Homework 1: DSSL2 Warmup

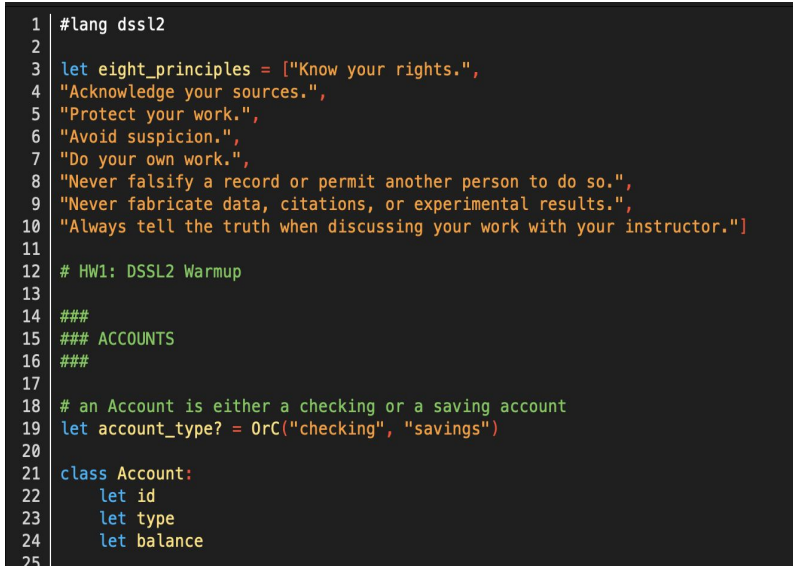
Welcome to 214!

Goals for this assignment

- Get used/comfortable writing DSSL2 code
  - Similar to Python but with some kinks (fundamentals still apply!)
  - Read the documentation
    - Yeah it sucks, but this is a skill and is very important when we start working with some of DSSl2 built in libraries
- Classes, Arrays, iterating
  - Will become very important for this class, get used to them now!
- Writing test cases
  - Extremely important to this class and many other CS classes that you take at Northwestern

Part I: Installing DSSL2

- The pdf already does a good job explaining this so I won't get into it
  - But if you have issues, get these resolved ASAP
- If you want DSSL2 in VScode on MacOS, come by my OH and I can try to help you get setup
  - Works just as fine as using DrRacket
  - If you're on M1 pro/max might be beneficial (I experienced many crashing issues)
- Alternatively, could use this DrRacket theme my friend and I made
  - Again, feel free to stop by my OH/email me if you want the steps to get your DrRacket to look like this



```
#lang dssl2

let eight_principles = ["Know your rights.",
"Acknowledge your sources.",
"Protect your work.",
"Avoid suspicion.",
"Do your own work.",
"Never falsify a record or permit another person to do so.",
"Never fabricate data, citations, or experimental results.",
"Always tell the truth when discussing your work with your instructor."]

# HW1: DSSL2 Warmup

###
### ACCOUNTS
###

# an Account is either a checking or a saving account
let account_type? = OrC("checking", "savings")

class Account:
    let id
    let type
    let balance
```

Part II: Class Practice

You're already given some useful "getter functions," use them!

"->" denotes a contract meaning that this is what the function **should** return

If None, then return nothing (don't need a return statement) unless you need to return an error (won't break contract)

When you have num? Or something similar to it, it just means that type should be there. i.e shouldn't be passing in a string into Account.deposit

Your Job:

- Account.deposit(num?) -> None
- Account.withdraw(num?) -> None
- Account.__eq__(Account?) -> bool?
- account_transfer(num?, Account?, Account?) -> None

Account.deposit(num?) -> None

- Add an specified amount to an account’s balance
  - This amount must be non-negative
  - Return an error if the passed in amount is invalid
    - return error(“your error message here”)
    - error(“your error message here”)
    - These could be anything, just needs to error

Account.withdraw(num?) -> None

- Subtracts passed in amount from an account’s balance
- Amount must be non-negative and the amount must not exceed the account’s current balance
- Otherwise, return an error

Account.__eq__(Account?) -> bool?

- Want to compare 2 different accounts, are all of their fields the same?
- If all 3 fields are the same -> True otherwise -> False

Account(5, "checking", 500) == Account(5, "checking", 500) -> True

Account(5, "savings", 500) == Account(5, "checking", 500) -> False

- I suggest using the code that you've written already/that was provided to you in the starter code

account_transfer(num?, Account?, Account?) -> None

- Withdraws a specified amount of money from one account, deposits that money into another account
- Again, amount transferred cannot be non-negative and cannot be greater than than the first Account’s balance (in the example, amount transferred cannot be > than account balance

let account1 = Account(2, "checking", 500)

let account2 = Account(3, "checking", 500)

account_transfer(250, account1, account2)

account1 now has a balance of 250, account2's balance is now 750

Part III: Customers

- Now kicking up difficulty but still manageable!
- Get feet wet with vectors and structs
  - Vectors are arrays, similar to Python list but are static, not dynamic
  - Essentially meaning that their length is of fixed size, can’t add more elements to an array/Vector that is full
  - No native “append” function!
  - Structs contain collections of data which you can access by its field names i.e customer.name

Your Job:

- max_account_id(VecC[customer?]) -> nat?
- open_account(str?, account_type?, VecC[customer?]) -> VecC[customer?]
- check_sharing(VecC[customer?]) -> bool?

max_account_id(VecC[customer?]) -> nat?

- Parameters: vector of customers
- Outputs: highest account id from that vector of customers (which is going to be natural number)
- Raise an error if no customers are given i.e max_account_id([]) -> error
- General approach:
  - Make a variable for a “winner” and assign it to nothing for now
  - Iterate through your customer array and get the id of that customer at that index
  - If greater than winner, reassign winner
  - Return winner

open_account(str?, account_type?, VecC[customer?]) -> VecC[customer?]

- Parameters: name of a customer, account type, and a vector of customers
- Outputs: New vector of customers with the new customer added to the vector
  - The new customer must have account with an initialized balance of 0, and an id +1 higher than the highest id in the customer vector… we just wrote something that could help us with this (if this account is first in our customer vector, it's id is set to 1
- General Approach:
  - Initialize a new vector of size +1 of the current customer vector
  - Figure out what the account id of the new customer should be and initialize the balance to 0 and type and name to whatever was passed in
  - Go through the previous customers in the passed in customer vector
  - Add the new customer at the very end
  - **Remember, array indexing is n-1**

Check_sharing(VecC[customer?]) -> bool?

- Parameters: Vector of customers
- Outputs: Boolean so True/False
- You want to check if any two accounts in the passed in customer vector have the same id (it doesn't matter if they don't have matching account types or balances)
- General Approach (brute force):
  - Use two for-loops, one that iterates over the vector of customers, another that iterates over the vector of customers as well but one over
    - So first for-loop is at vector[0] and the other is at vector[1] at the start
  - Check if the ids at this point are equal
  - **Be sure to stay within the bounds of the vector!**

That’s all for HW1!