Python Data-structures

Christoffer Berglund

September 14, 2018

Overview

- Repetition
- Lists
- Tuples
- Dictionaries
- Hashing
- Sets
- Pythonic
- Python as a functional language

Repetition

Class

```
class BankAccount:
    balance = 0
    def insert(self,money):
        self.balance = self.balance+money
```

Usage

```
minkonto = BankAccount()
minkonto.balance
minkonto.insert(10)
minkonto.balance
```

Constructor

In python: `__init__`

- Called once when creating an instance
- Populating the instance

Inheritance

```
class PingOne(Ping):
    def __init__(self,ip):
        Ping.__init__(self,ip)
    def runpingone(self):
        import os
        print(self.ip)
        os.system('ping -c 1 '+self.ip)
```

Hand-Raising Assignment

- What is the output?

```
class Base:
   def op1(self): self.op2()
   def op2(self): print ("Base")
class Sub(Base):
   def op2(self): print ("Sub")
   def op3(self): print ("Sub")
obj = Sub()
obj.op3()
```

sub

Hand-Raising Assignment

- What is the output?

```
class Base:
   def op1(self): self.op2()
   def op2(self): print ("Base")
class Sub(Base):
   def op2(self): print ("Sub")
```

```
   def op3(self): print ("Sub")
obj = Sub()
obj.op2()
```

sub

Hand-Raising Assignment

- What is the output?

```
class Base:
   def op1(self): self.op2()
   def op2(self): print ("Base")
class Sub(Base):
   def op2(self): print ("Sub")
   def op3(self): print ("Sub")
obj = Base()
obj.op3()
```

error

Lists

Creating lists

```
l = [1,2,3,4]
print(l[2])
for i in l:
    print(i)
```

Appending list

```
l = [1,2,2,3]
l.append(3)
l.remove(2)
l.pop(0)
```

l.remove = removed the number 2, not index l.pop = removes index 0

List versus range

Range

```
l = range(10)
```

- Lazy.
  - Only instantiated when needed.
  - Less memory

List

```
l = list(range(10))
```

- Eager.
  - Instantiated from the very start.
  - When you need all numbers, you need an eager evaluation

Hand-raising assignment

- Which is faster and why?

Alternative 1:

```
for i in list(range(10000000)):
  if i>10:
    break
  print(i)
```

Alternative 2:

```
for i in range(10000000):
  if i>10:
    break
  print(i)
```

alt 2, cuz lazy

Hand-raising assignment

- Which is faster and why?

Alternative 1:

```
for i in range(100):
  if i>10:
    break
  print(i)
```

Alternative 2:

```
for i in range(10000000):
  if i>10:
    break
  print(i)
```

almost the same

5 Minute Assignment

- Given the range from 1-10, shuffle it and print out the first and second number.

Hint, use

`random.shuffle`

Solution

```
l = list(range(1,11))
random.shuffle(l)
print(l[0],l[1])
```

Finding in lists

```
friends = ["Luke","Leia","Han","Chewbacca"]
if "Luke" in friends:
    print("Lucky you")
```

Removing friends

```
friends = ["Luke","Leia","Han","Chewbacca"]
friends.pop(0)
friends.remove("Chewbacca")
```

Max, Min and Sum

```
l = [1,2,2,4,5,6,6]
print(max(l))
print(min(l))
print(sum(l))
```

5 Minute Assignment

- Make an avg-function that takes in a list, runs sum and length of the list

Solution

```
def avg(l):
   return float(sum(l))/len(l)
```

Randomizing list

```
import random
a = [1,2,3,4]
random.shuffle(a)
```

Shuffle and sort strings

```
a = ['abc','cba']
import random
random.shuffle(a)
a.sort()
```

Random choice

```
random.choice([1,2,3,4])
```

select one integer from a list

Copying list

```
l = [1,1,1,1,2,2,2,2,3,3,3,3]
l2 = l[:]
l2 = l[2:5]
```

gives you from index 2 until index 5, but not include

Counting in a list

```
l = [1,1,1,1,1,2,2,2,2,2,3,3,3,3]
l.count(1)
l.count(2)
```

5 Minute Assignment

- Following is IPs that could have been collected from /var/log/messages.
- Given the following list of IPs. Count how many times the IP 192.168.1.1 appears

```
l = ['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.1', '192.168.1.9']
```

Solution

```
l = ['192.168.1.1', '192.168.1.9', '192.168.1.1', '192.168.1.1', '192.168.1.9']
l.count('192.168.1.1')
```

Double list

```
l = [[1,"One"],[2,"Two"],[3,"Three"]]
l[0][0]
dict(l)
```

cast list to a dict

Sorting double list

```
l = [[1,"one"], [2, "two"], [3, "three"], [4, "four"]]

def f(lst):
    return lst[1]
l.sort(key=f)
print(l)
```

```
[[4, 'four'], [1, 'one'], [3, 'three'], [2, 'two']]
```

Tuples

Tuples

```
a = (1,2)
a[0]
a[1]
```

Tuples are immutable

```
a = (1,2)
a[0] = 2
```

Using tuples

```
import random
def getTwoPins():
   pin1 = random.randint(1000,9999)
   pin2 = random.randint(1000,9999)
   return pin1,pin2

a,b = getTwoPins()
print(a)
print(b)
```

a,b = getTwoPins() is unpacking them both

5 Minute Assignment

- Make a function that returns three random numbers less than 10.

Solution

```
def f():
   a = random.randint(0,10)
   b = random.randint(0,10)
   c = random.randint(0,10)
   return a,b,c
one,two,three = f()
```

Dictionary

Creating dictionaries from double lists

```
l = [[1,"One"],[2,"Two"],[3,"Three"]]
l = dict(l)
```

Creating dictionaries

```
rainbowtable = {-587279054: 'secret', 942035612: 'hemmelig', -1438253177: 'hemmelig123
```

Changing dictionary into list

```
a = {1:"one",2:"two",3:"three"}
a.keys()
a.values()
a.items() # list of touple pairs
```

Iterations

```
a = {1:"one",2:"two",3:"three"}
for key,value in a.items():
    print(key,value)

for key in a.keys():
    print(key)
```

Hashing

Hash-value

```
hash("noe")
```

Getting an md5 hash

```
import hashlib
m = hashlib.md5()
m.update(b'noe')
print(m.hexdigest())
```

or

```
import hashlib
hashlib.md5(b'noe').hexdigest()
```

Get password from hash

```
rainbowtable = {-587279054: 'secret', 942035612: 'hemmelig', -1438253177: 'hemmelig123
rainbowtable.get(942035612)
```

Add a new hash

```
# rainbowtable[key] = value
password = 'sf242q'
rainbowtable[hash(password)] = password
```

Removing a key

```
rainbowtable.pop(942035612)
rainbowtable.pop(942035612)
```

Inserting hash value into the dictionary

```
s = "passord"
rainbowtable[hash(s)] = s
```

5 Minute Assignment

- Build a hash table for password of size 1 containing only ascii lowercase characters.
- Hint: To loop through all ascii lowercase use the following code:

```
import string
for i in string.ascii_lowercase:
    ....
```

Solution

```
import string
hashtable = {}
for i in string.ascii_lowercase:
   hashtable[hash(i)] = i
```

Better solution

```
dict(map(lambda x:(hash(x),x),string.ascii_lowercase))
```

5 Minute Assignment

- In the following rainbow table, print only the password values

```
rainbowtable = {-587279054: 'secret', 942035612: 'hemmelig', -1438253177: 'hemmelig123
```

Solution

```
rainbowtable = {-587279054: 'secret', 942035612: 'hemmelig', -1438253177: 'hemmelig123
for i in rainbowtable.values():
    print(i)
```

Sets

Making and adding

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

Subset

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

Union

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

union = combine
    difference = print 4,5,6
    difference = print 0
    intersection = numbers that are in both

Discard and remove

```
s = set([1,2,3])
s.discard(2) #Removes without error
s.remove(2) #Removes with error
```

discard does not produce error, even if 2 is not in set
    remove will keyerror if 2 is not in set

Changing a list to a set

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

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']
```

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))
```

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']
```

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)
b.intersects(l)
```

Pythonic

Pythonic iterations

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

Pythonic iterations with string

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

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']
```

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')]
```

Iteration over a list

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

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

Join

- Join a list into a String

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

Join with random letters

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

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']
```

Solution

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

Pythonic dictionary generation

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

Python as a functional language

Bubble sort http://www.algolist.net/Algorithms/Sorting/Bubble_sort

Sorted list

```
a = [2,3,4,1]
a.sort()
a.sort(reverse=True)
```

Specialized sort

```
a = [(1, 2), (4, 1), (9, 10), (13, -3)]
a.sort(key=lambda x: x[1])
```

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.

Solution

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

Filter

```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
print(less_than_zero)
```

In simple words, the filter() method filters the given iterable with the help of a function that tests each element in the iterable to be true or not

lambda

- One-line no-name function

Python lambda:

```
f = lambda x:x>2
```

Normal function equivalent:

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

Lambda for something useful

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

5 Minute Assignment

- With the following list of passwords use filter to only show the short passwords: both lambda and non-lambda.
- Hint: Make a function that takes in string and returns a true if the string is larger than 5.
- Hint: Use len(s) to find the length of the string s

Solution

Normal solution:

```
l = ['123456','abc','abcdefg','098']
def f(x):
   if(len(x)>5): return True
   else: return False
filter(f,l)
```

Lambda-solution:

```
filter(lambda x:len(x)>5)
```

Map-function

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

function - map() passes each item of the iterable to this function.

Reduce

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