#5: Strings, Lists

Reading: Chapter 4 and 5
Contents

• Strings – more operations with strings
• Lists
• Tuples and Lists
• Sequences

NOTE: Use python3 to run these examples, python (ver 2.7) will give errors for them!
String comparison

Operators: == for equality, != for inequality, < before, > after in alphabetical order

```
adminName = "John"
name = input("What is your name?\n")
if name == adminName:
    print("Welcome admin!")
    if name != adminName:
        print("Intruder!!!")
    if name < adminName:
        print("Your name appears before " + adminName + " in the phone book")
```

Be careful, single = causes the error here
Compound conditions

- A conditions can be built of many smaller conditions
- The smaller conditions have to be connected together by Boolean (logical) operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>mnt &gt;= 9 and mnt &lt;= 12</td>
<td>True only if mnt greater or equal than 9 and mnt less or equal than 12 (mnt may be 9, 10, 11, or 12); otherwise False</td>
</tr>
<tr>
<td>or</td>
<td>mnt == 5 or mnt == 12</td>
<td>True only if mnt is 5 or 12; otherwise False (e.g. for 1, 10, 15)</td>
</tr>
<tr>
<td>not</td>
<td>not name == adminName</td>
<td>It flips the result of comparison. In this case it is equal to condition name != adminName</td>
</tr>
</tbody>
</table>
String is a sequence

- String is a **sequence** of characters or letters here ("Jaros" in fact is J a r o s)
- You can count letters using the `len` function
  
  ```python
  numberOfCharacters = len("Jaros")
  print(numberOfCharacters)
  5
  ```

- String variables are also immutable:
  ```python
  name = "Jaros"; name.add("s")  #error
  name = "Jaros"; nameNew = name + "s"    #this way
  Jaros
  ```

Consequence: **All string methods do NOT change the string but create a NEW string you need to handle**
String is a sequence of letters

• We can access letters in a string in the same manner as we access elements of a tuple:

```python
name = "Jaros"
for letter in name:    # letter is the name to a variable
    print(letter)     # that holds as value each letter
                  # of the string name
```

J     A     R     O     S
String functions

Replace(old, new [,max]) - Replaces all occurrences of old in string with new, or at most max occurrences if max given

```python
nameVal = nameInv.replace("r", "b") #new valid name
```

Jabos

count(str, fromIndex, toIndex) - Counts how many times str occurs in string

find(str, fromIndex, toIndex) - Determine if str occurs in string; returns index if found and -1 otherwise

isdigit() - Returns true if string contains only digits and false otherwise

split(delim, num) - returns a list of all the words in the string, using delim as the separator (splits on all whitespace if left unspecified), optionally limiting the number of splits to num
Slicing (and more about indexing)

- sequence[x] allows to access a single element
- Python extends this concept and possesses methods to extract subsequences from the sequence (it will slice the sequence; slicing basically extracts a substring)
  
  ```
  word = "pizza"
  ```
- `word[0]` is `p`    `word[1]` is `i`
- `word[-1]` is the first letter from the end: `a`
  `word[-2]` is `z`, etc......  `word[10]` or `word[-10]` error
- `word[0:1]` is `pi`    `word[1:3]` is `iz`
  `word[-4:-1]` is `izz` – as you can see, it is the same as `word[1:4]`, so: slicing does not change order of subseq.
- `word[3:0]` is empty    `word[-100:3]` is `piz`
How to insert a substring into a string

```python
org = "This it"
sub = "is "
print(org[:5] + sub + org[5:])
This is it
pos = 5
print(org[:pos] + sub + org[pos:])
```

• Shortcuts:
  – `word[:3]` the same as `word[0:3]`, ie from beginning till index = 3
  – `word[3:]` the same as `word[3:len(word)]`, ie from index 3 till end
  – `word[:]` – the same sequence, ie From beginning till end, so it is the entire word
Tuples

• A tuple is a sequence of immutable (cannot be changed) Python objects.

• Creating tuples

  tupleName = (object1, object2, object3)

• Accessing values in a tuple:

  tupleName[index]
Empty Tuples

• Slicing tuples (obtaining substrings)
  \texttt{tup[firstPosition:lastPosition]}
Results in a new tuple containing every element
between these two positions

• Similarly to strings, empty tuple \texttt{tup1 = ()} is False,
  not empty is True

• Be careful, \texttt{(0,)} is not an empty tuple, it is a tuple
  that has one element that is zero. Alike \texttt{('',)}, \texttt{('',
  0, 0.0)}, etc
Sequences

• The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position, or index. The first index is zero, the second index is one, and so forth.

• Python has six built-in types of sequences, but the most common ones are lists, tuples and strings.

• Operations: indexing, slicing, adding, repetition (*numberOfTimes), and checking for membership.

• Python has built-in functions for finding the length of a sequence, and for finding its largest and smallest elements.
Lists

• A list is a sequence of Python objects that can be changed.

• Creating lists

\[
\text{listName} = [\text{object1, object2, object3}]
\]

Examples:

list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5]
list3 = ['a', 'b', 'c', 'd']

• The empty list: list11 = []

• List containing a single value list12 = [11] or [11,]
Lists

Accessing values in a list:

```python
listName[index]
```

```python
list1 = ['physics', 'chemistry', 1997, 2000]
print(list1[0])
print(list1[1])
print(list1[2])
print(list1[3])
physics
chemistry
1997
2000
```
Lists - index

list1 = ['physics', 'chemistry', 1997, 2000]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Positive index always starts from the first element with index = 0

Negative index starts from the last element with index = -1
Lists operations

Concatenation:
lst1 = [12, 34.56]
lst2 = ['abc', 'xyz']
lst3 = lst1 + lst2
print(lst3)
[12, 34.56, 'abc', 'xyz']

Length: len(lst) # len(lst2) =2, len(lst3) =4
Gives the total length of the list.

max(lst) # max(lst2) = ‘xyz’, max(lst3) – error in Python 3.x
Returns item from the list with max value.

min(lst) #min(lst2) = ‘abc’
Returns item from the list with min value.
Lists operations

**in operator (also not in operator):**

```python
list = [1,2,3,4,5]
3 in list
True
```

**Slicing lists:** `list[firstPosition:lastPosition]`
- Results in a new list containing every element between these two positions

```python
list[1:3]
[2,3]
```
Modifying lists

• You can **update** single or multiple elements of lists by giving the slice on the left-hand side of the **assignment** operator

```python
list = ['physics', 'chemistry', 1997, 2000]
print(list[2])
1997
print(list[2])
2001
```
Modifying lists

• you can **add** to elements in a list with the `append()` method (ie, adds at the end of the list).

```
list = ['physics', 'chemistry', 1997, 2000]
print(list)
['physics', 'chemistry', 1997, 2000]
list.append(12.55)
print(list)
['physics', 'chemistry', 1997, 2000, 12.55]
```
Modifying lists

• To **remove** a list element, you can use either the `del` statement if you know exactly which element(s) you are deleting or the `remove()` method if you do not know.

```python
list = ['physics', 'chemistry', 1997, 2000]
print(list)
['physics', 'chemistry', 1997, 2000]
del list[2]
['physics', 'chemistry', 2000]
list.remove('physics')
['chemistry', 2000]
The method **remove()** removes first *obj* from the list.
Modifying lists

• The method **insert()** inserts object *obj* into list at position *index*.

```
list.insert(index, obj)
list.insert(1, 'joe')  # list = ['chemistry', 2000]
print(list)
['chemistry', 'joe', 2000]
```
Modifying lists

• The method `sorted()` sorts objects of list, use compare `func` if given.

```
sorted(list,[func])
```

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
sorted(aList) # error in python3 – unorderable types
aList.remove(123)
sorted(aList)
print("List : ", aList)
List : ['abc', 'xyz', 'xyz', 'zara']
```
Lists

• The method `index()` returns the first index in list at which `obj` appears. It gives an error if there is no such object `obj`.

```python
list.index(obj)
```

```python
aList = [123, 'xyz', 'zara', 'abc']
print("Index for xyz : ", aList.index('xyz'))
Index for xyz : 1
```
Lists

The method <, >, == compares elements of two lists.

list1 < list2

list1 = [123, 'xyz']
list2 = [456, 'abc']
print(list1 < list2)
True
print(list1 == list2)
False
list3 = list2 + [786]
print(list3, '...', list2 > list3)
[456,'abc',789] ... False
Next lecture ...

- Functions