Lecture 17 (2-23-26)

Python: Functional Programming

Motivating Questions:

1. What is **functional programming** and how is it different from **imperitive programming?**
2. Why are map, filter, **and** reduce **called** higher-order functions**?
3. How do we replace **map** and **filter** with **list comprehensions?**

Notes

- remember `map` and `filter` return generators
- remember list comprehensions return lists

Imperative:

1. Programs as a sequence of instructions --> (Tenant 2 of Unix Philo)
2. Change state and mutate data (variables) --> (Tenant 1 of Unix Philo)
3.
    3. Explicitly enumerate order of operations --> (Tenant 3 of Unix Philo)

Functional:

1. Program as a composition of functions --> (Tenant 2 of Unix Philo)
2. Minimize state and avoid side-effects --> (Tenant 1 of Unix Philo)
3. Transform streams of data --> (Tenant 3 of Unix Philo)

`people.py` example

```
#!/usr/bin/env python3

import collections

# Structures

Person = collections.namedtuple('Person', 'first_name last_name')
```

```
# Variables

People = [
    Person('Peter'   , 'Bui'),
    Person('Alyssa'  , 'Ritter'),
    Person('Carlos'  , 'Basurto'),
    Person('Genesis' , 'Argueta'),
    Person('Joshua'  , 'Bui'),
]

# Default sort

People = sorted(People)

for person in People:
     print(person.last_name, person.first_name)

# Sort by last_name with helper function
def last_name(p):
     return p.last_name

or person in sorted(People, key=last_name):
    print(person.last_name, person.first_name)

# Sort by last_name with lambda expression

for person in sorted(People, key=lambda p: p.last_name):
    print(person.last_name, person.first_name)

# Sort by last_name then first_name using two sorts
People = sorted(People, key = lambda p: p.first_name)
People = sorted(People, key = lambda p : p.last_name)

# Sort by last_name then first_name using one sort
People = sorted(People, key = lambda p: (p.last_name, p.first_name))
```

`exam01.stats.py` example

```
#!/usr/bin/env python3

import csv
```

```
import requests

# Constants

URL    = 'https://yld.me/raw/gDQD'
POINTS = [1, 1.5, 1.5, 1, 1.5, 2.5, 2]
MAX    = sum(POINTS)

# Fetch data

data   = requests.get(URL).text.splitlines()

# Compute individual scores (Imperative)

scores = []
for student in csv.reader(data):
    points = []
    for point in student:
        points.append(float(point))
    scores.append(sum(points))

print(scores)
print(len(scores))

# Compute individual scores (Functional) - Phase I

scores = []
for student in csv.reader(data):
    points = map(float, student)
    scores.append(sum(points))

print(scores)
print(len(scores))

# Compute individual scores (Functional) - Phase II

scores = map(lambda student: sum(map(float, student)), csv.reader(data))


print(len(list(scores)) #you generally don't want to make a list from
generator

# Compute individual scores (List Comprehensions) - Phase I

scores = [sum(map(float, student)) for student in csv.reader(data)]
```

```python
# Compute individual scores (List Comprehensions) - Phase II

scores = [sum([float(point) for point in student]) for student in
csv.reader(data)]

# Filter scores (Imperative)

Bs = []
for score in scores:                              # Discuss: high-level goal
    if 0.8*MAX <= score < 0.9*MAX:                  # Discuss: common pattern?
        Bs.append(score)                          # Discuss: change to As?

print(len(Bs))

# Filter scores (Functional)

Bs = filter(lambda score: 0.8*MAX <= score < 0.9*MAX , scores)
print(len(list(Bs)))

# Filter scores (List Comprehensions)

Bs = [score for score in scores if 0.8*MAX <= score < 0.9*MAX]

print(len(Bs))
```