Python vs. Go

 **By:** Luciano Ramalho

 **Publisher:** O'Reilly Media, Inc.

 **Pub. Date:** June 15, 2018

 **Pages in Print Edition:** 1

 **Subscriber Rating:** 0 out of 5 ratinghttps://my.safaribooksonline.com/static/201903-8300-oreilly/images/6.0/catalog/icon_star_blue_none.gifhttps://my.safaribooksonline.com/static/201903-8300-oreilly/images/6.0/catalog/icon_star_blue_none.gifhttps://my.safaribooksonline.com/static/201903-8300-oreilly/images/6.0/catalog/icon_star_blue_none.gifhttps://my.safaribooksonline.com/static/201903-8300-oreilly/images/6.0/catalog/icon_star_blue_none.gif[**0 Ratings**]

 **My Rating:** [1 star rating2 star rating3 star rating4 star rating5 star rating](https://my.safaribooksonline.com/book/programming/9781492033172)

# Introduction

Comparing Python with Go is a bit like comparing an SUV with a sports car: they were created to serve different needs. Thanks to their simple syntax and careful design, you will probably find Python and Go easier to learn and use than other mainstream languages that you might have already studied. Given their gentle learning curve and phenomenal growth in several fields, getting to know them is a sound investment now.

# Basics

| Python | Go |
| --- | --- |
| Self-defined as | |
| “Python is a programming language that lets you work more quickly and integrate your systems more effectively.” (quote from python.org) | “Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.” (quote from golang.org) |
| Release Date | |
| January 26, 1994 | March 28, 2012 |
| Current Version | |
| 3.6 | 1.9 |
| Designed By | |
| Guido van Rossum | Robert Griesemer, Rob Pike, and Ken Thompson |
| Copyright by | |
| **Python Software Foundation (PSF)**, a nonprofit foundation operated by elected volunteers from the community. | The Go Authors, a list with more than 1,200 persons or corporations, among them Google, Intel, Facebook, Netflix, and so on. |
| License | |
| PSFL (BSD-like) | BSD 3-clause |
| Key Strengths | |
| Python is easy to learn and highly expressive. These strengths made it very successful in many areas, but they are the main reason why Python dominates machine learning. Operator overloading is important for productivity and readability in math-intensive domains like machine learning. Performance is very important for math, too, and you can easily extend Python with modules in C, C++, FORTRAN, and Rust. The NumPy and SciPy projects provide the extensions for numeric computing that made Python a leading data science language. Python can also be embedded in larger programs. That’s how it is used to automate tasks in high-end computer graphics applications. | Go is easy to learn and difficult to abuse with abstractions or metaprogramming affecting performance in unforeseen ways. It generates standalone, binary executables, but makes it easy to write safe and reliable programs thanks to a garbage collector, array bounds checking, and safer pointer semantics than C. It supports concurrency with parallel execution on all available CPU cores, encouraging the Communicating Sequential Processes (CSP) approach, which avoids some common pitfalls of threads communicating through shared memory and locks. Go has very lightweight threads called goroutines, which it can handle by the hundreds of thousands concurrently. Go uses static typing with type inference, providing type safety without verbose syntax. |
| Hurdles | |
| Python is interpreted and depends on a preinstalled runtime environment. These make performance slower and complicate distribution and deployment—compared to statically typed languages that compile to machine code. IDE support is less comprehensive for Python, due to dynamic typing. Concurrent programming in Python is hindered by a Global Interpreter Lock (GIL) that limits the CPython standard interpreter and PyPy to use only one CPU core. | Go is still young and lacks important pieces. The features missed by most are a widely supported dependency management tool and no generic types. Without generics, static typing leads to code duplication. When you want a Set type, for example, you need to write or find one that works with the specific element type you need. Code generation is offered as a solution to this, but brings its own problems. |

# Mind Share

| Python | Go |
| --- | --- |
| Popularity | |
| #4 on TIOBE index  #1 on IEEE Spectrum ranking  #2 on Github (by opened pull request) | #17 on TIOBE index  #9 on IEEE Spectrum ranking  #9 on Github (by opened pull request) |
| Usage | |
| Python is a leading language in many domains: data science, machine learning, infrastructure automation, computer graphics, web development, IoT, and scientific computing in many fields. MicroPython is a variant that runs on microcontrollers like the BBC micro:bit, pyBoard, and the CircuitPython line. | Go was created for highly scalable, highly concurrent, and distributed systems. Beginning with Docker, many projects in the cloud infrastructure market have adopted Go, and this is leading to rapid growth in the infrastructure automation domain. Go has strong potential to spread into many other domains. |
| Community | |
| PyCon, the largest Python conference, is community organized and attracts more than 2,000 attendees. There are national Python conferences in dozens of countries, on all continents, and countless meetups. There’s a lot of activity on Slack, but IRC is still the best place to get real-time community support. PyLadies and DjangoGirls are two diversity initiatives. | GopherCon, the largest Go conference, is also community organized, and attracts more than 1,000 attendees. There are national Go conferences in some countries, and countless meetups, as well. Slack is best place to get real-time community support, but there are Gophers on IRC, as well. Women Who Go and GoBridge are two diversity initiatives. |
| 8 marquee users | |
| Disney Animation, DropBox, Facebook, Google, ILM, Instagram, Red Hat, YouTube | Canonical, Cloud Foundry, Digital Ocean, Docker, DropBox, Facebook, Google, YouTube |

# Major characteristics

| Python | Go |
| --- | --- |
| Execution model | |
| Interpreted. Python is actually compiled to bytecode, which is then interpreted by the Python runtime. The PyPy implementation includes a Just-in-Time (JIT) compiler to generate machine code on the fly. | Compiled to native machine code. The go run command quickly compiles and executes the source code directly, emulating the user experience of an interpreted language. |
| Interactive console | |
| Bundled. Also, multiple enhanced consoles are available, including iPython, Jupyter Notebook, and Python Anywhere (online). Cloud infrastructure providers are beginning to offer easy ways of hosting and sharing Jupyter Notebooks. | Not bundled. The [Go Playground](https://play.golang.org) allows editing code in the browser and then compiling and running it on a server. Gophernotes is an experimental kernel to run Go in a Jupyter Notebook. |
| Standard library | |
| Comprehensive. Python’s biggest advantage over Go’s standard library is the wealth of generic data structures, like sets, mappings, queues, and the generic iterator/generators from the itertools package. These are building blocks that can really boost programmer productivity. The most significant recent addition was the asyncio package, for concurrent event-driven programming. | Comprehensive. Go’s biggest advantage over Python’s standard library is its network programming packages. For example, Go’s http package supports HTTP/2 and TLS out of the box, and includes tools and mocks to make it easy to test both client and servers. A concurrent HTTP proxy literally fits in a [tweet](https://twitter.com/heptajohn/status/957748274529734658), if error checking is omitted. |
| Typing | |
| Strong, dynamic. Python is a classic “duck-typed” language: object types are not as important as their interfaces (the methods they provide). Type hints were added in version 3.5 (2015) to support static checking and enhanced IDE automation; the type annotations have no effect on runtime behavior, so the language remains dynamically typed. | Strong, static, inferred. The := operator avoids redundant type declarations via local type inference: on the first assignment to a variable, the compiler sets the variable type to conform to the type of the value assigned to it. The way interfaces are declared in Go allows a disciplined form of duck typing. Go’s type assertions and reflect package provide flexibility to deal with types at runtime. |
| Interface enforcement | |
| At runtime, via duck typing. Duck typing (also in Smalltalk, Ruby, JavaScript, and PHP) means the runtime lazily checks whether an object provides the required attributes or methods, regardless of its class. A runtime error is raised if the object does not satisfy the implied interface. | At compile time, via structural typing. This is similar to duck typing but based on explicit interface definitions, and checked at compile time. Implementations don’t declare their interfaces. They merely implement the method(s) needed to satisfy the interface. For example: a function that receives a Reader will accept any object that implements that interface by providing a Read method with the correct signature, regardless of the actual type of the object. |
| First class functions | |
| Yes. This means that you can assign a function can be assigned to a variable, passed it to another function as an argument, and returned it as a result from another function. First class functions enable the use of many functional programming patterns. | Yes. Go has syntax to declare a variable to hold a function with a specific signature. For example, here is how to declare a variable named parse to hold a reference to function tha takes a string argument and returns a Tree and an error: var parse func(text string) (result Tree, err error) |
| Closures | |
| Yes. The scope of a function encompasses variables that appear in its body but that are not local variables or arguments. This is necessary for correct support of first class functions in a language with lexical scoping. | Yes. |
| Anonymous functions | |
| Limited syntax: allows only a single expression, and no statements such as loops, try/except, assignments etc. | Full syntax: no limitations. |

# Ecosystem

| Python | Go |
| --- | --- |
| Tooling | |
| The official Python download has one executable, the interpreter, and some modules that you can use scripts, like unittest for running tests and env for creating isolated development environments. Recent versions have added the pip package installer. Professional Python development requires many tools that are not bundled with Python. | Go comes with the go command, which bundles 16 subcommands such as run to execute source-code directly (by compiling to a temp file), build to compile executables, install to compile and make executables available system-wide, test to run test suites, get to download libraries, fmt to reformat code in a standard style, and vet to check code quality, to name a few. |
| External libraries | |
| PyPI (Python package index) is a central repository with more than 128 open source Python packages ready to install with the pip tool. Additionally, you can install packages that have a standard setup.py script in a Python development environment, and there are countless such packages on GitHub, Bitbucket, and other public code repository sites. Python also has a checkered past with dependencies, although env and pip provide a foundation. The new pipenv project combines both to achieve functionality similar to go get and go dep. | There is no public central repository of Go packages. The syntax of import statements in Go assumes that packages are published widely on the web. For example, at the top of a .go source file, the code... import “github.com/standupdev/wordset” ...tells the standard go get tool where to find and download the wordset package. In 2017 the Go core team finally endorsed the community-developed go dep tool to manage dependencies and facilitate reproducible builds. |

# Data types

| Python | Go |
| --- | --- |
| Data abstraction | |
| class, namedtuple, struct. Python offers advanced support for Object-Oriented Programming (OOP), including multiple inheritance, class decorators, and metaclasses. namedtuple is a factory for simple data-only classes. A Python struct is an object that encodes and decodes fields in packed binary data like Go or C structs. | struct in Go describes a type with fields (similar to C). You can bind methods to a type, enabling the familiar object.method() call syntax. Methods are not nested in the struct declaration: you can define them anywhere. This allows binding methods to native types, like int. The principle “Favor composition over inheritance” is at the core of the Go way: inheritance is not supported, but in some cases you can achieve the desired effect can be achieved by embedding structs: essentially, nesting structs as fields of larger structs. |
| Native collections | |
| list, tuple, dict, set: thanks to dynamic typing, these four data structures accept items of any type, except that dict keys and set elements must be hashable (most immutable objects in Python are hashable). You can use lists as flexible arrays or as stacks. A tuple is a record-like structure with no field names, useful for functions returning multiple values. A dict is a mapping of unique keys to values, with O(1)—constant time—performance for retrieval. A set is an unordered collection of elements guaranteed to be unique, with O(1) membership test and rich set operations (intersection, union, subset, etc.) | array, slice, map, chan: an array in Go is a contiguously allocated array of values of the same type, with a fixed size. slices are like views to parts of an array, which can shrink and grow, and can be shared without copying or explicit pointers. A map is like a Python dict with fixed key and value types. chan is a channel: a queue with single type items, shared between goroutines for communication and synchronization; sending through a channel blocks until the other goroutine receives (unless the channel is buffered). These four data structures, plus the string type, are the only ones the Go for/range statement supports. |
| Native string types | |
| str, bytes: The str type represents human text (elements are Unicode characters), whereas bytes are sequences of integers with values from 0 to 255. Conversion between these types uses the str.decode() and bytes.decode() methods, which default to UTF-8 encoding in Python 3. The most critical breaking change in Python 3 was to make the str/bytes separation more explicit. | string is similar to the old str type in Python 2: it’s used for text, but each element is a byte, so len(“caf√©”) is 5, because string constants in Go are UTF-8 byte sequences, and it takes 2 bytes to represent the accented “√©”. To count characters, you must remember to use the utf8.RuneCountInString(s) function. A for/range loops over strings character by character, not byte by byte. It assumes that the bytes in the string form valid UTF-8 encoded characters. |
| Numeric types | |
| int, float, and complex are the built-in numeric types in Python. int uses a machine-word representation if possible, but transparently changes to an elastic representation that is limited only by available memory, avoiding overflow errors in all realistic scenarios. A Python float is a C double-precision float. A complex is a pair of floats. The standard library also offers Fraction and Decimal types with arbitrary precision. Numeric types in the Python standard library and in popular extensions all use the same readable infix operator syntax, thanks to operator overloading. NumPy also allows infix operations with numerical vector and matrix types. For example, given matrices m and n, and a float x: m \* x computes a scalar multiplication and m @ n does matrix multiplication. | int and float are machine-dependent numeric types in Go—their size depends on the size of the machine word (usually 32 or 64 bits). Go also has machine-independent integer types, signed and unsigned, with 8, 16, 32, or 64 bits; for example, uint8 (alias byte) and int64. For machine-independent floating-point math, Go has the float32, float64, complex64, and complex128 types. The big package implements arbitrary precision Int, Float, and Rat (rational) numbers. Go lacks operator overloading, so you can use standard arithmetic notation only with the primitive numeric types, not with those in the big package. |
| Variable models | |
| All variables in Python hold references to objects—just like object variables in Java. | Go has three variable models: values (physically holding the bytes of the object), pointers as in C, and implicit reference variables for the few built-in generic types (slices, maps, and channels). Pointer declarations are clearly marked by a \* (asterisk character). |

# Flow control

| Python | Go |
| --- | --- |
| Conditionals | |
| if/elif/else. Python has no switch. Multiway branching is expressed as a series of if/elif/elif/elif... blocks, with the guarantee that only one such block will be executed. | if/else, switch/case/fallthrough/default. Go’s switch statement fixes one of the silliest mistakes in C. In Go, only one case is executed, unless it ends with the fallthrough keyword, which causes execution to fall through the block below. |
| Loops | |
| for, while. In Python, you can make any collection class work naturally with the for statement by implementing an \_\_iter\_\_ method. | for. The Go for statement has three forms: one looks like the one in C, the other behaves like a while, and the for/range form is similar to for in Python but works only with five built-in types. |
| Finalization | |
| with, try/finally. You use these statements to perform clean-up operations such as closing files, releasing resources, and releasing locks. with calls special methods to initialize and finalize its target object. | defer. Go’s defer is easier to understand and use than Python’s with. defer causes a function call to be delayed until the end of the current function block and guarantees that the call will happen even in the presence of exceptions or premature returns. |
| Exceptions | |
| try/except, raise. Python programmers use try/except not only for error handling, but also for flow control. For example: instead of checking whether a key is in a dict, go ahead and fetch it, but handle the exception. You use raise to throw exceptions. | panic, recover. The panic function is somewhat similar to Python’s raise: it starts unwinding the call stack to terminate the program. In the context of a defer function, a call to recover can stop the process. Go libraries rarely use panic. Error handling is done by checking return values, with a major improvement over C: functions report errors by returning a second value of type error. See the I/O example later onbelow to for typical error handling in Python and Go. |
| I/O example | |
| try: with open(filename, ‘b’) as fp: # do something with fp except IOError as ex: logging.exception(‘oops...’) | fp, err := os.Open(filename) if err != nil { log.Fatal(err) } defer fp.Close() // do something with fp |
| Concurrency | |
| async/await are new in Python 3.5. A function declared with async def is a coroutine: it must be called by another coroutine using the await keyword or by a low-level scheduler such as the event loop in the asyncio module. You can pause coroutines without blocking the event loop. To pause, use await to invoke another coroutine or a future object. async/for and async/with allow the special methods that support those constructs to be coroutines. Python also supports concurrent programming via its threading, multiprocessing, and concurrent.futures modules. The major drawback is the global interpreter lock (GIL) that ensures that only one Python bytecode thread is active at any time, regardless of the number of CPU cores. The async/await keywords were introduced to make single-threaded event oriented programming more convenient than the traditional callback-based style. The await keyword can only be used with coroutines, not regular functions. | go, <-, select. You can spawn any function as a concurrent goroutine using the go keyword: go myFunc(). To send/receive values, goroutines use channels. x = <-c reads from a channel; c <- x writes to it. select/case/default is a switch-like control structure designed for concurrency. Each case has a channel read/write operation; Go chooses one case that is ready, blocks while none are, or executes the default clause if present. This ensures that the select never blocks. Besides communicating through channels, goroutines can also access shared resources guarded by mutex locks. Go’s concurrency features are easier to understand, more expressive, and more performant. Go can run goroutines on all cores, in parallel; you can use the go keyword with any function; and the scheduler moves waiting goroutines from one core to another to balance the load. |

# Design and evolution

| Python | Go |
| --- | --- |
| Influences | |
| Previously, van Rossum was part of the team that implemented the ABC language in 10-year research project to create a language that reduced programming errors by applying human-centered design and ergonomics. Python inherited many traits from ABC. | Thompson created B, the direct predecessor of C. Griesemer coauthored Object Oberon, an object-oriented descendant of Pascal and Modula-2. Pike’s Squeak and Newsqueak languages implemented concurrency using Hoare’s CSP model. |
| Technical governance | |
| Guido van Rossum is the BDFL (Benevolent Dictator For Life): he makes the final calls about language features. New features are exhaustively discussed in the open. Most of them go through a well-structured and transparent Python Enhancement Proposal (PEP) process. Guido often delegates PEP review to other core developers. | From the start, Griesemer, Pike, and Thompson adopted a minimalist approach: a feature was added only if all three liked it. The current process for a change proposal starts with filing an issue. Go core developers can accept, decline, or request a design document—similar to a PEP—for further analysis. Russ Cox, an early Go author, is currently the arbiter of disputes in this process. |
| Backward compatibility | |
| Promised within major releases. Within a major release (1.x, 2.x, 3.x), it is very rare for compatibility problems to affect real programs; for example, practically all code written in Python 2.2 runs in 2.7. The same applies to code written in earlier releases of Python 3: the vast majority runs in the latest release, 3.6. The lifetime of Python 2.7 was extended to allow migration to Python 3. | The Go 1 compatibility promise is widely quoted by the Google Go advocates: “It is intended that programs written to the Go 1 specification will continue to compile and run correctly, unchanged, over the lifetime of that specification.” (source: [Go 1 and the Future of Go Programs](https://golang.org/doc/go1compat)) |

# How to Choose

## What is your background?

If you are just learning to program, Python is easier to get started and will give you more scope for practical application in different domains, thanks to the huge selection of libraries available for it. That is why Python is the most widely used teaching language in the top 10 and the [top 39 US undergraduate introductory programming courses](https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext).

If you already program in a statically typed language, Go is very easy to pick up, and you are more likely to feel comfortable with it than with Python. On the other hand, programmers coming from Ruby, JavaScript, PHP, or Perl will find Python’s duck typing familiar and might find its simple and consistent syntax either very readable or too limiting. Syntax is mostly a matter of taste, after all.

## What is your goal?

If your focus is data science or machine learning, Python is clearly the best choice. Go has a Data Science ecosystem, as well, but it is much smaller. For example: the TensorFlow machine learning framework developed by the Google Brain team offers better API support for Python at this time. The NumPy and SciPy packages are years ahead of similar offerings for Go. Python’s flexible operator syntax is better suited for scientific computing in general, but Go’s performance and concurrency make it excellent for large scale Data Engineering tasks.

Finally, if your goal is to build web-scale APIs, crawlers, databases, and infrastructure, the performance, concurrency and tooling of Go make it the safest bet. It is possible to write highly concurrent network applications in Python, as well—YouTube and Instagram are outstanding examples—but it is more challenging and requires more hardware. Web-scale Python apps demand more complicated architecture and infrastructure choices.

Pick any one to study first and happy hacking! Don’t fall into the trap of choosing one and completely ignoring the other. Python and Go are both too good and too important to miss. Plan to add both to your skill set.