1. Lists & tuples

1 Lists

```
list_1 = [1, 2, 3, 4]
```

- List is an ordered but changeable collection of any types as its elements (even functions, classes, and modules);
- List are equivalent to a cell array in MATLAB;
- Lists are **MUTABLE** i.e. elements can be added, deleted, shifted, and moved around at will i.e. it is dynamic.

1.1 Operators

in

```
2 in list_1
> True
```

not in

```
99 not in list_1
> True
```

del list[i]

```
del list_1[3]
> [1, 2, 3, 4]
> [1, 2, 3]
```

+

```
list_1 + [9, 0]
> [1, 2, 3, 9, 0]
```

*

```
list_1 * 3
> [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

1.2 Functions

max(), min()

```
max(list_1)
> 3
min(list_1)
> 1
```

len()

```
len(list_1)
> 3
```

sum()

```
sum(list_1)
> 6
```

list(): type transform

```
list('1 2 3')
> ['1', ' ', '2', ' ', '3']
```

enumerate(): for-loop with indices

```
a = ['a', 'b', 'c'] # define a list
for ind, items in enumerate(a, start = 0):
        print(ind, items)
> 0 a
  1 b
  2 c
```

1.3 zip function

zip(p,q,...) - built-in function, ZIP **iterables into a list of tuples**.

```
zip('abc', [3,4,5]) —> [('a', 3), ('b', 4), ('c', 5)]
zip('abc', [3,4,5], 'def')  —> [('a', 3, 'd'), ('b', 4, 'e'), ('c', 5, 'f')]
```

1.4 Methods

.append(arg)

```
list_1.append([100,101])

> [1, 2, 3]
> [1, 2, 3, [100, 101]]
```

.extend(list_arg)

```
list_2 = [11, 12, 13]
list_1 = [1, [33, 3], 2, 3, [100, 101]]
list_1.extend(list_2)

> [1, 2, 3, [100, 101]]
> [1, 2, 3, [100, 101], 11, 12, 13]
```

.pop(i)

- **Return** and **pop out** an element by its positional INDEX;
- Without index it **removes the last element** from a list.

```
list_1.pop()

> [1, 11, 12, 13]
> [1, 11, 12]
```

.insert(i, elmt)

```
list_1.insert(1, [33,3])

> [1, 2, 3]
> [1, [33, 3], 2, 3]
```

`.copy()`

Return a shallow copy of the list. Equivalent to a[:].

`.remove(elmt)`

Deletes a first occurrence of the argument. If it cannot find anything to remove, it will raise an error. You can catch this error by using exception-handling OR check the presence of an object to be removed in a list before attempting to remove it.

```
list_1.remove(11)

> [1, 2, 3, 11, 12]
> [1, 2, 3, 12]
```

`.index(elmt [,start [,end]])`

Returns the **index of arg** [within specified slice]. Error if no occurrence.

```
list_1.index(12)

> [1, [33, 3], 2, 3, [100, 101], 12]
> 5
```

`.sort([reverse=True])`

Sorts the comparable objects. This method changes the original list. If we want **a sorted copy with untackled original list** use:

`.sorted()`

`.count(elmt)`

Counts how many of arg are in list. Returns 0 if no occurrence.

```
list_1.count(3)
> [1, 2, 3]
> 1
```

2 Tuples

```
tuple_1 = (1, 2, 3, 4)
```

Tuple is an **fixed and ordered** collection of objects of any type.
Identical to lists in all respects, except hat tuples **IMMUTABLE**.
Why use a tuple instead of a list?

- Program execution is faster when manipulating a tuple than it is for the equivalent list;
- Sometimes data should not be modified. If the elements of the collection should remain constant, using a tuple instead of a list guards against accidental modification;
- There is a dictionary data type in Python, which requires as one of its components of an immutable type. A tuple can be used for this purpose.

There is a dictionary data type in Python, which requires as one of its components of an immutable type. A tuple can be used for this purpose.

A singletone tuple

```
tuple_sing = (1,)
```

Python allows to omit parentheses

```
t = 1, 2, 3
t = 2,
x1, x2, x3 = 4, 5, 6

> (1, 2, 3)
> (2,)
> (4, 5, 6)
```

Compound assignment

```
(s1, s2, s3, s4) = ('foo', 'bar', 'baz', 'qux')
```

2.1 Methods

Only 2 methods: `.count()`, `.index()`.

`.count(elmt)`
Counts how many of elements are in list. Returns 0 if no occurrence.

```
tuple_1.count(3)

> (1, 2, 3, 4)
> 1
```

`.index(elmt [,start [,end]])`
Returns the index of an element [within specified slice]. Returns error if no occurrence.

```
tuple_1.index(3)
> 2
```

2.2 Functions

Functions **same as for lists**. Type conversion: `tuple()` and `list()`