Functions

Date: .....
Page: .....

→ **Functions** - A set of statements (set of block of code) that takes input, does some specific task & gives output.

→ **Why do we need functions?**

i) Reducing code redundancy.
ii) To make code modular.
iii) for code reusability
iv) for code optimization.
v) To avoid length & bulky code.

→ **Syntax with-out Parameters**

```
return-type function-name () {
    // Body
    return statement;
}
```

→ **Syntax of function with parameters.**

```
return-type function-name (Parameter 1, Parameter 2, ----) {
    // Body
    return statement;
}
```

Eg:-

```
void printLine () {    // function without parameter
    for (int i = 0; i < 10; i++) {
        cout << "Suraj" << endl;
    }
}
int main () {
    printLine ();    // function call.
}
```

Date: .....
Page: .....

Eg:-
```
int add (int a, int b) {    // function with Parameters.
    int sum = a+b;
    return sum;
}
int main() {
    int sum = add (5, 3);
    cout << "Sum is :" << sum;
}
```

→ **Function Declaration.**

A function declaration tells the compiler about the no. of parameters, data type of parameters & return type of function.

Syntax:

```
return-type function-name ();    // without Parameter
return-type function-name (parameter list);    // with Parameter
```

Eg:-
```
void printMessage ();
int findMax (int num1, int num2);
```

No logic is defined during function declaration.

→ **Function Definition.**

A function definition is a complete implementation of a function that describes its behavior & how it's executed. It includes the function name, return type, parameter list & the body of the function.

Syntax:

```
Return_Type Function_Name (Parameter_List) {
    // Function Body
}
```

Date: .....
Page: .....

Eg:-
```
int sum (int a, int b) {
    return a+b;
}
```

→ Function call.

When you define a function, you tell the function what to do & to use that function, you have to call or invoke the function.

Syntax:

```
Function_Name (Parameters);
```

Eg:-
```
void greet () {
    cout << "Hello!";
}
int main () {
    greet ();
}
```

Function call

→ Function Parameters.

Function parameters are input variables that are passed to function so they can execute specific tasks. Parameters act as variable inside the function. & their values are used to manipulate data or perform calculations.

Function parameters are defined in the function declaration & definition & are specified within the parentheses following the function name.

Syntax:

```
Return Type   function Name ( Parameter Type   Parameter Name ) {
    // function Body
}
```

Date: .....
Page: .....

Eg!-
```
int add (int a, int b) {
    return a+b;
}
```
↳ Function Parameters with data types integer.

→ Those parameters which is use while function is define are called formal or normal parameters.

Eg!- int sum (int a, int b)

→ Here, int a & int b are formal or normal parameters.

→ Those parameters which is use while function is called in main function are called actual parameter or arguments.

Eg!- sum (a, b);

→ Here, a & b are actual parameter or arguments.

NOTE: When you are working with multiple parameters, the function call must have the same no. of arguments as there are parameters & the arguments must be passed in the same order.

Eg!-
```
int multiply (int a, int b) {
    return a*b;
}

int main () {
    int a = 5;
    int b = 4;
    int product = multiply (a, b);
    cout << product ;
    return 0;
}
```
↳ Formal Parameters

↳ Actual Parameters.

Date: .....
Page: .....

Some examples of functions.

1)

```
#include <iostream>
using namespace std;
// Function Declaration
void printLine();
int main() {  // Main Function
    // Function call
    printLine();
    return 0;
}
// Function Defination
void printLine() {  // void function which doesn't return anything
    for (int i = 0; i < 10; i++) {
        cout << " Suraj " << endl;
    }
}
```

2) Function that print sum of 3 numbers.

```
void printSum (int a, int b, int c) {  // Function defination.
    int sum = a+b+c;
    cout << sum;
}
int main () {
    printSum (5, 10, 15);  // Function call
    return 0;
}
```

NOTE: Whenever, we define the function above the main function like above eg. in this case function declaration is not required. But when we define the function below the main function like 1) eg.

Date: .....
Page: .....

in this case function declaration is must, otherwise it will give error.

3) Function To find max of 3 numbers.

```
void printMax (int a, int b, int c) {
    if ( a>=b && a>=c )
        cout << " max is :" << a ;
    elseif ( b>=a && b>=c )
        cout <<" max is :" << b;
    else
        cout <<" max is :" c ;
}
int main () {
    printMax ( 10. 15, 50) ;  // function call
    return 0;
}
```

→ C++ Inbuid function to find max number.

Syntax : max (First Num, Second Num) ;

```
void findMax (int a, int b, int c) {
    int max1 = max (a, b);
    int max2 = max (max1, c);
    cout << "max is :" << max2 ;
}
int main () {
    findMax ( 10,20, 30) ;
    return 0;
}
```

↑↑

Non-Return Type Function

```
int printMax (int a, int b, int c) {
    int ans = max (a, b);
    int maxi = max (ans, c);
    return maxi ;
}
int main () {
    int max = printMax (5,10.20);
    cout << "max is :" << max;
    return 0;
}
```

Return Type Function

Date: .....
Page: .....

4) Function that calculate sum of all even numbers from 1 → N.

```
void sum (int n) {
    int sum = 0;
    for (int i = 0; i <= n; i++) {
        if (i % 2 == 0)
            sum += i;
    }
    cout << sum;
}

int main () {
    // Function call
    sum (20);
    return;
}
```

5) Function that check given number is prime or not.

```
bool checkPrime (int n) {
    for (int i = 2; i < n; i++) {
        if (n % i == 0) // Perfectly divisible, means not prime
            return false;
    }
    // If 'n' isn't perfectly divisible from [2 → n-1], means Prime.
    return false;
}

int main () {
    bool prime = checkPrime (15);
    if (prime)
        cout << "Prime Number";
    else
        cout << "Not Prime Number";
    return 0;
}
```

Date: .....

Page: .....

Bonus Tips :

1) If we say, for (int i = 0 ; i → < n)

That means, i = 0, 1, 2, 3, ---, n-1.

2) If we say, for (int i = 0 ; i → <= n)

That means, i = 0, 1, 2, 3, ----, n.

3) If we say, for (int i = 1, ; i → <= n)

That means, i = 1, 2, 3, ---, n.

4) If we say, for (int i = 1, ; i → < n)

That means, i = 1, 2, 3, ---, n-1.