Computer Systems Modeling & Verification

Tristan Crolard

Department of Computer Science
CEDRIC lab / SYS team

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Evaluation Modalities

- **attendance and participation** (50% of the final grade)
  - **class attendance**: justified or unjustified absence, same rule (20%)
  - **punctuality**: be on time, for the session and after the break (5%)
  - **participation**: focus on the lesson, put your phone out of sight (25%)

- **on-paper exam** (50% of the final grade)

- **remedial exam** (beware: only for the "on-paper exam" part)

Course content

Computer Systems Modeling & Verification

- **Computer System**: hardware and software components
- **Modeling**
  - *Specification*[1]: requires some mathematical formalism
- **Verification**
  - *Properties*: requires a method to check or prove them

1. **Specification**: defines *what* behaviour or result we should expect.
   Implementation: describes *how* to obtain the expected behaviour or result.
   Verification: provides some evidence that an implementation is correct with respect to a specification.

Course content – details

Computer Systems Modeling & Verification

- **Computer System**: hardware and software components
- **Modeling**
  - *Specification*: requires some mathematical formalism
    Axiomatic systems (pure logic, non executable)
    Equational systems (pure functional programs)
    Deductive systems (pure logic programs)
- **Verification**
  - *Properties*: requires a method to check or prove them
    Static: from type checking to program proving
    Dynamic: from classical unit testing to property-based testing

Course content – outline

- **Preliminaries** — [8 sessions]
  - imperative programming and unit testing
  - functional programming and logic
- **Part I: static verification (static analysis)** — [4 sessions]
  - typing rules as a deductive system
  - extraction of a functional type-checker
- **Part II: dynamic verification (testing)** — [4 sessions]
  - code coverage (structural testing)
  - property-based testing (functional testing)

Course organization

- **Lectures, Tutorials and Practicals**
  (slides and assignments available on Moodle)

- **Prototypes** written in **Python**

  Why Python?

  - not the best programming language, but...
  - one of the most popular language today
  - and also used in several courses in M2

Prototypes

- an interpreter for mini-python[2]
- a type-checker for mini-python
- a transpiler from mini-python to a C-like language (*if time permits*)
- a property-based testing framework for mini-python
- a test coverage framework for mini-python (*if time permits*)

2. only a small fragment of the full python syntax

Python

What is Python?

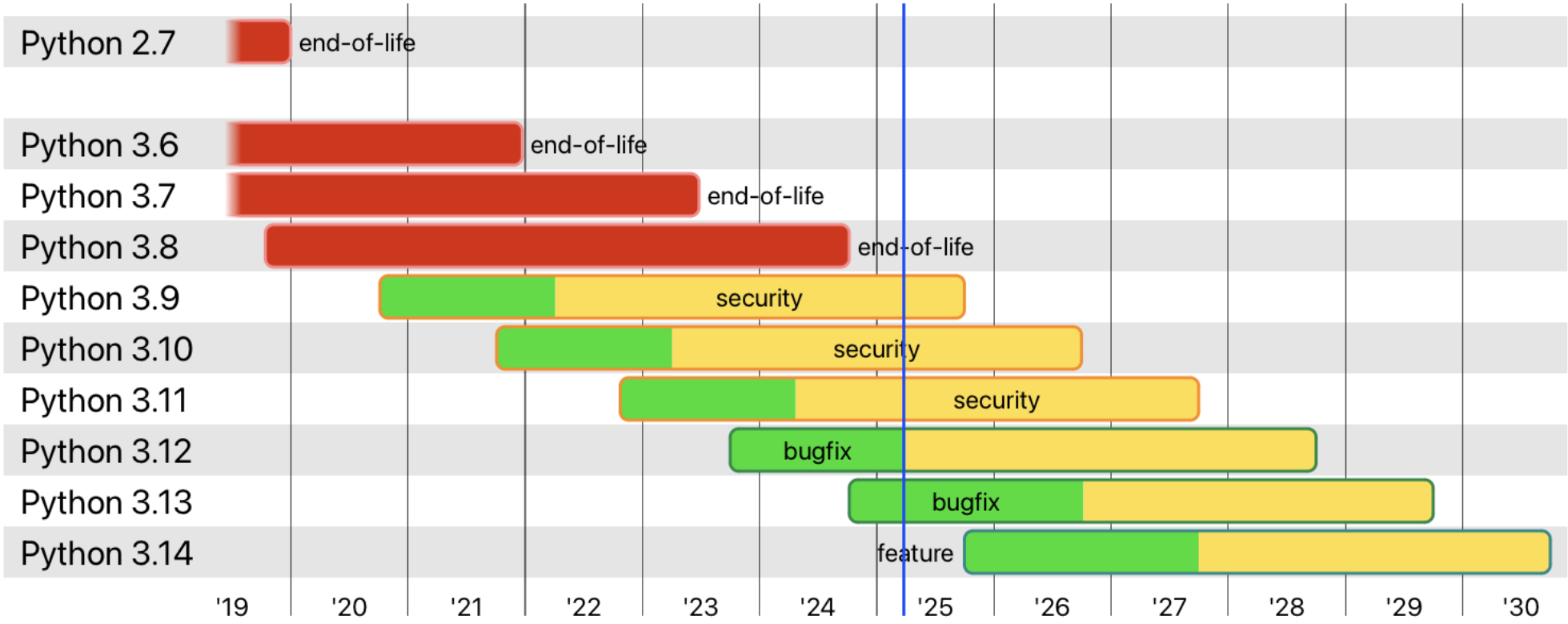
- high level programming language
- bytecode compiler, bytecode interpreted by default
  (but optimizing compilers are available)
- supports three main programming styles
  (imperative, functional, object-oriented)
- General purpose tool (yet also good for specialized work with extension libraries)

Availability

- Python is free (and open source)
- Python is platform independent (works on Windows, Linux/Unix, Mac OS, ...)

Status of Python Versions

https://devguide.python.org/versions



What's New in Python?

- https://docs.python.org/3/whatsnew/3.5.html
  PEP 484: Type Hints – 2015
- https://docs.python.org/3/whatsnew/3.6.html
  PEP 526: Syntax for Variable Annotations – 2016
- https://docs.python.org/3/whatsnew/3.7.html
  PEP 557: Data Classes – 2018
- https://docs.python.org/3/whatsnew/3.10.html
  PEP 634: Structural Pattern Matching – 2021
- https://docs.python.org/3.12/whatsnew/3.12.html
  PEP 695: Type Parameter Syntax – 2023

Python: a modern programming language?

	Java	(Java/Type)Script	Python
records	Y (java 17) [2021]	Y	Y (dataclasses)
pattern matching	Y (java 21) [2023]	Y (partial)	Y (3.10) [2021]
exceptions	Y	Y	Y
object-oriented	Y	Y	Y
functional programming	Y (java 8) [2014]	Y	Y
static type checking	Y	N/Y (gradual)	Y (gradual) [2014]
proper generics	Y (java 5) [2009]	Y	Y (3.12) [2023]

In this course:

- mostly **functional programming** with **pattern matching** (3.10)
- only **modern python** with **type hints** – including **generics** (3.12)

Python environment

A standard python installation comes with CPython:

- an interpreter: a bytecode compiler & a VM that runs the bytecode
- this interpreter can also be used interactively
  REPL: Read Eval Print Loop

Remarks

- There is no explicit compilation step (the compilation step is hidden).
- Type hints are ignored by (the bytecode compiler and) the interpreter.

```
>>> x: int = "ok"
>>> type(x)
<class 'str'>
```

Type Hints

Static type checkers

- **Mypy** (reference implementation)
  - https://mypy.readthedocs.io/en/stable/index.html (mypy documentation)
  - https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html (Type hints cheat sheet)
  - https://mypy.readthedocs.io/en/stable/builtin_types.html (Built-in types)
- **Pyright** (included in the recommended VS Code extension)
  - https://github.com/microsoft/pyright
- **Pytype** (google), **Pyre** (facebook), **PyCharm** (JetBrains), ...

Support materials

- Material from the course *Python for Computational Science*[3]
  Updated with type hints for this course
- *Introduction to Python for Computational Science and Engineering*
  Textbook (2024)
- *Python for Computational Science*
  Lecture slides (2024)

Recommended books

- *Robust Python* (2021), Patrick Viafore, O'Reilly.
- *Functional Python Programming*, 3rd edition (2022), Steven F. Lott, Packt Publishing.

3. *Python for Computational Science*, Hans Fangohr et al. University of Southampton

Python documentation

Official Python documentation:

- Python documentation page
- Python Standard Library
- Python tutorial
- Python Package Index (pip repository)

Online documentation about types, for example:

- Mypy documentation
- Type hints cheat sheet
- Built-in types
- Pyright

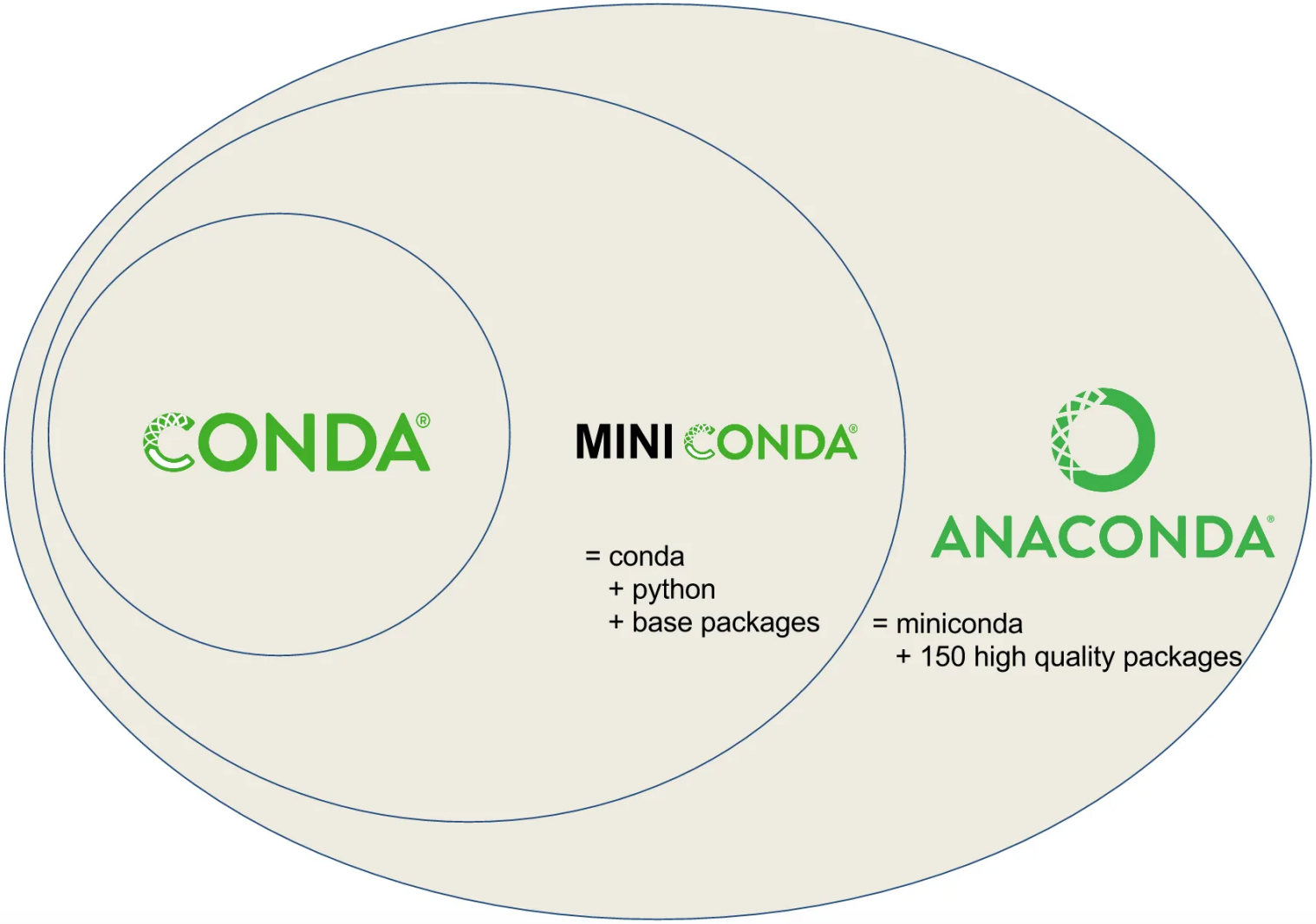
Installation of Python

Several options

- System's package manager (admin rights required)
- Official python binary installer (admin rights required)
  https://www.python.org/downloads
- Anaconda (popular distribution for data scientists)
  https://www.anaconda.com/download
- Miniconda (minimal anaconda distribution)
  https://docs.conda.io/en/main/miniconda.html

Note. Anaconda (or miniconda) allows you to install different versions of python and to switch easily between them (and admin rights are not required).

Conda package manager (optional)



Installing packages

Using pip (the official package manager)

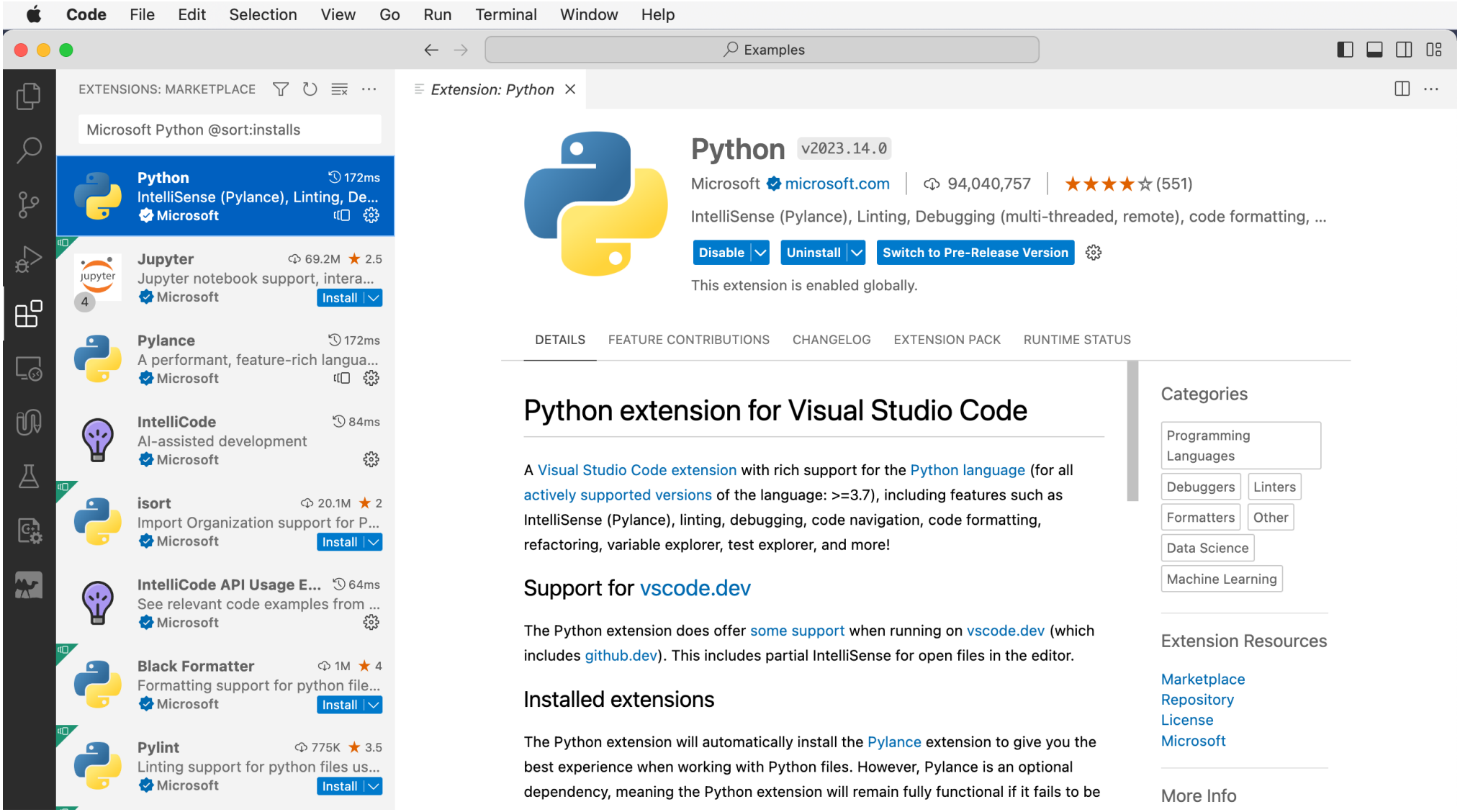
```
pip3 install some-package
```

Notes.

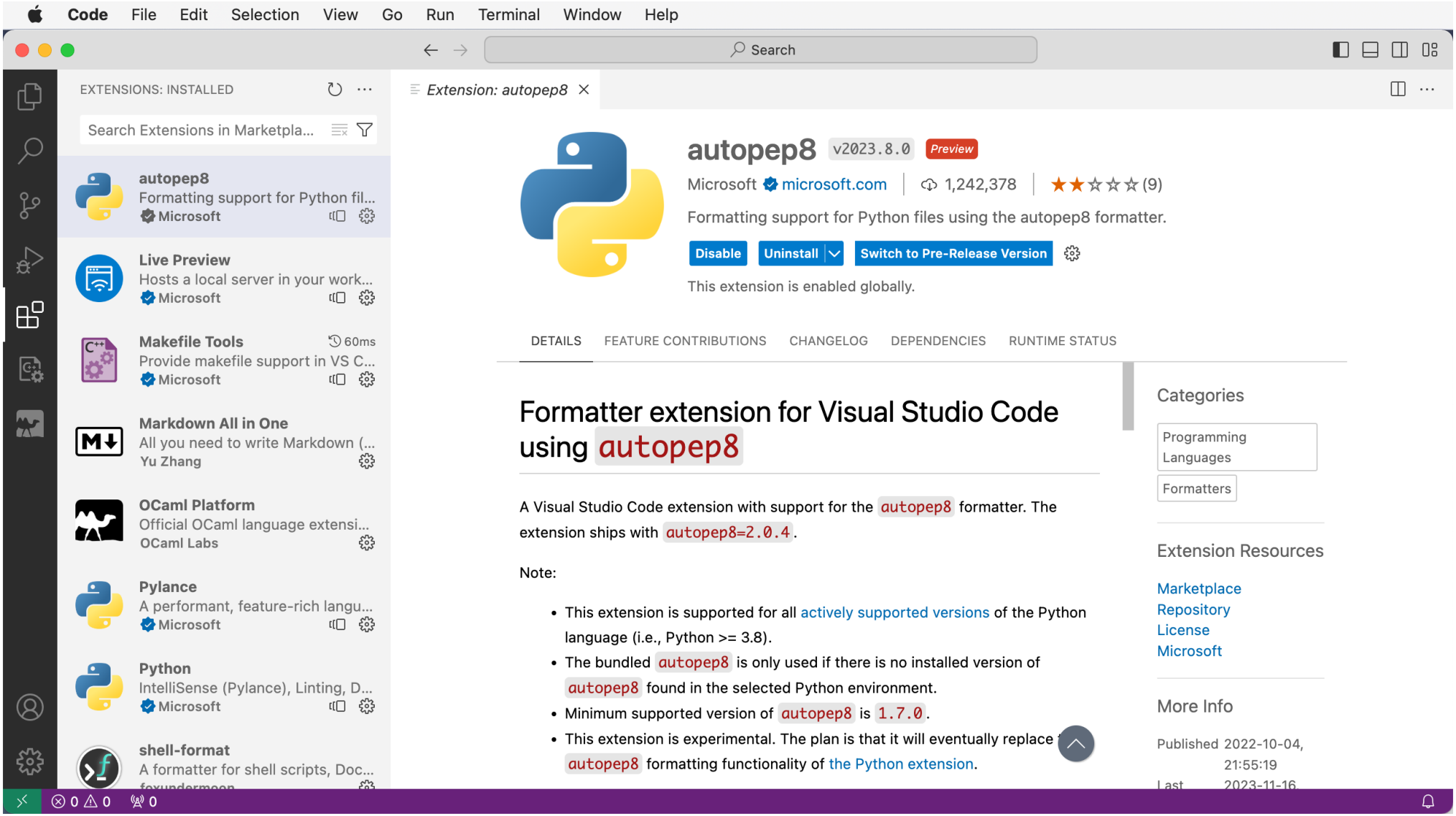
- On some Linux systems, you need to use your system package manager:
  `sudo apt install python3-some-package`
- If you use conda, you can also install conda packages[4] with the command:
  `conda install some-package`

4. You should avoid mixing both kind of packages, but if you really need to, see:
https://www.anaconda.com/blog/using-pip-in-a-conda-environment

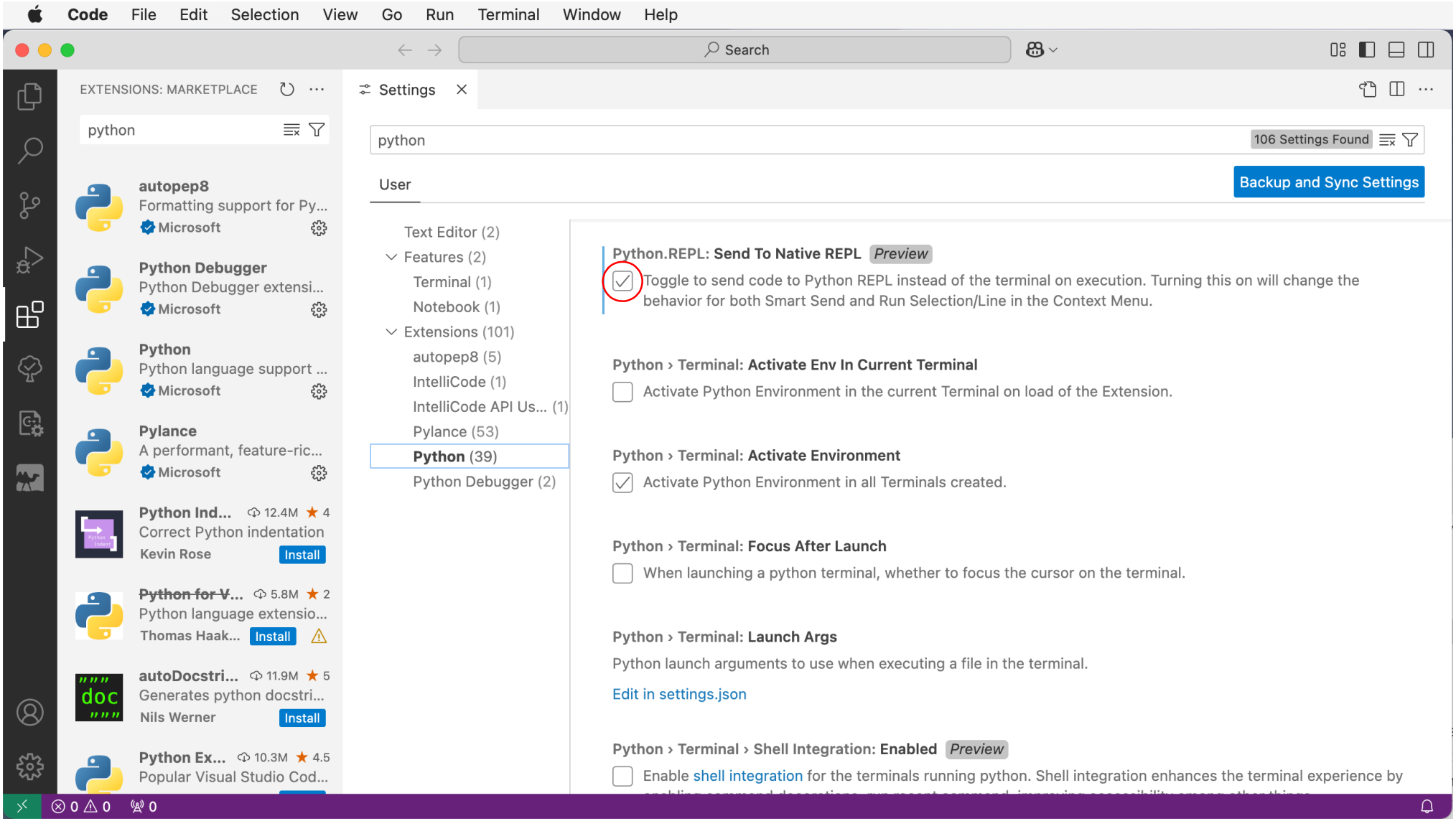
Installing Python extension for VS Code



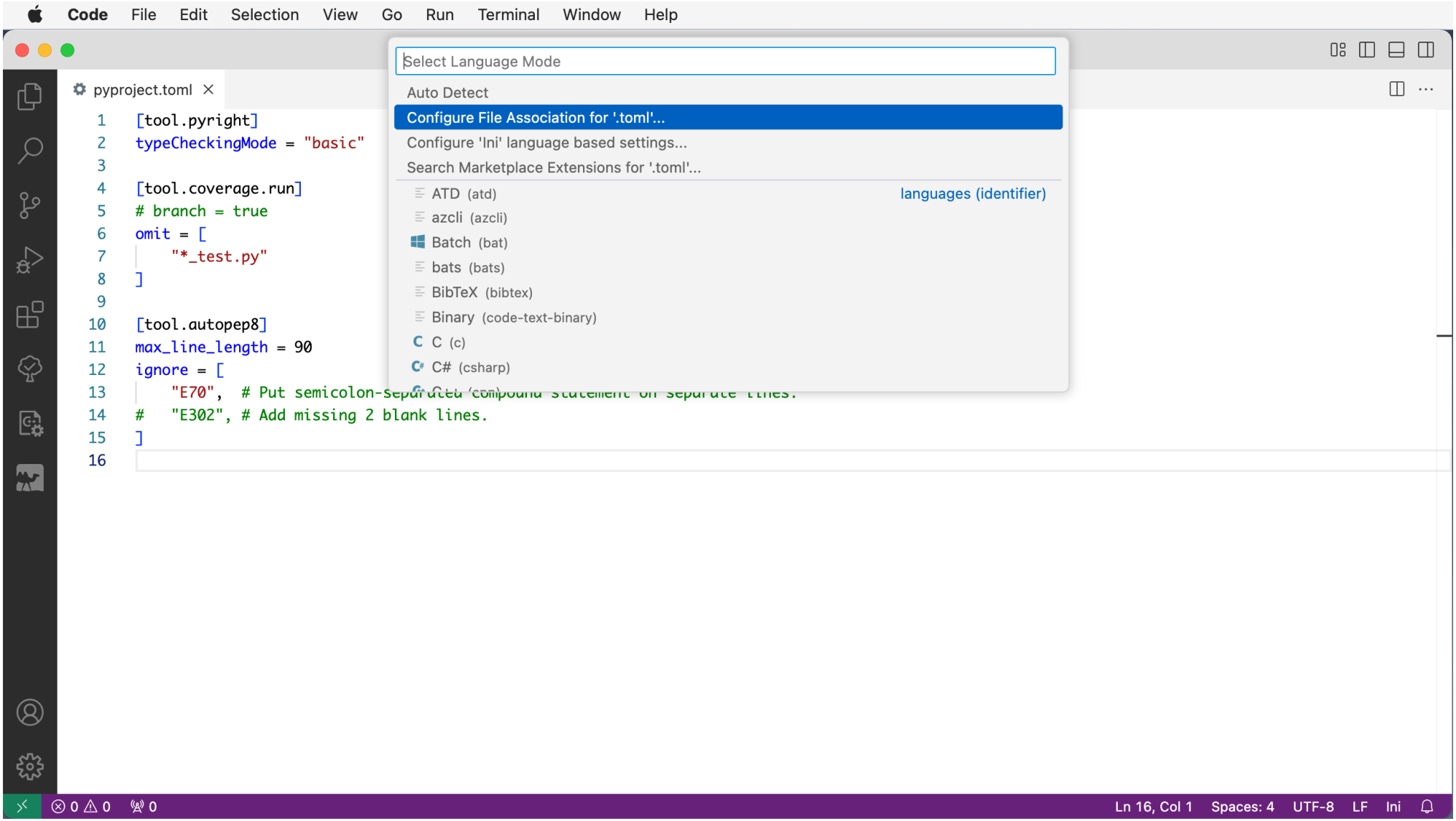
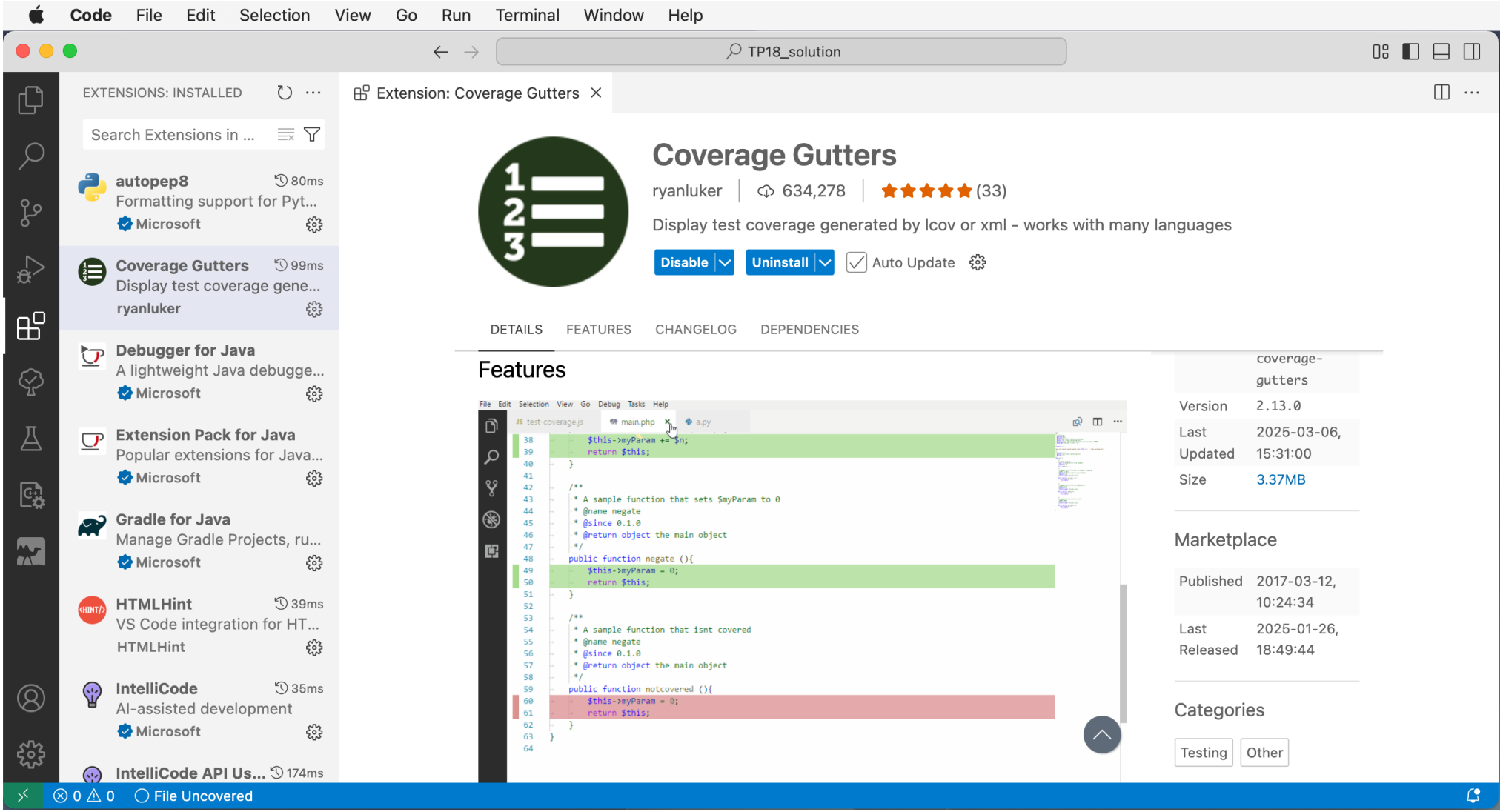
Installation of autopep8 extension



Configuration of Python extension: native REPL



Configuring mode “ini” for “pyproject.toml”



Coverage Gutters (VS Code older than 1.94)