Python while else

If this Python Tutorial saves you hours of work, please whitelist it in your ad blocker 😭 and

Donate Now

(https://www.pythontutorial.net/donation/)

to help us ❤️ pay for the web hosting fee and CDN to keep the website running.

Summary: in this tutorial, you'll learn about the Python `while else` statement and how to use it effectively.

Introduction to Python while else statement

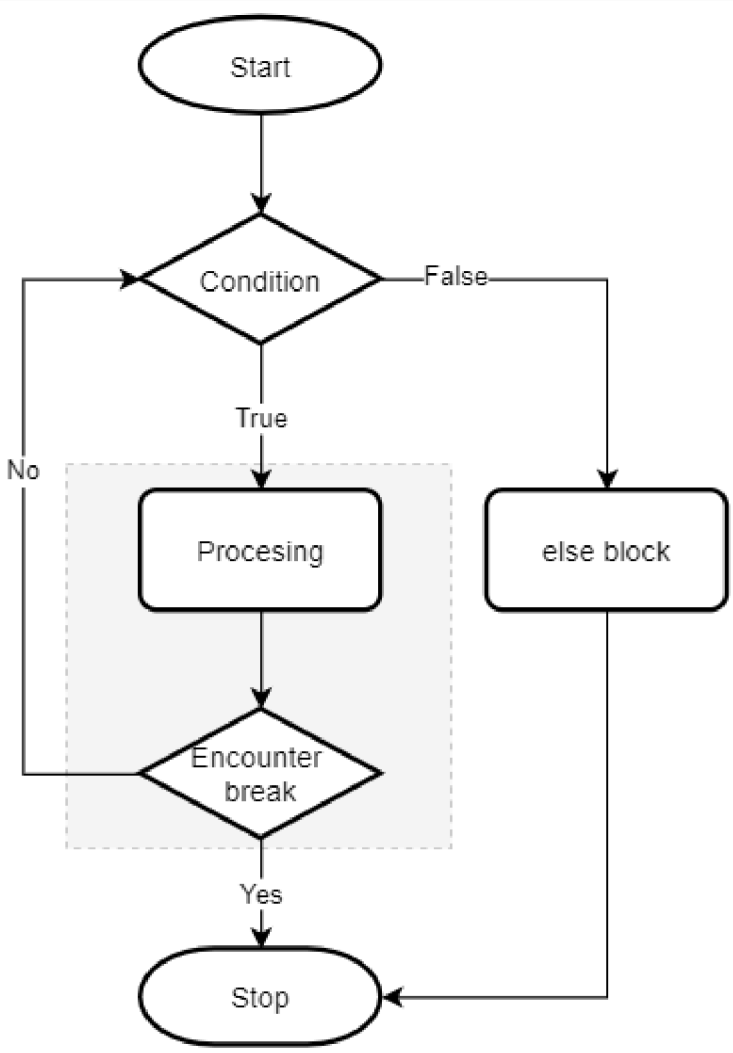
In Python, the `while` (https://www.pythontutorial.net/python-basics/python-while/) statement may have an optional `else` clause:

```
while condition:
    # code block to run
else:
    # else clause code block
```

In this syntax, the `condition` is checked at the beginning of each iteration. The code block inside the `while` statement will execute as long as the `condition` is `True`.

When the `condition` becomes `False` and the loop runs normally, the `else` clause will execute. However, if the loop is terminated prematurely by either a `break` (https://www.pythontutorial.net/python-basics/python-break/) or `return` statement, the `else` clause won't execute at all.

The follwing flowchart illustrates the `while...else` clause:



If you're familiar with other programming languages such as JavaScript (https://www.javascripttutorial.net/javascript-while-loop/) , Java, or C#, you'll find that the `else` clause is quite strange in the context of a loop.

However, the `while else` clause turns out to be very useful in some cases. Let's take a look at an example of using the `while else` statement.

Python while else statement example

Suppose that we have the following list of fruits where each fruit is a dictionary that consists of the `fruit` name and `qty` keys:

```
basket = [
    {'fruit': 'apple', 'qty': 20},
    {'fruit': 'banana', 'qty': 30},
    {'fruit': 'orange', 'qty': 10}
]
```

We want to make a program that allows the users to enter a fruit name. Based on the input name, we'll search for it from the `basket` list and show its quantity if the fruit is on the list.

In case the fruit is not found, we'll allow users to enter the quantity for that fruit and add it to the list.

The following program is the first attempt:

```
basket = [
    {'fruit': 'apple', 'qty': 20},
    {'fruit': 'banana', 'qty': 30},
    {'fruit': 'orange', 'qty': 10}
]

fruit = input('Enter a fruit:')

index = 0
found_it = False

while index < len(basket):
    item = basket[index]
    # check the fruit name
    if item['fruit'] == fruit:
        found_it = True
        print(f"The basket has {item['qty']} {item['fruit']}(s)")
        break

    index += 1
```

```
if not found_it:
    qty = int(input(f'Enter the qty for {fruit}:'))
    basket.append({'fruit': fruit, 'qty': qty})
    print(basket)
```

Note that there's better way to develop this program. The program in this example is solely for the demonstration purpose.

How it works:

- First, prompt for an user input by using the `input()` function.
- Second, initialize the `index` to zero and `found_it` flag to `False`. The `index` will be used for accessing the list by index. And the `found_it` flag will be set to `True` if the fruit name will be found.
- Third, iterate over the list and check if the fruit name matched with the input name. If yes, set the `found_it` flag to `True`, show the fruit's quantity, and exit the loop by using the `break` statement.
- Finally, check the `found_it` flag after the loop and add the new fruit to the list if the `found_it` is `False`.

The following runs the program when apple is the input:

```
Enter a fruit:apple
The basket has 20 apple(s)
```

And the following runs the program when lemon is the input:

```
Enter a fruit:lemon
Enter the qty for lemon:15
[{'fruit': 'apple', 'qty': 20}, {'fruit': 'banana', 'qty': 30}, {'fruit': 'orange
```

The program works as expected.

However, it'll be more concise if you use the `while else` statement instead.

The following shows the new version of the program that uses the `while else` statement:

```
basket = [
    {'fruit': 'apple', 'qty': 20},
    {'fruit': 'banana', 'qty': 30},
    {'fruit': 'orange', 'qty': 10}
]

fruit = input('Enter a fruit:')

index = 0

while index < len(basket):
    item = basket[index]
    # check the fruit name
    if item['fruit'] == fruit:
        print(f"The basket has {item['qty']} {item['fruit']}(s)")
        found_it = True
        break

    index += 1
else:
    qty = int(input(f'Enter the qty for {fruit}:'))
    basket.append({'fruit': fruit, 'qty': qty})
    print(basket)
```

In this program, the `else` clause replaces the need of having the `found_it` flag and the `if` statement after the loop.

If the fruit is not found, the `while` loop is terminated normally and the `else` clause will be executed to add a new fruit to the list.

However, if the fruit is found, the `while` loop will be encountered the `break` statement and terminated prematurely. In this case, the `else` clause won't be executed.

Summary

- The `else` clause in the `while else` statement will execute when the `condition` of the `while` loop is `False` and the loop runs normally without encountering the `break` or `return` statement.
- Try the Python `while else` statement whenever you need to have a flag in a `while` loop.