Python - Iterative Executions

Sometimes its required to execute the same code block multiple times. For example, if list of students marks is given and we need to compute the total or average marks of that class. We need to iterate over all students marks one by one and need to use each of them for the operation to be performed i.e. summation or averaging. In in each iteration, the opeartion is same but the data is changing. In such situations, loop statements helps us to execute same line of code multiple times in order to perform the same opeartion again and again.

While Statement

While statement helps to execute the code multiple times based on certain logical condition. Until the condition evaluates to true, the statements within the while block are executed. Once the condition evaluates to false, then control does not enter into the while block statements and continues the execution of the code that follows after the while block.

In [ ]:

```
#iterate over the first five numbers

my_number_list = range(5)  #generate first five integers starting from zero

i = 0   # initialize the iteration variable
while i < len(my_number_list):   #evaluate condition
    print('(i)   ', i, "    (number)   ", my_number_list[i])   # while block stmt 1
    i = i + 1                                                  # while block stmt 2

print("Length of list ", len(my_number_list))
```

In above code block, the code flows like -
(1) first list of five integers is generated.
(2) iteration variable (whose value needs to be changed in every iteration, otherwise loop will never terminate) is initialized, i.e. i = 0
(3) While loop condition is evaluated to true or false.
(4) If the condition evaluated to true then statement within while loop are executed. One of the statement increments iteration variable value.
(5) Step 3 and 4 are executed until the condition evaluates to true.
(6) Once the condition evaluates to false, the next statement is executed, i.e. print statement

Earlier we have written the programs where we explicitly asked user to enter the number of times the loop needs to be executed. It restricts the user. In order to avoid that situation, we can ask user to enter a specific number to state us that he is done with the entering the input and now processing can happen.

In [ ]:

```
#Example for loop with fixed number of inputs
count = 3
for i in range(count):
    name = input("Enter the name :")
    print('You entered ', name)
```

In [ ]:

```
#Example while loop without any restrictions on number of inputs
name = ""
while name != "Quit":
    name = input("Enter the name (use 'Quit' to stop :")
    if name != "Quit":
        print('You entered ', name)
```

Infinite Loops

In [ ]:

```
#Example of infinite loop
i = 0
while i < 5 :
    print("i ", i )
print("Out of while loop")
```

What is wrong with above code snippet. It will never terminate because value of iteration variable i.e. i is never changing. It will be always zero and hence the while loop condition will be always true and loop will be executed infinitely. Hence, while using loops one needs to keep in mind that iteration variable has to be changed.

Break Statement

The break statement is use to break out of a loop before the loop is finished.

In [ ]:

```
# Example
i = 0           #iteration variable initialized
while i < 5:    # condition is evaluated
    print(i)    # block statements are executed until the condition is true i.e. 5 times
    i = i + 1
```

In [ ]:

```
# Example
i = 0           #iteration variable initialized
while i < 5:    # condition is evaluated
    if i > 3 :
        break   # no more further execution of the loop, once i is greater than 4
    print(i)    # block statements are executed until the condition is true and iteration v
    i = i + 1
```

Continue Statement

Sometimes we want to finish the execution of the current iteration and jump to the next iteration without executing the further statements in the code block, then continue statement can be used. It helps to start a new iteration.

In [ ]:

```
stmt = ""

while stmt != 'Quit':
    stmt = input("Enter the stmt (Use 'Quit' to stop )")
    if stmt.startswith("#") :
        continue
    if len(stmt) > 0 :
        print(stmt)
```

In above code snippet, user enters the input statement. She can use "Quit" word to terminate the execution of code. Once the statement is received, the code checks whether it starts with '#'. If yes, no further statements in the block are executed. It directly jumps to the next iteration. If the statement does not start with '#' then further it checks if the statement has some characters in it or not. If yes, the statement is printed back again.

For Statement

Computers are experts at doing things in repeated manner, especially without any errors. Python for loops in one of the ways by which the code blocks can be executed in the repetative manners, same as while loops. With for loops, in advance you how many times the code execution will be repeated which is not the case with while loops.

In [ ]:

```
#Example : print Hello five times
for i in range(5):
    print("Hello")
```

In the for statement, word for is in lower case followed by loop variable ending with colon. The for loop statements needs to be indendated at same level.

In [ ]:

```
#Example : Compute cube of a number three times (take three inputs)
for i in range(3):
    number = int(input("Enter the number"))
    print("the cube is " , number*number*number )
```

In [ ]:

```
#Example : print first five integers
for i in range(5):
    print(i)
```

Here we expected that the program should print numbers 1 to 5 but actually it printed 0 to 4. This is because the loop variable i.e. i is always initialized to zero and then incremented by one after each iteration. The program loops 5 times, each time increasing the value of i by 1 , until we have looped five times.

In [ ]:

```
#Example : print first five integers along with value of i
for i in range(5):
    print("(i)  ", i, "(number) " , i+1)
```

Range function

The range function helps to generate the range of numbers based on given inputs to it. The value we put in range function determines how many times the loop should be executed. range function produces a list of numbers from zero to the value minus one. For example, range(3) generates 3 values : 0, 1 and 2.

Syntax - range(start_value, end_value, step)

In [ ]:

```
#Examples

range(5)                  # produces  five values 0, 1, 2, 3, 4
range(1, 5)               # produces  four values 1, 2, 3, 4
range(2, 4)               # produces  two values 2, 3
range(1, 10, 2)           # produces values starting from zero to nine , at step of 2 i.e.
range(9, 2, -1)           # produces values starting from nine to three , at step of -1 i.e
```

In [ ]:

```
for i in range(9, 2, -1):
    print(i)
```

For loop Variations

In [ ]:

```
# iterate over values directly
for i in (11,31,51):
    print(i)
```

In [ ]:

```
# iterate fixed number of times
for i in range(3):
    print(i)
```

In [ ]:

```
# iterate fixed number of times with start and end value
for i in range(1, 5):
    print(i)
```

In [ ]:

```
# iterate fixed number of times with start and end value with gaps in between
for i in range(1, 5, 2):
    print(i)
```

In [ ]:

```
#iterate over list of values

scores = [23, 34, 12, 34, 45]
length = len(scores)

for i in range(length):
    print(scores[i])
```

In [ ]:

```
#iterate over list of values directly

scores = [23, 34, 12, 34, 45]

for score in scores:
    print(score)
```

In [ ]:

```
#iterate over list of values directly along with index
squares = [0, 1, 4, 9, 16]

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

In [ ]:

```
#iterate over strings directly

my_string = "this is demo"

for letter in my_string:
    print(letter)
```

In [ ]:

```
#iterate over tuple elements directly

my_tuple = (1, 'two', 3.4, True)

for element in my_tuple:
    print(element)
```

In [ ]:

```
#iterate over set elements directly

my_set = {1, 'two', 3.4, True}

for element in my_set:
    print(element)
```

In [ ]:

```
#iterate over dictionary elements directly

my_dict = {"key1": "value1", "key2":"value2"}

for item in my_dict.items():
    print(item)
```

In [ ]:

```
#iterate over dictionary keys directly

my_dict = {"key1": "value1", "key2":"value2"}

for key in my_dict.keys():
    print(key)
```

In [ ]:

```
#iterate over dictionary values directly

my_dict = {"key1": "value1", "key2":"value2"}

for value in my_dict.values():
    print(value)
```

Exercise

Q1. Write a program using while loop that prints square and cube of numbers between 1 to 10.

In [ ]:

```
#Try it here :
```

Q2. Write a program that uses for loop to print numbers 100, 96, 92, 88 ..., 4.

In [ ]:

```
#Try it here :
```

Q3. Use a for loop to print a triangle like the one below. Allow the user to specify how high the triangle should be.

@
@@
@@@
@@@@

In [ ]:

```
#Try it here :
```

Q4. Write a program that asks the user to enter the password. If the user enters the right password, the program should tell them they are logged in to the system. Otherwise, the program should ask them to reenter the password. The user should get only five tries to enter the password, after which point the program should tell them that they are kicked off of the system.

In [ ]:

```
#Try it here :
```

In [ ]:

```
```