Loops in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- work with the loop statements in Python, including for-loop and while-loop.

Loops in Python

Welcome! This notebook will teach you about the loops in the Python Programming Language. By the end of this lab, you'll know how to use the loop statements in Python, including for loop, and while loop.

Table of Contents

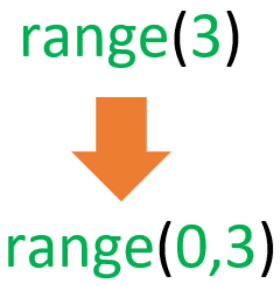
Loops

Range

Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by **loops**. We will look at two types of loops, `for` loops and `while` loops.

Before we discuss loops lets discuss the `range` object. It is helpful to think of the range object as an ordered list. For now, let's look at the simplest case. If we would like to generate an object that contains elements ordered from 0 to 2 we simply use the following command:

```
# Use the range

range(3)
```



NOTE: While in Python 2.x it returned a list as seen in video lessons, in 3.x it returns a range object.

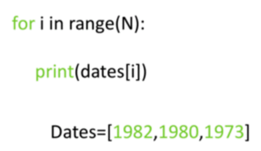
What is `for` loop?

The `for` loop enables you to execute a code block multiple times. For example, you would use this if you would like to print out every element in a list.
Let's try to use a `for` loop to print all the years presented in the list `dates`:

This can be done as follows:

```
# For loop example

dates = [1982,1980,1973]
N = len(dates)

for i in range(N):
    print(dates[i])
```

The code in the indent is executed `N` times, each time the value of `i` is increased by 1 for every execution. The statement executed is to `print` out the value in the list at index `i` as shown here:



```
for i in range(N):
    print(dates[i])
      Dates=[1982,1980,1973]
```

In this example we can print out a sequence of numbers from 0 to 7:

```
# Example of for loop

for i in range(0, 8):
    print(i)
```

In Python we can directly access the elements in the list as follows:

```
# Exmaple of for loop, loop through list

for year in dates:
    print(year)
```

For each iteration, the value of the variable `year` behaves like the value of `dates[i]` in the first example:

```
for year in dates:
    print(year)
      Dates=[1982,1980,1973]
```

We can change the elements in a list:

```
# Use for loop to change the elements in list

squares = ['red', 'yellow', 'green', 'purple', 'blue']

for i in range(0, 5):
    print("Before square ", i, 'is',  squares[i])
    squares[i] = 'white'
    print("After square ", i, 'is',  squares[i])
```

We can access the index and the elements of a list as follows:

```
# Loop through the list and iterate on both index and element value

squares=['red', 'yellow', 'green', 'purple', 'blue']

for i, square in enumerate(squares):
    print(i, square)
```

Filtering list elements using a loop

```
# Sample list
ages = [15, 18, 21, 14, 30, 17]

# Create a new list with ages above 18
adults = []
for age in ages:
    if age >= 18:
        adults.append(age)

print(adults)

```

List comprehension (a concise loop)

```
# Sample list
numbers = [1, 2, 3, 4, 5]

# Square each number using list comprehension
squared_numbers = [num ** 2 for num in numbers]

print(squared_numbers)

```

```
[1, 4, 9, 16, 25]
```

Looping through nested lists

```
# Sample nested list (matrix)
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Loop through rows
for row in matrix:
    print("Row:", row)

    # Loop through each element in the row
    for element in row:
        print(f"Element: {element}")

```

```
Row: [1, 2, 3]
Element: 1
Element: 2
Element: 3
Row: [4, 5, 6]
Element: 4
Element: 5
Element: 6
Row: [7, 8, 9]
Element: 7
Element: 8
Element: 9
```

Looping through dictionary keys

```
# Sample dictionary
fruits = {'apple': 3, 'banana': 5, 'orange': 2}

# Loop through keys
for fruit in fruits:
    print(f"Fruit: {fruit}")

```

```
Fruit: apple
Fruit: banana
Fruit: orange
```

Looping through dictionary keys and values

```
# Loop through keys and values
for fruit, quantity in fruits.items():
    print(f"{fruit.capitalize()}s: {quantity}")
```

Looping through dictionary values only

```
# Loop through values
for quantity in fruits.values():
    print(f"Quantity: {quantity}")

```

```
Quantity: 3
Quantity: 5
Quantity: 2
```

Modifying dictionary values within a loop

```
# Sample dictionary
prices = {'apple': 0.5, 'banana': 0.3, 'orange': 0.7}

# Increase each price by 10%
for fruit in prices:
    prices[fruit] *= 1.1

```

```
print(prices)

```

```
{'apple': 0.55, 'banana': 0.33, 'orange': 0.77}
```

Filtering dictionary items within a loop

```
# Sample dictionary
students_scores = {'Alice': 85, 'Bob': 92, 'Charlie': 78, 'Diana': 88}

# Filter and display students who scored above 80
for student, score in students_scores.items():
    if score > 80:
        print(f"{student} scored {score}")

```

```
Alice scored 85
Bob scored 92
Diana scored 88
```

What is `while` loop?

As you can see, the `for` loop is used for a controlled flow of repetition. However, what if we don't know when we want to stop the loop? What if we want to keep executing a code block until a certain condition is met? The `while` loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a **False** boolean value.

Here's how a while loop works:

1. First, you specify a condition that the loop will check before each iteration (repetition) of the code block.
2. If the condition is initially true, the code block is executed.
3. After executing the code block, the condition is checked again.
4. If the condition is still true, the code block is executed again.
5. Steps 3 and 4 repeat until the condition becomes false.
6. Once the condition becomes false, the loop stops, and the program continues with the next line of code after the loop.

Here's an example of a while loop that prints numbers from 1 to 5:

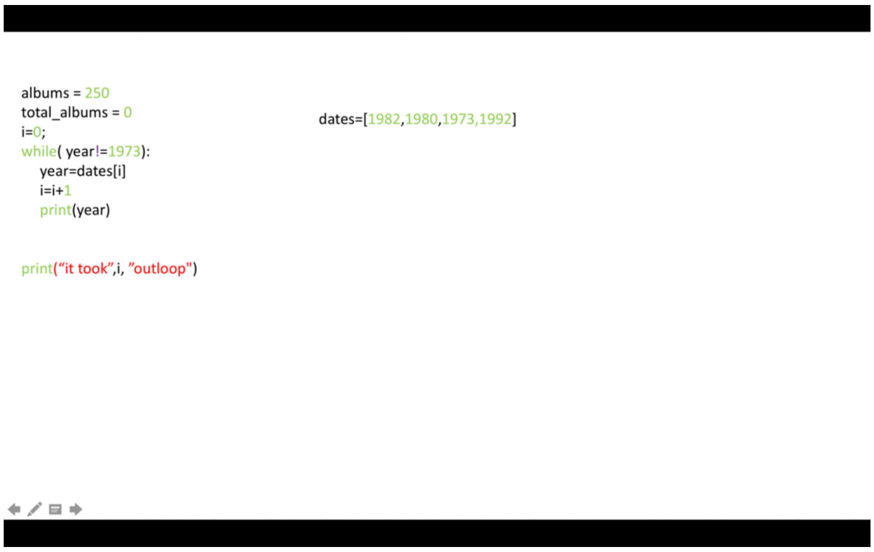
```
count = 1
while count <= 5:
    print(count)
    count += 1
```

In this example, the condition **count <= 5** is checked before each iteration. As long as count is less than or equal to 5, the code block inside the loop is executed. After each iteration, the value of count is incremented by 1 using count += 1. Once count reaches 6, the condition becomes false, and the loop stops.

Let's say we would like to iterate through list `dates` and stop at the year 1973, then print out the number of iterations. This can be done with the following block of code:

```
# While Loop Example

dates = [1982, 1980, 1973, 2000]

i = 0
year = dates[0]

while(year != 1973):
    print(year)
    i = i + 1
    year = dates[i]


print("It took ", i ,"repetitions to get out of loop.")
```

A while loop iterates merely until the condition in the argument is not met, as shown in the following figure:



```
albums = 250
total_albums = 0
i=0;
while( year!=1973):
    year=dates[i]
    i=i+1
    print(year)


print("it took",i, "outloop")
```

```
dates=[1982,1980,1973,1992]
```

The main difference between a while loop and a for loop in Python is how they control the flow of execution and handle iterations.

Key point of While Loop:

1. A while loop repeatedly executes a block of code as long as a given condition is true.
2. It does not have a fixed number of iterations but continues executing until the condition becomes false.
3. The condition is checked before each iteration, and if it's false initially, the code block is skipped entirely.
4. The condition is typically based on a variable or expression that can change during the execution of the loop.
5. It provides more flexibility in terms of controlling the loop's execution based on dynamic conditions.

Key point of For Loop:

1. A for loop iterates over a sequence (such as a list, string, or range) or any object that supports iteration.
2. It has a predefined number of iterations based on the length of the sequence or the number of items to iterate over.
3. It automatically handles the iteration and does not require maintaining a separate variable for tracking the iteration count.
4. It simplifies the code by encapsulating the iteration logic within the loop itself.
5. It is commonly used when you know the exact number of iterations or need to iterate over each item in a collection.

---

break Statement with for Loop

You can use the break statement with the for loop to terminate the loop when a certain condition is met. For example,

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

continue Statement with for Loop

You can use the continue statement with the for loop to skip the current iteration of the loop and jump to the next iteration. For example,

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Practise Exercises on Loops Lab-3

Write a `for` loop the prints out all the element between **-5** and **5** using the range function.

```
# Write your code below and press Shift+Enter to execute
for i in range(-5, 6): print(i, end=' ')
```

```
-5 -4 -3 -2 -1 0 1 2 3 4 5
```

▶ Click here for the solution

Print the elements of the following list: `Genres=[ 'rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']` Make sure you follow Python conventions.

```
# Write your code below and press Shift+Enter to execute
Genres=[ 'rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']
for gen in Genres:
    print(gen, end=' ')
```

```
rock R&B Soundtrack R&B soul pop
```

▶ Click here for the solution

Write a for loop that prints out the following list: `squares=['red', 'yellow', 'green', 'purple', 'blue']`

```
# Write your code below and press Shift+Enter to execute
squares=['red', 'yellow', 'green', 'purple', 'blue']
for sq in squares:
    print(sq, end=' ')
```

```
red yellow green purple blue
```

▶ Click here for the solution

Write a while loop to display the values of the Rating of an album playlist stored in the list `PlayListRatings`. If the score is less than 6, exit the loop. The list `PlayListRatings` is given by: `PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]`

```
# Write your code below and press Shift+Enter to execute
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]
rating = PlayListRatings[0]
i = 0
while(i < len(PlayListRatings) and rating >= 6):
    print(rating, end=' ')
    i = i + 1
    rating = PlayListRatings[i]
```

```
10 9.5 10 8 7.5
```

▶ Click here for the solution

Write a while loop to copy the strings `'orange'` of the list `squares` to the list `new_squares`. Stop and exit the loop if the value on the list is not `'orange'`:

```
# Write your code below and press Shift+Enter to execute

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []
i = 0
```

```python
while(i < len(squares) and squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1
print (new_squares)
```

```
['orange', 'orange']
```

▶ Click here for the solution

Some real-life problems!

Your little brother has just learned multiplication tables in school. Today he has learned tables of 6 and 7. Help him memorise both the tables by printing them using `for` loop.

```python
# Write your code here
print("Multiplication table of 6:")
for i in range (1, 11):
    print("6 *",i,"=",6*i)
print("Multiplication table of 7:")
for i in range (1, 11):
    print("7 *",i,"=",7*i)
```

```
Multiplication table of 6:
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
Multiplication table of 7:
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

▶ Click here for the hint

▶ Click here for the solution

The following is a list of animals in a National Zoo. `Animals = ["lion", "giraffe", "gorilla", "parrots", "crocodile","deer", "swan"]`

Your brother needs to write an essay on the animals whose names are made of 7 letters. Help him find those animals through a `while` loop and create a separate list of such animals.

```python
# Write your code here
Animals = ["lion", "giraffe", "gorilla", "parrots", "crocodile","deer", "swan"]
New = []
i=0
while i<len(Animals):
    j=Animals[i]
    if(len(j)==7):
        New.append(j)
    i=i+1
print(New)
```

```
['giraffe', 'gorilla', 'parrots']
```

▶ Click here for hint

▶ Click here for the solution

You are managing an online store's inventory. The inventory is stored in a dictionary, where the keys are the product names, and the values are the quantities in stock. Some items are out of stock (quantity is 0), and your task is to help your manager by going through the inventory and creating a new dictionary with only the items that are in stock.

Write a Python script to loop through the dictionary and create a new dictionary that contains only the items whose quantity is greater than zero.

```
#sample inventory
inventory = {
    "laptop": 5,
    "keyboard": 0,
    "mouse": 15,
    "monitor": 0,
    "printer": 7
}
```

```
#write your code here
# Initial inventory dictionary
inventory = {
    "laptop": 5,
    "keyboard": 0,
    "mouse": 15,
    "monitor": 0,
    "printer": 7
}

# Create a new dictionary for items in stock
in_stock = {}

# Loop through the inventory
for item, quantity in inventory.items():
    if quantity > 0:  # Only keep items with quantity greater than 0
        in_stock[item] = quantity

print(in_stock)
```

```
{'laptop': 5, 'mouse': 15, 'printer': 7}
```

▶ Click here for the solution

Congratulations, you have completed lab on loops

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Skills Network   IBM