Environment Diagrams and Recursion

Computer Science 61A

January 31, 2017

1 Environment Diagrams

1.1 Questions

1. Draw an environment diagram for the following code.

```
def lg(b, t):
    if b > t:
        return 'pride'
    elif b == 3:
        print('pride')
    if t == 5:
        return lg(t, b)
    return lg(b, b + t)

lg(2, 3)
```

2. Draw an environment diagram for the following code.

```
def f(x):
    if x == 1:
        return "Env diagrams"
    return x + " are fun!"

(lambda x: f(x))(f(1))
```

3. Draw an environment diagram for the following code. Put what is printed in the terminal in the box at the bottom.

```
def lamb(da):
    if not da:
        print("ron")
    elif da == "harry":
        print("potter")
    if lamb:
        print("hogwarts")
    return "61a"

(lambda x: x(x)(print("harry")))(lambda x: lamb)
```

2 Recursion

2.1 Questions

1. Recursively count the number of digits in a positive integer n that occur before the digit 7. View the doctests for more details.

```
def count_before_seven(n):
    """
    >>> count_before_seven(42)
    2
    >>> count_before_seven(707)
    0
    >>> count_before_seven(42742):
    2
    """
```

2. Its possible to use recursion to accumulate values, just like in a loop! Write a function that recursively calculates the nth power of 2. Assume n is an integer 0 or greater.

```
def nth_power_two(n):
    """
    >>> nth_power_two(0)
    1
    >>> nth_power_two(1)
    2
    >>> nth_power_two(4):
    16
    """
```

3. **Bonus** Write the recursive function so it works for any integer n.

```
def nth_power_two(n):
    """
    >>> nth_power_two(0)
    1
    >>> nth_power_two(-2)
    0.25
    >>> nth_power_two(4):
    16
    """
```

4. Implement the function nearest two, which takes a positive number x as input and returns the power of two $(\ldots, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8, \ldots)$

```
def nearest_two(x):
    """ Return the power of two that is nearest to x.
    >>> nearest_two(8)
    8.0
    >>> nearest_two(11.5) # closer to 8 than 16
    8.0
    >>> nearest_two(0.75) # tie between 0.5 and 1
    1.0
    """
```

1. Write an iterative solution.

2. Write a recursive solution.

5. Write a function that computes the *digital root* of a number n. The *digital root* is defined as a recursive summation of the digits of n until only one digit is left. Hint: you may find a separate function to sum the digits of a number useful.

```
def digital_root(n):
    """
    >>>digital_root(5789)
    2 #5+7+8+9 = 29; 2+9 = 11; 1+1 = 2
    >>>digital_root(37)
    1 #3+7 = 10; 1+0 = 1
    >>>digital_root(999888774)
    6 #9+9+9+8+8+8+7+7+4 = 69; 6+9 = 15; 1+5 = 6
    """
```