UNIVERSITY OF CALIFORNIA, BERKELEY
Department of Electrical Engineering and Computer Sciences
Computer Science Division

CS10 Fall 2025 **TA: Victoria**

SOLUTIONS

Here are the solutions in code: solutions file

Discussion 13: Postterm Practice

Instructions:

- If you're attending this section in-person, please log into iClicker!
- If you missed this discussion, fill out this entire worksheet, and upload it to the Gradescope assignment titled "Discussion 13" by next Discussion.
- For the worksheet, you can either explain the process in words, show a screenshot, or draw the block/process.
- Please complete the Feedback Form tinyurl.com/fa25-disc-form

Group Activity / Question of the Day

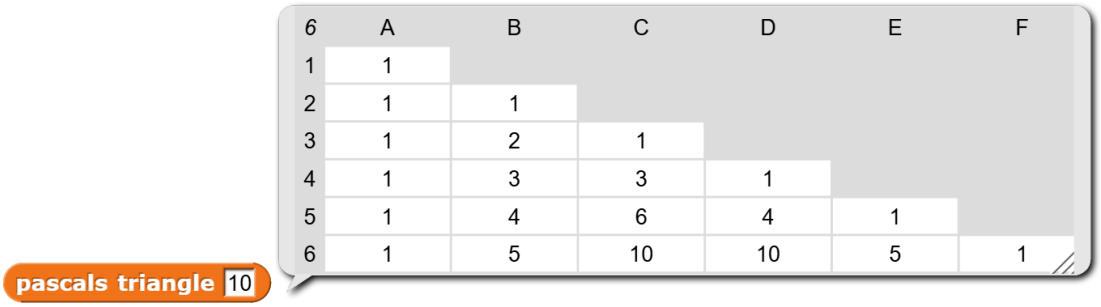
- If you had to delete all but three apps from your smartphone, which ones would you keep?

Required (Pages 2 - 5):

Section I - HOFs

---

1. You will write blocks that build Pascal's Triangle, but instead of drawing it, you will store it as a list of lists. Given a value (for example: 10), build all rows of Pascal's Triangle up to and including the row until the row contains that value. For example, if the value is 10, your output should include all rows up through the row that has the number 10 in it. The rows look like this:

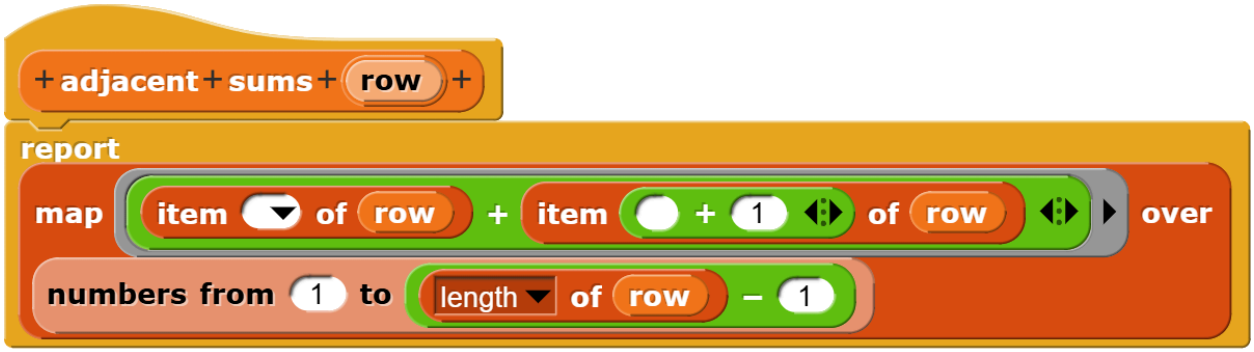


6	A	B	C	D	E	F
1	1					
2	1	1				
3	1	2	1			
4	1	3	3	1		
5	1	4	6	4	1	
6	1	5	10	10	5	1

You will be building this function in parts.

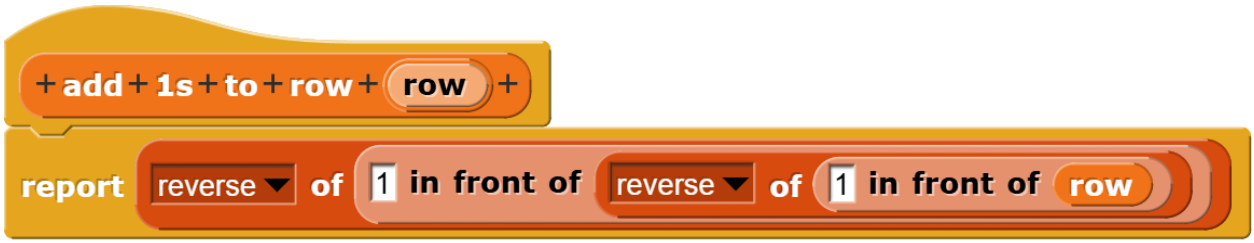
Part 1: Adjacent sums 

. Every number in the middle is the sum of two neighbors from the previous row. Given a row [1, 3, 3, 1], return the list of adjacent sums: [4, 6, 4] → *1+3=4, 3+3=6, 3+1=4.* **<u>Use only HOFs</u>**

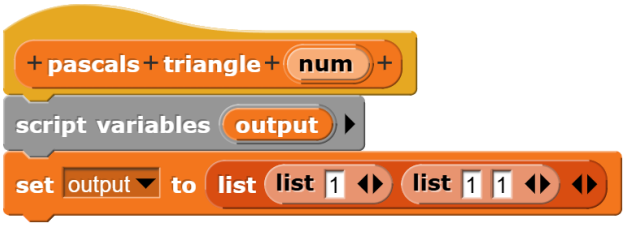


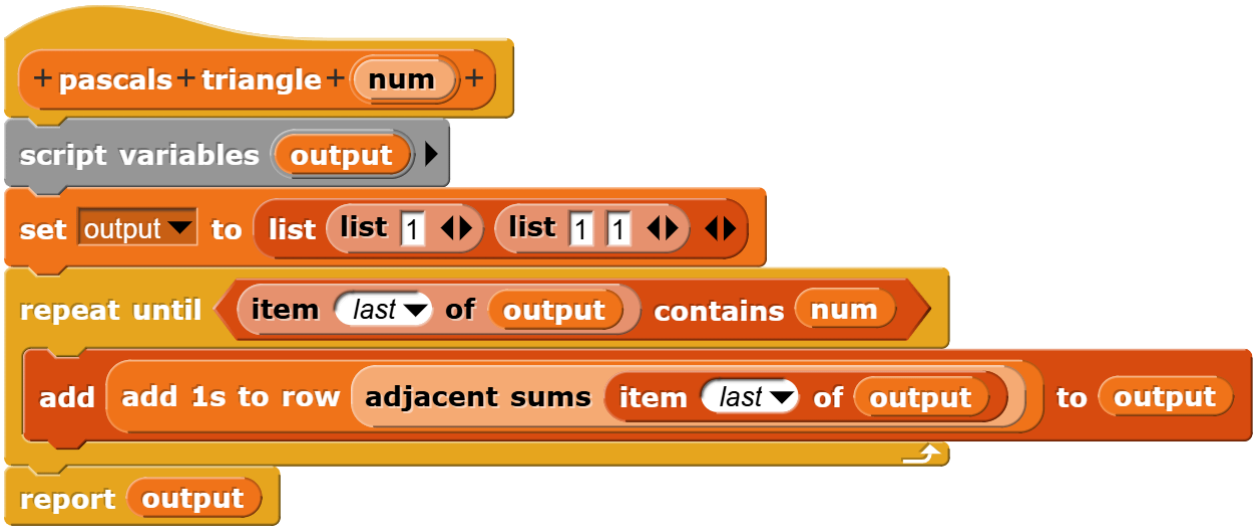
Part 2: Add ones to row 

. Given a row [4, 6, 4], return the corresponding value [1, 4, 6, 4, 1]. **<u>Cannot use iteration.</u>**

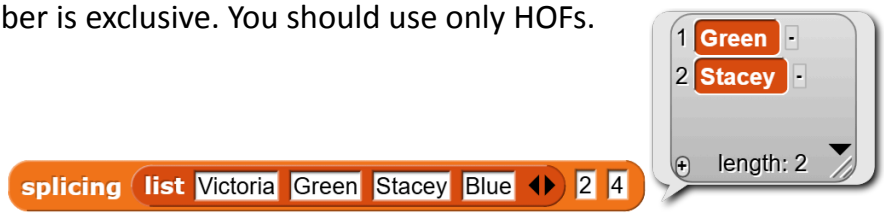


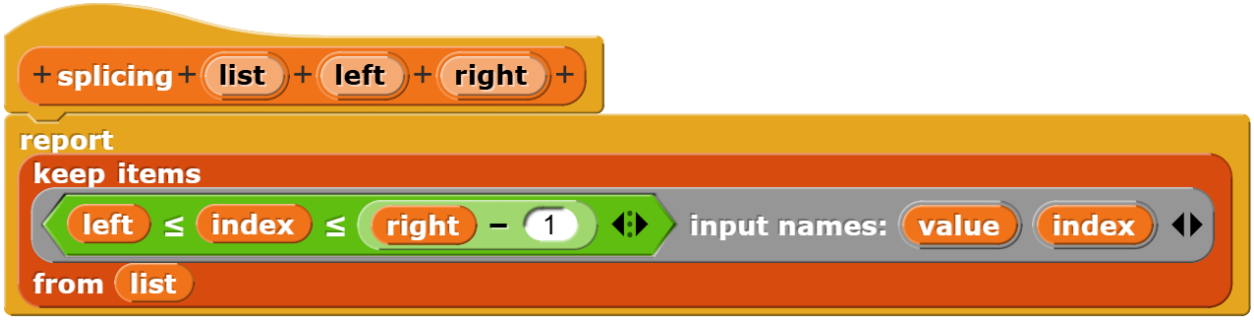
Part 3: Build Pascal's triangle, stopping when a row contains 'num'. Use anything! Here is some starter code to get you started:





2. Write a function `splicing(data, left, right)` that works like splicing in Python, taking in a list, and two numbers (left and right), which represent the left and right indices when splicing. Items should be indexed starting at 1 though, and the right number is exclusive. You should use only HOFs.





Both solutions are valid

Section II - Concurrency

Assume you have the following script. What would string be once all scripts are done running for the given x, y, z values? If all scripts are running at the same time, you can assume the left topmost goes first. If x = 1, y = 1, and z = 1, the string would be "112341234234". Assume each repeat takes 1 second.

a. x = 1, y = 2, z = 2

b. x = 0, y = 4, z = 2

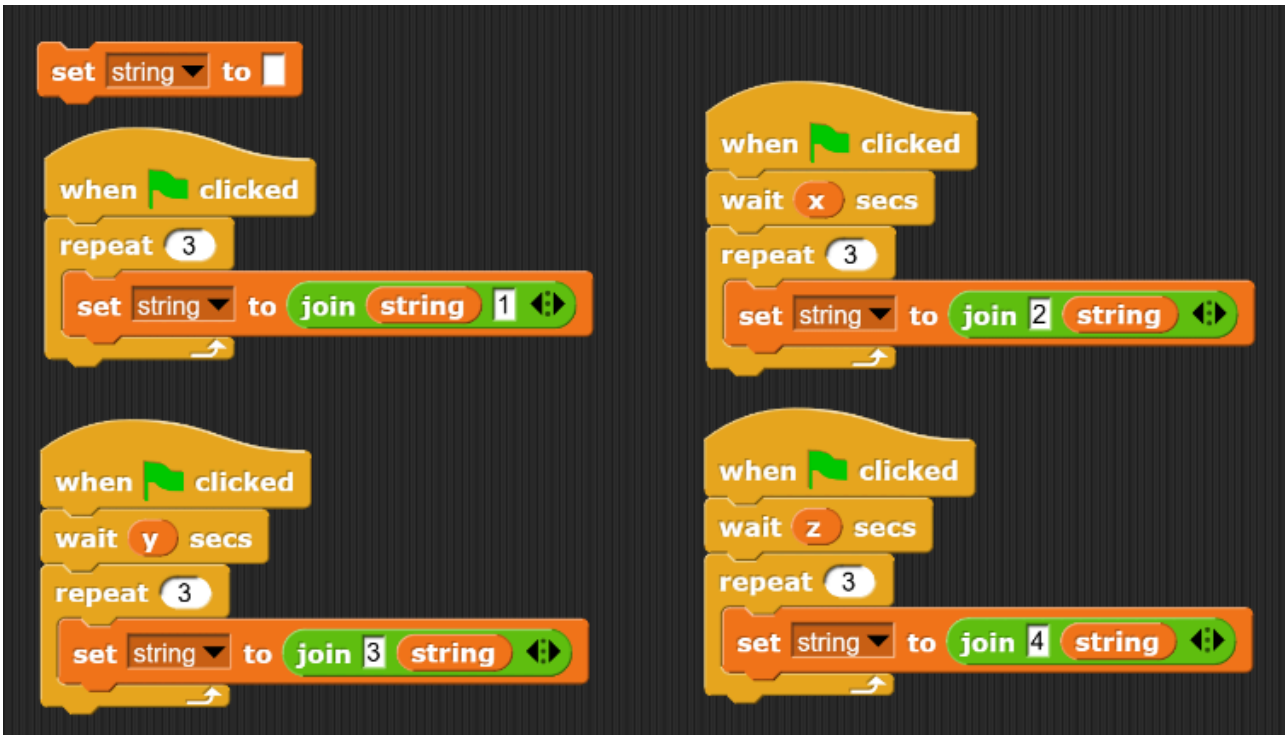
Answer: 112123423434

Answer: 121212443433

See slides for walkthrough. I messed this example up a bit. The answers above would only result if all the strings say:
Set string to join(string, 1)
Set string to join(string, 2)
Set string to join(string, 3)
Set string to join(string, 4)

However, the actual script says the following:
Set string to join(string, 1)
Set string to join(2, string)
Set string to join(3, string)
Set string to join(4, string)

So for 2,3,4 the numbers are placed in front of the string. This results in the following answers:
x = 1, y = 2, z = 2 → String = 432344322111
x = 0, y = 4, z = 2 → String = 334344222111



Section III - More Postterm Practice

---

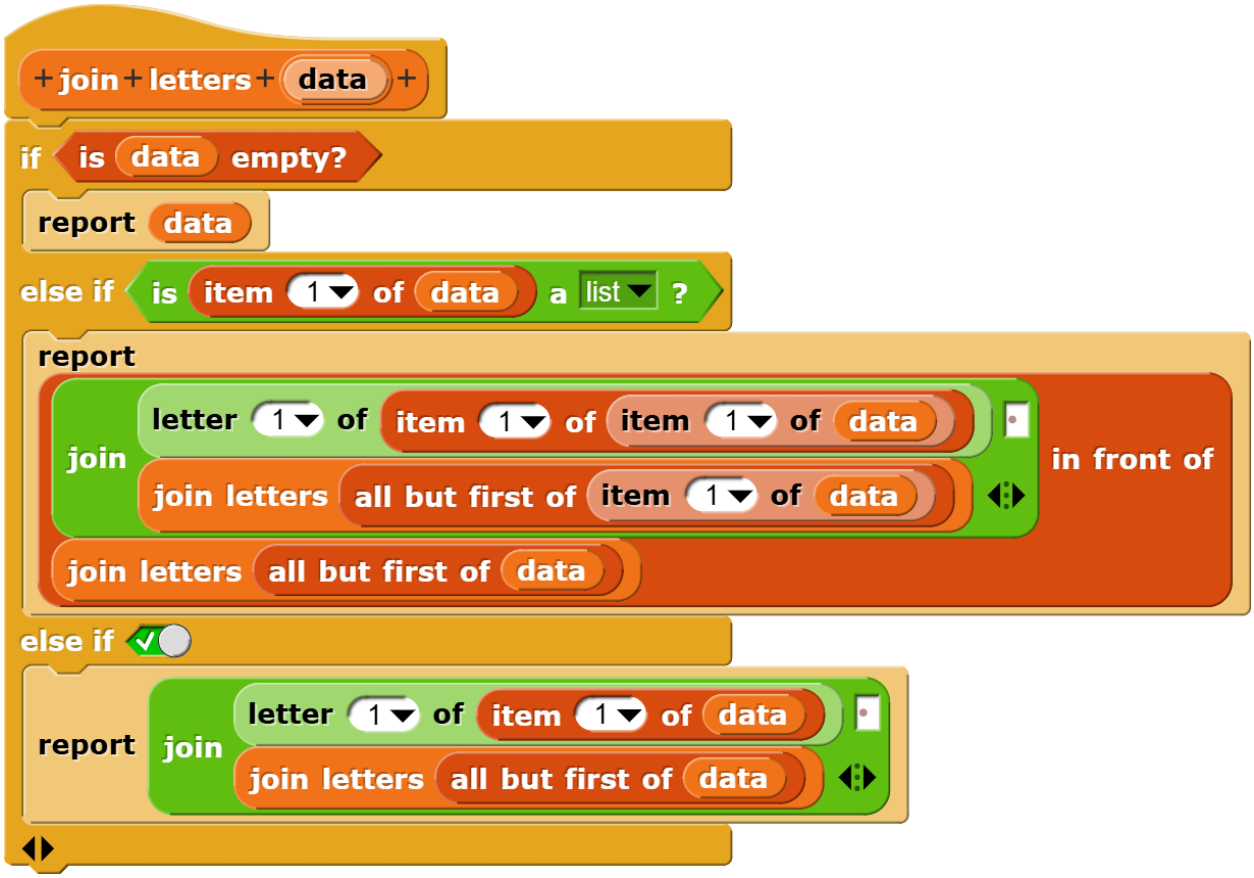
2. What's $\frac{AF}{101_2}$? Take the floor. Express your result in hexadecimal and in base-3

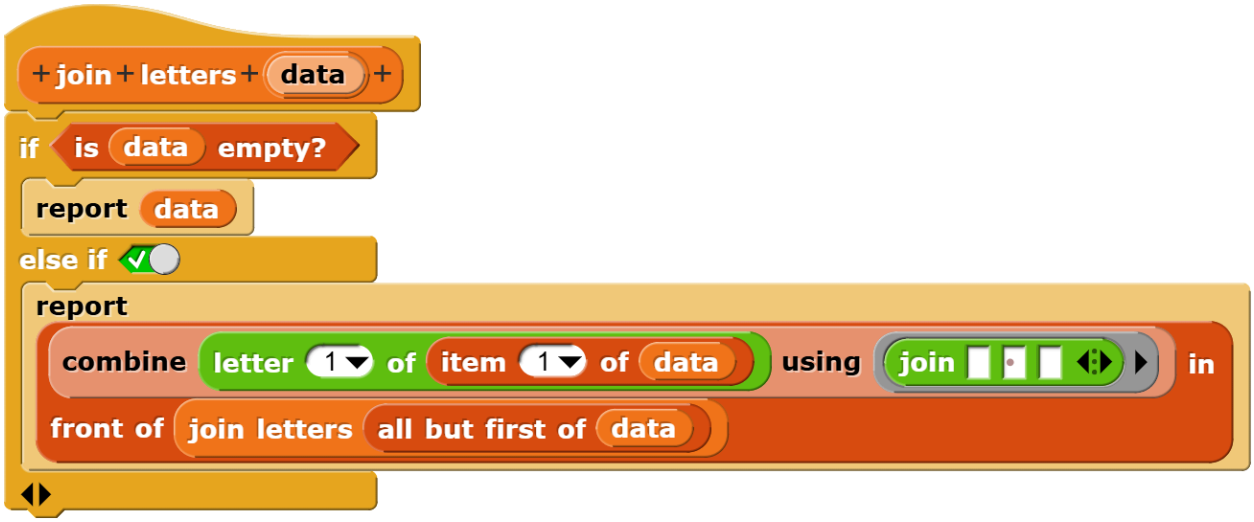
1000 (in base 3). See slides for walkthrough

3. What's $\frac{BEEF_{16}}{12467_8}$? Take the floor. Express your result in decimal and in base-7

5 (in base 7). See slides for walkthrough

4. Write a function `join_letters(data)` take takes in a 2D list of strings and returns a single 1D list with the first letters of each item separated by a space. The function should be completed recursively.





Both solutions are valid!

5. Write a function `sort_by_unicode_weight(s)` that takes a string `s` of words separated by spaces. Each word's Unicode weight is defined as the sum of the Unicode code points (ord(char)) of its lowercase alphabetic letters only. Ignore any non-alphabetic characters when calculating the score. Your function should:
    a. Compute the Unicode weight of each word.
    b. Sort the words in descending order of their Unicode weight.
    c. Return the sorted words joined by spaces.

    Here are some examples:

```
>>> sort_by_unicode_weight("Cat apple zOOm!")
'apple zOOm! Cat'
>>> sort_by_unicode_weight("hello world!")
'world! hello'
>>> sort_by_unicode_weight("Aa Bb Cc")
'Cc Bb Aa'
```

    Hint: Python has a built-in function called ord(char) that gives you the Unicode (ASCII) value of a character. For example: `ord('a')  # returns 97`

```
def sort_by_unicode_weight(s):
    def unicode_weight(word):
        total = 0
        for ch in word:
            if ch.isalpha():
                total += ord(ch.lower())
        return total

    words = s.split()
    words.sort(key=unicode_weight,
reverse=True)
```

```
    return " ".join(words)
```

6. Return a list containing every possible substring of string having length n. You can assume len(output) < len(string). Complete the function:
    a. In Python using List comprehension
    b. in Snap*!* Using HOFs

```
>>> def substring("Hello", 3)
['Hel', 'ell', 'llo']
>>> def substring("World", 2)
['Wo', 'or', 'rd', 'ld']
```

```
def substring(s, n):
    return [s[i:i+n] for i in range(len(s) - n + 1)]
```

7. Write a function named is_descending that takes an integer n as input and determines if n's digits are descending. The function should return True if each digit is in descending order, otherwise False.
    a. Write the function
        i. in Python using list comprehension
        ii. In Snap! Using HOFs
    b. Below is an example:

```
>>print(is_descending(1))
True
>>>print(is_descending(1111))
False
>>>print(is_descending(12))
False
>>>print(is_descending(321))
True
>>>print(is_descending(31))
True
```

```
def is_descending(n):
    s = str(n)
    return all([int(s[i]) > int(s[i+1]) for i
in range(len(s) - 1)])
```