CS1210 Lecture 14 Sep. 24, 2021

- Quiz 1 scores will take a few more days - sorry for the slow feedback
- HW3 due tonight
- HW4 will be released Monday

Last time

- Related to DS4: largest anagram set
- Finish lists
  - More on aliasing
  - is operator and object identify (vs ==)
  - Lists as arguments to functions
- review of HW3, Q1 hints

Today

- Quick introduction to conditional expressions and list comprehensions (Ch 10.22) - very useful but you are not required to know list comprehensions
- Start dictionaries

(Last time) Advice/comments on functions

- Some functions compute something and return a value without *side effects*. That is, they do any output and don't change the values of any objects that exist outside of the execution of that function.
- Other functions do have side effects. They either print something (or affect GUI elements) or change values of objects that exist outside the function execution. Such functions often don't return anything. And such functions can maybe helpfully be thought of as commands.

Thoughts on this function?

```
# Given a list of 1 or more numbers or strings, return the smallest
# and largest items.
#
def getSmallestAndBiggest(listOfStuff):
    listOfStuff.sort()
    return (listOfStuff[0], listOfStuff[-1])
```

Functions should not modify their input unless the function specification explicitly says to do so!

```
# Given a list of 1 or more numbers or
# strings, return the smallest and largest
# items.
#
def getSmallestAndBiggest(listOfStuff):
    sortedCopyOfList = sorted(listOfStuff)
    return (sortedCopyOfList[0], sortedCopyOfList[-1])
```

```
# Given a list of 1 or more numbers or
# strings, sort the list and return the smallest and
# largest items.
#
def sortAndGetSmallestAndBiggest(listOfStuff):
    listOfStuff.sort()
    return (listOfStuff[0], listOfStuff[-1])
```

Conditional expressions (common but not in text?)

not required for exams but you should understand if you see them

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

```
def oddEven(n):
    if n%2 == 1:
        return 'odd'
    else:
        return 'even'
```

```
def oddEven(n):
    return 'odd' if n%2==1 else 'even'
```

Is this better?

It's *shorter* but I don't find it as readable

They **are** clear and useful sometimes.

lec14comprehensions.py

Chapter 10.22: 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”

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]
```

lec14comprehensions.py

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)]
```

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.

dictionary (next topic) 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.

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

Chapter 12: Dictionaries

- Python supports the extremely useful **dictionary** 'dict' type in Python
- Dictionaries are:
  - collections of key – value pairs
- Similar to but importantly different from lists
  - could think of lists as *ordered* collection of key-value pairs, where the keys are integers 0, 1, 2, …
  - with dictionaries, the collection is *unordered* but the interesting thing is that the *keys can be any immutable values*
  - *E.g.* create dictionary numlegs

    >>> numlegs = { 'frog': 4, 'human': 2, 'ant':6, 'dog':4}

Ended Lecture 14 here due to internet instability - will review dictionaries from the beginning on Monday, Sept. 27

Dictionaries

- create with { k1:v1, k2:v2, …}
- empty dictionary: {}
- retrieve value: dict[key]
- modify (or insert) value for key: dict[key]=value

- one important feature of dictionaries is that they provide *very fast* access (we might discuss *how* later in term) to values associated with keys despite being more flexible (not restricted to integer keys, etc.) than lists (demo: dicttest.py for speed comparison with lists)

Dictionary operations

- len(d)
- d.keys()
- d.values()
- k in d
- del d[k]
- for key in dict:
- d.get(key, defaultVal) when you don't want possible KeyError for d[key]

But note: no slice – d[key1:key2] doesn't make sense

Looping over dictionaries with for

If we have a dictionary, d, of names as keys and ages as values, we can compute averages as follows:

```
averageAge = 0
sumOfAges = 0
for nameKey in d:
   sumOfAges += d[nameKey]
if sumOfAges != 0:
   averageAge = sumOfAges / len(d)
print("The average age is: ", averageAge)
```

lec14.py

A dictionary example

- Text file with info about people – name, birth year, favorite color, weight, home city, home country
- Read and store in dictionary
  - Name as key
  - Subdictionary (and sub-sub-dictionary) for other properties

```
{'birthyear': 1980,
 'favcolor': 'red',
 ...,
 'home': {'city': 'Tokyo', 'country': 'Japan'}
}
```

- Add simple password handling, storing "hash" in dict

Files: ppldata.py, people.text

Related news

- 2015 Turing Award winners: https://www.theguardian.com/science/2016/mar/01/turing-award-whitfield-diffie-martin-hellman-online-commerce
- http://amturing.acm.org/byyear.cfm
- Remember printFirstNPrimes problem? Finding prime factors of big numbers is super Important for cryptography. Internet security depends hugely on the fact that there is no known way to find factors of very large numbers quickly

A few little exercises

- Given a list of numbers, find the pair with greatest difference
- Given a list of numbers find the pair with smallest difference
- Given a list of numbers and a target number (call it k), find two numbers (if they exist) in the list that sum to k

lec14exercises.py has solutions for first two. Has slow (and not completely correct) solution for third one. Can you think of a much faster solution using dictionaries?

Small variants of/questions about third problem

4. Suppose:
   - No dictionaries allowed/available
   - Numbers in lists are know to have limited magnitude. E.g. all numbers between 0 and 10000

   Fast solution?
5. Modify findKPairFast to provide indices/location of found pair in list
6. Question: if we generate a list of, say, 10,000 random numbers between -1,000,000 and 1,000,000 how likely is it that the list contains a pair that sums to k for any k in, say, 0…999?

lec14exercisesB.py

(next time)

Next Time

- DS5 and HW4
- tuples and tuple assignment (10.26-28)
- default/optional and keyword arguments to function