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

CS 1110 Prelim 1, March 2018

This 90-minute exam has 7 questions worth a total of 69 points. You may separate the pages while working on the exam; we have a stapler available.

The second page of this exam gives you the specifications for some useful functions and methods.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help.
We also ask that you not discuss this exam with students who are scheduled to take a later makeup.
Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: _____ Date _____

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

`s.find(substr)`	Returns: index of first occurrence of string `substr` in string `s` (-1 if not found)
`s.strip()`	Returns: copy of string `s` where all whitespace has been removed from the beginning and the end of `s`. Whitespace not at the ends is preserved.
`s.split(sep)`	Returns: a list of the "words" in string `s`, using `sep` as the word delimiter (whitespace if `sep` not given)
`s.join(slist)`	Returns: a string that is the concatenation of the strings in list `slist` separated by string `s`

`lt.insert(i,item)`	Insert `item` into list `lt` at position `i`
`lt.append(item)`	Adds `item` to the end of list `lt`
`lt.count(item)`	Returns: count of how many times item occurs in list
`list(range(n))`	Returns: the list `[0 .. n-1]`
`lt.remove(item)`	Removes the first occurrence of `item` from list `lt`; raises an error if `item` not found.
`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` not found. (There's no "find" for lists.)
`lt[i:j]`	Returns: A new list `[lt[i], lt[i+1], ..., lt[j-1]]` under ordinary circumstances. Returns `[]` if `i` and `j` are not both sensible indices.
`lt.pop(i)`	Returns: element of list `lt` at index `i` **and also removes that element from the list** `lt`. Raises an error if `i` is an invalid index.

`list(map(func, lt))`	Returns: A list obtained by applying function `func` to each element in list `lt` and concatenating the results of each application.

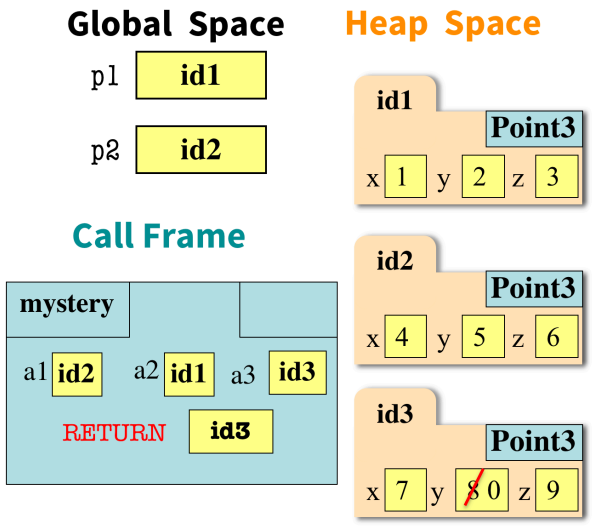
Question	Points	Score
1	7	
2	15	
3	12	
4	21	
5	8	
6	5	
7	1	
Total:	69	

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

1. [7 points] **What's the point?**

   Consider the Point3 class as it was defined in lecture, with 3 attributes: `x`, `y`, and `z`. Complete the code so that it will create the following memory diagram. Do not change any of the code provided, but (1) fill in the arguments to the incomplete call to `mystery` on line 18 and (2) complete the body of the function `mystery`. The third point on the Heap can be initialized with integers; it **does not** need to be a calculation involving the attributes of the first two points.

```
1  import shapes
2
3  p1 = shapes.Point3(1,2,3)
4  p2 = shapes.Point3(4,5,6)
5
6  def mystery(a1, a2):
7      # TASK #2 complete definition of mystery
8      # use as many lines as you need
9
10
11
12
13
14
15
16 # TASK #1: complete the call to mystery
17
18 mystery(    ,     )
```



2. **Make the grade!** A talented but morally questionable CS 1111 student has written a script `make_my_grade()` that takes a list of lab grades, which is a list of ints, and changes them. First, every grade that is 0 is turned into a 5. Then all grades are doubled.

   (a) [6 points] Write **three conceptually distinct test cases** for function `make_my_grade()`.

   Test case #1
   Input and expected output:

   Rationale:

   Test case #2
   Input and expected output:

   Rationale:

   Test case #3
   Input and expected output:

   Rationale:

(b) [7 points] Lab grades in CS 1111 are at most 10 points. Anything higher than 10 will raise a red flag and alert the staff that something fishy is going on, as will too many 10s.

Since you are also talented but morally questionable, you decide to write a helper function to be used by the previous script. It takes two lists (the original grades, `original`, and the doctored grades, `doctored`) and an index `i`.

If the element at index `i` in `doctored` is larger than 10, *and* there are no more than *three* 10s in `doctored`, then replace the element at index `i` in `doctored` with a 10.
Otherwise, replace the element of `doctored` at index `i` with the corresponding score from `original`.
Nothing is returned.

Implement the function as described, ignoring the need for any preconditions for now.

```
def check_grade(original, doctored, i):
```

(c) [2 points] Preconditions are often stated because if one violated them, the function would raise an error.

What is **one** precondition you should add to the specification of the function above? In other words, what condition (if violated) would cause your implementation to raise an error, and what kind of error would that be? (You do not have to give the exact name of the error; just describe it.)

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

3. **Come on, get happy!**

   (a) [11 points] **What does the Call Stack look like?**

   The Dutch version of "Happy Birthday" says "Long shall he/she live in glory". The first half of the sentence is repeated 3 times, then the second half of the song is repeated 3 times. Below on the left is the (error-free) code which prints out this Dutch song. On the right, draw the full call stack as it would look when **line 5 has just been executed** for the **second** time. Include crossed-out frames.

Call Stack

```
1  def he():
2      print("hij")
3
4  def she():
5      print("ze")
6
7  def live_long(is_female):
8      print("Lang zal")
9      if is_female:
10         she()
11     else:
12         he()
13     print("leven")
14
15 def in_glory():
16     print("In de gloria")
17
18 def song(is_female):
19     """ Happy Birthday in Dutch"""
20     for verse in list(range(3)):
21         live_long(is_female)
22     for verse in list(range(3)):
23         in_glory()
24
25 song(True)
```

(b) [1 point] **What went wrong?**

Below on the left is code with one or more errors. Below on the right is the error message that is printed when the code is run. Fix the **one** line of code that led to the error message in the traceback. (**Fix only the error reported in this traceback.**)

```
1  def happy_holiday(holiday):
2      print("Happy "+holiday)
3
4  def dear():
5      print("Dear "+name)
6
7  def to_you():
8      print("to "+"you")
9
10 def line_with_name(name):
11     happy_birthday(name)
12     dear(name)
13
14 def basic_line(holiday):
15     happy_holiday(holiday)
16     to_you()
17
18 def song():
19    basic_line("Birthday")
20    basic_line("Birthday")
21    line_with_name("Teo")
22    basic_line(200)
23
24 song()
```

```
Traceback (most recent call last):
  File "happy_error.py", line 24, in <module>
    song()
  File "happy_error.py", line 21, in song
    line_with_name("Teo")
  File "happy_error.py", line 11, in line_with_name
    happy_birthday(name)
NameError: name 'happy_birthday' is not defined
```

4. [21 points] Complete `edit(instring, c1, c2)` below so that it obeys the following specification. (Don't include a docstring.)

   Preconditions: `instring` is a (possibly empty) string; `c1` and `c2` are two different strings, both of length 1.

   If `instring` does not contain any `c1`s, return `False`

   if `instring` contains 2 or more `c1`s,
   or contains exactly 1 `c1` but no `c2`,
   or contains exactly 1 `c1` and a `c2` before the `c1`, return the int `0`.

   Otherwise, return a version of `instring` where the text between `c1` and the first `c2` after it, inclusive, has been removed.

   Examples:

instring	c1	c2	what to return
"exam"	'('	')'	False
"B. (Bats) Wayne, M. (Prez)"	'('	')'	0
"(:"	'('	')'	0
"1) CS1110 (Intro)"	'('	')'	0
"T'Challa,(B.P.)x yz"	'('	')'	"T'Challa,x yz"
"a.().a"	'('	')'	"a..a"
"a () a"	'a'	'b'	0

```
def edit(instring, c1, c2):
```

5. [8 points] Recall from A2 that Player objects have an int attribute `holdings`. So, if `p` were a variable storing (the identifier of) a Player object, we could access the value of its `holdings` atttribute with the expression `p.holdings`.

   Suppose another class, `Team`, has been defined, where each Team object has an attribute `plist` that is a non-empty list of Player objects.

   Imagine someone asks us to write a function `switch(t1, t2, i1, i2)` with the following specification.

   > *Preconditions*: `t1` and `t2` are Teams. (They are allowed to be the same Team.)
   > `i1` is a valid index for `t1`'s list of Players, and `i2` is a valid index for `t2`'s list of Players.
   >
   > Swaps the `holdings` of the Player at index `i1` in `t1`'s list of players with the `holdings` of the Player at index `i2` of `t2`'s list of players. Does not return anything.

   Is it possible to write such a function?

   - If yes, write the body of the function below.
   - If not, explain why this is not possible, in 3-5 sentences, in the space below. You may use folder/call-frame diagrams in your explanation. *And*, write a line of code that would store in a variable `temp` the `holdings` of the Player at index 0 of the player list of Team `t1`.

   Hints: You may wish to draw object diagrams to make sure you understand the setup of the classes and lists involved. Try writing the function to decide whether or not it can be implemented.

```
def switch(t1, t2, i1, i2):
```

6. [5 points] **A Lannister always pays his (or her) debts.**

   You are asked to complete a function (on the next page) that creates a payment plan for an amount owed. The amount owed is divided into a number of installments. After the first payment, a fee is assigned. The fee is a percentage of the installment amount and increases each month: no fee in the first month, 10% of the installment in the second month, 20% of the installment in the third month, and so on.

   Sample output for a few test cases are shown below:

```
>>> payment_plan(1000,1)
Each installment = $1000.0
#1: 1000.0
Total fees charged: $0.0

>>> payment_plan(1000,2)
Each installment = $500.0
#1: 500.0
#2: 550.0
Total fees charged: $50.0

>>> payment_plan(1000,3)
Each installment = $333.33
#1: 333.33
#2: 366.67
#3: 400.0
Total fees charged: $100.0

>>> payment_plan(1000,4)
Each installment = $250.0
#1: 250.0
#2: 275.0
#3: 300.0
#4: 325.0
Total fees charged: $150.0
```

   The `print` statements are already in the code and require you to create and give correct values to the variables `installment`, `curr_payment`, and `total_fees`.

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

```
def payment_plan(amount, num_payments):
    """prints out a payment plan
    amount: total amount owed (a float)
    num_payments: # of installments (an int)

    After the first payment, a fee is is assigned.
    The fee is a percentage of the installment amount and increases each month:
    - no fee in month 0
    - 10% of the installment in month 1
    - 20% of the installment in month 2, etc.

    The function also prints out the total amount of fees one will have paid at the end
    of the payment plan."""

    # STUDENTS: initialize variable installment here with an assignment statement


    print("Each installment = $"+str(round(installment,2)))

    # STUDENTS: Initialize accumulator variable total_fees here with an assignment



    # which_payment will have values 0 then 1 then 2...
    # DO NOT CHANGE THE FOR-LOOP STRUCTURE GIVEN
    for which_payment in list(range(num_payments)):
        # STUDENTS: Compute curr_payment and create/update other variables as appropriate









        print("#"+str(which_payment+1)+": "+str(round(curr_payment,2)))

    print("Total fees charged: $"+str(total_fees))
```

---

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

Did you write your name and netID on each page, re-read all specs, and check that your code works against test cases? Then you're good to go!