## Table of Contents



# Introduction

Using **relational operators** and **logic operators**, we can build **boolean expressions**, which allow us to ask questions within our program – is the balance greater than 0? is the age less than 21? is the user already registered?

These expressions can be used in if statements and while loops in order to execute portions of code if the statement is **true**.

# Boolean Expressions

A boolean expression is an expression that results in either **true** or **false**. For example:

- Is x greater than y?
- Is grade less than 70?
- Are width and length equal, and is length and height equal?

To write these statements in code, we use the **relational operators**. To ask more than one question at a time, we combine boolean expressions with **logic operators**.

| Boolean Expression | Representation |
|---|---|
| `x == y`<br>is x and y equal? |  |
| `x != y`<br>is x and y not equal? |  |
| `x <= y`<br>is x less than or equal to y? |  |
| `x < y`<br>is x less than to y? |  |
| `x < 0 \|\| x > 10`<br>True: x is less than 0, or greater than 10, exclusive.<br>False: x is between 0 and 10, inclusive. |  <br>An OR is the appropriate way to represent this. AND is incorrect, as x cannot be both less than 0 AND greater than 10. |
| `x >= 0 && x <= 10`<br>True: x is between 0 and 10, inclusive.<br>False: x is less than 0, or greater than 10, exclusive |  |

# If Statements

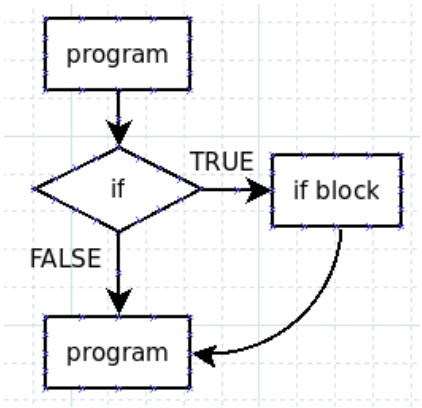
There are four ways to write if statements:

**Just one if statement**

A lone if-statement will check for a condition.

- **True:** Execute the internal code.
- **False:** Skip over the code-block.

```
if ( w == h )
{
    type = "square";
}
```

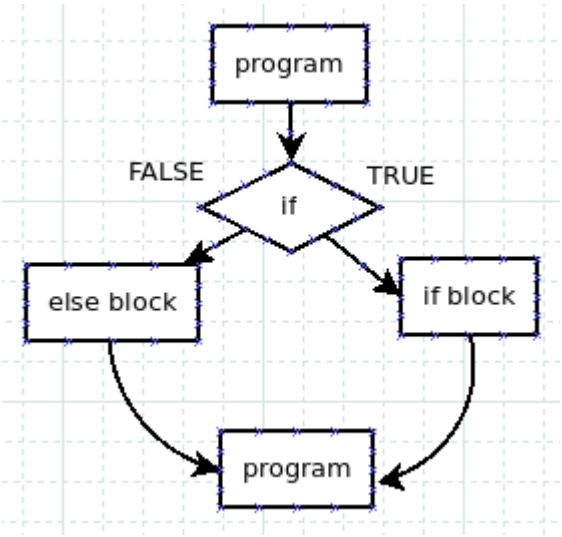



**An if and an else**

An if-else statement will have two code-blocks, and one of them will always be executed.

- **True:** Execute what is in the “if” block.
- **False:** Execute what is in the “else” block.

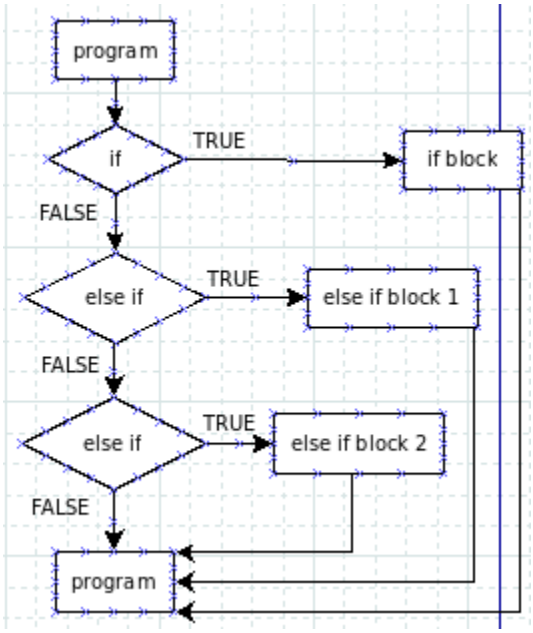
```
if ( w == h )
{
    type = "square";
}
else
{
    type = "rectangle";
}
```



## An if and one or more else if

Multiple conditions can be tested, but if none of them are **true**, then all internal code is skipped.

```
if ( age > 18 )
{
    canVote = true;
    canDrink = false;
}
else if ( age > 21 )
{
    canVote = true;
    canDrink = true;
}
```




## An if, one or more else if, and an else.

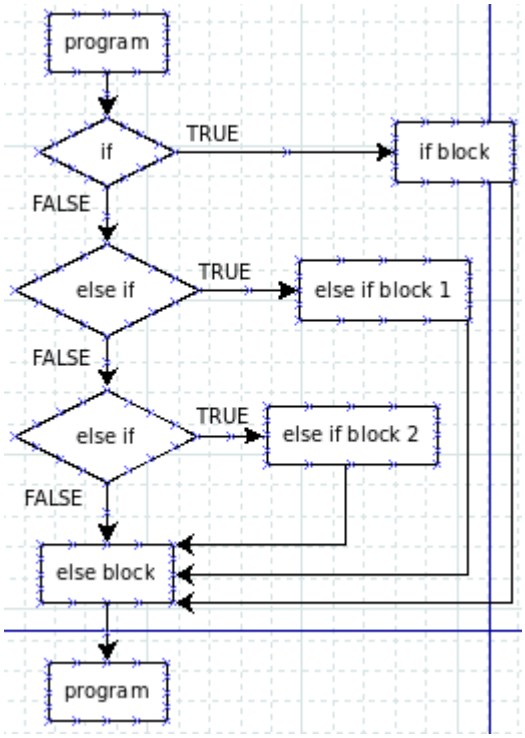
Multiple conditions can be tested, but if none of them are **true**, then the code within **else** is executed.

```
if ( age < 13 )
    price = kidPrice;

else if ( age < 20 )
    price = teenPrice;

else if ( age > 60 )
    price = seniorPrice;

else
    price = normalPrice;
```



## While Loops

While loops are similar to an if statement (without the **else if** or **else**), in that it checks a condition and executes its internal code if that condition is true.

However, with while loops, as long as that condition stays true, the loop will continue running the same code over and over. This can be useful for doing repetitive tasks.

```
while ( countdown > 0 )
{
    System.out.println( countdown );
    countdown--;
}
```

C++ output:
`cout << countdown;`

C# output:
`Console.WriteLine( countdown );`

You can use a boolean variable to make sure your loop continues running, and then make sure to set that variable to **false** within the while loop in order to trigger it to leave the loop.

```
boolean done = false;
while ( done == false )
{
    runProgram();

    if ( command.equals( "quit" ) )
    {
        done = true;
    }
}
```

C++ and C# if statement:
`if ( command == "quit" )`

Java can't do == to compare strings, so here it is using the .equals method to check if **command** is equal to **“quit”.**

You can also force the loop to run forever by saying **while true**, because true will never be false. Then, when you want to leave the while loop, you can use the **break;** command.

```
while ( true )
{
    runProgram();

    if ( command.equals( "quit" ) )
    {
        break; // leave loop
    }
}
```

Finally, you can use the **continue;** command to continue looping, but skip any more lines of code for the current run-through in the loop.

```
while ( true )
{
    runProgram();

    if ( error == true )
    {
        // skip the rest of the code
        // and start loop again
        continue;
    }

    // this is skipped if continue is hit
    if ( command.equals( "quit" ) )
    {
        break;
    }
}
```

# Sample uses of while loops

## Keep the program running

Keep your programming running until the user selects the **quit** option by using a while loop.

**Language** **Code**

C++

```
bool done = false;
while ( !done )
{
    printMenu();
    int choice;
    cin >> choice;
    // ...
}
```

C#

```
bool done = false;
while ( !done )
{
    printMenu();
    int choice = Convert.ToInt32( Console.Read() );
    // ...
}
```

Java

```
boolean done = false;
while ( !done )
{
    printMenu();
    int choice = input.nextInt();
    // ...
}
```

## Validate user input

Make sure that the user has entered valid input before continuing the program's execution.

**Language** **Code**

C++

```
public int getInput( int minval, int maxval )
{
    cout << "Enter an option: " ;
    int choice;
    cin >> choice;

    while ( choice < minval || choice > maxval )
    {
        cout << "Invalid input. Try again: ";
        cin >> choice;
    }

    // At this point, choice is guaranteed to be
    // between minval and maxval.
    return choice;
}
```

C#

```
public int getInput( int minval, int maxval )
{
    Console.Write( "Enter an option: " );
    int choice = Convert.ToInt32( Console.Read() );

    while ( choice < minval || choice > maxval )
    {
        Console.Write( "Invalid input. Try again: " );
        choice = Convert.ToInt32( Console.Read() );
    }

    // At this point, choice is guaranteed to be
    // between minval and maxval.
    return choice;
}
```

Java

```
public int getInput( int minval, int maxval )
{
    System.out.print( "Enter an option: " );
    int choice = input.nextInt();

    while ( choice < minval || choice > maxval )
    {
        System.out.print( "Invalid input. Try again: " );
        choice = input.nextInt();
    }

    // At this point, choice is guaranteed to be
    // between minval and maxval.
    return choice;
}
```

Remember that this is like the logic:


where we get the value of **true** if it is **true that the user's input was invalid.** If it is true, we enter the while loop and ask the user to re-input the value, until finally we get a valid number. Once the number is valid, our check for **invalidity** returns **false**, and the while loop exits.