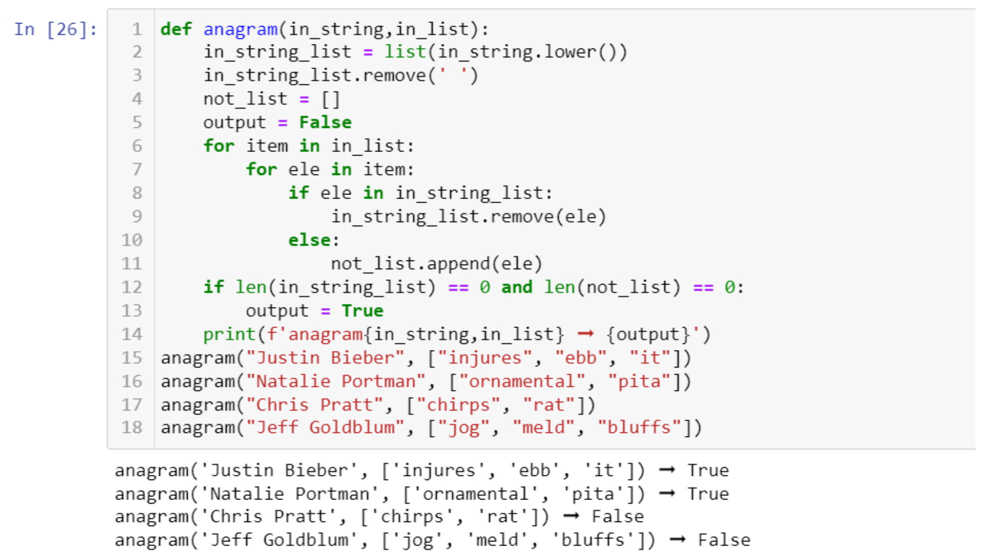
1. Write a function that returns True if a given name can generate an array of words.

Examples:
anagram("Justin Bieber", ["injures", "ebb", "it"]) ➞ True
anagram("Natalie Portman", ["ornamental", "pita"]) ➞ True
anagram("Chris Pratt", ["chirps", "rat"]) ➞ False
# Not all letters are used
anagram("Jeff Goldblum", ["jog", "meld", "bluffs"]) ➞ False
# "s" does not exist in the original name

Ans:



```
In [26]:
def anagram(in_string,in_list):
    in_string_list = list(in_string.lower())
    in_string_list.remove(' ')
    not_list = []
    output = False
    for item in in_list:
        for ele in item:
            if ele in in_string_list:
                in_string_list.remove(ele)
            else:
                not_list.append(ele)
    if len(in_string_list) == 0 and len(not_list) == 0:
        output = True
    print(f'anagram{in_string,in_list} ➞ {output}')
anagram("Justin Bieber", ["injures", "ebb", "it"])
anagram("Natalie Portman", ["ornamental", "pita"])
anagram("Chris Pratt", ["chirps", "rat"])
anagram("Jeff Goldblum", ["jog", "meld", "bluffs"])
```

```
anagram('Justin Bieber', ['injures', 'ebb', 'it']) ➞ True
anagram('Natalie Portman', ['ornamental', 'pita']) ➞ True
anagram('Chris Pratt', ['chirps', 'rat']) ➞ False
anagram('Jeff Goldblum', ['jog', 'meld', 'bluffs']) ➞ False
```

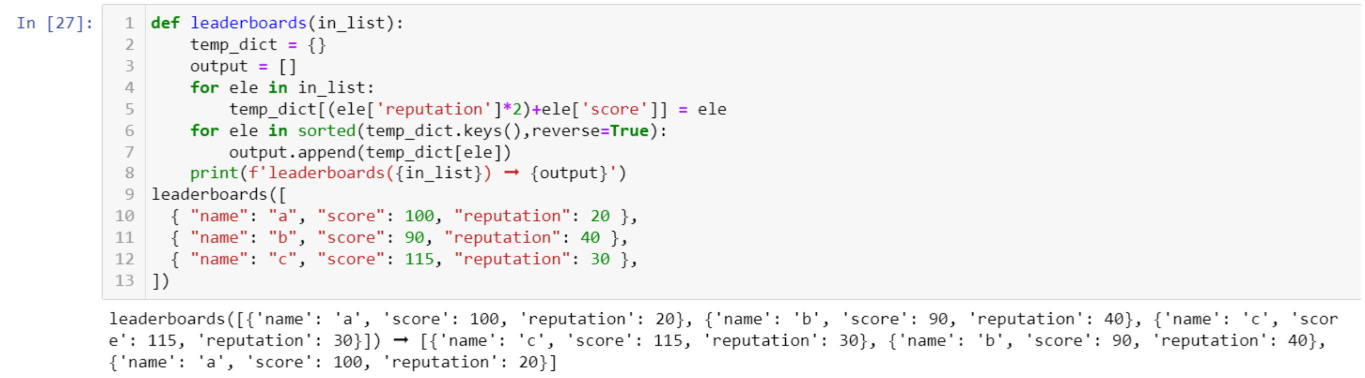
2. Given an array of users, each defined by an object with the following properties: name, score, reputation create a function that sorts the array to form the correct leaderboard.

The leaderboard takes into consideration the score of each user of course, but an emphasis is put on their reputation in the community, so to get the trueScore, you should add the reputation multiplied by 2 to the score.

Once you know the trueScore of each user, sort the array according to it in descending order.

Examples:
leaderboards([
  { "name": "a", "score": 100, "reputation": 20 },
  { "name": "b", "score": 90, "reputation": 40 },
  { "name": "c", "score": 115, "reputation": 30 },
]) ➞ [
  { "name": "c", "score": 115, "reputation": 30 }, # trueScore = 175
  { "name": "b", "score": 90, "reputation": 40 },   # trueScore = 170
  { "name": "a", "score": 100, "reputation": 20 }   # trueScore = 140
]

Ans:

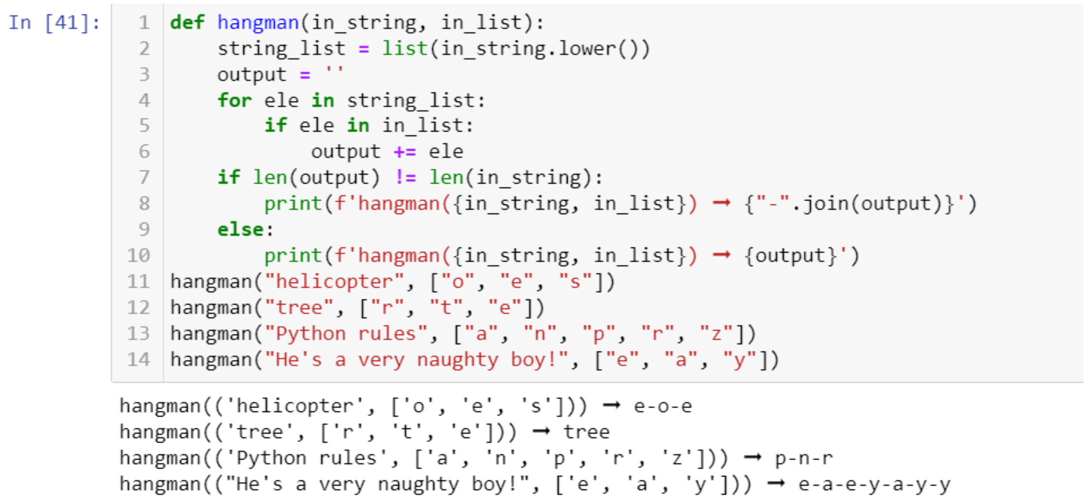


```
In [27]: def leaderboards(in_list):
             temp_dict = {}
             output = []
             for ele in in_list:
                 temp_dict[(ele['reputation']*2)+ele['score']] = ele
             for ele in sorted(temp_dict.keys(),reverse=True):
                 output.append(temp_dict[ele])
             print(f'leaderboards({in_list}) ➞ {output}')
         leaderboards([
           { "name": "a", "score": 100, "reputation": 20 },
           { "name": "b", "score": 90, "reputation": 40 },
           { "name": "c", "score": 115, "reputation": 30 },
         ])
```

```
leaderboards([{'name': 'a', 'score': 100, 'reputation': 20}, {'name': 'b', 'score': 90, 'reputation': 40}, {'name': 'c', 'scor
e': 115, 'reputation': 30}]) ➞ [{'name': 'c', 'score': 115, 'reputation': 30}, {'name': 'b', 'score': 90, 'reputation': 40},
{'name': 'a', 'score': 100, 'reputation': 20}]
```

3. Create a function that, given a phrase and a number of letters guessed, returns a string with hyphens - for every letter of the phrase not guessed, and each letter guessed in place.

Examples:
hangman("helicopter", ["o", "e", "s"]) ➞ "-e---o--e-"
hangman("tree", ["r", "t", "e"]) ➞ "tree"
hangman("Python rules", ["a", "n", "p", "r", "z"]) ➞ "P----n r----"
hangman("He"s a very naughty boy!", ["e", "a", "y"]) ➞ "-e"- a -e-y -a----y –y!"
Ans:



```
In [41]: def hangman(in_string, in_list):
             string_list = list(in_string.lower())
             output = ''
             for ele in string_list:
                 if ele in in_list:
                     output += ele
             if len(output) != len(in_string):
                 print(f'hangman({in_string, in_list}) ➞ {"-".join(output)}')
             else:
                 print(f'hangman({in_string, in_list}) ➞ {output}')
         hangman("helicopter", ["o", "e", "s"])
         hangman("tree", ["r", "t", "e"])
         hangman("Python rules", ["a", "n", "p", "r", "z"])
         hangman("He's a very naughty boy!", ["e", "a", "y"])
```

```
hangman(('helicopter', ['o', 'e', 's'])) ➞ e-o-e
hangman(('tree', ['r', 't', 'e'])) ➞ tree
hangman(('Python rules', ['a', 'n', 'p', 'r', 'z'])) ➞ p-n-r
hangman(("He's a very naughty boy!", ['e', 'a', 'y'])) ➞ e-a-e-y-a-y-y
```

4. The Collatz sequence is as follows:

Start with some given integer n.
If it is even, the next number will be n divided by 2.
If it is odd, multiply it by 3 and add 1 to make the next number.
The sequence stops when it reaches 1.
According to the Collatz conjecture, it will always reach 1. If that's true, you can construct a finite sequence following the aforementioned method for any given integer.

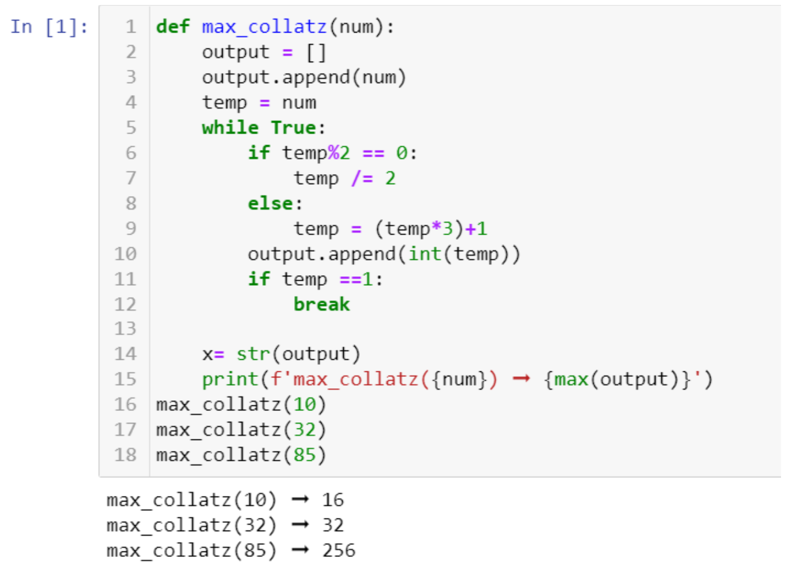
Write a function that takes in an integer n and returns the highest integer in the corresponding Collatz sequence.

Examples:
max_collatz(10) ➞ 16
# Collatz sequence: 10, 5, 16, 8, 4, 2, 1

max_collatz(32) ➞ 32
# Collatz sequence: 32, 16, 8, 4, 2, 1

max_collatz(85) ➞ 256
# Collatz sequence: 85, 256, 128, 64, 32, 16, 8, 4, 2, 1

Ans:



```
In [1]:
def max_collatz(num):
    output = []
    output.append(num)
    temp = num
    while True:
        if temp%2 == 0:
            temp /= 2
        else:
            temp = (temp*3)+1
        output.append(int(temp))
        if temp ==1:
            break

    x= str(output)
    print(f'max_collatz({num}) ➞ {max(output)}')
max_collatz(10)
max_collatz(32)
max_collatz(85)
```

```
max_collatz(10) ➞ 16
max_collatz(32) ➞ 32
max_collatz(85) ➞ 256
```

5. Write a function that sorts a list of integers by their digit length in descending order, then settles ties by sorting numbers with the same digit length in ascending order.

Examples:
digit_sort([77, 23, 5, 7, 101]) ➞ [101, 23, 77, 5, 7]
digit_sort([1, 5, 9, 2, 789, 563, 444]) ➞ [444, 563, 789, 1, 2, 5, 9]
digit_sort([53219, 3772, 564, 32, 1]) ➞ [53219, 3772, 564, 32, 1]
Ans:

In [2]:

```python
def digit_sort(in_list):
    max_len = len(str(max(in_list)))
    output = []
    for item in range(max_len,0,-1):
        temp = []
        for ele in in_list:
            if len(str(ele)) == item:
                temp.append(ele)
        output.extend(sorted(temp))
    print(f'digit_sort({in_list}) ➞ {output}')

digit_sort([77, 23, 5, 7, 101])
digit_sort([1, 5, 9, 2, 789, 563, 444])
digit_sort([53219, 3772, 564, 32, 1])
```

```
digit_sort([77, 23, 5, 7, 101]) ➞ [101, 23, 77, 5, 7]
digit_sort([1, 5, 9, 2, 789, 563, 444]) ➞ [444, 563, 789, 1, 2, 5, 9]
digit_sort([53219, 3772, 564, 32, 1]) ➞ [53219, 3772, 564, 32, 1]
```