Class 14 - Worksheet

Upcoming Schedule

Project 3 is due **Monday, 29 February**. Before submitting Project 3 you must have complete the Orange Belt (either by getting a "Gold star" on Project 2 or completing the promotion requirements.

Orange Belt promotion requirements. The requirements for the Orange Belt promotion are now updated (as of 5:30pm on Friday, 26 February). If you have not already submitted a solution, the requirements to earn Orange Belt promotion are now:

1. Complete everything listed on the Class 13 notes.
2. Define a function, `list_append` that takes as input two lists, and returns a new list that contains all of the elements of the first and second list in order. For example,

```
>>> p1 = [1, 2, 3]
>>> p2 = [4, 5, 6]
>>> p3 = list_append(p1, p2)
>>> p3
[1, 2, 3, 4, 5, 6]
>>> p2[1] = 7 # should not change p3
>>> p1[0] = 4 # should not change p3
>>> p3
[1, 2, 3, 4, 5, 6]
```

Code

class15.py

Generalizing List Functions

```
def list_map(fn, lst):
    if not lst:
        return []
    else:
        return [fn(lst[0])] + list_map(fn, lst[1:])

def list_map(fn, lst):
    result = []
    for e in lst:
        result.append(fn(e))
    return result
```

```
def list_map(fn, lst):
    return [fn(e) for e in lst]
```

Define `list_increment` and `list_print` using `list_map`.

```
def make_list_mapper(fn):
    def mapper(lst):
        return list_map(fn, lst)
    return mapper

list_doubler =
```

Lambda Expressions

`lambda` makes a function:

```
lambda param1, param2: expression
```

is comparable to: `Python def new_func(param1, param2):  return expression`

The name `lambda` comes from Lambda Calculus, which was invented by Alonzo Church in the 1930s. Along with the Turing machine model we have already informally introduced, Lambda Calculus was the earliest model of a universal computer (and still one of the most widely used models). We'll explore this more towards the end of the semester, but for now, you can use `lambda` as a shortcut to make a function in Python.

```
def make_list_mapper(fn):
    return lambda lst: list_map(fn, lst)


list_increment =
```

Generalizing Generalizers

```
def list_accumulator(fn, lst, base):
    result = base
    for el in lst:
        result = fn(result, el)
    return result


def list_map(fn, lst):
    return list_accumulator(__________________, ________, ______)

def list_length(fn, lst):
    return list_accumulator(__________________, ________, ______)
```