Lesson Guide

Cybersecurity Professional Program

Introduction to Python for Security

Functions

Course Path

- PY-01: Introduction to Programming
- PY-02: Data Types & Conditions
- PY-03: Loops
- PY-04: File System & Error Handling
- **PY-05: Functions**
- PY-06: Network Communication
- PY-07: Python for Security

Lesson Objectives

This lesson introduces Python functions and code handling. In addition, the concepts of recursion and object-oriented programming (OOP) are introduced and demonstrated.

Lesson Overview

The module focuses on the basics of functions and code handling. Course participants will learn the syntax used to create and call functions and the concepts of recursion, classes, and function implementation.

Learner Level

Learners have basic knowledge of Python programming. The main concepts of this module should be explained carefully to make sure they understand them well, especially recursion and OOP.

Lesson Notes

Make sure learners understand the concept of functions and can apply them in their code. Let them practice on their own after each sub-topic. Carefully cover the bonus recursion and object-oriented programming sections to ensure complete understanding.

Environment & Tools

- Windows, Linux, or macOS
    - PyCharm
    - Python 3

Introduction to Functions

Make sure learners understand each part of a function. This lesson teaches the basics of functions, which is important for the following material.

Why Functions Are Needed

Explain that, for code to be efficient and easily understandable, it must include functions.

What Are Functions?

Emphasize the importance of functions, how they help programmers avoid reusing code, and how they help make overall code management more efficient. Point out that functions can be shared among programmers and programs, thereby simplifying code management.

Function Declaration

Create a function, explain the importance of indentation, and explain the *def* keyword. Encourage learners to create a simple function like the one in the example. In the next slide, we will name it and invoke it.

Create a function that receives two variables and prints them. Invoke the function with and without the variables to show its execution and the error it can generate. The example shows how functions get parameters and print the values. It also shows what happens when we call a function without the number of parameters defined to receive.

Practice:

Ask learners to write a function that gets two integers and prints their sum.

Invoking a Function

Invoke the function for the class and show that the action included in the function is executed. Explain the structure of the function and the code block associated with it. Discuss the syntax used to call a function.

Practice:

Write a function that prints the time whenever you call it (use the ***datetime*** library.)

Default Values
Discuss default values passed to functions and demonstrate how a function with a default value is applied and overridden. In the explanation above, the function is called with and without a value.

Practice:
Create a function that gets ***full_name*** and prints "Hi [full name], welcome to our site." If no value was sent to the function, print "Name is required… Logging off."

Returning Values
The example shows a function that calculates the sum of two variables. Discuss the meaning of the *return* keyword and return the result. Explain that the ***return*** command must be the last to execute in the logic flow.

Practice:
Write a function that gets two strings (***name***, ***email***). The function will use string concatenation and return a ***string var***, as follows: "Welcome [name], your email address is [email]." Then print the string outside the function's scope.

Lab PY-05-L1: Calculator
Learners should complete this lab in class.

Returning Data
Talk about the need to return multiple values and how that is done, using commas to separate them. The code in the example will print both the first name and last name returned from the function. Note that it will return as a **tuple**. Mention the option of returning a list of items from a function.

Practice:
Create a function that gets a number and creates a list from 0 to the provided number. Save it in a variable and return the variable. Print the list in the main scope. Talk about what a dictionary is and let learners create a dictionary. Return the entire dictionary from a function and print it to show how the dictionary is returned. We can create the dictionary in the function and return it using curly brackets or as a variable.

Practice:
Create a function that generates a dictionary of three names and ages. Return the dictionary and print it in the main scope.

Complete Function Example
The function shown in the example receives two numbers, adds them together, returns the result, and prints it. Let learners create the function on their own.

Practice:
Write a function that gets a number that represents the number of learners. The function uses the number to ask for each learner's name in a loop. The function creates a list and returns the string ***Done***.

None
The example shows a case in which a function returns ***None***. In the example, the function does not include the *return* keyword. Explain that ***None*** is a data type that does not represent 0, false, or an empty string. It is a reserved word in Python that refers to an **empty value**. Let the class experiment with attribute manipulation in their functions.

Practice:
1. Work with the script in the slide.
2. Create a variable for a function that does not return a value.
3. Print the variable in the main scope and check the output.

Lab PY-05-L2: Returning Lists
Learners should complete this lab in class.

Code Handling

Discuss the correct way to organize code and break down each aspect of proper code management.

Scope

The area of the program to which a particular variable applies is called a **scope**. Emphasize the difference between ***local*** and ***global*** scopes. Note that in the example, the ***txt var*** is not recognized in the global scope. **Note:** It is important to explain the hierarchy from global scope down to local scope.

Global Scope

Talk about the importance of global scope. Different functions can work with the same variable defined in the global scope. **Note:** A global variable can be viewed and used in a local scope but cannot be changed.

Practice:

Create two variables with a company name and ID. Call a function that makes a dictionary from the two variables and returns it. Print the dictionary in the main scope.

Global Keyword

Specify the need for a global keyword, which allows a function to modify global variables. Without a global keyword, functions can only identify global variables but not modify them. **Note:** Once a global variable is used in a local scope, functions can use the variable and modify it, which means global keywords can be exploited.

Practice:

Create two inputs, ***name*** and ***last name***. Create a function that gets two variables and changes the ***name var*** and ***last var*** to first letters only. For example, *John* will be modified to *J*. Then, print the variables in the global scope and note the change.

__*name*__ Variable

Various modules have main functions that run when the code is executed. We would not want one of the main functions to be executed when importing external code. We, therefore, define a magic main in the code so that only it will be executed and not any other main function from an external source.

Practice:
Create two Python files, ***main*** and ***util***. Define in both a ***main()*** function, whereby each function prints the name of the file it executes from. Define a magic main in the main file and the util file. Examine the results.

Proper Code Management
Emphasize how proper code management enables easier code reading and helps other programmers understand its meaning and structure. **Note:** It is important to explain that proper code management is not a rule for coding. It refers to a structure that makes the code easier to understand. The code will run properly, even if a programmer does not follow the example shown on the slide.

Lab PY-05-L3: Scope Behavior
Learners should complete this lab in class.

Lab PY-05-L4: Main Identification
Learners should complete this lab in class.

Lab PY-05-L5: Bullseye
Learners should complete this lab in class.

TDX Arena Challenge: Troll
TDX Arena challenges can be done at home for extra practice. Make sure learners remember how to log in. The TDX Arena guide is provided in the course's LMS shell.

TDX Arena Challenge: Blocks
TDX Arena challenges can be done at home for extra practice. Make sure learners remember how to log in. The TDX Arena guide is provided in the course's LMS shell.

Recursion

You should teach this topic only after covering all the previous topics. Make sure to explain this topic slowly and thoroughly. You can repeat the material on this topic as many times as necessary until you feel the learners understand it well enough.

Recursion

Give a few examples that demonstrate the need for recursion, such as presenting all the numbers in a list, extracting all text files from a folder containing subdirectories, or opening a box of candies with other boxes.

Recursion Implementation

Let learners create a recursion like the one in the example to follow your explanation and demonstration. In the example, the function prints an asterisk multiplied by the variable count. The result is a triangle made from the asterisks.

Lab PY-05-L6: Recursive Search

Learners should complete this lab in class.

Lab PY-05-L7: Directory Listing

Learners should complete this lab in class.

Object-Oriented Programming (Extra)

This part is extra, and you should teach it only after you have covered all the previous topics. Point out the need to work with OOP before demonstrating it. Let learners ask questions and work on the code together with you.

Object-Oriented Programming

Explain the need for OOP. Provide several examples of objects and the data they may contain, such as people, books, roads, signs, etc. It is much easier to implement changes when the code is modular without changing the entire program.

Practice:

All cars have wheels, a size, and a color. However, the color, size, and wheels may differ from one car to another.

Class

Explain the difference between ***class*** and ***object*** and give examples of each. **Note:** Because each class represents a unique category, applications require many different classes to function.

Class

Each class represents a single category. For example, a class could be *cars*. An SUV, truck, sedan, and van are examples of objects associated with the *car's* class.

Object

Attributes describe the object, while **methods** describe what the object can do.

Defining a Class

Let learners define a class and write an ***init*** function. Explain ***self*** and how many objects can be controlled using the same function. Mention the following:

- The keyword *class* is used to create a class, followed by the name of the class and a colon.
- ***__init__*** is a reserved method in Python classes. It is known as a *constructor*.
- ***self*** represents the instance of a class.
- On the next slide, we will create a class and call it.

Object Creation
Explain that the code in the example shows two different objects of the same class. Please point out the differences between them. **Note:** When defining a class with ***init*** and variables**,** the variables must get values when creating the object. For example, an error will occur if a learner creates an object called a **Car** without defining its price.

Practice:
Create a class called *Animal* with an ***init*** function that initializes its name, color, age, and speech type (a dog barks). Create a loop that asks the user the following:

- How many animals do we have?
- What are their names, colors, ages, and speech types?
- List the animals.

Extra Practice:
Save the details to a file. For this exercise, remember to tell learners to use what they learned so far about implementation, functions, loops, and proper code management.

Lab PY-05-L8: Car Creation
This lab should be assigned as homework.

TDX Arena

Learners will need to navigate to the TDX Arena practice arena as described in the slide deck.

Homework Lab Assignment: Recursions

This assignment asks the learner to use the TDX Arena platform to better understand recursion within Python's context.

To complete the assignment, learners will need to visit the outlined TDX Arena lab.

This TDX Arena lab will be completed as a homework assignment before the next class.