**Course Code : MCS-201**

**Course Title : Programming in C and PYTHON**

**Assignment Number : PGDCA(I)/201/Assignment/2023**

**Last Date of Submission : 30th April, 2023 (for January session)**
**31st October, 2023 (for July session)**

**There are sixteen questions in this assignment (eight in each section i.e. Section A and Section B) which carries 80 marks. Each question carries 5 marks. Rest 20 marks are for viva-voce. Answer all the questions from both the sections i.e. Section A and Section B. You April use illustrations and diagrams to enhance the explanations. Include the screen layouts also along with your assignment responses. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation.**
**SECTION-A (C-Programming)**

**Question 1. Compare flowchart and algorithm. Write Algorithm and also draw flowchart to perform following:**
**a. Find factorial of a number entered by user.**
**b. Print Fibonacci series up to the number of terms entered by the uses**
**Ans.**

A flowchart and an algorithm are both used to represent a sequence of steps or instructions to solve a problem or perform a task. However, there are some differences between the two:

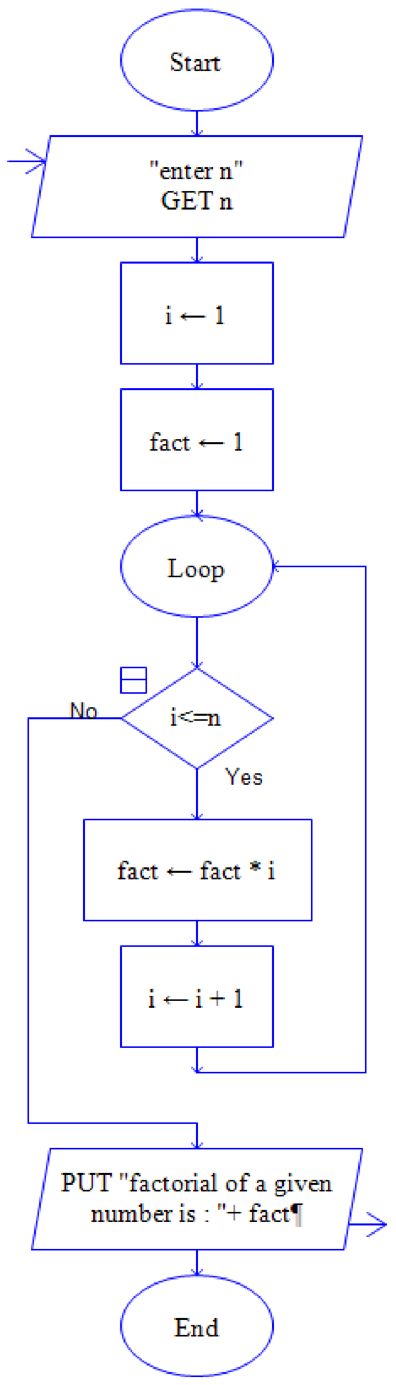
1. Definition: A flowchart is a visual representation of a process, showing the steps involved and the order in which they are executed. An algorithm is a step-by-step procedure for solving a problem or performing a task.
2. Representation: A flowchart is represented in a graphical form, using symbols and arrows to depict the flow of the process. An algorithm, on the other hand, is usually represented in a textual form, using a programming language or a pseudo-code.
3. Use: A flowchart is often used in the initial stages of software development, to help the developers understand the flow of the process and identify potential problems. An algorithm is used by programmers to write actual code that will implement the solution.
4. Level of detail: A flowchart can be more detailed and complex than an algorithm, as it can include graphical representations of decision points, loops, and other program structures. An algorithm is usually more high-level, describing the steps in a more abstract way.
5. Purpose: A flowchart is used to help visualize a process or system, while an algorithm is used to solve a specific problem or perform a specific task.

In summary, while both flowcharts and algorithms are useful tools for planning and developing programs, they have different purposes and representations. Flowcharts are more visual and can be used to represent complex processes, while algorithms are more focused on solving specific problems and are represented in a textual form.

**a)**

Here is the algorithm to find the factorial of a number entered by the user:

1. Read an integer value from the user and store it in a variable "n".
2. Initialize a variable "fact" to 1.
3. For each number "i" from 1 to "n", do the following: a. Multiply "fact" by "i".
4. Display the value of "fact" as the factorial of "n".



**b) Fibonacci Series Algorithm:**

- Start
- Declare variables i, a,b , show
- Initialize the variables, a=0, b=1, and show =0
- Enter the number of terms of Fibonacci series to be printed
- Print First two terms of series
- Use loop for the following steps

  -> show=a+b

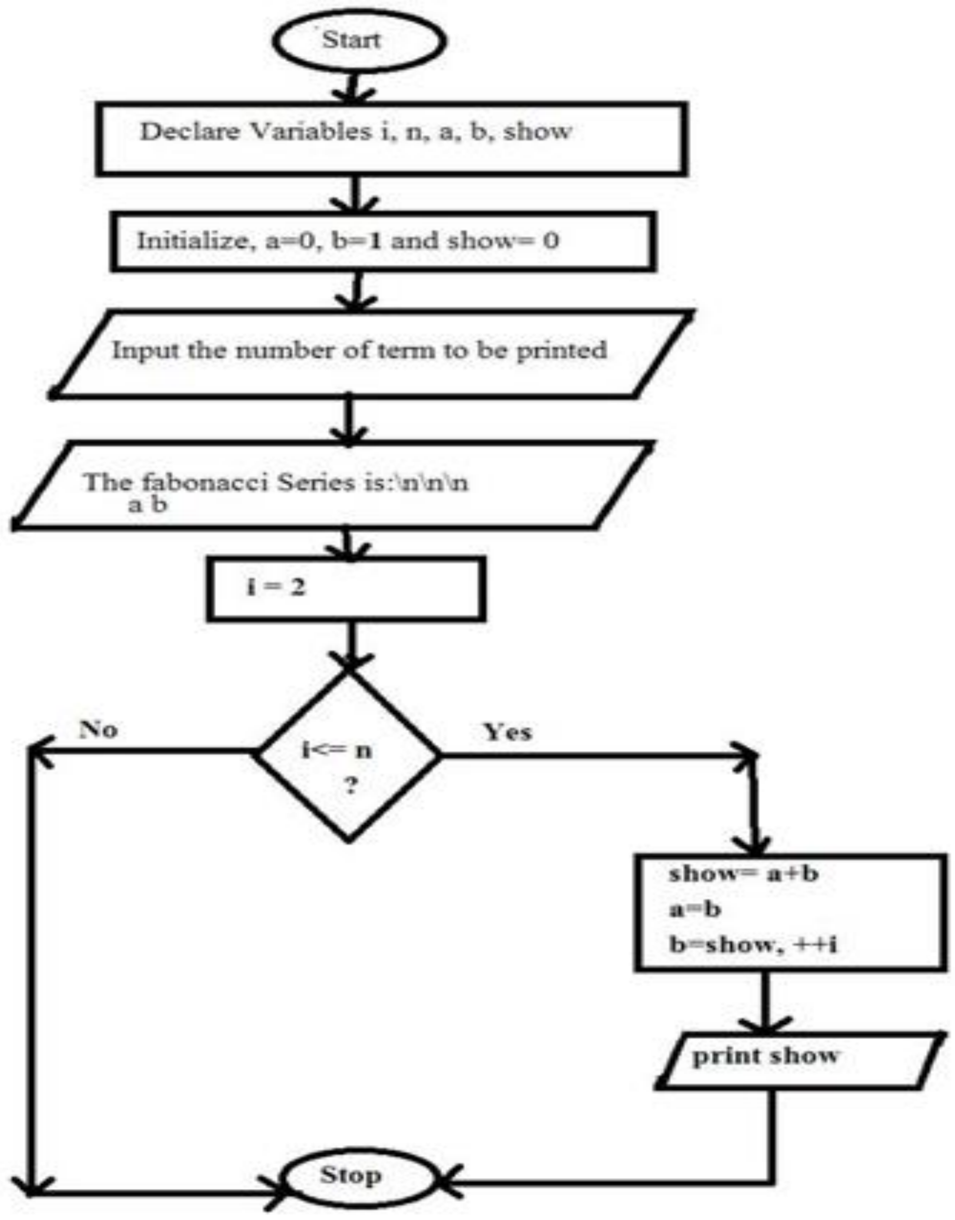
  -> a=b

  -> b=show

-> increase value of i each time by 1

-> print the value of show

- End


Start
Declare Variables i, n, a, b, show
Initialize, a=0, b=1 and show= 0
Input the number of term to be printed
The fabonacci Series is:\n\n\n a b
i = 2
No
i<= n ?
Yes
show= a+b
a=b
b=show, ++i
print show
Stop

**Question 2. Differentiate between Recursion and Iteration. Give suitable code to find factorial of a number entered by user in C for each.**

**Ans.**

Recursion and iteration are two ways of solving a problem by repeating a set of instructions. The main difference between recursion and iteration is the way in which the instructions are repeated.

Recursion involves a function calling itself repeatedly until a base condition is met. Each function call creates a new instance of the function on the call stack, with its own set of variables and parameters. Recursion can be a powerful tool for solving problems that can be broken down into smaller sub-problems.

Iteration, on the other hand, involves using loops to repeat a set of instructions a certain number of times or until a condition is met. Iteration is often used when the problem requires a fixed number of repetitions or when it is not practical to use recursion.

**Here's the C code to find the factorial of a number entered by the user using recursion:**

```
#include <stdio.h>
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int n, fact;
    printf("Enter a number: ");
    scanf("%d", &n);
    fact = factorial(n);
    printf("Factorial of %d is %d\n", n, fact);
    return 0;
}
```

**And here's the C code to find the factorial of a number entered by the user using iteration:**

```
#include <stdio.h>

int factorial(int n) {
    int fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

int main() {
    int n, fact;
    printf("Enter a number: ");
    scanf("%d", &n);
    fact = factorial(n);
    printf("Factorial of %d is %d\n", n, fact);
    return 0;
}
```

In the above code, the **factorial** function calculates the factorial of a number using recursion or iteration depending on the implementation. The **main** function reads a number from the user, calls the **factorial** function, and displays the result.

**Question 3. Explain the concept of call by reference, with suitable code in C for each. Give advantage and disadvantage of call by reference**
**Ans.**

In call by reference, a reference to a variable is passed to a function as a parameter, rather than a copy of the variable. This means that any changes made to the variable inside the function are reflected in the original variable outside the function as well. Call by reference is implemented in C using pointers.

**Here's an example code in C that demonstrates call by reference:**

```
#include <stdio.h>

void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}

int main() {
   int a = 5, b = 10;
   printf("Before swap: a = %d, b = %d\n", a, b);
   swap(&a, &b);
   printf("After swap: a = %d, b = %d\n", a, b);
   return 0;
}
```

In the above code, the **swap** function takes two integer pointers as parameters and swaps the values of the variables they point to. The **main** function initializes two integer variables **a** and **b** with values 5 and 10 respectively, displays their values before calling the **swap** function, calls the **swap** function by passing the addresses of **a** and **b**, and displays their values again after the function call. As you can see, the values of **a** and **b** are swapped even outside the **swap** function, because the function modified the values indirectly through their pointers.

Advantages of call by reference:

- Can modify the original variables passed as parameters, rather than creating copies that are discarded after the function call.
- Saves memory by avoiding the creation of extra copies of variables.

Disadvantages of call by reference:

- Can lead to unintended changes to the original variables if not used carefully.

- Requires more knowledge and care in handling pointers, which can be error-prone and difficult to debug.

**Question 4. Write an algorithm to find the HCF (Highest Common Factor) of the two numbers entered by a user. Transform your algorithm into a C program, support your program with suitable comments.**
**Ans.**

```
Algorithm to find HCF of two numbers:
Step 1: Read the two numbers from the user
Step 2: Set the smaller number as min and the larger number as max
Step 3: Set a variable named hcf to 1
Step 4: Loop from 1 to min a) If both min and max are divisible by the current loop variable
        i) Update hcf to the current loop variable
Step 5: Print the value of hcf as the HCF of the two numbers
```

**C program to find HCF of two numbers:**

```c
#include <stdio.h>

int main() {
  int num1, num2, min, max, hcf = 1;
  printf("Enter two numbers: ");
  scanf("%d %d", &num1, &num2);
  min = num1 < num2 ? num1 : num2;
  max = num1 > num2 ? num1 : num2;
  for (int i = 1; i <= min; i++) {
    if (min % i == 0 && max % i == 0) {
      hcf = i;
    }
  }
  printf("HCF of %d and %d is %d\n", num1, num2, hcf);
  return 0;
}
```

In this program, the user enters two numbers which are stored in the variables **num1** and **num2**. The **min** and **max** variables are set to the smaller and larger numbers respectively. The program then loops through all the numbers from 1 to **min** and checks if both **min** and **max** are divisible by the current loop variable. If so, the **hcf** variable is updated to the current loop variable. Finally, the program prints the value of **hcf** as the HCF of the two numbers.

**Question 5. Briefly discuss the relation between pointers and arrays, giving suitable example. Write a program in C, to print transpose of a 2D matrix entered by a user. Also give comments.**
**Ans.**

In C, arrays and pointers are closely related concepts. An array name can be used as a pointer to the first element of the array, and elements of the array can be accessed using pointer arithmetic. For example, if **arr** is an array of integers, **arr[0]** can be accessed as ***arr**, **arr[1]** can be accessed as ***(arr + 1)**, and so on.

**Here's an example program that demonstrates the relationship between pointers and arrays:**

```
#include <stdio.h>

int main() {
  int arr[] = {1, 2, 3, 4, 5};
  int *ptr = arr;
  for (int i = 0; i < 5; i++) {
    printf("%d ", *ptr);
    ptr++;
  }
  return 0;
}
```

In this program, the array **arr** is initialized with values 1 to 5. A pointer **ptr** is declared and initialized to the address of the first element of the array **arr**. The program then loops through the array using the pointer **ptr**, printing each element of the array using pointer dereferencing.

**Now, let's write a program in C to print the transpose of a 2D matrix entered by the user:**

```
#include <stdio.h>

#define ROWS 3
#define COLS 3

int main() {
  int matrix[ROWS][COLS], transpose[COLS][ROWS];
  printf("Enter the elements of the matrix:\n");
  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      scanf("%d", &matrix[i][j]);
    }
  }
  printf("Matrix entered:\n");
  for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
    printf("Transpose of matrix:\n");
    for (int i = 0; i < COLS; i++) {
        for (int j = 0; j < ROWS; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

In this program, we first define the size of the matrix as 3 rows and 3 columns using preprocessor directives. We declare two 2D arrays **matrix** and **transpose** to store the original matrix and its transpose respectively. The user enters the elements of the matrix using nested for loops, and the matrix is printed using another nested for loop.

The program then calculates the transpose of the matrix using two more nested for loops, and stores the result in the **transpose** array. Finally, the transpose of the matrix is printed using another nested for loop.

Note that to access the elements of the matrix using pointers, we can use a double pointer to point to the first element of the matrix, and use pointer arithmetic to access each element. However, this approach can be error-prone and difficult to read, so it is usually not recommended for simple programs like this.

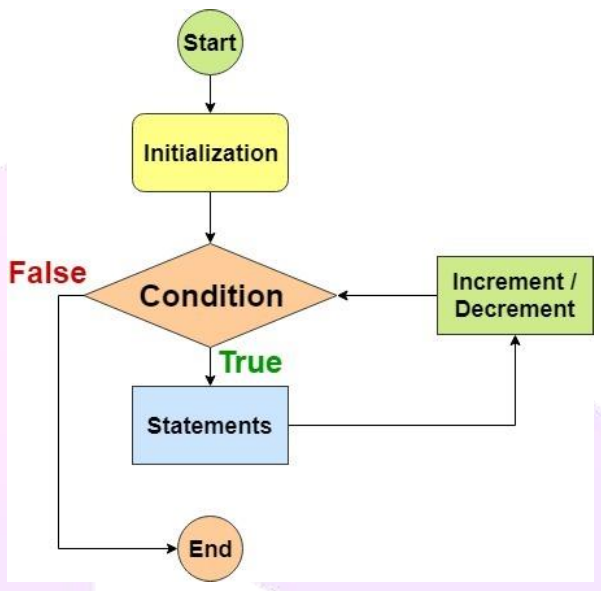
**Question 6. Write the syntax of looping control statements. Also draw the flowchart for each statement. Write a program in C to generate the following pattern :**

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

**Ans.**

```
#include <stdio.h>
int main() {
    int n = 3; // number of rows
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

**Flow Chart :-**



**Question 7. Differentiate between Random access and Sequential access of files in C. Discuss the syntax and role of fseek( ) and rewind( ) function, while accessing any file.**

**Ans:**

**C programming language, files can be accessed in two ways: random access and sequential access.**

**Random access means that you can read or write data from any position in a file, whereas sequential access means that you have to read or write data sequentially, one after the other.**

**In random access, the file can be read or written from any location, while in sequential access, the file can be read or written only from the beginning to the end.**

**Syntax of fseek() function:**

**int fseek(FILE *stream, long int offset, int whence);**
**This function sets the file position indicator for the stream pointed to by stream. The new position is calculated by adding offset bytes to the position specified by whence. The whence parameter can take one of the following three values:**

**SEEK_SET: The offset is relative to the beginning of the file.**
**SEEK_CUR: The offset is relative to the current position of the file pointer.**
**SEEK_END: The offset is relative to the end of the file.**
**Syntax of rewind() function:**

**void rewind(FILE *stream);**
**This function sets the file position indicator for the stream pointed to by stream to the beginning of the file.**

**Role of fseek() and rewind() functions:**

**fseek() function is used to move the file pointer to a specific location in the file.**
**rewind() function is used to move the file pointer to the beginning of the file.**

**Here's an example of how to use fseek() and rewind() functions:**

```
#include <stdio.h>

int main() {
   FILE *fp;
   char ch;

   fp = fopen("test.txt", "r");

   // move the file pointer to the 5th byte from the beginning of the file
   fseek(fp, 4, SEEK_SET);

   // read and print the contents of the file from the 5th byte
   while ((ch = getc(fp)) != EOF) {
      putchar(ch);
   }

   // move the file pointer back to the beginning of the file
   rewind(fp);

   // read and print the contents of the file from the beginning
   while ((ch = getc(fp)) != EOF) {
      putchar(ch);
   }

   fclose(fp);
   return 0;
}
```

**In this example, fseek() function is used to move the file pointer to the 5th byte from the beginning of the file, and then rewind() function is used to move the file pointer back to the beginning of the file. Finally, the contents of the file are read and printed twice, once from the 5th byte and then from the beginning.**

**Question 8. Compare any two of the following (give suitable C code for each) :**
**a. Break and Continue Statement**
**b. Structure and Union**
**Ans:---**

**A. Break and Continue Statement:**

**break and continue statements are used in loops to control the flow of execution.**

**The break statement is used to terminate the loop, whereas the continue statement is used to skip the current iteration and move to the next iteration.**

**Here's an example of using the break statement:**

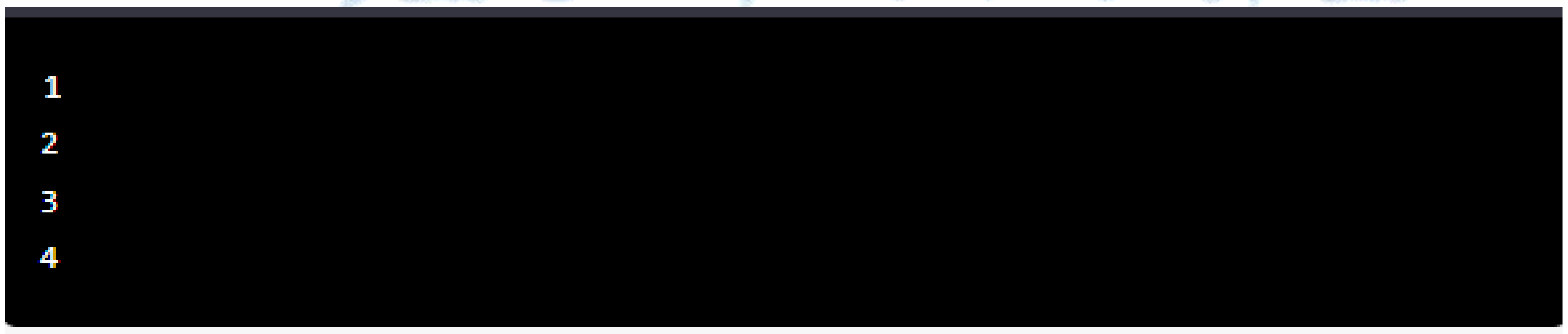
```
#include <stdio.h>

int main() {
   int i;

   for (i = 1; i <= 10; i++) {
      if (i == 5) {
         break;
      }
      printf("%d\n", i);
   }

   return 0;
}
```

**Output**


```
1
2
3
4
```

**Here's an example of using the continue statement:**

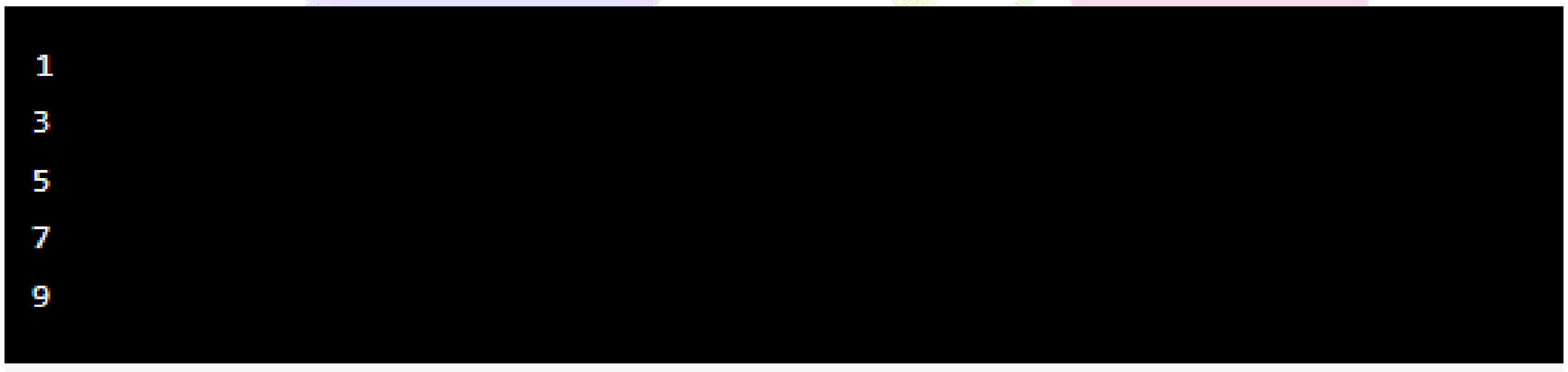
```
#include <stdio.h>

int main() {
   int i;

   for (i = 1; i <= 10; i++) {
      if (i % 2 == 0) {
         continue;
      }
      printf("%d\n", i);
   }

   return 0;
}
```

**Output**



```
1
3
5
7
9
```

## B. Structure and Union:

**Both structures and unions are used to store different data types in a single variable.**

**A structure is a user-defined data type that stores a collection of data types under a single name. Each data member in a structure has its own memory location.**

**Here's an example of a structure:**

```
#include <stdio.h>
```

```
struct student {
   char name[20];
   int age;
   float marks;
};

int main() {
   struct student s1 = {"John", 20, 75.5};

   printf("Name: %s\n", s1.name);
   printf("Age: %d\n", s1.age);
   printf("Marks: %.2f\n", s1.marks);

   return 0;
}
```

**Output:**

**makefile**
**Copy code**

```
Name: John
Age: 20
Marks: 75.50
```

**A union is also a user-defined data type that stores a collection of data types under a single name. However, in a union, all the data members share the same memory location, and only one data member can be accessed at a time.**

**Here's an example of a union:**

```
#include <stdio.h>

union data {
   int i;
   float f;
   char c;
};

int main() {
   union data d1;

   d1.i = 10;
   printf("d1.i = %d\n", d1.i);
   d1.f = 3.14;
   printf("d1.f = %.2f\n", d1.f);

   d1.c = 'A';
   printf("d1.c = %c\n", d1.c);

   return 0;
}
```

**Output:**

```
d1.i = 10
d1.f = 3.14
d1.c = A
```

**In this example, the union data has three data members i, f, and c, but only one of them can be accessed at a time. When the value of one data member is assigned, the values of the other data members become undefined.**

**To summarize, structures and unions are used to store different data types in a single variable, but structures store each data member in its own memory location, whereas unions store all the data members in the same memory location.**

## SECTION-B (PYTHON-Programming)

**Question 9. What is C-Python ? Briefly discuss the relation between framework, library, package and module in Python.**

**Ans:-**

**C-Python is the standard implementation of the Python programming language. It is written in the C language and provides a comprehensive set of libraries and tools for developing Python applications. C-Python includes an interpreter that executes Python code, as well as a standard library that provides a wide range of modules for various tasks such as file I/O, networking, and database access.**

**In Python, there are several terms that are used to refer to collections of code that can be reused in different programs:**

1. **A module is a single file that contains Python code. It usually defines a set of functions, classes, and variables that can be imported and used in other Python programs.**

2. **A package is a collection of modules that are organized into a directory hierarchy. Packages provide a way to group related functionality together and make it easier to manage and distribute code.**

3. A library is a collection of packages and modules that are designed to be used together. Libraries provide a set of high-level abstractions that simplify complex tasks and make it easier to write Python code.

4. A framework is a set of libraries and tools that provide a structure for building applications. Frameworks provide a way to organize code and enforce design patterns that promote good coding practices.

In Python, packages and modules are used to organize code and provide reusable functionality, while libraries and frameworks provide higher-level abstractions that simplify complex tasks and promote good coding practices.

**Question 10. Differentiate between mutable and immutable data types in Python. Briefly discuss the following data types of Python : a. Lists b. Tuples c. Dictionary**

**Ans:-**

In Python, data types can be classified into two categories: mutable and immutable.

Mutable data types are those that can be changed after they are created, while immutable data types cannot be changed once they are created. Examples of mutable data types in Python include lists, dictionaries, and sets. Examples of immutable data types in Python include strings, numbers, and tuples.

a. Lists:

A list is a mutable data type in Python that is used to store a collection of values. Lists are declared using square brackets and can contain any number of elements, separated by commas. Elements in a list can be of any data type, including other lists.

Example:

```
my_list = [1, 2, 3, 'hello', [4, 5, 6]]
```

b. Tuples:

A tuple is an immutable data type in Python that is used to store a collection of values. Tuples are declared using parentheses and can contain any number of elements, separated by commas. Elements in a tuple can be of any data type, including other tuples.

Example:

```
my_tuple = (1, 2, 3, 'hello', (4, 5, 6))
```

c. Dictionary:

A dictionary is a mutable data type in Python that is used to store a collection of key-value pairs. Dictionaries are declared using curly braces and each key-value pair is separated by a colon. Keys in a dictionary must be unique and can be of any immutable data type, while values can be of any data type.

Example:

**my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}**

**In summary, understanding the difference between mutable and immutable data types in Python is important to prevent unintended changes to variables. Lists and dictionaries are mutable data types, while tuples are immutable data types. Each data type has its own unique properties and use cases that make them useful in different programming scenarios.**

**Question 11. What is the utility of map( ) function do ? Write a program in Python to print the square of the numbers present in the list, by using map( ) function.**

**Ans:--**

**The map() function in Python is a built-in function that is used to apply a given function to each element of an iterable (list, tuple, dictionary, etc.) and return a new iterable with the results.**

**Example program to print the square of numbers present in a list using map() function:**

```
my_list = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x**2, my_list))
print(squares)
Output:

[1, 4, 9, 16, 25]
```

**Question 12. Compare overloading and overriding in Python. Give suitable example code for each in Python.**

**Ans:-**

**Overloading and overriding are two important concepts in object-oriented programming. Overloading is the ability to define multiple methods with the same name but different parameters in a class. On the other hand, overriding is the ability to provide a new implementation for a method that is already defined in a superclass.**

**Example code for overloading in Python:**

```
class MyClass:
    def add(self, a, b):
        return a + b

    def add(self, a, b, c):
        return a + b + c

obj = MyClass()
print(obj.add(1, 2)) # Output: TypeError: add() missing 1 required positional argument: 'c'
print(obj.add(1, 2, 3)) # Output: 6
```

**Example code for overriding in Python:**

```
class Parent:
    def say_hello(self):
        print("Hello from parent")

class Child(Parent):
    def say_hello(self):
        print("Hello from child")

obj = Child()
obj.say_hello() # Output: Hello from child
```

**Question 13. Write Python code to perform the following : a. Reading data from a file b. Creating a file and add content to it.**

**Ans:-**

**a. Reading data from a file:**

```
# Open file in read mode
file = open('file.txt', 'r')

# Read the entire file
contents = file.read()

# Print the file contents
print(contents)

# Close the file
file.close()
```

**b. Creating a file and add content to it:**

```
# Open file in write mode
file = open('file.txt', 'w')

# Write data to file
file.write('Hello World\n')

# Close the file
file.close()
```

**Question 14. What are Lambda functions ? How do Lambda functions differ from Built-in functions ? Write lambda function to calculate cube of a number. Also write the program to find cube of a number without using lambda function.**

**Ans:--**

Lambda functions are anonymous functions in Python that can be defined in a single line of code. They are used to create small, one-time-use functions that don't need to be named or defined elsewhere in your code. Lambda functions differ from built-in functions in that they are not predefined and are created on the fly.

Example code for a lambda function to calculate the cube of a number:

```
cube = lambda x: x**3
print(cube(2)) # Output: 8
```

Example code to find the cube of a number without using lambda function:

```
def cube(x):
    return x**3

print(cube(2)) # Output: 8
```

**Question 15. Differentiate between the following with the help of suitable example for each : a. Co-routines and subroutines b. Co-routines and threads**

**Ans:--**

**a. Coroutines and Subroutines:**

A subroutine is a callable unit of code that is executed and then returns control back to the caller. In other words, a subroutine is a function that is called from another function and returns to the calling function when its work is done. A coroutine, on the other hand, is a function that can be paused and resumed later, allowing it to perform its work incrementally over time.

Example of a subroutine:

python
Copy code

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

result = add(3, 5)
print(result) # Output: 8
```

Example of a coroutine:

python

```
Copy code
def my_coroutine():
    while True:
        x = yield
        print(x)

cor = my_coroutine()
next(cor)
cor.send("Hello") # Output: Hello
cor.send("World") # Output: World
```

**b. Coroutines and Threads:**

**Both coroutines and threads are used for concurrent programming, but they work differently. A coroutine is a single thread of execution that can be paused and resumed at specific points in the code, while a thread is a separate independent execution path that runs in parallel with other threads.**

**Example of a coroutine:**

```
async def my_coroutine():
    while True:
        print("Hello")
        await asyncio.sleep(1)

loop = asyncio.get_event_loop()
loop.run_until_complete(my_coroutine())
```

**Example of a thread:**

```
import threading
import time

def my_thread():
    while True:
        print("Hello")
        time.sleep(1)

t = threading.Thread(target=my_thread)
t.start()
```

**Question 16. What are Cursor Objects ? Briefly discuss the utility of cursor objects. Write Python code for a cursor to execute the SQL query, to print the version of database. Support your program with suitable comments.**

**Ans:--**

**A cursor object is an instance of the cursor class in Python that is used to interact with a database. It is used to execute SQL queries and manage transactions in a database. Cursor objects are used to execute SQL commands and retrieve results from a database.**

**Example code to execute an SQL query and print the version of the database:**

```
import sqlite3

# Connect to database
conn = sqlite3.connect('my_database.db')

# Create cursor object
cursor = conn.cursor()

# Execute SQL query
cursor.execute("SELECT SQLITE_VERSION()")

# Fetch results from cursor
data = cursor.fetchone()

# Print results
print("SQLite version: ", data[0])

# Close cursor and database connection
cursor.close()
conn.close()
```