Object-oriented Practice

COMP110 - CL22

2024/04/25

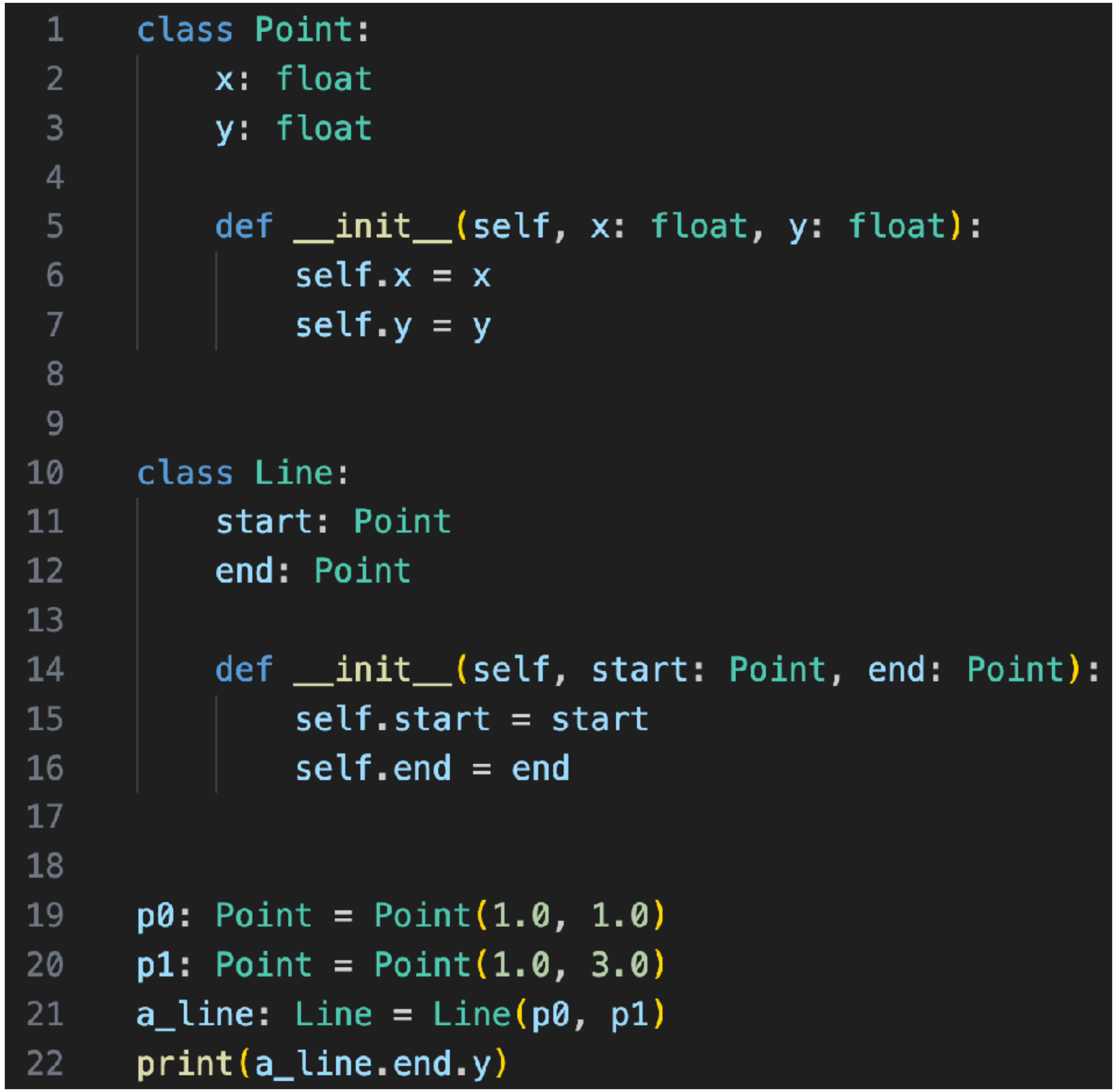
Closing Out the Semester

- EX07 - Compstagram - Build photo filters with classes, objects, and algorithms
  - Due LDOC

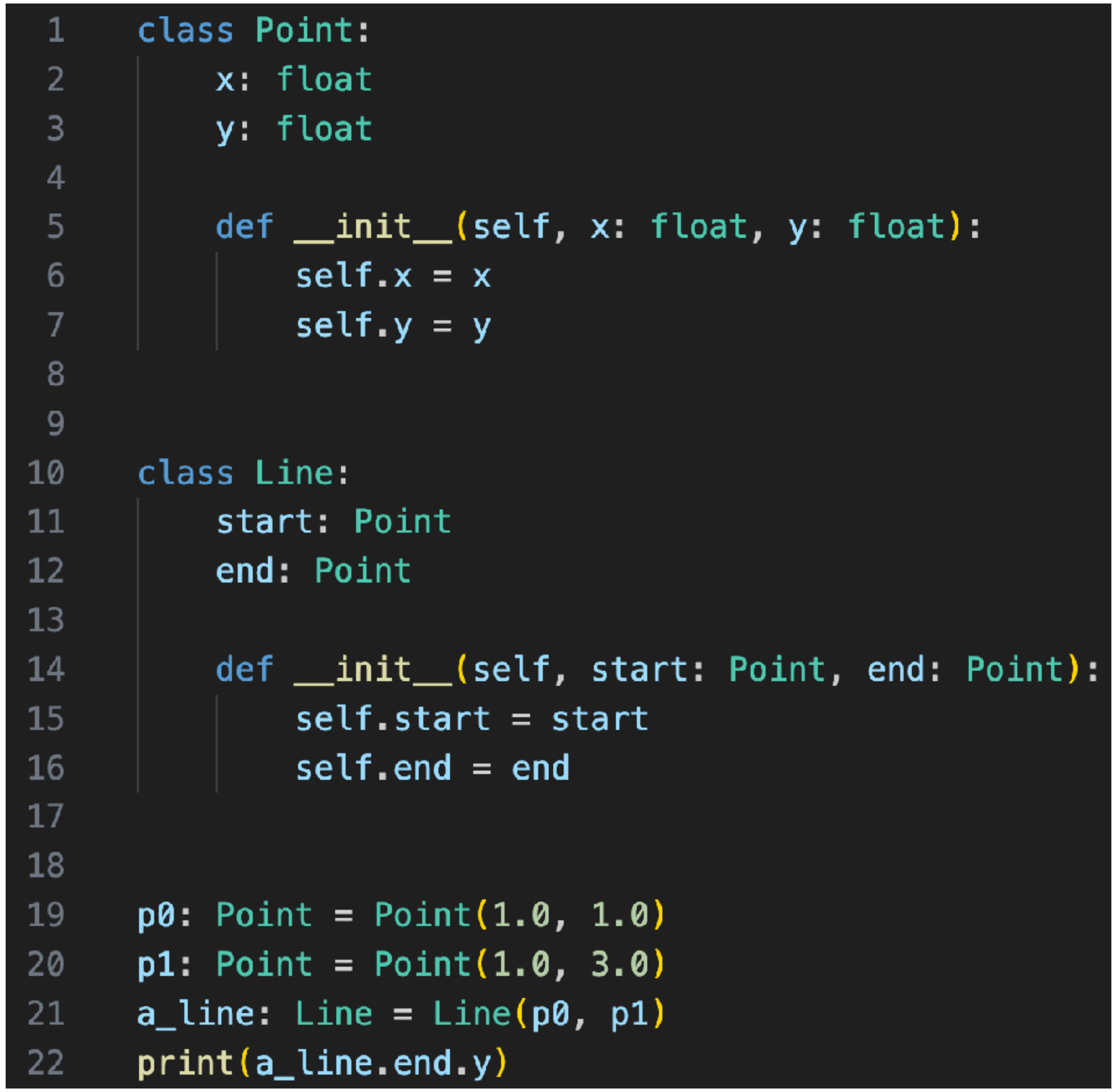
- Final Exam
  - Friday, May 3rd at 8am
  - Conflict? Makeup: Saturday, May 4th at 12pm in SN014

Warm-up: Trace a Memory Diagram

Submit responses to Gradescope

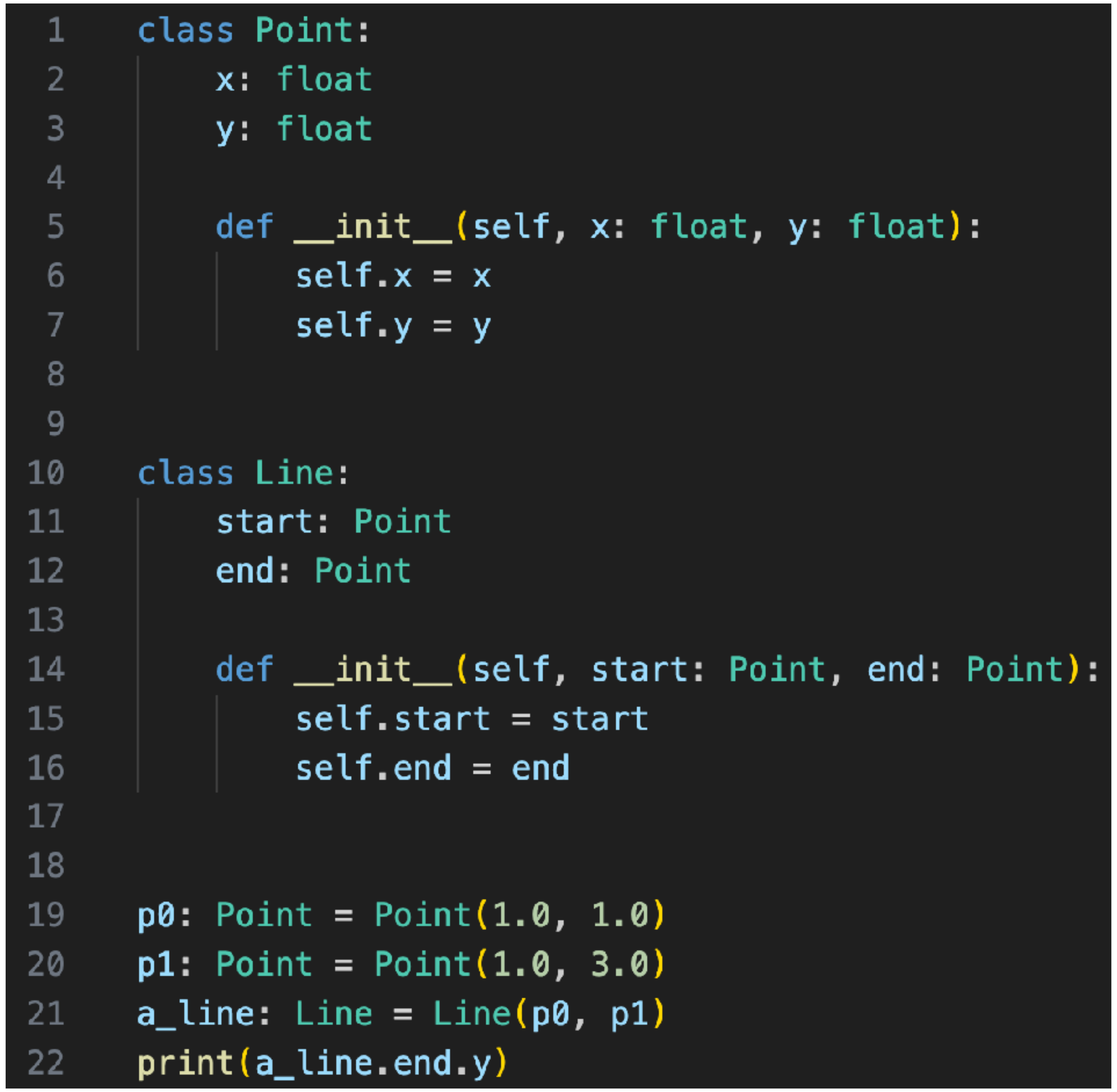


```python
class Point:
    x: float
    y: float

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


class Line:
    start: Point
    end: Point

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


p0: Point = Point(1.0, 1.0)
p1: Point = Point(1.0, 3.0)
a_line: Line = Line(p0, p1)
print(a_line.end.y)
```

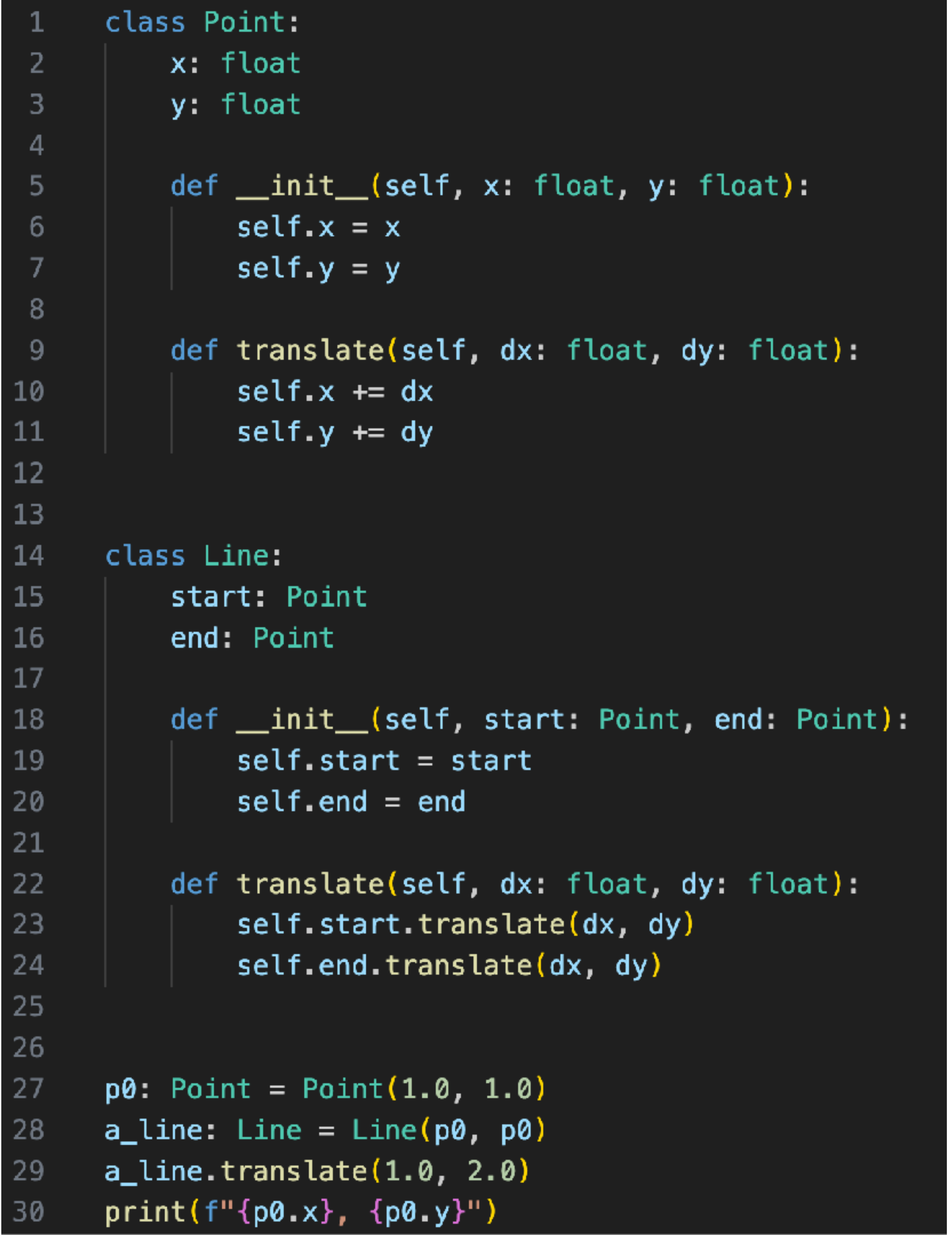


```python
class Point:
    x: float
    y: float

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


class Line:
    start: Point
    end: Point

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


p0: Point = Point(1.0, 1.0)
p1: Point = Point(1.0, 3.0)
a_line: Line = Line(p0, p1)
print(a_line.end.y)
```

Abbreviating __init__ Frames



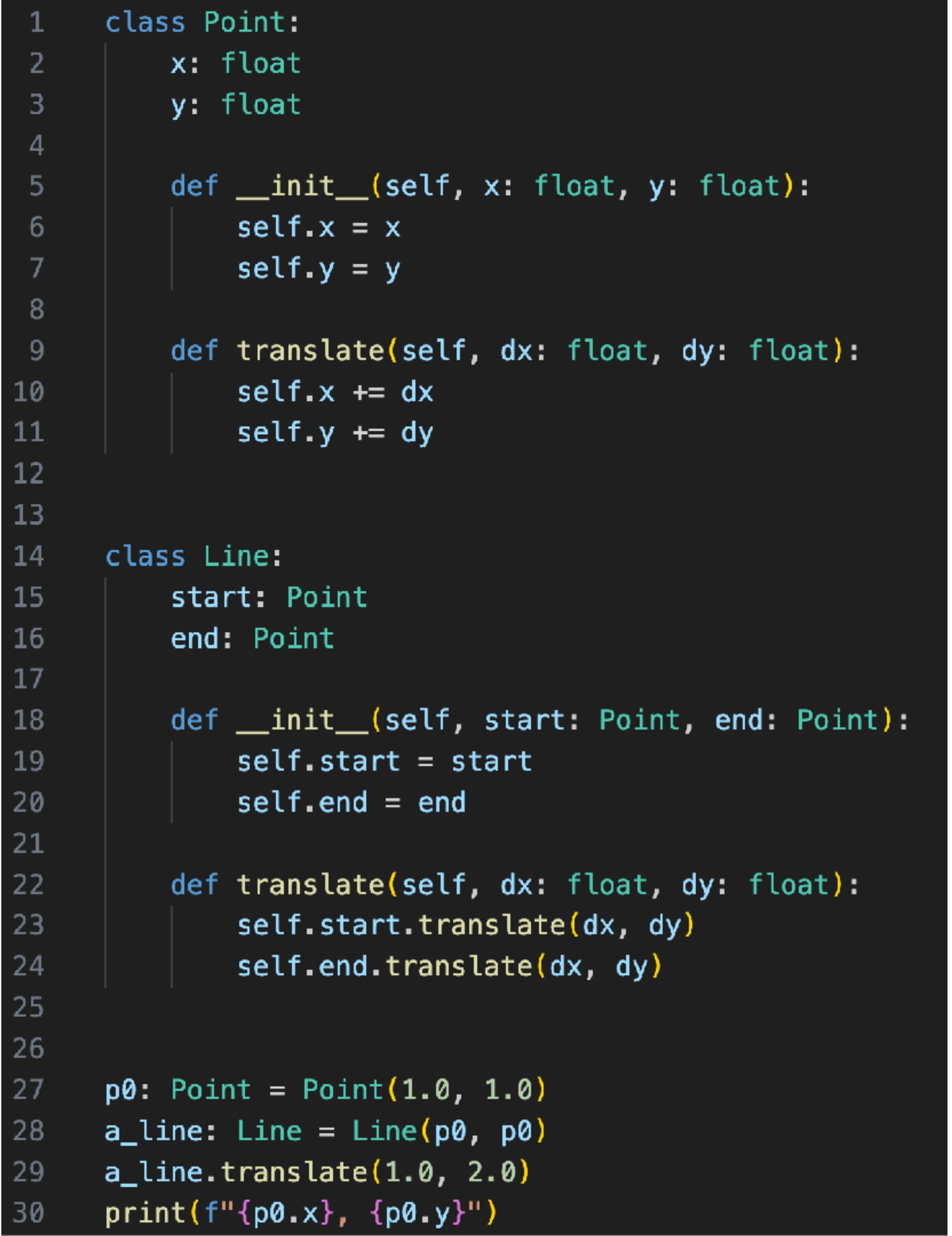
```python
class Point:
    x: float
    y: float

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


class Line:
    start: Point
    end: Point

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


p0: Point = Point(1.0, 1.0)
p1: Point = Point(1.0, 3.0)
a_line: Line = Line(p0, p1)
print(a_line.end.y)
```



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

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

    def translate(self, dx: float, dy: float):
        self.x += dx
        self.y += dy


class Line:
    start: Point
    end: Point

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

    def translate(self, dx: float, dy: float):
        self.start.translate(dx, dy)
        self.end.translate(dx, dy)


p0: Point = Point(1.0, 1.0)
a_line: Line = Line(p0, p0)
a_line.translate(1.0, 2.0)
print(f"{p0.x}, {p0.y}")
```

Trace another Memory Diagram

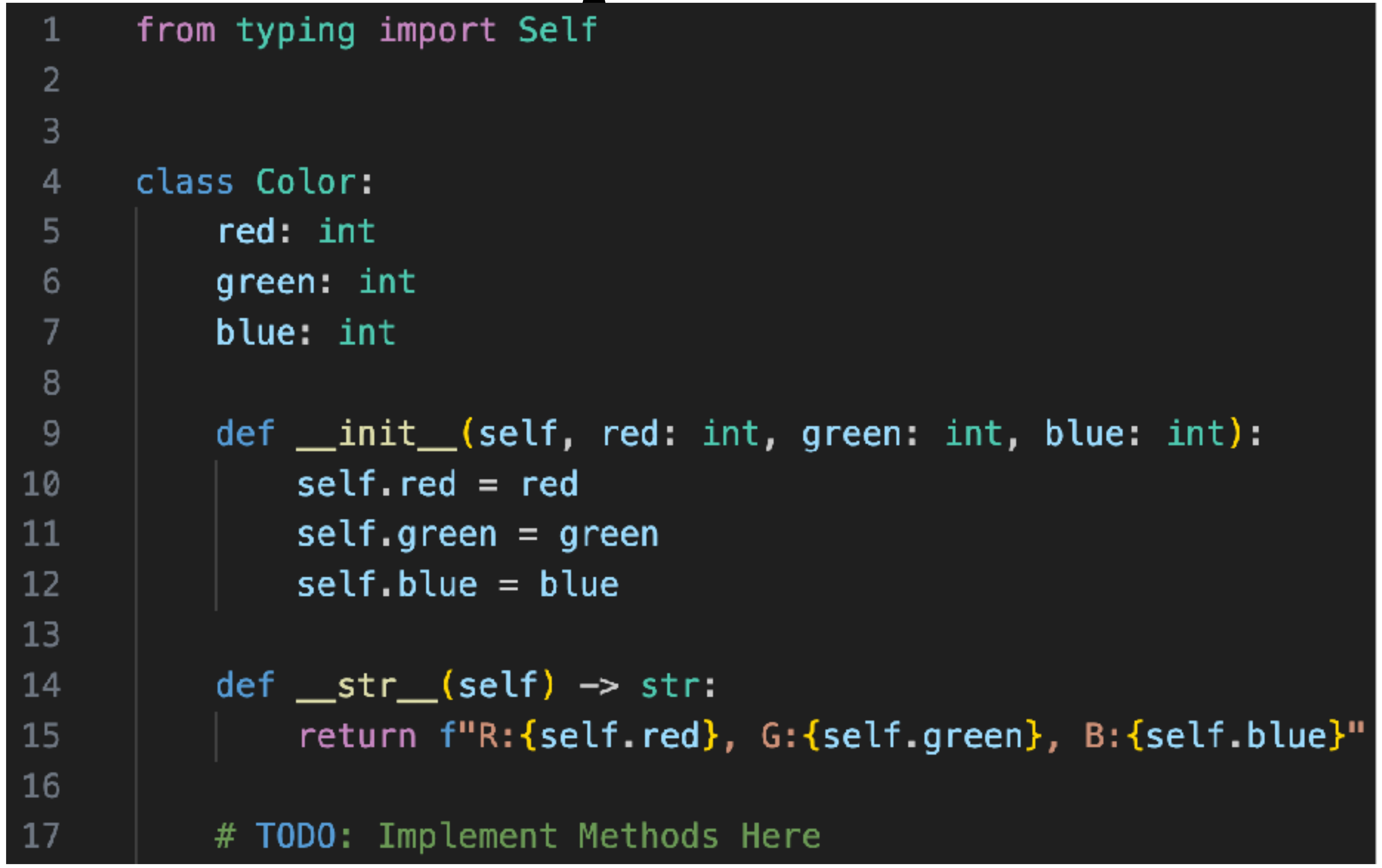
__init__ frames should be abbreviated



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

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

    def translate(self, dx: float, dy: float):
        self.x += dx
        self.y += dy


class Line:
    start: Point
    end: Point

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

    def translate(self, dx: float, dy: float):
        self.start.translate(dx, dy)
        self.end.translate(dx, dy)


p0: Point = Point(1.0, 1.0)
a_line: Line = Line(p0, p0)
a_line.translate(1.0, 2.0)
print(f"{p0.x}, {p0.y}")
```

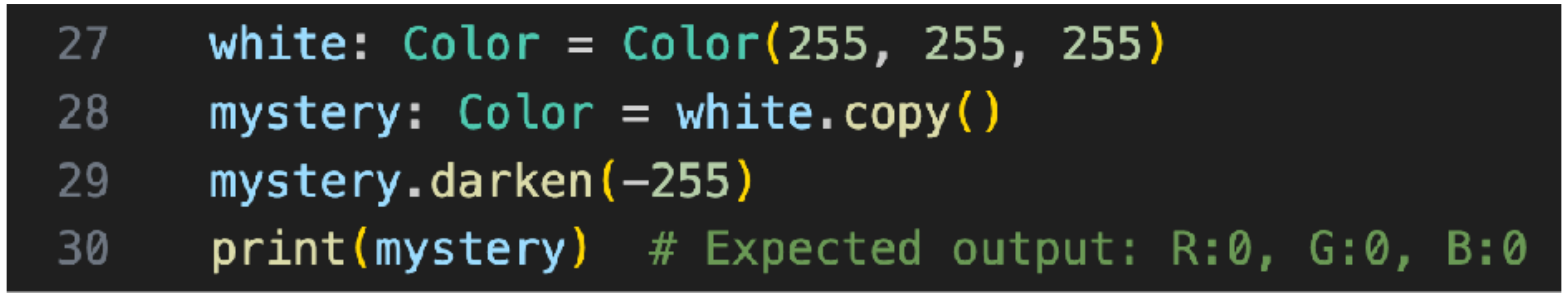
Given the following class and usage (below) implement methods *copy* and *darken*.

Starter



```python
from typing import Self


class Color:
    red: int
    green: int
    blue: int

    def __init__(self, red: int, green: int, blue: int):
        self.red = red
        self.green = green
        self.blue = blue

    def __str__(self) -> str:
        return f"R:{self.red}, G:{self.green}, B:{self.blue}"

    # TODO: Implement Methods Here
```

Usage

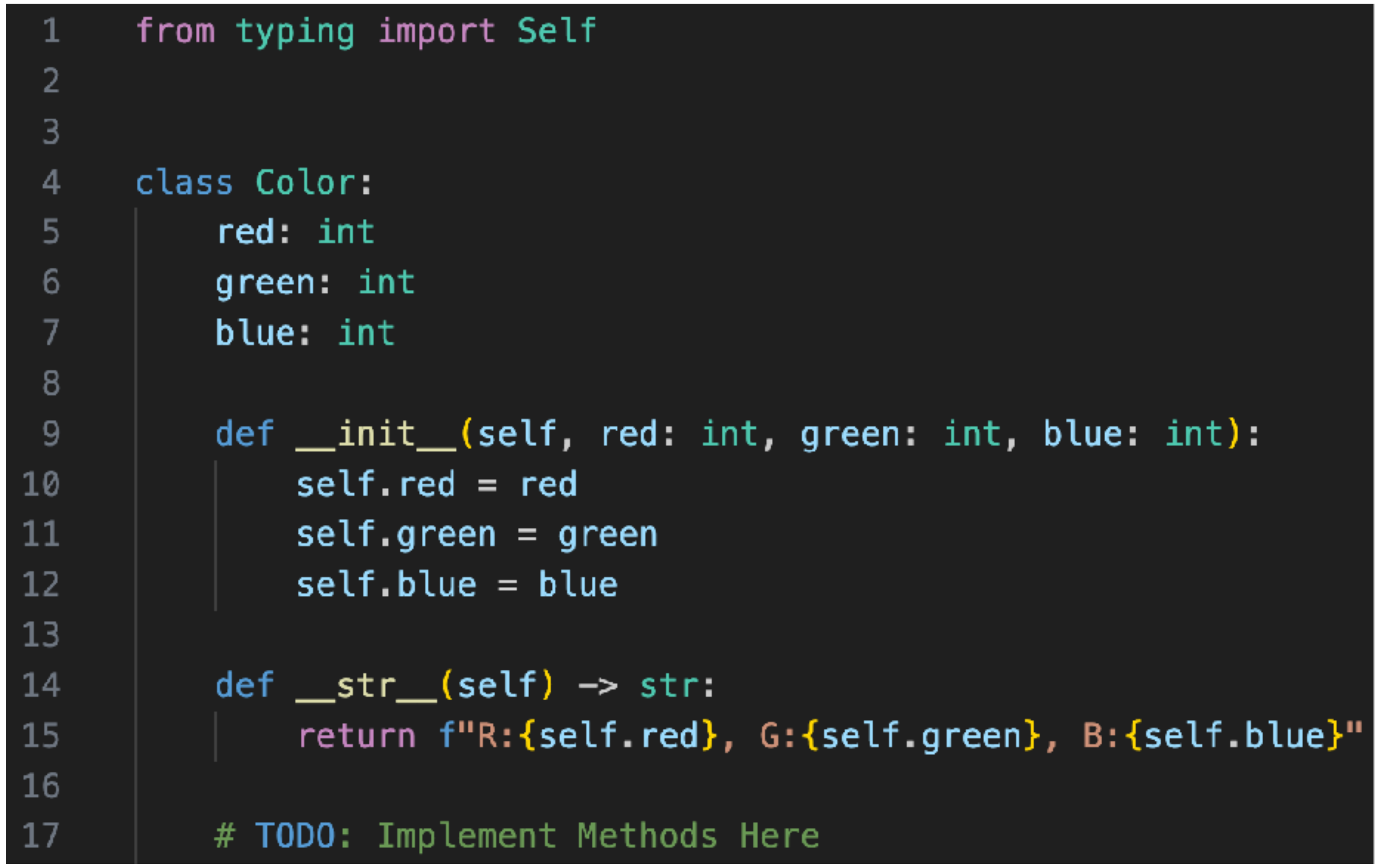


```python
white: Color = Color(255, 255, 255)
mystery: Color = white.copy()
mystery.darken(-255)
print(mystery)  # Expected output: R:0, G:0, B:0
```

The *copy* method should return a new Color object with the same red, green, and blue attribute values as the Color object it is called on.

The *darken* method should mutate the object it is called on by decreasing each of its attributes by the int argument it is called with.

Starter

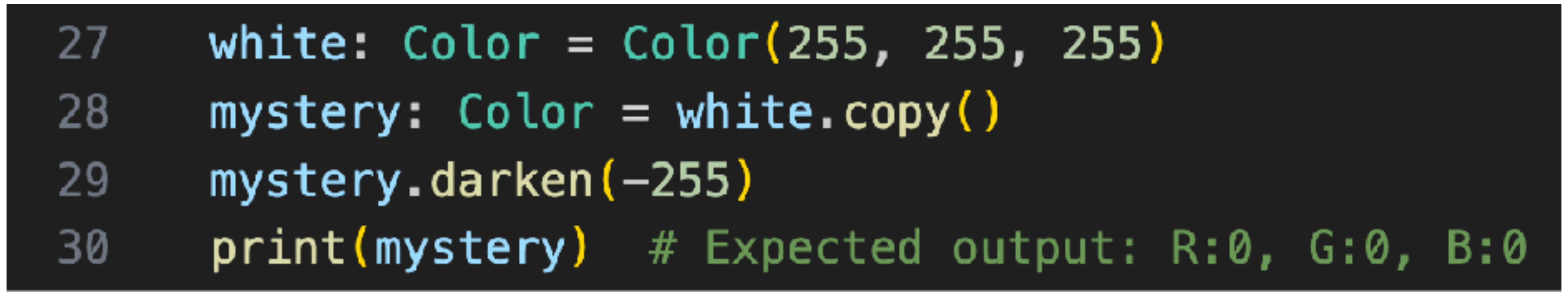


```python
from typing import Self


class Color:
    red: int
    green: int
    blue: int

    def __init__(self, red: int, green: int, blue: int):
        self.red = red
        self.green = green
        self.blue = blue

    def __str__(self) -> str:
        return f"R:{self.red}, G:{self.green}, B:{self.blue}"

    # TODO: Implement Methods Here
```

The *copy* method should return a new Color object with the same red, green, and blue attribute values as the Color object it is called on.

The *darken* method should mutate the object it is called on by decreasing each of its attributes by the int argument it is called with.

Usage

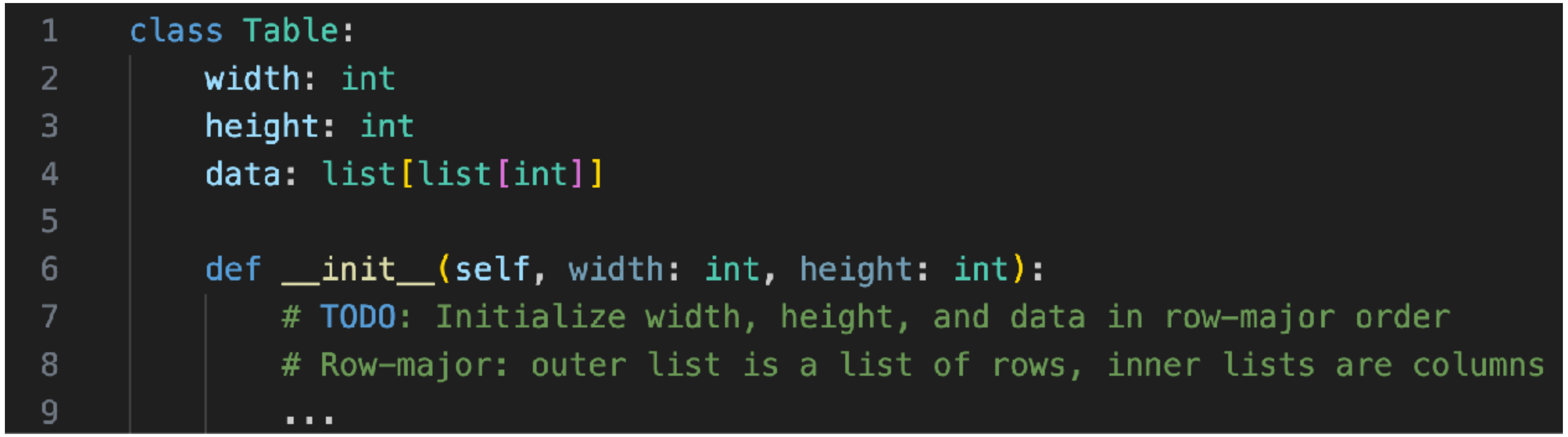


```python
white: Color = Color(255, 255, 255)
mystery: Color = white.copy()
mystery.darken(-255)
print(mystery)  # Expected output: R:0, G:0, B:0
```

Representing 2D Grids with Lists of Lists

Row-major vs. Column-major

Given the following class complete the implementation of the __init__ method.

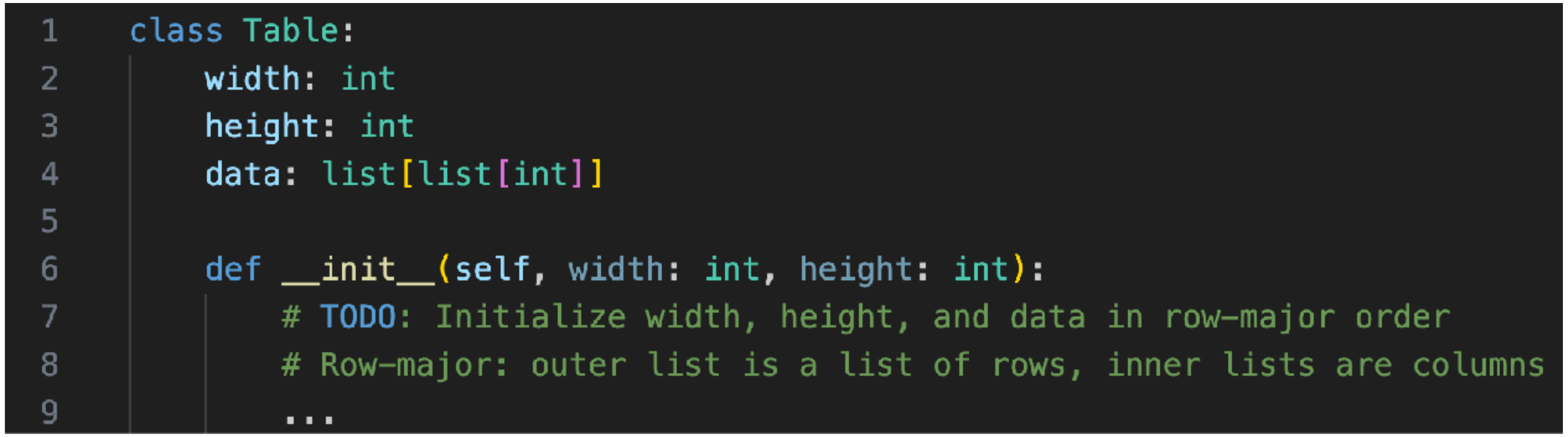


```python
class Table:
    width: int
    height: int
    data: list[list[int]]

    def __init__(self, width: int, height: int):
        # TODO: Initialize width, height, and data in row-major order
        # Row-major: outer list is a list of rows, inner lists are columns
        ...
```

The initialization algorithm of data is:

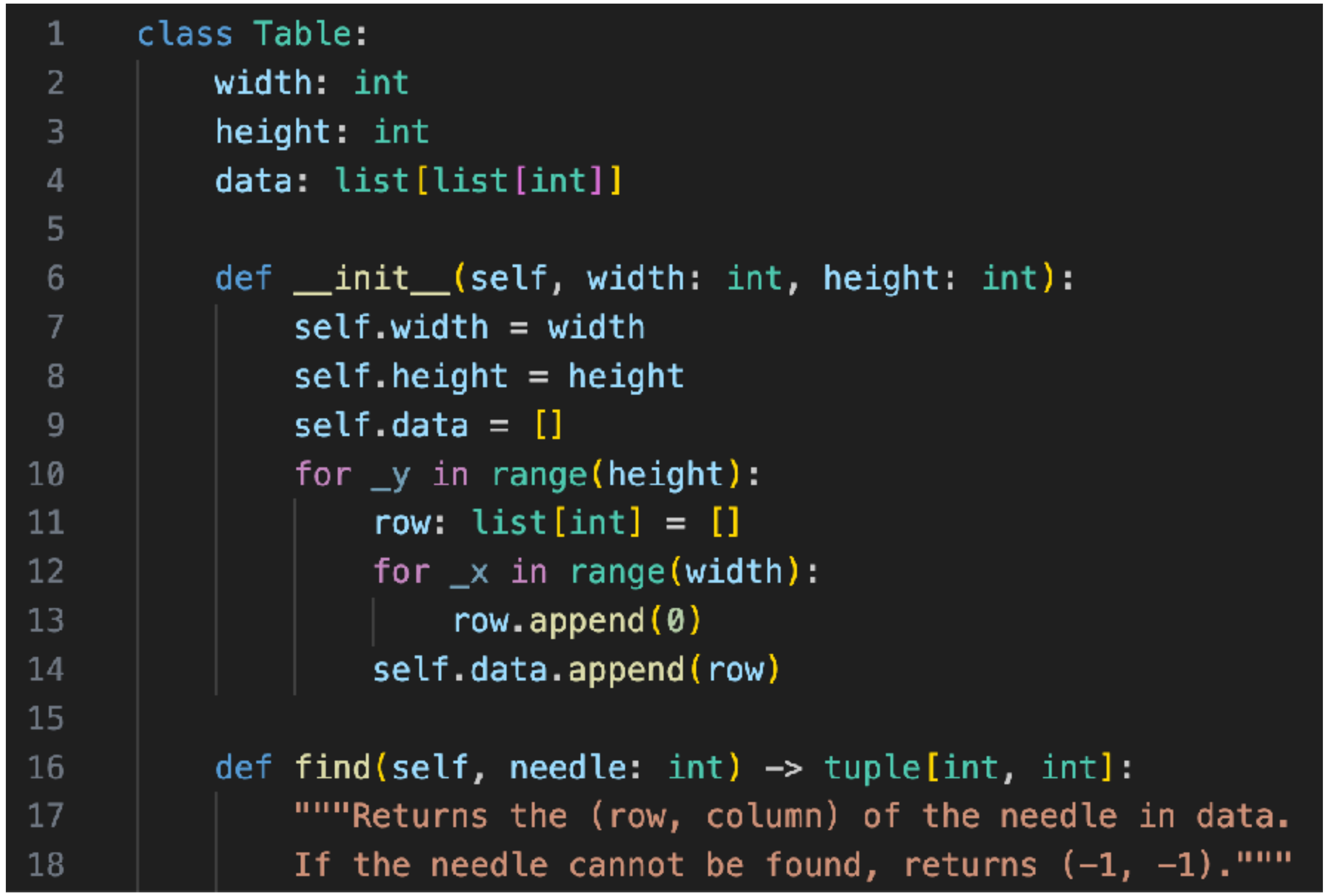
1. Assign data attribute of self a new, empty list
2. Loop through each "row" from 0 to height
   3. In the row-loop, establish a new empty list
   4. Loop through each "column" from 0 to width
      5. In the column-loop, append 0 to the row list
   6. Append the row to the data attribute of the object

Given the following class complete the implementation of the __init__ method.



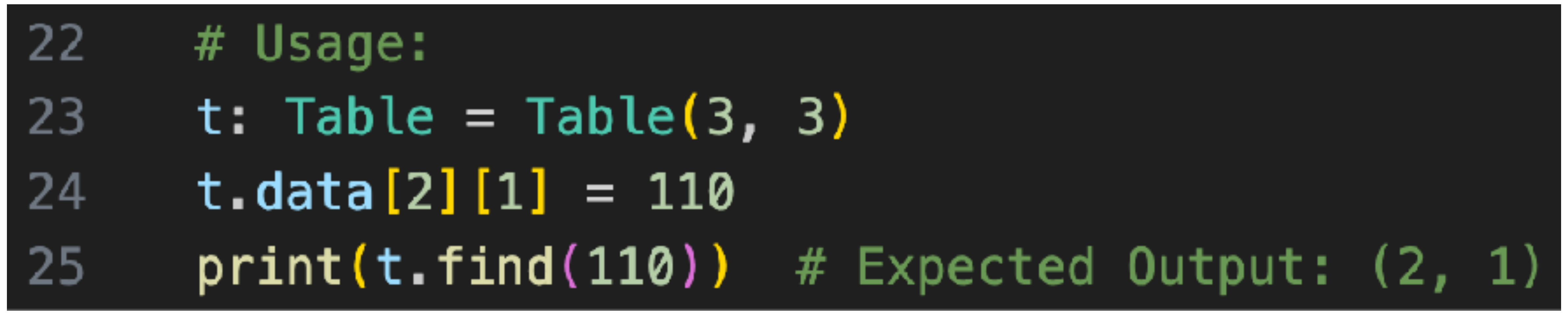
```python
class Table:
    width: int
    height: int
    data: list[list[int]]

    def __init__(self, width: int, height: int):
        # TODO: Initialize width, height, and data in row-major order
        # Row-major: outer list is a list of rows, inner lists are columns
        ...
```

Complete the implementation of *find*



```python
class Table:
    width: int
    height: int
    data: list[list[int]]

    def __init__(self, width: int, height: int):
        self.width = width
        self.height = height
        self.data = []
        for _y in range(height):
            row: list[int] = []
            for _x in range(width):
                row.append(0)
            self.data.append(row)

    def find(self, needle: int) -> tuple[int, int]:
        """Returns the (row, column) of the needle in data.
        If the needle cannot be found, returns (-1, -1)."""
```

Example Usage:



```python
# Usage:
t: Table = Table(3, 3)
t.data[2][1] = 110
print(t.find(110))  # Expected Output: (2, 1)
```