TOUCHPAD
Computer Textbook Series

11

Informatics Practices with PYTHON

(Focus on Documentation)

Beta*
*Updated Copy Coming Soon

ORANGE

Class XI

Answer Key

Informatics Practices

1. Introduction to Computer Systems

Assessment

A.

1. c	2. b	3. b	4. a	5. b
6. d	7. c	8. c	9. d	10. c
11. c	12. d	13. b	14. c	15. a

B.

1. False	2. True	3. False	4. False	5. True
6. True				

C.

1. hardware, software	2. CPU	3. scanner	4. control unit
5. cache memory	6. system bus	7. High-level	8. device driver
9. operating system			

D. 1. **Advantages of the Third-Generation of Computers:**

- Smaller in size as compared to previous generations.
- More reliable as compared to previous generations.
- Used less energy and produced less heat as compared to previous generations.

Disadvantages of Third-Generation of Computers:

- Size of memory was still limited for several applications in business and defense.
- It was difficult to repair ICs.

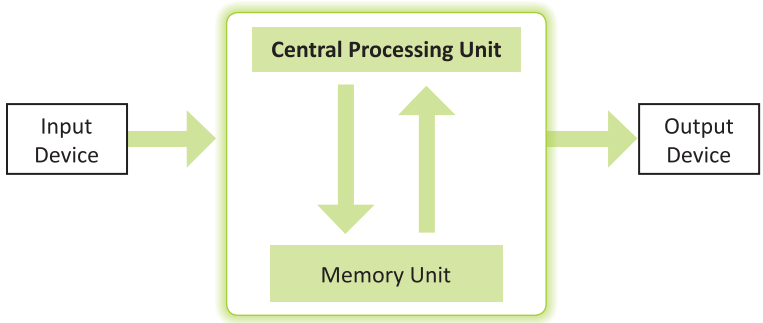
2. **Fourth Generation Computers:**

- Based on microprocessors.
- LSI (Large Scale Integration) and VLSI (Very Large Scale Integration) technology.
- Introduction of personal computers (PCs).
- Used for general-purpose tasks and networking.

Fifth Generation Computers:

- Based on artificial intelligence and parallel processing.
- Use of ULSI (Ultra Large Scale Integration) technology.
- Emphasis on natural language processing and machine learning.
- Development of advanced robotics and expert systems.

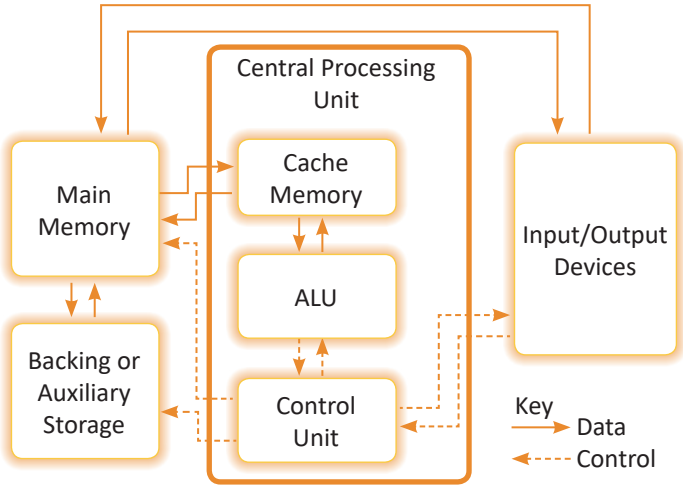
3. (i) EDSAC - Electronic Delay Storage Automatic Calculator
   (ii) GUI - Graphical User Interface
   (iii) VLSI - Very Large Scale Integration
4. A computer is an electronic device that accepts input data from the user, processes it, and produces results. It comprises of the following components:
   a. Input Unit
   b. Central Processing Unit (CPU)
   c. Memory Unit
   d. Output Unit
5. Hardware and software are both essential for a computer to function. Hardware refers to the physical components of a computer, such as the CPU, memory, storage devices, and input/output devices. Software, on the other hand, consists of the programs and operating systems that run on the hardware. Without hardware, software cannot operate, and without software, hardware is useless. Together, they enable the computer to perform tasks such as processing data, executing applications, and managing resources.
6. Computer is an electronic device that accepts input data from the user, processes it, and produces results. It comprises of the following components:
   - Input Unit
   - Central Processing Unit (CPU)
   - Memory Unit
   - Output Unit



7. A light pen is a pointing device that allows users to interact with a computer screen by detecting light from the screen, whereas a mouse is a handheld pointing device that detects two-dimensional motion relative to a surface. A light pen is used directly on the display surface, while a mouse is used on a separate flat surface.
8. A light pen is a pointing device that allows users to interact with a computer screen by detecting light from the screen, whereas a mouse is a handheld pointing device that detects two-dimensional motion relative to a surface. A light pen is used directly on the display surface, while a mouse is used on a separate flat surface.

9. The Central Processing Unit (CPU), also called the processor, is the computer's brain and is responsible for carrying out all the processing in the computer system. Given a task, the CPU is provided with a sequence of instructions in the form of a program.

   The CPU controls the entire data flow and instructions inside the computer. To facilitate functioning, the CPU comprises the following components:



- **Registers:** While processing an instruction, the CPU needs to store in its local storage the instruction, memory addresses being referred during the instruction execution, data operands (on which operation is to be carried out), and the result of the computation. For storing and accessing the information mentioned above, the CPU requires high-speed temporary storage units, called registers. Several registers are also equipped with circuitry for performing elementary operations like addition and subtraction. However, as the registers are much more expensive than the main memory, the CPU has only a limited number of them.
- **Arithmetic Logic Unit (ALU):** The arithmetic Logic Unit (ALU) of the CPU performs all arithmetic (+,-,*,/) and logical (>,<,>=,<=,<>) operations. While the result of an arithmetic operation is a numeric value, the result of a logical operation (such as 7<8) is either True or False. The values True and False are called Boolean values in honour of George Boole, who developed an algebra (again named, Boolean algebra, in his honour) that constitutes the basis of all computer computations. The data for executing an instruction may already be available in the registers or fetched in the ALU registers from the computer's memory. The result of a computation is often stored in the computer's memory.
- **Control Unit (CU):** The Control Unit (CU) controls the execution of instructions and the flow of data amongst the components of a computer, i.e., from input devices to memory, memory to ALU and vice versa, and from memory to output devices. It sends instructions in the form of control signals to ALU to perform the required arithmetic and/or logical operations.

10. Biometric sensors are used in:
    - **Security systems:** To authenticate users based on their unique biological traits, such as fingerprint, retina, or facial recognition.
    - **Time and attendance systems:** To track employee attendance and working hours using biometric data.

11. i. 1 GB = 2^{20} KB
    ii. 1 YB = 2^{50} GB
    iii. 1 KB = 2^{-30} TB
    iv. 1 PB = 2^{30} MB

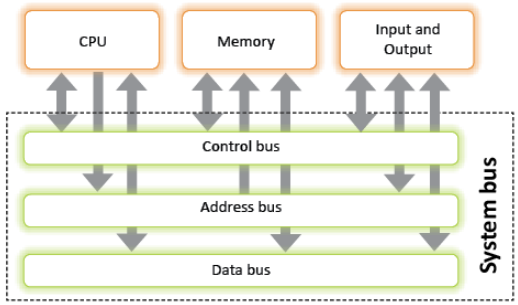
12. (i) **Primary memory and secondary memory:** Primary memory (RAM) is fast, volatile memory used for storing data temporarily while the computer is running. Secondary memory (HDD, SSD) is slower, non-volatile memory used for long-term storage of data and programs.

    (ii) **System software and application software:** System software includes the operating system and utility programs that manage computer resources and provide a platform for running application software. Application software includes programs designed to perform specific tasks for users, such as word processors, web browsers, and games.

    (iii) **Open source software and proprietary software:** Open source software is software with source code that anyone can inspect, modify, and enhance. Proprietary software is software that is owned by an individual or company and is distributed under a license that restricts modification and sharing.

13. Data is transferred between different components of a computer system (CPU to primary memory and vice versa or primary memory to secondary memory, or cache memory to CPU and vice versa) using physical connections called buses. There are three types of buses:
    - **Data Bus:** It is used to transfer data between different CPU components.
    - **Address Bus:** It is used to transfer addresses of memory locations for the CPU to perform read/write operations in the memory.
    - **Control Bus:** It is used to transfer control signals between various computer components.

    The data bus, address bus, and control bus together constitute the system bus. Figure 1.6 depicts the flow within the system bus. Data entered from an input device is stored in the main memory. The CPU then accesses this data using a data bus. The address of the memory location from where data will be accessed is communicated via the address bus. The read or write instructions are communicated through the control bus. Similarly, to write data into the memory, the CPU places the data on the data bus, and the address of the memory location where the data has to be written is placed on the address bus. Once this is done, a control signal is issued to write the data from the CPU to the designated memory location.



14. The functions of an operating system include:
    - **Process management:** Manages the creation, scheduling, and termination of processes.
    - **Memory management:** Allocates and deallocates memory space as needed by programs.
    - **File system management:** Manages the creation, deletion, and access of files on storage devices.
    - **Device management:** Controls and coordinates the use of hardware devices.
    - **User interface:** Provides a way for users to interact with the computer, such as through a graphical user interface (GUI) or command-line interface (CLI).
    - **Security and access control:** Protects data and resources from unauthorized access.
15. Three types of user interfaces are:
    - **Graphical User Interface (GUI):** Uses visual elements like windows, icons, and menus for interaction.
    - **Command-Line Interface (CLI):** Uses text-based commands for interaction.
    - **Touch User Interface (TUI):** Uses touch-based inputs for interaction, commonly found on smartphones and tablets.

Assertion and Reasoning Based Questions

1. a.
2. c.
3. a.
4. b.

Case-based Questions

1. (i) Joey should use presentation software to create his presentation. An example of such software is Microsoft PowerPoint.

   (ii) The school is using customized application software. This is because the software has been specifically modified or developed to meet the specific needs and feedback of the students.

(iii) There could be several reasons for this issue:

- The printer driver may not be installed or may be outdated. Installing or updating the printer driver can solve this problem.
- The printer may not be properly connected to the computer or network.
- There could be a hardware issue with the printer or the computer's USB port.
- The printer might be out of paper or ink, or there could be a paper jam.
- The printer may not be set as the default printer in the computer's settings.

2. Getting Started with Python Programming

Assessment

A. 1. c 2. b 3. d 4. a 5. b

B. 1. 2. 3. 4. 5.

C. 1. platform 2. script 3. .py 4. F5 5. print()

D. 1. Python has been developed under the Open-Source Initiative (OSI). Its license is administered by the Python Software Foundation, a non-profit corporation. One can download the source code, modify it, and then re-distribute the modified code.

2. Python libraries are collections of modules and packages that provide pre-written code to perform common tasks, saving developers time and effort. They extend the functionality of Python, enabling developers to perform a wide range of operations such as data analysis (e.g., Pandas), machine learning (e.g., Scikit-Learn), web development (e.g., Django, Flask), scientific computing (e.g., NumPy, SciPy), and more.

3. Python is easier to debug due to its simplicity and readability. The language's clear syntax makes it easier to spot errors. Python's dynamic typing and interpreted nature also allow developers to test and debug code interactively in real time, making it easier to identify and fix issues.

4. a. Open IDLE:
   - Click on the Start menu.
   - Scroll down to the Python folder and click to expand it.
   - Click on IDLE (Python 3.x) to open IDLE.

   b. Open the Python Script:
   - In IDLE, click on the File menu in the menu bar.
   - Select Open... from the dropdown menu.
   - In the file dialog that appears, navigate to the directory where your Python script is saved.
   - Select your Python file (e.g., script.py) and click Open. This will open the script in a new editor window within IDLE.

IMPRINT 1

c. Run the Script:

- With your script open in the IDLE editor window, click on the Run menu in the menu bar of the editor window.
- Select Run Module from the dropdown menu, or simply press the F5 key on your keyboard.
- The script will execute, and any output or error messages will be displayed in the IDLE shell window.

5. Python is considered a dynamically typed language because variable types are determined at runtime rather than at compile time. This means you do not need to explicitly declare the type of a variable when you create it. For example:

   var = 10 # var is an integer

   var = "Hello" # var is now a string

   The type of var can change during the execution of the program

6. Python programs are more prone to runtime errors because of the dynamic typing system. Errors related to type mismatches, missing attributes, or incorrect function arguments may not be detected until the code is executed. Without static type checking at compile time, these errors can only be caught during runtime, which can lead to more frequent runtime errors compared to statically typed languages.
7. IDLE stands for Integrated Development and Learning Environment.

E. 1. Download and Install Python

To download and install Python on a Windows system, follow these steps:

i. **Download Python:**

- Go to the official Python website: python.org.
- Click on the **Downloads** menu.
- Click on the **Download Python 3.x.x** button (the latest version).

ii. **Install Python:**

- Run the downloaded installer.
- Make sure to check the box that says **"Add Python 3.x to PATH"**.
- Click on **Install Now**.
- Follow the prompts to complete the installation.

To check the installed version of Python:

- Open the **Command Prompt**.
- Type python --version and press Enter.

2. **In Python Shell:**

- Open IDLE (Python GUI).
- In the Python Shell, type:

```
print("Your Name")
```

- Press Enter.

In a Script:

- Open IDLE.
- Click on **File** → **New File**.
- In the new file window, type:

  python

  Copy code

  ```
  print("Your Name")
  ```

- Save the file with a .py extension, e.g., `name_script.py`.
- Click on **Run** → **Run Module** or press **F5** to execute the script.

3. (i)

```
>>> 13/6+5*4
    22.166666666666668
```

(ii)

```
>>> 22/7*5*5
    78.57142857142857
```

4.
```
print("I am learning Python") # Output: I am learning Python
print("I am a class 11 student") # Output: I am a class 11 student
print("I will now play with numbers") # Output: I will now play with numbers
print(10 + 20 + 30 + 40 + 50) # Output: 150
print(2 * 22 / 7 * 4) # Output: 25.142857142857142
```

Assertion and Reasoning Based Questions

1. b.
2. a.

Case-based Questions

1. Displaying Name, Class, and Section in Interactive Mode and Script Mode

Interactive Mode:

i. Open IDLE (Python GUI).

ii. In the Python Shell that appears, type the following commands one by one and press Enter after each:

```
print("Name: Ritwika")
print("Class: 11")
print("Section: A")
```

Output:

```
Name: Ritwika
Class: 11
Section: A
```

Script Mode:

i. Open IDLE (Python GUI).

ii. Click on **File** → **New File**.

iii. In the new file window, type the following code:

```
print("Name: Ritwika")
print("Class: 11")
print("Section: A")
```

iv. Save the file with a .py extension, e.g., display_info.py.

v. To run the script, click on **Run** → **Run Module** or press **F5**.

Output in Shell:

```
Name: Ritwika
Class: 11
Section: A
```

Difference Between Interactive Mode and Script Mode:

- Interactive Mode: Allows executing commands one at a time, providing immediate feedback. Ideal for testing and experimenting with small snippets of code.
- Script Mode: Involves writing code in a file and executing the entire script at once. Useful for writing and running longer programs.

2. Features of Python for a Presentation

Title Slide:

- Title: Features of Python
- Subtitle: An Overview by Krishnan

Slide 1: Introduction

- Python is a high-level, interpreted, and general-purpose programming language.
- Created by Guido van Rossum and first released in 1991.

Slide 2: Simple and Easy to Learn

- Python has a clean and straightforward syntax that is easy to understand.
- Great for beginners and experienced programmers alike.

Slide 3: Interpreted Language

- Python code is executed line-by-line, making debugging easier.
- No need for compilation, reducing development time.

Slide 4: Dynamically Typed

- Variables do not need explicit declarations.
- Types are determined at runtime, providing flexibility.

Slide 5: Platform-Independent

- Python can run on various operating systems such as Windows, macOS, and Linux without modification.
- Write once, run anywhere.

3. Basics of Python Programming

Assessment

A.

1. b	2. a	3. c	4. b	5. a
6. c	7. b	8. d	9. d	10. b
11. d	12. a	13. c	14. c	15. b
16. c	17. a	18. b	19. a	20. c
21. c				

B.

1. True	2. False	3. False	4. True	5. False
6. False	7. True	8. True	9. False	10. True
11. True	12. False			

C.

1. keyword	2. underscore(_)	3. Delimiters	4. variable	5. assignment
6. newline	7. print()	8. int()		

D. 1. Yes, space is included in the character set of Python.

2. Five different tokens used in Python are:
   - Keywords
   - Identifiers
   - Literals
   - Delimiters
   - Operators

3. `Asia&Europe` : it is NOT valid as special symbols, such as !,@,#,$,%^,&,*,(,) cannot be included in an identifier.

   `Item-name` : it is NOT valid as special symbols, such as !,@,#,$,%^,&,*,(,) cannot be included in an identifier.

   `while` : it is NOT valid as an identifier must be different from any of the keywords in the Python language.

IMPRINT 1

`For` : it is a valid identifier.

`False` : it is NOT valid as an identifier must be different from any of the keywords in the Python language.

`your_choice` : it is a valid identifier.

`input()` : it is NOT valid as () cannot be used in identifiers and input is a keyword.

`output` : it is a valid identifier.

4. `Yes`, the operation performed by a particular operator depend on the type of data on which it is being performed. The type of the operator should be compatible for the operator.

   e.g : if we apply addition operator on two integer values it adds them but on being applied on two strings it concatenates them. If we try to add integer and string it will give error.

5. Yes, a variable is an example of an identifier as it is also a name for a Python object that denotes a data value.

6. Rules for forming a variable are :

   (i) A variable should begin with an uppercase or lowercase alphabet or an underscore (_).

   (ii) The first letter of the variable(including an underscore) may be followed by an arbitrary combination of alphabets (a-z, A-Z), digits (0-9), and underscore (_).

   (iii) A variable cannot start with a digit.

   (iv) Variables are case-sensitive.

   (v) Variables must be different from any of the keywords in the Python language.

   (vi) Blank spaces within a variable are disallowed.

   (vii) The special symbols, such as ! (,) , cannot be included in a variable.

7. `weight = 15.6`

8. (i) operator : =

   operand : choice, 'yes'

   (ii) Name of the variable is **choice**.

   (iii) The data type associated with `'yes'` is string.

9. Python uses the following syntax for assigning the same value to multiple variables :

   <identifier_1> = <identifier_2> = ... = <identifier_N> = <value>

   e.g : var1 = var2 = var3 = 123

10. Output :

    20

    20 -1

11. No, it will not produce an output as we have not printed the result of the expression.

12. Yes, it will produce an output as we have used the print() function to display the output.

13. No, it will not produce an output and will give error instead as we have used the '/' operator on a string.

14. The text beginning with a hash character is called a comment. A comment may also be a multi-line string, enclosed within triple single quotes or triple double quotes. Comments are freely used in a program to facilitate the reading of the program.

    It is different from other statements in Python as the comments are ignored by the Python interpreter as if they are not present.

15. The purpose of **sep** and **end** clauses while invoking the `print()` function is:

    sep : to specify the separator between the values passed.

    end : to change the the default value of the end from end of line character.

16. (i) #this is a single line comment

    (ii) ''' this is a multiline comment

    it can also be made by using triple double quotes instead '''

17. A print ( ) without any argument will print a blank line. So, the Python statement is :

```
print()
```

18. A comment in a line in a Python program begins with the symbol hash (#) . As a comment beginning with the hash symbol extends up to the end of the line in which the symbol # appears, it is also known as a single-line comment. However, a comment may also be a multi-line string, enclosed within triple single quotes or triple double quotes.

19. Comments are included in a program to make the program more readable. During the program execution, the comments are ignored by the Python interpreter as if they were not present. The comments constitute an important part of the code as these comments convey the purpose and approach of the program, thus making the source code easier to understand for everyone.

E. 1.

```
'''
Objective: To accept the name and age of a user
Output: To display the name and age of a user
'''
name = input('Enter name : ')
age = input('Enter age : ')
print('Name: ',name)
print('Age: ',age)
```

2.

```
'''
Objective: To accept the city and state from a user
Output: To display the city and state
'''
city= input('Enter city : ')
state= input('Enter state : ')
print('City: ',city,'\n','State: ',state)
```

3.
```
'''
Objective: To accept the side of a cube
Output: To display the volume of the cube
'''
side = int(input('Enter side : '))
volume = side * side * side
print('Volume : ',volume)
```
4.
```
'''
Objective: To accept the school name and address
Output: To display the school name and address
'''
schoolName = input('Enter the name of your school : ')
schoolAddress = input('Enter the address of your school : ')
print('School Name : ',schoolName)
print('Address : ',schoolAddress)
```

Assertion and Reasoning Based Questions

1. c.
2. d.
3. b.
4. c.
5. a.

Case-based Questions

1.
```
'''
Objective: To perform arithmetic operations
'''
num1 = int(input('Enter first number : '))
num2 = int(input('Enter second number : '))
print('Sum = ',num1+num2)
print('Difference = ',num1-num2)
print('Product = ',num1*num2)
print('Quotient = ',num1//num2)
print('Remainder = ',num1%num2)
```

2. Anish edited the print function and changed the sep to end as follows:

```
print(num1,   num2,   num3,   num4,   num5,   end = '$')
```

3.
```
'''
Objective: To accept the radius of circle to find area and perimeter
Output: To display the area and perimeter of the circle
'''
radius = int(input('Enter radius : '))
area = 3.14*radius*radius
perimeter = 2*3.14*radius
print('Area : ',area)
print('Perimeter : ',perimeter)
```

4.
```
'''
Objective: To accept and display the details of user
'''
name = input('Enter name : ')
class_ = input('Enter class : ')
age = input('Enter age : ')
favSport = input('Enter favorite sport : ')
print('Name : ',name, end='\n\n')
print('Class : ',class_, end='\n\n')
print('Age : ',age, end='\n\n')
print('Favorite Sport : ',favSport, end='\n\n')
```

4. Data Types and Operators

Assessment

A.	1. a	2. b	3. a	4. c	5. d
	6. b	7. b	8. c	9. c	10. b
B.	1. True	2. False	3. True	4. True	5. True
	6. True	7. True	8. True	9. False	10. False
C.	1. type()	2. False, True	3. curly {}	4. identity	5. ord()

D. 1.
```
"kilogram": string
78.3: float
[90,45,23,76]: list
 ('a', 'b'): tuple
```


IMPRINT 1

2. `message[2] : 'u'`

   `message[5] : 'r'`

3. `t = [1, 3, 4, 5, 6, 3, 10] is a list`

   `t = {1, 3, 4, 5, 6, 3, 10} is a set`

4. This results in error because we are trying to modify a tuple which is not allowed as tuple is immutable data type.

5. Mutable data types : **list, dictionary**

   Immutable data types : **string, tuple**

6. **String** data type is used to store textual data.

7. **Tuple:** (i) A tuple is a sequence of objects (not necessarily of the same type) enclosed in parenthesis: ().

   (ii) A tuple is immutable.

   Dictionary: (i) A dictionary (diet) is an unordered set of key-value pairs enclosed in curly brackets:{ }. It maps a set of keys to a set of values.

   (ii) A dictionary is mutable.

8. A tuple should be used to store the days of the week. It is so because tuples are immutable which means they cannot be modified.

9. Binary operators can act on a pair of values as need two operands.

10. (i) 5

    (ii) -64

    (iii) True

11. (i) *** :** multiplies two numbers or concatenates two strings

    **** :** is for exponentiation

    (ii) **= :** assigns value

    == : compares and checks equality

    (iii) **relational :** A relational operator is used to compare the values of operands on either side. The result of applying a relational operator is either True or False.

    logical : A logical value is True or False. The logical operators yield True or False, depending upon the value of logical operands on either side.

    (iv) **/ :** does division

    // : does floor division

    (v) **identity :** Identity operators are typically used for checking the data type of an object. The identity operator returns True if the operands on either side of the operator are the same object or same type, and False otherwise

    membership : These operators are used to check whether a particular value is a member of a given sequence and return either True or False.

12. An expression in Python is a valid combination of constants, variables, and operators. A single value of any type or the name of an object (i.e. a variable) are examples of the simplest expressions. On evaluation, an expression yields a value. The type of an expression is based on the types of operators and operands that are used in it.
13. It won't be executed as the input taken is string by default and we have not changed it to integer type to perform subtraction on it as we cannot subtract integer from string.
14.
```
'''
Objective: To display a string n number of times
'''
n = int(input('Enter value of n : '))
string = input('Enter string : ')
print(string*n)
```
15. **Logical errors:** A logical error does not give the desired output, even though there is no syntax error in the program. Such errors are sometimes difficult to identify as all the statements in the program are executed successfully. When a logical error is encountered, often it helps to work backwards by examining the output produced by the program and looking for the cause of the error.

    e.g: percentage = maxMarks/marksObtained * 100

    Runtime errors: As the name suggests, a runtime error occurs during the execution of the program. The statement is syntactically correct but cannot be executed by the interpreter.

    e.g: result = numerator/denominator #(denominator =0)
16. 30.0
17. 22
18. No, **'None'** is not the same as **None**. This is so because **None** is the NoneType data type while **'None'** is a string as it is enclosed between ''.
19. False
20. True
21. (i) False

    (ii) False

    (iii) False

    (iv) True

    (v) 1.0
22. Output: **2.75 2 7 6.5 5**
23. Logical errors are hard to locate as all the statements in the program are executed successfully. When a logical error is encountered, often it helps to work backwards by examining the output produced by the program and looking for the cause of the error.

24.
```
'''
Objective: To display area and perimeter of parallelogram
'''
length = int(input('Enter length of parallelogram : '))
breadth = int(input('Enter breadth of parallelogram : '))
height = int(input('Enter height of parallelogram : '))
perimeter = 2*(length+breadth)
area=breadth*height
print('Perimeter of parallelogram : ',perimeter)
print('Area of parallelogram : ',area)
```

Assertion and Reasoning Based Questions

1. c.
2. b.
3. a.
4. a.
5. b.

Case-based Questions

1. On execution of the snippet Purvi will get **False** as the output.
2.
```
'''
Objective: To accept a 2 digit number and reverse it
'''
num = int(input('Enter a 2 digit number : '))
quotient = num//10
remainder = num%10
revNum = remainder*10 + quotient
print('The number reversed is : ',revNum)
```
3.
```
'''
Objective: To accept time in seconds and display in minutes and seconds
'''
time = int(input('Enter a time in seconds : '))
minutes = time//60
seconds = time%60
```

```
print('Time : ',minutes,' minutes ',seconds,' seconds')
```

4.
```
'''
Objective: To accept height in feet and inches and display in centimetres
'''
feet = int(input('Enter height in feet : '))
inch = int(input('Enter height in inch : '))
centimetre = ((feet * 12) + inch)*2.54
print('Height in centimetre : ',centimetre)
```

5. Conditional Statements

Assessment

A. 1. b 2. d 3. a 4. b 5. c 6. d

B. 1. True 2. False 3. True 4. True 5. True

C. 1. sequential 2. first 3. conditional 4. colon

D. 1. In Python, the pass statement is ignored by the interpreter. Thus, it is a dummy statement. Often, it is used to leave a slot for the code to be filled in later. Sometimes, it is also used to simplify program logic when no action is required for a particular condition.

2. (i) **Simple statement:** A simple statement appears by itself in a line. However, Python allows several statements in a single line, separated by semicolons. The assignment statement, input, and output statements are examples of simple statements. An expression by itself is also a simple statement and is called an expression statement.

Compound statement: A compound statement often spans several lines. It comprises one or more header clauses and one or more sequences of statements, called suites. A header clause begins with a keyword and ends with a colon. A suite may comprise one or more statements. The suite, following a header clause, appears at the next level of indentation.

(ii) **Simple statement:** A simple statement appears by itself in a line. However, Python allows several statements in a single line, separated by semicolons. The assignment statement, input, and output statements are examples of simple statements. An expression by itself is also a simple statement and is called an expression statement.

Pass Statement: In Python, the pass statement is ignored by the interpreter. Thus, it is a dummy statement. Often, it is used to leave a slot for the code to be filled in later. Sometimes, it is also used to simplify program logic when no action is required for a particular condition.

(iii) **`if statement:`** An if statement is the simplest compound statement that tests a condition. If the conditional expression yields True on its evaluation, the sequence of statements following the if header clause is executed. All statements in a suite are aligned at the same indentation level and at a level next to the if header clause. However, if the conditional expression yields False, the suite following the header is ignored by the interpreter.

`if-else statement:` The if statement executes a sequence of statements when the conditional expression yields True. The statement(s) in the if block is/are ignored by the Python interpreter when the conditional expression yields False. However, sometimes, certain statements need to be executed when the conditional expression is False. In such a situation, an else clause is used.

3. Both the if and if-elif-else statements check for one conditional expression and, based on whether the conditional expression yields (on evaluation) True or False,execute the if suite (also called the if block) or the else suite (also called the else block) respectively. But sometimes, there arises a need to check multiple conditional expressions. Several conditional expressions involving multiple header clauses can be tested one by one using an if-elif-else statement. One can specify as many elif clauses as required in an if-elif-else statement.

4. (i) name == 'Gaurav' or marks>50 and marks<90

   (ii) P>78 or Q<=60

   (iii) city == 'Delhi' and qualification == 'Graduate'

5.
```
if num1 == num2 :           #assignment operator and colon
    print("Equal")          #indentation
else:
    print("Unequal")        #semicolon not required
```

6. Example of nested if :

```
if percentage >= cutOffA:
  grade = 'A'
else:
  if percentage >= cutOffB:
      grade = 'B'
  else:
      if percentage >= cutOffC:
          grade = 'C'
      else:
          if percentage >= cutoffD:
              grade = 'D'
          else:
              grade = 'F'
```

7. (i) Multiple of 5
   (ii) Not Multiple of 5
8. (i) 66
   (ii) 16
   (iii) 8
   (iv) 17
9. (i) Suceess
   (ii) 4
   (iii) Not Same
   (iv) There will be no output in this case
   (v) TRUE
   (vi) Hey

E. 1.
```
'''
Objective: To display the smaller number
'''
num1 = int(input('Enter first number : '))
num2 = int(input('Enter second number : '))
if num1<num2:
    print('Smaller number = ',num1)
else:
    print('Smaller number = ',num2)
```

2.
```
'''
Objective: To display the smallest number
'''
num1 = int(input('Enter first number : '))
num2 = int(input('Enter second number : '))
num3 = int(input('Enter third number : '))
if num1<num2 and num1<num3:
    print('Smallest number = ',num1)
elif num2<num1 and num2<num3:
    print('Smallest number = ',num2)
else:
    print('Smallest number = ',num3)
```

3.
```
'''
Objective: To display the smallest number
```

```
'''
num1 = int(input('Enter first number : '))
num2 = int(input('Enter second number : '))
num3 = int(input('Enter third number : '))
num4 = int(input('Enter fourth number : '))
if num1<num2 and num1<num3 and num1<num4:
    print('Smallest number = ',num1)
elif num2<num1 and num2<num3 and num2<num4:
    print('Smallest number = ',num2)
elif num3<num1 and num3<num2 and num3<num4:
    print('Smallest number = ',num3)
else:
    print('Smallest number = ',num4)
```

4.
```
'''
Objective: To display the numbers in descending order
Inputs:
num1, num2, num3: numbers to be arranged in descending order
Output: Numbers in ascending order
'''
num1 = int(input('Enter first number: '))
num2 = int(input('Enter second number: '))
num3 = int(input('Enter third number: '))
print('The numbers in DESCENDING order are: ')
if num1 > num2:
    if num2 > num3:
        print(num1, num2, num3)
    else:
        if num1 > num3:
            print(num1, num3, num2)
        else:
            print(num3, num1, num2)
else:
    if num3 > num2:
        print(num3, num2, num1)
    else:
        if num3 > num1:
```

```
            print(num2, num3, num1)
        else:
            print(num2, num1, num3)
```

5.
```
'''
Objective: To check if a number is multiple of another
Inputs:
num1: Number to be tested  - numeric value
num2: Number to be tested on - numeric value
Output: message to print whether num1 is multiple of num2 or not
'''
num1 = int (input('Enter first number : '))
num2 = int (input('Enter second number : '))
if num1 % num2 == 0:
    print(num1, ' is a multiple of', num2)
else:
    print(num1, ' is not a multiple of', num2)
```

6.
```
'''
Objective: To find roots of quadratic  equation
Inputs:
a,b,c: coefficients - float value
Output: value of roots of equation
'''
a = float(input('Enter a (a should not be 0): '))
b = float(input('Enter b : '))
c = float(input('Enter c : '))
print('Equation entered = ',a,'x^2 + ',b,'x + ',c)
root1 = (-b + (b**2 - 4*a*c)**0.5)/(2*a)
root2 = (-b - (b**2 - 4*a*c)**0.5)/(2*a)
print('Roots of the equation are : ',root1,' and ',root2)
```

7.
```
'''
Objective: To find area or perimeter of square
User input: side of square and a character
'''
side = float(input('Enter the side of square(in cm) : '))
character = input('Enter A for area and P for perimeter : ')
area = side**2
```

```
perimeter = 4*side
if character == 'A':
    print('Area = ',area)
elif character == 'P':
    print('Perimeter = ',perimeter)
else:
    print('You entered an invalid character!')
```

8.
```
'''
Objective: To display a student's grade, based on his/her percentage
User input: marks obtained by the student
Output: Student's grade: A, B, C, D, or F respectively
'''
name = input("Enter your Name: ")
marks = int(input("Enter your Marks: "))
maxMarks = int(input("Enter maximum marks in examination: "))
percentage = (marks/maxMarks)*100

assert percentage >=0 and percentage <=100

if percentage >= 90:
    grade = 'A'
elif percentage >= 75 and percentage < 90:
    grade = 'B'
elif percentage >= 60 and percentage < 75:
    grade = 'C'
elif percentage >= 40 and percentage < 60:
    grade = 'D'
elif percentage >= 33 and percentage < 40:
    grade = 'E'
else:
    grade = 'F'
print('Your Grade is', grade)
```

9.
```
'''
Objective: To check if the entered character is uppercase, lowercase or
digit
User input: a character
```

```
'''
character = input('Enter a character : ')
if character >= 'A' and character <= 'Z':
    print(character,' is an uppercase character')
elif character >= 'a' and character <= 'z':
    print(character,' is a lowercase character')
elif character >= '0' and character <= '9':
    print(character,' is a digit')
else :
    print(character,' is a special character')
```

10.
```
'''
Objective: To find BMI
User input: height and weight
Output: BMI calculated
'''
height = float(input('Enter height(in m) : '))
weight = float(input('Enter weight(in kg) : '))
bmi = weight/(height**2)
print('BMI = ',bmi)
'''
```

Assertion and Reasoning Based Questions

1. d.

Case-based Questions

1.
```
'''
Objective: To find BMI
User input: height and weight
Output: BMI calculated
'''
height = float(input('Enter height(in m) : '))
weight = float(input('Enter weight(in kg) : '))
bmi = weight/(height**2)
print('BMI = ',bmi)
```

```
    if (bmi < 18.5):
        print("Underweight")
    elif ( bmi >= 18.5 and bmi <= 24.9):
        print("Normal Weight")
    elif ( bmi >= 25 and bmi < 30):
        print("Overweight")
    elif ( bmi >=30):
        print("Obese")
```

2.
```
    '''
    Objective: To make a menu driven program to perform various operations
    '''
    print('---MATHEMAGIC---')
    choice  = int(input('''
    1. Display the largest number
    2. Display the smallest number
    3. Display only even numbers
    4. Display only odd numbers
    5. Exit
    Enter your choice : '''))
    num1 = int(input('Enter a number : '))
    num2 = int(input('Enter a number : '))
    num3 = int(input('Enter a number : '))

    if choice == 1:
        if num1>num2 and num1>num3:
            print('Largest number = ',num1)
        elif num2>num1 and num2>num3:
            print('Largest number = ',num2)
        else:
            print('Largest number = ',num3)
    elif choice == 2:
        if num1<num2 and num1<num3:
            print('Smallest number = ',num1)
        elif num2<num1 and num2<num3:
            print('Smallest number = ',num2)
        else:
```

```
            print('Smallest number = ',num3)
    elif choice == 3:
        if num1%2 == 0:
            print(num1)
        if num2%2 == 0:
            print(num2)
        if num3%2 == 0:
            print(num3)
    elif choice == 4:
        if num1%2 != 0:
            print(num1)
        if num2%2 != 0:
            print(num2)
        if num3%2 != 0:
            print(num3)
    elif choice == 5:
        exit()
    else:
        print('INVALID CHOICE!')
```

6. Looping in Python

Assessment

A. 1. a 2. a 3. c 4. c 5. a
6. c 7. d 8. c 9. d

B. 1. False 2. True 3. False 4. False 5. True
6. True

C. 1. iteration or looping 2. else 3. break 4. inner
5. iteration

D. 1. (i) **for:** The for loop is used to execute a sequence of statements over and over again and will be executed for each value of the control variable in the sequence.

while: The while statement is used to execute a sequence of statements over and over again as long as the condition specified in the while statement is True.

(ii) **break:** The break statement terminates the same loop in which it is defined and moves the control to the next statement immediately following the loop. Once the break statement is encountered and executed, no further statement in the loop will be executed.

IMPRINT 1

continue: The continue statement is also a jump statement, just like the break statement. When a continue statement is encountered, the remaining statement(s) in the loop is skipped and the control jumps to the beginning of the loop for the next iteration.

2. **else** clause in an **if** statement is executed if the conditional expression yields False while **else** block in a **for** loop is executed after all iterations of the loop.
3. A while loop becomes an infinite loop if the test condition never yields False. For example,

```
while True:
        s = 1
print(s)
```

4. Error in the code is due to:

   (i) count not being defined

   (ii) when input f n is non-zero then the while loop will be infinite loop

   (iii) using + instead of , in print() as we are not concatenating strings

5. for moon in range(50,0,-5):

```
    print(moon//5)
```

   Here, for loop is better than while loop as all we need to do is count down and it will make our program more readable.

6.
```
sun=0
while (sun<5):
    moon=0
    while (moon<3):
        if sun**moon>=5:
            print(sun)
        else:
            print(moon)
        moon+=1
    sun+=1
```

   Here, for loop is better than while loop as all it will make our program more understandable.

7. (i) 5#2
   4#3

   (ii) W o r l d P e a c e

   (iii) 5
   4
   3

8. A pass statement is ignored by the Python interpreter. It is often used as a stub when we want to leave some functionality to be defined in the future. For example, pass statement can be used when no action is required on some conditions.

For example,

```
for letter in 'world' :
    if letter == '1':
        pass
    else:
        print(letter, end='')
```

E. 1.
```
'''Objective: To find sum of first n natural numbers
User input: n - numeric value
'''
n = int(input('Enter a number : '))
sum = 0
while n>0:
    sum += n
    n -= 1
print('Sum = ',sum)
```

2.
```
'''
Objective: To find odd numbers entered
'''
sum = 0
for i in range(10):
    n = int(input('Enter a number : '))
    if n%2 != 0:
        sum += n
print('Sum = ',sum)
```

3.
```
'''
Objective: To display the second largest number
'''
n = int(input('Enter the value of n : '))
max1 = 0
max2 = 0
for i in range(n):
    num = int(input('Enter a number : '))
    if num > max1:
        max2 = max1
        max1 =  num
print('second largest number = ',max2)
```

4.
```
'''
Objective: To display the Fibonacci series
'''
n = int(input('Enter the value of n : '))
a = 0
b = 1
for i in range(n):
    c = a
    a = a + b
    b = c
     print(c,end=' ')
```

5.
```
'''
Objective: To display the prime numbers between two given numbers
'''
num1 = int(input('Enter first number : '))
num2 = int(input('Enter second number : '))
for i in range(num1,num2+1):
    for j in range(2,i//2 + 1):
        if i%j == 0:
            break
    else:
        print(i,end=' ')
```

6.
```
' ' '
    Objective : To check whether a number is a palindrome
    Input : num - numeric value
    Output : Appropriate Message
'''
num = int(input("Enter a number: "))
n = num
reverseNum 0
remainder = 0
while num > 0:
    remainder = num %10
    reverseNum = reverseNum * 10 + remainder
    num //= 10
```

```
if n == reverseNum:
    print(n, 'is palindrome')
else:
    print(n, 'is not palindrome')
```

7.
```
'''
Objective: To display the highest, lowest and average marks
'''
name = input('Enter name : ')
marks = int(input('Enter marks : '))
nameMax = name
nameMin = name
maxMarks = marks
minMarks = marks
sumMarks = marks
sum = 0
for i in range(9):
  name = input('Enter name : ')
  marks = int(input('Enter marks : '))
  if marks > maxMarks:
      maxMarks = marks
      nameMax = name
  if marks < minMarks:
      minMarks = marks
      nameMin = name
  sum += marks
average = sum//10
print('(i) Average Marsks = ',average)
print('(ii) Student with highest marks : ',nameMax,' with ',maxMarks,'
marks')
print('(iii) Student with lowest marks : ',nameMin,' with ',minMarks,'
marks')
```

8.
```
'''
Objective: Display sum of series 1-p+p^2-p^3.....
'''
n = int(input("Enter n : "))
p = int(input("Enter p : "))
sum = 1
```

```
sign = -1
for i in range(1,n+1):
    sum += sign*(p**i)
    sign *= -1
print('Sum = ',sum)
```

9.
```
'''
Objective:
To check whether a number is perfect number
Input: number
Output: result whether it is perfect number or not
'''
number = int(input("Enter a Number: "))
assert number>0
sum = 0
for i in range(1, number):
    if(number % i == 0):
        sum = sum + i
if (sum == number):
    print("Is a Perfect Number")
else:
    print("Is NOT a Perfect Number")
```

10. (i)
```
for i in range(num):
        for j in range(0,i+1):
            print(chr(65+j),end=' ')
        print('')
```

(ii)
```
num = int(input('Enter a number : '))
space=0
    symbol=num
    for i in range(num):
        print(space*' ',end='')
        print(symbol*'* ')
        space+=1
        symbol-=1
```

(iii)

```
num = int(input('Enter a number : '))
for i in range(num//2):
    for j in range(num//2-i+1):
        print('*',end=' ')
    print('')
for i in range(num//2+1):
    for j in range(i+1):
        print('*',end=' ')
    print('')
```

(iv)

```
num = int(input('Enter a number : '))
for i in range(num//2+1):
    for j in range(i+1):
        print('*',end=' ')
    print('')
for i in range(num//2):
    for j in range(num//2-i):
        print('*',end=' ')
    print('')
```

(v)

```
num = int(input('Enter a number : '))
for i in range(num):
    for j in range(num):
        if (i==j):
            print(i+1,end=' ')
        else:
            print(0,end=' ')
    print('')
```

(vi)

```
num = int(input('Enter a number : '))
for i in range(num):
    for j in range(i+1):
       print(i+1,end='  ')
    print('')
```

(vii)

```
num = int(input('Enter a number : '))
for i in range(num):
    for j in range(num,i,-1):
        print(i+1,end=' ')
    print('')
```

(viii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(0,i+1):
            print(j+1,end='  ')
        print('')
```

(ix)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1,0,-1):
            print(j,end='  ')
        print('')
```

(x)

```
num = int(input('Enter a number : '))
c=1
    for i in range(num):
        k=c
        for j in range(i+1):
            print(c,end=' ')
            c-=1
        c=k+i+2
        print('')
```

(xi)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num,i,-1):
            print(j,end=' ')
        for k in range(i,num):
            print(k+1,end=' ')
        print('')
```

(xii)

```
num = int(input('Enter a number : '))
for i in range(num):
    for j in range(i+1):
        print((num-j)*2,end=' ')
    print('')
```

(xiii)

```
num = int(input('Enter a number : '))
for i in range(num+1):
        for j in range(i+1):
            print(i*j,end=' ')
        print('')
```

(xiv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1):
            print(2*i+1,end=' ')
        print('')
```

(xv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            print('*',end=' ')
        print('')
```

(xvi)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if i==j:
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xvii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==0) or (j==0) or (i==num-1) or (j==num-1):
                print('*',end=' ')
            elif i==j:
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xviii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i+j==num-1) and (i!=0) and (j!=0):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xix)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==0) or (j==0) or (i==num-1) or (j==num-1):
                print('*',end=' ')
            elif (i+j==num-1):
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xx)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==0) or (j==0) or (i==num-1) or (j==num-1):
```

```
                print('*',end=' ')
            elif (i+j==num-1)or (i==j):
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxi)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1):
            print('*',end=' ')
        print('')
```

(xxii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1):
            if (j==i):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxiii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1):
            if (j==0) or (i==num-1) or (i==j):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxiv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(i+1):
            if (j==0) or (i==num-1) or (i==j):
                print('^',end=' ')
```

```
            else:
                print(' ',end=' ')
        print('')
```

(xxv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i+j>=num-1):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxvi)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i+j>num-1):
                print('*',end=' ')
            elif (i+j==num-1):
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxvii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i+j==num-1) or (i==num-1) or (j==num-1):
                print('^',end=' ')
            elif (i+j>num-1):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxviii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i+j==num-1) or (i==num-1) or (j==num-1):
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxix)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num-i):
            print('*',end=' ')
        print('')
```

(xxx)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num-i):
            if j==num-i-1:
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxxi)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num-i):
            if (j==num-i-1)or (i==0)or(j==0):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxxii)

```
num = int(input('Enter a number : '))
for i in range(num):
```

```
        for j in range(num-i):
            if (j==num-i-1)or (i==0)or(j==0):
                print('^',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxxiii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if(i==0) or (j==0)or(i==num-1)or(j==num-1):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxxiv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if(i==0) or (j==0)or(i==num-1)or(j==num-1)or(j==num//2):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxxv)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if(i==0) or (j==0)or(i==num-1)or(j==num-1)or(i==num//2):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxxvi)

```
num = int(input('Enter a number : '))
for i in range(num):
```

```
        for j in range(num):
            if(i==0)  or  (j==0)or(i==num-1)or(j==num-1)or(i==num//2)
            or(j==num//2):
                print('*',end=' ')
            else:
                print(' ',end=' ')
        print('')
```

(xxxvii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==num//2) or (j==num//2):
                print('#',end=' ')
           elif((i<num//2) and (j<num//2))or((i>num//2)and(j>num//2)):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxxviii)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==num//2) or (j==num//2):
                print('#',end=' ')
            elif(i>num//2):
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxxix)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==num//2) or (j==num//2):
                print('#',end=' ')
            elif(j>num//2):
```

```
                print('^',end=' ')
            else:
                print('*',end=' ')
        print('')
```

(xxxx)

```
num = int(input('Enter a number : '))
for i in range(num):
        for j in range(num):
            if (i==num//2) or (j==num//2):
                print('#',end=' ')
            elif(i<num//2)and (j<num//2):
                print('*',end=' ')
            elif(i<num//2) and (j>num//2):
                print('^',end=' ')
            elif(i>num//2) and (j<num//2):
                print('!',end=' ')
            else:
                print('$',end=' ')
        print('')
```

(xxxxi)

```
num = int(input('Enter a number : '))
space=num-1
    symbol=2
    for i in range(num//2):
        print(space*' ',end='')
        print(symbol*'* ')
        space-=2
        symbol+=2
    space+=2
    symbol-=2
    for j in range(num//2):
        print(space*' ',end='')
        print(symbol*'$ ')
        space+=2
        symbol-=2
```

Assertion and Reasoning Based Questions

1. d.

Case-based Questions

1.
```
'''
Objective: To generate all divisors of a given number
'''
num = int(input("Enter a number : "))
print('The divisors are : 1',end=' ')
for i in range(2,num//2):
    if num%i == 0:
        print(i,end=' ')
print(num)
```
2.
```
'''
Objective: To display first n Mersenne numbers
'''
n = int(input("Enter n : "))
for i in range(1,n+1):
    print('2 ^',i,'- 1 =',2**i -1,' ',end=' ')
```

7. Python Lists

Assessment

A. 1. b 2. d 3. c 4. b 5. a 6. c

B. 1. True 2. False 3. True 4. True

C. 1. square brackets 2. split() 3. three 4. sorted() 5. +

D. 1.
```
myList = eval(input('Enter List:'))
largestElement = myList[0]
for element in myList:
    if element > largestElement:
```

```
        largestElement = element
print('largest element',largestElement)
```

2. 8

   Explanation: The newList becomes [1, 0, -1, 1, 0, -1, 1, 0, 5, -1] after insertion. The index of the first occurrence of 5 is 8.

3.
```
myList = [1, 2, 3, 4, 5, 6]
i = 0
while i < len(myList):
    if i + 1 < len(myList):
        temp = myList[i]
        myList[i] = myList[i + 1]
        myList[i + 1] = temp
    i += 2

# Print the resulting list
print(myList)  # Output: [2, 1, 4, 3, 6, 5]
```

4.
```
myList = [1, 2, 3, 4, 5]
i = 0
while i < len(myList) - 1:
    temp = myList[i]
    myList[i] = myList[i + 1]
    myList[i + 1] = temp
    i += 2

# Print the resulting list
print(myList)  # Output: [2, 1, 4, 3, 5]
```

5.
```
myList = [3, 5, 5, 2, 8, 9, 7, 21, 6, 5, 4]
i = 0
while i < len(myList):
    if i % 3 == 0:
        myList[i] = myList[i] ** 3
    i += 1

# Print the resulting list
print(myList)  # Output: [27, 5, 5, 2, 8, 9, 343, 21, 6, 5, 4]
```

6.
```
myList = [3, 5, 5, 2, 8, 9, 7, 21, 6, 5, 4]
key = 5
i = 0
found = -1
while i < len(myList):
    if myList[i] == key:
        found = i
        break
    i += 1

if found != -1:
    print(found)  # Output: 1
else:
    print("Key not found")
```

7.
```
myList = [3, 5, 5, 2, 8, 9, 7, 21, 6, 5, 4]
key = 5
i = len(myList) - 1
found = -1
while i >= 0:
    if myList[i] == key:
        found = i
        break
    i -= 1

if found != -1:
    print(found)  # Output: 9
else:
    print("Key not found")
```

8.
```
myList = [3, 5, 5, 2, 8, 9, 7, 21, 6, 5, 4]
key = 5
i = 0
found = -1
while i < len(myList):
    if myList[i] == key:
        found = -i
        break
```

```
        i += 1

    if found != -1:
        print(found)  # Output: -1
    else:
        print("Key not found")
```

9.
```
myList = [3, 5, 5, 2, 8, 9, 7, 21, 6, 5, 4]
key = 5
i = len(myList) - 1
found = -1
while i >= 0:
    if myList[i] == key:
        found = -i
        break
    i -= 1

if found != -1:
    print(found)  # Output: -9
else:
    print("Key not found")
```

10.
```
rollNoMarksList = [[101, 50], [102, 60], [103, 70]]
rollNo = 102
i = 0
while i < len(rollNoMarksList):
    if rollNoMarksList[i][0] == rollNo:
        rollNoMarksList[i][1] += 5
    i += 1

# Print the updated list
print(rollNoMarksList)  # Output: [[101, 50], [102, 65], [103, 70]]
```

11.
```
nameMarksList = [[101, 35], [102, 45], [103, 55], [104, 65], [105, 75],
 [106, 85], [107, 95]]
i = 0
while i < len(nameMarksList):
    marks = nameMarksList[i][1]
    if marks < 40:
```

```
            nameMarksList[i][1] += 6
        elif 40 <= marks < 50:
            nameMarksList[i][1] += 5
        elif 50 <= marks < 60:
            nameMarksList[i][1] += 4
        elif 60 <= marks < 70:
            nameMarksList[i][1] += 3
        elif 70 <= marks < 80:
            nameMarksList[i][1] += 2
        elif 80 <= marks < 90:
            nameMarksList[i][1] += 1
        # No change for marks >= 90
        i += 1

# Print the updated list
print(nameMarksList)  # Output: [[101, 41], [102, 50], [103, 59], [104,
68], [105, 77], [106, 86], [107, 95]]
```

Assertion and Reasoning Based Questions

1. a. 2. b

Case-based Questions

1.
```
# Initialize an empty list to store sponsor information
sponsorList = []

# Function to add a sponsor
def addSponsor(name, amount):
    sponsorList.append([name, amount])

# Function to calculate the total amount collected
def calculateTotalAmount():
    totalAmount = 0
    for sponsor in sponsorList:
        totalAmount += sponsor[1]
    return totalAmount
```

```
# Adding some sponsors
addSponsor("Company A", 5000)
addSponsor("Company B", 3000)
addSponsor("Company C", 7000)

# Calculate the total amount collected
totalAmountCollected = calculateTotalAmount()

# Display the total amount collected
print("Total amount collected from sponsors: $", totalAmountCollected)
```

8. Python Dictionaries

Assessment

A.

1. c	2. d	3. a	4. a	5. b
6. d	7. b	8. c	9. d	10. c
11. a				

B.

1. True	2. True	3. False	4. True	5. True

C.

1. curly	2. key	3. del	4. dict()	5. copy()

D.

1. A dictionary is an unordered set of key: value pairs. A dictionary (an object of type diet) is defined in Python by enclosing the comma-separated key: value pairs in braces.

   The syntax for defining and naming a dictionary is as follows:

   ```
   {keyl:valuel, key2:value2, ...}
   For example,
   >>> subjects = {'Sanskrit 78, 'English':85, 'Maths':88, 'Hindi':90}
   >>> subjects
   {'Sanskrit': 78, 'English': 85, 'Maths': 88, 'Hindi': 90}
   ```

2. No, slicing is not applicable to dictionaries because they are unordered collections and do not support indexing.
3. Immutable objects such as strings, numbers, and tuples can be used as keys in dictionaries.
4. `Using the dict() function: emptyDict = dict()`

   `Using curly braces: emptyDict = {}`
5. A dictionary is termed an unordered collection because it does not maintain any order for the key-value pairs. The order of the pairs is not guaranteed to be the same as the order in which they were inserted.

6.
```
rollnoMarksDict = {1:60, 2:25, 3:90, 4:32, 5:12, 6:78}

rollnosAbove40 = []
for rollno, marks in rollnoMarksDict.items():
    if marks >= 40:
        rollnosAbove40.append(rollno)

print(rollnosAbove40)
```
7.
```
string = input('Enter a string:')
countDict = {}

for char in string:
    if char in countDict:
        countDict[char] += 1
    else:
        countDict[char] = 1

print(countDict)
```
8.
```
numStudents = int(input("Enter the number of students: "))
studentMarks = {}

for _ in range(numStudents):
    studentName = input("Enter the name of the student: ")
    studentMarksValue = int(input(f"Enter the marks of
    {studentName}: "))
    studentMarks[studentName] = studentMarksValue

maxMarksStudent = max(studentMarks, key=studentMarks.get)
print(f"The student with the highest marks is {maxMarksStudent} with
{studentMarks[maxMarksStudent]} marks.")
```
9.
```
monthDays = {
    "January": 31, "February": 28, "March": 31, "April": 30,
    "May": 31, "June": 30, "July": 31, "August": 31,
    "September": 30, "October": 31, "November": 30, "December": 31
}

print("Months with 31 days:")
```

```
for month, days in monthDays.items():
    if days == 31:
        print(month)

userMonth = input("Enter a month name: ")
if userMonth in monthDays:
    print(f"{userMonth} has {monthDays[userMonth]} days.")
else:
    print("Invalid month name.")

print("Months in alphabetical order:")
for month in sorted(monthDays):
    print(month)
```

10.
```
employees = {
    "1234-5678-9012": {"name": "Ankur", "salary": 50000},
    "2345-6789-0123": {"name": "Babita", "salary": 60000},
    "3456-7890-1234": {"name": "Chahat", "salary": 55000}
}

while True:
    aadhar = input("Enter Aadhar card number (or 'exit' to quit): ")
    if aadhar.lower() == 'exit':
        break

    if aadhar in employees:
        employeeDetails = employees[aadhar]
        print(employeeDetails)
    else:
        print("Employee not found.")
```

11.
```
# Create a dictionary of employees' id mapped to a list storing their
 name, department, and salary.
employees = {
    101: ["Amar", "HR", 50000],
    102: ["Bosco", "IT", 60000],
    103: ["Chameli", "Finance", 55000],
    104: ["Dheeraj", "IT", 48000],
```

```
    105: ["Elvish", "Marketing", 65000]
}

# Display the id and name of employees having a salary greater than
50,000.
print("Employees with salary greater than 50,000:")
for empId in employees:
    if employees[empId][2] > 50000:
        print(f"ID: {empId}, Name: {employees[empId][0]}")

# Accept the employee id from a user and display the corresponding
details of the employee.
userInputId = int(input("Enter the employee ID to fetch details: "))
if userInputId in employees:
    empDetails = employees[userInputId]
    print(f"Details of employee ID {userInputId}: Name: {empDetails[0]},
    Department: {empDetails[1]}, Salary: {empDetails[2]}")
else:
    print("Employee ID not found.")

# Display the ids of employees in sorted order.
print("Employee IDs in sorted order:")
for empId in sorted(employees):
    print(empId)
```

Assertion and Reasoning Based Questions

1. b. 2. a

Case-based Questions

1.
```
# Create a dictionary for digit spellings
digitSpellings = {
    0: "Zero", 1: "One", 2: "Two", 3: "Three", 4: "Four",
    5: "Five", 6: "Six", 7: "Seven", 8: "Eight", 9: "Nine"
}
```

```
# Accept a number from the user
userInput = input("Enter a number: ")

# Display the spellings of each digit
result = []
for digit in userInput:
    result.append(digitSpellings[int(digit)])
print(" ".join(result))
```

2.
```
# Create an empty dictionary to store employee data
employeeData = {}

# Accept the number of employees
numEmployees = int(input("Enter the number of employees: "))

# Accept name and date of birth for each employee
for _ in range(numEmployees):
    name = input("Enter the employee's name: ")
    dob = input("Enter the employee's date of birth (DD-MM-YYYY): ")
    employeeData[name] = dob

# Display the data of employees ordered by their names
print("Employee data sorted by names:")
for name in sorted(employeeData):
    print(f"Name: {name}, Date of Birth: {employeeData[name]}")
```


9. Database Concepts and the Structured Query Language

Assessment

A.

1. a	2. b	3. a	4. c	5. a
6. b	7. b	8. b	9. d	10. d
11. b	12. c	13. a	14. d	15. b
16. c				

IMPRINT 1

B. 1. False 2. False 3. True 4. True 5. True
6. False

C. 1. DISTINCT 2. primary 3. CHECK constraint 4. UNIQUE 5. GROUP BY
6. outer, inner 7. ORDER BY

D. 1. (i) True (ii) True (iii) True (iv) True (v) False
(vi) True (vii) True (viii) False (ix) False (x) True
(xi) True

2. `ItemCode: INT`
   - This will store integer values.

   `ItemName: VARCHAR`
   - You can specify the maximum length for the item name. For example, if the maximum length of the item name is expected to be 255 characters, you can use **`VARCHAR(255)`**.

   `Price: DECIMAL`
   - The **`DECIMAL`** type is used for storing precise numeric values, especially when dealing with financial data. You can specify the precision and scale. For example, **`DECIMAL(10, 2)`** allows up to 10 digits in total, with 2 digits after the decimal point.

3. No, a relation cannot have two identical tuples because it would violate the principle of a set in relational databases, where all tuples must be unique.

4. (i) **Domain:** A domain is the set of permissible values that an attribute can have.
   (ii) **Constraint:** A constraint is a rule that restricts the values that can be stored in a database to ensure data integrity.
   (iii) **Candidate Key:** A candidate key is an attribute, or a set of attributes, that can uniquely identify a tuple in a relation.
   (iv) **Alternate Key:** An alternate key is any candidate key that is not chosen as the primary key.

5. **Primary Keys:**
   - Employee Table: `Emp_id`
   - Department Table: `Dept_no`

   Insert Operations:

   (i) An insert operation that will be consistent with the current state of the Employee table:

   ```
   INSERT INTO Employee (Emp_id, Name, Dept_no) VALUES ('E005', 'Anjali Gupta', 20);
   ```

   (ii) An insert operation that will be consistent with the current state of the Department table:

   ```
   INSERT INTO Department (Dept_no, Dept_name) VALUES (40, 'Human Resources');
   ```

   (iii) An insert operation that will be inconsistent with the current state of the Employee table, but would be fine if the Employee table were empty:

   ```
   INSERT INTO Employee (Emp_id, Name, Dept_no) VALUES ('E001', 'Rajesh Kumar', 10);
   ```

(Note: This will cause a duplicate primary key error because E001 already exists.)

(iv) An insert operation that will be inconsistent with the current state of the Department table, but would be fine if the Department table were empty:

```
INSERT INTO Department (Dept_no, Dept_name) VALUES (10, 'Finance');
```

(Note: This will cause a duplicate primary key error because 10 already exists.)

(v) An insert operation that will be invalid on the empty table Employee:

```
INSERT INTO Employee (Emp_id, Name, Dept_no) VALUES (NULL, 'Ravi Shastri', 10);
```

(Note: Emp_id cannot be NULL as it is a primary key.)

(vi) An insert operation that will be invalid on the empty table Department:

```
INSERT INTO Department (Dept_no, Dept_name) VALUES (NULL, 'Operations');
```

(Note: Dept_no cannot be NULL as it is a primary key.)

(vii) Will an operation to delete the tuple having Dept_no 20 be consistent with the current state of the Employee and Department tables? Justify your answer:

No, it will be inconsistent because there are employees (Emp_id E003) associated with Dept_no 20 in the Employee table. Deleting the department will violate the foreign key constraint.

6. **Primary Keys:**

- Employee Table: `E_id`
- Project Table: `P_no`
- WorksOn Table: `(P_no, E_id)` (Composite primary key)

Foreign Keys:

- WorksOn Table: `P_no` (references `Project.P_no`), `E_id` (references `Employee.E_id`)

Insert Operations:

(i) A tuple insertion that will be invalid on the empty table Employee:

```
INSERT INTO Employee (E_id, Ename, City, Salary, Department, YearofJoining) VALUES (NULL, 'Alok Sharma', 'Delhi', 50000, 'HR', 2018);
```

(Note: `E_id` cannot be `NULL` as it is a primary key.)

(ii) A tuple insertion that will be invalid on the empty table Project:

```
INSERT INTO Project (P_no, PName, City, DeptName, StartYear) VALUES (NULL, 'New Project', 'Mumbai', 'IT', 2024);
```

(Note: `P_no` cannot be `NULL` as it is a primary key.)

(iii) A tuple insertion that will be invalid on the empty table WorksOn:

```
INSERT INTO WorksOn (P_no, E_id) VALUES (1, 1);
```

(Note: Foreign keys `P_no` and `E_id` do not exist in the `Project` and `Employee` tables respectively.)

(iv) A tuple insertion that will be valid on the empty table Employee:

```
INSERT INTO Employee (E_id, Ename, City, Salary, Department, YearofJoining) VALUES (1, 'Alok Sharma', 'Delhi', 50000, 'HR', 2018);
```

(v) A tuple insertion that will be valid on the empty table Project:

```
INSERT INTO Project (P_no, PName, City, DeptName, StartYear) VALUES (1, 'New Project', 'Mumbai', 'IT', 2024);
```

(vi) A tuple insertion that will be valid on the empty table WorksOn:

```
INSERT INTO WorksOn (P_no, E_id) VALUES (1, 1);
```

(Note: This would require the `Employee` and `Project` tables to have the corresponding entries.)

7. **Primary Keys:**
   - Suppliers Table: `SNo`
   - Parts Table: `PNo`
   - Project Table: `JNo`
   - Shipment Table: `(SNo, PNo, JNo) (Composite primary key)`

   Foreign Keys:
   - Shipment Table: `SNo (references Suppliers.SNo), PNo (references Parts.PNo), JNo (references Project.JNo)`

8. **Insert Operations:**

(i) An insertion on the empty table Suppliers that will generate an error:

```
INSERT INTO Suppliers (SNo, SName, Status, SCity) VALUES (NULL, 'ABC Supplies', 'Active', 'Mumbai');
```

(Note: `SNo` cannot be `NULL` as it is a primary key.)

(ii) An insertion on the empty table Parts that will generate an error:

```
INSERT INTO Parts (PNo, PName, Colour, Weight, City) VALUES (NULL, 'Bolt', 'Silver', 50, 'Delhi');
```

(Note: `PNo` cannot be `NULL` as it is a primary key.)

(iii) An insertion on the empty table Suppliers that will be successful:

```
INSERT INTO Suppliers (SNo, SName, Status, SCity) VALUES (1, 'ABC Supplies', 'Active', 'Mumbai');
```

(iv) An insertion on the empty table Parts that will be successful:

```
INSERT INTO Parts (PNo, PName, Colour, Weight, City) VALUES (1, 'Bolt', 'Silver', 50, 'Delhi');
```

(v) An insertion on the empty table Project that will be successful:

```
INSERT INTO Project (JNo, JName, JCity) VALUES (1, 'Project Alpha', 'Bangalore');
```

(vi) An insertion on the empty table Shipment that will be successful:

```
INSERT INTO Shipment (SNo, PNo, JNo, Quantity) VALUES (1, 1, 1, 100);
```

(Note: This would require the `Suppliers`, `Parts`, and `Project` tables to have the corresponding entries.)

9. **Primary and Foreign Keys:**
   - Employee Table:
     - Primary Key: `employee_id`
     - Foreign Key: `department_id (references Department.department_id)`
   - Department Table:
     - Primary Key: `department_id`
   - Resources Table:
     - Primary Key: `resource_id`
     - Foreign Key: `department_id (references Department.department_id)`

   Table Creation Order:
   1. Department (since other tables reference `department_id`)
   2. Employee
   3. Resources
10. **Entity Integrity:**
    - Ensures that each table has a primary key and that the column or columns chosen to be the primary key are unique and not null.
    - Example: In the Employee table, `employee_id` must be unique and not null.

    Referential Integrity:
    - Ensures that a foreign key value always points to an existing row.
    - Example: In the Employee table, `department_id` must match a `department_id` in the Department table.
11. Insert Operations:

    (i) <18, "Mukesh Agrawal", 11, "C", 88>
    - Executed successfully.

    (ii) <21, "Sanjay", 11, "A", 76>
    - Executed successfully.

    (iii) <NULL, "Phule Bai", 11, "A", 88>
    - Not executed successfully. Violates the primary key constraint (S_ID cannot be NULL).

    (iv) <20, NULL, 11, "A", 88>
    - Executed successfully.

    (v) <20, NULL, 11, NULL, NULL>
    - Executed successfully.

    (vi) <20, NULL, NULL, NULL, NULL>
    - Executed successfully.

12. Insert Operations:

(i) <799575933699, NULL, "Pooja", "1990-10-01", 9776626565>

- Not executed successfully. Violates the NOT NULL constraint on last_name.

(ii) <712349049911, "Singh", "Gagan", "1990-11-11", 9812476543>

- Not executed successfully. Violates the uniqueness constraint on AadharNo.

13. To avoid violating referential integrity constraints when inserting data into the tables, follow these steps:

1. Insert the manager into the Employee table first with a temporary Dept_no (e.g., NULL if allowed or a placeholder department):

```
INSERT INTO Employee (Emp_id, Name, Dept_no, Gender, Address, City, Pincode, Birthdate, Salary, Manager_id) VALUES ('E0011', 'Rashmi Singhania', NULL, 'F', '7, First Floor, MSFC Building, Shivajinagar', 'Pune', 411016, '1990-10-08', 90000, 4);
```

2. Insert the new department with the correct Dept_no:

```
INSERT INTO Department (Dept_no, Dept_name, Location) VALUES (6, 'Food and Beverage', 'Pune');
```

3. Update the manager's Dept_no to the correct value:

```
UPDATE Employee
SET Dept_no = 6
WHERE Emp_id = 'E0011';
```

4. Insert the new employee:

```
INSERT INTO Employee (Emp_id, Name, Dept_no, Gender, Address, City, Pincode, Birthdate, Salary, Manager_id) VALUES ('E0012', 'John Doe', 6, 'M', '123, Example Street', 'Pune', 411016, '1992-05-15', 50000, 'E0011');
```

14. (i) To view the list of databases:

```
SHOW DATABASES;
```

(ii) To start using the database named TEST:

```
USE TEST;
```

(iii) To view the structure of table PRACTICE:

```
DESCRIBE PRACTICE;
```

(iv) To display all the records from table PRACTICE:

```
SELECT * FROM PRACTICE;
```

15. ORDER BY: This clause is used to sort the result set in either ascending or descending order. It does not affect the grouping of rows.

Example:

```
SELECT * FROM Employee ORDER BY Salary DESC;
```

GROUP BY: This clause is used to arrange identical data into groups. It is often used with aggregate functions like COUNT, SUM, AVG, etc.

Example:

```
SELECT Dept_no, COUNT(*) FROM Employee GROUP BY Dept_no;
```

16. Wildcard Characters for Pattern Matching with LIKE
    - %: Matches any sequence of characters (including zero characters).
    - _: Matches any single character.
17. (i) `CREATE TABLE` statement for table `TEACHER`:

```
CREATE TABLE TEACHER (
    ID CHAR(5) PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Dept VARCHAR(50),
    Contact_Num VARCHAR(15),
    Salary INT,
    Email_ID VARCHAR(100)
);
```

(ii) `SELECT` Statements:

a. List the salary of those teachers whose name starts with `'S'`:

```
SELECT Salary FROM TEACHER WHERE First_Name LIKE 'S%';
```

b. List the first name and last name of the teachers who have salary more than 70000:

```
SELECT First_Name, Last_Name FROM TEACHER WHERE Salary > 70000;
```

c. List the count of number of teachers of each department:

```
SELECT Dept, COUNT(*) FROM TEACHER GROUP BY Dept;
```

d. List the first name and contact number of teachers whose mail ID is not known:

```
SELECT First_Name, Contact_Num FROM TEACHER WHERE Email_ID IS NULL;
```

e. Display the rows from the table TEACHER in descending order of salary:

```
SELECT * FROM TEACHER ORDER BY Salary DESC;
```

f. Add an attribute Subject to the table TEACHER:

```
ALTER TABLE TEACHER ADD Subject VARCHAR(50);
```

g. Drop attribute Email_ID from the table TEACHER:

```
ALTER TABLE TEACHER DROP COLUMN Email_ID;
```

(iii) Output of SQL Queries:

a.

```
SELECT AVG(Salary) FROM TEACHER WHERE Dept = 'Economics';
```

- Output: 67500

b.

```
SELECT First_Name, Last_Name, Contact_Num FROM TEACHER WHERE Salary
BETWEEN 40000 AND 80000;
```

Output:

```
Naishadh Kumar 9965789799
Tenzin Wangdi 8023456780
Krishan Kumar 9977885566
```

c.

```
SELECT DISTINCT Dept FROM TEACHER;
```

Output:

```
Political Science
English
History
Computer Science
Economics
```

d.

```
SELECT MAX(Salary) FROM TEACHER GROUP BY Dept HAVING Dept = 'Political
Science';
```

- Output: 85000

18. (i) `CREATE TABLE` statement for table `STUDENT`:

```
CREATE TABLE STUDENT (
    Roll_Num INT PRIMARY KEY,
    Student_Name VARCHAR(50),
    Course_Name VARCHAR(50),
    Duration INT,
    Fee INT,
   Batch_Prefer VARCHAR(10)
);
```

(ii) `SELECT` Statements:

a. List the names of students in each course:

```
SELECT Student_Name FROM STUDENT;
```

b. Display the course name along with their duration for the courses whose fee is more than 30000:

```
SELECT Course_Name, Duration FROM STUDENT WHERE Fee > 30000;
```

c. Display the names of students in alphabetical order:

```
SELECT Student_Name FROM STUDENT ORDER BY Student_Name;
```

d. Display the count of students who prefer the Evening batch:

```
SELECT COUNT(*) FROM STUDENT WHERE Batch_Prefer = 'Evening';
```

e. Display the average duration of courses from the table:

```
SELECT AVG(Duration) FROM STUDENT;
```

f. Increase the fee by 10% for Mobile App course:

```
UPDATE STUDENT SET Fee = Fee * 1.10 WHERE Course_Name = 'Mobile App';
```

g. Delete rows from table student where duration of the course is not known:

```
DELETE FROM STUDENT WHERE Duration IS NULL;
```

(iii) Output of SQL Queries:

a. `SELECT * FROM STUDENT WHERE Student_Name LIKE '%h' AND Fee < 25000;`

Output:

Roll_Num	Student_Name	Course_Name	Duration(months)	Fee	Batch_Prefer
1	Bhaskar Dhyani	Web Development	3	20000	Morning

b. `SELECT MIN(Fee) FROM STUDENT GROUP BY Batch_Prefer;`

Output:

Batch_Prefer	MIN(Fre)
Morning	18000
Evening	25000

c. SELECT `Roll_Num, Course_Name,` Duration `FROM STUDENT ORDER BY` Duration `DESC;`

Output:

Roll_Num	Course_Name	Duration
5	Python Programming	6
2	Machine Learning	4
3	Office Tools	4
1	Web Development	3
4	Mobile App	3
3	Office Tools	NULL

19. (i)

a.
```
SELECT Name
FROM SALESPERSON
WHERE Product = 'Books' AND City = 'Noida';
```

Output:

Name
TENZIN JACK

b. Display the names of cities without any repetition.

```
SELECT DISTINCT City
FROM SALESPERSON;
```

Output:

City
New Delhi
Gurugram
Noida

c. Display the count of salespersons in each city.

```
SELECT City, COUNT(*) AS Salesperson_Count
FROM SALESPERSON
GROUP BY City;
```

Output:

City	Salesperson_Count
New Delhi	1
Gurugram	2
Noida	2

d. Display the name and salary of salespersons in descending order of Salary.

```
SELECT Name, Salary
FROM SALESPERSON
ORDER BY Salary DESC;
```

Output:

Name	Salary
YOGRAJ SINGH	70000
SANDEEP JHA	60000
TARANA SEN	55000
TENZIM JACK	45000
ANOKHI RAJ	45000

e. Delete the salesperson whose salary is more than 60000.

```
DELETE FROM SALESPERSON
WHERE Salary > 60000;
```

Name	Salary
SANDEEP JHA	60000
TARANA SEN	55000
TENZIM JACK	45000
ANOKHI RAJ	45000

F. `ALTER TABLE SALESPERSON`

`ADD Contact_No VARCHAR(15);`

(ii)

(a) `SELECT Product, SUM(Salary) FROM SALESPERSON GROUP BY Product;`

Output:

Product	SUM(Salary)
Stationary	60000
Footwear	90000
Books	45000
Toys	55000

(b) `SELECT Name, Salary FROM SALESPERSON WHERE Product IN ('Toys','Footwear');`

Output:

Name	Salary
YOGRAJ SINGH	70000
TARANA SEN	55000
ANOKHI RAJ	45000

(C) `SELECT Code, Name FROM SALESPERSON WHERE Salary > 50000 AND Name LIKE '%E%';`

Output:

Code	name
1001	SANDEEP JHA

(d) `SELECT AVG(Salary) FROM SALESPERSON WHERE CITY='Gurugram';`

Output:

AVG(Salary)
62500

20. (i) SQL Queries

a. To insert a new tuple in the MOBILE table whose M_Price is not yet known.

```
INSERT INTO MOBILE (M_Id, M_Company, M_Name, Launch_Date)
VALUES ('MB007', 'OnePlus', 'OnePlus5', NULL, '2018-01-01');
```

b. Display the mobile name and name of the company of the mobile phones whose price is greater than 5000.

```
SELECT M_Name, M_Company
```

```
FROM MOBILE
WHERE M_Price > 5000;
```

Output:

M_Name	M_Company
XperiaM	Sony
SefieEx	Oppo

c. List the name of the mobile phones along with their price that were launched in the year 2017.

```
SELECT M_Name, M_Price
FROM MOBILE
WHERE YEAR(Launch_Date) = 2017;
```

Output:

M_Name	M_Price
XperiaM	7500

d. Display M_Company, M_Name, and M_Price in descending order of their launch date.

```
SELECT M_Company, M_Name, M_Price
FROM MOBILE
ORDER BY Launch_Date DESC;
```

Output:

M_Company	M_Name	M_Price
Sony	XperiaM	7500
Micromax	Unite3	4500
Samsung	Galaxy	4500
Nokia	N1100	2250
Oppo	SefieEx	8500

e. List the details of a mobile whose name starts with "S" or ends with "a".

```
SELECT *
FROM MOBILE
WHERE M_Name LIKE 'S%' OR M_Name LIKE '%a';
```

Output:

M_Id	M_Company	M_Name	M_Price	Launch_Date
MB001	Samsung	Galaxy	4500	2013-02-12
MB006	Oppo	SefieEx	8500	2010-08-21

f. Display the names of the mobile companies having prices between 3000 and 5000.

```
SELECT M_Company
FROM MOBILE
WHERE M_Price BETWEEN 3000 AND 5000;
```


IMPRINT 1

Output:

```
M_Company
Samsung
Micromax
```

(ii) Outputs for the given SQL queries

a. `SELECT MAX(Launch_Date), MIN(Launch_Date) FROM MOBILE;`

Output:

```
MAX(Launch_Date)          MIN(Launch_Date)
2017-11-20                2010-08-21
```

b. `SELECT AVG(M_Price) FROM MOBILE;`

Output:

```
AVG(M_Price)
5450.00
```

c.
```
SELECT M_Name, Launch_Date FROM MOBILE WHERE M_Company='Nokia' OR
M_Price IS NULL;
```

Output:

```
M_Name                    Launch_Date
N1100                     2011-04-15
OnePlus5                  2018-01-01
```

The OnePlus5 entry is included since it has a NULL price, matching the condition.

21. (i) SQL Queries

a. To display details of all trains which start from Pune Junction.

```
SELECT * FROM TRAINS
WHERE Start = 'Pune Junction';
```

Output:

```
+--------+-------------------+----------------+--------------------+---------------+
| TNO    | TName             | Start          | End                | Journey_Time  |
+--------+-------------------+----------------+--------------------+---------------+
| 1651   | Pune Hbj Special  | Pune Junction  | Habibganj          | 6             |
| 11096  | Ahimsa Express    | Pune Junction  | Ahmedabad Junction | 12            |
+--------+-------------------+----------------+--------------------+---------------+
```

b. To display details of trains that have a duration of more than 15 hours.

```
SELECT * FROM TRAINS
WHERE Journey_Time > 15;
```

Output:

TNO	TName	Start	End	Journey_Time
12002	Bhopal Shatabdi	New Delhi	Habibganj	28
12314	Sealdah Rajdhani	New Delhi	Sealdah	37
13005	Amritsar Mail	Howrah Junction	Amritsar Junction	40
14673	Shaheed Express	Jaynagar	Amritsar Junction	36

c. To display the names of trains that end with the word "Shatabdi".

```
SELECT TName FROM TRAINS
WHERE TName LIKE '%Shatabdi';
```

Output:

TNAME
Bhopal Shatabdi
Ajmer Shatabdi
Swarna Shatabdi

d. To delete the record of trains that start from Jaynagar.

```
DELETE FROM TRAINS
WHERE Start = 'Jaynagar';
```

e. To change the duration of Swarna Shatabdi to 7 hours.

```
UPDATE TRAINS
SET Journey_Time = 7
WHERE TName = 'Swarna Shatabdi';
```

f. To display the count of trains that end at New Delhi.

```
SELECT COUNT(*) AS Train_Count
FROM TRAINS
WHERE End = 'New Delhi';
```

Output:

Train_Count
5

(ii) Outputs for the given SQL queries

a. `SELECT START, COUNT(*) FROM TRAINS GROUP BY START HAVING COUNT(*)>1;`

Output:

START	COUNT(*)
Pune Junction	2
New Delhi	3
Amritsar Junction	2

b. `SELECT TNO, TName FROM TRAINS WHERE Journey_Time > 6 ORDER BY TNO;`

Output:

```
+--------+----------------------+
| TNO    | TName                |
+--------+----------------------+
| 11096  | Ahinsa Express       |
| 12002  | Bhopal Shatabdi      |
| 12314  | Sealdah Rajdhani     |
| 12417  | Prayag Raj Express   |
| 12451  | Shram Shakti Express |
| 12498  | Shan-e-Punjab        |
| 13005  | Amritsar Mail        |
| 14673  | Shaheed Express      |
+--------+----------------------+
```

c.

```
SELECT Start, End, COUNT(*)
FROM TRAINS
GROUP BY Start, End
HAVING COUNT(*)=2;
```

Output:

```
+------------------+------------+----------+
| Start            |    End     | COUNT(*) |
+------------------+------------+----------+
| Amritsar Junction | New Delhi  |  2       |
+------------------+------------+----------+
```

Assertion and Reasoning Based Questions

1. a. 2. b 3. c 4. a 5. a

Case-based Questions

When inserting data into tables that have foreign key constraints, it's crucial to ensure that the referenced records exist. Here's how you can handle the scenario described:

- Inserting a New Department and Employee:

  To insert a new department (`Dept_No = 6`) and a new employee (with manager `E0011`):

  - **Step 1:** Insert the employee who will be the manager (`E0011`) into the Employee table first.
  - **Step 2:** Once the employee record (`E0011`) exists, then insert the department (`Dept_No = 6`) into the Department table, referencing the manager (`E0011`).

- This sequence ensures that when you insert the department record, the foreign key reference to the manager (`E0011`) exists in the `Employee` table, hence preserving referential integrity

1. Relational Schema:

```
CREATE TABLE myClass (
    Name VARCHAR(50),
    Father_Name VARCHAR(50),
    Contact_Number VARCHAR(15),
    Birth_Date DATE,
    Hobbies TEXT,
    Class VARCHAR(10),
    PRIMARY KEY (Name)
);
```

- Primary Key: Name (assuming names are unique within the class).

2. Table1: Events

```
CREATE TABLE Events (
    Event_Id INT,
    Event_Name VARCHAR(50),
    Event_Date DATE,
    PRIMARY KEY (Event_Id)
);
```

- Primary Key: Event_Id

Table2: Players

```
CREATE TABLE Players (
    Player_Id INT,
    Player_Name VARCHAR(50),
    Age INT,
    Event_Id INT,
    Result VARCHAR(50),
    PRIMARY KEY (Player_Id),
    FOREIGN KEY (Event_Id) REFERENCES Events(Event_Id)
);
```

- Primary Key: Player_Id
- Foreign Key: Event_Id references Events(Event_Id)

3. **Trains Table:**
   - Primary Key: `TrainID`
   - Foreign Key: No explicit foreign key constraint shown, assuming `TrainID` may reference other tables like Stations or Schedules.

   Passengers Table:
   - Primary Key: `RefNo`
   - Foreign Key: `TrainID` references `Trains(TrainID)`

   Degree and Cardinality:
   - Trains Table:
     - Degree: 4 (TrainID, TrainName, Source, Destination)
     - Cardinality: Number of rows (5 in this case)
   - Passengers Table:
     - Degree: 4 (RefNo, TrainID, PassengerName, DOJ)
     - Cardinality: Number of rows (4 in this case)

   Can TrainID in Passengers Table have the value 9999?
   - It depends on the constraints defined. Normally, if 9999 is not an existing TrainID in the Trains table (assuming it's a foreign key), inserting 9999 into Passengers(TrainID) would violate referential integrity unless NULLs are allowed.

4. Creating THEATRE Table:

```
CREATE TABLE THEATRE (
    TCODE VARCHAR(5) PRIMARY KEY,
    TITLE VARCHAR(100),
    AUDI INT,
    LANGUAGE VARCHAR(20),
    SHOWDATE DATE,
    TICKET_PRICE INT,
    TROUPE VARCHAR(40) NOT NULL
);
```

SQL Queries:

a. Display plays staged in Audi 1:

```
SELECT TITLE FROM THEATRE WHERE AUDI = 1;
```

b. Display plays with language as Hindi:

```
SELECT * FROM THEATRE WHERE LANGUAGE = 'Hindi';
```

c. Display plays with ticket price more than 200:

```
SELECT TITLE FROM THEATRE WHERE TICKET_PRICE > 200;
```

d. Count of plays in Hindi and English:

```
SELECT LANGUAGE, COUNT(*) AS Count_Plays FROM THEATRE
WHERE LANGUAGE IN ('Hindi', 'English')
GROUP BY LANGUAGE;
```

e. Display plays in alphabetical order:

```
SELECT TITLE FROM THEATRE ORDER BY TITLE ASC;
```

f. Add TROUPE attribute:

```
ALTER TABLE THEATRE ADD COLUMN TROUPE VARCHAR(40) NOT NULL;
```

g. Delete plays staged in Audi 1:

```
DELETE FROM THEATRE WHERE AUDI = 1;
```


10. Emerging Trends

Assessment

A.

1. a	2. b	3. a	4. d	5. b
6. d	7. a	8. b	9. d	

B.

1. False	2. True	3. True	4. False	5. True
6. True	7. False	8. True	9. True	10. True
11. False				

C.

1. Reinforcement	2. NLP	3. Augmented	4. IoT
5. accelerometer	6. PaaS	7. Hybrid cloud	
8. Public, private	9. blockchain		

D. 1.

(i) **AI:** Artificial Intelligence (AI) is the simulation of human intelligence processes by machines, especially computer systems. These processes include learning, reasoning, and self-correction.

(ii) **ML:** Machine Learning (ML) is a subset of AI that involves the use of algorithms and statistical models to enable computers to improve their performance on a task through experience and data.

(iii) **Big Data:** Big Data refers to extremely large datasets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

(iv) **Blockchain:** Blockchain is a distributed ledger technology that securely records transactions across many computers in such a way that the registered transactions cannot be altered retroactively.

IMPRINT 1

2. **Machine Learning:** Machine Learning is a method of data analysis that automates analytical model building, using algorithms that iteratively learn from data to find hidden insights without being explicitly programmed.

   Supervised Learning: In supervised learning, the model is trained on a labeled dataset, which means that each training example is paired with an output label. The model makes predictions and is corrected when wrong, learning over time.

   Unsupervised Learning: In unsupervised learning, the model is given data without explicit instructions on what to do with it. The model tries to identify patterns and relationships in the data without prior training.

3. **Ambiguity:** Natural language is often ambiguous, meaning that the same word or sentence can have multiple interpretations. Resolving this ambiguity is a significant challenge.

   Context Understanding: Understanding the context in which words are used is crucial for accurate interpretation. This involves understanding nuances, idioms, and the cultural context, which is difficult for machines.

4. **Augmented Reality (AR):** AR overlays digital content on the real-world environment. Users can see and interact with both the physical and digital worlds simultaneously.

   Virtual Reality (VR): VR immerses users in a completely virtual environment, replacing the physical world entirely. Users interact with this environment through a headset and controllers.

5. **Healthcare:** Predicting disease outbreaks, personalized medicine, and improving patient care.

   Finance: Fraud detection, risk management, and algorithmic trading.

   Retail: Customer behavior analysis, inventory management, and personalized marketing.

   Transportation: Traffic management, route optimization, and predictive maintenance.

6. IoT can significantly enhance the efficiency, convenience, and safety of various aspects of life and work. For example, smart homes can automate lighting, heating, and security, while smart cities can optimize traffic flow and energy usage.

   Purpose of IoT sensors:

   - Gyroscopes: Measure the orientation and angular velocity, used in navigation systems.
   - Proximity Sensors: Detect the presence of objects or people, used in automated doors and security systems.
   - Light Sensors: Measure the intensity of light, used in automatic lighting systems and smartphones.

   IoT for Smart Cities:

   - Key Technology: IoT enables real-time monitoring and management of city infrastructure, improving efficiency and reducing costs. Examples include smart traffic lights, waste management systems, and energy grids.

7. Cloud computing is the delivery of computing services (servers, storage, databases, networking, software, etc.) over the Internet (the cloud).

   Popularity Reasons: Cost efficiency, scalability, flexibility, accessibility, and maintenance handled by service providers.

8. **Private Cloud:** Cloud infrastructure operated solely for a single organization, offering greater control and security.

   Public Cloud: Cloud infrastructure provided by third-party providers over the Internet, shared among multiple organizations.

   Hybrid Cloud: A combination of private and public clouds, allowing data and applications to be shared between them for greater flexibility and optimization.

9. Grid computing is a distributed computing model that uses a network of computers to work together to solve complex problems.

   Advantage: Resource Utilization - It allows efficient use of underutilized resources across multiple locations.

10. Blockchain is a decentralized ledger technology where transactions are recorded in blocks. Each block is linked to the previous one through cryptographic hashes, forming a chain. This ensures data integrity and security, as altering a single block would require changes to all subsequent blocks.

11. **Security:** Transactions are encrypted and linked to previous transactions, making it difficult to alter data without detection.

    Transparency: All participants in the network have access to the same information, ensuring accountability and transparency.

Assertion and Reasoning Based Questions

1. c. 2. b 3.a

Case-based Questions

1. Suvidha can use **Natural Language Processing (NLP)** technology to convert her speech to text. This technique of artificial intelligence will help in transcribing spoken words into a text document.
2. Ashraf is implementing a **supervised learning** machine learning model. This type of model can analyze the collected data to identify patterns and make predictions or classifications based on labeled data, such as identifying age groups and comparing dropout rates between private and government schools.

IMPRINT 1