Python: Looping Processing (While Statement)

2nd Lecture

1. Introduction

Earlier we examined the for loop, which executes a set of statements a definite number of times specified by the programmer. In many situations, however, the number of iterations in a loop is unpredictable. The loop eventually completes its work, but only when a condition changes. For example, the user might be asked for a set of input values. In that case, only the user knows the number she will enter. The program's input loop accepts these values until the user enters a special value or sentinel that terminates the input. This type of process is called conditional iteration. In this section, we explore the use of the while loop to describe conditional iteration.

2. The Structure and Behavior of a while Loop

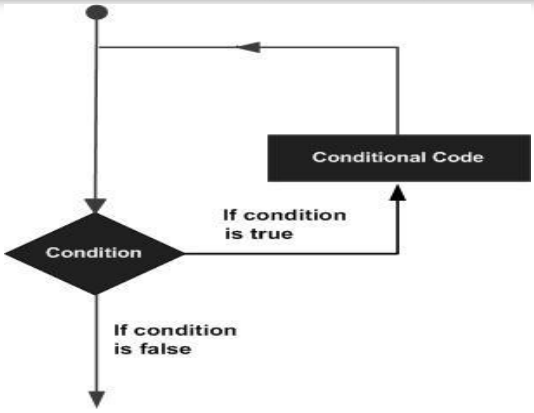
Conditional iteration requires that a condition be tested within the loop to determine whether the loop should continue. Such a condition is called the loop's continuation condition. If the continuation condition is false, the loop ends. If the continuation condition is true, the statements within the loop are executed again. The while loop is tailor-made for this type of control logic.

Here is the syntax of a while loop:

while <condition>:

<sequence of statements>

While the flow diagram for the semantics of a while loop:



When using while loop, need to consider these points:

- The form of this statement is almost identical to that of the one-way selection statement.
- The first input statement initializes a variable to a value that the loop condition can test. This variable is also called the **loop control variable**. The
- The sequence of statements might be executed many times, as long as the condition remains true.
- Otherwise, the loop is broken when the condition becomes false.
- If loop will continue forever, an error known as an **infinite loop**.
- At least one statement in the body of the loop must update a variable that affects the value of the condition.
- The while loop is also called an entry-control loop, because its condition is tested at the top of the loop.

Example 1: This example is a short script that prompts the user for a series of numbers, computes their sum, and outputs the result. Instead of forcing the user to enter a definite number of values, the program stops the input process when the user simply presses the return or enter key. We first present a rough draft in the form of a Pseudocode algorithm:

SummationNumber algorithm:

1. set the sum to 0.0
2. input a string
3. while the string is not the empty
4. string convert the string to a float
5. add the float to the sum

6. input a string
7. print the sum

Here is the Python code for this script, followed by a trace of a sample run:

```
sum = 0.0
data = input("Enter a number or just enter to quit: ")
while data != "":
  number = float(data)
  sum += number
  data = input("Enter a number or just enter to quit: ")
print("The sum is", sum)
```

During running

```
Enter a number or just enter to quit: 3
Enter a number or just enter to quit: 4
Enter a number or just enter to quit: 5
Enter a number or just enter to quit:
The sum is 12.0
```

3. Count Control with a while Loop

You can also use a while loop for a count-controlled loop. The next two code segments show the same summations with a for loop and a while loop, respectively.

Example 2: Write the Python code to convert this equation: sum=$\sum_{i=1}^{100000} i$

```
sum = 0
for count in range(1, 100001):
  sum += count
print(sum)
```

```
sum = 0
count = 1
while count <= 100000:
  sum += count
  count += 1
print(sum)
```

Although both loops produce the same result, there is a **tradeoff**. The second code segment is noticeably more complex. *It includes a Boolean expression and two extra statements that refer to the count variable.*

Example 3: Write a script that counts down, from an upper bound of 10 to a lower bound of 1?

```
for count in range(10, 0, -1):
  print(count, end=" ")
```

```
count = 10
while count >= 1:
  print(count, end=" ")
  count -= 1
```

4. The while True Loop and the break Statement

The first example script of this section, which contained two input statements. This loop's structure can be simplified if we receive the first input inside the loop, and **break** out of the loop if a test shows that the continuation condition is false. Here is the modified script:

```
sum = 0.0
while True:
   data = input("Enter a number or just enter to quit: ")
   if data == "":
     break
   number = float(data)
   sum += number
print("The sum is", sum)
```

During running

```
Enter a number or just enter to quit: 3
Enter a number or just enter to quit: 4
Enter a number or just enter to quit: 5
Enter a number or just enter to quit:
The sum is 12.0
```

In this code, the loop's entry condition is the Boolean value True. As well as, if the user wants to quit, the input will equal the empty string, and the break statement will cause an exit from the loop.

Example 4: This example modifies the input section of the grade-conversion pro- gram to continue taking input numbers from the user until she enters an acceptable value:

```
while True:
   number = int(input("Enter the numeric grade: "))
   if number >= 0 and number <= 100:
      break
   else:
      print("Error: grade must be between 100 and 0")
print(number) # Just echo the valid input
```

During running

```
Enter a numeric grade: 101
Error: grade must be between 100 and 0
Enter a numeric grade: 25
25
```

Some cases where the body of the loop must execute at least once, this technique simplifies the code and actually makes the program's logic clearer.

Example 5: Here is a version of the numeric input loop that uses a Boolean variable:

```
done= False
while not done:
  number = int(input("Enter the numeric grade: "))
  if number >= 0 and number <= 100:
      done = True
  else:
     print("Error: grade must be between 100 and 0")
print(number) # Just echo the valid input
```

5. Loop Logic, Errors, and Testing

- *Because while loops can be the most complex control statements*, to ensure their correct behavior, careful design and testing are needed.
- Testing a while loop must combine elements of testing used with for loops and with selection statements.
- Errors to rule out during testing the while loop includes an incorrectly initialized loop control variable, failure to update this variable correctly within the loop, and failure to test it correctly in the continuation condition.
- Moreover, if one simply forgets to update the control variable. To halt a loop that appears to be hung during testing, type Control+c in the terminal window or in the IDLE shell.
- If the loop must run at least once, use a while True loop and delay the examination of the termination condition until it becomes available in the body of the loop.
- Ensure that something occurs in the loop to allow the condition to be checked and a break statement to be reached eventually.

6. Exercises

1. Translate the following for loops to equivalent while loops:
   a) for count in range(100):
      print(count)
   b) for count in range(1, 101):
      print(count)
   c) for count in range(100, 0, -1):
      print(count)
2. The factorial of an integer N is the product of all of the integers between 1 and N, inclusive. Write a while loop that computes the factorial of a given integer N?
3. Write the code of Infinite loop using while statement?

<Best Regards>

Dr. Raaid Alubady