COMP110

OOP Practice + a 🪄Magic Method🪄

Let’s use some Point objects to make a Line!

Announcements

RD00 due Wednesday at 11:59pm!

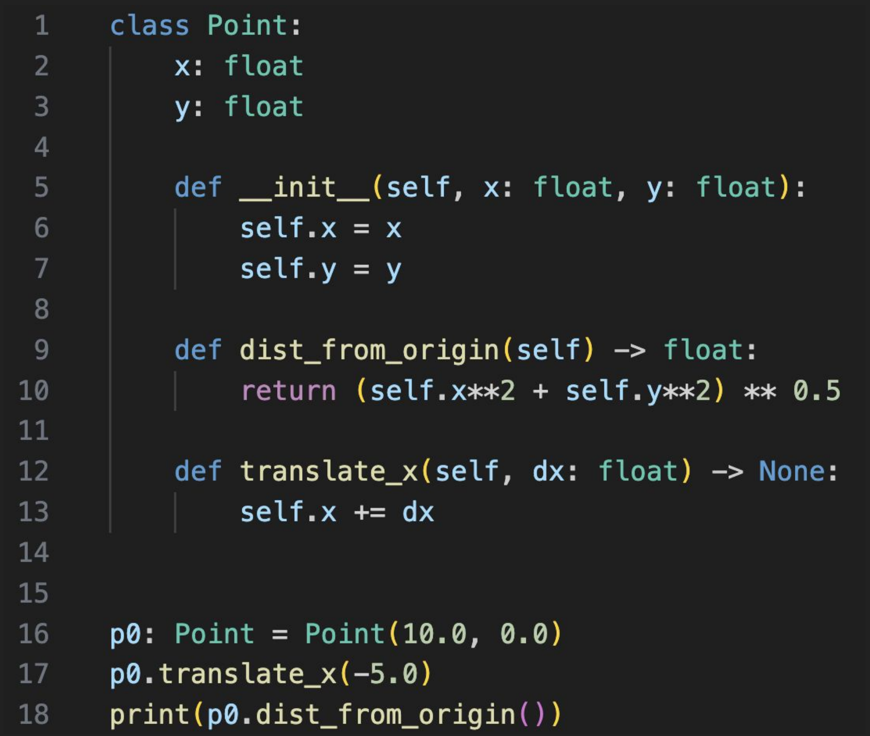
Re: Quiz 03:

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

Re: Quiz 04:

- Practice quiz will be available on the site today
  - Please visit Office Hours if you have questions!

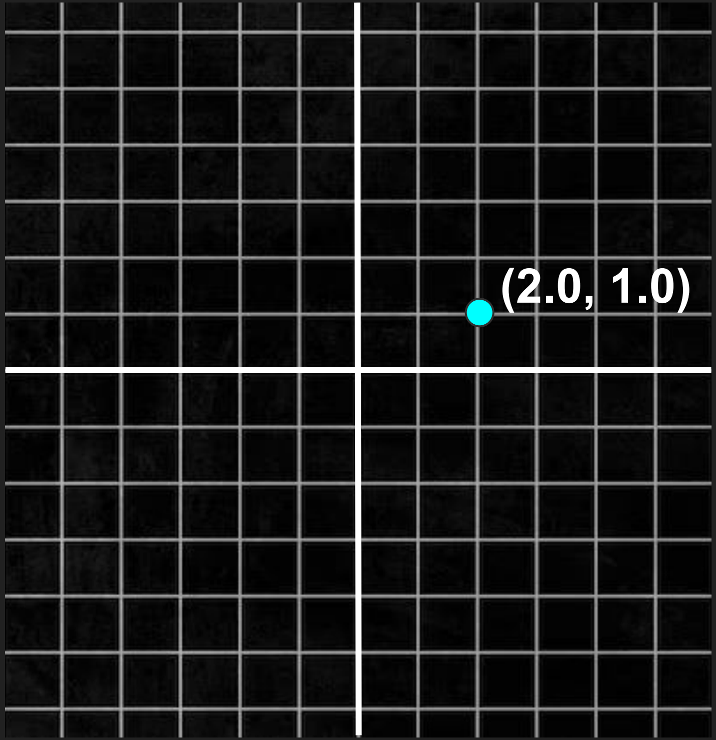
Memory Diagram (to submit to Gradescope!)



```
class Point:
    x: float
    y: float

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def dist_from_origin(self) -> float:
        return (self.x**2 + self.y**2) ** 0.5

    def translate_x(self, dx: float) -> None:
        self.x += dx


p0: Point = Point(10.0, 0.0)
p0.translate_x(-5.0)
print(p0.dist_from_origin())
```

Consider this Point class

```python
class Point:
    x: float
    y: float

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def dist_from_origin(self) -> float:
        return (self.x**2 + self.y**2) ** 0.5

    def translate_x(self, dx: float) -> None:
        self.x += dx

    def translate_y(self, dy: float) -> None:
        self.y += dy

pt: Point = Point(2.0, 1.0)
```



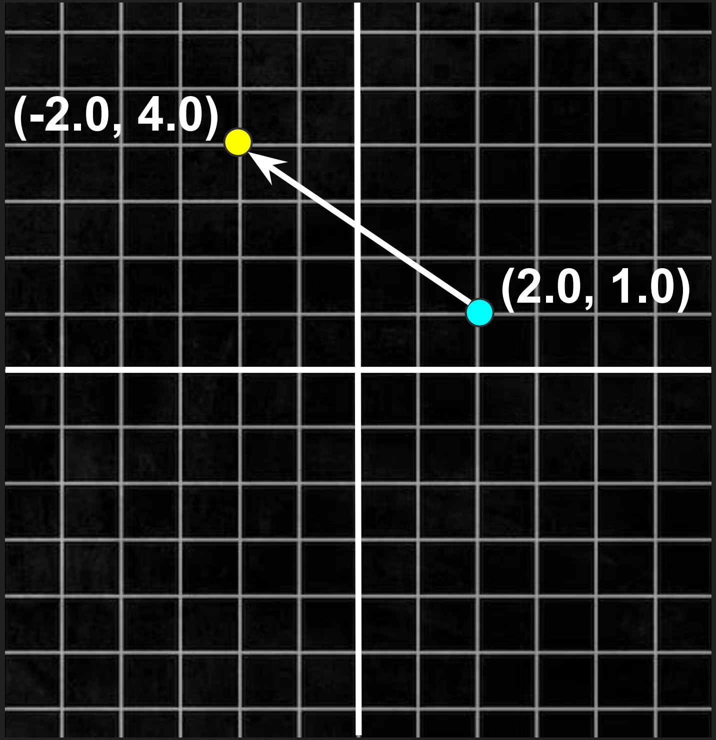
“Two points make a line”

Finding the length of a line:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Finding the slope of a line:

$$m = \frac{\text{Rise}}{\text{Run}} = \frac{y_2 - y_1}{x_2 - x_1}$$



“Two points make a line”

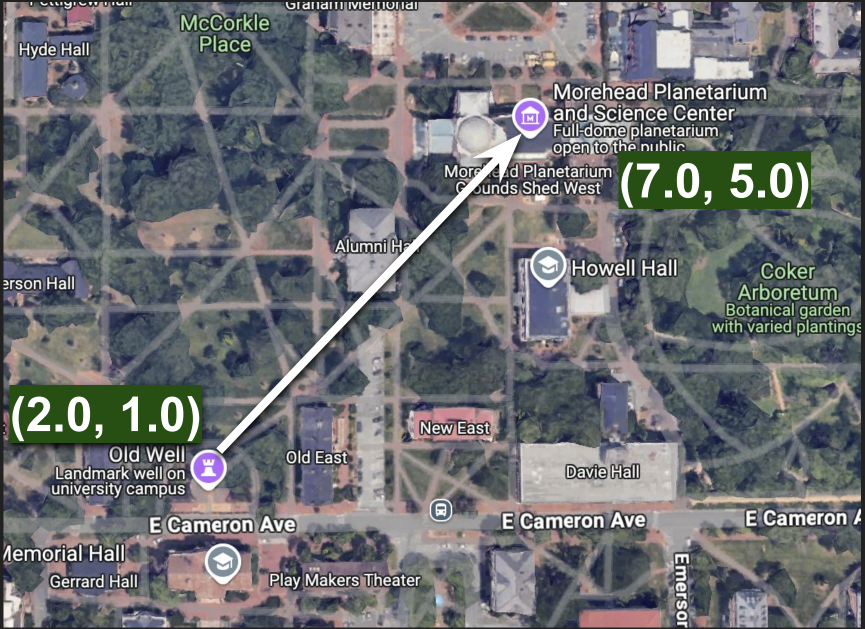
Let’s define a Line class and use it to see the distance from the Old Well to the Planetarium!

Create a **`Line`** class with two attributes: a starting point (`start: Point`) and an ending point (`end: Point`).
The **`Line`** class should have the following method definitions:

-
  ```
  def __init__(self, start: Point, end: Point):
  ```

-
  ```
  def get_length(self) -> float:
  ```
  - Calculates the length of the line

-
  ```
  def get_slope(self) -> float:
  ```
  - Calculates the slope (from `start` to `end`)



“Two points make a line” – Let’s make a Line class!

Finding the length of a line:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Finding the slope of a line:

$$m = \frac{\text{Rise}}{\text{Run}} = \frac{y_2 - y_1}{x_2 - x_1}$$

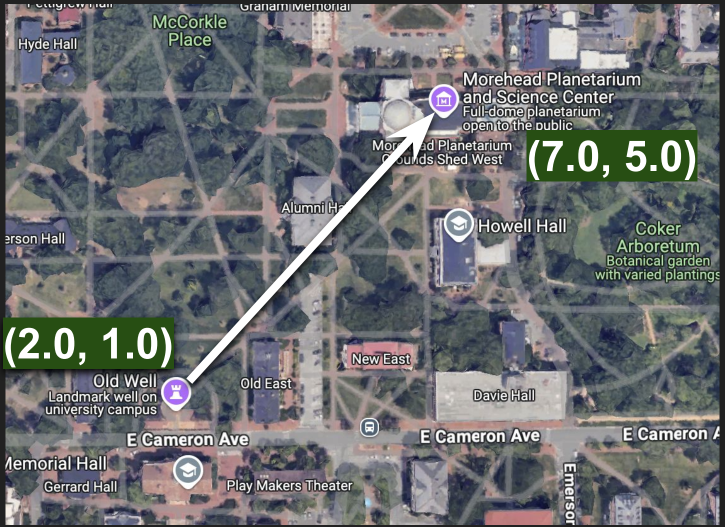
Step 1: Create a `Line` class with two attributes: a starting point (`start: Point`) and an ending point (`end: Point`).

The `Line` class should have the following method definitions:

- **Step 2:** `def __init__(self, start: Point, end: Point):`
- **Step 3:** `def get_length(self) -> float:` calculates the length of the line
- **Step 4:** `def get_slope(self) -> float:` calculates the slope (from `start` to `end`)

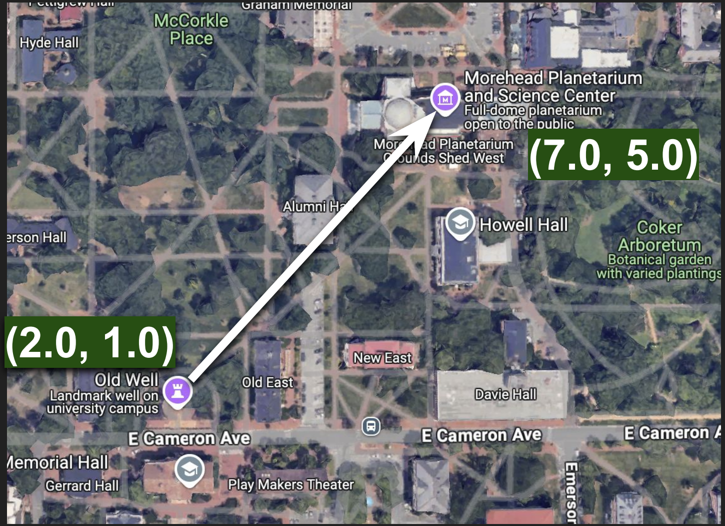
Let’s go over it together! →

```python
class Point: … # collapsed for space

class Line:
    start: Point
    end: Point

    def __init__(self, start: Point, end: Point):
        self.start = start
        self.end = end

    def get_length(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return (x_diffs**2 + y_diffs**2) ** 0.5

    def get_slope(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return y_diffs / x_diffs
```



Create a Line object and find the distance from the Old Well to the Planetarium:

```python
class Point: … # collapsed for space

class Line:
    start: Point
    end: Point

    def __init__(self, start: Point, end: Point):
        self.start = start
        self.end = end

    def get_length(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return (x_diffs**2 + y_diffs**2) ** 0.5

    def get_slope(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return y_diffs / x_diffs
```



Create a Line object and find the distance from the Old Well to the Planetarium:

```python
me: Point = Point(2.0, 1.0)
planet_y: float = 5.0
planet_loc: Point = Point(me.x + 5, planet_y) # Example of accessing an attribute’s value
path: Line = Line(me, planet_loc)
print(path.get_slope())
```

On your own: try printing a Point or Line object!

```
class Point:
    x: float
    y: float

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def dist_from_origin(self) -> float:
        return (self.x**2 + self.y**2) ** 0.5

    def translate_x(self, dx: float) -> None:
        self.x += dx

    def translate_y(self, dy: float) -> None:
        self.y += dy

pt: Point = Point(2.0, 1.0)
```

What happens?

On your own: try printing a Point or Line object!

```python
class Point:
    x: float
    y: float

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def dist_from_origin(self) -> float:
        return (self.x**2 + self.y**2) ** 0.5

    def translate_x(self, dx: float) -> None:
        self.x += dx

    def translate_y(self, dy: float) -> None:
        self.y += dy

pt: Point = Point(2.0, 1.0)
```

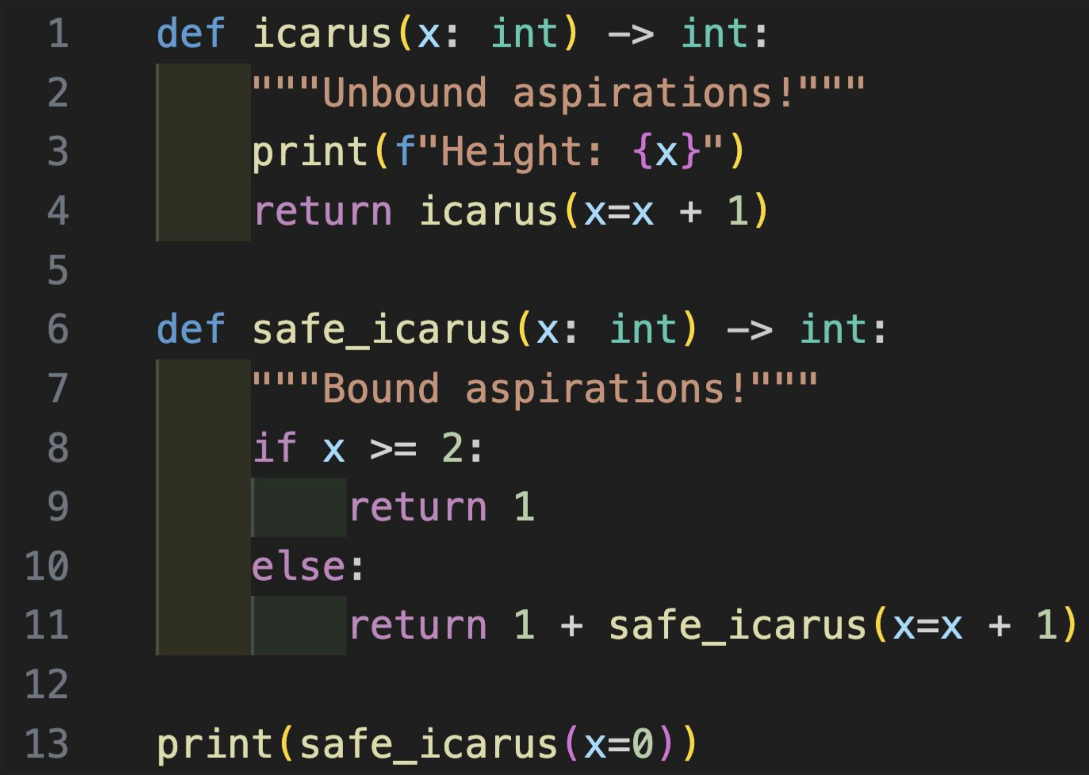
What happens?

```
<__main__.Point object at 0xffff9506d9a0>
```

Let’s implement some magic in VS Code!

Shifting gears… remember recursion?

Recall these functions: what was the issue with the icarus function?



```python
def icarus(x: int) -> int:
    """Unbound aspirations!"""
    print(f"Height: {x}")
    return icarus(x=x + 1)

def safe_icarus(x: int) -> int:
    """Bound aspirations!"""
    if x >= 2:
        return 1
    else:
        return 1 + safe_icarus(x=x + 1)

print(safe_icarus(x=0))
```

The dreaded `Recursion Error`!

Stack Overflow and Recursion Errors

When a programmer writes a function that calls itself indefinitely (*infinitely*), the **function call stack** will *overflow…*

This leads to a **`Stack Overflow`** or **`Recursion Error`**:

```
RecursionError: maximum recursion depth exceeded while
calling a Python object
```

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?

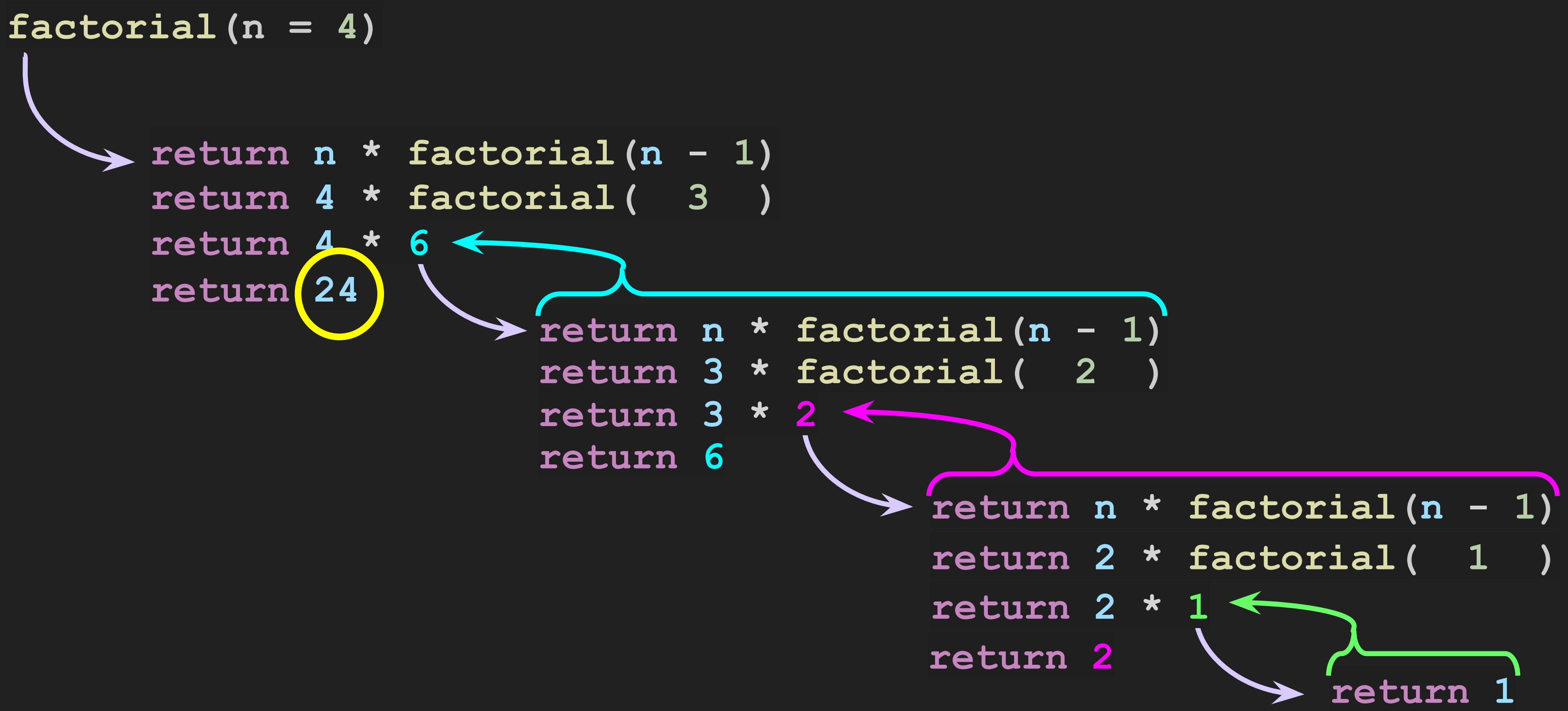
Another example of recursion: factorial!

To calculate the factorial of an int, n, we would multiply n by (n-1), then (n-2), and so on, until we reach 1.

For instance, to calculate 5!, we would do: 5 * 4 * 3 * 2 * 1, which would evaluate to 120.

```python
def factorial(n: int) -> int:
    # Base case: factorial of 0 or 1 is 1
    if n <= 1:
        return 1
    # Recursive case: n! = n × (n-1)!
    return n * factorial(n - 1)
```

Visualizing recursive calls to `factorial`

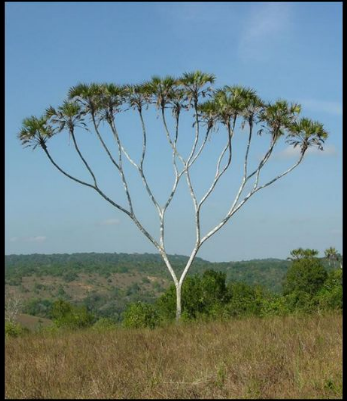


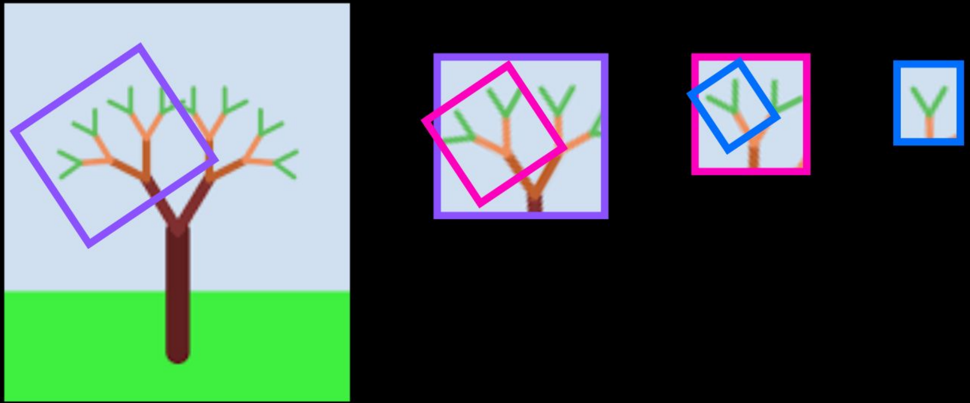
```
factorial(n = 4)

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
```

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**





Want extra practice? Try diagramming this!

```python
class Point: … # collapsed for space

class Line:
    start: Point
    end: Point

    def __init__(self, start: Point, end: Point):
        self.start = start
        self.end = end

    def get_length(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return (x_diffs**2 + y_diffs**2) ** 0.5

    def get_slope(self) -> float:
        x_diffs: float = self.end.x - self.start.x
        y_diffs: float = self.end.y - self.start.y
        return y_diffs / x_diffs

me: Point = Point(2.0, 1.0)
planet_y: float = 5.0
planet_loc: Point = Point(me.x + 5, planet_y)
path: Line = Line(me, planet_loc)
print(path.get_slope())
```