Histoire de l'informatique

Fiche de synthèse

1 Des machines à calculer aux ordinateurs

Calculer avec des machines

1642 Pascal construit la **Pascaline**, qui additionne et soustrait avec des roues dentées.

1840 Babbage décrit sa **machine analytique** : entrée, mémoire, unité de calcul, commande et sortie, l'organisation des ordinateurs actuels.

1843 Ada Lovelace publie le premier **programme**, pour cette machine.

1854 Boole calcule sur des propositions vraies ou fausses : l'**algèbre de Boole**.

1884 Hollerith invente une machine à **cartes perforées** ; son entreprise deviendra IBM.

Les débuts de l'informatique moderne

1936 Turing définit l'**algorithme** avec sa machine abstraite, la **machine de Turing**.

1938 Shannon montre que les circuits à interrupteurs suivent l'algèbre de Boole.

1946 L'**ENIAC**, 30 tonnes et 18 000 tubes, se programme en rebranchant des câbles.

1947 Invention du **transistor** aux laboratoires Bell ; un insecte coincé dans le Mark II popularise le mot *bug*.

1950 Le **langage assembleur** remplace l'écriture directe en binaire.

2 Repères

— Le mot **algorithme** vient du nom du savant persan Al-Khwarizmi, au IXe siècle.

— Le **bit**, chiffre binaire, est le mot proposé par Tukey en 1947 et popularisé par Shannon en 1948.

— Du transistor au microprocesseur, la miniaturisation a rendu les ordinateurs plus petits, plus fiables et moins chers : l'Intel 4004 calculait aussi vite que l'ENIAC, qui remplissait une salle.

— Les processeurs **multicœurs**, à partir de 2005, font plusieurs calculs à la fois plutôt que d'augmenter une fréquence qui ferait trop chauffer la puce.

— Le **cloud** loue à la demande de la puissance de calcul et du stockage ; le **Raspberry Pi**, à 35 dollars, rend la programmation accessible.

3 Réseaux, grand public et intelligence artificielle

Les réseaux

1969 **ARPANET** relie ses premiers ordinateurs.

1983 ARPANET adopte **TCP/IP** : naissance d'Internet.

1989 Tim Berners-Lee invente le **Web** au CERN : URL, HTML et HTTP.

L'informatique grand public

1969 Thompson et Ritchie créent **Unix**, réécrit en C en 1973.

1971 Intel 4004, premier **microprocesseur** : 2 300 transistors sur une puce.

1981 L'**IBM PC** devient le standard des ordinateurs personnels.

1984 Le **Macintosh** popularise souris et interface graphique.

1991 **Unicode**, le noyau **Linux** et le langage **Python**.

1998 L'expression **open source** ; le logiciel libre existe depuis 1983.

2007 L'**iPhone** impose le smartphone tactile.

Le numérique et l'intelligence artificielle

2013 Snowden révèle la surveillance de masse.

2016 AlphaGo bat le champion du monde de go.

2017 L'architecture **Transformer** renouvelle le traitement du langage.

2018 Le **RGPD** encadre les données personnelles en Europe.

2022 ChatGPT fait connaître l'IA générative au grand public.

À retenir.

— L'organisation de la machine analytique de Babbage annonce celle des ordinateurs actuels.

— Turing définit l'algorithme en 1936 ; le transistor, en 1947, permet la miniaturisation.

— Internet naît de l'adoption de TCP/IP en 1983 ; le Web, de l'invention de Tim Berners-Lee en 1989.

Syntaxe

Fiche de synthèse

1 Variables et affectation

Définition 1 **(Variable)** Une **variable** est un nom qui désigne une donnée rangée en mémoire. On lui donne une valeur par une **affectation**, écrite avec le signe `=` : le nom à gauche, la valeur à droite.

```
age = 17
nom = "Alice"
taille = 1.65
```

L'affectation n'est pas une égalité mathématique : c'est une **instruction**, un ordre que Python exécute. Une **expression** est un calcul qui produit une valeur, comme `age + 1`. L'instruction `score = score + 10` calcule `score + 10`, puis donne ce résultat à `score`. Python exécute les lignes dans l'ordre ; la **trace d'exécution** note la valeur de chaque variable après chaque ligne, et une valeur déjà calculée ne se recalcule pas.

Choisir un nom de variable.

- Lettres, chiffres et tiret bas _ seulement, sans chiffre au début.
- Majuscules et minuscules comptent : `age` et `Age` sont deux variables différentes.
- Ni mot réservé (`if`, `for`, `def`…), ni nom de fonction comme `print`.
- Par convention, des mots en minuscules séparés par des tirets bas : `nombre_de_joueurs`.

2 Afficher : print

`print(...)` affiche une valeur : `print("Bonjour")`, `print(nom)`.

Pour mêler du texte et des valeurs, on écrit une **f-string** : la lettre `f` avant les guillemets, puis chaque variable ou calcul entre accolades.

```
prenom = "Alice"
age = 17
print(f"{prenom} a {age} ans")
print(f"Dans 10 ans : {age + 10} ans")
```

Ces lignes affichent `Alice a 17 ans`, puis `Dans 10 ans : 27 ans`.

3 Saisir : input

`input("question")` affiche la question et renvoie ce que tape l'utilisateur, **toujours** sous forme de chaîne de caractères. Pour calculer, on convertit avec `int` (entier) ou `float` (nombre à virgule).

```
nom = input("Quel est votre nom ? ")
age = int(input("Quel est votre âge ? "))
print(f"Bonjour {nom}, vous avez {age} ans")
```

Attention. Sans conversion, `input("Âge ? ") + 1` provoque l'erreur `TypeError` : Python n'additionne pas une chaîne et un nombre.

4 Types

Chaque donnée a un **type**, que donne la fonction `type` :

Valeur	Type	Nom
`42`	`int`	entier
`3.14`	`float`	flottant
`"Bonjour"`	`str`	chaîne de caractères
`True`	`bool`	booléen

C'est la donnée qui a un type, pas la variable : après `x = 42` puis `x = "quarante-deux"`, `x` désigne une chaîne. Attention, `42` est un entier, mais `"42"` est une chaîne.

5 Commentaires

Tout ce qui suit le caractère `#` sur une ligne est un **commentaire**, ignoré par Python : il explique le code à qui le lit.

6 Échanger deux variables

```
a = 5
b = 3
temp = a    # sauvegarde de a
a = b
b = temp
```

Écrire `a = b` puis `b = a` perdrait la valeur 5. Python permet aussi l'**affectation multiple** : `a, b = b, a`.

À retenir.

- `variable = valeur` affecte une valeur à une variable.
- `print(f"... {variable} ...")` affiche du texte et des valeurs.
- `input` renvoie toujours une chaîne : on la convertit avec `int` ou `float`.
- Les quatre types de base : `int`, `float`, `str`, `bool`.

Types de base

Fiche de synthèse

1 Le type d'une valeur

Le **type** d'une valeur détermine les opérations permises : on ne multiplie pas un mot par un mot, on ne met pas un nombre en majuscules. La fonction `type` donne le type d'une valeur ; `print(type(42))` affiche `<class 'int'>`.

Type	Nom	Exemples	Conversion
entier	`int`	`42`, `-7`	`int()`
flottant	`float`	`3.14`, `7.0`	`float()`
chaîne	`str`	`'Bonjour'`, `"42"`	`str()`
booléen	`bool`	`True`, `False`	`bool()`

C'est la donnée qui est typée, pas la variable : une même variable peut désigner successivement un entier puis une chaîne.

2 Entiers

Un **entier** est un nombre sans partie décimale. En Python, sa taille est illimitée : `2 ** 100` se calcule exactement.

```
print(10 / 2)     # 5.0 : toujours un flottant
print(10 // 2)    # 5 : division entière
print(7 // 2)     # 3 : quotient de 7 par 2
```

3 Flottants

Un **flottant** est un nombre à virgule, écrit avec un **point** : `3.14`. Le point suffit : `3.0` est un flottant. Un calcul qui mêle un entier et un flottant donne un flottant : `3 + 4.0` vaut `7.0`.

Les flottants sont rangés sur 64 bits, avec environ quinze chiffres significatifs : leurs valeurs sont souvent approchées.

```
print(0.1 + 0.2)   # 0.30000000000000004
print(0.1 + 0.2 == 0.3)   # False
print(abs(0.1 + 0.2 - 0.3) < 1e-10)  # True
```

On ne compare jamais deux flottants avec `==` : on vérifie que leur écart est très petit. `abs(x)` est la valeur absolue de `x`, et `1e-10` s'écrit en notation scientifique pour 10^{-10}.

4 Chaînes de caractères

Une **chaîne de caractères** s'écrit entre apostrophes ou entre guillemets : `"L'informatique"` contient une apostrophe, on l'entoure donc de guillemets.

```
mot = "Python"
print(len(mot))            # 6 caractères
print("Ada" + " " + "Lovelace")   # assemblage
print("ha" * 3)            # hahaha
print(42 + 1)              # 43
print("42" + "1")          # 421
```

Assembler deux chaînes s'appelle la **concaténation**. On ne peut pas additionner une chaîne et un nombre : `"J'ai " + 17` provoque une erreur `TypeError`. On écrit `"J'ai " + str(17)`, ou plus simplement la f-string `f"J'ai {age} ans"`.

5 Booléens et None

Un **booléen** vaut `True` ou `False`, avec une majuscule. C'est le résultat d'une comparaison : `5 > 3` vaut `True`.

`None` représente l'absence de valeur ; son type est `NoneType`.

`0`, `0.0`, la chaîne vide `''` et `None` sont considérés comme faux par `bool` ; toute autre valeur est considérée comme vraie, même la chaîne `"False"`, qui n'est pas vide.

6 Conversions

```
print(int("15"))       # 15
print(float("3.14"))   # 3.14
print(str(42))         # la chaîne '42'
print(int(3.7))        # 3
print(int(-2.9))       # -2
print(round(3.7))      # 4 : arrondi
print(round(3.14159, 2))   # 3.14
```

`int("abc")` et `int("3.14")` provoquent une erreur `ValueError`.

Calculer avec une saisie. `input` renvoie toujours une chaîne : on la convertit dès la saisie.

```
age = int(input("Ton âge ? "))
print(f"Dans 10 ans : {age + 10}")
```

À retenir.

- Quatre types de base : `int`, `float`, `str`, `bool`.
- `/` donne toujours un flottant, `//` la division entière.
- On ne compare jamais deux flottants avec `==`.
- Les guillemets font la chaîne : `"42" + "1"` vaut `"421"`.
- `int` supprime la partie décimale sans arrondir ; `round` arrondit.

Opérateurs

Fiche de synthèse

1 Opérateurs arithmétiques

Opérateur	Nom	Exemple	Résultat
`+`, `-`, `*`	addition, soustraction, multiplication	`7 * 3`	`21`
`/`	division	`7 / 2`	`3.5`
`//`	division entière	`7 // 3`	`2`
`%`	modulo, le reste	`7 % 3`	`1`
`**`	puissance	`2 ** 10`	`1024`

La division `/` donne toujours un flottant : `10 / 2` vaut `5.0`.

Définition 1 **(Quotient et reste)** Avec 17 billes rangées par paquets de 5, on forme 3 paquets et il reste 2 billes : $17 = 5 \times 3 + 2$. Le **quotient** 3 s'obtient par `17 // 5`, le **reste** 2 par `17 % 5`.

```
n = 42
print(n % 2 == 0)      # True : 42 est pair
print(2025 % 10)       # 5 : dernier chiffre
duree = 195
minutes = duree // 60      # 3
secondes = duree % 60      # 15
```

2 Priorité des opérations

Python suit l'ordre des mathématiques : parenthèses, puissance, signe moins, puis `*`, `/`, `//`, `%`, enfin `+` et `-`. `2 + 3 * 4` vaut `14` et `(2 + 3) * 4` vaut `20`. Plusieurs puissances se calculent de droite à gauche : `2 ** 3 ** 2` vaut $2^9 = 512$. En cas de doute, on ajoute des parenthèses.

3 Comparaisons

Opérateur	Signification	Exemple	Résultat
`==`	égal à	`5 == 5`	`True`
`!=`	différent de	`5 != 3`	`True`
`<`, `>`	strictement inférieur, supérieur	`3 > 5`	`False`
`<=`, `>=`	inférieur ou égal, supérieur ou égal	`5 <= 5`	`True`

Une comparaison donne un booléen. Un seul `=` affecte une valeur, deux `==` testent une égalité. Deux chaînes se comparent dans l'ordre alphabétique, caractère par caractère : `"abc" < "abd"` vaut `True`.

4 Opérateurs logiques

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

`and` est vrai si les deux conditions sont vraies, `or` si au moins une l'est, et `not` inverse une condition. Ce tableau est une **table de vérité**.

Python s'arrête dès que le résultat est connu : avec `and`, si la première condition est fausse, il ne calcule pas la seconde ; avec `or`, si la première est vraie. Ainsi `x == 0 or 1 / x > 2` ne provoque pas d'erreur quand `x` vaut 0.

```
annee = 2024
bissextile = (annee % 4 == 0) and (
    annee % 100 != 0 or annee % 400 == 0)
print(f"{annee} est bissextile : {bissextile}")
```

Une année est bissextile si elle est divisible par 4, sauf si elle l'est par 100 sans l'être par 400.

5 Affectation augmentée

`score += 10` est l'écriture courte de `score = score + 10`. De même pour `-=`, `*=`, `/=`, `//=`, `%=` et `**=`.

6 Appartenance : in

`in` teste si un caractère, ou une suite de caractères, figure dans une chaîne : `'a' in "aeiouy"` vaut `True`, `'th' in "Python"` aussi ; `not in` teste l'absence.

7 Modules math et random

```
import math
import random

print(math.sqrt(16))       # 4.0
print(math.floor(3.7))     # 3
de = random.randint(1, 6)   # entre 1 et 6
x = random.random()      # 0 <= x < 1
```

À retenir.

- `17 // 5` vaut 3, le quotient ; `17 % 5` vaut 2, le reste.
- `n % 2 == 0` teste si `n` est pair.
- Une comparaison donne un booléen ; `=` affecte, `==` compare.

Conditions

Fiche de synthèse

1 Blocs et indentation

Définition 1 **(Bloc d'instructions)** Un **bloc** est un groupe d'instructions qui vont ensemble. En Python, il est délimité par l'**indentation** : quatre espaces au début de chaque ligne du bloc.

Toutes les lignes d'un bloc ont exactement la même indentation ; sinon, Python signale une erreur `IndentationError`. La ligne qui ouvre le bloc se termine par deux-points.

2 if, else, elif

```
temperature = 35
if temperature > 30:
    print("Il fait très chaud !")
print("Bonne journée")
```

Le bloc sous `if` ne s'exécute que si la condition vaut `True` ; la dernière ligne, non indentée, s'exécute toujours. La condition est une expression booléenne, écrite avec des comparaisons (`==`, `!=`, `<`, `>`, `<=`, `>=`) et des opérateurs logiques.

```
if age >= 18:
    print("Vous êtes majeur")
else:
    print("Vous êtes mineur")
```

Avec `else`, sinon, exactement un des deux blocs s'exécute.

```
note = 14
if note >= 16:
    mention = "Très bien"
elif note >= 14:
    mention = "Bien"
elif note >= 10:
    mention = "Admis"
else:
    mention = "Ajourné"
```

`elif`, contraction de « sinon si », ajoute des cas. Python teste les conditions de haut en bas ; à la première vraie, il exécute son bloc et ignore tout le reste. Ici, `mention` vaut `"Bien"`. On commence donc par la condition la plus exigeante : si l'on testait `note >= 10` en premier, une note de 18 donnerait `"Admis"`.

3 Plusieurs if ou elif

— Plusieurs `if` à la suite sont testés chacun, indépendamment : pour `n = 12`, `if n % 2 == 0` et `if n % 3 == 0` affichent tous deux leur message.

— Avec `elif`, un seul cas est choisi : seul le premier message s'affiche.

4 Combiner et imbriquer

```
if age >= 18 and permis:
    print("Vous pouvez conduire")
if jour == "samedi" or jour == "dimanche":
    print("C'est le week-end")
```

On peut aussi placer un `if` dans un autre : chaque niveau ajoute quatre espaces. L'imbrication traduit un **arbre de décision** ; elle est utile quand chaque issue a son propre message.

```
if age >= 18:
    if permis:
        print("Vous pouvez conduire")
    else:
        print("Passez d'abord le permis")
else:
    print("Vous êtes trop jeune")
```

5 Expression conditionnelle et pass

`statut = "majeur" if age >= 18 else "mineur"` choisit une valeur en une ligne : on la réserve aux cas simples. Le mot-clé `pass` est un bloc vide volontaire, car un bloc ne peut pas rester vide.

Attention. Erreurs fréquentes : oublier les deux-points à la fin de la ligne `if` ; écrire `if x = 5` au lieu de `if x == 5` ; aligner mal les lignes d'un bloc ; écrire plusieurs `if` quand un seul cas doit être choisi.

À retenir.

— L'indentation, quatre espaces, délimite les blocs.

— `if` exécute un bloc si la condition est vraie ; `else` traite l'autre cas ; `elif` ajoute des cas, testés dans l'ordre.

— Avec `elif`, seul le premier cas vrai s'exécute ; plusieurs `if` sont tous testés.

Boucles

Fiche de synthèse

1 Deux boucles

Une **boucle** répète un bloc d'instructions indenté, sans le recopier.

— La boucle `for` parcourt une séquence : on l'utilise quand on connaît le nombre de tours.

— La boucle `while` répète tant qu'une condition est vraie : on l'utilise quand on ne connaît pas le nombre de tours.

2 La boucle for

```
for lettre in "NSI":
    print(lettre)      # N, puis S, puis I

for i in range(5):
    print(i)           # 0, 1, 2, 3, 4
```

À chaque tour, la variable de boucle prend la valeur suivante de la séquence : les caractères d'une chaîne, ou les entiers fournis par `range`.

Appel	Entiers obtenus
`range(5)`	0, 1, 2, 3, 4
`range(2, 7)`	2, 3, 4, 5, 6
`range(0, 20, 5)`	0, 5, 10, 15

Dans `range(début, fin, pas)`, la fin est toujours **exclue** : pour aller de 1 à 5, on écrit `range(1, 6)`. Le **pas** est l'écart entre deux valeurs.

3 Accumuler un résultat

```
total = 0
for i in range(1, 6):
    total += i
print(f"La somme vaut {total}")   # 15
```

i	1	2	3	4	5
total	1	3	6	10	15

La variable `total` est un **accumulateur** : elle reçoit sa valeur de départ **avant** la boucle, puis change à chaque tour. Ce tableau, qui suit la valeur des variables tour après tour, est une **trace d'exécution**. Le même schéma construit une chaîne en partant de la chaîne vide `""`.

4 La boucle while

```
compteur = 1
while compteur <= 5:
    print(compteur)
    compteur += 1
```

La condition est testée avant chaque tour : ici six fois, la dernière donnant `False`. Le corps de la boucle doit faire évoluer la variable testée ; sinon la condition reste vraie et la boucle ne s'arrête jamais : c'est une **boucle infinie**, qu'on interrompt avec Ctrl+C.

```
mot_de_passe = ""
while mot_de_passe != "nsi2026":
    mot_de_passe = input("Mot de passe : ")
print("Accès autorisé !")
```

Personne ne sait combien d'essais il faudra : c'est un cas pour `while`.

5 break et continue

— `break` sort immédiatement de la boucle : on s'arrête, par exemple, à la première voyelle trouvée.

— `continue` saute la fin du bloc et passe au tour suivant.

— `while True:` suivi d'un `break` exécute le bloc au moins une fois, par exemple pour redemander une saisie incorrecte.

6 Boucles imbriquées

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} × {j} = {i * j}")
```

La boucle intérieure s'exécute entièrement à chaque tour de la boucle extérieure : $3 \times 3 = 9$ produits.

Attention. Erreurs fréquentes : écrire `range(1, 5)` pour aller jusqu'à 5 ; donner sa valeur de départ à l'accumulateur dans la boucle ; oublier de faire évoluer la condition d'un `while` ; croire que modifier `i` dans un `for` change le parcours.

À retenir.

— `for` : nombre de tours connu ; `while` : tant qu'une condition est vraie.

— `range(n)` donne $0, 1, \ldots, n-1$; la fin est toujours exclue.

— Un accumulateur reçoit sa valeur de départ avant la boucle.

Chaînes de caractères

Fiche de synthèse

1 Définition

Définition 1 **(Chaîne de caractères)** Une **chaîne de caractères**, de type `str`, est une séquence de caractères : un prénom, une phrase, un mot de passe. Elle s'écrit entre guillemets `"` ou entre apostrophes `'`.

On entoure de guillemets une chaîne qui contient une apostrophe : `"L'informatique"`. Le caractère `\` permet d'écrire un guillemet dans une chaîne entourée de guillemets : `"Elle a dit \"oui\""`.

`len(s)` donne le nombre de caractères de `s`, espaces compris : `len("Bonjour !")` vaut 9 et `len("")` vaut 0.

2 Indices

Chaque caractère a un **indice**, à partir de **zéro**. Les indices négatifs comptent depuis la fin : `-1` désigne le dernier caractère.

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

```
mot = "Python"
print(mot[0])     # P
print(mot[3])     # h
print(mot[-1])    # n
print(mot[6])     # IndexError
```

Les indices valides vont de `0` à `len(s) - 1` ; au-delà, Python signale une erreur `IndexError`.

3 Une chaîne ne se modifie pas

Une chaîne est **immuable** : `mot[0] = "J"` provoque l'erreur `TypeError`. Pour obtenir une chaîne modifiée, on en construit une nouvelle.

```
mot = "Python"
nouveau = "J"
for i in range(1, len(mot)):
    nouveau = nouveau + mot[i]
print(nouveau)    # Jython
```

4 Assembler et répéter

```
complet = "Ada" + " " + "Lovelace"
tirets = "-" * 20
print("ha" * 3 + "!")    # hahaha!
```

`+` **concatène**, c'est-à-dire assemble, deux chaînes ; `*` répète une chaîne. Pour ajouter un nombre, on le convertit d'abord : `"J'ai " + str(17) + " ans"`.

5 f-strings

```
print(f"{prenom} a {age} ans")
print(f"Pi vaut environ {math.pi:.2f}")   # 3.14
print(f"Réussite : {0.85:.0%}")           # 85%
```

Entre les accolades, on écrit une variable ou un calcul ; après deux-points, un format : `.2f` pour deux chiffres après la virgule, `.0%` pour un pourcentage.

6 Caractères spéciaux

Écriture	Caractère
`\n`	retour à la ligne
`\t`	tabulation
`\\`	le caractère \
`\'`, `\"`	apostrophe, guillemet

Une chaîne entourée de triples guillemets `"""` peut s'écrire sur plusieurs lignes.

7 Chercher un texte : in

`"NSI" in "Bienvenue en NSI"` vaut `True` ; `"nsi" in "Bienvenue en NSI"` vaut `False`, car majuscules et minuscules sont distinguées. `not in` teste l'absence.

À retenir.

- Les indices commencent à 0 ; `s[-1]` est le dernier caractère.
- Les indices valides vont de `0` à `len(s) - 1`.
- Une chaîne est immuable : on en construit une nouvelle.
- `+` assemble, `*` répète, `in` cherche.

Chaînes : méthodes

Fiche de synthèse

① Parcourir une chaîne

```
mot = "programmation"
compteur = 0
for lettre in mot:
    if lettre in "aeiouy":
        compteur += 1
print(f"{compteur} voyelles")    # 5 voyelles
```

La boucle `for` parcourt les caractères dans l'ordre. Pour obtenir à la fois l'indice et le caractère, on utilise `enumerate` :

```
for i, lettre in enumerate("NSI"):
    print(f"Indice {i} : {lettre}")
```

② Méthodes

Définition 1 **(Méthode)** Une **méthode** est une fonction qu'on applique à une valeur en écrivant un point après elle : `mot.upper()`. Une chaîne étant immuable, ses méthodes **renvoient une nouvelle chaîne** et laissent l'originale intacte.

```
mot = "Bonjour"
print(mot.upper())    # BONJOUR
print(mot)            # Bonjour : inchangé
```

Méthode	Résultat
`s.upper()`	tout en majuscules
`s.lower()`	tout en minuscules
`s.capitalize()`	première lettre en majuscule
`s.strip()`	sans espaces ni retours à la ligne aux deux bouts
`s.find(x)`	indice de la première occurrence de `x`, ou -1
`s.count(x)`	nombre d'occurrences de `x`
`s.replace(a, b)`	chaque `a` remplacé par `b`
`s.isdigit()`	`True` si `s` n'a que des chiffres
`s.isalpha()`	`True` si `s` n'a que des lettres

`lstrip` et `rstrip` ne nettoient que la gauche ou la droite ; `isalnum`, `isupper` et `islower` testent lettres et chiffres, majuscules, minuscules.

③ Exemples

```
phrase = "Python est un langage puissant"
print(phrase.count("an"))        # 2
print(phrase.find("langage"))    # 14
print(phrase.find("Java"))       # -1
```

Comparer sans tenir compte des majuscules.

```
saisie = "OUI"
if saisie.lower() == "oui":
    print("Confirmé")
```

Vérifier une saisie avant de la convertir.

```
saisie = input("Entrez un nombre : ")
if saisie.isdigit():
    nombre = int(saisie)
    print(f"Le double est {nombre * 2}")
else:
    print("Ce n'est pas un nombre valide")
```

À retenir.

- `for lettre in s` parcourt les caractères ; `enumerate` donne aussi les indices.
- Une méthode s'écrit `s.methode()` et renvoie une nouvelle chaîne : `s` ne change pas.
- `find` renvoie -1 si le texte est absent ; `replace` remplace toutes les occurrences.
- `strip` nettoie une saisie ; `isdigit` vérifie qu'elle ne contient que des chiffres.

Fonctions

Fiche de synthèse

1 Définir et appeler

Définition 1 **(Fonction)** Une **fonction** regroupe des instructions sous un nom. On la **définit** une fois avec `def`, puis on l'**appelle** autant de fois qu'on veut en écrivant son nom suivi de parenthèses.

Les fonctions évitent de recopier du code et découpent un problème en sous-problèmes. La définition n'exécute rien : le corps de la fonction ne s'exécute qu'à l'appel.

```
def saluer(prenom):
    print(f"Bonjour {prenom} !")

saluer("Alice")     # Bonjour Alice !
saluer("Bob")       # Bonjour Bob !
```

Les **paramètres** sont les noms écrits dans la définition, ici `prenom` ; les **arguments** sont les valeurs données à l'appel, ici `"Alice"`. Plusieurs paramètres se séparent par des virgules.

2 Renvoyer une valeur : return

```
def carre(n):
    return n ** 2

resultat = carre(5)        # 25
print(carre(3) + 1)        # 10

def valeur_absolue(n):
    if n >= 0:
        return n
    return -n
```

- `return` renvoie une valeur et **termine** la fonction : la suite du corps est ignorée.
- Une fonction sans `return` renvoie la valeur `None`.
- Une fonction qui **affiche** avec `print` ne renvoie rien d'utilisable ; une fonction qui **renvoie** une valeur permet de la réutiliser dans un calcul. On préfère presque toujours renvoyer.

3 Valeurs par défaut et arguments nommés

```
def saluer(prenom, formule="Bonjour"):
    print(f"{formule} {prenom} !")

saluer("Alice")                 # Bonjour Alice !
saluer("Alice", "Bonsoir")      # Bonsoir Alice !
saluer(formule="Salut", prenom="Bob")
```

Un paramètre qui a une **valeur par défaut** peut être omis ; il se place après les paramètres obligatoires. À l'appel, un **argument nommé** précise le paramètre et permet de changer l'ordre.

4 Portée des variables

- Une variable créée dans une fonction est **locale** : elle n'existe que pendant l'appel. L'utiliser ensuite provoque l'erreur `NameError`.
- Une variable créée hors de toute fonction est **globale** : une fonction peut la lire.
- Une variable locale qui porte le nom d'une variable globale la masque sans la modifier.

```
x = 100
def modifier():
    x = 0           # x locale
modifier()
print(x)            # 100 : x global inchangé
```

5 Documenter et tester

```
def moyenne(a: float, b: float) -> float:
    """Renvoie la moyenne de a et de b.

    >>> moyenne(12, 15)
    13.5
    """
    return (a + b) / 2

assert moyenne(12, 15) == 13.5
assert moyenne(0, 0) == 0
```

- La **docstring**, chaîne placée juste après la ligne `def`, décrit la fonction ; `help(moyenne)` l'affiche.
- Les **annotations** `a: float` et `-> float` indiquent les types attendus ; Python ne les vérifie pas.
- `assert condition` ne fait rien si la condition est vraie, et arrête le programme sur une erreur `AssertionError` sinon. On teste ainsi la fonction sur des exemples dont on connaît le résultat.

À retenir.

- `def` définit une fonction, l'appel l'exécute.
- `return` renvoie une valeur et termine la fonction ; sans `return`, la fonction renvoie `None`.
- Les variables créées dans une fonction sont locales.
- On documente avec une docstring et on teste avec `assert`.

Listes

Fiche de synthèse

1 Définition

Définition 1 **(Liste)** Une **liste** regroupe plusieurs valeurs dans une seule variable, entre crochets et séparées par des virgules. Ses éléments sont ordonnés : chacun a un **indice**. Une liste est **mutable** : on peut ajouter, supprimer ou remplacer des éléments.

```
notes = [15, 12, 18, 14, 16]
panier = []                  # liste vide
lettres = list("NSI")        # ['N', 'S', 'I']
```

2 Indices

```
print(notes[0])      # 15, premier élément
print(notes[-1])     # 16, dernier élément
print(len(notes))    # 5 éléments
notes[1] = 14        # remplace le 2e élément
```

Comme pour une chaîne, les indices commencent à 0 et vont jusqu'à `len(L) - 1` ; `notes[5]` provoque l'erreur `IndexError`. Contrairement à une chaîne, on peut affecter une nouvelle valeur à un indice.

3 Méthodes

Méthode	Effet
`L.append(x)`	ajoute `x` à la fin
`L.insert(i, x)`	insère `x` à l'indice `i`
`L.pop()`	retire et renvoie le dernier élément
`L.pop(i)`	retire et renvoie l'élément d'indice `i`
`L.remove(x)`	retire la première occurrence de `x`
`L.sort()`	trie la liste
`L.reverse()`	inverse l'ordre
`L.index(x)`	indice de la première occurrence de `x`
`L.count(x)`	nombre d'occurrences de `x`

⚠ **Attention.** `append`, `sort`, `reverse` et `remove` modifient la liste **en place** et renvoient `None` : on n'écrit jamais `L = L.sort()`.

4 Parcourir une liste

```
for note in notes:                  # les valeurs
    print(note)

for i in range(len(notes)):         # les indices
    notes[i] = notes[i] + 1

for i, note in enumerate(notes):    # les deux
    print(f"Note {i} : {note}")
```

— Le parcours par les **valeurs** suffit pour afficher ou additionner les éléments.

— Le parcours par les **indices** donne la position : il permet de modifier la liste. Écrire `note = note + 1` dans une boucle sur les valeurs ne change pas la liste.

— On ne modifie pas la longueur d'une liste pendant qu'on la parcourt.

5 Construire une liste

```
admis = []
for note in [15, 8, 12, 6, 18]:
    if note >= 10:
        admis.append(note)
print(admis)        # [15, 12, 18]
```

On part d'une liste vide et on ajoute un élément à chaque tour avec `append`, éventuellement sous condition, pour **filtrer**.

Une **liste en compréhension** fait la même chose en une ligne : l'élément à ajouter, puis la boucle, puis la condition éventuelle.

```
carres = [x ** 2 for x in range(5)]
# [0, 1, 4, 9, 16]
pairs = [x for x in range(10) if x % 2 == 0]
```

6 Opérations et fonctions

— `[1, 2] + [3]` vaut `[1, 2, 3]` ; `[0] * 3` vaut `[0, 0, 0]` ; `x in L` teste la présence de `x`.

— `len`, `min`, `max` et `sum` donnent la longueur, le minimum, le maximum et la somme : la moyenne est `sum(L) / len(L)`.

7 Chaînes et listes : split et join

```
"28-03-2026".split("-")
# ['28', '03', '2026']
"  Python   est   génial ".split()
```

```
# ['Python', 'est', 'génial']
" ".join(["Python", "est", "génial"])
# 'Python est génial'
```

`split` découpe une chaîne en liste, sur le séparateur donné ou, sans argument, sur les espaces. `join` s'écrit sur le séparateur et assemble une liste de chaînes.

⑧ Copie et effets de bord

```
a = [1, 2, 3]
b = a            # b est un autre nom pour a
b[0] = 99
print(a)         # [99, 2, 3]
c = a.copy()     # vraie copie, comme list(a)
```

— `b = a` ne copie pas la liste : `b` est un **alias**, un autre nom de la même liste. `a.copy()` ou `list(a)` créent une nouvelle liste.

— Une fonction qui reçoit une liste reçoit un alias : si elle la modifie, avec `L.append(0)` ou `L[i] = ...`, la modification est visible après l'appel. C'est un **effet de bord**. Réaffecter le paramètre, `L = []`, ne touche pas la liste de l'appelant.

— Une fonction modifie la liste en place et ne renvoie rien, comme `L.sort()`, ou renvoie une nouvelle liste sans toucher à son argument, comme `sorted(L)` : sa docstring le précise.

⑨ Listes de listes

```
matrice = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]
print(matrice[1])       # [4, 5, 6] : ligne 1
print(matrice[1][2])    # 6 : ligne 1, colonne 2
```

`M[i][j]` est l'élément de la ligne `i` et de la colonne `j`. On parcourt tous les éléments avec deux boucles imbriquées. `a.copy()` ne copie pas les listes intérieures : `deepcopy(a)`, du module `copy`, les copie aussi.

À retenir.

— Indices de `0` à `len(L) - 1` ; `L[-1]` est le dernier élément.

— `append`, `insert`, `pop`, `remove` modifient la liste.

— `for x in L` parcourt les valeurs, `for i in range(len(L))` les indices.

— Liste vide, boucle et `append` : la façon courante de construire une liste.

— `b = a` crée un alias ; `a.copy()` crée une copie.

Tuples

Fiche de synthèse

1 Définition

Définition 1 **(Tuple)** Un **tuple** regroupe plusieurs valeurs, écrites entre parenthèses et séparées par des virgules. Comme une liste, il est ordonné ; contrairement à une liste, il est **immuable** : on ne peut ni le modifier, ni lui ajouter ou retirer une valeur.

```
point = (3, 5)
date = (14, "juillet", 1789)
singleton = (5,)        # un seul élément
vide = ()
```

C'est la **virgule** qui fait le tuple : `(5)` est l'entier 5, `(5,)` est un tuple. On écrit tout de même les parenthèses, pour la lisibilité.

2 Lire, ne pas modifier

```
print(point[0])     # 3
print(date[-1])     # 1789
print(len(date))    # 3
print(5 in point)   # True
point[0] = 4        # TypeError
```

On lit une valeur par son indice, à partir de 0, comme dans une liste. Pour « modifier » un tuple, on en construit un nouveau : `(1, 2) + (3,)` vaut `(1, 2, 3)`. `tuple(L)` transforme une liste en tuple, `list(t)` un tuple en liste.

3 Le déballage

Définition 2 **(Déballage)** Le **déballage** range chaque valeur d'un tuple dans une variable, en une seule affectation : il faut autant de variables que de valeurs.

```
x, y = point        # x vaut 3, y vaut 5
a, b = b, a         # échange de a et de b
```

L'échange `a, b = b, a` remplace les trois affectations avec une variable auxiliaire : Python construit d'abord le tuple `(b, a)`, puis le déballe.

4 Renvoyer plusieurs valeurs

```
def min_max(L):
    mini = L[0]
    maxi = L[0]
    for x in L:
        if x < mini:
            mini = x
        if x > maxi:
            maxi = x
    return mini, maxi

petit, grand = min_max([4, 9, 1, 7])   # 1 et 9
```

`return mini, maxi` renvoie un tuple, que l'appel déballe dans deux variables.

5 Parcourir

```
for jour in ("lundi", "mardi", "mercredi"):
    print(f"Cours le {jour}")

eleves = [("Alice", 15), ("Bob", 12)]
for prenom, note in eleves:
    print(f"{prenom} : {note}")
```

On parcourt un tuple comme une liste. Une liste de tuples se parcourt avec le déballage : chaque tour range le prénom et la note dans deux variables.

6 Tuple ou liste ?

	Liste	Tuple
Écriture	`[1, 2, 3]`	`(1, 2, 3)`
Modifiable	oui, mutable	non, immuable
Usage	collection qui évolue	données fixes : coordonnées, date

Remarque 1. *Pour aller plus loin.* Un tuple nommé, créé avec `namedtuple` du module `collections`, permet d'écrire `p.x` au lieu de `p[0]`.

À retenir.

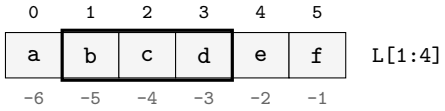
- Un tuple s'écrit entre parenthèses et ne se modifie pas.
- La virgule fait le tuple : `(5,)`.
- `x, y = point` déballe un tuple ; `a, b = b, a` échange deux variables.
- `return mini, maxi` renvoie plusieurs valeurs.

Tranches

Fiche de synthèse

① Extraire une tranche

Définition 1 **(Tranche)** Une **tranche** `L[début:fin]` est la partie d'une liste, d'une chaîne de caractères ou d'un tuple qui va de l'indice `début`, **inclus**, à l'indice `fin`, **exclu**. En anglais, on parle de *slicing*.




```
L = ['a', 'b', 'c', 'd', 'e', 'f']
print(L[1:4])     # ['b', 'c', 'd']
print(L[:3])      # ['a', 'b', 'c']
print(L[3:])      # ['d', 'e', 'f']
```

`L[1:4]` prend les indices 1, 2 et 3 : elle contient $4 - 1 = 3$ éléments. Sans début, la tranche part du premier élément ; sans fin, elle va jusqu'au dernier.

② Copier une liste

`L[:]`, sans début ni fin, est une nouvelle liste qui contient tous les éléments : c'est une copie. Modifier la copie ne change pas `L`.

③ Indices négatifs

```
print(L[-3:])     # ['d', 'e', 'f'] : 3 derniers
print(L[:-1])     # tous sauf le dernier
```

④ Le pas

```
print(L[::2])     # ['a', 'c', 'e']
print(L[1::2])    # ['b', 'd', 'f']
print(L[::-1])    # de 'f' à 'a'
```

Dans `L[début:fin:pas]`, le **pas** est l'écart entre deux indices pris. Un pas négatif parcourt la séquence à l'envers : `L[::-1]` l'inverse.

⑤ Élément ou tranche

Un indice seul donne un **élément** : `L[2]` vaut `'c'`. Une tranche donne toujours une **nouvelle séquence** du même type : `L[2:3]` vaut `['c']`, une liste d'un élément.

⑥ Chaînes et tuples

```
mot = "Python"
print(mot[0:3])      # Pyt
print(mot[::-1])     # nohtyP
date = (14, 7, 1789)
print(date[:2])      # (14, 7)
```

La tranche d'une chaîne est une chaîne, celle d'un tuple est un tuple.

⑦ Remplacer une tranche

```
L = ['a', 'b', 'c', 'd', 'e', 'f']
L[2:4] = ['X', 'Y', 'Z']
print(L)   # ['a', 'b', 'X', 'Y', 'Z', 'e', 'f']
```

Les deux éléments d'indices 2 et 3 sont remplacés par trois éléments : la liste en compte maintenant sept. On ne peut pas remplacer la tranche d'une chaîne ou d'un tuple, qui sont immuables : `mot[0:1] = "P"` provoque l'erreur `TypeError`.

Tranche	Éléments
`L[i:j]`	des indices `i` à `j - 1`
`L[:j]`	les `j` premiers
`L[i:]`	tous sauf les `i` premiers
`L[-k:]`	les `k` derniers
`L[::k]`	un élément sur `k`
`L[::-1]`	tous, à l'envers

À retenir.

- Dans `L[début:fin]`, le début est inclus, la fin exclue.
- `L[:]` copie une liste ; `L[::-1]` l'inverse.
- Une tranche est une nouvelle séquence du même type.
- Seule une liste permet de remplacer une tranche.

Dictionnaires

Fiche de synthèse

1 Définition

Définition 1 **(Dictionnaire)** Un **dictionnaire** associe des **clés** à des **valeurs**. Il s'écrit entre accolades ; chaque **entrée** s'écrit `clé: valeur`, et les entrées sont séparées par des virgules.

```
eleve = {"nom": "Dupont", "prenom": "Marie",
         "age": 16}
vide = {}
```

Un dictionnaire n'a pas d'indices : `eleve[0]` provoque une erreur.

2 Lire une valeur

```
print(eleve["prenom"])           # Marie
print(eleve["classe"])           # KeyError
print(eleve.get("classe"))       # None
print(eleve.get("classe", "?")) # ?
```

On lit une valeur en écrivant sa clé entre crochets. Une clé absente provoque l'erreur `KeyError` ; la méthode `get` renvoie alors `None`, ou la valeur par défaut donnée en second argument.

3 Ajouter, modifier, supprimer

```
eleve["classe"] = "1NSI"   # ajout
eleve["classe"] = "TNSI"   # modification
del eleve["classe"]        # suppression
```

Une clé nouvelle provoque un ajout, une clé existante une modification : un dictionnaire ne contient jamais deux fois la même clé.

4 Longueur et appartenance

`len(eleve)` donne le nombre d'entrées. `in` cherche parmi les **clés** : `"nom" in eleve` vaut `True`, mais `"Dupont" in eleve` vaut `False`. Pour chercher une valeur, on écrit `"Dupont" in eleve.values()`.

5 Parcourir

```
for cle in eleve:                  # les clés
    print(f"{cle} : {eleve[cle]}")
for valeur in eleve.values():      # les valeurs
    print(valeur)
for cle, valeur in eleve.items():  # les couples
    print(f"{cle} → {valeur}")
```

Les clés sont parcourues dans l'ordre où les entrées ont été ajoutées. `items` donne des couples (clé, valeur), que le déballage sépare en deux variables.

6 Clés et valeurs

- Une **clé** est immuable : entier, chaîne de caractères ou tuple. Une liste ne peut pas être une clé. On repère ainsi les cases d'un plateau de jeu par des tuples : `plateau[(0, 1)] = "cavalier"`.
- Une **valeur** peut être de n'importe quel type, liste ou dictionnaire compris : `fiche["notes"][2]` donne l'élément d'indice 2 de la liste `fiche["notes"]`.

7 Une liste de dictionnaires

```
classe = [{"prenom": "Marie", "moyenne": 14.5},
          {"prenom": "Alice", "moyenne": 16.0}]
for e in classe:
    print(f"{e['prenom']} : {e['moyenne']}")
```

Un ensemble de valeurs repérées par des noms, et non par des positions, s'appelle un **p-uplet nommé** : un dictionnaire le représente.

8 Compter les apparitions

```
frequences = {}
for lettre in "abracadabra":
    n = frequences.get(lettre, 0)
    frequences[lettre] = n + 1
# {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
```

Pour une lettre pas encore rencontrée, `get` renvoie 0, et le calcul donne 1.

9 Dictionnaire en compréhension

`{n: n ** 2 for n in range(6)}` associe à chaque entier de 0 à 5 son carré. `{prenom: note for prenom, note in eleves}` transforme une liste de couples en dictionnaire.

À retenir.

- On lit une valeur par sa clé : `d[cle]` ; `d.get(cle, defaut)` évite l'erreur `KeyError`.
- `d[cle] = valeur` ajoute ou modifie ; `del d[cle]` supprime.
- `in` cherche parmi les clés ; `keys`, `values` et `items` parcourent clés, valeurs et couples.
- Une clé est immuable ; une valeur peut être de tout type.

Fichiers

Fiche de synthèse

1 Pourquoi des fichiers

La mémoire vive est **volatile** : tout ce qu'elle contient disparaît quand le programme s'arrête. Un **fichier**, enregistré sur un disque, conserve les données d'une exécution à l'autre : scores, réglages, relevés de mesures.

2 Ouvrir un fichier

```
with open('eleves.txt', 'r',
          encoding='utf-8') as f:
    contenu = f.read()
# ici, le fichier est fermé
```

`open` reçoit le nom du fichier et le **mode d'ouverture**. Le bloc `with ... as f` range le fichier ouvert dans la variable `f` et le **ferme automatiquement** à sa sortie, même en cas d'erreur. Sans `with`, il faudrait appeler `f.close()`. L'encodage `utf-8` garantit l'affichage correct des lettres accentuées.

Mode	Rôle	Fichier existant	Fichier absent
`'r'`	lecture	lit le contenu	erreur
`'w'`	écriture	**efface** le contenu	crée le fichier
`'a'`	ajout	écrit à la fin	crée le fichier

3 Écrire

```
with open('animaux.txt', 'w',
          encoding='utf-8') as f:
    f.write('Chat\n')
    f.write('Chien\n')

with open('animaux.txt', 'a',
          encoding='utf-8') as f:
    f.writelines(['Girafe\n', 'Tigre\n'])
```

`write` écrit une chaîne, `writelines` une liste de chaînes ; ni l'une ni l'autre n'ajoute de retour à la ligne : on écrit `\n` soi-même. Elles n'acceptent que des chaînes : une note s'écrit avec une chaîne formatée, `f'{nom};{note}\n'`. Le mode `'w'` efface d'abord le fichier ; le mode `'a'` ajoute à la fin.

4 Lire

```
with open('eleves.txt', 'r',
          encoding='utf-8') as f:
    for ligne in f:
        print(ligne.strip())
```

Chaque ligne lue se termine par `\n` ; sans `strip`, `print` ajouterait un second retour à la ligne et l'on verrait une ligne vide entre deux noms. `strip` retire espaces, tabulations et `\n` aux deux bouts.

Lecture	Résultat
`for ligne in f`	une ligne à chaque tour, la plus économe en mémoire
`f.readline()`	la ligne suivante, par exemple pour sauter un en-tête
`f.read()`	tout le fichier en une seule chaîne, pour un petit fichier
`f.readlines()`	la liste des lignes, chacune terminée par `\n`

5 Découper une ligne

Lire un fichier structuré. Chaque ligne de `notes.txt` contient un nom et une note séparés par un point-virgule, comme `Alice Martin;16`.

```
with open('notes.txt', 'r',
          encoding='utf-8') as f:
    for ligne in f:
        parties = ligne.strip().split(';')
        nom = parties[0]
        note = int(parties[1])
        print(f'{nom} a obtenu {note}/20')
```

On retire le `\n`, on découpe sur le séparateur, puis on convertit en nombre ce qui doit l'être : une ligne lue est toujours une chaîne.

À retenir.

- `with open(nom, mode, encoding='utf-8') as f` ouvre un fichier et le ferme tout seul.
- `'r'` lit, `'w'` efface puis écrit, `'a'` ajoute à la fin.
- `write` n'ajoute pas de `\n` ; `strip` le retire à la lecture.
- `split(';')` découpe une ligne ; on convertit les nombres avec `int` ou `float`.

CSV

Fiche de synthèse

1 Le format CSV

Définition 1 **(Fichier CSV)** Un fichier **CSV** (valeurs séparées par des virgules) est un fichier texte qui contient une table : chaque ligne est un **enregistrement**, et ses valeurs sont séparées par un caractère, le **délimiteur**. La première ligne, l'**en-tête**, donne le nom des colonnes, appelées **descripteurs** ; un enregistrement donne une **valeur** à chaque descripteur.

Le fichier `meteo.csv` commence ainsi :

```
ville,région,température,pluie
Lyon,Auvergne-Rhône-Alpes,24,0
Brest,Bretagne,17,6
```

En France, les tableurs utilisent souvent le point-virgule, car la virgule écrit les nombres décimaux. Une valeur qui contient le délimiteur s'écrit entre guillemets : `"Paris, 5e"`.

2 Lire avec csv.reader

```
import csv
with open('meteo.csv', encoding='utf-8') as f:
    lecteur = csv.reader(f, delimiter=',')
    next(lecteur)           # passe l'en-tête
    total = 0
    nombre = 0
    for ligne in lecteur:
        total = total + int(ligne[2])
        nombre = nombre + 1
print(f'Moyenne : {total / nombre} degrés')
```

`csv.reader` donne chaque ligne sous forme de **liste** : `['Lyon', 'Auvergne-Rhône-Alpes', '24', '0']`. `next(lecteur)` saute une ligne, ici l'en-tête.

Le module `csv` donne toujours des **chaînes de caractères**, même pour des nombres : on convertit avec `int` ou `float` avant de calculer.

3 Lire avec csv.DictReader

```
with open('meteo.csv', encoding='utf-8') as f:
    lecteur = csv.DictReader(f, delimiter=',')
    for ligne in lecteur:
        ville = ligne["ville"]
        print(f'{ville} : {ligne["pluie"]} mm')
```

`csv.DictReader` donne chaque ligne sous forme de **dictionnaire** dont les clés sont les descripteurs. `ligne["pluie"]` est plus clair que `ligne[3]`, et reste juste si l'ordre des colonnes change.

4 Écrire

```
releves = [['ville', 'température'],
           ['Rennes', 18], ['Toulouse', 27]]
with open('releves.csv', 'w', encoding='utf-8',
          newline='') as f:
    ecrivain = csv.writer(f, delimiter=',')
    for ligne in releves:
        ecrivain.writerow(ligne)
```

`writerow` écrit une liste ; les nombres deviennent du texte. Le mode `'a'` ajoute des lignes à la fin d'un fichier existant ; `newline=''` évite une ligne vide après chaque enregistrement sous Windows. `csv.DictWriter(fichier, fieldnames=[...])` écrit des dictionnaires : `writeheader` écrit l'en-tête, puis `writerow` une ligne.

5 Une table en mémoire

```
def charger_csv(nom):
    with open(nom, encoding='utf-8') as f:
        return list(csv.DictReader(f))

table = charger_csv('meteo.csv')
seches = [ligne['ville'] for ligne in table
          if int(ligne['pluie']) == 0]
```

Sans `delimiter`, le délimiteur est la virgule. La table devient une **liste de dictionnaires**, une par ligne, qu'on peut parcourir, filtrer ou trier autant qu'on veut. Ici, `seches` vaut `['Lyon', 'Nice']`.

6 Rechercher avec plusieurs critères

```
ensoleillees = [
    ligne['ville'] for ligne in table
    if int(ligne['température']) > 25
    and int(ligne['pluie']) == 0
]
```

Avec `and`, une ligne est gardée si toutes les conditions sont vraies ; avec `or`, si au moins une l'est. Ici, `ensoleillees` vaut `['Nice']`.

⑦ Doublons et cohérence

Un **doublon** est un enregistrement présent deux fois : on le repère en gardant dans une liste `vus` les valeurs déjà rencontrées. Chaque descripteur a un **domaine de valeurs** : une valeur qui en sort (une pluie négative) signale une erreur de saisie.

⑧ Trier selon une colonne

```
def note(ligne):
    return ligne["note"]

par_note = sorted(eleves, key=note)
decroissant = sorted(eleves, key=note,
                     reverse=True)
```

Le paramètre `key` reçoit une fonction, écrite sans parenthèses : `sorted` l'applique à chaque ligne et trie selon les valeurs obtenues. `reverse=True` inverse l'ordre ; `max(eleves, key=note)` donne la ligne de meilleure note.

⑨ Fusionner deux tables

```
def fusionner(table1, table2, cle):
    resultat = []
    for ligne1 in table1:
        for ligne2 in table2:
            if ligne1[cle] == ligne2[cle]:
                nouvelle = {}
                for k in ligne1:
                    nouvelle[k] = ligne1[k]
                for k in ligne2:
                    nouvelle[k] = ligne2[k]
                resultat.append(nouvelle)
    return resultat
```

Les deux tables ont un descripteur commun, la **clé**, de même domaine de valeurs : deux lignes sont réunies quand elles ont la même valeur pour ce descripteur. Une ligne sans partenaire dans l'autre table n'est pas gardée.

À retenir.

- `csv.reader` donne des listes, `csv.DictReader` des dictionnaires.
- Les valeurs lues sont des chaînes : on les convertit avant de calculer.
- On recherche en combinant des conditions avec `and` et `or` ; on fusionne deux tables sur un descripteur commun.
- Avant d'exploiter une table, on vérifie les doublons et les domaines de valeurs.

Interfaces graphiques

Fiche de synthèse

1 Le principe

Ouverture hors programme : `tkinter` est fournie avec Python, `pygame` sert aux jeux, `streamlit` aux applications web. Une **interface graphique** crée une fenêtre et ses composants, les **widgets** (boutons, champs de saisie, textes), associe des fonctions aux actions de l'utilisateur, puis lance une **boucle d'événements**, qui attend ces actions et appelle les fonctions correspondantes.

2 Tkinter

```
import tkinter as tk

def convertir(event=None):
    f = float(temperature.get())
    c = 5 / 9 * (f - 32)
    resultat.config(text=f"{c:.2f} °C")

fenetre = tk.Tk()
fenetre.title("Fahrenheit en Celsius")
temperature = tk.Entry(fenetre)
temperature.bind("<Return>", convertir)
resultat = tk.Label(fenetre, text="")
bouton = tk.Button(fenetre, text="Convertir",
                   command=convertir)
temperature.grid(row=0, column=0)
bouton.grid(row=0, column=1)
resultat.grid(row=1, column=0)
fenetre.mainloop()
```

Instruction	Rôle
`tk.Tk()`	crée la fenêtre principale
`tk.Label`	affiche un texte
`tk.Entry`	champ de saisie, lu par `get()`
`tk.Button`	bouton ; `command` donne la fonction appelée
`bind("<Return>", f)`	appelle `f` à l'appui sur Entrée
`pack()`, `grid()`	place le widget, empilé ou en grille
`mainloop()`	lance la boucle d'événements

Le paramètre `event=None` permet d'appeler `convertir` depuis le bouton, sans argument, comme depuis la touche Entrée, qui transmet l'événement. `command=fenetre.destroy` ferme la fenêtre.

3 Pygame

Un programme Pygame est une boucle qui, à chaque tour, lit les événements, met à jour l'état du jeu, dessine l'écran, puis attend le tour suivant.

```
import pygame

pygame.init()
ecran = pygame.display.set_mode((600, 400))
horloge = pygame.time.Clock()
x, y = 280, 180
fini = False
while not fini:
    for evenement in pygame.event.get():
        if evenement.type == pygame.QUIT:
            fini = True
    touches = pygame.key.get_pressed()
    if touches[pygame.K_RIGHT]:
        x = x + 4
    ecran.fill((30, 30, 60))
    pygame.draw.rect(ecran, (250, 200, 40),
                     [x, y, 40, 40])
    pygame.display.flip()
    horloge.tick(60)
pygame.quit()
```

— L'origine des coordonnées est le coin **supérieur gauche** : `y` augmente vers le bas. `flip` affiche l'image dessinée, `tick(60)` limite à 60 images par seconde.

— Un clic produit l'événement `MOUSEBUTTONDOWN`, dont l'attribut `pos` donne la position. Le clic est dans un disque de centre (c_x, c_y) et de rayon r si $d_x^2 + d_y^2 \leqslant r^2$, avec d_x et d_y les écarts au centre : comparer les carrés évite une racine carrée.

4 Streamlit

```
import streamlit as st

st.title("Convertisseur de températures")
f = st.slider("Degrés Fahrenheit", -40, 120, 68)
c = 5 / 9 * (f - 32)
st.write(f"{f} °F font {c:.1f} °C.")
```

Streamlit transforme un script en application web : chaque appel `st.` ajoute un composant à la page. On lance le script par la commande `streamlit run mon_fichier.py`, qui ouvre l'application dans le navigateur ; à chaque action de l'utilisateur, le script est réexécuté.

À retenir.

— Une interface crée des widgets, leur associe des fonctions et lance une boucle d'événements.

— Tkinter : `Label`, `Entry`, `Button`, placés par `pack` ou `grid`, puis `mainloop`.

Pandas

Fiche de synthèse

Important. Pandas est une ouverture : cette bibliothèque d'analyse de données fait en une ligne les traitements de tables écrits avec des boucles dans le chapitre CSV.

1 Lire une table

```
import pandas as pd

df = pd.read_csv("notes.csv")
print(df.shape)       # (6, 5)
print(df.head(3))     # 3 premières lignes
```

Définition 1 (DataFrame et Series)

- Un **DataFrame** est un tableau à deux dimensions : chaque colonne porte le nom donné par l'en-tête, chaque ligne a un numéro, son **index**, qui commence à 0.
- Une **Series** est une colonne : une suite de valeurs, chacune avec son index.

`shape` donne le nombre de lignes et de colonnes. Pandas reconnaît les nombres, contrairement au module `csv`. Une valeur manquante s'affiche `NaN`. Avec un point-virgule comme délimiteur, on écrit `pd.read_csv("notes.csv", sep=";")`.

2 Colonnes et lignes

```
df["maths"]                 # une colonne
df["maths"].mean()          # sa moyenne
df[["prénom", "nsi"]]       # deux colonnes
df.loc[2]                   # la ligne d'index 2
df.loc[2, "nsi"]            # une seule case
```

Une colonne calcule sans boucle `mean`, `max`, `min`, `sum` et `count`, le nombre de valeurs présentes. Pour plusieurs colonnes, on donne la liste de leurs noms, d'où les doubles crochets.

3 Sélectionner et trier

```
df[df["maths"] >= 12]
df[(df["classe"] == "1A") & (df["nsi"] >= 15)]
df.sort_values(by="nsi", ascending=False)
```

`df["maths"] >= 12` compare chaque note à 12 et donne une série de booléens ; seules les lignes associées à `True` sont gardées. Pour combiner deux conditions, on écrit `&` pour « et », `|` pour « ou », et on met chaque condition entre parenthèses. `sort_values` renvoie un nouveau DataFrame et laisse `df` intact.

4 Valeurs manquantes

`df.isna().sum()` compte les cases vides de chaque colonne. `df.fillna(v)` les remplace par `v` ; `df.dropna()` supprime les lignes incomplètes. Le choix dépend de la situation : remplacer une note absente par 0 serait injuste.

5 Calculer

```
matieres = ["maths", "nsi", "physique"]
df["moyenne"] = df[matieres].mean(axis=1)
df.groupby("classe")[matieres].mean()
df.describe()
```

- Une affectation crée une colonne, calculée pour toutes les lignes à la fois ; `axis=1` calcule la moyenne de chaque ligne, en ignorant les valeurs manquantes.
- `groupby("classe")` regroupe les lignes de même classe et calcule la moyenne de chaque groupe.
- `describe` donne, pour chaque colonne numérique, l'effectif, la moyenne, l'écart type, le minimum, les quartiles et le maximum.

6 Tracer un graphique

```
import matplotlib.pyplot as plt

df.plot(kind="bar", x="prénom", y=matieres)
plt.show()
```

Le paramètre `kind` choisit le graphique : `"bar"` pour des barres, `"line"` pour une courbe, `"scatter"` pour un nuage de points.

À retenir.

- `pd.read_csv` charge une table dans un DataFrame.
- `df["col"]` donne une colonne, `df.loc[i]` une ligne.
- `df[condition]` sélectionne, `sort_values` trie, `groupby` regroupe.

Diversité des langages

Fiche de synthèse

1 Constructions élémentaires

Définition 1 **(Instruction)** Un programme est une suite d'**instructions** : chacune est un ordre donné à l'ordinateur, comme une affectation, un appel de fonction ou un `return`.

Presque tous les langages de programmation offrent les mêmes **constructions élémentaires** : l'affectation, la condition, la boucle et l'appel de fonction. Seule leur écriture change.

2 Un même calcul, trois écritures

Garder et additionner les notes d'au moins 10, en Python :

```
for note in notes:
    if note >= 10:
        somme = somme + note
```

En C : types déclarés, accolades, point-virgule.

```
for (int i = 0; i < taille; i++) {
    if (notes[i] >= 10) {
        somme = somme + notes[i];
    }
}
```

En Haskell, sans boucle ni variable modifiée :

```
admis = filter (>= 10) notes
```

D'un langage à l'autre changent la délimitation des **blocs** (indentation ou accolades), l'écriture des **types** (écrits, déduits ou connus seulement à l'exécution) et la **ponctuation**.

3 Interpréter ou compiler

Définition 2 **(Interpréteur)** Logiciel qui lit les instructions d'un programme une à une et les exécute aussitôt. Exemples : Python (`python3`), JavaScript, PHP, le shell.

Définition 3 **(Compilateur)** Logiciel qui traduit d'abord tout le programme en instructions du processeur et produit un **fichier exécutable**, lancé ensuite. Exemples : C (`gcc`), C++ (`g++`), Rust, Go.

Java et Kotlin sont compilés en un code intermédiaire, exécuté par la **machine virtuelle Java** : le même fichier fonctionne sur tous les ordinateurs qui la possèdent. TypeScript est compilé en JavaScript, après vérification des types.

4 Styles de programmation

- **Impératif** : le programme décrit des étapes, et des variables changent de valeur (`somme = somme + note`).
- **Orienté objet** : le programme est organisé en **classes**, qui regroupent des données et les fonctions qui agissent sur elles (Java). Étudié en terminale.
- **Fonctionnel** : le programme combine des fonctions, sans variable qui change de valeur (Haskell, OCaml, Clojure). Le mot variable y a le sens des mathématiques : sa valeur change d'un appel à l'autre, jamais pendant un calcul.
- **Événementiel** : des fonctions sont associées à des événements (un clic, une touche), et appelées quand ils se produisent (pages web, interfaces graphiques).

Beaucoup de langages permettent plusieurs styles, comme Python. Il en existe d'autres, comme la programmation logique (Prolog).

5 Autres langages

- **SQL** est un langage de **requêtes** : il interroge une base de données (étudié en terminale).
- **HTML** est un langage de **description** : il décrit une page web, sans rien calculer.

6 Choisir un langage

Chaque langage est conçu pour un domaine : pages web (JavaScript dans le navigateur, PHP sur le serveur), applications de téléphone (Kotlin, Swift), programmes rapides (C, C++, Rust), analyse de données et intelligence artificielle (Python), bases de données (SQL).

7 Lire un nouveau langage

1. Repérer l'affectation, la condition, la boucle, les fonctions et la délimitation des blocs.
2. Regarder si les types sont écrits, et si le langage est interprété ou compilé.

À retenir

- Les mêmes constructions élémentaires se retrouvent dans presque tous les langages ; seule leur écriture change.
- Un programme est interprété (Python) ou compilé avant d'être exécuté (C, C++, Rust, Go).
- Styles : impératif, orienté objet, fonctionnel, événementiel ; on choisit un langage selon son domaine.
- SQL et HTML ne sont pas des langages de programmation.

Turtle

Fiche de synthèse

① Le principe

Ouverture hors programme, fournie avec Python. Une **tortue** tient un **crayon** : quand elle avance, elle trace. Elle part du centre $(0, 0)$ en regardant vers la droite ; l'axe des y va vers le **haut**. Directions : 0 à droite, 90 en haut, 180 à gauche, 270 en bas.

```
import turtle

t = turtle.Turtle()
t.forward(100)      # avancer de 100 pas
t.left(90)          # tourner de 90 degrés
t.forward(50)
turtle.done()       # garder la fenêtre ouverte
```

Instruction	Rôle
`forward(d)`, `backward(d)`	avancer, reculer
`left(a)`, `right(a)`	tourner sur place
`goto(x, y)`	aller au point
`penup()`, `pendown()`	lever, baisser le crayon
`color(trait, fond)`	couleurs
`begin_fill()`, `end_fill()`	remplir la forme
`circle(r)`, `dot(d)`	cercle, point
`speed(0)`, `hideturtle()`	vitesse, cacher

② Boucles et polygones

Un polygone régulier à n côtés : n fois « avancer, tourner ». La tortue fait un tour complet, donc chaque rotation vaut $360/n$ degrés (120 pour un triangle, et non 60).

```
def polygone(t, n, cote):
    for i in range(n):
        t.forward(cote)
        t.left(360 / n)
```

Une fonction qui dessine commence par placer la tortue (`penup`, `goto`, `setheading(0)`, `pendown`). Une variable modifiée à chaque tour (longueur, angle, couleur) donne spirales et rosaces. `turtle.tracer(0)` puis `turtle.update()` dessinent d'un coup.

③ Événements

```
ecran = turtle.Screen()

def avancer():
    t.forward(20)

def relier(x, y):           # reçoit le clic
    t.goto(x, y)

ecran.onkey(avancer, "Up") # nom, sans ()
ecran.onclick(relier)
ecran.listen()              # écouter le clavier
turtle.done()               # attendre
```

`ecran.ontimer(f, 20)` appelle `f` une fois, 20 ms plus tard : la fonction se reprogramme pour animer. Une fonction qui modifie une variable du programme principal la déclare avec `global`.

À retenir.

- Polygone à n côtés : rotation de $360/n$ degrés.
- `penup()` avant `goto` pour ne pas tracer.
- `onkey(f, "Up")` : le nom de la fonction, sans parenthèses.
- Ne jamais nommer son fichier `turtle.py`.

Tkinter

Fiche de synthèse

1 Programmation événementielle

Ouverture hors programme, fournie avec Python. Le programme crée les **widgets**, leur associe des fonctions, puis lance la **boucle d'événements** `mainloop()`, qui attend les actions de l'utilisateur (clic, touche) et appelle les fonctions associées.

```
import tkinter as tk

compteur = 0

def cliquer():
    global compteur
    compteur = compteur + 1
    texte.config(text=f"Clics : {compteur}")

fenetre = tk.Tk()
fenetre.title("Compteur")
texte = tk.Label(fenetre, text="Clics : 0")
texte.pack()
bouton = tk.Button(fenetre, text="Cliquer",
                   command=cliquer)
bouton.pack()
fenetre.mainloop()
```

— `command=cliquer` : le **nom** de la fonction, sans parenthèses. Avec `cliquer()`, elle serait appelée tout de suite, une seule fois.

— `global` pour modifier une variable du programme principal.

— `config(text=...)` modifie un widget affiché.

2 Saisie

```
def verifier(event=None):
    saisie = champ.get()       # une chaîne
    if saisie.isdigit() and int(saisie) == 56:
        message.config(text="Bravo")
    champ.delete(0, "end")     # vider

champ.bind("<Return>", verifier)
```

`get()` renvoie toujours une chaîne : la tester, puis la convertir. Une fonction liée par `bind` reçoit un argument `event`.

3 Disposition

Une méthode par conteneur : `pack()` empile, `grid(row=..., column=...)` range en grille (`sticky`, `columnspan`), `place(x=..., y=...)` positionne. Un `Frame` regroupe des widgets. Créer et placer en deux lignes : `grid()` renvoie `None`.

4 Le temps et le dessin

```
def decompter():
    global restant
    if restant > 0:
        restant = restant - 1
        fenetre.after(1000, decompter)
```

Jamais de `while` ni de `time.sleep` dans un gestionnaire : la fenêtre se figerait. Un `Canvas` dessine (`create_rectangle`, `create_oval`, `move`, `coords`) ; l'origine est en **haut à gauche**, y augmente vers le bas. Un bouton qui transmet une valeur : `command=lambda c=c: ajouter(c)`.

À retenir.

— Créer, placer, associer, puis `mainloop()` en dernier.

— `command=f` sans parenthèses ; `get()` renvoie une chaîne.

— `after(ms, f)` remplace `while` et `sleep`.

Pygame

Fiche de synthèse

1 La boucle de jeu

Ouverture hors programme (`pip install pygame`). À chaque image : lire les **événements**, **mettre à jour** l'état, **dessiner** puis afficher, **attendre**.

```
import pygame

pygame.init()
ecran = pygame.display.set_mode((600, 400))
horloge = pygame.time.Clock()
x, vx = 100, 5
en_cours = True
while en_cours:
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            en_cours = False
    x = x + vx                  # mise à jour
    if x < 20 or x > 580:
        vx = -vx                # rebond
    ecran.fill((20, 30, 60))    # effacer
    pygame.draw.circle(ecran, (255, 140, 0),
                       (x, 200), 20)
    pygame.display.flip()       # afficher
    horloge.tick(60)            # 60 images/s
pygame.quit()
```

Origine en **haut à gauche**, y vers le bas ; couleur `(rouge, vert, bleu)` de 0 à 255. Formes : `pygame.draw.rect`, `circle`, `line`, `ellipse`, `polygon`.

2 Clavier et souris

- Événement `KEYDOWN`, `e.key == pygame.K_SPACE` : une fois par appui (sauter, tirer).
- `pygame.key.get_pressed()[pygame.K_LEFT]` : tant que la touche est maintenue (se déplacer).
- `MOUSEBUTTONDOWN`, `e.pos` : position du clic.

3 Rectangles et collisions

`r = pygame.Rect(x, y, l, h)` : attributs `x`, `y`, `right`, `bottom`, `center`, modifiables. `r.colliderect(r2)` : chevauchement ; `r.collidepoint(pos)` : clic dans un bouton.

4 Texte, images, sons

```
police = pygame.font.SysFont("Arial", 24)
img = police.render("Score : 3", True, BLANC)
ecran.blit(img, (10, 10))

DOSSIER = os.path.dirname(
    os.path.abspath(__file__))
v = pygame.image.load(os.path.join(
    DOSSIER, "images", "v.png")).convert_alpha()
```

Polices, images et sons se chargent **avant** la boucle. `pygame.time.get_ticks()` donne le temps en millisecondes.

5 Organiser

Une variable `etat` (`"menu"`, `"jeu"`, `"fin"`) choisit quoi traiter et dessiner. Les objets nombreux vont dans une liste, qu'on reconstruit pour retirer des éléments. Une fonction par tâche, des constantes en majuscules.

À retenir.

- `event.get()`, `fill`, `flip`, `tick` à chaque tour.
- `KEYDOWN` une fois, `get_pressed` en continu.
- Charger avant la boucle, chemins depuis `__file__`.

Flask

Fiche de synthèse

1 Le principe

Flask (`pip install flask`) est une bibliothèque Python pour écrire un **serveur web**. Le navigateur (client) envoie une requête HTTP ; Flask appelle la **vue**, la fonction associée à l'adresse ; ce qu'elle renvoie est la page.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def accueil():
    return "<h1>Bonjour !</h1>"

@app.route("/table/<int:n>")
def table(n):
    return f"<p>{n} x 2 = {n * 2}</p>"

if __name__ == "__main__":
    app.run(debug=True)
```

Le site se visite à l'adresse `http://127.0.0.1:5000`. Adresse sans route : erreur 404.

2 Gabarits

Dans le dossier `templates` ; `render_template("page.html", x=valeur)` transmet les variables. Jinja : `{{ x }}` insère, `{% for %}` répète, `{% if %}` choisit. Une page hérite d'une base : `{% extends "base.html" %}`, puis `{% block contenu %}`. CSS et images dans `static`, appelés par `url_for('static', filename='style.css')`.

3 Formulaires

```
from flask import request

@app.route("/imc", methods=["GET", "POST"])
def imc():
    message = ""
    if request.method == "POST":
        m = float(request.form["masse"])
        t = float(request.form["taille"])
        message = f"IMC : {m / t ** 2:.1f}"
    return render_template("imc.html",
                           message=message)
```

GET : données dans l'adresse, `request.args.get("mot", "")`. POST : `request.form`, et `methods` sinon erreur 405. Les données arrivent en chaînes.

4 Session

HTTP est sans mémoire. `session["score"] = 3` garde une valeur propre à chaque visiteur, dans un cookie signé par `app.secret_key`. Après un POST, `redirect(url_for("vue"))`. Mots de passe : enregistrer une empreinte (`generate_password_hash`), jamais le mot de passe.

5 Serveur et navigateur

Python et Jinja s'exécutent sur le **serveur** (partie serveur, *back end*) ; HTML, CSS et JavaScript dans le **navigateur** (partie client, *front end*). Flask envoie les fichiers de `static` sans les exécuter ; un `print` d'une vue s'affiche dans le terminal du serveur. En ligne : un hébergeur, un fichier `requirements.txt` (`flask`, `gunicorn`) et la commande `gunicorn app:app` ; jamais `debug=True`.

À retenir.

- Une adresse, une vue ; `templates` et `static`.
- `request.args` (GET), `request.form` (POST).
- `session` et `secret_key` pour se souvenir d'un visiteur.
- Python sur le serveur, JavaScript dans le navigateur.

Bases de numération

Fiche de synthèse

1 Écrire un nombre dans une base

Définition 1 **(Base b)** En **base** b, on dispose de b chiffres, de 0 à $b-1$. En partant de la droite, le chiffre de rang k a pour **poids** b^k : il compte des paquets de b^k.

En base 10, $374 = 3\times10^2 + 7\times10^1 + 4\times10^0$. On note la base en indice : 101_2 vaut $1\times4 + 0\times2 + 1\times1 = 5$; sans indice, un nombre est écrit en base 10.

— **Base 2**, ou binaire : un circuit distingue deux états, notés 0 et 1. Un chiffre binaire est un **bit** ; huit bits forment un **octet**.

— **Base 16**, ou hexadécimal : seize chiffres, de 0 à 9 puis A, B, C, D, E et F pour 10 à 15. Elle écrit les suites de bits quatre fois plus court.

Base 10	8	9	10	11	12	13	14	15
Base 2	1000	1001	1010	1011	1100	1101	1110	1111
Base 16	8	9	A	B	C	D	E	F

2 Du binaire au décimal

On additionne les poids des bits qui valent 1.

Bit	1	0	1	1	0	1	1	0
Poids	128	64	32	16	8	4	2	1

$10110110_2 = 128 + 32 + 16 + 4 + 2 = 182$. Un octet va de $00000000_2 = 0$ à $11111111_2 = 255$: il prend 256 valeurs.

3 Du décimal au binaire

Divisions successives. On divise le nombre par 2 : le reste donne le bit des unités. On recommence avec le quotient, jusqu'à obtenir un quotient **nul**. On lit les restes du dernier au premier.

Division	Quotient	Reste
$89 = 2\times44 + 1$	44	1
$44 = 2\times22 + 0$	22	0
$22 = 2\times11 + 0$	11	0
$11 = 2\times5 + 1$	5	1
$5 = 2\times2 + 1$	2	1
$2 = 2\times1 + 0$	1	0
$1 = 2\times0 + 1$	0	1

Lus de bas en haut, les restes donnent $89 = 1011001_2$. Vérification : $64 + 16 + 8 + 1 = 89$. On ne s'arrête pas quand le quotient vaut 1 : la dernière division donne encore le bit de gauche. Autre méthode : retirer la plus grande puissance de 2 possible, et recommencer, $89 = 64 + 16 + 8 + 1$.

4 Binaire et hexadécimal

Comme $16 = 2^4$, un chiffre hexadécimal correspond à quatre bits.

— Du binaire vers l'hexadécimal : on découpe en paquets de quatre bits **à partir de la droite**, puis on remplace chaque paquet. 10110110_2 donne 1011 et 0110, soit $B6_{16}$.

— De l'hexadécimal vers le binaire : on remplace chaque chiffre par ses quatre bits. $2C9_{16} = 0010\ 1100\ 1001_2$.

— Du décimal vers l'hexadécimal : divisions successives par 16. $2026 = 16\times126+10$, $126 = 16\times7+14$, $7 = 16\times0+7$: $2026 = 7EA_{16}$.

L'hexadécimal écrit les couleurs du Web, `#2E86C1` pour rouge 46, vert 134 et bleu 193, les adresses mémoire et les adresses MAC des cartes réseau.

5 En Python

```
print(bin(89))              # 0b1011001
print(int('1011001', 2))    # 89
print(hex(89))              # 0x59
print(int('59', 16))        # 89
```

6 Addition en binaire

On additionne colonne par colonne en partant de la droite : $1 + 1 = 10_2$, on écrit 0 et on retient 1 ; $1 + 1 + 1 = 11_2$, on écrit 1 et on retient 1. Ainsi $0110_2 + 0111_2 = 1101_2$, soit $6 + 7 = 13$. Sur un nombre fixé de bits, une retenue qui sort à gauche est perdue : c'est un **dépassement de capacité**. Sur un octet, $11111111_2 + 1$ donne 00000000_2.

À retenir.

— Le chiffre de rang k pèse b^k en base b.

— Binaire vers décimal : additionner les poids des bits à 1.

— Décimal vers binaire : divisions successives par 2 jusqu'au quotient nul, restes lus de bas en haut.

— Un chiffre hexadécimal vaut quatre bits.

Représentation des entiers

Fiche de synthèse

1 Entiers naturels

Un **entier naturel** est un entier positif ou nul. Sur n bits, on représente les entiers de 0 à $2^n - 1$: de 0 à 255 sur un octet, de 0 à 65 535 sur deux octets. Par exemple, $9 = 00001001_2$ et $255 = 11111111_2$.

En Python, la taille d'un entier n'est pas limitée : `2 ** 100` se calcule exactement, Python réservant la mémoire nécessaire.

2 Pourquoi pas un simple bit de signe

Le **bit de poids fort** est le bit de gauche, de plus grand poids ; le **bit de poids faible** est celui de droite. Utiliser le bit de poids fort comme signe, 10000010 pour −2, pose deux problèmes : zéro a deux écritures, 00000000 et 10000000, et l'addition binaire donne un résultat faux : 00000011 + 10000110 = 10001001, soit 3 + (−6) = −9.

3 Le complément à deux

Définition 1 **(Complément à deux)** Sur n bits, le **complément à deux** représente un entier positif comme un entier naturel, avec un bit de poids fort à 0, et un entier négatif x par l'écriture binaire de $2^n - |x|$. Le bit de poids fort vaut alors 1. On représente ainsi les entiers de -2^{n-1} à $2^{n-1} - 1$.

Sur huit bits, de −128 à 127 ; sur quatre bits, de −8 à 7.

Coder un entier négatif sur n bits.

1. Écrire sa valeur absolue en binaire sur n bits : $19 = 00010011_2$.
2. Inverser tous les bits : 11101100.
3. Ajouter 1 : 11101101. Donc −19 s'écrit 11101101_2.

Vérification : 00010011 + 11101101 = 00000000, la retenue qui sort à gauche étant ignorée, et 19 + (−19) = 0.

Raccourci : en lisant de droite à gauche, on recopie les bits jusqu'au premier 1 compris, puis on inverse tous les autres. $20 = 00010100_2$ donne $-20 = 11101100_2$.

4 Lire un entier relatif

— Si le bit de poids fort vaut 0, le nombre est positif : on le lit directement.

— S'il vaut 1, le nombre est négatif : on applique de nouveau le complément à deux pour obtenir sa valeur absolue. 11101101 donne 00010010, puis 00010011 = 19 : le nombre vaut −19.

Cas particulier : 10000000_2 vaut −128 sur huit bits, et +128 n'est pas représentable. Il y a un nombre négatif de plus que de nombres positifs non nuls.

5 L'addition fonctionne

Le même circuit d'addition sert pour les entiers naturels et les entiers relatifs : 00000011 + 11111010 = 11111101, soit 3 + (−6) = −3.

6 Dépassement de capacité

Un résultat hors de l'intervalle de codage provoque un **dépassement de capacité**. Sur huit bits, 127 + 1 donne 01111111 + 00000001 = 10000000, qui vaut −128 : on passe du plus grand positif au plus petit négatif. C'est un piège des langages à entiers de taille fixe, comme C ou Java.

À retenir.

— Sur n bits : entiers naturels de 0 à $2^n - 1$; entiers relatifs de -2^{n-1} à $2^{n-1} - 1$.

— Pour coder $-x$: écrire x, inverser les bits, ajouter 1.

— Le bit de poids fort donne le signe : 1 pour un nombre négatif.

— Le complément à deux permet d'additionner avec le circuit habituel.

Représentation des réels

Fiche de synthèse

① Le problème

En Python, `0.1 + 0.2` vaut `0.30000000000000004`, et `0.1 + 0.2 == 0.3` vaut `False`. Ce n'est pas une erreur de Python : c'est une conséquence de l'écriture des nombres à virgule en binaire.

② Partie décimale en binaire

Après la virgule, les chiffres binaires ont pour poids les puissances négatives de 2 : $2^{-1} = 0{,}5$, $2^{-2} = 0{,}25$, $2^{-3} = 0{,}125$, $2^{-4} = 0{,}0625$.

$$110{,}101_2 = 4 + 2 + 0{,}5 + 0{,}125 = 6{,}625$$

Partie décimale vers le binaire. On multiplie la partie décimale par 2 : la partie entière du résultat est le bit suivant. On recommence avec la nouvelle partie décimale, jusqu'à obtenir 0.

Calcul	Bit
$0{,}6875 \times 2 = 1{,}375$	1
$0{,}375 \times 2 = 0{,}75$	0
$0{,}75 \times 2 = 1{,}5$	1
$0{,}5 \times 2 = 1$	1

Donc $0{,}6875 = 0{,}1011_2$. Mais pour 0,1, les calculs ne s'arrêtent jamais : $0{,}1 = 0{,}0001100110011\ldots_2$, le motif 0011 se répétant sans fin. L'ordinateur, qui n'a qu'un nombre fini de bits, ne stocke qu'une **valeur approchée** de 0,1.

En 1991, l'horloge d'une batterie de missiles Patriot comptait en dixièmes de seconde : l'erreur d'arrondi accumulée en cent heures, 0,34 seconde, l'a empêchée d'intercepter un missile.

③ La norme IEEE 754

Définition 1 **(Nombre flottant)** La norme **IEEE 754** (1985) écrit un nombre non nul en notation scientifique binaire :

$$(-1)^S \times 1{,}F \times 2^e$$

avec un bit de **signe** S, une **mantisse** F, les chiffres après la virgule, et un **exposant** e, enregistré avec un **biais** : $E = e + 127$ en simple précision.

Format	Signe	Exposant	Mantisse	Chiffres
simple, 32 bits	1	8, biais 127	23	environ 7
double, 64 bits	1	11, biais 1023	52	environ 15

Le type `float` de Python est en double précision. Le biais permet d'écrire un exposant négatif sans bit de signe : $e = -3$ est enregistré $E = 124$.

Coder $-6{,}625$ en simple précision.

1. Signe : le nombre est négatif, $S = 1$.
2. Binaire : $6{,}625 = 110{,}101_2$.
3. Normaliser : $110{,}101_2 = 1{,}10101_2 \times 2^2$, donc $e = 2$.
4. Exposant : $E = 2 + 127 = 129 = 10000001_2$.
5. Mantisse : les chiffres après « 1, », complétés à 23 bits : 10101 suivi de 18 zéros.

Résultat : 1 | 10000001 | 10101000000000000000000.

④ En Python

On ne compare jamais deux flottants avec `==` :

```
import math

print(round(0.1 + 0.2, 10) == 0.3)        # True
print(math.isclose(0.1 + 0.2, 0.3))       # True
```

Les erreurs d'arrondi s'accumulent : ajouter cent mille fois 0,01 en partant de 0 donne 999,9999999992356 et non 1000. Le module `decimal` calcule exactement en base 10.

Remarque 1. *Pour aller plus loin.* Les exposants extrêmes représentent l'infini, le dépassement de capacité, et NaN, « pas un nombre », résultat d'un calcul indéfini comme 0/0.

À retenir.

- Un nombre décimal fini, comme 0,1, peut avoir une écriture binaire infinie : il est arrondi.
- Un flottant s'écrit $(-1)^S \times 1{,}F \times 2^e$, avec un exposant biaisé.
- Python utilise la double précision, 64 bits, environ 15 chiffres significatifs.
- On compare deux flottants avec une tolérance, jamais avec `==`.

Représentation des textes

Fiche de synthèse

1 Encoder un texte

Définition 1 **(Encodage)** Un **encodage** est une convention qui associe un nombre, donc une suite de bits, à chaque caractère : lettre, chiffre, signe de ponctuation, symbole.

2 Le code ASCII

Le code **ASCII** (1963), conçu pour l'anglais, code 128 caractères de 0 à 127 : 95 caractères affichables et 33 caractères de commande. « A » vaut 65, soit 41_{16} ou 01000001_2 ; « a » vaut 97. Sept bits suffisent, mais chaque caractère occupe un octet dont le premier bit vaut 0. L'ASCII ne contient aucune lettre accentuée.

Décoder un texte en ASCII binaire. Découper la suite de bits en octets, traduire chaque octet en décimal, puis chercher le caractère correspondant dans la table ASCII.

3 L'ASCII étendu

Le huitième bit, libre, a servi à ajouter 128 caractères, lettres accentuées et symboles. Mais chaque système a fait ses propres choix, appelés **pages de code** : ISO-8859-1, ou Latin-1, pour l'Europe occidentale, sa variante Windows-1252, ISO-8859-2 pour l'Europe centrale. Un même octet désigne ainsi des caractères différents d'une page à l'autre.

4 Unicode et UTF-8

Unicode donne à chaque caractère de toutes les écritures un numéro unique, son **point de code**, noté `U+` suivi de chiffres hexadécimaux : `U+0041` pour « A », `U+00E9` pour « é », `U+20AC` pour « € ». Ses 128 premiers caractères sont ceux de l'ASCII. Trois formats écrivent un point de code en octets : UTF-32 sur 4 octets, UTF-16 sur 2 ou 4, et **UTF-8** sur 1 à 4 octets, le plus utilisé, car un texte ASCII y reste identique.

Point de code	Octets	Forme binaire
jusqu'à `U+007F`	1	`0xxxxxxx`
jusqu'à `U+07FF`	2	`110xxxxx 10xxxxxx`
jusqu'à `U+FFFF`	3	`1110xxxx` puis deux `10xxxxxx`
jusqu'à `U+10FFFF`	4	`11110xxx` puis trois `10xxxxxx`

« é » a pour point de code 233, soit 11101001_2 : il faut deux octets. Ses onze bits, 000 1110 1001, remplissent les `x` : 11**000011** 10**101001**, soit `C3 A9`. En Latin-1, « é » est l'unique octet `E9`.

Attention. Un texte UTF-8 lu comme du Latin-1 affiche « Ã© » à la place de « é » ; un texte Latin-1 lu comme de l'UTF-8 affiche des caractères de remplacement, un point d'interrogation dans un losange. Ces défauts signalent toujours un mauvais choix d'encodage à la lecture.

5 En Python

```
print(ord('B'))                  # 66
print(chr(67))                   # C
print("é".encode("utf-8"))       # b'\xc3\xa9'
print("é".encode("latin-1"))     # b'\xe9'
print(b"\xc3\xa9".decode("utf-8"))   # é
```

`ord` donne le point de code d'un caractère, `chr` le caractère d'un point de code. `encode` donne les octets d'une chaîne dans un encodage, `decode` reconstitue la chaîne. `\xc3` désigne l'octet `C3`. Pour lire ou écrire un fichier, on précise toujours l'encodage : `open(nom, encoding="utf-8")`.

6 Texte simple et texte enrichi

Un **texte enrichi** ajoute au texte une mise en forme, police, gras, titres, liens, décrite par des **balises** : HTML, XML, RTF, les formats de bureautique `.odt` et `.docx`, ou LaTeX, qui excelle dans les formules mathématiques. Un texte simple n'est qu'une suite de caractères.

À retenir.

- Un encodage associe un nombre à chaque caractère.
- ASCII : 128 caractères sur 7 bits, sans accents ; les pages de code de 8 bits sont incompatibles entre elles.
- Unicode numérote tous les caractères ; UTF-8 les écrit sur 1 à 4 octets.
- `ord` et `chr` passent du caractère au point de code et inversement.

Booléens

Fiche de synthèse

1 Booléens et tables de vérité

Définition 1 **(Booléen et table de vérité)**

- Un **booléen** ne prend que deux valeurs : vrai, noté 1, ou faux, noté 0. En Python, `True` et `False`. Le nom vient de George Boole (1815-1864).
- Une **fonction logique** prend des booléens en entrée et renvoie un booléen.
- Sa **table de vérité** donne la sortie pour chaque combinaison des entrées : 2^n lignes pour n entrées.

Les comparaisons, `==`, `!=`, `<`, `>`, `<=` et `>=`, produisent des booléens.

2 Les opérateurs

a	b	$\overline{a}$	$a \cdot b$	$a + b$	$a \oplus b$	$\overline{a \cdot b}$
0	0	1	0	0	0	1
0	1	1	0	1	1	1
1	0	0	0	1	1	1
1	1	0	1	1	0	0

Opérateur	Notation	Python	Vrai si
NON	$\overline{a}$	`not a`	a est faux
ET	$a \cdot b$	`a and b`	les deux sont vrais
OU	$a + b$	`a or b`	au moins un est vrai
OU exclusif	$a \oplus b$	`a ^ b`	exactement un est vrai
NON-ET	$\overline{a \cdot b}$	`not (a and b)`	les deux ne sont pas vrais

Le OU est **inclusif** : il vaut 1 quand les deux entrées valent 1, comme le « ou » de « on entre avec un billet ou une carte d'abonnement ». Le OU exclusif est le « ou » de « la mer ou la montagne », où l'on ne choisit pas les deux. Le ET donne le même résultat qu'une multiplication, d'où le point.

3 Propriétés

Propriété	ET	OU
élément neutre	$a \cdot 1 = a$	$a + 0 = a$
élément absorbant	$a \cdot 0 = 0$	$a + 1 = 1$
idempotence	$a \cdot a = a$	$a + a = a$
complément	$a \cdot \overline{a} = 0$	$a + \overline{a} = 1$
commutativité	$a \cdot b = b \cdot a$	$a + b = b + a$

- Double négation : $\overline{\overline{a}} = a$. Les deux opérateurs sont aussi associatifs.
- Distributivité : $a \cdot (b + c) = a \cdot b + a \cdot c$, et, sans équivalent en arithmétique, $a + b \cdot c = (a + b) \cdot (a + c)$.
- Absorption : $a + a \cdot b = a$ et $a \cdot (a + b) = a$.

Théorème 1 **(Lois de De Morgan)**

$$\overline{a + b} = \overline{a} \cdot \overline{b} \qquad \overline{a \cdot b} = \overline{a} + \overline{b}$$

« Ne pas (A ou B) » revient à « ne pas A et ne pas B ». En Python, `not (age < 18 or solde < 0)` s'écrit plus lisiblement `age >= 18 and solde >= 0`.

4 Des interrupteurs aux transistors

- Deux interrupteurs en **série** réalisent le ET, deux interrupteurs en **parallèle** le OU, un interrupteur normalement fermé le NON.
- Shannon a montré que l'algèbre de Boole s'applique aux circuits électriques. Dans un processeur, les **portes logiques** sont faites de **transistors**, des interrupteurs commandés électriquement.
- La porte NON-ET, ou NAND, est **universelle** : elle suffit à construire les trois opérateurs. Si l'on note $a \uparrow b = \overline{a \cdot b}$, alors $\overline{a} = a \uparrow a$, $a \cdot b = (a \uparrow b) \uparrow (a \uparrow b)$ et $a + b = (a \uparrow a) \uparrow (b \uparrow b)$.

À retenir.

- ET : les deux vrais ; OU : au moins un vrai ; OU exclusif : exactement un vrai.
- Une table de vérité à n entrées compte 2^n lignes.
- De Morgan : $\overline{a + b} = \overline{a} \cdot \overline{b}$ et $\overline{a \cdot b} = \overline{a} + \overline{b}$.
- Les portes logiques sont faites de transistors ; la porte NON-ET est universelle.

Circuits logiques

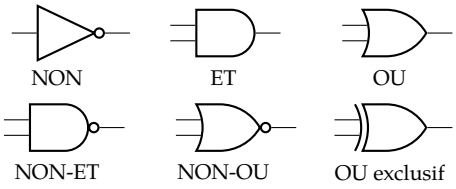
Fiche de synthèse

1 Des transistors aux portes

Définition 1 **(Transistor et porte logique)** Un **transistor** est un interrupteur commandé électriquement : selon le signal reçu, il laisse passer ou bloque le courant. En assemblant des transistors, on construit des **portes logiques**, qui réalisent les opérations booléennes.

Une puce Apple M2 (2022) contient environ 20 milliards de transistors, gravés à 5 nanomètres.

2 Les portes



Porte	Expression	Sortie à 1 quand
NON	$\overline{a}$	a vaut 0
ET	$a \cdot b$	les deux entrées valent 1
OU	$a + b$	au moins une entrée vaut 1
NON-ET	$\overline{a \cdot b}$	au moins une entrée vaut 0
NON-OU	$\overline{a + b}$	les deux entrées valent 0
OU exclusif	$a \oplus b$	exactement une entrée vaut 1

Le petit cercle en sortie d'une porte indique une négation. Ces symboles sont ceux de la norme américaine ; la norme européenne dessine des rectangles. La porte XNOR, non-OU exclusif, vaut 1 quand les deux entrées sont égales.

3 Circuits combinatoires

Définition 2 **(Circuit combinatoire)** Un **circuit combinatoire** est un assemblage de portes dont les sorties ne dépendent que des valeurs actuelles des entrées : il n'a pas de mémoire.

Analyser un circuit. Écrire l'expression de la sortie en suivant les portes de l'entrée vers la sortie, puis dresser la table de vérité, une colonne par résultat intermédiaire. Les lois de De Morgan permettent ensuite de simplifier : $\overline{\overline{a} + \overline{b}} = a \cdot b$.

Concevoir un circuit. Dresser la table de vérité voulue ; pour chaque ligne où la sortie vaut 1, écrire le produit des entrées, chacune niée si elle vaut 0 ; faire la somme de ces produits, puis simplifier. Le vote majoritaire à trois entrées, qui vaut 1 si au moins deux entrées valent 1, se simplifie en $M = a \cdot b + a \cdot c + b \cdot c$.

Le OU exclusif se construit avec des portes ET, OU et NON : $a \oplus b = \overline{a} \cdot b + a \cdot \overline{b}$.

4 Circuits élémentaires

— Le **demi-additionneur** additionne deux bits a et b. Il donne la somme $S = a \oplus b$ et la retenue $C = a \cdot b$: $1 + 1 = 10_2$ donne S = 0 et C = 1.

— L'**additionneur complet** additionne trois bits, a, b et une retenue entrante. Il se construit avec deux demi-additionneurs et une porte OU ; en en enchaînant plusieurs, on additionne des nombres de plusieurs bits.

— Le **multiplexeur** 2 vers 1 recopie l'entrée e_0 si le signal de sélection s vaut 0, l'entrée e_1 s'il vaut 1 : $Y = \overline{s} \cdot e_0 + s \cdot e_1$. C'est un aiguillage.

a	b	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

À retenir.

— Les portes logiques, faites de transistors, réalisent NON, ET, OU, NON-ET, NON-OU et OU exclusif.

— Un circuit combinatoire s'analyse par son expression et sa table de vérité.

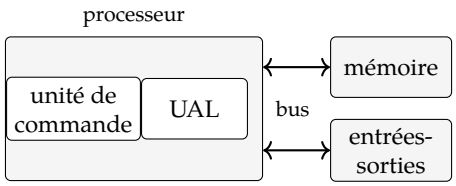
— Le demi-additionneur donne $S = a \oplus b$ et $C = a \cdot b$.

Architecture des ordinateurs

Fiche de synthèse

1 Le modèle de von Neumann

Définition 1 **(Modèle de von Neumann)** Décrit par John von Neumann en 1945, ce modèle organise l'ordinateur en quatre parties : l'**unité arithmétique et logique**, l'**unité de commande**, la **mémoire** et les **entrées-sorties**, reliées par des **bus**. Le **programme est enregistré en mémoire**, sous forme de nombres, comme les données.



- L'**UAL** calcule : additions, comparaisons, opérations logiques.
- L'**unité de commande** lit les instructions en mémoire, les décode et coordonne leur exécution.
- Les **bus** relient les composants : le bus de données transporte les valeurs, le bus d'adresse indique l'emplacement mémoire visé, le bus de commande transmet les ordres, comme lire ou écrire.

Aujourd'hui, des contrôleurs gèrent seuls les entrées-sorties, et les processeurs ont plusieurs **cœurs**, qui exécutent des instructions en parallèle.

2 Le processeur

Le processeur contient l'UAL, l'unité de commande, avec son **séquenceur** rythmé par une horloge, et des **registres**, petites mémoires ultrarapides :

- le **compteur ordinal** contient l'adresse de la prochaine instruction ;
- le **registre d'instruction** contient l'instruction en cours ;
- l'**accumulateur** garde les résultats intermédiaires.

Le cycle d'exécution d'une instruction.

1. **Charger** : lire en mémoire l'instruction dont l'adresse est dans le compteur ordinal, la copier dans le registre d'instruction, puis augmenter le compteur ordinal.
2. **Décoder** : déterminer l'opération à faire et les données concernées.
3. **Exécuter** : l'UAL effectue l'opération ; le résultat va dans un registre ou en mémoire.

La **fréquence d'horloge** est le nombre d'impulsions par seconde : un processeur à 3 GHz en reçoit trois milliards. Les instructions sont de quatre familles : transfert de données, arithmétique, logique et contrôle, comme les sauts conditionnels.

3 Les mémoires

Mémoire	Accès	Capacité	Volatile
registres	moins de 1 ns	quelques dizaines	oui
caches L1 à L3	1 à 10 ns	64 Ko à 32 Mo	oui
RAM	environ 50 ns	8 à 32 Go	oui
SSD	environ 100 µs	256 Go à 2 To	non
disque dur	environ 10 ms	1 à 4 To	non

Plus une mémoire est rapide, plus elle est chère, donc petite. Le processeur cherche une donnée dans le cache L1, puis L2, puis L3, puis en RAM.

- La **RAM**, mémoire vive, contient les programmes en cours et leurs données ; elle est **volatile** : son contenu disparaît à l'extinction.
- La **mémoire de masse** est **persistante** : le disque dur, à plateaux magnétiques, et le SSD, à mémoire flash, plus rapide et sans pièce mobile.

4 Carte mère et loi de Moore

La **carte mère** accueille le processeur et la RAM et relie tous les composants par des bus. Une mémoire non volatile y contient le **BIOS**, remplacé aujourd'hui par l'**UEFI** : le premier programme exécuté à la mise sous tension, qui vérifie le matériel et lance le système d'exploitation.

La **loi de Moore** constate que le nombre de transistors d'une puce double environ tous les deux ans. La miniaturisation approchant des limites physiques, les fabricants multiplient les cœurs plutôt que d'augmenter la fréquence.

À retenir.

- Von Neumann : UAL, unité de commande, mémoire et entrées-sorties, reliées par des bus ; le programme est en mémoire.
- Chaque instruction est chargée, décodée puis exécutée ; le compteur ordinal pointe la suivante.
- Les mémoires vont des registres, rapides et petits, aux disques, lents et grands ; la RAM est volatile.

Langage assembleur

Fiche de synthèse

1 Données et instructions

Dans le modèle de von Neumann, la mémoire contient à la fois les données et le programme. Un même octet, 01000001, vaut 65 lu comme un entier, « A » lu en ASCII, ou un ordre lu par le processeur comme une instruction : c'est l'usage qui décide. Le processeur exécute le mot rangé à l'adresse que contient son **compteur de programme**.

2 Langage machine et assembleur

Définition 1 **(Instruction machine)** Une **instruction machine** est un mot binaire formé d'un **code d'opération**, qui dit quoi faire, et d'**opérandes**, qui disent sur quoi agir : un registre, une valeur ou une adresse. L'ensemble des opérations d'un processeur est son **jeu d'instructions**.

- Le **langage machine** est la suite de bits que le processeur exécute directement.
- Le **langage assembleur** écrit les mêmes instructions avec des noms courts, des **mnémoniques** : `ADD`, `SUB`, `MOV`, `LDR`.
- Le programme **assembleur** traduit le langage assembleur en langage machine, presque ligne pour ligne.

Chaque famille de processeurs a son jeu d'instructions : un programme en langage machine ne fonctionne que sur la sienne. L'assembleur est un langage de **bas niveau** : chaque ligne est une opération élémentaire du processeur. Les instructions se rangent en quatre familles : **transfert**, **calcul**, **comparaison** et **saut**.

3 Instructions du simulateur AQA

Les opérandes s'écrivent `Rn` pour le registre numéro n, `#n` pour la valeur n, et `n` pour le mot de mémoire d'adresse n. Il n'y a pas de variables : le processeur ne calcule qu'avec ses registres, `R0` à `R12`.

Instruction	Effet
`MOV R1, #25`	met 25 dans R1
`LDR R1, 20`	copie dans R1 le mot d'adresse 20
`STR R1, 22`	écrit R1 dans le mot d'adresse 22
`ADD R3, R1, R2`	range R1 + R2 dans R3
`SUB R3, R1, #1`	range R1 − 1 dans R3
`CMP R1, #0`	compare R1 à 0
`BGT etiquette`	saute si « strictement plus grand »
`INP R1, 2`	lit un nombre au clavier dans R1
`OUT R1, 4`	affiche R1
`HALT`	arrête le programme

4 Une boucle

```
        MOV R0, #5
boucle: OUT R0, 4
        SUB R0, R0, #1
        CMP R0, #0
        BGT boucle
        HALT
```

Ce programme affiche 5, 4, 3, 2, 1. L'assembleur n'a ni `if`, ni `while` : on les fabrique avec des **sauts conditionnels**, vers une ligne repérée par une **étiquette**, ici `boucle`. Après une comparaison, des indicateurs signalent un résultat nul, négatif, une retenue ou un dépassement.

5 Compiler ou interpréter

Un langage de **haut niveau**, comme C ou Python, manipule des variables, des listes et des fonctions, loin des registres et des adresses. Il faut le traduire en langage machine :

- un **compilateur** traduit une fois le programme entier en un fichier exécutable : c'est le cas de C ;
- un **interpréteur** traduit et exécute les instructions une à une, à chaque lancement ;
- en méthode mixte, le programme est traduit en **bytecode**, exécuté par une **machine virtuelle** : c'est le cas de Python et de Java.

6 À quoi sert l'assembleur

On n'écrit presque plus de logiciel entier en assembleur : c'est long, difficile à relire, et lié à un seul type de processeur. Il reste utile pour quelques parties du système d'exploitation, pour les calculs les plus exigeants, pour les petits microcontrôleurs, et pour comprendre ce que fait réellement la machine.

À retenir.

- Une instruction machine est un code d'opération suivi d'opérandes.
- L'assembleur écrit le langage machine avec des mnémoniques, propres à chaque famille de processeurs.
- Boucles et conditions se programment avec des comparaisons et des sauts.
- Un compilateur traduit tout le programme ; Python passe par un bytecode et une machine virtuelle.

HTML

Fiche de synthèse

1 Un langage de balises

Définition 1 **(HTML)** **HTML** est le langage qui structure les pages Web. Il décrit le contenu et son rôle, titre, paragraphe, lien, image, à l'aide de **balises** écrites entre chevrons. Il ne fixe pas l'apparence : c'est le rôle du navigateur et de la feuille de style CSS.

Le raccourci Ctrl+U affiche le code HTML d'une page.

2 Structure d'une page

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title>Mon premier document</title>
  </head>
  <body>
    <h1>Titre de mon document</h1>
    <!-- un commentaire -->
    <p>Bonjour, bienvenue sur ma page !</p>
  </body>
</html>
```

— `<head>` contient des informations pour le navigateur : encodage, titre de l'onglet, feuilles de style.

— `<body>` contient le contenu visible.

— Le fichier commence par `<!DOCTYPE html>` ; les balises s'écrivent en minuscules ; un commentaire s'écrit entre `<!--` et `-->`.

3 Balises et attributs

— Une **balise ouvrante** et une **balise fermante** encadrent un contenu : `<p>Un paragraphe.</p>`. Certaines balises n'ont pas de contenu ni de balise fermante : `<br>`, `<img>`.

— Un **attribut** précise une balise ; il s'écrit `nom="valeur"` dans la balise ouvrante : `<a href="page2.html">`.

Les balises s'**imbriquent** : on ferme d'abord la dernière ouverte.

```
<em>Très <strong>bien</strong></em> <!-- oui -->
<em>Très <strong>bien</em></strong> <!-- non -->
```

4 Organiser le texte

Balise	Rôle
`<h1>` à `<h6>`	titres, du plus au moins important
`<p>`	paragraphe
` `	saut de ligne
`<strong>`, `<em>`	texte important, texte mis en valeur
`<mark>`	texte surligné
`<ul>`, `<ol>`	liste à puces, liste numérotée
`<li>`	élément d'une liste
`<div>`	conteneur de blocs
`<span>`	conteneur d'une portion de texte

5 Liens

```
<a href="https://www.csilyon.fr/">CSI</a>
<a href="https://www.csilyon.fr/"
   target="_blank">Nouvel onglet</a>
<a href="page2.html">Page 2</a>
<a href="#introduction">Introduction</a>
```

Un lien `href="#introduction"` conduit à l'élément de la page dont l'attribut `id` vaut `introduction`, appelé **ancre** : `<h2 id="introduction">`. Un même `id` n'apparaît qu'une fois dans une page.

6 Images

```
<figure>
  <img src="elephant.jpg"
       alt="Un éléphant dans la savane">
  <figcaption>Un éléphant</figcaption>
</figure>
```

L'attribut `alt` est obligatoire : il donne un texte de remplacement si l'image ne s'affiche pas, et rend la page accessible. On utilise le format JPEG pour les photographies, PNG pour les illustrations et les captures d'écran, GIF pour les images simples ou animées. Une image placée dans `<a>` devient un lien.

À retenir.

— HTML structure le contenu ; il ne fixe pas l'apparence.

— Les balises s'imbriquent correctement ; les attributs s'écrivent `nom="valeur"`.

— `<a href="...">` crée un lien ; `<img src="..." alt="...">` insère une image.

CSS

Fiche de synthèse

1 Le fond et la forme

Définition 1 **(CSS)** Le **CSS**, feuilles de style en cascade, décrit l'**apparence** d'une page Web : couleurs, polices, tailles, marges, disposition. HTML dit ce que contient la page ; CSS dit comment l'afficher.

Séparer le fond et la forme permet de changer l'apparence de tout un site en modifiant un seul fichier, et d'afficher la même page différemment sur un ordinateur et sur un téléphone. Comme HTML, CSS est normalisé par le W3C.

2 Où écrire le CSS

— **Feuille externe**, la méthode recommandée : les règles sont dans un fichier `style.css`, que chaque page relie dans son en-tête. Tout le site partage ce fichier.

— **Feuille interne** : les règles sont dans la page, entre `<style>` et `</style>`, dans l'en-tête ; pour une page isolée.

— **Attribut** `style` dans une balise, à éviter : `<p style="color: red;">` mélange de nouveau le fond et la forme.

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
```

3 Syntaxe d'une règle

```
p {
    color: red;
    text-align: center;
}
```

Une règle commence par un **sélecteur**, qui désigne les éléments concernés, ici tous les paragraphes. Viennent ensuite, entre accolades, des **déclarations** `propriété: valeur;`. Le point-virgule est obligatoire : sans lui, deux déclarations se fondent en une seule, invalide, et le navigateur les ignore. Un commentaire s'écrit entre `/*` et `*/`.

```
body {
    font-family: Arial, sans-serif;
    font-size: 16px;
}
```

`font-family` donne une liste de polices : le navigateur prend la première installée, sinon la famille générique de la fin, `serif`, avec empattements, `sans-serif`, sans empattements, ou `monospace`, à chasse fixe.

4 Les sélecteurs

Sélecteur	Éléments choisis
`p`	tous les paragraphes
`h1, h2, h3`	les titres `h1`, `h2` et `h3`
`div p`	les paragraphes contenus dans un `div`
`.gras`	les éléments de classe `gras`
`#menu`	l'élément d'identifiant `menu`

La virgule sépare des sélecteurs ; l'espace sélectionne un élément contenu dans un autre.

```
<p class="italique gras" id="intro">...</p>
```

— Une **classe**, attribut `class`, s'applique à plusieurs éléments, et un élément peut avoir plusieurs classes, séparées par des espaces. On la sélectionne avec un point : `.italique`.

— Un **identifiant**, attribut `id`, désigne un seul élément de la page. On le sélectionne avec un dièse : `#intro`.

5 La cascade

Quand plusieurs règles s'appliquent au même élément, la plus **spécifique** l'emporte, dans l'ordre croissant : une règle sur une balise (`p`), sur une classe (`.note`), sur un identifiant (`#menu`), puis un style écrit dans la balise. À spécificité égale, la dernière règle lue l'emporte. L'inspecteur du navigateur montre les règles appliquées et barre celles qui sont supplantées.

À retenir.

— CSS fixe l'apparence ; on le range de préférence dans une feuille externe.

— Une règle : `sélecteur { propriété: valeur; }`.

— `.classe` pour plusieurs éléments, `#id` pour un seul.

— La règle la plus spécifique l'emporte ; à égalité, la dernière.

IHM Web et protocole HTTP

Fiche de synthèse

1 Interface homme-machine

Une **interface homme-machine** (IHM) est tout ce qui permet à une personne de communiquer avec une machine : matériel et logiciel qui réagit aux gestes. Le navigateur Web en est l'exemple quotidien.

2 Le protocole HTTP

Définition 1 **(HTTP)** Un **protocole** est un ensemble de règles que deux machines suivent pour se comprendre. **HTTP** fixe la façon dont un **client**, le navigateur, et un **serveur Web**, comme Apache ou nginx, échangent des messages : une **requête** du client, puis une **réponse** du serveur.

Afficher http://www.exemple.fr/cours/http.html.

1. Le navigateur isole le nom de domaine `www.exemple.fr`.
2. Un serveur DNS lui donne l'adresse IP ; il ouvre une connexion TCP.
3. Il envoie une requête HTTP qui demande `/cours/http.html`.
4. Le serveur renvoie le fichier dans une réponse HTTP.
5. Le navigateur lit le code HTML reçu et construit la page.

3 La requête

```
GET /cours/http.html HTTP/1.1
Host: www.exemple.fr
User-Agent: Mozilla/5.0 ... Firefox/128.0
Accept: text/html
```

Première ligne : **méthode**, chemin, version. Puis des **champs** `Nom: valeur` : `Host` (domaine visé), `User-Agent` (navigateur), `Referer` (page d'où l'on vient).

Méthodes : `GET` demande une ressource (clic sur un lien), `POST` envoie des données à traiter, `PUT` dépose ou remplace une ressource, `DELETE` la supprime.

4 La réponse

```
HTTP/1.1 200 OK
Date: Tue, 13 Oct 2026 08:15:42 GMT
Server: nginx/1.26.2
Content-Type: text/html; charset=UTF-8
```

La **ligne de statut** donne un **code de réponse**. Après l'entête et une ligne vide vient le **corps**, le code HTML que le navigateur interprète. `Content-Type` donne le type de la ressource, `Location` l'adresse d'une redirection.

Code	Signification
200	la requête a réussi
301, 302	ressource déplacée, définitivement ou provisoirement
400	requête mal formée
401, 403	identification requise ; accès refusé
404	aucune ressource à cette adresse
500	erreur du serveur

Le premier chiffre donne la famille : 2 succès, 3 redirection, 4 erreur du client, 5 erreur du serveur.

5 Formulaires : GET et POST

La balise `<form>` envoie les champs saisis au programme désigné par son attribut `action`, avec la méthode de son attribut `method`. Chaque champ est transmis sous la forme `name=valeur`.

— `GET` : paramètres ajoutés à l'adresse, visibles (`/inscription?prenom=Leo&age=16`), pour une recherche.

— `POST` : paramètres dans le corps de la requête ; pour un envoi qui modifie des données. Pas chiffré pour autant : seul HTTPS protège.

Ordre : script JavaScript éventuel sur le **client**, envoi de la requête, programme du **serveur** qui fabrique la page, affichage. Un **cookie**, reçu par `Set-Cookie`, est mémorisé par le client et renvoyé au site à chaque requête (champ `Cookie`).

6 HTTPS

HTTP circule en clair. **HTTPS** chiffre les échanges : le serveur présente un **certificat**, délivré par un organisme de confiance et vérifié par le navigateur, puis client et serveur conviennent d'une clé de chiffrement. Le certificat protège contre l'attaque de l'intermédiaire ; le chiffrement rend illisibles les données interceptées. Avant de saisir un mot de passe, on vérifie que l'adresse commence par `https://`.

À retenir.

— HTTP : une requête du client, une réponse du serveur, en texte lisible.

— `GET` met les paramètres dans l'adresse, `POST` dans le corps de la requête.

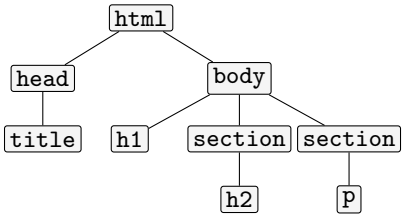
— Le code 200 signale un succès, 404 une ressource introuvable.

HTML DOM

Fiche de synthèse

1 L'arbre d'une page

Définition 1 **(DOM)** Le **DOM**, modèle objet de document, est l'**arbre** que le navigateur construit en mémoire à partir d'une page HTML : chaque balise, chaque texte et chaque attribut y devient un objet, un **nœud**. Un programme JavaScript lit et modifie la page à travers cet arbre.



Les nœuds sont des **éléments**, un par balise, des **textes**, écrits entre les balises, et des **attributs**, rattachés à leur élément. `html` est la **racine** ; `head` et `body` sont ses **enfants** ; `body` est le **parent** de `h1` ; les deux `section` sont **frères**.

2 JavaScript en bref

JavaScript est le langage des navigateurs. Un script s'écrit entre `<script>` et `</script>`, en fin de `body`, pour que les éléments qu'il manipule existent déjà.

Python	JavaScript
`x = 0`	`let x = 0;`
`TAUX = 20`	`const TAUX = 20;`
`True, False`	`true, false`
`a == b, a != b`	`a === b, a !== b`
`and, or, not`	`&&, ||, !`
`for x in liste:`	`for (const x of liste) {...}`
`def f(a):`	`function f(a) {...}`
`print(x)`	`console.log(x);`

Les blocs sont délimités par des accolades, chaque instruction finit par un point-virgule. `let` déclare une variable qui change, `const` une variable jamais réaffectée. `===` compare sans conversion : `"1" === 1` vaut `false`, alors que `"1" == 1` vaut `true`.

3 Atteindre et lire les éléments

- `document.getElementById("intro")` : l'élément d'identifiant `intro`.
- `document.querySelector("nav a")` : le premier élément qui correspond au sélecteur CSS.
- `document.querySelectorAll("p")` : tous ces éléments, parcourus avec `for ... of`.
- `elt.textContent`, le texte ; `elt.innerHTML`, le code HTML ; `elt.getAttribute("href")`, un attribut ; `elt.value`, le contenu d'un champ.
- `elt.children`, `elt.parentNode`, `elt.nextElementSibling` : les enfants, le parent, le frère suivant.

4 Modifier la page

```
<ul id="liste"><li>Pain</li></ul>
<input type="text" id="art">
<script>
  function ajouter() {
    const li = document.createElement("li");
    const c = document.getElementById("art");
    li.textContent = c.value;
    const ul = document.getElementById("liste");
    ul.appendChild(li);
  }
</script>
```

Toute modification s'affiche aussitôt. On remplace le texte par `textContent`, un attribut par `setAttribute`, le style par `style` ; `createElement` crée un élément, `appendChild` l'insère, `remove` le retire.

5 Réagir aux événements

```
<input type="text" id="prenom">
<p id="salut"></p>
<script>
  const c = document.getElementById("prenom");
  c.addEventListener("input", function (event) {
    const p = document.getElementById("salut");
    p.textContent = "Bonjour " + c.value;
  });
</script>
```

Un **événement** est un fait qui se produit dans la page ; un **gestionnaire** est la fonction que le navigateur exécute alors. Le paramètre `event` décrit l'événement ; `event.target` est l'élément concerné. Événements courants : `click`, `input`, `change`, `submit`, `keydown`, `mouseover`, `load`.

À retenir.

- Le DOM est l'arbre de la page ; `getElementById` et `querySelector` en atteignent les éléments.
- `addEventListener` associe une fonction à un événement.

JavaScript

Fiche de synthèse

① Le langage des navigateurs

JavaScript rend les pages Web dynamiques : il réagit aux actions de l'utilisateur et modifie la page à travers le DOM.

La même fonction en Python, puis en JavaScript :

```python
def ma_fonction():
    for i in range(5):
        print(i)
```

```javascript
function maFonction() {
  for (let i = 0; i < 5; i++) {
    console.log(i);
  }
}
```

— JavaScript délimite les blocs par des accolades ; l'indentation ne sert qu'à la lisibilité.

— La boucle `for (let i = 0; i < 5; i++)` donne à `i` les valeurs 0 à 4 ; `i++` ajoute 1 à `i`.

— Python nomme ses variables en `snake_case`, mots en minuscules séparés par des tirets bas ; JavaScript en `camelCase`, chaque mot après le premier commençant par une majuscule.

On range les scripts dans un fichier `script.js`, appelé dans l'en-tête :

```html
<script src="script.js" defer></script>
```

L'attribut `defer` retarde l'exécution du script jusqu'à la fin du chargement de la page, pour que ses éléments existent déjà.

② Les événements

Un **événement** est un fait qui se produit dans la page : clic, touche enfoncée, saisie, fin du chargement. Un **gestionnaire d'événement** est la fonction que le navigateur exécute alors. On l'attache à un élément avec `addEventListener` : le HTML ne contient plus de code.

```javascript
const aff = document.getElementById("valeur");
const plus = document.getElementById("plus");
plus.addEventListener("click", function () {
  aff.textContent = Number(aff.textContent) + 1;
});
```

Événements courants : `click`, `input`, `change`, `submit`, `keydown`, `load`. La fonction reçoit un objet `event` : `event.target` est l'élément concerné, `event.preventDefault()` annule l'action par défaut, comme l'envoi d'un formulaire.

③ Ce que JavaScript peut faire

Action	Moyen
changer un texte	`elt.innerHTML = "..."`
changer une image	`img.src = "pic_bulbon.gif"`
changer un style	`elt.style.color = "red"`
cacher, montrer	`elt.style.display = "none"` ou `"block"`
lire un formulaire	parcourir `formulaire.elements` et lire `champ.value`
lire une liste déroulante	parcourir `liste.options` et lire `option.text`
afficher un message	`alert("...")`

```javascript
function basculer() {
  const x = document.getElementById("maDIV");
  if (x.style.display === "none") {
    x.style.display = "block";
  } else {
    x.style.display = "none";
  }
}
```

Cette fonction cache un bloc s'il est visible et le montre s'il est caché.

À retenir.

— Accolades et points-virgules remplacent l'indentation de Python.

— Les scripts vont dans un fichier `.js`, chargé avec `defer`.

— `addEventListener` exécute une fonction quand un événement se produit.

— Le script modifie la page : texte, attributs, style, affichage.

Complexité

Fiche de synthèse

1 Trois questions

Devant un algorithme, on se demande s'il s'arrête, c'est la **terminaison** ; s'il donne le résultat attendu, c'est la **correction** ; et en combien de temps, c'est la **complexité**.

Définition 1 **(Complexité en temps)** La **complexité en temps** d'un algorithme mesure le nombre d'**opérations élémentaires** qu'il effectue, en fonction de la **taille** n des données. Une opération élémentaire, comparaison, affectation, opération arithmétique, accès à un élément, compte pour une unité. Cette mesure ne dépend pas de la vitesse de l'ordinateur.

Pour une même taille, le nombre d'opérations dépend des données : on distingue le **meilleur cas**, le **cas moyen** et le **pire cas**. On étudie surtout le pire cas, qui garantit que l'algorithme ne sera jamais plus lent. Chercher un élément dans une liste de n éléments demande une comparaison s'il est en tête, n s'il est à la fin ou absent.

2 La notation O

On ne garde qu'un **ordre de grandeur** : le terme qui croît le plus vite, sans son coefficient.

- $4n + 3$ opérations : $O(n)$.
- $2n^2 + 7n + 4$ opérations : $O(n^2)$.
- $n/3$ opérations : $O(n)$.

Classe	Nom	Exemple
$O(1)$	constante	accès à `L[i]`
$O(\log n)$	logarithmique	recherche dichotomique
$O(n)$	linéaire	parcours d'une liste
$O(n \log n)$	quasi linéaire	tri fusion, `sort`
$O(n^2)$	quadratique	tris par sélection et par insertion
$O(2^n)$	exponentielle	essayer tous les sous-ensembles
$O(n!)$	factorielle	essayer tous les ordres

$\log n$ est le logarithme de base 2 : le nombre de fois qu'on peut diviser n par 2 avant d'atteindre 1. $\log 1\,024 = 10$, et $\log$ d'un million vaut environ 20.

3 Conséquences pratiques

À un milliard d'opérations par seconde, pour un million de données, un algorithme en $O(n)$ met 1 ms, en $O(n \log n)$ 20 ms, en $O(n^2)$ environ 17 minutes. En $O(2^n)$, cent données suffisent à dépasser l'âge de l'Univers.

Quand la taille des données est multipliée par 10, le temps est multiplié par 10 pour une complexité linéaire, par 100 pour une complexité quadratique ; il augmente seulement d'une constante pour une complexité logarithmique.

4 Compter les opérations

```
def recherche(L, x):
    for i in range(len(L)):
        if L[i] == x:
            return True
    return False
```

Dans le pire cas, `x` est absent : la boucle fait n tours, avec un accès et une comparaison à chaque tour, soit $2n$ opérations. La complexité est en $O(n)$.

```
def rencontres(eq):
    matchs = []
    for i in range(len(eq)):
        for j in range(i):
            matchs.append((eq[j], eq[i]))
    return matchs
```

Pour chaque i, la boucle intérieure fait i tours : en tout,

$$0 + 1 + \cdots + (n-1) = \frac{n(n-1)}{2}.$$

Deux boucles imbriquées donnent ici une complexité en $O(n^2)$.

Déterminer une complexité. Compter le nombre de tours de chaque boucle dans le pire cas, multiplier pour des boucles imbriquées, puis ne garder que le terme dominant, sans coefficient.

À retenir.

- La complexité compte les opérations élémentaires en fonction de la taille n, dans le pire cas.
- On ne garde que le terme dominant : $2n^2 + 7n + 4$ est en $O(n^2)$.
- Une boucle sur n éléments donne $O(n)$, deux boucles imbriquées souvent $O(n^2)$.

Terminaison et correction

Fiche de synthèse

1 Deux questions

Un algorithme **termine** s'il s'arrête toujours ; il est **correct** s'il donne le résultat attendu. On le prouve avec un variant et un invariant de boucle.

```
def division(a: int, b: int):
    """a et b entiers positifs non nuls.
    Renvoie le quotient et le reste
    de la division de a par b."""
    q = 0
    r = a
    while r >= b:
        q = q + 1
        r = r - b
    return (q, r)
```

2 La trace

La **trace** d'un algorithme note la valeur de chaque variable à chaque étape. Pour $a = 22$ et $b = 3$:

Tour	0	1	2	3	4	5	6	7
q	0	1	2	3	4	5	6	7
r	22	19	16	13	10	7	4	1

Après le septième tour, $r = 1 < 3$: la boucle s'arrête, et $22 = 3 \times 7 + 1$.

3 Terminaison : le variant

Définition 1 **(Variant de boucle)** Un **variant de boucle** est une quantité entière, positive ou nulle à l'entrée de chaque tour, qui **décroît strictement** à chaque tour. Un entier positif ne peut pas décroître indéfiniment : la boucle s'arrête.

Pour la division, r est un variant : à l'entrée d'un tour, $r \geqslant b > 0$; et r devient $r - b$, strictement plus petit car $b > 0$.

Une boucle `for i in range(n)` se termine toujours : son nombre de tours est fixé avant d'y entrer.

4 Correction : l'invariant

Définition 2 **(Invariant de boucle)** Un **invariant de boucle** est une propriété vraie avant le premier tour, et qui reste vraie après chaque tour.

Pour la division, l'invariant est $a = b \times q + r$ avec $r \geqslant 0$.

— **Initialisation** : avant la boucle, $q = 0$ et $r = a$, donc $b \times 0 + a = a$, et $r = a \geqslant 0$.

— **Conservation** : si $a = bq + r$ au début d'un tour, alors après le tour $b(q + 1) + (r - b) = bq + r = a$; et $r - b \geqslant 0$, puisque le tour n'a lieu que si $r \geqslant b$.

— **Conclusion** : à la sortie, l'invariant est vrai et la condition d'arrêt donne $r < b$. Donc $a = bq + r$ avec $0 \leqslant r < b$: c'est la division euclidienne.

Prouver la correction d'une boucle.

1. Choisir un invariant.
2. Montrer qu'il est vrai avant la boucle, à l'aide des préconditions, les conditions que vérifient les entrées.
3. Montrer que, s'il est vrai au début d'un tour, il l'est encore à la fin.
4. À la sortie, combiner l'invariant et la négation de la condition du `while` pour obtenir la **postcondition**, la propriété voulue du résultat.

5 Deuxième exemple

```
def multiplication(a, b):
    resultat = 0
    compteur = 0
    while compteur < b:
        resultat = resultat + a
        compteur = compteur + 1
    return resultat
```

Le variant est b − compteur, qui diminue de 1 à chaque tour. L'invariant est resultat = $a \times$ compteur : vrai au départ, $0 = a \times 0$, et conservé, car $a \times \text{compteur} + a = a \times (\text{compteur} + 1)$. À la sortie, compteur = b, donc resultat = $a \times b$.

À retenir.

— Un variant, entier positif strictement décroissant, prouve qu'une boucle se termine.

— Un invariant, vrai avant la boucle et conservé à chaque tour, prouve la correction.

— En sortie, invariant et condition d'arrêt donnent le résultat attendu.

Spécification et tests

Fiche de synthèse

1 Prototype

Définition 1 **(Prototype)** Le **prototype** d'une fonction est la première ligne de sa définition : le nom de la fonction, ses paramètres avec leurs types, et le type de la valeur renvoyée.

```
def aire_rectangle(longueur: float,
                   largeur: float) -> float:
```

Cette fonction prend deux paramètres de type `float` et renvoie un `float`.

2 Docstring et doctests

Définition 2 **(Docstring)** La **docstring** est la chaîne de documentation placée juste après le prototype. Elle donne une description brève, les paramètres et leurs types, la valeur renvoyée, des exemples et, si besoin, les préconditions.

```
def aire_rectangle(longueur: float,
                   largeur: float) -> float:
    """Calcule l'aire d'un rectangle.

    Paramètres :
        longueur (float) : la longueur.
        largeur (float) : la largeur.
    Renvoie :
        float : l'aire du rectangle.
    Exemples :
        >>> aire_rectangle(10.0, 5.0)
        50.0
        >>> aire_rectangle(0.0, 10.0)
        0.0
    """
    assert longueur >= 0 and largeur >= 0
    return longueur * largeur
```

Un **doctest** est un exemple écrit dans la docstring : une ligne `>>>` suivie de l'appel, puis le résultat attendu. On les exécute en ajoutant à la fin du fichier :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

Avec `verbose=True`, chaque test est affiché, réussi ou non. On choisit des exemples variés : des cas courants et des cas limites, comme une longueur nulle.

3 Préconditions

Définition 3 **(Précondition)** Une **précondition** est une condition que les paramètres doivent vérifier pour que la fonction donne un résultat correct.

```
def racine_carree(x: float) -> float:
    """Renvoie la racine carrée de x."""
    assert x >= 0, "x doit être positif ou nul"
    return x ** 0.5
```

`assert condition, "message"` ne fait rien si la condition est vraie ; sinon, le programme s'arrête sur l'erreur `AssertionError`, suivie du message. `racine_carree(-4)` s'arrête ainsi. Pour une division, la précondition est un diviseur différent de 0.

4 Postconditions

Définition 4 **(Postcondition)** Une **postcondition** est une condition que vérifie le résultat quand les préconditions sont respectées : ce que la fonction garantit. Elle figure dans la section `Renvoie :` de la docstring.

```
def note_sur_20(points, total):
    assert total > 0 and 0 <= points <= total
    note = points / total * 20
    assert 0 <= note <= 20
    return note
```

La postcondition se vérifie avec `assert`, juste avant le `return` : une formule fausse, comme `total / points * 20`, arrête le programme dans la fonction elle-même.

5 Jeux de tests

Un **jeu de tests** est un ensemble d'appels dont on connaît le résultat attendu : des doctests, ou une instruction `assert` par appel.

```
assert aire_rectangle(7.0, 3.0) == 21.0
assert aire_rectangle(0.0, 10.0) == 0.0
```

La qualité des tests compte autant que leur nombre : cas courants et cas limites. Le succès d'un jeu de tests ne garantit pas la correction : `longueur * longueur` réussit le test d'une longueur nulle.

6 Bibliothèques

Une **bibliothèque**, ou module, regroupe des fonctions réutilisables, comme `math` ou `random` : inutile de les réécrire.

```
import math
print(math.sqrt(16))      # 4.0

from math import sqrt, pi
print(sqrt(25))           # 5.0

import math as m
print(m.sqrt(36))         # 6.0
```

`import math` importe tout le module, et l'on écrit `math.` devant chaque fonction ; `from math import sqrt` importe une seule fonction, écrite sans préfixe ; `import math as m` donne au module un nom plus court.

À retenir.

- Une fonction bien écrite a un prototype annoté et une docstring complète.
- Les doctests vérifient les exemples de la docstring ; on y met des cas courants et des cas limites.
- `assert` vérifie les préconditions au début de la fonction, et la postcondition avant le `return`.
- Un jeu de tests réussi ne prouve pas que la fonction est correcte.

Recherche dichotomique

Fiche de synthèse

1 Recherche séquentielle

```
def chercher(liste, x):
    for element in liste:
        if element == x:
            return True
    return False
```

La **recherche séquentielle** compare les éléments un par un. Elle fonctionne même si la liste n'est pas triée, mais, dans le pire cas, élément absent ou en dernière position, elle examine les n éléments : sa complexité est en $O(n)$.

2 Le principe de la dichotomie

Dans un dictionnaire, on ouvre vers le milieu, puis on continue dans la moitié où se trouve le mot. Pour deviner un nombre entre 1 et 100, on propose toujours le milieu de l'intervalle restant : 73 est trouvé en six essais, 50, 75, 62, 68, 71, puis 73.

Définition 1 **(Recherche dichotomique)** Dans une liste **triée**, la **recherche dichotomique** compare l'élément cherché à celui du milieu de la zone de recherche, puis ne garde que la moitié qui peut le contenir. À chaque étape, la zone est divisée par deux.

```
def recherche_dichotomique(liste, element):
    debut = 0
    fin = len(liste) - 1
    while debut <= fin:
        centre = (debut + fin) // 2
        if liste[centre] == element:
            return True
        elif liste[centre] < element:
            debut = centre + 1
        else:
            fin = centre - 1
    return False
```

Pour renvoyer la position de l'élément, on écrit `return centre` au lieu de `return True`, et `return None` à la fin s'il est absent.

3 Exemple

On cherche 14 dans `[1, 2, 5, 9, 10, 14, 17, 24, 41]`.

Étape	debut	fin	centre	Comparaison
1	0	8	4	$10 < 14$: à droite
2	5	8	6	$17 > 14$: à gauche
3	5	5	5	$14 = 14$: trouvé

Pour 7, absent, quatre étapes mènent à `debut = 3` et `fin = 2` : la condition `debut <= fin` est fausse, la fonction renvoie `False`.

4 Complexité

Après k étapes, il reste au plus $n/2^k$ éléments : il faut au plus $\lfloor \log_2 n \rfloor + 1$ étapes. $\log_2 n$ est l'exposant k tel que $2^k = n$; $\lfloor x \rfloor$ est la partie entière de x. La complexité est en $O(\log n)$, **logarithmique**.

n	8	16	64	128	10^6
Séquentielle	8	16	64	128	10^6
Dichotomie	4	5	7	8	20

5 Terminaison et condition

Le **variant** fin − debut + 1, le nombre d'éléments restant à examiner, est un entier positif qui diminue d'au moins 1 à chaque tour : la boucle se termine.

Attention. La recherche dichotomique exige une liste **triée** : sinon, elle peut déclarer absent un élément présent. On trie une fois, puis chaque recherche est très rapide.

À retenir.

- La recherche séquentielle est en $O(n)$, et fonctionne sur toute liste.
- La dichotomie divise la zone par deux à chaque étape : $O(\log n)$, 20 comparaisons pour un million d'éléments.
- Elle exige une liste triée.

Algorithmes de tri

Fiche de synthèse

1 Généralités

Un **algorithme de tri** range les éléments d'une liste du plus petit au plus grand, en les comparant deux à deux. Entiers et flottants se comparent par $\leqslant$; les chaînes de caractères par l'ordre lexicographique, où toutes les majuscules passent avant les minuscules : `'Z' < 'a'` vaut `True`.

Définition 1 **(Tri en place, tri stable)**

— Un tri est **en place** s'il travaille dans la liste elle-même, sans en créer une autre.

— Un tri est **stable** s'il garde dans leur ordre de départ les éléments égaux pour le critère de tri.

Des élèves rangés par ordre alphabétique, triés par note avec un tri stable, restent dans l'ordre alphabétique à note égale : on peut trier selon un critère, puis un autre, sans défaire le premier classement.

```
def echanger(L, i, j):
    L[i], L[j] = L[j], L[i]
    return L

def est_triee(L):
    for i in range(len(L) - 1):
        if L[i] > L[i + 1]:
            return False
    return True
```

2 Tri par sélection

On cherche le plus petit élément et on l'échange avec le premier ; puis on cherche le plus petit du reste et on l'échange avec le deuxième, et ainsi de suite jusqu'à l'avant-dernier.

départ	12	9	3	7	14	11
$i = 0$	**3**	9	12	7	14	11
$i = 1$	3	**7**	12	9	14	11
$i = 2$	3	7	**9**	12	14	11
$i = 3$	3	7	9	**11**	14	12
$i = 4$	3	7	9	11	**12**	14

```
def tri_selection(L):
    for i in range(len(L) - 1):
        indice_du_min = i
        for j in range(i + 1, len(L)):
            if L[j] < L[indice_du_min]:
                indice_du_min = j
        echanger(L, i, indice_du_min)
    return L
```

Invariant : après le tour i, les $i + 1$ premiers éléments sont triés et à leur place définitive. Le tri par sélection est en place, mais pas stable.

3 Tri par insertion

Comme on range des cartes à jouer : le début de la liste est trié, et l'on fait remonter l'élément suivant jusqu'à sa place parmi ceux qui le précèdent.

```
def tri_insertion(L):
    for i in range(1, len(L)):
        en_cours = L[i]
        j = i
        while j > 0 and L[j - 1] > en_cours:
            L[j] = L[j - 1]
            j -= 1
        L[j] = en_cours
    return L
```

Pour `[12, 9, 3, 7, 14, 11]`, la partie triée devient `[9, 12]`, puis `[3, 9, 12]`, `[3, 7, 9, 12]`, `[3, 7, 9, 12, 14]` et enfin `[3, 7, 9, 11, 12, 14]`. Invariant : après le tour i, les $i + 1$ premiers éléments sont triés, pas forcément à leur place définitive. Le tri par insertion est en place et stable.

4 Complexité

Le tri par sélection compare toujours $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$ fois. Le tri par insertion ne fait que $n-1$ comparaisons sur une liste déjà triée, mais $\frac{n(n-1)}{2}$ décalages sur une liste triée à l'envers.

Tri	Meilleur	Moyen	Pire	Stable	En place
sélection	$O(n^2)$	$O(n^2)$	$O(n^2)$	non	oui
insertion	$O(n)$	$O(n^2)$	$O(n^2)$	oui	oui

À retenir.

— Sélection : placer à chaque tour le plus petit des éléments restants.

— Insertion : insérer chaque élément à sa place dans le début déjà trié.

— Les deux sont quadratiques ; l'insertion est linéaire sur une liste déjà triée.

Réseaux

Fiche de synthèse

1 Réseaux et Internet

Définition 1 **(Réseau)** Un **réseau informatique** relie des machines pour qu'elles communiquent et partagent des ressources. Elles doivent être connectées, par câble, fibre ou Wi-Fi, et suivre les mêmes règles de communication, un **protocole**.

Sigle	Échelle	Exemple
PAN	personnelle, quelques mètres	Bluetooth
LAN	locale, un bâtiment	Wi-Fi du lycée
MAN	une ville	réseau d'une université
WAN	un pays, un continent	réseau d'un opérateur

Internet, de *inter-network*, est le réseau mondial : l'interconnexion de millions de réseaux qui parlent les mêmes protocoles, TCP/IP.

2 Adresses

— Une **adresse IPv4** identifie une machine sur un réseau. Elle compte 32 bits, écrits en quatre octets de 0 à 255 : `192.168.1.42`. Il en existe 2^{32}, environ 4,3 milliards.

— **IPv6** utilise 128 bits, écrits en hexadécimal : `2001:0db8:85a3::8a2e:0370:7334`.

— Une adresse **publique**, unique au monde, est donnée par le fournisseur d'accès ; une adresse **privée** sert dans un réseau local : `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`.

— L'**adresse MAC**, sur 48 bits, identifie physiquement une carte réseau : `4C:ED:FB:A2:33:7E`. L'adresse IP change avec le réseau, comme une adresse postale ; l'adresse MAC, non. Les smartphones utilisent aujourd'hui une adresse MAC aléatoire par réseau Wi-Fi, pour éviter d'être pistés.

3 Masque et adresse réseau

Une adresse IP a une partie **réseau**, commune aux machines d'un même réseau local, et une partie **hôte**, propre à chaque machine. Le **masque** indique la partie réseau par ses bits à 1. La notation **CIDR** `/n` donne le nombre de ces bits : `/24` correspond à `255.255.255.0`.

Calculer l'adresse réseau. Faire un ET logique bit à bit entre l'adresse et le masque.

```
192.168.1.42   11000000.10101000.00000001.00101010
255.255.255.0  11111111.11111111.11111111.00000000
192.168.1.0    11000000.10101000.00000001.00000000
```

Deux machines sont sur le même réseau local si elles ont la même adresse réseau : `192.168.1.42` et `192.168.1.99` communiquent directement, `192.168.2.10` passe par un routeur.

Dans un réseau, deux adresses ne sont données à aucune machine : l'adresse réseau, partie hôte à 0, et l'adresse de diffusion, partie hôte à 1. Un réseau `/24` accueille donc 254 machines.

4 Sous-réseaux

Le réseau `172.18.0.0/22` compte 1 024 adresses. Passer à un masque `/24` le découpe en $2^{24-22} = 4$ sous-réseaux de 256 adresses : élèves, enseignants, imprimantes, serveurs. Le découpage organise, sécurise, car le routeur filtre les échanges entre sous-réseaux, et limite la diffusion des messages.

5 Switch et routeur

— Un **switch**, ou commutateur, relie les machines d'un même réseau local, par leurs adresses MAC.

— Un **routeur** relie plusieurs réseaux : il consulte sa **table de routage** pour choisir par où envoyer chaque paquet, d'après son adresse IP.

Un paquet saute de routeur en routeur jusqu'à sa destination ; chaque passage est un **saut**, que la commande `traceroute` affiche. La box Internet est à la fois routeur et switch.

À retenir.

— Internet interconnecte des millions de réseaux grâce aux protocoles TCP/IP.
— IPv4 : 32 bits ; MAC : 48 bits, propre à la carte réseau.
— Adresse réseau : adresse ET masque ; même adresse réseau, même réseau local.
— Le switch relie un réseau local, le routeur relie des réseaux.

Le modèle TCP/IP

Fiche de synthèse

1 Un modèle en couches

Faire communiquer deux machines pose plusieurs problèmes : transmettre les bits, trouver le destinataire à travers plusieurs réseaux, vérifier que rien n'est perdu, comprendre le message. On les répartit en **couches** : chaque couche résout un problème et s'appuie sur la couche du dessous. Une couche peut changer sans toucher aux autres : on passe du Wi-Fi à Ethernet sans changer la façon de demander une page.

Couche	Rôle	Protocoles
4. Application	ce que veut faire le logiciel	HTTP, DNS, SMTP
3. Transport	acheminement de bout en bout, fiable ou non	TCP, UDP
2. Internet	adressage et routage entre réseaux	IP
1. Accès réseau	transmission sur le support	Ethernet, Wi-Fi

2 TCP et UDP

— **TCP** garantit que les données arrivent **dans l'ordre et sans perte** : il découpe le message en morceaux numérotés, attend un accusé de réception pour chacun et retransmet ce qui est perdu. Le Web et le courrier électronique l'utilisent.

— **UDP** envoie sans vérification : plus rapide, il accepte de perdre des données. Appels vidéo et jeux en ligne l'utilisent.

— Un **numéro de port** désigne le programme destinataire sur la machine : 80 pour HTTP, 443 pour HTTPS, 53 pour DNS.

3 Le protocole du bit alterné

Protocole du bit alterné.

1. L'émetteur envoie un paquet avec un bit de contrôle, 0 pour le premier.
2. Le récepteur renvoie un **accusé de réception** (ACK) portant le même bit.
3. À sa réception, l'émetteur envoie le paquet suivant avec le bit inversé.
4. Sans accusé avant la fin d'un délai, fixé par un **temporisateur**, l'émetteur renvoie le même paquet, avec le même bit.

Un récepteur qui reçoit deux fois de suite le même bit sait que c'est un doublon, son accusé s'étant perdu : il l'ignore et renvoie l'accusé. Chaque paquet arrive une fois et dans l'ordre ; en contrepartie, attendre chaque accusé limite le débit.

Émetteur	Réseau	Récepteur
envoie A, bit 0	→	reçoit A ; ACK 0
reçoit ACK 0 ; envoie B, bit 1	perdu	
délai écoulé ; renvoie B, bit 1	→	reçoit B ; ACK 1
reçoit ACK 1 ; envoie C, bit 0	→	reçoit C ; ACK 0

4 L'encapsulation

À l'envoi, chaque couche ajoute son **en-tête** autour de ce que lui donne la couche du dessus : c'est l'**encapsulation**, comme une lettre dans une enveloppe, dans un carton, dans un conteneur. À l'arrivée, la **désencapsulation** retire les en-têtes un à un.

Couche	En-tête	Résultat	Repère
Application	HTTP	données	nom de domaine
Transport	TCP ou UDP	segment	port
Internet	IP	paquet	adresse IP
Accès réseau	Ethernet	trame	adresse MAC

5 DNS, URL et HTTP

— Le **DNS** traduit un nom de domaine, `google.com`, en adresse IP : c'est l'annuaire d'Internet.

— Une **URL** donne l'adresse complète d'une ressource : protocole, nom de domaine, chemin, et éventuellement des paramètres.

— **HTTP** suit le schéma requête puis réponse : `GET` demande une page, `POST` envoie des données ; la réponse contient un code, 200 pour un succès, 404 pour une page introuvable. **HTTPS** chiffre la communication.

À retenir.

— Quatre couches : application, transport, internet, accès réseau.
— TCP est fiable, UDP est rapide sans garantie.
— Chaque couche ajoute un en-tête : segment, paquet, trame.
— Le DNS traduit les noms de domaine en adresses IP.

Systèmes d'exploitation

Fiche de synthèse

① Rôle du système d'exploitation

Définition 1 **(Système d'exploitation)** Un **système d'exploitation** est un programme chargé en mémoire au démarrage et présent jusqu'à l'extinction. Il sert d'intermédiaire entre le matériel et les applications.

- **Programmes** : il lance les applications et partage le temps du processeur ; en basculant très vite de l'une à l'autre, il donne l'illusion qu'elles s'exécutent en même temps, c'est le **multitâche**.
- **Mémoire** : il répartit la mémoire vive et empêche un programme d'accéder à celle d'un autre.
- **Fichiers** : il range les données en dossiers et fichiers et contrôle les droits d'accès.
- **Périphériques** : il communique avec le matériel grâce à des **pilotes**.

② Libre ou propriétaire

Windows et macOS sont **propriétaires** : leur code source n'est pas public, on ne peut ni l'étudier, ni le modifier, ni le redistribuer. **GNU/Linux** est **libre** : utiliser, étudier, modifier et redistribuer son code est permis. Cette démarche vient du projet GNU de Richard Stallman (1983) et du noyau de Linus Torvalds (1991). Linux domine les serveurs, les supercalculateurs, les systèmes embarqués et, avec Android, les smartphones.

③ Le shell

Un **shell** permet de communiquer avec le système : interface graphique, avec fenêtres et icônes, ou interface en ligne de commande, dans un **terminal**. **Bash** est le shell standard de GNU/Linux. L'invite `utilisateur@machine:~$` indique l'utilisateur, la machine et le répertoire courant ; une commande s'écrit `commande -options arguments`.

④ L'arborescence

Tous les fichiers forment une **arborescence** issue d'une **racine** unique, notée `/`. On y trouve `/home`, les dossiers personnels, `/bin`, les commandes de base, `/etc`, la configuration, `/tmp`, les fichiers temporaires. Un fichier dont le nom commence par un point est caché.

- Un **chemin absolu** part de la racine : `/home/elie/documents/tutu.txt`.
- Un **chemin relatif** part du répertoire courant : `photos/famille/toto.txt`, ou `../documents/notes.txt` pour passer par le dossier parent.
- `~` désigne le dossier personnel, `.` le répertoire courant, `..` le répertoire parent.

⑤ Commandes Bash

Commande	Rôle
`pwd`	affiche le répertoire courant
`ls`, `ls -l`	liste le contenu, avec les détails
`cd chemin`, `cd ..`	change de répertoire, remonte au parent
`cat fichier`	affiche un fichier
`touch fichier`	crée un fichier vide
`mkdir dossier`	crée un dossier
`cp source destination`	copie
`mv source destination`	déplace ou renomme
`rm fichier`	supprime définitivement
`man commande`	affiche le manuel
`grep motif fichier`	cherche un texte dans un fichier
`sudo commande`	exécute en administrateur

⑥ Les droits d'accès

Chaque fichier a trois droits, **lire** (`r`), **écrire** (`w`) et **exécuter** (`x`), fixés pour trois catégories : le **propriétaire**, son **groupe** et les **autres**. `ls -l` les affiche :

`-`	`rwx`	`r-x`	`r--`
type	propriétaire	groupe	autres

Ici, le propriétaire peut tout faire, le groupe lire et exécuter, les autres seulement lire ; un tiret marque un droit refusé, `d` en tête un dossier. `chmod g+w fichier` ajoute l'écriture au groupe, `chmod o-r fichier` retire la lecture aux autres.

À retenir.

- Le système d'exploitation gère programmes, mémoire, fichiers et périphériques.
- Un chemin absolu part de `/`, un chemin relatif du répertoire courant.
- Droits `r`, `w`, `x` pour le propriétaire, le groupe et les autres.

Périphériques et IHM

Fiche de synthèse

1 Périphériques

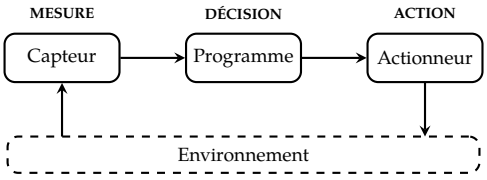
Un **périphérique d'entrée** fait entrer une information dans l'ordinateur (clavier, souris, micro), un **périphérique de sortie** la fait sortir (écran, imprimante, haut-parleur). Un écran tactile fait les deux.

2 Capteurs et actionneurs

Définition 1 **(Capteur)** Un **capteur** mesure une **grandeur physique** (température, lumière, distance, présence, appui) et la transmet au programme sous forme de **nombre**. C'est un périphérique d'entrée.

Définition 2 **(Actionneur)** Un **actionneur** transforme une **commande** du programme en **action physique** : mouvement (moteur), lumière (lampe), son (haut-parleur), chaleur (résistance), passage d'eau (électrovanne). C'est un périphérique de sortie.

3 Mesure, décision, action



L'action modifie la grandeur mesurée : la chaîne se répète **sans fin**. Elle tourne dans un **système embarqué** (ordinateur intégré à un appareil), un **objet connecté** (relié à un réseau) ou un **robot**.

Capteur	Décision	Actionneur
Sonde de température	$t <$ consigne ?	Radiateur
Luminosité, mouvement	sombre et présence ?	Lampes
Distance	obstacle proche ?	Moteurs

4 La chaîne en Python

On simule capteur et actionneur par des fonctions ; la décision est une fonction testable. Le programme principal répète mesure, décision, action dans une boucle.

```
import random

def lire_temperature():
    return random.randint(15, 25)

def decider(temperature, consigne):
    return temperature < consigne

assert decider(17, 19) == True
assert decider(19, 19) == False
```

5 Interface homme-machine

Définition 3 **(IHM)** L'**IHM** est l'ensemble des moyens par lesquels un utilisateur et une machine échangent des informations : périphériques et programme. On la conçoit à partir d'un **cahier des charges** : ce qu'elle affiche, les actions possibles, les réponses du programme.

En **programmation événementielle**, le programme crée ses composants, associe une fonction à chaque action, puis attend dans la **boucle d'événements** : un clic est un **événement**, qui appelle la fonction associée.

```
import tkinter as tk

def basculer():
    if etat.cget("text") == "Éteinte":
        etat.configure(text="Allumée")
    else:
        etat.configure(text="Éteinte")

fenetre = tk.Tk()
etat = tk.Label(fenetre, text="Éteinte")
etat.pack()
bouton = tk.Button(fenetre, text="Lampe",
                   command=basculer)
bouton.pack()
fenetre.mainloop()
```

`Label` affiche (sortie), `Button` et `Scale` (curseur) font entrer une action ; `configure` modifie un texte, `get` lit un curseur. On écrit `command=basculer` **sans parenthèses**.

À retenir.

- Capteur : grandeur physique vers nombre ; actionneur : commande vers action physique.
- Un système réactif répète sans fin la chaîne mesure, décision, action.
- La décision s'écrit dans une fonction, testée avec `assert`.
- Une IHM répond à un cahier des charges : composants d'entrée et de sortie, fonctions appelées lors des événements.

Algorithmes gloutons

Fiche de synthèse

① Problèmes d'optimisation

Un problème d'**optimisation** cherche la meilleure solution parmi de nombreuses possibilités : rendre la monnaie avec le moins de pièces, remplir un sac de la plus grande valeur. Essayer toutes les solutions est vite impossible : avec n objets, il y a 2^n choix, plus d'un milliard pour 30 objets.

Définition 1 **(Algorithme glouton)** Un **algorithme glouton** fait à chaque étape le choix qui semble le meilleur sur le moment, le **choix localement optimal**, sans jamais revenir en arrière.

Schéma d'un algorithme glouton.

1. Trier les éléments selon un critère.
2. Les parcourir dans cet ordre.
3. Prendre chaque élément s'il respecte les contraintes.
4. Ne jamais revenir sur un choix.

Un algorithme glouton est simple et rapide, souvent en $O(n \log n)$ à cause du tri ; mais son résultat n'est pas toujours optimal.

② Le rendu de monnaie

```
def rendu_monnaie(montant, pieces):
    pieces_triees = sorted(pieces, reverse=True)
    resultat = []
    reste = montant
    for piece in pieces_triees:
        while reste >= piece:
            resultat.append(piece)
            reste -= piece
    return resultat

systeme = [200, 100, 50, 20, 10, 5, 2, 1]
rendu_monnaie(678, systeme)
# [200, 200, 200, 50, 20, 5, 2, 1]
```

On prend à chaque fois la plus grosse pièce possible. Avec les pièces en euros, le résultat est toujours optimal. Avec un système `[6, 4, 1]` et 8 centimes, le glouton rend $6+1+1$, trois pièces, alors que $4+4$ n'en demande que deux : le meilleur choix local ne mène pas au meilleur résultat global.

③ Le sac à dos

Un sac supporte 15 kg ; on veut emporter la plus grande valeur.

Objet	1	2	3	4	5	6	7
Valeur	500	1000	1500	700	800	900	400
Poids	1	3	5	4	1	2	2

Critère de tri	**Valeur emportée**
valeurs décroissantes	4 900
poids croissants	4 300
rapport valeur/poids décroissant	5 100

Le rapport valeur/poids donne ici le meilleur résultat.

— **Sac à dos fractionnaire**, où l'on peut prendre une partie d'un objet : le glouton par rapport valeur/poids est toujours optimal.

— **Sac à dos 0/1**, où chaque objet est pris en entier ou pas du tout : il ne l'est plus. Avec A (165, 3 kg), B (100, 2 kg) et C (100, 2 kg) et un sac de 4 kg, le glouton prend A seul, 165 ; B et C valent 200. La programmation dynamique, vue en terminale, résout ce problème.

```
def rapport(o):
    return o[1] / o[2]

def sac_a_dos_01(poids_max, objets):
    tries = sorted(objets, key=rapport,
                   reverse=True)
    poids_actuel = 0
    sac = []
    for nom, valeur, poids in tries:
        if poids_actuel + poids <= poids_max:
            sac.append(nom)
            poids_actuel += poids
    return sac
```

Chaque objet est un tuple (nom, valeur, poids) ; `sorted` les range par rapport valeur/poids décroissant.

À retenir.

— Un glouton fait à chaque étape le meilleur choix local, sans revenir en arrière.

— Il est simple et rapide, mais pas toujours optimal.

— Rendu de monnaie en euros et sac à dos fractionnaire : optimal ; sac à dos 0/1 : pas toujours.

Intelligence artificielle

Fiche de synthèse

1 Apprentissage automatique

Un programme classique résout un problème dont on connaît la méthode : calculer une moyenne. Pour reconnaître un chiffre écrit à la main, on ne sait pas écrire cette méthode ; mais on dispose de très nombreux exemples.

Définition 1 **(Apprentissage automatique)** L'**apprentissage automatique** consiste à écrire un algorithme qui, à partir de données et d'expériences passées, apprend à résoudre au mieux un problème.

2 Modes et types de problèmes

— **Apprentissage supervisé** : les exemples sont étiquetés par des humains, « chat » ou « chien » ; le nombre de classes est connu.

— **Apprentissage non supervisé** : l'algorithme trouve lui-même des groupes de données semblables, c'est le **partitionnement** (*clustering*).

— Un problème de **classification** donne une valeur parmi quelques possibilités, chat ou chien ; un problème de **régression** donne un nombre réel, comme le prix d'un logement d'après son quartier et sa surface.

3 Entraîner et tester

Entraîner un modèle, c'est ajuster ses paramètres pour que ses prédictions se trompent le moins possible sur les exemples. On ne peut pas juger le modèle sur ces mêmes exemples : il les a en quelque sorte appris par cœur.

— Les **données d'entraînement**, environ 80 % des données, servent à ajuster le modèle.

— Les **données de test**, les 20 % restants, ne sont jamais montrées pendant l'entraînement ; le taux d'erreur sur ces données mesure honnêtement la qualité du modèle. On ne modifie plus le modèle après l'avoir testé.

Algorithmes supervisés : régression linéaire, arbres de décision, k plus proches voisins. Algorithme non supervisé : les k-moyennes, qui regroupent les données proches selon une distance.

4 Les k plus proches voisins

Algorithme des k plus proches voisins.

1. Choisir un nombre k de voisins et une **distance** entre les données.
2. Calculer la distance entre la nouvelle donnée et chaque donnée étiquetée.
3. Garder les k données les plus proches.
4. Attribuer à la nouvelle donnée la classe la plus représentée parmi elles : les voisins « votent ».

Avec $k = 5$, si trois voisins sont des chats et deux des chiens, la nouvelle image est classée « chat » ; avec $k = 3$, le résultat peut changer. Le réglage consiste à choisir les caractéristiques pertinentes, une bonne valeur de k, ni trop petite ni trop grande, et une distance adaptée : euclidienne, de Manhattan.

5 Usages et questions

— **Apports** : aide au diagnostic médical, traduction, recommandations, prévision des embouteillages, assistants conversationnels, voitures autonomes.

— **Questions** : décisions automatisées en justice ou en police, biais des données, confidentialité, droit d'auteur, consommation d'énergie de l'entraînement.

Un **modèle de langage**, comme ChatGPT, Claude, Gemini ou Mistral, est un réseau de neurones entraîné sur des milliards de textes à prédire le mot suivant. Il **génère** une suite plausible sans consulter de base de faits : il peut affirmer avec assurance une chose fausse, c'est une **hallucination**. Ses connaissances s'arrêtent à ses données d'entraînement, sauf accès à des documents ou à la recherche en ligne.

À retenir.

— L'apprentissage automatique apprend à partir d'exemples, supervisés ou non.

— Classification : une classe ; régression : un nombre.

— On sépare les données en entraînement, 80 %, et test, 20 %.

— Les k plus proches voisins font voter les k exemples les plus proches.