Lecture 3

Istvan Albert

Huck Institutes for the Life Sciences
Homework deadline change

• Homework will be due on next week’s **Tuesday** rather than **Thursday**.

• Allows us to focus the entire week on the upcoming homework, rather than mixing the two

• Note: There are office hours on Monday before the Tuesday’s lecture.
List Containers

Lists are one of the simplest yet most important data containers. Think of them as a row or column of values.

Designated using the square brackets

- [ ] represents an empty list
- [ 1, 2, 3 ] is a list containing just numbers
- [ 1, “Hello”, [] ] is a list various other types
List is a type
Indexing and slicing. Index starts at 0!

```python
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

# first, last elements
print data[0]
print data[-1]

# first two elements
print data[0:2]

# last two elements
print data[-2:None]
print data[-2:]
```

Command Output:
```
python -t C:cygwin\home\ialbert\sources\ialbert-web\ppt\week2\test-index.py 2>&1` returned 0
```

```
a
g
['a', 'b']
['f', 'g']
['f', 'g']
```
Advanced slicing

data = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

# going by two for the first four
print data[0:4:2]

# going by three over the entire data
print data[None:None:3]
print data[::3]

# this actually reverses the data
print data[:::-1]

# negative strides
# can be counter intuitive
Functions that operate on lists

```python
data = [1.0, 2.0, 3.0, 4.0, 5.0]

# sum of values in the list
print sum(data)

# length of the list
print len(data)

# maximum value
print max(data)

print min(data)
```
Sorting a list

def data = [ 3.0, 1.0, 5.0, 2.0, 4.0 ]

data.sort()

print data

[1.0, 2.0, 3.0, 4.0, 5.0]
Mapping Functions

Map = “Mass Transformation”

\[
\text{map}(\cos, [1, 2, 3]) = [\cos(1), \cos(2), \cos(3)]
\]

A way to execute a function on each element of a list, and return the mapped values as a new list
Mapping Example

```
# test-map.py

data = [ '1', '4', '2', '5', '3']

print data

values = map(float, data)

print values
```

```
[ '1', '4', '2', '5', '3']
[1.0, 4.0, 2.0, 5.0, 3.0]
```
Onto files: first check your settings

Tells the editor to look for files in the same location that the program runs.
Let’s check the file’s content

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.02</td>
</tr>
<tr>
<td>2</td>
<td>-6.92</td>
</tr>
<tr>
<td>3</td>
<td>5.45</td>
</tr>
<tr>
<td>4</td>
<td>3.48</td>
</tr>
<tr>
<td>5</td>
<td>4.40</td>
</tr>
<tr>
<td>6</td>
<td>3.27</td>
</tr>
<tr>
<td>7</td>
<td>2.60</td>
</tr>
<tr>
<td>8</td>
<td>-2.69</td>
</tr>
<tr>
<td>9</td>
<td>-3.59</td>
</tr>
<tr>
<td>10</td>
<td>5.27</td>
</tr>
<tr>
<td>11</td>
<td>-0.02</td>
</tr>
<tr>
<td>12</td>
<td>-3.68</td>
</tr>
<tr>
<td>13</td>
<td>-2.62</td>
</tr>
<tr>
<td>14</td>
<td>0.75</td>
</tr>
<tr>
<td>15</td>
<td>-0.62</td>
</tr>
<tr>
<td>16</td>
<td>-7.01</td>
</tr>
</tbody>
</table>
File access checklist:

1. Open a file, this returns a stream (a.k.a file handle)

2. Read from the stream

3. Close stream (optional for now)
1. Open stream

```python
stream = file('data1.txt')
print stream
```

Command Output:
```
<open file 'data1.txt', mode 'r' at 0x02211260>
```
You may want to look at stream attributes

```python
stream = file('data1.txt')
print dir(stream)
```
Casting a file as list

```python
stream = file('data1.txt')
print(list(stream))
```
1. Find the **sum**, **average**, **minimum** and **maximum** of the data contained in the file named `data1.txt`.

2. Suppose the highest and lowest values are statistical outliers. Remove these then redo point 1.

3. If you finish early, read the next slides then do the same on `data2.txt` then on `data3.txt`.
Lecture 4

Istvan Albert

Huck Institutes for the Life Sciences
1. We have `sum, max, min, len` - why isn’t there a function that returns the average?

2. How do we know what functions are available?
Built-in functions

```python
1
2   print type(sum)
3
4   #print dir(__builtins__)
5
```

Command Output

```
<type 'builtin_function_or_method'>
```
There is an average function in Numerical Python.
1. \( a = 1 + 2 \) returns a value that sets \( a=3 \)

2. But \texttt{values = data.sort()} is incorrect why?

\texttt{data.sort()} mutates the original dataset
Yes! There is a `sorted()` function

```python
# A recent addition to the language – that I forgot about – let’s use it this way

data = [ 3.0, 1.0, 5.0, 2.0, 4.0 ]
data = sorted(data)
print(data)

# Reverse sorting

data = sorted(data, reverse=True)
print(data)
```

```
[1.0, 2.0, 3.0, 4.0, 5.0]
[5.0, 4.0, 3.0, 2.0, 1.0]
```
Data naming tips

As long as the data type does not change and old versions are not needed later you may overwrite their names.
Continuing from the last lecture

```python
stream = file('data2.txt')
lines = list(stream)

print lines

values = map(float, lines)
```

```
['4.34\n', '\n', '1.73\n', '5.39\n', '\n', '-0.89\n']
Traceback (most recent call last):
  File "C:\cygwin\home\ialbert\sources\ialbert-web\ppt\week2\test-stream.py", line 7, in <module>
    values = map(float, lines)
ValueError: empty string for float()
```
Introducing functions

• So far we only named values `greeting=“Hello”`

• But we can also assign a *series instructions* to a name – these are called *functions*

• Makes it easy to call the same instructions repeatedly

• You can also pass them to `map` and other tools
The anatomy of a function

```python
def name ( parameters ):
    instruction 1
    instruction 2
    instruction 3
    return value
```

The indentation determines what instructions are part of the function.
A simple function

```python
def greet(name):
    print "Hello", name

print type(greet)

greet('World!')
greet('Universe!')
```

```
<type 'function'>
Hello World!
Hello Universe!
```
We could simplify a previous task.

```python
def report(values):
    print 'Sum =', sum(values)
    print 'Avg =', sum(values)/len(values)

stream = file('data1.txt')
lines = list(stream)
values = map(float, lines)
values = sorted(values)

report(values)
report(values[1:-1])
```

```
Sum  = 6.62
Avg  = 0.331
Sum  = 8.18
Avg  = 0.454444444444
```
We can group the tasks in other ways

```python
def report(filename):
    stream = file(filename)
    lines = list(stream)
    values = map(float, lines)
    values = sorted(values)
    print 'Sum =', sum(values)
    print 'Avg =', sum(values)/len(values)

report('data1.txt')
report('data1.txt')
```

Command Output:

```
Sum = 6.62
Avg = 0.331
Sum = 6.62
Avg = 0.331
```
What happens when we map functions

```python
def myfloat(x):
    print 'Inside myfloat, parameter =', x
    return float(x)

data = [1, 2, 3, 4]

values = map(myfloat, data)

print values
```

```
Inside myfloat, parameter = 1
Inside myfloat, parameter = 2
Inside myfloat, parameter = 3
Inside myfloat, parameter = 4
[1.0, 2.0, 3.0, 4.0]
```
Filtering data

Filter = “Mass Filtering”

filter( function, data ) = [ keep all elements in data where function(element) is True ]

Or you can think of it as removing those elements from the list where the function evaluates to false.
Define a simple function

```python
def positive(x):
    return x >= 0

print positive(300.0)
print positive(-20.0)
```
Applying a filter. Keep positive values

```python
def positive(x):
    return x >= 0

values = [-1, 2, 3, -7, 4]

print(filter(positive, values))
```

Command Output:
```
[2, 3, 4]
```
Map and filter allow you to solve a lot of problems

Identify the steps needed to process the file then:

- To remove data use **filter**
- To transform data use **map**
Filtering data from files

It is a quick and easy solution but is not “robust”. It can only handle empty lines. We will see better approaches later.
Homework: next Tuesday

Using the file `data2.txt` generate the sum and average under the following restrictions:

- all numbers in the file
- after removing the two largest values
- using only the values that are larger than 2 but smaller than 5

Turn in the code and the output
In class exercise

We can do part of the homework.

Even if we finish here try to repeat the steps later independently.