COEN 166 Artificial Intelligence

Lab Assignment #1

Carlo Bilbao **1608742**

Part I

Exercise 1: Numbers

```
>>> a=123+222
>>> print(a)
345
>>> b=1.5*4
>>> print(b)
6.0
>>> c=2**10 #2 to the power 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= 0.5703699004607917
>>> b = random.choice([1,2,3,4])
>>> print('b=', b)
b= 2
```

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] # A list of three different-type objects
>>> len(L) # number of items in the list
3
>>> L[0]
123
>>> L[:-1] # Slicing a list returns a new list
[123, ‘spam’]
>>> L+[4,5,6] # concatenation makes a new list
[123, ‘spam’, 1.23, 4, 5, 6]
>>> L.insert(0, ‘apple’) # insert an item at the beginning of the list
>>> L
[‘apple’, 123, ‘spam’, 1.23]
>>> L=[123, ‘spam’, 1.23]
>>> L.insert(2, ‘apple’) # insert an item at position 2 of the list
>>> L
[123, ‘spam’, ‘apple’, 1.23]
>>> L=[1,10,2,5,8]
>>> L.sort()
>>> L
[1, 2, 5, 8, 10]
>>> L.reverse()
>>> L
[10, 8, 5, 2, 1]
>>> list(range(4)) # make a list with elements 0, 1, 2, 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)] # calculate the power of x, for x = 0, 1, 2, 3
[[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: Tuple

```
>>> pair = (3,5)
>>> pair
(3, 5)
>>> pair[0]
3
>>> x,y = pair
>>> x
3
>>> y
5
```

Exercise 5: if Tests and Syntax Rules

Create a file test_if.py that contains the following lines, then run the module, observe the results.

```
x =1
if x:
        y= 2
        if y>0:
                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')
```

Answer:

The program prints

```
block2
block1
block0
1.99
```

Comments:

Thus, indentations in the code designate what is associated with each if block. Everything is executed from top to bottom, as usual. You don’t have to specify the data type.

Exercise 6: 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

>>> thislist = ["spam", "eggs", "ham"]
>>> for x in thislist:
        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 7: Functions

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

fun1.py

```
def times(x,y): # create a function
return x*y # Body executed when called
a = times(2,4)
b = times(‘Ok’, 4) # Functions are “typeless”
print(a,’\n’, b)
```

Answer:

The program prints

```
8
OkOkOkOk
```

Comments:

The Fun1.py program defines a function “times.” This function takes in two arguments x and y, and calculates their product

Then the function times is called with parameters (2, 4), and the results are stored in a

The function times is called again with parameters (‘Ok’, 4), and the results are stored in b. It appears that ‘Ok’ is repeated 4 times, due to functions being “typeless”

Exercise 8: 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) # Go through the module name “module” to fetch
# its attributes “adder”, “a”, and “b”
print(result)
```

6

Answer:

The program prints

```
30
```

Comments:

The program imports another file, module. It takes the variables a and b from the module file and then passes it as arguments into the adder function with its inherent values, storing it into the variable result. Then we print the result

test2_module.py

```
c = 5
d = 10
from module import adder # Copy out an attribute
result = adder(c, d) # No need go through the module name “module” to fetch
# its attribute “adder”
print(result)
from module import a, b, multiplier # Copy out multiple attributes
result = multiplier(a, b)
print(result)
```

Answer:

The program prints

```
15
200
```

Comments:

The program imports attributes from another file, module. In doing this, we do not need to access the attributes of module with the (.) operator. We are importing the function adder and then defining our own variables, c and d, to pass into the function. We store the output of the function, the sum of c and d, into result. We will print the result

We do this process again, this time importing the defined variables a and b, and the function multiplier. We will pass a and b into multiplier and store the output of the function into variable result. Then we print result

test3_module.py

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

Answer:

The program prints

```
30
```

```
200
```

Comments:
The program imports attributes from another file, module. The * means that we are importing everything from the file. So, we are importing a, b, adder, and multiplier. We run the adder function, passing in a and b, and storing it into result1
We then call the multiplier function, passing in a and b, and storing it into result2
After, we print result1 and result2, separated by a new line and an extra space

Exercise 9: 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(fun,array):
 result = array[0]
 for arg in array[1:]:
  if fun(arg, result):
   result = arg
 return result
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]))
```

Answer:
The program prints

```
1
6
```

Comments:
The program defines a function named minmax, which takes in two arguments, a function and an array. In this minmax function, we set a result variable to the first element in the array

parameter. Then, in a for loop, we traverse the rest of the array and check if true is returned by the function that passes in the current element in the array with the result. If it is true, then the current element is the item we are looking for, so store it in result
We define other functions such as lessthan, which takes in two arguments and returns true if x is less than y, false if otherwise
We define grtrthan, which takes in two arguments and returns true if x is greater than y, false if otherwise
Lastly, we print the output of minmax passing in the lessthan function with an array and passing in grtrthan with the same array, with each of their own print statements

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

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

Answer:
The program prints

```
1
6
```

Comments:
The program defines a function named minmax, which takes in two arguments, a function and an array. In this minmax function, we set a result variable to the first element in the array parameter. Then, in a for loop, we traverse the rest of the array and check if true is returned by the function that passes in the current element in the array with the result. If it is true, then the current element is the item we are looking for, so store it in result
We define other functions such as lessthan, which takes in two arguments and returns true if x is less than y, false if otherwise
We define grtrthan, which takes in two arguments and returns true if x is greater than y, false if otherwise
This time, we are defining a main function. It will call two separate print functions, with lessthan and grtrthan as arguments respectively along with an array

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

```
import minmax
import minmax2
```

Answer:

The program prints

```
1
6
1
6
```

Comments:

The program is importing the minmax and minmax2 files. It will execute both files (the print functions for minmax and the print functions for minmax2), yielding the same answers. Although the functions are redefined, it is still possible to execute the program without errors.

Exercise 10: Object-Oriented Programming and Classes

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

```
class PersonClass: # define a class object
 def basicinfo(self, name, age): # Define class's methods
  self.name=name # self is the instance
  self.age=age
 def display(self):
  print(self.name, '\n', self.age, '\n')
person=PersonClass() # make one instance of the class

person.basicinfo("John", 20) # Call methods: self is person
person.display()

person.name="Mary"
person.age=30
person.display()

person.school="Engineering"
person.display()
print(person.school)
```

Answer:

The program prints

```
John
 20

Mary
```

30

Mary
 30

Engineering

Comments:
The program defines a class object called PersonClass
A function called basicinfo is defined to set the attributes of the class, namely name and age
A display function is created in order to print the name and age attributes, with new lines separating the attributes
We then make an instance of the class by calling the default constructor. We will use the basicinfo function in order to set the name and age of this particular instance. Then print the current attribute values using the display function
We then modify the name attribute of the class directly to “Mary.” We do the same with the age and set it to 30. We display the current attributes
Lastly, we set the school attribute to “Engineering.” We display the attributes, which should show “Mary” and 30 again, since it was unchanged. Then we print the school attribute

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

```
class PersonClass: # define a class object
 def basicinfo(self, name, age):
 self.name=name
 self.age=age
 def display(self):
 print(self.name, '\n', self.age, '\n')
class StudentClass(PersonClass): # inherits basicinfo and display
 def moreinfo(self, university, major): # create moreinfo
 print(self.name + ' is a ' + major + ' student ' + 'from ' + university)
student=StudentClass() # make one instance
student.basicinfo('Cathy', 20)
student.display()
student.moreinfo('Santa Clara University', 'CSE')
```

Answer:
The program prints

```
Cathy
 20

Cathy is a CSE student from Santa Clara University
```

Comments:

The program defines a class object called PersonClass
A function called basicinfo is defined to set the attributes of the class, namely name and age
A display function is created in order to print the name and age attributes, with new lines separating the attributes
We define another class named StudentClass that inherits from PersonClass. We define a function called moreinfo, which takes in three arguments and prints them
We make an instance of StudentClass by calling its default constructor. Then we call the function basicInfo and pass in the string ‘Cathy’ and the integer 20 for its attributes. We call its display function to print the name and age. We call moreInfo to display the university and major, set to ‘Santa Clara University’ and ‘CSE’ respectively

(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())
```

Answer:
The program prints

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

Comments:
The program defines a class object called Person
We define a constructor called __init__, that takes in a name variable, a jobs, and sets age initially to None when not specified. It modifies the attributes of the class using these parameters
Then an info function is defined, which only returns the name and jobs attributes as a pair
We create an instance of the Person class with name ‘Bob’, jobs ‘dev’ and ‘mgr’ and age 40.5 and store this information in the object rec1
We do the same thing for rec2, with its name as ‘Sue,’ ‘dev’ and ‘cto’ and age set to None
We print rec1’s jobs and then print the info function call of rec2, which prints its name and jobs

(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())
```

Answer:

The program prints

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

Comments:

The program will import from class3, which takes in the Person class
We use the class3 module to create an instance of the Person class, with name ‘Jane,’ jobs of ‘dev’ and ‘mgr’ and age 30, storing this into the variable rec3. We print the age attribute of rec3 and then print and call rec3’s info function, which returns the object’s name and jobs
We repeat this process again, this time using the from keyword and importing the Person class. We no longer need to use the (.) operator to access attributes or member functions
We create an instance of person and store it into the rec4 variable, with its name set to ‘Mike,’ its jobs as ‘dev’ and ‘mgr,’ and age as 35. We will print the object’s age attribute, and then print the object’s info function, which will return the name and age