Week 6

Chapter 1

Outline

- Practice with dictionaries
- Functions with a variable number of arguments and with keyword arguments
- Lambda Functions and higher-order programming
- Some interesting `collections` datatypes

Chapter 2

Python Collections

- `list`: mutable; sequence
- `tuple`: immutable; sequence
- `set`: mutable but *the elements must be of an immutable type*; no multiplicity of elements.
- `frozenset`: immutable set datatype.

In fact, sets are special cases of **dictionary** objects (see later).

Chapter 3

Set Operations

All standard set operations are implemented either as methods or as **operators**. For instance:

- `<s1>.union(<s2>)` returns the union of the sets, as does the expression `<s1> | <s2>`
- `<s1>.difference(<s2>)` returns the set containing all the elements of `<s1>` that are not in `<s2>`

Chapter 4

Using loops to iterate over collections

Naturally, all Python collections are **iterable**, and it is possible to loop over the elements in different ways:

- using **indices** as target variable values (for sequences)
- using **elements** as target variable values

Chapter 5

Dictionaries

A Python dictionary object implements a **mapping** from **immutable keys** to **values**. The dictionary can be initialized in many ways:

- by creating a new, empty dictionary
- by using an existing mapping of key-value pairs
- from an *iterable* consisting of tuples of key-value pairs
- from a keyword-argument list

The notation `<dict>[<key>]` allows us to obtain (or set) the value that we want associated with `<key>` in the dictionary!

> Caution
>
> Be especially aware that (a) keys must be **immutable** objects, and (b) the notation with square brackets is similar to list indexing.

5.1 Example

Design a program that reads a CSV file containing grades of students (one line per student) to obtain two dictionaries:

1. where the keys are the exams (numbered from 1 through the total number of exams), and the values are the lists of grades in order of the students in the file.
2. where the keys are the row numbers and the values are themselves nested dictionaries whose keys are student names and values are the list of grades.

Chapter 6

Dictionary Operations

Dictionaries support many operations, including:

- iterators (so-called **view objects**) for the keys, values and *items* (key-value tuples)
- operations to `get` a value for a key (or default to one if the key does not have a value) or set a default value via `setdefault`
- `pop`: removing a key (while returning the associated value, or a default if specified)
- `popitem`: remove the last key inserted and return the item for that key.
- `update`: add/overwrite the key-value pairs based on the argument dictionary or iterables. See the signature for details!

6.1 Example

Huffman encoding, one of the earliest **lossless** compression schemes, starts by creating a dictionary with frequency counts of the symbols in the text being compressed.

We will create such a dictionary for the text of Huckleberry Finn.

Chapter 7

Using `for` loops with dictionaries

The different views of the dictionary (`keys`, `values` and `items`) can be used to iterate over dictionaries.

By default,

```
for <target> in <dictionary>:
    <loop body>
```

iterates over the keys in the dictionary.

Since the items are tuples, we can deconstruct them as follows:

```
for <key>, <value> in <dictionary>.items():
    # Use <key> and <value> in the body
```

7.1 Example (continuation of Huffman coding)

Create a dictionary of symbol-frequency counts in a text where the keys are ordered by increasing frequency counts.

Huffman encoding uses such an ordered dictionary to construct a **tree** that can be used to produce an **optimal**, prefix-free code!