Homework 4

Tejas Kamtam

305749402

CS 131 - Fall 2023

Problem 1

```
from functools import reduce
def convert_to_decimal(bits):
  exponents = range(len(bits)-1, -1, -1)
  nums = [bit*2**exp for bit,exp in zip(bits, exponents)]
  return reduce(lambda acc, num: acc + num, nums)
```

Problem 2

Part a

```
def parse_csv(l: list):
  return [(a,b) for a,b in [x.split(',') for x in l]]
```

Part b

```
def unique_characters(s: str):
  return {c for c in s}
```

Part c

```
def squares_dict(n: int, m: int):
  return {i:i**2 for i in range(1,m+1)}
```

Problem 3

```
def strip_characters(sentence, chars_to_remove):
  return ''.join([c for c in sentence if c not in
chars_to_remove])
```

Problem 4

```
def greet(world):
  def speak():
    return "Hello, " + world
  print(speak())

greet("Earth")
```

The closure above outputs `"Hello, Earth"` because the function `speak` is defined within the scope of `greet` and has access to the variable (encloses) `world` from the outer scope.

Problem 5

Part a

```
from math import sqrt
def nth_fibonacci(n: int) → int:
  phi: int = (1 + sqrt(5))/2
  psi: int = (1 - sqrt(5))/2
  return (phi**n - psi**n)/sqrt(5)
```

From a readability standpoint, you might interpret it to mean for the programmer, in which case Python annotations can be used to clarify what parameters *should* be passed in and used within the function implementation, as shown above. However, unlike Haskell, these type annotations are

not the same as Haskell type definitions as they do not coerce the inputs to the specific annnotated type - even outputs may not be the same as the annotation.

Part c

This is not the best annotation for a few reasons. Firstly, this annotation is more Haskell-esque than the code which is more Python-esque (if not directly from those languages). Secondly, the annotation is very ambiguous and does not clarify the type of the inputs and outputs distinctly.

Problem 6

Part a

The provided code tells us C++ is **weakly** typed because we are able to access a possible value of the union without checking if it's value has already been determined or defined (plus, we see some random number output so it hints at undefined behavior). We may also be able to say C++ is **statically** typed due to the clear type definitions of the union, but this is not as clear as the weak typing as we can't check whether the type definition is being adhered to or treated as an annotation for a reader (we know in C/C++ it is but it may be ambiguous in another language).

Part b

We can see that the Zig code throws an error specifically because we are trying to access an invalid memory location with the code, so we can say Zig uses **strong type checking** for memory safety when accessing undefined values of a union (but may not be a compeltely strong lanugage since

we need to check off all other criteria for strong typing). We cannot say much about whether or not Zig is statically or dynamically typed from the code provided, but hints at dynamic typing with the lack of clear type definitions.

Problem 7

Part a

Based on the link provided, `Num` is a superclass for `Int, Integer, Float, Double` are primitive subclasses and `Num` is a subclass of `Eq`. `Fractional` is a superclass of `Float, Double` and `Integral` is a superclass of `Int, Integer`.

Part b

It looks like `(+)` can operate on any number types in Haskell while `div` and `mod` can only operate on `Int,Integer` types. This isbecause Haskell division is integer division and so must only operate on integer types.

Part c

Each of `int,float,const float,const int` are all independent type classes. They are all derived from other superclasses like `null`. `const` is a modifier that modifies the `int` or `float` primitive to make it immutable.