secure scripting (python)

**(SesPython)**

This module focuses on basic scripting using Python. It focuses on the basic constructs and useful commands such as conditional, looping, and logical constructs as well as variables and user and system interfaces. Students create basic scripts and have a foundation upon which to learn more advanced Python scripting. One of the exercises is to implement a simple file system scanner, so the students will practice scripting security applications and administrative scripts by invoking Linux commands through Python.

# learning objectives

Primary Learning Objective

Students will understand how to approach a problem to solve it using scripting.   
Students will understand bottom-up programming techniques suitable for use in scripting.  
Students will demonstrate proficiency in scripting by using elements of Python, and of scripting languages for various Linux tools.  
  
Associated Sub Learning Objectives  
NSA/DHS CAE Knowledge Units for Basic Scriptingand Programming (BSP)

**Topics**

1. Implement basic security concepts
   1. Permissions, bounds checking, input validation, type checking and parameter validation
2. Be familiar with the concept and basic implementation of regular expressions.
3. Understand basic data structures and algorithms
4. Basic Boolean logic/operations.
   1. AND / OR / XOR / NOT
5. Scripting on both Windows and Linux
   1. Language (e.g. PERL, Python, BASH, JAVA, VB Scripting, Powershell)
6. Properly apply basic programming constructs and concepts including:
   1. Variables and types (int, float, char, etc.)
   2. Strings, arrays, structures
   3. Sequential and parallel execution
   4. Assignments (:=, =, ++, --, etc.)
   5. Decisions and branching (if, if ... else, elseif, switch, case, etc.)
   6. Loops (for, while, repeat, etc.)
   7. Functions, procedures, and calls
   8. Debugging techniques

**Outcome:** 1. Demonstrate their proficiency in the use of scripting languages to write simple scripts (e.g., to automate system administration tasks).

**Outcome:** 2. Write simple linear and looping scripts.

**Outcome:** 3. Write simple and compound conditions within a programming language or similar environment (e.g., scripts, macros, SQL).

**Outcome:** 4. Demonstrate proficiency in the use of a programming language to solve complex problems in a sdecure and robust manner.

(This module introduces simple but not compound conditions, and it does not cover macros or SQL.)

AP Computer Science Principles Course, Big Idea 1: Creativity

**L.O. 1.1.1** Apply a creative development process when creating computational artifacts.  
 **EK 1.1.1A** A creative process in the development of a computational   
 artifact can include, but is not limited to, employing nontraditional,   
 nonprescribed techniques; the use of novel combinations of artifacts, tools,   
 and techniques; and the exploration of personal curiosities.

**EK 1.1.1B** Creating computational artifacts employs an iterative and often   
 exploratory process to translate ideas into tangible form.  
**L.O. 1.2.2** Create a computational artifact using computing tools and techniques to solve a problem.  
 **EK 1.2.2A** Computing tools and techniques can enhance the process of   
 finding a solution to a problem.

**EK 1.2.2B** A creative development process for creating computational artifacts can be used to solve problems when traditional or prescribed computing techniques are not effective.

**L.O. 1.2.3** Create a new computational artifact by combining or modifying existing artifacts.  
 **EK 1.2.3A** Creating computational artifacts can be done by combining and   
 modifying existing artifacts or by creating new artifacts.

**EK 1.2.3B** Computation facilitates the creation and modification of computational artifacts with enhanced detail and precision.

**EK 1.2.3C** Combining or modifying existing artifacts can show personal   
 expression of ideas.

AP Computer Science Principles Course, Big Idea 2: Abstraction

**L.O. 2.1.1** Describe the variety of abstractions used to represent data.  
 **EK 2.1.1A** Digital data is represented by abstractions at different levels.  
 **EK 2.1.1B** At the lowest level, all digital data are represented by bits.

**EK 2.1.1C** At a higher level, bits are grouped to represent abstractions, including but not limited to numbers, characters, and color.

**EK 2.1.1D** Number bases, including binary, decimal, and hexadecimal, are used to represent and investigate digital data.

**EK 2.1.1E** At one of the lowest levels of abstraction, digital data is represented in binary (base 2) using only combinations of the digits zero and one. EXCLUSION STATEMENT (for EK 2.1.1E): Two’s complement conversions are beyond the scope of this course and the AP Exam.

**L.O. 2.2.2** Use multiple levels of abstraction to write programs.  
 **EK. 2.2.2A** Software is developed using multiple levels of abstractions,   
 such as constants, expressions, statements, and procedures, and libraries.   
 **EK 2.2.2B** Being aware of and using multiple levels of abstractions in   
 developing programs help to more effectively apply available resources and   
 tools to solve problems.

**L.O. 2.2.3** Identify multiple levels of abstractions that are used when writing programs.

**EK 2.2.3A** Different programming languages offer different levels of   
 abstraction. EXCLUSION STATEMENT (for EK 2.2.3A): Knowledge of the   
 abstraction capabilities of all programming languages is beyond the scope   
 of this course and the AP Exam.

**EK 2.2.3B** High-level programming languages provide more abstractions for the programmer and make it easier for people to read and write a program.

**EK 2.2.3C** Code in a programming language is often translated into code in another (lower-level) language to be executed on a computer.

**EK 2.2.3D** In an abstraction hierarchy, higher levels of abstraction (the most general concepts) would be placed toward the top and lower-level abstractions (the more specific concepts) toward the bottom.

AP Computer Science Principles Course, Big Idea 3: Data and Information

**L.O. 3.3.1** Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information.  
 **EK 3.3.1A** Digital data representations involve trade-offs related to   
 storage, security, and privacy concerns.

**EK 3.3.1B** Security concerns engender trade-offs in storing and transmitting information.

**EK 3.3.1C** There are trade-offs in using lossy and lossless compression techniques for storing and transmitting data.

**EK 3.3.1G** Data is stored in many formats depending on its characteristics (e.g., size and intended use).

**EK 3.3.1I** Reading data and updating data have different storage requirements

AP Computer Science Principles Course, Big Idea 4: Algorithms

**L.O. 4.1.1** Develop an algorithm for implementation in a program.  
 **EK 4.1.1A** Sequencing, selection, and iteration are building blocks of   
 algorithms.

**EK 4.1.1B** Sequencing is the application of each step of an algorithm in the order in which the statements are given.

**EK 4.1.1C** Selection uses a Boolean condition to determine which of two parts of an algorithm is used.

**EK 4.1.1D** Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.

**EK 4.1.1E** Algorithms can be combined to make new algorithms.

**EK 4.1.1F** Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct.

**EK 4.1.1G** Knowledge of standard algorithms can help in constructing new algorithms.

**EK 4.1.1H** Different algorithms can be developed to solve the same problem.

**EK 4.1.1I** Developing a new algorithm to solve a problem can yield insight into the problem.

**L.O. 4.1.2** Express an algorithm in a language.  
 **EK 4.1.2A** Languages for algorithms include natural language, pseudocode,   
 and visual and textual programming languages.

**EK 4.1.2B** Natural language and pseudocode describe algorithms so that humans can understand them.

**EK 4.1.2C** Algorithms described in programming languages can be executed on a computer.

**EK 4.1.2D** Different languages are better suited for expressing

different algorithms.

**EK 4.1.2E** Some programming languages are designed for specific domains and are better for expressing algorithms in those domains.

**EK 4.1.2F** The language used to express an algorithm can affect characteristics such as clarity or readability but not whether an algorithmic solution exists.

**EK 4.1.2G** Every algorithm can be constructed using only sequencing, selection, and iteration.

**EK 4.1.2H** Nearly all programming languages are equivalent in terms of being able to express any algorithm.

**EK 4.1.2I** Clarity and readability are important considerations when expressing an algorithm in a language.

AP Computer Science Principles Course, Big Idea 5 Programming

**L.O. 5.1.2** Develop a correct program to solve problems.  
 **EK 5.1.2A** An iterative process of program development helps in   
 developing a correct program to solve problems.

**EK 5.1.2B** Developing correct program components and then combining them helps in creating correct programs.

**EK 5.1.2C** Incrementally adding tested program segments to correct working programs helps create large correct programs.

**EK 5.1.2D** Program documentation helps programmers develop and maintain correct programs to efficiently solve problems.

**EK 5.1.2E** Documentation about program components, such as code segments and procedures, helps in developing and maintaining programs.

**EK 5.1.2F** Documentation helps in developing and maintaining programs when working individually or in collaborative programming environments.

**EK 5.1.2G** Program development includes identifying programmer and user concerns that affect the solution to problems.

**EK 5.1.2H** Consultation and communication with program users is an important aspect of program development to solve problems.

**EK 5.1.2I** A programmer’s knowledge and skill affects how a program is developed and how it is used to solve a problem.

**EK 5.1.2J** A programmer designs, implements, tests, debugs, and maintains programs when solving problems.

**L.O. 5.4.1** Evaluate the correctness of a program.  
 **EK 5.4.1A** Program style can affect the determination of program   
 correctness.

**EK 5.4.1B** Duplicated code can make it harder to reason about a program.

**EK 5.4.1C** Meaningful names for variables and procedures help people better understand programs.

**EK 5.4.1D** Longer code segments are harder to reason about than shorter code segments in a program.

**EK 5.4.1E** Locating and correcting errors in a program is called debugging the program.

**EK 5.4.1F** Knowledge of what a program is supposed to do is required in order to find most program errors.

**EK 5.4.1G** Examples of intended behavior on specific inputs help people understand what a program is supposed to do.

**EK 5.4.1H** Visual displays (or different modalities) of program state can help in finding errors.

**EK 5.4.1I** Programmers justify and explain a program’s correctness.

**EK 5.4.1J** Justification can include a written explanation about how a program meets its specifications.  
**EK 5.4.1K** Correctness of a program depends on correctness of program components, including code segments and procedures.

**EK 5.4.1L** An explanation of a program helps people understand the functionality and purpose of it.

**EK 5.4.1M** The functionality of a program is often described by how a user interacts with it.

**EK 5.4.1N** The functionality of a program is best described at a high level by what the program does, not at the lower level of how the program statements work to accomplish this

**L.O. 5.5.1** Employ appropriate mathematical and logical concepts in programming.  
 **EK 5.5.1A** Numbers and numerical concepts are fundamental to   
 programming.

**EK 5.5.1B** Integers may be constrained in the maximum and minimum values that can be represented in a program because of storage limitations. EXCLUSION STATEMENT (for EK 5.5.1B): Specific range limitations of all programming languages are beyond the scope of this course and the AP Exam.

**EK 5.5.1C** Real numbers are approximated by floating-point representations that do not necessarily have infinite precision. EXCLUSION STATEMENT (for EK 5.5.1C): Specific sets of values that cannot be exactly represented by floating-point numbers are beyond the scope of this course and the AP Exam.

**EK 5.5.1D** Mathematical expressions using arithmetic operators are part of most programming languages.

**EK 5.5.1E** Logical concepts and Boolean algebra are fundamental to programming.

**EK 5.5.1F** Compound expressions using and, or, and not are part of most programming languages.

**EK 5.5.1G** Intuitive and formal reasoning about program components using Boolean concepts helps in developing correct programs.

**EK 5.5.1H** Computational methods may use lists and collections to solve problems.

**EK 5.5.1I** Lists and other collections can be treated as abstract data types (ADTs) in developing programs.

**EK 5.5.1J** Basic operations on collections include adding elements, removing elements, iterating over all elements, and determining whether an element is in a collection

ACM/IEEE-CS Computer Science Curricula 2013: Software Development Fundamentals (SDF)

**SDF/AD Topic** Problem-solving strategies.

**SDF/AD Topic** Fundamental design concepts and principles.

**SDF/AD LO 3** Create algorithms for solving simple problems

**SDF/AD LO 4** Use a programming language to implement, test, and debug algorithms for solving simple problems.

**SDF/AD LO 8** Apply the techniques of decomposition to break a program into smaller pieces.

**SDF/FPC Topic** Basic syntax and semantics of a higher-level language.

**SDF/FPC Topic** Variables and primitive data types.

**SDF/FPC Topic** Expressions and assignments.

**SDF/FPC Topic** Simple I/O including file I/O.

**SDF/FPC Topic** Conditional and iterative structures.

**SDF/FPC LO 3** Write programs that use primitive data types.

**SDF/FPC LO 4** Modify and expand short programs that use standard conditional and iterative control structures and functions.

**SDF/FPC LO 5** Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures.

**SDF/FPC LO 6** Write a program that uses file I/O to provide persistence across multiple executions.

**SDF/FPC LO 7** Choose appropriate conditional and iteration constructs for a given programming task.

**SDF/DM Topic** Program correctness, defensive programming.

**SDF/DM Topic** Simple refactoring.

**SDF/DM LO 3** Identify common coding errors that lead to insecure programs (failure to check inputs) and apply strategies for avoiding such.

# module details

**Recommended Time:** 8–9 contact hours

**Prerequisite Knowledge:** Basic knowledge of programming (variables, loops, conditionals); knowledge of a Linux or other UNIX-like system including pipes and input and output redirection. In particular, students should be able to read a manual page with guidance. Instructors should conduct a pre-assessment to verify that students have the above background knowledge. If students do not know these concepts and have no experience with them, much of the module will be more difficult than intended.

**Pilot-tested in the Following Courses:** ITSE 1350 Intro to Scripting Languages, System Analysis and Design; CSC 240, Introduction to Different Programming Languages; NTWK 2013, Introduction to Networking; CIS 215, Operating Systems; CSC 200, Introduction to Computer Science.

**Lab Environment:** The students should have access to the contents of the “data” directories in Units 1 and 3, the "sample" and "example-scripts" directories in Unit 2, and the Lab file in all three units. They will also need access to a Windows and/or Linux system, preferably Linux Fedora 22. They will not need Internet or network connectivity once the directories and Laboratory Exercise files are downloaded.

For additional guidance, here are the specific programs that each unit uses. The system used must have them. All are standard, with the possible exception of *shasum*, which produces a cryptographic checksum. If that program is not available, you can replace the occurrence of

shasum $1

with

sum $1 | awk '{ print $1, $3 }'

• Unit 1 uses the commands *bash* (1) or *sh* (1), *grep* (1), *sed* (1), and *awk* (1).

• Unit 2 uses the commands *bash* (1) or *sh* (1), *ls* (1), *shasum* (1), *diff* (1), *rm* (1), *sort* (1), and *uniq* (1).

• Unit 3 uses the commands *bash* (1) or *sh* (1), *date* (1), *expr* (1), *sed* (1), and *tr* (1).

**Homework:** Labs are included in each unit. Lab exercises can be started as a class, during the presentations, and completed as homework assignments. Many of the lab exercises involve writing and later revising Python scripts. It is recommended that students save each script as a separate file, using the suggested filename provided.

Instructors can check students’ scripts either by running them to test their outputs or by comparing them to the correct answer scripts provided in the directories that end with “ … \_LabScripts” and “… \_LabSolutions.” In the directories ending with “ … \_LabScripts” are versions of the scripts that have been created as plain text files and usually have the filename extension “.py”; docx versions of the same files, which may be easier for instructors to manage, are in the directory “… \_LabSolutions.”

**Instructional Files and Online Resources Needed:** PowerPoint slides and data files are included for each unit.

**Assessment:** See the assessment guide for this unit, “19.SeSPython\_AssessmentGuide.docx.”

Files containing scripts that are the correct answers to assessment questions are provided in the directory 20.SeSPython\_AssessmentAnswerFiles.docx. Each script is available in either plain text (“.py”) format or PDF format, so instructors can use whichever file format they find easier to manage.

### Unit 1. The Basics

**Presentation:** 02.SeSPython\_Unit1\_TheBasics\_Presentation.pptx

To demonstrate the examples included in the presentation, you will need the following data files from the directory  
 03.SeSPython\_Unit1\_TheBasics\_DemoScripts:

* abcscript.py — Use for the example shown on Slides 6–7.
* hello.py — Use for example shown on Slide 13
* var2.py — Use for example shown on Slide 14
* nameQuestion.py — Use for example shown on Slide 17
* nameQuestionCheck.py — Use for example shown on Slide 20
* numbersQuestion.py — Use for example shown on Slide 22
* dict.txt — This file contains a list of words, one per line. It’s first used starting on Slide 23.
* argsProcess.py — Use for example shown on Slide 24
* argsExample.py — This script is introduced on Slide 28
* stringSearch.py — Use for example shown on Slide 30
* stringSearchMessage.py — Use for example shown on Slide 34

Files in the directory 04.SeSPython\_Unit1\_TheBasics\_DataFiles  
Part 1 are used in examples throughout the presentation.

If you want the students to try the exercises themselves, copy the files to the students’ systems or make available for them to download.

**Lab:** 05.SeSPython\_Unit1\_TheBasics\_Lab.docx

Students will need the following data files from the directory 06.SeSPython\_Unit1\_TheBasics\_LabScripts. These files can be copied to the students’ systems or made available for them to download.

* dict.txt
* mycat.py
* x
* y
* x y

Solutions to lab questions are located in 07.SeSPython\_Unit1\_TheBasics\_LabSolutions.docx.

**Learning Objectives:**

Upon completion of Unit 1, students will be able to:

1.1 Analyze a problem and develop a script to solve it.

1.2 Create and execute a Python script.

1.3 Use variables in the script.

1.4 Use command-line arguments in the script.

1.5 Use conditional (*if,* *elif,* *else*) statements to test for various conditions and act accordingly.

1.6 Perform basic error checking in the script.

### Unit 2. Advanced Control

**Presentation:** 07.SeSPython\_Unit2\_AdvancedControl\_Presentation.pptx

Files in the directory 09.SeSPython\_Unit2\_AdvancedControl\_DemoScripts are used in the examples.

* for1.py — This script is an example of a basic *for* loop (see Slide 16)
* for2.py — This script is an example of a *continue* statement (see Slide 18)
* processFileContent.py — This script is an example of a basic file process (see Slide 22)
* checkForOccurrence.py — This script is an example of a basic file process (see Slide 23)
* myCat.py — This script is an example of file processing and conditional statements (see Slide 26)
* forFilesProcessing.py — This script demonstrates an iterative process to validate existence of files in a working directory (see Slide 30)
* combining.py — This script demonstrates subprocess invocation (see Slide 33)
* functionExampl1.py — This script demonstrates function definition and function call (see Slide 37)
* functionExampl2.py — This script demonstrates function definition and function call (see Slide 38)
* functionExampl3.py — This script demonstrates function definition and function call (see Slide 39)
* function.py — This script is introduced as a demonstration of combining functions to wrap subprocess invocation (see Slide 40)
* bool1.py, bool2.py — These demonstrate various Boolean operator (see Slides 46–47)
* testTemp.py — Demonstration of temporary files (see slide 50)
* x1, x2 — These two files containing very similar content are used to demonstrate the output from *diff*. They are referenced on Slide 52.

Files in the directory 10.SeSPython\_Unit2\_AdvancedControl\_DataFiles  
Part II are used in examples throughout the presentation

**Lab:** 11.SeSPython\_Unit2\_AdvancedControl\_Lab.docx

Students will need a copy of the directory “sample”. It can be copied to the students’ systems or put out for them to download. Please do not change anything in that directory – one of the tests the students run depends on those files being unchanged.

* abc xyz — This file is empty. Note the blank space; that is part of the filename.
* abcde — This file is also empty.
* demofor.py, demofor2.py, demofor3.py — These scripts demonstrate various things about *for* loops.

Solutions to lab questions are located in 12.SeSPython\_Unit2\_AdvancedControl\_LabScripts.  
13.SeSPython\_Unit2\_AdvancedControl\_LabSolutions.docx

**Learning Objectives:**

Upon completion of Unit 2, students will be able to:

2.1 Merge two or more scripts into one that performs the same functions.

2.2 Use *for* loops in the script.

2.3 Use variables in the script.   
2.4 Students will be able to do simple arithmetic in the script.

2.5 Use Boolean operators to test conditions in the script.  
2.6 Read, write, execute files  
2.7 Define and call functions

2.8 Process options given to the script.

2.9 Perform basic error checking in the script.

2.10 Invoke subprocess from Linux inside the script

2.11 Process output obtained from routines called by other process

2.12 Use defined functions as a way to invoke subprocess

**Suggestion**  
If you want to split this unit into two parts, we recommend using Slides 1 through 24 as the first part; use Slide 3 then Slides 25 through 60 as the second part. There is a natural breaking point after Slide 24, because the class has finished the loops and done an exercise. You should begin the second part with Slide 25, though, to remind students of where the class left off. Also, if you split this unit, we strongly recommend briefly beginning the second part with Slide 3 to remind students of the problem being solved.

### Unit 3. Advanced Scripting

**Presentation:** 14.SesPython\_Unit3\_AdvancedScripting\_Presentation.pptx

Files in the directory 15.SeSPython\_Unit3\_AdvancedScripting\_DemoScripts are used in examples during the presentation.

• connect.csv — This is a comma-separated value (CSV) version of the spreadsheet used in Secure Scripting Module. Located in 19.SeS\_Unit3\_AdvancedScripting\_DataFiles

* replacingExample1.py — Script that demonstrate replace functionality (see Slide 6 and 7)
* whileExample1.py, whileExample2.py — Scripts demonstrating basic while loop mechanism and file processing (see slides 9 and 10)
* gleep — This is used to demonstrate reading fields from a file. It is referenced on Slide 11
* cvsHandler.py — script used to demonstrate CSV file (see slide 13)
* datePython — script demonstrates the date function in Python (see slide 18)
* skippingLines.py, countLines.py — scripts that demonstrate looping mechanisms along with arithmetic operations to process the lines of the file (see slide 19)
* checkPhoneFormat.py — script demonstrates regular expression mechanisms to validate a given pattern matching (see slide 25)

**Lab:** 16.SeSPython\_Unit3\_AdvancedScripting\_Lab.docx

Students will need the following data file from the directory 17.SeSPython\_Unit3\_AdvancedScripting\_LabScripts. This file can be copied to the students’ systems or made available for them to download.

* connect.csv — This is a comma-separated value (CSV) version of the spreadsheet used Secure Scripting Module located in 19.SeS\_Unit3\_AdvancedScripting\_DataFiles

Solutions to the lab are located in  
18.SeSPython\_Unit3\_AdvancedScripting\_LabSolutions.docx

**Learning Objectives:**

Upon completion of Unit 3, students will be able to:

3.1 Analyze data in a file using a script.

3.2 Use *while* loops in the script.

3.3 Do simple arithmetic in the script.

3.4 Edit values of variables and data.

3.5 Perform pattern-matching.

**Note:** The material in Unit 3 includes the use of regular expressions for pattern-matching in programs such as *expr* (1) and *sed* (1). If students have never seen this, the instructor should be prepared to expand on what is in the slides. This may increase the amount of time needed to cover this unit.

**Suggestion**

If you want to split this module into two parts, we recommend using Slides 1 through 14 as the first part and Slides 15 through 34 as the second part. There is a natural breaking point after Slide 15, because the class has finished counting and the *while* loop and done two exercises on the subject. The second part begins with editing variable values, which is different enough that a break here makes sense.