Loops

Loops in Python are used to execute a block of code repeatedly for a specified number of times or until a certain condition is met. There are two main types of loops in Python: `for` loops and `while` loops.

`for` Loops

For loops in Python are used to iterate over a sequence (such as a list, tuple, or string) or other iterable object and execute a block of code repeatedly for each item in the sequence.

Here's the general syntax for a for loop in Python:

```
for item in sequence:
    # do something with item
```

The `item` variable is assigned the value of the next item in the sequence on each iteration of the loop. The loop continues until there are no items left in the sequence.

Similar to conditional statements, the body inside the `for` loop is consistently indented (typically 4 spaces) and the statement declaring the `for` loop ends with a colon (`:`).

Here's a simple example that prints the numbers 0 to 9:

```
for i in range(10):
    print(i)
```

The `range` function generates a sequence of numbers, starting from 0 and increments by 1 (by default) and stops before a specified number. In this case, `range(10)` generates a sequence of numbers from 0 to 9.

Here's an example that loops through a list of strings:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

If you need to access the index of each item in the sequence, you can use the `enumerate` function. This returns a tuple containing the index and value of each item in the sequence. Here's an example that uses `enumerate` to loop through a list of strings and print the index and value of each item:

```
fruits = ['apple', 'banana', 'cherry']
for i, fruit in enumerate(fruits):
    print(i, fruit)
```

You can also use conditional statements inside a `for` loop. Here's an example that loops through a list of strings and prints only the items that start with the letter `"a"`:

```
fruits = ['apple', 'banana', 'cherry']

for fruit in fruits:
    if fruit[0] == 'a':
        print(fruit)
```

`while` Loops

The `for` loop has its limitations, as it assumes how many items are in the sequence. If you don't know how many items are in the sequence, you can use a `while` loop instead. `while` loops in Python are used to execute a block of code repeatedly until a certain condition is met. The general syntax for a `while` loop in Python is:

```
while condition:
    # do something
```

The `condition` is evaluated before each iteration of the loop. If the condition is `True`, the code inside the loop is executed. If the condition is `False`, the code inside the loop is skipped and the program continues executing the code after the loop. For example:

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

It is essential to ensure that the `while` loop condition eventually evaluates to `False`, otherwise the loop will run indefinitely. If unchecked, this will eat up computational resources and potentially crash your system entirely. Before constructing a `while` loop, first consider whether the loop could be expressed as an equivalent `for` loop instead.

Exiting Out of Loops

It's possible to exit out of loops, i.e. before the `for` loop sequence ends, or before the `while` loop condition is met. This is done using the `break` statement. For example:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    if fruit == 'banana':
        break
    print(fruit)
```

```
count = 1
while True:
    print(count)
    if count == 5:
        break
    count += 1
```

Skipping a Loop Iteration

Alternatively, you can skip to the next iteration of the loop using the `continue` statement. For example:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    if fruit == 'banana':
        continue
    print(fruit)
```

```
count = 1
while count <= 5:
    if count == 3:
        count += 1
        continue
    print(count)
    count += 1
```

Passing Over a Loop Iteration

You can use the `pass` statement to pass over a loop iteration without executing any code. This is useful if you want to write the code for a loop later, but don't want to get an error when you run the program. For example:

```
fruits = ['apple', 'banana', 'cherry']

for fruit in fruits:
    if fruit == 'banana':
        pass
    print(fruit)
```

Nesting Loops

You may have noticed that the `for` loop body can contain any type of code, including conditional statements. We could even include an additional `for` loop inside the body of the first `for` loop. This is called nesting loops. For example:

```
colors = ['red', 'green', 'blue']
sizes = ['small', 'medium', 'large']

for color in colors:
    for size in sizes:
        print(f"{color} {size}")
```

Generally, you should avoid nesting loops if possible, as it can make your code difficult to read and debug. It can also slow down your code, as the inner loop will be executed for each iteration of the outer loop. If you find yourself nesting loops, try to see if you can rewrite your code to avoid nesting loops.

Exercises

1. Write a program to find the sum of all the numbers in a list using a for loop.
2. Write a program to print the multiplication table of a number using a for loop.
3. Write a program to print the Fibonacci sequence up to a given number using a while loop.
4. Write a program to find the largest number in a list using a for loop.
5. Write a program to check if a number is prime or not using a for loop.
6. Write a while loop to print the characters in a string, but skip the spaces.
7. Write a program to print the first n prime numbers using a while loop.
8. Write a for loop to find the index of the first occurrence of a specific word in a list of words.

9. Write a program to find the second smallest number in a list using a for loop.
10. Write a Python program that generates a multiplication table for the numbers 1 to 5 using nested for loops. The output should be in the following format:

    1 x 1 = 1
    1 x 2 = 2
    1 x 3 = 3
    1 x 4 = 4
    1 x 5 = 5
    2 x 1 = 2
    2 x 2 = 4
    ...
    5 x 5 = 25

Further Reading

Check out these resources for more information on `for` loops in Python: - W3Schools - Real Python

Also, check out these resources for more information on `while` loops in Python: - W3Schools - Real Python

Finally, check out the Python documentation for more information on `break` and `continue` statements: - Python Documentation