Mémoriser n'est pas comprendre !
Comprendre n'est pas mémoriser !

Relire ne permet pas de mémoriser.

Pour mémoriser,
il faut **se poser des questions**
(elles sont dans la colonne de gauche) et **y répondre** !

Je **compare** alors ma réponse
(je fais l'effort de me souvenir)
à la réponse qui est dans la colonne de droite afin de **consolider**
ou **corriger** mon souvenir.

Pour comprendre, il faut savoir !

Cette page est faite pour être pliée en deux dans le sens de la hauteur, pour cacher la colonne de droite et on la fait glisser au fur et à mesure.

Mémorisation – 1ère – Spécialité NSI

Chapitre 1 – Python – Variables, affectation, entrées / sorties

Dans un programme, on a besoin d'accéder à des données enregistrées dans la mémoire ou de stocker des valeurs. Cela se fait grâce à des **variables**. Le **nom de la variable** sert de référence à une certaine zone mémoire de l'ordinateur dans laquelle est stocké le **contenu** de la variable.
Ainsi l'instruction `x = 2` signifie qu'il y a une zone mémoire contenant la valeur 2 dont le chemin d'accès est cachée derrière le nom `x`. **Déclarer** une variable consiste à réserver une zone de la mémoire. La taille de cette zone mémoire dépend évidemment de ce qu'on veut y stocker. C'est pour cela qu'en python, une déclaration de variable est toujours accompagnée d'une **initialisation**.
Une fois qu'une variable a été déclarée et initialisée (`x = 2`), on peut l'utiliser. L'instruction `x = x+1` va procéder à l'**évaluation** du membre de droite (`2+1=3`) et va affecter ce résultat dans la zone mémoire associée au nom `x`.

1	Une variable python commence par	une lettre ou _, mais pas par un chiffre
2	Qu'est-ce que le type d'une variable ?	La nature de l'information stockée dans la variable
3	Quels sont les 4 types de base en python ?	**str** = chaîne de caractère **int** = entier (integer) **float** = nombre réel **boolean** = booléen (True / False)
4	Quelle est la fonction python qui renvoie le type d'une variable ?	La fonction `type()`
5	Opérateur d'affectation ? Affectation ? ordre important ?	opérateur `=` variable = valeur ou expression (ordre) nom de variable toujours à gauche
6	Incrémenter / décrémenter une variable (+ opérateurs)	ajouter ou soustraire 1 au contenu de la variable (opérateurs `+=` ou `-=`
7	Déf : transtypage d'une variable	changer le type d'une variable
8	Qu'est-ce qu'une expression ?	Ensemble de variables combinées à l'aide d'opérateurs

Une expression informatique est en réalité bien plus générale que la première ébauche donnée ici : une expression est tout ce qui a une valeur : elle peut contenir une expression conditionnelle (comme `x = 1+3 if 2>1 else 5`), une boucle, une fonction ...

9	Quelle fonction python permet de récupérer ce qui est entré par l'utilisateur au clavier ?	La fonction `input()`
10	Quelle fonction python permet d'afficher le contenu d'une variable ?	La fonction `print()`

Chapitre 2 – Python – Chaîne de caractère

Une chaîne de caractère (*string* en anglais et de type `str` en python) permet de représenter des textes. Une chaîne vide se déclare à l'aide de guillemets simple ou double. Une chaîne vide se note `''` ou `""`. On peut déclarer une chaîne de caractère sur plusieurs lignes à l'aide de guillemets triples `"""`.
Si une chaîne de caractère contient une apostrophe, on la déclare alors à l'aide de guillemets doubles (`"l'oiseau"`).

11	Pourquoi dit-on qu'une chaîne est itérable ?	On peut la parcourir caractère par caractère

12	Miroir d'une chaîne de caractère `chaine`	`miroir = ""`
		`for caractere in chaine :`
		`  miroir = caractere + miroir`

13	Nombre de caractère dans une chaîne `s` ? L'index d'un caractère dans une chaîne `s` varie entre ____ et ____.	`len(s)` 0 / `len(s)-1`

14	`s='azertyuiop'`
	donner `s[2]`
	donner `s[-2]`
	donner `s[:2]`
	donner `s[:-2]`
	donner `s[2:4]`
		`s[2]='e'` (3ème élément)
		`s[-2]='o'` (en partant de la fin)
		`s[:2]='az'`
		`s[:-2]='azertyui'`
		`s[2:4]='er'` (indice de fin exclu)

15	concaténer 2 chaînes de caractères ? opérateur de concaténation ?	ajouter la 2ème chaîne après la première opérateur `+`

Chapitre 3 – Python – Conditions

16	Quelle est la structure complète d'un bloc test en python ?	`if condition1 :`
		`  bloc1`
		`elif condition2 :`
		`  bloc2`
		`else :`
		`  bloc3`

Il faut bien comprendre quel bloc est évalué et à quel moment. Si `condition1 == True` alors `bloc1` est évalué, mais ni `condition2` ni les `bloc2` et `bloc3` ne sont évalués. Si `condition1 == False` alors `condition2` est évaluée : si elle est vraie, alors `bloc2` est évalué (mais pas `bloc1` et `bloc3`). Si aucune des deux conditions ne sont vraies, alors `bloc3` sera le seul à être évalué.

17	Quels sont les opérateurs de comparaison `égal`, `différent` et `appartient` ?	égal : `==`, différent : `!=` appartient : `in`
18	Que vaut l'expression `cdt1 and cdt2` ?	Elle est **vraie** si les 2 conditions le sont
19	Que vaut l'expression `cdt1 or cdt2` ?	Elle est **vraie** si l'une des 2 conditions l'est
20	Que vaut l'expression `not cdt` ?	Elle est **vraie** si la condition `cdt` est fausse

Chapitre 4 – Python – Boucles

21	Structure d'une boucle `for`	`for variable in liste :` `  bloc`

22	Structure d'une boucle `while`
	
	Que faut-il vérifier dans la boucle `while` ?
		`while condition :`
		`  bloc`
		
		Que `condition` devienne *fausse*
		à un moment pour sortir de la boucle

23	L'instruction ____ fait sortir de la boucle, l'instruction ____ passe à l'itération suivante.	`break` `continue`

24	Que représentent `i` et les valeurs a,b,c dans l'instruction `for i in range(a,b,c)` ?
	Que renvoie `list(range(5))` ?
	Que renvoie `list(range(2,5))` ?
	Que renvoie `list(range(10,2))` ?
	Que renvoie `list(range(10,1,-3))` ?
	Que renvoie `list(range(4,10,2))` ?
		i parcourt toutes les valeurs entières depuis a inclu jusqu'à b exclu par pas de c
		`[0, 1, 2, 3, 4]`
		`[2, 3, 4]`
		`[]`
		`[10, 7, 4]`
		`[4, 6, 8]`

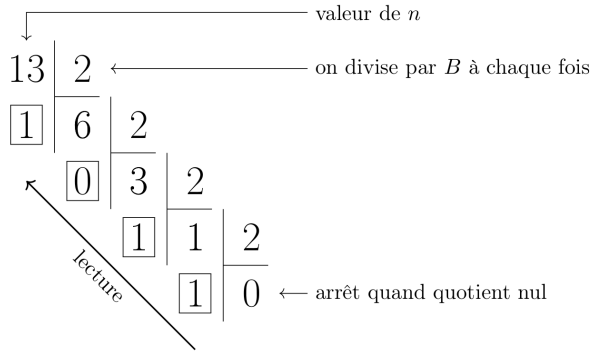
Chapitre 5 – Spécifications d'un programme

La programmation informatique devient très rapidement une question de travail en équipe. Comment alors savoir ce qu'on doit programmer ?
On découpe le programme en bloc élémentaire (chaque bloc élémentaire correspondant souvent à une fonction) et on précise ce qui sera fourni en entrée (et sous quelle forme, soit le formattage des données). Ensuite, on précise ce qu'on doit obtenir (et sous quelle forme).
Chaque informaticien programme un ou plusieurs blocs élémentaires, puis le chef de projet réalise l'intégration du travail de chacun dans le programme.

25	Quelles sont les spécifications d'un programme ? (2 noms + définitions)	Les **pré-conditions** (les données en entrée du programme et leur formattage) et les **post-conditions** les données en sortie du programme et leur formattage
26	Avant de coder un programme sur ordinateur, ...	il faut le découper en bloc élémentaire ; et pour chaque bloc, préciser les spécifications

Chapitre 6 – Représentation d'un entier naturel n dans une base B

Connaître la représentation d'un entier n dans une base B donnée, c'est être capable de donner les chiffres (ou symboles) qui les composent, et ce, dans le bon ordre. En base binaire ($B = 2$), les symboles utilisés sont 0 et 1, en base décimale ($B = 10$), les symboles sont les chiffres de 0 à 9 et en base hexadécimale ($B = 16$), les symboles sont les chiffres de 0 à 9, suivis des lettres A à F.
On appelle **poids** d'un chiffre dans la représentation obtenue sa position, comptée depuis la droite, et valant 0 pour le chiffre des unités.
Un nombre binaire est donc une suite de **bit** (contraction de binary digit) comme $001001010_{(2)}$ comprenant le bit de poids faible (celui de poids nul) et le bit de poids fort (le premier bit non nul depuis la gauche).



$$13_{(10)} = 1101_{(2)} = \mathrm{D}_{(16)}$$

Algorithme 1 : représentation d'un entier n dans la base B

Nécessite en entrée : un entier n non nul et une base B
Garantie en sortie : la représentation de l'entier n en base B sous forme d'une série de chiffre

1: quotient $\leftarrow n$
2: **Tant que** quotient est non nul **faire**
3: chiffre $\leftarrow$ reste division euclidienne de quotient par B # à sauvegarder
4: quotient $\leftarrow$ quotient division euclidienne de quotient par B
5: **Fin Tant que**

27	Comment savoir si un nb binaire est pair ?	bit de poids faible nul
28	Avec N bits, on peut coder ____ valeurs comprises entre ____ et ____.	2^N valeurs comprises entre 0 et $2^N - 1$

29 | Comment convertir un nb binaire en hexadécimal ? | On groupe les bits par 4 en commençant par la droite, à chaque groupe son code hexadécimal correspondant

A partir de la représentation d'un entier dans une base donnée, comment retrouve-t-on la valeur de l'entier n en base 10 ?

$$1101_{(2)} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{(10)}$$

Algorithme 2 : valeur d'un entier n à partir de sa représentation en base B

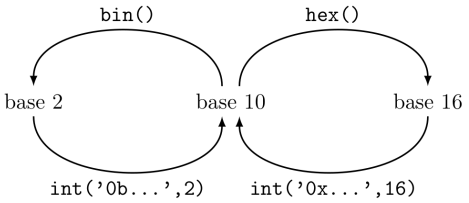
Nécessite en entrée : les chiffres de la représentation de n en base B

Garantie en sortie : la valeur de n via la variable `somme`

1: somme $\leftarrow 0$
2: poids $\leftarrow 0$
3: **Pour** chacun des chiffres en commençant par celui de poids nul à droite **faire**
4: somme $\leftarrow$ somme $+$ chiffre $\times B^{\text{poids}}$
5: poids $\leftarrow$ poids $+1$
6: **Fin Pour**

30 | `[Python]` Les entiers en base 10 sont de type `int`, tandis que les entiers en base 2 et 16 sont de type `str` commençant par `'0b...'` en base 2 et `'0x...'` en base 16.

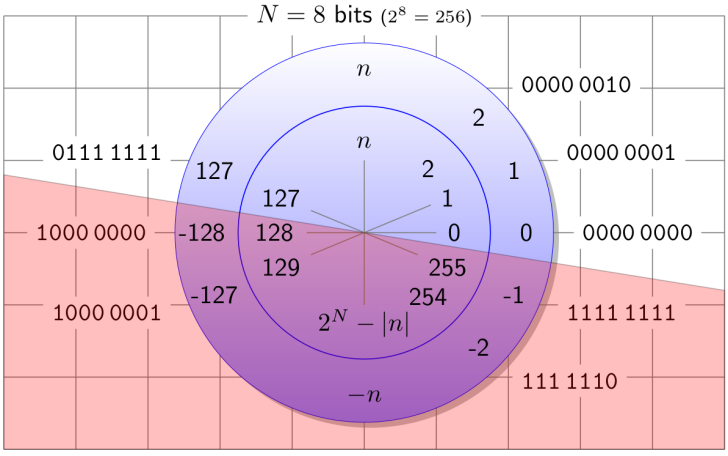
31 | `[Python]` Comment passer d'une base à une autre ? schéma $2 \leftrightarrow 10 \leftrightarrow 16$



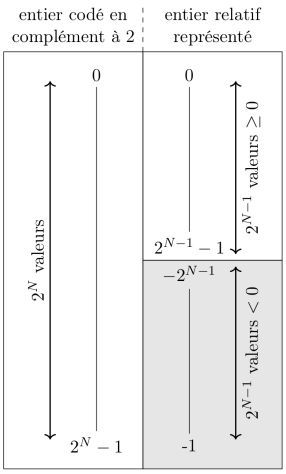
Chapitre 7 – Réprésentation des entiers relatifs

Un entier relatif est représenté en complément à deux **par un entier naturel non signé** afin de préserver les algorithmes d'addition (une soustraction = 1 complément et 1 addition) et de multiplication (additions et décalages), sous certaines réserves (on oublie l'éventuelle retenue sur le bit de poids fort pour l'addition, ...).

entier non signé	représentation binaire sur 8 bits	entier signé
0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
	...	
127	0111 1111	127
128	1000 0000	-128
129	1000 0001	-127
	...	
254	1111 1110	-2
255	1111 1111	-1



Les exemples ci-dessus sont donnés sur $N = 8$ bits. Si la somme de deux entiers relatifs dépasse $127 = 2^{N-1} - 1$, il y a débordement et le résultat sera faux. De même si le résultat est inférieur à $-128 = -2^{N-1}$. Une telle situation se repère facilement : les 2 opérandes ont le même signe, mais le résultat est de signe opposé.

32	On connaît la représentation d'un entier relatif $n > 0$ sur N bits. Comment trouver celle de $-n$?	Procédure du complément à 2 dans l'ordre : 1) inverser les N bits 2) ajouter 1 (oubli éventuelle retenue sur bit de poids fort)
33	Schéma entre l'entier relatif à représenter et sa correspondance en complément à 2	

Chapitre 8 – Python – Les fonctions

Une fonction permet d'isoler une suite d'instructions qui revient plusieurs fois dans un programme : cela évite de l'écrire à plusieurs reprises et d'avoir à la corriger à plusieurs endroits si on a fait une erreur. Une fonction est définir par son nom et ses arguments. Les instructions peuvent (ou non) dépendre de paramètres appelés *arguments formels*. A chaque appel de la fonction, le programme transmet une *valeur effective* à chaque argument formel. Eventuellement, la fonction peut *retourner* un résultat.
On peut passer une fonction en argument à une autre fonction, on peut définir une fonction à l'intérieur d'une fonction et on peut appeler une fonction à l'intérieur d'elle-même (on a alors une fonction récursive).

Spécifités du langage python : une fonction qui ne retourne rien retournera en réalité `None`. Une fonction se documente via sa `docstring`. Si besoin, on peut écrire une chaîne de caractère sur plusieurs lignes avec les guillements triples `"""`.

34	Quelle est la structure complète d'une fonction en python ?
	
	Comment appeler cette fonction dans un programme ?
	Comment obtenir de l'aide sur cette fonction ?
		`def nom_fonction (paramètres) :`
		`  """ docstring """`
		`  ...`
		`  return résultat`
		
		`nom_fonction(valeurs des paramètres)`
		
		`help(nom_fonction)`

35	Qu'est-ce que la portée d'une variable ? Une variable peut être de portée ____ dans une fonction ou de portée ____ à tout le programme.	C'est la « zone » du programme où on peut l'utiliser. locale globale

Chapitre 9 – Algorithmique – Partie 1

Un algorithme, c'est la recette de cuisine que l'on va suivre pour résoudre un problème : on veut obtenir une solution **exacte**. Et si possible, une solution efficace. En effet, on sait que certains problèmes ne peuvent pas être résolus en un temps humain (100 ans) avec des ordinateurs. Se pose donc la question de l'efficacité d'un algorithme. C'est en utilisant de tels problèmes qu'on peut créer de la sécurité en informatique.

36	Qu'est-ce qu'un algorithme ?	Ensemble d'étapes qui permettent d'accomplir une tâche
37	Une première mesure de l'efficacité de l'algorithme est ____	son temps d'exécution
38	Comme le temps d'exécution de l'algorithme dépend de paramètres ____, on ne regarde que le ____ du temps d'exécution en fonction de ____	internes ou externes à l'algorithme taux de croissance la taille n des entrées
39	Définir le logarithme en base 2 Formule	Fonction inverse de la fonction exponentielle $n = 2^x \Leftrightarrow x = \lg n$
40	Classer par ordre croissant le comportement asymptotique des fonctions constante (1) / linéaire (n) / carrée (n^2) / exponentielle (2^n) / logarithmique ($\lg n$)	$1 \leq \lg n \leq n \leq n^2 \leq 2^n$
41	La notation $\mathcal{O}(f(n))$ est utilisée pour indiquer qu'un temps d'exécution ____ qu'une constante multipliée par la fonction $f(n)$. Elle donne le comportement ____ du taux de croissance du temps d'exécution de ____.	n'est jamais pire asymptotique l'algorithme

Chapitre 10 – Algorithmique – Partie 2

Le **tri par sélection** consiste

1. à chercher l'indice `mini` de la valeur minimale du sous-tableau `T[i..n]`
2. à permuter `T[i]` avec `T[mini]` : la valeur minimale du sous-tableau est mise en premier
3. à recommencer sur le sous-tableau `T[i+1..n]`

On commence à l'indice `i=0` et on arrête quand le tableau `T` a été parcouru en entier.

Le **tri par insertion** consiste, lui,

1. à mémoriser l'élément courant `x=T[i]`
2. à décaler vers la droite tous les éléments du sous-tableau `T[1..i-1]` qui sont plus grands que `x` en commençant par `T[i-1]`
3. à placer `x=T[i]` dans le trou laissé par le décalage

Il ne sert à rien de traiter le cas `i=0` car le sous-tableau `T[0]` est trié. On parcourt les valeurs `1<=i<n`.

42	Expliquer le tri par sélection sur le tableau `T[1..n]`	`min`=indice valeur min sous-tableau `T[i..n]` échanger `T[i]` et `T[min]` recommencer sur le sous-tableau `T[i+1..n]`
43	Expliquer le tri par insertion sur le tableau `T[1..n]`	insertion de `T[i+1]` à la bonne place dans le sous-tableau `T[1 .. i]`
44	Tri par sélection : comportement asymptotique	$\mathcal{O}(n^2)$
45	Tri par insertion : comportement asymptotique	$\mathcal{O}(n^2)$

46	La recherche dichotomique est efficace car elle travaille sur un tableau [ ] de n valeurs en [ ].	trié / $\mathcal{O}(\lg n)$
47	Un algorithme récursif comprend nécessairement 2 points. Lesquels ?	1) un **cas de base** 2) chaque appel récursif doit se faire sur une instance **plus petite du même problème**.
48	Algorithme des k plus proches voisins Connaissant une certaine [ ] des données, cet algorithme prédit la propriété d'une donnée inconnue par [ ] des propriétés des k plus proches voisins. Il est donc nécessaire de calculer une certaine [ ].	propriété / moyenne / distance

Chapitre 11 – Histoire des ordinateurs – Ruptures technologiques

49	Date du premier ordinateur (ENIAC)	1945
50	Date de découverte du transistor	1947
51	Date du premier circuit intégré	1958
52	Date Microprocesseur 4004	1971
53	Date du langage C / de python	1970 / 1991

Chapitre 12 – Portes Logiques

54 Quelle est la table de vérité de la porte `NOT` ?

A	`NOT` A
0	1
1	0

55 Quelle est la table de vérité de la porte `AND` ?

A	B	A `AND` B
0	0	0
1	0	0
0	1	0
1	1	1

56 Quelle est la table de vérité de la porte `OR` ?

A	B	A `OR` B
0	0	0
1	0	1
0	1	1
1	1	1

57 Quelle est la table de vérité de la porte `XOR` ?

A	B	A `XOR` B
0	0	0
1	0	1
0	1	1
1	1	0

58	Les [ ] permettent de faire des portes logiques qui, à leur tour, permettent de faire des [ ] logiques qui, à leur tour, permettent de faire des [ ] électroniques.	transistors / circuits / composants

Chapitre 13 – Les tableaux

Un tableau python est de type `list`. Un tableau permet de manipuler une collection de données. Une de ces données est accessible par son indice dans le tableau. Tout ce qui a été vu sur les chaînes de caractère est valable ici : `len(t)` renvoie le nombre d'éléments dans le tableau `t`. L'élément `i+1` du tableau est accessible via `t[i]` avec `0 <= i <= len(t)-1` .
L'indice d'un tableau commence à `0` et s'arrête donc à `len(t)-1`. Une erreur classique est d'essayer d'appeler `t[len(t)]`, ce qui lève l'erreur `IndexError: list index out of range`.
Un tableau est itérable : on peut le parcourir. (Vocabulaire : range → intervalle, index → indice)

L'instruction `t1=t` ne copie pas le tableau `t` : les variables `t1` et `t` pointent vers les mêmes cases mémoires. Dans ce cas, modifier `t1` revient aussi à modifier `t`. Si on veut travailler sur `t2` une copie indépendante de `t`, alors il y a deux possibilités : `t2=t[:]` ou `t2=copy.deepcopy(t)` du module `copy`.
L'instruction `t2=t[:]` sera en général suffisante pour copier un tableau.
Cette instruction permet aussi de renvoyer le tableau `t` à l'envers : `t[::-1]` !

59	`[Tableau python]`
	Que renvoie `[1,2]+[3]` ?
	Déclarer un tableau `t` vide ?
	Déclarer un tableau t de 0 de taille `N` ?
	2 manières d'ajouter le contenu de la variable `i` au tableau `t`
		`[1,2,3]`
		`t = []`
		`t = [0]*N`
		`t.append(i)` ou `t+=[i]`

60	`[Tableau python]`
	2 manières de parcourir un tableau `t` pour afficher les éléments de `t` ?
	
	Lien entre ces 2 manières ?
		`for x in t :`
		`  print(x)`
		`for i in range(len(t)) :`
		`  print(t[i])`
		`x` correspond à `t[i]`

61	`[Tableau python]` Créer par compréhension le tableau `t` des entiers pairs entre 0 et 10 inclu le tableau `t1` correspondant aux éléments de `t` élevés au cube	`t = [n for n in range(11) if n%2==0]` `t1 = [x**3 for x in t]`
62	Inverser le tableau `T` en python	`T[::-1]`

Chapitre 14 – Les dictionnaires

On accède à un élément d'un tableau par son indice (entier). Un dictionnaire (de type `dict`) est une généralisation de ce concept qui permet d'accéder à une valeur à partir, non pas d'un entier, mais d'une **clé** qui peut être un entier, une chaîne de caractère, ...
Un dictionnaire est donc un ensemble de couple **clé/valeur** et il est implémenté de manière à ce que l'accès à n'importe quelle valeur se fasse en temps constant, peu importe la taille du dictionnaire.

`d={cle1:valeur1, cle2:valeur2 }`

Les clés doivent être uniques. On ajoute un nouveau couple `cle3/valeur3` de la manière suivante : `d[cle3]=valeur3` ; on affiche la valeur associée à `cle2` avec `d[cle2]`. On efface la clé `cle` d'un dictionnaire par `del(d[cle])`. On vide un dictionnaire par l'instruction `d.clear()`.
Un dictionnaire implémente un certain nombre de méthodes :

- `d.keys()` renvoie un itérateur sur l'ensemble des clés du dictionnaire `d` (attention, l'ordre des clés n'est pas forcément préservé suivant la version de python utilisée)
- `d.values()` renvoie un itérateur sur l'ensemble des valeurs du dictionnaire `d`
- `d.items()` renvoie un itérateur sur l'ensemble des couples clé/valeur du dictionnaire `d`

63

2 manières d'afficher l'ensemble des couples clé/valeur du dictionnaire `d` ?	`for cle in d :`
	`  print(cle, d[cle])`
	
	`for cle, valeur in d.items() :`
	`  print(cle, valeur)`

Chapitre 15 – Algorithmes de recherche sur les tableaux (1)

64

Algorithme de recherche séquentielle de `valeur` dans le tableau `T`	`trouve = False`
	`for x in T :`
	`  if x == valeur :`
	`  trouve = True`

65

Algorithme de recherche du maximum d'un tableau `T` d'entier	`maximum = T[0]`
	`for x in T :`
	`  if x > maximum :`
	`  maximum = x`

66

Algorithme de recherche du second maximum d'un tableau `T` d'entier `max1 > max2`	`max1, max2 = T[0], T[1]`
	`if max1 < max2 :`
	`  max1, max2 = max2, max1`
	`for x in T[2:] :`
	`  if x > max1 :`
	`  max1, max2 = x, max1`
	`  elif x > max2 :`
	`  max2 = x`

Une erreur fréquente consiste à mal initialiser la valeur `maximum` : il faut impérativement `maximum = T[0]` car sinon, il faut avoir des informations supplémentaires sur les données contenues dans le tableau pour que l'algorithme soit exact (exemple : si toutes les valeurs sont positives ou nulles, alors l'initialisation `maximum = -1` serait correct).

Chapitre 16 – Recherche dichotomique

La recherche dichotomique opère sur un tableau **trié**. Recherchons la valeur `x=6.5` dans le tableau

```
          t = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
étape 1       g           m                 d
étape 2                      g  m           d
étape 3                      g=m=d  → est-ce que la valeur recherchée est ici ?
étape 4                      d  g   → arrêt
```

On divise le tableau en 2 en calculant `m`. Est-ce que la valeur recherchée est située à l'indice `m` ? Non : comme le tableau est trié, alors on sait que la valeur recherchée, si elle est dans le tableau, est dans la partie située à droite de `m`, soit dans `t[m+1,d]`, on fixe donc `g=m+1` et on recommence (étape 2). Dans cet exemple, la valeur recherchée n'est pas dans `t` pour montrer le cas le plus défavorable. Si la valeur recherchée était dans le tableau, elle serait trouvée à l'étape 3. Ce n'est pas le cas, on arrive à l'étape 4 : on s'arrête au moment où l'indice gauche `g` devient **strictement** supérieur à l'indice droit `d`.

67

Quelle est la condition **fondamentale** pour pouvoir faire une recherche dichotomique ?

Quel est le taux de croissance de la recherche dichotomique ?

La recherche dichotomique opère sur un tableau **trié** !

$\mathcal{O}(\lg n)$

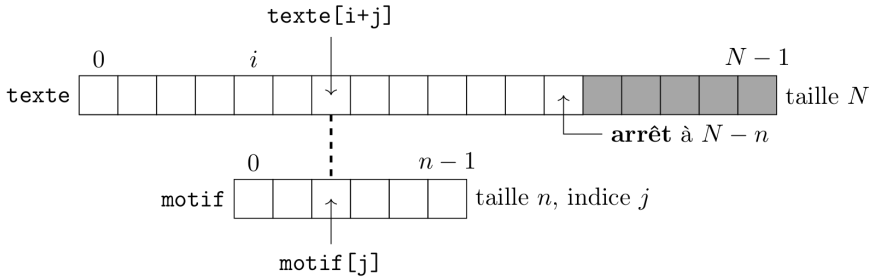
68

Quelle est la version impérative de `recherche_dichotomique(t,x)` ?
On recherche `x` dans le tableau `t`

(On note `g,d` les indices gauche/droite du sous-tableau considéré)

```python
def recherche_dichotomique(t, x) :
    g, d = 0, len(t)-1
    while g <= d : # piège
        m = (g+d)//2
        y = t[m]
        if y == x :
            return True
        if y > x : # piège
            d = m - 1
        else :
            g = m + 1
    return False
```

Chapitre 17 – Recherche textuelle naïve



On recherche si `motif` est contenu dans `texte` : pour cela, il faudra deux boucles. Une boucle va parcourir `texte`, et à partir de l'endroit où on est dans `texte`, il faut une autre boucle pour parcourir `motif`.
Il faut bien comprendre où on arrête le parcours de la boucle sur `texte` pour ne pas essayer d'accéder à des cases du tableau qui n'existent pas et lever une `Index error : out of range`.
La version proposée ici fait appel à une fonction `occurrence` qui est définie à l'intérieur d'une autre fonction (ici `recherche`), on parle dans ce cas de fonction auxiliaire pour la fonction `occurrence`.

Fonction `recherche(motif, texte)` naïve d'un `motif` dans un `texte` avec une fonction auxiliaire `occurence()`	`def recherche (motif, texte) :`
	`  def occurrence() :`
	`  for j in range(n) :`
	`  if texte[i+j] != motif[j] :`
	`  return False`
	`  return True`
	
	`  N = len(texte)`
	`  n = len(motif)`
	`  for i in range(N-n+1) :`
	`  if occurence() :`
	`  return True`
	`  return False`

69

Fin des fiches de mémorisation