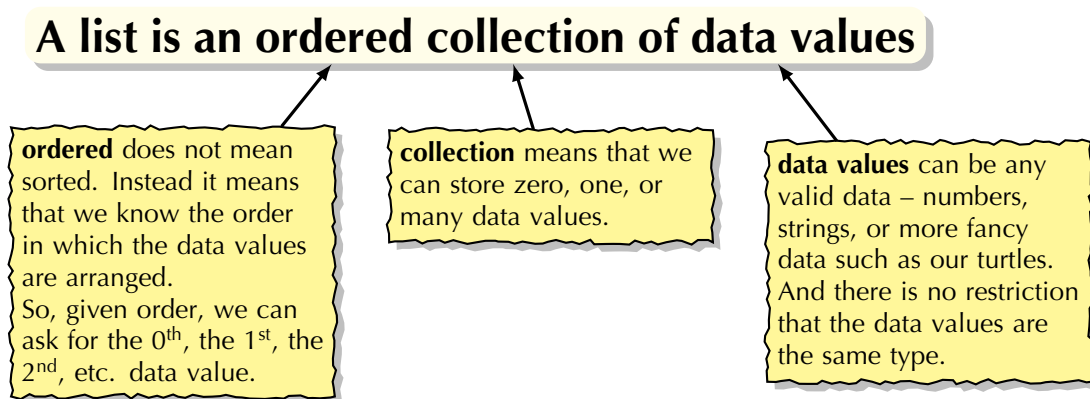


# Working with Lists I

## Introduction

This week we are going to something different. We are going to take a closer look at python lists. We have used lists before, in our graphical *Take Away* game to store the messages that we wanted to display, but we need to look at lists more carefully if we are going to develop more complicated programmes/games.

## So what is a list?



## How To Create a List?

Creating a list is very simple. You simply put your data values inside square brackets, and separate each value with a comma. So, for example,

- `list1 = [0, 1, 2, 3, 4, 5]`  
Creates a list storing the integers from 0 to 5 inclusive.
- `list2 = ["My", "name", "is", "bob"]`  
Lists can store any data type — the above list stores 4 strings.
- `list3 = [107, 0.4, "hello", [2,3]]`

Lists can store a mixture of data types — including other lists! The above list stores four data values each of a different type:

- 107 is an integer in python terminology this is an **int**.
- 0.4 is a floating point number since it has a decimal point in python terminology this is a **float**.
- "hello" is a string in python terminology this is a **str**.
- [2,3] is a list containing two integers.



## How big is a List ?

To determine how many data values are in a list we use the command `len`. For example the code

```
list3 = [107, 0.4, "hello", [2,3]]
print(len(list3))
```

outputs 4 because the list contains four data values — an integer, a float, a string, and a list.

## How To Access Data Stored in a List ?

We have three ways to get at the data that is stored in a list

- **indexing** — count up positions from the left, starting at zero.
- **negative indexing** — count down positions from the right, starting at negative one.
- **slicing** — given a pair of indexes, extract a sub list from a list.

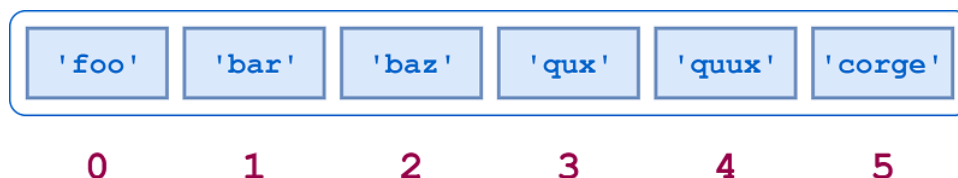
Each method is covered in the following sections.

### Indexing

When we create a list we give it a name — I used `list1`, `list2`, etc in the previous page, but you can come up with much better names than that<sup>1</sup>. Using the list name and square brackets<sup>2</sup> we can access any value in the list, counting from the left and starting with zero. For example, the code

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

creates the list



Then

- `a[0]` refers to the string `'foo'`.
- `a[1]` refers to the string `'bar'`.
- `a[2]` refers to the string `'baz'`.
- `a[3]` refers to the string `'qux'`.
- `a[4]` refers to the string `'quux'`.
- `a[5]` refers to the string `'corge'`.

The number in the square brackets is called the **index** and it needs to be an integer.

So something like `a["2"]` will not work.

If the index does not refer to a valid position in the list a **IndexError** will occur.

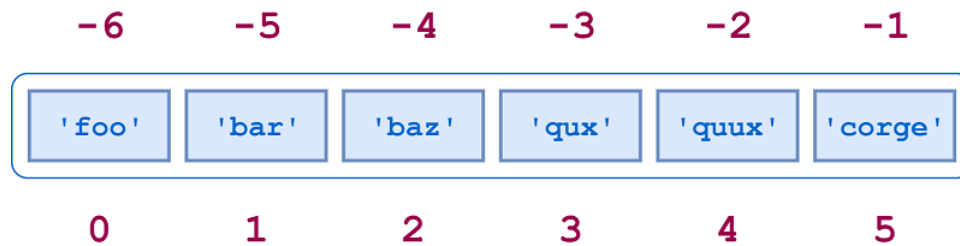
<sup>1</sup>In programming we put a lot of effort in to generating sensible, informative names for our variables. It makes our code much easier to read.

<sup>2</sup>square brackets `[]` are called **index operator**.



## Negative Indexing

Python has this cool feature — if we pass a negative integer as an index then it will start counting from the right, starting with negative 1.



So

- `a[-1]` refers to the string `'corge'`.
- `a[-2]` refers to the string `'quux'`.
- $\vdots$
- `a[-6]` refers to the string `'foo'`.

## Slicing

If we put two integers inside the square brackets, as in `[m:n]` we don't just get a single data value back — we get a sub-list back, of all data values from position `m` up to but not including position `n`.

If either, or both `m` and `n` can be missing then the sub-list extends to start or end of the original list.

For example, given the code

```
a = ['well', 'good', 'google', 'book', 'red', 'blue']
```

then

- `a[2:5]` returns sub-list `['google', 'book', 'red']`  
Returned list was a sub-list of the original list, starting with data value at index 2 and up to but not including data value at index 5.
- `a[2:]` returns sub-list `['google', 'book', 'red', 'blue']`  
(Note `n` was missing.)  
Returned list was a sub-list of the original list, starting with data value at index 2 and up to the end of original list.
- `a[:5]` returns sub-list `['well', 'good', 'google', 'book', 'red']`  
(Note `m` was missing.)  
Returned list was a sub-list of the original list, starting at start of original list and up to but not including data value at index 5.
- `a[:]` returns sub-list `['well', 'good', 'google', 'book', 'red', 'blue']`  
(Note `m` was missing.)  
Returned list was a sub-list of the original list, starting at start of original list and ending at end, i.e., the entire list.



## Programming Tasks

-  Open a new file and type in the following code. We will use this as a starting point for the following exercises.

`list_tasks_start.py`


```

1 import random as rnd
2
3 # generate some random data to play with
4 rnd.seed(42)
5 data = rnd.choices(range(100), k=10)
6
7 print("Generated data is")
8 print(data)


```

Generated data is


[63, 2, 27, 22, 73, 67, 89, 8, 42, 2]

-  Save your `list_tasks_start.py` as `list_tasks_1.py` and add code to print out the data value at index 5.

67


-  Save your `list_tasks_start.py` as `list_tasks_2.py` and add code to print out the data values, with one value per line — you need a for loop.

63  
2  
27  
22  
73  
67  
89  
8  
42  
2


-  Save your `list_tasks_start.py` as `list_tasks_3.py` and add code to print out the data values, with one value per line but in reverse order.

2  
42  
8  
89  
67  
73  
22  
27  
2  
63




-  Save your `list_tasks_start.py` as `list_tasks_4.py` and add code to print out the maximum data value in the list.


89

-  Save your `list_tasks_start.py` as `list_tasks_5.py` and add code to print out the minimum data value in the list.


2

-  Save your `list_tasks_start.py` as `list_tasks_6.py` and add code to print out the difference between the maximum value and the minimum data value in the list.


87

-  Save your `list_tasks_start.py` as `list_tasks_7.py` and add code to print out the index of the maximum value in the list.  
If the maximum value appears more than once, then print out index of first occurrence.


6

-  Save your `list_tasks_start.py` as `list_tasks_8.py` and add code to print out the index of the minimum value in the list.  
If the minimum value appears more than once, then print out index of first occurrence.


0

-  Save your `list_tasks_start.py` as `list_tasks_9.py` and add code to print out the total of the data values in the list.

395

-  Save your `list_tasks_start.py` as `list_tasks_10.py` and add code to print out the total of all of the even data values minus the total of the odd data values in the list.

-243

-  Save your `list_tasks_start.py` as `list_tasks_11.py` and add code to print out the second largest data value in the list.

73

Hints:


- Use `for k in range(len(data)):`  
To have a loop variable, `k`, that runs over the indexes needed to access all the data values in list `data`.
- Use `for d in data:`  
To have a loop variable, `d`, that runs over the data values in the list `data`.
- Use `for k,d in enumerate(data):`  
To have two loop variables, `k` and `d`, that runs over the indexes and the data values in the list `data`.




## Harder Programming Tasks

Some extra problems, again using the same generated dataset:

**[63, 2, 27, 22, 73, 67, 89, 8, 42, 2]**


-  Save your `list_tasks_start.py` as `list_tasks_extra_1.py` and add code to print out the data value of the number that appears to the left of the maximum value in the list. If the maximum value appears more than once, then print out index of first occurrence. If maximum value appears in index 0 then return `None`

67

-  Save your `list_tasks_start.py` as `list_tasks_extra_2.py` and add code to print out length of the longest sublist of increasing data values in the list.

For example, sub-lists `[2,27]`, `[22,73]` are both sub-list with increasing data values.

2

-  Save your `list_tasks_start.py` as `list_tasks_extra_3.py` and add code to print out the longest sublist of increasing data values in the list **where you can skip over data values that result in decreasing value**.

For example, sub-list `[2,22,73]`, has increasing data values, and is of length 3. It was obtained by starting at 2 but skipping over data value 27.

[2,22,67,89]