TASK

# Beginner Control Structures - If, Elif, Else & the Boolean Data type

Visit our website

## Introduction

### WELCOME TO THE BEGINNER CONTROL STRUCTURES - IF, ELIF, ELSE & THE BOOLEAN DATA TYPE TASK!

In this task, you will learn about a program's flow control. A control structure is a block of code that analyses variables and chooses a direction in which to go based on given parameters. In essence, it is a decision-making process in computing that determines how a computer responds to certain conditions. We'll also consider the boolean, a built-in data type, represented by the True or False keywords to determine an expression's truth value that can be used to steer your program's control.

A note from the
**Hyperion Team**

Let's consider how we make decisions in real life. When you are faced with a problem, you have to see what the problem entails. Once you figure out the crux of the problem, you then follow through with a way to solve this problem. Teaching a computer how to solve problems works in a similar way. We tell the computer to look out for a certain problem and how to solve the problem when faced with it.

**What is Computational Thinking?**

Computational Thinking is the process of solving a problem using a certain manner of thinking. Computational Thinking is used in the development of computer applications, but it can also be used to solve problems across a wide variety of categories, including maths, science, geography, and art. After learning about Computational Thinking, you will be able to apply this to both work and real-life problems.

**Simple Daily Examples of Computational Thinking**

- Looking up a name in your contact list
- Cooking a gourmet meal

## IF STATEMENTS

We are now going to learn about a vital concept when it comes to programming. We will be teaching the computer how to make decisions for itself using an *if statement*. As the name suggests, it is essentially a question. It can compare two or more variables or scenarios and perform a certain action based on the outcome of that comparison.

*If statements* contain a condition. The condition is the part of the *if statement* in brackets in the example below. Conditions are statements that can only evaluate to True or False. If the condition is True, then the indented statements are executed. If the condition is False, then the indented statements are skipped.

In Python, *if statements* have the following general syntax:

```
if condition :
        indented statements
```

Here's an example of a Python *if statement*:

```
num = 10

if num < 12:
      print("the variable num is lower than 12")
```

This *if statement* checks if the variable number is less than 12. If it is, then it will print the sentence letting us know. If *num* were greater than 12, then it would not print out that sentence.

**Notice the following important syntax rules for an *if statement*:**

- In Python, the colon is followed by code that can only run if the statement's condition is True.
- The indentation is there to demonstrate that the program's logic will follow the path indicated by it. Every indented line will run if the program's *if statement's* condition is True.
- In Python, when writing a conditional statement such as the *if statement* above, you have the option of putting the condition in brackets (i.e. *if (num < 12):* ). While your code will still run without them, when your code gets more complicated it could help with readability, which is a very important requirement in coding.

## THE STRUCTURE OF IF-ELSE STATEMENTS

The basic structure of an if-else statement can be represented by this diagram:

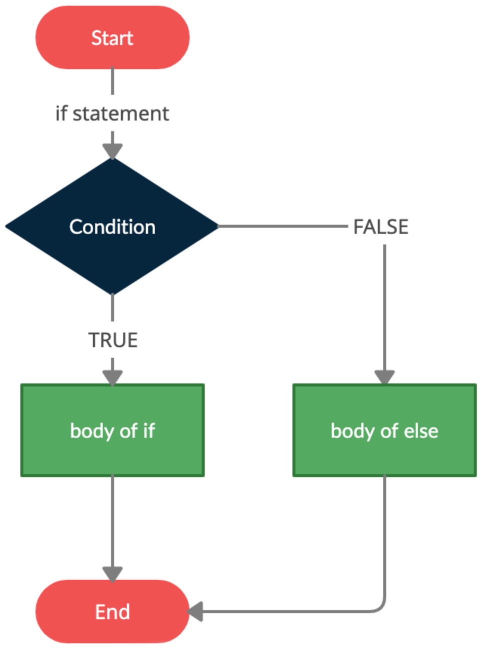

**Image source:**
**https://www.toppr.com/guides/python-guide/tutorials/python-flow-control/if-elif-else/python-if-if-else-if-elif-else-and-nested-if-statement/**

It is mainly used when you want one thing to happen when a condition is *True*, and something else to happen when it is *False*.

## COMPARISON OPERATORS

You may have also noticed the **less than** (<) symbol in line 1. As a programmer, it's important to remember the basic logical commands. We use comparison operators to compare values or variables in programming. These operators work well with *if statements* and *loops* to control what goes on in our programs.

The four basic comparative operators are:

| | |
|---|---|
| greater than | > |
| less than | < |
| equal to | == |
| not | ! |

Take note that the symbol we use for **equals to** is '==', and not '='. This is because '==' literally means 'equals' (e.g. *i == 4* means *i* is equal to 4). We use '==' in conditional statements. On the other hand, '=' is used to assign a value to a variable (e.g. *i* = *"blue"* means I assign the value of *"blue"* to *i*). It is a subtle but important difference.

We can combine the greater than, less than, and the **not operator** with the equals operator and form three new operators.

| | |
|---|---|
| ● greater than or equal to | >= |
| ● less than or equal to | <= |
| ● **not** equal to | != |

As you can see, the *if statement* is pretty limited as is. You can only really make decisions that result in one of two possible outcomes. What happens if we have more options? What if one decision will have further ramifications and we will need to make more decisions to fully solve the problem at hand?

Control structures are not limited to *if statements*. You will soon learn how to expand on an *if statement* by adding an *else statement* or an *elif statement* (else if).

## WHAT ARE BOOLEANS?

*Booleans* were developed by an English mathematician and computer pioneer named George Boole (1815-1864). A boolean data type can only store one of two values, namely *True* or *False*. One byte is reserved for the boolean data type.

Use booleans when checking if one of two outcomes is *True*. For example: is the car insured? Is the password correct? Is the person lying? Do you love me?

Once the information is stored in a variable, it is easy to use *loops* and *if statements* to check an extensive sample of items and base your calculations on the result of a boolean value.

## ASSIGNING BOOLEAN VARIABLES

Assigning a boolean variable is very simple. You declare the variable name and then choose its starting value. This value can then be changed as the program runs.

For example:

```
pass_word = False
pass_word = True
```

## BOOLEANS IN CONTROL STATEMENTS

Control statements allow you to use booleans to their full potential.  As of now we only know how to declare a boolean variable as either *True* or *False*, but how would this benefit us? How would we use it? This is where the *if statement* comes into play. Let's look at a simple decision we might make in our everyday lives.

When you are about to leave your house, do you always take an umbrella? No, you would only take an umbrella if it is raining outside. This is a very rudimentary example of decision-making where there are only two outcomes. We can apply these basic principles to create more complex programs.

```
umbrella = "Leave me at home"
rain = False
if rain:
   umbrella = "Bring me with"
```

Take a look at line 3 in the example above. That is shorthand for `if rain == True:`, but because the default of a boolean is True, to write all of that is redundant, so we only use '==' with boolean conditionals if a condition specifically needs to be False.

## ELIF STATEMENTS

The last piece of the puzzle when it comes to *if statements* is called an *elif statement*. Elif stands for "else if". With this statement, we can add more conditions to the if statement. Therefore, we can test multiple parameters in the same statement.

Unlike the *else statement*, you can have multiple *elif statements* in an *if-elif-else* statement. If the condition of the *if statement* is *False*, the condition of the next *elif statement* is checked. If the first *elif statement* condition is also *False*, the condition of the next *elif statement* is checked, etc. If all the elif conditions are *False*, the *else statement* and its indented block of statements is executed.

In Python, *if-elif-else statements* have the following syntax:

```
if condition1:
      indented Statements
elif condition2:
      indented Statements
elif condition3:
      indented Statements
elif condition4:
      indented Statements
else:
      indented Statements
```

Look at the following example:

```
num = 10

if num < 12:
    print("the variable num is lower than 12")
else:
    print("the variable num is greater than 12")
```

What happens if we want to test multiple conditions? This is where elif comes in.

```
num = 10

if num > 12:
    print("the variable num is greater than 12")
elif num > 10:
    print("the variable num is greater than 10")
elif num < 10:
    print("the variable num is less than 10")
else:
    print("the variable num is 10")
```

Remember that you can combine if, else, and elif into one big statement. This is what we refer to as a *conditional statement*.

**Some important points to note on the syntax of if-elif-else statements:**

- Make sure that the if/elif/else statements end with a colon (the '**:**' symbol).
- Ensure that your indentation is done correctly (i.e. statements that are part of a certain control structure's 'code block' need the same indentation).
- To have an *elif* you must have an *if* above it.
- To have an *else* you must have an *if* or *elif* above it.
- You can't have an *else* without an *if* — think about it!
- You can have many *elif* statements under an *if*, but you may only have one *else* right at the bottom. It's like the fail-safe statement that executes if the other *if/elif* statements fail!

**Think of your own example**

Try to come up with the code for your own example of a simple if/elif/else statement. Remember, start by defining your variables. Maybe you have a variable called breakfast_time and you assign it the value of 11. Maybe you'd like to remind yourself to do something before 11, and if the time is later than 11, you'll do something **else**.

Experiment with different conditional operators like **<**, **!=**, etc. Have fun; these are the building blocks to more complex code.

```
DEFINE YOUR VARIABLE(S)

variable_1 = ...
variable_x = ...

CHECK YOUR VARIABLE(S) AGAINST SPECIFIC VALUES

if variable_1 ... xx:
        What action should take place here?
elif variable_1 ... xx:
        If the first condition isn't met, then what should happen here?
else:
        Finally, if none of the above conditions are met, what should happen?
```

## RECAP

### THE IF STATEMENT

You are already familiar with *if statements*. Remember that in Python, an *if statement* looks like this:

```
num = 10

if num < 12:
    print("the variable num is lower than 12")
```

### ELSE STATEMENTS

The *else statement* represents an alternative path for the flow of logic if the condition of the if statement turns out to be *False*.

Imagine if you were hungry and you sent your friend to the shop to buy chocolate. When they get to the shop, they find no chocolates and just leave because you told them of no alternatives. They would have to keep coming back for instructions unless you provide them with an alternative. Instead of us having many *if statements* to test each scenario, we can add an *else statement* to give us a single alternative.

Remember that, in Python, the general if-else syntax is:

```
if condition :
    indented Statements
else:
    indented Statements
```

If the condition turns out to be *False*, the statements in the indented block following the *if statement* are skipped and the statements in the indented block following the *else statement* are executed.

Take a look at the following example:

```
current_time = 12

if current_time < 11:
    print("Time for a short jog – let’s go!")
```

```python
else:
    print("It’s after 11 – it’s lunch time.")
```

Now instead of nothing happening if the condition of the *if statement* is *False* (*time* ends up being greater than 11), the *else statement* will be executed. In this example, what do you think would happen if the time was set to 11 exactly? Copy, paste, and run the code sample above if you're unsure. Do you think we should amend the code at all to handle this case? How would you do that? Try adding to the code sample and running it.

Another example of using an *else statement* with an *if statement* can be found below, but this one includes an *elif*. The value that the variable *hour* holds determines which string is assigned to the variable *greeting*:

```python
if hour < 18:
     greeting = "Good morning"
elif hour < 20:
    greeting = "Good evening"
else:
     greeting = "Good night"
```

We are faced with decisions like this on a daily basis. For instance, if it is cold outside you would likely wear a jacket. However, if it is not cold, you might not find a jacket necessary. This, as you have learned, is a type of branching: if one condition is true, do one thing, and if the condition is false, do something else. This type of branching decision-making can be implemented in Python programming using 'if–elif-else' statements.

## Instructions

Read through the **example** programs that accompany this task in your task folder. They will give you some helpful tips on how to implement conditional checks in your compulsory task below.

## Compulsory Task 1

Follow these steps:

- Create a Python file called **full_name.py** in this folder.

This program will be used to validate that a user inputs at least two names when asked to enter their full name.

- Ask the user to input their full name.
- Perform some validation to check that the user has entered a **full** name. Give an appropriate error message if they haven't. One of the following messages should be displayed based on the user's input:
  - "You haven't entered anything. Please enter your full name."
  - "You have entered less than 4 characters. Please make sure that you have entered your name and surname."
  - "You have entered more than 25 characters. Please make sure that you have only entered your full name."
  - "Thank you for entering your name."

  The error message examples should help you to determine the sorts of checks your program will need to perform on the data that the user provides.

Rate us

## Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

**Click here** to share your thoughts anonymously.