CSCI 145 Exam 3 Practice — Lectures 10–14 ANSWER KEY

50 minutes, 50 points. You may use the Exam 3 reference sheet; no other notes, devices, computers, or calculators.
Answer as many questions as you can; if one stalls you, move on and come back to it later if time allows.
When asked for a value, write it as Python would show it (for example, `3.0` vs. `3`, `'hi'` for a string, `[2, 3]` for a list). A set may list its values in any order.

PART 1 — RECURSION

1. Consider this function: *(3 pts)*

```
def wrap(word):
    if len(word) == 0:
        print("|")
    else:
        print(word)
        wrap(word[1:])
        print(word[0])
```

a) Write the exact output of `wrap("cab")`, one printed line per line.

```
cab
ab
b
|
b
a
c
```

b) How many times is `wrap` called in all, counting the first call? **4**

2. Each line below is the recursive case of a different function. Circle **Tail** if the recursive call is the last thing done, or **Pending** if work is left after the call. *(2 pts)*

a) `return 2 + steps(n - 1)` **Tail** **Pending**

b) `return find_max(nums[1:], best)` **Tail** **Pending**

c) `return count_x(word[1:]) + 1` **Tail** **Pending**

d) `return last_word(words[1:])` **Tail** **Pending**

3. Consider this function: *(3 pts)*

```
def glue(words):
    if len(words) == 0:
        return ""
    return words[0] + "-" + glue(words[1:])
```

a) What does `glue(["a", "b", "c"])` return? **'a-b-c-'**

b) How many times is `glue` called in all by `glue(["a", "b", "c"])`, counting the first call? **4**

4. Consider this turtle function and the call `ray(100)`: *(2 pts)*

```
import turtle as t

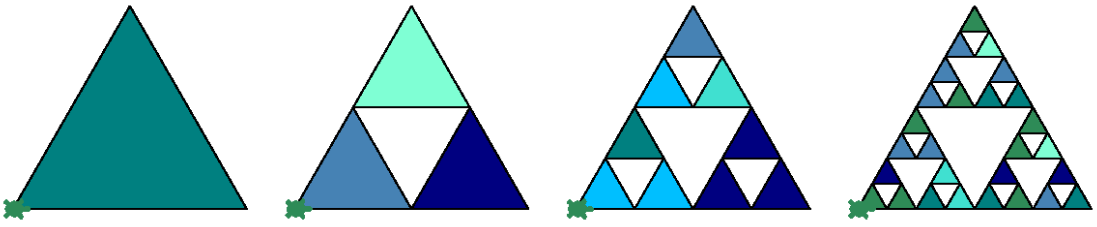
def ray(length):
    if length >= 15:
        t.forward(length)
        t.right(30)
        ray(length - 25)
        t.left(30)
        t.backward(length)

ray(100)
```

a) How many times is `t.forward` called? **4**

b) When `ray(100)` finishes, is the turtle back where it started, facing the same way? Write Yes or No. **Yes**

5. A Sierpinski triangle with 0 levels is one filled triangle. With more levels, it is made of three smaller Sierpinski triangles, each with one fewer level: *(5 pts)*



Write a recursive function `count_triangles(levels)` that returns the number of filled triangles in a Sierpinski triangle with `levels` levels. Assume `levels` is not negative. Do not use a loop or **, and do not include a docstring. For example:

```
>>> count_triangles(0)
1
>>> count_triangles(3)
27
```

```
def count_triangles(levels):
    if levels == 0:
        return 1
    return 3 * count_triangles(levels - 1)
```

PART 2 — TUPLES AND DICTIONARIES

6. Consider this code. What is the value of each expression at the end? *(3 pts)*

```
spot = (2, 9)
x, y = spot
spot = (y, x + 1)
```

a) `x` **2**

b) `spot` **(9, 3)**

c) `spot[0] - x` **7**

7. Write the exact output of this code, one printed line per line. *(4 pts)*

```
paint = {"red": 2, "blue": 5}
paint["green"] = 1
paint["blue"] = paint["blue"] - 1
print(paint)
print(paint.get("pink", 0))
for color, count in paint.items():
    if count > 3:
        print(color)
print(len(paint))
```

```
{'red': 2, 'blue': 4,
 'green': 1}
0
blue
3
```

8. Write a function `first_letters(words)` that returns a dictionary that maps each first letter to the number of words in `words` that start with it. Assume no word is empty. If `words` is empty, it returns an empty dictionary. Do not include a docstring. For example: *(5 pts)*

```
>>> first_letters(["ant", "bee", "ape", "cat", "bat"])
{'a': 2, 'b': 2, 'c': 1}
>>> first_letters([])
{}
```

```
def first_letters(words):
    counts = {}
    for word in words:
        letter = word[0]
        if letter in counts:
            counts[letter] += 1
        else:
            counts[letter] = 1
    return counts
```

PART 3 — SETS, REFERENCES, AND COPIES

9. Suppose `p = {2, 4, 6, 8}` and `q = set([6, 1, 6, 4])`. Write the value of each expression. *(2 pts)*

`p & q`	**{4, 6}**	`p - q`	**{2, 8}**
`len(p | q)`	**5**	`len(set("letter"))`	**4**

10. Consider this code: *(4 pts)*

```
grid = [[0], [1, 2]]
twin = grid.copy()
twin[0].append(9)
twin[1] = [5]
```

a) What is `grid` at the end? **[[0, 9], [1, 2]]**

b) What is `twin` at the end? **[[0, 9], [5]]**

c) `twin[0] is grid[0]` (True or False) **True**

11. Write the exact output of this code, one printed line per line. *(3 pts)*

```
def add_zero(nums):
    nums.append(0)

data = [4, 1]
result = add_zero(data)
backup = data.copy()
add_zero(backup)
print(data)
print(backup)
print(result)
```

```
[4, 1, 0]
[4, 1, 0, 0]
None
```

PART 4 — FILES

12. The file `pets.txt` holds these three lines. Consider the code below. *(4 pts)*

pets.txt

```
cat,3
dog,10
eel,1
```

```
count = 0
with open("pets.txt") as f:
    for line in f:
        parts = line.strip().split(",")
        count += int(parts[1])
        print(parts[0].upper())
print(count)
```

a) Write the exact output, one printed line per line.

```
CAT
DOG
EEL
14
```

b) Suppose the line `count += int(parts[1])` were `count += parts[1]` instead. What would happen when the code runs?

```
An error (TypeError): parts[1] is a
string, and an int + a string fails.
```

13. The files `zoo.csv` and `cfg.json` hold the lines shown. Consider this code: *(2 pts)*

zoo.csv

```
animal,legs
emu,2
ant,6
```

cfg.json

```
{
  "size": 4,
  "dark": true,
  "tags": ["a", "b"]
}
```

```
import csv
import json
with open("zoo.csv") as f:
    rows = list(csv.reader(f))
with open("cfg.json") as f:
    data = json.load(f)
```

Write the value of each expression.

`rows[2][1]`	`'6'`	`len(rows)`	`3`
`data["size"] + 1`	`5`	`data["tags"][-1]`	`'b'`

14. This program writes the file `todo.txt`: *(3 pts)*

```
with open("todo.txt", "w") as f:
    for task in ["mop", "dust"]:
        f.write(task + "\n")
```

Then this second program runs **twice**:

```
with open("todo.txt", "a") as f:
    f.write("nap\n")
```

Write the lines `todo.txt` holds, one per line:

a) after the first program

```
mop
dust
```

b) after the second program runs twice

```
mop
dust
nap
nap
```

c) instead, if the second program used `"w"`

```
nap
```

15. Write a function `max_temp(filename)`. The file has a header line, then one `day,temp` line per day. It returns the highest temperature in the file. If the file has no days (only the header), it returns `None`. Do not include a docstring. For example: *(5 pts)*

`week.csv`

```
day,temp
Mon,41
Tue,-3
Wed,47
```

```
>>> max_temp("week.csv")
47
>>> print(max_temp("start.csv"))
None
```

Here `start.csv` holds only the header line.

```
def max_temp(filename):
    temps = []
    with open(filename) as f:
        lines = f.readlines()
    for line in lines[1:]:
        parts = line.strip().split(",")
        temps.append(int(parts[1]))
    if len(temps) == 0:
        return None
    return max(temps)
```