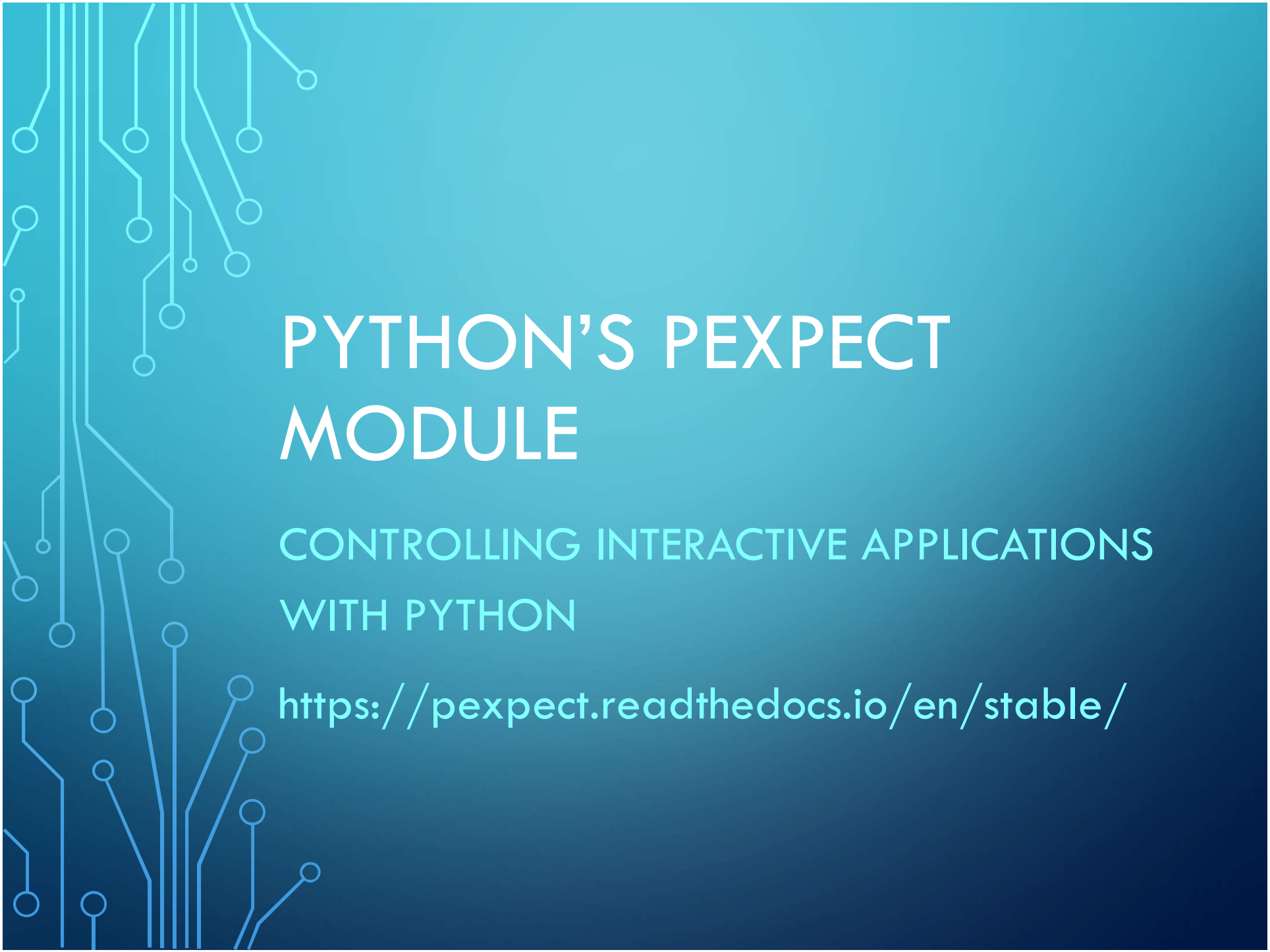
PYTHON'S PEXPECT MODULE
CONTROLLING INTERACTIVE APPLICATIONS WITH PYTHON
https://pexpect.readthedocs.io/en/stable/

# WHY PEXPECT?

- You want to automate OS commands including remote access (ssh, scp, ftp, rsync, etc.)
- You want to automate software that expects human interaction, e.g. for testing or because you don't have source code
- You don't want to learn the subtleties of Python's subprocess module (https://docs.python.org/2.7/library/subprocess.html)

# INSTALLATION UNDER ANACONDA

```
conda config --add channels conda-forge
conda install pexpect
```

# CAVEAT: BEST ON LINUX FOR NOW

Pexpect can be used on Windows to wait for a pattern to be produced by a child process, using pexpect.popen_spawn.PopenSpawn, or a file descriptor, using pexpect.fdpexpect.fdspawn. This should be considered experimental for now.

pexpect.spawn and pexpect.run() are not available on Windows, as they rely on Unix pseudoterminals (ptys). Cross platform code must not use these.

# AUTOMATION OF OS COMMANDS

```
# This connects to the openbsd ftp site and
# downloads the recursive directory listing.
import pexpect
child = pexpect.spawn('ftp ftp.openbsd.org')
child.expect('Name .*: ')
child.sendline('anonymous')
child.expect('Password:')
child.sendline('noah@example.com')
child.expect('ftp> ')
child.sendline('lcd /tmp')
child.expect('ftp> ')
child.sendline('cd pub/OpenBSD')
child.expect('ftp> ')
child.sendline('get README')
child.expect('ftp> ')
child.sendline('bye')
```

# AUTOMATION OF INTERACTIVE CODE

**interactivecode.py**

```
var = raw_input("Please enter something: ")
print "you entered", var
```

**runcode.py**

```
import pexpect
onerun = "python interactivecode.py"
oneinput = "hi"
child = pexpect.spawn('/bin/bash', ['-c',onerun])
child.expect("Please enter something:")
child.sendline(oneinput)
child.expect("you entered")
print child.after + child.before
```

# AUTOMATION OF INTERACTIVE SOFTWARE YOU CAN'T CHANGE DIRECTLY

`genmodels.py`

Pexpect code I'm writing to generate SSP models with

`csp_galaxev`

a compiled binary code with many manual choices

(Bruzual-Charlot stellar population models)

*likewise, an undergrad working with me wrote a Pexpect code to automate Pegase SPS model generation*

# WHAT I'VE LEARNED 1

- **If you want to SEE what's happening to debug, use "interact":**

```
cspexpect =
csp.expect([expectphrase,pexpect.TIMEOUT])
if (cspexpect == 0):
    csp.delaybeforesend=3.0
    csp.sendcontrol('c')
else:
    if (cspexpect == 1):
        print "timed out"
        csp.interact()
```

- **Experiment with delays to ensure program completion**

# WHAT I'VE LEARNED 2

- **You can't "expect" anything non-unique**

```
expectphrase = "Computing model No. %2u" % imodel
   print expectphrase
   cspexpect = csp.expect([expectphrase,pexpect.TIMEOUT])
   if (cspexpect == 0):
       #"BC_GALAXEV SSP sed in file [_] ="
       csp.sendline(flatparsi[0])
       #"Include attenuation by dust? Y/[N]"
       csp.sendline(flatparsi[1])
       #"Choice ="
       csp.sendline(flatparsi[2])
       #"Exponential with e-folding time TAU (Gyr) ="
       csp.sendline(flatparsi[3])
```

- **Use packages like 'os' and 'shutil' to change directories and copy files**