COEN 166 Artificial Intelligence
Lab Assignment #1
Tanner Kaczmarek 00001443463

PART I

Exercise 1: Numbers

```
>>> a=123+122
>>> print(a)
245
>>> b = 1.5*4
>>> print(b)
6.0
>>> c=2**10
>>> print(c)
1024
>>> import math
>>> print(math.pi)
3.14159265359
>>> print(math.sqrt(36))
6.0
>>> import random
>>> a = random.random()
>>> print('a=', a)
a= 0.7846515078431336
>>> b = random.choice([1,2,3,4])
>>> print('b=', b)
b= 3
```

Exercise 2: Strings

```
>>> S='Spam'
>>> len(S)
4
>>> S[0]
'S'
>>> S[1]
'p'
>>> S[-1]
'm'
>>> S[-2]
'a'
>>> S[len(S)-1]
'm'
```

```
>>> S[1:3]
'pa'
>>> S= 'z' +S[1:]
>>> S
'zpam'
```

Exercise 3: Lists

```
L=[123, 'spam', 1.23]
>>> len(L)
3
>>> L[0]
123
>>> L[:-1]
[123, 'spam']
>>> L+[4,5,6]
[123, 'spam', 1.23, 4, 5, 6]
>>> L*2
[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]]
>>> M[1]
[4, 5, 6]
>>> M[1][2]
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']
```

Exercise 4: Dictionaries

```
D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
>>> D['food']
'Spam'
>>> D['quantity']+=1
```

```
>>> D
{'food': 'Spam', 'quantity': 5, 'color': 'pink'}
>>> D = {}
>>> D['name']='Bob'
>>> D['job']='dev'
>>> D['age']=40
>>> D
{'name': 'Bob', 'job': 'dev', 'age': 40}
>>> print(D['name'])
Bob
>>> bob1 = dict(name='Bob', job='dev', age=40)
>>> bob1
{'name': 'Bob', 'job': 'dev', 'age': 40}
>>> bob2 = dict(zip(['name', 'job', 'age'], ['Bob', 'dev', 40])) #
Zipping
>>> bob2
{'name': 'Bob', 'job': 'dev', 'age': 40}
```

Exercise 5: Tuples

```
>>> T = (1,2,3,4)
>>> len(T)
4
>>> T + (5,6)
(1, 2, 3, 4, 5, 6)
>>> T[0]  # Indexing, slicing, and more
1
>>> T.index(4)   # Tuple methods: 4 appears at offset 3
3
>>> T.count(4)   # 4 appears once
1
>>> T[0] = 2 # Tuples are immutable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> T = (2,) + T[1:] # Make a new tuple for a new value
>>> 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

```
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’)
```

```
python3 test_if.py
block2
block1
block0
1.99
```

Exercise 7: while and for Loops

```
>>> x = 'spam'
>>> while x:
...    print(x,end='')
...    x =x[1:]
...
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 II

Exercise 8: Functions

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)
```

Answer:

```
8
 OkOkOkOk
```

Comments:
Fun1.py has a function defined called times. Times takes in two arguments x and y and returns the calculated product of x and y. The is typeless so on the first call it takes in two numbers and returns the multiplication of those two numbers (e.g 8). However on the second time because it is typeless it takes in the string ‘Ok’ and the number 4 and returns Ok 4 times in a row as it does its best to work with the different types it was given.

Fun2.py

```
def intersect(seq1, seq2):
```

```
   res = []
   for x in seq1:
      if x in seq2:
         res.append(x)
   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)
```

Answer:

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

Comments:
Fun2.py has a function called interest that is called twice. Intersect is a function that takes to sequences and appends the common items in both of these and returns the common item in both of these as a list. The first time it is called it appends the common items of "SPAM" and "SCAM" and returns the common items of both (e.g "SAM"). The second time it is called it is passed a list and a tuple and returns 1.

Exercise 9: modules

module.py:

```
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)
```

Answer:
```
30
```

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)
```

Answer:
```
15
200
```

test3_module.py:

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

Answer:
```
30
 200
```

Comments:
Module.py has a multiplier function and an adder function which multiply and add as one would expect from their function names. For test1_module.py it sums up module.a and module.b which are declared in module.py. Test1 prints out 30 as expected from the sum of 20 + 10. For test2_module.py it defines two new variables, c=5 and d=10 and sends these two to the adder function in Module.py to get a sum of 15. Test2 also imports a and b and the multiplier function and sends these a and b to the newly imported multiplier function. Test3_module.py imports the entirety of the module functions and uses it to add a and b together and to multiply a and b together. I learned from this that everything is public and can be imported. If you do import module you will need to call stuff with a module.attribute but if you do from module import attribute you will import desired attribute and do not need to do module.attribute.

Exercise 10: built-in attribute of modules

runme.py

```
def tester():
   print “It’s Christmas in Heaven…”
if __name__==’__main__’: # Only when run
   tester() # Not when imported
```

minmax.py

```
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]))
```

```
python3 minmax.py
1
6
>>> import minmax
1
6
```

minmax2.py

```
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]))
```

```
python3 minmax2.py
1
6
>>> import minmax2
>>>
```

Comments:
Both minmax.py and minmax2.py have a function called minmax that when given an array to test, it will return the maximum and minimum values in that array. It does this with two helper functions called lessthan and grtrthan that do as their names explain. When run by python3 minmax.py or python3 minmax2.py they print out the same answer but when you import the function minmax.py acts as it used to but minmax2.py does not print anything. That is because of this additional line: i*f __name__ == '__main__':* This if line of code only prints and runs minmax when the file's name is '_main_'. The file's name is main when it is ran as a top-level program file but when it is imported the files name is actually minmax2.py so minmax2.py will not run when it is imported.

Exercise 11: Object-Oriented Programming and Classes

Class1.py

```
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)
```

Answer:

```
King Arthur
 -5

QQ
 -3

QQ
 -3

spam
```

Comments:
Class1.py has a class called FirstClass with member function that allow you to set and display the data of the class. Class1.py defines x as an instance of FirstClass. It sets the data using the setData the first time and displays the information. Next time it displays, it accesses the variables of the class directly to change the values of data1 and data2 instead of using the set data function. The last display shows no changes done to x when used x.anothername="spam" but another name is still saved with x as shown by using the print function instead (e.g. print(x.anothername)).

Class2.py

```
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)
```

Answer:

```
10
 20

-15
```

Comments:

For class2.py, there are two different classes, and the SecondClass inherits from the FirstClass. When it inherits from the first class it has all the functions of the SecondClass as well as the functions from the FirstClass. That is why when Z is an instance of the SecondClass it can still use functions from the FirstClass such as z.setdata(10,20).

Class3.py

```
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())
```

Answer:

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

Comments:
Class3.py has a class called Person which has function which is a constructor that initializes any instance of the class. It also has another function called self that will return the instance of the class's name and jobs. The first instance of the class is rec1 named 'Bob' who has jobs of ['dev', 'mgr'] and is 40.5 years old. Later it asks to print the rec1.jobs and prints out ['dev', 'mgr']. The second instance is rec2 named sue with jobs of ['dev', 'cto']. When asked to print out rec2.info it prints the entire info for it meaning jobs and name.

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())
```

Answer:

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

Comments:
The first thing class_as_module.py does is it imports class3, which makes class3.py run as discussed before and is the first two lines printed from the program. Next it creates another instance of person but because it did import class3 instead of from class3 import it needs to create the person with a class3.Person in front of it. This new person is created name Jane and Jane's age is printed, then Jane's entire information. After that the class_as_module.py now imports from class3 Person so it no longer needs to index class3.Person to create a person. The new instance of Person is created called rec4 named Mike, and Mike's age and entire information is printed.