COEN 140
Lab 1 Report
Stephen Tambussi - 00001469512

Part 1

Exercise 1: Numbers

```
>>> a=123+222 #integer addition
>>> print(a)
345
>>> b=1.5*4 #floating-point multiplication
>>> print(b)
6.0
>>> c=2**10 #2 to the power of 10
>>> print(c)
1024
>>> import math
>>> print(math.pi)
3.141592653589793
>>> print(math.sqrt(36))
6.0
>>> import random
>>> a=random.random()
>>> print('a=', a)
a= 0.5939158862237326
>>> b=random.choice([1,2,3,4])
>>> print('b=', b)
b= 3
```

Exercise 2: Strings

```
>>> S='Spam' #make a 4-character string, and assign it to a name
>>> len(S) #length
4
>>> S[0] #first item in S
'S'
>>> S[1] #second item in S
'p'
>>> S[-1] #the last item from the end in S
'm'
>>> S[-2] #the second to last item from the end
'a'
>>> S[len(S)-1]
```

```
'm'
>>> S[1:3] #Slice of S from offsets 1 through 2 (not 3)
'pa'
>>> S = 'z' + S[1:]
>>> S
'zpam'
```

Exercise 3: Lists

```
>>> L=[123, 'spam', 1.23] # List of three different type objects
>>> len(L)
3
>>> L[0]
123
>>> L[:-1] #Slicing a list returns a new list
[123, 'spam']
>>> L+[4,5,6] #contact/repeat make new lists too
[123, 'spam', 1.23, 4, 5, 6]
>>> L*2 #repeat
[123, 'spam', 1.23, 123, 'spam', 1.23]
>>> L
[123, 'spam', 1.23]
>>> M = ['bb', 'aa', 'cc']
>>> M.sort()
>>> M
['aa', 'bb', 'cc']
>>> M.reverse()
>>> M
['cc', 'bb', 'aa']
>>> M = [[1,2,3],[4,5,6],[7,8,9]] #a list contains three other lists, 3x3 matrix
>>> M[1] #gets row 2
[4, 5, 6]
>>> M[1][2] #get row 2, then get item 3 within the row
6
>>> diag = [M[i][i] for i in [0,1,2]]
>>> diag
[1, 5, 9]
>>> doubles = [c * 2 for c in 'spam']
>>> doubles
['ss', 'pp', 'aa', 'mm']
>>> list(range(4)) #0..3
[0, 1, 2, 3]
```

```
>>> list(range(-6,7,2)) #-6 to +6 by 2
[-6, -4, -2, 0, 2, 4, 6]
>>> [[x ** 2, x ** 3] for x in range(4)] #Multiple values, "if" filters
[[0, 0], [1, 1], [4, 8], [9, 27]]
>>> [[x, x/2, x*2] for x in range(-6, 7, 2) if x > 0]
[[2, 1.0, 4], [4, 2.0, 8], [6, 3.0, 12]]
```

Exercise 4: Linear Algebra

4.1

```
>>> p=[[1,0],[0,1],[1,1]]
>>> q=[[1,2],[3,4]]
>>> numpy.dot(p,q)
array([[1, 2],
       [3, 4],
       [4, 6]])
>>> numpy.matmul(p,q)
array([[1, 2],
       [3, 4],
       [4, 6]])
```

4.2

```
v=numpy.arange(7)
m=numpy.matrix('1,2; 3,4')
s1 = numpy.linalg.norm(v)
s2 = numpy.linalg.norm(m)
print(s1)
print(s2)
9.539392014169456
5.477225575051661
```

4.3

```
>>> m=numpy.array([[1,2],[3,4]])
>>> numpy.linalg.inv(m)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Exercise 5: Tuples

```
>>> T=(1,2,3,4) #4 item tuple
>>> len(T)
4
>>> T + (5,6)
(1, 2, 3, 4, 5, 6)
>>> T[0]
1
```

```
>>> T.index(4)
3
>>> T.count(4)
1
>>> T[0]=2
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    T[0]=2
TypeError: 'tuple' object does not support item assignment
>>> T=(2,) + T[1:]
>>> T
(2, 2, 3, 4)
>>> T='spam',3.0,[11,22,33]
>>> T[1]
3.0
>>> T[2][1]
22
```

Exercise 6: if Tests and Syntax Rules

```
block2
block1
block0
1.99
```

Exercise 7: while and for loops

```
>>> x='spam'
>>> while x:
        print(x,end='')
        x=x[1:] #strip first character off x
spampamamm
>>> a=0; b=10
>>> while a < b:
        print(a,end='')
        a+=1
0123456789
>>> x=10
>>> while x:
        x=x-1
        if x%2 != 0: continue
        print(x,end='')
86420
>>> for x in ["spam", "eggs", "ham"]:
```

```
        print(x,end='')
spameggsham
>>> sum=0
>>> for x in [1,2,3,4]:
        sum=sum+x
>>> sum
10
>>> prod=1
>>> for item in [1,2,3,4]: prod *= item
>>> prod
24
```

Part 2

Exercise 8: Functions

fun1.py

```
8
 OkOkOkOk
```

- This function multiplies the arguments and returns the results. For integers this returns as expected, but for another variable type (e.g. ‘Ok’) it returns that value with the amount of instances multiplied by.

fun2.py

```
['S', 'A', 'M']
[1]
```

- This function takes two arguments, finds the common elements between them, and then returns the common elements. In the case of SPAM and SCAM, the function returned SAM. With [1,2,3] and (1,4), the function returned 1.

Exercise 9: Modules

test1_module.py

```
30
```

- This program imports the module.py program, uses the adder function defined in that program to add the a and b variables also in that program, and prints the results.

test2_module.py

```
15
200
```

- This program creates two variables c and d, imports the adder function from module.py, and then uses the adder function to get the sum of c and d. The second portion of the program imports the variables a and b with the multiplier function from module.py. Then, the program gets the product of a and b using the multiplier function.

test3_module.py

```
30
 200
```

- This program imports all the variables and functions from module.py. Then, it gets the sum and product of a and b to print out the result.

Exercise 10: built-in attribute of modules

1) minmax.py

1

6

- This program creates a function (minmax) that takes a function and array as arguments and then calls the function argument on each element in the array except for the first element. Minmax will update the res variable if the called function meets its condition. The program also creates two functions called lessthan and grtrthan to be used as arguments to the minmax function. Finally, the program prints the results of calling minmax with the lessthan, grtrthan functions and an integer array.

2) minmax2.py

1

6

- This program does the same thing as minmax.py except that it will only execute the minmax function call with the print statement if the program is being run as a top-level program and not imported.

3)

```
>>> import minmax
1
6
>>> import minmax2
>>>
```

- Importing minmax still executes because it does not have the conditional statement at the end of the program that is included in minmax2. This conditional statement in minmax2 ensures that the code is only executed when minmax2 is the top-level program and not imported.

Exercise 11: Object-Oriented Programming and Classes

1) class1.py

```
King Arthur
 -5

QQ
 -3

QQ
 -3

spam
```

- This program defines a class (FirstClass) with methods setdata and display. Then, it creates an instance of the class, x, and sets the data to “King Arthur” and -5 and displays this data. The program continues with other values for the class instance variables.

2) class2.py

10
20

-15

- This program creates a SecondClass that inherits the FirstClass with its methods. Then an instance of the SecondClass is created, z, and the data for this instance is set and displayed. Finally, the z instance calls the adder function and displays the result of adding the two arguments.

3) class3.py

['dev', 'mgr']
('Sue', ['dev', 'cto'])

- The class Person has a constructor method to initialize the object’s states and an info method to display the values of an instance of a class. Two instances of the class are created, rec1 and rec2 with different arguments. Then, the values of the jobs variable for rec1 are printed out followed by the call of the info method on rec2.

4) class_as_module.py

['dev', 'mgr']
('Sue', ['dev', 'cto'])
30
('Jane', ['dev', 'mgr'])
35
('Mike', ['dev', 'mgr'])

- This program has the output from class3.py as it is completely imported and then creates an instance of the class Person, rec3. The age value of rec3 is printed and then the info method is called on rec3. Then, only the Person class from class3.py is imported and an instance is created called rec4. The age value of rec4 is printed and then the info method is called on rec4.