Repetition/Loops

Python, like all programming languages, allow us to repeat a set of instructions. This is known as repetition, iteration, or loops. Loops are used to repeat one or more statements up to a desired number of times. This makes writing programs more easily and efficiently.

The two types of commonly used loops in Python are:

- For Loop
- While Loop

For Loop

Syntax

```
for item in iterable:

        Code block to be executed
```

Note the colon at the end of the “for” statement

To print “hello” five times in Python without using a loop you could write the code the following way:

```
print(“hello”)

print(“hello”)

print(“hello”)

print(“hello”)

print(“hello”)
```

This is rather tedious. Instead, you can use a FOR loop to reduce the amount of typing and repetition.

```
for i in range(0, 5):

                print(“hello”)
```

This code is telling Python to do the following:

1. Start counting from 0 and stop before reaching 5
2. For each number we count, store the value in the variable i
3. Execute the block of code in the for loop
4. Repeat step 1

Examples

1. The following program asks for your name and prints out your name ten times. Here the loop variable, i, starts from 0 and goes until 9, which is 10 times in total.

```
name = input("Enter your name ")
for i in range (0, 10):

    print(name)
```

2. This program prints the numbers from 10 to 1 backwards.

```
for x in range (10, 0, -1):

    print(x)
```

3. In this example, the program outputs only the even numbers from 1 to 100. The if statement checks to see if the value in i is even by using the mod operator (any even number % 2 will result in 0). If the number is even, then print the value in i.

```
for i in range (1, 101, 1):

    if i % 2 == 0:
        print(i)
```

We could have also written the above program as follows and obtained the same result:

```
for i in range (2, 101, 2):

    print(i)
```

In this case we start the loop variable, i, with 2 and skip by 2 with every iteration, so i will go from 2 to 4, 6, 8… and so on until it reaches 100.

4. The following program finds the sum of the first 10 integers and outputs the sum. The variable sum is used to hold the value of the sum which keeps increasing as you iterate

through the loop. We first initialize sum to 0. In the first iteration, sum = 0+1 (as i is 1). In the second iteration, sum = 1+2 (i is 2), the third iteration, sum=3+3 and so forth. Note that we need to have the starting value of i as 1 and the ending value as 11, and not 10, because the for loop always stops one number short of the ending point. So, if we had used 10 as the ending value, i.e., for i in range (1, 10, 1), the highest value i would reach would have been 9.

```
sum = 0

for i in range (1, 11, 1):

    sum = sum + i
print("The sum of the first ten integers is "+ str(sum))
```

While Loop

Syntax

```
while condition:

        Code block to be executed
```

The major difference between the for loop and the while loop is that the number of iterations in the while loop does not have iterate a fixed number of times. The user input can control how many times the loop iterates.

Examples

1. The following code outputs the numbers from 1 to 5.

```
current_number = 1

while current_number < = 5:

    print(current_number)

    current_number = current_number+1
```

2. The following code results in an infinite loop because the value of the variable num never changes as the equation num = num + 1 is outside the loop. So num always stays at 1 and never gets above 5 in order for the loop to end.

```
num = 1

while num < = 5:
```

```
    print(num)

num = num+1
```

In order to correct this problem, make sure that num = num + 1 is inside the while loop.

```
num = 1

while num < = 5:

    print(num)

    num = num+1
```

3. In this example, the while loop will continue to output the message until the user enters, “quit.”

```
prompt = "Tell me something, and I will repeat it back to you. \nEnter quit
to end the program "

message = ""
while message != ‘quit’:

     print(message)

     message = input(prompt)
```

Using **break** to exit a loop. To exit a while loop immediately without executing any remaining code in the loop, use the break statement. The break statement can also be used in for loops.

4. The following program asks the user about places they have visited. We can stop the while loop in this program by calling break as soon as the user enters the ‘quit’ value.

```
prompt = "\n Please enter the name of a city you have visited.\n Enter ‘quit’
when you are finished."

while True:

    city = input(prompt)

    if city =='quit':

        break

   else:

       print(“I would love to go to “+ city.title() + “!”)
```

Using **continue** in a loop. Rather than breaking out of a loop entirely without executing the rest of the code, you can use the continue statement to return to the beginning of the loop based on the result of a conditional test.

6. The following program counts from 1 to 10 but prints only the odd numbers in that range:

```
number = 0

while number < 10:

     number = number + 1

    if number % 2 == 0:

        continue

    print(number)
```

Whenever the variable number becomes an even number, the if statement will evaluate to True causing control to return to the beginning of the loop thus skipping the print(number) statement.