CIT-590 Midterm
Fall 2017

Name:______________________________________________________
Pennkey (not number):_________________________________

Please read the following and sign:

- I agree to not discuss the contents exam with other students until cleared by the instructor to do so. I realize that other students have not taken the exam yet, and I will not provide academically dishonest assistance to them.
- I have received no advanced knowledge of the contents of this exam.
- I will not use any notes, calculators, cell phones or electronic devices on this exam.
- **MY CELL PHONE IS TURNED OFF AND SILENCED.**
- I acknowledge that my answer must go in the space provided, and that notes on the back of the page will not be graded. If I run out of space, I will write a note saying "See the extra back page" where I will write my answer. My answer must go on the front of this page.
- I agree to all of the following under penalty of Penn’s policies regarding academic honesty.

Sign:_____________________________________________________ Date: ______________

Do not flip to the next page until given permission by the instructor.
1) Just warming up [15 points – 3 points each].

Answer each of the following questions in a few sentences.

a) Briefly explain how Test-Driven Development works.

b) What is the benefit of Test-Driven Development as opposed to a program that has unit tests but did not use Test-Driven Development?

c) What are the pros and cons of the garbage collector?

d) What is the difference between a local and a global variable?

e) Name one disadvantage of a linked list as opposed to a standard Python list.
2) Think like a Python [15 points – 1.5 points per box].
What will each segment of code print, and what is the value of x at the end? If this generates an error, say Error under Print and why it's an error under X. Use the first two as a guide.

<table>
<thead>
<tr>
<th>Code Segment</th>
<th>What prints</th>
<th>Value of X</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = &quot;cat&quot;</code></td>
<td></td>
<td>&quot;catdog&quot;</td>
</tr>
<tr>
<td><code>x = x + &quot;dog&quot;</code></td>
<td>&quot;catdog&quot;</td>
<td>&quot;catdog&quot;</td>
</tr>
<tr>
<td><code>import math</code></td>
<td>ERROR</td>
<td>Cannot take sqrt of negative number</td>
</tr>
<tr>
<td><code>x = math.sqrt(-5)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;this never prints&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>a = 6</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>b = 2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = 2 * a / b ** 2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(a/b)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>lst1 = [[1, 2], 5]</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>lst2 = [2, 3]</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = lst1[-2]</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(lst1 + lst2)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = &quot;this test is easy&quot;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x[13:16] = &quot;hard&quot;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(x[5:])</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>import math</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>a = math.sqrt(10)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = a ** 2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>assert x == 10</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(a)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = []</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x.append({2: &quot;two&quot;})</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x.append({3: &quot;three&quot;})</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(x[0][2])</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3) Life is a script 😊 [15 points].
Remember that Python is a scripted language, so there is no type checking until a line is actually executed. Suppose I write the following function:

```python
def modify_collection(col):
    for index in range(len(col) - 1):
        col[index] = col[index] + col[index + 1]
```

Now suppose I call modify_collection with each of the following five collections. Write down what each collection looks like after modify_collection is called. If the function throws an error for some reason, say why [3 points each].

a) [1, 2, 3, 4]

b) ["a", "b", "c", "d"]

c) "abcd"

d) {2: "two", 1: "one", 0: "zero"}

e) (1, 2, 3, 4)
4) I never get movie references [15 points].
Suppose I have compiled a movie database and stored it in the global dictionary below. Please help me figure out some important information about my movies by filling out the code for each of the two functions below [7.5 points each].

```python
actor_db = {"Harrison Ford": ["Indiana Jones", "Star Wars"],
           "Leonardo DiCaprio": ["Inception", "Blood Diamond", "The Beach"],
           "Joseph Gordon-Levitt": ["Inception", "The Dark Knight Rises"]}
```

def all_movies_in_common(actor1, actor2):
    """Returns a list of all movies with both actor1 and actor2 (assume inputs are strings) """
    # Your code goes here.

def actor_with_most_movies():
    """Returns the name of the actor who has played in the most movies"""
    # Your code goes here.
5) But I do get football references [20 points].

Now suppose I also like to keep track of some famous football players. For each player, I keep track of their age, number of goals scored, teams they have played for, and name. When a player moves to a different team, I call the transfer_player function. Also, for each team, I keep track of all of their present and past players and all of these players' teams (in player_history). Please answer the questions after the code below.

```python
import copy
def transfer_player(player, team):
    """Returns a new list for player, with team added to player’s list of teams.""
    new_player = copy.copy(player)
    new_player[2].append(team[0])
team[1].append(new_player)
return new_player

def player_history(team):
    """Returns a dictionary where the keys are all players who have ever played for the team and the values are all teams they have ever played for""
    players = {}
    for player in team[1]:
        players[player[3]] = player[2]
    return players

def main():
    messi = [30, 532, ["Barcelona"], "Messi"]
    weird_player = [25, 249, ["Santos", "Barcelona"], "Neymar"]

    barca = ["Barcelona", [messi, weird_player]]
    psg = ["Paris Saint-Germain", []]

    transfer_player(weird_player, psg)
    print(player_history(barca))

if __name__ == "__main__":
    main()
```
a) Please draw the state of the stack and heap before return new_player is executed. You need to draw the state of player and new_player (assume strings fit in one box). You can ignore team. (Hint: I have drawn all the memory “boxes” you need to use.) [10 points].

Stack

Heap

b) What does print(player_history(barca)) print [3 points]?

c) What would print(player_history(barca)) print if I had done a deep copy instead of a shallow copy [3 points]?

d) In general, what is the difference between a deep copy and a shallow copy [2 points]?

e) When is the if-case if __name__ == "__main__" true [2 points]?
7) You gotta have class [20 points].
Finally, suppose I also like to keep digital track of my (imaginary) dogs. Look at the class definitions below and answer the following questions.

class Animal():
    def __init__(animal, age, name):
        animal.age = age
        animal.name = name

class Dog(Animal):
    def __init__(self, age, name = "Even dogs know Ronaldo > Messi"):  # 3 points
        Animal.__init__(self, age, name)

def main():
    my_dog = Dog(1)

a) Why do I need to use Animal in Animal.__init__(self, age, name) [3 points]?

b) How would I have to change the __init__ method in Animal if I want to use super().__init__(age, name) [3 points]?

c) What is the difference between methods and functions [2 points]?

d) Why can I just provide one parameter when I create my_dog in main() [2 points]?
e) As is clear from their class definition, dogs like to work with numbers. They also like to change their names in weird ways by playing with numbers. Please implement the method below (you can assume it is inside the Dog class) [10 points + 5-point bonus].

def change_name(self):
    """Changes the name of the Dog object in the following way: if the name is a single word, then it doesn’t change; if the name is more than one word, the new name is a string containing the words in odd positions in the old name (for example, if the dog’s name was the default name, the new name would be "dogsRonaldoMessi")""

    # Your code goes here.
    # Bonus 5 points: change the name so that there is space
    # between the words but no space after the last word