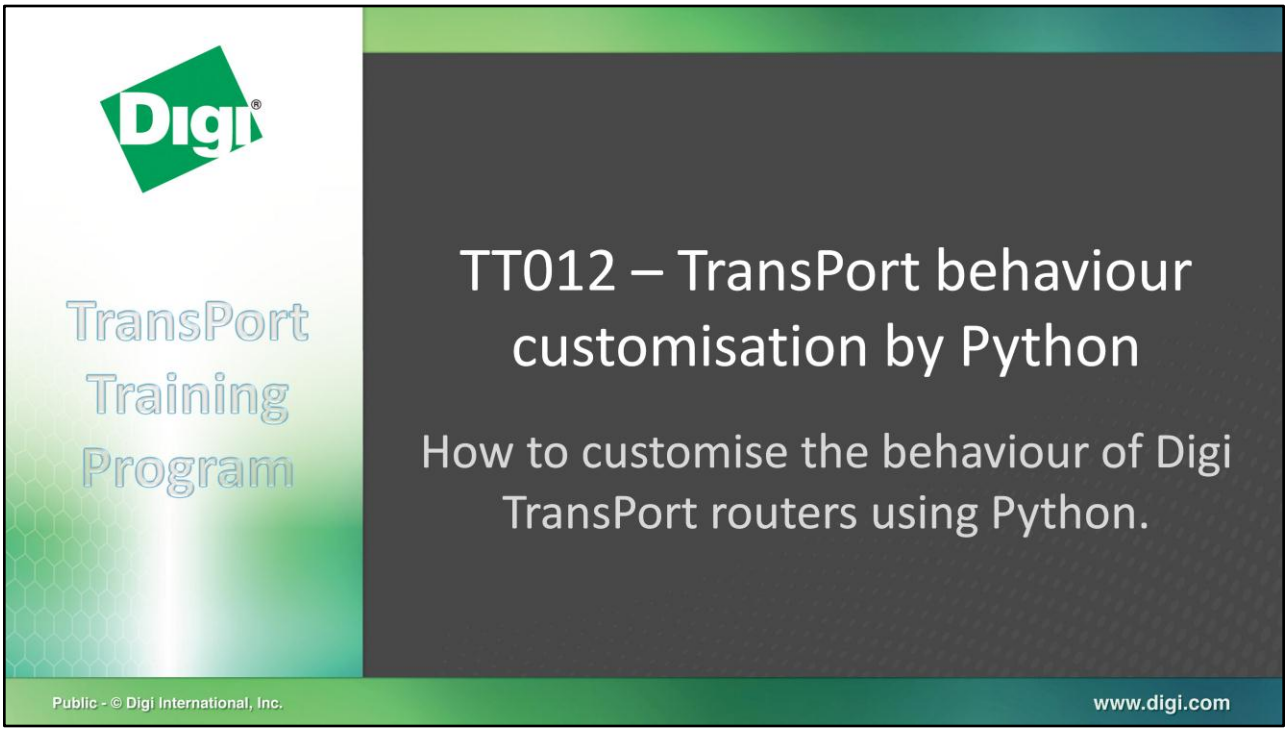

TransPort
Training
Program
TT012 – TransPort behaviour customisation by Python
How to customise the behaviour of Digi TransPort routers using Python.
Public - © Digi International, Inc.
www.digi.com

Low Level Vs High Level Programming Languages

- Which of you has ever written a program or script?
- Assembler – extremely low level
- C++ very low level. Compiled code is platform specific.
- Java – higher – compiled “byte code” can run on multiple operating systems.
- VB.net very high – runs on Windows only
- Python – very high
  - Huge number of library resources
  - Automatic memory management
  - Normally interpreted but with the option to compile
  - Platform independent compiled code can run on multiple operating systems
  - Small amount of code can achieve a lot. Normally many more lines of code needed to achieve the same result in C.
  - Object Oriented kind of, its optional.
  - Multi-threaded

How many of you have done any programming ?
You’re favourite languages are ?

python

- Open-source, flexible language that puts decision-making intelligence on the router under end user control.
- Very simply to write basic code
- Easy to experiment interactively
- Powerful extensions
- Broad support
- Less to type, less to debug, less to maintain
- It's named after Monty Python's Flying Circus

It's named Python after Monty Python's Flying Circus! And the official IDE is named IDLE after Eric Idle

Lots of resources

- Online
  - Python.org (http://docs.python.org)
  - Dive Into Python (http://diveintopython.org)
  - Python Challenge (http://www.pythonchallenge.com)
- Books
  - Learning Python by Mark Lutz
  - Programming Python by Mark Lutz
  - Python Cookbook by Alex Martelli, Anna Martelli Ravenscroft, David Ascher

A basic program

```
def hello():
  """Prints a salutation to the user"""
  # Say hello
  print "Hello World!"

if __name__ == "__main__":
  hello()
```

Note that indentation dictates structure, not keywords or special tokens.

A Really Basic Program

```
print "Hello World!"
```

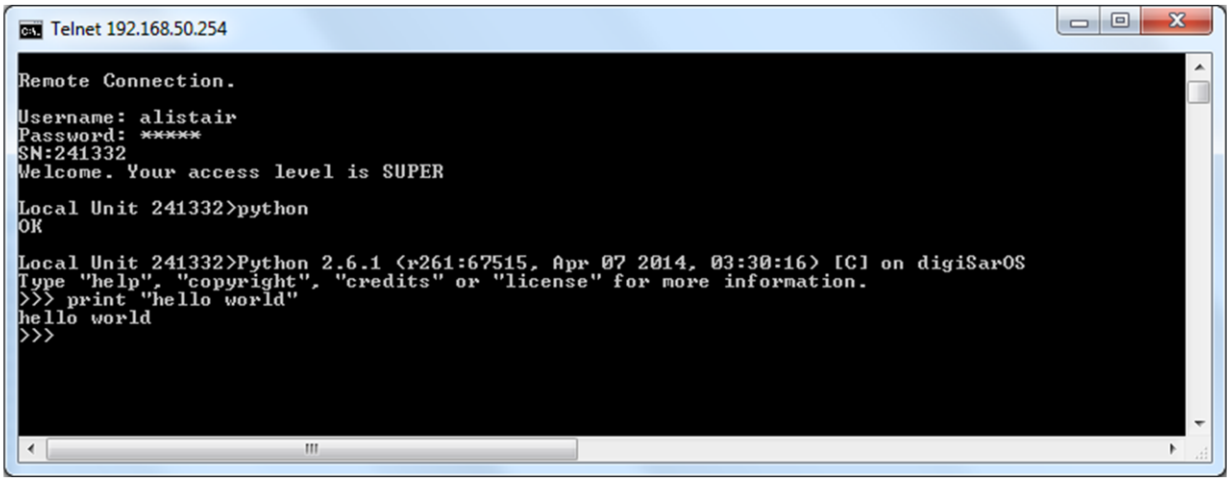
Indentation Matters

```
if 1+1==2:
   print "answer correct!"
   print "you did maths!"


if 1+1==3:
   print "answer correct"
print "you did maths!"
```


Explain the output of the above programs

Interactive Interpreter



Really cool for checking syntax and debugging.

Core Data Types

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Files

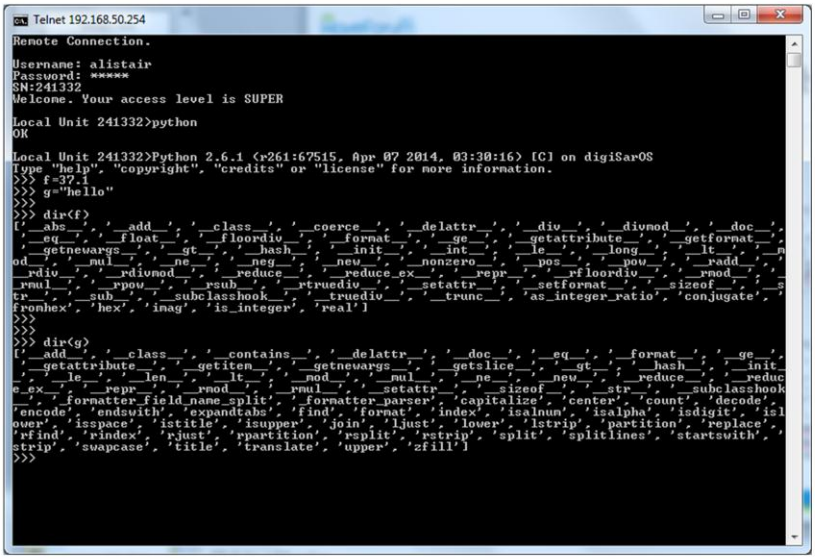
Data types – Scalars

- Numbers
  - Three types (int, float and long)
  - Varying representations (10, 0xa, 012)
  - Don't worry too much about type (except division)
  - 5 /2 = 2
  - 5.0 / 2 = 2.5
- bool
  - Values (True and False)
- None

Strings

- Normal – "Hello\nThere" (allows escapes)
  Triple-quoted (preserves entered text)
  """Hello
  There"""“
- Raw – r"Hello\nThere" (escapes ignored)
- Unicode – u"\u0496"(inserts Җ)
  - Allows unicode escapes, creates unicode strings, a different type

dir shows available attributes



Conditional Execution

```
if <expr>:
  <…>
elif <expr>:
  <…>
else <expr>:
  <…>
```

There is no *goto* or *case* statement in Python

Loops

```
for <var> in <sequence>:
  <…>
```

```
while <expr>:
  <…>
```

Variations:

- ***break* and *continue* affect looping**
- ***pass* can act as a placeholder**
- ***(x)range(i[, j[, k]])* is extremely useful for *for* statements**

Function

- Functions organize your code and minimise redundancy
- Function blocks begin with the keyword def followed by the function name and parentheses ()
- Any input parameters (arguments) should be inserted into the parentheses
- Functions can optionally return one or more results (comma separated)

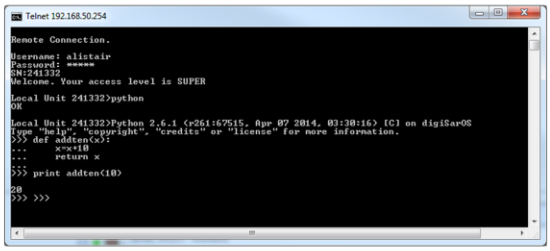
Function Declaration:

```
def addten(x):
    """This function adds 10 to supplied argument"""
    x = x +10
    return x
```

Function Call:

```
print addten(10)
```

Demo:



Functions

```
def arg_info(arg=42, *pos_args, **kw_args):
  """Doc-string"""

  if (arg == 42):
     print "arg has default value"

  # Print positional arguments
  for i in len(pos_args):
  # Use string formatting operator
   print "Positional argument %d is %v" % (i, pos_args[i])

  print "Keyword arguments"
  for key in kw_args.keys():
    print "\t", key, ": ", kw_args[key]

  return len(pos_args) + len(kw_args)
```


http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/

Modules

We already have seen one

```
# hello.py
def hello():
  """Prints a salutation to the user"""
  # Say hello
  print "Hello World!"

if __name__ == "__main__":
  hello()
```

```
import hello
hello.hello()
```

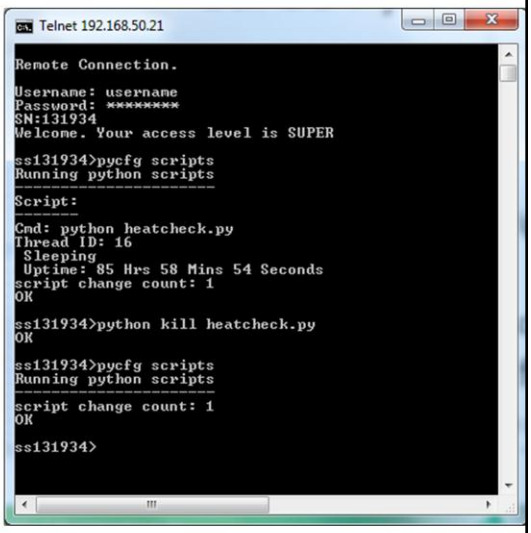
Threading

To run a Python program

- Write the program and save as a text file “filename.py”
- Use an FTP client to FTP it to the root folder via the TransPort routers built in FTP server
- Execute the program from the CLI (e.g. telnet or SSH) “python filename.py”

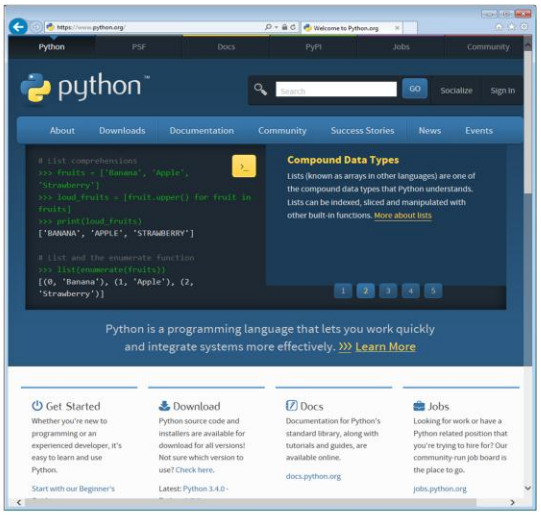
SarOS Python Commands

- pycfg scripts
  - View running scripts
- python kill filename.py
  - Terminate a named python script
- python kill all
  - Terminate all running python scripts
- cmd <n> autocmd “python filename.py”
  - configure a Python program to run automatically on start up



Customisation by Python

- SAROS is very configurable and has a huge amount of flexibility built in. Flexibility is further increased by the Python scripting language.
- The behaviour of the router can be quite easily customised by Python.
- One “trick” is to find the CLI command to do what you need. E.g. if you want to send an email use the SarOS “email” command instead of trying to use a Python SMTP library.
- Typical simple customisation: monitor the status of something, and based on that change the configuration on the fly – or generate an event etc
- The Python Version is 2.6.1



Some Simple Real Customisations

- Check for MAC addresses of all clients that connect to WiFi Access Point and send to server via TCP.
- Python Web Server running completely custom pages which are capable of changing router's configuration.
- Check available cellular networks (dual SIM) on start-up. Make choice based upon signal strength, ping test, radio technology.
- Control LEDs on front panel based on status of VPN tunnel
- Receive SMS messages and send data out of serial port
- Log GPS information and Wi-Fi/cellular status to USB FLASH drive
- Interact programmatically with T1/T2 cards
- CCTV – turn on lights after PIR detection
- Roaming SIM network support – check through all networks available to a roaming SIM card and make the selection based upon prioritisation rules and tests.

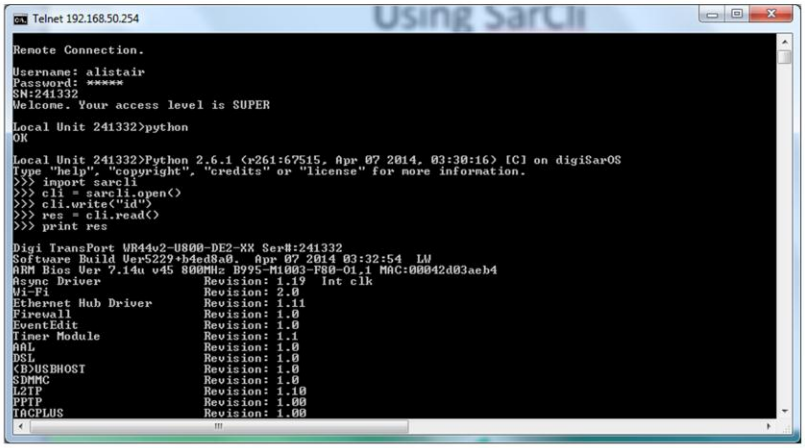
Import the sarcli module

It's amazing what can be achieved by issuing CLI commands.

```
import sarcli

cli = sarcli.open()
cli.write("id")
res = cli.read()
print res
cli.close()
```

Demo



Other Built in Modules

From the interactive interpreter:

>>>help()
help> modules

This will list all the Python modules available in SarOS !

User can add more modules:

1) FTP zip file with modules onto router's root folder
2) Import it into Python code:

>>>import sys
>>> sys.path.insert(0,'paramiko.zip')
>>>import paramiko.sftp

BaseHTTPServer	cmd	keyword	sets
BaseWizard	code	led	sgmllib
Bastion	codecs	linecache	sha
CGIHTTPServer	codeop	locale	shelve
ConfigParser	collections	logging	shlex
Cookie	colorsys	macpath	shutil
DocXMLRPCServer	commands	macurl2path	site
Fleet	compileall	mailbox	smtpd
FleetApps	contextlib	mailcap	smtplib
HTMLParser	cookielib	markupbase	sndhdr
MimeWriter	copy	marshal	socket
Queue	copy_reg	math	sre
SimpleHTTPServer	csv	md5	sre_compile
SimpleXMLRPCServer	cwm	mhlib	sre_constants
SocketServer	datetime	mime	sre_parse
StringIO	dbhash	mimetools	ssl
UserDict	dcloudconnect	mimetypes	stat
UserList	decimal	mimify	statvfs
UserString	difflib	misc	string
_LWPCookieJar	digicanbus	modulefinder	stringold
_MozillaCookieJar	digihw	multifile	stringprep
__builtin__	digij1708	mutex	strop
__future__	digilicense	netrc	struct
_abcoll	digilock	new	subprocess
_ast	digisms	nntplib	sunau
_bisect	digiwdog	ntpath	sunaudio
_bytesio	digiweb	nturl2path	symbol
_codecs	dircache	numbers	symtable
_collections	dis	opcode	sys
_csv	doctest	operator	tabnanny
_fileio	dumbdbm	optparse	tarfile
_functools	dummy_thread	os	telnetlib
_heapq	dummy_threading	os2emxpath	tempfile
_md5	email	owner	termios
_random	encodings	parser	textwrap
_sha	errno	pdb	this
_sha256	exceptions	pickle	thread
_sha512	filecmp	pickletools	threading
_socket	fileinput	pih	time
_sre	fnmatch	pipes	timeit
_ssl	formatter	pkgutil	toaiff
_strptime	fpformat	platform	token
_struct	fractions	plistlib	tokenize
_symtable	ftplib	pocean2	trace

Useful SarOS commands for scripts

- Send an email
  - email "EMAIL <TO> <FROM> <SUBJECT> <TEMPLATE> [ATTACHMENT LIST]"
- Send an SMS
  - sendsms "SENDSMS <dest> <message>"
- Create an event
  - setevent "SETEVENT <text> [<priority> [trapcode]]"
    - Can trigger SMS
    - Can trigger email
    - Can trigger SNMP trap
    - Can trigger SYSLOG event
- Flash the LEDs
  - flashleds
- Take full control of the LEDs under software control
  - ledbnk LEDBNK <led> <bank>|?
  - ledn LEDN <led> <bank> [<state>]

Digi Python Modules

- `sarcli`
  - Interact with the Digi TransPort Command Line Interface
- `digicanbus`
  - Use with the fleet daughter card Interfaces J1708, CAN bus /J1939
  - See http://ftp1.digi.com/support/documentation/AN_49_Digi_TransPort_Fleet_card.pdf
- `digihw`
  - Interact with telemetry cards, GPS and WR44 DIO
  - See http://ftp1.digi.com/support/documentation/AN_49_Digi_TransPort_Fleet_card.pdf
- `digisms`
  - Receive SMS messages in Python scripts
  - http://www.digi.com/wiki/developer/index.php/Module:digisms
- `digiweb`
  - Use the built in SarOS web server to display content

HOPS I

1. Write a small script to flash the LEDs or use the example below to print out SMS messages
2. FTP the script to your router
3. Execute the script from the command line
4. Configure the script to autostart and reboot

```
import sys, digisms, time

# Test script to receive SMS messages
# Note also that a script will only get the SMS if it
# matches the callerid list in modemcc

def smscb(SMSMessage):
    print "sms source: %s" % SMSMessage["source_addr"]
    print "sms message: %s" % SMSMessage["message"]

if __name__ == '__main__':
    h = digisms.Callback(smscb)
    while 1:
        time.sleep(1)
```

HOPS II

- Custom Web Server
- WR44 GPIO

httpd.zip

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from os import curdir, sep
import cgi
from BaseWizard import write_cli

PORT_NUMBER = 8000

#This class will handles any incoming request from
#the browser
class myHandler(BaseHTTPRequestHandler):

    #Handler for the GET requests
    def do_GET(self):
        if self.path=="/":
            self.path="/index.html"

        try:
            #Check the file extension required and
            #set the right mime type

            sendReply = False
            if self.path.endswith(".html"):
                mimetype='text/html'
                sendReply = True
            if self.path.endswith(".jpg"):
                mimetype='image/jpg'
                sendReply = True
            if self.path.endswith(".gif"):
                mimetype='image/gif'
                sendReply = True
            if self.path.endswith(".js"):
                mimetype='application/javascript'
                sendReply = True
            if self.path.endswith(".css"):
                mimetype='text/css'
                sendReply = True

            if sendReply == True:
                #Open the static file requested and send it
                f = open(curdir + sep + self.path)
                self.send_response(200)
                self.send_header('Content-type',mimetype)
                self.end_headers()
                self.wfile.write(f.read())
                f.close()
            return

        except IOError:
            self.send_error(404,'File Not Found: %s' % self.path)

    #Handler for the POST requests

    def do_POST(self):
        print "POST HANDLING"

        if self.path=="/send":
            form = cgi.FieldStorage(
                            fp=self.rfile,
                            headers=self.headers,
                            environ={'REQUEST_METHOD':'POST',
             'CONTENT_TYPE':self.headers['Content-Type'],
                            })

            result1 = write_cli('cmd 0 unitid %s' % form["unitid"].value)
            result2 = write_cli('ppp 1 password %s' % form["password"].value)
            print "Unit ID is: %s" % form["unitid"].value
            print "Password is: %s" % form["password"].value
            self.send_response(200)
            self.end_headers()
            self.wfile.write(result1)
            #self.wfile.write("Thanks %s !" % form["password"].value)
            return


try:
    #Create a web server and define the handler to manage the
    #incoming request
    server = HTTPServer(('', PORT_NUMBER), myHandler)
    print 'Started httpserver on port ' , PORT_NUMBER

    #Wait forever for incoming http requests
    server.serve_forever()

except KeyboardInterrupt:
    print '^C received, shutting down the web server'
    server.socket.close()
```