COMP110

# CL06 – f-Strings, Positional Arguments, Optional Parameters, and an Intro to Recursion

## Reminders

- **Quiz 00:** 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
- **Quiz 01** tomorrow
  - Practice quiz and key available on site

**Want extra support?** We're here and *want* to help!

## 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?

## `factorial` Algorithm

Create a recursive function called `factorial` that will calculate the product of all positive integers less than or equal to an int, `n`. E.g.,

| | |
|---|---|
| `factorial(n=5)` | would return: 5*4*3*2*1 = **120** |
| `factorial(n=2)` | would return: 2*1 = **2** |
| `factorial(n=1)` | would return: 1 = **1** |
| `factorial(n=0)` | would return: **1** |

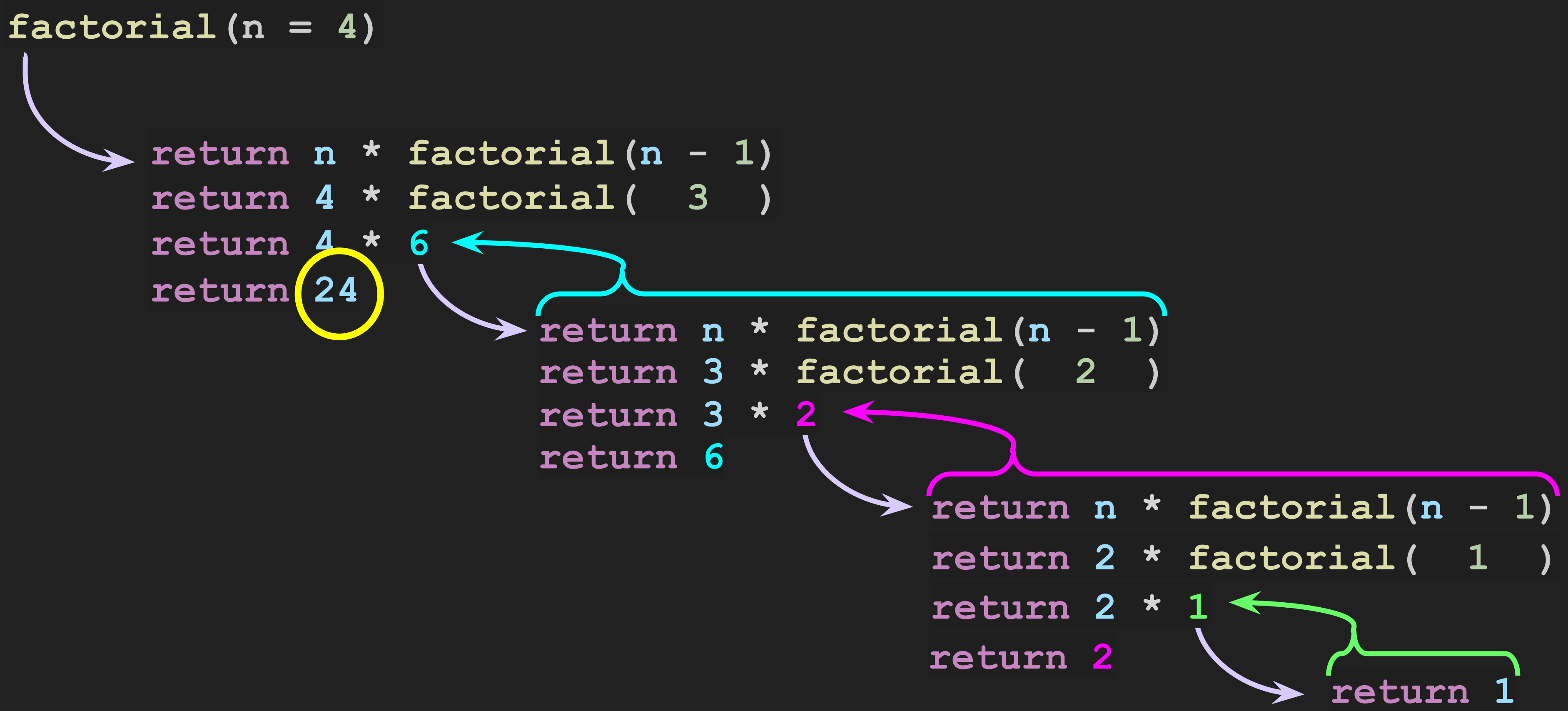
Conceptually, what will our **base case** be?

What will our **recursive case** be?

What is an **edge case** for this function? How could we account for it?

## Visualizing recursive calls to `factorial`

# Visualizing recursive calls to `factorial`

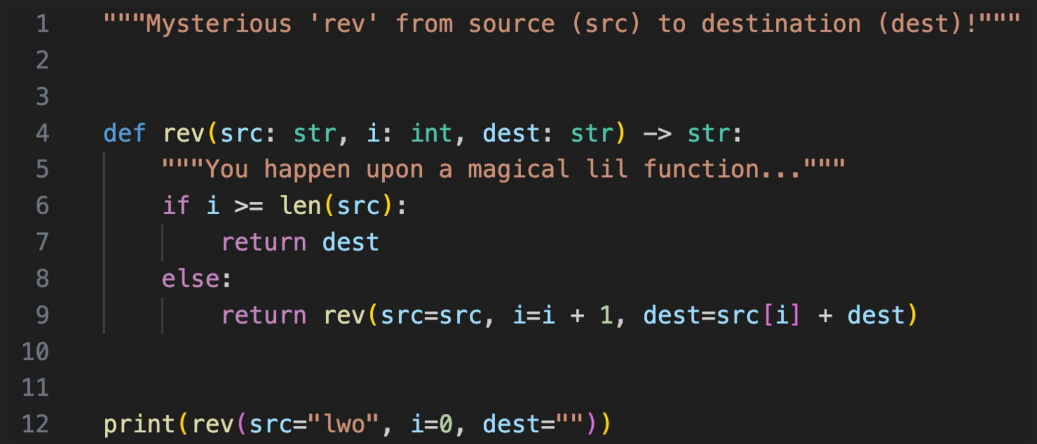


```
factorial(n = 4)

      return n * factorial(n - 1)
      return 4 * factorial(  3  )
      return 4 * 6
      return 24
                return n * factorial(n - 1)
                return 3 * factorial(  2  )
                return 3 * 2
                return 6
                                return n * factorial(n - 1)
                                return 2 * factorial(  1  )
                                return 2 * 1
                                return 2
                                                return 1
```

## Let’s write the **`factorial`** function in VS Code!

## Memory diagram

```python
# Factorial
def factorial(n: int) -> int:
    """Calculates factorial of int n."""
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    else:
        return n * factorial(n - 1)

# Example usage
print(factorial(3))
```



```python
"""Mysterious 'rev' from source (src) to destination (dest)!"""


def rev(src: str, i: int, dest: str) -> str:
    """You happen upon a magical lil function..."""
    if i >= len(src):
        return dest
    else:
        return rev(src=src, i=i + 1, dest=src[i] + dest)


print(rev(src="lwo", i=0, dest=""))
```