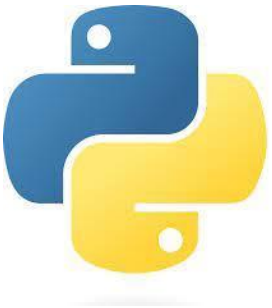
Code Clock

Tutorial 2: Loops and Lists

Time.to.code



Programming with Python

@ QUB

Loops

Loops in any programming language offer the functionality of executing a particular set of code again and again in an iterative fashion. Python supports three different kinds of loops, which are discussed below one by one in detail.

Today we will loop at two types of loop:

1. **While Loop**
2. **For loop**
3. **Nested loop**

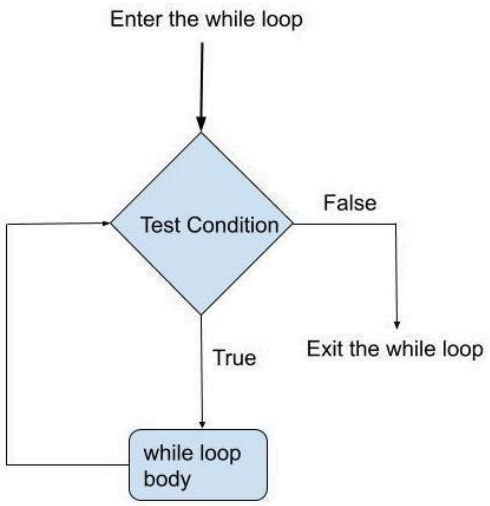
WHILE LOOP

A while loop is sued to iterate a particular set of statements multiple times until the test condition returns a Fals value. It is generally used in situations in which we do not know how many times we need to iterate a particular code. If the test condition returns a True value every time, the loop runs indefinitely

Syntax:

while (conditional expression):

Statement to be executed



The following code shows a counter being used to control the number of iterations of a while loop. The counter is set to 1 before the loop. This counter is used as part of the condition to determine if the loop runs. Once inside the loop the counter is incremented by 1 (counter+1). When the counter become greater than 5, the while condition will return false and the loop will terminate.

```
counter=1

while(counter<=5):
    print(2*counter)
    #increments counter by 1 each iteration of the loop
    counter=counter+1
    print("The value of the counter is " + counter)

print("The loop has now terminated")
```

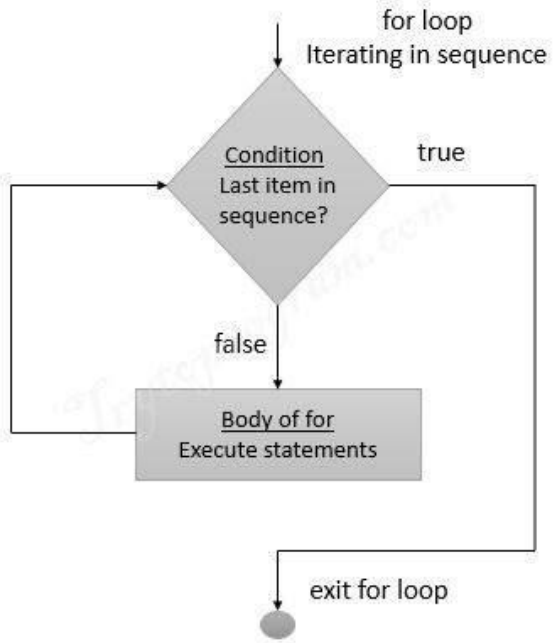
Challenge: Using a While Loop print the following:

1. All numbers from 1 to 10
2. All numbers from 10 to 1
3. All even numbers from 1 to 10
4. All odd numbers from 1 to 10
5. All numbers exactly divisible by 4 between 1 and 40 inclusive
6. All even numbers between 6 and 30
7. Adds the value of a counter which increments by 1 during each iteration to a variable **total** until the **total** exceeds 100.

FOR LOOP

To iterate over sequence-type objects e.g. lists, strings etc we use the for loop.

A for loop starts iterating from the beginning of a sequence and traverses until the last element in the sequence. This is known as sequence traversal. The advantage of using a foor loop over a while loop is that a counter variable does not need to be setup before the loop starts. However, we need a separate iterating variable to iterate over our sequence



Syntax:

for iteration_variable in sequence:

Statement to be executed

Using the range() Function in a for loop

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Enter the following code which will print the numbers 0 to 5.

```
for x in range(6):
  print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter:

Enter the following code: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
  print(x)
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

Enter the following code: increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

The break Statement

With the **break** statement we can stop the loop before it has looped through all the items:

Enter the following code which will exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Print all items by referring to their index number:

Enter the following code to iterate through and print the contents of `thislist`

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Challenge: Using a For Loop print the following:

1. All numbers from 1 to 10
2. All numbers from 10 to 1
3. All even numbers from 1 to 10
4. All odd numbers from 1 to 10
5. All numbers exactly divisible by 4 between 1 and 40 inclusive
6. All even numbers between 6 and 30
7. Adds the value of a counter which increments by 1 during each iteration to a variable **total** until the **total** exceeds 100.
8. Runs a maximum of ten times each time asking the user to enter a value which is added to the value of x. When this combined value exceeds 20, the loop terminates printing the following statements “Combined total exceeds 20”.

Looping/Traversing over a String using for loop:

A string is also known as a sequence data type which means you can iterate over the elements within the string using a for loop.

The for loop starts traversing the string from the beginning and goes on until the last element in the sequence is traversed.

Type in the following example to see the results of a traversal:

text= “Time to Code”

for each_letter in text

print(“The character reached during traversal is: “, each_letter)

```
text= "Time to Code"
for each_letter in text:
  print("The character reached during traversal is: " + each_letter)
```

Beginning at the letter “T”, each time through the loop, the for loop moves through each letter printing the letter during the each iteration. The loop will terminate when the last letter is reached.

Challenge: Using a For Loop write the code which will:

1. Print out each letter of a word inputted by the user
2. Count the number of letters in a word entered by a user.
3. Count the number of vowels in a word entered by an user.
4. Count the number of non-vowels in a word entered by an user

Lists

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Enter the following code which creates a List entitled `thislist`

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Enter the following code which creates a List of fruits containing duplicates

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

To determine how many items a list has, use the len() function:

Enter the following code which creates a List containing three items and counts the number of items in the list.

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

List Items - Data Types

List items can be of any data type: String, Int and Boolean

Enter the following code which creates three lists each storing three different data types: a List entitled `thislist`

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

A list can also contain different data types:

Enter the following code which creates a list with strings, integers and Boolean values

```
list1 = ["abc", 34, True, 40, "male"]
```

Access Items

List items are indexed and you can access them by referring to the index number:

Negative Indexing

Negative indexing means start from the end

`-1` refers to the last item, `-2` refers to the second last item etc.

Enter the following code to print the last item of the list

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Enter the following code to return the 3rd, 4th and 5th items.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

By leaving out the start value, the range will start at the first item:

Enter the following code to return the items from the beginning to, but not including, "kiwi"

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Enter the following code to return the items from "cherry" to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Enter the following code to check if "apple" is present in the list

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Append Items

To add an item to the end of the list, use the `append()` method:

Enter the following code to to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Insert Items

To insert a list item at a specified index, use the `insert()` method.

The `insert()` method inserts an item at the specified index:

Enter the following code to insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Extend List

To append elements from *another list* to the current list, use the extend() method.

Enter the following code to add the elements of `tropical` to `thislist`:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Remove Specified Item

The `remove()` method removes the specified item.

Enter the following code to remove "banana" from the list

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Remove Specified Index

The `pop()` method removes the specified index.

Enter the following code to remove the second item

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

If you do not specify the index, the `pop()` method removes the last item

Enter the following code to remove the last item

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

The `del` keyword also removes the specified index:

Enter the following code to remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The `del` keyword can also delete the list completely.

Enter the following code which will delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Enter the following code which will clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Challenge: Using a List print the following:

1. Create a list called weekDays populating with the first three days of the working week.
2. Add the remaining two days of the working week to this list
3. Create another list called weekend and populate it with weekend days.
4. Extend weekDays with the weekEnd list
5. Print the number of days in weekDays.
6. Remove the first item in weekDays and reprint the number of remaining days in this list
7. Check if Monday is listed in the list weekDays and output a suitable statement to confirm its absence or not.
8. Clear the weekDays list and confirm the length is now 0.

Extension: Lottery

Your program must generate 6 random numbers between 1 and 20.
The program must ask the user for 6 numbers between 1 and 20 after which the program will print these out in numerical order.
The should then cross reference the user numbers with the lottery numbers to determine how many matches and in the event of 3+ matches print out the following:
- 3 Matching balls- "You've won £10"
- 4 Matching balls- "You've won £100"
- 5 Matching balls- "You've won £1000"
- 6 Matching balls- "You've won £1,000,000"

Once this particular game is over the user must be choose to either play the game again OR return to the top-level menu from where they can choose other games.