LECTURE 11
PROJECTS IN PYTHON

Fundamentals of Programming

COMP1005 / COMP5005

Discipline of Computing

Updated 24/11/19

Copyright Warning

Learning Outcomes

- Access and assess packages in the Python Package Index
- Evaluate risks in using packages and code from other developers
- Be aware of how to build and share a package
- Know some useful tools to support your software development
- Practice designing and implementing projects

PROGRAMMING ENVIRONMENTS

Fundamentals of Programming

Lecture 11

Programming Environments : CLI

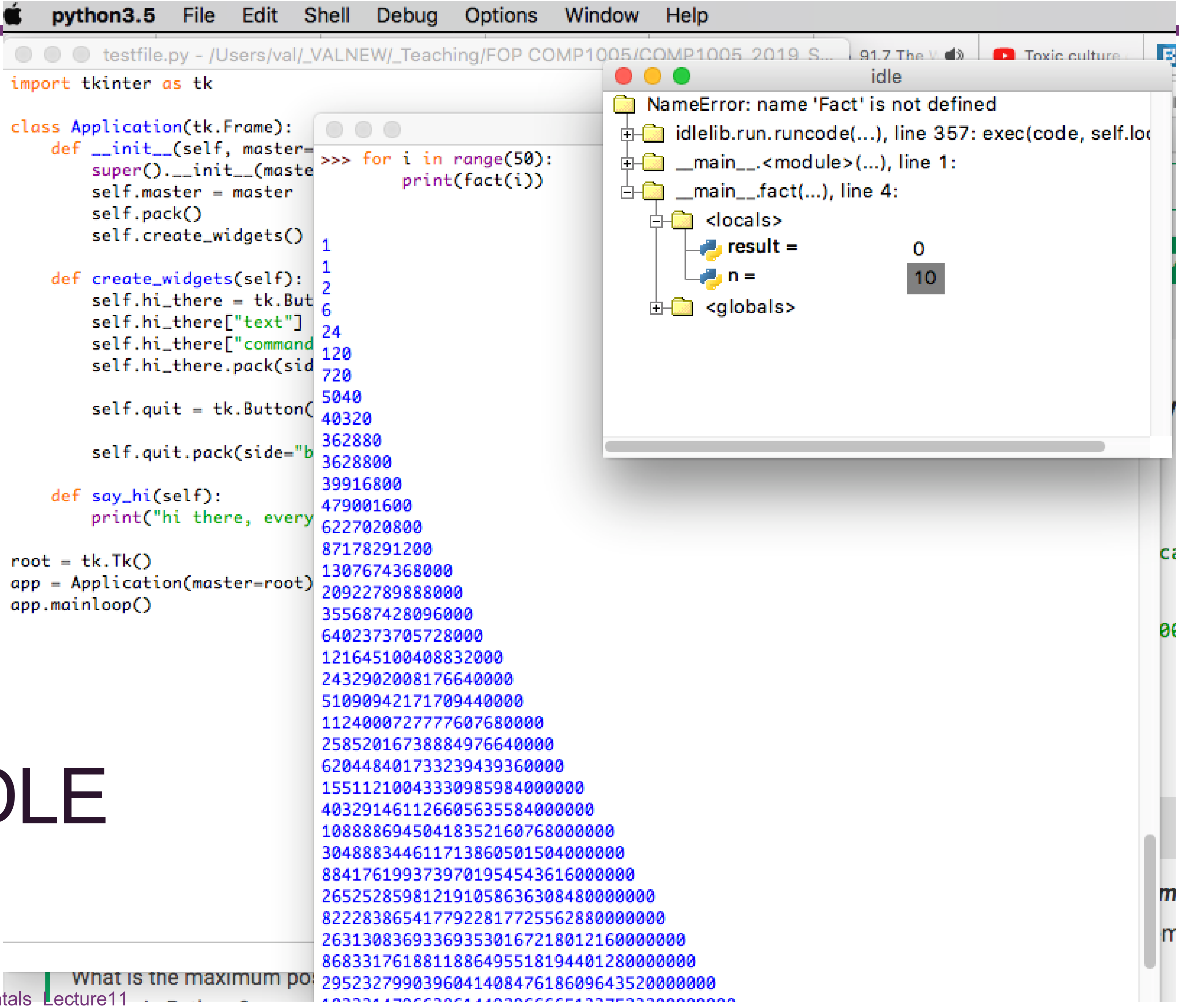
- A priority in this unit has been to expose you to Unix (Linux) and coding in the basic environment
- For this we've used vim (aka vi) and written scripts to run at the command line
- Other Command Line Interface (CLI) text editors are:
  - nano
  - gedit
  - atom
  - emacs
  - and many more: https://fossbytes.com/9-best-text-editors-linux-programming-2017/
- They're all "easy" to learn and some people get very passionate about them!

Programming Environments : IDLE

- **I**ntegrated **D**evelopment and **L**earning **E**nvironment
- Part of the Python installation
- Features:
  - Coded in 100% pure Python, using the tkinter GUI toolkit
  - Cross-platform: works mostly the same on Windows, Unix, and macOS
  - Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
  - Multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
  - Debugger with persistent breakpoints, stepping, and viewing of global and local namespaces

https://docs.python.org/3/library/idle.html

IDLE


python3.5
File
Edit
Shell
Debug
Options
Window
Help
testfile.py - /Users/val/_VALNEW/_Teaching/FOP COMP1005/COMP1005_2019_S...
91.7 The
Toxic culture
idle

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=
        super().__init__(maste
        self.master = master
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = tk.But
        self.hi_there["text"]
        self.hi_there["command
        self.hi_there.pack(sid

        self.quit = tk.Button(

        self.quit.pack(side="b

    def say_hi(self):
        print("hi there, every

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

```
>>> for i in range(50):
        print(fact(i))


1
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
6402373705728000
121645100408832000
2432902008176640000
51090942171709440000
1124000727777607680000
25852016738884976640000
620448401733239439360000
15511210043330985984000000
403291461126605635584000000
10888869450418352160768000000
304888344611713860501504000000
8841761993739701954543616000000
265252859812191058636308480000000
8222838654177922817725562880000000
263130836933693530167218012160000000
8683317618811886495518194401280000000
295232799039604140847618609643520000000
```

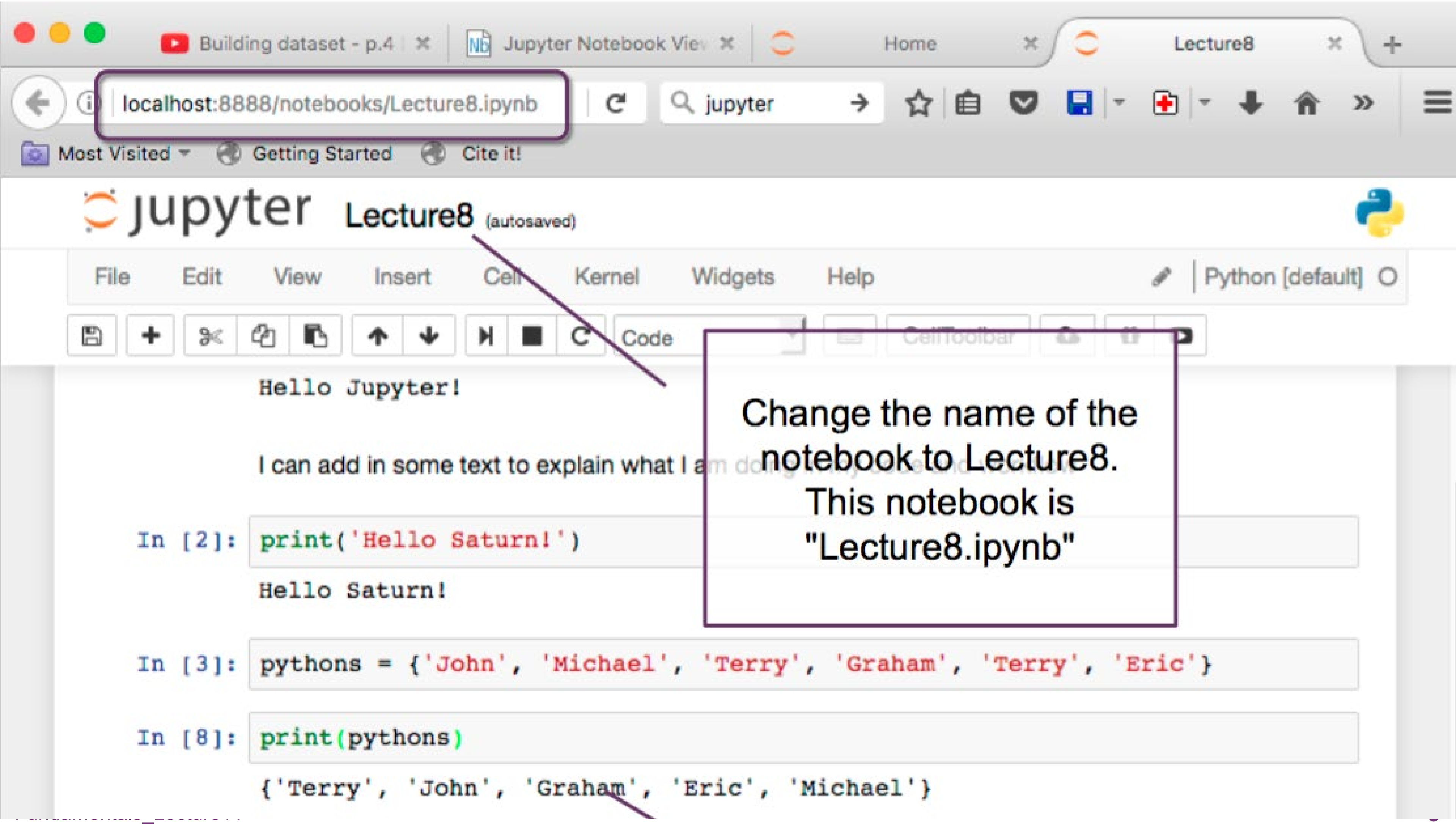
```
NameError: name 'Fact' is not defined
  idlelib.run.runcode(...), line 357: exec(code, self.loc
  __main__.<module>(...), line 1:
  __main__.fact(...), line 4:
    <locals>
      result =        0
      n =             10
    <globals>
```

What is the maximum po

Programming Env. : Jupyter Notebook

- Features:
  - Combine code and text, formatting, equations
  - Save as PDF
  - Share easily with others
  - Environments and kernels for selecting software versions
  - Inline images and plots
  - %run to run Python scripts
  - Auto-indent, auto-brackets, syntax highlighting

Programming Env. : Jupyter Notebook



```
Hello Jupyter!
```

I can add in some text to explain what I am do

```
In [2]: print('Hello Saturn!')

Hello Saturn!

In [3]: pythons = {'John', 'Michael', 'Terry', 'Graham', 'Terry', 'Eric'}

In [8]: print(pythons)

{'Terry', 'John', 'Graham', 'Eric', 'Michael'}
```

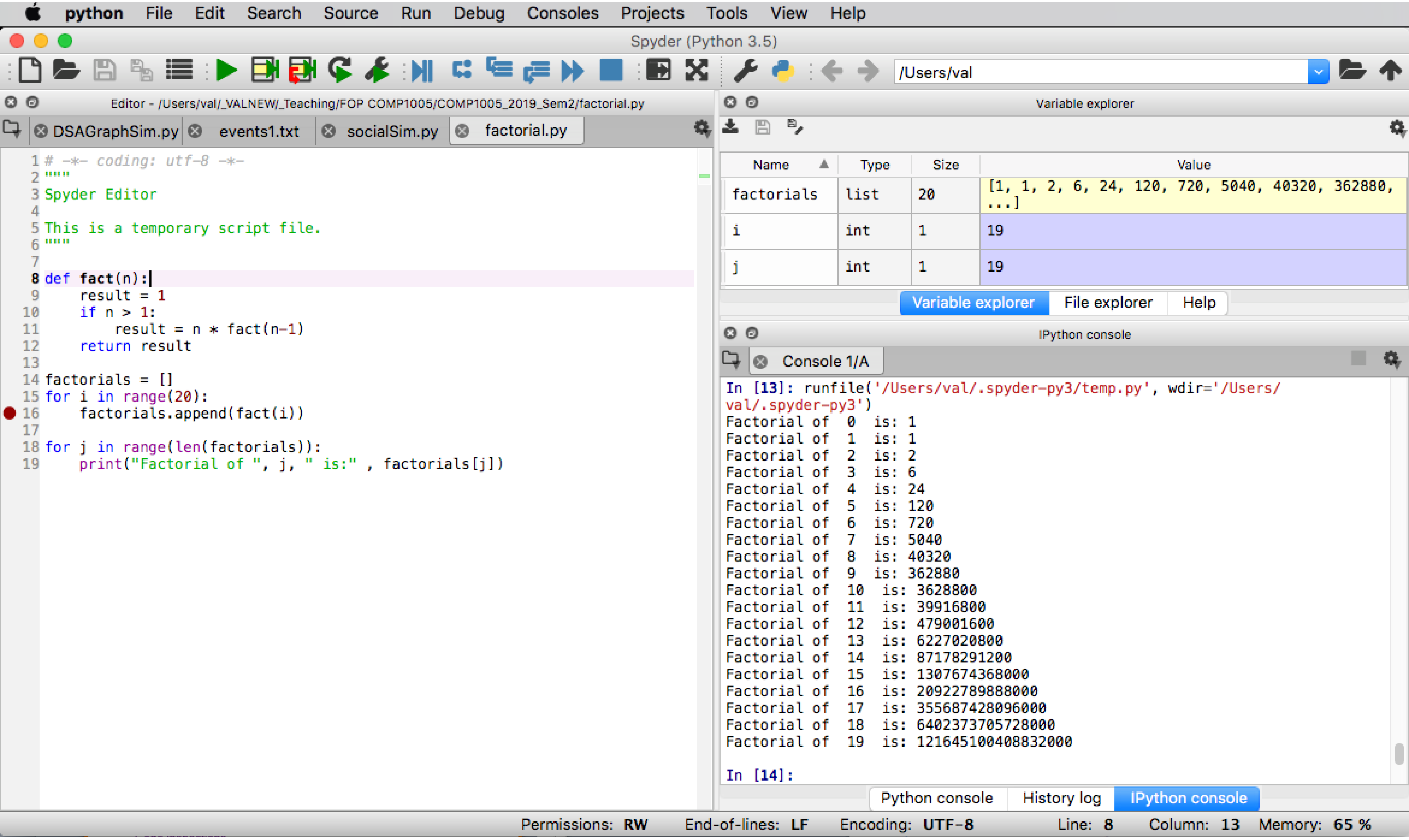
Change the name of the notebook to Lecture8. This notebook is "Lecture8.ipynb"

Programming Env. : Spyder

- Spyder is a powerful scientific environment written in Python, for Python
- Designed by and for scientists, engineers and data analysts
- Included in most Python installations
- Features:
  - Advanced editing, analysis, debugging, and profiling functionality for code development
  - Data exploration, interactive execution, deep inspection, and visualization capabilities.
  - Its abilities can be extended even further via its plugin system and API.

https://www.spyder-ide.org/

Programming Env. : Spyder



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

This is a temporary script file.
"""

def fact(n):
    result = 1
    if n > 1:
        result = n * fact(n-1)
    return result

factorials = []
for i in range(20):
    factorials.append(fact(i))

for j in range(len(factorials)):
    print("Factorial of ", j, " is:" , factorials[j])
```

Name	Type	Size	Value
factorials	list	20	[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...]
i	int	1	19
j	int	1	19

```
In [13]: runfile('/Users/val/.spyder-py3/temp.py', wdir='/Users/
val/.spyder-py3')
Factorial of  0  is: 1
Factorial of  1  is: 1
Factorial of  2  is: 2
Factorial of  3  is: 6
Factorial of  4  is: 24
Factorial of  5  is: 120
Factorial of  6  is: 720
Factorial of  7  is: 5040
Factorial of  8  is: 40320
Factorial of  9  is: 362880
Factorial of  10  is: 3628800
Factorial of  11  is: 39916800
Factorial of  12  is: 479001600
Factorial of  13  is: 6227020800
Factorial of  14  is: 87178291200
Factorial of  15  is: 1307674368000
Factorial of  16  is: 20922789888000
Factorial of  17  is: 355687428096000
Factorial of  18  is: 6402373705728000
Factorial of  19  is: 121645100408832000

In [14]:
```

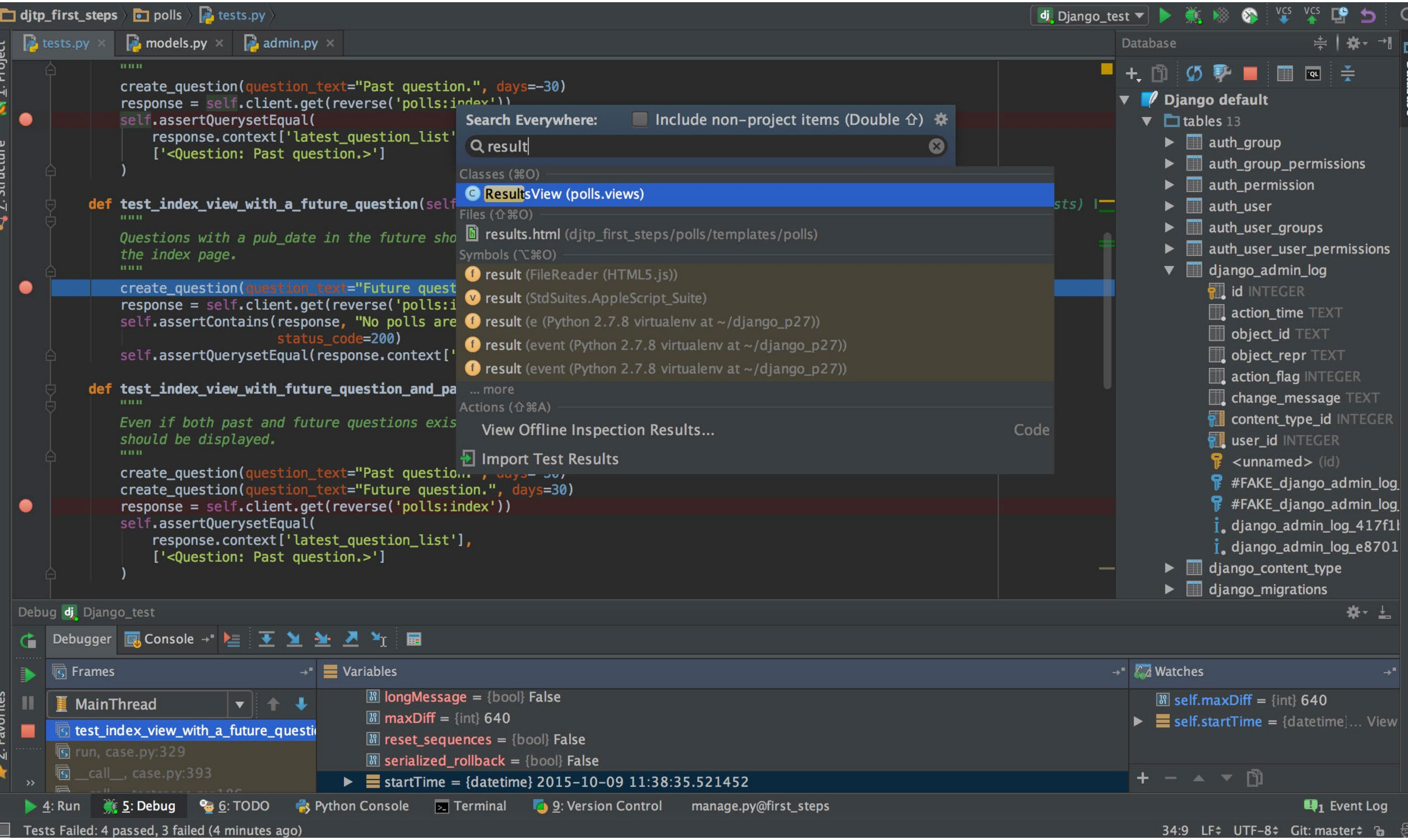
Permissions: RW   End-of-lines: LF   Encoding: UTF-8   Line: 8   Column: 13   Memory: 65 %

Programming Env. : Pycharm

- "The Python IDE for Professional Developers"
- Features (community version):
  - Intelligent Python editor
  - Graphical debugger and test runner
  - Navigation and Refactorings
  - Code inspections
  - VCS support
- Paid version:
  - Scientific tools, Web development, Python web frameworks, Python Profiler, Remote development capabilities, Database & SQL support

https://www.jetbrains.com/pycharm/

Programming Env. : Pycharm



djtp_first_steps › polls › tests.py
Django_test
tests.py × models.py × admin.py ×
Database
1: Project
7: Structure
2: Favorites

```
        """
        create_question(question_text="Past question.", days=-30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'
            ['<Question: Past question.>']
        )

    def test_index_view_with_a_future_question(self
        """
        Questions with a pub_date in the future sho
        the index page.
        """
        create_question(question_text="Future quest
        response = self.client.get(reverse('polls:i
        self.assertContains(response, "No polls are
                            status_code=200)
        self.assertQuerysetEqual(response.context['

    def test_index_view_with_future_question_and_pa
        """
        Even if both past and future questions exis
        should be displayed.
        """
        create_question(question_text="Past questio
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )
```

Search Everywhere: □ Include non-project items (Double ⇧)

result

Classes (⌘O)
- ResultsView (polls.views)

Files (⇧⌘O)
- results.html (djtp_first_steps/polls/templates/polls)

Symbols (⌥⌘O)
- result (FileReader (HTML5.js))
- result (StdSuites.AppleScript_Suite)
- result (e (Python 2.7.8 virtualenv at ~/django_p27))
- result (event (Python 2.7.8 virtualenv at ~/django_p27))
- result (event (Python 2.7.8 virtualenv at ~/django_p27))

... more

Actions (⇧⌘A)
- View Offline Inspection Results... Code
- Import Test Results

Django default
- tables 13
  - auth_group
  - auth_group_permissions
  - auth_permission
  - auth_user
  - auth_user_groups
  - auth_user_user_permissions
  - django_admin_log
    - id INTEGER
    - action_time TEXT
    - object_id TEXT
    - object_repr TEXT
    - action_flag INTEGER
    - change_message TEXT
    - content_type_id INTEGER
    - user_id INTEGER
    - <unnamed> (id)
    - #FAKE_django_admin_log
    - #FAKE_django_admin_log
    - django_admin_log_417f1
    - django_admin_log_e8701
  - django_content_type
  - django_migrations

Debug Django_test

MainThread
- test_index_view_with_a_future_questi
- run, case.py:329
- __call__, case.py:393

```
longMessage = {bool} False
maxDiff = {int} 640
reset_sequences = {bool} False
serialized_rollback = {bool} False
startTime = {datetime} 2015-10-09 11:38:35.521452
```

```
self.maxDiff = {int} 640
self.startTime = {datetime}... View
```

Tests Failed: 4 passed, 3 failed (4 minutes ago)

34:9 LF÷ UTF-8÷ Git: master÷

https://www.jetbrains.com/pycharm/

DEBUGGING AND CODE INSPECTION TOOLS

Fundamentals of Programming

Lecture 11

Debugging Tools

- There are some automated tools which can help us to debug errors
- They can also to keep our code as correct as possible to minimise the chances of new errors creeping in
- Some of these tools analyse our program’s syntax, reporting errors and bad programming style
- Others let us analyse the program as it is running

http://python-textbok.readthedocs.io/en/1.0/Errors_and_Exceptions.html

Examples – code checkers

- Pyflakes parses code instead of importing it, which means that it can’t detect as many errors as other tools – but it is also safer to use
  - It doesn't execute broken code which does permanent damage to our system
- Pylint and PyChecker import the code that they check, and they produce more extensive lists of errors and warnings. They are used by programmers wanting greater functionality
- Pep8 specifically targets bad coding style – it checks whether our code conforms to Pep 8, a specification document for good coding style.

pep8 – style checking

```
Mac:Lecture11$ pep8 accounts.py
accounts.py:3:5: E301 expected 1 blank line, found 0
accounts.py:11:8: E271 multiple spaces after keyword


Mac:Lecture11$ pep8 testAccounts.py
testAccounts.py:3:1: E302 expected 2 blank lines, found 1
testAccounts.py:6:80: E501 line too long (111 > 79
characters)


Mac:Lecture11$ pep8 shelters.py
shelters.py:1:24: E231 missing whitespace after ','
shelters.py:1:28: E231 missing whitespace after ','
shelters.py:1:33: E231 missing whitespace after ','
shelters.py:28:1: W391 blank line at end of file
```

pyflakes

```
Mac:Lecture11 $ pyflakes shelters.py
shelters.py:1: 'animals.Cat' imported but unused
shelters.py:1: 'animals.Dog' imported but unused
shelters.py:1: 'animals.Bird' imported but unused

Mac:Lecture11 $ pyflakes accounts.py

Mac:Lecture11 $ pyflakes testAccounts.py

Mac:Lecture11 $ pyflakes animals.py
animals.py:57: local variable 'temp' is assigned
to but never used
animals.py:68: local variable 'temp' is assigned
to but never used
```

Pylint – extract of output

```
Messages by category
--------------------


+-----------+-------+---------+-----------+
|type       |number |previous |difference |
+===========+=======+=========+===========+
|convention |15     |15       |=          |
+-----------+-------+---------+-----------+
|refactor   |0      |0        |=          |
+-----------+-------+---------+-----------+
|warning    |0      |0        |=          |
+-----------+-------+---------+-----------+
|error      |0      |0        |=          |
+-----------+-------+---------+-----------+


Messages
--------


+------------------+------------+
|message id        |occurrences |
+==================+============+
|invalid-name      |14          |
+------------------+------------+
|missing-docstring |1           |
+------------------+------------+


Global evaluation
-----------------
Your code has been rated at 4.83/10 (previous run: 4.64/10, +0.18)
```

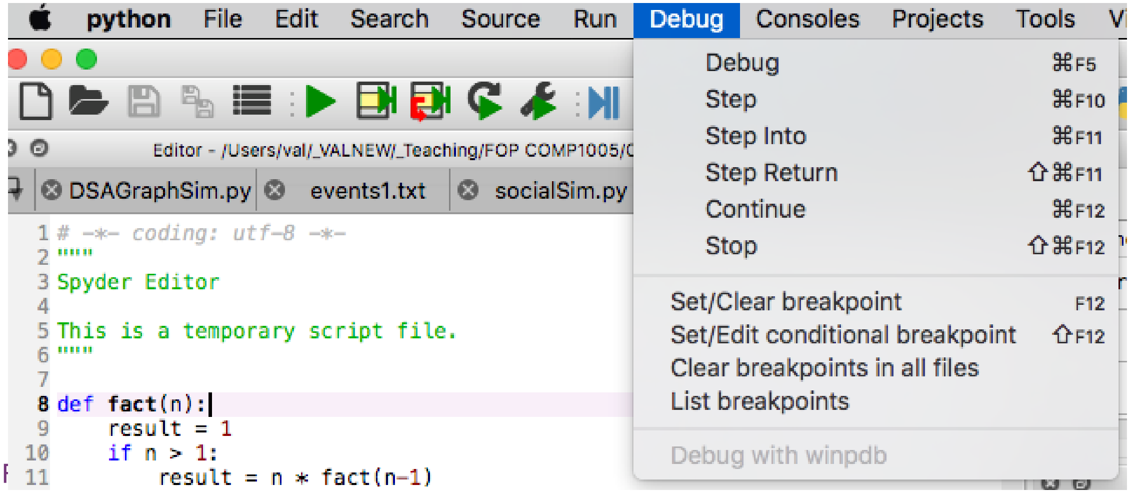
pdb

- The module pdb provides an interactive source code debugger for Python programs
- It supports :
  - setting (conditional) breakpoints
  - single stepping at the source line level
  - inspection of stack frames
  - source code listing
  - evaluation of arbitrary Python code in the context of any stack frame
- It also supports post-mortem debugging (after a crash) and can be called under program control

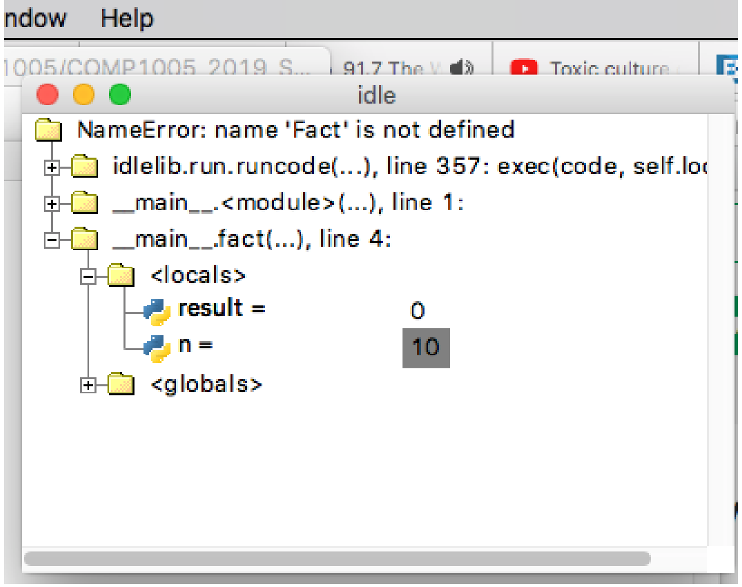
https://docs.python.org/3.3/library/pdb.html

Integrated Development Environments

- IDLE, Spyder and Pycharm all provide debugging support
- Options include :
  - setting (conditional) breakpoints
  - single stepping at the source line level
  - stack trace inspection



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

This is a temporary script file.
"""

def fact(n):
    result = 1
    if n > 1:
        result = n * fact(n-1)
```



```
NameError: name 'Fact' is not defined
  idlelib.run.runcode(...), line 357: exec(code, self.loc
  __main__.<module>(...), line 1:
  __main__.fact(...), line 4:
    <locals>
      result =    0
      n =         10
    <globals>
```

VERSION CONTROL

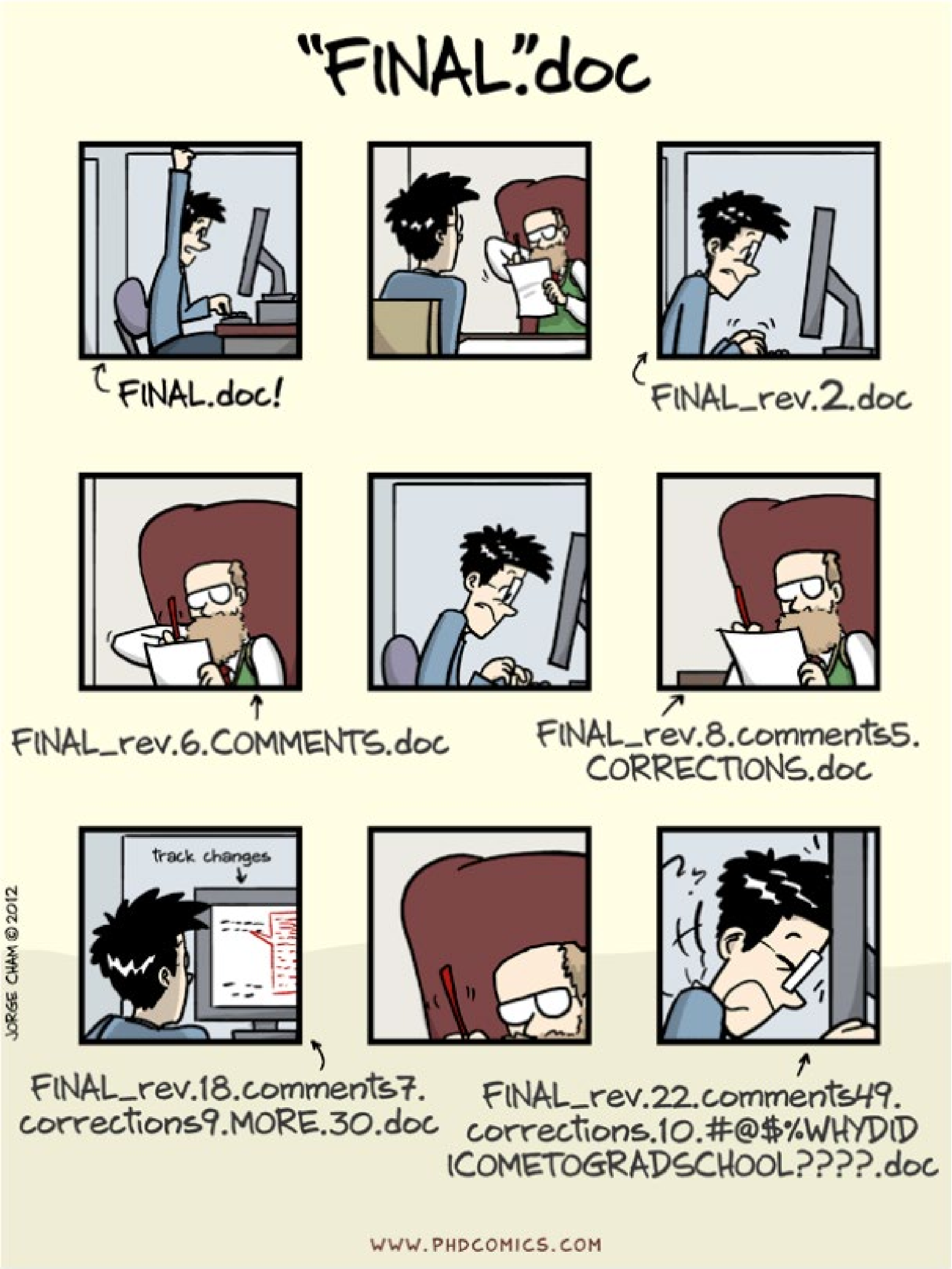
Fundamentals of Programming

Lecture 11

Version Control

- Version Control (aka Revision Control aka Source Control) lets you track your files over time
- Why do you care?
- So when you mess up you can easily get back to a previous working version

https://betterexplained.com/articles/a-visual-guide-to-version-control/


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
JORGE CHAM © 2012
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
WWW.PHDCOMICS.COM

Why Version Control?

- **You’ve probably made your own** version control system without realizing it had such a geeky name.
- Do you have files like this?
  - FOP_Assignment.doc
  - FOP_Assignment_v1.doc
  - FOP_Assignment_final.doc
  - ResumeMar2017b.doc
  - FOP_Part1.py
  - FOP_Part1_old.py
  - FOP_Part1_20May.py

DIY Version Control

- **It’s why we use “Save As” or "Save copy as".**
- You want to checkpoint your work, then keep making changes

- It’s a common need, and the DIY approach is:
  - Make a **single backup copy**
    - importantProg.py.old
  - If we’re clever, we add a **version number or date**:
    - importantProg_V1.py
    - importantProg_12March2019.py
  - We may even save our work to a **shared folder** so team members can see and edit files without sending them over email

Managing code

- Our shared folder/naming system may be OK for assignments or one-time papers.
- But for larger software projects?
- There must be a better way!

- Surely someone has made a program to manage all these versions for us…

Version Control Benefits

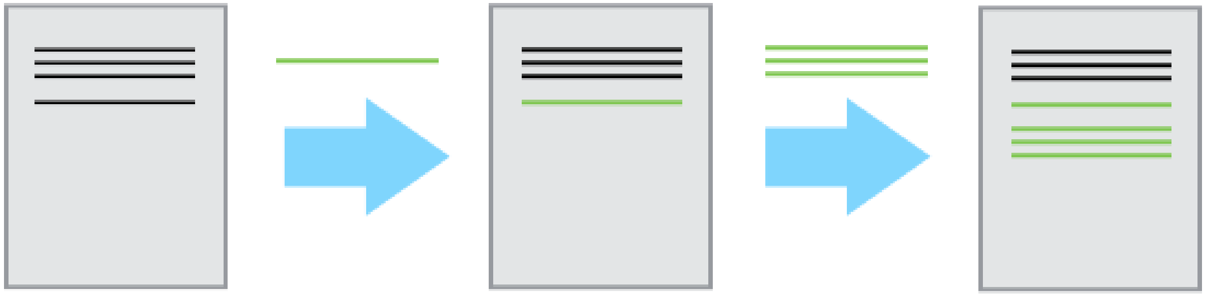
- Large projects with many authors need a Version Control System (VCS) to track changes and avoid chaos
- A good VCS does the following:
    - **Backup and Restore.** Files are saved as they are edited, and you can jump to any moment in time.
    - **Synchronization.** Lets people share files and stay up-to-date with the latest version.
    - **Short-term undo.** Changed your mind or lost some useful code? Throw away your changes and go back to the “last known good” version in the database.
    - **Long-term undo.** Suppose you made a change a year ago, and it had a bug. Jump back to the old (working) version, and see what change was made that day.

Version Control Benefits

- A good VCS does the following (continued):
  - **Track Changes**. As files are updated, you can leave messages explaining why the change happened (stored in the VCS). This makes it easy to see how a file is evolving over time, and why.
  - **Track Ownership.** A VCS tags every change with the name of the person who made it. Helpful for blamestorming or giving credit.
  - **Sandboxing.** Make temporary changes in an isolated area, test and work out the kinks before “checking in” your changes.
  - **Branching and merging**. You can **branch** a copy of your code into a separate area and modify it in isolation (tracking changes). Later, you can **merge** your work back into the common area.
- Shared folders are quick and simple, but can’t do all of this

Tracking changes

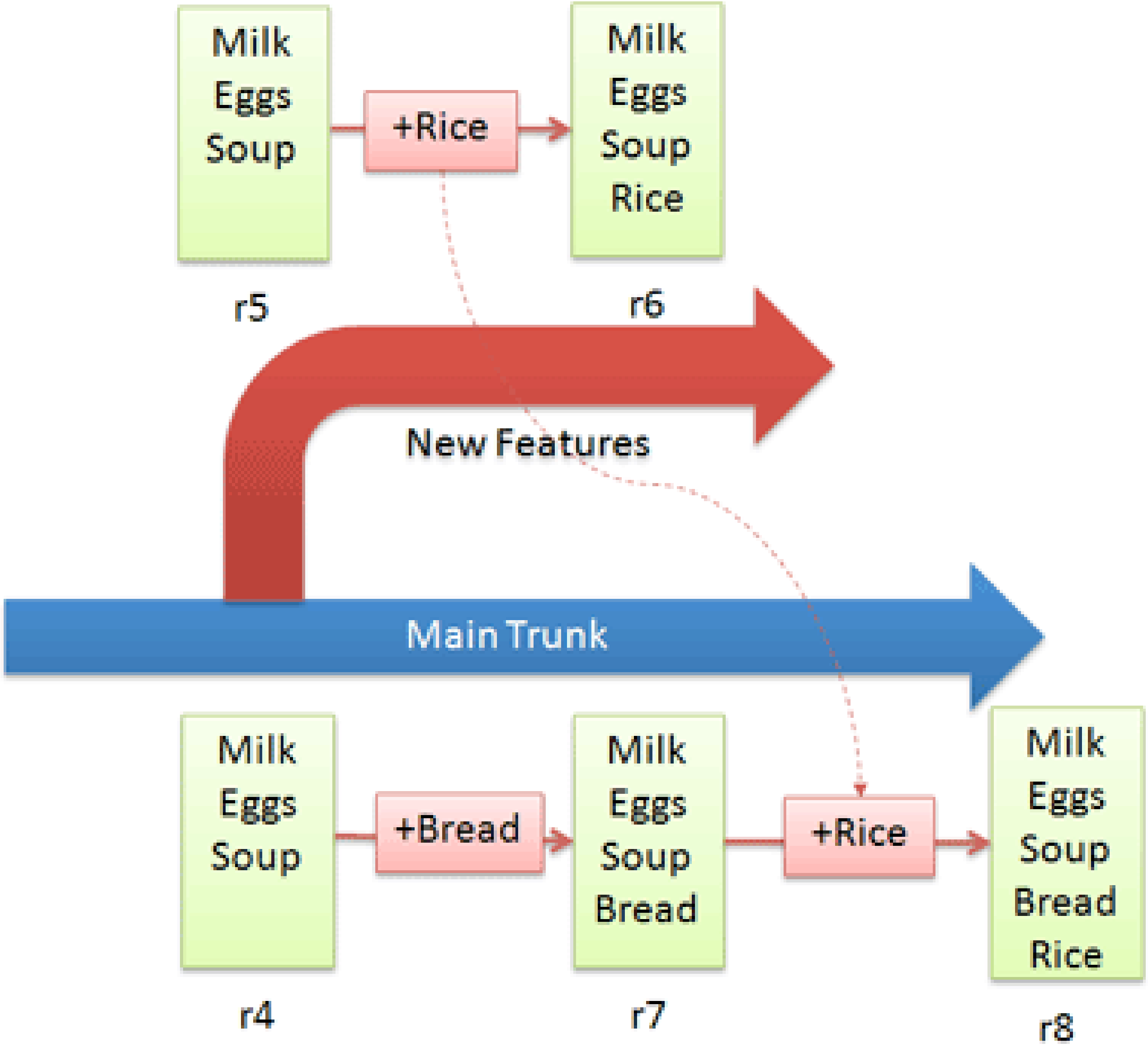
- Version control systems start with a base version of the document and then record changes you make each step of the way.
- You can think of it as a recording of your progress: you can rewind to start at the base document and play back each change you made, eventually arriving at your more recent version.



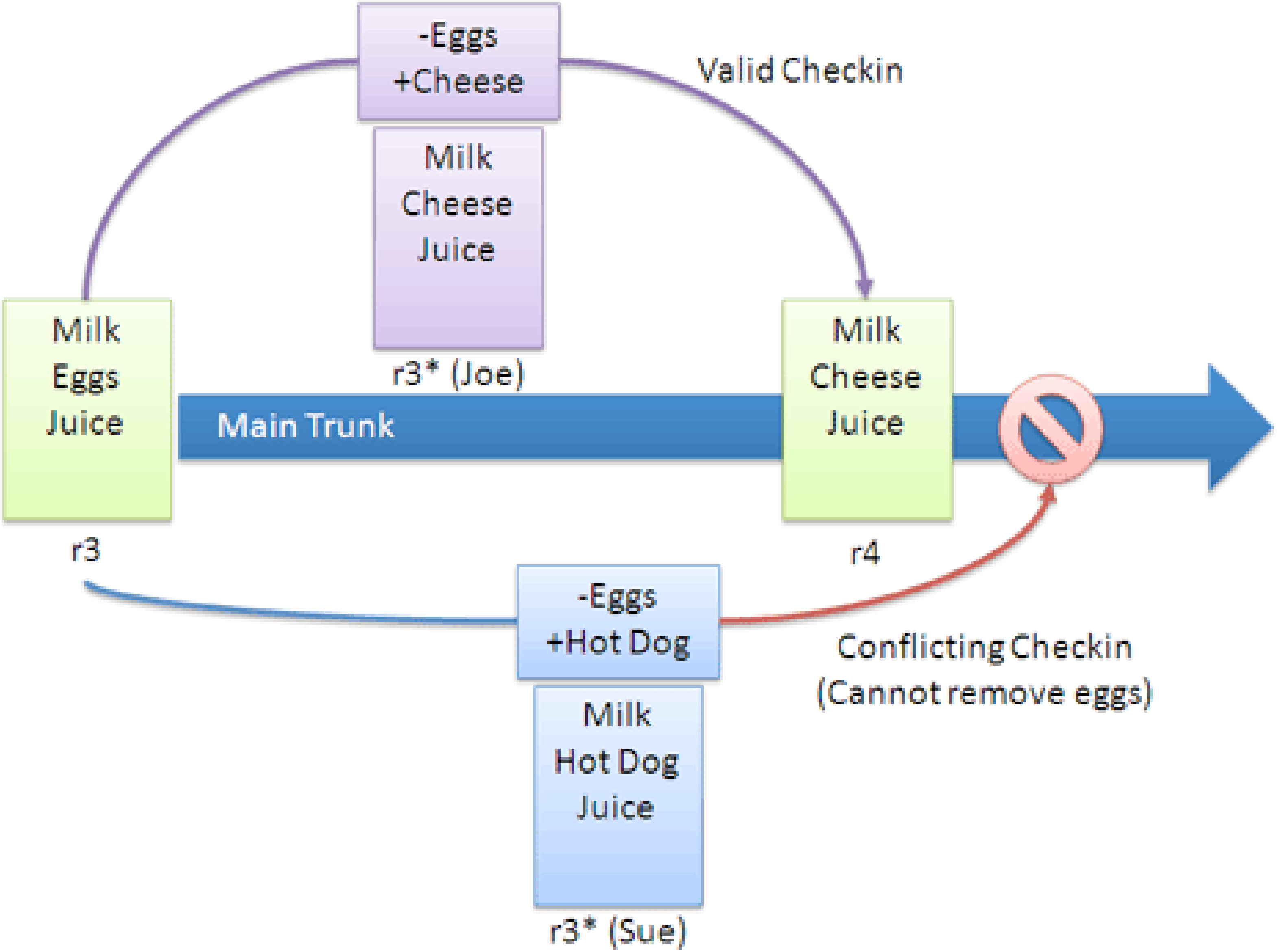
Terminology

- Most version control systems involve the following concepts, though the labels may be different.
- Basic Setup
  - **Repository (repo)**: The database storing the files.
  - **Server**: The computer storing the repo.
  - **Client**: The computer connecting to the repo.
  - **Working Set/Working Copy**: Your local directory of files, where you make changes.
  - **Trunk/Main**: The primary location for code in the repo. Think of code as a family tree — the trunk is the main line.

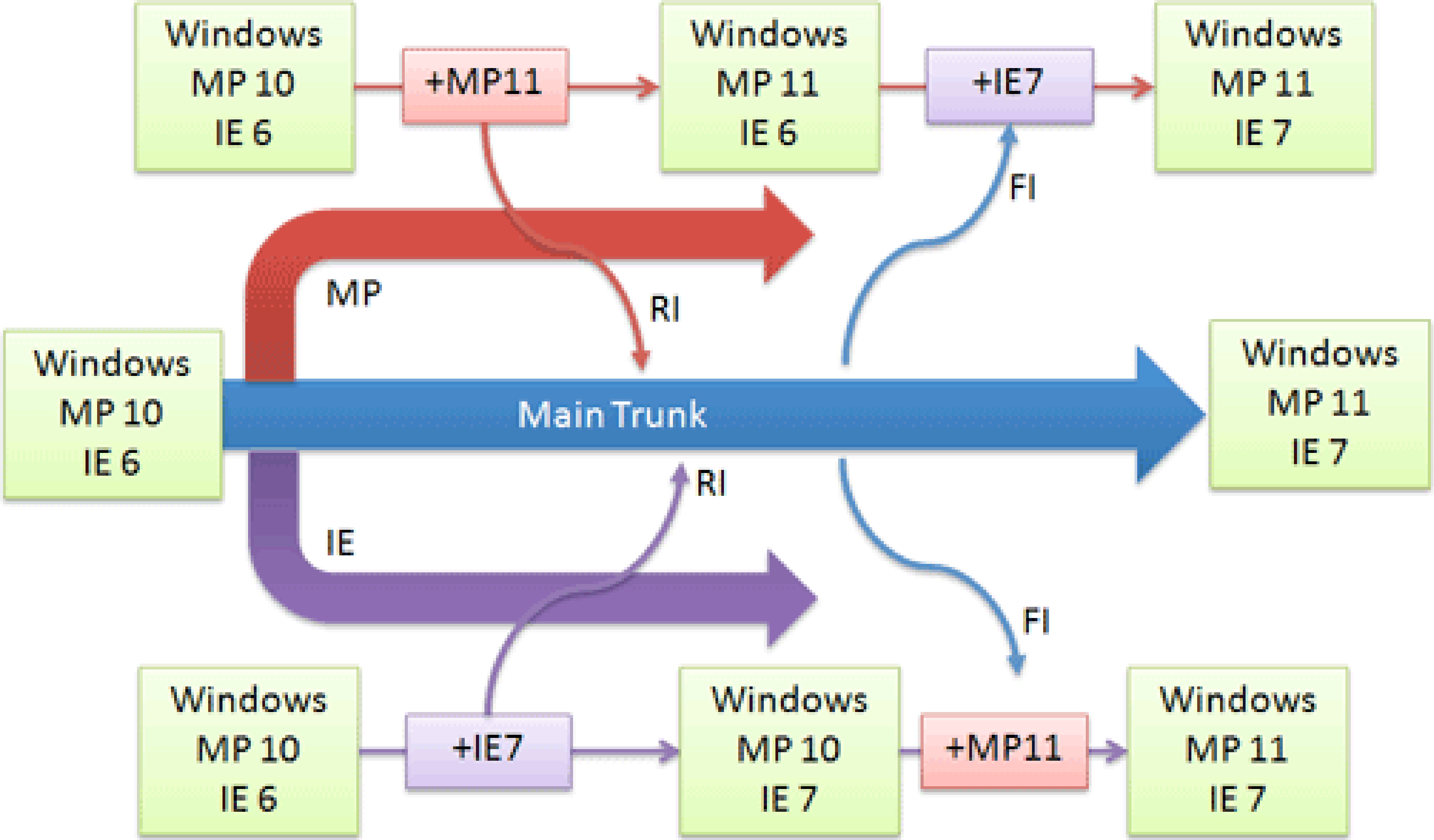
Merging


Milk
Eggs
Soup
r5
+Rice
Milk
Eggs
Soup
Rice
r6
New Features
Main Trunk
Milk
Eggs
Soup
r4
+Bread
Milk
Eggs
Soup
Bread
r7
+Rice
Milk
Eggs
Soup
Bread
Rice
r8

Conflicts


-Eggs
+Cheese
Valid Checkin
Milk
Cheese
Juice
r3* (Joe)
Milk
Eggs
Juice
Main Trunk
Milk
Cheese
Juice
r3
r4
-Eggs
+Hot Dog
Conflicting Checkin
(Cannot remove eggs)
Milk
Hot Dog
Juice
r3* (Sue)

Managing Windows



git ↗

mercurial (hg)

bazaar

subversion (svn)

version control

concurrent version system (cvs)

perforce

visual source safe

Git Advantages

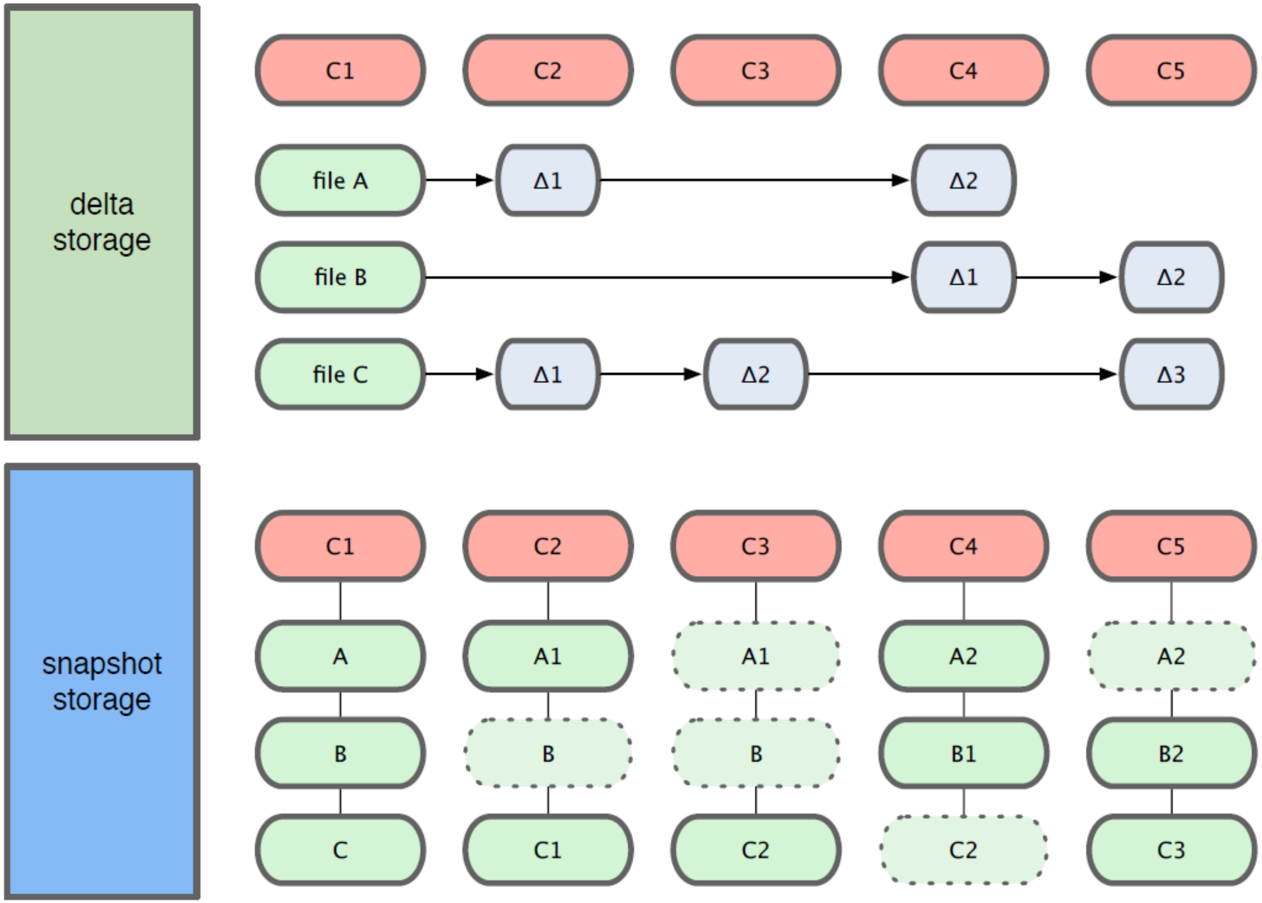
- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I’m looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

Some GIT Disadvantages

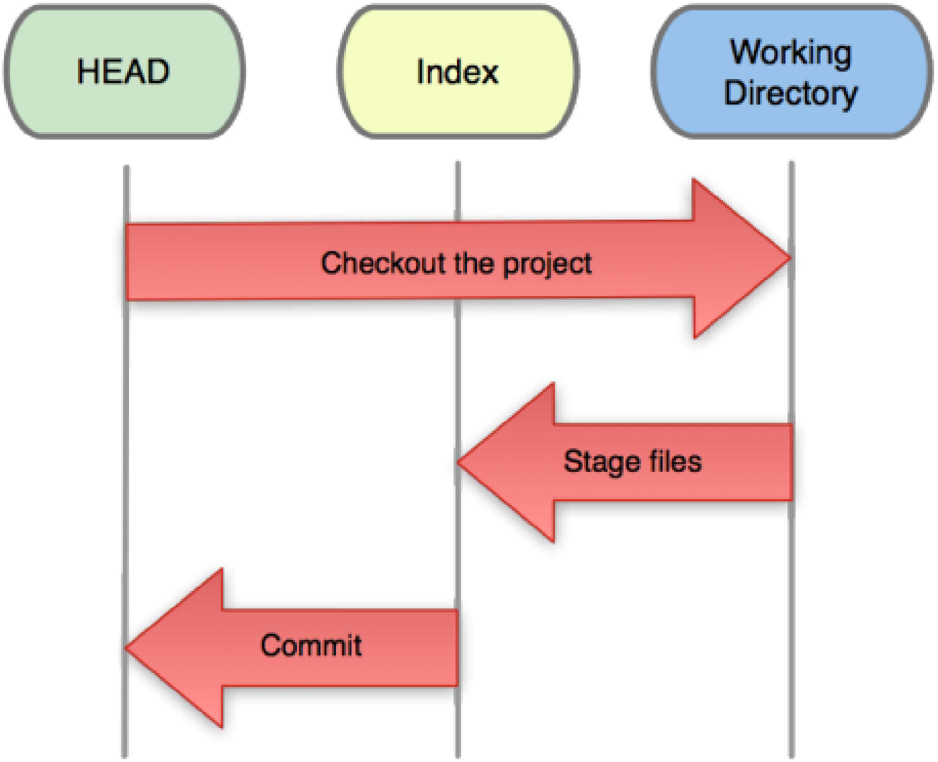
- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
    - Conceptual difference
    - Huge amount of commends

Getting Started

- Git uses snapshot storage



Getting Started



- Three trees of Git
  - The HEAD
    - last commit snapshot, next parent
  - Index
    - Proposed next commit snapshot
  - Working directory
    - Sandbox

Getting Started - Workflow

- A basic workflow
  - (Possible init or clone) Init a repo
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

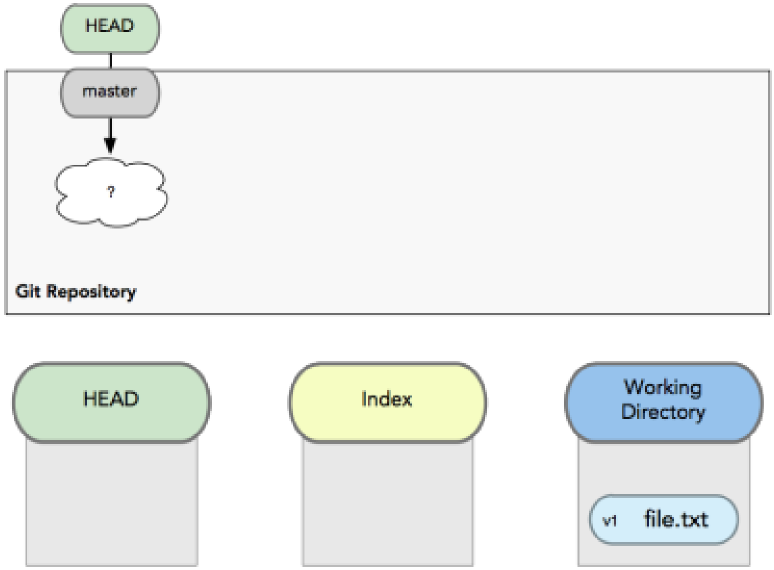
Getting Started – initialise a new repository

- Init a repository
- Git init

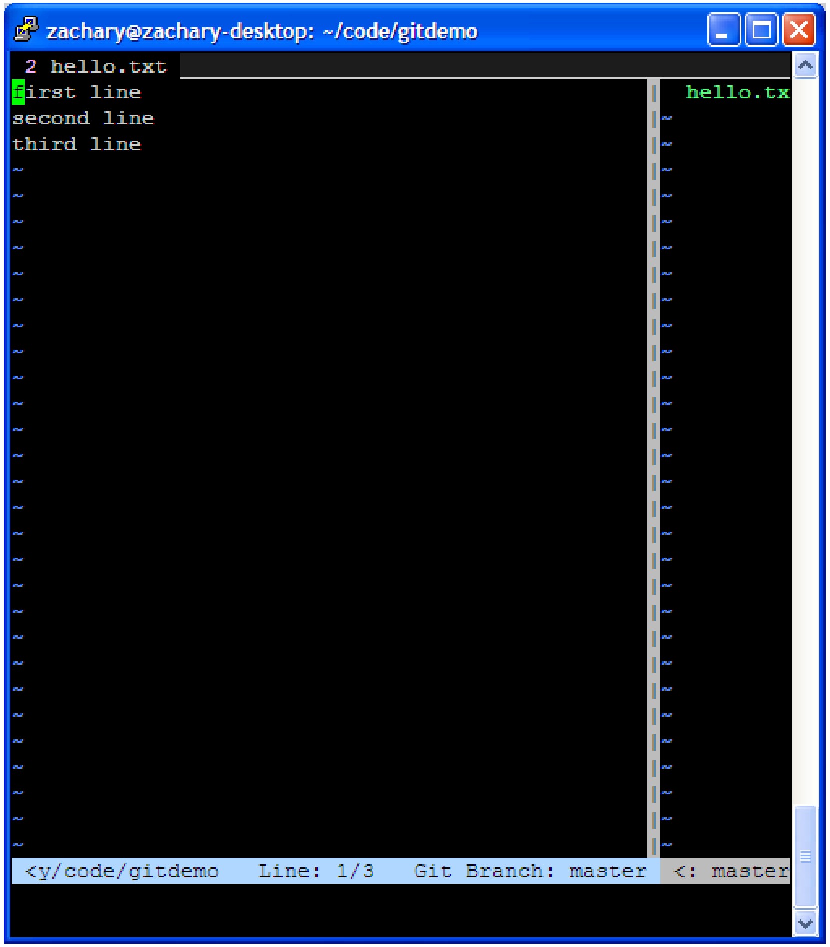
```
zachary@zachary-desktop:~/code/gitdemo$ git init
Initialized empty Git repository in /home/zachary/code/gitdemo/.git/


zachary@zachary-desktop:~/code/gitdemo$ ls -l .git/
total 32
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 branches
-rw-r--r-- 1 zachary zachary   92 2011-08-28 14:51 config
-rw-r--r-- 1 zachary zachary   73 2011-08-28 14:51 description
-rw-r--r-- 1 zachary zachary   23 2011-08-28 14:51 HEAD
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 hooks
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 info
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 objects
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 refs
```

Getting Started – do some work on files

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

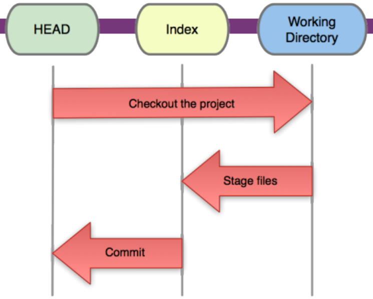


- Use your favorite editor


zachary@zachary-desktop: ~/code/gitdemo

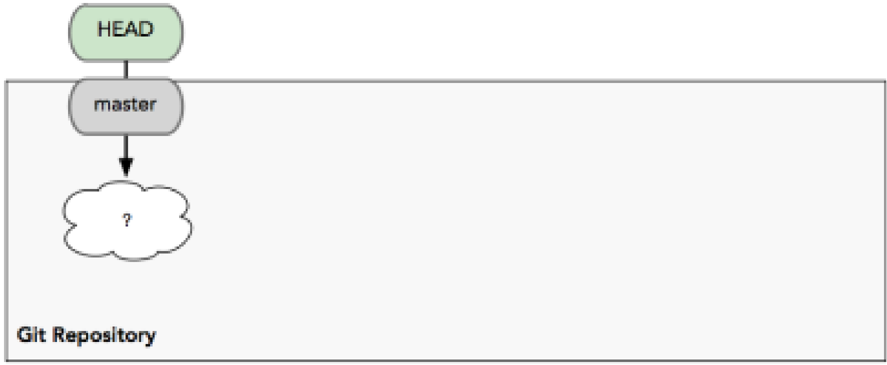
```
 2 hello.txt
first line                                           | hello.tx
second line                                          |~
third line                                           |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
~                                                    |~
<y/code/gitdemo   Line: 1/3   Git Branch: master      <: master
```

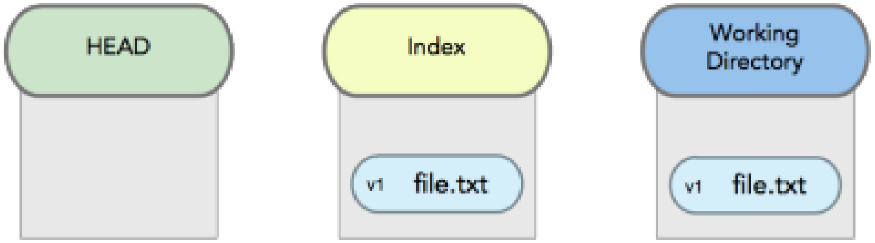
Getting Started – tell Git about it



- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git add filename

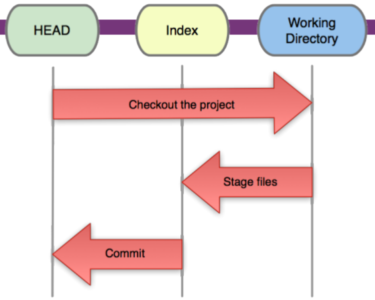




git add

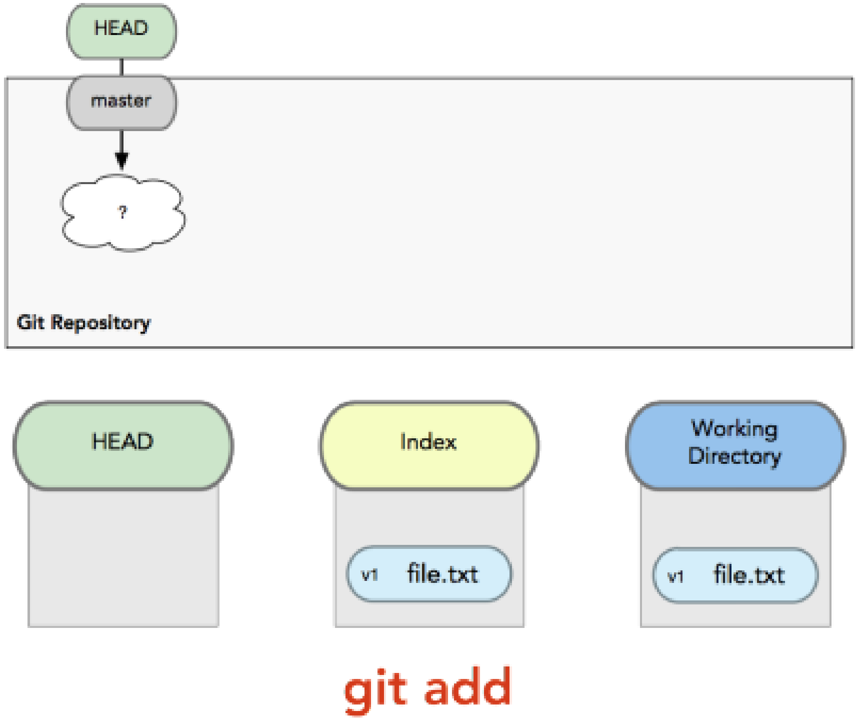
```
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Getting Started – add and check

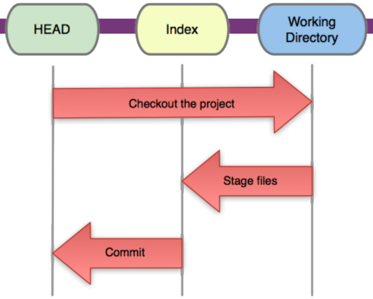


- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git status



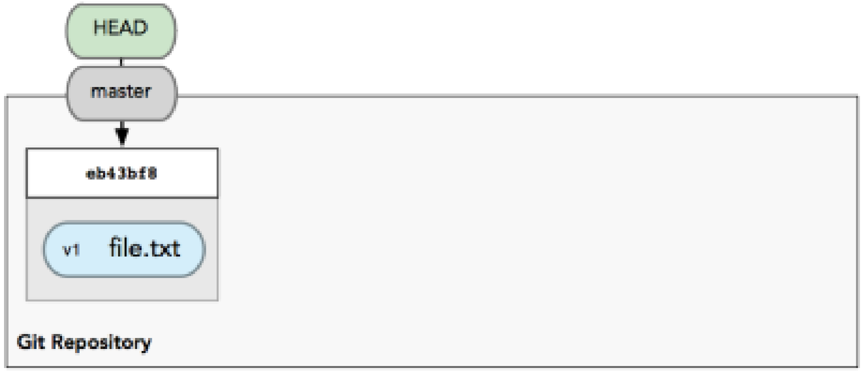
```
zachary@zachary-desktop:~/code/gitdemo$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.txt
#
```



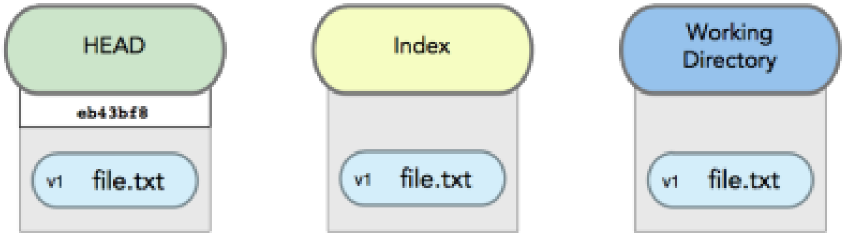
Getting Started – commit to repo

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git commit



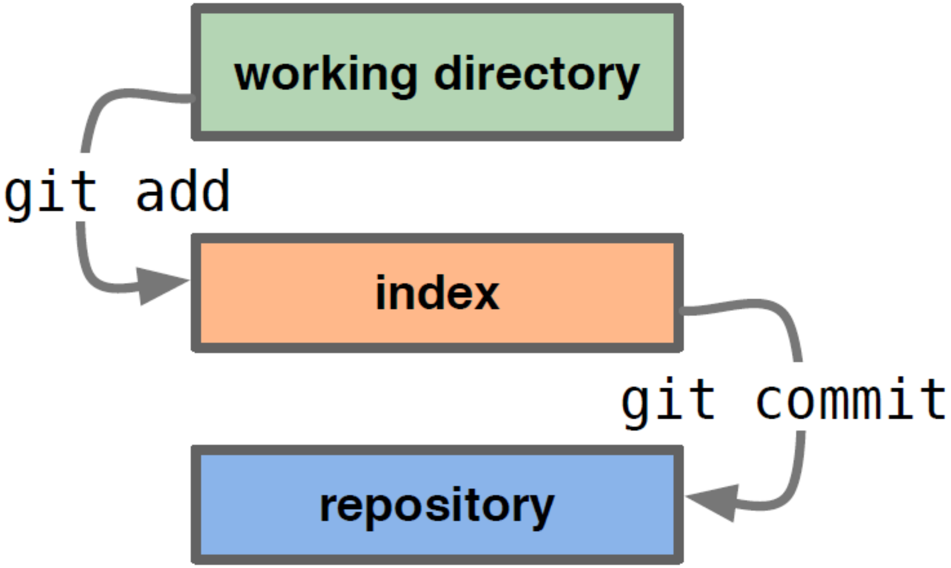
```
# Please enter the commit message for your changes.
# Lines starting with '#' will be ignored, and an
# empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.txt
#
```



git commit

Getting files into the repository...

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



Checking changes and history

- View changes
- Git diff
  - Show the difference between working directory and staged
- Git diff --cached
  - Show the difference between staged and the HEAD

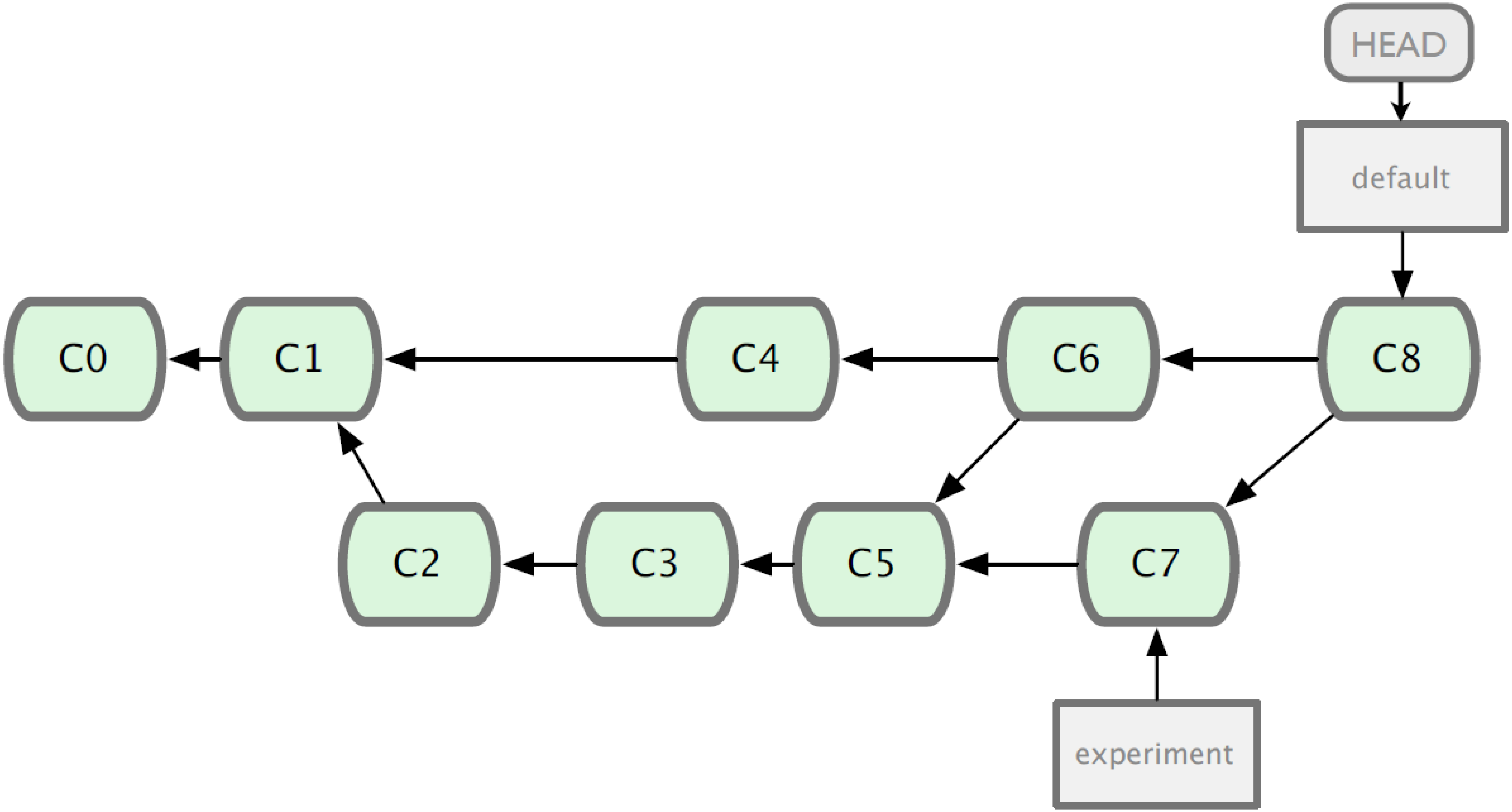
- View history
- Git log

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
```

Using the backups

- Revert changes (Get back to a previous version)
  - Git checkout <commit_hash>

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
zachary@zachary-desktop:~/code/gitdemo$ git checkout 4539
Note: checking out '4539'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 4539141... Initial commit
```



git merge experiment

Working with remote repository

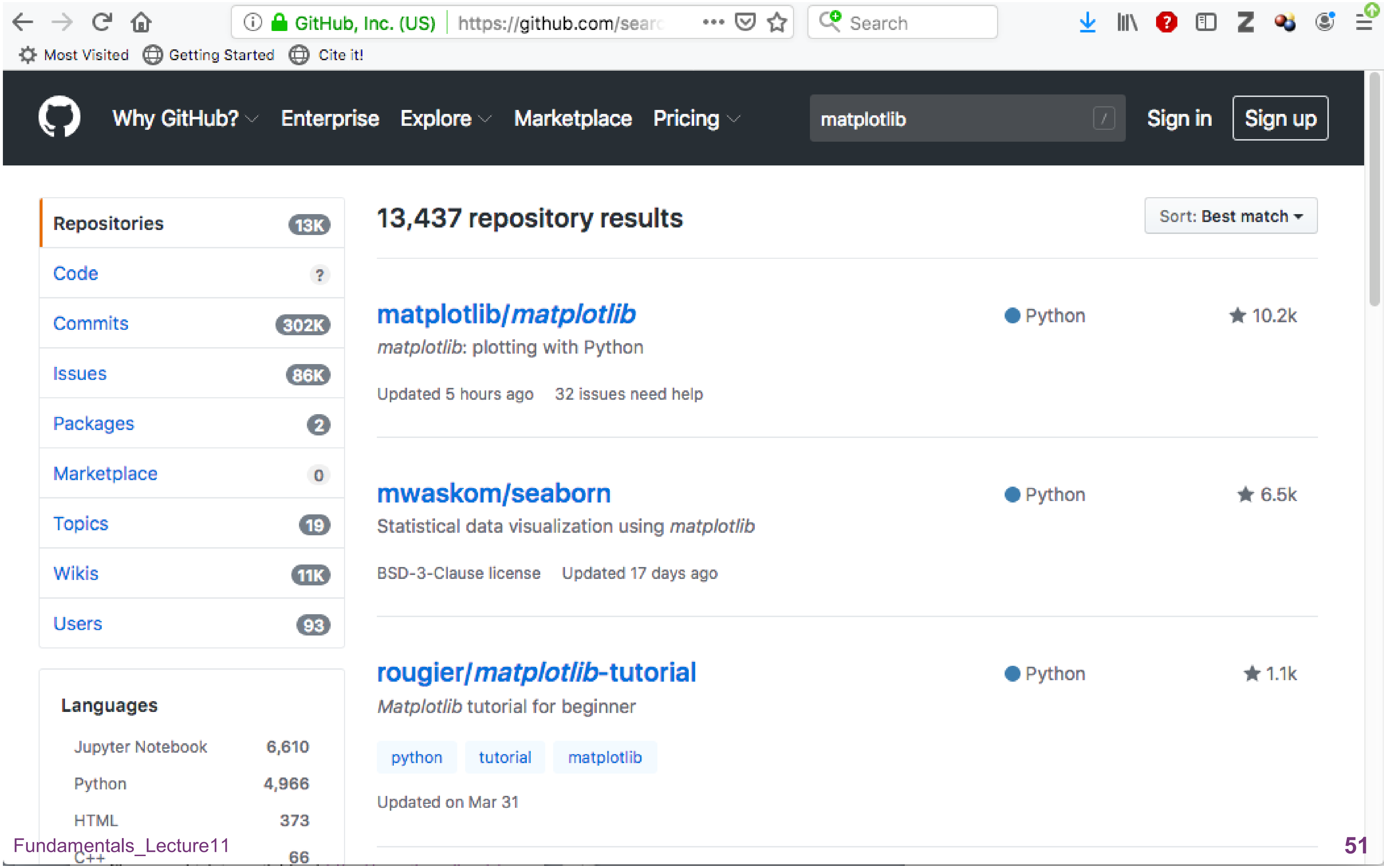
- Use git clone to replicate repository

- Get changes with
  - git fetch
  - git pull (fetches and merges)

- Propagate changes with
  - git push

- Protocols
  - Local filesystem (file://)
  - SSH (ssh://)
  - HTTP (http:// https://)
  - Git protocol (git://)

Github – online, collaborative CVS



Repositories	13K
Code	?
Commits	302K
Issues	86K
Packages	2
Marketplace	0
Topics	19
Wikis	11K
Users	93

Languages

Jupyter Notebook	6,610
Python	4,966
HTML	373
C++	66

13,437 repository results

matplotlib/*matplotlib* ● Python ★ 10.2k

matplotlib: plotting with Python

Updated 5 hours ago 32 issues need help

mwaskom/seaborn ● Python ★ 6.5k

Statistical data visualization using *matplotlib*

BSD-3-Clause license Updated 17 days ago

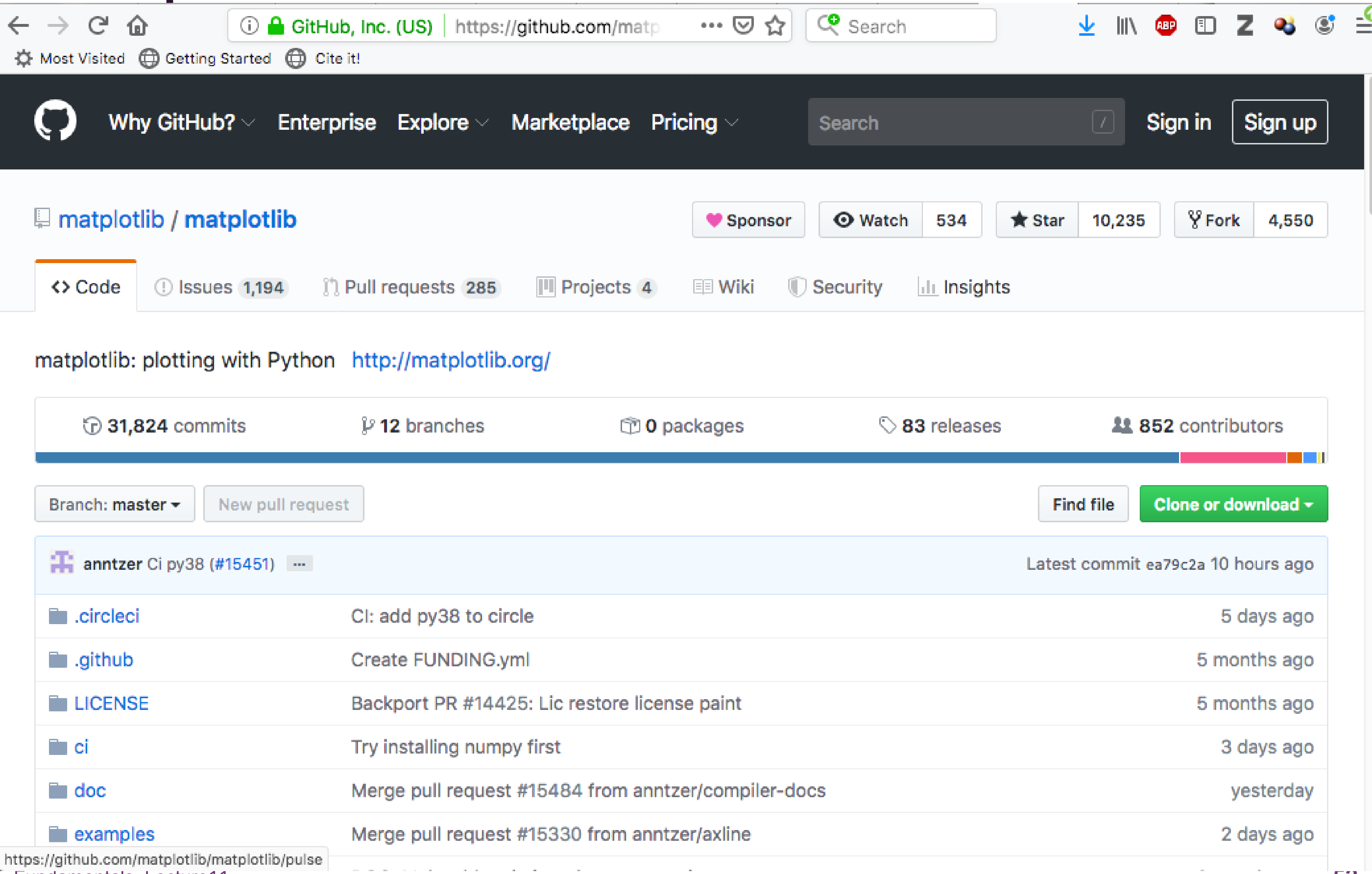
rougier/*matplotlib*-tutorial ● Python ★ 1.1k

Matplotlib tutorial for beginner

python tutorial matplotlib

Updated on Mar 31

Matplotlib on Github



matplotlib / matplotlib

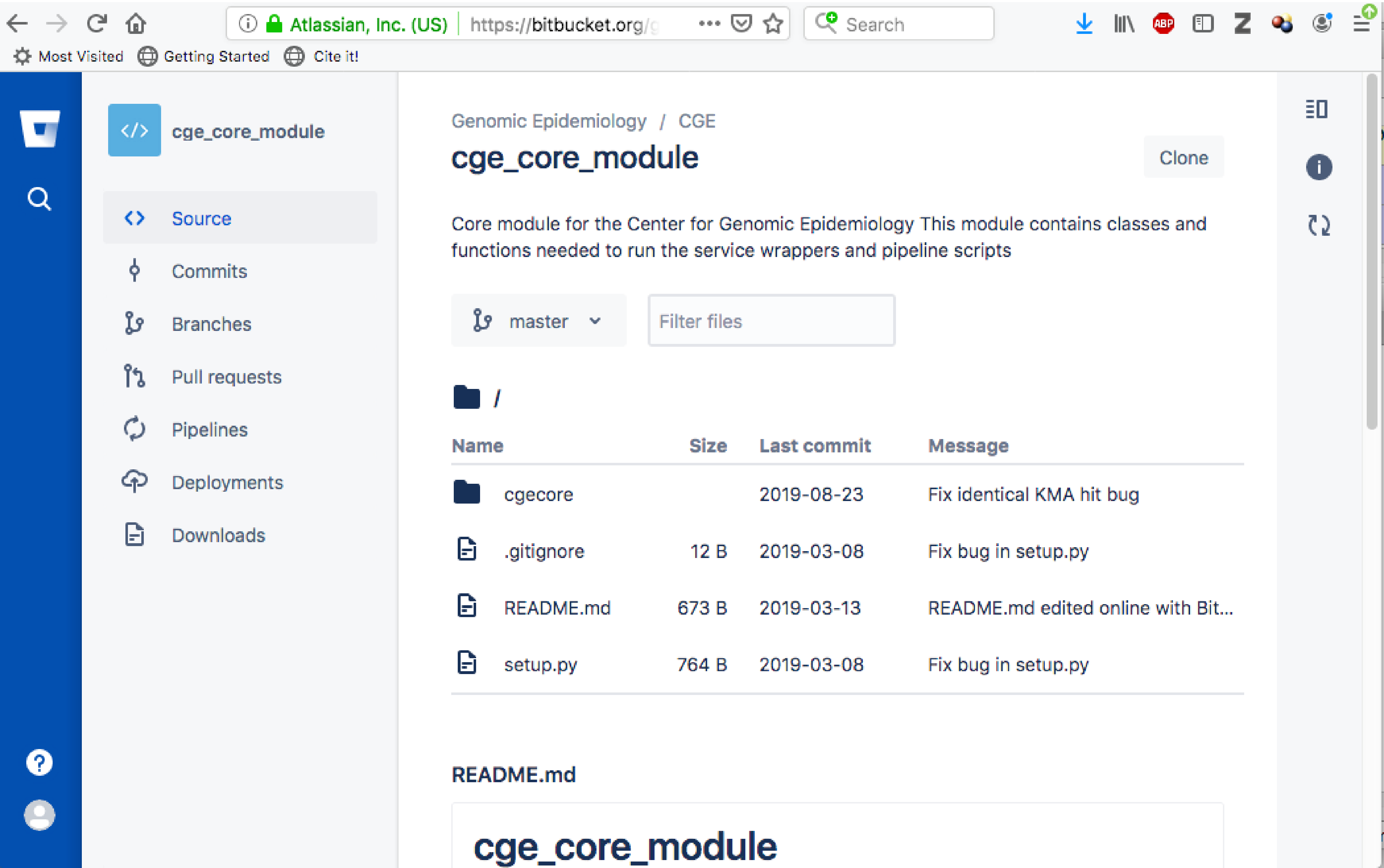
matplotlib: plotting with Python http://matplotlib.org/

31,824 commits | 12 branches | 0 packages | 83 releases | 852 contributors

anntzer Ci py38 (#15451) ... Latest commit ea79c2a 10 hours ago

.circleci	CI: add py38 to circle	5 days ago
.github	Create FUNDING.yml	5 months ago
LICENSE	Backport PR #14425: Lic restore license paint	5 months ago
ci	Try installing numpy first	3 days ago
doc	Merge pull request #15484 from anntzer/compiler-docs	yesterday
examples	Merge pull request #15330 from anntzer/axline	2 days ago

Bitbucket – another VCS



Genomic Epidemiology / CGE

cge_core_module

Core module for the Center for Genomic Epidemiology This module contains classes and functions needed to run the service wrappers and pipeline scripts

/

Name	Size	Last commit	Message
cgecore		2019-08-23	Fix identical KMA hit bug
.gitignore	12 B	2019-03-08	Fix bug in setup.py
README.md	673 B	2019-03-13	README.md edited online with Bit...
setup.py	764 B	2019-03-08	Fix bug in setup.py

README.md

cge_core_module

Git References

- Software Carpentry tutorial https://swcarpentry.github.io/git-novice/
- https://betterexplained.com/articles/a-visual-guide-to-version-control/
- Some of the slides are adopted from “Introduction to Git” available at http://innovationontherun.com/presentation-files/Introduction%20To%20GIT.ppt
- Some of the figure are adopted from Pro GIT by Chacon, which is available at http://progit.org/book/
- Some of the slides are adopted from “Git 101” available at http://assets.en.oreilly.com/1/event/45/Git%20101%20Tutorial%20Presentation.pdf

PACKAGES

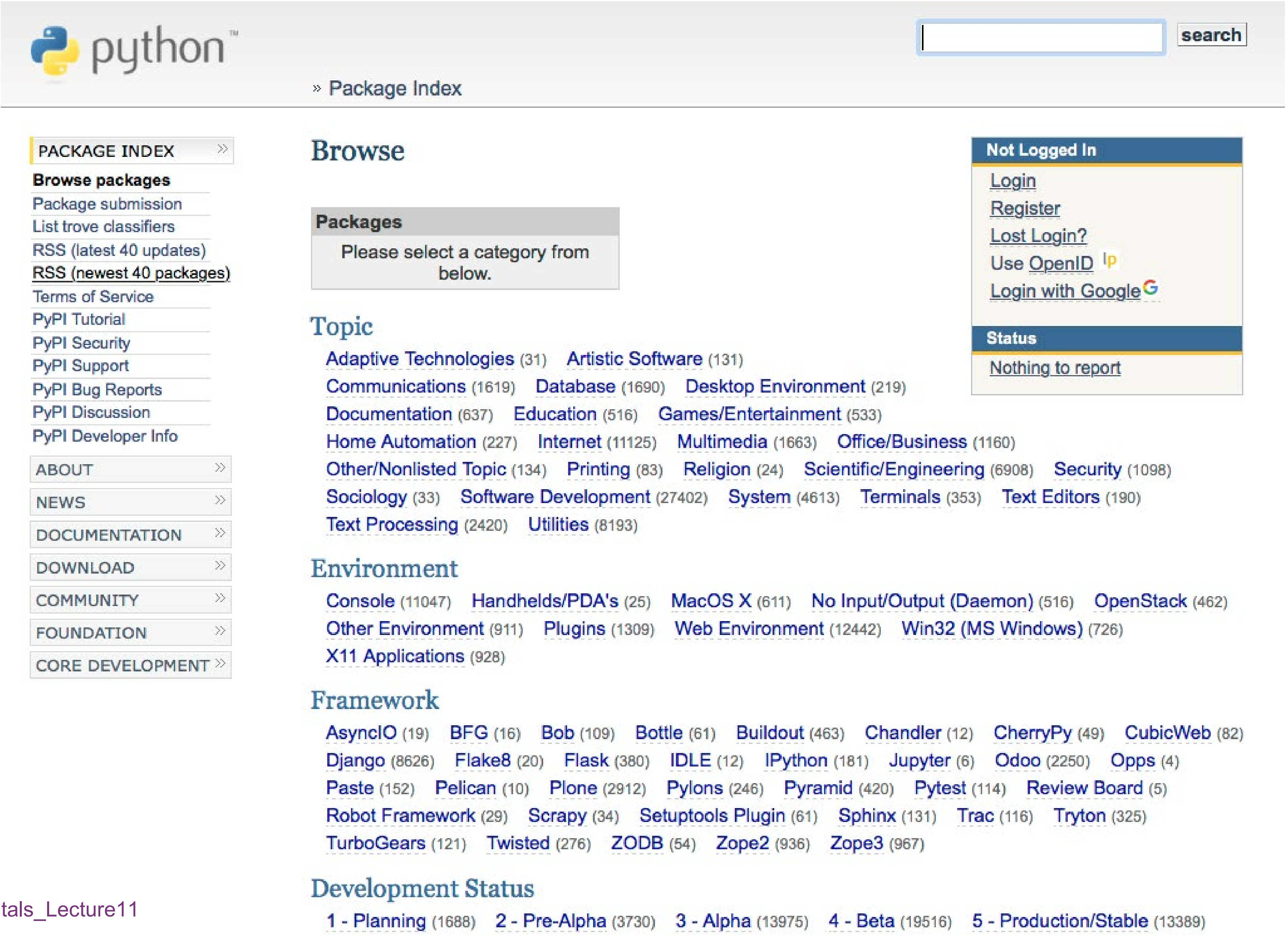
Fundamentals of Programming

Lecture 11

Packages

- Across the semester we've used packages:
  - numpy, scipy, matplotlib, pandas, random, seaborn, bokeh
- We looked at PyPI to see some of the packages that were available
- We've made our own modules
- Now to take the next step – writing a package!

PyPI - https://pypi.python.org/



Browse

Packages

Please select a category from below.

Topic

Adaptive Technologies (31) Artistic Software (131) Communications (1619) Database (1690) Desktop Environment (219) Documentation (637) Education (516) Games/Entertainment (533) Home Automation (227) Internet (11125) Multimedia (1663) Office/Business (1160) Other/Nonlisted Topic (134) Printing (83) Religion (24) Scientific/Engineering (6908) Security (1098) Sociology (33) Software Development (27402) System (4613) Terminals (353) Text Editors (190) Text Processing (2420) Utilities (8193)

Environment

Console (11047) Handhelds/PDA's (25) MacOS X (611) No Input/Output (Daemon) (516) OpenStack (462) Other Environment (911) Plugins (1309) Web Environment (12442) Win32 (MS Windows) (726) X11 Applications (928)

Framework

AsyncIO (19) BFG (16) Bob (109) Bottle (61) Buildout (463) Chandler (12) CherryPy (49) CubicWeb (82) Django (8626) Flake8 (20) Flask (380) IDLE (12) IPython (181) Jupyter (6) Odoo (2250) Opps (4) Paste (152) Pelican (10) Plone (2912) Pylons (246) Pyramid (420) Pytest (114) Review Board (5) Robot Framework (29) Scrapy (34) Setuptools Plugin (61) Sphinx (131) Trac (116) Tryton (325) TurboGears (121) Twisted (276) ZODB (54) Zope2 (936) Zope3 (967)

Development Status

1 - Planning (1688) 2 - Pre-Alpha (3730) 3 - Alpha (13975) 4 - Beta (19516) 5 - Production/Stable (13389)

Guidelines

- Scott Torborg wrote release 0.1 of the python-packaging Documentation
- We'll use it to structure our approach…

PEPs and guides

- We know about the PEP8 style guide
- There are many others to consider before sending your code out into the wild
- e.g. PEP257 – docstring conventions:
  - All modules should normally have docstrings, and all functions and classes exported by a module should also have docstrings.
  - Public methods (including the __init__ constructor) should also have docstrings.
  - A package may be documented in the module docstring of the __init__.py file in the package directory.

Docstring example

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

More detail:
https://www.python.org/dev/peps/pep-0257/

Package guidelines

- Packages should make it easy:
  - To install with pip or easy_install
  - To specify as a dependency for another package
  - For other users to download and run tests
  - For other users to work on and have immediate familiary with the basic directory structure
  - To add and distribute documentation

Picking a Name

- Python module/package names should generally follow the following constraints:
  - All lowercase
  - Unique on pypi, even if you don’t want to make your package publicly available (you might want to specify it privately as a dependency later)
  - Underscore-separated or no word separators at all (don’t use hyphens)
  - We’ve decided to turn our bit of code into a module called **funniest**

The Code

- We’ll start with some Python code to package up
- Native German speakers, please proceed with caution:

```
def joke():
    return (u'Wenn ist das Nunst\u00fcck’ +
             ‘git und Slotermeyer? Ja! ... '
            u'Beiherhund das Oder die’ +
             ‘Flipperwaldt gersput.')
```

Creating the Scaffolding

- The initial directory structure for funniest should look like this:

```
funniest/
    funniest/
        __init__.py
    setup.py
```

- The top level directory is the root of our SCM repo, e.g. funniest.git.
- The subdir, also called funniest, is the actual Python module.

- For starters we’ll put the joke() function in __init__.py, so it just contains:

```
def joke():
    return (u'Wenn ist das Nunst\u00fcck git und Slotermeyer? Ja! ... '
            u'Beiherhund das Oder die Flipperwaldt gersput.')
```

Setup.py

- The main setup config file, setup.py, should contain a single call to setuptools.setup(), including the values for the project metadata:

```
from setuptools import setup

setup(name     ='funniest',
      version  ='0.1',
      description='The funniest joke in the world',
      url      ='http://github.com/storborg/funniest',
      author   ='Flying Circus',
      author_email='flyingcircus@example.com',
      license  ='MIT',
      packages =['funniest'],
      zip_safe =False)
```

Installing the package

- Now we can install the package locally (for use on our system), with:

```
$ pip install .
```

- Anywhere else in our system using the same version/location of Python, we can do this now:

```
>>> import funniest
>>> print funniest.joke()
```

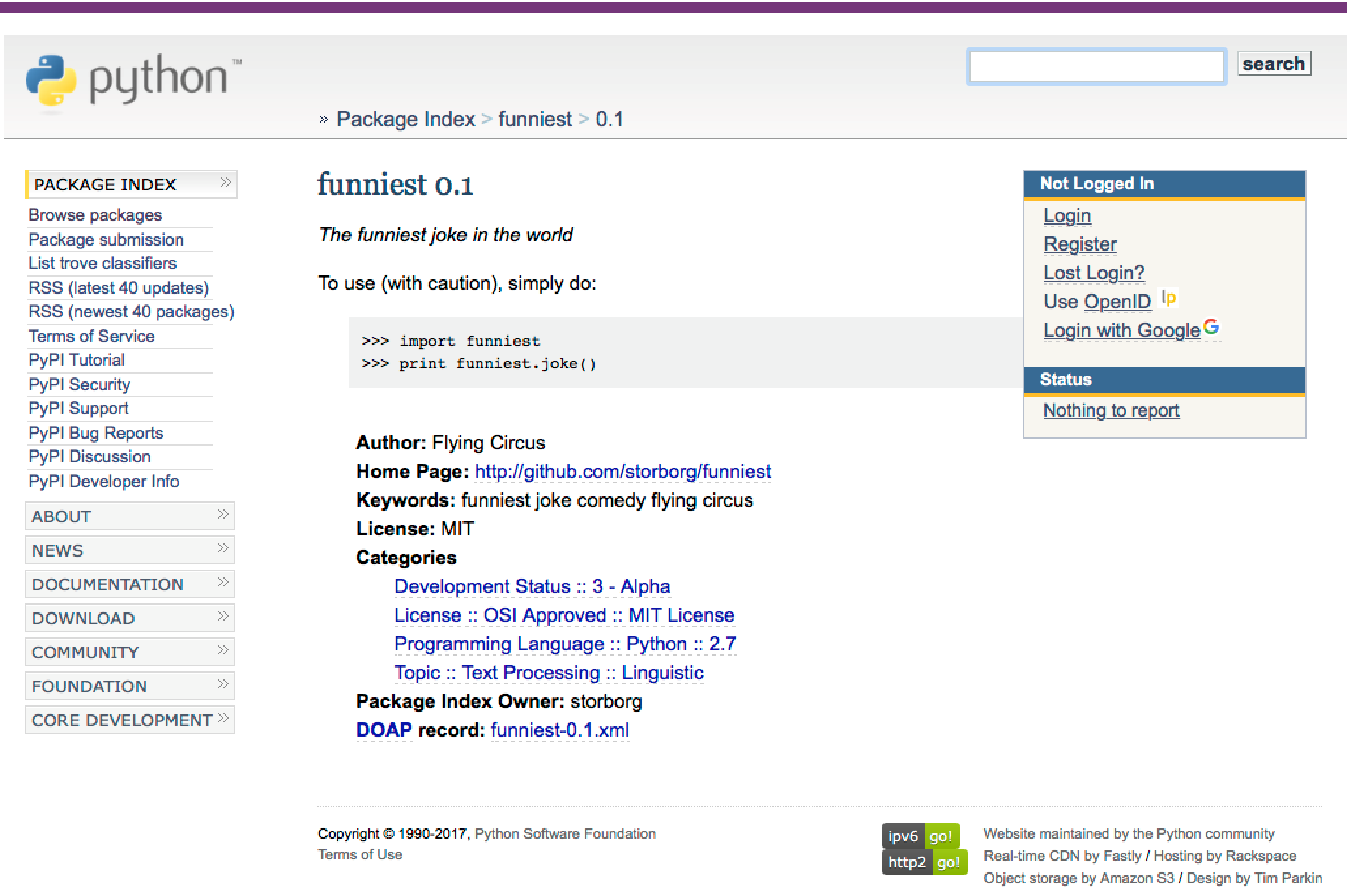
Publishing On PyPI

- The **setup.py** script is our main entrypoint to register the package name on PyPI and upload source distributions.

- To “register” the package (this will reserve the name, upload package metadata, and create the pypi.python.org webpage):

  - `$ python setup.py register`

  - If you haven’t published things on PyPI before, you’ll need to create an account by following the steps provided at this point.



funniest 0.1

The funniest joke in the world

To use (with caution), simply do:

```
>>> import funniest
>>> print funniest.joke()
```

Author: Flying Circus
Home Page: http://github.com/storborg/funniest
Keywords: funniest joke comedy flying circus
License: MIT
Categories

- Development Status :: 3 - Alpha
- License :: OSI Approved :: MIT License
- Programming Language :: Python :: 2.7
- Topic :: Text Processing :: Linguistic

Package Index Owner: storborg
DOAP record: funniest-0.1.xml

Website maintained by the Python community
Real-time CDN by Fastly / Hosting by Rackspace
Object storage by Amazon S3 / Design by Tim Parkin

Installing the Package

- At this point, other consumers of this package can install the package with pip:

```
$ pip install funniest
```

- They can specify it as a dependency for another package, and it will be automatically installed when that package is installed

Ignoring Files (.gitignore, etc)

- When we upload the project to PyPI, we may want to exclude some files the we used during development.
- The Python build system creates a number of intermediary files we'll want to be careful to not commit to source control.

- We can use a **.gitignore** to automate this (or the equivalent for other SCM/VCS's)

```
# Compiled python modules.
*.pyc

# Setuptools distribution folder.
/dist/

# Python egg metadata, regenerated from source by setuptools
/*.egg-info
```

That's all you need…

- The structure described so far is all that's necessary to create reusable simple packages
- If every published Python tool or library used followed these rules, the world would be a better place.

- But wait, there's more!
- Most packages will want to add things like command line scripts, documentation, tests, and analysis tools
- See the full documentation for more…

- So you have all you need to be able to run and distribute a Python project

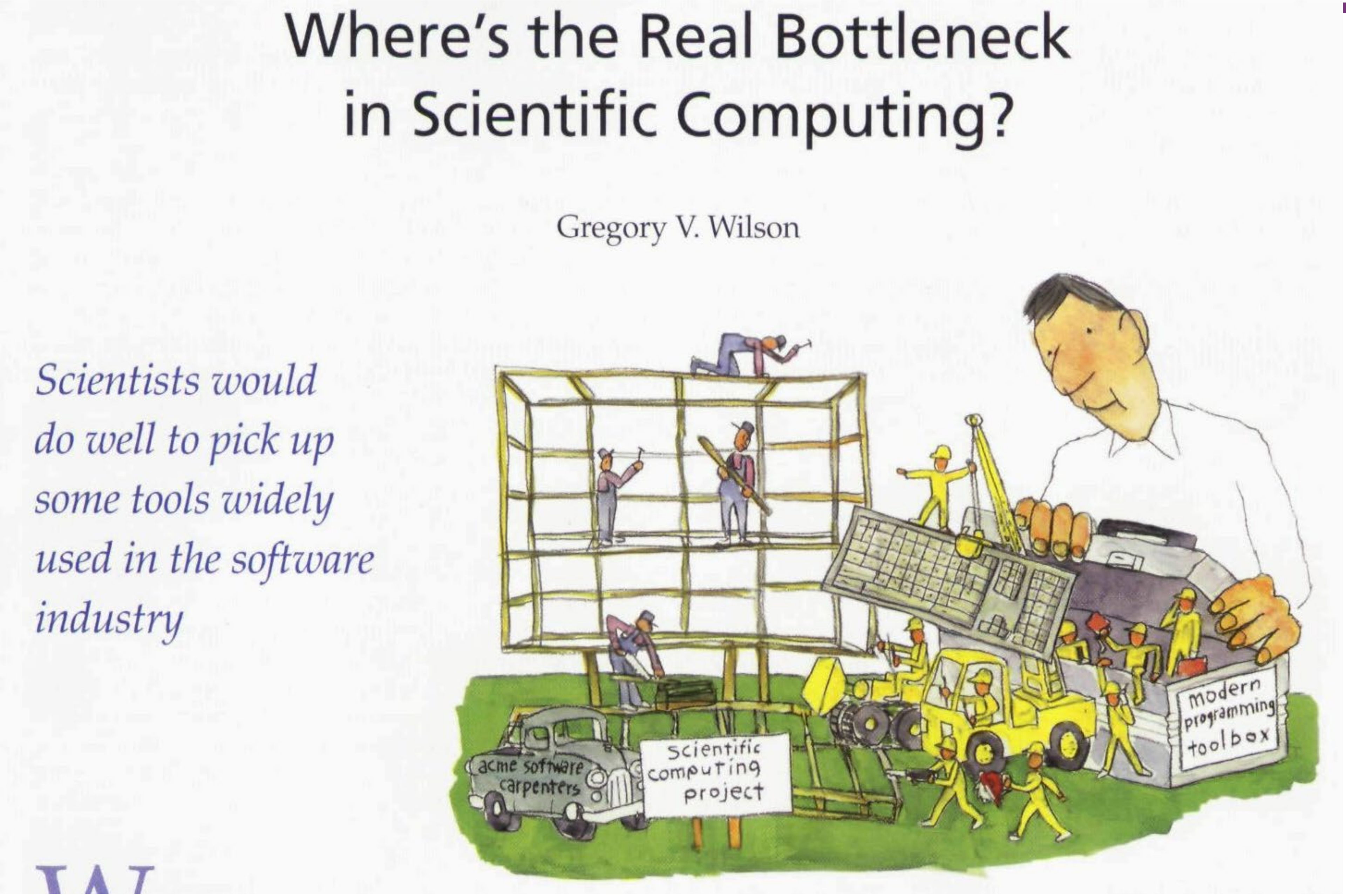
Package Risks

- Any code that you haven't written yourself presents a risk (and your code does too!)
  - Errors in the code
  - Slow or no support for updates
  - Becoming unsupported
  - Dependencies on other packages
- What to consider
  - Is it developed by an individual or community?
  - How responsive are the developers?
  - How recently has it been updated?
  - Does it depend on other packages that are neglected?

Packages and Science

- Last week we saw that **Jupyter notebooks** allow us to share our workflows and reasoning
- The **community** can then move forward together

- With **packages**, we also share the code to help make science possible
- Instead of showing a particular approach, this gives others the building blocks to do their own research
- Again, the **community** moves forward, based on the shared packages that save researchers from having to implement everything themselves
  - e.g. Matplotlib, Pandas, Numpy etc.



Where's the Real Bottleneck in Scientific Computing?

Gregory V. Wilson

Scientists would do well to pick up some tools widely used in the software industry

acme software carpenters
scientific computing project
modern programming toolbox

http://halfonlab.ccr.buffalo.edu/other_docs/Wilson_2006_American%20Scientist.pdf

AND NOW FOR SOMETHING COMPLETELY DIFFERENT

THINGS THAT MIGHT HELP YOU ON THE ASSIGNMENT

Fundamentals of Programming

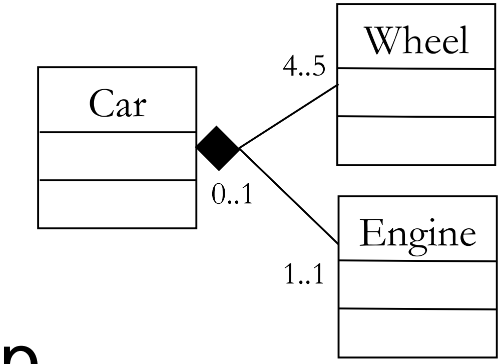
Lecture 11

CLASS RELATIONSHIPS A REVIEW

Fundamentals of Programming

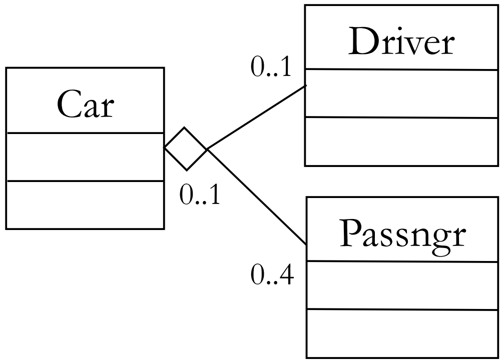
Lecture 11

Class Relationships (1)



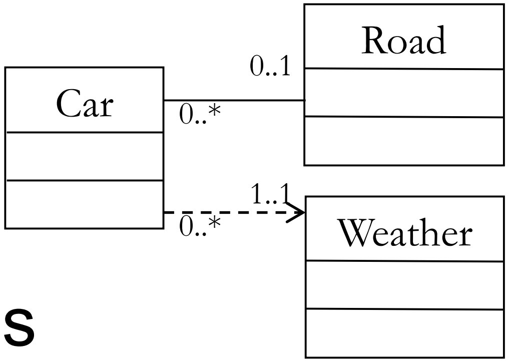
- Composition
  - “**has-a**” or “whole-part” relationship
    - UML: Shown with solid diamond beside container class
    - *e.g.*, Car “has-a” Wheel
  - Strong lifecycle dependency between classes
    - Car is not a car without four Wheels and an Engine
    - When Car is destroyed, so are the Wheels and Engine
  - In code:
    - Car would have Wheel and Engine as class fields

Class Relationships (2)



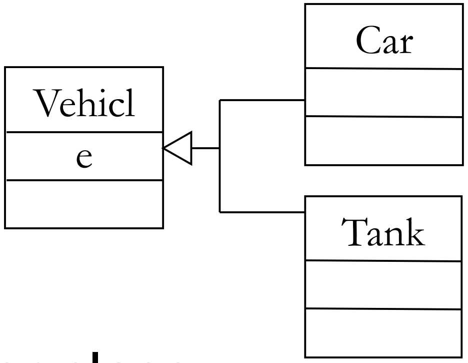
- Aggregation
  - Weaker form of composition, but is still “**has-a**”
    - UML: Shown with open/unfilled diamond beside container
  - Lifecycle dependency usually not strong
    - Car does not always have a driver
    - When Car is destroyed driver and passengers are not
    - Drivers can drive different cars
  - In code:
    - Car would have Driver and Passenger as class fields
    - …exactly like composition!

Class Relationships (3)



- Association and Dependency
  - Indicates interaction between classes
    - Association = solid line, Dependency = dashed line
    - Difference is murky: UML is a *guide*, not a *law*
  - Used to show that one class invokes methods on another
    - … but that there is no other relationship beyond this
    - With arrow, implies *unidirectional* (Car calls Weather, not vice-versa)
    - No arrow implies *bidirectional* (Car and Road call each other)
  - In code: Any way that a method call can be set up and made
    - *e.g.*, Weather object is passed as a parameter to a Car method
      - *e.g.*, Car.setAggressiveness(Weather currentConditions)
    - *e.g.*, Road has a class field of all Cars on that Road (aggregation?)

Class Relationships (4)



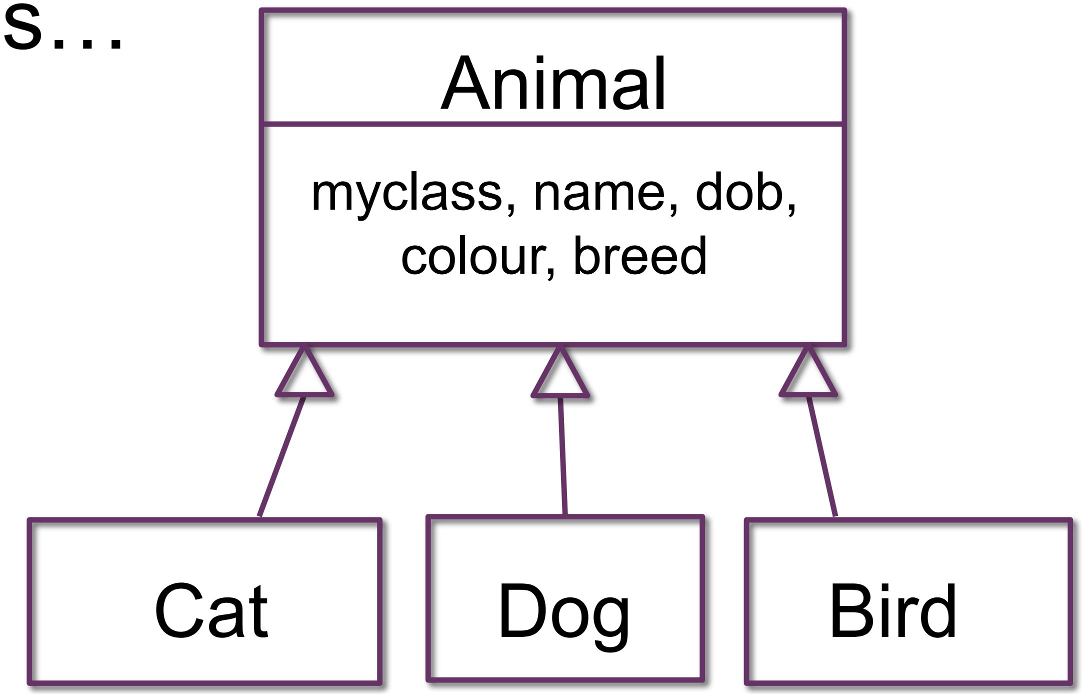
- Inheritance
  - **“is-a”** relationship
    - Indicates one class is a sub-type of another class
    - Shown with an open triangle arrowhead beside super-type
  - Implies the specialisation of the super-type
    - Super-type synonyms: ‘parent’, ‘base’
    - Sub-type synonyms: ‘child’, ‘derived’
  - In code: During class declaration; syntax is language-specific
    - Python: class Car(Vehicle):
    - Java: public class Car **extends** Vehicle
    - C++/C#: public class Car **:** Vehicle

Example: Pet Shelter (shelters.py)

```
from animals import Dog,Cat,Bird,Shelter

print('\nPet shelter program...\n')

rspca = Shelter('RSPCA', 'Serpentine Meander', '123456')
rspca.newAnimal('Dog', 'Dude', '1/1/2011', 'Brown', 'Jack Russell')
rspca.newAnimal('Dog', 'Brutus', '1/1/1982', 'Brown', 'Rhodesian Ridgeback')
rspca.newAnimal('Cat', 'Oogie', '1/1/2006', 'Grey', 'Fluffy')
rspca.newAnimal('Bird', 'Big Bird', '10/11/1969', 'Yellow', 'Canary')
rspca.newAnimal('Bird', 'Dead Parrot', '1/1/2011', 'Dead', 'Parrot')

print('\nAnimals added\n')

print('Listing animals for processing...\n')

rspca.displayProcessing()

print('Processing animals...\n')

rspca.makeAvailable('Dude')
rspca.makeAvailable('Oogie')
rspca.makeAvailable('Big Bird')
rspca.makeAdopted('Oogie')

print('\nPrinting updated list...\n')

rspca.displayAll()
```

Inheritance Example: Animals

- Repetition should be avoided if possible
- Cat, Dog and Bird are nearly identical
- Factor out the duplicated fields and methods…



Example: Pet Shelter (animals.py)

```
class Animal():

    myclass = "Animal"         # class variable myclass

    def __init__(self, name, dob, colour, breed):
        self.name = name       # instance variable name
        self.dob = dob         # instance variable dob
        self.colour = colour   # instance variable colour
        self.breed = breed     # instance variable breed

    def __str__(self):
        return(self.name + '|' + self.dob + '|' + self.colour + '|' + self.breed)

    def printit(self):
        spacing = 5 - len(self.myclass)
        print(self.myclass.upper(), spacing*' ' + ': ', self.name,'\tDOB: ',
              self.dob,'\tColour: ', self.colour,'\tBreed: ', self.breed)

class Dog(Animal):
    myclass = "Dog"

class Cat(Animal):
    myclass = "Cat"

class Bird(Animal):
    myclass = "Bird"
```

__str__() connects to the str() method to easily print a representation of an object

Assignment Classes (1/2)

- The supplied code has a shrimp class with a “state” instance variable
  - Egg → hatchling → juvenile → adult
  - At all stages, there’s a probability of death
  - As an adult, there’s a probability (+ perhaps a “collision”) for reproduction
- This can be implemented as:

You may have more stages in your lifecycle

```
if self.state == “egg”:
     <do something>
elif self.state == “hatchling”:
    <do something else>     # as in given code
```

Assignment Classes (2/2)

- Alternatively “state” could be modelled as subclasses of a “Shrimp” superclass
- This is a more advanced approach, as the egg object has to replace itself with a hatchling
  - Egg → hatchling → juvenile → adult

You may have more stages in your lifecycle

- Position, age etc have to be transferred to the new object
- This is more complicated and **not required** for the assignment
- It could be implemented as:

```
for thisShrimp in shrimps:
    <do something>
    thisShrimp = thisShrimp.stepChange() # returns itself or the
    <do something else>                  # new object
```

Exceptions

- Python only lets objects of type Exception or it’s descendants to be thrown
- Python has a range of classes descending (inheriting, extends) from Exception
  - eg: ValueError, ZeroDivisionError
- You can define your own exception class, as long as it inherits from Exception (or one of it’s subclasses)

Exception Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
   +-- StopIteration
   +-- ArithmeticError
   | +-- FloatingPointError
   | +-- OverflowError
   | +-- ZeroDivisionError
   +-- AssertionError
   +-- AttributeError
   +-- BufferError
   +-- EOFError
   +-- ImportError
   +-- LookupError
   | +-- IndexError
   | +-- KeyError
   +-- MemoryError
   +-- NameError
   | +-- UnboundLocalError
   +-- OSError
   | +-- FileExistsError
   | +-- FileNotFoundError
   | +-- InterruptedError
   | +-- IsADirectoryError
   | +-- NotADirectoryError
   | +-- PermissionError
   | +-- ProcessLookupError
   | +-- TimeoutError
   +-- ReferenceError
   +-- RuntimeError
   | +-- NotImplementedError
   +-- SyntaxError
   | +-- IndentationError
   | +-- TabError
   +-- SystemError
   +-- TypeError
   +-- ValueError
   | +-- UnicodeError
   | +-- UnicodeDecodeError
   | +-- UnicodeEncodeError
   | +-- UnicodeTranslateError
   +-- Warning
   +-- DeprecationWarning
   +-- PendingDeprecationWarning
   +-- RuntimeWarning
   +-- SyntaxWarning
   +-- UserWarning
   +-- FutureWarning
   +-- ImportWarning
   +-- UnicodeWarning
   +-- BytesWarning
   +-- ResourceWarning
```

https://docs.python.org/3.3/library/exceptions.html

Exceptions - Where and When

- To make your code robust, put it anywhere your code could crash
  - File IO
  - User input
- An exception being thrown doesn’t have to mean there’s an “error”
- It’s the best way for objects to tell the program they’re not happy
  - E.g. raise an exception if a method is called with an invalid value (withdraw when no $$ in account)

COOL STUFF (4)

AGENT-BASED MODELS

Fundamentals of Programming

Was in Lecture 12

Agent-based models

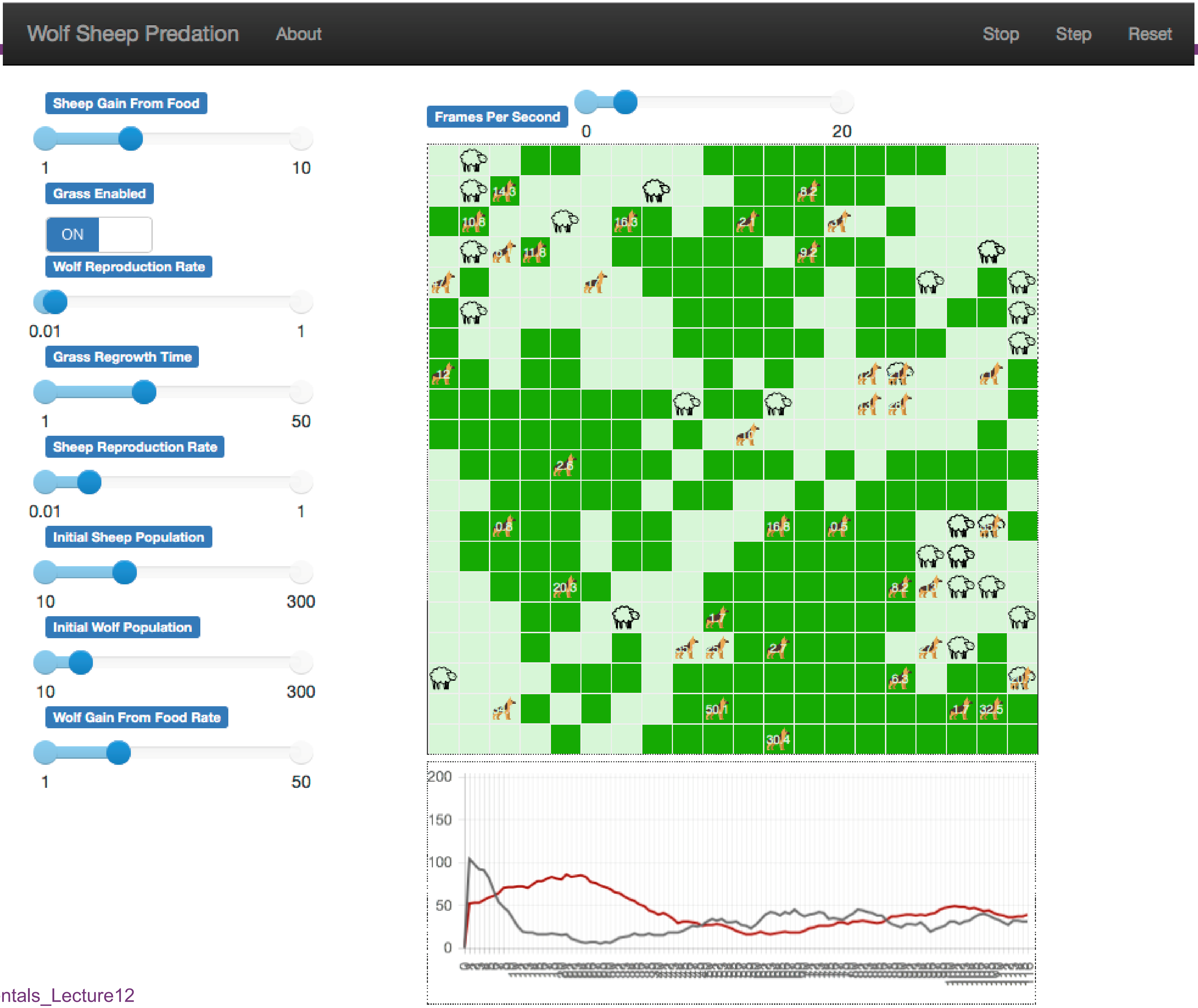
- ABM combines simulation and object-oriented models to simulate the behaviour of autonomous objects over time
- A simple behavioural model can generate complex results
- Each simulation will give different results as there is a random factor in the behaviour/position/environment for each agent

http://www.agent-based-models.com/blog/2010/03/30/agent-based-modeling/

Wolf-sheep simulation

- Two types of agents – wolves and sheep
- Randomly place an initial population in a grid
- Environment has grassy areas that the sheep eat
- Wolves eat the sheep
- Wolves and sheep reproduce at a given rate
- Grass regrows at a rate

https://github.com/projectmesa/mesa/tree/master/examples/wolf_sheep/wolf_sheep



Sheep Gain From Food
1 10

Grass Enabled
ON

Wolf Reproduction Rate
0.01 1

Grass Regrowth Time
1 50

Sheep Reproduction Rate
0.01 1

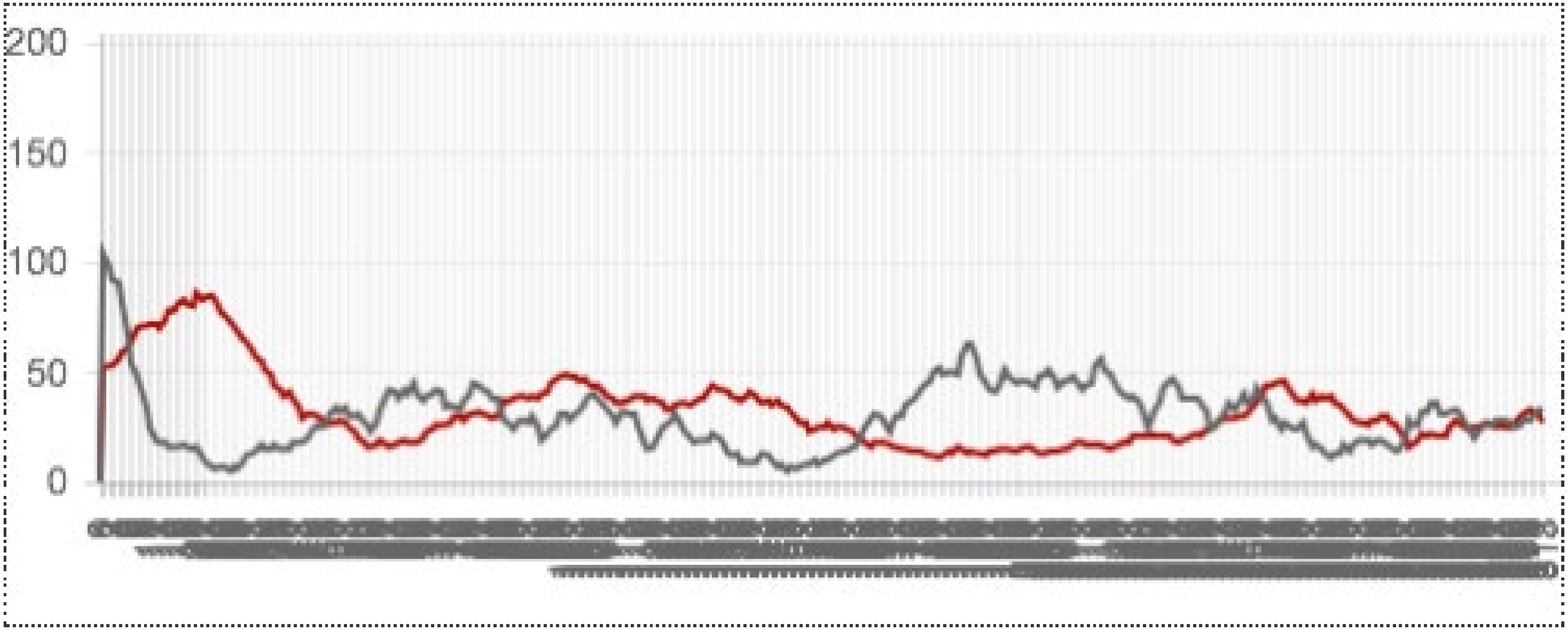
Initial Sheep Population
10 300

Initial Wolf Population
10 300

Wolf Gain From Food Rate
1 50

Frames Per Second
0 20

Wolf_sheep over time



Random walker (1/2)

```
class RandomWalker(Agent):
    '''
    Class implementing random walker methods in a generalized manner.
    Not indended to be used on its own, but to inherit its methods to multiple
    other agents.
    '''

    grid = None
    x = None
    y = None
    moore = True

    def __init__(self, pos, model, moore=True):
        '''
        grid: The MultiGrid object in which the agent lives.
        x: The agent's current x coordinate
        y: The agent's current y coordinate
        moore: If True, may move in all 8 directions. Else, N/S/E/W.
        '''
        super().__init__(pos, model)
        self.pos = pos
        self.moore = moore
```

Random walker (2/2)

```
def random_move(self):
     '''
     Step one cell in any allowable direction.
     '''

     # Pick the next cell from the adjacent cells.
     next_moves = self.model.grid.get_neighborhood(self.pos,
                   self.moore, True)
     next_move = random.choice(next_moves)

     # Now move:
     self.model.grid.move_agent(self, next_move)
```

Wolf (1/3)

This the the class definition for the wolf...

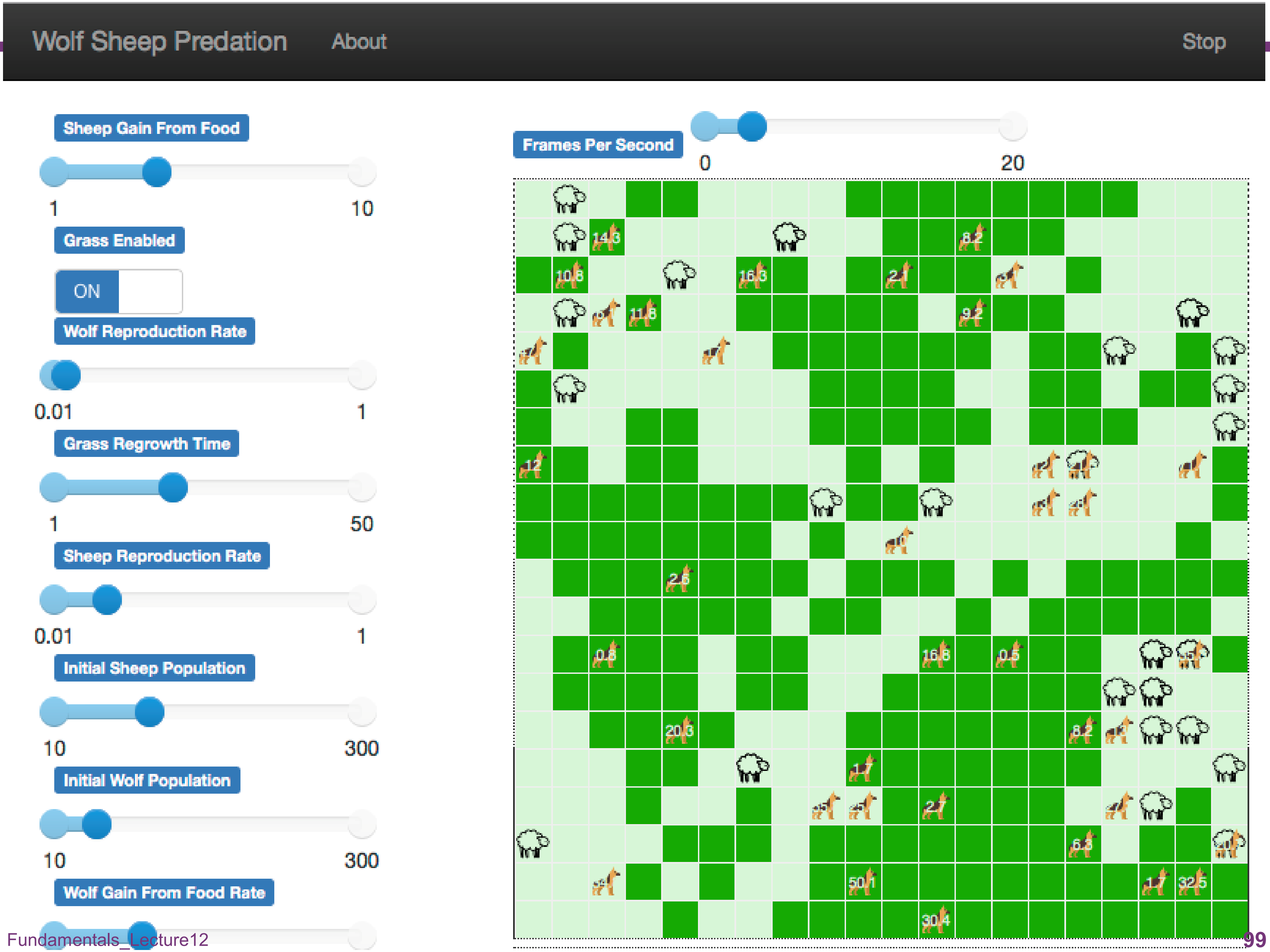
```
class Wolf(RandomWalker):
    '''
    A wolf that walks around, reproduces
    (asexually) and eats sheep.
    '''

    energy = None

    def __init__(self, pos, model, moore, energy=None):
        super().__init__(pos, model, moore=moore)
        self.energy = energy
```

Wolf (2/3)

```
def step(self):
        self.random_move()
        self.energy -= 1

        # If there are sheep present, eat one
        x, y = self.pos
        this_cell = self.model.grid.get_cell_list_contents([self.pos])
        sheep = [obj for obj in this_cell if isinstance(obj, Sheep)]

        if len(sheep) > 0:
            sheep_to_eat = random.choice(sheep)
            self.energy += self.model.wolf_gain_from_food

            # Kill the sheep
            self.model.grid._remove_agent(self.pos, sheep_to_eat)
            self.model.schedule.remove(sheep_to_eat)
```

Wolf (3/3)

```
# Death or reproduction
    if self.energy < 0:
        self.model.grid._remove_agent(self.pos, self)
        self.model.schedule.remove(self)
    else:
        if random.random() < self.model.wolf_reproduce:
            # Create a new wolf cub
            self.energy /= 2
            cub = Wolf(self.pos, self.model,
                       self.moore, self.energy)

            self.model.grid.place_agent(cub, cub.pos)
            self.model.schedule.add(cub)
```



Wolf Sheep Predation

About

Stop

Sheep Gain From Food

1 10

Grass Enabled

ON

Wolf Reproduction Rate

0.01 1

Grass Regrowth Time

1 50

Sheep Reproduction Rate

0.01 1

Initial Sheep Population

10 300

Initial Wolf Population

10 300

Wolf Gain From Food Rate

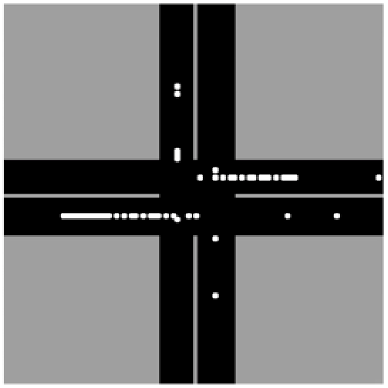
Frames Per Second

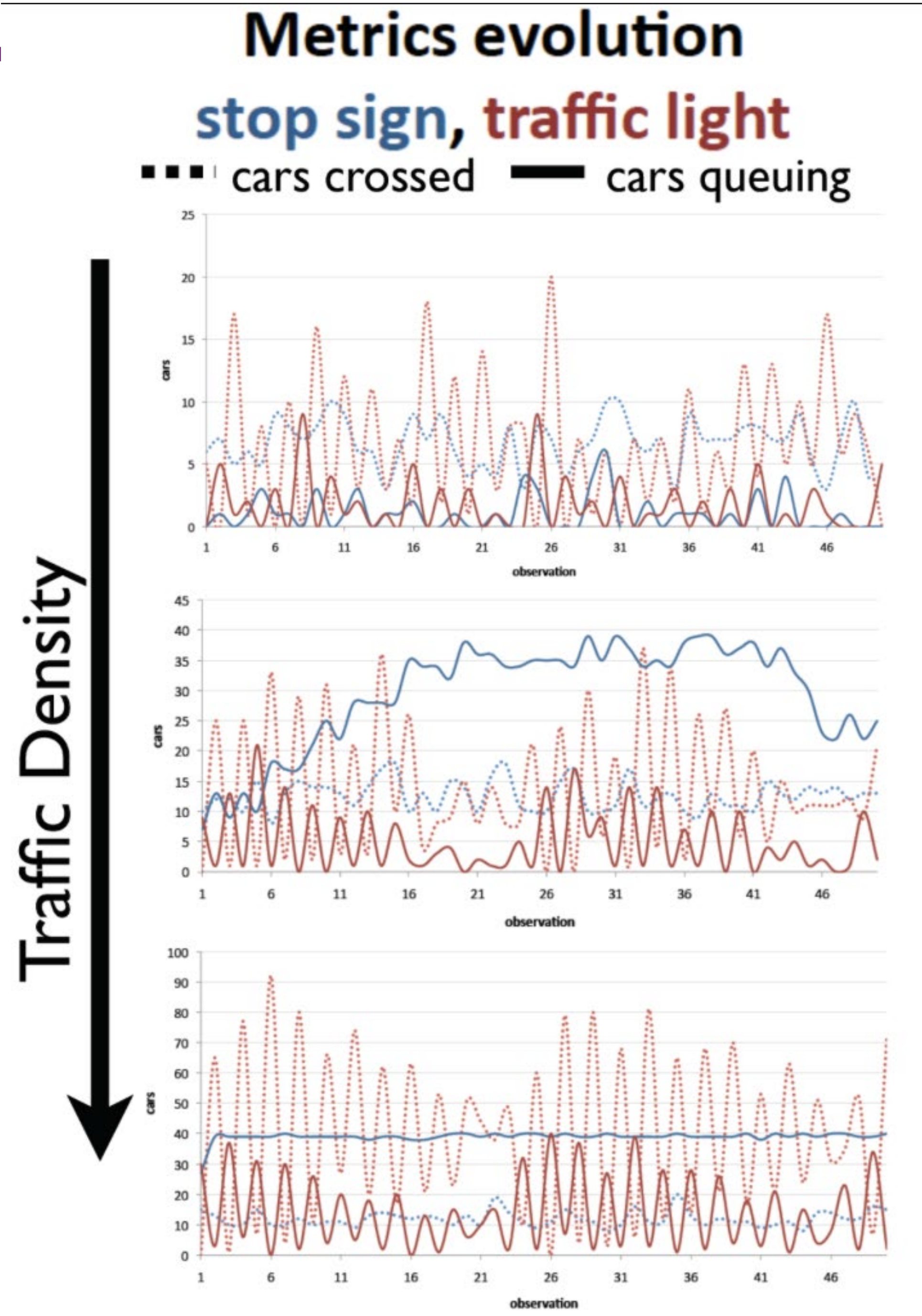
0 20

Traffic modelling

- Andrew Crooks – single intersection model
  - http://www.gisagents.org/2011/03/using-agents-to-explore-traffic.html
  - https://youtu.be/GlNbFfklg_Q

Traffic Light

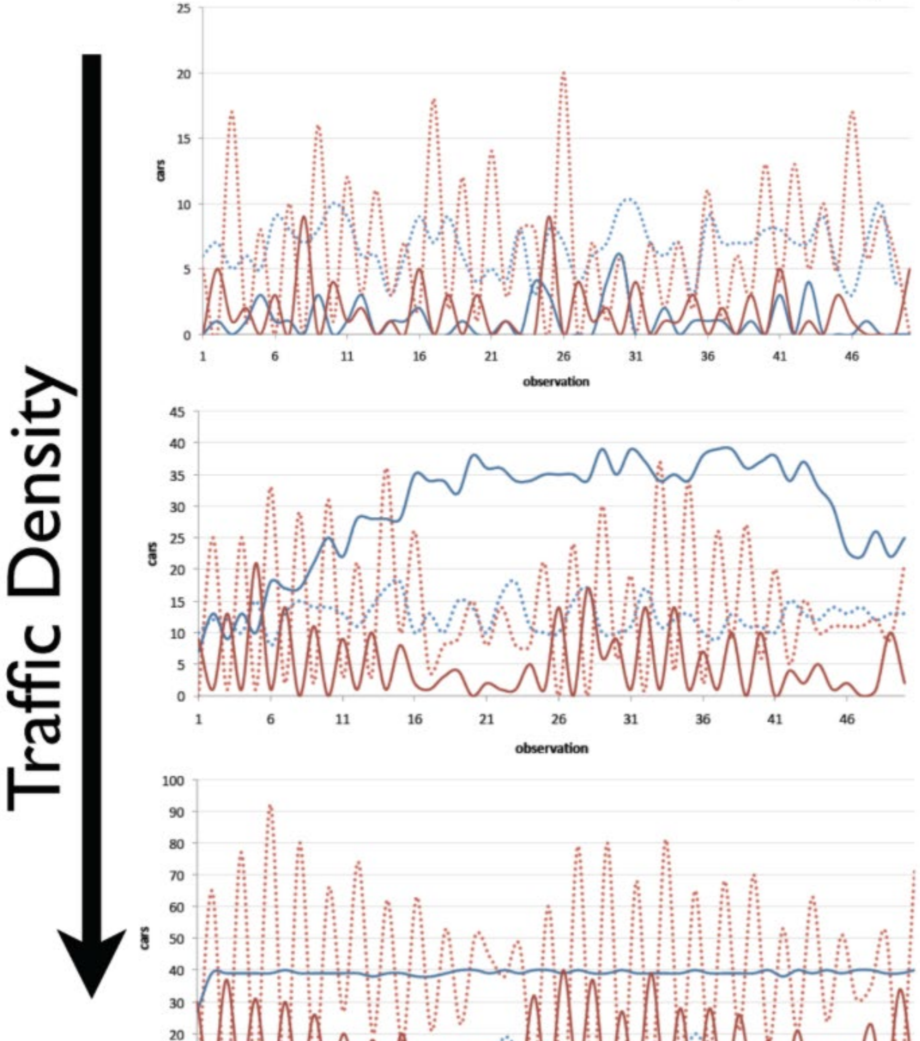




Metrics evolution

stop sign, traffic light

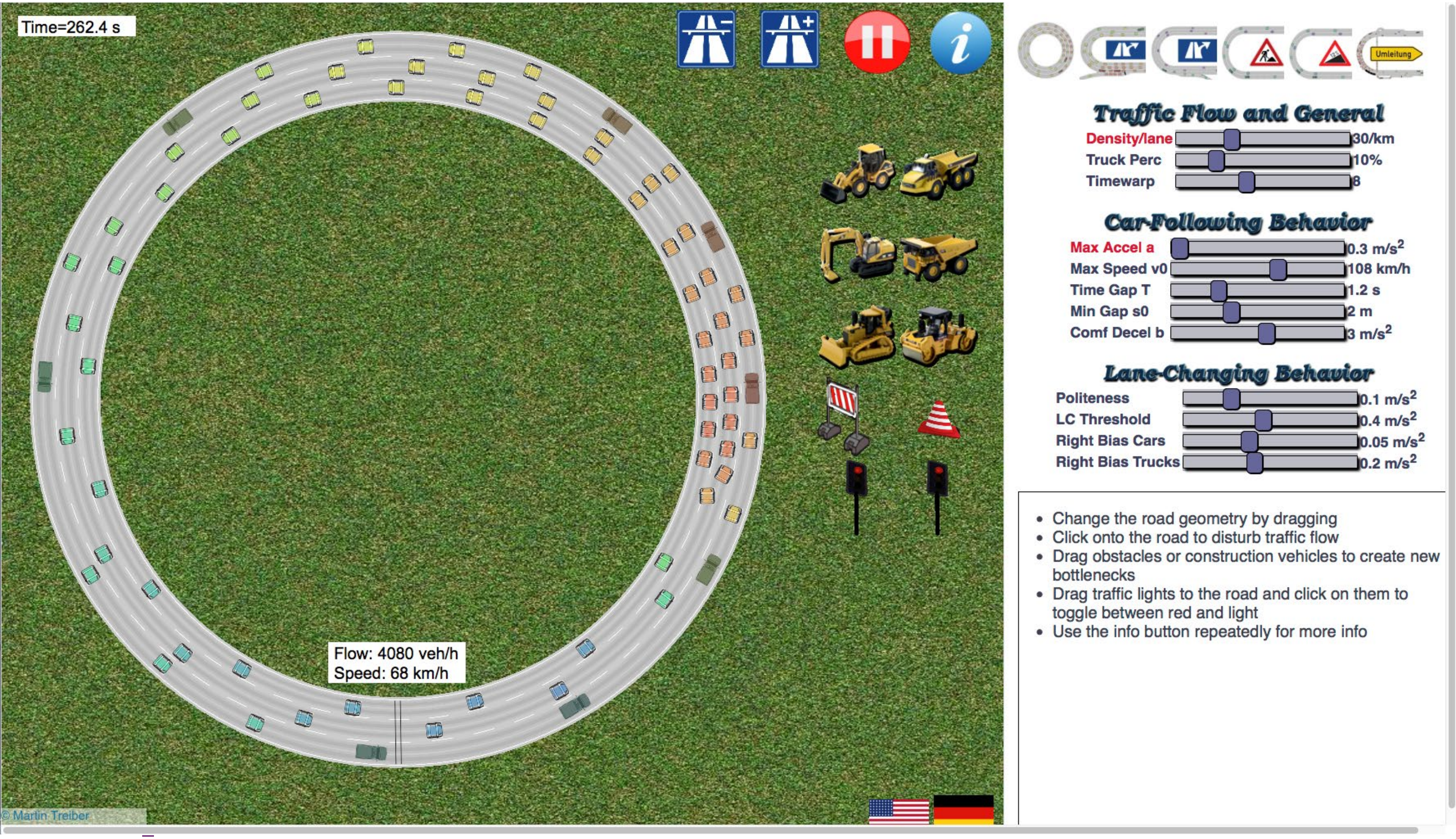
▪▪▪ cars crossed ▬ cars queuing



Traffic simulation

http://traffic-simulation.de

Martin Treiber



Time=262.4 s

Flow: 4080 veh/h
Speed: 68 km/h

Traffic Flow and General

Density/lane 30/km
Truck Perc 10%
Timewarp 8

Car-Following Behavior

Max Accel a 0.3 m/s^2
Max Speed v0 108 km/h
Time Gap T 1.2 s
Min Gap s0 2 m
Comf Decel b 3 m/s^2

Lane-Changing Behavior

Politeness 0.1 m/s^2
LC Threshold 0.4 m/s^2
Right Bias Cars 0.05 m/s^2
Right Bias Trucks 0.2 m/s^2

- Change the road geometry by dragging
- Click onto the road to disturb traffic flow
- Drag obstacles or construction vehicles to create new bottlenecks
- Drag traffic lights to the road and click on them to toggle between red and light
- Use the info button repeatedly for more info

COOL STUFF (3)

GAMES AND GRAPHICS

Fundamentals of Programming

Was in Lecture 12

Games and Graphics

- So far we've worked with the command line, notebooks and plot windows
- There are a range of packages for Python to provide graphics and games development capability
  - Pyglet, Pygame... etc. See:https://wiki.python.org/moin/PythonGameLibraries
- They need to give functionality for drawing, moving, selecting and modifying objects
- We will look at Pyglet...

Pyglet

- Pyglet is a pure python cross-platform application framework intended for game development.
- It supports windowing, user interface event handling, OpenGL graphics, loading images and videos and playing sounds and music.
- It works on Windows, OS X and Linux.

http://pyglet.readthedocs.io/en/pyglet-1.2-maintenance/index.html

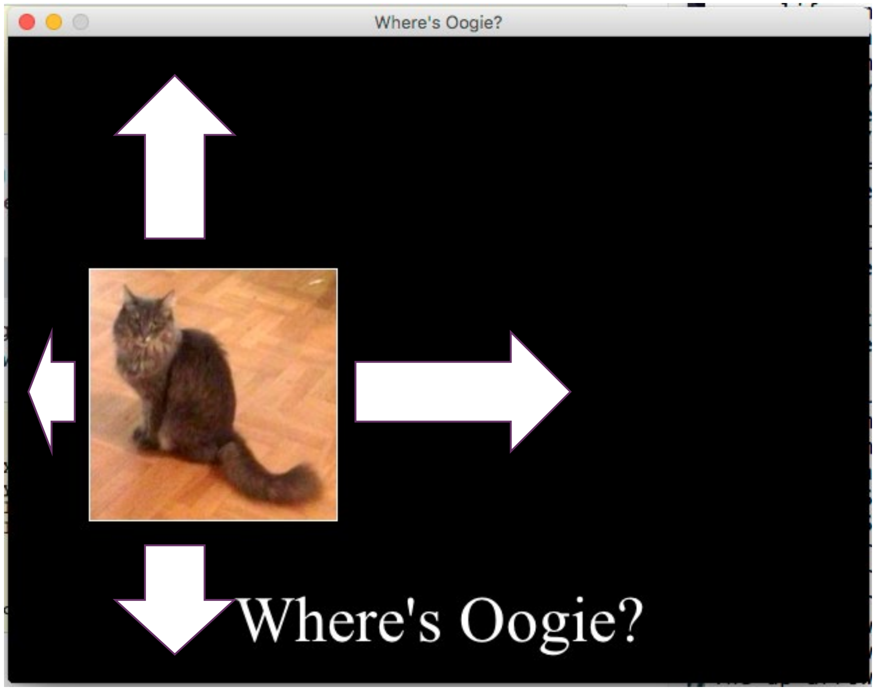
Pyglet Features

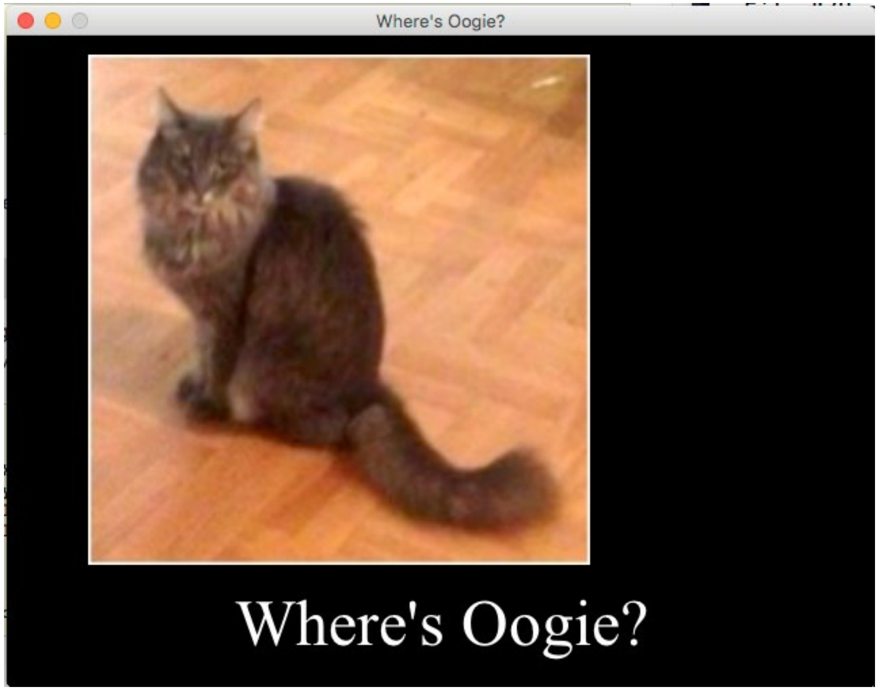
- No external dependencies or installation requirements.
- For most application and game requirements, pyglet needs nothing else besides Python.
- Take advantage of multiple windows and multi-monitor desktops.
- Load images, sound, music and video in almost any format.
- pyglet is provided under the BSD open-source license, allowing you to use it for both commercial and other open-source projects with very little restriction.

https://bitbucket.org/pyglet/pyglet/wiki/Home

An example: oogie.py

- Program to draw a cat
- Can move, resize or show/hide image of cat





oogie.py (1/4)

```
import pyglet
from pyglet.window import key
from pyglet.window import mouse


window = pyglet.window.Window(caption="Where's Oogie?")
image = pyglet.resource.image('kitten.jpg')

sprite = pyglet.sprite.Sprite(image)

label = pyglet.text.Label("Where's Oogie?",
                          font_name='Times New Roman',
                          font_size=36,
                          x=window.width//2, y=50,
                          anchor_x='center',
                          anchor_y='center')
```

oogie.py (2/4)

```
@window.event
def on_draw():
    window.clear()
    sprite.draw()
    label.draw()
```

Draw sprite (Oogie) and label

```
@window.event
def on_mouse_press(x, y, button, modifiers):
    if button == mouse.LEFT:
        print('The left mouse button was
              pressed (', x, ', ', y, ')')
    sprite.x = x
    sprite.y = y
```

If left mouse button is pressed, print position to the screen and update sprite position

oogie.py (3/4)

```
@window.event
def on_key_press(symbol, modifiers):
    if symbol == key.LEFT:
        print('The left arrow key was pressed.')
        sprite.x = sprite.x - 10
    elif symbol == key.RIGHT:
        print('The right arrow key was pressed.')
        sprite.x = sprite.x + 10
    elif symbol == key.UP:
        print('The up arrow key was pressed.')
        sprite.y = sprite.y + 10
    elif symbol == key.DOWN:
        print('The down arrow key was pressed.')
        sprite.y = sprite.y - 10
```

If arrow keys are pressed, print to the screen and update sprite position by ±10

oogie.py (4/4)

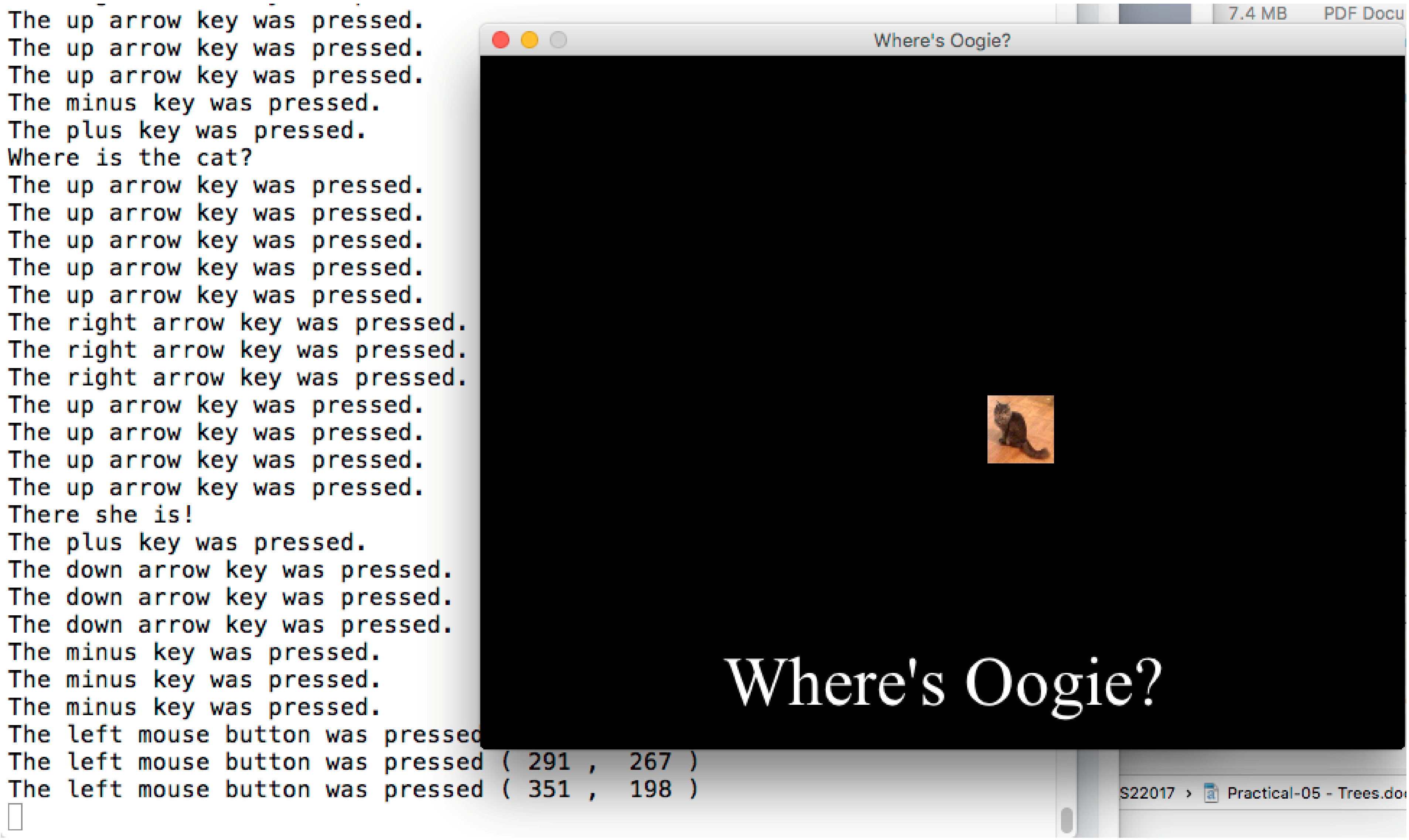
Continuation of on_key_press...

```
    elif symbol == key.H:
        sprite.visible = False
        print('Where is the cat?')
    elif symbol == key.S:
        sprite.visible = True
        print('There she is!')
    elif (symbol == key.EQUAL and
          modifiers & key.MOD_SHIFT):
        print('The plus key was pressed.')
        sprite.scale = sprite.scale * 2
    elif symbol == key.MINUS:
        print('The minus key was pressed.')
        sprite.scale = sprite.scale / 2

pyglet.app.run()
```

If "H" or "S" keys are pressed, print to the screen and hide or show cat

If plus or minus keys are pressed, print to the screen and rescale sprite by double/half

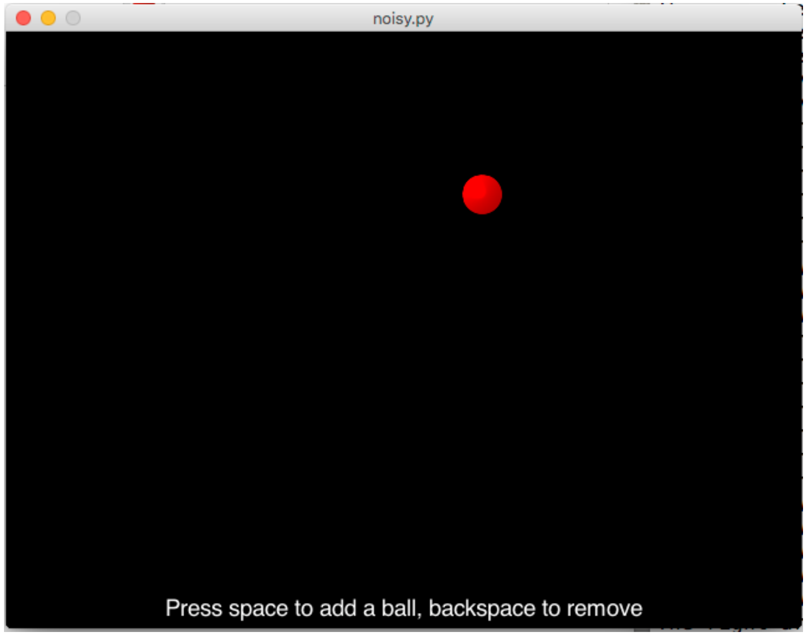
oogie.py



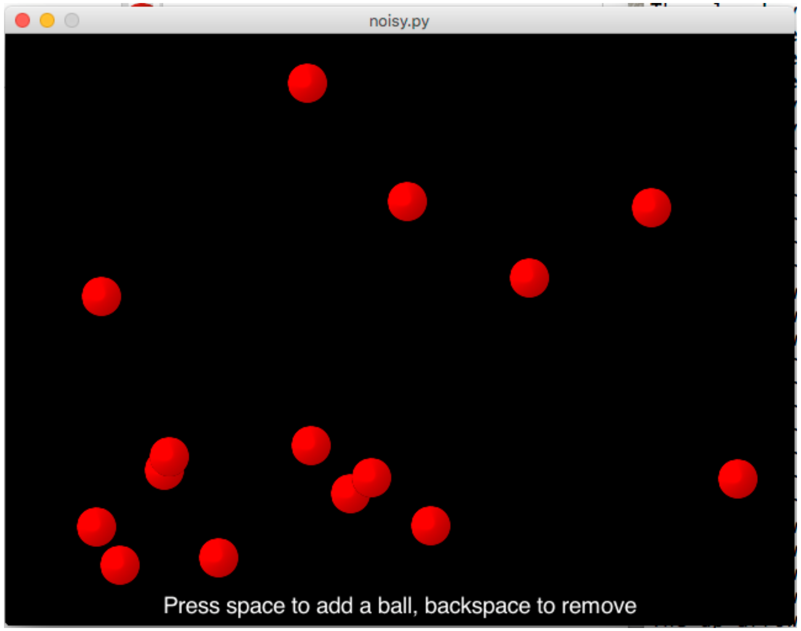
```
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The minus key was pressed.
The plus key was pressed.
Where is the cat?
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The right arrow key was pressed.
The right arrow key was pressed.
The right arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
The up arrow key was pressed.
There she is!
The plus key was pressed.
The down arrow key was pressed.
The down arrow key was pressed.
The down arrow key was pressed.
The minus key was pressed.
The minus key was pressed.
The minus key was pressed.
The left mouse button was presse
The left mouse button was pressed ( 291 ,  267 )
The left mouse button was pressed ( 351 ,  198 )
```

Noisy.py example

- Demo program to bounce balls around a window.
- A noise sounds when the balls hit the sides



Press space to add a ball, backspace to remove



Press space to add a ball, backspace to remove

Noisy.py (1/6)

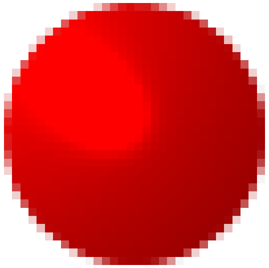
```
#!/usr/bin/env python
# ----------------------------------------------------------------------------
# pyglet
# Copyright (c) 2006-2008 Alex Holkner
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
#    the documentation and/or other materials provided with the
#    distribution.
#  * Neither the name of pyglet nor the names of its
#    contributors may be used to endorse or promote products
#    derived from this software without specific prior written
#    permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
# ----------------------------------------------------------------------------
```

https://bitbucket.org/pyglet/pyglet/src/31872c3bbb8e180da47e88568a75018dad7a8a9c/examples/noisy/?at=default

Noisy.py (2/6)

```
'''Bounces balls around a window and plays noises.
This is a simple demonstration of how pyglet efficiently
manages many sound channels without intervention.
'''

import os
import random
import sys

from pyglet.gl import *
import pyglet
from pyglet.window import key

BALL_IMAGE = 'ball.png'
BALL_SOUND = 'ball.wav'

if len(sys.argv) > 1:
    BALL_SOUND = sys.argv[1]

sound = pyglet.resource.media(BALL_SOUND, streaming=False)
```

Docstring



Option to give alternative sound as command line argument

Noisy.py (3/6)

```
class Ball(pyglet.sprite.Sprite):
    ball_image = pyglet.resource.image(BALL_IMAGE)
    width = ball_image.width
    height = ball_image.height
```

Class variables

```
    def __init__(self):
        x = random.random()*(window.width-self.width)
        y = random.random()*(window.height- self.height)
```

Temporary variables

```
        super(Ball, self).__init__(self.ball_image, x,
              y, batch=balls_batch)
```

Call to parent __init__ method

```
        self.dx = (random.random() - 0.5) * 1000
        self.dy = (random.random() - 0.5) * 1000
```

Instance variables

Noisy.py (4/6)

```
def update(self, dt):
       if (self.x <= 0 or self.x +
          self.width >= window.width):
           self.dx *= -1
           sound.play()

       if (self.y <= 0 or self.y +
           self.height >= window.height):
           self.dy *= -1
           sound.play()

       self.x += self.dx * dt
       self.y += self.dy * dt

       self.x = min(max(self.x, 0),
                    window.width - self.width)
       self.y = min(max(self.y, 0),
                    window.height - self.height)
```

Did it hit a wall?

Move the ball in x and y direction

Noisy.py (5/6)

```
window = pyglet.window.Window(640, 480)

@window.event
def on_key_press(symbol, modifiers):
    if symbol == key.SPACE:
        balls.append(Ball())
    elif symbol == key.BACKSPACE:
        if balls:
            del balls[-1]
    elif symbol == key.ESCAPE:
        window.has_exit = True

@window.event
def on_draw():
    window.clear()
    balls_batch.draw()
    label.draw()
```

Press space = Add a ball

Press backspace = Delete a ball

Redraw window: clear, then draw all balls and label

Noisy.py (6/6)

```
def update(dt):
    for ball in balls:
        ball.update(dt)
pyglet.clock.schedule_interval(update, 1/30.)
```

Update all each interval = Call update method on each ball

```
balls_batch = pyglet.graphics.Batch()
balls = []
label = pyglet.text.Label('Press space to add a ball,
                               backspace to remove',
                  font_size=14, x=window.width // 2,
                  y=10, anchor_x='center')
```

Empty list for balls

```
if __name__ == '__main__':
    pyglet.app.run()
```

If called directly, run pyglet.app.run() method

Summary

- Access and assess packages in the Python Package Index
- Evaluate risks in using packages and code from other developers
- Be aware of how to build and share a package
- Know some useful tools to support your software development

Practical Sessions

- We'll be doing some coding exercises
- We'll also use some tools and environments to explore what they can do for us

Assessments

- All students need to give a **quick demo of your assignments** during the pracs this coming week (28/10-1/11)
  - It's OK if they're not complete…

- The final in-class practical test will be during the prac sessions this coming week

- Mid-semester test results are available and papers can be collected from the tutors

Next week…

- Revision
- Some cool stuff
- Where to go next