Python Loops

In computer programming, loops are used to repeat a block of code. We perform a process of *iteration* (repeating tasks).

There are 2 types of loops in Python:

- for loop
- while loop

Programming languages like Python implement two types of iteration:

1. *Indefinite iteration*, where the number of times the loop is executed depends on how many times a condition is met.
2. *Definite iteration*, where the number of times the loop will be executed is defined in advance (usually based on the collection size).

In a `for` loop, we will know in advance how many times the loop will need to iterate because we will be working on a collection with a predefined length.

With `for` loops, on each iteration, we will be able to perform an action on each element of the collection.

For loop syntax:

```
# Example of a for loop
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(f"I love {fruit}!")
```

Example Explanation:

1. **List**: `fruits` is a list containing `"apple"`, `"banana"`, and `"cherry"`.
2. **Loop**: The `for` loop iterates through each item in the list.
3. **Action**: During each iteration, `fruit` represents the current item, and `print` outputs a message for each fruit.

Let's break down each of these components:

1. A `for` keyword indicates the start of a `for` loop.
2. A `<temporary variable>` is used to represent the value of the element in the collection the loop is currently on.
3. An `in` keyword separates the temporary variable from the collection used for iteration.

4. A <collection> to loop over. In our examples, we will be using a list.
5. An <action> to do anything on each iteration of the loop.

For Loops: Using Range

A range is a series of values between two numeric intervals.

We use Python's built-in function range() to define a range of values.

For example,

five_steps = range(5)

five_steps is now a collection with 5 elements:

0, 1, 2, 3, 4

```
# Example of a for loop with range
for number in range(1, 6):  # Loops from 1 to 5
    print(f"Number: {number}")
```

Example Explanation:

1. **`range(1, 6)`**: Generates numbers starting from 1 up to (but not including) 6.
2. **Loop**: The `for` loop iterates over each number in this range.
3. **Action**: During each iteration, `number` represent the current value, and `print` outputs it.

While Loops

A while loop performs a set of instructions as long as a given condition is true.

While loop structure

```
# Example of a while loop
count = 1
```

```
while count <= 5:
    print(f"Count: {count}")
    count += 1  # Increment the counter
```

Explanation:

1. **Initialization**: `count = 1` sets the starting value.
2. **Condition**: `while count <= 5` keeps the loop running as long as `count` is less than or equal to 5.
3. **Action**: Inside the loop:
    - It prints the current value of `count`.
    - The `count` the variable is incremented by 1 (`count += 1`) to eventually meet the exit condition.

Loop controls: Break and continue

The **break** statement is used to terminate the loop immediately when it is encountered.

```
# Example of loop controls: break and continue
for number in range(1, 10):  # Loop through numbers 1 to 9
    if number == 5:
        print("Breaking the loop at 5")
        break  # Exit the loop when number is 5
    elif number % 2 == 0:
        print(f"Skipping {number} because it's even")
        continue  # Skip the rest of the loop body for even numbers
    print(f"Processing number: {number}")
```

Explanation:

1. **`break`**: Stops the loop entirely when `number == 5`.
2. **`continue`**: Skips the current iteration when `number` is even and moves to the next loop iteration.

Python continue Statement

The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

While the break control statement will come in handy, there are other situations where we don’t want to end the loop entirely. What if we only want to skip the current iteration of the loop?

Nested Loops

In Python, loops can be nested inside other loops. Nested loops can be used to access items of lists which are inside other lists. The item selected from the outer loop can be used as the list for the inner loop to iterate over.

Example code

```
# Example of a nested loop
for i in range(1, 4):  # Outer loop
    for j in range(1, 4):  # Inner loop
        print(f"Outer loop: {i}, Inner loop: {j}")
```

More Resources for loops

https://www.codecademy.com/resources/docs/python/loops

https://www.programiz.com/python-programming/if-elif-else

https://www.programiz.com/python-programming/for-loop

https://www.programiz.com/python-programming/while-loop

https://www.programiz.com/python-programming/break-continue