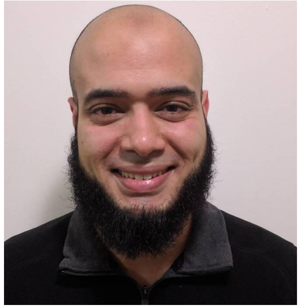
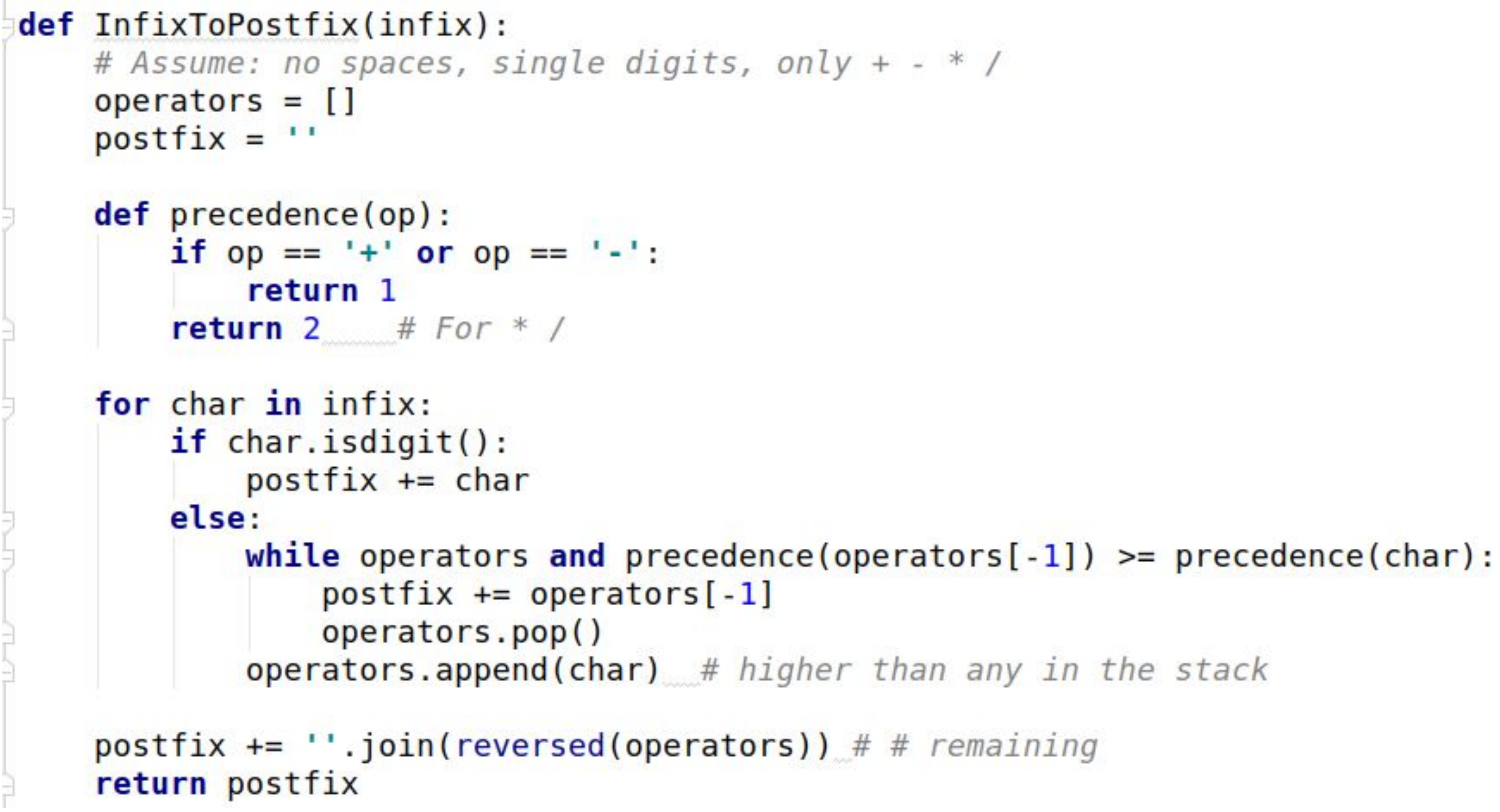
Data Structures

Infix to Postfix Code

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

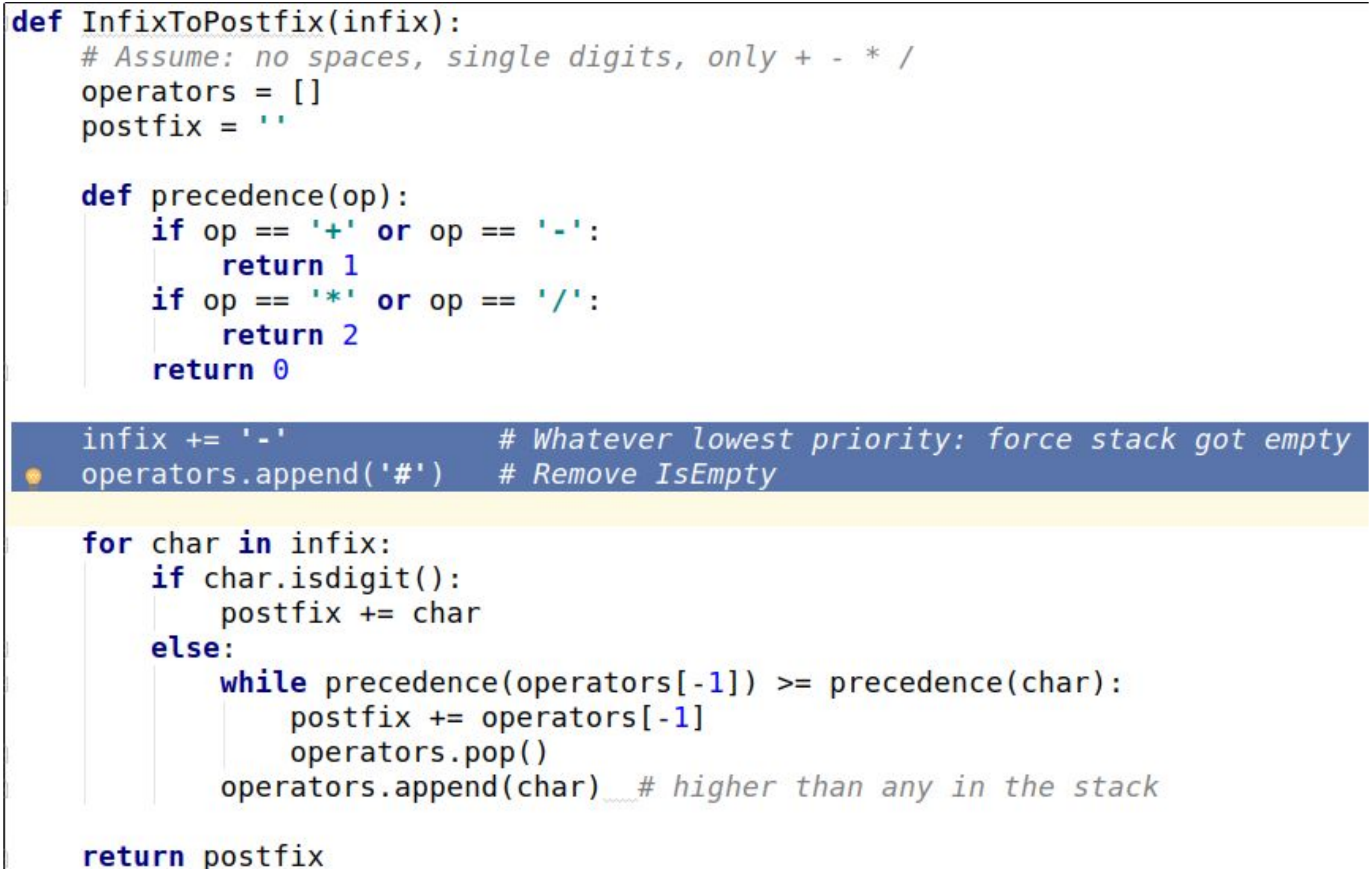




```python
def InfixToPostfix(infix):
    # Assume: no spaces, single digits, only + - * /
    operators = []
    postfix = ''

    def precedence(op):
        if op == '+' or op == '-':
            return 1
        return 2    # For * /

    for char in infix:
        if char.isdigit():
            postfix += char
        else:
            while operators and precedence(operators[-1]) >= precedence(char):
                postfix += operators[-1]
                operators.pop()
            operators.append(char)  # higher than any in the stack

    postfix += ''.join(reversed(operators)) # # remaining
    return postfix
```

Shorter Code!

- Find 2 changes that will simplify the code by:
  - Removing the check if the stack empty
  - Removing the last line to handle remaining of the stack



```python
def InfixToPostfix(infix):
    # Assume: no spaces, single digits, only + - * /
    operators = []
    postfix = ''

    def precedence(op):
        if op == '+' or op == '-':
            return 1
        if op == '*' or op == '/':
            return 2
        return 0

    infix += '-'            # Whatever lowest priority: force stack got empty
    operators.append('#')   # Remove IsEmpty

    for char in infix:
        if char.isdigit():
            postfix += char
        else:
            while precedence(operators[-1]) >= precedence(char):
                postfix += operators[-1]
                operators.pop()
            operators.append(char)  # higher than any in the stack

    return postfix
```

What is the time complexity?

- It looks like we have a while-loop nested inside a for-loop
  - Intuitively, this seems like it should be O(n^2), right?
  - No. The devil is in the details
- The maximum number of operators added to the stack is O(n)
  - And each will be removed once
  - So added once and removed once
- In fact, the code behaves like 2-3 parallel linear loops
  - E.g. 3N operations
  - Verify this in detail to make sure you fully comprehend it

Your Turn:

- Simulate using the code: 1+3*5-8/2
- By hand, convert 2+3*4-5*6+7 and compare it with the algorithm output
- Think for 15 minutes: What if we have ( )
  - Recall that they have higher order (i.e. are of higher precedence)
  - 2+(3*(4-5*2)*(9/3+6))
  - We know each expression () is independent
    - Kind of a separate sub-problem!

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”