Hacettepe University
Computer Engineering Department

Programming in Python

BBM103 Introduction to Programming Lab 1
Week 8

Fall 2019
Collections

- A Collection Groups Similar Things
  - **List**: ordered
  - **Set**: unordered, no duplicates
  - **Tuple**: unmodifiable list
  - **Dictionary**: maps from keys to values
Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list:

```python
my_list = [1, 2, 3, 4, 5]
```

Splitting a string to create a list:

```python
s = 'spam-spam-spam'
delimiter = '-'
s.split(delimiter)
```

Output: ['spam', 'spam', 'spam']
Lists

Making a list of chars in a string:

```
s = 'spam'
t = list(s)
print(t)
```

Output: 
```
['s', 'p', 'a', 'm']
```

Joining elements of a list into a string:

```
t = ['programming', 'is', 'fun']
delimiter = ' '
delimiter.join(t)
```

Output: 'programming is fun'
Accessing Values in Lists by Index:

```python
list1 = [1, 2, 3, 4, 5]
print("list1[0]: ", list1[0])
print("list1[1:5]: ", list1[1:5])
```

Output:
```
list1[0]: 1
list1[1:5]:  [2, 3, 4, 5]
```
Updating Lists:

```python
list = [1, 2, 3, 4, 5]
print ("Value available at index 2: ", list[2])
list[2] = 6
print ("New value available at index 2: ", list[2])
```

Output:
Value available at index 2:  3
New value available at index 2:  6
Deleting List Elements

```python
list = [1, 2, 3, 4, 5]
print(list)
del list[2]
print("After deleting value at index 2: ", list)
```

Output:

```
[1, 2, 3, 4, 5]
After deleting value at index 2:  [1, 2, 4, 5]
```
# Basic List Operations

<table>
<thead>
<tr>
<th>Python Expression</th>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len([1, 2, 3])</td>
<td>3</td>
<td>Length</td>
</tr>
<tr>
<td>[1, 2, 3] + [4, 5, 6]</td>
<td>[1, 2, 3, 4, 5, 6]</td>
<td>Concatenation</td>
</tr>
<tr>
<td>['Hi!'] * 4</td>
<td>['Hi!', 'Hi!', 'Hi!', 'Hi!']</td>
<td>Repetition</td>
</tr>
<tr>
<td>3 in [1, 2, 3]</td>
<td>True</td>
<td>Membership</td>
</tr>
<tr>
<td>for x in [1, 2, 3]: print x,</td>
<td>1 2 3</td>
<td>Iteration</td>
</tr>
</tbody>
</table>
List Functions & Methods:

**len (list)**: Gives the total length of the list.

Example:

```python
list = [1, 2, 3, 4, 5]
print ('length of the list is', len(list))
```

Output:
```
length of the list is 5
```
max(list): Returns the item from the list with the maximum value.

Example:

```python
list=[456, 700, 200]
print("Max value element: ", max(list))
```

Output:
Max value element: 700
min(list): Returns the item from the list with the minimum value.

Example:

```python
list=[456, 700, 200]
print ("Min value element:", min(list))
```

Output:
Min value element: 200
list.append(obj): Appends object obj to list

Example:

```
list = [123, 'xyz', 'zara', 'abc']
list.append(2009)
print("Updated List: ", list)
```

Output:
Updated List:   [123, 'xyz', 'zara', 'abc', 2009]
list.count(obj): Returns the count of how many times obj occurs in a list

Example:

```python
aList = [123, 'xyz', 'zara', 'abc', 123]
print("Count for 123: ", aList.count(123))
print("Count for zara: ", aList.count('zara'))
```

Output:
Count for 123:  2
Count for zara:  1
list.extend(seq) : Appends the contents of seq to list

Example:

```python
aList = [123, 'xyz', 'zara']
bList = [2009, 'manni']
aList.extend(bList)
print("Extended List: ", aList)
```

Output:
Extended List:  [123, 'xyz', 'zara', 2009, 'manni']
`list.index(obj)`: Returns the lowest index of `obj` in the list

Example:

```python
list=[456, 700, 200]
print ("Index of 700: ", list.index(700) )
```

Output:

```
Index of 700:  1
```
list.insert(index, obj): Inserts object obj into the list at offset index

Example:

```python
aList = [123, 'xyz', 'zara', 'abc']
aList.insert(3, 2009)
print("Final List: ", aList)
```

Output:
Final List:  [123, 'xyz', 'zara', 2009, 'abc']
list.pop(obj=list[-1]): Removes and returns the last obj from list

Example:

```python
aList = [123, 'xyz', 'zara', 'abc']
print ("A List: ", aList.pop())
print ("B List: ", aList.pop(2))
```

Output:

A List:  abc
B List:  zara
list.remove(obj): Removes object obj from list

Example:

```python
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.remove('xyz')
print ("List: ", aList)
aList.remove('abc');
print ("List: ", aList)
```

Output:
```
List :  [123, 'zara', 'abc', 'xyz']
List :  [123, 'zara', 'xyz']
```
Example:

```python
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.reverse()
print("List: ", aList)
```

Output:

List: ['xyz', 'abc', 'zara', 'xyz', 123]
list.sort([]): Sorts objects of list, uses compare func if given

Example:

```python
aList = ['xyz', 'zara', 'abc', 'xyz']
aList.sort()
print("List: ", aList)
```

Output:
List: ['abc', 'xyz', 'xyz', 'zara']
List Comprehensions

```python
liste = [i for i in range(1000)]
```

**Method 1:**
```python
liste = [i for i in range(1000) if i % 2 == 0]
```

**Method 2:**
```python
liste = []
for i in range(1000):
    if i % 2 == 0:
        liste += [i]
```
Sets

Sets are lists with no duplicate entries.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Equivalent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.update(t)</td>
<td>s</td>
<td>= t</td>
</tr>
<tr>
<td>s.intersection_update(t)</td>
<td>s &amp;= t</td>
<td>return set s keeping only elements also found in t</td>
</tr>
<tr>
<td>s.difference_update(t)</td>
<td>s -= t</td>
<td>return set s after removing elements found in t</td>
</tr>
<tr>
<td>s.symmetric_difference_update(t)</td>
<td>s ^= t</td>
<td>return set s with elements from s or t but not both</td>
</tr>
<tr>
<td>s.add(x)</td>
<td></td>
<td>add element x to set s</td>
</tr>
<tr>
<td>s.remove(x)</td>
<td></td>
<td>remove x from set s; raises <strong>KeyError</strong> if not present</td>
</tr>
<tr>
<td>s.discard(x)</td>
<td></td>
<td>removes x from set s if present</td>
</tr>
<tr>
<td>s.pop()</td>
<td></td>
<td>remove and return an arbitrary element from s; raises <strong>KeyError</strong> if empty</td>
</tr>
<tr>
<td>s.clear()</td>
<td></td>
<td>remove all elements from set s</td>
</tr>
</tbody>
</table>
**Example:**

```python
list = ["elma", "armut", "elma", "kebap", "şeker",
... "armut", "çilek", "ağac", "şeker", "kebap", "şeker"]
for i in set(list):
    print(i)
```

**Output:**

```
çilek
elma
kebap
ağac
şeker
```

**Example:**

```python
list = ["elma", "armut", "elma", "kiraz",
... "çilek", "kiraz", "elma", "kebap"]
for i in set(list):
    print("{} count:  {}".format(i, list.count(i)))
```

**Output:**

```
armut count:  1
çilek count:  1
elma count:  3
kiraz count:  2
kebap count:  1
```
Tuples

• A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.
• What are the differences between tuples and lists?
Tuples

- A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.

- The differences between tuples and lists are,
  - the **tuples cannot be changed** unlike lists,
  - tuples use parentheses, whereas lists use square brackets.

Creating a tuple: \[
\texttt{tup} = (1, 2, 3, 4, 5)
\]
Accessing Values in Tuples:

tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print("tup1[0]: ", tup1[0])
print("tup2[1:5]: ", tup2[1:5])

Output:
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
Updating Tuples

tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
tup3 = tup1 + tup2
print (tup3)

Output:
(12, 34.56, 'abc', 'xyz')
1. Write a function that finds the \textit{nth} largest element of the given list.

\textbf{Input:} \(L = [1, 5, 6, 4, 2]\), \textit{n}=3

\textbf{Output:} 4

2. Write a function that determines if the given input string is a Palindrome or not.

- \textit{A palindrome} is a sequence of characters which reads the same backward as forward
3. Implement the following integer functions:

- a) Function `celcius` returns the Celsius equivalent of a Fahrenheit temperature.

- b) Function `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature.

\[ F = \frac{9}{5} C + 32 \]

Celsius to Fahrenheit Formula
4. Write a function `perfect` that determines if a number given as a parameter is a perfect number or not. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000.

**Perfect Number:**

- An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because $6 = 1 + 2 + 3$. 
Exercise

• Write a function that takes a string which contains lines separated by a pipe (|), and each line contains numbers separated by a colon (:), prints out the sum of the numbers divisible by 3 for each line and at the end of the lines prints out the sum of the previous sums.

Input:

Output:
1. line: 12
2. line: 96
3. line: 57
Sums: 165