Week 9 - Lecture 17

István Albert
Huck Institutes for the Life Sciences
Revisit our bmmb module
Back our code

```python
import bmmmb

fname = '../GPL9270/GSM455823.txt'
rows  = bmmmb.read_tabular(fname)
genes = bmmmb.get_column(rows, 'ID_REF')

print(genes[:10])
```

This module again?
We need the ability to override the function that is applied to each element.
But leave it work the old way
Need to change the bmmb.py module

The default value of the parameter is the float function that we created not so long ago.
Change the mapping step in bmmb.py

```python
# cut off the header
values = values[1:]

# apply the mapping function passed in as a parameter
values = map(mapfunc, values)

return values
```

```python
# Use the mapping function

def avg(data):

    # Command output
    'python -tu C:\cygwin\home\ialbert\sources\ialbert-web\ppt\week9\lecture17\bmmb.py 2> &1` returned 0
```
Verify that it works

```python
import bmmb

fname = '../GPL9270/GSM455823.txt'
rows = bmmb.read_tabular(fname)
genes = bmmb.get_column(rows, 'ID_REF', mapfunc=str)
vals = bmmb.get_column(rows, 'VALUE')

data = zip(genes, vals)

for row in data[:3]:
    print row
```

Command Output:
```
('opuCD', -0.071150778999999997)
('ssuA', -0.067378810999999997)
('yccK', -0.10953863799999999)
```
One more fix - usability

import bmbb

fname = '../GPL9270/GSM455823.txt'
rows = bmbb.read_tabular(fname)
vals = bmbb.get_column(rows, 'SOMETHING')

We want more informative error messages
More helpful error message

```python
import bmbb

fname = '..\GPL9270\GSM455823.txt'
rows = bmbb.read_tabular(fname)
vals = bmbb.get_column(rows, 'SOMETHING')
```
Back to fixing up bmmb.py

Where does this fix go? What function? At what step?
Done fixing up `bmmb.py`

- It is very common to have to go back to change previous code

- The real test of design is just how many lines do we have to redo

- The goal is to keep old code still working correctly
Containers hold various other objects.
We will cover three types: list, set and hash (dictionary)

Containers have different features and performances.
Use the combination of containers that fits the problem.

All containers are iterable!
Container objects

lists:
- ordered (!) → can be sorted
- access elements by indices
- fast access to any index

tuple:
- it is an immutable list (cannot be modified)
New container: set

```
data = ['A', 'B', 'C']
x = set(data)
print type(x)
print x
```

Note the order!
Sets contain unique elements

```python
data = ['A', 'A', 'A', 'A', 'B']
data = set(data)
print(data)

set(['A', 'B'])
```
Original order is not maintained

top = [ 'A', 'B', 'C' ]
mid = set( top )
bot = list( mid )

print top
print bot

Command Output

['A', 'B', 'C']
['A', 'C', 'B']
Set operations: union → |

```python
a = set(range(6))
b = set(range(3, 9))

print 'a = %s' % a
print 'b = %s' % b
print
print 'a union b = %s' % (a | b)
```

```python
a = set([0, 1, 2, 3, 4, 5])
b = set([3, 4, 5, 6, 7, 8])

a union b = set([0, 1, 2, 3, 4, 5, 6, 7, 8])
```
Set operations: subtraction → -

```python
a = set(range(6))
b = set(range(3, 9))

print 'a = %s' % a
print 'b = %s' % b
print
print 'a - b = %s' % (a - b)
print 'b - a = %s' % (b - a)

a = set([0, 1, 2, 3, 4, 5])
b = set([3, 4, 5, 6, 7, 8])

a - b = set([0, 1, 2])
b - a = set([8, 6, 7])
```
Set operations: intersect → &
Set is an iterable – different order

```python
vals = range(-3, 3)
data = set(vals)
print(len(vals))
print(vals)
print(len(data))
print(map(int, data))
```

Command Output:
```
6
[-3, -2, -1, 0, 1, 2]
6
[0, 1, 2, -2, -3, -1]
```
Framing set operations as biological questions

Assume that $G \rightarrow$ is a set of the names of all genes and

$A \rightarrow$ is a set of genes expressed in sample 1
$B \rightarrow$ is a set of genes expressed in sample 2

• Expressed in **both samples** $\rightarrow$ $A \cap B$ (intersect)

• Expressed in **at least one sample** $\rightarrow$ $A \cup B$ (union)

• Expressed in **B but not A** $\rightarrow$ $B - A$ (subtract)

• **Not expressed in any sample** $\rightarrow$ $G - (A \cup B)$
Homework 1/3

• Send in your project proposal. Slides 6 and 7 in Lecture 1 (see website)

• Apply the described corrections to bmmb.py

Homework continues on next slide
Homework 2/3

• Read the gene names from the file GPL9270/GSM455823.txt (see next slide for a code example for the start of the homework)

• Create separate sets for genes that start with the letter a and b
  • Create separate sets for genes that end with the letter A and B

1. Find the number of genes that start with a or b and end with A
2. Find the number of genes that start with a and end in A or B
3. Find the number of genes that do not start with a or b and do not end with A or B

It is possible to solve this problem by creating a separate filter for each question.

Solve it with sets to see how elegant and simple is to do so
```python
import bmm

fname = '../GPL9270/GSM455823.txt'
rows = bmm.read_tabular(fname)
gen = bmm.get_column(rows, 'ID_REF', mapfunc=str)

def start_filter(letter):
    "Returns a function that filters for the start letter"
    def condition(elem):
        return elem.startswith(letter)
    return condition

gene_a = filter(start_filter('a'), genes)
gene_b = filter(start_filter('b'), genes)

print gene_a[:10]
print gene_b[:10]
```
Dictionary Containers

key → value pairs
Dictionary – storage that permits very fast retrieval by any object

- Lists store objects at **integer indices**: index → value
- Sets store **unique elements**
- Dictionaries store objects in various keys: key → value
A listing of the data structures

```python
# data structures
a = list()
b = tuple()
c = set()
d = dict()

for x in (a, b, c, d):
    print(type(x), x)
```

Command Output:

```
<type 'list'> []
<type 'tuple'> ()
<type 'set'> set([])
<type 'dict'> {}
```
# shorthand notations for data structures

# no shorthand for set? awww, let's give it a hug

```python
a = []
b = ()
c = set()
d = {}

for x in (a, b, c, d):
    print(type(x), x)
```

```
<type 'list'> []
<type 'tuple'> ()
<type 'set'> set([[]])
<type 'dict'> {}
```
Creating dictionaries \( \Rightarrow \{ \} \)

```python
data = {}
vals = ["A", "B", "C"]

for key in vals:
data[key] = key + key + key

print data
print "The value for C is", data["C"]
```

```
{'A': 'AAA', 'C': 'CCC', 'B': 'BBB'}
The value for C is CCC
```
Pre-declare populated dictionaries

```python
data = { "A":100, "B":200, "C":300 }

print type(data), data
print data["A"]
print data["B"]
```

Command Output:
```
<type 'dict'> {'A': 100, 'C': 300, 'B': 200}
100
200
```
Dictionary methods

```python
data = { "A": 100, "B": 200, "C": 300 }

print data.keys()
print data.values()
print data.items()
```

Note the order!
Dictionary order

• Keys and values will be returned in an “random” but identical order

• The order keys and values is the same.

• Not random in the fully unpredictable sense → it depends on the method of construction.

  If you need order → use lists
You may build a dictionary step by step.

```python
names = ["Alice", "Bob", "Zoe"]
ages = [20, 25, 30]

data = {}
for name, age in zip(names, ages):
data[name] = age

print("Dict = ", data)
print("Bob's age is ", data["Bob"])
```

Command Output:
```
Dict = {'Bob': 25, 'Alice': 20, 'Zoe': 30}
Bob's age is 25
```
Building with collated data

```python
def dict_create1():
    names = [ "Alice", "Bob", "Zoe" ]
    ages = [ 20, 25, 30 ]
    collate = zip(names, ages)
    data = dict(collate)
    print("Collate", collate)
    print("Dict =", data)
    print("Bob's age is", data["Bob"])

dict_create1()
```

Command Output:
```
Collate [('Alice', 20), ('Bob', 25), ('Zoe', 30)]
Dict = {'Bob': 25, 'Alice': 20, 'Zoe': 30}
Bob's age is 25
```
Inverse collation
compare to previous slide

```python
# Create a list of names and ages
names = ["Alice", "Bob", "Zoe"]
ages = [20, 25, 30]

collate = zip(ages, names)
data = dict(collate)

print("Collate", collate)
print("Dict =", data)
print("Who is 25? --->", data[25])
```

```
Collate [(20, 'Alice'), (25, 'Bob'), (30, 'Zoe')]
Who is 25? ---> Bob
```
What can be stored as keys, values?

- **Anything** can be a value \(\rightarrow\) string, list, set, another dict

- Only **immutable objects** may be keys \(\rightarrow\) strings, numbers, tuples, immutable sets

The keys in a dictionary are unique \(\rightarrow\) works like a set, a key may not change while stored in the dictionary
Nested dictionaries – whoa!
more on these later

```python
data = dict()

data[100] = [1, 2, 3]
data['ALPHA'] = dict(a=1, b=2, c=3)
data[(1,2,3)] = "What's this?"

for key, value in data.items():
    print(key, '->', value)
```

```
ALPHA -> {'a': 1, 'c': 3, 'b': 2}
100 -> [1, 2, 3]
(1, 2, 3) -> What's this?
```
Get the expression levels for certain genes

```python
import bmbb

rows = bmbb.read_tabular('..\GPL9270\GSM455822.txt')
genes = bmbb.get_column(rows, 'ID_REF', mapfunc=str)
vals = bmbb.get_column(rows, 'VALUE')

# uncomment for sanity check,
# that we indeed get something useful back
# print genes[:2], vals[:2]

genevals = dict(zip(genes, vals))

print 'ssuC ->', genevals['ssuC']
print 'ycdI ->', genevals['ycdI']
```

```
ssuC -> -0.201189818
ycdI -> -0.151834152
```
Homework 3/3

• Do the exercises on the next few slides ➔ predict the output ➔ discuss it with your peers/TA/lecturer

• These are seemingly simple examples but could push your knowledge to the limit
Fill in the blanks - 1

```python
data1 = 
data2 = 

for elem in zip(data1, data2):
    print elem
```

```
(0, 'a')
(1, 'b')
(2, 'c')
```
Fill in the blanks - 2

```python
mapping = 

for value in mapping.values():
    print value
```
Fill in the blanks - 3

data1 = "never gonna give you up, never gonna let you down"
data2 = [4, 8, 15, 16, 23, 42]

for elem in zip(data1, data2):
    print elem
Fill in the blanks - 4

```python
words = "never gonna give you up".split()
print words

values = [4, 8, 15, 16, 23, 42]

print data['give']
```

Command Output:
```
['never', 'gonna', 'give', 'you', 'up']
15
```
Fill in the blanks - 5

```python
data = {'a': 1, 'b': 2, 'c': 3}
```

```
1 a
3 c
2 b
```
Fill in the blanks - 6

```python
data = { 'a': 1, 'b': 2, 'c': 3 }

for key in keys:
    print key, data[key]
```

Tip: get the keys first then sort them

Note that output is sorted by key!
Fill in the blanks - 7

```python
print data[16]
```

```
<module>
    print data[16]
KeyError: 16
```
Fill in the blanks - 8

```
print type(data)
print type(data['A'])
print data['A']['B']
```
Done already?
Get a jump on next week’s homework

- Find the difference in gene expression levels for genes tetB, ycel, ydeG, pstA, trkA, ycgH as represented in files GSM455822.txt and GSM455823.txt

- See next slide for a possible partial solution
Homework example code – could be solved very differently and you are welcome to do it in other ways

```python
# your code goes above, write the get_genemap(fname) function
#
vals1 = get_genemap('./GPL9270/GSM455822.txt')
vals2 = get_genemap('./GPL9270/GSM455823.txt')

names = ['tetB', 'yceI', 'ydeG', 'pstA', 'trkA', 'ycgH']

for name in names:
    diff = vals1[name] - vals2[name]
    print "Change in %s -> %.3f" % (name, diff)
```

```
Change in tetB -> 0.058
Change in yceI -> 0.065
Change in ydeG -> 0.213
Change in pstA -> -0.009
Change in trkA -> -0.162
Change in ycgH -> 0.035
```