Programmation : introduction au langage Python

Table des matières

I Variables et types

1 Langage de programmation

Un programme est la traduction d'un algorithme, qui est une suite d'instructions, dans un langage "compréhensible" par une machine. Il s'agit ici de se familiariser avec Python, et les principaux objets qui nous permettrons de construire des programmes.
Les mots doivent avoir un sens pour la machine, certain font partie du langage `for, def, else, return`... Nous les définirons plus tard et ils sont réservés. Tous les autres mots peuvent servir de variable, comme un x en mathématique.
Une expression est une suite de caractères définissant une valeur. Par exemple : `42, 1+4, 5.2`... Ces expressions peuvent ensuite être stockées dans des variables. Le code suivant peut être tapé dans la console pyzo, qu'on appelle aussi le shell.

```
>>> 42
42

>>>4+1
5

>>>4.2
4.2
```

Les expressions sont typées. Un type est la nature d'un objet : un entier, un flottant ($\sim$ un réel), une chaîne de caractère... Le langage de programmation a besoin de savoir quelle est la nature, le `type`, de l'objet pour savoir quelles opérations sont réalisables sur cet objet. Par exemple, on peut additionner deux entiers. On peut aussi additionner un entier et un réel ; le résultat devient alors un réel. En revanche, on ne peut pas additionner un entier et une chaîne de caractère.

2 Types

Les principaux types que nous utiliserons pour le moment en Python seront les suivants :

Type	Description	Exemples
`int`	*integer* : nombre entier	`0`, `1`, `-15`, ...
`float`	*floatting* : nombre à virgule flottante	`0.`, `1.0`, `-15.35`, ...
`bool`	*boolean* : booléen	`True` et `False`
`str`	*string* : chaîne de caractère	`"hello"`, `'hello'` ,`'a'`, `"Le langage Python"`, `""`,...

La commande `type( )` en Python permet d'obtenir le type d'une expression.
Vous pouvez essayer dans le shell de pyzo quelques expressions :

```
>>> type(42)
<class 'int'>

>>> type(42.)
<class 'float'>

>>> type('bonjour')
<class 'str'>
```

Exercice 1

Que vont retourner les commandes `type("bonjour")`, `type(3<5)`, `type(bonjour)`, `type(1+4)` et `type(1+4.)` ?

3 Opérations

Les différents types admettent des opérations prédéfinies dans Python :

3.1 type `int`

— L'addition : `+`
— La soustraction : `-`
— La multiplication : `*`
— Le quotient de la division euclidienne : `//`
— Le reste de la division euclidienne : `%`
— La puissance : `**`

3.2 type `float`

— L'addition : `+`
— La soustraction : `-`
— La multiplication : `*`
— La division : `/`
— La puissance : `**`

3.3 type `bool`

— La négation : `not`
— La disjonction : `or`
— La conjonction : `and`

3.4 type `str`

— La concaténation : `+`
— La longueur : `len( )`
— Accès à un caractère : `chaine.[i]`
— Mise en majuscule : `chaine.upper()`
— Mise en minuscule : `chaine.lower()`

3.5 Exercices

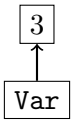
Prédire le résultat des expressions suivantes, puis les taper dans le shell de pyzo pour comparer à la solution rendue par Python :

Expression	Prédiction	Résultat affiché - commentaire
`3+4`		
`3+4.`		
`2*4`		
`2**4`		
`17//3`		
`17%3`		
`17./3`		
`17.//3`		

Expression	Prédiction	Résultat affiché - commentaire
`17//3.`		
`17/0`		
`3**1/2`		
`3**(1/2)`		
`2**100`		
`2.**100`		
`2.**1023`		
`2.**1024`		
`3>3`		
`3>=3`		
`3=3`		
`3==3`		
`3!=3`		
`(3<=5) or (3!=3)`		
`(3<=5) and (3!=3)`		
`(3<=5) and not(3!=3)`		
`pi`		
`from numpy import pi`		
`pi`		
`"Oussama"`		
`kebe.upper()`		
`'kebe'.upper()`		
`"Hamza".lower() + " " + "Dachour"`		
`len(KEBE)`		
`len("KEBE")`		
`Paul[0]`		
`"Paul"[0]`		
`"Paul"[1]`		
`"Paul"[4]`		

4 Variables

Les variables sont des cases mémoires qui doivent être nommées pour être utilisées. Une fois définies on peut y mettre de l'information, y accéder, la changer et communiquer avec l'ordinateur.
Attention, les noms de variables sont des séquences de lettres ou de chiffres, commençant toujours par une lettre, sans caractères spéciaux (hormis l'underscore _) et sensibles à la casse.
Si on appelle `var` une variable qui a la valeur 3 en mémoire, on le représentera souvent par un schéma de la forme :



Exercice 2

Lesquelles de ces variables ont un nom valide ?

1. `Positionpivot`
2. `taille_fenêtre`
3. `age_du_capitaine`
4. `date de naissance`

4.1 Affectation

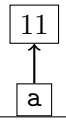
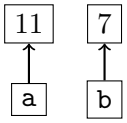
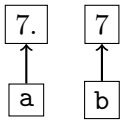
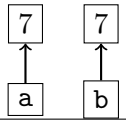
L'affectation d'une valeur dans une variable sert à stocker de l'information dans cette variable.
L'affectation s'écrit avec un signe `=`.

Exemple 3

Lorsque l'on tape

```
var = 3
```

La variable de nom `var` contient la valeur de l'entier 3.
On peut ensuite vérifier son type en tapant `type(var)`.

Instruction	Commentaire	Type de `a`	Type de `b`	Représentation
`a=2*3+5`	L'expression `2*3+5` est évaluée puis stockée dans la variable `a`	`int`		
`b = a-4`	Dans l'expression `a-4`, la variable `a` est remplacée par sa valeur, et c'est `11-4` qui est évalué puis stocké dans la variable `b`	`int`	`int`	
`a = a-4.`	Dans l'expression `a-4.`, la variable `a` est remplacée par sa valeur, et c'est `11-4.` qui est évalué et renvoie donc un flottant qu'on stocke dans `a`	`float`	`int`	
`a= b`	La valeur dans la variable `b` est affectée dans la variable `a`	`int`	`int`	

Pour connaitre la valeur dans une variable, on peut taper dans le shell le nom de la variable puis faire entrer.

Exemple 4

```
>>> a = 7*2+0.
>>> a
14.0
>>> type(a)
<class 'float'>
>>> a = int(a)+1
>>> a
15
>>> type(a)
<class 'int'>
```

Exercice 5

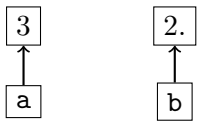
Compléter le tableau suivant :

Instruction	Type de `a`	Type de `b`	Valeur dans `a`	Valeur dans `b`
`a = 5*9-7`				
`b = a/2`				
`a = int(b)`				
`a = b-4*a`				
`a = "Python."`				
`b = len(a)//2`				
`b = len(a[1])`				

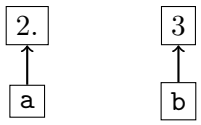
4.2 La double affectation

Exercice 6

On suppose que les variables `a` et `b` contiennent des valeurs qu'on veut échanger.
Écrire une suite d'instructions qui permet d'échanger leur valeur. Par exemple si on a



après les instructions, on aura



La double affectation en python permet d'échanger plus simplement des valeurs.
Il suffit pour cela de taper l'instruction `a,b = b,a`.

Exercice 7

Compléter le tableau suivant :

Instruction	Type de `a`	Type de `b`	Valeur dans `a`	Valeur dans `b`	Commentaire
`a = 5`					
`b = 5.`					
`a = a-1`					
`a,b = b,a`					
`a,b = a+2,a+2`					

5 Input/output

5.1 La fonction `print`

La fonction `print` permet d'écrire une chaîne de caractère dans la console.

Exemple 8

```
>>>print("bonjour !")
```

```
bonjour !

>>> "bonjour"
'bonjour'
```

Elle permet aussi d'écrire une valeur se trouvant dans une variable :

Exemple 9

```
>>> a = 3

>>> print(a+2)
5
```

On peut afficher plusieurs variables en les séparant par des `,`.

Exemple 10

```
>>> annee = 2018
>>> mois = "septembre"
>>> jour = 27
>>> print("Nous sommes le" , jour ,  mois , annee , "." )
???
```

Remarque 11

Naturellement, le `print` place un espace entre les variables. On peut contourner ce problème en utilisant la concaténation des chaînes de caractères, et la fonction `str()` qui transforme une valeur en la chaîne de caractère la représentant ; ou en ajoutant l'option `sep = ''` dans le print.

C'est une **option** de la fonction `print`. L'option `sep` permet de choisir ce que la fonction `print` met entre les variables (par defaut c'est un espace, écrire l'option `sep = ''` remplace la séparation entre les variables par la chaîne de caractère vide).

```
>>>  print("Nous sommes le" , jour ,  mois , annee , "." , sep='')
Nous sommes le27septembre2018.
>>>  print("Nous sommes le" , jour ,  mois , annee , "." , sep='*')
Nous sommes le*27*septembre*2018*.
```

Pour accéder aux options d'une fonction, on peut taper `help()`.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

La fonction `print` possède d'autres options. Par exemple l'option `end` permet de choisir ce par quoi termine un `print`. Par defaut, c'est un retour à la ligne : `end='\n'`.

Exemple 12

```
>>> print(a,end='')
3
>>> print(a)
3

>>>
```

Exercice 13

Reprendre l'exemple 10, et afficher la date de trois façons différentes, sans espace avant le point.

5.2 La fonction `input`

Pour lire une chaîne de caractère et la stocker dans une variable, on peut utiliser la fonction `input`.

Exemple 14

Taper le code suivant dans l'éditeur pyzo, puis exécuter le code.

```
print("Entrer votre nombre préféré")
a= input()
print("Votre nombre préféré est", a)
```

La variable `a` dans cet exemple contient une chaîne de caractère. Pour pouvoir faire des opérations sur ce nombre, il faut le convertir.

Exercice 15

Taper les instructions suivantes dans le shell de pyzo et noter le résultat.

```
>>> a = "4.5"
>>> float(a)
???
>>> int(a)
???
>>> int(float(a))
???
>>> a = "4"
???
>>> int(a)
???
```

Remarque 16

Les fonctions `int` et `float` sont des fonctions de conversion.

Exemple 17

On peut aussi écrire le `input` plus simplement :

```
a= input("Entrer votre nombre préféré :")
print("Vous n'aimez pas beaucoup le nombre", int(a)+4)
```

Exercice 18

Écrire un programme dans l'éditeur qui lit successivement le jour, le mois et l'année de naissance de l'utilisateur et affiche par exemple : "Vous êtes né le 19 novembre 2000".

Exercice 19

Écrire un programme qui lit deux entiers x et y et qui affiche la valeur $2x + 3y$.

Exercice 20

1. Écrire un programme qui lit successivement la valeur de trois entiers, et qui affiche leur somme.
2. Récrire ce programme en n'utilisant que deux variables.

Exercice 21

On veut calculer les solutions d'une équation de la forme (E) : $ax^2 + bx + c = 0$ où a, b et c sont trois réels tels que le polynôme $aX^2 + bX + c$ admet 2 racines réelles.
Écrire un programme qui lit successivement les valeurs des réels a, b et c, puis renvoie les solutions de l'équation (E).
Essayer le programme avec $a = 1, b = -3$ et $c = 2$ par exemple.

II Fonctions

1 Définition d'une fonction

Comme en mathématique, une fonction est un objet qui peut prendre en entrée une valeur et peut en retourner une.

Exemple 22

Taper dans la console les instructions, puis les exectuer. Attention à **l'indentation**.

```
def f(x):
    return 2*x
```

On a défini la fonction qui à x associe $2x$.

```
>>> type(f)
<class 'function'>
```

Remarque 23

Tout ce qui est écrit dans l'indentation du `def` sert à définir la fonction. Le code suivant ne définit **pas** la fonction.

```
def f(x):
return 2*x
```

De plus, contrairement aux mathématiques, on ne précise pas quels sont les ensembles d'entrée et de sortie d'une fonction. Cela permet aussi plus de flexibilité, mais attention à ne pas tout mélanger.

Exemple 24

```
>>> f(3)
6

>>> f(6.)
12.0

>>> f("bonjour")
'bonjourbonjour'
```

2 Variables d'entrée et de sortie

Il peut y avoir zéro, une ou plusieurs variables d'entrée :

Exemple 25

Tester les fonctions suivantes, et les appeler sur des exemples :

```
def etoile():
     print ('********')

def mini (a,b):
    return (1/2)*(a+b - abs(a-b))
```

De même, il peut y avoir zéro, une ou plusieurs variables de sortie.
Par exemple, la fonction `etoile` ne renvoie rien :

```
>>> y = etoile()
********

>>> type(y)
<class 'NoneType'>
```

On voit ici que `y` n'a pas de type (plus précisément cette variable est de type `NonType`).

Exemple 26

```
def minimaxi (a,b):
    minab = (1/2)*(a+b - abs(a-b))
    maxab = (1/2)*(a+b + abs(a-b))
    return minab,maxab
```

La fonction `minimaxi` renvoie la valeur contenue dans la variable minab et celle dans la variable maxab. On peut ensuite accéder à de telles valeurs en les assignant à des variables :

Exemple 27

```
>>> lemin,lemax = minimaxi(3.,-5.)

>>> lemin
-5.
```

Exercice 28

Écrire une fonction qui prend en entrée les coordonnées d'un vecteur (x, y), et qui renvoie sa norme $\sqrt{x^2 + y^2}$.

Les variables sont par défaut dites **locales**. Ce qui veut dire qu'elles n'existent pas hors de la fonction :

Exemple 29

```
def plus(x,y):
    z = x+y
    return z

>>> plus(2,3)
5
```

```
>>> z
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'z' is not defined
```

Il faut préciser dans le corps de la fonction si on veut pouvoir la rendre **globale**, c'est à dire modifiable par la fonction. Attention, si cette possibilité est pratique dans certain cas, elle est à éviter car elle entraîne souvent des erreurs.

Exemple 30

```
z=4
def f(x):
    global z
    z=z+1
    return x*z

>>> z
4

>>> f(2)
10

>>> z
5
```

Essayer de retaper les instructions précédentes sans la ligne `global z`.

3 Exercices

Exercice 31

Écrire les fonctions suivantes et les tester sur différents appels. Déterminer les types de sorties suivant les types d'entrée.

1. Une fonction qui prend en argument un nombre x et qui renvoie la valeur $\dfrac{x+3}{x-1}$. (L'appeler par exemple sur `0`, sur `0.`, sur `1`, sur `'bonjour'`)
2. Une fonction qui prend en entrée deux entiers et qui renvoie le quotient et le reste de leur division euclidienne.
3. Une fonction qui prend en entrée deux entiers et qui renvoie le plus petit des deux.
4. Une fonction qui affiche 100 fois une chaîne de caractères passée en argument (on pourra utiliser le produit d'une chaîne de caractères par un entier).

Exercice 32

Quelles sont les valeurs affichées par les programmes suivants :

1.
```
def f(a,b):
    z=2*a+b
    return z
x=f(1,2)
y=f(2,1)
print(x,y)
```

2.
```
def f(a,b):
    z=2*a+b
    return z
y=1
x=1
x=f(x,y)
y=f(x,y)
print(x,y)
```

3.
```
def f(x,y):
    x=2*x+y
    return x
x=1
y=2
z=f(x,y)
print(x,y,z)
```

4.
```
def f(x,y):
    x=2*x+y
    return x
x=1
y=2
y=f(x,y)
print(x,y)
```

Exercice 33

Quelles sont les valeurs affichées par les programmes suivants :

1.
```
z=4
def f(a,b):
    global z
    z=2*a+4*b
    return z
x=f(1,2)
y=f(2,1)
print(x,y,z)
```

2.
```
z=4
def f(a,b):
    z=2*a+4*b
    return z
x=f(1,2)
y=f(2,1)
print(x,y,z)
```

III Instructions conditionnelles

Une informaticienne travaille tranquillement quand sa femme l'interpelle depuis la cuisine : "Tu pourrais passer au magasin acheter une bouteille de lait ? Et puis s'il y a des oeufs prends-en six."

L'informaticienne revient du magasin avec 6 bouteilles de lait.

1 Syntaxe

Une instruction conditionnelle permet d'effectuer une suite d'instructions seulement si une condition est vérifiée.

Exemple 34

```
def plus_grand_que_2 (x):
    if x>2:
        print ("le nombre est plus grand que 2")
    else:
        print("le nombre est plus petit ou égal à 2")
```

La fonction `plus_grand_que_2` affiche si un nombre est plus grand que 2.

```
>>> plus_grand_que_2(5)
le nombre est plus grand que 2

>>> plus_grand_que_2(-1.4)
le nombre est plus petit ou égal à 2
```

Pour l'utilisation du `if`, on fera attention à plusieurs points :

— Après le `if`, il faut écrire un booléen, suivi d'un `:`

— Les instructions qui sont exécutées sous cette condition sont indentées.

— Si on veut aussi exécuter des instructions dans le cas contraire, on utilise un `else:` qui est au même niveau d'indentation que le `if`.

— Les instructions suivantes sont aussi indentées.

Exercice 35

Écrire une fonction qui prend en entrée un entier, et renvoie `True` s'il est plus grand que 2 **et** pair, et qui renvoie `False` sinon.

Comme pour la définition des fonctions, le niveau d'indentation représente ce qui est exécute ou non par l'instruction conditionnelle.

Exercice 36

Quelles sont les valeurs affichées par les programmes suivants :

1.

```
a = 2
b=4
if a<b:
    a = a*3
else:
    a=a+1
if a>b:
    print("premier résultat")
else:
    print("second résultat")
```

2.

```
a = 2
b=4
if a<b:
    a = a*3
else:
    a=a+1
    if a>b:
        print("premier résultat")
    else:
        print("second résultat")
```

Exercice 37

Écrire une fonction `est_pair` qui prend en entrée un entier, et renvoie `True` si le nombre est pair, et `False` sinon.

2 Instructions conditionnelles imbriquées

Pour éviter d'enchaîner des `if:`, ce qui n'est pas élégant et peut créer des erreurs si on se trompe sur l'indentation, on peut utiliser le mot `elif`, fabriqué à partir des mots `else` et `if`.

Exemple 38

```
def entre_deux_et_trois(x):
    if x <2:
     print("le nombre n'est pas entre 2 et 3")
    elif x>3:
     print("le nombre n'est pas entre 2 et 3")
    else:
        print("le nombre est entre 2 et 3")
```

Exercice 39

Écrire une fonction qui prend en entrée une chaîne de caractère de longueur 1, c'est-à-dire une lettre, et qui renvoie `True` si c'est une voyelle, et `False` si c'est une consonne.

Remarque 40

Un `return` fait sortir de la fonction.

Exercice 41

Réécrire la fonction précédente sans utiliser de `elif`.

3 Exercices

Exercice 42

1. Écrire une fonction qui prend en argument deux nombres et qui renvoie leur maximum.
2. Écrire une fonction qui prend en argument trois nombres et qui renvoie leur maximum.
3. Écrire une fonction qui prend en argument trois nombres et qui affiche ces trois nombres rangés dans l'ordre croissant.

Exercice 43

Un jeu consiste à lancer deux fois un dé équilibré. La mise de départ est de 5€.

— si le joueur a deux « six », il gagne 20€ ;

— si le joueur a un seul « six », il gagne 10€ ;

— sinon le joueur perd sa mise.

Écrire une fonction qui prend en argument deux valeurs issues des deux lancers d'un dé et qui renvoie le gain du joueur (somme perçue après les lancers moins la mise).

Exercice 44

On veut calculer les solutions d'une équation de la forme (E) : $ax^2 + bx + c = 0$ où a, b et c sont trois réels.
Écrire une fonction qui prend en entrée des réels a, b et c, puis renvoie les solutions de l'équation (E) sous le format suivant :

— si le trinôme possède deux racines réelles distinctes, la fonction renvoie ces deux racines.

— si le trinôme possède une racine double, la fonction renvoie cette racine

— si le trinôme possède deux racines complexes, la fonction renvoie ces deux racines, chacune d'elles étant représentée par une chaîne de caractères de la forme "partie réelle + i partie imaginaire".

Exercice 45

Une année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100. Toutefois, elle est considérée comme bissextile si c'est un multiple de 400. Écrire une fonction pour déterminer si une année demandée est bissextile ou non.

Exercice 46

Les programmes suivants sont-ils syntaxiquement corrects. Si oui qu'affichent-ils ?

1.
```
x = 4
y = 0
z = 0
if x==4 :
    y=1
else :
    y=2
    z=1
print(x,y,z)
```

2.
```
x = 5
y = 0
z = 0
if x == 4 :
     y=1
else :
    y=2
z=1
print(x,y,z)
```

3.
```
x = 4
y = 0
z = 0
if x == 5:
y=1
z=1
print(x,y,z)
```

4.
```
x = 4
y = 0
z = 0
if x == 5 :
    y=1
else :
y=2
z=1
print(x,y,z)
```

IV Les structures de données

1 Chaînes de caractères

1.1 Création

On peut créer une chaîne de caractère par trois méthodes équivalentes :

```
# on met la chaine entre guillemets
ma_chaine = "Bonjour tout le monde !"

# on met la chaine entre apostrophes
ma_chaine = 'Bonjour tout le monde !'

#on utilise le constructeur str (string en anglais).
#L'entier 3 est converti en '3'.
ma_chaine = str(3)
```

Remarque 47

Les deux premières méthodes existent pour qu'une chaîne de caractère puisse contenir ces caractères spéciaux : `"c'est ma chaine"` ou `'Ceci est un "string"'`.
La dernière est une méthode de conversion.

Exercice 48

Écrire une fonction qui prend en entrée une chaîne de caractère et qui renvoie une autre chaîne de caractère de la forme `"Cette chaine possède n caractères."` où n est calculé, on le rappelle, avec la fonction `len`.

Remarque 49

La chaîne de caractères vide se crée de manière équivalente en écrivant `ma_chaine_vide = ''` ou `ma_chaine_vide = str()`.

1.2 Concaténation

L'opérateur + sert à concaténer des chaînes de caractères, c'est-à-dire à les mettre bout à bout pour fabriquer une nouvelle chaîne.

Exemple 50

```
>>> nom = 'Terraha'
>>> prenom = 'Sofiane'
>>> identite = prenom + ' ' + nom
>>> print(identite)
Sofiane Terraha
```

Remarque 51

L'opérateur $*$ entre une chaîne de caractère et un entier `n` permet de concaténer n fois cette chaîne.

Exercice 52

Écrire une fonction `ecriture` qui prend en entrée un entier `n` et renvoie le nombre `m` s'écrivant de la forme $xxxxx$ où x est le nombre $n + 5$.
Par exemple :

```
>>> m =ecriture(3)

>>> m
888888

>>> m =ecriture(5)

>>> m
101010101010
```

1.3 Accéder à un caractère

On accède à un caractère d'une chaîne en écrivant sa position entre crochet après le nom de la chaîne. Le premier rang est 0.

Exemple 53

```
>>>ma_chaine = 'Python'
>>>ma_chaine[0]
'P'
>>> ma_chaine[1]
'y'
>>> ma_chaine[4]
'o'
>>> ma_chaine[6]
IndexError: string index out of range
```

Exercice 54

Écrire une fonction qui prend en entrée une chaîne de caractère et qui renvoie `True` si sa première et sa dernière lettre sont identiques, et `False` sinon.

1.4 Remplacer un caractère

Dans une chaîne, on peut remplacer un caractère par un autre en utilisant la méthode `replace` :

Exemple 55

```
>>> 'ma chaine de caractères'.replace(' ','**')
# on remplace chaque espace par deux étoiles
'ma**chaine**de**caractère'
```

Par la même méthode, on peut supprimer un caractère d'une chaîne en utilisant la chaîne vide :

Exemple 56

```
>>> 'ma chaine de caractères'.replace(' ','')
# on enlève les espaces
'machainedecaractère'
```

Exercice 57

Écrire une fonction qui prend en entrée une chaîne de caractère et qui compte le nombre d'étoiles (le symbole `*`) se trouvant dans cette chaîne.

1.5 Savoir si un caractère est dans une chaîne

L'opérateur `in` permet de savoir si un caractère ou une chaîne de caractères est dans une autre chaîne :

Exemple 58

```
>>> 'a' in 'Duchamps'
True
>>> 'b' in 'Duchamps'
False
>>> 'ch' in 'Duchamps'
True
>>> 'phs' in 'Duchamps'
False
```

Exercice 59

Écrire une fonction qui prend en entrée deux chaînes de caractères et qui renvoie `True` si l'une de ces chaînes est dans l'autre, et `False` sinon.

1.6 Comptage

La méthode `count` permet de compter le nombre d'occurrences d'une chaîne dans une autre :

Exemple 60

```
>>> 'Informatique'.count('i')
1
>>> 'Informatique'.count('b')
0
>>>'Bonjour tout le monde'.count('on')
2
```

Exercice 61

Écrire une fonction qui prend en entrée une chaîne de caractère et qui compte le nombre de voyelles qu'elle contient.

1.7 Comparaison

Les chaînes de caractères sont ordonnées par ordre alphabétique. Pour comparer deux chaînes, on utilise les symboles `<` ou `>` ; comme pour les nombres, l'égalité se teste avec les symboles `==` ou `!=`.

Exemple 62

```
>>> 'Amine' < 'Yassir'
True
>>> 'Amine' > 'Yassir'
False
>>> 'Sofyane' == 'Sofiane'
False
>>> 'Sofyane' != 'Sofiane'
True
```

Exercice 63

Écrire une fonction qui prend en entrée deux lettres et renvoie la première dans l'ordre alphabétique.

2 Listes

Les listes sont très utilisées en programmation. Elles permettent de structurer les données de manière à pouvoir les stocker, les parcourir ou les consulter facilement.

2.1 Création

On crée une liste en mettant ses éléments entre crochets, et en les séparant par des virgules.

Exemple 64

```
liste_notes = [11.0,18,9.5,17.5,14,15]
liste_mois = ['janvier','février', 'mars', 'avril', 'mai', 'juin', 'juillet', 'août', 'septemb
liste_numeros = ['0680541226', '0258654280', '0602035647', '0698754128',
'0145865987', '0487956324', '0387426412', '0582478268']

>>>type(liste_notes)
<class 'list'>
>>>type(liste_mois)
<class 'list'>
>>>type(liste_numero)
<class 'list'>
```

Remarque 65

On peut aussi créer une liste "vide" avec l'une des deux instructions suivantes : `liste_vide = []` ou `liste_vide =list()`.

Exercice 66

Créer la liste des élèves de la classe.

2.2 Fonctions de base

Comme pour les chaînes de caractère, on a facilement accès à la longueur, à la valeur d'un élément, et on peut concaténer des listes.

Exemple 67

```
>>>len(liste_notes)
6
>>>len(liste_mois)
12

>>>liste_notes[0]
11.0
>>> liste_notes[1]
18
>>> liste_notes[5]
15

>>> L1 = [1,2,3]
>>> L2 = [4,5,6]
>>>L1+L2
[1,2,3,4,5,6]

>>> L = [1,2,3]
>>> L*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

#créer une liste contenant 10 zéros
>>> L = [0]*10
>>> L
[0,0,0,0,0,0,0,0,0,0]
```

Exercice 68

Écrire une fonction qui dit si une liste est vide ou non.

Exercice 69

Écrire une fonction qui renvoie `True` si le premier et le dernier élément d'une liste sont identiques, et `False` sinon.

Exercice 70

Écrire une fonction qui prend en entrée une chaîne de caractère étant une date au format `jj/mm/aaaa`, et qui l'affiche en écrivant le mois en toute lettre. Par exemple, la date 05/06/1994 sera écrite au format « 5 juin 1994 ». On utilisera la liste `liste_mois`.

Remarque 71

Contrairement aux chaînes de caractères, on peut modifier un élément d'une liste :

```
>>> L =[1,2,3]
>>> L[1] = 4
>>> L
[1,4,3]
```

Exercice 72

Écrire une fonction qui change le premier et le dernier caractère d'une liste par les chaînes de caractères `'premier'` et `'dernier'`.

Exercice 73

1. Tester successivement les instructions suivantes :

```
>>> L1 = [1,2,3]
>>> L2 = L1
>>> L2[1] = 4
>>> L2
???
>>> L1
???
```

   Que remarque-t-on ?

2. Répéter la séquence d'instructions précédente en remplaçant la deuxième instruction par `L2=[L1[0],L1[1],L1[2]]`. Que peut-on conclure ?

Si un élément appartient à une liste, on peut obtenir son premier indice par la méthode `index`.

Exemple 74

```
>>> L = [1,2,3,1]
>>> L.index(1)
0
>>> L.index(4)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ValueError: 4 is not in list
```

Exercice 75

Écrire une fonction qui prend en entrée une chaîne de caractère étant un mois et qui renvoie son numéro. Par exemple, si elle prend `"mai"` en entrée, elle renvoie `5`.

Exercice 76

Écrire une fonction qui prend en entrée une liste `L` et un élément `e`, et qui renvoie `True` si l'élément `e` appartient à la liste, et `False` sinon.

2.3 Extraction

— On peut extraire la partie d'une liste `L` comprise entre deux indices `n` et `m` en écrivant `L[n : m+1]`.

— On peut extraire le début d'une liste `L` jusqu'au rang `m-1` en écrivant `L[ : m]`.

— On peut extraire la fin à partir d'un indice `m` d'une liste `L` en écrivant `L[m :]`.

— On peut extraire toute la liste `L` en écrivant `L[:]`.

Exemple 77

```
>>> L=[3,4,5,8,2,1,9]
>>> L[:2]
[3, 4]
>>> L[2:]
[5, 8, 2, 1, 9]
>>> L[2:7]
[5, 8, 2, 1, 9]
>>> L[2:3]
[5]
```

Exercice 78

Reprendre l'exercice 73, en changeant la deuxième instruction par `L2 = L1[:]`. Que peut-on conclure ?

Exercice 79

Écrire une fonction `extraire(L,n,m)`, où `n` et `m` sont des entiers plus petits que la longueur de la liste avec $n < m$. Elle renvoie la liste `L` dont on a extrait les valeurs entre les indices `n` et `m-1`.
Par exemple `extraire([1,2,3,4,5,6,7,8,9],2,6)` renvoie la liste `[1,2,7,8,9]`.

Exercice 80

Écrire une fonction qui prend en entrée une liste `L` et un élément `e`, et qui renvoie `True` si l'élément `e` appartient au moins deux fois à la liste, et `False` sinon.

La méthode `append` permet d'insérer un élément en fin de liste.

Exemple 81

```
>>> L=[3,4,5,8,2,1,9]
>>> L.append(3)
>>>L
[3,4,5,8,2,1,9,3]
>>> L.append('bonjour')
>>>L
[3,4,5,8,2,1,9,3,'bonjour']
```

Exercice 82

Écrire une fonction qui prend en entrée une liste et qui lui ajoute un `0` au début et à la fin.

2.4 Conversion en chaîne de caractères

La méthode `join` renvoie la concaténation des chaînes de caractères d'une liste.

Exemple 83

```
>>> L = ['g','a', 'd', 'x', 'y', 'z', 'b', 'e', 'c', 'f']
>>>''.join(L) # on ne sépare pas les éléments de L
'gadyzbecf'
>>>' '.join(L) # on sépare les éléments de L par des espaces
'g a d y z b e c f'
>>>'*'.join(L) # on sépare les éléments de L par des '*'
'g*a*d*y*z*b*e*c*f'
```

La méthode `split` renvoie une liste à partir d'une chaîne de caractère comme dans l'exemple suivant :

Exemple 84

```
>>>chaine = "Bonjour à tout le monde"
>>>liste = chaine.split()
>>>liste
['Bonjour', 'à', 'tout', 'le', 'monde']
>>>chaine2 = "\n".join(liste)
>>> chaine2
'Bonjour\nà\ntout\nle\nmonde'
>>> print(chaine2)
Bonjour
à
tout
le
monde
```

Exercice 85

Écrire une fonction qui fait l'inverse de celle de l'exercice 70. C'est-à-dire qu'elle prend en entrée une date écrite au format `''5 juin 1994''`, et elle renvoie une chaîne de caractère au format `''05/06/1994''`.

3 Tableaux à 2 dimensions

Certaines données peuvent se représenter sous forme de tableaux. Une image, par exemple, est représentée par un tableau de pixels, chaque pixel étant représenté par un triplet de couleur. En mathématiques, une matrice se représente sous la forme d'un tableau de nombres. En Python, on peut représenter les tableaux comme une liste de listes chacune représentant une ligne du tableau.

Exemple 86

```
>>> T = T=[[0]*5]*6
>>> T
[[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0]]
```

Comme il s'agit d'une liste de listes, les méthodes et fonctions précédentes s'utilisent de la même façon.

Exemple 87

```
>>> T[1]
[0, 0, 0, 0, 0]
>>> T[1] = [1,2,3,7]
>>> T
[[0, 0, 0, 0, 0],
[1, 2, 3, 7],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0]]
>>> T[1][2]
3
>>> T[1].append(5)
>>>T
[[0, 0, 0, 0, 0],
[1, 2, 3, 7,5],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0],
[0, 0, 0, 0, 0]]
>>> T[1][0]=8
>>> T
[[0, 0, 0, 0, 0], [8, 2, 3, 7,5], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0,
```

Exercice 88

1. Créer le tableau `Tab` composé des éléments suivants :

$$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 4 & -2 \\ 0 & 1 & 2 \end{bmatrix}.$$

2. Écrire une séquence d'instruction qui modifie le tableau `Tab` en ajoutant 1 aux éléments des diagonales.

4 uplets

Comme en mathématiques, les *n-uplets* généralisent la notion de couple ou de triplet. Les couples ou les triplets permettent par exemple de représenter les coordonnées de points dans le plan ou dans l'espace.
On crée un n-uplet en mettant ses éléments entre parenthèses, et en les séparant par les virgules.

Exemple 89

```
point = (2.3,-3.15)
date = (5,'janvier',2013)

>>>type(point)
<class 'tuple'>
```

Remarque 90

Comme pour les listes et les chaînes de caractères, il y a deux méthodes équivalentes pour créer un 0-uplet (i.e. un uplet vide) `zero_uplet = ()` ou `zero_uplet =tuple()`.
Pour la création d'un 1-uplet, la virgule est indispensable pour le distinguer de la valeur :

```
>>> vrai_un_uplet = (2,)

>>> type(vrai_un_uplet)
<class 'tuple'>

>>> faux_un_uplet = (2)

>>> type(faux_un_uplet)
<class 'int'>
```

Les fonctions de base sont les mêmes de que précédemment, mais attention, comme pour les chaînes de caractères, on ne peut pas affecter une nouvelle valeur à une coordonnée d'un n-uplet.

Exercice 91

Taper les commandes et compléter :

```
>>> mon_uple = (3,4.,1.2)
>>> mon_uple[1]
???
>>> mon_uple[1]=2
???
```

```
>>> coord1,coord2,coord3 = mon_uple
>>> coord1
???
>>> abscisse = mon_uple[0]
>>> abscisse
???
>>> nouveau_uple = mon_uple + (5,'bonjour')
>>> nouveau_uple
???
>>> nouveau_uple[2:]
>>> 3 in nouveau_uple
???
>>> 'bonjour' in nouveau_uple
???
>>> 'Bonjour' in nouveau_uple
???
>>> 'jour' in nouveau_uple
???
>>> 'jour' in nouveau_uple[4]
???
>>> len(nouveau_uple)
???
```

Comme l'expression d'un uplet est évaluée avant l'affectation, on utilise cette structure de donnée pour faire de l'affectation multiple, comme vu à l'exercice 6.

Exemple 92

```
>>> (x,y) = (1,2)
>>> x
1
>>>y
2
>>> (x,y) = (y,x)
>>> x
2
>>>y
1
>>> x,y = y,x
>>> x
1
>>>y
2
```

Exercice 93

Écrire une fonction qui prend en entrée deux vecteurs représentés par deux couples et qui renvoie leur produit scalaire.

V Instructions itératives

1 Boucle for

1.1 Notion de boucle

On veut afficher une pyramide d'étoiles de la forme

```
*
**
***
****
```

Exercice 94

1. Écrire une fonction qui affiche une telle pyramide.
2. Écrire maintenant une fonction qui affiche une telle pyramide mais avec 5 lignes plutôt que 4.
3. Écrire une fonction avec maintenant 10 lignes.

Pour effectuer une tâche répétitive, l'ordinateur est très adapté. C'est la notion de boucle. Lorsqu'on veut executer n fois une instruction, avec n connu, on utilise la boucle `for` de Python.
Sa structure est :

```
# Boucle for, pour un nombre d'itérations connu d'avance
for i in objet_iterable:
    instructions      # qui peuvent dépendre de i
```

Remarque 95

Comme pour les fonctions et les `if`, on fera attention à l'indentation et aux `:`.

Qu'est-ce qu'un **objet itérable** ? C'est un objet dont on peut parcourir les valeurs. Par exemple une liste, une chaîne de caractère, un uplet...

Exemple 96

```
>>> L = [1,2,3,4,5]

for i in L:
    print(i)

???
```

```
>>> machaine = 'bonjour'

for i in machaine:
    print(i)

???
```

Remarque 97

Le i est une variable muette, on peut écrire k, ou j ou n'importe quel autre nom de variable à la place.

La fonction `range` est très utile en Python pour créer des indices à parcourir sans avoir à créer la liste comme dans l'exemple 96.

Exercice 98

Exéctuer les codes suivants depuis l'éditeur, et regarder ce que renvoie la console.

1.
```
for i in range(10):
    print(i)

???
```
2.
```
for i in range(10):
print(i)

???
```
3.
```
for i in range(10)
    print(i)

???
```
4.
```
for i in range(0,10):
    print(i)

???
```
5.
```
for i in range(2,10):
    print(i)

???
```
6.
```
for i in range(10,20):
    print(i)

???
```
7.
```
for i in range(0,20,2):
    print(i)

???
```
8.
```
for i in range(0,20,3):
    print(i)

???
```
9.
```
for i in range(0,20,-1):
    print(i)

???
```
10.
```
for i in range(20,0):
    print(i)

???
```
11.
```
for i in range(20,0,-1):
    print(i)

???
```
12.
```
for i in range(20,0,-2):
    print(i)

???
```

Exercice 99

Écrire une fonction `etoile` qui prend en entrée un entier `n` et qui affiche la pyramide d'étoiles à n lignes.

Exercice 100

1. Écrire une fonction qui prend en entrée un entier `n` et qui affiche les `n` premiers nombres impairs.
2. Écrire une fonction qui prend en entrée un entier `n` et qui affiche les nombres impairs strictement plus petits que `n`.

Exercice 101

1. Que fait la fonction suivante ?

```
def S(n):
    s=0
    for i in range(n+1):
        s = s+i
    return s
```

2. Écrire une fonction qui prend en entrée deux entiers `p` et `q` et qui renvoie la valeur de la somme $\sum_{i=p}^{q} \frac{1}{i^2}$.

Exercice 102

Soit $(u_n)_n$ la suite définie pour tout entier naturel n par $u_n = \dfrac{1}{n^2+5}$.
Écrire un programme qui affiche les 100 premiers termes de la suite (u_n) arrondis à 10^{-6} près. (On utilisera la fonction `round`).

Exercice 103

1. Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la somme de ses valeurs.
2. Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la moyenne de ses valeurs.

Exercice 104

1. Écrire une fonction qui prend en entrée une liste `L` et un élément `e` et qui renvoie le premier indice où se trouve `e` s'il appartient à `L`, et `False` si `e` n'appartient pas à `L`.
2. Écrire une fonction `indices(L,e)` qui prend en entrée une liste `L` et un élément `e` et qui renvoie la liste de tous les indices où se trouve `e`.
   Par exemple, `indices([1,2,5,1,7,1,2],1)` renvoie la liste `[0,3,5]`.

Exercice 105

1. Écrire une fonction `miniliste` qui prend en entrée une liste d'entiers, et renvoie la valeur de son plus petit élément.
2. Écrire une fonction similaire, mais qui renvoie le couple de son plus petit élément, et le premier indice où se trouve cet élément.

Remarque 106

Dans une fonction, un `return` sort de la fonction, et arrête la boucle `for`.

Exercice 107

Que fait le programme suivant ?

```
def prog(ch,lettre):
    ind = 0
    for elem in ch:
       if elem == lettre:
           return ind
       ind = ind +1
    return False
```

Exercice 108 (*Un jeu très amusant*)

Pour générer un entier de manière aléatoire on utilise les commandes suivantes :

```
from random import randint  #à ne taper qu'une seule fois, pour ajouter la fonction randint
>>> k = randint(0,99)       # k est un entier aléatoire entre 0 et 99
```

Créer une fonction qui prend en argument un entier `n`, et qui exécute le jeu suivant. Elle choisit en secret un nombre entre 0 et 99, puis demande à l'utilisateur de le deviner en moins de `n` coups. À chaque proposition, le programme indique à l'utilisateur si son nombre est plus grand ou plus petit que le nombre à deviner ; ou bien s'il a gagné.
Si l'utilisateur trouve le nombre mystère, il gagne. S'il ne trouve pas au bout de `n` coups, il perd.

1.2 D'autres parcours

Une boucle `for` permet de créer simplement une liste de longueur donnée :

Exemple 109

```
>>> L = [i for i in range(2,11)]
>>> L
???
```

Exercice 110

Créer la liste `L = [2,4,6,8, ..., 100]` à l'aide d'une boucle `for`.

La fonction `enumerate` permet un parcours de `list` ou de `str` donnant à la fois la valeur et l'indice :

Exemple 111

```
for ind, elem in enumerate([7,5,3,'bonjour',5.4]):
    print (ind, elem)

0 7
1 5
2 3
3 bonjour
4 5.4
```

Exercice 112

1. Écrire une fonction qui prend en argument une chaîne de caractère et qui renvoie les indices où se trouvent les `a`.
2. Écrire une fonction qui prend en argument une chaîne de caractère et qui renvoie les indices où se trouvent les voyelles.
3. Écrire une fonction qui prend en argument une chaîne de caractère et qui renvoie les indices où se trouvent les consonnes.

Pour travailler sur les tableaux, par exemple, on peut imbriquer les boucles `for` :

Exemple 113

Taper le code suivant dans l'éditeur et regarder la valeur de `T` après l'avoir exécuté.

```
T = [[0]*5 for i in range(10) ]
for i in range(10):
    for j in range(5):
        T[i][j] = i+j
```

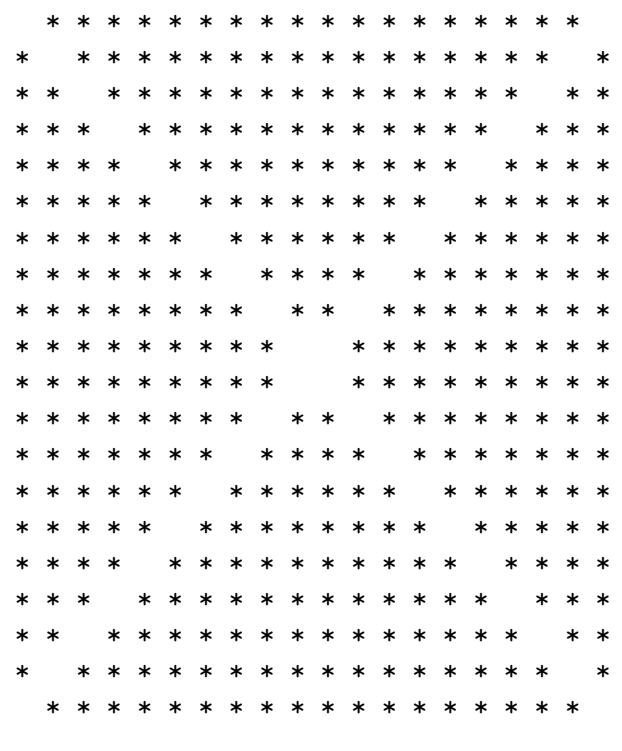
Exercice 114

1. Écrire une fonction qui prend en entrée un entier `n` et qui renvoie un tableau à deux dimensions de taille $n \times n$ dont le coefficient en ligne i et colonne j vaut $i + j$.
2. Écrire une fonction `tableauentier(n)` qui prend en entrée un entier `n` et qui renvoie un tableau à deux dimensions de taille $n \times n$ dont les coefficients sont les nombres de 1 à n^2 :

   ```
   >>> tab = tableauentier(3)
   >>> tab
   [[1,2,3],[4,5,6],[7,8,9]]
   ```

Exercice 115

1. À l'aide de deux boucles imbriquées, écrire un programme qui affiche 20 lignes de 20 étoiles chacune.
2. Modifier le programme pour qu'il affiche ceci :



Exercice 116

Écrire une fonction qui affiche tous les jours de l'année écrit en toutes lettres :

```
1 janvier 2019
2 janvier 2019
...
31 décembre 2019
```

(On pourra s'aider d'une liste `liste_mois`.)

Exercice 117

1. Écrire une fonction `i_min(L,p)` qui renvoie un indice supérieur ou égal `p` où le minimum la liste `L[p:]` est atteint.
2. Écrire une fonction `place_min_en_tete(L,p)` qui échange les valeurs de `L[p]` et `L[i_min(L,p)]`.
3. En utilisant les fonctions précédentes, écrire une fonction `tri(L)` qui trie une liste `L` dans l'ordre croissant.

2 Boucle while

> *L'informaticienne se fait enguirlander par sa femme qui lui dit : "Retourne au magasin, et tant que tu y es, prends de l'eau".*
> *L'informaticienne n'est jamais revenue.*

Comme les boucles `for`, les boucles `while` de Python permettent de répéter une action, mais on les utilise lorsqu'on ne sait pas au bout de combien d'itération la boucle doit s'arrêter.

La boucle `for` se fait sur un itérable, la boucle `while` demande une condition d'arrêt :

```
# Boucle while, pour l'arrêt des itérations par un test
while condition:
    instructions      # qui modifient généralement la condition
```

Remarque 118

Comme pour les fonctions, les `if` et les `for`, on fera attention à l'indentation et aux `:`.
Dans cette structure, `condition` est un booléen (qui est souvent modifié dans le corps de la boucle). Tant que le booléen `condition` est `True`, les `instructions` sont executées. Dès qu'il devient faux, on sort du corps de la boucle.

Exemple 119

```
def compte(n):
    i=0
    while i<n:
        print(i)
        i = i+1
```

Exercice 120

1. Rééecrire le programme de l'exemple 119 avec une boucle `for`.
2. Réécrire le programme `etoile` de l'exercice 99 avec une boucle `while`.

Remarque 121

Une boucle `for` est un cas particulier de la boucle `while`. On l'utilise lorsqu'on connait à l'avance le nombre d'itérations car elle est plus simple à écrire.

Exercice 122

Réécrire les programmes de l'exercice 101 à l'aide d'une boucle `while`.

Exercice 123

On reprend la suite de l'exercice 102 définie par $u_n = \dfrac{1}{n^2+5}$.

1. Écrire un programme qui affiche les termes de la suite (u_n) qui sont strictement supérieurs à 10^{-4}.
2. Écrire un programme qui affiche le plus petit entier naturel n tel que u_n est strictement inférieur à 10^{-8}.

Le risque principal avec les boucles `while`, c'est que la condition d'arrêt ne soit jamais obtenue, et que le programme tourne à l'infini :

Exemple 124

Essayer de lancer le programme suivant :

```
i = 0
while i >= 0:
    print(i)
    i=i+1
```

Lorsque cela est possible, on utilisera donc une boucle `for`. Cependant, elles ne sont pas toujours adaptées pour résoudre un problème de programmation. Essayer par exemple de reprendre les questions de l'exercice 123 avec des boucles `for`.

Remarque 125

Un `return` dans une fonction fera sortir d'une boucle `while` et de la fonction, même si la condition n'a pas été réalisée.

Exercice 126

Réécrire le jeu très amusant de l'exercice 108, mais on veut maintenant forcer l'utilisateur à trouver la solution, quel que soit le temps que cela lui prendra. Modifier le jeu pour qu'il ne s'arrête que lorsque le joueur a gagné.