Last Name: _____ First Name: _____ Cornell NetID, all caps: _____

CS 1110 Regular Prelim 1 Solutions March 2021

Wed Apr 7: the first line of the solution for Q3 (initialization of `back_sum` had been cut off; it is now restored).
early Monday Apr 5th: added another alternate solution to Q3

1. [6 points] **Lists.** For the following function, **student_netids** should be a list of 1000 unique strings corresponding to student netids, and **student_names** should be a list of 1000 strings.

   The two lists "line up": each item in `student_names` is the name for the student with netid in the corresponding location in `student_netids`.

   Using this information, complete the function below.

```
def get_name_from_netid(student_netids, student_names, spec_netid):
    """Returns the name of the student who has netid spec_netid.

    Preconditions:
        student_netids and student_names are as described in the question.
        spec_netid: string that appears exactly once in student_netids."""
    # STUDENTS: loops are NOT ALLOWED.

    i = student_netids.index(spec_netid)
    return student_names[i]
```

2. [9 points] **Strings.** Implement the following function.

```
def figlatin(s, k):
    """If s has length at least k+1, returns the string formed by adding in the
    string 'fig' just after the character at index k in s.

    Otherwise, returns the string formed by adding string 'fig' to the end of s.

    Examples:
        figlatin("012345", 3) --> "0123fig45"   figlatin("012345", 5) --> "012345fig"
        figlatin("012345", 0) --> "0fig12345"   figlatin("012345", 55)--> "012345fig"

    Precondition: k>=0 is an int.  `s` is a non-empty string."""
    # STUDENTS: WARNING: strings do NOT have an insert method the way lists do.
    #           Do NOT use loops; instead use string operations and methods.


    if len(s) <= k:
        return s + "fig"
    else:
        return s[:k+1] + 'fig' + s[k+1:]

    # ALTERNATE SOLUTION
```

```
    if len(s) > k:
        return s[:k+1] + 'fig' + s[k+1:]
    else:
        return s + "fig"

    # UNEXPECTED SOLUTION
    # This happens to work because Python evaluates list slices with out-of-bounds
    # indices to the empty list, but we can't say we approve.
    return s[:k+1] + 'fig' + s[k+1:]
```

3. [16 points] We define the **half-shift** of a list of integers *ol* to be a new list *hsl*, where the i^{th} item of *hsl* is the **float** that is $\frac{1}{2}$ of the sum of the entries in *ol* up to but not including the i^{th} entry of *ol*. (The sum of zero numbers is 0.)

   For example, suppose *ol* were [4, 5, 9, 2].
   Then, *hsl*[0] would be the float that is one-half of the sum of zero numbers, or 0.0.
   hsl[1] would be one-half of 4, or 2.0
   hsl[2] would be one-half of 4+5, or 4.5.
   And *hsl*[3] would be one-half of 4+5+9, or 9.0.
   In other words, *hsl* would be [0.0, 2.0, 4.5, 9.0], and one would change the four elements of *ol* to these values.

   Implement the following function.

```
def half_shift(ol):
    """Transforms the entries in `ol` so that `ol` becomes the half-shift of
    what it used to be. (Does not return anything.)

    Preconditions: ol is a nonempty list of positive integers."""
    # STUDENTS: You must use for-loops effectively.
    #           You are allowed to create new lists in your solution.
    #           You may *not* use the sum() function.

    back_sum = 0.0
    for i in range(len(ol)):
        new_val = back_sum/2  # Save value to eventually write into ol[i]

        # Prepare for next loop
        back_sum += ol[i]
        ol[i] = new_val

    return # Not necessary except to keep the alt. solution below from also running.

    # ALTERNATE SOLUTION

    back_sum = 0.0 # sum of number up to but not including current point
```

```
    for i in range(len(ol)):
        curr = ol[i] # Save value to eventually add into back_sum
        ol[i] = back_sum/2

        back_sum += curr  # Prepare for next round of loop

    return # Not necessary except to keep the alt. solution below from also running.

    # ALTERNATE SOLUTION that for safety, doesn't overwrite `ol` until after all
    # the computation is done.

    # Set up the case of i being 0
    back_sum = 0.0 # sum of numbers in ol[:i]
    hsl = []  # half-shift of ol[:i], or empty list if ol[:i] is empty.

    for i in range(len(ol)):
        hsl.append(back_sum/2)

        back_sum += ol[i]  # Set up for next loop round

    # Copy items in hsl into ol
    for i in range(len(ol)):
        ol[i] = hsl[i]


    # ALTERNATE SOLUTION - "preload" hsl with the extra starter value
    back_sum = 0.0
    hsl = [0.0] # pre-load hsl with an extra value, so in the ith iteration
                # of the loop below, we will be adding to hsl[i+1].
    for i in range(len(ol)-1):
        back_sum += ol[i]
        hsl.append(back_sum/2)

    for j in range(len(hsl)):
        ol[j] = hsl[j]
```

Note that `ol = hsl` would *not* change the input list, but would instead only affect the local parameter `ol`.

4. [26 points] The Dairy Bar has coupons for free ice cream! An ice cream coupon is an object whose attributes are `flavor` and `size`. Assume that a call of the form `Coupon(f, s)` creates a new Coupon with attribute `flavor` set to `f` and attribute `size` set to `s`. And assume that class Coupon is accessible from some previous import.

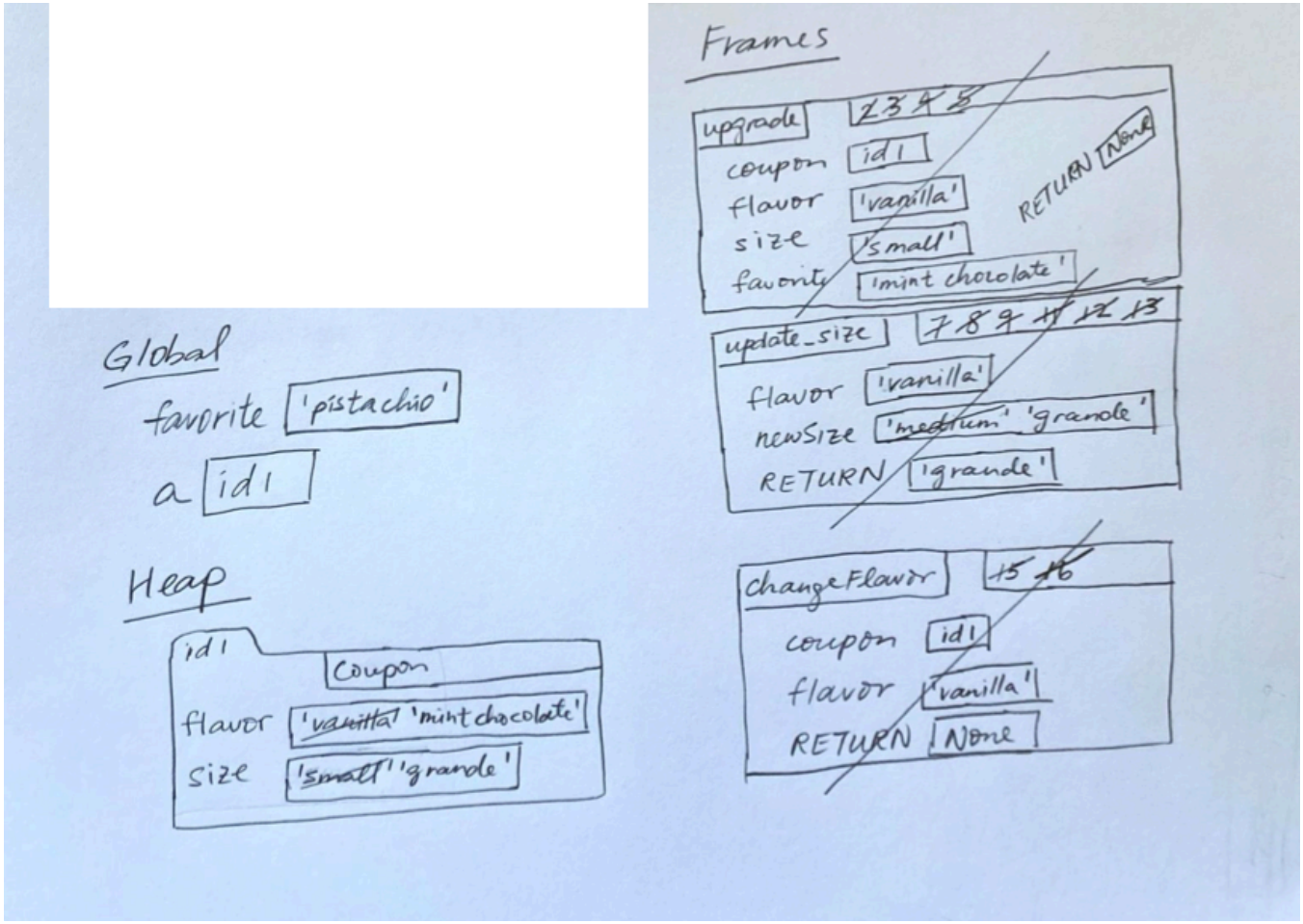
   Run the entire 19 lines of code below and draw the appropriate call frames, the heap space and global space, using the notation from class and Assignment 2.

Last Name: _____ First Name: _____ Cornell NetID: _____

```
def upgrade(coupon, flavor, size):
  if size == 'small':
      coupon.size = update_size(flavor)
  changeFlavor(coupon, flavor)
  favorite = coupon.flavor


def update_size(flavor):
  if flavor in ['vanilla', 'chocolate']:
      newSize =  'medium'

  if flavor == 'strawberry':
       newSize =  'large'
  else:
       newSize = 'grande'
  return newSize
```

```
def changeFlavor (coupon, flavor):
   if flavor == 'vanilla':
     coupon.flavor = 'mint chocolate'

favorite = 'pistachio'
a = Coupon('vanilla', 'small' )
upgrade(a, a.flavor, a.size)
```

Last Name: _____ First Name: _____ Cornell NetID: _____



5. [6 points] **Test cases.**

Consider the following function specification.

```
def tradeElems(list1, list2, repVal):
    """
    list1 and list2 are same-length lists of integers. repVal is an integer.

    At every index where 'repVal' appears in either list, swap the corresponding
    elements of list1 and list2.    """
```

We've given an example of one set of sample inputs and expected output below.

Provide **two** more conceptually distinct test cases, using the same format. Include a short statement (1-2 sentences) explaining what situation each of your test cases represents.

	First Test Case
list1:	[1, 2, 3, 1]
list2:	[5, 3, 1, 2]
repVal:	1
Expected **list1**:	[5, 2, 1, 2]
Expected **list2**:	[1, 3, 3, 1]

Some possibilities:

```
#Test list of length 1
list1 = [1]
list2 = [3]
list1Expected = [3]
list2Expected = [1]
repVal = 1

#Test with no swaps
list1 = [2, 3, 4, 5]
list2 = [5, 4, 3, 2]
list1Expected = [2, 3, 4, 5]
list2Expected = [5, 4, 3, 2]
repVal = 1
```

Or, empty lists.

6. **Object access.**

Assume objects of new class Academy have four attributes: string `name`, and three lists of ints (standing for student tag numbers, like in Assignment 3): `accepted`, `rejected`, and `waitlisted`.

(a) [7 points] Suppose `a` is an Academy object whose three attribute lists are all empty.

And, suppose that `slist` is a list of 100 unique ints.

Write code that performs each of the following actions. (Don't write a function header or assume one is given.)

Add int `12` to `a`'s `accepted` list.
Set `a`'s `waitlisted` to the list `[4, 7, 9]`.
Set `a`'s `rejected` list to be a list of the items in `slist` starting from the item at index 4 and up to and *including* the item at index 58.

```
a.accepted.append(12)
a.waitlisted = [4, 7, 9]
a.rejected = slist[4:59]
```

(b) [14 points] Implement the following function.

(You don't need any more info about the `pair` object beyond what's given.)

```
def print_j(pair, j):
    """ pair is an object with two attributes, a1 and a2, which are
    both Academy objects (not None).

    If both Academy objects in pair have at least j+1 accepted students,
        print the int at index j in the first Academy's accepted list,
        print the int at index j in the second Academy's accepted list,
        and return True.
    Otherwise, return False. (And don't print anything.)

    Preconditions: pair is as described above;  j >= 0 is an int. """

    if len(pair.a1.accepted) >= j+1 and len(pair.a2.accepted) >= j+1:
        print(pair.a1.accepted[j])
        print(pair.a2.accepted[j])
        return True
    return False


    # ALTERNATE SOLUTION
    if len(pair.a1.accepted) <  j+1  or len(pair.a2.accepted) < j+1:
        return False
    else:
        print(pair.a1.accepted[j])
        print(pair.a2.accepted[j])
        return True
```

(c) [3 points] Implement the following function.

```
def print_k(thelist, k):
    """thelist is a list of at least k+1 Academy objects, each of which
    has at least one accepted student.

    Prints the first int in the accepted list of the Academy
    at index k in thelist."""

    print(thelist[k].accepted[0])
```

---

7. [1 point] **Fill in your last name, first name, and Cornell NetID at the top of each page.**

   Also, remember that you should not discuss this exam with students who are scheduled to take a later makeup.

   Always do this! It prevents disaster in cases where a staple fails.