ECS10

2/8

Announcements

- I have some midterms with no section time, nonsense section time, etc. See me after class.
- We will not grade checkpoint but we will check that you submitted it. Final program due next week.

String methods

- Google "Python string methods"

separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings. New in version 2.5.

replace(*old, new*[, *count*])
Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub* [,*start* [,*end*]])
Return the highest index in the string where substring *sub* is found, such that *sub* is contained within

- There are lots of them!
- Check here for things you can use.

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")
```

The strip() method, revisited

- Removes all whitespace from the beginning and end of a string.
- Whitespace is any character that prints as space rather than ink; space, tab, newline.

The split() method, revisited

- Splits on whitespace
- Removes tabs and newlines as well as spaces

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
```

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"…..
- Each character in turn.

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…), or with break statement.
- while loops are more versatile, since you don't need to know how many times it will loop.
- for loops are a shorter and tidier.

for over a list

```
s = "Double bacon cheeseburger (Hamburgers) 900"
words = s.split()
for w in words:
    if w == "(Hamburgers)":
        break
    print(w,end=" ")
```

- Variable w contains each word in turn; first "Double", then "bacon"….

for over different things

```
for x in thing:
    # type of x depends on type of thing
```

- If thing is a string, x is a character
- If thing is a list, x is an element of the list
- And….

Loop on integers

```
count = 0
while count < 5:
    print count
    count = count+1
```

Count is the index variable.

Loop over integers

```
for count in [0,1,2,3,4]:
    print( count)
```

Shorter with a **for** loop. But we might need to make a very long list.

Loop over integers

```
for count in range(5):
    print count
```

Prettier with the range function.

Range function

```
for count in range(5):
   print count
```

- Prints 0-4
- **range(5)** is a built-in Python function
- **range(1,6)** prints 1-5

Range function

```
string = ''
for count in range(5):
   string = string + 'ha'
```

- Standard way to do something a fixed number of times.

Iterator

- **range()** produces a data object of type "range", which is a specific kind of iterator.
- To see the values that will be produced by the iterator, try

  list(range(5))

  or

  tuple(range(5))

range() vs randrange()

- Two different functions.
- The value produced by range() is an iterator.

```
x = range(10)
```

- The value produced by randrange() is a single random integer.

```
randNum = randrange(10)
```

while version

```
balance = 100.0
annualRate = 7.0
monthlyRate = annualRate/12.0
month = 0
while month < 12:
   balance=balance+monthlyRate/100.0*balance
   month = month+1
```

for version

```
balance = 100.0
annualRate = 7.0
monthlyRate = annualRate/12.0
for  month in range (12):
   balance = balance+monthlyRate/100.0*balance
```

- Two lines shorter than while version….
- There is no such thing as an infinite for loop!
- Most common way to do it.

for on a file

```
inFile = open("menu.txt","r")
for line in inFile:
        print(line)
```

- Here line is a string
- Each time through the loop, it contains the next line of the file.