Recap form last time

- How to do for loops
- map, filter, reduce
- Next up: dictionaries

Key data structure: Dictionaries

Associative arrays, Hash tables ...

A table storing a set of “keys”, And a “value” for each key.

Any (immutable) object can be a key!
- int, float, string, tuples...

Very useful!

Using Dictionaries

Unsorted list of key,value pairs

Empty Dictionary: `{}`

Non-empty Dictionary: `{k1:v1, k2:v2,…}`

Membership: is k in dict: `k in d`

Lookup value of key: `d[k]`

Set value of key: `d[k]=v`

Dictionaries

```python
>>> d={}
>>> d=dict(mexmenu)
>>> d[“ceviche”] = 3.95
>>> d
{…}
>>> d[“burrito”]
3.50
>>> d.keys()
{…}
>>> d.values()
```
Dictionaries

```python
def freq(s):
    d = {}
    for c in s:
        if c in d:
            d[c] += 1
        else:
            d[c] = 1
    return d
```

```python
>>> d = plotfreq([1, 1, 3.0, "A", 3.0, "A", "A", 1, 2, 3.0, 1, "A"])
>>> d
...
```

```python
def plotfreq(s):
    d = freq(s)
    for k in d.keys():
        print k, "*"*d[k]
```

```python
>>> d = plotfreq("avrakedavra")
>>> d.keys()
>>> d
...
```

Today

- Data model

- States how the programmer should think of data

- Answers the question: what do names refer to?

Data model in functional langs

- Environment of bindings (phonebook)

<table>
<thead>
<tr>
<th>x</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>&quot;abc&quot;</td>
</tr>
<tr>
<td>z</td>
<td>[1, 2, 3]</td>
</tr>
</tbody>
</table>

- Never change a binding
  - add new bindings at the end of the phonebook

Data model in functional langs

- Variables are names that refer into the phonebook
- Most recent entry looked up during evaluation
- Environment "frozen" inside function value so that the behavior of the function cannot be changed later on (easier reasoning)
Data model in OO langs

- Vars point to Objects
- Objects are boxes with data inside of them

Can update the “link” that a name refers to, but can also update the “content” of the box that a name refers to. More on this later.

Namespaces in Python

- Namespaces are used to manage variable names in Python
- In some ways similar to the concept of environments in ML
- However, different in some ways though: Python programmers can explicitly define new namespaces, and refer to them

Namespaces in Python

- Core PL concept that unifies many different ideas
- Not unique to Python of course, but in Python, namespaces are taken to the next level, as we shall soon see
- Remember our friend Morpheus: forget all you know about namespaces for now...

What’s a namespace

- Just a mapping from names to objects
What's a namespace

- Wait, hold on a second!!!
- A mapping from names to objects?
- But, but, but... Isn't that just a ... 

Namespaces vs. Environments

- Battle of the language concepts!

What's a namespace

- Wait, hold on a second!!!
- A mapping from names to objects?
- But, but, but... Isn't that just a ... 
- Dictionary in Python? Yes, but let's defer that discussion to later...

Namespaces vs. Environments

- Battle of the language concepts!
- Let the race begin!
Seriously...

- Python Namespaces different from ML environments in two ways
- First difference: assignment!
  - Basic operation in imperative langs: x = e
  - In ML, let binding just added new binding
  - In Python, assgn updates binding to point to new object

Simple example

```
i, s = 0, 0
while (i <= 3):
    i, s = i+1, s+i
```

Second difference

- Ok, so first difference is update: we can update what a name refers to
- Second difference we can also change what’s inside the box

Note: it’s the same name s, which points to different objects. The namespace is not extended, like an environment
Updates

- Well, for integers we can’t change what’s inside the box - these are immutable objects
- But for mutable objects, we can change what’s inside the box
- Examples of mutable objects:

Updates: who cares?

- Ok, so we can change what’s inside the box. Big deal! Who cares?
- Well: Why is this new model conceptually different from saying “new box is built with the new updated value inside”?

Answer: aliasing!!! Go to code
Aliasing

- When two or more names refer to the same object, we call this aliasing

- This never happens in pure functional languages
  - actually, that’s not really true: it does happen, but you just can’t tell because you can’t mutate any memory objects...

Aliasing happens in almost any imperative language

- Java, Python: names map to objects (implicit pointers...)
- C: names “point” to memory cells (different from Java Python because the pointers are explicit)

Primitive types in Java

- So what about those primitives types in Java, like int, float, etc.
- For these types, var holds the value, not a pointer to value.
  - Can you tell the difference?
**Answer: no!**

- You can’t tell the difference!

- You can explain all of Java by telling someone that $x = y + z$ just creates a new immutable integer that $x$ points to...

- The reason you can’t tell the different is that there are no explicit pointers.

**Back to namespaces**

- Recall: namespace is a map from names to objects

- Notion of namespace pervades Python

- You can even name a namespace, and then look it up

- Go to code!

**Block namespace example**

```python
y = 0
x = [10]
def f(y):
    z = len(x) + y
    return z
f(5)
```

- to get $x$: not in local, found in globals
- to get $\text{len}$: not in local, not in globals, found in builtin
- to get $y$: found in current (global binding shaded)
- write $z$ in current

when $f$ is done, in this case throw the namespace away
Questions

- Why did I say “in this case”? Why is a stack not enough?

- When do we need to freeze a namespace and store it?

- Before answering this, let’s see some more examples (go back to code)

Pattern

- So what’s the pattern here?
- It’s free variables!
- Be very careful with free vars...

- In fact, because of free vars are tricky, Python takes pains to ensure you don’t accidentally overwrite variables outside the current scope (go back to code)

Moral of our code

- The moral: can change a function’s behavior based on something we did AFTER the function definition. This could not happen in ML!!!

- This is very scary, because completely unrelated code can change the behavior of our function g.

What happened?

- How come we didn’t get “smash”?
- Python treats assignments differently from reads...

- When processing an assignment x = e, Python computes e, the looks up x in the current scope (namespace).

- If x is there, then it updates the binding
- If not, it introduces new binding for x
What happened?

- This means that an assignment cannot change globals. Wow, that’s pretty amazing!!!

- Well, there is a workaround...

- Go back to code

Recall our questions

- Why did I say “in this case”? Why is a stack not enough?

- When do we need to freeze a namespace and store it?
Well, recall ML

- It has to do with capturing a name inside a function definition

- Let’s see this! Back to code!

Next time

- Next time we’ll finally see classes