Engineer

Python

Sprache für den Großteil der Werkzeuge dieser Website

Was es ist

Python ist eine universell einsetzbare Programmiersprache, deren Quelltext ein Interpreter ausführt, statt ihn vorab in eine separate Binärdatei zu übersetzen. Die eigene Dokumentation bezeichnet sie als interpretiert, interaktiv und objektorientiert, mit eingebauten Modulen, Ausnahmen, dynamischer Typisierung und höheren Datentypen. Was den meisten zuerst auffällt, ist, dass Einrückung zur Syntax gehört: Ein Block besteht aus dem, was unter seiner eröffnenden Zeile eingerückt ist. Das sichtbare Layout ist damit die Struktur, die der Interpreter ausführt. Guido van Rossum begann während der Weihnachtsferien 1989 mit der Entwicklung, aufbauend auf seiner Arbeit an der Sprache ABC am CWI in Amsterdam. Er veröffentlichte Python im Februar 1991 und benannte es nach den damals von ihm gelesenen Drehbüchern von Monty Python’s Flying Circus.

Die Typisierung ist dynamisch. Während der Ausführung wird ein Name an ein Objekt gebunden; der Typ gehört zum Objekt, nicht zum Namen. Dieselbe Variable kann daher in einer Zeile eine Zahl und in der nächsten eine Zeichenkette enthalten. Die durch PEP 484 standardisierten optionalen Annotationen erlauben einer Funktion anzugeben, was sie erwartet und zurückgibt. Der PEP hält aber ausdrücklich fest, dass keine Typprüfung zur Laufzeit stattfindet und Python dynamisch typisiert bleiben wird. Separate Werkzeuge, Typprüfer und Editoren lesen die Annotationen; der Interpreter speichert sie, ohne sie durchzusetzen.

Die Sprache bringt eine umfangreiche Standardbibliothek mit, im Tutorial als „Batterien inklusive“ beschrieben: Dateien, JSON, SQLite, reguläre Ausdrücke, Unterprozesse, Argumentverarbeitung und mehr, ohne zusätzliche Installation. Alles Weitere kommt als Paket von Dritten, üblicherweise mit pip aus dem Python Package Index installiert. Weil ein Interpreter nur eine Version jedes

Pakets aufnehmen kann, enthält die Standardbibliothek auch `venv`. Es erzeugt für jedes Projekt eine eigenständige Umgebung, sodass eine Anwendung Version 1.0 einer Bibliothek verwenden kann, während eine andere 2.0 nutzt.

Neue Versionen folgen einem festen Zeitplan. Seit Python 3.9 sieht PEP 602 jeden Oktober nach etwa siebzehn Monaten Entwicklung eine Version mit neuen Funktionen vor. Jede wird fünf Jahre gepflegt; ab 3.13 bedeutet das zwei Jahre reguläre Fehlerkorrekturversionen ungefähr alle zwei Monate, danach drei Jahre ausschließlich Sicherheitskorrekturen. Python 3.15.0 erschien am 9. Oktober 2026. Python 3.10 erreichte am 1. Oktober 2026 sein Lebensende; Python 2.7, die letzte Version der zweiten Reihe, erhält seit dem 1. Januar 2020 keine Unterstützung mehr. Der Referenzinterpreter CPython besitzt seit Langem einen globalen Interpreter-Lock, der jeweils nur einem Thread erlaubt, Python-Code auszuführen. PEP 703 ergänzte eine Variante ohne diesen Lock; PEP 779 benannte 3.14 als Zeitrahmen für die Phase, in der diese Variante ohne globalen Interpreter-Lock offiziell unterstützt, aber weiterhin optional ist.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Python?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Python ist oft der kürzeste Weg von einer Frage zu einem funktionierenden Programm. Ein Skript, das eine Datei liest, eine Datenbank abfragt und einen Bericht schreibt, lässt sich allein mit der Standardbibliothek verfassen, sofort ausführen und ohne Buildschritt ändern. Deshalb verbindet Python so oft die Teile von Datenpipelines, Buildwerkzeugen und Testgerüsten.

Lesbarkeit ist ein bewusstes Entwurfsziel, kein Nebeneffekt. Der Zen of Python, PEP 20, sagt, dass Lesbarkeit zählt. PEP 8, die Stilrichtlinie für die Standardbibliothek, klärt die Fragen, über die ein Team sonst streiten würde, von der Benennung bis zur Zeile mit 79 Zeichen.

Die Flexibilität hat einen Preis, der spät fällig wird. Ein falsch geschriebenes Attribut, ein `None` anstelle einer erwarteten Zeichenkette oder ein Argument mit falschem Typ wird erst beim Ausführen dieser Zeile zum Fehler, womöglich in Produktion. Teams, die Python dauerhaft einsetzen, ergänzen deshalb die von der Sprache ausgelassenen Prüfungen: einen Linter, Annotationen mit einem Typprüfer in der verbindlichen Qualitätsprüfung und Tests, die die Codepfade tatsächlich ausführen.

Der Kalender ist ebenso wichtig wie der Code. Jede Interpreterversion hat ein festes Enddatum. Eine unangetastete Installation verliert daher nach veröffentlichtem Zeitplan die Sicherheitsunterstützung. Ein Projekt, das seine Mindestversion festlegt und innerhalb der zwei Jahre mit Fehlerkorrekturen auf eine neuere Version wechselt, tauscht einen Notfall gegen ein geplantes Upgrade pro Jahr.

Wie es in der Praxis aussieht

Im Portfolio war Python in einem dokumentierten Entwicklungsstand die Grundlage für einen datenbankgestützten Lebenslauf- und Portfoliogenerator: 41 Module mit 10.580 Zeilen erzeugten sechs Ausgabeformate von HTML und PDF bis Markdown und JSON aus einer einzigen SQLite-Datenbank mit 23 Tabellen. Eine Pipeline mit 19 Schritten baute diese Datenbank auf; jeder Schritt konnte sicher wiederholt werden. Die Laufzeitabhängigkeiten wurden auf genau zwei begrenzt, eine Vorlagen-Engine und einen PDF-Renderer. Die Begründung lautete, dass ein Dokumentgenerator nichts bewahrt hat, wenn er sich in fünf Jahren nicht mehr installieren lässt. Dafür hing damals jedes Dokument von diesem Python-Code ab. Ein anderer Weg, ein Dokument zu erzeugen, blieb nicht. Diese Zahlen beschreiben die Implementierung in jenem dokumentierten Stand.

Python gewann auch einen direkten Vergleich bei einer Pipeline für stündlichen Stromverbrauch. Sie wurde auf drei Arten gebaut: in Python, in JQ und in SQL auf PostgreSQL. Unregelmäßige Messungen wurden zu einer Zeile je Stunde und Zone mit Nettoverbrauch als Erzeugung plus Importe minus Exporte. Die mit reinen Python-Funktionen geschriebene Fassung war die schnellste. Sie verarbeitete einen Tag Daten in 34 ms und einen Datensatz mit 30 Tagen in 180 ms.

Diese Website wird mit Hugo gebaut. Im dokumentierten Werkzeugstand vom Oktober 2026 waren ihre 22 Hilfsskripte mit insgesamt 8.408 Zeilen in Python geschrieben. Alle verwendeten ausschließlich die Standardbibliothek, außer dem Skript zum Import von Zeichnungen, das Pillow zum Beschneiden benötigte. Die Datei `.tool-versions` nannte Python 3.14.7 mit der Regel, dass der Interpreter neuer, aber niemals älter sein durfte. `task doctor` prüfte dies am installierten Programm statt anhand der Datei allein. Ruff war der Linter; basedpyright prüfte Typen auf der Stufe `standard`, weil seine voreingestellte Prüfstufe 358 Befunde gemeldet hatte, fast alle zu fehlenden Annotationen in bewusst schlichtem Code. Bei seinem ersten Lauf fand der Typprüfer einen als Wörterbuchschlüssel verwendeten Wert, der `None` sein konnte: einen Defekt, den Ruff nicht beanstandet hatte.

Wo es schiefgeht

Die meisten Python-Probleme entstehen durch Code, den der Interpreter klaglos akzeptiert und der sich anders verhält, als er sich liest.

Ein Standardargument erinnert sich an frühere Aufrufe. Eine Funktion gibt eine Liste zurück, die noch Einträge früherer Aufrufe enthält. Das Ergebnis hängt dadurch von der Aufrufreihenfolge ab. Ein Standardwert wird einmal beim Ausführen der `def` -Anweisung ausgewertet. Ein veränderlicher Standardwert wie eine Liste oder ein Wörterbuch ist daher dasselbe gemeinsam genutzte Objekt für jeden Aufruf, der das Argument auslässt. Verwenden Sie `None` als Vorgabe und erzeugen Sie die Liste im Funktionsrumpf; Ruffs Regel B006 meldet dieses Muster.

```
def add_tag(tag, tags=None):
    if tags is None:
        tags = []  # a new list on every call
    tags.append(tag)
    return tags
```

Projekte verursachen gegenseitig Abhängigkeitskonflikte. Eine für ein Projekt aktualisierte Bibliothek führt in einem anderen zu Fehlern, oder `pip install` bricht mit einem Fehler zu einer extern verwalteten Umgebung ab. In beiden Fällen wurden Pakete direkt in einen gemeinsam genutzten Interpreter installiert: Die Projekte konkurrieren um dieselbe Kopie jeder Bibliothek. Unter PEP 668 darf ein Betriebssystemanbieter sein eigenes Python außerdem für pip

sperren. Geben Sie jedem Projekt mit `python -m venv .venv` eine eigene Umgebung, installieren Sie Pakete in dieser Umgebung und halten Sie die Abhängigkeiten in einer Datei fest, damit sich die Umgebung neu aufbauen lässt.

Typangaben werden für Prüfungen gehalten. Eine Funktion mit einem als `int` annotierten Parameter erhält eine Zeichenkette aus einer JSON-Datei; der Fehler zeigt sich drei Aufrufe später. Annotationen beschreiben die Absicht, der Interpreter setzt sie nicht durch. Lassen Sie einen Typprüfer in der verbindlichen Qualitätsprüfung laufen, damit Unstimmigkeiten zwischen annotierten Codeteilen den Build scheitern lassen. Validieren Sie Daten dort, wo sie ins Programm gelangen: Keine Annotation kann für Eingabedaten bürgen.

Ein Interpreter erreicht sein Support-Ende. Ein Server behält seine bei der Installation gewählte Version, bis Sicherheitsveröffentlichungen enden und Pakete sie aus ihrem unterstützten Bereich streichen. Jede Version mit neuen Funktionen bekommt fünf Jahre; die von 3.10 endeten am 1. Oktober 2026. Nennen Sie die Mindestversion in einer von einer Prüfung gelesenen Datei, aktualisieren Sie innerhalb der zwei Fehlerkorrekturjahre jeder Version und führen Sie vor dem Wechsel die Testsuite mit der neuen Version aus.

Weiterführende Lektüre

- Python-Dokumentation — Tutorial, Bibliotheksreferenz und Sprachreferenz für die aktuelle Version.
- Das Python-Tutorial — die Sprache von Grund auf, einschließlich der Warnung vor veränderlichen Standardargumenten.
- PEP 8: Stilrichtlinie für Python-Code — die üblichen Konventionen für Anordnung und Benennung von Python-Code.
- PEP 602: Jährlicher Veröffentlichungszyklus für Python — warum jeden Oktober eine Version mit neuen Funktionen erscheint und wie lange sie unterstützt wird.
- PEP 484: Typangaben — die Annotationssyntax und warum der Interpreter die Prüfung anderen Werkzeugen überlässt.
- Virtuelle Umgebungen und Pakete — pro Projekt eine Umgebung mit `venv` anlegen und darin mit pip installieren.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Python?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/python/