Zaawansowany Python

Znajomość składni **Pythona** to dopiero punkt startowy — prawdziwa biegłość zaczyna się tam, gdzie kończy się pisanie prostych skryptów i zaczyna świadome kształtowanie architektury, wydajności i niezawodności kodu.
Szkolenie przeprowadza uczestników przez zaawansowane mechanizmy języka: od protokołu iteracji i menedżerów kontekstu, przez system typów i metaklasy, po współbieżność z asyncio i profilowanie na poziomie opkodów.
Dużą uwagę poświęcamy jakości kodu — statycznej analizie typów z mypy, testowaniu z pytest oraz budowaniu środowisk z uv i pyproject.toml.
Szkolenie adresowane jest do programistów **Pythona** z co najmniej rocznym doświadczeniem, którzy chcą pisać kod gotowy na środowiska produkcyjne — czytelny, testowalny, wydajny i łatwy w utrzymaniu.

Zakres tematyczny

Zaawansowane konstrukcje językowe

- Protokół iteracji: __iter__, __next__, generatory i yield from
- Moduł itertools i more-itertools — kompozycja potoków danych
- Menedżery kontekstu: __enter__/__exit__, contextlib.contextmanager
- Dekoratory funkcyjne i klasowe — wzorce i pułapki
- Ćwiczenie: budowa lazy pipeline'u do przetwarzania dużych zbiorów danych

System typów i jakość kodu

- Type hints w praktyce: adnotacje, TypeVar, Generic, Protocol
- Statyczna analiza z mypy — konfiguracja i interpretacja błędów
- dataclasses i attrs — nowoczesne definiowanie struktur danych
- Zarządzanie środowiskami i zależnościami z uv i pyproject.toml
- Ćwiczenie: typowanie istniejącej bazy kodu i eliminacja błędów mypy

Typy, klasy i metaklasy

- Introspekcja: inspect, dir, getattr, __dict__
- Deskryptory — mechanizm stojący za @property i @classmethod
- Metaklasy: kiedy i po co ich używać
- __init_subclass__ i __class_getitem__ jako lżejsze alternatywy

Imperium Szkoleniowe
Tel. 22 402 15 04, Fax: 22 258 23 62
biuro@imperiumszkoleniowe.pl

- → Dynamiczne tworzenie klas z type() i types.new_class
- → Ćwiczenie: implementacja frameworka walidacji opartego na deskryptorach

Testowanie

- → pytest jako standard: fixtures, parametryzacja, markery
- → Mockowanie z unittest.mock i pytest-mock
- → Pokrycie kodu: coverage.py — pokrycie linii vs rozgałęzień
- → Testowanie własnościowe z hypothesis
- → tox i nox — automatyzacja testów na wielu wersjach Pythona
- → Ćwiczenie: napisanie pełnego zestawu testów dla modułu z użyciem TDD

Profilowanie i optymalizacja

- → Profilowanie czasowe: cProfile, line_profiler, py-spy
- → Profilowanie pamięci: tracemalloc, memray
- → Czytanie opkodów — moduł dis
- → Optymalizacja: słabe referencje, __slots__, lokalne zmienne
- → Kiedy sięgać po Cython, ctypes lub rozszerzenia w C
- → Ćwiczenie: identyfikacja i eliminacja bottlenecku w aplikacji

Kodowania, serializacja i dane binarne

- → Unicode, UTF-8 i pułapki kodowań w I/O
- → struct — czytanie i pisanie binarnych formatów danych
- → Serializacja: pickle, json, msgpack, protobuf
- → Bezpieczeństwo deserializacji — pickle jako wektor ataku
- → Ćwiczenie: parser binarnego formatu pliku

Współbieżność i asynchroniczność

- → GIL — co blokuje, a czego nie blokuje
- → Wątki (threading) vs procesy (multiprocessing) — kiedy co stosować
- → concurrent.futures — unified API dla wątków i procesów
- → Programowanie asynchroniczne: async/await, pętla zdarzeń, coroutines
- → asyncio w praktyce: Tasks, Queues, synchronizacja
- → Wzorce: producer-consumer, ograniczanie współbieżności z Semaphore
- → Przegląd ekosystemu: anyio, trio, asynchroniczne klienty HTTP
- → Ćwiczenie: asynchroniczny scraper z ograniczeniem rate-limit

Imperium Szkoleniowe
Tel. 22 402 15 04, Fax: 22 258 23 62
biuro@imperiumszkoleniowe.pl