MATH-661

Advanced Scientific Programming in Python

Buffa Annalisa, Hinz Jochen Peter

Cursus	Sem.	Type
Mathematics		Opt.

Language of teaching	English
Credits	1
Session	
Exam	Project report
Workload	30h
Hours	**12**
Lecture	12
Number of positions	

Frequency

Only this year

Remark

The course expects participants to at least have an intermediate understanding of the Python programming language

Summary

This seminar teaches the participants to use advanced Python concepts for writing easier to read, more flexible and faster code. It teaches concepts in a hands-on and tangible fashion, providing example use cases that all applied mathematicians can relate to.

Content

Lecture 1: Idiomatic Python. The first lecture teaches the use of Python "idioms" for writing code that is more readable and extendible.
- What are code characteristics that enhance readability?
- *args, **kwargs and their use. The use of dict.setdefault.
- Pythonic use of "truthiness". Use of `or` and `and` on non-boolean objects.
- Idiomatic variable unpacking with star syntax. Using syntactic sugar to enhance readability and for catching errors early.
- Advanced iteration: enumerate, zip, use of star syntax in loops, for - else clause. The use of map (...)
- Advanced use of list, set, dict. Using comprehensions and iterators to avoid instantiation via for loops.
- Creative use of dictionaries, dict.setdefault, dict.get(item, None) to avoid if-else clauses.
- Use of itertools: loop flattening with itertools.product, counting with itertools.count, zip_longest + tangible use cases from applied math.
- Use of collections: namedtuple, ChainMap, defaultdict + use cases mathematicians can relate to
- dataclass
- The concept of Hashing
- The use of functools and decorators - functools.partial for specialising functions, caching via functools.lru_cache, writing decorators and using functools.wraps, functools.reduce
- Iterating over numpy.ndarray's.
- Writing functions that can handle several input types (achieving multiple dispatch).
- Unit testing.

Lecture 2: Object-oriented Python (requires a basic understanding of OOP)
- The basics of OOP. Favouring out-of-place operations for code robustness wherever computationally feasible.
- The art of catching errors early in __init__.
- Properties, classmethods and staticmethods. Lazy evaluation via functools.cached_property.
- "Magic methods" __call__, __eq__. Implementing comparison >, >=, ...using functools.total_ordering to avoid boilerplate.
- Inheritance and the use of ABC - abstract base classes, abstractmethods, abstractproperties, ... The use of super()
- Structural subtyping as an alternative to ABC - static type checks with mypy.
- Design patterns: factory pattern, strategy pattern, decorator pattern.
- Correctly inheriting from np.ndarray's. Compatibility with numpy broadcasting and ufuncs.
- Admissible use cases of multiple inheritance (MixIns)

- Writing hashable classes
- Writing your own context managers + applications
- The use of collections.abc to correctly inherit from Mapping, Hashable, Container, ...
- The use of self.__class__ to avoid boilerplate
- Application: writing a class capable of representing various mesh types.

Lecture 3: The best of two worlds - writing code that is fast and readable
- Advanced numpy vectorisation - avoiding loops, advanced broadcasting using (slice(...), slice (...), ...) and ellipsis.
- The use of numpy.reshape.
- Parallelisation in Python.
- The differences / similarities between Python and compiled languages like C.
- Writing jit-compiled functions in Numba; Numba parallelization.
- Using the creole language "Cython" to statically type where possible.
- Interfacing with C using ctypes.
- The Python "lowest hanging fruit" tree. The 80-20 rule. Profiling to catch performance bottlenecks.

Keywords

scientific computing, python

Learning Prerequisites

Required courses

intermediate familiarity with the Python programming language, numpy and (if possible) scipy.

Learning Outcomes

By the end of the course, the student must be able to:

- Compose idiomatic python code, specifically geared toward topics in applied math

Resources

Bibliography

Ipython notebooks will be made available on GitHub.

Moodle Link

- https://go.epfl.ch/MATH-661