First, a message from our sponsor...
What I picture in my head...
What it ends up looking like...

Actually that’s not quite right. The guy who made this is **clearly** more talented.
Who am I?

Frank Seesink

• Senior Network Engineer, UNC Chapel Hill
• Part of network DevOps group
• Involved in network automation for years
• Love languages, both human & computer
• Programming since I was 12 years old
• Formally B.S. in Computer Science with all coursework for an M.S. in same
• JOAT - databases, OSes, networking,...
Story time...

Red Hat
OpenShift

db

SQLite

Work environment

NETMíKO

Nornir

Gunicorn

django
Story time...

In January 2022, I was in a rut...
Why Go?

- Python’s creator, Guido van Rossum, worked at Google from 2005-2012.
- For years Google heavily used Python internally and even offered Python classes to its employees.
  - https://developers.google.com/edu/python
- Google had also hired Rob Pike and Ken Thompson of Bell Labs (UNIX, C) fame. They, along with Robert Griesemer, created Go.
- In 2013 Guido van Rossum went to work at Dropbox. (Dropbox was known to use Python.) That seemed odd.
- In 2014 Google publicly released Kubernetes, which is written in Go.

The writing was on the wall?
Why Go?

“Language of the cloud”
Go (Golang)

https://go.dev/
Go (Golang)

- **Learning Go**
  https://www.linkedin.com/learning/learning-go

- **Go for Python Developers**
  https://www.linkedin.com/learning/go-for-python-developers

- https://learnxinyminutes.com/docs/go/
EASILY BUILD NATIVE APPS THAT WORK EVERYWHERE

An easy to learn toolkit for creating graphical apps for desktop, mobile and web. Our free and open source libraries combine the simplicity of the Go programming language with a carefully crafted library of widgets to simplify coding any app. But also, Fyne apps can be built for all platforms and stores!

https://fyne.io/
To Learn a Programming Language...

1. You need to program in it
2. You need to program in it
3. You need to program in it
4. You need to have a project/goal
Initial Go Test Project

https://github.com/fseesink/mysetup
History of Programming Languages

0/1

1940s

Present
History of Programming Languages

0/1

1940s

BASIC

C

C++

&

Present

PHP

R

GO
History of Programming Languages
## History/Comparison

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First appeared</strong></td>
<td>1972</td>
<td>1992</td>
<td>2009</td>
</tr>
<tr>
<td><strong>Designed by</strong></td>
<td>Dennis Ritchie</td>
<td>Guido van Rossum</td>
<td>Robert Griesemer, Rob Pike, Ken Thompson</td>
</tr>
<tr>
<td><strong>Typing</strong></td>
<td>Static, weak, manifest, nominal</td>
<td>Duck, dynamic, strong typing</td>
<td>Inferred, static, strong, structural, nominal</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>32</td>
<td>35</td>
<td>25</td>
</tr>
</tbody>
</table>
# Features

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Built-in concurrency</strong></td>
<td>N/A</td>
<td>N/A</td>
<td>Go routines</td>
</tr>
<tr>
<td><strong>Concurrency via libraries</strong></td>
<td>fork() *provides access to underlying OS concurrency features</td>
<td>multiprocessing.concurrent.futures asyncio</td>
<td></td>
</tr>
<tr>
<td><strong>Native multi-core support</strong></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A due to GIL</td>
</tr>
<tr>
<td><strong>Memory Management</strong></td>
<td>malloc()/free() *developer responsible for all memory mgmt</td>
<td>Garbage Collection</td>
<td>Garbage Collection</td>
</tr>
</tbody>
</table>
## Libraries/Modules

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Library</strong></td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Package ecosystem</strong></td>
<td>N/A</td>
<td>PyPI.org</td>
<td>via VCS such as Git</td>
</tr>
<tr>
<td><strong>Example package import</strong></td>
<td><code>#include &lt;stdio.h&gt;</code></td>
<td><code>import netmiko</code></td>
<td><code>import ( &quot;github.com/nornir-automation/gornir/pkg/gornir&quot;)</code></td>
</tr>
<tr>
<td><strong>Largest library (e.g., AI/ML, data analysis)</strong></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
</tbody>
</table>
## Workflow

<table>
<thead>
<tr>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/usr/local/bin/python3</code></td>
<td><code>package main</code></td>
</tr>
<tr>
<td><code>print(&quot;Hello world&quot;)</code></td>
<td><code>import &quot;fmt&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>func main() {</code></td>
</tr>
<tr>
<td></td>
<td><code>  fmt.Println(&quot;Hello world&quot;)</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td>$ python3 helloworld.py</td>
<td>$ go run helloworld.go</td>
</tr>
<tr>
<td>or if permissions set, simply</td>
<td>or</td>
</tr>
<tr>
<td>$ helloworld.py</td>
<td>$ go run .</td>
</tr>
<tr>
<td>Compile and run executable with</td>
<td>to run interactively.</td>
</tr>
<tr>
<td>$ go build .</td>
<td></td>
</tr>
<tr>
<td>$ helloworld</td>
<td></td>
</tr>
</tbody>
</table>
## Workflow Performance

<table>
<thead>
<tr>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/usr/local/bin/python3</code></td>
<td><code>package main</code></td>
</tr>
<tr>
<td><code>print(&quot;Hello world&quot;)</code></td>
<td><code>import &quot;fmt&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>func main() {</code></td>
</tr>
<tr>
<td></td>
<td><code>    fmt.Println(&quot;Hello world&quot;)</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

$ time python3 helloworld.py
Hello world
python3 helloworld.py 0.02s user 0.02s system 36% cpu 0.111 total

$ time go run helloworld.go
Hello world
go run helloworld.go 0.14s user 0.29s system 49% cpu 0.860 total

$ go build helloworld.go
$ time ./helloworld
Hello world
./helloworld 0.00s user 0.00s system 2% cpu 0.135 total
## Final Program Size

<table>
<thead>
<tr>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>#!/usr/local/bin/python3</td>
<td>package main</td>
</tr>
<tr>
<td>print(&quot;Hello world&quot;)</td>
<td>import &quot;fmt&quot;</td>
</tr>
</tbody>
</table>
| **C version**<br>TOTAL == 32 KB<br>or 0.032 MB | func main() {<br>fmt.Println("Hello world")<br>}

| 46 bytes: helloworld.py | 72 bytes: helloworld.go |
| 310 MB: Python install (*)&nbsp; | 238 MB: Go install (*) |
| To run a Python script, you need Python installed. | 1.8 MB: helloworld binary |
| TOTAL == **~310 MB** | To run a Go compiled app, you just need the binary. |
| (*) v3.11.5 macOS installation on disk | (*) v1.21.0 macOS installation on disk |
# Language Similarities

<table>
<thead>
<tr>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>import os</code></td>
<td><code>package main</code></td>
</tr>
<tr>
<td><code>def itsvalid():</code></td>
<td><code>func itsvalid()</code></td>
</tr>
<tr>
<td><code>    print(&quot;Valid day of the month&quot;)</code></td>
<td><code>    fmt.Println(&quot;Valid day of the month&quot;)</code></td>
</tr>
<tr>
<td><code>    cwd = os.getcwd()</code></td>
<td><code>    cwd, _ := os.Getwd()</code></td>
</tr>
<tr>
<td><code>    print(cwd)</code></td>
<td><code>    fmt.Println(cwd)</code></td>
</tr>
<tr>
<td><code>def main():</code></td>
<td><code>func main()</code></td>
</tr>
<tr>
<td><code>    # Variable assignment</code></td>
<td><code>    // Variable assignment</code></td>
</tr>
<tr>
<td><code>    name = &quot;Frank&quot;</code></td>
<td><code>    name := &quot;Frank&quot;</code></td>
</tr>
<tr>
<td><code>    day = 19</code></td>
<td><code>    day := 19</code></td>
</tr>
<tr>
<td><code>    if day &gt;= 1 and day &lt; 31:</code></td>
<td><code>    if day &gt;= 1 &amp;&amp; day &lt; 31 {</code></td>
</tr>
<tr>
<td><code>        itsvalid()</code></td>
<td><code>        itsvalid()</code></td>
</tr>
<tr>
<td><code>if __name__ == &quot;__main__&quot;:</code></td>
<td><code>if __name__ == &quot;__main__&quot;:</code></td>
</tr>
<tr>
<td><code>    main()</code></td>
<td><code>    main()</code></td>
</tr>
<tr>
<td><code>fmt.Println(name)</code></td>
<td><code>fmt.Println(name)</code></td>
</tr>
</tbody>
</table>
Global Interpreter Lock (GIL)
“In CPython, the global interpreter lock, or GIL, is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once. The GIL prevents race conditions and ensures thread safety. A nice explanation of how the Python GIL helps in these areas can be found here. In short, this mutex is necessary mainly because CPython's memory management is not thread-safe.”

- https://wiki.python.org/moin/GlobalInterpreterLock
Python’s Lack of Concurrency

1990s

100%

50%

2000s

25%

6.25%

3.125%

== CPU core

== Python
To bypass the GIL

To use multiple threads/cores, you must take action. This requires extra effort.

For example,

- *multiprocessing* or *concurrent.futures* module in Python standard library. (You must use processes and not threads in latter. Otherwise it stays within a single core.)

- Use modules like Nornir (which use *concurrent futures*)

Async is does NOT help here. That is cooperative multi-threading. Again, single core.
Disclaimer

Network Automation tends to be **I/O-bound vs. CPU-bound**
Possible Python Future

PEP 703 – Making the Global Interpreter Lock Optional in CPython

CPython’s global interpreter lock ("GIL") prevents multiple threads from executing Python code at the same time. The GIL is an obstacle to using multi-core CPUs from Python efficiently. This PEP proposes adding a build configuration (--disable-gil) to CPython to let it run Python code without the global interpreter lock and with the necessary changes needed to make the interpreter thread-safe.

https://peps.python.org/pep-0703/
Go routines

1. Put ‘go’ in front of a function call.
2. ...
3. Profit!

---

<table>
<thead>
<tr>
<th>Main routine waits for function</th>
<th>Main routine keeps going</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func main() {</code></td>
<td><code>func main() {</code></td>
</tr>
<tr>
<td><code>// Variable assignment</code></td>
<td><code>// Variable assignment</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>dosomething()</code></td>
<td><code>go dosomething()</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
Dependency Hell
Dependency Hell

Program 1

Module X
v1

Program 2

Module X
v2
Dependency Hell

Program 1

Module X
v1

Module X
v2

Program 2

site-packages

X
Dependency Hell (cont.)

Program

Netmiko

Nornir

site-packages
When you first learn Python, it’s like this

https://xkcd.com/353/
Eventually, it becomes this...

https://xkcd.com/1987/

MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADEd THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.
What about Python’s ability to run on different platforms?
Go Cross-Compilation
Go Cross-Compilation

- Go creates binary executables specific to an OS/architecture (e.g., x64 Windows, ARM64 Linux)

- Go can cross-compile to ANY supported OS/architecture combination FROM any supported OS/architecture. Simply set GOOS and GOARCH environment variables.

```bash
$ GOOS=linux GOARCH=arm64 go build .
```
So when should you use Go?
It Depends.
What makes Go worth considering

- Pythonic code (relatively easy transition)
- Go routines / native multi-core support
- Single binary executable with NO EXTERNAL DEPENDENCIES
- Can compile to any supported architecture/OS from a single platform
- Performant: best balance between coding speed and execution speed
- BONUS: Fyne is a nice, cross-platform GUI framework
Thank You

https://frank.seesink.com/presentations/Internet2TechEx-Fall2023/

Frank Seesink
frank@seesink.com
frank@unc.edu