La **programmation**, appelée aussi **codage**, est l'écriture de **programmes informatiques**, c'est-à-dire la description d'un algorithme dans un langage de programmation compréhensible par une machine et par un humain.

Un **langage de programmation** est un moyen de communication avec l'ordinateur mais aussi entre programmeurs ; les programmes étant d'ordinaire écrits, lus et modifiés par des équipes de programmeurs.

Il existe des centaines de langages de programmation¹, par exemple C, C++, JavaScript, Java, PHP, etc. Le langage utilisé dans ce chapitre est Python.

Python² est un langage informatique inventé par Guido Van Rossum. La première version publique date de 1991. Il est multiplateforme (Linux, MacOS, Windows, android, iOS), libre et gratuit, mis à jour régulièrement (version actuelle 3.11³).

1. Variables et affectation

1.1. Variables

L'informatique désigne le traitement automatique de l'information⁴. Dans les programmes informatiques, l'information est représentée par des données.



- Cours

Les programmes informatiques manipulent des données en utilisant des **variables**. Une variable permet d'associer un nom à une valeur.

Une variable informatique peut se concevoir comme une sorte de "boite" étiquetée avec un nom, dans laquelle un programme enregistre une valeur pour la consulter ou la modifier pendant son exécution⁵.

En Python, comme dans la plupart des langages informatiques, le nom d'une variable :

- s'écrit en lettres minuscules (« a » à « z ») et majuscules (« A » à « z ») et peut contenir des chiffres (« 0 » à « 9 ») et le caractère blanc souligné (« _ »);
- ne doit pas comporter d'espace, de signes d'opération « + », « », « * » ou « / », ni de caractères spéciaux comme des signes de ponctuation « ' », « " », « , », «
 . », « : », « @ », etc.;
- · ne doit pas commencer par un chiffre ;
- ne doit pas être un mot réservé de Python, par exemple « for », « if », « print », etc.; et
- est sensible à la casse, ce qui signifie que les variables « TesT », « test » ou « TEST » sont différentes.

En pratique cela permet d'éviter les noms de variable réduits à une lettre et d'utiliser des noms qui ont un sens!





PEP 8

La PEP 8 ⁶ donne un grand nombre de recommandations de style pour écrire du code Python agréable à lire et recommande en particulier de nommer les variables par des mots en minuscule séparés par des blancs soulignés () ⁷, par exemple d'appeler une variable somme_des_nombres plutôt que s dans un programme qui additionne des nombres.

1.2. Types de variable



- Cours

Les variables peuvent être de types différents en fonction des données qu'elles représentent.

Les principaux types de variable sont :

- les nombres entiers (type int);
- les nombres décimaux, appelés « flottants » (type float) qui s'écrivent toujours avec un point (1. le séparateur décimal est un point, pas une virgule), par exemple 5.0.

Noter que .5 et 5. permettent d'écrire rapidement les flottants 0.5 et 5.0 et que 2e5 ou 2E5 (pour 2×10^5) permettent d'écrire le nombre flottant 200000.0;

- les booléens prenant seulement les valeurs True ou False (type bool);
- les textes ou chaines des caractères (type str) écrits entre une paire de guillemets (") ou d'apostrophes (');
- d'autres types dits "construits" comme les p_uplets, tableaux, dictionnaires⁸, etc.

13 Affectation



- Cours

L'affectation consiste à donner une valeur à une variable. En Python, comme dans la plupart des langages informatiques, l'affectation d'une valeur à une variable est représentée par le signe « = ».9

Par exemple, saisir les commandes suivantes dans la console Python permet d'affecter les valeurs 3 (type int), 3.0 (type float) et "3" (type str) à des variables nommées respectivement a, b et c :

La console Python, ou interpréteur Python, est un moyen rapide d'exécuter des commandes. Il suffit de taper une instruction en réponse à l'invite >>> puis d'appuyer sur la touche « Entrée » pour lancer son exécution.



PEP 8

Mettre des espaces autour d'un égal (=).

```
>>> a = 3
>>> b = 3.0
>>> c = "3"
```

En Python, c'est l'affectation qui définit le type d'une variable 10.

⚠ C'est bien la valeur qui se trouve à droite du signe « = » qui est affectée à la variable à gauche, et pas dans l'autre sens.

```
>>> 3 = a
 File "<interactive input>", line 1
SyntaxError: can't assign to literal
```

PEP 8

Mettre un espace après une virgule (,), mais pas avant.

Il est aussi possible d'affecter des valeurs à plusieurs variables en même temps en une seule ligne.

```
>>> a, b = 3, 4
>>> a
3
>>> b
```

et d'affecter la valeur d'une variable à une autre variable, par exemple :

Quand on tape le nom d'une variable dans la console, elle affiche sa valeur.

```
>>> a = 3
>>> b = a
>>> b
3
```

⚠ Il n'est pas possible d'utiliser une variable avant de lui avoir affecté une valeur.

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

? Exercice corrigé

On affecte les valeurs 5 et 6 (de type int) à deux variables nommées respectivement a et b :

```
>>> a = 5
>>> b = 6
```

Compléter ces instructions pour échanger les valeurs de a et de b (sans utiliser les chiffres 5, 6 ou tout autre chiffre).

Réponse

1. Voyons d'abord la solution qui ne fonctionne pas :

```
>>> a = b
>>> b = a
>>> a
6
>>> b
```

Cette solution ne fonctionne pas car la valeur intiale de a, c'est-à-dire 5, est perdue quand on écrit a = b, on dit qu'elle est "écrasée" par la valeur de b. Ensuite quand on écrit b = a on affecte la nouvelle valeur de a, c'est-à-dire 6 au lieu de 5, à b.

2. Une première solution consiste donc à utiliser une autre variable pour conserver la valeur intiale de a temporairement, appelons la temp :

```
>>> temp = a
```

Et ensuite de faire l'échange :

```
>>> a = b
>>> b = temp
>>> a
6
>>> b
```

Notons que la dernière instruction affecte la valeur de temp, c'est-à-dire 5, à b, et pas la valeur de a puisqu'elle vaut 6 à ce moment là.

3. Une deuxième solution plus élégante consiste à utiliser l'affectation de plusieurs variables sur une seule ligne :

```
>>> a, b = b, a
>>> a
6
>>> b
5
```

4. Pour information, il existe une troisième solution un peu plus compliquée, qui n'utilise ni variable temporaire, ni affectation de plusieurs variables en une seule ligne :

```
>>> a = a + b

>>> b = a - b

>>> a = a - b

>>> a

6

>>> b
```

2. Opérations, comparaisons, expression

2.1. Opérateurs arithmétiques sur les nombres

Les opérations arithmétiques usuelles sont effectuées sur des nombres de types int ou float :

opérateur	notation
addition	a + b
soustraction	a - b
multiplication	a * b
puissance	a**b
divisions décimale	a / b

```
>>> a = 5
>>> b = 2
```



PEP 8

Entourer les opérateurs mathématiques (+, -, /, *) d'un espace avant et d'un espace après.

```
>>> a + b
>>> a / b
2.5
>>> a**b
25
```

À noter :

Si a et b sont deux variables toutes les deux de type int alors le résultat d'une opération entre les deux est de type int, sauf pour la division qui est toujours de type float même si le résultat est un entier :

```
>>> 10 / 5
2.0
```

et si l'un de a ou de b est de type float alors le résultat est toujours de type float.

La racine carrée d'un nombre peut s'obtenir avec : a**0.5 11.

L'ordre des priorités mathématiques est respecté.

Il est possible d'affecter une valeur à une variable qui dépend de son ancienne valeur, par exemple l'augmenter d'une quantité donnée (on dit incrémenter)12.

```
>>> a = 3
>>> a = a + 1
>>> a
```



Dans ce cas particuliers, on peut omettre les espaces autour de la multiplication (*) pour montrer la priorité sur l'addition et améliorer la lisibilité de la formule.

```
>>> a = 2*a + 1
>>> a
```

12

Des raccourcis d'écriture existent pour aller plus vite (mais attention aux erreurs en les utilisant!).

```
• a += 1 signifie a = a + 1;
```

- a += b signifie a = a + b ; et
- a *= 2 signifie a = a * 2.

2.1.1. Division entière (ou division euclidienne)

L'opérateur de division entière // et l'opération modulo % utilisés avec des entiers (de type int) donnent respectivement le quotient et le reste d'une division euclidienne : si a et b sont des entiers tels que $a = b \times q + r$, alors a // b donne q et a % b donne r^{13} .

opérateur	notation
quotient	a // b
reste	a % b

Par exemple, le quotient et le reste de la division entière de 17 par 5 sont 3 et 2 respectivement (car $17 = 3 \times 5 + 2$):

```
>>> a = 17
>>> b = 5
>>> a // b
>>> a % b
```

L'opérateur modulo, %, qui donne le reste d'une division entière, est très utile pour déterminer si un nombre est divisible par un autre nombre, dans ce cas le reste est égal à zéro :

```
>>> 10 % 5
>>> 10 % 3
```

10 est divisible par 5 mais pas par 3.

2.2. Opérateurs sur les chaines de caractères

Les textes ou chaines des caractères, de type str (abréviation de *string*), sont définis entre une paire de guillemets (") ou d'apostrophes (') ¹⁴.

```
>>> chaine1 = 'Hello '
>>> chaine2 = "world"
```

Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication 15 :

• L'opérateur d'addition « + » concatène (assemble) deux chaînes de caractères.

```
>>> chaine1 + chaine2
'Hello world'
```

16

• L'opérateur de multiplication « * » entre un nombre entier et une chaîne de caractères **duplique** (répète) plusieurs fois une chaîne de caractères.

```
>>> chaine1 * 3
'Hello Hello '
```

La fonction len() donne le nombre de caractère d'une chaine (y compris les espaces et les signes de ponctuation).

```
>>> ch = 'Hello world'
>>> len(ch)
11
```

Chaque caractère d'une chaine de caractères ch a une position qui va de 0 à len(ch) - 1.

- ch[0] permet d'accéder au premier caractère en position 0 de la chaine ch,
- ch[1] au second caractère en position 1,
- ...
- ch[i] au caractère en i ième position,
- •
- ch[len(ch) 1] au dernier caractère.

⚠ Les positions sont comptées en commençant à la position 0, le premier caractère est ch[0] et non pas ch[1]!

```
>>> ch[6]
'w'
```

• De même, en partant de la fin, ch[-1] permet d'accéder au dernier caractère, ch[-2] à l'avant dernier, etc.

```
>>> ch[-<mark>1</mark>]
'd'
```



Pas d'espace autour d'un deux-points (:).

• Enfin ch[i:j] permet d'obtenir la sous-chaîne de tous les caractères entre les positions i (inclus) et j (exclus), appelée une tranche.

```
>>> ch[2:5]
'llo'
```

Les mots-clés in et not in permettent de vérifier l'appartenance, ou pas, d'une sous-chaine dans une chaine :

```
>>> "py" in "python"
True
>>> "Py" not in "python"
True
```

Il existe de nombreuses méthodes 17 pour traiter les chaines de caractères, quelques exemples :

fonction	description	exemple
.index('c')	trouve l'index du premier caractère "c" dans une chaîne.	<pre>>>> chaine = 'aaabbbccc' >>> chaine.index('b') 3</pre>
.find('sc')	cherche la position d'une sous-chaîne sc dans la chaîne.	<pre>>>> chaine.find('bc') 5</pre>
.count('sc')	compte le nombre de sous-chaînes sc dans la chaîne.	<pre>>>> chaine.count('bc') 1</pre>
.lower('sc')	onvertit une chaîne en minuscules.	<pre>>>> 'ABCdef'.lower() 'abcdef'</pre>
.upper('sc')	onvertit une chaîne en majuscules.	<pre>>>> 'ABCdef'.upper() 'ABCDEF'</pre>
.replace('old', 'new')	remplace tous les caractères old par new dans la chaîne.	<pre>>>> 'aaabbbccc'.replace('c', 'e') 'aaabbbeee'</pre>

2.3. Opérateurs de comparaison

Les opérations de comparaison usuelless permettent de comparer des valeurs de même type entre elles. Le résultat est toujours un booléen (de type bool) égal à True ou False 18.



Entourer les opérateurs de comparaison (==, !=, >=, etc.) d'un espace avant et d'un espace après.

opérateur	notation
=	a == b
≠	a != b
<	a < b
≤	a <= b
>	a > b
2	a >= b

19

⚠ Une erreur courante consiste à confondre l'opérateur de comparaison == pour vérifier si deux valeurs sont égales avec l'affectation qui utilise le signe = !

```
>>> a, b, c = 5, 5, 6
>>> a == b
True
>>> a == c
False
```

Il est possible de combiner les comparaisons, par exemple pour vérifier si a est compris entre 2 et 6 :

```
>>> 2 <= a < 6
True
```

entre 7 et 8:

```
>>> 7 < a < 8
False
```

mais ce n'est pas recommandé car c'est en fait une combinaison de plusieurs comparaisons, ce qui peut donner des hérésies mathématiques :

```
>>> 4 < a > 2
True
```

Les chaines de caractères, quant à elles, sont comparées en ordre lexicographique, c'est-à-dire caractère par caractère comme l'ordre des mots dans un dictionnaire : on commence par comparer le premier caractère de

chaque chaîne, puis en cas d'égalité le deuxième de chaque, et ainsi de suite jusqu'à trouver un caractère qui est différent de l'autre²⁰.

```
>>> 'aa'>'ab'
False
>>> "python" == "python"
True
>>> "python" != "PYTHON"
True
```

⚠ Attention aux majuscules (elles sont "avant" toutes les minuscules) :

```
>>> "java" < "python"
True
>>> "java" > "Python"
True
```

et aux nombres écrits dans des chaînes de caractères :

```
>>> "10" < "2"
True
```

Les nombres de type int ou float peuvent être comparés entre eux même s'ils sont de types différents :

```
>>> 7 == 7.0
True
>>> 0.0 < 1
True
```

Mais pas les nombres avec les chaines de caractères :

```
>>> 7 == "7"
False
>>> 7 < '8'
Traceback (most recent call last):
   File "<interactive input>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'</pre>
```

⚠ Attention aux égalités entre nombres de type float qui ne sont pas toujours encodés de façon exacte²¹ :

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
```

2.4. Opérateurs logiques (ou booléens)

Les opérations logiques peuvent être effectuées sur des booléens (type bool). Le résultat est un booléen égal à True OU False .

opérateur	notation	description	priorité
Négation de a	not a	True si a est False, False sinon	1

opérateur	notation	description	priorité
a et b (conjonction)	a and b	True Si a et b sont True tous les deux, False sinon	2
a ou b (disjonction)	a or b	True si a ou b (ou les deux) est True, False sinon	3

(a et b sont des booléens).

Comme pour les opérations mathématiques, les opérations logiques suivent des règles de priorité :

- 1. Négation (not),
- 2. Conjonction (and),
- 3. Disjonction (or).

a or not b and c est équivalent à a or ((not b) and c) mais en pratique les parenthèses sont plus lisibles.

2.5. Expressions



Cours

Une **expression** (ne pas confondre avec une instruction) est un calcul d'opérations et de comparaisons qui donne une valeur.

Exemples:

- 2*a + 5 est une expression, elle a une valeur (qui dépend de la valeur de a).
- a == 5 est une expression booléene, elle vaut True ou False.
- a = 5 n'est **pas** une expression, c'est une affectation de la valeur 5 à la variable a.

À noter:

Quand une affectation est saisie dans la console Python, par exemple >>> a = 5, rien n'est affiché par l'interpréteur car ce n'est pas une expression.



Quand une expression est saisie dans la console Python, par exemple >>> a == 5, elle est évaluée par l'interpréteur et le résultat est affiché en dessous.

Puisqu'elle a une valeur, une expression peut être affectée à une variable : b = a**2 est une affectation de la valeur de l'expression a**2 (le carré de a) à la variable b .

?) Exercice corrigé

La valeur d'une variable annee de type int est donnée, par exemple >>> annee = 2023.

Ecrire dans l'interpréteur une expression booléenne, qui vaut True si annee est une année bissextile ou False sinon.

- « Depuis l'ajustement du calendrier grégorien, l'année sera bissextile (elle aura 366 jours) seulement si elle respecte l'un des deux critères suivants :
 - 1. C1 : l'année est divisible par 4 sans être divisible par 100 (cas des années qui ne sont pas des multiples de 100) ;
 - 2. C2 : l'année est divisible par 400 (cas des années multiples de 100).

Si une année ne respecte ni le critère C1 ni le critère C2, l'année n'est pas bissextile ». Source: https://fr.wikipedia.org/wiki/Année_bissextile.

Réponse



Avant d'écrire cette expression on peut se poser quelques questions :

• Comment savoir si un nombre est divisible par un autre ? Il suffit de vérifier si le reste de la division entière est égal à zéro ou pas. Par exemple 2023 n'est pas divisible par 4 car le reste de la division entière de 2023 par 4 est 3 :

```
>>> annee = 2023
>>> annee % 4
```

Par contre 2024 est divisible par 4 car le reste de la division entière de 2024 par 4 est bien 0 :

```
>>> annee = 2024
>>> annee % 4
```

- On peut traduire directement en Python chaque condition C1 et C2 :
 - C1 : l'année est divisible par 4 sans être divisible par 100 (cas des années qui ne sont pas des multiples de 100) ;

```
>>> annee % 4 == 0 and annee % 100 != 0
```

C2 : l'année est divisible par 400 (cas des années multiples de 100).

```
>>> annee % 400 == 0
```

• 1 la dernière clause indique qu'une année n'est pas bissextile si les conditions C1 et C2 sont toutes les deux fausses. Il faut donc comprendre qu'une année est bissextile si l'une des conditions C1 ou C2 est vraie (ou les deux en même temps).

Traduit en Python, on obtient l'expression suivante que l'on peut tester dans la console.

On pourrait se passer des parenthèses et utiliser les règles de priorités des opérateurs booléens : annee % 4 == 0 and annee % 100 != 0 or annee % 400 == 0 , mais en pratique ce n'est pas recommandé.

```
>>> annee = 2023
>>> (annee % 4 == 0 and annee % 100 != 0) or annee % 400 == 0
False
```

3. Instructions



Cours

Une instruction (ne pas confondre avec une expression) est une commande qui doit être effectuée par un programme.

Une séquence est une suite d'instructions.

Par exemple:

- a = 2 est une instruction qui affecte la valeur 2 à la variable a.
- print('Hello world') est une instruction qui affiche la chaine 'Hello world' dans la console.
- a == 2 n'est pas une instruction, c'est une expression qui compare la valeur de a à la valeur 2.

3.1. type()

La fonction type() permet de connaître le type d'une variable.²²



Pas d'espace avant et à l'intérieur des parenthèses d'une fonction.

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> y = 2.0
>>> type(y)
<class 'float'>
>>> z = '2'
>>> type(z)
<class 'str'>
```

3.2. Conversion de type

Les fonctions suivantes permettent de convertir une variable d'un type à un autre :

<pre>int() Convertit une chaine de caractères ou un flottant en entier. >>> int(2.8) 2 >>> int('2') 2</pre>	fonction	description	exemple
>>> int('2')	<pre>int()</pre>	Convertit une chaine de caractères ou un flottant en entier.	
2			
			2

fonction	description	exemple
float()	Convertit une chaine de caractères ou un entier en flottant.	>> float(5) 5.0 >>> float('5.5') 5.5
str()	Convertit un entier ou un flottant en une chaine de caractères.	>>> str(5.5)

Observons dans la console comment une variable de type float qui a une valeur entière est affiché avec un point :

```
>>> a = 5
>>> a
5
>>> float(a)
5.0
```

3.3. Instructions d'entrée et sortie

- Cours

Une instruction d'entrée permet à un programme de lire une valeur saisie au clavier par l'utilisateur. Une instruction de sortie affiche un message sur l'écran de l'utilisateur.

En Python, la fonction input() permet d'écrire une instruction d'entrée qui affecte la valeur saisie par l'utilisateur à une variable.

```
>>> saisie = input('Saisir un message')
>>> saisie
'abc'
```

La valeur renvoyée par input() est toujours du type str:

```
>>> nombre entier = input('Entrez un nombre entier')
>>> nombre_entier
'25'
```

lci la valeur affectée à nombre_entier est une chaine de caractères : '25' . Pour obtenir un nombre, de type int ou float, afin de faire des calculs par la suite par exemple, il faut la convertir :

```
>>> nombre_entier = int(input('Entrez un nombre entier'))
>>> nombre_entier
25
```

Si l'utilisateur ne saisit pas un nombre entier, cette instruction génère un message d'erreur.

Une instruction de sortie s'écrit en utilisant print() pour afficher à l'écran des chaines de caractère et/ou des variables, séparés par des virgules.



PEP 8

Un espace après une virgule (,), mais pas avant.

```
>>> print('Hello')
Hello
>>> message='world'
>>> print('Hello', message)
Hello world
>>> nombre = 5
>>> print(nombre)
>>> print('le nombre est', nombre)
le nombre est 5
>>> a = 5
>>> b = 6
>>> print('la somme de', a, 'et de', b, 'est', a + b)
la somme de 5 et de 6 est 11
```

Par défaut, print() provoque un retour à la ligne après chaque affichage. Pour changer ce comportement il faut préciser la fin de l'affichage en ajoutant un paramètre end= suivi d'une chaine de caractères, par exemple un espace end=' ' ou même une chaine vide end=''.

```
>>> print('Hello', end=' ')
Hello >>>
```

Python 3.6 a introduit les chaine de caractères f-strings (formatted string) qui s'écrivent avec f devant et permettent d'y insérer des variables, ou même des expressions, entre accolades.

```
>>> prenom = 'Paul'
>>> annee_naissance = 2010
>>> print(f'Bonjour {prenom}, vous avez {2023 - annee_naissance} ans')
Votre nom est un Paul et vous avez 13 ans
```

3.4. Premier programme

Pour permettre à l'utilisateur d'entrer son prénom et sa date de naissance et d'affecter ses réponses aux variables prenom et annee naissance, il faut à chaque fois écrire dans la console les instructions suivantes :

```
>>> prenom = input('Entrez votre prénom : ')
Entrez votre prénom : Paul
>>> annee naissance = int(input('Entrez votre date de naissance : '))
Entrez votre date de naissance : 2010
>>> print(f'Bonjour {prenom}, vous avez {2023 - annee_naissance} ans')
Bonjour Paul, vous avez 13 ans
>>>
```

Cette séquence montre les limites de la console, qui répond à des commandes de façon interactive, mais ne permet pas d'écrire un programme élaboré!

Ouvrons IDLE (/python/Lib/idlelib/idle.bat) pour écrire un premier programme, l'interpréteur de commande avec l'invite Python >>> apparaît :

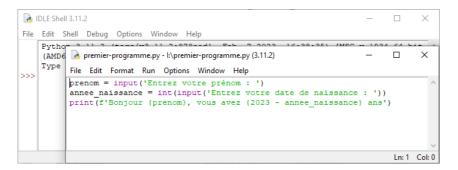
Python propose par défaut un **IDE** (pour *Integrated Development Environment*) appelé IDLE. Il existe de nombreux IDE, certains dédiés à Python comme PyScripter, Thonny, etc. et d'autres généralistes comme VS Codium, VS Code, etc. acceptant plusieurs langages informatiques.

```
File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878eadl, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit ^ (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
```

Ouvrons un nouveau fichier avec le menu File/New pour entrer le programme Python suivant :

```
prenom = input('Entrez votre prénom : ')
annee_naissance = int(input('Entrez votre date de naissance : '))
print(f'Bonjour {prenom}, vous avez {2023 - annee_naissance} ans')
```



Enregistrons le programme dans nos fichiers avec le menu File/Save As puis Run/Run Module pour exécuter le programme. Le résultat est affiché dans la console :

Nous avons écrit notre premier programme informatique!

Notons au passage une différence importante entre l'affichage d'une variable depuis la console et depuis un programme :

Depuis la console

Il suffit de saisir le nom de la variable à l'invite de commande pour afficher sa valeur :

```
>>> a = 5
>>> a
5
```

Depuis un programme

Le programme suivant n'affiche rien dans la console :

```
1 a = 5
2 a
```

```
File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD ^ 64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>>

RESTART: I:\tilde{Interval between the company to the
```

Depuis un programme avec print()

Il faut utiliser l'instruction print() dans un programme pour afficher la valeur d'une variable dans la console.

```
1 a = 5
2 print(a)
```

```
File Edit Shell Debug Options Window Help

Python 3.11.2 (tags/v3.11.2:878eadl, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64) |
] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>>

Type "help", "copyright", "credits" or "license()" for more information.

File Edit Format Run Options Window Help

a = 5
print(a)
```

? Exercice corrigé

Pour passer d'un pixel couleur codé RGB (mélange des trois couleurs rouge, vert, bleu) à un pixel en nuance de gris, on utilise la formule suivante qui donne le niveau de gris : $G=0,11\times R+0,83\times V+0,06\times B$ où R,V et B sont les niveaux de rouge, vert et bleu.

Ecrire le programme qui demande en entrée les 3 couleurs d'un pixel et affiche en sortie la nuance de gris.

Réponse

~

Quelques questions à se poser avant d'écrire le programme demandé :

- Quelles sont les informations à saisir par l'utilisateur ? Les trois niveaux de couleurs R, V et B.
- Où stocker ces informations ? Dans trois variables de type int nommées par exemple R, V et B comme dans la formule.
- Que doit calculer le programme ? Le niveau de gris calculé en utilisant la formule et stocké dans une variable, nommée par exemple G, de type int.
- Que doit faire ensuite le programme ? Le programme doit afficher le niveau de gris.

Traduit en Python, le programme s'écrit simplement :

Noter la présence de commentaires dans le code, commençant par le signe #, ils sont ignorés par l'interpréteur Python.

Essayer le programme sans faire la conversion des variables R, V et B en int et constater l'erreur produite.

```
# Demande les 3 couleurs R, V et B de type int
R = int(input('Rouge:'))
V = int(input('Vert:'))
B = int(input('Bleu:'))
# Calcule le niveau de gris G, de type int
G = int(0.11 * R + 0.83 * V + 0.06 * B)
# Affiche le niveau de gris
print(f'Le niveau de Gris est {G}')
```

4. Constructions élémentaires



- Cours

En Python, l'indentation, ou décalage vers la droite du début de ligne, délimite les séquences d'instructions et facilite la lisibilité en permettant d'identifier des blocs. La ligne précédant une indentation se termine toujours par le signe deux-points.



Préférer les espaces aux tabulations.

L'indentation est normalement réalisée par quatre caractères « espace ». Python accèpte aussi une tabulation, voire un seul ou un autre nombre d'espaces, mais dans tous les cas il faut être consistant à travers tout un programme.

4.1. Instructions conditionnelles

- Cours

Une instruction conditionnelle exécute, ou pas, une séquence d'instructions suivant la valeur d'une condition (une expression booléenne qui prend la valeur True ou False).

```
if condition:
   instructions
```

Par exemple, ce programme détermine le stade de la vie d'une personne selon son age.



PEP 8

Pas d'espace avant le deux-points (:).

```
age = int(input("Quel age avez-vous ?"))
   if age >= 18:
       stade = "adulte"
3
    print(f"Vous êtes un {stade}")
4
       print(f"Vous avez {age} ans")
```

Ne pas oublier les deux-points « : » après la condition et l'indentation sur les lignes suivantes.

L'indentation détermine la séquence à exécuter (ou pas), dans ce cas les instructions qui suivent aux lignes 3, 4 et 5. Quand la condition (l'expression booléenne) age >= 18 n'est pas vérifiée, aucune des trois instructions n'est exécutée.

Il est possible de ne pas indenter l'instruction en ligne 5 print(f"Vous avez {age} ans"), dans ce cas elle ne ferait plus partie de l'instruction conditionnelle et serait alors exécutée dans tous les cas.

```
age = int(input("Quel age avez-vous ?"))
if age >= 18:
    stade = "adulte"
    print(f"Vous êtes {stade}")
print(f"Vous avez {age} ans")
```

Par contre, si l'instruction en ligne 4 print(f"Vous êtes {stade}") n'était pas indentée, la variable stade ne serait pas définie quand la condition n'est pas vérifiée et dans ce cas il y aura un message erreur à la ligne 4.

La structure if-else permet de gérer le cas où la condition est fausse :

```
if condition:
    instructions
else:
    instructions_sinon
```

L'instuction else n'a pas de condition, elle est toujours suivie des deux-points « : ».

```
1  age = int(input("Quel age avez-vous ?"))
2  if age >= 18:
3    stade = "adulte"
4  else:
5    stade = "enfant"
6  print(f"Vous êtes {stade}")
7  print(f"Vous avez {age} ans")
```

Dans ce cas, la variable stade est toujours définie, et l'instruction en ligne 6 print(f"Vous êtes {stade}") peut ne pas être indentée.

La structure if-elif-else permet de remplacer des instructions conditionnelles imbriquées pour gérer plusieurs cas distincts :

```
if condition_1:
    instructions_si_1
elif condition_2:
    instructions_si_2
elif condition_3:
    instructions_si_3
...
else :
    instructions_sinon
```

Ces deux programmes font exactement la même chose, mais le second est plsu lisible :

Instructions conditionnelles imbriquées

```
1  if age >= 18:
2    stade = "adulte"  # au dessus de 18
3  else:
```

```
4
        if age >= 12:
          stade = "ado" # entre 12 et 18
5
6
      else:
          if age >= 2:
7
              stade = "enfant" # entre 2 et 12
8
9
           else :
              stade = "bébé" # moins de 2
10
11
12
   print(f"Vous êtes {stade}, vous avez {age} ans")
```

Chaque fois qu'une condition if n'est pas vérifiée, le programme exécute toute la séquence else correspondante. Il "descend" ainsi de suite dans les conditions imbriquées jusqu'à ce qu'une condition soit vérifiée, et à ce stade il sort de tous les blocs conditionnels et reprend à l'instruction qui suit tous les blocs conditionnels imbriqués, ici à ligne 12 print(f"Vous êtes {stade}, vous avez {age} ans").

Par exemple, si on affecte la valeur 10 à la variable age, la première condition en ligne 1 if age >= 18: est fausse, le programme exécute donc toute la partie indentée après le premier else: correspondant en ligne 3. Il passe à la seconde condition en ligne 4 if age >= 12: qui est encore fausse, il "passe" donc à la séquence indentée après le else: correspondant en ligne 6. La troisième condition en ligne 7 if age >= 2: est cette fois vraie, il exécute la ligne 8 stade = "enfant" et sort de toutes les instructions conditionnelles pour reprendre à la ligne 12 et afficher le message Vous êtes enfant, vous avez 10 ans.

Noter qu'il faut bien prendre soin à l'indentation et que ce programme n'est pas très lisible!

Instructions conditionnelles en utilisant la structure if-elif-else

```
1  if age >= 18:
2    stade = "adulte"  # au dessus de 18
3  elif age >= 12:
4    stade = "ado"  # entre 12 et 18
5  elif age >= 2:
6    stade = "enfant"  # entre 2 et 12
7  else :
8    stade = "bébé"  # moins de 2
9
10  print(f"Vous êtes {stade}, vous avez {age} ans")
```

Dès que la première conditions if ou une condition elif est vérifiée, le programme ne teste plus les conditions elif suivantes ni le else final, mais reprend directement à l'instruction qui suit le bloc conditionnel, ici print(f"Vous êtes {stade}, vous avez {age} ans").

Par exemple, si on affecte la valeur 15 à la variable age, la première condition en ligne 1 if age >= 18: est fausse, le programme passe donc à la conditions suivante elif age >= 12:..., celle-ci est vérifiée, il exécute donc la ligne 4 stade = "ado" et n'a pas besoin de vérifier la condition suivante en ligne 5 elif age >= 2 :... ni d'exécuter le else:.... Dès que la condition elif age >= 12:... a été vérifiée, il sort de toute l'instruction conditionnelle pour reprendre à la ligne 10 et afficher le message Vous êtes ado, vous avez 15 ans.



(i) Rappel

Eviter les conditions d'égalité avec les nombres de type float. Par exemple :

```
if 2.3 - 0.3 == 2:
   print('Python sait bien calculer avec les float')
   print('Python ne sait pas bien calculer avec les float')
```

PEP 8

Eviter de comparer des variables booléennes à True ou False avec == ou avec is.

On écrit :

```
cond = True
if cond:
   print("la condition est vraie"
```

mais pas if cond == True: Ni if cond is True:

? Exercice corrigé

Écrire un programme qui demande une année et affiche si elle est bissextile ou pas :

- 1. en utilisant des conditions imbriquées.
- 2. en utilisant une structure if-elif-else.
- « Depuis l'ajustement du calendrier grégorien, l'année sera bissextile (elle aura 366 jours) seulement si elle respecte l'un des deux critères suivants :
 - 1. C1 : l'année est divisible par 4 sans être divisible par 100 (cas des années qui ne sont pas des multiples de 100) ;
 - 2. C2 : l'année est divisible par 400 (cas des années multiples de 100).

Si une année ne respecte ni le critère C1 ni le critère C2, l'année n'est pas bissextile ».

Source: https://fr.wikipedia.org/wiki/Année bissextile.

Réponse 1.en utilisant des conditions imbriquées

Analysons la définition donnée par Wikipedia sous la forme d'un arbre :

```
graph LR
A[annee%4 == 0] --> |True| B;
A -->|False| C{pas bissextile};
B[annee%100 != 0] --> |True| D{bissextile};
B --> |False| E;
E[annee%400 == 0] --> |True| F{bissextile};
E --> |False| G{pas bissextile};
```

Traduit en Python, on obtient le programme suivant.

```
annee = int(input('annee: ') )
1
2 if annee % 4 == 0:
                                        # si annee est divisible par 4 ...
       if annee % 100 != 0:
 3
                                           # ... mais pas par 100,
           print(annee, "est bissextile")
4
                                               # alors elle est bissextile
                                            # ... et par 100, alors
5
       else:
                                                # soit elle est divisible par 400 ...
 6
          if annee % 400 == 0:
               print(annee, "est bissextile")
7
                                                    # ...donc elle est bissextile
                                                # soit elle n'est pas divisible par 400
8
           else:
               print(annee, "n'est pas bissextile") # ...donc elle n'est pas bissextile
9
                                         # sinon, elle n'est divisible par 4 ...
10 else:
      print(annee, "n'est pas bissextile") # ...donc elle n'est pas bissextile
11
```

Le programme n'est pas facile à lire. On note ici l'importance de l'indentation!

✓ Réponse 2.en utilisant une structure if-elif-else

Chaque condition if, elif, else vérifiée termine la structure conditionnelle. Il faut donc tester les divisibilités en partant du plus grand nombre, 400, puis 100, puis 4 (car une fois qu'on a testé la divisibilité par 4, on ne peut plus tester la divisibilité par 100 ou 400). Modifions l'arbre précédent:

```
graph LR
A[annee%400 == 0] --> |True| C{bissextile};
A -->|False| B;
B[annee%100 == 0] --> |True| E;
B -->|False| D{pas bissextile};
E[annee%4 == 0] --> |True| F{bissextile};
E --> |False| G{pas bissextile};
```

Traduit en Python, on obtient le programme suivant.

```
annee = int(input('annee: ') )
if annee % 400 == 0:  # si annee est divisible par 400
print(annee, "est bissextile")
elif annee % 100 == 0:  # sinon, si annee est divisible par 100 (et pas par 400 car déjà testé)
print(annee, "n'est pas est bissextile")
elif annee % 4 == 0:  # sinon, si annee est divisible par 4 (et pas par 100 et 400 car déjà testés)
print(annee, "est bissextile")
else:  # sinon (l'annee n'est pas divisible par 4, 100 et 400 car déjà testés)
print(annee, "n'est pas est bissextile")
```

Le programme est beaucoup plus lisible.



(i) Note

il est bien sûr aussi possible d'utiliser l'expression trouvée dans l'exercice corrigé précédent :

```
annee = int(input('annee: ') )
if (annee % 4 == 0 and not annee % 100 == 0) or annee % 400 == 0:
   print(annee, 'est bissextile')
else:
    print(annee, "n'est pas bissextile")
```

mais ce n'est pas très lisible.

4.2. Boucles non bornées



- Cours

Une boucle non bornée (ou boucle conditionnelle) permet de répéter une instruction ou une séquence d'instructions, tant qu'une condition (une expression booléenne) est vraie.

```
while condition:
   instructions
```

La structure est similaire à l'instruction conditionnelle. La condition qui suit le mot while est une expression de type booléen qui prend la valeur True ou False. Le bloc d'instructions qui suit est exécuté tant que la condition est vraie. Ce bloc d'instruction doit impérativement modifier la valeur de la condition afin qu'elle finisse par ne plus être vérifiée, sinon la boucle est sans fin et le programme ne se terminera jamais!

Exemple:

Rappel

i += 2 est une abréviation de i = i + 2

```
>>> i = 0
>>> while i <= 10:
      print(i)
        i += 2
. . .
. . .
0
2
4
6
8
10
```

Il faut toujours impérativement vérifier que la condition ne sera plus vérifiée après un nombre fini de passage, sinon le programme ne s'arrête jamais, le programme boucle ou diverge. lci, c'est bien le cas grâce à l'instruction i += 2, i finira bien par être plus grand que 10.

Exemple de programme qui boucle (erreur d'indentation dans l'instruction i += 2):

```
1    i = 0
2    while i <= 10:
3         print(i)
4    i += 2</pre>
```

⚠ Comme pour les instructions conditionnelles, il faut faire particulièrement attention aux boucles avec les nombres de type float . La boucle suivante qui semble écrite correctement ne finira jamais :

```
i = 1
while i != 2:
    i += 0.1
    print(i)
```

Testons la même boucle écrite correctement :

```
i = 1
while i < 2:
    i += 0.1
    print(i)</pre>
```

Elle affiche dans la console :

i ne prend donc jamais la valeur 2.

4.3. Boucles bornées



Une boucle bornée (ou boucle non conditionnelle) permet de répéter n fois, n étant un nombre entier connu, une instruction ou une séquence d'instructions.

```
for i in range(n):
   instructions
```

i est appelé l'indice de boucle ou compteur de boucle, il prend les n valeurs entières comprises entre 0 et n - 1.

i ne prend pas la valeur n.

Il est aussi possible d'utiliser :

- range(d, f) qui énumère les f-d nombres entiers compris entre d et f-1. 23
- range(d, f, p) qui énumère les nombres entiers compris entre d et f-1 avec un pas de p : d, d+p, d+2p,
 etc. ²⁴

La boucle bornée ci-dessus est très similaire à la boucle non bornée suivante :

```
i = 0
while i < n :
    instructions
    i += 1</pre>
```

mais attention, comparons ces deux programmes :

Programme 1

```
for i in range(5):
    print(i, end=" ")
```

Programme 2

```
i = 0
while i < 5:
    print(i, end=" "))
    i = i + 1</pre>
```

Les deux programmes semblent afficher la même chose, tous les chiffres de 0 à 4 : 0 1 2 3 4

Pourtant ils sont différents. Ajoutons une instruction print(i) à la fin les deux programmes.

Programme 1

```
for i in range(5):
    print(i, end=" ")
print(i)
```

affiche: 0 1 2 3 4 4

La valeur finale de i est 4

Programme 2

```
i = 0
while i < 5:
    print(i, end=" ")
    i = i + 1
print(i)</pre>
```

affiche: 0 1 2 3 4 5 La valeur finale de i est 5 afin que la condition ne soit plus valide.

Dans le programme 1, la valeur finale de i est 4, alors que dans le programme 2 c'est 5 afin que la condition ne soit plus valide.

Autre différence, quand on modifie l'indice de boucle dans une boucle for , il reprend la valeur suivante à la prochaine répétition.

Programme 1

```
for i in range(5):
    print(i, end=" ")
    i = i + 2
    print(i, end=" ")
```

affiche: 0 2 1 3 2 4 3 5 4 6

La valeur de i est modifiée à l'intérieur de la boucle mais reprend la valeur suivante à la prochaine répétition.

Programme 2

```
i = 0
while i < 5:
    print(i, end=" ")
    i = i + 2
    print(i, end=" ")</pre>
```

affiche: 0 2 2 4 4 6

La valeur de i est modifiée à l'intérieur de la boucle.

A chaque passage dans une boucle for, l'indice de boucle repart de sa valeur au dernier passage dans la boucle, même si cette valeur a changé dans la boucle. En pratique, il n'est pas recommendé de changer sa valeur dans la boucle.

La boucle for possède d'autres possibilités très utiles, par exemple elle permet d'énumérer chaque caractère d'une chaine de caractères. Le programme ci-dessous affiche chaque lettre d'une variable message l'une après l'autre.

```
message = 'Hello world'
for c in message:
    print(c)
```

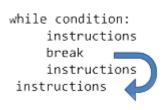
4.4. Instructions break et continue

Il existe deux instructions qui permettent de modifier l'exécution des boucles while et for, il s'agit de break et de continue.



- Cours

L'instruction break permet de sortir de la boucle courante et de passer à l'instruction suivante.



Par exemple, voici un programme qui redemande un mot de passe jusqu'à obtenir le bon :

```
1
    while True:
       mdp = input('mot de passe')
2
        if mdp == '123456':
3
4
           break
   print('trouvé')
```

Cours

L'instruction continue permet de sauter les instructions qui restent jusqu'à la fin de la boucle et de reprendre à la prochaine itération de boucle.

while condition: instructions continue | instructions instructions

Imaginons par exemple un programme qui affiche la valeur de 1/(n-7) pour tous les entiers n compris entre 1 et 10. Il est évident que quand n prend la valeur 7 il y aura une erreur. Grâce à l'instruction continue, il est possible de traiter cette valeur à part puis de continuer la boucle.

```
for n in range(10):
1
2
     if n == 7:
3
           continue
4
       print(1 / (7 - n))
```

4.5. Boucles imbriquées



- Cours

Il est possible d'imbriquer des boucles. A chaque passage dans la première boucle (la boucle externe), la seconde boucle (la boucle interne) est effectuée entièrement.



Attention à l'indentation pour déterminer à quelle boucle appartiennent les instructions.

Par exemple le programme suivant affiche toutes les heures, minutes et seconde de la journée (de 0h0min0s à 23h59min59s):

```
for heure in range(24):
1
2
        for minute in range(60):
3
            for seconde in range(60):
4
                print(f"{heure}h{minute}min{seconde}s")
```

? Exercice corrigé

Exercice corrigé : Écrire un programme en Python qui affiche tous les nombres premiers inférieurs à 100.

Rappel: un nombre est premier s'il n'a que deux diviseurs, 1 et lui-même.

Réponse



On utilise deux boucles for imbriquées :

• La première boucle parcourt tous les nombres n allant de 2 (0 et 1 ne sont pas premiers) à 100 pour vérifier s'ils sont premiers ou pas.

On peut se contenter de chercher un diviseur div seulement entre 2 et \sqrt{n} avec une boucle while div**2 <= n: OU while div <= n**0.5:.

• Pour chaque nombre n, la deuxième boucle vérifie s'il est divisible par un nombre div compris entre 2 et n - 1. Si n est divisible par div, alors il n'est pas premier et on affecte la valeur False à la variable est_premier. Dans ce cas, inutile de chercher d'autres diviseurs, on peut sortie de la boucle avec une instruction break.

Ensuite, à la fin de la deuxième boucle, on vérifie si la variable est_premier est True et dans ce cas, cela signifie que le n est premier et il est affiché à l'écran.

nombres_premiers.py

```
for n in range(2, 101):
      est_premier = True
       for div in range(2, n): #on cherche un diviseur compris entre 2 et n - 1
4
          if n % div == 0:
5
               est premier = False # on a trouvé un diviseur de n, il n'est pas premier
               break # on peut sortir de la boucle interne ici, inutile de continuer
6
7
       if est premier:
                              # on préfère à : if premier == True
8
           print(n, end="-")
```

- 1. https://fr.wikipedia.org/wiki/Liste_de_langages_de_programmation. ←
- 2. Nommé en hommage à la série britannique Monty Python Flying Circus. <
- 3. en 2023. ←
- 4. Le terme « informatique » résulte de l'association du terme « information » au suffixe « -ique » signifiant « qui est propre à ». ←
- 5. La notion de variable en informatique diffère des mathématiques. En mathématique une variable apparait dans l'expression symbolique d'une fonction f(x) = 2x + 3, ou dans une équation 2x + 3 = 5x - 3 pour désigner une inconnue qu'il faut trouver, ou encore dans une formule comme $(a + b)^2 = a^2 + 2ab + b^2$ pour indiquer que l'égalité est vraie pour toutes les valeurs de a et b. \leftarrow

- 6. Une PEP (pour *Python Enhancement Proposal*) est un document fournissant des informations à la communauté Python, ou décrivant une nouvelle fonctionnalité. En particulier la PEP 8 décrit les conventions de style de code agréable à lire.
- 7. Le style qui consiste à nommer les variables par des mots écritsen minuscule séparés par des blancs soulignés, par exemple somme_des_nombres, est appelé « snake case » en opposition au style qui consiste à écrire les mots attachés en commençant par des majuscules, par exemple SommeDesNombres, appelé « camel case ». ←
- 8. Les p_uplet, tableaux, dictionnaires sont étudiés dans un autre chapitre du programme de 1ère. 🖰
- 9. En algorithmique, l'affectation est symbolisée par une flèche allant de la valeur (à droite) vers le nom de la variable (à gauche), par exemple *a* ← 3 pour affecter la valeur 3 à la variable *a*. ←
- 10. Python est un langage de typage dynamique, ce n'est pas le cas de nombreux langages comme le C ou le C++ qui forcent à définir le type d'une variable et à le conserver au cours de la vie de la variable, ils sont de typage statique. Exemple d'affectation en C :

```
int a;
a = 3;
```

- 11. En mathématique $\sqrt{a} = a^{\frac{1}{2}}$.
- 12. Noter dans cet exemple la différence entre variable informatique et mathématique, et la signification du signe « = ». En mathématique a=2*a+1 est une équation dont l'inconnue est a (elle peut être facilement résolue pour trouver la solution a=-1). En informatique, c'est l'affection du résultat de a=10 a qui prend une nouvelle valeur (même si $a\neq -1$). \leftarrow
- 13. Vrai pour des entiers positifs. Attention aux surprises avec des nombres relatifs! Les résultats sont différents entre langages/systèmes informatiques. En Python on peut tester 7 // -5 et -17 // 5 qui donnent tous les deux -4 mais 17 % -5 donne -3 alors que -17 % 5 donne 3. ←
- 14. Pouvoir utiliser les apostrophes ou les guillemets offre un énorme avantage : les guillemets permettent d'écrire une chaîne qui contient des apostrophes et vis-versa, par exemple "J'aime Python" ou 'Il dit "hello".'. ←
- 15. Attention : les opérateurs + et * se comportent différemment s'il s'agit d'entiers ou de chaînes de caractères : 2 + 2 est une addition alors que '2' + '2' est une concaténation, 2 * 3 est une multiplication alors que '2' * 3 est une duplication. ←
- 16. « Hello world » (traduit littéralement en français par « Bonjour le monde ») sont les mots traditionnellement écrits par un programme informatique simple dont le but est de faire la démonstration rapide de son exécution sans erreur. Source : https://fr.wikipedia.org/wiki/Hello world ←
- 17. Une méthode est un type de fonction particulier propre aux langages orientés objet. Remarquer la construction nom_variable.nom_methode() dans ces cas différente de nom_fonction(nom_variable) par exemple len('abc'). \(\lefta \)
- 18. True et False (et None) sont les rares mots en Python qui s'écrivent avec une majuscule. TRUE ou true ne sont pas acceptés. ←
- 19. Préférer is et is not à == et != pour comparer à None, par exemple a is not None plutôt que a != None. ←
- 20. Les comparaisons entre chaînes de caractère se font en comparant le point de code Unicode de chaque caractère. Il est donné par la fonction ord() (la fonction chr() fait 'inverse). Par exemple, ord('A') vaut 65 et ord('a') vaut 97 donc 'A' < 'a' est vrai.

 'A' = 'a' est vrai.
- 21. Les nombres de type float sont encodés par des fractions binaires qui "approchent" leur valeur le plus précisément possible sans être toujours parfaitement exactes. Par exemple le nombre 0, 1 est représenté par la valeur 0.1000000000000000055511151231257827021181583404541015625 en Python (format(0.1,'.55f') permet d'afficher toutes les décimales). Une particularité de Python est de ne pas limiter l'encodage des int, par exemple comparer >>> 2*1000 avec >>> 2.**1000 dans la console.

Langages et programmation (1)

- 22. Nous n'abordons pas la notion de classe ici. 🖰
- 23. range(d, f) montre l'avantage d'exclure la borne supérieure, il y f-d nombres compris entre d (inclus) et f (exclus), comme l'a expliqué Edsger W. Dijkstra dans une note de 1982 ←
- 24. L'instruction range() fonctionne sur le modèle range([début,] fin [, pas]). Les arguments entre crochets sont optionnels. ←