



# Recursive Structures & Processes

# Announcements

**LS13 (Recursive Structures)** due today at 11:59pm

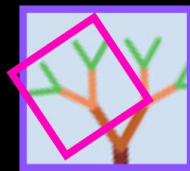
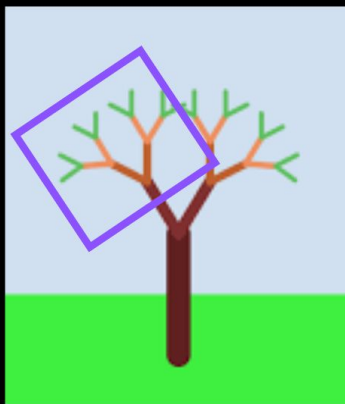
**Re: Quiz 03:**

- Grades will be returned to you today. Thank you for your patience!

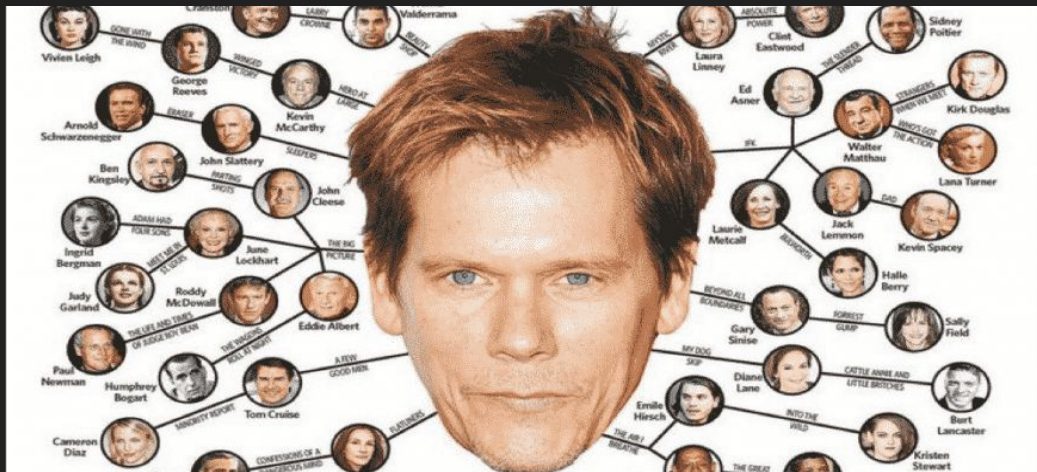
# Recursion: defining an operation/object in terms of itself

A real-world phenomenon! Examples:

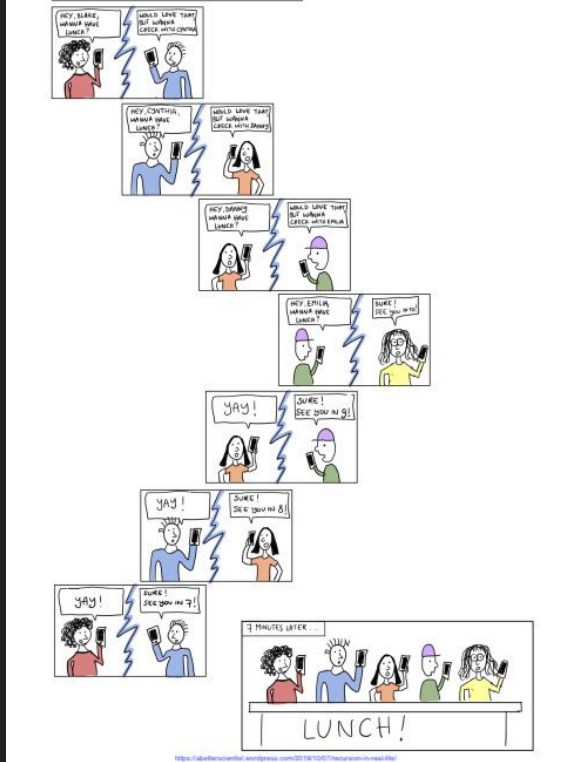
- **You** have **parents**, who have **parents**, who have **parents**, who have **parents**, who...  
... were **early humans**
- A **tree** has **branches**, which have **branches**, which have **branches**, which...  
... have **leaves**



# Different recursive structures for different purposes



Six degrees of Kevin Bacon  
graph/network



Coordinating plans with  
individual phone calls  
linked list

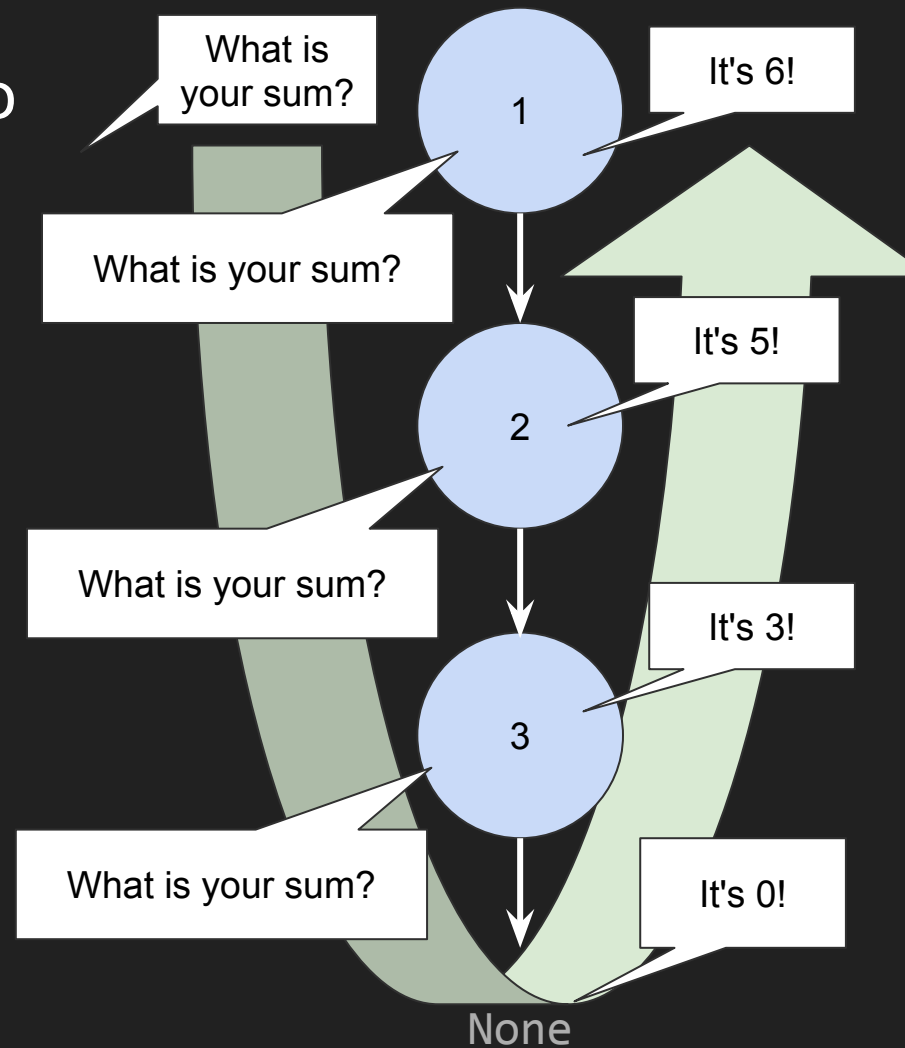
# Anatomy of a Singly-Linked List

# Memory diagram

```
1  from __future__ import annotations # Ignore for now!
2
3  class Node:
4      value: int
5      next: Node | None
6
7      def __init__(self, val: int, next: Node | None):
8          self.value = val
9          self.next = next
10
11  # Note: There are no errors!
12  two: Node = Node(2, None)
13  one: Node = Node(1, two)
14  # We'll extend this diagram shortly, leave room
```

# A Recursive `sum` Algorithm Demo

1. When you are asked, "what is your sum?"
2. Ask the **next** Node, "what is your sum?"  
Wait patiently for an answer!
3. Once the answer is returned back to you, add **your value to it**, then turn to the person who asked you and give them this answer.



# Let's write a recursive function called `sum`!

```
1  from __future__ import annotations  # Ignore for now!
2
3  class Node:
4      value: int
5      next: Node | None
6
7      def __init__(self, val: int, next: Node | None):
8          self.value = val
9          self.next = next
10
11  # Note: There are no errors!
12  two: Node = Node(2, None)
13  one: Node = Node(1, two)
14  # We'll extend this diagram shortly, leave room
```

Write a function called `sum` that adds up the `values` of all `Nodes` in the linked list.



# Diagramming the `sum` function call

```
1  from __future__ import annotations
2
3  class Node:
4      value: int
5      next: Node | None
6
7      def __init__(self, val: int, next: Node | None):
8          self.value = val
9          self.next = next
10
11  # Note: There are no errors!
12  two: Node = Node(2, None)
13  one: Node = Node(1, two)
14
15  def sum(head: Node | None) -> int:
16      if head is None:
17          return 0
18      else:
19          rest: int = sum(head.next)
20          return head.value + rest
21
22  print(sum(one))
```

# For reference: recursive function checklist:

## Base case:

- ❑ Does the function have a clear base case?
  - ❑ Ensure the base case returns a result directly (without calling the function again).
- ❑ Will the base case *always* be reached?

## Recursive case:

- ❑ Does the function have a recursive case that *progresses toward the base case*?
  - ❑ Does the recursive call have the right arguments? The function should call itself on a simpler or smaller version of the problem.
- ❑ Have you tested your function with multiple cases, including edge cases?