



# Functions



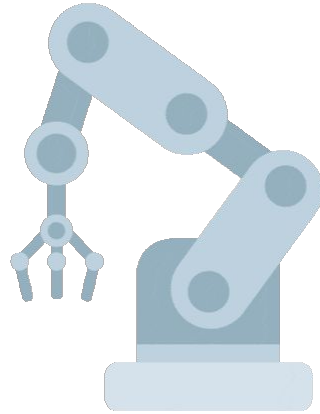
# Repeated Tasks

Sometimes in **real world** there are tasks that we have to **repeat many times**.

# Repeated Tasks

For example, we have a robot and we want it to **pick a glass**. In order to do that we give the instructions in following way.

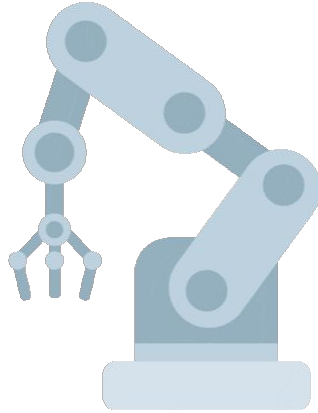
1. Move your arm at 90 degree.
2. Move your hand toward left side.
3. Open fingers.
4. Grip the Glass.
5. Put the Glass on the Table.
6. Once Put it on table give a beep.



# Repeated Tasks

let say, we have to give such kind of instructions many time to robot only the **angle** and **direction** of hand changes

1. Move your arm at **90** degree.
2. Move your hand toward **left** side.
3. Open fingers.
4. Grip the Glass.
5. Put the Glass on the Table.
6. Once Put it on table give a beep.



# Repeated Tasks

In real life, we encounter many such problems when there are **same kind of tasks** but some **parameters changes**.

# Repeated Tasks

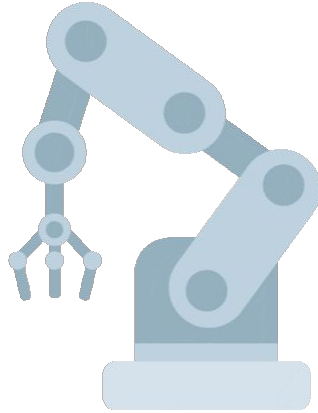
- For such problems, we use functions.
- Function is **set of instructions** with some specific **name**, and when we need to run those instructions we only use that name.
- A Function can input some **parameters** that it may use to perform some task.
- A Function can **return** some value after its completion.

# Repeated Tasks

For example, we can make following instructions as function by assigning some **name** with two **parameters** (angle and direction) and it shall **return** if robot successfully completed its task.

**bool pickGlass (angle , direction)**

1. Move your arm at **angle** degree.
2. Move your hand toward **direction** side.
3. Open fingers.
4. Grip the Glass.
5. Put the Glass on the Table.
6. Once Put it on table give a beep.
7. If put on the table **Return true** else **Return false**



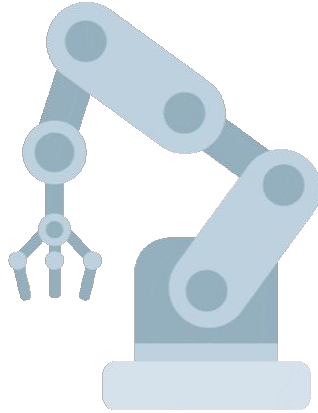
# Repeated Tasks

Whenever we need to ask the robot to pick any glass, we **call** the function with the name and required parameters.

**pickGlass(90,left)**

**bool pickGlass (angle , direction)**

1. Move your arm at **angle** degree.
2. Move your hand toward **left** side.
3. Open fingers.
4. Grip the Glass.
5. Put the Glass on the Table.
6. Once Put it on table give a beep.
7. If put on the table **Return true** else **Return false**





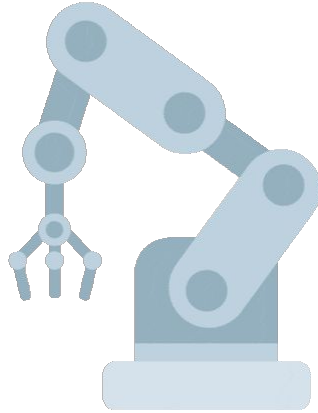
# Repeated Tasks

If we want to know what was the result then we can save that into some memory/variable

```
bool result = pickGlass(90,left)
```

**bool pickGlass (angle , direction)**

1. Move your arm at **angle** degree.
2. Move your hand toward **left** side.
3. Open fingers.
4. Grip the Glass.
5. Put the Glass on the Table.
6. Once Put it on table give a beep.
7. If put on the table **Return true** else **Return false**



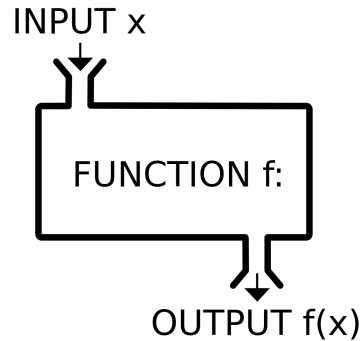
# What is a Function?

A **function** is a block of code that performs a specific task.

# Function: Definition

A **function** is a block of code that performs a specific task.

We give **inputs** to the function, it performs some calculations on it, and returns the **output**.



# Types of Functions

In C++, we have

1. User-Defined Functions
2. Pre-Defined (Library) Functions

# Pre-Defined Functions

- We can use **library functions** by invoking the functions directly; we don't need to write the functions ourselves.
- In order to use **library functions**, we usually need to include the **header file** in which these library functions are defined.

# Pre-Defined Functions

- In C++, pre-defined functions are organized into separate libraries.
- For example, the header file `iostream` contains I/O functions; such as `cout` and `cin` functions.

1	<code>#include &lt;iostream&gt;</code>
---	--

# Pre-Defined Functions

- Similarly, the header file `cmath` contains math functions; such as `pow`, `sqrt`, `fabs` and `floor` etc.

```
1  #include <iostream>
2  #include <cmath>
```

# Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code>&lt;cmath&gt;</code>	Returns $x^y$ ; if $x$ is negative, $y$ must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double



# Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code>&lt;cmath&gt;</code>	Returns $x^y$ ; if $x$ is negative, $y$ must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double
<code>sqrt(x)</code>	<code>&lt;cmath&gt;</code>	Returns the nonnegative square root of $x$ ; $x$ must be nonnegative <code>sqrt(4.0) = 2.0</code>	double	double

# Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code>&lt;cmath&gt;</code>	Returns $x^y$ ; if $x$ is negative, $y$ must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double
<code>sqrt(x)</code>	<code>&lt;cmath&gt;</code>	Returns the nonnegative square root of $x$ ; $x$ must be nonnegative <code>sqrt(4.0) = 2.0</code>	double	double
<code>fabs(x)</code>	<code>&lt;cmath&gt;</code>	Returns the absolute value of its argument <code>fabs(-5.67) = 5.67</code>	double	double

# Working Example

Write a **C++** program that calculates the power using the **pre-defined** function.



# Working Example: Output

```
C:\C++>c++ example.cpp -o example.exe
```

```
C:\C++>example.exe
```

```
Enter Value: 2
```

```
Enter Power: 4
```

```
Answer is: 16
```



# Solution

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  main(){
6      double number1, number2, result;
7      cout << "Enter Value: ";
8      cin >> number1;
9      cout << "Enter Power: ";
10     cin >> number2;
11     result = pow(number1, number2);
12     cout << "Answer is: " << result;
13 }
```

# || User Defined Functions

Sometimes, we as a developer, write the code that may be **reused** in the future; therefore, we need to **create our own functions**. These functions are called **user defined functions**.

# Review: Working Example

Before creating a user defined function, let first Write a **C++** program that **add two numbers**.



# Solution

```
1  #include <iostream>
2  using namespace std;
3  main(){
4      int number1, number2, sum;
5      cout << "Enter First Number: ";
6      cin >> number1;
7      cout << "Enter Second Number: ";
8      cin >> number2;
9      sum = number1 + number2;
10     cout << "Sum is: " << sum;
11 }
```



# Working Example: Output

```
C:\C++>c++ example.cpp -o example.exe
```

```
C:\C++>example.exe
```

```
Enter First Number: 5
```

```
Enter Second Number: 4
```

```
Sum is: 9
```



# Any Problem in the Solution?

Do you see any problem in the solution?

```
1  #include <iostream>
2  using namespace std;
3  main() {
4      int number1, number2, sum;
5      cout << "Enter First Number: ";
6      cin >> number1;
7      cout << "Enter Second Number: ";
8      cin >> number2;
9      sum = number1 + number2;
10     cout << "Sum is: " << sum;
11 }
```

# Any Problem in the Solution?

What if i ask you to make the following changes and instead of adding the two numbers now i want you to multiply those two numbers?

# Addition

What will you do?

```
1  #include <iostream>
2  using namespace std;
3  main() {
4      int number1, number2, sum;
5      cout << "Enter First Number: ";
6      cin >> number1;
7      cout << "Enter Second Number: ";
8      cin >> number2;
9      sum = number1 + number2;
10     cout << "Sum is: " << sum;
11 }
```

# Multiplication

Make a **separate program** with the following changes.

```
1  #include <iostream>
2  using namespace std;
3  main() {
4      int number1, number2, mul;
5      cout << "Enter First Number: ";
6      cin >> number1;
7      cout << "Enter Second Number: ";
8      cin >> number2;
9      mul = number1 * number2;
10     cout << "Multiplication is: " << mul;
11 }
```

# Any Problem in the Solution?

What if i again want to add the numbers instead of multiplication?

Will we keep on changing the program?

## || Better Solution

What if we write the code of both the addition and multiplication in the same program?

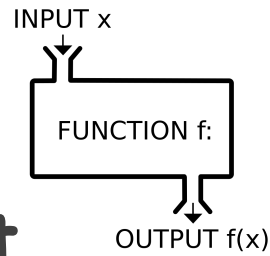
## || Better Solution

What if we write the code of both the addition and multiplication in the same program?

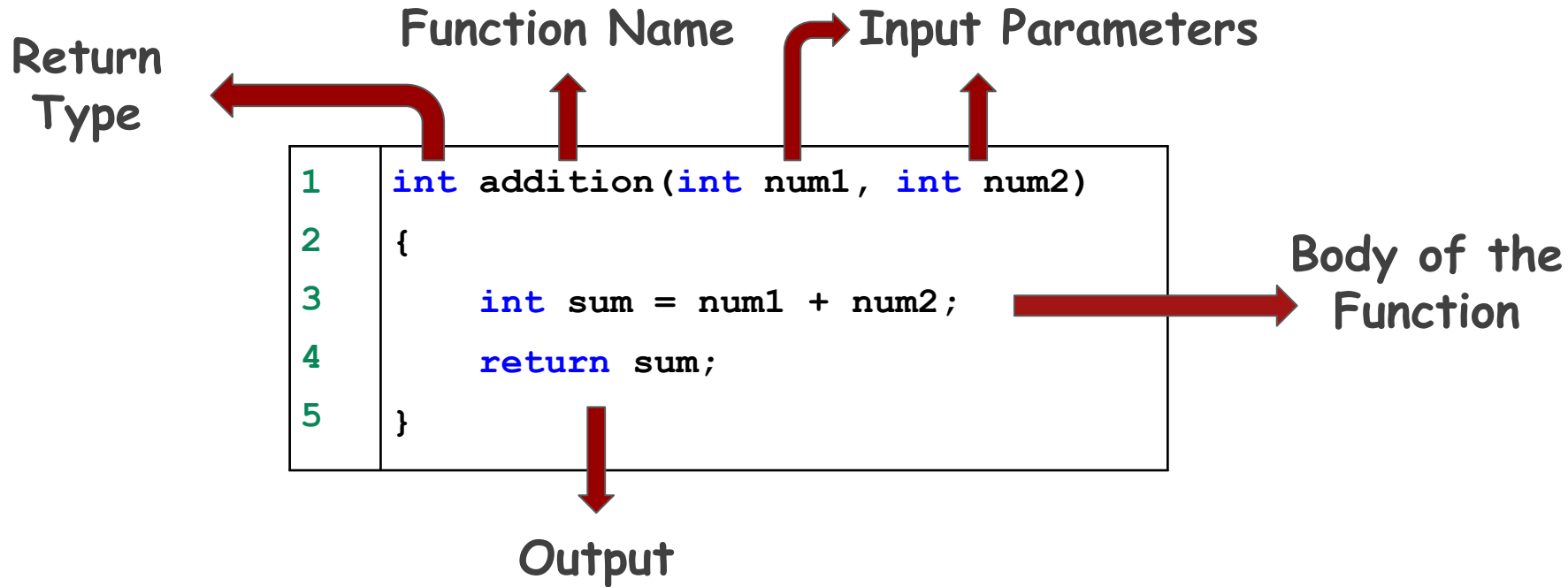
We can do that with the help of functions.



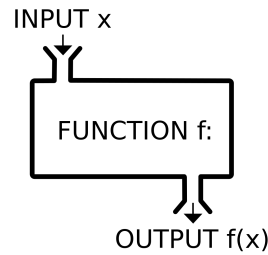
# Function in C++



Let's write the function of **Addition** first.



# Function in C++



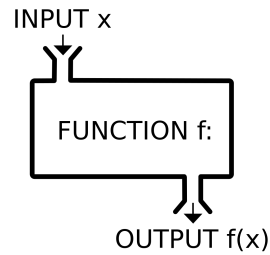
Now, Let's see how to use the function of **Addition**.

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Function Call

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

# Function in C++



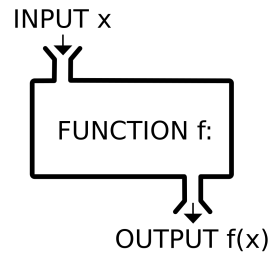
Now, Let's see how to use the function of **Addition**.

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Parameter  
Passing

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

# Function in C++



Now, Let's see how to use the function of **Addition**.

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Function  
returning the  
output

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

# Functions in C++

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

```
1 int multiplication(int num1, int num2)
2 {
3     int mul = num1 * num2;
4     return mul;
5 }
```

```
1 main(){
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     //result = addition(number1, number2);
8     result = multiplication(number1, number2);
9 }
```

# Functions in C++

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

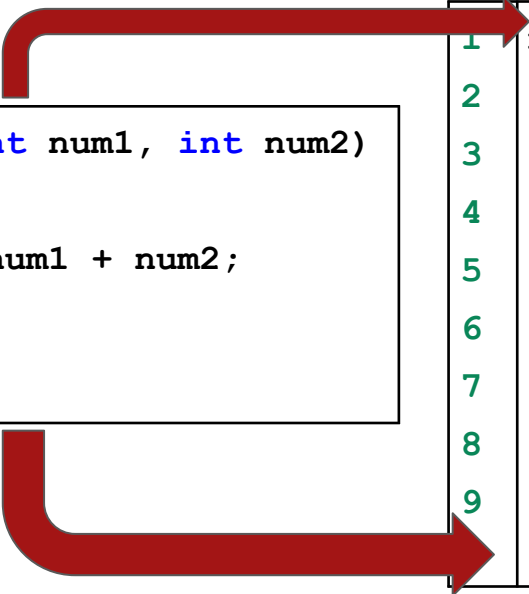
```
1 int multiplication(int num1, int num2)
2 {
3     int mul = num1 * num2;
4     return mul;
5 }
```

```
1 float division(float num1, float num2)
2 {
3     float div = num1/num2;
4     return div;
5 }
```

# Functions in C++

Now, the question is **where to write** these functions?

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```




```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

The diagram illustrates the flow of control between two C++ code blocks. A red arrow originates from the `addition` function call on line 7 of the `main()` function and points to the `addition` function definition. A second red arrow originates from the closing brace of the `addition` function and points back to the `main()` function, indicating the return of control to the caller.

# Functions in C++

In C++, the code of function declaration should be **before** the function call.

```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```



```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```



# Functions in C++

In C++, the code of function declaration should be **before** the function call.

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2)
5  {
6      int sum = num1 + num2;
7      return sum;
8  }
9
10 main() {
11     float number1, number2, result;
12     cout << "Enter First Number: ";
13     cin >> number1;
14     cout << "Enter Second Number: ";
15     cin >> number2;
16     result = addition(number1, number2);
17     cout << "Sum is: " << result;
18 }
```

# Functions in C++

However, if we want to define a function **after** the function call, we need to use the **function prototype**.

## Function Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int, int);
5
6  main() {
7      float number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```

# Functions in C++

However, if we want to define a function **after** the function call, we need to use the **function prototype**.

Function  
Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2);
5
6  main() {
7      float number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```

# Functions in C++

However, if we want to define a function **after** the function call, we need to use the **function prototype**.

```
1 #include <iostream>
2 using namespace std;
3
4 int addition(int num1, int num2);
5
6 main() {
7     float number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    result = addition(number1, number2);
13    cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```

Function  
Prototype

Function  
Definition

# Conclusion

The syntax of a  
function  
prototype

```
returnType functionName(dataType parameters);
```

# Conclusion

The syntax of a  
function

prototype,

function

definition

```
returnType functionName(dataType parameters);
```

```
returnType functionName(dataType parameters)
{
    statements;
    return output;
}
```

# Conclusion

The syntax of a  
function  
**prototype**,  
function  
**definition**, and  
function **calling** is

```
returnType functionName(dataType parameters);
```

```
returnType functionName(dataType parameters)
{
    statements;
    return output;
}
```

```
main()
{
    receivingVariable = functionName(parameters)
}
```

## || Conclusion: Functions

These functions are **written** by the user himself, therefore, these are called **User-Defined Functions**.



## Conclusion: Benefits

1. Functions make the code **reusable**. We can declare them once and use them multiple times.
2. Functions make the **program easier** as each small task is divided into a function.
3. Functions increase **readability**.

# Learning Outcome

In this lecture, we learnt how to write a **C++** Program that solves a problem using **User-defined Functions** and **Pre-defined Functions**.



# Self Assessment

1. The **Euclidean distance** is the straight line distance between two points in Euclidean space. Formula to calculate the Euclidean distance between 2 points is given

$$d(p, q) = \sqrt{(p - q)^2}.$$

Write a **user defined function** to calculate the Euclidean distance between 2 points. In that function use the **built-in functions** of **pow** and **sqrt** to compute the results.



# Solution

```
#include <iostream>
#include <cmath> // for using absolute maths function
using namespace std;

double d(int p, int q);

main(){
    int a, b;
    double c;
    cout << "Enter First Number: ";
    cin >> a;
    cout << "Enter Second Number: ";
    cin >> b;
    c = d(a, b);
    cout << "Euclidean Distance is: " << c;
}

/* function Definition */
double d(int p, int q)
{
    int diff = p - q;
    double sq = pow(diff,2);
    double result = sqrt(sq);
    return result;
}
```

# Self Assessment

The digit distance between two numbers is the total value of the difference between each pair of digits.

To illustrate:

`digitDistance(234, 489) → 12`

// Since  $|2 - 4| + |3 - 8| + |4 - 9| = 2 + 5 + 5$

Create a function that returns the digit distance between two integers.

Note

- Both integers will be exactly of length 3.

Input	Output
<code>digitDistance(121, 599)</code>	19
<code>digitDistance(234, 489)</code>	12
<code>digitDistance(200, 100)</code>	1

# Solution

```
#include <iostream>
#include <cmath> // for using absolute abs() function
using namespace std;

int digitDistance(int num1, int num2);

main(){
    int a, b, c;
    cout << "Enter First Digit: ";
    cin >> a;
    cout << "Enter Second Digit: ";
    cin >> b;
    c = digitDistance(a, b);
    cout << "Digit Distance is: " << c;
}

int digitDistance(int num1, int num2) /* function Definition */
{
    int unit1 = num1 % 10;
    int ten1 = (num1 / 10) % 10;
    int hun1 = (num1 / 100) % 10;
    int unit2 = num2 % 10;
    int ten2 = (num2 / 10) % 10;
    int hun2 = (num2 / 100) % 10;
    int result = abs(unit1 - unit2) + abs(ten1 - ten2) + abs(hun1 - hun2);
    return result;
}
```

# Self Assessment

Create a function that determines if the temperature of the water is considered boiling or not. Temperature will be measured in fahrenheit or celsius.

## Note

The boiling point of water is 212F in Fahrenheit and 100C in celsius.

## Test Cases

Input	Output
isBoiling(212, 'F')	1
isBoiling(100, 'C')	1
isBoiling(0, 'F')	0

