CL30 – Recursive Structures & Processes

Announcements

Re: Quiz 03:

- *Regrade requests will be open **till 11:59pm (tonight)!***
  - Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric

Re: Quiz 04:

- Practice quiz is on the site
  - Come to Tutoring to work through it with TAs today (5-7pm in SN011)!
- If you have a UAA and want to reschedule your quiz to another date, please let me know!

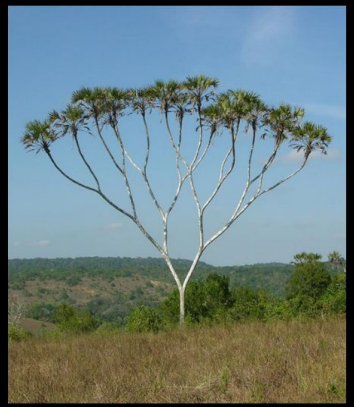
Assignments:

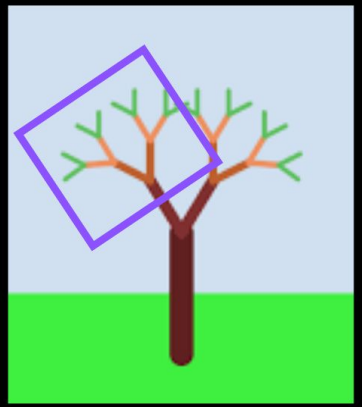
- LS13: Recursive Structures released today, due *tomorrow* (April 10)

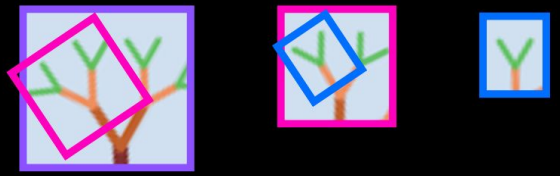
Recursion: defining an operation/object in terms of itself

A real-world phenomenon! Examples:

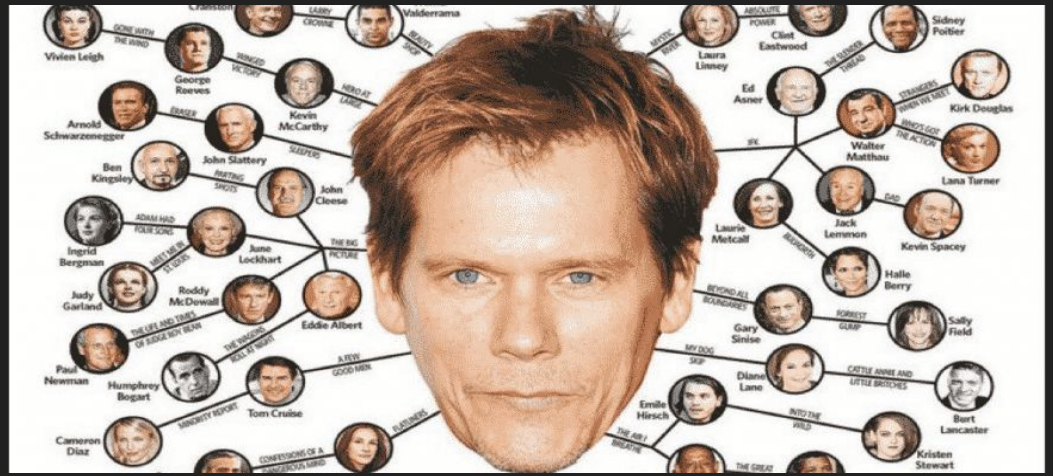
- **You** have **parents**, who have **parents**, who have **parents**, who have **parents**, who…
  … were the **first humans**
- A **tree** has **branches**, which have **branches**, which have **branches**, which…
  … have **leaves**





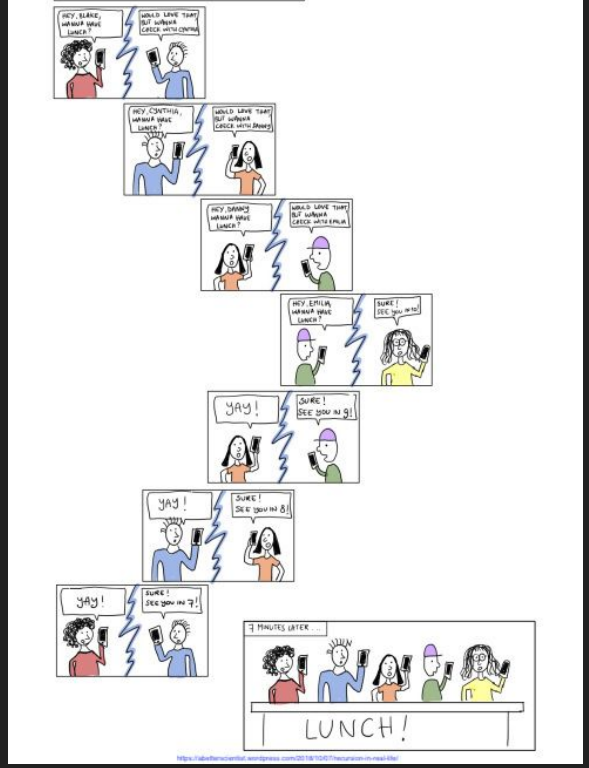


Different recursive structures for different purposes



Six degrees of Kevin Bacon

graph/network



Coordinating plans before 3-way calls were possible

linked list

Anatomy of a Singly-Linked List

Memory diagram

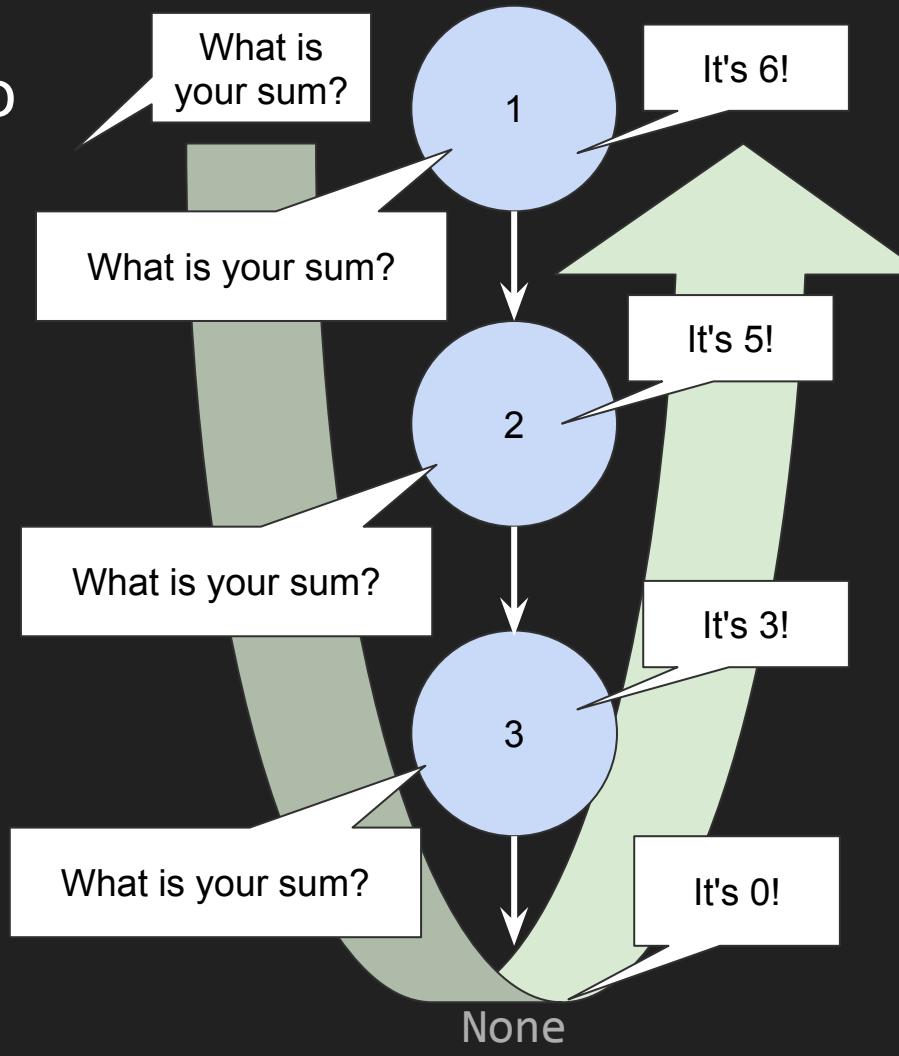
```python
from __future__ import annotations  # Ignore for now!

class Node:
    value: int
    next: Node | None

    def __init__(self, val: int, next: Node | None):
        self.value = val
        self.next = next

# Note: There are no errors!
two: Node = Node(2, None)
one: Node = Node(1, two)
# 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`!

```python
from __future__ import annotations  # Ignore for now!

class Node:
    value: int
    next: Node | None

    def __init__(self, val: int, next: Node | None):
        self.value = val
        self.next = next

# Note: There are no errors!
two: Node = Node(2, None)
one: Node = Node(1, two)
# We'll extend this diagram shortly, leave room
```

Write a function called `sum` that adds up the `value`s of all `Node`s in the linked list.

Diagramming the `sum` function call

```python
from __future__ import annotations

class Node:
    value: int
    next: Node | None

    def __init__(self, val: int, next: Node | None):
        self.value = val
        self.next = next

# Note: There are no errors!
two: Node = Node(2, None)
one: Node = Node(1, two)

def sum(head: Node | None) -> int:
    if head is None:
        return 0
    else:
        rest: int = sum(head.next)
        return head.value + rest

print(sum(one))
```

For reference: checklist for developing a recursive function:

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:

- ❏ Ensure the function moves closer to the base case with each recursive call.
- ❏ Combine returned results from recursive calls where necessary.
- ❏ Test the function with edge cases (e.g., empty inputs, smallest and largest valid inputs, etc.). Does the function account for these cases?