PRG404 : Advanced Python Programming

Code :	PRG404	**Duration :**	3 days	**Category :**	Scripting

Audience :

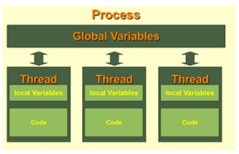
This course is for Python developers who want to know more about the Python language and who wish to become proficient in advanced aspects of Python.

Prerequisites :

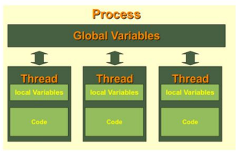
To participate in this course knowledge of and experience with programming in Python is required.

Realization :

The theory is discussed on the basis of presentation slides. Illustrative demos illustrate the concepts. The theory is interspersed with exercises. The course material is in English.



Advanced Python Programming



Contents :

In this course advanced aspects of the Python programming language that simplify and accelerate the development of Python software are discussed. The latest versions of Python 2.x and 3.x add interesting features to the language and in this course participants learn how to use them. Among other subjects iterators are addressed that allow lazy evaluation in the sense that an object is generated only when needed and generators and coroutines for concurrent programming are discussed. The course continues with decorators that enable the addition of functionality to existing functions and classes such as caching and proxying. Also context managers are discussed and it is shown that the with statement makes code more robust and exception handling easier. In the module patterns several design patterns are examined in Python and attention is paid to the pythonic principle that states "It's Easier to ask for forgiveness than permission (EFAP)." This principle supports the robust exception handling in Python. For many problems Python offers standard solutions that need Design Patterns in other environments. These pythonic solutions are discussed in the module conventions. Furthermore, attention is paid to advanced features such as meta programming and the use of comprehensions and descriptors. Finally the coding style according to the Python style guide (PEP8) is treated and the performance optimalization of Python code.

Module 1 : Iterators and generators

What are Iterators?
Lazy evaluation
yielding versus returning
itertools module
What are Generators?
Generator expressions
Bidirectional communication
Chaining generators
Coroutines

Module 2 : Decorators

What are Decorators?
Tweaking original object
Replacing original object
Decorators on classes
Decorators on functions
Copying the docstring
Examples in library
Deprecation of functions
while-loop removing decorator
Plugin registration system

Module 3 : Context Managers

What are Context managers?
with statement
Catching exceptions
Defining context managers
Using Context managers
Examples standard library
contextlib

Module 4 : Patterns in Python

EFAP principle
Singletons
Singleton variants
null Objects
null versus None
Proxies
Proxy examples
Observer
Publish and subscribe
Constructor

Module 5 : Conventions in Python

Pythonic principles
Out of the box solutions
Wrapping instead of inheritance
Dependency injections
Factories
Duck typing
Monkey patching
Callbacks

Module 6 : Meta Programming

What are meta classes?
Default meta class
Dynamic classes
Creating classes
Creating object
Adding base classes
Adding fields
Adding methods
Meta class hook

Module 7 : Comprehensions

What are comprehensions?
Lambda Operator
Filter
Reduce and Map
Functional Programming
Generator comprehensions
List comprehensions
Dictionary comprehensions
Set comprehensions

Module 8 : Descriptors and Style

Python descriptors
Descriptors protocol
set, get and delete
Property type descriptors
Decorator type descriptors
Run time descriptors
Python style
Style guide PEP8
pylint and pep8.py

Module 9 : Python Performance

Optimization Guidelines
Influencing speed factors
Optimization strategies
Improving algorithms
Caching
Data Structures
Testing speed
Psyco JIT Compile