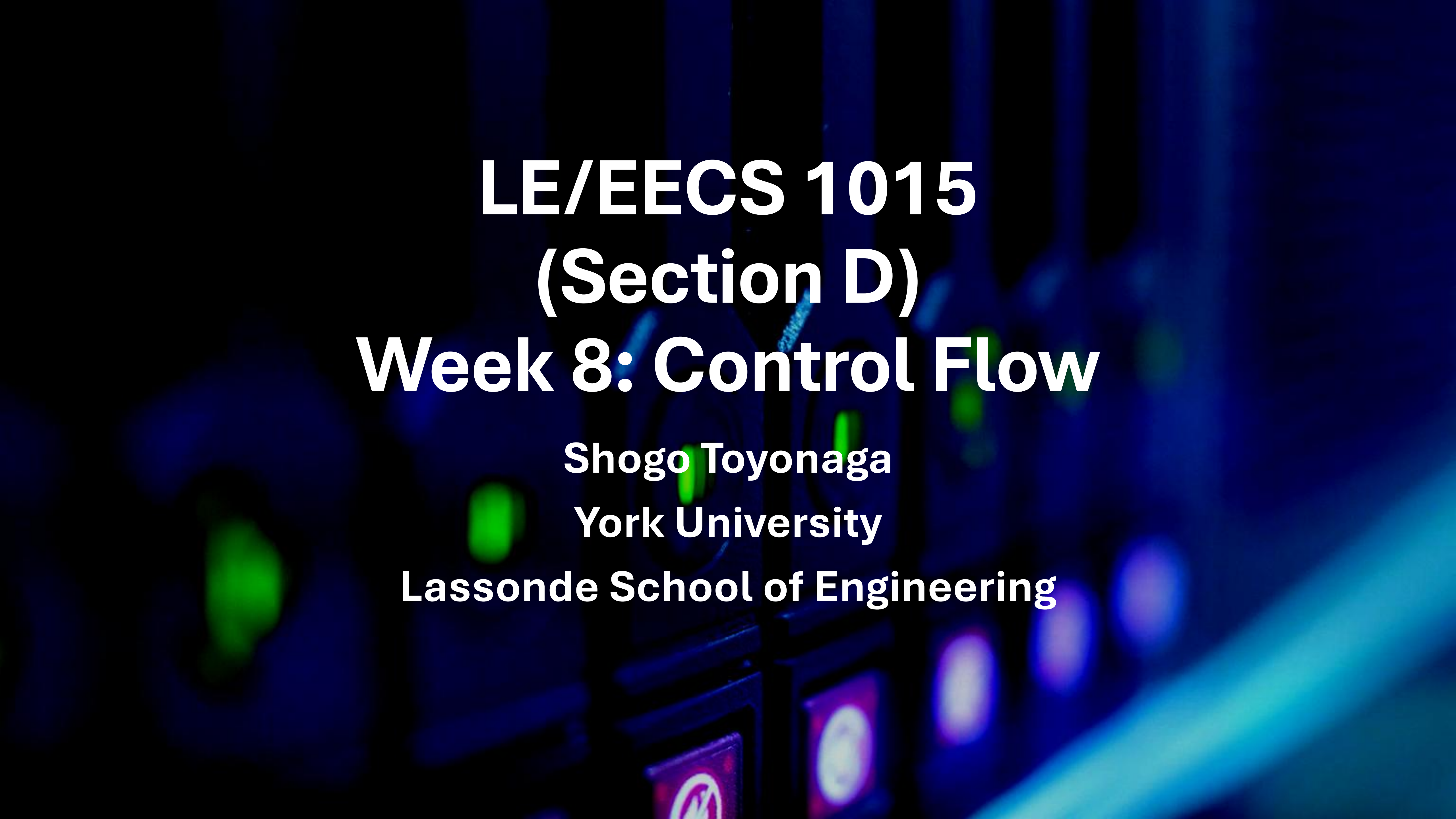


LE/EECS 1015 (Section D) Week 8: Control Flow

Shogo Toyonaga

York University

Lassonde School of Engineering

This Week…

1. If-Statements
   - If-Else
   - If-Elif-Else
2. Loops (For, While)
   - For
   - While
   - Keywords: continue, break
3. Refactoring
   - Pre-Formatting (Keyword: pass)
   - Avoiding Nesting (As much as possible)

Goals of Lab 6

1. Writing and debugging algorithms that have non-linear execution flow.
2. Understanding the differences between for and while loops.
3. Writing clean code and avoiding unnecessary nesting.
4. Gaining hands on-experience with other built-in Python functions.

Lab 6 – What You Do….

Task	Points
Follow the Steps (Count Primes)	30
Debugging (Repeat Sum)	30
Implementation (Leibniz Formula)	10
Implementation (Caesar Cipher)	10
Implementation (Reverse String)	10
Implementation (Remove Vowels)	10

Lab 6 – Useful Resources

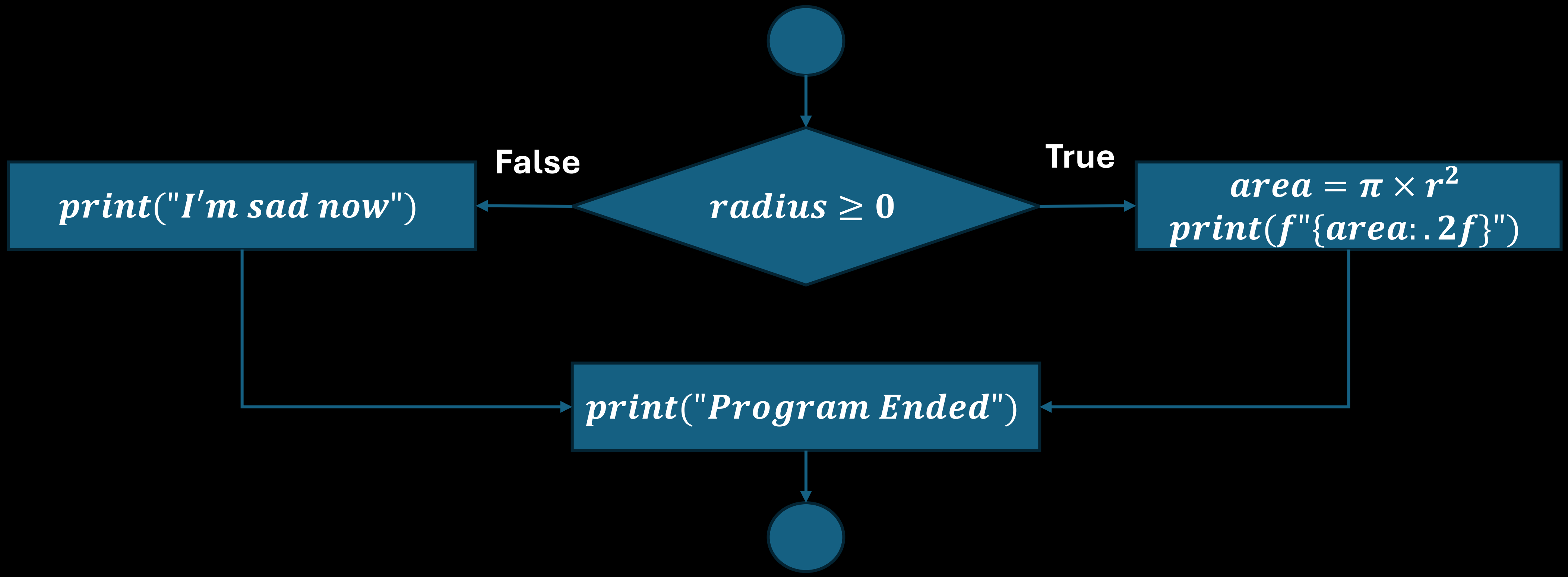
- If statements in Python are easy (if, elif, else) (BroCode)
- For loops in Python are easy (BroCode)
- While loops in Python are easy (BroCode)
- Python break continue pass (BroCode)
- Converting Between while and for Loops (Caleb Curry)
- Python Documentation
- pyflowchart module

Introduction (Motivation)

- All the programs we have written in this course have been linear in behaviour.
- What if we want our code to behave differently based on certain (Boolean) conditions of an input? *This behaviour could be count-controlled or based on a sentinel guard*.

Example: If-Else Statement

Given the radius of a circle, calculate the area if and only if the input is non-negative.

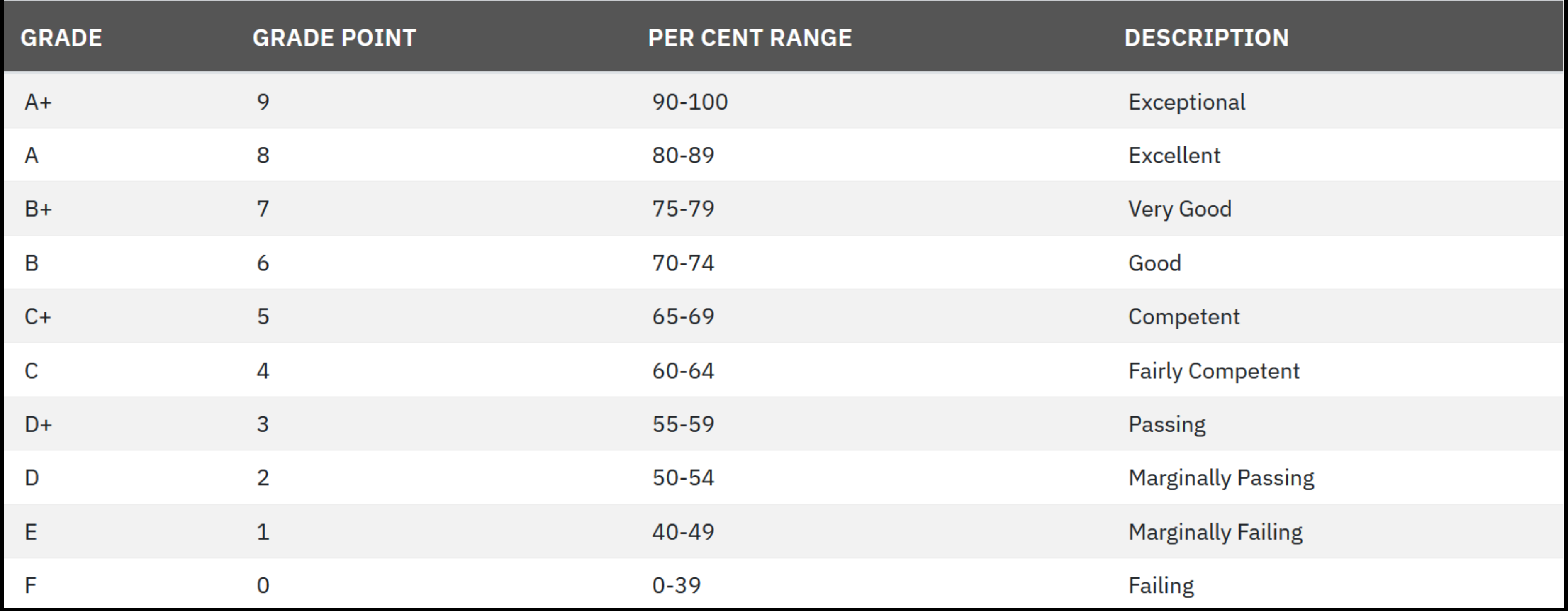


Example: If-Elif-Else

Given a percentage range for a course grade, use Yorks Grading Scheme to print out the Grade Letter, Grade Point, and Description. You are not allowed to use lists, sets, tuples, or dicts to solve this problem.

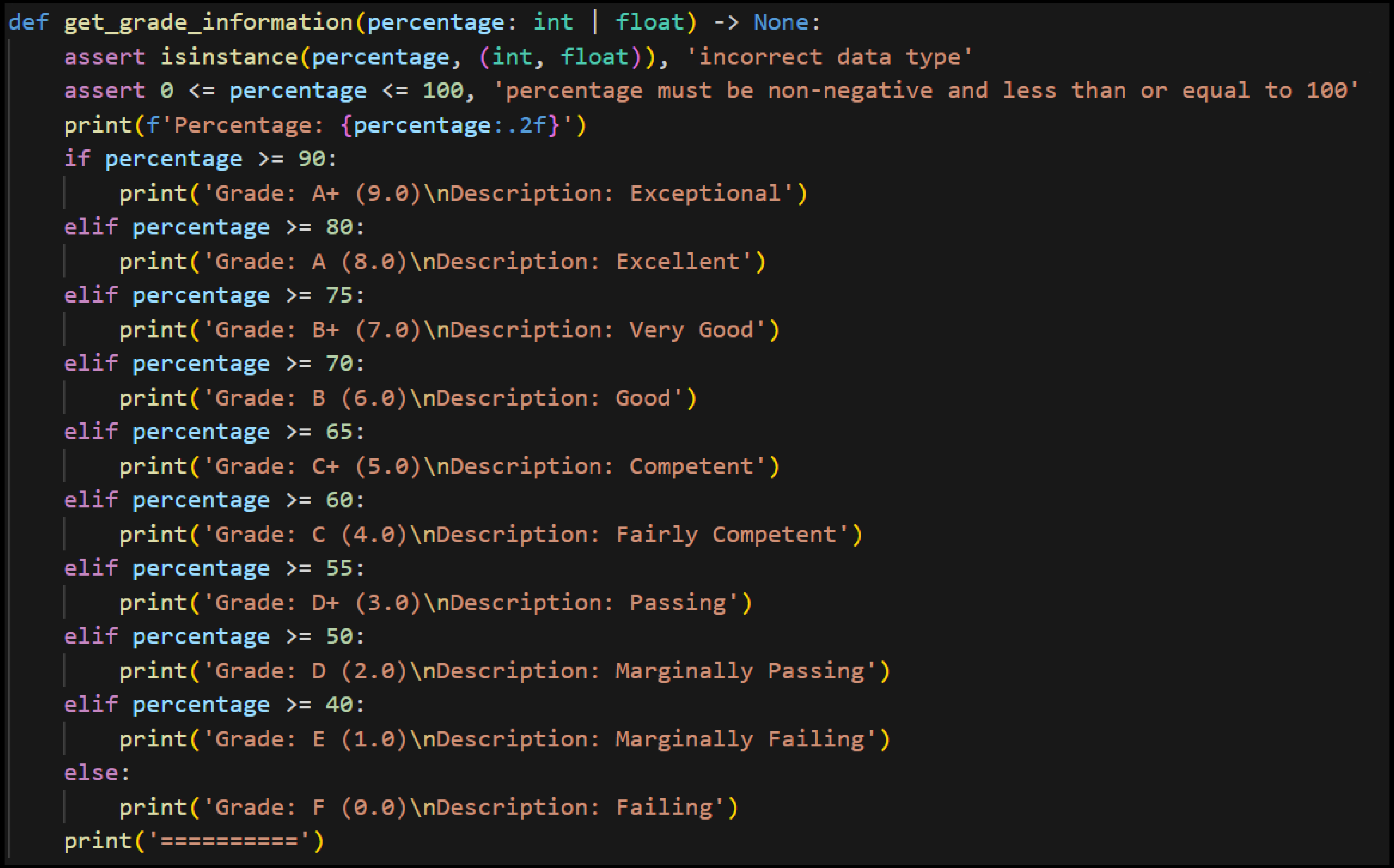
Example: If-Elif-Else

Given a percentage range for a course, use Yorks Grading Scheme to print out the Grade Letter, Grade Point, and Description.



GRADE	GRADE POINT	PER CENT RANGE	DESCRIPTION
A+	9	90-100	Exceptional
A	8	80-89	Excellent
B+	7	75-79	Very Good
B	6	70-74	Good
C+	5	65-69	Competent
C	4	60-64	Fairly Competent
D+	3	55-59	Passing
D	2	50-54	Marginally Passing
E	1	40-49	Marginally Failing
F	0	0-39	Failing

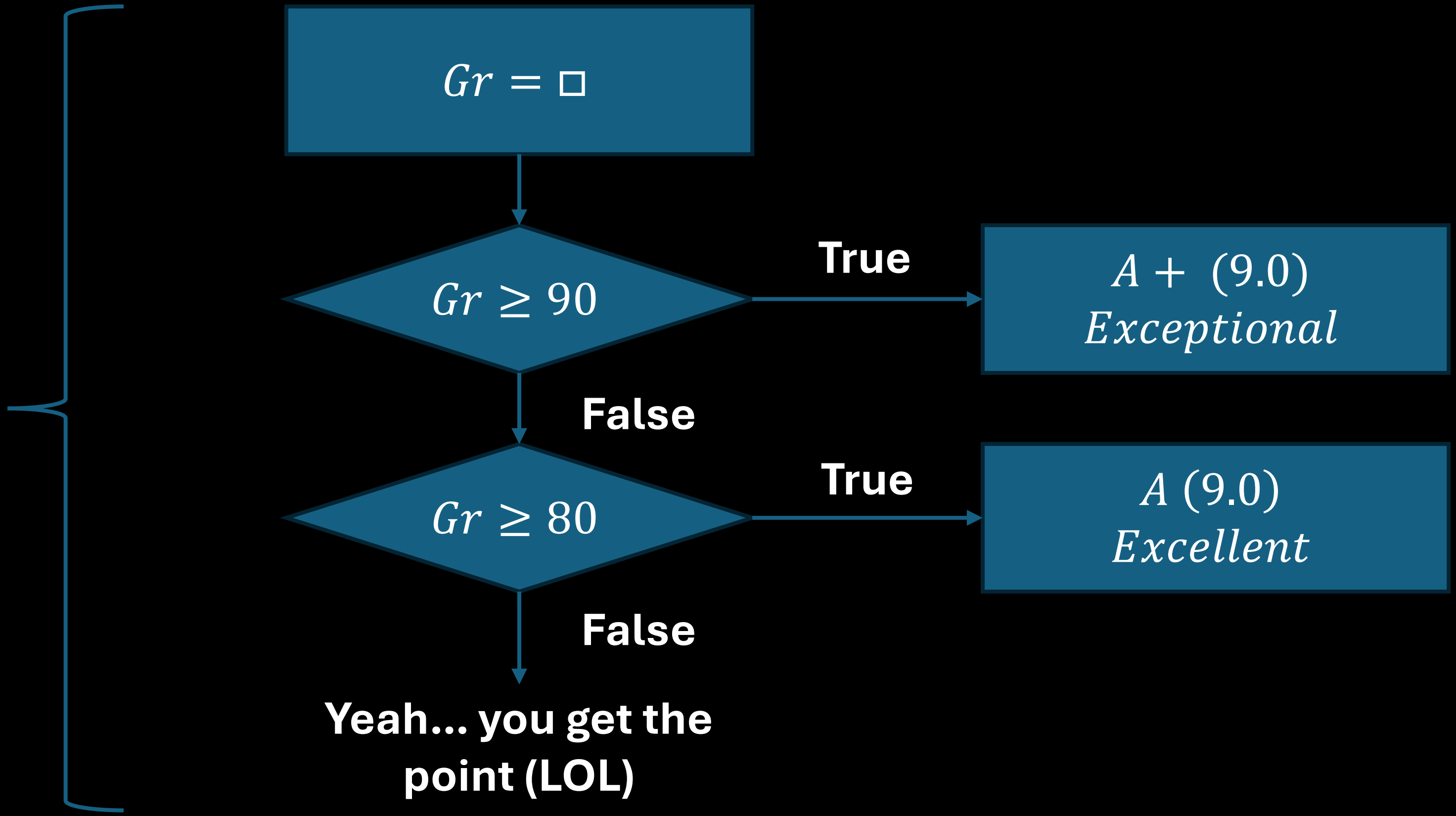
Example: If-Elif-Else



```python
def get_grade_information(percentage: int | float) -> None:
    assert isinstance(percentage, (int, float)), 'incorrect data type'
    assert 0 <= percentage <= 100, 'percentage must be non-negative and less than or equal to 100'
    print(f'Percentage: {percentage:.2f}')
    if percentage >= 90:
        print('Grade: A+ (9.0)\nDescription: Exceptional')
    elif percentage >= 80:
        print('Grade: A (8.0)\nDescription: Excellent')
    elif percentage >= 75:
        print('Grade: B+ (7.0)\nDescription: Very Good')
    elif percentage >= 70:
        print('Grade: B (6.0)\nDescription: Good')
    elif percentage >= 65:
        print('Grade: C+ (5.0)\nDescription: Competent')
    elif percentage >= 60:
        print('Grade: C (4.0)\nDescription: Fairly Competent')
    elif percentage >= 55:
        print('Grade: D+ (3.0)\nDescription: Passing')
    elif percentage >= 50:
        print('Grade: D (2.0)\nDescription: Marginally Passing')
    elif percentage >= 40:
        print('Grade: E (1.0)\nDescription: Marginally Failing')
    else:
        print('Grade: F (0.0)\nDescription: Failing')
    print('==========')
```

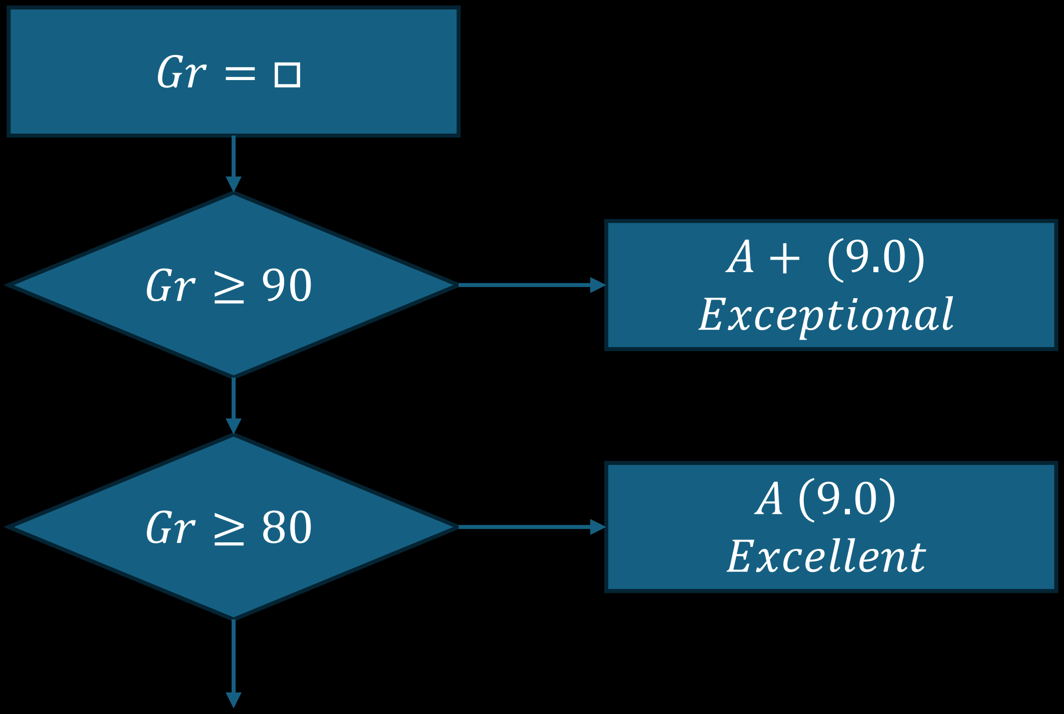
Example: If-Elif-Else

Given the solution to the Grade Calculator, draw the corresponding Control-Flow Diagram!



Example: If-Elif-Else

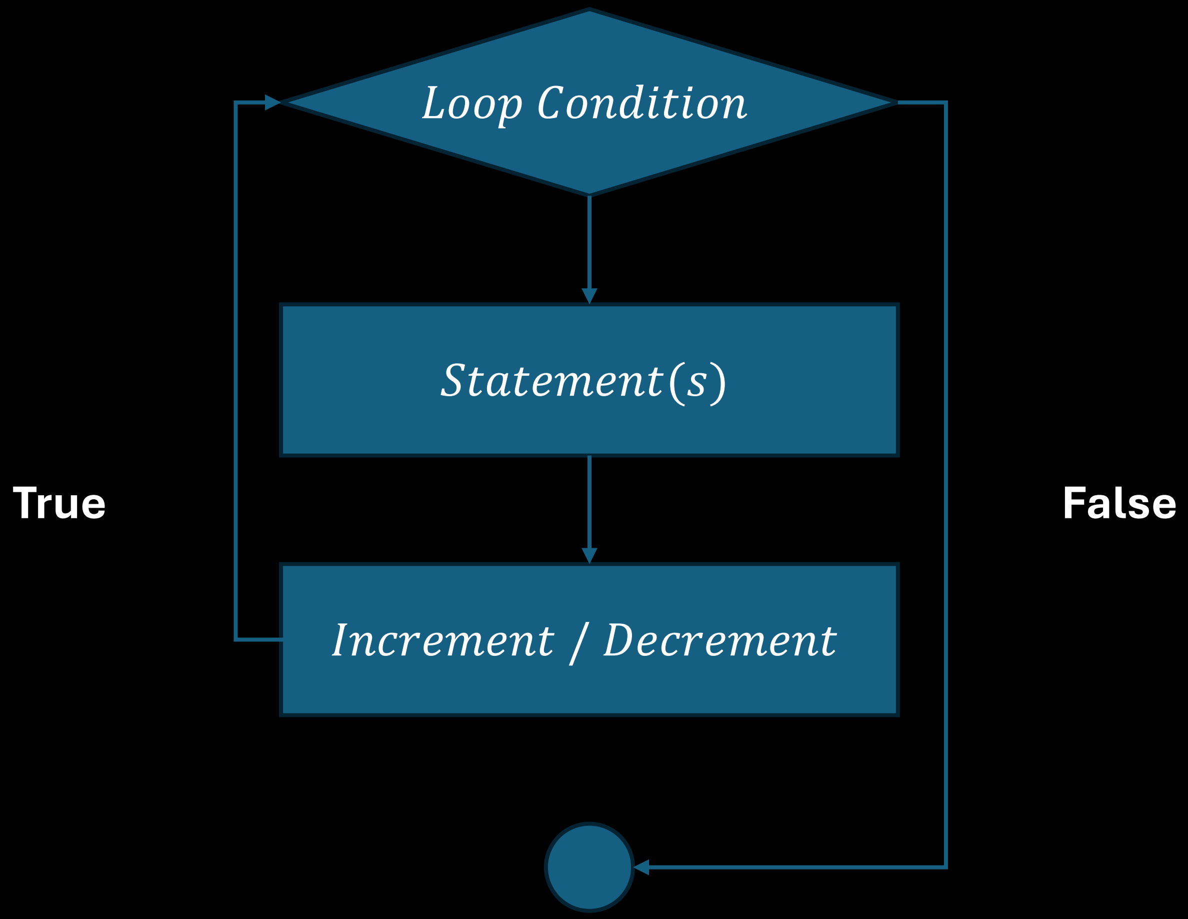
Given the solution, what would happen to the program if all of the, “elifs/else” were replaced with, “if”? Draw the corresponding control flow diagram.



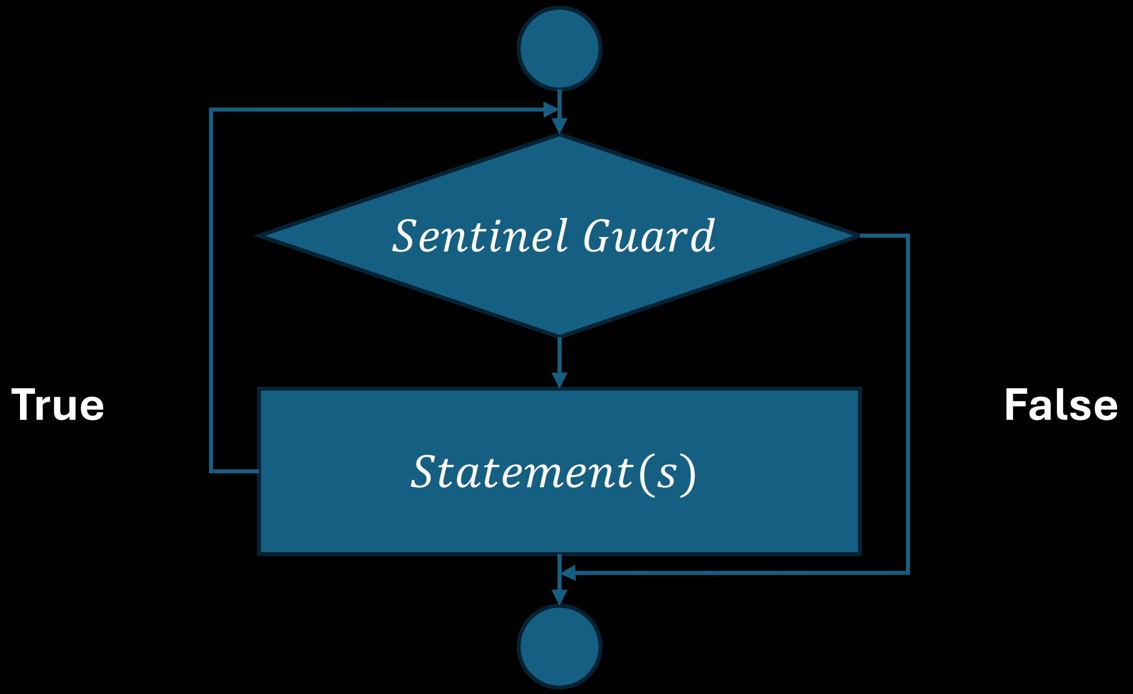
A Preface to Loops (For, While)

- You will often need to write code that performs multiple repetitions of similar tasks. We want to write the code in a clean, concise, and readable form.
- Counted loops (for) will execute a set of statements for a specified number of iterations and/or objects.
- Conditional loops (while) will execute a set of statements until a Boolean condition (sentinel guard) evaluates to false.

For Loop

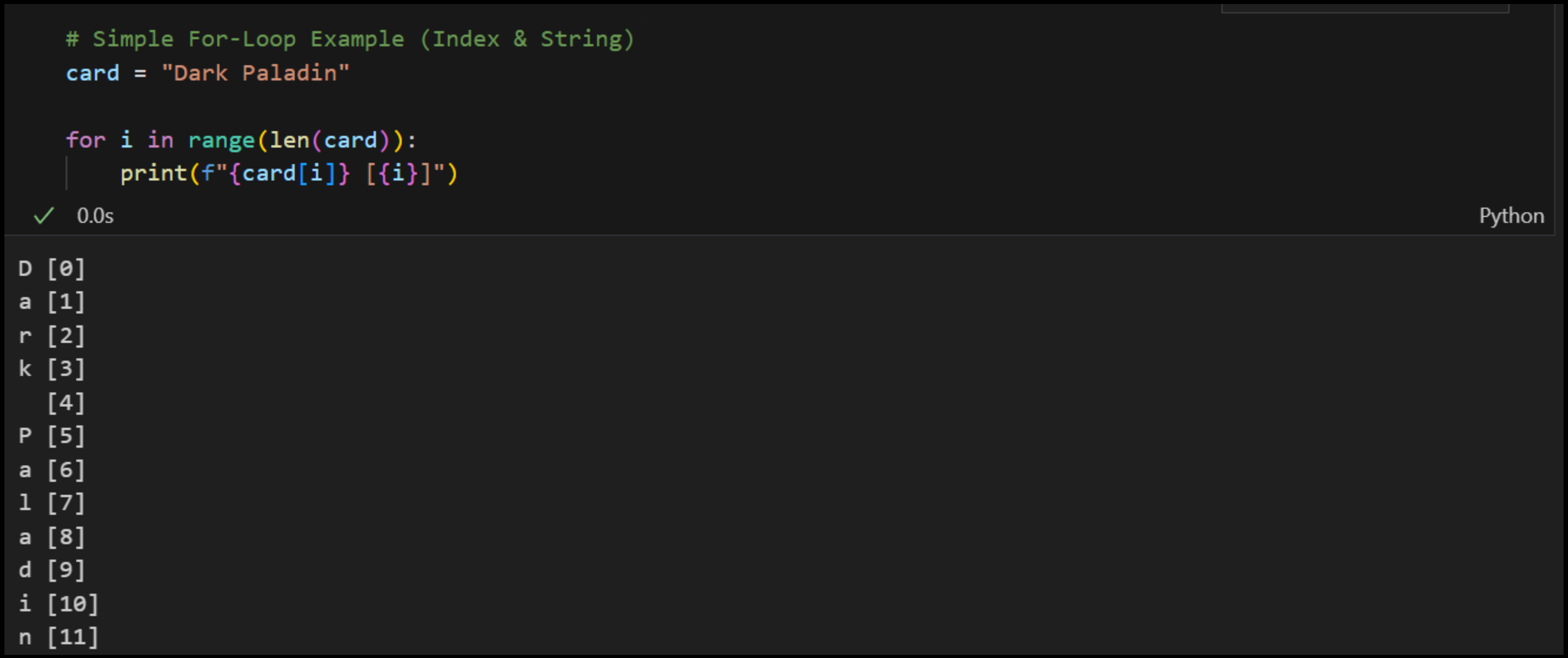

Loop Condition
Statement(s)
True
False
Increment / Decrement

While Loop



Example: For Loop

Given a string (e.g., “Dark Paladin”), print out each letter with its corresponding index.



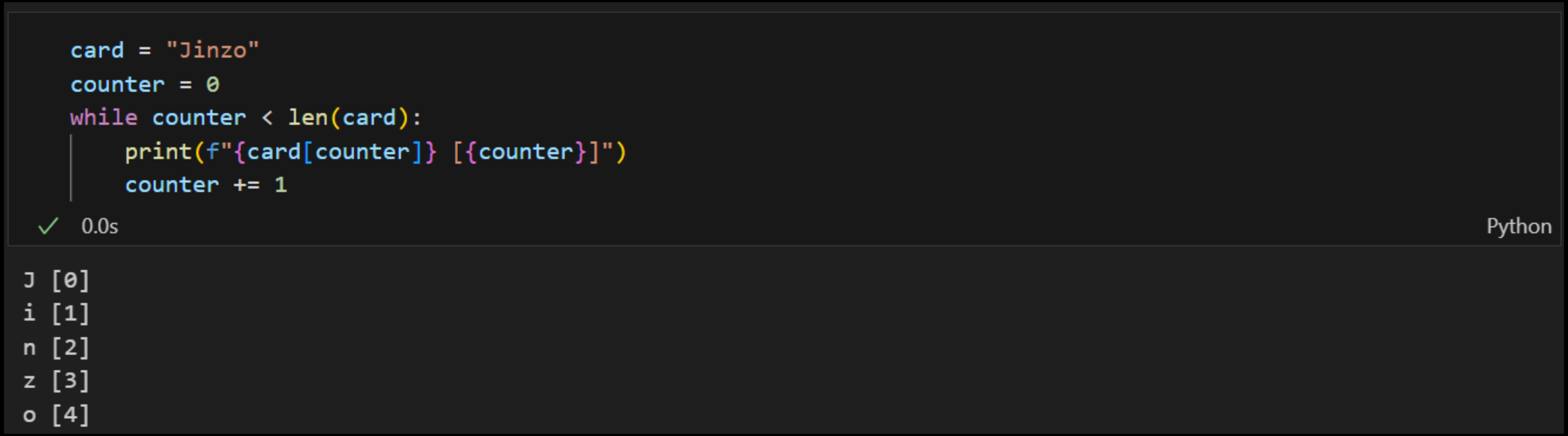
```python
# Simple For-Loop Example (Index & String)
card = "Dark Paladin"

for i in range(len(card)):
    print(f"{card[i]} [{i}]")
```

```
D [0]
a [1]
r [2]
k [3]
  [4]
P [5]
a [6]
l [7]
a [8]
d [9]
i [10]
n [11]
```

Example: While Loop

Given a string (e.g., “Jinzo”), print out each letter with its corresponding index.



```python
card = "Jinzo"
counter = 0
while counter < len(card):
    print(f"{card[counter]} [{counter}]")
    counter += 1
```

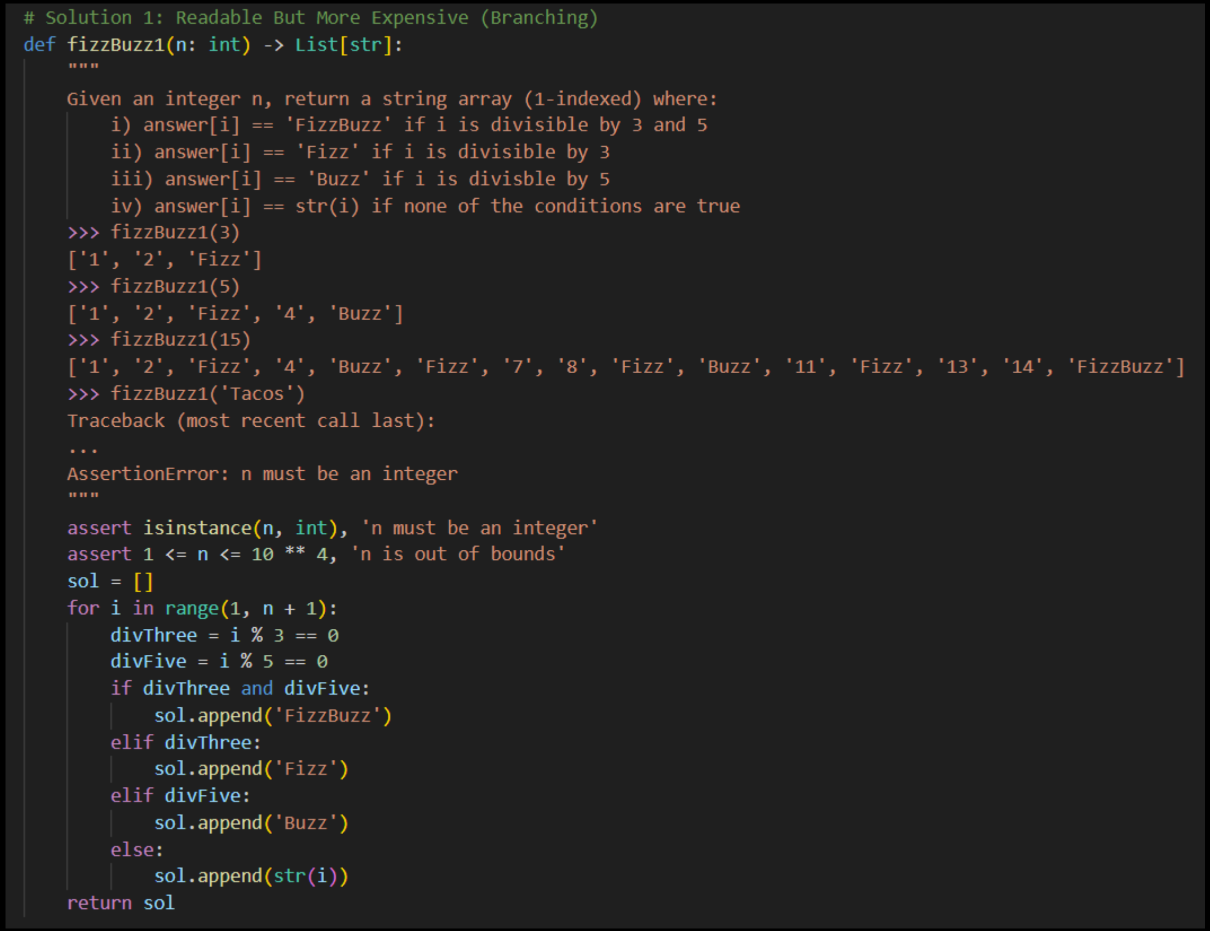
```
J [0]
i [1]
n [2]
z [3]
o [4]
```

Putting It All Together (FizzBuzz)

- Given an integer n, return a string array (1-indexed) where:
  1. answer[i] == 'FizzBuzz' if i is divisible by 3 and 5
  2. answer[i] == 'Fizz' if i is divisible by 3
  3. answer[i] == 'Buzz' if i is divisible by 5
  4. answer[i] == str(i) if none of the conditions are true

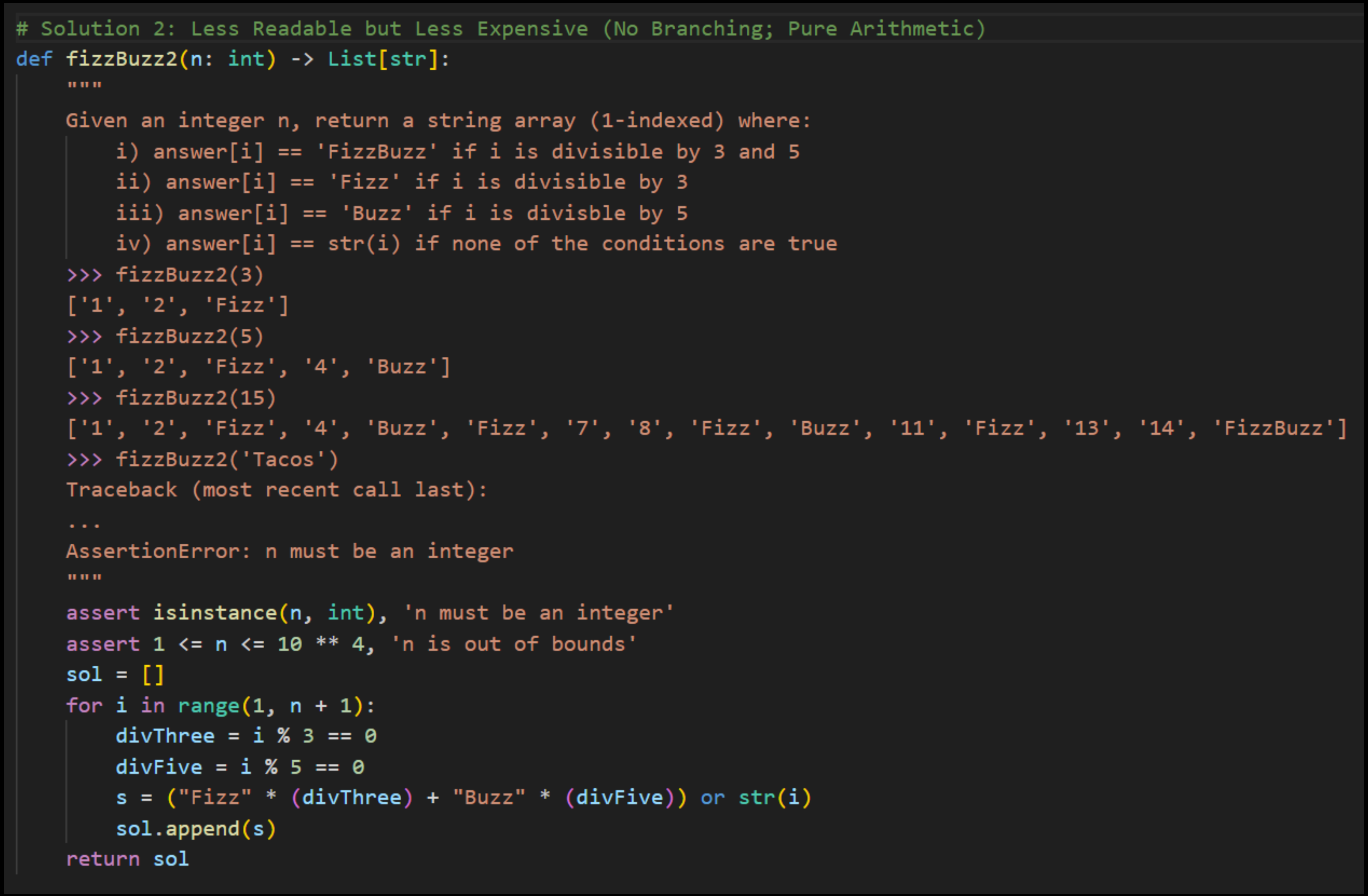
- Examples
  1. fizzBuzz(3) ➔ ['1', '2', 'Fizz']
  2. fizzBuzz(5) ➔ ['1', '2', 'Fizz', '4', 'Buzz’]
  3. fizzBuzz(15) ➔ ['1', '2', 'Fizz', '4', 'Buzz', 'Fizz', '7', '8', 'Fizz', 'Buzz', '11', 'Fizz', '13', '14', 'FizzBuzz']

Putting It All Together (FizzBuzz)



```python
# Solution 1: Readable But More Expensive (Branching)
def fizzBuzz1(n: int) -> List[str]:
    """
    Given an integer n, return a string array (1-indexed) where:
        i) answer[i] == 'FizzBuzz' if i is divisible by 3 and 5
        ii) answer[i] == 'Fizz' if i is divisible by 3
        iii) answer[i] == 'Buzz' if i is divisble by 5
        iv) answer[i] == str(i) if none of the conditions are true
    >>> fizzBuzz1(3)
    ['1', '2', 'Fizz']
    >>> fizzBuzz1(5)
    ['1', '2', 'Fizz', '4', 'Buzz']
    >>> fizzBuzz1(15)
    ['1', '2', 'Fizz', '4', 'Buzz', 'Fizz', '7', '8', 'Fizz', 'Buzz', '11', 'Fizz', '13', '14', 'FizzBuzz']
    >>> fizzBuzz1('Tacos')
    Traceback (most recent call last):
    ...
    AssertionError: n must be an integer
    """
    assert isinstance(n, int), 'n must be an integer'
    assert 1 <= n <= 10 ** 4, 'n is out of bounds'
    sol = []
    for i in range(1, n + 1):
        divThree = i % 3 == 0
        divFive = i % 5 == 0
        if divThree and divFive:
            sol.append('FizzBuzz')
        elif divThree:
            sol.append('Fizz')
        elif divFive:
            sol.append('Buzz')
        else:
            sol.append(str(i))
    return sol
```

Putting It All Together (FizzBuzz)



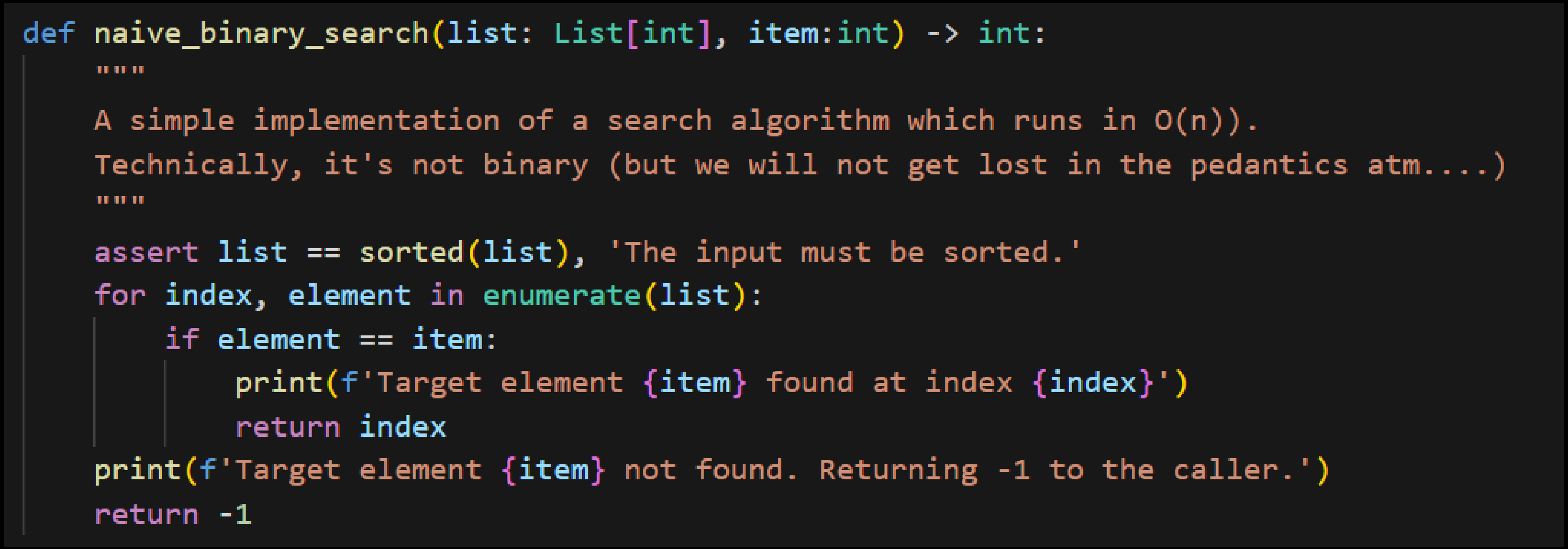
```python
# Solution 2: Less Readable but Less Expensive (No Branching; Pure Arithmetic)
def fizzBuzz2(n: int) -> List[str]:
    """
    Given an integer n, return a string array (1-indexed) where:
        i) answer[i] == 'FizzBuzz' if i is divisible by 3 and 5
        ii) answer[i] == 'Fizz' if i is divisible by 3
        iii) answer[i] == 'Buzz' if i is divisble by 5
        iv) answer[i] == str(i) if none of the conditions are true
    >>> fizzBuzz2(3)
    ['1', '2', 'Fizz']
    >>> fizzBuzz2(5)
    ['1', '2', 'Fizz', '4', 'Buzz']
    >>> fizzBuzz2(15)
    ['1', '2', 'Fizz', '4', 'Buzz', 'Fizz', '7', '8', 'Fizz', 'Buzz', '11', 'Fizz', '13', '14', 'FizzBuzz']
    >>> fizzBuzz2('Tacos')
    Traceback (most recent call last):
    ...
    AssertionError: n must be an integer
    """
    assert isinstance(n, int), 'n must be an integer'
    assert 1 <= n <= 10 ** 4, 'n is out of bounds'
    sol = []
    for i in range(1, n + 1):
        divThree = i % 3 == 0
        divFive = i % 5 == 0
        s = ("Fizz" * (divThree) + "Buzz" * (divFive)) or str(i)
        sol.append(s)
    return sol
```

Putting It All Together (Binary Search)

Assume that we have a sorted list of integer elements.

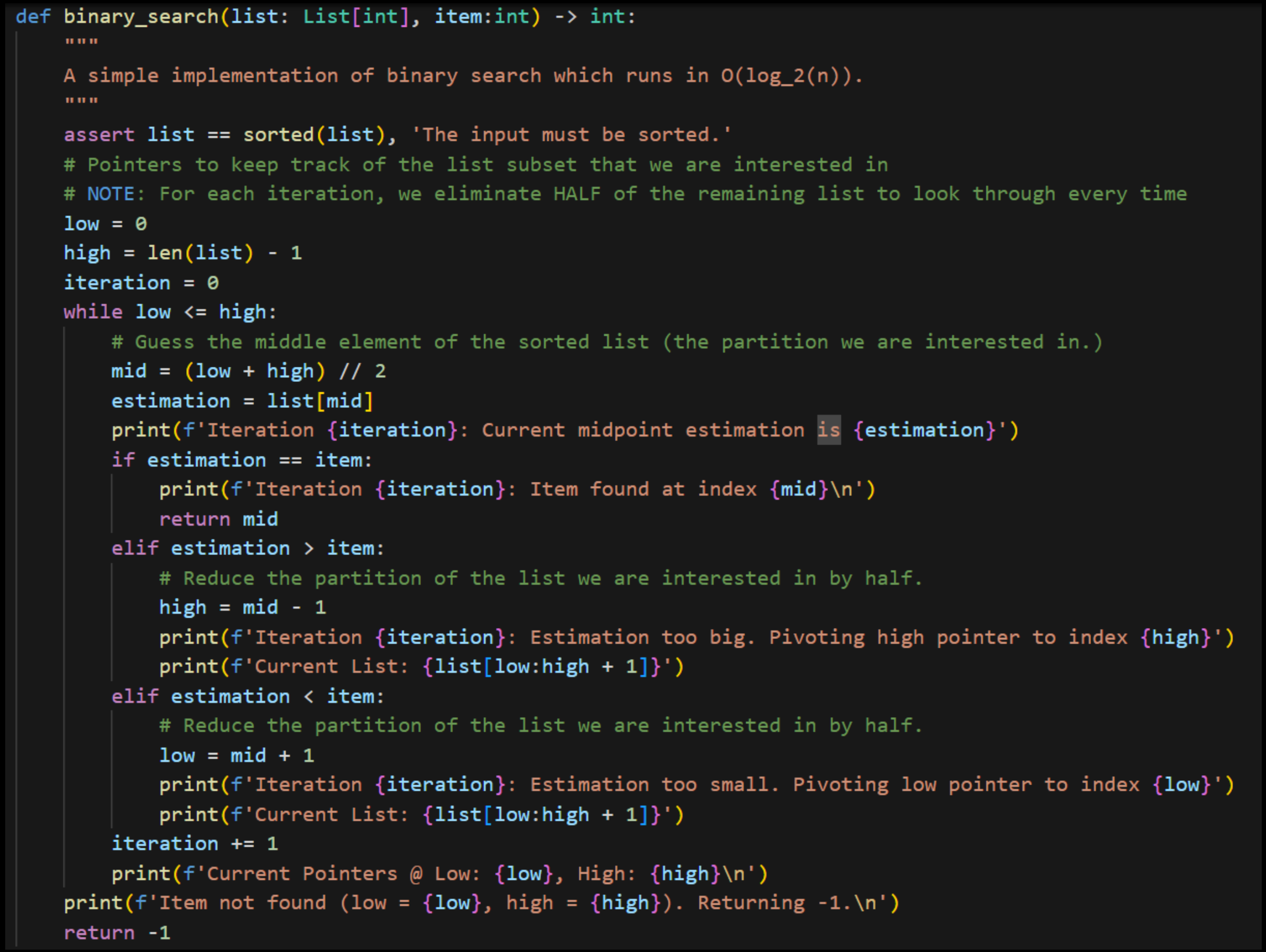
Write a function called $binary\_search(List[int], int) \rightarrow int$ which returns the index of the target element if it exists within the list. Otherwise, return -1.

Naïve Approach



```python
def naive_binary_search(list: List[int], item:int) -> int:
    """
    A simple implementation of a search algorithm which runs in O(n)).
    Technically, it's not binary (but we will not get lost in the pedantics atm....)
    """
    assert list == sorted(list), 'The input must be sorted.'
    for index, element in enumerate(list):
        if element == item:
            print(f'Target element {item} found at index {index}')
            return index
    print(f'Target element {item} not found. Returning -1 to the caller.')
    return -1
```

Optimal Approach



```python
def binary_search(list: List[int], item:int) -> int:
    """
    A simple implementation of binary search which runs in O(log_2(n)).
    """
    assert list == sorted(list), 'The input must be sorted.'
    # Pointers to keep track of the list subset that we are interested in
    # NOTE: For each iteration, we eliminate HALF of the remaining list to look through every time
    low = 0
    high = len(list) - 1
    iteration = 0
    while low <= high:
        # Guess the middle element of the sorted list (the partition we are interested in.)
        mid = (low + high) // 2
        estimation = list[mid]
        print(f'Iteration {iteration}: Current midpoint estimation is {estimation}')
        if estimation == item:
            print(f'Iteration {iteration}: Item found at index {mid}\n')
            return mid
        elif estimation > item:
            # Reduce the partition of the list we are interested in by half.
            high = mid - 1
            print(f'Iteration {iteration}: Estimation too big. Pivoting high pointer to index {high}')
            print(f'Current List: {list[low:high + 1]}')
        elif estimation < item:
            # Reduce the partition of the list we are interested in by half.
            low = mid + 1
            print(f'Iteration {iteration}: Estimation too small. Pivoting low pointer to index {low}')
            print(f'Current List: {list[low:high + 1]}')
        iteration += 1
        print(f'Current Pointers @ Low: {low}, High: {high}\n')
    print(f'Item not found (low = {low}, high = {high}). Returning -1.\n')
    return -1
```

Thank You!

Shogo Toyonaga
Lassonde School of
Engineering