CS150 - More on Objects

HW due Friday
Last time

- Basic Account class (for our example)
  - id (variable): returns a unique identifier for underlying object

- is versus ==

  myCheeating
  myDebit
  mySavings

  Account
  balance: 100

  Account
  balance: 250
Primitive Types

Lists: Say we create 2 lists, which have the same contents.

Will they be the same object?

No - different ids

Strings: Will 2 identical strings have the same id? - Yes

(Test!)
mutable versus immutable

The difference is in the type of operations.

If one list is changed, the other should not change.

But — strings are immutable! We'll never be able to change that object.

Some variation here:

\[
\begin{align*}
\text{id}(4) &= 4 \\
\text{id}(2+2) &= 4 \\
\text{id}(\text{"aloha"}) &= \text{id}(\text{Aloha! lower()})
\end{align*}
\]
Garbage collection

Creating an object allocates space in memory.

What happens to that data once we are done with it?

Ex: `list1 = range(10)`
    `list2 = range(11)`
    `list1 = list2`

![Diagram of Garbage Collection](attachment:image.png)
Garbage Collection

The task of deallocating memory that is no longer used is called garbage collection.

Python does this for you:
- each object keeps a reference count
- when ref count == 0, deletes that object

(This takes time, so is one of the reasons Python is a slower language than some others.)
Objects referencing other objects

Technically, most of our classes reference other objects.

So our Account really looks like:

```
.mySavings
 Account
   balance
   .float
      0.0
```

(Other picture is simpler so may still use it)
Caution:

Immutable objects can contain mutable ones.

Ex: Frozen Assets = (mySavings, myChecking)

Can we change the Accounts?

tuple is immutable

but Accounts are mutable!

mySavings.deposit(100), deposit(50) both ok

Frozen Assets [[1]], deposit(50)
More caution with aliasing: Shallow copy

Say we have:

```ruby
myAssets = [myChecking, mySavings]
spouseAssets = myAssets
```

Shallow

```
myAssets
```

```
spouseAssets
```

```
list
```

```
myChecking
```

```
Account
- balance = 100.0
```

```
Account
- balance = 250.0
```

```
list
```

```
spouseAssets.append()
```
Another case:

If we have a command like:

```python
spouse Assets.append(spoouse Retirement)
```

This will change our list, too!

To keep our list unchange, need to actually create a second list.

Loop to append things to a new list
Shallow versus deep copies

In a shallow copy, the attributes of the object reference the same objects as the original.

In a deep copy, the attributes are independent copies of the original.

Previous examples were both shallow.
Example:

```python
names = ['alice', 'bob', 'eve']
for name in names:
    name = name.capitalize()
    print(name)
```

Output?

How to fix?
Now:

Suppose we want original list unchanged:

```python
names = ['alice', 'bob', 'eve']
upper_names = list(names)
```

```python
for i in range(len(upper_names)):
    upper_names[i] = upper_names[i].capitalize()
```

print names

Output? (and why?)

```
['alice', 'bob', 'eve']
```
The picture:

names

list

str 'alice'
str 'bob'
str 'eve'

list

uppernames

deep
To fix:

Make a deep copy:

```python
upper_names = []
for name in names:
    upper_names.append(name.capitalize())
```

Safe way to make a deep copy,
Copy x deep copy
Python has 2 modules:
- copy
- deepcopy

Copy (x) → shallow copy

Caution: not allowed on some objects (like files)

But only gives deep (or shallow) for level down.
Next: Functions

def multiply(value, input_list):
    for item in input_list:
        item *= value
    for i in range(len(input_list)):
        input_list[i] *= value

Q: Is input_list changed outside the function?
Practice 10.1 vs 10.2