Week 5 - Lecture 9

István Albert
BMB, Bioinformatics
Collect all of your useful functions into the bmmb.py module

Make it globally accessible to all python programs (add it to the import path)

Improve and refine (refactor) them
“Hardcoded” column name – not flexible
New desired behavior

```python
import bmmmb

values = bmmmb.getcolumn('GPL9270/GSM455822.txt', 'CH2_SIG_MEAN')

print values[:10]
```

Takes a filename and column name

```
['9181', '4541', '8165', '8036', '10136', '5652', '8550', '8245', '5397', '8596']
```

No header, values are still strings
Step 2: setting the right scope

• **Scope** ➔ visibility of names in a program

• Python: every name in a code block will be **readable** from a sub-block

• Can be used to build interesting solutions
Simple scope example

```python
# Simple scope example

def greet(name):
    print(name)

greet('John')

# Uncomment and see what happens
```
Creating an add function that can be controlled externally

Two approaches:
1. Global variables
2. Closures
Global variables – simple but verbose

```python
data = [1, 2, 3]

def add(x):
    global VALUE
    return x + VALUE

VALUE = 10
print map(add, data)

VALUE = 100
print map(add, data)
```

```
[11, 12, 13]
[101, 102, 103]
```
Closures – elegant but abstract

```python
def adder(value):
    def add(x):
        return x + value
    return add

func = adder(1)
print type(func)
print func(100)
```

A function defined inside another function. It returns the new function!
Closures – in practice

```python
data = [1, 2, 3]

def adder(value):
    def add(x):
        return x + value
    return add

print map(adder(10), data)
print map(adder(100), data)
```

```
[11, 12, 13]
[101, 102, 103]
```
Back to the original problem
what is the index of an element

```python
import pandas as pd
import numpy as np

data = ['ID_REF', 'VALUE', 'CH1_SIG_MEAN', 'CH1_BKD_MEAN', 'CH2_SIG_MEAN', 'CH2_BKD_MEAN']

print(data.index('CH2_SIG_MEAN'))
```

```
4
```
Homework: create the bmmb.py module that operates as in the screenshot

```python
import bmmb

values = bmmb.getcolumn('GPL9270/GSM455822.txt', 'CH2_SIG_MEAN')

print values[:10]
```

Takes a filename and column name

No headers
Start with the original module.
Modify it one step at a time

```python
6 def extract(row):
7     return row[2]

9 def getcolumn2(fname):
10    "Returns column 2 from a file"
11    stream = open(fname)
12    reader = csv.reader(stream, delimiter='\t')
13    reader = list(reader)
14    reader = filter(valid, reader)
15    # find the header here
16    # find the column index
17    # inject the column index into the extractor
18    values = map(extract, reader)
19    return values
```
1. Rename w5mid.py to bmmb.py

2. Change the function to take both a filename and a column name parameter

3. Once filtered the remaining data’ (list of lists) first element is the header

4. Find the column index of the column name

5. Make the extract function use this column index

6. Keep only the valid data without the headers

7. Return the data
Homework: extra credit

• Create two different solutions

1. with global variables
2. with closures
Recap and homework

• Today **you’ll** be the computer! Solve every problem on paper first!

• Read the content of the code box and write the output in the output box.

• All programs will produce some useful output. Some may raise errors at some stages of the program execution.
You may write types with shortcuts (int, float) rather than <type ‘int’>

Make sure to produce all output
def check(x):
    print type(x)
    return len(x)

data = ['hello', 'world', '!']
print map(check, data)
```python
def shorten(x):
    print type(x)
    return x[-2:]

states = ['OHIO', 'HAWAII', 'NEW YORK']
print map(shorten, states)
```
def simplify(x):
    print type(x)
    return 81

values = ['More', 22, 'cheese', [1, 2]]

print map(simplify, values)
```python
def report(x):
    if type(x) == str:
        return len(x)
    else:
        return x * 10

data = ['YES', 22, 'NO', 1.0]
print(map(report, data))
```
```python
def check(elem):
    print '->', elem
    return elem[-1] > 2

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

print filter(check, data)
```

Remember to produce all output!
```python
def multiply(elem):
    result = elem[0] * 10
    print '->', elem, result
    return result

data = [[1, 2], [3, 4], 99]

print map(multiply, data)
```
```python
def rules(elem):
    print '->', elem
    if type(elem) == list:
        return elem[0] * 10
    if type(elem) == str:
        return elem[::-2]
    return elem * 10

data = [ [1, 2], "HELLO", 0.1, [4, 5] ]
print map(rules, data)
```
```python
def check(elem):
    print 'Filter ->', elem
    return len(elem) < 3

def report(elem):
    print 'Report ->', elem
    return elem[-1]

data = [ [1, 2], [4, 5], "Idaho"]
data = filter(check, data)
print map(report, data)
```
```python
import bmmb

print bmmb.avg(data)
```

Traceback (most recent call last):
  File "C:\cygwin\home\code\km10.py", line 5, in <module>
    print bmmb.avg(data)
  File "C:\cygwin\home\code\bmmb.py", line 20, in avg
    return sum(data, 0.0)/len(data)
TypeError: unsupported operand type(s) for +: 'float' and 'str'
def builder(lower, upper):
    def cutoff(value):
        return lower < value < upper
    return cutoff

myfunc = builder(10, 20)
print myfunc(100)
print filter(myfunc, [1, 10, 15, 20, 30])
```python
data = range(10)
data.reverse()
data = data[1:8:2]
print data
```