Computer Science 1 — CSci 1100 — Fall 2016
Exam 2
March 27, 2017

SOLUTIONS

1. (**16 points**) Show the output of the following:

Code:	Solution:

# Part (a) 	
`for i in range(33, -7, -5):`
`  if i % 3 == 0:`
`  print("Yes", i)`
`  elif i // 23 == 0:`
`  break`
`  elif i % 4 == 0:`
`  continue`
`  else:`
`  print("No", i)`
`  print("Looping", i)`
	`Yes 33`
	`Looping 33`
	`No 23`
	`Looping 23`
	`Yes 18`
	`Looping 18`

# Part (b) 	
`t = (8, 4, 'Pelican')`
`s = "Partridge"`
`l = ["Peregrine Falcon", 71.1]`
`u = t`
`r = s`
`m = l`
`n = l[:]`
`m[1] = "Peacock"`
`s = r.replace("Part", "Whole")`
`t = t + (4, 9)`
`print(r)`
`print(l)`
`print(n)`
`print(u)`
	`Partridge`
	`['Peregrine Falcon', 'Peacock']`
	`['Peregrine Falcon', 71.1]`
	`(8, 4, 'Pelican')`

# Part (c) 	
`def ping(a, b):`
`  print("Limit:", b)`
`  return a > b`
`def pong(a, b):`
`  print("Val:", a)`
`  return a < b`
`a, b = 50, 51`
`v1 = ping(37, 101)`
`v2 = pong(37, 101)`
`print(ping(b, a) or pong(b, a))`
`print(v1 and v2)`
	`Limit: 101`
	`Val: 37`
	`Limit: 50`
	`True`
	`False`

# Part (d) 	
`l = [list(range(2, 12, 2)), list(range(3, 15, 3)),`
`  [64, "whee", 12]]`
`print(len(l))`
`print(len(l[0]), len(l[1]), len(l[2]))`
`print(l[1])`
`print(l[0][3])`
	`3`
	`5 4 3`
	`[3, 6, 9, 12]`
	`8`

2. (**16 points**) For each of the following write a **single line of Python code** to solve the problem. You can assume you are typing the command into the Python Shell. You must only use techniques we have covered thus far in the course and may not use loops. Any use of a `for` or a `while` will result in a 0 for that answer.

Code:	**Solution:**
(a) Write an expression to generate a list containing the values: `[-37, -44, -51, -58, -65, -72]`	`list(range(-37, -73, -7))` where the second number can be any value from 73 up to and including 79.

(b) Given two lists `u` and `v`, write an expression to create a new list associated with the variable `x` that has the even values of `u`, `[u[0],u[2],u[4],...]` followed by the odd members from the first half of `v`, for example, assuming the length of `v` is 10, this would be `[v[1],v[3],v[5]]`. Make your solution work for any `u` or `v`, but as an example, if
`u = [0, 1, 2, 3, 4, 5, 6]`
`v = ["a","b","c","d","e","f","g","h","i","j",`
`"k","l","m","n","o","p","q","r","s","t",`
`"u","v","w","x","y","z"]`
then `x` would be refer to the list
`[0, 2, 4, 6, 'b', 'd', 'f', 'h', 'j', 'l']`
	The two examples in the question conflict. We will take either:
	`x = u[::2] + v[1:len(v)//2:2]`
	or
	`x2 = u2[::2] + v2[1:len(v2)//2+1:2]`

(c) Given a list, `v1` and a string `s2`, write an expression that evaluates to `True` if the string `"a"` occurs at an earlier index in `v1` than in `s2`. You can assume that "a" occurs in both the list and the string. For example, the expression should evaluate to `True` if
`v1 = ['a', -9.3, 8.5, 3.0, 'a', ['t', 'u']]`
`s2 = "cdagjdgh"`
and `False` if
`v1 = [-9.3, 8.5, 3.0, 'a', ['t', 'u']]`
`s2 = "cdagjdgh"`
	`v1.index('a') < s2.find('a')`
	or
	`(v1.index("a") < s2.find("a")) or \`
	`(s2.find("a") < 0)`
	or
	`("a" in v1) and \`
	`(v1.index("a") < s2.find("a")) or \`
	`(s2.find("a") < 0)`

(d) Given a sorted list `l` where each value can appear multiple times, write an expression to find the index of the first occurrence of the third value in the sequence. For example, if `l = [1, 1, 1, 1, 1, 3, 3, 3, 7, 7, 9, 9]` your expression should return the index of the first occurrence of `7` which is 8.	`l.count(l[0])+l.count(l[l.count(l[0])])`

3. (**16 points**) Write a **while** loop to repeatedly ask for an integer between `0` and `9`. Keep count of how many times each value occurs in a **list** and end the loop when any value reaches 3 occurrences, at which point you should print out the list of occurrences. You can assume that the user always enters an integer value, but must verify that the value lies within the correct range. If an invalid value is entered, simply ignore it and go to the next input.

   Here is an example.

```
Enter a digit (0-9): 4
Enter a digit (0-9): 9
Enter a digit (0-9): 1
Enter a digit (0-9): 2
Enter a digit (0-9): -1
Enter a digit (0-9): 10
Enter a digit (0-9): 6
Enter a digit (0-9): 1
Enter a digit (0-9): 1
[0, 3, 1, 0, 1, 0, 1, 0, 0, 1]
```

   Solution:

```
occurrences = 10*[0]
while True:
    digit = int(input("Enter a digit (0-9): "))
    if 0 <= digit <= 9:
        occurrences[digit] += 1
    else:
        continue
    if max(occurrences) == 3:
        break
print(occurrences)count = 0
```

   or

```
l = [0]*10
while max(l) < 3:
    val = int(input("Enter a digit (0-9): "))
    if 0 <= val <= 9:
        l[val] += 1
print(l)
```

4. (**16 points**) We explored truth tables briefly during our discussions of booleans and decisions. A truth table is just a specification of the inputs to a boolean function and the outputs. This questions asks you to write code to generate an **XOR** truth table as follows.

   Part a: Write a function called `xor` that takes two booleans as input and returns a boolean representing the output of the `xor`. The **XOR** function is similar to an **OR** function, except that `a xor b` returns `False` when both `a` and `b` are `True`. See the example below for more details. Full credit requires that you do not use an `if` statement in your function, but you will receive partial credit for a solution that uses an `if`.

```
>>> xor(True, True)
False
>>> xor(True, False)
True
>>> xor(False, True)
True
>>> xor(False, False)
False
```

   In creating your function remember precedence. The `and` has higher precedence than the `or`. **Solution:**

```
def xor(a, b):
    return (a or b) and not (a and b)
```

   or using the bitwise xor

```
def xor(a, b):
    return (a ^ b)
```

   or

```
def xor(a,b):
    return a != b
```

   or, with using an `if` for partial credit:

```
def xor(a, b):
    if a:
        return not b
    else:
        return b
```

   or, again for partial credit:

```
def xor(a, b):
    if a and b:
        return False
    elif a or b:
        return True
    else:
        return False
```

Part b: Now write a program that **calls the function you just wrote** to create the **XOR** table. You will not receive credit for solutions that do not use the `xor` function. The code should print the following:

```
Truth Table for XOR:
a     b     a xor b
True  True  False
True  False True
False True  True
False False False
```

Note that there are tab characters between each column.

Solution:

```
print("Truth Table for XOR:")
print("{}\t{}\t{}".format("a", "b", "a xor b"))
print("{}\t{}\t{}".format(True, True, xor(True, True)))
print("{}\t{}\t{}".format(True, False, xor(True, False)))
print("{}\t{}\t{}".format(False, True, xor(False, True)))
print("{}\t{}\t{}".format(False, False, xor(False, False)))
```

or using loops:

```
values = [True, False]
print("Truth Table for XOR:")
print("{}\t{}\t{}".format("a", "b", "a xor b"))
for a in values:
    for b in values:
        print("{}\t{}\t{}".format(a, b, xor(a, b)))
```

5. (**20 points**) This question is in two parts. Consider a simplified version of our yelp data containing only the restaurant name, the type, and the reviews. An excerpt from the file is:

```
Meka's Lounge   |   Bars|5|2|4|4|3|4|5
    Tosca Grille|American (New)|1|3|2|4
Happy Lunch|American (Traditional)| 5  | 2
Hoosick Street Discount Beverage Center|  Beer, Wine & Spirits|4|5|5|5|5|4
Uncle Ricky's Bagel Heaven|Bagels| 5|3| 5|4|3|5|4
Confectionery House|Candy   |5|4
Uncle John Diner|Diners|4|5|5
China Wok| Chinese |2|1
...
```

Part a: Write a function called `parse_line` that takes a string representing a line from the file with `|` separating the different parts of the line. The function should return a list with 4 entries representing the number of reviews, the average review, the restaurant name, and the type. You can assume there is always at least one review. Be careful of extra white space.

```
print(parse_line("Meka's Lounge   |   Bars|5|2|4|4|3|4|5"))
print(parse_line("    Tosca Grille|American (New)|1|3|2|4"))
print(parse_line("Happy Lunch|American (Traditional)| 5  | 2"))
```

should be

```
[7, 3.857142857142857, "Meka's Lounge", 'Bars']
[4, 2.5, 'Tosca Grille', 'American (New)']
[2, 3.5, 'Happy Lunch', 'American (Traditional)']
```

Solution:

```
def parse_line(string):
    parsed = string.split('|')
    average = 0
    total = len(parsed)-2
    for i in range(2,len(parsed)):
        average += int(parsed[i].strip())
    average /= total
    return [total, average, parsed[0].strip(), parsed[1].strip()]
```

or

```
def parse_line(string):
l = []
line = string.split('|')
index = 2
total= 0
while index < len(line):
    total += int(line[index].strip())
    index += 1
num = len(line) - 2
l.append(num)
l.append(total / num)
l.append(line[0].strip())
l.append(line[1].strip())
return l
```

Part b: Write Python code that assumes the `parse_line` function from `Part a` is written and correct. (You should **not** rewrite the function here.) Your code should ask the user for the name of an input file that contains the simplified restaurant records, and for the name of an output file. Your code should then read each line of the input file, call `parse_line`, and write out to the output file the name of any restaurant with both more than 3 reviews and an average review greater than 2.5. You can assume the input file exists.

For example if the input file contains:

```
Meka's Lounge   |   Bars|5|2|4|4|3|4|5
    Tosca Grille|American (New)|1|3|2|4
Happy Lunch|American (Traditional)| 5  | 2
Hoosick Street Discount Beverage Center|  Beer, Wine & Spirits|4|5|5|5|5|4
Uncle Ricky's Bagel Heaven|Bagels| 5|3| 5|4|3|5|4
Confectionery House|Candy   |5|4
Uncle John Diner|Diners|4|5|5
China Wok| Chinese |2|1
```

then after your program runs, the output file should read:

```
Meka's Lounge
Hoosick Street Discount Beverage Center
Uncle Ricky's Bagel Heaven
```

Solution:

```
infile = input("Input Filename? ")
outfile = input("Output Filename? ")
lines = []
file = open(outfile, "w")
for line in open(infile):
    parsed = parse_line(line)
    if parsed[0] > 3 and parsed[1] > 2.5:
        file.write(parsed[2]+'\n')
file.close()
```

6. (**16 points**) Given a list of lists representing a chess board with pieces, write a function `move_piece` that takes `4` arguments:

   - **board** - a list of lists having the position of pieces
   - **piece** - a string representing the piece name
   - **starting** - a (row, column) tuple representing the starting position of the piece
   - **ending** - a (row, column) tuple representing the ending position of the piece

   When called, the function checks to see if the `piece` is currently at the `starting` position in the board.

   (a) If the piece is at the `starting` position,
   - The `starting` position is marked as empty (replaced with `'.'`),
   - The piece is moved to the `ending` position
   - Whatever was stored at the ending position, either an empty space `'.'` or a piece name is returned by the function.

   (b) If the piece is **not** at the `starting` position,
   - The function does not return a value.

   Here is an example of a starting board.

```
[['R', 'Kn', 'B', 'Q', 'K', 'B', 'Kn', 'R']
 ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P']
 ['R', 'Kn', 'B', 'Q', 'K', 'B', 'Kn', 'R']]
```

   And here are some sample calls. Note that you do not need to check that the rules of chess are followed.

```
>>> print(move_piece(board, 'P', (6, 3), (4, 3)))
.
>>> print(move_piece(board, 'Q', (7, 3), (0, 0)))
R
>>> print(move_piece(board, 'P', (4, 4), (4, 2)))
None
```

   At the end of the sequence above, here is the final board.

```
[['Q', 'Kn', 'B', 'Q', 'K', 'B', 'Kn', 'R']
 ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['.', '.', '.', 'P', '.', '.', '.', '.']
 ['.', '.', '.', '.', '.', '.', '.', '.']
 ['P', 'P', 'P', '.', 'P', 'P', 'P', 'P']
 ['R', 'Kn', 'B', '.', 'K', 'B', 'Kn', 'R']]
```

WRITE YOUR SOLUTION IN THE BOX ON THE NEXT PAGE!

Solution:

```
def move_piece(board, piece, start, end):
    if board[start[0]][start[1]] == piece:
        board[start[0]][start[1]] = '.'
        taken = board[end[0]][end[1]]
        board[end[0]][end[1]] = piece
        return taken
    return
```