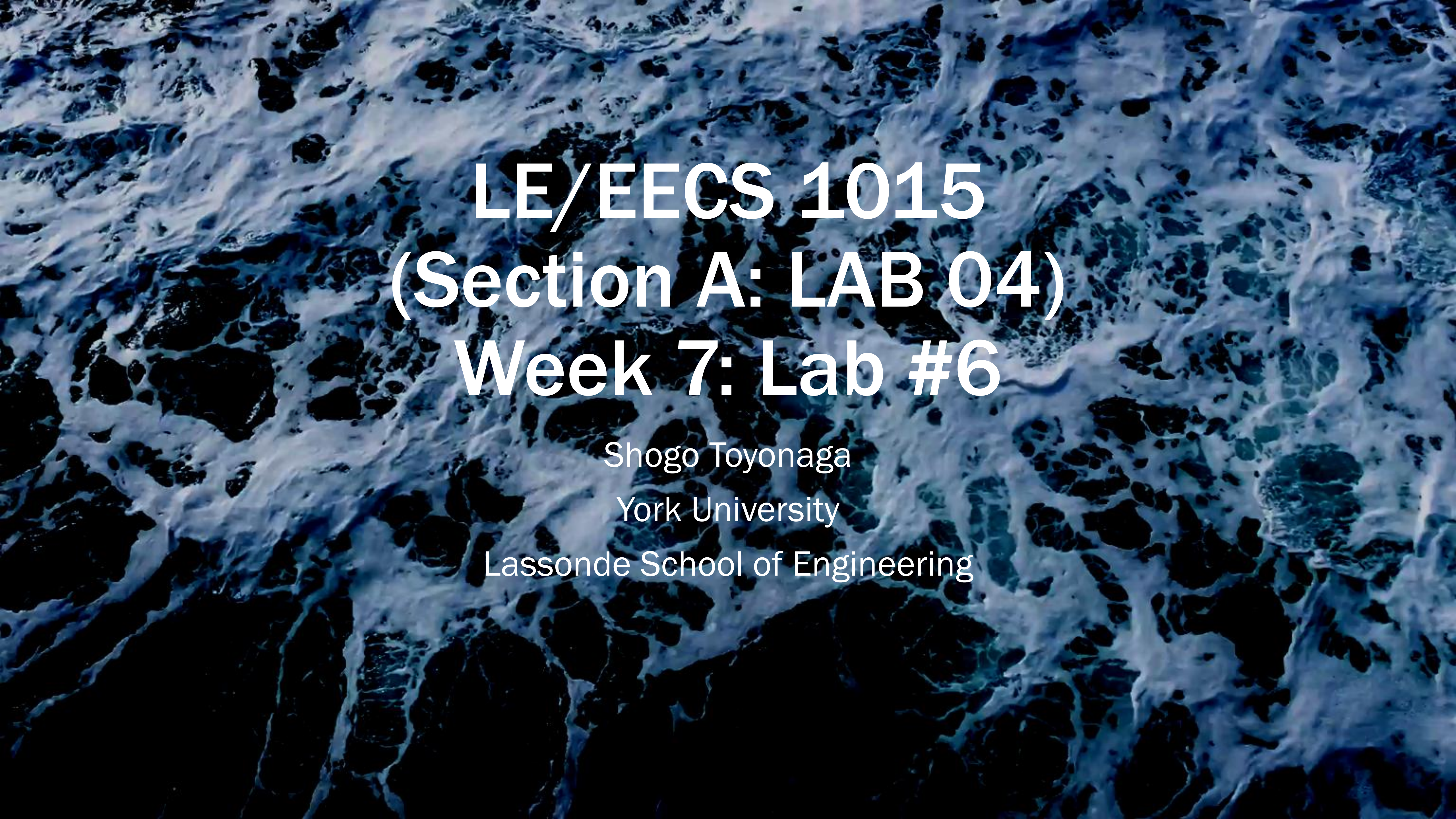


LE/EECS 1015 (Section A: LAB 04) Week 7: Lab #6

Shogo Toyonaga

York University

Lassonde School of Engineering

Goals of Lab 6

1. Computational thinking with conditional statements, loops, and functions (modules).
2. Debugging

Concept Review

1. Booleans (Propositional Statements)
2. If-Statements
   - if-else
   - if-elif-else
3. Loops
   - For
   - While

   break
   continue
4. Refactoring
   - Clean code is good code 😊
   - Pre-structuring and formatting with, “pass” keyword

Relational Operators

Operator	Example $(\boldsymbol{Assume\ r = 5})$
Less than ($<$)	$r < 0\ is\ False$
Less than or equal to ($\leq$)	$r \leq 0\ is\ False$
Greater than ($>$)	$r > 0\ is\ True$
Greater than or equal to ($\geq$)	$r \geq 0\ is\ True$
Equality ($==$)	$r == 0\ is\ False$
Inequality ($! =$)	$r\ ! = 0\ is\ True$

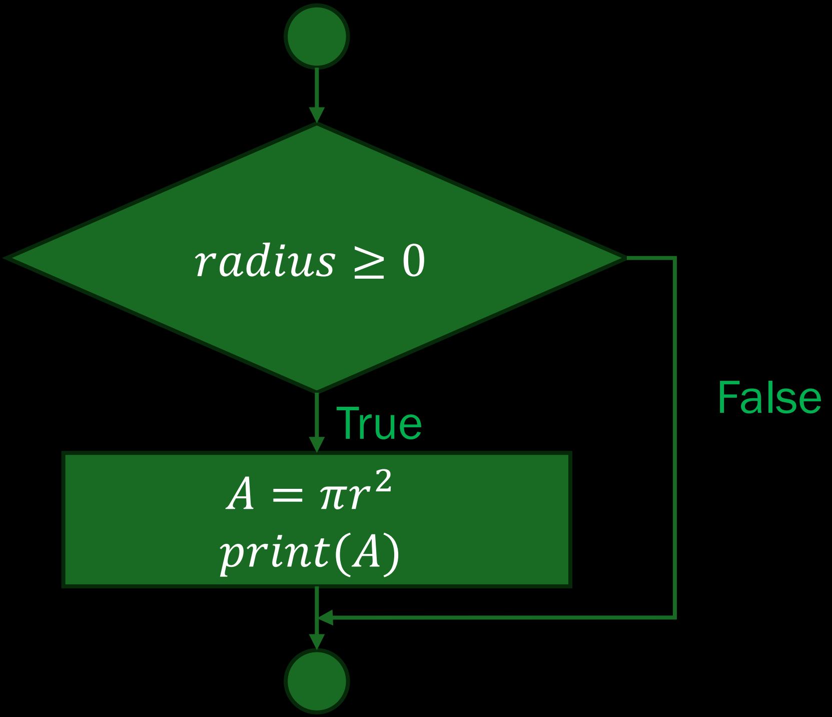
Boolean Operators (Only Some...)

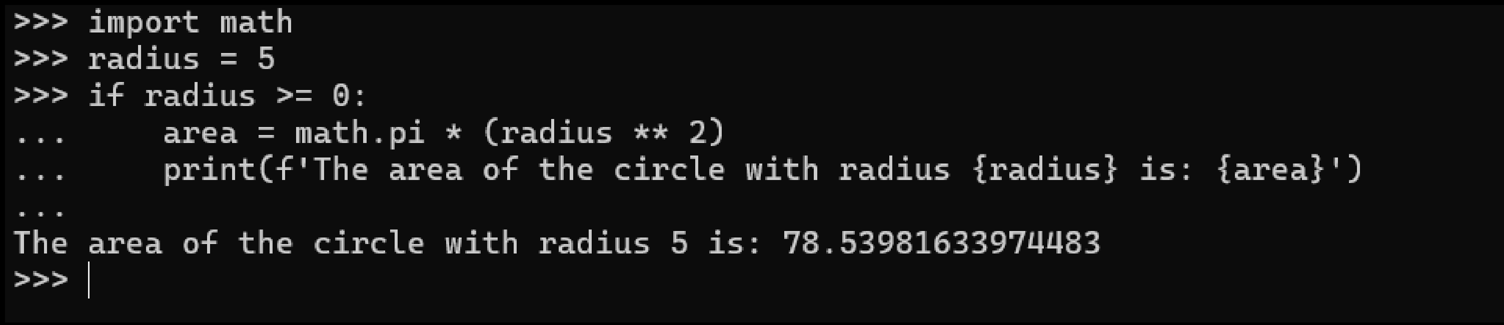
p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	T
F	F	T	F	F	T

If-Else Statements (Branching)

Assume that we are given the radius of a circle.

Draw a flowchart of a program that only calculates the area if $\boldsymbol{radius \geq 0}$.





```
>>> import math
>>> radius = 5
>>> if radius >= 0:
...     area = math.pi * (radius ** 2)
...     print(f'The area of the circle with radius {radius} is: {area}')
...
The area of the circle with radius 5 is: 78.53981633974483
>>>
```

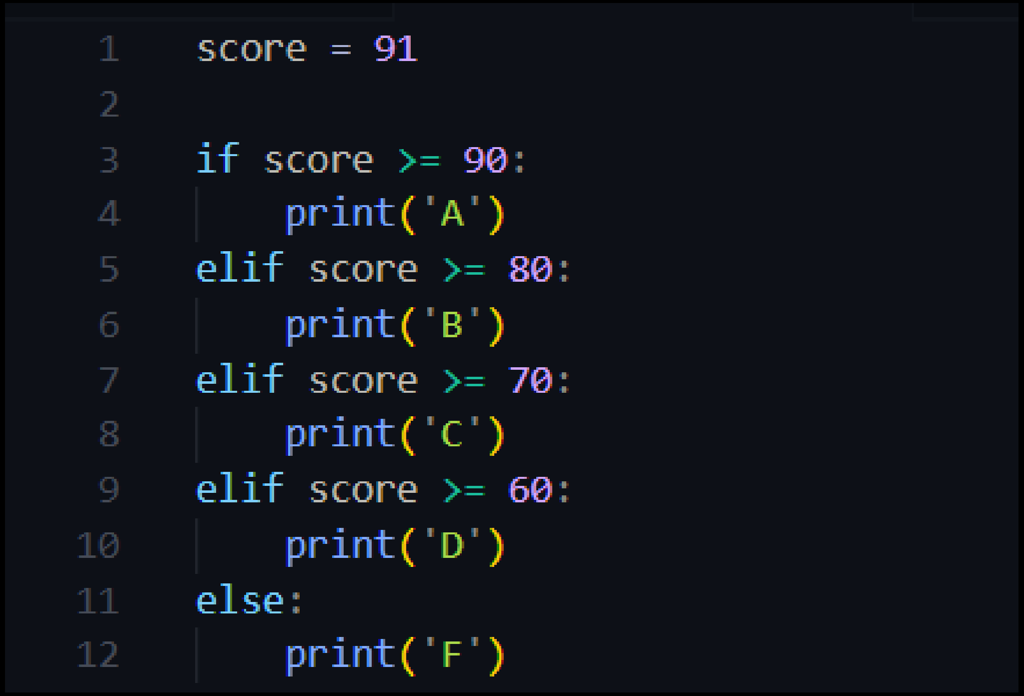
If-Elif-Else Statements (Branching)

Write a function that calculates your letter grade given the raw score.

Raw Mark	Letter Grade
$score \geq 90$	A
$score \geq 80$	B
$score \geq 70$	C
$score \geq 60$	D
$score < 60$	F

If-Elif-Else Statements (Branching)

Write a function that calculates your letter grade given the raw score.



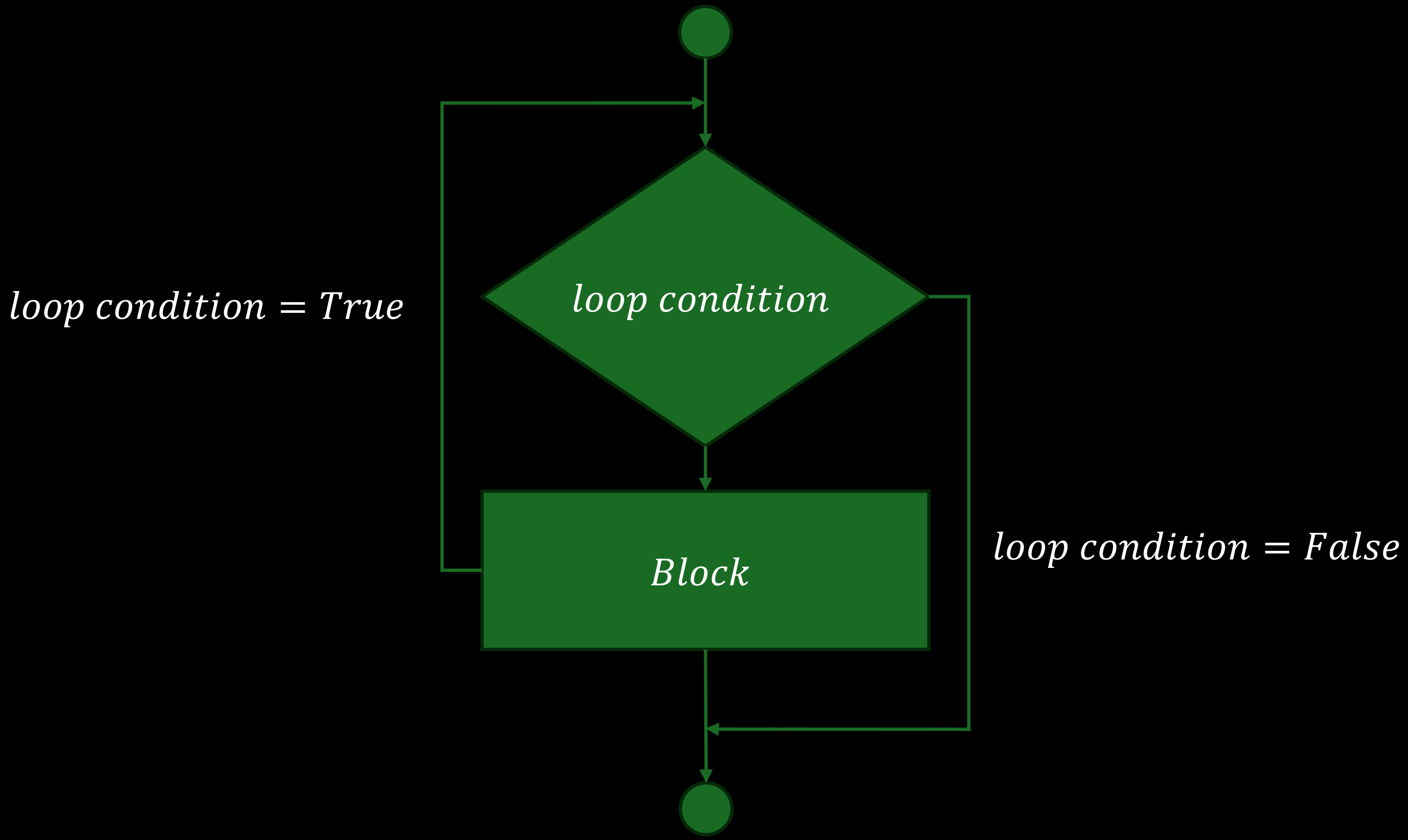
```python
score = 91

if score >= 90:
    print('A')
elif score >= 80:
    print('B')
elif score >= 70:
    print('C')
elif score >= 60:
    print('D')
else:
    print('F')
```

- What happens if all the statements use if instead of $if, elif, else$?
- What happens if we check the statements in reverse order? Try to think about why we ordered the statement checking this way.

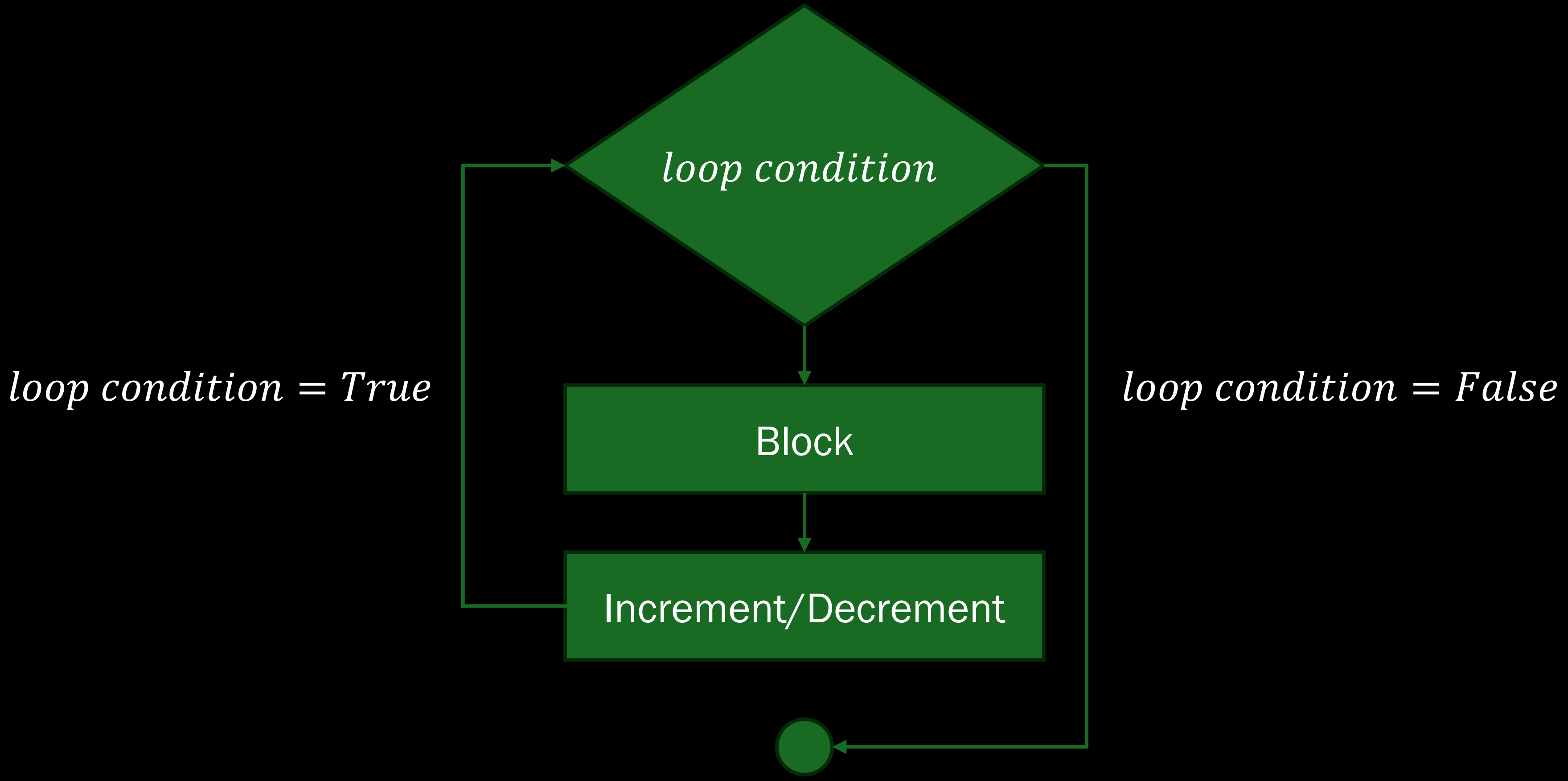
For & While (Looping)

- A while loop will execute statements while its loop-condition remains true. It gives us some more flexibility.
- A for-loop will execute statements for a specified number of iterations or objects. It is very clear and well-defined.
- $\boldsymbol{break}$ and $\boldsymbol{continue}$ offer ways to forcibly exit a loop or an iteration.

While Loop (Structure)



For Loop (Structure)

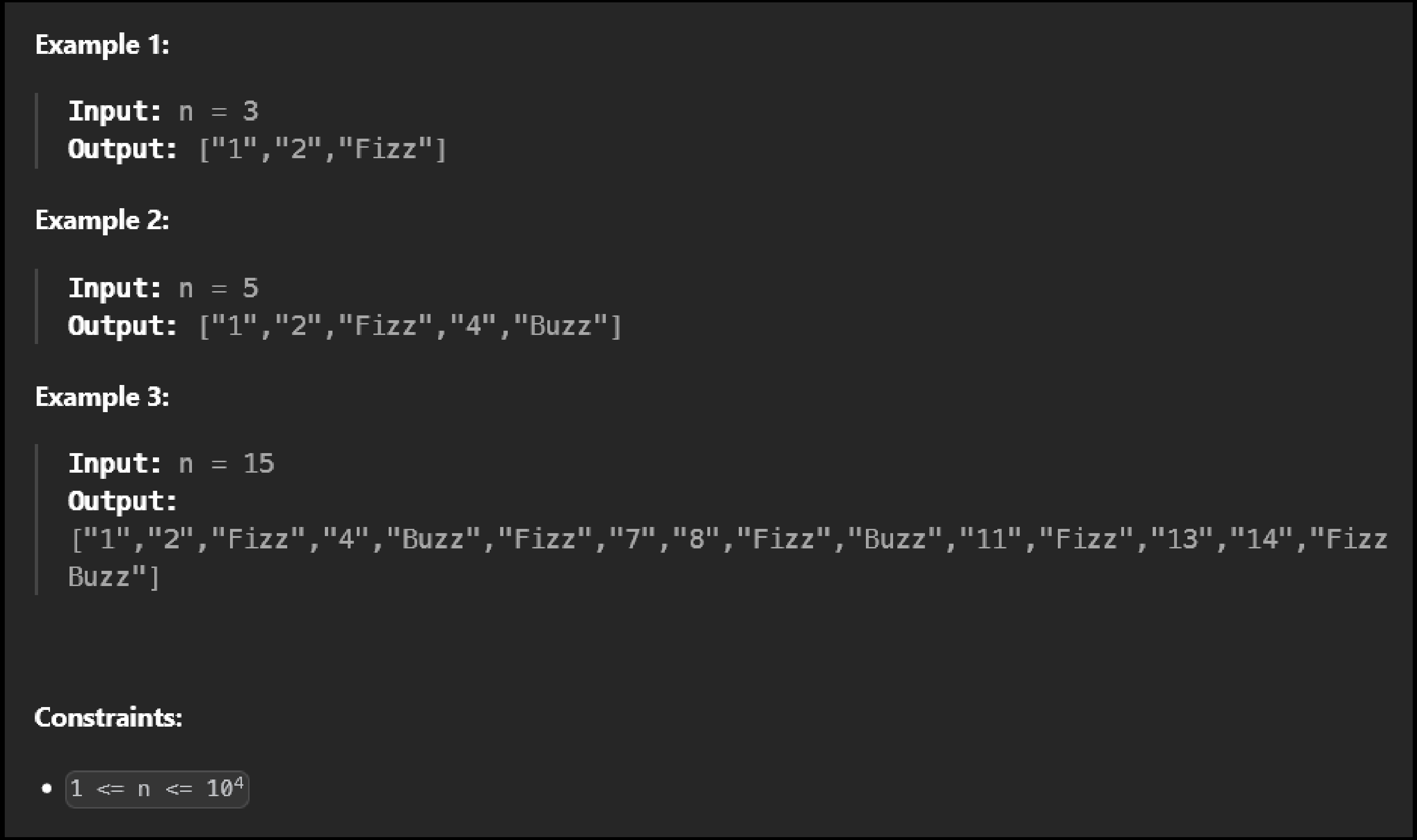


Putting it Together (FizzBuzz)

Given an integer n, return a string array $answer$ (1-indexed), where:

- $answer[i] ==$ "$FizzBuzz$" if i is divisible by 3 and 5
- $answer[i] ==$ "$FizzBuzz$" if i is divisible by 3
- $answer[i] ==$ "$Buzz$" if i is divisible by 5
- $answer[i] == i$ (as a string) if none of the above conditions are true.

Putting it Together (FizzBuzz)



Example 1:

```
Input: n = 3
Output: ["1","2","Fizz"]
```

Example 2:

```
Input: n = 5
Output: ["1","2","Fizz","4","Buzz"]
```

Example 3:

```
Input: n = 15
Output:
["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","Fizz
Buzz"]
```

Constraints:

- `1 <= n <= 10`^{4}

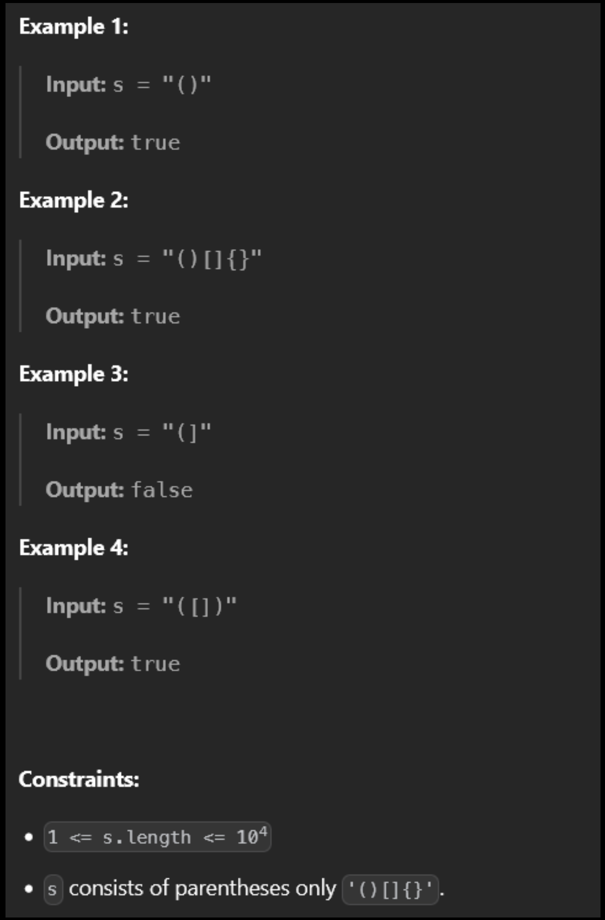
Putting it Together (Valid Parentheses)

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Putting it Together (Valid Parentheses)


Example 1:

> **Input:** `s = "()"`
>
> **Output:** `true`

Example 2:

> **Input:** `s = "()[]{}"`
>
> **Output:** `true`

Example 3:

> **Input:** `s = "(]"`
>
> **Output:** `false`

Example 4:

> **Input:** `s = "([])"`
>
> **Output:** `true`

Constraints:

- `1 <= s.length <= 10`^{4}
- `s` consists of parentheses only `'()[]{}'`.

Nested Loops (🤯)

- For every iteration of the outer loop, the inner-loop is always run from start-to-finish.
    - What do you think this does to the time-complexity of the code?
    - What use-cases do you think nested-loops have?

Recap: Good Software Design Principles

1. “Oak's words echoed... "There's a time and place for everything but not now!"”
    - Picking the right tools (for vs while), number of nested loops, etc., is important.
    - You must be able to justify your choices and keep in mind readability and complexity.
2. Always remember your indentation and colons!
3. Doublecheck your loop conditions or you may get trapped infinitely. Avoid tautologies at all costs!
4. Never leave a block empty, use “pass” if necessary
5. A good rule of thumb is to reconsider your approach to a problem if you must indent more than 3 times...

Lab 6 – Objectives

1. Task 1: Follow the Steps (Primes) (/30)
2. Task 2:Debugging (Sums) (/30)
3. Task 3: Implementation (Leibniz) (/10)
4. Task 4: Implementation (Caesar) (/10)
5. Task 5: Implementation (Reverse String) (/10)
6. Task 6: Implementation (Remove Vowels) (/10)

Thank You!

Shogo Toyonaga

Lassonde School of Engineering