HyperionDev

Control Structures – If, Elif, Else, and the Boolean Data Types

Task

Visit our website

Introduction

How do 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.

In this task, you will learn about a program's flow of 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 will also consider the boolean data type (a built-in data type, represented by the "True" or "False" keywords) to determine an expression's truth value so as to steer your program's control.

If statements

We are now going to learn about a vital concept in programming. We will be teaching the computer how to make decisions for itself using something called an `if` statement. As the name suggests, this is essentially a question. `If` statements 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, as you can see 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 a code example of a Python `if` statement:

```python
# Checks if the variable num is lower than 12 and prints a message if it is
num = 10

if num < 12: # Don't forget the colon!
    print("The variable num is lower than 12")
```

This `if` statement uses the less than (<) operator to check if the value of the variable `num` is less than 12. If it is, then the program will print the sentence within the body of the `if` statement. If `num` was greater than 12, then it will not print out that sentence. This is a slightly contrived example for the sake of simplicity because we already know `num` is less than 12, having assigned it the value 10.

However, imagine that the value of `num` was input at run-time by a user; in that case, we could not know in advance what they would enter, and the condition the `if` statement evaluates would offer more value. Note that as all user input is assigned the data type string, we need to cast the input to an `int` if we want to perform operations based on its value. See the example below:

```
# Prompts the user to input a number between 1 and 100
num = int(input("Please input a number between 1 and 100: "))

# Checks if the number is less than 12
if num < 12: # Don't forget the colon!
    print("The value you entered is lower than 12")
```

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 it indicates. 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 if you'd prefer to (i.e., `if (num < 12):`). While your code will still run without the brackets, using brackets can help with readability when your code gets more complicated, which is a very important consideration in coding.

Comparison operators

You may have also noticed the less than (<) symbol in the previous code examples (`if num < 12:`). As a programmer, it's important to remember the basic logical operators. We use comparison operators to compare values or variables in programming. These operators work well with `if` statements, `if-else` 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 equal to is `==`, and not `=`, 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 string blue to i). It is a subtle but important difference.

We can combine the greater than, less than, and 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). First, however, let’s find out about the boolean data type.

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 storing this data type.

We 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?

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 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
```

Control statements allow you to use booleans to their full potential. You have just learnt how to declare a boolean variable (either "True" or "False"), but how can this benefit you? How can you use a boolean value once you have declared it? This is where the `if` statement comes into play. Let's look at a simple decision we might make in our everyday lives.

If it's cold outside you would likely wear a jacket. However, if it's not cold, you might find a jacket unnecessary. This, as you have learnt, 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-else` and `if-elif-else` statements.

Let's try another example. When you are about to leave your house, do you always take an umbrella? No, you only take an umbrella if it's raining outside. Although this is another very rudimentary example of decision-making where there are only two outcomes, keep in mind that we can apply these basic principles to create more complex programs. Here's the scenario represented in code:

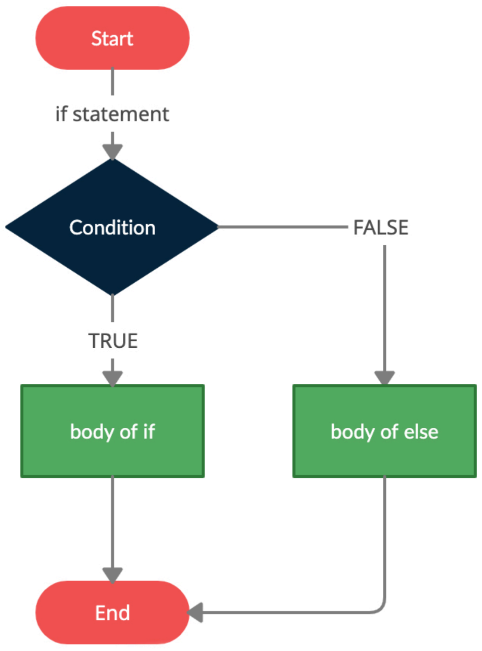
```
# Decide whether to bring an umbrella based on the rain
umbrella = "Leave me at home"
rain = False
if rain:
    umbrella = "Bring me with"
```

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

If-else statements

In addition to the basic `if` statement functionality you have just learnt about, it's possible to check the condition supplied to the `if` statement and not only do something if it evaluates to True, but also do something else if it is False, as we discussed when talking about branching decisions. This is called an `if-else` statement.

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



Python if-else statement flowchart (Toprr, n.d.)

The example below extends the earlier `if` statement example to include `else`:

```python
# Prompts the user to input a number between 1 and 100
num = int(input("Please input a number between 1 and 100: "))

# Checks if the number is less than 12 and prints an appropriate message
if num < 12:
    print("The value you entered is lower than 12")
else:
    print("The value you entered is higher than 12")
```

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, you can add more conditions to the `if` statement, and in doing so, you can test multiple parameters in the same code block.

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 Statement(s)
elif condition2:
      indented Statement(s)
elif condition3:
      indented Statement(s)
elif condition4:
      indented Statement(s)
else:
      indented Statement(s)
```

`Elif` can be used to test multiple conditions, as in the following example:

```python
# Prompts the user to input a number between 1 and 100
num = int(input("Please input a number between 1 and 100"))

# Checks if the number is lower than 12, higher than 13 or exactly 13.
if num < 12:
    print("The value you entered is lower than 12")
elif num > 13:
    print("The value you entered is higher than 13")
elif num < 13:
    print("The value you entered is lower than 13")
else:
    print("The value you entered is 13")
```

Try copying and pasting the code above into VS Code and running it, with different numbers as input. Once you're happy you can follow the logic, try changing the numbers used and creating different output strings until you really feel comfortable with how `if-elif-else` works.

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 all evaluate to False!

Remember this overview of the basic structure you're learning:

```
# Define your variable(s)

variable_1 = ...  # Replace '...' with your variable
variable_x = ...   # Replace '...' with your variable

# Check your variable(s) against specific values

if variable_1 ...:  # Replace '...' with a comparison to a variable
    # What action should take place here?
elif variable_1 ...:  # Replace '...' with a comparison to a variable
    # What should happen here if the first condition isn't met?
else:
    # Finally, what should happen if none of the above conditions are met?
```

Extra resource

Read the **[PEP 8 style guide section on code layout]** for best practices on structuring your code.

Simple examples to try

Take a look at the following example:

```
# Checks the current time and suggests an activity
current_time = 12
if current_time < 11:
    print("Time for a short jog - let's go!")
else:
    print("It's after 11 - it's lunch time.")
```

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 and paste the code into VS Code, and run it. Do you think we should amend the code 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 **hour** variable holds determines which string is assigned to the **greeting** variable:

```
# Checks the current hour and sets a greeting based on the time of the day
if hour < 18:
    greeting = "Good day"
elif hour < 20:
    greeting = "Good evening"
else:
    greeting = "Good night"
```

How could you change the code above to get the value of **hour** from the user, so that the output of the program would be dependent on the user's input? What would you have to do to the input value to use it in the logical tests of the conditions? Copy, paste, and run the code sample above first, and then try amending it to accept and act on user input. Hopefully, by now you are starting to see the immense value that **if-elif-else** statements can offer to you as a programmer!

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 9. Maybe you'd like to remind yourself to do something before 9, and if the time is later than 9, you will do something else. Experiment with different conditional operators like `<`, `!=`, and others, and with using values input by a user. Have fun as you are mastering the building blocks for more complex code.

Instructions

Read through the example programs in your task folder that accompany this task. They will give you some additional simple examples of how to implement conditional checks that you can draw on when doing your task below.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select "Request Review", the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you've done that, feel free to progress to the next task.

Auto-graded task

Follow these steps:

- Create a new Python file called **age-quiz.py**. The program you create in this file will be used to output a variety of responses determined by the data the user enters.
- Write code to take in a user's age and store it in an integer variable called `age`.
- Assume that the oldest someone can be is 100; if the user enters a higher number, output the message: "Sorry, you're dead."
- If the user is 40 or over, output the message: "You're over the hill."
- If the user is 65 or older, output the message: "Enjoy your retirement!"
- If the user is under 13, output the message: "You qualify for the kiddie discount."
- If the user is 21, output the message: "Congrats on your 21st!"
- For any other age, output the message: "Age is but a number."
- Tips:
  - You should use the `if-elif-else` statement structure to create the program.
  - Note that the order in which you set up the conditions is important; you will need to work out what makes sense in terms of the logical tests applied to the ages.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Toprr. (n.d.). *Python if, if...else, if...elif...else and nested if statement*. **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/**