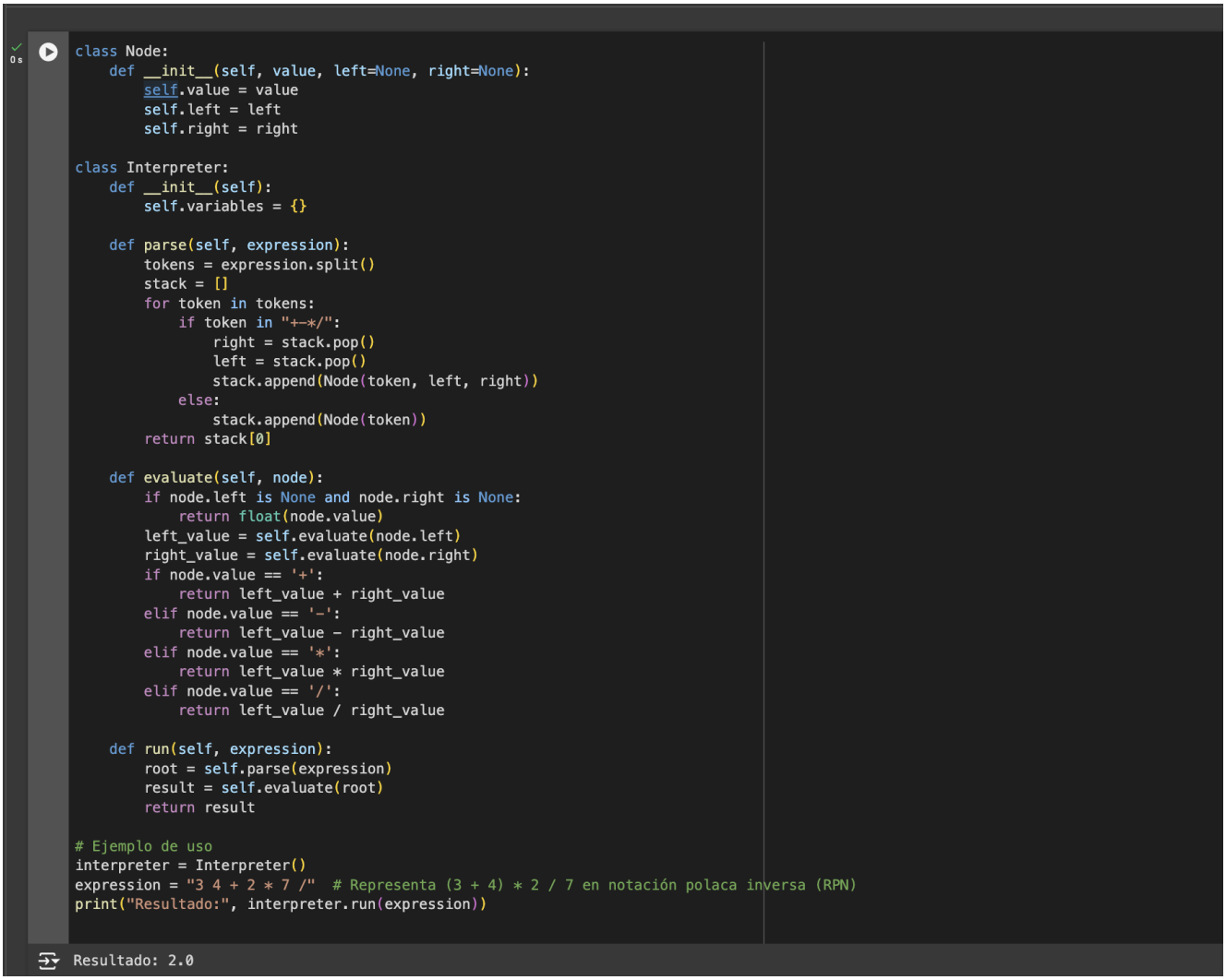
Simulación en Python de un lenguaje interactivo muy básico que maneja expresiones aritméticas con una estructura jerárquica para evaluar las expresiones



```
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

class Interpreter:
    def __init__(self):
        self.variables = {}

    def parse(self, expression):
        tokens = expression.split()
        stack = []
        for token in tokens:
            if token in "+-*/":
                right = stack.pop()
                left = stack.pop()
                stack.append(Node(token, left, right))
            else:
                stack.append(Node(token))
        return stack[0]

    def evaluate(self, node):
        if node.left is None and node.right is None:
            return float(node.value)
        left_value = self.evaluate(node.left)
        right_value = self.evaluate(node.right)
        if node.value == '+':
            return left_value + right_value
        elif node.value == '-':
            return left_value - right_value
        elif node.value == '*':
            return left_value * right_value
        elif node.value == '/':
            return left_value / right_value

    def run(self, expression):
        root = self.parse(expression)
        result = self.evaluate(root)
        return result

# Ejemplo de uso
interpreter = Interpreter()
expression = "3 4 + 2 * 7 /"  # Representa (3 + 4) * 2 / 7 en notación polaca inversa (RPN)
print("Resultado:", interpreter.run(expression))
```

```
Resultado: 2.0
```

[crayon-6a9d4f8d1764d864572522/]

Explicación del código:

1. **Clase Node**: Representa un nodo en el árbol de expresión. Cada nodo puede ser un operador (+, -, *, /) o un operando (un número).
2. **Clase Interpreter**:
3. **Ejemplo de uso**: La expresión «3 4 + 2 * 7 /» se traduce en el árbol de expresión y luego se evalúa para obtener el resultado.