Homework 4 – Fall 2023

In this homework, you'll explore some of the concepts in data palooza like type systems.. Some questions have multiple, distinct answers which would be acceptable, so there might not be a "right" answer: what's important is your ability to justify your answer with clear and concise reasoning that utilizes the appropriate terminology discussed in class. Each question has a time estimate; you'll know you're ready for the exam when you can solve them roughly within their time constraints.

We understand, however, that as you learn these questions may take more time. For that reason, only **starred red** questions need to be completed when you submit this homework (the rest should be used as exam prep materials). Note that for some multi-part questions, not all parts are marked as red so you may skip unstarred subparts in this homework.

You must turn in a PDF file with your answers via Gradescope - you may include both typed and hand-written solutions, so long as they are legible and make sense to our TAs. Make sure to clearly label each answer with the problem number you're solving so our TAs can more easily evaluate your work.

1. ** (3 min.) Consider the following Python function that takes in a list of integers (either 0 or 1) representing a binary number and returns the decimal value of that number. Fill in the blanks in the list comprehension so the function works correctly.

Example:

`convert_to_decimal([1, 0, 1, 1, 0])` should return 22.

`convert_to_decimal([1, 0, 1])` should return 5.

```
from functools import reduce
def convert_to_decimal(bits):
  exponents = range(len(bits)-1, -1, -1)
  nums = [_______________ for ________ in zip(bits, exponents)]
  return reduce(lambda acc, num: acc + num, nums)
```

2. **

a) ** (5 min.) Write a Python function named `parse_csv` that takes in a list of strings named `lines`. Each string in `lines` contains a word, followed by a comma, followed by some number (e.g. `"apple,8"`). Your function should return a new list where each string has been converted to a tuple containing the word and the integer (i.e. the tuple should be of type `(string, int)`). **Your function's implementation should be a single, one-line nested list comprehension**.

Example:
`parse_csv(["apple,8", "pear,24", "gooseberry,-2"]` should return `[("apple", 8), ("pear", 24), ("gooseberry", -2)]`.

Hint: You may find list unpacking useful.

b) ** (2 min.) Write a Python function named `unique_characters` that takes in a string `sentence` and returns a set of every unique character in that string. **Your function's implementation should be a single, one-line set comprehension**.

Example:
`unique_characters("happy")` should return `{"h", "a", "p", "y"}`.

c) ** (2 min.) Write a Python function named `squares_dict` that takes in an integer `lower_bound` and an integer `upper_bound` and returns a dictionary of all integers between `lower_bound` and `upper_bound` (inclusive) mapped to their squared value. You may assume `lower_bound` is strictly less than `upper_bound`. **Your function's implementation should be a single, one-line dictionary comprehension**.

Example:
`squares_dict(1, 5)` should return
`{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}`.

3. ** (4 min.) Consider the following Python function that takes in a string `sentence` and a set of characters `chars_to_remove` and returns `sentence` with all characters in `chars_to_remove` removed. Fill in the blank so the function works correctly. **You may not use a separate helper function**.

Hint: You may find some of the concepts we learned in our functional programming discussions useful 😁.

Example:
`strip_characters("Hello, world!", {"o", "h", "l"})` should return `"He, wrd!"`

```
def strip_characters(sentence, chars_to_remove):
    return "".join(____________________________________________)
```

4. ** (4 min.) Show, by example, whether or not Python supports closures. Briefly explain your reasoning.

Zeta Programming

5. ** You're working on a project at Zeta to add Haskell-like type inference to Python.

   (Fun fact: Meta actually does have an open-source gradual typing/type inference Python system called Pyre. Microsoft has one called Pyright)

   a) ** (2 min.) Consider this function: if you could add a Haskell-like type annotation, what would it look like? Alternatively, if you can't annotate it like Haskell, why not?

```python
from math import sqrt

def nth_fibonacci(n):
  phi = (1 + sqrt(5))/2
  psi = (1 - sqrt(5))/2
  return (phi**n - psi**n)/sqrt(5)
```

(Fun fact: this actually works!)

b) (2 min.) Consider this function. Assume that `network_type` is always a String. Your boss says that it's not possible to annotate this function, because its return type is not the same for every input (which breaks Haskell's rules about functions). You're convinced there's still a way to do this. What could that look like?

```
def get_network_type(obj):
  if not hasattr(obj, 'network_type'):
    return None
  return obj.network_type
```

Hint: get algebraic!

c) ** (3 min.) Consider this function.

```
def add(a,b):
  return a + b
```

It seems easy, right? Your boss provides the annotation:

```
add :: Num -> Num -> Num
```

Is this the best annotation? Why or why not?

Union Busting

6. **

   a) ** (4 min.) Examine the following C++ code:

```
union WeirdNumber {
  int n;
  float f;
};

int main() {
  WeirdNumber w;
  w.n = 123;
  std::cout << w.n << std::endl;
  std::cout << w.f << std::endl;
}
```

This evaluates to:

```
123
1.7236e-43
```

Why? What does this say about C++'s type system?

b) ** (7 min.) Zig is an up-and-coming language that in some ways, is a direct competitor to C and C++. Let's examine a similar union in Zig.

```
const WeirdNumber = union {
  n: i64,
  f: f64,
};

test "simple union" {
  var result = WeirdNumber { .n = 123 };
  result.f = 12.3;
}
```

```
test "simple union"...access of inactive union field
.\tests.zig:342:12: 0x7ff62c89244a in test "simple union"
(test.obj)
result.float = 12.3;
```

Assuming you haven't used Zig (but, crucially – you do know how to read error messages): what does this tell you about Zig's type system? How would you compare it to C++'s – and in particular, do you think one is better than the other?

Type, Set, Math

7. Type relations are very important for mathematical operators in programming languages.

a) ** (8 min.) Recall that in Haskell, we've seen a variety of number types (well, type classes): `Int`, `Integer`, `Num`, `Float`, and `Fractional`. Since we didn't cover these in-depth in class, you may want to read A Gentle Introduction to Haskell's section on Numbers.

Which of these types are supertypes of other types? Which are subtypes? Which are not related at all?

b) ** (4 min.) The input and output types for Haskell's `div`, `mod`, and `+` are different, even though they all operate on numbers. Why?

Hint: Try running `:t div` and `:t +` in `ghci`).

c) ** (4 min.) C++ has `float`, `int`, `const float`, and `const int` (among other number types). Which of these are supertypes or subtypes of each other, and which are unrelated?

d) (4 min.) Give one example of an expression with a mathematical operation in any programming language where one of the input types is a subtype of the result's type. Briefly explain why.