Python Files

Christoffer Berglund

September 21, 2018

Contents

1 Overview

- Repetition
- Basic file reading
- Basic file writing
- Options for open
- Context manager
- Working with files
- Pickle
- Error handling

2 Repetition

2.1 lambda

- One-line no-name function

Python lambda:

```
f = lambda x: x>2
f(6)
```

Normal function equivalent:

```
def f(x):
    return x>2

f(10)
```

2.2 Lambda for something useful

```
l = [-1,-0,1,2,3]
print(list(filter(lambda x:x>0,l)))
```

2.3 5 Minute Assignment

- Make a sorting function that sorts based on the length of strings.
- Hint: Use len() to check the length of a string.

2.4 Solution

```
l = ["hei","hallo","n"]
l.sort(key=lambda x: len(x))
print(l)
```

2.5 Map-function

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))
print(squared)
```

2.6 Reduce

```
from functools import reduce
product = reduce( (lambda x, y: x * y), [1, 2, 3, 4] )
print(product)
```

2.7 Pythonic iterations

```
[i for i in range(10)]

[x**2 for x in range(10)]

[i for i in range(10) if i%2==0]
```

2.8 Pythonic iterations with string

```
friends = ['Luke','Leia','Han','Chewbacca']
[i for i in friends if(i.startswith('L'))]
```

2.9 5 Minute Assignment

- Given the following list of IPs use python iteration to find the IPs ending with 9

```
ips = ['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.2']
```

2.10 Solution

```
ips = ['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.1', '192.168.1.9', '192
[i for i in ips if i.endswith('.9')]
```

2.11 Iteration over a list

```
import random

random_num = random.randint(64,128)
character = chr(random_num)
character = chr(random.randint(64,128))
print(character)

[chr(random.randint(64,128)) for i in range(10)]
```

2.12 Join

- Join a list into a String

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

2.13 Join with random letters

```
''.join([chr(random.randint(64,128)) for i in range(10)])
```

2.14 5 Minute Assignment

- With the following list of letters iterate through it, select those that are uppercase, and join it.
- Hint: Use isupper to find the uppercase letters.

```
l = ['EX','Ex','eqwwe','P','ER','sdaAA','TSs','TS']
```

2.15 Solution

```
l = ['EX','Ex','eqwwe','P','ER','sdaAA','TSs','TS']
''.join([i for i in l if i.isupper()])
```

2.16 Pythonic dictionary generation

```
{hash(i):i for i in ['en','to','tre']}
```

2.17 Making and adding

```
a = set()
a = set([1,2,3])
a = {1,2,3}
a.add(4)
a.add(3)
print(a)
```

2.18 Subset

```
a = set([2,3,4,5])
b = set([1,2,3,4,5,6])
print(a.issubset(b))
```

2.19 Union

```
a = set([0,1,2,3])
b = set([1,2,3,4,5,6])
print('union:', a.union(b))
print('b diff a:', b.difference(a))
print('a diff b:', a.difference(b))
print('a intersection b:', a.intersection(b))
```

2.20 Discard and remove

```
s = set([1,2,3])
s.discard(2)
s.discard(2)
print(s)
```

```
s = set([1,2,3])
s.remove(2)
s.remove(2)
print(s)
```

2.21 Changing a list to a set

```
l = [1,2,2,2,2,3,3,3,3,3]
print(type(l))
s = set(l)
print(type(s))
```

2.22 5 Minute Assignment

- Following is IPs that could have been collected from /var/log/messages.
- These IPs represent login attempts from other IPs.
- Print out the unique IPs, and how many login attempts have come from each IP.

```
['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.1', '192.168.1.9']
```

2.23 Solution

```
ips = ['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.1', '192.168.1.9', '192
for i in set(ips):
   print(i,ips.count(i))
```

2.24 5 Minute Assignment

- Following is a list of **bad** IPs (which e.g. have malware) and IPs in your network.
- Use set, union,intersection and/or difference to quickly find out which IPs in your network are bad.

```
bad = ['192.168.1.1','192,168.1.5']
allips = ['192.168.1.1','192,168.1.2','192,168.1.5','192,168.1.9']
```

2.25 Solution

```
bad = ['192.168.1.1','192,168.1.5']
allips = ['192.168.1.1','192,168.1.2','192,168.1.5','192,168.1.9']
b = set(bad)
l = set(allips)
print(b.intersection(l))
```

3 Basic file reading

```
f = open('somefile.txt', 'r')
f.read()  # read's the hole file
f.close()  # close the file
```

```
f = open('somefile.txt', 'r')
f.readline()  # read's the first line
f.close()  # close the file

f = open('somefile.txt', 'r')
f.readlines()  # read's all lines inn as a list
f.close()  # close the file
```

4 Basic file writing

```
f = open('anotherfile.txt', 'w')
f.write('Hello from lecture!')  # write a string to the file
f.close()  # close the file

f = open('anotherfile.txt', 'r')
f.read()  # read's the hole file
f.close()  # close the file

lst = ['Hello from lecture!', 'We are scripting!']
f = open('anotherfile.txt', 'w')
f.writelines(lst)  # write a list of strings
f.close()  # close the file

f = open('anotherfile.txt', 'r')
f.read()  # read's the hole file
f.close()  # close the file
```

5 Options for open

Character	Meaning
r	read
w	write
x	create
a	append
b	binary
t	text mode
+	update, read and write
U	universal newline mode

6 Context manager

- Garantees the the file is exits in the correct way
- No need for file.close()

```
with open('somefile.txt', 'r') as f:
    f.read()
```

7 Working with files

7.1 Directory manipulation

- Getting the working directory

```
import os
os.getcwd()
```

- list files and dir's

```
path = os.getcwd()
os.listdir(path)
```

- change directory

```
#os.chdir("/home/christoffer/Code/Local/Scripting_and_hacking/")
os.getcwd()
os.chdir("/home/christoffer/Code/Local/Scripting_and_hacking/Presentation/pythonFiles/"
```

7.2 Make into a set

- strip() for removing white-space
- count() for counting each unique letter

```
f = open('livehack.txt', 'r')
data = f.read()
f.close()
s = set(data)
for i in s:
    print(i.strip(),':',str(data.count(i)))
```

7.3 Print first IP

- pythonic way to validate
- index() +4 -> each letter has a index

```
f = open('livehack.txt','r')
l = f.readlines() # As list of lines
f.close()
validlines = [i for i in l if('from' in i and 'port' in i)]
firstline = validlines[0]
print(firstline)
start = firstline.index('from')+4
end = firstline.index('port')
print(end)
ip = firstline[start:end].strip()
print(ip)
```

7.4 Print all IPs

```
allips = set()
for l in validlines:
    start = l.index('from')+4
    end = l.index('port')
    ip = l[start:end].strip()
    allips.add(ip)
print(allips)
```

7.5 All IPs and number of attacks from IPs

```
allips = []
for l in validlines:
    start = l.index('from')+4
    end = l.index('port')
    ip = l[start:end].strip()
    allips.append(ip)
s = set(allips)
for ip in s:
    print(ip,':',allips.count(ip))
```

7.6 5 Minute Assignment

- Print all ports
- Hint: Between the text port and ssh2.

7.7 Solution

```
f = open('livehack.txt','r')
data = f.readlines()
f.close()
validlines = [i for i in data if('port' in i and 'ssh2' in i)]
allports = set()
for l in validlines:
    start = l.index('port')+4
    end = l.index('ssh2')
    port = l[start:end].strip()
    allports.add(port)
print(allports)
```

7.8 5 Minute Assignment

- Build a hash table for usernames in the file.
- Note, rainbow tables use passwords, not usernames.
- Hint, to get hash of something:
  - `hash("noe")`

7.9 Solution

```
f = open('livehack.txt','r')
data = f.readlines()
validlines = [i for i in data if('port' in i and 'ssh2' in i)]
rainbowtable = {}
for l in validlines:
    start = l.index('for')+3
    end = l.index('from')
    username = l[start:end].strip()
    rainbowtable[hash(username)] = username
print(rainbowtable)
```

7.10 Write to a file

```
f = open('data','w')
s = set(allips)
for ip in s:
    f.write(ip + ': ' + str(allips.count(ip)) + '\n')
f.close()

f = open('data', 'r')
f.read()  # read's the hole file
f.close()  # close the file
```

7.11 Appending to a file

```
f = open('data','a')
f.write('New line at the bottom of the file')
f.close()

f = open('data', 'r')
f.read()  # read's the hole file
f.close()  # close the file
```

7.12 Write +

```
f = open('data','w+') # Read and write. Overwrite the file if it exists.
f.close()
```

8 Pickle

8.1 Write a serialized file.

```
import pickle
s = {'hello': 'world'}
f = open('picklefile.pickle','wb')
pickle.dump(s,f)
f.close()
```

8.2 Reading from a file

- trying to open it the normal way

```
f = open('picklefile.pickle', 'r')
f.read()  # read's the hole file
f.close()  # close the file
```

- need to use pickle to make sense of it

```
f = open('picklefile.pickle','rb')
x = pickle.load(f)
print(x)
```

8.3 Maybe reading from a file

```
import pickle
try:
   f = open('picklefie.pickle','rb')
   x = pickle.load(f)
except:
   x = set()
print(x)
```

8.4 File parameters

```
import sys

try:
    filename = sys.argv[sys.argv.index('-f')+1]
except ValueError:
    print('Syntax error')
    sys.exit(1)
print(filename)
```

9 Error handling

9.1 Dealing with error's

- When something can go wrong, we should do something
- If an exception is raised, your script will crash
  - Prevent crash = catch the exception
  - Exceptions will back-trace

```
class A:
    def b(self):
        if 1 < 2:
            if 2 > 1:
                f = open('bla', 'r')

a = A()
a.b()
```

- Catching it when opening the file

```
class A:
    def b(self):
        if 1 < 2:
            if 2 > 1:
                try:
                    f = open('bla', 'r')
                except IOError:
                    print("File does not exists")

a = A()
a.b()
```

- Catching it when trying to run the function

```
class A:
    def b(self):
        if 1 < 2:
            if 2 > 1:
                f = open('bla', 'r')

a = A()
try:
    a.b()
except IOError:
    print("File does not exists")
```

- Or around a if statement

```
class A:
    def b(self):
```

```
        if 1 < 2:
            try:
                if 2 > 1:
                    f = open('bla', 'r')
            except FileNotFoundError:
                print("File does not exists")

a = A()
a.b()
```

9.2 LBYL vs EAFP

- Look Before You Leap vs Easier to Ask Forgiveness than Permission
- Two methods of catching error
- When to use them? **It depends**
  - Sometimes you want errors to show, sometimes you want them not to show

9.2.1 LBYL

- Checking before using doing the division
- Can be useful in cases like this
- Low overhead

```
def divide(a, b):
    if a == 0 or b == 0:
        print("You can't divide with 0")
        return
    print(a/b)

divide(10, 2)
```

9.2.2 EAFP

- If you expect something could go wrong
- More overhead

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        b = 1
        print(a/b)
        print("You can't divide with 0")

divide(10, 0)
```

9.3 More on try except

- To catch every exception, **bad practice**
- Be as specific as possible

```
def divide(a, b):
    try:
        print(a/b)
    except Exeptions as e:
        print("You can't divide with 0")

devide(10, 0)
```

- This is better:

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("You can't divide with 0")

devide(10, 0)
```

- What will happen here?

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("You can't divide with 0")

divide(10, "5")
```

- Can catch more than one thing

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("You can't divide with 0")
    except TypeError:
        print("wrong type")
    except:
        print("all other")
    # or
    # except ZeroDivisionError, TypeError
divide(5, "1")
```

9.3.1 Else

- Will only run when no errors
- Can be used for return statements

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("You can't divide with 0")
    else:
        print("Will run if no errors occurred")

divide(10, 1)
```

9.3.2 Finally

- Will run no matter what
- Perfect for cleanup/closing files

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("You can't divide with 0")
```

```
    else:
        print("Will run if no errors occurred")
    finally:
        print("Will run with or without errors")
    print("hello")


divide(10, 0)
```

9.4 Assert and rise

- Assert can be used for testing
- Only run/compiled when debug mode is on(default)
- Runs when evaluated to false

```
def ultimate_question(string):
    assert(string == 42), "Wrong awnser"

print(ultimate_question("code"))
```

- Better examples

```
print("Bool check 1:", (1 + 1 == 2))
print("Bool check 2:", (1 + 1 != 2))
assert(1 + 1 == 2), "First check fail"
assert(1 + 1 != 2), "Second check fail"
```

- Raising exceptions

```
def ultimate_question(string):
    if string != "42":
        raise(ValueError, "You have not enterd the correct value")

print(ultimate_question("code"))
```

9.5 Bonus

- You can create your own exceptions

```
class MovieReferenceError(Exception):
    def __init__(self, message="Go watch: The hitchhiker's guide to the galaxy"):
        super(MovieReferenceError, self).__init__(message)

def ultimate_question(string):
    if string != "42":
        raise(MovieReferenceError)

print(ultimate_question("code"))
```