

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** is required: any absence must be justified (before the session if possible, after otherwise)
 - **active participation in lessons** is part of the evaluation
- ▶ **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:* requires some mathematical formalism
- ▶ **Verification**
 - *Properties:* requires a logic to check or prove them

Course content – details

Computer Systems Modeling & Verification

- ▶ **Computer System:** hardware and software components
- ▶ **Modeling**
 - *Specification:* requires some mathematical formalism
 - Axiomatic systems (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

- imperative programming and unit testing
- functional programming and logic

▶ Part I: static verification (static analysis)

- typing rules as a deductive system
- mode-based extraction of a functional type-checker

▶ Part II: dynamic verification (testing)

- test coverage
- design-by-contract
- self-testing and property-based testing

Course organization

- ▶ **Lectures, Tutorials and Practicals**
(slides and assignments available on Moodle)
- ▶ **Prototypes** written in **Python**¹

Why Python?

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

1. up to last year, we used OCaml and Ada

Prototypes

- ▶ a **type-checker** for mini-python²
- ▶ a **transpiler** from mini-python to a C-like language
- ▶ an **interpreter** for mini-python
- ▶ a **test coverage** framework for mini-python
- ▶ a **property-based testing** framework for mini-python

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)

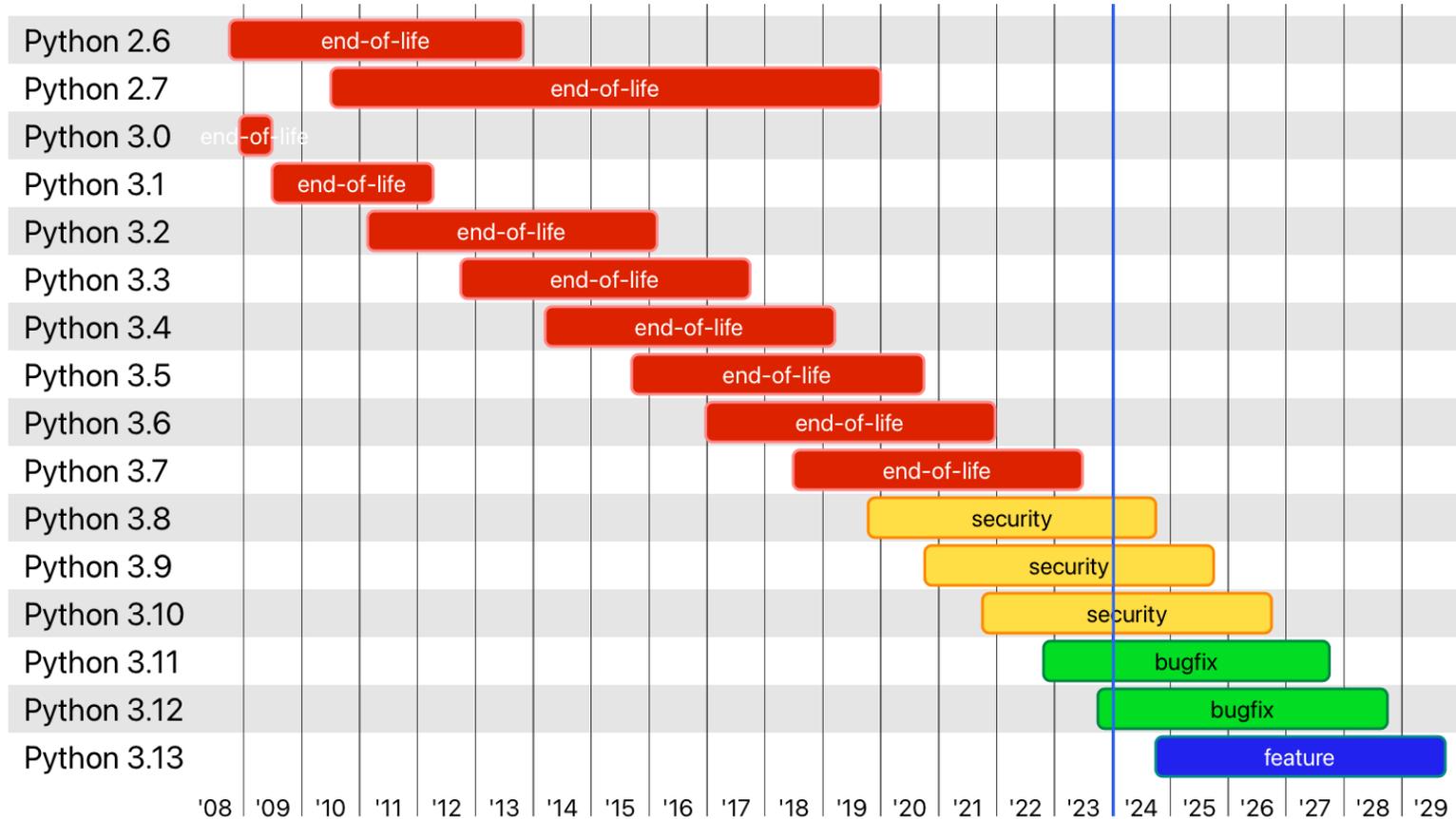
Availability

- ▶ 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>

Python release cycle



What's New in Python?

- ▶ <https://peps.python.org/pep-0484>
(Type Hints - 2014)
- ▶ <https://peps.python.org/pep-0526>
(Syntax for Variable Annotations - 2016)
- ▶ <https://peps.python.org/pep-0589>
(TypedDict - 2019)
- ▶ <https://docs.python.org/3/whatsnew/3.10.html>
(PEP 634: Structural Pattern Matching)
- ▶ <https://docs.python.org/3.12/whatsnew/3.12.html>
(PEP 695: Type Parameter Syntax)

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:

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

Python environment

A standard python installation comes with CPython:

- an interpreter (a bytecode compiler & a VM that runs for the bytecode)
- the 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), ...

Support materials

- ▶ Material from the course *Python for Computational Science*³
Beware: no type hints in this course
- ▶ *Introduction to Python for Computational Science and Engineering* Textbook (2022)
- ▶ *Python for Computational Science* Lecture slides (2024)

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

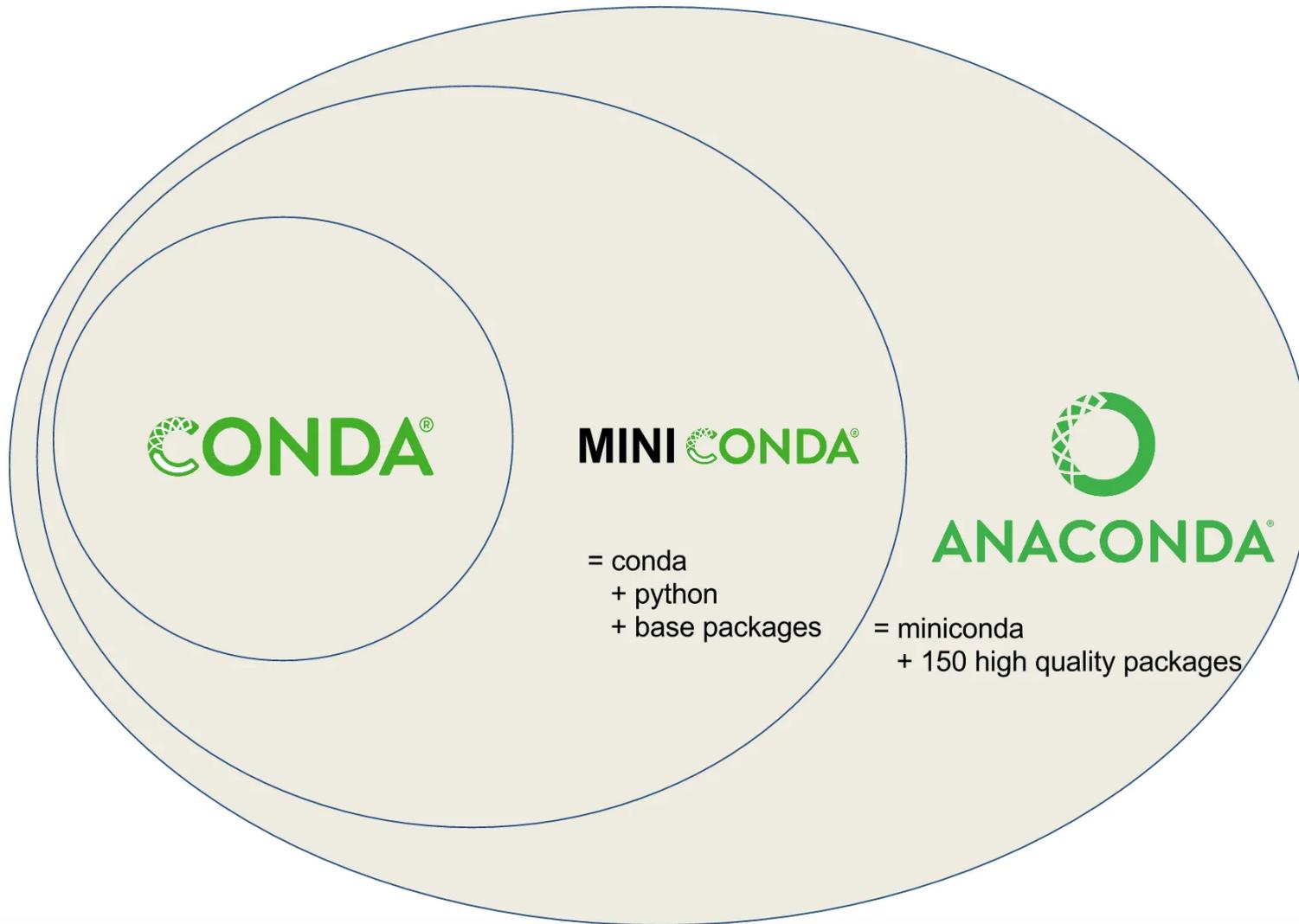
Installation of Python

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



Configuration of Python for CSMV (optional)

Using either anaconda or miniconda

Create a new conda environment for CSMV (with the default python version):

In the terminal, type

```
conda create -n CSMV
```

You can then then switch between versions using

```
conda activate CSMV
```

or

```
conda activate base
```

Note. In VS Code, you need to select the interpreter (and then “activation” is automatic in the integrated terminal).

Installing packages

Using conda or pip (official package manager)

In the terminal, type

```
conda install some-package
```

or

```
pip install some-package
```

Note. You should avoid mixing both kind of packages. However, if you need to, you should install conda packages first, and then pip packages⁴.

4. <https://www.anaconda.com/blog/using-pip-in-a-conda-environment>

Installing Python extension for VS Code

The image shows a screenshot of the Visual Studio Code interface. On the left, the 'EXTENSIONS: MARKETPLACE' sidebar is open, displaying a list of Python-related extensions. The top extension is 'Python' by Microsoft, which is highlighted in blue. Below it are 'Jupyter', 'Pylance', 'IntelliCode', 'isort', 'IntelliCode API Usage E...', 'Black Formatter', and 'Pylint'. The main editor area shows the details for the 'Python' extension (version v2023.14.0) by Microsoft. The extension is described as providing IntelliSense (Pylance), Linting, Debugging, code formatting, etc. It is currently installed, and the 'Disable' and 'Uninstall' buttons are visible. The 'Switch to Pre-Release Version' button is also present. The extension is enabled globally. Below the main description, there are tabs for 'DETAILS', 'FEATURE CONTRIBUTIONS', 'CHANGELOG', 'EXTENSION PACK', and 'RUNTIME STATUS'. The 'DETAILS' tab is active, showing a sub-section titled 'Python extension for Visual Studio Code' with a description of the extension's capabilities. To the right of the main content, there are sections for 'Categories' (Programming Languages, Debuggers, Linters, Formatters, Other, Data Science, Machine Learning), 'Extension Resources' (Marketplace, Repository, License, Microsoft), and 'More Info'.

EXTENSIONS: MARKETPLACE

Microsoft Python @sort:installs

Python v2023.14.0
Microsoft microsoft.com | 94,040,757 | ★★★★★ (551)
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), code formatting, ...
Disable Uninstall Switch to Pre-Release Version
This extension is enabled globally.

DETAILS FEATURE CONTRIBUTIONS CHANGELOG EXTENSION PACK RUNTIME STATUS

Python extension for Visual Studio Code

A [Visual Studio Code extension](#) with rich support for the [Python language](#) (for all [actively supported versions](#) of the language: ≥ 3.7), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!

Support for vscodetesting.com

The Python extension does offer [some support](#) when running on vscodetesting.com (which includes github.com). This includes partial IntelliSense for open files in the editor.

Installed extensions

The Python extension will automatically install the [Pylance](#) extension to give you the best experience when working with Python files. However, Pylance is an optional dependency, meaning the Python extension will remain fully functional if it fails to be

Categories

- Programming Languages
- Debuggers
- Linters
- Formatters
- Other
- Data Science
- Machine Learning

Extension Resources

- [Marketplace](#)
- [Repository](#)
- [License](#)
- [Microsoft](#)

More Info

Configuration of Python extension

The screenshot shows the VS Code interface with the Settings window open. The search bar contains 'pylance', and 34 settings are found. The 'Pylance' section is expanded, showing the following settings:

- Python > Analysis > Inlay Hints: Variable Types**
 - Enable/disable inlay hints for variable types:
`foo':List[str]' = ["a"]`
- Python > Analysis: Log Level**
 - Specifies the level of logging for the Output panel
 - Information
- Python > Analysis: Persist All Indices**
 - Indices for all third party libraries will be persisted to disk.
- Python > Analysis: Stub Path**
 - Path to directory containing custom type stub files.
 - typings
- Python > Analysis: Type Checking Mode**
 - Defines the default rule set for type checking. Note that the default value is set to "basic" when using VS Code Insiders, and to "off" otherwise.
 - basic

Installation of autopep8 extension

The screenshot shows the Visual Studio Code interface with the Extensions Marketplace open. The 'autopep8' extension by Microsoft is selected. The extension page displays the following information:

- Extension Name:** autopep8
- Version:** v2023.8.0 (Preview)
- Author:** Microsoft (microsoft.com)
- Downloads:** 1,242,378
- Rating:** 4.5 stars (9 reviews)
- Description:** Formatting support for Python files using the autopep8 formatter.
- Buttons:** Disable, Uninstall, Switch to Pre-Release Version
- Status:** This extension is enabled globally.

The extension page also includes a detailed description:

Formatter extension for Visual Studio Code using autopep8

A Visual Studio Code extension with support for the `autopep8` formatter. The extension ships with `autopep8=2.0.4`.

Note:

- This extension is supported for all [actively supported versions](#) of the Python language (i.e., Python >= 3.8).
- The bundled `autopep8` is only used if there is no installed version of `autopep8` found in the selected Python environment.
- Minimum supported version of `autopep8` is `1.7.0`.
- This extension is experimental. The plan is that it will eventually replace `autopep8` formatting functionality of [the Python extension](#).

Categories: Programming Languages, Formatters

Extension Resources: Marketplace, Repository, License, Microsoft

More Info: Published 2022-10-04, 21:55:19; Last updated 2023-11-16