CPSC 1301, Computer Science I Lab Assignment

Learning Python, 5th Edition, Chapter14 Lab

Run the following commands in your Python interpreter. These are from *Learning Python, 5th Edition*, Chapter14. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named pythonlab-Chapter14_lastname.txt.

1 Chapter 14

```
>>> for x in [1, 2, 3, 4]: print(x ** 2, end=' ') # In 2.X: print x ** 2,
>>>
>>> for x in (1, 2, 3, 4): print(x ** 3, end=' ')
>>>
>>> for x in 'spam': print(x * 2, end=' ')
>>>
>>> print(open('script2.py').read())
>>> open('script2.py').read()
>>> f = open('script2.py') # Read a four-line script file in this directory
>>> f.readline() # readline loads one line on each call
>>> f.readline()
>>> f.readline()
>>> f.readline() # Last lines may have a \n or not
>>> f.readline() # Returns empty string at end-of-file
>>> f = open('script2.py') # __next__ loads one line on each call too
>>> f.__next__() # But raises an exception at end-of-file
>>> f.__next__() # Use f.next() in 2.X, or next(f) in 2.X or 3.X
>>> f.__next__()
>>> f.__next__()
>>> f.__next__()
>>> for line in open('script2.py'): # Use file iterators to read by lines
>>>     print(line.upper(), end='') # Calls __next__, catches StopIteration
>>>
>>> for line in open('script2.py').readlines():
>>>     print(line.upper(), end='')
>>>
>>> f = open('script2.py')
>>> while True:
>>>     line = f.readline()
>>>     if not line: break
>>>     print(line.upper(), end='')
>>>
>>>    same output...
>>> f = open('script2.py')
>>> f.__next__() # Call iteration method directly
>>> f.__next__()
>>> f = open('script2.py')
>>> next(f) # The next(f) built-in calls f.__next__() in 3.X
```

```
>>> next(f) # next(f) => [3.X: f.__next__()], [2.X: f.next()]
>>> L = [1, 2, 3]
>>> I = iter(L) # Obtain an iterator object from an iterable
>>> I.__next__() # Call iterator's next to advance to next item
>>> I.__next__() # Or use I.next() in 2.X, next(I) in either line
>>> I.__next__()
>>> I.__next__()
>>>    error text omitted...
>>> f = open('script2.py')
>>> iter(f) is f
>>> iter(f) is f.__iter__()
>>> f.__next__(
>>> f = open('script2.py')
>>> iter(f) is f
>>> iter(f) is f.__iter__()
>>> f.__next__()
>>> L = [1, 2, 3]
>>> iter(L) is L
>>> L.__next__()
>>> I = iter(L)
>>> I.__next__()
>>> next(I) # Same as I.__next__()
>>> L = [1, 2, 3]
>>>
>>> for X in L: # Automatic iteration
>>>     print(X ** 2, end=' ') # Obtains iter, calls __next__, catches exceptions
>>>
>>> I = iter(L) # Manual iteration: what for loops usually do
>>> while True:
>>>     try: # try statement catches exceptions
>>>     X = next(I) # Or call I.__next__ in 3.X
>>>     except StopIteration:
>>>     break
>>>     print(X ** 2, end=' ')
>>>
>>> D = {'a':1, 'b':2, 'c':3}
>>> for key in D.keys():
>>>     print(key, D[key])
>>>
>>> I = iter(D)
>>> next(I)
>>> next(I)
>>> next(I)
>>> next(I)
>>> for key in D:
>>>     print(key, D[key])
>>>
>>> import os
>>> P = os.popen('dir')
>>> P.__next__()
>>> P.__next__()
>>> next(P)
>>> import os
>>> P = os.popen('dir')
```

```
>>> P.__next__()
>>> P.__next__()
>>> next(P)
>>> P = os.popen('dir')
>>> I = iter(P)
>>> next(I)
>>> I.__next__()
>>> R = range(5)
>>> R # Ranges are iterables in 3.X
>>> I = iter(R) # Use iteration protocol to produce results
>>> next(I)
>>> next(I)
>>> list(range(5)) # Or use list to collect all results at once
>>> for key in D:
>>>     print(key, D[key])
>>>
>>> import os
>>> P = os.popen('dir')
>>> P.__next__()
>>> P.__next__()
>>> next(P)
>>> P = os.popen('dir')
>>> I = iter(P)
>>> next(I)
>>> I.__next__()
>>> R = range(5)
>>> R # Ranges are iterables in 3.X
>>> I = iter(R) # Use iteration protocol to produce results
>>> next(I)
>>> next(I)
>>> list(range(5)) # Or use list to collect all results at once
>>> E = enumerate('spam') # enumerate is an iterable too
>>> E
>>> I = iter(E)
>>> next(I) # Generate results with iteration protocol
>>> next(I) # Or use list to force generation to run
>>> list(enumerate('spam'))
>>> L = [1, 2, 3, 4, 5]
>>> for i in range(len(L)):
>>>     L[i] += 10
>>>
>>> L
>>> L = [x + 10 for x in L]
>>> L
>>> res = []
>>> for x in L:
>>>     res.append(x + 10)
>>>
>>> res
>>> f = open('script2.py')
>>> lines = f.readlines()
>>> lines
>>> lines = [line.rstrip() for line in lines]
>>> lines
```

```
>>> lines = [line.rstrip() for line in open('script2.py')]
>>> lines
>>> [line.upper() for line in open('script2.py')]
>>> [line.rstrip().upper() for line in open('script2.py')]
>>> [line.split() for line in open('script2.py')]
>>> [line.replace(' ', '!') for line in open('script2.py')]
>>> [('sys' in line, line[:5]) for line in open('script2.py')]
[(True, 'impor'), (True, 'print'), (False, 'x = 2'), (False, 'print')]>>> lines = [line.rstrip() for li
>>> lines
>>> res = []
>>> for line in open('script2.py'):
...     if line[0] == 'p':
...     res.append(line.rstrip())
...
>>> res
>>> [line.rstrip() for line in open('script2.py') if line.rstrip()[-1].isdigit()]
>>> fname = r'd:\books\5e\lp5e\draft1typos.txt'
>>> len(open(fname).readlines()) # All lines
>>> len([line for line in open(fname) if line.strip() != '']) # Nonblank lines
>>> [x + y for x in 'abc' for y in 'lmn']
>>> res = []
>>> for x in 'abc':
...     for y in 'lmn':
...     res.append(x + y)
...
>>> res
>>> uppers = [line.upper() for line in open('script2.py')]
>>> uppers
>>> map(str.upper, open('script2.py')) # map is itself an iterable in 3.X
>>> list(map(str.upper, open('script2.py')))
>>> sorted(open('script2.py'))
>>> list(zip(open('script2.py'), open('script2.py')))
>>> list(enumerate(open('script2.py')))
>>> list(filter(bool, open('script2.py'))) # nonempty=True
>>> import functools, operator
>>> functools.reduce(operator.add, open('script2.py'))
'import sys\nprint(sys.path)\nx = 2\nprint(x ** 32)\n'>>> list(open('script2.py'))
>>> tuple(open('script2.py'))
>>> '&&'.join(open('script2.py'))
>>> a, b, c, d = open('script2.py') # Sequence assignment
>>> a, d
>>> a, *b = open('script2.py') # 3.X extended form
>>> a, b
>>> 'y = 2\n' in open('script2.py') # Membership test
>>> 'x = 2\n' in open('script2.py')
>>> L = [11, 22, 33, 44] # Slice assignment
>>> L[1:3] = open('script2.py')
>>> L
>>> L = [11]
>>> L.extend(open('script2.py')) # list.extend method
>>> L
>>> L = [11]
>>> L.append(open('script2.py')) # list.append does not iterate
>>> L
```

```
>>> list(L[1])
>>> set(open('script2.py'))
>>> {line for line in open('script2.py')}
>>> {ix: line for ix, line in enumerate(open('script2.py'))}
>>> {line for line in open('script2.py') if line[0] == 'p'}
>>> {ix: line for (ix, line) in enumerate(open('script2.py')) if line[0] == 'p'}
>>> list(line.upper() for line in open('script2.py')) # See Chapter 20
>>> any(['spam', '', 'ni'])
>>> all(['spam', '', 'ni'])
>>> max([3, 2, 5, 1, 4])
>>> min([3, 2, 5, 1, 4])
>>> max(open('script2.py')) # Line with max/min string value
>>> min(open('script2.py'))
>>>
>>> f(1, 2, 3, 4)
>>> f(*[1, 2, 3, 4]) # Unpacks into arguments
>>>
>>> f(*open('script2.py')) # Iterates by lines too!
>>> X = (1, 2)
>>> Y = (3, 4)
>>>
>>> list(zip(X, Y)
>>>
>>> A, B = zip(*zip(X, Y)) # Unzip a zip!
>>> A
>>> B
```

2 Chapter 15

```
>>> len(dir(sys)) # Number names in sys
>>> len([x for x in dir(sys) if not x.startswith('__')]) # Non __X names only
>>> len([x for x in dir(sys) if not x[0] == '_']) # Non underscore names
>>> dir([])
>>> dir('')
>>> len(dir([])), len([x for x in dir([]) if not x.startswith('__')])
>>> len(dir('')), len([x for x in dir('') if not x.startswith('__')])
>>> [a for a in dir(list) if not a.startswith('__')]
>>> [a for a in dir(dict) if not a.startswith('__')]
>>> dir1(tuple)
>>> dir(str) == dir('') # Same result, type name or literal
>>> dir(list) == dir([])
>>> import sys
>>> print(sys.__doc__)
>>> print(sys.getrefcount.__doc__)
>>> print(int.__doc__)
>>> print(map.__doc__)
>>> import sys
>>> help(sys.getrefcount)
>>> help(sys)
>>> help(dict)
>>> help(str.replace)
>>> help(docstrings.square)
```

```
>>> help(docstrings.Employee)
```