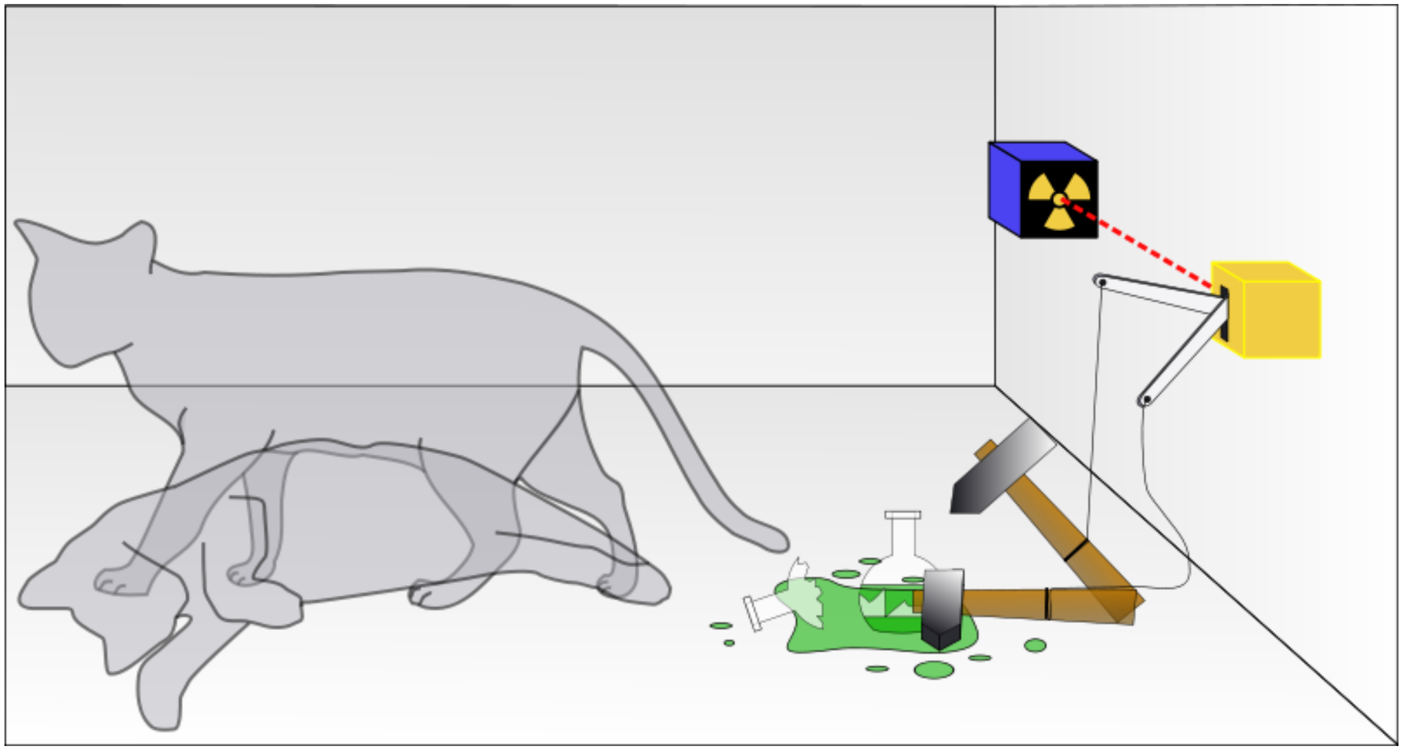
Ashish Mahabal
California Institute of Technology

Best Programming Practices - I

Sponsored by the Jet Propulsion Laboratory and by the Keck Institute for Space Studies, California Institute of Technology

Code Divides the Universe

Treat your coding accordingly



"Schrodingers cat" by Dhatfield – SA 3.0 (Wikimedia)

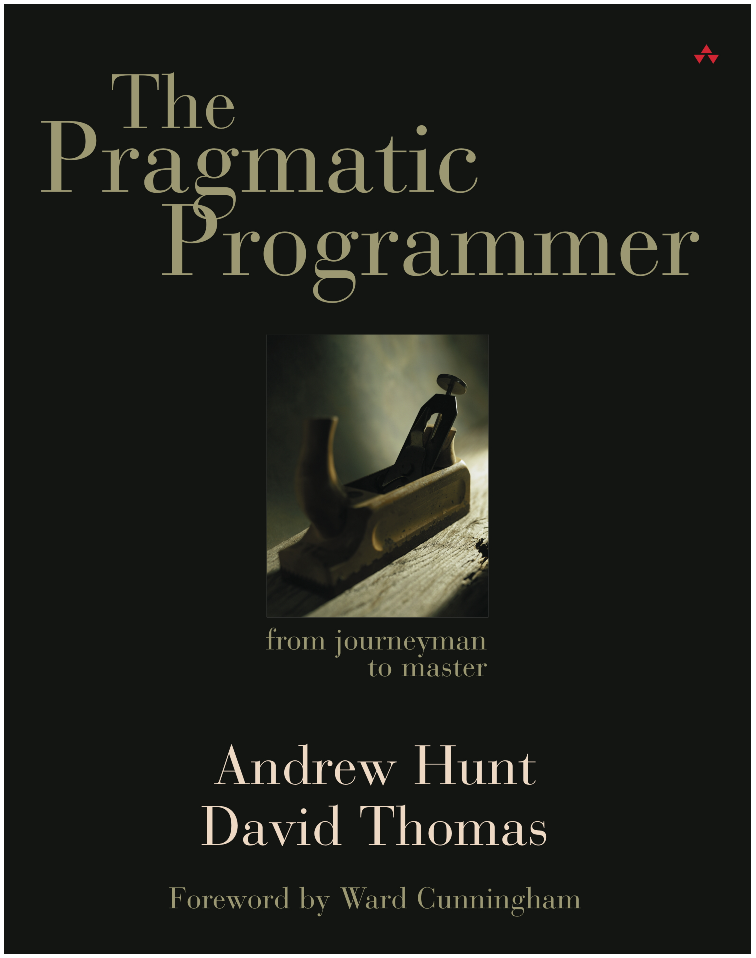
Vision of a program

- Variables
  - their names
- Subroutines
  - their names
  - their functions
- Structure of a program
- Evolution and well being

Only small variations based on tools

The Pragmatic Programmer

By Andrew Hunt and David Thomas



for Python:

When in doubt: import this

Source Code Control

Allow versions to be stored in one place

Allow multiple people to work on a piece of code

Allow access from multiple computers easily

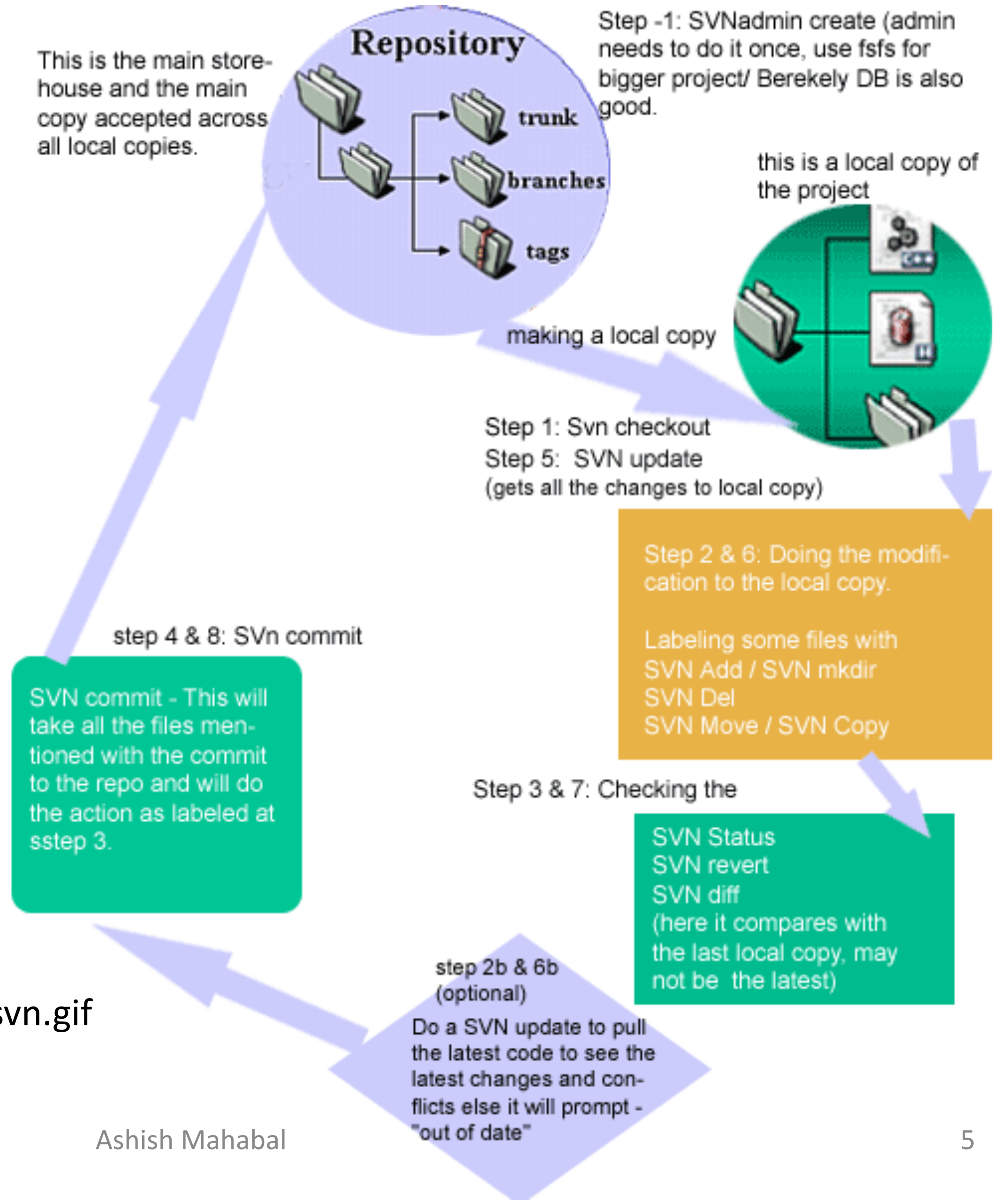
[Concurrent Version Systems (CVS)]

Apache SubVersion (SVN)

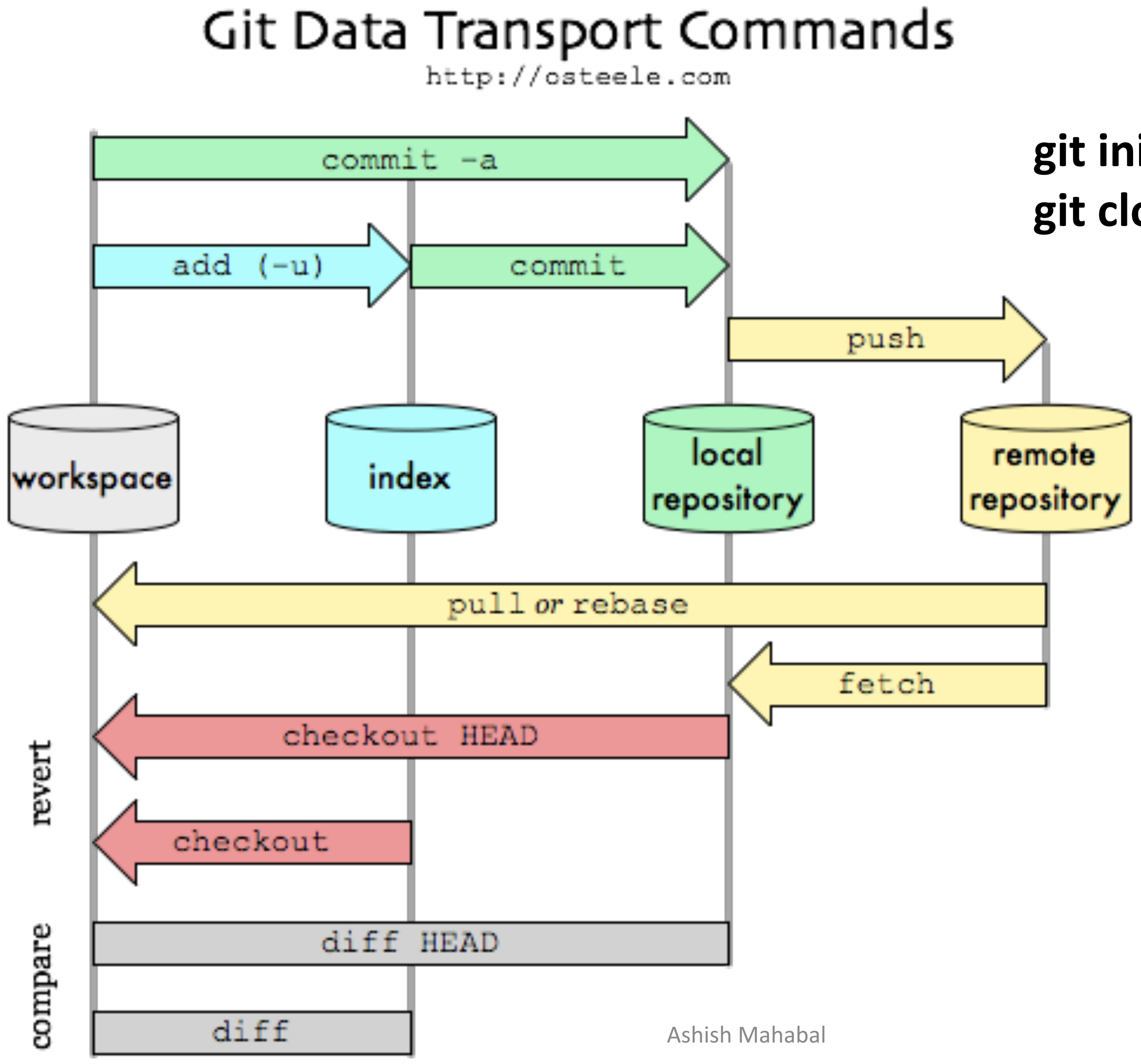
Git

[Mercurial]

- SVN
  - Checkin
  - Checkout
  - Comment
  - Merge


Repository
trunk
branches
tags
This is the main store-house and the main copy accepted across all local copies.
Step -1: SVNadmin create (admin needs to do it once, use fsfs for bigger project/ Berekely DB is also good.
this is a local copy of the project
making a local copy
Step 1: Svn checkout
Step 5: SVN update
(gets all the changes to local copy)
Step 2 & 6: Doing the modification to the local copy.
Labeling some files with
SVN Add / SVN mkdir
SVN Del
SVN Move / SVN Copy
Step 3 & 7: Checking the
SVN Status
SVN revert
SVN diff
(here it compares with the last local copy, may not be the latest)
step 2b & 6b (optional)
Do a SVN update to pull the latest code to see the latest changes and conflicts else it will prompt - "out of date"
step 4 & 8: SVn commit
SVN commit - This will take all the files mentioned with the commit to the repo and will do the action as labeled at sstep 3.

http://img.idealwebtools.com/blog/svn.gif



Git

git init

git clone repo

Online “hubs” that allow versioning

- github
- bitbucket
- google drive
- authorea – collaborative papers
- sharelatex – collaborative latexing

Coding by instinct

- Variable names
  - UpperCamelCase,
  - lowerCamelCase,
  - alllowercase (bad!)
  - period.separated (issues with modules)
  - underscore_separated (possible issues with latex)

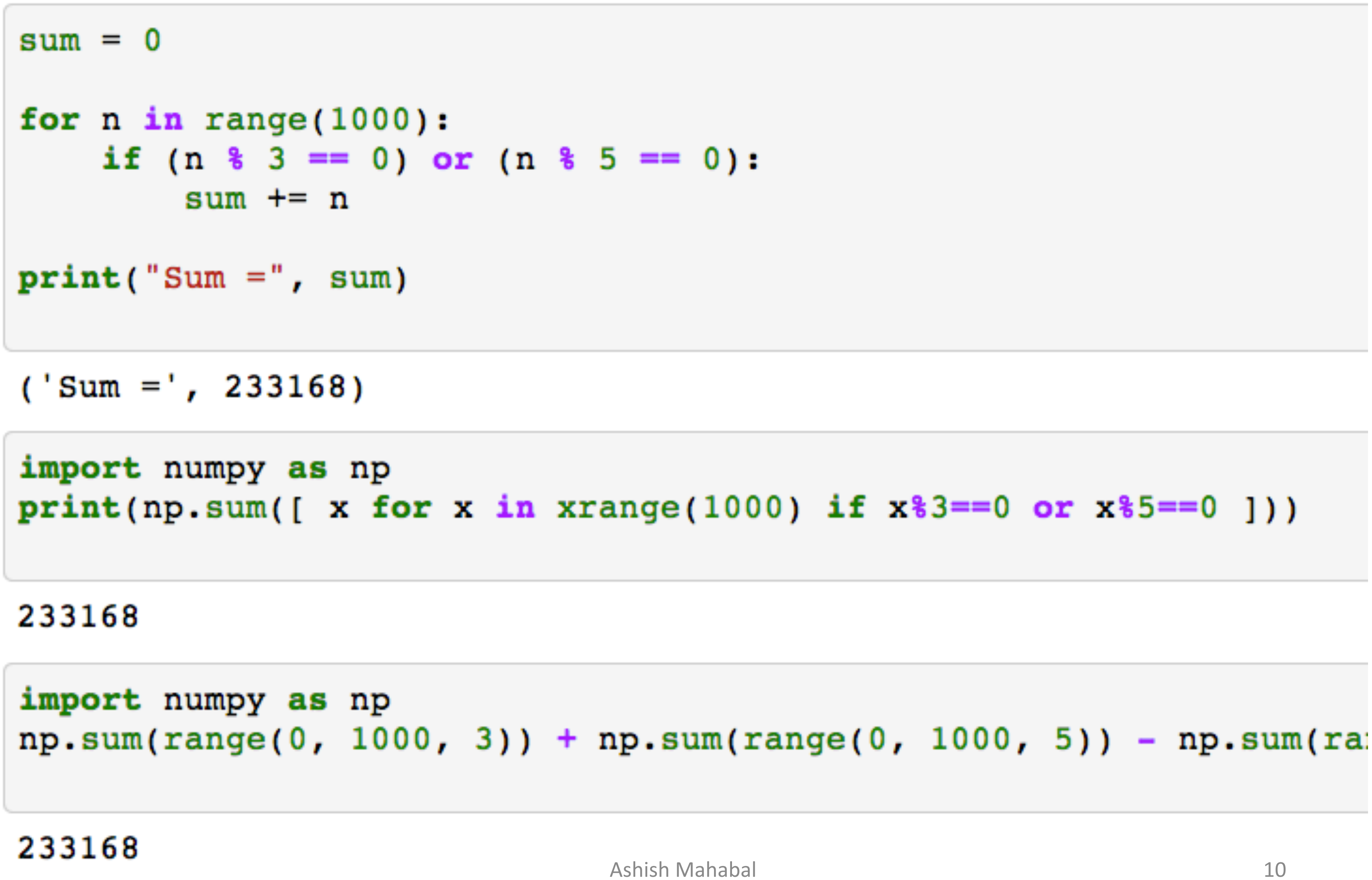
Coding by instinct (loops)

- Types of loops (for, while, …)

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>


for k in {"x": 1, "y": 2}:
     print k
```

Even avoiding explicit loops …



```python
sum = 0

for n in range(1000):
    if (n % 3 == 0) or (n % 5 == 0):
        sum += n

print("Sum =", sum)
```

```
('Sum =', 233168)
```

```python
import numpy as np
print(np.sum([ x for x in xrange(1000) if x%3==0 or x%5==0 ]))
```

```
233168
```

```python
import numpy as np
np.sum(range(0, 1000, 3)) + np.sum(range(0, 1000, 5)) - np.sum(ra
```

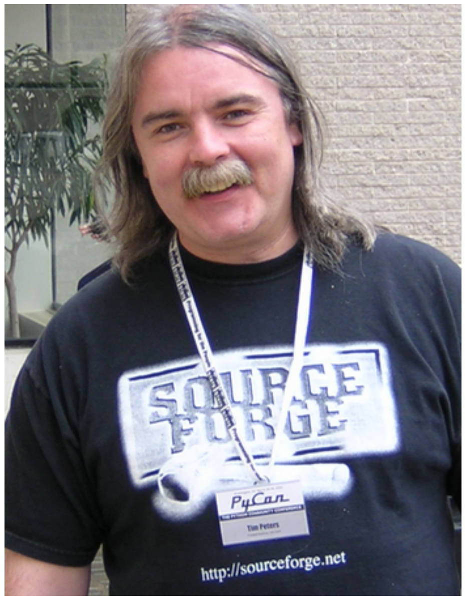
```
233168
```

Coding by instinct

- Variable names
- Types of loops (for, while, …)
- Formatting
  – Indents, brackets, braces, semicolons
- Procedural versus object oriented approach

Conscious and consistent programming style

Zen of Python (PEP 20 – Aug 2004)



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

(a poem by Tim Peters)

Available as "import this"

Modification cycle

Write test

Run and make sure it fails

Checkout

Change, comment, edit readme etc.

Compile

Run: make sure test passes

Checkin

A simple test

```python
#python -m unittest test_module1 test_module2

import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(3), 4)
```

Next time …

- Project requirements, and
- Necessary ingredients