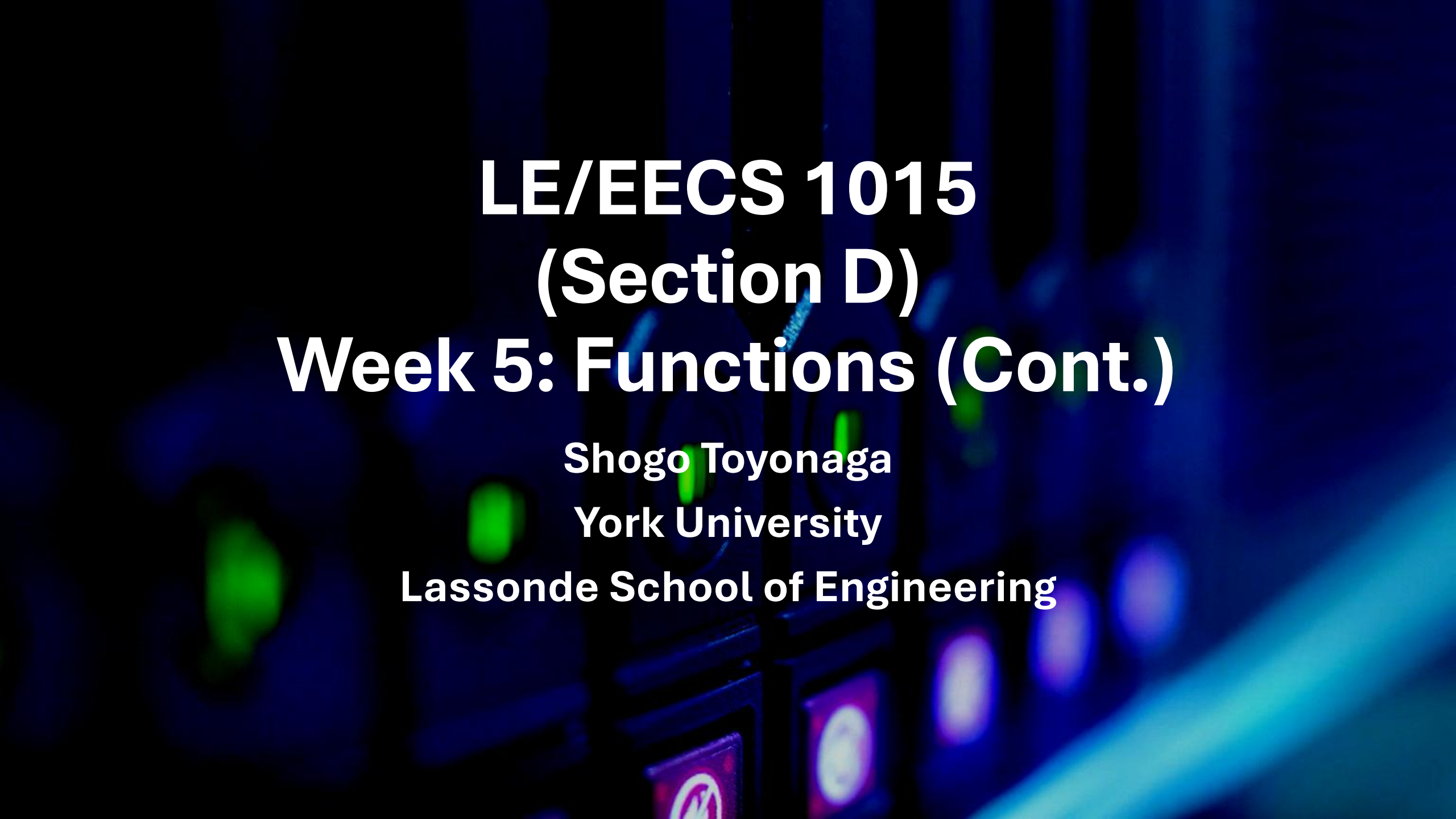

LE/EECS 1015 (Section D) Week 5: Functions (Cont.)

Shogo Toyonaga

York University

Lassonde School of Engineering

This Week...

1. Functions
    - Scoping
    - Memory Model

Goals of Lab 5

1. Familiarization and Debugging with Local & Global Variables
2. Function Scope & Memory Model (ft. Debugger)
3. Function Chaining
4. Test-Driven Development (ft. DocTest)

Lab 5 – What You Do….

Task	Points
Follow the Steps (Circle)	30
Debugging (Global Variable)	30
Implementation (Revenue)	10
Implementation (Competitive Eating)	10
Implementation (Absolute Value)	10
Implementation (Even Numbers)	10

Lab 5 – Useful Resources

- The Call Stack [Python Visualizer]
- A Word on the Usage of Global Variables
- Please refer to the Week 4 Resources for video resources on Test-Driven Development (TDD)

Function Scoping

- The behaviour and accessibility of a program's variables are contingent upon where / how they are defined.

- Local Variables
  - Defined within a function and is only accessible during the function's execution.

- Global Variables
  - Defined outside of all functions; it can be accessed anywhere in the program.

Function Scoping

- Case 1: Unique Local/Global Variable Names
  - If a variable is used but not defined inside of a function, Python automatically searches the global scope.

- Case 2: Repeated Local/Global Variable Names
  - If a variable is defined locally and globally with the same name, the local value will be used in the called function.
  - By default, Python will not allow you to modify a global variable within a function unless you declare it as, “global”, first.
  - Run $help("global")$ in interactive mode for more information

Case 1: Unique Local/Global Variable Names

```python
# Global Variable
ACCELERATION_GRAVITY = 9.8

def force_of_gravity(object_weight: int | float) -> float:
    # Preconditions
    assert isinstance(object_weight, (int,float)), 'data type of object_weight should be int/float'
    assert object_weight >= 0, 'object_weight must be greater than or equal to 0 kg'
    # We use the global variable here (it is not defined in the function)
    return round(object_weight * ACCELERATION_GRAVITY, 2)
```

Case 2: Repeated Local/Global Variable Names

```python
# This is a VERY simplified program (that you should NOT use in actual practice)
# Demonstrates basic use of `global` keyword for a mutex

LOCKED = False

def access_shared_resources(username: str) -> None:
    # Preconditions
    assert isinstance(username, str), 'username must be a string data type'
    # Define the Global Variable (so that we can also modify it in here....)
    global LOCKED
    print(f'Current Lock Status: {LOCKED}')
    if not LOCKED:
        print(f'\t{username}, you now have access to the shared resource.')
        print(f'\t We will set LOCKED to True so that no one else can enter while you are here.')
        LOCKED = True
    else:
        print(f'\t{username}, someone is already accessing the resource; try again later please.')


doctest.testmod(optionflags=doctest.NORMALIZE_WHITESPACE, verbose=True)
```

Good Style for Global Variables

1. If you need to access a global variable in a function, pass it as an argument instead (unless it is a constant)

2. Rather than declaring a global variable in a function, return the value and assign it to a variable in __main__.

3. If you need to modify the value of global variables, let the caller assign the value in __main__.

But wait... Should we always follow these rules....!?

A Word on the Usage of Global Variables

While global variables often have a very negative connotation (e.g., students are taught to avoid using them), arguments can be made for their application:

1. Global variables are required in threaded programs to communicate with each other. That is, shared memory for memory-parallel programming (multi-processor systems) requires their use.

2. We often use the concept of a global variable in member variables for object-oriented programming. Often, we do not pass the member variable as a parameter to a function to make small modifications (e.g., increment)

Function Memory Model

- A Call Stack is used to keep track of function calls
- The Stack Frame maintains the state(s) of local variables
- Global variables are maintained in a separate memory partition
- Your debugger is useful for visualizing the call stack, stack frame, and a view of the global variables! We will do a hands-on live demo soon!

Function Memory Model

- Every time you make a function call, we push it to the Call Stack.

- The program terminates once the Call Stack has popped all the function calls.

Recursion

- Recursion is where a function calls itself. It is used to solve a problem by breaking the input into a, “smaller” instance of itself.
- We push the smaller instances to the Call Stack until we hit the base-case (at which point, we begin popping as the solution can procedurally be computed!)

1. Base Case
    - Under a specific condition, return a discrete value. This is the beginning of where we can compute the final answer!
2. Recursive Case
    - Calls the same method on a smaller instance of the problem.

Recursion: Challenge

The factorial of a non-negative integer n (denoted as $n!$) is the product of all positive integers less than or equal to n.

$$n! = \begin{cases} 1 \; if \; 0 \leq n \leq 1 \; (Base \; Case) \\ n \times (n-1)! \; if \; n > 1 \; (Recursive \; Case) \end{cases}$$

$$Example: 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Try to implement it (and write doctest!)

Recursion: Challenge

The Fibonacci Sequence is a sequence where each element is the sum of the two elements that precede it.

$$fib(n)\begin{cases} 0\ if\ n = 0\ (Base\ Case) \\ 1\ if\ n = 1\ (Base\ Case) \\ fib(n-2) + fib(n-1)\ ifn \geq 2\ (Recursive\ Case) \end{cases}$$

Try to implement it (and write doctest!)

Recursion: Challenge

Given a directory, write a recursive method that returns every path for each file that exists within it.

Hint: You will need to use the operating system (os) module!

Thank You!

Shogo Toyonaga
Lassonde School of Engineering