1. Rondo Form is a type of musical structure, in which there is a recurring theme/refrain, notated as A. Here are the rules for valid rondo forms:

Rondo forms always start and end with an A section.
In between the A sections, there should be contrasting sections notated as B, then C, then D, etc... No letter should be skipped.
There shouldn't be any repeats in the sequence (such as ABBACCA).
Create a function which validates whether a given string is a valid Rondo Form.

Examples:
valid_rondo("ABACADAEAFAGAHAIAJA") → True
valid_rondo("ABA") → True
valid_rondo("ABBACCA") → False
valid_rondo("ACAC") → False
valid_rondo("A") → False

Ans:

In [3]:

```
def valid_rondo(in_string):
    string_list = list(in_string.replace('A',''))
    output = False
    if len(string_list) == 1:
            output= True
    else:
        for ele in range(len(string_list)-1):
            if ord(string_list[ele]) < ord(string_list[ele+1]):
                output= True
            else:
                output = False
                break
    print(f"valid_rondo({in_string}) → {output}")
valid_rondo("ABACADAEAFAGAHAIAJA")
valid_rondo("ABA")
valid_rondo("ABBACCA")
valid_rondo("ACAC")
valid_rondo("A")
```

```
valid_rondo(ABACADAEAFAGAHAIAJA) → True
valid_rondo(ABA) → True
valid_rondo(ABBACCA) → False
valid_rondo(ACAC) → False
valid_rondo(A) → False
```

2. Create a function that returns the whole of the first sentence which contains a specific word. Include the full stop at the end of the sentence.

Examples:
txt = "I have a cat. I have a mat. Things are going swell."
sentence_searcher(txt, "have") → "I have a cat."
sentence_searcher(txt, "MAT") → "I have a mat."
sentence_searcher(txt, "things") → "Things are going swell."
sentence_searcher(txt, "flat") → ""

Ans:

In [5]:
```
def sentence_searcher(in_string, text):
    output = '""'
    for ele in in_string.split(". "):
        if len(ele.lower().replace(text.lower(),'')) != len(ele):
            output = ele
            break
    print(f'sentence_searcher{in_string,text} ⟶ {output}')
txt = "I have a cat. I have a mat. Things are going swell."
sentence_searcher(txt, "have")
sentence_searcher(txt, "MAT")
sentence_searcher(txt, "things")
sentence_searcher(txt, "flat")
```

```
sentence_searcher('I have a cat. I have a mat. Things are going swell.', 'have') ⟶ I have a cat
sentence_searcher('I have a cat. I have a mat. Things are going swell.', 'MAT') ⟶ I have a mat
sentence_searcher('I have a cat. I have a mat. Things are going swell.', 'things') ⟶ Things are going swell.
sentence_searcher('I have a cat. I have a mat. Things are going swell.', 'flat') ⟶ ""
```

3. Given a number, find the "round "of each digit of the number. An integer is called "round" if all its digits except the leftmost (most significant) are equal to zero.

- Round numbers: 4000, 1, 9, 800, 90
- Not round numbers: 110, 707, 222, 1001

Create a function that takes a number and returns the "round" of each digit (except if the digit is zero) as a string. Check out the following examples for more clarification.

Examples:
sum_round(101) ⟶ "1 100"
sum_round(1234) ⟶ "4 30 200 1000"
sum_round(54210) ⟶ "10 200 4000 50000"

Ans:

In [7]:
```
def sum_round(num):
    output = []
    num = str(num)
    for ele in range(len(num)):
        if num[ele] != '0':
            output.append(num[ele]+len(num[ele+1:])*'0')
    print(f'sum_round({num}) ⟶ {" ".join(output[::-1])}')
sum_round(101)
sum_round(1234)
sum_round(54210)
```

```
sum_round(101) ⟶ 1 100
sum_round(1234) ⟶ 4 30 200 1000
sum_round(54210) ⟶ 10 200 4000 50000
```

4. Your task, is to create N x N multiplication table, of size n provided in parameter.

For example, when n is 5, the multiplication table is:

1, 2, 3, 4, 5
2, 4, 6, 8, 10
3, 6, 9, 12, 15
4, 8, 12, 16, 20
5, 10, 15, 20, 25
This example will result in:[[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20], [5, 10, 15, 20, 25]]

Examples:
multiplication_table(1) → [[1]]
multiplication_table(3) → [[1, 2, 3], [2, 4, 6], [3, 6, 9]]

Ans:

In [8]:
```
def multiplication_table(num):
    output =[]
    for a in range(1,num+1):
        temp_list = []
        for b in range(1,num+1):
            temp_list.append(a*b)
        output.append(temp_list)
    print(f'multiplication_table({num}) → {output}')
multiplication_table(3)
multiplication_table(1)
multiplication_table(5)
```

```
multiplication_table(3) → [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
multiplication_table(1) → [[1]]
multiplication_table(5) → [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20], [5, 10, 15, 20, 25]]
```

5. Create a function that returns True if two lines rhyme and False otherwise. For the purposes of this exercise, two lines rhyme if the last word from each sentence contains the same vowels.

Examples:
does_rhyme("Sam I am!", "Green eggs and ham.") → True
does_rhyme("Sam I am!", "Green eggs and HAM.") → True
Capitalization and punctuation should not matter.
does_rhyme("You are off to the races", "a splendid day.") → False
does_rhyme("and frequently do?", "you gotta move.") → False

Ans:

In [10]:
```
def does_rhyme(string1,string2):
    vowels = 'aeiou'
    output= False
    rhyme1 = [x.lower() for x in string1.split(" ")[-1] if x.lower() in vowels]
    rhyme2 = [x.lower() for x in string2.split(" ")[-1] if x.lower() in vowels]
    if rhyme1 == rhyme2:
        output = True
    print(f'does_rhyme{string1,string2} → {output}')
does_rhyme("Sam I am!", "Green eggs and ham.")
does_rhyme("Sam I am!", "Green eggs and HAM.")
does_rhyme("You are off to the races", "a splendid day.")
does_rhyme("and frequently do?", "you gotta move.")
```

```
does_rhyme('Sam I am!', 'Green eggs and ham.') → True
does_rhyme('Sam I am!', 'Green eggs and HAM.') → True
does_rhyme('You are off to the races', 'a splendid day.') → False
does_rhyme('and frequently do?', 'you gotta move.') → False
```