CS 111 Spring 2023 Midterm 1: Solutions for Review Problems

Problem 1: Mystery while loop

Study the **`mystery`** function below, which uses the provided **`isVowel`** function.

```
def isVowel(char):
    return len(char) == 1 and char.lower() in 'aeiou'

def mystery(word, bound):
    """Docstring withheld."""
    result = ''
    i = 0

    while len(result) < bound and i < len(word):

        if (not isVowel(word[i])) and word[i] not in result:
            result += word[i]

        i += 1

    if result == '':
        return 'No result'

    return result
```

Predict the outcome of the following invocations of the **`mystery`** function:

Function call	Value returned by function call
`mystery('coconut', 1)`	`'c'`
`mystery('coconut', 4)`	`'cnt'`
`mystery('apple', 2)`	`'pl'`
`mystery('oooooh', 2)`	`'h'`

Problem 2: List processing

Below define a function `check` that takes two parameters: 1) a word and 2) a list of words and **returns** the list containing all the words that are alphabetically before the given word.

Here are some example calls of this function and their expected results.

Function call	Value returned by function call
`check('candy', ['bear', 'apple', 'donut', 'cave'])`	`['bear', 'apple']`
`check('cook', ['bear', 'apple', 'donut', 'cave'])`	`['bear', 'apple', 'cave']`
`check('egg', ['bear', 'apple', 'donut', 'cave'])`	`['bear', 'apple', 'donut', 'cave']`
`check('ant', ['bear', 'apple', 'donut', 'cave'])`	`[]`
`check('best', ['baby', 'butter', 'bear', 'beast', 'boo'])`	`['baby', 'bear', 'beast']`

```
# Type your code inside the box

def check(pivot, wordlist):
    result = []
    for word in wordlist:
        if word < pivot:
            result.append(word)
    return result
```

Problem 3: Loop with conditionals

Below define a function `pigLatin` that accepts a *list* of words and returns a *list* of those same words translated into “Pig Latin.” "Pig Latin" is a made-up language that involves shifting letters of a word around and appending the sound "ay."

Here are our rules for this language:

- Words that are shorter than 3 characters are left as is e.g. `'an' => 'an'`

- Words that begin with a consonant shift the first letter to the end and append `'ay'` e.g. `'hello' => 'ellohay'`
- Words that begin with vowels get `'ay'` appended e.g. `'apple' => 'appleay'`

Here are some example calls of this function and their expected results

Function call	**Result**
`pigLatin(['this','is','a','great', 'example'])`	`['histay', 'is', 'a', 'reatgay', 'exampleay']`
`pigLatin(['is'])`	`['is']`
`pigLatin(['great'])`	`['reatgay']`
`pigLatin(['example'])`	`['exampleay']`

Complete the definition of the **`pigLatin`** function below. Your function must use either a **`for`** loop or a **`while`** loop. You may use **`isVowel`** or other helper functions, though you don't need to.

(Please keep all your code within the box)

```
def pigLatin(words):

    results = []                # accumulator variable
    for word in words:
        if(len(word) < 3):
            results.append(word)
        elif(word[0] in 'aeiou'):
            results.append(word + 'ay')
        else:
            results.append(word[1:] + word[0] + 'ay')

    return results
```

Problem 4: Understanding conditionals

In the table below, show what is printed for various calls of this **`analyze`** function:

```
def analyze(word):
    if len(word) <= 4:
        print('S')
    else:
        print('L')
    if isVowel(word[0]):
        print('V0')
        if not isVowel(word[1]):
            print('C1')
    elif isVowel(word[1]):
        print('V1')
    else:
        print('C01')
    if isVowel(word[-1]): # last letter of word
        print('VU')
        if not isVowel(word[-2]): # next to last letter of word
            print('CP')

def isVowel(char):
    return char.lower() in 'aeiou'
```

Function call	Printed Output		Function call	Printed Output
`analyze('cat')`	S V1		`analyze('spree')`	L C01 VU
`analyze('oats')`	S V0		`analyze('apple')`	L V0 C1 VU CP

Problem 5: Printing Time [Loop with Conditionals & Boolean Expressions

On the next page, define a function **printTime** that takes three arguments:

1. **`day`**: a day of the week, which is one of the strings **`'Sun'`**, **`'Mon'`**, **`'Tue'`**, **`'Wed'`**, **`'Thu'`**, **`'Fri'`**, **`'Sat'`**
2. **`hour`**: an integer between 1 and 12, inclusive
3. **`ampm`**: one of the strings **`'AM'`** or **`'PM'`**

printTime prints *exactly one word* as specified below. It does not return anything.

- For a weekend day (Sat or Sun), it prints **`weekend`**.
- For a weekday (Mon through Fri):
  - It prints **`evening`** from 5PM up to and including 11PM
  - It prints **`sleep`** from midnight (12AM) up to and including 8AM.
    Note that midnight is considered the beginning of a new day, not the end of a previous day.
  - It prints **`class`** for all other times — i.e., from 9AM up to and including 4PM.
    This range includes noon (12PM).

Here are some examples:

Function call	Printed Output		Function call	Printed Output
`printTime('Sat',12,'AM')`	`weekend`		`printTime('Mon',12,'AM')`	`sleep`
`printTime('Sat',10,'AM')`	`weekend`		`printTime('Wed',3,'AM')`	`sleep`
`printTime('Sun',11,'PM')`	`weekend`		`printTime('Fri',8,'AM')`	`sleep`
`printTime('Mon',5,'PM')`	`evening`		`printTime('Tue',9,'AM')`	`class`
`printTime('Thu',8,'PM')`	`evening`		`printTime('Wed',12,'PM')`	`class`
`printTime('Fri',11,'PM')`	`evening`		`printTime('Thu',4,'PM')`	`class`

In your definition you do **not** need to handle cases where an input is an unexpected value (e.g., an invalid day or ampm string or an hour that is not an integer in the range 1 to 12 inclusive).

(Please keep all your code within the box)

```
def printTime(day, hour, ampm):
    if day in ["Sat", "Sun"]:
        print("weekend")
```

```
    elif ampm == "PM" and 5 <= hour and hour <= 11:
        print("evening")
    elif ampm == "AM" and (hour == 12 or hour <= 8): # 12AM is special case
        print("sleep")
    else:
        # Although it's not needed (since ELSE catches everything else)
        # we could use this explicit test instead for this case:
        # ((ampm = "AM" and 9 <= hour <= 11)
        #  or (ampm = "PM" and (hour == 12 or hour <= 4)))
        print("class")
```

Problem 6: Strings & Loops

Define a function **block**(width, string) that prints a string with width characters per line. Below are some sample invocations. Hint: you might find the function range() and slicing helpful.

block(4,'abcdefghijklmnopqrstuvwxyz')	abcd efgh ijkl mnop qrst uvwx yz
block(10,'abcdefghijklmnopqrstuvwxyz')	abcdefghij klmnopqrst uvwxyz
block(3,'THANK YOU')	THA NK YOU

Write your **block** function here (keep all code within the box below):

Solution Nr. 1

```
def block(width, string):
    for x in range(0, len(string), width):
        print(string[x:x+width])
```

Solution Nr. 2:

```
# version using while loop and only simple slicing
def block(width, string):
     while string != '':
         print(string[0:width])
         string = string[width:]
```

Broken Solution Nr. 3:

```
# This doesn't work because the range for x is too large,
# causing many blank lines to be printed at the end
def block(width, string):
    for x in range(len(string)):
        print(string[width*x:(x+1)*width])
```

Corrected Solution Nr. 3

```
def block(width, string):
    for i in range(((len(string)-1)//width)+1):  # int division //
        print(string[width*i:width*(i+1)])

# The formula for the end argument to the range requires explanation.
# Consider a concrete example when the width is 10.
# For len 0 we want there to be no index i, so end of range must be 0
# For len 1 to 10, want the last i to be 0, so end of range must be 1
# For len 11 to 20, want the last i to be 1, so end must be 2
# For len 21 to 30, want the last i to be 2, so end must be 3
# ... and so on ...
# Generalizing the above, the formula for the end of range is:
# ((len-1)/width)+1
```

Problem 7: Iteration Table (old quiz problem)

For the following function:
 	
`def divisibleBy(stop, el):`
`    divList = []`
	The iteration table contains all variables used in the problem, although not all of them change.

```
    i = 0
    while i < stop:
        if i % el == 0:
            divList.append(i)
        i += 1
    return divList
```

In the box at right, write the iteration table that captures how its state variables change for the function call:

```
divisibleBy(9, 3)
```

stop	el	i	divList
9	3	0	[]
9	3	1	[0]
9	3	2	[0]
9	3	3	[0]
9	3	4	[0, 3]
9	3	5	[0, 3]
9	3	6	[0, 3]
9	3	7	[0, 3, 6]
9	3	8	[0, 3, 6]
9	3	9	[0, 3, 6]

Problem 8: Selective Summing [Challenging]

Below define a function `sum78` that takes a list of numbers and returns the sum of the numbers in the list, ignoring sections of numbers starting with a 7 and extending to the next 8 (or to the end of the list, if there is no corresponding 8). Return 0 when no numbers are summed.

Here are some example calls of this function and their expected results. Numbers with a gray background are ignored.

Function call	Value returned by function call
`sum78([1, 4, 2])`	`7 = 1 + 4 + 2`
`sum78([1, 4, 2, 7, 77, 54, 8, 5])`	`12 = 1 + 4 + 2 + 5`

sum78([1, **7, 17, 8,** 2, **7, 23, 42, 8,** 3, **7, 91, 8,** 4])	10 # 1 + 2 + 3 + 4
sum78([9, **7, 2, 7, 2, 8**, 3, 4])	16 # 9 + 3 + 4
sum78([4, 1, **7, 2, 7, 2, 8**, 5, 2, **7, 10, 20, 30**])	12 # 4 + 1 + 5 + 2
sum78([**7, 6, 1, 6, 8**])	0

(Please keep all your code within the box)

```
def sum78(nums):
    sumSoFar = 0
    ignoreMode = False # Initially not ignoring numbers
    for num in nums:
        if ignoreMode:
            if num == 8:
                ignoreMode = False # Stop ignoreMode
        elif num == 7:
            ignoreMode = True      # Start ignoreMode
        else: # Not in ignoreMode, so summing numbers
            sumSoFar += num
    return sumSoFar
```