CSc 120
Introduction to Computer Programming II

Adapted from slides by Dr. Saumya Debray

01-b: Python review
Lists: sorting

```python
>>> x = [1, 4, 3, 2, 5]
>>> x
[1, 4, 3, 2, 5]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]

>>> y = [1, 4, 3, 2, 5]
>>> y
[1, 4, 3, 2, 5]
>>> sorted(y)
[1, 2, 3, 4, 5]
>>> y
[1, 4, 3, 2, 5]
```
Lists: sorting

```python
global x, y
>>> x = [1, 4, 3, 2, 5]
>>> x
[1, 4, 3, 2, 5]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]

>>> y = [1, 4, 3, 2, 5]
>>> y
[1, 4, 3, 2, 5]
>>> sorted(y)
[1, 2, 3, 4, 5]
>>> y
[1, 4, 3, 2, 5]
```

- `sort()` : sorts a list
- `sorted()` : creates a sorted copy of a list; the original list is not changed
python review: for loops
Loops II: for

- The for loop iterates over the items of any sequence in order
- **for**-statement syntax:

  ```python
  for Var in Expr :
    stmt_1
    ...
    stmt_n
  ```

- **Expr** is evaluated. **stmt_1 ... stmt_n** are executed for each element of the sequence that **Expr** produces; **Var** is assigned to each successive element.
Loops II: `for`

```python
>>> nums = [18, 3, 24, 63, 18, 4, 7]
>>> evens = []
>>> for n in nums:
    if n % 2 == 0:
        evens.append(n)
```

```python
>>> evens
[18, 24, 18, 4]
```
range

- **range** generates a sequence of numbers

- **range** syntax:
  
  range(start, stop, step)

  range(start, stop)

Produces the sequence of integers from **start** to **stop** (exclusive). If **step** is omitted, it defaults to 1.
for with range

```python
>>> nums = [18, 3, 24, 63, 18, 4, 7]

>>> evens = []

>>> for i in range(0, len(nums)):
    if nums[i] % 2 == 0:
        evens.append(nums[i])

>>> evens
[18, 24, 18, 4]
```

• generates the numbers 0,1,2,3,4,5,6
EXERCISE

```python
>>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]

>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]

>>> total = 0

>>> for i in range(len(x)):
    total += x[i][0]

>>> total
61
```
EXERCISE

```python
>>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
```
python review:
lists ↔ strings
Strings → lists

>>> names = "John, Paul, Megan, Bill, Mary"

>>> names
'John, Paul, Megan, Bill, Mary'

>>> names.split()
['John,', 'Paul,', 'Megan,', 'Bill,', 'Mary']

>>> names.split('n')
['Joh', ', Paul, Mega', ', Bill, Mary']

>>> names.split(',')
['John', ', Paul', ', Megan', ', Bill', ', Mary']

split() : splits a string on whitespace returns a list of strings
Strings → lists

```python
>>> names = "John, Paul, Megan, Bill, Mary"
>>> names
'John, Paul, Megan, Bill, Mary'
```

```python
>>> names.split()
['John,', 'Paul,', 'Megan,', 'Bill,', 'Mary']
```

```python
>>> names.split('n')
['Joh', 'Paul, Mega', ', Bill, Mary']
```

```python
>>> names.split(',)
['John', ' Paul', ' Megan', ' Bill', ' Mary']
```
Lists → strings

>>> x = ['one', 'two', 'three', 'four']

>>> 

>>> "-".join(x)
'one-two-three-four'

>>> 

>>> "!".join(x)
'one!two!three!four'

>>> delimiters.join(list) : joins the strings in list using the string delimiters as the delimiter

returns a string
String trimming

>>> x = '  abcd  '
>>> x.strip()
'abcd'

>>> y = "Hey!!!"

>>> y.strip('!')
'Hey'

>>> z = "*%^^stuff stuff stuff%^%**"

>>> z.strip('%^%')
'stuff stuff stuff'
String trimming

```python
>>> x = ' abcde '  # x.strip() : removes whitespace from either end of the string x

>>> x.strip()
'abcd'

>>> y = "Hey!!!"

>>> y.strip('!')
'Hey'

>>> z = "*%^stuff stuff stuff%^%**"  # x.strip(string) : given an optional argument string, removes any character in string from either end of x

>>> z.strip("*%^")
'stuff stuff stuff'
```
String trimming

\[ x\.strip() : \text{removes whitespace from either end of the string } x \]

\[ x\.strip(string) : \text{given an optional argument } string, \text{ removes any character in } string \text{ from either end of } x \]

\[ \text{rstrip()}, \text{lstrip()} : \text{similar to } \text{strip() but trims from one end of the string} \]
EXERCISE

```python
>>> text = "Bear Down, Arizona. Bear Down, Red and Blue."
>>> text_lst = text.split()  # create a list of words with no punctuation
['Bear', 'Down,', 'Arizona.', 'Bear', 'Down,', 'Red', 'and', 'Blue.]
>>> words_lst = []
>>> for w in words:
...     words_lst.append(w.strip(".,"))

>>> words_lst
['Bear', 'Down', 'Arizona', 'Bear', 'Down', 'Red', 'and', 'Blue']
```
Lists – Shallow copies

concatenation (+ and *) : similar to strings

to list mutability, this can cause unexpected behavior

```python
>>> x = [ [12, 34, 56] ]
>>> y = x * 3
>>> y
[[[12, 34, 56], [12, 34, 56], [12, 34, 56]]

>>> y[0].append(78)

>>> y
[[[12, 34, 56, 78], [12, 34, 56, 78], [12, 34, 56, 78]]
```
Lists

concatenation (+ and *): similar to strings

these operators create “shallow” copies
• due to list mutability, this can cause unexpected behavior

shallow copying
Lists

concatenation (+ and *) : similar to strings

these operators create “shallow” copies
• due to list mutability, this can cause unexpected behavior

shallow copying

\[
\begin{align*}
\text{x} & \quad \rightarrow \quad 0 \\
\text{y} & \quad \rightarrow \\
& \quad 12 \quad 34 \quad 56 \quad 78
\end{align*}
\]

after \(y[0].append(78)\)
python review: reading user input II: file I/O
suppose we want to read (and process) a file "this_file.txt"
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")
>>> for line in infile:
    print(line)

line 1 line 1 line 1
line 2 line 2
line 3 line 3
```
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")
>>> for line in infile:
    print(line)

line 1 line 1 line 1
line 2 line 2
line 3 line 3
```
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")
>>> for line in infile:
    print(line)
```

```
line 1 line 1 line 1
line 2 line 2
line 3 line 3
```
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")
```
```python
>>> for line in infile:
    print(line)
```
```plaintext
line 1 line 1 line 1
line 2 line 2
line 3 line 3
```
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")

>>> for line in infile:
    print(line)

line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>> At this point we've reached the end of the file and there is nothing left to read
```
Reading user input II: file I/O

>>> infile = open("this_file.txt")

>>> for line in infile:
    print(line)

line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>>(infile.close()

at this point we've reached the end of the file so there's nothing left to read

to re-read the file, we have to close it and then re-open it

>>> infile = open("this_file.txt")
Reading user input II: file I/O

```python
>>> infile = open("this_file.txt")

>>> for line in infile:
    print(line.strip())
```

NOTE: we can use strip() to get rid of the newline character at the end of each line

```
line 1 line 1 line 1
line 2 line 2
line 3 line 3
```
Writing output to a file

```python
open(filename, "w") : opens filename in write mode, i.e., for output
```
Writing output to a file

`open(filename, "w")`: opens `filename` in write mode, i.e., for output

`fileobj.write(string)`: writes `string` to `fileobj`
Writing output to a file

```python
open(filename, "w") : opens filename in write mode, i.e., for output

fileobj.write(string) : writes string to fileobj

open the file in read mode ("r") to see what was written
```
python review: tuples
Tuples

A tuple is a sequence of values (like lists).
Tuples

A tuple is a sequence of values (like lists)

- Tuples use parens ()
- By contrast, lists use square brackets []
  - Parentheses can be omitted if no confusion is possible
- Special cases for tuples:
  - Empty tuple: ()
  - Single-element tuple: must have comma after the element:
    
    (111,)

Tuples

- A tuple is a sequence of values (like lists).
- Tuples use parens ( ).
  - By contrast, lists use square brackets [ ].
  - Parens can be omitted if no confusion is possible.
- Special cases for tuples:
  - Empty tuple: ()
  - Single-element tuple: must have comma after the element:
    
    \[(111,)\]

Indexing in tuples works similarly to strings and lists.
Tuples

computing a length of a tuple: similar to strings and lists
Tuples

computing a length of a tuple: similar to strings and lists

computing slices of a tuple: similar to strings and lists
Tuples

+ and * work similarly on tuples as for lists and strings
Tuples

iterating through the elements of a tuple: similar to lists and strings
Tuples

iterating through the elements of a tuple: similar to lists and strings

checking membership in a tuple: similar to lists and strings
Tuples

tuples are not mutable
Sequence types: mutability

tuples are immutable
Sequence types: mutability

tuples are immutable

lists are mutable (even if the list is an element of a [immutable] tuple)
Sequence types: mutability

- Tuples are immutable
- Lists are mutable (even if the list is an element of a [immutable] tuple)
- Strings are immutable (even if the string is an element of a [mutable] list)
Sequence types: mutability

tuple (immutable)
list (mutable)
string (immutable)
Sequence types: mutability

```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'] )
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
```

- **tuple** (immutable)
- **list** (mutable)
- **string** (immutable)
Why use tuples?

At the implementation level, tuples are much simpler than lists:

• lists are mutable; tuples are immutable
  • this means that the implementation can process tuples without having to worry about the possibility of updates

• lists have methods (e.g., append); tuples do not have methods

⇒ Tuples can be implemented more efficiently than lists
Summary: sequence types

Sequence types include: strings, lists, and tuples

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x in s</code></td>
<td>True if an item of <code>s</code> is equal to <code>x</code>, else False</td>
</tr>
<tr>
<td><code>x not in s</code></td>
<td>False if an item of <code>s</code> is equal to <code>x</code>, else True</td>
</tr>
<tr>
<td><code>s + t</code></td>
<td>the concatenation of <code>s</code> and <code>t</code></td>
</tr>
<tr>
<td><code>s * n</code> or <code>n * s</code></td>
<td>equivalent to adding <code>s</code> to itself <code>n</code> times</td>
</tr>
<tr>
<td><code>s[i]</code></td>
<td><code>i</code>th item of <code>s</code>, origin 0</td>
</tr>
<tr>
<td><code>s[i:j]</code></td>
<td>slice of <code>s</code> from <code>i</code> to <code>j</code></td>
</tr>
<tr>
<td><code>s[i:j:k]</code></td>
<td>slice of <code>s</code> from <code>i</code> to <code>j</code> with step <code>k</code></td>
</tr>
<tr>
<td><code>len(s)</code></td>
<td>length of <code>s</code></td>
</tr>
<tr>
<td><code>min(s)</code></td>
<td>smallest item of <code>s</code></td>
</tr>
<tr>
<td><code>max(s)</code></td>
<td>largest item of <code>s</code></td>
</tr>
<tr>
<td><code>s.index(x[, i[, j]])</code></td>
<td>index of the first occurrence of <code>x</code> in <code>s</code> (at or after index <code>i</code> and before index <code>j</code>)</td>
</tr>
<tr>
<td><code>s.count(x)</code></td>
<td>total number of occurrences of <code>x</code> in <code>s</code></td>
</tr>
</tbody>
</table>

Source: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

The elements are: `i`, `i+k`, `i+2k`, ...
EXERCISE

```python
>>> x = [ (1, 2, 3), (4, 5, 6), (7, 8, 9) ]

>>> x[0][0] = (2, 3, 4)

what do you think will be printed out?

what do you think will be printed out?

>>> x[0] = [ 2, 3, 4 ]
```
python review: dictionaries
Dictionaries

• A dictionary is like an array, but it can be indexed using strings (or numbers, or tuples, or any immutable type)
  • the values used as indexes for a particular dictionary are called its keys
  • think of a dictionary as an unordered collection of key : value pairs
  • empty dictionary: {}

• It is an error to index into a dictionary using a non-existent key
Dictionaries

```python
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>> ...
```
Dictionaries

empty dictionary
populating the dictionary
• in this example, one item at a time
Dictionaries

empty dictionary
populating the dictionary
• in this example, one item at a time
looking up the dictionary (indexing)
Dictionaries

empty dictionary
populating the dictionary
• in this example, one item at a time
looking up the dictionary (indexing)

looking at the dictionary
• we can use this syntax to populate the dictionary too
Dictionaries

empty dictionary
populating the dictionary
  • in this example, one item at a time
looking up the dictionary (indexing)
looking at the dictionary
  • we can use this syntax to populate the dictionary too

indexing with a key not in the dictionary is an error (KeyError)

```python
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3

>>> course = 'csc 110'

>>> crs_units[course]
4

>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>> crs_units['mis 115']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    crs_units['mis 115']
KeyError: 'mis 115'
```
Dictionaries

- initializing the dictionary
- in this example, several items at once
Dictionaries

initializing the dictionary
• in this example, several items at once

getting a list of keys in the dictionary
• useful since it’s an error to index into a dictionary with a key that is not in it
Dictionaries

We can use a `for` loop to iterate through a dictionary.
We can use a **for** loop to iterate through a dictionary

Notice that this iteration may not list the items in the dictionary in the same order as when they were inserted.
EXERCISE

```python
>>> crs_units = { 'csc 352' : 3, 'csc 120': 4, 'csc 110': 4 }
>>> for crs in crs_units:
    print( '{0} : {1} units'.format( crs, crs_units[crs] ) )

csc 110 : 4 units
**csc 120 : 4 units**
csc 352 : 3 units
```

How can we get the dictionary contents to be printed out in sorted order of the keys? (i.e., what goes in the box?)
Lists of Lists (Review)

a list can consist of elements of many types, including lists

```python
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]

>>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
```
Lists of Lists (Review)

a list can consist of elements of many types, including lists

a list of lists is called a 2-d list

if the number of rows and columns are equal, it is a grid

```python
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
```

```python
>>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
```
Lists of Lists (Review)

```python
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]

>>> y[0]
['aa', 'bb', 'cc']

>>> y[1]
['dd', 'ee', 'ff']

>>> y[2]
['hh', 'ii', 'jj']

>>> len(y)
3

>>> len(y[0])
3
```

- A list can consist of elements of many types, including lists.
- A list of lists is called a 2-d list.
- If the number of rows and columns are equal, it is a grid.
- *Must check the length of each row.*
a list can consist of elements of many types, including lists

```python
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
```
EXERCISE

>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]

>>> y[0]
['aa', 'bb', 'cc']

>>> y[1]
['dd', 'ee', 'ff']

>>> y[2]
['hh', 'ii', 'jj']

>>> y[0][1]
'bb'

how do we access 'bb'?
EXERCISE

```python
>>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> r, total = 0, 0
>>> while r < len(x):
...     total += x[r][0]
...     r += 1

>>> total
61
```

write the code to sum the first column of \( x \)