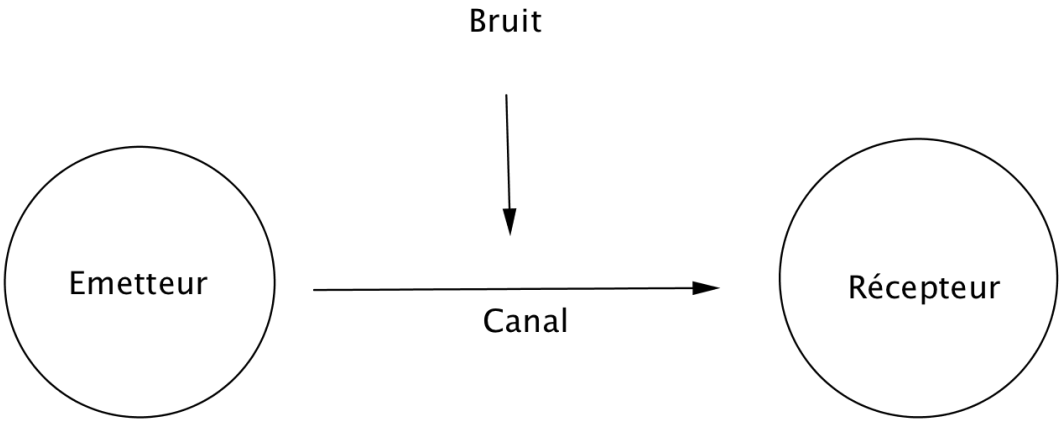
Un peu d'histoire...

"Le premier engin spatial de la série Mariner prend des photographies en noir et blanc de la planète Mars en 1965. Les images ont une résolution de 200 x 200 pixels avec 64 nuances de gris codées sur 6 bits. Les données sont transmises avec un débit binaire de 8 bits par seconde et par conséquent la transmission d'une image prend environ 8 heures.
Plus tard en 1972, Mariner 9 arrive en orbite autour de Mars. Pour permettre une correction des erreurs de transmission, le code Reed-Muller avec des mots de code de 32 bits est utilisé (6 bits pour la nuance de gris du pixel et 26 bits de clé de redondance). Par contre les progrès au niveau des techniques de transmission ont alors permis d'atteindre un débit de 16000 bits par seconde" (Architecture des ordinateurs- Jean-Jacques Schwarz- Eyrolles- 2005)(Voir aussi sur Youtube Mariner4 Mars)



Code détecteur d'erreurs De manière générale, la plupart des messages (texte, son, image) sont **numérisés** et transformés en une suite de bits c'est à dire de 0 et de 1. Au cours de leur transmission de l'émetteur au récepteur avec une certaine probabilité (taux d'erreur) qui dépend du canal de transmission un bit transmis peut être différent du bit émis et ceci n'est pas dû à un acte de malveillance mais à des perturbations physiques dans le canal de transmission, on parle de canal bruité.

Il existe des techniques, pour détecter les erreurs , on parle de codes détecteurs d'erreurs et d'autres pour détecter et corriger les erreurs, on parle de codes correcteurs d'erreurs. Regardons sur un exemple simplifié l'idée de la détection d'erreurs.

Exemple 1 : bit de parité

On veut envoyer le message "Bonjour".

1. Associer à chaque lettre du mot ci-dessus les 7 bits correspondants du tableau ASCII (American Standard Code for Information Interchange). (de 'A' à 'Z' le code en décimal est 65 jusqu'à 90, puis de 'a' à 'z' 97 à 122, pour convertir en binaire, utiliser l'interpréteur Python et la fonction bin())

   Si $s_7 s_6 ... s_1 s_0$ avec $s_i \in \{0, 1\}$ est l'équivalent numérique d'une lettre on associe à ce bloc 1 bit supplémentaire s_7 dit de parité , dans le but de **détecter** s'il y a une erreur dans la transmission, ainsi :

   Si la somme $s_0 + s_1 + ... + s_6$ est paire alors $s_7 = 0$ sinon $s_7 = 1$ (cela revient à compter le nombre de 1)

   Par exemple la lettre A est codée par bin(65) = 1000001,le bit de parité sera ici **0** que l'on ajoute aux bits précédents on obtient **0**1000001

   Donc le début du message est "Bonjour" est $\mathbf{0}\underbrace{1000010}_{\text{B}}\ \mathbf{0}\underbrace{1101111}_{\text{o}}$ Continuer.
2. Vérifier qu'on peut détecter à coup sûr une erreur sur 1 bit avec le bit de parité
3. Peut on détecter deux erreurs sur 7 bits ?
4. On aimerait avoir un ordre de grandeur qu'il y ait une erreur sur ce message. On suppose que la probabilité qu'il y ait une erreur sur un bit sur le canal de transmission est p (p variant de 10^{-4} à 10^{-12}) Proposer un modèle mathématique permettant de calculer la probabilité en fonction de p qu'il y ait une erreur sur un message de longueur L (par exemple $L = 50$. Faire le calcul comparer cet évènement avec un autre évènement "rare" .
5. Est il "rare" qu'il y ait deux erreurs sur 7 bits consécutifs ?

Exemple 2 : numéro INSEE

Le numéro INSEE (dit de sécurité sociale) attribué à chaque Français est un numéro N de $n = 15$ chiffres composé d'un numéro d'identification K de $k = 13$ chiffres et d'une clé R de $r = 2$ chiffres calculée ainsi :

On calcule le reste R' de la division euclidienne par 97 de K , la clé R vérifie $R = 97 - R'$

K est obtenu de la manière suivante :

1. Le premier chiffre est 1 si vous êtes un homme 2 si vous êtes une femme
2. Les deux suivants sont les deux derniers chiffres de l'année de naissance de 00 à 99
3. Les deux suivants forment le mois de naissance de 01 à 12
4. Les chiffres de position 6 à 10 désignent le lieu de naissance en tenant compte des différents cas : France métropolitaine , Corse, Départements d'outre-mer, Etranger.
5. Les trois derniers chiffres de 001 à 999 désignent l'ordre de naissance dans le mois et le lieu

De manière générale on représente le numéro d'identification sous sa forme mathématique

$K = c_{12}c_{11}...c_0 = \sum_{i=0}^{i=12} c_i 10^i$

1. Quel est la clé du numéro K = 1 17 12 91471 001 ? (Utiliser l'interpréteur Python et entrer 1171291471 001 % 97) (L'opérateur % modulo donne le reste de la division euclidienne)
2. On définit le rendement du code détecteur ainsi pour $k = 7$ bits d'information on a ajouté $r = 1$ bits de détection pour obtenir $n = k + r$ bits de code, le rendement est définit par $\frac{k}{n} = \frac{7}{8}$ Quel est le rendement du code INSEE ?
3. Combien d'erreurs de chiffres la clef permet elle de détecter ? (Pourquoi 97 ?)
   (On traitera dans un premier temps le cas où il y a une erreur sur 1 chiffre.Si on considère $K = c_{12}c_{11}...c_0 = \sum_{i=0}^{i=12} c_i 10^i$, s'il y a une erreur sur un chiffre , le nouveau chiffre K' notons c_i et $c_{i'}$ les deux chiffres différents de K et K', est il possible que K et K' ont la même clé ?(Mettre en équation cette question)
   Dans un second temps on traitera le cas de deux erreurs en cherchant un contre-exemple simple)
4. **Comment faire sans ordinateur ?**
   Le reste de la division euclidienne de 100 par 97 est 3, que l'on écrit $10^2 \equiv 3\ (97)$ (lire 100 est congru à 3 modulo 97)

   $$10^4 = (10^2)^2 \equiv ?\ (97)$$

   En vous aidant de l'interpréteur Python conjecturer des règles de simplification qui permettront de calculer le reste
   de 1171291471 001 par 97 à partir de calculs faits sur des nombres plus "petits"
   Finir en remarquant que :
   $1171291471001 = 11712 \times 10^8 + 9147 \times 10^4 + 10 \times 10^2 + 1$
   $11712 \times 10^8 + 9147 \times 10^4 + 10 \times 10^2 + 1 =$
   $(10^4 + 17 \times 10^2 + 2) \times 10^8 + (91 \times 10^2 + 47) \times 10^4 + 10 \times 10^2 + 1$

Exemple 3 : code ISBN et code Barre EAN-13

Sur la quatrième de couverture du livre de Simon SINGH, *Histoire des codes secrets* livre de Poche, on peut voir un code barre 9 782253 150978 et un code ISBN (International standard book number)978-2-253-15097-8.

Comment fonctionne ces codes ?

Le code EAN-13 (*European Article Numbering*) est un code barres composé d'un numéro de 13 chiffres $c_{12} - c_{11}c_{10}c_9c_8c_7c_6 - -c_5c_4c_3c_2c_1c_0$

Les 12 chiffres de c_1 à c_{12} identifient le produit la clé c_0 est calculée ainsi :

On calcule la somme des chiffres de rang pair $p = c_2 + c_4 + ... + c_{12}$ et la somme des chiffres impairs $i = c_1 + c_3 + ... + c_{11}$ puis on calcule le reste r de la division $p + 3 \times i$ par 10 enfin $c_0 = 10 - r$

Le code ISBN depuis le 1 janvier 2007 est le code EAN-13

1. Vérifier la validité du code ISBN du livre de Simon SINGH
2. Quel est le rendement de ce code ?
3. Combien d'erreurs de chiffres la clef permet elle de détecter ?

Ecriture décimale, binaire ou hexadécimale d'un nombre : Nous avons tellement l'habitude d'écrire les nombres sous forme décimale que nous nous interrogeons plus sur le fonctionnement de cette écriture.
Le nombre 256 en décimal que l'on devrait écrire 256_{10}, signifie 2 centaines + 5 dizaines + 6 unités ce qui mathématiquement s'écrit

$$256_{10} = 2 \times 10^2 + 5 \times 10 + 6$$

Dénombrer en décimal c'est donc faire des "paquets " de puissances de 10. Combien de paquets ? Pas plus que le plus grand des chiffres, c'est à dire 9, parce que $10 \times 10^2 = 1 \times 10^3$ par exemple
Au lieu de faire des paquets de puissances de 10 on peut faire des paquets de puissances de 2 or $256_{10} = 1 \times 2^8$ (égalité dans le système décimal)
Donc $256_{10} = 1 \times 2^7 = 10000000_2$ (1 octet)
Une autre base très pratique en informatique est la base 16. Quels sont les chiffres à notre disposition ? 0,1 jusqu' à 9 puis A,B,C,D,E,F. Pourquoi ?
Mettez 16 jetons sur une table, nous avons déjà donné une indication du nombre de jetons en donnant leur quantité sous forme décimale mais on aurait pu dire 10000 jetons (en base deux) ou 10 jetons (en base 16 = 1 base + 0 unité)
Une nuance de gris est codé de 0 à 255 (en décimal) donc 256 nuances de gris possible , donc sur 2 octets c'est à dire 16 bits qui se réduisent alors à deux chiffres en hexadécimal car $255_{10} = FF_{16}$
En informatique on n'écrit pas FF_{16} mais 0xFF
Utiliser l'interpréteur Python et faire hex(255)