1 Variables et affectation

le langage naturel

On peut écrire des algorithmes sans passer par un langage informatique, c'est ce qu'on appelle le langage naturel. Il sert à exprimer les programmes en français, avec un peu de syntaxe spécifique, afin de pouvoir les écrire en langage informatique plus efficacement.

Voici un algorithme simple qui demande à l'utilisateur de donner un nombre entier et ensuite affichera l'entier suivant, en langage naturel et en **Python** :

Dire bonjour
Demander de saisir un nombre entier x
$x \leftarrow$ nombre saisi
$x \leftarrow x+1$
Afficher "L'entier suivant est" , Afficher x

```
print('Bonjour !')
x = int(input('Donner un nombre entier :'))
x=x+1
print('Le nombre entier suivant est :')
print(x)
```

A retenir :

La **console** est la zone à partir de laquelle on lance un commande (simple ou sous la forme d'un script), et dans laquelle les résultats s'affichent. Plus tard on verra que les sorties peuvent se faire aussi dans une fenêtre graphique, un fichier...
La **zone de saisie** est en fait un éditeur de texte, qui la plupart du temps possède une **coloration syntaxique** : les mots ont une couleur en fonction de leur rôle. Les mots en vert sont des mots spécifiques du langage Python.

La notion de variable

Dans le script, on utilise une variable : c'est x. En informatique, définir une **variable** consiste à réserver de la place pour **stocker de l'information** dans la mémoire vive d'une machine. Pour pouvoir la retrouver, on lui donne un nom, ce qui correspond à une **adresse**.
Les **noms de variables** en Python doivent obéir aux **critères** suivants :

- débuter par une lettre (minuscule ou majuscule) ou par le caractère de soulignement "_" ;
- ne comporter que des lettres (minuscules ou majuscules) et/ou des chiffres et/ou le caractère de soulignement "_".

L'idéal est de choisir des noms de variables qui ont du sens. Les mots **réservés** de Python, comme « print » ne sont pas utilisables comme noms de variable. Voici les principaux mots réservés de Python : and, as, assert, break, class, continue, def, del, elif, else, exec, except, finally, for, from, global, if, import, is, lambda, not, or, pass, print, raise, return, try, while, with, yield . Les variables sont principalement de trois types : **entier**, **flottant** (assimilable à un décimal), **chaîne de caractère** (lettres et/ou chiffres).

Création et affectation d'un variable

En langage naturel, pour créer et affecter une variable, on écrit : $x \leftarrow 2$
qui se traduit par : « on définit la variable x à laquelle on affecte la valeur 2 ». **Affecter** signifie donc stocker une valeur dans une case mémoire repérée par un nom. En Python, c'est le signe « = » qui permet de traduire l'affectation. **Il ne faut pas confondre l'affectation en Python et la notion d'équation en maths!**.
Voici le code d'une affectation en Python :

```
x = 2
```

Exercice 1 - Voici un algorithme en langage naturel .

Dire bonjour
$x \leftarrow$ nombre saisi au clavier
$x \leftarrow x * 2$
afficher x

Expliquer ce que fait cet algorithme. Traduire cet algorithme en Python.
Programmer et tester cet algorithme plusieurs fois.

Exercice 2 - Voici un algorithme en langage naturel .

Dire bonjour
$x \leftarrow$ nombre saisi au clavier
$x \leftarrow x * x$
afficher x

Expliquer cet algorithme. Traduire cet algorithme en Python.
Programmer et tester cet algorithme plusieurs fois.

Exercice 3 - Voici un algorithme en langage naturel .

Dire bonjour
$x \leftarrow$ nombre saisi au clavier
$x \leftarrow x$ augmenté de 30 %
afficher x

Expliquer cet algorithme. Traduire cet algorithme en Python. Programmer et tester cet algorithme plusieurs fois.

Exercice 4 - Voici un algorithme en langage naturel .

Dire bonjour
$x \leftarrow$ nombre saisi au clavier
$y \leftarrow$ nombre saisi au clavier
$z \leftarrow (x + y)/2$
afficher z

Expliquer ce que fait cet algorithme. Traduire cet algorithme en Python.
Programmer et tester cet algorithme plusieurs fois.

Exercice 5 - Voici un algorithme en langage naturel .

Voici un algorithme en langage Python.

```
print('Bonjour !')
x = float(input('Donner un nombre :'))
x = x * 0.88
print('Le résultat est :')
print(x)
```

Traduire cet algorithme en langage naturel, puis expliquer concrètement ce qu'il fait. Utiliser dans votre réponse le mot « pourcentage ».

2 Les structures de contrôles

2.1 Tests, conditions et instructions conditionnelles

Qu'est-ce qu'une instruction conditionnelle ?

Une instruction conditionnelle est une instruction qui n'est exécutée que si une condition est vérifiée. Ce principe est de la forme **SI ... ALORS ... SINON ...**, ou encore en anglais **IF... THEN ... ELSE ...**

Voici un algorithme qui demande à un utilisateur de saisir un nombre entier, puis ensuite affiche si ce nombre est pair ou impair, d'abord en langage naturel puis en **Python**.

Dire bonjour
Demander de saisir un nombre entier x
Si (x est pair) **alors :**
    afficher "Le nombre est pair"
Sinon :
    afficher "Le nombre est impair"
Fin Si

```
print('Bonjour !')
x = int(input('Donner un nombre entier :'))
if (x % 2 == 0) :
   print('Le nombre est pair')
else :
   print('Le nombre est impair')
```

Quelques explications : en langage naturel, la condition est « x **est pair** ». En Scratch, on voit que cette condition s'écrit « x **modulo 2 = 0** ». La commande « **modulo** » signifie « reste dans la division euclidienne ». En langage naturel, le test se traduit donc par « **le reste dans la division de** x **par 2 est égal à 0** ».
En Python, la commande « reste de la division euclidienne » est la symbole « % ». La traduction de « alors se fait juste par « : ». Les instructions à exécuter si la condition est vérifiée sont alors **indentées** pour former un bloc décalé vers la droite. Les instructions après le « else » ne sont exécutées que si la condition n'est pas vérifiée.
On remarquera que la condition à la ligne 3 s'écrit avec « == », pour ne pas le confondre avec le "égal" simple de l'affectation, comme dans « x = 2 ».

Les **logigrammes** permettent de visualiser cette structure.

Programmer cet algorithme, puis le tester plusieurs fois.

La syntaxe d'une instruction conditionnelle en Python

```
1  if (condition) :
2          instruction(s)
3
4  if (condition) :
5          instruction(s)1
6  else :
7          instruction(s)2
8
9  if (condition1) :
10         instruction(s)1
11 elif (condition2) :
12         instruction(s)2
13 else :
14         instruction(s)3
```

Ci-contre la syntaxe de trois instructions conditionnelles en Python.
Les instructions de la ligne 2 ne sont effectuées que si la condition écrite à la ligne 1 est vérifiée.Il faut noter que le « alors » n'est pas écrit en Python. Seul « : » est noté, et les instructions conditionnées sont toutes indentées.
A la ligne 6, « else » signifie « sinon ». Les instruction(s) de la ligne 7 sont exécutées si la condition de la ligne 4 n'est pas vérifiée. « else » signifie « sinon ».
Ligne 11, « elif » est la contraction de « ou sinon », donc les « instruction(s)2 » sont exécutées si la « condition2 » est vérifiée, sinon c'est finalement les « instructions(3) » qui est exécuté.

Syntaxe d'une condition en Python

```
1 a == b
2 a != b
3 a < b
4 a <= b
5 a > b
6 a >= b
7 condition1 and condition2
8 condition1 or condition2
```

Ci-contre différents tests en Python. La ligne 1 est une condition qui teste l'égalité entre a et b ; la ligne 2 si les valeurs sont différentes.
Les lignes 3 à 6 testent l'infériorité ou la supériorité.
La ligne 7 teste si les deux conditions sont vérifiées en même temps, et la ligne 8 teste si l'une ou l'autre est vérifiée.

Exercice 6 - Divisibilité par 13.

Voici un algorithme en langage naturel.

Dire bonjour
Demander de saisir un nombre entier x
Si (x est multiple de 13) **alors :**
    afficher "Le nombre divisible par 13"
Sinon :
    afficher "Le nombre n'est pas divisible par 13"
Fin Si

1. Expliquer ce que fait cet algorithme.
2. Traduire cet algorithme en Python, le programmer et le tester plusieurs fois.

Exercice 7 - Des réductions conditionnelles.

Dans un magasin, des promotions sont réalisées. Le principe est le suivant : si le prix de l'article est inférieur ou égal à 200 euros, alors la réduction est de 20 %, sinon elle est de 30 %.

1. proposer un algorithme en langage naturel qui demande le prix de départ puis finalement donne le prix après réduction.
2. Exécuter à la main l'algorithme avec comme valeur saisie pour P égale à 150.
3. Exécuter à la main l'algorithme avec comme valeur saisie pour P égale à 250.
4. Traduire cet algorithme en Python, puis le tester avec plusieurs valeur pour vérifier le fonctionnement.

Exercice 8 - Dans une école de rugby, il y a quatre groupes.

- Le groupe U8 pour les joueurs entre 8 ans inclus et 10 ans exclu ;
- le groupe U10 pour les joueurs entre 10 ans inclus et 12 ans exclu ;
- le groupe U12 pour les joueurs entre 12 ans inclus et 14 ans exclu ;
- le groupe U14 pour les joueurs entre 14 ans inclus et 16 ans exclu.

Compléter le script pour qu'il affiche le groupe lorsque l'utilisateur entre l'âge du joueur. Si le joueur est trop jeune ou bien trop âgé, un message d'information doit être affiché. Le début du code est donné ci-dessous.

```
age = int(input('Donner age du joueur :'))
if age < 8 :
        print('Trop jeune')
elif age < 10 :
        print('U8')
```

2.2 La boucle bornée POUR

Qu'est-ce qu'une boucle bornée POUR ?

Il est parfois utile dans un programme de **répéter** une ou plusieurs instructions un **nombre défini de fois**. Lorsque le nombre de répétitions est connu à l'avance, on utilise la boucle bornée POUR.

Voici un algorithme qui permet d'afficher les n premiers nombres entiers,c'est à dire les entiers de 0 à $n-1$:

Dire bonjour
Demander de saisir un nombre entier n
POUR i allant de 0 à $n-1$ **FAIRE :**
    afficher i
    dire "boucle"
FIN POUR

```
print('Bonjour !')
n = int(input('Donner un nombre entier :'))

for i in range(n) :
        print(i)
        print('boucle')
```

A savoir : la variable i est un peu comme un compteur, qui commence à 0 et qui est incrémenté d'une unité à chacun des n passages dans la boucle. Programmer et tester plusieurs fois cet algorithme.

Les différentes syntaxes de la boucle bornée « FOR » en Python :

```
1  for variable in range(n) :
2          instruction(s)
3
4
5  for variable in range (n,m) :
6          instruction(s)
7
8
9  for variable in range(n,m,k) :
10         instruction(s)
11
12
13 for lettre in 'mot' :
14         instruction(s)
```

Le mot **range** en Python désigne l'intervalle entre la valeur minimale et la valeur maximale. En français, « range » peut être traduit comme « éventail ».
Les instructions indentées à partir de la ligne 2 sont effectuées « n » fois, en faisant prendre successivement à la variable les valeurs de 0 à $n-1$, un peu comme on monte une échelle.
Les instructions indentées à partir de la ligne 6 sont effectuées pour la « variable » allant de la valeur n jusqu'à la valeur $m-1$.
Ligne 9, on peut préciser un « pas », qui par défaut est 1, mais on peut le changer en précisant k.
ligne 13 : l'échelle parcourue avec la boucle for peut être numérique, mais on peut aussi parcourir toutes les lettres d'une chaine de caractères.

Exercice 9 - Quelques exemples simples.

1. Écrire et programmer un algorithme en Python qui écrit les nombres entiers de 5 à 30.
2. Écrire et programmer un algorithme en Python qui écrit les nombres entiers pairs de 8 à 24.
3. Écrire et programmer un algorithme en Python qui écrit les nombres entiers multiples de 5 de 10 à 50.
4. Écrire et programmer un algorithme en Python qui épèle les lettres du mot « algorithme ».

Exercice 10 - Générer les tables de multiplication.

Les tables de multiplication, c'est bien de les connaître! Nous allons étudier un algorithme en Python qui permet de les générer automatiquement. Observer le code ci-dessous.

```
n = int(input('Donner un nombre entier : '))

for i in range(...,...) :
        print(n , 'x' , i , '=' , n*i)
```

1. Programmer cet algorithme, et préciser quelles valeurs il faut écrire dans le « range » à la ligne 3 pour obtenir la table complète.
2. Tester cet algorithme avec plusieurs valeurs de n.

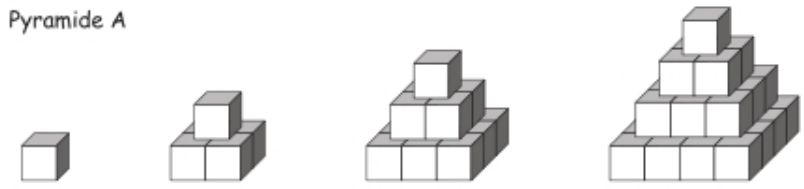
Exercice 11 - Réviser les tables de multiplication.

On souhaite écrire un algorithme qui nous interroge sur les tables de multiplications, et qui au final nous donne un note. Observer le code ci-dessous.

```
1 n = int(input('Quelle table voulez-vous réviser ? : '))
2
3 total_points = 0
4
5 for i in range(1,11) :
6     print(n , '*' , i , '=' ,'?')
7     reponse = int(input())
8     if reponse == ..... :
9         total_points = .....
10
11 print('Votre score est :' , total_points)
```

1. Que faut-il écrire au test de la ligne 8 pour que la réponse soit validée?
2. Que faut-il écrire à la ligne 9 pour qu'une réponse bonne soit comptée?
3. Tester cet algorithme avec plusieurs valeurs de n.

Exercice 12 - On construit une pyramide de cubes sur le modèle ci-dessous.



On souhaite savoir, en fonction de la hauteur de la pyramide, le nombre de cube qu'il faut prévoir. Par exemple, pour une hauteur de 1 cube, il faut prévoir 1 cube. Pour une hauteur de 2 cubes, il faut prévoir 5 cubes.

1. Indiquer à côté de chaque pyramide le nombre de cubes nécessaires.
2. On souhaite automatiser le calcul avec un algorithme.Observer le code ci-dessous.

```
n = int(input('Hauteur de la pyramide ? : '))

nombre_cubes = 0

for i in range(n) :
    nombre_cubes = nombre_cubes + .........

print('Nombre total de cubes' , nombre_cubes)
```

Compléter l'algorithme, à la ligne 6, pour que le nombre total de cubes soit correct.

Tester cet algorithme avec les réponses de la première question.

2.3 La boucle non bornée TANT QUE

A quoi sert la boucle non bornée TANT QUE?

Elle sert à exécuter une instruction, ou un bloc d'instruction, tant qu'une condition est réalisée. Le nombre de répétitions n'est pas connu à l'avance, il dépend de la condition.

La boucle non bornée « tant que » en Python : WHILE

```
1 while condition :
2         instruction(s)
```

Tant que la condition de la ligne 1 est vraie, les instructions après l'indentation seront effectuées. Dès que la condition n'est plus réalisée, on sort de la boucle.

Un exemple pour bien comprendre : on se pose la question : à partir de quand 2^n dépasse un nombre choisi. Observer l'algorithme ci-dessous. Tant que 2^n est inférieur au nombre choisi, on augmente n d'une unité. On sort de la boucle dès que 2^n est **supérieur** au nombre choisi. Programmer et tester cet algorithme plusieurs fois.

```
nombre = int(input('Donner un nombre : '))
n = 0
while 2**n < nombre :
    n = n+1

print('La valeur de n est ' , n, ' et on obtient ', 2**n)
```

Remarque si on veut répéter indéfiniment une suite d'instructions, on peut écrire while True :. Notons enfin que si l'on souhaite sortir d'une prématurément d'une boucle **for** ou **while**, on peut utiliser la commande break.

Exercice 13 - La suite de SYRACUSE.

Pour chaque nombre entier, on construit sa suite de SYRACUSE de la manière suivante :

- si le nombre est pair, on écrit le nombre divisé par 2 ;
- si le nombre est impair, on écrit le multiplie par 3 puis on ajoute 1 et on écrit le résultat ;
- on recommence avec le nouveau nombre obtenu...

Exemple : on choisit comme nombre de départ 10, on obtient alors la suite :

10 ; 5 ; 16 ; 8 ; 4 ; 2 ; 1 ; 4 ; 2 ; 1 ....

1. Construire la suite de Syracuse en prenant comme nombres de départ les valeurs suivantes :
   7 ; .....
2. Quelle conjecture peut-on faire pour les nombres obtenus avec la méthode de Syracuse ?
3. On cherche à programmer un algorithme qui nous permet de donner automatiquement les nombres de la suite de Syracuse. Voici un algorithme qui réalise cela en Python, utilisant la boucle TANT QUE.

```
import time
n=int(input("Donner un nombre entier :"))

while n!=1 :
    if (n%2==0):
        n=n/2
    else:
        n=3*n+1
    print(n)
    time.sleep(1)
```

(a) Traduire en langage naturel cet algorithme.

(b) Quand sort-on de la boucle .

(c) Programmer et tester cet algorithme avec la valeurs de l'exemple et des deux premières questions.

4. On souhaite savoir au bout de combien de termes calculés la suite donne le nombre 1 . On appelle ce nombre la « durée de vol », c'est à dire le nombre d'itérations (comprendre « répétitions ») nécessaires pour finalement obtenir 1. Compléter l'algorithme ci-dessus pour y parvenir, et le tester.

2.4 Les fonctions

Qu'est-ce qu'une fonction en Python ?

Comme en mathématiques, une fonction est un mécanisme, un processus, qui à partir de paramètres (appelés « antécédents » en mathématiques) retourne (on dit **associe** en mathématiques) un résultat (que l'on appelle **image** en mathématiques).
Quand on conçoit un algorithme, il est souvent pratique dès le départ de penser à regrouper sous forme de fonctions certaines séquences, qui peuvent après être utilisées plusieurs fois sans être ré-écrites. On appelle cela **factoriser** le code. Ainsi regroupé en modules, le code est facilement ré-utilisable.

Création d'une fonction en Python

```
1 def nom_de_la_fonction(var1,var2,...) :
2         """
3         informations sur la fonction
4         qui s'affichent avec help
5         """
6         instruction(s)
7         return resultat
```

Ligne 1 : l'instruction « def » permet de définir une fonction, son nom est donné juste après. Entre parenthèses, on donne les variables utilisées par cette fonction. On remarque aussi la présence de « : » qui marque le début du bloc.
Quand elle est appelée, la fonction exécute les instructions après l'indentation.
Ligne 3 : la fonction renvoie la valeur de la variable après l'instruction « return ».

Exemple de fonction : le volume d'un cône

```
1  def volume_cylindre(r,h) :
2          """
3          renvoit le volume d'un cylindre
4          avec le rayon et le volume
5          """
6          v = 3.14 * r**2 * h
7          return v
8
9  v1 = volume_cylindre(3,6)
10 print(v1)
```

Les lignes 1 à 7 définissent la fonction qui calcule le volume d'un cylindre à partir de son rayon r et sa hauteur h.
Ligne 9 : appel de la fonction, la variable $v1$ contient le volume d'un cylindre de rayon 3 et de hauteur 6.
Lignes 2 à 5 : documentation sur la fonction, accessible depuis la console avec **help**.

Une fonction peut aussi être **appelée à partir de la console**. Par exemple, si on saisit :

```
>>> volume_cylindre(7,8)
```

le résultat apparait directement dans la ligne de commande.
D'une manière générale, tout programme peut être écrit comme une fonction ! Si on veut afficher la documentation de la fonction, alors on sait dans la console :

```
>>> help(volume_cylindre)
```

C'est pratique en particulier pour savoir dans quel ordre doivent donnés les arguments.

Programmer l'algorithme ci-dessus, et vérifier qu'il fonctionne. Ensuite avec la ligne de commande, faire plusieurs essais d'appels de cette fonction.

Notion de variable locale et de variable globale

```
def test():
    b = 5
    c = 8
    print(a, b)


a = 2
b = 7
print(a,b)
test()
print(a, b)
print(c)
```

Dans une fonction on peut être amené à définir des variables. Il faut savoir que ces variables sont **LOCALES** , c'est à dire qu'elles servent pendant l'exécution de la fonction uniquement, et sont après détruites. Enfin il faut savoir que si une variable b est utilisée dans un programme, le fait de la redéfinir dans une fonction ne change pas la valeur initiale b. C'est ce que montre l'algorithme ci-contre, que vous pouvez tester. Vous pouvez alors constater que l'affichage de la variable c génère une erreur.

Exercice 14 - Une fonction volume_sphere.

Sur le modèle de l'exemple ci-dessus, programmer une fonction qui, à partir du rayon r associe le volume de la sphère correspondante.

Exercice 15 - Conversions.

Pour convertir une température exprimée en Fahrenheit en son équivalent en Celsius, il faut utiliser la formule :

$$C = (F - 32) \times \frac{5}{9}$$

1. Écrire une fonction « celsius » qui, à partir de la température en Fahrenheit renvoie la température en Celsius.
2. Écrire une fonction « fahrenheit » qui fait l'inverse.
3. Afficher une table de correspondance entre -20°C et 50°C, avec un pas de 1°C.

2.5 Simuler des expériences aléatoires avec Python

Tous le monde connait le jeu suivant : une personne choisit un nombre au hasard, puis demande à une autre de le deviner. A chaque proposition, on indique si c'est en dessous ou bien au dessus. On note finalement le nombre de propositions qu'il a fallu pour trouver le bon résultat.
L'encadré ci-dessous donne quelques exemples de fonctions en Python qui permettent de simuler des nombres aléatoires. On remarque que la première ligne comporte l'importation de la bibliothèque « random », qui regroupe les fonctions liées au hasard.

Comment simuler le hasard avec Python ?

```
1  from random import *
2
3  a = randint(0,1)
4
5  def simule_de():
6          return randint(1,6)
7
8  b = simule_de()
9
10 c = random()
11
12 print(a)
13 print(b)
14 print(c)
```

La première indique que l'on importe toutes les fonctions de la bibliothèque « random ».
La fonction **randint(a,b)** renvoie une nombre entier aléatoire entre les entiers a et b inclus.
Ligne 3 : la variable a contient 0 ou 1, avec la même chance pour l'un ou l'autre.
Ligne 5 : la fonction **simule_de** renvoie un nombre entier entre 1 et 6 inclus, avec la même probabilité pour chaque : elle simule donc le jet d'un dé cubique.
Ligne 10 : la fonction **random** renvoie un nombre **décimal** aléatoire compris entre 0 et 1, avec une probabilité uniformément répartie entre 0 et 1. Elle n'a pas d'argument.

A faire : programmer les lignes de code de l'encadré, puis vérifier que les résultats donnés correspondent avec ce qui est attendu.

Exercice 16 - Devine le nombre que j'ai choisi entre 1 et 100.

1. En langage naturel, rédiger un algorithme qui simule le jeu décrit en haut de page.
2. Programmer et tester plusieurs fois cet algorithme sur une machine.

Compléments ...

Exercice 17 - Dedans ou Dehors.

Écrire une programme qui :

- demande trois valeurs décimales a, b et c ;
- affiche « Dedans » si b est compris entre a et c, ou bien « Dehors » sinon.

Attention : rien n'oblige que a soit plus petit que c. Pour vérifier votre algorithme, procéder aux tests suivants.

Tests...

a = 1 ; b = 2 ; c = 3 → « Dedans » ; a = 3 ; b = 2 ; c = 1 → « Dedans »
a = 1 ; b = 4 ; c = 3 → « Dehors » ; a = 3 ; b = 4 ; c = 1 → « Dehors »

Exercice 18 - Utilisation d'un accumulateur : calcul d'une moyenne.

On souhaite écrire une programme qui :

- demande combien de valeurs (décimales) sont prévues (n) ;
- demande chaque valeur en les numérotant en 1 à n ;
- affiche la moyenne des valeurs.

Tests...

n = 3 ; valeur 1 = 12 ; valeur 2 = 15 ; valeur 3 = 18 → moyenne = 15

Exercice 19 - Le jeu du mölkky.

Au jeu du mölkky, chaque joueur marque à son tour entre 0 et 12 points, qui s'ajoutent au score précédent. Le premier à attendre 51 gagne, **mais** s'il dépasse 51, son score revient à 25. On souhaite écrire une programme qui :

- demande le score actuel du joueur ;
- demande le gain en cours ;
- affiche finalement le nouveau score, et éventuellement la victoire.

Tests...

score = 40 ; gain = 10 → nouveau score = 50 ; score = 40 ; gain = 11 → gagné ! ;
score = 40 ; gain = 15 → dommage ! nouveau score = 25 ;

Exercice 20 - Diviseurs d'un nombre entier.

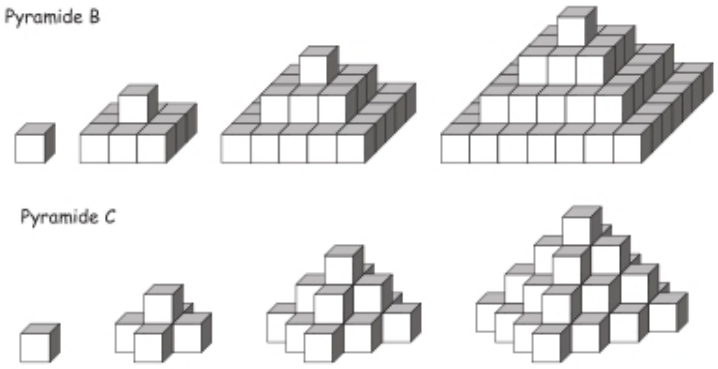
Écrire un programme qui demande un nombre entier n, puis qui affiche la liste de ses diviseurs.

Tests...

n = 5 → {1 ;5} ; n = 10 → {1 ; 2 ; 5 ; 10 } ; n = 12 → {1 ; 2 ; 3 ; 4 ; 6 ; 12}

Exercice 21 - Pyramides de cubes... la suite.

Voici plusieurs pyramides de cubes. Sur le modèle de la pyramide A, proposer pour la B et C un algorithme qui demande la hauteur de la pyramide, et qui affiche finalement le nombre total de cubes. Pour tester votre programme, commencer par calculer « à la main » le nombre de cubes de chaque pyramide, puis vérifier les valeurs obtenues avec votre programme.



Exercice 22 - Une suite croissante.

On s'intéresse à la suite de nombre générée de la manière suivante .

$$(1,2^0;1,2^1;1,2^3;1,2^4...)$$

Les nombres qui constituent cette suite sont les puissances successives de $1,2$. C'est une suite croissante, et on admettra que ses valeurs tendent vers l'infini.
Programmer un algorithme qui lit une valeur « v » donnée par l'utilisateur, et qui renvoie LA valeur de « n » pour laquelle :

$$1,2^n > v$$

Application : déterminer n de sorte que $1,2^n > 10^6$

Exercice 23 - Une suite décroissante.

On s'intéresse à la suite de nombre générée de la manière suivante .

$$(0,8^0;0,8^1;0.8^3;0.8^4...)$$

Les nombres qui constituent cette suite sont les puissances successives de $0,8$. C'est une suite décroissante, et on admettra que ses valeurs tendent vers zéro.
Programmer un algorithme qui lit une valeur « v » donnée par l'utilisateur, et qui renvoie LA valeur de « n » pour laquelle :

$$0,8^n < v$$

Application : déterminer n de sorte que $0,8^n < 10^{-6}$

Exercice 24 - Somme de cubes.

On s'intéresse à la somme S_n des n premiers cubes.

$$S_n = 1 + 2^3 + 3^3 + 4^3 + 5^3 + ... + n^3$$

Programmer une fonction somme_cube(n) qui renvoie la valeur de S_n. Tester votre fonction avec les valeurs n allant de 1 à 4.