Week 5: Lists and `for` loops

Intended learning outcomes

By the end of this class, you will be able to:

- write for loops to perform repeated commands;
- create, modify, iterate over, and pick slices of lists;
- test if something is a member of a list.

Remember that when you see a line of code with >>> you should type in the code and run it immediately. When you see a block of code without >>> you should type in all the code before running it.

1 The `for` loop

We have seen how we can repeat a piece of code using the `while` loop. There is another looping structure in Python called the `for` loop. There are two ways to use the `for` loop: one is very similar to things we have done with the `while` loop, the other involves lists and we will look at it after we have looked at lists.

×	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9
4	4	8	12

We are going to write a program to draw a multiplication table (something like the picture) and we will do this in steps. Guess what the following program will do, then type it in and run it.

```
n = 2
print("Here are multiples of", n)
for i in range(10):
    print(i * n)
```

Change `range(10)` to `range(6)` and run the program again. Are these the numbers you were expecting? Remember that Python starts counting at zero. The command `range(m)` will include all the numbers from zero up to, but not including, `m`.

Now change `range(6)` to `range(2, 6)` and run the program again.

In general, using `range(start, stop)`, where `start` and `stop` are integers, will give you a loop from the value `start` up to *but not including* the value `stop`. This might seem a little odd, but it has certain advantages, such as there being `m` numbers in the range `range(start, start + m)`.

Thinking mathematically, the range `range(start, stop)` is like a half-open interval $[a, b)$, which contains a and doesn't contain b. Like the convention of counting from zero, this can cause confusion initially, but makes sense when you get used to it.

This makes things easier than using a `while` loop when you are just counting. You don't need to type the following in, but you can replace the following four lines with the two lines

```
count = start
while count < stop:
    print(count)
    count = count + 1
```

→

```
for count in range(start, stop):
    print(count)
```

Returning to our program above, we would like to put all of the multiples on a single line. Remember from last time (look back if you need to) that the `print()` command moves you onto a new line, which is why all of our multiples are on different lines. We can change this by using the `end=` construction (see last week's sheet). Change the print line in your program to

```
    print(i * n, end="\t")
```

Run the program again and observe that the output appears on a single line.

Now we have a single line of multiples of `n`, we can loop over different values of `n`. Modify your code so that it now looks like this:

```
for n in range(1, 10):
    for i in range(1, 6):
        print(i * n, end="\t")
    # At the end of each row we need to start a new line
    print()
```

Run the code. Modify the code so there are 15 rows and 7 columns in your table. What do you think will happen if you remove the `print()` line? Try it.

Exercise 5.1. Now you will write a very similar program that will print out a triangle of numbers, like in the picture in the margin. First write a program that inputs an integer `n` and uses a `for` loop to print the numbers 1 to `n` in a row with no gaps between them, like `1234`. Now modify the program so that `n` varies in a `for` loop from 1 to 5, so that you get a triangle of height 5 like in the picture. (It should be similar to the program above.)

```
1
12
123
1234
12345
```

You can also use a third parameter in the `range` command to give something of the form `range(start, stop, step)`. In this case you go from `start` up to, but not including, `stop` in jumps of `step`.

```
>>> for i in range(0, 101, 10):
...     print("i =", i, " and  i**2 =", i**2)
```

2 The list type: `list`

So far we have used the Python types string, floating point number, integer and boolean. Another Python type is the list type.

```
>>> menu = ["spam", "eggs", "beans", "sausage"]
>>> type(menu)
```

A list consists of an ordered list of expressions. Those expressions can be of any type. The above list consists of four strings. We can pick out individual elements of the list or sublists using *slicing* as we did with strings. Before typing in the following, *write down* what you expect each print statement to give you.

```
>>> print(menu[1])
>>> print(menu[-1])
```

Remember that Python starts counting at zero!

Not only can we pick out individual elements in a list, but we can also pick out **slices** of a list. The syntax for a slice is `my_list[start:stop]`, where `start` and `stop` are integers: this gives the list of all elements from index `start` up to *but not including* index `stop`. So the slice `my_list[3:5]` would be the list `[my_list[3], my_list[4]]`. This is just like the form of the `range()` command, but here you use a colon rather than a comma.

What will the following slices be?

```
>>> menu[2:4]
>>> menu[2:3]
>>> menu[0:3]
```

If you want your slice to be "from the beginning" or "to the end" you can just leave out either `start` or `stop` as appropriate.

```
>>> menu[:2]
>>> menu[2:]
```

Just as with strings, you can get the length of lists and concatenate them.

```
>>> print(len(menu))
>>> menu = menu + ["spam", "tomato", "spam"]
>>> print(menu, len(menu))
```

Note that we can have something occurring multiple times in a list.

We change items in a list explicitly. We can make our sausage vegetarian by observing that it's fourth in the list, so will have index 3, then doing an explicit assignment of element three in the list.

```
>>> menu[3] = "vegetarian sausage"
>>> print(menu)
```

There are various so-called *methods* that can be used with lists. (We will see more methods in future weeks.) Here are some methods you can use; the term `list` will be replaced by the name of your list in general.

- `list.append(x)` will add `x` onto the end of your list;
- `list.remove(x)` will remove the first instance of `x` from your list;
- `list.index(x)` will return the index of the first instance of `x` in your list;
- `list.count(x)` will return the number of times that `x` occurs in your list;
- `list.insert(i, x)` will insert `x` into your list at index i and will shift everything along to make room for it;
- `list.reverse()` will reverse the list.

Here these methods are in context. As usual, write down what you think will happen after each command before typing it in.

```
>>> menu.remove("spam")
>>> print(menu)
>>> print(menu.index("spam"))
>>> print(menu.count("spam"))
>>> menu.insert(2, "fried bread")
>>> print(menu)
>>> menu.reverse()
>>> print(menu)
```

All the methods for the `list` type can be found by googling.

You can test if something is a member of a list using the `in` command. Run the following program.

```
city = input("Please enter a UK capital city: ")
if city in ["Cardiff", "Edinburgh", "London", "Belfast"]:
    print("Correct.")
else:
    print("That is not a capital city.")
```

Note that you will get an error if you try to remove an item that is not in a list. You can avoid the error by checking, if necessary, that the item you want to remove is actually a member of the list.

```
>>> numbers = [1, 2, 3]
>>> numbers.remove(4)
>>> if 4 in numbers:
...     numbers.remove(4)
```

You can obtain a list of numbers such as `[0, 1, 2, 3, 4]` by using the `list` command to convert a range into a list.

```
>>> list(range(5))
```

Exercise 5.2. Obtain these lists using the `list()` and `range()` commands:

- `[1, 2, 3]`;
- `[2, 4, 6, 8]`;
- `[-3, -8, -13, -18]`.

3 Looping over lists

As well as using `for` loops to run through a range of numbers, you can use them to run through the elements of a list. Here is a simple program to type in and save. (What does `sep=""` do?)

```
1 CARD_SUITS = ["clubs", "spades", "diamonds", "hearts"]
2 for suit in CARD_SUITS:
3     print("One suit is ", suit, ".", sep="")
```

The program will loop through the indented code, in this case just line 3, each time taking `suit` to be one of the items in the list `CARD_SUITS`, in order. In this example we seem to be breaking our convention about variable names having lower case letters, but the list of suits in a deck of cards is a **constant** — we never want it to change.

Style convention 1. In Python, constants are given names with capital letters and underscores between words, for example `MONTHS_OF_YEAR`.

Exercise 5.3. Write a Python program that defines a list consisting of the days of the week — `DAYS_OF_WEEK = ["Monday", "Tuesday", ...]` — and uses a `for` loop to tell you the first letter of each day, so that the output starts as follows:

```
The first letter of Monday is M.
The first letter of Tuesday is T.
```

Homework

1. Finish off this sheet.

2. (*Quick review.*) Answer these without using a computer, then type them in to check your answer.

   - What is the output of the following: `list(range(5)),list(range(10,13))` and `list(range(2, 0, -1))`?
   - What does the following program do? What would you call the output (see MAS114)?

     ```
     TRUTH_VALUES = [False, True]
     print("a\tb\ta and b")
     print("-------------------------")
     for a in TRUTH_VALUES:
         for b in TRUTH_VALUES:
             print(a, b, a and b, sep="\t")
     ```

3. (*Assessed homework.*) This week's homework involves writing a program implementing the **Sieve of Eratosthenes**, a method of giving a list of all the prime numbers up to a given number. To access the homework, you must navigate to the following webpage:

   `https://aim.shef.ac.uk/moodle/mod/quiz/view.php?id=426`

   Your code will be run with some test strings. If it gives the correct answer you will get a mark, if it does not then you will get no mark. You can submit your homework anytime from now til 2pm on Monday of Week 6 (1st November).

4. (*Optional.*) There are four integers greater than one which are equal to the sum of the cubes of their digits. Write a program to find them.

   [Hint: A `for` loop can be used to run through the characters in a string: e.g. `for character in "Hello"`.]