Name: _____

This test has 15 questions across 12 pages worth a total of 111 points, and is to be completed in 120 minutes. Point totals are labeled next to each question. The exam is closed book, and no calculators or other electronic devices are permitted.

Write the statement out below in the blank provided and sign your name. You may do this before the exam begins.

"I have neither given nor received any assistance in the taking of this exam."

_____

_____

Signature: _____

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- Use your time wisely. If you are having too much trouble on a question, skip it and return to it later. **Work through the ones you are comfortable with first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.**
- If you are given skeleton code to fill in, we will not grade your answer if you alter the skeleton code or write outside the solution box.
- For questions with *square checkboxes*, you may select *more than one* choice.
- For questions with *circles*, you may *only select one* choice.
- Relax, and try to have fun with it! Think of this exam as a checkpoint along your JamCoders journey. You got this!

Staff use only.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Q9	Q10	Q11	Q12	Q13	Q14	Q15	Total

1. ***(5 points)*** For each of the following lines, fill in the blank (_____) with **one of the following types**: `int`, `list`, `str`, `bool`, `float`.

```
print(type("Thi5 i5 c00l"))   # Printed: _________________________
print(type(67.6))             # Printed: _________________________
print(type([4.5, 2, 4, 8]))   # Printed: _________________________
print(type(10 > 10))          # Printed: _________________________
print(type(5 // 2))           # Printed: _________________________
```

2. ***(4 points)*** For which value(s) of `a` and `b` will the following code print `Devon_House`?

```
if a and b:
    print("Kingston")
elif a:
    print("Hope_Gardens")
else:
    print("Devon_House")
```

Fill in the boxes next to all answers that print `Devon_House`.

- ☐ `a = True, b = True`
- ☐ `a = True, b = False`
- ☐ `a = False, b = True`
- ☐ `a = False, b = False`

3. ***(6 points)*** What does the code below print?

```
L = [[0, 1, 2],[3, 0, 4],[0, 5],[6, 7, 0]]
print(L[1][0])
```

Answer here: _____

```
better_than_kgn = "Montego_Bay" + "Ocho_Rios"
print(better_than_kgn[:8] + better_than_kgn[-4:])
```

Answer here: _____

4. ***(5 points)*** Fill in the missing code below to print the elements of `cool`.

```
lst = [0, 1, 2, 3, 4]
cool = ["Ice cream", "Lemonade", "Popsicles"]

for ______________ in lst:

  if _________________________:

    print(____________________)

  else:

    ________________________
```

5. ***(5 points)*** Consider the following code:

```
nums = [1, 2, 8]
i = 0
while i < len(nums):
    print(nums[i])
    if nums[i] % 2 == 0:
        i += 2
    else:
        i -= 1
```

When it is run, what is printed?

- ○ `1 2 8`
- ○ `1 8 2`
- ○ An error occurs.
- ○ Infinite loop.
- ○ None of the above.

6. ***(5 points)*** The following function searches a non-empty list of ***integers*** (positive or negative) and returns the ***maximum value***. Fill in the blanks to complete the implementation.

```
def max_in_list(lst):

    max_value = ___________________

    for i in ________________________:

        if ___________ > max_value:

           ______________________

    return max_value
```

7. ***(5 points)*** Consider the following lines of code:

```
def naila(fruit):
    if fruit == "Pineapple":
        print("Allergic")
    else:
        print("Yayy!!")

def kanav(fruit):
    if fruit != "Pineapple":
        print("Nah")
    return fruit

kanav("Berries")
print(naila("Pineapple"))
naila(kanav("Apple"))
```

What is the printed output of the code above?

- ○ `"Berries"` ↵ `None` ↵ `"Yayy!!"`
- ○ `"Nah"` ↵ `"Allergic"` ↵ `None` ↵ `"Nah"` ↵ `"Yayy!!"`
- ○ `"Nah"` ↵ `"naila("Pineapple")"` ↵ `"Nah"` ↵ `"Yayy!!"`
- ○ `"Berries"` ↵ `"Allergic"` ↵ `"Nah"` ↵ `"Apple"` ↵ `"Yayy!!"`
- ○ None of the above.

8. ***(2 points)*** What type of error (if any) is printed with the following code blocks? (Don't worry about getting the name spot on!)

```
i = 0
lst = ['x', 'y', 'z']
while i <= len(lst):
    print(lst[i])
    i += 1
```

Answer here: _____

9. ***(14 points)*** What is the output printed by each of the following code snippets? If a snippet results in an infinite loop, write `"Infinite"`. If it produces an error, write `"Error"` and briefly explain the reason. Write your answer in the empty box after each snippet.

a.

```
x = [6, 2, 1, 2]
y = [1, 2, 2, 4]
z = 0

for i in range(len(x)):
    z = z + (x[i] * y[i])

print(z)
```

b.

```
num1 = 5
num2 = 12

def max_value(num1, num2):
    if num1 > num2:
        return num1
    return num2

print(max_value(45, 32))
```

c.

```
def foo(num):
  if num == 1:
    return 1
  return num * foo(num - 1)

print(foo(5))
```

d.

```
def bar(lst):
  if len(lst) == 1:
    print(lst[0])
  bar(lst[1:])

print(bar(["Hello", "world!"]))
```

10. Favorite Fruits ***(14 points)***

Daniel, Anushka, and Andrey all love fruits—but each of them has their own favorite! Ms. B uses the function `pour_juice` to serve them their favorite fruit.

As an example, the function call:

```
pour_juice([["Daniel", "Grapes"], ["Anuska", "Banana"],
            ["Andrey", "Watermelon"]], "Daniel")
```

Will print:

```
Daniel LOVES Grapes
```

a. ***(5 points)*** Fill in the blanks to complete the function so that it behaves correctly.

```
def pour_juice(name_fruit_lst, name):

    for name_fruit_pair in name_fruit_lst:
        if name == ____________________:

            print(__________________ + " LOVES " + __________________)
            __________________

    print("Who name suh?")
```

b. ***(3 points)*** Now, consider the following statement:

```
pour_juice([["Daniel", "Grapes"], ["Anushka", "Banana"],
            ["Andrey", "Watermelon"]], "Frank")
```

What will be printed? Answer here: _____

c. ***(6 points)*** Now, solve the `pour_juice` function ***recursively***. Fill in the blanks to complete the function.

```
def pour_juice(name_fruit_lst, name):

    if ____________________________:
        print("Who name suh?")
        _________________

    __________________ = __________________

    if name == ____________________:

        print(__________________ + " LOVES " + __________________)
        _________________

    pour_juice(_________________________, ____________________)
```

11. Draw the Triangle *(10 points)*

Write a function `draw_triangle(n)` that prints a right triangle using the character `"x"`, where `n >= 1`.

For example, the function call `draw_triangle(5)` should print:

```
x
xx
xxx
xxxx
xxxxx
```

The function call `draw_triangle(8)` should print:

```
x
xx
xxx
xxxx
xxxxx
xxxxxx
xxxxxxx
xxxxxxxx
```

a. *(4 points)* First, we'll solve the function using *iteration*.

```
def draw_triangle(n):

    for i in _________________________:

        print(____________________)
```

b. *(6 points)* Next, we'll solve the function using *recursion*.

```
def draw_triangle(n):

    if ____________________________:
        return

    ______________________________

    ______________________________
```

12. Pairs From List ***(16 points)***

The function `pairs_from_list(lst)` takes a list of one-digit integers as input and returns a new list by combining each pair of adjacent digits into a two-digit integer. If the list contains an odd number of elements, the final digit is left unchanged.

Examples:

- `pairs_from_list([1, 2, 3, 4, 5])` returns `[12, 34, 5]`
- `pairs_from_list([8, 6, 4, 2])` returns `[86, 42]`
- `pairs_from_list([7])` returns `[7]`
- `pairs_from_list([])` returns `[]`

a. ***(8 points)*** First, we'll solve the function using *iteration*.

```
def pair_from_list(lst):

    result = []

    for i in ____________________________:

        if ______________________________:

            result.append(_________________)

        else:

            result.append(___________________ + ___________________)

    return ________________________
```

b. ***(8 points)*** Next, we'll solve the function using *recursion*.

```
def pair_from_list(lst):

    if ______________________________:

        return ________________________

    if ______________________________:

        return ________________________

    return ______________________________ + pair_from_list(_____________)
```

13. Increasing Digits ***(20 points)***

Write a function `increasing_digits(n)` that takes a positive integer `n` and returns `True` if every pair of adjacent digits is strictly increasing from left to right, and `False` otherwise.

Examples:

- `increasing_digits(123456)` returns `True`
- `increasing_digits(13579)` returns `True`
- `increasing_digits(122345)` returns `False`
- `increasing_digits(521)` returns `False`
- `increasing_digits(7)` returns `True`

a. ***(10 points)*** First, we'll solve the function using *iteration*.

```
def increasing_digits(n):

    prev = 10

    while __________ > __________:

        digit = ____________ % ____________

        if _______________ >= _______________:

            return _________________

        prev = ____________________

        n = ________________________

    return ________________________
```

b. ***(10 points)*** Next, we'll solve the function using *recursion*.

```
def increasing_digits(n):

    if ____________________________:
        return ____________________

    last = ______________________________

    second_last = ________________________

    if ____________________________:
        return ____________________

    return ____________________________
```

14. ***(10 points) Optional*** **Challenge Problem:** *Finish other problems before attempting.*

Let's write a `challenge(lst)` function that takes as input a list of non-empty lists of integers. The function should return a new list where each element is `True` if all of the integers in the corresponding inner list are the same, and `False` otherwise.

Below is an example function call.

```
challenge([[1, 1, 1], [4, 5], [6, 6, 8, 6], [5]])
```

The function *returns*:

```
[True, False, False, True]
```

You may assume every inner list contains at least one integer.

Write your code below. You may not need to use all of the lines.

15. ***(10 points) Optional*** **Challenge Problem:** *Finish other problems before attempting.*

Let's write a `valid_bracket(lst)` function that takes as input a string `lst` consisting only of the characters `"("` and `")"`.

The function should return `True` if the brackets are balanced. In other words, every opening bracket must have a matching closing bracket, every closing bracket must have a corresponding opening bracket, and the brackets must be properly nested. Otherwise, the function should return `False`.

Below are two example function calls.

```
valid_bracket("(((()")
valid_bracket("(())")
```

The functions *return*:

```
False
True
```

Write your code below. You may not need to use all of the lines.