Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[31]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.[32][33]

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0.[34] Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2.[35]

Python consistently ranks as one of the most popular programming languages, and has gained widespread use in the machine learning community.[36][37][38][39]

History

The designer of Python, Guido van Rossum, at OSCON 2006

Main article: History of Python

Python was conceived in the late 1980s[40] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL,[41] capable of exception handling and interfacing with the Amoeba operating system.[10] Its implementation began in December 1989.[42] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[43] In January 2019, active Python core developers elected a five-member Steering Council to lead the project.[44][45]

Python 2.0 was released on 16 October 2000, with many major new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support.[46] Python 3.0, released on 3 December 2008, with many of its major features backported to Python 2.6.x[47] and 2.7.x. Releases of Python 3 include the 2to3 utility, which automates the translation of Python 2 code to Python 3.[48]

Python 2.7's end-of-life was initially set for 2015, then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[49][50] No further security patches or other improvements will be released for it.[51][52] Currently only 3.8 and later are supported (2023 security issues were fixed in e.g. 3.7.17, the final 3.7.x release[53]). While Python

2.7 and older is officially unsupported, a different unofficial Python implementation, PyPy, continues to support Python 2, i.e. "2.7.18+" (plus 3.9 and 3.10), with the plus meaning (at least some) "backported security updates".[54]

In 2021 (and again twice in 2022), security updates were expedited, since all Python versions were insecure (including 2.7[55]) because of security issues leading to possible remote code execution[56] and web-cache poisoning.[57] In 2022, Python 3.10.4 and 3.9.12 were expedited[58] and 3.8.13, because of many security issues.[59] When Python 3.9.13 was released in May 2022, it was announced that the 3.9 series (joining the older series 3.8 and 3.7) would only receive security fixes in the future.[60] On 7 September 2022, four new releases were made due to a potential denial-of-service attack: 3.10.7, 3.9.14, 3.8.14, and 3.7.14.[61][62]

As of October 2023, Python 3.12 is the stable release, and 3.12 and 3.11 are the only versions with active (as opposed to just security) support. Notable changes in 3.11 from 3.10 include increased program execution speed and improved error reporting.[63]

Python 3.12 adds syntax (and in fact every Python since at least 3.5 adds some syntax) to the language, the new (soft) keyword type (recent releases have added a lot of typing support e.g. new type union operator in 3.10), and 3.11 for exception handling, and 3.10 the match and case (soft) keywords, for structural pattern matching statements. Python 3.12 also drops outdated modules and functionality, and future versions will too, see below in Development section.

Python 3.11 claims to be between 10 and 60% faster than Python 3.10, and Python 3.12 adds another 5% on top of that. It also has improved error messages, and many other changes.

Since 27 June 2023, Python 3.8 is the oldest supported version of Python (albeit in the 'security support' phase), due to Python 3.7 reaching end-of-life.[64]

Design philosophy and features

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming (including metaprogramming[65] and metaobjects).[66] Many other paradigms are supported via extensions, including design by contract[67][68] and logic programming.[69]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[70] It uses dynamic name resolution (late binding), which binds method and variable names during program execution.

Its design offers some support for functional programming in the Lisp tradition. It has filter,mapandreduce functions; list comprehensions, dictionaries, sets, and generator expressions.[71] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[72]

Its core philosophy is summarized in the Zen of Python (PEP 20), which includes aphorisms such as:[73]

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Readability counts.

However, Python features regularly violate these principles and received criticism for adding unnecessary language bloat.[74][75] Responses to these criticisms are that the Zen of Python is a guideline rather than a rule.[76] The addition of some new features had been so controversial that Guido van Rossum resigned as Benevolent Dictator for Life following vitriol over the addition of the assignment expression operator in Python 3.8.[77][78]

Nevertheless, rather than building all of its functionality into its core, Python was designed to be highly extensible via modules. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[40]

Python claims to strive for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it." philosophy.[73] In practice, however, Python provides many ways to achieve the same task. There are, for example, at least three ways to format a string literal, with no certainty as to which one a programmer should use.[79] Alex Martelli, a Fellow at the Python Software Foundation and Python book author, wrote: "To describe something as 'clever' is not considered a compliment in the Python culture."[80]

Python's developers usually strive to avoid premature optimization and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[81] Execution speed can be improved by moving speed-critical functions to extension modules written in languages such as C, or by using a just-in-time compiler like PyPy. It is also possible to cross-compile to other languages, but it either doesn't provide the full speed-up that might be expected, since Python is a very dynamic language, or a restricted subset of Python is compiled, and possibly semantics are slightly changed.[82]

Python's developers aim for it to be fun to use. This is reflected in its name—a tribute to the British comedy group Monty Python[83]—and in occasionally playful approaches to tutorials and reference materials, such as the use of the terms "spam" and "eggs" (a reference to a Monty Python sketch) in examples, instead of the often-used "foo" and "bar".[84][85]A common neologism in the Python community is pythonic, which has a wide range of meanings related to program style. "Pythonic" code may use Python idioms well, be natural or show fluency in the language, or conform with Python's minimalist philosophy and emphasis on readability. Code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.[86][87]

Syntax and semantics

Main article: Python syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered and often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but rarely used. It has fewer syntactic exceptions and special cases than C or Pascal.[88]

Indentation

Main article: Python syntax and semantics § Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[89] Thus, the program's visual structure accurately represents its semantic structure.[90] This feature is sometimes termed the off-side rule. Some other languages use indentation this way; but in most, indentation has no semantic meaning. The recommended indent size is four spaces.[91]

Statements and control flow

Python's statements include:

The assignment statement, using a single equals sign =

The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if)

The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block

The while statement, which executes a block of code as long as its condition is true

The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses (or new syntax except* in Python 3.11 for exception groups[92]); it also ensures that clean-up code in a finally block is always run regardless of how the block exits

The raise statement, used to raise a specified exception or re-raise a caught exception

The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming

The def statement, which defines a function or method

The with statement, which encloses a code block within a context manager (for example, acquiring a lock before it is run, then releasing the lock; or opening and closing a file), allowing resource-acquisition-is-initialization (RAII)-like behavior and replacing a common try/finally idiom[93]

The break statement, which exits a loop

The continue statement, which skips the rest of the current iteration and continues with the next

The del statement, which removes a variable—deleting the reference from the name to the value, and producing an error if the variable is referred to before it is redefined

The pass statement, serving as a NOP, syntactically needed to create an empty code block

The assert statement, used in debugging to check for conditions that should apply

The yield statement, which returns a value from a generator function (and also an operator); used to implement coroutines

The return statement, used to return a value from a function

The import and from statements, used to import modules whose functions or variables can be used in the current program

The assignment statement (=) binds a name as a reference to a separate, dynamically allocated object. Variables may subsequently be rebound at any time to any object. In Python, a variable name is a generic reference holder without a fixed data type; however, it always refers to some object with a type. This is called dynamic typing—in contrast to statically-typed languages, where each variable may contain only a value of a certain type.

Python does not support tail call optimization or first-class continuations, and, according to Van Rossum, it never will.[94][95] However, better support for coroutine-like functionality is provided by extending Python's generators.[96] Before 2.5, generators were lazy iterators; data was passed unidirectionally out of the generator. From Python 2.5 on, it is possible to pass data back into a generator function; and from version 3.3, it can be passed through multiple stack levels.[97]

Expressions

Python's expressions include:

The +, -, and * operators for mathematical addition, subtraction, and multiplication are similar to other languages, but the behavior of division differs. There are two types of divisions in Python: floor division (or integer division) // and floating-point/division.[98] Python uses the ** operator for exponentiation.

Python uses the + operator for string concatenation. Python uses the * operator for duplicating a string a specified number of times.

The @ infix operator. It is intended to be used by libraries such as NumPy for matrix multiplication.[99][100]

The syntax :=, called the "walrus operator", was introduced in Python 3.8. It assigns values to variables as part of a larger expression.[101]

In Python, == compares by value. Python's is operator may be used to compare object identities (comparison by reference), and comparisons may be chained—for example, a <= b <= c.

Python uses and, or, and not as Boolean operators.

Python has a type of expression called a list comprehension, as well as a more general expression called a generator expression.[71]

Anonymous functions are implemented using lambda expressions; however, there may be only one expression in each body.

Conditional expressions are written as x if c else y[102] (different in order of operands from the c ? x : y operator common to many other languages).

Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples, written as (1, 2, 3), are immutable and thus can be used as keys of dictionaries, provided all of the tuple's elements are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but produces a new tuple containing the elements of both. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t—thereby effectively "modifying the contents" of t while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.[103]

Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned (to a variable, writable property, etc.) are associated in an identical manner to that forming tuple literals—and, as a whole, are put on the left-hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions; when iterated through them, it assigns each of the produced values to the corresponding expression on the left.[104]

Python has a "string format" operator % that functions analogously to printf format strings in C—e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 2.6+ and 3+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": spam = "blah"; eggs = 2; f'spam={spam} eggs={eggs}'.[105]

Strings in Python can be concatenated by "adding" them (with the same operator as for adding integers and floats), e.g. "spam" + "eggs" returns "spameggs". If strings contain numbers, they are added as strings rather than integers, e.g. "2" + "2" returns "22".

Python has various string literals:

Delimited by single or double quotes; unlike in Unix shells, Perl, and Perl-influenced languages, single and double quotes work the same. Both use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".[105]

Triple-quoted (beginning and ending with three single or double quotes), which may span multiple lines and function like here documents in shells, Perl, and Ruby.

Raw string varieties, denoted by prefixing the string literal with r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. (Compare "@-quoting" in C#.)

Python has array index and array slicing expressions in lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted—for example, a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

List comprehensions vs. for-loops

Conditional expressions vs. if blocks

The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements

Statements cannot be a part of an expression—so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code, but if c = 1: ... causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, Ruby).[106] Python also provides methods, often called dunder methods (due to their names beginning and ending with double-underscores), to allow userdefined classes to modify how they are handled by native operations including length, comparison, in arithmetic operations and type conversion.[107]

Typing

The standard type hierarchy in Python 3

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that it is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes (both using the same syntax): old-style and new-style,[108] current Python versions only support the semantics new style.

Python supports optional type annotations.[4][109] These annotations are not enforced by the language, but may be used by external tools such as mypy to catch errors.[110][111] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[112]