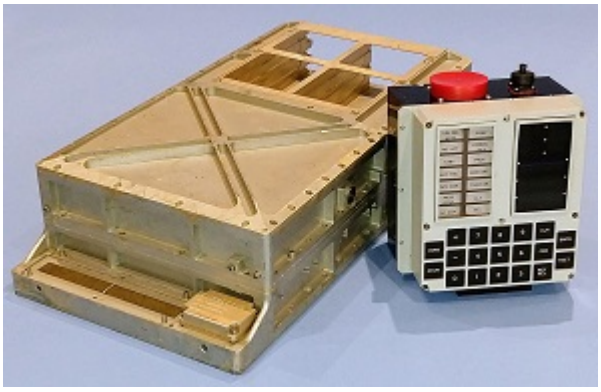
CODAGE DE L'INFORMATION – ROVER PERSEVERANCE

- ▷ Histoire de l'informatique
- ▷ **Représentation des données : types et valeurs de base**
- ▷ Représentation des données : types construits
- ▷ Architectures matérielles et systèmes d'exploitation
- ▷ Interactions entre l'homme et la machine sur le Web
- ▷ Architectures matérielles et systèmes d'exploitation
- ▷ **Langages et programmation**
- ▷ Algorithmique

> L'**Apollo Guidance Computer** (**AGC**) est l'ordinateur embarqué de navigation et de pilotage installé dans les vaisseaux spatiaux des **missions Apollo**. L'AGC est un ordinateur **16 bits** effectuant des traitements en temps réel. Sa mémoire est composée de **72 ko** de mémoire morte contenant l'ensemble des programmes et de **4 ko** de mémoire vive (effaçable) utilisée par les traitements. Le poids de l'AGC est d'environ **32 kg**.



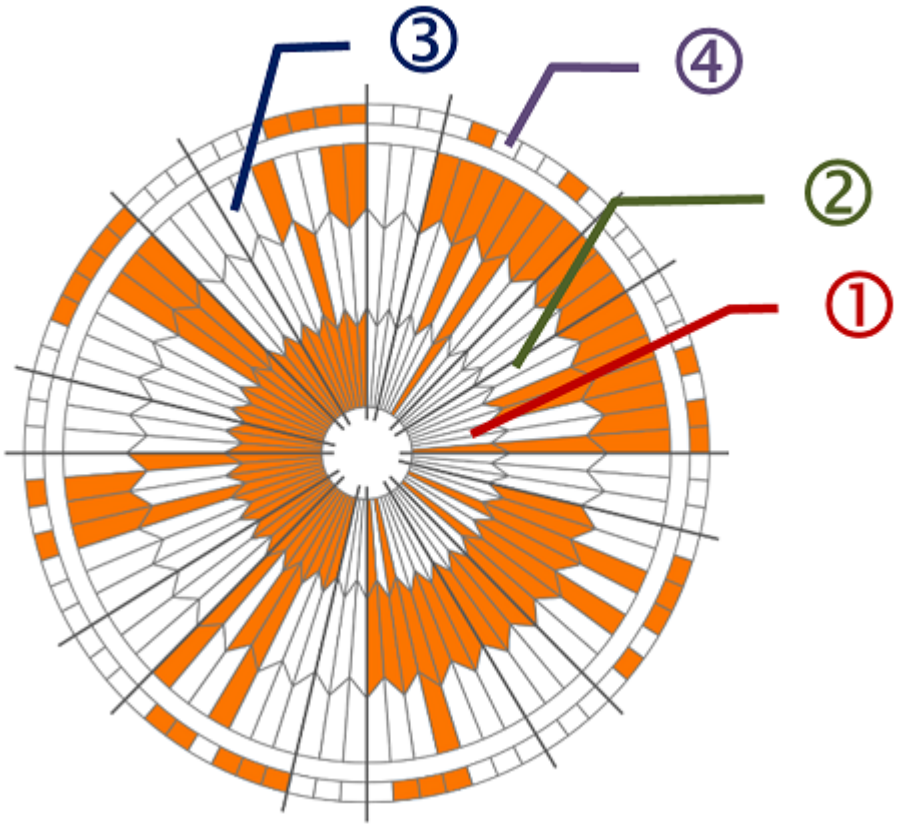
1 – MESSAGE CACHE DANS LE PARACHUTE DE PERSEVERANCE

Sur le site « lemonde.fr » du 23 février 2021, nous pouvions lire : « Après un voyage de sept mois, le rover Perseverance, conçu par les équipes de la NASA, s'est finalement posé sur le sol de la planète Mars jeudi 18 février : un accomplissement scientifique pour l'agence spatiale américaine, qui a passionné certains internautes. Lundi 22 février, la NASA a publié la vidéo de l'atterrissage de l'engin sur Mars, ajoutant lors d'une conférence de presse, qu'un message secret avait été glissé dans la vidéo ». Vidéo de l'atterrissage de l'atterrissage de Perseverance.

On peut en effet voir que le parachute du rover utilisé pour atterrir présente un motif particulier à base de barres rouges sur fond blanc, et que ces barres suivent un motif qui pourrait cacher un message. Et effectivement, le parachute est divisé en quatre cercles, eux-mêmes composés d'ensembles de dix barres formant un code binaire : chaque barre rouge correspond à un 1, et chaque barre blanche à un 0.

2 – DECODAGE DU MESSAGE CACHE

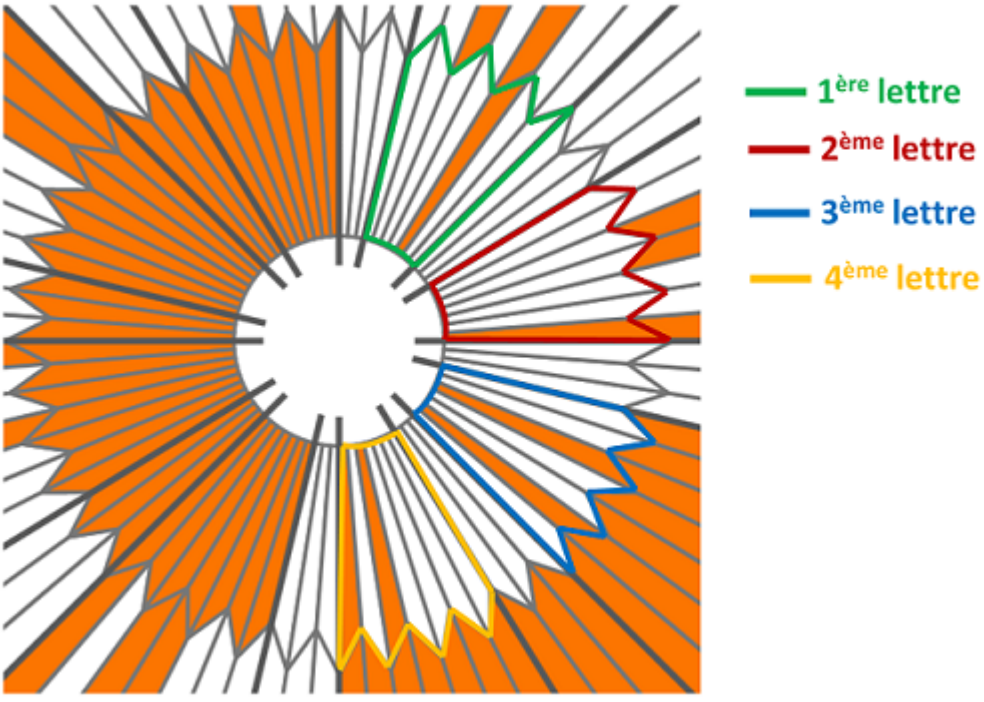
Nous observons 4 cercles. Dans chacun des cercles se cache un message constitué de caractères codés sur 7 bits. Chaque caractère est séparé par 3 bits à 0. On parcourt les cercles dans le sens horaire et lorsqu'on en a terminé un, il suffit de passer sur le suivant en spirale.



Chacun des cercles 1, 2 et 3 code un mot constitué d'un certain nombre de lettres. Pour obtenir le code ASCII de chacune des lettres, il faut additionner 64 à la valeur décimale du code binaire :

code binaire sur 7 bits --> code décimal + 64 ==> code ASCII

2.1 – Décodage du cercle 1



Le code binaire de la 1ère lettre est égal à : **0b000 0100 = 4**

Le code ASCII de cette première lettres est : **4 + 64 = 68**

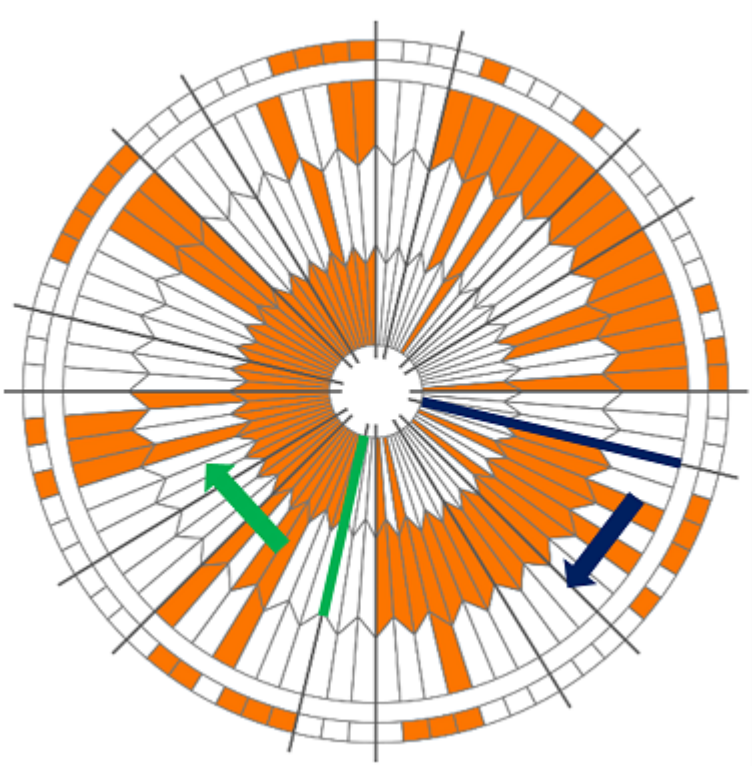
Le code ASCII **68** correspond à la lettre **D**.

> **Exercice 1** :
> **1. Décoder** les 3 autres lettres du 1er cercle. **2. Donner** le mot caché dans le 1er cercle.

> **Réponses** :

2.2 – Décodage des cercles 2 et 3

Après avoir décodé le cercle 1, il suffit de passer sur le cercle suivant en spirale, puis sur le cercle 3 après avoir décodé le cercle 2.



> **Exercice 2** :
> **1. Décoder** les 2 mots cachés dans les 2 autres cercles.
> **2. Donner** la phrase cachée dans les 3 premiers cercles. **Traduire** la phrase en français.

> **Réponses** :

2.3 – Programme de décodage des messages des trois premiers cercles

L'algorithme de la fonction `decode_cercle(code_cercle)` qui permet le décodage des trois premiers cercles est donné ci-dessous. Cette fonction reçoit en paramètre `code_cercle` qui est la suite binaire inscrite dans chaque cercle et renvoie le mot correspondant.

fonction decode_cercle(code_cercle: chaine de caractères) mot <-- Chaine de caractères vide Tant que la taille de codeCercle > 0 Supprimer les trois premiers bits à 0 Déterminer le caractère correspondant aux 7 bits Ajouter le caractère à la chaine mot Supprimer ces 7 bits de codeCercle Renvoyer la chaine mot

Pour réaliser ce programme, vous aurez besoin de la fonction python `chr(int)` qui permet de convertir un nombre entier en caractère et de la fonction `int(strbin, 2)` qui permet convertir une chaîne de caractères représentant un nombre binaire en un nombre décimal.

In [ ]:
```
chr(89)
```

In [ ]:
```
int('0000111', 2)
```

> **Exercice 3** :
> **1. Editer** la fonction `decode_cercle(code_cercle)` afin d'implémenter l'algorithme ci-dessus.

In [ ]:
```
def decode_cercle(code_cercle):
    # A compléter
```

> **2. Tester** votre fonction sur les 3 permiers cercles et **vérifier** que l'on obtient le message décodé dans les exercices précédent.

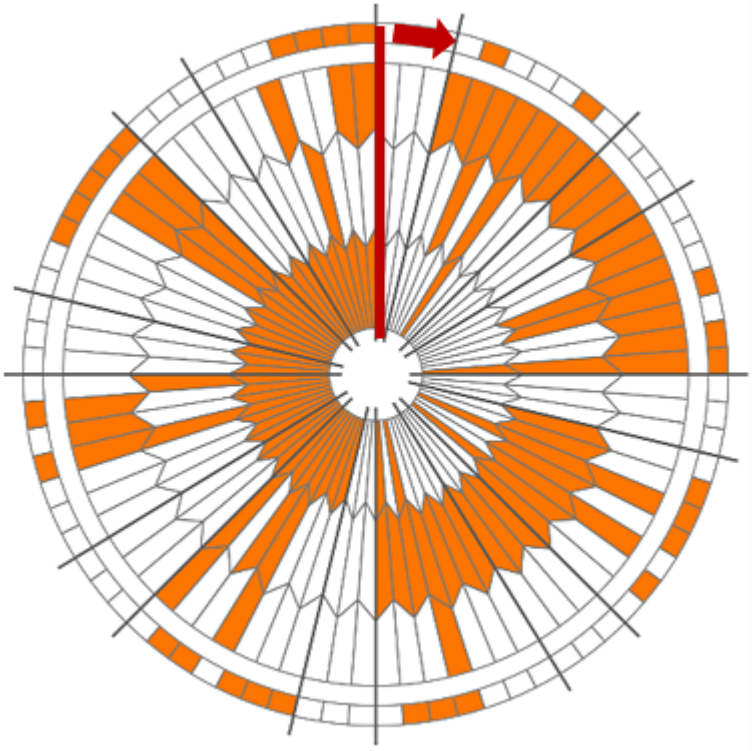
In [ ]:
```
decode_cercle("00000001000000000001000001001000000000101")   # Cercle 1
```

In [ ]:
```
decode_cercle("00000011010000001001000000011100000010000000010100000011001")   # Ce
```

In [ ]:
```
decode_cercle("00000101000000001000000000100100000011100000000111000001001")   # Ce
```

2.4 – Décodage du cercle 4

Le cercle 4 contient les **coordonnées géographiques** (sous la forme degrés, minutes, secondes) d'un lieu que voulait nous montrer les ingénieurs de la NASA. Les caractères 4 et 8 sont les **directions cardinales** : **N**, **S**, **E** ou **W**. Les autres caractères sont les données numériques obtenues en convertissant le code en décimal.



> **Exercice 4** :
> **1. Décoder** les informations codées sur le 4ième cercle.
> **2. Donner** la latitude et la longitude codées dans le 4ième cercle.
> **3. Convertir** ces coordonnées géographiques en coordonnées DD.
> **4. Situer**, sur une carte (https://www.openstreetmap.fr/) le lieu auquel correspondent ces coordonnées géographiques. **Donner** le nom de l'organisation à laquelle correspond ce lieu. **Indiquer** son lien avec le rover Perseverance.

> **Réponses** :

2.5 – Programme de décodage du message du quatrième cercle

L'algorithme de la fonction `decode_cercle4(code_cercle)` , qui permet le décodage du quatrième cercle, est donné ci-dessous. Cette fonction reçoit en paramètre `code_cercle` qui est la suite binaire inscrite dans le quatrième cercle et renvoie les coordonnées géographiques sous forme d'une chaîne de caractère : `"48 52 29 N 2 17 42 E"`

fonction decodeCercle4(code_cercle: Chaine de caractère) coord <-- Chaine de caractère vide Tant que la taille de code_cercle > 0 Supprimer les trois premiers bits à 0 Déterminer le caractère correspondant aux 7 bits Si le caractère est différent de N, S, E et W Déterminer la valeur décimale correspondant aux 7 bits Convertir cette la valeur décimale en chaine de caractères Ajouter le caractère ou à la chaine de caractères à coord Ajouter un espace à coord Supprimer ces 7 bits de codeCercle Renvoyer la chaine coord

> **Exercice 5** :
> **1. Editer** la fonction `decode_cercle4(code_cercle)` afin d'implémenter l'algorithme ci-dessus.

In [ ]:
```
def decode_cercle4(code_cercle):
    # A Compléter
```

> **2. Tester** votre fonction sur le code binaire du 4ième cercle et **vérifier** que l'on obtient les mêmes coordonnées géographiques obtenues dans l'exercice précédent.

In [ ]:
```
decode_cercle4("00001000100000001011000011101000000011100001110110000000\
10100000011111000000111")
```

La fonction `localise(coord)` permet d'afficher le point représenté par les coordonnées géographiques `coord` à l'aide de la bibliothèque `folium` . Les coordonnées géographiques `coord` sont passées en paramètre sous la forme d'une chaîne de caractères : `coord = '48 52 29 N 2 17 42 E'`

Les valeurs contenues dans cette chaîne de caractères vont être découpées et placées dans le tableau `tab_coord` :
`tab_coord = [latD, latM, N_S, latDir, longD, longM, longS, O_E]`

Nous verrons très bientôt les « tableaux » en NSI.

> **Exercice 6** :
> **1. Compléter** la fonction `localise(coord)` qui permet d'afficher sur une carte le lieu correspondant aux coordonnées géographiques passées en paramètre.

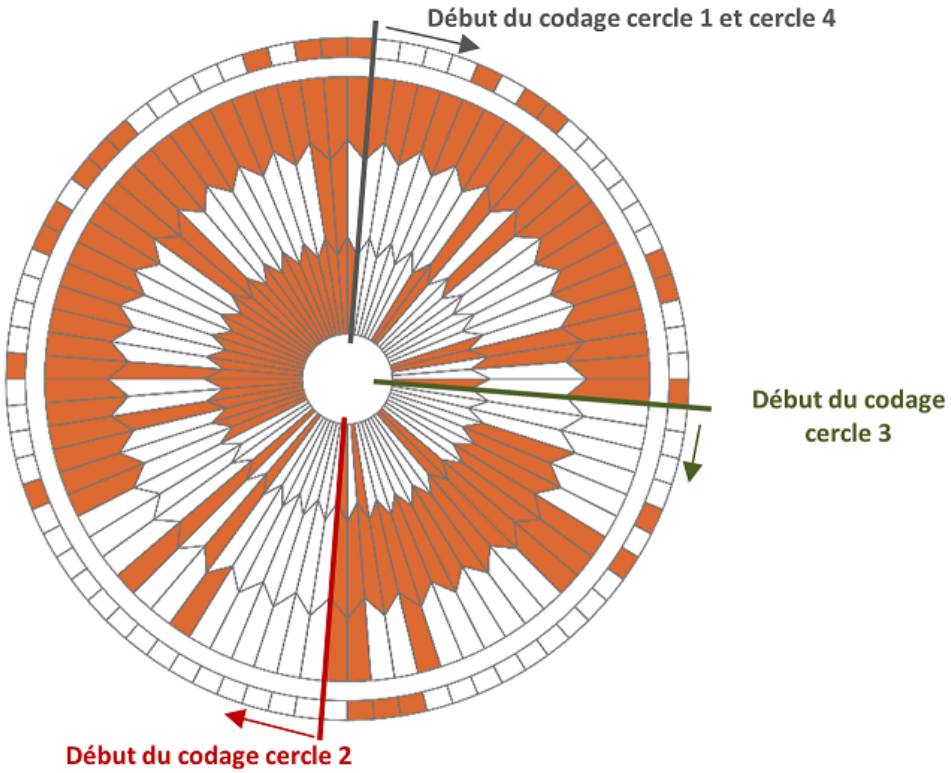
In [ ]:
```
import folium
def localise(coord):
    tab_coord = coord.split(' ')   # Découpe la chaine de caractères coord - > tab
    latD = int(tab_coord[0])       # Valeur décimale des dégrés de la latitude
    latM = int(tab_coord[1])       # Valeur décimale des minutes de la latitude
    latS = int(tab_coord[2])       # Valeur décimale des secondes de la latitude
    N_S = tab_coord[3]             # Direction cardinale de la latitude
    longD = int(tab_coord[4])      # Valeur décimale des dégrés de la longitude
    longM = int(tab_coord[5])      # Valeur décimale des minutes de la longitude
    longS = int(tab_coord[6])      # Valeur décimale des secondes de la longitude
    W_E = tab_coord[7]             # Direction cardinale de la longitude

    latDD =                        # Calcul latitute en DD
    if                             # Si la direction cardinale est SUD
        latDD =

    longDD =                       # Calcul longitude en DD
    if                             # Si la direction cardinale est l'OUEST
        longDD =

    m = folium.Map(location = [           ,           ], zoom_start=17)  # Affiche u
    folium.Marker([           ,        ]).add_to(c)                      # Affiche u
    m.save('ma_carte.html')                                               # Permet d
    return m
```

> **2. Tester** la fonction `localise(coord)` et vérifier que le lieu obtenu est identique à celui de l'exercice 3.

In [ ]:
```
coord = decode_cercle4("0000100010000000101100001110100000001110000111011000000\
1010000001111110000001111")
m = localise(coord)
m
```

4 – PARACHUTE DE SECOURS

La Nasa avait prévu un parachute de secours qui fort heureusement n'a pas eu a être utilisé. Celui-ci aussi possédait un message caché dont voici le code :



> **Exercice 7** :
> **1. Donner** le code binaire de chacun des 4 cercles.
> **2. Tester** la fonction `decode_cercle` sur les codes binaires des 3 premiers cercles afin de décoder le message. **Donner** le message codé.
> **3. Tester** la fonction **`decode_cercle4`** sur le code binaire du 4ième cercle. **Donner** les coordonnées géographiques.
> **4. Tester** la fonction **`localise`** sur le code binaire du 4ième cercle. **Indiquer** le lieu correspondant aux coordonnées géographiques codées sur le parachute.