CSC 108H: Introduction to Computer Programming

Summer 2011

Marek Janicki

Administration

- Next week lecture will be in BA1170
- Assignment 1 marks have been released on Markus.
  - Mean was a 71.
- Assignment 2 has been released.
  - Can ask questions at the end.
  - Office hours will be held Monday instead of Tuesday before it is due.
- The midterm will be held June 30th at the regular lecture time and regular lecture room.
  - Going to look to release old midterms this weekend.

Lists addendum

- As lists are mutable, nested lists can cause situations in which aliasing is non-obvious.
- Be wary of slicing lists that contain mutable elements.
- While the slicing creates new lists, the items in the list are aliases to the original elements.
- So you might be changing both lists when you think you're only changing one.

Lists addendum.

- + and * are overloaded in the same way that they are for strings.
  - So + will concatenate lists.
  - * will take an int, and make that many copies of the list.
  - Be wary of nested lists and mutability issues!
- `x in list` is a boolean operation that tests if x is in the list.
  - Useful because it doesn't spit out an error like index().
- `list.pop()` removes and returns the last item on the list.

Limitations of Lists

- So far we have a pretty powerful set of primitives.
- But lists have a few limitations.
- In particular, searching through a list takes a lon time.
  - We need to go over every element and check if it's the one we want.
- Problematic if we only want to alter one small record.

Limitations of Lists

- What if we don't know a lot about the data that we're getting?
- We can create a list using a loop and .append().
- But what if we have duplicates?
- Well, we can go back through the list after getting the whole thing and process it.
  - Slow.
- Also, we still can't index by the data.
  - So no `list_name[data]`, only `list_name[i]`

Example

- A lot of searching is based on word counts.
  - This is especially true in fixed data bases like Academic journals.
- One reads through a document, and counts words; and then normalises the word counts.
- Related documents should have similar normalised word counts.
- But you don't know what words you're looking for beforehand.

Dictionaries

- In one sentence, dictionaries are (key, value) pairs. Sometimes they are called maps.
- Python syntax:

```
{key0 : value0, key1 : value1, ...,
keyn : valuen}
```

- Dictionaries are of type `dict`
  - Since they have a type, they can be assigned to a variable.
- To refer to a value associated with a key in a dictionary we use `dictionary_name[key]`

Dictionaries

- Dictionaries are unsorted.
- Dictionary keys must be immutable, but the values can be anything.
  - Cannot be `None`.
- Once you've created a dictionary you can add key-value pairs by assigning the value to the key.

```
dictionary_name[key] = value
```

- Keys must be unique.

Dictionary methods.

- `len(dict_name)` works in the same way as it does for strings and lists.
- `+` and * are not defined for dictionaries.
- `dict.keys()` - returns the keys in some order.
- `dict.values()` - returns the values in some order.
- `dict.items()` - returns the (key, value) pairs in some order.
  - All of these methods have iter* variants that return the keys|values|key-value pairs one by one.

Dictionary methods.

- `dict.has_key(key)` - returns `True` iff the dictionary has the key in it.
- `dict.get(key)` – returns the value that is paired with the key, or `None` if no such key exists.
  - `get(key, d)` returns `d` rather than `None` if no such key exists.
- `dict.clear()` - removes all the key-value pairs from the dictionary.

Dictionary methods.

- `dict.copy() - copy the entire dictionary.`
  - Be wary if the dictionary has mutable objects.
  - Can have the same issue has with nested lists.
- `dict.update(dict_name)` - adds the key-value pairs in dict_name to dict.
- `dict.pop(key)` – removes and returns the key-value pair indexed by the key.
  - `popitem` returns the `(key, value)` pair.

Why dictionaries?

- Dictionaries are useful if you want to have really big sparse data structures.
  - You can implement spreadsheet, or alarms with dictionaries.
- Or if you get a big amount of data but you're not quite sure how complete it is.
  - So you have a bunch of names, but don't know how many of them you'll actually see.

Looping over dictionaries.

```
for key in d:
    print key, d[keys]
```

- Works, but is a bit slow.

```
for key in d.iterkeys():
    print key, d[keys]
```

- This is a bit better.
- However, the order is still arbitrary.
- How can we make the loop ordered?

Inverting a dictionary.

- Sometimes we want to figure out what the key corresponding to a given value is.
  - This is impossible to do naively.
  - That is, `dict[value]` will not return the key.
- That is we want an identical dictionary, except with keys and values switched.
- If we haven't built the dictionary yet, then we can build two at the same time, where they are inverses of each other.
- Otherwise we need to build an inverse dictionary.

A problem.

- While the keys in a dictionary must be unique, the values don't have this restriction.
- So multiple keys can have the same value.
- How do we build our reverse dictionary?
- We still need to make the values into keys, but we won't have enough values to give each key a unique value.
- We can solve this by pairing the original values with lists of original keys.

Assignment 2