CPSC 1301, Computer Science I Lab Assignment

Lab 03a

1 Modules

We will study modules late this semester, but this previews one way to use modules in Python. You have written the following line in your first two Exercises, and you will continue to write this line:

```
if __name__ == "__main__":
    #various function calls and other stuff
```

This line of code allows you to run your program as a stand-alone program, and also to include your program as a *module* in other programs as well as the command line. You will do exactly this in the lab.

Open your `python` interpreter in your console (terminal), and run the following commands. This will import Exercise 1 and Exercise 2 as modules and exercise the functions you have defined. When you finish, copy and past the contents of your terminal window as the deliverable for this lab.

```
C:\Users\ccc31\cols-st\cpsc1301\python-progs>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on
    win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import Hello                #this imports Exercise 1 as a module
>>> Hello.hello()               #the following lines call Exercise 1 functions
>>> Hello.helloname()
>>> Hello.sys_info()
>>> Hello.platform_info()
>>> Hello.print_csu()
>>> import MathFormulas         #this imports Exercise 2 as a module
>>> MathFormulas.hello()        #these lines call the Exercise 2 functions
>>> MathFormulas.circle()
>>> MathFormulas.hemisphere()
>>> MathFormulas.triangle()
>>> MathFormulas.quadratic()
>>> exit()                      #This exits your Python
```

2 CodePost Labs

We will start running CodePost Labs. I will do the first several to demonstrate. After that, I will call on individual students to lead the labs. These are graded labs. You can find them at `https://github.com/ccc31807/CPSC-1301/tree/master/CodePostLabs`.

3 Running Python scripts

Create scripts in files and run them. Run these scripts in your Python prompt.

```
# Name: lab02a01.py
# A first Python script
import sys              # Load a library module
print(sys.platform)
print(2 ** 100)         # Raise 2 to a power
```

```
x = 'Spam!'
print(x * 8)          # String repetition

# Name: lab02a02.py
a = 'dead'            # Define three attributes
b = 'parrot'          # Exported to other files
c = 'sketch'
print(a, b, c)        # Also used in this file (in 2.X: print a, b, c)
```

1.

4 At the Python prompt

Run the following commands in your Python interpreter. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named `lab02a_lastname.txt`.

```
>>> 'spam'
>>> type('spam')
>>> 123 + 222 # Integer addition
>>> 1.5 * 4 # Floating-point multiplication
>>> 2 ** 100 # 2 to the power 100, again
>>> len(str(2 ** 1000000))
>>> 3.1415 * 2 # repr: as code (Pythons < 2.7 and 3.1)
>>> print(3.1415 * 2) # str: user-friendly
>>> import math
>>> math.pi
>>> math.sqrt(85)
>>> import random
>>> random.random()
>>> random.choice([1, 2, 3, 4])
>>> S = 'Spam' # Make a 4-character string, and assign it to a name
>>> len(S) # Length
>>> S[0] # The first item in S, indexing by zero-based position
>>> S[1] # The second 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[-1] # The last item in S
>>> S[len(S)-1] # Negative indexing, the hard way
>>> S # A 4-character string
>>> S[1:3] # Slice of S from offsets 1 through 2 (not 3)
>>> S[1:] # Everything past the first (1:len(S))
>>> S # S itself hasn't changed
>>> S[0:3] # Everything but the last
>>> S[:3] # Same as S[0:3]
>>> S[0] = 'z' # Immutable objects cannot be changed
>>> S = 'z' + S[1:] # But we can run expressions to make new objects
>>> S
>>> S = 'shrubbery'
>>> L = list(S) # Expand to a list: [...]
>>> L
>>> L[1] = 'c' # Change it in place
>>> ''.join(L) # Join with empty delimiter
>>> B = bytearray(b'spam') # A bytes/list hybrid (ahead)
>>> B.extend(b'eggs') # 'b' needed in 3.X, not 2.X
>>> B # B[i] = ord(c) works here too
>>> B.decode() # Translate to normal string
>>> S = 'Spam'
>>> S.find('pa') # Find the offset of a substring in S
>>> S
>>> S.replace('pa', 'XYZ') # Replace occurrences of a string in S with another
>>> S
>>> line = 'aaa,bbb,ccccc,dd'
>>> line.split(',') # Split on a delimiter into a list of substrings
>>> S = 'spam'
>>> S.upper() # Upper- and lowercase conversions
>>> S.isalpha() # Content tests: isalpha, isdigit, etc.
>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line.rstrip() # Remove whitespace characters on the right side
>>> line.rstrip().split(',') # Combine two operations
>>> '%s, eggs, and %s' % ('spam', 'SPAM!') # Formatting expression (all)
>>> '{0}, eggs, and {1}'.format('spam', 'SPAM!') # Formatting method (2.6+, 3.0+)
>>> '{}, eggs, and {}'.format('spam', 'SPAM!') # Numbers optional (2.7+, 3.1+)
>>> '%.2f | %+05d' % (3.14159, -42) # Digits, padding, signs
>>> dir(S)
>>> S + 'NI!'
>>> S.__add__('NI!')
>>> S[:-1] # Everything but the last again, but simpler (0:-1)
>>> S[:] # All of S as a top-level copy (0:len(S))
>>> S
>>> S + 'xyz' # Concatenation
>>> S # S is unchanged
>>> S * 8 # Repetitio
>>> help(S.replace)
>>> S = 'A\nB\tC' # \n is end-of-line, \t is tab
>>> len(S) # Each stands for just one character
>>> ord('\n') # \n is a byte with the binary value 10 in ASCII
>>> S = 'A\0B\0C' # \0, a binary zero byte, does not terminate string
>>> len(S)
>>> S # Non-printables are displayed as \xNN hex escapes
>>> msg = """
>>> msg
>>> L = [123, 'spam', 1.23] # A list of three different-type objects
>>> len(L) # Number of items in the list
>>> L[0] # Indexing by position
>>> L[:-1] # Slicing a list returns a new list
>>> L + [4, 5, 6] # Concat/repeat make new lists too
>>> L * 2
>>> L # We're not changing the original list
>>> L.append('NI') # Growing: add object at end of list
>>> L
>>> L.pop(2) # Shrinking: delete an item in the middle
>>> L # "del L[2]" deletes from a list too
>>> M = ['bb', 'aa', 'cc']
>>> M.sort()
>>> M
>>> M.reverse()
>>> M
>>> L
>>> L[99]
>>> L[99] = 1
>>> M = [[1, 2, 3],[4, 5, 6], [7, 8, 9]] # A 3 × 3 matrix, as nested lists
>>> M
>>> M[1] # Get row 2
>>> M[1][2] # Get row 2, then get item 3 within the row
>>> col2 = [row[1] for row in M] # Collect the items in column 2
>>> col2
>>> M # The matrix is unchanged
>>> [row[1] + 1 for row in M] # Add 1 to each item in column 2
>>> [row[1] for row in M if row[1] % 2 == 0] # Filter out odd items
>>> diag = [M[i][i] for i in [0, 1, 2]] # Collect a diagonal from matrix
>>> diag
>>> doubles = [c * 2 for c in 'spam'] # Repeat characters in a string
>>> 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]
>>> G = (sum(row) for row in M) # Create a generator of row sums
>>> next(G)
>>> next(G) # Run the iteration protocol next()
>>> next(G)
>>> list(map(sum, M)) # Map sum over items in M
>>> {sum(row) for row in M} # Create a set of row sums
>>> {i : sum(M[i]) for i in range(3)} # Creates key/value table of row sums
>>> [ord(x) for x in 'spaam'] # List of character ordinals
>>> {ord(x) for x in 'spaam'} # Sets remove duplicates
>>> {x: ord(x) for x in 'spaam'} # Dictionary keys are unique
>>> (ord(x) for x in 'spaam') # Generator of values
>>> D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
>>> D['food'] # Fetch value of key 'food'
>>> D['quantity'] += 1 # Add 1 to 'quantity' value
>>> D
>>> D = {}
>>> D['name'] = 'Bob' # Create keys by assignment
>>> D['job'] = 'dev'
>>> D['age'] = 40
>>> D
>>> print(D['name'])
>>> bob1 = dict(name='Bob', job='dev', age=40) # Keywords
>>> bob1
>>> bob2 = dict(zip(['name', 'job', 'age'], ['Bob', 'dev', 40])) # Zipping
>>> bob2
>>> rec = {'name': {'first': 'Bob', 'last': 'Smith'},
>>> rec['name'] # 'name' is a nested dictionary
>>> rec['name']['last'] # Index the nested dictionary
>>> rec['jobs'] # 'jobs' is a nested list
>>> rec['jobs'][-1] # Index the nested list
>>> rec['jobs'].append('janitor') # Expand Bob's job description in place
>>> rec
>>> rec = 0
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> D
>>> D['e'] = 99 # Assigning new keys grows dictionaries
>>> D
>>> D['f']
>>> 'f' in D
>>> if not 'f' in D: # Python's sole selection statement
>>>   print('missing')
>>> value = D.get('x', 0) # Index but with a default
>>> value
>>> value = D['x'] if 'x' in D else 0 # if/else expression form
>>> value
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> D
>>> Ks = list(D.keys()) # Unordered keys list
>>> Ks # A list in 2.X, "view" in 3.X: use list()
>>> Ks.sort() # Sorted keys list
>>> Ks
>>> for key in Ks: # Iterate though sorted keys
>>> D
>>> for key in sorted(D):
>>>    print(key, '=>', D[key])
>>> for c in 'spam':
>>>    print(c.upper())
>>> x = 4
>>> while x > 0:
>>>     print('spam!' * x)
>>>     x -= 1
>>> squares = [x ** 2 for x in [1, 2, 3, 4, 5]]
>>> T = 'spam', 3.0, [11, 22, 33]
>>> T[1]
>>> T[2][1]
>>> T.append(4)
>>> squares
>>> squares = []
>>> for x in [1, 2, 3, 4, 5]: # This is what a list comprehension does
>>>     squares.append(x ** 2) # Both run the iteration protocol internally
>>> squares
>>> T = (1, 2, 3, 4) # A 4-item tuple
>>> len(T) # Length
>>> 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
>>> T = (2,) + T[1:] # Make a new tuple for a new value
>>> T
>>> f = open('data.txt', 'w') # Make a new file in output mode ('w' is write)
>>> f.write('Hello\n') # Write strings of characters to it
>>> f.write('world\n') # Return number of items written in Python 3.X
>>> f.close()
>>> f = open('data.txt') # 'r' (read) is the default processing mode
>>> text = f.read() # Read entire file into a string
>>> text
>>> print(text) # print interprets control characters
>>> text.split() # File content is always a string
```