LISTS, SEQUENCES, DICTIONARIES, DATA ABSTRACTION, AND LINKED LISTS

COMPUTER SCIENCE 61A

July 11 to July 17, 2015

1 Lists

1. What would Python print?

```
>>> watch_me = ["whip", "nae nae", "bop", "superman"]
>>> len(watch_me)
```

Solution: `4`

```
>>> watch_me[0]
```

Solution: `'whip'`

```
>>> watch_me[-1]
```

Solution: `'superman'`

```
>>> watch_me[0][1]
```

Solution: `'h'`

```
>>> watch_me[:-1]
```

Solution: `['whip', 'nae nae', 'bop']`

```
>>> watch_me[-1:]
```

Solution: `['superman']`

2. Implement `list_of_digits(n)`, which takes in a positive integer and returns a list of that number's digits. Write both a recursive and iterative version.

```
def list_of_digits(n):
    """
    >>> list_of_digits(3)
    [3]
    >>> list_of_digits(435)
    [5, 3, 4]
    """
```

Solution:

```
def list_of_digits_iterative(n):
    x = []
    while n > 0:
        x += [n % 10]
        n = n//10
    return x

def list_of_digits_recursive(n):
    if n < 10:
        return [n]
    return [n%10] + list_of_digits(n//10)
```

2 Dictionaries

1. Implement `gpa_calc(students)`, which takes in a dictionary of student names to their GPAs and returns a list of students whose GPA is above the 3.0 requirement to be a CS major.

```
def gpa_calc(students):
    """
    >>> students = {"Peter": 2.54, "Henry: 3.98, "Alice":3.29,
    "Lynn":2.46, "Kyle":3.54}
    >>> gpa_calc(students)
    ["Alice", "Kyle", "Henry"]
    """
```

Solution:

```
  return [name for name in students if students[name] > 3]
```

3 Data Abstraction

```
def create_team(a, b, c):
    """Constructor for creating a team of three pokemon.
    There always must be a pokemon given in parameter a.
    If not all three slots are full, then it will be an
    empty string"".
    """
    return {"first":a, "second":b, "third":c}

def choose(team, index):
    """Selects the pokemon at the given index of the team.
    """
    return team[index]

def add_to_team(pokemon, team):
    """Adds the given pokemon to the team. It will add
    to the next available spot in the team.
    If there are no more spots, the pokemon will not be added.
    Returns the team.
    """
    for key in team:
        if team[key] == "":
            team[key] = pokemon
            return team
    return team
```

1. Correct the following code so it does not violate abstraction barriers:

```
def catch(pokemon, team):
    """Changes the team to reflect caught pokemon.
    Returns the team.
    """
    team["third"] = pokemon
    return team
```

Solution:

```
    return add_to_team(pokemon, team)
```

```
def battle(team1, team2):
    """Team1 wins if the first pokemon is greater than team2's
        first pokemon.
    Team2 wins if the first pokemon is greater than team1's
       first pokemon.
    """
    if team1["first"] > team2["first"]:
        print("Team 1 wins!")
    else:
        print("Team 2 wins!")
```

Solution:

```
    if choose(team1, 'first') > choose(team2, 'first'):
        print("Team 1 wins!")
    else:
        print("Team 2 wins!")
```

4 Linked Lists

1. Implement `empty`, `is_link`, `link(first, rest)`, `first(s)`, and `rest(s)`.
`empty =`

Solution: `empty = 'empty'`

```
def is_link(s):
    """s is a linked list if it is empty or a (first, rest)
    pair."""
```

Solution:

```
    return s == empty or (type(s) == list and len(s) == 2
       and is_link(s[1]))
```

```
def link(first, rest):
    """Construct a linked list from its first element and the
    rest."""
```

Solution:

```
    assert is_link(rest), 'rest must be a linked list.'
    return [first, rest]
```

```
def first(s):
    """Return the first element of a linked list s."""
```

Solution:

```
    assert is_link(s) 'first only applies to linked lists.'
    assert s != empty 'empty linked list has no first
       element.'
    return s[0]
```

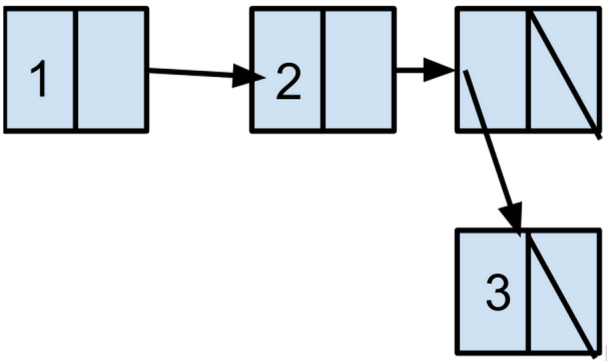
```
def rest(s):
    """Return the rest of the elements of a linked list s."""
```

Solution:

```
    assert is_link(s), 'rest only applies to linked lists.'
    assert s != empty, 'empty linked list has no rest.'
    return s[1]
```

2. Draw a box and pointer for the following code:

```
>>> a = link(1, link(2, link(link(3, empty), empty)))
```



Solution:

3. Implement `everyother(lst)`, which takes in a linked list and returns a new linked list with only the even index elements.

```
def everyother(lst):
    """
    >>> linked_lst = link(1, link(3, link(7, link(5, link(6,
       empty)))))
    >>> new_linked_lst = everyother(linked_lst)
    >>> first(new_linked_lst)
    1
    >>> first(rest(new_linked_lst))
    7
    >>> first(rest(rest(new_linked_lst)))
    6
    """
```

Solution:

```
    if lst == empty or rest(lst) == empty:
        return lst
    else:
        return link(first(lst), everyother(rest(rest(lst))))
```