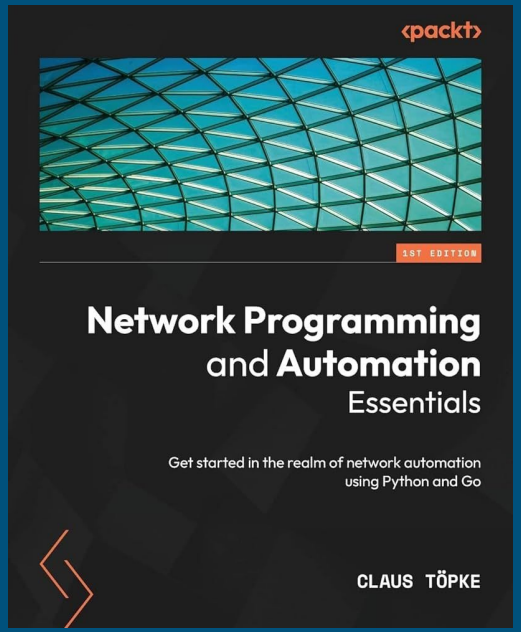
Using Python and Go for Network Automation

A lovely versus battle

Based on some experience and book

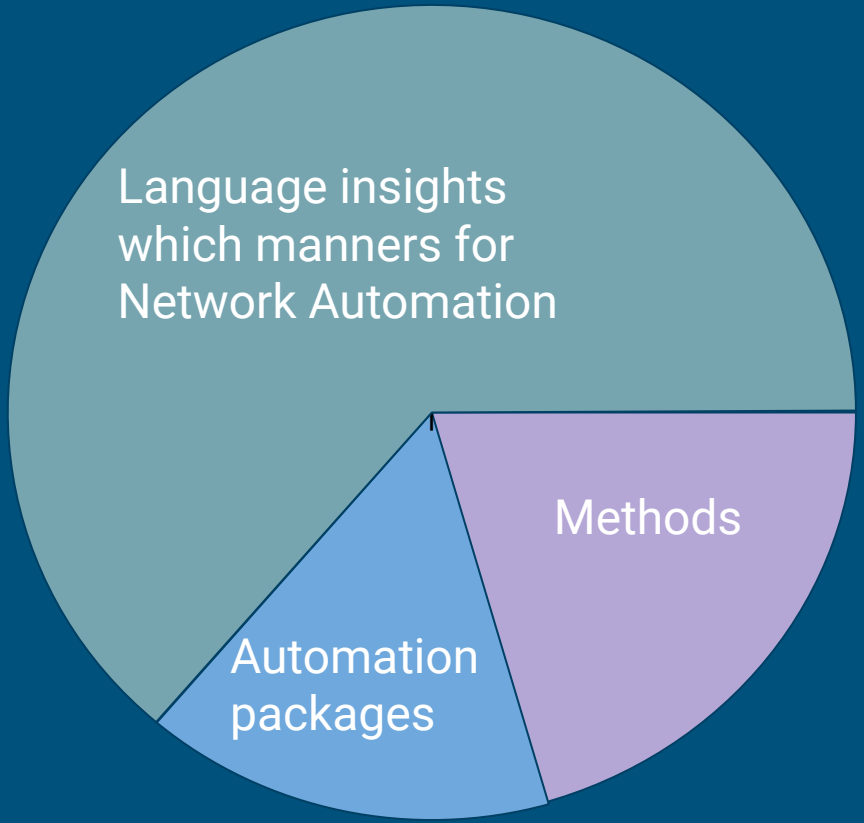


nbn™ Australia's broadband network bring it on

amazon web services

TELCO MANAGER

Presentation Summary



Disclaimer

No intention to give Software Engineering or Coding guidance

Intended for NEs, DevOps and SDEs willing to dive into Network Automation

No affiliation to any of the language creators or maintainers

A bit shallow but deep enough to offer insights for comparing Go and Python

Memes were taken from Internet, URLs of original included when possible

Let’s help to find a common ground

https://qit.software/go-vs-python-comparing-performance-features/

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods and tools		
Deployment and dependencies		
Network Automation community tools		

Language release cycle and implementations

Python language Release Timeline

1989	1991	1994	2000	2008	2010	2018	2021	2023	2024
	0.9	1.0	2.0	2.6 3.0	2.7 3.1	2.7.15 3.7.0	3.10	3.12	3.13

by Guido van Rossum

https://en.wikipedia.org/wiki/History_of_Python

https://www.python.org/doc/sunset-python-2/ ← Jan/2020

https://www.python.org/downloads/release/python-3120/

https://www.python.org/psf/sponsors/

https://us.pycon.org/2024/

https://www.etsy.com/hk-en/listing/1241340884/python-plushie-python-plush-python-plush

Python language Release Cycle

To clarify terminology, Python uses a major.minor.micro nomenclature for production-ready releases. So for Python 3.1.2 final, that is a major version of 3, a minor version of 1, and a micro version of 2.

- ➔ new major versions are exceptional; they only come when strongly incompatible changes are deemed necessary, and are planned very long in advance;
- ➔ new minor versions are feature releases; they get released annually, from the current in-development branch;
- ➔ new micro versions are bugfix releases; they get released roughly every 2 months

https://devguide.python.org/developer-workflow/development-cycle/

Python implementations

CPython (https://github.com/python/cpython) ← Pre Python release

PyPy (https://www.pypy.org/ https://foss.heptapod.net/pypy/pypy)

Numba (https://numba.pydata.org/)

IronPython (.NET) (https://ironpython.net/)

Jython (JVM) (https://github.com/jython/jython)

MicroPython (https://github.com/micropython/micropython/)

IPython (https://ipython.org/)

Jupyter notebook using IPython

Iteratively work with web interface with Python (https://jupyter.org/)

Share easily notebooks

Great for Proof of Concepts for Network Automation

Go language Release Timeline

2009	2011	2012	2014	2016	2019	2021	2023	2024
Weekly Snapshots	r56	1.0	1.4	1.6	1.12	1.17	1.21	1.22

by Robert Griesemer, Rob Pike, and Ken Thompson

https://go.dev/doc/devel/weekly

https://groups.google.com/g/golang-dev/

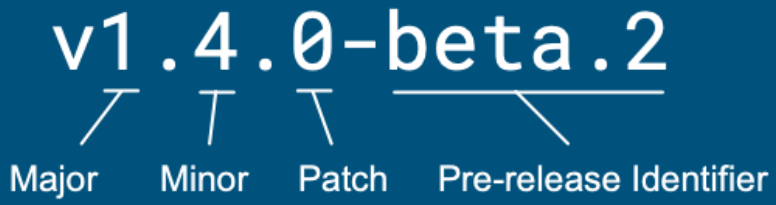
https://go.dev/blog

https://go.dev/blog/14years

https://go.dev/blog/go119runtime

https://www.amazon.com/Desktop-Decor-Golang-Mascot-Gopher/dp/B0C1CSSV2Z

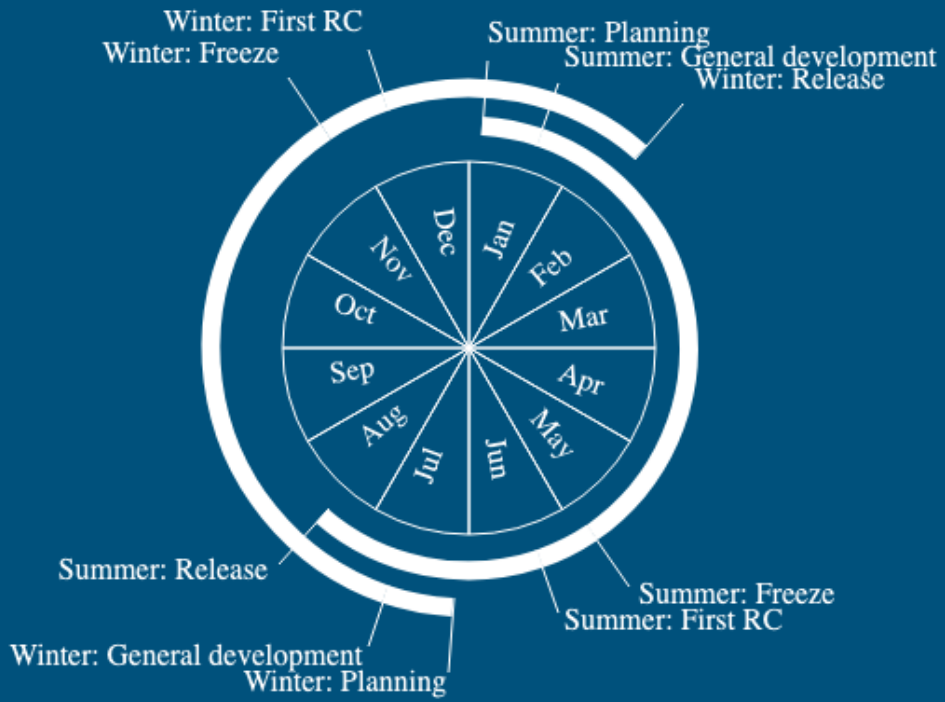
Go language Version Numbers



Major version	**v1.x.x**	**Signals backward-incompatible public API changes. This release carries no guarantee that it will be backward compatible with preceding major versions.**
Minor version	**vx.4.x**	**Signals backward-compatible public API changes. This release guarantees backward compatibility and stability.**
Patch version	**vx.x.1**	**Signals changes that don't affect the module's public API or its dependencies. This release guarantees backward compatibility and stability.**
Pre-release version	**vx.x.x-beta.2**	**Signals that this is a pre-release milestone, such as an alpha or beta. This release carries no stability guarantees.**

https://go.dev/doc/modules/version-numbers

Go language Release Cycle



https://zchee.github.io/golang-wiki/Go-Release-Cycle/

Go Implementations

gccgo (https://github.com/golang/go)

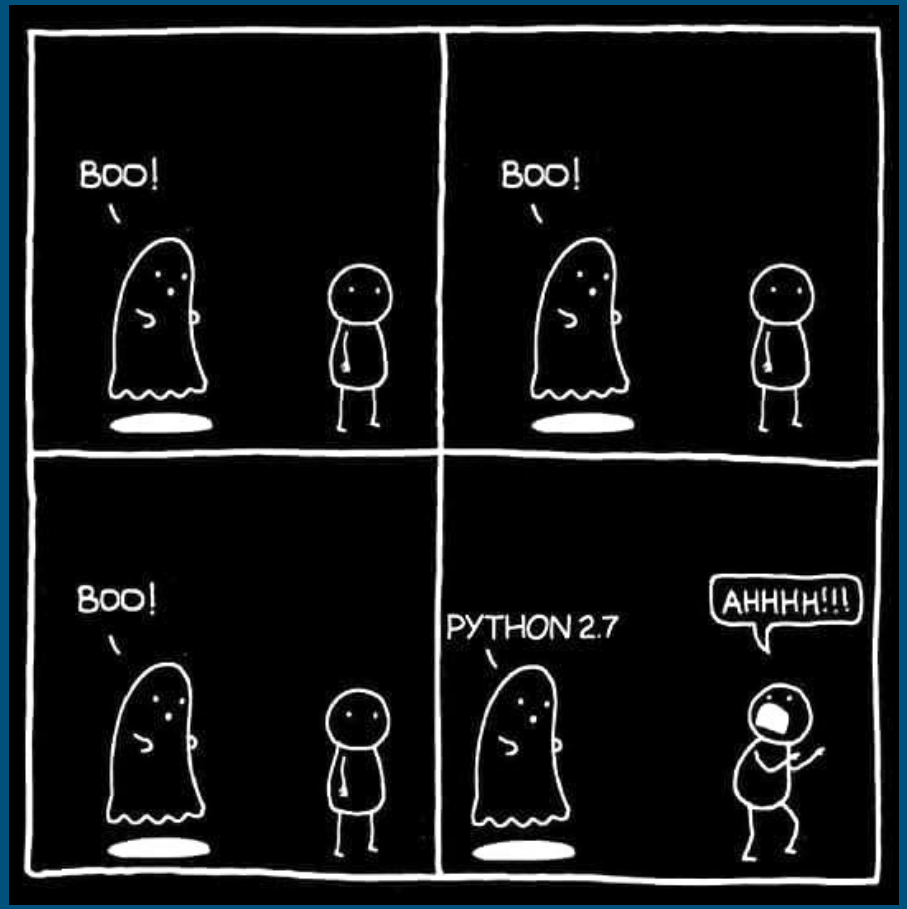
llgo (go1.3 - 9 years old https://github.com/go-llvm/llgo moved to llvm)

tinygo (go1.21 https://tinygo.org/ combination gccgo tools and llvm https://llvm.org/)

gopherjs (go1.18 https://github.com/gopherjs/gopherjs)

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		



Jan/2020

https://www.python.org/doc/sunset-python-2/

Memory, speed, parallelism, concurrency and Network (IO)

Interpreted and Compiled

- Go is compiled
  - Run compiler and create a binary file static linked

```
 # file countdown

countdown: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked
```

- Python is interpreted
  - CPython interprets the Python bytecode

Dynamically typed and statically typed

```
x = 5
y = 5.5

print(x + y)
```

```
package main

import "fmt"

func main() {
    var x int = 5
    var y float64 = 5.5

    fmt.Println(float64(x) + y)
}
```

Python bytecode

```
$ python3.12
Python 3.12.0 (main, Oct 21 2023, 17:42:12) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def sayHello():
...     print("Hello people!");
...
>>> import dis
>>> dis.dis(sayHello)
  1           0 RESUME                   0

  2           2 LOAD_GLOBAL              1 (NULL + print)
             12 LOAD_CONST               1 ('Hello people!')
             14 CALL                     1
             22 POP_TOP
             24 RETURN_CONST             0 (None)
>>>
```

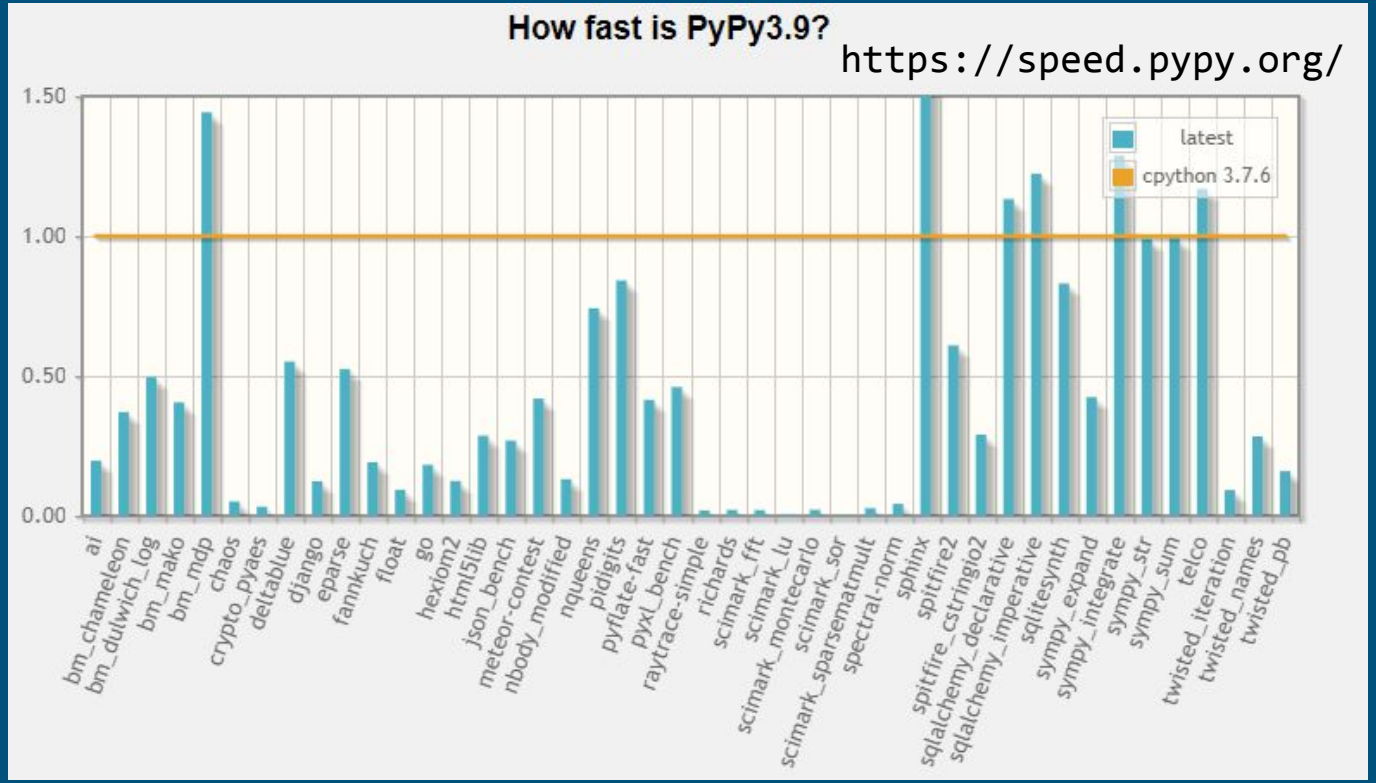
https://docs.python.org/3/library/dis.html

Python bytecode

```
% python3.12 -m py_compile countdown.py

% ls -l
% ls -l
total 7768
drwxr-xr-x  3 claustopke  staff       96 Nov 30 14:54 __pycache__
-rw-rw-r--  1 claustopke  staff      229 Nov 30 13:09 countdown.py

% ls -l __pycache__
total 8
-rw-r--r--  1 claustopke  staff  598 Nov 30 14:54 countdown.cpython-312.pyc
```

https://docs.python.org/3/library/dis.html

PyPy versus CPython



Which cases PyPy excel ?

PyPy tends to be fast in pure numerically-intensive codes with hot loops dealing with (small) integers/float number as it can directly use native types instead of variable-sized dynamic integer/float objects.

PyPy does not like (large) dynamic codes. It uses a tracing just-in-time compiler that can track which part of the code are more likely to be executed and compile this path dynamically at runtime if it is executed often. When there are many path executed changing dynamically, the overhead of the JIT can be significant, and in the worst case, PyPy can choose not to compile any path.

PyPy use a Garbage Collector (GC) as opposed to CPython which use Automatic Reference Counting (ARC). GCs can be faster to allocate/free many objects (especially small temporary objects), but they needs to track the object alive to know which one are dead and then free them. This means codes dealing with a huge amount of references and regular object allocations can actually be slower.

https://doc.pypy.org/en/latest/gc_info.html

https://www.pypy.org/performance.html

Just in Time Compiling

Standard Python distribution comes with CPython, which is written in C and Python and includes an interpreter and a Python bytecode compiler. So it does not convert directly to the machine instructions for specific CPU. CPython interprets line by line Python bytecode.

Other implementations like Pypy, IronPython and Numba does include Just in Time Compiler (JIT), which compiles during runtime the Python bytecode to the machine code (similar to JAVA). The consequence remove lots of features, and does support only a set of packages and capabilities of standard Python code.

Why CPython does not have JIT ?

A large reason CPython doesn't have a JIT is the extra complexity it introduces to the core implementation and issues regarding C extensions.

Due mainly because of some Python capabilities like dynamic typing, polymorphism, and various introspective features. Therefore, some code will run as fast as with or without JIT.

https://www.theregister.com/2021/05/19/faster_python_mark_shannon_author/

Parallelism and Concurrency

Oracle Multithreaded Programming Guide (1994):

Defining Concurrency and Parallelism

"Concurrency exists when at least two threads are in progress at the same time.

Parallelism arises when at least two threads are executing simultaneously.

In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution.

In the same multithreaded process on a shared-memory multiprocessor, each thread in the process can run on a separate processor at the same time, resulting in parallel execution. "

https://docs.oracle.com/cd/E19457-01/801-6659/801-6659.pdf

Threads, Routines and Processes

- **Processes** are independent, isolated units of execution with their own memory space.
- **Threads** are lightweight execution units within a process, sharing memory and resources.
- **Routines** are similar to threads but run within a single execution context, often enabling cooperative multitasking.

Python Threading, Multiprocessing and Coroutines

Python 2.4.1 (2005) introduced **`threading`** standard library

https://docs.python.org/3/whatsnew/2.4.html

Python 2.6 (2008) introduced **`multiprocessing`** standard library

https://docs.python.org/3/whatsnew/2.6.html

Python 3.14 (2014) introduced **`asyncio`** standard library

https://docs.python.org/3/whatsnew/3.4.html

CPU-bound and I/O-bound tasks

CPU-bound tasks benefit from parallel execution across multiple cores.

I/O-bound tasks benefit from concurrency to minimize idle time during I/O operations, including Network access.

CPU-bound tasks benefit from parallel execution to fully utilize CPU resources, while I/O-bound tasks benefit from asynchronous or concurrent execution to minimize waiting time for I/O operations.

Network automation tasks might be only I/O-bound, depending on the computation required. Complementary systems, like APIs and interfaces might rely on the same I/O-bound.

Python Global Interpreter Lock (GIL)

Python has the Global Interpreter Lock (GIL) primarily for simplicity and ease of implementation within its interpreter, ensuring thread safety by allowing only one thread to execute Python bytecode at a time, preventing conflicts in memory access.

Example of conflict is reference counting for memory management, which involves assigning a reference count variable to objects created in Python.

The GIL ensures thread safety in Python by allowing only one thread to execute Python bytecode at a time. This restriction simplifies memory management, by avoiding potential conflicts when multiple threads access and modify the reference counts of objects simultaneously.

https://realpython.com/python-gil

Python Global Interpreter Lock (GIL)

```
import time

NUMBER = 10 ** 9  # 1 Billion

def countdown(n):
    while n > 0:
        n = n - 1

print("Counting down", NUMBER)

start = time.time()
countdown(NUMBER)
end = time.time()

print("Took {:f}s".format(end - start))
```

```
$ python3.12 countdown.py
Counting down 1000000000
Took 38.686405s
```

Python Global Interpreter Lock (GIL)

```
import time

NUMBER = 10 ** 9  # 1 Billion

def countdown(n):
    while n > 0:
        n = n - 1

print("Counting down", NUMBER)

start = time.time()
countdown(NUMBER)
end = time.time()

print("Took {:f}s".format(end - start))
```

```
import time
from threading import Thread

NUMBER = 10 ** 9  # 1 Billion

def countdown(n):
    while n > 0:
        n = n - 1

t0 = Thread(target=countdown, args=(NUMBER // 2,))
t1 = Thread(target=countdown, args=(NUMBER // 2,))

print("Counting down two threads with", NUMBER // 2)

start = time.time()
t0.start()
t1.start()
t0.join()
t1.join()
end = time.time()

print("Took {:f}s".format(end - start))
```

Python Global Interpreter Lock (GIL)

```
import time
from threading import Thread

NUMBER = 10 ** 9  # 1 Billion

def countdown(n):
    while n > 0:
        n = n - 1

t0 = Thread(target=countdown, args=(NUMBER // 2,))
t1 = Thread(target=countdown, args=(NUMBER // 2,))

print("Counting down two threads with", NUMBER // 2)

start = time.time()
t0.start()
t1.start()
t0.join()
t1.join()
end = time.time()

print("Took {:f}s".format(end - start))
```

```
$ python3.12 countdown-thread.py
Counting down two threads with 500000000
Took 38.277773s
```

Python Global Interpreter Lock (GIL)

```
import time
from multiprocessing import Process

NUMBER = 10 ** 9  # 1 Billion

def countdown(n):
    while n > 0:
        n = n - 1

p0 = Process(target=countdown, args=(NUMBER // 2,))
p1 = Process(target=countdown, args=(NUMBER // 2,))

print("Counting down two processes with", NUMBER // 2)

start = time.time()
p0.start()
p1.start()
p0.join()
p1.join()
end = time.time()

print("Took {:f}s".format(end - start))
```

```
$ python3.12 countdown-multiproc.py
Counting down two processes with 500000000
Took 18.983889s
```

JIT and GIL influence on PyPy and CPython

```
$ python3.12 countdown.py
Counting down 1000000000
Took 38.686405s
```

```
$ pypy3 countdown.py
Counting down 1000000000
Took 0.904030s
```

```
$ python3.12 countdown-thread.py
Counting down two threads with 500000000
Took 38.277773s
```

```
$ pypy3 countdown-thread.py
Counting down two threads with 500000000
Took 1.908610s
```

```
$ python3.12 countdown-multiproc.py
Counting down two processes with 500000000
Took 18.983889s
```

```
$ pypy3 countdown-multiproc.py
Counting down two processes with 500000000
Took 0.465511s
```

```
pypy3
Python 3.6.9 (7.3.1+dfsg-4ubuntu0.1, Nov 15 2022, 06:22:42) 36
[PyPy 7.3.1 with GCC 9.4.0] on linux
```

The Countdown in Go

```
package main

import (
	"fmt"
	"time"
)

const Number uint = 1e9 // 1 Billion

func countdown(n uint) {
	for n > 0 {
		n = n - 1
	}
	return
}

func main() {
	fmt.Printf("Counting down %d\n", Number)
	start := time.Now()
	countdown(Number)
	end := time.Now()
	fmt.Printf("Took %v\n", end.Sub(start))
}
```

```
$ go build countdown.go
$ ./countdown
Counting down 1000000000
Took 602.624634ms
```

The Countdown exercise with goroutines

```
package main

import (
	"fmt"
	"sync"
	"time"
)

const Number uint = 10e9 // 1 Billion

func countdown(n uint, wg *sync.WaitGroup) {
	defer wg.Done()
	for n > 0 {
		n = n - 1
	}
	return
}

func main() {
	var wg sync.WaitGroup
	wg.Add(2)

	fmt.Printf("Counting down %d two goroutines\n", Number/2)
	start := time.Now()
	go countdown(Number/2, &wg)
	go countdown(Number/2, &wg)
	wg.Wait()
	end := time.Now()

	fmt.Printf("Took %v\n", end.Sub(start))
}
```

```
$ go build countdown-goroutine.go
$ ./countdown-goroutine
Counting down 500000000 two goroutines
Took 306.083274ms
```

CPU and Memory usage side-by-side

Using with `time -f "%MKB %P %e"`

```
python3.12 countdown.py
Took 37.943554s
10292KB 99% 37.95s

python3.12 countdown-thread.py
Took 38.402318s
11004KB 100% 38.42s

python3.12 countdown-multiproc.py
Took 18.978695s
13308KB 199% 19.01s
```

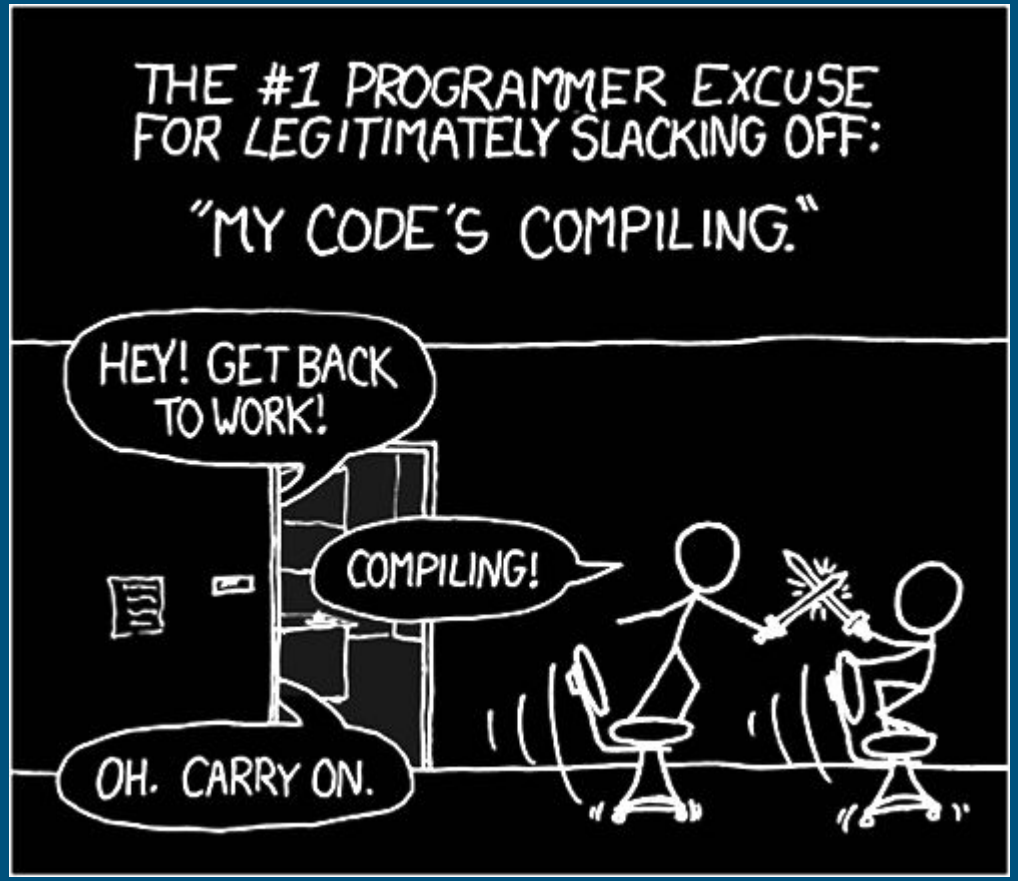
```
pypy3 countdown.py
Took 0.904377s
61812KB 98% 1.00s

pypy3 countdown-thread.py
Took 1.876769s
63536KB 103% 1.99s

pypy3 countdown-multiproc.py
Took 0.463419s
66956KB 171% 0.61s
```

```
./countdown
Took 602.623829ms
1844KB 100% 0.60s
```

```
./countdown-goroutine
Took 303.088477ms
1872KB 199% 0.30s
```

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

IP and network native libraries

Python standard network libraries

Python 0.9.1 (1991) introduced **socket** standard library module.

https://github.com/smontanaro/python-0.9.1

https://www.python.org/download/releases/early/

Python 3.0 (2008) introduced **ssl**, **http**, **html** and **urllib** standard libraries.

https://docs.python.org/3/whatsnew/3.0.html

Python standard network libraries

Python 3.3 (2012) introduced **`ipaddress`** standard library

https://docs.python.org/3/whatsnew/3.3.html

It does include wide operations with IPv4 and IPv6, including network range, validation, multicast, loopback, CIDR, subnet, supernet, compare, collapse, among others.

https://docs.python.org/3/library/ipaddress.html

Example:

```
network_v4 = ipaddress.IPv4Network('192.168.0.0/24')
print("\nIPv4 Network Hosts:")
for host in network_v4.hosts():
    print(host)
```

Go standard network libraries

The majority of the network libraries were introduced in early Go, like version 1.0 and 1.2.

For IP manipulation is `net` package. Other important ones are `crypto/tls` , `http`, `net/http`, `net/rpc`, among others.

The IP manipulation capabilities are inferior to Python, for instance it is not possible to loop a subnet without a helper function.

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		

Program Logging, Error handling and Exceptions

Python logging

Python 2.3 (2008) introduced **`logging`** standard library

https://docs.python.org/3/whatsnew/2.3.html

The logging module offers functionalities to create loggers, set logging levels, route logs to different destinations (handlers) like files or the console, format log messages, and filter log records based on specific criteria.

Hierarchical logging in Python is achieved through the use of logger names separated by periods (dot notation).

Go logging

The initial standard logging package in Go is called `log`, but is very limited in compare to Python, for instance does not have log levels.

Version 1.21 (2023) introduced new package called log/slog, which includes some of the missing capabilities, like levels and handlers. (https://tip.golang.org/doc/go1.21)

More on Go slog:

https://go.dev/blog/slog

https://betterstack.com/community/guides/logging/logging-in-go/

Go versus Python for error handling

Go uses explicit return values checks.

Python uses Exception based and tracebacks.

Python's exception-based approach can make code more readable and avoid repetition, whereas Go's explicit error returns can lead to more explicit error handling and more tedious repetitions.

Built-in exceptions in Python are hierarchical, and easy to work with:

https://docs.python.org/3/library/exceptions.html

Go error handling is up to developer

Most programmers would not implement correctly error handling, due to cascade error check and message syntax.

So, the syntax still allows developers to make errors, with wrong or imprecise signatures of the error.

Checking every single method for errors and returning a proper message is extremely frustrating and most of the time not well maintained.

Therefore, is likely to introduce bugs. Specially if you have 1.000 places where you have to add error checks, the likelihood you forgot or write wrong message is high.

Go example of lack context error handling

```
        if err = c.realm.GetFrom(res); err != nil {
+               log.Println("GetFrom2 problem")
                return relayed, lifetime, nonce, err
        }
        c.realm = append([]byte(nil), c.realm...)
@@ -281,11 +288,13 @@ func (c *Client) sendAllocateRequest(protocol proto.Protocol)
(proto.RelayedAddr
                stun.Fingerprint,
        )
        if err != nil {
+               log.Println("Build Authorize problem")
                return relayed, lifetime, nonce, err
        }

        trRes, err = c.PerformTransaction(msg, c.turnServerAddr, false)
        if err != nil {
+               log.Println("PerformTransation problem")
                return relayed, lifetime, nonce, err
        }
        res = trRes.Msg
```

https://github.com/pion/turn

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		

Network access methods

Some methods to access network

Command Line Interface (Serial, TELNET, RSH, SSH)

SNMP (v2, v2c and v3) (bad implementation traps)

Private APIs

NETCONF, RESTCONF and YANG

gRPC

gNMI (Telemetry future ?)

* note

Pull

(like SNMP, private APIs, CLI scraping)

versus

Streaming telemetry

(like SNMP traps, netflow, sflow, syslog)

CLI - Python

- Using local shell via **`subprocess`** (invoke shell **`ssh`** command).
- Using standard libraries with **`socket`**.
- Python bindings for libssh2 (C code), called **`ssh2-python3`**
- Using **`pexpect`** (not being updated).
- Using **`paramiko`**.
- Using **`netmiko`**.
- Using **`scrapli`**.

`asyncio` aware:

- Using **`asyncssh`**.
- Using **`scrapli-asyncssh`** plugin for **`scrapli`**. (Dec 2020)

CLI Python examples

subprocess

```
try:
    ssh_process = subprocess.Popen(...)
        ['/usr/bin/ssh', 'user@hostname'],
        .
        .
    )
    command_to_send = 'ls -l'
    ssh_process.stdin.write(command_to_send + '\n')
    output, error = ssh_process.communicate(timeout=5)
    print("Output:", output)
    print("Error:", error)

except subprocess.TimeoutExpired:
    print("Timeout: Process took too long.")

except Exception as e:
    print("Error:", e)
```

paramiko

```
try:
    client.connect(
        hostname=hostname,
        port=port,
        username=username,
        password=password,
    )
    stdin, stdout, stderr = client.exec_command('ls -l')
    output = stdout.read().decode('utf-8')
    print("Command output:", output)

except paramiko.AuthenticationException:
    print("Authentication failed")
except paramiko.SSHException as exc:
    print("Unable to connect:", exc)
```

CLI Python examples

netmiko

```
device = {
    'device_type': 'cisco_ios',
    'host': host,
    'username': username,
    'password': password,
}
try:
    net_connect = ConnectHandler(**device)
    net_connect.enable()
    output = net_connect.send_command('show version')
    print("Command output:", output)

except Exception as e:
    print("failed:", e)
```

scrapli

```
device = {
    'host': hostname,
    'Auth_username': username,
    'Auth_password': password,
}
try:
    conn = IOSXEDriver(**device)
    conn.open()
    response = conn.send_command('show version')
    print("Command output:", output)

except ScrapliException as err:
    print("An error occurred: {}", err)
```

CLI Python asyncio aware

asyncssh

```
async def run_ssh_command():
    async with asyncssh.connect(
        'hostname',
        username='user',
        password='password') as conn:

            result = await conn.run('ls -l')
            print(result.stdout)

asyncio.run(run_ssh_command())
```

scrapli plugin

```
async def run_cisco_command():
    iosxe = {
        "host": host,
        "auth_username": username,
        "auth_password": password,
        "platform": "cisco_iosxe",
    }
    async with AsyncScrapli(**iosxe, plugin="asyncssh") as conn:
        resp = await conn.send_command("show version")
        print(resp.result)

asyncio.run(run_cisco_command())
```

CLI - Go

- Using local shell via **`os/exec`** (invoke shell **`ssh`** command).
- Using standard libraries with **`socket`**.
- Using **`golang.org/x/crypto/ssh`**.
- Using **`github.com/google/goexpect`**. (with ssh wrapper)
- Using **`github.com/gliderlabs/ssh`**.
- Using **`github.com/yahoo/vssh`**.
- Using **`github.com/scrapli/scrapligo`**.

CLI Go examples

os/exec

```
func execCmd(host, user, cmd string) {

	sshCmd := fmt.Sprintf("ssh %s@%s '%s'",user, host, command)

	cmd := exec.Command("bash", "-c", sshCmd)
	output, err := cmd.CombinedOutput()
	if err != nil {
		log.Fatalf("Error on %s: %s\n", host, err)
	}

	fmt.Printf("Output for %s:\n%s\n", host, output)
}
```

golang.org/x/crypto/ssh

```
func execCmd(host, user, passwd, cmd string) {
	config := &ssh.ClientConfig{
		User: user,
		Auth: []ssh.AuthMethod{
			ssh.Password(passwd),
		},
	}
	client, err := ssh.Dial("tcp", host+":22", config)
	if err != nil {
		log.Fatalf("Error on dial: %s", err)
	}
	defer client.Close()
	session, err := client.NewSession()
	if err != nil {
		log.Fatalf("Error creating session: %s", err)
	}
	defer session.Close()
	output, err := session.CombinedOutput(command)
	if err != nil {
		log.Fatalf("Error running command: %s", err)
	}
	fmt.Printf("Output:", string(output))
}
```

CLI Go examples

yahoo/vss

```go
func main() {
	hosts := []string{"hostA", "hostB", "hostC", }
	username := "mynameisbob"
	passwd := "mypasswordissecret"

	var wg sync.WaitGroup

	for _, host := range hosts {
		wg.Add(1)
		go execComd(host, username, passwd, &wg)
	}

	wg.Wait()
}
```

```go
func execCmd(host string, user string, pass string, wg *sync.WaitGroup) {
	defer wg.Done()

	config := &vssh.Config{
		User:            username,
		Password:        pass,
		Host:            host,
		ConnectionRetry: 3,
		Timeout:         10 * time.Second,
	}
	client, err := vssh.Dial(config)
	if err != nil {
		log.Printf("Dial error %s: %v\n", host, err)
		return
	}
	defer client.Close()
	output, err := client.Run("ls -l")
	if err != nil {
		log.Printf("Error on %s: %v\n", host, err)
		return
	}
	fmt.Printf("%s:\n", host, output)
}
```

SNMP

Python:

- Using `pysnmp`. (does have asyncio `pysnmp.hlapi.asyncio`)
- Using `pysmi`. (MIB parser)
- Net-snmp Python bindings (https://pypi.org/project/python3-netsnmp/).
- Using `puresnmp`. (no dependencies)
- Asyncio aware use `aiosnmp`.

Go:

- Using `github.com/gosnmp/gosnmp`.
- Using `github.com/k-sone/snmpgo`.

Private APIs

Amazon AWS :

- Python (boto3): https://pypi.org/project/boto3/
- Go: https://github.com/aws/aws-sdk-go

Microsoft Azure:

- Python: https://pypi.org/project/azure-mgmt-compute/
- Go: https://github.com/Azure/azure-sdk-for-go/

NETCONF, RESTCONF and YANG

Python:

- Using **`ncclient`** (supports subscription of YANG events, telemetry?).
- Using **`aio-ncclient`** (Juniper initiative).
- Using **`libyang`** (YANG parser, including yanglint).

Go:

- Using go-netconf (Juniper initiative github.com/Juniper/go-netconf)
- Using goyang (openconfig github.com/openconfig/goyang)

gRPC

Use HTTP2/TLS

Required protobuf (serialize structured data), `.proto` file. (service definition)

Generation of client and server code using protobuf compiler./

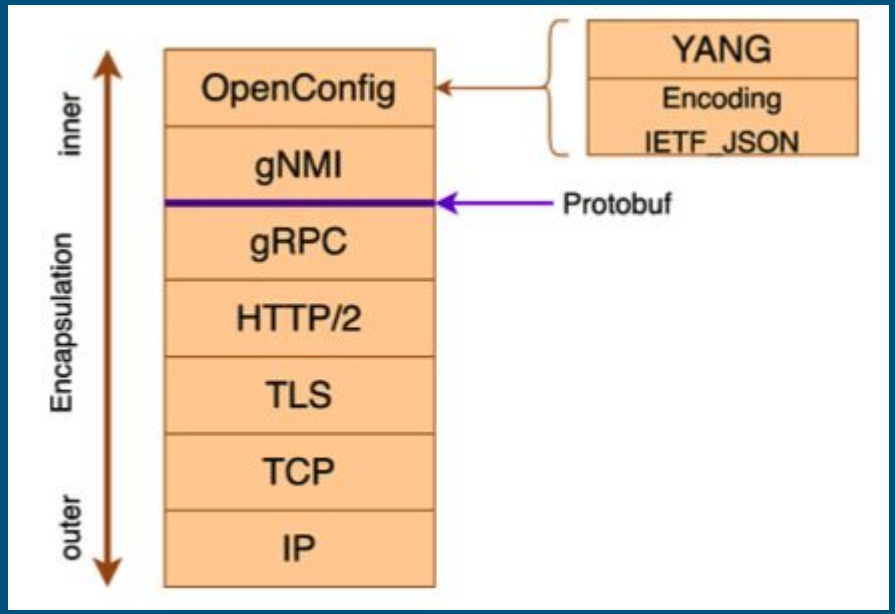
Python:

- https://grpc.io/docs/languages/python/basics/
  - `grpcio` and `grpcio-tools`
- Asyncio https://grpc.github.io/grpc/python/grpc_asyncio.html

Go:

- https://grpc.io/docs/languages/go/basics/
  - `google.golang.org/protobuf/`
  - `google.golang.org/grpc`

gNMI - gRPC Network Management Interface



gNMI - gRPC Network Management Interface

NETCONF/RESTCONF uses HTTP/HTTPS, JSON and XML (slow)

Future of Telemetry subscription?

Uses Openconfig data models https://www.openconfig.net/projects/models/

BGP neighbour data model:
https://github.com/openconfig/public/blob/master/release/models/bgp/openconfig-bgp-neighbor.yang

Interface IP data model (RFC7277 -> RFC8343):
https://github.com/openconfig/public/blob/master/release/models/interfaces/openconfig-if-ip.yang

gNMI - RFC8343

YANG data node in /interfaces/interface	IF-MIB object
name	ifName
type	ifType
description	ifAlias
admin-status	ifAdminStatus
oper-status	ifOperStatus
last-change	ifLastChange
if-index	ifIndex
link-up-down-trap-enable	ifLinkUpDownTrapEnable
phys-address	ifPhysAddress
higher-layer-if and lower-layer-if	ifStackTable
speed	ifSpeed and ifHighSpeed
discontinuity-time	ifCounterDiscontinuityTime
in-octets	ifHCInOctets
in-unicast-pkts	ifHCInUcastPkts
in-broadcast-pkts	ifHCInBroadcastPkts
in-multicast-pkts	ifHCInMulticastPkts
in-discards	ifInDiscards
in-errors	ifInErrors
in-unknown-protos	ifInUnknownProtos
out-octets	ifHCOutOctets
out-unicast-pkts	ifHCOutUcastPkts
out-broadcast-pkts	ifHCOutBroadcastPkts
out-multicast-pkts	ifHCOutMulticastPkts
out-discards	ifOutDiscards
out-errors	ifOutErrors

gNMI - Python

General https://github.com/akarneliuk/pygnmi

Cisco: https://github.com/cisco-ie/cisco-gnmi-python (uses Go implementation)

```
https://github.com/cisco-ie/cisco-gnmi-python/tree/master/github.com/openconfig
```

General https://github.com/google/gnxi (basic operations in Go)

```
https://github.com/google/gnxi/blob/master/gnmi_subscribe/gnmi_subscribe.go
```

gNMI - Go

CLI version: https://github.com/openconfig/gnmic/

Server: https://pkg.go.dev/github.com/google/gnxi/gnmi

Client: https://github.com/openconfig/gnmi

Arista https://github.com/aristanetworks/goarista

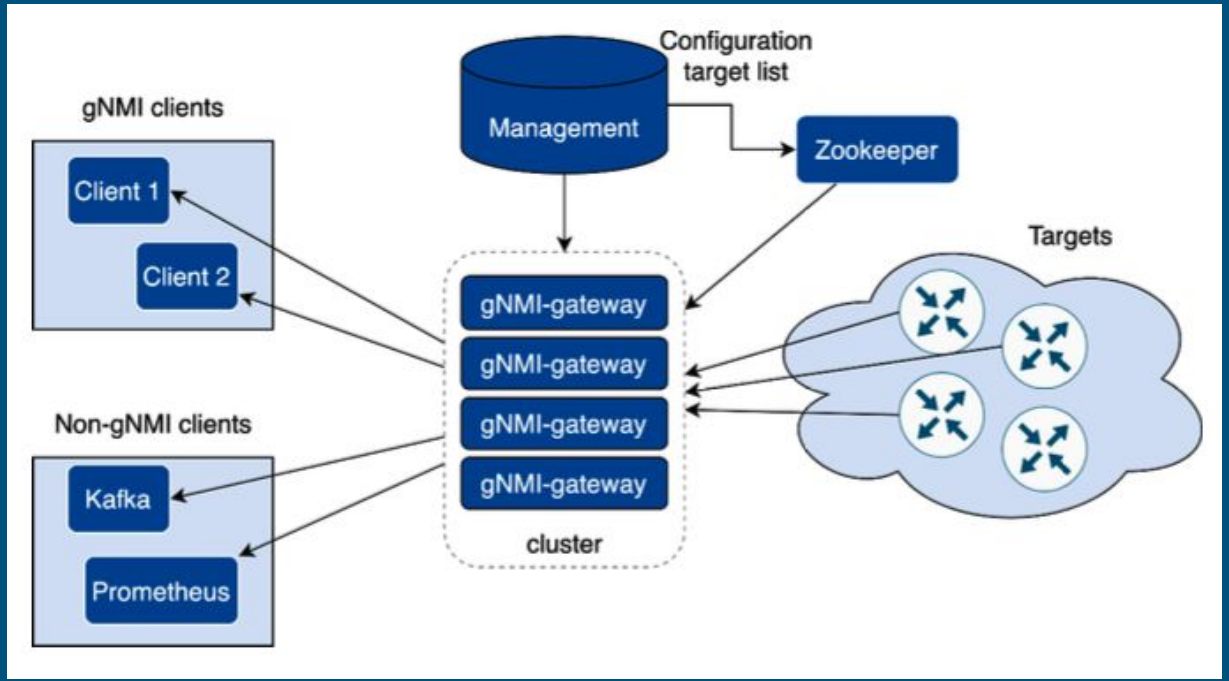
Juniper https://github.com/Juniper/openconfig-gnmi/

gNMI - Gateway https://github.com/openconfig/gnmi-gateway (Netflix initiative)

gNMI - Go

```
subReq := &gnmi.SubscribeRequest{
	Request: &gnmi.SubscribeRequest_Subscribe{
		Subscribe: &gnmi.SubscriptionList{
			Subscription: []*gnmi.Subscription{
				{
					Path: &gnmi.Path{
						Origin: "openconfig",
						Elem: []*gnmi.PathElem{
							{Name: "interfaces"},
							{Name: "interface"},
							{Name: "state"},
							{Name: "counters"},
							{Name: "in-octets"},
						},
					},
					Mode: gnmi.SubscriptionMode_ON_CHANGE, // Or SAMPLE 30s default

				},
			},
		},
	},
}
```

gNMI Gateway



https://netflixtechblog.com/simple-streaming-telemetry-27447416e68f

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		

Deployment and dependencies

Python importing modules

When importing a module without specifying its specific version, Python typically imports the first encountered module that matches the given name following a file path criteria (built-in modules, sys.path, site-packages)

Therefore, effectively managing dependencies and understanding the import search sequence becomes crucial to ensure the correct module version is imported.

In complex projects with many dependencies, managing versions and resolving conflicting dependencies can become challenging.

Different versions of dependencies might introduce breaking changes or compatibility issues with other libraries, leading to problems.

Python dependency files

command: `strace -e openat python3.8 <program.py> # with paramiko & pysnmp)`

```
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/extensions.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/ipaddress.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/general_name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/random.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bisect.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/_parseaddr.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/calendar.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/locale.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/charset.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/base64mime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/quoprimime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/errors.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/encoders.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/quopri.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/oid.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/aead.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ciphers.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/cmac.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ec.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/rsa.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/binding.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/_conditional.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/opt/pyca/cryptography/openssl/openssl.cnf", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/primitives/serialization/__pycache__/pkcs12.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/client.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/agent.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/tempfile.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/shutil.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bz2.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/_compression.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/lib-dynload/_bz2.cpython-38-x86_64-linux-gnu.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libbz2.so.1.0", O_RDONLY|O_CLOEXEC) = 3
.
.
.
```

269 files !!

Python upgrade and deployment nightmare

What to pay attention:

- Third-Party libraries compatibility.
- System and OS version.
- RollBack Plan
- Follow OS standard installation when possible.

https://medium.com/@damngoodtech/the-great-python-package-management-war-49f25df33d26

https://packaging.python.org/en/latest/guides/tool-recommendations/

https://opensource.com/article/19/4/managing-python-packages

Conclusion: Python Package Management is a Nightmare!

Python versions and OS

Python 3.3 (2012) created **venv** (https://docs.python.org/3/whatsnew/3.3.html)

Virtual environments allow developers to isolate project dependencies, ensuring that each project can have its own set of libraries and packages without interfering with system-wide or other project dependencies.

Using pip freeze a good practice.

Go dependencies is simple

```
$ cat go.mod
go 1.20

require (
      github.com/urfave/cli/v3 v3.0.0-alpha2
      golang.org/x/crypto v0.7.0
      gopkg.in/yaml.v3 v3.0.1
      github.com/cpuguy83/go-md2man/v2 v2.0.2 // indirect
      github.com/mattn/go-colorable v0.1.12 // indirect
      github.com/mattn/go-isatty v0.0.14 // indirect
      github.com/rs/zerolog v1.29.0 // indirect
      github.com/russross/blackfriday/v2 v2.1.0 // indirect
      github.com/xrash/smetrics v0.0.0-20201216005158-039620a65673 // indirect
      golang.org/x/sys v0.6.0 // indirect
)

$ # strace -e openat ./vlab -c -f readconfig.go
openat(AT_FDCWD, "/sys/kernel/mm/transparent_hugepage/hpage_pmd_size", O_RDONLY) = 3
openat(AT_FDCWD, "/etc/localtime", O_RDONLY) = 3
```

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		

Network Automation community tools

Some Python tools

Paramiko (https://github.com/paramiko/paramiko)

Netmiko (https://github.com/ktbyers/netmiko)

PyGNMI (https://github.com/akarneliuk/pygnmi)

Scrapli (https://github.com/carlmontanari/scrapli)

NetTowel (https://github.com/InfrastructureAsCode-ch/nettowel)

Netconan (https://github.com/intentionet/netconan)

Some Python frameworks

Vrnetlab (https://github.com/vrnetlab/vrnetlab)

Netlab (https://github.com/ipspace/netlab)

Mininet (https://github.com/mininet/mininet)

StackStorm (https://github.com/StackStorm/st2)

Noir (https://github.com/nornir-automation/nornir)

Ansible (https://github.com/ansible/ansible)

Salt/Saltstack (https://github.com/saltstack/salt)

Some Go initiatives

Scrapligo (https://github.com/scrapli/scrapligo)

Terraform (https://github.com/hashicorp/terraform)

Prometheus (https://github.com/prometheus/prometheus)

Telegraf (https://github.com/influxdata/telegraf)

The versus table

	Python	Go
Language release and core development		
Implementations		
Memory, speed, parallelism, concurrency and Network (IO)		
IP and network native libraries		
Program Logging, Error handling and Exceptions		
Network access methods		
Deployment and dependencies		
Network Automation community tools		

Conclusions

Conclusions

Today Python and Go are powerful, and leaders for Network Automation.

Go main advertising flag in performance.

Python main advertising flag is community and implementation.

Python has so many good projects besides Network Automation.

Go is new and definitely has perhaps more potential for Network Automation.

Other languages to consider ? Rust, Swift, Ruby ?

Thank you !

Drop a message and let’s talk about network automation, discuss solutions for network performance, network simulation, traffic analysis, network management, and more. :-)

www.telcomanager.com

Linkedin : https://www.linkedin.com/in/claus-topke/

Work email: claus@telcomanager.com

Personal email: claus.topke@gmail.com