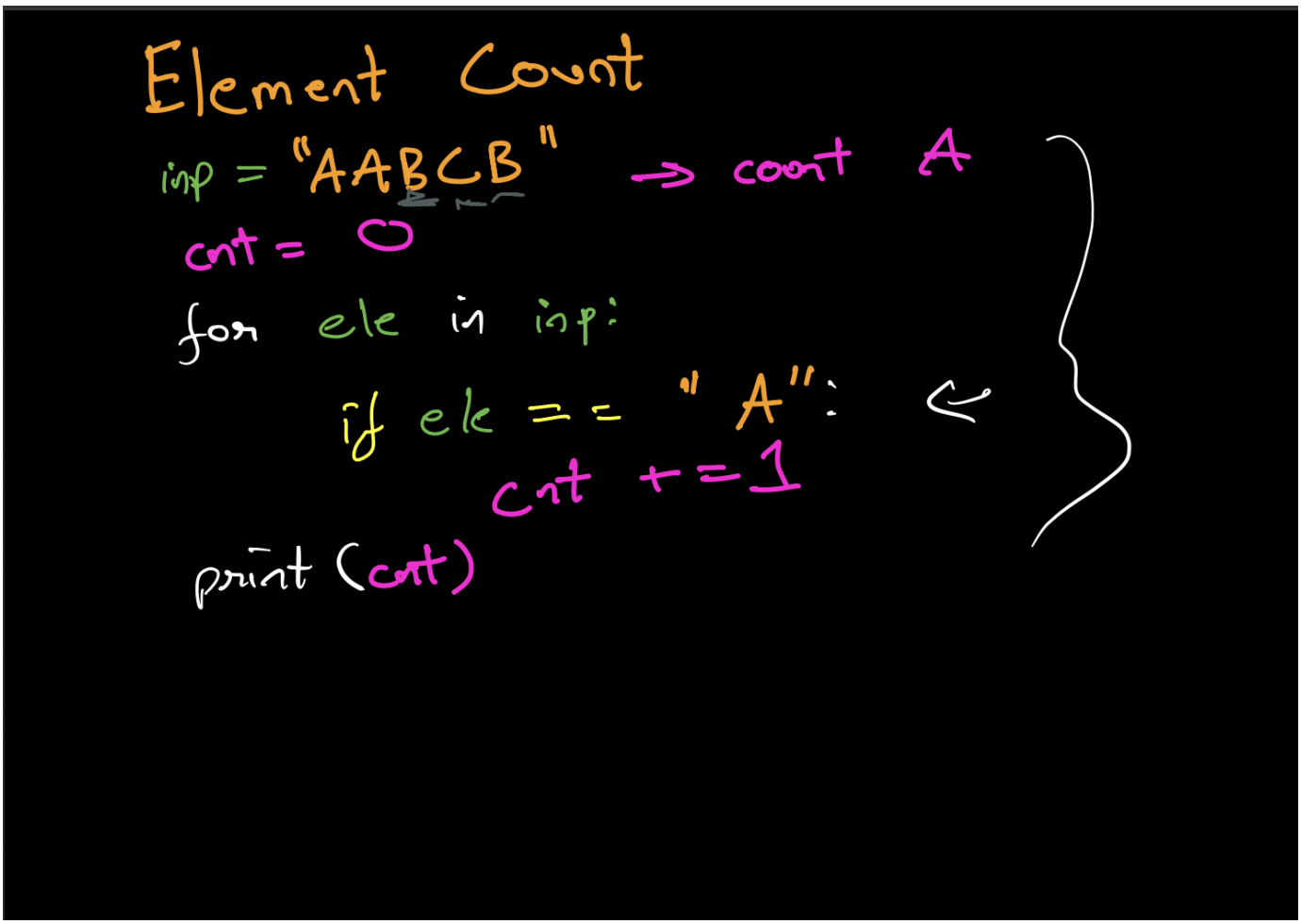
Link to read only copy of this notebook: https://colab.research.google.com/drive/1Shgfnl_lui-ve62Nk1MxGTB1SYKeUcKv?usp=sharing

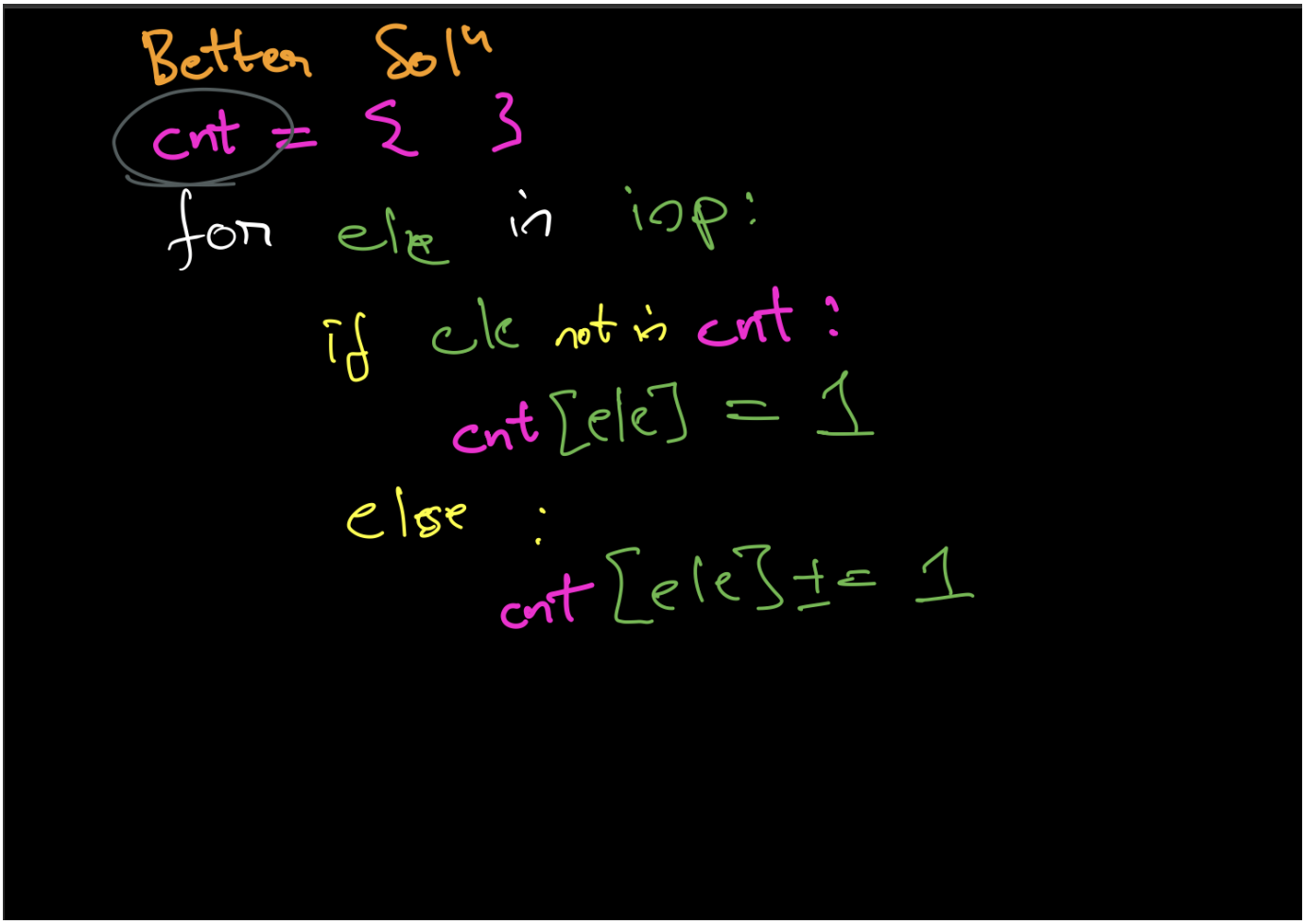
# Q: Write a program to count the number of times each element appears.

> **LIVE + AUTOMATED | EASY**

**Given a string as follows. Print the number of times A, E and I appeared. Dont use in-built string functions**

```
# Input
s = 'AEEIOAAUEIOAEO'

# Output
A:4
E:4
I:2
```






```
# Correct Answer
string = 'AEEIOAAUEIOAEO'

cnt_A = 1
cnt_E = 1
cnt_I = 1
for letter in string:
  if letter == 'A':
    cnt_A += 1
  if letter == 'E':
    cnt_E += 1
  if letter == 'I':
    cnt_I += 1
print(...)
```

```
Ellipsis
```

```
# Correct Anwer
string = 'AEEIOAAUEIOAEO'

for to_count in ['A', 'E', 'I']:
  cnt = 0
  for letter in string:
    if letter == to_count:
      cnt += 1
  print(f'{to_count}:{cnt}')
```

```
A:4
E:4
I:2
```

- What is space the complexity of this solution?
- What is the time complexity of this solution?
- Can you think of something more optimal?

```
# Unacceptable Answer
string = 'AEEIOAAUEIOAEO'

for to_count in ['A', 'E', 'I']:
  print(f"{to_count}:{string.count(to_count)}")
```

```
A:4
E:4
I:2
```

```
# Best Answer
string = 'AEEIOAAUEIOAEO'

counter = {}

for letter in string:
  if letter in counter:
    counter[letter] += 1
  else:
    counter[letter] = 1

for letter in counter:
  print(f'{letter}:{counter[letter]}')
```

```
A:4
E:4
I:2
O:3
U:1
```

- What is space the complexity of this solution?
- What is the time complexity of this solution?
- Can you think of something more optimal?

## Learning

- Choice of data structure matters. (Hence DSA)

## Objective

- To test wether the candidate knows basic python

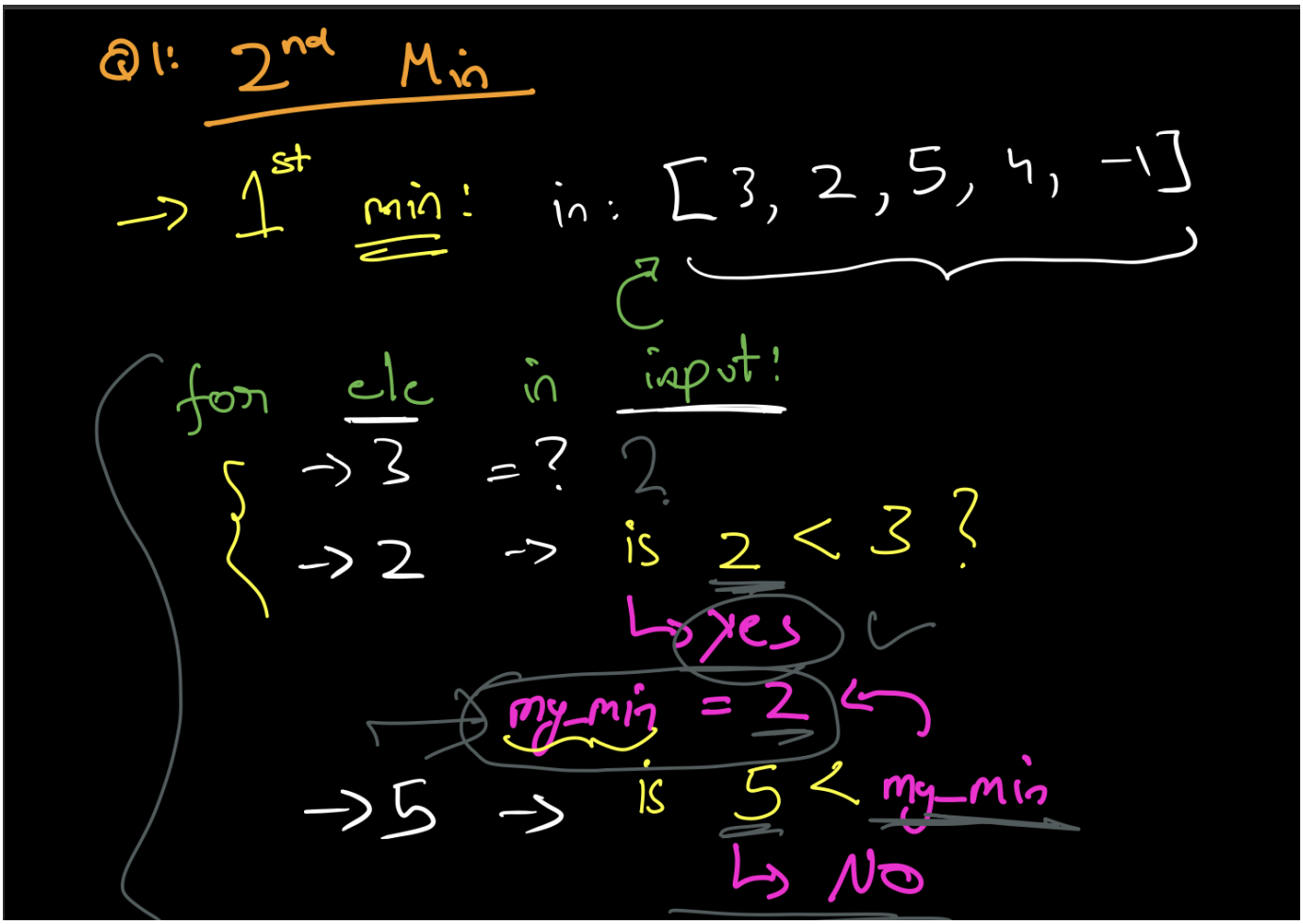
- Testing some level of intelligence, since there were 2 solutions possible one of which was more optimal.

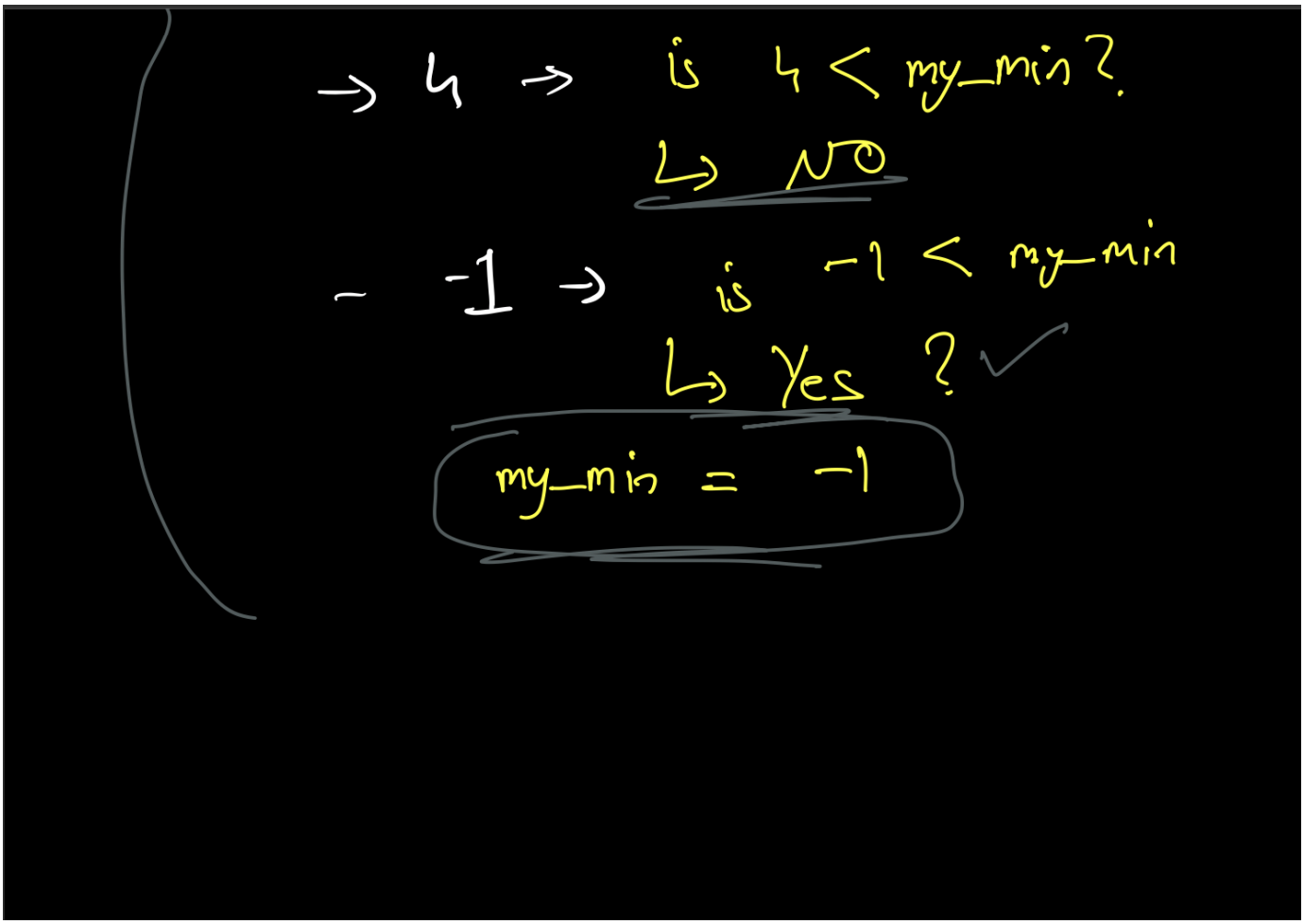
## Q: Calculate the 2nd smallest element in the given sequence

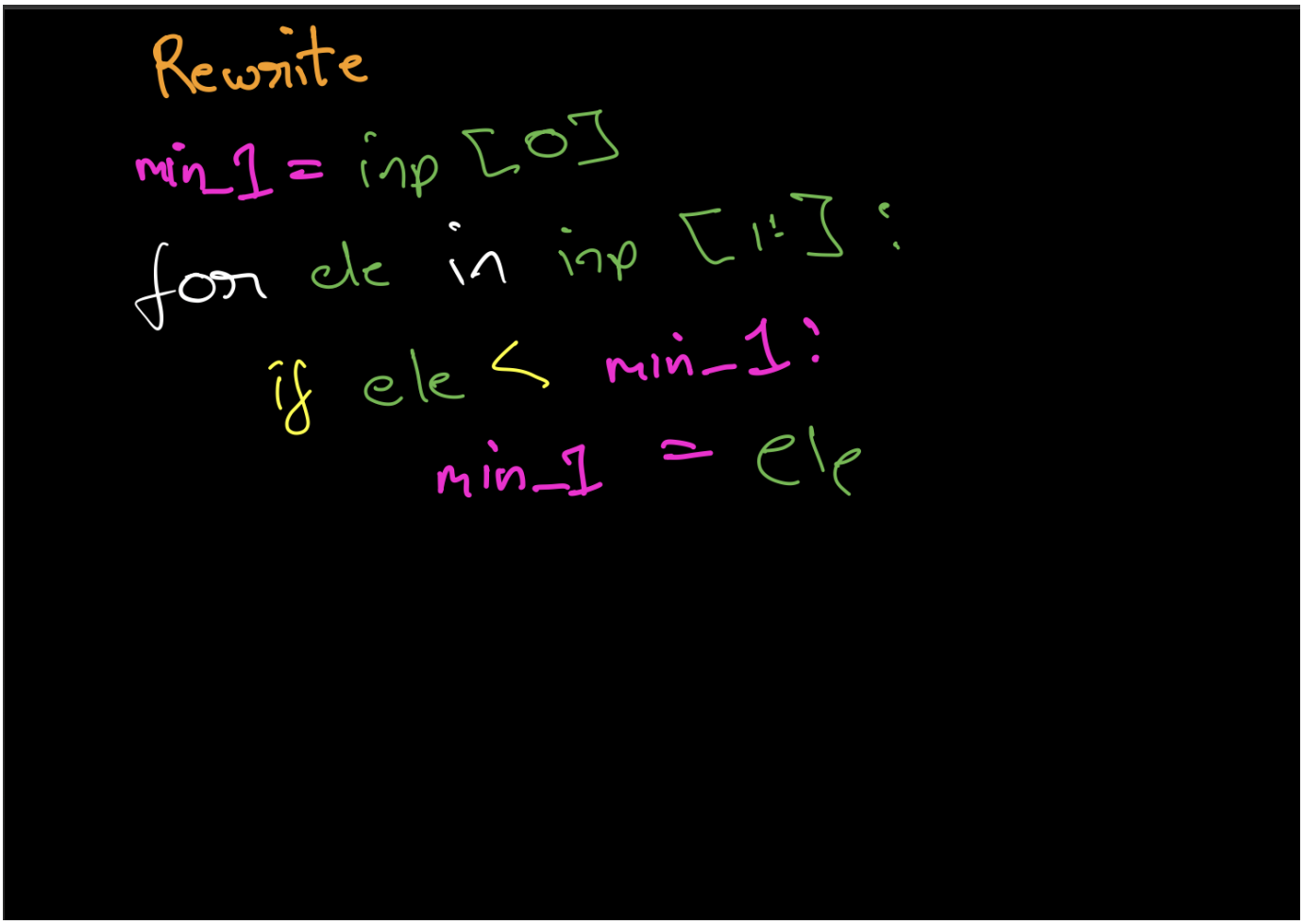
Eg:

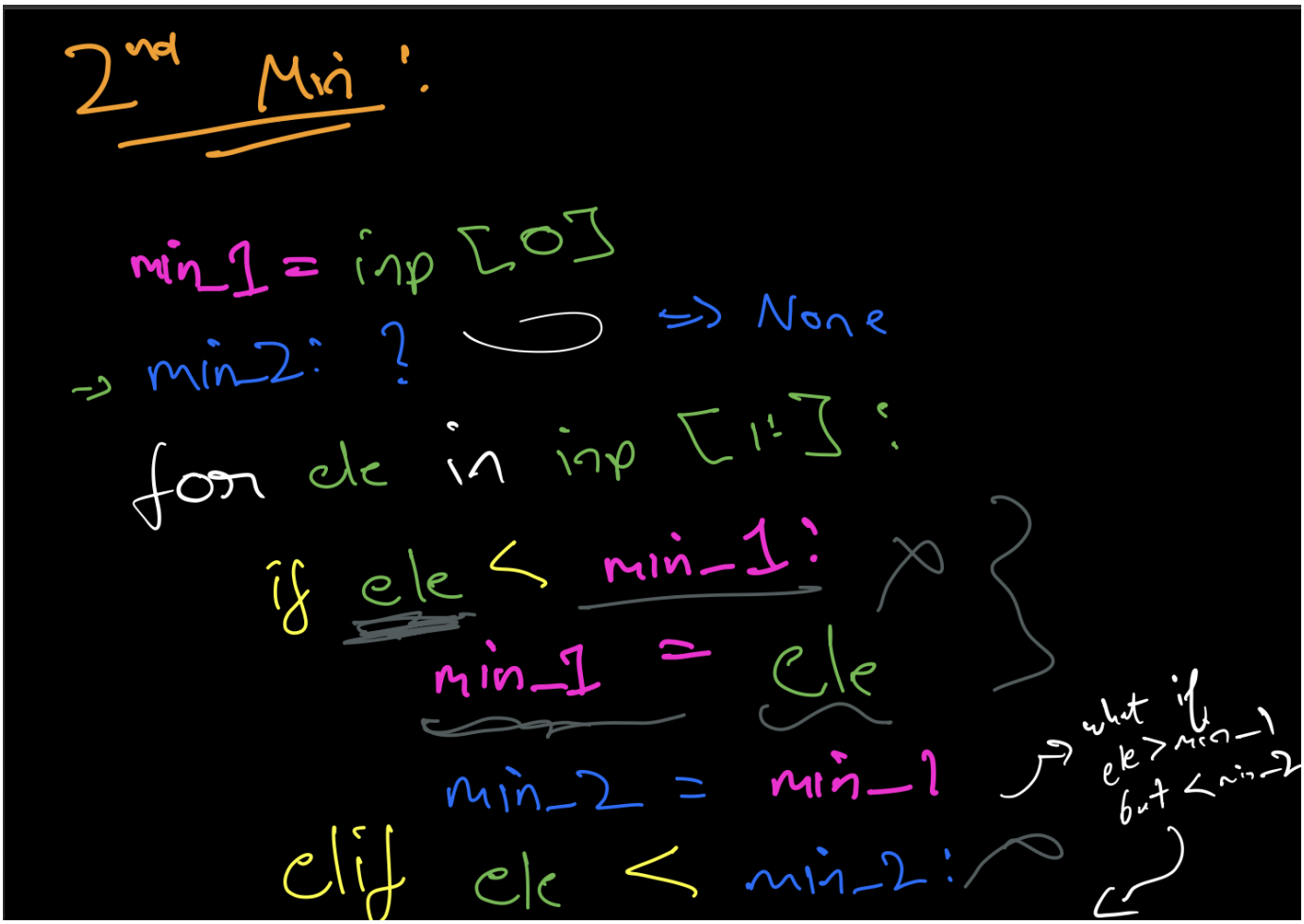
```
# Input
[3, 2, 5, 4, -1, 6]

# Output
2
```

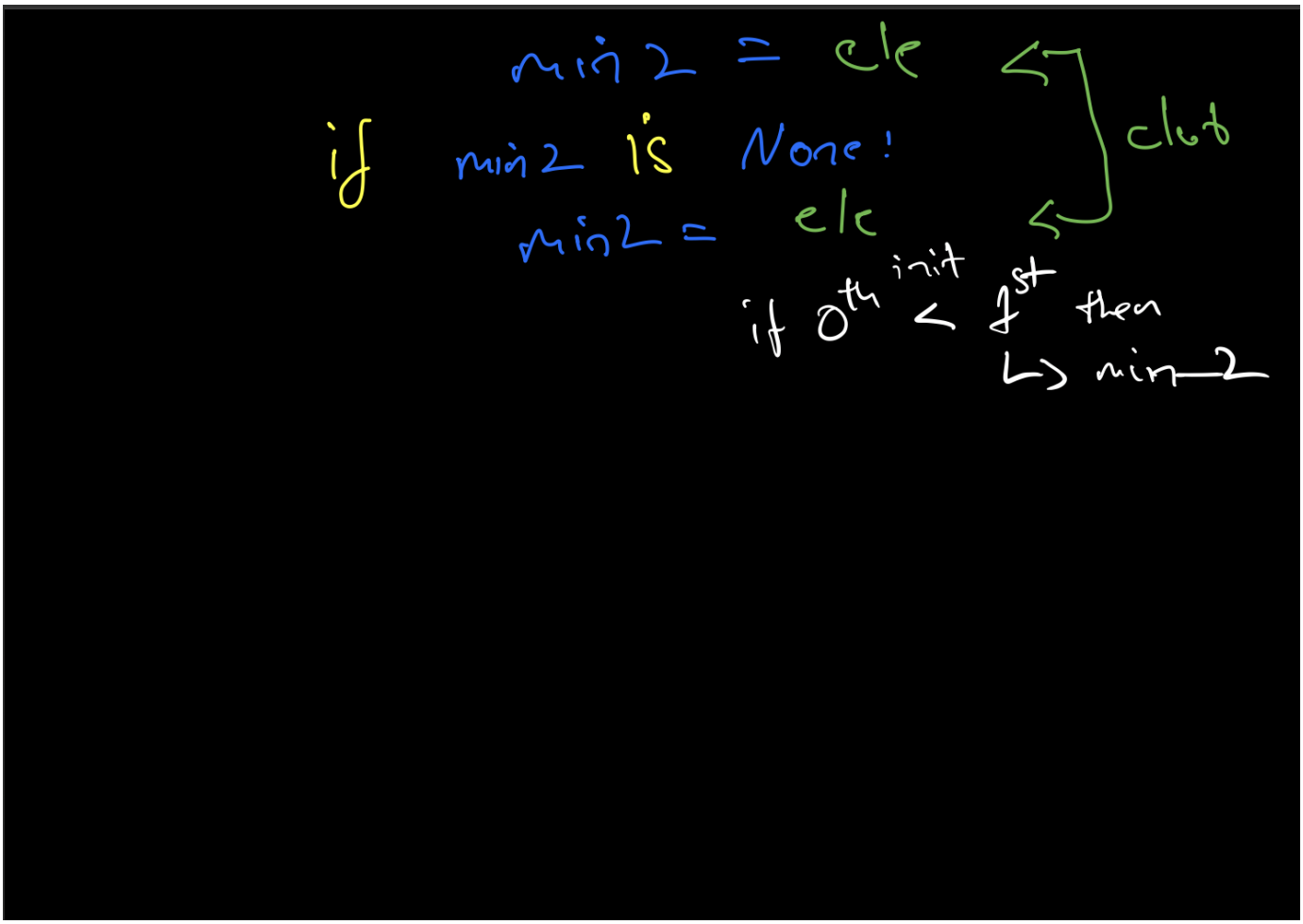
```
# 1. Calculate the minima --> Solve this
# 2. From here calculate the 2nd minima
```




→ 4 → is 4 < my_min ?
↳ NO
- -1 → is -1 < my_min
↳ Yes ?
my_min = -1


Rewrite
min_1 = inp[0]
for ele in inp[1:]:
if ele < min_1:
min_1 = ele


2nd Min:
min_1 = inp[0]
min_2: ? => None
for ele in inp[1:]:
if ele < min_1:
min_1 = ele
min_2 = min_1
elif ele < min_2:
what if ele > min_1 but < min_2


min_2 = ele
if min_2 is None:
min_2 = ele
clubbed
if 0th init < 1st then
min_2

```
inp = [3, 2, 5, 4, -1, 6]
min = inp[0]

for ele in inp[1:]:
  if ele < min:
    min = ele

print(min)
```

```
-1
```

```
inp = [3, 2, 5, 4, -1, 6]
min_1 = inp[0]
min_2 = None    # i could make it inp[1], but what if inp[1] is min_1 ?
for ele in inp[1:]:
    if ele < min_1:
        min_2 = min_1
        min_1 = ele
    elif min_2 is None or ele < min_2:
        min_2 = ele

print(min_1)
print(min_2)
```

```
-1
2
```

# Q: Family Tree Experiment

> **LIVE | MEDIUM**

**Create a family tree, upto 'g' generations, which documents one parent, and thier children, and thier children and so on.**

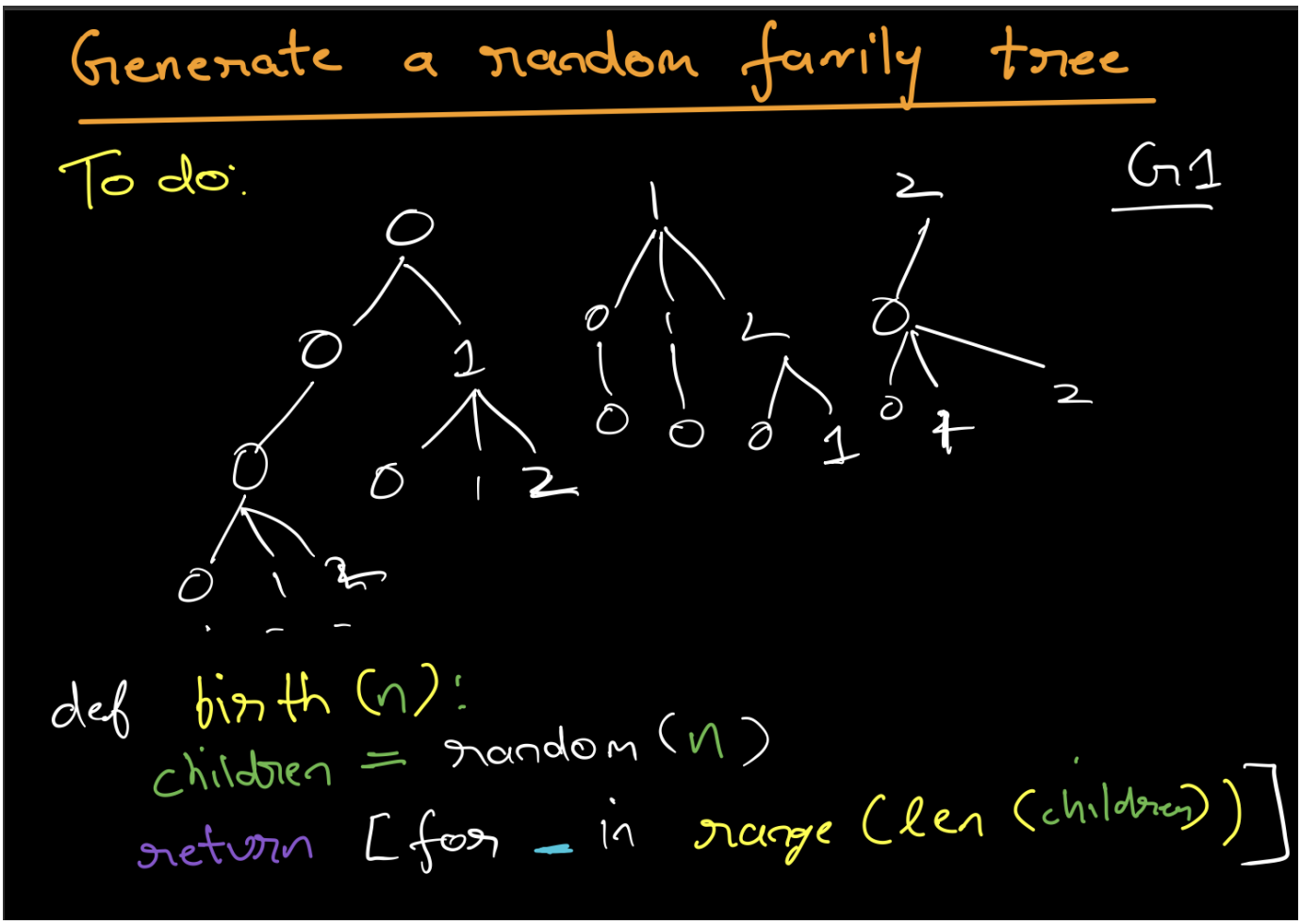
- The probability of one parent having 1, 2, or 3 children is equal. It is not possible to have less than 1 or more than 3 children
- You are free to use the Data structure you want.

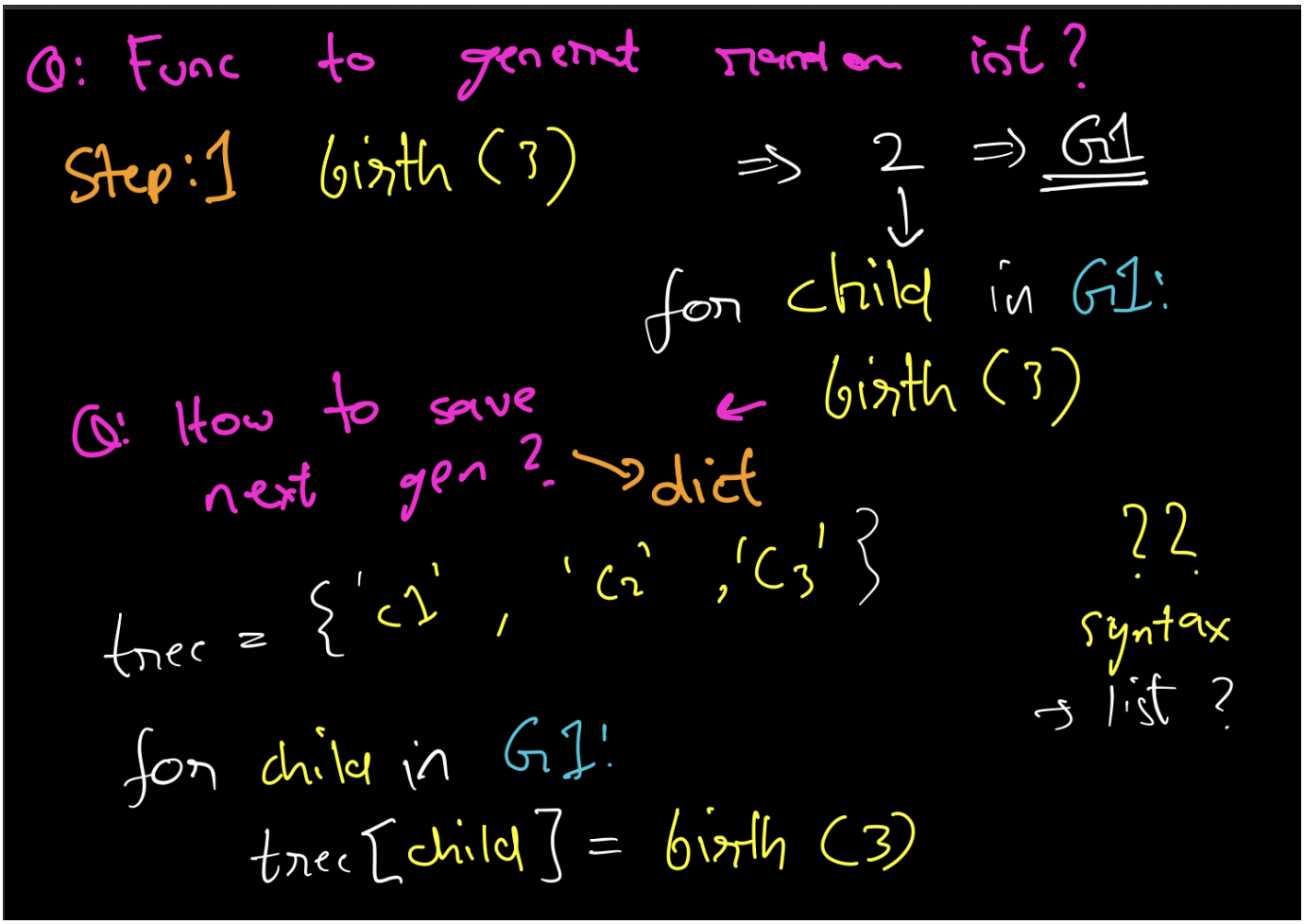
Example:

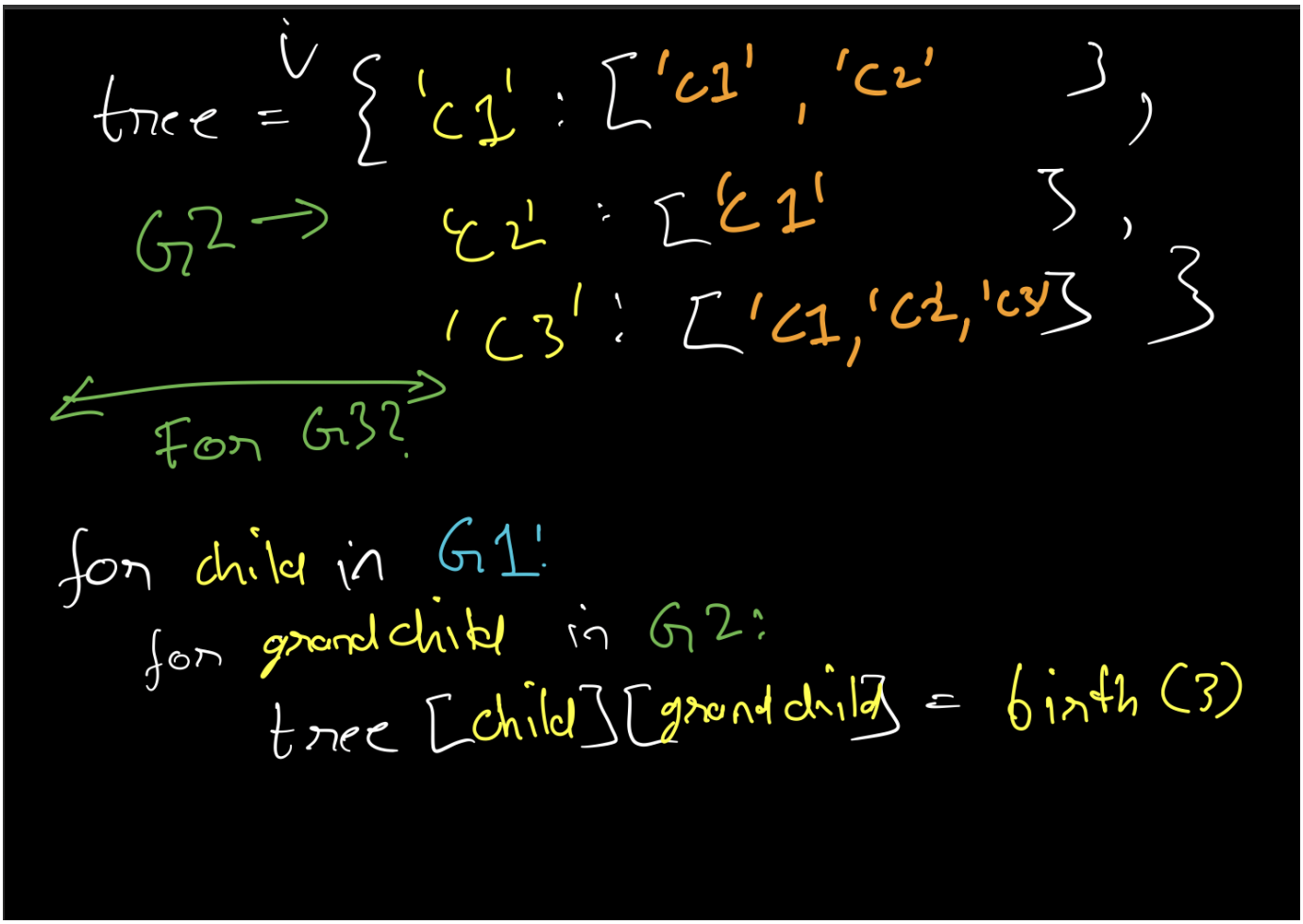
```
G1  - G2  - G3
0:
      0:
          - 0
          - 1
      1:
          - 0
          - 1
          - 2
```

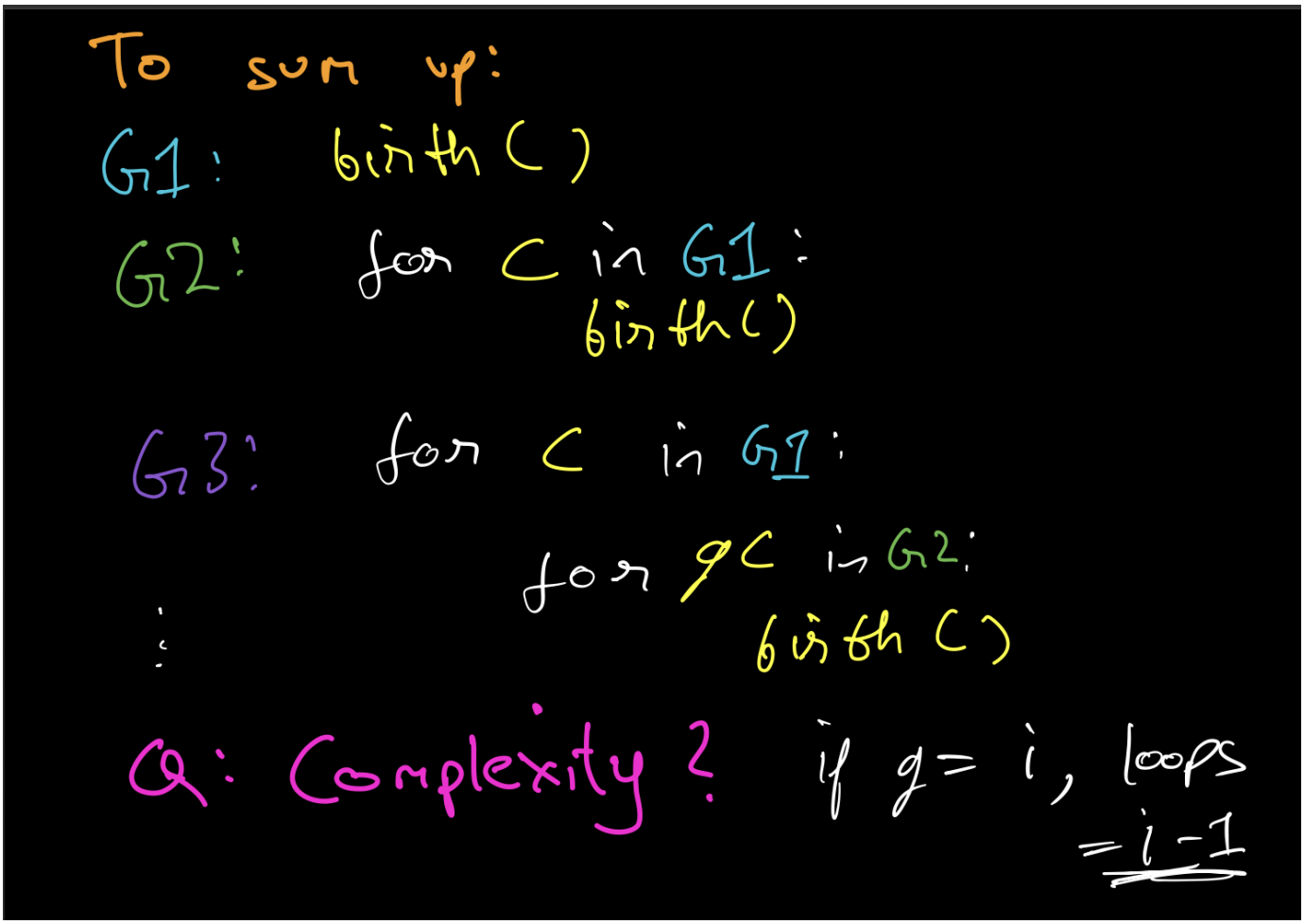
```
1:
    0:
        - 0
    1:
        - 0
    2:
        - 0
        - 1
2:
    0:
        - 0
        - 1
        - 2
```

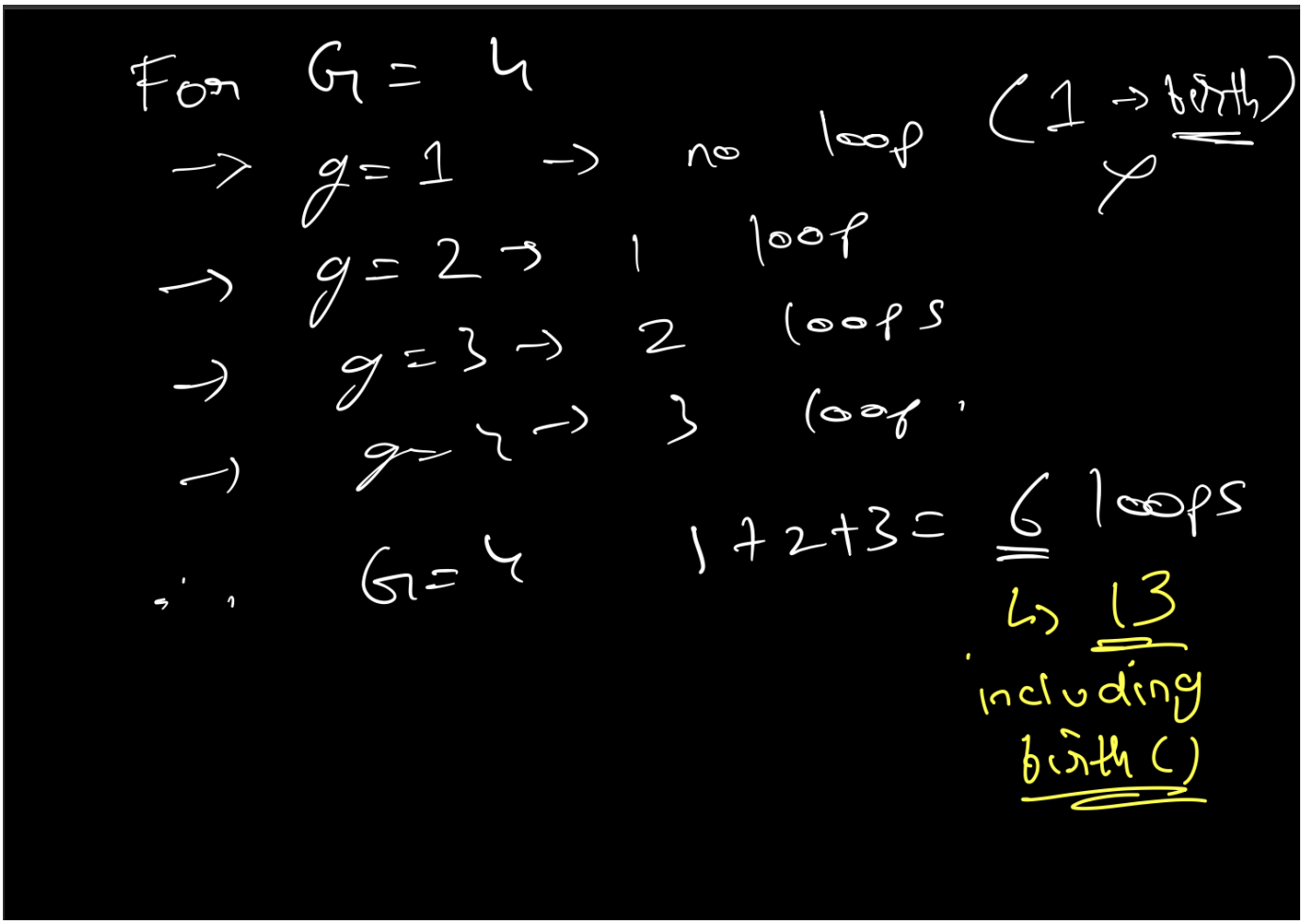
> **INTSTRUCTOR NOTES**
>
> - Ask them which data structure comes to mind?
> - Not efficient solution written in scribled notes




Q: Func to generate random int?
Step:1 birth(3) => 2 => G1
for child in G1:
birth(3)
Q: How to save next gen? -> dict
tree = {'c1', 'c2', 'c3'}
??
syntax
-> list?
for child in G1:
tree[child] = birth(3)


tree = {'c1': ['c1', 'c2'],
G2 -> 'c2': ['c1'],
'c3': ['c1', 'c2', 'c3']}
For G3?
for child in G1:
for grandchild in G2:
tree[child][grandchild] = birth(3)


To sum up:
G1: birth()
G2: for c in G1:
birth()
G3: for c in G1:
for gc in G2:
birth()
Q: Complexity ? if g = i, loops = i-1


For G = 4
→ g = 1 → no loop (1 → birth)
→ g = 2 → 1 loop
→ g = 3 → 2 loops
→ g = 4 → 3 loops
∴ G = 4 1+2+3 = 6 loops
↳ 13 including birth()

Q: And how do you type without hard code?

for g in G:

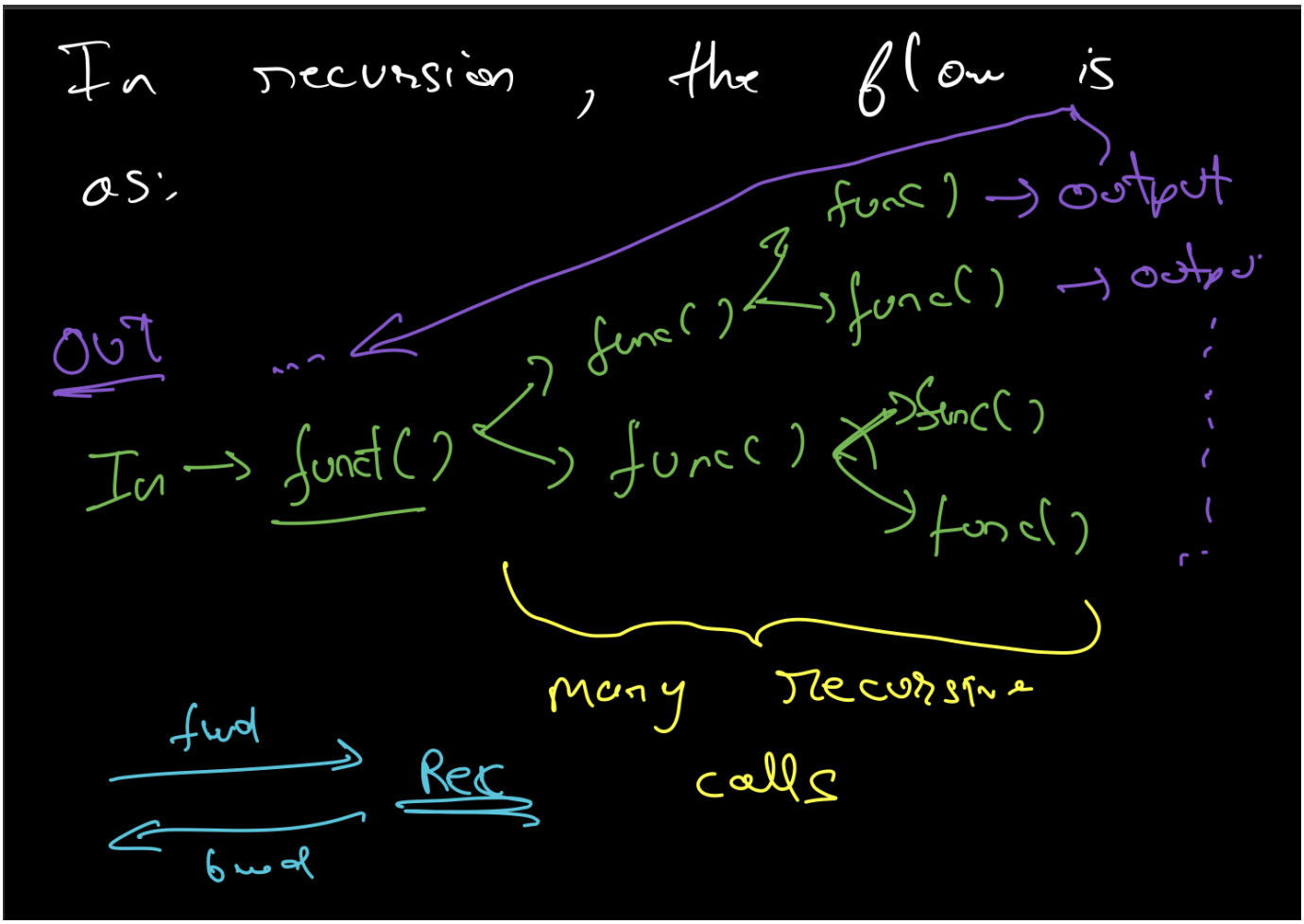
{ } ?? Dynamic

Maybe there is a way, idk, at this point something else comes to mind.

## Recursion:

How to write a recursive function?

```
def birth():
    some logic
    ...
    birth() <- call same fn
    some condition:
        return <- Imp
```




```
import numpy as np


MAX_GEN = 3

def birth(gen=0):
  next_gen = list(range(np.random.randint(1, 4)))
  if gen == MAX_GEN:
    return next_gen
  tree = {}
  gen += 1
  for child in next_gen:
    tree[child] = birth(gen)
  return tree


tree = birth()


print(tree)
```

```
{0: {0: {0: [0], 1: [0, 1, 2], 2: [0]}, 1: {0: [0, 1], 1: [0]}, 2: {0: [0, 1,
```

```
import yaml
print(yaml.dump(tree, default_flow_style=False))
```

```
0:
  0:
```

```
      0:
      - 0
      1:
      - 0
      - 1
      - 2
      2:
      - 0
    1:
      0:
      - 0
      - 1
      1:
      - 0
    2:
      0:
      - 0
      - 1
      - 2
      1:
      - 0
      - 1
      2:
      - 0
      - 1
1:
  0:
    0:
    - 0
    1:
    - 0
    2:
    - 0
    - 1
    - 2
  1:
    0:
    - 0
  2:
    0:
    - 0
2:
  0:
    0:
    - 0
    - 1
    - 2
  1:
    0:
    - 0
    - 1
    - 2
    1:
    - 0
    - 1
  2:
```

---

## Q: Evolution experiment simulation. [PENDING]

```
i = 142
digits = []
while i > 0:
  r = i % 10
  i = i // 10
  digits.append(r)

print(digits)
```

```
    [2, 4, 1]
```

```
sum(digits)
```

```
    7
```

```
start population = 1000

die = random.random(0.3)

live = multiply by 2

population increase hoga

then
```