Data Structure - List

Objectives:

- List Basics
- List Indexing
- List Slicing
- Working with List
- Modifying List

1. List Basics

Lists are the most commonly used data structure in Python.

- It is a **mutable** collection, i.e. its items can be added and removed.
- Each of these data can be accessed by calling it's index value.

Lists are declared/created by just equating a variable to `[ ]` or list.

In [10]:
```
1 s1 = []
2 print(type(s1))
```

```
<class 'list'>
```

Items in the list are seperated by comma `,` .

In [4]:
```
1 nums = [1,2,3,4]
2 nums
```

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

Length of the list can be found by built-in function `len()` .

In [9]:
```
1 len(nums)
```

Out[9]: `4`

Mixed Data Type

List is able to hold elements of mixed data types, although this is not commonly used.

In [11]:
```
mixed = ['apple', 1, 2.3, True]
mixed
```

Out[11]: ['apple', 3, 5.0, True]

Nested List

List can also have lists as its element, which creates a nested list.

In [13]:
```
Y = [ [10, 11, 12, 13], [20, 21, 22, 23]]
Y
```

Out[13]: [[10, 11, 12, 13], [20, 21, 22, 23]]

Question: What's the size of above nested list?

2. How to access an item in list? Indexing

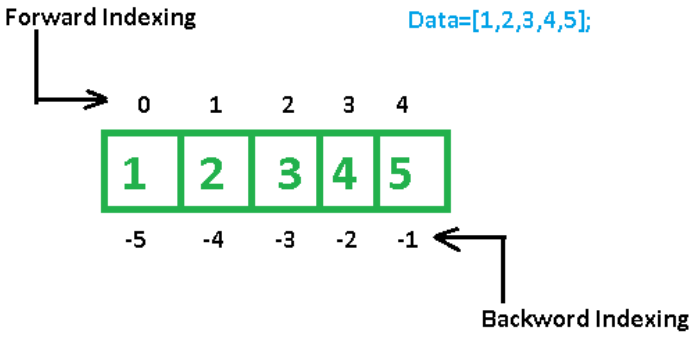
Items in collection can be accessed by their indexes. Python uses zero-based indexing, i.e. index starts from 0.

In [16]:
```
s = ['a', 'b', 'c', 'd']
print(s[0], s[1])
```

```
a b
```

Negative Indexing

Indexing can also be done in reverse order, where the last element has an index of -1, and second last element has index of -2.



In [17]:
```
print(s)
print(s[-1], s[-2])
```

```
['a', 'b', 'c', 'd']
d c
```

Multi-level Indexing

For nested list, we can access items by multi-level indexing. Each level of the index always starts from 0.

Exercise: In a nested-list `[[10, 11, 12, 13], [20, 21, 22, 23]]` , access its 1st element in 1st sub-list, and 2nd element in 2nd sub-list.

In [19]:
```
nested = [[10, 11, 12, 13], [20, 21, 22, 23]]
print(nested[0][0], nested[1][1])
```

```
10 21
```

Question:

How do you access element `Blackcurrant` in following list?

```
nested_fruits = [
    ['Apple', 'Apricots', 'Avocado'],
    ['Banana', 'Blackcurrant', 'Blueberries'],
    ['Cherries', 'Cranberries', 'Custard-Apple']]
# YOUR CODE HERE
```

In [20]:
```
nested_fruits = [
    ['Apple', 'Apricots', 'Avocado'],
    ['Banana', 'Blackcurrant', 'Blueberries'],
    ['Cherries', 'Cranberries', 'Custard-Apple']]
print(nested_fruits[1])
print(nested_fruits[1][1])
```

```
['Banana', 'Blackcurrant', 'Blueberries']
Blackcurrant
```

3. How to access subset of items? Slicing

Indexing was only limited to accessing a single element. **Slicing** on the other hand is accessing a sequence of data inside the list.

Slicing is done by defining the index values of the `first element` and the `last element` from the parent list that is required in the sliced list.

```
sub = num[a : b]
sub = num[a : ]
sub = num[: b]
sub = num[:]
```

- if both `a` and `b` are specified, `a` is the first index, `b` is the **last index + 1**.
- if `b` is omitted, it will slice till last element.
- if `a` is omitted, it will starts from first element.

- if neither `a` or `b` is specified, it is effectively copy the whole list

Note: the upper bound index is NOT inclusive!

Example:

- Create a list contain number 0-9
- Print 3rd to 5th items
- Print all items at and after 6th position
- Print first 5 items

In [22]:
```
num = list(range(10))
print(num)
print(num[2:5])
print(num[5:])
print(num[:5])
```

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

Exercise:

The `num` is a list of integers from 0 to 9, split the list into 2 equal size sub list, `sub1` and `sub2` .

In [23]:
```
num = list(range(10))
n = len(num) // 2
print(n)
sub1 = num[:n]
sub2 = num[n:]
print(sub1, sub2)
```

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

Slice with Negative Index

Remember list items can be accessed using `negative index` . Same technique can be applied for slicing too.

Exercise: For a list with integer 0-9,

- How to get last 3 items from a list?
- How to ignore last 3 items from a list?
- How to strip first and last items from a list?

In [24]:
```
num = [0,1,2,3,4,5,6,7,8,9]
# Get last 3 elements
print(num[-3:])
# Ignore last 3 items
print(num[:-3])
# Strip 1st and last element
print(num[1:-1])
```

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

4. Working with List

Min, Max and Sum

If the list consists of all integer elements, the **min( )**, **max( )** and **sum()** gives the minimum item, maximum itme and total sum value of the list.

Exercise: For a list with integers 0 - 9, use `format()` function of string to print out following message.

```
min = 0, max = 9, sum = 45
```

In [27]:
```
num = list(range(10))
s = 'min = {}, max = {}, sum = {}'.format(min(num), max(num), sum(num))
print(s)
```

```
min = 0, max = 9, sum = 45
```

Question:

What is the output of `min()` and `max()` on a list which contains string values?

Sorted

Built-in function `sorted()` can be used to sort a list in ascending order.

Exercise: Check out the documentation of the `sorted()` function

In [34]:
```
sorted?
```

For **descending** order, specify the named argument `reverse = True` .

- By default the reverse condition will be `False` for reverse. Hence changing it to True would arrange the elements in descending order.

Exercise: Sort following list in ascending order, and then in descending order

```
words = ['have','a','good','day']
```

In [35]:
```
words = 'Have a good day'.split()
x = sorted(words)
print(x)

y = sorted(words, reverse=True)
print(y)
```

```
['Have', 'a', 'day', 'good']
['good', 'day', 'a', 'Have']
```

Question: How can `key` parameter of `sorted()` function be used?

5. Membership and Searching

You might need to check if a particular item is in a list.

Instead of using `for` loop to iterate over the list and use the if condition, Python provides a simple **`in`** statement to check membership of an item.

Questions:

Write code to find out whether `duck` and `dog` are in the list `['duck', 'chicken', 'goose']` respectively.

In [23]:
```
s = ['duck', 'chicken', 'goose']
print('duck' in s, 'dog' in s)
```

```
True False
```

count()

It is used to count the occurence of a particular item in a list.

Question:

- Create a list `['duck', 'chicken', 'goose', 'duck', 'chicken', 'goose', 'duck', 'chicken', 'goose']` from `['duck', 'chicken', 'goose']`
- Count number of occurence of `'duck'`

In [24]:
```
1 s = ['duck', 'chicken', 'goose'] * 3
2 s.count('duck')
```

Out[24]: 3

index( )

It is used to find the index value of a particular item.

- If there are multiple items of the same value, only the first index value of that item is returned.
- You can add 2nd argument `x` to start searching from index `x` onwards.

Note: the string functions `find()` and `rfind()` are not available for list.

In [25]:
```
1 s = ['duck', 'chicken', 'goose'] * 3
2 s.index('chicken')
```

Out[25]: 1

6. Modifying List

List is a **mutable** collection, i.e. list can be modified and item value can be updated.

Update an Item

It is easy to update an item in the list by its index value.

Question:

For a list `s = [0,1,2,3,4]` , set item with index `2` to value `9` .

In [36]:
```
1 s = [0,1,2,3,4]
2 s[2] = 9
3 print(s)
```

```
[0, 1, 9, 3, 4]
```

Append an Item to List

The **append( )** is used to append a element at the end of the list.

Question:

For a list `s = [0,1,2,3,4,5]` , append a value `6` to it.

In [27]:
```
1 s = [0,1,2,3,4,5]
2 s.append(6)
3 s
```

Out[27]: [0, 1, 2, 3, 4, 5, 6]

Remove Item by Index

The **list.pop( )** function remove the last element in the list. This is similar to the operation of a stack.

Question:

Use `list.pop()` function to remove items in list `[0,1,2,3,4]` in reverse order.

In [31]:
```
1 s = [0,1,2,3,4]
2 s.pop()
3 print(s)
```

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

Index value can be specified to pop a ceratin element corresponding to that index value.

Check out documentation using `list.pop?`

In [32]:
```
1 list.pop?
```

Question:

Use `list.pop()` functiont to remove `'c'` from list `['a','b','c','d','e']` .

In [33]:
```
1 s = ['a','b','c','d','e']
2 # s.pop(2)
3 s.pop(2)
4 print(s)
```

```
['a', 'b', 'd', 'e']
```