IIT Madras
BSc Degree

Home Lesson-5.2

Lesson-5.1

Lists

Introduction

A list in Python is a data structure that is used to store a sequence of objects. Some examples are given below:

```
numbers = [1, 2, 3, 4, 5]
letters = ['a', 'b', 'c', 'd']
words = ['this', 'is', 'a', 'list']
```

- Lists can be printed, just like the other types we have seen so far. `print(numbers)` will give the following output:

```
[1, 2, 3, 4, 5]
```

- Lists could contain objects of different types. Python permits lists such as this:

```
mixture = [1, 1.0, '1', True]
```

- Lists have a separate type - `list`. We can also check if a given variable holds an object of type `list`:

```
numbers = [1, 2, 3]
print(type(numbers))
print(isinstance(numbers, list))
```

- The `len` function can be used to find the number of elements in a list:

```
numbers = [1, 2, 3]
print(f'This list has {len(numbers)} elements in it')
```

- Lists support indexing and slicing. These two operations work exactly the same way as they did for strings:

```
numbers = [1, 2, 3, 4]
print(numbers[0], numbers[1], numbers[2], numbers[3])
print(numbers[1 : 3])
print(numbers[-2])
```

Iterating through lists

As a list is a sequence, we can iterate through it using `for`. This is one of the primary uses of the `for` loop:

```
# Method-1
numbers = [1, 2, 3, 4]
for num in numbers:
    print(num)
```

The loop variable — `num` — picks one item at a time from the sequence. In the body of the loop, we are just printing this item. We can rewrite the code given above using a `while` loop:

```
# Method-2
numbers = [1, 2, 3, 4]
index = 0
while index < len(numbers):
    print(numbers[index])
    index += 1
```

Finally, we can also use the `for` loop to iterate through the indices of the list. For this, we take the help of the `range` function.

```
# Method-3
numbers = [1, 2, 3, 4]
for index in range(len(numbers)):
    print(numbers[index])
```

In the example given above, `len(numbers)` is equal to `4`. So, the `range` sequence will be `0, 1, 2, 3`. `index` is the loop variable that iterates through this sequence.

Methods 2 and 3 are very similar. Both iterate through the sequence of indices, and use list indexing to access the corresponding element in the list. The only difference is that method-2 uses `while`, while method-3 uses `for`. Method-1 stands out from the other two as it directly pulls elements from the sequence.

Growing a list

Lists are typically used in problems where we wish to store a collection of items. Usually, we start with an empty list. Python provides two ways to create an empty list:

```
list1 = []
list2 = list()
```

Both `list1` and `list2` are empty lists. The interpreter doesn't mind spaces between the opening and closing braces, so `list1 = [ ]` also works. Given an empty list, how do we add items to it? Python provides two ways to do this:

```
list1 = list1 + [1]
print(list1)
list2 = list2.append(1)
print(list2)
```

Both lists end up having just the one element. The first method is called **list concatenation**, i.e., two lists are being concatenated or combined together. Treat concatenation like joining two compartments of a train together. It is very similar to string concatenation. The second way uses a method called `append` that is essentially a function defined for the `list` type. Append adds elements at the end of the list.

Consider the following problem:

> Generate the list of positive integers less than 100 that are divisible by 3.

There are at least two ways of doing this. The first one uses `while`:

```
# Method-1
num = 3
nums_div = []
while num < 100:
    nums_div.append(num)
    num += 3
```

The next method uses `for`:

```
# Method-2
nums_div = []
for num in range(3, 100, 3):
    nums_div.append(num)
```

Operations on Lists

We have already seen how the `+` operator works with lists:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list12 = list1 + list2
print(list12)
list21 = list2 + list1
print(list21)
```

This gives the concatenated output:

```
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 1, 2, 3]
```

The order matters when two lists are being concatenated! The next is the `*` operator:

```
list1 = [0] * 5
print(list1)
list2 = [1, 2, 3] * 3
print(list2)
```

This replicates the list. The following is the output:

```
[0, 0, 0, 0, 0]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Two lists are equal if they have the same sequence of elements:

```
l1 = [1, 2, 3]
l2 = [1, 2, 3]
l3 = [3, 2, 1]
print(l1 == l2)
print(l2 == l3)
```

This results in:

```
True
False
```

Finally, two lists can be compared with the `>` or the `<` operator. List comparison works very similar to string comparison, in that it uses lexicographic ordering. We looked at this in the first chapter:

> **Lexicographic ordering**
>
> First element from both lists are compared. If they differ this determines the outcome of the comparison. If they are equal, then the second element of both the lists are compared. This process continues until either list is exhausted.

Some example comparisons:

```
print([1, 2] < [2, 1])
print([1] < [1, 2, 3])
print([2, 3, 4] < [3])
print([] < [1])
```

All four of them result in `True`.

Useful Functions

Let us look at some built-in functions that operate on lists:

- `sum`: this is used to find the sum of the elements in a list of numbers:

```
a = [1, 2, 3]
print(sum(a))
```

- `max` and `min`: these two functions find the maximum and minimum value in a list respectively.

```
a = [1, 2, 3]
print(min(a), max(a))
```

What happens if `a` is a list of strings? What would `max(a)` and `min(a)` produce?

- `sorted`: this function returns a sorted list

```
a = [2, 1, 3]
print(sorted(a))
```

We have come across the `range` object and seen how useful it was in iterating through a sequence. So far `range` has been associated with the `for` loop. Its time has come to break out of the loopy prison:

```
numbers = range(10)
print(numbers)
```

This gives `range(0, 10)` as an output. This is a sequence that we can iterate over. Python provides a way of turning this object into a list:

```
numbers = list(range(10))
print(numbers)
```

This gives `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]` as the output.