Question 1

Your goal in this kata is to implement a difference function, which subtracts one list from another and returns the result.

It should remove all values from list a, which are present in list b keeping their order.

```
array_diff([1,2],[1]) == [2]
```

If a value is present in b, all of its occurrences must be removed from the other:

```
array_diff([1,2,2,2,3],[2]) == [1,3]
```

In [1]:
```
def array_diff(a, b):
    ## Code Here
    result = []
    for item in a:
        if item not in b:
            result.append(item)

    return result
```

In [2]:
```
# @title Test code should run and not return error
def test_1():
    sample = ([1,2], [1])
    assert(array_diff(sample[0],sample[1])) == [2]
def test_2():
    sample = ([1,2,2], [1])
    assert(array_diff(sample[0],sample[1])) == [2,2]
def test_3():
    sample = ([1,2,2], [2])
    assert(array_diff(sample[0],sample[1])) == [1]
def test_4():
    sample = ([1,2,2], [])
    assert(array_diff(sample[0],sample[1])) == [1,2,2]
def test_5():
    sample = ([], [1,2])
    assert(array_diff(sample[0],sample[1])) == []
def test_6():
    sample = ([1,2,3], [1, 2])
    assert(array_diff(sample[0],sample[1])) == [3]

test_1()
test_2()
test_3()
test_4()
test_5()
test_6()
```

Analysis Here

This function compares two lists and removes any items from the first list that are also found in the second list. It goes through the first list item by item and only keeps the ones that don't appear in the second list, preserving their original order exactly as they were.

Question 2

Your task is to make a function that can take any non-negative integer as an argument and return it with its digits in descending order. Essentially, rearrange the digits to create the highest possible number.

Examples: Input: 42145 Output: 54421

Input: 145263 Output: 654321

Input: 123456789 Output: 987654321

In [13]:
```
def ips_between(num):
    ## Answer Here
    digits = list(str(num))
    digits.sort(reverse=True)
    result = int(''.join(digits))
    print(f"ips_between({num}) = {result}")
    return result
```

In [15]:
```
# @title Test code should run and not return error

def test_1():
    assert(ips_between(0)== 0)
def test_2():
    assert(ips_between(15)== 51)
def test_3():
    assert(ips_between(123456789)== 987654321)
test_1()
test_2()
test_3()
```

```
ips_between(0) = 0
ips_between(15) = 51
ips_between(123456789) = 987654321
```

Analysis Here

This function takes a non-negative whole number and sorts its individual digits from largest to smallest to create the highest possible number you can form using those same digits. For example, 42145 becomes 54421 because 5, 4, 4, 2, 1 is the descending order of those digits.

Question 3

Complete the square sum function so that it squares each number passed into it and then sums the results together. For example, for [1, 2, 2] it should return 9 because $1^2+2^2+2^2=9$.

In [5]:
```
def square_sum(numbers):
    #your code here
    squaredSum = 0
    for number in numbers:
        squaredSum+=number**2
    print(squaredSum)
    return squaredSum
```

In [6]:
```
# @title Test code should run and not return error

def test_1():
    sample = [1,2]
    assert(square_sum(sample)) == 5
def test_2():
    sample = [0,3,4,5]
    assert(square_sum(sample)) == 50
def test_3():
    sample = []
    assert(square_sum(sample)) == 0
def test_4():
    sample = [-1,-2]
    assert(square_sum(sample)) == 5
def test_5():
    sample = [-1,0,1]
    assert(square_sum(sample)) == 2

test_1()
test_2()
test_3()
test_4()
test_5()
```

```
5
50
0
5
2
```

Analysis Here

This function calculates the sum of squares for a list of numbers. It works by taking each number in the list, multiplying it by itself to get the square, and then adding all those squared values together to get a single total result, handling both positive and negative numbers correctly.

Question 4

Write a function that takes in a string of one or more words, and returns the same string, but with all words that have five or more letters reversed (Just like the name of this Kata). Strings passed in will consist of only letters and spaces. Spaces will be included only when more than one word is present.

Examples:

```
"Hey fellow warriors"  --> "Hey wollef sroirraw"
"This is a test        --> "This is a test"
"This is another test" --> "This is rehtona test"
```

In [7]:
```python
def spin_words(sentence):
    # Answer here
    words = sentence.split()
    result_words = [word[::-1] if len(word) >= 5 else word for word in words
    result = ' '.join(result_words)
    return result
```

In [8]:
```python
# @title Test code should run and not return error

def test_1():
    sample = "Welcome"
    assert(spin_words(sample)) == "emocleW"
def test_2():
    sample = "to"
    assert(spin_words(sample)) == "to"
def test_3():
    sample = "CodeWars"
    assert(spin_words(sample)) == "sraWedoC"
def test_4():
    sample = "Hey fellow warriors"
    assert(spin_words(sample)) == "Hey wollef sroirraw"
def test_5():
    sample = "This sentence is a sentence"
    assert(spin_words(sample)) == "This ecnetnes is a ecnetnes"

test_1()
test_2()
test_3()
test_4()
test_5()
```

Analysis Here

This function processes a sentence by looking at each word separately. Any word that contains five or more letters gets completely reversed (written backwards), while words with four or fewer letters remain unchanged. The function then puts all the words back together with spaces between them just like the original sentence.

Question 5

DESCRIPTION: Well met with Fibonacci bigger brother, AKA Tribonacci.

As the name may already reveal, it works basically like a Fibonacci, but summing the last 3 (instead of 2) numbers of the sequence to generate the next. And, worse part of it, regrettably I won't get to hear non-native Italian speakers trying to pronounce it :(

So, if we are to start our Tribonacci sequence with [1, 1, 1] as a starting input (AKA signature), we have this sequence:

```
[1, 1 ,1, 3, 5, 9, 17, 31, ...]
```

But what if we started with [0, 0, 1] as a signature? As starting with [0, 1] instead of [1, 1] basically shifts the common Fibonacci sequence by once place, you may be tempted to think that we would get the same sequence shifted by 2 places, but that is not the case and we would get:

```
[0, 0, 1, 1, 2, 4, 7, 13, 24, ...]
```

Well, you may have guessed it by now, but to be clear: you need to create a fibonacci function that given a signature array/list, returns the first n elements - signature included of the so seeded sequence.

Signature will always contain 3 numbers; n will always be a non-negative number; if n == 0, then return an empty array (except in C return NULL) and be ready for anything else which is not clearly specified ;)

If you enjoyed this kata more advanced and generalized version of it can be found in the Xbonacci kata

[Personal thanks to Professor Jim Fowler on Coursera for his awesome classes that I really recommend to any math enthusiast and for showing me this mathematical curiosity too with his usual contagious passion :)]

In [9]:
```python
def tribonacci(signature, n):
    # Answer Here
    result = signature[:n]

    while len(result) < n:
        result.append(sum(result[-3:]))

    return result
```

In [10]:
```python
# @title Test code should run and not return error
```

```
def test_1():
    sample = ([1, 1, 1], 10)
    assert(tribonacci(sample[0],sample[1])) == [1, 1, 1, 3, 5, 9, 17, 31, 57
def test_2():
    sample = ([0, 0, 1], 10)
    assert(tribonacci(sample[0],sample[1])) == [0, 0, 1, 1, 2, 4, 7, 13, 24,
def test_3():
    sample = ([0, 1, 1], 10)
    assert(tribonacci(sample[0],sample[1])) == [0, 1, 1, 2, 4, 7, 13, 24, 44
def test_4():
    sample = ([1, 0, 0], 10)
    assert(tribonacci(sample[0],sample[1])) == [1, 0, 0, 1, 1, 2, 4, 7, 13,
def test_5():
    sample = ([0, 0, 0], 10)
    assert(tribonacci(sample[0],sample[1])) == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0
def test_6():
    sample = ([1, 2, 3], 10)
    assert(tribonacci(sample[0],sample[1])) == [1, 2, 3, 6, 11, 20, 37, 68,
def test_7():
    sample = ([3, 2, 1], 10)
    assert(tribonacci(sample[0],sample[1])) == [3, 2, 1, 6, 9, 16, 31, 56, 1
def test_8():
    sample = ([1, 1, 1], 1)
    assert(tribonacci(sample[0],sample[1])) == [1]
def test_9():
    sample = ([300, 200, 100], 0)
    assert(tribonacci(sample[0],sample[1])) == []
def test_10():
    sample = ([0, 0, 1], 10)
    assert(tribonacci(sample[0],sample[1])) == [0, 0, 1, 1, 2, 4, 7, 13, 24,
def test_11():
    sample = ([0.5, 0.5, 0.5], 30)
    assert(tribonacci(sample[0],sample[1])) == [0.5, 0.5, 0.5, 1.5, 2.5, 4.5

test_1()
test_2()
test_3()
test_4()
test_5()
test_6()
test_7()
test_8()
test_9()
test_10()
test_11()
```

Analysis Here

This function generates a Tribonacci sequence, which is like the Fibonacci sequence but instead of adding the last two numbers, you add the last three numbers to get the next one. Starting with three numbers you provide, it continues this pattern to produce exactly the requested number of elements in the sequence.