TASK

Beginner Control Structures — While Loop

Visit our website

Introduction

WELCOME TO THE BEGINNER CONTROL STRUCTURES — WHILE LOOP TASK!

In this task, you will be exposed to *loop statements* to understand how they can be utilised in reducing lengthy code, preventing coding errors, as well as paving the way towards code reusability. This task begins with the *while loop*, which is the simplest loop of the group.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

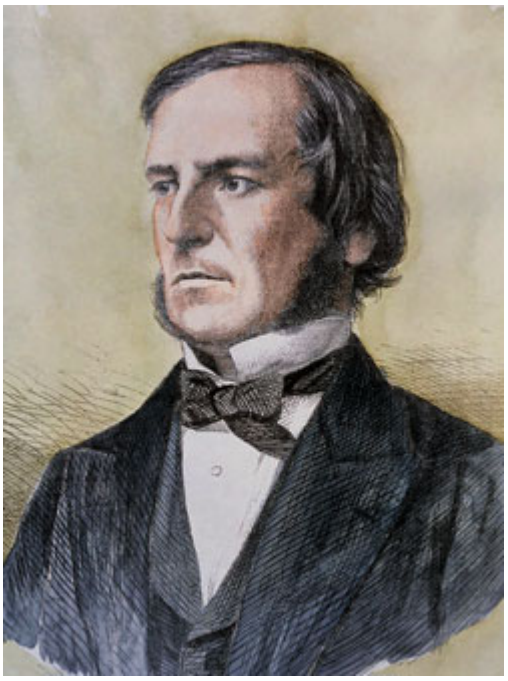
Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the

HyperionDev Team

By now you should be familiar with the Boolean data type, and the vital role Boolean algebra plays in computer programming. Why is it called a Boolean datatype? It is named after George Boole (1815 - 1864) an English mathematician who helped establish modern symbolic logic and devised Boolean algebra. Boole is best known as the author of The Laws of Thought (1854), which contains Boolean algebra.

Boole was born in Lincoln, Lincolnshire, England. His father, a struggling shoemaker, encouraged him to take an interest in mathematics. Boole went to an elementary school and trade school for a short time, but he mostly educated himself. By the age of fifteen, Boole began teaching. At age nineteen, he set up a school in Lincoln. Boole was appointed as a professor of mathematics at Queen's College, Cork in 1849, despite not holding any university degree.



George Boole

WHAT IS A WHILE LOOP?

A *while loop* is the most general form of loop statements. The *while statement* repeats its action until the controlling condition becomes False. In other words, the statements indented in the loop repeatedly executes "while" the condition is True (hence the name). The while statement begins with the keyword *while*, followed by a boolean expression. The expression is tested before beginning each iteration or repetition. If the test is True, then the program passes control to the indented statements in the loop body; if False, control passes to the first statement after the body.

Syntax:

```
while boolean expression:
   statement(s)
```

The following code shows a while statement which sums successive even integers 2 + 4 + 6 + 8 + ... until the total is greater than 250. An update statement increments i by 2 so that it becomes the next even integer. This is an event-controlled loop (as opposed to counter-controlled loops, like the *for loop*) because iterations continue until some non-counter-related condition (event) stops the process.

```
sum1 = 0
i = 2                    # initial even integer for the sum
while sum1 <= 250:
    sum1 += i
    i += 2               # update statement, shorthand for i = i + 2
    print(sum1)
```

Compile and run the **example.py** file to see the output of the above program.

GET INTO THE LOOP OF THINGS

Loops are handy tools that enable programmers to do repetitive tasks with minimal effort. To count from 1 to 10, we could write the following program:

```
print(1)
print(2)
print(3)
print(4)
```

```
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

The task will be completed correctly. The numbers 1 to 10 will be printed, but there are a few problems with this solution:

- **Efficiency:** repeatedly coding the same statements takes a lot of time.
- **Flexibility:** what if we wanted to change the start number or end number? We would have to go through and change them, adding extra lines of code where they're needed.
- **Scalability:** 10 repetitions are trivial, but what if we wanted 100 or even 1000 repetitions? The number of lines of code needed would be overwhelming and very tedious for a large number of iterations.
- **Maintenance:** where there is a large amount of code, one is more likely to make a mistake.
- **Feature:** the number of tasks is fixed and doesn't change at each execution.

Using loops, we can solve all these problems. Once you get your head around them, they will be invaluable in solving many problems in programming.

Consider the following code:

```
i=0
while i < 10:
    i+=1                    # shorthand for i = i + 1
    print(i)
```

If we run the program, the same result is produced, but looking at the code, we immediately see the advantages of loops. Instead of executing 10 different lines of code, line 4 executes 10 times. 10 lines of code have been reduced to just 4. This is done through the update statement in line 3, that adds 1 to variable *i* each iteration until *i == 10*. (Update statements will be discussed in more detail in the next Task.)Furthermore, we may change the number 10 to any number we like. Try it yourself, replace the 10 with another number.

INFINITE LOOPS

A *while loop* runs the risk of running forever if the condition never becomes False. A loop that never ends is called an infinite loop. Creating an infinite loop is not desirable. Make sure that your loop condition eventually becomes False and that your loop is exited.

In any loop, the following three steps are usually used:

1. Declare a counter/control variable. The code above does this when it says “i = 0”. This creates a variable called i that contains the value 0.

2. Increase the counter/control variable in the loop. In the loop above, this is done with the instruction “i+=1” which increases i by one with each pass of the loop.

3. Specify a condition to control when the loop ends. The condition of the for loop below is “i < 10”. This loop will carry on executing as long as i is less than 10. This loop will, therefore, execute ten times.

Instructions

Before you get started, we strongly suggest you start using Notepad++ or IDLE to open all text files (.txt) and python files (.py). Do not use the normal Windows notepad as it will be much harder to read.

First read **example.py**, open it using IDLE (Right-click the file and select 'Edit with IDLE').

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run example.py to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Create a new file called **even.py**.
- Write a program that asks the user to enter a number.
- Make use of the *while loop* repetition structure so the program prints out all the even numbers from 1 up to (and possibly including) the number given by the user.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Create a program called **names.py** to do the following:
    - Require the user to enter the names of all pupils in a class. The user should be able to type "Stop" to indicate that the names of all the students have been entered.
    - Print out the total number of names the users entered after the loop has been exited.

- Compile, save and run your file.

Compulsory Task 3

Follow these steps:

- Create a new file called **while.py**.
- Write a program that always asks the user to enter a number.
- When the user enters -1, the program should stop requesting the user to enter a number,
- The program must then calculate the average of the numbers entered, excluding the -1.
- Make use of the *while loop* repetition structure to implement the program.
- Compile, save and run your file.

Optional Bonus Task

Follow these steps:

- Create a new file in this folder called **optional_task.py**.
- Ask the user to enter a number less than or equal to 100.
- If they enter one above 100, ask them to try again (and continue to do so until they follow instructions).
- Once they have entered a valid number, check if it is even. If it is, multiply it by 2 and print it out. If it isn't, multiply it by 3 and print it out.

Thing(s) to look out for:

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

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.