CS 61A Summer 2017

Aggregation & Final Review

Mentoring 14: August 7, 2017

1 Fish

Species [species]	**Population** [pop]	**Breeding Rate** [rate]	**$/piece** [price]	**# of pieces per fish** [pieces]
Salmon	500	3.3	4	30
Eel	100	1.3	4	15
Yellowtail	700	2.0	3	30
Tuna	600	1.1	3	20

CS 61A wants to start a fish hatchery, and we need your help to analyze the data we've collected for the fish populations! Running a hatchery is expensive – we'd like to make some money on the side by selling some seafood (only older fish of course) to make delicious sushi.

The table `fish` contains a subset of the data that has been collected. The SQL column names are listed in brackets.

1.1 Profit is good, but more profit is better. Write a query to select the species that yields the most number of pieces for each price. Your output should include the species, price, and pieces.

1.2 Write a query to find the three most populated fish species.

1.3 Write a query to find the total number of fish in the ocean. Additionally, include the number of species we summed. Your output should have the number of species and the total population.

The table `competitor` contains the competitor's `price` for each `species`.

Species [species]	**$/piece [price]**
Salmon	2
Eel	3.4
Yellowtail	3.2
Tuna	2.6

1.4 Business is good, but a bunch of competition has sprung up! Through some cunning corporate espionage, we have determined one such competitor's selling prices.

Write a query that returns, for each species, the difference between our hatchery's revenue versus the competitor's revenue for one whole fish. For example, the table should contain the following row `Salmon|60`.

Because we make 30 pieces at $4 a piece for $120, whereas the competitor will make 30 pieces at $2 a piece for $60. Finally, the difference is 60.

1.5 At our current market rates, there are two price points for fish: $4 fish (salmon and eel) and $3 fish (yellowtail and tuna). We'd like to figure out how much money we might give up per fish if we were to reduce our prices to that of the competitor's lowest price at each price point.

Between eel and salmon, our competitor's lowest priced fish is salmon at $2. How much money would we give up if we sold 30 pieces of salmon at $2 (instead of $4) and 15 pieces of eel at $2 (instead of $4)?

2 Iterators & Generators

2.1 What is difference between an iterator and an iterable?

2.2 What Would Python Display?

```
class SkipMachine:
    skip = 1
    def __init__(self, n=2):
        self.skip = n + SkipMachine.skip

    def generate(self):
        current = SkipMachine.skip
        while True:
            yield current
            current += self.skip
            SkipMachine.skip += 1

p = SkipMachine()
twos = p.generate()
SkipMachine.skip += 1
twos2 = p.generate()
threes = SkipMachine(3).generate()
```

(a) `next(twos)`

(b) `next(threes)`

(c) `next(twos)`

(d) `next(twos)`

(e) `next(threes)`

(f) `next(twos2)`

2.3 Implement `run_length_decoder`, a generator that yields the decoded run length sequence from a list of `(value, length)` pairs.

```
def run_length_decoder(encoding):
    """
    >>> rld = run_length_decoder([('h', 1), ('e', 1), ('l', 2), ('o', 1)])
    >>> lst(rld)
    ['h', 'e', 'l', 'l', 'o']
    """
```

2.4 Write a generator that will take in two iterators and compares the first element of each iterator and yields the smaller of the two values.

```
def interleave(iter1, iter2):
    """
    >>> gen = interleave(iter([1, 3, 5, 7, 9]),
                         iter([2, 4, 6, 8, 10]))
    >>> for elem in gen:
    ...     print(elem)
    ...
    1
    2
    3
    4
    5
    6
    7
    8
    9
    """
```

2.5 A flat-map operation maps a function over a sequence and flattens the result. Implement the `flat_map` method of the `FlatMapper` class in 3 lines of code.

```
class FlatMapper:
    """A FlatMapper takes a function fn that returns an iterable value. The
    flat_map method takes an iterable s and returns a generator over all values
    that are within the iterables returned by calling fn on each element of s.

    >>> stutter = lambda x: [x, x]
    >>> m = FlatMapper(stutter)
    >>> g = m.flat_map((2, 3, 4, 5))
    >>> type(g)
    <class 'generator'>
    >>> list(g)
    [2, 2, 3, 3, 4, 4, 5, 5]
    """
    def __init__(self, fn):
        self.fn = fn
```