Feedback

• “Like the PAs!”
• “Hate the PAs!”
• “Stop skipping slides!”
• “Give examples instead of slides!”

General Trends

• Keep:
  - Liked code examples
• Start doing:
  - Syntax highlighting
  - More tutor hours
  - Post slides sooner
  - Stop people from chatting during lectures
  - Throw things at people who are asleep

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
...
```

```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"])
```

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

```python
>>> d.plotfreq(["avrakedavra"]
>>> d.keys()
...
```
You now know enough to do PA5
- Python Tutorial: How to open files, read lines
- Use the `help` command
- Document every function: What does it do?

Next: What’s in a name?
More precisely:
- How should programmer think of data
- What does a variable “x” really mean?

What’s in a name?
ML (or Functional Languages)
- Name refers to a Value
- Binding maps Names to Values
- Environment list of bindings
- Environment can be extended
- Environment can’t be changed

Data model in functional PL
- Vars = names in phonebook
- Evaluation = Most recent
- Environment “frozen” in function value
  - behavior of function cannot be changed
  - easier reasoning

Data model in OO langs
- Variables “point to” objects
- Objects = boxes with data inside
Namespaces

- Manage variable names in Python
- Similar to, but different from Environments
  - Core PL concept, unifies many ideas
- We will see very important differences

Ok, but what IS a namespace?

A mapping from names to objects

```
X "pumpkin"
Y 3.142
Z [1, 2, 3]
```

Namespaces vs. Environments

Both are maps from variables to something

Namespace | Environment
---|---

What’s the difference?
1. Assignment
2. Updates/Mutation

1. Assignment

Basic operation in Imperative PL:

```
x = e
```

1. Compute object corresponding to e
2. Change the name “x” to refer to object

Simple example

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

Simple example

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

Same name “s”
- points to different objects
- namespace is not extended
1. Assignment

Basic operation in Imperative PL:

\[ x = e \]

1. Compute object corresponding to \( e \)
2. Change the name “\( x \)” to refer to object

Assignment: changes box that name refers to

2. Update/Mutation

Change what’s inside the box (object)

- Not with immutable objects
  - eg. integers
- But with mutable objects
  - eg. arrays, lists, dictionaries

```
>>> x = [100,200]
>>> x
[100, 200]
>>> x[0] = "gobble gobble"
>>> x
['gobble gobble', 200]
```

How is it different from “build a new box with updated value inside”?

Aliasing

Two or more names refer to same object

“Peter Parker”

“Spider-Man”

Aliasing and Update

Two or more names refer to same object

```
>>> x = [100,200]
>>> y = x
>>> y[0] = "gobble gobble"
>>> x
['gobble gobble', 200]
```

If multiple names refer to same object, update affects values of all names
Aliasing

Does not happen in Ocaml/Functional PLs
• actually it does happen (where ?)
• but not exposed to the (130) programmer

Does happen in every imperative PL
• Java, Python: names point to objects
• C: names point to memory cells

Aliasing

Good because ?

Bad because ?

Namespaces everywhere

Namespace = map from names to objects

Notion of namespace pervades Python

• Can create namespace,
• Can name a namespace,
• Can peep inside a namespace (see what's bound)

Go to code!

Creating Namespaces

a.py
```
x = 22
y = "this sentence is false"
```

b.py
```
x = "pumpkin"
y = 3.142
```

```python
>>> import a
>>> a.x
22
```

Namespaces

Different names can point to same object!

```
X a
Y "this"
```

```
X a
Y "22 pumpkins"
```

```
X b
Y 3.142
```

```
X b
Y 3.142
```

Namespaces

For two namespaces a, b:
• names inside unrelated
• names in different spaces
  a.x:
  attribute/name "x" in space "a"
  b.x:
  attribute/name "x" in space "a"

Different names can point to same object!
Creating Namespaces: Fun Calls

Call-by-Value:
- New local namespace for call
- y bound to same object (value) as arg x
- x binding unchanged by call

In this case, after call, local namespace disappears...

Questions:
- Why “new local namespace” (not just stack) ?
- What’s the deal with “x” not declared/bound in “f” ?
- When do we need to freeze a namespace ?

Creating Namespaces: Fun Calls 2

Static Scoping
Lookup at runtime
Not compile time
Missing z added

What happened ?
Looks for “n” at run-time, when “g” is called
Can’t find “n” in local, global, builtins
Throws run-time error...

Creating Namespaces: Fun Calls 3

What happened ?
Looks for “n” at run-time, when “g” is called
Finds “n” in global, returns 15
Here “n” is a “free variable” of “g”
Needs to be “bound” in some enclosing scope

Aaargh!

Changed behavior after definition
whether or not fun works depends on what we did after fundef
Change I/O behavior too ...
Unlike ML, no new binding: just change what “n” is bound to be careful with free variables!
Python tries to avoid “overwrites”

>>> n
100
>>> def f():
   n = “smash”
   print n
>>> 
>>> f()
smash
>>> n
100

Python tries to ensure you don’t overwrite outer variables

>>> n
100
>>> def f():
   n = “smash”
   print n
>>> 
>>> f()
smash
>>> n
100

Python tries to ensure you don’t overwrite outer variables

How?
- unlike C/Java
- assignment different from reads
- no variable “declarations”
- assignment = declaration!

Python tries to avoid “overwrites”

Assignment Revisited

x = e

1. Compute object corresponding to e
2. Change the name “x” to refer to object in the current namespace (added if missing)

>>> x
10
>>> def g():
   x = x + 1
   print x
>>> 
>>> g()
>>> x

What happens?

>>> x
10
>>> def g():
   global x
   x = x + 1
   print x
>>> 
>>> g()
>>> x
What happens?

```python
>>> x = [10]
>>> def g():
...   x[0] = “abc”
...   print x
...   g()
>>> x
```

What happens?

```python
>>> x = [10]
>>> def f(y):
...   def h(z):
...     return (y+x[0]+z)
...   return h
... foo = f(5)
>>> foo
<function object>
>>> foo(100)
115
>>> foo1 = f(-5)
>>> foo1(100)
105
```