1 Writing recursive functions

Recall from the previous handout these tips on writing a recursive function:

1) (Doc) Write the docstring first... trust me, it helps!

2) (Base) Figure out the base case: think of inputs where the answer is easy. If the input is a number, this is often 0 or 1. If it's a list or a string, this is often the empty string or list, or sometimes a string/list with just one letter/item.

3) For the recursive case:

   a) (Divide) Break the problem into two pieces: a piece you can "handle" easily and another piece which is a *smaller* version of the *same* problem.

   b) (Recurse) Follow the "have faith" principle. Make a recursive call and have faith the function will work correctly. This is where the docstring is helpful.

   c) (Combine) Take the result of the recursive call and the solution to the other smaller piece and combine them into a complete solution.

Exercises

One of these questions will be very similar to a final exam question.

1. Write a recursive function `is_pal` that takes a string `s` and returns `True` if `s` is a palindrome and `False` otherwise. A string is a palindrome if it reads the same forwards as backwards (e.g., `'kayak'`, `'civic'`, `'racecar'`). Hint: think carefully about the *divide* step.

   Solution:

```
def is_pal(s):
    '''(str) -> bool
    Returns True if s is a palindrome and
    False otherwise.
    >>> is_pal('civic')
    True
    >>> is_pal('civil')
    False
    '''
    if len(s) <= 1:
        return True
    return s[0] == s[-1] and is_pal(s[1:-1])
```

2. Write a recursive function `is_pal_ignore` that takes a string `s` and returns `True` if `s` is a palindrome *when spaces are ignored* and `False` otherwise. For example, `'a man a plan a canal panama'`

is a palindrome if you ignore spaces. Do **not** use `split`, `join`, or loops.

Solution:

```
def is_pal_ignore(s):
    '''(str) -> bool
    Returns True if s is a palindrome when
    spaces are ignored and False otherwise.
    >>> is_pal_ignore('     civil')
    False
    >>> is_pal_ignore('     civic')
    True
    >>> is_pal_ignore('civic    ')
    True
    >>> is_pal_ignore('civil    ')
    False
    >>> is_pal_ignore('a man a plan a canal panama')
    True
    >>> is_pal_ignore('a man a plan a canal panama!')
    False
    '''
    if len(s) <= 1:
        return True
    elif s[0] == ' ':
        return is_pal_ignore(s[1:])
    elif s[-1] == ' ':
        return is_pal_ignore(s[:-1])
    else:
        return s[0] == s[-1] and is_pal_ignore(s[1:-1])
```

3. Write a recursive function `r_max` that takes a list of numbers `L` and returns the largest number. You cannot use the built-in function **`max`**. You can assume that the input list has at least one number. Hint: think carefully about the *combine* step.

Solution:

```
def r_max(L):
    '''(list of int) -> int
    Returns the largest number in
    non-empty list L.
    >>> r_max([5])
    5
    >>> r_max([5, 2, 10])
    10
```

```
    >>> r_max([5, 12, 10])
    12
    '''
    if len(L) == 1:
        return L[0]
    else:
        max_rest = r_max(L[1:])
        if L[0] > max_rest:
            return L[0]
        else:
            return max_rest
```

4. Write a recursive function `e_parity` that takes a string `s` and returns `True` if `s` has an even number of e's and `False` otherwise. Hint: think carefully about the *combine* step.

Solution:

```
def e_parity(s):
    '''(str) ->
    Returns True if s has an even number
    of e's and False otherwise.
    >>> e_parity('ez pz')
    False
    >>> e_parity('bees knees')
    True
    '''
    if s == '':
        return True  # zero is an even number

    is_rest_even = e_parity(s[1:])
    if s[0] == 'e' and is_rest_even:
        return False
    elif s[0] == 'e':
        return True
    else:
        return is_rest_even
```

5. Write a recursive function `power` that takes a number `y` and returns 2 raised to the power `y` (e.g., `power(4)` should return 16). Do not use the built-in exponentiation operator or function.

Solution:

```
def power(y):
    '''(int) -> int
    Returns 2**y.
    >>> power(3)
    8
    >>> power(4)
    16
    '''
    if y == 0:
        return 1
    else:
        # this approach is more efficient
        x = power(y/2)
        if y % 2 == 0:
            return x*x
        else:
            return x*x*2
```

6. Write a recursive function `is_power` that takes a number `x` and returns `True` is `x` is a power of 2. Here's how to think about it recursively: a number `x` is a power of 2 if `x` is divisible by 2 and `x/2` is a power of 2. Hint: what is the base case?

Solution:

```
def is_power(x):
    '''(int) -> bool
    Returns True if x is a power of 2.  In other
    words there is some y such that 2**y = x.
    Returns False otherwise.
    >>> is_power(3)
    False
    >>> is_power(8)
    True
    >>> is_power(24)
    False
    >>> is_power(128)
    True
    '''
    if x == 1:
        return True
    elif x % 2 == 0:  # must be evenly divided by 2
```

```
        return is_power(x/2)
    else:
        return False
```

2 A logic puzzle

Here's a logic puzzle that you might try to solve using your powers of recursive thinking. This puzzle is taken directly from `http://xkcd.com/blue_eyes.html`.

> A group of people with assorted eye colors live on an island. They are all perfect logicians – if a conclusion can be logically deduced, they will do it instantly. No one knows the color of their eyes. Every night at midnight, a ferry stops at the island. Any islanders who have figured out the color of their own eyes then leave the island, and the rest stay. Everyone can see everyone else at all times and keeps a count of the number of people they see with each eye color (excluding themselves), but they cannot otherwise communicate. Everyone on the island knows all the rules in this paragraph.
>
> On this island there are 100 blue-eyed people, 100 brown-eyed people, and the Guru (she happens to have green eyes). So any given blue-eyed person can see 100 people with brown eyes and 99 people with blue eyes (and one with green), but that does not tell him his own eye color; as far as he knows the totals could be 101 brown and 99 blue. Or 100 brown, 99 blue, and he could have red eyes.
>
> The Guru is allowed to speak once (let's say at noon), on one day in all their endless years on the island. Standing before the islanders, she says the following:
>
> "I can see someone who has blue eyes."
>
> Who leaves the island, and on what night?
>
> There are no mirrors or reflecting surfaces, nothing dumb. It is not a trick question, and the answer is logical. It doesn't depend on tricky wording or anyone lying or guessing, and it doesn't involve people doing something silly like creating a sign language or doing genetics. The Guru is not making eye contact with anyone in particular; she's simply saying "I count at least one blue-eyed person on this island who isn't me."
>
> And lastly, the answer is not "no one leaves."