Linux Academy

Study Guide

Learning Python Development

Contents

Installing Python

Installation on CentOS:

```
sudo yum install epel-release
sudo yum install python python-pip
```

Starting Python in interactive mode:

```
python
```

Zen of Python:

```
import this
```

First "Hello World":

```
print "Hello World"
```

Python Variable Basics

Variable assignment:

```
my_variable = "Hello World"
```

Numeric variable:

```
number_of_items = 1247
```

String variable:

```
number_of_items = "1247"
```

Force a string variable to be used as a integer:

```
int(number_of_items)
```

Force a string variable to be used as a float:

```
float(number_of_items)
```

Determine the type of a variable:

```
type(number_of_items)
```

Basic example:

```
number_of_items = 1247

total_price = 31

price_of_one_item = total_price / number_of_items

print price_of_one_item
```

Floor division: Returns the value as an integer and throws away the decimals.

```
841.1//4
```

Modulus: Returns just the remainder.

```
841 % 4
```

Exponent:

```
2**4
```

Number precision, using round:

```
price_of_one_item = round(total_price / number_of_items, 4)
```

Comparisons:

```
# greater than
1 > 2

# less than
600<5
```

```
# less than or equal to
600≤5

# equal to
2==2

# complex
1 ≤ 2 ≤ 4
```

Minimum number from a list of numbers:

```
min(1,5,2,5,300,.03)
```

Maximum number from a list of numbers:

```
max(1,5,2,5,300,.03)
```

Strings

Can be quoted in single, double or even triple quotes. The triple quotes allow for multi-line strings.

```
server_type = 'centos'
server_name = "alpha_01"
server_desc = """ My test server
    at Linux Academy
"""
```

Variables can be printed as part of another string using the %'s placeholder and passing in the variables:

```
print "Server name: %s" % (server_name)
```

Length of a string variable:

```
len(server_desc)
```

Concatenate strings together:

```
server_desc = "Server " + server_name + " is running " + server_type
server_desc = " ".join(("Server", server_name, "is running", server_
type))
```

Searching a string (is case-sensitive):

```
"centos" in server_desc
"CentOS" in server_desc
```

Location of a substring:

```
server_desc.find("centos")
```

Splicing or creating a substring:

```
# characters 7 to 15
server_desc[7:15]

# all characters up to the 6th one
server_desc[0:6]
server_desc[:6]

# last 6 characters
server_desc[-6:]

# from the 7th to the end
server_desc[7:]
```

Search and replace:

```
server_desc = server_desc.replace('centos','CentOS')
```

Removing characters:

```
# remove all white spaces around a string plus any carriage return
my_str.strip()

# remove all white spaces from the left of the string
my_str.lstrip()

# remove white spaces and carriage from the right of a string
my_str.rstrip()

# remove the characters 'abc' from the end of a my_str
my_str.rstrip('abc')
```

Standard Modules

Online documentation:

- https://docs.python.org/2/library/
- https://docs.python.org/2/search.html

Import a module for use:

```
import subprocess
```

Functions available in a module:

```
dir(subprocess)
```

Using a module:

```
import subprocess
host_name = subprocess.check_output("hostname")
host_name
```

Installing new modules:

```
pip install boto3
```

List of useful modules:

```
collections
datetime
glob
logging
os
pexpect
re
requests
string
subprocess
sys
timeit
urllib
```

Public Python module repository:

- https://pypi.python.org/pypi

Conditionals

Example IF statement:

```
# this module has a function called linux distro which returns the linux
distro the server is running
import platform
platform.linux_distribution()
sys_info = ' '.join(platform.linux_distribution())
sys_info
if "CentOS" in sys_info:
    print "CentOS"
elif "Ubuntu" in sys_info:
    print "Debian OS"
else:
    print "Unknown OS"
```

Another example:

```
""" Generate a random number between 0 and 100 and grade it
    A if > 90, B ≥ 80-89 """

# module for generating random number
import random

# and generate a random integer between 1 and 100
test_score = random.randint(0,100)

# evaluate using if
if test_score ≥ 90:
    print "A"
elif test_score ≥ 80:
    print "B"
elif test_score ≥ 70:
    print "C"
elif test_score ≥ 60:
    print "D"
else:
    print "F"
```

Loops

FOR loop using `range()`:

```
""" Loop to print the odd numbers between  1 and 10 """

for number in range(1, 11):
    if number % 2 > 0:
        print number
```

Break out of a loop using `break`:

```
for number in range(1, 10):
    if number == 5:
        print "I have counted to %s" % number
        break
```

ELSE statement with a FOR loop:

```
# ELSE statement will execute
for number in range(1, 10):
    if number == 5:
        print "I have counted to %s" % number
else:
    # will print out since we did not exit the loop
    print "I counted from 1 to 10 "
# ELSE statement will not execute
for number in range(1, 10):
    if number == 5:
        print "I have counted to %s" % number
        break
else:
    # will not print out since we did not exit the loop
    print "I counted from 1 to 10 "
```

Loops to iterate over strings:

```
notes = "And a 1 and a 2 and a 3"
for x in notes:
    if x.isdigit():
        # only print char if numeric
        print x
```

WHILE loops:

```
""" Coin toss using while loop and count the number of tosses before we
get 10 heads in a row """

import random

heads_in_a_row_needed = 10
heads_in_a_row = 0
total_tries = 0
while heads_in_a_row_needed != heads_in_a_row:
    # mimic a coin toss by generating 0 and 1 randomly
    toss = random.randint(0,1)
    if toss == 1:
        heads_in_a_row +=1
    else:
        heads_in_a_row = 0
    total_tries +=1
print "It took %s tries to get %s heads in a row" % (total_tries, heads_
in_a_row)
```

WHILE loops with a break:

```
heads_in_a_row_needed = 10
heads_in_a_row = 0
total_tries = 0
while True:
    toss = random.randint(0,1)
    if toss == 1:
        heads_in_a_row +=1
    else:
        heads_in_a_row = 0
    total_tries +=1
    if heads_in_a_row_needed == heads_in_a_row:
        break

print "It took %s tries to get %s heads in a row" % (total_tries, heads_
in_a_row)
```

Lists

Creating lists:

```
list1 = [1,2,3,4]
list2 = ['a','b','c']
list3 = ['1','62',"q","I am also a member"]

# other variables in a list
str1 = "string var"
list4 = ["4", str1, list3]

people = ["jet", "donny", "jerome", "paul"]
```

Accessing list item (positions are zero-relative):

```
# Single item
people[2]

# First 2 item
people[:2]
```

Appending to a list:

```
people.append("george")
```

Position of an item in a list:

```
people.index("jerome")
people.index("mark")
```

Length:

```
len(people)
```

Removing items:

```
people.remove("donny")
```

Example script:

```
""" Monitors the partitions on a linux server
    and notifies if the partition usage is beyond the set threshold """

import subprocess

# threshold at which we should raise the alarm
partition_usage_threshold = 5

# run the linux command using the subprocess module
df_cmd = subprocess.check_output(['df','-k'])
print df_cmd

# split the output into a list called lines
lines = df_cmd.splitlines()
print lines

# for each line except the first once, since that has the column names
for line in lines[1:]:
    # split the line into columns
    columns = line.split()
    # get the used percentage val from column 4
    used_percentage = columns[4]
    # remove the % sign
    used_percentage = used_percentage.replace('%','')
    # check for threshold breach
    if int(used_percentage) ≥ partition_usage_threshold:
        print "partition %s usage is beyond threshold at %s " % 
(columns[0], columns[4])
        # you can have an email function here that alerts you about this
```

Dictionaries

Creating a dictionary:

```
# empty dictionary
traffic_signal = {}
```

Adding elements:

```
# one at a time
traffic_signal{'red'}='stop'
traffic_signal{'yellow'}='about to be red'
traffic_signal{'green']='go'

# multiple keys at a time
traffic_signal = {'red' : 'stop', 'yellow' : 'about to be red',
'green':'go'}
```

Retrieving elements:

```
# if you know the key exists
traffic_signal{'red'}

# if unsure if key exists
traffic_signal.get('xyw')

# if unsure if key exists with return value if not found
traffic_signal.get('xyw', 'that key is not in there')
```

Example script 1:

```
""" script that does 1000 coin tosses and counts how many were heads and
tails """
import random
# dict to keep track of heads and tails
results = {'heads':0, 'tails':0}
for i in range(0,1000)
    toss = random.randint(0,2)
    if toss == 1:
        results{'heads'}+=1
    else:
        results{'tails'}+=1

for toss in results.keys():
    print "Coinface %s showed up %s times " % (toss, results{'toss'})
```

Example script 2:

```
""" Count the number of processes being run by each user on CentOS """

import subprocess
users = {}
ps_cmd = subprocess.check_output(['ps','-ef'])
for line in ps_cmd.splitlines()[1:]:
    user = line.split()[0]
    if users.get(user):
        users[user]+=1
    else:
        user[user]=1

print "Active users on the system are " + ','.join(users.keys())
for user, process_count in users.items():
    print "%s is running %s processes" % (user, process_count)

print users
del users['root']
print users
```

Example 3:

```
""" Count the number of processes being run by each user on CentOS """

import subprocess
users = {}
ps_cmd = subprocess.check_output(['ps','-ef'])
for line in ps_cmd.splitlines()[1:]:
    user = line.split()[0]
    users[user]=users.get(user,0)+1
```

Tuples and Sets

A tuple consists of a number of values separated by commas. They are useful for ordered pairs and returning several values from a function.

They cannot be modified like lists thus they are much faster than a list to iterate over.

Tuples

Creating a tuple:

```
constant_vals = ()
constant_vals = ("Jack","Arizona",9,"Atlanta",9,14,6)

# a tuple can contain other data structes
```

```
tuples = ( [1,2,3], "a", 9 )

# when crating a tuple with one value only you have to use a comma at
the end
constant_vals = (39,)
```

Length of a tuple:

```
len(constant_vals)
```

Counting the occurrence of a string/number within a tuple:

```
constant_vals.count(9)
```

Locating a string/number within a tuple:

```
constant_vals.index("Atlanta")
"Texas" in constant_vals
```

Iterating:

```
# and you can iterate over tuples just like lists
for val in constant_vals:
    print val
```

Sets

One of the most useful data structures when dealing with data analysis. A set can extract the unique values from a list, which then can be used for unions, intersections, etc.

Create a set:

```
linux_essentials =
set(["John","Kevin","Anthony","James","Sara","Marge","John"])
lpic_level1 = set(["Kevin","James","Marge","Lewis","Nancy"])
```

Set intersections:

```
# people who watch both the linux_essentials_certification and lpic
videos
both_courses = linux_essentials & lpic_level1
print both_courses
```

Set subtraction:

```
# peole who should watch the linux essensial video but skipped it
need_to_watch_le = lpic_level1 - linux_essentials
print(need_to_watch_le)

# peole who should watch the lpic video
need_to_watch_lpic =  linux_essentials - lpic_level1
print(need_to_watch_lpic)
```

Functions

Create a function:

```
def functionname(variables_passed_in):
    # do something here
    return some_value
```

Example script 1:

```
def activeProcesses(lookup_user):
    """ Look up how many processes a user is running """
    processes_running = 0
    for line in subprocess.check_output("ps -ef", shell=True).
splitlines()[1:]:
        user = line.split()[0]
        if lookup_user == user:
            processes_running+=1
    return "User %s has %s processes running" % (lookup_user, processes_
running)
print activeProcesses('root')
print activeProcesses('postfix')

def activeProcesses(lookup_user, lookup_cmd):
    """ Look up how many command of a particular type a user is running
"""
    processes_running_all = 0
    processes_running_searched = 0
    for line in subprocess.check_output("ps -ef", shell=True).
splitlines()[1:]:
        user = line.split()[0]
        if lookup_user == user:
            processes_running_all+=1
            if lookup_cmd in line:
                processes_running_searched+=1
    return processes_running_all, processes_running_searched
procs_total, procs_searched  = activeProcesses('root', 'aws')
print procs_total, procs_searched
```

Exceptions

Catch bad code:

```
# division by zero
try:
    print 1/0
except ZeroDivisionError:
    print "Cannot divide by a zero"
else:
    print "All good"

# bad command
try:
    import subprocess
    subprocess.check_output(['k'])
except Exception as ex:
    print "A %s exception happened because  %s" % (type(ex).__name__, 
ex.args)
else:
    print "all good"
```

Catch importing a module that is not installed:

```
try:
    import some_module
except Exception as ex:
    print "A %s exception happened because  %s" % (type(ex).__name__, 
ex.args)
else:
    print "all good"
```

File Processing

Open a file and read the lines one at a time:

```
filename = '/var/log/secure'
# one line at a time
for line in open(filename):
    print line
```

Slurp the file whole:

```
with open(filename) as file_handle:
    lines = file_handle.readlines()
    for line in lines:
        print(line)
```

Write to a file:

```
filename = 'textfile.txt'
with open(filename, 'w') as file_handle:
    file_handle.write("here is some text\n")
```

Append to a file:

```
with open(filename, 'a') as file_handle:
    file_handle.write("here is more text")
```

Reading a CSV:

```
import csv
file_handle = open('servers.csv')
reader = csv.reader(file_handle)
os_counts = {}
for row in reader:
    os_counts[row[2]] = os_counts.get(row[2],0)+1

print os_counts
```

Open a file safely:

```
try:
   filename = '/var/log/notthere'
   for line in open(filename):
       print line
       # do any processing here
except IOError:
    print "File does not exist"
except:
    print "Can't open file for other reason"
else:
    print "Done processing file"
```

Class

Create a class:

```
class Car():
    def __init__(self):
        self.color = ''
        print "car started"
    def accel(self,speed):
        print "speeding up to %s mph" % speed
    def turn(self, direction):
        print "turning " + direction
    def stop(self):
        print "stop"
```

Inherit from another class:

```
    class RaceCar(Car):
        def __init__(self, color):
            self.color = color
            self.top_speed = 200
            print "%s race car started with a top speed of %s" % (self.
    color, self.top_speed)
        def accel(self, speed):
            print "speeding up to %s mph very very fast" % speed
```

Instantiate:

```
    car1 = Car()
    car2 = RaceCar('blue')
```

Change variables within a class:

```
    car1.color='red'
    car2.color='red'
```

Call functions within a class:

```
    car1.accel(10)
    car1.turn('right')
    car1.stop()
    car2.accel(10)
    car2.turn('left')
    car2.stop()
```

Available functions in a class:

```
    vars(car1)
    vars(car2)
```

Decorators

Think of decorators like wrappers. They essentially allow one function to be passed into another function.

```
""" Decorater elapsed_time to time how long it takes to download a
webpage"""

# module for timing
import time

# module to open web pages
import urllib2

def elapsed_time(function_to_time):
    def wrapper():
        t0 = time.time()
        function_to_time()
        t1 = time.time()
        print "Elapsed time: %s\n" % (t1 - t0)
    return wrapper

@elapsed_time
def download_webpage():
    url = 'http://linuxacademy-static-blogpost.s3-website-us-east-1.
amazonaws.com/'
    response = urllib2.urlopen(url, timeout = 60)
    return response.read()

webpage = download_webpage()

@elapsed_time
def another_function():
    print "Doing something else"
    for i in range(1,1000000):
        pass
```

Generators

Generators iterate without creating all values at ones:

```
def counter():
    i=0
    while True:
        i+=1
        return i
a = counter()
print a
type(a)

### Instead of returning a value, you generate a series of values (using
the yield statement)
def counter():
    i=0
    while True:
        i+=1
        yield i
```

```
a = counter()
type(a)
print next(a)
print next(a)
print next(a)
```

Regular Expressions (regex)

Cheat sheet available in the course downloads.

Example script 1:

```
line = "Oct  7 17:28:59 shirazk2141 sshd[2877]: Failed password for root
from 31.220.3.180 port 50388 ssh2"

import re
match = re.search('sshd', line)
print match

match = re.search('hello', line)
print match

#Bad way to do it
match = re.search('[A-Z][a-z]{2}\s{1,2}\d{1,2}\s\d{2}:\d{2}:\d{2}\
s\w*\ssshd\[\d*\]: Failed password for \w+ from \d{1,3}\.\d{1,3}\.\
d{1,3}\.\d{1,3} port \d* ssh2', line)
print match

#A little simpler
match = re.search('^(.*?)\s(\w+)\ssshd.*?Failed\spassw.*?from\s(.*?)\
sport.*$', line)
print match
print match.groups()

#With named groups
match = re.search('^(?P<date>.*?)\s(\w+)\ssshd.*?Failed\
spassw.*?from\s(.*?)\sport', line)
print match.group('date')
```

Example script 2:

```
""" Script to parse the '/var/log/secure' to find all the hacking
attempts and the originating IP addresses using  regex """

import re
filename = '/var/log/secure'
ips = []
for line in open(filename):
    # Failed password attempts
    match = re.search('^(.*?)\s(\w+)\ssshd.*?Failed\
spassw.*?from\s(.*?)\sport.*$', line)
    if match:
        if match.group()

    # Bad user attempts
    match = re.search('^(.*?)\s\w+\sssh.*?Invalid\suser\s(\w+)\
sfrom\s(.*)', line);
```

JSON

```
# module for json parsing
import json

# module for opening webpages
import urllib

# open a url
url = "https://labfiles.linuxacademy.com/python/ec2-response.json"
response = urllib.urlopen(url)
# get the json data into json_string
json_string = response.read()
print json_string

# safe JSON parsing using exception catching
data = None
try:
    # parse the JSON
    data = json.loads(str(json_string))
except:
    data = None

# if valid JSON
if ( data ):
    # access the data just like a dictionary/list
    print "InstanceID %s is %s" % (data['InstanceStatuses'][0]
['InstanceId'],  data['InstanceStatuses'][0]['InstanceState']['Name'])

# create a JSON string using a dictionary variable
data = {
    'course_name' : 'python',
    'videos' : ['strings','classes','json'],
    'id' : 5
}

# pass the variable to the JSON module for serializing.
# the indent=4 gets it nicely formatted for viewing
json_string = json.dumps(data, indent=4)
print json_string
json_string = json.dumps(data)
print json_string
```

List Comprehension

```
# create a list var with values 1 thru 49
numbers = range(0,50)
print numbers

# list of even numbers
even_numbers = [i for i in range(0,50) if i % 2 == 0 and i >=2 ]
print even_numbers

even_numbers = [i for i in numbers if i % 2 == 0 and i >=2 ]
print even_numbers

# squared numbers
squares = [i*i for i in numbers if i % 2 == 0 and i >=2]
print squares

# generate a list of random 50 numbers
import random
random_numbers = [ random.randint(1,100) for x in range(50) ]
print random_numbers

# let's get the unique numbers from random_numbers
unique_random_numbers = list(set(random_numbers))
print unique_random_numbers

# list comprehension using strings
names = ['adam', 'Justin', 'joe', 'tony', 'zoe']
names_formatted = [ x.title() for x in names]
print names_formatted

#using map
names_formatted = map(lambda x: x.title(), names)
print names_formatted

#names that start with a j, case-insensitive
names_starting_with_j = [name for name in names_formatted if name[0].
lower() == 'j' ]
print names_starting_with_j
```

Example script 1:

```
""" Create a random encoding system that only encodes letters and
nothing else using random letters """

# module for string functions
import string

# module to randomize lists
from random import shuffle
raw_text = "Hello and welcome to Linux Academy!!"
```

```
# get letters a to z in both upper and lower case
letters = list(string.ascii_letters)
print letters

# create a copy of the letters list
encoded_letters = letters[:]

# randomize it
shuffle(encoded_letters)
print encoded_letters

#initialize the keys
encoding_key = {}
decoding_key = {}
# zip will iterate over two lists at the smae time
for k, v in zip(letters, encoded_letters):
    encoding_key[k]=v
    decoding_key[v]=k

print encoding_key

encoded_text = ''
# for each letter in the raw_text
for letter in raw_text:
    # lookup the encoding and append it to the encoded text
    encoded_text += encoding_key.get(letter, letter)
print encoded_text

# a better way to do the encoding and decoding
encoding_key = dict(zip(letters,encoded_letters))
decoding_key = dict(zip(encoding_key.values(),encoding_key.keys()))
encoded_text = ''.join([ encoding_key.get(w, w) for w in raw_text])
print encoded_text

decoded_text = ''.join([ decoding_key.get(w, w) for w in encoded_text])
print decoded_text
```

System Automation with Fabric

Source file with comments available at:

- http://labfiles.linuxacademy.com/python/fabric/scripts/fabfile.py

Web Scraping

Source file with comments available at:

- http://labfiles.linuxacademy.com/python/scraping/scripts/webscraping.py