XMC 8.5 Workshop
Python Basics

Markus Nikulski
Sr. Corporate System Engineer

October 2020

Extreme™

string handling (concatenation)

```
str1 = "Hello"
str2 = "World"

print str1 + " " + str2 + "!"
print str1 , " " , str2 , "!"

print "%s %s!" % (str1,str2)

print "{} {}!" format(str1, str2)
```

Hello World!

string handling (padding)

```
print '%10s' % ('test',)
print '%4d' % (42,)
print '{:>10}'.format('test’)
print '{:4d}'.format(42)
```

```
     align right
```

```
print '%-10s' % ('test',)
print '{:10}'.format('test')
```

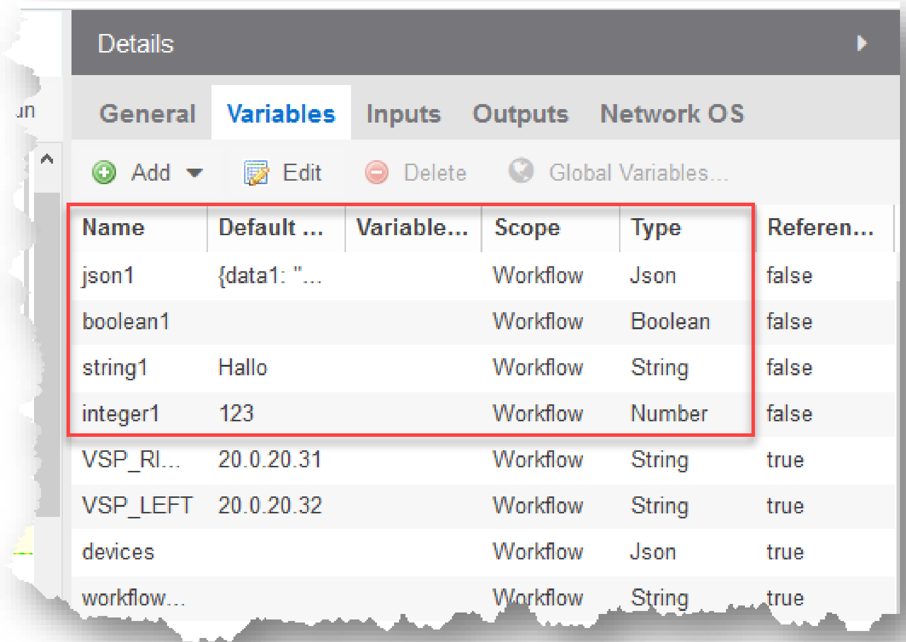
```
align left
```

```
print '{:^10}'.format('test')
```

```
  align center
```

variable casting

```
str1 = "10"
int1 = 20

print str1 + " " + str( int1 )
print "%s %s" % (str1,int2)

result = int(str1) * int1
```



Name	Default ...	Variable...	Scope	Type	Referen...
json1	{data1: "...		Workflow	Json	false
boolean1			Workflow	Boolean	false
string1	Hallo		Workflow	String	false
integer1	123		Workflow	Number	false
VSP_RI...	20.0.20.31		Workflow	String	true
VSP_LEFT	20.0.20.32		Workflow	String	true
devices			Workflow	Json	true
workflow...			Workflow	String	true

all **emc_vars** items are **strings**!

even if you declare the variable type

if expression

```
a = 1
b = 2

if a < b:
    print "a is lower than b"
elif a > b:
    print "a is grater than b"
else:
    print "a is equal to b"
```

```
a = 1
b = 2

if a != b:
    print "a is not equal b"

if not a == b:
    print "a is not equal b"

if a is not b:
    print "a is not equal b"
```

```
print "a is lower than b" if a < b else print "a is grater than b"
```

for loop

```
fruits = ["apple", "banana", "cherry"]

for x in fruits:
    if x == "apple":
        continue
    print x
    if x == "banana":
        break
```

```
for x in range(3):
        print x
```

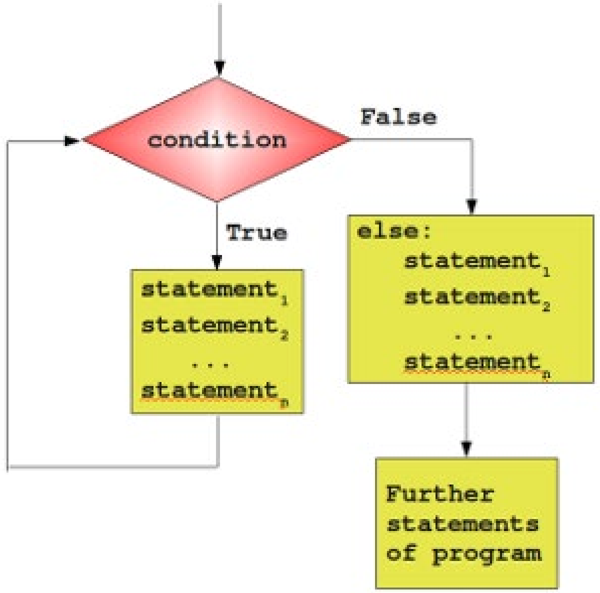
```
0
1
2
```

```
a_dict = {'color': 'blue', 'fruit': 'apple'}

for key, value in a_dict.iteritems():
    print key + '->' + value
```

while loop

```
i = 1

while i <= 7:
    print i
    i = i + 1
```

```
1
2
3
4
5
6
7
```



Python Function

Python Function

```
name = 'User'

################################
def my_function(name):

    print('Hello ' + name)

    return len(name)

################################

result = my_function( name )

print "found in name %s characters" % result
```

```
name = 'User'

################################
def my_function(name, age):

    print('Hello ' + name)

    moth = age * 12

    return len(name),moth

################################

(lenght,month) = my_function( name, 35 )

print "found in %s %s characters" % (lenght, month)
```

Use function on any place where you have repeating jobs to do.
Encapsulate it to small blocks of code called function.

Coding conventions

coding recommendation

1. Comment and Document
2. Create Descriptive Names
3. Don't Repeat Yourself
4. Check for Errors and Respond to Them
5. Split Your Code into Short, Focused Units
6. Don't Overdesign
7. Be Consistent
8. Keep Your Code Portable
9. Limit the line length and amount of lines
10. Code alignment

good vs bad coding

variable and function naming
Names have to explain the meaning of the content
Alignment helps faster reading

```
ssss='User'
c=0
bool=False
```

```
userName   = 'User'
userAmount = 0
userNew    = False
```

```
user_name   = 'User'
user_amount = 0
user_new    = False
```

good vs bad coding

```
name   = 'User'
result = 0

################################
def my_function():
    global result

    print('Hello ' + name)

    result = len(name)

################################

my_function()

print "found in name %s characters" %s result
```

Keep Your Code Portable

use function in/out data exchange

```
name = 'User’

################################
def my_function(name):

    print('Hello ' + name)

    return len(name)

################################

result = my_function( name )

print "found in name %s characters" %s result
```

good vs bad coding

Document your code

Header

```
#############################################
# XMC 8.2 Python script
# written by: Markus Nikulski
# e-mail:     mnikulski@extremenetworks.com
# date:       01. Oct.
# purpose:    upgrade older BOSS switches
#
# 1.4  20. Sep.  donald@duck.fun
#                extend the flexibility

__version__ = '1.4'

import time
import json

settings = {}
```

Function

```
####################################################
# written by:  Markus Nikulski
# propose:     determinate device site relationship
# inbound:     IP address (string)
# outbound:    site path (string)

def getSite(ipAddress):
    query = ''
```

variables

```
tftprootdir  = '/tftpboot/'          # root directory
imageDir     = 'firmware/images/'
sleepTimer   = {'diag' :1,           # minutes (min 1)
                'image':4            # minutes (3-5)
               }
```

good vs bad coding

Check for Errors and Respond to Them

```
search_list = []

for name in search_list:
    if name.startswith('C'):
        print 'Found'
        break
else:
    print 'No data exists'
```

Debugging

Debugging

syntax issues

```python
def my_funtion(data);
    print "%s World" % data

my_funtion('Hello')
```

```python
def my_funtion(data):
    print "%s World" % data

my_funtion('Hello')
```

```
C:\Temp\test.py
  File "C:\Temp\test.py", line 1
    def my_funtion(data);
                        ^
SyntaxError: invalid syntax
```

```
C:\Temp\test.py
Hello World
```

Debugging

alignment issues

```
1  def my_funtion(data):
2     print "%s World" % data
3    print data
4
5  my_funtion("Hello")
```

```
C:\Temp\test.py
  File "C:\Temp\test.py", line 3
    print data
             ^
IndentationError: unindent does not match any
                  outer indentation level
```

more than one solution exists

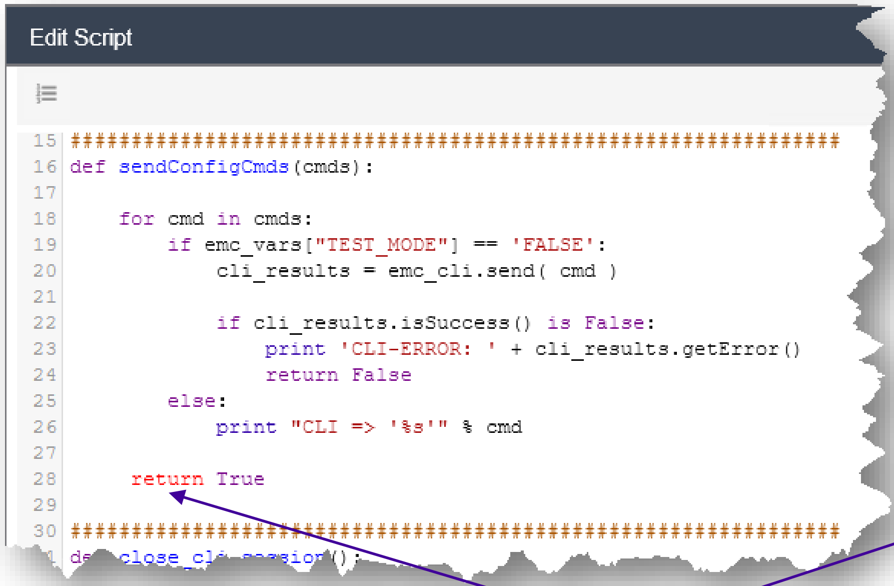
```
1  def my_funtion(data):
2     print "%s World" % data
3     print data
4
5  my_funtion('Hello')
```

```
C:\Temp\test.py
Hello World
Hello
```

Please never use TABs, just space
most of the editors and IDEs support a TAB to 4 space conversion

Debugging

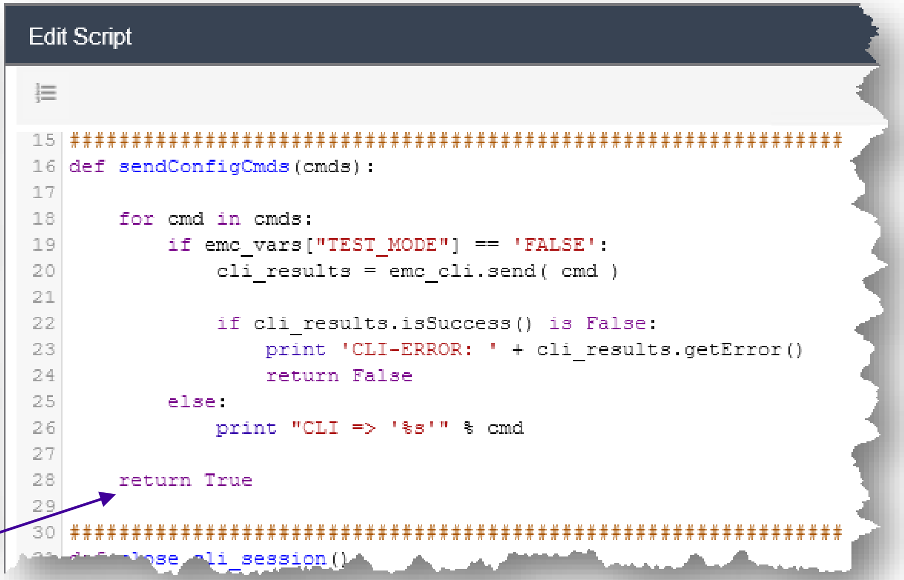
alignment issues



Edit Script

```
###############################################################
def sendConfigCmds(cmds):

    for cmd in cmds:
        if emc_vars["TEST_MODE"] == 'FALSE':
            cli_results = emc_cli.send( cmd )

            if cli_results.isSuccess() is False:
                print 'CLI-ERROR: ' + cli_results.getError()
                return False
        else:
            print "CLI => '%s'" % cmd

     return True

###############################################################
def close_cli_session():
```

alignment okay



Edit Script

```
###############################################################
def sendConfigCmds(cmds):

    for cmd in cmds:
        if emc_vars["TEST_MODE"] == 'FALSE':
            cli_results = emc_cli.send( cmd )

            if cli_results.isSuccess() is False:
                print 'CLI-ERROR: ' + cli_results.getError()
                return False
        else:
            print "CLI => '%s'" % cmd

    return True

###############################################################
def close_cli_session()
```

XMC WEB-UI editor give you a red indication

Debugging

execution issues

```
1  def my_funtion(data):
2      print "%s World" % data
3      result = data + 1
4
5  my_funtion('Hello')
```

```
1  def my_funtion(data):
2      print "%s World" % data
3      result = data + str( 1 )
4
5  my_funtion('Hello')
```

```
C:\Temp\test.py
Hello World
Traceback (most recent call last):
  File "C:\Temp\test.py", line 5, in <module>
    my_funtion('Hello')
  File "C:\Temp\test.py", line 3, in my_funtion
    result = data + 1
TypeError: cannot concatenate 'str' and 'int' objects
```

```
C:\Temp\test.py
Hello World
```

Debugging

scope issue

```python
myData = 'My text'

def my_funtion():
    print myData
    myData = 'Other text'
    print myData

my_funtion()
print myData
```

```
C:\Temp\test.py
My text
Other text
My text
```

```python
myData = 'My text'

def my_funtion():
    global myData
    print myData
    myData = 'Other text'
    print myData

my_funtion()
print myData
```

```
C:\Temp\test.py
My text
Other text
Other text
```

Debugging

measure execution time

```python
import time

################################
def my_funtion(data):
   print "%s World" % data
   number = 0
   while  number <= 2:
      number += 1
      time.sleep( 0.1 + number )

################################

startTime = time.time()
my_funtion("Hello")
endTime = time.time()

elapsedTime = float("%.3f" % ( endTime - startTime ) )
print "elapsed time: %s sec" % elapsedTime
```

```
C:\Temp\test.py
Hello World
elapsed time: 2.112 sec
```

Regular expressions (REGEX)

REGEX information

https://regexr.com/

https://regexone.com/

https://docs.python.org/2/howto/regex.html

https://www.tutorialspoint.com/python/python_reg_expressions.htm

Python REGEX search

```
re.search(pattern, string, flags=0)
```

```
import re
```

```
re.search(r'cookie', 'Cake and cookie').group()

pattern = re.compile(r"cookie")
sequence = "Cake and cookie“

re.search(pattern, sequence).group()

pattern.search(sequence).group()
```

```
re.match('test', 'TeSt', re.IGNORECASE)

re.search(r'(?i)test', 'TeSt').group()
```

Python REGEX **search** vs **match**

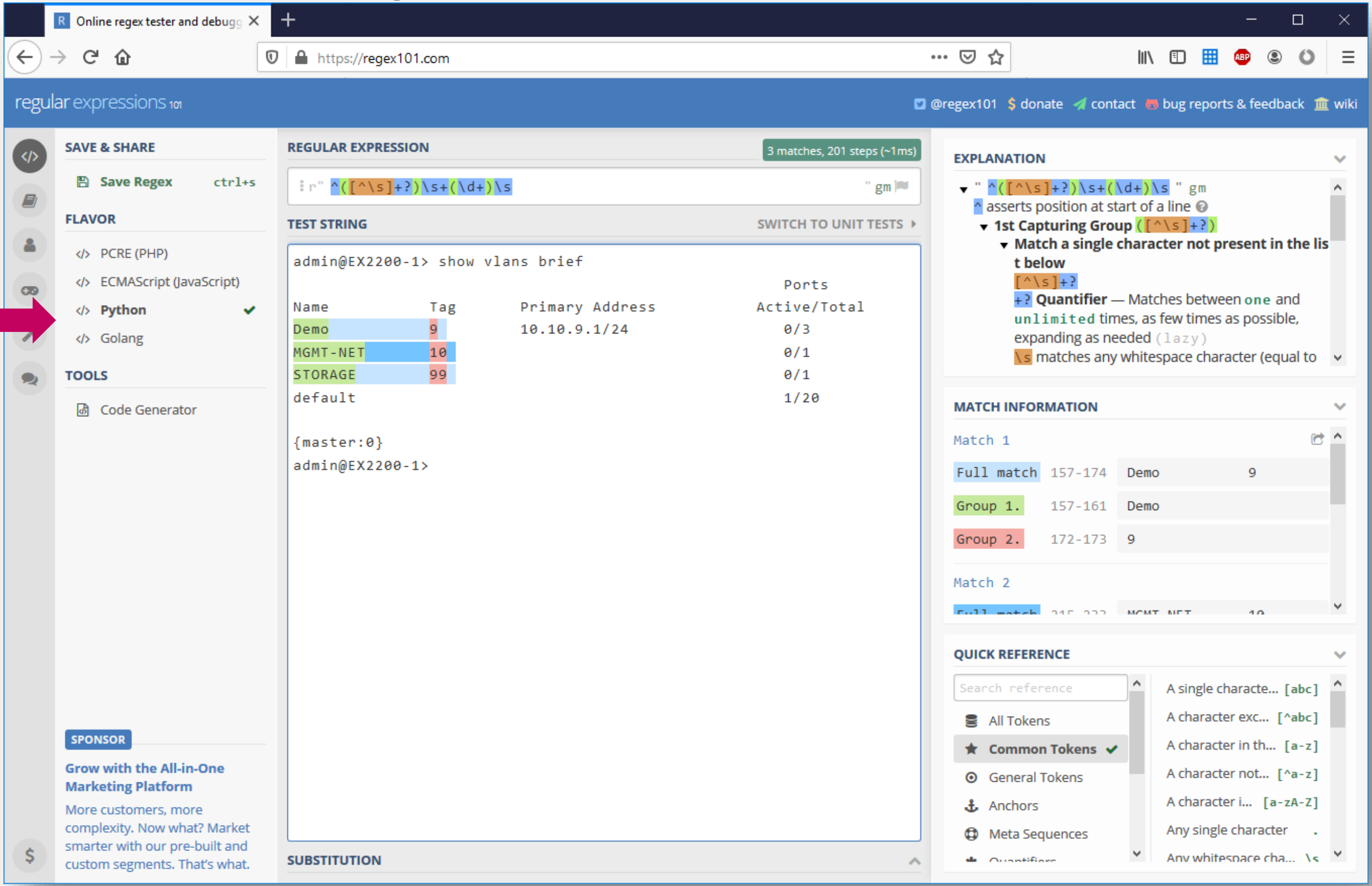
Note: Based on the regular expressions,
Python offers two different primitive operations.

The ***match*** method checks for a match only at the
beginning of the string.

while ***search*** checks for a match anywhere in the string.

test your REGEX upfront

https://regex101.com/



```
r" ^([^\s]+?)\s+(\d+)\s " gm
```

```
admin@EX2200-1> show vlans brief
                                                   Ports
Name            Tag        Primary Address        Active/Total
Demo            9          10.10.9.1/24              0/3
MGMT-NET        10                                   0/1
STORAGE         99                                   0/1
default                                              1/20

{master:0}
admin@EX2200-1>
```

" ^([^\s]+?)\s+(\d+)\s " gm

^ asserts position at start of a line

1st Capturing Group ([^\s]+?)

Match a single character not present in the list below [^\s]+?

+? Quantifier — Matches between one and unlimited times, as few times as possible, expanding as needed (lazy)

\s matches any whitespace character (equal to

Match 1

Full match	157-174	Demo 9
Group 1.	157-161	Demo
Group 2.	172-173	9

Match 2

A single characte...	[abc]
A character exc...	[^abc]
A character in th...	[a-z]
A character not...	[^a-z]
A character i...	[a-zA-Z]
Any single character	.

Python REGEX

catch an IP address

```
here is text 192.168.0.11 and other text
```

```
\d+.\d+.\d+.\d+
```

not good

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

better

```
(\d{1,3}\.){3}\.\d{1,3}
```

even better

```
(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])
```

just perfect, but

Python REGEX

not so greedy matching

configure policy profile 1 name "**Failsafe**" pvid-status "**enable**" pvid 4095

\"(.*)\" → Failsafe" pvid-status "enable

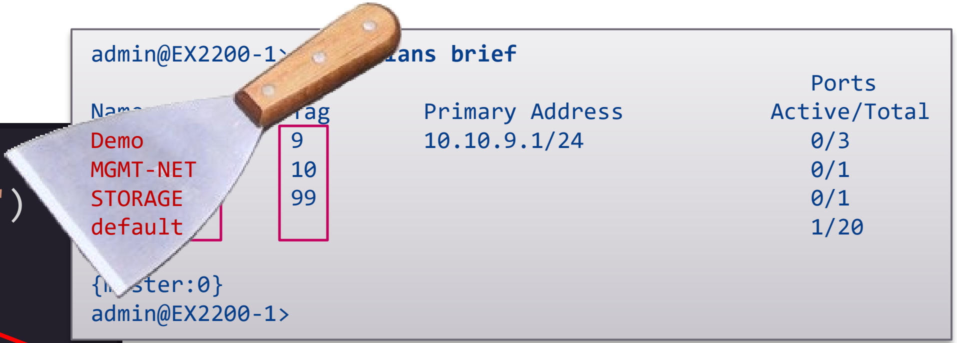
\"(.*?)\" → Failsafe
enable

\"([^\"]*?)\" → Failsafe
enable

Python REGEX

CLI scraping

```python
def getVlanList():
  regex = re.compile(r"^([^\s]+?)\s+(\d+)\s")
  vlans = {}
  cmds  = []

  cmds.append( 'show vlans brief' )

  cli_result = sendConfigCmds( cmds )

  for line in cli_result:
     result = regex.search( line )
     if result:
        vlanName = str( result.group(1) )
        vlanId   = int( result.group(2) )
        vlans[vlanName] = vlanId

  return vlans
```



```
admin@EX2200-1>             lans brief
                                                            Ports
Nam            ag        Primary Address             Active/Total
Demo           9         10.10.9.1/24                     0/3
MGMT-NET       10                                         0/1
STORAGE        99                                         0/1
default                                                   1/20

{   ster:0}
admin@EX2200-1>
```

Regular Expression (REGEX)

Anchor

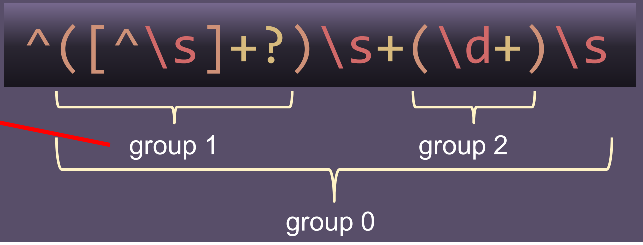
`^`	begin of the string

Quantifier

`+`	one or more
`+?`	one or more (not so greedy)

matching Operator

`\s`	space or tab
`\d`	digit (number)
`[^\s]`	negate (anything except space)



```
^([^\s]+?)\s+(\d+)\s
```

Next Presentation

Use the following link to advance to the next PDF in the Workflow education presentation.

WWW.EXTREMENETWORKS.COM