Mutable Data Structures and Data Abstractions

Computer Science Mentors 61A

February 15 to February 19, 2016

1 Recursion

Every Recursive function has three things.

1. One or more base cases
2. One or more ways to break the problem down into a smaller problem
    - E.g. Given a number as input, we need to break it down into a smaller number
3. Solve the smaller problem recursively; from that, form a solution to the original problem

1. Write `num_digits`, which takes in a number `n` and returns the number of digits it has.

```
def num_digits(n):
    """Takes in an positive integer and returns the number of
    digits.

    >>> num_digits(0)
    1
    >>> num_digits(1)
    1
    >>> num_digits(7)
    1
    >>> num_digits(1093)
    4
    """
```

2. Write a function `is_sorted` that takes in an integer `n` and returns true if the digits of that number are increasing from right to left.

```
def is_sorted(n):
    """Return true if the digit is in increasing order from
    rightmost digit to leftmost digit. (Consecutive same digits
    are allowed).
    Also return true if it has only one digit. Return false
    otherwise.

    >>> is_sorted(2)
    True
    >>> is_sorted(22222)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(9087654321)
    False
    """
```

2 Environment Diagrams

1. Draw an environment diagram for the following code:

```
x = 20
def foo(y):
    x = 5
    def bar():
        return lambda y: x - y
    return bar

y = foo(7)
z = y()
print(z(2))
```

2. What would change here?

```
x = 20

def bar():
    return lambda y: x-y

def foo(y):
    x = 5
    return bar

y = foo(7)
z = y()
print(z(2))
```

3 Higher Order Functions

1. Write a higher order function that passes the following doctests. *Challenge:* Write the function body in one line.

```
"""
>>> from operator import add, mul
>>> a = mystery(add, 3)
>>> a(4) #equivalent to add(3,4)
7
>>> a(12)
15
>>> b = mystery(mul, 5)
>>> b(7) #equivalent to mul(5,7)
35
>>> b(1)
5
>>> c = mystery(lambda x, y: x*x + y, 4)
>>> c(5)
21
>>> c(7)
23
"""
```

2. What do these print out?

```
>>>foo = mystery(lambda a,b: a(b), lambda c: 5 + square(c))
>>>foo(-2)
```

4 Data Abstraction

1. The following is an **Abstract Data Type (ADT)** for elephants. Each elephant keeps track of its name, age, and whether or not it can fly. Given our provided constructor, fill out the selectors:

```
def elephant(name, age, can_fly):
    """
    Takes in a string name, an int age, and a boolean can_fly.
    Constructs an elephant with these attributes.
    >>> dumbo = elephant("Dumbo", 10, True)
    >>> elephant_name(dumbo)
    "Dumbo"
    >>> elephant_age(dumbo)
    10
    >>> elephant_can_fly(dumbo)
    True
    """
    return [name, age, can_fly]
def elephant_name(e):




def elephant_age(e):




def elephant_can_fly(e):
```

2. This function returns the correct result, but there's something wrong about its implementation. How do we fix it?

```
def elephant_roster(elephants):
    """
    Takes in a list of elephants and returns a list of their
       names.
    """
    return [elephant[0] for elephant in elephants]
```

3. Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
```

```
def elephant_name(e):
    return e[0][0]
def elephant_age(e):
    return e[0][1]
def elephant_can_fly(e):
    return e[1]
```

4. How can we write the fixed `elephant_roster` function for the constructors and selectors in the previous question?

5. **(Optional)** Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
    """
    >>> chris = elephant("Chris Martin", 38, False)
    >>> elephant_name(chris)
        "Chris Martin"
    >>> elephant_age(chris)
        38
    >>> elephant_can_fly(chris)
        False
    """
    def select(command)
```

```
    return select
def elephant_name(e):
    return e("name")
def elephant_age(e):
    return e("age")
def elephant_can_fly(e):
    return e("can_fly")
```

5 Lists

1. Draw box-and-pointer diagrams for the following:

```
>>> a = [1, 2, 3]
>>> a

>>> a[2]

>>> b = a
>>> a = a + [4, 5]
>>> a

>>> b

>>> c = a
>>> a = [4, 5]
>>> a

>>> c

>>> d = c[0:2]
>>> c[0] = 9
>>> d
```

2. Write a function that takes in a list `nums` and returns a new list with only the primes from `nums`. Assume that `is_prime(n)` is defined. You may use a `while` loop, a `for` loop, or a list comprehension.

```
def all_primes(nums):
```