CS1 Lecture 22 Mar. 8, 2017

- HW5 available due this Friday, 5pm

CS1 Lecture 22 Mar. 8, 2017

- HW5 available due this Friday, 5pm
- HW4 scores later today

LAST TIME

- Tuples
- Zip (and a bit on iterators)
- Use of key and reverse arguments to sort/sorted
  - Incl. lambda, anonymous one-line functions

TODAY

- HW5 Q1
- Optional/default/keyword arguments
- Ch 19: conditional expressions and list comprehensions

One year ago

AlphaGo beat Lee Sedol

- Lee Sedol, from Korea, is a 9-dan professional. Considered by many to be the top player in the world over the past 10 years, and is number 2 all time in international match victories.

Go considered a huge challenge for computers - “the only game left above chess”

- Number of possible games/moves **enormous**, far more than chess. 391 legal first moves, and a game might last 150 moves. Number of legal board states for a game of Go larger than numbers of atoms in the universe.” Approximate number of possible games 10**360 (or between 10**10**48 and 10**10**171, depending on your measure) – far too many to do brute-force search
- AlphaGO had beaten a 2-dan champion earlier, the first time a program defeated a top professional in Go.
- The match last night was the first of five games. Many experts were surprised by the win.
- AlphaGO uses “deep learning” – advanced computer science techniques combining Monte Carlo tree search and neural networks
- AlphaGO won the match 4-1, again a surprise.
- Recent developments – late last year and so far this year – on-line version beating the worlds’s top players

Last time – sorting with key and reverse

sorted (and sort) have two useful optional arguments:

1) key: a little function that is applied to item to generate key to use to sort

```
>>> tl = [(‘free’, 23), (‘you’, 50), (‘go’, 10), (‘zoo’, 1)]
>>>sorted(tl, key = item1)
>>>sorted(tl, key = lambda item: item[1])
```

if item1 function exists

but don’t need to write separate function. ‘lambda’ allows you to define (anonymous function) anywhere

```
[('zoo', 1), ('go', 10), ('free', 23), ('you', 50)]
```

2) reverse: True if you want list from largest to smallest instead of default of smallest to largest

```
>>> sorted(tl, key = lambda item: item[1], reverse = True)
[('you', 50), ('free', 23), ('go', 10), ('zoo', 1)]
```

HW 5 Q1

- Don’t print giant dictionaries or lists! Use print statements on data from smaller test files. Or print parts of the big lists/dictionaries.
- Last time, discussed how to use sorting to extract items with highest counts from the two dictionaries you create..
  1. Saw that sorted(dict) doesn’t quite give us what we needed
  2. Saw instead that we can usefully sort list of tuples
  3. So … can you make a list of tuples [… (word, count) …] from a dictionary? Using zip, dict.values(), dict.keys() is one approach …

HW5 Q1 reminders

1. File has some non-Ascii characters.

   use: open(fileName, encoding = 'utf-8’)

2. To break line into tokens – individual elements of a line, learn how to use string **split**

   ```
   for line in fileStream:
       lineAsList = line.split()
   ```

3. get rid of extra stuff “…cool!?” learn how to use string **strip** (and/or lstrip, rstrip, translate)

Default argument values and keyword arguments to functions

Note: although it is used a few places in the book, it’s not discussed much. See official Python documentation for more info.

With sort/sorted, we saw a way of calling a function that I never really explained before:

```
tl = [(‘free’, 23), (‘you’, 50), (‘go’, 10), (‘zoo’, 1)]
sorted(tl, key = lambda item: item[1], reverse = True)
```

The ‘key = …’ and ‘reverse = …’ are called keyword arguments and are very convenient.

Default argument values and keyword arguments to functions

```
def foo(a, b = 0, c = True):
    if c == True:
        return a + b
    else:
        return a – b
```

Parameters b and c are optional! They have default values so you don’t need to provide more than one argument when calling foo.

```
>>> foo(3)
 3
```

Default argument values and keyword arguments to functions

```
def foo(a, b = 0, c = True):
    if c == True:
        return a + b
    else:
        return a – b
```

```
>>> foo(3,1)
4
>>> foo(3,1,False)
2
```

Default argument values and keyword arguments to functions

```
def foo(a, b = 0, c = True):
	if c == True:
		return a + b
	else:
		return a – 1
```

It sometimes makes code easier to read to use the parameter name when making the function call.

```
>>> foo(3, b=1)
??? 4
>>> foo(3, b=1,c=False)
??? 2
>>> foo(3, c=False, b=1)
??? 2
>>> foo(b=1, c=False, 3)
??? Error!
>>> foo(c=False, b=1, a = 3)
??? 2
```

Unnamed arguments must come first, before keyword arguments!

And BE CAREFUL. Mutable default arguments are evaluated once, at function definition (different than some other languages). See bar in lec22.py

Chapter 19 (“The Goodies”): Conditional expressions

You should be comfortable with if/elif/else statements by now. Some can be very large and complicated.

But some are of a simple form where, e.g., both the if and else part set the value of the same variable. E.g.

```
if x > 0:
    s = 1
else:
    s = -1
```

Python provides a shorthand for this called **conditional expressions:**

```
>>> x = 3
>>> s = 1 if x > 0 else -1
>>> s
??? 1
```

Conditional expressions

Can also be used in place of an if/else where both return some value

```
def oddEven(n):
    if n%2 == 0:
        return ‘odd’
    else:
        return ‘even’
```

```
def oddEven(n):
    return ‘odd’ if n%2==0 else ‘even’
```

Is this better?

It’s *shorter* but

I don’t find it as readable

Chapter 19: list comprehensions

Python provides another shorthand - **list comprehensions** – concise expressions for constructing lists.

Consider common pattern:

```
    result = []
    for item in someList:
        result.append(someFunc(item))
```

Can do this is one line with list comprehension:

```
    result = [someFunc(item) for item in someList]
```

“apply someFunc to each item in someList and gather the results in a new list”

Chapter 19: list comprehensions

Examples:

```
>>> [i * i for i in range(5)]
???
[0, 1, 4, 9, 16]
>>> [s.lower() for s in [“Hi”, “Bye”]]
???
['hi', 'bye']
```

Can also use an if:

```
>>> [num for num in [1, -2, 3, -4, 5] if num > 0]
???
[1, 3, 5]
```

Chapter 19: list comprehensions

Can also have more than one ‘for’ in a comprehension

```
>>> [(i,j) for i in range(10) for j in range(5)]
???
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (2, 0), (2, 1), (2, 2), (2,
3), (2, 4), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 0), (5, 1),
(5, 2), (5, 3), (5, 4), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (8,
0), (8, 1), (8, 2), (8, 3), (8, 4), (9, 0), (9, 1), (9, 2), (9, 3), (9, 4)]
>>> [(i,j) for i in range(10) for j in range(i)]
???
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2), (4, 0), (4, 1), (4, 2), (4, 3), (5, 0), (5, 1),
(5, 2), (5, 3), (5, 4), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (7, 0), (7, 1), (7, 2), (7,
3), (7, 4), (7, 5), (7, 6), (8, 0), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (9, 0),
(9, 1), (9, 2), (9, 3), (9, 4), (9, 5), (9, 6), (9, 7), (9, 8)]
```

Chapter 19: list comprehensions

Example: matrices are common in science and engineering and are often depicted as a table having n rows and m columns. Below is a matrix with 3 rows and 5 columns:

```
1 12 3 4 5
5 -1 4 1 1
0 -3 4 0 9
```

It's easy to represent a matrix in Python using a list of sublists, where each sublist represents one row of the matrix. Thus the matrix above would be represented as: [[1, 12, 3, 4, 5], [5, -1, 4, 1, 1], [0, -3, 4, 0, 9]]

If A, B are matrices **CAN** multiply them using list comprehension:

```
def matrixMult(A, B):
   return [ [sum([ A[i][j] * B[j][k] for j in range(len(B))]) for k in
                  range(len(B[0])) ] for i in range(len(A))]

def mmatrixMult2(A, B):
   # Multiply row by (transposed w/zip) col
   return([[ sum([ a*b for (a,b) in zip(row,col) ]) for col in zip(*B) ] for row in A ])
```

I don't do this – for me, too concise/dense to be easily readable and clear. I use comprehensions for small simple things.

Chapter 19: dictionary comprehensions and generator expressions

There are also dictionary comprehensions:

```
 { i : i*i for i in range(4) }
```

And, there are things that look like comprehensions but are called generator expressions:

```
>>> genSq = (i*i for i in range(4))
```

yields a generator object, which is an iterator, so you can use it with next(…) like we did with zip object.

(Dictionary comprehensions and generator expressions are not required for this class; you don’t need to know them for exams/homework.)

Note: HW5 Q3 should use list comprehension, not a generator.

Chapter 19: any and all

Two other sometimes-useful Python functions:

any(someSequence): returns True is one or more elements in the given sequence are True

all(someSequence): returns True is all elements in the given sequence are True

Common to use any/all with list comprehensions.

List comprehension that tests if at least one number in a list is even?

```
any( [ (i%2 == 0) for i in L] )
```

if all strings in a list contain an ‘e’

```
all( [ ‘e’ in s for s in l ] )
```

Next Time

- Grades so far
- GUI and image manipulation, examples of things we will do in the rest of the course