

Spring 2019 CS 1110 Prelim 1

Please turn off and stow away all electronic devices. You may not use them for any reason during the exam. Do not bring them with you if you leave the room temporarily.

This is a closed book and notes examination. You may use the reference sheet provided.

There are **7 problems**. Make sure you have the whole exam.

You have **90 minutes** to complete 90 points. Use your time accordingly.

Question	Points	Score
1	8	
2	12	
3	17	
4	15	
5	13	
6	14	
7	11	
Total:	90	

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 _____

Name: _____ NetID _____

1. Start Your Engines

(a) [2 points] Python is a *dynamically typed* language. This means that a variable can hold values of any type and a variable can hold different types at different times. Write 2 lines of Python code that illustrate this.

(b) [2 points] A type is defined as “a set of values and operations on these values”. The meaning of an operator can change depending on the type. Give an example of an operator that has two *distinct* meanings when used in the context of two different types. What are these two meanings?

(c) [2 points] If Python has just executed line 6 and is about to execute line 7, what does the call stack look like?

```

1 def f3():
2     print("f3")
3
4 def f2():
5     print("f2")
6     f3()
7     f3()
8     f3()
9
10 def f1():
11     print("f1")
12     f2()
13
14 f1()
        
```

	A	B	C	D	E
	f1	f1	f1	f1	f1
	f2	f2	f2	f2	
	f3	f3	f3		
	f3	f3			
	f3				

Your Answer: _____

(d) [2 points] After the following lines of code have been executed:

```

x = [0, 1, 2, 3, 4]
z = x
y = x[1:4]
z[y[1]] = y[2]
        
```

What does `x[2]` evaluate to? Your Answer: _____

2. [12 points] **The Sticking Point**

Consider the Point3 class as it was used in lecture, with 3 attributes: x, y, and z.

```
1 import shapes
2
3 def stick(p1, p2):
4     p2.y = p1.x
5     y = x
6     z = y + x
7     return z
8
9 x = 1
10 y = 2
11
12 p1 = shapes.Point3(3,4,5)
13 p2 = shapes.Point3(6,7,8)
14 p3 = p1
15 p1.z = 9
16
17 x = stick(p2, p1)
```

Lines 1-17 execute without any error. After they are executed, what would the following python expressions evaluate to?

(a) x

(b) y

(c) z

(d) p1.y

(e) p2.y

(f) p3.y

3. **Above your pay grade.** iClicker software creates a list of iClicker scores for each student. Each score is the iClicker participation points for one lecture. If a student is *absent*, the software does **not** enter a zero; the student receives no score for that day, making the length of the `scores` list shorter than the total number of lectures. To calculate each student's average iClicker score, Professor Bracy wants a zero added to the `scores` list for each missed lecture. She implements a function `make_complete_scores(scores, num_lectures)` which takes `scores`, a (possibly empty) list of floats with values > 0.0 but ≤ 2.0 that represent a student's iClicker grades, and `num_lectures`, a non-negative integer that represents the current number of lectures in the semester. `num_lectures` \geq the length of `scores`. (Since each lecture is worth at most 2 points, the ordering of the `scores` list doesn't matter.) It returns a list of scores of length `num_lectures`, with zeroes explicitly present at the end of the list for missed lectures.

(a) [9 points] Write **3 conceptually distinct test cases** for `make_complete_scores(scores, num_lectures)`. Make sure your input values are ordered `scores, num_lectures`.

#1: Input:

Expected output:

Rationale:

#2: Input:

Expected output:

Rationale:

#3: Input:

Expected output:

Rationale:

- (b) [6 points] The famous clock maker, Timex, would like to donate an alarm clock to anyone who seems to be struggling to make it to class. To identify students who would benefit from an alarm clock, the head TA for CS 1110 writes a function `needs_alarm(complete_scores, num_lectures)`. For this function, `complete_scores` will represent a student's iClicker scores fully constructed from `make_complete_scores()`. The second parameter `num_lectures` is identical to that used in `make_complete_scores()`. This function will return a `bool`; `True` if the student has missed between $1/3$ and $2/3$ (exclusive) of the `num_lectures` and `False` otherwise. (If they skip $2/3$ or more of the lectures, they probably just need more sleep, not an alarm clock.)

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

```
def needs_alarm(complete_scores, num_lectures):
    """checks to see if the student could benefit from an alarm clock
    Returns: True if the student has missed between 1/3 and 2/3 of all
    lectures. (a missed lecture is indicated by a score of 0)
    Otherwise, returns False"""
```

- (c) [2 points] 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 either behave incorrectly or raise an error?

4. [15 points] **Stack Attack!** Aliens are invading the world. Captain Dan needs your help to save humanity. Are you up for the challenge? A Captain has two attacks, each attack is its own object with a name and damage attributes. The damage attribute determines how many aliens it can kill. The code below has begun executing, resulting in the memory diagram on the right. **Execute the code to completion**, beginning at the line indicated in the current Call Frame. Update existing variables and objects and draw new variables and call frames as needed. If you cross out a value or call frame, make sure it is still legible.

```

1 def fight(attack):
2     d = attack.damage
3     if d <= n_alien:
4         return d
5
6 def check_victory():
7     if n_alien <= 0:
8         return True
9     return False
10
11 def defend_universe(cap, n_alien):
12     kills = 0
13     if n_alien >= 2:
14         kills = fight(cap.attack1)
15     else:
16         kills = fight(cap.attack2)
17     n_alien = n_alien - kills
18     victory = check_victory()
19     if victory:
20         print("WE WON!")
21
22 n_alien = 3
23 a1 = Attack("tackle", 2)
24 a2 = Attack("bodyslam", 1)
25 c = Captain("Dan", a1, a2)
26 defend_universe(c, n_alien)

```

Global Space

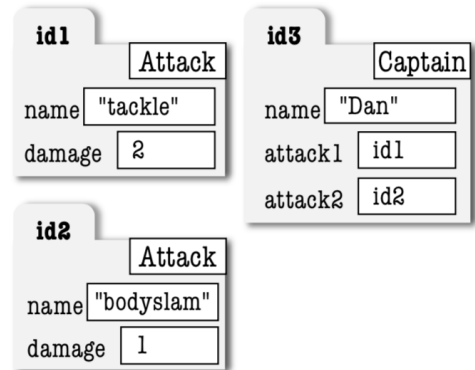
n_alien 3

a1 id1

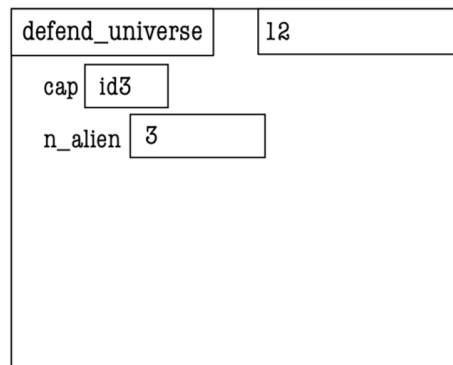
a2 id2

c id3

Heap Space



Call Frames



5. **What's the frequent problem, Kenneth?** Fix the errors in the code below. If you change a function *definition*, please update the calls to that function, as necessary.

```
1 def fight(attack):
2     d = attack.damage
3     if d <= n_alien:
4         return d
5
6 def check_victory():
7     if n_alien <= 0:
8         return True
9     return False
10
11 def defend_universe(cap, n_alien):
12     kills = 0
13     if n_alien >= 2:
14         kills = fight(cap.attack1)
15     else:
16         kills = fight(cap.attack2)
17     n_alien = n_alien - kills
18     victory = check_victory()
19     if victory:
20         print("WE WON!")
21
22 n_alien = 3
23 a1 = Attack("tackle", 2)
24 a2 = Attack("bodyslam", 1)
25 c = Captain("Dan", a1, a2)
26 defend_universe(c, n_alien)
```

The code from the previous page is copied here for your convenience.

- (a) [4 points] The function `fight` sometimes triggers a Python error. Explain why and fix the problem by modifying the definition of `fight`.

- (b) [4 points] Even when Captain Dan kills all the aliens, the code never prints "WE WON!". Explain why and fix the problem by modifying the definition of `check_victory`.

```
1 def name(s):
2     print(s + " had a " + lamb())
3
4 def lamb():
5     print("little lamb")
6
7 def fleece():
8     print("Its fleece was white as snow")
9
10 def sing():
11     for i in range(2):
12         name(person)
13         if i == 0:
14             for j in range(2):
15                 lamb()
16         fleece()
17
18 person = "Mary"
19 sing()
```

- (c) [5 points] The code to the left should print out the following lyrics:

```
Mary had a little lamb
little lamb
little lamb
Mary had a little lamb
```

```
Its fleece was white as snow
```

Instead, it throws an error. Explain why and fix code that causes the problem. Note: the for-loops (lines 11 and 14) are correct.

6. [14 points] **Hang in there!** In the game *Hangman*, a player must guess a hidden word in some number of guesses. At first, each letter is shown as a '_'. As the player correctly guesses the letters in the word, they are revealed. Complete the function `process_guess(hidden, shown, guess, guesses_left)` below so that it obeys the following specification in support of the game hangman. (Don't include a docstring.)

Preconditions:

- `hidden` is a string with length ≥ 1 with only lower-case letters and **no repeating letters**
- `shown` is a string identical to `hidden`, but 1 or more (not guessed) letters are replaced by '_'
- `guess` is a lower-case character
- `guesses_left` is an int ≥ 1

`process_guess()` should:

(1) print "YOU WIN!" or "YOU LOSE!" when applicable:

- If after this `guess`, the whole hidden word is now known/shown, the player has won.
- If the player didn't just win the game and they only had 1 `guesses_left`, then they lose.

(2) return the string `shown`, updated in response to `guess`:

- If `guess` is **not** a letter in `hidden`, return `shown`.
- If `guess` is a letter in `hidden`, return `shown` but with the '_' corresponding to that letter replaced with `guess`.

Examples:

<code>hidden</code>	<code>shown</code>	<code>guess</code>	<code>guesses_left</code>	what to print	what to return
"world"	"_ _ _ _ _"	'o'	6		"_o_ _ _"
"world"	"_o_ _ _d"	'e'	4		"_o_ _ _d"
"world"	"_o_ _ _d"	'o'	3		"_o_ _ _d"
"world"	"_o_ _ _d"	'r'	1	"YOU LOSE!"	"_or_ _d"
"world"	"w_rld"	'o'	1	"YOU WIN!"	"world"
"world"	"worl_ "	'd'	4	"YOU WIN!"	"world"

```
def process_guess(hidden, shown, guess, guesses_left):
```


7. [11 points] **Home is where the Address folder specifies.**

Consider an `Address` class with the attributes:

- `num`: an `int` representing the street number
- `street`: a `str` representing the street name
- `city`: a `str` representing the city name
- `zip`: a `str` representing the zip code

If `a1` were a variable storing (the identifier of) an `Address` object, we could access the value of its `city` attribute with the expression `a1.city`

Consider a second class, `Contact`, with the attributes:

- `name`: a `name` representing a person's name
- `home`: the identifier of an `Address` representing where they live
- `work`: the identifier of an `Address` representing where they work

If `c1` were a variable storing (the identifier of) a `Contact` object, we could access the value of its `home` attribute (which stores (the identifier of) an `Address` object) with the expression `c1.home`

You may wish to draw object diagrams to make sure you understand the setup of the classes and attributes involved.

Your task is to write two functions `work_together(c1, c2)` and `live_together(c1, c2)` with the following specifications:

`live_together(c1,c2):`

Preconditions: `c1` and `c2` are `Contacts` with distinct names and non-empty `home` addresses.
Returns `True` if `c1` and `c2`'s `home` addresses are the same. Return `False` if they differ.

`work_together(c1,c2):`

Preconditions: `c1` and `c2` are `Contacts` with distinct names and non-empty `work` addresses.
Returns `True` if `c1` and `c2`'s `work` addresses are the same. Return `False` if they differ.

Two addresses are considered the same if all four attributes are equal.

Notice that these two functions have *almost* the same functionality. Instead of writing two separate functions that have a large overlap in behavior (*redundancy is bad!*), define a helper function that these two functions can both call to accomplish their overlapping work.

```
def _____ ( _____ ):

    """
    Inputs & Preconditions:

    Functionality: What does it do/print/return/modify ?

    """
    # Your helper function definition goes here
```

```
def live_together (c1, c2):
    """ c1 and c2 are Contacts with distinct names and non-empty home addresses
    Return True if c1 and c2's home addresses are the same, False otherwise
    """
```

```
def work_together (c1, c2):
    """ c1 and c2 are Contacts with distinct names and non-empty work addresses
    Return True if c1 and c2's work addresses are the same, False otherwise
    """
```