Week 5 - Lecture 9

Istvan Albert
Huck Institutes for the Life Sciences
Recap and homework

• Today **you’ll** be the computer!

• 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.
Homework ½ - more details

• No code writing today.

• Ask your neighbors/TA for their opinion on correctness.

• Keep the slides. Show them to lecturer or either TA when you’re done.
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)
```
def shorten(x):
    print type(x)
    return x[-2:]

states = ['OHIO', 'HAWAII', 'NEW YORK']

print map(shorten, states)
```python
def simplify(x):
    print type(x)
    return 81

values = ['More', 22, 'cheese', [1, 2]]
print map(simplify, values)
```
KM 4 – halfway

def report(x):
    if type(x) == str:
        return len(x)
    else:
        return x * 10

data = ['YES', 22, 'NO', 1.0]
print map(report, 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)
```
Week 5 - Lecture 10

Istvan Albert
Huck Institutes for the Life Sciences
The Zen of Python

Write and run: `import this`
Namespaces and Scope

- Used when **resolving names** → variable names, function names exist in a **namespace** (scope)

- Python first searches the current **local scope**

- If there is no match → it expands the search in the outer, enclosing scopes until it reaches the **global scope**

- Functions and modules create their own local namespaces
Note: global name error
Global names will be visible inside the function

```python
# Define the greeting variable
greeting = "Hello"

# Define the report function
def report(name):
    print(greeting, name)

# Call the report function
report('World!')
```
This works too ... unfortunately...

By the time the `report()` function is called, the **global name greeting** exists.

**Personal opinion**: global names defined after the function should not leak into the namespaces → can lead to hard to debug errors
Will this work?

```python
def set_greeting():
    greeting = "Hello"

def report(name):
    print greeting, name

set_greeting()
report('World!')
```
A more sensible way to use globals

```python
# make global variables all caps
# declare them as global

def report(name):
    global GREETING
    print GREETING, name

GREETING = "Hello"
report(\'World!\')

GREETING = "Goodbye"
report(\'World!\')
```

These are conventions that help you later understand what is going on.
How do nested scopes work?

```python
def report():
    greeting = 'Hello'
    def say_hello():
        "A local function inside report()"
        print greeting, 'World!'

    # still inside the report function
    say_hello()

report()
```
Why do we need globals

• They can help us access a value across multiple functions

• Some consider global variables to be “impure”, some languages try to forbid their use.

• Yet they can simplify code – although simplicity is in the Eye of the Beholder

• Practicality beats purity – from the Zen of Python.
Two different approaches

• We’ll show code with and without global variables. Both will do the same thing.

• Help me understand which approach do you think makes more sense and is easier to learn.

• Either approach is fine for this class
An example using globals

def remove(x):
    "Removes values below the level"
    global LEVEL
    return x > LEVEL

data = range(10)

LEVEL = 5
print filter(remove, data)

LEVEL = 7
print filter(remove, data)
How to do it without global variables?

• The solutions are considered more elegant, but may also be more abstract.

• Create nested scopes. A function defined inside of another scope.

• You may use either approach.
```python
def factory(level):
    """
    Returns a filtering function that removes values below the level
    """
    def remove(x):
        return x > level
    return remove

data = range(10)
remove1 = factory(5)
print filter(remove1, data)

remove2 = factory(7)
print filter(remove2, data)
```
Comparison: The factory approach produces shorter and reusable functions.
Comparison: The global variable approach makes for more explicit states,
Back to our csv file reader (Week 4)

- Look at homework and data for week 4.

- We will expand this file reader to fetch columns by name

- Always run it on the small dataset (200 lines), when everything works use the large dataset (40,000 lines)
Recall the module week4.py

```python
import week4

filename = 'short-expression.txt'
rows = week4.read_tabular(filename)

# rows is a list of list
# each element is a row in the file,
# each element of a row is a column

# the header row is the first row
header = rows[0]
print(type(header), header)
```

```shell
python -tu C:\cygwin\home\ialbert\sources\ialbert-web\ppt\week5\lecture10\hw10.py 2>&1` returned 0
```

```
<type 'list'> ['ID_REF', 'CH1I_MEAN', 'CH2I_MEAN', 'CH1B_MEDIAN', 'CH2B_MEDIAN', 'CH1D_MEAN', 'CH2D_MEAN', 'CH1I_MEDIAN', 'CH2I_MEDIAN', 'CH1B_MEAN', 'CH2B_MEAN', 'CH1D_MEDIAN',
```
A few helpful list methods

```python
data = ['A', 'T', 'G', 'C', 'A', 'A']

# index returns the list index of
# where an element is located
print 'Index for G =', data.index('G')
print 'Index for A =', data.index('A')

# count returns the number of times
# an element occurs in the list
print 'Count of A =', data.count('A')
```

```
Index for G = 2
Index for A = 0
Count of A = 3
```
Find an index in the header column

```python
import week4

filename = 'short-expression.txt'
rows = week4.read_tabular(filename)

# the header row is the first row
header = rows[0]
print header[:10]

# channel 1 intensity median value
colname = 'CH1I_MEDIAN'
print header.index(colname)
```

```
['ID_REF', 'CH1I_MEAN', 'CH2I_MEAN', 'CH1B_MEDIAN', 'CH2B_MEDIAN', 'CH1D_MEAN', 'CH2D_MEAN', 'CH1I_MEDIAN', 'CH2I_MEDIAN', 'CH1B_MEAN']
```
What we really want is this

```python
import week4

filename = 'short-expression.txt'
rows = week4.read_tabular(filename)
vals = week4.get_column(rows, 'CH1I_Median')

print(vals)
```

Even cut off the first element and cast float on all values
We’ll do it in two ways

• With global variables

• With closures

• Please feel free to comment on which approach seems simpler
With global variables in week4.py

Step 1 – an extractor function

We will need an function that returns a certain element (column) by index.

We want to be able to set this index from the outside so it needs to be a global variable name.

By default it extracts the first column (INDEX=0).
INDEX is global in week4.py module.
With global variables in week4.py
Step 2: find the column index

```python
def get_column(rows, colname):
    global INDEX
    header = rows[0]
    INDEX = header.index(colname)
    return map(extract, rows)
```

We will also need a function that can figure out what the index is for a given column name then sets the global INDEX to that value so that the extractor can use it.
With nested scopes:
There is just one step, nest the extractor

The extract function is defined inside the `get_column()` function and therefore sees the variable `index` declared in that function.
For the homework

Modify the get_column() function in the week4.py module to remove the column name and to return float values.
We are interested in the average intensity for measurements where the hybridization actually took place.

1. Find the averages for the median background intensities for both channels (532nm, 637nm)

2. Find the averages of the median feature intensities for the subset of values that are at least twice over the average of the background for each channel.

Run for the small dataset short-expression.txt then the large one gene-expression.txt