CS 111: Program Design I
Lecture 13: How to: Pandas & Supreme Court DB, Sup. Ct.

Robert H. Sloan & Richard Warner
University of Illinois at Chicago
Oct. 10, 2019

UIC

Today

- Answer to student question: map and lists
- Few clicker questions reviewing points relating to our data analytics project
- Using pandas module to analyze Supreme Court DB: didactic how-to
- Intro to Supreme Court

You asked for it

DOING SOMETHING TO EACH ITEM OF LIST

You asked

- How do we do something to each item of list?
- map() does that (almost)
- map(fun, ls) returns new list made by applying fun to each item in ls
  - Does *not* alter ls

Mapping with Python... (*not* in book)

```
def dbl(x):
    """returns 2 * x"""
    return 2 * x

>>> map(dbl, [0, 1, 2, 3, 4, 5])
[0, 2, 4, 6, 8, 10]
```

```
def evens(n):
    list_n = range(n)
    doubled = map(dbl, list_n)
    return doubled
```

Docstring would help

…or alternatively

```
def evens(n):
    return map(dbl, range(n))
```

`Mapping` with Python... Your turn

```
def plusone(x):
  """returns x + 1"""
  return x + 1

>>> map(plusone, [1, 2, 3, 4])
```

A. [1, 2, 3, 4, 5]
B. 11
C. [1, 2, 3, 4, 1]
D. [2, 3, 4, 5]
E. None of the above
F. No clue ☺

Building a list: map often helps

- Common to want to build up a list by doing something to a simpler list
- E.g.,

```
ls = [1, 2, 3, 4, 5]
squares = []
for n in ls:
    squares.append(n**2)
```

Building a list: map often helps

- Common to want to build up a list by doing something to a simpler list
- E.g., list of squares of integers:

```
ls = [1, 2, 3, 4, 5]
squares = []
for n in ls:
    squares.append(n**2)
```

Easier with map

```
def square(x):
    return x**2

ls = [1, 2, 3, 4, 5]
squares = map(square, ls)
```

vs.

```
ls = [1, 2, 3, 4, 5]
squares = []
for n in ls:
    squares.append(n**2)
```

- Simpler (to write & to understand!), nicer
- (Example of "functional" style of programming)

REVIEW OF LAST TIME STUFF WE'RE ABOUT TO USE

Using modules

Which statement should be at the top of my code if I need to use Python's antigravity module?

A. use antigravity
B. include antigravity
C. #include antigravity
D. import antigravity
E. allow antigravity

Encodings

- The ASCII character set has 95 specified printing characters (including, a-z, A-Z, 0-9, space, some punctuation), and 3 to 33 nonprinting characters including \n
- How many bits are needed to have enough distinct patterns for all ASCII characters

A. < 7
B. 7
C. 8
D. > 8

With 8 bits

- Can encode 256 characters: Way more than ASCII; way less than all of Unicode
- Unicode's 2019 most common encoding (UTF-8) uses 1–4 bytes per character; and uses the same 1 byte as ASCII for the ASCII characters
- 94% of web today is Unicode encoded as UTF-8 (counting ASCII as part of that)