Advanced Topics

May 30, 2017

Poll Results

1) Basic ML Tools (64%)
2) Building a Web App (43%)
3) Idiomatic Python (36%)

Machine Learning and Data Science

Well…

"ML" and "Data Science" are incredibly broad topics

Span computer science, statistics, mathematics, etc

Job titles: "Data Scientist" ≠ "Software Engineer"

We'll do some ML examples, and then see some resources

TL;DR: this section won't teach you data science or ML

… but it can show you some examples in Python

It Probably Exists

TL;DR:

Before You *Really* Do Data Science / ML

Read official tutorials and documentation!

NumPy Quickstart (35 pages)

NumPy Basics (53 pages)

Get familiar with jupyter (iPython notebooks and science)

Read problem-/domain-specific documentation too!

Example: Digit Classification

Credit

MNIST in TensorFlow for Beginners

Example: Exploring NLTK

Credit and Credit

Where to Find Interesting Datasets

Kaggle – hundreds of publicly-accessible datasets for DS

AWS – public data repositories hosted on AWS

www.data.gov – >100,000 government datasets

or… build your own! – log files, APIs, web scraping

Web Applications

Web Applications

Connect your program to the outside world

We've seen Flask, Django(, Twisted)

Assuming cursory background in HTML/CSS/JS

- If not, W3Schools has great tutorials

Easy to deploy Django/Flask on Heroku/AWS

- Django-on-Heroku or Flask-on-Heroku
- Alternatively, use ngrok to expose local ports to the web

Example: Flask Microblog

Credit - Mega-Tutorial through Part IV

Time-Out for Announcements

Final Projects

Due Friday, June 2nd @ midnight

At most one late day

Submit code and writeup on AFS (myth)

Classwide poll to choose presenters

Next Tuesday

Project Presentations

EOQ Activities

Last Class :(

Back to Python!

Honorable Mentions

"In-Depth Machine Learning Tools"

1) Learn Python

2) Foundational Machine Learning Skills

    Take CS229 or CS221! Or just read the course notes =)

3) Learn the Python Libraries

    `numpy` / `scipy` / `matplotlib` / `scikit-learn`

    `tensorflow` / `keras` for machine intelligence

Check out CME 193 and CS 20SI at Stanford!

"Surprising Random Facts about Python"

`'x' in ('x', )` is faster than `'x' == 'x'`. Why?

The Zen of Python is encoded in ROT13

for loops can have an optional else block

`float('inf')` returns a "positive infinity" upper bound

Python has a small-integer cache for -5 to 256

The name "Python" refers to Monty Python

As with CS41, the official docs are full of inside jokes

Idiomatic Python

21 Common Python Style Tricks

We're not talking about PEP8 here…

… though you should still be PEP8-compliant

All about using Python's tools to simplify programming

But… practicality beats purity

"A foolish consistency is the hobgoblin of little minds."

Practicality shouldn't beat purity to a pulp!

An Example

Bad Python

Good Python

1

Swap Two Variables

```
temp = a
a = b
b = temp
```

```
a, b = b, a
```

2

Loop Unpacking

```
for bundle in zip([1,2,3],'abc'):
    num, let = bundle
    print(let * num)

for key in d:
    val = d[key]
    print('{}: {}'.format(key,
                          val))
```

```
for num, let in
zip([1,2,3],'abc'):
    print(let * num)

for key, val in d.items():
    print('{}->{}'.format(key,
                          val))
```

3

Enumerate Iterables

```
for index in range(len(arr)):
    elem = arr[index]
    print(elem)

for index in range(len(arr)):
    elem = array[index]
    print(index, elem)
```

```
for elem in arr:
    print(elem)


for index, elem in enumerate(arr):
    print(index, elem)
```

4

Joining Strings

```
s = ''
for color in colors:
    s += color


s = ''
for color in colors:
    s += color + ', '
s = s[:-2]
```

```
s = ''.join(colors)



s = ', '.join(colors)
```

5

Reduce In-Memory Buffering

```
', '.join([color.upper()
           for color in colors])

map(lambda x: int(x) ** 2,
    [line.strip() for line in
file])

sum([n ** 2 for n in range(1000)])
```

```
', '.join(color.upper()
          for color in colors)

map(lambda x: int(x) ** 2,
    (line.strip() for line in
file))

sum(n ** 2 for n in range(1000))
```

6

Chained Comparison Tests

```
return 0 < x and x < 10
```

```
return 0 < x < 10
```

7

Use `in` Where Possible

```
if d.has_key(key):
    print("Here!")

if x == 1 or x == 2 or x == 3:
    return True

if 'hello'.find('lo') != -1:
    print("Found")
```

```
if key in d:
    print("Here!")

if x in [1, 2, 3]:
    return True

if 'lo' in 'hello':
    print("Found")
```

Boolean Tests

```
if x == True:
    print("Yes")

if len(items) > 0:
    print("Nonempty")

if items != []:
    print("Nonempty")

if x != None:
    print("Something")
```

```
if x:
    print("Yes")

if items:
    print("Nonempty")

if items:
    print("Nonempty")

if x is not None:
    print("Something")
```

9

Use _ for ignored variables

```
for i in range(10):
    x = input("> ")
    print(x[::-1])
```

```
for _ in range(10):
    x = input("> ")
    print(x[::-1])
```

Loop Techniques

```
for i in range(len(colors)):
    color = colors[i]
    name = names[i]
    print(color, name)

for ind in range(len(elems) - 1,
                 -1, -1):
    print(elems[ind])
```

```
for color, name in zip(colors,
                       names):

    print(color, name)

for elem in reversed(elems):
    print(elem)
```

11

Initialize List with Minimum Capacity

```
nones = [None, None, None, None]

two_dim = [[None] * 4] * 5]
```

```
nones = [None] * 4

two_dim = [[None] * 4
           for _ in range(5)]
```

Mutable Default Parameters

```
def foo(n, x=[]):
    x.append(n)
    print(x)



foo(1, [4])  # => [4, 1]
foo(3)  # => [3]
foo(3)  # => [3, 3]
foo(3)  # => [3, 3, 3]
```

```
def foo(n, x=None):
    if x is None:
        x = []
    x.append(n)
    print(x)

foo(1, [4])  # => [4, 1]
foo(3)  # => [3]
foo(3)  # => [3, 3]
foo(3)  # => [3, 3, 3]
```

Format Strings (for now)

```
print("Hi %s, you have %i texts"
      % ("Sam", 6))

print("Hi %(name)s, you have
      %(num)i texts"
      % {'name': Sam', 'num': 6})
```

```
print("Hi {}, you have {} texts"
      .format("Sam", 6))

print("Hi {name}, you have {num}
texts".format(name="Sam", num=6))
```

Comprehensions

```
out = []
for word in lex:
    if word.endswith('py'):
        out.append(word[:-2])

lengths = set()
for word in lex:
    lengths.add(len(word))
```

```
out = [word[:-2] for word in lex
       if word.endswith('py')]



lengths = {len(word) for word in
lex}
```

15

Use `collections` and `itertools`

```
d = {}
for word in lex:
    if len(word) not in d:
        d[len(word)] = []
    d[len(word)].append(word)
```

```
d = collections.defaultdict(list)
for word in lex:
    d[len(word)].append(word)
```

16

Use Context Managers

```
f = open('path/to/file')
raw = f.read()
print(1/0)
f.close()


lock = threading.Lock()
lock.acquire()
try:
    print(1/0)
finally:
    lock.release()
```

```
with open('path/to/file') as f:
    raw = f.read()
    print(1/0)



with threading.Lock():
    print(1/0)
```

EAFP > LBYL

```
def safe_div(m, n):
    if n == 0:
        print("Can't divide by 0")
        return None
    return m / n
```

```
def safe_div(m, n):
    try:
        return m / n
    except ZeroDivisionError:
        print("Can't divide by 0")
        return None
```

Avoid using Catch-Alls

```
while True:
    try:
        n = int(input("> "))
    except:
        print("Invalid input.")
    else:
        return n ** 2
```

```
while True:
    try:
        n = int(input("> "))
    except ValueError:
        print("Invalid input.")
    else:
        return n ** 2
```

19

Use Custom Exceptions Abundantly

```
if not self.available_cheeses:
    raise ValueError("No cheese!")
```

```
class NoCheeseError(ValueError):
    pass

if not self.available_cheeses:
    raise NoCheeseError("I'm
                         afraid
we're right out, sir.")
```

Magic Methods for Custom Classes

```
class Vector():
    def __init__(self, elems):
        self.elems = elems

    def size(self):
        return len(self.elems)

v = Vector([1,2])
len(v)  # => fails
```

```
class Vector():
    def __init__(self, elems):
        self.elems = elms

    def __len__(self):
        return len(self.elems)

v = Vector([1,2])
len(v)  # => succeeds
```

Using __name__ for scripts

```
def stall():
    time.sleep(10)

stall()
```

```
def stall():
    time.sleep(10)

if __name__ == '__main__':
    stall()
```

Specific Advice

Use keyword arguments for optional, tunable parameters

Utilize functional programming concepts to simplify code

Employ decorators to factor out administrative logic

Simplify resource management with context managers

General Advice

Don't reinvent the wheel!

- Check standard library and PyPI for existing solutions.
- Search StackOverflow and Google for helpful tips!

Know all operations on builtin types + common one-liners

One line of code shouldn't be more than one English line

"We are all responsible users"

Zen of Python

```
>>> import this

The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

>>> import this

```
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Programmers are more important than programs

Closing Remarks