FORMATION PYTHON - LES BASES

Python

Apprendre a programmer clairement, pas a pas.

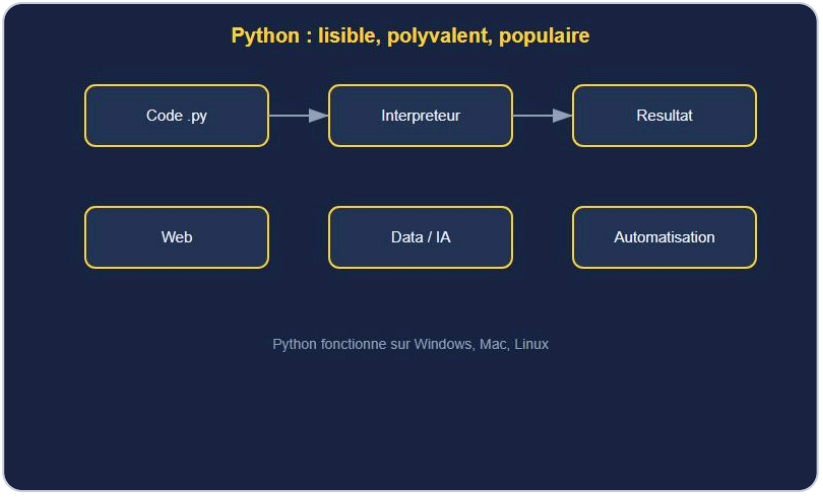
DanielCraft - Livre debutant - Pack Python

Sommaire

Clique un chapitre ou un sous-chapitre pour y aller.

CHAPITRE 1

C'est quoi Python ?



Python : lisible, polyvalent, populaire.

Le langage en une phrase

Python est un langage de programmation lisible, polyvalent et tres populaire. Il sert a creer des sites, des scripts, de l'IA, de l'automatisation et bien plus. Sa syntaxe claire le rend ideal pour debuter.

Pourquoi apprendre Python ?

Python est le langage le plus enseigne dans le monde. Il est utilise par des geants comme Google, Netflix et Instagram. Les offres d'emploi mentionnant Python explosent chaque annee.

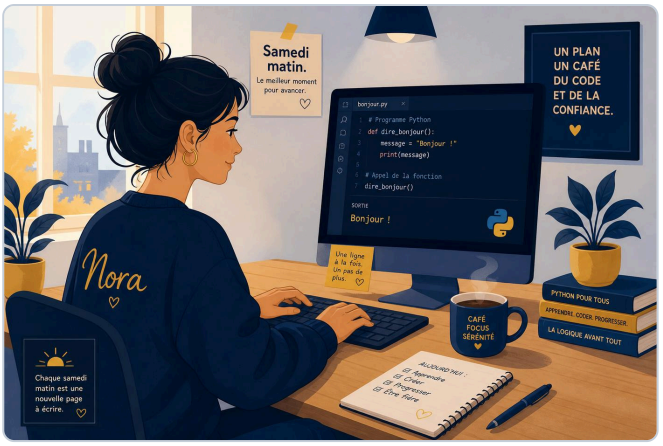
> **Astuce DanielCraft** - Tu n'as pas besoin d'etre bon en maths. Il suffit de savoir lire et taper du texte.

Nora decouvre Python un samedi matin. Elle installe l'interpreteur, tape `print("Bonjour")` et voit le resultat immediatement. Pas de compilation, pas d'attente.

A quoi ca ressemble ?

```
nom = "Lea"
print(f"Salut {nom}, bienvenue en Python !")
```

Ce code affiche : `Salut Lea, bienvenue en Python !`



Nora decouvre Python un samedi matin.

Petite histoire

Guido van Rossum cree Python en 1991 aux Pays-Bas. Il choisit le nom en reference aux Monty Python, pas au serpent. Depuis, Python a conquis la data science, le web et l'embarque.

En vrai

Sam utilise Python pour renommer 500 fichiers en 3 lignes. Max l'utilise pour analyser ses depenses mensuelles. Nora cree un petit bot Discord qui repond a ses amis.

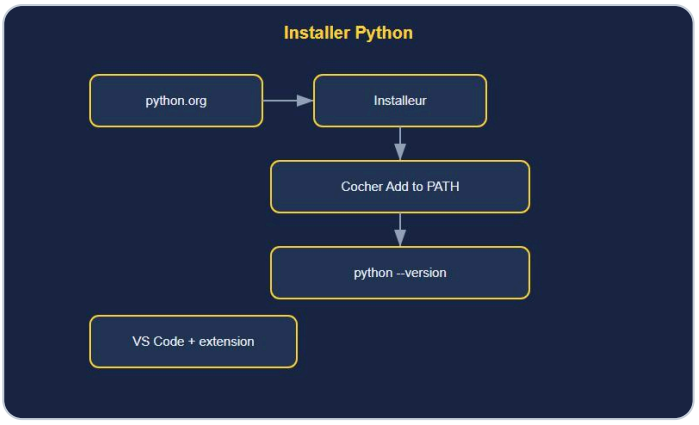
> **Piege** - Python est sensible a l'indentation. Un espace en trop provoque une erreur. On s'y habitue vite.

A retenir

- Python est lisible, polyvalent, gratuit.
- Il fonctionne sur Windows, Mac, Linux.
- On ecrit du code, on l'execute, on voit le resultat.

CHAPITRE 2

Installer Python



Telecharge, installe, verifie.

Ce qu'il te faut

Pour coder en Python tu as besoin de deux choses : l'interpreteur Python (le programme qui execute ton code) et un editeur de texte. On recommande VS Code, gratuit et leger.

Installer sur Windows

1. 1. Va sur python.org, telecharge la derniere version stable.
2. 2. Coche la case "Add Python to PATH" avant de cliquer Install.
3. 3. Ouvre un terminal (cmd ou PowerShell) et tape `python --version`.
4. 4. Si tu vois un numero de version, c'est bon.

> **Astuce DanielCraft** - La case PATH est cruciale. Sans elle, ton terminal ne trouvera pas Python.

Installer sur Mac

Mac a souvent Python pre-installe, mais en version ancienne. Installe la version recente via python.org ou Homebrew (`brew install python`).

Installer sur Linux

La plupart des distributions incluent Python. Tape `python3 --version`. Si absent : `sudo apt install python3`.

Verifier l'installation

```
python --version
```

Tu dois voir quelque chose comme `Python 3.12.x`.

Installer VS Code

Telecharge VS Code sur code.visualstudio.com. Installe l'extension "Python" de Microsoft. Elle apporte la coloration, l'auto-completion et le lancement facile.

Petite histoire

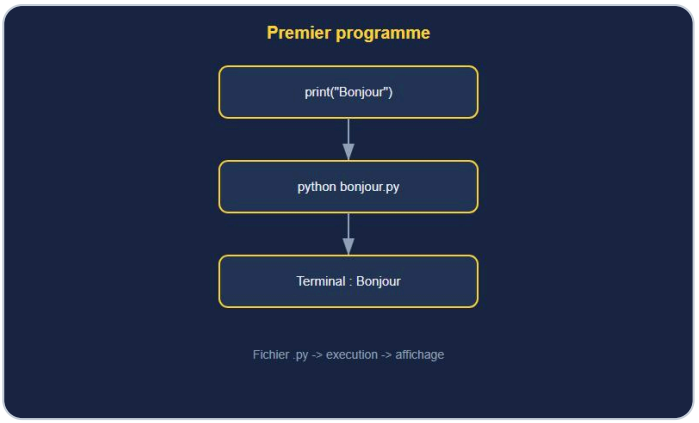
Max installe Python en 4 minutes. Il oublie la case PATH, son terminal dit "python n'est pas reconnu". Il reprend l'installeur, coche la case, et tout fonctionne.

A retenir

- Telecharge Python sur python.org.
- Coche "Add to PATH".
- Verifie avec `python --version`.
- Installe VS Code + extension Python.

CHAPITRE 3

Premier programme



print() : ta premiere ligne de code.

Le classique : Hello World

Chaque langage commence par afficher un message. En Python c'est une seule ligne :

```
print("Bonjour le monde !")
```

Cree un fichier `bonjour.py`, colle cette ligne, puis execute avec `python bonjour.py`.

Comment ca marche ?

`print()` est une fonction integree. Elle affiche ce qu'on lui donne entre parentheses. Les guillemets entourent du texte (une chaine de caracteres).

> **Astuce DanielCraft** - Donne un nom explicite a tes fichiers. `bonjour.py` est mieux que `test1.py`.

Afficher plusieurs lignes

```
print("Ligne 1")
print("Ligne 2")
print("Python, c'est simple.")
```

Python execute les lignes de haut en bas, dans l'ordre.

Les commentaires

```
# Ceci est un commentaire, Python l'ignore
print("Seul ceci est execute")
```

Les commentaires commencent par `#`. Ils servent a expliquer le code pour toi ou tes collegues.

Petite histoire

Nora cree `premier.py`. Elle tape `print("Ca marche !")` et lance le fichier. Le terminal affiche le message. Elle sourit : c'est son premier programme.

Erreur classique

```
print("Oups"
```

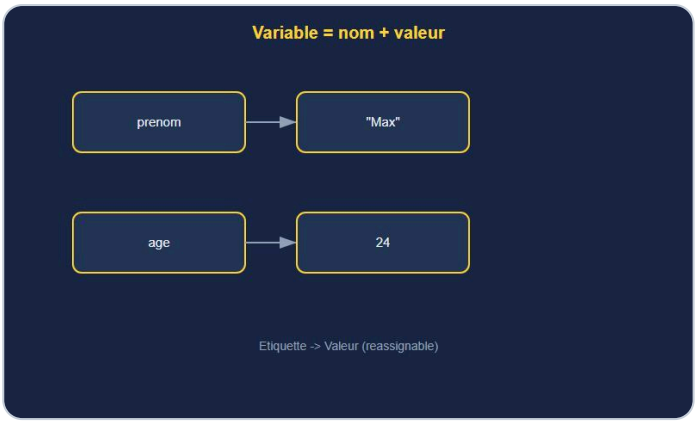
Il manque la parenthese fermante. Python affiche `SyntaxError`. Lis le message, corrige, relance.

A retenir

- `print()` affiche du texte.
- Un fichier `.py` s'execute avec `python fichier.py`.
- Les commentaires commencent par `#`.

CHAPITRE 4

Les variables



Variable = etiquette sur une boite.

C'est quoi une variable ?

Une variable est un nom qui pointe vers une valeur. Imagine une etiquette collee sur une boite : l'etiquette est le nom, le contenu de la boite est la valeur.

```
prenom = "Max"
age = 24
```

Ici `prenom` contient le texte "Max" et `age` contient le nombre 24.

Regles de nommage

- Commence par une lettre ou un underscore.
- Pas d'espaces : utilise des underscores (`mon_score`).
- Pas de mots reserves (`if`, `for`, `class`...).
- Sensible a la casse : `nom` et `Nom` sont differents.

> **Astuce DanielCraft** - Choisis des noms explicites. `prix_total` est mieux que `x`.

Modifier une variable

```
score = 0
score = score + 10
print(score)  # 10
```

On peut reassigner une variable a tout moment.

Afficher avec f-string

```
ville = "Lyon"
print(f"Je vis a {ville}")
```

Les f-strings permettent d'inserer des variables dans du texte.

Petite histoire

Sam cree un programme qui calcule son budget. Il stocke `salaire = 1800` et `loyer = 650`, puis affiche `reste = salaire - loyer`. Python lui repond 1150.

Erreur classique

```
print(prnom)  # NameError : faute de frappe
```

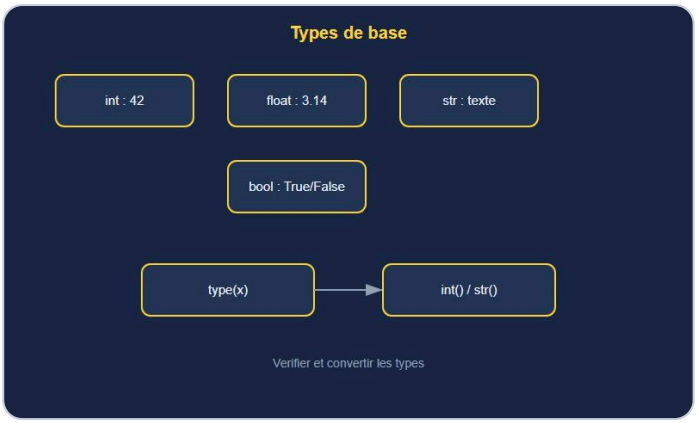
Python ne devine pas. Si le nom est mal ecrit, il dit `NameError`.

A retenir

- Variable = nom + valeur.
- Noms explicites, sans espaces, sensibles a la casse.
- f-string pour afficher des variables dans du texte.

CHAPITRE 5

Les types de donnees



int, float, str, bool : les types de base.

Les types de base

Python a plusieurs types principaux :

Type	Exemple	Usage
`int`	`42`	Nombres entiers
`float`	`3.14`	Nombres decimaux
`str`	`"Bonjour"`	Texte
`bool`	`True` / `False`	Vrai ou faux

Verifier un type

```
age = 25
print(type(age))  # <class 'int'>
```

La fonction `type()` te dit quel type a une valeur.

Conversion de types

```
texte = "42"
nombre = int(texte)  # Convertit la chaine en entier
print(nombre + 8)    # 50
```

> **Astuce DanielCraft** - `input()` retourne toujours une chaine. Convertis avec `int()` ou `float()` si tu veux un nombre.

Operations sur les nombres

```
a = 10
b = 3
print(a + b)   # 13
print(a - b)   # 7
print(a * b)   # 30
print(a / b)   # 3.333...
print(a // b)  # 3 (division entiere)
print(a % b)   # 1 (reste)
print(a ** b)  # 1000 (puissance)
```

Operations sur les chaines

```
nom = "Lea"
print(nom.upper())    # LEA
print(nom.lower())    # lea
print(len(nom))       # 3
print("a" in nom)     # False (sensible casse: 'a' pas dans 'Lea')
```

Petite histoire

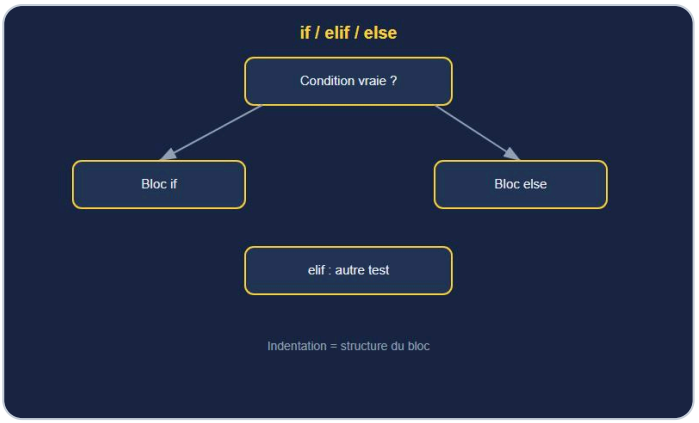
Nora demande l'age de l'utilisateur avec `input()`. Elle oublie de convertir et fait `age + 1`. Python concatene au lieu d'additionner. Elle ajoute `int()` et tout fonctionne.

A retenir

- `int`, `float`, `str`, `bool` sont les types de base.
- `type()` pour verifier, `int()` / `str()` pour convertir.
- `input()` retourne toujours une chaine.

CHAPITRE 6

Les conditions



if / elif / else : choisir un chemin.

Prendre une decision

Un programme doit souvent choisir : si une condition est vraie, faire ceci, sinon faire cela.

```
age = 17
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
```

if / elif / else

```
note = 14
if note >= 16:
    print("Tres bien")
elif note >= 12:
    print("Bien")
elif note >= 10:
    print("Passable")
else:
    print("Insuffisant")
```

Python teste les conditions de haut en bas. Des qu'une est vraie, il execute le bloc et ignore le reste.

> **Astuce DanielCraft** - L'indentation (4 espaces) definit le bloc. Pas d'accolades en Python.

Les operateurs de comparaison

Operateur	Signification
`==`	Egal a
`!=`	Different de
`<`	Inferieur
`>`	Superieur
`<=`	Inferieur ou egal
`>=`	Superieur ou egal

Combiner avec and / or / not

```
age = 20
permis = True
if age >= 18 and permis:
    print("Tu peux conduire")
```

Petite histoire

Max cree un programme qui verifie si un mot de passe fait au moins 8 caracteres. Si oui, il affiche "Mot de passe valide". Sinon, "Trop court". Trois lignes suffisent.

Erreur classique

```
if age = 18:  # SyntaxError ! Utilise == pour comparer
```

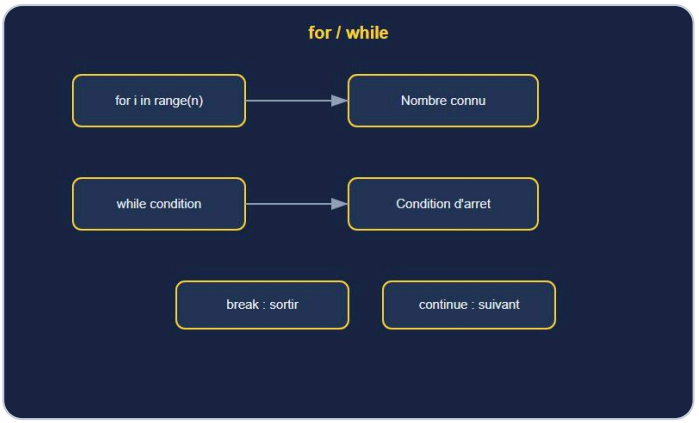
Un seul `=` assigne. Deux `==` comparent.

A retenir

- `if`, `elif`, `else` pour decider.
- Indentation obligatoire (4 espaces).
- `==` pour comparer, `=` pour assigner.

CHAPITRE 7

Les boucles



for / while : repeter sans copier.

Repeter sans copier

Une boucle execute un bloc plusieurs fois. Deux types : `for` (nombre de fois connu) et `while` (condition).

La boucle for

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

`range(5)` genere les nombres de 0 a 4. Le bloc indente se repete 5 fois.

Parcourir une liste

```
fruits = ["pomme", "banane", "cerise"]
for fruit in fruits:
    print(fruit)
```

La boucle while

```
compteur = 0
while compteur < 3:
    print(compteur)
    compteur += 1
```

> **Piege** - Si tu oublies `compteur += 1`, la boucle tourne a l'infini. Ctrl+C pour arreter.

break et continue

```
for i in range(10):
    if i == 5:
        break  # Sort de la boucle
    if i % 2 == 0:
        continue  # Passe au tour suivant
    print(i)  # Affiche 1, 3
```

> **Astuce DanielCraft** - Prefere `for` quand tu connais le nombre de tours. `while` quand tu attends une condition.

Sam automatise avec une boucle.

Petite histoire

Sam doit afficher les tables de multiplication de 1 a 10. Deux boucles imbriquees et c'est fait en 4 lignes.

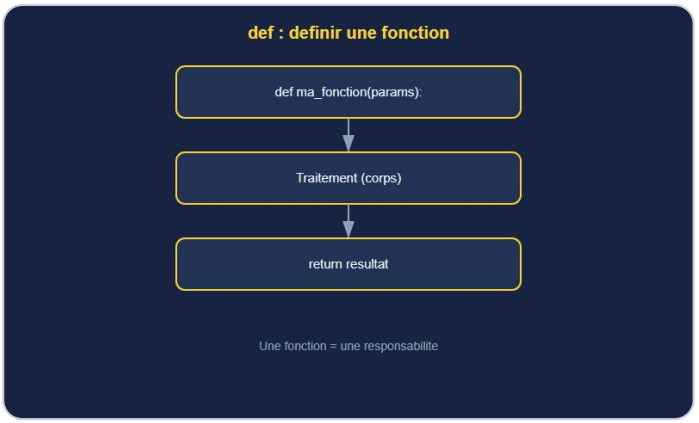
```
for i in range(1, 11):
    for j in range(1, 11):
        print(f"{i} x {j} = {i*j}")
```

A retenir

- `for` pour un nombre connu d'iterations.
- `while` pour une condition.
- `break` sort, `continue` passe au suivant.

CHAPITRE 8

Les fonctions



def : ecrire une fois, appeler souvent.

Pourquoi des fonctions ?

Une fonction regroupe du code reutilisable sous un nom. Au lieu de copier-coller, tu appelles la fonction.

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

saluer("Nora")
saluer("Max")
```

Parametres et retour

```
def additionner(a, b):
    return a + b

resultat = additionner(3, 7)
print(resultat)  # 10
```

`return` renvoie une valeur. Sans `return`, la fonction retourne `None`.

Parametres par defaut

```
def presenter(nom, langue="francais"):
    print(f"{nom} parle {langue}")

presenter("Lea")             # Lea parle francais
presenter("Tom", "anglais")  # Tom parle anglais
```

> **Astuce DanielCraft** - Une fonction doit faire une seule chose. Si elle fait trop, decoupe-la.

Fonctions integrees utiles

Fonction	Role
`len()`	Longueur
`max()` / `min()`	Plus grand / petit
`abs()`	Valeur absolue
`round()`	Arrondir
`input()`	Lire une saisie
`type()`	Type d'une valeur

Petite histoire

Max ecrit 3 fois le meme calcul de TVA. Sam lui montre comment creer `calculer_ttc(prix_ht)` et l'appeler partout. Le code passe de 15 lignes a 6.

Erreur classique

```
def dire_bonjour():
    print("Bonjour")

resultat = dire_bonjour()
print(resultat)  # None ! La fonction n'a pas de return.
```

A retenir

- `def nom(params):` pour definir.
- `return` pour renvoyer une valeur.
- Une fonction = une responsabilite.

CHAPITRE 9

Les listes

Liste : collection ordonnee et mutable.

C'est quoi une liste ?

Une liste stocke plusieurs valeurs dans un seul conteneur, dans un ordre precis.

```
notes = [14, 17, 11, 19, 8]
print(notes[0])   # 14 (premier element)
print(notes[-1])  # 8 (dernier element)
```

Ajouter et supprimer

```
fruits = ["pomme", "banane"]
fruits.append("cerise")      # Ajoute a la fin
fruits.insert(0, "kiwi")     # Insere au debut
fruits.remove("banane")      # Supprime par valeur
del fruits[0]                # Supprime par index
```

Parcourir une liste

```
for fruit in fruits:
    print(fruit)
```

Avec l'index :

```
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")
```

Tranches (slicing)

```
nombres = [0, 1, 2, 3, 4, 5]
print(nombres[1:4])   # [1, 2, 3]
print(nombres[:3])    # [0, 1, 2]
print(nombres[3:])    # [3, 4, 5]
```

> **Astuce DanielCraft** - Les indices commencent a 0. `liste[0]` est le premier element.

Methodes utiles

Methode	Role
`.append(x)`	Ajoute x a la fin
`.pop()`	Retire et retourne le dernier
`.sort()`	Trie sur place
`.reverse()`	Inverse l'ordre
`len(liste)`	Nombre d'elements

Petite histoire

Nora stocke les prenoms de sa classe dans une liste. Elle utilise `.sort()` pour les afficher par ordre alphabetique et `len()` pour compter les eleves.

A retenir

- Liste = collection ordonnee, modifiable.
- Index a partir de 0, negatifs depuis la fin.
- `.append()`, `.remove()`, `.sort()` pour manipuler.

CHAPITRE 10

Les dictionnaires

Dictionnaire : cle -> valeur.

C'est quoi un dictionnaire ?

Un dictionnaire associe des cles a des valeurs. Comme un vrai dictionnaire : mot -> definition.

```
personne = {
    "nom": "Lea",
    "age": 28,
    "ville": "Bordeaux"
}
print(personne["nom"])  # Lea
```

Ajouter et modifier

```
personne["email"] = "lea@exemple.fr"  # Ajoute
personne["age"] = 29                   # Modifie
del personne["ville"]                  # Supprime
```

Parcourir un dictionnaire

```
for cle, valeur in personne.items():
    print(f"{cle}: {valeur}")
```

Verifier l'existence d'une cle

```
if "email" in personne:
    print(personne["email"])
```

> **Astuce DanielCraft** - Utilise `.get(cle, defaut)` pour eviter une erreur si la cle n'existe pas.

```
tel = personne.get("telephone", "Non renseigne")
```

Methodes utiles

Methode	Role
`.keys()`	Toutes les cles
`.values()`	Toutes les valeurs
`.items()`	Paires cle-valeur
`.get(k, d)`	Valeur ou defaut
`len(d)`	Nombre de paires

Petite histoire

Max cree un dictionnaire pour stocker les prix de ses produits. Il parcourt avec `.items()` et calcule le total en 3 lignes.

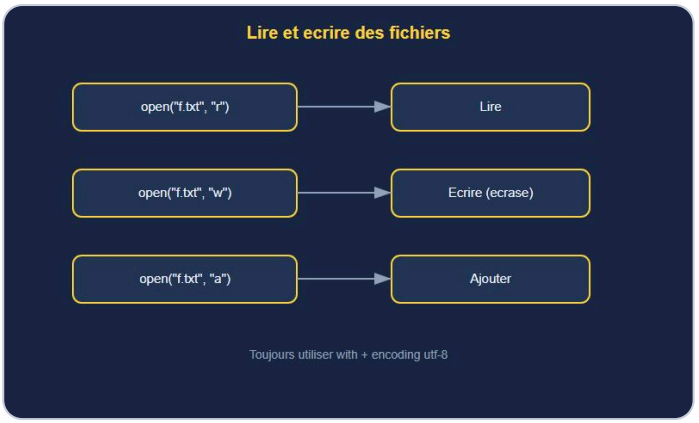
```
panier = {"pain": 1.20, "lait": 0.95, "oeufs": 2.50}
total = sum(panier.values())
print(f"Total : {total:.2f} EUR")
```

A retenir

- Dictionnaire = paires cle-valeur.
- Acces par cle : `dico["cle"]`.
- `.get()` pour eviter les erreurs.

CHAPITRE 11

Lire et ecrire des fichiers



open() + with : lire et ecrire proprement.

Pourquoi manipuler des fichiers ?

Les programmes ont souvent besoin de sauvegarder des donnees ou d'en lire. Python rend la lecture et l'ecriture de fichiers texte tres simples.

Ecrire dans un fichier

```
with open("notes.txt", "w", encoding="utf-8") as f:
    f.write("Maths : 15\n")
    f.write("Francais : 13\n")
```

`"w"` cree le fichier (ou l'ecrase). `with` ferme le fichier automatiquement.

Lire un fichier

```
with open("notes.txt", "r", encoding="utf-8") as f:
    contenu = f.read()
print(contenu)
```

Lire ligne par ligne

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

> **Astuce DanielCraft** - Precise toujours `encoding="utf-8"` pour eviter les problemes d'accents.

Ajouter sans ecraser

```
with open("notes.txt", "a", encoding="utf-8") as f:
    f.write("Histoire : 16\n")
```

`"a"` (append) ajoute a la fin sans supprimer le contenu existant.

Petite histoire

Nora ecrit un programme qui sauvegarde ses depenses dans un fichier CSV. Chaque jour elle ajoute une ligne. A la fin du mois elle relit le fichier et calcule le total.

Erreur classique

```
f = open("data.txt")
contenu = f.read()
# Si une erreur survient ici, le fichier reste ouvert
```

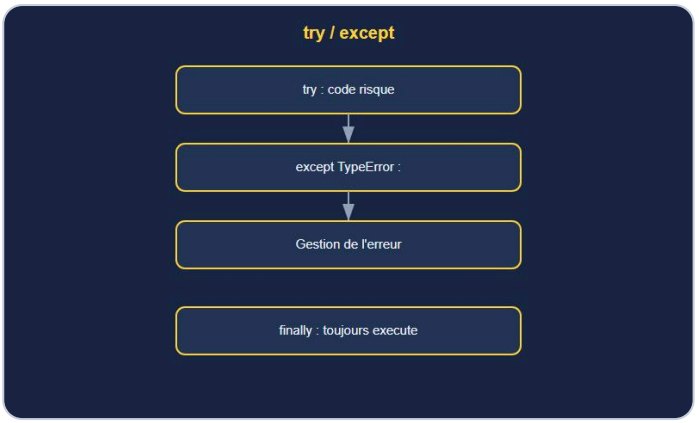
Utilise toujours `with` pour garantir la fermeture.

A retenir

- `with open(...) as f:` pour ouvrir proprement.
- `"r"` lire, `"w"` ecrire (ecrase), `"a"` ajouter.
- Toujours preciser `encoding="utf-8"`.

CHAPITRE 12

Gerer les erreurs



try/except : attraper au lieu de planter.

Les erreurs arrivent

Meme un bon programme peut rencontrer des situations imprevues : fichier introuvable, division par zero, saisie invalide. Python permet de les attraper au lieu de planter.

try / except

```
try:
    nombre = int(input("Un nombre : "))
    print(10 / nombre)
except ValueError:
    print("Ce n'est pas un nombre valide.")
except ZeroDivisionError:
    print("Impossible de diviser par zero.")
```

Les erreurs courantes

Erreur	Cause
`SyntaxError`	Code mal ecrit (parenthese, indentation)
`NameError`	Variable inexistante
`TypeError`	Operation sur mauvais type
`ValueError`	Valeur inappropriee
`IndexError`	Index hors limites
`KeyError`	Cle absente d'un dictionnaire
`FileNotFoundError`	Fichier introuvable

finally et else

```
try:
    f = open("data.txt")
    contenu = f.read()
except FileNotFoundError:
    print("Fichier absent")
else:
    print(f"Lu : {len(contenu)} caracteres")
finally:
    print("Fin du bloc")
```

> **Astuce DanielCraft** - N'attrape que les erreurs que tu sais gerer. Laisser remonter les autres aide au debug.

Lever une erreur

```
def retirer(solde, montant):
    if montant > solde:
        raise ValueError("Solde insuffisant")
    return solde - montant
```

Max lit le message d'erreur calmement.

Petite histoire

Max demande un nombre a l'utilisateur pour calculer une moyenne. L'utilisateur tape "abc". Sans `try/except`, le programme plante. Avec, il redemande poliment.

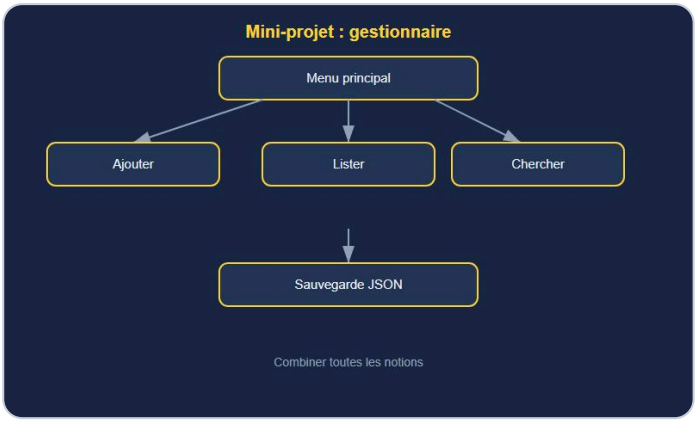
A retenir

- `try/except` pour attraper les erreurs.
- Precise le type d'erreur (`ValueError`, etc.).
- `raise` pour lever tes propres erreurs.

CHAPITRE 13

Mini-projet : gestionnaire de contacts

Nora, Max et Sam valident le projet.



Mini-projet : un programme complet.

L'objectif

On cree un petit programme en ligne de commande qui permet d'ajouter, lister et chercher des contacts. Ce projet combine variables, listes, dictionnaires, fonctions, boucles, conditions et fichiers.

Structure du programme

```
import json
from pathlib import Path

FICHIER = Path("contacts.json")

def charger():
    if FICHIER.exists():
        return json.loads(FICHIER.read_text(encoding="utf-8"))
    return []

def sauvegarder(contacts):
    FICHIER.write_text(
        json.dumps(contacts, indent=2, ensure_ascii=False),
        encoding="utf-8"
    )
```

```
def ajouter(contacts):
    nom = input("Nom : ")
    tel = input("Telephone : ")
    contacts.append({"nom": nom, "tel": tel})
    sauvegarder(contacts)
    print(f"{nom} ajoute.")

def lister(contacts):
    if not contacts:
        print("Aucun contact.")
        return
    for i, c in enumerate(contacts, 1):
        print(f"  {i}. {c['nom']} - {c['tel']}")

def chercher(contacts):
    terme = input("Recherche : ").lower()
    resultats = [c for c in contacts if terme in c["nom"].lower()]
    if resultats:
        for c in resultats:
            print(f"  {c['nom']} - {c['tel']}")
    else:
        print("Aucun resultat.")

def main():
    contacts = charger()
    while True:
        print("\n1. Ajouter  2. Lister  3. Chercher  4. Quitter")
        choix = input("> ")
        if choix == "1":
            ajouter(contacts)
        elif choix == "2":
            lister(contacts)
        elif choix == "3":
            chercher(contacts)
        elif choix == "4":
            print("A bientot !")
            break

if __name__ == "__main__":
    main()
```

Ce que tu apprends

- Decouverte de `json` pour sauvegarder des donnees structurees.
- Utilisation de `Path` pour verifier l'existence d'un fichier.
- Boucle principale avec menu textuel.
- Fonctions bien decoupees (une par action).

> **Astuce DanielCraft** - Commence par le squelette (menu + fonctions vides), puis remplis une fonction a la fois.

Pour aller plus loin

- Ajouter un champ email.
- Permettre la suppression d'un contact.
- Trier les contacts par nom.

A retenir

- Un projet = assemblage de notions deja vues.
- Decouper en fonctions rend le code lisible.
- `json` permet de sauvegarder des donnees facilement.

CHAPITRE 14

Ce qu'il faut retenir

La carte des notions essentielles.

Les briques essentielles

Apres ces chapitres, tu maitrises les bases de Python :

Concept	Resume
Variables	Noms qui pointent vers des valeurs
Types	`int`, `float`, `str`, `bool`
Conditions	`if / elif / else`
Boucles	`for`, `while`, `break`, `continue`
Fonctions	`def`, `return`, parametres
Listes	Collections ordonnees, mutables
Dictionnaires	Paires cle-valeur
Fichiers	`open()`, `with`, lire/ecrire
Erreurs	`try/except`, `raise`

La methode

1. 1. Lis le chapitre.
2. 2. Tape les exemples toi-meme (pas de copier-coller).
3. 3. Modifie-les pour voir ce qui change.
4. 4. Fais les ateliers.
5. 5. Reviens quand tu bloques.

> **Astuce DanielCraft** - Apprendre a coder, c'est comme apprendre a nager : il faut se jeter a l'eau.

Ce qui vient apres

- Les modules et imports.
- La programmation orientee objet.
- Les bibliotheques (requests, pandas, flask...).
- Les tests unitaires.
- Les projets plus ambitieux.

A retenir

- Tu as les bases. Le reste, c'est de la pratique.
- Chaque erreur est une lecon.
- Python est vaste, mais les fondations sont la.

CHAPITRE 15

Atelier : variables et types

Atelier : variables et conversions.

Objectif

Mettre en pratique les variables, les types et les conversions vus aux chapitres 4 et 5.

Exercice 1 : carte d'identite

Cree un programme qui stocke ton prenom, ton age et ta ville dans des variables, puis affiche une phrase complete avec une f-string.

```
prenom = "Sam"
age = 22
ville = "Nantes"
print(f"Je suis {prenom}, {age} ans, je vis a {ville}.")
```

Exercice 2 : convertisseur EUR -> USD

Demande un montant en euros, convertis en dollars (taux 1.08) et affiche le resultat arrondi a 2 decimales.

```
euros = float(input("Montant en EUR : "))
dollars = euros * 1.08
print(f"{euros} EUR = {dollars:.2f} USD")
```

Exercice 3 : swap de variables

Echange les valeurs de deux variables sans utiliser de troisieme variable.

```
a = 5
b = 9
a, b = b, a
print(a, b)  # 9 5
```

Defi bonus

Demande le rayon d'un cercle et affiche son perimetre et son aire.

```
import math
rayon = float(input("Rayon : "))
print(f"Perimetre : {2 * math.pi * rayon:.2f}")
print(f"Aire : {math.pi * rayon**2:.2f}")
```

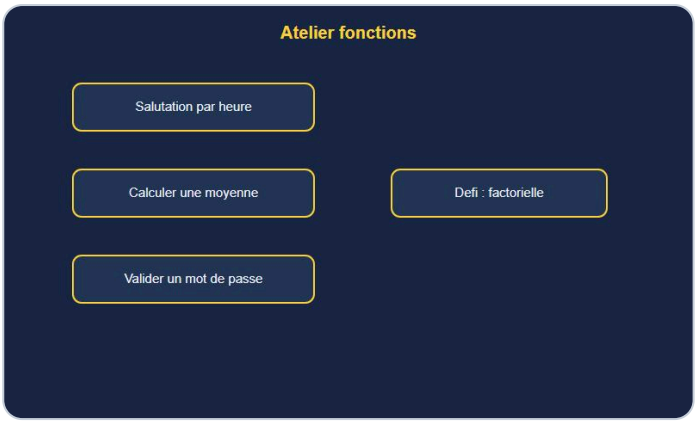
> **Astuce DanielCraft** - Teste avec des valeurs extremes : 0, negatif, tres grand.

A retenir

- Les f-strings facilitent l'affichage formate.
- `float()` pour convertir une saisie en nombre decimal.
- Python permet le swap elegant : `a, b = b, a`.

CHAPITRE 16

Atelier : fonctions



Atelier : creer des fonctions.

Objectif

Pratiquer la creation et l'utilisation de fonctions. Chaque exercice te fait ecrire une fonction avec un role precis.

Exercice 1 : salutation personnalisee

```
def saluer(nom, heure):
    if heure < 12:
        moment = "Bonjour"
    elif heure < 18:
        moment = "Bon apres-midi"
    else:
        moment = "Bonsoir"
    return f"{moment} {nom} !"

print(saluer("Lea", 9))   # Bonjour Lea !
print(saluer("Max", 20))  # Bonsoir Max !
```

Exercice 2 : calculer une moyenne

```
def moyenne(notes):
    if not notes:
        return 0
    return sum(notes) / len(notes)

print(moyenne([14, 16, 11, 18]))  # 14.75
```

Exercice 3 : mot de passe valide ?

Ecris une fonction qui verifie qu'un mot de passe fait au moins 8 caracteres et contient un chiffre.

```python
def mdp_valide(mdp):
    if len(mdp) < 8:
        return False
    return any(c.isdigit() for c in mdp)

print(mdp_valide("abc"))       # False
print(mdp_valide("Python3!"))  # True
```

Exercice 4 : factorielle recursive

```python
def factorielle(n):
    if n <= 1:
        return 1
    return n * factorielle(n - 1)

print(factorielle(5))  # 120
```

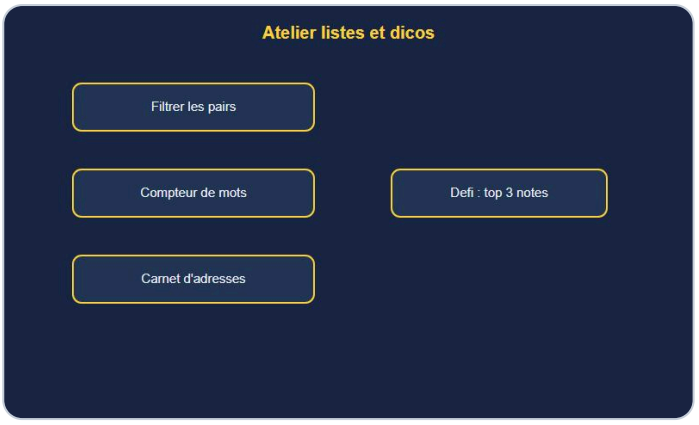
> **Astuce DanielCraft** - Teste tes fonctions avec plusieurs cas : cas normal, cas limite (0, vide), cas erreur.

A retenir

- Une fonction = un nom + des parametres + un `return`.
- Teste chaque fonction isolement avant de l'integrer.
- La recursion est puissante mais attention a la profondeur.

CHAPITRE 17

Atelier : listes et dictionnaires



Atelier : listes et dictionnaires.

Objectif

Manipuler des collections : ajouter, filtrer, trier, transformer.

Exercice 1 : filtrer les pairs

```
nombres = [3, 8, 1, 12, 7, 4, 15, 2]
pairs = [n for n in nombres if n % 2 == 0]
print(pairs)  # [8, 12, 4, 2]
```

Exercice 2 : compteur de mots

```
phrase = "le chat dort le chat mange le chat joue"
mots = phrase.split()
compteur = {}
for mot in mots:
    compteur[mot] = compteur.get(mot, 0) + 1
print(compteur)
# {'le': 3, 'chat': 3, 'dort': 1, 'mange': 1, 'joue': 1}
```

Exercice 3 : inverser une liste sans .reverse()

```
def inverser(lst):
    return lst[::-1]

print(inverser([1, 2, 3, 4]))  # [4, 3, 2, 1]
```

Exercice 4 : carnet d'adresses

```
carnet = []

def ajouter_contact(nom, email):
    carnet.append({"nom": nom, "email": email})

def trouver(nom):
    return [c for c in carnet if nom.lower() in c["nom"].lower()]

ajouter_contact("Nora Duval", "nora@exemple.fr")
ajouter_contact("Max Petit", "max@exemple.fr")
print(trouver("nora"))
```

Defi : top 3 des notes

```
notes = [8, 14, 19, 11, 17, 6, 15]
top3 = sorted(notes, reverse=True)[:3]
print(f"Top 3 : {top3}")  # [19, 17, 15]
```

> **Astuce DanielCraft** - Les list comprehensions rendent le code compact. Mais si ca devient illisible, utilise une boucle classique.

A retenir

- `[expr for x in liste if cond]` : list comprehension.
- `.get(cle, defaut)` evite les KeyError.
- `sorted()` cree une nouvelle liste triee.

CHAPITRE 18

Erreurs classiques en Python

Pieges : indentation, =, index.

Les pieges des debutants

Chaque debutant tombe dans les memes pieges. Les connaitre te fait gagner des heures.

1. Indentation incorrecte

```
if True:
print("Erreur")  # IndentationError
```

Toujours 4 espaces apres `:`.

2. Confondre = et ==

```
if x = 5:   # SyntaxError
if x == 5:  # Correct
```

3. Modifier une liste pendant l'iteration

```
nombres = [1, 2, 3, 4, 5]
for n in nombres:
    if n % 2 == 0:
        nombres.remove(n)  # Comportement imprevisible
```

Solution : creer une nouvelle liste filtree.

4. Oublier les deux-points

```
if x > 5    # SyntaxError : il manque le :
    print(x)
```

5. Index hors limites

```
liste = [1, 2, 3]
print(liste[3])  # IndexError : indices 0, 1, 2 seulement
```

6. Variable non initialisee

```
print(total)  # NameError si total n'existe pas encore
```

7. Mutable par defaut dans les fonctions

```
def ajouter(element, liste=[]):  # Piege !
    liste.append(element)
    return liste
```

La liste par defaut est partagee entre les appels. Utilise `None` :

```
def ajouter(element, liste=None):
    if liste is None:
        liste = []
    liste.append(element)
    return liste
```

> **Astuce DanielCraft** - Lis toujours le message d'erreur en entier. La derniere ligne donne le type, les lignes au-dessus montrent ou.

A retenir

- Indentation = structure du code.
- `=` assigne, `==` compare.
- Ne modifie pas une liste pendant que tu la parcours.
- Lis les messages d'erreur : ils sont explicites en Python.

CHAPITRE 19

Bonnes pratiques

PEP 8, noms clairs, tester.

Ecrire du code lisible

Le code est lu bien plus souvent qu'il n'est ecrit. La lisibilite est prioritaire.

Nommage

- Variables et fonctions : `snake_case` (ex: `prix_total`, `calculer_moyenne`).
- Constantes : `MAJUSCULES` (ex: `TVA = 0.20`).
- Classes : `PascalCase` (ex: `CompteBancaire`).
- Noms explicites : `age` plutot que `a`, `liste_clients` plutot que `lc`.

Structure du code

```
# 1. Imports
import json
from pathlib import Path

# 2. Constantes
FICHIER_CONFIG = Path("config.json")

# 3. Fonctions
def charger_config():
    ...

# 4. Point d'entree
if __name__ == "__main__":
    config = charger_config()
```

Regles de style (PEP 8)

- 4 espaces pour l'indentation (pas de tabulations).
- Lignes de 79 caracteres max (tolerance a 100).
- Deux lignes vides entre les fonctions de premier niveau.
- Un espace autour des operateurs : `x = 5`, pas `x=5`.

> **Astuce DanielCraft** - Installe un linter (flake8 ou ruff) pour reperer les problemes automatiquement.

Documenter

```
def calculer_ttc(prix_ht, tva=0.20):
    """Calcule le prix TTC a partir du HT et du taux de TVA."""
    return prix_ht * (1 + tva)
```

Les docstrings expliquent le "pourquoi", pas le "comment".

Tester

```
assert calculer_ttc(100) == 120.0
assert calculer_ttc(100, 0.055) == 105.5
```

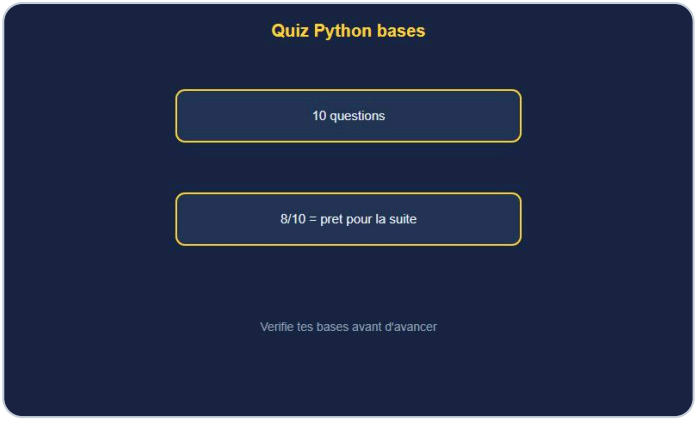
Meme des tests simples avec `assert` detectent des regressions.

A retenir

- Noms explicites en snake_case.
- PEP 8 pour le style.
- Docstrings pour les fonctions importantes.
- Tester meme simplement vaut mieux que ne pas tester.

QUIZ

Quiz Python - Les bases



Quiz Python bases.

Teste tes connaissances

Reponds de tete avant de verifier. Pas de piege, juste les bases.

Q1. Quel mot-cle definit une fonction en Python ?

A) function B) def C) fn D) func

Q2. Quel est le resultat de `type(3.14)` ?

A) `<class 'int'>` B) `<class 'float'>` C) `<class 'str'>` D) `<class 'decimal'>`

Q3. Comment acceder au dernier element d'une liste `ma_liste` ?

A) `ma_liste[last]` B) `ma_liste[-1]` C) `ma_liste[len(ma_liste)]` D) `ma_liste.last()`

Q4. Quel mode ouvre un fichier en ajout sans ecraser ?

A) `"w"` B) `"r"` C) `"a"` D) `"x"`

Q5. Que retourne une fonction sans `return` ?

A) 0 B) `""` C) `None` D) `False`

Q6. Quel operateur verifie l'egalite ?

A) `=` B) `==` C) `===` D) `eq()`

Q7. `range(3)` genere :

A) `[1, 2, 3]` B) `[0, 1, 2]` C) `[0, 1, 2, 3]` D) `[3]`

Q8. Comment attraper une erreur en Python ?

A) `catch/throw` B) `try/except` C) `begin/rescue` D) `do/handle`

Q9. Quel est le type de `{"nom": "Lea"}` ?

A) list B) tuple C) dict D) set

Q10. `"hello".upper()` retourne :

A) `"Hello"` B) `"HELLO"` C) `"hello"` D) Erreur

Reponses

1-B, 2-B, 3-B, 4-C, 5-C, 6-B, 7-B, 8-B, 9-C, 10-B

> **Astuce DanielCraft** - 8/10 ou plus ? Tu es pret pour le niveau intermediaire. Sinon, relis les chapitres concernes.

FINAL

Bravo !

Bravo. Tu codes en Python.

Tu as termine Python - Les bases

Felicitations. Tu sais maintenant ecrire des programmes Python clairs et fonctionnels. Tu maitrises les variables, les types, les conditions, les boucles, les fonctions, les listes, les dictionnaires, les fichiers et la gestion d'erreurs.

Ce que tu as construit

- Un premier programme qui affiche du texte.
- Des fonctions reutilisables.
- Un gestionnaire de contacts complet.
- Des exercices pratiques sur chaque notion.

La suite avec DanielCraft

Le livre **Python - Intermediaire** couvre :

- Les classes et l'orientee objet.
- Les modules et packages.
- Les decorateurs et generateurs.
- Les bibliotheques populaires (requests, pandas).
- Les tests unitaires avec pytest.

> **Astuce DanielCraft** - Ne reste pas dans la theorie. Choisis un petit projet personnel et code-le. C'est la meilleure facon de progresser.

Un dernier conseil

Python est un outil. Plus tu l'utilises, plus il devient naturel. Chaque bug resolu te rend meilleur. Chaque projet termine te donne confiance.

Tu as les bases. Maintenant, construis.

Tu as les briques. Maintenant, construis.