

Introduction à Python pour l'analyse de données

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue.

Bonjour à tous !

Florent COLLOT

PROQFIND

<https://www.profind.net>

A decorative network diagram in the bottom-right corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue.

Table des matières

<u>Introduction</u>	3
<u>Python</u>	18
<u>Programmation Orientée Objet (POO)</u>	116
<u>Les expression régulière</u>	135
<u>La récursivité</u>	149
<u>Les bases de données</u>	164
<u>La mesure de performance</u>	179
<u>La programmation asynchrone</u>	198
<u>Tkinter</u>	208
<u>Les API</u>	241

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels of connectivity or importance. The lines are thin and gray, creating a mesh-like structure.

1.

Introduction

A decorative network diagram in the bottom-right corner, similar to the one in the top-left, showing a cluster of nodes connected by lines. The nodes vary in size and some have concentric circles, and the lines are thin and gray.



Nom

Florent COLLOT

Email

florent.collot@outlook.com

Activité

Freelance Consultant / Formateur

Spécialisation

Data Science / Développement Applicatif

Diplôme

Master en Informatique (SUPINFO)



Description et objectifs de la formation

Description

Cette formation présente les **fondamentaux** de la **programmation** en mettant l'accent sur le **développement logiciel** via le langage de programmation **Python**.

Objectifs

Cette formation a pour objectifs :

- la **maitrise** des **fondamentaux** du langage de programmation **Python**.
- Etre en **mesure** de **développer** des **logiciels** en **Python**.



Pré-requis

Des connaissances de base en informatique.



Planning

4 jours de 9h30 à 12h30 et de 13h30 à 16h30.

L'histoire de Python

1989 Création du langage **Python** par **Guido Van Rossum**.

2000 Sortie de la **version 2.0** de **Python**.

2001 Association du langage **Python** à la **Python Software Foundation**.

2008 Sortie de la **version 3.0** de **Python**.



Les versions de Python

Version 2

N'est **plus supportée** depuis le **1^{er} janvier 2020**.

En revanche, elle est toujours présente dans les systèmes existants.

Version 3

Nous sommes actuellement à la version **3.12** de **Python**.

Il est **recommandé** d'utiliser la **version 3** de python pour les nouveaux développements.

Téléchargement et installation de Python

Vous pouvez vous référer à l'excellente [documentation officielle](#) de Python.

Environnement de développement

Les **trois principaux IDE** pour développer en **Python** sont :

- **Visual Studio Code** : IDE gratuit de Microsoft.
- **PyCharm** : IDE gratuit avec une version payante de JetBrains.
- **Spyder** : IDE gratuit et open source orienté pour la Data Science.

Les caractéristiques de Python

Python est un **langage** :

- **Interprété** et **compilé à la volée**, avec les modules **C**.
- **Typage dynamique fort**, ainsi il n'est **pas nécessaire** de **spécifier** le **type** des **variables**.
- **Orienté objet** (mais pas seulement).
- **Portable** car **compatible** avec **toutes les plateformes** actuelles.
- **Flexible**, il est utilisé de l'administration système au développement web.
- **Populaire**, il est dans le **Top 5** des **langages** les **plus utilisés** depuis des années.

L'indentation

L'**indentation** désigne les **espaces ou tabulations** situés au **début** d'une **ligne** de code.

Alors que dans d'autres langages de programmation, l'**indentation** du code ne sert qu'à **faciliter** la **lecture**, l'indentation en Python est très importante. **Python** utilise l'**indentation** pour **délimiter** les **blocs de code**.

Les avantages et inconvénients de Python

Forces

Stable

Cross-plateforme

Facile à apprendre

Grande communauté

Grand nombre de module

Faiblesses

Pas entièrement compilé, donc plus lent

L'optimisation est complexe à apprendre

Les plateformes

Il existe **différents interpréteurs** pour **Python** :

- CPython/Pypy \Rightarrow C/C++
- Jython \Rightarrow JVM
- IronPython \Rightarrow .Net

Les domaines d'exploitation de Python

Les **domaines d'application** de **Python** :

- **Sciences** : Data mining, Machine Learning, Physiques, Mathématiques, ...
- **OS** : Linux, Raspberry Pi, scripting pour l'administration système.
- **Education** : Introduction à la programmation.
- **Web** : Django, Flask, ...
- **3D CAD** : FreeCAD, pythonCAD, ...
- **Multimédia** : Kodi, ...



**Avez-vous des
questions ?**

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some solid and some hollow, connected by thin lines. The overall structure is organic and branching, resembling a molecular or biological network.

2.

Les fondamentaux de Python

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes highlighted by concentric circles. The diagram is positioned in the lower right, leaving space for the main title.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines connecting them are thin and grey, creating a dense, organic structure.

2.1

Les variables et les types de bases

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid grey and others hollow with grey outlines. The overall style is minimalist and technical.

Les variables

Une variable permet de **stocker en mémoire**, le temps que le programme s'exécute, des **données**.

Pour **stocker** en mémoire une **valeur** dans une **variable**, on utilise le **signe =**.

Exemple :

```
nomVariable = 1
```

Le nommage des variables

Il y a deux règles à respecter en ce qui concerne le nommage des variables :

- Les noms doivent commencer par une lettre minuscule, une lettre majuscule ou un underscore.
- Les noms doivent contenir uniquement les éléments précédents ainsi que des chiffres.

ATTENTION, il est donc **interdit** d'**utiliser** des **espaces**.

Les types en Python

Les types de base en Python :

- **int** : Un entier
- **float** : Un réel
- **str** : Une chaîne de caractères
- **bool** : Un booléen

Exemples :

- `compteur = 0`
- `pourcentageReduction = 3.5`
- `motATrouver = 'code'`
- `motEstTrouvé = True`

La fonction `type`

La fonction `type` permet de **connaître** le **type** d'une **variable**.

Exemple :

```
question = "Quelle est la réponse à l'univers ?"  
answer = 42  
type(question) ← <class 'str'>  
type(answer) ← <class 'int'>
```


La conversion de type

Nous pouvons utiliser les classes des types de base pour **convertir** des **valeurs** en **leur type**.

Exemple :

`int(42.5)` ← 42

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with concentric rings, and the lines are thin and grey. The diagram is partially cut off by the top and left edges of the slide.

2.2

Les opérateurs

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of interconnected nodes and lines, with some nodes having concentric circles. The diagram is partially cut off by the bottom and right edges of the slide.

Les opérateurs arithmétiques

`nomVariable = élément1 opérateur élément2`

Les opérateurs arithmétiques en Python :

- `+` : Addition
- `-` : Soustraction
- `/` : Division
- `//` : Division Euclidienne
- `*` : Multiplication
- `**` : Puissance
- `%` : Modulo

Exemples :

- `compteur = compteur + 1`
- `prime = salaire / 10`
- `resultat = var1 * 8 + var2 - var3`

Notation raccourcie

`nomVariable` opérateur= élément1

Lorsqu'on affecte le résultat de notre opération sur la même variable, on peut utiliser une notation raccourcie.

Exemples :

- `a += 8` est équivalent à `a = a+8`

- `c %= 2` est équivalent à `c = c%2`

Les opérateurs de comparaison

élément1 opérateur élément2

Les opérateurs de comparaison :

- **>** : Strictement supérieur
- **>=** : Supérieur ou égal
- **<** : Strictement inférieur
- **<=** : Inférieur ou égal
- **==** : Egal
- **!=** : Différent

Exemples :

- | | |
|-----------|----------|
| - 9 >= 10 | est faux |
| - 5 > 5 | est faux |
| - 5 != 10 | est vrai |

Les opérateurs logiques

ET (and)

A	B	A and B
1	1	1
1	0	0
0	1	0
0	0	0

OU (or)

A	B	A or B
1	1	1
1	0	1
0	1	1
0	0	0

NON (not)

A	not A
1	0
0	1

Exemples :

- not(5 <= 10) and (3 == 3) est faux
- (10 != 10) or (5 > 3) est vrai
- (10 != 10) and (5 > 3) est faux
- not(10 != 10 and 5 > 3) est vrai



2.3

Les structures conditionnelles



If

if condition:
instructions

Les **instructions** du **bloc if** seront **exécutées** uniquement si la **condition** est **vraie**. Dans le cas contraire, les instructions du bloc ne seront pas exécutées.

Exemple :

```
if age < 18:  
    print("Vous êtes mineur.")
```


If ... else

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

Les **instructions** du **bloc if** seront **exécutées** uniquement si la **condition** est **vraie**. Dans le cas contraire, les instructions du **bloc else** seront **exécutées**.

Exemple :

```
if age < 18:  
    print("Vous êtes mineur.")  
else:  
    print("Vous êtes majeur.")
```

Imbrication des instructions if ... else

```
if condition1:  
    instructions  
elif condition2:  
    instructions  
else:  
    instructions
```

Les **instructions** du **bloc if** seront **exécutées** uniquement si la **condition1** est **vraie**. Dans le cas contraire, si la **condition2** est **vraie**, les **instructions** du **bloc elif** seront **exécutées**. Sinon, les instructions du **bloc else** seront **exécutées**.

Exemple :

```
if note < 8:  
    print("Vous avez rate votre BAC.")  
elif note < 10:  
    print("Vous devez passer les rattrapages.")  
else:  
    print("Vous avez votre BAC.")
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines connecting them are thin and grey, creating a dense, organic structure.

2.4

Les entrées/sorties

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid grey and others hollow with a grey outline. The overall structure is a complex, interconnected web.

Afficher une valeur

Pour afficher un message, on utilise la fonction **print**.

```
print("texte", variable)
```

Exemple :

```
print("Age")  
print(age)  
print("Vous avez", age, "ans")
```

Les arguments de la fonction **print**

- **sep** : Caractères affichés entre chaque argument. Par défaut **' '**.
- **end** : Caractères affichés après le dernier argument. Par défaut **'\n'**.
- **file** : Flux de sortie. Par défaut **sys.stdout**.
- **flush** : Doit-on vider le tampon ? Par défaut **False**.

Exemple :

```
print("Hello", "World", sep=', ', end="!\n")
```



Hello, World!

Les strings formatées

Il est possible de construire des **strings complexes** à l'aide de la fonction **format**:

Exemple :

```
"My name is {lastname}, {firstname} {lastname}.".format(  
    firstname="James",  
    lastname="Bond")
```

└───────────> My name is Bond, James Bond.

Les strings formatées


La fonction **format** dispose d'un raccourci pratique :

Exemple :

```
firstname="James"
```

```
lastname="Bond"
```

```
f"My name is {lastname}, {firstname} {lastname}."
```



My name is Bond, James Bond.

Récupérer une valeur

Pour récupérer une valeur, on utilise la fonction **input**.

```
variable = input("texte")
```

Exemple :

```
print("Donner votre nom : ")  
nom = input()
```


Les séquences d'échappement

Les sequences d'échappement sont :

- `\n` : Nouvelle ligne
- `\r` : Retour chariot
- `\f` : Nouvelle page
- `\b` : Retour arrière
- `\t` : Tabulation horizontale
- `\v` : Tabulation vertical
- `\a` : Bip machine

Récupérer une valeur

La fonction **input**, renvoie par défaut un type **str**. Ce comportement est un soucis lorsqu'on demande à l'utilisateur un nombre. Pour obtenir le type que l'on souhaite, il est necessaire d'utiliser la fonction **eval**.

```
variable = eval(input("texte"))
```

Exemple :

```
age = eval(input("Donner votre age : "))
```



**Avez-vous des
questions ?**





Exercices

Les variables



2.4

Les structures séquentielles



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure.

2.4.1

Les séquences





Définition


Une **séquence** est un **regroupement** au sein d'une même **variable** de **plusieurs valeurs**. Ces **valeurs** seront **accessibles** par leur **position**.





Objectif

Une **séquence** a pour **objectif** d'**optimiser** certaines **opérations** tel que la recherche d'un élément, le tri de ces valeurs, le calcul de leur maximum.



Accès à un élément

L'**accès** à un **élément** d'une **séquence** se fait à l'aide de la **position** de cet élément et de l'**opérateur crochets []**.

`maSéquence[position]`

ATTENTION, la position commence à **0**.

Les 3 principaux types de séquences

- Les **listes** dont les éléments sont **quelconques** et **modifiables**.
- Les **t-uples** dont les éléments sont **quelconques** et **non modifiables**.
- Les **chaînes de caractères** dont les éléments sont des **caractères** et **non modifiables**.

Les opérations communes aux séquences

Opération	Résultat
$x \text{ in } s$	Teste si x appartient à s
$x \text{ not in } s$	Teste si x n'appartient pas à s
$s + t$	Concaténation de s et t
$s * n$ ou $n * s$	Concaténation de n copies de s
$\text{len}(s)$	Nombre d'éléments de s
$\text{min}(s)$	Plus petit élément de s
$\text{max}(s)$	Plus grand élément de s
$s.\text{count}(x)$	Nombre d'occurrences de x dans s
$s.\text{index}(x)$	Indice de x dans s



2.4.2

Les listes



Déclaration d'une liste

```
maListeVide = []
```

```
maListeVide = list()
```

```
maListeAvecUnElement = [valeur]
```

```
maListe = [valeur1, valeur2, valeur3]
```

Les listes sont **muables**

- Les **listes** sont **modifiables**, ainsi il est **possible** de **modifier**, **supprimer** ou **ajouter** des éléments.
- Une **fonction** qui a en **paramètre** une **liste**, sera en mesure de la **modifier**.

Les opérations propres aux listes

Opération	Résultat
<code>list(s)</code>	Transforme une séquence s en une liste
<code>s.append(x)</code>	Ajoute l'élément x à la fin de s
<code>s.extend(t)</code>	Étend s avec la séquence t
<code>s.insert(i,x)</code>	Insère l'élément x à la position i
<code>s.clear()</code>	Supprime tous les éléments de s
<code>s.remove(x)</code>	Retire l'élément x de s
<code>s.pop(i)</code>	Renvoie l'élément d'indice i et le supprime
<code>s.reverse()</code>	Inverse l'ordre des éléments de s
<code>s.sort()</code>	Trie les éléments de s par ordre croissant

Les opérations propres aux listes

```
maListe = [1, 3, 5]
```

```
maListe.append(7) → [1, 3, 5, 7]
```

```
maListe.extend((8, 11)) → [1, 3, 5, 7, 8, 11]
```

```
maListe.remove(8) → [1, 3, 5, 7, 11]
```

```
maListe.insert(4, 9) → [1, 3, 5, 7, 9, 11]
```


Les operations propres aux listes

```
maListe = list(range(0, 11, 2))
```

↳ [0, 2, 4, 6, 8, 10]

```
taListe = list('Wakanda Forever')
```

↳ ['W', 'a', 'k', 'a', 'n', 'd', 'a', ' ', 'F', 'o', 'r', 'e', 'v', 'e', 'r']



2.4.3

Les t-uples



Déclaration d'un t-uples

```
monTupleVide = ()
```

```
monTupleVide = tuple()
```

```
monTupleAvecUnElement = (valeur)
```

```
monTuple = (valeur1, valeur2, valeur3)
```

Les t-uples sont **immuables**

- Les **t-uples** ne sont **pas modifiables**, ainsi il est **impossible** de **modifier**, **supprimer** ou **ajouter** des éléments.
- Une **fonction** qui a en **paramètre** un **t-uple**, ne sera **pas** en mesure de le **modifier**.

Les opérations propres aux t-uples

Opération	Résultat
<code>tuple(s)</code>	Transforme une séquence s en un t-uple

```
monTuple = tuple(range(0, 11, 2))  
↳ (0, 2, 4, 6, 8, 10)
```

Les intérêts des t-uples

- Si l'on souhaite définir une **séquence non modifiable**, utiliser un **t-uple sécurise** votre code (par exemple, définir la largeur et longueur de votre fenêtre).
- **Itérer** sur les éléments d'un **t-uple** est **plus rapide** que sur ceux d'une **liste**.
- Une **fonction** qui **retourne** « **plusieurs valeurs** », retourne en fait un **t-uple**.



2.4.4

Le slicing



L'accès aux éléments (slicing)

Le **slicing** peut être appliqué sur **toutes** les **séquences**.

Opération	Résultat
<code>s[i]</code>	i -ème élément de s
<code>s[i:j]</code>	Sous-séquence de s constituée des éléments entre le i -ème (inclus) et le j -ème (exclus)
<code>s[i:j:k]</code>	Sous-séquence de s constituée des éléments entre le i -ème (inclus) et le j -ème (exclus) pris avec un pas de k

L'accès aux éléments (slicing)

`monTuple = (10, 20, 30, 40, 50, 60)`

`monTuple[3:]` \longrightarrow `(40, 50, 60)`

`monTuple[1:4]` \longrightarrow `(20, 30, 40)`

`monTuple[1::2]` \longrightarrow `(20, 40, 60)`

Modification à l'aide du “slicing”

La **modification** à l'aide du **slicing** peut être appliqué **uniquement** sur les **listes**.

Opération	Résultat
$s[i] = x$	Remplacement de l'élément s[i] par x
$s[i:j] = t$	Remplacement des éléments de s[i:j] par ceux de la séquence t
$\text{del}(s[i:j])$	Suppression des éléments de s[i:j]
$s[i:j:k] = t$	Remplacement des éléments de s[i:j:k] par ceux de la séquence t
$\text{del}(s[i:j:k])$	Suppression des éléments de s[i:j:k]

Modification à l'aide du "slicing"

```
maListe = [1, 2, 3, 4, 5]
```

```
maListe[2:4] = (6, 'x', 7) → [1, 2, 6, 'x', 7, 5]
```

```
maListe[1:6:2] = 'tes' → [1, 't', 6, 'e', 7, 's']
```



2.4.5

Les sets



Déclaration d'un set

```
monEnsembleVide = set()
```

```
monEnsembleAvecUnElement = {valeur}
```

```
monEnsemble = {valeur1, valeur2, valeur3}
```

Les sets

- Les **ensembles** sont des **collections non ordonnées** et **sans répétitions**.
- Les **ensembles** sont **modifiables**, en revanche il est possible d'utiliser les **frozenset** pour les rendre **non modifiables**.

Les opérations des sets

Opération	Résultat
<code>set(s)</code>	Transforme une séquence s en un ensemble.
<code>s.add(x)</code>	Ajoute l'élément x à l'ensemble s .
<code>s.update(t)</code>	Étend s avec la séquence t .
<code>s.discard(x)</code>	Supprime l'élément x à l'ensemble s .
<code>s.remove(x)</code>	Supprime l'élément x à l'ensemble s , en levant une exception si x n'est pas présent dans s .

Les opérations des sets (suite)

Opération	Résultat
<code>s1.union(s2)</code> ou <code>s1 s2</code>	Créer un set avec les éléments de s1 et s2 .
<code>s1.intersection(s2)</code> ou <code>s1 & s2</code>	Créer un set avec les éléments communs de s1 et s2 .
<code>s1.difference(s2)</code> ou <code>s1 - s2</code>	Créer un set avec les éléments de s1 non compris dans s2 .
<code>s1.symmetric_difference(s2)</code> ou <code>s1 ^ s2</code>	Créer un set avec les éléments de s1 et s2 , mais qui ne sont pas dans les deux à la fois.
<code>s1.issubset(s2)</code> ou <code>s1 <= s2</code>	Renvoie true si s1 est un sous-ensemble de s2 .
<code>s1.issuperset(s2)</code> ou <code>s1 >= s2</code>	Renvoie true si s2 est un sous-ensemble de s1 .

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark gray, while others are hollow with a light gray outline. The lines connecting them are thin and light gray, creating a dense, organic structure that tapers off towards the right.

2.4.6

Les dictionnaires



Déclaration d'un dictionnaire

```
monDictionnaireVide = {}
```

```
monDictionnaireVide = dict()
```

```
monDictionnaireAvecUnElement = {clé:valeur}
```

```
monDictionnaire = {clé1:valeur1, clé2:valeur2}
```

Les dictionnaires

- Les **dictionnaires** sont des **collections** d'objets **non-ordonnées**.
- Un **dictionnaire** est composé d'**éléments** et chaque **élément** se **compose** d'une **paire clé: valeur**.
- Les **dictionnaires** sont des **objets modifiables**.
- Un **dictionnaire** peut contenir des **objets** de **tous les types**, mais les **clés** doivent **être uniques**.

Les opérations des dictionnaires

Opération	Résultat
<code>dict(s)</code>	Transforme une séquence s de paire clé-valeur en un dictionnaire.
<code>d.get(k)</code>	Retourne la valeur v se trouvant à la clé k du dictionnaire d . Renvoie None si la clé k n'existe pas.
<code>d.pop(k)</code>	Supprime l'élément qui possède la clé k , tout en renvoyant sa valeur v .
<code>d.popitem()</code>	Supprime le dernier élément, tout en renvoyant un tuple contenant sa clé et sa valeur.
<code>d.clear()</code>	Vide le dictionnaire d .
<code>del d</code>	Supprime le dictionnaire d .



Avez-vous des questions ?



Exercices

Les collections

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure that tapers off towards the right.

2.5

Les structures itératives

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of interconnected nodes and lines, with some nodes being solid dark grey and others being hollow with a light grey outline. The lines are thin and light grey, forming a complex, web-like pattern that tapers off towards the left.

Boucles bornées et non bornées

Boucle bornée

Quand on **sait combien** de **fois** doit avoir lieu la **répétition**, on utilise généralement une boucle **for**.

Boucle non bornée

Si on **ne connaît pas** à l'avance le nombre de **répétitions**, on choisit une boucle **while**.

La boucle **for**

```
for compteur in range:  
    instructions
```

Les **instructions** du **bloc for** seront **exécutées** autant de fois que la **range** le permet. On dit qu'on réalise une **itération**, à **chaque fois** que les **instructions** de la boucle sont **exécutées**.

Exemple :

```
for i in range(5):  
    print(i)
```

Parcours d'une séquence avec un **for**.

```
for x in maSequence:  
    instructions
```

```
for i, x in enumerate(maSequence):  
    instructions
```

```
for i in range(len(maSequence)):  
    instructions
```

```
for x, y in zip(maSequence1, maSequence2):  
    instructions
```

Parcours d'un dictionnaire avec un **for**.

```
for key in monDictionnaire:  
    instructions
```

```
for key, value in monDictionnaire.items():  
    instructions
```

La boucle **while**

while condition:
instructions

Les **instructions** du **bloc while** seront **exécutées**
tant que la condition est **vraie**.

ATTENTION à la boucle infinie !!!

Exemple :

```
i = 0  
while i < 5:  
    print(i)  
    i += 1
```

Break

L'instruction **break** permet de « **casser** » l'**exécution** d'une **boucle** (**while** ou **for**). Elle fait **sortir** de la **boucle** et passer à l'instruction suivante.

Continue

L'instruction **continue** permet de **passer prématurément** au **tour** de **boucle suivant**(**while** ou **for**). Elle fait **continuer** sur la **prochaine itération** de la boucle.

L'instruction **else** après une boucle

```
for compteur in range:  
    instructions  
else:  
    instructions
```

Les **instructions** du **bloc else** seront **exécutées** uniquement si la **boucle arrive à son terme** « **normalement** » (pas de **break**).

Exemple :

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

La syntaxe pour définir une liste en compréhension

Le **but** est de **construire** une **liste** à partir d'une **séquence** déjà **existant**.

```
[expression for x in maSequence if conditions]
```


La syntaxe pour définir une liste en compréhension

```
t = tuple(range(5))
```

```
maListe1 = [x ** 2 for x in t]
```

```
print(maListe1) → [0, 1, 4, 9, 16]
```

```
maListe2 = [x for x in t if x%2 == 0]
```

```
print(maListe2) → [0, 2, 4]
```



**Avez-vous des
questions ?**



Exercices

Les structures itératives



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure that resembles a molecular or biological network.

2.6

Les fonctions

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It consists of a cluster of nodes (solid dark grey circles and hollow light grey circles) connected by thin, light grey lines, forming a complex, interconnected web.



2.6.1

Les généralités

Le principe

Une **fonction** est un **bloc d'instructions réalisant une certaine tâche**.

Elle possède un **nom** et est **exécutée** lorsqu'on **l'appelle**.

Un **programme bien structuré** contiendra une **fonction** dite « **principale** », et **plusieurs fonctions** dédiées à des fonctionnalités spécifiques.

Quand une **fonction** dite « **principale** » fait appel à une **autre fonction**, elle **suspend** son **déroulement**, et **exécute l'autre fonction**, puis **reprend** ensuite son **fonctionnement**.

Les avantages

L'utilisation de **fonction** possède **3 avantages** :

- **Eviter** la **duplication** de code.
- **Favoriser** la **réutilisation**.
- **Améliorer** la **conception** (en réduisant la complexité).

Les paramètres

Une **fonction** sert donc à effectuer un **traitement générique**.

Ce **traitement** porte sur des **données**, dont la **valeur** pourra ainsi **changer** d'un **appel à l'autre** de la **fonction**.

Ces **données** sont **appelées paramètre**.

Lors de l'**implémentation** d'une **fonction**, on va donc préciser la **liste de tous les paramètres** qu'elle va utiliser.

Les paramètres

Lors de l'**implémentation** d'une **fonction**, on va donc préciser la **liste de tous les paramètres** qu'elle va utiliser.

Lors de l'**utilisation** d'une **fonction**, on va alors **préciser** la **valeur** de **chacun** des **paramètres** qu'elle possède.

Les paramètres par défaut

Les **paramètres** d'une **fonction** peuvent comporter des **valeurs** par **défaut**.

Lorsqu'on **appelle** une **fonction**, on a **deux cas possibles** :

- On **ne précise pas** de **valeurs** pour les **paramètres** et la **fonction** utilise celles par **défaut**.
- On **précise** des **valeurs** ce sont celles-ci qui sont **utilisées**.

ATTENTION !

Les **paramètres** par **défaut** sont **obligatoirement** positionnés à **droite** des **paramètres**.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure that tapers off towards the right.

2.6.2

La portée des variables



Les variables locales

Pour réaliser sa **tâche**, une **fonction** aura besoin de ses **propres variables**. On parle alors de « **variables locales** ».

Ces **variables** ne sont **accessibles** qu'**au sein** de la **fonction** qui les définit.

Les variables globales

Une **fonction** reçoit donc des **données** à traiter, les **paramètres**, et pour ce faire peut **avoir besoin** de **variables locales**.

Une **fonction** peut également **manipuler directement** des **variables** définies par le **programme principal**. On parle alors de « **variables globales** ».

ATTENTION !

Il s'agit souvent d'une **mauvaise pratique** car cela **limite** les **performances** et la **réutilisabilité** du code.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and grey, creating a dense, organic structure that tapers off towards the right.

2.6.3

L'implémentation

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes and connecting lines, with some nodes being solid dark grey and others being hollow light grey circles. The lines are thin and grey, forming a web-like pattern that extends from the right edge towards the center.

Syntaxe pour déclarer une fonction

Il existe **deux types** de **fonction** : Celles qui **retournent** une **valeur** et celles qui ne **retournent rien**.

Exemples :

```
def maFonction(param1, param2 = 0):  
    instructions  
    return monResultat
```

```
def maFonction(param1, param2):  
    instructions
```

Syntaxe pour appeler une fonction

Comme une instruction prédéfinie du langage. On appelle la **fonction** par son **nom**, en lui **passant** autant de **paramètres** qu'elle en **possède**.

Exemples :

```
maFonction(42)
```

```
a, b = 5, 10
```

```
resultat = maFonction(a, b)
```


Une particularité de Python

Une **fonction** peut **retourner plusieurs valeurs**, il suffit de **séparer** celles-ci par des **virgules**.

Exemples :

```
def maFonction(param1, param2):  
    instructions  
    return monResultat1, monResultat2
```

```
a, b = 5, 10  
longueur, largeur = maFonction(a, b)
```



**Avez-vous des
questions ?**



Exercices

Les fonctions



2.7

Les fichiers

La fonction **open**

```
open(chemin, mode, encoding="utf8")
```

La fonction **open** permet d'ouvrir un fichier. Celle-ci attend **deux arguments** : un **chemin d'accès** vers un fichier et un **mode** qui détermine le **type** (texte ou binaire) et la **nature** des opérations qui seront réalisées sur le fichier (lecture, écriture ou les deux). Elle **retourne** une variable qui contient le contenu du fichier.

Le mode d'ouverture

Le **mode** est une **chaîne de caractères** composés d'une ou plusieurs lettres qui **décrit** le **type** du **flux** et la **nature** des **opérations** qu'il doit **réaliser**.

Mode	Type(s) d'opération(s)	Effets
r	Lecture	Rien
r+	Lecture et écriture	
w	Ecriture	Si le fichier n'existe pas, il est créé. Si le fichier existe, son contenu est effacé.
w+	Lecture et écriture	
a	Ecriture	Si le fichier n'existe pas, il est créé. Si le fichier existe, on écrit à la suite.
a+	Lecture et écriture	

La fonction **close**

close()

La fonction **close** permet de fermer un fichier.

Exemple :

```
fichier = open(file, "r")  
instructions  
fichier.close()
```

Lire le contenu d'un fichier

`read(size)`

La fonction **read** permet de **lire** le contenu d'un **fichier**.

L'argument **size** permet de **définir** le **nombre** de **caractère** à **lire**, il est **facultatif**.

Par défaut, la fonction **read** lit l'**intégralité** du **fichier**.

`readline()`

La fonction **readline** permet de **lire** le contenu d'un **fichier ligne par ligne**.

Ecrire dans un fichier

`write(variable)`

La fonction **write** permet d'**écrire** le contenu d'une **variable** dans un **fichier**.

`writelines(list)`

La fonction **writelines** permet d'**écrire** le contenu d'une **liste** dans un **fichier**.

Le mot clé **with**

Exemples :

```
fichier = open(file, "a+")  
texte = fichier.read()  
fichier.write(texte)  
fichier.close()
```

```
with open(file, "a+") as file:  
    texte = file.read()  
    file.write(texte)
```



**Avez-vous des
questions ?**





Exercices

Les fichiers

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines connecting them are thin and grey, creating a dense, organic structure.

3.


La P00

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid grey and others hollow with grey outlines. The overall shape is more triangular and less spread out than the top-left diagram.



Définition


La **POO** repose sur le **concept** de **classe** qui sont des **entités** qui vont pouvoir **posséder** un ensemble d'**attributs** et de **méthodes** qui leur sont propres.





Objectif

La **POO** a pour **objectif** de **rendre** nos **scripts** plus **clairs**, **mieux structurés**, **plus modulable** et **plus facile à maintenir** et à **débugger**.



La notion de classe et d'objet

Une **classe** est un “**moule**” qui va nous **permettre** de **créer** des **objets**. Chaque **objet** aura les **attributs** de **sa classe**, mais nous pourrons les **personnaliser**.

Les **classes** sont la **base** de la **POO**, car elles permettent de mettre en place les **trois concepts fondamentaux** de cette dernière, à savoir :

- L'**encapsulation**
- L'**heritage**
- Le **polymorphisme**

Création d'une classe

```
class CompteBancaire:  
    id = 1  
    solde = 126
```



Création de la classe

```
    def setSolde(self, n):  
        self.solde = n
```

```
monCompte = CompteBancaire()
```



Création de l'objet /
Instanciation de la classe

L'opérateur .

L'**accès** à un **attribut** d'une **classe** se réalise à l'aide de l'**opérateur .** suivi du **nom** de l'**attribut**. Ce dernier s'utilise comme une variable classique.

Le fonctionnement est le même pour l'accès aux **méthodes**.

Exemple :

```
monCompte.setSolde(5000)  
monCompte.id ← 1
```

Les constructeurs

Il existe une méthode particulière qui permet « **d'initialiser** » nos **objets**. On appelle cette **méthode** un **constructeur** et elle se code `__init__()`.

La méthode `__init__()` va être **automatiquement exécutée** au moment de l'**instanciation** d'une **classe**. Cette fonction va pouvoir **recevoir** des **arguments** pour « **personnaliser** » nos **objets**.

Exemple :

```
class CompteBancaire:
    def __init__(self, id, prenom, solde):
        self.id = id
        self.prenom = prenom
        self.solde = solde
```

```
monCompte = CompteBancaire(125, "Florent", 550)
```

Les méthodes “magiques”

Les **méthodes** “**magiques**” sont les méthodes **prédéfinies** par **python**, à l'image de la méthode **`__init()`**. Elles sont appelées **automatiquement** par l'**interpréteur** et elles sont toujours **définies** avec **`__`**.

Voici un exemple de **méthode magiques** :

Méthode	Fonctionnement
<code>__str__()</code>	Définir la représentation de l' objet sous forme de string .
<code>__len__()</code>	Définir la longueur de l' objet .
<code>__getitem__()</code>	Accéder à un élément de l'objet à l'aide de l' opérateur <code>[]</code> .
<code>__add__()</code>	Ajouter deux objets ensemble.

L'encapsulation

L'**encapsulation** décrit l'idée « d'**enfermer** » les **attributs** et les **méthodes** au **sein** d'une **classe**. Cela **limite l'accès aux données de la classe** en dehors de cette dernière, dans le but de les **protéger**.

La visibilité des données

La majorité des langages de programmation ont **3 types** de **visibilité** pour les **données de classe** : **private**, **protected** et **public**.

Attention, la notion de visibilité n'existe pas en Python.

Exemple :

```
class CompteBancaire:
    def __init__(self, solde):
        self.solde = solde

    def getSolde(self):
        return self.solde

    def setSolde(self, n):
        self.solde = n
```

L'héritage

En **POO**, « **hériter** » signifie « **avoir également accès à** ».

La notion d'**héritage** va être particulièrement intéressante lorsqu'on va l'**implémenter** entre **deux classes**. En **POO**, nous allons pouvoir créer des **classes « enfants »** à partir de **classes de base** ou « **classes parentes** ».

Exemple :

```
class CompteBancaire:
    numeroCompte = 0

    def __init__(self, solde):
        self.__class__.numeroCompte += 1
        self.solde = solde

class CompteEpargne(CompteBancaire):
    estEpargne = True
```


La surcharge des méthodes de classe

« **Surcharger** » une **méthode** signifie la **redéfinir** d'une façon différente. En Python, les **classes filles** vont pouvoir **surcharger** les **méthodes héritées** de leur **classe parent**.

Souvent lors de la **redéfinition** d'une **méthode**, nous souhaitons **utiliser** la **méthode de base**. Pour se faire, nous allons l'**appeler directement** avec la **syntaxe** suivante : **NomClasseDeBase.nomMethode()**.

Exemple de surcharge

```
class CompteBancaire:
    def __init__(self, solde):
        self.solde = solde

    def __str__(self):
        return "Le compte " + str(self.numeroCompte) + " a un solde de " + str(self.solde)

class CompteEpargne(CompteBancaire):
    estEpargne = True

    def __init__(self, solde, taux):
        CompteBancaire.__init__(self, solde)
        self.taux = taux

    def __str__(self):
        return "Le compte d'épargne " + str(self.numeroCompte) + " a un solde de " + str(self.solde)
```

Le polymorphisme

« **Polymorphisme** » signifie littéralement « **plusieurs formes** ». En **POO**, le **polymorphisme** est un concept qui fait référence à la **capacité** d'un **attribut**, d'une **méthode** ou d'un **objet** à prendre plusieurs formes. Autrement dit, à sa **capacité** de **posséder plusieurs définitions** différentes.

Exemple de polymorphisme

```
class CompteBancaire:
    def __init__(self, solde):
        self.solde = solde
    def connaitrePlafond(self):
        pass

class CompteEnfant(CompteBancaire):
    def __init__(self, solde):
        CompteBancaire.__init__(self, solde)
        self.plafond = 50
    def connaitrePlafond(self):
        print("Vous êtes limité à", self.plafond, "€.")

class CompteEtudiant(CompteBancaire):
    def __init__(self, solde):
        CompteBancaire.__init__(self, solde)
        self.plafond = 500
    def connaitrePlafond(self):
        print("Vous êtes limité à", self.plafond, "€.")
```

Le duck typing

Il est possible d'**appliquer** le **polymorphisme** à des **types de base** en Python.
On nomme cela le **duck typing**.

Exemple :

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

```
resultat1 = ajouter(5, 10)           ← 15  
resultat2 = ajouter("Hello", " World") ← Hello World  
resultat3 = ajouter([0, 2, 4], [1, 3, 5]) ← [0, 2, 4, 1, 3, 5]
```



**Avez-vous des
questions ?**



Exercices

La POO

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, while others are smaller and solid. The lines are thin and gray, connecting the nodes in a non-linear fashion.

4.

Les expressions régulières

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being larger and having concentric circles, and others being smaller and solid. The lines are thin and gray.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark gray, while others are hollow with a light gray outline. The lines connecting them are thin and light gray, creating a dense, organic structure that tapers off towards the right.

4.1


Les généralités

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid dark gray and others being hollow with a light gray outline. The lines are thin and light gray, forming a complex, interconnected web that tapers off towards the left.



Définition

Une **expression régulière** (**Regex**) est une **suite de caractères** qui a pour **but** de **décrire** un **fragment de texte**.



Les expressions régulières

Une **expression régulière** est une **suite de caractères** que l'on appelle **motif** (*pattern* en anglais), **motif** qui est **constitué** de deux **types** de **caractères** :

- Les **caractères** dits **normaux**.
- Les **métacaractères** ayant une **signification particulière**.

Les métacaractères – Partie 1

Métacaractère	Description
^	Début de chaîne de caractères ou de ligne.
\$	Fin de chaîne de caractères ou de ligne.
.	N'importe quel caractère (sauf le saut de ligne).
[ABC]	Le caractère A, B, ou C (un seul caractère).
[A-Z]	N'importe quelle lettre majuscule.
[a-z]	N'importe quelle lettre minuscule.
[0-9]	N'importe quel chiffre.
[A-Za-z0-9]	N'importe quel caractère alphanumérique.
[^AB]	N'importe quel caractère sauf A et B.
\	Caractère d'échappement.

Les métacaractères – Partie 2

Métacaractère	Description
*	0 à n fois le caractère ou l'expression précédent.
+	1 à n fois le caractère ou l'expression précédent.
?	0 à 1 fois le caractère ou l'expression précédent.
{n}	n fois le caractère ou l'expression précédent.
{n, m}	n à m fois le caractère ou l'expression précédent.
{n, }	Au moins n fois le caractère ou l'expression précédent.
{,m}	Au plus n fois le caractère ou l'expression précédent.
(AB CD)	Les chaînes de caractères AB ou CD.
\d	N'importe quel chiffre (équivalent à [0-9]).
\w	N'importe quel caractère alphanumérique et _ (équivalent à [0-9A-Za-z_]).
\s	N'importe quel « espace blanc » (équivalent à [\t\n\r\f]).

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with solid outlines and others with dashed outlines, connected by thin, light gray lines. The overall structure is dense and organic, resembling a molecular or biological network.

4.2

Le module *re*



Le module **re**

Le **module** **re** permet l'**utilisation** d'**expressions régulières** avec **Python**.

La fonction `search()`

Dans le **module** `re`, la fonction `search()` est incontournable. Elle permet de **rechercher** un **motif**, au **sein** d'une **chaîne de caractères**.

Exemple :

```
import re
animaux = "girafe tigre singe"
re.search("tigre", animaux)
if re.search("tigre", animaux):
    print("OK")
```


Les fonctions `match()` et `fullmatch()`

Il existe aussi la fonction `match()` dans le module `re` qui fonctionne sur le modèle de `search()`. La différence est que `match()` ne fonctionne que si la **regex** correspond au **début** de la chaîne de caractères.

Exemple :

```
import re
animaux = "girafe tigre singe"
re.search("tigre", animaux) ← True
```

```
re.match("tigre", animaux) ← False
```


```
animaux = "tigre singe"
re.match("tigre", animaux) ← True
```

La fonction `findall()`

Pour **récupérer chaque zone**, vous pouvez utiliser la **méthode `findall()`** qui **renvoie** une **liste des éléments** en **correspondance**.

Exemple :

```
import re  
chaine = "pi vaut 3.14 et e vaut 2.72"  
resultat = re.findall("[0-9]+\.[0-9]+", chaine)
```



`['3.14', '2.72']`

La fonction `sub()`

Enfin, la **méthode** `sub()` permet d'effectuer des **remplacements**. Par défaut la **méthode** `sub(chaine1, chaine2)` remplace toutes les **occurrences trouvées** par l'**expression régulière** dans *chaine2* par *chaine1*. Si vous souhaitez ne remplacer que les ***n* premières occurrences**, utilisez l'**argument** `count=n`.

Exemple :

```
import re
chaine = "pi vaut 3.14 et e vaut 2.72"
resultat = re.sub(".", ",", chaine)
```

└───────────▶ pi vaut 3,14 et e vaut 2,72



Avez-vous des questions ?



Exercices

Les regex

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting a hierarchical or multi-layered structure. The lines are thin and gray, connecting the nodes in a non-linear fashion.

5.

La récursivité

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being larger and having concentric circles, indicating a recursive or self-referential structure.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and edges. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting a hierarchical or multi-layered structure. The edges are thin lines connecting the nodes, creating a dense, organic pattern.

5.1

La récursivité “simple”



Définition



Une fonction est dite **récursive** si elle **s'appelle elle même**.

Objectif



Pour effectuer une tâche ou un calcul, on se ramène à la réalisation d'une tâche similaire mais de **complexité moindre**. On recommence jusqu'à obtenir une **tâche élémentaire**.

Attention



Il est indispensable de prévoir une condition d'arrêt à la récursion sinon la programme ne se termine jamais.



Exemple avec le calcul de la factorielle

Pour rappel : $n! = 1 \times 2 \times 3 \times \dots \times n-1 \times n$.

On obtient facilement la relation suivante : $n! = n \times (n-1)!$

Ainsi en calculant $(n-1)!$ on sera en mesure d'obtenir $n!$

Mais $(n-1)! = (n-1) \times (n-2)!$, on est donc ramené au calcul de $(n-2)!$

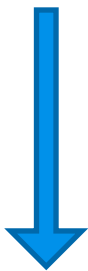
Ainsi de suite jusqu'à $1!$ dont on connaît la valeur : 1

Exemple avec le calcul de la factorielle

```
def factorielle(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        return n * factorielle(n-1)
```

Exemple avec le calcul de la factorielle

Déroulement du programme dans le cas où $n = 4$:


$$\text{factorielle}(4) = 4 \times \text{factorielle}(3)$$


$$\text{factorielle}(3) = 3 \times \text{factorielle}(2)$$

$$\text{factorielle}(2) = 2 \times \text{factorielle}(1)$$

$$\text{factorielle}(1) = 1$$

$$\text{factorielle}(4) = 4 \times 6 = 24$$

$$\text{factorielle}(3) = 3 \times 2 = 6$$

$$\text{factorielle}(2) = 2 \times 1 = 2$$


Les avantages et les inconvénients de la récursivité

Avantages

Elle est très utile pour concevoir des algorithmes sur des structures complexes comme les **listes**, les **arbres** et les **graphes**.

Technique de programmation plus **lisible**.

Inconvénients

Elle est plus **gourmande** en **espace mémoire**, pouvant même générer des débordements de capacité.

Technique de programmation plus **complexe**.

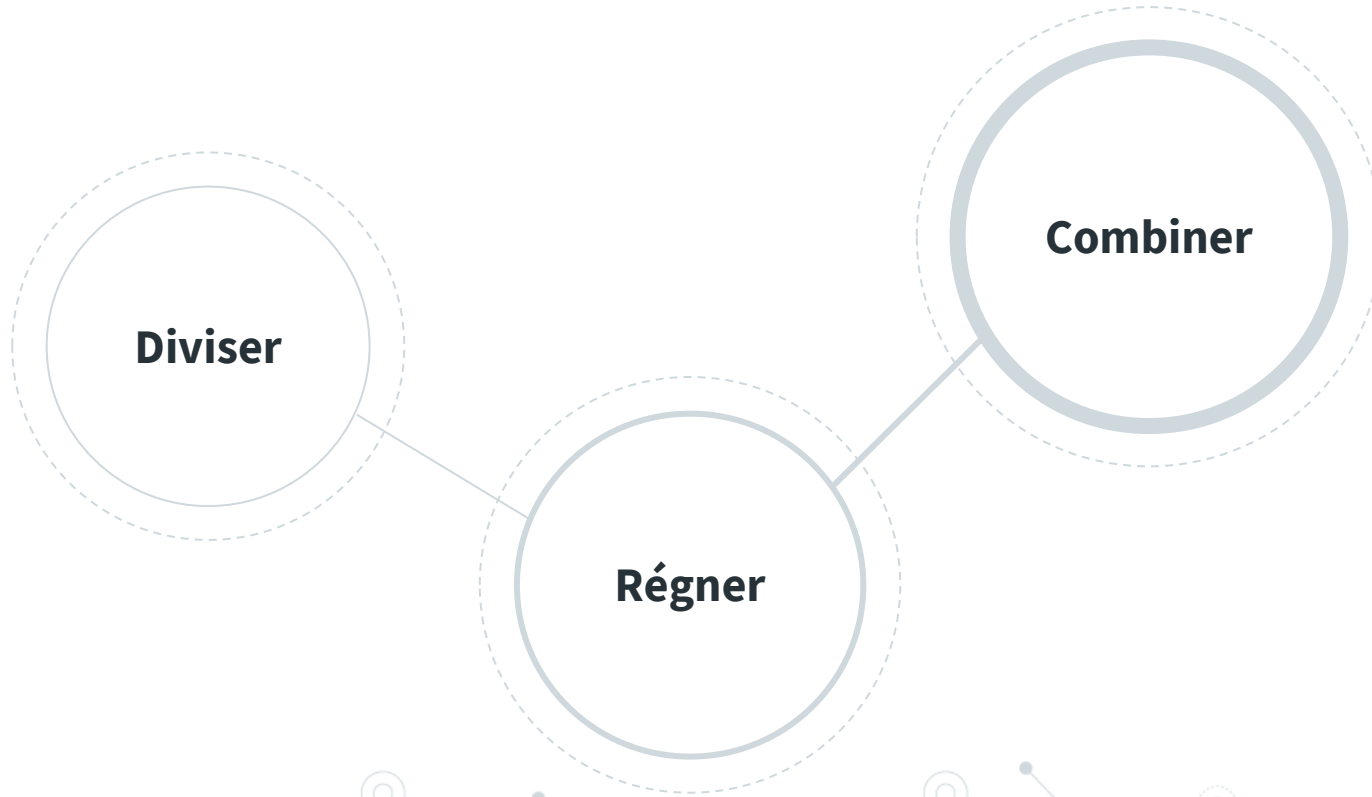
A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines connecting them are thin and grey, creating a dense, organic structure.

5.2

Paradigme “diviser pour régner”

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid grey and others hollow with grey outlines. The overall shape is more triangular and less spread out than the top-left diagram.

Le principe du paradigme “diviser pour régner”



Paradigme “diviser pour régner”

Enoncé : Calculer le maximum d'un tableau de nombres

Résolution :

1. Diviser le tableau en deux sous-tableaux en le « coupant » par la moitié.
2. Rechercher le maximum de chacune de ces sous-tableaux.
3. Comparer les résultats obtenus.

Paradigme “diviser pour régner”

```
def maximum(tab, d, f):  
    if d == f:  
        return tab[d]  
  
    m = (d+f) // 2  
    x = maximum(tab, d, m)  
    y = maximum(tab, m+1, f)  
  
    return x if x > y else y
```



**Avez-vous des
questions ?**



Exercices

La récursivité

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels of connectivity or importance. The lines are thin and gray, creating a mesh-like structure.

6.

Les bases de données

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being larger and more prominent than others. The overall style is minimalist and technical.

Définition



SQLite est un **système de gestion de base de données relationnelle embarqué.**



Les caractéristiques

- Aucun serveur requis.
- Stockage dans un seul fichier.
- Idéal pour des applications légères.

Les avantages de l'intégration SQLite avec Python

- **Simplicité** : **SQLite** est **facile** à **utiliser** et s'intègre **bien** avec **Python**.
- **Légèreté** : Parfait pour les **projets** nécessitant une **gestion de données simple** **sans** **nécessité** d'un **serveur de base de données**.

Comment utiliser SQLite avec Python

- **'sqlite3'** est le **module** intégré à **Python** pour **travailler** avec **SQLite**.
- **Pas besoin** d'installation **externe**, il est **inclus** dans la **bibliothèque standard**.

Connexion à une base de données SQLite

Exemple :

```
import sqlite3
conn = sqlite3.connect('ma_base_de_donnees.db')
...
conn.close()
```

Si la **base de données** n'existe pas, un **fichier** sera **créé** dans le dossier de votre programme. Dans le cas contraire, le **fichier déjà existant** sera **réutilisé**.

Manipulation de données – Création de Table

Exemple :

```
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS users(
    id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
    name TEXT,
    age INTERGER
)
""")
conn.commit()
```

Manipulation de données – Supression de Table

Exemple :

```
cursor = conn.cursor()  
cursor.execute("""DROP TABLE users""")  
conn.commit()
```

Manipulation de données – Insertion des données

Exemple :

```
cursor = conn.cursor()
cursor.execute("""INSERT INTO users(name, age) VALUES(?, ?)""",
("olivier", 30))
conn.commit()
```

Manipulation de données – Récupération des données

Exemple :

```
cursor = conn.cursor()  
cursor.execute("""SELECT * FROM users""")  
users = cursor.fetchall()
```

Manipulation de données – Récupération des données

Exemple :

```
cursor = conn.cursor()
cursor.execute("""SELECT * FROM users WHERE NAME = 'Jones'""")
jones = cursor.fetchone()
```

Manipulation de données – Modification des données

Exemple :

```
cursor = conn.cursor()  
cursor.execute("""UPDATE users SET age = ? WHERE id = 2""",  
(31,))
```


Rollback

Exemple :

```
conn.rollback()
```

La **commande rollback** permet de **revenir au dernier commit**.



**Avez-vous des
questions ?**





Exercices


Les bases de données



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting a hierarchical or central structure. The lines are thin and gray, connecting the nodes in a non-linear fashion.

7.

La mesure de performance

A decorative network diagram in the bottom-right corner, similar to the one in the top-left, showing a cluster of interconnected nodes and lines, with some nodes being larger and more prominent than others.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and grey, creating a dense, organic structure that tapers off towards the right.

7.1

Les types de temps

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid dark grey and others being hollow with a light grey outline. The lines are thin and grey, forming a complex, interconnected web that tapers off towards the left.

Temps processeur

Le **temps processeur** correspond au temps où le programme est effectivement **en cours d'exécution sur le processeur**.

Temps utilisateur

Le **temps utilisateur** correspond à la **quantité de temps** qui s'est écoulée **entre le démarrage du programme et la fin de son exécution**.

Temps de réponse

Le **temps utilisateur** correspond au temps qu'il faut attendre **entre** le **lancement du programme** et l'**affichage** de ce dernier.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and edges. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels of connectivity or importance. The edges are thin lines connecting the nodes, forming a dense, branching structure.

7.2

Calcul des temps d'exécution

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a complex web of interconnected nodes and edges, with some nodes being larger and having concentric circles, indicating a hierarchical or weighted network structure.

Outil du système d'exploitation

Si vous exécutez le programme en **ligne de commande**, et grâce à certaines **commandes** vous pourrez récupérer **les temps d'exécutions**.

Nous n'aborderons pas cette méthode dans ce cours, car les commandes sont dépendantes des systèmes d'exploitations et elle ne permet pas la mesure d'une partie du programme.



Le module *time*

Pour réaliser le calcul des temps d'exécution, nous allons **ajouter du code** directement dans notre programme python à l'aide du module ***time***.

Ce dernier va nous permettre d'obtenir le **temps utilisateur** ainsi que le **temps processeur**.



Le module *time* – temps utilisateur

```
import time
```

```
start = time.time()
```

```
Fib(25)
```

```
end = time.time()
```

```
print(end - start, "s")
```

Le module *time* – temps processeur

```
import time
```

```
start = time.process_time ()
```

```
Fib(25)
```

```
end = time.process_time ()
```


```
print(end - start, "s")
```



Le module *time*

Le soucis de la méthode précédente est qu'on effectue qu'une seule mesure. Cette dernière est impactée par l'occupation de l'ordinateur (temps processeur).

Il va être nécessaire d'effectuer la mesure à plusieurs reprises, et d'utiliser des outils statistiques pour interpréter les résultats.



Le module *time* – temps utilisateur

```
import time
import statistics

mesures = []

for i in range(100):
    start = time.time()
    Fib(25)
    end = time.time()
    mesures.append(end - start)

mean = statistics.mean(mesures)
stdev = statistics.stdev(mesures)
```

Le module *time* – temps processeur

```
import time
import statistics

mesures = []

for i in range(100):
    start = time.process_time()
    Fib(25)
    end = time.process_time()
    mesures.append(end - start)

mean = statistics.mean(mesures)
stdev = statistics.stdev(mesures)
```




Le module *time*

Au final le code que nous avons ajouté pour réaliser le calcul des temps d'exécution est **assez intrusif**.

Pour simplifier la procédure et rendre le code moins intrusif nous allons utiliser un autre module python qui est ***timeit***.

Le module *timeit* – temps utilisateur

```
import timeit
```

```
import time
```

```
import statistics
```

```
N = 100
```

```
fct = lambda: fib(25)
```

```
measures = timeit.repeat(fct, repeat=N, number=1, timer=time.time)
```

```
mean = statistics.mean(measures)
```

```
stdev = statistics.stdev(measures)
```

Le module *timeit* – temps processeur

```
import timeit
```

```
import time
```

```
import statistics
```

```
N = 100
```

```
fct = lambda: fib(25)
```

```
measures = timeit.repeat(fct, repeat=N, number=1, timer=time.process_time)
```

```
mean = statistics.mean(measures)
```

```
stdev = statistics.stdev(measures)
```



Le module *timeit*

Le module **timeit** permet d'effectuer exactement les **mêmes opérations** que le module **time**, tout en étant **moins intrusif**.

Il est également possible de **simplifier** encore plus le code en utilisant la fonction **timeit** à place de **repeat**, mais le calcul des statistiques est plus “complexe”.

Le module *timeit* – la fonction *timeit*

```
import timeit
import time

result = timeit.timeit(
    'Fib(15)',
    number=100,
    globals=globals(),
    timer=time.process_time)
```



Avez-vous des questions ?



Exercices


La mesure de performance



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines are thin and grey, connecting the nodes in a non-linear fashion. The overall shape of the network is roughly triangular, pointing towards the top-left corner of the slide.

8.

La programmation asynchrone

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It consists of a cluster of nodes connected by lines. The nodes are small circles, some solid grey and some hollow with a grey outline. The lines are thin and grey, forming a complex, interconnected web. The network is located in the bottom-right corner of the slide.

Définition



La **programmation asynchrone** permet de **lancer plusieurs processus simultanément**.

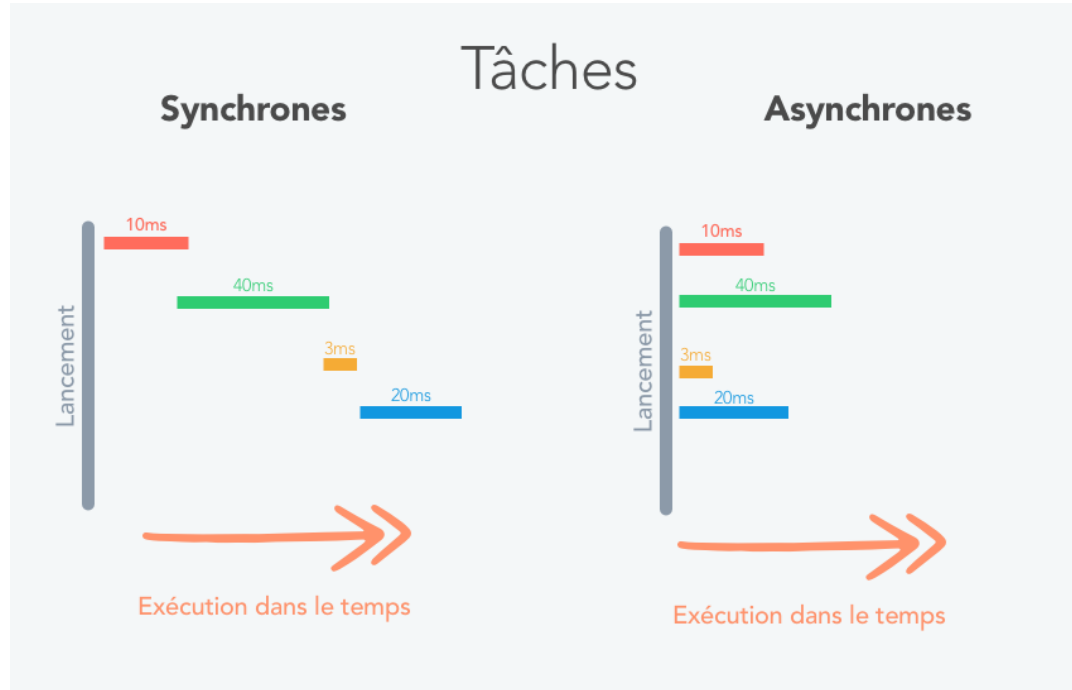


Objectif



L'**objectif** de la **programmation asynchrone** va être que votre **programme** puisse **faire plusieurs choses en même temps**, et ainsi un **gain de temps** important.

Le fonctionnement de la programmation asynchrone



Contexte d'utilisation de la programmation asynchrone

- **Scénario** : Imaginez une **application** qui doit **traiter plusieurs requêtes simultanées**.
- **Problème** : L'approche **synchrone** peut entraîner des **temps d'attente inutiles**.
- **Solution** : La programmation **asynchrone** permet de gérer ces **tâches simultanément, optimisant** ainsi l'**utilisation** des **ressources**.

Utilisation des mots clé **async** et **await**

Pour réaliser un **programme asynchrone** en **python**, il est nécessaire d'utiliser les **mots clés** **async** et **await**, ainsi que des **modules complémentaires** (tel que **asyncio**).

Exemple :

```
async def maFonctionAsynchrone():  
    # Code asynchrone ici  
    result = await autreFonctionAsynchrone()  
    return result
```

Programme synchrone

Exemple :

```
import requests
```

```
def interrogerSite(url):  
    response = requests.get(url)  
    contenu = response.text  
    print(f"Site interrogé ({url}), taille de la réponse : {len(contenu)}")
```

```
def main():  
    urls = [  
        'https://www.example.com',  
        'https://www.example.org',  
        'https://www.example.net']
```

```
    for url in urls:  
        interrogerSite(url)
```

```
main()
```

Programme asynchrone

Exemple :

```
import aiohttp
import asyncio

async def interrogerSite(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            contenu = await response.text()
            print(f"Site interrogé ({url}), taille de la réponse : {len(contenu)}")

async def main():
    urls = [
        'https://www.example.com',
        'https://www.example.org',
        'https://www.example.net'
    ]

    tasks = [interrogerSite(url) for url in urls]
    await asyncio.gather(*tasks)

await main()
```



**Avez-vous des
questions ?**



Exercices

Programmation Async



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines are thin and grey, connecting the nodes in a non-linear fashion.

9. **Tkinter**

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It consists of a cluster of nodes (solid grey circles and hollow circles with grey outlines) connected by thin grey lines, forming a web-like structure.

Définition



Tkinter est un **module** de **base intégré** dans **Python** qui permet de **réaliser** des **interfaces graphiques**.



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure.

9.1

Les widgets

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid dark grey and others being hollow light grey circles. The overall pattern is a complex, interconnected web.

Notre premier programme

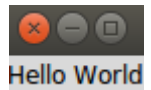
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
label = Label(fenetre, text="Hello World")  
label.pack()
```

```
fenetre.mainloop()
```



Les boutons

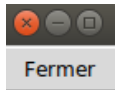
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
bouton = Button(fenetre, text="Fermer", command=fenetre.quit)  
bouton.pack()
```

```
fenetre.mainloop()
```



Les labels

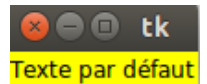
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
label = Label(fenetre, text="Texte par défaut", bg="yellow")  
label.pack()
```

```
fenetre.mainloop()
```



Entrée / Input

Exemple :

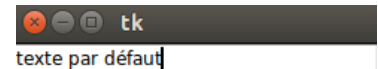
```
from tkinter import *
```

```
fenetre = Tk()
```

```
value = StringVar()  
value.set("texte par défaut")
```

```
entree = Entry(fenetre, textvariable=value, width=30)  
entree.pack()
```

```
fenetre.mainloop()
```



Case à cocher

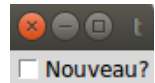
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
bouton = Checkbutton(fenetre, text="Nouveau?")  
bouton.pack()
```

```
fenetre.mainloop()
```



Boutons radio

Exemple :

```
from tkinter import *

fenetre = Tk()

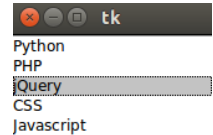
value = StringVar()
bouton1 = Radiobutton(fenetre, text="Oui", variable=value, value=1)
bouton2 = Radiobutton(fenetre, text="Non", variable=value, value=2)
bouton3 = Radiobutton(fenetre, text="Peu être", variable=value, value=3)
bouton1.pack()
bouton2.pack()
bouton3.pack()

fenetre.mainloop()
```

Liste

Exemple :

```
from tkinter import *  
  
fenetre = Tk()  
  
liste = Listbox(fenetre)  
liste.insert(1, "Python")  
liste.insert(2, "PHP")  
liste.insert(3, "jQuery")  
liste.insert(4, "CSS")  
liste.insert(5, "Javascript")  
  
liste.pack()  
  
fenetre.mainloop()
```



Canvas

Exemple :

```
from tkinter import *

fenetre = Tk()

canvas = Canvas(fenetre, width=150, height=120, background='yellow')
ligne1 = canvas.create_line(75, 0, 75, 120)
ligne2 = canvas.create_line(0, 60, 150, 60)
txt = canvas.create_text(75, 60, text="Cible", font="Arial 16 italic", fill="blue")
canvas.pack()

fenetre.mainloop()
```



Scale

Exemple :

```
from tkinter import *
```

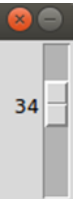
```
fenetre = Tk()
```

```
value = DoubleVar()
```

```
scale = Scale(fenetre, variable=value)
```

```
scale.pack()
```

```
fenetre.mainloop()
```



Frames

Exemple :

```
from tkinter import *
fenetre = Tk()

fenetre['bg']='white'

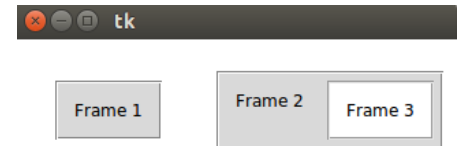
# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.pack(side=LEFT, padx=30, pady=30)

# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.pack(side=LEFT, padx=10, pady=10)

# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.pack(side=RIGHT, padx=5, pady=5)

# Ajout de labels
Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
Label(Frame3, text="Frame 3", bg="white").pack(padx=10, pady=10)

fenetre.mainloop()
```



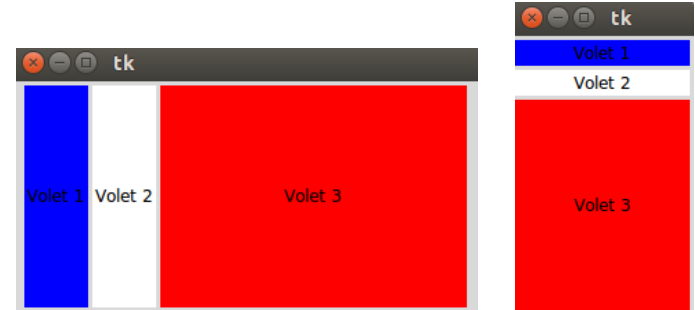
PanedWindow

Exemple :

```
from tkinter import *  
fenetre = Tk()
```

```
p = PanedWindow(fenetre, orient=HORIZONTAL)  
p.pack(side=TOP, expand=Y, fill=BOTH, pady=2, padx=2)  
p.add(Label(p, text='Volet 1', background='blue', anchor=CENTER))  
p.add(Label(p, text='Volet 2', background='white', anchor=CENTER))  
p.add(Label(p, text='Volet 3', background='red', anchor=CENTER))  
p.pack()
```

```
fenetre.mainloop()
```



Spinbox

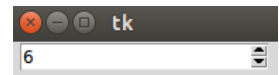
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
s = Spinbox(fenetre, from_=0, to=10)  
s.pack()
```

```
fenetre.mainloop()
```



LabelFrame

Exemple :

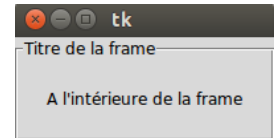
```
from tkinter import *
```

```
fenetre = Tk()
```

```
l = LabelFrame(fenetre, text="Titre de la frame", padx=20, pady=20)  
l.pack(fill="both", expand="yes")
```

```
Label(l, text="A l'intérieure de la frame").pack()
```

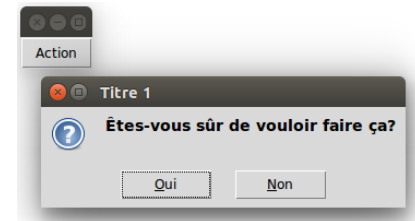
```
fenetre.mainloop()
```



Alertes

Exemple :

```
from tkinter import *  
from tkinter.messagebox import *  
  
fenetre = Tk()  
  
def callback():  
    if askyesno('Titre 1', 'Êtes-vous sûr de vouloir faire ça?):  
        showwarning('Titre 2', 'Tant pis...')  
    else:  
        showinfo('Titre 3', 'Vous avez peur!')  
        showerror("Titre 4", "Aha")  
  
Button(text='Action', command=callback).pack()  
  
fenetre.mainloop()
```



Barre de menu

Exemple :

```
from tkinter import *
from tkinter.messagebox import *

fenetre = Tk()

def alert():
    showinfo("alerte", "Bravo!")

menubar = Menu(fenetre)

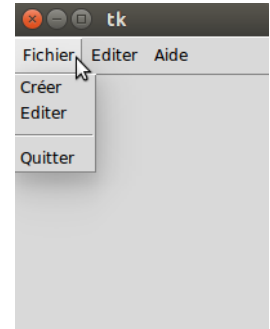
menu1 = Menu(menubar, tearoff=0)
menu1.add_command(label="Créer", command=alert)
menu1.add_command(label="Editer", command=alert)
menu1.add_separator()
menu1.add_command(label="Quitter", command=fenetre.quit)
menubar.add_cascade(label="Fichier", menu=menu1)

menu2 = Menu(menubar, tearoff=0)
menu2.add_command(label="Couper", command=alert)
menu2.add_command(label="Copier", command=alert)
menu2.add_command(label="Coller", command=alert)
menubar.add_cascade(label="Editer", menu=menu2)

menu3 = Menu(menubar, tearoff=0)
menu3.add_command(label="A propos", command=alert)
menubar.add_cascade(label="Aide", menu=menu3)

fenetre.config(menu=menubar)

fenetre.mainloop()
```





9.2

Les options



Help

```
print(dir(Button()))
```

Permet de **connaître** toutes les **méthodes/options** d'un **widget**.

Side

Exemple :

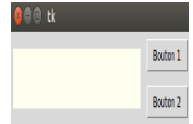
```
from tkinter import *
```

```
fenetre = Tk()
```

```
Canvas(fenetre, width=250, height=100, bg='ivory').pack(side=TOP, padx=5, pady=5)  
Button(fenetre, text = 'Bouton 1').pack(side=LEFT, padx=5, pady=5)  
Button(fenetre, text = 'Bouton 2').pack(side=RIGHT, padx=5, pady=5)
```



```
Canvas(fenetre, width=250, height=100, bg='ivory').pack(side=TOP, padx=5, pady=5)  
Button(fenetre, text = 'Bouton 1').pack(side=TOP, padx=5, pady=5)  
Button(fenetre, text = 'Bouton 2').pack(side=BOTTOM, padx=5, pady=5)
```



```
fenetre.mainloop()
```

Unités de dimensions

i : pouces
m : millimètre
c : centimètre

Options de dimensions

height	: Hauteur du widget.
width	: Largeur du widget.
padx, pady	: Espace supplémentaire autour du widget. X pour horizontal et V pour vertical.
borderwidth	: Taille de la bordure.
highlightthickness	: Largeur du rectangle lorsque le widget a le focus.
selectborderwidth	: Largeur de la bordure tridimensionnel autour du widget sélectionné.
wrlength	: Nombre de ligne maximum pour les widget en mode "word wrapping".

Options de couleurs

background (ou bg)	: couleur de fond du widget.
foreground (ou fg)	: couleur de premier plan du widget.
activebackground	: couleur de fond du widget lorsque celui-ci est actif.
activeForeground	: couleur de premier plan du widget lorsque le widget est actif.
disabledForeground	: couleur de premier plan du widget lorsque le widget est désactivé.
highlightbackground	: Couleur de fond de la région de surbrillance lorsque le widget a le focus.
highlightcolor	: couleur de premier plan de la région en surbrillance lorsque le widget a le focus.
selectbackground	: Couleur de fond pour les éléments sélectionnés.
selectforeground	: couleur de premier plan pour les éléments sélectionnés.

Relief

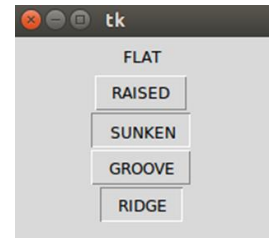
Exemple :

```
from tkinter import *
```

```
fenetre = Tk()
```

```
b1 = Button(fenetre, text = "FLAT", relief=FLAT).pack()  
b2 = Button(fenetre, text = "RAISED", relief=RAISED).pack()  
b3 = Button(fenetre, text = "SUNKEN", relief=SUNKEN).pack()  
b4 = Button(fenetre, text = "GROOVE", relief=GROOVE).pack()  
b5 = Button(fenetre, text = "RIDGE", relief=RIDGE).pack()
```

```
fenetre.mainloop()
```



Grille

Exemple :

```
fenetre = Tk()
```

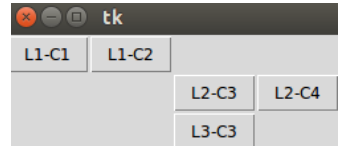
```
for ligne in range(5):  
    for colonne in range(5):  
        Button(fenetre, text='L%s-C%s' % (ligne, colonne),  
borderwidth=1).grid(row=ligne, column=colonne)
```

```
Button(fenetre, text='L1-C1', borderwidth=1).grid(row=1, column=1)  
Button(fenetre, text='L1-C2', borderwidth=1).grid(row=1, column=2)  
Button(fenetre, text='L2-C3', borderwidth=1).grid(row=2, column=3)  
Button(fenetre, text='L2-C4', borderwidth=1).grid(row=2, column=4)  
Button(fenetre, text='L3-C3', borderwidth=1).grid(row=3, column=3)
```

```
fenetre.mainloop()
```



L0-C0	L0-C1	L0-C2	L0-C3	L0-C4
L1-C0	L1-C1	L1-C2	L1-C3	L1-C4
L2-C0	L2-C1	L2-C2	L2-C3	L2-C4
L3-C0	L3-C1	L3-C2	L3-C3	L3-C4
L4-C0	L4-C1	L4-C2	L4-C3	L4-C4



L1-C1	L1-C2		
		L2-C3	L2-C4
		L3-C3	

Image

Exemple :

```
fenetre = Tk()
```

```
photo = PhotoImage(file="ma_photo.png")
```

```
canvas = Canvas(fenetre, width=350, height=200)  
canvas.create_image(0, 0, anchor=NW, image=photo)  
canvas.pack()
```

```
fenetre.mainloop()
```

Récupérer un fichier

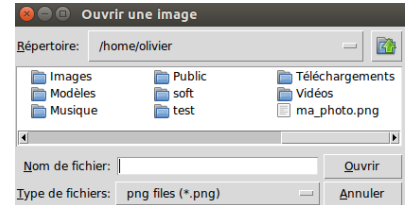
Exemple :

```
from tkinter import *
from tkinter.filedialog import *

fenetre = Tk()

filepath = askopenfilename(title="Ouvrir une image", filetypes=[('png files', '.png'),
('all files', '*.*)])
photo = PhotoImage(file=filepath)
canvas = Canvas(fenetre, width=photo.width(), height=photo.height(), bg="yellow")
canvas.create_image(0, 0, anchor=NW, image=photo)
canvas.pack()

fenetre.mainloop()
```



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure.

9.3

Les évènements

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being solid dark grey and others being hollow light grey circles. The lines are thin and light grey, forming a web-like pattern.

Evènements

<Button-1>	: Click gauche
<Button-2>	: Click milieu
<Button-3>	: Click droit
<Double-Button-1>	: Double click droit
<Double-Button-2>	: Double click gauche
<KeyPress>	: Pression sur une touche
<KeyPress-a>	: Pression sur la touche A (minuscule)
<KeyPress-A>	: Pression sur la touche A (majuscule)
<Return>	: Pression sur la touche entrée
<Escape>	: Touche Echap
<Up>	: Pression sur la flèche directionnelle haut
<Down>	: Pression sur la flèche directionnelle bas
<ButtonRelease>	: Lorsque qu'on relache le click
<Motion>	: Mouvement de la souris
<B1-Motion>	: Mouvement de la souris avec click gauche
<Enter>	: Entrée du curseur dans un widget
<Leave>	: Sortie du curseur dans un widget
<Configure>	: Redimensionnement de la fenêtre
<Map> <Unmap>	: Ouverture et iconification de la fenêtre
<MouseWheel>	: Utilisation de la roulette

Evènements

Exemple:

```
from tkinter import *

fenetre = Tk()

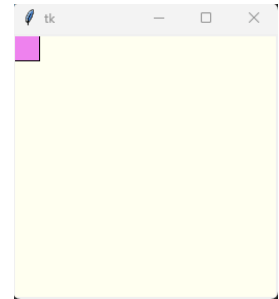
# fonction appelée lorsque l'utilisateur presse une touche
def clavier(event):
    global coords

    touche = event.keysym

    if touche == "Up":
        coords = (coords[0], coords[1] - 10)
    elif touche == "Down":
        coords = (coords[0], coords[1] + 10)
    elif touche == "Right":
        coords = (coords[0] + 10, coords[1])
    elif touche == "Left":
        coords = (coords[0] - 10, coords[1])
    # changement de coordonnées pour le rectangle
    canvas.coords(rectangle, coords[0], coords[1], coords[0]+25, coords[1]+25)

# création du canvas
canvas = Canvas(fenetre, width=250, height=250, bg="ivory")
# coordonnées initiales
coords = (0, 0)
# création du rectangle
rectangle = canvas.create_rectangle(0,0,25,25,fill="violet")
# ajout du bond sur les touches du clavier
canvas.focus_set()
canvas.bind("<Key>", clavier)
# création du canvas
canvas.pack()

fenetre.mainloop()
```





Avez-vous des questions ?



Exercices

Tkinter

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels of connectivity or importance. The lines are thin and gray, creating a mesh-like structure.

10.

Les API

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes being larger and more prominent than others. The overall style is minimalist and technical.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with solid centers and others with dashed outlines. The lines are thin and gray, creating a dense, organic structure.

10.1

Les généralités

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a cluster of nodes connected by lines, with some nodes having solid centers and others having dashed outlines. The overall structure is a complex, interconnected web.

Définition



API (Interface de Programmation d'Application) est une **application web** qui, pour **chaque demande**, **renvoie** des **données** ou **écrit** des **données** dans une **base de données**.

Objectif



Les **API** permettent : L'accès à des fonctionnalités **externes**, l'échange de **données** entre **applications** et l'intégration de **services tiers**.



Les architectures d'API Populaire

REST
**(representational
state transfer)**

SOAP
(Simple object
access protocol)

GraphQL

gRPC
(API Google)

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with concentric rings, and the lines are thin and grey. The diagram is partially cut off by the top and left edges of the slide.

10.1

Les API REST



REST - Les méthodes

Méthodes	Description
POST	Envoie des données.
GET	Récupère des données.
PUT	Modifie des données existantes.
DELETE	Supprime des données existantes.

REST – Les six contraintes de conception d'une API RESTful

Architecture client-serveur

une architecture REST est composée de clients, de serveurs et de ressources et elle traite les requêtes via le protocole HTTP.

Serveur stateless

le contenu du client n'est jamais stocké sur le serveur entre les requêtes. Les informations sur l'état de la session sont, quant à elles, stockées sur le client.

Mémoire cache

la mise en mémoire cache permet de se passer de certaines interactions entre le client et le serveur.

Interface uniforme

le serveur fournira une interface uniforme pour l'accès aux ressources sans définir leur représentation.

Système à couches

des couches supplémentaires peuvent assurer la médiation dans les interactions entre le client et le serveur. Ces couches peuvent remplir des fonctions supplémentaires, telles que l'équilibrage de charge, le partage des caches ou la sécurité.

Code à la demande (facultatif)

un serveur peut étendre les fonctionnalités d'un client en lui transférant du code exécutable.

REST – Les codes de retour

Code	Signification	Description
200	OK	L'action demandée a été réalisée avec succès.
201	Créé	Une nouvelle ressource a été créée.
202	Accepté	La demande a été reçue, mais aucune modification n'a encore été effectuée.
204	Pas de contenu	La demande a abouti, mais la réponse n'a pas de contenu.
400	Mauvais requête	La demande était malformée.
401	Non autorisé	Le client n'est pas autorisé à effectuer l'action demandée.
404	Non trouvé	La ressource demandée n'a pas été trouvée.
415	Non pris en charge du type de média	Le format des données demandées n'est pas pris en charge par le serveur.
422	Entité non traitable	Les données de la demande étaient correctement formatées mais contenaient des données invalides ou manquantes.
500	Erreur interne du serveur	Le serveur a provoqué une erreur lors du traitement de la demande.

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid dark grey, while others are hollow with a light grey outline. The lines connecting them are thin and light grey, creating a dense, organic structure that tapers off towards the right.

10.3

L'API requests

A decorative network diagram in the bottom-right corner, mirroring the style of the top-left one. It consists of a cluster of interconnected nodes and lines. The nodes are small circles, some solid dark grey and some hollow with light grey outlines. The lines are thin and light grey, forming a complex, web-like pattern that tapers off towards the left.



API requests - installation

La **bibliothèque requests** facilite l'envoi de requêtes **HTTP**.

Pour l'installer il faut exécuter la commande suivante :

```
pip install requests
```

API requests - requête GET

Exemple :

```
import requests

api_url = "https://jsonplaceholder.typicode.com/todos/1"
response = requests.get(api_url)
print(response.json())
print(response.status_code)
print(response.headers["Content-Type"])
```

```
{"userId": 1, "id": 1, "title": "delectus aut autem", "completed": false}
200
application/json; charset=utf-8
```

API requests - requête POST

Exemple :

```
import requests
import json
```

```
api_url = "https://jsonplaceholder.typicode.com/todos"
```

```
todo = {"userId": 1, "title": "Buy milk", "completed": False}
headers = {"Content-Type": "application/json"}
```

```
response = requests.post(api_url, json=todo)
print(response.json())
response = requests.post(api_url, data=json.dumps(todo), headers=headers)
print(response.json())
print(response.status_code)
```

```
{'userId': 1, 'title': 'Buy milk', 'completed': False, 'id': 201}
{'userId': 1, 'title': 'Buy milk', 'completed': False, 'id': 201}
201
```

API requests - requête PUT

Exemple :

```
import requests
```

```
api_url = "https://jsonplaceholder.typicode.com/todos/10"  
response = requests.get(api_url)  
print(response.json())
```

```
todo = {"userId": 1, "title": "Wash car", "completed": True}  
response = requests.put(api_url, json=todo)  
print(response.json())  
print(response.status_code)
```

```
{'userId': 1, 'id': 10, 'title': 'illo est ratione doloreque quia maiores aut', 'completed': True}  
{'userId': 1, 'title': 'Wash car', 'completed': True, 'id': 10}  
200
```


API requests - requête DELETE

Exemple :

```
import requests
```

```
api_url = "https://jsonplaceholder.typicode.com/todos/10"
```

```
response = requests.get(api_url)
```

```
print(response.json())
```

```
response = requests.delete(api_url)
```

```
print(response.json())
```

```
print(response.status_code)
```

```
{'userId': 1, 'id': 10, 'title': 'illo est ratione doloremque quia maiores aut', 'completed': True}
```

```
200
```



Avez-vous des questions ?



Exercices

API