CORVIT
Reaching end to end

Introducing Python

Lecture# 6

by

python™

Umair bin Mansoor

Network Programming Planner

OBJECTIVES

After this session, students will be able to:

- **Write programs for executing statements repeatedly by using a while loop**
- **Write programs for executing statements repeatedly by using a for loop**
- **Write nested for loops**
- **Write concise statements using list comprehension**
- **Write nested if statements using list comprehension**
- **Access *seaborn* database using *Pandas* module and apply list comprehension**

INTRODUCTION

- A loop can be used to tell a program to execute statements repeatedly.
- Suppose that you need to display a string (e.g., Programming is fun!) 100 times. It would be tedious to type the statement 100 times:

100 times
```
print("Programming is fun!")
print("Programming is fun!")
...
print("Programming is fun!")
```

- A **`while`** or a **`for`** loop can be used to solve this problem

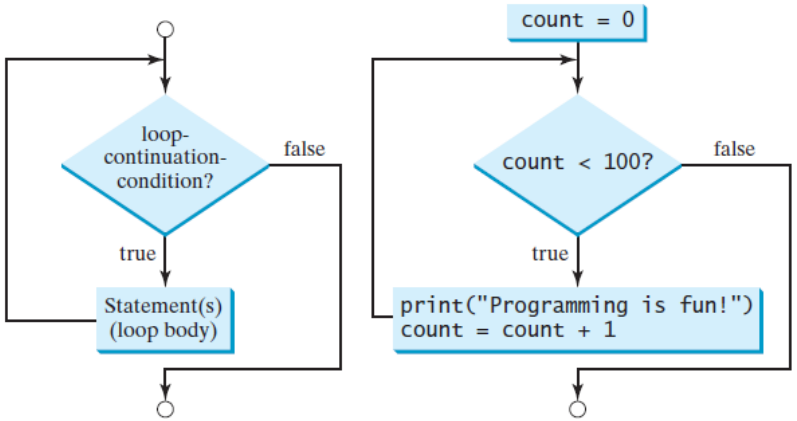
```
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
```

THE `while` LOOP

- A while loop executes statements repeatedly as long as a condition remains true.
- The syntax for the `while` loop is:

```
while loop-continuation-condition:
    # Loop body
    Statement(s)
```

- A single execution of a loop body is called an iteration (or repetition) of the loop.



(a) A while loop (b) A while loop example

RepeatSubtractionQuiz(6_1).py

```
1  import random
2
3  # 1. Generate two random single-digit integers
4  number1 = random.randint(0, 9)
5  number2 = random.randint(0, 9)
6
7  # 2. If number1 < number2, swap number1 with number2
8  if number1 < number2:
9      number1, number2 = number2, number1
10
11 # 3. Prompt the student to answer "What is number1 - number2?"
12 answer = eval(input("What is " + str(number1) + " - "
13     + str(number2) + "? "))
14
15 # 4. Repeatedly ask the question until the answer is correct
16 while number1 - number2 != answer:
17     answer = eval(input("Wrong answer. Try again. What is "
18         + str(number1) + " - " + str(number2) + "? "))
19
20 print("You got it!")
```

NESTED LOOPS

- A loop can be nested inside another loop.
- Nested loops consist of an outer loop and one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered and started anew.

```
1  print("          Multiplication Table")
2  # Display the number title
3  print("   |", end = '')
4  for j in range(1, 10):
5      print("  ", j, end = '')
6  print()  # Jump to the new line
7  print("--------------------------------------")
8
9  # Display table body
10 for i in range(1, 10):
11     print(i, "|", end = '')
12     for j in range(1, 10):
13         # Display the product and align properly
14         print(format(i * j, "4d"), end = '')
15     print()  # Jump to the new line
```

- The indentation of each statement defines the body of each loop.
- The following program MultiplicationTable.py uses nested **for** loops to display a multiplication table.

LIST COMPREHENSION

```
1. h_letters = []
2. for letter in 'human':
3.     h_letters.append(letter)
4. print(h_letters)   # ['h', 'u', 'm', 'a', 'n']
```

- The above code can be compressed using list comprehension. The example is as follows:

```
1. h_letters = [letter for letter in 'human']
2. print(h_letters)
```

- The `for` loop statement returns a single letter at a time and appends it in the list `h_letters`. The square brackets signifies a list.
- `h_letters` is a list of strings. Each string contains a single character.

NESTED IF WITH LIST COMPREHENSION

- **Example 1**

```
1. num_list1 = [y for y in range(100) if y % 10 == 0]
2. print(num_list1) #[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

- **Example 2**

```
1. num_list2 = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]
   print(num_list2) #[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

- **Example 3**

```
1. obj = ["Even" if i%2==0 else "Odd" for i in range(10)]
2. print(obj) #['Even','Odd','Even','Odd','Even','Odd','Even','Odd','Even','Odd']
```

LIST COMPREHENSION SUMMARY

1. List comprehension is an elegant way to define and create lists based on existing lists.
2. List comprehension is generally more compact and faster than normal functions and loops for creating list.
3. However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
4. Remember, every list comprehension can be rewritten in for loop, but every for loop can't be rewritten in the form of list comprehension.
5. A further, more detailed example on list comprehension is available on GitHub.

Questions & Answers