CS319 Week 06 : Review of some Python list basics

In this notebook we will revise a little about working with lists in `Python`. This was skimmed in class.

Storage types in `Python`

In C++ we use arrays to store collections of values of the same type.

In Python, there are several ways of doing this:

- **Lists**: a list is a mutable (= changeable) ordered collection. Duplicates are allowed.
- **Tuples**: a tuple is an immutable ordered collection. Duplicates are allowed.
- **Sets**: a (Python) set is a mutable unordered collection without duplicates.
- **Dictionaries**: a dictionary is an unordered but indexed mutable collection without duplicate indices.

But today we only care about `list` s and, later `numpy` arrays.

Lists

- **A `list` is a sequence of values**. But, whereas a string is a sequence of characters, a list can be a sequence of any type.
- The values in a list are called *items* or *elements*.
- The list starts with `[`, ends with `]`, and items are separated by commas.

We'll remind ourselves of...

- What a list is
- How to create one
- Modifying items in a list
- Indexing and slicing
- How to traverse a list with a `for` loop
- Some functions/methods for operating on a list

Making lists

Making a list with `[` and `]`

The simplest way to create a list is using square backets, `[` and `]`, with a comma between elements.

In [1]:
```python
[0, 1, 2, 3, 4] # a list of integers
```

Out[1]:
```
[0, 1, 2, 3, 4]
```

In [2]:
```python
['zero', 'one', 'two', 'three'] # a list of strings
```

```
Out[2]: ['zero', 'one', 'two', 'three']
```

Usually, we assign a variable name to the list.

```
In [3]: my_favourite_numbers = [0, 4, 1024]
```

```
In [4]: print(f"{my_favourite_numbers}")
```

```
[0, 4, 1024]
```

```
In [5]: print(f"The list has {len(my_favourite_numbers)} items")
```

```
The list has 3 items
```

More about lists in `Python`

The next parts are not so important for today: you can skip to the section on `numpy` , if you like

A list with different types of items

An item in a list can be just about anything. And items in a list can be of different types from each other. This list includes strings, an integer, a float, and a boolean.

```
In [6]: mixed_list = ["CS319", "Scientific Computing", 22, 78.5, True]
        print(mixed_list)
```

```
['CS319', 'Scientific Computing', 22, 78.5, True]
```

```
In [7]: type(mixed_list)
```

```
Out[7]: list
```

```
In [8]: type(mixed_list[2])
```

```
Out[8]: int
```

You can even make a list with includes another list as an item. This is called *nesting*

```
In [9]: code         = "CS319" # string
        instance     = "3BS2"  # string
        num_students = 22      # int
        ave_grade    = 87.5    # float
        has_exam     = False;  # boolean
        modules = ["CS319", "MA378", "CS211", "MA385"]  # list

        new_list = [code, instance, num_students, ave_grade, has_exam, modules ]
        print(new_list)
```

```
['CS319', '3BS2', 22, 87.5, False, ['CS319', 'MA378', 'CS211', 'MA385']]
```

```
In [10]: new_list[5][0]
```

```
Out[10]: 'CS319'
```

`len()`

There are many operations that can be preformed on lists. One of the most important is counting the number of items in a list. In Python, the `len()` function can be applied to various types, including `list` s.

```
In [11]: modules_in_3BS2 = ["CS300", "MP311", "MA322", "ST333", 'CS344']
         number_of_modules = len(modules_in_3BS2)
         print(f"You can choose from {number_of_modules} modules in 3BS2")
```

```
You can choose from 5 modules in 3BS2
```

Making a list with `list()`

One can also use the function `list()` to create a new list from an object, such as a `range` of numbers, or a `string` .

```
In [12]: list_of_numbers = list(range(10))
         print(list_of_numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [13]: list("hello")
```

```
Out[13]: ['h', 'e', 'l', 'l', 'o']
```

Indexing

- Lists are indexed from zero
- the first element is `L[0]` , the second is `L[1]` , etc.
- The last is `L[-1]` which is the same as `L[len(L)-1]` .

```
In [14]: modules_in_3BS2
```

```
Out[14]: ['CS300', 'MP311', 'MA322', 'ST333', 'CS344']
```

```
In [15]: modules_in_3BS2[0]  # first item
```

```
Out[15]: 'CS300'
```

```
In [16]: modules_in_3BS2[-1]  # last item
```

```
Out[16]: 'CS344'
```

Slicing

We create a sub-list from a string using the " `colon` " notation. The syntax is `L[a:b]` which is the same as `[L[a], L[a+1], ..., L[b-1]]`

```
In [17]: print(modules_in_3BS2)
```

```
['CS300', 'MP311', 'MA322', 'ST333', 'CS344']
```

```
In [18]: print(modules_in_3BS2[0:3])  # first 3
```

```
['CS300', 'MP311', 'MA322']
```

```
In [19]: print(modules_in_3BS2[1:4])  # middle 3
```

```
['MP311', 'MA322', 'ST333']
```

In [20]:
```
print(modules_in_3BS2[2:5])  # last 3
```
```
['MA322', 'ST333', 'CS344']
```

We can also use the notation `a:b:s` which means `start at a, go to b in steps of s`.

In [21]:
```
print(modules_in_3BS2[0:5:2])  # every second one
```
```
['CS300', 'MA322', 'CS344']
```

If you leave out either `a` or `b` they are assumed to be the start and end of the list, if `s` is positive.

In [22]:
```
print(modules_in_3BS2[::2])  # every second one
```
```
['CS300', 'MA322', 'CS344']
```

More list functions

- Add a new item to the end

In [23]:
```
modules_in_3BS2 = modules_in_3BS2 + ["DS555"]
print(f"The modules in 3BS2 are : {modules_in_3BS2}")
```
```
The modules in 3BS2 are : ['CS300', 'MP311', 'MA322', 'ST333', 'CS344', 'DS555']
```

- Similarly `L.extend(list2)` adds the items in `list2` to the end of `L`.
- `L.append("new item")` adds a single new item to a list.
- `L.insert(<position>, <item>)` adds a new `<item>` to the specified `<position>`. Anything to the right of that position is shuffled right.
- `L.remove(entry)` removes the specific entry from a list (error if it is no present)
- Use `del` to remove an entry by index.

Taversing a list with a `for` loop

`for` loops work well with lists. One form of the `for` statement is

```
for <var> in <list>:
    <body>
```
It consists of a **heading** and a **body** The heading, between the keyword `for` and the colon (`:`) introduces a **loop variable** `<var>` and refers to a list `<list>`. The `<body>` is a consistently indented sequence of statements.

When executed, a `for` statement results in the execution of its `<body>` of statements once for each item of the list (in order). The particular item for each iteration is stored as the value of the variable `<var>`.

In [24]:
```
numbers = [10, 3, -1, -11, 12]
numbers
```
Out[24]:
```
[10, 3, -1, -11, 12]
```

In [25]:
```
for item in numbers:
    print(item, end="--**--")
```

```
10--**--3--**---1--**---11--**--12--**--
```

Often we want to change items in the list. In the following example, we'll take a list of 3BS2 modules, and add a prefix of `2324-` to each code. For that we'll need to keep count of the items

There are lots more we could cover on lists, Map/Filter/Reduce, the `in` operator, adding and multiplying lists, sorting, etc...

But they are not so relevant for CS319. So we'll skip (for now).