Python tools for MIR research

Alastair Porter
23 May 2019
MIP Frontiers Summer School

We're back!

- Last time: general comments on starting project, writing code, tests, publishing code, dealing with data

- Now: some specific tips on how to organise your python projects better, and to write better* code

* organised, easy to maintain, easy to understand, faster

Overview

- project setup and dependency management
- code best practises for layout and separation
- defensive programming
- strings and files
- data formats
- notebooks
- visualisations
- some other python tips

Other material

- ISMIR tutorial from last year
- MTG python tips: https://mtg.github.io/pymtg/tips/tips.html

- Not in this presentation (No time)
    - Testing
    - Docker
    - More interesting things in the python standard library
    - MIR tools (!)

first things first

don’t use python 2

Setting up projects

- You need to know what dependencies are needed to run your software
- Sometimes a library might change from version to version, keep a record of which one you used
- Different projects of yours may need different versions of libraries, or of python

Installing python dependencies

- You should never need to use sudo to install dependencies with pip

- virtualenv, pipenv, anaconda

virtualenv

-
  ```
  $ virtualenv env
  Using base prefix '/Users/alastair/.pyenv/versions/3.7.2'
  New python executable in /Users/alastair/2019-05-mipfrontiers/
  env/bin/python3.7
  Also creating executable in /Users/alastair/2019-05-mipfrontiers/
  env/bin/python
  Installing setuptools, pip, wheel...
  done.
  ```

- What should your environment be called? It's up to you. I use `env`, others use `ve`

- https://virtualenv.pypa.io/en/latest/

Fully-contained python

-
  ```
  $ ls env/
  bin include   lib
  ```

-
  ```
  $ ls env/bin/
  activate           easy_install       python-config
  activate.csh       easy_install-3.7   python3
  activate.fish      pip                python3.7
  activate.ps1       pip3               wheel
  activate.xsh       pip3.7
  activate_this.py   python
  ```
- Turn on your environment
  ```
  . env/bin/activate
  source env/bin/activate
  ```

pip

-
  ```
  (env) $ pip install numpy matplotlib
   [...]
  Successfully installed cycler-0.10.0 kiwisolver-1.1.0 matplotlib-3.1.0
  numpy-1.16.3 pyparsing-2.4.0 python-dateutil-2.8.0 six-1.12.0
  ```

-
  ```
  $ ls env/lib/python3.7/site-packages/
  __pycache__             matplotlib-3.1.0.dist-info      pyparsing.py
  cycler-0.10.0.dist-info    mpl_toolkits
  python_dateutil-2.8.0.dist-info
  cycler.py               numpy                    setuptools
  dateutil            numpy-1.16.3.dist-info
  setuptools-41.0.1.dist-info
  easy_install.py            pip                   six-1.12.0.dist-info
  kiwisolver-1.1.0.dist-info    pip-19.1.1.dist-info          six.py
  kiwisolver.cpython-37m-darwin.so  pkg_resources              wheel
  matplotlib              pylab.py                 wheel-0.33.4.dist-info
  matplotlib-3.1.0-py3.7-nspkg.pth  pyparsing-2.4.0.dist-info
  ```

why does a venv work?

- If you run pip or python, the version from your virtualenv will be called
- This python has access to all of the packages you installed

how?

- the activate script changes $PATH

-
  ```
  (env) $ which python
  /Users/alastair/2019-05-mipfrontiers/env/bin/python
  (env) $ which python
  /Users/alastair/2019-05-mipfrontiers/env/bin/pip
  ```

What does this mean?

- You can actually run python or pip with a full path, and it will use the dependencies from your virtualenv
- This is really useful when you're calling your python from a script (e.g. on a cluster)

-
  ```
  /scratch/aporter/project/env/bin/python -c 'import numpy;
  print(numpy.array(2))'
  ```

saving and loading dependencies

- `pip freeze > requirements.txt`

-
  ```
  $ cat requirements.txt
  cycler==0.10.0
  kiwisolver==1.1.0
  matplotlib==3.1.0
  numpy==1.16.3
  pyparsing==2.4.0
  python-dateutil==2.8.0
  six==1.12.0
  ```

- `pip install -r requirements.txt`

pipenv

- virtualenv and pip have some problems
  - if you use pip freeze, you don't know if package versions are selected specifically, or if they just came from a dependency
  - some people don't like the behaviour of the activate script
  - virtualenv and pip are two different programs, pipenv does the same as both in one program
- https://docs.pipenv.org/en/latest/

using pipenv

- install once with pip install pipenv, or with homebrew
- `pipenv install numpy`
  - Will automatically create a virtualenv if you don't already have one
  - will create a `Pipfile` (your explicit packages) and `Pipfile.lock` (implicit dependencies, with exact versions)

virtualenvwrapper and anaconda

- https://virtualenvwrapper.readthedocs.io/en/latest/
- Allows you to give names to your virtualenvs, manages the location of them

- https://docs.conda.io/en/latest/
- Dependency and environment management for Python and other languages
- Contains compiled binary packages for a lot of software, including non-python software

let’s write some code

Project structure

- If you're writing a software package to distribute to other people, consider your package name
  - does it already exist on https://pypi.org/ ?
- README.md file, basic outline about what this package does
- License

Package structure

```
mirth/
  __init__.py
  data.py
process.py
README.md
COPYING
setup.py
```

stop

git init

- `git init`
- `git add README.md requirements.txt Pipfile`
- `git commit`
- `git remote add origin`
- `git push -u origin master`

- Don't use `git add .` because you might add items that you don't want in the repository

using git efficiently

- Some types of automatically generated files shouldn't be included in your git repository
  - .pyc files (compiled python code)
  - your entire virtual env
  - .DS_Store (from a mac)
- put these in a `.gitignore` file so that you don't accidentally commit them https://github.com/github/gitignore

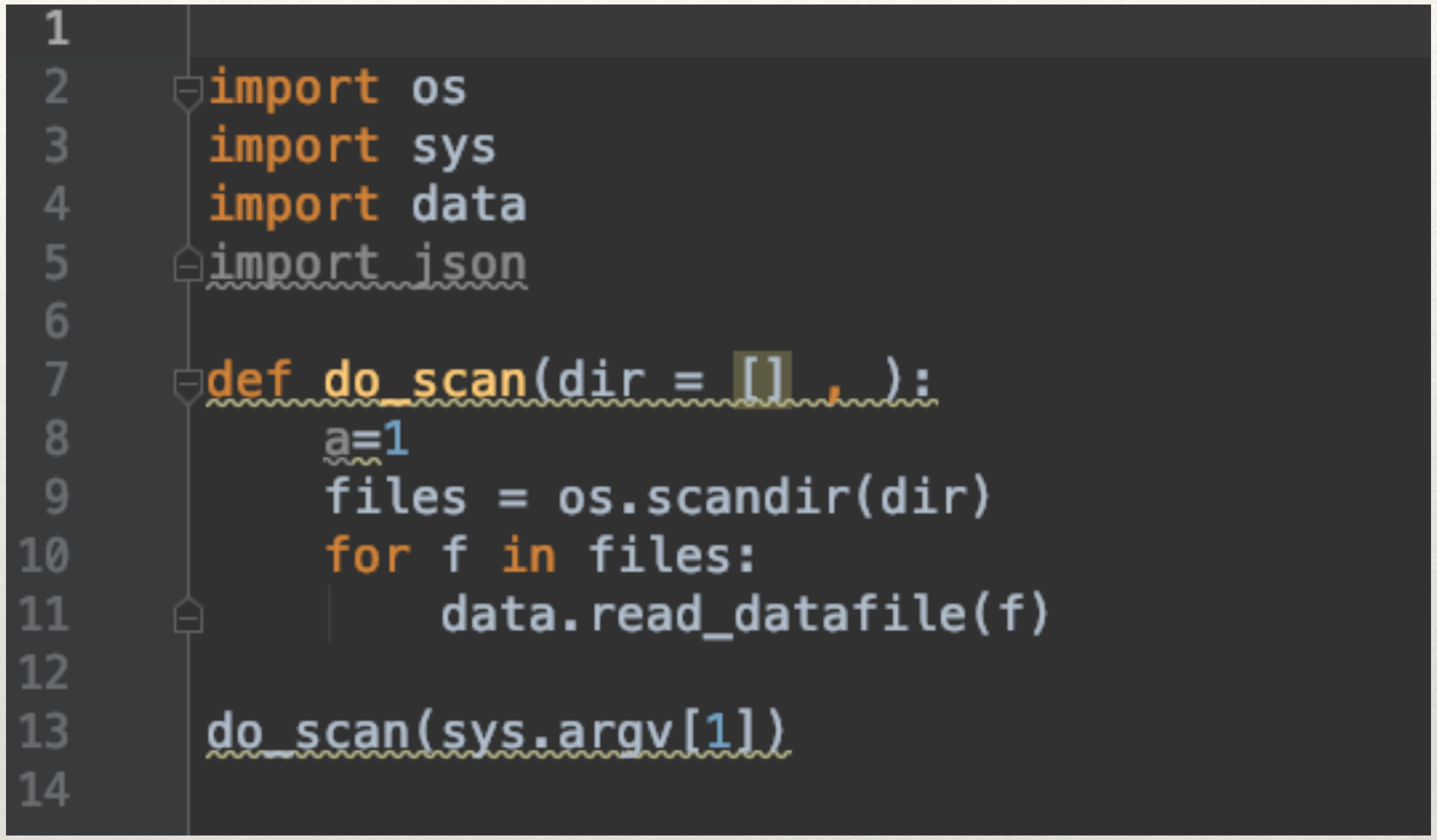
git out of here

- Other types of files you shouldn't include
  - Large data files
    - github has a limit of ~100MB per file, 1000MB per repo
    - If you have small data that you want to include, it's generally OK, but remember that this stays in your git history forever
  - Secrets! Be very careful about access codes (e.g. for AWS). People scan github and will steal your key within *seconds* https://medium.com/@nagguru/exposing-your-aws-access-keys-on-github-can-be-extremely-costly-a-personal-experience-960be7aad039

code layout

- Python doesn't put many requirements on the structure or appearance of your code
- However, consistency in code makes it easier to see patterns, find mistakes
- Choose a style and stick with it, but use tools to help you
- The pep8 styleguide lists some best-practises https://www.python.org/dev/peps/pep-0008/
- black will automatically format your code for you https://github.com/python/black

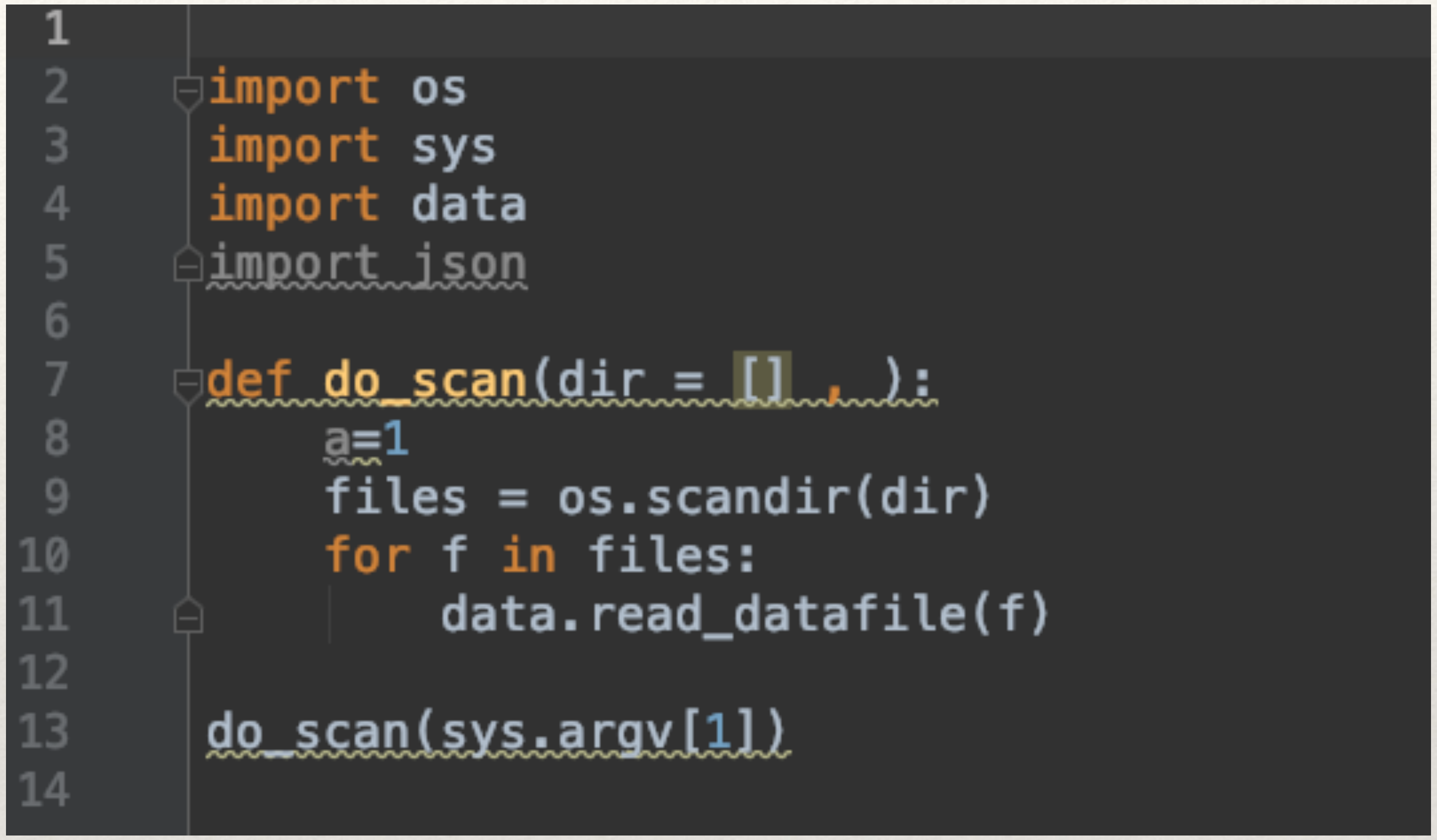
what problems do you see?



```

import os
import sys
import data
import json

def do_scan(dir = [] , ):
    a=1
    files = os.scandir(dir)
    for f in files:
        data.read_datafile(f)

do_scan(sys.argv[1])

```

what problems do you see?

- imports not ordered
- imports not separated
- unused import
- spacing between functions
- extra spaces in fuction def
- no spaces in assignment
- reserved keyword as variable
- use of [] in function definition
- no main guard
- sys.argv instead of argument parser



```

import os
import sys
import data
import json

def do_scan(dir = [] , ):
    a=1
    files = os.scandir(dir)
    for f in files:
        data.read_datafile(f)

do_scan(sys.argv[1])

```

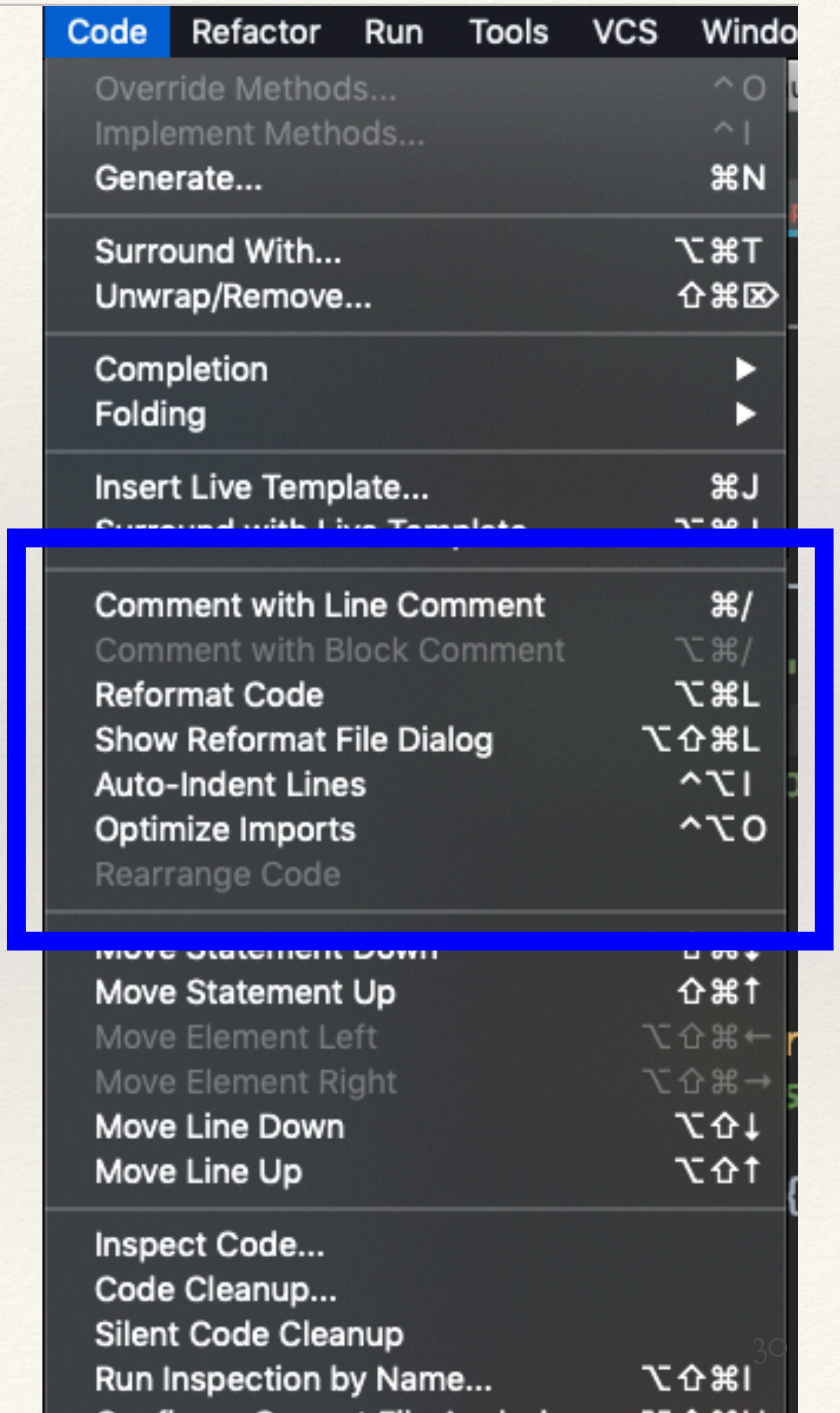
code linting

- the dynamic nature of python makes it easy to make mistakes by missing variables, ordering code incorrectly, or making spelling mistakes
- pylint is a static code checker to look at your code and find common errors
- flake8 integrates pylint checks and pep8 formatting checks: https://flake8.readthedocs.io/en/latest/index.html

```
$ flake8 process.py
process.py:5:1: F401 'json' imported but unused
process.py:7:1: E302 expected 2 blank lines, found 1
process.py:7:16: E251 unexpected spaces around keyword /
parameter equals
process.py:7:18: E251 unexpected spaces around keyword /
parameter equals
process.py:7:21: E203 whitespace before ','
process.py:8:5: F841 local variable 'a' is assigned to but
never used
process.py:8:6: E225 missing whitespace around operator
process.py:13:1: E305 expected 2 blank lines after class or
function definition, found 1
process.py:13:21: W292 no newline at end of file
```

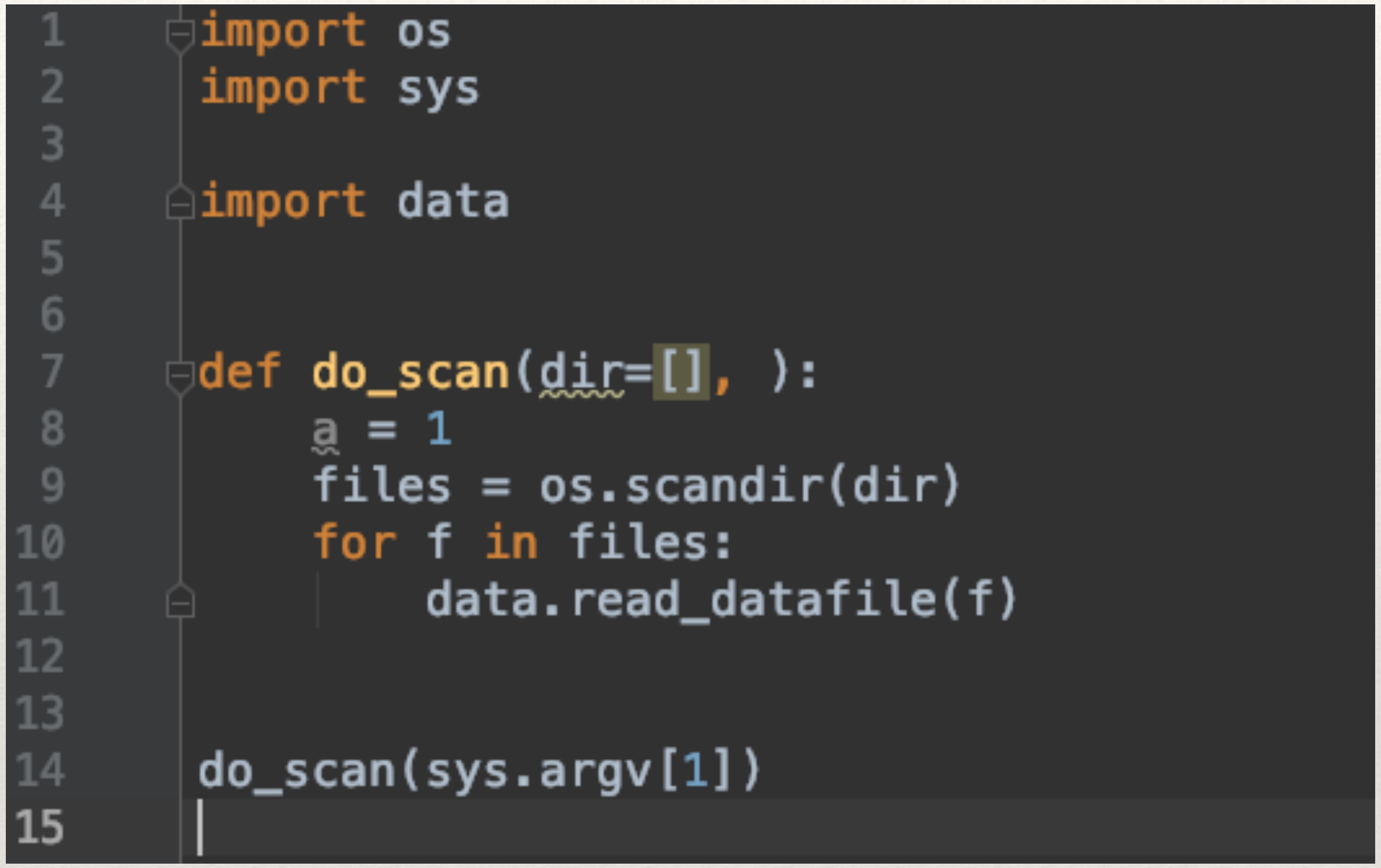
real-time tools for linting and formatting

- pycharm does this for you!



after formatting

- Doesn't rename variables for you



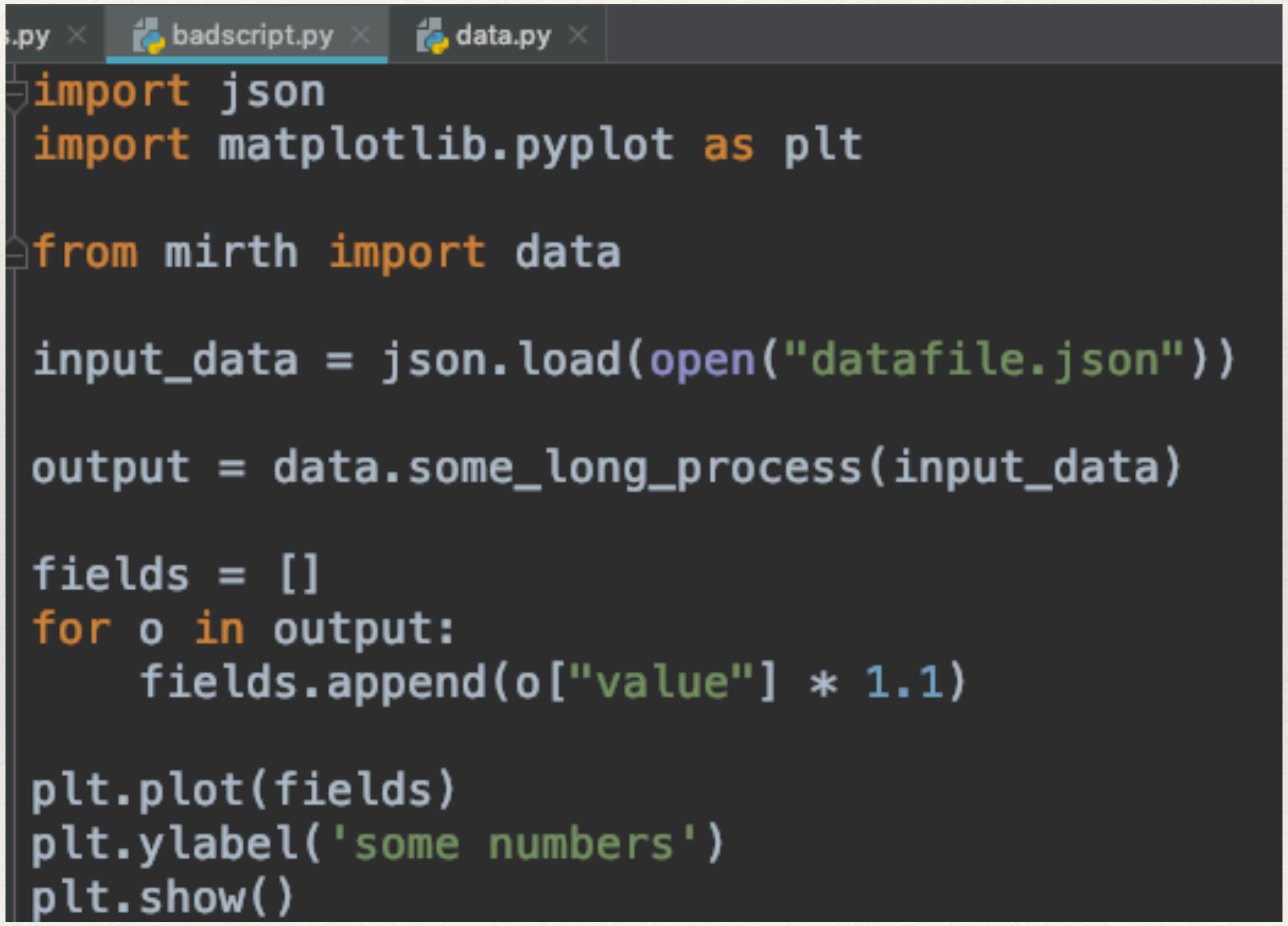
```
import os
import sys

import data


def do_scan(dir=[], ):
    a = 1
    files = os.scandir(dir)
    for f in files:
        data.read_datafile(f)


do_scan(sys.argv[1])
```

learn your tools

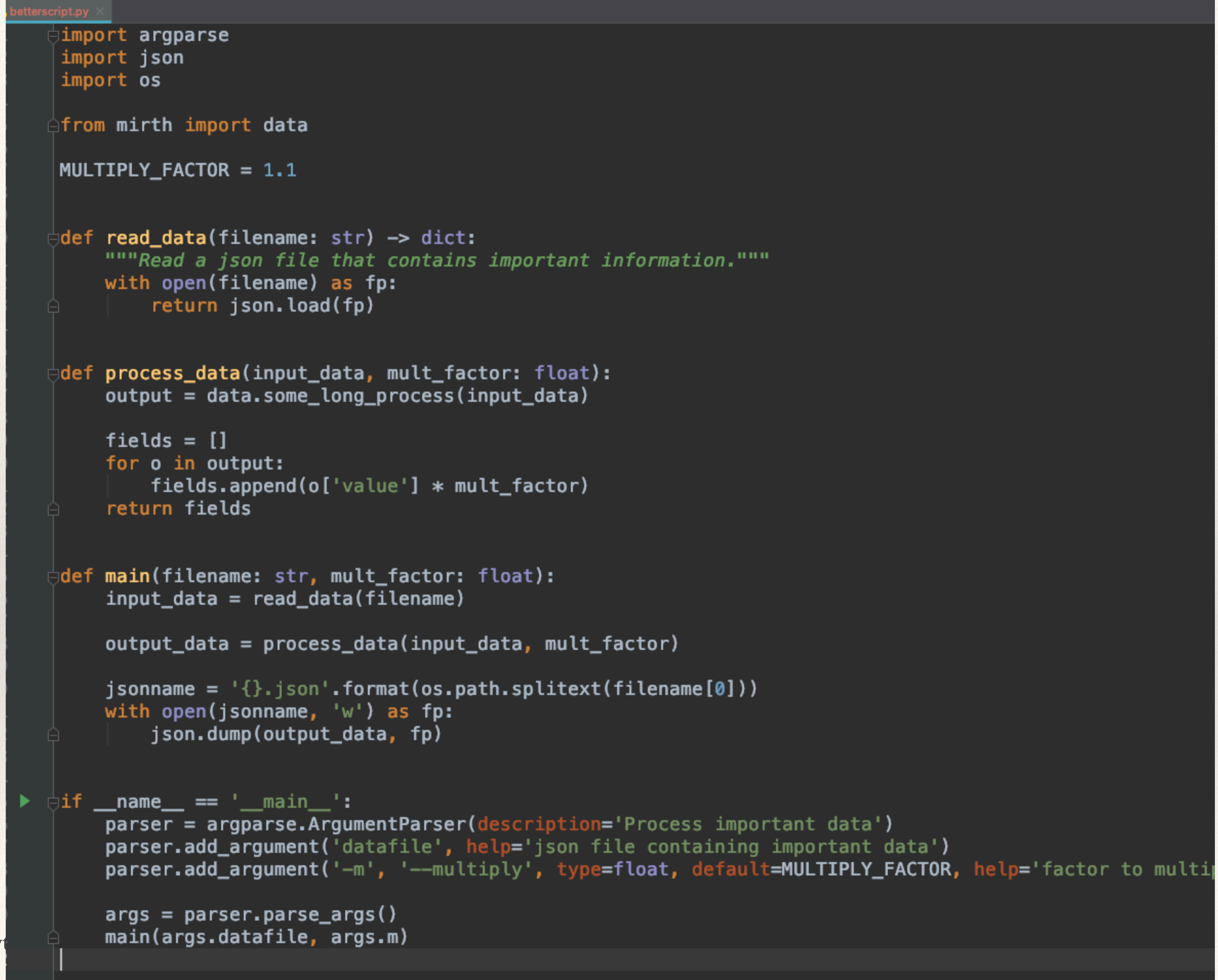
- Find some tutorials for your favourite text editor and learn how to let them help you
- PyCharm:
  - https://twitter.com/pycharm
  - 42 PyCharm tips and tricks: https://www.youtube.com/watch?v=NSuHlqD2y94

scripts

- Don't write all of your script at the same level without functions or a main guard
- Why?
  - if you import this file, it will re-run everything
  - if a processing step fails, you have to start again
  - if you need to change a parameter you have to make many edits in the file



```python
import json
import matplotlib.pyplot as plt

from mirth import data

input_data = json.load(open("datafile.json"))

output = data.some_long_process(input_data)

fields = []
for o in output:
    fields.append(o["value"] * 1.1)

plt.plot(fields)
plt.ylabel('some numbers')
plt.show()
```



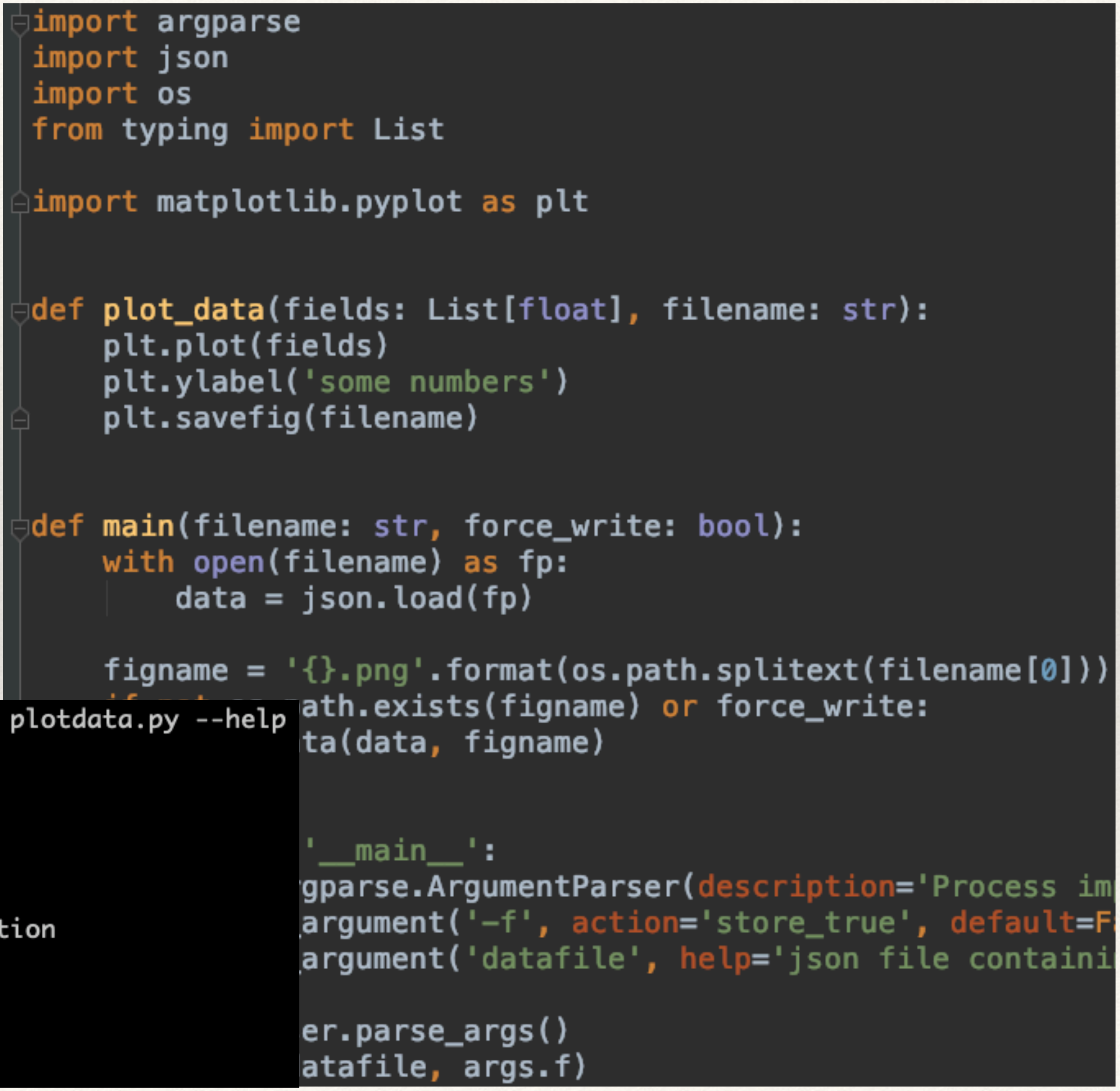
betterscript.py

```
import argparse
import json
import os

from mirth import data

MULTIPLY_FACTOR = 1.1


def read_data(filename: str) -> dict:
    """Read a json file that contains important information."""
    with open(filename) as fp:
        return json.load(fp)


def process_data(input_data, mult_factor: float):
    output = data.some_long_process(input_data)

    fields = []
    for o in output:
        fields.append(o['value'] * mult_factor)
    return fields


def main(filename: str, mult_factor: float):
    input_data = read_data(filename)

    output_data = process_data(input_data, mult_factor)

    jsonname = '{}.json'.format(os.path.splitext(filename[0]))
    with open(jsonname, 'w') as fp:
        json.dump(output_data, fp)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Process important data')
    parser.add_argument('datafile', help='json file containing important data')
    parser.add_argument('-m', '--multiply', type=float, default=MULTIPLY_FACTOR, help='factor to multi

    args = parser.parse_args()
    main(args.datafile, args.m)
```

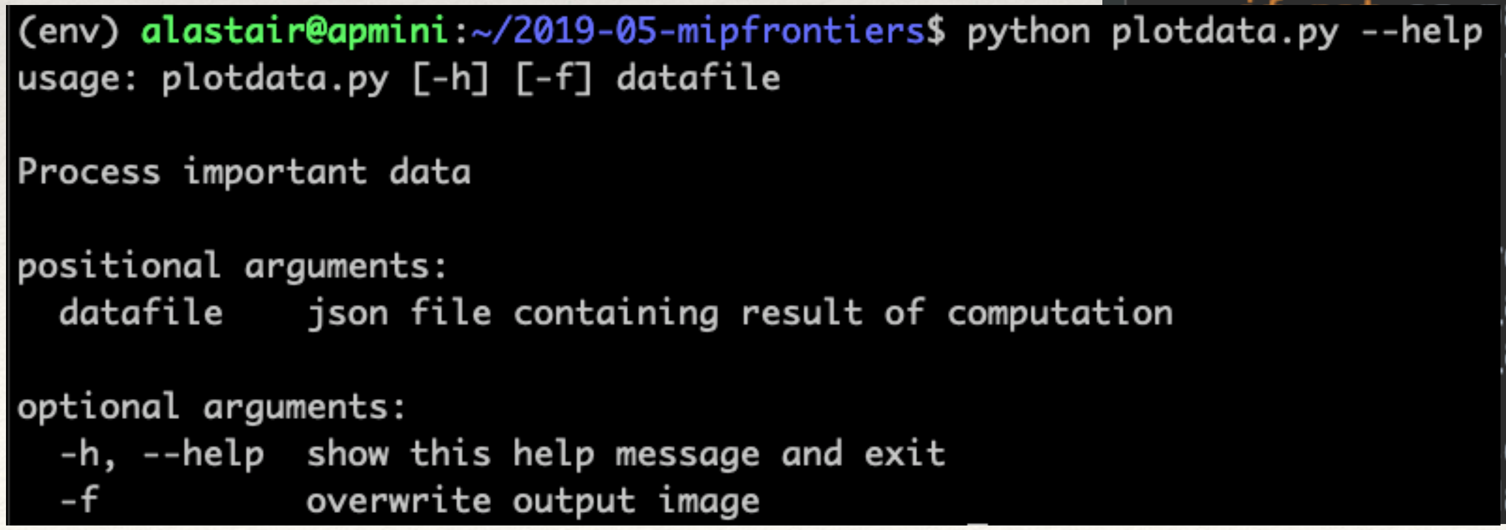
Py

plotting in a separate file

- Save intermediate data
- If you want to change your plot layout you don't need to recompute data



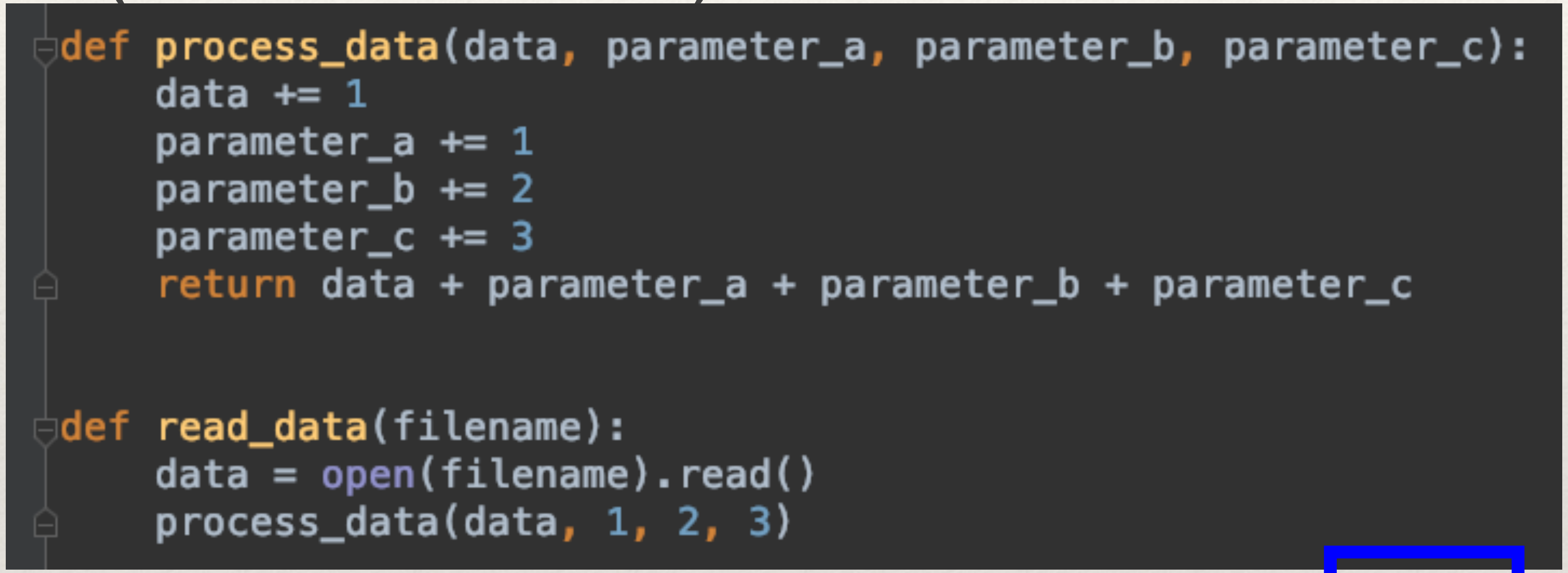
```python
import argparse
import json
import os
from typing import List

import matplotlib.pyplot as plt


def plot_data(fields: List[float], filename: str):
    plt.plot(fields)
    plt.ylabel('some numbers')
    plt.savefig(filename)


def main(filename: str, force_write: bool):
    with open(filename) as fp:
        data = json.load(fp)

    figname = '{}.png'.format(os.path.splitext(filename[0]))
ath.exists(figname) or force_write:
ta(data, figname)


'__main__':
gparse.ArgumentParser(description='Process im
argument('-f', action='store_true', default=F
argument('datafile', help='json file containi

er.parse_args()
atafile, args.f)
```



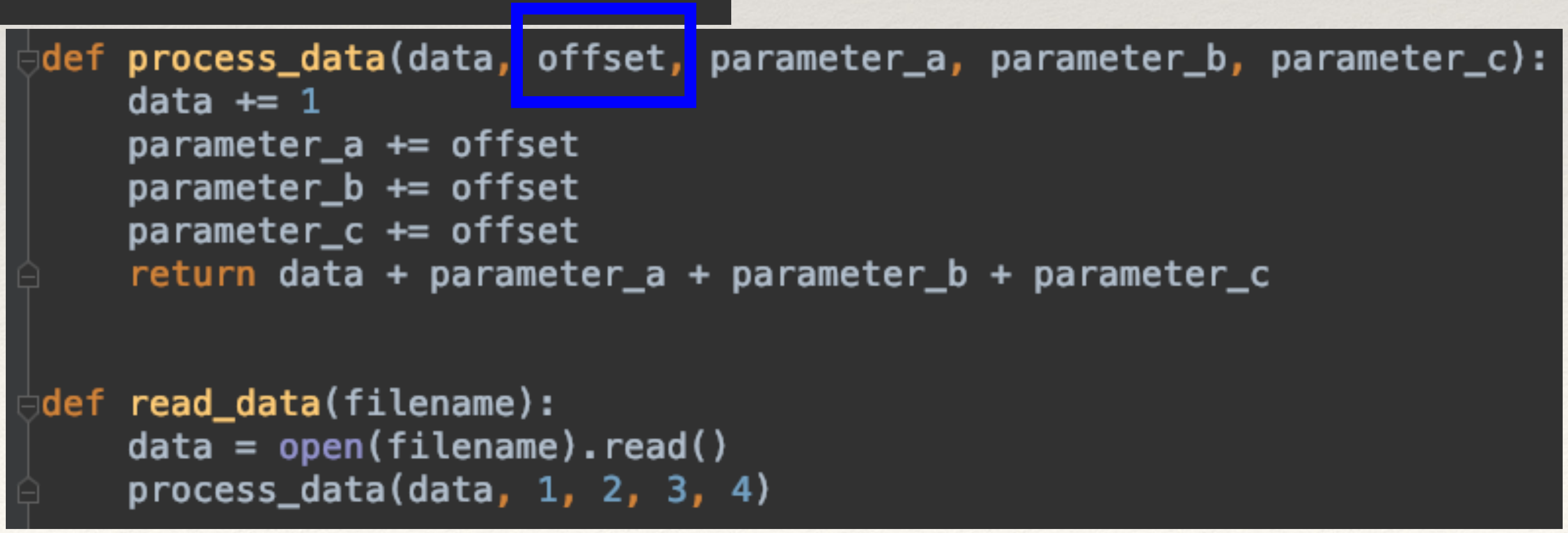
```
(env) alastair@apmini:~/2019-05-mipfrontiers$ python plotdata.py --help
usage: plotdata.py [-h] [-f] datafile

Process important data

positional arguments:
  datafile    json file containing result of computation

optional arguments:
  -h, --help  show this help message and exit
  -f          overwrite output image
```

functions

- how do you remember what parameter is for what? (documentation!)



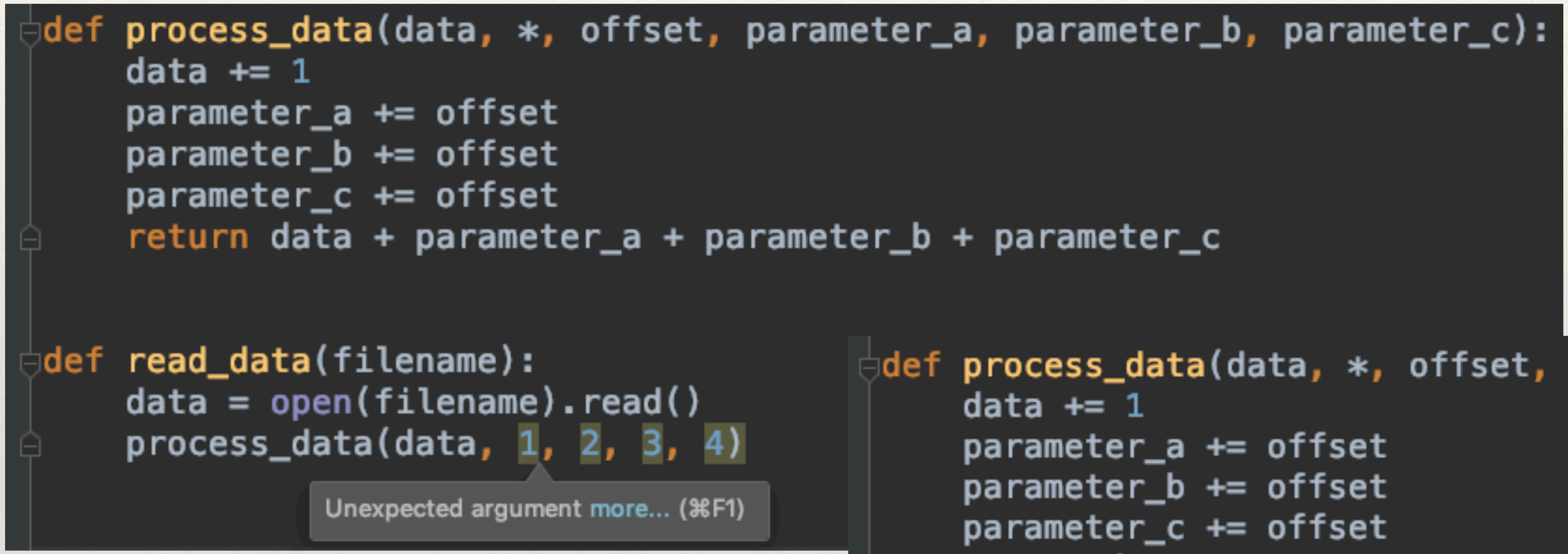
```python
def process_data(data, parameter_a, parameter_b, parameter_c):
    data += 1
    parameter_a += 1
    parameter_b += 2
    parameter_c += 3
    return data + parameter_a + parameter_b + parameter_c


def read_data(filename):
    data = open(filename).read()
    process_data(data, 1, 2, 3)
```



```python
def process_data(data, offset, parameter_a, parameter_b, parameter_c):
    data += 1
    parameter_a += offset
    parameter_b += offset
    parameter_c += offset
    return data + parameter_a + parameter_b + parameter_c


def read_data(filename):
    data = open(filename).read()
    process_data(data, 1, 2, 3, 4)
```

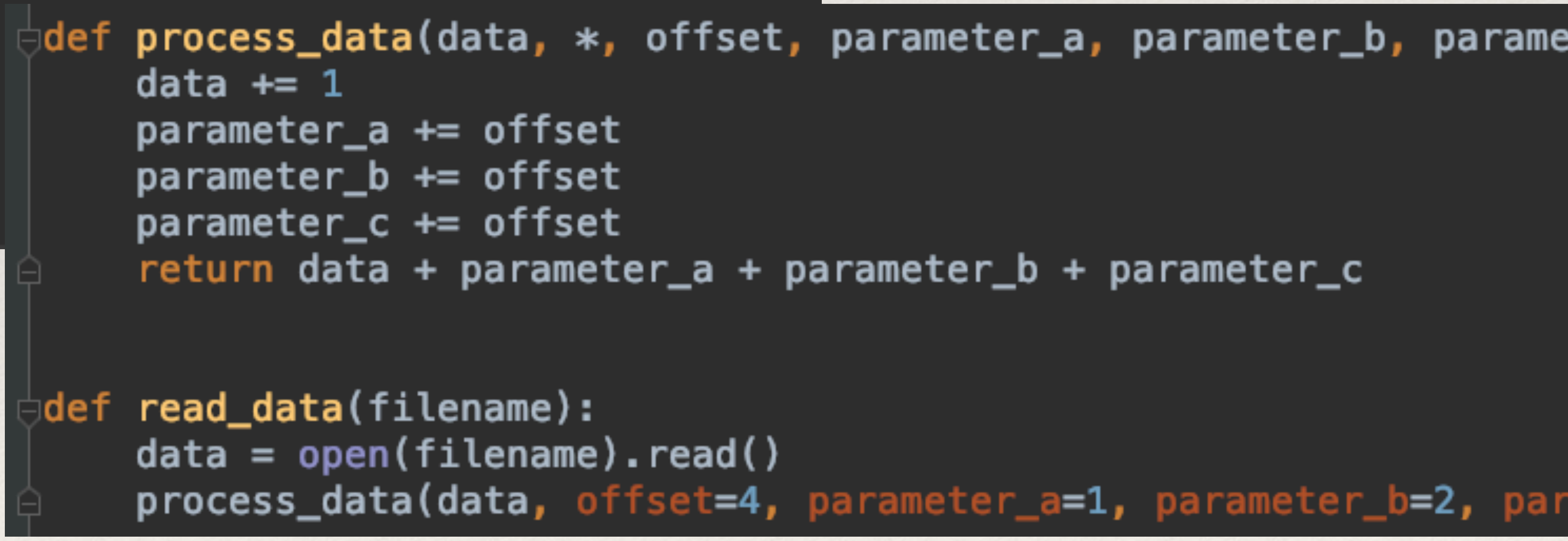
named function parameters

- are parameter names important? Make them required



```
def process_data(data, *, offset, parameter_a, parameter_b, parameter_c):
    data += 1
    parameter_a += offset
    parameter_b += offset
    parameter_c += offset
    return data + parameter_a + parameter_b + parameter_c


def read_data(filename):
    data = open(filename).read()
    process_data(data, 1, 2, 3, 4)
```

Unexpected argument more... (⌘F1)

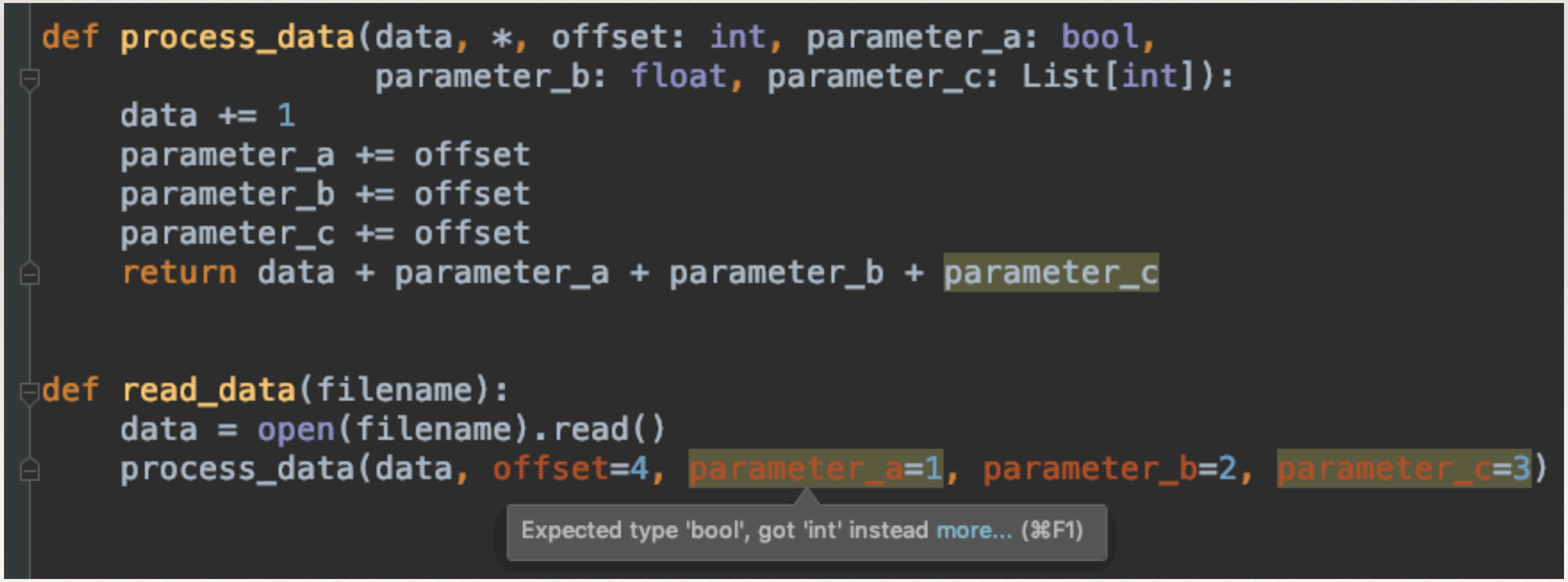


```
def process_data(data, *, offset, parameter_a, parameter_b, parame
    data += 1
    parameter_a += offset
    parameter_b += offset
    parameter_c += offset
    return data + parameter_a + parameter_b + parameter_c


def read_data(filename):
    data = open(filename).read()
    process_data(data, offset=4, parameter_a=1, parameter_b=2, par
```

- https://python-3-for-scientists.readthedocs.io/en/latest/python3_advanced.html

type hints

- Ensure that the values of your parameters are correct
- This is not checked by python when you run it, but external tools (e.g. pycharm) can use it for notifying you of potential errors

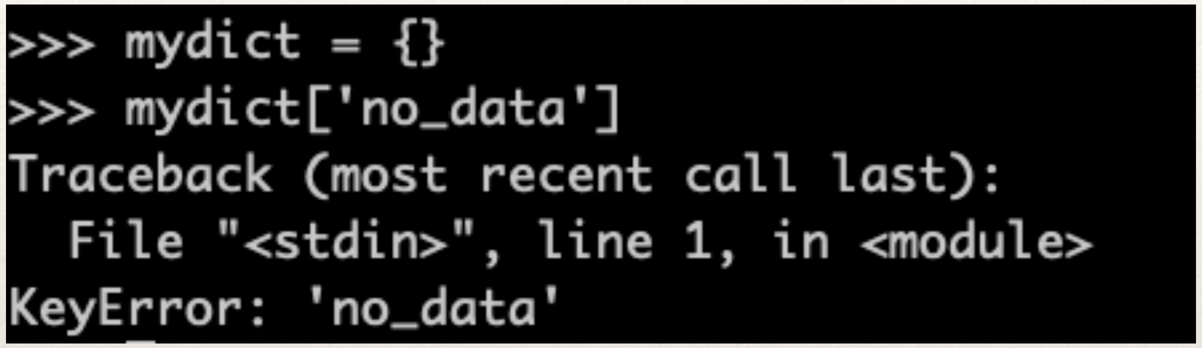


```
def process_data(data, *, offset: int, parameter_a: bool,
                 parameter_b: float, parameter_c: List[int]):
    data += 1
    parameter_a += offset
    parameter_b += offset
    parameter_c += offset
    return data + parameter_a + parameter_b + parameter_c


def read_data(filename):
    data = open(filename).read()
    process_data(data, offset=4, parameter_a=1, parameter_b=2, parameter_c=3)
```

Expected type 'bool', got 'int' instead more... (⌘F1)

defensive programming

- Python's dynamic nature makes it easy to make mistakes



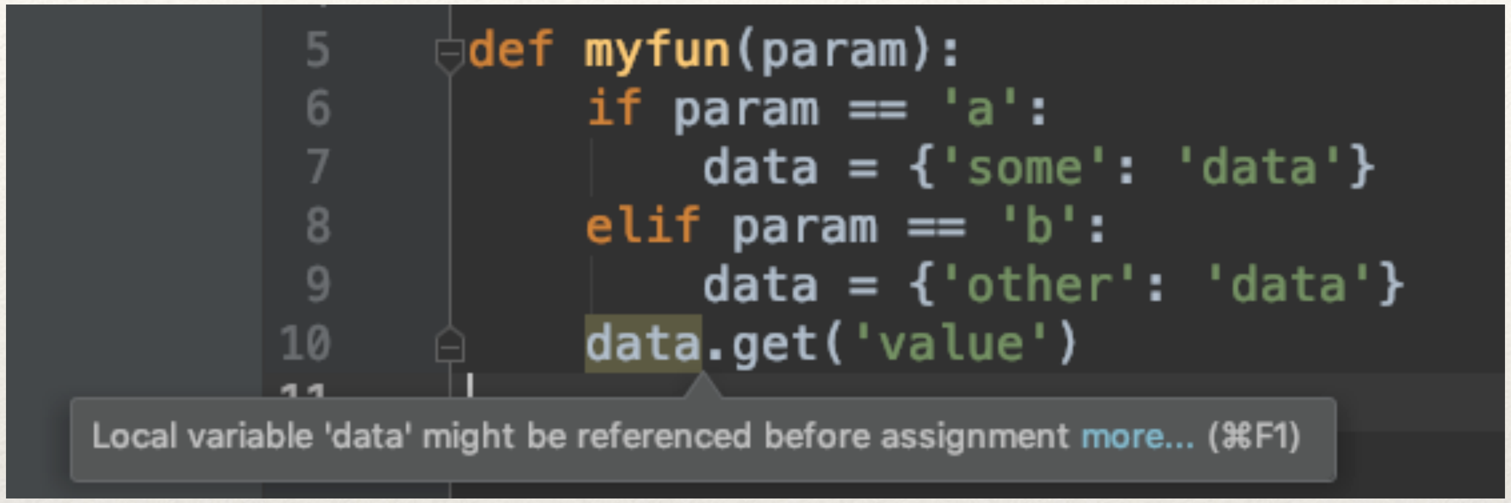
```
>>> mydict = {}
>>> mydict['no_data']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'no_data'
```

- Did you get this data from an external source?
    - first thing to ask is "what happens if this data isn't available?"
    - Are you sure that all items in the dictionary have the key that you want?
- KeyError, AttributeError - imagine if this happens after it's been running for hours or days

defensive programming

- Getting data from the web, what if the site goes down?
    - always consider possible HTTPErrors
- reading files, what if there's a parsing error?
    - json can raise ValueError
- If the file doesn't exist?
    - Or the folder that you want to write something to?
- if you call int() to turn a string into an integer, what if it's not an integer?

defensive programming



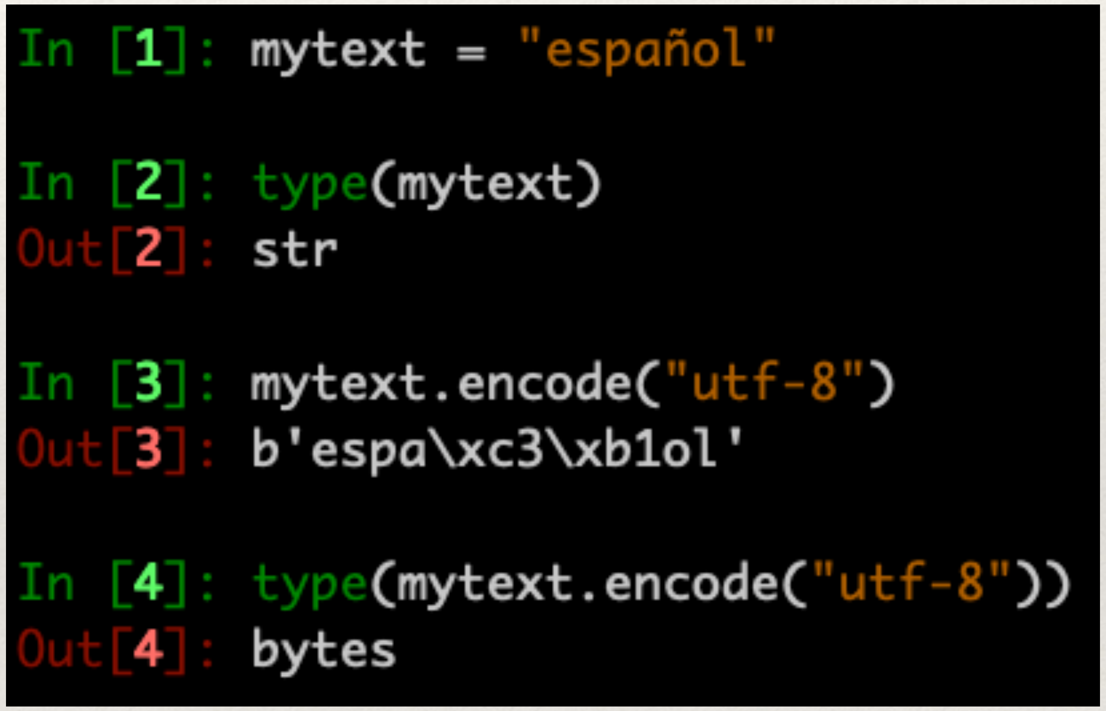
```
def myfun(param):
    if param == 'a':
        data = {'some': 'data'}
    elif param == 'b':
        data = {'other': 'data'}
    data.get('value')
```

Local variable 'data' might be referenced before assignment more... (⌘F1)

- Tools can help
- Think carefully about your data and the operations you are performing on it
- Never trust data from other people (or even from yourself)

strings!

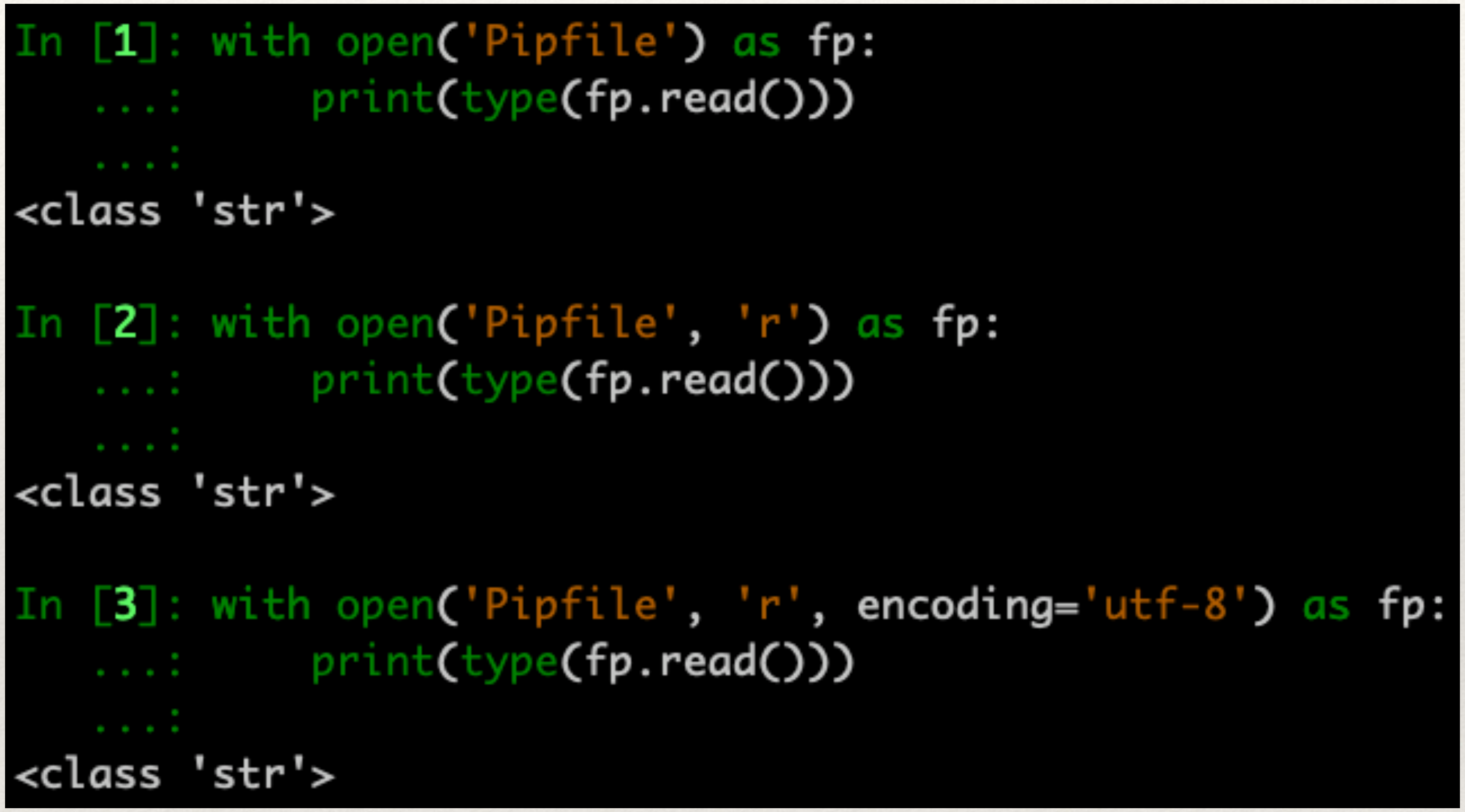
- Lots of people complained about the changes to strings in python 3, but I think it's made things a lot simpler
  - written text is a string
  - when you read and write files, the string is turned into bytes using an *encoding* (e.g. utf-8)
  - some functions implicitly perform this translation for you, but you can force it if you need to



```
In [1]: mytext = "español"

In [2]: type(mytext)
Out[2]: str

In [3]: mytext.encode("utf-8")
Out[3]: b'espa\xc3\xb1ol'

In [4]: type(mytext.encode("utf-8"))
Out[4]: bytes
```

- https://medium.com/@andreacolangelo/strings-unicode-and-bytes-in-python-3-everything-you-always-wanted-to-know-27dc02ff2686

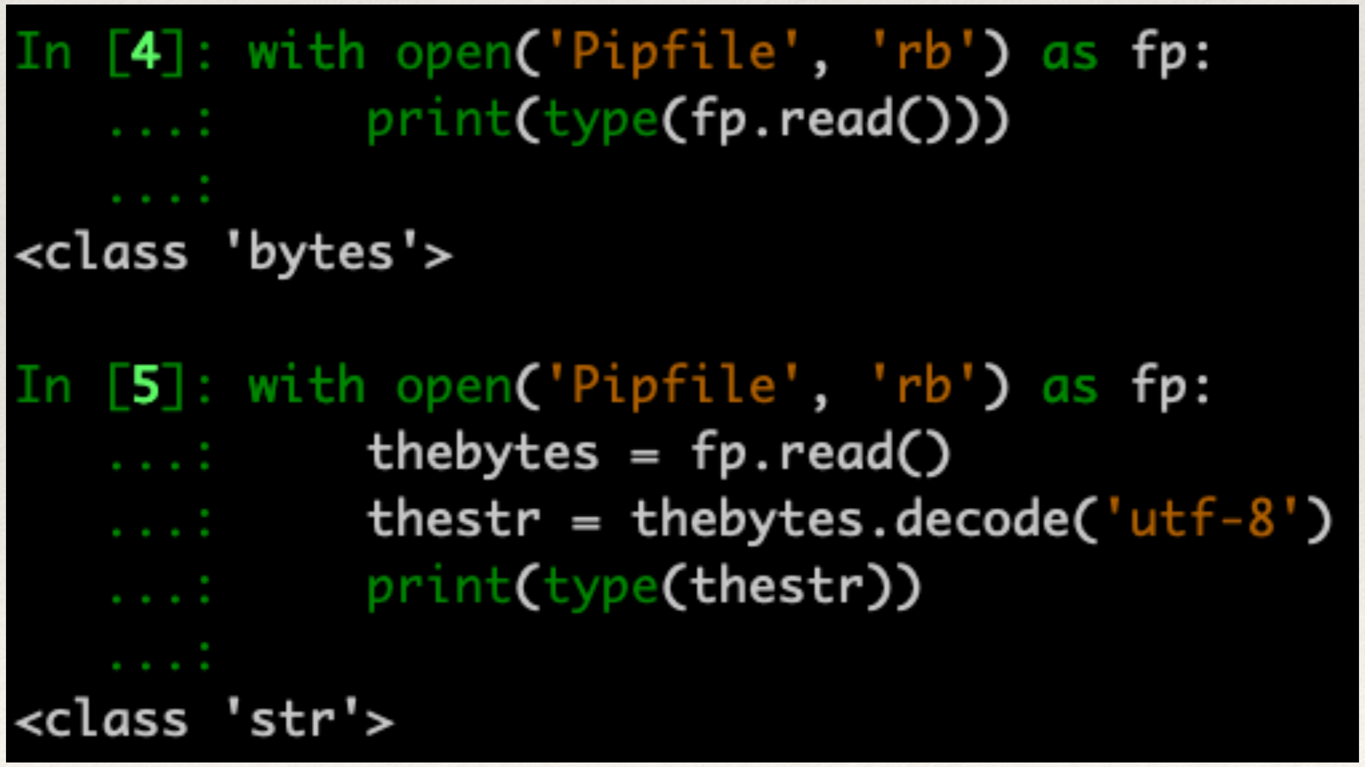
reading strings



```
In [1]: with open('Pipfile') as fp:
   ...:     print(type(fp.read()))
   ...:
<class 'str'>

In [2]: with open('Pipfile', 'r') as fp:
   ...:     print(type(fp.read()))
   ...:
<class 'str'>

In [3]: with open('Pipfile', 'r', encoding='utf-8') as fp:
   ...:     print(type(fp.read()))
   ...:
<class 'str'>
```

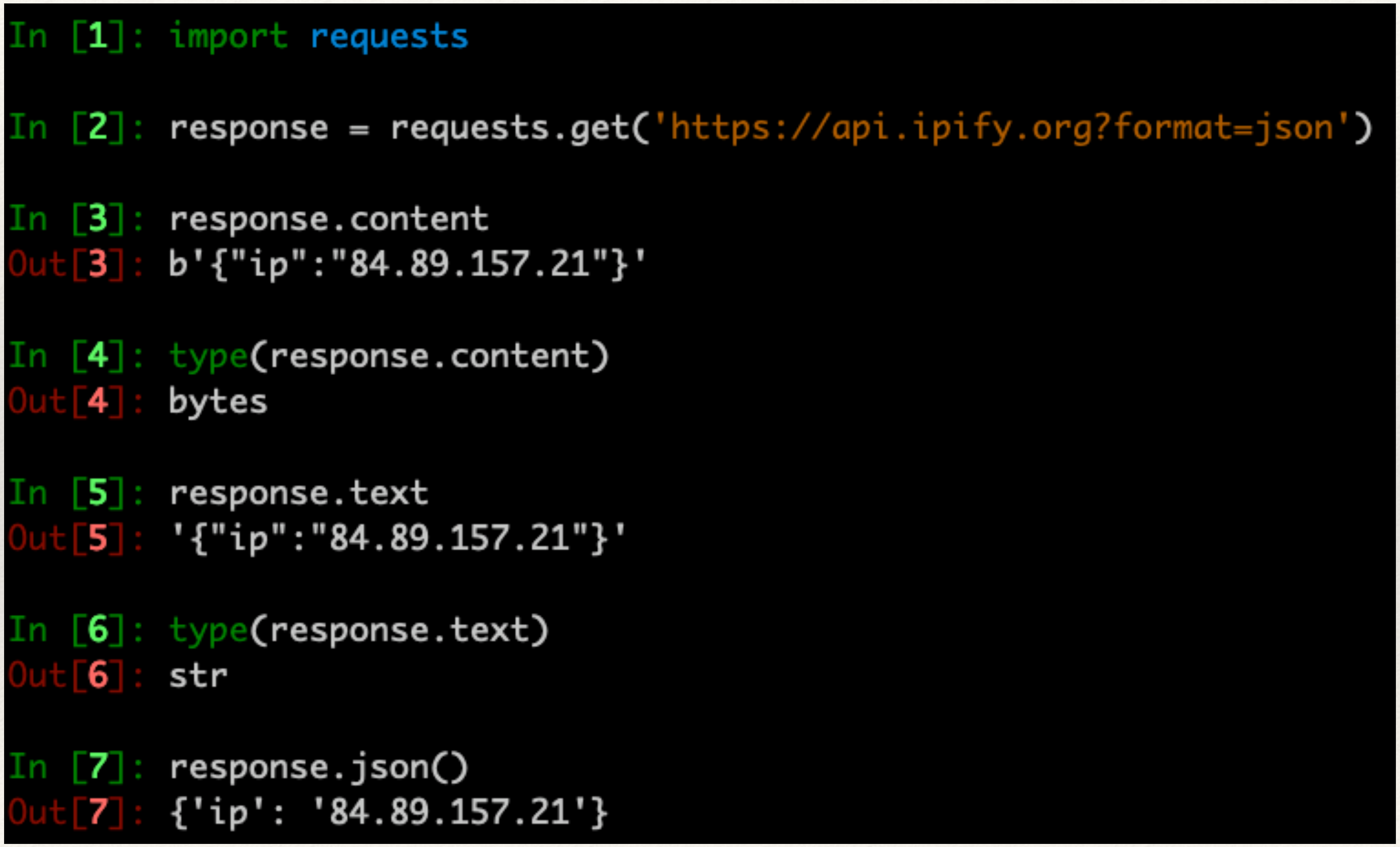
reading bytes

- Why might you want to read bytes?
    - data file that you want to process
    - spectrogram images?
    - manually-encoded data



```
In [4]: with open('Pipfile', 'rb') as fp:
   ...:     print(type(fp.read()))
   ...:
<class 'bytes'>

In [5]: with open('Pipfile', 'rb') as fp:
   ...:     thebytes = fp.read()
   ...:     thestr = thebytes.decode('utf-8')
   ...:     print(type(thestr))
   ...:
<class 'str'>
```

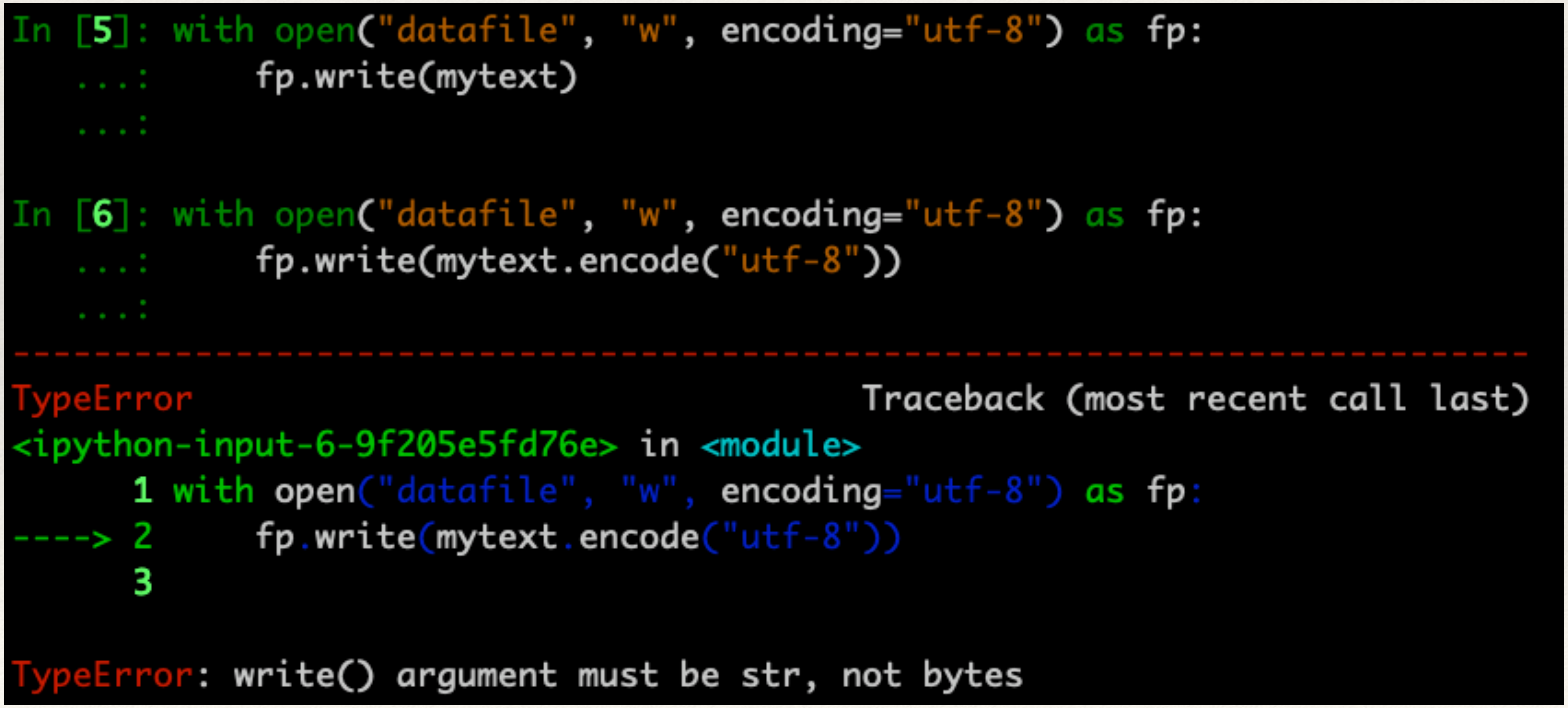
strings from the internet



```
In [1]: import requests

In [2]: response = requests.get('https://api.ipify.org?format=json')

In [3]: response.content
Out[3]: b'{"ip":"84.89.157.21"}'

In [4]: type(response.content)
Out[4]: bytes

In [5]: response.text
Out[5]: '{"ip":"84.89.157.21"}'

In [6]: type(response.text)
Out[6]: str

In [7]: response.json()
Out[7]: {'ip': '84.89.157.21'}
```

writing strings

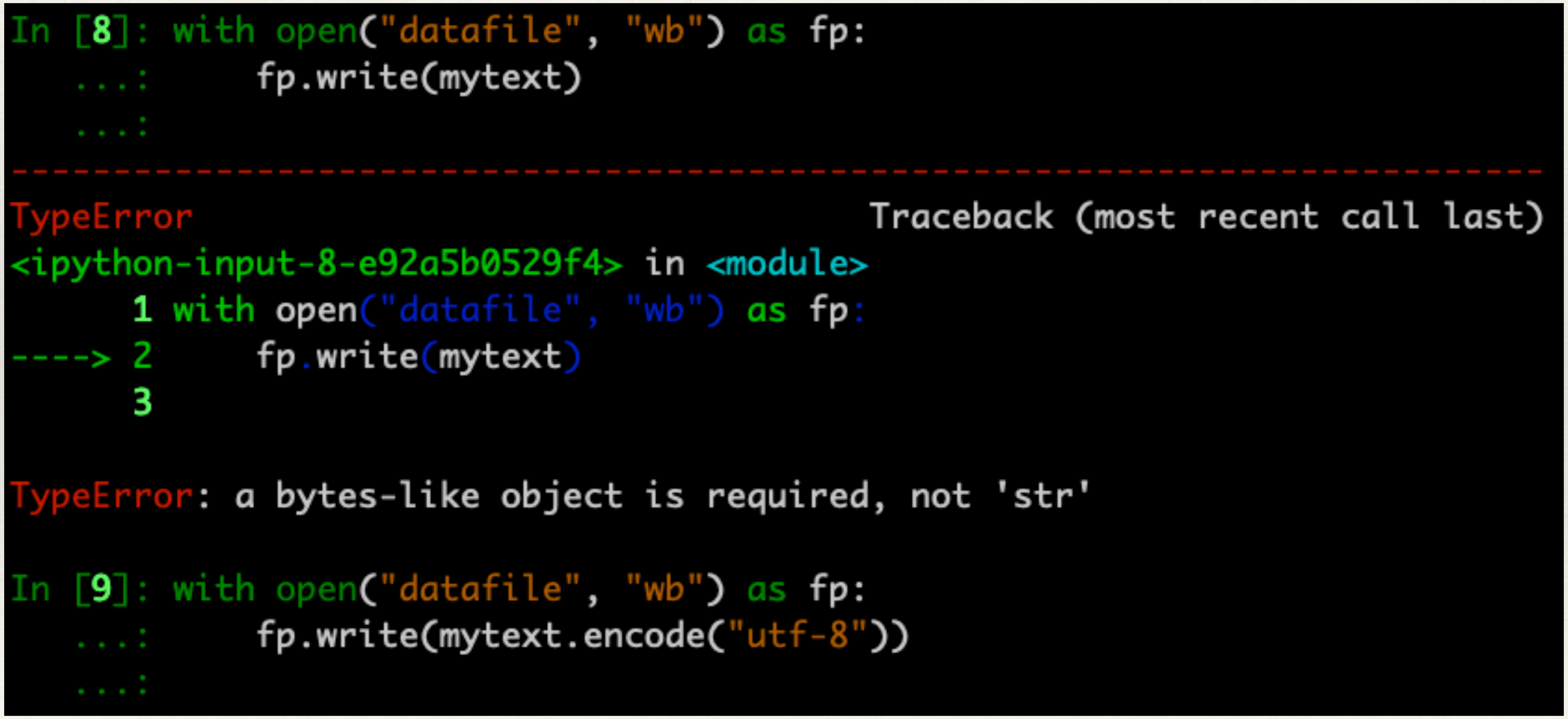
- Reading in reverse!
- Make sure everything is a string and use implicit conversion



```
In [5]: with open("datafile", "w", encoding="utf-8") as fp:
   ...:     fp.write(mytext)
   ...:

In [6]: with open("datafile", "w", encoding="utf-8") as fp:
   ...:     fp.write(mytext.encode("utf-8"))
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-9f205e5fd76e> in <module>
      1 with open("datafile", "w", encoding="utf-8") as fp:
----> 2     fp.write(mytext.encode("utf-8"))
      3

TypeError: write() argument must be str, not bytes
```

writing strings

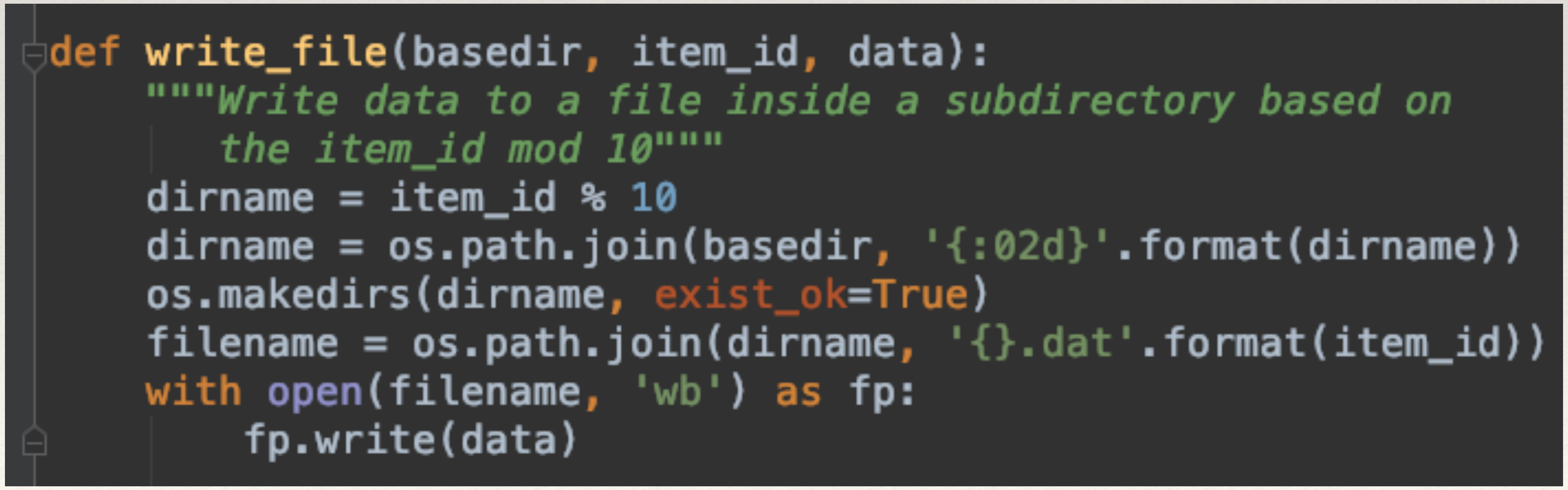
- or make sure everything is a byte and use explicit conversion



```
In [8]: with open("datafile", "wb") as fp:
   ...:     fp.write(mytext)
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-e92a5b0529f4> in <module>
      1 with open("datafile", "wb") as fp:
----> 2     fp.write(mytext)
      3

TypeError: a bytes-like object is required, not 'str'

In [9]: with open("datafile", "wb") as fp:
   ...:     fp.write(mytext.encode("utf-8"))
   ...:
```

storing data on disk

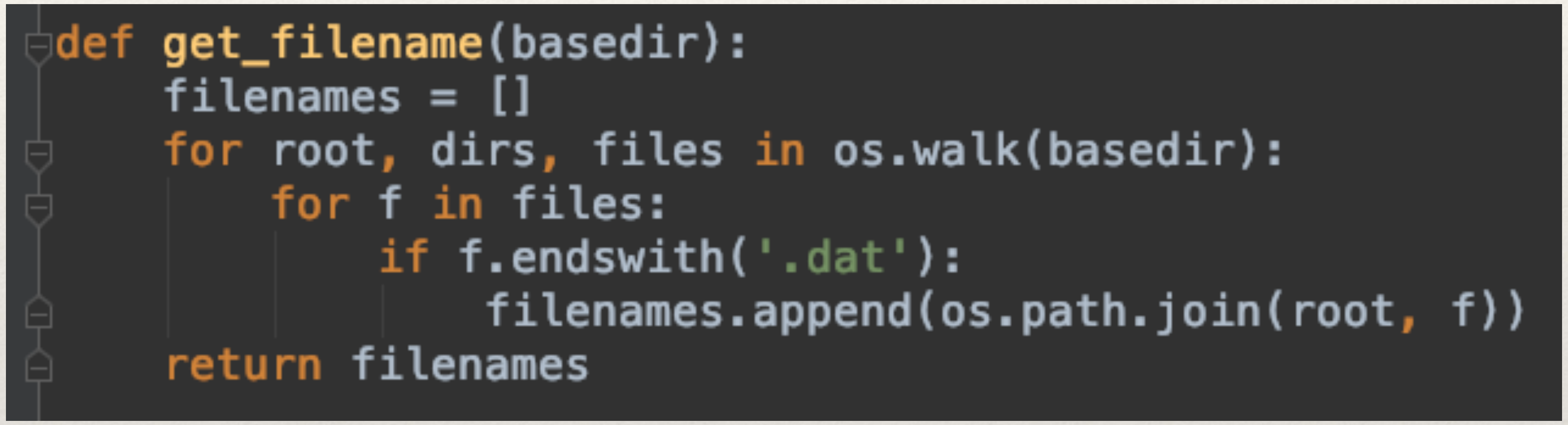
- Do you have more than 5,000 items? split them into multiple folders based on part of the filename or id
- Why? the way filesystems work can make it slow to get listings of files when there are thousands of files in a directory



```python
def write_file(basedir, item_id, data):
    """Write data to a file inside a subdirectory based on
       the item_id mod 10"""
    dirname = item_id % 10
    dirname = os.path.join(basedir, '{:02d}'.format(dirname))
    os.makedirs(dirname, exist_ok=True)
    filename = os.path.join(dirname, '{}.dat'.format(item_id))
    with open(filename, 'wb') as fp:
        fp.write(data)
```

reading files

- use `os.walk` to get a list of filenames in a directory



```python
def get_filename(basedir):
    filenames = []
    for root, dirs, files in os.walk(basedir):
        for f in files:
            if f.endswith('.dat'):
                filenames.append(os.path.join(root, f))
    return filenames
```

how many files should you have?

- Consider also how many files you want to write
  - Lots of small files (10s of thousands or more) means that you'll spend a lot of time just opening or writing files
  - This can impact runtime
- Consider bundling data into fewer larger files
- Don't read/write the same file over and over again

file formats

- What format should you use to store data?
- Python has many formats that it can read and write natively
  - pickle, json, csv
- Many other libraries that do similar things, depending on the type of data
  - numpy array, hdf5
- use modules for these libraries, don't write it yourself

pickle

- A representation of the memory structure of the object in python
  - Quite fast to read and write
  - Has some different versions - e.g. you can't open a pickle from python 3 in python 2
  - Don't share data to other people with it - a pickle can contain executable code and is therefore a potential security issue. Use a data format instead

CSV

- See our CSV tip: https://mtg.github.io/pymtg/tips/tips.html
- always use the csv module
- Don't do this:

```
data = open('myfile.csv').read()
lines = data.split('\n')
rows = lines[0].split(',')
```

- pandas will read csv files, but they must be "square"

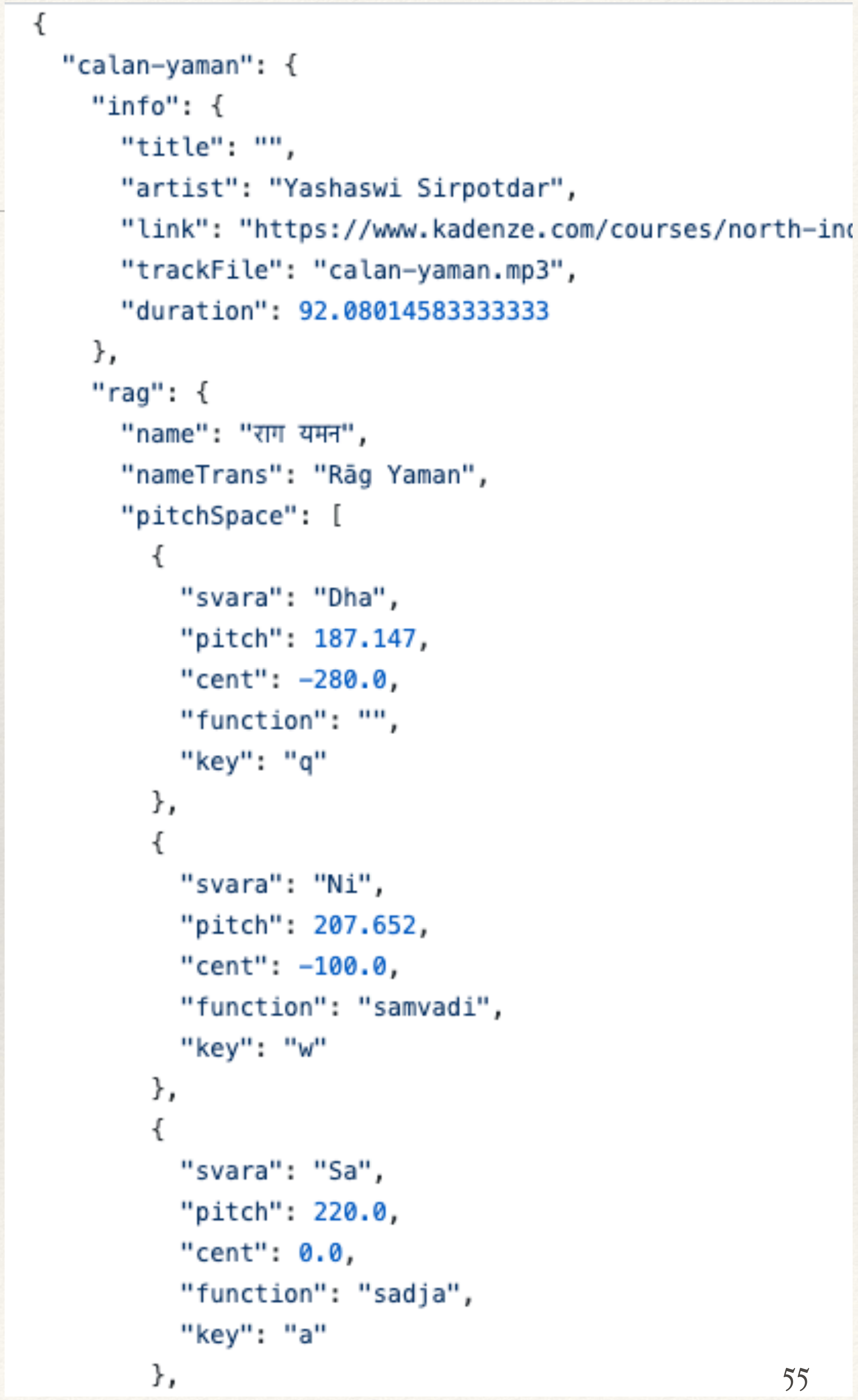
CSV

- Remember that you can read large files line by line to get just the data that you need

```
with open('myfile.csv') as fp:
    r = csv.reader(fp)
    for line in r:
        data.append(line[9])
```

json

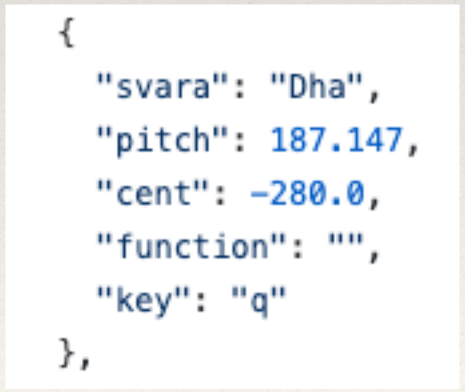
- json is easy to deal with, and reflects python dictionaries really well
- However, it's *really* inefficient
- You load all of the data into memory to read just a single field



```
{
  "calan-yaman": {
    "info": {
      "title": "",
      "artist": "Yashaswi Sirpotdar",
      "link": "https://www.kadenze.com/courses/north-ind
      "trackFile": "calan-yaman.mp3",
      "duration": 92.08014583333333
    },
    "rag": {
      "name": "राग यमन",
      "nameTrans": "Rāg Yaman",
      "pitchSpace": [
        {
          "svara": "Dha",
          "pitch": 187.147,
          "cent": -280.0,
          "function": "",
          "key": "q"
        },
        {
          "svara": "Ni",
          "pitch": 207.652,
          "cent": -100.0,
          "function": "samvadi",
          "key": "w"
        },
        {
          "svara": "Sa",
          "pitch": 220.0,
          "cent": 0.0,
          "function": "sadja",
          "key": "a"
        },
```

improving json

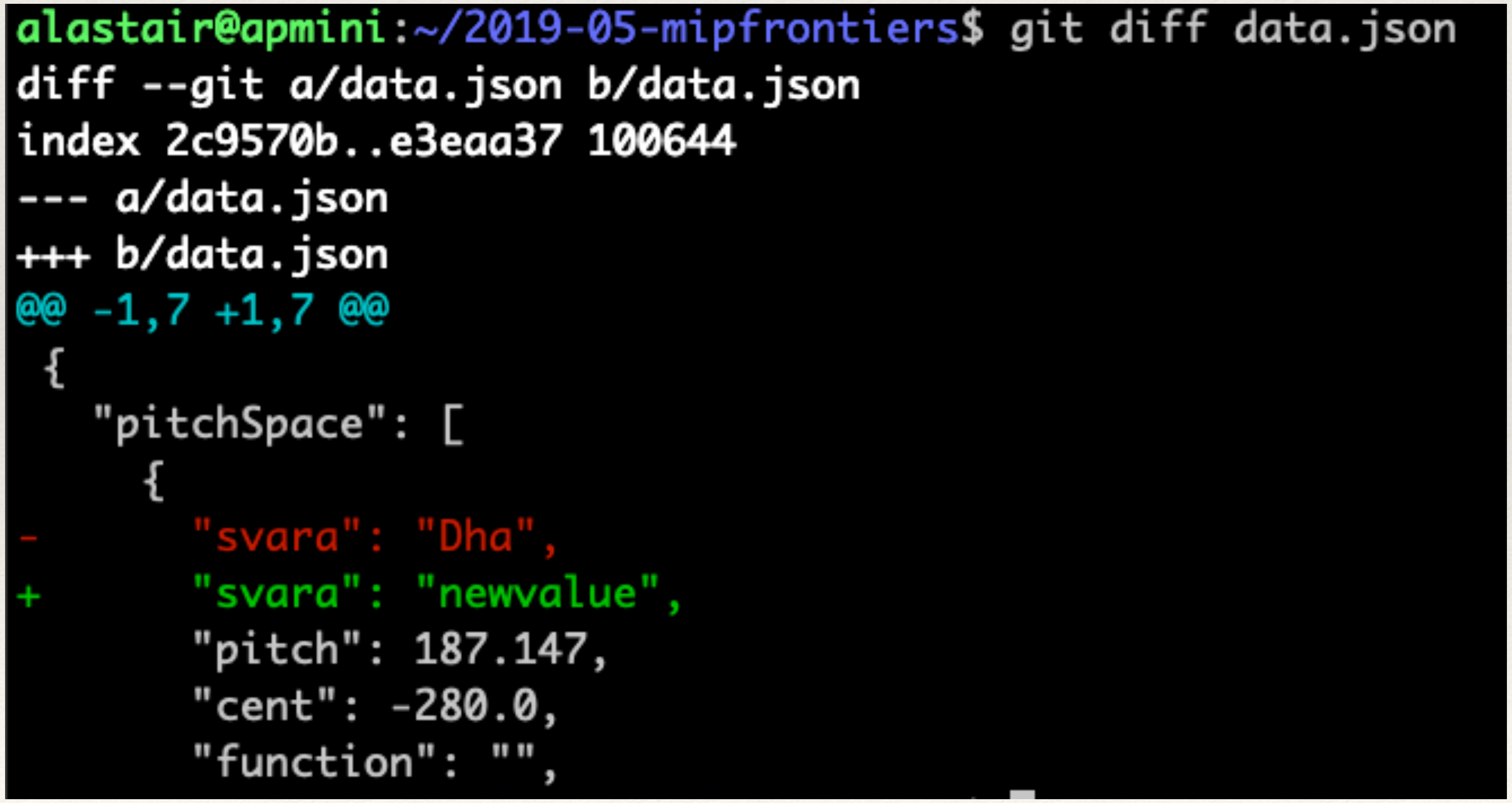
- Consider the size of repeated key names
  - if you have 1000 items with 5 keys each, length 10 that's almost half a megabyte of just keys!
- numbers are stored as text! pitch here is 7 bytes (56 bits)
  - do you really need decimal points in Hz? a 16-bit integer would store the same data
- spaces after key names are optional, as are newlines
  - save 2 bytes per line



```
{
  "svara": "Dha",
  "pitch": 187.147,
  "cent": -280.0,
  "function": "",
  "key": "q"
},
```

json caveats

-
  ```
  with open(filename, 'w') as fp:
      json.write(data, fp, indent=0, separators=(',', ':'))
  ```
- but! if you have newlines, git diffs look good. consider your data

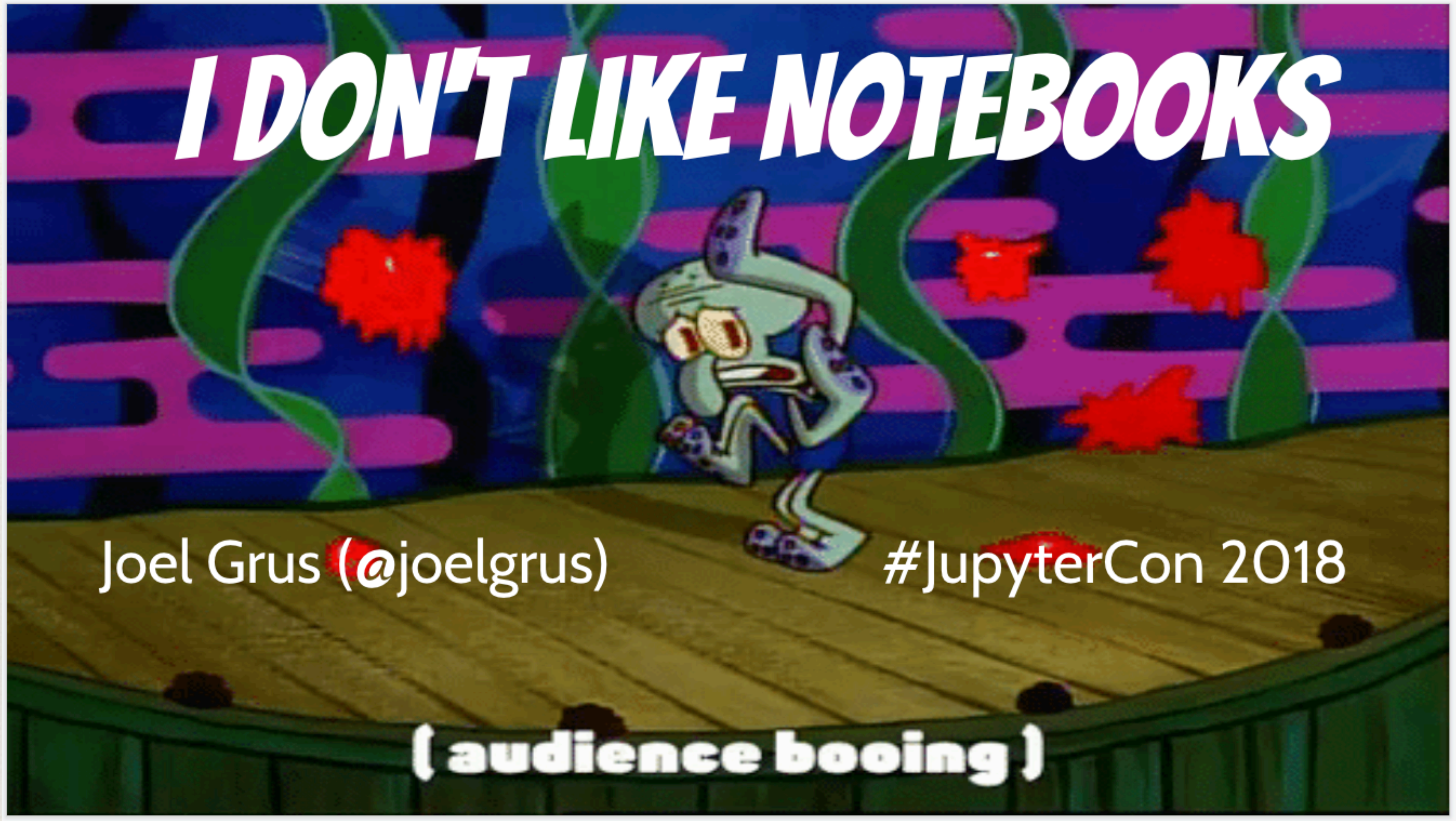


```
alastair@apmini:~/2019-05-mipfrontiers$ git diff data.json
diff --git a/data.json b/data.json
index 2c9570b..e3eaa37 100644
--- a/data.json
+++ b/data.json
@@ -1,7 +1,7 @@
 {
   "pitchSpace": [
     {
-      "svara": "Dha",
+      "svara": "newvalue",
       "pitch": 187.147,
       "cent": -280.0,
       "function": "",
```

other file limitations

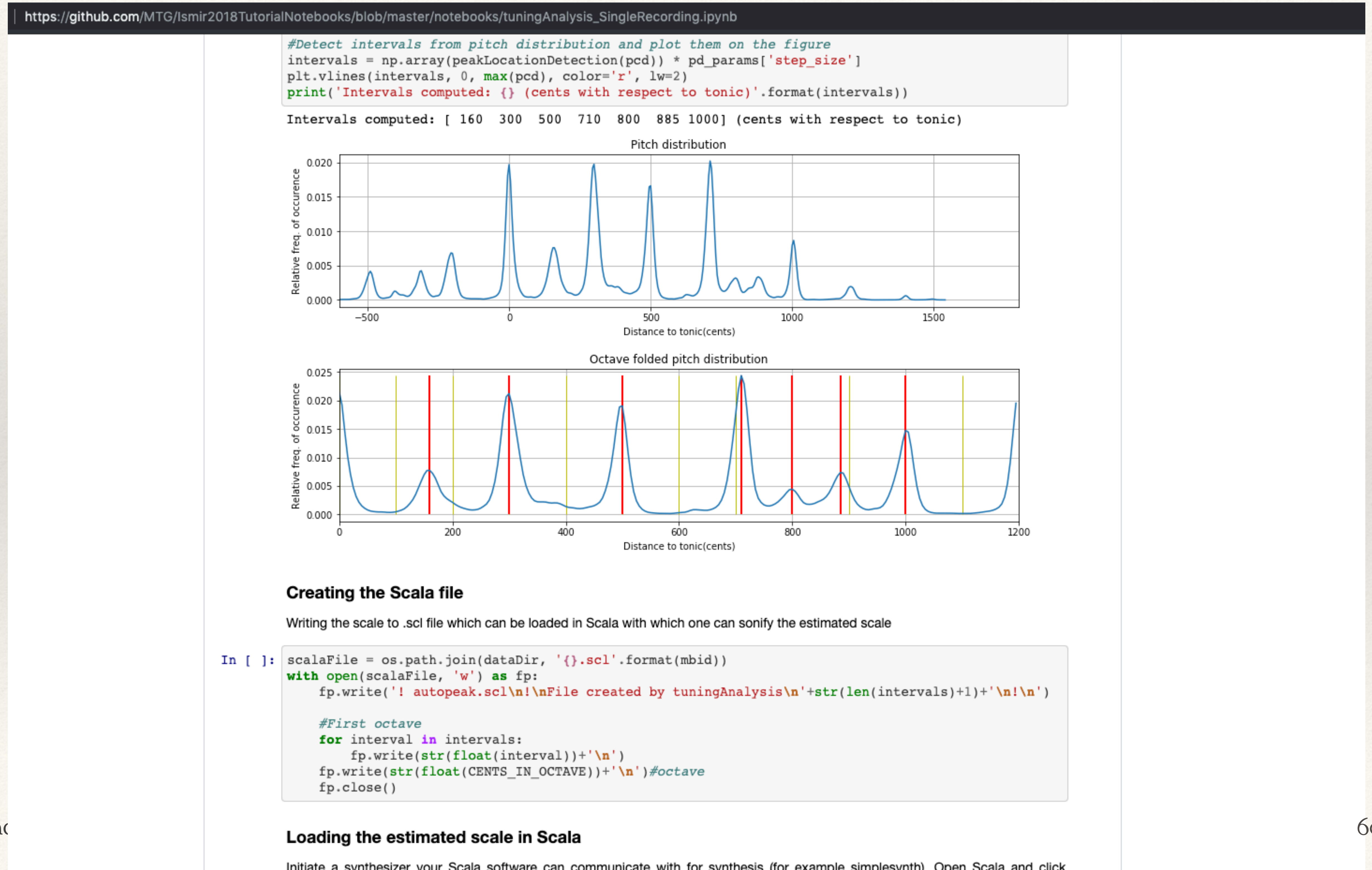
- Some filesize limits to keep in mind
  - FAT32 has a 4gb file limit
  - The original zip format has a 4gb archive size limit
- People might not always want to download a huge archive to get just a small amount of data
  - if you're splitting data up into folders, consider making archives of each folder - smaller to download

notebooks



- https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/edit#slide=id.g362da58057_0_1
- video of presentation https://www.youtube.com/watch?v=7jiPeIFXb6U

notebooks can be great…



https://github.com/MTG/Ismir2018TutorialNotebooks/blob/master/notebooks/tuningAnalysis_SingleRecording.ipynb
Pitch distribution
Relative freq. of occurence
Distance to tonic(cents)
Octave folded pitch distribution
Relative freq. of occurence
Distance to tonic(cents)

```
#Detect intervals from pitch distribution and plot them on the figure
intervals = np.array(peakLocationDetection(pcd)) * pd_params['step_size']
plt.vlines(intervals, 0, max(pcd), color='r', lw=2)
print('Intervals computed: {} (cents with respect to tonic)'.format(intervals))
```

```
Intervals computed: [ 160  300  500  710  800  885 1000] (cents with respect to tonic)
```

Creating the Scala file

Writing the scale to .scl file which can be loaded in Scala with which one can sonify the estimated scale

```
In [ ]: scalaFile = os.path.join(dataDir, '{}.scl'.format(mbid))
        with open(scalaFile, 'w') as fp:
            fp.write('! autopeak.scl\n!\nFile created by tuningAnalysis\n'+str(len(intervals)+1)+'\n!\n')

            #First octave
            for interval in intervals:
                fp.write(str(float(interval))+'\n')
            fp.write(str(float(CENTS_IN_OCTAVE))+'\n')#octave
            fp.close()
```

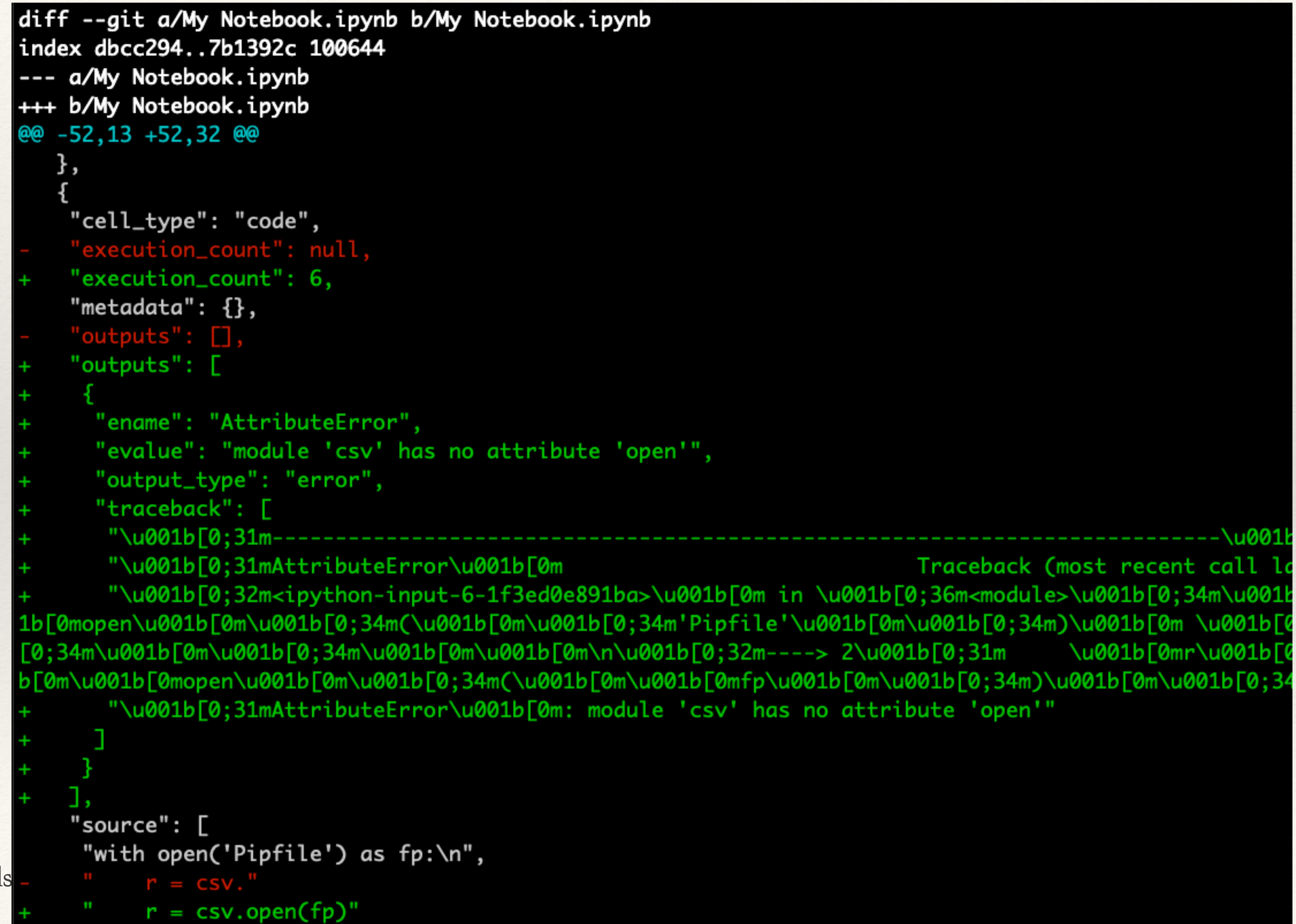
Loading the estimated scale in Scala

Initiate a synthesizer your Scala software can communicate with for synthesis (for example simplesynth). Open Scala and click

but…

- You can execute cells out of order, encouraging you to not think about your program flow
- The editor doesn't have a lot of great functionality that other tools give you (code formatting, autocomplete, code checking)
- It encourages you to not think about the structure and reproducibility of your code
- difficult to test
- difficult to copy/paste examples

how do you use git?



```
diff --git a/My Notebook.ipynb b/My Notebook.ipynb
index dbcc294..7b1392c 100644
--- a/My Notebook.ipynb
+++ b/My Notebook.ipynb
@@ -52,13 +52,32 @@
   },
   {
    "cell_type": "code",
-   "execution_count": null,
+   "execution_count": 6,
    "metadata": {},
-   "outputs": [],
+   "outputs": [
+    {
+     "ename": "AttributeError",
+     "evalue": "module 'csv' has no attribute 'open'",
+     "output_type": "error",
+     "traceback": [
+      "\u001b[0;31m---------------------------------------------------------------------------\u001b
+      "\u001b[0;31mAttributeError\u001b[0m                            Traceback (most recent call la
+      "\u001b[0;32m<ipython-input-6-1f3ed0e891ba>\u001b[0m in \u001b[0;36m<module>\u001b[0;34m\u001b
1b[0mopen\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m'Pipfile'\u001b[0m\u001b[0;34m)\u001b[0m \u001b[0
[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0;32m----> 2\u001b[0;31m     \u001b[0mr\u001b[0
b[0m\u001b[0mopen\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mfp\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34
+      "\u001b[0;31mAttributeError\u001b[0m: module 'csv' has no attribute 'open'"
+     ]
+    }
+   ],
    "source": [
     "with open('Pipfile') as fp:\n",
-    "    r = csv."
+    "    r = csv.open(fp)"
```

notebook recommendations

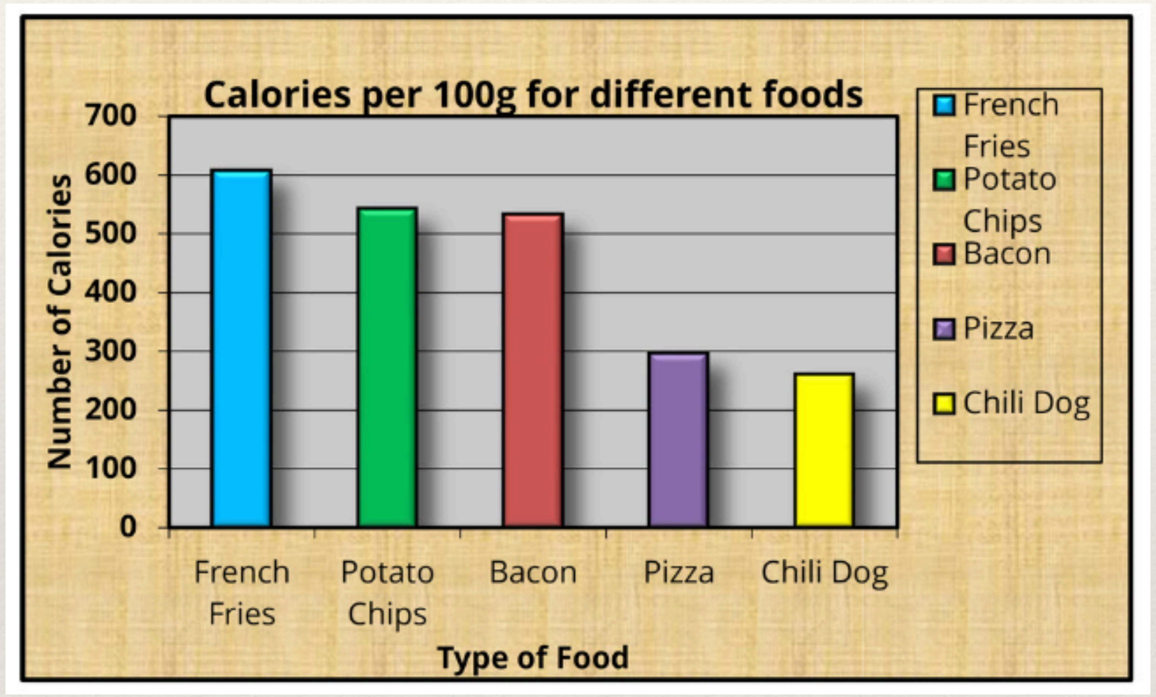
- Go and watch the presentation
- This doesn't mean that you shouldn't use notebooks. Know their strengths and weaknesses
- A good compromise is to write your code in a file and then use a notebook to show it off and display visualisations
    - your code benefits from source control, tests
    - you still get pretty inline pictures, and you can tell a story with your code

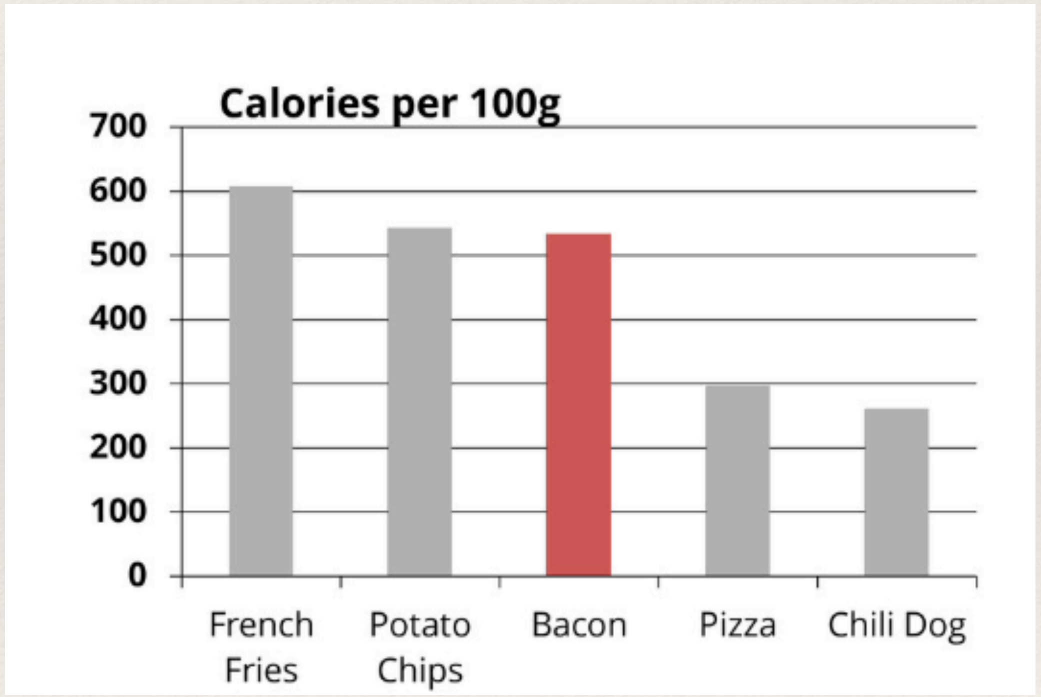
visualisation

- presentation of your data is important to get your point across
- be careful with matplotlib default settings, they often don't look great
  - If you have time, play around with the colour pallets and themes (https://matplotlib.org/users/dflt_style_changes.html)
  - Take a look at seaborn for graphs (https://seaborn.pydata.org/)
  - Think about the layout of your data

data-to-ink ratio

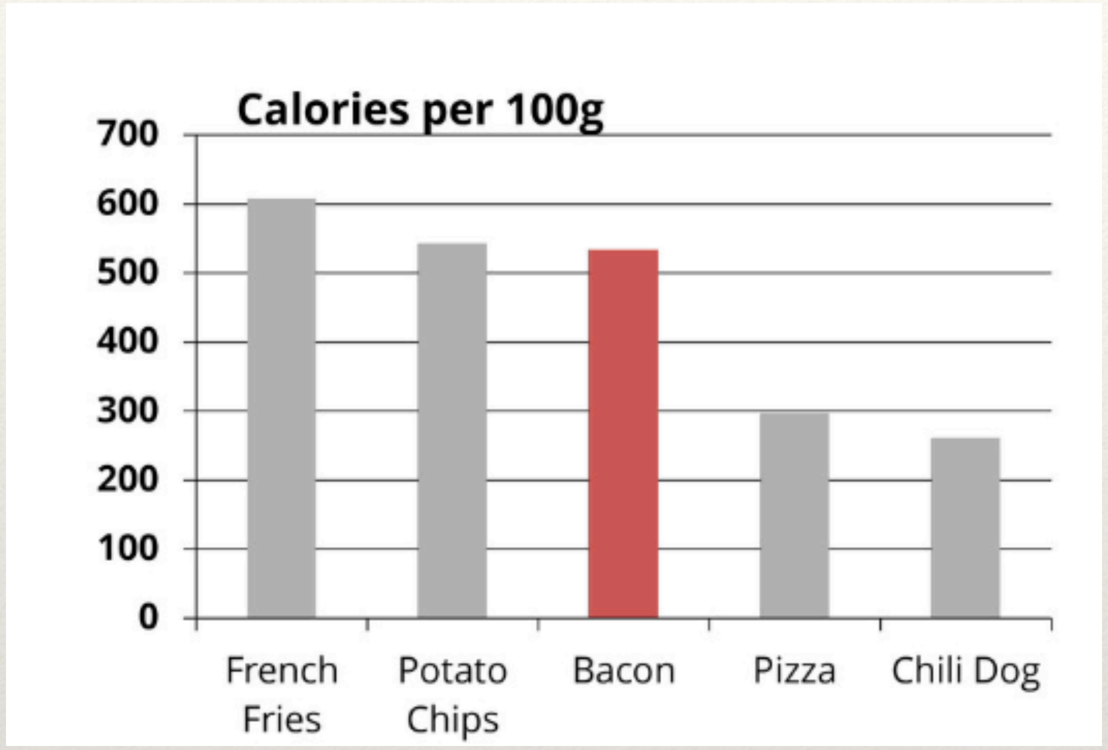
- Take some time to think about the data that you're presenting

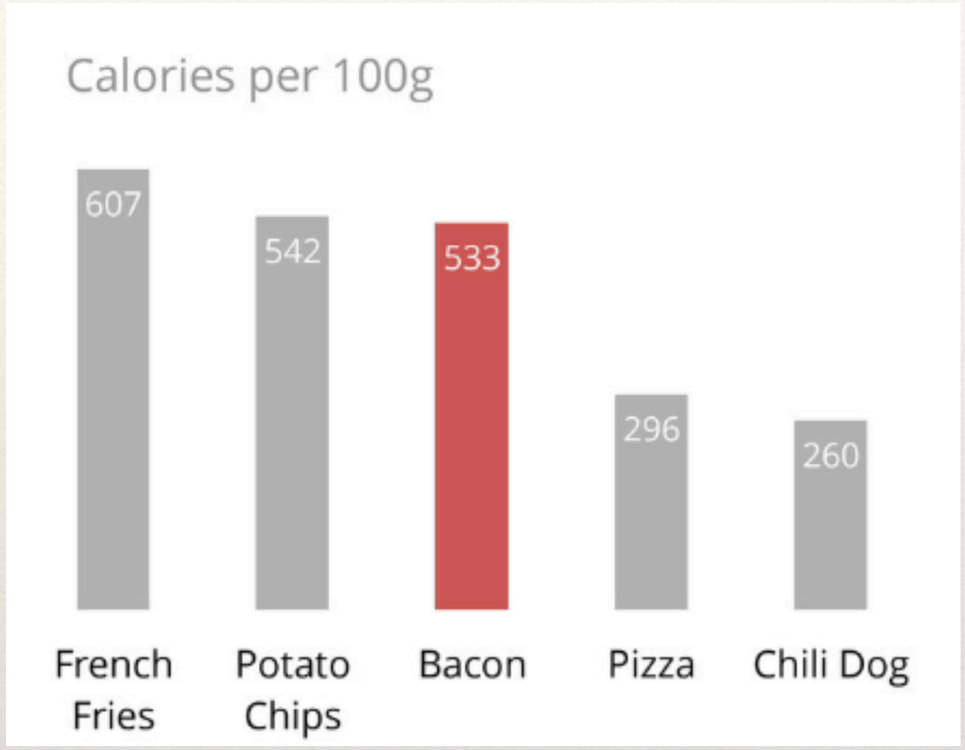




- https://www.darkhorseanalytics.com/blog/data-looks-better-naked

can you remove more?

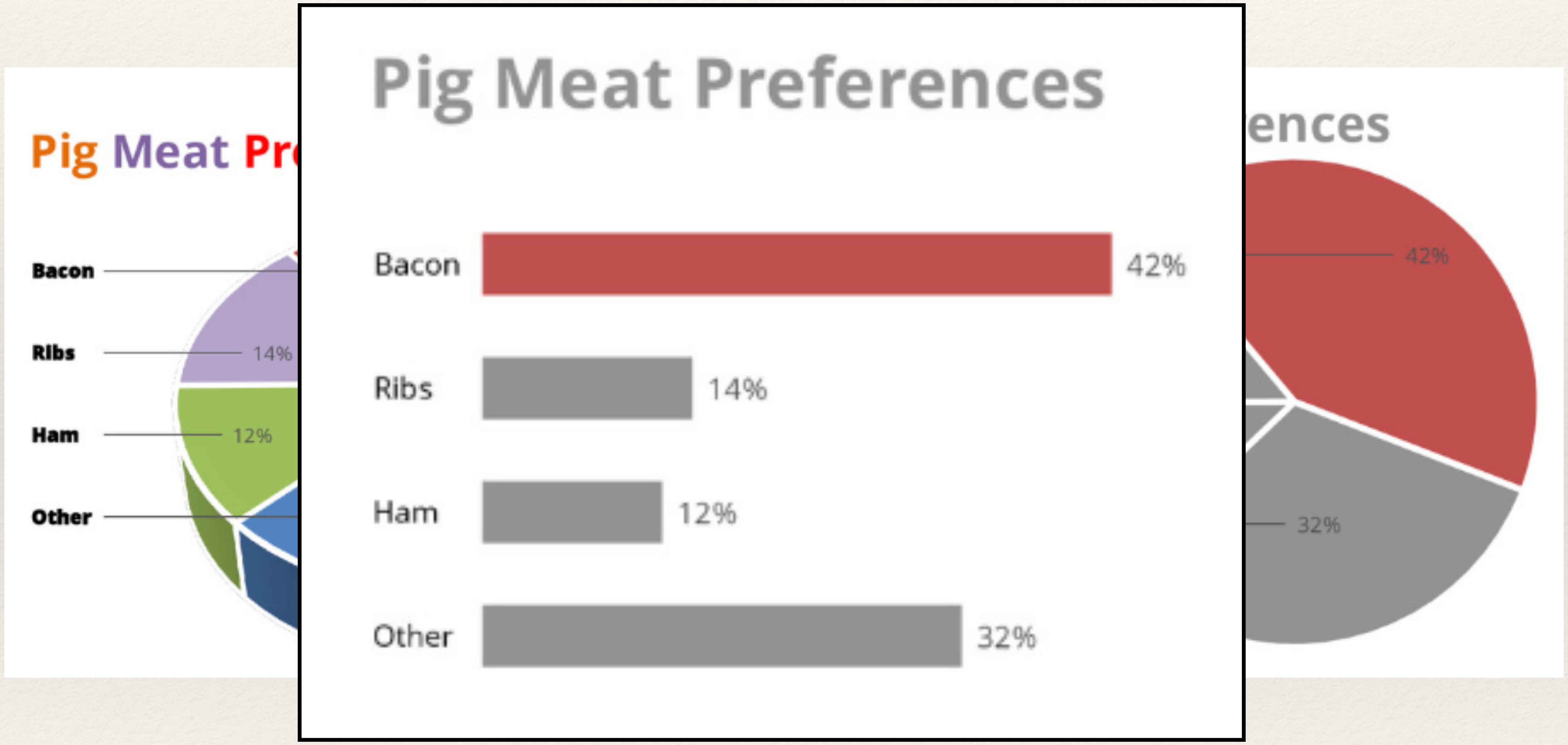




- see also https://gorelik.net/2018/03/21/three-most-common-mistakes-in-data-visualization-%E2%80%A8and-how-to-avoid-them-now-the-slides/

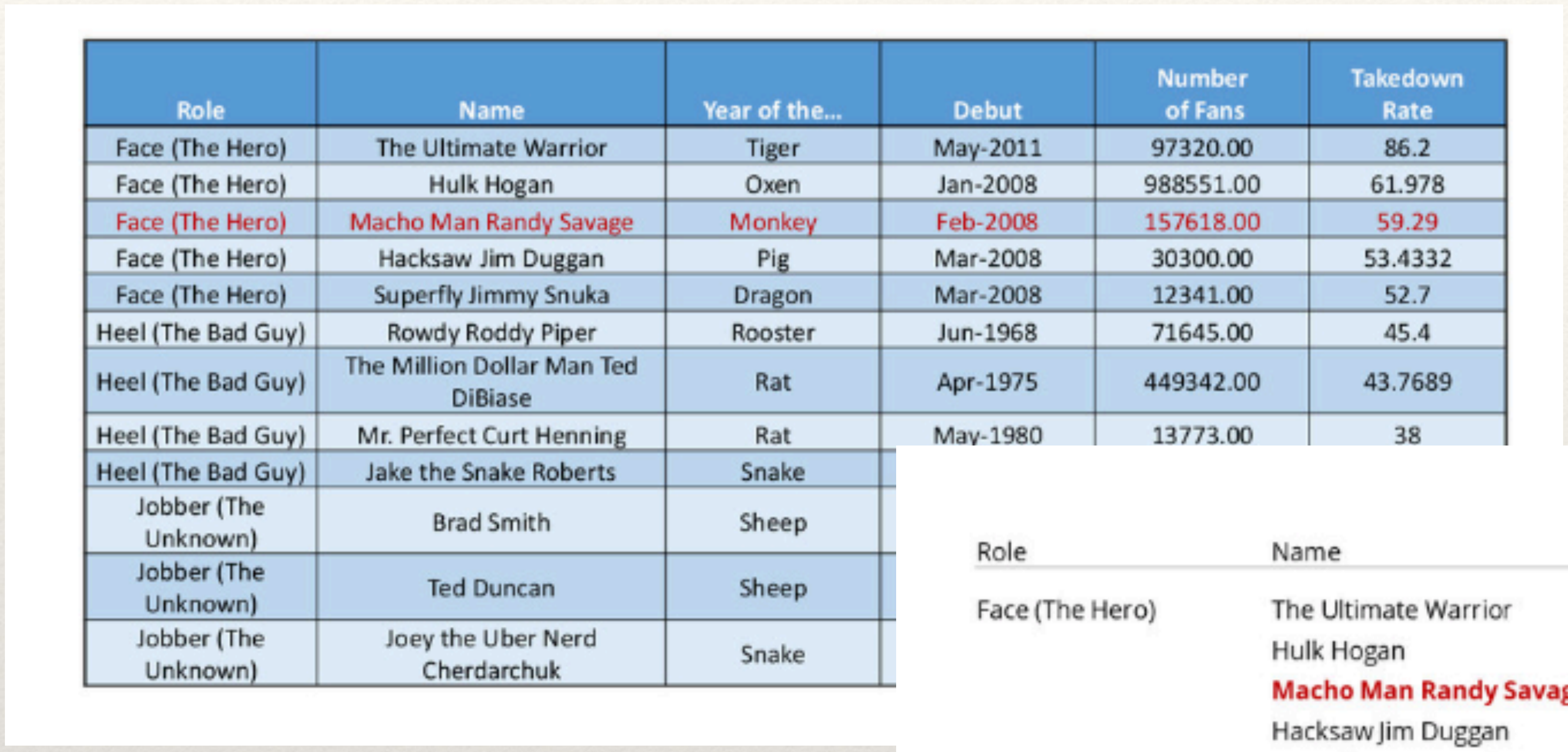
pie charts

- Pie charts make it difficult for people to intuit ratios

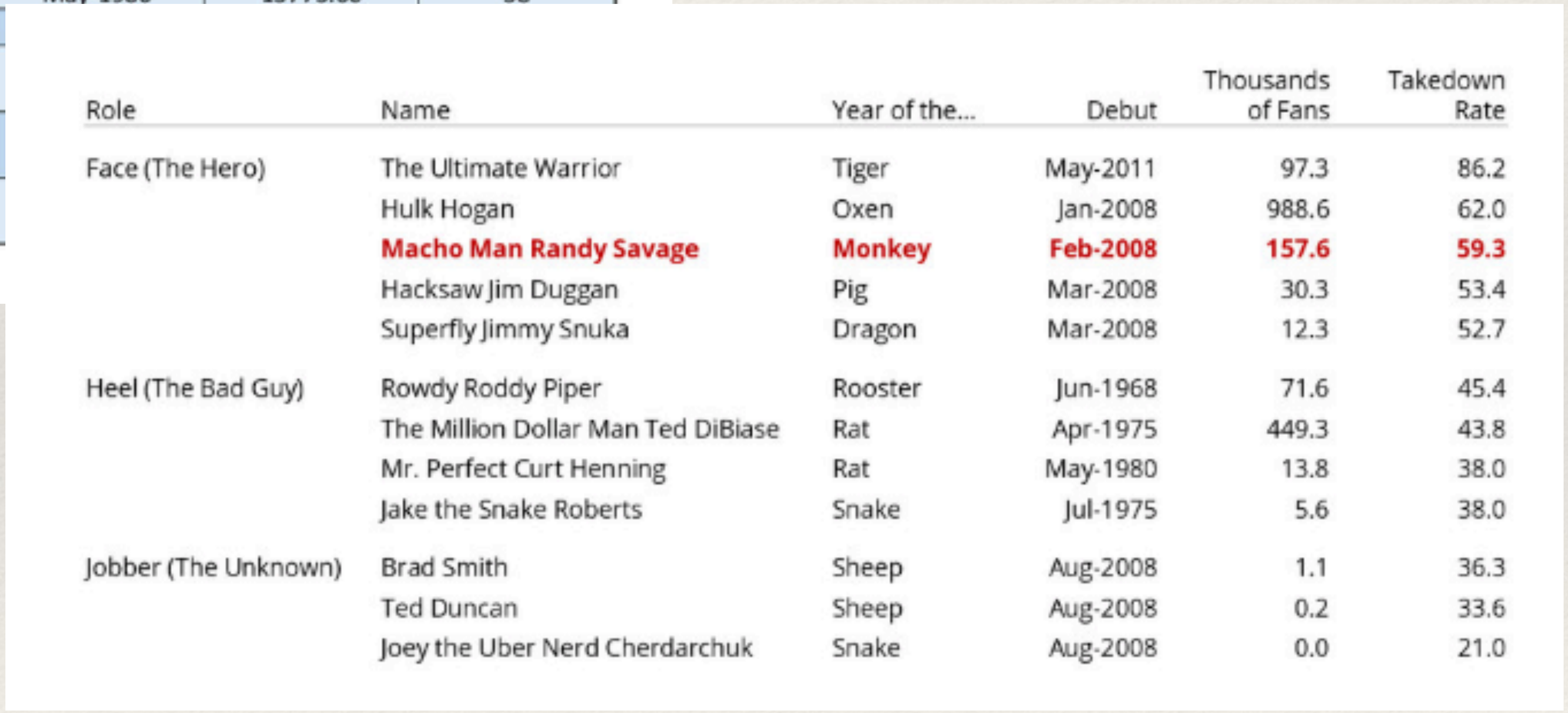


- https://www.darkhorseanalytics.com/blog/salvaging-the-pie

tables



Role	Name	Year of the...	Debut	Number of Fans	Takedown Rate
Face (The Hero)	The Ultimate Warrior	Tiger	May-2011	97320.00	86.2
Face (The Hero)	Hulk Hogan	Oxen	Jan-2008	988551.00	61.978
Face (The Hero)	Macho Man Randy Savage	Monkey	Feb-2008	157618.00	59.29
Face (The Hero)	Hacksaw Jim Duggan	Pig	Mar-2008	30300.00	53.4332
Face (The Hero)	Superfly Jimmy Snuka	Dragon	Mar-2008	12341.00	52.7
Heel (The Bad Guy)	Rowdy Roddy Piper	Rooster	Jun-1968	71645.00	45.4
Heel (The Bad Guy)	The Million Dollar Man Ted DiBiase	Rat	Apr-1975	449342.00	43.7689
Heel (The Bad Guy)	Mr. Perfect Curt Henning	Rat	May-1980	13773.00	38
Heel (The Bad Guy)	Jake the Snake Roberts	Snake			
Jobber (The Unknown)	Brad Smith	Sheep			
Jobber (The Unknown)	Ted Duncan	Sheep			
Jobber (The Unknown)	Joey the Uber Nerd Cherdarchuk	Snake			

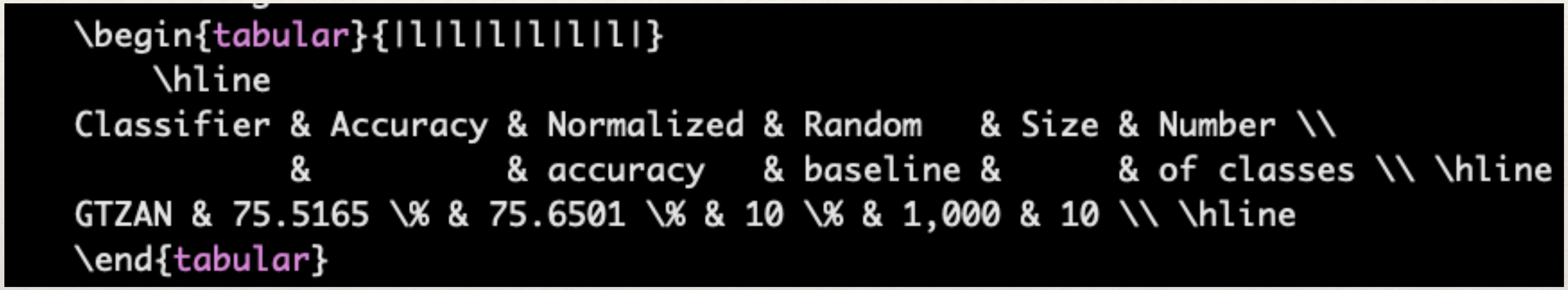


Role	Name	Year of the...	Debut	Thousands of Fans	Takedown Rate
Face (The Hero)	The Ultimate Warrior	Tiger	May-2011	97.3	86.2
	Hulk Hogan	Oxen	Jan-2008	988.6	62.0
	Macho Man Randy Savage	**Monkey**	**Feb-2008**	**157.6**	**59.3**
	Hacksaw Jim Duggan	Pig	Mar-2008	30.3	53.4
	Superfly Jimmy Snuka	Dragon	Mar-2008	12.3	52.7
Heel (The Bad Guy)	Rowdy Roddy Piper	Rooster	Jun-1968	71.6	45.4
	The Million Dollar Man Ted DiBiase	Rat	Apr-1975	449.3	43.8
	Mr. Perfect Curt Henning	Rat	May-1980	13.8	38.0
	Jake the Snake Roberts	Snake	Jul-1975	5.6	38.0
Jobber (The Unknown)	Brad Smith	Sheep	Aug-2008	1.1	36.3
	Ted Duncan	Sheep	Aug-2008	0.2	33.6
	Joey the Uber Nerd Cherdarchuk	Snake	Aug-2008	0.0	21.0

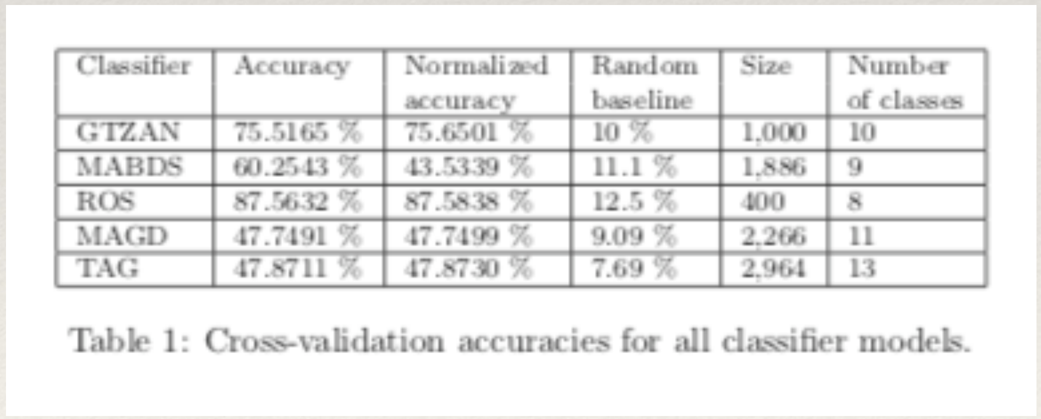
- https://www.darkhorseanalytics.com/blog/clear-off-the-table

tables in latex

- like visualisations, consider what data you can remove



```
\begin{tabular}{|l|l|l|l|l|l|}
    \hline
Classifier & Accuracy & Normalized & Random   & Size & Number \\
           &          & accuracy   & baseline &      & of classes \\ \hline
GTZAN & 75.5165 \% & 75.6501 \% & 10 \% & 1,000 & 10 \\ \hline
\end{tabular}
```

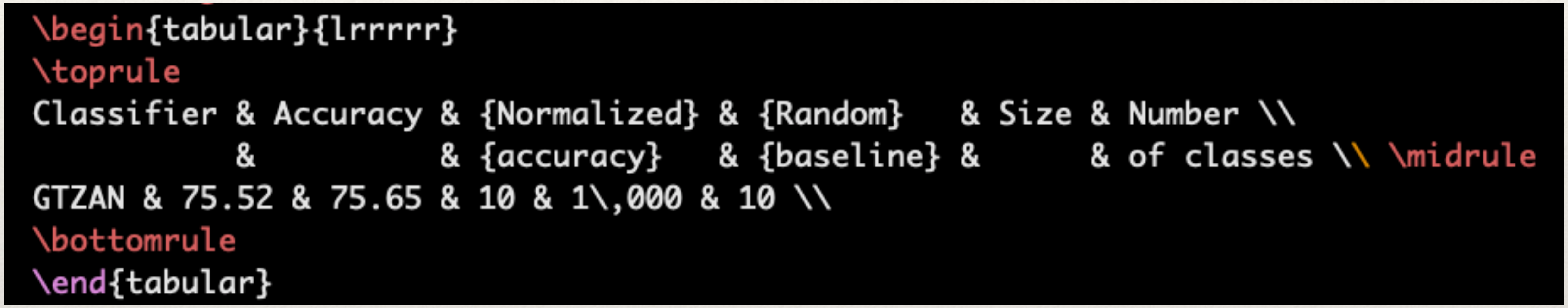


Classifier	Accuracy	Normalized accuracy	Random baseline	Size	Number of classes
GTZAN	75.5165 %	75.6501 %	10 %	1,000	10
MABDS	60.2543 %	43.5339 %	11.1 %	1,886	9
ROS	87.5632 %	87.5838 %	12.5 %	400	8
MAGD	47.7491 %	47.7499 %	9.09 %	2,266	11
TAG	47.8711 %	47.8730 %	7.69 %	2,964	13

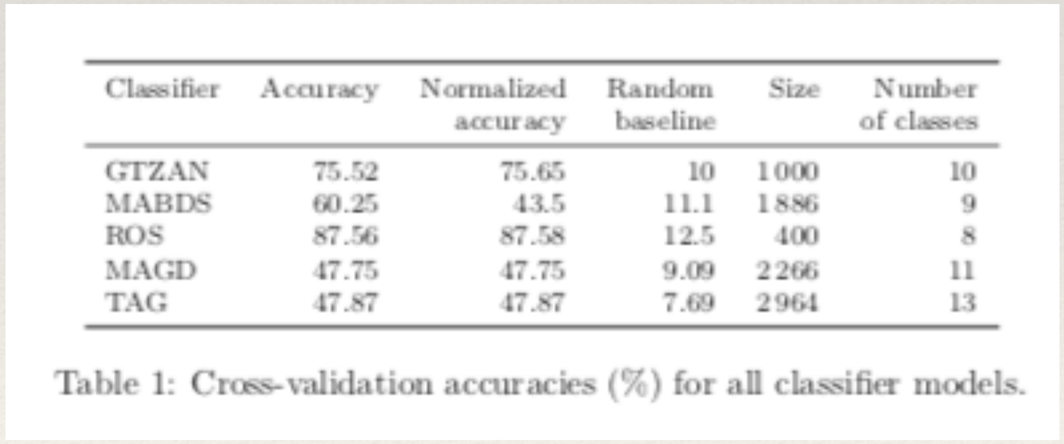
Table 1: Cross-validation accuracies for all classifier models.

tables in latex

- use `booktabs` for nice looking tables



```latex
\begin{tabular}{lrrrrr}
\toprule
Classifier & Accuracy & {Normalized} & {Random}   & Size & Number \\
           &          & {accuracy}   & {baseline} &      & of classes \\ \midrule
GTZAN & 75.52 & 75.65 & 10 & 1\,000 & 10 \\
\bottomrule
\end{tabular}
```



Classifier	Accuracy	Normalized accuracy	Random baseline	Size	Number of classes
GTZAN	75.52	75.65	10	1 000	10
MABDS	60.25	43.5	11.1	1 886	9
ROS	87.56	87.58	12.5	400	8
MAGD	47.75	47.75	9.09	2 266	11
TAG	47.87	47.87	7.69	2 964	13

Table 1: Cross-validation accuracies (%) for all classifier models.

a bunch of python tips

mtg python tips

- we publish "tip of the week" (every few weeks/once a month)
- See them here: https://mtg.github.io/pymtg/tips/tips.html

speed of access in datastructures

- As a list gets longer, checking for set membership takes longer

```
>>> mylist = [1, 2, 3, 4, ...., 1million, ...., 2million]
>>> 50 in mylist
False
```

- Use a set when you want to check

```
>>> myset = set(mylist)
>>> 50 in myset    # <- super fast
```

- https://docs.python.org/3.7/library/stdtypes.html#set-types-set-frozenset

looping

- dictionaries
- don't do this
  ```
  for k in mydict.keys():
      v = mydict[k]
  ```
- do this
  ```
  for k, v in mydict.items():
      ...
  ```
- lists

-
  ```
  i = 0
  for item in mylist:
      print(i, item)
      i += 1
  ```

-
  ```
  for i, item in
  enumerate(mylist):
      print(i, item)
  ```

dictionary tricks

```
data = {}
for r in response:
    if r not in data:
        data[r] = 1
    else:
        data[r] += 1
```

```
import collections
data = collections.defaultdict(int)
for r in response:
    data[r] += 1
```

```
data = collections.Counter(response)
data.most_common()
```

files

- Don't concatenate directories and filenames

  -
    ```
    mydir = "output/"
    filename = mydir + "filename.json"
    ```
  - `filename = os.path.join(mydir, "filename.json")`
  - Works even if `mydir` doesn't end in a `/`
- Making a directory tree
  - `os.makedirs(os.path.join("full", "directory", "path"), exist_ok=True)`
  - `exist_ok` means that it won't fail if the directory already exists (`os.mkdir` will fail)

the internet

- the requests package makes loading data from the internet easy, use it!
- Remember to consider your failure cases
    - site down, url doesn't exist, data isn't the format that you expect
- https://www.peterbe.com/plog/best-practice-with-retries-with-requests

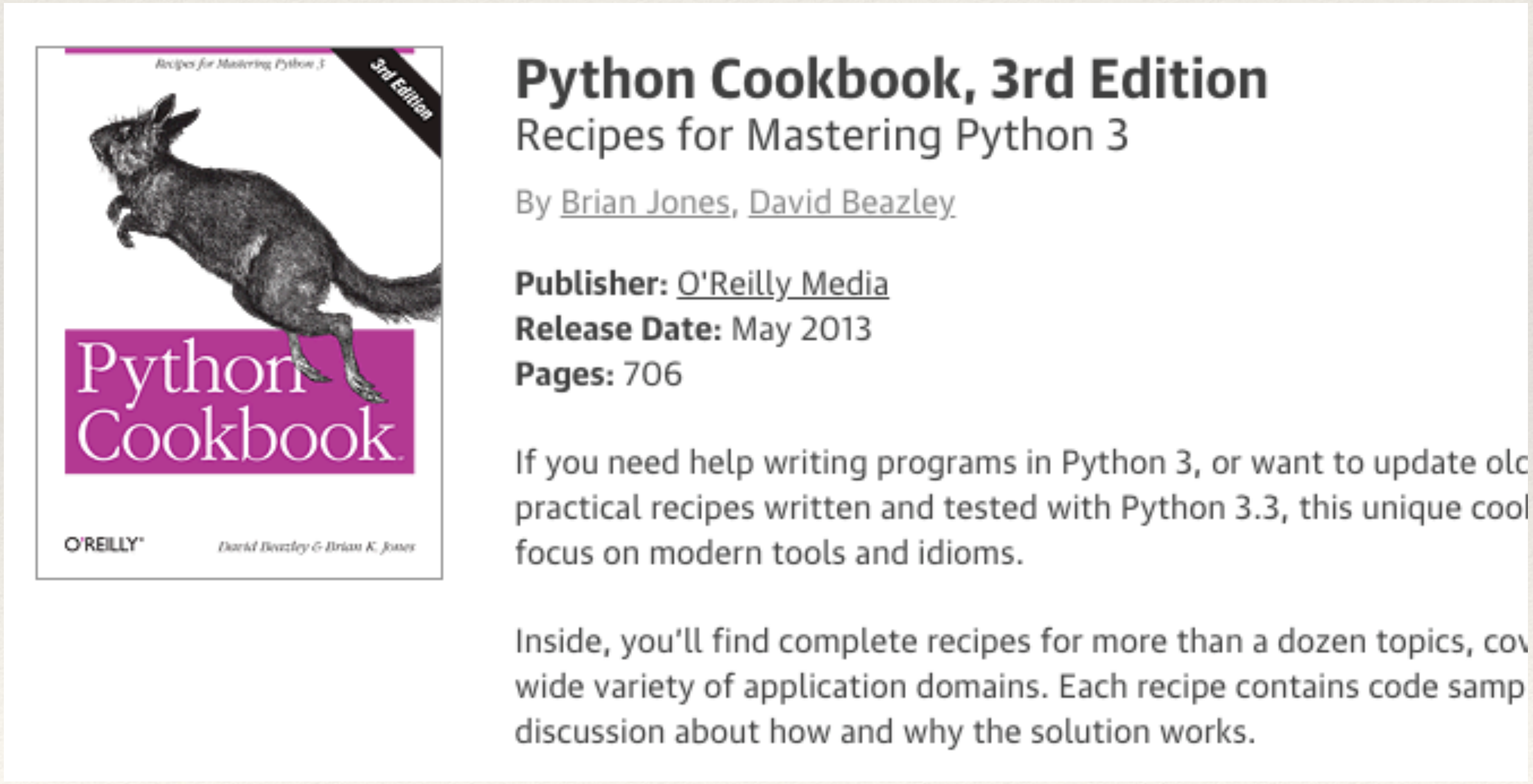
requests

-
  ```
  import requests
  response = requests.get('https://api.ipify.org?format=json').json()
  ```

-
  ```
  from requests.adapters import HTTPAdapter
  from requests.packages.urllib3.util.retry import Retry
  ret = Retry(total=10, backoff_factor=0.2)
  adaptor = HTTPAdapter(max_retries=ret)
  session = requests.Session()
  session.mount('https://', adaptor)

  result = session.get('https://api.ipify.org?format=json')
  try:
      data = result.raise_for_status()
      parsed = data.json()
  except HTTPError:
      # there was an error
  except ValueError:
      # it wasn't json
  ```

more tips

- https://realpython.com/
- Python 3 cookbook: http://shop.oreilly.com/product/0636920027072.do


Recipes for Mastering Python 3
3rd Edition
Python Cookbook
O'REILLY
David Beazley & Brian K. Jones

Python Cookbook, 3rd Edition
Recipes for Mastering Python 3

By Brian Jones, David Beazley

Publisher: O'Reilly Media
Release Date: May 2013
Pages: 706

If you need help writing programs in Python 3, or want to update olc
practical recipes written and tested with Python 3.3, this unique cool
focus on modern tools and idioms.

Inside, you'll find complete recipes for more than a dozen topics, cov
wide variety of application domains. Each recipe contains code samp
discussion about how and why the solution works.