C200 Programming Assignment №4

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

September 26, 2019

Contents

Introduction

In this homework, you'll start mastering your skill in writing for loops and continue with choice and dictionaries. Three are now hundreds of PowerPoint slides with many, many programs. You should, as a part of your routine, copy these and explore. You must engage this material. The PowerPoint, homework, lab, and additional problems are there for you to develop your skills–moving toward the eureka moment. In lecture we saw how we would try to solve a problem of finding from n boxes containing money, the one that had that most. Review that (and the other slightly weirder implementation). Here are the steps to remind you:

1. Start with pencil and paper
2. Start with the smallest problem that is allowed
3. Solve it!
4. Increment it by the smallest way possible–generally this is one more piece of data...
5. You'll do this a few times, then begin to see a pattern.
6. Translate your thoughts to **comments** that tell you what you should do–programming is really just like telling a story. So you're developing your outline.
7. Begin to program exactly the same way– smallest data available, building up slowly.
8. And *always* remember, a mistake is like a medal of honor. As we've said in lecture, the more mistakes you encounter then overcome, the better you'll become!
9. One thing I do is squeeze my stress ball then take long breaths 7 seconds in, hold for four seconds, 7 seconds out for about 5 times.
10. I guarantee the "ah-ha" moment is coming!

All the Deliverables for Homework

Add a new folder to your C200 folder named Assignment. In this new folder you will add the following Python files for Assignment4.

- Add the Project to your **C200-Assignments** Solution.
- Name the Project **Assignment4**. You will turn-in
  - **funtriangle.py**
  - **makeitrain.py**
  - **donor.py**
  - **palindrome.py**
  - **roman.py**
  - **moreloops.py**
  - **farm.py**
- Make sure and commit your project and modules by **11:00P Wednesday October 2th 2019**.

As always, all the work should be your own. You will complete this before Wednesday October 2, 2019 11:00P. You will submit your work by committing your code to your GitHub repository. You will *not* turn anything in on canvas. If your timestamp is 11:01P or greater, the homework cannot be graded. So do not wait until 10:58P to turn it in. For programs that do not explicitly have it, you **MUST** place the following at the end:

```python
if __name__ == "__main__":
     pass
```

There are other problems to do that aren't turned-in—these are sessions that you do with your computer and some questions to help improve your understanding of the overall homework.

Problem 1

In this problem, you'll use for-loops to draw triangles (shadows the great pyramids of Egypt. You can only use for and print. You'll need to use multiplication if you're looking for a hint. Here is →**NOT**← a solution:

```
for i in range(0,1):
    print("*")
    print("**")
    print("***")
    print("**")
    print("*"

#OUTPUT
*
**
***
**
*
```

You might need to use two for-loops, but you won't need to use three.

```
*
**
***
****
*****
******
*******
********
*********
**********
*********
********
*******
******
*****
****
***
**
*
```

```
*
***
******
**********
***************
*********************
****************************
************************************
*********************************************
************************************
****************************
*********************
***************
**********
******
***
*
```

```
*********************
 *******************
  *****************
   ***************
    *************
     ***********
      *********
       *******
        *****
         ***
          *
```

Deilverables Programming Problem 1: for loops

- Using only for-loops and print statements, recreate the three outputs above.
- You do not have use functions for this problem.
- Put all of these in a new module called **funtriangle.py**.

Problem 2: Make it Rain

Sometimes problems are a lot simpler than you initially imagine. Write a function that takes an amount in dollars and returns a list:

$$[q, d, n, p]$$

where q is the number of quarters, d is the number of dimes, n is the number of nickels, and p is the number of pennies. Your list should contain the fewest total coins possible that equals the dollar amount. Again, some of you will encounter slight rounding errors–your code is likely correct! It's because of the way the Python on your computer is implemented and its CPU.

Listing 1: makeitrain.py

```
# Input Parameter: dollar amount x
# Return Value: list of quarters, dimes, nickels, and pennies (in that ↩
    order)
def dollars(x):
    #TODO: implement this function

print(dollars(2.24))
print(dollars(1.19))
print(dollars(4.16))
```

Output

```
[8.0, 2.0, 0.0, 4.0]
[4.0, 1.0, 1.0, 4.0]
[16.0, 1.0, 1.0, 1.0]
```

Deliverables Programming Problem 2

- Complete the program.
- Put your code for this problem in a new module named **makeitrain.py**

Problem 3: Blood Drives

The table below shows the eight blood types and red blood cell compability. For example, someone with A+ can give blood to A+ or AB+. O- can donate to every type. The blood bank's stock in pints is shown as the rightmost column (this is an example blood bank–the one supplied to you for the problem has different amounts of units). I've included two dictionaries to help you with this code. Bank which has the units of blood on hand to donate and then a donor-recipient table. Because there's so much text scrolling down, I've imported os and use a method os.system("pause") which waits for the user to hit a key for the output to resume. Make sure you're in the active window when you hit the key.

Donor	Red Blood Cell Receiver								Stock (Pints)
	O-	O+	A-	A+	B-	B+	AB-	AB+	
O-	1	1	1	1	1	1	1	1	0
O+	0	1	0	1	0	1	0	1	2
A-	1	1	1	1	0	0	0	0	2
A+	0	0	0	1	0	0	0	1	4
B-	0	0	0	0	1	1	1	1	2
B+	0	0	0	0	0	1	0	1	5
AB-	0	0	0	0	0	0	1	1	2
AB+	0	0	0	0	0	0	0	1	2

We talked about the in predicate. Do this session to see how it works. It will prove useful to the homework.

Interactive Session

```
>>> x = (1,2,3)
>>> 1 in x
True
>>> 1 in (1,2,3)
True
>>> 5 in x
False
>>> x = [[1,2],3,"cat",["cat",(1,2,3)]]
>>> 3 in x
True
>>> "cat" in x
True
>>> 1 in x
False
>>> [1,2] in x
True
>>> ["cat"] in x
False
>>> ["cat", (1,2,4)] in x
False
>>> ["cat", (1,2,3)] in x
True
```

donor.py

```python
# allows to pause
import os

#blood bank type:units
bank = {"A+":5, "A-":6, "O+":4, "O-":2, "B+":5, "B-":6, "AB+":7, "AB-":8}
#blood donor:(recipient1, ..., recipientk)
donorReceiver = {
        "O-":("O-","O+","A+","A-","B+","B-","AB+","AB-"),
        "O+":("O+","A+","B+","AB+"),
        "A-":("O-","O+","A+","A-",),
        "A+":("A+","AB+"),
        "B+":("B-","B+","AB+","AB-"),
        "B-":("B-","B+","AB+","AB-"),
        "AB-":("AB+","AB-"),
        "AB+":("AB+") }

#INPUT donor’s blood type donorBloodType
```

```
#Return a tuple of the types that can accept the blood
def red_blood_compability(donorBloodType):
#TO DO: IMPLEMENT FUNCTION


# Show the current bank stock
def showBank():
    print("Current Blood Bank")
    print("{0:>4}  {1:>5}".format("Type","Units"))
    for k,v in bank.items():
        print("{0:>4}  {1:>4}".format(k,v))

#Show number of units of particular type in the bank
def showTypeInBank(bloodType):
    units = bank[bloodType]
    print("{0:>1} units of {1:>2} in bank".format(units, bloodType))

#INPUT
#3 Parameters donor blood type is donor, recipient’s type is recipient, ↩
    and pints is the
# number of pints that donor will give to recipient using the bank.
# Return 1  if the blood bank has enough pints to give
# and remove the amount of pints used from the bank
# Return 0  if either the recipient can’t recieve the type or there’s not ↩
    enough blood
def transfusion(donor, recipient, pints):
#TO DO: IMPLEMENT FUNCTION

###############################
#TESTING
#Shows current Bank
showBank()
if __name__ == "__main__":

    os.system("pause")


#Enough A+==5 units, and 4 can be given to AB+
#Result is A+==1
    if (transfusion("A+","AB+",4)):
            print("Transfusion successful")
    else:
            print("Transfusion failed")
    showTypeInBank("A+")
    os.system("pause")

#Fail because O+ cannot donate to B-
```

```
    if (transfusion("O+", "B-",1)):
            print("Transfusion successful")
    else:
            print("Transfusion failed")

#Fail because insufficient units of blood
    if (transfusion("A-","O-",7)):
            print("Transfusion successful")
    else:
            print("Transfusion failed")

#Succeed and AB-==0 at end
    if (transfusion("AB-","AB+",8)):
            print("Transfusion successful")
    else:
            print("Transfusion failed")
    showTypeInBank("AB-")
    os.system("pause")

    showBank()
```

Output to Problem 3

```
Current Blood Bank
Type  Units
  A+     5
  A-     6
  O+     4
  O-     2
  B+     5
  B-     6
 AB+     7
 AB-     8
Press any key to continue . . .
Transfusion successful
1 units of A+ in bank
Press any key to continue . . .
Transfusion failed
Transfusion failed
Transfusion successful
0 units of AB- in bank
Press any key to continue . . .
Current Blood Bank
Type  Units
  A+     1
  A-     6
  O+     4
  O-     2
  B+     5
  B-     6
 AB+     7
 AB-     0
```

Deliverables Programming Problem 3

- Complete the session.
- Complete program functions.
- The template with the dictionaries are in **donor.txt**
- Put the code in a new module named **donor.py**.

Problem 4: Palindrome

In this problem, you'll use a for loop to determine whether a string is a palindrome. A palindrome reads the same forwards as it does backwards (ignoring punctuation and spaces). You **cannot** use .reverse() or [::-1]. If you need to reverse the string, you are required to use a for loop.

String	Palindrome?
A man, a plan, a canal. Panama!	Yes
Rats live on no evil star.	Yes
Abba.	Yes
I love CS.	No
abcd.	No

The test cases have all punctuation and spaces removed as well as making all letters lowercase.

Listing 2: palindrome.py

```python
# Input Parameter: a string x
# Return Value: True if x is a palindrome, False otherwise
def palindrome(x):
    #TODO: implement this function

print(palindrome("aba"))
print(palindrome("a"))
print(palindrome("abba"))
print(palindrome("amanaplanacanalpanama"))
print(palindrome("abca"))
print(palindrome("ac"))
print(palindrome("adabbba"))
print(palindrome("amandaplanacanalpanama"))
```

Output

```
True
True
True
True
False
False
False
False
```

Deilverables Programming Problem 4

- Put your finished code in a new module **palindrome.py**

Problem 5: Roman Numeral Conversion

Western civilization used Roman numerals for over a millenium. We still use it, in fact, to mark special dates. You will write a conversion from decimal 1-99 to its Roman numeral counterpart. The numbers are shown below. Your task will be made easier if you use modulo and integer division shown in the session below. You are encouraged to add extra functions–likely at least two that deal with the values less than or equal to 9 and values that are multiples of tens.

Because students tend to make this much more difficult I am providing *hint* code and outout:

```
def roman(number):

    one = "l"
    five = "V"
    romanNumber = (number <= 3)*number*one + (number == 4)*(one + five) + ↩
       (number == 5)*five
    print("{0:<4} {1:<4}".format(number, romanNumber))

for number in range(1,6):
    roman(number)
```

```
1    l
2    ll
3    lll
4    lV
5    V
```

1	I	26	XXVI	51	LI	76	LXXVI
2	II	27	XXVII	52	LII	77	LXXVII
3	III	28	XXVIII	53	LIII	78	LXXVIII
4	IV	29	XXIX	54	LIV	79	LXXIX
5	V	30	XXX	55	LV	80	LXXX
6	VI	31	XXXI	56	LVI	81	LXXXI
7	VII	32	XXXII	57	LVII	82	LXXXII
8	VIII	33	XXXIII	58	LVIII	83	LXXXIII
9	IX	34	XXXIV	59	LIX	84	LXXXIV
10	X	35	XXXV	60	LX	85	LXXXV
11	XI	36	XXXVI	61	LXI	86	LXXXVI
12	XII	37	XXXVII	62	LXII	87	LXXXVII
13	XIII	38	XXXVIII	63	LXIII	88	LXXXVIII
14	XIV	39	XXXIX	64	LXIV	89	LXXXIX
15	XV	40	XL	65	LXV	90	XC
16	XVI	41	XLI	66	LXVI	91	XCI
17	XVII	42	XLII	67	LXVII	92	XCII
18	XVIII	43	XLIII	68	LXVIII	93	XCIII
19	XIX	44	XLIV	69	LXIX	94	XCIV
20	XX	45	XLV	70	LXX	95	XCV
21	XXI	46	XLVI	71	LXXI	96	XCVI
22	XXII	47	XLVII	72	LXXII	97	XCVII
23	XXIII	48	XLVIII	73	LXXIII	98	XCVIII
24	XXIV	49	XLIX	74	LXXIV	99	XCIX
25	XXV	50	L	75	LXXV	100	C

Interactive Session

```
// is the integer division operator
% is the modulo operator
>>> 15 // 10
1
>>> 30 // 10
3
>>> 35 // 10
3
>>> 30 % 10
0
>>> 15 % 10
5
>>> 11 // 10
1
>>> 12 // 10
1
>>> 13 // 10
1
>>> 13 % 10
3
>>> 27 % 20
7
```

Listing 3: roman.py

```python
# Input Parameter: a decimal value n
# Return Value: a string containing the Roman numeral representation of n
def roman(n):
    #TODO: implement this function

for i in range(1,100):
    if i % 5 == 0:
        print()
    print(i, roman(i), ", ", end="")
```

Output

```
1 I , 2 II , 3 III , 4 IV ,
5 V , 6 VI , 7 VII , 8 VIII , 9 IX ,
10 X , 11 XI , 12 XII , 13 XIII , 14 XIV ,
15 XV , 16 XVI , 17 XVII , 18 XVIII , 19 XIX ,
20 XX , 21 XXI , 22 XXII , 23 XXIII , 24 XXIV ,
25 XXV , 26 XXVI , 27 XXVII , 28 XXVIII , 29 XXIX ,
30 XXX , 31 XXXI , 32 XXXII , 33 XXXIII , 34 XXXIV ,
35 XXXV , 36 XXXVI , 37 XXXVII , 38 XXXVIII , 39 XXXIX ,
40 XL , 41 XLI , 42 XLII , 43 XLIII , 44 XLIV ,
45 XLV , 46 XLVI , 47 XLVII , 48 XLVIII , 49 XLIX ,
50 L , 51 LI , 52 LII , 53 LIII , 54 LIV ,
55 LV , 56 LVI , 57 LVII , 58 LVIII , 59 LIX ,
60 LX , 61 LXI , 62 LXII , 63 LXIII , 64 LXIV ,
65 LXV , 66 LXVI , 67 LXVII , 68 LXVIII , 69 LXIX ,
70 LXX , 71 LXXI , 72 LXXII , 73 LXXIII , 74 LXXIV ,
75 LXXV , 76 LXXVI , 77 LXXVII , 78 LXXVIII , 79 LXXIX ,
80 LXXX , 81 LXXXI , 82 LXXXII , 83 LXXXIII , 84 LXXXIV ,
85 LXXXV , 86 LXXXVI , 87 LXXXVII , 88 LXXXVIII , 89 LXXXIX ,
90 XC , 91 XCI , 92 XCII , 93 XCIII , 94 XCIV ,
95 XCV , 96 XCVI , 97 XCVII , 98 XCVIII , 99 XCIX ,
```

Observe the output is the same as in figure.

Deliverables Programming Problem 5

- Rum the small program and make sure you understand how it works.
- Complete the function–but the task will be considerably easier if you break it up into parts...one that deals with 0-9 and one that deals with 10, 20, 30,...90.
- Repeat the session to understand the modulo and integer division operators % and //
- Put your finished code in a new module **roman.py**

Problem 6

This sections helps you become proficient with loops. Each function is described and a template is provided. You are **NOT** allowed to use any pre-defined (built-in to Python) list operations for this problem. In the past, students have encountered problems, because they changed values of the variables in lines 48-54. The variables cannot be reassigned for the correct output to result.

Session

```
>>> x = "Hellokitty"
>>> y = list(x)
>>> y
['H', 'e', 'l', 'l', 'o', 'k', 'i', 't', 't', 'y']
>>> z = "".join(y)
>>> z
'Hellokitty'
>>> type("hi") is str
True
>>> type(3) is str
False
>>> type([2,"a"]) is list
True
```

Listing 4: moreloops.py

```
1  # Input Parameter: a list of, possibly empty, numbers x
2  # Returns: the max number in list x (does not have to be unique)
3  def maxFor(xlst):
4  #TODO: implement this function with a for loop
5
6  # Input Parameter: a list of, possibly empty, numbers x
7  # Returns: the max number in list x (does not have to be unique)
8  def maxWhile(xlst):
9  #TODO: implement this function with a while loop
10
11 # Input Parameter: a non-empty list of numbers x
12 # Returns: the minimal number value in list x (does not have to be unique)
13 def minFor(xlst):
14 #TODO: implement this function
15
16 #Input Parameters:a list of numbers lst
17 #Returns: the list lst with all occurrences of evens removed
18 def RemoveEvens(xlst):
19 #TODO: implement this function using for
20
```

```python
# Input Parameters: list that contains either integers or strings
# Return Value: oldX replaced with num copies newX
def myReplace(oldX, num, newX, xlst):
#TODO: implement this function

# Input Parameters: list of numbers
# Returns: sum of the odd numbers
# if there are no odd numbers, then print zero
# if the list is empty, print the empty list
def sumOdd(xlst):
#TODO: implement this function using while

# Input Parameter: a list x of objects [x1, x2, x3,..., xn]
# Returns: a string that is the concatenation of all the strings
# in the list in order encountered
def StringConcat(xlst):
#TODO: implement this function

# Data
w = []
x = [1,42,24,22,61,100,0,42]
y = [2]
z = [555,333,222]
nlst = [w,x,y,z]
c = [0,1,1,0,2,1,4]
a = ["a","b","b", "a", "c","b","e"]
b = [1,1,2,1,1,2,1,1,2,1,3,1]
d = ["a",1,"row",0,3,"d","ef",453]

print("maxFor_____")
for i in nlst:
    print(maxFor(i))
print("\nmaxWhile_____")
for i in nlst:
    print(maxWhile(i))
print("\nminFor_____")
for i in nlst:
    print(minFor(i))
print("\nRemoveEvens_____")
print(RemoveEvens(b))
print(RemoveEvens(c))
print("\nmyReplace_____")
print(myReplace(1,2,"dog",c))
print(myReplace(1,2,1,b))
print("\nsumOdd_____")
for i in nlst:
    print(sumOdd(i))
print("\nStringConcat_____")
print(StringConcat(a))
print(StringConcat(d))


if __name__ == "__main__":
    pass
```

Output

```
maxFor_____
[]
100
2
555

maxWhile_____
[]
100
2
555

minFor_____
[]
0
2
222

RemoveEvens_____
[1, 1, 1, 1, 1, 1, 1, 3, 1]
[1, 1, 1]

myReplace_____
[0, ’dog’, ’dog’, ’dog’, ’dog’, 0, 2, ’dog’, ’dog’, 4]
[1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 3, 1, 1]

sumOdd_____
[]
62
0
888

StringConcat_____
abbacbe
arowdef
```

Deliverables Programming Problem 5

- The starting code is in the text file **bestloops.txt**
- Do the sessions to help familiarize you with some helpful functions.
- Complete the functions in listing 5.
- Put your finished code in a new module **moreloops.py**

Problem 7

In this problem, you'll use brute force search to find an optimal answer. Suppose you've been tasked to design a normal, rectangular building, unlike Luddy, that has a loss of heat given by:

$$f(x, y, z) \quad = \quad 11xy + 14yz + 15xz$$

where the variables x, y, z are the three dimensions. We want to have a building with minimal heat loss. Suppose the volume of the building is $147,840\text{ft}^3$; in other words:

$$xyz \quad = \quad 147840$$

Although we can solve this using calculus, we prefer search! We want to find values for each variable from zero to 100 inclusive that minimizes the heat loss that equals the volume.

Listing 5: farm.py

```python
def f(x,y,z):
#TO DO: IMPLEMENT FUNCTION

def vol(x,y,z):
    return x*y*z == 147840

#some arbitrary starting point
a,b,c = 2,2,36960

#TO DO: LOOPS SEARCHING THROUGH POSSIBLE
#VALUES

print("H  W  L  Value")
print(a,b,c,f(a,b,c))

if __name__ == "main":
    pass
```

Output

```
H  W  L  Value
56 60 44 110880
```

Deliverables Programming Problem 7

- Complete the program.
- *hint: this is much simpler than you might think–review the slides...*
- Put your code for this problem in a new module named **farm.py**