Discussion 4

www.yellkey.com/instead

Feedback!

- could we work through more examples during disc?
  - We can try!
- More participation from the audience

Python Lists

a data structure that can store multiple elements

Constructing Lists:

```
>>> list_of_ints = [1, 2, 3, 4]
>>> list_of_bools = [True, True, False, False]
>>> nested_lists = [1, [2, 3], [4, [5]]]
```

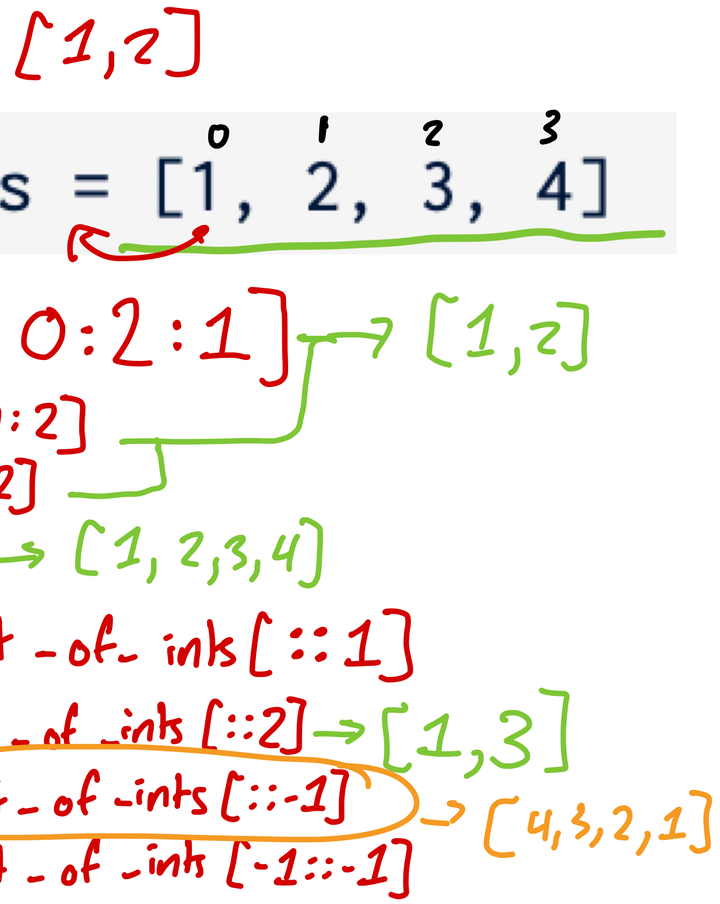
Getting elements:

```
>>> lst = [6, 5, 4, 3, 2, 1, 0]
>>> lst[0]
6
>>> lst[3]
3
>>> lst[-1] # Same as lst[6]
0
```

List Slicing

copy part or all of a list

```
>>> list_of_ints = [1, 2, 3, 4]
```



`lst[start:end:step]`

- Does not include element at index `end`
- `step`: step size for selecting elements
  - `step=2`: every other element

List Slicing

```
>>> directors = ['jenkins', 'spielberg', 'bigelow', 'kubrick']

>>> directors[:2]

['jenkins', 'spielberg']

>>> directors[1:3]

['spielberg', 'bigelow']

>>> directors[1:]

['spielberg', 'bigelow', 'kubrick']

>>> directors[0:4:2]

['jenkins', 'bigelow']

>>> directors[::-1]

['kubrick', ‘bigelow, ‘spielberg, 'jenkins']
```

List Comprehensions

For loops: Iterate over every element in list

for i in list_of_ints:
    print(i)

1
2
3
4

List Comprehensions

list_of_ints

```
>>> [x * x - 3 for x in [1, 2, 3, 4, 5]]
[-2, 1, 6, 13, 22]
>>> [x * x - 3 for x in [1, 2, 3, 4, 5] if x % 2 == 1]
[-2, 6, 22]
```

1st

2·2-3 = 4-3=1

1st

(x+1 if x%2==0 else x-1)

Q3: WWPD: Lists

What would Python display?

```
>>> a = [1, 5, 4, [2, 3], 3]
>>> print(a[0], a[-1])
```

```
>>> len(a)
```
5

```
>>> 2 in a
```
False

```
>>> a[3][0]
```
2

range(len(a))
[0,1,2,3,4]

2 in a[3] → 2 in [2,3]

a[3]
a[0]

(a[3])[0]
([2,3])[0]
2

[[x+1 for e2 in e] for e in l]

[[2,3],
[4,5]]

[1,2]
[3,4]

l = [[1,2]
[3,4]]

a[0][0]
a[1][1]

Q4: Even weighted

Write a function that takes a list s and returns a new list that keeps only the even-indexed elements of s and multiplies them by their corresponding index.

```
def even_weighted(s):
    """
    >>> x = [1, 2, 3, 4, 5, 6]
    >>> even_weighted(x)
    [0, 6, 20]
    """
    return [s[i] * i for i in range(len(s)) if i % 2 == 0]
```

range(0, len(s))

range(len(s))

range(start, stop)

0 1 2 3 4 5

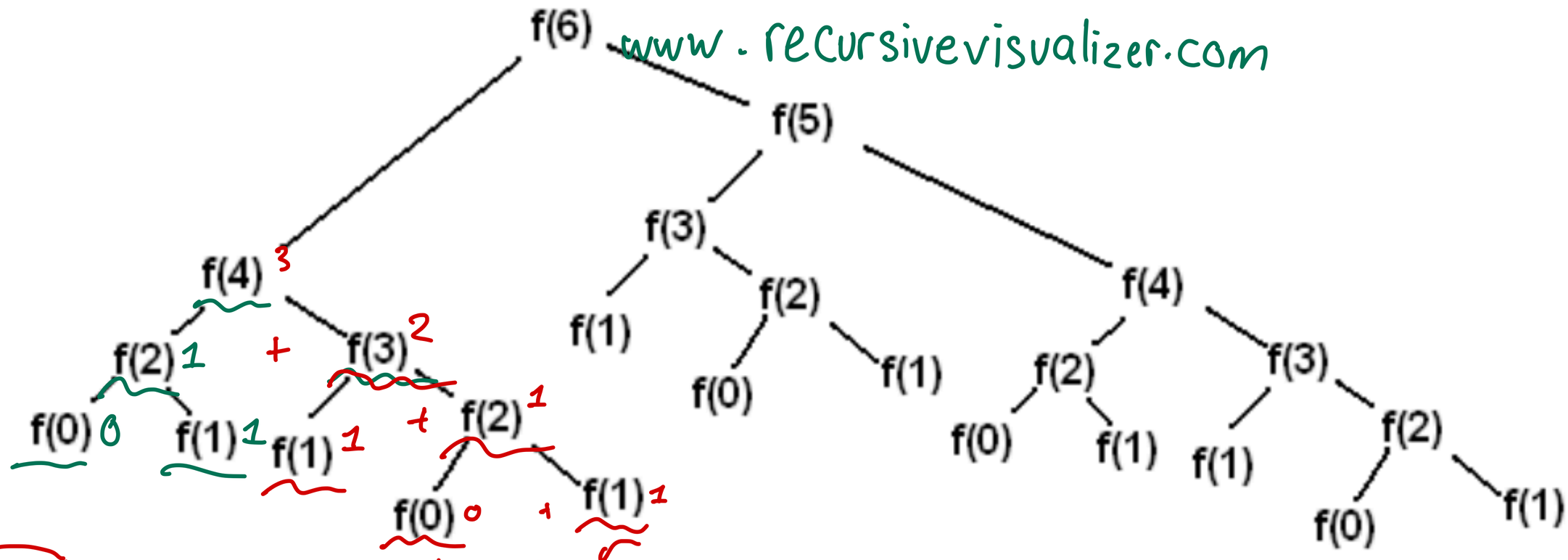
[1, 3, 5]

×0 ×2 ×4

range(1,3) → [1,2]

[0, 6, 20]

Tree Recursion



0th, 1st, 2nd, 3rd, 4th
0, 1, 1, 2, 3

n-1

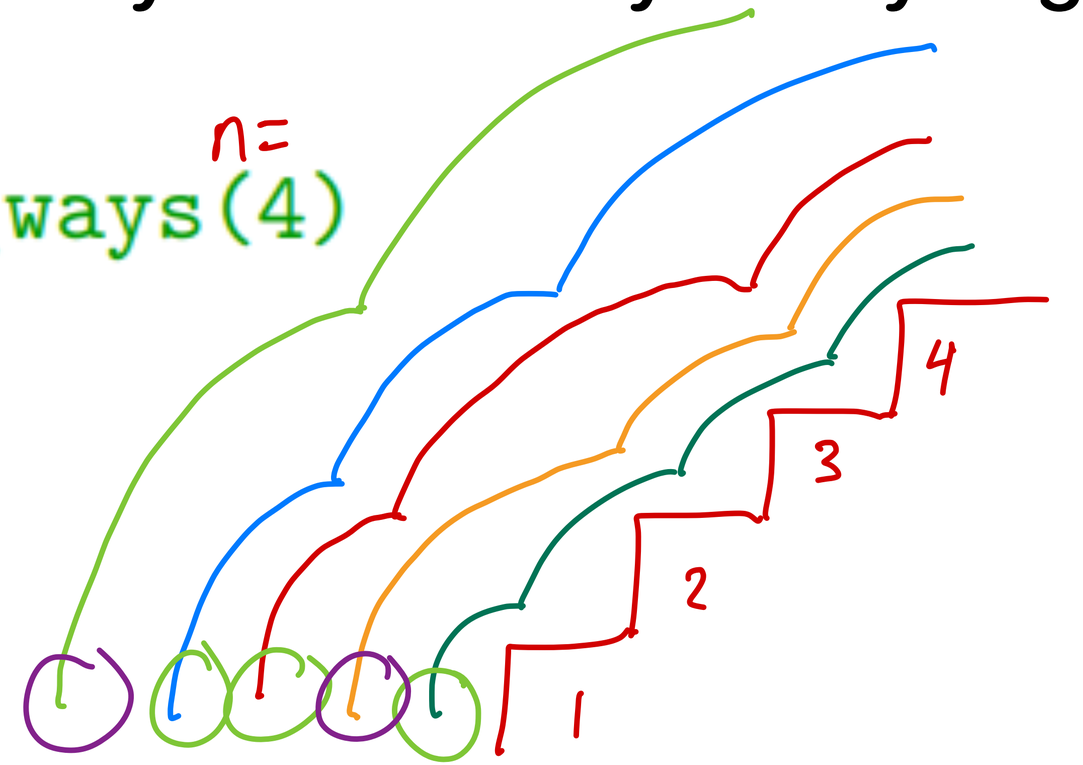
n-2

```
def fib(n):
    if n==0 or n==1:
        return n
    return fib(n-1)+ fib(n-2
```

Q1: Count Stair Ways

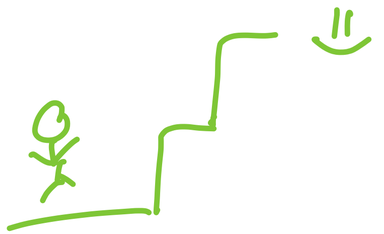
Imagine that you want to go up a flight of stairs that has n steps, where n is a positive integer. You can either take 1 or 2 steps each time. How many different ways can you go up this flight of stairs?

```
>>> count_stair_ways(4)
5
```



Q1: Count Stair Ways

Before you code your approach, consider these questions.



How many different ways are there to go up a flight of stairs with n = 1 step? How about n = 2 steps? Try writing out some other examples and see if you notice any patterns.

1 way

2

What's the base case for this question? What is the simplest input?

n==1 , n==2

return n

What do count_stair_ways(n - 1) and count_stair_ways(n - 2) represent?

ways to get there starting w/ 1 step

Q1: Count Stair Ways

Fill in the code for count_stair_ways:

```
def count_stair_ways(n):
    """Returns the number of ways to climb up a flight of
    n stairs, moving either 1 step or 2 steps at a time.
```

Q5: Max Product

Write a function that takes in a list and returns the maximum product that can be formed using nonconsecutive elements of the list. The input list will contain only numbers greater than or equal to 1.

```
>>> max_product([10,3,1,9,2]) # 10 * 9
90
>>> max_product([5,10,5,10,5]) # 5 * 5 * 5
125
>>> max_product([])
1
```

10* → 1) include 10 → can't inc. 3 → 10×9=90
2) dont include 10 → can inc. 3

```
if len(s)==0:
    return 1
if len(s)==1:
    return s[0]
```

Q5: Max Product

```
def max_product(s):
    """Return the maximum product that can be formed using
    non-consecutive elements of s.
```

Q2: Count K

Consider a special version of the count_stair_ways problem, where instead of taking 1 or 2 steps, we are able to take up to and including k steps at a time. Write a function count_k that figures out the number of paths for this scenario. Assume n and k are positive.

```
>>> count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
4
```

Q2: Count K

```
def count_k(n, k):
    """ Counts the number of paths up a flight of n stairs
    when taking up to and including k steps at a time.
```

Feedback + Attendance

www.yellkey.com/safe

Pwd: heftyhogs