CSCI 145 Exam 2 Practice — Lectures 5–9 ANSWER KEY

50 minutes, 50 points. You may use the Exam 2 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 the way Python would show it (for example, `3.0` vs. `3`, `'hi'` for a string).

PART 1 — CONDITIONALS

1. Evaluate each expression. Write True or False. *(2 pts)*

`15 % 4 == 3 and 2 > 5` **False**

`10 // 3 == 3 or 5 < 2` **True**

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

```
def shipping(weight):
    cost = 5
    if weight > 10:
        cost = cost + 8
    if weight > 4:
        cost = cost + 3
    return cost
```

a) What does `shipping(6)` return? **8**

b) What does `shipping(12)` return? **16**

c) Suppose the second `if` is changed to `elif`. Now what does `shipping(12)` return? **13**

3. `drink_size(oz)` should return `'large'` for 16 ounces or more, `'medium'` for 12 to 15, and `'small'` for 8 to 11. Otherwise, it should return `'kids'`. *(3 pts)*

```
def drink_size(oz):
    if oz >= 8:
        return 'small'
    elif oz >= 12:
        return 'medium'
    elif oz >= 16:
        return 'large'
    else:
        return 'kids'
```

a) What does `drink_size(20)` return? **'small'**

b) Rewrite the function so it is correct.

```
def drink_size(oz):
    if oz >= 16:
        return 'large'
    elif oz >= 12:
        return 'medium'
    elif oz >= 8:
        return 'small'
    else:
        return 'kids'
```

PART 2 — LOOPS

4. Answer each question about `range`. *(2 pts)*

a) List the values `range(4, 20, 6)` produces, in order. **4, 10, 16**

b) How many times does the body of `for i in range(7, 3):` run? **0**

5. Write the exact output of each snippet, one printed value per line. *(3 pts)*

A

```
c = 4
if c < 9:
    print(c)
    c = c + 3
print("end")
```

```
4
end
```

B

```
c = 4
while c < 9:
    print(c)
    c = c + 3
print("end")
```

```
4
7
end
```

6. For each loop, circle whether it **stops** or **never stops**, and explain why in one line. *(3 pts)*

A

```
count = 0
total = 0
while total < 20:
    count = count + 5
```

stops **never stops**

Why:

The body changes count, but the condition checks total, which stays 0.

B

```
x = 40
while x > 0:
    x = x - 7
```

stops **never stops**

Why:

x goes down by 7 each pass (40, 33, ..., 5, -2), and -2 > 0 is False.

7. A colony starts with `start` cells, and the number of cells doubles every day. Write a function `days_to_reach(start, goal)` that uses a `while` loop to return how many days it takes until there are at least `goal` cells. If `start` is already at least `goal`, it returns `0`. Assume `start` is positive. Do not include a docstring. For example: *(3 pts)*

```
>>> days_to_reach(5, 40)
3
>>> days_to_reach(3, 20)
3
>>> days_to_reach(8, 5)
0
```

```
def days_to_reach(start, goal):
    days = 0
    while start < goal:
        start = start * 2
        days += 1
    return days
```

PART 3 — LOOP PATTERNS

8. `count_vowels(word)` should return how many characters of `word` are vowels. *(3 pts)*

```
def count_vowels(word):
    for ch in word:
        count = 0
        if ch in "aeiou":
            count += 1
    return count
```

a) What does `count_vowels("piano")` return? **1**

b) Explain what is wrong, and how to fix it.

```
count = 0 is inside the loop, so it resets
on every character; only the last character
counts. Fix: move count = 0 before the loop.
```

9. When this code runs, the user types `-5`, then `200`, then `30`. *(3 pts)*

```
age = int(input("Age? "))
while age < 0 or age > 120:
    print("Try again")
    age = int(input("Age? "))
print("Thanks,", age)
```

a) Write the lines the `print` calls display, in order. (Leave out the `Age?` prompts.)

```
Try again
Try again
Thanks, 30
```

b) Which values of `age` make the loop stop? **0 through 120**

10. Write the exact output, one printed line per line. *(3 pts)*

```
for i in range(2, 4):
    for j in range(i):
        print(i, j)
```

```
2 0
2 1
3 0
3 1
3 2
```

11. Write a function `first_index(s, ch)` without using `.find`. If `ch` appears in the string `s`, it returns the index of its first appearance. Otherwise, it returns `-1`. Do not include a docstring. For example: *(4 pts)*

```
>>> first_index("banana", "n")
2
>>> first_index("cat", "z")
-1
```

```
def first_index(s, ch):
    for i in range(len(s)):
        if s[i] == ch:
            return i
    return -1
```

PART 4 — LISTS

12. Suppose `temps = [62, 70, 58, 75, 66]`. Evaluate each expression. If it causes an error, write Error. *(3 pts)*

`max(temps) - temps[-1]` **9**

`temps[1:4]` **[70, 58, 75]**

`temps[5]` **Error (IndexError)**

13. Consider this code: *(2 pts)*

```
stack = [3, 6]
stack.append(9)
top = stack.pop(0)
stack.insert(0, top * 10)
```

a) What is `stack` at the end? **[30, 6, 9]**

b) What is `top` at the end? **3**

14. Write a function `positions(nums, target)`. If `target` appears in `nums`, it returns a list of every index where it appears, smallest first. Otherwise, it returns `[]`. Do not use a list comprehension, and do not include a docstring. For example: *(4 pts)*

```
>>> positions([4, 7, 4, 1], 4)
[0, 2]
>>> positions([4, 7], 9)
[]
```

```
def positions(nums, target):
    result = []
    for i in range(len(nums)):
        if nums[i] == target:
            result.append(i)
    return result
```

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

```
def add_item(items, x):
    items.append(x)

a = [1, 2]
b = a
c = a[:]
add_item(a, 3)
```

a) a **[1, 2, 3]**

b) b **[1, 2, 3]**

c) c **[1, 2]**

16. Consider this code: *(3 pts)*

```
nums = [8, 3, 6, 1]
print(sorted(nums))
print(nums)
result = nums.sort()
print(nums)
print(result)
```

a) Write the exact output.

```
[1, 3, 6, 8]
[8, 3, 6, 1]
[1, 3, 6, 8]
None
```

b) Write one expression whose value is a list of the two smallest numbers in `scores`, smallest first, without changing `scores`.

`sorted(scores)[:2]`

17. List comprehensions. *(3 pts)*

a) What is the value of `[x - 1 for x in [4, 9, 2] if x > 3]`? **`[3, 8]`**

b) Rewrite this loop as a one-line list comprehension that builds the same list.

```
halves = []
for num in nums:
    if num % 2 == 0:
        halves.append(num // 2)
```

`halves = [num // 2 for num in nums if num % 2 == 0]`