CSCI E-29

Advanced Python for Data Science

Dr. Scott Gorlin

Harvard University

Fall 2018

Agenda

- Introductions
- What is ***Advanced*** Python?
- Tech Requirements
- Workflows
- Skeletons
- Data
- Algorithms
- Helpful Themes
- Mechanics

Introductions

Instructor

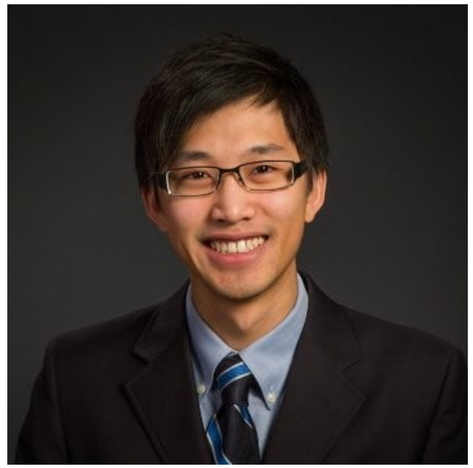
Dr. Scott Gorlin

Director of Applied Science, Solaria Labs

- Liked writing tools more than my PhD
- Led R&D for an ad tech startup
- Leading Data Science for an incubator

> *I'm impatient, so I end up building everything myself*
> *— Scott Gorlin*

TA’s



I’m occasionally confused for Andrew Ng

— Andrew Ang

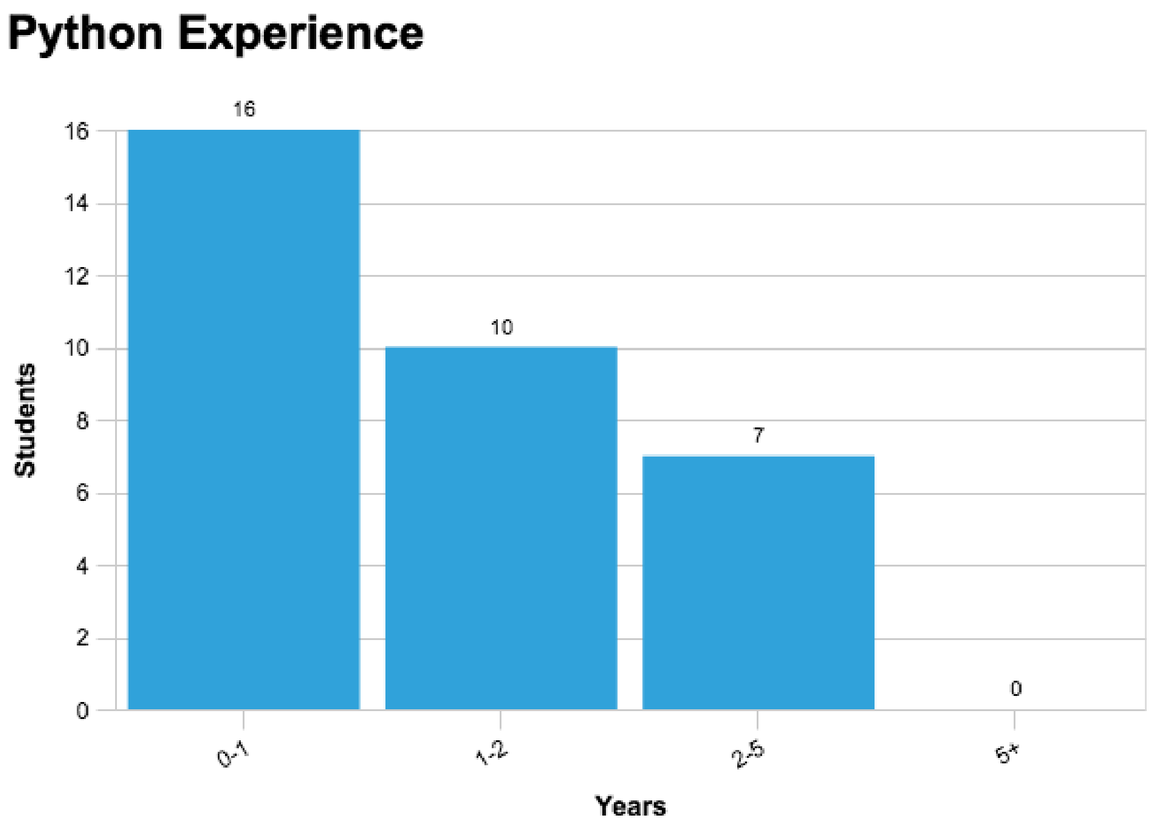
I once deleted the Atlanta Journal Constitution website

— August Schlubach

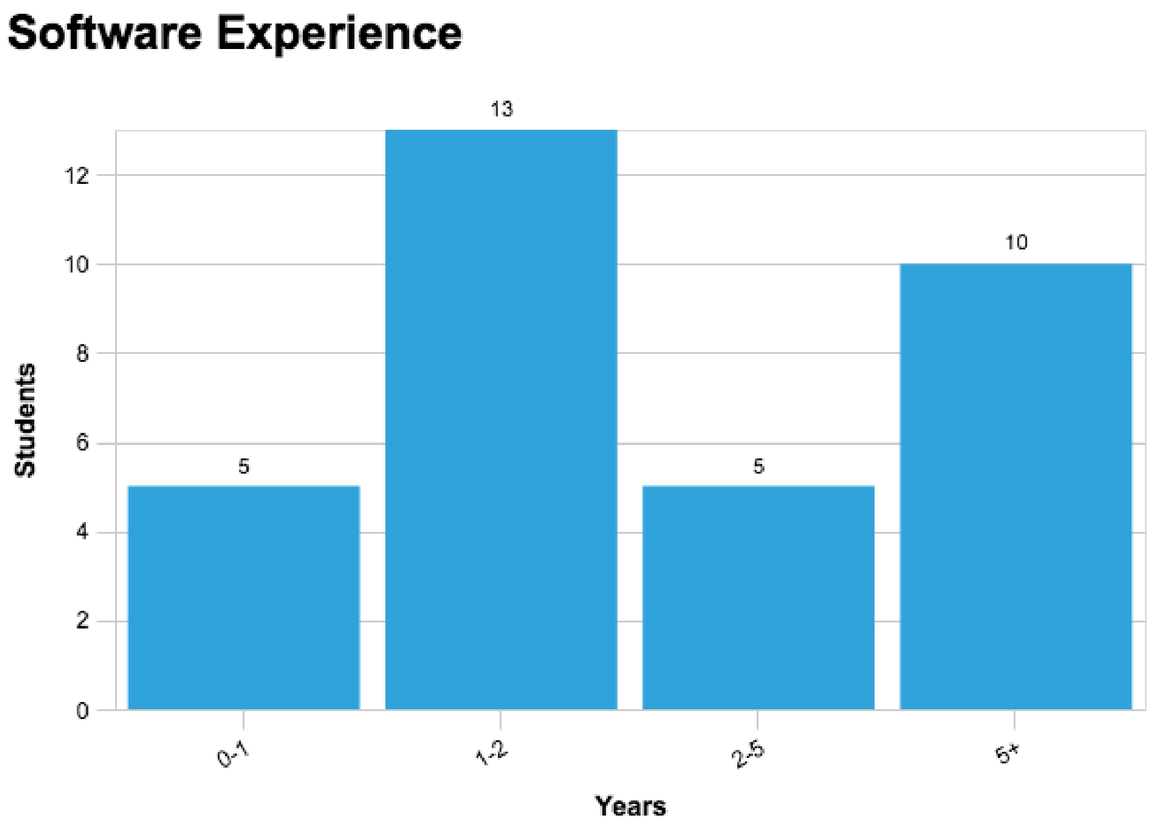
I’m a life-long IT enthusiast

— Zhenyu Zhao

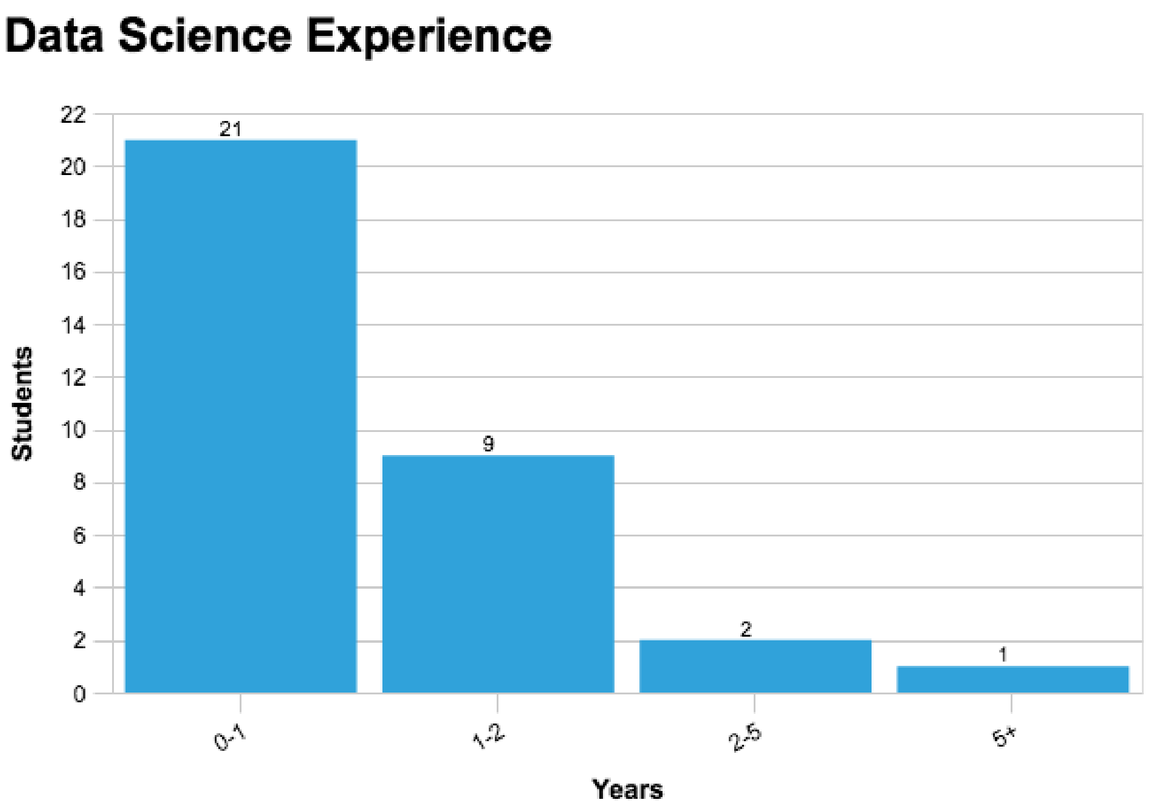
Students


Python Experience
Students
Years
0-1
1-2
2-5
5+
16
10
7
0
0
2
4
6
8
10
12
14
16

Students

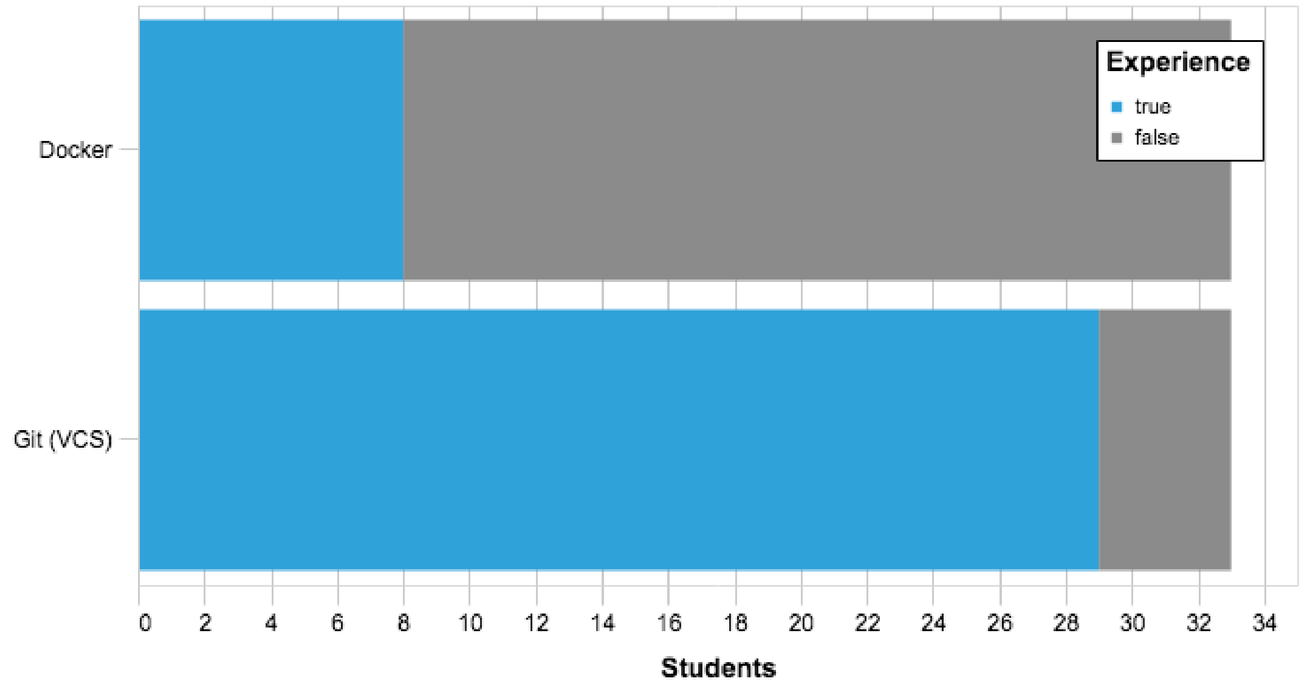

Software Experience
Students
0
2
4
6
8
10
12
13
10
5
5
0-1
1-2
2-5
5+
Years

Students


Data Science Experience
Students
Years
0
2
4
6
8
10
12
14
16
18
20
22
21
9
2
1
0-1
1-2
2-5
5+

Students

Other Experience



Admin Deets

See Canvas

Pick a section:

Tues 8

Schlubach

Thurs 8

Zhao

Fri 6:15

Ang

```cython
def primes(int nb_primes):
    cdef int n, i, len_p
    cdef int p[1000]
    if nb_primes > 1000:
        nb_primes = 1000

    len_p = 0  # The current number of elements in p.
    n = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p[:len_p]:
            if n % i == 0:
                break

        # If no break occurred in the loop, we have a prime.
        else:
            p[len_p] = n
            len_p += 1
        n += 1

    # Let's return the result in a python list:
    result_as_list  = [prime for prime in p[:len_p]]
    return result_as_list
```

What is *Advanced* Python?

File Edit View Insert Cell Kernel Help Python 3

Markdown CellToolbar

Simple spectral analysis

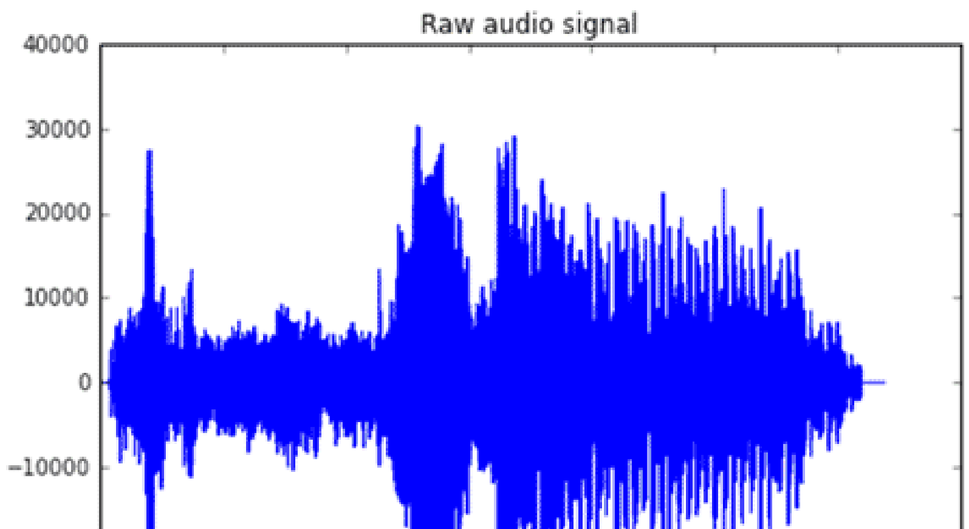
An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n exp^{\frac{-2\pi i}{N} kn} \quad k = 0, \ldots, N-1$$

In [2]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

... IS IT AN AWESOME NOTEBOOK?

In [3]:
```
fig, (ax1, ax2) = plt.subplots(1,2,figsize(16,5))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



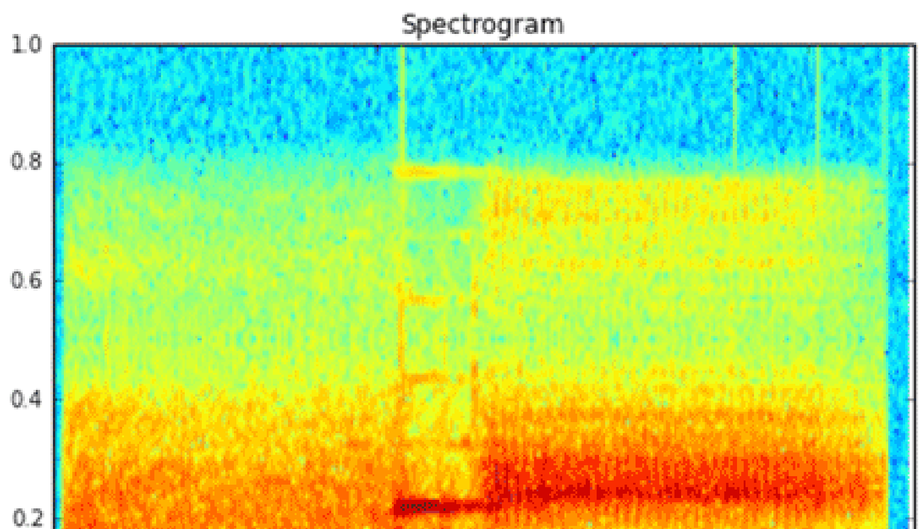


Table Of Contents

pandas.pivot_table

`pandas.`**`pivot_table`**(*data*, *values=None*, *index=None*, *columns=None*, *aggfunc='mean'*, *fill_value=None*, *margins=False*, *dropna=True*, *margins_name='All'*) [source]

Create a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame

data *: DataFrame*

values *: column to aggregate, optional*

index *: column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.

columns *: column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table column. If an array is passed, it is being used as the same manner as column values.

Parameters:

aggfunc *: function or list of functions, default numpy.mean*

If list of functions passed, the resulting pivot table will have hierarchical columns whose top level are the function names (inferred from the function objects themselves)

fill_value *: scalar, default None*

Value to replace missing values with

margins *: boolean, default False*

Add all row / columns (e.g. for subtotal / grand totals)

dropna *: boolean, default True*

Do not include columns whose entries are all NaN

margins_name *: string, default 'All'*

Name of the row / column that will contain the totals when margins is True

... OR KNOWING EVERY FUNCTION?

```
1 %timeit ts_idx_1 = df['High'].sort_values().
2 ts_idx_1
```

```
1000 loops, best of 3: 301 µs per loop

Timestamp('2017-05-15 00:00:00')
```

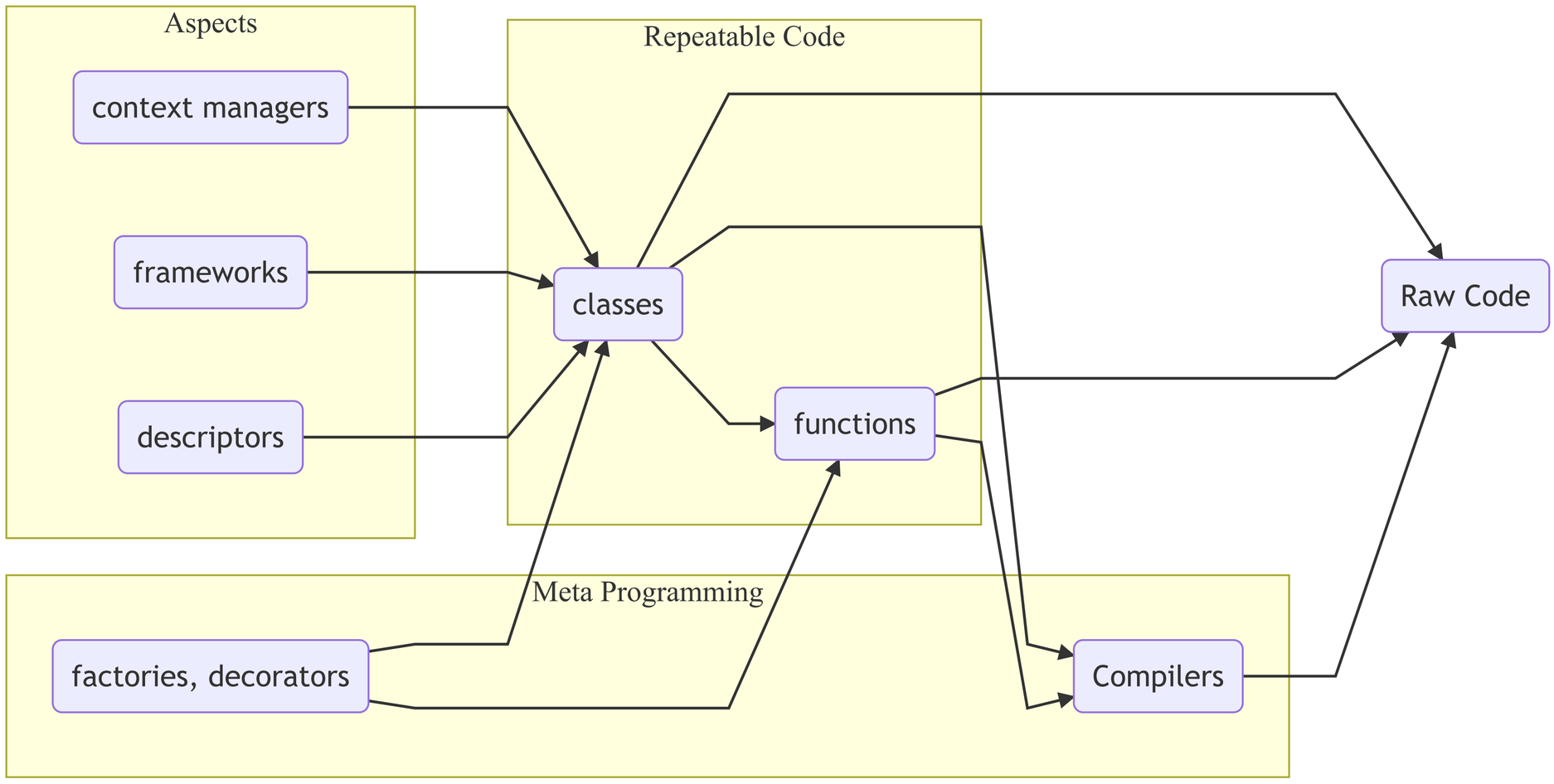
... AND WRITING FASTER CODE?

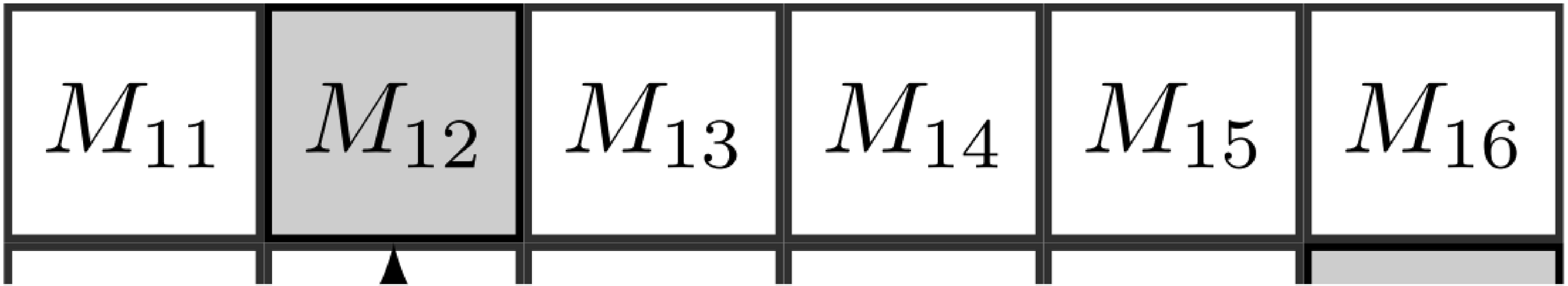
```
1 %timeit ts_idx_1 = df['High'].idxmax()
2 ts_idx_1
```

```
10000 loops, best of 3: 79.3 µs per loop

Timestamp('2017-05-15 00:00:00')
```

It's about higher levels



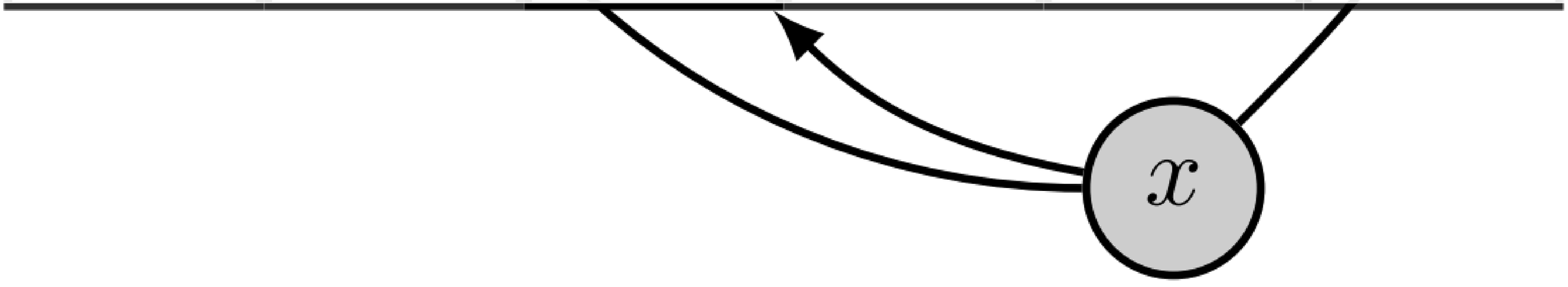


It's about the science

Columnar & Sparse Representations

Probabilistic Data Structures

Parallel & Distributed Algorithms



It's about repeatability

Manual Everything

```
pip install numpy
python script.py in.csv out.csv
```

- No env
- No data flow

Minimal

```
pip install -r reqs.txt
python script.py in.csv
```

- Minimal reqs
- Implicit data flow

Fully Repeatable

```
docker-compose build
docker-compose run make out.csv
```

- Full environment
- Explicit data flow

... AND REPEATABLE ENVS

Environment Specification	Python packages	Compiled Extensions	Non-python	OS
setup.py, pip	Minimal	Sometimes	Never	
Pipenv, pip freeze	Frozen	Sometimes	Never	
Conda envs	Minimal or frozen			Some
Docker/Containers	Can be fully specified, minimal or frozen			

It's about the ecosystem

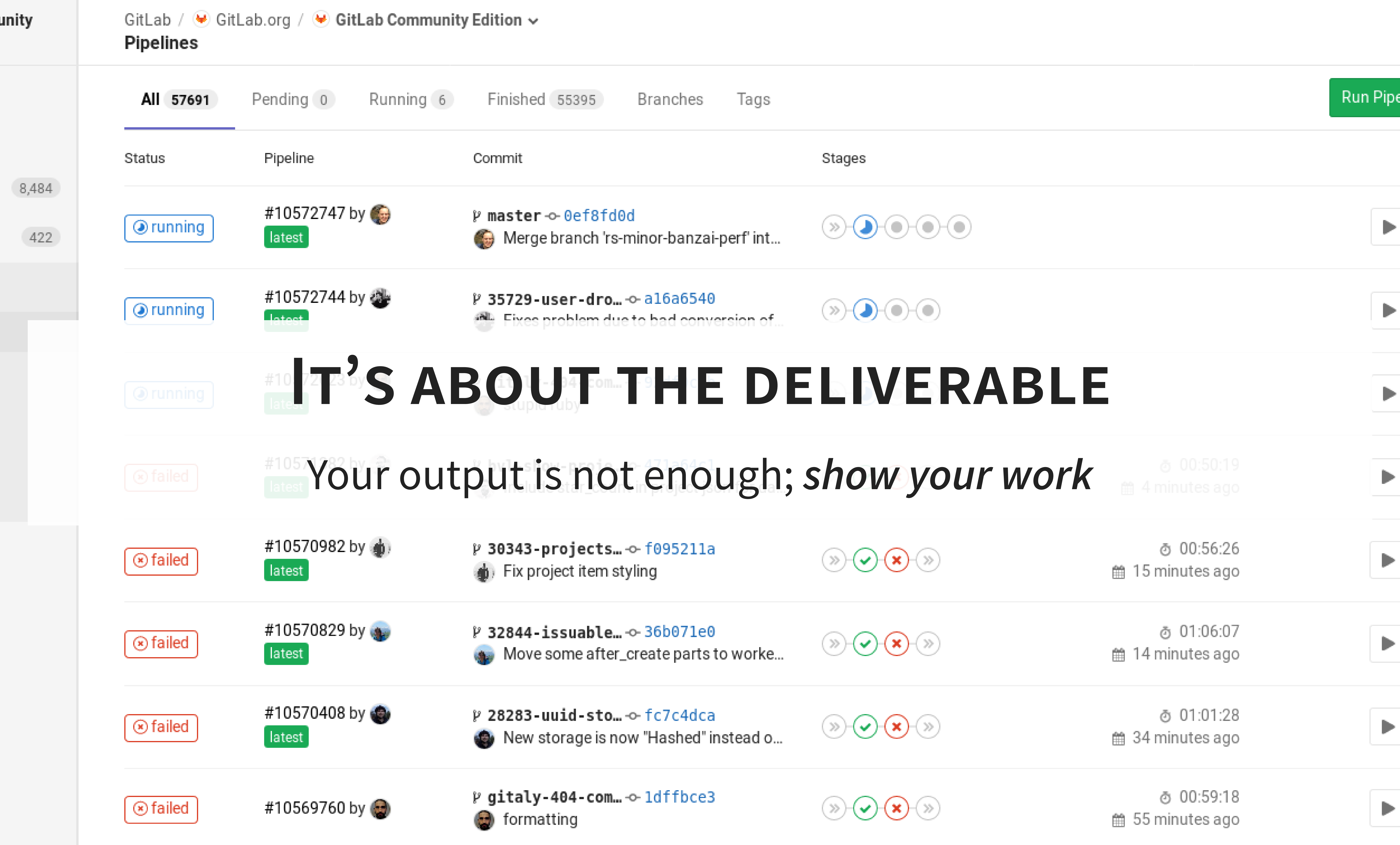
The modern Data Scientist leverages a broader ecosystem of tools than anyone else in the history of man

Python is a ***glue*** language, and is popular because it is the ***best interface*** to all of these tools

We will ***embrace*** the ecosystem, rather than have you work in a sandbox

IT'S ABOUT THE DELIVERABLE

Your output is not enough; *show your work*



Tech Requirements

Prerequisites

You should already know Python!

Class assumes ***fluency*** in basic to intermediate python syntax

You should know how to use libraries, functions, and classes

You should be familiar with numpy, scipy, and pandas

Many good tutorials to brush up, eg from Software Carpentry:

- Python
- Python Testing
- Make and Makefiles
- git

Python 3.6

This class will target 3.6, mainly for syntax and back-ported features

2.7 If 2.x Needed

- `__future__`, `six`
- Backports
- **Time to let it go**

3.5/6 Just Right

- Unicode
- Consistent syntax
- Type hints

>=3.7 Too new

- **Data classes**
- Lazy annotations
- `Resource`

There are a number of new minor features in 3.6 vs 3.5 we may explore

However, we will largely write code that is **2/3 compliant**

IDE

Pick one that helps with:

- Debugging!
- Source code/doc lookups
- Syntax checks, autocomplete, runtime config…

PyCharm is free for educational use with your Harvard email: https://www.jetbrains.com/pycharm/

Git

Your work will be submitted through git and GitHub

- Version everything
  - Don't just submit one uber commit
  - You will be graded on *appropriate* git history
- Don't version ***everything*** (eg `.pyc` or `.idea`)
  - `.gitignore`
- TA's can assist, but…
- Git usage is largely outside the scope of this course
- Pick a good Git Gui like Sourcetree

Resources: Github, Pro Git, Software Carpentry

Git Commit Guidelines

Pick a convention and stick with it

```
<type>[optional scope]: <description>
-- eg --
feat: allow provided config object to extend others
feat(lang): added Polish language
test(alg): new tests for dot products
fix(training): remove training data from test set
docs: correct spelling of CHANGELOG
```

People ***rarely*** read past the subject

http://conventionalcommits.org/

XKCD 1296

Documentation

You need to document and comment your code!

Pick one of:

- Google Docstring
- Numpy
- SphinxReST

```
def fetch_bigtable_rows(big_table):
    """Fetches rows from a Bigtable

    More notes...

    Args:
        big_table: A Bigtable Table
        ...

    Returns:
        A dict...

    """
```

Whitespace

For the love of all things…

2 spaces

```
def f(x):
··pass
```

4 spaces

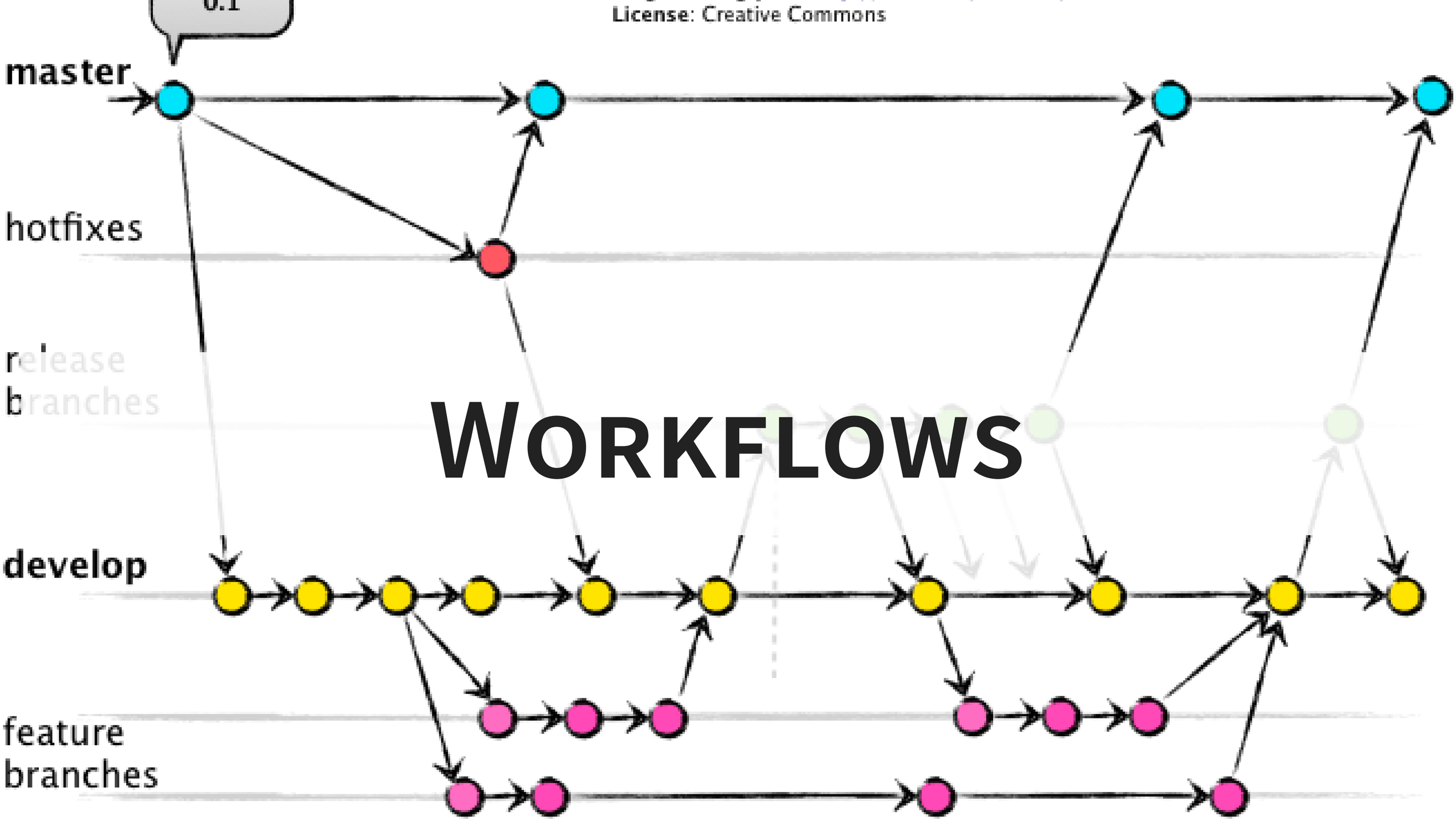
```
def f(x):
····pass
```

⇥ Tab

```
def f(x):
⇥pass
```

Whitespace means ***four spaces***. Set this in your IDE.

Workflows



“I get why a rigorous workflow is theoretically better, but if it’s mostly just me working on one thing at a time, I don’t really think it’s worth it”

— Most people at some point in their career

FALSE

Two Developers

You team is always at least size two:

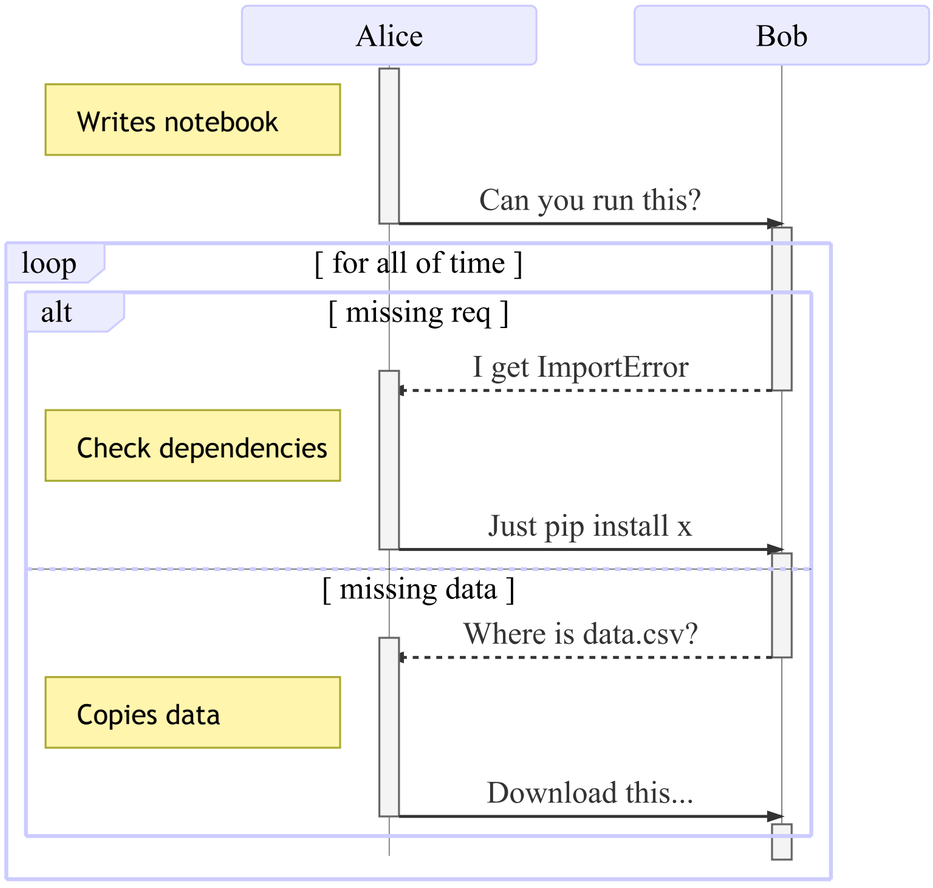
... the
original developer

... the
original developer
few months later

Tarin Gamberini

Reproducible Science

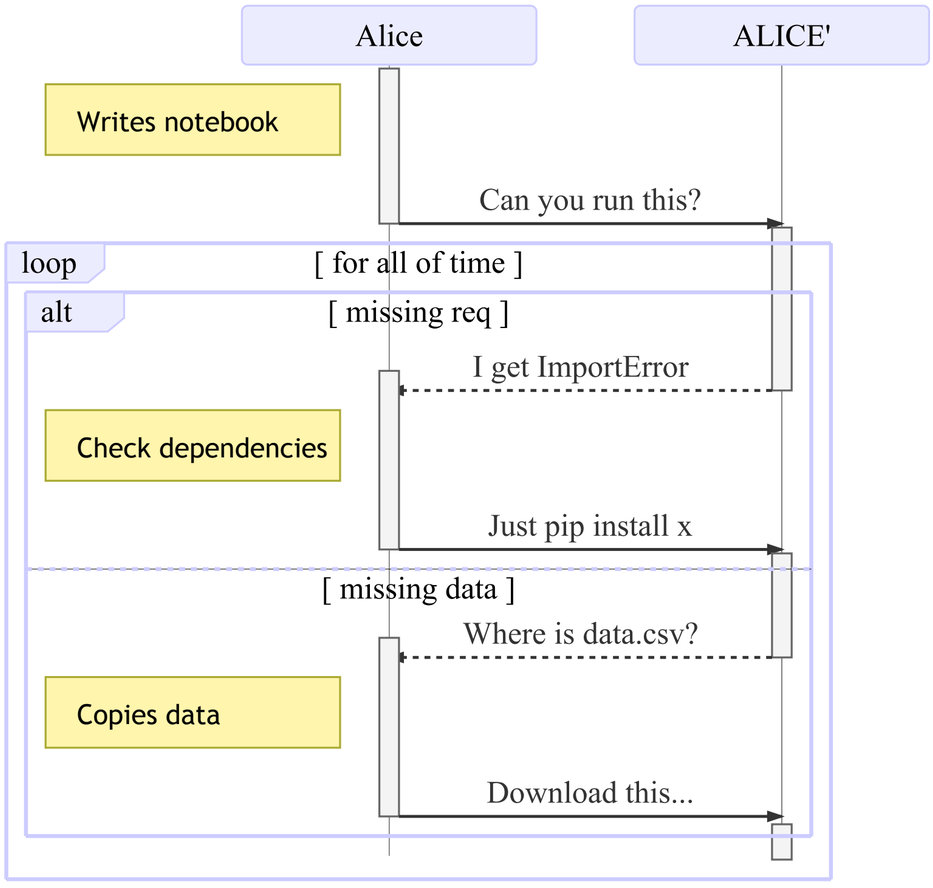
Does this look familiar?



Reproducible Science

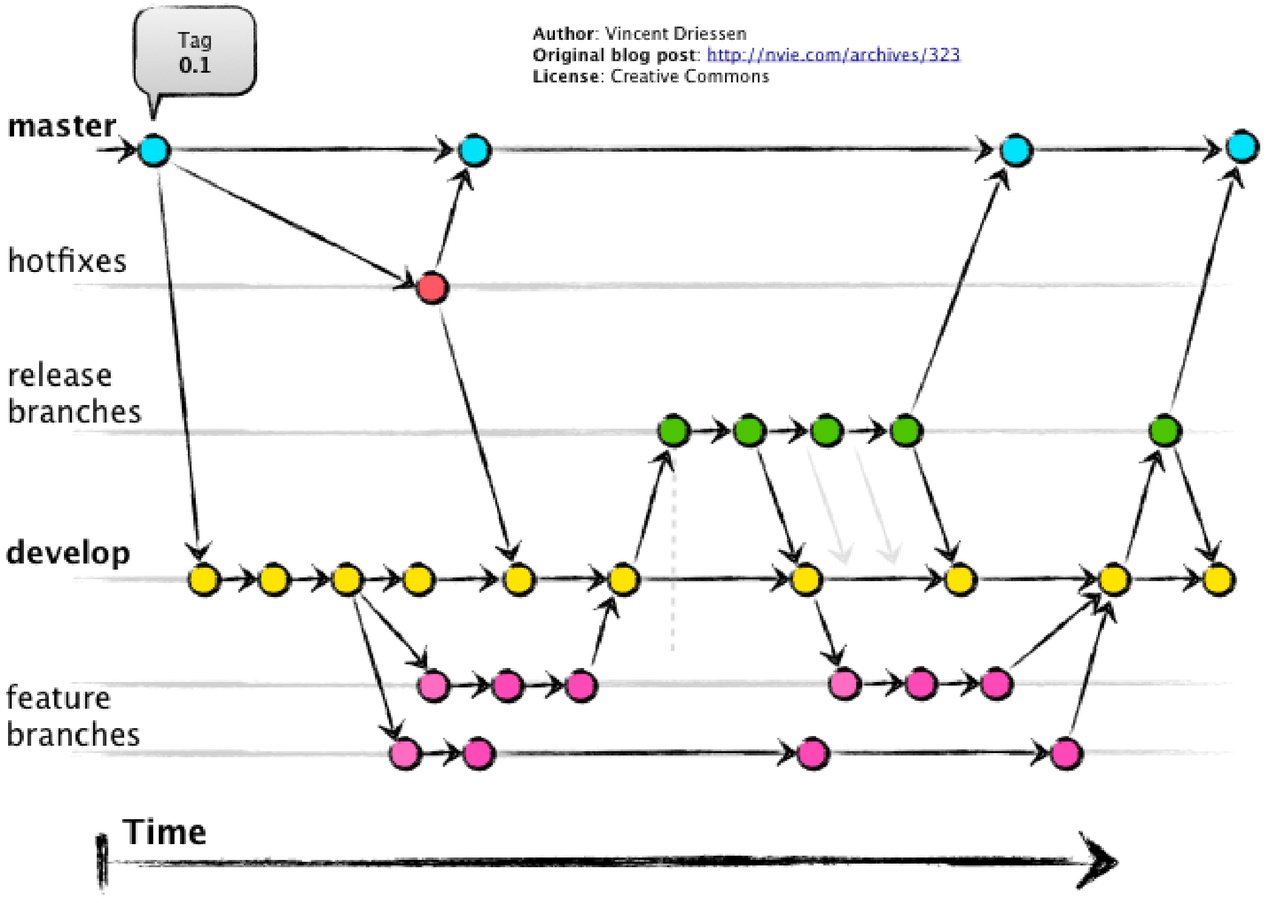
Does this look familiar?

… six months later



Git Flow

It matters how you ***save your work***



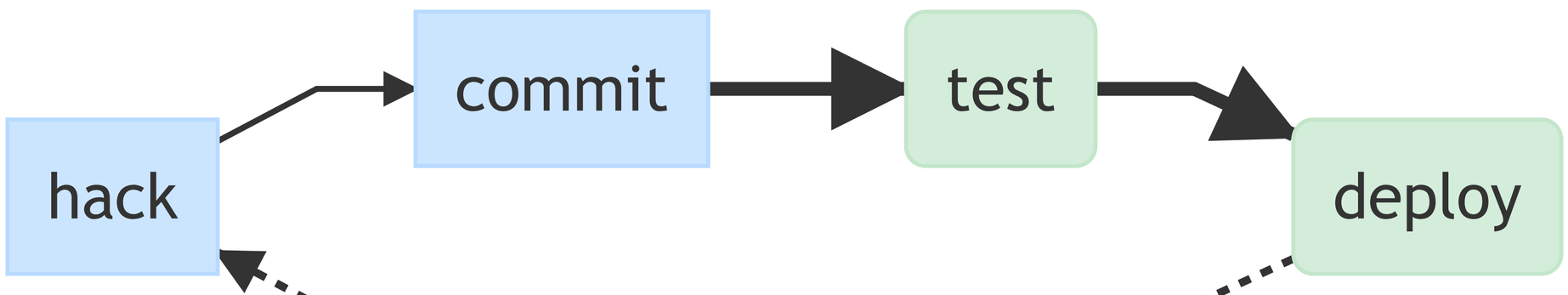
Continuous Science

CI/CD

- Rapidly merge code
- Well tested
- If tests pass, deploy
- Repeat

Science!

- Repeatable envs
- Repeatable data
- Tested algorithms



```python
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

Skeletons

The business logic of the task

Where it writes output

What other tasks it depends on

Frameworks

A skeleton is the backbone and foundation of your project

It is a framework you build upon, within, and around

It may help you see your problem in a new light

> *“A language that doesn’t affect the way you think about programming, is not worth knowing”*
>
> *— Alan Perlis*

Cookiecutter

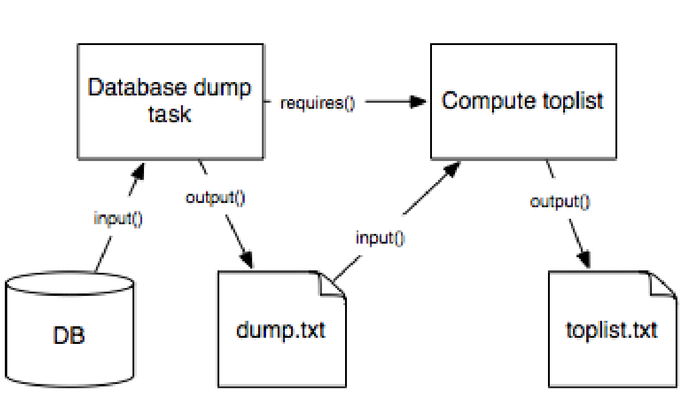
Project templates!


```
cookiecutter-something/
├── {{ cookiecutter.project_name }}/  <--------- Project template
│   └── ...
├── blah.txt                          <--------- Non-templated files/dirs
│                                                go outside
│
└── cookiecutter.json                 <--------- Prompts & default values
```

Luigi

DAG workflow:

- **D**irected
- **A**cyclic
- **G**raphs




```python
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

The business logic of the task

Where it writes output

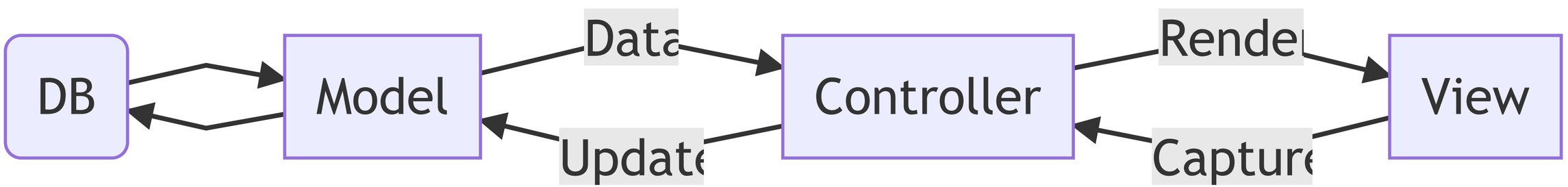
What other tasks it depends on

Parameters for this task

Django

Model-View-Controller

Separation of data, logic, view, and interaction



ORM

Pythonic, testable DB's and SQL

Row group 0

Column a

Page 0

Page header (ThriftCompactProtocol)

Repetition levels

Definition levels

values

Page 1

Column b

Row group 1

Footer

FileMetaData (ThriftCompactProtocol)

- Version (of the format)
- Schema
- extra key/value pairs

Row group 0 meta data:

Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs

column "b" meta data

Row group 1 meta data

Footer length (4 bytes)

DATA

Column Stores

Row Store

Vertical Partitions

id	name	grade
1	Scott	89
2	Andrew	91
99	Mary	87

Column Store

Vertical *and* horizontal partitions

id	name
1	Scott
2	Andrew
99	Mary

id	grade
1	89
2	91
99	87

Most operations only care about a few columns

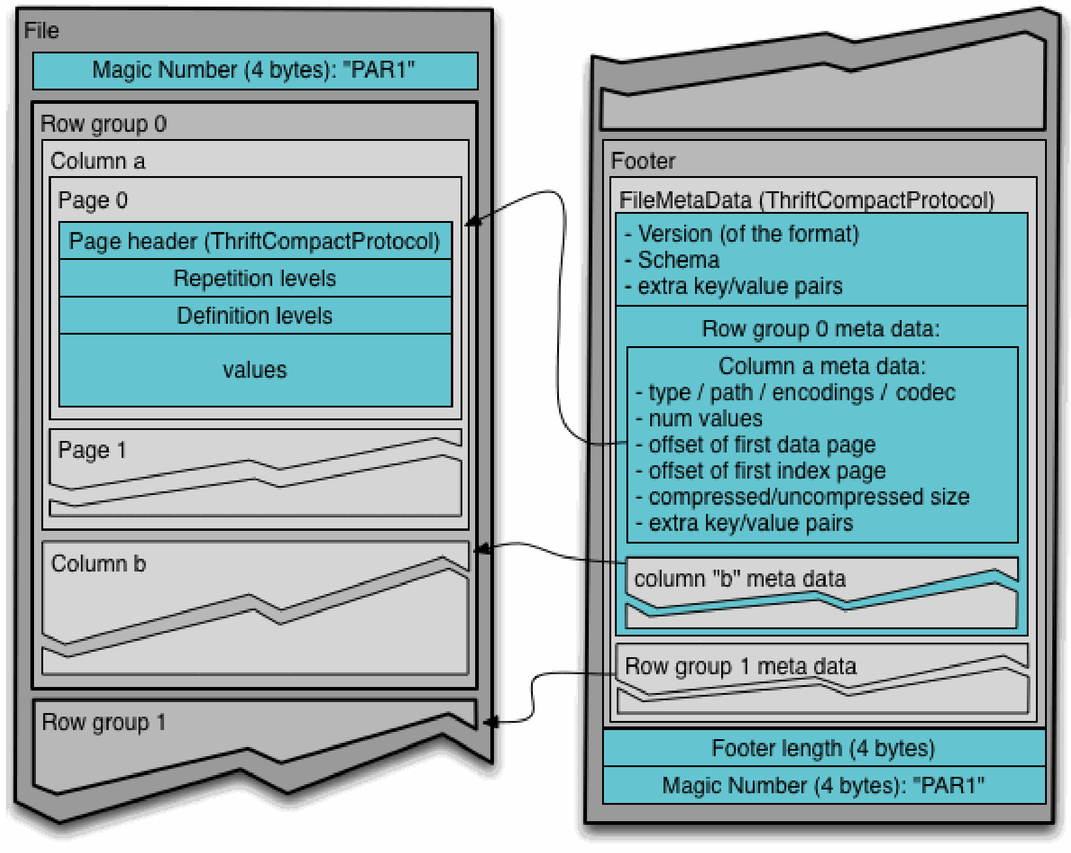
Don’t read them all if you don’t need to!

Apache Parquet

Emerging standard for distributed, columnar binary data

Statistics for predicate pushdown

http://parquet.apache.org/

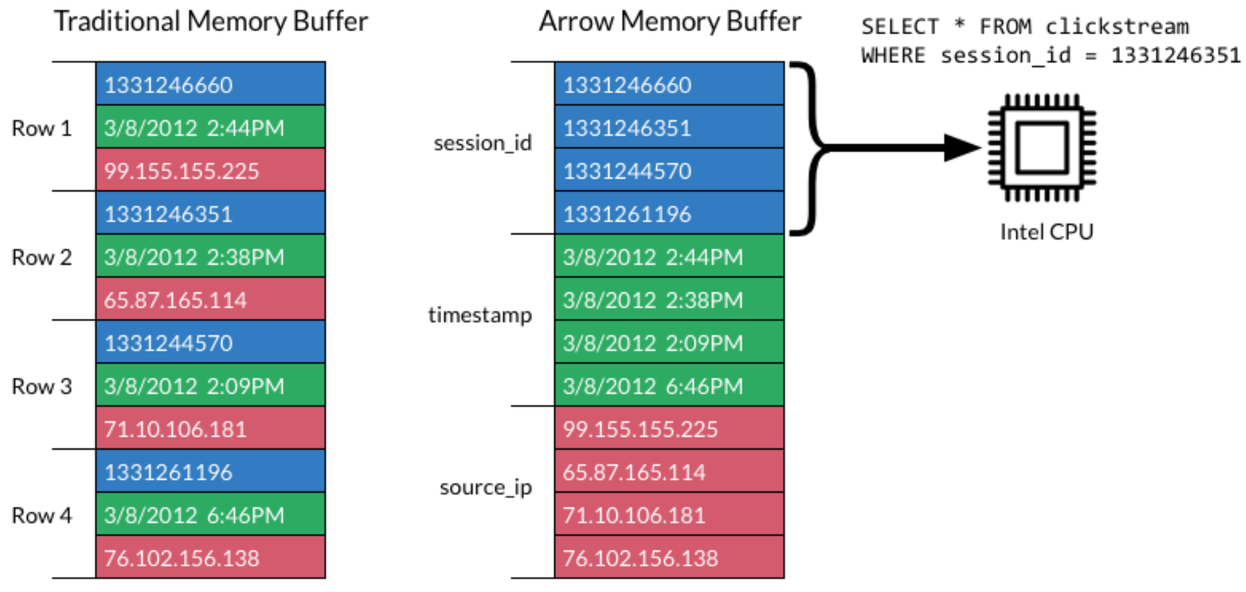


Apache Arrow

Unification of column stores - Pandas, Parquet, Spark, …

http://arrow.apache.org/

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138



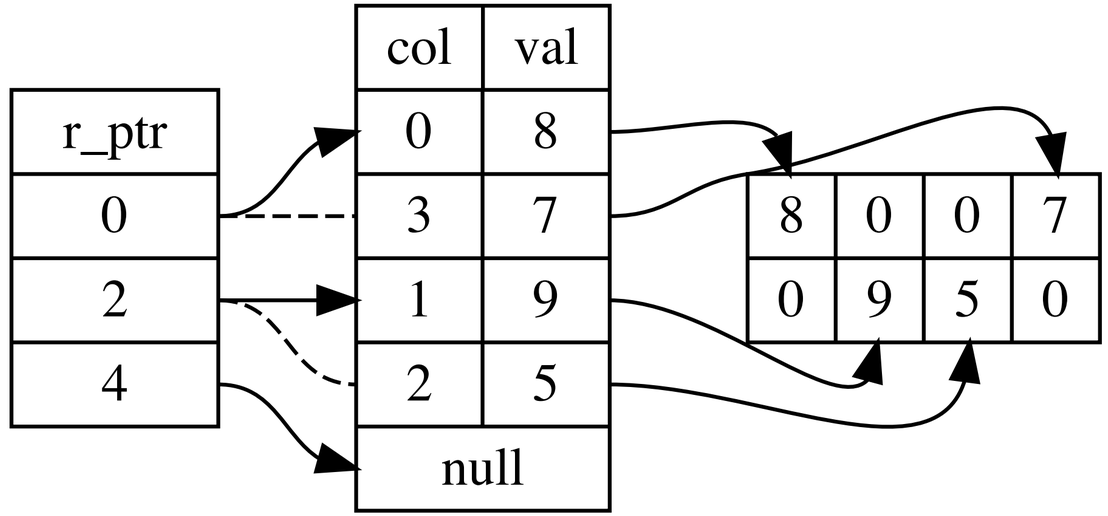
Sparse Structures

Many ways to say "I 0 You"

Compressed Sparse Rows (dense rows, sparse columns)

For R rows, C columns, and N nonzero elements, storage $O(R + N)$

```
M = np.zeros(R, C)
for r in range(R):
    slc = slice(r_ptr[r], r_ptr[r+1])
    M[r, col[slc]] = val[slc]
```



Memoization

Caching and Decorators

Quick and easy speedups

LRU Cache

‘Ideal’ constant-memory cache

```
from functools import lru_cache

@lru_cache
def f(x):
    ...
```

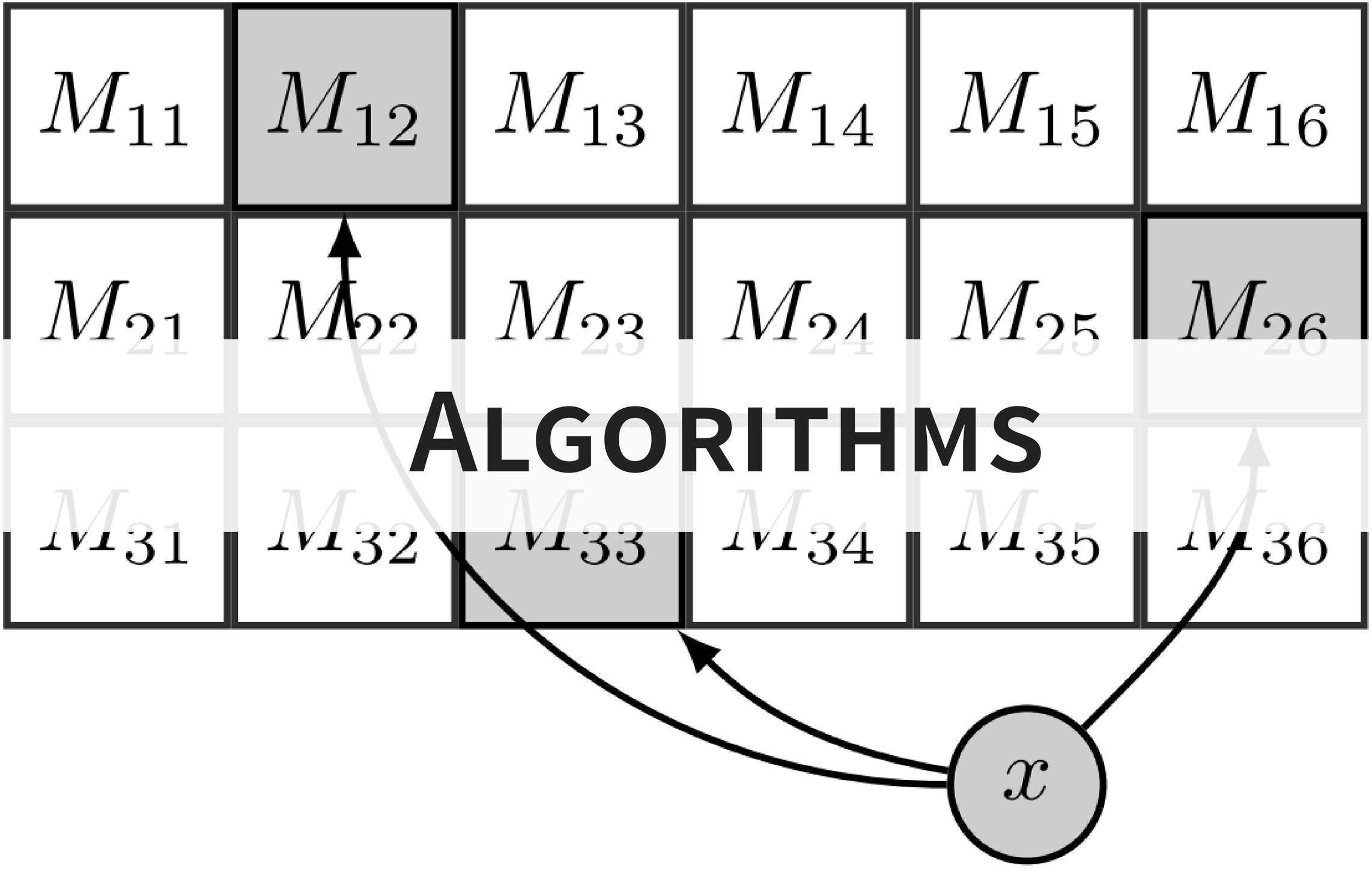
```
from functools import wraps

class Memoizer(dict):

    def __call__(self, func):
        @wraps(func)
        def wrapped(*args):
            if args in self:
                return self[args]
            o = func(*args)
            self[args] = o
            return o
        return wrapped
```

```
cache = Memoizer()

@cache
def f(x):
    return x + 1
```


M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} . . .
M_{21} M_{22} M_{23} M_{24} M_{25} M_{26} . . .
M_{31} M_{32} M_{33} M_{34} M_{35} M_{36} . . .
x

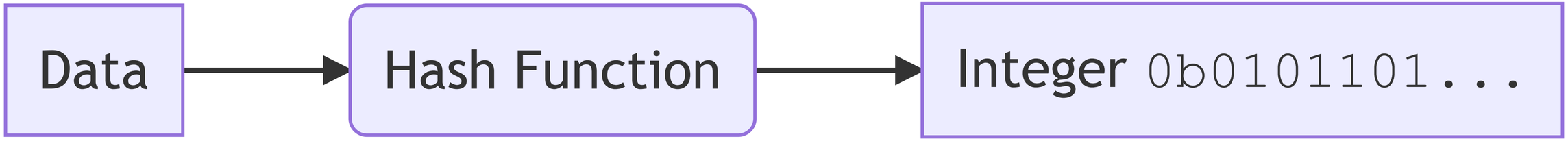
ALGORITHMS

What's important

The vast majority of algorithms you encounter will be project specific

There are a few core ideas that underly many of the algorithms you actually use

The Hash

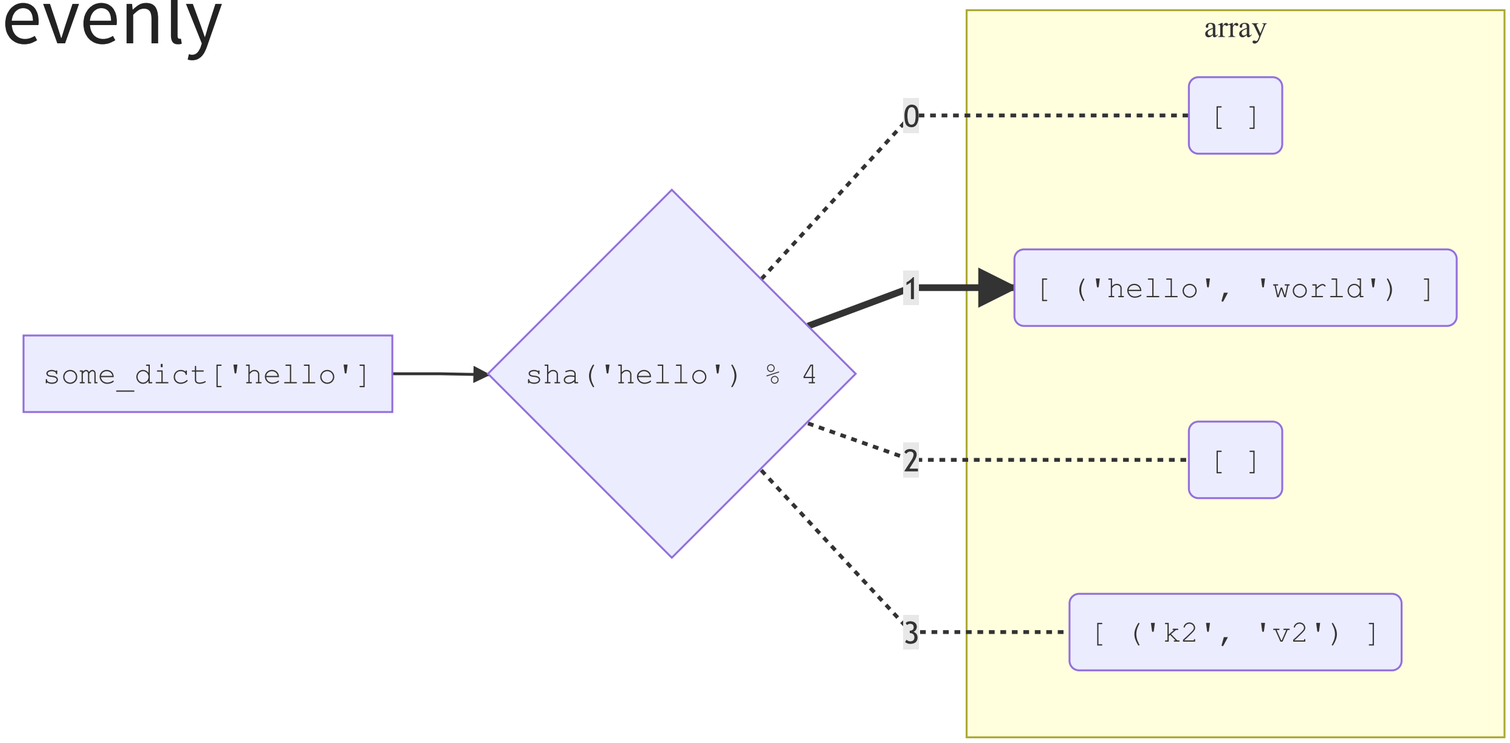


A hash function, like sha256, maps an arbitrary string of data into a fixed-width integer, eg 256 bits

The integer appears ***uniform*** and is deterministic, which makes it very useful for many algorithms…

The Hash (Lookups)

Hashes distribute lookups evenly



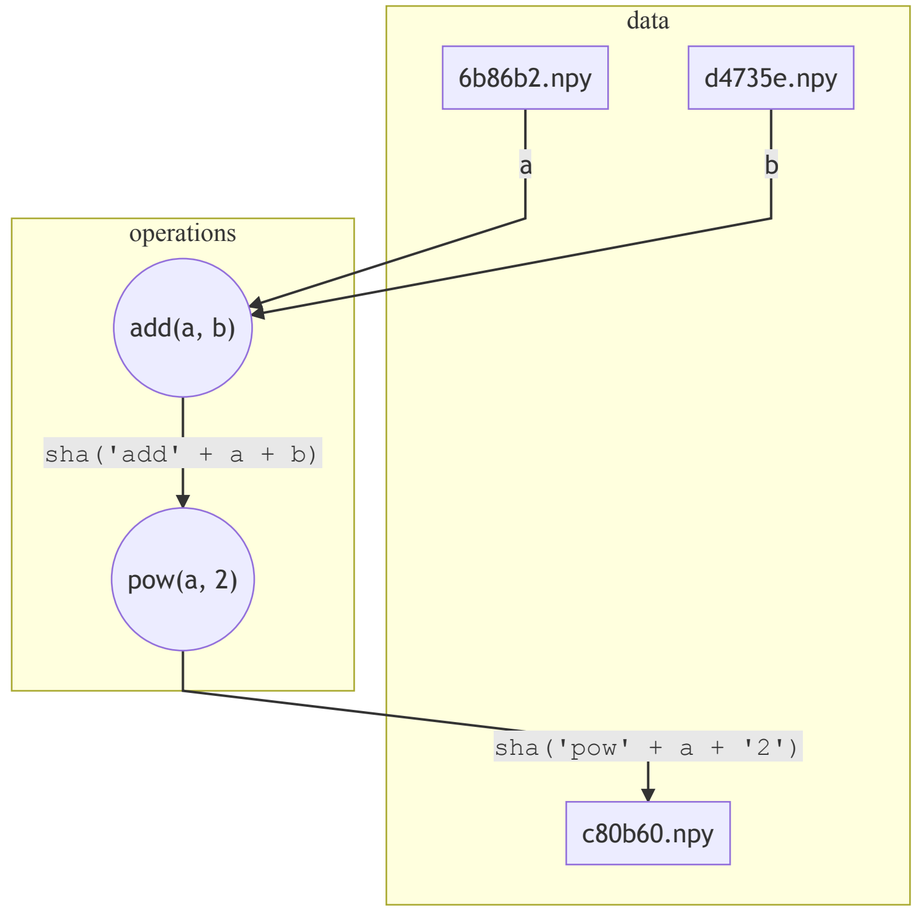
The Hash (Signatures)

Hashes sign and validate data

An operation may return a new hash of its signature and the hashes of its inputs

This mechanism underlies git, blockchain, and many other systems that guarantee some form of data lineage and integrity

$$f(a, b) = (a + b)^2$$



The Hash (Probability)

How do you count unique items without sorting or storing in memory?

Measure rarity (probability) of each item,

$$|S| \propto \frac{1}{\min_i p_i}$$

$$sha(d) = 0b\mathbf{00000}10110..., p(d) := 2^{-5}$$

Helpful Themes

Typing

Python doesn’t require types, but you shouldn’t forget them!

The following all ‘safely’ handle a basic lookup, but make different decisions about the type and content of the default value

```
ENCODING = {
    1: 'one',
    2: 'two',
}
def encode(x):
    # Preserves type, but is a new 'magic' value
    return ENCODING.get(x, 'none')
```

```
def encode(x):
    # Standard tombstone value, but
    # may cause problems eg encode(9).upper()
    return ENCODING.get(x, None)
```

```
def encode(x):
    # Preserve type and boolean eg 'if encode(x)'
    return ENCODING.get(x, '')
```

Typing

Functions should operate on an expected data type, and return an expected type.

If this changes depending on the arguments, mass confusion will follow.

Strive for consistent, self-explanatory argument and return types

Never ever ever

```
def sqrt(x):
    if x < 0:
        return 'invalid'
    return x**0.5
```

Typing (Overkill)

Let your IDE and common sense do the heavy lifting

Good practices in the past are not *as* necessary today

See: Hungarian Notation

```python
def float_square_root(float_x):
    assert isinstance(float_x, float)
    return x**0.5
```

This is overkill!

> *“Encoding the type of a function into the name … is brain damaged - the compiler knows the types anyway … and it only confuses the programmer.”*
>
> — *Linus Torvalds*

Code as Data

Try to write every program as an operation on data, cf:

Good

```
print('one')
print('two')
```

Better

```
for arg in ['one', 'two']:
    print(arg)
```

Best

```
def map(func, seq):
    for el in seq:
        func(el)

map(print, ['one', 'two'])
```

Code-as-data, flow-based, object-oriented, and other higher-level programming concepts will be major focus areas in this course

Exceptions

They're not *just* for handling

```
from scipy.stats import binom, norm

def binom_normal(x, n):
    """Approximate 95% bounds of x successes

    :param int x: successes
    :param int n: trials
    """

    # TODO: continuity correction
    if x <= 5 or n - x <= 5:
        raise NotImplementedError('Need exact')
    return norm.interval(.95, x, (x*(1-x/n))**0.5)
```

```
def binom_exact(x, n):
    return binom.interval(.95, n, x/float(n))
```

Use wisely, eg message passing:

```
class ApproxOrCache(object):
    """Approximate a function if possible"""

    def __init__(self, approx, exact):
        self.approx = approx
        self.exact = lru_cache()(exact)

    def __call__(self, *args, **kwargs):
        try:
            return self.approx(*args, **kwargs)
        except NotImplementedError:
            return self.exact(*args, **kwargs)
```

```
get_bounds = ApproxOrCache(
    binom_normal, binom_exact)
get_bounds(50, 100)
```

Mechanics

Grading

	Undergraduate	Graduate
Homework	65%	65%
Exams (2)	30%	20%
Class Participation	5%	5%
Graduate Project	Not Required	10%

Sections

Section sessions begin next week.

Pick one and be consistent.

Attendance is mandatory

Problem Sets

Submit problem sets through GitHub Classroom. You must have a GitHub account.

Pset 0 is due on Friday!

Grading will be:

- Partially automatic
- Include subjective python, git quality
- Include tests and coverage
- Include environment repeatability

Real-world tools, engaged students

GITHUB

GitHub Education helps students, teachers, and schools access the tools and events they need to shape the next generation of software development.

Activate your student developer pack

Github Classroom: special place for us to share private code

ub Student eloper Pack

veloper tools, free for students

GitHub Campus Experts

Training to enrich the technology community at your school

GitHub Field Day

Unconferences for leaders of technical student communities

GitHub Classroom

The GitHub workflow, scaled for the needs of students

GitHub Campus

Teacher training to m GitHub

No description, website, or topics provided.

Edit

Add topics

11 commits | **1** branch | **0** releases | **3** contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gorlins Merge pull request #3 from csci-e-29/feature/final ... Latest commit 7f78f1b 21 days ago

.gitignore	feat(scaffolding): provide interface and scaffolding for answers	21 days ago
Dockerfile	feat(docker): boilerplate and instructions for using a docker python	29 days ago
README.md	feat(scaffolding): provide interface and scaffolding for answers	21 days ago
docker-compose.yml	feat(docker): boilerplate and instructions for using a docker python	29 days ago
fibonacci.py	feat(scaffolding): provide interface and scaffolding for answers	21 days ago
pyramid.py	feat(scaffolding): provide interface and scaffolding for answers	21 days ago
requirements.txt	feat(docker): boilerplate and instructions for using a docker python	29 days ago

Pset 0

Hopefully you have started!

Submit by midnight Friday

Alert TA's if you need early feedback for drop date

README.md

Pset 0

Readings

- Continuous Integration
- A Successful Git Branching Model
- API First Data Science
- Test Driven Dev for Data Science
- CI for Data Science
- Pipfile vs Setup.py
- Freezing Python's Dependency Hell in 2018