CS131 Practice Midterm

Problem #1:

For this problem, it may be helpful to recall Python's `str.index(substr)` function:

- if `substr` is in `str`, it returns the starting index of the first occurrence
- otherwise, it errors

For example, `"Matthew".index("ew")` returns `5`.

Part A

Rohit is building an email validator. Given a list of emails, he wants to remove inputs that are "obviously" not emails: either they don't have the `'@'` character in them, or the portion *after* the `'@'` (called the *domain*) does not have the `'.'` character.

In Python, help him write a function `remove_invalid_emails(inputs)` that accomplishes his task. **You may not define any nested functions. Your solution must be a single, one-line filter expression that includes a lambda.**

For example,

```
remove_invalid_emails(["matt@matthewwang.me", "notreal", "stay@home",
"st.ay@home"])
```

should return

```
["matt@matthewwang.me"]
```

```
def remove_invalid_emails(inputs):
  return list(filter(_____________________________________________, inputs))
```

Part B

Ellie is curious how many remaining email addresses belong to students, which she characterizes by having the string `'edu'` in the domain. Since she's friends with Rohit, she'll make use of his `remove_invalid_emails` function.

She wants a `count_edu_emails` that behaves like this:

```
count_edu_emails(["matt@matthewwang.me", "tree@stanford.edu",
"beaver@mit.edu", "thisisntreallyan@edu.email"])
```

should return

```
3
```

In Python, help her implement the following function that accomplishes this task. **You must use reduce in your solution.**

```
from functools import reduce

def count_edu_emails(inputs):
  valid_emails = remove_invalid_emails(inputs)
  return reduce(lambda accum, current: _____________________________)
```

Part C

Makar likes Rohit and Ellie's work, but thinks that it's not very *extensible*: what if, later, somebody comes up with better heuristics to check if an email is valid (or if it belongs to a student)?

In Python, write a function called `generate_email_validator(predicate)` that *returns a function* which filters emails by whether or not they pass the `predicate`.

For example,

```
bad_validator = generate_email_validator(lambda x: len(x) > 3)
bad_validator(["a", "thisissonotanemail", "matt@matthewwang.me"])

returns

["thisissonotanemail", "matt@matthewwang.me"]
```

Reminder: `filter` returns an iterator; you need to call `list()` on its result to get a list back, like Part A.

Part D

Now, it's Makar's time to shine. He wants to test the validators against each other! In particular, given a list of inputs *and* a list of email validation predicates, he wants to see how many emails each validator removes.

In Python, write a function `compare_validators(inputs, predicates)` that returns a list of numbers, where the `i`th item is the number of emails predicate `i` kept. **You must use the `generate_email_validator` you wrote in Part C. For full marks, your function body should be one line.**

For example,

```
compare_validators(["matt@matthewwang.me", "makarlATucd.edu", "a@a.a"],
[lambda x: len(x) > 3, lambda x: x != "makarlATucd.edu"])
```

would return

```
[3, 2]
```

(since we remove no emails with the first validator, and `"makarlATucd.edu"` with the second).

Problem #2

Rithika and Isabel are brushing up on interview questions *and* their knowledge of list comprehensions. They remember the all-too-classic *FizzBuzz* question.

For any number, FizzBuzz should:

- output "`Fizz`" if the number is divisible by `3`
- output "`Buzz`" if the number is divisible by `5`
- output "`FizzBuzz`" if the number is divisible by `3` **and** `5`
- output the number otherwise

In Python, write two functions:

- `fizzbuzz(n)` that returns the output of FizzBuzz on `n`
- `fizzbuzz_list(start, end)` that returns a list of numbers where FizzBuzz is applied on the numbers from `start` to `end` inclusive.

In Python, write a function `fizzbuzz_list(start, end)` that returns a list of numbers where FizzBuzz is applied on the numbers from `start` to `end` inclusive.

Your implementation of `fizzbuzz_list(start, end)` should be a single, one-line list comprehension.

For example

```
fizzbuzz_list(10,15)
```

should return

```
[“Buzz”, 11, “Fizz”, 13, 14, “FizzBuzz”]
```

Problem #3

Write a Haskell function called *dedup* that eliminates consecutive elements of a list that have the same value:

*Main> dedup [1,1,3,4,4,1,6,7,7]
[1,3,4,1,6,7]
*Main> dedup [1,2,3,4]
[1,2,3,4]

Problem #4

Consider the following snippet below written in a hypothetical language:

```
fn foo(person1, person2):
  person1 = new Person(“Siddharth”)
  person2.set_name(“Ashwin”)

fn main():
  persona = new Person(“Boyan”)
  personb = new Person(“Matt”)
  foo(persona, personb)
  print(persona.get_name())
  print(personb.get_name())
```

A. When this program runs, it prints out:

Siddarth
Ashwin

What type of binding strategy is this language using to pass parameters? Why?

B. If instead, the program printed:

```
Boyan
Ashwin
```

What type of binding strategy would this language be using to pass parameters? Why?

Problem #5

Given the following Haskell function which accepts Ints as arguments:

```
foo x y = \q -> \r -> q+r+x+y
```

A. Write the type signature for this function.

B. Rewrite this function so it is explicitly in curried form.

C. Assuming we call:

```
q = foo 10
```

What is the type of this function?

D. In the original version of the function, which free variables are captured by the following lambda function (which is returned by outer lambda function):

```
 \r -> q+r+x+y
```

Problem #6

Consider the following function which processes shapes, written in a dynamically typed language:

```
fn compute_volume(shape, depth) {
        area = shape.area()
        return depth * area
}
```

```
fn main() {
        Circle c(10);    // radius = 10
        Square s(20);  // side = 20
}
```

A. Assuming that Square and Circle objects are derived from a common base Shape object, would the compute_volume() function be using (subtype) polymorphism like we learned in CS32?

B. Assuming that Square and Circle objects are unrelated classes, not derived from some base Shape object, would the compute_volume() function be using polymorphism? If not, what approach is being used to call the area() method.

Problem #7

You have been assigned to write a program which needs to allocate and discard millions of arrays of the same size over time. At any one time, there might be tens of millions of active arrays. You can choose the language you use to write your program. Would you choose a language with Mark and Sweep garbage collection or Mark and Compact. Why?

Problem #8

Consider the following Haskell code:

```
foo z x = if z == 10 then (bar 5) else x
bar y
  | y <= 0 = 1
  | otherwise = bar (y-2) + bar (y-1)

main = do
  let q = foo 10 (bar 1000000)
  putStrLn (show q)
```

Will this Haskell program hang (running for a very long time) or will it finish very quickly? Why?

Solutions

1.A.

```
def remove_invalid_emails(inputs):
  return list(filter(lambda x: '@' in x and '.' in x[x.index('@'):], inputs))
```

1.B.

```
def count_edu_emails(inputs):
  valid_emails = remove_invalid_emails(inputs)
  return reduce(lambda accum, current: accum + int('edu' in current), valid_emails, 0)
```

1.C.

```
def generate_email_validator(predicate):
  return lambda inputs: list(filter(predicate, inputs))
```

1.D.

Here's a map version:

```
def compare_validators(inputs, predicates):
  return list(map(lambda predicate: len(generate_email_validator(predicate)(inputs)),
predicates))
```

Here's a comprehension version:

```
def compare_validators(inputs, predicates):
  [len(generate_email_validator(predicate)(inputs)) for predicate in predicates]
```

2.

```
def fizzbuzz(n):
  if n % 3 == 0 and n % 5 == 0:
    return "FizzBuzz"
  if n % 3 == 0:
    return "Fizz"
  if n % 5 == 0:
    return "Buzz"
  return n

def fizzbuzz_list(start, end):
  return [fizzbuzz(x) for x in range(start, end+1)]
```

3.

```
dedup :: [Int] -> [Int]
dedup [] = []
dedup (x:xs) = dedupAux xs x [x]

dedupAux :: [Int] -> Int -> [Int] -> [Int]
dedupAux [] y acc = acc
dedupAux (x:xs) y acc | x == y = dedupAux xs y acc
                      | x /= y = dedupAux xs x (acc ++ [x])
```

4.A. Since both objects changed, the only consistent binding strategy would be the reference binding strategy (NOT object reference). Why? First, the assignment of person1 to a new Person object changed our persona variable in our main function. The only binding strategy consistent with this result is reference binding. person1 is an alias for persona, so any changes to person1 also happen to the persona variable (not just the object, but the variable itself) Second, the person2 object was mutated via the call to set_name(); this is consistent with both object reference and reference binding strategies. So overall, this language must be using reference binding.

4.B. This would be consistent with the object reference binding strategy. We can deduce this because mutation of the object pointed to via person2 via a method call works, but reassignment of the object reference person1 in the foo() function has no impact on persona in main().

5.A. Write the type signature for this function.

```
foo :: Int -> Int -> (Int -> (Int -> Int))
```

This function takes two Ints, and returns a function which accepts an Int and returns a function which takes an Int and returns an Int.

5.B. Rewrite this function so it is explicitly in curried form:

```
foo x = \y -> \q -> \r -> q+r+x+y
```

5.C. Assuming we call:

```
q = foo 10
```

What is the type of this function?

```
foo :: Int -> (Int -> (Int -> Int))
```

5.D. In the original version of the function, which free variables are captured by this lambda function:

```
\r -> q+r+x+y
```

The x, y and q free variables are captured by the lambda shown above.

6.A. Assuming that Square and Circle objects are derived from a common base Shape object, would the compute_volume() function be using polymorphism?

In order to use subtype polymorphism, we have to be accessing a derived object (e.g., a Circle) via a variable of the base-class type (e.g., a Shape reference/object reference/pointer). Since in a dynamically typed language, variables/parameters do not have a type (just the values they refer to have a type), we are NOT referring to a derived object using a variable of the base-class type - the variable has no type! Thus this is not using polymorphism like we learned about in C++.

6.B. Assuming that Square and Circle objects are unrelated classes, not derived from some base Shape object, would the compute_volume() function be using polymorphism? If not, what approach is being used to call the area() method.

This is also not using polymorphism for the reason in 6.A. In this case, the program would be using duck typing.

7. There are two things to consider to make your decision:

The arrays are the same size → Using Mark and Sweep would not produce much fragmentation, since freed areas of memory can immediately be reused by the next allocation since all of the arrays are the same size. Using Mark and Compact would be OK too given this requirement.

There are a large number (tens of millions) of active arrays at any one time → Mark and Sweep would be better here, since Mark and Compact would have to copy over tens of millions of arrays during each compaction, whereas Mark and Sweep would not have to do any copies.

Based on both of these points, you should use a language with Mark and Sweep garbage collection.

8. Because Haskell is a lazy language, the call to (bar 1000000) will not be computed by the foo function. Why? the z value passed in (10) is an even number. This code should run in a few milliseconds or less.