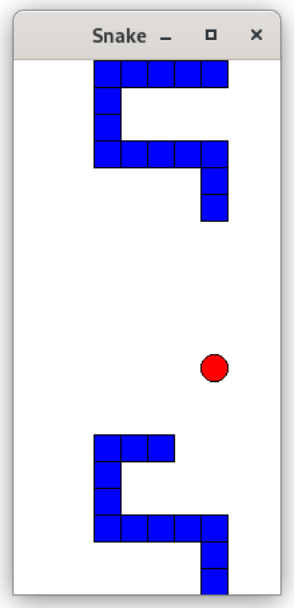
À rendre au plus tard le 05/05/2024 à 20h à l'adresse mail habituelle.
Faites un et un seul des deux exercices de ce document.

Exercice 1. Jeu du serpent

Le but du sujet est de programmer le jeu « snake » :

`http://www.jeux.org/jeu/snake-nokia-3310.html`



Il ne s'agit pas d'obtenir un jeu parfaitement identique à celui du lien ci-dessus, mais simplement un jeu basé sur le même principe :

→ Un serpent se déplace sur une grille rectangulaire dont les bords gauche et droit (resp. haut et bas) sont connectés.

→ À chaque étape il se déplace d'une case vers le haut, vers la droite, vers la gauche ou vers le bas.

→ Tant que le joueur ne lui demande pas de changer de direction, le serpent se déplace tout droit.

→ Le but est de récolter des pommes. À chaque fois qu'une pomme est récoltée, le serpent augmente de taille.

→ Si le serpent mord sa propre queue, la partie est perdue.

Pour l'interface graphique, utilisez le module `tkinter` (voir le TP 10 et l'exercice 5 du TP 1). On pourra s'inspirer du fichier annexe disponible sur la page du cours :

`https://informatique-lhp.fr/itc-mpsi.html`

Exercice 2. Étude d'une « suite univers »

Rappels sur quelques points de syntaxe en Python.

⋆ On peut convertir un entier en chaîne de caractères, et vice-versa.

```
In [1]: x,y = 14,18

In [2]: str(x)
Out[2]: '14'
```

```
In [3]: str(x)+str(y)
Out[3]: '1418'

In [4]: a,b = '39','45'
```

```
In [5]: int(a)
Out[5]: 39

In [6]: int(a)+int(b)
Out[6]: 84
```

Ne pas faire de confusion entre un entier comme `x = 22`, et la chaîne `str(x)` qui vaut `"22"` !

⋆ On peut extraire les k derniers éléments d'une liste ou d'une chaîne de caractères :

```
In [7]: x = 'bonjour'
```

```
In [8]: x[-4:]
Out[8]: 'jour'
```

```
In [9]: x[-20:]
Out[9]: 'bonjour'
```

⋆ On dit que `y` est une sous-chaîne de `x` si les caractères de `y` figurent de manière consécutive dans `x`. Par exemple `"format"` est une sous-chaîne de `"informatique"`.

⋆ Un dictionnaire contient des couples `clé: valeur`. On peut facilement lire la valeur d'une clé, et affecter une valeur à une clé (déjà existante ou pas) :

```
In [10]: D = {1:'un', 2:'deux', 3:'trois', 4:'quatre'}

In [11]: D[2]
Out[11]: 'deux'

In [12]: D[1] = 'une'

In [13]: D
Out[13]: {1: 'une', 2: 'deux', 3: 'trois', 4: 'quatre'}

In [14]: D[5] = 'cinq'

In [15]: D
Out[15]: {1: 'une', 2: 'deux', 3: 'trois', 4: 'quatre', 5: 'cinq'}
```

1 La suite de Champernowme

Une ***suite univers*** en base 10 est une suite u de chiffres (de 0 à 9) telle que toute séquence finie de chiffres apparaît comme sous-suite formée de termes consécutifs de cette suite. Par exemple on soupçonne que les décimales de π forment une suite univers (mais ceci n'a jamais été prouvé). Si c'est le cas alors toute séquence, quelle que soit sa longueur, se trouve dans les décimales de π. Prenons par exemple la séquence 419 :

$$3141592653589793238462643383279502884\underline{419}7169399375\ldots \qquad \text{c'est gagné !}$$

La ***suite de Champernowme*** offre un exemple de suite univers. On l'obtient par concaténation des chiffres composant les entiers naturels non nuls, pris dans l'ordre croissant :

$$01234567891011121314151617181920 21\ldots$$

On définit ainsi la suite C :

$$C = (0,1,2,3,4,5,6,7,8,9,1,0,1,1,1,2,1,3,1,4,1,5,1,6,1,7,1,8,1,9,2,0,2,1,2,\ldots).$$

On comprend aisément que C est une suite univers, par construction. Pour $i \in \mathbb{N}$ on note $x_i \in [\![0,9]\!]$ le i-ème terme de C. Par exemple $x_9 = 9$, $x_{10} = 1$, $x_{15} = 2$ ...

Dans cet exercice, nous décidons de travailler avec des chaînes de caractère. Ainsi la séquence $(x_0, x_1, \ldots, x_{20})$ sera codée par la chaîne `"01234567891011121314 1"`.

1. Écrire une fonction `champ(N:int) -> str`, qui prend en argument un entier naturel `N` et qui renvoie la chaîne de caractères obtenue en concaténant les entiers entre `0` et `N`.

```
In [17]: champ(21)
Out[17]: '012345678910111213141516171819202 1'
```

2. Écrire une fonction `champ2(N:int) -> str` qui prend en argument un entier naturel `N` et qui renvoie $(x_0, x_1, \ldots, x_N)$ sous forme de chaîne de caractères.

```
In [19]: champ2(30)
Out[19]: '0123456789101112131415161718192'
```

2 La suite de Champernowme réduite, algorithme naïf

On peut remarquer que la séquence `"12"` (par exemple) apparaît dès le début de la séquence C, mais qu'elle a quand même été réécrite un peu plus loin, ce qui n'est pas nécessaire pour obtenir une suite univers :

$$0\underline{12}345678910111\underline{12}131415161718192.$$

La ***suite de Champernowme réduite*** est définie de manière similaire à celle de Champernowme. Ses termes sont obtenus en concaténant, pour $n \in \mathbb{N}$, la séquence des chiffres composant n, mais uniquement si cette séquence ne figure pas déjà dans les termes précédents :

$$C_r = (0,1,2,3,4,5,6,7,8,9,1,0,\underline{1,1,1,3},1,4,1,5,1,6,1,7,1,8,1,9,2,0,2,1,\underline{2,2,2,4},2,5,2,$$

$$6,2,7,2,8,2,9,\underline{3,0,3,2},\underline{3,3,3,5},3,6,3,7,3,8,3,9,\underline{4,0,4,3},4,4,\ldots).$$

On souhaite écrire une fonction `champred(N:int) -> str` qui prend en argument un entier naturel `N` et qui renvoie les termes de C_r obtenus en en allant jusqu'à l'écriture (éventuelle) des chiffres de l'entier `N` :

```
In [2]: champred(200)
Out[2]:
'01234567891011131415161718192021222425262728293032333536373839404344464748495054555758596065666869707677798087889099100102103104105106107108109110112114115116117118119120124125126127128129130132133134135136137138139140142143144145146147148149150152153154155156157158159160162163164165166167168169170172173174175176177178179180182183184185186187188189190193194195196197198199200'
```

3. Écrire une fonction `cherche(mot:str,texte:str) -> bool` qui renvoie `True` si la chaîne `mot` est une sous-chaîne de la chaîne `texte`, et `False` sinon.

```
In [21]: cherche('pa','barbapapa')
Out[21]: True

In [22]: cherche('pa','barbamama')
Out[22]: False
```

4. On considère dans ce problème un test d'égalité entre deux chaînes comme une opération élémentaire (coût constant), quelle que soit les longueurs des chaînes. Justifier que la complexité de la fonction `cherche` est un $\mathcal{O}(n)$, où n est la longueur de la chaîne `texte`.
5. Compléter la fonction ci-contre pour qu'elle renvoie le résultat attendu.

```
def champred(N):
    C = .....
    for k in ...... :
        if ............:
            C = C + str(k)
    return C
```

6. Quelle est la complexité de la fonction `champred` ?

3 La suite de Champernowme réduite, deuxième algorithme

On cherche maintenant à écrire une fonction `champred2(N:int) -> str`, qui renvoie le même résultat que `champred`, mais de manière plus efficace. On propose de stocker, au cours de l'algorithme, les séquences déjà écrites dans une variable dédiée, de manière à pouvoir tester rapidement si la séquence des chiffres d'un entier donné a déjà été écrite ou pas.

7. Expliquer pourquoi dans la fonction `champred2`, si `N` est de longueur `L`, alors on peut se contenter de stocker les séquences déjà écrites de longueur inférieure ou égale à `L` uniquement.

On envisage alors deux possibilités :

A - stocker simplement les séquences déjà écrites comme éléments d'une liste `L`.

B - stocker la même information dans un dictionnaire `D`, où `D[i]` serait une valeur booléenne indiquant si le nombre `i` a déjà été écrit.

Par exemple pour `N = 100`, après avoir écrit les nombres jusque 3 (c'est-à-dire 0123), l'état de ces variables serait :

```
L = ['0','1','01','2','12','012','3','23','123']
D = {'0':True, '1':True, '2':True, '3':True, '4':False, ..., '9':False,
        '10':False, '11':False, '12':True, '13':False, ...,'100':False}
```

8. Quel est le coût de la recherche d'une séquence `s` dans la liste `L`, en fonction de la longueur de `L` : constant, linéaire ou quadratique ? Quel est le coût pour obtenir la même information avec le dictionnaire `D` ?

On décide dans la suite de retenir la solution B -

9. Au début de l'algorithme, les valeurs dans `D` doivent toutes être `False`. Écrire une fonction `initdico(N:int) -> dict` qui renvoie cette valeur initiale du dictionnaire.

```
In [3]: initdico(100)
Out[3]: {'0': False, '1': False, '2': False, '3': False, ..., '100': False}
```

10. Écrire une fonction `longueur(N:int) -> int`, qui renvoie le nombre de chiffres qui composent l'entier `N`. Dans la suite ce nombre de chiffres sera noté M.

```
In [4]: longueur(2000)
Out[4]: 4
```

11. Montrer qu'il existe une constante α telle pour tout $N \geqslant 2$: $M \leqslant \alpha \ln(N)$.

Dans notre algorithme, pour chaque entier k qu'on concatène (si on le concatène), la variable `D` doit être actualisée. Pour cela il faut déclarer comme déjà écrites toutes les séquences de longueur $\leqslant M$ qu'on fait apparaître après la concaténation de chaque chiffre composant l'entier k.
Exemple. Pour $N = 100$ (donc $M = 3$), lors de la concaténation de $k = 25$:

$$012345678910111314151617181920212224\underline{\mathbf{25}}$$

Il faut déclarer écrites les séquences suivantes :

- `"2"`, `"42"`, `"242"` après concaténation du chiffre 2,
- `"5"`, `"25"`, `"425"` après concaténation du chiffre 5.

12. Compléter la fonction suivante, qui effectue le travail décrit ci-dessus :

```
def champred2(N):
    M = longueur(N)
    # initialisations :
    C = ''
    D = ............
    for k in ..............:
        if D[str(k)] == ........ :
            for chiffre in str(k):
                # ajout de chiffre dans C :
                C = .........
                for j in range(1,M+1):
                    # déclarer la séquence des j derniers
                    # caractères de C comme déjà vue :
                    D[...........] = ..........
    return C
```

13. Montrer que la complexité de la fonction `champred2` est en $\mathcal{O}(N \ln^2(N))$. Comparer avec la fonction écrite en 5.

4 Statistiques

14. Modifier la fonction `champred2` pour qu'elle renvoie non pas `C` mais le nombre X d'entiers, parmi les entiers $k \in [\![0, N]\!]$, qui ont effectivement été concaténés (nous avons vu plus haut que c'est par exemple le cas de 13, mais pas de 12). Appelez la fonction obtenue `champred2_bis`

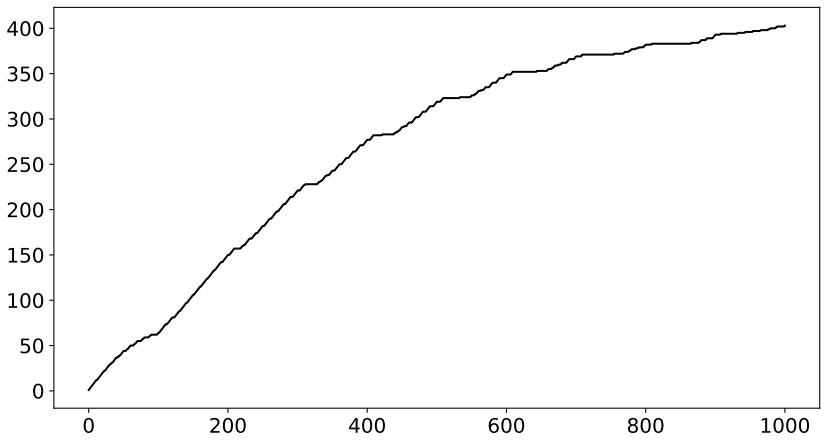


FIGURE 1 – pour faire joli : X en fonction de N

15. On souhaite comparer les fréquences d'apparition des chiffres 0, 1, 2, ... , 9 dans le résultat renvoyé par `champred2(N)`. Écrire pour cela une fonction `comptage(chaine:str) -> dict` qui renvoie un dictionnaire donnant le nombre d'apparition de chaque chiffre dans `chaine`.

```
In [6]: comptage('64511111136516581114635111115361615841')
Out[6]: {'0': 0, '1': 17, '2': 0, '3': 3, '4': 3, '5': 6, '6': 6, '7': 0, '8': 2, '9': 0}
```