RECURSION AND THE CALL STACK

LECTURE 06-1

JIM FIX, REED COLLEGE CSCI 121

SOME THINGS

- On Thursday our class consultant is going to give a survey.
- Project 2 is posted.
  - You’ll work with strings, lists, and dictionaries.
  - You’ll write functions that make secret codes.
  - You’ll write functions that break secret codes!

RECURSIVE FUNCTIONS

- Recall: we invented a recursive function to compute this sum:

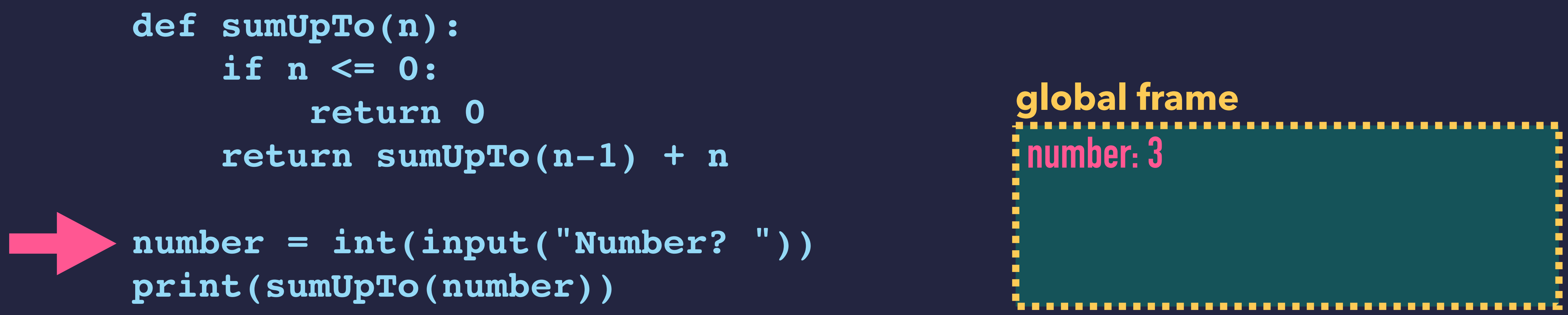
```
(1 + 2 + 3 + ... + (n-1)) + n == ????
```

- This one considers non-positive sums as "trivially 0":

```
def sumUpTo(n):
    if n <= 0:
        return 0
    else:
        return sumUpTo(n-1) + n
```

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:



```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```

global frame

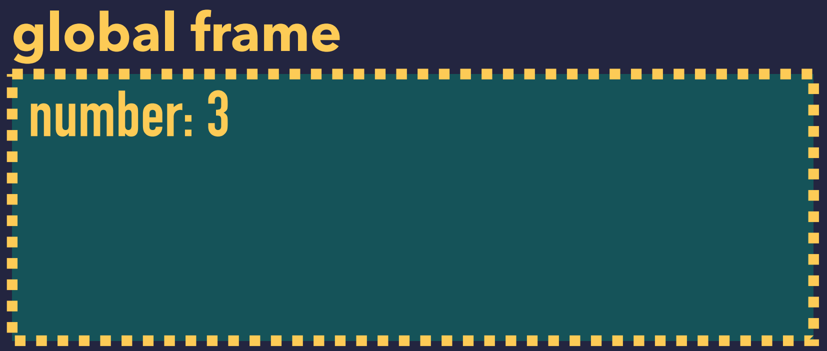
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



global frame

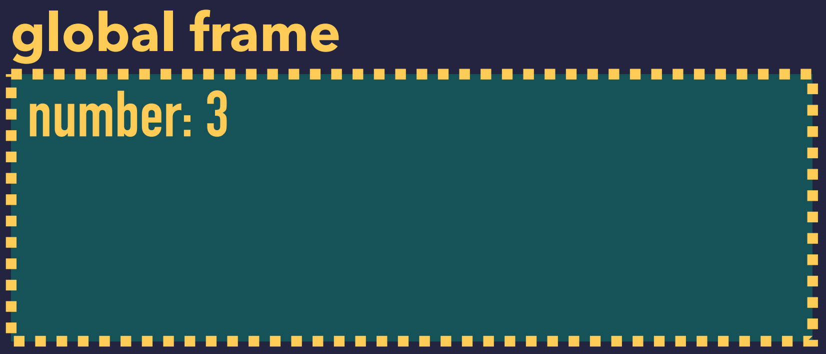
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



global frame

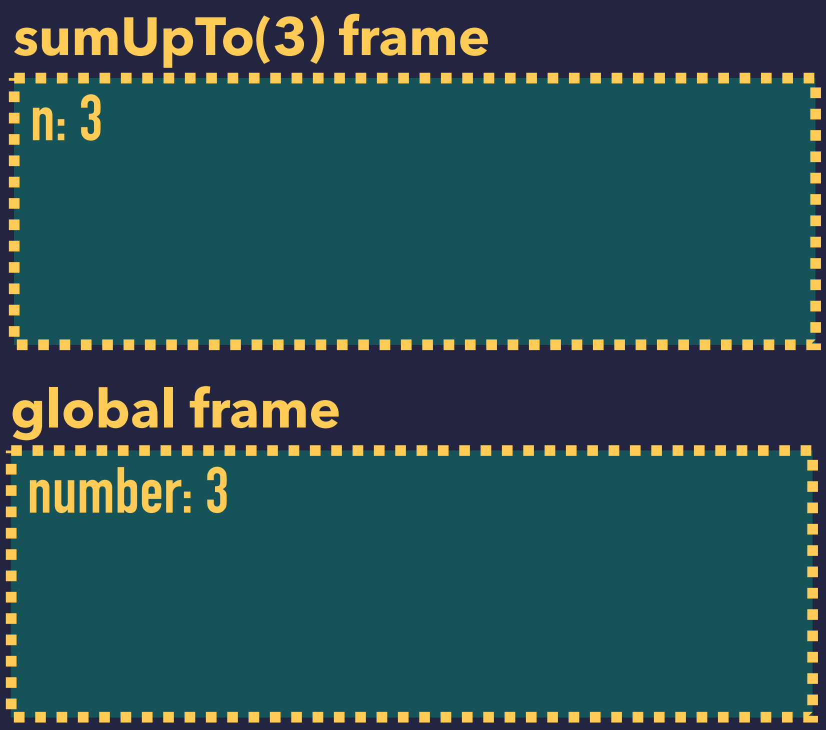
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(3) frame

n: 3

global frame

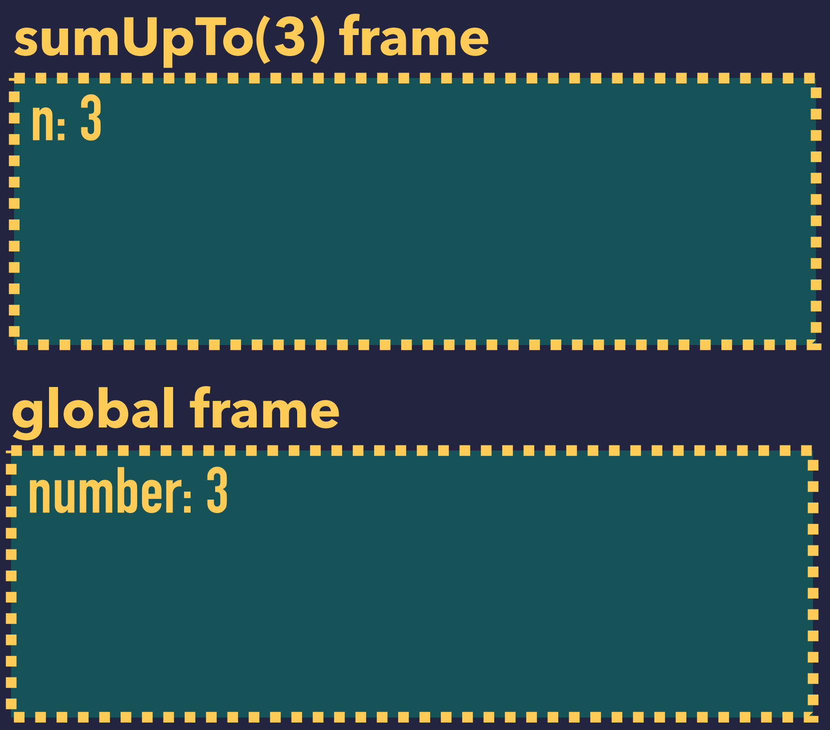
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(3) frame

n: 3

global frame

number: 3

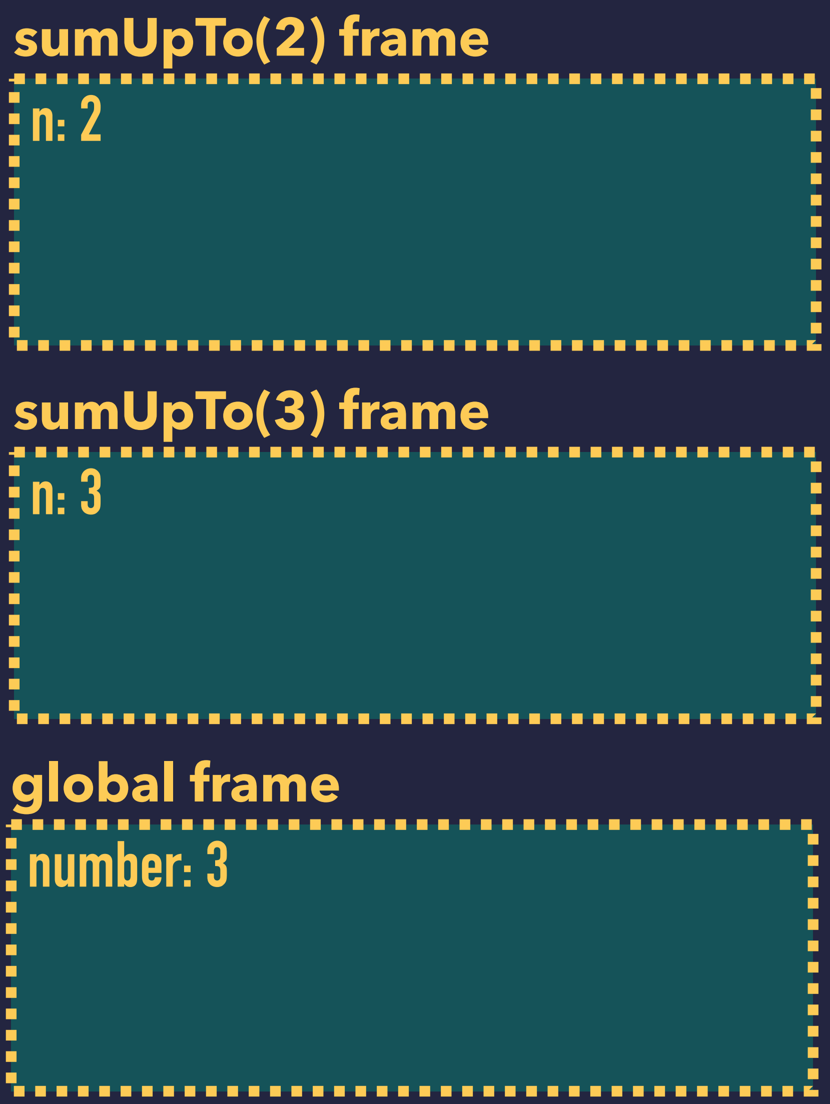
PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(2) frame

n: 2

sumUpTo(3) frame

n: 3

global frame

number: 3

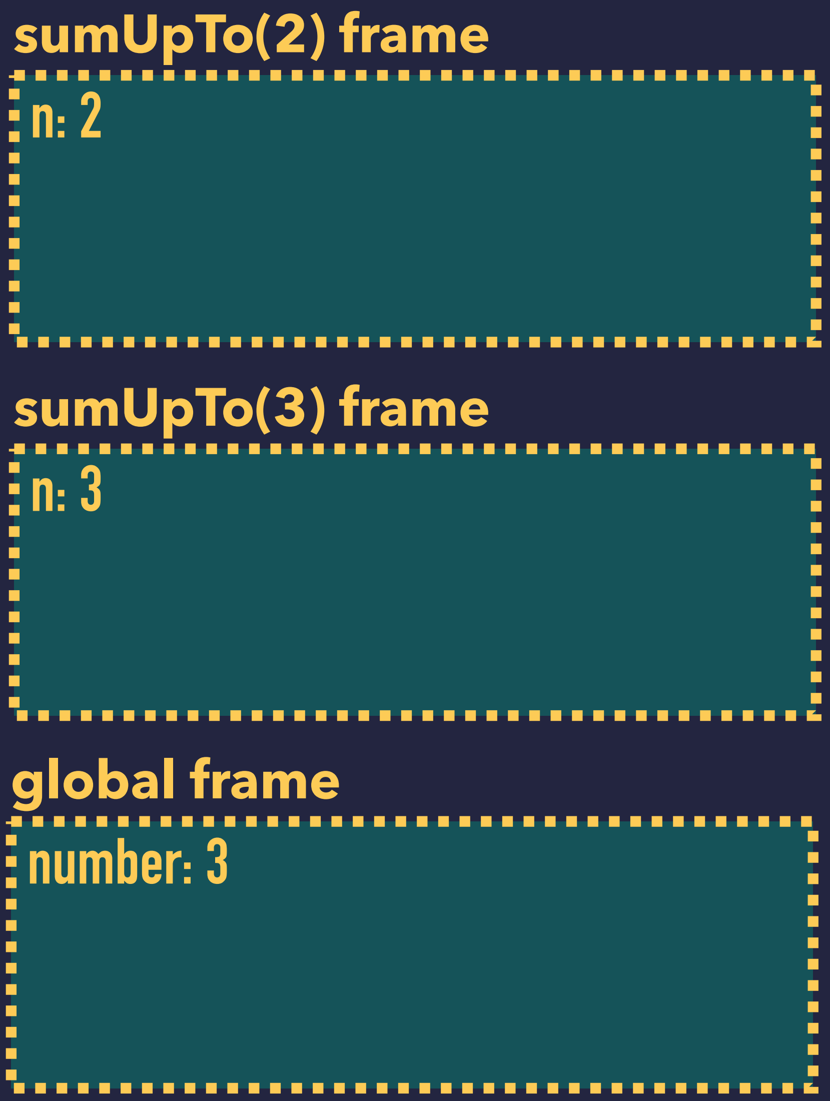
PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

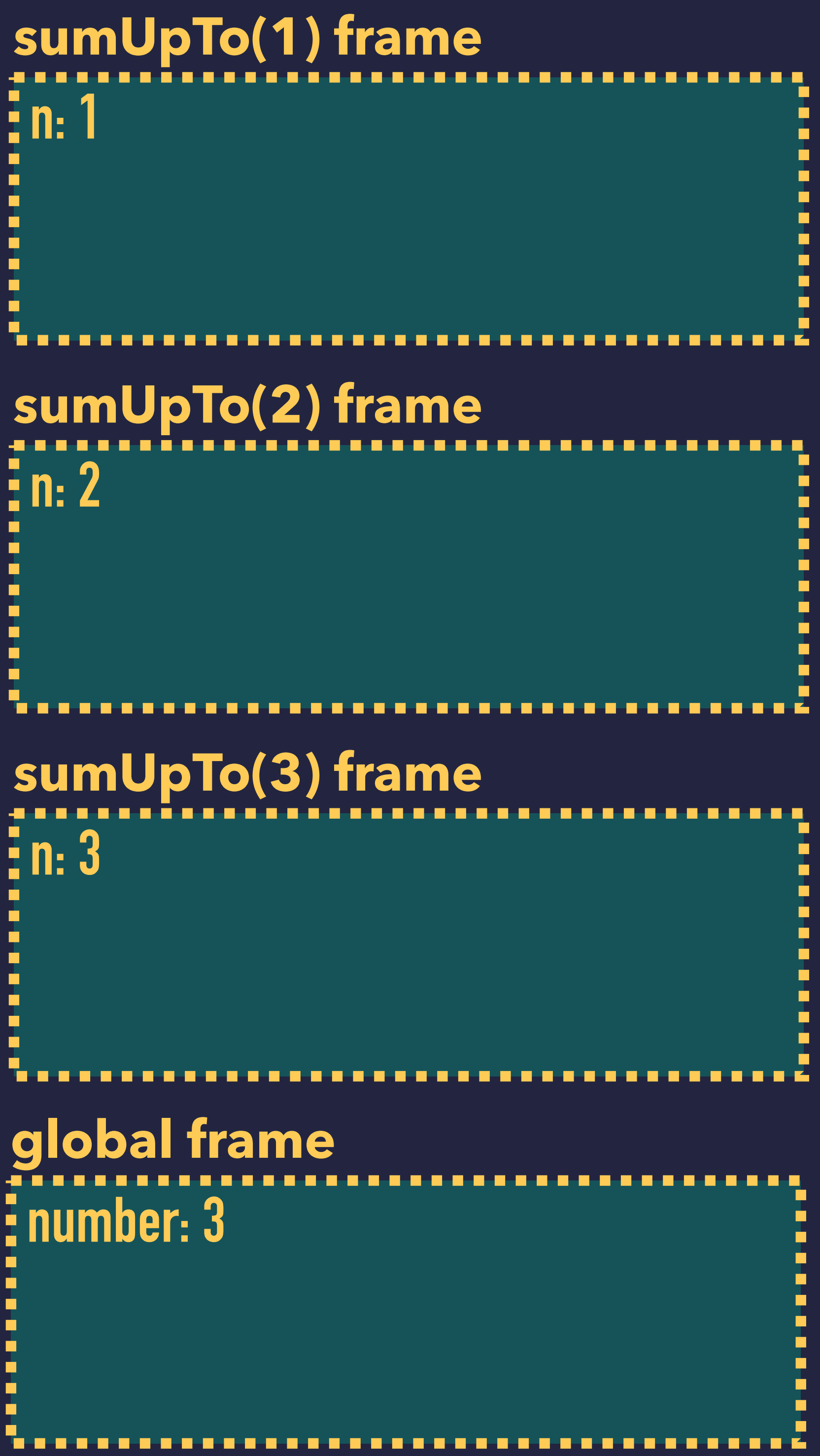
- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(1) frame

n: 1

sumUpTo(2) frame

n: 2

sumUpTo(3) frame

n: 3

global frame

number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

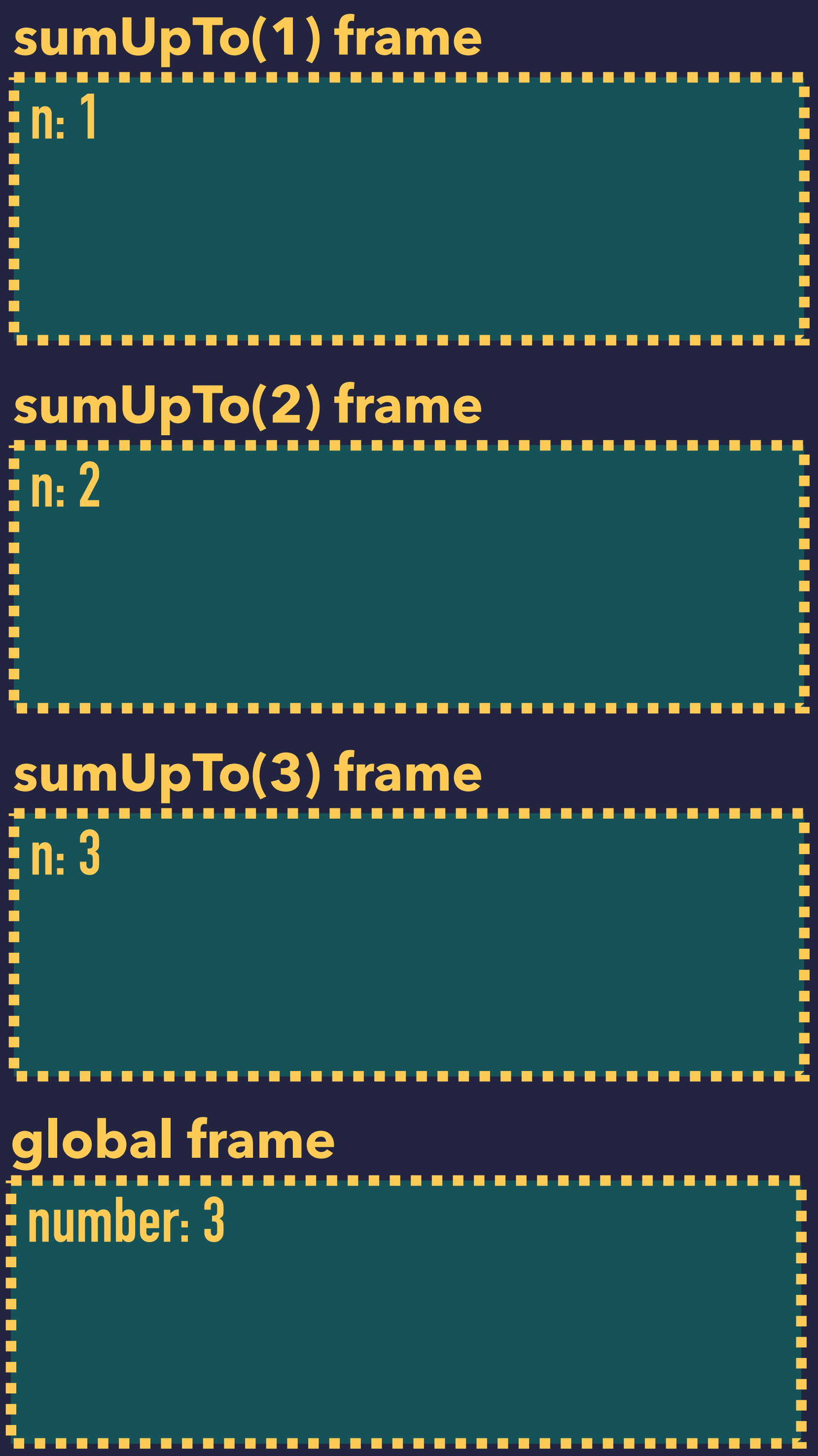
- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(1) frame

n: 1

sumUpTo(2) frame

n: 2

sumUpTo(3) frame

n: 3

global frame

number: 3

PYTHON'S EXECUTION OF A RECURSIVE

- Let's take a look at Python's execution of this script:

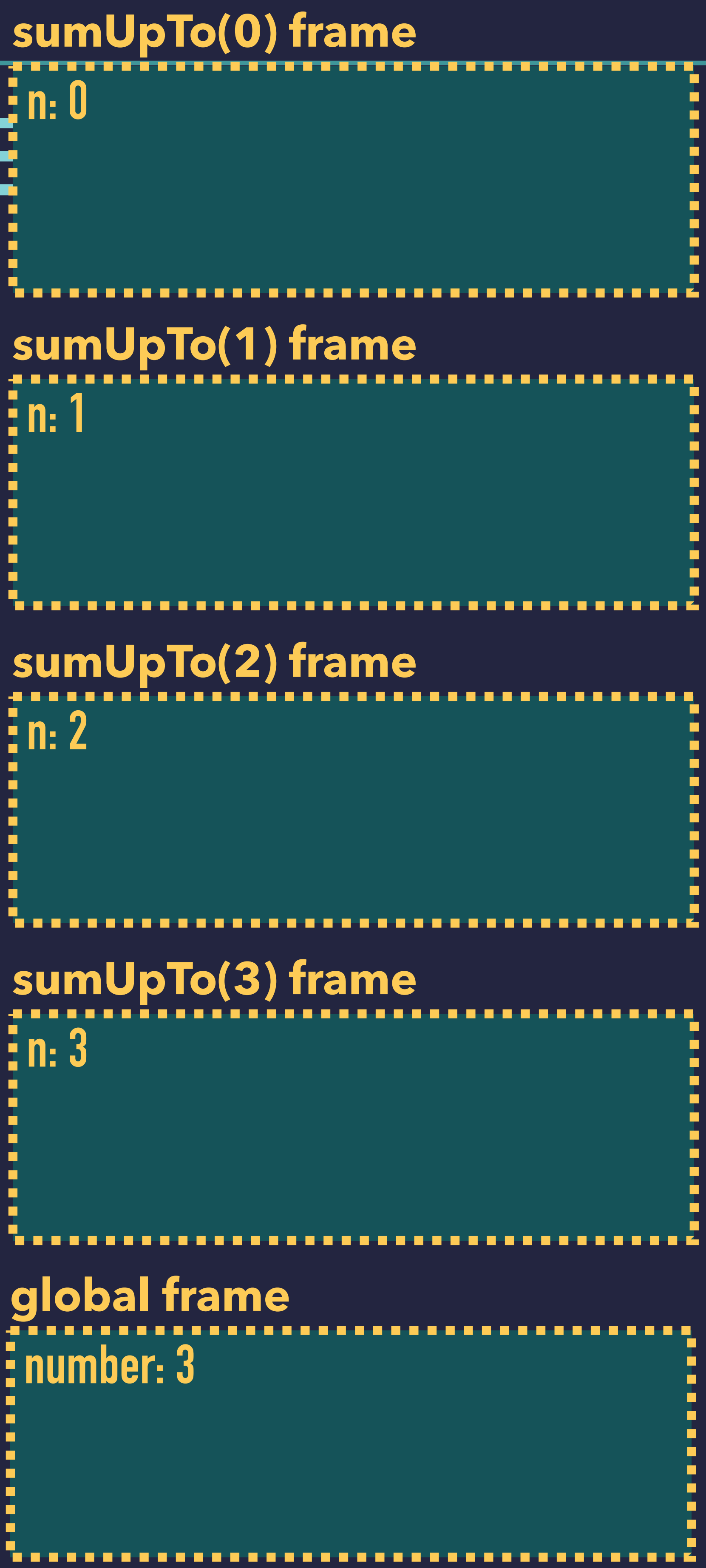
```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



PYTHON'S EXECUTION OF A RECURSIVE

- Let's take a look at Python's execution of this script:

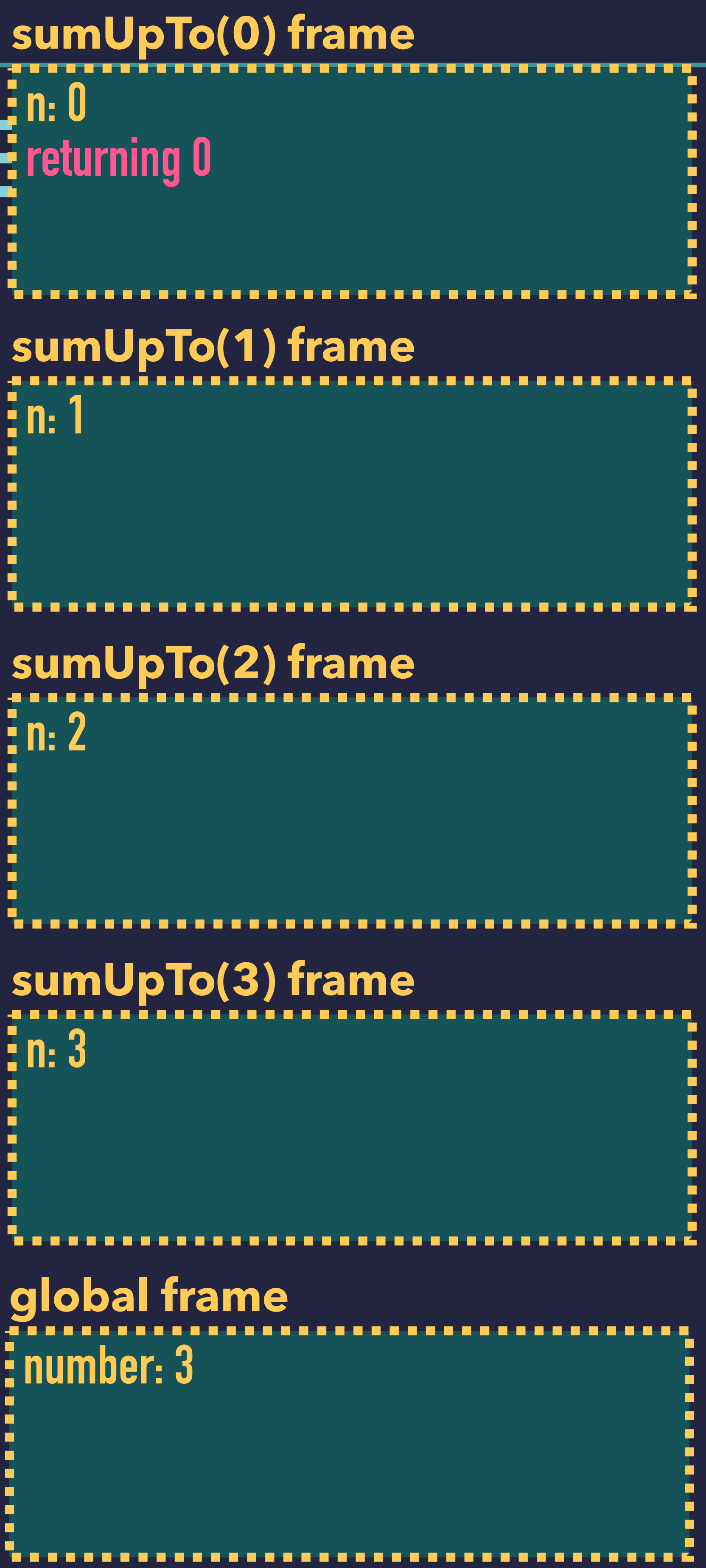
```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(0) frame

n: 0

returning 0

sumUpTo(1) frame

n: 1

sumUpTo(2) frame

n: 2

sumUpTo(3) frame

n: 3

global frame

number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

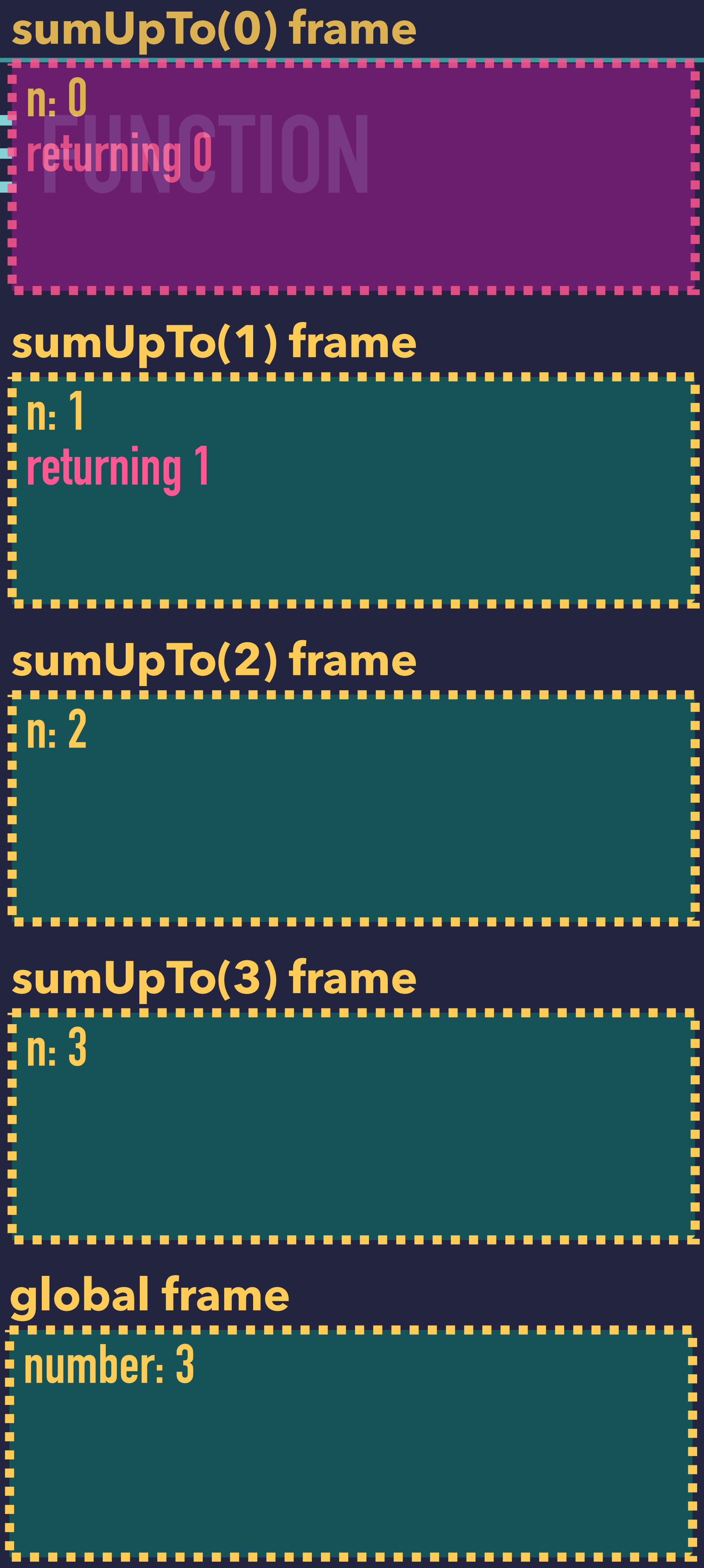
- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(0) frame

n: 0

returning 0

sumUpTo(1) frame

n: 1

returning 1

sumUpTo(2) frame

n: 2

sumUpTo(3) frame

n: 3

global frame

number: 3

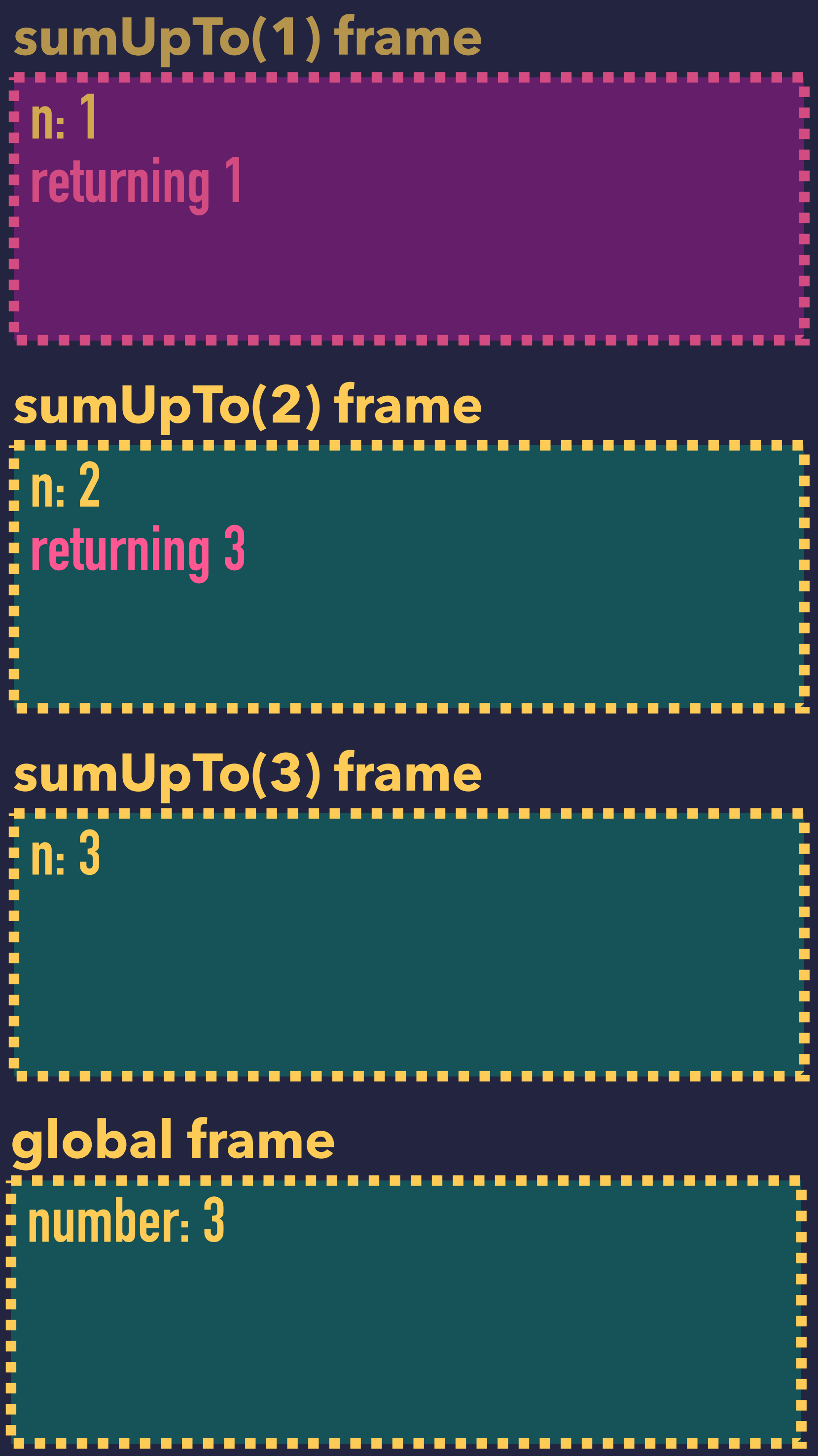
PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(1) frame

n: 1
returning 1

sumUpTo(2) frame

n: 2
returning 3

sumUpTo(3) frame

n: 3

global frame

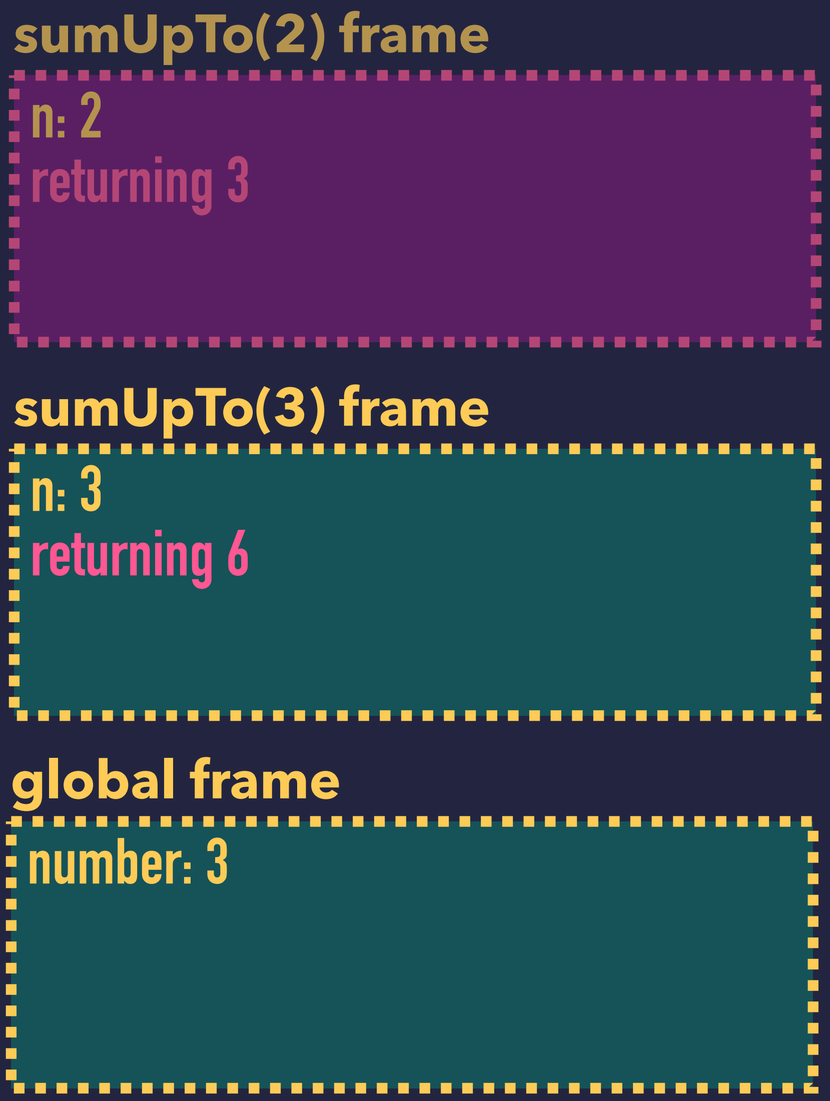
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



sumUpTo(2) frame

n: 2

returning 3

sumUpTo(3) frame

n: 3

returning 6

global frame

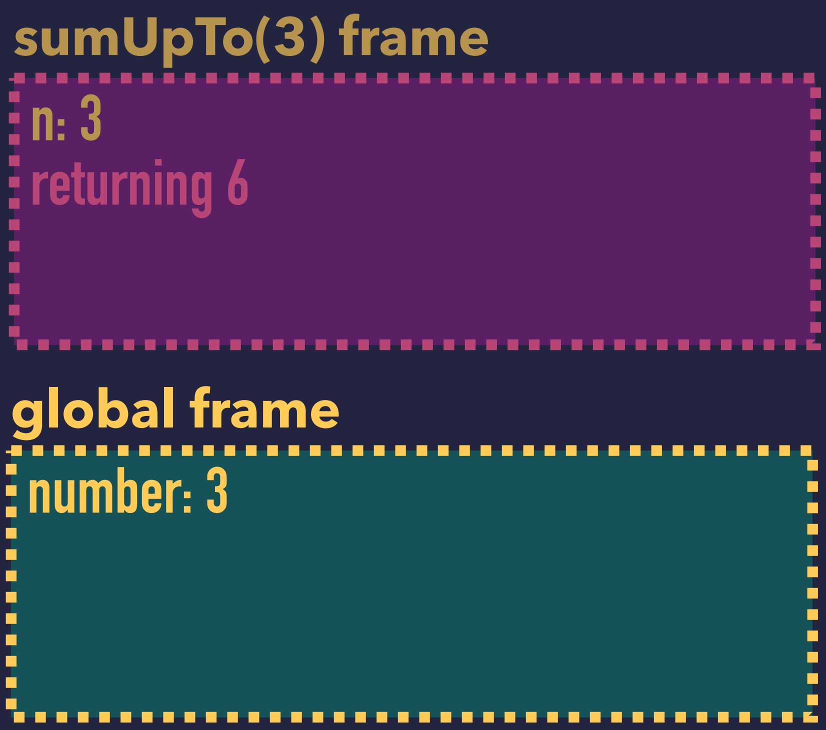
number: 3

PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```



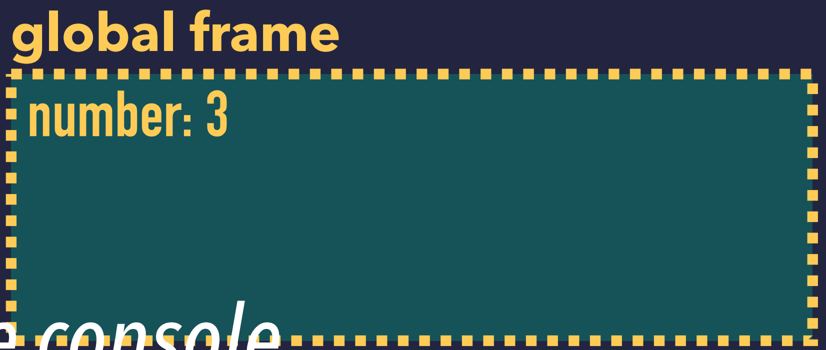
PYTHON'S EXECUTION OF A RECURSIVE FUNCTION

- Let's take a look at Python's execution of this script:

```
def sumUpTo(n):
    if n <= 0:
        return 0
    return sumUpTo(n-1) + n

number = int(input("Number? "))
print(sumUpTo(number))
```

Outputs 6 to the console.



RECURSION EXAMPLE: WORD COLLAPSE

LEC 06-1: RECURSION

JIM FIX, REED COLLEGE CSCI 121

WORD COLLAPSE

Here's a nice puzzle:

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CA~~R~~TS**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT~~S~~**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- CARTS
- CATS
- CAT

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT**
- **~~C~~AT**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT**
- **AT**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT**
- **AT**
- **A~~T~~**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT**
- **AT**
- **A**

WORD COLLAPSE

Here's a nice puzzle:

- Find a word where **you can remove a letter to form another word**.
- And you can keep doing that until **all its letters have been removed**.

EXAMPLE:

- **CARTS**
- **CATS**
- **CAT**
- **AT**
- **A**

- **Checking whether a word collapses** lends itself to **recursive solution**.

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING**: an old word for interleaving

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING**: an old word for interleaving
- **COMPLETING**

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING**: an old word for interleaving
- **COMPLETING**
- **COMPETING**
- **COMPTING**: an old word for counting or calculating

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING**: an old word for interleaving
- **COMPLETING**
- **COMPETING**
- **COMPTING**: an old word for counting or calculating
- **COMPING**
- **COPING**
- **OPING**: an old word for opening

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING**: an old word for interleaving
- **COMPLETING**
- **COMPETING**
- **COMPTING**: an old word for counting or calculating
- **COMPING**
- **COPING**
- **OPING**: an old word for opening
- **PING**
- **PIN**
- **IN**
- **I**

WORD COLLAPSE

- **Checking whether a word collapses** lends itself to **recursive solution**.

WORD COLLAPSE

- **Checking whether a word collapses** lends itself to **recursive solution**.

- **Definition:** A sequence of letters ***word collapses*** whenever…
  - it is a word, and
  - you can remove a letter to obtain something that word collapses.

WORD COLLAPSE

- **Checking whether a word collapses** lends itself to **recursive solution**.

- **Definition:** A sequence of letters ***word collapses*** whenever...
  - it is a word, and
  - you can remove a letter to obtain something that **word collapses**.

WORD COLLAPSE

- **Checking whether a word collapses** lends itself to **recursive solution**.

- **Definition:** A sequence of letters ***word collapses*** whenever…
  - it is a word, and
  - you can remove a letter to obtain something that word collapses.

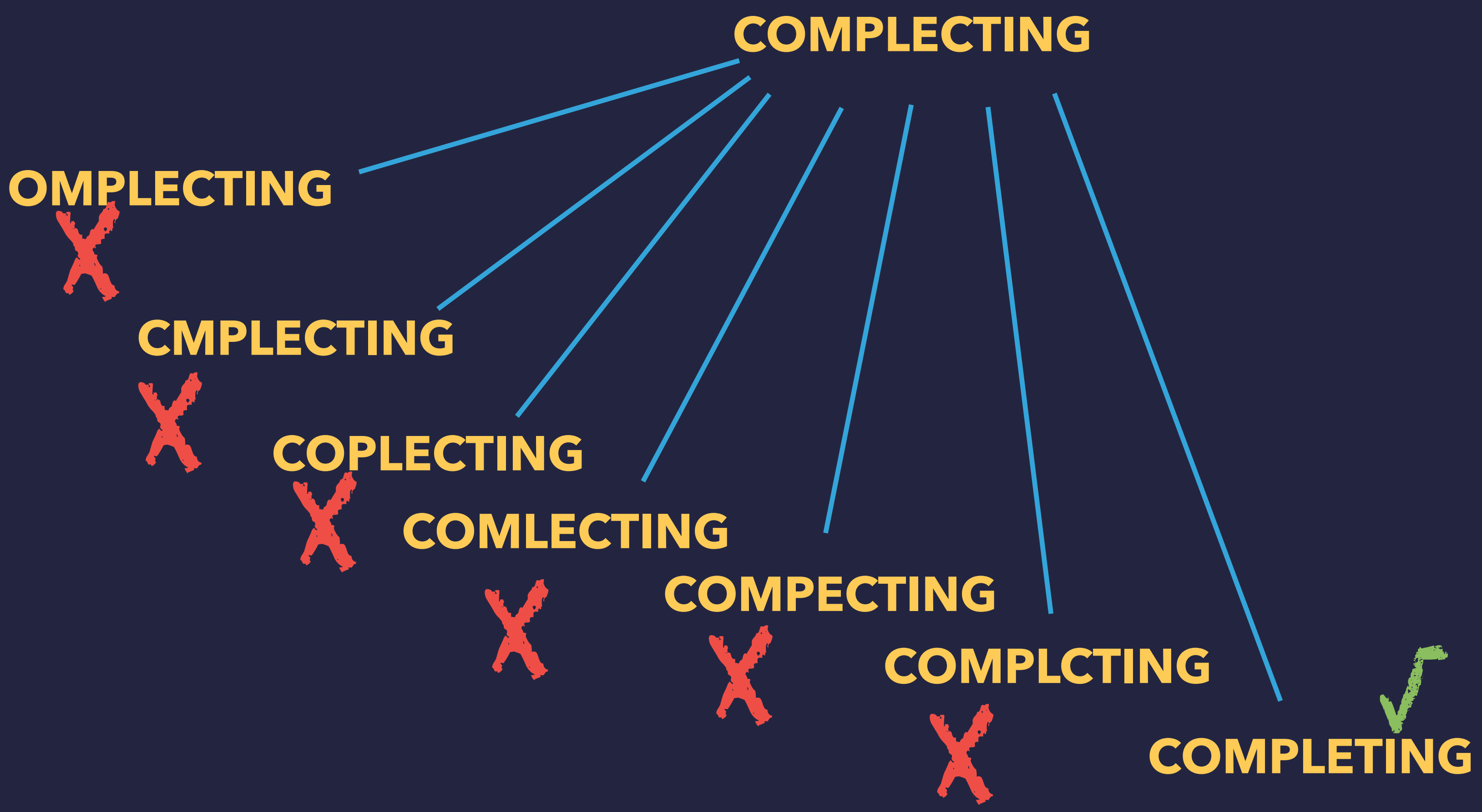
Note: This is on-track for a precise definition, but not quite there yet.

WORD COLLAPSE

ANOTHER EXAMPLE: an eleven letter word

- **COMPLECTING** : an old word for interleaving
- **COMPLETING**
- **COMPETING**
- **COMPTING** : an old word for counting or calculating
- **COMPING**
- **COPING**
- **OPING** : an old word for counting or calculating
- **PING**
- **PIN**
- **IN**
- **I**

FINDING A COLLAPSE USING EXHAUSTIVE SEARCH



COMPLECTING

OMPLECTING ✗

CMPLECTING ✗

COPLECTING ✗

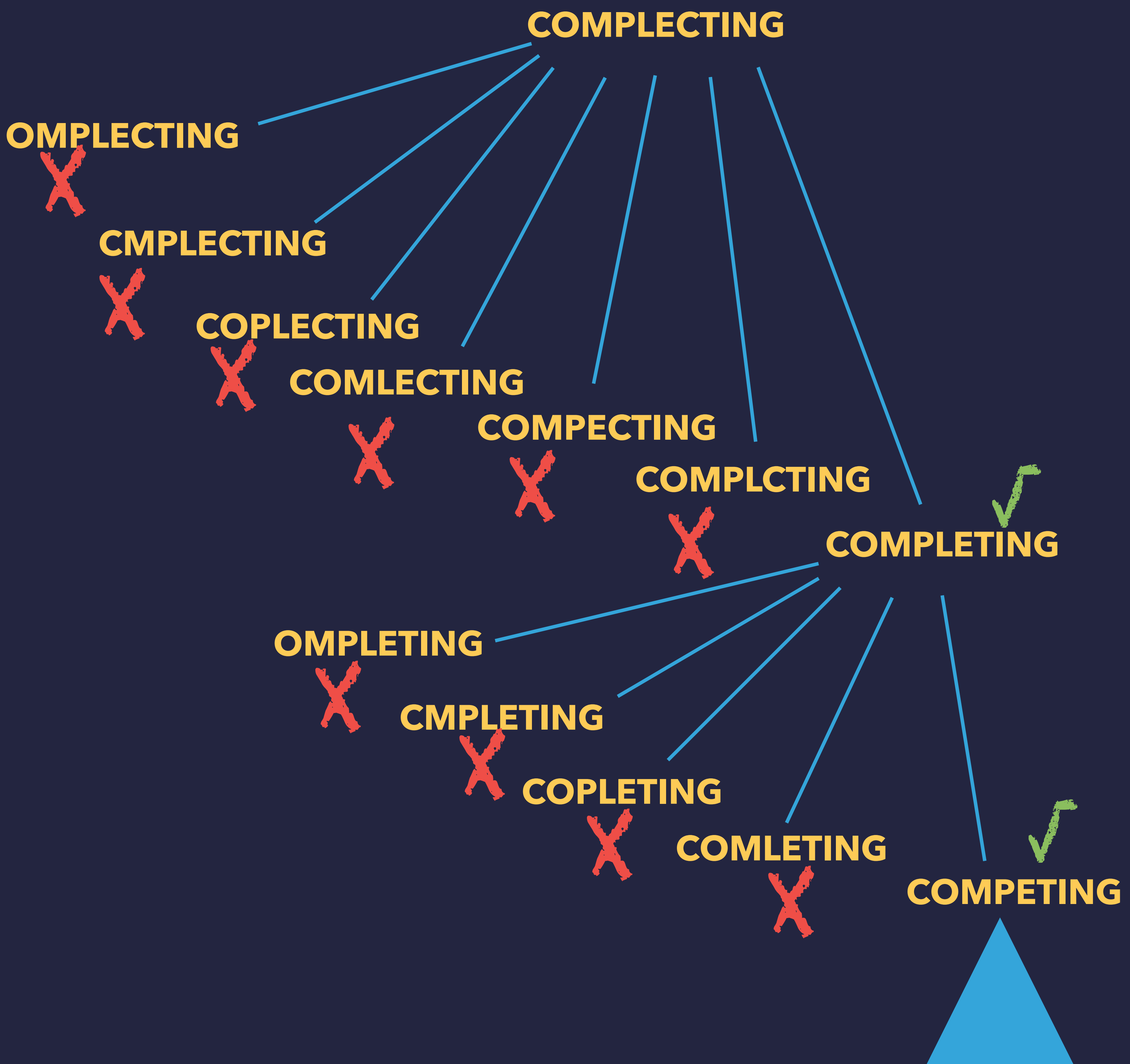
COMLECTING ✗

COMPECTING ✗

COMPLCTING ✗

COMPLETING ✓

FINDING A COLLAPSE USING EXHAUSTIVE SEARCH



COMPLECTING

OMPLECTING ✗

CMPLECTING ✗

COPLECTING ✗

COMLECTING ✗

COMPECTING ✗

COMPLCTING ✗

COMPLETING ✓

OMPLETING ✗

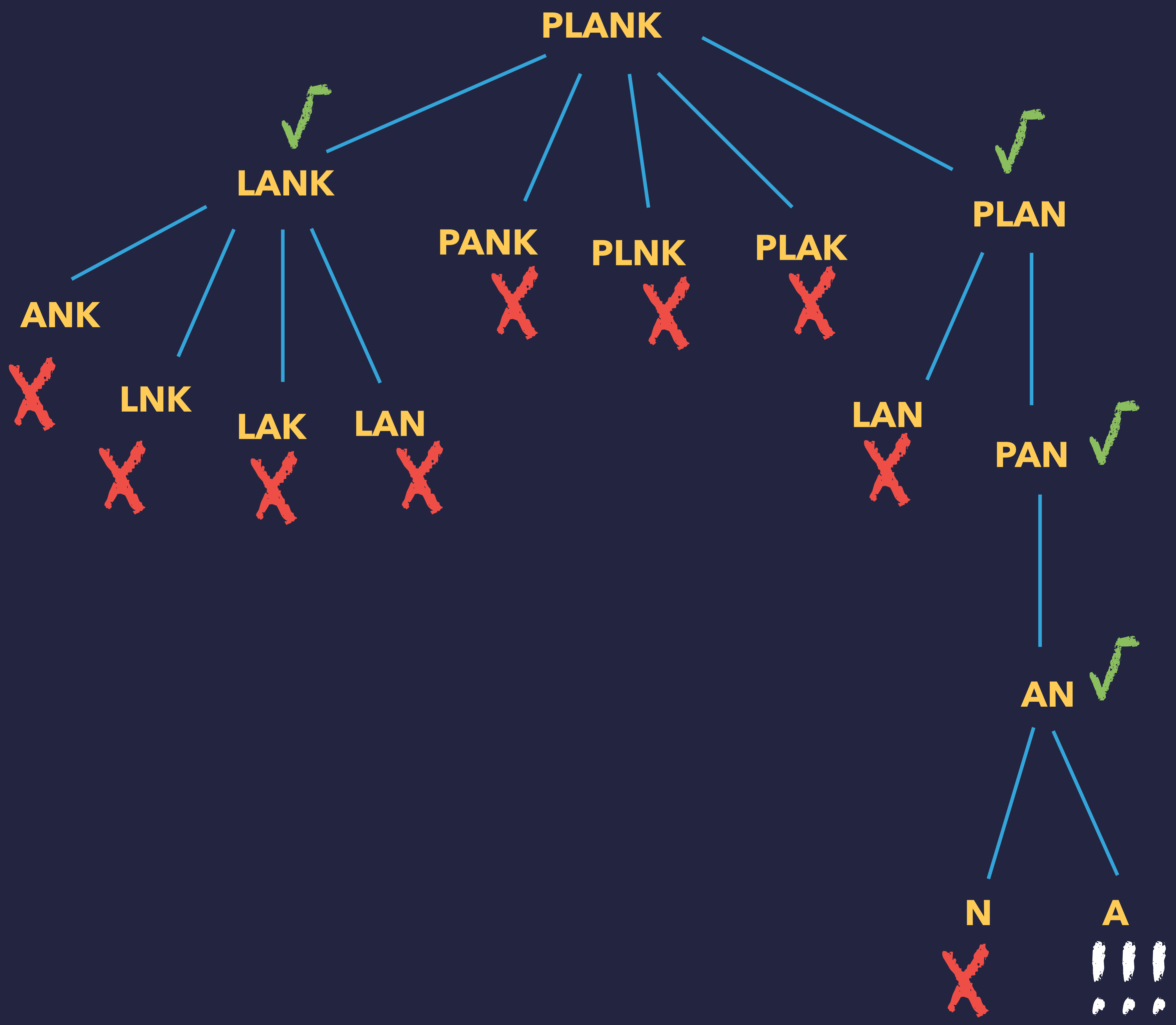
CMPLETING ✗

COPLETING ✗

COMLETING ✗

COMPETING ✓

BACKTRACKING TO DO EXHAUSTIVE SEARCH



WORD COLLAPSE IN CODE

WORD COLLAPSE IN CODE

```
def collapses(s):
    # A sequence of letters s _word collapses_ whenever

    # it is a word, and...
    if is_word(s):

        # removing a letter you obtain t that word collapses.
        i = 0
        while i < len(s):
            t = s[:i]+s[i+1:]
            if collapses(t):
                return True
            i += 1

    return False
```

WORD COLLAPSE IN CODE

```
def collapses(s):
    if len(s) == 0:
        return True

    # A sequence of letters s _word collapses_ whenever

    # it is a word, and...
    elif is_word(s):

        # removing a letter you obtain t that word collapses.
        i = 0
        while i < len(s):
            t = s[:i]+s[i+1:]
            if collapses(t):
                return True
            i += 1

    else:
        return False
```

WORD COLLAPSE IN CODE

```
def word_collapse(s):
    if len(s) == 0:
        return []
    elif is_word(s):
        i = 0
        while i < len(s):
            t = s[:i]+s[i+1:]
            t_collapse = word_collapse(t)
            if t_collapse is not None:
                return [s] + t_collapase
            i += 1
        return None
    else:
        return None
```

SUMMARY

- Functions and procedures can call other functions and procedures.
  - ➡ They can also *call themselves*. This makes them **recursive**.
- Each active function has its local variables stored in its **call frame**.
  - ➡ With recursion, several call frames for the same-named function ***stack*** up.
  - ➡ Each call has a different value for the parameter in each frame.
- Recursive functions are designed to handle two cases:
  - a **recursive case**: this leads the function to call itself
    - ➡ usually a (slightly) simpler case
  - a **base case**: this stops the "unwinding" or "deepening" of the recursive calls
    - ➡ they are (usually) easy cases; immediately return a result
- The tricky part is learning to express algorithms in this way. **Homework 6**.