



# RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL

## M103 - Programmer en Orienté Objet



100 heures



# SOMMAIRE

## 01 - APPRÉHENDER LE PARADIGME DE LA POO

- Introduire la POO
- Définir une classe
- Créer un objet
- Connaître l'encapsulation
- Manipuler les méthodes

## 02 - CONNAÎTRE LES PRINCIPAUX PILIERS DE LA POO

- Définir l'héritage
- Définir le polymorphisme
- Caractériser l'abstraction
- Manipuler les interfaces

## 03 - CODER DES SOLUTIONS ORIENTÉES OBJET

- Coder une solution orientée objet
- Manipuler les données
- Utiliser les expressions régulières
- Administrer les exceptions

## 04 - MANIPULER LES MODULES ET LES BIBLIOTHÈQUES

- Manipuler les modules
- Manipuler les bibliothèques

# MODALITÉS PÉDAGOGIQUES



1

## LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques.



2

## LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life.



3

## DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life



4

## LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life.



5

## DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage



## PARTIE 1

### Appréhender le paradigme de la POO

Dans ce module, vous allez :

- Comprendre le principe de la POO
- Maîtriser la définition d'une classe
- Maîtriser le concept d'objet
- Connaître le principe de l'encapsulation
- Savoir manipuler les méthodes



05 heures

# CHAPITRE 1

## Introduire la POO

### Ce que vous allez apprendre dans ce chapitre :

- Acquérir une compréhension de la méthode de programmation POO
- Connaitre le principe de la POO
- Citer ses avantages par rapports aux autres paradigmes de programmation



01 heure



# CHAPITRE 1

## Introduire la POO

1. **Introduction à la programmation Orientée Objet**
2. Brève historique de l'évolution des langages de programmation Orientée Objet
3. Connaissance des avantages de la POO par rapport aux autres paradigmes



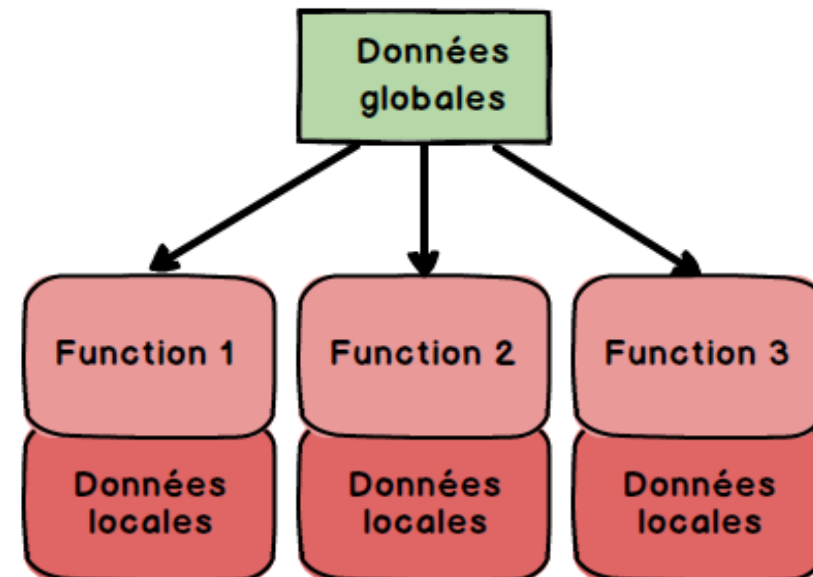
# 01 - Introduire la POO

## Introduction à la programmation Orientée Objet

### Programmation procédurale

- Dans la **programmation procédurale**, le programme est divisé en petites parties appelées **procédures** ou **fonctions**.
- Ensuite, pour résoudre chaque partie, une ou plusieurs procédures/fonctions sont utilisées
- Dans la **programmation procédurale**, les notions de données et de traitement de ces données sont **séparées**

Programmes = Procédures/Fonctions + Données



# 01 - Introduire la POO

## Introduction à la programmation Orientée Objet

### Programmation procédurale

Dans la programmation procédurale :

- Les données constituent **la partie passive du programme**
- Les procédures et les fonctions **constituent la partie active**

Programmer dans ce cas revenait à :

- définir un certain nombre de variables (entier, chaîne de caractères, structures, tableaux, etc.)
- écrire des procédures pour les manipuler

Inconvénients :

- Difficulté de réutilisation du code
- Difficulté de la maintenance de grandes applications

```
CalculSomme100PremierNombre  
Var
```

```
    I,S: entier
```

```
Somme
```

```
Début /*début de la procédure*/
```

```
    S:=0
```

```
    Pour i:=1 à 100 Faire
```

```
        S:=S+1
```

```
    FinPour
```

```
    Ecrire(" La somme des 100 premiers  
    nombres est ",S);
```

```
    Fin /*Fin de la procédure*/
```

```
    Début /*début algorithme*/
```

```
    Somme
```

```
    Fin /*fin algorithme*/
```





# 01 - Introduire la POO

## Introduction à la programmation Orientée Objet



### Programmation orientée objet

- Séparation (données, procédures) est elle utile?
- Pourquoi privilégier les procédures sur les données ?
- Pourquoi ne pas considérer que les programmes sont avant tout des ensembles objets informatiques caractérisés par les opérations qu'ils connaissent ?

- Les **langages orientés objets** sont nés pour répondre à ces questions.  
Ils sont fondés sur la connaissance d'une seule catégorie d'entités informatiques : **les objets**
- Un **objet** incorpore des aspects **statiques** et **dynamiques** au sein d'une même notion
- Un programme est constitué d'un ensemble d'objets chacun disposant d'une partie procédures (traitement) et d'une partie données.

# 01 - Introduire la POO

## Introduction à la programmation Orientée Objet



Programmation procédurale

Que doit faire le système ?



Programmation Orientée Objet

Sur quoi doit-il le faire?

De quoi doit être composé  
mon programme ?

# 01 - Introduire la POO

## Introduction à la programmation Orientée Objet



Programmation procédurale



Programmation Orientée Objet

Créditer un compte ?  
Débiter un compte ?

Programme de Gestion  
des comptes bancaires

Comment se présente (structure)  
un compte bancaire ?

# CHAPITRE 1

## Introduire la POO

1. Introduction à la programmation Orientée Objet
2. **Brève historique de l'évolution des langages de programmation Orientée Objet**
3. Connaissance des avantages de la POO par rapport aux autres paradigmes



# 01 - Introduire la POO

## Brève historique de l'évolution des POO

Les années 70

Les années 80

Les années 90

De nos jours

- Les concepts de la POO naissent au cours des années 1970 dans des laboratoires de recherche en informatique.
- Les premiers langages de programmation véritablement orientés objet ont été **Simula**, puis **Smalltalk**.
  - **Simula** (1966) regroupe données et procédures.
  - **Simula I** (1972) formalise les concepts d'objet et de classe. Un programme est constitué d'une collection d'objets actifs et autonomes.
  - **Smalltalk** (1972) : généralisation de la notion d'objet.

# 01 - Introduire la POO

## Brève historique de l'évolution des POO

Les années 70

Les années 80

Les années 90

De nos jours

- A partir des années 80, les principes de la POO sont appliqués dans de nombreux langages
  - Eiffel créé par le Français Bertrand Meyer
  - C++ est une extension objet du langage C créé par le Danois Bjarne Stroustrup
  - Objective C une autre extension objet du C utilisé, entre autres, par l'iOS d'Apple

# 01 - Introduire la POO

## Brève historique de l'évolution des POO

Les années 70

Les années 80

Les années 90

De nos jours

- Les années 1990 ont vu l'avènement des POOs dans de nombreux secteurs du développement logiciel, et la création du langage **Java** par la société Sun Microsystems

# 01 - Introduire la POO

## Brève historique de l'évolution des POO

Les années 70

Les années 80

Les années 90

De nos jours

- De nos jours, de très nombreux langages permettent d'utiliser les principes de la POO dans des domaines variés tels que :
  - **PHP (à partir de la version 5),**
  - **VB.NET,**
  - **PowerShell,**
  - **Python, etc.**



# CHAPITRE 1

## Introduire la POO

1. Introduction à la programmation Orientée Objet
2. Brève historique de l'évolution des langages de programmation Orientée Objet
3. **Connaissance des avantages de la POO par rapport aux autres paradigmes**



# 01 - Introduire la POO

## Connaissance des avantages de la POO par rapport aux autres paradigmes



### Motivation et avantages de la POO

**Motivation** : concevoir, maintenir et exploiter facilement de gros logiciels

**Avantages** :

- **Modularité** : les objets forment des modules compacts regroupant des données et un ensemble d'opérations ce qui **réduit la complexité de l'application** (classe = module)
- **Abstraction** :
  - Les entités objets de la POO sont proches de celles du monde réel (objet est un compte, stagiaire, formateur, etc.).
  - La POO se base sur un processus qui consiste à masquer des détails non pertinents à l'utilisateur.

**Exemple** : Pour envoyer un SMS, vous tapez simplement le message, sélectionnez le contact et cliquez sur Envoyer

→ Ce qui se passe réellement en arrière-plan est masqué car il n'est pas pertinent à vous.

- **Réutilisabilité** :

- La POO favorise la réutilisation de composants logiciels et même d'architectures complexes
- La définition d'une relation d'héritage entre les entités logicielles évite la duplication de code
- → Facilité de la **maintenance logicielle** et amélioration de la **productivité**

## CHAPITRE 2

### Définir une classe

Ce que vous allez apprendre dans ce chapitre :

- Comprendre le concept de classe
- Savoir modéliser une classe
- Savoir définir les composantes d'une classe



01 heure



# CHAPITRE 2

## Définir une classe

1. Définition d'une classe
2. Modélisation d'une classe
3. Composantes d'une classe

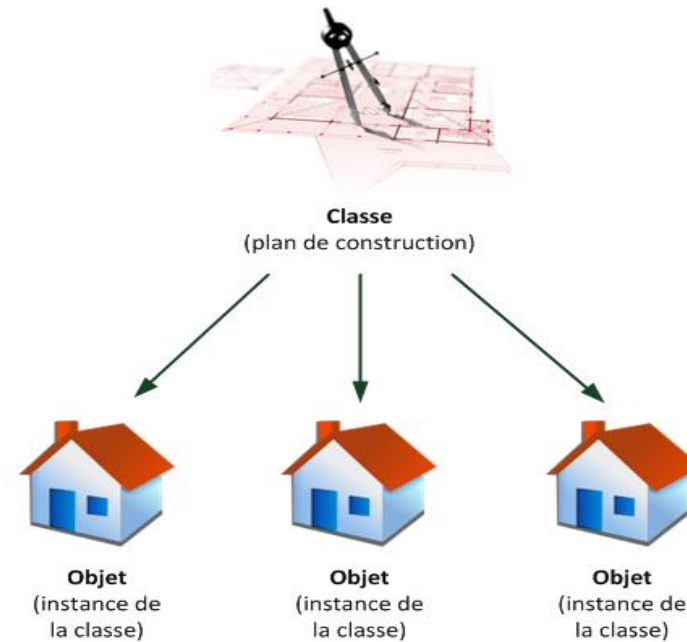


## 02 - Définir une classe

### Définition d'une classe

#### Notion de classe

- Une classe correspond à la généralisation de la notion de type que l'on rencontre dans les langages classiques
- Une classe est la **description** d'un ensemble d'objets ayant une **structure de données commune** et un même **comportement**
- Les classes sont des **moules**, des **patrons qui permettent de créer des objets en série sur le même modèle**



# CHAPITRE 2

## Définir une classe

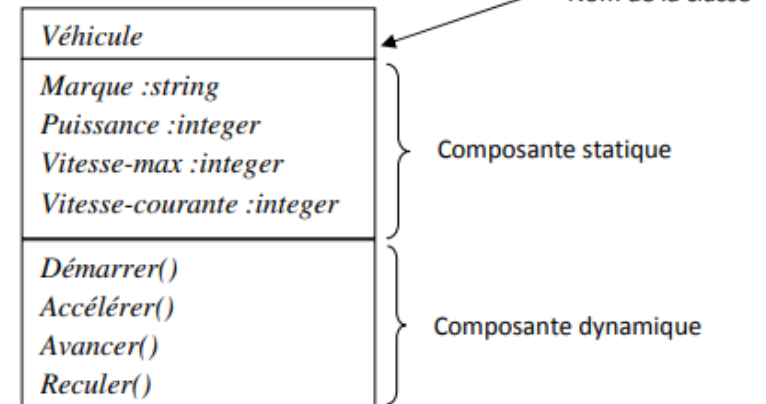
1. Définition d'une classe
- 2. Modélisation d'une classe**
3. Composantes d'une classe



### Modélisation d'une classe

- Une classe est caractérisée par :
  - **Un nom**
  - **Une composante statique** : des champs (ou **attributs**). Ils caractérisent l'état des objets pendant l'exécution du programme
  - **Une composante dynamique** : des **méthodes** représentant le comportement des objets de cette classe. Elles manipulent les champs des objets et caractérisent les actions pouvant être effectuées par les objets

**Exemple :** la classe *Véhicule*



## CHAPITRE 2

### Définir une classe

1. Définition d'une classe
2. Modélisation d'une classe
- 3. Composantes d'une classe**



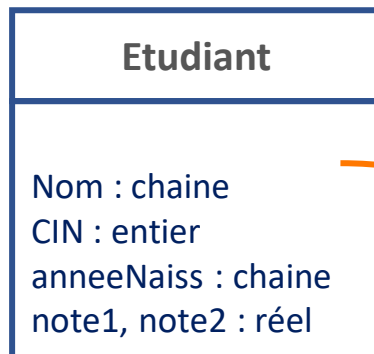
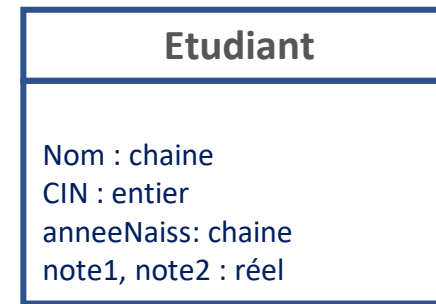


## 02 - Définir une classe

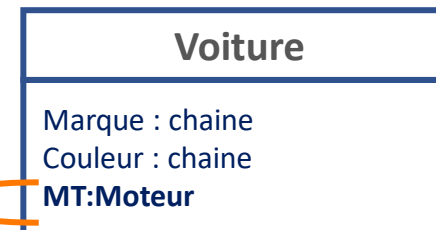
### Composantes d'une classe

#### Attribut

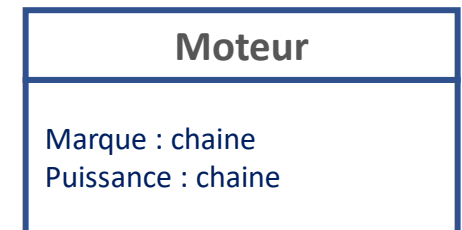
- Un **attribut** appelé également **champ** ou **donnée membre** correspond à une propriété de la classe
- Un **attribut est défini par**:
  - un nom,
  - un type de données
  - une valeur initiale (éventuellement)
- Un **attribut peut être de type** :
  - simple: entier, réel, chaîne de caractères, caractère, etc
  - Objet de type classe: Etudiant, Voiture, etc



Attributs de  
type simple



Attribut de  
type objet



## 02 - Définir une classe

### Composantes d'une classe

#### Accès aux attributs

- Pour accéder à un attribut d'un objet on indique le nom de la référence de l'objet suivi par le nom de l'attribut dans l'objet de la manière suivante :

**nomObjet. nomAttribut**

- nomObjet** : nom de de la référence à l'objet
- nomAttribut** = nom de l'attribut

#### Visibilité des attributs

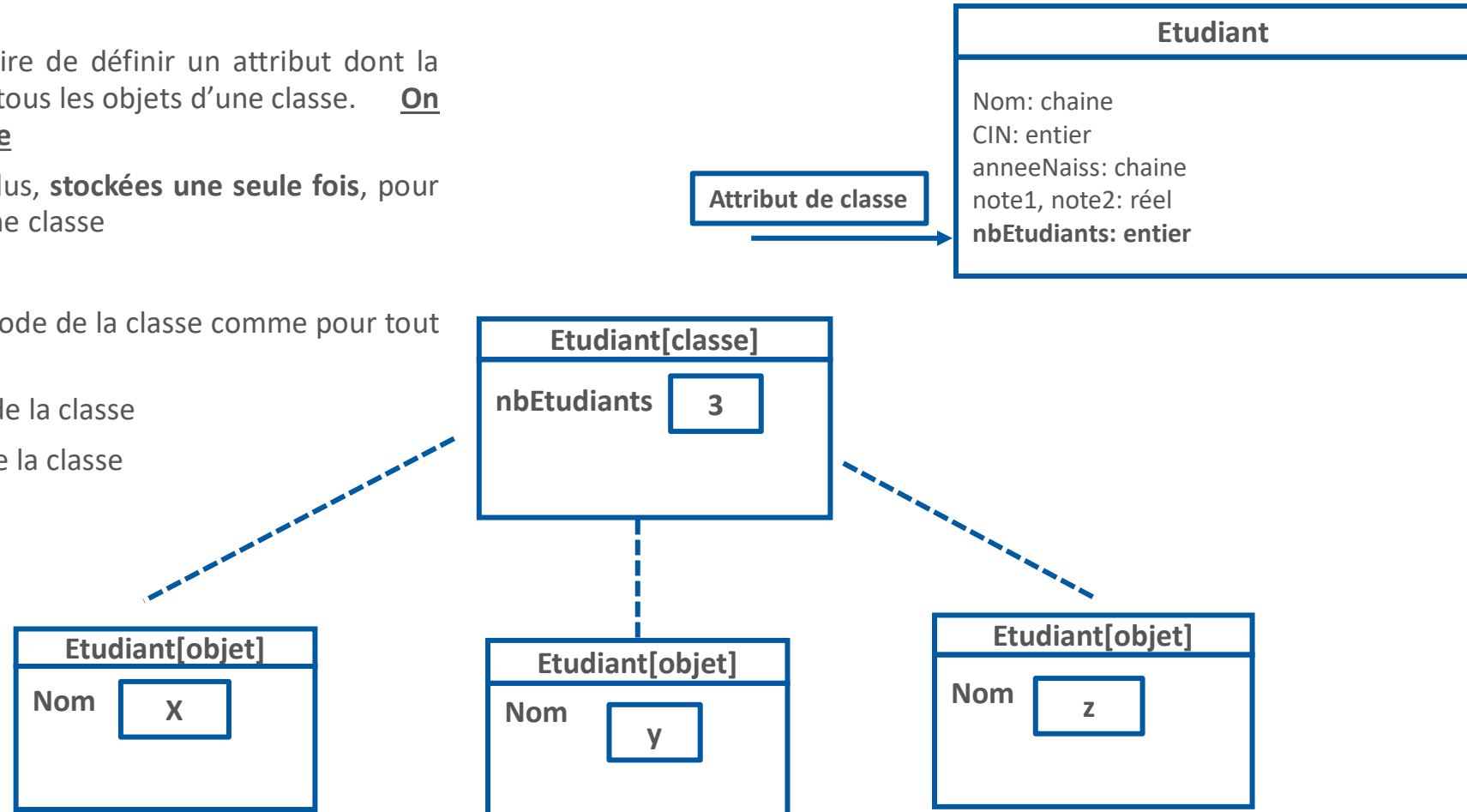
- La visibilité des attributs définit les droits d'accès aux données d'une classe :
- Publique (+) :**
  - Toute classe peut accéder aux données d'une classe définie avec le niveau de visibilité publique.
- Protégée (#) :**
  - L'accès aux données est réservé aux méthodes des classes héritières
  - ( A voir ultérieurement dans la partie Héritage)
- Privée (-) :**
  - L'accès aux données est limité aux méthodes de la classe elle-même

Nom_de_la_classe
# Attribut1 : Type - Attribut2 : Type ...
+ méthode1 () Méthode2 () ...

## 02 - Définir une classe

### Composantes d'une classe

- Il peut s'avérer nécessaire de définir un attribut dont la valeur est **partagée** par tous les objets d'une classe. On parle d'attribut de classe
- Ces données sont, de plus, **stockées une seule fois**, pour toutes les instances d'une classe
- Accès :**
  - Depuis, une méthode de la classe comme pour tout autre attribut
  - Via une instance de la classe
  - À l'aide du nom de la classe



## 02 - Définir une classe

### Composantes d'une classe



#### Constructeur

- Un **constructeur** est une méthode particulière invoquée implicitement lors de la création d'un objet
- Un constructeur **permet d'initialiser** les données des objets (les attributs) de la classe dont elle dépend
- Le constructeur ne doit pas avoir un type de retour
- Une classe peut posséder plusieurs constructeurs, mais un objet donné n'aura pu être produit que par un seul constructeur

#### Types de Constructeurs

- **Constructeur par défaut**
  - Un constructeur sans aucun paramètre est appelé un constructeur par défaut
  - Si nous ne créons pas de constructeur, la classe appellera automatiquement le constructeur par défaut lorsqu'un objet est créé
- **Constructeur paramétré:**
  - Un constructeur avec au moins un paramètre s'appelle un constructeur paramétré
- **Constructeur de copie:**
  - Le constructeur qui crée un objet en copiant les données d'un autre objet s'appelle un constructeur de copie

## 02 - Définir une classe

### Composantes d'une classe



#### Destructeur

- Le destructeur est une méthode particulière qui permet **la destruction d'un objet non référencé**.
- Le destructeur ne doit pas avoir un type de retour

#### Un destructeurs permet de :

- **Gérer les erreurs**
- **Libérer** les ressources utilisées de manière certaine
- **Assurer la fermeture** de certaines parties du code.

- Les langages qui utilisent des ramasse-miettes (exemple Python) n'offrent pas le mécanisme des destructeurs puisque le programmeur ne gère pas la mémoire lui-même
  - Un ramasse-miettes est un programme de gestion automatique de la mémoire. Il est responsable du recyclage de la mémoire préalablement allouée puis inutilisée.

## CHAPITRE 3

### Créer un objet

Ce que vous allez apprendre dans ce chapitre :

- Comprendre le concept d'objet
- Maîtriser les concepts d'instanciation d'une classe et la destruction d'un objet



01 heure

# CHAPITRE 3

## Créer un objet

1. Définition d'un objet
2. Instanciation
3. Destruction explicite d'un objet



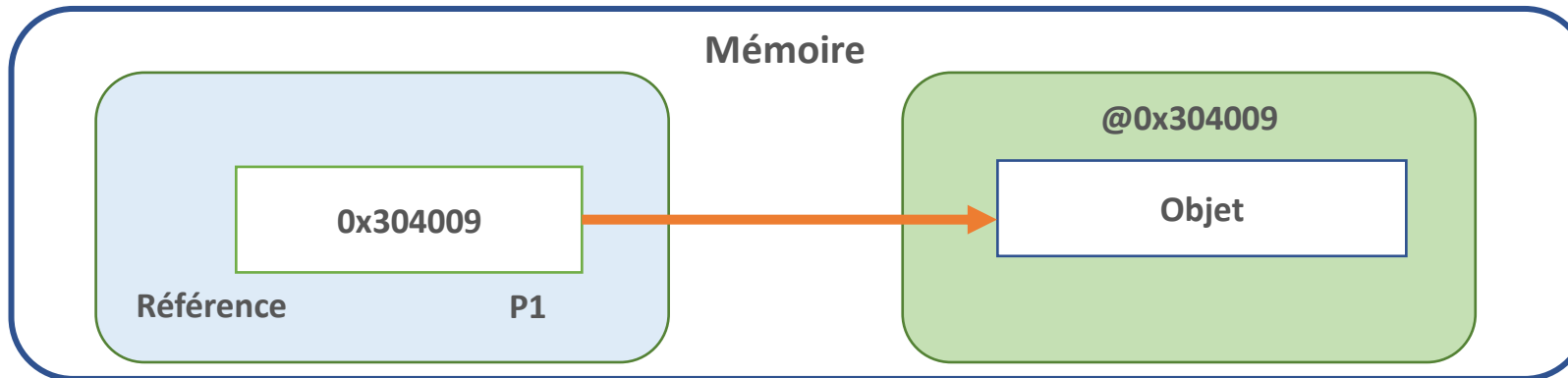
## 03 - Créer un objet

### Définition d'un objet

#### Notion d'objet

**OBJET = Référent + Etat + Comportement**

- Chaque objet doit avoir un nom (référence) « qui lui est propre » pour l'identifier
- La référence permet d'accéder à l'objet, mais n'est pas l'objet lui-même. Elle contient l'adresse de l'emplacement mémoire dans lequel est stocké l'objet.



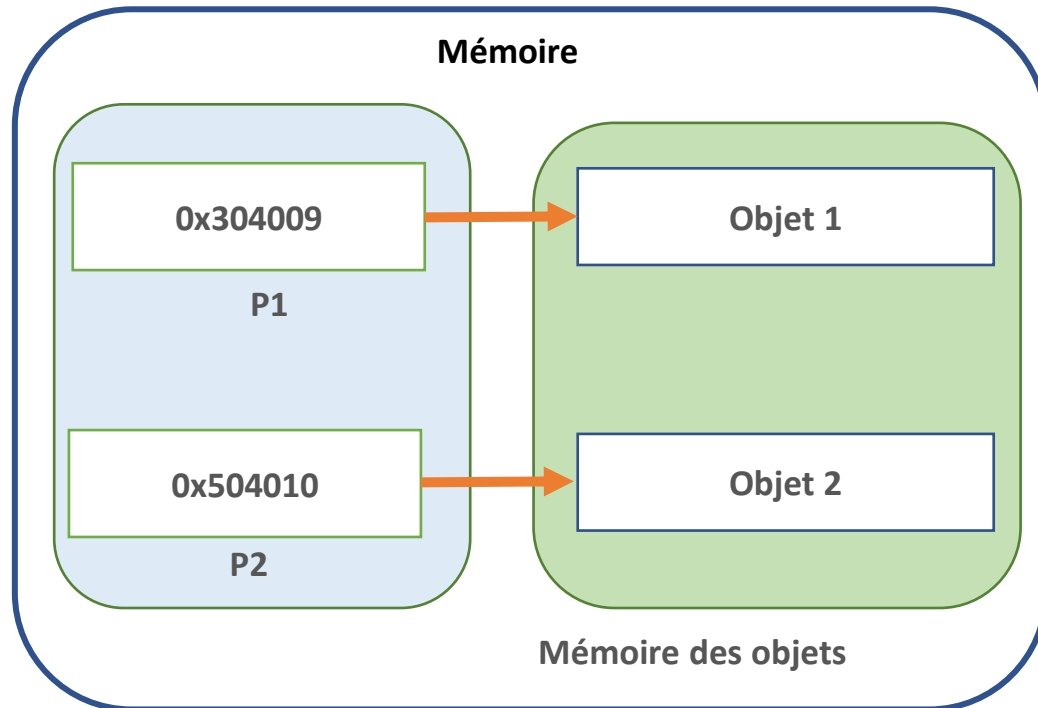


## 03 - Créer un objet

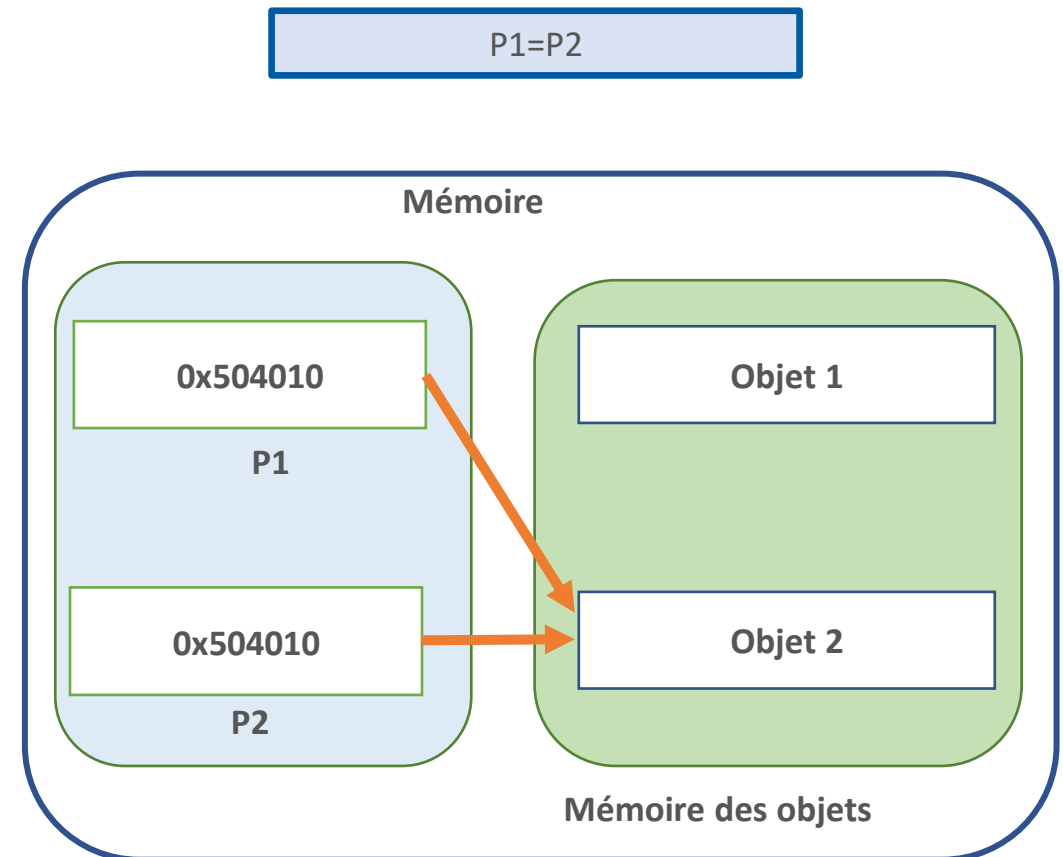
### Définition d'un objet

#### Notion d'objet

- Plusieurs référents peuvent référer un même objet → **Adressage indirect**



Recopie l'adresse de P2 dans le référent de P1



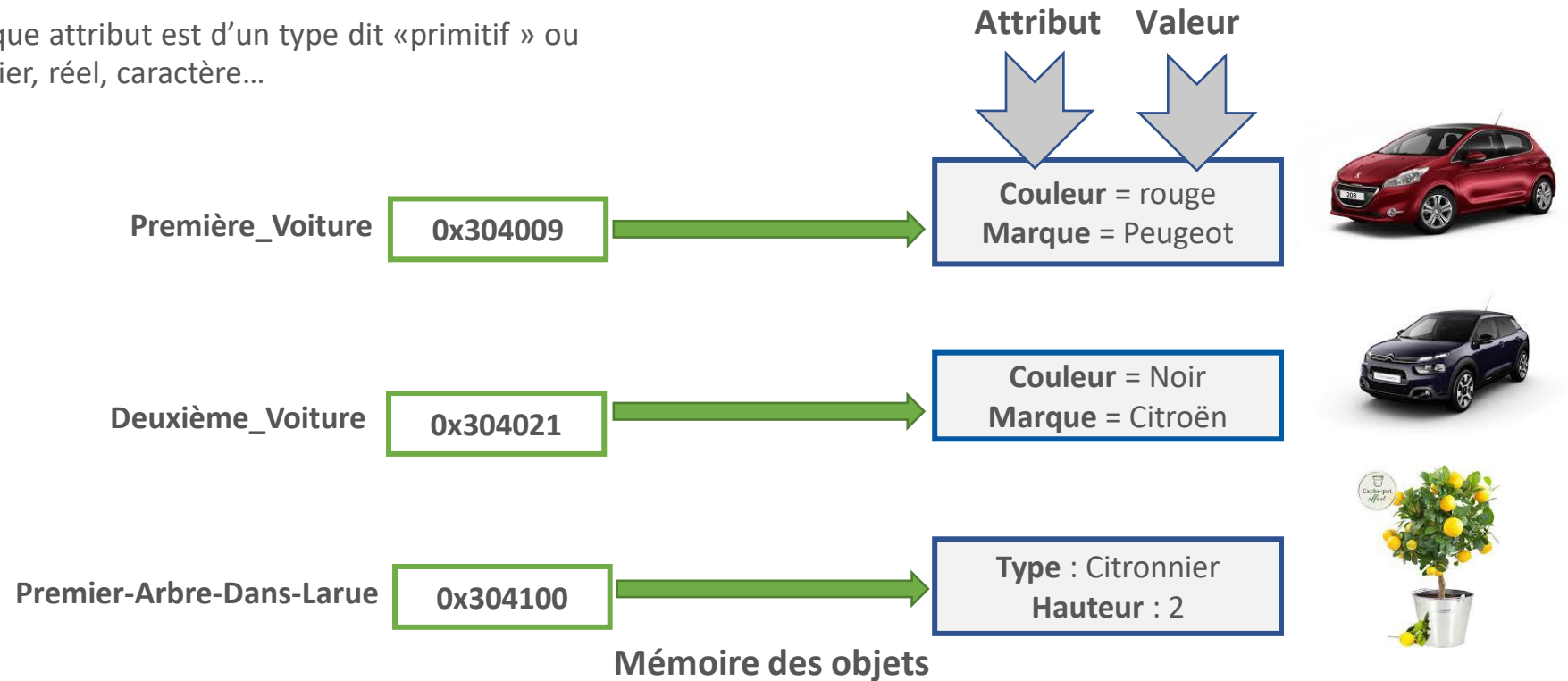
## 03 - Créer un objet

### Définition d'un objet

#### Notion d'objet

- Un objet est capable de sauvegarder un état c'est-à-dire un ensemble d'information dans les **attributs**
- **Les attributs** sont l'ensemble des informations permettant de représenter l'état de l'objet
- Exemple d'objets où chaque attribut est d'un type dit « primitif » ou « prédéfini », comme entier, réel, caractère...

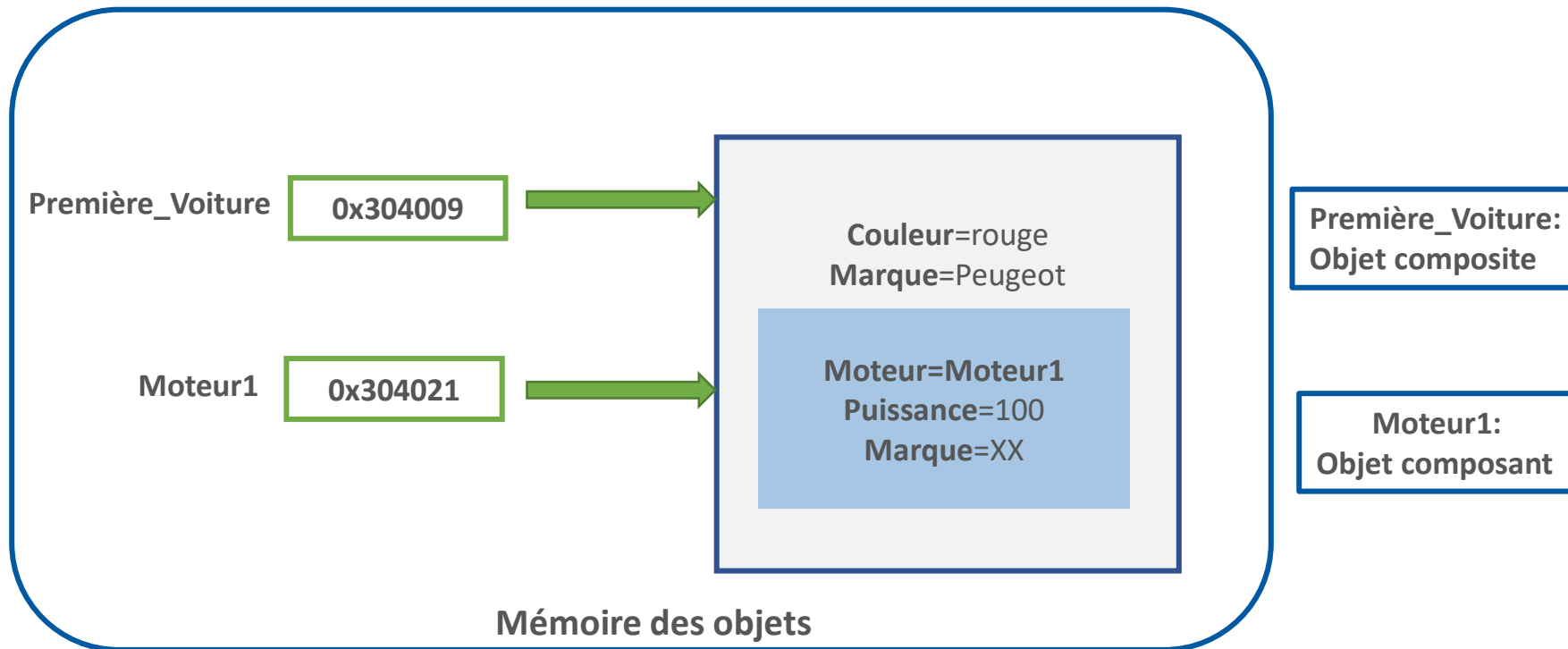
**OBJET = Référent + Etat + Comportement**



## 03 - Créer un objet

### Définition d'un objet

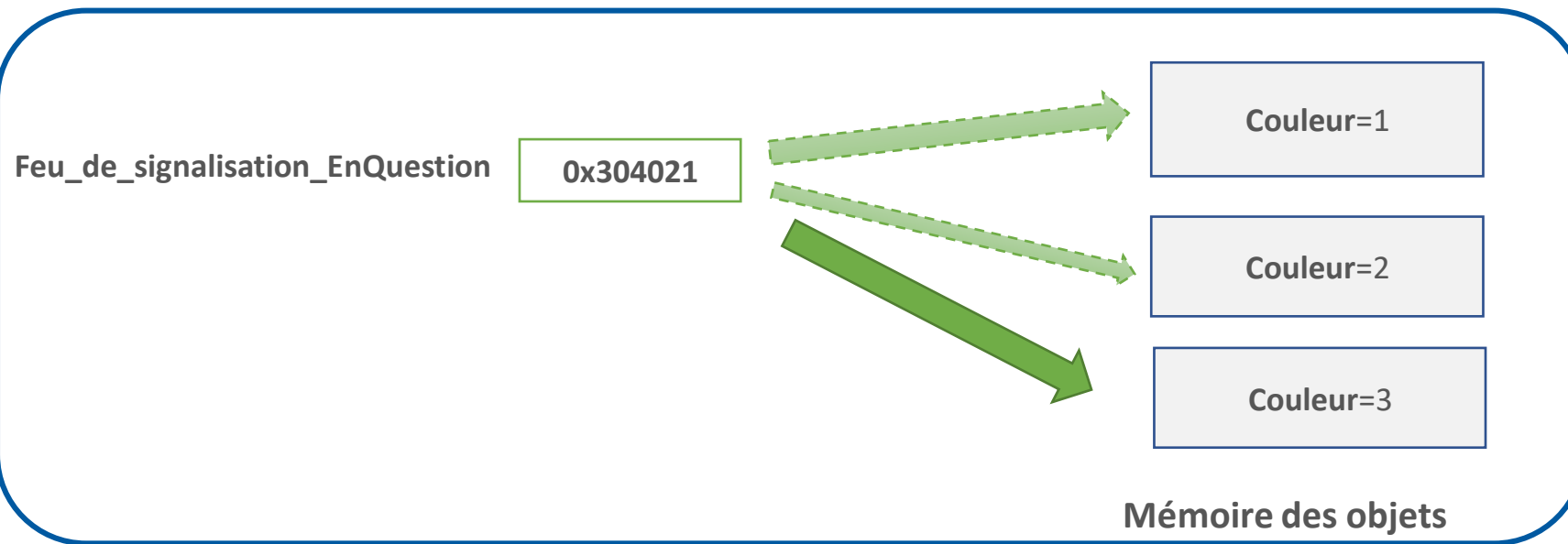
- Un objet stocké en mémoire peut être placé à l'intérieur de l'espace mémoire réservé à un autre. Il s'agit d'un **Objet Composite**



## 03 - Créer un objet

### Définition d'un objet

- Les objets changent d'état
- Le cycle de vie d'un objet se limite à une succession de changements d'états
- Soit l'objet Feu\_de\_signalisation\_EnQuestion avec sa couleur et ses trois valeurs

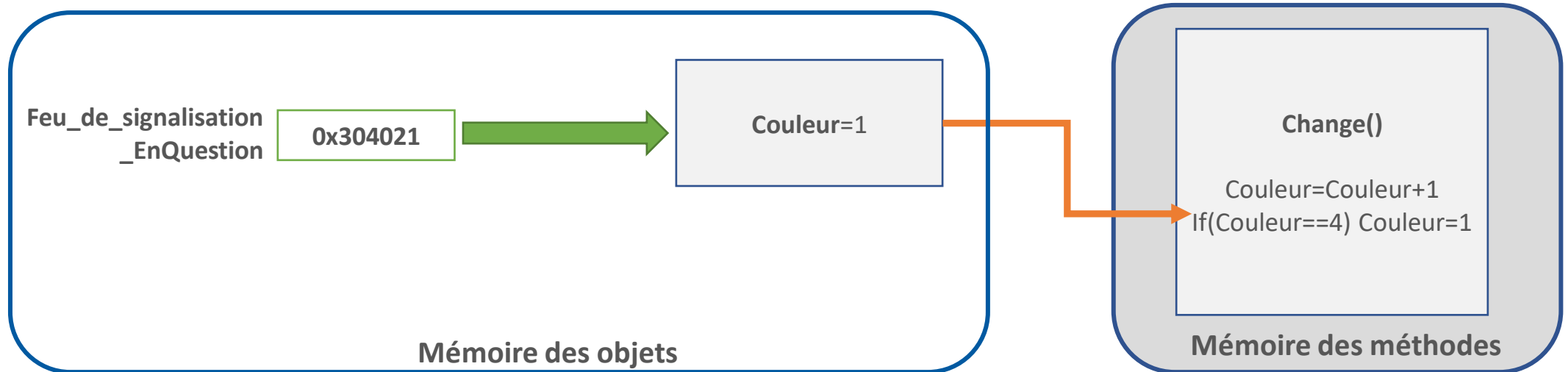


## 03 - Créer un objet

### Définition d'un objet

**OBJET= Référent + Etat + Comportement**

**Mais quelle est la cause des changements d'états ?  
– LES METHODES**



# CHAPITRE 3

## Créer un objet

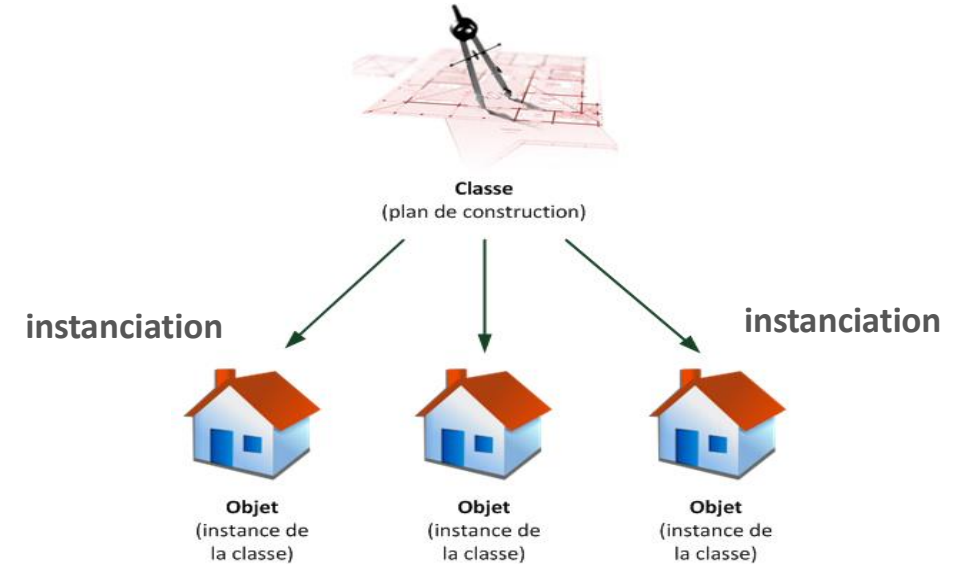
1. Définition d'un objet
2. **Instanciation**
3. Destruction explicite d'un objet



## 03 - Créer un objet

### Instanciation

- L'instanciation d'un objet est constituée de deux phases :
  - **Une phase de ressort de la classe** : créer l'objet en mémoire
  - **Une phase du ressort de l'objet** : initialiser les attributs
- L'instanciation explicite d'un objet se fait via un **constructeur**. Il est appelé automatiquement lors de la création de l'objet dans la mémoire



## CHAPITRE 3

### Créer un objet

1. Définition d'un objet
2. Instanciation
3. **Destruction explicite d'un objet**





## 03 - Créer un objet

### Destruction explicite d'un objet



- **Rappel** : un destructeur est une méthode particulière qui permet **la destruction d'un objet non référencé**
  - La destruction explicite d'un objet se fait en faisant appel au destructeur de la classe

# CHAPITRE 4

## Connaitre l'encapsulation

**Ce que vous allez apprendre dans ce chapitre :**

- Comprendre le principe d'encapsulation
- Connaitre les modificateurs et les accesseurs d'une classe : Syntaxe et utilité



**01 heure**

# CHAPITRE 4

## Connaitre l'encapsulation

1. Principe de l'encapsulation
2. Modificateurs et accesseurs (getters, setters, ...)

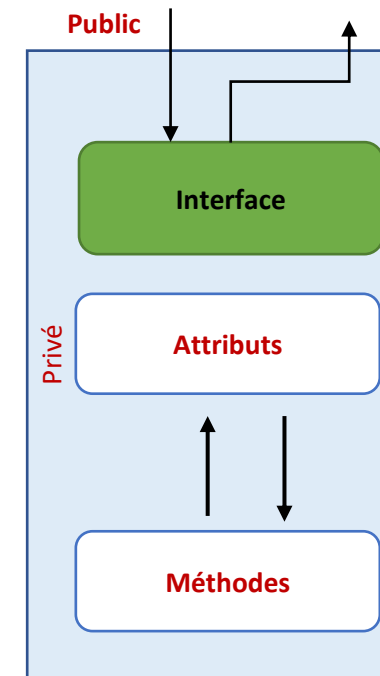
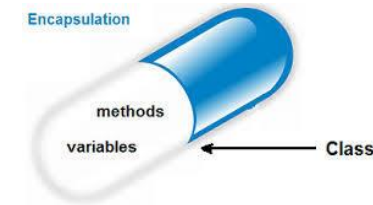


## 04 - Connaitre l'encapsulation

### Principe de l'encapsulation

#### Principe de l'encapsulation

- L'encapsulation est le fait de réunir à l'intérieur d'une même entité (objet) le code (méthodes) + données (attributs)
- L'encapsulation consiste à protéger l'information contenue dans un objet
- **Il est donc possible de masquer les informations d'un objet aux autres objets.**
- L'encapsulation consiste donc à masquer les détails d'implémentation d'un objet, en définissant une **interface**
- L'interface est **la vue externe d'un objet**, elle définit les services **accessibles** (Attributs et méthodes) aux utilisateurs de l'objet
  - **interface = liste des signatures des méthodes accessibles**
  - **Interface = Carte de visite de l'objet**



## 04 - Connaitre l'encapsulation

### Principe de l'encapsulation

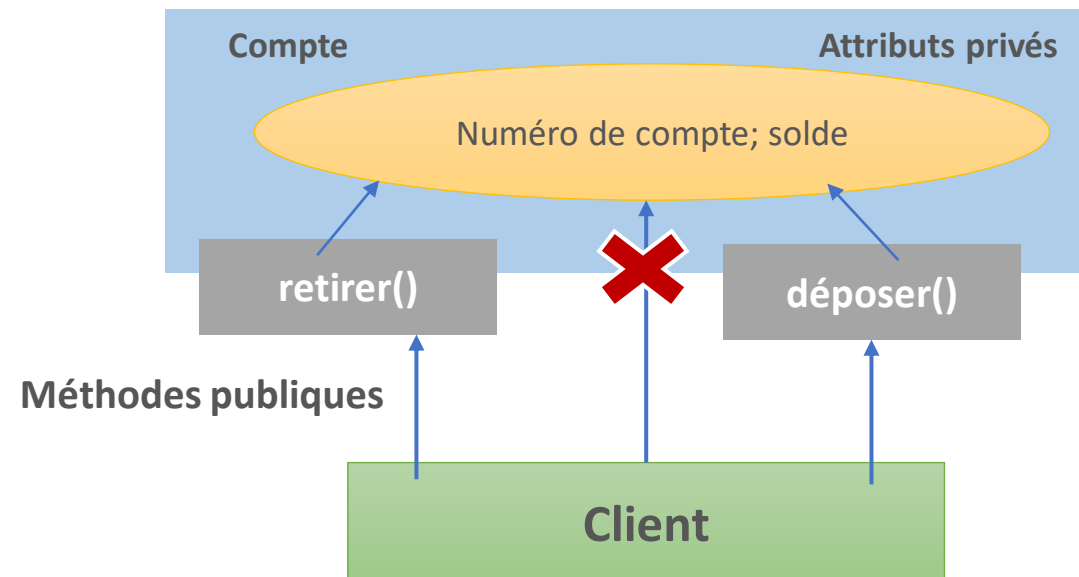
- Les objets ne restreignent leur accès qu'aux méthodes de leur classe

→ Ils protègent leurs attributs

- Cela permet d'avoir un contrôle sur tous les accès

Exemple :

- Les attributs de la classe compte sont privés
  - Ainsi le solde d'un compte n'est pas accessible par un client qu'à travers les méthodes retirer() et déposer() qui sont publiques
- Un client ne peut modifier son solde qu'en effectuant une opération de dépôt ou de retrait



# CHAPITRE 4

## Connaitre l'encapsulation

1. Principe de l'encapsulation
2. **Modificateurs et accesseurs (getters, setters, ...)**



## 04 - Connaitre l'encapsulation

### Modificateurs et accesseurs (getters, setters, ...)

#### Modificateurs et accesseurs

- Pour interagir avec les attributs d'un objet de l'extérieur, il suffit de créer des méthodes publiques dans la classe appelée **Accesseurs et Modificateurs**
- **Accesseurs (getters):** Méthodes qui retournent la valeur d'un attribut d'un objet (un attribut est généralement privé)
- La notation utilisée est **getXXX** avec XXX l'attribut retourné
- **Modificateurs (setters):** Méthodes qui modifient la valeur d'un attribut d'un objet
- La notation utilisée est **setXXX** avec XXX l'attribut modifié

Voiture
Couleur: chaîne Marque: chaîne

Première_Voiture
Couleur=rouge Marque=Peugeot

getCouleur()

Quelle est la couleur de Première\_Voiture?

Rouge

Première_Voiture
Couleur=rouge Marque=Peugeot

setCouleur(Bleu)

Couleur = bleu

Couleur = Bleu

# CHAPITRE 5

## Manipuler les méthodes

**Ce que vous allez apprendre dans ce chapitre :**

- Manipuler les méthodes : définition et appel
- Différencier entre méthode d'instance et méthode de classe



**01 heure**



# CHAPITRE 5

## Manipuler les méthodes

1. Définition d'une méthode
2. Visibilité d'une méthode
3. Paramètres d'une méthode
4. Appel d'une méthode
5. Méthodes de classe

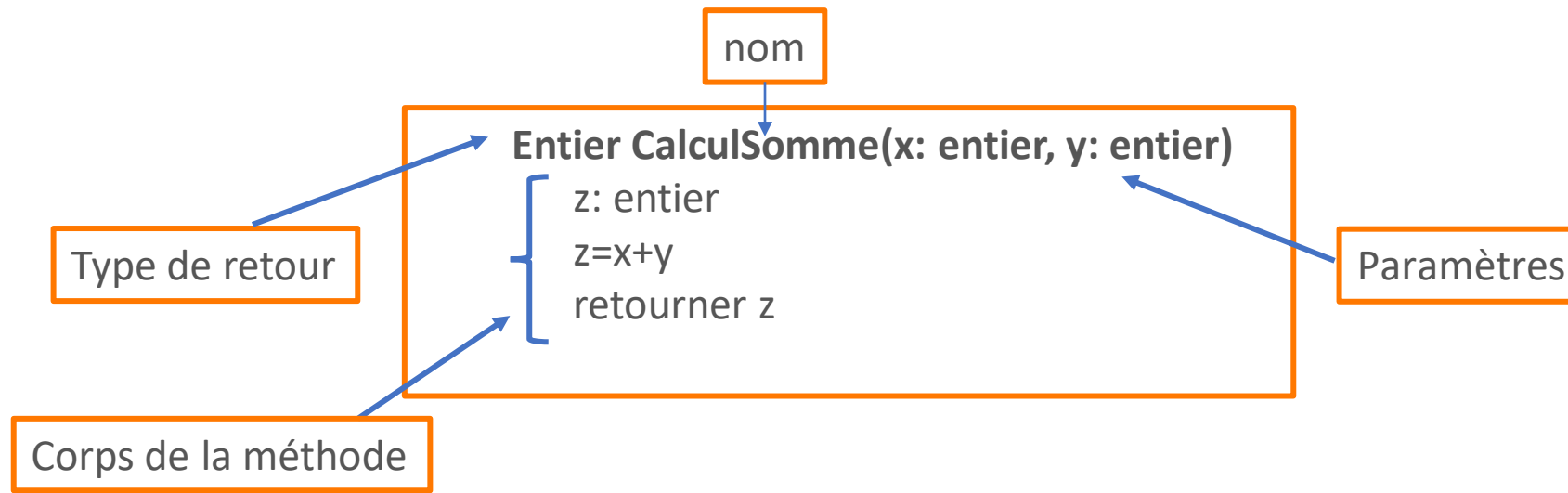


## 05 - Manipuler les méthodes

### Définition d'une méthode

- La définition d'une méthode ressemble à celle d'une procédure ou d'une fonction. Elle se compose d'une entête et un bloc.
- **L'entête** précise :
  - **Un nom**
  - **Un type de retour**
  - Une liste (éventuellement vide) de **paramètres** typés en entrée
  - Le **bloc** est constitué d'une suite d'instructions (un bloc d'instructions) qui constituent le corps de la méthode

Exemple de syntaxe :



# CHAPITRE 5

## Manipuler les méthodes

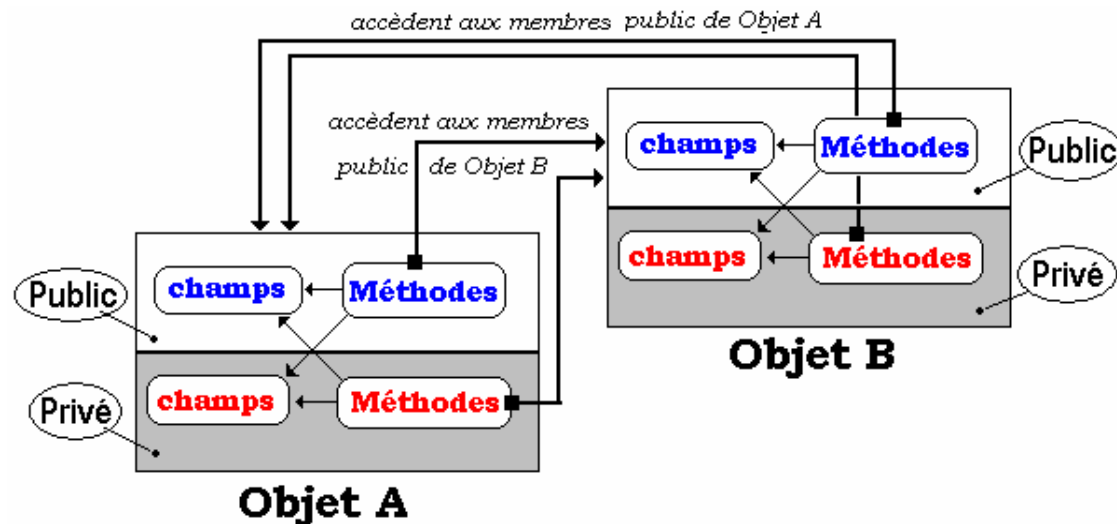
1. Définition d'une méthode
- 2. Visibilité d'une méthode**
3. Paramètres d'une méthode
4. Appel d'une méthode
5. Méthodes de classe



## 05 - Manipuler les méthodes

### Visibilité d'une méthode

- La visibilité d'une méthode définit les droits d'accès autorisant l'appel aux méthodes de la classe
- Publique (+) :**
  - L'appel de la méthode est autorisé aux méthodes de toutes les classes
- Protégée (#) :**
  - L'appel de la méthode est réservé aux méthodes des classes héritières  
( A voir ultérieurement dans la partie Héritage)
- Privée(-) :**
  - L'appel de la méthode est limité aux méthodes de la classe elle-même



# CHAPITRE 5

## Manipuler les méthodes

1. Définition d'une méthode
2. Visibilité d'une méthode
- 3. Paramètres d'une méthode**
4. Appel d'une méthode
5. Méthodes de classe



## 05 - Manipuler les méthodes

### Paramètres d'une méthode

- Il est possible de passer des arguments (appelés aussi **paramètres**) à une **méthode**, c'est-à-dire lui fournir le nom d'une variable afin que la **méthode** puisse effectuer des opérations sur ces arguments ou bien grâce à ces arguments.

**Entier CalculSomme(x: entier, y: entier)**

z: entier

z=x+y

retourner z

Paramètres



# CHAPITRE 5

## Manipuler les méthodes

1. Définition d'une méthode
2. Visibilité d'une méthode
3. Paramètres d'une méthode
- 4. Appel d'une méthode**
5. Méthodes de classe



## 05 - Manipuler les méthodes

### Appel d'une méthode

- Les méthodes ne peuvent être définies comme des composantes d'une classe.
- Une méthode est appelée pour un objet. Cette méthode est appelée **méthode d'instance**.
- L'appel d'une méthode pour un objet se réalise en nommant l'objet suivi d'un point suivi du nom de la méthode et sa liste d'arguments

**nomObjet.nomMethode(arg1, arg2,..)**

Où

- **nomObjet** : nom de la référence à l'objet
- **nomMéthode** : nom de la méthode.

**Exemple :** la classe *Véhicule*

<i>Véhicule</i>
<i>Marque :string</i>
<i>Puissance :integer</i>
<i>Vitesse-max :integer</i>
<i>Vitesse-courante :ini</i>
<i>Démarrer()</i>
<i>Accélérer()</i>
<i>Avancer()</i>
<i>Reculer()</i>

**v est objet de type Véhicule**  
**v.Démarrer()**



# CHAPITRE 5

## Manipuler les méthodes

1. Définition d'une méthode
2. Visibilité d'une méthode
3. Paramètres d'une méthode
4. Appel d'une méthode
5. **Méthodes de classe**



## 05 - Manipuler les méthodes

### Méthodes de classe

- Méthode de classe est une méthode déclarée dont l'exécution ne dépend pas des objets de la classe
- Étant donné qu'elle est liée à la classe et non à ses instances, on préfixe le nom de la méthode par le nom de la classe

**nomClasse.nomMethode(arg1, arg2,..)**

Exemple :



X
Entier CalculSomme(x: entier, y: entier)

**X.CalculSomme (2,5)**

- Puisque les méthodes de classe appartiennent à la classe, elles ne peuvent en aucun cas accéder aux attributs d'instances qui appartiennent aux instances de la classe
- Les méthodes d'instances accèdent aux attributs d'instance et méthodes d'instance
- Les méthodes d'instances accèdent aux attributs de classe et méthodes de classe
- Les méthodes de classe accèdent aux attributs de classe et méthodes de classe
- Les méthodes de classe n'accèdent pas aux attributs d'instance et méthodes d'instance



## PARTIE 2

### Connaître les principaux piliers de la POO

**Dans ce module, vous allez :**

- Maitriser le principe d'héritage en POO
- Maitriser le concept du polymorphisme en POO
- Maitriser le principe de l'abstraction en POO et son utilité
- Manipuler les interfaces (définition et implémentation)



**15 heures**

# CHAPITRE 1

## Définir l'héritage

### Ce que vous allez apprendre dans ce chapitre :

- Comprendre le principe de l'héritage et distinguer ses types
- Comprendre le principe de surcharge de constructeur
- Maitriser le chainage des constructeurs
- Maitriser la visibilité des membres d'une classe dérivée



06 heures



# CHAPITRE 1

## Définir l'héritage

1. Principe de l'héritage
2. Types d'héritage
3. Surcharge des constructeurs et chaînage
4. Visibilité des attributs et des méthodes de la classe fille



# 01 - Définir l'héritage

## Principe de l'héritage



### Principe de l'héritage

- Le concept de l'héritage spécifie une relation de **spécialisation/généralisation** entre les classes (document : livre, revu, .../Personne : étudiant, employé...)
- Lorsqu'une classe D hérite d'une classe B :
  - D possède toutes les caractéristiques de B et aussi, d'autres caractéristiques qui sont spécifiques à D
  - D est une spécialisation de B (un cas particulier)
  - B est une généralisation de D (cas général)
  - D est appelée classe dérivée (fille)
  - B est appelée classe de base (classe mère ou super-classe)
- **Tout objet instancié de D est considéré, aussi, comme un objet de type B**
- **Un objet instancié de B n'est pas forcément un objet de type D**

# 01 - Définir l'héritage

## Principe de l'héritage

### Principe de l'héritage

#### Exemple :

- Considérons la définition des 3 classes Personne, Etudiant et Employé suivantes :

Personne
Nom: chaîne CIN: entier anneeNaiss: entier
age()

Etudiant
Nom: chaîne CIN: entier anneeNaiss: entier note1, note2: réel
age() moyenne()

Employé
Nom: chaîne CIN: entier anneeNaiss: entier prixHeure: réel nbreHeure: réel
age() Salaire()

#### Problème :

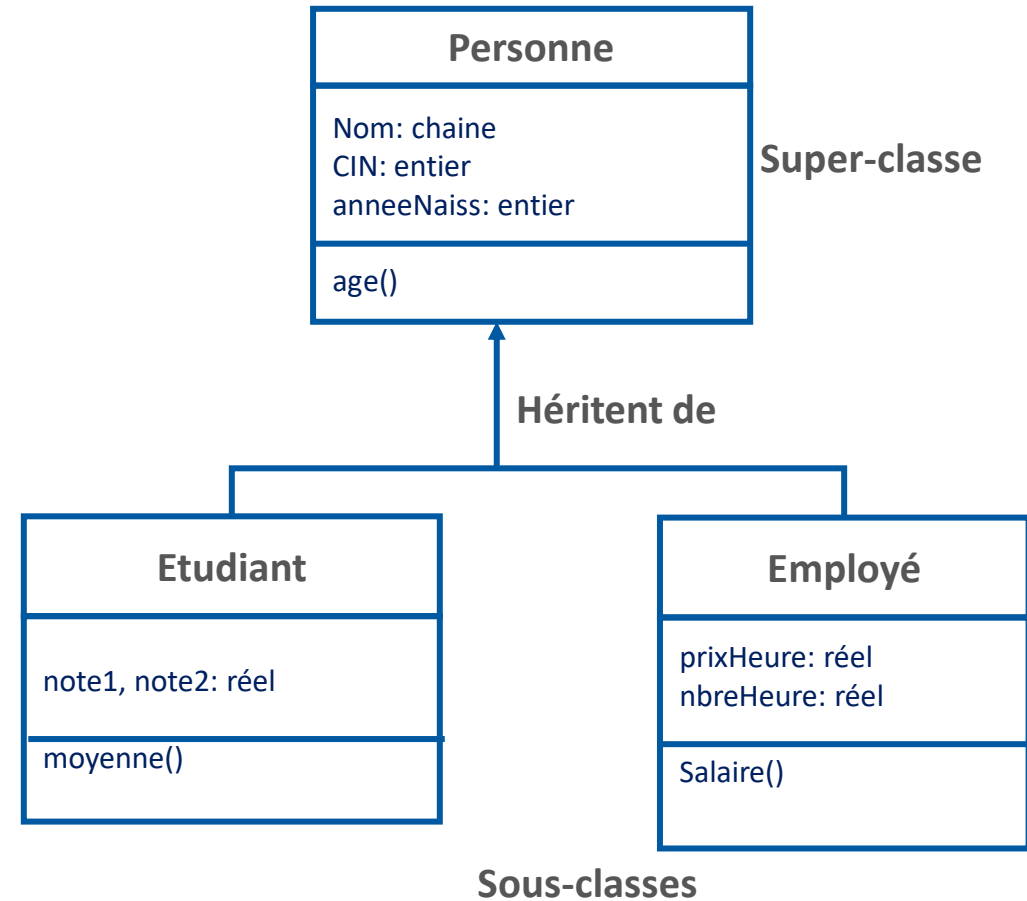
- Duplication du code
- Une modification faite sur un attribut ou méthode doit être refaite sur les autres classes

## 01 - Définir l'héritage

### Principe de l'héritage

#### Solution :

- Placer dans la **classe mère** toutes les informations communes à toutes les classes.
- Les classes filles ne comportent que les attributs ou méthodes **plus spécifiques**.
- Les classes filles **héritent** automatiquement des attributs (et des méthodes) qui n'ont pas besoin d'être réécrits.





# 01 - Définir l'héritage

## Principe de l'héritage



### Intérêt de l'héritage

- L'héritage minimise l'écriture du code en **regroupant les caractéristiques communes** entre classes au sein d'une seule (la classe de base) sans duplication
- La rectification du code se fait dans des endroits uniques grâce à la **non-redondance de description**
- **L'extension** ou **l'ajout de nouvelles classes** est favorisée surtout en cas d'une hiérarchie de classes bien conçue
- Définition des informations et des comportements aux niveaux opportuns
- **Rapprochement** de la modélisation des systèmes d'information **aux cas réels**

# CHAPITRE 1

## Définir l'héritage

1. Principe de l'héritage
- 2. Types d'héritage**
3. Surcharge des constructeurs et chaînage
4. Visibilité des attributs et des méthodes de la classe fille



## 01 - Définir l'héritage

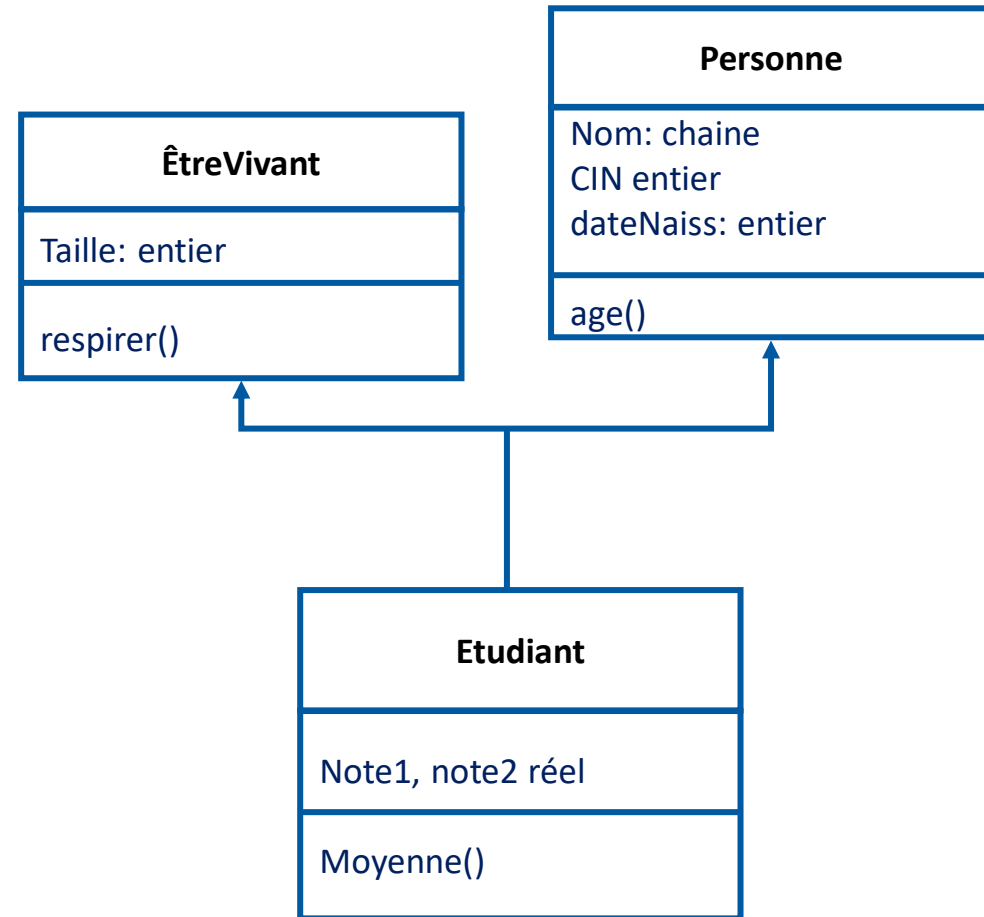
### Types d'héritage

### Héritage multiple

- Une classe peut hériter de plusieurs classes

#### Exemple :

- Un Etudiant est à la fois une Personne et un EtreVivant
  - La classe Etudiant hérite des attributs et des méthodes des
  - deux classes
- Il deviennent ses propres attributs et méthodes



## 01 - Définir l'héritage

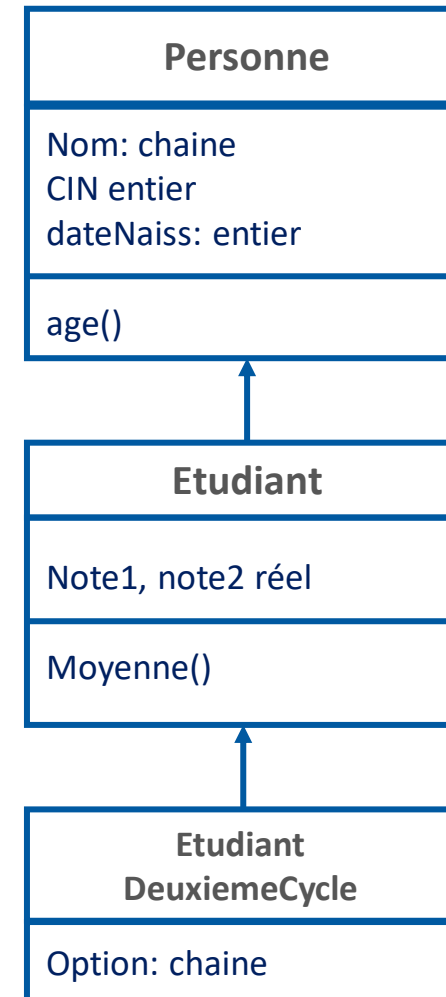
### Types d'héritage

#### Héritage en cascade

- Une classe sous-classe peut être elle-même une super-classe

##### Exemple :

- Etudiant hérite de Personne
  - EtudiantDeuxièmeCycle hérite de Etudiant
- EtudiantDeuxièmeCycle hérite de Personne



# CHAPITRE 1

## Définir l'héritage

1. Principe de l'héritage
2. Types d'héritage
- 3. Surcharge des constructeurs et chaînage**
4. Visibilité des attributs et des méthodes de la classe fille



# 01 - Définir l'héritage

## Surcharge des constructeurs et chaînage

### Surcharge du constructeur

- Les constructeurs portant le **même nom** mais **une signature différente** sont appelés constructeurs surchargés
- La signature d'un constructeur comprend les types des paramètres et le nombre des paramètres
- Dans une classe, les constructeurs peuvent avoir différents nombres d'arguments, différentes séquences d'arguments ou différents types d'arguments
- Nous pouvons avoir plusieurs constructeurs mais lors de la création d'un objet, le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments

Surcharge du  
constructeur  
CréerPersonne

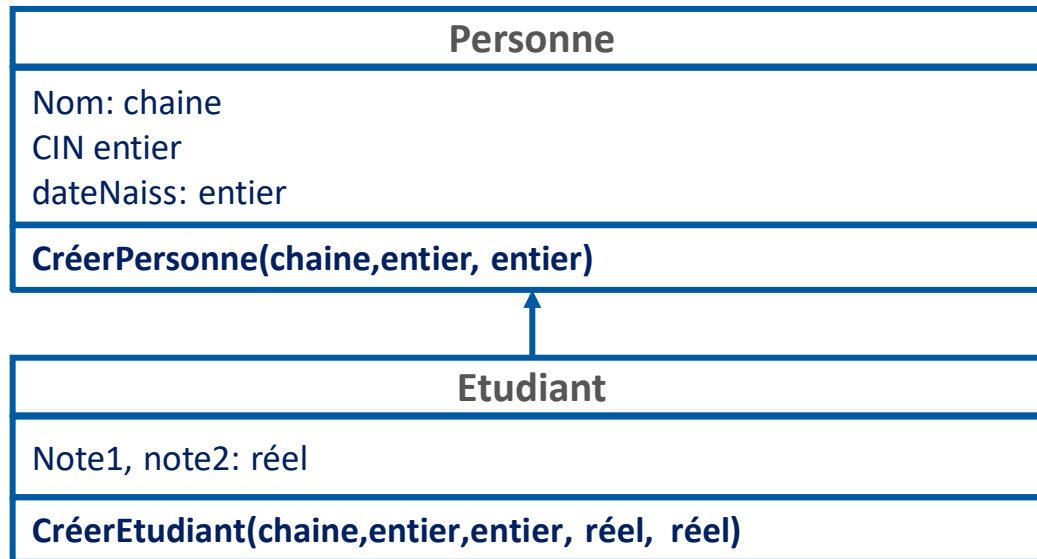
Personne
Nom: chaine CIN: entier dateNaiss: entier
age() <b>CréerPersonne(chaine, entier, entier)</b> <b>CréerPersonne(chaine, entier)</b> <b>CréerPersonne(entier, chaine, entier)</b>

# 01 - Définir l'héritage

## Surcharge des constructeurs et chaînage

### Chainage des constructeurs

- Le chaînage de constructeurs est une technique **d'appel d'un constructeur de la classe mère à partir d'un constructeur de la classe fille**
- Le chaînage des constructeurs permet **d'éviter la duplication du code** d'initialisation des attributs qui sont hérités par la classe fille



CréerEtudiant(Nom: chaine, CIN entier, dateNaiss: entier, note1: réel, note2: réel)

CréerPersonne(Nom, CIN, dateNaiss)

note1=15

note2=13.5

appel du constructeur de Personne

Corps du constructeur de Etudiant

# CHAPITRE 1

## Définir l'héritage

1. Principe de l'héritage
2. Types d'héritage
3. Surcharge des constructeurs et chaînage
4. **Visibilité des attributs et des méthodes de la classe fille**





# 01 - Définir l'héritage

## Visibilité des attributs et des méthodes de la classe fille

### Visibilité des membres et héritage

- Si les **membres (attributs et méthodes) hérités** sont déclarés **privés** dans la **classe mère**, la **classe fille** n'a pas le droit de les **manipuler** directement malgré le fait qu'ils fassent parti de sa description
- Si les **membres (attributs et méthodes)** sont déclarés **protégés** dans la **classe mère**, la **classe fille** a le droit de les manipuler directement

Exemple :

**meth1Etudiant()**

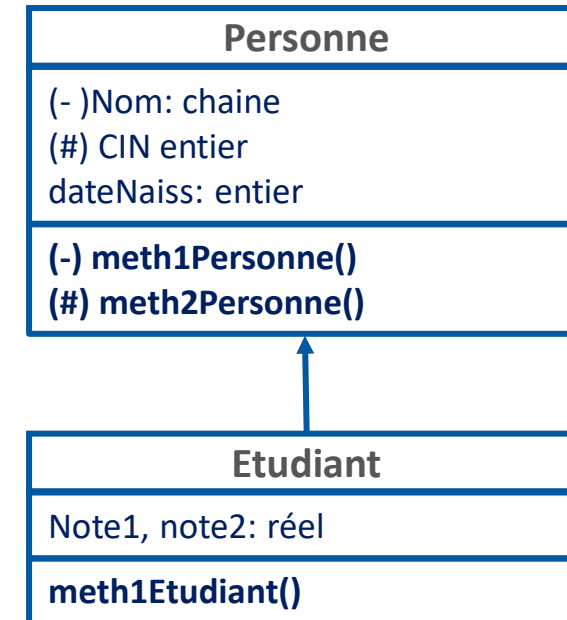
**Nom="X" → incorrect car Nom est un attribut privé dans Personne**

**CIN=12365 → Correct car CIN est déclaré protégé dans Personne**

**meth1Personne() → incorrect car la méthode est privée**

**meth2Personne() → correct car la méthode est protégée**

Corps de la méthode



## CHAPITRE 2

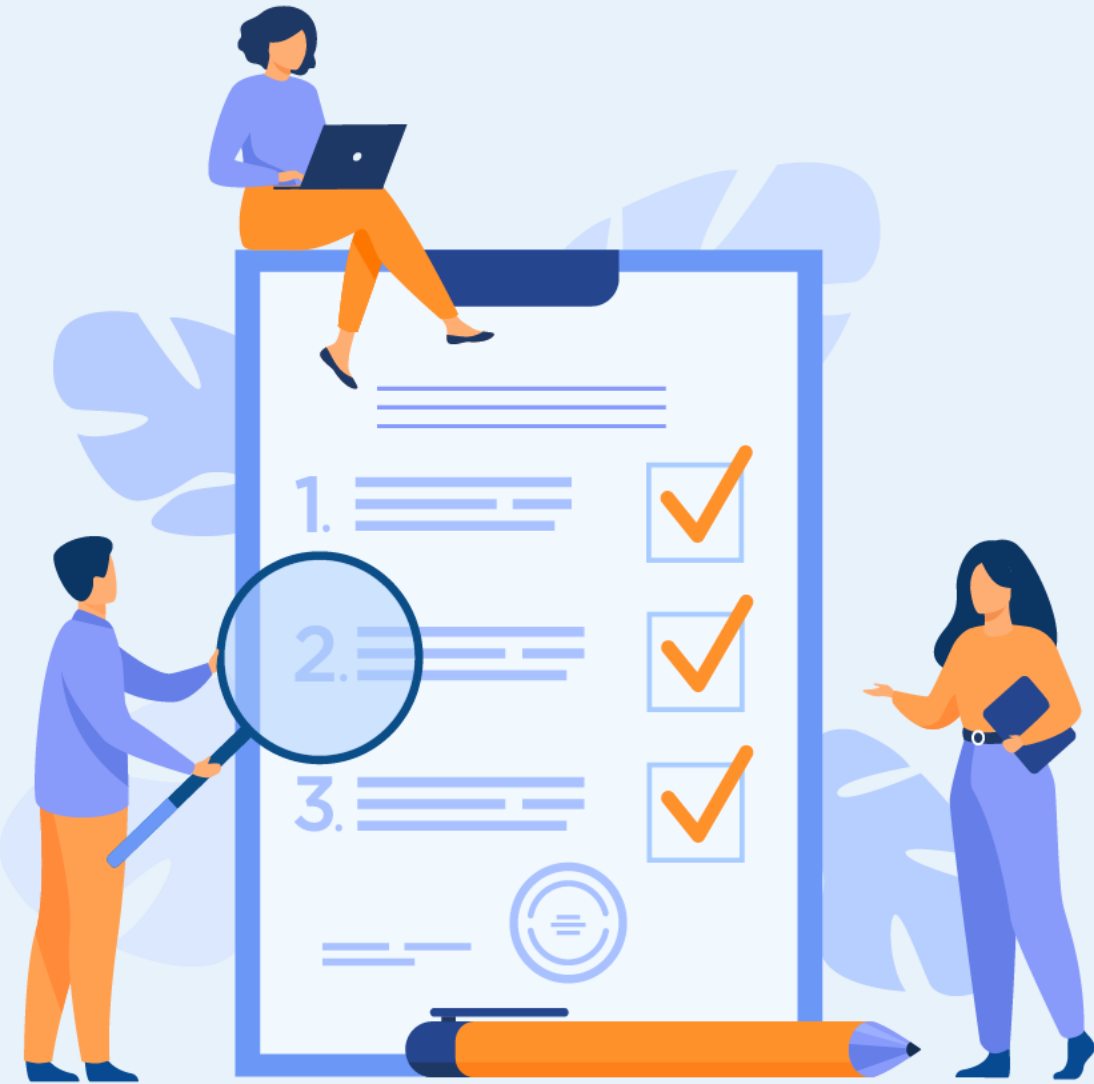
### Définir le polymorphisme

Ce que vous allez apprendre dans ce chapitre :

- Comprendre le principe de polymorphisme
- Savoir redéfinir et surcharger des méthodes



03 heures



# CHAPITRE 2

## Définir le polymorphisme

1. Principe du polymorphisme
2. Redéfinition des méthodes
3. Surcharge des méthodes



## 02 - Définir le polymorphisme

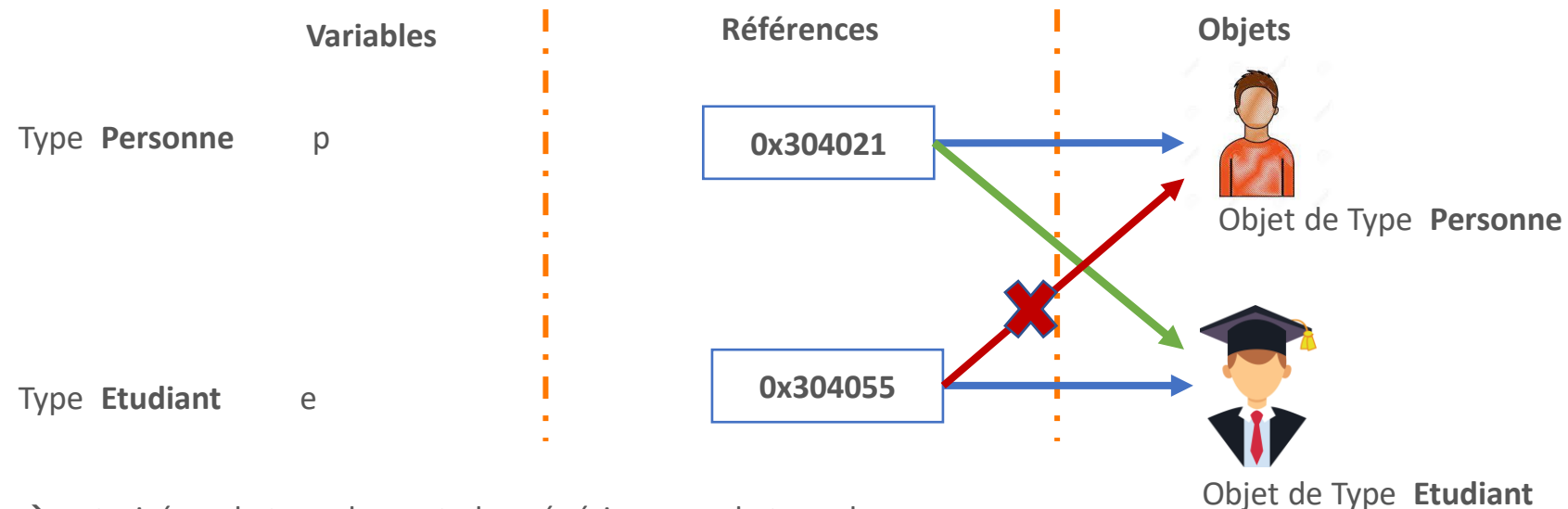
### Principe du polymorphisme

#### Principe de polymorphisme

- Le polymorphisme désigne un concept de la théorie des types, selon lequel un nom d'objet peut désigner des instances de classes différentes issues d'une même arborescence

#### Exemple :

- Une instance de Etudiant peut « être vue comme » une instance de Personne (**Pas l'inverse!!**)

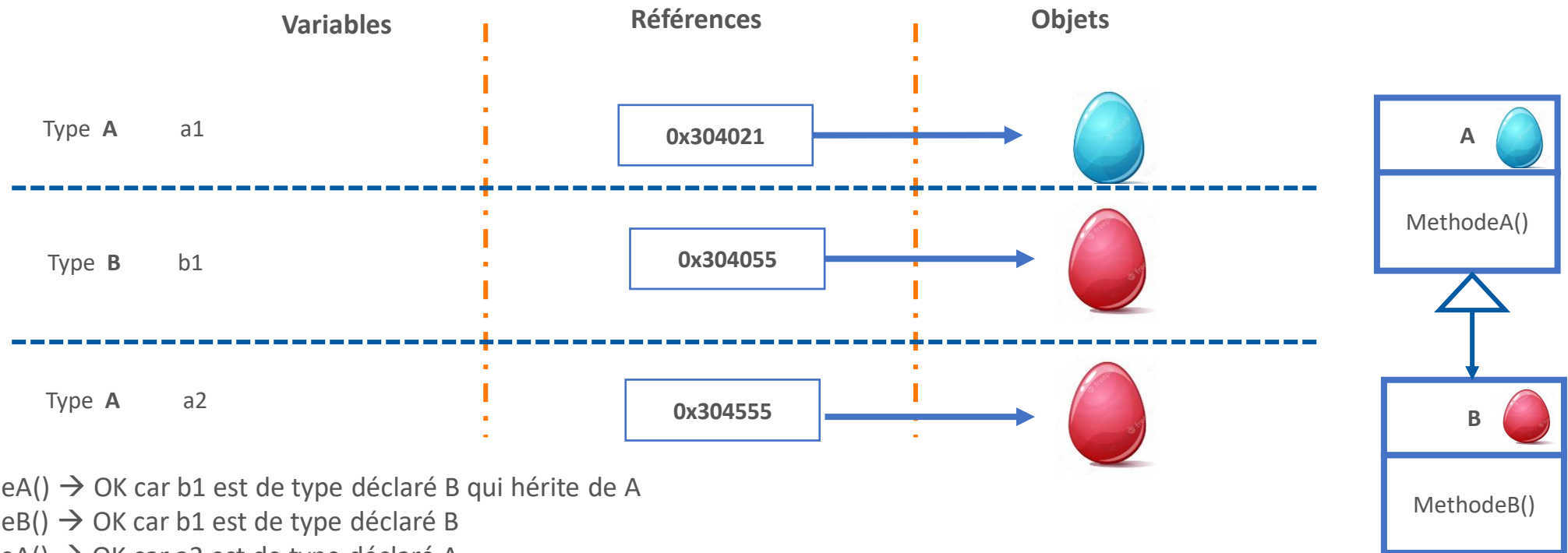


- $= e \rightarrow$  autorisé car le type de p est plus générique que le type de e
- $e = p \rightarrow$  non autorisé car le type de e est plus spécifique que celui de p

## 02 - Définir le polymorphisme

### Principe de polymorphisme

- Le type de la variable est utilisé par le compilateur pour déterminer si on accède à un membre (attribut ou méthode) valide



`b1.MethodeA()` → OK car b1 est de type déclaré B qui hérite de A

`b1.MethodeB()` → OK car b1 est de type déclaré B

`a2.MethodeA()` → OK car a2 est de type déclaré A

`a2.MethodeB()` → **ERREUR** car a2 est de type A (même si le type l'objet référencé est B)

# CHAPITRE 2

## Définir le polymorphisme

1. Principe du polymorphisme
2. **Redéfinition des méthodes**
3. Surcharge des méthodes



## 02 - Définir le polymorphisme

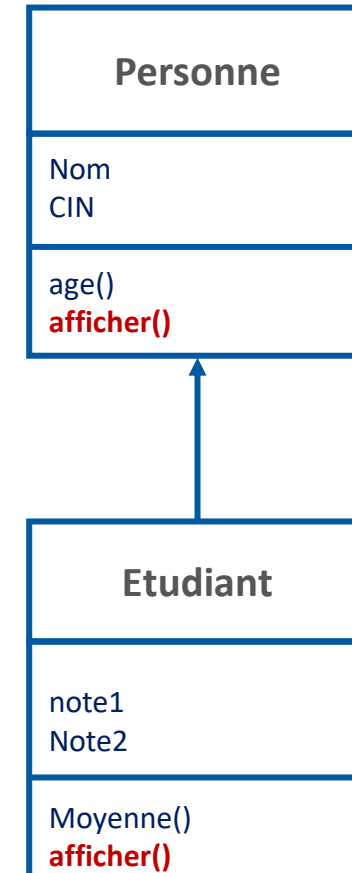
### Redéfinition des méthodes

#### Redéfinition des méthodes héritées

- Une méthode héritée peut être redéfinie si sa version initiale n'est pas satisfaisante pour la classe dérivée
- La redéfinition consiste à **conserver l'entête de la méthode** et à **proposer un code différent**
- Lors de la redéfinition d'une méthode, l'appel de l'ancienne version (celle de la classe de base) est possible
- Si une méthode héritée est redéfinie, c'est uniquement **la nouvelle version qui fait partie de la description de la classe dérivée**
- Si la méthode définie au niveau de la classe dérivée est de type différent, ou de paramètres différents, alors il s'agit d'une nouvelle méthode qui s'ajoute à celle héritée de la classe de base

#### Exemple :

- Soit la classe Etudiant qui hérite de la classe Personne
- La méthode afficher() de la classe Personne affiche les attributs Nom, CIN d'une personne
- La classe Etudiant hérite la méthode afficher() de la classe Personne et
- la **redéfinit**, elle propose un nouveau code (la classe Etudiant **ajoute** l'affichage
- des attributs notes1 et note2 de l'étudiant)



## 02 - Définir le polymorphisme

### Redéfinition des méthodes

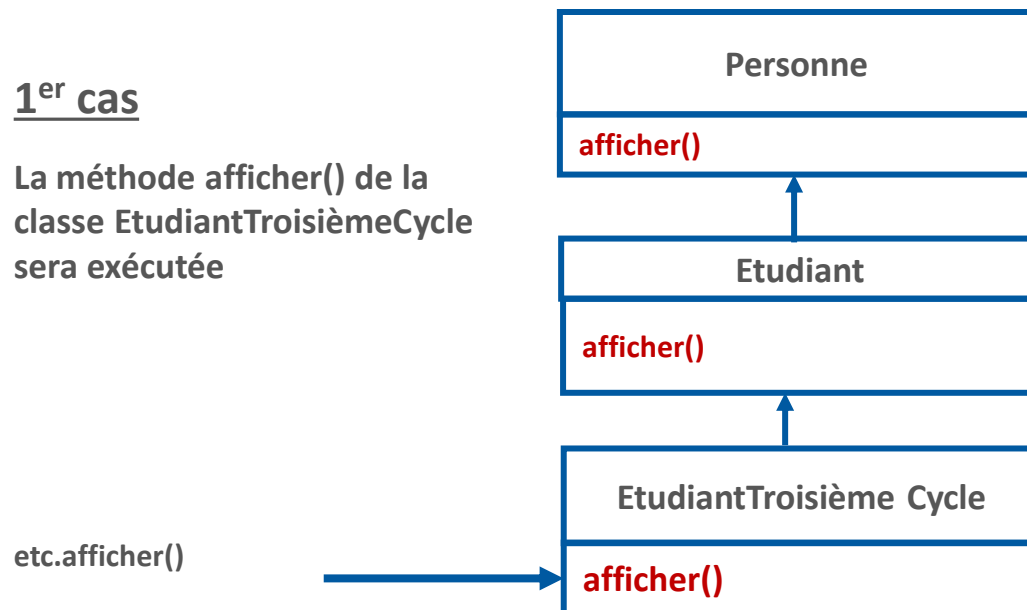
#### Mécanisme de la liaison retardée

- Soit **C** la classe réelle d'un objet **o** à qui on envoie un message « **o.m()** »
- **o.m()** peut appeler la méthode **m()** de **C** ou de n'importe quelle sous-classe de **C**
- Si le code de la classe **C** contient la définition (ou la redéfinition) d'une méthode **m()**, c'est cette méthode qui sera exécutée
- Sinon, la recherche de la méthode **m()** se poursuit dans la classe mère de **C**, puis dans la classe mère de cette classe mère, et ainsi de suite, jusqu'à trouver la définition d'une méthode **m()** qui est alors exécutée

**Exemple :** Soit **etc** un objet de type **EtudiantTroisièmeCycle**

#### 1<sup>er</sup> cas

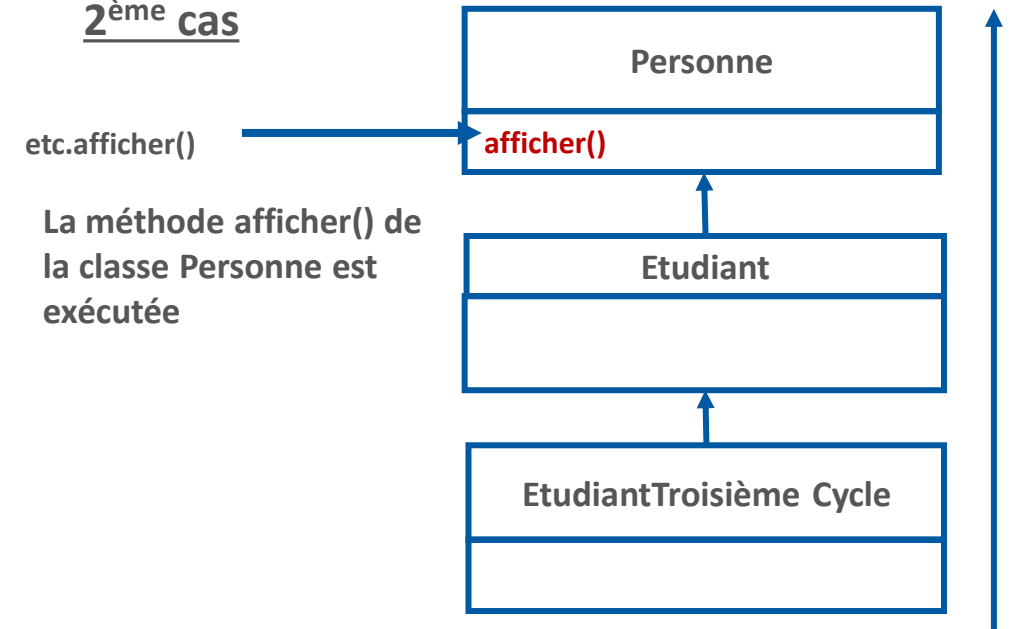
La méthode **afficher()** de la classe **EtudiantTroisièmeCycle** sera exécutée



#### 2<sup>ème</sup> cas

**etc.afficher()**

La méthode **afficher()** de la classe **Personne** est exécutée





## CHAPITRE 2

### Définir le polymorphisme

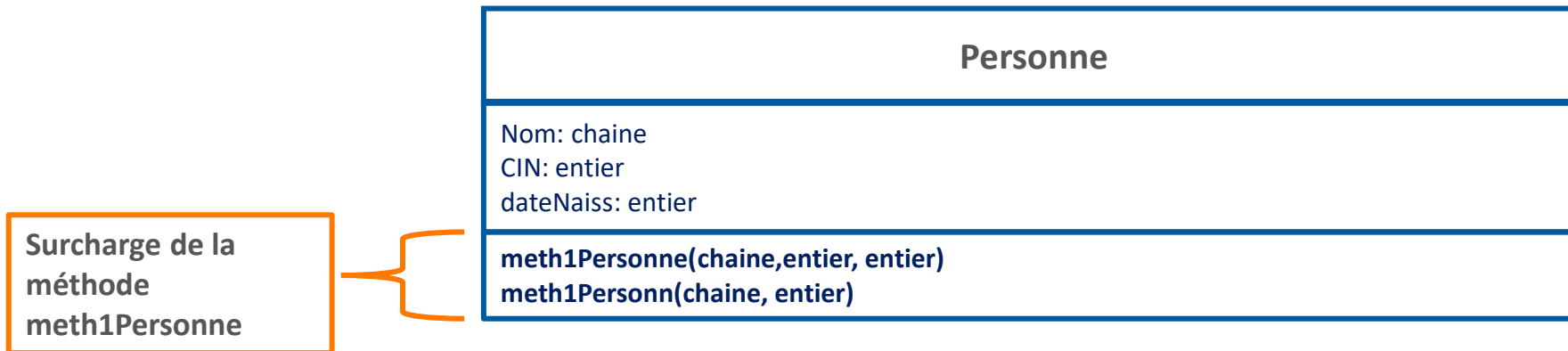
1. Principe du polymorphisme
2. Redéfinition des méthodes
3. **Surcharge des méthodes**



## 02 - Définir le polymorphisme

### Surcharge des méthodes

- La surcharge d'une méthode permet de **définir plusieurs fois une même méthode** avec **des arguments différents**.
- Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments.
- Une méthode est surchargée lorsqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.



## CHAPITRE 3

### Caractériser l'abstraction

**Ce que vous allez apprendre dans ce chapitre :**

- Comprendre le principe d'abstraction en POO
- Maîtriser les concepts de classes abstraites et méthodes abstraites



**03 heures**

# CHAPITRE 3

## Caractériser l'abstraction

1. **Principes**
2. Classes abstraites
3. Méthodes abstraites



## 03 - Caractériser l'abstraction

### Principes



#### Principe de l'abstraction

- L'abstraction est un principe qui consiste à **ignorer certains aspects** qui ne sont pas importants pour le problème dans le but de **se concentrer sur ceux qui le sont**
- En POO, c'est le fait de **se concentrer sur les caractéristiques importantes** d'un objet selon le point de vue de l'observateur
  - Son objectif principal est de **gérer la complexité** en masquant les détails inutiles à l'utilisateur
  - Cela permet à l'utilisateur d'implémenter une logique plus complexe sans comprendre ni même penser à toute la complexité cachée

# CHAPITRE 3

## Caractériser l'abstraction

1. Principes
2. **Classes abstraites**
3. Méthodes abstraites



## 03 - Caractériser l'abstraction

### Classes abstraites



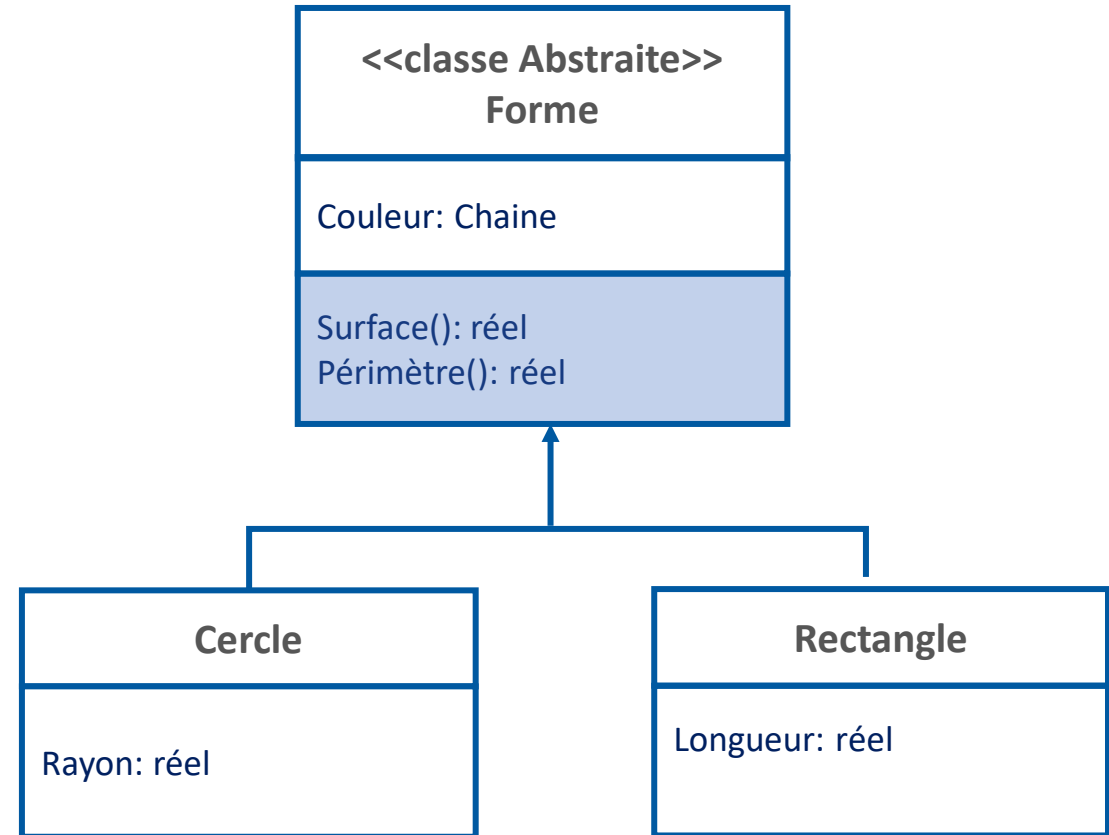
- Par opposition à une classe concrète, **une classe abstraite ne peut pas être instanciée**
- Une classe abstraite ne peut servir que de classe de base pour une dérivation
- L'utilité d'une classe abstraite est de définir un type abstrait qui regroupe des caractéristiques communes à d'autres types. En effet, elle sert :
  - **Comme racine pour un sous-arbre d'héritage :**
    - Dans lequel les sous-classes apporteront leurs particularités
    - Favorise le polymorphisme
  - **Au partage de certaines descriptions :**
    - Variables d'instances (ou de classes) communes
    - Méthodes définissant un comportement par défaut
- Les classes abstraites permettent de définir des fonctionnalités (des méthodes) que **les sous-classes** devront **impérativement implémenter**
- Les utilisateurs des sous-classes d'une classe abstraite sont donc assurés de trouver toutes les méthodes définies dans la classe abstraite dans chacune des sous-classes concrètes
- Les classes abstraites constituent donc **une sorte de contrat** (spécification contraignante) qui garantit que certaines méthodes seront disponibles dans les sous-classes et qui oblige les programmeurs à les implémenter dans toutes les sous-classes concrètes

## 03 - Caractériser l'abstraction

### Classes abstraites

#### Exemple :

- Sachant que toutes les formes possèdent les propriétés périmètre et surface, il est judicieux de placer les méthodes périmètre() et surface() dans la classe qui est à la racine de l'arborescence (Forme)
- Les méthodes surface() et périmètre() définies dans la classe Forme ne présentent pas de code et doivent impérativement être implémentées dans les classes filles
- Les méthodes surface() et périmètre() sont des méthodes abstraites





# CHAPITRE 3

## Caractériser l'abstraction

1. Principes
2. Classes abstraites
3. **Méthodes abstraites**



## 03 - Caractériser l'abstraction

### Méthodes abstraites



- Une **méthode abstraite** est une méthode qui **ne contient pas de corps**. Elle possède simplement une signature de définition (pas de bloc d'instructions)
- Une méthode abstraite est déclarée dans **une classe abstraite**
- Une classe possédant une ou plusieurs méthodes abstraites devient obligatoirement une classe abstraite
- Une classe abstraite peut ne pas comporter de méthodes abstraites

#### Les règles suivantes s'appliquent aux classes abstraites :

- Une sous-classe d'une classe abstraite ne peut être instanciée que si elle redéfinit chaque méthode abstraite de sa classe parente et qu'elle fournit une implémentation (un corps) pour chacune des méthodes abstraites
- Si une sous-classe d'une classe abstraite n'implémente pas toutes les méthodes abstraites dont elle hérite, cette sous-classe est elle-même abstraite (et ne peut donc pas être instanciée)

## CHAPITRE 4

### Manipuler les interfaces

Ce que vous allez apprendre dans ce chapitre :

- Comprendre le principe des interfaces et leurs utilités
- Maîtriser l'implémentation d'une interface



03 heures

# CHAPITRE 4

## Manipuler les interfaces

1. Définition des interfaces
2. Utilité des interfaces
3. Implémentation des interfaces



## 04 - Manipuler les interfaces

### Définition des interfaces

- Comme une classe et une classe abstraite, une interface permet de définir un nouveau type (référence).
- Une interface est une forme particulière de classe où **toutes les méthodes sont abstraites**.

<<interface>>  
imprimable

Imprimer()

# CHAPITRE 4

## Manipuler les interfaces

1. Définition des interfaces
2. **Utilité des interfaces**
3. Implémentation des interfaces



## 04 - Manipuler les interfaces

### Utilité des interfaces



### Utilité des interfaces

L'utilisation des interfaces permet de :

- Définir (regrouper) des propriétés qui peuvent être transférées aux classes par un mécanisme particulier (l'implémentation)
- Garantir aux « clients » d'une classe que ses instances peuvent assurer certains services, ou qu'elles possèdent certaines propriétés (par exemple, être comparables à d'autres instances)
- Faire du polymorphisme avec des objets dont les classes n'appartiennent pas à la même hiérarchie d'héritage (l'interface joue le rôle de la classe mère)

# CHAPITRE 4

## Manipuler les interfaces

1. Définition des interfaces
2. Utilité des interfaces
- 3. Implémentation des interfaces**





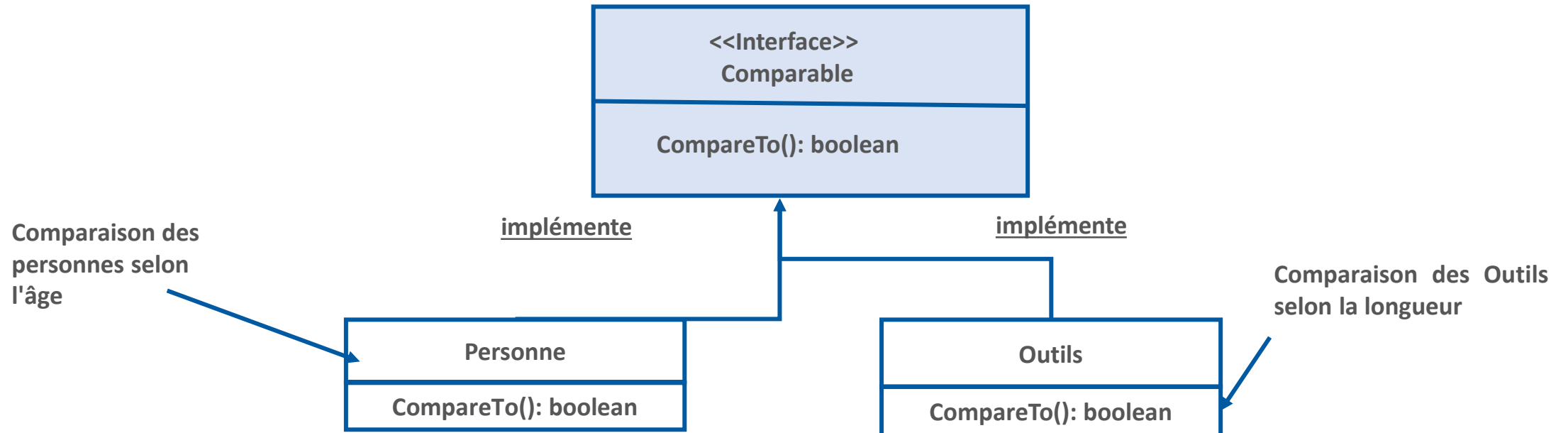
## 04 - Manipuler les interfaces

### Implémentation des interfaces

- On dit qu'une **classe implémente une interface**, si elle fournit une implémentation (c'est-à-dire un corps) pour chacune des méthodes abstraites de cette interface.
- Si une classe implémente plus d'une interface, elle doit implémenter toutes les méthodes abstraites de chacune des interfaces.

#### Exemple1 :

- Si l'on souhaite caractériser la fonctionnalité de comparaison qui est commune à tous les objets qui ont une relation d'ordre (plus petit, égal, plus grand), on peut définir l'interface Comparable.
- Les classes Personne et Outils qui implémentent l'interface Comparable **doivent présenter une implémentation de la méthode CompareTo() sinon elles seront abstraites.**



## 04 - Manipuler les interfaces

### Implémentation des interfaces

#### Exemple 2 :

- Supposons que nous voulions que les classes dérivées de la classe Forme disposent toutes d'une méthode imprimer() permettant d'imprimer les formes géométriques.

#### Solution 1:

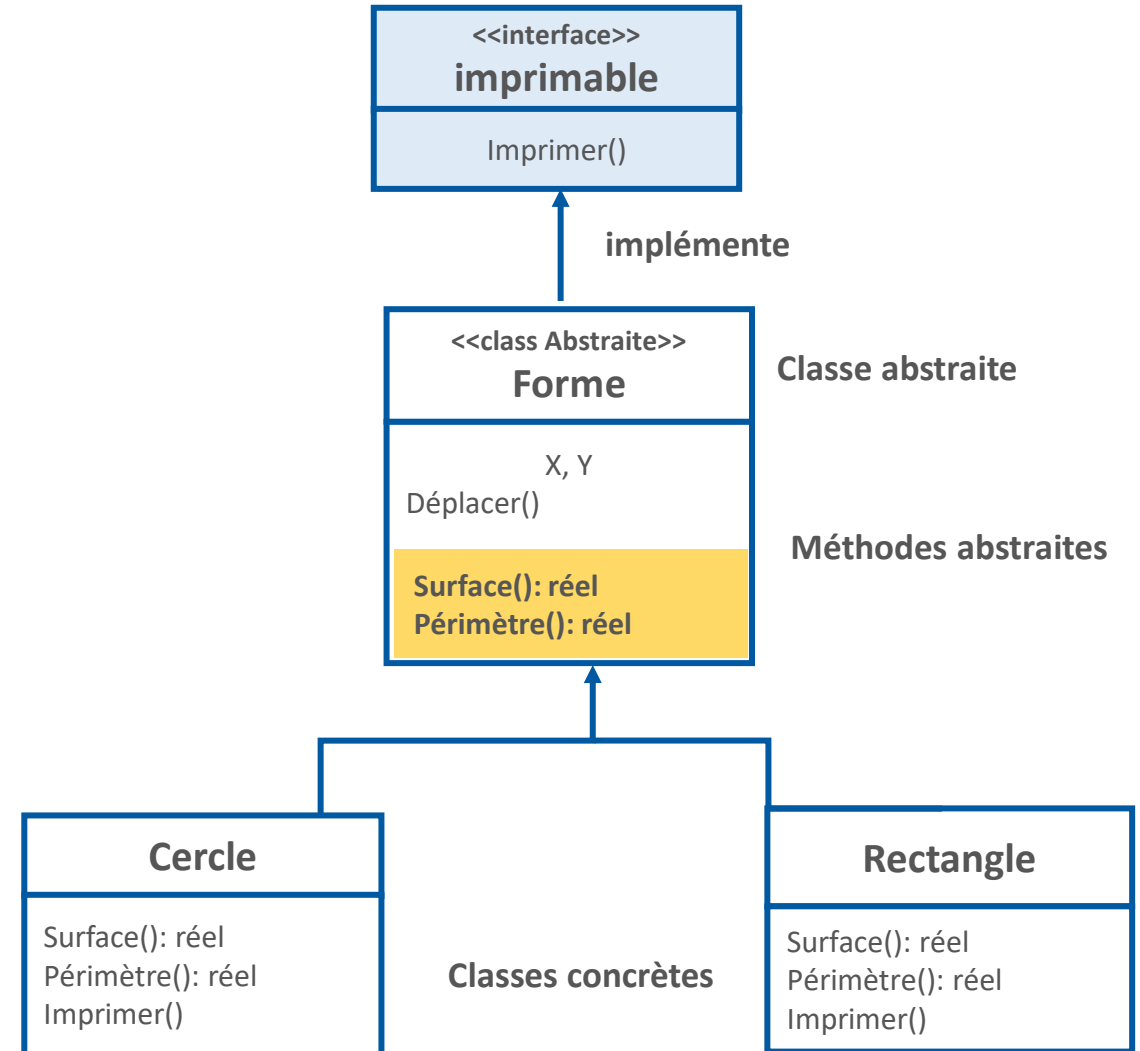
- Ajouter une méthode abstraite Imprimer() à la classe Forme et ainsi chacune des sous-classes concrètes devrait implémenter cette méthode.

→ Si d'autres classes (qui n'héritent pas de Forme) souhaitent également disposer des fonctions d'impression, elles devraient à nouveau déclarer des méthodes abstraites d'impression dans leur arborescence.

#### Solution 2 :

La classe forme implémente l'interface imprimable

- ayant la méthode Imprimer()





## PARTIE 3

### Coder des solutions orientées objet

**Dans ce module, vous allez :**

- Explorer les concepts de base de l'orienté objet en langage Python
- Manipuler correctement des données en Python (collections, listes, fichiers)
- Manipuler correctement les expressions régulières en Python
- Gérer les exceptions en Python



**50 heures**

# CHAPITRE 1

## Coder une solution orientée objet

### Ce que vous allez apprendre dans ce chapitre :

- Maîtriser le codage d'une classe en Python
- Maîtriser le codage des principaux piliers de la POO en Python à savoir l'encapsulation, l'héritage, le polymorphisme et l'abstraction



25 heures

# CHAPITRE 1

## Coder une solution orientée objet

1. **Création d'un package**
2. Codage d'une classe
3. Intégration des concepts POO



## 01 - Coder une solution orientée objet

### Création d'un package

- Un répertoire contenant des fichiers et/ou répertoires = package
- Pour créer votre propre package, commencez par créer dans le même dossier que votre programme principal (exemple main.py), un dossier portant le nom de votre package (exemple src).
- Le package src contient des fichiers sources classés dans des sous-packages
- Avant Python 3.3, un package contenant des modules Python doit contenir un fichier `__init__.py`

 src	2021-10-15 22:15	Dossier de fichiers
 main	2021-10-15 22:08	Python File

#### package src

 <code>__init__</code>	2021-10-15 22:08	Python File
 ExempleClass	2021-10-15 22:08	Python File

## 01 - Coder une solution orientée objet

### Création d'un package



- Pour importer et utiliser les classes (classe1 et classe2) définies dans ExempleClass.py, il faut ajouter dans main.py la ligne suivante :

```
from src.ExempleClass import class1, class2
```

- Si ExempleClass.py est dans subpackage qui est défini dans src il faut ajouter dans main.py la ligne suivante :

```
from src.subpackage.ExempleClass import class1, class2
```

# CHAPITRE 1

## Coder une solution orientée objet

1. Création d'un package
2. **Codage d'une classe**
3. Intégration des concepts POO



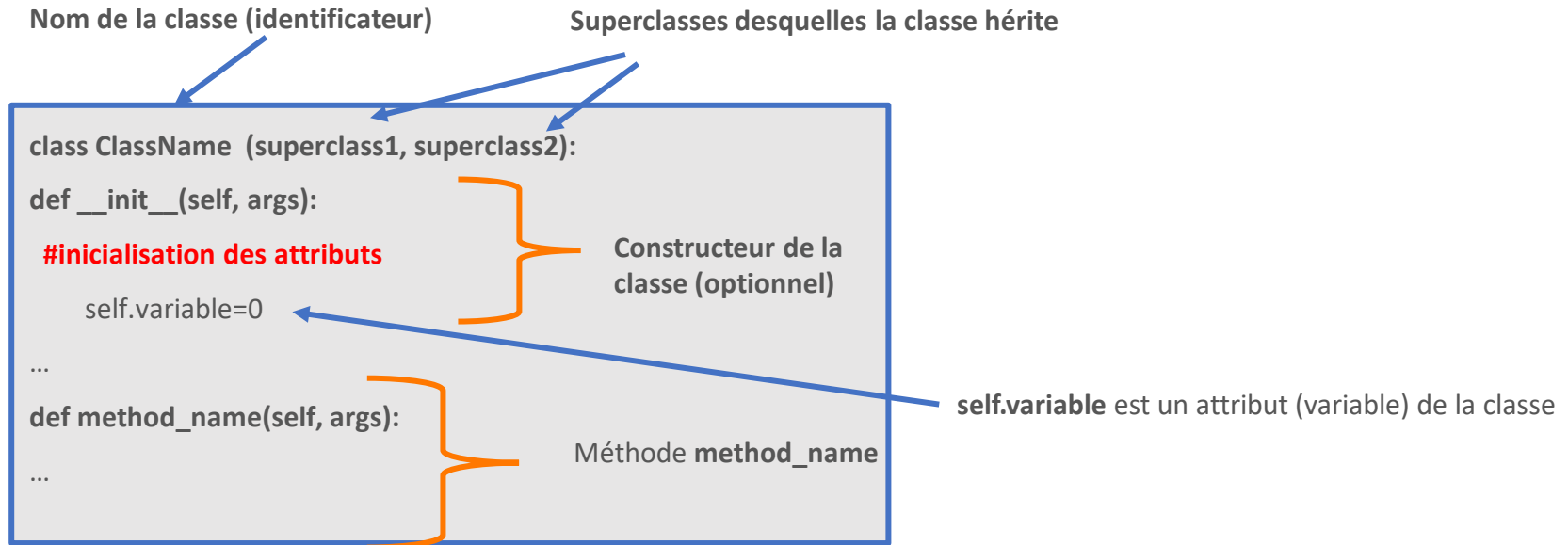


# 01 - Coder une solution orientée objet

## Codage d'une classe

### Présentation d'une classe

- Définition de classe en Python



- Par convention le nom d'une classe commence par une majuscule
- Le mot-clé **pass** permet de préciser que le corps de la classe ne contient rien

```
class classe_vide:  
    pass
```

# 01 - Coder une solution orientée objet

## Codage d'une classe



### Méthodes

- Les méthodes sont des fonctions qui sont associées de manière explicite à une classe. Elles ont comme particularité un accès privilégié aux données de la classe elle-même.
- Les méthodes sont des fonctions pour lesquelles la liste des paramètres contient obligatoirement un paramètre explicite (self) qui est l'instance de la classe à laquelle cette méthode est associée.
- Ce paramètre est le moyen d'accéder aux données de la classe.
- Déclaration d'une méthode

### Déclaration d'une méthode

```
class nom_classe :  
    def nom_methode(self, param_1, ..., param_n):  
        # corps de la méthode...
```

Self désigne l'instance courante de la classe

### Appel d'une méthode

```
cl = nom_classe() # variable de type nom_classe  
t = cl.nom_methode (valeur_1, ..., valeur_n)
```

## 01 - Coder une solution orientée objet

### Codage d'une classe

#### Attributs

- Les attributs sont des variables qui sont associées de manière explicite à une classe.
- Les attributs de la classe se comportent comme des variables globales pour toutes les méthodes de cette classe.

#### Déclaration d'un attribut

```
class nom_classe :  
    def nom_methode (self, param_1, ..., param_n) :  
        self.nom_attribut = param_1
```

Déclaration d'un attribut en précédant son nom par **self**

## 01 - Coder une solution orientée objet

### Codage d'une classe



### Méthodes et attributs

Exemple :

```
class exemple_classe:

    def methodel(self, n):
        """simule la génération d'un nombre aléatoire
        compris entre 0 et n-1 inclus"""
        self.rnd = 42 # déclaration de l'attribut rnd à l'intérieur de la méthode,
                      # doit être précédé du mot-clé self
        self.rnd = 397204094 * self.rnd % 2147483647
        return self.rnd % n

nb = exemple_classe() # création d'une instance de la classe
x=nb.methodel(100) # appel de la méthode
print(x) # affiche 19
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Constructeur et instantiation

- Le constructeur d'une classe se présente comme une méthode et suit la même syntaxe à ceci près que son nom est imposé : `__init__`
- Hormis le premier paramètre, invariablement `self`, il n'existe pas de contrainte concernant la liste des paramètres excepté que le constructeur ne doit pas retourner de résultat.

#### Déclaration d'un constructeur

```
class nom_classe :  
    def __init__(self, param_1, ..., param_n):  
        # code du constructeur
```

Déclaration d'un constructeur à n paramètres

#### Appel d'un constructeur

```
x = nom_classe (valeur_1,...,valeur_n)
```

#### Exemple :

```
class classe1:  
    def __init__(self):  
        # pas de paramètre supplémentaire  
        print("constructeur de la classe classe1")  
        self.n = 1 # ajout de l'attribut n  
  
x = classe1() # affiche constructeur de la classe classe1  
print(x.n) | # affiche 1
```

```
class classe2:  
    def __init__(self, a, b):  
        # deux paramètres supplémentaires  
        print("constructeur de la classe classe2")  
        self.n = (a + b) / 2 # ajout de l'attribut n  
  
x = classe2(5, 9) # affiche constructeur de la classe classe2  
print(x.n) # affiche 7
```

# 01 - Coder une solution orientée objet

## Codage d'une classe



### Constructeur et instanciation

- Par défaut, toute classe en Python a un constructeur par défaut sans paramètre
- Le constructeur par défaut n'existe plus si la classe présente un constructeur à paramètre
- Contrairement à d'autres langues, la classe de Python n'a qu'un seul constructeur. Cependant, Python permet à un paramètre de prendre une valeur par défaut.
- Remarque : Tous les paramètres requis doivent précéder tous les paramètres qui ont des valeurs par défaut.

Exemple :

```
class Person :  
  
    # Les paramètres d'âge (age) et de genre (gender) ont une valeur par défaut.  
    def __init__ (self, name, age = "1", gender = "Male" ):  
        self.name = name  
        self.age = age  
        self.gender= gender  
  
    def showInfo(self):  
        print (self.name + " " +self.age + " " +self.gender)
```

```
if __name__ == '__main__':  
    aimee = Person("Aimee", "21", "Female")  
    aimee.showInfo() #affiche Aimee 21 Female  
  
    alice = Person( "Alice" )# age, gender par défaut.  
    alice.showInfo() #affiche Alice 1 Male  
    |  
    tran = Person("Tran", "37")# gender par défaut.  
    tran.showInfo()#affiche Tran 37 Male
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Instanciation d'une classe

#### Affichage de l'instance

```
class Person :  
    # Les paramètres d'âge (age) et de genre (gender) ont une valeur par défaut.  
    def __init__(self, name, age=1, gender = "Male" ):  
        self.name = name  
        self.age = age  
        self.gender= gender  
  
if __name__ == "__main__":  
    aimee = Person("Aimee", 21, "Female")  
    print(aimee) # affiche: <__main__.Person object at 0x0000024FE3FE7F40>
```

- Il est également possible de vérifier qu'une instance est bien issue d'une classe donnée avec la fonction isinstance()

```
class Person :  
    # Les paramètres d'âge (age) et de genre (gender) ont une valeur par défaut.  
    def __init__(self, name, age=1, gender = "Male" ):  
        self.name = name  
        self.age = age  
        self.gender= gender  
  
if __name__ == "__main__":  
    aimee = Person("Aimee", 21, "Female")  
    print(isinstance(aimee, Person)) # affiche: True
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Destructeur

- Les destructeurs sont appelés lorsqu'un objet est détruit.
- En Python, les destructeurs ne sont pas aussi nécessaires qu'en C++ (par exemple), car Python dispose d'un ramasse-miettes qui gère automatiquement la gestion de la mémoire.
- La méthode `__del__()` est une méthode appelée destructeur en Python. Il est appelé lorsque toutes les références à l'objet ont été supprimées, c'est-à-dire lorsqu'un objet est nettoyé.

### Syntaxe de destructeur

```
def __del__(self):  
    # actions
```

### Exemple :

```
class Person :  
  
    def __init__ (self, name, age = 1, gender = "Male" ):  
        self.name = name  
        self.age = age  
        self.gender= gender  
  
    def showInfo(self):  
        print (self.name + " " + self.age + " " + self.gender)  
  
    def __del__(self):  
        print("je suis le destructeur")
```

```
if __name__ == '__main__':  
    aimee = Person("Aimee", "21", "Female")  
    aimee.showInfo()  
  
    del aimee #affiche je suis le destructeur  
    print(aimee) #NameError: name ' aimee ' is not defined
```



## 01 - Coder une solution orientée objet

### Codage d'une classe

#### Apport du langage Python

L'attribut spécial `__doc__`

```
class Voiture():  
    """  
    Classe voiture avec option couleur  
    """  
    def __init__(self):  
        self.couleur="rouge"  
    def get_couleur(self):  
        print ("Recuperation de la couleur")  
        return self.couleur
```

Dans un code destiné à être réutilisé, il faut absolument définir dans la documentation ce que fait la classe et ses entrées et sorties

```
if __name__ == '__main__':  
    ma_voiture = Voiture()  
    print(ma_voiture.__doc__)  
    #affiche '\n Classe voiture avec option couleur\n '
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Apport du langage Python

#### L'attribut spécial `__dict__`

- Cet attribut spécial donne les valeurs des attributs de l'instance :

```
if __name__ == '__main__':  
    ma_voiture = Voiture()  
    print(ma_voiture.__dict__)  
    # affiche {'couleur': 'rouge'}
```

#### Fonction `dir`

- La fonction `dir` donne un aperçu des méthodes de l'objet :

```
if __name__ == '__main__':  
    ma_voiture = Voiture()  
    print(dir(ma_voiture))  
    # affiche ['__doc__', '__init__', '__module__', 'get_couleur', 'set_couleur']  
    |
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Apport du langage Python

#### Affichage d'un objet

- `__str__` qui permet de redéfinir l'affichage d'un objet lors d'un appel à l'instruction `print`.

```
class Voiture():  
    """  
    Classe voiture avec option couleur  
    """  
    def __init__(self):  
        self.couleur="rouge"  
    def get_couleur(self):  
        print ("Recuperation de la couleur")  
        return self.couleur  
  
    def __str__(self):  
        return ("la couleur de la voiture est:" + self.couleur)
```

```
if __name__ == '__main__':  
    ma_voiture = Voiture()  
    print(ma_voiture) #affiche la couleur de la voiture est:rouge
```

# 01 - Coder une solution orientée objet

## Codage d'une classe



### Apport du langage Python

#### Ajout d'un attribut d'instance

- L'ajout d'un attribut depuis l'extérieur de la classe avec une syntaxe **instance.nouvel\_attribut = valeur**, créera ce nouvel attribut uniquement pour cette instance :

```
if __name__ == "__main__":
    ma_voiture = Voiture()
    print("Attributs de ma voiture")
    print(ma_voiture.__dict__)
    sa_voiture = Voiture()
    sa_voiture.matricule=1235
    print("Attributs de sa voiture")
    print(sa_voiture.__dict__)
#affiche
#Attributs de ma voiture
#{'couleur': 'rouge'}
#Attributs de sa voiture
#{'couleur': 'rouge', 'matricule': 1235}
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Attribut de classe

- Les **attributs de classe** sont différents des attributs d'instance.
- Un attribut dont la valeur est la même pour toutes les instances d'une classe est appelé un attribut de classe. Par conséquent, la valeur de l'attribut de classe est partagée par tous les objets.
- Les attributs de classe sont définis au niveau de la classe plutôt qu'à l'intérieur de la méthode `__init__()`.
- Contrairement aux attributs d'instance, les attributs de classe sont accessibles à l'aide du nom de la classe ou le nom d'instance.

```
class Citron:  
    forme=""  
    def donnerForme(self, params):  
        Citron.forme = params
```

forme est un attribut statique

Exemple :

```
class Fruit:  
    nom = 'fruit'  
    def __init__(self, couleur, poids_g):  
        print("J'aime manger des fruits")  
        self.couleur = couleur  
        self.poids_g = poids_g  
if __name__ == '__main__':  
    pomme = Fruit("verte", 100)  
    print(pomme.nom) #affiche fruit  
    print(Fruit.nom) #affiche fruit
```

## 01 - Coder une solution orientée objet

### Codage d'une classe

#### Attribut de classe

- La modification de l'attribut de classe à l'aide du nom de la classe affectera toutes les instances d'une classe
- La modification de l'attribut de classe à l'aide de l'instance n'affectera pas la classe et les autres instances. Cela n'affectera que l'instance modifiée.

Exemple :

```
class Fruit:
    nom = 'fruit'
    def __init__(self, couleur, poids_g):
        print("J'aime manger des fruits")
        self.couleur = couleur
        self.poids_g = poids_g
if __name__ == '__main__':
    pomme = Fruit("verte", 100)
    banane = Fruit("jaune", 100)
    print(banane.nom) #affiche fruit
    pomme.nom="fruit d'été"
    print(pomme.nom) # affiche fruit d'été
    print(Fruit.nom)#affiche fruit
    print(banane.nom)#affiche fruit
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Méthodes statiques

**Rappel :** Les méthodes statiques sont des méthodes qui peuvent être appelées même si aucune instance de la classe où elles sont définies n'a été créée

- Comme toutes les méthodes présentées jusqu'à présent, une méthode inclut le paramètre `self` qui correspond à l'instance pour laquelle elle est appelée
- **Une méthode statique** ne nécessite pas qu'une instance soit créée pour être appelée. C'est donc une méthode n'ayant pas besoin du paramètre `self`.
- La déclaration d'une méthode statique se fait avec le décorateur `@staticmethod`

Nom\_méthode est une méthode statique

```
class nom_class :  
    @staticmethod  
    def nom_methode(params, ...) :  
        # corps de la méthode
```

Exemple :

```
class essai_class:  
    @staticmethod  
    def methode():  
        print("méthode statique")  
  
if __name__ == '__main__':  
    essai_class.methode()  
#L'exécution de ce code affichera :  
#méthode statique
```

La méthode statique **methode** ne nécessite aucune création d'instance pour être appelée

## 01 - Coder une solution orientée objet

### Codage d'une classe



#### Ajout de méthodes

- Il est possible de déclarer une fonction statique à l'extérieur d'une classe puis de l'ajouter en tant que méthode statique à cette classe.
- Le programme suivant déclare une fonction **methode** puis indique à la classe `essai_class` que la fonction est aussi une méthode statique de sa classe à l'aide de `staticmethod`.

Exemple :

```
def methode():  
    print("méthode statique")  
  
class essai_class:  
    pass  
  
if __name__ == '__main__':  
    essai_class.methode = staticmethod(methode)  
    essai_class.methode()  
    #L'exécution de ce code affichera :  
    #méthode statique
```



# 01 - Coder une solution orientée objet

## Codage d'une classe

### Modificateur d'accès public

- Les membres d'une classe déclarés publics sont facilement accessibles depuis n'importe quelle partie du programme.
- Tous les membres de données et les fonctions membres d'une classe sont publics par défaut.

```
class Geek:  
  
    def __init__(self, name, age):  
  
        self.geekName = name  
        self.geekAge = age |  
  
    def displayAge(self):  
  
        print("Age: ", self.geekAge)
```

geekName et geekAge  
sont des attributs  
publics

Méthode publique

```
if __name__ == '__main__':  
    obj = Geek("R2J", 20)  
    print("Name: ", obj.geekName)  
    obj.displayAge() #affiche:  
    #Nom: R2J  
    # Âge: 20 ans
```

# 01 - Coder une solution orientée objet

## Codage d'une classe

### Modificateur d'accès protégé

- Les membres d'une classe déclarés protégés ne sont accessibles qu'à une classe qui en dérive.
- Les données membres d'une classe sont déclarées protégées en ajoutant un seul symbole de soulignement «\_» avant le membre de données de cette classe.

Exemple :

`_name`, `_roll`, et `_branch`  
sont des attributs protégés

`_displayRollAndBranch`  
est une méthode protégée

```
class Student:
    def __init__(self, name, roll, branch):
        self._name = name
        self._roll = roll
        self._branch = branch

    def _displayRollAndBranch(self):
        print("Roll: ", self._roll)
        print("Branch: ", self._branch)
```

```
if __name__ == '__main__':
    obj = Student("R2J", 1706256, "Information Technology")
    obj._displayRollAndBranch()
#affiche:
# Roll: 1706256
# Branch: Technologie de l'information
```

- Si on considère que la classe **Geek** est dérivée de la classe **Student** alors :
  - `name`, `_roll` et `_branch` sont des membres de données protégées et la méthode `_displayRollAndBranch()` est une méthode protégée de la super classe **Student**.
  - La méthode `displayDetails()` est une fonction membre publique de la classe **Geek** qui est dérivée de la classe **Student**, la méthode `displayDetails()` de la classe **Geek** accède aux données membres protégées de la classe **Student**.

# 01 - Coder une solution orientée objet

## Codage d'une classe



### Modificateur d'accès privé

- Les membres d'une classe qui sont déclarés privés sont accessibles uniquement dans la classe, le modificateur d'accès privé est le modificateur d'accès le plus sécurisé.
- Les données membres d'une classe sont déclarées privées en ajoutant un double trait de soulignement «\_\_» avant le membre de données de cette classe.

Exemple :

```
class Geek:
    def __init__(self, name, roll, branch):
        self.__name = name
        self.__roll = roll
        self.__branch = branch

    def __displayDetails(self):
        print("Name: ", self.__name)
        print("Roll: ", self.__roll)
        print("Branch: ", self.__branch)

    def accessPrivateFunction(self):
        self.__displayDetails()
```

```
if __name__ == '__main__':
    obj = Geek("R2J", 1706256, "Information Technology")
    obj.accessPrivateFunction()
#affiche:
# Nom: R2J
# Rouleau: 1706256
# Direction: Technologie de l'information
```

- \_\_name, \_\_roll et \_\_branch sont des membres privés, \_\_displayDetails() est une fonction membre privée (elle ne peut être accédée que dans la classe) et accessPrivateFunction() est une fonction membre publique de la classe Geek et accessible de n'importe où dans le programme.
- La accessPrivateFunction() est une méthode qui accède aux membres privés de la classe Geek.

# CHAPITRE 1

## Coder une solution orientée objet

1. Création d'un package
2. Codage d'une classe
- 3. Intégration des concepts POO**



### Getter et Setter en Python

- Les Getters et Setters en Python sont souvent utilisés pour éviter l'accès direct à un champ de classe, c'est-à-dire que les variables privées ne peuvent pas être accessibles directement ou modifiées par un utilisateur externe.

#### Utilisation de la fonction normale pour obtenir le comportement des Getters et des Setters

- Pour obtenir la propriété Getters et Setters, si nous définissons les méthodes get() et set(), cela ne reflètera aucune implémentation spéciale.
- Exemple :

```
class Geek:
    def __init__(self, age = 0):
        self._age = age
    def get_age(self):
        return self._age
    def set_age(self, x):
        self._age = x
```

```
if __name__ == '__main__':
    raj = Geek()
    raj.set_age(21)
    print(raj.get_age())
    print(raj._age)

#affiche:
#21
#21
```

### Getter et Setter avec property()

- En Python `property()` est une fonction intégrée qui crée et renvoie un objet de propriété.
- Un objet de propriété a trois méthodes, `getter()`, `setter()` et `delete()`.
- La fonction `property()` en Python a trois arguments **property (fget, fset, fdel, doc)**
  - **fget** est une fonction pour récupérer une valeur d'attribut
  - **fset** est une fonction pour définir une valeur d'attribut
  - **fdel** est une fonction pour supprimer une valeur d'attribut
  - **doc** est une chaîne contenant la documentation (docstring à voir ultérieurement) de l'attribut

```
class Geeks:
    def __init__(self):
        self._age = 0
    def get_age(self):
        print("getter method called")
        return self._age
    def set_age(self, a):
        print("setter method called")
        self._age = a
    def del_age(self):
        del self._age
    age = property(get_age, set_age, del_age)
```

appel

```
if __name__ == '__main__':
    mark = Geeks()
    mark.age = 10
    print(mark.age)

#affiche:
#setter method called
#setter method called
#10
```

# 01 - Coder une solution orientée objet

## Intégration des concepts POO

### Utilisation de @property

- **@property** est un décorateur qui évite d'utiliser des fonctions setter et getter explicites

Exemple :

```
class Etudiant:

    def __init__(self, nom_famille, prenom):
        self.nom_famille = nom_famille
        self.prenom = prenom

    @property
    def nomcomplet(self):
        return self.prenom + ' ' + self.nom_famille
```

```
if __name__ == '__main__':
    E1 = Etudiant('Richard', 'Marie')
    print('Le nom complet de E1 est :', E1.nomcomplet)

    # nous allons modifier le nom de l'objet E1
    print("\n La modification : ")
    E1.nom_famille = 'Bernard'
    print('Le nom complet de E1 est :', E1.nomcomplet)
```

nomcomplet() fonctionne comme un getter à cause du décorateur @property.  
Appel de nomCompleet au lieu de nomcomplet()

le décorateur @property est utilisé avec la fonction nomcomplet()

# 01 - Coder une solution orientée objet

## Intégration des concepts POO

### Héritage en Python

- **Rappel :** l'héritage est défini comme la capacité d'une classe à dériver ou à hériter des propriétés d'une autre classe et à l'utiliser chaque fois que nécessaire.

#### Syntaxe en Python

**Class Nom\_classe(Nom\_SuperClass)**

#### Exemple :

```
class Child:
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

    def isStudent(self):
        return False

class Student(Child):
    def isStudent(self):
        return True
```

```
if __name__ == '__main__':
    std = Child("Ram")
    print(std._getName(), std.isStudent())
    std = Student("Shivam")
    print(std._getName(), std.isStudent())
#affiche:
#Ram False
#Shivam True
```

Classe Student hérite de Child



## 01 - Coder une solution orientée objet

### Intégration des concepts POO

### Héritage unique

- **Rappel** : L'héritage unique permet à une classe dérivée d'hériter des propriétés d'une seule classe parente, permettant ainsi la réutilisation du code et l'ajout de nouvelles fonctionnalités au code existant.

Exemple :

```
class Parent:
    def func1(self):
        print("This function is in parent class.")

class Child(Parent):
    def func2(self):
        print("This function is in child class.")
```

```
if __name__ == '__main__':
    object = Child()
    object.func1()
    object.func2()

#affiche:
# Cette fonction est dans la classe parente.
# Cette fonction est dans la classe enfant.
```

### Héritage multiple

- **Rappel :** Lorsqu'une classe peut être dérivée de plusieurs classes de base, ce type d'héritage est appelé héritage multiple. Dans l'héritage multiple, toutes les fonctionnalités des classes de base sont héritées dans la classe dérivée.

Exemple :

```
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
```

```
if __name__ == '__main__':
    s1 = Son()
    s1.fathername = "RAM"
    s1.mothername = "SITA"
    s1.parents()
#affiche:
#Father: RAM
#Mother: SITA
```

### Héritage en cascade

- Dans l'héritage en cascade, les fonctionnalités de la classe de base et de la classe dérivée sont ensuite héritées dans la nouvelle classe dérivée

```
class Grandfather:
    def Grandfather(self):
        print(self.grandfathername)

class Father(Grandfather):
    def Father(fathername):
        print(self.fathername)

class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname
        self.fathername = fathername
        self.grandfathername = grandfathername

    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)
```

```
if __name__ == '__main__':
    sl = Son('Prince', 'Rampal', 'Lal mani')
    sl.print_name()
#affiche:
#Grandfather name : Lal mani
# Father name :Rampal
# Son name: Prince
```

### Chainage de constructeurs

- Le chaînage de constructeurs est une technique **d'appel d'un constructeur de la classe mère à partir d'un constructeur de la classe fille**

Exemple :

```
class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        Grandfather.__init__(self, grandfathername)
```

Appel du constructeur de la classe mère pour initialiser l'attribut hérité grandfathername

## 01 - Coder une solution orientée objet

### Intégration des concepts POO

### Chainage de constructeurs

```
class Son(Father):  
    def __init__(self, sonname, fathername, grandfathername):  
        self.sonname = sonname  
        Father.__init__(self, fathername, grandfathername)  
  
    def print_name(self):  
        print('Grandfather name :', self.grandfathername)  
        print("Father name :", self.fathername)  
        print("Son name :", self.sonname)
```

Appel du constructeur de la classe mère pour initialiser les attributs hérités fathername et grandfathername

```
if __name__ == '__main__':  
    s1 = Son('Prince', 'Rampal', 'Lal mani')  
    print(s1.grandfathername)  
    s1.print_name()  
    #affiche:  
    #Lal mani  
    #Grandfather name : Lal mani  
    # Father name :Rampal  
    # Son name: Prince
```

## 01 - Coder une solution orientée objet

### Intégration des concepts POO



### Surcharge des opérateurs en Python

- La surcharge d'opérateurs vous permet de redéfinir la signification d'opérateur en fonction de votre classe.
- Cette fonctionnalité en Python, qui permet à un même opérateur d'avoir une signification différente en fonction du contexte.

#### Opérateurs arithmétiques

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

if __name__ == "__main__":
    p1 = Point(2, 4)
    p2 = Point(5, 1)
    print(p1+p2) #affiche: TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

**Erreur!!!**

**Python ne savait pas comment ajouter deux objets  
Point ensemble.**

### Surcharge des opérateurs en Python

- Pour surcharger l'opérateur +, nous devons implémenter la fonction `__add__()` dans la classe

Exemple :

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x, self.y)

    def __add__(self, p):
        a = self.x + p.x
        b = self.y + p.y
        return Point(a, b)

if __name__ == "__main__":
    p1 = Point(2, 4)
    p2 = Point(5, 1)
    p3 = p1+p2
    print(p3) #affiche (7,5)
```

## 01 - Coder une solution orientée objet

### Intégration des concepts POO

### Fonctions spéciales de surcharge de l'opérateur en Python

Opérateur	Expression	Interprétation Python
Addition	<code>p1 + p2</code>	<code>p1.__add__(p2)</code>
Soustraction	<code>p1 - p2</code>	<code>p1.__sub__(p2)</code>
Multiplication	<code>p1 * p2</code>	<code>p1.__mul__(p2)</code>
Puissance	<code>p1 ** p2</code>	<code>p1.__pow__(p2)</code>
Division	<code>p1 / p2</code>	<code>p1.__truediv__(p2)</code>
Division entière	<code>p1 // p2</code>	<code>p1.__floordiv__(p2)</code>
le reste (modulo)	<code>p1 % p2</code>	<code>p1.__mod__(p2)</code>
Décalage binaire gauche	<code>p1 &lt;&lt; p2</code>	<code>p1.__lshift__(p2)</code>
Décalage binaire droite	<code>p1 &gt;&gt; p2</code>	<code>p1.__rshift__(p2)</code>
ET binaire	<code>p1 &amp; p2</code>	<code>p1.__and__(p2)</code>
OU binaire	<code>p1   p2</code>	<code>p1.__or__(p2)</code>
XOR	<code>p1 ^ p2</code>	<code>p1.__xor__(p2)</code>
NON binaire	<code>~p1</code>	<code>p1.__invert__()</code>



### Surcharge des opérateurs en Python

#### Opérateurs de comparaison

- En Python, il est possible de surcharger les opérateurs de comparaison.

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __lt__(self, p):
        m_self = math.sqrt((self.x ** 2) + (self.y ** 2))
        m2_p = math.sqrt((p.x ** 2) + (p.y ** 2))
        return m_self < m2_p

if __name__ == "__main__":
    p1 = Point(2, 4)
    p2 = Point(5, 1)
    if p1 < p2:
        print("p2 est plus loin que p1") # affiche: p2 est plus loin que p1
```

Surcharge de l'opérateur inférieur

### Fonctions spéciales de surcharge de l'opérateur en Python

Opérateur	Expression	Interprétation Python
Inférieur à	<code>p1 &lt; p2</code>	<code>p1._lt_(p2)</code>
Inférieur ou égal	<code>p1 &lt;= p2</code>	<code>p1._le_(p2)</code>
Egal	<code>p1 == p2</code>	<code>p1._eq_(p2)</code>
différent	<code>p1 != p2</code>	<code>p1._ne_(p2)</code>
Supérieur à	<code>p1 &gt; p2</code>	<code>p1._gt_(p2)</code>
Supérieur ou égal	<code>p1 &gt;= p2</code>	<code>p1._ge_(p2)</code>

# 01 - Coder une solution orientée objet

## Intégration des concepts POO



### Polymorphisme et héritage

- En Python, le polymorphisme permet de définir des méthodes dans la classe enfant qui ont le même nom que les méthodes de la classe parent.
- En héritage, la classe enfant hérite des méthodes de la classe parent.
- Il est possible de modifier une méthode dans une classe enfant dont elle a hérité de la classe parent (redéfinition de la méthode).

Exemple :

```
if __name__ == '__main__':  
    obj_bird = Bird()  
    obj_spr = sparrow()  
    obj_ost = ostrich()  
    obj_bird.intro() #affiche: There are many types of birds  
    obj_bird.flight() #affiche: Most of the birds can fly but some cannot  
    obj_spr.intro() #affiche: There are many types of birds  
    obj_spr.flight() #affiche: Sparrows can fly  
    obj_ost.intro() #affiche: There are many types of birds  
    obj_ost.flight() #affiche: Ostriches cannot fly
```

```
class Bird:  
    def intro(self):  
        print("There are many types of birds.")  
  
    def flight(self):  
        print("Most of the birds can fly but some cannot.")  
  
class sparrow(Bird):  
    def flight(self):  
        print("Sparrows can fly.")  
  
class ostrich(Bird):  
    def flight(self):  
        print("Ostriches cannot fly.")
```

## 01 - Coder une solution orientée objet

### Intégration des concepts POO



#### Classe abstraite

- Rappel: Les classes abstraites sont des classes qui ne peuvent pas être instanciées, elles contiennent une ou plusieurs méthodes abstraites (méthodes sans code)
- Une classe abstraite nécessite des sous-classes qui fournissent des implémentations pour les méthodes abstraites sinon ces sous-classes sont déclarées abstraites
- Une classe abstraite hérite de la **classe abstraite de base – ABC**
- L'objectif principal de la classe de base abstraite est de fournir un moyen standardisé de tester si un objet adhère à une spécification donnée
- Pour définir une méthode abstraite dans la classe abstraite, on utilise un décorateur **@abstractmethod**

Exemple :

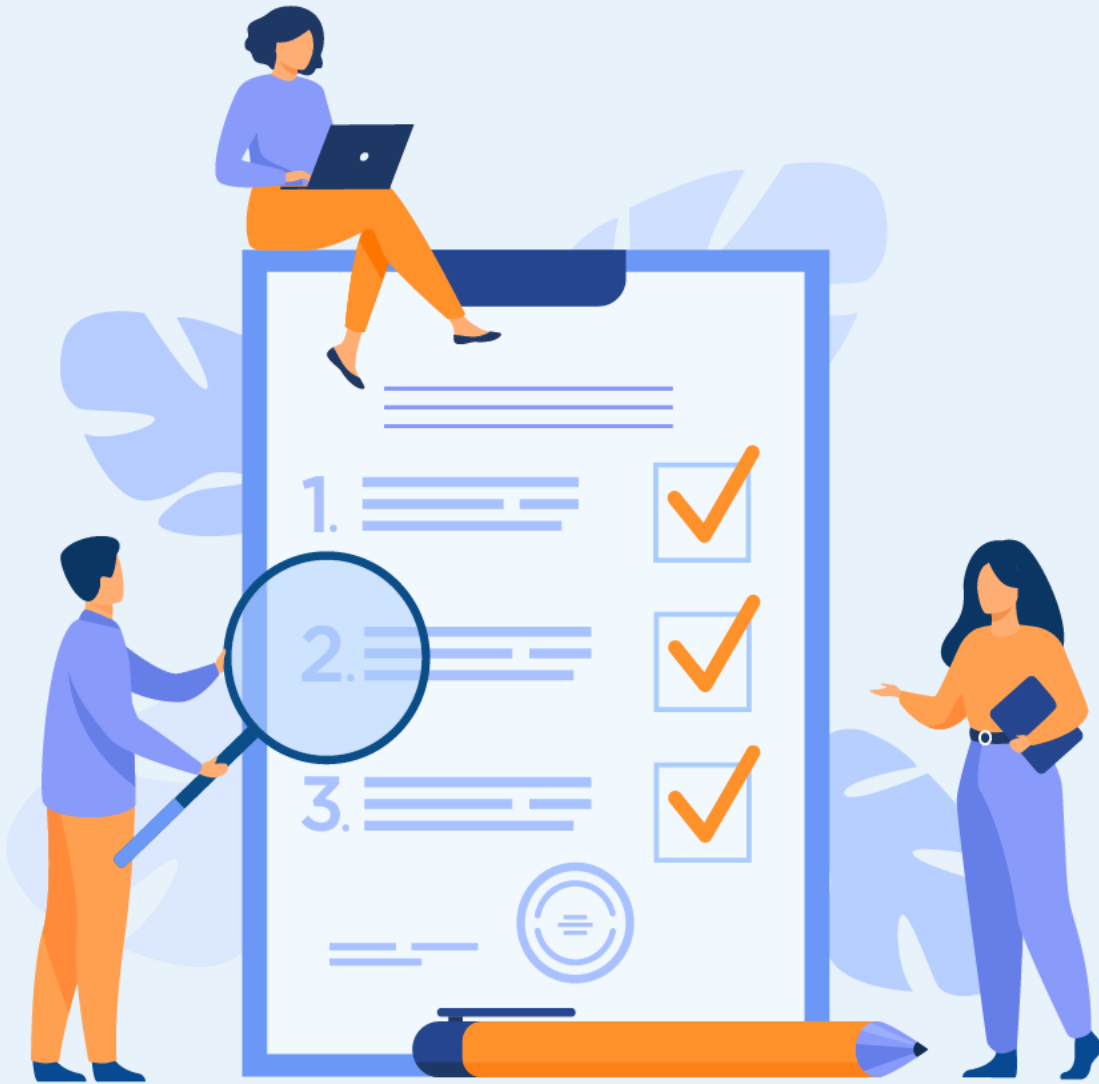
```
from abc import ABC, abstractmethod # abc est un module python intégré, nous importons ABC et abstractmethod
class Animal(ABC): # hériter de ABC(Abstract base class)
    @abstractmethod # un décorateur pour définir une méthode abstraite
    def nourrir(self):
        pass
```

## CHAPITRE 2

### Manipuler les données

Ce que vous allez apprendre dans ce chapitre :

- Manipuler les structures de données de Python à savoir les collections et les listes
- Manipuler les fichiers de données en Python



10 heures

# CHAPITRE 2

## Manipuler les données

1. Liste
2. Collections
3. Fichiers



## 02 - Manipuler les données

### Listes

### Listes

- Une liste est une collection qui est commandée et modifiable.
  - En Python, les liste sont écrites entre crochets.

```
>>> thislist= ["apple","banana","cherry"]
>>> print(thislist)
['apple', 'banana', 'cherry']
>>> print(thislist[1])
banana
```

```
>>> thislist= ["apple","banana","cherry"]
>>> print(thislist)
['apple', 'banana', 'cherry']
>>> print(thislist[1])
banana
>>> thislist= ["apple","banana","cherry"]

>>> print(thislist[-1]) #afficher la valeur de la position -1 (cycle)

cherry
>>> thislist=["apple","banana","cherry","orange","kiwi","melon","mango"]

>>> print(thislist[2:5])#afficher les valeurs de la position 2 jusqu'à 5 (5 non inclus)
['cherry', 'orange', 'kiwi']
```

## 02 - Manipuler les données

### Listes

### Listes

- Modification de la valeur d'un élément du tableau :

```
>>> thislist= ["apple","banana","cherry"]  
>>> thislist[1] ="blackcurrant" # modifier la valeur de la 1ere position  
>>> print(thislist)  
['apple', 'blackcurrant', 'cherry']
```

- Parcoure une liste :

```
>>> for x in thislist:  
    print(x)  
  
apple  
banana  
cherry
```

- Recherche d'un élément dans une liste :

```
>>> thislist= ["apple", "banana", "cherry"]  
>>> if "apple" in thislist:  
    print("Yes, 'apple' is in the fruits list")  
  
Yes, 'apple' is in the fruits list
```



## 02 - Manipuler les données

### Listes

### Listes

- Fonction len(): Longueur d'une liste (fonction len()) :

```
>>> thislist= ["apple", "banana", "cherry"]  
>>> print(len(thislist))  
3
```

- Fonction append(): Ajout d'un élément à la liste :

```
>>> thislist= ["apple", "banana", "cherry"]  
  
>>> thislist.append("orange")  
  
>>> print(thislist)  
['apple', 'banana', 'cherry', 'orange']
```

- Fonction insert(): Ajout d'un élément à une position de la liste :

```
>>> thislist= ["apple", "banana", "cherry"]  
>>> thislist.insert(1, "orange")  
>>> print(thislist)  
['apple', 'orange', 'banana', 'cherry']
```

## 02 - Manipuler les données

### Listes

### Listes

- Fonction `pop()`: Suppression du dernier élément de la liste :

```
>>> thislist= ["apple", "banana", "cherry"]
>>> thislist.pop()
'cherry'
>>> print(thislist)
['apple', 'banana']
```

- Fonction `del()`: Suppression d'un élément de la liste :

```
>>> thislist= ["apple", "banana", "cherry"]
>>> del(thislist[0])
>>> print(thislist)
['banana', 'cherry']
```

- Fonction `extend()`: Fusion de deux listes :

```
>>> list1 = ["a", "b" , "c"]

>>> list2 = [1,2,3]
>>> list1.extend(list2)

>>> print(list1)
['a', 'b', 'c', 1, 2, 3]
```

### Listes

- Autres méthodes :

Method	Description
<code>append()</code>	Adds an element at the end of the list
<code>clear()</code>	Removes all the elements from the list
<code>copy()</code>	Returns a copy of the list
<code>count()</code>	Returns the number of elements with the specified value
<code>extend()</code>	Add the elements of a list (or any iterable), to the end of the current list
<code>index()</code>	Returns the index of the first element with the specified value
<code>insert()</code>	Adds an element at the specified position
<code>pop()</code>	Removes the element at the specified position
<code>remove()</code>	Removes the item with the specified value
<code>reverse()</code>	Reverses the order of the list
<code>sort()</code>	Sorts the list

# CHAPITRE 2

## Manipuler les données

1. Liste
2. **Collections**
3. Fichiers



## 02 - Manipuler les données

### Collections

- Le module collections contient des conteneurs de données spécialisés
- La liste des conteneurs est la suivante

Conteneur	Utilité
namedtuple()	une fonction permettant de créer une sous-classe de tuple avec des champs nommés
deque	un conteneur ressemblant a une liste mais avec ajout et suppression rapide a chacun des bouts
ChainMap	permet de linker entre eux plusieurs mappings ensemble pour les gérer comme un tout
Counter	Permet de compter les occurrences d'objets hachable
OrderedDict	une sous classe de dictionnaire permettant de savoir l'ordre des entrées
defaultdict	une sous classe de dictionnaire permettant de spécifier une valeur par défaut dans le constructeur

### Collections - namedtuple

- un tuple est une collection immuable de données souvent hétérogène.

```
t = (11, 22)
print(t)
print(t[0])
print(t[1])
```

- La classe **namedtuple** du module `collections` permet d'ajouter des noms explicites à chaque élément d'un tuple pour rendre ces significations claires dans un programme Python

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22) # instantiation par position ou en utilisant le nom du champs
print(p[0] + p[1])  # affiche 33
print(p.x + p.y)    # affiche 33
print(p)             # Point(x=11, y=22)
```

les champs sont accessibles par nom

indexable comme les tuples de base (11, 22)

lisible dans un style nom=valeur

### Collections - namedtuple

- La méthode `_asdict()` permet de convertir une instance en un dictionnaire.
- Appeler `p._asdict()` renvoie un dictionnaire mettant en correspondance les noms de chacun des deux champs de `p` avec leurs valeurs correspondantes.

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p._asdict()) # affiche {'x': 11, 'y': 22}
```

- La fonction `_replace(key=args)` permet de retourner une nouvelle instance de notre tuple avec une valeur modifiée.

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(x=11, y=22)
print(p._replace(x=4)) # affiche: Point(x=4, y=22)
```

### Collections - namedtuple

- La fonction `_mytuple._fields` permet de récupérer les noms des champs de notre tuple. Elle est utile si on veut créer un nouveau tuple avec les champs d'un tuple existant

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
print(Point._fields) # retourne les noms de champs affiche: ('x', 'y')
Color = namedtuple('Color', 'red green blue')
Pixel = namedtuple('Pixel', Point._fields + Color._fields) #on créé un nouveau tuple avec les champs de point et de color
print(Pixel(11, 22, 128, 255, 0)) # affiche Pixel(x=11, y=22, red=128, green=255, blue=0)
```



### Collections - deque

- Les listes Python sont une séquence d'éléments ordonnés, mutable ou modifiable.
- Python peut ajouter des listes en temps constant mais l'insertion au début d'une liste peut être plus lente (le temps nécessaire augmente à mesure que la liste s'agrandit).

Exemple :

```
favorite_fish_list = ["Sammy", "Jamie", "Mary"]
favorite_fish_list.insert(0, "Alice")
print(favorite_fish_list) # affiche ['Alice', 'Sammy', 'Jamie', 'Mary']
```

La méthode .insert (index, objet) sur list permet d'insérer "Alice" au début de favorite\_fish\_list.

- La classe deque du module collections est un objet de type liste qui permet d'insérer des éléments au début ou à la fin d'une séquence avec une performance à temps constant (O(1)).
- O(1) performance signifie que le temps nécessaire pour ajouter un élément au début de la liste n'augmentera pas, même si cette liste a des milliers ou des millions d'éléments.

```
from collections import deque
favorite_fish_deque = deque(["Sammy", "Jamie", "Mary"])
favorite_fish_deque.appendleft("Alice")
print(favorite_fish_deque) # affiche: deque(['Alice', 'Sammy', 'Jamie', 'Mary'])
```

### Collections - deque

- Les fonctions **append(x)**, **appendleft(x)**: **append** ajoute une seule valeur du côté droit du deque et **appendleft** du côté gauche

```
from collections import namedtuple
from collections import deque
favorite_fish_deque = deque(["Sammy", "Jamie", "Mary"])
favorite_fish_deque.appendleft("Alice")
favorite_fish_deque.append("Bob")
print(favorite_fish_deque) # affiche deque(['Alice', 'Sammy', 'Jamie', 'Mary', 'Bob'])
```

- Les fonctions **pop()**, **popleft()** et **clear()**: **pop()** et **popleft()** permettent de faire sortir un objet d'un deque et **clear()** le vide.

```
from collections import namedtuple
from collections import deque
favorite_fish_deque = deque(["Alice", "Sammy", "Jamie", "Mary", "Bob"])
print(favorite_fish_deque.pop()) #affiche Bob
print(favorite_fish_deque.popleft())#affiche: Alice
print(favorite_fish_deque.clear())#affiche: None
```

### Collections - ChainMap

- La classe `collections.ChainMap` permet de linker plusieurs mappings pour qu'ils soient gérés comme un seul.
- `class collections.ChainMap(*maps)` cette fonction retourne une nouvelle ChainMap. Si il n'y a pas de maps spécifiés en paramètres la ChainMap sera vide.

Exemple :

```
from collections import ChainMap
x = {'a': 1, 'b': 2}
y = {'b': 10, 'c': 11}
z = ChainMap(y, x)
for k, v in z.items():
    print(k, v)
#affiche: a 1
#      c 11
#      b 10
```

- Dans l'exemple précédent on remarque que la clé b a pris la valeur 10 et pas 2 car y est passé avant x dans le constructeur de ChainMap.

### Collections - Counter

- La classe **collections.Counter** est une sous-classe de dict. qui permet de compter des objets hachable.
- En fait c'est un dictionnaire avec comme clé les éléments et comme valeurs leur nombre.
- `class collections.Counter([iterable-or-mapping])` ceci retourne un Counter. L'argument permet de spécifier ce que l'on veut mettre dedans et qui doit être compté.

Exemple :

```
from collections import Counter
c = Counter()                # compteur vide
c = Counter('gallahad')      #compteur avec un iterable
c = Counter({'red': 4, 'blue': 2}) # un compteur avec un mapping
c = Counter(cats=4, dogs=8)   #un compteur avec key=valeur
|
```

### Collections - Counter

- Si on demande une valeur n'étant pas dans notre liste il retourne 0 et non pas KeyError

```
from collections import Counter
c = Counter(['eggs', 'ham'])
print (c['bacon']) # clé inconnue, affiche : 0
```

- La fonction **elements()**: retourne une liste de tous les éléments du compteur :

```
from collections import Counter
c = Counter(a=4, b=2, c=0, d=-2)
print (sorted(c.elements())) #affiche: ['a', 'a', 'a', 'a', 'b', 'b']
|
```

- La fonction **most\_common([n])**: retourne les n éléments les plus présents dans le compteur

```
from collections import Counter
print (Counter('abracadabra').most_common(3))
|
```

### Collections - Counter

- La fonction **subtract([iterable or mapping])** : permet de soustraire des éléments d'un compteur (mais pas de les supprimer)

```
from collections import Counter
c = Counter(a=4, b=2, c=0, d=-2)
d = Counter(a=1, b=2, c=3, d=4)
c.subtract(d)
print (c) # affiche: Counter({'a': 3, 'b': 0, 'c': -3, 'd': -6})|
```

### Collections - OrderedDict

- Les **collections.OrderedDict** sont comme les dict. mais ils se rappellent l'ordre d'entrée des valeurs. Si on itère dessus les données seront retournées dans l'ordre d'ajout dans notre dict.
- La fonction **popitem(last=True)** : fait sortir une paire clé valeur de notre dictionnaire et si l'argument last est a 'True' alors les paires seront retournées en LIFO sinon ce sera en FIFO.
- La fonction **move\_to\_end(key, last=True)** : permet de déplacer une clé à la fin de notre dictionnaire si last est à True sinon au début de notre dict.

Exemple :

```
from collections import OrderedDict
d = OrderedDict.fromkeys('abcde')
d.move_to_end('b')
print(d.keys())# affiche: odict_keys(['a', 'c', 'd', 'e', 'b'])
d.move_to_end('b', last=False)
print(d.keys())# affiche: odict_keys(['b', 'a', 'c', 'd', 'e'])
print(d.popitem())
```

#### Collections - defaultdict

- **defaultdict** du module de collections qui permet de rassembler les informations dans les dictionnaires de manière rapide et concise.
- **defaultdict** ne soulève jamais une **KeyError**.
- Si une clé n'est pas présente, **defaultdict** se contente d'insérer et de renvoyer une valeur de remplacement à la place.
- **defaultdict** se comporte différemment d'un dictionnaire ordinaire. Au lieu de soulever une **KeyError** sur une clé manquante, defaultdict appelle la valeur de remplacement sans argument pour créer un nouvel objet. Dans l'exemple ci-dessous, `list()` pour créer une liste vide.

```
from collections import defaultdict
my_defaultdict = defaultdict(list)
print(my_defaultdict["missing"]) #affiche []
```



### Collections - defaultdict

Exemple :

```
from collections import defaultdict
def default_message():
    return "Key is not there"

defaultdict_obj = defaultdict(default_message)
defaultdict_obj ["key1"] = "value1"
defaultdict_obj ["key2"] = "value2"
print(defaultdict_obj["key1"]) # affiche: value1
print(defaultdict_obj["key2"])# affiche: value2
print(defaultdict_obj["key3"])# affiche: Key is not there
```

Remplissage d'un defaultdict

Affichage des éléments d'un defaultdict

Si la clé n'existe pas alors appeler la fonction default\_message

Utilisation de la  
fonction lambda

```
from collections import defaultdict
defaultdict_obj = defaultdict(lambda: "Key is missing")
defaultdict_obj ["key1"] = "value1"
defaultdict_obj ["key2"] = "value2"
print(defaultdict_obj["key1"]) # affiche: value1
print(defaultdict_obj["key2"])# affiche: value2
print(defaultdict_obj["key3"])# affiche: Key is not there
```

### Collections - defaultdict

- Lorsque la classe int est fournie comme fonction par défaut, la valeur par défaut retournée est zéro.

```
from collections import defaultdict
defaultdict_obj = defaultdict(int)
defaultdict_obj ["key1"] = "value1"
defaultdict_obj ["key2"] = "value2"
print(defaultdict_obj ["key3"])#affiche: 0
```

- Nous pouvons également utiliser l'objet Set comme objet par défaut

```
from collections import defaultdict
defaultdict_obj = defaultdict(set)
defaultdict_obj ["key1"].add(1)
defaultdict_obj ["key1"].add(11)
defaultdict_obj ["key2"].add(2)
print(defaultdict_obj ["key3"])#affiche: set()
print(defaultdict_obj)#affiche: defaultdict(<class 'set'>, {'key1': {1, 11}, 'key2': {2}, 'key3': set()})
```

# CHAPITRE 2

## Manipuler les données

1. Liste
2. Collections
- 3. Fichiers**



### Utilisation des fichiers

- Python a plusieurs fonctions pour créer, lire, mettre à jour et supprimer des fichiers texte.
- La fonction clé pour travailler avec des fichiers en Python est `open()`.
  - Elle prend deux paramètres; **nom de fichier et mode**.
  - Il existe quatre méthodes (modes) différentes pour ouvrir un fichier:
    - "r" -Lecture -Par défaut. Ouvre un fichier en lecture, erreur si le fichier n'existe pas
    - "a" -Ajouter -Ouvre un fichier à ajouter, crée le fichier s'il n'existe pas
    - "w" -Écrire -Ouvre un fichier pour l'écriture, crée le fichier s'il n'existe pas
    - "x" -Créer -Crée le fichier spécifié, renvoie une erreur si le fichier existe
- Ouvrir et lire un fichier
  - Pour ouvrir le fichier, utilisez la fonction `open()` intégrée
  - La fonction **`open()`** renvoie un objet fichier, qui a une méthode **`read()`** pour lire le contenu du fichier

```
>>>f = open("demofile.txt", "r")  
>>>print(f.read())
```

- Renvoyez les 5 premiers caractères du fichier :

```
>>>f = open("demofile.txt", "r")  
>>>print(f.read(5))
```

### Utilisation des fichiers

- Vous pouvez renvoyer une ligne en utilisant la méthode `readline()`:

```
>>>f = open("demofile.txt", "r")
>>>print(f.readline())
```

- Ecrire dans un fichier
  - Pour écrire dans un fichier existant, vous devez ajouter un paramètre à la fonction **`open()`**:

```
>>>f = open("demofile2.txt", "a")
>>>f.write("Now the file has more content!")
>>>f.close()
>>>f = open("demofile2.txt", "r")#afficher le fichier après modification
>>>print(f.read())

>>>f = open("demofile3.txt", "w")
>>>f.write("Woops! I have deleted the content!")
>>>f.close()
# ouvrez et lisez le fichier après l'ajout:
>>>f = open("demofile3.txt", "r")
>>>print(f.read())

La méthode "w" écrasera tout le fichier.
```

### Utilisation des fichiers

- **Fermer un fichier**
  - Il est recommandé de toujours fermer le fichier lorsque vous en avez terminé.

```
>>>f = open("demofile.txt", "r")  
>>>print(f.readline())  
>>>f.close()
```

- **Supprimer d'un fichier**
  - Pour supprimer un fichier, vous devez importer le module OS et exécuter sa fonction **os.remove ()**:

```
>>>import os  
>>>os.remove("demofile.txt")
```

### Format CSV

- Il existe différents formats standards de stockage de données. Il est recommandé de favoriser ces formats car il existe déjà des modules Python permettant de simplifier leur utilisation.
- Le fichier **Comma-separated values (CSV)** est un format permettant de stocker des tableaux dans un fichier texte. Chaque ligne est représentée par une ligne de texte et chaque colonne est séparée par un **séparateur** (virgule, point-virgule ...).
- Les champs texte peuvent également être délimités par des guillemets.
- Lorsqu'un champ contient lui-même des guillemets, ils sont doublés afin de ne pas être considérés comme début ou fin du champ.
- Si un champ contient un signe pouvant être utilisé comme séparateur de colonne (virgule, point-virgule ...) ou comme séparateur de ligne, les guillemets sont donc obligatoires afin que ce signe ne soit pas confondu avec un séparateur.

- Données sous la forme d'un tableau

Nom	Prenom	Age
Dubois	Marie	29
Duval	Julien "Paul"	47
Jacquet	Bernard	51
Martin	Lucie;Clara	14

- Données sous la forme d'un fichier CSV

```
Nom;Prénom;Age
"Dubois";"Marie";29
"Dupal";"Julien ""Paul""";47
Jacquet;Bernard;51
Martin;"Lucie;Clara";14
```

### Format CSV

- Le module csv de Python permet de simplifier l'utilisation des fichiers CSV
- Lecture d'un fichier CSV
  - Pour lire un fichier CSV, il faut ouvrir un flux de lecture de fichier et ouvrir à partir de ce flux un lecteur CSV.

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.reader(fichier, delimiter=";") # Ouverture du lecteur CSV en lui fournissant le caractère séparateur (ici ";")
for ligne in lecteurCSV:
    print(ligne)
fichier.close()
```

```
['Nom ', 'Prenom', 'Age']
['Dubois', 'Marie', '29']
['Duval', 'Julien "Paul" ', '47']
['Jacquet', 'Bernard', '51']
['Martin', 'Lucie;Clara', '14']
```

- Il est également possible de lire les données et obtenir un dictionnaire par ligne contenant les données en utilisant **DictReader** au lieu de **reader**

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.DictReader(fichier, delimiter=";")
for ligne in lecteurCSV:
    print(ligne)
fichier.close()
```

```
{'Nom ': 'Dubois', 'Prenom': 'Marie', 'Age': '29'}
{'Nom ': 'Duval', 'Prenom': 'Julien "Paul" ', 'Age': '47'}
{'Nom ': 'Jacquet', 'Prenom': 'Bernard', 'Age': '51'}
{'Nom ': 'Martin', 'Prenom': 'Lucie;Clara', 'Age': '14'}
```



### Format CSV

- Écriture dans un fichier CSV

- À l'instar de la lecture, on ouvre un flux d'écriture et on ouvre un écrivain CSV à partir de ce flux :

```
import csv
fichier = open("annuaire.csv", "wt")
ecrivainCSV = csv.writer(fichier, delimiter=";")
ecrivainCSV.writerow(["Nom", "Prénom", "Téléphone"])      # On écrit la ligne d'en-tête avec le titre des colonnes
ecrivainCSV.writerow(["Dubois", "Marie", "0198546372"])
ecrivainCSV.writerow(["Duval", "Julien \ "Paul\\"", "0399741052"])
ecrivainCSV.writerow(["Jacquet", "Bernard", "0200749685"])
ecrivainCSV.writerow(["Martin", "Julie;Clara", "0399731590"])
fichier.close()
```

```
Nom;Prénom;Téléphone
Dubois;Marie;0198546372
Duval;"Julien ""Paul""";0399741052
Jacquet;Bernard;0200749685
Martin;"Julie;Clara";0399731590
```

### Format CSV

- Il est également possible d'écrire le fichier en fournissant un dictionnaire par ligne à condition que chaque dictionnaire possède les mêmes clés.
- Il faut également fournir la liste des clés des dictionnaires avec l'argument **fieldnames** :

```
bonCommande = [  
    {"produit": "cahier", "reference": "F452CP", "quantite": 41, "prixUnitaire": 1.6},  
    {"produit": "stylo bleu", "reference": "D857BL", "quantite": 18, "prixUnitaire": 0.95},  
    {"produit": "stylo noir", "reference": "D857NO", "quantite": 18, "prixUnitaire": 0.95},  
    {"produit": "équerre", "reference": "GF955K", "quantite": 4, "prixUnitaire": 5.10},  
    {"produit": "compas", "reference": "RT42AX", "quantite": 13, "prixUnitaire": 5.25}  
]  
fichier = open("bon-commande.csv", "wt")  
ecrivainCSV = csv.DictWriter(fichier, delimiter=";", fieldnames=bonCommande[0].keys())  
ecrivainCSV.writeheader() # On écrit la ligne d'en-tête avec le titre des colonnes  
for ligne in bonCommande:  
    ecrivainCSV.writerow(ligne)  
fichier.close()
```

```
reference;quantite;produit;prixUnitaire  
F452CP;41;cahier;1.6  
D857BL;18;stylo bleu;0.95  
D857NO;18;stylo noir;0.95  
GF955K;4;équerre;5.1  
RT42AX;13;compas;5.25
```

### Format JSON

- Le format JavaScript Object Notation (JSON) est issu de la notation des objets dans le langage JavaScript.
- Il s'agit aujourd'hui d'un format de données très répandu permettant de stocker des données sous une forme structurée.
- Il ne comporte que des associations **clés** → **valeurs** (à l'instar des dictionnaires), ainsi que des listes ordonnées de valeurs (comme les listes en Python).
- Une valeur peut être une autre association clés → valeurs, une liste de valeurs, un entier, un nombre réel, une chaîne de caractères, un booléen ou une valeur nulle.
- Sa syntaxe est similaire à celle des dictionnaires Python.

- Exemple de fichier JSON

```
{
  "Dijon":{
    "nomDepartement": "Côte d'Or",
    "codePostal": 21000,
    "population": {
      "2006": 151504,
      "2011": 151672,
      "2014": 153668
    }
  },
  "Troyes":{
    "nomDepartement": "Aube",
    "codePostal": 10000,
    "population": {
      "2006": 61344,
      "2011": 60013,
      "2014": 60750
    }
  }
}
```

### Format JSON

- Lire un fichier JSON

- La fonction `loads` (texteJSON) permet de décoder le texte JSON passé en argument et de le transformer en dictionnaire ou une liste.

```
import json
fichier = open("villes.json", "rt")
villes = json.loads(fichier.read())
print(villes)
fichier.close()
```

```
{'Troyes': {'population': {'2006': 61344, '2011': 60013, '2014': 60750}, 'codePostal': 10000, 'nomDepartement': 'Aube'}, 'Dijon': {'population': {'2006': 151504, '2011': 151672, '2014': 153668}, 'codePostal': 21000, 'nomDepartement': 'Côte d'Or'}}
```

- Écrire un fichier JSON

- la fonction `dumps(variable, sort_keys=False)` transforme un dictionnaire ou une liste en texte JSON en fournissant en argument la variable à transformer.
- La variable `sort_keys` permet de trier les clés dans l'ordre alphabétique.

```
import json
quantiteFournitures = {"cahiers":134, "stylos":{"rouge":41,"bleu":74}, "gommes": 85}
fichier = open("quantiteFournitures.json", "wt")
fichier.write(json.dumps(quantiteFournitures))
fichier.close()
```

## CHAPITRE 3

### Utiliser les expressions régulières

**Ce que vous allez apprendre dans ce chapitre :**

- Connaître le principe des expressions régulières
- Manipuler les fonctions sur les expressions régulières à savoir les fonctions de recherche, de division et de substitution



**10 heures**



## CHAPITRE 3

### Utiliser les expressions régulières

1. **Création des expressions régulières**
2. Manipulation des expressions régulières



## 03 - Utiliser les expressions régulières

### Création des expressions régulières



#### Expressions régulières de base

- Les expressions régulières fournissent une notation générale permettant de décrire abstraitement des éléments textuels
- Une expression régulière se lit de gauche à droite.
- Elle constitue ce qu'on appelle traditionnellement un motif de recherche.
- Elles utilisent **six** symboles qui, dans le contexte des expressions régulières, acquièrent les significations suivantes :

**1. le point .** représente une seule instance de n'importe quel caractère sauf le caractère de fin de ligne.

**Exemple :** l'expression t.c représente toutes les combinaisons de trois lettres commençant par « t » et finissant par « c », comme tic, tac, tqc ou t9c

**2. la paire de crochets [ ]** représente une occurrence quelconque des caractères qu'elle contient.

**Exemple :** [aeiouy] représente une voyelle, et Duran[dt] désigne Durand ou Durant.

- Entre les crochets, on peut noter un intervalle en utilisant le tiret.

**Exemple :** [0-9] représente les chiffres de 0 à 9, et [a-zA-Z] représente une lettre minuscule ou majuscule.

- On peut de plus utiliser l'accent circonflexe en première position dans les crochets pour indiquer le contraire de

**Exemple :** [^a-z] représente autre chose qu'une lettre minuscule, et [^"] n'est ni une apostrophe ni un guillemet.

**3. l'astérisque \*** est un quantificateur, il signifie aucune ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

**Exemple :** L'expression ab\* signifie la lettre a suivie de zéro ou plusieurs lettres b, par exemple ab, a ou abbb et [A-Z]\* correspond à zéro ou plusieurs lettres majuscules.

## 03 - Utiliser les expressions régulières

### Création des expressions régulières



#### Expressions régulières de base

4. **l'accent circonflexe ^** est une ancre. Il indique que l'expression qui le suit se trouve en début de ligne.

**Exemple :** l'expression ^Depuis indique que l'on recherche les lignes commençant par le mot Depuis.

5. **le symbole dollar \$** est aussi une ancre. Il indique que l'expression qui le précède se trouve en fin de ligne.

**Exemple :** L'expression suivante :\$ indique que l'on recherche les lignes se terminant par « suivante : ».

6. **la contre-oblique \** permet d'échapper à la signification des métacaractères. Ainsi \. désigne un véritable point, \\* un astérisque, \^ un accent circonflexe, \$ un dollar et \\ une contre-oblique

#### Expressions régulières étendues

- Elles ajoutent cinq symboles qui ont les significations suivantes :

1. **la paire de parenthèses ( )** : est utilisée à la fois pour former des sous-motifs et pour délimiter des sous-expressions, ce qui permettra d'extraire des parties d'une chaîne de caractères.

**Exemple :** L'expression (to)\* désignera to, tototo, etc.

2. **le signe plus +** : est un quantificateur comme \*, mais il signifie une ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

**Exemple :** L'expression ab+ signifie la lettre a suivie d'une ou plus

3. **le point d'interrogation ?** : il signifie zéro ou une instance de l'expression qui le précède.

**Exemple :** écran(s)? désigne écran ou écrans ;



## 03 - Utiliser les expressions régulières

### Création des expressions régulières



#### Expressions régulières étendues

4. **la paire d'accolades { }** : précise le nombre d'occurrences permises pour le motif qui le précède.

**Exemple :** `[0-9]{2,5}` attend entre deux et cinq nombres décimaux.

- Les variantes suivantes sont disponibles : `[0-9]{2,}` signifie au minimum deux occurrences d'entiers décimaux et `[0-9]{2}` deux occurrences exactement

5. **la barre verticale |** : représente des choix multiples dans un sous-motif.

**Exemple :** L'expression `Duran[d|t]` peut aussi s'écrire (`Durand|Durant`). On pourrait utiliser l'expression (`lu|ma|me|je|ve|sa|di`) dans l'écriture d'une date.

- **la syntaxe étendue comprend aussi une série de séquences d'échappement :**

- `\` : symbole d'échappement ;
- `\e` : séquence de contrôle escape ;
- `\f` : saut de page ;
- `\n` : fin de ligne ;
- `\r` : retour-chariot ;
- `\t` : tabulation horizontale ;
- `\v` : tabulation verticale ;
- `\d` : classe des nombres entiers ;
- `\s` : classe des caractères d'espacement ;
- `\w` : classe des caractères alphanumériques ;
- `\b` : délimiteurs de début ou de fin de mot ;
- `\D` : négation de la classe `\d` ;
- `\S` : négation de la classe `\s` ;
- `\W` : négation de la classe `\w` ;
- `\B` : négation de la classe `\b`.

## CHAPITRE 3

### Utiliser les expressions régulières

1. Création des expressions régulières
2. **Manipulation des expressions régulières**



## 03 - Utiliser les expressions régulières

### Manipulation des expressions régulières

- Le module `re` permet d'utiliser les expressions régulières dans les scripts Python.
- Les scripts devront donc comporter la ligne : **`import re`**
- **Fonction de compilation d'une regex en Python:**
  - Pour initialiser une expression régulière avec Python, il est possible de la compiler, surtout si vous serez amené à l'utiliser plusieurs fois tout au long du programme. **Pour ce faire, il faut utiliser la fonction `compile()`**

```
expression = re.compile(r'\d{1,3}')
```

### Les options de compilation :

- Grâce à un jeu d'options de compilation, il est possible de piloter le comportement des expressions régulières. On utilise pour cela la syntaxe `(?...)` avec les drapeaux suivants :
  - **a** : correspondance ASCII (Unicode par défaut) ;
  - **i** : correspondance non sensible à la casse ;
  - **L** : les correspondances utilisent la locale, c'est-à-dire les particularités du pays ;
  - **m** : correspondance dans des chaînes multilignes ;
  - **s** : modifie le comportement du métacaractère point qui représentera alors aussi le saut de ligne ;
  - **u** : correspondance Unicode (par défaut) ;
  - **x** : mode verbeux.

```
import re
case = re.compile(r"[a-z]+")
ignore_case = re.compile(r"(?i)[a-z]+")
print(case.search("Bastille").group()) #affiche: astille
print(ignore_case.search("Bastille").group()) #affiche: Bastille
```

## 03 - Utiliser les expressions régulières

### Manipulation des expressions régulières

- Le module 're' propose un ensemble de fonctions qui nous permet de rechercher une chaîne pour une correspondance :

Fonction	Description
findall	Renvoie une liste contenant toutes les correspondances
search	Renvoie un objet Match s'il existe une correspondance n'importe où dans la chaîne
split	Renvoie une liste où la chaîne a été divisée à chaque correspondance
sub	Remplace une ou plusieurs correspondances par une chaîne donnée

- La fonction **findall()** renvoie une liste contenant toutes les correspondances. La liste contient les correspondances dans l'ordre où elles sont trouvées. Si aucune correspondance n'est trouvée, une liste vide est renvoyée.

Exemple :

```
import re
Nameage = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
expression = re.compile(r'\d{1,3}')

ages = expression.findall(Nameage)
print(ages)#affiche: ['22', '33', '44', '21']
names = re.findall(r'[A-Z][a-z]*',Nameage)
print(names)#affiche: ['Janice', 'Theon', 'Gabriel', 'Joey']
```

## 03 - Utiliser les expressions régulières

### Manipulation des expressions régulières

- La fonction `search()` recherche une correspondance dans la chaîne et renvoie un objet `Match` s'il existe une correspondance. S'il y a plus d'une correspondance, seule la première occurrence de la correspondance sera renvoyée:

#### Exemple : Extraction simple

- La variable `expression` reçoit la forme compilée de l'expression régulière, Puis on applique à ce motif compilé la méthode `search()` qui retourne la première position du motif dans la chaîne `Nameage` et l'affecte à la variable `ages`. Enfin on affiche la correspondance complète (en ne donnant pas d'argument à `group()`)

```
import re
Nameage = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
expression = re.compile(r'\d{1,3}')

ages = expression.search(Nameage)
print(ages.group())#affiche: 22
```

Recherche de un, deux ou trois entiers décimaux

#### Exemple : Extraction des sous-groupes

- Il est possible d'affiner l'affichage du résultat en modifiant l'expression régulière de recherche de façon à pouvoir capturer les éléments du motif

```
import re
motif_date = re.compile(r"(\d\d ?) (\w+) (\d{4})")
corresp = motif_date.search("Bastille le 14 juillet 1789")
print("corresp.group() :", corresp.group())
print("corresp.group(1) :", corresp.group(1))
print("corresp.group(2) :", corresp.group(2))
print("corresp.group(3) :", corresp.group(3))
print("corresp.group(1,3) :", corresp.group(1,3))
print("corresp.groups() :", corresp.groups())
```

```
corresp.group() : 14 juillet 1789
corresp.group(1) : 14
corresp.group(2) : juillet
corresp.group(3) : 1789
corresp.group(1,3) : ('14', '1789')
corresp.groups() : ('14', 'juillet', '1789')
```

## 03 - Utiliser les expressions régulières

### Manipulation des expressions régulières

- Python possède une syntaxe qui permet de nommer des parties de motif délimitées par des parenthèses, ce qu'on appelle un motif nominatif :
  - syntaxe de création d'un motif nominatif : `(?P<nom_du_motif>)` ;
  - syntaxe permettant de s'y référer : `(?P=nom_du_motif)` ;

#### Exemple : Extraction des sous-groupes nommés

- la méthode **groupdict()** renvoie une liste comportant le nom et la valeur des sous-groupes trouvés (ce qui nécessite de nommer les sous-groupes).

```
import re
motif_date = re.compile(r"(?P<jour>\d\d ?) (?P<mois>\w+) (\d{4})")
corresp = motif_date.search("Bastille le 14 juillet 1789")
print(corresp.groupdict())
print(corresp.group('jour'))
print(corresp.group('mois'))
```

```
{'jour': '14', 'mois': 'juillet'}
14
juillet
```

- La fonction **split()** renvoie une liste où la chaîne a été divisée à chaque correspondance
- L'exemple suivant divise la chaîne à chaque espace trouvé. `\s` est utilisé pour faire correspondre les espaces.

```
import re
adresse = "Rue 41 de la République"
liste = re.split("\s", adresse)
print(liste) # affiche: ['Rue', '41', 'de', 'la', 'République']
```

## 03 - Utiliser les expressions régulières

### Manipulation des expressions régulières

- Il est possible de contrôler le nombre d'occurrences en spécifiant le paramètre **maxsplit**:

Exemple : maxsplit=1

```
import re
adresse = "Rue 41 de la République"
liste = re.split("\s", adresse, 1)
print(liste) # affiche: ['Rue', '41 de la République']
```

- La fonction **sub()** remplace les correspondances par le texte de votre choix

Exemple : Remplacer chaque espace par un tiret '-':

```
import re
adresse = "Rue 41 de la République"
res = re.sub("\s", "-", adresse) # affiche: Rue-41-de-la-République
print(res)
```

- Il est possible de contrôler le nombre de remplacements en spécifiant le paramètre **count**:

Exemple : count=2

```
import re
adresse = "Rue 41 de la République"
res = re.sub("\s", "-", adresse, 2)
print(res) # affiche: Rue-41-de la République
```

## CHAPITRE 4

### Administrer les exceptions

Ce que vous allez apprendre dans ce chapitre :

- Identifier les différents types d'erreurs
- Connaître les principaux types d'exceptions en Python
- Maîtriser la gestion des exceptions en Python



05 heures





# CHAPITRE 4

## Administrer les exceptions

1. **Types d'erreur et expérimentation**
2. Types des exceptions
3. Gestion des exceptions



## 04 - Administrer les exceptions

### Types d'erreur et expérimentation

#### Erreurs de syntaxe

- Les erreurs de syntaxe sont des erreurs d'analyse du code

Exemple :

```
>>> while True print('Hello world')
File "<stdin>", line 1
    while True print('Hello world')
            ^
SyntaxError: invalid syntax
```

- L'analyseur indique la ligne incriminée et affiche une petite « flèche » pointant vers le premier endroit de la ligne où l'erreur a été détectée.
- L'erreur est causée par le symbole placé avant la flèche.
- Dans cet exemple, la flèche est sur la fonction print() car il manque deux points (':') juste avant. Le nom du fichier et le numéro de ligne sont affichés pour vous permettre de localiser facilement l'erreur lorsque le code provient d'un script.

## 04 - Administrer les exceptions

### Types d'erreur et expérimentation



#### Exceptions

- Même si une instruction ou une expression est syntaxiquement correcte, elle peut générer une erreur lors de son exécution.
- Les erreurs détectées durant l'exécution sont appelées des exceptions et ne sont pas toujours fatales
- La plupart des **exceptions** toutefois ne sont pas prises en charge par les programmes, ce qui génère des messages d'erreurs comme celui-ci :

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- La dernière ligne du message d'erreur indique ce qui s'est passé. Les exceptions peuvent être de différents types et ce type est indiqué dans le message : le type indiqué dans l'exemple est ZeroDivisionError

## CHAPITRE 4

### Administrer les exceptions

1. Types d'erreur et expérimentation
2. **Types des exceptions**
3. Gestion des exceptions



## 04 - Administrer les exceptions

### Types des exceptions

- En Python, les erreurs détectées durant l'exécution d'un script sont appelées des exceptions car elles correspondent à un état "exceptionnel" du script
- Python analyse le type d'erreur déclenché
- Python possède de nombreuses classes d'exceptions natives et toute exception est une instance (un objet) créée à partir d'une classe exception
- La classe d'exception de base pour les exceptions natives est **BaseException**
- Quatre classes d'exception dérivent de la classe **BaseException** à savoir :

#### Exception

- Toutes les exceptions intégrées non-exit du système sont dérivées de cette classe
- Toutes les exceptions définies par l'utilisateur doivent également être dérivées de cette classe

#### SystemExit

- Déclenchée par la fonction `sys.exit()` si la valeur associée est un entier simple, elle spécifie l'état de sortie du système (passé à la fonction `exit()` de C)

#### GeneratorExit

- Levée lorsque la méthode `close()` d'un générateur est appelée

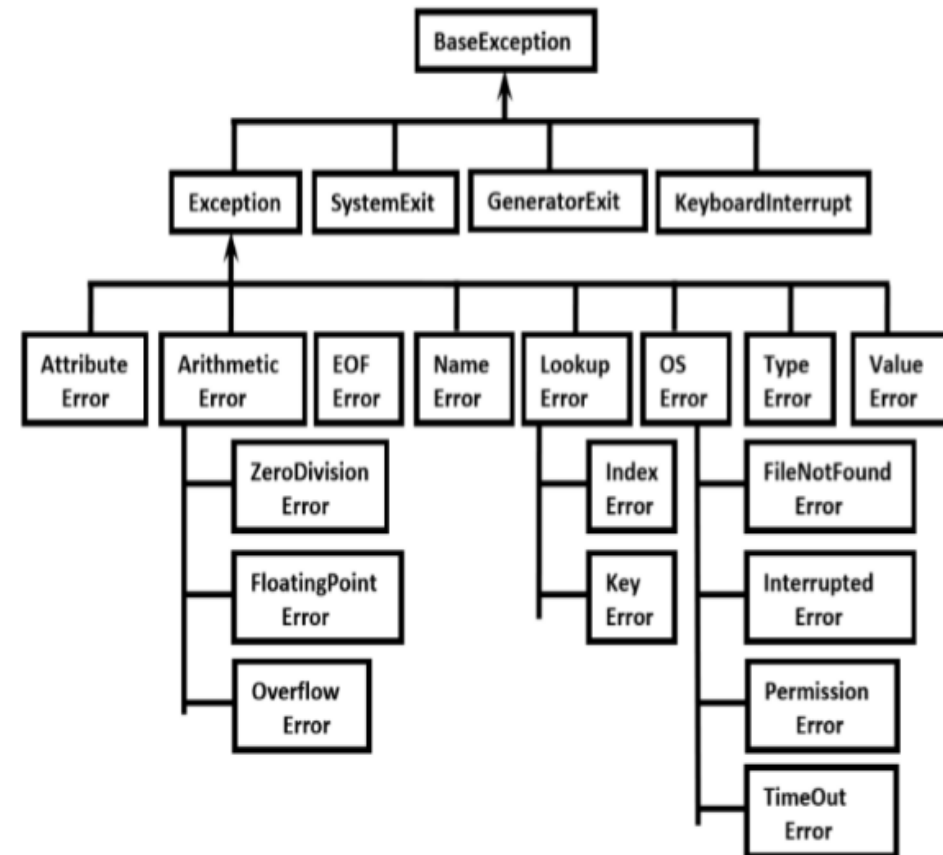
#### KeyboardInterrupt

- Levée lorsque l'utilisateur appuie sur la touche d'interruption (normalement Control-C ou Delete)

## 04 - Administrer les exceptions

### Types des exceptions

- Il y a également d'autres classes **d'exception** qui dérivent de Exception telles que:
  - La classe **ArithmeticError** est la classe de base pour les exceptions natives qui sont levées pour diverses erreurs arithmétiques et notamment pour les classes **OverflowError**, **ZeroDivisionError** et **FloatInfgPointError** ;
  - La classe **LookupError** est la classe de base pour les exceptions qui sont levées lorsqu'une clé ou un index utilisé sur un tableau de correspondances où une séquence est invalide.
- De nombreuses classes dérivent ensuite de ces classes.
- En fonction de l'erreur rencontrée par l'analyseur Python, un objet exception appartenant à telle ou telle classe exception va être créé et renvoyé. Cet objet est intercepté et manipulé.



## CHAPITRE 4

### Administrer les exceptions

1. Types d'erreur et expérimentation
2. Types des exceptions
3. **Gestion des exceptions**



## 04 - Administrer les exceptions

### Gestion des exceptions

#### Détection et traitement des exceptions en Python

- On peut détecter les exceptions en plaçant les instructions qui peuvent générer des exceptions dans un bloc **try**.
- Il existe 2 formes d'expressions try mutuellement exclusives (on ne peut en employer qu'une à la fois) : **try-except** et **try-finally**.
- Une instruction **try** peut être accompagnée d'une ou plusieurs clauses **except**, d'une seule clause **finally** ou d'une combinaison **try-except-finally**.

#### Exemple :

- On souhaite calculer l'âge saisi par l'utilisateur en soustrayant son année de naissance à 2016. Pour cela, il faut convertir la valeur de la variable `birthyear` en un `int`.
- Cette conversion peut échouer si la chaîne de caractères entrée par l'utilisateur n'est pas un nombre.

```
birthyear = input('Année de naissance ? ')

try:
    print('Tu as', 2021- int(birthyear), 'ans.')
except:
    print('Erreur, veuillez entrer un nombre.')

print('Fin du programme.')
```



## 04 - Administrer les exceptions

### Gestion des exceptions

#### Instruction try-except

```
birthyear = input('Année de naissance ? ')
try:
    print('Tu as', 2016 - int(birthyear), 'ans.')
except:
    print('Erreur, veuillez entrer un nombre.')
print('Fin du programme.')
```

- Dans le premier cas, la conversion s'est passée normalement, et le bloc try a donc pu s'exécuter intégralement sans erreur.
- Dans le second cas, une erreur se produit dans le bloc try, lors de la conversion. L'exécution de ce bloc s'arrête donc immédiatement et passe au bloc except, avant de continuer également après l'instruction try-except.

#### 1er cas

- Si l'utilisateur entre un nombre entier, l'exécution se passe sans erreur et son âge est calculé et affiché

Année de naissance ? 1994  
Tu as 22 ans.  
Fin du programme.

#### 2ème cas

- Si l'utilisateur entre une chaîne de caractères quelconque, qui ne représente pas un nombre entier, un message d'erreur est affiché

Année de naissance ? deux  
Erreur, veuillez entrer un nombre.  
Fin du programme.

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Type Exception

- Lorsqu'on utilise l'instruction **try-except**, le bloc **except** capture toutes les erreurs possibles qui peuvent survenir dans le bloc try correspondant.
- Une **exception** est en fait représentée par un objet, instance de la classe Exception.
- On peut récupérer cet objet en précisant un nom de variable après **except**.

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)
```

- On récupère donc l'objet de type Exception dans la variable e.
- Dans le bloc except, on affiche son type et sa valeur.

## 04 - Administrer les exceptions

### Gestion des exceptions

### Type Exception

Exemples d'exécution qui révèlent deux types d'erreurs différents :

- Si on ne fournit pas un nombre entier, il ne pourra être converti en **int** et une erreur de type **ValueError** se produit :

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)

a ? trois
<class 'ValueError'>
invalid literal for int() with base 10: 'trois'
```

- Si on fournit une valeur de 00 pour b, on aura une division par zéro qui produit une erreur de type **ZeroDivisionError** :

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)

a ? 5
b ? 0
<class 'ZeroDivisionError'>
division by zero
```

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)
```

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Capture d'erreur spécifique

- Chaque type d'erreur est donc défini par une classe spécifique.
- Il est possible d'associer plusieurs blocs except à un même bloc try, pour exécuter un code différent en fonction de l'erreur capturée.
- Lorsqu'une erreur se produit, les blocs except sont parcourus l'un après l'autre, du premier au dernier, jusqu'à en trouver un qui corresponde à l'erreur capturée.

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)

except ValueError:
    print('Erreur de conversion.')
except ZeroDivisionError:
    print('Division par zéro.')
except:
    print('Autre erreur.')
```

#### Exemple :

- Lorsqu'une erreur se produit dans le bloc try l'un des blocs except seulement qui sera exécuté, selon le type de l'erreur qui s'est produite.
- Le dernier bloc except est là pour prendre toutes les autres erreurs.
- Lorsqu'une erreur se produit dans le bloc try l'un des blocs except seulement qui sera exécuté, selon le type de l'erreur qui s'est produite.
- Le dernier bloc except est là pour prendre toutes les autres erreurs.
- **L'ordre des blocs except est très important et il faut les classer du plus spécifique au plus général, celui par défaut devant venir en dernier.**

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Gestionnaire d'erreur partagé

- Il est possible d'exécuter le même code pour différents types d'erreur, en les listant dans un tuple après le mot réservé **except**.
- Si on souhaite exécuter le même code pour une erreur de conversion et de division par zéro, il faudrait écrire :

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)

except (ValueError, ZeroDivisionError) as e:
    print('Erreur de calcul :', e)
except:
    print('Autre erreur.')
```

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Bloc finally

- Le mot réservé **finally** permet d'introduire un bloc qui sera exécuté soit après que le bloc try se soit exécuté complètement sans erreur, soit après avoir exécuté le bloc **except** correspondant à l'erreur qui s'est produite lors de l'exécution du bloc try.
- On obtient ainsi une instruction **try-except-finally**

```
print('Début du calcul.')
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print('Résultat : ', a / b)
except:
    print('Erreur.')
finally:
    print('Nettoyage de la mémoire.')
print('Fin du calcul.')
```

- Si l'utilisateur fournit des valeurs correctes pour a et b l'affichage est le suivant :

```
Début du calcul.
a ? 2
b ? 8
Résultat : 0.25
Nettoyage de la mémoire.
Fin du calcul.
```

- Si une erreur se produit l'affichage est le suivant :

```
Début du calcul.
a ? 2
b ? 0
Erreur.
Nettoyage de la mémoire.
Fin du calcul.
```

Dans les 2 cas le  
**bloc finally** a  
été exécuté

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Génération d'erreur

- Il est possible de générer une erreur dans un programme grâce à l'instruction **raise**.
- Il suffit en fait simplement d'utiliser le mot réservé **raise** suivi d'une référence vers un objet représentant une exception.

Exemple :

```
def fact(n):  
    if n < 0:  
        raise ArithmeticError()  
    if n == 0:  
        return 1  
    return n * fact(n - 1)  
  
print(fact(-12))
```

- Le programme suivant permet de capturer spécifiquement l'exception de type **ArithmeticError** lors de l'appel de la fonction **fact**.

```
try:  
    n = int(input('Entrez un nombre : '))  
    print(fact(n))  
except ArithmeticError:  
    print('Veuillez entrer un nombre positif.')  
except:  
    print('Veuillez entrer un nombre.')
```

- Si **n** est strictement négatif, une exception de type **ArithmeticError** est générée.

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Créer un type d'exception

- Il est parfois plus pratique et plus lisible de définir nos propres types d'exceptions
- Pour cela, il suffit de définir une nouvelle classe qui hérite de la classe Exception

Exemple :

```
class NoRootException(Exception):  
    pass
```

- Cette classe est tout simplement vide puisque son corps n'est constitué que de l'instruction pass

Exemple :

- Définissons une fonction **trinomialroots** qui calcule et renvoie les racines d'un trinôme du second degré de la forme  $ax^2+bx+c$  et qui génère une erreur lorsqu'il n'y a pas de racine réelle :

```
from math import sqrt  
  
def trinomialroots(a, b, c):  
    delta = b ** 2 - 4 * a * c  
    # Aucune racine réelle  
    if delta < 0:  
        raise NoRootException()  
    # Une racine réelle double  
    if delta == 0:  
        return -b / (2 * a)  
    # Deux racines réelles simples  
    x1 = (-b + sqrt(delta)) / (2 * a)  
    x2 = (-b - sqrt(delta)) / (2 * a)  
    return (x1, x2)
```

Lever une exception avec **raise**



### Exception paramétrée

- Lorsqu'on appelle la fonction **trinomialroots**, on va donc pouvoir utiliser l'instruction try-except pour attraper cette erreur, lorsqu'elle survient
- Essayons, par exemple, de calculer et d'afficher les racines réelles du trinôme  $x+2$ . Pour cela, on appelle donc la fonction **trinomialroots** en lui passant en paramètres 1, 0 et 2 puisque  $x+2$  correspond à  $a=1$ ,  $b=0$  et  $c=2$

```
try:
    print(trinomialroots(1, 0, 2))
except NoRootException:
    print('Pas de racine réelle.')

#Pas de racine réelle.
```

Attraper une exception avec le bloc try-except

- Pour l'exemple précédent, il pourrait être utile de connaître la valeur du discriminant
- Lorsqu'aucune racine réelle n'existe. Pour cela il faut ajouter une variable d'instance et
- Un accesseur à la classe **NoRootException**

```
class NoRootException(Exception):
    def __init__(self, delta):
        self.__delta = delta

    @property
    def delta(self):
        return self.__delta
```

```
from math import sqrt

def trinomialroots(a, b, c):
    delta = b ** 2 - 4 * a * c
    # Aucune racine réelle
    if delta < 0:
        raise NoRootException(delta)
    # Une racine réelle double
    if delta == 0:
        return -b / (2 * a)
    # Deux racines réelles simples
    x1 = (-b + sqrt(delta)) / (2 * a)
    x2 = (-b - sqrt(delta)) / (2 * a)
    return (x1, x2)
```

## 04 - Administrer les exceptions

### Gestion des exceptions

#### Exception paramétrée

- Il est possible de récupérer la valeur du discriminant dans le bloc **except**, à partir de l'objet représentant l'exception qui s'est produite

```
from math import sqrt

def trinomialroots(a, b, c):
    delta = b ** 2 - 4 * a * c
    # Aucune racine réelle
    if delta < 0:
        raise NoRootException(delta)
    # Une racine réelle double
    if delta == 0:
        return -b / (2 * a)
    # Deux racines réelles simples
    x1 = (-b + sqrt(delta)) / (2 * a)
    x2 = (-b - sqrt(delta)) / (2 * a)
    return (x1, x2)
```

```
try:
    print(trinomialroots(1, 0, 2))
except NoRootException as e:
    print('Pas de racine réelle.')
    print('Delta =', e.delta)

# Pas de racine réelle.
# Delta = -8
```

## PARTIE 4

# MANIPULER LES MODULES ET LES BIBLIOTHEQUES

Dans ce module, vous allez :

- Apprendre la création de modules en Python
- Maîtriser l'importation des modules
- Installer des bibliothèques en Python
- Créer des bibliothèques en Python
- Importer des bibliothèques en Python



**30 heures**

# CHAPITRE 1

## Manipuler les modules

Ce que vous allez apprendre dans ce chapitre :

- Créer des modules en Python
- Catégoriser les types de modules
- Maitriser l'importation absolue
- Maitriser l'importation relative



05 heures



# CHAPITRE 1

## Manipuler les modules

1. Création des modules
2. Importation des modules



# 01 - Manipuler les modules

## Création des modules

### Création de modules

- Un module est **un fichier « .py »** contenant un ensemble de **variables, fonctions et classes** que l'on peut importer et utiliser dans le programme principal (ou dans d'autres modules).
- Pour créer un module, il suffit de programmer les variables/fonctions et classes qui le constituent dans un fichier portant le nom du module, suivi du suffixe « .py ». Depuis un (autre) programme en Python, il suffit alors d'utiliser la primitive import pour pouvoir utiliser ces variables/fonctions/classes.
- **Les modules :**
  - Permettent la **séparation** du code et donc une **meilleure organisation du code**
  - Maximisent la **réutilisation**
  - **Facilitent le partage du code**

### Exemple :

```
"""
exemple de module, aide associée
"""

exemple_variable = 3

def exemple_fonction():
    """exemple de fonction"""
    return 0

class exemple_classe:
    """exemple de classe"""

    def __str__(self):
        return "exemple_classe"
```

- Un module qui contient une fonction, une classe et une variable. Ces trois éléments peuvent être utilisés par n'importe quel fichier qui importe ce module. Le nom d'un module correspond au nom du fichier sans son extension.

# 01 - Manipuler les modules

## Création des modules



En Python, on peut distinguer trois grandes catégories de modules en les classant selon leur éditeur :

- Des modules standards prêts à l'emploi sont livrés avec la distribution Python. Ex. random, math, os, hashlib, etc ;
- Des modules développés par des développeurs externes qu'on va pouvoir utiliser en les important ;
- Des modules qu'on va développer nous-mêmes.

# CHAPITRE 1

## Manipuler les modules

1. Création des modules
2. **Importation des modules**





# 01 - Manipuler les modules

## Importation des modules



- Pour importer un module, il suffit d'insérer l'instruction **import nom\_module** avant d'utiliser une des choses qu'il définit.
- Les importations sont souvent regroupées au début du programme, elles sont de cette façon mises en évidence même s'il est possible de les faire n'importe où.

```
import module_exemple

c = module_exemple.exemple_classe ()
print(c)
print(module_exemple.exemple_fonction())
```

### Exemple :

- Il en existe une autre méthode d'importation de modules qui permet d'affecter à un module un identificateur différent du nom du fichier dans lequel il est décrit.
- En ajoutant l'instruction **as** suivi d'un autre nom alias, le module sera désigné par la suite par l'identificateur **alias**.

```
import module_exemple as alias

c = alias.exemple_classe()
print(c)
print(alias.exemple_fonction())
```

- La syntaxe suivante n'est pas recommandée car elle masque le module d'où provient une fonction en plus de tout importer.

# 01 - Manipuler les modules

## Importation des modules

- **import \*** permet d'importer toutes les classes, attributs ou fonctions d'un module mais il est possible d'écrire **from module\_exemple import exemple\_class** pour n'importer que cette classe.
- Lorsqu'on importe un module, l'interpréteur Python le recherche dans différents répertoires selon l'ordre suivant :
  1. Le répertoire courant ;
  2. Si le module est introuvable, Python recherche ensuite chaque répertoire listé dans la variable shell PYTHONPATH ;
  3. Si tout échoue, Python vérifie le chemin par défaut (exemple pour windows \Python\Python39\Lib)

```
from module_exemple import * # décommandé
from module_exemple import exemple_classe, exemple_fonction

c = exemple_classe()
print(c)
print(exemple_fonction())|
```

# 01 - Manipuler les modules

## Importation des modules

### Arborescence de modules

- Lorsque le nombre de modules devient conséquent, il est parfois souhaitable de répartir tous ces fichiers dans plusieurs répertoires. Il faudrait alors inclure tous ces répertoires dans la liste **sys.path** ce qui paraît fastidieux
- Python propose la définition de paquetage, ce dernier englobe tous les fichiers python d'un répertoire (package)
- Avant Python 3.3, un package contenant des modules Python doit contenir un fichier **\_\_init\_\_.py**

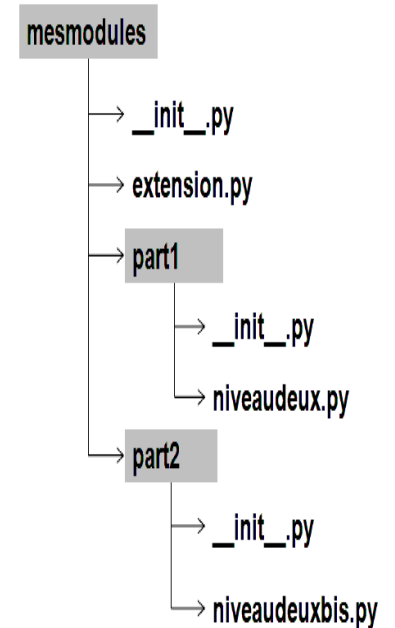
### Importation absolue

- Une importation absolue spécifie la ressource à importer à l'aide de son chemin d'accès complet à partir du répertoire racine.

Exemple :

```
import mesmodules.extension
import mesmodules.part1.niveaudeux
import mesmodules.part2.niveaudeuxbis
```

- Lors de la première instruction **import mesmodules.extension**, le langage Python ne s'intéresse pas qu'au seul fichier extension.py, il exécute également le contenu du fichier **\_\_init\_\_.py**
- Dans **\_\_init\_\_.py**, il faut insérer les instructions à exécuter avant l'import de n'importe quel module du paquetage



# 01 - Manipuler les modules

## Importation des modules

### Importation relative

- Une importation relative spécifie la ressource à importer par rapport à l'emplacement actuel, c'est-à-dire l'emplacement où se trouve l'instruction import.
  - Le **symbole** `.` permet d'importer un module dans le même répertoire.
  - Le **symbole** `..` permet d'importer un module dans le répertoire parent.

#### Exemple :

- La **fonction A** peut utiliser la **fonction B** ou **C** en les important de la façon suivante :

```
from .subpackage1 import B
from .subpackage1.moduleX import C
```

- La **fonction E** peut utiliser la **fonction F** ou **A** ou **C** en les important de la façon suivante :

```
from ..moduleA import F
from .. import A
from ..subpackage1.moduleX import C
```

```
package/
  __init__.py      # fonction A
  subpackage1/
    __init__.py    # fonction B
    moduleX.py     # fonction C
  subpackage2/
    __init__.py    # fonction D
    moduleY.py     # fonction E
  moduleA.py       # fonction F
```

# 01 - Manipuler les modules

## Importation des modules

### PYTHONPATH

- Pour ajouter un dossier au PYTHONPATH en Python, il faut indiquer directement dans le code les lignes suivantes :

```
import sys  
sys.path.insert(0, "E:/exempleImport")
```

Chemin du module à emporter

- La fonction path du module sys permet de vérifier la variable PYTHONPATH

```
import sys  
print(sys.path)
```

- Remarque :** insert ne permet pas d'ajouter le dossier en question dans PYTHONPATH de façon permanente

```
['', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39\\Lib\\idlelib', 'E:\\exempleImport', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39\\python39.zip', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39\\DLLs', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39\\lib', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39', 'C:\\Users\\DELL\\AppData\\Local\\Programs\\Python\\Python39\\lib\\site-packages']
```

## CHAPITRE 2

### Manipuler les bibliothèques

**Ce que vous allez apprendre dans ce chapitre :**

- Installer des bibliothèques standards en Python
- Manipuler la bibliothèque graphique Tinker
- Créer des bibliothèques en Python
- Importer des bibliothèques en Python



**25 heures**



# CHAPITRE 2

## Manipuler les bibliothèques

1. **Installation des bibliothèques externes (pip)**
2. Création des bibliothèques
3. Importation des bibliothèques



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



#### Bibliothèque en Python

- Dans Python, une bibliothèque est un ensemble logiciel de modules (classes (types d'objets), fonctions, constantes...) ajoutant des possibilités étendues à Python : calcul numérique, graphisme, programmation internet ou réseau, formatage de texte, génération de documents, etc.
- Il en existe un très grand nombre, et c'est d'ailleurs une des grandes forces de Python.
- La plupart est regroupée dans PyPI (Python Package Index) le dépôt tiers officiel du langage de programmation Python.

#### Bibliothèque standard

- La distribution standard de Python dispose d'une très riche bibliothèque de modules étendant les capacités du langage dans de nombreux domaines.
- La bibliothèque standard couvre un large éventail de fonctionnalités, notamment :
  - Modules de date et d'heure
  - Modules des interfaces graphiques
  - Modules numériques et mathématiques
  - Modules de système de fichiers
  - Modules de système d'exploitation
  - Modules pour la lecture et l'écriture de formats de données spécifiques tels que HTML, XML et JSON
  - Modules pour l'utilisation de protocoles Internet tels que HTTP, SMTP, FTP, etc.
  - Modules pour l'utilisation de données multimédias telles que les données audio et vidéo



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



#### Pip (Python Installer Package)

- Pip (Python Installer Package) est le manager de package pour Python
- Pip est un moyen d'installer et de gérer des packages et des dépendances supplémentaires qui ne sont pas encore distribués dans le cadre de la version standard du package
- Pip package est intégré dans l'installation du Python depuis les versions 3.4 pour Python3 et les versions 2.7.9 pour Python2, et utilisé dans nombreux projets du Python.
- En exécutant la commande ci-dessus, il est possible de vérifier que pip est disponible ou non

```
pip --version
```

- Résultat de l'exécution :

```
pip 21.2.4 from C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pip (python 3.9)
```

- La sortie permet d'afficher la version de **pip** dans votre machine, ainsi que l'emplacement de votre version du Python
- Si vous utilisez une ancienne version de Python qui n'inclut pas **pip**, vous devez l'installer

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Installation de Pip

##### 1ère méthode :

- Téléchargez get-pip.py (<https://bootstrap.pypa.io/get-pip.py>) dans un dossier de votre ordinateur.
- Ouvrez l'invite de commande et accédez au dossier contenant le programme d'installation get-pip.py.
- Exécutez la commande suivante :

```
py get-pip.py
```

##### 2ème méthode :

- Voici la commande pour le télécharger avec l'outil **Wget** pour Windows

```
wget https://bootstrap.pypa.io/get-pip.py
```

- Pour se renseigner sur les commandes supportées par **pip** utilisez la commande suivante :

```
pip help
```

- Résultat de l'exécution :

```
Usage:
  C:\Users\DELL\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip <command> [options]

Commands:
  install             Install packages.
  download            Download packages.
  uninstall           Uninstall packages.
  freeze             Output installed packages in requirements format.
  list               List installed packages.
  show               Show information about installed packages.
  check              Verify installed packages have compatible dependencies.
  config             Manage local and global configuration.
  search             Search PyPI for packages.
  cache              Inspect and manage pip's wheel cache.
  index              Inspect information available from package indexes.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion         A helper command used for command completion.
  debug             Show information useful for debugging.
  help              Show help for commands.

General Options:
  -h, --help          Show help.
  --isolated          Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose       Give more output. Option is additive, and can be used up to 3 times.
  -V, --version       Show version and exit.
  -q, --quiet         Give less output. Option is additive, and can be used up to 3 times (corresponding to
                     WARNING, ERROR, and CRITICAL logging levels).
  --log <path>       Path to a verbose appending log.
  --no-input          Disable prompting for input.
  --proxy <proxy>     Specify a proxy in the form [user:passwd@]proxy.server:port.
  --retries <retries> Maximum number of retries each connection should attempt (default 5 times).
  --timeout <sec>     Set the socket timeout (default 15 seconds).
  --exists-action <action> Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup,
  --trusted-host <hostname> (a)bort.
  --cert <path>       Mark this host or host:port pair as trusted, even though it does not have valid or any
                     HTTPS.
                     Path to PEM-encoded CA certificate bundle. If provided, overrides the default. See 'SSL
                     Certificate Verification' in pip documentation for more information.
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Installation des bibliothèques

- **pip** fournit une commande d'installation pour installer des packages/bibliothèques

```
py -m pip install sampleproject
```

```
Uninstalling sampleproject:  
[...]  
Proceed (y/n)? y  
Successfully uninstalled sampleproject
```

- Il est aussi possible de préciser une version minimum exacte directement depuis la ligne de commande
- Utiliser des caractères de comparaison tel que >, < ou d'autres caractères spéciaux qui sont interprétés par le shell, le nom du paquet et la version doivent être mis entre guillemets

```
py -m pip install SomePackage==1.0.4 # specific version  
py -m pip install "SomePackage>=1.0.4" # minimum version
```

- Normalement, si une bibliothèque appropriée est déjà installée, l'installer à nouveau n'aura aucun effet

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



#### Installation des bibliothèques

- La mise à jour des bibliothèques existantes doit être demandée explicitement :

```
py -m pip install --upgrade SomePackage
```

- Pour désinstaller une bibliothèque, il faut utiliser la commande suivante

```
py -m pip uninstall sampleproject
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Bibliothèques graphiques

- La bibliothèque **matplotlib** (et sa sous-bibliothèque pyplot) sert essentiellement à afficher des graphismes. Son utilisation ressemble beaucoup à celle de Matlab.
- Pour installer **matplotlib**, il faut taper la commande :

```
py -m pip install matplotlib
```

```
PS C:\Users\DELL> pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.4.3-cp39-cp39-win_amd64.whl (7.1 MB)
    | 7.1 MB 2.2 MB/s
Collecting numpy>=1.16
  Using cached numpy-1.21.3-cp39-cp39-win_amd64.whl (14.0 MB)
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.2-cp39-cp39-win_amd64.whl (52 kB)
    | 52 kB 84 kB/s
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\dell\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages (from matplotlib) (2.4.7)
Collecting pillow>=6.2.0
  Downloading Pillow-8.4.0-cp39-cp39-win_amd64.whl (3.2 MB)
    | 3.2 MB 1.7 MB/s
Collecting python-dateutil>=2.7
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    | 247 kB 2.2 MB/s
Requirement already satisfied: six in c:\users\dell\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages (from cycler>=0.10->matplotlib) (1.16.0)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Bibliothèques graphiques

- PyQt est un module libre qui permet de créer des interfaces graphiques en Python.
- Pour installer PyQt, il faut taper la commande :

```
py -m pip install pyqt5
```

```
PS C:\Users\DELL> pip install pyqt5
Collecting pyqt5
  Downloading PyQt5-5.15.5-cp36-abi3-win_amd64.whl (6.7 MB)
    |████████████████████| 6.7 MB 2.2 MB/s
Collecting PyQt5-sip<13,>=12.8
  Downloading PyQt5_sip-12.9.0-cp39-cp39-win_amd64.whl (63 kB)
    |██████████████████| 63 kB 280 kB/s
Collecting PyQt5-Qt5>=5.15.2
  Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
    |██████████████████| 50.1 MB 1.1 MB/s
Installing collected packages: PyQt5-sip, PyQt5-Qt5, pyqt5
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.9.0 pyqt5-5.15.5
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\DELL\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip' command.
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



#### Bibliothèques graphiques

- Tkinter est un module intégré à Python pour développer des applications graphiques.
- Ce module se base sur la bibliothèque graphique Tcl/Tk.
- Pour installer Tk, il faut taper la commande :

```
py -m pip install tk
```

```
PS C:\Users\DELL> pip install tk
Collecting tk
  Downloading tk-0.1.0-py3-none-any.whl (3.9 kB)
Installing collected packages: tk
Successfully installed tk-0.1.0
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\DELL\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip' command.
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Tkinter

- Le programme ci-dessous montre le principe de base de tkinter

Importation du module tkinter

```
import tkinter as tk
```

Création d'un nouvel objet Tk. Cet objet représente la fenêtre principale de l'application graphique

```
app = tk.Tk()
```

```
message = tk.Label(app, text="Bonjour le monde")  
message.pack()
```

Création d'un composant graphique de type Label. Notez que l'on passe l'objet app comme premier paramètre de construction pour indiquer qu'il appartient à la fenêtre principale

```
app.mainloop()
```

Pack permet de calculer la taille du composant à l'écran

mainloop() affiche la fenêtre et lance la boucle d'événements



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Tkinter - Lancer une fonction lorsqu'un bouton est

```
import tkinter
root = tkinter.Tk ()
b = tkinter.Button (text = "fonction change_legende")
b.pack ()

def change_legende () :
    global b
    b.config (text = "nouvelle légende")

b.config (command = change_legende)
root.mainloop ()
```

Créer un bouton dont l'identificateur est b.  
Il a pour intitulé fonction change\_legende

Fonction change\_legende qui change la  
légende de ce bouton

Associer au bouton b la  
fonction change\_legende qui est alors  
appelée lorsque le bouton est pressé



Afficher la fenêtre principale et lance  
l'application

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Tkinter - Associer un événement à un objet

- Il est possible de faire en sorte que le programme exécute une fonction au moindre déplacement de la souris, à la pression d'une touche.
- Il est possible d'associer une fonction au moindre événement susceptible d'être intercepté par l'interface graphique.
- Un événement est décrit par la classe **Event** dont les attributs listés par la table suivante décrivent l'événement qui sera la plupart du temps la pression d'une touche du clavier ou le mouvement de la souris.

<code>char</code>	Lorsqu'une touche a été pressée, cet attribut contient son code, il ne tient pas compte des touches dites muettes comme les touches <code>shift</code> , <code>ctrl</code> , <code>alt</code> . Il tient pas compte non plus des touches <code>return</code> ou <code>suppr</code> .
<code>keysym</code>	Lorsqu'une touche a été pressée, cet attribut contient son code, quelque soit la touche, muette ou non.
<code>num</code>	Contient un identificateur de l'objet ayant reçu l'événement.
<code>x,y</code>	Coordonnées relatives de la souris par rapport au coin supérieur gauche de l'objet ayant reçu l'événement.
<code>x_root, y_root</code>	Coordonnées absolues de la souris par rapport au coin supérieur gauche de l'écran.
<code>widget</code>	Identifiant permettant d'accéder à l'objet ayant reçu l'événement.

- La liste complète est accessible avec l'instruction suivante :

```
import tkinter
help(tkinter.Event)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Tkinter - Associer un événement à un objet

- La méthode **bind** permet d'exécuter une fonction lorsqu'un certain événement donné est intercepté par un objet donné.
- La fonction exécutée accepte un seul paramètre de type Event qui est l'événement qui l'a déclenchée. Cette méthode a pour syntaxe :

```
w.bind(ev, fonction)
```

- **w** est l'identificateur de l'objet devant intercepter l'événement désigné par la chaîne de caractères **ev** dont les valeurs possibles sont décrites ci-dessous.
- **fonction** est la fonction qui est appelée lorsque l'événement survient. Cette fonction ne prend qu'un paramètre de type Event.

<Key>	Intercepter la pression de n'importe quelle touche du clavier.
<Button-i>	Intercepter la pression d'un bouton de la souris. <b>i</b> doit être remplacé par 1,2,3.
<ButtonRelease-i>	Intercepter le relâchement d'un bouton de la souris. <b>i</b> doit être remplacé par 1,2,3.
<Double-Button-i>	Intercepter la double pression d'un bouton de la souris. <b>i</b> doit être remplacé par 1,2,3.
<Motion>	Intercepter le mouvement de la souris, dès que le curseur bouge, la fonction liée à l'événement est appelée.
<Enter>	Intercepter un événement correspondant au fait que le curseur de la souris entre la zone graphique de l'objet.
<Leave>	Intercepter un événement correspondant au fait que le curseur de la souris sorte la zone graphique de l'objet.

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

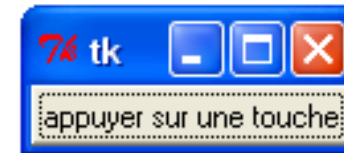
- L'exemple suivant utilise la méthode **bind** pour que le seul bouton de la fenêtre intercepte toute pression d'une touche, tout mouvement et toute pression du premier bouton de la souris lorsque le curseur est au-dessus de la zone graphique du bouton.

```
import tkinter
root = tkinter.Tk()
b = tkinter.Button(text="appuyer sur une touche")
b.pack()

def affiche_touche_pressee (evt) :
    print("----- touche pressee")
    print("evt.char = ", evt.char)
    print("evt.keysym = ", evt.keysym)
    print("evt.num = ", evt.num)
    print("evt.x,evt.y = ", evt.x, ",", evt.y)
    print("evt.x_root,evt.y_root = ", evt.x_root, ",", evt.y_root)
    print("evt.widget = ", evt.widget)

b.bind("<Key>", affiche_touche_pressee)
b.bind("<Button-1>", affiche_touche_pressee)
b.bind("<Motion>", affiche_touche_pressee)
b.focus_set ()

root.mainloop ()
```



**focus\_set** stipule que le bouton doit recevoir le focus. C'est-à-dire que cet objet est celui qui peut intercepter les événements liés au clavier.

## 02 - Manipuler les bibliothèques

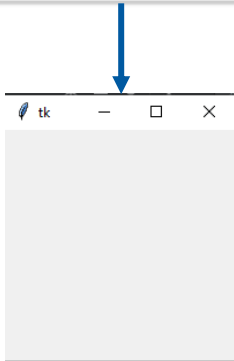
### Installation des bibliothèques externes (pip)

### Composants graphiques (Widgets)

#### Fenêtre principale

- Les interfaces graphiques sont composées d'objets ou widgets ou contrôles.
- Les différents objets sont disposés dans une fenêtre. Pour afficher cette fenêtre, il suffit d'ajouter au programme les deux lignes suivantes :

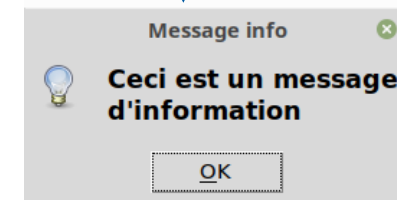
```
root = tkinter.Tk ()  
# ici, on trouve le code qui définit les objets  
# et leur positionnement  
root.mainloop ()
```



#### Boîte de message

- Tkinter fournit des fonctions simples pour afficher des boîtes de message à l'utilisateur. Ces fonctions prennent comme premier paramètre le titre de la fenêtre de dialogue et comme second paramètre le message à afficher.
  - **Message d'information**

```
from tkinter import messagebox  
  
messagebox.showinfo("Message info", "Ceci est un message d'information")
```



## 02 - Manipuler les bibliothèques

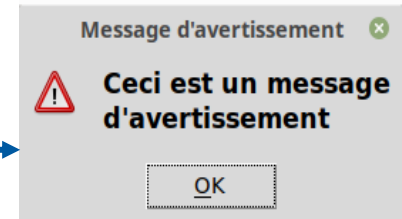
### Installation des bibliothèques externes (pip)

### Composants graphiques (Widgets)

#### Boite de message

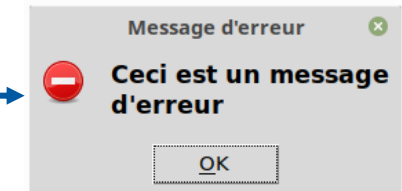
- Message d'avertissement

```
from tkinter import messagebox  
messagebox.showwarning("Message d'avertissement", "Ceci est un message d'avertissement")
```



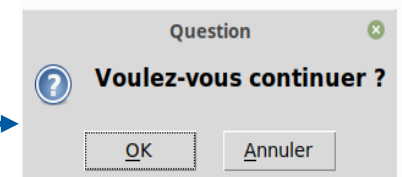
- Message d'erreur

```
from tkinter import messagebox  
messagebox.showerror("Message d'erreur", "Ceci est un message d'erreur")
```



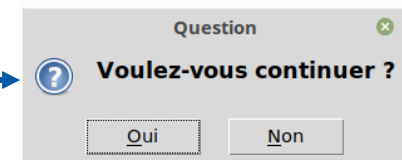
- Question à réponse ok / annuler

```
from tkinter import messagebox  
reponse = messagebox.askokcancel("Question", "Voulez-vous continuer ?")
```



- Question à réponse oui / non

```
from tkinter import messagebox  
reponse = messagebox.askyesno("Question", "Voulez-vous continuer ?")
```



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

### Composants graphiques (Widgets)

#### Zone de texte

- Une zone de texte sert à insérer dans une fenêtre graphique une légende indiquant ce qu'il faut insérer dans une zone de saisie voisine
- Pour créer une zone de texte, il suffit d'écrire la ligne suivante :

```
zone_texte = tkinter.Label (text = "zone de texte")
```



zone de texte

- Il est possible que le texte de cette zone de texte doive changer après quelques temps. Dans ce cas, il faut appeler la méthode config comme suit :

```
zone_texte = tkinter.Label (text = "premier texte")  
# ...  
# pour changer de texte  
zone_texte.config (text = "second texte")
```

Une zone de texte peut être à l'état **DISABLED**

- Pour obtenir cet état, il suffit d'utiliser l'instruction suivante : `zone_texte.config (state = tkinter.DISABLED)`

- Et pour revenir à un état normal : `zone_texte.config (state = tkinter.NORMAL)`

Ces deux dernières options sont communes à tous les objets d'une interface graphique

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



### Composants graphiques (Widgets)

#### Bouton

- Un bouton a pour but de faire le lien entre une fonction et un clic de souris. Un bouton correspond à la classe Button. Pour créer un bouton, il suffit d'écrire la ligne suivante :

```
bouton = tkinter.Button (text = "zone de texte")
```

- Il est possible que le texte de ce bouton doive changer après quelques temps. Dans ce cas, il faut appeler la méthode config comme suit :

```
bouton = tkinter.Button (text = "premier texte")  
# ...  
# pour changer de texte  
bouton.config (text = "second texte")
```



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



### Composants graphiques (Widgets)

#### Zone de saisie

- Une zone de saisie a pour but de recevoir une information entrée par l'utilisateur.
- Une zone de saisie correspond à la classe Entry ; pour en créer une, il suffit d'écrire la ligne suivante :

```
saisie = tkinter.Entry ()
```

- Pour modifier le contenu de la zone de saisie, il faut utiliser la méthode insert qui insère un texte à une position donnée.

```
# Le premier paramètre est la position  
# où insérer le texte (second paramètre)  
saisie.insert (pos, "contenu")
```

- Pour obtenir le contenu de la zone de saisie, il faut utiliser la méthode **get** :

```
contenu = saisie.get ()
```

- Pour supprimer le contenu de la zone de saisie, il faut utiliser la méthode delete. Cette méthode supprime le texte entre deux positions.

```
# supprime le texte entre les positions pos1, pos2  
saisie.delete (pos1, pos2)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

## Composants graphiques (Widgets)

### Case à cocher

- Une case à cocher correspond à la classe Checkbutton. Pour créer une case à cocher, il suffit d'écrire la ligne suivante :

```
# crée un objet entier pour récupérer la valeur de la case à cocher,  
# 0 pour non cochée, 1 pour cochée  
v = tkinter.IntVar ()  
case = tkinter.Checkbutton (variable = v)
```



- Pour savoir si la case est cochée ou non, il suffit d'exécuter l'instruction :

```
v.get () # égal à 1 si la case est cochée, 0 sinon
```

- Pour cocher et décocher la case, il faut utiliser les instructions suivantes :

```
case.select () # pour cocher  
case.deselect () # pour décocher
```

- Il est possible d'associer du texte à l'objet case à cocher :

```
case.config (text = "case à cocher")
```

```
# crée un objet entier partagé pour récupérer le numéro du bouton radio activé  
v = tkinter.IntVar ()  
case1 = tkinter.Radiobutton (variable = v, value = 10)  
case2 = tkinter.Radiobutton (variable = v, value = 20)  
case3 = tkinter.Radiobutton (variable = v, value = 30)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Bouton radio

- Un bouton radio correspond à la classe **Radiobutton**.
- Elles fonctionnent de manière semblable à des cases à cocher excepté le fait qu'elles n'apparaissent jamais seules : elles fonctionnent en groupe. Pour créer un groupe de trois cases rondes, il suffit d'écrire la ligne suivante :

```
# crée un objet entier partagé pour récupérer le numéro du bouton radio activé  
v = tkinter.IntVar ()  
case1 = tkinter.Radiobutton (variable = v, value = 10)  
case2 = tkinter.Radiobutton (variable = v, value = 20)  
case3 = tkinter.Radiobutton (variable = v, value = 30)
```

- La variable v est partagée par les trois cases rondes. L'option value du constructeur permet d'associer un bouton radio à une valeur de v.
  - Si v == 10, seul le premier bouton radio sera sélectionné.
  - Si v == 20, seul le second bouton radio le sera.
- Pour savoir quel bouton radio est coché ou non, il suffit d'exécuter l'instruction :

```
v.get () # retourne le numéro du bouton radio coché (ici, 10, 20 ou 30)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)


#### Bouton radio

- Pour cocher un des boutons radio, il faut utiliser l'instruction suivante :

```
v.set (numero) # numéro du bouton radio à cocher  
                # pour cet exemple, 10, 20 ou 30
```

- Il est possible d'associer du texte à un bouton radio :

```
case1.config (text = "premier bouton")  
case2.config (text = "second bouton")  
case3.config (text = "troisième bouton")
```



☒ premier bouton  
☐ second bouton  
☐ troisième bouton

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Liste

- Un objet liste contient une liste d'intitulés qu'il est possible de sélectionner. Une liste correspond à la classe `ListBox`. Pour la créer, il suffit d'écrire la ligne suivante :

```
li = tkinter.Listbox ()
```

- Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions de la liste  
# width <--> largeur  
# height <--> hauteur en lignes  
li.config (width = 10, height = 5)
```



première ligne

- On peut insérer un élément dans la liste avec la méthode `insert` :

```
pos = 0 # un entier, "end" ou tkinter.END pour insérer ce mot à la fin  
li.insert (pos, "première ligne")
```

- Les intitulés de cette liste peuvent ou non être sélectionnés. Cliquer sur un intitulé le sélectionne mais la méthode `select_set` permet aussi de le faire.

```
pos1 = 0  
li.select_set (pos1, pos2 = None)  
# sélectionne tous les éléments entre les indices pos1 et  
# pos2 inclus ou seulement celui d'indice pos1 si pos2 == None
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Liste

- La méthode **curselection** permet d'obtenir la liste des indices des éléments sélectionnés.

```
sel = li.curselection ()
```

- La méthode **get** permet de récupérer un élément de la liste tandis que la méthode `codes{size}` retourne le nombre d'éléments.

```
for i in range (0, li.size()):  
    print(li.get (i))
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)



#### Tableau (TreeView)

- Le widget Treeview permet d'afficher les données en lignes et en colonnes. La création d'un TreeView doit suivre les étapes suivantes

##### Etape 1 : Création du treeView

- Columns=ac définit les nom des colonnes, show='headings' signifie que la première ligne est l'en-tête du tableau, height=7 signifie que le tableau est de 7 lignes

```
tv=ttk.Treeview(root,columns=ac,show='headings',height=7)
```

##### Etape 2 : Définir les propriétés des colonnes

- a[i] est le nom de la colonne, width=70 est la taille de la colonne, anchor='e' est le type de l'alignement

```
tv.column(ac[i],width=70,anchor='e')
```

##### Etape 3 : Définition de la première ligne du tableau

- Text= area[i] présente le contenu de chaque colonne de la première ligne

```
tv.heading(ac[i],text=area[i])
```

##### Etape 4 : Remplissage du tableau

- Values=sales\_data[i] définit le contenu de chaque ligne du tableau

```
tv.insert('', 'end', values=sales_data[i])
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Tableau (TreeView)

Exemple :

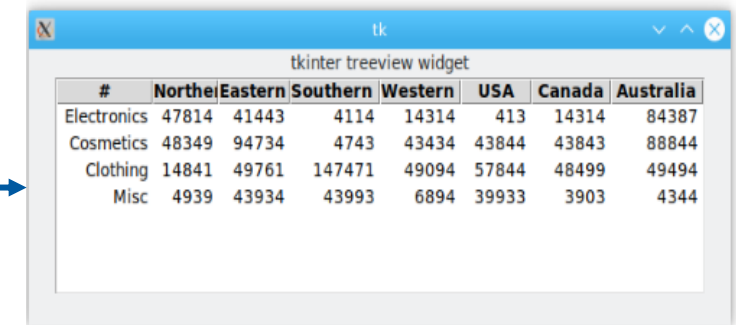
```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.geometry('320x240')
tk.Label(root, text='tkinter treeview widget').pack()
area=(' #', 'Northern Europe', 'Eastern Europe', 'Southern Europe', 'Western Europe', 'USA', 'Canada', 'Australia')

ac=('all', 'n', 'e', 's', 'ne', 'nw', 'sw', 'c')
sales_data=[('Electronics', '47814', '41443', '4114', '14314', '413', '14314', '84387'),
             ('Cosmetics', '48349', '94734', '4743', '43434', '43844', '43843', '88844'),
             ('Clothing', '14841', '49761', '147471', '49094', '57844', '48499', '49494'),
             ('Misc', '4939', '43934', '43993', '6894', '39933', '3903', '4344')]

tv=ttk.Treeview(root, columns=ac, show='headings', height=7)
for i in range(8):
    tv.column(ac[i], width=70, anchor='e')
    tv.heading(ac[i], text=area[i])
tv.pack()

for i in range(4):
    tv.insert('', 'end', values=sales_data[i])
root.mainloop()
```



#	Northern Europe	Eastern Europe	Southern Europe	Western Europe	USA	Canada	Australia
Electronics	47814	41443	4114	14314	413	14314	84387
Cosmetics	48349	94734	4743	43434	43844	43843	88844
Clothing	14841	49761	147471	49094	57844	48499	49494
Misc	4939	43934	43993	6894	39933	3903	4344



## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

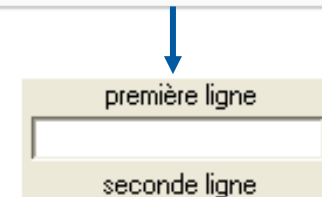
#### Disposition des objets dans une fenêtre

- Chacun des objets (ou widgets) présentés au paragraphe précédent possède trois méthodes qui permettent de déterminer sa position dans une fenêtre : pack, grid, place.
- **pack, grid** : permettent de disposer les objets sans se soucier ni de leur dimension ni de leur position. La fenêtre gère cela automatiquement.
- **Place** : place les objets dans une fenêtre à l'aide de coordonnées sans utiliser l'aide d'aucune grille.

#### Méthode pack

- Cette méthode empile les objets les uns à la suite des autres. Par défaut, elle les empile les uns en-dessous des autres. Par exemple, l'exemple suivant produit l'empilement des objets :

```
l = tkinter.Label (text = "première ligne")
l.pack ()
s = tkinter.Entry ()
s.pack ()
e = tkinter.Label (text = "seconde ligne")
e.pack ()
```



- Les objets sont empilés à l'aide de la méthode pack les uns en-dessous des autres.
- On peut aussi les empiler les uns à droite des autres grâce à l'option side :

```
l = tkinter.Label (text = "première ligne")
l.pack (side = tkinter.RIGHT)
s = tkinter.Entry ()
s.pack (side = tkinter.RIGHT)
e = tkinter.Label (text = "seconde ligne")
e.pack (side = tkinter.RIGHT)
```

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Méthode grid

- La méthode grid suppose que la fenêtre qui les contient est organisée selon une grille dont chaque case peut recevoir un objet. L'exemple suivant place trois objets dans les cases de coordonnées (0,0), (1,0) et (0,1).

```
l = tkinter.Label (text = "première ligne")  
l.grid (column = 0, row = 0)  
s = tkinter.Entry ()  
s.grid (column = 0, row = 1)  
e = tkinter.Label (text = "seconde ligne")  
e.grid (column = 1, row = 0)
```



- La méthode grid possède plusieurs options telles que:
  - column** : colonne dans laquelle sera placé l'objet.
  - columnspan** : nombre de colonnes que doit occuper l'objet.
  - row** : ligne dans laquelle sera placé l'objet.
  - rowspan** : nombre de lignes que doit occuper l'objet.

## 02 - Manipuler les bibliothèques

### Installation des bibliothèques externes (pip)

#### Méthode place

- La méthode place est sans doute la plus simple à comprendre puisqu'elle permet de placer chaque objet à une position définie par des coordonnées :

```
l = tkinter.Label(text="première ligne")  
l.place (x=10, y=50)
```

# CHAPITRE 2

## Manipuler les bibliothèques

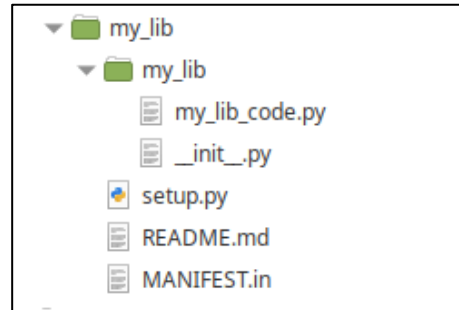
1. Installation des bibliothèques externes (pip)
2. **Création des bibliothèques**
3. Importation des bibliothèques



## 02 - Manipuler les bibliothèques

### Création des bibliothèques

- Parfois les bibliothèques standard ne suffisent pas et on espère créer une bibliothèque contenant des fonctions spécifiques.
- Pour mettre en place une bibliothèque, le projet doit respecter la structure suivante :



- Le dossier **`my_lib`**, qui contient l'ensemble de code de notre librairie, ainsi que le fichier **`__init__.py`** permettant à Python de considérer le dossier comme contenant des paquets.
- Le fichier **`README.md`** qui contient la description complète de la librairie
- Le fichier **`MANIFEST.in`** qui liste tous les fichiers non-Python de la librairie
- Le fichier **`setup.py`** qui contient la fonction `setup` permettant l'installation de la librairie sur le système

## 02 - Manipuler les bibliothèques

### Création des bibliothèques



#### Le fichier setup.py

- Ce fichier contient la **méthode setup** qui permettra l'installation de la librairie sur le système.
- La fonction **setup** peut prendre en argument une trentaine d'argument, mais voici dans l'exemple ceux qui seront le plus souvent utilisés.
- Les premiers paramètres de cette méthode sont surtout des paramètres descriptifs de la librairie, de son auteur, sa licence, sa version, etc.

```
from setuptools import setup

setup(
    name='my_lib',
    version='1.0.0',
    author='Author Name',
    author_email='author.name@mail.com',
    description='Courte description de la librairie',
    license='Other/Proprietary License',
    keywords='lib',
    url='Lien vers la page officielle de la librairie',
    packages=[
        'my_lib'
    ],
    long_description=open('README.md').read(),
    classifiers=[
        'Development Status :: 4 - Beta',
        'Intended Audience :: Developers',
        'License :: Other/Proprietary License',
        'Natural Language :: French',
        'Operating System :: OS Independent',
        'Programming Language :: Python :: 3.7',
        'Topic :: Software Development',
        'Topic :: Software Development :: Libraries :: Python Modules',
        'Topic :: Utilities'
    ],
    install_requires=[
        'pymongo==3.7.2'
    ],
    include_package_data=True
)
```

## 02 - Manipuler les bibliothèques

### Création des bibliothèques



- Concernant les autres paramètres :
  - **package** : contient la liste de tous les packages de la librairie qui seront insérés dans la distribution
  - **classifier** : méta donnée permettant aux robots de correctement classer la librairie
  - **instal\_requires** : cette partie contient toutes les dépendances nécessaires au bon fonctionnement de votre librairie
  - **include\_package\_data** : permet d'activer la prise en charge du fichier MANIFEST.in
- Une fois l'ensemble de ces éléments défini, la librairie peut être installée sur le système en utilisant la commande suivante (exécuter la commande dans le répertoire du projet) :

```
py setup.py install
```

## 02 - Manipuler les bibliothèques

### Création des bibliothèques

#### Exemple :

- Création de la bibliothèque

↑ Ce PC > Disque local (E:) > Nouvelle\_Bib

Nom	Modifié le	Type	Taille
Nouvelle_Bib	2021-10-25 12:42	Dossier de fichiers	
setup	2021-10-25 12:43	Python File	1 Ko

↑ Ce PC > Disque local (E:) > Nouvelle\_Bib > Nouvelle\_Bib

Nom	Modifié le	Type	Taille
__init__	2021-10-23 7:59	Python File	1 Ko
bonjour	2021-10-25 11:44	Python File	1 Ko

```
from setuptools import setup

setup(
    name="Nouvelle_Bib",
    version="0.0.1",
    author="authorX",
    author_email="authorX@yahoo.fr",
    description="courte description de la librairie",
    url="lien vers la page officielle de la librairie",
    licence='Other/Propertary Licence',
    packages={
        'Nouvelle_Bib',
    },
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    install_requires=[
        ],
    include_package_data=True
)
```

```
class Bonjour:
    def __init__(self,nom):
        self.nom=nom
    def affiche(self):
        print("Bonjour:"+self.nom)
```



## 02 - Manipuler les bibliothèques

### Création des bibliothèques

#### Installation de la bibliothèque

- Exécution de la commande :

**py setup.py install**

```
PS E:\nouvelle_Bib> py setup.py install
C:\Users\DELL\AppData\Local\Programs\Python\Python39\lib\distutils\dist.py:259: UserWarning: 'licence' distribution option is deprecated; use 'license'
  warnings.warn(msg)
running install
running bdist_egg
running egg_info
creating Nouvelle_Bib.egg-info
writing Nouvelle_Bib.egg-info\PKG-INFO
writing dependency_links to Nouvelle_Bib.egg-info\dependency_links.txt
writing top-level names to Nouvelle_Bib.egg-info\top_level.txt
writing manifest file 'Nouvelle_Bib.egg-info\SOURCES.txt'
reading manifest file 'Nouvelle_Bib.egg-info\SOURCES.txt'
writing manifest file 'Nouvelle_Bib.egg-info\SOURCES.txt'
installing library code to build\bdist.win-amd64\egg
running install_lib
running build_py
creating build
creating build\lib
creating build\lib\Nouvelle_Bib
copying Nouvelle_Bib\bonjour.py -> build\lib\Nouvelle_Bib
copying Nouvelle_Bib\__init__.py -> build\lib\Nouvelle_Bib
creating build\bdist.win-amd64
creating build\bdist.win-amd64\egg
creating build\bdist.win-amd64\egg\Nouvelle_Bib
copying build\lib\Nouvelle_Bib\bonjour.py -> build\bdist.win-amd64\egg\Nouvelle_Bib
copying build\lib\Nouvelle_Bib\__init__.py -> build\bdist.win-amd64\egg\Nouvelle_Bib
byte-compiling build\bdist.win-amd64\egg\Nouvelle_Bib\bonjour.py to bonjour.cpython-39.pyc
byte-compiling build\bdist.win-amd64\egg\Nouvelle_Bib\__init__.py to __init__.cpython-39.pyc
creating build\bdist.win-amd64\egg\EGG-INFO
copying Nouvelle_Bib.egg-info\PKG-INFO -> build\bdist.win-amd64\egg\EGG-INFO
copying Nouvelle_Bib.egg-info\SOURCES.txt -> build\bdist.win-amd64\egg\EGG-INFO
copying Nouvelle_Bib.egg-info\dependency_links.txt -> build\bdist.win-amd64\egg\EGG-INFO
copying Nouvelle_Bib.egg-info\top_level.txt -> build\bdist.win-amd64\egg\EGG-INFO
zip_safe flag not set; analyzing archive contents...
creating dist
creating 'dist\Nouvelle_Bib-0.0.1-py3.9.egg' and adding 'build\bdist.win-amd64\egg' to it
removing 'build\bdist.win-amd64\egg' (and everything under it)
Processing Nouvelle_Bib-0.0.1-py3.9.egg
Copying Nouvelle_Bib-0.0.1-py3.9.egg to c:\users\dell\appdata\local\programs\python\python39\lib\site-packages
Adding Nouvelle_Bib 0.0.1 to easy-install.pth file

Installed c:\users\dell\appdata\local\programs\python\python39\lib\site-packages\nouvelle_bib-0.0.1-py3.9.egg
```

**Bibliothèque bien installée!!**

## 02 - Manipuler les bibliothèques

### Création des bibliothèques

- L'installation de la bibliothèque entraîne la création d'un fichier : Nouvelle\_Bib-0.0.1-py3.9.egg dans la dossier \Lib\site-packages de python

```
Installed c:\users\dell\appdata\local\programs\python\python39\lib\site-packages\nouvelle_bib-0.0.1-py3.9.egg
```

- Utilisation de la bibliothèque créée

```
>>> from Nouvelle_Bib import bonjour
>>> e=bonjour.Bonjour("Meriam")
>>> e.affiche()
Bonjour:Meriam
```

Importation du module bonjour de la bibliothèque Nouvelle\_Bib

Appel de la classe Bonjour du module bonjour

# CHAPITRE 2

## Manipuler les bibliothèques

1. Installation des bibliothèques externes (pip)
2. Création des bibliothèques
- 3. Importation des bibliothèques**



## 02 - Manipuler les bibliothèques

### Importation des bibliothèques

#### Importer un module

- Pour utiliser un module Python dans la bibliothèque standard de Python ou un module d'une bibliothèque créée, il faut utiliser la syntaxe suivante :

```
import <module>
```

- Par exemple, le module random, on utilise la syntaxe suivante :

```
import random
```

- On identifiera les fonctions importées en préfixant leur nom par celui du module.

Par exemple :

```
>>> random.choice('aeiouy')  
'u'
```

La fonction choice: choisit un élément au hasard dans une liste

## 02 - Manipuler les bibliothèques

### Importation des bibliothèques

#### Importer une fonction particulière d'un module

- Pour importer une fonction d'un module, par exemple la fonction **choice()** du module **random**, on utilise la syntaxe suivante :

```
from random import choice
```

- On identifiera simplement la fonction importée par son nom.

Par exemple :

```
>>> choice('aeiouy')  
'y'
```

- Les autres fonctions du module ne sont pas disponibles :

```
>>> randint(12, 42)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'randint' is not defined
```

## 02 - Manipuler les bibliothèques

### Importation des bibliothèques

#### Importer toutes les fonctions d'un module

- Pour importer l'ensemble des fonctions d'un module.

On utilise alors la syntaxe :

```
from random import *
```

- On identifiera simplement les fonctions importées par leur nom.

Par exemple :

```
>>> choice('aeiouy')  
'o'
```

Et :

```
>>> randint(12, 42)  
17
```