ECS10
10/21

Announcements
- Raw midterm scores are on SmartSite.
- Programs handed back in section.
- If you want to see your Scantron, ask me in my Lab Hours (Mon AM 9-11) or Office Hours (Wds 2-3:30).
- If you did poorly, especially on the program, take it as a wake-up call. You need to master Chapters 1-3 in the book.

Loop over a string

```
strIn = "5,236,320"
i = 0
strOut = ""
while i < len(strIn):
    char = strIn[i]
    if char != ",":
        strOut = strOut+char
    i = i+1
```

Eliminates the commas.

for vs while
- Anything you can do with a for loop, you could also do with a while.
- for loops can only be used when you know how many times they will run before you start (length of list…)
- while loops are much more versatile, since you don’t need to know how many times it will loop.
- for loop will be a little shorter and tidier, especially if you don’t want the index variable.

for loop over a string

```
strIn = "5,342,750"
strOut = ""
for char in strIn:
    if char != ",":
        strOut = strOut+char
```

Exactly the same effect as version using while.
- Prettier, shorter.
- char takes on values “5”, then “,”, then “3”…..

The replace method

```
inString = "2,407,018"
popString = inString.replace(",", ")
population = int(popString)
```

- Replaces all copies of the first argument with the second.
- Here, replaces all commas with the empty string; that is, eliminates commas.

```s = ‘Flinch’
s = s.replace(‘Fl’, ‘Gr’)```
Methods

- If replace was a normal function we'd expect the name of the function to come first:
  ```python
  num = int("45")
  length = len("A", "B", "C")
  name = raw_input("What is your name? ")
  ```
- Instead, the name of the string the function works on comes first:
  ```python
  popString = inString.replace("", ")
  ```

New Assignment

- Extract numbers from a table of data for the user.
- Table is a big string, use string methods to break it up put the data into lists.
- Find the information the user wants in the lists.

String methods

- Google "Python string methods"
  - `split` separates the string into parts, and the part after the separator. If the separator is not found, return a 0-tuple containing the string itself, followed by two empty strings. New in version 2.5
  ```python
  replace(old, new, count)
  ```
  - Returns a copy of the string with all occurrences of string old replaced by new. If the optional count is present, only the first count occurrences are replaced
  ```python
  rfind(strip, [start], [end])
  ```
  - Returns the highest index in the string where substring strip is found, such that strip is contained within self.
- There are lots of them!

UC Budget

- Total Current Operations – how much we spend each year to keep running. Salaries, buildings, stuff.
- General Campus Instruction – mostly Professor’s salaries.
- State Appropriation – money we get from the State of California.
- Tuition and Fees
- Grant and Contract Overhead – taken off the top of research grants.

Making a list from a big string

- Cut up string by cutting out all copies of some character:
  ```python
  sentence = ‘How do you get the words?’
  words = sentence.split(’ ’)
  ```
- Produces a list of littler strings (here, the words in the sentence).
- Notice the question mark ends up as part of the last string, ‘words?’

Splitting on newline

- Multi-line strings can be put in triple quotes.
- Each line ends with the invisible `newline` character.
- Refer to newline in Python with ‘\n’.
  ```python
  lines = poem.split("\n")
  ```
- Produces a list of lines, each of them a string.
Splitting on whitespace

- Cutting out everything invisible (spaces, newlines, tabs...):
  ```python
  words = poem.split()
  ```
- No argument to split method.

The find method

- `string = "$23.95"
  decimal = string.find('.')`
  - `decimal` gets the value 3.
  - `find` returns the first index of the string argument in the parenthesis.
- `string = "romania"
  string.find('oman')`
  - If string is not found, returns -1

The in operator

- Checks to see if one string is part of another
  ```python
  sentence = "Practice makes perfect."
  if 'act' in sentence:
    print 'Found it!'
  ```
  - The string 'act' is part of the sentence.
  - So the Boolean expression
    ```python
    'act' in sentence
    ```
    - ...is True, and the program prints 'Found it!'

Using in with lists

- Checks to see if an item is in a list.
  ```python
  beasts = ['cow', 'goat', 'rabbit']
  if 'cow' in beasts:
    print 'Have a cow.'
  ```
  - This prints 'Have a cow.', because the string 'cow' is one of the items in the list.
- The expression
  ```python
  'ow' in beasts
  ```
  - ...has value False.

Boolean operators

- Examples of Boolean operators:
  ```python
  reply == 'r'
  x <= 11
  'i' in 'team'
  'cow' in ['sheep', 'pig', 'rabbit']
  ```
  - Each of these has value either True or False
  - The in operator produces True if the data item on the left is part of the sequence on the right.