UNIT-I

Define Computer and brief about its Hardware and Software?

A computer is an electronic device that is designed to perform various tasks by executing programs and processing data. It is composed of both hardware and software components that work together to accomplish these tasks.

Hardware refers to the physical components of a computer that can be touched and seen. Examples of hardware components include the central processing unit (CPU), random access memory (RAM), hard disk drive (HDD), motherboard, graphics card, and input/output devices such as the keyboard, mouse, and monitor.

Software refers to the programs and data that are used by the computer to perform specific tasks. Examples of software include the operating system, applications, and utilities. The operating system is responsible for managing the computer's hardware and software resources, and it provides a platform for other software applications to run. Applications are programs that are designed for specific purposes, such as word processing, spreadsheets, and photo editing.

Some examples of popular computers and their hardware and software are:

Desktop computer: hardware components include a CPU, RAM, HDD or solid-state drive (SSD), motherboard, graphics card, keyboard, mouse, and monitor. Popular operating systems for desktop computers include Windows and macOS. Popular applications include Microsoft Office, Adobe Photoshop, and Google Chrome.

Laptop computer: hardware components are similar to desktop computers but are typically smaller and more portable. Popular operating systems for laptops include Windows and macOS. Popular applications include Microsoft Office, Adobe Creative Suite, and web browsers like Chrome and Safari.

Tablet computer: a portable device that is designed primarily for web browsing, media consumption, and some basic productivity tasks. Popular examples of tablet computers include the Apple iPad and Samsung Galaxy Tab. These devices typically use a touch screen interface and may have a detachable keyboard. Popular operating systems for tablets include iOS and Android. Popular applications include social media apps, video streaming services, and web browsers.

Explain about the types of Programming Languages?

There are several types of programming languages, each with its own set of characteristics and purposes. Some of the most common types of programming languages are:

Procedural Programming Languages: These are the oldest and most commonly used programming languages. They use a sequence of commands to tell the computer what to do, and the program is executed in a linear fashion, one statement after another. Examples of procedural programming languages include C, Pascal, and Fortran.

Object-Oriented Programming Languages: Object-oriented programming (OOP) is a programming paradigm that uses objects to represent data and behavior. These languages use classes and objects to create reusable code, encapsulate data, and provide inheritance. Examples of object-oriented programming languages include Java, Python, and C++.

Functional Programming Languages: Functional programming languages are based on the concept of mathematical functions. They treat computation as the evaluation of functions and avoid changing state and mutable data. Examples of functional programming languages include Haskell, Lisp, and Erlang.

Scripting Programming Languages: Scripting languages are interpreted programming languages that are used for scripting tasks such as automating repetitive tasks, web development, and data analysis. Examples of scripting languages include Python, Ruby, and JavaScript.

Markup Languages: Markup languages are used to create and format documents, web pages, and other text-based data. They use tags and attributes to define the structure and presentation of the content. Examples of markup languages include HTML, XML, and Markdown.

Query Languages: Query languages are used to retrieve and manipulate data stored in databases. They provide a way to specify what data is required and how to retrieve it. Examples of query languages include SQL, MongoDB Query Language (MQL), and GraphQL.

Low-Level Programming Languages: Low-level programming languages are used to write code that interacts directly with the computer's hardware, such as device drivers and operating system components. They provide direct access to memory and the CPU, allowing for more efficient code. Examples of low-level programming languages include Assembly language and Machine code.

Each type of programming language has its own strengths and weaknesses, and different languages are suited for different tasks. Developers choose programming languages based on factors such as the project requirements, the developer's experience and preference, and the availability of libraries and tools.

Explain the steps to write an Algorithm. Write an algorithm to find sum of two numbers?

An algorithm is a step-by-step procedure for solving a problem or performing a task. To write an algorithm, follow these steps:

Define the problem: Clearly define the problem you want to solve or the task you want to perform. Identify the inputs and outputs of the algorithm.

Determine the approach: Choose a general approach for solving the problem, such as using a specific algorithmic technique or data structure.

Create a flowchart: Draw a diagram that shows the logical steps of the algorithm. Use symbols to represent each step, such as a rectangle for a process or a diamond for a decision point.

Write pseudocode: Use English-like statements to describe each step in the algorithm. The pseudocode should be easy to understand and not include programming syntax.

Review and refine: Review the algorithm and refine it as needed. Test the algorithm with sample inputs and outputs to ensure that it works as expected.

Here is an example algorithm in pseudocode that finds the sum of two numbers:

```
Algorithm: Find Sum of Two Numbers
Inputs: num1, num2 (two numbers)
Output: sum (the sum of the two numbers)

1. Start
2. Read num1 and num2 from the user
3. Add num1 and num2
4. Store the result in the variable sum
5. Print the value of sum
6. Stop
```

In this algorithm, the inputs are two numbers (num1 and num2), and the output is the sum of the two numbers (sum). The algorithm reads the two numbers from the user, adds them together, stores the result in the sum variable, and then prints the value of sum.

Define Flowchart and brief about the symbols of flowchart with an example program?

A flowchart is a graphical representation of an algorithm or a process that uses symbols and arrows to show the flow of steps. Flowcharts are often used by programmers, analysts, and engineers to document, plan, and analyze processes and systems.

The symbols used in flowcharts represent different types of actions, decisions, and inputs/outputs. Some of the most common symbols used in flowcharts are:

Start/End Symbol: The start/end symbol represents the beginning and end of a flowchart. It is usually an oval or rounded rectangle.

Process Symbol: The process symbol represents a step or action in the flowchart. It is usually a rectangle or other geometric shape.

Decision Symbol: The decision symbol represents a decision point in the flowchart where a choice must be made. It is usually a diamond-shaped symbol.

Input/Output Symbol: The input/output symbol represents the input or output of data to or from the process. It is usually a parallelogram or trapezoid.

Connector Symbol: The connector symbol is used to connect different parts of the flowchart. It is usually a small circle or dot.

Here is an example flowchart that represents the same algorithm as the one provided in the previous question, finding the sum of two numbers:

```
          [Start]
             |
             v
        [Input num1]
             |
             v
        [Input num2]
             |
             v
```

```
                [Process: Add]
                          |
                          v
              [Output: sum = num1 + num2]
                          |
                          v
                     [End]
```

In this flowchart, the start symbol is represented by the word "Start" in a rectangle. The input symbols are represented by parallelograms labeled "Input num1" and "Input num2". The process symbol is represented by a rectangle labeled "Add", and the output symbol is represented by a parallelogram labeled "Output: sum = num1 + num2". The end symbol is represented by the word "End" in a rectangle.

The symbols are connected by arrows to show the flow of the algorithm from one step to the next. In this flowchart, the arrows connect the symbols in the order that the algorithm is executed, from input to output.

5) . a) Define Python and write a short note on features of Python Programming?
b) Why Python is called as an Interpreted Language? Briefly explain about PVM with neat diagram?

a) Python is a popular high-level, interpreted programming language that was first released in 1991. It is designed to be easy to read and write, and to emphasize code readability and simplicity. Python is known for its clean syntax, ease of use, and versatility, and it has become one of the most widely used programming languages in the world.

Some of the key features of Python programming include:

Simple and easy to learn: Python has a simple and easy-to-learn syntax that emphasizes code readability and simplicity, making it an ideal language for beginners.

Cross-platform: Python is a cross-platform language, meaning that it can run on a wide range of operating systems and hardware architectures, including Windows, Linux, macOS, and more.

Large standard library: Python comes with a large standard library that includes a wide range of modules and functions for tasks such as file I/O, network programming, web development, and more.

Dynamic typing: Python is dynamically typed, which means that variable types are determined at runtime, making it easier to write code quickly and without worrying about variable types.

Object-oriented: Python is an object-oriented language, which means that it supports object-oriented programming paradigms such as encapsulation, inheritance, and polymorphism.

Interpreted: Python is an interpreted language, meaning that code is executed directly by the interpreter, rather than being compiled into machine code like some other languages.

Large community: Python has a large and active community of developers, which means that there are plenty of resources, libraries, and tools available to help with programming.

Overall, Python is a powerful and versatile language that can be used for a wide range of applications, including web development, data analysis, machine learning, game development, and more. Its simplicity, flexibility, and readability make it a popular choice for both beginners and experienced programmers alike.

b) Python is called an interpreted language because Python code is executed by an interpreter rather than being compiled into machine code like some other languages. The interpreter reads each line of code and executes it immediately, rather than compiling the entire program into machine code before execution.

When a Python program is run, the Python interpreter reads the code line by line and executes it in real-time. This makes it easy to write and debug code quickly, as changes to the code can be seen and tested immediately. However, it can also make Python programs slower than compiled languages like C++ or Java.

PVM, or the Python Virtual Machine, is the component of the Python interpreter that executes Python bytecode. Python code is compiled into bytecode, which is a low-level, platform-independent representation of the code. The PVM then reads and executes the bytecode, which allows the same Python code to run on any platform that has a Python interpreter.

Here is a simple diagram that illustrates the relationship between the Python interpreter, the PVM, and the Python code:

```
        +-----------------+
        |Python Interpreter|
        +-----------------+
                 |
                 |
                 v
        +-----------------+
        |   PVM (Python   |
        |  Virtual Machine)|
        +-----------------+
                 |
                 |
                 v
        +-----------------+
        |   Python Code   |
        +-----------------+
```

In this diagram, the Python code is compiled into bytecode and executed by the PVM, which is part of the Python interpreter. The interpreter reads the code line by line and executes it in real-time, making it an interpreted language.

The PVM is responsible for managing memory, executing bytecode, and handling exceptions, among other tasks. It is designed to be efficient and portable, which allows Python code to run on a wide range of platforms without modification.

6. a) List the steps used to view the byte code of a python program?
b) Distinguish between C and Python programming?

a) the steps to view the bytecode of a Python program:

Write your Python code in a text editor and save it as a .py file. For example, you could create a file called example.py.

Open a command prompt or terminal window.

Navigate to the directory where the example.py file is located.

Type python -m py_compile example.py and press Enter. This will compile the Python code in the example.py file and create a bytecode file called example.pyc in the same directory.

Type python -m dis example and press Enter. This will disassemble the bytecode in the example.pyc file and display it in a human-readable format.

The output will show the bytecode of the example.py file in a human-readable format.

Note that the py_compile module compiles the Python code in the specified file to bytecode, and the dis module disassembles the bytecode and displays it in a human-readable format. This can be useful for understanding how Python code is executed at a low level and for debugging complex programs.

Example Program:

```
import dis

def add_numbers(a, b):
    return a + b

result = add_numbers(5, 10)
print(result)

dis.dis(add_numbers)
```

This program defines a function called add_numbers() that takes two arguments and returns their sum. It then calls the add_numbers() function with the arguments 5 and 10 and prints the result. Finally, it uses the dis module to disassemble the bytecode of the add_numbers() function and display it in a human-readable format.

When you run this program, the output will show the bytecode of the add_numbers() function in a human-readable format:

Output:

```
  3           0 LOAD_FAST                0 (a)
              2 LOAD_FAST                1 (b)
              4 BINARY_ADD
              6 RETURN_VALUE
```

7. Brief about comments and documentation string in python program with a sample program?

Comments and documentation strings are both used in Python to add explanatory text to your code. While comments are used to annotate your code for the benefit of other programmers, documentation strings are used to provide documentation for the users of your code.

In Python, comments are denoted by a hash symbol (#). Anything that follows the hash symbol on a given line is treated as a comment and is ignored by the Python interpreter. Here's an example:

Program:

```
# This is a comment. It is ignored by the Python interpreter.
print("Hello, World!")  # This is another comment.
```

Documentation strings, on the other hand, are a special type of comment that are used to provide documentation for functions, modules, classes, and methods. They are enclosed in triple quotes (""") and are placed immediately after the definition of the function, module, class, or method. Here's an example:

```
def add_numbers(a, b):
    """This function takes two numbers as arguments and returns their sum."""
    return a + b
```

In this example, the documentation string provides a brief description of what the add_numbers() function does.

Documentation strings are often referred to as "docstrings" in Python, and they can be accessed using the __doc__ attribute of the function, module, class, or method. Here's an example of how to access the docstring for the add_numbers() function:

```
print(add_numbers.__doc__)
```

In summary, comments and documentation strings are both important tools for adding explanatory text to your Python code. Comments are used to annotate your code for other programmers, while documentation strings are used to provide documentation for the users of your code.

UNIT-II

**1. Classify the built-in data types based on mutable and immutable. Explain any two data types of each with
example program.**

In Python, the built-in data types can be classified based on whether they are mutable or immutable. Mutable data types are those that can be changed after they are created, while immutable data types are those that cannot be changed once they are created. Here are some examples of built-in data types and their mutability:

Immutable data types:

Numbers (int, float, complex)
Strings
Tuples
Mutable data types:

Lists
Dictionaries
Sets
Here are some examples of two immutable data types and two mutable data types, along with example programs:

Immutable Data Types:

Numbers - The number data type in Python includes integers, floating-point numbers, and complex numbers. Once a number is created, it cannot be changed. For example:

```
x = 5   # integer
y = 3.14   # floating-point number
z = 2 + 3j   # complex number
```

Strings - The string data type represents a sequence of characters. Once a string is created, it cannot be changed. For example:

```
s = "Hello, World!"   # string
t = 'This is a string.'   # string
u = """This is a
multiline string."""   # multiline string
```

Mutable Data Types:

Lists - A list is an ordered collection of items, which can be of any data type. Lists are mutable, which means that you can add, remove, or modify items after the list is created. For example:

```
my_list = [1, 2, 3, "apple", "banana", "cherry"]   # a list of integers and strings
my_list.append(4)   # add an item to the end of the list
my_list[3] = "orange"   # change an item in the list
```

Dictionaries - A dictionary is an unordered collection of key-value pairs. Dictionaries are mutable, which means that you can add, remove, or modify items after the dictionary is created. For example:

```
my_dict = {"name": "John", "age": 30, "city": "New York"}   # a dictionary of strings and integers
my_dict["age"] = 31   # change the value of a key
my_dict["country"] = "USA"   # add a new key-value pair
```

In summary, mutable data types can be changed after they are created, while immutable data types cannot be changed once they are created. Understanding the mutability of data types is important for designing efficient and bug-free programs.

2. a) Explain the following built-in data types
i) None
ii) Numeric
iii) Bool
iv) Strings
b) Describe the list comprehension with an example program?

a) i) **None**: None is a special data type in Python that represents the absence of a value. It is commonly used to indicate that a variable or function returns nothing. For example, a function that performs a computation but doesn't need to return anything can have a None return value:

```
def my_function():
    # do some computation
    return None
```

ii) **Numeric**: Numeric data types in Python include integers, floating-point numbers, and complex numbers. They are used for arithmetic operations, and support the usual mathematical operations like addition, subtraction, multiplication, and division. Examples of numeric data types are:

- Integers: Integer data type represents positive or negative whole numbers without a decimal point. Example: x = 10.
- Floating-point numbers: Floating-point data type represents real numbers with a decimal point. Example: y = 3.14.
- Complex numbers: Complex number data type represents numbers in the form of a + bj, where a and b are real numbers and j is the square root of -1. Example: z = 2 + 3j.

iii) **Bool**: The boolean data type in Python represents logical values that can be either True or False. Boolean values are used to control the flow of a program, such as in conditional statements and loops. For example:

```
x = 10

y = 5

print(x > y)   # outputs True

print(x < y)   # outputs False
```

iv) **Strings**: The string data type in Python represents a sequence of characters. Strings are used for representing text and can be enclosed in either single or double quotes. They support many useful operations such as indexing, slicing, concatenation, and formatting. Examples of string data types are:

```
my_string = "Hello, World!"
your_string = 'This is a string.'
```

In summary, these built-in data types provide a wide range of functionality and can be used to represent a variety of information in Python programs.

3. List few operations and methods on Tuple and dictionaries with an example programs?

Some common operations and methods on tuples and dictionaries in Python, along with example programs:

Tuples:

Creating a tuple:

```
my_tuple = (1, 2, 3)
```

2) Accessing elements of a tuple using indexing:

```
my_tuple = (1, 2, 3)

print(my_tuple[0])  # prints 1
```

3) Concatenating tuples:

```
my_tuple = (1, 2, 3)

your_tuple = (4, 5, 6)

our_tuple = my_tuple + your_tuple

print(our_tuple)  # prints (1, 2, 3, 4, 5, 6)
```

4. Repeating a tuple:

```
my_tuple = (1, 2, 3)

repeated_tuple = my_tuple * 3
```

```
print(repeated_tuple)  # prints (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

5. Finding the length of a tuple:

```
my_tuple = (1, 2, 3)

print(len(my_tuple))  # prints 3
```

6. Converting a list to a tuple:

```
my_list = [1, 2, 3]

my_tuple = tuple(my_list)

print(my_tuple)  # prints (1, 2, 3)
```

Dictionaries:

1. Creating a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

2. Accessing values from a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

print(my_dict['name'])  # prints 'Alice'
```

3. Adding a key-value pair to a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

```
my_dict['gender'] = 'Female'

print(my_dict)  # prints {'name': 'Alice', 'age': 30, 'city': 'New York', 'gender': 'Female'}
```

4. Removing a key-value pair from a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

del my_dict['age']

print(my_dict)  # prints {'name': 'Alice', 'city': 'New York'}
```

5. Checking if a key exists in a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

print('name' in my_dict)  # prints True

print('state' in my_dict)  # prints False
```

6. Getting all keys or values from a dictionary:

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

print(my_dict.keys())  # prints dict_keys(['name', 'age', 'city'])

print(my_dict.values())  # prints dict_values(['Alice', 30, 'New York'])
```

In summary, these are just a few examples of the many operations and methods available for tuples and dictionaries in Python.

4. Brief about Set and frozenset with an example program?

In Python, a set is an unordered collection of unique elements, while a frozenset is an immutable version of a set. Here's a brief overview of each with an example program:

Sets:

1. Creating a set:

```
my_set = {1, 2, 3}
```

2. Adding elements to a set:

```
my_set = {1, 2, 3}

my_set.add(4)

print(my_set)  # prints {1, 2, 3, 4}
```

3. Removing elements from a set:

```
my_set = {1, 2, 3}

my_set.remove(2)

print(my_set)  # prints {1, 3}
```

4. Combining sets:

```
my_set = {1, 2, 3}
```

```
your_set = {2, 3, 4}

our_set = my_set.union(your_set)

print(our_set)  # prints {1, 2, 3, 4}
```

5. Checking if an element is in a set:

```
my_set = {1, 2, 3}

print(2 in my_set)  # prints True

print(4 in my_set)  # prints False
```

Frozen sets:

1. Creating a frozenset:

```
my_frozenset = frozenset([1, 2, 3])
```

2. Accessing elements of a frozenset:

```
my_frozenset = frozenset([1, 2, 3])

for element in my_frozenset:

    print(element)
```

3. Combining frozensets:

```
my_frozenset = frozenset([1, 2, 3])
```

```
your_frozenset = frozenset([2, 3, 4])

our_frozenset = my_frozenset.union(your_frozenset)

print(our_frozenset)  # prints frozenset({1, 2, 3, 4})
```

4. Converting a set to a frozenset:

```
my_set = {1, 2, 3}

my_frozenset = frozenset(my_set)

print(my_frozenset)  # prints frozenset({1, 2, 3})
```

In summary, a set is a mutable collection of unique elements, while a frozenset is an immutable version of a set. Both are useful for managing collections of data in Python, with the main difference being that sets can be modified while frozensets cannot.

5. State the use of constants, identifiers, reserve words and naming conventions?

In Python programming, constants, identifiers, reserve words, and naming conventions are used for different purposes as follows:

1. Constants: Constants are variables whose values do not change during the execution of the program. Constants are created by assigning a value to a variable, and this value cannot be changed later. In Python, constant variables are usually written in all capital letters.

2. Identifiers: Identifiers are names given to entities such as variables, functions, classes, etc. in a Python program. An identifier can consist of letters, digits, and underscores, but it cannot start with a digit. In addition, Python is case-sensitive, which means that an identifier in uppercase and lowercase letters are treated as different.

3. Reserved words: Reserved words are words that have a predefined meaning in the Python language, and they cannot be used as variable names, function names, or any other identifiers. Examples of reserved words in Python include if, else, while, for, class, def, and, or, not, etc.
4. Naming conventions: Naming conventions are rules for giving names to different entities in a Python program to make the code more readable and consistent. For example, it is common to use lowercase letters and underscores to separate words in variable and function names. In addition, class names usually start with an uppercase letter, and module names are usually written in lowercase letters. Here are some examples of naming conventions:

- Variables: my_variable, age, height
- Functions: my_function, calculate_area
- Classes: MyClass, Person
- Modules: my_module.py, helper_functions.py

In summary, constants, identifiers, reserve words, and naming conventions are all important aspects of Python programming. They are used to make the code more organized, readable, and consistent, and to avoid common errors and mistakes in the code.

6. Demonstrate the following with an example program?

i) Assignment operator

ii) Logical operator

iii) Logical operator

iv) Bitwise operator

Here are examples of Python programs that demonstrate the use of different types of operators:

i) Assignment operator:

The assignment operator = is used to assign a value to a variable in Python. The value on the right side of the = is assigned to the variable on the left side. Here is an example:

```
x = 5

y = 3

z = x + y

print(z)  # Output: 8
```

In this example, the variables x and y are assigned the values 5 and 3, respectively. The variable z is assigned the sum of x and y, which is 8.

ii) Logical operator:

Logical operators are used to combine two or more conditions in a Python program. The three main logical operators in Python are and, or, and not. Here is an example:

```
x = 5

y = 3

z = 7

if x > y and x < z:

    print("x is between y and z")

else:

    print("x is not between y and z")
```

In this example, the and operator is used to combine two conditions: x > y and x < z. If both conditions are true, the message "x is between y and z" is printed. Otherwise, the message "x is not between y and z" is printed.

iii) Comparison operator:

Comparison operators are used to compare two values in a Python program. The result of a comparison is a Boolean value (True or False). Here is an example:

```
x = 5

y = 3

if x > y:

    print("x is greater than y")

else:

    print("x is not greater than y")
```

In this example, the > operator is used to compare the values of x and y. Since x is greater than y, the message "x is greater than y" is printed.

iv) Bitwise operator:

Bitwise operators are used to perform operations on the binary representation of numbers in a Python program. The three main bitwise operators in Python are & (AND), | (OR), and ^ (XOR). Here is an example:

```
x = 7

y = 3

z = x & y
```

```
print(z)  # Output: 3
```

In this example, the & operator is used to perform a bitwise AND operation on the binary representation of x and y. The result is a binary number 0011, which is equivalent to the decimal number 3. The value of z is therefore 3.

7. Demonstrate the following with an example programs?

i) Relational Operator

ii) Membership operator

iii) Identity operator

iv) Operator precedence and associativity

Python programs that demonstrate the use of different types of operators:

i) Relational Operator:

Relational operators are used to compare two values in a Python program. The result of a comparison is a Boolean value (True or False). The main relational operators in Python are >, <, >=, <=, ==, and !=. Here is an example:

```
x = 5

y = 3

if x > y:

    print("x is greater than y")

else:
    print("x is not greater than y")
```

In this example, the > operator is used to compare the values of x and y. Since x is greater than y, the message "x is greater than y" is printed.

ii) Membership operator:

Membership operators are used to test whether a value is a member of a sequence (such as a string, list, or tuple) in a Python program. The two main membership operators in Python are in and not in. Here is an example:

```
x = "Hello, world!"

if "world" in x:

    print("The string 'world' is a member of x")

else:

    print("The string 'world' is not a member of x")
```

In this example, the in operator is used to test whether the string "world" is a member of the string x. Since it is, the message "The string 'world' is a member of x" is printed.

iii) Identity operator:

Identity operators are used to compare the memory addresses of two objects in a Python program. The two main identity operators in Python are is and is not. Here is an example:

```
x = [1, 2, 3]

y = [1, 2, 3]

if x is y:
    print("x and y refer to the same object")

else:

    print("x and y do not refer to the same object")
```

In this example, the is operator is used to compare the memory addresses of the lists x and y. Since x and y are different objects (even though they have the same values), the message "x and y do not refer to the same object" is printed.

iv) Operator Precedence and Associativity:

Operator precedence determines the order in which operators are evaluated in a Python expression. For example, multiplication (*) has a higher precedence than addition (+), so the expression 2 + 3 * 4 is evaluated as 2 + (3 * 4), which equals 14.

Operator associativity determines the order in which operators with the same precedence are evaluated. For example, the operators * and / have the same precedence, so the expression 4 / 2 * 2 could be evaluated as (4 / 2) * 2, which equals 4, or as 4 / (2 * 2), which equals 1.

Here is an example:

```
x = 10

y = 5

z = 2

result = x / y * z

print(result)  # Output: 4.0
```

In this example, the expression x / y * z is evaluated as (x / y) * z, which equals 4.0. This is because the / and * operators have the same precedence, and they are evaluated from left to right.

8. Describe about input and output statements with a sample program for each type of statement?

In Python, the input and output statements are used to interact with the user and the system.

The input() function is used to accept input from the user. It reads a line of text from the standard input and returns it as a string. Here's an example program that uses the input() function to get the user's name and then prints a greeting:

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

print("Hello, " + name + "!")
```

The output statements in Python are used to print text to the console or to a file. The most commonly used output statement is the print() function, which takes one or more expressions as arguments and prints them to the console. Here's an example program that uses the print() function to display the sum of two numbers:

```
x = 5

y = 7

print("The sum of", x, "and", y, "is", x + y)
```

In this example, the print() function takes four arguments, separated by commas. The first three arguments are strings that contain text and the values of x and y. The fourth argument is an expression that adds x and y. When the program is run, the output will be:

```
The sum of 5 and 7 is 12
```

9. What is a command line argument? Explain briefly how to implement it in python programming with an examples?

Command-line arguments are values passed to a program when it is run from the command line. These arguments allow a user to customize the behavior of a program without modifying the program's source code. In Python, we can access command-line arguments using the sys.argv variable.

The sys.argv variable is a list that contains the command-line arguments passed to the program. The first element of the list (sys.argv[0]) is the name of the program itself, and the remaining elements are the arguments. For example, if we run a Python program with the command python my_program.py arg1 arg2, then sys.argv will be ['my_program.py', 'arg1', 'arg2'].

Here is an example program that accepts a command-line argument and prints it to the console:

```
import sys



if len(sys.argv) > 1:

    argument = sys.argv[1]

    print("The argument is:", argument)

else:

    print("No argument provided.")
```

In this program, we first check whether there is at least one command-line argument (len(sys.argv) > 1). If there is, we access the first argument (sys.argv[1]) and store it in the argument variable. We then print a message that includes the value of the argument. If no argument is provided, we print a message indicating that.

To run this program with a command-line argument, we can use a command like python my_program.py hello. This will produce the output:

```
The argument is: hello
```

If we run the program without any arguments (python my_program.py), the output will be:

```
No argument provided.
```

~Prepared by CSE Beta Students