Problem 1: Mystery while loop

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

```python
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:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned by function call</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery('coconut', 1)</td>
<td>'c'</td>
</tr>
<tr>
<td>mystery('coconut', 4)</td>
<td>'cnt'</td>
</tr>
<tr>
<td>mystery('apple', 2)</td>
<td>'pl'</td>
</tr>
<tr>
<td>mystery('oooooh', 2)</td>
<td>'h'</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned by function call</th>
</tr>
</thead>
<tbody>
<tr>
<td>check('candy', ['bear', 'apple', 'donut', 'cave'])</td>
<td>['bear', 'apple']</td>
</tr>
<tr>
<td>check('cook', ['bear', 'apple', 'donut', 'cave'])</td>
<td>['bear', 'apple', 'cave']</td>
</tr>
<tr>
<td>check('egg', ['bear', 'apple', 'donut', 'cave'])</td>
<td>['bear', 'apple', 'donut', 'cave']</td>
</tr>
<tr>
<td>check('ant', ['bear', 'apple', 'donut', 'cave'])</td>
<td>[]</td>
</tr>
<tr>
<td>check('best', ['baby', 'butter', 'bear', 'beast', 'boo'])</td>
<td>['baby', 'bear', 'beast']</td>
</tr>
</tbody>
</table>

# Type your code inside the box

```python
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:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>pigLatin(['this', 'is', 'a', 'great',</td>
<td></td>
</tr>
<tr>
<td>'example'])</td>
<td>['histay', 'is', 'a', 'reatgay',</td>
</tr>
<tr>
<td></td>
<td>'exampleay']</td>
</tr>
<tr>
<td>pigLatin(['is'])</td>
<td>['is']</td>
</tr>
<tr>
<td>pigLatin(['great'])</td>
<td>['reatgay']</td>
</tr>
<tr>
<td>pigLatin(['example'])</td>
<td>['exampleay']</td>
</tr>
</tbody>
</table>

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.

```python
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:
```python
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')
        else:
            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'
```

<table>
<thead>
<tr>
<th>Function call</th>
<th>Printed Output</th>
<th>Function call</th>
<th>Printed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyze('cat')</td>
<td>S V1</td>
<td>analyze('spree')</td>
<td>L C01 VU</td>
</tr>
<tr>
<td>analyze('oats')</td>
<td>S V0</td>
<td>analyze('apple')</td>
<td>L V0 C1 VU CP</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Printed Output</th>
<th>Function call</th>
<th>Printed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>printTime('Sat',12,'AM')</td>
<td>weekend</td>
<td>printTime('Mon',12,'AM')</td>
<td>sleep</td>
</tr>
<tr>
<td>printTime('Sat',10,'AM')</td>
<td>weekend</td>
<td>printTime('Wed',3,'AM')</td>
<td>sleep</td>
</tr>
<tr>
<td>printTime('Sun',11,'PM')</td>
<td>weekend</td>
<td>printTime('Fri',8,'AM')</td>
<td>sleep</td>
</tr>
<tr>
<td>printTime('Mon',5,'PM')</td>
<td>evening</td>
<td>printTime('Tue',9,'AM')</td>
<td>class</td>
</tr>
<tr>
<td>printTime('Thu',8,'PM')</td>
<td>evening</td>
<td>printTime('Wed',12,'PM')</td>
<td>class</td>
</tr>
<tr>
<td>printTime('Fri',11,'PM')</td>
<td>evening</td>
<td>printTime('Thu',4,'PM')</td>
<td>class</td>
</tr>
</tbody>
</table>

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)

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

```
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

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

Solution Nr. 2:

```python
# 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:

```python
# 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

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

Problem 7: Iteration Table (old quiz problem)

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

<table>
<thead>
<tr>
<th>stop</th>
<th>el</th>
<th>i</th>
<th>divList</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>[]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>[0]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>[0]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>[0]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
<td>[0, 3]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>[0, 3]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>6</td>
<td>[0, 3]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>7</td>
<td>[0, 3, 6]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>8</td>
<td>[0, 3, 6]</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>9</td>
<td>[0, 3, 6]</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned by function call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum78([1, 4, 2])</code></td>
<td>7 = 1 + 4 + 2</td>
</tr>
<tr>
<td><code>sum78([1, 4, 2, 7, 77, 54, 8, 5])</code></td>
<td>12 = 1 + 4 + 2 + 5</td>
</tr>
</tbody>
</table>
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