1 Tuples

Exercise 1.1: Swapping two values

Suppose you have two variables: `a` and `b`. Now you want to set `a` equal to the value of `b` and at the same time set `b` equal to the value of `a`.

The following obviously does not work

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

so in some languages, you need to define a third variable like this

```
t = a
a = b
b = t
```

However, in Python you don't need to do this. How can you swap `a` and `b` in one line?

Exercise 1.2: Zip

Suppose we have two lists, `x` and `y` that give the x and y coordinates of a set of points. Create a list with the coordinates (x,y) as a tuple. Hint: Find out about the `zip` function.

You have decided that actually, you need the two seperate lists, but unfortunately, you have thrown them away. How can we use zip to *unzip* the list of tuples to get two lists again?

Exercise 1.3: Distances

Suppose we have two vectors, `x` and `y`, stored as tuples with n elements. Implement functions that compute the l_1 and l_2 distances between `x` and `y`. Note that n is not explicitly given.

2 Dictionaries

Exercise 2.1: Printing a dictionary

Write a function that prints key-value pairs of a dictionary.

Exercise 2.2: Histogram

Write a function that takes a list, and returns a dictionary with keys the elements of the list and as value the number of occurances of that element in the list.

After you are done, look up 'python collections counter' in Google. Could you use a counter instead?

Exercise 2.3: Get method

Dictionaries have a `get` method, which takes a key and a default value. If the key is in the dictionary, it returns the value, otherwise, it returns the default value.

Rewrite your code from the previous problem to make use of this `get` method.

Exercise 2.4: Vector functions

Let's implement some vector functions. There are two types of vectors, normal or dense vectors, which we can represent using lists. For sparse vectors, where many of the elements are zero, this is inefficient. Instead, we use a dictionary with keys the indices of non-zero values, and then the value corresponding to the key is the value of the vector at that index. Hence, the vector $[1, 2, 4]$ can be stored as a list: `[1, 2, 4]` or as a dictionary {`0:1, 1: 2, 2: 4`}.

(a) Write a function that adds two (dense) vectors

(b) Write a function that multiplies (i.e. inner product) two (dense) vectors

(c) Write a function that adds two sparse vectors

(d) Write a function that multiplies two sparse vectors

(e) Write a function that adds a sparse vector and a dense vector

(f) Write a function that multiplies a sparse vector and a dense vector

Exercise 2.5: Reverse look-up

Dictionaries are made to look up values by keys. Suppose however, we want to find the key that is associated with some value. Write a function that takes a dictionary and a value, and returns the key associated with this value.

What challenges do you face? How would you deal with those challenges?

3 Strings

Exercise 3.1: Palindrome

Write a function that takes a string as input and returns `True` if the string is a palindrome, and `False` otherwise. Note that a palindrome ia word that reads the same backwards, such as *radar*.

Exercise 3.2: Tokenizer

In Natural Language Processing, a key step to analyzing text is *tokenization* – the process of breaking up text into words and symbols that are atomic units (check out the wikipedia page).

In this exercise, you'll write a function called `tokenize(text)` which will take as input a `string`, and return a `list` with each element a lowercase version of the word / punctuation found in the original text. As an example:

```
text = 'Luke is in ICME, and he likes machine learning.'
print tokenize(text)
# ['luke', 'is', 'in', 'icme', ',', 'and', 'he', 'likes', 'machine', 'learning', '.']
```

Notice the result is a list with no extra whitespace and all lower case. We'll make this simple, and ask that conjunctions like `don't` be converted into `['don', ''', 't']`.

4 File I/O

Exercise 4.1: Open a file

Write a function that opens a file (input: filename), and prints the file line by line.

Exercise 4.2: Wordcount

On the course website you can find a text file containing the complete works of William Shapespeare.

(a) Find the 20 most common words

(b) How many unique words are used?

(c) How many words are used at least 5 times?

(d) Write the 200 most common words, and their counts, to a file.

Exercise 4.3: Sum of lists

Before you start coding, please read the entire problem.

(a) Data generation

Write a function that takes three integers, n, a and b and a filename and writes to the file a list with n random integers between a and b.

(b) Reading the data

Write a function that can read the files as generated above and return the values.

(c) Sum problem

Write a function that given two filenames (pointing to files as generated by the above function) and an integer k, finds all u and v such that $u + v = k$, and u is an element of the first list and v is a member of the second list.

(d) Testing

Test your functions by generating 2 files with $n = 2000$, $a = 1$, $b = 10000$ and $k = 5000$ and $k = 12000$.

(e) Bonus: Efficiency

If you are up to a challenge, write a function that solves the sum problem with the restriction that you can only go over every number in the both lists once.

5 Classes

Exercise 5.1: Rational numbers

In this problem, we will write a class that can represent rational numbers, i.e. fractions $\frac{p}{q}$.

(a) Create a class `Rational` which is initialized by two integers, p and q, the nominator and denominator

(b) Add a method to print the rational number as p/q (the `__str__` or `__repr__` method is useful).

(c) We would like to represent $\frac{10}{20}$ by $\frac{1}{2}$ instead, hence write a function that computes the greatest common divisor, and ensure that every rational number is simplified

(d) Add a method so that we can add two rational numbers with `r1 + r2`, here the `__add__()` method is useful.

(e) Add a method to subtract two rational numbers. (`__sub__`)

(f) Add a method to multiply two rational numbers. (`__mul__`)

(g) Add a method to divide two rational numbers. (`__div__`)

(h) Add a method that compares whether two rational numbers are equal.

(i) Add a method to convert the rational number to a floating point (the `__float__()` method may be handy).

(j) Add any more functionality that you think is useful but I failed to mention.

Exercise 5.2: Sparse and dense vectors

In exercise 2.4 you implemented functions for sparse and dense vector multiplications using lists and dictionaries. However, this is a bit clumsy to use in practice. Really, we would like to represent sparse and dense vectors as classes, this way we can overload operators such as `+` (`__add__`) and get sensible output. For example, using `+` on two dense vectors implemented as lists would append the second vector to the first, instead of adding the two together.

Implement sparse and dense vectors. Both classes should have the following capabilities:

(a) Print vector

(b) Add two vectors (both if other is dense and sparse)

(c) Multiply two vectors (both if other is dense and sparse)

Do re-use your code from the previous exercise.

Hint: `isinstance()` might be useful.

Exercise 5.3: Implementing the set class
Write a class `mySet` that has the same basic functionality as the Python `set` data structure. Base your implementation on a dictionary.

Exercise 5.4: Binary search tree
In this exercise, we will implement a binary search tree. See http://en.wikipedia.org/wiki/Binary_search_tree for an explanation.

(a) Define a class `Node`, and write the constructor, which takes one argument, `value`, and initializes the left and right children to None.

(b) Write a function to print the tree.

(c) Write a function that inserts a new value in the tree at the right location.

(d) Write a function that looks up a value in the tree.

(e) Write a function that removes a value from the tree.

Exercise 5.5: Ordinary least squares
Our goal in this exercise is to write our own least-squares solver to solve regression problems:

$$\arg\min_{\beta} \|y - X\beta\|_2$$

See for example statsmodels `ols` or `LinearRegression`. While one can, and should, use written solvers, it's a good practice exercise.

(a) Setup an OLS class with fit and predict methods, to be coded later

(b) Write the fit method using numpy's or scipy's linear algebra module.

(c) Now write the predict function, that predicts y_n given new X_n.

(d) Add a function that summarizes the model

(e) (Optional) Use Patsy and Pandas to support DataFrames and formulas, similar to `R`.