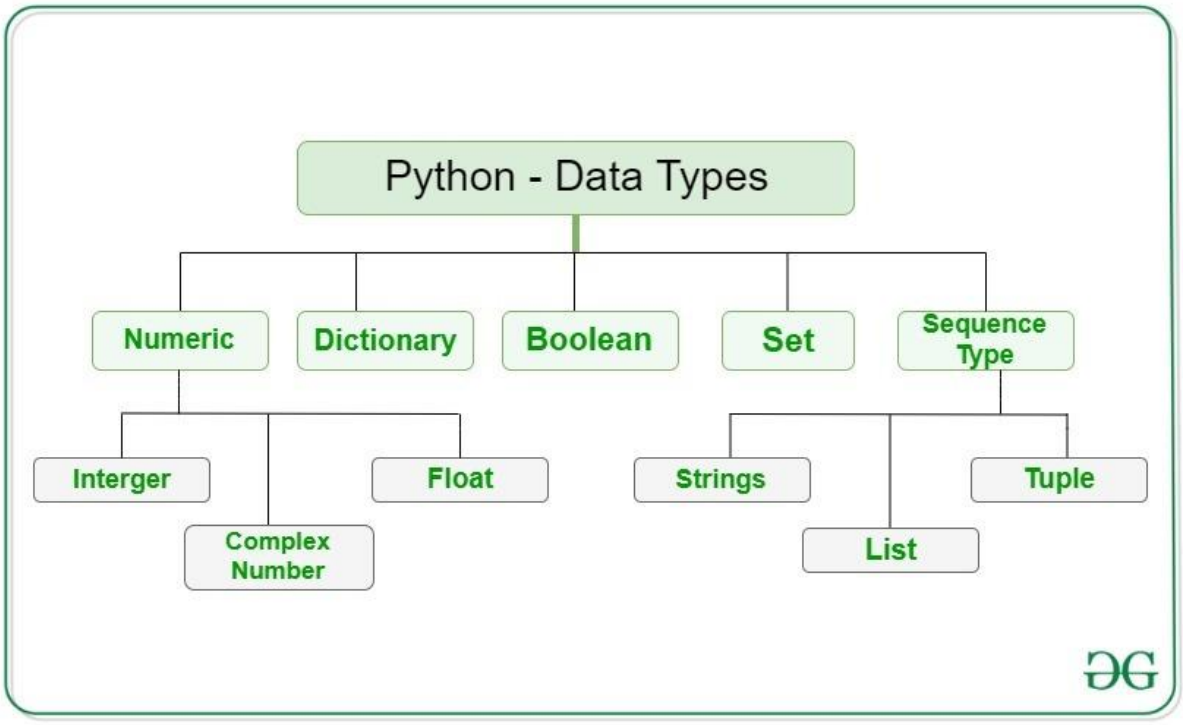
Strings and Lists

Strings and Lists are two built in data-types in Python.

They share many of the same properties and are both sequence datatypes and so a lot of the same operations can be carried out on them. Just as with int and float which are numeric data-types.



Strings

A string is a collection of characters enclosed in single or double quotes or to put it another way strings are arrays of bytes representing Unicode characters. (We will look at the latter in detail later on.)

```
My_string = "hello there"
My_string = 'hello there'
```

You can not mix quotes:

```
My_string = "hello there'
```

will give an error message

Lists

A list is a collection of data items enclosed in square brackets and separated by commas.

```
My_list = ["computer", "science"]
```

The items in a list can be of mixed data types

```
My_list = ["computer","science",12,15.4]
```

Because List is a data-type, you can also have lists inside lists

```
My_list = ["computer",'science',12,15.4,["inner","list"] ]
```

Lists can contain duplicate data

```
My_list= ["computer",'science',12,15.4,["inner","list"],"computer","computer"]
```

The items in a list are ordered*, if you add new items to a list, the new items will be placed at the end of the list.

*You can change the order by calling particular methods, like sort(), but by default new items just go to the end of the list

Important difference between lists and strings

Lists are mutable – this means you can change the contents of a list

Strings are immutable – this means you can not change the contents of a string

Iterable objects

Any Python object capable of **returning its members one at a time**, permitting it to be iterated over in a for-loop, is called iterable.

All the sequence data-types in python are iterable and can be iterated over using a for-loop.

Type the following code, run it in a debugger and make sure you know how it works:

To iterate through a string:

```
for x in "alphabet":
  print(x)
```

To iterate through a list:

```
My_list = ["Monday", "Tuesday", "Wednesday"]
for x in My_list:
  print(x)
```

Indexing and slicing

Both the characters in a string and the items in a list are assigned numbers (called indices) starting from 0 left to right and from -1 working right to left.

G	e	e	k	s		f	o	r		G	e	e	k	s	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

G	e	e	k	s		f	o	r		G	e	e	k	s	!
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

If you wish to access a character in a string or an element/member in a list you can type the variable name followed by the index number of the data item you in square brackets.

```
b = "Hello, World!"
first_char = b[0]
third_char = b[2]
```

NOTE: IF YOU TYPE AN INVALID INDEX NUMBER – YOU WILL GET AN INDEX OUT OF RANGE ERROR

To find the number of characters in a string you can use the len() command

```
b = "Hello, World!"
Num_char_in_b = len(b) # store the returned value
Last_char = b[Num_char_in_b -1] # Why are we minusing 1 at the end?
Last_char = b[-1]
```

Exactly the same operations work on lists

```
b= ["Hello","world","I","need",'some','other','words',21,35.789]
First_item = b[0]
Third_item = b[2]
Num_items_in_lst = len(b)
Last_item = b[Num_items_in_lst -1]
Last_item = b[-1]
```

Substrings and subsets/sublists – slicing

The examples above all returned 1 item from the list or string. Sometimes we need a group of items – this is called slicing.

To slice a list or string, inside the square brackets you put the index of the first item you want, and one more than the index of the last item you want separated by a colon.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])#returns items at index 2,3 and 4
```

```
b = "Hello, World!"
print(b[2:5]) # returns characters at index 2,3 and 4
```

If you leave out the starting value it starts from the first item on the list/string

```
b = "Hello, World!"
print(b[:5])#print everything from the start to position 5
```

If you leave out the end value it takes everything from the start to the end of the list/string

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:]) # take everything from position 2 to the end
```

If you leave out the start and end value it takes everything

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:]) # print the entire list
```

You can also use negative indexing for slicing

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1]) #print from orange up to but not including mango
```

Stepping through a list or string

There is a third argument you can add in square brackets, if you wish to take every nth character.

```
b = "Hello, World!"
print(b[2:10:2]) # return every second character starting at 2 up to but not
including the 10th
print(b[2:10:3]) # return every third character starting at 2 up to but not
including the 10th
```

Concatenate or join together two strings or lists

To join two or more strings or lists you can use the + operator

```
a = "Hello"
b = "World"
c = a + b
print(c) # prints “HelloWorld” notice there is no space, they become 1 new
string

list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
print(list3) #prints ["a","b”,”c”,1,2,3]
```

Note you can also use the * operator on strings and lists

```
a = "Hello"
output_string = a*2
print(output_string) # outputs "HelloHello"
b=["hello"]
output_list = b*2
print(output_list) # outputs ["hello","hello"]
```

Modifying strings

Remember you cannot change a string so when you call a modifier on a string it returns a new string, it does not change the old one. You need to store the new string in a new variable or overwrite the old one.

```
My_string = 'hello'
#Make My_string uppercase and store it in a new variable:
My_string_upper = My_string.upper()
print(My_string_upper) # output is 'HELLO'
#Make My_string uppercase and store it in the old variable
My_string = My_string.upper()
print(My_string) # output is 'HELLO'
```

Modifying lists

To change an item in a list you assign a new data value to item at that index number. YOU CAN NOT DO THIS WITH STRINGS

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)

#Change more than one item at a time
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

NOTE: IF YOU USE THE LEN() COMMAND ON A LIST AND THEN MODIFY THE LIST THE LENGTH MAY HAVE CHANGED AND SO YOU MAY NEED TO CALL THE LEN() COMMAND AGAIN.

To check if a character/item is in a string/list

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.\n")

lst=[ 1, 6, 3, 5, 3, 4 ]
if 7 in lst:
    print("7 is in lst")
else:
    print("7 is not in lst")
```

To check if a character or item is not in a list or string

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.\n")

lst=[ 1, 6, 3, 5, 3, 4 ]
if 7 not in lst:
    print("7 is not in lst")
else:
    print("7 is in lst")
```

Python has a lot of built in methods for modifying lists and strings. Go through each of the methods and operations below and make sure you can use them all. **This means write some small programs which make use of all of them.**

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

String methods

string.upper()	returns uppercase string
string.lower()	returns lowercase string
string.count(x)	counts how many times x appears
string.find(x)	position of the first occurrence of x
string.replace(x,y)	replaces x with y
string.islower()	returns True if all characters are lowercase
string.isupper()	returns True if all characters are uppercase
string.isalnum()	returns True if all characters are alphanumeric
string.isalpha()	returns True if all characters are alphabetic
string.isdigit()	returns True if all characters are digits
string.index(s)	returns index of substring s in string
string.strip(x)	returns a string with leading and trailing characters removed

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index i
list[-i]	retrieves last i item from list
list[i:j]	retrieves items in the range i to j
list[i:]	retrieves items from i to the end
del list[i]	removes the item with index i

List methods

list.append(x)	appends x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns the position of the first occurrence of x in a list
list.count(x)	returns the number of times x appears in a list
list.sort()	sorts items in a list
sorted(L)	returns a new list with L items sorted
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Legend: x, y = any data values; s = string; n = number; L = List