# Série 2

Maintenant que vous connaisez un peu le Jupyter, ça devrait être plus simple. Dans cette série, nous allons découvrir les functions de hachage. On va d'abord faire quelques essais avec des functions de hacahge peu performant, pour voire comment on peut les attaquer. Ensuite on va passer aux *vraies* functions de hachage, et voir que les attaques sont beaucoup plus difficiles que ça.

# Exercice 1

Pour cette exercice, nous allons utiliser une function de hachage très simple pour protéger un message. Puis on va attaquer cette function afin de changer le message sans que la vérification le remarque.

On verra le jour 3 comment ceci peut être utilisé dans des produits commerciaux. Pour cet exercice, on va juste assumer que la chose suivante se passe:

1. Un long texte est disponible sur internet
2. Sur le site se trouve le numéro obtenu après hachage de ce texte
3. Un ami nous donne une copie du texte, et on veut vérifier si le texte est authentique.

## 1. Connaissance

On va inventer une function de hachage très simple, qui fait la chose suivante:

- pour chaque charactère d'une phrase, prendre son numéro ASCII, multipliez le par la position dans la phrase, et prenez la somme de tous ces numéros

Dans la partie de code en bas, cette function s'appelle `simple_hash` et prend en entrée une phrase, pour sortir un numéro.

Vous pouvez démarrer la première partie, et vérifier que:

1. Les phrases 1 et 2 ont bien un numéro différent
2. Les phrases 2 et 3 produisent le même numéro, il y a donc une collision!

Aussi important pour une function de hachage, cette fois-ci chaque fois que vous lancez le block, vous recevez le même résultat!

## 2. Compréhension

Maintenant on va utiliser cette function de hachage pour coment casser la résistance aux collisions qui sont définies dans une function de hachage cryptographique:

1. Collision libre
2. Résistance aux pre-image
3. Résistance aux 2nd pre-image

### Collision libre

Pour cette attaque, il vous faut trouver deux phrases qui donnent le même résultat de hachage:

- aucune restriction
- les phrases doivent être au moins 10 caractères de long
- comme avant, mais les 10 premières caractères doivent être les mêmes

### Résistance au pre-image

Maintenant on s'intéresse à une attaque qui essaie d'introduire un message aléatoire, donné le hachage. Ce qui complique cette exercice, c'est qu'on n'a aucune idée du message qui a généré ce hachage!

Le numéro de hachage est le: 29033!

### Résistance au 2nd pre-image

Cet exercice est en fait un peu plus simple: en plus du numéro de hachage, on possède un message qui a été utilisé pour créer ce numéro de hachage. On peut donc comencer à changer les chiffres et lettres jusqu'à obtenir le même numéro de hachage.

La phrase pour laquelle il faut trouver une collision est:

"So long and thanks for all the fish"

## 3. Application

Si vous connaissez un peu la programmation en Python, il est possible de faire une attaque plus automatisée pour les trois collisions de la partie "Compréhension":

### Collision libre

Ecrivez la fonction `collision_any` qui prend en argument la longueur de la phrase et qui répond avec deux phrases qui donnent le même hachage. Pour les phrases, prenez seulement les lettres alphanumériques. Ou pour faire plus joli, incluez un dictionnaire avec des mots que vous choisissez pour créer les phrases.

### Pre-image

Ecrivez la fonction `collision_pre_image` qui prend un numéro de hachage et qui retourne une phrase qui a le même numéro de hachage.

### 2nd pre-image

Ecrivez la fonction `collision_2nd_pre_image` qui prend une phrase et qui retourne une autre phrase avec des changements minimaux pour donner le même numéro de hachage.

Vous pouvez 'tricher' et résoudre ce problème dans un temps constant indépendant de la phrase donnée. Mais dans ce cas il faudra bien faire attention au cas limites...

In [1]:

```
# Exercice 1 - Partie 1

# This calculates the hash as the sum of all characters in a phrase. Of course
# this is very insecure and should never be used in real life.
def simple_hash(phrase: str) -> int:
  hash = 0
  for i, c in enumerate(phrase):
    hash = hash + ord(c) * i

  return hash

# Pretty print the hash of a phrase.
def print_hash(phrase: str):
  print('Hash of "{}" is: {}'.format(phrase, simple_hash(phrase)))

print_hash("Payez 150CHF pour la guitare")
print_hash("Payez 1500CHF pour la guitare")
print_hash("Payez 2310CHF pour la guitare")
```

```
Hash of "Payez 150CHF pour la guitare" is: 34428
Hash of "Payez 1500CHF pour la guitare" is: 36577
Hash of "Payez 2310CHF pour la guitare" is: 36577
```

In [30]:

```
# Exercice 1 - Partie 2

# Créer une collision libre
print_hash("Première phtase")
print_hash("Deuxième phrase")

# Créer une pre-image du numéro de hachage 29033
print_hash(" ,                          1      29033")

# Créer une second pre-image de "So long and thanks for all the fish"
print_hash("So long and thanks for all the fish")
print_hash("To long and thanks for all the fish")
print_hash("So plng and thanks for all the fish")
```

```
Hash of "Première phtase" is: 11267
Hash of "Deuxième phrase" is: 11267
Hash of " ,                          1      29033" is: 29033
Hash of "So long and thanks for all the fish" is: 54845
Hash of "To long and thanks for all the fish" is: 54845
Hash of "So plng and thanks for all the fish" is: 54845
```

In [58]:

```
# Exercice 1 - Partie 3
import random, string

def rotate_char(c: str) -> str:
    letter_case = ord(c[0]) & 0xe0
    letter_pos = ((ord(c[0]) & 0x1f) + 1) % 26
    return chr(letter_case + letter_pos)

# Abuses the fact that the hash doesn't take into account the
# first possition.
def collision_any(length: int) -> [str, str]:
    phrase_1 = random.choices(string.ascii_lowercase, k=length)
    phrase_2 = list(phrase_1)
    phrase_2[0] = rotate_char(phrase_1)

    return [''.join(phrase_1), ''.join(phrase_2)]

def collision_pre_image(hash: int) -> str:
  pass

# Also using the fact that the first position is not taken into account.
def collision_2nd_pre_image(phrase: str) -> str:
    phrase_2 = list(phrase)
    phrase_2[0] = rotate_char(phrase)
    return ''.join(phrase_2)

print(collision_any(10))
print(collision_2nd_pre_image("Marvin the depressive robot"))
```

```
['fptfnakioa', 'gptfnakioa']
Narvin the depressive robot
```