LU1IN001 : Elements de programmation 1

Séance de tutorat n°1:

- Informations utiles
- Python 3.4.1
- Pytudes

Présentation:

Elias Bendjaballah
MIPI 2017-2018
Licence Informatique (juin 2020)
Master ANDROIDE (IA et Recherche Opérationnelle)

Page web perso:

Informations utiles:

- Cette rentrée est spéciale
  - Forte influence du Contrôle continu (CC) dans la note finale : 50% CC et 50% Examen
  - Probabilité non négligeable de continuer le semestre en distanciel
  - Toute évaluation réalisée en présentiel compte et peut être déterminante.
- Elle implique des efforts particuliers
  - Rester informé et connecté
  - Etre équipé (possibilités de prêts d'ordinateurs chez ALIAS pour le semestre)
  - Pouvoir s'adapter et continuer d'apprendre
  - Travail continu, soigné et **Formez des groupes d'études**

- Associations :
  - Alias : FB https://www.facebook.com/ALIAS.asso/
  - Association Défis Sorbonne : https://www.defis-sorbonne.fr/
  - Fablab Sorbonne Université : https://www.facebook.com/fablabsu/
  - Good game Fest : https://www.facebook.com/GoodGameFest/
  - Axio Sorbonne U : FB https://www.facebook.com/AXIOBDF

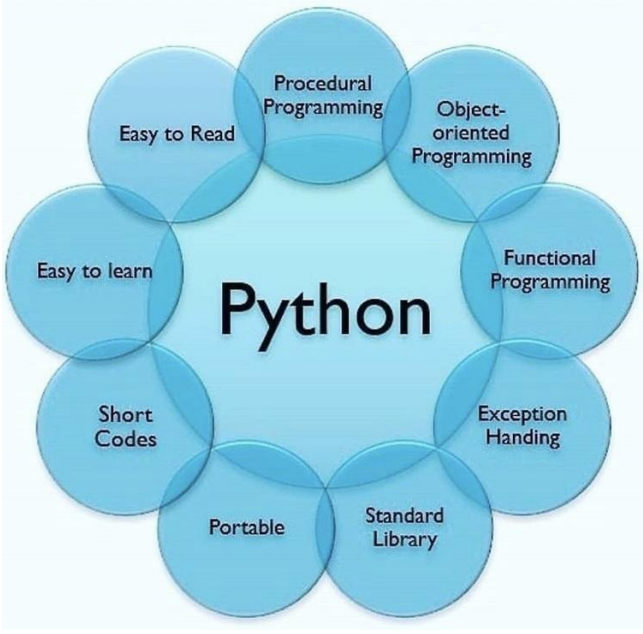
- Groupe Whatsapp : https://chat.whatsapp.com/KHzAntc3h7FBsoMklzatW1 :
- Github Repository (notebooks, codes et solutions) : https://github.com/EliasGit2017/Tutorat_LU1IN001 :

Organisation de la séance :

- **Format**: Notebooks Jupyter (possibilité de suivre la séance sur votre ordi)
- Série de problèmes à résoudre en python (similaires aux questions d'exams et de TME Solo)
- Intéractions autant que possible
- Possibilité de modifier le déroulement et la forme des séances (ouvert à toute suggestion)
- Toutes les questions sont les bienvenues
- Migration vers Discord ou Zoom en cas de distanciel (vous serez informé à temps sur **WHATSAPP**)

Python:

- Développé par Guido Von Rossum (première version créée fin 1989)
- Inspiré du langage ABC (développé au CWI d'Amsterdam), mais aussi du C et de Modula-3 (proche de Pascal)
- Avantages :
  - Lisible, épuré et intuitif
  - Peu de constructions syntaxiques
  - Optimisation de la productivité et outils de haut niveau utilisables rapidement (un peu trop haut niveau pour certains)



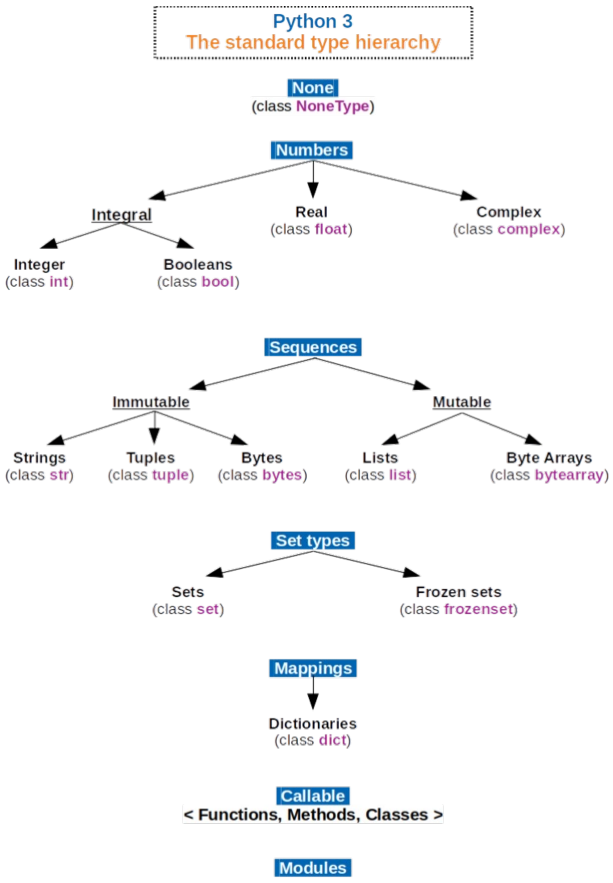
Caractéristiques:

- **Indenté** et typage dynamique fort
- Idéal pour la programmation impérative (mais aussi la fonctionnelle et la POO)
- Gestion automatique de la mémoire avec un garbage collector (les memory leaks sont limitées, ce qui n'est pas le cas du C)

- Très forte influence de la POO (aussi bien dans l'usage que dans le développement du langage)
  *Toute donnée est stockée sous la forme d'un objet

In [3]:
```
print(type([1,2,3]))
print(type('a'), "\t", type("salut"))
print(type(1) ,"\t", type(1.0))
```

```
<class 'list'>
<class 'str'>    <class 'str'>
<class 'int'>    <class 'float'>
```



Python 3.4.1:

Version 3.4 publiée le 16 mars 2014 et avec pour principale feature la correction de la vulnérabilité **HeartBleed** (qui est toujours active avec environ 180 000 appareils connectés encore vulnérables au 23 janvier 2017). Mais aussi ajout de nombreuses fonctionnalités :

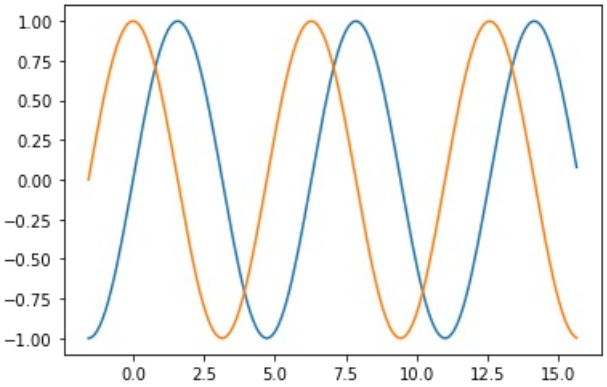
- `pip` présent par défaut
- ajout de `tracemalloc`
- ajout de la méthode `get_instructions()` au module `dis`
- ajout du module `statistics`
  Et bien d'autres fonctionalités qui en plus des avantages inhérents au langage en font un outil incontournable pour **toutes** les tâches de traitement d'information.
  Python est l'un des langages les plus utilisés dans la recherche et reste la principale solution aux projets et logiciels devant être livrés rapidement.

```
# Exemple de Script (implémentation d'une cascade de Haar)
import cv2
import numpy as np

face_cascade = cv2.CascadeClassifier('./Face_detection/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('./Face_detection/haarcascade_eye.xml')

cap = cv2.VideoCapture(0)

while(True):
    ret, img =cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for(x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h), (0,0,255),2)
        roi_gray = gray[y:y+h, x:x+w]
```

```
            roi_color = img[y:y+h, x:x+w]
            eyes = eye_cascade.detectMultiScale(roi_gray)
            for(ex,ey,ew,eh) in eyes:
                cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (255,255,0),2)

        cv2.imshow('Face Recognition (legit)',img)
        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break
    cap.release()
    cv2.destroyAllWindows()
```

In [12]:
```
# librairies math, numpy et tracé des fonctions sinus et cosinus
from math import sqrt, pi, sin, cos
import matplotlib.pyplot as plt
import numpy as np

print(cos(pi / 2))
print(sqrt(2 * pi))
x = np.arange(-pi / 2, 5 * pi, 0.1)
plt.plot(x, np.sin(x), x, np.cos(x))
plt.show()
```

```
6.123233995736766e-17
2.5066282746310002
```



Règles de cette UE:

- Pas de variable globale (`global x = 100`)
- Pas de bloc lexical
- Votre code doit garder des types simples (pas d'éléments hétérogènes) `L = [1, "chaîne", 2.0, ["inception"]]`
- Pas d'accès direct des n-uplets
- **TOUJOURS** écrire la documentation des fonctions et clairement identifier les variables utilisées

Petits tips pour vos premiers pas:

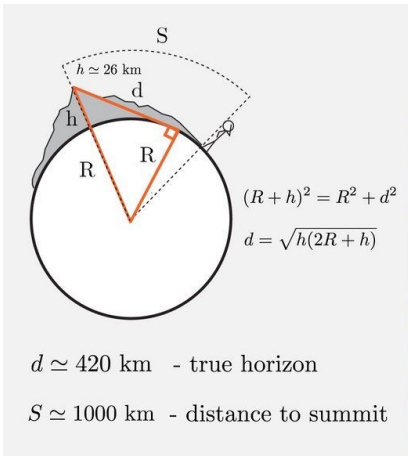
1. Faire attentions aux types de vos variables, aux opérateurs utilisés ainsi qu'aux différentes intéractions
2. Rester simple (L'UE est facile, mais demande de la rapidité et des raisonnements logiques et biens structurés)
3. Décomposer les problèmes qui paraissent compliqués en plusieurs fonctions
4. Débugger votre code à coups de `print()` et éventuellement avec `input()` (même sur MrPython)
5. Analyser les `assert` (types obtenus en sortie et traitements des différents cas).
6. Ecrire de bons tests `assert` et envisager les différents cas possibles

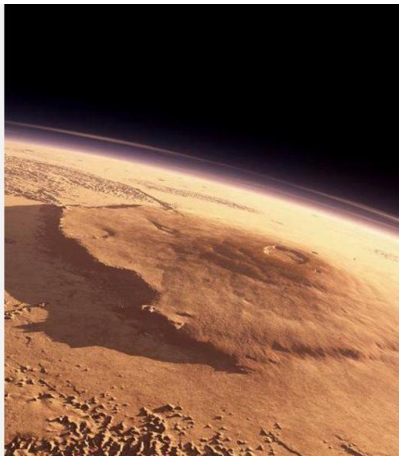
In [4]:
```
def perim(l, L):
    """float * float -> float
    hyp: l <= L
    Retourne le périmètre d'un quadrilatère de largeur l et de longueur L"""
    #return float(2 * (l + L))
    return 2 * (l + L)

assert perim(5, 6) == 22
assert perim(5, 6) == 21.999999999999999
```

In [5]:
```
assert 6 == 5.9999999999999999
```

Le mont Olympe martien:

L'un des plus hauts volcans connus du système solaire se trouve sur Mars et porte le nom de "Olympus Mons". Son altitude d'environ $26\ km$ lui donne la particularité d'avoir une pente telle qu'une personne se trouvant à sa base ne puisse voir son sommet.





Ecrire une fonction `MarsR` qui à partir de la hauteur `h` du volcan Olympe et de la distance `d` de l'horizon réel martien, retourne une estimation du rayon de Mars.

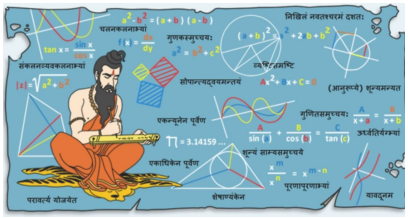
In [8]:
```
# Votre code doit être écrit ici :
```

In [9]:
```
assert MarsR(26*10**3, 420*10**3) == 3379307.6923076925
# Il est aussi possible d'appliquer cette fonction pour estimer le rayon de la Terre (et d'autres planètes)
# Pour un observateur situé en bord de mer à 1.8 m de hauteur et pour qui l'horizon se trouve à environ 4789
assert MarsR(1.8, 4789.11) == 6370992.042249999 # Rayon équatorial de la Terre
```

Une approximation de sinus vieille de 1400 ans:

Au VI^{eme} siècle, le mathématicien Mahabhaskariya de Bhaskara I a donné une approximation de sinus (pour un angle x tel que $0\leq x \leq \pi$) remarquablement précise pour l'époque.

$$\sin(x) = \frac{16 x (\pi - x)}{5\pi^{2}-4x(\pi -x)}$$

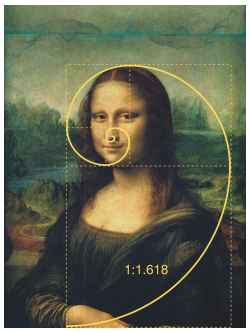


Ecrire la fonction `approSin` qui étant donné la mesure en radian d'un angle `x` tel que $0\leq x \leq \pi$ retourne l'approximation de son sinus.

In [19]:
```
# Votre code doit être écrit ici :
```

In [20]:
```
assert approSin(0.79) == 0.7091189442375202
assert approSin(1.57) == 0.9999996787427855
assert approSin(2.36) == 0.7031948565648719
```

Fibonacci (une fois n'est pas coutume) :



La suite de Fibonnaci est un classique en informatique. La suite doit son nom à Leonardo Fibonacci qui, dans un problème récréatif posé dans l'ouvrage Liber abaci publié en 1202, décrit la croissance d'une population de lapins.

« Quelqu'un a déposé un couple de lapins dans un certain lieu, clos de toutes parts, pour savoir combien de couples seraient issus de cette paire en une année, car il est dans leur nature de générer un autre couple en un seul mois, et qu'ils enfantent dans le second mois après leur naissance. »

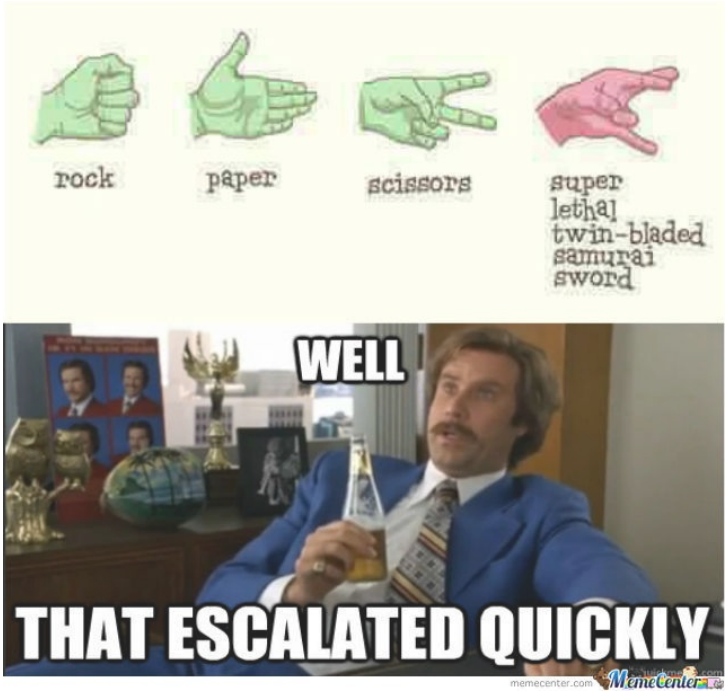
Ecrivez une fonction `fibonacci` permettant de calculer le n^{eme} terme de la suite de fibonnaci. $$ F(n) =\begin{cases} 0 & \text{si n = 0}\\ 1 & \text{si n = 1}\\ F_{n-1} + F{n-2} & \text{si n > 1} \end{cases}$$

In [1]:
```
# Votre code doit être écrit ici :
```

In [3]:
```
assert fibonacci(0) == 0
assert fibonacci(5) == 5
assert fibonacci(7) == 13
assert fibonacci(25) == 75025
```

Pierre, Papier, Ciseaux:

Très motivé par l'étude des comportements humains et fort de vos skills en python, vous entreprenez la réalisation d'une expérience de simulation de comportement afin de prédire les résultats de parties de chifoumi.
Vous décidez donc d'écrire une fonction `AI_beats_humans` qui étant donné le choix `n` de votre adversaire (0 : pierre, 1 : papier, 2 : ciseaux) retourne un choix gagnant.



In [1]:
```
# Votre code doit être écrit ici :
```

In [2]:
```
assert AI_beats_humans(2) == 0
assert AI_beats_humans(1) == 2
assert AI_beats_humans(0) == 1
```

Maintenant que vous disposez du principe de base du jeu, vous décidez d'implémenter la possibilité de réaliser plusieurs parties successives afin de mieux cerner les habitudes de votre adversaire. En fin de jeu, vous retournerez les fréquences d'utilisation de la pierre, du papier et des ciseaux de votre adversaire.
Ecrivez une fonction `AI_analyses_you` qui simule `p` parties de jeu et affiche la frequence de jeu résultante.
Le comportement de votre adversaire sera simulé par un tirage aléatoire entre les entiers 0, 1 et 2.

In [24]:
```
# Votre code doit être écrit ici :
```

All pairs that sum to target :

$$\sum$$

Ecrivez une fonction `sum_to_target` qui étant donné un nombre `n`, affiche l'ensemble des paires dont la somme est égale `n`.

```
>>> sum_to_target(7)
>>> 2+5
    3+4
    6+1
```

In [2]:
```
# Votre code doit être écrit ici :
```

In [3]:
```
assert sum_to_target(7) == None
```

```
1 + 6
```

```
2 + 5
3 + 4
```

In [4]:
```
assert sum_to_target(15) == None
```

```
1 + 14
2 + 13
3 + 12
4 + 11
5 + 10
6 + 9
7 + 8
```

Exercice 1 : Estimation de la fonction exponentielle

Préambule La fonction exponentielle **exp** : $\mathbb{R} \to \mathbb{R}$, peut s'écrire [1] comme une somme de série :

$$\mathbf{exp}(x) = \sum_{n=0}^{+\infty} \frac{x^n}{n!}$$

Dans cet exercice, on se propose de calculer une approximation de la fonction exponentielle.

Question 1.1 : [2/41]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `factorielle` qui, étant donné un entier naturel n, retourne la valeur de $n!$, définie pour $n \geq 0$ par :

$$n! = \prod_{k=1}^{n} k$$

Par convention, la factorielle de 0 vaut 1.

Par exemple :

```
>>> factorielle(0)
1
>>> factorielle(1)
1
>>> factorielle(5)
120
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.2 : [2/41]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `terme_U` qui, étant donné un nombre x et un entier naturel n, retourne la valeur de U_n^x, le terme d'indice n de la suite $(U^x)_{n\in\mathbb{N}}$, paramétrée par x, définie par :

$$U_n^x = \frac{x^n}{n!}$$

Par exemple :

```
>>> terme_u(0, 3)
0.0
>>> terme_u(3, 0)
1.0
>>> terme_u(2.0, 2)
2.0
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.3 : [4/41]

Pour un x fixé, on définit la somme partielle au rang n des termes de (U_n^x) de la façon suivante :

$$S_n^x = \sum_{k=0}^{n} U_k^x = U_0^x + U_1^x + \ldots + U_n^x \quad \text{pour tout } n \geq 0$$

Donner une définition seule de la fonction `somme_u` qui, étant donné un nombre x et un entier naturel n, retourne la valeur de S_n^x.

Par exemple :

```
>>> somme_u(0, 10)
1.0
>>> somme_u(2.0, 0)
1.0
>>> somme_u(1.0, 10)
2.7182818011463845
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.4 : [4/41]

Pour un x fixé, quand on fait tendre n vers l'infini, la valeur de S_n^x tend vers $\mathbf{exp}(x)$. Cette estimation est d'autant plus précise que n est grand.

Cela signifie, entre autres, que pour un x fixé et pour toute valeur réelle $\epsilon > 0$ donnée, il existe un

entier naturel n tel que

$$|S_{n+1}^x - S_n^x| < \epsilon.$$

Nous allons utiliser ce critère comme critère d'arrêt pour notre estimation de **exp**. C'est-à-dire, pour un ϵ et un x donné, nous allons calculer le premier indice n de la suite tel que $|S_{n+1}^x - S_n^x| < \epsilon$.

Donner une définition seule (sans signature ni hypothèse) de la fonction `rang_estim_exp` dont la spécification est la suivante :

```
def rang_estim_exp(x, epsilon):
    """ Number * float -> int
        Hypothèse : epsilon > 0.0

        retourne le premier rang n tel que la différence entre
        l'estimation de exp pour x au rang n et l'estimation
        de exp pour x au rang n+1 est plus petite que epsilon."""
```

Voici quelques exemples d'estimation :

```
>>> rang_estim_exp(2.0, 1.0)
3
>>> rang_estim_exp(1.0, 0.001)
6
```

In [ ]:
```
# Ecrivez votre code ici
```

Exercice 1 : Suites de Lehmer

En 1948, le mathématicien D.H. Lehmer a proposé un algorithme, utilisant une suite numérique, pour générer des nombres pseudo-aléatoires (comme, par exemple, ce qui est utilisé dans la librairie Random vue en TME). Cet exercice propose de programmer des fonctions Python implémentant cette suite numérique.

Une *suite de Lehmer* est définie, pour tout n entier naturel, par :

$$u_n = \begin{cases} v & \text{si } n = 0 \\ (a \cdot u_{n-1} + b) \bmod m & \text{si } n \geq 1 \end{cases}$$

avec m, un entier naturel non nul, et a, b, et v, trois entiers naturels supérieurs ou égaux à 0 et strictement inférieurs à m. On peut noter que l'utilisation du modulo (a mod b désigne le reste de la division euclidienne de a par b) fait que les valeurs prises par u_n sont forcément comprises entre 0 et $m - 1$.

Dans cet exercice, de façon générale, étant donné m, a, et b, on appelle *suite de Lehmer de v* une telle suite.

Par exemple, pour $m = 8$, $a = 1$, $b = 3$, la suite de Lehmer de $v = 4$ (ou plus simplement, suite de Lehmer de 4) est la suite 4, 7, 2, 5, 0, 3, 6, 1, 4, 7...

Pour $m = 10$, $a = 3$, et $b = 1$, la suite de Lehmer de 2 est 2, 7, 2, 7...

Pour $m = 10$, $a = 5$, et $b = 1$, la suite de Lehmer de 2 est 2, 1, 6, 1...

Remarque : dans cet exercice, on utilisera l'opérateur `%` de Python pour le modulo (mod). Ainsi, x mod y sera écrit `x % y` en Python.

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.1 : [2/61]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `suivant` qui, étant donné 4 entiers naturels : m, a, b et u, le n-ième terme de la suite de Lehmer, rend la valeur du terme qui suit u dans la suite de Lehmer.

Par exemple :

```
>>> suivant(8,1,3,0)
3
>>> suivant(10,3,1,2)
7
>>> suivant(10,5,1,1)
6
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.2 : [3/61]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `terme` qui, étant donné 5 entiers naturels : m, a, b, v, et n rend la valeur du terme u_n de la suite de Lehmer correspondante.

Par exemple :

```
>>> terme(8,1,3,4,0)
4
>>> terme(8,1,3,4,5)
3
>>> terme(10,3,1,2,4)
2
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.3 : [3/61]

Une suite $(u_n)_{n\in\mathbb{N}}$ est *périodique* quand il existe $k > 0$, tel que pour tout n, $u_{n+k} = u_n$. Dans ce

cas, le plus petit entier k vérifiant cette propriété est appelé *période de la suite*.

Dans le cas particulier des suites de Lehmer, on peut montrer qu'une suite de Lehmer de v est périodique si et seulement si l'un des m termes qui suivent u_0 est égal à v. Dans ce cas, la période de la suite est le nombre de termes existants entre u_0 et le premier terme suivant qui prend la valeur v, en le comptant.

Ainsi, la suite de Lehmer de v n'est pas périodique si aucun des m termes qui suivent u_0 n'est égal à v.

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `est_periodique` qui, étant donné 4 entiers naturels m, a, b, et v rend le booléen vrai si la suite ainsi définie est périodique, ou le booléen faux sinon.

Par exemple :

```
>>> est_periodique(8,1,3,4)
True
>>> est_periodique(10,3,1,2)
True
>>> est_periodique(10,5,1,2)
False
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.4 : [4/61]

Donner une définition avec signature et hypothèse(s) éventuelle(s) de la fonction `periode` qui, étant donné 4 entiers naturels m, a, b, et v qui définissent une suite de Lehmer de v qui est périodique, rend la valeur de la période de cette suite.

Par exemple :

```
>>> periode(8,1,3,4)
8
>>> periode(10,3,1,2)
2
```

In [ ]:
```
# Ecrivez votre code ici
```

Question 1.5 : [3/61]

Une suite de Lehmer est dite *pleine* si elle prend m valeurs différentes, ce qui équivaut à dire qu'elle est de période m.

En utilisant la fonction `periode`, donner une définition avec signature et hypothèse(s) éventuelle(s) du prédicat `est_pleine` qui, étant donné 4 entiers naturels m, a, b, et v rend le booléen vrai si la suite ainsi définie est pleine, ou le booléen faux dans le cas contraire.

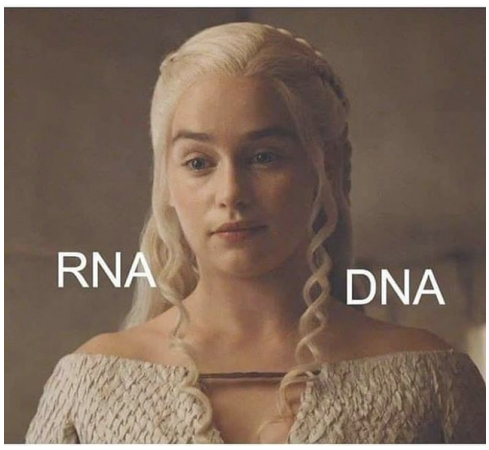
Par exemple :

In [ ]:
```
# Ecrivez votre code ici
```

```
>>> est_pleine(11,2,1,5)
False
>>> est_pleine(10,3,1,7)
False
>>> est_pleine(10,5,1,2)
False
>>> est_pleine(256,25,15,12)
True
```

In [ ]:
```
# Ecrivez votre code ici
```

Traduction en mRNA:



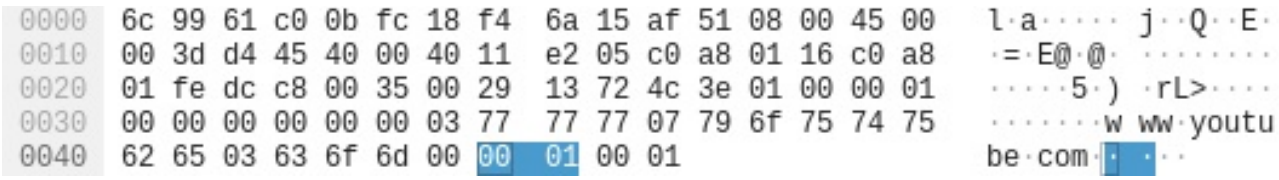
Ecrire une fonction `DNA_to_mRNA` qui étant donné une séquence de nucléotides `adn` (sous forme de chaîne de caractères formées par les bases A,T,C,G) retourne la séquence d'ARN messager correspondante. Observez les `assert` pour en déduire les règles de conversion ADN -> ARNm.

In [2]:
```
# Votre code doit être écrit ici :
```

In [2]:
```
assert DNA_to_mRNA("ACTGCCAATTTGGACCC") == "UGACGGUUAAACCUGGG"
assert DNA_to_mRNA("GTCATACGACGTA") == "CAGUAUGCUGCAU"
assert DNA_to_mRNA("AGGCAAAXTGG") == "Erreur, nucleotide inconnu...veuillez retourner sur Mars"
```

Conversions Bin, Oct, Hex, Dec:

Convertir les données vers les différentes bases est une opération courante. Bien que les conversions soient implémentées nativement dans la plupart des langages, il est nécessaire de maîtriser les représentations en bases 2, 6 et 8 et de pouvoir les lire rapidement.



Requête DNS pour youtube.com

Ecrivez les fonctions `dectobin`, `bintodec`, `dectohex`, `hextodec`, `dectooct`, `octtodec` qui réalisent les conversions d'un entier `n` entre le système decimal et les systèmes binaire, hexadecimal, octal et vice-versa.
Exemple :
42 :

- 0b101010 en binaire
- 0x2a en hexadecimal
- 0o52 en octal

In [4]:
```
# Votre code doit être écrit ici :
```

In [5]:
```
assert dectobin(42) == "101010"
assert dectobin(420) == "110100100"
assert dectobin(6857) == "1101011001001"
assert dectobin(11) == "1011"
```

In [6]:
```
# Votre code doit être écrit ici :
```

In [7]:
```
assert bintodec("101010") == 42
assert bintodec("110100100") == 420
assert bintodec("1101011001001") == 6857
assert bintodec("1011") == 11
```