COEN 140 Machine Learning and Data Mining

Lab Assignment #1: Python Practice

Guideline: Follow the format in LabAssignment1_report_sample.pdf to do the following exercises. Part I exercises only need results. Part II exercises need both results and comments. Convert the report to one pdf file and submit it to Camino.

Online tutorial: https://docs.python.org/3/

Part I

Exercise 1: Numbers

Run the following in python shell.

```
>>> a=123+222 # integer addition
>>> print(a)
>>> b = 1.5*4    # floating-point multiplication
>>> print(b)
>>> c=2**10      # 2 to the power 10
>>> print(c)
>>> import math
>>> print(math.pi)
>>> print(math.sqrt(36))
>>> import random
>>> a=random.random()
>>> print('a=', a)
>>> b = random.choice([1,2,3,4])
>>> print('b=', b)
```

Exercise 2: Strings

Run the following in python shell.

```
>>> S='Spam'   # make a 4-character string, and assign it to a name
```

```
>>> len(S)        # Length
>>> S[0]  # the 1st item in S, indexing by zero-based position
>>> S[1]  # the 2nd item from the left
>>> S[-1]       # the last item from the end in S
>>> S[-2] # the second-to-last item from the end
>>> S[len(S)-1]
>>> S[1:3] # Slice of S from offsets 1 through 2 (not 3)
>>> S = 'z' + S[1:]
>>> S
```

Exercise 3: Lists

Run the following in python shell.

```
>>> L=[123, 'spam', 1.23]  # A list of three different-type objects
>>> len(L) # number of items in the list
>>> L[0]
>>> L[:-1]   # Slicing a list returns a new list
>>> L+[4,5,6]  # contact/repeat make new lists too
>>> L*2 # repeat
>>> L  # we are not changing the original list
>>> M = ['bb', 'aa', 'cc']
>>> M.sort()
>>> M
>>> M.reverse()
>>> M
>>> M = [[1,2,3], [4,5,6], [7,8,9]]   # a list that contains three other lists, 3x3 matrix
>>> M[1]   # get row 2
>>> M[1][2]  # get row 2, then get item 3 within the row
>>> diag = [M[i][i] for i in [0, 1, 2]]
>>> diag
>>> doubles = [c * 2 for c in 'spam']
```

>>> doubles

>>> list(range(4)) *# 0..3 (list() required in 3.X)*

>>> list(range(−6, 7, 2)) *# −6 to +6 by 2 (need list() in 3.X)*

>>> [[x ** 2, x ** 3] for x in range(4)] # Multiple values, "if" filters

>>> [[x, x/2, x * 2] for x in range(−6, 7, 2) if x > 0]

Exercise 4: Linear Algebra

4.1. Write a NumPy program to compute the multiplication of the following two matrixes, using numpy.dot and numpy.matmul, and print the results.

p = [[1, 0], [0, 1], [1,1]]

q = [[1, 2], [3, 4]]

4.2 Write a NumPy program to find the L2-norm of vector v=numpy.arange(7) and matrix numpy.matrix('1, 2; 3, 4'). Print the results.

Note: the L2-norm of a vector $\mathbf{v} = [v_1, v_2, \ldots, v_N]$ is defined as:

$$\|\mathbf{v}\|_2 \triangleq \sqrt{v_1^2 + v_2^2 + \ldots + v_N^2}$$

4.3 Write a NumPy program to compute the inverse of matrix m = np.array([[1,2],[3,4]]).

Exercise 5: Tuples

Run the following in python shell.
>>> T = (1, 2, 3, 4) *# A 4-item tuple*

>>> len(T) *# Length*

>> T + (5, 6) *# Concatenation*

>>> T[0] *# Indexing, slicing, and more*

>>> T.index(4) *# Tuple methods: 4 appears at offset 3*

>>> T.count(4) *# 4 appears once*

>>> T[0] = 2 *# Tuples are immutable*

...error text omitted...

TypeError: 'tuple' object does not support item assignment

```
>>> T = (2,) + T[1:] # Make a new tuple for a new value
 >>> T
   >>>  T = 'spam', 3.0, [11, 22, 33]
   >>>  T[1]
   >>>  T[2][1]
```

Exercise 6: if Tests and Syntax Rules

Create a file test_if.py that contains the following lines, then run the module, show the results in the lab report.

```
x =1
if x:
    y= 2
    if y:
        print('block2')
    print('block1')
print('block0')


choice = 'ham'
if choice == 'spam':  # the equivalent if statement
   print(1.25)
elif choice == 'ham':
   print(1.99)
elif choice == 'eggs':
   print(0.99)
elif choice == 'bacon':
   print(1.10)
else:
   print('Bad choice')
```

Exercise 7: while and for Loops

Run the following in python shell.

```
>>> x = 'spam'
>>> while x:                # while x is not empty
        print(x,end=' ')  # in 2.X use print x
        x=x[1:]            # strip first character off x

>>> a=0; b=10
>>> while a<b:         # one way to code counter loops
        print(a,end=' ')
        a+=1              # or, a = a + 1

>>> x=10
>>> while x:
        x=x-1                       # or, x -=1
        if x%2 !=0: continue     # odd? – skip print
        print(x, end=' ')

>>> for x in ["spam", "eggs", "ham"]:
         print(x, end=' ')

>>> sum = 0
>>> for x in [1, 2, 3, 4]:
        sum = sum + x
>>> sum

>>> prod = 1
>>> for item in [1,2,3,4]: prod*= item
>>> prod
```

Part II

Exercise 8: Functions

Create fun1.py and fun2.py as the following, run them, show the results, and explain what each function does (and the results) in your own words.

fun1.py

```
def times(x,y):       # create and assign function
    return x*y        # Body executed when called

a = times(2,4)
b = times('Ok', 4)          # Functions are "typeless"

print(a,'\n', b)
```

fun2.py

```
def intersect(seq1, seq2):
    res = []                        # Start empty
    for x in seq1:                  # Scan seq1
        if x in seq2:               # Common item?
            res.append(x)           # Add to end
    return res

s1 = "SPAM"
s2 = "SCAM"
result1 = intersect(s1, s2)
print(result1)

result2 = intersect([1, 2, 3], (1, 4))       # mixed type: list & tuple
print(result2)
```

Exercise 9: modules

Create module.py, test1_module.py, test2_module.py, and test3_module.py as the following, run test1_module.py, test2_module, and test3_module.py, show the results, and explain your findings.

module.py

```
a = 10
b = 20

def adder(x, y):      # module attribute
    z = x + y
    return z

def multiplier(x, y):
    z = x*y
    return z
```

test1_module.py

```
import module                          # Get module as a whole
result = module.adder(module.a, module.b)      # Qualify to get names
print(result)
```

test2_module.py

```
c = 5
d = 10
from module import adder          # Copy out an attribute
result = adder(c, d)                    # No need to qualify name
print(result)

from module import a, b, multiplier   # Copy out multiple attributes
result = multiplier(a, b)
print(result)
```

test3_module.py

```
from module import *     # Copy out all attributes
result1 = adder(a, b)
result2 = multiplier(a, b)
print(result1, '\n', result2)
```

Exercise 10: built-in attribute of modules

Each module has a built-in attribute: __name__.
If the file is being run as a top-level program file, __name__ is set to the string "__main__" when it starts.
If the file is being imported, __name__ is instead set to the module's name as known by its clients.

Read the following example:

runme.py

```
def tester()
    print “It’s Christmas in Heaven…”

if __name__==’__main__’:      # Only when run
  tester()                    # Not when imported
```

(1) Create minmax.py as the following, run it, show the results, and explain what this .py file does.

```
def minmax(test,array):
    res = array[0]
    for arg in array[1:]:
        if test(arg, res):
            res = arg
    return res

def lessthan(x,y): return x<y
def grtrthan(x,y): return x>y

print(minmax(lessthan, [4,2,1,5,6,3]))    # self-test code
print(minmax(grtrthan, [4,2,1,5,6,3]))
```

(2) Create minimax2.py as the following, run it, show the results, and explain what this .py file does.

```
def minmax(test,array):
    res = array[0]
    for arg in array[1:]:
        if test(arg, res):
            res = arg
    return res

def lessthan(x,y): return x<y
def grtrthan(x,y): return x>y

if __name__ == '__main__':
    print(minmax(lessthan, [4,2,1,5,6,3]))    # self-test code
    print(minmax(grtrthan, [4,2,1,5,6,3]))
```

(3) From python shell, execute the following two commands, show the results, and explain your findings.

```
import minmax
import minmax2
```

Exercise 11: Object-Oriented Programming and Classes

(1) Create class1.py as the following, run it, show the results, and explain what the code does.

```
class FirstClass:  # define a class object
   def setdata(self,value1, value2): # Define class's methods
      self.data1=value1                # self is the instance
      self.data2=value2
   def display(self):
      print(self.data1, '\n', self.data2, '\n')


x=FirstClass() # make one instance

x.setdata("King Arthur",-5)   # Call methods: self is x
x.display()


x.data1="QQ"
x.data2=-3
x.display()

x.anothername="spam"
x.display()
print(x.anothername)
```

(2) Class Inheritance: create class2.py as the following, run it, show the results, and explain what the code does.

```
class FirstClass:
   def setdata(self,value1, value2):
      self.data1=value1
      self.data2=value2
   def display(self):
      print(self.data1, '\n', self.data2, '\n')
```

```
class SecondClass(FirstClass): # inherits setdata and display
   def adder(self,val1,val2):     # create adder
            a=val1+val2
            print(a)

z=SecondClass() # make one instance
z.setdata(10,20)
z.display()
z.adder(-5,-10)
```

(3) The constructor method: create class3.py as the following, run it, show the results, and explain what the code does.

```
class Person:
   def __init__(self, name, jobs, age=None): # __init__ is the constructor method of a class
                                #it is to initialize the object's states
      self.name=name
      self.jobs=jobs
      self.age=age

   def info(self):       # another method of the class
      return (self.name, self.jobs)

rec1 = Person('Bob', ['dev', 'mgr'], 40.5)
rec2 = Person('Sue', ['dev', 'cto'])

print(rec1.jobs)
print(rec2.info())
```

(4) Classes are attributes in modules: create class_as_module.py as the following, run it, show the results, and explain what the code does.

class_as_module.py

```
import class3

rec3 = class3.Person('Jane', ['dev', 'mgr'], 30)


print(rec3.age)
print(rec3.info())
```

```
from class3 import Person

rec4 = Person('Mike', ['dev', 'mgr'], 35)
print(rec4.age)
print(rec4.info())
```