DxMT 東京科学大学拠点 D²MatE 2026年度チュートリアルシリーズ
東京科学大学 総合研究院 フロンティア材料研究所学術講演会

生成AIとToyプログラムで理解するバンド理論・データ解析

第２回第１部　科学計算用pythonのバージョン選定

神谷利夫

東京科学大学
総合研究院
元素戦略MDX研究センター

Institute of SCIENCE TOKYO

本チュートリアルには、
Microsoft365 Copilot / Chat GPT / Gemini / Claude等
を使用した結果が含まれています

Pythonのバージョンを選ぶ基準

環境・目的依存

- 以前作ったプログラムが動くか
  - 3.XXのプログラムのほとんどは新しいpythonでうごく (前方互換性)
  - ただし、標準ライブラリから削除される、ライブラリの仕様が変わる、ライブラリのサポート中止・公開停止などで動かなくなることがある
- 配布先 (古いpythonなど) で動くか (後方互換性)
  - 新しいpythonで実装された機能は使わない

Pythonバージョンの選択に要求される条件

- サポート期間が残っていること
- 使用するNumPy・SciPy・専門ライブラリが対応すること
- 保守やバグ発見を支援する機能があるとよい: 型ヒントとエラー表示の改善など
- 新構文の多さより、動作確認済みの環境 (後方互換性) を優先
- ここでは、試験中の新機能 (並列実行やJITの評価など) を選定理由に含めない

推奨バージョンとサポート期限

Python	終了予定	用途の目安
3.15	2031年10月	公開直後。依存ライブラリの対応を確認
3.14	2030年10月	新規環境の第一候補
3.13	2029年10月	互換性に制約がある場合の候補
3.12	2028年10月	既存環境は継続利用可能
3.11	2027年10月	新規採用では残期間に注意
3.10	終了済み	更新対象

3.11〜3.13：セキュリティ修正のみ

3.11以降の重要な追加機能

導入	主な機能	古いPythonへの対応
3.11	エラー位置の詳細表示	既存コードの修正不要
3.11	Self、Requiredなどの型ヒント	多くはtyping_extensionsで補完
3.11	TOML読み込み：tomllib	旧環境ではtomliを利用
3.12	@overrideで継承時の誤りを検出	typing_extensionsで補完
3.12	type文・型パラメータ構文	3.11以前は構文エラー
3.13	TypeIs・型パラメータの既定値	通常の計算コードでは優先度低め

保守性に役立つ主要機能は3.11〜3.12でほぼ充足。match構文は3.10から利用可能

後方互換性の注意点

更新先	既存コードへの主な影響	対応
3.12	distutils・impの削除	古いビルド手順・依存を更新
3.13	cgiなどの削除	古い補助コード・依存を確認
3.14	型注釈を遅延評価	型注釈を実行時に読む処理を確認
3.15	既定文字コードがUTF-8に	open()でencodingを明示

Python・環境・パッケージ管理ツールの比較

項目	pip	conda	uv
主目的	Pythonパッケージ管理	Python + ネイティブライブラリ管理	Python管理 + 仮想環境 + パッケージ管理
Python本体のインストール	×	○	○
複数Python管理	pyenv等が必要	○	○
仮想環境	venv併用	内蔵	内蔵
パッケージソース	PyPI	conda-forge等	PyPI
C/C++ライブラリ管理	弱い	非常に強い	pipと同等
インストール速度	普通	やや遅い	非常に高速
requirements.txt	○	△	○
pyproject.toml	○	△	◎
pip互換	本人	-	高い
オフライン再現性	普通	高い	高い
ライセンス	MIT系	BSD-3-Clause	MIT/Apache系OSS
お勧め度(2026)	○	○	◎

Python・環境・パッケージ管理ツールの比較

項目	venv/pip Python標準のパッケージ管理	Conda 科学計算向け統合環境管理	Uv 次世代Python環境管理
インストール (Windows)	Python公式インストーラ	Miniforge3-Windows-x86_64.exe	winget install astral-sh.uv
インストール (Linux)	OS付属Pythonまたは公式Python	wget Miniforge.shbash Miniforge.sh	curl -LsSf https://astral.sh/uv/install.sh | sh
Python本体管理	×	○	○
仮想環境	venv利用	内蔵	内蔵
パッケージソース	PyPI	conda-forge, defaults	PyPI
pip互換	本体	△	◎
科学計算	○	◎	○
速度	○	△	◎

環境・パッケージ管理ツール チートシート

venv/pip: Python標準の仮想環境・パッケージ管理
pythonのインストールはできない
conda:科学計算向け統合環境管理
pythonのインストール可能
uv: 次世代Python環境管理
pythonのインストール可能

pythonのインストール

conda:
conda create -n py311 python=3.11
uv:
uv python install 3.11
uv python list --only-installed

仮想環境作成

venv/pip:
python -m venv .venv
Win: .venv¥Scripts¥activate
Linux: source .venv/bin/activate
conda:
conda create -n test python=3.12
conda activate test
conda deactivate
uv: プロジェクトディレクトリごとに
.venvをつくることが想定されている
uv venv
uv venv --python 3.12
uv venv py312 --python 3.12
Win: .venv¥Scripts¥activate
Linux: source .venv/bin/activate

パッケージ関係

pip:
pip install numpy
pip install numpy==2.3.0
pip uninstall numpy
pip list
conda:
conda install numpy
conda install numpy=2.3
conda remove numpy
conda list
uv:
uv add numpy
(推奨: uv.lock,pyproject.tomlを自動生成)
uv add numpy==2.3.0
uv remove numpy
uv tree
pip的な使い方:
uv pip install numpy
uv pip list

実行:

uv:
uv run --python 3.12 script.py

環境保存・復元

pip:
pip freeze > requirements.txt
pip install -r requirements.txt
conda:
conda env export > environment.yml
conda env create -f environment.yml
uv:
uv.lock
uv sync (uv.lock, pyproject.tomlから復元)
pip的な使い方: uv export > requirements.txt

一括インストール

pip:
pip install -r requirements.txt
conda:
conda env create -f environment.yml
uv:
uv pip install -r requirements.txt
uv sync (uv.lock, pyproject.tomlから復元)

uvでは、uv pip で、pipと同様のコマンドを使えることがある

推奨:
新規プロジェクト→uv
科学計算・CUDA→conda
既存プロジェクト→pip

まとめ: 科学計算用Pythonのバージョンを選ぶ基準

- Pythonは毎年マイナーバージョンアップをしている
- 最新版に対応するのに時間がかかるライブラリがある
  特に、研究グループなど小グループで開発・保守している
  科学計算ライブラリ

推奨方針

- 公開直後の最新版はさける (2026年10月時点で 3.15)
- サポート残期間が短いものも避ける (過去3年程度が候補: 3.13以降)
- 主として使うライブラリが対応しているかを確認
  生成AIの検索を活用し、公式の対応表・インストール情報で確認
- 2026年11月時点で、3.13 ~ 3.14推奨
- uv/conda でバージョン管理・パッケージ管理