Daren Liu
00001471643

Part I:

Exercise 1: Numbers

```
>>> a = 123 + 222
>>> print(a)
345
>>> b = 1.5*4
>>> print(b)
6.0
>>> c=2**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= 0.13226885735943905
>>> b=random.choice([1,2,3,4])
>>> print('b=', b)
b= 2
```

Exercise 2: Strings

```
>>> S = 'Spam' #make a 4-character string, and assign it to a name
>>> len(S)
4
>>> S[0] #the 1st item in S, indexing by zero-based position
'S'
>>> S[1] #the 2nd item from the left
'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]
>>> 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']
>>> list(range(4))
[0, 1, 2, 3]
>>> list(range(-6,7,2))
[-6, -4, -2, 0, 2, 4, 6]
>>> [[x**2, x**3] for x in range(4)]
[[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: Dictionaries

```
>>> D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
>>> D['food']
'Spam'
>>> D['quantity'] += 1
>>> D
{'food': 'Spam', 'quantity': 5, 'color': 'pink'}
>>> D = {}
>>> D['job'] = 'dev'
>>> D['age'] = 40
>>> D
{'job': 'dev', 'age': 40}
>>> 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]))
>>> 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]
1
>>> T.index(4)
3
>>> T.count(4)
1
>>> T[0] = 2
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
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
Code:

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

choice = 'ham'
if choice == 'spam':
   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')
```

Output:

```
block2
block1
block0
1.99
```

Exercise 7: while and for Loops

```
>>> x = 'spam'
>>> while x:
...     print(x,end=' ')
...     x = x[1:]
...
spam pam am m
>>> a=0; b=10
>>> while a<b:
...     print(a,end=' ')
...     a+=1
...
0 1 2 3 4 5 6 7 8 9
>>> while x:
...     x = x-1
...     if x%2 != 0: continue
...     print(x, end=' ')
...
8 6 4 2 0
>>>for x in ["spam", "eggs", "ham"]:
...     print(x,end=' ')
...
spam eggs ham
>>> 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)
```

Output:
8
 OkOkOkOk
Explanation:
The parameters x and y in the function times are typeless, which means that they accept any type of variable. This makes sense, since in python, you do not have to declare, “int x =” or “string s = “, just “x =”. In this case, the code computes a = times(2, 4), which returns 8. However, when the code reaches b = times(‘Ok’, 4), since functions in python do not specify what type of parameters it requires or what type to return, it returns ‘Ok’ 4 times, thus resulting in ‘OkOkOkOk’. Finally, since there is a ‘\n’ in between a and b in print, the ‘8’ and ‘OkOkOkOk’ print on different lines.

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)
```

Output:
['S', 'A', 'M']
[1]
Explanation:
When intersect is called for the first time, both of the parameter inputs are strings. It first iterates through “SPAM” and checks if any of the letters are in “SCAM”, and if so, it appends the letter to the list res. Thus, the first output is [‘S’, ‘A’, ‘M’], since these letters that are in SCAM are in SPAM. The second output prints [1], since it is the only integer that is in both the list [1,2,3] and the tuple (1, 4), which proves that the function can iterate through both a list and a tuple.

Exercise 9: modules
module.py
```
a = 10
```

```python
b = 20
def adder(x, y): #module attribute
   z = x + y
   return z

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

Output:
Nothing
Explanation:
module.py doesn't print anything because it's only returning values in the functions, there are no print statements in the code.

test1_module.py

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

Output:
30
Explanation:
Import module imports everything in the module. In the code, we are assigning result to the output of the function adder in the module, with variables a and b in the module acting as the parameter of module.adder, which ends up to be 30.

Test2_module.py

```python
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)
```

Output:
15
200

Explanation:
From module import adder only imports the adder function from module.py, not a and b. So the first result prints out the sum of c and d, which is 15. From module import a, b, multiplier imports variables a and b as well as the function multiplier from module. Thus, the second print result prints 20*10, which is 200.

test3_module.py

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

Output:

```
30
 200
```

Explanation:
From module import * imports all variables and functions from the module. So the result of adder(a, b) is 30, and the result of the multiplier(a, b) is 200.

Exercise 10: built-in attribute of modules
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]))
```

Output:

```
1
6
```

Explanation:

In the function minmax, we are able to call the function lessthan and grtrthan as a parameter, and an array as our other parameter. In minmax, we are parsing through the entire array, and testing each element if it is greater than or less than res, depending on the function being called. If an element is less than or greater than res, then res is set to this element. After going through the entire list, the min or max element is returned.

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]))
```

Output:

```
1
6
```

Explanation:

The output is the same as minmax.py. However, minmax2.py would not run if it was imported into another file, since it has if __name__ == '__main__' at the bottom. This ensures that the function minmax is only called if it is the highest level.

Python shell:

Import minmax.py

Output:

```
1
6
```

Explanation:

Since there is no if '__name__' == '__main__' at the bottom, there is nothing stopping minmax from being imported

Import minmax2.py

Output:
None
Explanation:
Since minmax2 has if __name__ == '__main__' at the bottom and we are importing the file in the python shell, nothing is outputted.

Exercise 11: Object-Oriented Programming and Classes
class1.py
```python
class FirstClass:
   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)
```

Output:
```
King Arthur
 -5

QQ
 -3

QQ
 -3

spam
```
Explanation:
The code first creates one instance of the class FirstClass. The code then sets two sets of data by calling the setdata() function, with "King Arthur" and -5 as parameters. Then, x.display() prints

data1 and data2 stored in the class, which was already set as “King Arthur” and -5 earlier. Thus “King Arthur” and -5 are printed. Next, data1 and data2 in the class are reassigned to “QQ” and -3, which is why when x.display() is called, it prints “QQ” and -3 instead of the previous outputs. Next, a variable anothername is set to spam in the class. However,since data1 and data2 wasn’t changed, when x.display() is called, the code still outputs “QQ” and -3. It is only when print(x.anothername) is called is when spam is outputted.

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)
```

Output:

```
10
 20

-15
```

Explanation:

First, one instance of SecondClass is made. Since SecondClass inherits all attributes from FirstClass, when z.display() is called, the code outputs 10 ‘\n’ 20 since z.setdata(10, 20) was also called earlier. When z.adder(-5, -10) is called, the code outputs -15 since it calls the member function in SecondClass, which prints the sum of two parameters.

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())
```

Output:
['dev', 'mgr']
('Sue', ['dev', 'cto'])
Explanation:
Rec1 is assigned to the Person class once, which invokes the constructor. However, in the class, the constructor only requires a name and jobs, not an age, which is why when rec2 is assigned, there's only a name and job parameter. print(rect1.jobs) simply prints the first instance's jobs, while print(rec2.info) prints the name and job of rec2, since info() is a member function of the class.

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())
```

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

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

Explanation:

The code first prints ['dev', 'mgr'] and ('Sue', ['dev', 'cto']) since those were the outputs of class3, so any outputs of any imported modules are outputted first. rec3 then makes an instance of Person, which is in class3. The parameters passed include age, which is why when print(rec3.age) is run, it prints 30. When print(rec3,info()) is fun, it prints ('Jane', ['dev', 'mgr']), which only asks for the name and jobs in the class. However, when rec4 makes an instance of Person, there’s no need to type class.Person since it is only importing Person from class3. This is proved when ['dev', 'mgr'] and ('Sue', ['dev', 'cto']) is not outputted when from class3 import Person is typed, but is printed when only import class3 is. Once again, print(rec4.age) prints 35, which was specified in rec4, and the member class infor() is also available, as it returns ('Mike', ['dev', 'mgr']).