Python
(and why you might care)

why its good for Science/Engineering

MICHIGAN STATE
UNIVERSITY

ICER

First Question

If someone comes and tells you about the newest, latest programming language, the first question you should ask is

WHY!

More Algorithm, less Language

.

First language should be:

- General
- Practical
- Is "straightforward" or at least doesn't get in your way

Purposeful Effort

Common questions:

- I'm an scientist, not a programmer. I just want to get some work done.
- You computer guys are crazy for computer languages.
- I don't have time to learn programming. I need to do science (engineering/whatever)
- "Give me what I need to do the job and go away"

Why Python?

PRACTICAL!

Python allows Purposeful Effort

Languages come and go, but if you put your time into learning one you want to know that you will reap some benefits: practical!

Our experience is that Python:

- gets in the way less
- is more practically useful to students
- allows students to apply it readily outside of class

Program because you **can**

What we say we want a student to do when they have a problem to solve is:

"Hey, I'll just write a program to do that"

… because they can, because it is natural, because it makes sense timewise, to do so.

The take away

- relatively simple syntax (readable!)
- "one way to do it"
  - choices are bad
- "batteries included"
  - common needs provided in the language
- "plays well with others"
  - hooks in with other languages/paradigms
- strong open-source community
  - http://pypi.python.org/pypi
- Free (all of it, all packages)

Science Goal: simple, then fast/better

- Develop code simply at first, fast later
  - *"Premature optimization is the root of all evil"*

    Donald Knuth, famous computer guy
  - you only have to make the slow parts fast (duh).
- Readability is important! You are likely going to have to read this %@#$ again.

Do as much the "easy way" as possible, then change up and do the "hard things" as you can/must.

Python for science

- Python allows you to do many of the things you need to do simply (file access, string manipulation, graphing, GUI).
- With what is left, you do what you can to solve the problem (faster, work with other stuff)

FOCUS YOUR EFFORTS.

The End

- Python is a good way to examine ideas with a straightforward programming approach
- Python has many built in tools (or avail tools) that make many tasks easier
- Python can be integrated with other stuff to work better/faster

PYTHON IS A GOOD FOUNDATION FOR ENGINEERING/SCIENTIFIC PROGRAMMING!

Slashdot, 10/17/13
Topic: lang for Sci Comp

http://science.slashdot.org/story/13/10/17/1433208/ask-slashdot-best-language-to-learn-for-scientific-computing

Since you mention VBA, I suspect that your data is in Excel spreadsheets? If you want to try to speed this up with minimum effort, then consider using Python with Pyvot [codeplex.com] to access the data, and then numpy [numpy.org]/scipy [scipy.org]/pandas [pydata.org] to do whatever processing you need. This should give you a significant perf boost without the need to significantly rearchitecture everything or change your workflow much.

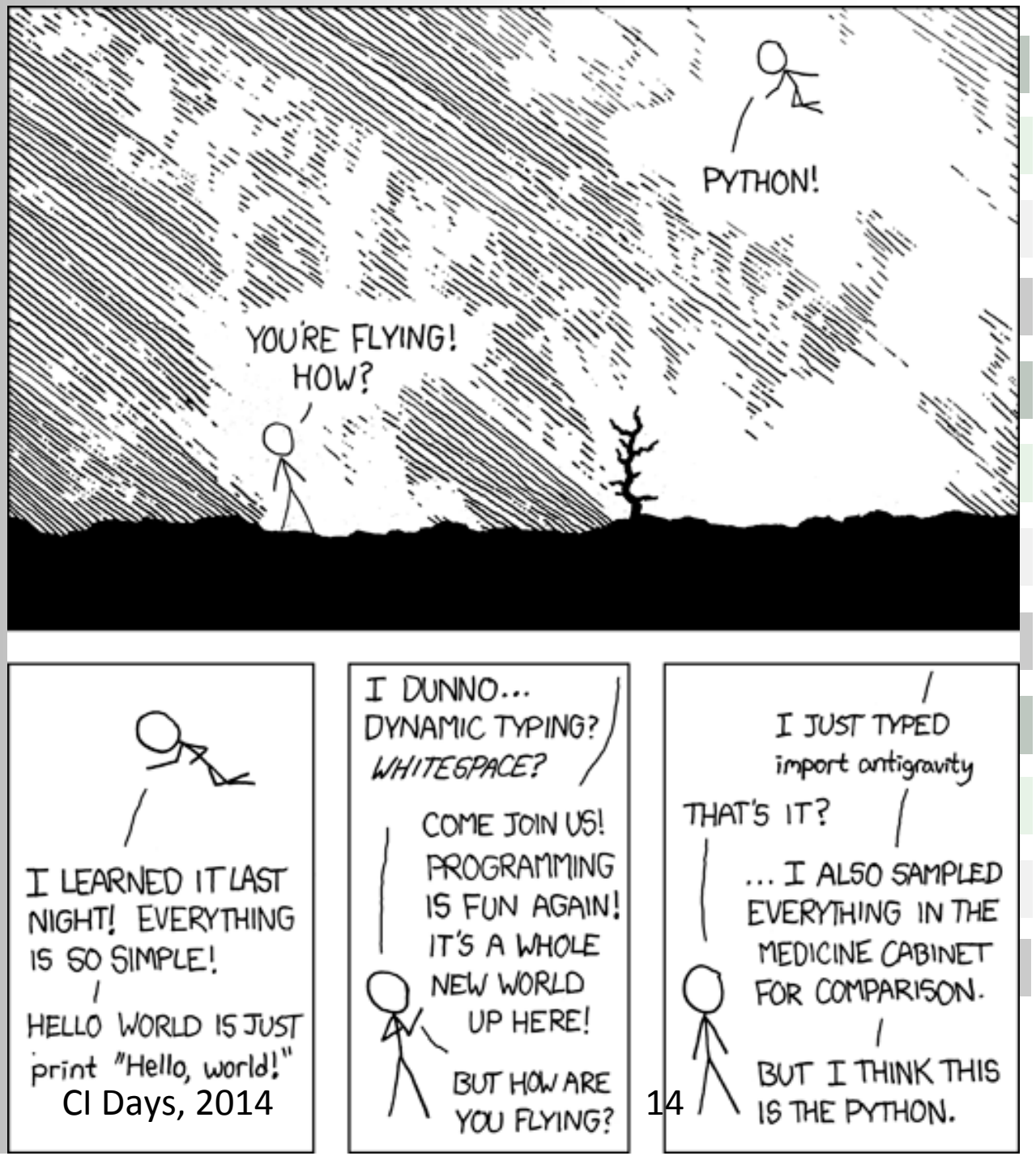
In addition, using Python this way gives you the ability to use IPython [ipython.org] to work with your data in interactive mode - it's kinda like a scientific Python REPL, with graphing etc.

pypi, packages for days!

https://pypi.python.org/pypi

50196!

requisite xkcd comic

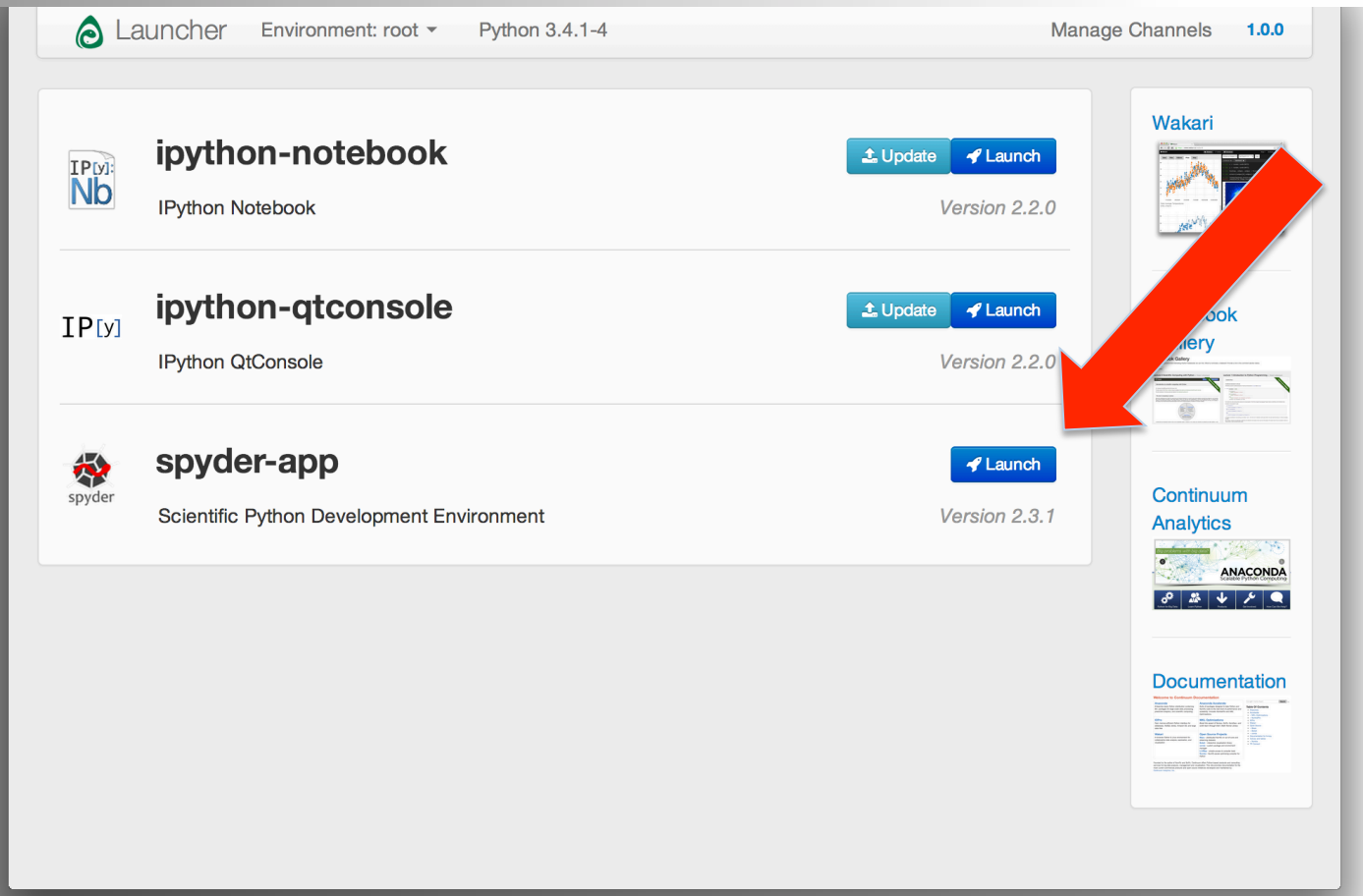


Let's get started

MICHIGAN STATE
UNIVERSITY

ICER

Launcher


Launcher
Environment: root
Python 3.4.1-4
Manage Channels
1.0.0
ipython-notebook
Update
Launch
IPython Notebook
Version 2.2.0
ipython-qtconsole
Update
Launch
IPython QtConsole
Version 2.2.0
spyder-app
Launch
Scientific Python Development Environment
Version 2.3.1
Wakari
Continuum Analytics
Documentation

spyder

Editor

Program Info

Spyder

/Users/bill

Editor - /Users/bill/.spyder2/.temp.py

.temp.py

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This temporary script file is located here:
/Users/bill/.spyder2/.temp.py
"""
```

Object inspector

Source Console Object Options

No documentation available

Object inspector | Variable explorer | File explorer

Console

Python 1 00:00:30

```
Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.7.1, SciPy 0.12.0, Matplotlib 1.2.1
Type "scientific" for more details.
>>> 1 + 2
3
>>> x = 12
>>>
```

iPython Console

Console | History log

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 1 Column: 1 Memory: 44 %

interpreted, we can play

bottom right is the interpreter:

- you can type in something and see what happens
- you should do this a lot!
  - if you are asking yourself "What will happen if", then the answer should be "try it"

hello world

A common first program

- in Python, type
  - print("hello world")

Hit the enter key.

Congratulations!!!

Do some math

2 + 2

4

3 * 3

9

fun!

Write a program

lots going on here

First Program

```
# input two numbers, add them, print them out

num_str1 = input("Please enter an integer:")
num_str2 = input("Please enter a decimal number:")

str1_int = int(num_str1)
str2_float = float(num_str2)  # this is a  comment

print("The numbers are: ",str1_int," and ",str2_float)
print("Their sum is:",str1_int + str2_float,\
       "and their product is:",str1_int* str2_float)
```

Getting input

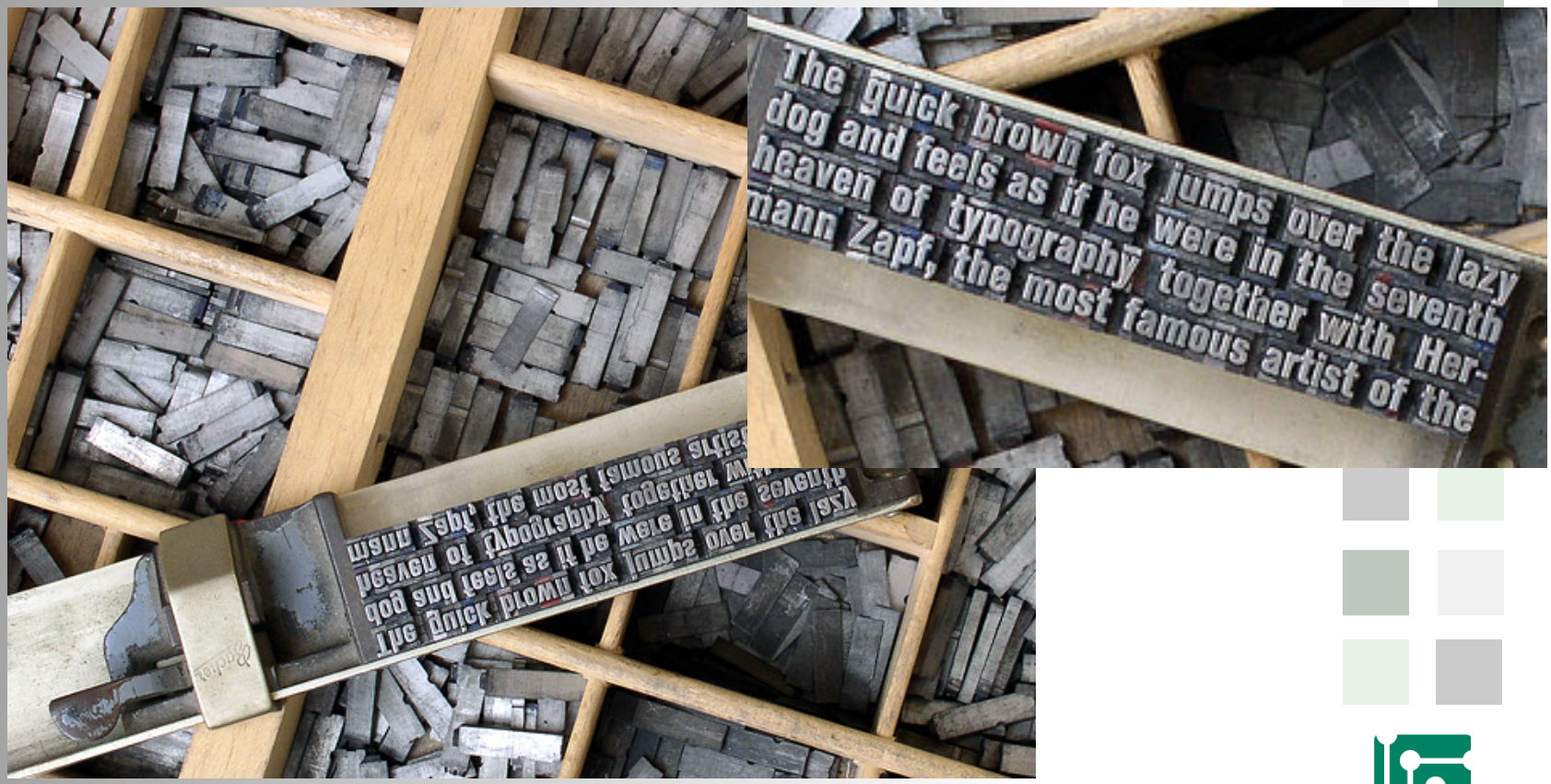
The function:

```
my_str = input("Give me a
   value")
```

- prints `"Give me a value"` on the python screen and waits till the user types something (anything), ending with Enter
- associates `my_str` with what the user typed.
- No matter what, it returns a **string**.

What's a string

The word "string" is used to indicate a sequence of characters, a compositor"s term



Printing output

```
my_int = 12
print("My var has a value of:",my_int)
```

- `print` takes a list of elements to print, separated by commas
  - if the element is a string, bracketed by " " , prints it as is
  - if the element is a variable, prints the value associated with the variable
  - after printing, moves on to a new line of output

Python name conventions

- must begin with a letter or _
  - `Ab123` is OK, but `123ABC` is not.
- may contain letters, digits, and underscores
  - `this_is_an_identifier_123`
- may be of any length
- upper and lower case letters are different
  - `LengthOfRope` is not `lengthofrope`
- names starting with _ have special meaning. Be careful (meaning, don't do it right now)

Naming variables

Our rule is "lower with under"

- means lower case letters with words separated by an underscore.
- This is the "python way". We are trying to fit in with the rules.
- See the course web page

Python comments

- A comment begins with a #
- This means that from the # to the end of that line, nothing will be interpreted by Python.
- You can write information that will help the reader with the code

Code as essay, an aside

- What is the primary goal of writing code:
  - to get it to do something
  - an essay on my problem solving thoughts
- Code is something to be read. You provide comments to help readability.

Knuth, Literate Programming (84)

"Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."

Python "types"

- integers: **5**
- floats: **1.2**
- booleans: **True**
- strings: **"anything"** or **"something"**
- lists: [,] or ["a",1,1.3]
- dictionaries: {"bill":4.0, "rich":2.0}
- others we will see

What is a type

- a type in Python essentially defines two things:
  - the internal structure of the type (what is contains)
  - the kinds of operations you can perform on things of that type

`"abc".capitalize()` is a method you can call on strings, but not integers

some types have multiple elements (collections), we"ll see those later

Fundamental Types

- Integers
  - `1, -27` (to +/- $2^{32} - 1$)
  - `123L` L suffix means any length, but potentially very slow. Python will convert if an integer gets too long automatically
- Floating Point (Real)
  - `3.14, 10., .001, 3.14e-10, 0e0`
- Booleans (True or False values)
  - `True, False` note the capital

When = doesn't mean equal

- It is most confusing at first to see the following kind of expression:

  my_int = my_int + 7

- You don't have to be a math genius to figure out something is wrong there.
- What's wrong is that = doesn't mean equal

= is assignment

- In many computer languages, = means assignment.

  my_int = my_int + 7

  lhs = rhs

- What "assignment" means is:
  - evaluate all the "stuff" on the rhs of the =
  - take the resulting value and associate it with the name on the lhs

Variable Objects

- Python maintains a list of pairs for every variable:
  - variable's name
  - variable's value
- A variable is created when a value is assigned the first time. It associates a name and a value
- subsequent assignments update the associated value.
- we say name references value
- A variable's type depends on what is assigned.

my_int = 7

Name	Value
my_int	7

Assignment Statement

- Example: result_int = 2 + 3 * 5
  - evaluate expression (2+3*5): 17
  - change the value of result_int to reference 17
- Example (val_int has value 2):

  val_int = val_int + 3
  - evaluate expression (val_in+3): 5
  - change the value of val_int to reference 5

What can go on the lhs

- There are limits therefore as to what can go on the lhs of an assignment statement.
- The lhs must indicate a name with which a value can be associated
- must follow the naming rules

my_int = 5	Yes
my_int + 5 = 7	No

Type follows the object

In Python, type follows the object, not the variable

- variables have no type information. They are an association of a name with an object
- objects associated with a variable have type
- C++/Java people, this is a big change!

An exercise

- create a file "division.py"
- prompt for two numbers
- divide the first number by the second, save the result
- print the two provided numbers and the resulting quotient
- try it out with some numbers, see what you get.

Some Data Analysis

Sunspots

MICHIGAN STATE UNIVERSITY

ICER

Best to have an application

You have the basics, now let's develop the rest using an example

We"ll do some data analysis on sunspot data
http://solarscience.msfc.nasa.gov/SunspotCycle.shtml

Could do lots of things

- data analysis is common, important, pervasive
- get the general idea of how it works to write a program
- will show some other stuff later

sunspots

YEAR	MON	SSN	DEV
1749	1	58.0	24.1
1749	2	62.6	25.1
1749	3	70.0	26.6
1749	4	55.7	23.6
1749	5	85.0	29.4
1749	6	83.5	29.2
1749	7	94.8	31.1
1749	8	66.3	25.9
1749	9	75.9	27.7
1749	10	75.5	27.7
1749	11	158.6	40.6
1749	12	85.2	29.5
1750	1	73.3	27.3
1750	2	75.9	27.7

Text file, where each line has:

- year
- month
- Sunspot count
- Different count
  - we'll ignore this one

a small version

It's a big file, let's use a very small version we can experiment with.

little.txt

Get data from the file

We'll play some games in the console, get a feel for what's going on, then write the program on the left side.

Create a new file on the left side, call it sunspot.py

get data from the file

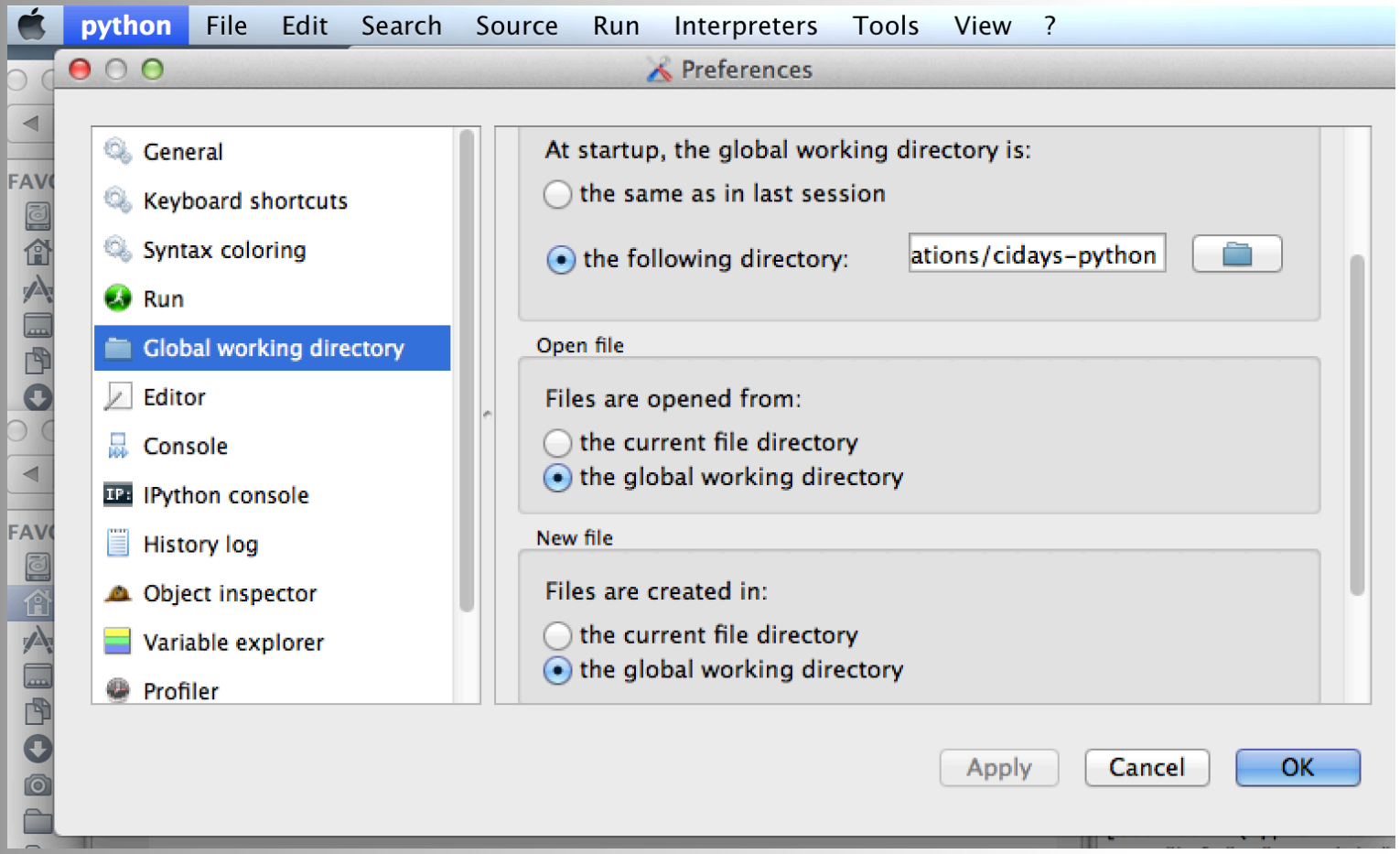
```
file_obj = open("little.txt")
```

Makes a connection (represented by the variable `file_obj`) to the text file.

open requires string, the name of the file.

where is the file?

mac, python→Preferences



windows

Same, but under Tools →Preferences

try this in console

Having run the open command, try the below

```
for line in file_obj:
    print(line)
```

More Compound Statement

keyword

optional expression

```
keywrd expression :
   statement
   statement
   statement
```

ends all headers

"suite" of statements

indentation under the keyword

to read a file

Many ways but the easiest is iteration.

close then open the file

```
file_obj.close()
file_obj = open("little.txt")
for line in file_obj:
    process line
```

Each iteration gets one line (ending in carriage return) of the file *as a string*!

All file interaction is by string

For this class, we only work with text files (files with characters/strings) so all iteraction is by strings:

- read only as a string
- must write only strings (meaning you might have to convert some things)

try this

```
"this is a test".split()
```

The dot ('.') means:

call the function (rhs of the dot) on the object on the the lhs (a string).

component parts

Split the string into pieces (based on some separator, by default).

the string split method

- What the split method returns of a string is a list of the individual string elements, broken into pieces by a particular character
  - the default is whitespace

examples

```
'my mother the car'.split() # default
→['my', 'mother', 'the', 'car']

'name, date, age'.split(',') # on comma
→['name', ' date', ' age']
```

note the space in front of `' date'` and `' age'`.

Indexing objects (string)

character	h	e	l	l	o		w	o	r	l	d
index	0	1	2	3	4	5	6	7	8	9	10
										…-2	-1

- every character has an *index*, a sequence number, starting at 0
- the index operator is `[ ]`. Sequence number goes between the `[ ]`

```
my_str = 'hello world'
my_str[2] ⇒ 'l'
my_str[-1] ⇒ 'd'
my_str[11] ⇒ ERROR, index out of range
```

Lists

Strings are a sequence of characters (really strings). What if I want a sequence of other things.

That's a list. A list is bracketed by `[ ]` (bit confusing, two meanings to `[ ]`)

Can be a sequence of any types, even mixed.

Lists, sequence of elements

```
my_list = [0, 11, "hi", 1.2]
```

also have an index

```
my_list[0] → 0
my_list[1] → 11
my_list[2] → "hi"
my_list[3] → 1.2
```

get parts of the line

```
for line in file_obj:
    lst = line.split()
    print(lst[2])
```

YEAR	MON	SSN	DEV
1749	1	58.0	24.1
1749	2	62.6	25.1
1749	3	70.0	26.6
1749	4	55.7	23.6
1749	5	85.0	29.4
1749	6	83.5	29.2
1749	7	94.8	31.1
1749	8	66.3	25.9
1749	9	75.9	27.7
1749	10	75.5	27.7
1749	11	158.6	40.6
1749	12	85.2	29.5
1750	1	73.3	27.3
1750	2	75.9	27.7

- year is index 0
- month is index 1
- SSN is index 2
  - that's what we want

sum up the averages

```
the_sum = 0.0
for line in file_obj:
    lst = line.split()
    the_sum = the_sum + float(lst[2])
```

Only new thing is the `float()`

- everything is a string, we need to turn that third field (index 2) to an floating point number
- `the_sum` is the sum of all SSN

whole program

```
the_sum = 0.0
file_obj = open("little.txt")
for line in file_obj:
    lst = line.split()
    the_sum = the_sum+float(lst[2])
print("sum is:",the_sum)
```

Exercise 2

- file "grades.txt" in the directory
- each line is name followed by 3 scores
- print the name and the average for each line

run it

```
>python3.3 avg.py
Traceback (most recent call last):
  File "avg.py", line 5, in <module>
    the_sum = the_sum + float(lst[2])
ValueError: could not convert string to float:
'SSN'
```

What went wrong?

first line of the file

we need to ignore that first line (the column headers).

How to do?

conditional execution

```
if boolean-condition:
    # stuff to do if true
else:
    # stuff to do if false
```

Do one or the other suite

```
the_sum = 0.0
file_obj = open("little.txt")
for line in file_obj:
   lst = line.split()
   if lst[0] != "YEAR":
      the_sum = the_sum + float(lst[2])
print("sum is:",the_sum)
```

conditionals

- == equal
- != not equal
- < less than
- > greater than
- <= less than or equal
- >= greater than or equal

how about the average

```
the_sum = 0.0
cnt = 0
file_obj = open("little.txt")
for line in file_obj:
   lst = line.split()
   if lst[0] != "YEAR":
      the_sum = the_sum + float(lst[2])
      cnt = cnt + 1
print "sum is:",the_sum,\
      "average is:",the_sum/cnt
```

import

Not all the packages are available in the default Python set

You can import a new package and use the stuff provided

If you import, then you precede all references with the name of the package

we can graph using pylab

Rich Enbody will show more in the afternoon session, but basic plotting is awfully easy.

Their motto:

"matplotlib tries to make easy things easy and hard things possible"

collect the numbers

To plot the numbers, we need them each individually collected in a list.

Lists have methods as well. One is .append()

```
my_lst = [1,2,3]
my_lst.append(4)
print(my_lst) → [1,2,3,4]
```

maybe graph it?

```
import pylab

month_averages= []
file_obj = open("little.txt")
for line in file_obj:
    lst = line.split()
    if lst[0] != "YEAR":
        month_averages).append(lst[2])
pylab.plot(month_averages)
pylab.show()
```

graph the januarys

```
import pylab

jans= []
file_obj = open("little.txt")
for line in file_obj:
    lst = line.split()
    if lst[1] == "1":
        jans.append(lst[2])
pylab.plot(jans)
pylab.show()
```

average again

We have all the numbers in a list now, how to get that average back?

Try this:

```
len(month_averages)
```

great, we know how long.

try this

Sum them up with sum

try this

```
sum(month_averages)
```

What went wrong? How to fix

```
import pylab

avg = []
file_obj = open("little.txt")
for line in file_obj:
    lst = line.split()
    if lst[0] != "YEAR":
        avg.append(float(lst[2]))
print("Average is:", sum(avg)/len(avg))
pylab.plot(avg)
pylab.show()
```

isn't average somewhere?

Yes, all kinds of stats can be done in numpy

```
import numpy

numpy.mean(month_averages)
```

what others