SUBJECT-Python (programming language)

bca –ii yr

Python	
	
Paradigm	Multi-paradigm: object-oriented,[1] procedural (imperative), functional, structured, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	20 February 1991; 33 years ago[2]
Stable release	3.13.0 / 7 October 2024; 6 days ago
Typing discipline	duck, dynamic, strong;[3] optional type annotations (since 3.5, but those hints are ignored, except with unofficial tools)[4]
OS	**Tier 1**: 64-bit Linux, macOS; 64- and 32-bit Windows 10+[5] **Tier 2**: E.g. 32-bit WebAssembly (WASI) **Tier 3**: 64-bit Android,[6] iOS, FreeBSD, and (32-bit) Raspberry Pi OS Unofficial (or has been known to work): Other Unix-like/BSD variants) and a few other platforms[7][8][9]
License	Python Software Foundation License
Filename extensions	.py, .pyw, .pyz,[10] .pyi, .pyc, .pyd
Website	python.org

Major implementations
CPython, PyPy, Stackless Python, MicroPython, CircuitPython, IronPython, Jython
Dialects
Cython, RPython, Starlark[11]
Influenced by
ABC,[12] Ada,[13] ALGOL 68,[14] APL,[15] C,[16] C++,[17] CLU,[18] Dylan,[19] Haskell,[20][15] Icon,[21] Lisp,[22] Modula-3,[14][17] Perl,[23] Standard ML[15]
Influenced
Apache Groovy, Boo, Cobra, CoffeeScript,[24] D, F#, GDScript, Go, JavaScript,[25][26] Julia,[27] Mojo,[28] Nim, Ring,[29] Ruby,[30] Swift[31]
• Python Programming at Wikibooks

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[32]

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.[33][34]

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.[35] 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.[36]

Python consistently ranks as one of the most popular programming languages, and has gained widespread use in the machine learning community.[37][38][39][40]

History

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

Python was invented in the late 1980s[41] 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,[42] capable of exception handling and interfacing with the Amoeba operating system.[12] Its implementation began in December 1989.[43] 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" (BDFL), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker[44] (he has since come out of retirement and is self-titled "BDFL-emeritus"). In January 2019, active Python core developers elected a five-member Steering Council to lead the project.[45][46]

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.[47] Python 3.0 was released on 3 December 2008, with many of its major features backported to Python 2.6.x[48] and 2.7.x. Releases of Python 3 include the `2to3` utility, which automates the translation of Python 2 code to Python 3.[49]

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.[50][51] No further security patches or other improvements will be released for it.[52][53] Currently only 3.9 and later are supported (2023 security issues were fixed in e.g. 3.7.17, the final 3.7.x release[54]). 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 3.10), with the plus meaning (at least some) "backported security updates".[55]

In 2021 (and again twice in 2022, and in September 2024 for Python 3.12.6 down to 3.8.20), security updates were expedited, since all Python versions were insecure (including 2.7[56]) because of security issues leading to possible remote code execution[57] and web-cache poisoning.[58] In 2022, Python 3.10.4 and 3.9.12 were expedited[59] and 3.8.13, because of many security issues.[60] 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.[61] 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.[62][63]

Every Python release since 3.5 has added some syntax to the language. 3.10 added the `|` union type operator[64] and the `match` and `case` keywords (for structural pattern matching statements). 3.11 expanded exception handling functionality. Python 3.12 added the new keyword `type`. Notable changes in 3.11 from 3.10 include increased program execution speed and improved error reporting.[65] 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.

Python 3.13 introduces more syntax for types, a new and improved interactive interpreter (REPL), featuring multi-line editing and color support; an incremental garbage collector (producing shorter pauses for collection in programs with a lot of objects, and addition to the improved speed in 3.11 and 3.12), and an *experimental* just-in-time (JIT) compiler (such features, can/needs to be enabled specifically for the increase in speed),[66] and an *experimental* free-threaded build mode, which disables the global interpreter lock (GIL), allowing threads to run more concurrently, that latter feature enabled with `python3.13t` or `python3.13t.exe`.

Python 3.13 introduces some change in behavior, i.e. new "well-defined semantics", fixing bugs (plus many removals of deprecated classes, functions and methods, and removed some of the C API and outdated modules): "The [old] implementation of `locals()` and `frame.f_locals` is slow, inconsistent and buggy [and it has] has many corner cases and oddities. Code that works around those may need to be changed. Code that uses `locals()` for simple templating, or print debugging, will continue to work correctly."[67]

Since 7 October 2024, Python 3.13 is the latest stable release, and 3.13 and 3.12 are the only versions with active (as opposed to just security) support and Python 3.9 is the oldest supported version of Python (albeit in the 'security support' phase), due to Python 3.8 reaching end-of-life.[68] Starting with 3.13, it and later versions have 2 years of full support (up from one and a half); followed by 3 years of security support (for same total support as before).

Some (more) standard library modules and many deprecated classes, functions and methods, will be removed in Python 3.15 or 3.16.[69][70]

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[71] and metaobjects).[72] Many other paradigms are supported via extensions, including design by contract[73][74] and logic programming.[75] Python is known as a *glue language*,[76] able to work very well with many other languages with ease of access.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[77] 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`, `map` and `reduce` functions; list comprehensions, dictionaries, sets, and generator expressions.[78] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[79]

Its core philosophy is summarized in the Zen of Python (PEP 20), which includes aphorisms such as:[80]

- 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 have received criticism for adding unnecessary language bloat.[81] Responses to these criticisms are that the Zen of Python is a guideline rather than a rule.[82] 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.[83][84]

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.[41]

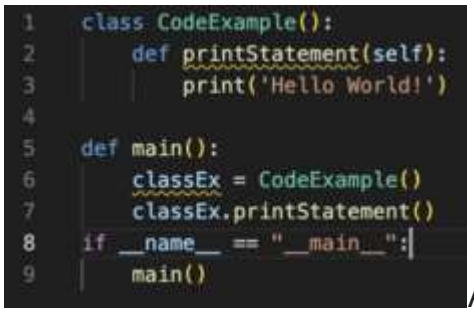
Python claims to strive for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it." philosophy.[80] 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.[85] 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."[86]

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.[87] 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.[88]

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[89]—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".[90][91] 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*.[92]

Syntax and semantics



An example of Python code and indentation

Example of C# code with curly braces and semicolons

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.[93]

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.[94] Thus, the program's visual structure accurately represents its semantic structure.[95] 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.[96]

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[97]); 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[98]
- 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 `match` and `case` statements, an analog of the switch statement construct, that compares an expression against one or more cases as a control-of-flow measure.

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.[99][100] However, better support for coroutine-like functionality is provided by extending Python's generators.[101] 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.[102]

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.[103] 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.[104][105]
- The syntax `:=`, called the "walrus operator", was introduced in Python 3.8. It assigns values to variables as part of a larger expression.[106]
- 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 named a *list comprehension*, and a more general expression named a *generator expression*.[78]
- 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`[107] (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.[108]

- 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.[109]
- 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}'`.[110]
- 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".[110]
  - 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

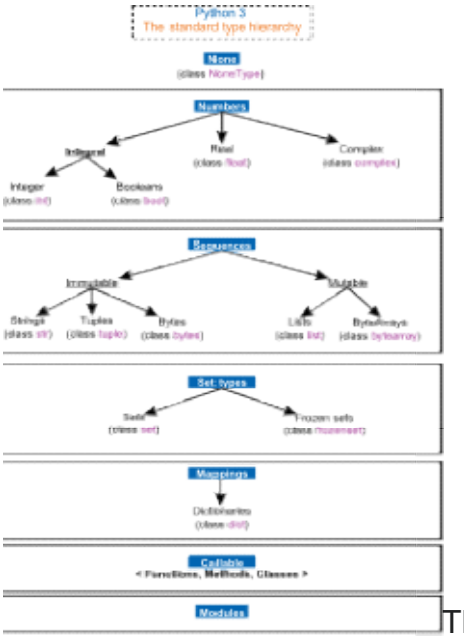
- The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements

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.

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).[111] Python also provides methods, often called *dunder methods* (due to their names beginning and ending with double-underscores), to allow user-defined classes to modify how they are handled by native operations including length, comparison, in arithmetic operations and type conversion.[112]

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*;[113] current Python versions only support the semantics of the new style.

Python supports optional type annotations.[4][114] These annotations are not enforced by the language, but may be used by external tools such as mypy to catch errors.[115][116] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[117]

Summary of Python 3's built-in types

Type	Mutability	Description	Syntax examples
`bool`	immutable	Boolean value	`True` `False`

`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')`
			`bytearray(b"Some ASCII")`
			`bytearray([119, 105, 107, 105])`

`bytes`	immutable	Sequence of bytes	`b'Some ASCII'`
			`b"Some ASCII"`
			`bytes([119, 105, 107, 105])`

`complex`	immutable	Complex number with real and imaginary parts	`3+2.7j` `3 + 2.7j`
`dict`	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	`{'key1': 1.0, 3: False}` `{}`
`types.EllipsisType`	immutable	An ellipsis placeholder to be used as an index	`...` `Ellipsis`

		in NumPy arrays	
`float`	immutable	Double-precision floating-point number. The precision is machine-dependent but in practice is generally implemented as a 64-bit IEEE 754 number with 53 bits of precision.[118]	`1.33333`
`frozenset`	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`frozenset([4.0, 'string', True])`
`int`	immutable	Integer of unlimited magnitude[119]	`42`
`list`	mutable	List, can contain mixed types	`[4.0, 'string', True]` `[]`
`types.NoneType`	immutable	An object representing the absence of a value, often called null in other languages	`None`
`types.NotImplementedType`	immutable	A placeholder that can be returned from overloaded operators to indicate unsupported operand types.	`NotImplemented`
`range`	immutable	An *immutable sequence* of numbers commonly used for looping a specific number of times in `for` loops[120]	`range(-1, 10)` `range(10, -5, -2)`
`set`	mutable	Unordered set,	`{4.0, 'string', True}`

		contains no duplicates; can contain mixed types, if hashable	`set()`

`str`	immutable	A character string: sequence of Unicode codepoints	`'Wikipedia'`
			`"Wikipedia"`
			`"""Spanning multiple lines"""`
			Spanning
			multiple
			lines

`tuple`	immutable	Can contain mixed types	`(4.0, 'string', True)`
			`('single element',)`
			`()`

Arithmetic operations

Python has the usual symbols for arithmetic operators (`+`, `-`, `*`, `/`), the floor division operator `//` and the modulo operation `%` (where the remainder can be negative, e.g. `4 % -3 == -2`). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a matrix-multiplication operator `@` .[121] These operators work like in traditional math; with the same precedence rules, the operators infix (`+` and `-` can also be unary to represent positive and negative numbers respectively).

The division between integers produces floating-point results. The behavior of division has changed significantly over time:[122]

- Current Python (i.e. since 3.0) changed `/` to always be floating-point division, e.g. `5/2 == 2.5`.
- The floor division `//` operator was introduced. So `7//3 == 2`, `-7//3 == -3`, `7.5//3 == 2.0` and `-7.5//3 == -3.0`. Adding `from __future__ import division` causes a module used in Python 2.7 to use Python 3.0 rules for division (see above).

In Python terms, `/` is *true division* (or simply *division*), and `//` is *floor division*. `/` before version 3.0 is *classic division*.[122]

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation `(a + b)//b == a//b + 1` is always true. It also means that the equation `b*(a//b) + a%b == a` is valid for both positive and negative values of `a`. However, maintaining the validity of this equation means that while the result of `a%b` is, as expected, in the half-open interval [0, *b*), where `b` is a positive integer, it has to lie in the interval (*b*, 0] when `b` is negative.[123]

Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, Python 3 uses round to even: `round(1.5)` and `round(2.5)` both produce `2`.[124] Versions before 3 used round-away-from-zero: `round(0.5)` is `1.0`, `round(-0.5)` is `-1.0`.[125]

Python allows Boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression `a < b < c` tests whether `a` is less than `b` and `b` is less than `c`.[126] C-derived languages interpret this expression differently: in C, the expression would first evaluate `a < b`, resulting in 0 or 1, and that result would then be compared with `c`.[127]

Python uses arbitrary-precision arithmetic for all integer operations. The `Decimal` type/class in the `decimal` module provides decimal floating-point numbers to a pre-defined arbitrary precision and several rounding modes.[128] The `Fraction` class in the `fractions` module provides arbitrary precision for rational numbers.[129]

Due to Python's extensive mathematics library, and the third-party library NumPy that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.[130][131]

Programming examples

"Hello, World!" program:

```
print('Hello, world!')
```

Program to calculate the factorial of a positive integer:

```
n = int(input('Type a number, and its factorial will be printed: '))

if n < 0:
    raise ValueError('You must enter a non-negative integer')

factorial = 1
for i in range(2, n + 1):
    factorial *= i

print(factorial)
```