Introduction to Python

While and Nested Loops

Indefinite Loop

A **for** loop, we discussed earlier is an example of a **definite loop**, the number of iterations can be specified ahead of time by the programmer.

In some cases, however, the number of iterations can be unknown. This type of loop is called an **indefinite loop.** For example, a user is asked to enter a set of inputs. The number of inputs the user enter is not known in advance. The program's input loop accepts these values until the user enters a special value or **sentinel** that terminates the input.

In this section, we explore the use of the **while** loop to describe conditional iteration: iteration that repeats as long as a condition is true.

Between the two loops, the for loop is more common.

While Loop

The **while loop** has the syntax:

```
while <condition>:
    block
```

The loop **r**epeatedly executes its block of code as long as the condition is true.

At least one statement in the block of the loop must update a variable that affects the value of the condition. Otherwise, the loop will continue forever, an error known as an **infinite loop**.

While vs For Loops I

The following two segments of code are equivalent.

```
# Counting using a for loop from 1 to 10.
for i in range(1, 11):
    print(i, end=" ")
```

For loop syntax is more rigid and structured.
The index i initialization and increment is done compactly.

```
# Counting using a while loop from 1 to 10.
i = 1
while i < 11:
    print(i, end=" ")
    i += 1
```

The while loop has a looser syntax.
The index i initialization, conditional check and increment are at three different places.

While vs For Loops 2

The following two segments of code are equivalent.

```
# Summation with a for loop
sum = 0
for i in range(1, 101):
     sum += i
print(sum)
# Summation with a while loop
sum = 0
i = 1
while i <= 100:
     sum += i
     i += 1
print(sum)
```

Indefinite Loop

For loops are commonly used for definite loops: loops where the number of iterations is known in advance.

While loops are used for indefinite loops: loops where the number of iterations is unknown in advance.

While Loop

Here's an example of a loop where the number of iterations is unknown in advance. Suppose a program computes a sum of a set of inputs. We don't know when the user finishes entering the inputs. Instead of a for loop, a while loop is more appropriate.

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

Sample Run

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

Sample Run:

```
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
```

While and Break

Alternatively, the previous program can be simplified so that only one input is used. The program uses the **break** statement to exit the loop if the input is an empty string.

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

While and Break

As another example of an indefinite loop, suppose the user is asked to enter a numeric grade. The program will keep asking the user until he enters a value in the appropriate range(0 – 100).

```
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
```

While Loop

```
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
```

```
Enter the numeric grade: 101
Error: grade must be between 100 and 0
Enter the numeric grade: –1
Error: grade must be between 100 and 0
Enter the numeric grade: 45
45
```

Random Numbers

In some situation, we like to be able to simulate randomness. For example, we might toss a coin or roll a die.

The Python's random module contains many functions to do this. The function **randrange()** is easy to use since it is similar to the **range()** function we used in for loops. We must first import the random module to access its code. This is done using the statement "import random".

randrange(start, stop, step) generates a random integer beginning with start(including) and ending with stop(not including) with step.

```
import random

for i in range(10):

    num = random.randrange(1, 5)

    print(num, end=” ”)
```

Output:

```
3 3 3 2 1 3 2 2 3 4
```

Generate random sequence

Write a segment of code that prints out a sequence of random numbers from 1 to 100. Stop once there are exactly 5 numbers from 1 to 10.

Note that, we don't know how many iterations we need to get 5 numbers from 1 to 10. This is an indefinite loop. It's better to use a while loop.

```
import random

count = 0

while count < 5:

    num = random.randrange(1, 101)

    print(num, end=" ")

    if num <= 10:

        count += 1
```

Output:

```
3 26 22 5 69 86 45 30 45 53 34 82 94 4 21 30 2 87 94 80 41 8
```

Nested Loops

A *nested loop* is a loop inside of another loop.

```
for i in range(1, 4):
    for j in range(1, 5):
        print(i * j, end=‘ ’)
    print()
```

```
1 2 3 4
2 4 6 8
3 6 9 12
```

Nested Loops Example 1

```
for i in range(1, 6):
     for j in range(1, i+1):
          print(j, end=‘ ’)
     print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Nested Loops Example 2

```
for i in range(1, 6):
     for j in range(6, i, -1):
          print(j, end=‘ ’)
     print()
```

```
6 5 4 3 2
6 5 4 3
6 5 4
6 5
6
```

Sum of 2D lists

A 2D list is a list of lists.

```
list2d = [[1, 2, 5],
          [3, -1, 6],
          [10, 1, 0]]
sum = 0
for row in list2d:
     for num in row:
          sum += num
print(sum)
```

Equivalently:

```
list2d = [[1, 2, 5],[3, -1, 6],[10, 1, 0]]
```

Output:

```
27
```

List of Students

```
students = [["Mike", 80],
            ["John", 90],
            ["Sarah", 85]]
sum = 0
for student in students:
     sum += student[1]        # why student[1]?
ave = sum / len(students)
print(ave)
```

Output:

```
85.0
```

Lab 1

In this lab, we will write a simple guessing game.

The computer randomly generate a number in some given range. On each pass through the loop, the user enters a number to attempt to guess the number selected by the computer.

The program responds by saying "You've got it," "Too large, try again," or "Too small, try again." When the user finally guesses the correct number, the program congratulates him and tells him the total number of guesses.

There is a template repl for this lab at here:

https://repl.it/@LongNguyen18/GuessingGameLab

Let's look at a sample run. What's the strategy for guessing the number?

Guessing Game

```
Enter the smaller number: 10
Enter the larger number: 5
Error! The small number must be <= the larger number. Try again!
 Enter the smaller number: 1
Enter the larger number: 100
Enter your guess: 50
Too small!
Enter your guess: 75
Too large!
Enter your guess: 62
Too small!
Enter your guess: 68
Too large!
Enter your guess: 65
You've got it in 5 tries!
```

There is a template repl for this lab at here:

https://repl.it/@LongNguyen18/GuessingGameLab

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.
2) Lambert, Kenneth, Fundamentals of Python: First Programs, Cengage Learning, 2017.