More on recursion

Last time we went through a workshop on recursive functions.

Let’s look at some of the problems together.

Problem 6

Given a string *my_string*, return a string WITHOUT any of the letter x's. For example,

`no_x("x1xx2x3")` should return "123"

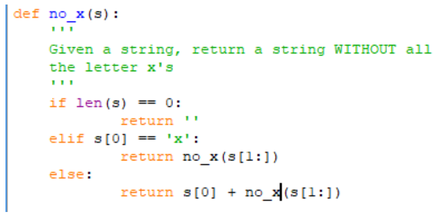
`no_x("xxx")` should return "“

`no_x(“123”)` should return “123”

Solution to Problem 6

- Base case(s)
  - If len(s) == 0:
    - return “”
- Recursion(s)
  - Check to see if the first letter s[0] is an ‘x’, if so, recursive call with string slicing s[1:] without s[0]
  - If the first letter s[0] is not an ‘x’, return s[0] + recursive call with string slicing s[1:]

Function no_x(s)



```
def no_x(s):
    '''
    Given a string, return a string WITHOUT all
    the letter x's
    '''
    if len(s) == 0:
            return ''
    elif s[0] == 'x':
            return no_x(s[1:])
    else:
            return s[0] + no_x(s[1:])
```

Problem 7

Given a string my_string, return a string in which all the characters are separated by *. For example,

all_star("hello") should return "h*e*l*l*o“

all_star("hi") should return "h*i“

all_star(“A”) should return “A”

Solution to Problem 7

- Base case(s)
  - If len(s) == 0 or len(s) == 1 # check len(s) == 1 is important, consider the case with one letter only
    - return s
- Recursive calls
  - return s[0] + ‘*’ + recursive call with slicing s[1:]

Function: all_star(s)

```
def all_star(s):
        '''
        Given a string s, computer a string in which all chars
        are separated by *. For example ‘hello’ becomes ‘h*e*l*l*o’
        '''
        if len(s) == 0 or len(s) == 1:
                return s
        else:
                return s[0] + '*' + all_star(s[1:])
```

New problem: counting vowels

- Given a string, return the number of vowls
  – E.g., count_vowels(‘hello’) returns 2
  – count_vowels(‘world’) returns 1
  – count_vowels(‘how are you?’) returns 5
  – count_vowels(‘12345’) returns 0

Solution to counting vowels

- Base case(s)
  – If string length is zero, return 0
- Recursive calls
  – If the first letter is a vowel, return 1 + call with slicing s[1:]
  – If the first letter is not a vowel, return call with slicing s[1:]

Function: count_vowels()

```
def count_vowels(s):
    '''
    Given a string, return the number of vowels
    '''
    if len(s) == 0:
           return 0
    elif s[0] in 'aeiou':
           return 1 + count_vowels(s[1:])
    else:
           return count_vowels(s[1:])
```

Edit distance

- One of our early reading quiz asks what is an edit distance.
  – Edit distance is the minimum number of operations required to transfer one string to another.
- E.g,

```
>>> distance(‘boy’, ‘joy’) # replace ‘b’ by ‘j’
1
>>> distance('spam', 'poems') # del ‘s’
4                               # add ‘s’ to end ‘pams’
                                # repl ‘a’ with ‘o’ ‘poms’
                                # insrt ‘e’   ‘poems’
>>> distance('alien', 'sales')  # see textbook
3
```

How to tackle the problem

1. We want to write a function *distance(s1, s2)* to measure the edit distance between s1 and s2
2. Base case(s)
   – If both strings are empty, the distance is zero
   – If one string is empty and the other is not, the distance is the length of the non-empty string

```
def distance(first, second):
   '''Returns the edit distance between first and second.'''
   if first == '':
      return len(second)
   elif second == '':
      return len(first)
```

How to tackle the problem (2)

3. If s1[0] == s2[0], we just call the function recursively with the remaining part of the string

```
elif s1[0] == s2[0]:
    return distance(s1[1:], s2[1:])
```

How to tackle the problem (3)

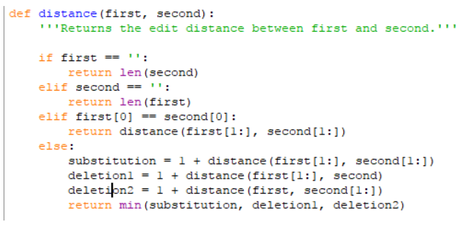
4. Now we need consider cases when the first letter is different
   1. Substitute the first letter of s1 by that of s2 (vice versa results the same.)

      Should return 1 + distance(s1[1:], s2[1:])
   2. Delete the first letter of s1, comparing the remaining

      Should return 1 + distance(s1[1:], s2)
   3. Delete the first letter of s2, comparing the remaining

      Should return 1 + distance(s1, s2[1:])
5. Use Python's `min()` function to find which one is the best!

The complete program



```
def distance(first, second):
    '''Returns the edit distance between first and second.'''

    if first == '':
        return len(second)
    elif second == '':
        return len(first)
    elif first[0] == second[0]:
        return distance(first[1:], second[1:])
    else:
        substitution = 1 + distance(first[1:], second[1:])
        deletion1 = 1 + distance(first[1:], second)
        deletion2 = 1 + distance(first, second[1:])
        return min(substitution, deletion1, deletion2)
```

Try it out! (distance.py)