PCAP – Certified Associate in Python Programming Exam Preparatory

(Exam Code: PCAP-31-03)

Overview

In this PCAP – Certified Associate in Python Programming Exam Preparatory course, you will learn how to accomplish coding tasks related to the basics of programming in the Python language and the fundamental notions and techniques used in object-oriented programming.

In this course, you will learn to become familiar with general computer programming concepts like conditional execution, loops, Python programming language syntax, semantics, and the runtime environment, as well as with general coding techniques and object-oriented programming.

This course will help you prepare for the PCAP examination. Becoming PCAP certified ensures that you are fully acquainted with all the primary means provided by Python 3 to enable you to start your own studies, and to open a path to a developer's career.

Prerequisites

There are no formal prerequisites for this course. However, you need to have basic computer literacy. Previous experience working with any programming language or basic python knowledge is recommended.

Objectives

After completing this course, you should be able to demonstrate the sufficient knowledge of the following concepts:

1. The fundamentals of computer programming, i.e. how the computer works, how the program is executed, how the programming language is defined and

constructed, what the difference is between compilation and interpretation, what Python is, how it is positioned among other programming languages, and what distinguishes the different versions of Python;

2. The basic methods of formatting and outputting data offered by Python, together with the primary kinds of data and numerical operators, their mutual relations and bindings; the concept of variables and variable naming conventions; the assignment operator, the rules governing the building of expressions; the inputting and converting of data;

3. Boolean values to compare difference values and control the execution paths using the if and if-else instructions; the utilization of loops (while and for) and how to control their behavior using the break and continue instructions; the difference between logical and bitwise operations; the concept of lists and list processing, including the iteration provided by the for loop, and slicing; the idea of multi-dimensional arrays;

4. The defining and using of functions – their rationale, purpose, conventions, and traps; the concept of passing arguments in different ways and setting their default values, along with the mechanisms of returning the function's results; name scope issues; new data aggregates: tuples and dictionaries, and their role in data processing;

5. Python modules: their rationale, function, how to import them in different ways, and present the content of some standard modules provided by Python; the way in which modules are coupled together to make packages; the concept of an exception and Python's implementation of exceptions, including the try-except instruction, with its applications, and the raise instruction; strings and their specific methods, together with their similarities and differences compared to lists;

6. The fundamentals of OOP (Object Oriented Programming) and the way they are adopted in Python, showing the difference between OOP and the classical, procedural approach; the standard objective features: inheritance, abstraction, encapsulation, and polymorphism, along with Python-specific issues like instance vs. class variables, and Python's implementation of inheritance; objective nature of exceptions; Python's generators (the yield instruction) and closures (the lambda keyword); the means Python developers can use to process (create, read, and write) files.

Course Outlines

Module 1: Modules and Packages

Objectives covered in this module
• import variants; advanced qualifiying for nested modules
• dir(); sys.path variable
• math: ceil(), floor(), trunc(), factorial(), hypot(), sqrt(); random: random(), seed(), choice(), sample()
• platform: platform(), machine(), processor(), system(), version(), python_implementation(), python_version_tuple()
• idea, __pycache__, __name__, public variables, __init__.py
• searching for modules/packages; nested packages vs directory tree

Module 2: Exceptions

Objectives covered in this module
• except, except:-except; except:-else:, except (e1,e2)
• the hierarchy of exceptions
• raise, raise ex, assert
• event classes, except E as e, arg property
• self-defined exceptions, defining and using

Module 3: Strings

Objectives covered in this module
• ASCII, UNICODE, UTF-8, codepoints, escape sequences
• ord(), chr(), literals
• indexing, slicing, immutability
• iterating through,
• concatenating, multiplying, comparing (against strings and numbers)
• in, not in
• .isxxx(), .join(), .split()
• .sort(), sorted(), .index(), .find(), .rfind()

Module 4: Object-Oriented Programming

Objectives covered in this module
• ideas: class, object, property, method, encapsulation, inheritance, grammar vs class, superclass, subclass
• instance vs class variables: declaring, initializing

• __dict__ property (objects vs classes)
• private components (instance vs classes), name mangling
• methods: declaring, using, self parameter
• instrospection: hasattr() (objects vs classes), __name__, __module__, __bases__ properties
• inheritance: single, multiple, isinstance(), overriding, not is and is operators
• inheritance: single, multiple, isinstance(), overriding, not is and is operators
• constructors: declaring and invoking
• polymorphism
• __name__, __module__, __bases__ properties, __str__() method
• multiple inheritance, diamonds

Module 5: Miscellaneous (List Comprehensions, Lambdas, Closures, and I/O Operations)

Objectives covered in this module
• list comprehension: if operator, using list comprehensions
• lambdas: defining and using lambdas, self-defined functions taking lambda as as arguments; map(), filter();
• closures: meaning, defining, and using closures
• I/O Operations: I/O modes, predefined streams, handles; text/binary modes open(), errno and its values; close() .read(), .write(), .readline(); readlines() (along with bytearray())