ECS 10

10/31

Announcements

- Assignment due tonight.
- Many Eyes had a bug on their Web page, US counties had disappeared! We sent a bug report, and now it's back.
- Also, bug on my Web page that was not showing the lecture notes with some version of Explorer is fixed.
- Midterm on Monday. Bring a Scantron 2000.

Exceptions

```
popStr = raw_input('Enter the population: ')
try:
    pop = float(popStr)          # Try conversion
except:
    # Conversion failed!
    print 'Not a number.'
    pop = None

if pop != None:
    # Put rest of program here.
```

helper.py

- This is how helper.py worked.
- Recall specifically:

```
if helper.isFloat(popStr):
    pop = float(popStr)
else:
    print 'Not a number!'
```

- The expression **helper.isFloat(popStr)** is a Boolean, True if conversion will work.

Functions

- Today we consider how to put this bit of code into a function that we can include in a module.
- Any program we write with functions could have been written without them. No new magic….
- Makes it easy to pass code around, extend the language.
- Also makes for neater programs.

Parts of a Function

- We have used lots of functions. Example:
  x = int (53.66) - takes the float 53.66 and computes the integer 53, value of **int(53.66)** is 53, variable **x** gets 53.
- Function "parts of speech":
  - Float 53.66 is the **argument**
  - Int 53 is the **return value**

Defining functions

```
def isFloat(s):
```

- Begin defining function with **def**
- Parameters go inside parenthesis, eg. **(s)**
- Parameters are variable to hold values of arguments, inside the function.
- Might have no parameters.
- Might have lots of parameters.
- Colon!

Arguments and Parameters

- When I defined isFloat, I had no idea what you were going to call the variable containing the input string when you called helper.isFloat.
- But I needed a variable to put that data (the input string) into.
- My variable - s - is the parameter.
- The argument is the variable in your program that holds the input data.

Return Value

```
return True
```

- Functions usually produce an output data value.
- **return** is a Python command, only used in functions.
- The data value after **return** is the output data value produced by the function.
- We say this value is 'returned' by the function.

Let's make a new function

- Remove commas from a string.
- We'll just put it into helper.

```
def noComma(s):
s = s.replace(',' , '')
return s
```

- It is OK to change the parameter.
- It is OK to return the parameter.

From the outside…

- Function arguments don't change. After this:

```
reply = raw_input('Enter interest rate: ')
rate = float(reply)
```

- **reply** is still a string; the function **float()** did nothing to it.
- **rate** is a float (if there was no error).
- Functions we define ourselves are the same; the **arguments** stay the same, even if the **parameters** change in the function.

Functions in Modules

- Need to import the module before you can use the functions in it.
- Function **isFloat** in module **helper** is called **helper.isFloat.**
- You can shorten the prefix:

```
import helper as h
…
if h.isFloat(inStr):
    …
```

Functions in Program

- You can also define functions in the same file as the program that uses them.
- Put function definitions at the top of the file, right after the imports. Functions must be defined before they can be used!
- The code in a function is not run until the function is called, whether it is in a separate module or in the file with the program.

On the inside…

- What happens in functions, stays in functions.
- Changing parameters does not change arguments

```
def add_one(x):
    x = x+1
    return x

age = 5
next = add_one(age)
print "I am",age,"and soon I will be",next
```