type Conversion functions

Python provides built in functions that convert values from one type to another

`int()` String, floating point $\rightarrow$ integer $\gg$ `int(2014)`
2014

$\gg$ `int(3.1415)`

3

float() String, integer → float >> float('1.99')

1.99

>> float(5)

5.0

Str() Integer, float → String >> str(510)

'510'

>> str(3.14)

'3.14'

list() String, tuple → list >> list('mary')

['m', 'a', 'r', 'y']

>>> list((1, 2, 3, 4))

[1, 2, 3, 4]

tuple() String, list → Tuple >> tuple('mary')

('m', 'a', 'r', 'y')

>>> tuple([1, 2, 3, 4])

(1, 2, 3, 4)

Math functions

* math & cmath are mathematical modules available in Python to support familiar mathematical functions.

* Before using built-in math functions, import math module

>>> import math

Some of the familiar math functions are

$\text{Sqrt}(n) \rightarrow$ Returns the square root of number

```
>>> math.sqrt(256)
```

16

$\text{factorial}(n) \rightarrow$ Returns a factorial

```
>>> math.factorial(5)
```

120

$\text{Pow}(n, d) \rightarrow$ Returns d raised to the power n

```
>>> math.pow(10, 3)
```

1000

$\text{gcd}(n, m) \rightarrow$ Returns greatest common divisor of (n, m)

```
>>> math.gcd(10, 125)
```

5

User defined function

* The functions defined by the user according to their requirements are called user-defined functions.

* User can modify the function according to their requirements.

(Example)

$\text{multiply}()$, $\text{sum}()$, $\text{avg-of-numbers}()$

Elements of user defined Function

(24)

1. Defining a function
2. Calling a function

Defining a Function

* Function in Python is defined using the keyword `def` followed by the function name and paranthese `()`.

* Syntax

```
def functionname (Parameters):
```

```
    Statement 1
```

```
    Statement 2
```

```
    return [expression]
```

The Function definition mainly consist of 2 element

- (1) Function Header 2) Function body

Function header

It mainly consist of 4 elements

- 1) `def` keyword
- 2) Function name
- 3) List of Parameters
- 4) Colon `(:)` symbol

Function Body

It consist of 3 elements

- 1) Local declaration variable if any (optional)

2) Executable Statements

(25)

3) Return Statements

Calling a function

Calling a function in python is similar to other programming languages, using the function name, Parenthesis () and Parameters.

Syntax

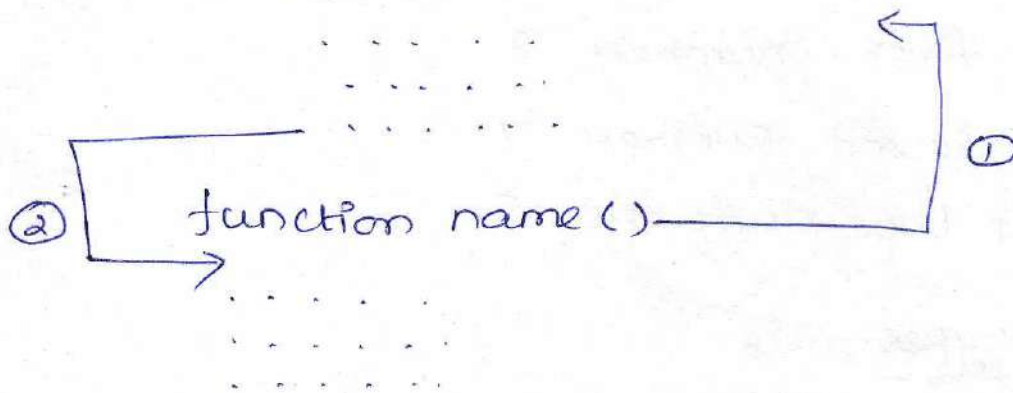
functionname (Parameters)

Flow of Execution

- * Flow of execution specifies the order in which statements are executed.
- * Program execution starts from first statement of the program. one statement is executed at a time from top to bottom.
- * Function definitions do not alter the flow of execution of the program & the statements inside the function are not executed until the function is called.
- * When a function is called, the control flow jumps to the body of the function, executes the function & return back to the program where the function call was made.

def functionname():

(26)



Parameters & Arguments

Arguments are the values provided to the function during the function call.

* Parameters are name used inside the function definition to refer to the value passed as an argument.

* Inside the function, the value of arguments passed during function call is assigned to Parameters.

Example

Program to find average of two numbers using function

def avg(x, y):

num = (x + y) / 2

Print("Average of two numbers", num)

a = int(input("enter the first number"))

b = int(input("enter the second number"))

avg(a, b) -----> Arguments

Output

Enter the first number 3

Enter the second number 7

Average of two numbers 5

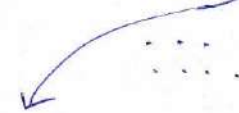
Function Prototypes

- Function with no arguments and no return values
- Function with arguments and no return values
- Function with arguments and with return value
- Function with no arguments and with return value

Function with no arguments and no return value

*The called program does not receive any data from the calling program and does not send back any value to the calling program.

Syntax:

fun ()

 function call
 with no
 argument

def fun():

.....

function not return any
 value

Example: Addition of two numbers.

def add():

a=int(input("enter value for a"))

b=int(input("enter value for b"))

$C = a + b$

(28)

Print("addition of a and b", C)

add()

Output

enter value for a 10

enter value for b 20

addition of a and b 30

Function with arguments and no return values

*The Called Program receives some data from the calling program & does not send back any values to calling program.

Syntax:

fun(a, b)

function call
with arguments

→ def fun(x, y)

function not return any
values.

Example:

```
def add(x, y):
```

```
    z = x + y
```

```
    Print("addition of a and b", z)
```

```
a = int(input("Enter value for a"))
```

```
b = int(input("Enter value for b"))
```

```
add(a, b)
```

Output

(29)

enter the value for a 10

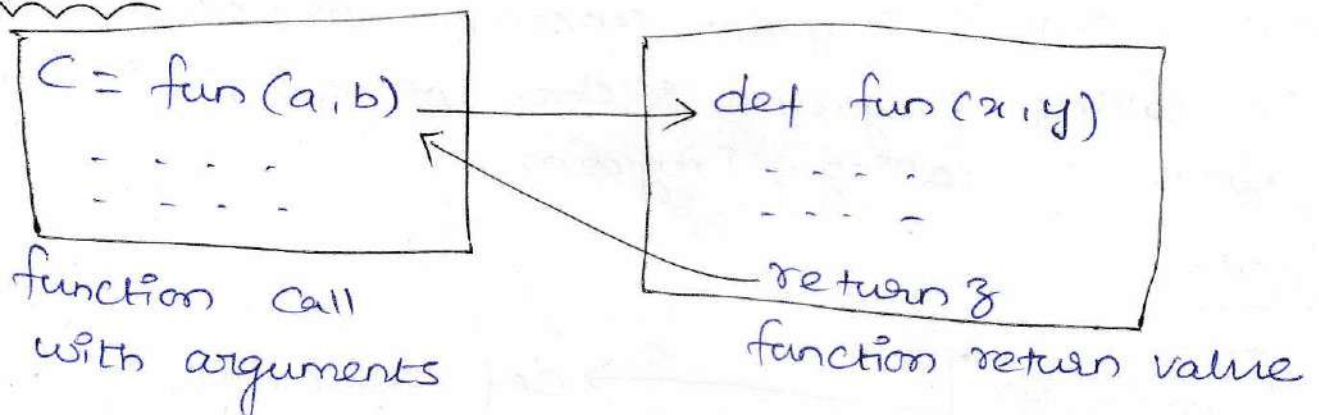
enter the value for b 20

addition of a and b 30

Function with arguments and with return value

*The Called Program receives some data from the Calling Program & send back a value return to the Calling Program.

Syntax



(Example)

```
def add(x, y):
```

```
    z = x + y
```

```
    return z
```

```
a = int(input("Enter value for a"))
```

```
b = int(input("Enter value for b"))
```

```
c = add(a, b)
```

```
Print("addition of a and b", c)
```

Output

Enter value for a 10

Enter value for b 20

Addition of a and b 30

(30)

Function with no arguments and with return Value

*The called program does not receive any data from the calling program and send back a value to the calling program.

Syntax

```
x = fun()  
-----  
-----
```

function call with
no argument

```
def fun():  
-----  
-----  
return c
```

function return value

(Example)

```
def add():  
    a = int(input("enter value for a"))  
    b = int(input("enter value for b"))  
    c = a + b  
    return c
```

```
x = add()
```

```
Print("addition of a and b", x)
```

Output

enter value for a 10

enter value for b 20

Addition of a and b 30

(31)

Fruitful function

* Python function is said to be fruitful function if it returns any value to the calling function.

Return values

* While using fruitful function, the return value must be handled properly by assigning it to a variable or use it as part of expression.

Example

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

```
    c = a + b
```

```
    return c
```

Output

Result 30

```
x = 10
```

```
y = 20
```

```
z = add(x, y)
```

```
Print("Result", z)
```

Void function

A function that does not return any value to the calling function is called void function.

Example

```
def add(a, b)
```

```
    c = a + b
```

Output

None

```
x = 10
```

```
y = 20
```

```
z = add(x, y)
```

```
Print("Result", z)
```


Parameters

(32)

Actual Parameter

- It is founded in the function call
- when a function is called, pass a value as a Parameter
- This value is called as actual Parameter

Formal Parameter

- It is founded in the function definition
- The variables defined in the function header are known as formal Parameters

Example

def add(a, b):

c = a + b

Print(c)

Output

30

x = 10

y = 20

add(x, y)

Actual Parameter

Parameter passing method (or) Argument passing method

Call by value (or) Pass by value

Call by reference (or) Pass by reference

Call by value (or) Pass by value

* In this method, the values of actual Parameters are copied to the formal parameters of the function.

* Even though Parameters's value are changed in the function, it will not be reflected back into the Calling Program.

Example Program

Swapping of two numbers

```
def swap(x, y):
```

```
    temp = x
```

```
    x = y
```

```
    y = x
```

```
    print("value of a & b inside swap function", x, y)
```

```
a = int(input("Enter value for a"))
```

```
b = int(input("Enter value for b"))
```

```
print("value of a & b before swapping", a, b)
```

```
swap(a, b)
```

```
print("value of a & b after swapping", a, b)
```

Output

```
Enter value for a 10
```

```
Enter value for b 20
```

```
Value of a & b before swapping 10 20
```

```
Value of a & b inside swap function: 20 10
```


Value of a & b after swapping 10 20

Call by Reference (or) Pass by Reference

* In this method, the addresses (or) references of the actual Parameters are passed to the formal Parameters of the function.

* If the actual Parameters are passed by reference the changes made in the values pointed by the formal Parameters in the called function are reflected back to the calling function.

Example

```
def processlist(list2):
```

```
    . . . . .
    list2[1] = 99
```

```
list1 = [5, 2, 19]
```

```
Print("values in the list before calling function  
are", list1)
```

```
processlist(list1)
```

```
Print("values in the list after calling function  
are", list1)
```

Output

Values in the list before calling function are
[5, 2, 19]

Values in the list after calling function are

Local & Global Scope

Scope of variable

- * All variables in a program may not be accessible at all locations in that program.
- * This depends on where you have declared a variable.

Local variable

- * These are the variables that are declared inside the function.
- * Its lifetime & scope is with function only.
- * We cannot access this outside the function.
- * If we try to access those variables then error will show by interpreter.

Global variable

- * These are the variables that are declared in the main program or above the function definition.
- * Its lifetime scope is within full program.

Example:

demo of local & global variable

```
a=1
b=2 } # global variable
```

```
def func():
```

```
    c=3
```

```
    d=4
```

```
    print("Value of a inside function", a)
```

```
    print("Value of b inside function", b)
```


Print("Value of c inside function", c) (36)
Print("Value of d inside function", d)

func()

Print("Value of a outside function", a)
Print("Value of b outside function", b)
Print("Value of c outside function", c)
Print("Value of d outside function", d)

Output

Value of a inside function: 1

Value of b inside function: 2

Value of c inside function: 3

Value of d inside function: 4

Value of a outside function: 1

Value of b outside function: 2

NameError: name 'c' is not defined

NameError: name 'd' is not defined

Access a function inside a function (or) Nested Function

*Python supports definition of a function inside another function

Example

x = 0

def outer():

x = 1

```

def inner():
    x = 2
    print("inner", x)

inner()
print("outer", x)

outer()
print("global", x)

```

Output

inner 2
outer 1
global 0

Non local Variable

- * Python introduced non local variable as a new kind of variable scope.
- * non local variable can only be used inside the nested function.

Example Using Non local

```

x = 0
def outer():
    x = 1
    def inner():
        nonlocal x
        print("inner", x)
    inner()
    print("outer", x)

outer()
print("global", x)

```

Output

inner 1
outer 1
global 0

Function Composition

- * Function composition is the ability to call one function from within another function.
- * The result of one function is passed as the argument of the next function.

Example

Compute area of circle with the given input
Centre Point (x_c, y_c) and Perimeter Point (x_p, y_p)

Note:

- * First find the radius of circle by computing distance between (x_c, y_c) and (x_p, y_p)
- * Later find the area of circle using radius

Import math

```
def distance(x1, y1, x2, y2):
```

```
    P = x2 - x1
```

```
    Q = y2 - y1
```

```
    r = P ** 2
```

```
    S = Q ** 2
```

```
    t = r + S
```

```
    U = math.Sqrt(t)
```

```
    return U
```

```
def area(r):
```

```
    a = 3.14 * r * r
```

```
    return(a)
```

$$x_c = 10$$

$$y_c = 10$$

$$x_p = 15$$

$$y_p = 10$$

Print (area (distance (x_c, y_c, x_p, y_p)))

Output

78.539

Suggested Questions / Assignments / Home works / any other

1. Write a Python Program using function to find the sum of first 'n' odd numbers and print the result. (AU-Dec 2022)
2. Brief fruitful function & void functions with example. (AU-Dec 2020)
3. Explain the syntax & structure of user defined functions in Python with examples.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 24 & 25 String

Topic(s) to be covered	Strings-String operations-String Predefined functions-Immutability
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand the string	Understand
LO2	Know the string functions & methods & immutable characteristics	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Strings

- * Strings are the sequence of characters represented within quotation marks.
- * It allows either pairs of single or double quotes.
- * String in Python can be combination of alphabets, numbers & special characters.
- * Some valid strings are

name = "Python"

name = 'Python'

n = '526'

String Operations

i) Index Operator

* Each character can be accessed through index operator

Syntax

variablename[index]

* Index starting value is zero

Example

S = "python"

Character	p	y	t	h	o	n
Index from left end	0	1	2	3	4	5
Index from right end	-6	-5	-4	-3	-2	-1

Example Program

S = "python"

Print (S[0])

Print (S[2])

Print (S[-3])

Output

p

t

h

ii) String slicing (or) slicing operator

* Slicing operator is used to select the Particular substring based on user requirements

Syntax

Variable name [start : end]

example Program

Output

S = "Python"

Pyth

Print (S [0: 4])

thon

Print (S [2:])

iii) Concatenation operator (+)

* We can concatenate one string with another string using Concatenation operator +

example Program

Output

a = "Python"

b = "Programming"

PythonProgramming

c = a + b

Print (c)

IV) Repetition operator (*)

* It is used to repeat the string several time. It is done with * operator.

Example Program

```
a = "Python"
```

```
Print(a * 3)
```

Output

```
PythonPythonPython
```

V) in and not in operator

* We can use in and not in operator to test whether a string is present in another string or not

Example Program

```
a = "python"
```

```
b = "then"
```

```
Print(b in a)
```

```
Print(b not in a)
```

Output

```
True
```

```
False
```

VI) String Comparison

* We can compare two string by using comparison operators such as ==, !=, <, >, <=, >=

* Python compares string based on their corresponding ASCII values.

Example Program

(45)

S = "green"

t = "red"

Print (S == t)

Print (S != t)

Print (S > t)

Print (S < t)

Output

False

True

True

False

String functions & Methods

Predefined String Functions

len() → len() function is used to find out the number of characters in string.

min() → min() function is used to return smallest character in a string.

max() → max() function is used to return largest character in a string.

Example Program

name = "Python"

Print (len(name))

Print (min(name))

Print (max(name))

output

6

h

y

Testing String Functions

`isalnum()` → Return True, if all characters in this string are alphanumeric

`isalpha()` → Return True, if all characters in this string are alphabetic

`isdigit()` → Return True, if this string contains only numbers.

`islower()` → Return True, if all characters in this string are lowercase letters

`isupper()` → Return True, if all characters in this string are uppercase letters.

Sample Program

```
a = "python"
```

```
Print(a.isalpha())
```

```
Print(a.isalnum())
```

```
Print(a.isdigit())
```

```
Print(a.islower())
```

```
Print(a.isupper())
```

Output

True

False

False

True

False

Converting String Functions

`capitalize()` → Returns a copy of this string with only the first character capitalized.

`lower()` → Returns a copy of this string with all the characters converted to

lowercase

`Upper()` → Returns a copy of this string with all the characters converted to uppercase.

`Title()` → Returns a copy of this string with the first letter capitalized in each word.

`Swapcase()` → Returns a copy of this string with lowercase letters to uppercase & uppercase letters to lowercase.

`replace(Old, new)` → Returns a new string that replaces all the occurrence of the old string with new string.

Example

Program

```
a = "Welcome to Python"
```

```
Print(a.capitalize())
```

```
Print(a.lower())
```

```
Print(a.upper())
```

```
Print(a.title())
```

```
Print(a.swapcase())
```

```
Print(a.replace("python", "Python Programming"))
```

Output

Welcome to python

welcome to python

WELCOME TO PYTHON

Welcome To Python

WELCOME TO PYTHON

Welcome to Python Programming

Formatting String Functions

- i) `Center(width)` → Returns a copy of this string centered in a field of the given width.
- ii) `ljust(width)` → Return a string left justified in a field of the given width.
- iii) `rjust(width)` → Return a string right justified in a field of the given width.
- iv) `Center(width, fill character)`

example

```
a = "Python"
```

```
Print(a.center(20))
```

```
Print(a.ljust(20))
```

```
Print(a.rjust(20))
```

```
Print(a.center(20, "*"))
```

output

```
Python
```

```
Python
```

```
Python
```

```
*** Python ***
```

Immutability → String is an immutable data structure. That means its characters can be accessed but string cannot be modified.

(eg) `a = "Python"`

```
a[0] = "H"
```

Output

```
Error
```

Searching for Substring

`endswith(substring)` → Return True, if the string ends with the substring.

`startswith(substring)` → Return True, if the string starts with the substring.

`Count(substring)` → Returns the number of occurrences of this substring.

(eg)

```
a = "welcome to python"
```

```
Print(a.endswith("thon"))
```

```
Print(a.startswith("good"))
```

output

```
True
```

```
False
```

Suggested Questions / Assignments / Home works / any other

1. what is String? How is it accessed?
2. what is Slice? Give example
3. Python Strings are immutable. Justify with an example (AU-Dec 2019)
4. Write a Python program to generate all Permutations of a given String using built in function. (AU-Dec 2019)

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 26 Array

Topic(s) to be covered	Arrays in Python - List as arrays - one dimensional array, two dimensional array.
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know the arrays in python	Understand
LO2	Understand the similarities between array & list	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Arrays in Python

* An array is a collection of similar data items that are stored under a common name

* Array is similar to list. The main difference is that arrays can store only one type of elements whereas list can store different type of elements.

Creating an array

Syntax:

```
arrayname = array (Typecode, [element])
```

(5)

We know that arrays can hold similar data types, so type should be specified by using a typecode.

<u>Typecode</u>	<u>Datatype</u>
i	Integer
f	float
U	Character

Importing the array module

*Array is not a fundamental datatype in python
So the standard array module has to be imported as

```
from array import *
```

(Example) A Python Program to Create an integer type array

```
from array import *
```

```
a = array('i', [1, 2, 3, 4, 5])
```

```
Print("The array elements are")
```

```
for i in a:
```

```
    Print(i)
```

Output

The array elements are

1
2
3
4

(Example 2) A Python Program to create an 52
Array with a group of characters

```
from array import *  
a = array('u', ['p', 'y', 't', 'h', 'o', 'n'])  
Print("The array elements are")  
for i in a:  
    Print(i)
```

Output

The array elements are

p
y
t
h
o
n

Indexing an array

(* An index represents the position number of an element in an array.

a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5

(Example)

```
from array import *  
a = array('i', [1, 2, 3, 4, 5])  
Print(a[0])  
Print(a[4])
```

Output

1
5

Slicing an array

When we perform slicing operation on any array, we can retrieve a piece of the array that contains a group of elements.

Syntax

arrayname[start : stop]

(Example)

```
from array import *
a = array('i', [1, 2, 3, 4, 5])
Print(a[1:4])
```

Output

2
3
4

List as Array

Arrays look a lot like a list, list can be employed as array.

One dimensional array

* One dimensional array can easily be represented as a list.

* These array elements represents only one column (or) row

(Example Program)

(54)

```
a = [45, 23, 76, 12, 33]
```

```
Print("The given elements are")
```

```
for i in range(len(a)):
```

```
    Print(a[i])
```

Output

The given elements are

45

23

76

12

33

Two dimensional array

* Two dimensional array can be created using list within list.

* These array represent 2 row & 2 column of elements

(Example)

```
a = [[1, 2], [3, 4]] (or) a = [[1, 2], [3, 4]]
```

Here

[1, 2] → First Row

$a =$

(0,0)	(0,1)
(1,0)	(1,1)

$$a[0][0] = 1$$

$$a[0][1] = 2$$

$$a[1][0] = 3$$

$$a[1][1] = 4$$

$$a[0] = [1, 2]$$

$$a[1] = [3, 4]$$

(Sample Program)

$$a = [[1, 2], [3, 4]]$$

for i in range(len(a)):

for j in range(len(a[i])):

print(a[i][j])

Output

1

2

3

4

Suggested Questions / Assignments / Home works / any other

1. Compare lists & array with example.
Can list be considered as an array?
Justify. (AU-Dec 2019)
2. Given an array of n elements, write a function to search a given element x in the array. (AU-Dec 2020)
3. Write a Python Program to find sum of array of numbers.

Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 27 & 28 Illustrative Programs

Topic(s) to be covered	Illustrative Programs: Square root, GCD, Exponentiation, Sum an array of numbers, linear, Binary search
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand ^{& write} the concept of linear, binary search, GCD, Square root, exponentiation	Apply

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Illustrative Programs

GCD of two numbers

```
x = int (input ("Enter num1: "))
y = int (input ("Enter num2: "))
if x < y:
    Small = x
else:
    Small = y
```



```
for i in range(1, small+1):
```

```
    if ((x % i == 0) and (y % i == 0)):
```

```
        gcd = i
```

```
Print("GCD = ", gcd)
```

Output

Enter num1: 6

Enter num2: 12

GCD = 6

Exponentiation of a Number

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

$n = 2$

```
e = int(input("Enter the Exponent value:"))
```

$e = 3$

```
r = n
```

```
for i in range(1, e):
```

$r \rightarrow r = 2$

```
    r = n * r
```

$i \rightarrow (1, 3) \rightarrow 1$

$i \rightarrow 1$

$r = 2 * 2$

$i \rightarrow 2$

```
Print(n, "Power", e, "is", r)
```

$r = 2 * 4$
 $= 8$

Output

Enter the number: 2

Enter the Exponent value: 3

2 Power 3 is 8

(59)

Square root of a Number

```
import math
```

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

```
Sqroot = math.sqrt(num)
```

```
Print("The square root of", num, "is", Sqroot)
```

Output

Enter a number: 16

The square root of 16 is 4.0

Program to compute GCD, square root & Exponentiation

```
import math
```

```
Print(math.gcd(12, 18))
```

```
Print(math.sqrt(100))
```

```
Print(math.exp(5, 2))
```

Output

6

10

25

(60)

Sum an array of elements

from array import *

a = array('i', [])

n = int(input("enter number of elements in list"))

for i in range(0, n):

 v = int(input("enter number"))

 a.insert(i, v)

Print("Elements are", a)

Sum = 0

for j in a:

 Sum = Sum + j

Print("sum of elements", Sum)

Output:

enter number of elements in list 5

enter number 10

enter number 20

enter number 30

enter number 40

enter number 50

Elements are 10 20 30 40 50

Sum of elements 150

Sequential Search (or) Linear Search

(61)

* Sequential Search is a method for finding a Particular value in a list.

* Linear Search Checks each elements in Sequen until the desired element is found (or) the list is exhausted.

(Example)

$a = [20, 30, 40, 50, 60]$

	↑	↑	↑	↑	↑
Index	0	1	2	3	4

Case 1: Search value = 50

Step 1: Compare 50 with value at Index 0

Step 2: Compare 50 with value at Index 1

Step 3: Compare 50 with value at Index 2

Step 4: Compare 50 with value at Index 3

Element is found (Success)

Case 2: Search value = 70

Step 1: Compare 70 with value at Index 0

Step 2: Compare 70 with value at Index 1

Step 3: Compare 70 with value at Index 2

Step 4: Compare 70 with value at Index 3

Step 5: Compare 70 with value at Index 4

Program

(62)

```
Item = [ ]
n = int(input("enter number of elements"))
for val in range(0, n):
    num = int(input("Enter number"))
    Item.append(num)
Print("list of Item is :", Item)
x = int(input("Enter search element"))
i = 0
Pos = 0
while i < len(Item):
    if Item[i] == x:
        Pos = 1
        break
    else:
        i = i + 1
if Pos == 1:
    Print("Item found at Position", i)
else:
    Print("Item not found")
```

Output:

```
enter number of elements 5
enter number 5
enter number 7
```

enter number 12

(63)

enter number 15

list of item is [5, 7, 10, 12, 15]

Enter search element 12

Item found at position 4.

Binary Search

* Binary search is simpler & faster than the linear search.

* Binary search requires the list to be sorted in ascending order.

* Binary search algorithm begins with comparing the element present at the middle of the list.

* If there is match, then the element is found. Search ends.

* If there is no match with the middle element & the search element is less than the middle element then first part of the list is searched. Otherwise second part of the list is searched.

* This process continues until the search element is equal to the middle element (or) list is exhausted.

(Program)

```
Item = [ ]
```

```
n = int(input("enter number of elements in Item"))
```

```
for val in range(0, n):
```

```
    num = int(input("enter number"))
```

```
    Item.append(num)
```



```

print ("list of Item is: ", Item)
x = int (input ("enter search element: "))
first = 0
last = len (Item) - 1
while first <= last:
    mid = (first + last) // 2
    if Item[mid] == x:
        print ("element found at ", mid)
        break
    elif x < Item[mid]:
        last = mid - 1
    else:
        first = mid + 1
else:
    print ("element not found")

```

Output

```

enter number of elements in Item 5
enter number 5
enter number 7
enter number 10
enter number 12
enter number 15
list of Item is: [5, 7, 10, 12, 15]
enter search element 7
element found at 1.

```

Suggested Questions / Assignments / Home works / any other

1. Write recursive implementation of binary Search. (AU- Dec 2020)
2. Write a python code to perform binary Search. Trace it with an example of your choice. (AU- Dec 2019)
3. Write a Python program to find GCD, Square root & exponentiation.

Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 29 & 30 List

Topic(s) to be covered	List - operations - methods & Functions - Mutability - Cloning - Aliasing - List Parameters
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	understand the list datatype	understand
LO2	List operation & methods in list datatype	understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

List

- * List is a datatype of Python
- * A list is a sequence of values, they can be of any type.
- * The values in a list are called elements or items.
- * List is enclosed by square brackets [] where the items are separated by commas.

②

Creating a list

There are several ways to create a new list

1. The simplest way to enclose the elements in square brackets []

(Example)

```
list1 = [1, 2, 3, 4, 5]
```

```
list2 = ['Python', 'Program', 'Simple']
```

```
list3 = [1, 5.6, 'python']
```

2. A list contains no element called an empty list. It can be created with empty brackets []

(Example)

```
l1 = [ ]
```

3. A list that is an element of another list is called nested list

(Example)

```
l2 = ['Python', 2.0, [50, 100]]
```


(3)

List operations

i) Index operator []

* An element in the list can be accessed through index operator

Syntax

listname[index]

(Eq) list1 = [1, 2, "python", 5.6, "abcd"]

list values	1	2	python	5.6	abcd
left end index	0	1	2	3	4
right end index	-5	-4	-3	-2	-1

Sample program

```
list1 = [1, 2, "python", 5.6, "abcd"]
```

```
print(list1[0])
```

```
print(list1[3])
```

```
print(list1[-3])
```

output

1

5.6

"python"

ii) Concatenation operator +

* It is used to join two lists into single list

(Eq) list1 = ["welcome"]

list2 = ["python"]

list3 = list1 + list2

```
print(list3)
```

output

["welcome", "python"]

④

eg - 2

list1 = ["abc", 2017]

list2 = list1 + ["CSE"]

Print(list2)

output

["abc", 2017, "CSE"]

iii) Repeat operator *

We can use * operator to repeating a list to specified number of times.

(Eg) list1 = [10, 20]

list2 = list1 * 3

Print(list2)

output

[10, 20, 10, 20, 10, 20]

iv) In and not in operator

We can determine whether an element is present in the list by using in or not in operator.

(Eg)

list1 = [1, 2, 3, 4, 5]

Print(1 in list1)

Print(100 in list1)

Print(5 not in list1)

Print(10 not in list1)

output

True

False

False

True

List Slices

We can extract sub part of list based on user requirements

Syntax listname[start : end]

(eg) list1 = [1, 2, "Python", 5.6, "abcd"]

Print(list1[0:2])

Print(list1[1:4])

Print(list1[:5])

Print(list1[?])

Print (list1[-2:])

(5)

Output

[1, 2]

[2, "Python", 5.6]

[1, 2, "Python", 5.6, "abcd"]

[1, 2, "Python", 5.6, "abcd"]

[5.6, "abcd"]

List Methods and Functions

Python list provide many built in functions & methods

1) append()

Add a new element to end of list

Syntax

listname.append(new element)

(eg)

list1 = [1, 2, 3, 4, 5]

list1.append(20)

Print (list1)

Output

[1, 2, 3, 4, 5, 20]

2) Count()

Returns the number of occurrences of an element in a list

Syntax

listname.Count(element)

Output

(eg)

list1 = [1, 2, 3, 4, 5, 1, 2, 1, 1]

Print (list1.Count(1))

4

⑥

3) Extend()

This appends all the elements in the second list at end of first list

Syntax firstlist.extend(second list)

(Eg) list1 = [1, 2, 3, 4, 5] output
 list2 = [6, 7, 8] [1, 2, 3, 4, 5, 6, 7, 8]
 list1.extend(list2)
 Print(list1)

4) Index() : Return the index of an element

Syntax : listname.index(element)

(Eg) list1 = [10, 20, 30, 40, 50] output
 Print(list1.index(30)) 2

5) Insert() : Insert an element at any given index

Syntax : listname.insert(index value, element)

(Eg) list1 = [10, 20, 30, 40, 50] output
 list1.insert(1, 15) [10, 15, 20, 30, 40, 50]
 Print(list1)

6) Pop() : Remove & returns an element at the given index

Syntax : listname.pop(index)

(Eg) list1 = [10, 20, 30, 40, 50]
 Print(list1.pop(2))

Print (list 1)

output
[10, 20, 40, 50]
30

7

7) remove() → Remove an element from the list

Syntax: `listname.remove(element)`

(eg) list1 = [10, 20, 30, 40, 50]

output

list1.remove(30)

[10, 20, 40, 50]

Print (list1)

8) reverse() → Reverse the entire list

Syntax: `listname.reverse()`

(eg) list1 = [10, 20, 30, 40, 50]

output

list1.reverse()

[50, 40, 30, 20, 10]

Print (list1)

9) sort() → Sort the list in ascending order

Syntax: `listname.sort()`

(eg) list1 = [40, 10, 50, 20, 30]

output

list1.sort()

[10, 20, 30, 40, 50]

Print (list1)

10) del() → Delete a Particular element in the list based on Index values.

Syntax: `del listname[Index value]`

(eg)

list1 = [10, 20, 30, 40, 50]

output

del list1[3]

[10, 20, 30, 50]

Print (list1)

To delete whole list

`del listname`

(eg) `list1 = [10, 20, 30, 40, 50]`

`del list1`

`Print(list1)`

Output
error

11) len() → Find the number of elements in list

Syntax: `len(listname)`

(eg) `list1 = [10, 20, 30, 40, 50]`

`Print(len(list1))`

Output
5

12) min() → Returns minimum value that present in the list

Syntax `min(listname)`

(eg) `list1 = [10, 20, 30, 40, 50]`

`Print(min(list1))`

Output
10

13) max() → Returns maximum value that present in the list

Syntax `max(listname)`

(eg) `list1 = [10, 20, 30, 40, 50]`

`Print(max(list1))`

Output
50

14) Sum() → Returns the total value by adding all the element in the list

Syntax `sum(listname)`

(eg) `list1 = [10, 20, 30, 40, 50]`

Print(Sum(list1))

output
150

⑨

List loop

* Accessing & processing each of the element in a list

Syntax for variablename in listname:
Process each items

(eg 1) list1 = [1, 2, 3, 4, 5]

for i in list1:

Print(i)

output
1
2
3
4
5

(eg 2) using range function

list1 = [1, 2, 3, 4, 5]

for i in range(len(list1)):

Print(list1[i])

output
1
2
3
4
5

Creating list using input() function

list1 = []

n = int(input("Enter number of elements"))

for i in range(0, n):

a = eval(input("Enter element"))

list1.append(a)

Print(list1)

output: Enter number of elements 3

Enter element 10

Enter element 20.5

Enter element 30

[10, 20.5, 30]

Mutability

* list is a mutable data structure

* This means we can add/delete/replace any value in list at any time

(Eg) list1 = [10, 20, 30, 40, 50]

Print ("Old values", list1)

list1[2] = 100

Print ("New values", list1)

o/p

Old values

[10, 20, 30, 40, 50]

New value

[10, 20, 100, 40, 50]

Aliasing

If list1 is a list object which holds data & you assign list1 to list2 then both list refer same object.

(eg) list1 = [1, 2, 3]

list2 = list1

If an object has more than one reference & one name, then the object is said to be aliased.

Aliased object is mutable in nature. So modification done in one object affect the other object.

(Eg) list1 = [1, 2, 3]

list2 = list1

Print ("list1 values", list1)

Print ("list2 values", list2)

list1[0] = 100

Print ("new list1 values", list1)

Print ("new list2 values", list2)

Output

list1 values [1, 2, 3]

list2 values [1, 2, 3]

new list1 values [100, 2, 3]

new list2 values [100, 2, 3]

Cloning list

* Cloning operation can be used to create a copy of an existing list.

* So that changes made in one copy of list will not affect another.

* The easiest way to clone a list is to use the slice operator

* Taking any slice of a list creates a new list. In this case, the slice consists of the whole list.

* The changes made in the cloned list will not be reflected back in the original list.

(Eg) 1 list = ['A', 'B', 'C', 'D'] Using slice operator

temp = list[:]

Print("list = ", list)

Print("Temp = ", temp)

Print("Changes in the cloned list will not affect original list")

temp.append('E')

Print("list = ", list)

Print("Temp = ", temp)

Output

list = ['A', 'B', 'C', 'D']

Temp = ['A', 'B', 'C', 'D']

Changes in the cloned list will not affect original list

list = ['A', 'B', 'C', 'D']

Temp = ['A', 'B', 'C', 'D', 'E']

(Eg) 2 Using list() function

list1 = [10, 20, 30]

list2 = list(list1)

Print("list1 = ", list1)

Print("list2 = ", list2)

```
list1[0] = 15
```

```
Print("list 1 = ", list1)
```

```
Print("list 2 = ", list2)
```

output

```
list1 = [10, 20, 30]
```

```
list2 = [10, 20, 30]
```

```
list1 = [15, 20, 30]
```

```
list2 = [10, 20, 30]
```

(Eg) 3) Using copy.copy() function

```
import copy
```

```
list1 = [10, 20, 30]
```

```
list2 = copy.copy(list1)
```

```
Print("list1 values", list1)
```

```
Print("list2 values", list2)
```

```
list1[0] = 15
```

```
Print("list1 value", list1)
```

```
Print("list2 value", list2)
```

output

```
list1 values [10, 20, 30]
```

```
list2 values [10, 20, 30]
```

```
list1 value [15, 20, 30]
```

```
list2 value [10, 20, 30]
```

List Parameters

* when a list is passed as a parameter to the function, the function gets a reference to the list

(Eg)

```
def processlist(list2):
```

```
    list2[1] = 99
```

```
list1 = [5, 2, 19]
```

```
Print("values in the list before calling function",
```

```
      list1)
```

```
processlist(list1)
```

```
Print("values in the list after calling function",
```

```
      list1)
```

Output

```
values in the list before calling function
```

```
[5, 2, 19]
```

```
values in the list after calling function
```

```
[5, 99, 19]
```


Suggested Questions / Assignments / Home works / any other

1. In Python, how the values are stored in a list are accessed? Should the elements of a list be of the same data type? (AU- Dec 2022)
2. Name the operations that can be performed on a list & outline any four with an example. (AU- Dec 2022)
3. What is cloning list? Give example. (AU- Dec 2020)
4. How will you update list items? Give one example (AU- Dec 2019)

Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 31 & 32 Tuples

Topic(s) to be covered	Tuples - Creating Tuples - Operations - Function & methods - Tuple Assignment
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand tuple data type	Understand
LO2	Describe the operations & methods in tuple	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Tuples

- * Tuple is datatype in Python
- * It is similar to list
- * A Tuple can hold items of different data types
- * The tuple is enclosed by Parentheses () where the items are separated by commas.
- * The tuple value cannot be updated. They are treated as read only list.

(15)

* Once a tuple is created, you cannot add/delete/replace any element in tuple.

* Tuple is immutable in nature.

Creating Tuples

Syntax

`tuple name = (element 1, element 2... element n)`

Example

`t1 = (1, 2, 3, 4, 5)`

`t2 = (1, "Python", 5.6)`

`t3 = ()`

Creating tuple with one element

`tuple = (1,)`

⇒ The built in function tuple() can be used to create tuple.

Creating tuple from String

`a = "hello"`

`Print(a)`

`b = tuple(a)`

`Print(b)`

Output

`"hello"`

`('h', 'e', 'l', 'l', 'o')`

(16)

Creating tuple from list

```
list1 = ['abc', 5.2, 10]
```

```
Print(list1)
```

```
tuple1 = tuple(list1)
```

```
Print(tuple1)
```

Output

```
['abc', 5.2, 10]
```

```
('abc', 5.2, 10)
```

Creating tuple from input() function

```
tuple1 = ()
```

```
name = input("Enter name")
```

```
no = int(input("Enter number"))
```

```
age = int(input("Enter age"))
```

```
tuple1 = (name, no, age)
```

```
Print(tuple1)
```

Output

```
Enter name xyz
```

```
Enter no 100
```

```
Enter age 17
```

```
("xyz", 100, 17)
```

Tuples Operations

i) Index operator

*An element in the tuple can be accessed through Index operator

Syntax

tuplename[Index]

(example)

```
t1 = (1, 2, "python", 5.6, "abcd")
```

```
Print(t1[0])
```

```
Print(t1[3])
```

```
Print(t1[-3])
```

Output

```
1
```

```
5.6
```

```
"python"
```


ii) Slicing of tuples [:]

* Slicing used to extract sub part of tuple based on user requirements.

Syntax

tuplename[start:end]

(eg) tuple1 = (1, 2, "python", 5.6) Output

Print(tuple1[0:3])

(1, 2, "python")

Print(tuple1[2:4])

("python", 5.6)

Print(tuple1[:])

(1, 2, "python", 5.6)

iii) Concatenation operator '+'

* It is used to join 2 ~~pro~~ tuples into single tuple

(eg)

t1 = (1, 2, 3)

Output

t2 = ('a', 'b', 'c')

(1, 2, 3, 'a', 'b', 'c')

t3 = t1 + t2

Print(t3)

iv) Repeating operator '*'

* It is used to repeating a tuple to specified number of times

(eg) t1 = (1, 2, 3)

Output

t2 = t1 * 3

(1, 2, 3, 1, 2, 3, 1, 2, 3)

Print(t2)

v) In and not in Operator

* Determine whether an element is Present in the tuple by using in and not in operator.

(Eg) `t1 = (1, 2, 3)` Output
`Print(1 in t1)` True
`Print(2 not in t1)` False

(Tuples functions & methods)

1) `len()` → Returns The length of the tuple

Syntax: `len(tuplename)`

(example) `t1 = (1, 2, 3, 4, 5)` Output
`Print(len(t1))` 5

2) `min()` → Find the minimum element from the tuple

Syntax: `min(tuplename)`

3) `max()` → Find The maximum element from the tuple

Syntax: `max(tuplename)` Output

(eg) `t1 = (10, 20, 30, 40, 50, 25)` 10
`Print(min(t1))` 50
`Print(max(t1))`

(19)

4) Sum() → Returns a value total by adding all the elements in the tuple

Syntax: `Sum(tuplename)`

(eg) `t1 = (10, 20, 30, 40, 50)`

`Print (sum(t1))`

Output
150

5) Cmp() → Used to Compare 2 tuples based on their elements

Syntax: `Cmp(tuple1, tuple2)`

Returns a value

0 → If both tuples are equal

-1 → If `tuple1 < tuple2`

+1 → If `tuple1 > tuple2`

(eg) `t1 = (1, 2, 3)`

`t2 = (1, 2, 3)`

`Print (Cmp(t1, t2))`

Output
0

6) Count() → Returns the number of occurrences of an element in a tuple

Syntax: `tuplename.Count(element)`

(eg)

`t1 = 1, 2, 3, 1, 4, 1)`

`Print (t1.Count(1))`

Output
3

7) Index() → This return the index of an element

Syntax: `tuplename.index(element)`

(eg) `t1 = (10, 20, 30, 40, 50)` Output
`Print(t1.index(30))` 2

8) del() → Used to delete the whole tuple

Syntax: `del tuplename`

(eg) `t1 = (10, 20, 30, 40)` Output
`del t1` Error
`Print(t1)`

Tuples loop

* Accessing & Processing each of the element in a tuple

Syntax: `for variablename in tuplename:`
 Process each items

(eg)-1
`t1 = (1, 2, 3, 4, 5)`
`for i in t1:`
`Print(i)`

Output
 1
 2
 3
 4
 5

(eg)-2
`t1 = (1, 2, 3, 4, 5)`
`for i in range(len(t1)):`
`Print(t1[i])`

Output
 1
 2
 3
 4
 5

Updating Tuple

*Tuples are Immutable means that The tuples Elements Cannot be Updated or Changed.

*No elements can be modified, If you try to modify you will get an error.

(eg) `t1 = (1, 2, 3, 4, 5)`

`t1[0] = 15`

Type Error: Tuple doesn't support item assignment

*But If the element is itself a mutable datatype like list, its nested item can be changed.

(eg) `t1 = (1, 2, 3, [8, 10])`

`t1[3][0] = 45`

`Print(t1)`

output

`(1, 2, 3, [45, 10])`

Tuple Assignment

*Tuple assignment makes it possible to create & assign value: for more than one tuple in a single statement

(eg) 1 $x, y = (1, 2)$

Print(x)

Print(y)

Print(x, y)

temp = x

x = y

y = temp

Print(x, y)

output

1

2

(1, 2)

(2, 1)

(eg) 2 $x, y = (2, 3)$

Print(x, y)

 $x, y = y, x$

Print(x, y)

o/p

(2, 3)

(3, 2)

(eg) 3 $x, y = (1, 2, 3)$

This Statement Create error

Value error : too many values

(eg) 4

 $t1 = (1, 2, 3, 4)$ $a, b, c, d = t1$ $\text{Print}(a)$ $\text{Print}(b)$ $\text{Print}(c)$ $\text{Print}(d)$ Output

1

2

3

4

Tuple as return values

* Functions usually return only one value, but if the value is a tuple, then function can return multiple values.

(Eg) To divide two integers & compute the quotient & remainder, normally it is divided using x/y and $x\%y$ but using Python, it's better to compute them both at the same time.

(Example Program)

method - 1 $t1 = \text{divmod}(7, 3)$ $\text{Print}(t1)$ Output

(2, 1)

Method - 2 $x, y = (7, 3)$ $z = \text{divmod}(x, y)$ $\text{Print}(z)$ Output

(2, 1)

Suggested Questions / Assignments / Home works / any other

1. Write Separate Python Programs to illustrate create, access, concatenate and delete operations in a tuple. (AU-Dec 22)
2. What is the difference between list and tuple in Python? (AU-Dec 20) (AU-Dec 19)
3. Can functions return tuples? If yes give example. (AU-Dec 19)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

(25)

Lecture No. 33 & 34 Dictionary

Topic(s) to be covered	Dictionary - Create a dictionary - Properties of keys - Methods & functions in dictionary.
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	understand dictionary	understand
LO2	know the dictionary operations & methods	understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Dictionaries

- * Dictionary is an unordered collection of items
- * Each item in dictionary has a key:value pair and list of items are enclosed inside curly braces {} separated by comma.
- * Keys must be immutable types (String, number or tuple) and must be unique.
- * Values can be any data type & can repeat.

* Dictionary is similar to a list, but in list elements can be accessed using index, in dictionary we access value by looking up a key instead of an index.

Creating a dictionary

- * We can create a dictionary by enclosing the item inside a pair of curly braces {}
- * Each item consists of a key & value. Each item may be separated with commas.

* Syntax

dictionary name = { key1: value1, key2: value2, ..., keyn: valuen }

Example

Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }

To create an empty dictionary

dictionaryname = { }

Accessing elements of dictionary

- * In dictionary elements are accessed by their keys only

Syntax

dictionaryname[keyvalue]

(Eg)

```
Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }
```

```
Print (Players[3])
```

Output

"rohit"

To access all keys & values

key: Syntax: dictionaryname.keys()

(Eg)

```
Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }
```

```
Print ("Players.keys()")
```

Output

([1, 2, 3])

Value: Syntax: dictionaryname.values()

(Eg)

```
Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }
```

```
Print (Players.values())
```

Output

("dhoni", "kohli", "rohit")

→ An attempt to access a key, that does not exist in the dictionary will cause an error

(Eg)

$$\text{Players} = \{1: \text{"dhoni"}, 2: \text{"kohli"}, 3: \text{"rohit"}\}$$

$$\text{Print}(\text{Players}[10])$$

Output : Error

Properties of dictionary keys

1. More Than one entry key not allowed
which means no duplicate key is allowed.
When duplicate keys encountered during assignment, the last assignment is taken

(Eg)

$$\text{Players} = \{1: \text{"dhoni"}, 2: \text{"kohli"}, 1: \text{"rohit"}\}$$

$$\text{Print}(\text{Players}[1])$$

Output

"rohit"

2. keys must be immutable

which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed

(Eg)

$$\text{Players} = \{[1]: \text{"dhoni"}, 2: \text{"kohli"}, 3: \text{"rohit"}\}$$

$$\text{Print}(\text{Players}[1])$$

output

TypeError

Dictionary Operations

1. Adding element to dictionary

We can add new elements (key, value) to dictionary using assignment operator.

Syntax

dictionaryname[new keyvalue]=value

(eg)

```
Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }
```

```
Players[4] = "Sachin"
```

```
Print (Players)
```

Output

```
{ 1: "dhoni", 2: "kohli", 3: "rohit", 4: "Sachin" }
```

2. Updating / modifying existing element in dictionary

* We can update element (key: value) to dictionary using assignment operator.

Syntax: dictionaryname[old key value]= new value

(eg)

```
Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }
```

```
Players[3] = "Sachin"
```

```
Print (Players)
```

Output

{ 1: "dhoni", 2: "kohli", 3: "sachin" }

3. Deleting element in dictionary

* There are 2 methods to delete elements in dictionary.

Method 1 : To delete dictionary elements by
del Command

Syntax : del dictionaryname[key]

(eg)

Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }

del Players[3]

Print (Players)

Output

{ 1: "dhoni", 2: "kohli" }

Method 2 : Delete The dictionary element by Pop method

Syntax: dictionaryname.pop(key)

(eg)

Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }

Players.pop(3)

Print (Players)

Output

{ 1: "dhoni", 2: "kohli" }

4) In and not in operator: we can determine whether an element is present in the dictionary by using in and not in operator.

Syntax

```
key in dictionaryname
key not in dictionaryname
```

(eg) `Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }`
`Print ( 1 in Players )`
`Print ( 3 not in Players )`

Output

```
True
False
```

5) Dictionary loop

For loop is mainly used to traverse over the items of the dictionary.

Syntax:

```
for variable in dictionaryname
    Processing each item
```

(eg) `Players = { 1: "dhoni", 2: "kohli", 3: "rohit" }`
`for i in Players:`
 `Print ("keys", i, "values", Players[i])`

Output

```
keys 1 values "dhoni"
keys 2 values "kohli"
keys 3 values "rohit"
```

Dictionary Functions & Methods

1. len() → This method returns the length of the dictionary i.e. Count of (key: value pair)

Syntax

`len(dictionaryname)`

(eg)

```
emp = {"name": "xyz", "age": 20, "salary": 10000}
```

```
Print(len(emp))
```

Output

3

2. clear() → This removes all items from the dictionary

Syntax:

`dictionaryname.clear()`

(eg)

```
emp = {"name": "xyz", "age": 20, "salary": 10000}
```

```
emp.clear()
```

```
Print(emp)
```

Output

{ }

3. get() → This method you can get the item with given key

Syntax:

`dictionaryname.get(key)`

(eg)

```
emp = {"name": "xyz", "age": 20, "Salary": 10000}
```

```
Print(emp.get("name"))
```

Output

"xyz"

4. has-key() → This method checks for the presence of given key in the dictionary.

Syntax: dictionaryname.has-key(key)

(eg)

```
emp = {"name": "xyz", "age": 20, "Salary": 10000}
```

```
Print(emp.has-key("age"))
```

Output

True

5. Items() → This method returns all the items in the dictionary.

Syntax: dictionaryname.items()

(eg)

```
emp = {"name": "xyz", "age": 20, "Salary": 10000}
```

```
Print(emp.items())
```

Output

```
{'name': 'xyz', 'age': 20, 'Salary': 10000}
```

6. `Cmp()` → This method is used to compare the dictionary based on their key values.

Answer will be 0 (or) 1 (or) -1
 (same) $\text{dict1} > \text{dict2}$ $\text{dict1} < \text{dict2}$

Syntax:

`Cmp(dict1, dict2)`

(eg)

`emp1 = {"name": "xyz", "age": 20,
 "Salary": 10000}`

`emp2 = {"name": "abc", "age": 25,
 "Salary": 20000}`

`Print(cmp(emp1, emp2))`

Output

0

Suggested Questions / Assignments / Home works / any other

1. Define dictionary in Python. Do the following operations i) Create ii) Compare iii) Find keys
2. How Python's dictionaries store data? Give example.
3. How can you access data from a dictionary?

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 35 Advanced List Processing

Topic(s) to be covered	Advanced list processing - List Comprehension
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know the short hand rules in list through list comprehension	Understand
LO2	Understand list comprehension with example	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Advanced List Processing

* Python has a better way to apply some short hand rules on lists through list comprehension & it is called advanced list processing.

* List comprehension is a way to define & make lists based on existing lists.

List Comprehension

* A Python list comprehension is made up of brackets containing the expression.

* It is executed for each element, as well as the for loop.

* Looping is used to iterate over the Python list elements.

* When we want to create a new list based on the values of an existing list, list Comprehension provides a shorter syntax.

Example 1

```
alist = [x for x in range(10)]
```

```
Print(alist)
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[x for x in range(10)]
```

output
↓

Variable
↘

* We can add conditional expressions after the loop sequence. But this is an optional feature

```
[x for x in range(10) if x % 2 == 0]
```

output
Sequence
Expression
↓

Variable of
Input
Sequence
↘

Input
Sequence
↘

Conditional
Expression
(optional)
↘

Sample 5

```
Pets = ["kiwi", "nana", "lulu", "Peggy", "maxi"]
```

```
newlist = [x for x in Pets if "a" in x]
```

```
Print (newlist)
```

Output

```
['nana', 'maxi']
```

Sample 6

```
alist = [x for x in "python" if x in  
         ['a', 'e', 'o', 'i', 'u']]
```

```
Print (alist)
```

Output

```
0
```

Sample 7

```
matrix = [[j for j in range(5)] for i in  
           range(4)]
```

```
Print (matrix[0])
```

```
Print (matrix[1])
```

```
Print (matrix[2])
```

```
Print (matrix[3])
```

Output

```
[0, 1, 2, 3, 4]
```

```
[0, 1, 2, 3, 4]
```

```
[0, 1, 2, 3, 4]
```

```
[0, 1, 2, 3, 4]
```


Example 2

```
aList = [x for x in range(10) if x%2==0]
```

```
Print(aList)
```

output

```
[0, 2, 4, 6, 8]
```

Example 3 we can use a list instead of defining separately outside, the list comprehension in a single line of code.

```
Print([x for x in [1, 2, 3, 4] if x%2==0])
```

output

```
[2, 4]
```

Example 4

```
letter = [letter for letter in 'Python']
```

```
Print(letter)
```

output

```
['P', 'y', 't', 'h', 'o', 'n']
```

The variable letter gets letters one by one from the string python.

Suggested Questions / Assignments / Home works / any other

1. What is list comprehension in Python?
Expression with example. (AU-Dec 2019)
2. Define Advanced list processing.
3. Define the rules for list comprehension.

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 36 Illustrative Program

Topic(s) to be covered	Illustrative programs - Student mark Statement - Retail bill Preparation
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Write programs for student mark Statement & Retail bill Preparation using Python	Apply

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Illustrative Programs
Student Mark Statement

Students = []

```
num1 = int(input("Enter number of Students:"))
for i in range(num1):
    tot = 0
    mark = [ ]
    x = input("Enter Student name")
    Students.append(x)
```

```
Print ("Enter marks obtained in 5 subjects:")
```

```
for i in range(5):
```

```
    mark.append(input())
```

```
for i in range(5):
```

```
    tot = tot + int(mark[i])
```

```
avg = tot/5
```

```
Print ("Name = ", x)
```

```
if avg >= 91 and avg <= 100:
```

```
    Print ("Grade = O")
```

```
elif avg >= 81 and avg <= 90:
```

```
    Print ("Grade = A+")
```

```
elif avg >= 71 and avg <= 80:
```

```
    Print ("Grade = A")
```

```
elif avg >= 61 and avg <= 70:
```

```
    Print ("Grade = B+")
```

```
elif avg >= 50 and avg <= 60:
```

```
    Print ("Grade = B")
```

```
else:
```

```
    Print ("Fail")
```


Output

Enter number of Students : 2

Enter name of Students : Arun

Enter marks obtained in 5 subjects :

98

97

96

91

95

Name = Arun

Grade = O

Enter name of Students : Rani

Enter marks obtained in 5 subjects

88

85

86

83

84

Name = Rani

Grade = A+

Retail bill preparation

44

tax = 0.18

Rate = { "Pencil": 10, "Pen": 20, "Scale": 7, "A4paper": 165,
"Writingpad": 15 }

Print ("Retail Bill Calculation")

Print ("Enter the quantity of ordered item: ")

Pencil = int (input ("Pencil: "))

Pen = int (input ("Pen: "))

Scale = int (input ("Scale: "))

A4 Paper = int (input ("A4 Paper: "))

Writing Pad = int (input ("Writing Pad: "))

Cost = Pencil * Rate ["Pencil"] + Pen * Rate ["Pen"] + Scale *
Rate ["Scale"] + A4 Paper * Rate ["A4paper"] +
Writingpad * Rate ["Writingpad"]

Bill = Cost + Cost * tax

Print ("Please Pay Rs. %f " % Bill)

Output

Retail Bill Calculation

Enter the quantity of ordered item:

Pencil: 5

Pen: 7

Scale: 4

A4 Paper: 3

Writing pad: 2

Please pay Rs. 876.74

(45)

Suggested Questions / Assignments / Home works / any other

1. Write a python program for Student mark Statement
2. Write a python program for retail bill Preparation.

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 37 Illustrative Program

Topic(s) to be covered	Histogram - Merge sort - Definition - Basic algorithm - Program
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Understand the Histogram	Understand
LO2	Write histogram, mergesort Program	Apply

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Histogram

- * A histogram is a graphical representation of frequency distributions.
- * It is a graph that displays the number of observations within each interval.
- * To generate histogram in matplotlib, we use hist() function.
- * The hist() function will generate a histogram from number of observation, which is passed into the function as an argument.

The arguments of hist() function are as

```
hist(x, bins=None, range=None, density=False,
     heights=None, Cumulative=False, bottom=None,
     histtype='bar', align='mid', orientation='vertical',
     label=None, stacked=False, *, data=None,
     **kwargs)
```

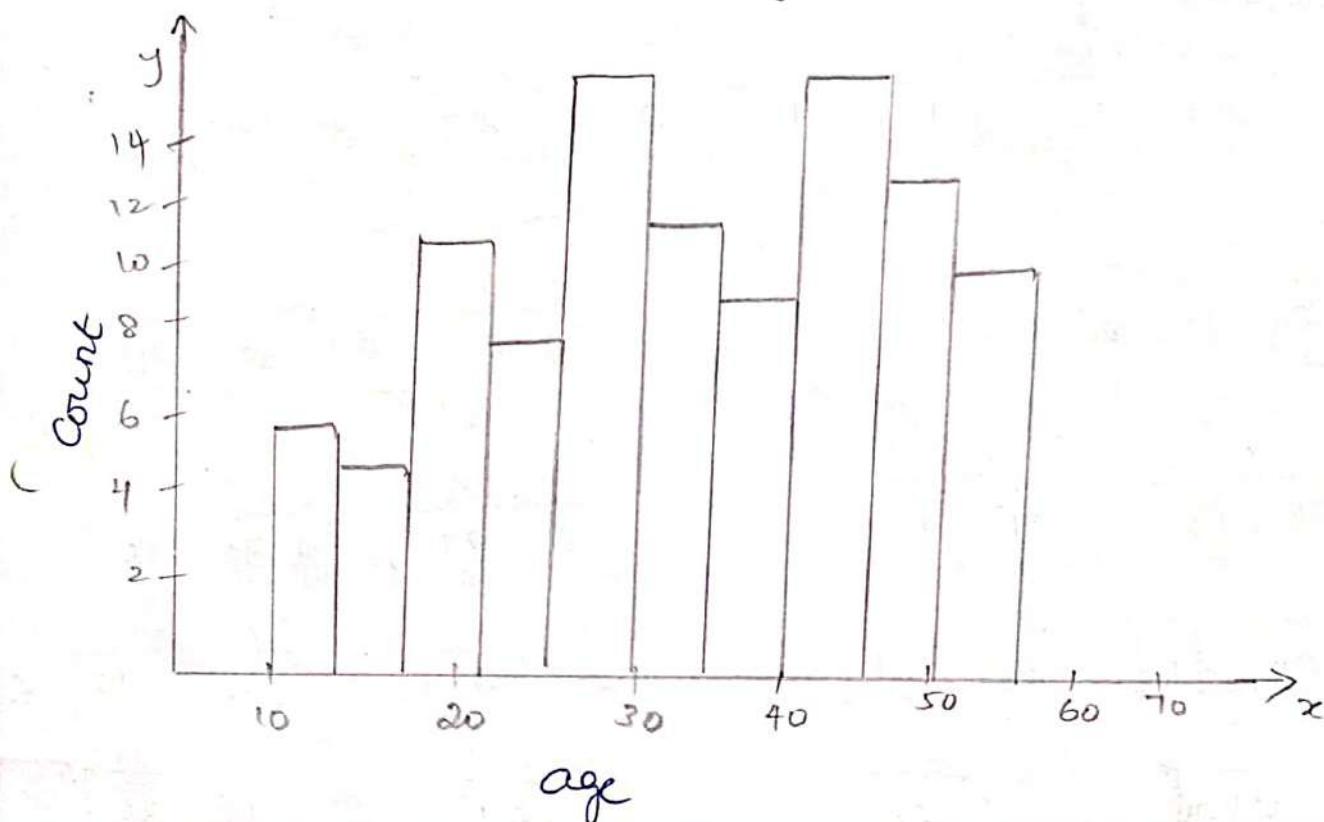
- * x is Compulsory argument in hist function
- * Other arguments are optional
- * x is a set of data
- bins used for providing intervals (default is 10)
- range used for range purpose
- kwargs means keyword argument, that provides edge color in graph.

Program

```
import matplotlib.pyplot as plt
x = [1, 12, 22, 21, 20, 21]
plt.hist(x, ec="red")
plt.xlabel("age")
plt.ylabel("Count")
plt.title("Population age Count")
plt.show()
```

Output

Population age Count

Merge Sort

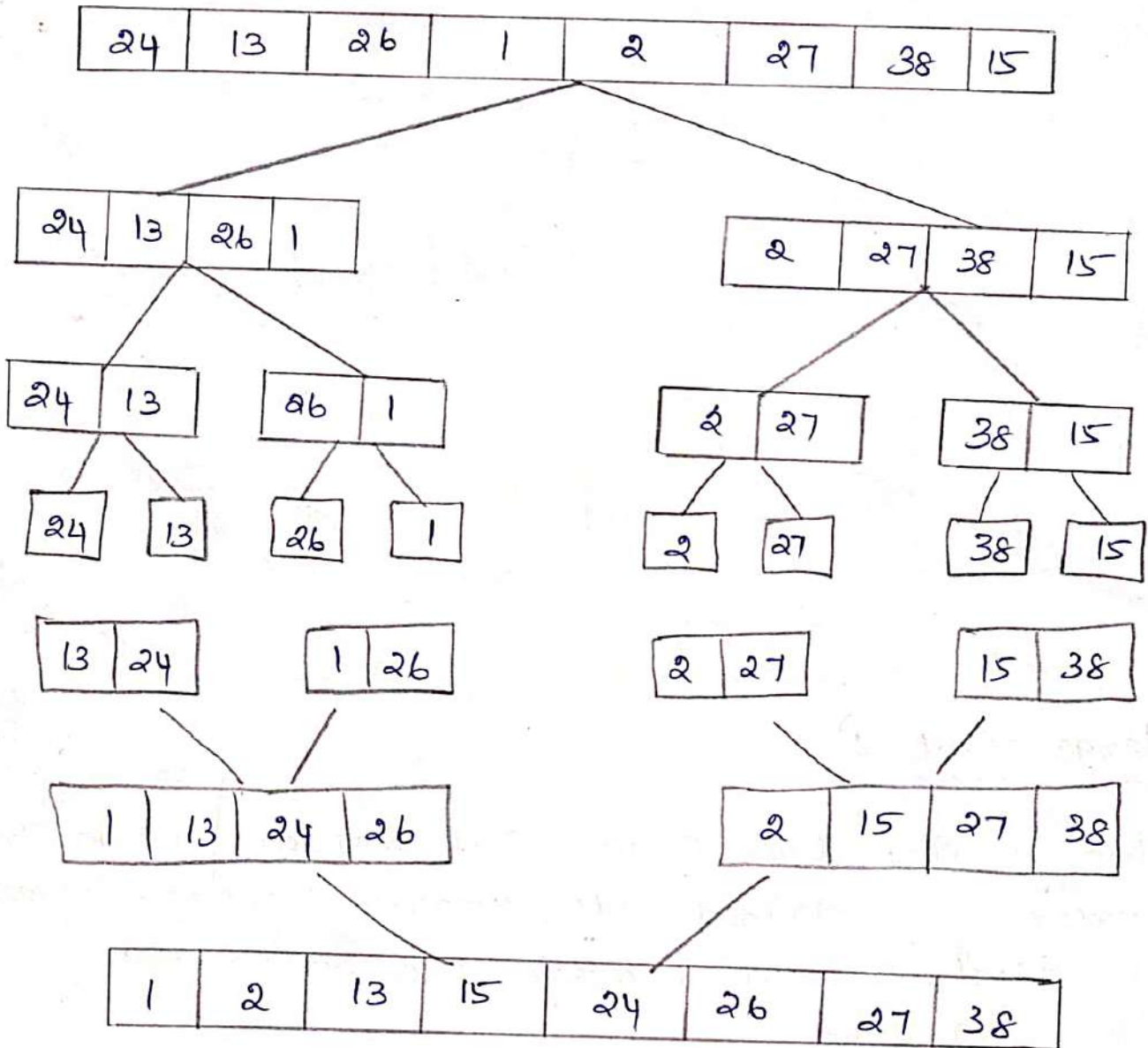
*) Merge Sort uses divide and conquer method. The problem is divided into smaller problems & solved recursively. Finally it merges two sorted lists.

Basic algorithm

- 1) Divide the given array of elements into two halves
- 2) Recursively sort the first half & second half of array elements
- 3) Take two input arrays A and B, an output array C
- 4) The first element of A array & B array are compared, then the smaller element is stored in output array C, the corresponding pointer is incremented.

5) This step is repeated until all the elements are compared.

Example: 24 13 26 1 2 27 38 15



Program

```
def mergesort(list):
```

```
    print("splitting", list)
```

```
    if len(list) > 1:
```

```
        mid = len(list) // 2
```

```
        lefthalf = list[:mid]
```

right half = list[mid :]

mergesort(left half)

mergesort(right half)

i = 0

j = 0

k = 0

while i < len(left half) and j < len(right half):

if left half[i] < right half[j]:

list[k] = left half[i]

i = i + 1

else:

list[k] = right half[j]

j = j + 1

k = k + 1

while i < len(left half):

list[k] = left half[i]

i = i + 1

k = k + 1

while j < len(right half):

list[k] = right half[j]

j = j + 1

k = k + 1

Print("merging", list)

list = [5, 2, 3]

mergesort(list)

Print(list)

Output

(5)

Splitting [5, 2, 3]

Splitting 5

merging 5

Splitting [2, 3]

Splitting [2]

merging [2]

Splitting [3]

merging [3]

merging [2, 3]

merging [2, 3, 5]

[2, 3, 5]

Suggested Questions / Assignments / Home works / any other

1. Write a Python Program to sort n numbers Using merge Sort. (AU- Dec 2019)
2. Write a Script in Python for histogram.

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 38 & 39 Illustrative Program

Topic(s) to be covered	Quick sort- Insertion Sort - Selection sort
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know & understand how to arrange the elements in order using Quick, Selection, Insertion	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen & Participate

Lecture Notes

Quick Sort

* Quick sort uses divide & conquer method

Basic algorithm

1. Choose Pivot element (usually last or random)
2. Stores elements less than pivot in left subarray.
3. Stores elements greater than pivot in right subarray.
4. Call quicksort recursively on left subarray.
5. Call quicksort recursively on right subarray.

Program

(54)

```
def quicksort(arr, left, right):  
    if left < right:  
        Partition-Pos = Partition(arr, left, right)  
        quicksort(arr, left, Partition-Pos - 1)  
        quicksort(arr, Partition-Pos + 1, right)
```

```
def Partition(arr, left, right):  
    i = left  
    j = right - 1  
    Pivot = arr[right]  
    while i < j:  
        while i < right and arr[i] < Pivot:  
            i = i + 1  
        while j > left and arr[j] >= Pivot:  
            j = j - 1  
        if i < j:  
            arr[i], arr[j] = arr[j], arr[i]  
    if arr[i] > Pivot:  
        arr[i], arr[right] = arr[right], arr[i]  
    return i
```

arr = [22, 11, 88, 66, 55, 77, 33, 44]

quicksort(arr, 0, len(arr) - 1)

Print(arr)

Output

[11, 22, 33, 44, 55,
66, 77, 88]

Selection Sort

(55)

* Selection sort is a simple sorting algorithm

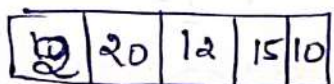
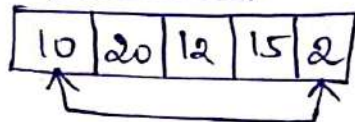
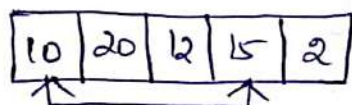
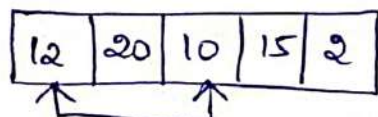
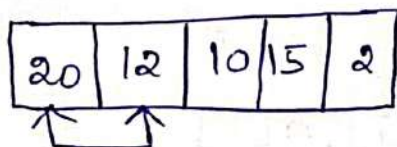
* Selection sort algorithm starts by comparing first two elements of an list & swapping if necessary

* If you want to sort the elements of list in ascending order if the first element is greater than second, then you need to swap the elements but if the first element is smaller than second leave the element as it is.

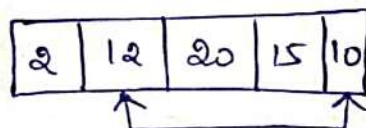
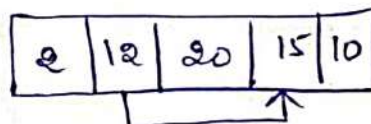
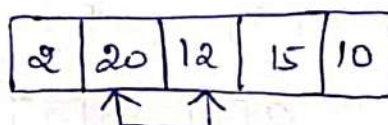
* Then first element & third element is compared & swapped if necessary. This process goes on until first element & last element of a list is compared

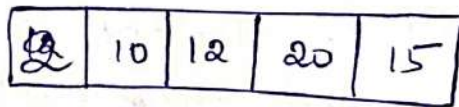
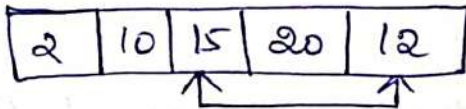
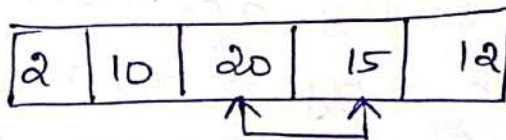
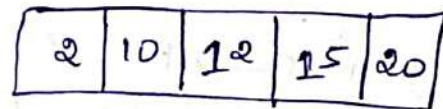
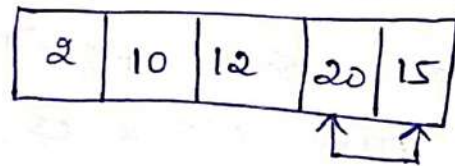
(Example) elements = [20, 12, 10, 15, 2]

Step 1:



Step 2:



Step 3:Step 4:Program

Selection Sort

a = []

n = int(input("Enter number of elements"))

for i in range(n):

element = int(input("enter element"))

a.append(element)

Print(a)

for i in range(len(a)):

for j in range(i+1, len(a)):

if a[i] > a[j]:

a[i], a[j] = a[j], a[i]

Print("Sorted list", a)

Output

Enter number of elements 5

enter elements 20

enter elements 12

enter elements 10

enter elements 15

enter elements 2

[20, 12, 10, 15, 2]

Sorted list: [2, 10, 12, 15, 20]

Insertion Sort

* The principle of insertion sort is quite simple.

* Each successive element in the list to be sorted is inserted into its proper place with respect to the other already sorted element

* Sort with second element & put it in its correct place so that the 1st & 2nd elements of the list are in order.

* This process continues until all the elements have been sorted.

(Example)

23	78	45	8	32	56
----	----	----	---	----	----

original list

Sorted

23

Unsorted

78	45	8	32	56
----	----	---	----	----

23	78
----	----

45	8	32	56
----	---	----	----

pass 1

23	45	78
----	----	----

8	32	56
---	----	----

pass 2

8	23	45	78
---	----	----	----

32	56
----	----

pass 3

8	23	32	45	78
---	----	----	----	----

56

pass 4

8	23	32	45	56	78
---	----	----	----	----	----

sorted list

Program Insertion Sort

(58)

```
a = [ ]
```

```
n = int (input ("Enter number of elements"))
```

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

```
    element = int (input ("Enter element"))
```

```
    a.append (element)
```

```
Print (a)
```

```
for i in range(1, len(a)):
```

```
    j = i
```

```
    while a[j-1] > a[j] and j > 0:
```

```
        a[j-1], a[j] = a[j], a[j-1]
```

```
        j = j - 1
```

```
Print ("Sorted list", a)
```

Output

Enter number of elements 5

Enter element 5

Enter element 3

Enter element 10

Enter element 2

Enter element 1

[5, 3, 10, 2, 1]

Sorted list [1, 2, 3, 5, 10]

Suggested Questions / Assignments / Home works / any other

1. Write a Python Program to sort n numbers using Quick Sort.
2. Write a Script in python to sort n numbers using selection sort.
3. Write a Python Program to sort n numbers using Insertion Sort.

	Text Books / Reference Books / Any other suggested Materials		
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 40, 41 Files

Topic(s) to be covered	Files - text files, reading & writing files - Rename & Remove methods
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know The file Concepts	Understand
LO2	Know how to open, read/write & close file methods	Understand

Teaching Learning Material	Student Activity
Chalk & Talk	Listen

Lecture Notes

Files

- * File is a named location on disk to store related information (Collection of data stored on a disk)
- * Files are used to store data permanently. These files can be used whenever required, it may be reopened, modified & copied.
- * Files can be accessed either by absolute file names (or) relative file names.

Absolute Filename: A file name can specify all the directory names starting from the root

(Example)

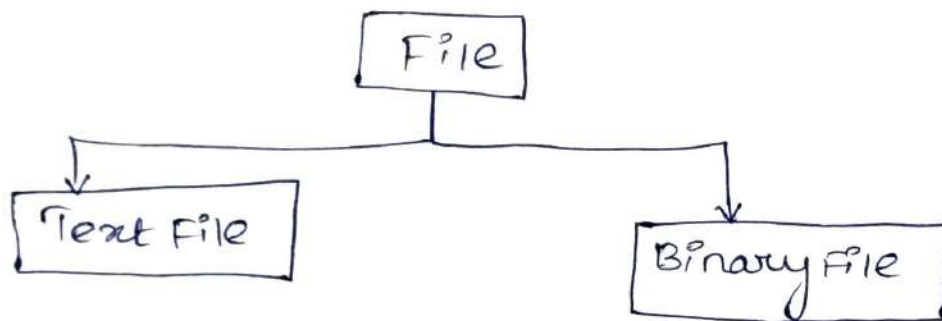
D:\Python\unit5.txt

Relative File Name: It is the relative to current working directory.

(Example) unit5.txt

Types of files

* In Python, a file is classified as either text or binary



Text File

- * A file that can be processed (ie) read, write, modify) using text editor such as notepad
- * Text files are stored in a form that is human readable
- * Text files are structured as a sequence of lines, where each line includes a sequence of characters.

③

Binary File

- * Other than text files, all other files are called binary file
- * Binary file stored in binary format
- * Binary file is computer readable but not human readable.

File Operations

- 1) Open the file
- 2) Read / write
- 3) Close the file

Opening a File

- * To read (or) write a file, it is necessary to open it using Python built in function named `open()` function.

Syntax

`file object = open(filename, [access mode])`

here,

filename → Name of the file to access

accessmode → How the file will be used
(either read or write etc.)

(Example)

```
f = open("file.txt", "r")
```


The list of different file opening modes

Modes	Description
r	Opens a file for reading only
rb	Opens a file for reading only in binary format
r+	Opens a file for both reading & writing
rb+	Opens a file for both readings writing in binary format
w	Opens a file for writing only
wb	Opens a file for writing only in binary format
wt	Opens a file for both writing & reading
wb+	Opens a file for both writing & reading in binary format
a	Opens a file for appending
ab	Opens a file for appending in binary format
at	Opens a file for both appending & reading
abt	Opens a file for both appending & reading in binary format

Fileobject Attributes

* Fileobject, to get various information related to that file

Attributes	Description
fileobject.closed	Returns true, if the file is closed otherwise false
fileobject.mode	Returns the file access mode
fileobject.name	Returns the name of the file

(Example)

```
f = open("file1.txt", "w+")
Print("name of the file", f.name)
Print("Closed or not", f.closed)
Print("opening mode", f.mode)
```

Output

name of the file : file1
 Close or not : False
 Opening mode : w+

Closing a File

*The close() method of a file object flushes any unwritten information & closes the file object, after which no more writing can be done

Syntax

fileobject.close()

(Eg) The program to perform the open & close operation of a file

```
f = open("file1.txt", "w+")
Print("name of the file", f.name)
f.close()
```

Output

name of the file : file1.txt

⑥

Writing to a File

- * The `write()` method is used to write a string to an already opened file.
- * To write into a file, it is needed to open a file in write 'w' & append 'a'
- * The 'w' mode will overwrite a file if it already exists. So all previous data are erased.

Syntax

`fileobject.write(String)`

(Example)

```
f1 = open("Sample.txt", "w")
```

```
Str = "Problem solving & Python Programming"
```

```
f1.write(Str)
```

```
f1.close()
```

```
f2 = open("Sample.txt", "r")
```

```
line = f2.readline()
```

```
Print("The Content of Sample.txt:", line)
```

```
f2.close()
```

Output

The Content of Sample.txt:

Problem solving & Python Programming

Sample.txt
file
Problem solving
Python Programming

writelines() method

* The writelines() method is used to write a list of strings.

Example

```
f1 = open("file1.txt", "w")
lines = ["Hello", "Students", "welcome to python",
         "Enjoy learning"]
f1.writelines(lines)
f1.close()
Print("writing to file...")
```

O/P

writing to file

Append() method

* Append() method is used to add the content at end of file. It doesn't overwrite already existing content. Just add at end of previous content.

Example

```
f1 = open("file1.txt", "w")
L = ["This is Delhi\n", "This is Paris\n", "This is London"]
f1.writelines(L)
f1.close()
```



```
f2 = open("file1.txt", "a")
```

```
f2.write("Today\n")
```

```
f2.close()
```

```
f3 = open("file1.txt", "a")
```

```
Print("Output of readlines after appending")
```

```
Print(f3.read())
```

```
f3 f3.close()
```

Output

output of readlines after appending

This is Delhi

This is Paris

This is London Today

Reading from a File

* To read the content of a file, it must be opened in read mode.

* There are 3 types of methods for reading

1) Reading a file using read(size) method

2) Reading a file using for loop

3) Reading a file using readline() method

Reading a file using read(size) method

* The read(size) method is used to read in size number of data

* If size not specified, it reads & returns upto the end of the file.

Syntax

fileobject.read([size])

It reads from the beginning of the file until the size given. If no size is provided, it ends up reading until the end of the file.

Example

```
f = open("file1.txt", "wt")
```

```
str1 = "Python is a programming language"
```

```
f.write(str1)
```

```
f.close()
```

```
f = open("file1.txt", "rt")
```

```
str2 = f.read(20)
```

```
print("The string read is:", str2)
```

```
f.close()
```

Output

The string read is: Python is a programm

Reading a file using for loop

* A file can be read line-by-line using a for loop. This is efficient & fast.

(eg)

```
f = open("file1.txt", "r")
for line in f:
    print(line, end=" ")
```

o/p

Hello
Students
Welcome to Python
Enjoy learning

Reading a file using readline() method

* The readline() method is used to read individual lines of a file. This method reads a file till the newline, including the newline character.

(eg)

```
f = open("file1.txt", "r")
f.readline()
f.close()
```

o/p

'Hello\n'
'Students\n'
'Welcome to Python\n'
'Enjoy learning\n'

File Position

- * The `tell()` method gives the current object position within the file.
- * The `Seek (offset, from)` method modifies the current file position.
 - The offset specifies the number of bytes to be moved
 - from specifies the reference

(eg)

```
f=open("file1.txt", "w+")
Str1="Python is a programming language"
f.write(Str1)
f.close()
f=open("file1.txt", "r+")
Str2=f.read(20)
Print("The string read is:", Str2)
Pos=f.tell()
Print("Current file position:", Pos)
Pos=f.seek(0,0)
Str3=f.read(10)
Print("Again the string read is:", Str3)
f.close()
```

Output

The string read is: Python is a Programm
 Current file position: 20
 Again the string read is: Python is

Renaming & Deleting Files

* Python os module provides methods that enable us to perform file processing operations like renaming & deleting a file.

* To use this module it is necessary to import os module.

The rename() method

* The rename() method takes two arguments, the current file name & the new file name.

Syntax

`os.rename(current filename, new filename)`

(Eg) `import os`
`os.rename("file1.txt", "file2.txt")`

The remove() method

* The remove() method can be used to delete files by supplying the name of the file.

Syntax

`os.remove(filename)`

(Eg)

`import os`
`os.remove("file2.txt")`

Suggested Questions / Assignments / Home works / any other

1. Write the syntax for opening a file to write in binary mode. (AU-Dec 22)
2. Name the different access modes for opening a file & present an outline of the same. (AU-Dec 22)
3. How will you update the content of one file to another file in Python? (AU-Dec 20)
4. Write methods to rename & delete files. (AU-Dec 19)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 42 Command line arguments, Format operator

Topic(s) to be covered	Command line arguments- sys module - format operator
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Know how to execute Command line arguments	Apply
LO1	Understand Format operator	understand

Teaching Learning Material	Student Activity
Chalk & Talk / ICT Tool / Any other	Listen / Participate / Discuss / Peer to Peer Learning / Quiz / Role Play / Any other

Lecture Notes

Command line argument

*The main objectives of this concept are

- 1) To run a python script in command line
- 2) To capture the arguments at run time in command line.
- 3) To parse command line arguments

- * Command line arguments are input Parameters Passed to the Script when executing them.
- * All programming language provides support for Command line arguments.
- * There are many options to read Python Command line arguments. The most common module is "Sys".

Sys module:

- * This module Stores the Command line arguments into a list, we can access it using sys.argv
- * This is a very useful & simple way to read Command line arguments as String.

(Example) 1

```
Import sys
```

```
Print ("Number of arguments:", len(sys.argv))
```

```
Print ("Argument List:", Str(sys.argv))
```

Output

```
C:\Python310> Python arg1.py 10 20 30
```

```
Number of arguments : 4
```

```
Argument List : ['arg1.py', '10', '20', '30']
```

- * The first argument is the name of the program followed by the other arguments 10 20 & 30

(Example 2)

```

import sys
n = len(sys.argv)
Print("Total arguments passed:", n)
Print("Name of python script:" sys.argv[0])
Print("Arguments Passed:", end=" ")
for i in range(1, n):
    Print(sys.argv[i], end=" ")
Sum = 0
for i in range(1, n):
    Sum += int(sys.argv[i])
Print("Result:", sum)

```

Output

C:\Python310>python args.py 1 5 7

Total arguments Passed: 4

Name of Python Script: args.py

Arguments passed: 1 5 7

Result: 13

	args.py	
argv[0]	args.py	App Name
argv[1]	1	
argv[2]	5	
argv[3]	7	

Format Operator

- * One of the Python's feature is String format Operator %.
- * The % Operator is used to format a set of Variables enclosed in a tuple.

Syntax

% Conversion type % values

Some of basic format symbols are

%s → String

%d → Integers

%c → Character

%f → Floating Point

%. <number of digits>f → Floating Point with fixed amount of digits to the right of the dot

1. String (%s):

```
text = "kiwi"
```

```
Print("Hello %s" % text)
```

Output

Hello kiwi

2. Integer (%d):

text = "kiwi"

age = 3

Print ("%s is %d years old" %(text, age))

Output

kiwi is 3 years old

3. Float (%f):

Pi = 3.141592

Print ("The value of pi is %f" % Pi)

Output

The value of pi is 3.141592

4. Floating point with a fixed amount of digits (%.nf)

Slope = 1.12345

Print ("Slope : %.3f" % slope)

Output

Slope : 1.123

(Examples) 1

```
Print("%d %d %d" %(1, 2))
```

Output

Type error: not enough arguments for format String

Example 2

```
Print("%d" . "%dollars")
```

Output

Type error: %d format: a number is required not str.

Suggested Questions / Assignments / Home works / any other

1. How to use Command line arguments in Python? (AU-Dec 2019)
2. What is format operator.
3. Discuss about the use of format Operator. (AU-Dec 2019)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 43, 44 Exception

Topic(s) to be covered	Errors and Exception - Handling Exception - try - except Statements
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Describe the errors & types	Understand
LO2	Know how to handle the exception	Understand

Teaching Learning Material	Student Activity
Chalk & Talk / ICT Tool / Any other	Listen / Participate / Discuss / Peer to Peer Learning / Quiz / Role Play / Any other

Lecture Notes

Errors and Exception

(Note: "Errors" topic refers to "debugging" concept in UNIT-II)

Exceptions

- * An exception is an error that occurs during execution of a program.
- * It is also called as runtime errors

Example

- * Division by Zero
- * Using an Identifier, which has not been defined
- * Accessing a list, tuple element, dictionary key which doesn't exist.
- * Try to access a file which doesn't exist.

(eg)

a=10

b=0

c=a/b

Print(c)

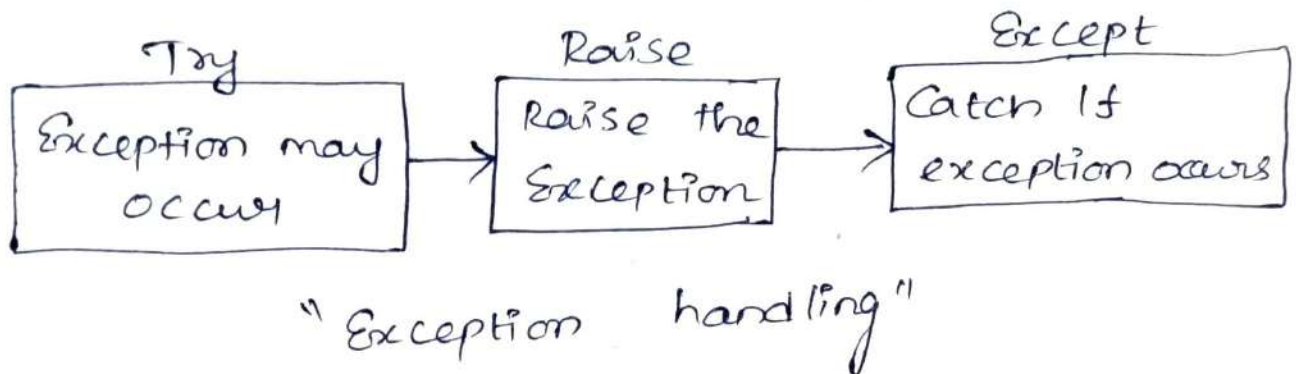
Error message: ZeroDivisionError

Common built-in Exceptions in Python

Exception Name	Description
ZeroDivisionError	Raised when division by zero
EOF Error	Raised when there is no input from input() function
Import Error	Raised when the import module is not found.
Index Error	Raised when the index is not found in sequence
Key Error	Raised when a key is not found in a dictionary.
Name Error	Raised when identifier not found
IO Error	Raised when input/output operation fails
Value Error	Raised when invalid values.

Handling Exceptions

- * The simplest way to handle exceptions is with a "try-except" block.
- * Exceptions that are caught in try blocks are handled in except block.



Syntax

try:

Statements

except ErrorName:

Statements

try statement work as follows

- 1) First try statement is executed
- 2) If no exception occurs, the except block is skipped & execution of the try statement is finished.
- 3) If an exception occurs during execution of the try block, the rest of block is skipped & control transferred down to except block based on except keyword.

(Example)

x=10

y=0

try :

z = x/y

Print (z)

except ZeroDivisionError:

Print("divide by zero")

Output

divide by zero

A try Statement may have more than one
except class.

Syntax:

try :

Statements

except Errorname1:

handler code

except Errorname2:

handler code

Simple example to handle multiple exception

try :

a=int(input("enter the dividend:"))

b=int(input("enter the divisor:"))

c = a/b

Print(c)

except ZeroDivisionError:

Print("The divisor may not be Zero")

except ValueError:

Print("The dividend & divisor have to be numbers")

Output 1 (successful)

enter the dividend: 20

enter the divisor: 5

4

Output 2 (unsuccessful)

enter the dividend: 20

enter the divisor: 0

The dividend may not be zero

else & finally Statements

* Two blocks (clause) that can be added optionally to try - except block are else and finally.

else

→ else will be executed only if the try clause doesn't raise an exception.

finally

→ finally will be executed regardless of whether an exception occurs & even if we exit the block.

Output (3) unsuccessful

enter the dividend: 20

enter the divisor: 5

The dividend & divisor have to be numbers.

(example)

try:

age = int(input("Please enter your age"))

except ValueError:

Print("Hey, that wasn't a number")

else:

Print("You are %d years old", %age)

finally:

Print("Good bye")

Output 1

Please enter your age: 20

You are 20 years old

Good bye

Output 2

Please enter your age: a

Hey, that wasn't a number

Good bye

Raising Exception

- * The raise Statement Initiates a new exception
- * It allows The programmer to force a specified exception to occur.
- * Its Commonly used for raising user defined exception.

Syntax

```
raise Exception
```

Example

```
try:
    raise ValueError
except ValueError:
    Print("error")
```

Output

error

User defined Exception

- * Python allows the user to create their custom exception by creating a new class

Example

```
Class Error(Exception):
    Pass

Class PosError(Error):
    Pass
```



```
Class NegError(Error):
```

```
    Pass
```

```
try:
```

```
    number = int(input("enter a number"))
```

```
    if number < 0:
```

```
        raise NegError
```

```
    else:
```

```
        raise PosError
```

```
    break
```

```
except PosError:
```

```
    print("This value is positive")
```

```
except NegError:
```

```
    print("This value is negative")
```

Output 1

enter a number: 10

This value is positive

Output 2

enter a number -5

This value is ~~neg~~ative.

Suggested Questions / Assignments / Home works / any other

1. what are exception?
2. Find the syntax error in the given code

```
while True print("Hello world")
```

(AU-Dec 2019)
3. what are exceptions? Explain the method to handle them with example. (AU-Dec 2019)
(AU-Dec 2022)
4. Appraise the use of try & except block in python with syntax. (AU-Dec 2018)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 45 Modules

Topic(s) to be covered	Modules - Creating a module - Locating module - Types of module.
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
Lo1	understand the modules	understand
Lo2	know how to import modules in Python program	understand

Teaching Learning Material	Student Activity
Chalk & Talk / ICT Tool / Any other	Listen / Participate / Discuss / Peer to Peer Learning / Quiz / Role Play / Any other

Lecture Notes

Modules

- * A Module is a python file consisting of a python code that can define functions, classes & variables related to particular task.
- * A module allows us to organize our code by grouping related code, which make the code easier to understand & use.
- * Any python source file can be imported as a module into another python source file.

Create a Module

* Create a module is quite easy.

* Just write a python program & save the program with .py extension

example

Support.py

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

```
    c = a + b
```

```
    print(c)
```

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

```
    c = a - b
```

```
    print(c)
```

⇒ The file containing python code Support.py is called module.

⇒ The module name should be Support

⇒ This process is called Modularity.

Locating Modules

* When a module is imported, Python interpreter searches for the module in the following sequence.

1) The current directory

2) Searches for Python Path [.py]

3) Checks the location, where Python has been installed.

Types of Modules

There are two types of modules

1) Built in module

2) User Defined Module

Built in Module

→ It is predefined module that are already available in Python.

(eg) Math module

Random module

Date-time module

User Defined Module

→ The user creates the user defined module to ease complex task into simpler.

(Eg) Support module.

Import module in a Program

There are 5 ways to import module

i) `import module-name`

(Eg) `import Numpy`

2) `import module-name as alias name`

(Eg)

`import Numpy as np`

3) `from module-name import Var-name`

(Eg)

`from math import Sqrt`

4) `from module-name import Var-name as alias name`

(Eg) `from math import multiply as Product`

5) `from module-name import *`

(Eg) `from math import *`

We can use add, sub, multiply, divide, Gcd, Sqrt... under math

Different types of modules

1) Math module: This module provide access to the mathematical functions.

2) Open Cv Module: This module is used to Capture video from webcam

3) Sys module: Used to get Command line arguments.

4) JSON Module: Java Script Object Notation is used to web development for encoding & decoding.

5) Matplot Module: A library for creating data visualization.

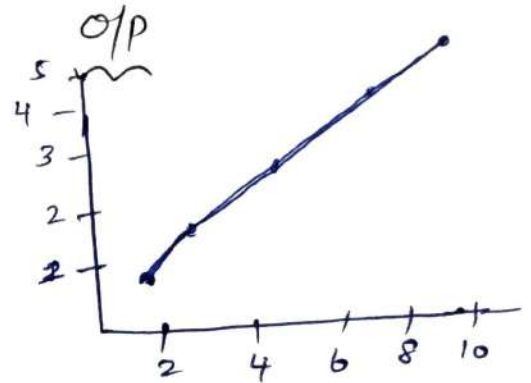
(Eg) Import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [10, 8, 6, 4, 2]

plt.plot(x, y)

plt.show()



6) Random module:

Generates Pseudo random number for various distribution

(Eg) Import random

num = random.randint(1, 10)

Print(num)

O/p
6

7) Datetime module: It is used to work with dates & time in the recent past to nearer future.

(Eg) Import datetime

now = datetime.datetime.now()

Print("Date & Time", now)

O/p

1.03.2023

09:09

8) Numpy Module: It is used to deal with array & matrices

(Eg) Import numpy as np

arr = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9])

arr *= 2

Print(arr)

O/p

2	4	6
8	10	12
14	16	18

Suggested Questions / Assignments / Home works / any other
1. what are the different modules in Python. (AU- Dec 2022)
2. what is a module? Give example. (AU- Dec 2019)
3. write notes on modular design. (AU- Jan 2018)
4. what are modules in Python? How will you import them? Explain the concept by creating & importing a module.

	Text Books / Reference Books / Any other suggested Materials		
	S.No	Title	Publisher
	1	Think Python: How to Think like a Computer Scientist	Allen B. Downey 2nd Edition, O'Reilly Publishers, 2016.
	2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher 1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 46 Packages

Topic(s) to be covered	Packages - Creating Packages Importing modules from package
------------------------	--

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
L01	understand the packages	understand
L02	know how to import modules from packages	understand

Teaching Learning Material	Student Activity
Chalk & Talk / ICT Tool / Any other	Listen / Participate / Discuss / Peer to Peer Learning / Quiz / Role Play / Any other

Lecture Notes

Packages

- * Collection of modules
- * when we have a large number of Python modules they can be organized into packages
- * Such that similar modules are placed in one package and different modules are placed in different packages.

Creating Package

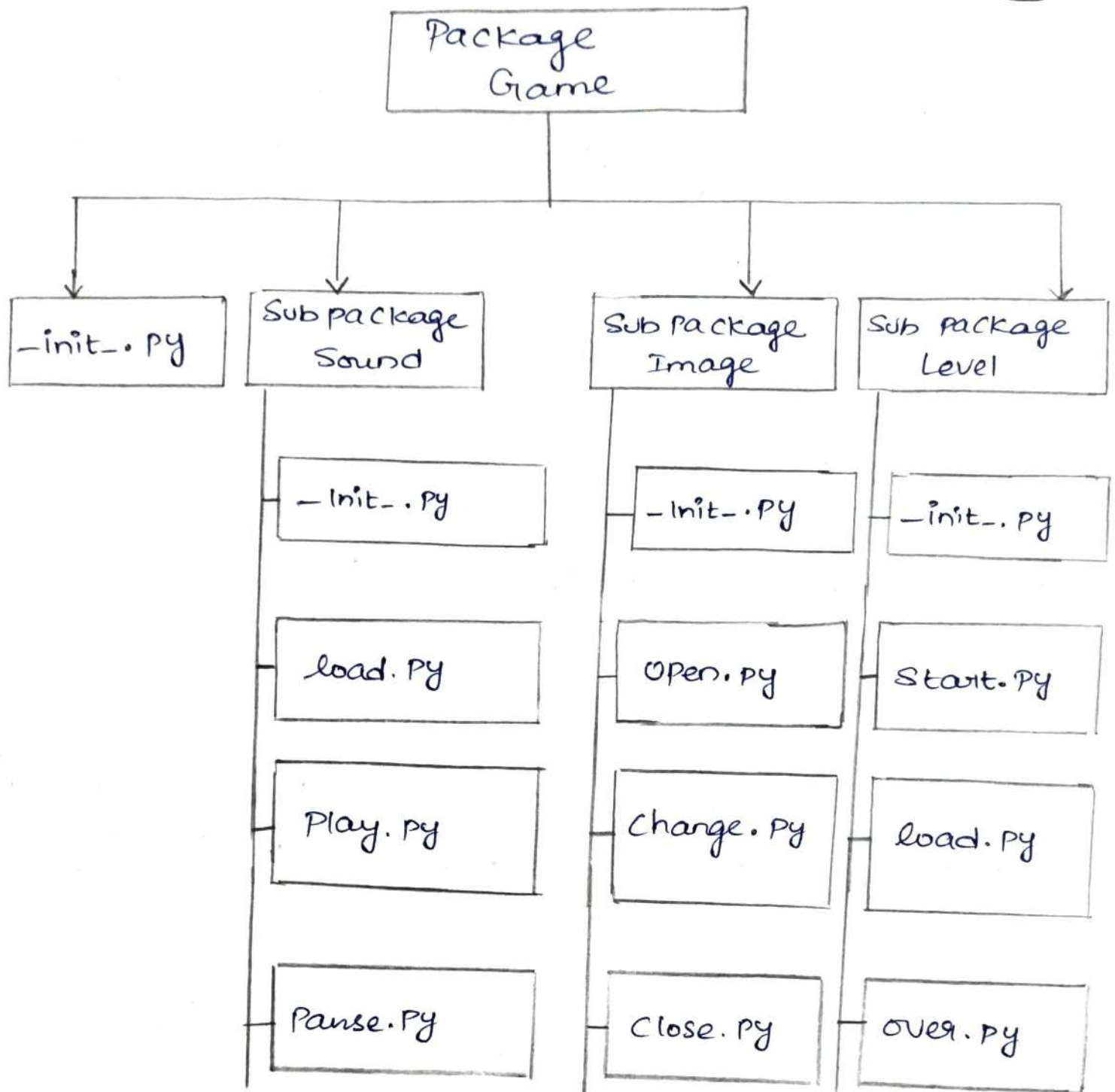
- i) Create a directory & name it with a Package name
- ii) Keep Subdirectories (Subpackages) & modules in it.
- iii) Create `-init-.py` file in the directory

⇒ The `-init-.py` file can be empty but generally the Initialization code is placed for that Package in this file.

⇒ This file is necessary since Python will know that this directory is a Package directory other than an ordinary directory.

Example

To develop a game, one possible organization of Packages and modules could be done as shown in below



"Organization of Packages & modules"

Importing modules from Package

*The modules are imported from Packages using the dot (.) Operator.

*To import the start module use,

```
import Game.Level.Start
```

⇒ If this module contains a function named `Select-difficulty()`, use the full name to reference it

```
Game.Level.Start.Select-difficulty()
```

⇒ Import the module without the package prefix.

```
from Game.Level import Start
```

We can now call the function simply as follows

```
Start.Selectdifficulty()
```

⇒ Another way of importing, just the required function from a module within package

```
from Game.Level.Start import selectdifficulty
```

Now we can directly call this function

```
selectdifficulty()
```


Suggested Questions / Assignments / Home works / any other

1. what are packages?
2. Explain with an example package & how to import package in python program.

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.

Lecture No. 47, 48 Illustrative Programs

Topic(s) to be covered	Voter's age validation, Student marks range validation, word count, copy file
------------------------	---

	Lecture Outcome (LO)	Bloom's Level
	At the end of this lecture, students will be able to	
LO1	Apply exception handling concept for voter's age validation, mark range validation	Apply

Teaching Learning Material	Student Activity
Chalk & Talk / ICT Tool / Any other	Listen / Participate / Discuss / Peer to Peer Learning / Quiz / Role Play / Any other

Lecture Notes

Illustrative Programs

1) Voter's age validation (using exception handling)

try:

age = int(input("Please enter your age:"))

If age >= 18:

Print("You are eligible to vote")

else:

Print("You are not eligible to vote")

except ValueError:

Print ("Invalid input. Please enter a valid age.")

Output:

Please enter your age 18

You are eligible to vote

Voters age validation (Using Modules)

import datetime

year-of-birth = int(input("Enter your year of birth"))

Current-year = datetime.datetime.now().year

Current-age = Current-year - year-of-birth

Print("your current age is", Current-age)

If (Current-age $\geq$ 18):

Print("you are eligible to vote")

else:

Print("you are not eligible to vote")

Output

Enter your year of birth 2015

Your current age is 8

You are not eligible to vote.

2) Marks range validation (0-100)

try:

mark = int(input("Enter student's mark:"))

If mark < 0 or mark > 100:

raise ValueError("Mark must be between
0 and 100")

else:

Print("mark is valid")

except ValueError as e:

Print("Error:" + str(e))

output 1

Enter student's mark: 100

Mark is valid

output 2

Enter student's mark: 105

Error: mark must be between 0 and 100

3) Counting number of words in String

```
String = input("Enter any String")
```

```
word_length = len(String.split())
```

```
Print("Number of words = ", word_length)
```

Output

Enter any String: Problem Solving and Python
Programming

Number of words = 5

Counting number of words in file

```
file = open("Input.txt", "r")
```

```
data = file.read()
```

```
word = data.split()
```

```
Print("number of words in text file:", len(word))
```

Output

number of words in text file: 5

4) Copy The Content of one file to another
file

```
sfile = input("Enter the source file")
```

```
tfile = input("Enter the target file")
```

```
filehandle = open(sfile, "r")
```

```
text = filehandle.readlines()
```

```
filehandle.close()
```

```
filehandle = open(tfile, "w")
```

```
for s in text:
```

```
    filehandle.write(s)
```

```
filehandle.close()
```

```
Print("file content copied successfully")
```

Output

Enter the source file: source.txt

Enter the target file: target.txt

file content copied successfully.

Suggested Questions / Assignments / Home works / any other

1. Write a Python Script to display The Current date and time. (AU- Dec 2018)
2. Write a Python Program to Count the number of words in a text file (AU- Dec 2019)

 Text Books / Reference Books / Any other suggested Materials			
S.No	Title	Author	Publisher
1	Think Python: How to Think like a Computer Scientist	Allen B. Downey	2nd Edition, O'Reilly Publishers, 2016.
2	Computational Thinking: A Beginner's Guide to Problem Solving and Programming	Karl Beecher	1st Edition, BCS Learning & Development Limited, 2017.