Welcome to BMMB 597D

Practical Data Analysis for Life Scientists

Istvan Albert
Huck Institutes for the Life Sciences
Information about the course

• Course webpage: [http://www.personal.psu.edu/iua1/](http://www.personal.psu.edu/iua1/)

• Office hours: MW: 2-3pm (iua1@psu.edu)

• Homework due Thursday the week after it was given out.

• TA: Zhenhai Zhang (zyz5012@psu.edu)
  TA: Zhaorong Ma (zum112@psu.edu)
Rationale for this course

• Life sciences are becoming a data driven science

• Data is represented as text files in various formats

• Computational classes are usually focused on computer science, algorithms.

• We will focus on information processing
Lecture topics

15 weeks – two lectures per week

- 5 weeks - computational foundations
- 3 weeks - biological data formats
- 3 weeks - statistical methods and visualization
- 3 weeks - software tools
- 1 week - presentations

We will discuss multiple issues at once
Lecture formats

• A main topic each week

• Practical examples that tie in with the topic

• Finishing with in class exercises depending on the subject
Project formats

• The best motivator is doing something useful for yourself

• Let’s solve some of your problems

• Send me a proposal on what type of data analysis would you like to see

• We will try to tailor the course to cover those topics in the second half of the semester
1. We have microarray data and sequencing data for a certain organism

2. We’d like to correlate gene expression values with sequencing read coverage around certain genomic markers

Then you need to bring data – either already published ones - or a small percentage of unpublished data (always ask permission from your advisor)
You can only learn by doing

Spend 2-3 hours outside class each week:

– Explore behaviors
– Expand the scope of the study
– Try new solutions

Time flies when you know what you are doing.
Good News

There is no other domain of knowledge where simply

exploring, experimenting,

leads to immediate and dramatic improvement in one’s abilities!
Computation = Thought

- Computational approaches reflect the thought process

- When we learn informatics, we learn how to think in a way that is easy to translate into computation

- There is no magic – it is just like any other subject matter
Computers speak a different language

It is called a **programming** language.

But designed by humans!

No wonder there are thousands of programming languages – each reflecting someone’s ideal on **how things should really** work.
Common traits

A problem that can be solved in one particular programming language can be solved in any other computational language.

Are programming languages all alike?
Not quite the same

“A problem that can be solved in one particular programming language can be solved in any other computational language”

The actual solution may be substantially different

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

The cognitive overhead may be very different
Time to solution

Two things determine time to solution

- How long it takes to create this solution (human time)
- How long it takes that solution to execute (machine time)

Different languages make different tradeoffs

- High-level languages allow individuals to express their thoughts more quickly ...
- ... but the more abstract the language, the more slowly it runs
Food for thought

What makes computers so inhuman?

They always do exactly what we tell them to do

Often we’re not sure what we’ve actually said to the computer.
What is Python

- A simple (but not simplistic!) programming language
- Very popular in certain sciences
- Named after the Monty Python comedy group (not the snake)
What is Python for

• We will use it to interact with data:
  – Extract information
  – Filter datasets
  – Automate tasks
  – Create charts
  – Transform datasets for other bioinformatics tools
This course isn't really about Python
   – It's about solving data analysis problems

But we have to write the examples in something
   – Might as well choose something useful
   – And it puts everyone on an equal footing

We want to build skills that are transferable
Sources of information

All things Python: http://www.python.org

Software carpentry: http://software-carpentry.org/

(some slides were based on material from this lecture series, this lecture series is for those who already know how to program)
Intermezzo

Goal: set up a process where tasks can be performed here in class

The idea is to see right away when something does not make sense
Python fundamentals

Understand the basic building blocks:

1. Every object has a type
2. Every object has a name

Today’s main lesson in one sentence.

When you have a problem, check the type and the name!

1. Unassigned expressions are allowed but in practice we always name them
```python
age = 23
print age
print type(age)
```

```
23
<type 'int'>
```
More types and names

```python
age=23
weight=100.0
prod = age * weight
print age, type(age)
print weight, type(weight)
print prod, type(prod)
```

Command Output:
```
23 <type 'int'>
100.0 <type 'float'>
2300.0 <type 'float'>
```
Unexpected problem. Integer division.

Learn to recognize it! There is a zero in my data soup!
Solution: don’t use integers instead of real (float) numbers

```python
age=23
weight=100.0
prod = age/100.0 * weight
print age/100.0, type(age)
print prod, type(prod)
```

```
0.23 <type 'int'>
23.0 <type 'float'>
```
This you’ll often see this error

```python
greeting="Hello World"
print greetings
```

Traceback (most recent call last):
  File "C:/cygwin/home/ialbert/sources/ialbert-web/ppt/week1/test.py", line 3, in <module>
    print greetings
NameError: name 'greetings' is not defined
Just about all data starts out as a string type!
Some objects already know how to do certain tasks

Python has introspection – we can ask it to tell us what an object can do

1. The function that does this is called `dir`

2. Moreover we can always find more information by invoking the `help`
More about string attributes

Ignore the attributes with underscores for now (not visible here due to large fonts)

These are the attributes of interest:
Check for these attributes on the previous slide

greeting = "Hello World"

print greeting.lower()
print greeting.upper()
This does not work. Why?

```python
import string

# This is a string containing multiple spaces
string = "This is a string with spaces!"

# Trying to replace all spaces with underscores
new_string = string.replace(' ', '_')

print(new_string)
```

```
Traceback (most recent call last):
  File "C:\cygwin\home\ialbert\sources\ialbert-web\ppt\week1\test.py", line 2, in <module>
    print greeting.replace()
TypeError: replace() takes at least 2 arguments (0 given)
```
Find out more about the replace attribute

```python
greeting = "Hello World"
print help(greeting.replace)
```

Help on built-in function replace:

```python
replace(...)  
S.replace (old, new[, count]) -> string
```

Return a copy of string S with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.
Now we know the correct use case

greeting = "Hello World!"
print greeting.replace('World', 'Universe')

Hello Universe!
Lecture 2

Istvan Albert
Huck Institutes for the Life Sciences
More on projects

Two step process:

1. Send me a description of the type of data analysis you are interested in (a few paragraphs) by the first week of October.

2. In the last two weeks give a 5-10 minute presentation describing the data and some of the data analysis methods that have to be performed to extract what you need.

3. We’re not looking for a presentation of the scientific results!
Finalizing your setup

• Let’s make sure everyone can run the examples

TAs to the rescue

• Komodo Edit has huge number of features

Important features:

• Line numbering, syntax coloring, key bindings,
Homework

• By this time next week perform every Python example that was presented in both lectures. Type them in, run them and see the output yourself.

• Honor code based. You can download larger slides from the web (updated screenshots to large fonts)

• Next Thursday turn in a sheet of paper with your name on stating that you’ve done the exercises.
Today’s topic are numbers

• Every data analysis task ends up manipulating numbers in some fashion

• There are some properties that we need to be aware of.

• Most of the time they work as you think they should. When they don’t … very frustrating and makes on feel powerless.
Revisit a previous example

If it leads to such subtle errors why is there a distinction at all? Why can’t we have a single “number” type that always does the “right “ thing
### Short detour – Excel worksheet

Type the formula `=A1+A2-B1-B2` in the box.

What does it show?

Make sure cell number formatting set to "General" (not rounding)
Same problem – now with Python

```python
A1 = 69.82
A2 = 0
B1 = 69.2
B2 = 0.62
print A1 + A2 - B1 - B2
```

What does it show? Tell me.
These are representation errors

Not all numbers are affected
The actual representation for 0.1

0.1000000000000000055511151231257827021181583404541015625

More information details at

http://docs.python.org/tutorial/floatingpoint.html

And many other resources. Not required for the course, more for your personal enlightenment
Our data already has errors

• Multiple orders of magnitude larger than the representation error

• So you are fine. Usually.

There is one rule to remember

Never compare floating point numbers for equality!
Integer comparison are exact

```python
a = 6982
b = 6920
c = 62

print a == b + c
```

Use the ‘==’ as equality operator
Float comparison are not exact

```python
a = 69.82
b = 69.20
c = 0.62

print a == b + c
```

Output:

```
False
```
How do we compare floats?

I know. It seems more work than it should be.
Other traps when using floats

• Care must be taken when combining numbers of widely different magnitudes!

• It is not absolute sizes, but relative ones:
  • 1 vs 1000 billion
  • 1 vs 1000 billionth fraction.
Type the formula =A1+B1-B1 into the box

What does it show?
Same operation with Python

```python
a = 1E6
b = 1E30

print a, b
print a + b - b
```

What does it show? See for yourself.
Integer computation works as expected

```python
# these are comments
# the ** is the power operator
a = 10**6
b = 10**30

# you can put them anywhere
# to document what you are doing
#
print type(a), a, b
print a + b - b
```

```
<type 'int'> 1000000 10000000000000000000000000000000
1000000
```
Long, looooonng integers

```python
# you can make them so huge
# that it cannot finish computing
# in a reasonable amount of time
a = 894**3039
print type(a)
print a
```

```
<type 'long'>
1302325729433506319956158197765966581725227413
00273239931614420614783300267149000023662346604
5488704651700983107551228386304595546772260023
3740906839862262731140840762669756531224198699
6975018460006537926230248992206127422119365819
761503447826432602117002266649809270936633510
2741340547016232643639894655498155239591162347
```
Floating point conclusions

• Don’t test equality for floats! Use the size of the difference instead.

• Don’t put widely different magnitude numbers into the same formula

• Small numbers with small ones, large ones with large ones.

• Most problems naturally lend themselves to such processing. Usually it is us who end up trying to be too smart and “save” some time.
Casting between types

• We often want to transform one type to another type. The process is called: 

  type conversion, typecasting
  usually we just call it casting

• Python makes this simple, use the same name that you found via the type() function
Casting in practice

```python
test.py
a = "123"
print type(a), a

# cast to float
a = float(a)
print type(a), a
```

```
<type 'str'> 123
<type 'float'> 123.0
```
Casting back and forth **may** change values
Some casts may fail. This is **good**!
Homework. Help me cast this string to an integer!

What do I need to have in the code box to get the output below?