Iteration

Imagine you are asked to print “Hi” 5 times. Till now what you have learned, the most simple way to do is to print “Hi” using 5 individual print statements. What if you are asked to print “Hi” 10,000 times? then using the previous way of writing individual print statements might seem unrealistic and inefficient. Using “loops or iteration” we can solve this problem very easily. Loop gives us the advantage of doing repetitive work or we can say executing a block of statements with very few lines of code.

We have two looping constructs: while and for.

A while loop is used in programming to repeatedly execute a block of code as long as a specified condition remains true. The key characteristic of a while loop is that it continues to execute the code block as long as the condition is true, and it only stops when the condition becomes false

Whether to use “while” or “for” loop?

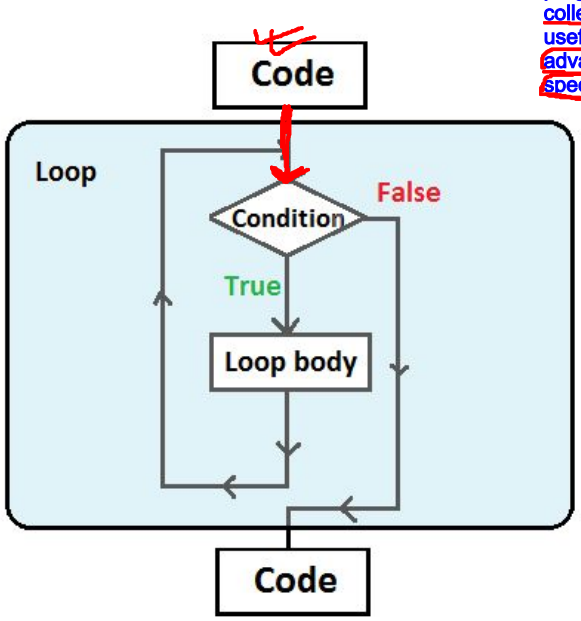
It depends on what we want to do in our program. If we want to run a block of code forever or check the condition of doing repetitive work after each iteration, then use the “while” loop. On the contrary, if we want to do repetitive work for a fixed number of times or iterate over the members/items of a sequence (string, list, range, etc.) while doing the repetitive work, then we need to use the “for” loop. Both the looping constructs have almost the same working mechanism and it is illustrated with the flowchart below.

A for loop is a fundamental control structure in programming used for iterating over a sequence, collection, or a range of values. It's particularly useful when you know the number of iterations in advance or when you need to iterate over a specific set of elements.

range(1, 5, 1)

start end dist of two

one execution of the loop's body,




```
print("Hi")
print("Hi")
print("Hi")
print("Hi")
print("Hi")
```

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

```
for number in range(5):
    print("Hi")
```

While loops:

The basic structure of while loop

`initialize loop controller`
`#codes outside while loop`
`while condition:# condition to terminate the loop`
 	
`  #codes inside while loop`
`  Repetitive work`
`  update loop controller #(to eventually terminate the loop.)`
`  #(By update, we meant any arithmetic operation)`
`  #codes inside while loop`
 	
`#codes outside while loop`

Example: This program will print "Hi" for 100 times

`count = 1`
`while count <= 100:`
`  print("Hi")`
`  count = count + 1`

Break:

What will happen if we forget to update the loop controller?

Example: This program will print "Hi" for infinite times	Output

`count = 1`
`while count <= 100:`
`  #codes inside while loop`
`  print("Hi")`
`  #codes inside while loop`
 	
`#codes outside while loop`
	`Hi`
	`Hi`
	`Hi`
	`Hi`
	`……..`
	`……..`
	`Hi`

Here the program will never terminate and it will print "Hi" for an infinite number of times (until the memory runs out). It is called "**Infinite loops**".

Now, what can we use to stop/break this cycle or loop? For this, we have a "break" statement. It can terminate the current iteration or even the whole loop without checking the "loop terminating condition".

Example: This program will print "Hi" for 5 times	Output

`count = 1`
`while count <= 100:`
`  #codes inside while loop`
`  print("Hi")`
`  if count == 5:`
`  break`
`  count = count + 1`
`  #codes inside while loop`
 	
`#codes outside while loop`
	Hi
	Hi
	Hi
	Hi
	Hi

Example: This program will print "Hi" for 5 times (Another version of the previous program)	Output

`count = 1`
`while True:`
`  #codes inside while loop`
`  print("Hi")`
`  if count == 5:`
`  break`
`  count = count + 1`
`  #codes inside while loop`
 	
`#codes outside while loop`
	Hi
	Hi
	Hi
	Hi
	Hi

Continue:

Unlike "break", "continue" does not terminate the current loop; instead, it skips the rest of the codes of the current iteration and moves on to the next iteration.

Example: Code	Output

`count = 1`
`while count <= 5:`
`  #codes inside while loop`
`  print("=====")`
`  print("Hi")`
`  count = count + 1`
 	
`  if count == 3:`
`  continue`
	=====
	Hi
	Bye
	=====
	=====
	Hi
	(Here, in the 3rd iteration after the checking skipped printing "bye" and the design)
	=====

`    print("Bye")`
`    print("=====")`
`    #codes inside while loop`
`#codes outside while loop`
	`Hi`
	`Bye`
	`=====`
	`=====`
	`Hi`
	`Bye`
	`=====`
	`=====`
	`Hi`
	`Bye`
	`=====`

Now, if I change the position of `count = count + 1` to later in code, what problems might arise due to it? Run the code by yourself and check.

Example: Code	Output

`count = 1`
`while count <= 5:`
`    #codes inside while loop`
`    print("=====")`
`    print("Hi")`
 	
`    if count == 3:`
`        continue`
 	
`    print("Bye")`
`    print("=====")`
`    count = count + 1`
`    #codes inside while loop`
 	
`#codes outside while loop`
	Check the output by running the code by yourself

Solution: During the 3rd iteration, the update of the loop controller, "counter" has been skipped by continue. So after the 3rd iteration, it prints "Hi" for an infinite time.

For loops:

In this for loop, i is initialized to 0, and the loop body is executed. After each iteration, i is automatically incremented by 1. The loop continues to run until i reaches a value of 4 (in this case) because the range(5) generates values from 0 to 4. So, i keeps track of which iteration we are in.

Tracking Progress: The loop control variable (i) serves as a simple and effective way to track the progress of the loop. You can use it to make decisions within the loop, calculate values, or access specific elements in a data structure

The basic structure of for loop:
for <iterating_var> in <sequence/ collection>: repetitive work or block of statements

A) Iteration of each list item using for loop:

Example: Code	Output

`marvel_heroes = ["Black Panther", "Captain America", "Iron Man", "Ant-Man"]`
`#codes outside for loop`
`for hero in marvel_heroes:  # by item`
`    #codes inside for loop`
`    print(hero)`
`    #codes inside for loop`
 	
`#codes outside for loop`
	`Black Panther`
	`Captain America`
	`Iron Man`
	`Ant-Man`

Explanation: Here, we are iterating through a list of strings named "marvel_heroes" and "hero" is the iterating variable or loop iterator which is going through all the elements of "marvel_heroes" sequentially from left to right. During each iteration, first checking is done to see whether any element or item of the sequence is left to execute and it is called the loop terminating condition. If the checking gives True, then the next item of the list is referred to the loop iterator, "hero" and the loop body is executed. Inside the loop body, the value of the loop iterator "hero" is being printed. If the checking is False, then the for loop terminates and codes outside the loop start executing sequentially.

B) Iteration of each character of String using for loop:

Example: Code	Output

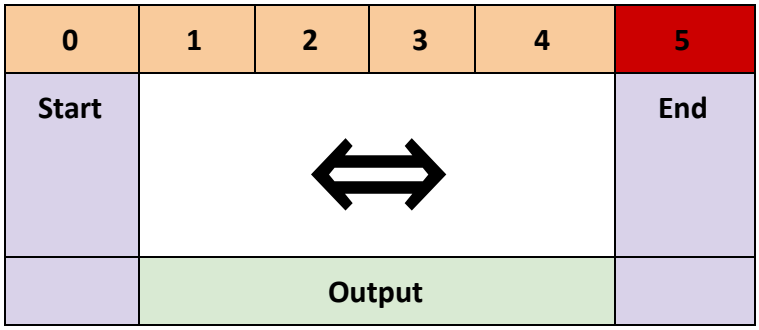
`iron_man = "Tony Stark"` i is a loop index,
`#codes outside for loop`
 	
`for achar in iron_man: # by each character`
 	
`    #codes inside for loop`
`    print(achar)`
`    #codes inside for loop`
 	
`#codes outside for loop`
	`T`
	`o`
	`n`
	`y`
	
	`S`
	`t`
	`a`
	`r`
	`k`

Explanation: Here, we are iterating through a string named "iron_man" and "achar" is the loop iterator which is going through all the characters of "iron_man" sequentially from left to right. During each iteration, first checking is done to see whether any character of the string, "iron_man" is left and it is the loop terminating condition. If the checking gives True, then the next character of the string, "iron_man" is referred to the loop iterator, "achar" and the loop body is executed. Inside the loop body, the value of the loop iterator "achar" is being printed. If the checking is False,

then the for loop terminates and codes outside the loop start executing sequentially.

C) for loop using the range function:

The range function takes 3 integers (can be positive and negative) as inputs: start, end, and step size. Among these three writing end value is mandatory and when not mentioned in the function, the start has a default value of 0, and the step size has a default value of 1. Taking these inputs, the range function returns a sequence of numbers beginning from start and going up to but not including the end.



Example: Code	Explanation	Output
`for number in range(5):` `  print(number)`	It has only an end value, 5. So by default, the start will be 0, the step size will be 1.	0, 1, 2, 3, 4.
`for number in range(2, 5):` `  print(number)`	It has start value 2 and end value, 5. So by default, the step size will be 1.	2, 3, 4.
`for number in range(0, 5, 2):` `  print(number)`	It has start value 0, end value, 5 and step size, 2	0, 2, 4.

Example: This program will print "Hi" for 5 times	
Code	Output

i is a loop index,
`for number in range(5):`
`  #codes inside for loop`
`  print("Hi")`
`  #codes inside for loop`
`#codes outside for loop`
	Hi
	Hi
	Hi
	Hi
	Hi

Explanation: Here, we are iterating using the range function with an end value of 5. So the range function returns a sequence [0, 1, 2, 3, 4] and “number” is the loop iterator which is going through all items of sequence serially from left to right.
During each iteration, the first checking is done to see whether any item is remaining and it is the loop terminating condition. If the checking gives True, then the item of the sequence is referred to the loop iterator, “number” and the loop body is executed. Inside the loop body, “Hi” is being printed. If the checking is False, then the for loop terminates and codes outside the loop start executing sequentially.

Style Guide for Python Code

For every programming language, there are few coding conventions followed by the coding community of that language. All those conventions or rules are stored in a collected document manner for the convenience of the coders, and it is called the “Style Guide” of that particular programming language. The provided link gives the style guidance for Python code comprising the standard library in the main Python distribution.

Python style guide link: https://www.python.org/dev/peps/pep-0008/