

# BDW - Programmation web - Bases de Python

Fabien Duchateau

*fabien.duchateau [at] univ-lyon1.fr*

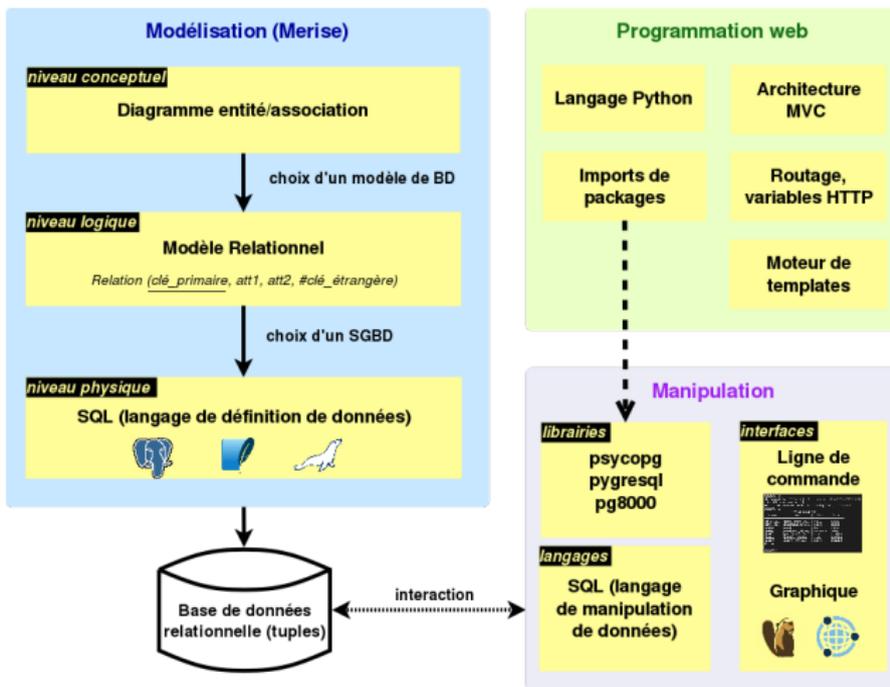
Université Claude Bernard Lyon 1

2024 - 2025



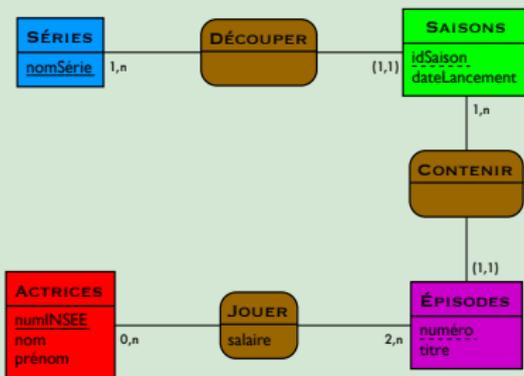
<https://perso.liris.cnrs.fr/fabien.duchateau/BDW/>

# Positionnement dans BDW



Ces diapositives utilisent **le genre féminin** (e.g., chercheuse, développeuses) plutôt que **l'écriture inclusive** (moins accessible, moins concise, et pas totalement inclusive)

# Motivation - site sur les séries



SÉRIES (nomSérie)

SAISONS (idSaison, dateLancement, #nomSérie)

ÉPISODES (numéro, #idSaison, titre)

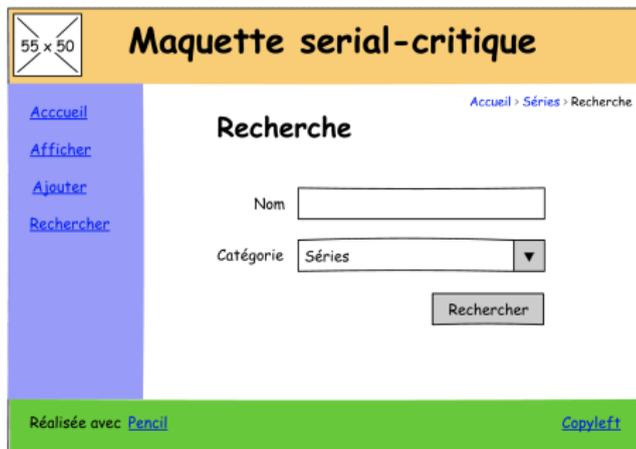
ACTRICES (numINSEE, nom, prénom)

JOUER (#numéro, #idSaison, #numINSEE, salaire)

# Motivation - site sur les séries (2)

Réalisation d'un site web :

- ▶ Afficher les séries
- ▶ Ajouter une série
- ▶ Rechercher une série ou une actrice
- ▶ ...



55 x 50

## Maquette serial-critique

[Accueil](#) > [Séries](#) > Recherche

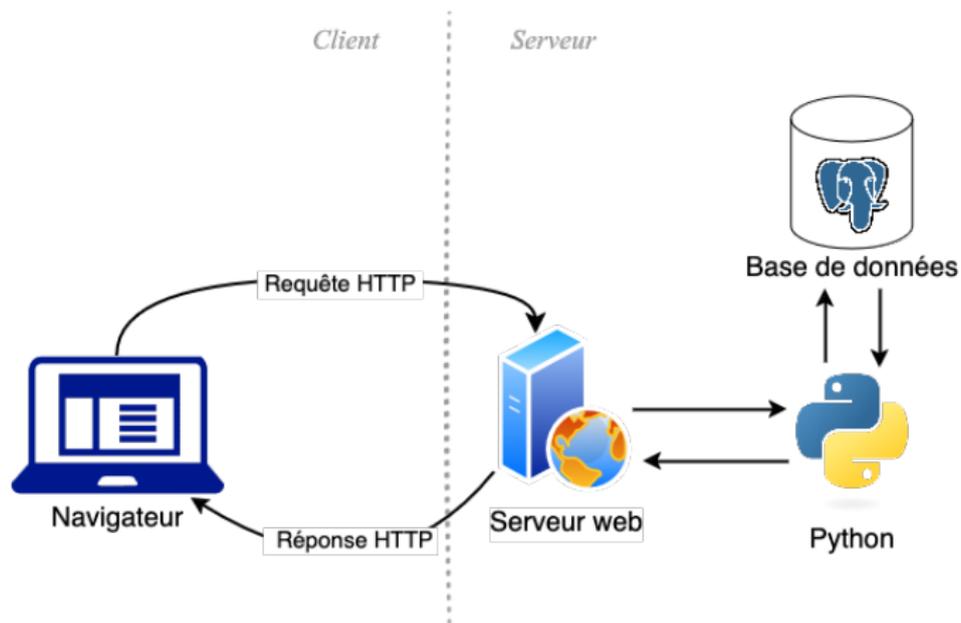
### Recherche

Nom

Catégorie  ▼

Réalisée avec [Pencil](#) [Copyleft](#)

# Rappels



HTML pour le contenu, CSS pour la mise en page/forme, et **Python pour l'aspect dynamique** (e.g., interrogation BD)

# Rappels

Une page web dynamique est générée à la volée par un serveur

HTML et CSS insuffisants pour des besoins comme :

- ▶ Manipulation de bases de données
- ▶ Interactions avec le système de fichiers
- ▶ Utilisation de bibliothèques logicielles
- ▶ Plus généralement pour des traitements complexes

Besoin d'un langage (côté serveur) comme Python

# Plan

Prérequis

Syntaxe et types

Opérations

Structures de contrôle

Fonctions

# Généralités

Langage Python (en référence aux Monty Python) :

- ▶ Origine : 1991 (Guido van Rossum), désormais maintenu par la Python Software Foundation
- ▶ Version 3 (ne plus utiliser la version 2)
- ▶ Python Licence (code ouvert)
- ▶ Populaire et largement utilisé (NASA, Netflix, YouTube, Spotify, Reddit, Google) mais 80% des sites web utilisent PHP



---

<https://www.python.org/>

[https://en.wikipedia.org/wiki/Monty\\_Python](https://en.wikipedia.org/wiki/Monty_Python)

<https://wiki.python.org/moin/PythonBooks>

<https://python.doctor/>

<https://calmcode.io/>

# Caractéristiques du langage

- ▶ Python est un langage de haut niveau, à usage général
- ▶ Sources d'inspiration : C++, Lisp, Perl, etc.
- ▶ Utilisations : scripting, applications de bureau, programmation web, machine learning, etc.
- ▶ Multi-paradigmes (de programmation), dont procédural, fonctionnel, impératif, orienté objet et réflexif
- ▶ *Batteries included*, avec la Python Standard Library

Dans notre contexte, utilisation de Python en procédural

# Philosophie

- ▶ Zen of Python, une liste de 19 recommandations pour écrire du code *pythonique* :
  - ▶ facile à lire et comprendre
  - ▶ simple plutôt que complexe
  - ▶ épars plutôt que dense
  - ▶ ...
  
- ▶ Python Enhancement Proposal (PEP), pour proposer des améliorations du langage (e.g., style de code dans la PEP8)

EXPLICIT IS BETTER THAN IMPLICIT.



READABILITY COUNTS.



<https://peps.python.org/>

[https://en.wikipedia.org/wiki/Zen\\_of\\_Python](https://en.wikipedia.org/wiki/Zen_of_Python)

# Installation de Python

Cette UE nécessite une version de Python  $\geq 3.11$

À la fac ou sur Basthon (pour de petits tests uniquement) :

- ▶ Python 3.11 😊

Sur une machine personnelle :

- ▶ Vérifier sa version de Python (en ligne de commande)

```
$ python --version  
Python 3.11.6      # ok, version > 3.11
```

- ▶ Si besoin, installer ou mettre à jour (<https://www.python.org/>)

---

<https://console.basthon.fr/>

# Environnements virtuels

Un environnement virtuel est une installation de Python autonome, dans un répertoire auto-suffisant

- ▶ Évite de "polluer" les installations Python du système
- ▶ Évite les conflits entre packages (ou entre différentes versions d'un même package)
- ▶ Évite un dysfonctionnement quand les installations Python du système sont mises à jour

Pour chaque projet, il est recommandé de créer un environnement virtuel pour exécuter indépendamment son code

# Environnements virtuels - création et activation

Lancer un terminal (ou invite de commande sous Windows) pour saisir les commandes suivantes.

Pour créer un environnement virtuel dans le répertoire `.venv` :

```
python -m venv .venv # à ne lancer qu'une fois !  
# ou en précisant la version de python pour l'environnement  
python3.10 -m venv .venv # environnement avec python 3.10
```

Pour utiliser/activer l'environnement virtuel du répertoire `.venv` :

```
source .venv/bin/activate # sous linux, macos  
# ou  
.venv\Scripts\activate # sous windows
```

## Environnements virtuels - exemple

```
$ which python
/usr/local/bin/python3
$ python --version
Python 3.7.7
$ python3.11 -m venv mon-nouvel-env
$ source mon-nouvel-env/bin/activate
(mon-nouvel-env) $ which python
/usr/fabien/mon-nouvel-env/bin/python
(mon-nouvel-env) $ python --version
Python 3.11.6
$
```

*La commande `python` actuelle pointe vers `/usr/local/bin` en version 3.7.7. On utilise une version spécifique de python (3.11) pour créer un environnement virtuel et l'activer. Les dernières lignes confirment que python pointe sur la version 3.11 du répertoire `mon-nouvel-env`.*

# Exécution de code Python

Pour exécuter un fichier de code (ici `hello.py`) :

```
hello.py  
print('Hello, world!')
```

```
$ python hello.py  
Hello, world!  
$
```

Pour exécuter du code dans un shell interactif (petits tests) :

```
$ python  
Python 3.11.6 (main, Oct 2 2023, 18 :01 :19) on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print('Hello, world!')  
Hello, world!  
>>>
```

# Éditeurs Python

Liste non exhaustive :

- ▶ Spécifiques Python : Pycharm, IDLE, Thonny, Spyder, ...
- ▶ Génériques : VScodium, Geany, Notepad++, Brackets, Atom, Emacs, ...
- ▶ Frameworks : Django, Flask, FastAPI, ...(pas pour cette UE)

# Plan

Prérequis

**Syntaxe et types**

Opérations

Structures de contrôle

Fonctions

# Généralités

Le langage Python est :

- ▶ Interprété (pas de compilation)
- ▶ À typage dynamique (au moment de l'exécution)
- ▶ À typage fort (e.g., pas d'opération interdite comme `string + integer`)
- ▶ Responsable de la gestion mémoire (*garbage collector*)
- ▶ Basé sur l'**indentation** pour séparer les blocs de code

Un fichier Python (extension `.py`) est un code source exécutable

# Bonnes pratiques

Guide de style dans la PEP8 :

- ▶ Encodage UTF-8
- ▶ Indentation de 4 espaces (par niveau d'indentation), etc.
- ▶ Nommage d'une variable ou d'une fonction : en minuscules, les termes étant séparés par des '\_' (e.g., `ma_variable`)
- ▶ Nommage d'une constante : en majuscules, les termes étant séparés par des '\_' (e.g., `MAX_LINES`)
- ▶ ...

---

<https://peps.python.org/pep-0008/>

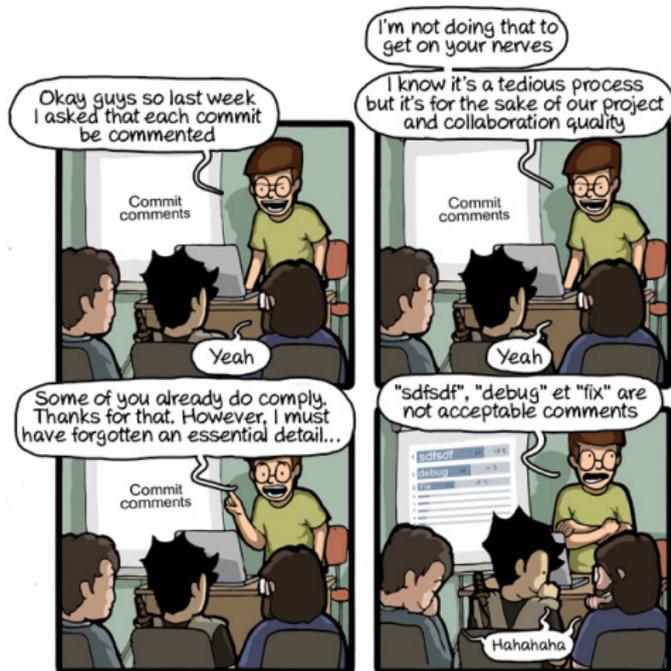
# Commentaires

Commentaire uniligne (dièse) :

```
# commentaire
a = 1 # commentaire
```

Commentaire multiligne (paire de 3 guillemets, simples ou doubles) :

```
'''
commentaire
multiligne
'''
```



# Variables

- ▶ Python assigne un type automatiquement lors de l'exécution
- ▶ Affectation : un nom de variable suivi du symbole = et d'une valeur
- ▶ Utiliser des noms explicites

```
compteur = 1  
nom_ue = 'BDW'
```

<https://luc-damas.fr/hop/>



# Types de données

- ▶ Types simples : `bool`, `int`, `float`, `str` (chaînes de caractères)
- ▶ Types composés : `list` et `tuple` (séquence ordonnée d'éléments), `dict` (mappings)
- ▶ Absence de valeur : `None`
- ▶ Les variables et objets peuvent s'évaluer à vrai ou faux (`True` ou `False`)
- ▶ Certaines opérations sont valables sur tous les objets (e.g., comparaison par égalité, test booléen, conversion en chaîne de caractères)

---

<https://docs.python.org/3/library/stdtypes.html>

Également d'autres types comme `complex`, `set`, `bytearray`, ...

# Chaînes de caractères

- ▶ Valeur d'une chaîne délimitée par des guillemets simples ('chaîne') ou doubles ("chaîne"), sans aucune différence
- ▶ Concaténation de chaînes avec le symbole +
- ▶ Méthode `print()` pour afficher une chaîne (dont la variante *f-string* permettant d'y concaténer des valeurs)

```
>>> acro = 'BD'
>>> print(acro + ' : base de données')
BD : base de données
>>> print(acro + " : bande-dessinée")
BD : bande-dessinée
>>> print(f"{acro} : blu-ray disk")
BD : blu-ray disk
```

# Tuples

Un tuple est *immutable*, donc **non modifiable** !

```
mes_tuples = tuple() # nouveau tuple vide
mes_tuples = (1, False, 'a') # nouveau tuple avec éléments
```

Quelques opérations communes aux types composés :

```
mes_tuples[0] # 1 (premier indice = 0)
mes_tuples[0:2] # 1, False (borne supérieure exclue)
```

Test d'appartenance (ou non) à un tuple :

```
'a' in mes_tuples # True
0 not in mes_tuples # True
```

Autres opérations (concaténation, plus petit/grand élément, etc.)

<https://docs.python.org/3/library/stdtypes.html#tuples>

# Listes

Une liste est *mutable*, donc modifiable

```
ma_liste = list() # nouvelle liste vide
ma_liste = [0, 1, 'a', "b", True] # nouvelle liste évaluée
```

Quelques opérations communes aux types composés :

```
ma_liste[2] # 'a' (premier indice = 0)
len(ma_liste) # 5
'a' not in ma_liste # False
```

Opérations de modification de la liste :

```
ma_liste.append('Z') # [0, 1, 'a', "b", True, 'Z']
ma_liste[3] = 'x' # [0, 1, 'a', 'x', True, 'Z']
del ma_liste[3] # [0, 1, 'a', True, 'Z']
```

Autres opérations (copie, inversion, tri, etc.)

<https://docs.python.org/3/library/stdtypes.html#lists>

# Dictionnaires (mappings)

Un dictionnaire associe des clés à des objets

```
mon_dict = dict() # nouveau dict vide
mon_dict = {1: 'a', 'b': True} # nouveau dict valué
```

Quelques opérations communes aux types composés :

```
mon_dict[1] # 'a'
len(mon_dict) # 2
'a' in mon_dict # False (pas de clé 'a')
```

Opérations spécifiques au dictionnaire :

```
keys(mon_dict) # [1, 'b']
values(mon_dict) # ['a', True]
mon_dict['x'] = 'Y' # {1: 'a', 'b': True, 'x': 'Y'}
del mon_dict['b'] # {1: 'a', 'x': 'Y'}
```

Autres opérations (copie, mises à jour, etc.)

<https://docs.python.org/3/library/stdtypes.html#mapping-types-dict>

# Plan

Prérequis

Syntaxe et types

**Opérations**

Structures de contrôle

Fonctions

# Opérateurs booléens

- ▶ `and` : et logique
- ▶ `or` : ou logique
- ▶ `not` : négation

```
1 and 0 # 0
1 or 0 # 1
' or [1] # [1]
not(42) # False
not('') # True
```

# Opérateurs arithmétiques

- ▶ `+`, `-`, `*`, `/`, `//`, `%`, `**` : addition, soustraction, multiplication, division, division entière, modulo (reste de  $x / y$ ), puissance
- ▶ `abs(x)` : valeur absolue de  $x$
- ▶ `round(x, n)` : arrondi de  $x$  à  $n$  chiffres après la virgule
- ▶ `int(x)`, `float(x)` : conversion de  $x$  en entier ou décimal

```
42 + 1 # 43
42 // 9 # 4
42 % 9 # 6
2 ** 3 # 8
round(1.2345, 2) # 1.23
int("42") # 42
```

<https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>

# Opérateurs de comparaison

- ▶ `==`, `!=` : égalité, différence (deux objets de type différents ne sont jamais égaux)
- ▶ `<`, `<=`, `>`, `>=` : infériorité et supériorité (strictes ou non)
- ▶ `is`, `is not` : identité d'objet (et sa négation)

```
1 <= 2 # True
1 != 2 # True
42 == 42.0 # True
42 == '42' # False
42 is None # False
0 is not "0" # True
```

<https://docs.python.org/3/library/stdtypes.html#comparisons>

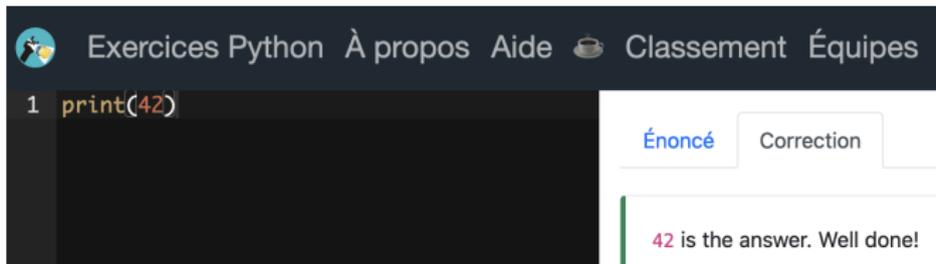
## Quelques fonctions standard

- ▶ `type(x)` : type de données de la variable `x`
- ▶ `len(x)` : nombre d'éléments dans la séquence `x` (e.g., list, str, dict)
- ▶ `range([start], stop, [step])` : retourne un objet représentant une séquence de l'entier `start` jusque l'entier `stop` par pas de `step`

```
type(1) # int
type([1, 2]) # list
len('abcde') # 5
len([0, 1, 2, 3]) # 4
list(range(4)) # [0, 1, 2, 3]
list(range(0, 9, 2)) # [0, 2, 4, 6, 8]
```

<https://docs.python.org/3/library/functions.html>

# En résumé

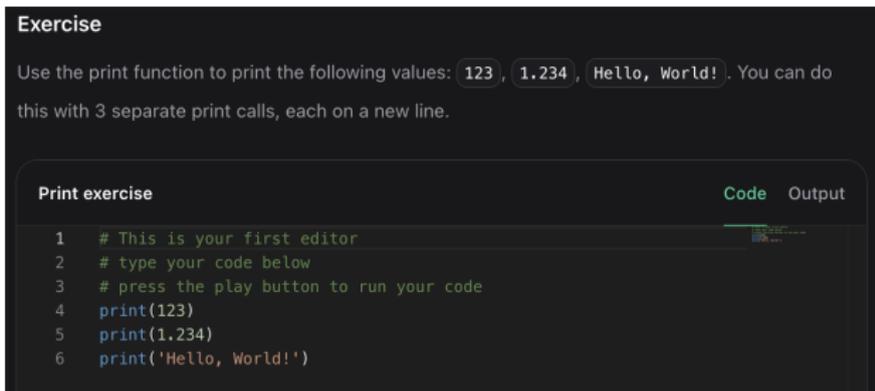


Exercices Python À propos Aide 🍷 Classement Équipes

```
1 print(42)
```

Énoncé Correction

42 is the answer. Well done!



**Exercise**

Use the print function to print the following values: `123`, `1.234`, `Hello, World!`. You can do this with 3 separate print calls, each on a new line.

Print exercise Code Output

```
1 # This is your first editor
2 # type your code below
3 # press the play button to run your code
4 print(123)
5 print(1.234)
6 print('Hello, World!')
```

<https://www.hackinscience.org/exercises/>  
<https://learn-python.adamemery.dev/basics>

# Plan

Prérequis

Syntaxe et types

Opérations

**Structures de contrôle**

Fonctions

# Syntaxe générale

Un **programme**, c'est une suite d'instructions (e.g., assignation, appel de fonction, instruction conditionnelle)

Un **bloc d'instructions** s'identifie par son **indentation** (recommandation de 4 espaces par niveau d'indentation)

```
a = 5 # niveau 0 indentation
if a > 0:
    i = 0 # bloc d'instructions niveau 1
    while i < a:
        print(i) # bloc d'instructions niveau 2
        i += 1
print(a)
```

# Instruction IF...ELSE

La structure conditionnelle IF...ELSE est similaire aux autres langages de programmation :

- ▶ Si la condition renvoie True, exécution des instructions dans le bloc du if
- ▶ Sinon, on exécute les instructions dans le bloc du else (qui est facultatif)

```
chaine = 'hello'  
if chaine:  
    print(chaine)  
else:  
    print("La variable n'existe pas.")
```

*Résultat : hello*

[https://docs.python.org/3/reference/compound\\_stmts.html#the-if-statement](https://docs.python.org/3/reference/compound_stmts.html#the-if-statement)

## Instruction IF...ELIF...ELSE

En cas de tests multiples, on peut ajouter une ou plusieurs instructions `elif`

- ▶ Aussi une instruction `match...case` pour de nombreux choix (e.g, menu)

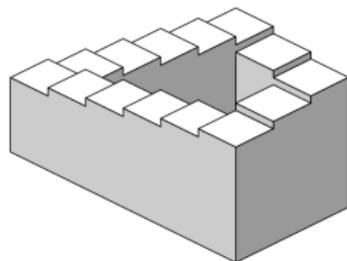
```
chaine = 'hello'
if not chaine:
    print("La variable n'existe pas ou ne contient que des
    ↪ espaces.")
elif chaine.islower(): # vraie si chaine en minuscules
    print("La variable chaine est en minuscules.")
else:
    print(f"La variable chaine est en majuscules")
```

Résultat : *La variable chaine est en minuscules.*

[https://docs.python.org/3/reference/compound\\_stmts.html#the-match-statement](https://docs.python.org/3/reference/compound_stmts.html#the-match-statement)

# Les boucles

- ▶ Permettent de répéter un bloc d'instructions tant qu'une condition est vérifiée
- ▶ Utile pour parcourir une séquence, etc.
- ▶ Deux types de boucles en Python :
  - ▶ `while`
  - ▶ `for`



---

[https://docs.python.org/3/reference/compound\\_stmts.html#the-while-statement](https://docs.python.org/3/reference/compound_stmts.html#the-while-statement)

[https://docs.python.org/3/reference/compound\\_stmts.html#the-for-statement](https://docs.python.org/3/reference/compound_stmts.html#the-for-statement)

[http://fr.wikipedia.org/wiki/Escalier\\_de\\_Penrose](http://fr.wikipedia.org/wiki/Escalier_de_Penrose)

# Boucle WHILE

Une boucle `while` ("tant que") exécute les *instructions* tant que la *condition* est vérifiée

```
n = 2
while n < 10:
    print(n)
    n += 3
```

Résultat : 2 5 8

# Boucle FOR

Une boucle `for` ("pour") permet d'itérer sur une séquence (e.g., liste, tuple, chaîne de caractères ou tout autre objet itérable) en exécutant les *instructions* à chaque itération

```
for i in range(3, 10, 2):  
    print(i)
```

Résultat : 3 5 7 9

## Boucle FOR (2)

```
ma_liste = [0, 1, 'a', "b", True]
for _ in ma_liste:
    print(_, end=' ') # paramètre end pour remplacer le retour à
    ↪ la ligne par un espace
```

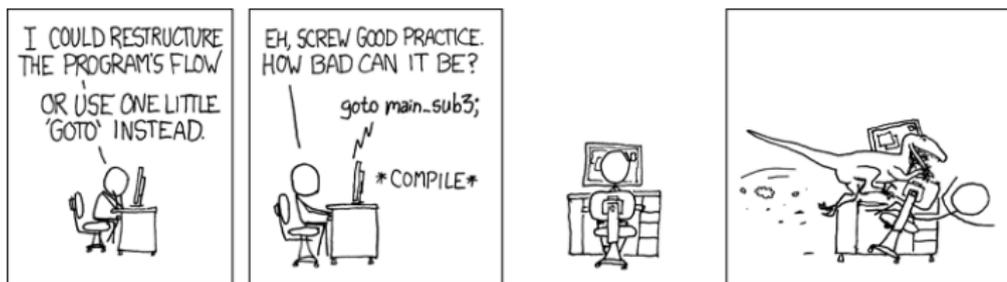
Résultat : 0 1 a b True

```
mon_dict = {1: 'a', 'b': True}
for k, v in mon_dict.items():
    print(f"{k} : {v}", end='\t') # \t correspond à une
    ↪ tabulation
```

Résultat : 1 : a    b : True

# Autres structures de contrôle

- ▶ `assert`, pour vérifier des assertions (tests unitaires)
- ▶ `continue`, `break` pour continuer à l'itération suivante ou sortir d'une boucle
- ▶ ...



[https://docs.python.org/3/reference/simple\\_stmts.html](https://docs.python.org/3/reference/simple_stmts.html)

[https://docs.python.org/3/reference/compound\\_stmts.html](https://docs.python.org/3/reference/compound_stmts.html)

<http://xkcd.com/292>

# Plan

Prérequis

Syntaxe et types

Opérations

Structures de contrôle

**Fonctions**

# Généralités

Une fonction est un ensemble d'instructions réutilisable :

- ▶ En entrée, 0 ou plusieurs paramètres
- ▶ Retourne une valeur (conseillé, e.g., un booléen indiquant le succès/échec de l'exécution de la fonction)
- ▶ Est un objet en Python

Quand utiliser des fonctions ?

- ▶ Traitement récurrent (affichage, calcul, etc.)
- ▶ Traitement similaire mais avec des valeurs différentes en entrée
- ▶ ...

# Définition d'une fonction

En Python, une fonction se définit par le mot clé `def` :

- ▶ `nom_fonction` = le nom de la fonction
- ▶ `param1, param2, ...` = paramètres, pour lesquels il est possible de préciser une valeur par défaut (ici `True` pour `param2`)
- ▶ `valeur` = l'une des valeurs de retour, introduite par le mot clé `return`

```
def nom_fonction(param1, param2=True, ...):  
    # instructions  
    return valeur
```

---

[https://docs.python.org/3/reference/compound\\_stmts.html#function-definitions](https://docs.python.org/3/reference/compound_stmts.html#function-definitions)

Fonctions anonymes (`lambda`), <https://docs.python.org/3/reference/expressions.html#lambda>

# Exemples de fonction

```
def sommer(a, b):  
    return a + b  
  
sommer(1, 2)  # appel de la fonction sommer()
```

*Une fonction qui somme deux nombres passés en paramètres*

```
def sommer(a: int, b: int) -> int:  
    return a + b  
  
sommer(1, 2)  # appel de la fonction sommer()
```

*Même fonction, avec des annotations précisant le type attendu des paramètres et de la valeur retour (mais c'est **non** contraignant)*

# Un moment de réflexion

Écrire une fonction pour colorer des notes stockées dans une liste (vert pour les notes supérieures à 10, rouge sinon)

```
1 def colorier_note(note):
2
3
4
5
6
7 notes = [12, 17, 8, 10, 14, 3] # liste de notes
8
9
```

```
1 def colorier_note(note):
2     couleur = 'red' # couleur rouge
3     if note > 10:
4         couleur = 'green' # couleur vert si note > 10
5
```

12 17 8 10 14 3

# En résumé

S'entraîner en Python :

- ▶ Syntaxe de base (types de données, opérations, structures de contrôle)
- ▶ REPL (Read Evaluate Print Loop, e.g., le shell interactif Python)
- ▶ Exercices progressifs en ligne

**Prochain cours : Python avancé (imports, exceptions, bases de données)**

<https://www.hackinscience.org/>

<https://awesome-python.com/>

<https://www.afterhoursprogramming.com/>

<https://python.swaroopch.com/>

