Networking Academy

Programmability Webinar Series with DevNet

Session 5: The New Toolbox of a Network Engineer

Speaker: Matt Denapoli

Hostess: Kara Sullivan

Jointly presented by DevNet & NetAcad

13 February, 2019

Welcome to the 5th session of the Programmability with Cisco DevNet webinar series

- Use the Q and A panel to ask questions.
- Use the Chat panel to communicate with attendees and panelists.
- A link to a recording of the session will be sent to all registered attendees.
- Please take the feedback survey at the end of the webinar.

The Webinar Series

Date	Topic
Oct'18	Networking with Programmability is Easy
Oct'18	A Network Engineer in the Programmable Age
Nov'18	Software Defined Networking and Controllers
Jan'19	Adding API Skills to Your Networking Toolbox
Feb'19	The New Toolbox of a Networking Engineer
Mar'19	Program Networking Devices using their APIs
Apr'19	Before, During, and After a Security Attack
May'19	Play with Linux & Python on Networking Devices
Jun'19	Automate your Network with a Bot

All Series Details can be Found @ http://bit.ly/devnet2

The Webinar Series – Raffle & Certificates

Raffle

- ✓ We will be raffling off a total of 15 Amazon gift cards in the amount of $25 US dollars at the end of this series.*
- ✓ 10 Amazon gift cards in the amount of $25 US dollars raffled off to everyone who participates in all of the live sessions
- ✓ 5 Amazon gift cards in the amount of $25 US dollars raffled off to everyone who participates in all of the sessions by either attending the live sessions or viewing/downloading the recording (can be a combination of the two in this raffle).

* Please note that this is a raffle and not everyone who qualifies will receive a gift card. There will be a total of 15 winners.

Certificate of Participation

LOLLY INTERIOR DESIGN SOLUTIONS
3465 Highway Street, Wellington, New Zealand
456 - 564 - 578 lollyinteriordesignsolutions@email.com

CERTIFICATE OF PARTICIPATION

This is to certify that ______________ participated at this year's event entitled ______________ on ____________ at ____________

Your hard work, dedication, and achievement will be cherished.

Certified by:

Chairman Coordinator Certified Instructor

- ✓ There will be an opportunity to sign up for a Certificate of Participation at the end of this series.
- ✓ To qualify, you must have participated in all sessions of the series.
- ✓ You can do this by attending the live sessions, viewing the recordings, or a combination of the two.
- ✓ Certificates will not be given out for individual sessions, but for the series as a whole.

Joining You Today:

Matt Denapoli
Developer Evangelist
Cisco DevNet

Session 5

The New Programmable Toolbox of a Network Engineer

Matthew
DeNapoli

@theDeNap

DevNet Developer Advocate

Stone Age

Spanning Tree

VLANs

Bronze Age

Routing Protocols

WAN Design

IP-magedon

The Renaissance

SDN

OpenFlow

Controllers

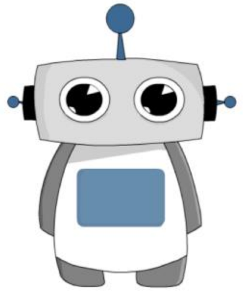
Overlays

MP-BGP

VXLAN

Micro-Segmentation

White Box



Programmable Age

Cloud

Python

REST / APIs

NETCONF / YANG

"Fabrics"

Network Function Virtualization (NFV)

Containers

DevOps

NetDevOps!

The Four Ages of Networking…..

Common Challenges

Difficult to Secure

Ever increasing number of users and endpoint types

Increase in complexity to increase scale

Difficult to Integrate and Manage

Multiple steps, user credentials, complex interactions

Multiple touch-points

Slower Issue Resolution

Separate user policies for wired and wireless networks

Unable to find users when troubleshooting

Traditional Networks Cannot Keep Up!

Network as a Platform Considerations
Where to Start?

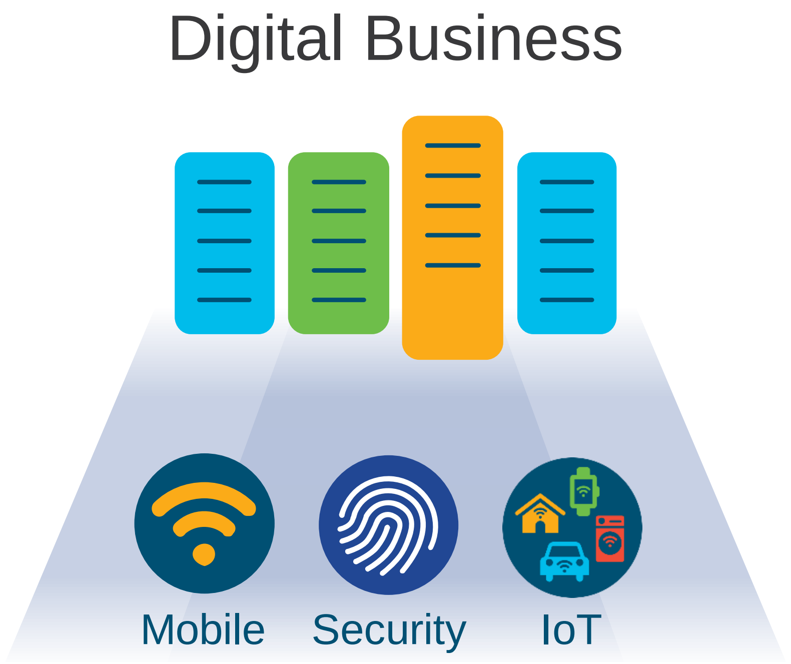
FASTER INNOVATION
Insights & Experiences

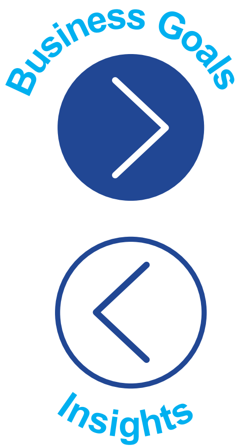
REDUCED COST & COMPLEXITY
Automation & Assurance

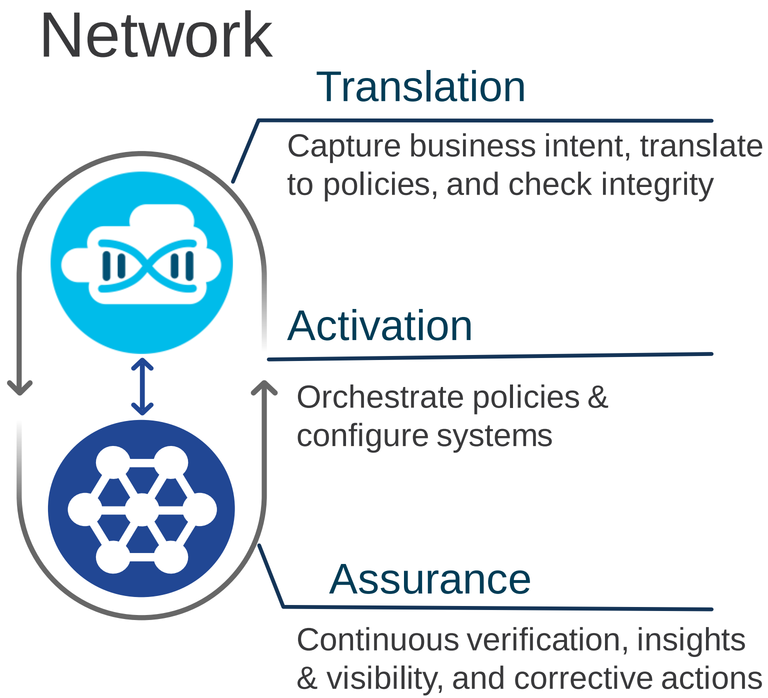
LOWER RISK
Security & Compliance

Analytics
Mobility
Automation
Security
Virtualization
Open APIs
Multi-Cloud
Policy
IoT
SDN

The Network Intuitive = Intent-based Networking







Translation

Capture business intent, translate to policies, and check integrity

Activation

Orchestrate policies & configure systems

Assurance

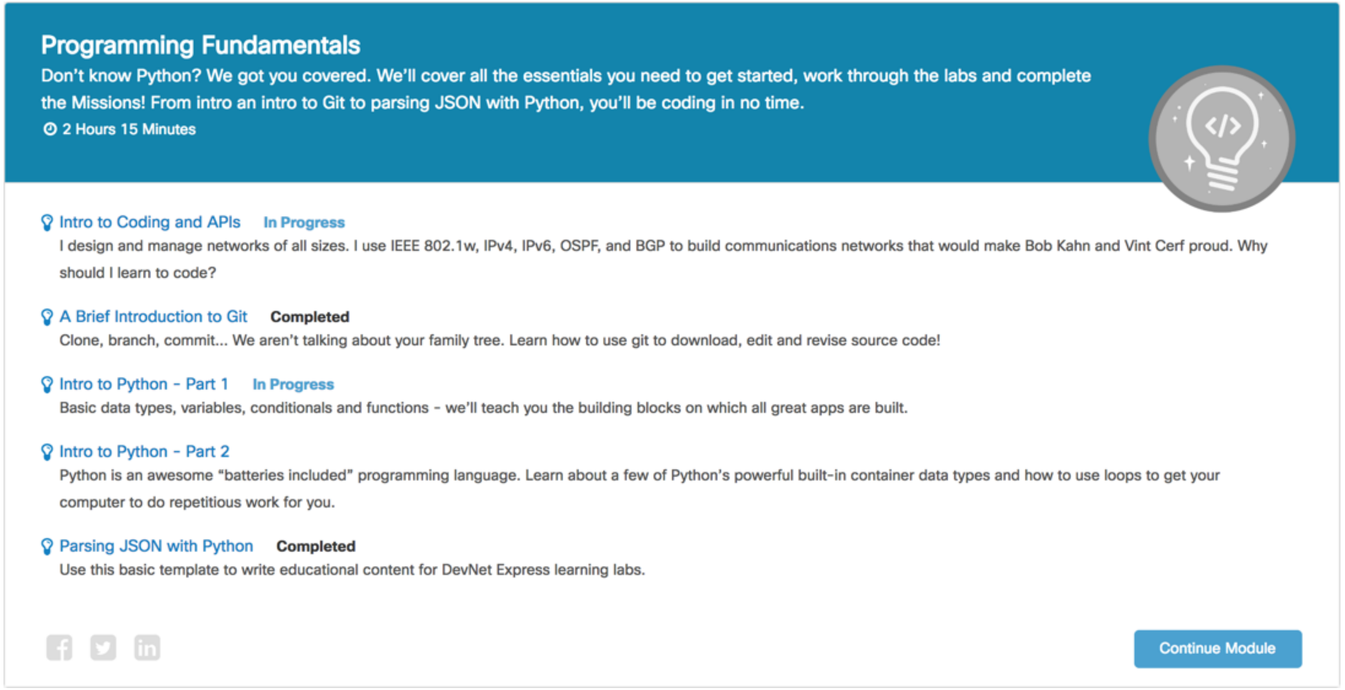
Continuous verification, insights & visibility, and corrective actions

Powered By Intent. Informed by Context.

Agenda

- Why Python?
- Using the Python Interpreter
- Basic Python Syntax
- Collections and Loops
- Script Structure and Execution

learninglabs.cisco.com/modules/intro-python



Programming Fundamentals

Don't know Python? We got you covered. We'll cover all the essentials you need to get started, work through the labs and complete the Missions! From intro an intro to Git to parsing JSON with Python, you'll be coding in no time.

2 Hours 15 Minutes

Intro to Coding and APIs **In Progress**
I design and manage networks of all sizes. I use IEEE 802.1w, IPv4, IPv6, OSPF, and BGP to build communications networks that would make Bob Kahn and Vint Cerf proud. Why should I learn to code?

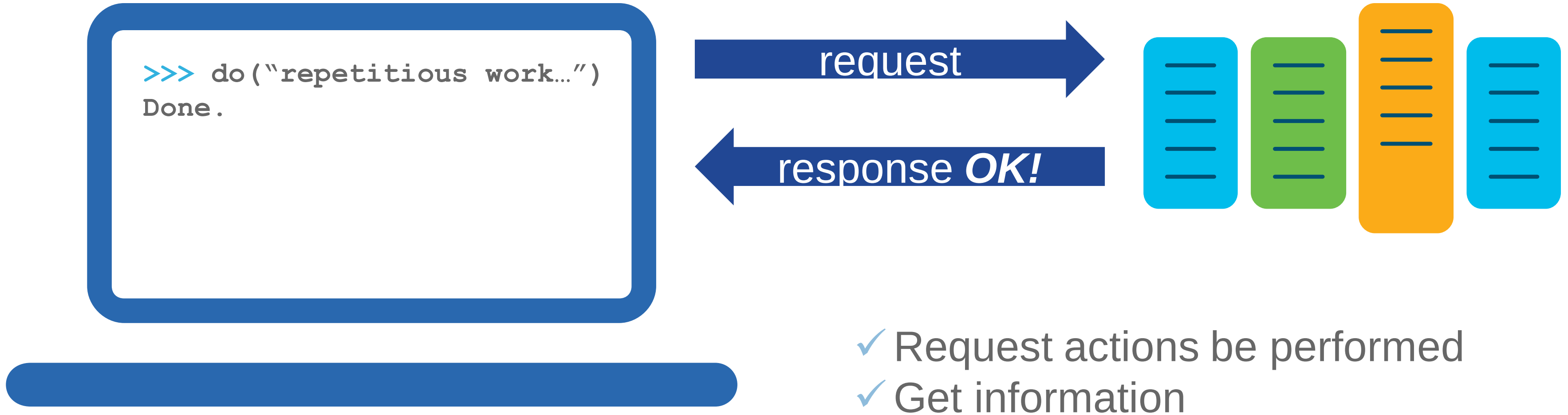
A Brief Introduction to Git **Completed**
Clone, branch, commit... We aren't talking about your family tree. Learn how to use git to download, edit and revise source code!

Intro to Python - Part 1 **In Progress**
Basic data types, variables, conditionals and functions - we'll teach you the building blocks on which all great apps are built.

Intro to Python - Part 2
Python is an awesome "batteries included" programming language. Learn about a few of Python's powerful built-in container data types and how to use loops to get your computer to do repetitious work for you.

Parsing JSON with Python **Completed**
Use this basic template to write educational content for DevNet Express learning labs.

The Value-Proposition for APIs



```
>>> do("repetitious work…")
Done.
```

✓ Request actions be performed
✓ Get information
✓ Store information

13

“It’s a way for two pieces of software to talk to each other”

Application Programming Interface (API)

The Value-Proposition for Programmability

Coding is the process of writing down instructions, in a language a computer can understand, to complete a specific task.

Q: What task?
A: Your task.

```
for switch in my_network:
    for interface in switch:
        if interface.is_down() and interface.last_change() > thirty_days:
            interface.shutdown()
            interface.set_description("Interface disabled per Policy")
```

What Changed?

API & Language Maturity

- ✓ RESTful APIs
- ✓ Expressive Modern Languages

Online Communities

- ✓ Open Source
- ✓ Social Code Sharing (GitHub)
- ✓ Public Package Repositories

```
$ pip install requests
Collecting requests
  Using cached
<-- output omitted for brevity -->
$ python
>>> import requests
>>> requests.get("https://api.github.com")
<Response [200]>
```

You can get powerful things done with relatively small amounts of code!

Why Python?

- **Domain Applicability**
  Established online DevOps Community
- **Power and Flexibility**
  Create & Work With: Shell Scripts, Back-end Web APIs, Databases, Machine Learning, …
- **Platform Flexibility**
  Run Your Code: Laptop, Server, VM, Container, Cloud, Cisco IOS Device
- **We Like It!**
  We have: Laptop Stickers, T-Shirts, Social Profiles, and Emotional Connections to Our Code

Python Scripts

- ✓ Text Files (UTF-8)
- ✓ May contain Unicode
  Some editors / terminals don’t support Unicode
- ✓ Use any Text Editor
  Using a Python-aware editor will make your life better
- ✓ No Need to Compile Them

18

Using a Python Interpreter

Know Thy Interpreter

What interpreter are you using?

- ❑ `python`
- ❑ `python2`
- ❑ `python3`
- ❑ `python3.5`
- ❑ `python3.6`
- ❑ *`other`*

What version is it?

```
$ python -V
```

Where is it?

```
$ where command
```

What is a Virtual Environment?

- Directory Structure
- Usually associated with a Project
- An *isolated* environment for installing and working with **Python Packages**

```
$ python3 -m venv venv
$
$ tree -L 1 venv/
venv/
├── bin
├── include
├── lib
 └── pyvenv.cfg
$
$ source venv/bin/activate
(venv) $
```

Remember

Activating a Python Virtual Environment

source *environment-name*/**bin/activate**

- ✓ The activation script will modify your prompt.
- ✓ Inside a virtual environment your interpreter will always be `` `python` ``.

```
$ source venv/bin/activate
(venv) $
(venv) $
(venv) $ deactivate
$
```

PIP Installs Packages

- Included with Python v3+
  Coupled with a Python installation; may be called `pip3` outside a venv
- Uses the open PyPI Repository
  Python Package Index
- Installs packages and their dependencies
- You can post your packages to PyPI!

```
(venv) $ pip install requests
Collecting requests
 Downloading
<-- output omitted for brevity -->
Installing collected packages: idna,
certifi, chardet, urllib3, requests
Successfully installed certifi-
2018.4.16 chardet-3.0.4 idna-2.6
requests-2.18.4 urllib3-1.22
(venv) $
```

Using your Python Interpreter

How to…	Command
Access the Python Interactive Shell	`$ python`
Running a Python script	`$ python` *`script.py`*
Running a script in 'Interactive' mode Execute the script and then remain in the Interactive Shell	`$ python -i` *`script.py`*

Python’s Interactive Shell

Accepts all valid Python statements

Use It To:

- ✓ Play with Python syntax
- ✓ Incrementally write Code
- ✓ Play with APIs and Data

To Exit:

`Ctrl + D` or `exit()`

```
(venv) $ python
Python 3.6.5 (default, Apr 2 2018, 15:31:03)[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on
linuxType "help", "copyright", "credits" or "license" for more information.
>>>
```

Basic Python Syntax

Basic Data Types

Python type()	Values (examples)
`int`	`-128, 0, 42`
`float`	`-1.12, 0, 3.14159`
`bool`	`True, False`
`str`	`“Hello 😎”` Can use ‘ ’, “ ”, and “”” ”””
`bytes`	`b”Hello \xf0\x9f\x98\x8e”`

```
>>> type(3)
<class ‘int’>

>>> type(1.4)
<class ‘float’>

>>> type(True)
<class ’bool’>

>>> type("Hello")
<class ’str’>

>>> type(b"Hello")
<class ‘bytes’>
```

Cisco live!

Numerical Operators

Math Operations

Addition:	+
Subtraction:	-
Multiplication:	*
Division:	/
Floor Division:	//
Modulo:	%
Power:	**

```
>>> 5 + 2
7
>>> 9 * 12
108
>>> 13 / 4
3.25
>>> 13 // 4
3
>>> 13 % 4
1
>>> 2 ** 10
1024
```

Variables

Names

- Cannot start with a number [0-9]
- Cannot conflict with a language keyword
- Can contain: [A-Za-z0-9_-]
- Recommendations for naming (variables, classes, functions, etc.) can be found in PEP8

Created with the = assignment operator

Can see list of variables in the current scope with `dir()`

```
>>> b = 7
>>> c = 3
>>> a = b + c
>>> a
10

>>> string_one = "Foo"
>>> string_two = "Bar"
>>> new_string = string_one + string_two
>>> new_string
'FooBar'
```

In Python, Everything is an Object!

Use `.` *(dot)* syntax to access *"things"* inside an object.

Terminology

When contained inside an object, we call…

Variable → Attribute

Function → Method

Check an object's type with `type(object)`

Look inside an object with `dir(object)`

```
>>> a = 57
>>> a.bit_length()
6
>>> "WhO wRoTe THIs?".lower()
'who wrote this?'
```

Working with Strings

String Operations

Concatenation: **+**

Multiplication: *****

Some Useful String Methods

Composition:	`"{}".format()`
Splitting:	`"".split()`
Joining:	`"".join()`

```
>>> "One" + "Two"
'OneTwo'

>>> "Abc" * 3
'AbcAbcAbc'

>>> "Hi, my name is {}!".format("Chris")
'Hi, my name is Chris!'

>>> "a b c".split(" ")
['a’, 'b’, 'c']

>>> ",".join(['a’, 'b’, 'c'])
'a,b,c'
```

Basic I/O

Get Input with `input()`

- Pass it a prompt string
- It will return the user’s input as a string
- You can convert the returned string to the data type you need int(), float(), etc.

Display Output with `print()`

- Can pass multiple values
- It will concatenate those values with separators in between (default = spaces)
- It will add (by default) a newline (‘\n’) to the end

```
>>> print(‘a’, ‘b’, ‘c’)
a b c

>>> i = input(“Enter a Number: ”)
Enter a Number: 1
>>> int(i)
1
```

Conditionals

Syntax:

```
if expression1:
    statements…
elif expression2:
    statements…
else:
    statements…
```

- ✓ Indentation is important!
- ✓ 4 spaces indent recommended
- ✓ You can nest if statements

Comparison Operators:

Less than	`<`
Greater than	`>`
Less than or equal to	`<=`
Greater than or equal to	`>=`
Equal	`==`
Not Equal	`!=`
Contains element	`in`

Combine expressions with: **`and`**, **`or`**

Negate with: **`not`**

Conditionals | Examples

```
>>> b = 5
>>> if b < 0:
...     print("b is less than zero")
... elif b == 0:
...     print("b is exactly zero")
... elif b > 0:
...     print("b is greater than zero")
... else:
...     print("b is something else")
...
b is greater than zero
```

```
>>> words = "Foo Bar"
>>> if "Bar" in words:
...     print("words contains 'Bar'")
... elif "Foo” in words:
...     print("words contains 'Foo'")
...
words contains 'Bar'
```

Cisco live!

Functions | Don’t Repeat Yourself

Modularize your code

- Defining your own Functions
- (optionally) Receive arguments
- (optionally) Return a value

Syntax:

```
def function_name(arg_names):
        statements…
        return value
...
function_name(arg_values)
```

```
>>> def add(num1, num2):
...     result = num1 + num2
...     return result
...
>>>
>>> add(3, 5)
8


>>> def say_hello():
...     print("Hello!")
>>>
>>> say_hello()
Hello!
```

Data Structures / Collection Data Types

Name `type()`	Notes	Example
`list`	• Ordered list of items • Items can be different data types • Can contain duplicate items • Mutable (can be changed after created)	`[‘a’, 1, 18.2]`
`tuple`	• Just like a list; except: • Immutable (cannot be changed)	`(‘a’, 1, 18.2)`

dictionary
`dict`
	• Unordered key-value pairs
	• Keys are unique; must be immutable
	• Keys don’t have to be the same data type
	• Values may be any data type
		`{“apples”: 5,`
		` “pears”: 2,`
		` “oranges”: 9}`

Working with Collections

Name `type()`	Creating	Accessing Indexing	Updating

`list`	`l = [‘a’, 1, 18.2]`	`>>> l[2]`
		`18.2`
			`>>> l[2] = 20.4`
			`>>> l`
			`[‘a’, 1, 20.4]`

`tuple`	`t = (‘a’, 1, 18.2)`	`>>> t[0]`
		`‘a’`
			You cannot update tuples after they have been created.

`dict`	`d = {“apples”: 5,`
	`“pears”: 2,`
	`“oranges”: 9}`
		`>>> d[“pears”]`
		`2`
			`>>> d[“pears”] = 6`
			`>>> d`
			`{“apples”: 5, “pears”: 6, “oranges”: 9}`

Dictionary Methods

Some useful dictionary methods:

```
{}.items()
{}.keys()
{}.values()
```

There are many more! 😎

```
>>> d = {"a": 1, "b": 2, "c": 3}

>>> d.items()
dict_items([('a’,1), ('b’,2), ('c',3)])

>>> d.keys()
dict_keys(['a’, 'b’, 'c’])

>>> d.values()
dict_values([1, 2, 3])
```

Loops

Iterative Loops

for *individual_item* **in** *iterator*:

    statements…

```
>>> names = ["chris", "iftach", "jay"]
>>> for name in names:
...     print(name)
...
chris
iftach
jay
```

Conditional Loops

while *logical_expression*:

    statements…

```
>>> i = 0
>>> while True:
...     print(i)
...     i += 1
...
0
1
2
3
4
```

Unpacking

Q: What if you wanted to break out a collection to separate variables?

A: *Unpack them!*

```
>>> a, b, c = [1, 2, 3]
>>> a
1
>>> b
2
>>> c
3
```

Iterating through a Dictionary

- Use the dictionary `.items()` method, which returns a "list of tuples"
- **Unpack** each tuple into variable names of your choosing to use within your block of statements

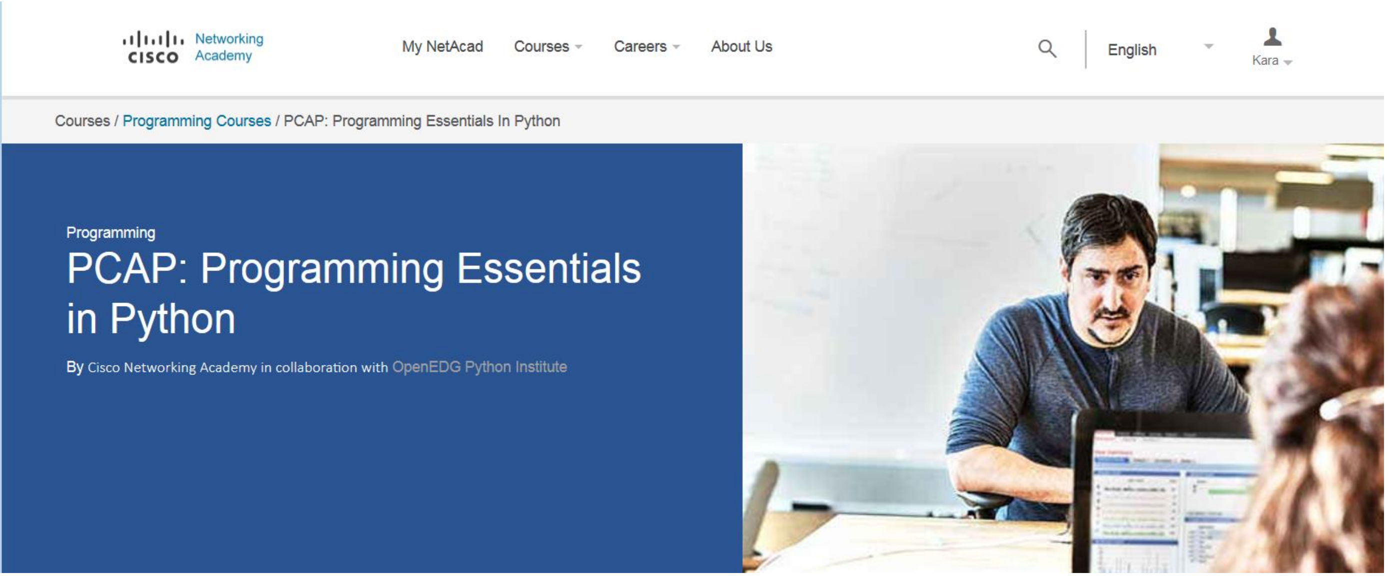
Method returns dictionary items as a list of **(key, value) tuples**, which the **for** loop will iteratively **unpack** into your variable names.

```
>>> for fruit, quantity in fruit.items():
...     print("You have {} {}.".format(quantity, fruit))
...
You have 5 apples.
You have 2 pears.
You have 9 oranges.
```

Go Forth and CODE!

Questions?

Want to Learn More About Python?



Courses / Programming Courses / PCAP: Programming Essentials In Python

Programming

PCAP: Programming Essentials in Python

By Cisco Networking Academy in collaboration with OpenEDG Python Institute

- Free online self-paced course
- 70 Hours
- Level: Intermediate
- No prior knowledge of programming is required

Enroll at: http://bit.ly/pythonessentialscourse

Next DevNet Webinar: 20 March 2019

Date	Topic
Oct'18	Networking with Programmability is Easy
Oct'18	A Network Engineer in the Programmable Age
Nov'18	Software Defined Networking and Controllers
Jan'19	Adding API Skills to Your Networking Toolbox
Feb'19	The New Toolbox of a Networking Engineer
Mar'19	Program Networking Devices using their APIs
Apr'19	Before, During, and After a Security Attack
May'19	Play with Linux & Python on Networking Devices
Jun'19	Automate your Network with a Bot

All Series Details can be Found @ http://bit.ly/devnet2