Today - Under the hood

- More on how lists and dictionaries really work.
- You don’t need to know this to use them.
- But maybe it helps?
- Introduce a good trick you might be able to use someday.

Lists

- Items in a list are stored in order.
- Look up an item by its position.
- A list is always indexed by integers, starting with zero.
- An index which is >= the length of the list causes an error.

```python
shopping = ["milk", "eggs", "tea"]
print shopping[2]
```

Dictionaries

- Values in a dictionary are indexed by keys, which can be anything.
- Items in a dictionary are not in any particular order.
- Looking up a key that is not in the dictionary causes an error; check first:

```python
if "rice" in foodDict:
    print foodDict["rice"]
```

How lists work

- The computer memory is like a single big list.
- A variable is a name for one item.
Memory chip

Google image search for “memory chip dip14”

Memory chip

- Has pins for address (number of memory cell where data is stored) A0-A12
- Has pins for data in/out DQ0-DQ7

How lists work

- A Python list is a whole chunk of memory.

But dictionaries?

- Memory is sort of like “list hardware”
- There is no dictionary hardware
- Need to construct a dictionary out of a list
- Trick involving the mod operator
- Phonebook data

Mod operator

- ```python
  >>> 8%3
  2
  >>> 17%6
  5
  ```
- Integer operator
- Produces the remainder when int on left is divided by int on right

Dictionary trick

- A classic CompSci trick called “hashing”

```python
# I pick a prime number, larger than the number of things I want to store.
# This will be the length of my list.
listLen = 7

poser = [] # A list pretending to be a dictionary
# Fill it up with zeros
for i in range(listLen):
  poser = poser.append[0]
```

If I pick a prime number, larger than the number of things I want to store.
If this will be the length of my list.

```python
poser = [] # A list pretending to be a dictionary
# Fill it up with zeros
for i in range(listLen):
  poser = poser.append[0]
```
Putting stuff in the “dictionary”

```python
for item in phoneList:
    name, number = item
    # Compute the index from the number!
    index = number % listLen
    poser[index] = [number, name]
```

- Key idea: compute the index from the key, somehow.

Looking up a number

```python
index = number % listLen
if poser[index] == 0:
    print("The phone number is not here.")
else:
    dataList = poser[index]
    if number != dataList[0]:
        print("The phone number is not here.")
    else:
        print("Name is", dataList[1])
```

Example Problem

- Say I have a file of phone numbers and names.
  - 5302204728, "Oswald, Astrid"
  - 5307547821, "Ortiz, Esteban"

- I want to write a program that will let me enter a number, and get back the name.
- Phone number is an integer, name is a string.

Possible data structures

- Which would be a better data structure:
  - A list of names, indexed by number
  - A dictionary, using the numbers as keys
  - A list of lists [number, name]

Problem with list

- Many possible phone numbers won’t have a corresponding person
- If this does not cause an error:
  - [5302208945]
  - then the length of the list has to be >= 5302208945.
- Takes up a huge amount of memory.

Problem with list of lists

- Might have to read the whole list to find the phone number we want.
- So it is slow at answering queries (if there is a lot of data).
Dictionary is best

...for this problem, anyway.

- Length of dictionary is the number of items in it, not the size of the biggest key.
- You can access items using the key, not by looking through the whole data structure.
- Even though the keys are integers, if lots of possible keys are not used, then a dictionary still works best.

When is a list a better choice?

- When order is important.
- We can sort lists, but not dictionaries.
- Dictionaries are always in some weird arbitrary order.

Dictionary vs list, take 2.

- Dictionaries are a little slower, but not much.
- Dictionaries are a little bigger, but not much.
- Dictionaries have to be in “random” order to work properly.

Strings as Keys?

- This works for integer keys, but how about strings?
- Turn string into a big integer...
- which you use as an index!
- Basic idea: ord() function turns one character into an integer. Compute the index from these integers.