4.2 Loops and Iterations

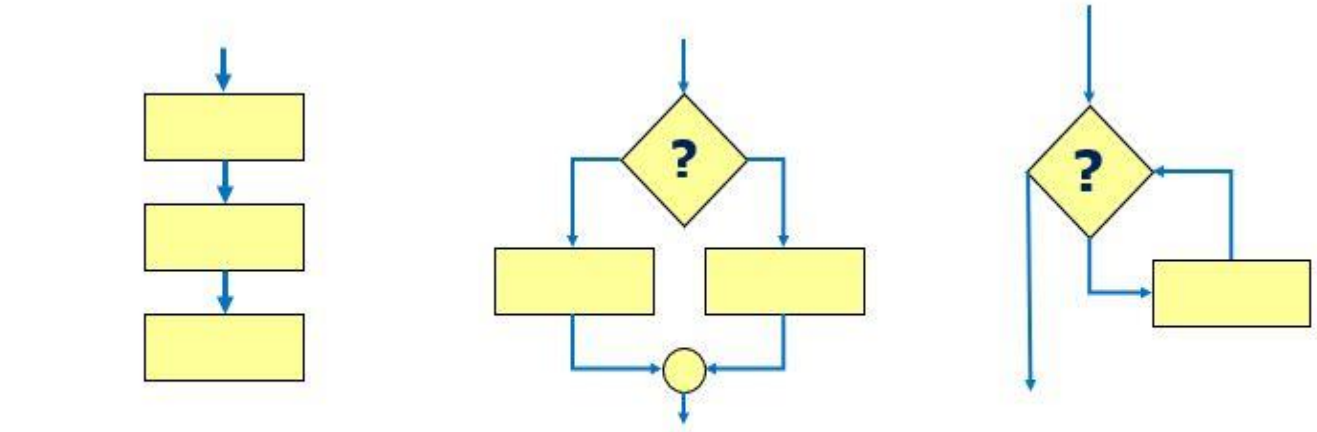
Iteration is the repeated execution of the same block of code.

Why do we need Loops and Iterations ?

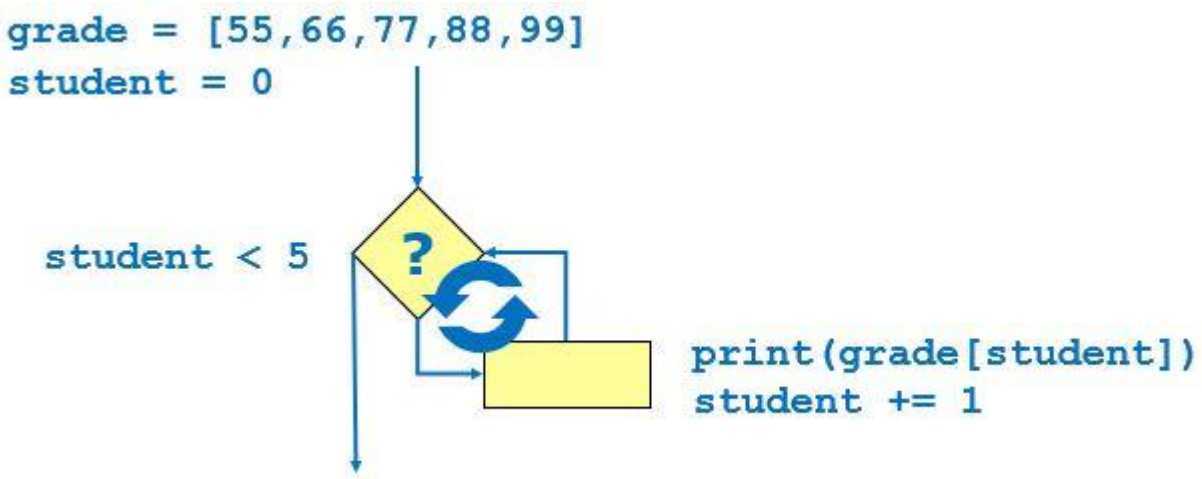
Why do we repeat? Let's say that we want to enter marks for a student and check to see if the student has passed the exam by getting more than 50 marks. We would write one line of code for one student. Now let's say thea we have 5 students and we want to do the same. One way of doing this is to copy the code written for one student 5 times.

Is this the best way of doing this? If we had a way of writing the code once yet repeat it 5 times. it would be easier to write and if necessary, change languages allow us a way to repeatedly execute code multiple times.

We already know how to write code that goes in a sequence. We also we know how to write code to check some condition and decide which way to go usingif-else and switch- case in python. We can also repeat or iterate as shown in the diagram.



Let's look at how this works. We have 5 grades in a list and want to print the grades. We start with student 1 or list item 0. We need to repeat or iterate this for all 5 students so we check to see if the student is less than 5. Then we have the print statement to print. And also we have to make sure that we increase the student number by 1 so that next time we can take the next student in the list.



Considering the nature of iterations, these can be categorized as,

1. Definite iteration, in which the number of repetitions is specified explicitly in advance

2. Indefinite iteration, in which the code block executes until some condition is met

In Python, indefinite iteration is performed with a while loop, and definite iteration is implemented with for loops.

The for Loop

In the for loop the block of code is repeated or iterated a specific number of times. The format of the code or the syntax of the code is to have the for Keyword and tell that for each iterating variable in the sequence needs to loop through the code.

```
for iterating_var in sequence:
    statements(s)
```

"For" and "in" are the two keywords. The iterating variable and the sequence are variables that we define. The sequence can be a list of numbers where each time the iterating variable will be taking an item starting from the first one until it reaches the end of the list. And we also have a colon. Don't forget the colon if you forget you will get a syntax error

The block of statements repeated is called the loop body. The lopp body needs to be indented to the right so that the computer understands the block of code belongs to this loop.

Let us look at an example in python. Play around with the code to see if you can add one more iteration to the given code example.

```
This is a loop: counter =  6
```

Range Function

It may be not practical for us to type the sequence of items that we want the iterating variable to take. So we have the range function to create the sequence of numbers we want.

One way of indicating range function is to input when to stop the sequence. So if we input 10 the sequence will have 10 numbers from 0 to 9. The sequence ends at 9 because 0 is counted as an item in the sequence.

range(stop)

```
example:
list(range(10))
output:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

If we do not want start from 0 we can indicate where we want to start the sequence. In this example we want the start variable to be 5 and the stop variable to be 10. Note that the list will go from 5 to 9. Let's create a range less than 10

range(start,stop)

```
example:
list(range(5, 10))
output:
[5, 6, 7, 8, 9]
```

If you do not want to increment by 1 then you can indicate how you want to step or increment each item.

range(start,stop,step)

```
example:
list(range(0, 10, 3))
Output:
[0, 3, 6, 9]
```

Assume that we want to write code to calculate the total of all even numbers up to 100.

First, we will set the total to need to have a place to collect the total and make it 0 so that we do not have anything in that. Then we write the for loop to go through all the numbers. First we need an iterating variable that would change during each iteration. We use counter for that. Then we need to get a sequence of even numbers generated. We get the help from range function where we start at 0, stop at 101 and step through in 2. Next we need to add the counter to the existing total. Note on this line of code the old value of total and the counter that is in the right hand side will be given to the left hand side total and it will be used in the next iteration of the loop. Finally after we finish the loop, we need a print the total.

Play around the code and try to see if you can change the given code example to add all the odd numbers beetween 0 and 100. Or the even numbers between 100 to 200, etc. More you try your willl learn. IT is hard to learn python by reading. You need to try coding!

The basic syntax of the Python while loop is,

```
while <condition>:
      <statement(s)>
```

The line/set of lines to be executed as the loop body is denoted here as <statement(s)> with the imperative indentation, just as in all python control structures. The <condition> is the loop controlling condition. It has to be declared and assigned an initial value before the start of the while loop. Its value is then modified within the loop body to keep the loop running.
When the loop strats, the <condition> is evaluated in the form of a Boolean expression, which turns out either to be true or false. The loop body is executed only if the <condition> is true. If it is evaluated to False , the loop will not be executed at all .As mentioned before, the loop controlling variable in the <expr> is modified repeatedly within the lop body and is repeatedly evaluated in each iteration before entering into the loop body.

```
num = 5
while (num !=0) :
    print ('Hello World!')
        num = num - 1
```

Output will be

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

This is how it will execute. First we make num to be 5. Then it will check the condition which is true and procced to execute the loop body. At each iteration it will reduce number by 1 and finaly when it becomes 0 it will get out of the loop and complete th code.

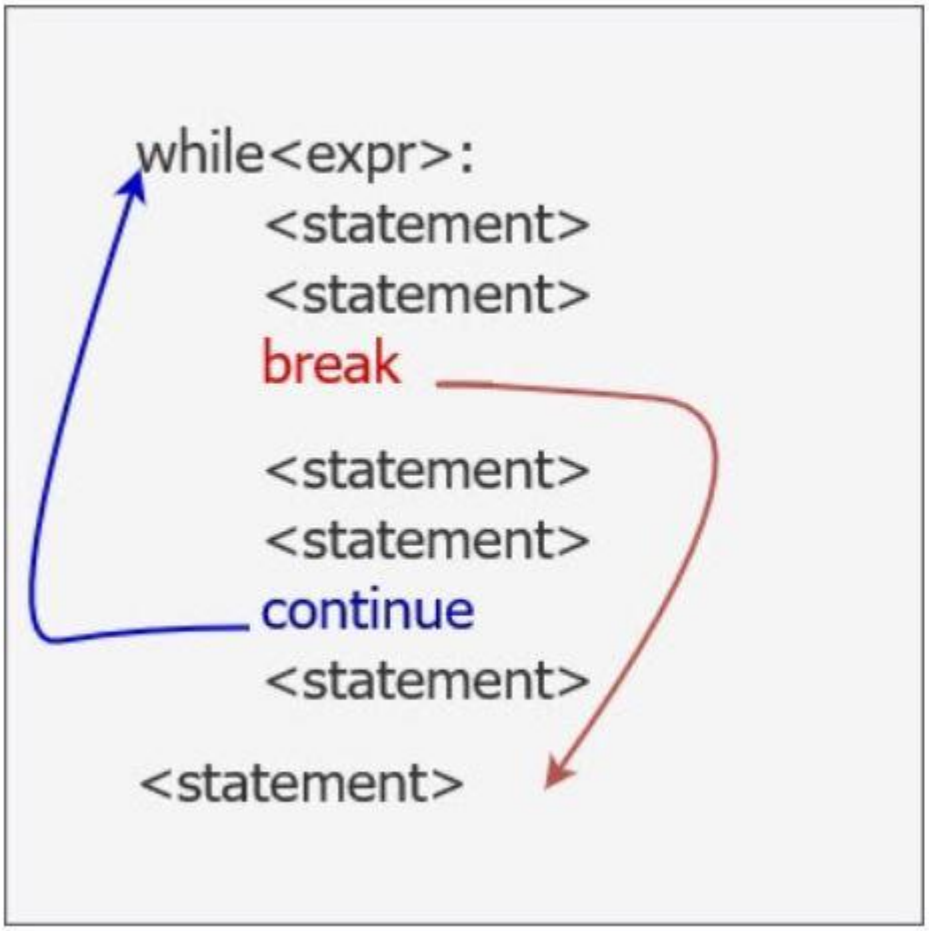
Break and Continue statements

Python allows terminating the iteration of the loop prematurely with the break and continue keywords.

With the break statement, the loop can immediately be terminated in entirety. Once a break statement is met within the loop ,the loop is exited and the program execution is passed to the first statement that appears after the loop body.

With the continue statement, the current loop iteration is immediately terminated.This statement skips the rest of the loop statement and starts the next iteration of the loop to take place.

In while loops,


while<expr>:
<statement>
<statement>
break
<statement>
<statement>
continue
<statement>
<statement>

Example Programs

break

```
n = 10
while n > 0:
    n -= 1
    if n == 5:
        break
    print(n)
```

```
9
8
7
6
>>>
```

Here, the iteration breaks out when n's value become 5.

continue

```
n = 10
while n > 0:
    n -= 1
    if n == 5:
        continue
    print(n)
```

Here , the loop skips execution of the remaining statements when n's value become 5, and resumes from

In for loops

break and continue work the same way with for loops as with while loops. break terminates the loop completely and proceeds to the first statement following the loop, and continue terminates the current iteration and proceeds to the next iteration:

break

```
for i in ['rabbit', 'deer', 'lion', 'giraffe']:
    if 'lion' in i:
        break
    print(i)
```

```
rabbit
deer
>>>
```

continue

```
for i in ['rabbit', 'deer', 'lion', 'giraffe']:
    if 'deer' in i:
        continue
    print(i)
```

```
rabbit
lion
giraffe
>>>
```

Using ‘else’ with Loops

Python allows to use an ‘else’ clause with loops:

- If the ‘else’ clause is used with a for loop, the ‘else’ clause is executed when the loop has iterating iterating the sequence given.
- If the ‘else’ clause is used with a while loop, the ‘else’ clause is executed when the loop condition becomes false.

Assuming loop is not exiting with a break.

Look at the example code. In this example with an else clause for loop will be executed until it finds the number 5. If it finds number 5 it will exit the loop with a break and after printing number found. If it does not find number 5 it will complete the loop and then go to the else clause and say that the number is not found.

```
numbers = [3,2,5,8,2,9]
for number in numbers:
    print("looking at:",number)
    if number == 5:
        print("FOUND number")
        break
else:
    print("Number NOT FOUND")

print("End")
```

Output:

```
looking at: 3
looking at: 2
looking at: 5
FOUND number
End
```

Selecting the Loop

- The 'for' loop is generally used when the number of iterations or the sequence can be identified at program compilation or before execution of loop.
- The 'while' loop can be used for any other situations.

Nested Loops

We can also have nested loops. We can put a loop inside a loop to make a nested loop. The syntax is as given where we indent the loop inside the out loop. When we execute for each iteration of the outer loop the entire inner loop will be iterated. Once it is finished it will go to the outer loop and take the next iterating variable from the outer loop and once again completely execute the inner loop iteration. We call it nesting because we put a loop inside another loop. We can have any number of loops insides loop. For example we can have a loop inside a loop inside another loop. Note if each loop is long it will take a long time to complete all the nested loops.

```
for iterating_var in sequence:
    while expression:
        statement(s)
        statements(s)
```

We can also have different combinations when nesting loops. We can have for and while as in the last example or while and for as given in this example or even while

and a while or for and a for loop. Any combination of loops are possible and any number of nesting levels are also possible.

Here is an example for a nested loop. If we look at the example we can think about printing one line of characters by using a for loop that goes from 0 to 9. Now we want to have all 5 lines. So we put another loop outside this loop to go from 0 to 4 iterating 5 times. This will print 5 lines.

```
for x in range (0, 5):
    for y in range (0, 10):
        print('$',end='')
    print('')
```

Output:

```
$$$$$$$$$$
$$$$$$$$$$
$$$$$$$$$$
$$$$$$$$$$
$$$$$$$$$$
```

The pass Keyword

Pass does not change the loop execution but it can be used to do nothing in a loop. In a loop we cannot have an empty block of code inside a loop. So we can use the pass to write some code with loop that does nothing.

In this example we have list of fruits. At the moment we do not know what to do with them but we want to have loop so that we would not forget to do something with the fruits later. If we write the code without any executable code for the loop body other than the comment it will give a syntax error.

How we can avoid that is to put a pass statement inside the loop. And later when we know what to do with the fruits inside the loop we can remove the pass and put some code there.

```
fruits = ['Apple', 'Orange', 'Grapes', 'Banana']
for fruit in fruits:
    # Not sure what to do!
print("done")
```

Syntax Error

```
fruits = ['Apple', 'Orange', 'Grapes', 'Banana']
for fruit in fruits:
    pass # Not sure what to do!
print("done")
```