Python2

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[59] and metaobjects [magic methods]).[60] Many other paradigms are supported via extensions, including design by contract[61][62] and logic programming.[63]

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management.[64] 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.[65] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[66]

Its core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[67]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

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.[39]

Python strives 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.[67] 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."[68]

Python's developers 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.[69] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C; or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

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[70]—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a Monty Python sketch) instead of the standard foo and bar.[71][72]

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*.[73][74]

Python users and admirers, especially those considered knowledgeable or experienced, are often referred to as *Pythonistas*.[75][76]

Syntax and semantics[edit]

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.[77]

Indentation[edit]

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.[78] Thus, the program's visual structure accurately represents its semantic structure.[79] 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.[80]

Statements and control flow[edit]

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; 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[81]
- The `break` statement, which exits a loop
- The `continue` statement, which skips 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` statement, 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.[82][83] However, better support for coroutine-like functionality is provided by extending Python's generators.[84] 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.[85]