Physics Without Frontiers

The Abdus Salam
International Centre
for Theoretical Physics

Practical Programming
in Python

Inspired by 'Practical Programming' by Paul Gries, Jennifer Campbell, Jason Montojo

Lecture 11: Summary & Exercises
More Collection Types

Sets, Tuples, Dictionaries, What to use when

> *"All complexities should, if possible, be buried out of sight."*
> – David J. Wheeler

Lecture 11: Summary

In this lecture you learned the following:

- Sets are used in Python to store unordered collections of unique values. They support the same operations as sets in mathematics.
- Tuples are another kind of Python sequence. Tuples are ordered sequences like lists, except they are immutable.
- Dictionaries are used to store unordered collections of key/value pairs. The keys must be immutable, but the values need not be.
- Looking things up in sets and dictionaries is much faster than searching through lists. If you have a program that is doing the latter, consider changing your choice of data structures.

Lecture 11: Exercises

When writing code, only use Python concepts that have been introduced in the lectures already.

Exercise 1:

Write a function called `find_dups` that takes a list of integers as its input argument and returns a set of those integers that occur two or more times in the list.

Exercise 2:

Python's `set` objects have a method called `pop` that removes and returns an arbitrary element from the `set`. If the `set` `gerbils` contains five cuddly little animals, for example, calling `gerbils.pop()` five times will return those animals one by one, leaving the `set` empty at the end. Use this to write a function called `mating_pairs` that takes two equal-sized `set`s called `males` and `females` as input and returns a `set` of pairs; each pair must be a `tuple` containing one male and one female. (The elements of males and females may be strings containing gerbil names or gerbil ID numbers – your function must work with both.)

Exercise 3:

The PDB file format is often used to store information about molecules. A PDB file may contain zero or more lines that begin with the word `AUTHOR` (which may be in uppercase, lowercase, or mixed case), followed by spaces or tabs, followed by the name of the person who created the file. Write a function that takes a list of filenames as an input argument and returns the set of all author names found in those files.

Exercise 4:

The keys in a dictionary are guaranteed to be unique, but the values are not. Write a function called `count_values` that takes a single dictionary as an argument and returns the number of distinct values it contains. Given the input input below , for example, it should return 2.

```
{'red': 1, 'green': 1, 'blue': 2}
```

Exercise 5:

After doing a series of experiments, you have compiled a dictionary showing the probability of detecting certain kinds of subatomic particles. The particles' names are the dictionary's keys, and the probabilities are the values:

```
{'neutron': 0.55, 'proton': 0.21, 'meson': 0.03,
    'muon': 0.07, 'neutrino': 0.14}
```

Write a function that takes a single dictionary of this kind as input and returns the particle that is least likely to be observed. Given the dictionary shown earlier, for example, the function would return 'meson'.

Exercise 6:

Write a function called `count_duplicates` that takes a dictionary as an argument and returns the number of values that appear two or more times.

Exercise 7:

A balanced color is one whose red, green, and blue values add up to 1.0. Write a function called `is_balanced` that takes a dictionary whose keys are `'R'`, `'G'`, and `'B'` and whose values are between 0 and 1 as input and returns `True` if they represent a balanced color.

Exercise 8:

Write a function called `dict_intersect` that takes two dictionaries as arguments and returns a dictionary that contains only the key/value pairs found in both of the original dictionaries.

Exercise 9:

Programmers sometimes use a dictionary of dictionaries as a simple database. For example, to keep track of information about famous scientists, you might have a dictionary where the keys are strings and the values are dictionaries, like this:

```
{
    'jgoodall'  : {'surname'  : 'Goodall',
        'forename' : 'Jane',
        'born'     : 1934,
        'died'     : None,
        'notes'    : 'primate researcher',
        'author'   : ['In the Shadow of Man',
        'The Chimpanzees of Gombe']},
    'rfranklin' : {'surname'  : 'Franklin',
        'forename' : 'Rosalind',
        'born'     : 1920,
        'died'     : 1957,
        'notes'    : 'contributed to discovery of DNA'},
    'rcarson'  : {'surname'  : 'Carson',
        'forename' : 'Rachel',
        'born'     : 1907,
        'died'     : 1964,
        'notes'    : 'raised awareness of effects of DDT',
        'author'   : ['Silent Spring']
}
```

Write a function called `db_headings` that returns the set of keys used in any of the inner dictionaries. In this example, the function should return

```
set('author', 'forename', 'surname', 'notes', 'born', 'died').
```

Exercise 10:

Write another function called `db_consistent` that takes a dictionary of dictionaries in the format described in the previous question and returns `True` if and only if every one of the inner dictionaries has exactly the same keys. (This function would return `False` for the previous example, since Rosalind Franklin's entry doesn't contain the `'author'` key.)

Exercise 11:

A *sparse* vector is a vector whose entries are almost all zero, like

```
[1, 0, 0, 0, 0, 0, 3, 0, 0, 0]
```

Storing all those zeros in a list wastes memory, so programmers often use dictionaries instead to keep track of just the nonzero entries. For example, the vector shown earlier would be represented as

```
{0:1, 6:3}
```

because the vector it is meant to represent has the value 1 at index 0 and the value 3 at index 6.

a. The sum of two vectors is just the element-wise sum of their elements. For example, the sum of `[1, 2, 3]` and `[4, 5, 6]` is `[5, 7, 9]`. Write a function called `sparse_add` that takes two sparse vectors stored as dictionaries and returns a new dictionary representing their sum.

b. The dot product of two vectors is the sum of the products of corresponding elements. For example, the dot product of `[1, 2, 3]` and `[4, 5, 6]` is 4+10+18, or 32. Write another function called i`sparse_dot` that calculates the dot product of two sparse vectors.

c. Your boss has asked you to write a function called `sparse_len` that will return the length of a sparse vector (just as Python's `len` returns the length of a `list`). What do you need to ask her before you can start writing it?