Der Ursprung von Python

Im Bereich Data Science stoßen Anwender immer wieder auf zwei Programmiersprachen: **Python** und **R**. Während R überwiegend zur Berechnung und Erstellung von Grafiken verwendet wird, dient Python als Allzweck-Werkzeug. Besonders Letztere kommt in der Praxis immer mehr Beachtung zu.

Dies ist auch bedingt durch eine Vielzahl an verfügbaren Bibliotheken, die, je nach Anwendungsfall, Python um Funktionen erweitert. Dies sind z.B:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- ….

Python ist eine universelle, üblicherweise interpretierte, höhere Programmiersprache. Sie hat den Anspruch, einen gut lesbaren, knappen Programmierstil zu fördern. So werden beispielsweise Blöcke nicht durch geschweifte Klammern, sondern durch Einrückungen strukturiert.

Python unterstützt mehrere Programmierparadigmen, z. B. die objektorientierte, die aspektorientierte und die funktionale Programmierung. Ferner bietet es eine dynamische Typisierung. Wie viele dynamische Sprachen wird Python oft als Skriptsprache genutzt. Die Sprache weist ein offenes, gemeinschaftsbasiertes Entwicklungsmodell auf, das durch die gemeinnützige Python Software Foundation gestützt wird, die die Definition der Sprache in der Referenzumsetzung CPython pflegt.

1 Inhaltsverzeichnis

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm

2 Entwicklungsgeschichte

Die Sprache wurde Anfang der 1990er Jahre von Guido van Rossum am Centrum Wiskunde & Informatica in Amsterdam als Nachfolger für die Programmier-Lehrsprache ABC entwickelt und war ursprünglich für das verteilte Betriebssystem Amoeba gedacht.

Der Name geht nicht, wie das Logo vermuten lässt, auf die gleichnamige Schlangengattung Python zurück, sondern bezog sich ursprünglich auf die englische Komikertruppe Monty Python. In der Dokumentation finden sich daher auch einige Anspielungen auf Sketche aus dem Flying Circus. Trotzdem etablierte sich die Assoziation zur Schlange, was sich unter anderem in der Programmiersprache Cobra sowie dem Python-Toolkit „Boa“ äußert. Die erste Vollversion erschien im Januar 1994 unter der Bezeichnung Python 1.0. Gegenüber früheren Versionen wurden einige Konzepte der funktionalen Programmierung implementiert, die allerdings später wieder aufgegeben wurden. Von 1995 bis 2000 erschienen neue Versionen, die fortlaufend als Python 1.1, 1.2 etc. bezeichnet wurden.

Python 2.0 erschien am 16. Oktober 2000. Neue Funktionen umfassten eine voll funktionsfähige Garbage Collection (automatische Speicherbereinigung) und die Unterstützung für den Unicode-Zeichensatz. In Version 2.6 wurde eine Hilfe eingebaut, mit der angezeigt werden kann, welche Code-Sequenzen vom Nachfolger Python 3 nicht mehr unterstützt werden und daher in darauf aufbauenden Versionen nicht mehr lauffähig sind.

Python 3.0 (auch Python 3000) erschien am 3. Dezember 2008 nach längerer Entwicklungszeit. Es beinhaltet einige tiefgreifende Änderungen an der Sprache, etwa das Entfernen von Redundanzen bei Befehlssätzen und veralteten Konstrukten. Da Python 3.0 hierdurch teilweise inkompatibel zu früheren Versionen ist, beschloss die Python Software Foundation, Python 2.7 parallel zu Python 3 bis Ende 2019 weiter mit neuen Versionen zu unterstützen.

Die Unterstützung für Python 2 wurde 2020 beendet. Die letzte 2er-Version war die 2.7.18 vom 20. April 2020. Seit diesem Datum wird Python 2 nicht mehr unterstützt.
Es gibt aber vielfältige und umfangreiche Dokumentationen zum Umstieg und auch Tools, die bei der Migration helfen oder es ermöglichen, Code zu schreiben, der mit Python 2 und 3 funktioniert.

3 Ziele

Python wurde mit dem Ziel größter Einfachheit und Übersichtlichkeit entworfen. Dies wird vor allem durch zwei Maßnahmen erreicht. Zum einen kommt die Sprache mit relativ wenigen Schlüsselwörtern aus. Zum anderen ist die Syntax reduziert und auf Übersichtlichkeit optimiert. Dadurch lassen sich Python-basierte Skripte deutlich knapper formulieren als in anderen Sprachen.

Van Rossum legte bei der Entwicklung großen Wert auf eine Standardbibliothek, die überschaubar und leicht erweiterbar ist. Dies war Ergebnis seiner schlechten Erfahrung mit der Sprache ABC, in der das Gegenteil der Fall ist. Dieses Konzept ermöglicht, in Python Module aufzurufen, die in anderen Programmiersprachen geschrieben wurden, etwa um Schwächen von Python auszugleichen. Beispielsweise können für zeitkritische Teile Routinen in maschinennäheren Sprachen wie C aufgerufen werden. Umgekehrt lassen sich mit Python Module und Plug-ins für andere Programme schreiben, die die entsprechende Unterstützung bieten.

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm

Python ist eine Multiparadigmensprache. Das bedeutet, Python zwingt den Programmierer nicht zu einem einzigen Programmierstil, sondern erlaubt, das für die jeweilige Aufgabe am besten geeignete Paradigma zu wählen. Objektorientierte und strukturierte Programmierung werden vollständig unterstützt, funktionale und aspektorientierte Programmierung werden durch einzelne Elemente der Sprache unterstützt.
Die Freigabe nicht mehr benutzter Speicherbereiche erfolgt durch Referenzzählung. Datentypen werden dynamisch verwaltet, eine automatische statische Typprüfung wie z. B. bei C++ gibt es nicht.

3.1 Programmierparadigmen:

Ein Programmierparadigma ist ein fundamentaler Programmierstil. „*Der Programmierung liegen je nach Design der einzelnen Programmiersprache verschiedene Prinzipien zugrunde. Diese sollen den Entwickler bei der Erstellung von ‚gutem Code' unterstützen, in manchen Fällen sogar zu einer bestimmten Herangehensweise bei der Lösung von Problemen zwingen*".
Darüber hinaus unterstützt Python mehrere Programmierparadigmen, wie die objektorientierte, aspektorientierte und funktionale Programmierung.

3.1.1 Objektorientiert

Klassen sind instanziierbare Module und Grundelemente in der objektorientierten Programmierung. Nach dem objektorientierten Programmierparadigma werden Objekte mit Daten und den darauf arbeitenden Routinen zu Einheiten zusammengefasst. Im Unterschied dazu werden beim prozeduralen Paradigma die Daten von den die Objekte verarbeitenden Routinen getrennt gehalten. Ein Computerprogramm ist realisiert als eine Menge interagierender Objekte.

Objektorientierte Programmierung lässt sich gut mit der ereignisorientierten Programmierung kombinieren, z. B. bei der Programmierung interaktiver, grafischer Benutzeroberflächen.

3.1.2 Aspektorientiert

Aspektorientierte Programmierung (AOP) ist ein Programmierparadigma für die objektorientierte Programmierung, um generische Funktionalitäten über mehrere Klassen hinweg zu verwenden (Cross-Cutting Concern). Logische Aspekte eines Anwendungsprogramms werden dabei von der eigentlichen Geschäftslogik getrennt. Typische Anwendungsbeispiele sind Transaktionsverwaltung, Auditfähigkeit und Loggingverhalten.

3.1.3 Funktionale Programmierung

Funktionale Programmierung ist ein Programmierparadigma, in dem Funktionen nicht nur definiert und angewendet werden können, sondern auch wie Daten miteinander verknüpft, als Parameter verwendet und als Funktionsergebnisse auftreten können. Dadurch kann ein funktionales Programm sehr weitreichend neue Berechnungsvorschriften zur Laufzeit zusammensetzen und anwenden. Programmiersprachen, die funktionale Programmierung ermöglichen, werden als funktionale Programmiersprachen bezeichnet.

3.2 Dynamische versus statischer Typisierung

Bei der dynamischen Typisierung finden Typ-Prüfungen (etwa des Datentyps von Variablen) vorrangig zur Laufzeit eines Programms statt. Im Gegensatz wird bei der statischen Typisierung die Typ-Prüfung bereits zum Zeitpunkt der Kompilierung durchgeführt.
Skriptsprachen wie JavaScript, Python und Ruby verwenden die dynamische Typisierung.

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm

Bei der statischen Typisierung wird im Gegensatz zur dynamischen Typisierung der Datentyp von Variablen und anderen Programmbausteinen schon während der Kompilierung festgelegt. Dies kann durch Typinferenz oder durch explizite Deklaration geschehen.
Python als Interpreter-Sprache unterstützt die dynamische Typisierung, aber seit Python 3.6 wird auch die statische Typisierung zusätzlich unterstützt.
Ab Python 3.6 deklarieren Sie einen Variablentyp wie folgt: *variable_name: type*

3.3 Compiler und Interpreter

Compiler und Interpreter sind Implementierungsformen von Software. Generell geht es beim Compilieren und Interpretieren darum, den Quelltext, der mit einer höheren Programmiersprache (zum Beispiel C++, C# oder Java) geschrieben wurde, in Maschinenbefehle umzusetzen. Das bedeutet, die lesbaren Programmierbefehle müssen in weniger komplexe Instruktionen übersetzt werden, damit der Prozessor diese ausführen kann.

In der Theorie kann der Quelltext jeder Programmiersprache sowohl mit einem Compiler als auch mit einem Interpreter implementiert werden. In der Praxis ist die Implementierung trotzdem festgelegt. Die Verwendung einer Programmiersprache legt den Einsatz eines Compilers oder Interpreters fest.

3.3.1 Compiler

Der Compiler ist ein Programm, der aus dem Quelltext das eigentliche Programm erstellt. Der Quelltext könnte zum Beispiel in C oder C++ geschrieben sein. Einzelne Anweisungen aus dem Quelltext werden in Folgen von Maschinenanweisungen übersetzt. Nach dem Compilieren wird das Programm erzeugt, in dem sich ausführbarer Code befindet. Bei der Ausführung des Programms werden diese Anweisungen "direkt" vom Prozessor ausgeführt. Jede Maschinenanweisung entspricht dabei einer festgelegten Bitfolge. Innerhalb des Prozessors wird diese Bitfolge verarbeitet. Der ausführbare Code verarbeitet auch Eingaben und erzeugt die Ausgabe.

Beim Compilieren von Software, legt der Compiler fest, welche Instruktionen in welcher Reihenfolge an den Prozessor geschickt werden. Wenn diese Instruktionen nicht auf andere warten müssen, kann der Prozessor sogar mehrere davon parallel verarbeiten.

Sobald ein neues Betriebssystem oder ein neuer Prozessor zum Einsatz kommt, muss der Quelltext neu compiliert werden. In der Praxis ist es so, dass Prozessoren und Betriebssysteme einen Kompatibilitätsmodus haben, so das alte Programme auch auf neuen Plattformen laufen. So müssen compilierte Programme nicht immer neu compiliert werden. Aber die Kompatibilität hat natürlich ihre Grenzen.

Für jede Programmiersprache (m) braucht es für jeden Prozessor (n) einen eigenen Compiler (m x n). Typische Programmiersprachen mit Compiler sind Pascal, Modula, COBOL, Fortran, C und C++.

Vorteile:

- Die Übersetzung in ausführbaren Code ist äußerst effizient und optimiert den generierten Code.
- Compilierte Programme arbeiten sehr schnell, was sich besonders bei lang laufenden Programmen lohnt.

Nachteile:

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm

- Der Aufwand bei der Software-Entwicklung steigt durch das Compilieren, was einiges an Zeit und Ressourcen in Anspruch nimmt.
- So muss bei jeder Quelltext-Änderung erneut compiliert werden, wenn das Programm getestet werden soll.
- Compiler muss den eingesetzten Prozessor unterstützen.

3.3.2 Interpreter

Der Quelltext wird von der Programmiersprache in einen Hardware-unabhängigen Bytecode umgewandelt. Der Interpreter setzt diesen Bytecode in Maschinenanweisungen um. Den Interpreter muss man sich dabei als eine virtuelle Maschine (VM) vorstellen, in der das Programm läuft. Hierbei darf man den Interpreter nicht mit einer virtuellen Maschine der Virtualisierungstechnik verwechseln. Es handelt sich dabei um zwei unterschiedliche Techniken mit der gleichen Bezeichnung.

Python wird interpretiert - Python wird zur Laufzeit vom Interpreter verarbeitet. Das Programm muss nicht kompilieren werden, bevor es ausgeführt wird.
Das bedeutet, daß bei Programmaufruf zunächst ein sog. „Interpreter“ anstelle des herkömmlichen Compilers aus der Eingabedatei mit der Endung „*.py“ eine Datei mit der Endung „Dateiname.pyc“ erzeugt, welche dann ausgeführt (VM) wird. Dies ist mit einer gewissen Verzögerung verbunden. Python-Programme sind nicht so schnell in der Ausführung wie klassisch kompilierte Programme. Jedoch ist der Unterschied bei steigender Computerleistung sinkend. Die erforderliche Arbeitszeit für die Programmerstellung mit Python ist jedoch deutlich geringer. Aus diesem Grund wird auch empfohlen, das „Rapid Prototyping“ von zeitkritischen Anwendungen zunehmend Richtung Python zu verlagern.
Dies ist ähnlich wie bei PERL, PHP und BASIC.

Vorteile:

- Bei der Entwicklung der Software kann man sofort testen, was das Debugging (Fehlersuche) erleichtert.
- Der verwendete ausführbare Code wird erst zur Laufzeit generiert.

Nachteile:

- Generell sind interpretierte Programme langsamer und ineffizient.
- Es müssen immer die selben Programmteile, wie zum Beispiel Schleifen und Funktionen, erneut übersetzt werden.

3.4 Bytecode

Der Bytecode ist in der Informatik eine Sammlung von Befehlen für eine virtuelle Maschine. Bei Kompilierung eines Quelltextes mancher Programmiersprachen oder Umgebungen – wie beispielsweise Java – wird nicht direkt Maschinencode, sondern ein Zwischencode, der Bytecode, erstellt. Dieser Code ist in der Regel unabhängig von realer Hardware. Er entsteht als Resultat einer semantischen Analyse des Quelltexts und ist im Vergleich zu diesem oft relativ kompakt und wesentlich effizienter interpretierbar als der originale Quelltext.

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm

3.5 Pseudocode

P-Code ist der Befehlssatz einer Pseudo-Maschine (oder P-Maschine), also einer virtuellen CPU, die P-Code als Maschinensprache ausführt. Der P-Code war ein Computer- bzw. CPU-unabhängiger Code und war Teil der Entwicklungsumgebung UCSD Pascal. Die Umsetzung in die Maschinensprache der CPU erfolgt durch den Interpreter der P-Maschine. Man kann den P-Code und das Konzept der virtuellen Maschine als geistigen Vorläufer der heutigen Java Virtual Machine betrachten.

4 Weitere Informatinen

Links:

https://www.bigdata-insider.de/was-ist-python-a-730480/

Quellen: https://de.wikipedia.org/wiki/Python_(Programmiersprache)
https://compamind.de/knowhow/python/
https://www.elektronik-kompendium.de/sites/com/1705231.htm