# Programmation en Python - Séance 2 : Débuter avec Python

Hervé TALE KALACHI

ENSPY – Université de Yaoundé 1

# Débuter avec Python : modes d'utilisation

- **REPL** (Read–Eval–Print Loop) : tests rapides, feedback immédiat.
- **Scripts** `.py` : programmes réutilisables, versionnés, exécutables.
- **Notebooks** (Jupyter) : mélange code/texte/visualisations pour exploration.
- **IDE/Éditeurs** : VS Code, PyCharm, Thonny (intégration, linting, débogage).

# REPL — premiers pas (à vous de jouer)

Ouvrez le REPL (`python` ou `python3`) et testez :

```
>>> 1 + 2
>>> "a" * 3
>>> type(3.14), type(True), type("ok")
>>> len([10, 20, 30])
```

**Objectif** : se familiariser avec l'évaluation immédiate et les types de base.

# Calculer avec Python — 1

**Objectif** : manipuler les opérations de base dans le REPL ou un script.

```
# Opérateurs arithmétiques de base
>>> 3 + 5 # addition -> 8
>>> 10 - 4 # soustraction -> 6
>>> 6 * 7 # multiplication -> 42
>>> 21 / 6 # division "réelle" -> 3.5
>>> 2 ** 5 # puissance -> 32

# Types : int vs float
>>> type(3), type(3.0) # (<class 'int'>, <class 'float'>)

# Affectation et réutilisation
>>> a = 12
>>> b = 3.5
>>> a * b # 42.0
```

**À vous de jouer** : Calculer $\pi r^2$ pour $r = 12$ et afficher le résultat.

# Calculer avec Python — 2

**Priorités (ordre d'évaluation)** : `**` ▷ `*` `/` `//` `%` ▷ `+` `-` (utiliser des **parenthèses** pour lever les ambiguïtés)

```
# Priorités et parenthèses
>>> 1 + 2 * 3 # 7 (et pas 9)
>>> (1 + 2) * 3 # 9
>>> 2 ** 3 ** 2 # 512 (car 3**2 = 9, puis 2**9)

# Division entière et modulo
>>> 17 // 5 # 3 (quotient entier)
>>> 17 % 5 # 2 (reste)
>>> 5 * (17 // 5) + (17 % 5) # 17 (identité euclidienne)
```

**À vous de jouer** : Écrire une expression qui renvoie le *chiffre des unités* d'un entier `n`.

# Calculer avec Python — 3

**Flottants** : représentation binaire ⇒ petites imprécisions possibles.

```
>>> 0.1 + 0.2 # 0.30000000000000004
>>> round(0.1 + 0.2, 2) # 0.3

# Conversions
>>> int(3.9) # 3 (tronque)
>>> float(3) # 3.0

# Formatage (affichage contrôlé)
>>> x = 22/7
>>> print(f"{x:.3f}") # 3.142
>>> print(f"{x:.6f}") # 3.142857
```

**À retenir** : utiliser `round(..., k)` ou le **formatage** pour l'affichage.

# Calculer avec Python — 4

**Bibliothèque standard** : `math` fournit des constantes et fonctions utiles.

```
>>> import math

>>> r = 12
>>> surface = math.pi * r**2
>>> perimetre = 2 * math.pi * r

>>> print(f"Surface = {surface:.2f}")
>>> print(f"Périmètre = {perimetre:.2f}")

# Exemples : racine carrée, trigonométrie (en radians)
>>> print(math.sqrt(81)) # 9.0
>>> print(math.sin(math.pi/2)) # 1.0
```

**À vous de jouer** : calculer l'aire d'un *secteur* d'angle $\theta$ (en radians) et rayon $r$ : $A = \frac{\theta}{2\pi} \times \pi r^2$.

# Noms de variables : règles essentielles

- **Caractères autorisés** : lettres (A–Z, a–z), chiffres (0–9), _.
- **Doit commencer par une lettre ou _** (jamais par un chiffre).
- **Sensible à la casse** : `compte`, `Compte` et `COMPTE` sont différents.
- **Pas d'espaces ni d'accents** (conseil pratique) ni de caractères spéciaux : `$ # @ - ...`
- **Ne pas utiliser un *mot réservé*** du langage.

**But du nommage** : lisibilité et intention claire ⇒ choisir des noms *parlants*.

# Valide vs invalide : exemples

**Valides**

```
age = 20
nom_complet = "Ada Lovelace"
x2 = 42
_total = 0
PI_APPROX = 3.14
```

**Invalides (à éviter)**

- `2x` (commence par un chiffre)
- `nom complet` (espace)
- `année` (accent)
- `prix-eur` (tiret)
- `class` (mot réservé)

**Astuce** : pour plusieurs mots, utiliser `snake_case` (`note_moyenne`).

# Tester un identifiant (pratique immédiate)

**Objectif** : savoir si une chaîne est un identifiant Python valide *et* non réservé.

```
import keyword

def identifiant_valide(name: str) -> bool:
    return name.isidentifier() and name not in keyword.kwlist

tests = ["age", "2x", "nom complet", "class", "note_moyenne"]
for t in tests:
    print(t, "->", identifiant_valide(t))
```

**À vous de jouer** : remplacer `tests` par vos propres propositions et vérifier.

# Mots réservés Python (extraits)

**Exemples** : `False`, `None`, `True`, `and`, `as`, `assert`, `async`, `await`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `nonlocal`, `if`, `import`, `in`, `is`, `lambda`, `match`, `case`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`.

**Lister sur votre machine**

```
import keyword
print(keyword.kwlist) # liste officielle des mots réservés
```

**Conseil** : si un nom est refusé alors qu'il « semble » valide, vérifiez s'il n'est pas réservé.

# Conventions de nommage (PEP 8)

- **Variables / fonctions** : `snake_case` (ex. `note_moyenne`, `calcul_aire`).
- **Constantes** : `UPPER_CASE_WITH_UNDERSCORES` (ex. `MAX_ITER`).
- **Classes** : `CapWords` (ex. `CompteBancaire`).
- **Privé interne (convention)** : préfixer par `_` (ex. `_cache`).
- **Éviter** : `l`, `O`, `I` (ambiguïtés visuelles), et `list`, `dict`, `str`, `type` (écrase des noms intégrés).

**Règle d'or** : un *bon* nom évite les commentaires superflus.

# Quiz express : vrai/faux & correction

**Consigne** : dire si *valide* et *non réservé*. Si non, proposer un nom correct.

```
candidats = [
    "2notes",
    "nom complet",
    "class",
    "élève",
    "Note",
    "note_moyenne",
    "MAX_ITER"
]

import keyword
def ok(name): return name.isidentifier() and name not in keyword.
    kwlist
for c in candidats:
    print(f"{c:12} -> {ok(c)}")
```

**À vous de jouer** : proposez une version *PEP 8* pour chaque cas invalide.

# Quiz express : vrai/faux & correction

**Consigne** : dire si *valide* et *non réservé*. Si non, proposer un nom correct.

```
candidats = [
    "2notes", # -> invalide
    "nom complet",# -> invalide
    "class", # -> mot réservé
    "élève", # -> accent (éviter)
    "Note", # -> valide (mais préférer snake_case "note")
    "note_moyenne",
    "MAX_ITER"
]

import keyword
def ok(name): return name.isidentifier() and name not in keyword.
    kwlist
for c in candidats:
    print(f"{c:12} -> {ok(c)}")
```

**À vous de jouer** : proposez une version *PEP 8* pour chaque cas invalide.

# Affectation et affichage — concepts

- **Affectation** : l'opérateur = *lie un nom* (*variable*) à un *objet* en mémoire.
- **Important** : = (affectation) $\neq$ == (comparaison d'égalité).
- Python est **dynamique** : le type découle de la valeur liée au nom.
- Un **nom** peut être lié à un objet puis à un autre au fil du programme.
- **Affichage** : `print(...)` convertit les objets en texte lisible et l'écrit sur la sortie standard.

# Affectation : formes utiles

```
# Simple
x = 3
y = 4.5

# Multiple (mêmes valeurs)
a = b = 0

# Déballage (unpacking) depuis itérable
p, q, r = [10, 20, 30] # p=10, q=20, r=30

# Échange idiomatique (sans variable temporaire)
x, y = y, x

# Affectations augmentées
n = 10
n += 5 # 15
n *= 2 # 30
n //= 7 # 4
```

# Affichage avec print

```
# Affichage simple
print("Bonjour")

# Plusieurs éléments (séparés par un espace par défaut)
nom, age = "Ada", 36
print("Nom:", nom, "| Age:", age)

# Personnaliser le séparateur et la fin de ligne
print("a", "b", "c", sep=" - ") # a - b - c
print("ligne sans retour", end="") # pas de \n final
print(" ... suite")

# Type des valeurs
print(type(nom), type(age)) # <class 'str'> <class 'int'>
```

**Astuce** : préférez plusieurs arguments à la concaténation `"A" + str(x)`.

# Affichage formaté (*f-strings*)

```
# Insérer des expressions directement dans la chaîne
r = 12
pi = 3.14159
surface = pi * r**2
print(f"Rayon = {r}, Surface = {surface:.2f}") # 452.39
# Alignement et largeur (utile pour tableaux)
for k in range(1, 4):
    print(f"k = {k:>2} | k^2 = {k*k:>3}")

# Séparateurs de milliers (en notation anglaise)
n = 1234567
print(f"n = {n:,}") # n = 1,234,567

# Représentation technique (debug) : !r
s = "Ada"
print(f"s = {s!r}") # s = 'Ada'
```

**À retenir** : `:.2f` (2 décimales), `:>3` (aligner à droite sur 3 colonnes), `:,` (milliers).

# Pièges courants

```
# 1) Confondre = et ==
x = 3
print(x == 3) # True

# 2) Aliasing: plusieurs noms -> même objet mutable
a = b = [0, 0, 0] # alias!
a[0] = 1
print(a, b) # [1, 0, 0] [1, 0, 0]
# Préférer deux listes distinctes:
a, b = [0, 0, 0], [0, 0, 0]

# 3) Sauts de ligne involontaires
print("A", end=""); print("B") # A B sur la même ligne
```

**Message clé** : aliasing sur **mutables** (listes, dict) peut surprendre ; utilisez des copies si besoin.

# Mini-exercices

```
# A) Échange
a, b = 7, 42
# ... échanger, puis afficher: "a=42 | b=7"

# B) Formatage
r, pi = 12, 3.14159
# afficher "r=12 | surface=452.39 | perimetre=75.40"

# C) Tableau aligné
# pour k = 1..5, afficher k, k^2 et k^3 alignés en colonnes (
    largeurs 2, 4, 6)

# D) Séparateur et fin de ligne
# afficher: "A - B - C" puis "FIN" sur la même ligne

# E) Aliasing (vérification)
# créer deux listes indépendantes, modifier l'une, montrer que l'
    autre ne bouge pas
```