CS61A

RECURSION, TREE RECURSION, LISTS

LOGISTICS AND REMINDERS

- Hog is due **Today**
- Hog Contest due **Tomorrow**
- Lab04 due **Today**
- Lab05 due **Friday**
- **CATS** project released

AGENDA

- Recursion
- Tree Recursion
- Lists

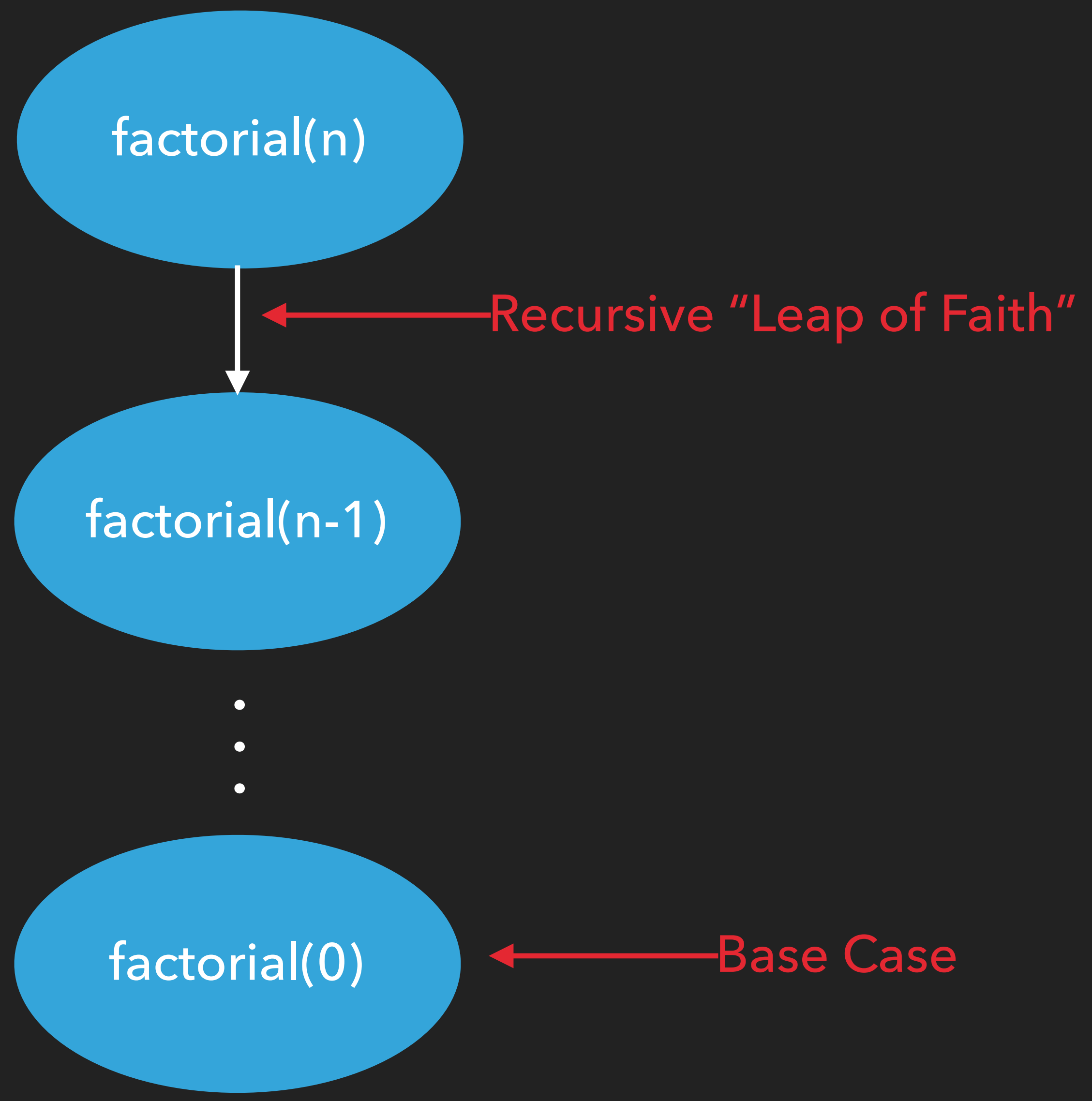
WHAT IS RECURSION

- When a function is defined in terms of **calls to itself**
- Canonical example: factorial

```
def factorial(n):
    total = 1
    for i in range(1, n + 1):
        total *= i
    return total
```

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

VISUALIZING RECURSION

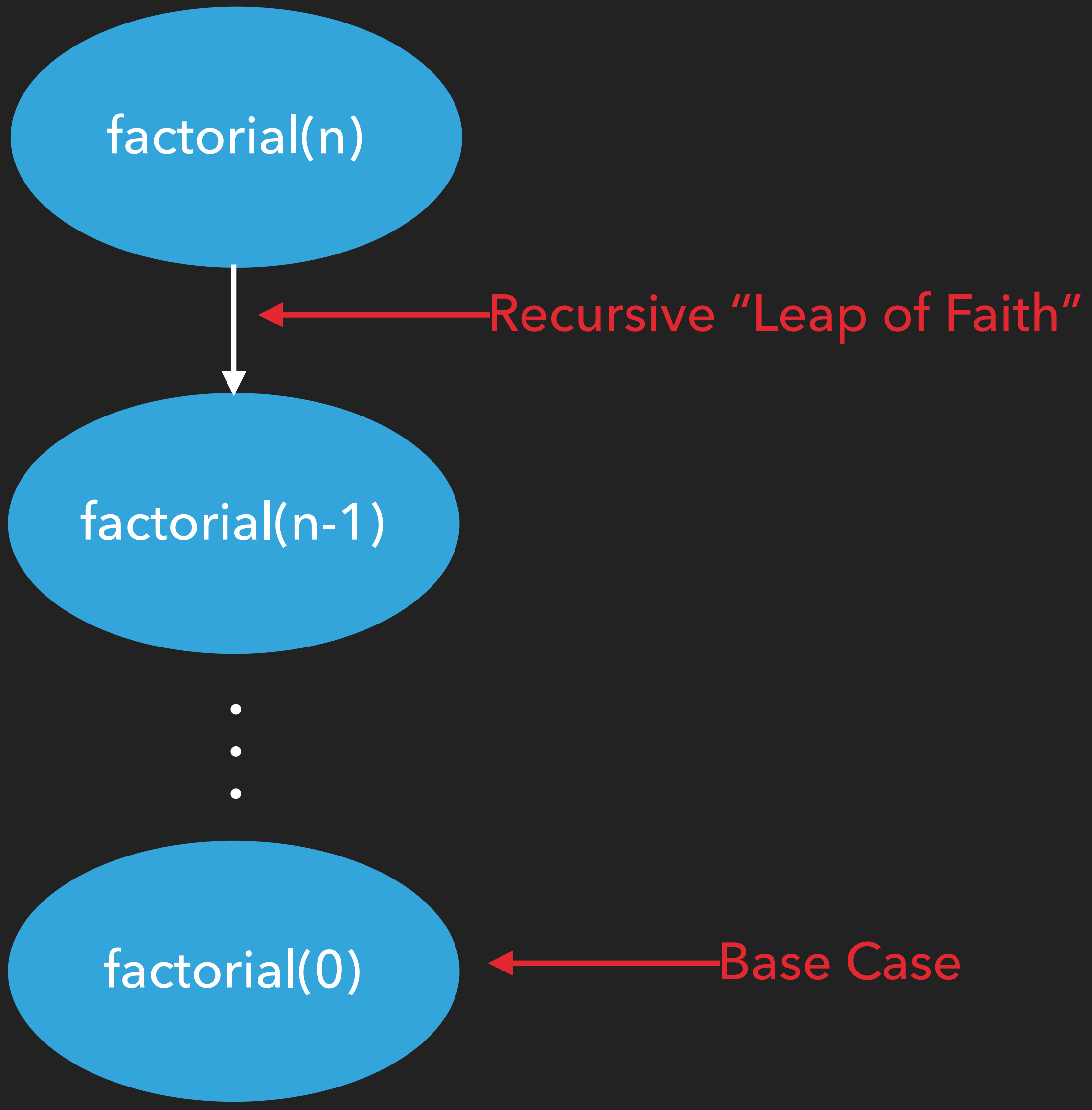


```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Tips

- Break problem into sub-problems
- Think in context of the problem
- Don't think past an arrow
- Write/draw out examples
- Think stupid and simple

VISUALIZING RECURSION

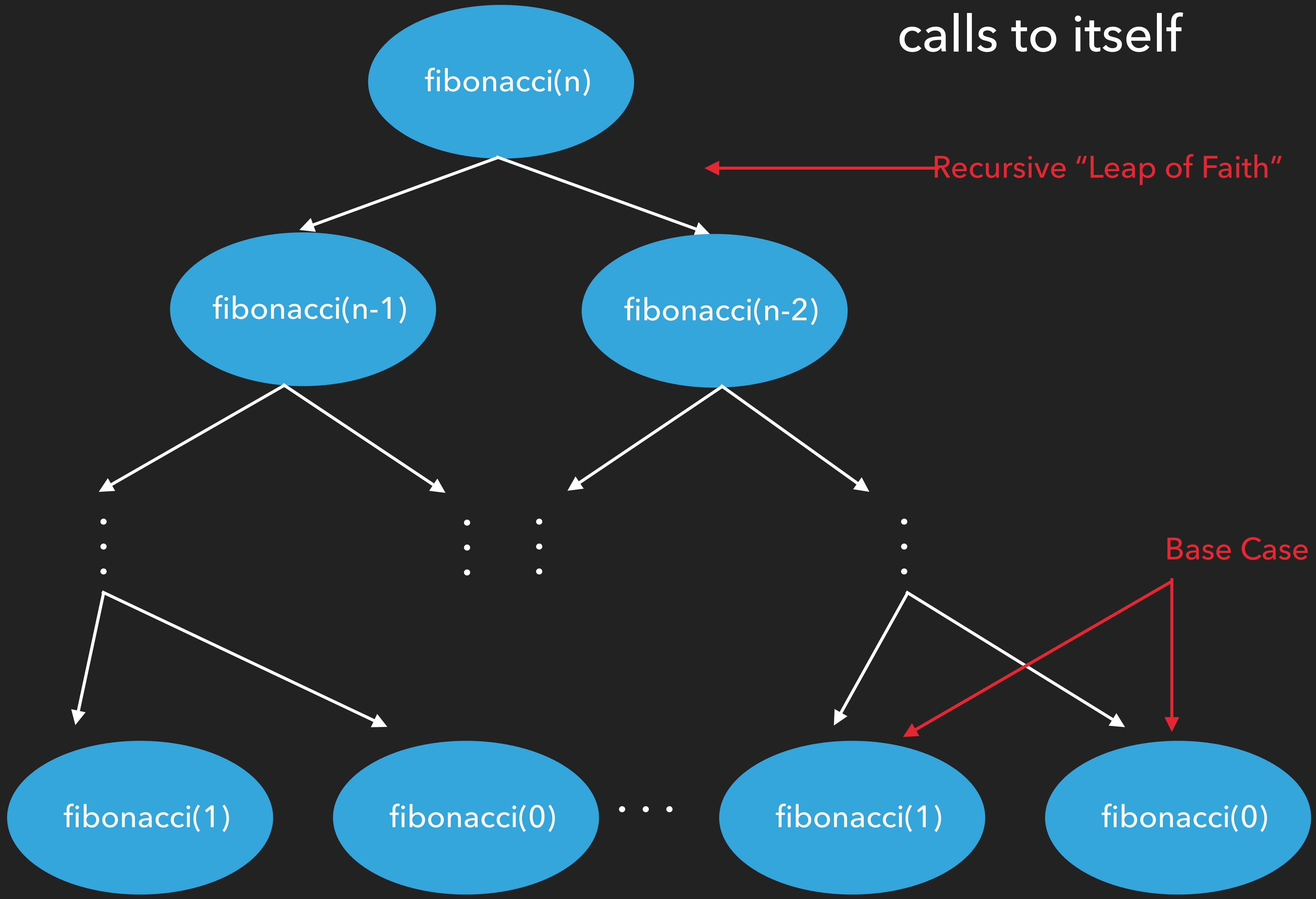


Why the leap of faith works?

For those of you who have taken discrete math of some kind have heard of this leap of faith as induction

TREE RECURSION

Recursion but with multiple calls to itself



OTHER THINGS TO KNOW

- Problems can have multiple sets of bases cases that work just fine
- Problems can potentially be solved with different recurrences (ex. count change)
- Many times drawing writing things out without looking at code makes the pattern very obvious
- Generally think in terms of “big” inputs

LISTS

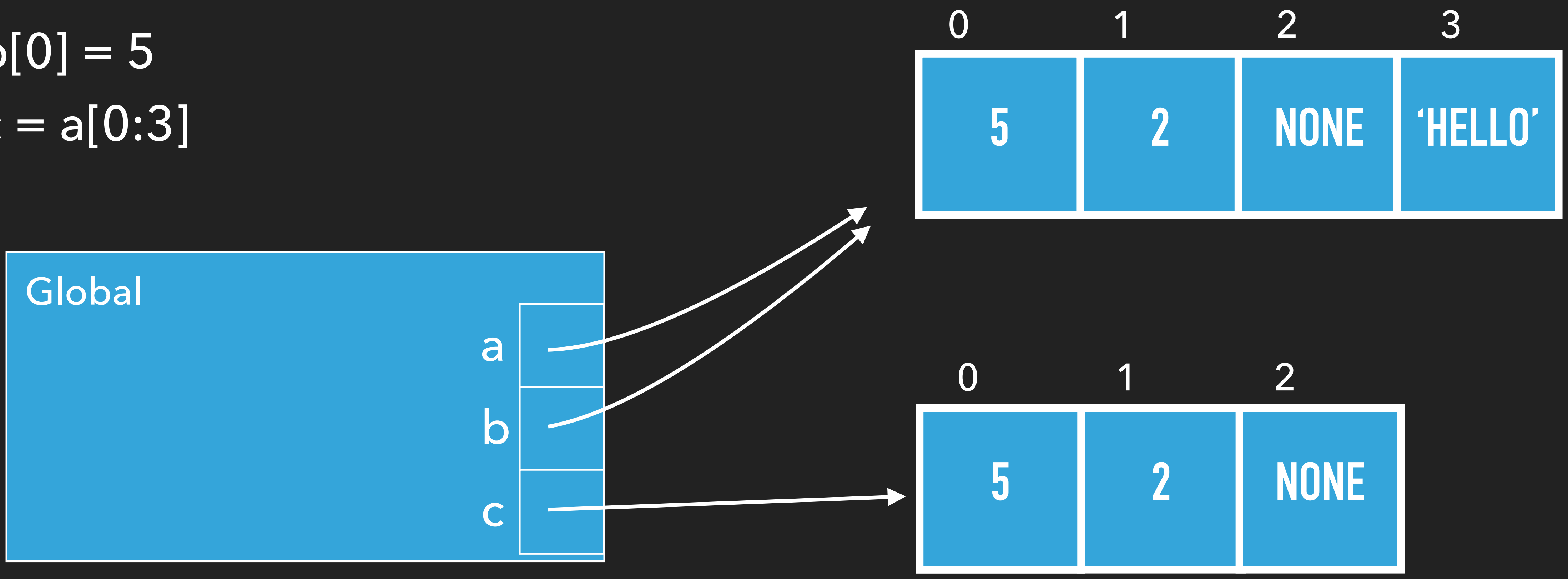
- a = [1,2, None, 'hello']
- a[0]

1

- a[3]

'hello'

- b = a
- b[0] = 5
- c = a[0:3]


Global
a
b
c
0
1
2
3
5
2
NONE
'HELLO'
0
1
2
5
2
NONE