Sequence Types

MARCH 2ND

Problem

Write a program that counts the number of numbers in the range 0 through 1000 that contain the digit 7.

- The program in its entirety:

```
def containsSeven(s):
    return "7" in s

print len(filter(containsSeven, map(str, range(0, 1001))))
```

Strings and Lists

- A *string* is a sequence of characters enclosed in quotes.
  Examples: "hello", "8.397", "7", '34'
  (The quotes can be single or double quotes)

- A *list* is a sequence of objects enclosed in square brackets.
  Examples: [0, 1, 2, 3], ["Alice", "Bob", "Catherine"],
  ["hello", 4.567, -22, 87L, 'bye']
  (Objects of different types can be part of the same list)

- Lists are more "general" than strings; strings can be viewed as special instances of lists.

Simple operations on lists

- The `in` operator is used as `x in L`, where `x` is an object and `L` is a list. This expression evaluates to `True` if `x` is an *element* in `L`; evaluates to `False` otherwise.

  Examples: `67 in [34, 12, 45]` evaluates to `False`

  `"hi" in []` evaluates to `False`, etc.

- Python has a built-in function `len(L)` that returns the length, i.e., the number of elements, in list `L`.

  Examples: `len([])` is `0`, `len([34, 12, 45])` is `3`, etc.

Both of these work on strings as well

Examples:

"hi" in "history" evaluates to True

"ei" in "piece" evaluates to False

"ace" in "Wallace" evaluates to True

Examples:

len("history") returns 7

len("") returns 0

len("piece") returns 5

Generating lists

- Python has a built-in function called **range** that allows us to generate lists using *arithmetic progressions*.
- It can have one, two, or three arguments, all of which must be integers.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 >>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
 []
```

Useful in `for`-loops

```
for i in range(1, 10, 2):
        print i*i
```

- Repeats the execution of the body of the `for`-loop for each value of i = 1, 3, 5, 7, and 9.
- Equivalent to

```
i = 1
while i < 10:
        print i*i
        i = i + 2
```

- But more convenient for simple loops because no need to initialize before loop and no need to update within loop.

The map function

- map(f, [a, b, c, d, e]) returns the list [f(a), f(b), f(c), f(d), f(e)]
- The first argument of map is a function f and the second argument is a list L; it returns a new list obtained by applying f onto every element of L.

Examples:

- map(round, [4.57, -9.876, math.pi]) returns [5.0, -10.0, 3.0]
- map(str, range(0, 6)) returns ['0', '1', '2', '3', '4', '5']

- The map function allows us to construct new lists from old ones.

The filter function

- filter(f, L) returns a sublist of L consisting of those elements in L (in the same order as they appear in L) for which the boolean function f evaluates to True.
- **Examples:**
  - filter(bool, [0, -10, 0.0, None, "hello"]) returns [-10, 'hello']
  - filter(containsSeven, map(str, range(1001))) returns a list containing all of the numbers in the range 0 through 1000 that contain 7.

Problem

- Write a program that reads some text and extracts words in the text to build a “dictionary.”