(/)

Curriculum
Short Specializations
Average: **97.3%**

0x00. Pascal's Triangle

Algorithm **Python**

By: Alexa Orrico, Software Engineer at Holberton School

Weight: 1

Project over - took place from Nov 27, 2023 6:00 AM to Dec 1, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 0.0/11 mandatory
- **Altogether: 0.0%**
  - Mandatory: 0.0%
  - Optional: no optional tasks

Resources

- What is Pascal's triangle (/rltoken/F458nFkW9StJum2zPI4khg)
- Pascal's Triangle - Numberphile (/rltoken/XXMN2RVCCGcF5I5ZnUIv8Q)
- What are Python Algorithms (/rltoken/q5v0xbgrVxG4Nf-fV-BW2w)

Additional Resources

- Mock Technical Interview (/rltoken/vKf7Spm4xxFMom3x4Jx52g)

Must Know

To successfully complete this project, you should revise the following Python concepts:

1. **Lists and List Comprehensions**:
   - Understand how to create, access, modify, and iterate over lists.

(/)

- Utilize list comprehensions for more concise and readable code, especially for generating rows of Pascal's Triangle.

2. **Functions**:
   - Know how to define and call functions.
   - Pass parameters and return values, particularly how to return a list of lists representing Pascal's Triangle.
3. **Loops**:
   - Use `for` and `while` loops to iterate through sequences.
   - Nested loops may be necessary for generating each row and calculating the values of Pascal's Triangle.
4. **Conditional Statements**:
   - Apply `if`, `elif`, and `else` conditions to implement logic based on the position within Pascal's Triangle (e.g., the edges of the triangle always being 1).
5. **Recursion (Optional)**:
   - While not strictly necessary, understanding recursion can provide an alternative approach to generating Pascal's Triangle.
   - Recognize base cases and recursive cases for a function that generates the triangle's rows.
6. **Arithmetic Operations**:
   - Perform addition, a fundamental operation for calculating each element of Pascal's Triangle as the sum of the two elements directly above it.
7. **Indexing and Slicing**:
   - Access elements and slices of lists, crucial for identifying and summing the correct elements when constructing each row of the triangle.
8. **Memory Management**:
   - Be mindful of how lists are stored and copied, especially when creating new rows based on the values of the previous row.
9. **Error and Exception Handling (Optional)**:
   - Use try-except blocks as needed to handle potential errors, such as invalid input types or values.
10. **Efficiency and Optimization**:
    - Consider the time and space complexity of different approaches to generating Pascal's Triangle.
    - Evaluate and apply optimizations to improve the performance of the solution.

By revisiting these concepts, you will be well-prepared to tackle the challenges of implementing Pascal's Triangle in Python, applying both your mathematical understanding and programming skills to develop an efficient and effective solution.

Tasks

0. Pascal's Triangle

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Create a function `def pascal_triangle(n):` that returns a list of lists of integers representing the Pascal's triangle of `n`:

- Returns an empty list if `n <= 0`
- You can assume `n` will be always an integer

```
guillaume@ubuntu:~/0x00$ cat 0-main.py
#!/usr/bin/python3
"""
0-main
"""
pascal_triangle = __import__('0-pascal_triangle').pascal_triangle

def print_triangle(triangle):
    """
    Print the triangle
    """
    for row in triangle:
        print("[{}]".format(",".join([str(x) for x in row])))


if __name__ == "__main__":
    print_triangle(pascal_triangle(5))

guillaume@ubuntu:~/0x00$
guillaume@ubuntu:~/0x00$ ./0-main.py
[1]
[1,1]
[1,2,1]
[1,3,3,1]
[1,4,6,4,1]
guillaume@ubuntu:~/0x00$
```

Repo:

- GitHub repository: `alx-interview`
- Directory: `0x00-pascal_triangle`
- File: `0-pascal_triangle.py`

☐ Done? Help Check your code Ask for a new correction QA Review

(/)