IT資格道場

Python 3 エンジニア認定実践試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

Pythonエンジニア育成推進協会「Python3 エンジニア認定実践試験」出題範囲(主教材『Python実践レシピ』改訂新版・2026年12月1日〜)準拠
2026年11月版

SAMPLE

© IT資格道場｜無料サンプル｜転載・再配布・AI学習利用禁止

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

SAMPLE

目次

この教材群の使い方(3点セット)

この試験が問うもの

第1章 Pythonの環境 [1.1/1.2/1.3]

- pip でパッケージを管理する [1.1]
- venv で仮想環境を作る [1.2]
- uv でプロジェクトを管理する [1.3]

サンプルはここまでです

SAMPLE

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

SAMPLE

この試験が問うもの

Python 3 エンジニア認定実践試験は、Python で実際にコードを書く力を問う試験です。出題範囲は主教材『Python実践レシピ』の内容で、章ごとの出題数が公式に決まっています。このテキストの章立ても、主教材の章の順番どおり。出題が無い章と、出題範囲から除かれた節は置きません。

項目	内容
運営	一般社団法人Pythonエンジニア育成推進協会
問題数	40問（すべて選択問題）
試験時間	75分
合格ライン	正答率70%
出題の多い章	3章（7問）・9章（5問）・6章（4問）・4章と16章（各3問）

本番はコードの実行結果や、場面に合う書き方を選ぶ問題が多く出ます。例のコードは、手元の Python で一度実行して出力を確かめてください。

SAMPLE

第1章 Pythonの環境 [1.1/1.2/1.3]

pip でパッケージを管理する [1.1]

pip は、PyPI などの索引からパッケージを取ってきて入れる道具です。`pip` と打っても `python -m pip` と打ってもよく、後者は「いま使っている Python」に入れると決まります。Python が何本も並ぶ環境でも、入れ先を取り違えません。EXTERNALLY-MANAGED の印がある Python は、`--break-system-packages` を付けない限り、pip が書き換えません。ふだんは仮想環境の中で使います。[^c1-1]

入れる・消す・調べる

コマンド	働き
`pip install 名前`	入れる。入っていれば何もせず `Requirement already satisfied` と出る
`pip install -U 名前`	`--upgrade`。新しい版があれば入れ替える
`pip uninstall 名前`	消す。確認が出る。`-y` で確認なし
`pip list`	入っている一覧。`-o`（`--outdated`）は更新がある物だけ。`--format` は columns（既定）・freeze・json
`pip show 名前`	1 つの詳細。`Requires` は使う相手、`Required-by` は使われる相手。`-f` で入ったファイルも出る

`--force-reinstall` を付けると、すでに入っている版でもインストールし直します。

SAMPLE

pip install は、入っている版を残します。`pip install requests` を何度打っても、版は変わりません。上げるのは `-U` を付けた時だけです。`-U` で必ず上がるのは名前を書いたパッケージで、依存先は必要な時だけ上がります。`--upgrade-strategy` の既定が only-if-needed だからで、`eager` にすると依存先も毎回上がります。[^c1-1]

pip uninstall は、名前を書いた物だけを消します。`requests` を入れて `pip uninstall -y requests` を実行しても、連れてきた `certifi`・`idna`・`urllib3` などは残ります。残ったことは `pip list` で確かめられます。依存先も消したい時は、名前を並べるか、`-r` でファイルに並べます。[^c1-2][^c1-3][^c1-4]

requirements.txt と pip freeze

`pip freeze` は、入っているパッケージを `名前==版` の形で 1 行ずつ出します。名前の大文字小文字は区別せず、並べ替えて出ます。直接入れた物も、連れてきた依存先も、版が固定されます。

```
python -m pip freeze > requirements.txt     # いまの環境を写す
python -m pip install -r requirements.txt   # 別の環境で再現する
```

freeze は pip 自身を出しません。Python 3.12 以降は pip だけ、3.11 までは pip・setuptools・wheel・distribute が除かれます。`--all` を付けると、全部出ます。`pip list --format=freeze` は同じ形で、pip も出ます。freeze は入っている物を写すだけで、ロックファイルは作りません。[^c1-5]

requirements.txt は、1 行に 1 つの指定を書くファイルです。

SAMPLE

```
# 開発用の道具は別ファイル
-r dev.txt
-c constraints.txt
six==1.16.0
idna>=3.0,<4
tomli>=2 ; python_version < "3.11"
```

行頭の `#` と、空白の後ろに置いた `#` から行末まではコメントで、`-r` は別の requirements を取り込み、`-c` は制約ファイルを指します。行末の `\` は次の行へ続けます。`;` の後ろは環境マーカーです。条件に合わない環境では、その行は入りません。入れる順番は、書いた順に保証されません。ファイルの中では、`>` や `<` を引用符で囲みません。シェルに直接打つ時は `pip install "six>=1.0,<2"` のように囲みます。囲まないと `>` がリダイレクトとして読まれます。[^c1-6][^c1-7]

環境の再現は、次の 3 手順です。

1. 元の環境で `pip freeze > requirements.txt` を実行する
2. 新しい仮想環境を作る
3. その中で `pip install -r requirements.txt` を実行する

freeze には依存先まで載ります。`pip uninstall -r requirements.txt` を使えば、依存先も一度に消せます。

バージョン指定子と制約ファイル

指定子	意味	例（1.4.2 の場合）
`==1.4.2`	ちょうどその版（`1.4.2.0` も一致）	1.4.3 は不一致

SAMPLE

指定子	意味	例（1.4.2 の場合）
`==1.4.*`	1.4 で始まる版	1.4.9 は一致、1.5.0 は不一致
`>=1.4.2` `<2.0`	以上・未満	`>=1.4.2,<2.0` は 1.9 が一致
`!=1.4.2`	その版だけ除く	1.4.3 は一致
`~=1.4.2`	`>=1.4.2, ==1.4.*` と同じ	1.4.9 は一致、1.5.0 は不一致
`~=1.4`	`>=1.4, ==1.*` と同じ（1.4 が下限）	1.9 は一致、2.0 は不一致

コンマは「かつ」です。`~=` は、最後の桁を落とした前の部分を固定します。`~=1` のように桁が 1 つだけの指定は、使えません。末尾の 0 が違うだけの `1.4.2` と `1.4.2.0` は、同じ版です。`a1` や `rc1` のような先行版は、明示しない限り選ばれません。選ばせるには `--pre` を付けます。[^c1-8][^c1-7]

制約ファイル（`-c`）は、requirements と同じ書き方で、版の範囲だけを決めます。入れるかどうかは決めません。`six==1.16.0` を書いた制約ファイルがあっても、`pip install -c constraints.txt idna` で six は入りません。`pip install -c constraints.txt six` を実行した時に、最新の 1.17.0 ではなく 1.16.0 が入ります。制約ファイルには、名前が必須で、`-e` と extras は書けません。組織で共通の版を、一か所で決める用途に向きます。[^c1-9]

venv で仮想環境を作る [1.2]

作成・有効化・無効化

仮想環境は、プロジェクトごとに分けたパッケージの置き場です。標準ライブラリの venv で作ります。

SAMPLE

```
python -m venv .venv
source .venv/bin/activate       # macOS・Linux の bash・zsh
python -m pip install six       # この環境にだけ入る
deactivate
```

作ると、設定ファイルの `pyvenv.cfg` と、実行ファイルの置き場 `bin`（Windows は `Scripts`）ができます。パッケージは `lib/python3.14/site-packages`（Windows は `Lib\site-packages`）に入ります。pip は既定で入り、`--without-pip` を付けた時だけ入りません。3.12 以降は setuptools が入りません。3.13 以降は、環境の全体を Git の対象外にする `*` を書いた `.gitignore` も作られます。[^c1-10]

環境	有効化のコマンド
bash・zsh	`source .venv/bin/activate`
fish・csh	`source .venv/bin/activate.fish`・`activate.csh`
Windows cmd	`.venv\Scripts\activate.bat`
Windows PowerShell	`.venv\Scripts\Activate.ps1`（実行ポリシーを RemoteSigned にする必要が出ることがある）

有効化は、`bin` を PATH の先頭に足し、環境変数 `VIRTUAL_ENV` を設定し、プロンプトに `(.venv)` を付けます。有効化は必須ではありません。`.venv/bin/python` をフルパスで呼べば、その環境の Python が動きます。環境に入ったスクリプト（`pip` など）も、有効化せずにそのまま動きます。先頭行（shebang）が環境の Python への絶対パスだからです。

SAMPLE

オプション	働き
`--system-site-packages`	元の Python のパッケージも見える。`pyvenv.cfg` の `include-system-site-packages` が `true` になる
`--clear`	既存のディレクトリの中身を消してから作る
`--prompt 名前`	プロンプトの接頭辞を変える
`--copies` `--symlinks`	実行ファイルをコピーにするか、リンクにするか
`--upgrade-deps`	pip を最新にする

既に仮想環境があるディレクトリへ `python -m venv` を重ねても、入れた物は消えません。消したい時だけ `--clear` を付けます。[^c1-10]

仮想環境は、Git に入れず、使い捨てにします。ディレクトリを移動やコピーしても動きません。絶対パスが埋まっているためです。場所を変える時は、新しい場所で作り直し、requirements.txt から入れ直します。

pyvenv.cfg と sys.prefix

`pyvenv.cfg` には、作った時の Python の場所を示す `home` と、`include-system-site-packages`、`version` が書かれます。`executable` と `command` は 3.11 以降で作った環境にだけ入り、3.10 で作った環境にはありません。

仮想環境の中で動く Python は、`sys.prefix` と `sys.exec_prefix` が仮想環境のディレクトリを指します。`sys.base_prefix` と `sys.base_exec_prefix` は、元の Python を指します。仮想環境の中かどうかは、次の式で判定できます。

SAMPLE

```
import sys
print(sys.prefix != sys.base_prefix)
```

仮想環境の外では `sys.prefix == sys.base_prefix` なので、出力は `False` です。環境変数 `VIRTUAL_ENV` は、有効化しないと設定されません。有効化せずにフルパスで呼んだ Python は、仮想環境の中で動いていても `VIRTUAL_ENV` は設定されません（空の文字列でもありません）。判定には `sys.prefix` を使います。[^c1-11]

uv でプロジェクトを管理する [1.3]

uv init・add・run・sync と uv.lock

uv は、Python の版・仮想環境・パッケージ・プロジェクトをまとめて扱う道具で、プロジェクトの情報は `pyproject.toml` に書きます。

```
uv init hello          # プロジェクトを作る
uv add six             # pyproject.toml に足し、uv.lock と環境も更新
uv run python app.py   # ロックと環境を確認してから実行
uv sync                # uv.lock のとおりに環境を揃える
uv remove six          # 外す（dev の依存は --dev を付ける）
```

`uv init` は、`pyproject.toml`・`.python-version`・`README.md` を作り、Git があれば `.git` と `.gitignore` も作ります。アプリの本体は、uv 0.12 より前が `main.py` の 1 ファイルです。0.12 以降は `src/名前/__init__.py` と `[build-system]` が既定で、`--no-package` を付けると `main.py` になります。`.venv` と `uv.lock` は、`uv init` の時点では作られません。`uv run`・`uv sync`・`uv lock` を初めて実行した時にできます。[^c1-12][^c1-13]

SAMPLE

ファイル	役目
`pyproject.toml`	必要な依存の大まかな範囲を書く。`uv add six` は `six>=1.17.0` と書く
`uv.lock`	解決した正確な版を持つ。OS をまたいで使える。手では直さず、Git に入れる
`.python-version`	プロジェクトが使う Python の版
`.venv`	プロジェクトの仮想環境。Git に入れない

`uv run` は、実行の前に、`uv.lock` が `pyproject.toml` に合っているかと、環境が `uv.lock` に合っているかを確かめます。合っていなければ直します。新しい版が出ただけでは、ロックは古いと見なされません。上げたい時は `uv lock --upgrade-package 名前` を使います。`uv lock --upgrade` はロック全体を上げます（`--upgrade-package` は指定した物だけを上げ、ほかのロック済みの版は保つ）。`--locked` を付けると、ロックが古い時にエラーになります。`--frozen` を付けると、ロックを確かめずに使います。`uv lock --check` は、ロックが最新かどうかだけを調べます。[^c1-14]

`uv sync` は、ロックに無いパッケージを環境から消します（exact）。`uv run` は、足りない物を入れるだけで、余計な物を消しません（inexact）。`uv pip install` で足した物は、`uv run` の後も残り、`uv sync` で消えます。

開発用の依存は `uv add --dev 名前` で足し、`pyproject.toml` の `[dependency-groups]` の `dev` に入ります。`uv sync` は dev も入れ、`--no-dev` で外せます。`uv export --format requirements.txt` で、requirements.txt の形に書き出せます。[^c1-14][^c1-12]

SAMPLE

uv venv・uv pip・uv python install・uvx

コマンド	働き
`uv venv`	`.venv` を作る。名前は `uv venv 名前` 。`--python 3.11` で版も指定できる
`uv pip install 名前`	仮想環境に入れる。pip と同じ書き方（`-r`・`-e`・`==`）が通る
`uv python install 3.12`	uv が Python 本体を入れる。引数なしなら最新
`uvx 名前`	`uv tool run` と同じ。一時の仮想環境で道具を実行する

`uv pip` は、pip と違い、仮想環境が無いとエラーになります（`No virtual environment found`）。カレントに `.venv` があれば、自動で使います。仮想環境の外へ入れるには、`--system` を付けます。`uv venv` は、要求された Python が無ければ、自動で取ってきます。`uv python install` で入れた Python は、`python3.12` のように版つきの名前で PATH に置かれます。`python` と `python3` の名前は、既定では置かれません。欲しい時は、実験的な `--default` を付けます。[^c1-15][^c1-16]

uvx は、道具を入れずに 1 回だけ動かす用途に向きます。`uvx ruff check` のように書くと、環境がキャッシュに作られ、プロジェクトには残りません。最初の実行で最新の版を使い、その後は別の版を求めるか、キャッシュを消すか更新するまで、キャッシュの版を使います。期限が来て自動で更新されることはありません。版は `uvx ruff@0.6.0` のように指定し、`uvx ruff@latest` で最新に更新します。PATH に置いて何度も使う時は、`uv tool install ruff` です。道具の環境は、プロジェクトの環境とは混ざりません。[^c1-17]プロジェクトのコ

SAMPLE

ードや依存を読み込む道具（pytest・mypy）は、`uvx` ではなく `uv run pytest` のようにプロジェクトの環境で動かします。

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、Python 3 エンジニア認定実践試験 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

Python 3 エンジニア認定実践試験 学習用テキスト(無料サンプル)

版

2026年11月版(Pythonエンジニア育成推進協会「Python3 エンジニア認定実践試験」出題範囲（主教材『Python実践レシピ』改訂新版・2026年12月1日〜）準拠)

発行

IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ

info@shikaku-doujou.com

Python は Python Software Foundation の登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式の出題範囲（主教材の章立て）と Python 3.14 のドキュメントの内容を基に作成したもので、一般社団法人Pythonエンジニア育成推進協会との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。

SAMPLE