Lists

List is a collection data type which is ordered and mutable(can change). Unlike Sets, Lists allow duplicate elements. They are useful for preserving a sequence of data and further iterating over it. Lists are created with square brackets.

```
my_list = ["banana", "cherry", "apple"]
```

Comparison of basic built-in collection data types in Python:

- List is a collection which is ordered and mutable. Allows duplicate members.
- Tuple is a collection which is ordered and immutable. Allows duplicate members.
- Set is a collection which is unordered and unindexed. No duplicate members.
- Dictionary is a collection which is unordered, mutable and indexed. No duplicate members.
- Strings are immutable sequences of Unicode code points.

Creating a list

Lists are created with square brackets or the built-in list function.

```
list_1 = ["banana", "cherry", "apple"]
print(list_1)

# Or create an empty list with the list function
list_2 = list()
print(list_2)

# Lists allow different data types
list_3 = [5, True, "apple"]
print(list_3)

# Lists allow duplicates
list_4 = [0, 0, 1, 1]
print(list_4)
```

```
['banana', 'cherry', 'apple']
[]
[5, True, 'apple']
[0, 0, 1, 1]
```

Access elements

You access the list items by referring to the index number. Note that the indices start at 0.

```
item = list_1[0]
print(item)
```

```python
# You can also use negative indexing, e.g -1 refers to the last item,
# -2 to the second last item, and so on
item = list_1[-1]
print(item)
```

```
banana
apple
```

Change items

Just refer to the index number and assign a new value.

```python
# Lists can be altered after their creation
list_1[2] = "lemon"
print(list_1)
```

```
['banana', 'cherry', 'lemon']
```

Useful methods

Have a look at the Python Documentation to see all list methods:
https://docs.python.org/3/tutorial/datastructures.html

```python
my_list = ["banana", "cherry", "apple"]

# len() : get the number of elements in a list
print("Length:", len(my_list))

# append() : adds an element to the end of the list
my_list.append("orange")

# insert() : adds an element at the specified position
my_list.insert(1, "blueberry")
print(my_list)

# pop() : removes and returns the item at the given position, default
is the last item
item = my_list.pop()
print("Popped item: ", item)

# remove() : removes an item from the list
my_list.remove("cherry") # Value error if not in the list
print(my_list)

# clear() : removes all items from the list
my_list.clear()
print(my_list)

# reverse() : reverse the items
my_list = ["banana", "cherry", "apple"]
```

```
my_list.reverse()
print('Reversed: ', my_list)

# sort() : sort items in ascending order
my_list.sort()
print('Sorted: ', my_list)
```

sort() modifies the list in place and does not return anything.

sorted() returns a new sorted list and does not change the original list

```
# use sorted() to get a new list, and leave the original unaffected.
# sorted() works on any iterable type, not just lists
my_list = ["banana", "cherry", "apple"]
new_list = sorted(my_list)
print("sorted() fn: ",new_list)

# create list with repeated elements
list_with_zeros = [0] * 5
print(list_with_zeros)

# concatenation
list_concat = list_with_zeros + my_list
print(list_concat)

# convert string to list
string_to_list = list('Hello')
print(string_to_list)
```

```
Length: 3
['banana', 'blueberry', 'cherry', 'apple', 'orange']
Popped item:  orange
['banana', 'blueberry', 'apple']
[]
Reversed:  ['apple', 'cherry', 'banana']
Sorted:  ['apple', 'banana', 'cherry']
sorted() fn:  ['apple', 'banana', 'cherry']
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 'banana', 'cherry', 'apple']
['H', 'e', 'l', 'l', 'o']
```

Copy a list

Be careful when copying references.

```
list_org = ["banana", "cherry", "apple"]

# this just copies the reference to the list, so be careful
list_copy = list_org
print("list_org : ",list_org)
```

```
print("list_copy : ",list_copy)
# now modifying the copy also affects the original
list_copy.append(True)
print("copy append: ",list_copy)
print("copy org append : ",list_org)

# use copy(), or list(x) to actually copy the list
# slicing also works: list_copy = list_org[:]
list_org = ["banana", "cherry", "apple"]

list_copy = list_org.copy()
# list_copy = list(list_org)
# list_copy = list_org[:]

# now modifying the copy does not affect the original
list_copy.append(True)
print("copy : ",list_copy)
print("org : ",list_org)
```

```
list_org :  ['banana', 'cherry', 'apple']
list_copy :  ['banana', 'cherry', 'apple']
copy append:  ['banana', 'cherry', 'apple', True]
copy org append :  ['banana', 'cherry', 'apple', True]
copy :  ['banana', 'cherry', 'apple', True]
org :  ['banana', 'cherry', 'apple']
```

Iterating

```
# Iterating over a list by using a for in loop
for i in list_1:
    print(i)
```

```
banana
cherry
lemon
```

Check if an item exists

```
if "banana" in list_1:
    print("yes")
else:
    print("no")
```

```
yes
```

Slicing

Access sub parts of the list wih the use of colon (:), just as with strings.

```
# a[start:stop:step], default step is 1
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = a[1:3] # Note that the last index is not included
print(b)
b = a[2:] # until the end
print(b)
b = a[:3] # from beginning
print(b)
a[0:3] = [0] # replace sub-parts, you need an iterable here
print("change to 0: ",a)
b = a[::2] # start to end with every second item
print(b)
a = a[::-1] # reverse the list with a negative step:
print(a)
b = a[:] # copy a list with slicing
print(b)
```

```
[2, 3]
[3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3]
change to 0:  [0, 4, 5, 6, 7, 8, 9, 10]
[0, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 0]
[10, 9, 8, 7, 6, 5, 4, 0]
```

List comprehension

A elegant and fast way to create a new list from an existing list.

List comprehension consists of an expression followed by a for statement inside square brackets.

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
b = [i * i for i in a] # squares each element
print(b)
print(a)
```

```
[1, 4, 9, 16, 25, 36, 49, 64]
[1, 2, 3, 4, 5, 6, 7, 8]
```

Nested lists

Lists can contain other lists (or other container types).

```
a = [[1, 2], [3, 4]]
print(a)
print(a[0])
```

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