Basic programming for drug discovery

Python

Margriet Palm, Brandon Bongers, and Gerard van Westen

14/10/2019

Recap

- Basic python
- Data processing:
  - numpy: basic data processing
  - pandas: structured data, build upon numpy
- Data visualization:
  - matplotlib: customizable plotting
  - seaborn: easy plotting, build upon matplotlib
- Programming tools:
  - jupyter(-lab): interactive coding, data visualization, and annotation

What next

- **Download the notebooks you edited during the course!**
- Continue programming!
- Develop good habits
- Explore and experiment!
- Follow other tutorials
- Learn how to solve problems
- Improve your toolkit
  - useful libraries
  - IDEs
  - version control

Good programming practices

- Why
  - Reliability: does my code what I think it's doing
  - Reproducibility: someone else (including future you) should be able to use the code
  - Maintainability: small changes should require little work
- How
  - Use functions instead of copy and pasting code
  - Use variables instead of hardcoding values
  - Use sensible variable/function names
  - Consistent style (see PEP8)
  - Document with someone else (or future you) in mind
  - Program together; review your colleagues code!
  - Use libraries!!

> *"An hour of searching for software libraries can save you days of programming."*
>
> — Margriet Palm (translated thesis proposition)

GPP - naming variables

Variable name should describe what it holds

- Bad:

```
a = np.random.random(100)

data1 = pd.read_csv('dna_data.csv')
data2 = pd.read_csv('rna_data.csv')
```

- Better

```
rand_nrs = np.random.random(100)

df_dna = pd.read_csv('dna_data.csv')
df_rna = pd.read_csv('rna_data.csv')
```

`snake_case` (all lowercase connected by underscores) is preferred in Python.

GPP - annotation

Annotate complex code

- Bad:

```
a = 10   # variable a
s = 0    # variable s
# for loop
for i in range(a+1):
    s += i   # increase a with 1
```

- (a bit) Better

```
a = 10   # maximum number for range to sum up
s = 0    # placeholder for the sum
# sum up values from 0 to a
for i in range(a+1):
    s += i
```

GPP - Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

GPP - explicit is better than implicit

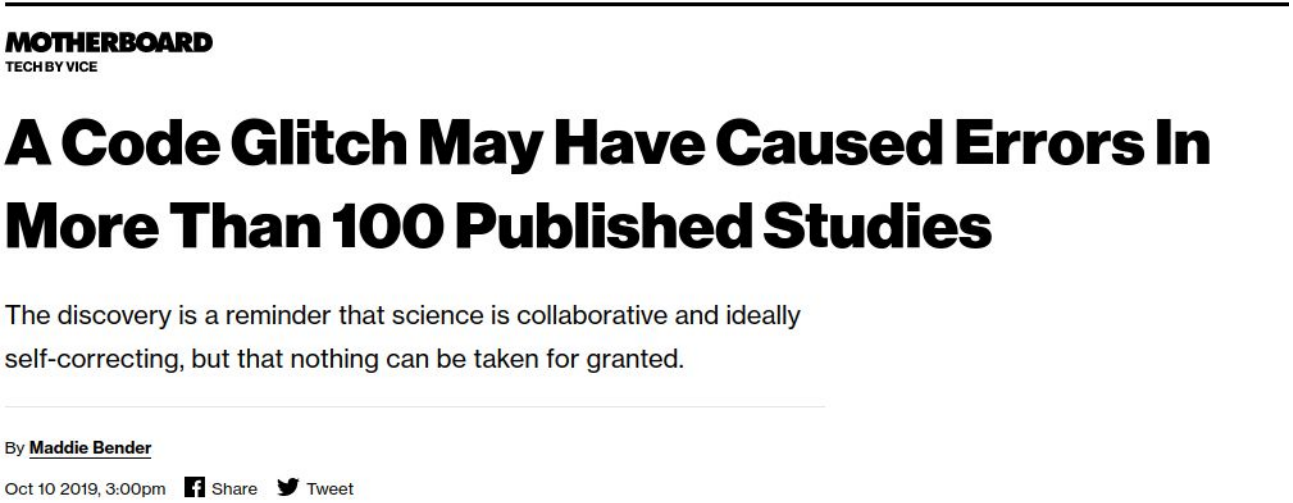
- Bad:

```
plt.plot(df.groupby('time').index,df.groupby('time')['value'])
```

- Good

```
g = df.groupby('time')
plt.plot(g.index,g['value'])
```

GPP - In the face of ambiguity, refuse the temptation to guess



MOTHERBOARD
TECH BY VICE

A Code Glitch May Have Caused Errors In More Than 100 Published Studies

The discovery is a reminder that science is collaborative and ideally self-correcting, but that nothing can be taken for granted.

By Maddie Bender

Oct 10 2019, 3:00pm Share Tweet

Problem: order of list returned by a function differed between windows/linux

> *"Some are reporting this as a glitch in Python, but glob has never guaranteed that is results were returned sorted. As always, I would recommend reading the documentation closely to fully understand what your code does." (Mike Driscoll)*

GPP - Validate, validate, validate, validate, …..

When you can't predict the outcome

[7]:
```
def get_nof_matching_values(a,val):
    return np.sum(a==val)

a = np.random.randint(0,5,(50,50))
print(get_nof_matching_values(a,2))
```

```
246
```

Make a predictable example

[21]:
```
a =  np.linspace(np.arange(5),np.arange(5),5)
print(a)
print(get_nof_matching_values(a,2))
```

```
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
5
```

How to solve problems

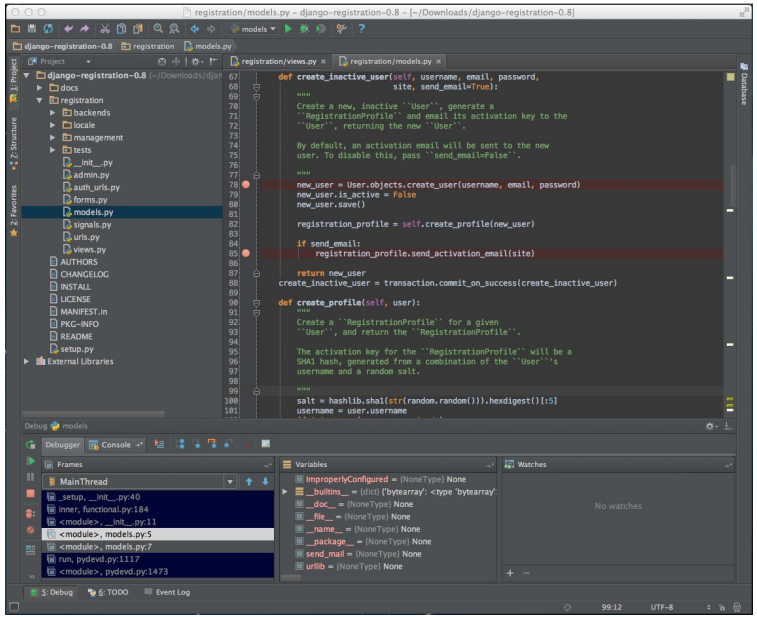
1. Is there really a problem
   - When using a notebook, restart kernel and rerun
   - When using interpreter, close and restart
   - When using a script, make sure you saved it
2. Identify what exactly is going wrong
   - Read error message (if any)
   - Print intermediate results
3. Search for similar issues online
   - Usually, this will bring you to stack overflow
   - Read the comments under the original question
   - Read the answers and look at score and comments
   - Try to understand the solution before implementing
4. Before asking for help if you can’t find the answer.
   - Boil down your code to the essentials of the problem
   - Create a minimal working example (MWE) and test this
   - Be nice to the people that are trying to help you

Toolkit - IDEs

Wikipedia: “An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.”

Why use an IDE:

- Project oriented display of code
- Syntax highlighting
- Autocompletion
- Syntax check for style
- Integration with version control tools
- Debugging tools
- You look like a *real* programmer
- and much more



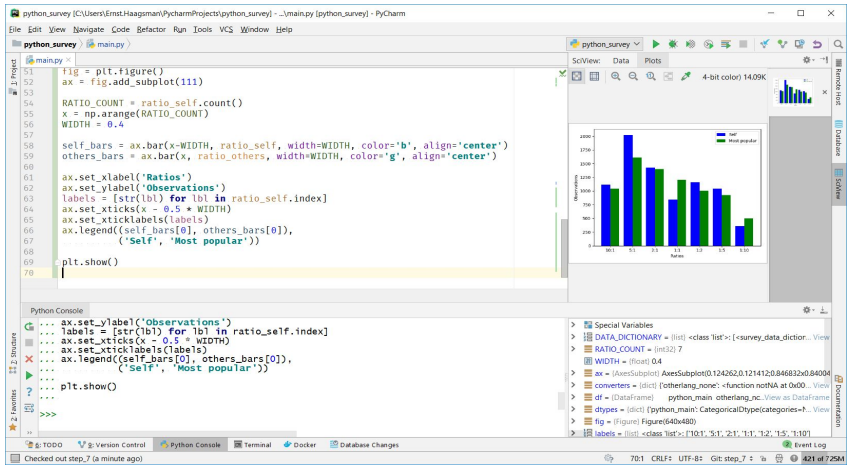
Toolkit - Suggested python IDEs

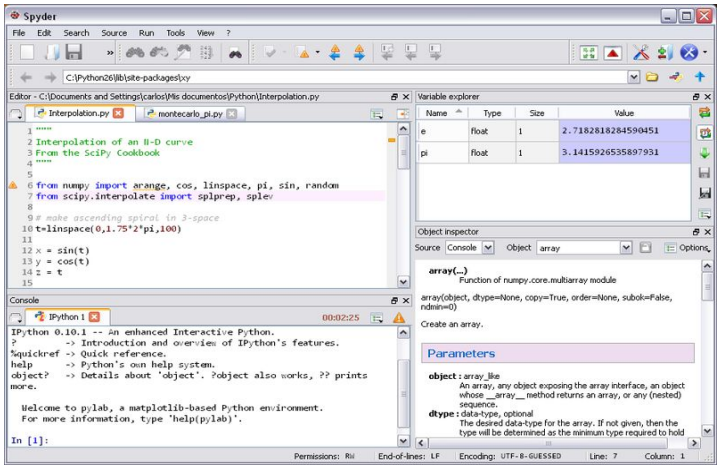
PyCharm

- Developer mode
- Scientific mode (below)
- Free community version + pro version free for academics

Spyder

- Similar to RStudio
- Included in anaconda
- free

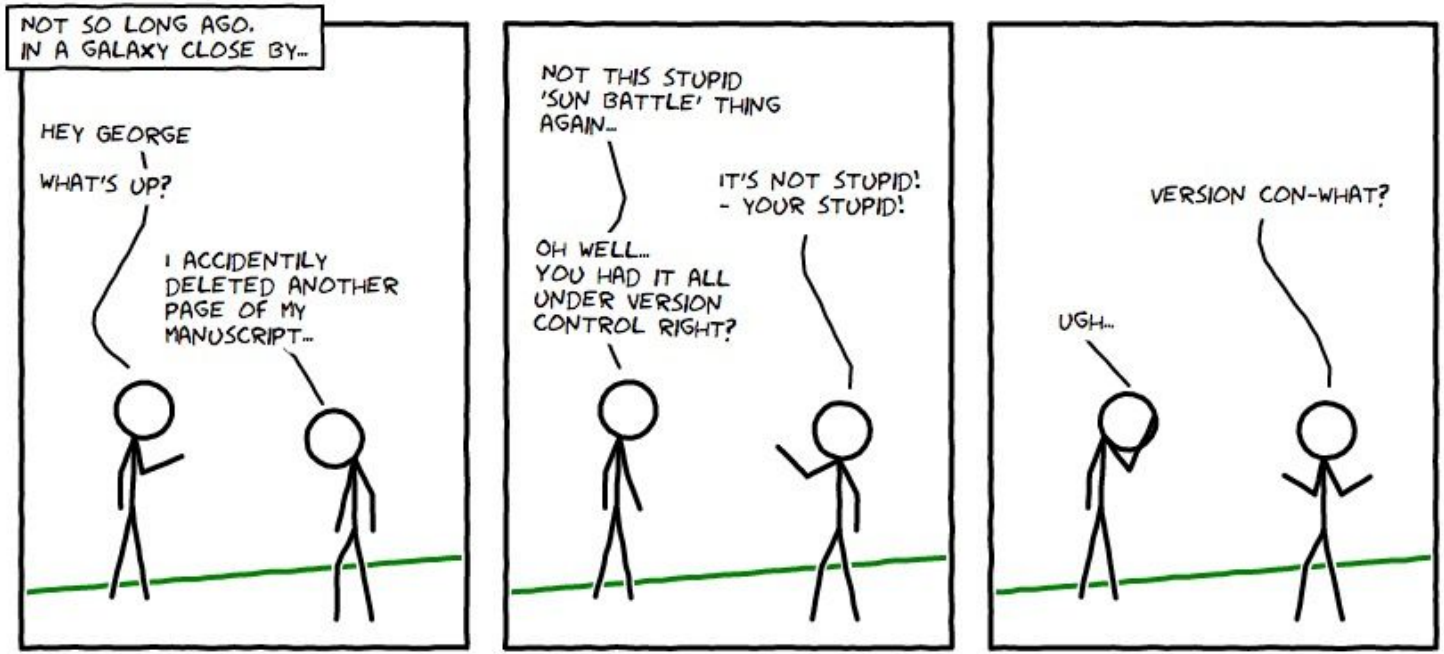




Jupyter notebook and Jupyterlab are also very good tools for data analysis; you don't have to use an IDE.

Toolkit - version control

- Keep track of changes and **why** they were made
- Merge code from multiple developers
- Develop features without breaking the original code
- Go back to previous versions (before you broke your code)



Case study

- See GitHub: https://github.com/lacdr-tox/python-course-materials (see email)
- Description:
  - Data file with data for a series of experiments on cell migration and proliferation
  - Use your Python skills to analyze the dataset:
    - which treatments in which cell line affect migration/proliferation
  - Collect your findings in a nice Jupyter notebook, which should contain
    - different kinds of plots, with proper annotation
    - multiple plots in one figure (subplot)
    - a written discussion of the results (markdown)
- You need:
  - Anaconda on your own computer
  - Seaborn (install via anaconda)
- Deadline: 15-11-19
- Hand in: send notebook (.ipynb) to Brandon (b.j.bongers@lacdr.leidenuniv.nl)