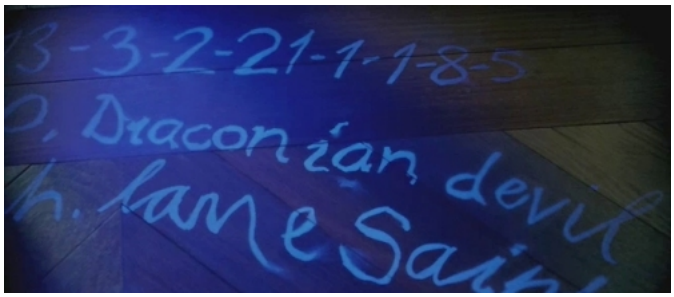
O draconian devil

An **anagram** is the result of rearranging the letters of a word or a phrase to produce a new word o f phrase, using all the original letters exactly once. Anagrams are sometimes used as pseudomyms (*Leonardo da Vinci*: *o draconian devil*; *The Mona Lisa*: *oh lame saint*).



A simple way to determine whether two words or phrases are anagrams, is to compute a hash value in such a way that only anagrams have an identical hash value. One way to compute such as hash value assigns each letter of the alphabet to the next prime number (a=2, b=3, …, z=101), and computes the hash as the product of all values corresponding to the letters of the word or phrase. In computing the hash value, only letters of the alphabet must be taken into account (no punctuation symbols, digits, etc.), and must be treated case insensitive. As such, the words callers, cellars and RECALLS all result in the same hash value 615461330, making them anagrams.

Hint: While working on this programming challenge you should take a break an watch the Monty Python sketch "The man who speaks in anagrams". You will hardly understand a single word of the conversation, unless you have a look at the transcript.

Assignment

1. Write a function nPrime that takes an optional integer argument n. The function must return a list containing the first n prime numbers. If no argument is passed to the function, a list containing the first 10 prime numbers should be returned.

   nPrime([n])

2. Write a function anagramHash that takes exactly two argument. The first argument is a phrase whose hash value must be returned by the function. The second argument is a list containing 26 (or more) integers that must be used to compute the hash value: the letter a corresponds to the first integer in the list, the letter b to the second number, and so on. The hash value is computed as the product of all integers that correspond to the letters in the phrase. When computing the hash value, only letters from the alphabet must be taken into account (no punctuation symbols, digits, etc.), and must be treated case insensitive.

   anagramHash(phrase, integers)

3. Use the functions nPrime and anagramHash to write a function areAnagrams. The function takes two phrases as its argument, and must return a Boolean value that indicates whether the phrases are anagrams (return value True) or not (return value False). In determining whether the phrases are anagrams, the function must compute the hash values of both phrases. By default, the hash value is computed with mapping letters to the first 26 prime numbers. If a list of integers is passed as a third argument to the function, a mapping to these integers

must be used to compute the hash values of the phrases.

areAnagrams(phrase1, phrase2[, integers])

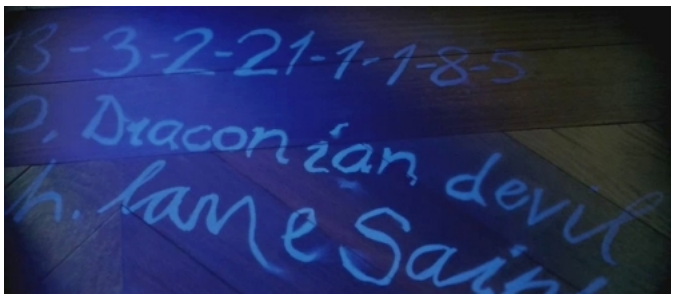
Example

```
>>> nPrime()
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
>>> nPrime(5)
[2, 3, 5, 7, 11]
>>> nPrime(8)
[2, 3, 5, 7, 11, 13, 17, 19]

>>> anagramHash('Leonardo Da Vinci', nPrime(26))
4153036139030192260
>>> anagramHash('Leonardo Da Vinci', range(1, 27))
4073908608000
>>> anagramHash('Leonardo Da Vinci', nPrime(52)[-26:])
280080025163413341541861091223919
>>> anagramHash('O Draconian Devil', nPrime(26))
4153036139030192260

>>> areAnagrams('Leonardo Da Vinci', 'O Draconian Devil')
True
>>> areAnagrams('The Mona Lisa', 'Oh Lame Saint')
True
>>> areAnagrams('The Mona Lisa', 'Oh Lame Saint', range(1, 27))
True
>>> areAnagrams('The Mona Lisa', 'Oh Lame Saint', nPrime(52)[-26:])
True
```

Een **anagram** is een woord dat of een zin die bestaat uit alle letters van een ander woord of een andere zin. Anagrammen worden vaak gebruikt als pseudoniem (*Leonardo da Vinci*: *o draconian devil*; *The Mona Lisa*: *oh lame saint*).



Een eenvoudige manier om te bepalen of woorden anagrammen zijn, bestaat erin om een hashwaarde te berekenen waarvoor geldt dat enkel anagrammen identieke hashwaarden hebben. Voor de berekening van een hashwaarde kan je bijvoorbeeld aan elke letter van het alfabet een priemgetal toekennen (a=2, b=3, …, z=101), en de waarden die corresponderen met elke letter van het originele woord met elkaar vermenigvuldigen. Communtativiteit van de vermenigvuldiging en uniciteit van de ontbinding in priemfactoren garanderen dat anagrammen een unieke hashwaarde hebben. Bij de berekening van de hashwaarde moeten enkel de letters uit het alfabet in rekening gebracht worden (geen leestekens, cijfers, etc.), en moeten deze niet-hoofdlettergevoelig behandeld worden. Zo resulteren de woorden callers, cellars en RECALLS allemaal in de hashwaarde 615461330, waardoor het dus anagrammen zijn.

Hint: In de context van deze opgave moet je zeker ook eens de Monty Python sketch "The man

who speaks in anagrams" bekijken. Daar valt nauwelijks iets van te begrijpen, tenzij je er de uitgeschreven tekst bijneemt.

Opgave

1. Schrijf een functie nPriem, waaraan optioneel een argument n kan meegegeven worden. De functie moet een lijst met de eerste n priemgetallen teruggeven. Indien geen argument aan de functie wordt meegegeven, moet een lijst met de eerste 10 priemgetallen teruggegeven worden.

   nPriem([n])

2. Schrijf een functie anagramHash waaraan exact twee argumenten moeten meegegeven worden. Het eerste argument is een zin waarvan de hashwaarde door de functie moet teruggegeven worden. Het tweede getal is een lijst van 26 (of meer) getallen die moeten gebruikt worden om de hashwaarde te berekenen: de letter a correspondeert met het eerste getal uit de lijst, de letter b met het tweede getal, enzoverder. De hashwaarde wordt berekend als het product van alle getallen die corresponderen met de letters uit de zin. Bij de berekening van de hashwaarde moeten enkel de letters uit het alfabet in rekening gebracht worden (geen leestekens, cijfers, etc.), en moeten deze niet-hoofdlettergevoelig behandeld worden.

   anagramHash(zin, getallenlijst)

3. Gebruik de functies nPriem en anagramHash om een functie zijnAnagrammen te schrijven. Aan deze functie moeten twee zinnen doorgegeven worden, waarvan de functie moet bepalen of het anagrammen zijn (waarde True teruggegeven) of niet (waarde False teruggegeven). Om de vaststelling te maken of het anagrammen zijn, moet de functie de hashwaarde van de twee zinnen berekenen. Standaard wordt deze hashwaarde bepaald aan de hand van een mapping naar de lijst van de eerste 26 priemgetallen. Indien er echter een lijst van getallen als derde argument aan de functie doorgegeven wordt, dan worden deze getallen gebruikt om de hashwaarde van de zinnen te berekenen.

   zijnAnagrammen(zin1, zin2[, getallenlijst])

Voorbeeld

```
>>> nPriem()
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
>>> nPriem(5)
[2, 3, 5, 7, 11]
>>> nPriem(8)
[2, 3, 5, 7, 11, 13, 17, 19]

>>> anagramHash('Leonardo Da Vinci', nPriem(26))
4153036139030192260
>>> anagramHash('Leonardo Da Vinci', range(1, 27))
4073908608000
>>> anagramHash('Leonardo Da Vinci', nPriem(52)[-26:])
28008002516341334154186109122391 9
>>> anagramHash('O Draconian Devil', nPriem(26))
4153036139030192260

>>> zijnAnagrammen('Leonardo Da Vinci', 'O Draconian Devil')
```

```
True
>>> zijnAnagrammen('The Mona Lisa', 'Oh Lame Saint')
True
>>> zijnAnagrammen('The Mona Lisa', 'Oh Lame Saint', range(1, 27))
True
>>> zijnAnagrammen('The Mona Lisa', 'Oh Lame Saint', nPriem(52)[-26:])
True
```