Creating Lists

The `list` class defines lists. To create a list, you can use `list`'s constructor, as follows:

```
list1 = list() # Create an empty list
list2 = list([2, 3, 4]) # Create a list with elements 2, 3, 4
list3 = list(["red", "green", "blue"]) # Create a list with strings
list4 = list(range(3, 6)) # Create a list with elements 3, 4, 5
list5 = list("abcd") # Create a list with characters a, b, c, d
```

You can also create a list by using the following syntax, which is a little simpler:

```
list1 = [] # Same as list()
list2 = [2, 3, 4] # Same as list([2, 3, 4])
list3 = ["red", "green"] # Same as list(["red", "green"])
```

The elements in a list are separated by commas and are enclosed by a pair of brackets (`[]`).

Note

A list can contain the elements of the same type or mixed types. For example, the following list is fine:

```
list4 = [2, "three", 4]
```

Inputting Lists

You may often need code that reads data from the console into a list. You can enter one data item per line and append it to a list in a loop. For example, the following code reads ten numbers *one per line* into a list.

```
lst = [] # Create a list
print("Enter 10 numbers: ")
for i in range(10):
    lst.append(eval(input()))
```

Sometimes it is more convenient to enter the data in one line separated by spaces. You can use the string's `split` method to extract data from a line of input. For example, the following code reads ten numbers separated by spaces from one line into a list.

```
# Read numbers as a string from the console
s = input("Enter 10 numbers separated by spaces from one line: ")
items = s.split() # Extract items from the string
lst = [eval(x) for x in items] # Convert items to numbers
```

Invoking `input()` reads a string. Using `s.split()` extracts the items delimited by spaces from string `s` and returns items in a list. The last line creates a list of numbers by converting the items into numbers.

List with more than one

Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.

```
# create two lists
>>> list = [[1,3,5,7,9], [2,4,6,8,10]]

# print the two lists
>>> print(list)
 [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]

# print the first list
>>> print(list[0])
 [1, 3, 5, 7, 9]

# print the second list
>>> print(list[1])
 [2, 4, 6, 8, 10]

# print the first location of first list
>>> print(list[0][0])
1

# print the first location of second list
>>> print(list[1][0])
2

# print first number in first list, and first number in second list
>>> print(list[0][0], list[1][0])
1 2
```

```
# Use Python loop to search if a number is in a list of numbers
myList = [10, 3, 22, 15, 30, 99, 11, 58, 7, 44]
searchN = eval(input("Enter a number between 1 - 100: "))

#search number in list
for i in range(0, len(myList)):
    if myList[i] == searchN:
        print ("Number ", searchN, "is in the list")

#print all numbers in list
for i in range(0, len(myList)):
    print(myList[i])
```

Example of data analysis Program

```
1   NUMBER_OF_ELEMENTS = 5 # For simplicity, use 5 instead of 100
2   numbers = [] # Create an empty list
3   sum = 0
4
5   for i in range(NUMBER_OF_ELEMENTS):
6       value = eval(input("Enter a new number: "))
7       numbers.append(value)
8       sum += value
9
10  average = sum / NUMBER_OF_ELEMENTS
11
12  count = 0 # The number of elements above average
13  for i in range(NUMBER_OF_ELEMENTS):
14      if numbers[i] > average:
15          count += 1
16
17  print("Average is", average)
18  print("Number of elements above the average is", count)
```

```
Enter a new number: 1  ↵Enter
Enter a new number: 2  ↵Enter
Enter a new number: 3  ↵Enter
Enter a new number: 4  ↵Enter
Enter a new number: 5  ↵Enter
Average is 3.0
Number of elements above the average is 2
```

Functions for Lists

Several Python built-in functions can be used with lists. You can use the `len` function to return the number of elements in the list, the `max`/`min` functions to return the elements with the greatest and lowest values in the list, and the `sum` function to return the sum of all elements in the list. You can also use the `shuffle` function in the `random` module to shuffle the elements randomly in the list. Here are some examples:

```
1  >>> list1 = [2, 3, 4, 1, 32]
2  >>> len(list1)
3  5
4  >>> max(list1)
5  32
6  >>> min(list1)
7  1
8  >>> sum(list1)
9  42
10 >>> import random
11 >>> random.shuffle(list1) # Shuffle the elements in list1
12 >>> list1
13 [4, 1, 2, 32, 3]
14 >>>
```

Invoking `random.shuffle(list1)` (line 11) randomly shuffles the elements in `list1`.

Built-in Functions with List	
Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.

enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Return the sum of all elements in the list.

Python List Methods
append() - Add an element to the end of the list
extend() - Add all elements of a list to the another list
insert() - Insert an item at the defined index
remove() - Removes an item from the list
pop() - Removes and returns an element at the given index
clear() - Removes all items from the list
index() - Returns the index of the first matched item
count() - Returns the count of number of items passed as an argument
sort() - Sort items in a list in ascending order
reverse() - Reverse the order of items in the list
copy() - Returns a shallow copy of the list

Some examples of Python list methods:

```
my_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1
print(my_list.index(8))

# Output: 2
print(my_list.count(8))

my_list.sort()

# Output: [0, 1, 3, 4, 6, 8, 8]
print(my_list)

my_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]
print(my_list)
```

How to slice lists in Python?

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])
```

```
# Output: 5
print(n_list[1][3])
```

Iterating Through a List using <mark>in</mark> statement

```
for fruit in ['apple','banana','mango']:
    print("I like",fruit)

my_list = ['p','r','o','b','l','e','m']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

What is tuple in Python?
(retrieved from https://www.pythonlearn.com/html-008/cfbook011.html)

A **tuple** is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are immutable. Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries.

Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists. Iterations or **looping** can be performed in python by 'for' and 'while' loops.

The main **difference between lists** and a **tuples** is the fact that **lists** are mutable whereas **tuples** are immutable. A mutable data type means that a **python** object of this type can be modified. Let's create a **list** and assign it to a variable.

Using a **tuple instead of a list** is like having an implied assert statement that this data is constant, and that special thought (and a specific function) is required to override that. Some **tuples** can be used as dictionary keys (specifically, **tuples** that contain immutable values like strings, numbers, and other **tuples**).

```python
# empty tuple
# Output: ()
my_tuple = ()
print(my_tuple)

# tuple having integers
# Output: (1, 2, 3)
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
# Output: (1, "Hello", 3.4)
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
# Output: ("mouse", [8, 4, 6], (1, 2, 3))
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)

# tuple can be created without parentheses
# also called tuple packing
# Output: 3, 4.6, "dog"

my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
# Output:
# 3
# 4.6
# dog
a, b, c = my_tuple
print(a)
print(b)
print(c)
```

```
#code to test that tuples are immutable

tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
```

Output

```
Traceback (most recent call last):

  File "e0eaddff843a8695575daec34506f126.py", line 3, in

    tuple1[0]=4

TypeError: 'tuple' object does not support item assignment
```