Introduction

à

python™

Octobre 2019

OUTOFPLUTO

Introduction à Python

1. Introduction

L'environnement de développement
Rappels d'algorithmique
La syntaxe
La documentation (docstring)
Les structures de données et de contrôle
Atelier n°1

2. Introduction avancée

Introspection
Fonctions
Portée de variables
string et séquences
Itérateurs et générateurs
Exceptions
Atelier n°2
Expressions régulières
Atelier n°3

3. Environnement

Environnements de développement
Environnements d'exécution
Inclusions et librairies
Atelier n°4
Interaction avec le shell
Atelier n°5

4. Manipulations de données

Les données multidimensionnelles
Fichiers
csv et json
Atelier n°6

Introduction

Les bases

Python est un language de programmation haut niveau, orienté objet, interprété et fortement typé dynamiquement.

Les bases

Python est un language de programmation haut niveau, orienté objet, interprété et fortement typé dynamiquement.

- **haut niveau** indique que la gestion de la mémoire est automatique (notamment grâce à un *garbage collector*).
- **orienté objet** signifie que la base même du langage suit le paradigme objet (tout est objet en Python).
- C'est un langage **interprété.** Il faut un logiciel capable de comprendre le python (l'interpréteur python) pour l'éxecuter à la volée.
- Un langage **fortement typé** ne permet de changer le type (la nature) d'une variable sans changer sa valeur.
- Un langage **typé dynamiquement** permet d'utiliser des données sans indiquer leur type, celui-ci est déterminé contextuellement.

Documentation

https://www.python.org/doc/

Quelques informations supplémentaires

- La **syntaxe** vise à être minimale avec peu de constructions syntaxiques.
- La première version de python a été créée en 1989 par Guido van Rossum. Aujourd'hui, les versions 2.7.x et 3.x sont les plus courantes.
- Python est utilisé dans de très nombreux contextes : recherche, web, automatisation, science des données...

L’environnement de développement

Cas commun

L’installation d’un package environnement python à partir du site officiel est souvent le plus rapide :

- Windows
  https://www.python.org/downloads/windows/
- Mac OS X
  https://www.python.org/downloads/mac-osx/
- Linux :
  gestionnaire de paquets comme apt-get ou yum

Pour exécuter du code python depuis la ligne de commande :

```
$ python monfichier.py
```

Puis un éditeur de texte de votre choix (de préférence avec indentation automatique et coloration

Edition en ligne

Il est possible d'utiliser des interpréteurs en ligne grâce à des outils tels que :

- Jupyter Notebook https://jupyter.org/
- Google Colab https://colab.research.google.com/

L’algorithmique en (très) bref

Définitions

- **Encyclopedia Universalis** Un *algorithme* est la spécification d’un schéma de calcul, sous forme d’une suite [finie] d’opérations élémentaires obéissant à un enchaînement déterminé.
- Une *opération élémentaire* est une étape de calcul définie (non ambiguë) et indivisible en elle-même (ex: une addition)

Analyse et vocabulaire

Pour étudier les algorithmes, plusieurs outils avec leur vocabulaire spécifique ont été formalisés :

- les structures algorithmiques
- la terminaison, la correction et la complétude
- la complexité

Implémentation informatique

Afin d’être exécuté par un ordinateur, un algorithme doit être traduit dans un langage informatique. C’est *l’implémentation* ou le *codage*.
La méthode est différente pour chaque langage.

Définition restrictive pour la formation : Méthode de résolution d’un problème pouvant être réalisée par un ordinateur

Les structures de données

Les données qui sont utilisées dans un algorithme doivent être déclarées avant leur usage.
Cela revient à définir les éléments du problème.

Une déclaration comporte :

- un nom, un mot ou une lettre, généralement explicite (*y* ou *temps* ou *diviseur)*
- un type, qui détermine comment la donnée est utilisable, i.e., quelles opérateurs peuvent lui être appliqués

Les principaux types sont :

- les nombres, sur lesquels il est possible d'appliquer tous les opérateurs arithmétiques
- les chaînes de caractères (*string*), sur lesquels il est possible d'appliquer des opérateurs spécifiques
- les tableaux (*array*) qui sont des regroupements de nombres

Exemple :

nombre Pi = 3,1415

chaine prénom = ”Anthony”

tableau base10 = 1 2 3 4 5 6 7 8 9 0

Les constantes et les variables

Les constantes et les variables sont des *données déclarées*.

Les constantes ont une valeur définie au départ de l'algorithme et ne changeront jamais.

Les variables ont une valeur définie (ou non) au départ de l'algorithme et qui peut changer au cours de l'exécution.

Remarques sur les variables :

Une variable qui n'a pas été affectée à une valeur inconnue.
Une variable peut rester inaffectée jusqu'à la fin d'un algorithme. Dans ce cas, sa valeur est inconnue.
Une variable qui n'est pas affectée durant un algorithme est superflue.
Il peut exister des variables intermédiaires.

Les opérateurs sur les types simples

Nombres

Lecture
Affectation
Opérateurs arithmétiques unaires : Signe, Valeur absolue, Inverse
Opérateurs de test unaires : Est positif, est nul, est négatif
Opérateurs arithmétiques binaires : Addition, Soustraction, Multiplication, Division, Modulo, Racine carrée...
Opérateurs de comparaison binaires : Plus grand que, plus petit que, égal

Chaînes de caractères

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Longueur
(Opérateurs de comparaisons)

Tableaux

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Longueur
(Opérateurs de comparaisons)

Les structures de contrôle

Les structures de contrôle permettent d'organiser l'algorithme, de définir la succession logique des opérations à effectuer.

On distingue trois catégories principales de structures de contrôle :

- Les séquences
- Les expressions conditionnelles
- Les boucles

La notion de *fonction* est également importante (même si elle n'est pas définie comme une structure de contrôle). Elle équivaut au sous-programme des organigrammes.

L’implémentation

- La finitude des moyens

Lors de l’implémentation d’un algorithmique sur un ordinateur, les ressources sont limités : mémoire finie et temps d’éxecution non nul.

- Les langages

L’implémentation dépend grandement du langage : C, python, javascript, PHP. Certains sont plus adaptés que d’autres pour la gestion des données ou pour l’execution de code mathématique. Un bon choix de départ peut être python.

- La syntaxe

Chaque langage a une syntaxe qui lui est propre pour les structures de données, de contrôle, pour les déclarations de variables. Cependant, cela n’a théoriquement que peu d’influence sur la “traduction” de l’algorithme.

- Les structures de données

Selon les langages, les structures de données disponibles peuvent être plus riches que les structures Nombre, Chaîne et Liste.
Typiquement, les nombres sont divisés entre entiers (*integer*) et réels (*float*). Il existe aussi des tableaux associatifs où les indices peuvent être des chaînes de caractères

- Les structures de contrôle

Selon les langages, les structures de contrôle peuvent également être plus riches. Les boucles pour peuvent prendre plusieurs conditions et il existe des syntaxes pour éviter les si/sinon imbriqués (syntaxe *switch case*)

- Les langages non typés

Certains langages (python par exemple), autorise la déclaration de variables sans type. Il faut alors être très rigoureux avec le code.

La syntaxe

L'indentation

La logique des séquences de code en Python est basée sur l'indentation.

Une séquence **doit posséder** une indentation unique.

Contrairement à beaucoup d'autres langages, les séquences ne sont pas délimitées par des accolades.
Une instruction donnant accès à une séquence termine par :

Exemple :

```
i = 0

if i == 0:
  print("nul")
else:
  print("non nul")
```

⚠ L'indentation définit l'appartenance à une séquence et donc le lieu d'éxecution du code

Autres éléments de syntaxe

- `#` pour insérer un commentaire sur une ligne

Structures de données 1/2

Les types de données

- `BooleanType/bool` `True` ou `False`
- `IntType/ int` un nombre entier
- `FloatType / float` un nombre à virgule flottante
- `StringType / str` une chaîne de caractères (délimitée par `"` ou `'`)
- `ListType / list` une série de données de n'importe quel type
- `TypeType / type` une structure contenant des attributs et des méthodes, déclarée avec `class`
- Tous les types de données : https://docs.python.org/2/library/types.html

Les variables

Une variable commence par une lettre ou un _ suivi d'une série de lettres, de chiffres et de _
Le premier caractère **ne peut pas** être un chiffre.
Les noms des variables sont sensibles à la casse.
Les variables sont implicitement typées.

```
ma_chaine = "avec doubles guillements"
maChaine = 'avec simples guillemets'
monEntierQuiVaut3 = 3
print(type(maChaine))
print(type(monEntierQuiVaut3) == int)
```

Les constantes

Il n'y a pas de constante dans le langage Python.
En cas de besoin, deux possibilités s'offrent aux développeurs :

- Les conventions de nommage : Une variable déclarée en majuscule est une constante (aucune garantie)
- L'utilisation d'une librairie, d'un objet ou de code "maison"

```
MA_CONSTANTE = "ma_valeur_de_constante"
```

Structures de données 2/2

Les déclarations de tableaux

```
array = {
    'tomates': 3,
    'carottes': 3,
    'courgettes': 3
}
```

```
array = [ 3, 4, 5 ]
```

Les déclarations d’objets

```
class myClass():
  def
myMethod(self):

print('bonjour')

myObject = myClass()
myObject.myMethod()
```

Opérateurs

Les principaux opérateurs

- `+ - / * %` Les opérateurs arithmétiques
- `== != < > <= >=` Les opérateurs de comparaison
- `+= -=` Les opérateurs d'incrémentation et de décrémentation
- `and or ! (xor)` Les opérateurs logiques
- `+ +=` Le plus sert à concaténer des chaînes de caractères
- `.` La point sert à accéder à une méthode ou un attribut d'un objet
- `[ ]` Les crochets servent à accéder à un élément d'un tableau

La liste complète : https://docs.python.org/2/library/stdtypes.html

Exercices

Etablir le parallèle entre les éléments de syntaxe du Python et les éléments de syntaxe d'un autre language (PHP ou javascript) :

- Les opérateurs d'incrémentation et de décrémentation
- Les opérateurs logiques
- La concaténation
- Les déclarations de tableaux et d'objets
- Les opérateurs sur les tableaux
- L'accesseur `.`

Structures de contrôle

Les séquences

Les séquences constituent la structure de contrôle la plus simple : c'est une succession d'opérations consécutives.

Les conditions

- `if else` — Une alternative, deux possibilités
- `elif / else: if` — Imbrication de tests
- `switch` — Pas de switch en Python
- `break` — Sort du `switch` sans tester les conditions suivantes

Les boucles

- `for` — Boucle de parcours
- `while` — Boucle avec condition d'arrêt

- `break` — Sort de la boucle sans effectuer les itérations suivantes
- `continue` — Passe immédiatement à l'itération suivante de la boucle
- `return` — Sort de la fonction (et de la boucle le cas échéant) en permettant de renvoyer une valeur optionnelle

La liste complète des structures de contrôle : https://docs.python.org/2/reference/compound_stmts.html

Exemple commenté

```
couleurs = {

    '#ff9900': 'orange',
    '#00ff00': 'vert',
    '#ff0000': 'rouge',
    '#ff00ff': 'violet',
    '#0000ff': 'bleu',
    '#000000': 'noir',
    '#ffffff': 'blanc',
    '#ffff00': 'jaune'
}

print('<select name="couleurs">')
for cnt in couleurs:
  print('\t<option value="{}" {} />{}</option>'.
        format(cnt,
               "selected" if couleurs[cnt] == 'rouge' else "",
               couleurs[cnt]))
print('</select>')
```

définition d'un tableau **associatif**
une clé: une valeur

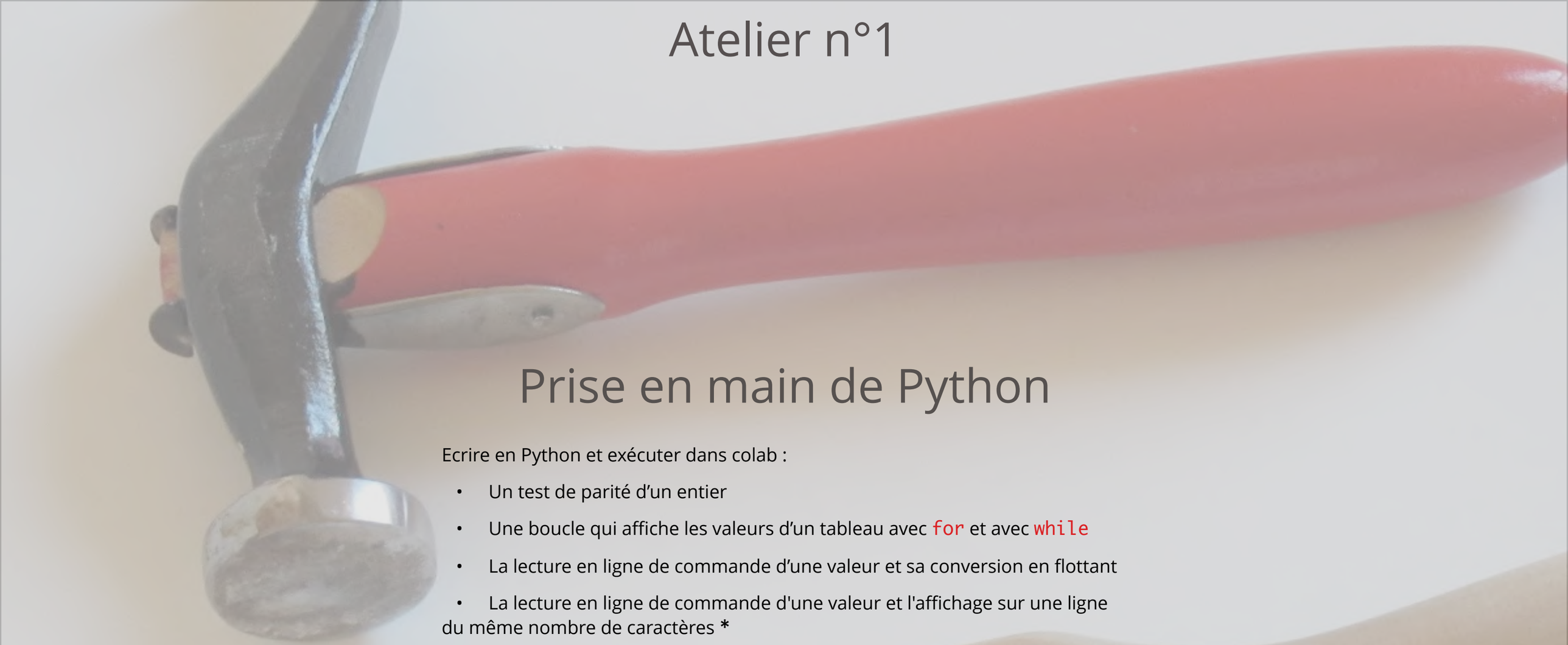
écriture dans la page

boucle sur le tableau

formatage de chaîne de caractères

test et choix d'une valeur par défaut

Atelier n°1



Prise en main de Python

Ecrire en Python et exécuter dans colab :

- Un test de parité d'un entier
- Une boucle qui affiche les valeurs d'un tableau avec `for` et avec `while`
- La lecture en ligne de commande d'une valeur et sa conversion en flottant
- La lecture en ligne de commande d'une valeur et l'affichage sur une ligne du même nombre de caractères *

Aide : opérateur modulo %, fonctions built-in `input()`, `float()` et `print()`

Introduction avancée

Octobre 2019

OUTOFPLUTO

Bonnes pratiques

Les bonnes pratiques et normes de codage sont répertoriées dans des Python Enhancement Proposals (PEP)

Nommages : https://www.python.org/dev/peps/pep-0008/

Documentation : https://www.python.org/dev/peps/pep-0257/

Conventions importantes, en bref :

- opérateurs unaires : collés à l'opérande
- opérateurs binaires : séparés par des espaces
- parenthèses, crochets, etc : collés
- , et ; un espace après seulement
- Nom des classes : *CamelCase*, mots collés avec capitale, par exemple MaClasse
- Variables et fonctions : minuscules, mots séparés par des _, par exemple ma_fonction ou ma_variable
- Attributs et méthodes : comme pour les variables et les méthodes
- Constantes : majuscules, mots séparés par des _, par exemple MA_CONSTANTE

Les commentaires et docstring

docstring

docstring permet de commenter le code : les modules, les classes et les fonctions.
Les commentaires docstring sont visibles notamment grâce à la fonction help()

```
def ma_fonction(mon_parametre1, mon_parametre2):
        """
        function ma_fonction: verbose description

        Parameters
        ----------
        mon_parametre1: string
          description
        mon_parametre2: int
          description

        Return
        -------
        string
          description
        """
        return mon_parametre1 * mon_parametre2

help(maFonction)
```

Introspection

Principe

L'introspection est la capacité de déterminer le type d'un objet lors de l'exécution du code.

En Python, toutes les variables et les types sont des objets. Il est donc possible de faire de l'introspection facilement.

Les fonctions essentielles

- Le type d'une variable `type`
- Liste des méthodes et attributs `dir`

Dans l'interpréteur

La configuration de l'interpréteur python permet de bénéficier de l'introspection en temps réel.

A suivre dans la section 6, le module `inspect`

Les fonctions

Principe

Une fonction permet de définir des séquences hors du flux d'exécution "standard".

```
def ma_fonction(mon_parametre1, mon_parametre2):
        # mon code
        return mon_parametre1 * mon_parametre2
```

Une fonction est définie par sa signature

- Un nom `ma_fonction`
- Des paramètres `mon_parametre1, mon_parametre2`
- Une valeur de retour `mon_parametre1 * mon_parametre2`

Déclaration et appel

Il faut distinguer la déclaration et l'appel de fonction.
La déclaration permet de rendre la fonction disponible dans le reste du code.
L'appel va effectivement exécuter le code de la fonction

Intérêts

Modularité et factorisation du code
Lisibilité

La portée des variables

Variables globales

Une variable globale est une variable définie au niveau 0 d'indentation d'un fichier Python

Variables locales

Une variable locale est une variable définie à un niveau n d'indentation non nul d'un fichier Python

Portée

Une variable globale est utilisable dans toutes les séquences du fichier Python

Une variable locale n'est utilisable que dans le contexte où elle a été déclarée.
Ce contexte peut être une boucle, une fonction, une des alternatives d'une condition.

⚠ La portée d'une variable dépend du lieu de sa déclaration

```
gbl = "global"
print(gbl)

def f():
  lcl = "local"
  gbl = "modifie"
  print(lcl)
  print(gbl)
```

`tuple`, `dict`, `list` et `set`

Tous ces types sont des conteneurs

Un conteneur (container) et une suite de valeurs du même type ou non.

Quelles sont les différences entre ces types ?

`tuple`, `dict`, `list` et `set`

Tous ces types sont des conteneurs

Un conteneur (container) est une suite de valeurs du même type ou non.

Les principales différences

- `tuple` et `list` conteneurs de type séquence
- `dict` et `set` autres conteneurs

- `tuple` séquence immuable
- `dict` dictionnaire de clés / valeurs
- `list` séquence éditable
- `set` jeu de valeurs *hashable*

Un conteneur fournit différentes propriétés et méthodes.

Toute la documentation

https://docs.python.org/2/library/functions.html#tuple
https://docs.python.org/2/library/stdtypes.html#dict
https://docs.python.org/2/library/stdtypes.html#set
https://docs.python.org/2/library/stdtypes.html#typesseq

Les exceptions

Le principe

Les exceptions permettent une gestion des erreurs selon un principe **d'arrêt d'exécution**.
Une exception peut être levée (raised) à tout endroit du code pour signaler une erreur.
Les erreurs levées peut être attrapée (catched) afin de gérer l'exception
Tout erreur non attrapée provoque l'arrêt de l'exécution du code.

La syntaxe

- `try` Début d'un bloc pouvant générer une exception
- `except` Code à exécuter si une exception est levée
- `finally` Code à exécuter qu'il y ait une exception ou non
- `raise` Générer une exception

Exemple

```
printed = False
try:
  print("{:2f}".format("test"))
  printed = True
except Exception as e:
  print("exception was raised")
finally:
  print("nevermind")

if printed == False:
  raise Exception("'test' was not printed")
```

Les itérateurs

Itérateur et itérable

Un itérateur est un objet représentant un flux de données.
Un itérateur dispose de la fonction `next()` qui renvoie l'élément suivant. A la fin du flux, une exception `StopIteration` est levée.

Un itérable est un objet capable de renvoyer ses élements un par un. Toutes les séquences et dict et certains autres types sont iterables.
Un itérateur est itérable. **Un itérable n'est pas nécessairement un itérateur**. Il peut le devenir grâce à la fonction built-on `iter()`

Exemple

```
seq = list([1, 2, 3])
print(hasattr(seq, "__iter__"))
try:
  next(seq)
except:
  print("list is iterable but is not an iterator")
iter_seq = iter(seq)
print(hasattr(iter_seq, "__iter__"))
next(iter_seq)
```

Les générateurs

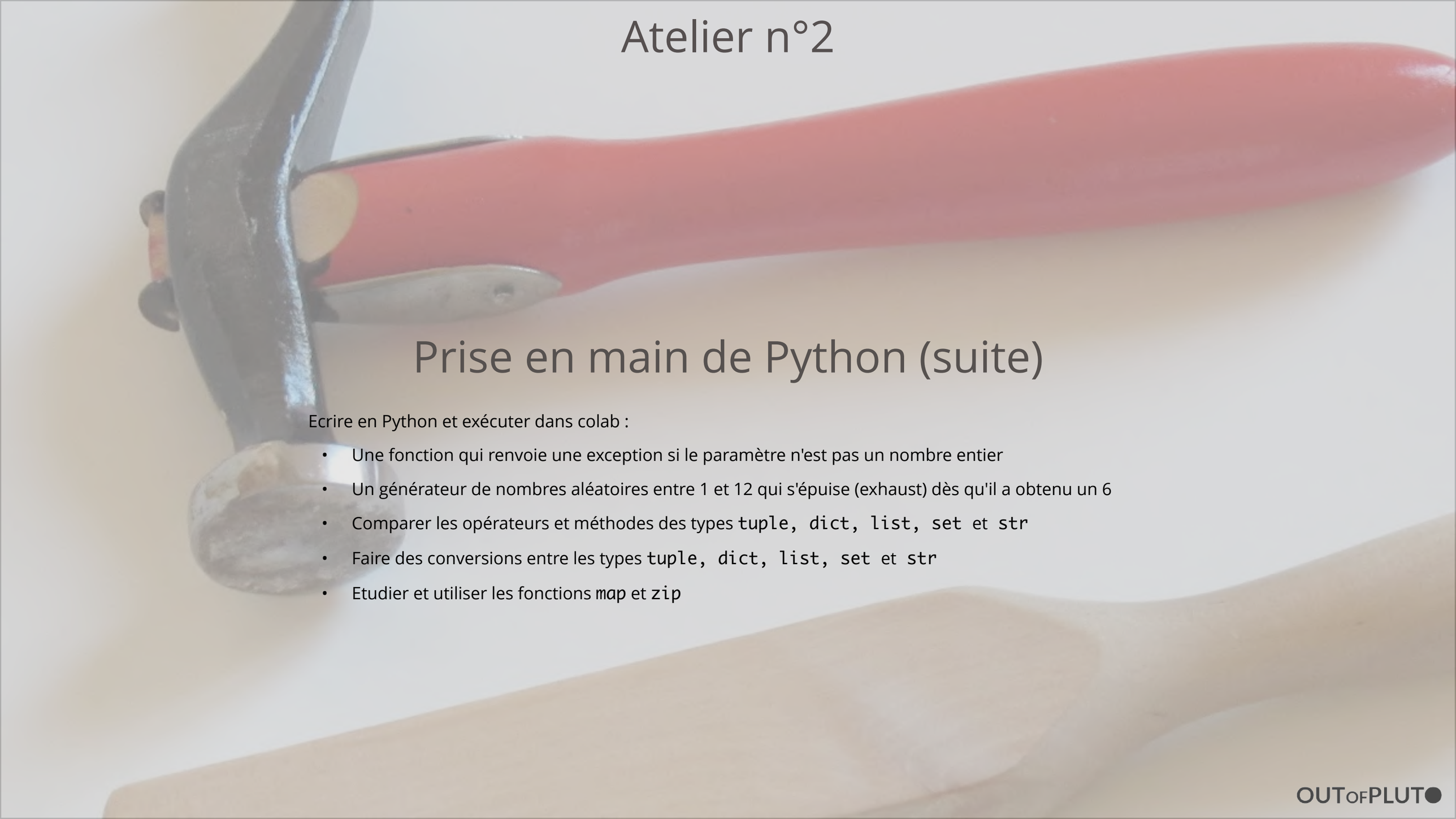
Les générateurs sont des fonctions qui renvoient des itérateurs générateurs

Exemple

```
def get_multiples(number):
    div = number
    while True:
        if number % div == 0:
            yield number
        number += 1

ite_gen = get_multiples(3)
print(next(ite_gen))
print(next(ite_gen))
```

Atelier n°2



Prise en main de Python (suite)

Ecrire en Python et exécuter dans colab :

- Une fonction qui renvoie une exception si le paramètre n'est pas un nombre entier
- Un générateur de nombres aléatoires entre 1 et 12 qui s'épuise (exhaust) dès qu'il a obtenu un 6
- Comparer les opérateurs et méthodes des types `tuple`, `dict`, `list`, `set` et `str`
- Faire des conversions entre les types `tuple`, `dict`, `list`, `set` et `str`
- Etudier et utiliser les fonctions `map` et `zip`

Les `string` et le formatage

Le formatage

Le type `string` fournit la possibilté de réaliser des substitutions complexes de variables et du formatage de valeurs grâce à la méthode `format()`

La substitution de variable s'effectue d'abord puis le formatage de valeurs.
Une mauvaise substitution ou un mauvais formatage lèvera une exception.

⚠ La syntaxe de formatage a changé entre Python 2 et Python 3

Exemples

```
python 2
print "%2f" % 4.34456
print "%s - %s" % ("chaine 1", "chaine 2")
print "%(mot)s %(f).2f %(mot)s" % ({'mot': "PI", 'f': 3.14})
```

```
python 3
print("{:2f}".format(4.34456))
print("{} - {}".format("chaine 1", "chaine 2"))
print("{mot} {f:2f} {mot}".format(mot="PI", f=3.14))
```

Les expressions régulières (en bref)

Une expression régulière est un motif qui permet de décrire un ensemble de chaînes de caractères.

Une expression régulière fonctionne grâce à un système complexe de substitution et de recherche linéaire.

Une expression régulière utilise des caractères spéciaux pour indiquer les motifs à retrouver. Par exemple :

- Des caractères entourés de `[]` constitue un groupe
- Le symbole `|` permet de définir une alternative
- Les symboles `*`, `+` ou `{n}` permet d'indiquer la cardinalité d'un groupe
- Le symbole `.` (point) représente n'importe quel caractère
- Le groupe `[A-Z]` représente toutes les lettres majuscules

Exemples

Quel(s) ensemble(s) de chaînes de caractères ces expressions régulières permettent-elles de décrire ?

```
Josiane|Emile
[A-Za-z]+
[0-9]+\.{0,1}[0-9]*
```

Les expressions régulières en Python

`re`

Le module permettant d'utiliser les expressions régulières en Python s'appelle re (pour regular expression)

https://docs.python.org/3/library/re.html?highlight=re#module-re

Usages

Les expressions régulières ont plusieurs usages différents, par exemple :

- Vérifier qu'une chaîne de caractères respectent certaines contraintes

`re.fullmatch` https://docs.python.org/3/library/re.html?highlight=re#re.Pattern.fullmatch

- Vérifier qu'une chaîne de caractères (ou une chaîne définie par un ensemble) est présente dans une autre

`re.search` https://docs.python.org/3/library/re.html?highlight=re#re.search

`re.findall` https://docs.python.org/3/library/re.html?highlight=re#re.findall

- Remplacer une ou plusieurs occurrences d'une chaîne (ou une chaîne définie par un ensemble) dans une autre

`re.sub` https://docs.python.org/3/library/re.html?highlight=re#re.sub

L'objet Match

Certaines fonctions du module re retourne un objet de type `Match.`

Cette object renvoie de façon organisée le résultat de la fonction, par exemple `match()`

Les intérêts notables de l'objet Match sont :

- La possibilité de ne conserver que certaines parties de l'expression trouvée
- La possibilité de nommer les parties de l'expression trouvée

```
m1 = re.match(r"(\w+) a (\d+) ans", "Joey a 5 ans")
print(m1.group(1))
print(m1.group(2))

m2 = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Ivan Skyvol")
print(m2.group("first_name"))
print(m2.group("last_name"))
```

Atelier n°3

Expressions régulières

Ecrire en Python :

- Une regex pour extraire l'extension d'un nom de fichier (simple)
- Une regex pour valider une adresse email (relativement simple)
- Une regex pour valider uniquement les nombres entre 100 et 140 (assez simple)
- Une regex qui crée un dictionnaire à partir d'une liste de paramètres GET telle que `?name=B;firstname=A;age=10` (moyen)

Les fonctions anonymes

Les fonctions anonymes ou fonctions lambda permettent de créer des fonctions d'une seule expression avec un nombre quelconque de paramètres (y compris aucun).

Exemples

```
mult3 = filter(lambda x: x % 3 == 0, [1, 2, 3, 4, 5, 6, 7, 8, 9])
print(list(mult3))

def add(x):
    return lambda x: x + y
add5 = add(5)
print(add5(3))

student_tuples = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
sorted(student_tuples, key=lambda student: student[2])
```

⚠ Il existe presque toujours une technique plus simple et plus lisible qu'une fonction lambda.

Exercice : trouver des équivalents aux exemples ci-dessus.

Les arguments de fonctions 1/3

Il existe deux types d'arguments en Python :

Les arguments obligatoires (ou positionnels)

Les arguments optionnels

Les règles pour leur déclaration sont les suivantes :

- Les arguments positionnels doivent tous être avant les arguments optionnels

- Les arguments optionnels doivent avoir une valeur par défaut

- Les arguments optionnels peuvent être utilisés comme arguments positionnels, leur ordre reste donc important

Les règles pour leur utilisation lors d'un appel :

- Les arguments positionnels sont donc obligatoires et doivent être dans le "bon" ordre

- Les arguments optionnels peuvent être nommés ou utilisés comme les autres arguments positionnels

- On peut mélanger l'utilisation positionnel/nommé même si cela n'est pas recommandé car

- Un argument optionnel peut être appelé soit en positionnel soit nommé

Les arguments de fonctions 2/3

Exemples :

Déclaration avec les deux types de paramètres :

```
def ma_function(param_1, param_2, option_1=0, option_2=5):
    pass
```

Appels valides :

```
ma_function(1, 2)
ma_function(1, 2, 4)
ma_function(1, 2, 4, 8)
ma_function(1, 2, option_2=7)
ma_function(1, 2, option_2=7, option_1=1)
ma_function(1, 2, 4, option_2=8) # déconseillé
```

Appels invalides :

```
ma_function(option_1=2, 1, 2)
ma_function(1, 2, 4, 7, option_2=8)
```

Les arguments de fonctions 3/3

*args et **kwargs

Les paramètres d'une fonction peuvent être définis dynamiquement grâce à l'utilisation de *args et **kwargs

Dans ce cas :

- *args correspond aux paramètres positionnels et se comporte comme un tuple
- **kwargs correspond aux paramètres nommés et se comporte comme un dictionnaire

Exemple :

```
def ma_function(*args, **kwargs):
    print(args)
    print(kwargs)
```

Appels :

```
ma_function(1, 2)
```

```
args = (1, 2) kwargs = {}
```

```
ma_function(1, 2, option_2=7, option_1=1)
```

```
args = (1, 2) kwargs = {'option_2': 7, 'option_1': 1}
```

Environnement

Quelques environnements d'exécutions

Basique : pip et virtualenv

pip est un gestionnaire d'installation de paquets Python depuis l'index PyPi. virtualenv permet de créer des environnements d'exécution séparés sur un même serveur.

La conjonction de ces deux outils permet de travailler sur plusieurs projets avec différentes versions de Python et de librairies et **d'éviter les conflits.**

- pip
  https://pip.pypa.io/en/stable/installing/
- virtualenv
  https://virtualenv.pypa.io/en/latest/

Plateforme de *Data Science* : Anaconda

https://www.anaconda.com/download/

Anaconda se base sur des outils tels que conda pour la gestion des paquets. C'est une "énorme" distribution Python fournissant à l'installation plus de 200 paquets.

Anaconda inclut pip.

Serveur web

Il est possible d'utiliser du code Python pour servir des pages web en l'interfaçant avec un serveur HTTP. Les deux cas les plus fréquents sont :

- nginx et uwsgi
- Apache avec mod_wsgi

Tutoriel au poil : https://www.digitalocean.com/community/tutorials/how-to-set-up-uwsgi-and-nginx-to-serve-python-apps-on-ubuntu-14-04

Organiser son environnement de développement

Modules

Le code Python peut être organisé en répertoires et sous-répertoires qui seront utilisés comme des modules.

import

Un module accède au code d'un autre module grâce à l'importation. Le plus simple est généralement l'utilisation de la déclaration `import` :

```
import random
from datetime import datetime
from random import randint as get_random_integer
from google import colab
```

Le fichier `__init__.py` joue un rôle particulier dans ce cas, **il est le point d'entrée du module**.

Les autres fichiers peuvent être importés individuellement.

Ils peuvent également être liés (*bound* - *bind*) au module dans `__init__.py`

Il est également possible d'importer seulement une partie des déclarations d'un fichier

⚠ Aux chemins relatifs et aux librairies

Organiser son environnement de développement

Attributs de module

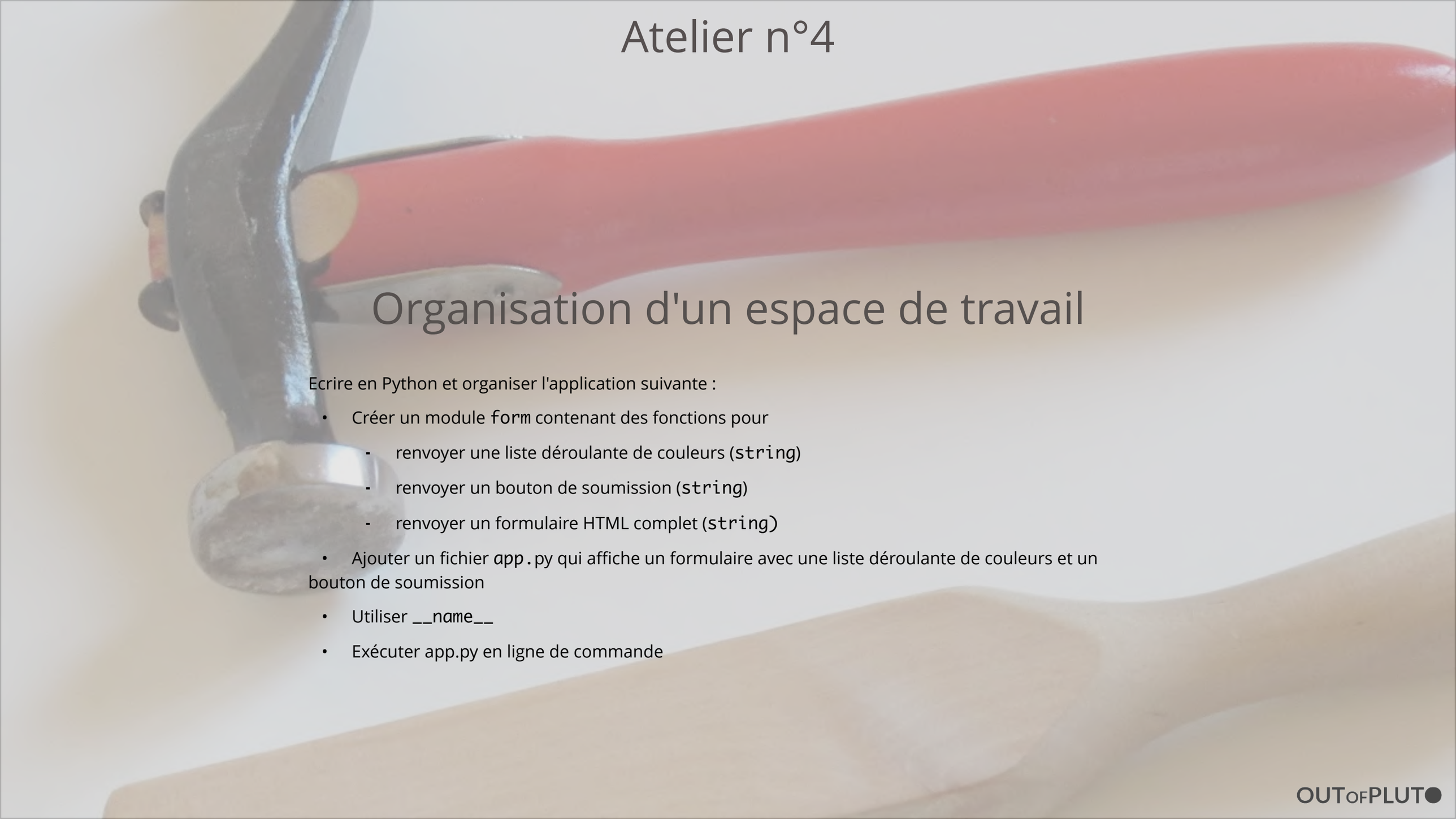
`__name__` `__package__` `__path__` `__file__`

https://docs.python.org/3/reference/import.html?highlight=__name__#import-related-module-attributes

Environnement de script de haut niveau (top-level)

```
if __name__ == "__main__":
    # execute only if run as a script
```

Atelier n°4



Organisation d'un espace de travail

Ecrire en Python et organiser l'application suivante :

- Créer un module `form` contenant des fonctions pour
  - renvoyer une liste déroulante de couleurs (`string`)
  - renvoyer un bouton de soumission (`string`)
  - renvoyer un formulaire HTML complet (`string`)
- Ajouter un fichier `app.py` qui affiche un formulaire avec une liste déroulante de couleurs et un bouton de soumission
- Utiliser `__name__`
- Exécuter app.py en ligne de commande

L'interaction avec le shell 1/2

Python, comme la plupart des interpréteurs peut être executé en ligne de commande (ou shell).

L'interpréteur accepte plusieurs paramètres supposant trois syntaxes principales :

```
$ python
$ python monFichier.py
$ python monFichier.py monParametre1 monParametre2 [...]
```

La première version permet d'accéder à un interpréteur python interactif.
La deuxième permet d'interpréter et exécuter le code du fichier `monFichier.py`
La troisième permet d'interpréter et exécuter le code du fichier `monFichier.py` en lui passant une liste de paramètres.

Question ? Comment récupérer les paramètres dans le code Python ? La réponse avec le module `sys`

L'interaction avec le shell 2/2

En cours d'exécution, Python peut interagir avec le shell via trois *descripteurs de fichier* qui sont :

- La sortie standard ou `stdout`
- La sortie d'erreur ou `stderr`
- L'entrée standard ou `stdin`

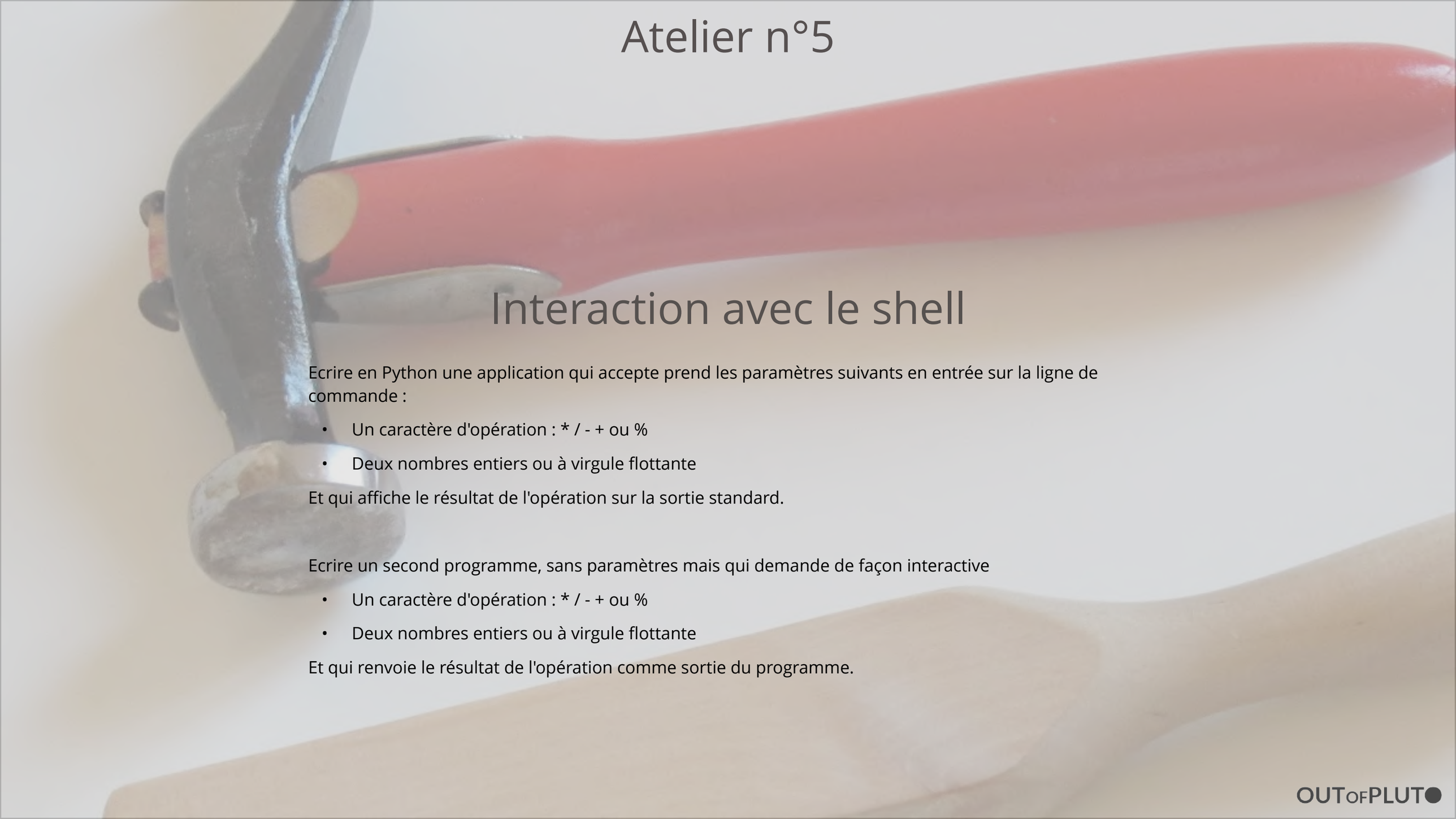
Par défaut, les trois correspondent à la fenêtre du shell mais peuvent être redirigés vers des fichiers.

`stdout` correspond aux affichages, effectués par exemple avec `print()`

`stderr` correspond aux affichages des erreurs, générés par exemple par les exceptions

`stdin` correspond aux interactions d'entrée, provoquées par exemple avec `input()`

Atelier n°5



Interaction avec le shell

Ecrire en Python une application qui accepte prend les paramètres suivants en entrée sur la ligne de commande :

- Un caractère d'opération : * / - + ou %
- Deux nombres entiers ou à virgule flottante

Et qui affiche le résultat de l'opération sur la sortie standard.

Ecrire un second programme, sans paramètres mais qui demande de façon interactive

- Un caractère d'opération : * / - + ou %
- Deux nombres entiers ou à virgule flottante

Et qui renvoie le résultat de l'opération comme sortie du programme.

Import et librairies

Les librairies sont des modules installés dans un(des) répertoire(s) spécifique(s) de l'environnement

Typiquement

<venv>/lib/python<version>/site-packages/

<venv>/lib/python<version>/dist-packages/

En cas de conflit, les fichiers de modules locaux au projet seront utilisés prioritairement sur une librairie.

Explorer

La librairie standard : https://docs.python.org/3/library/index.html

Pillow : https://github.com/python-imaging/Pillow

numpy : http://numpy.scipy.org/

matplotlib : https://matplotlib.org/

Manipulation de données

Manipulation de séquences

Tranchage (*Slicing*)

L'opérateur `[ ]` peut être utilisé en conjonction avec l'opérateur `:` pour couper des tranches dans les séquences

Exemples :

```
ints = [1, 2, 3, 4, 5, 20, 21, 22, 23]
first_two = ints[:2]
last = ints[-1]
all_but_last = ints[:-1]
lower_than_twenty = ints[:ints.index(20)]
2nd_and_3rd = ints[2:3]
```

Concaténation

```
double = ints + ints
```

append, pop et insert

```
ints.pop()
ints.append(31)
ints.insert(5, 15)
```

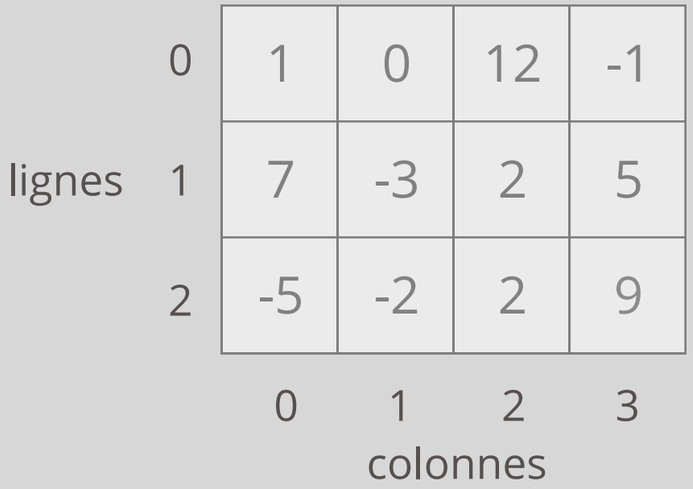
Les tableaux multidimensionnels

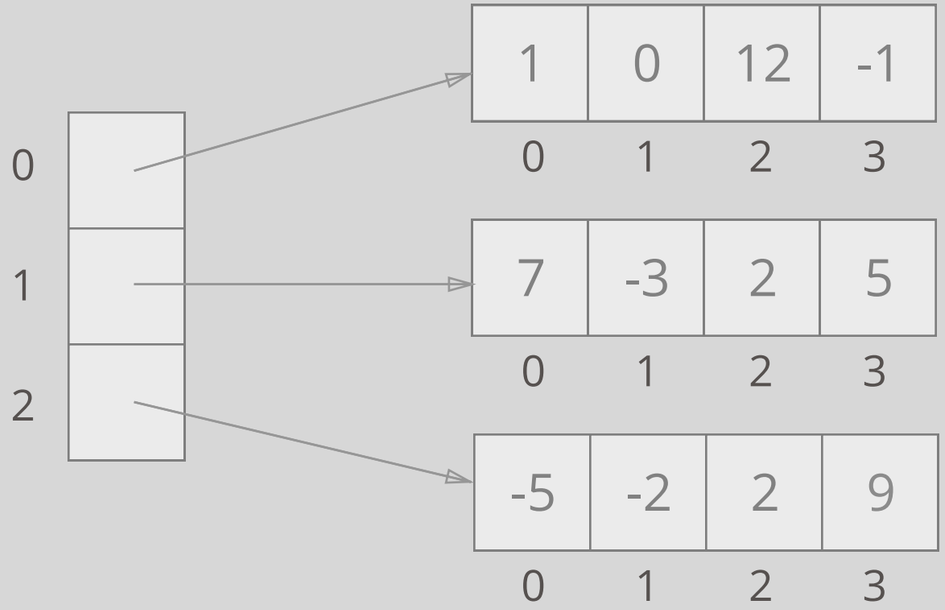
Introduction aux tableaux à plusieurs dimensions

lignes				
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9
	0	1	2	3
	colonnes			

Les tableaux multidimensionnels

Introduction aux tableaux à plusieurs dimensions



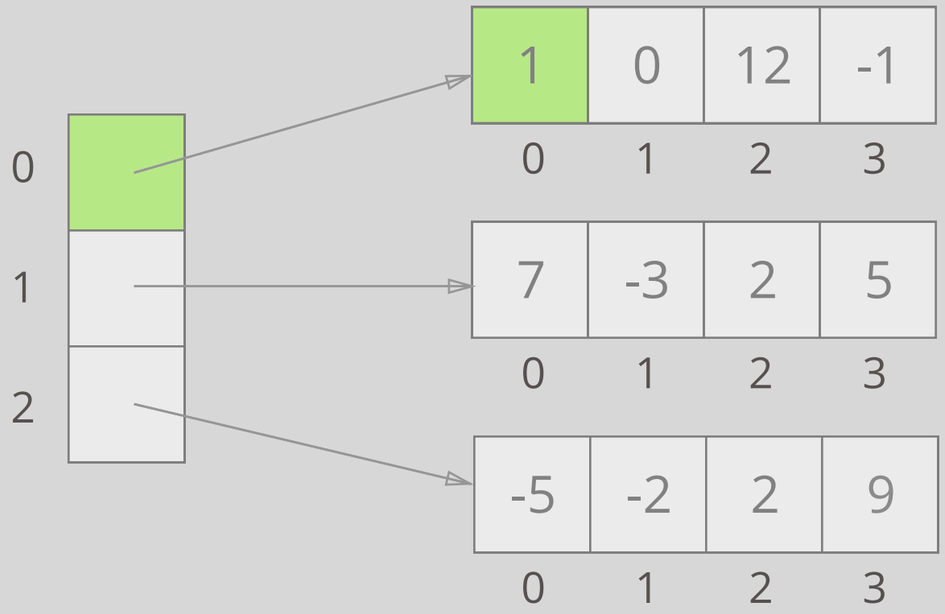


Les tableaux multidimensionnels

Introduction aux tableaux à plusieurs dimensions

lignes	0	1	2	3
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9

colonnes



`tableau[0][0]` vaut 1

Combien vaut `tableau[1][3]` ?
Combien vaut `tableau[2][1]` ?
A quelle(s) position(s) peut-on trouver la valeur 2 ?

Les tableaux multidimensionnels

Représenter les tableaux multidimensionnels avec les tableaux à une dimension

lignes	0	1	2	3
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9

colonnes

Définissons `nombre_colonnes = 4`

Alors

`tableau[1][0] = tableau[1*nombre_colonnes+0] = tableau[4]`

Quel est l'index de `tableau[1][3]` ?

Quelle est la position de `tableau[9]` ?

Quelle est la règle générique pour accéder à un élément ?

Les tableaux multidimensionnels

Représenter les tableaux multidimensionnels avec les tableaux à une dimension

Règles générique :

Définissons `nombre_colonnes = 4`

Pour trouver l'index d'un élément en connaissant sa position :
`index = row*nombre_colonnes+col`

Pour trouver la valeur d'un élément selon sa position dans le tableau :
`tableau[row][col] = tableau[row*nombre_colonnes+col]`

Pour trouver la position d'un élément en connaissant son index :
`col = index % nombre_colonnes`
`row = (index - col) / nombre_colonnes`

lignes		colonnes			
	0	1	0	12	-1
	1	7	-3	2	5
	2	-5	-2	2	9
		0	1	2	3

Manipulation de fichiers

`file object`

Le type built-in `file object` permet d'utiliser le système de fichier.

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

Colab

Il faut préalablement téléverser le fichier (penser à débloquer temporaire les cookies tiers) :

```
from google.colab import files

files.upload()
```

os

La librairie os permet d'interagir notamment avec le système de fichier.

Par exemple :

```
os.lstat(path)
os.listdir(path)
os.mkdir(path)
```

Ouverture / Lecture / Ecriture / Fermeture

Ouverture

`os.open(path, flags)` et `os.fopen(fd)` ou `open(path, mode)`

https://docs.python.org/3/library/os.html?highlight=os%20open#os.open

Les drapeaux (*flags*) vont déterminer les actions qui seront possibles sur le fichier ouvert. Les plus communs sont:

`os.O_RDONLY` ou `'r'` : lecture seule
`os.O_WRONLY` ou `'w'` : écriture seule
`os.O_RDWR` : lecture et écriture
`os.O_APPEND` : ouverture du fichier en mode 'ajout à la fin'
`os.O_CREAT` : création du fichier à l'ouverture
`os.O_EXCL` : en combinaison avec , `os.O_CREAT` ne crée le fichier que s'il n'existe pas
`os.O_TRUNC` : ouverture du fichier existant et suppression de son contenu

Lectures

```
<file object>.read()
<file object>.read(size)
<file object>.readline()
```

Ecritures

```
<file object>.write(str)
```

Fermeture

```
<file object>.close()
```

Manipulation de fichiers csv

csv

Le *comma-separated-values* est un des formats de fichier de données tabulaires les plus simples.
Les lignes sont stockées une par une et les colonnes sont séparées par une virgule (ou parfois un point-virgule) :

```
nom;prenom;age
Lapointe;Bobby;52
Darc;Jeanne;18
Beaudoin;Marc;29
```

La librairie csv permet de lire ou d'écrire des données *comma-separated-values*
https://docs.python.org/3/library/csv.html

Lecture

```
import csv
with open('lorem.csv') as csvfile:
  rows = csv.reader(csvfile)
  for row in rows:
    print(row)
```

Ecriture

```
with open('persons.csv', 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile, delimiter=';')
    csvwriter.writerow(['LaPointe', 'Bobby', 52])
    csvwriter.writerow(['Darc', 'Jeanne', 18])
    csvwriter.writerow(['Darc', 'Jeanne', 18])
```

Le JSON

Définitions

Un document JSON a pour fonction de représenter de l'information hiérarchisée.

Un document JSON ne comprend que deux types d'éléments structurels :

- Des ensembles de paires clé/valeur (key/value)
- Des listes **ordonnées** de valeurs.

Ces mêmes éléments représentent trois types de données :

- Des objets ;
- Des tableaux ;
- Des types simples : chaînes de caractères, booléens, nombres.

La syntaxe

- `[ ]` pour définir des listes ordonnées
- `{ }` pour définir des ensembles
- `"key": value` pour définir une paire clé/valeur

valeur peut être

- une chaîne : `"par exemple"`
- un nombre : `4.65`
- un ensemble : `{ "cle_1_sous_ensemble": "valeur", "cle_2_sous_ensemble": 0 }`
- un tableau : `[ 0, 2, 8, 14 ]`

Manipulation de fichiers `json`

`json`

La librairie `json` permet de manipuler des fichiers json.

https://docs.python.org/3/library/json.html?highlight=json#module-json

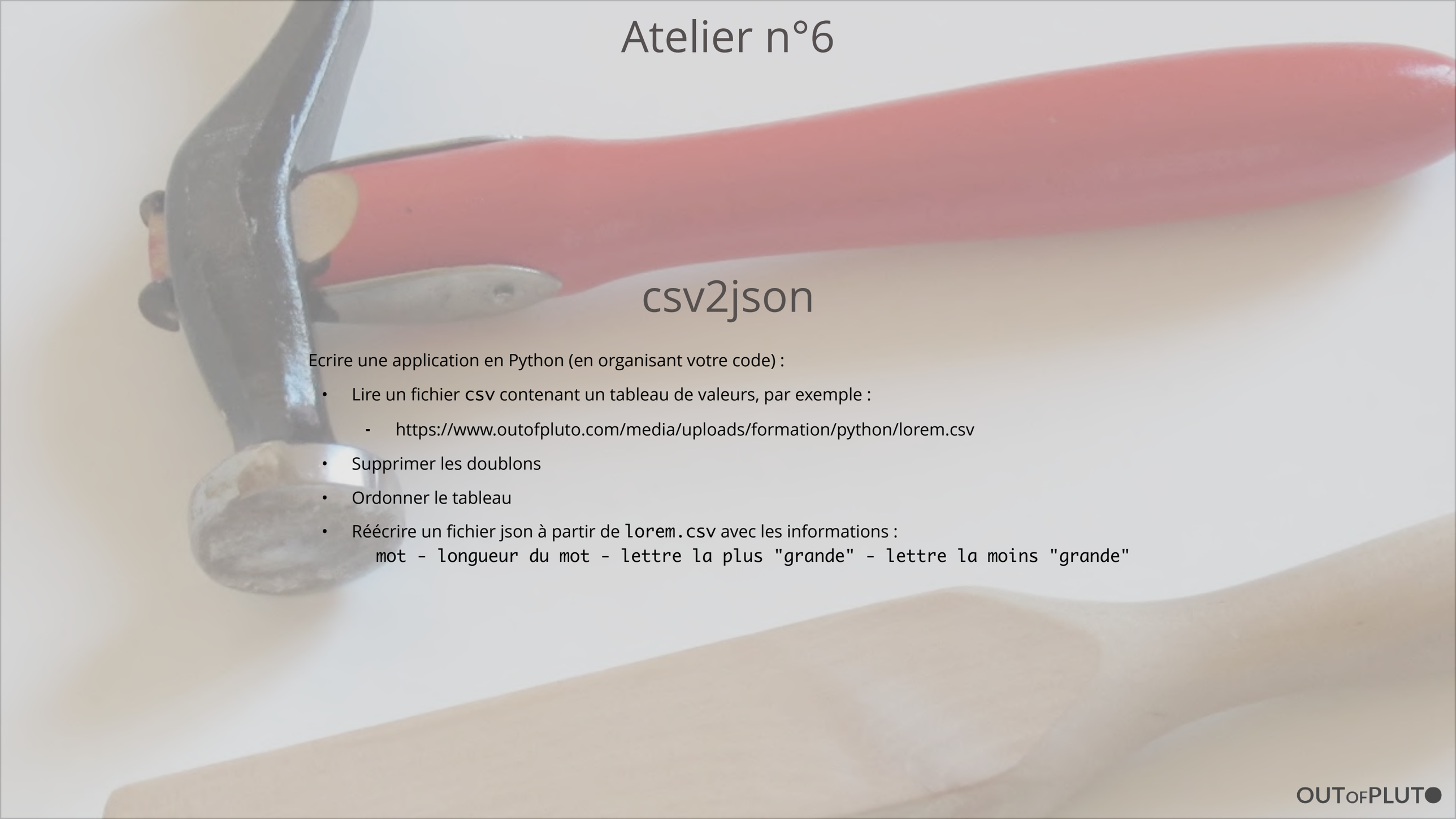
Lecture

```
with open('lorem.json', 'r') as jsonfile:
   result = json.loads(jsonfile.read())
print(result)
```

Ecriture

```
import json
with open('persons.json', 'w') as jsonfile:
   jsonfile.write(json.dumps([{'last': 'LaPointe', 'first': 'Bobby', 'age': 52}]))
```

Atelier n°6



csv2json

Ecrire une application en Python (en organisant votre code) :

- Lire un fichier `csv` contenant un tableau de valeurs, par exemple :
  - https://www.outofpluto.com/media/uploads/formation/python/lorem.csv
- Supprimer les doublons
- Ordonner le tableau
- Réécrire un fichier json à partir de `lorem.csv` avec les informations :
  `mot - longueur du mot - lettre la plus "grande" - lettre la moins "grande"`

Opérations

CRUD

Create-Read-Update-Delete

Il s'agit des 4 opérations élémentaires pour assurer la persistance des données, notamment dans les schémas de base de données.

Opération	MySQL	HTTP
Create (Créer)	INSERT	POST
Read (Lire)	SELECT	GET
Update (Modifier)	UPDATE	PUT/PATCH
Delete (Détruire)	DELETE	DELETE

Logique d’accès à une base de données

La démarche

Utiliser un langage tel que PHP permet d’avoir une “couche” logique et logicielle entre l’utilisateur et la base de données.
C’est une interface permettant une communication plus simple, plus efficace et plus sûre.
Elle permet d’insérer une **connaissance métier**.

La méthodologie standard

- Connexion (dans l’ordre)
  - se connecter au service de base de données (hôte, port, utilisateur, mot de passe…)
  - sélectionner la base de données à consulter ou modifier

- Manipulation (dans n’importe quel ordre et plusieurs fois)
  - demander des informations à l’utilisateur ou récupérer des données grâce à d’autres ressources depuis le PHP
  - envoyer des requêtes de création/lecture/mise à jour/suppression (CRUD)
  - gérer les réponses et les erreurs renvoyées par le service de base de données
  - afficher des informations

- Clôture (à la fin)
  - se déconnecter

Python et MySQL, l'essentiel

L'API originale :

https://dev.mysql.com/doc/connector-python/en/

https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html

```
$ pip install mysql-connector-python
```

Quelques requêtes MySQL

```
SELECT `field1`, `field2` FROM `ma_table`
SELECT * FROM `ma_table`
SELECT * FROM `ma_table` WHERE `field1`="value1"

INSERT INTO `ma_table` (`field1`, `field2`) VALUES ("value1", "value2")

UPDATE `ma_table` SET `field1`="value1", `field2`="value2" WHERE `field3`="value3"

DELETE FROM `ma_table` WHERE `field3`="value3"
```

Petit tour guidé de PHPMyAdmin

Python et MySQL, un exemple

```
# Connexion et sélection de la base
import mysql.connector
cnx = mysql.connector.connect(user='user', password='password',
                              host='host',
                              database='ma_bdd')

# Exécution des requêtes SQL
cursor = cnx.cursor()
query = ("SELECT * FROM ma_table")
cursor.execute(query)

# Affichage des résultats dans un tableau HTML
for row in cursor:
  print(row)

# Important en cas de requête INSERT, UPDATE ou DELETE pour "forcer" le connecteur à envoyer la requête
cnx.commit()

# Libération du curseur et fermeture de la connexion
cursor.close()
cnx.close()
```

Atelier n°7

Base de données MySQL

- Utiliser les informations d'authentification
- Créer un fichier python pour se connecter, lire et modifier la base de données

Principes d'un ORM 1/2

Un ORM (Object-relational Mapper) permet d'accéder à une base de données sans en connaître son formalisme ou sa syntaxe.

La plupart des ORMs peuvent s'interfacer avec différents types de bases de données (SQLite, MySQL, PostGreSQL, MongoDB, Oracle).

Un ORM permet de définir des objets/classes qui, en descendant des classes de l'ORM vont pouvoir exécuter les actions essentielles d'une base de données, le CRUD :

- Create : Création d'un enregistrement en base de données
- Read : Lecture d'un enregistrement en base de données
- Update : Mise à jour d'un enregistrement en base de données
- Delete : Suppression d'un enregistrement en base de données

Un ORM fait automatiquement la conversion entre :

- Le schéma de base de données et les classes
- Les enregistrements dans la base et les instances

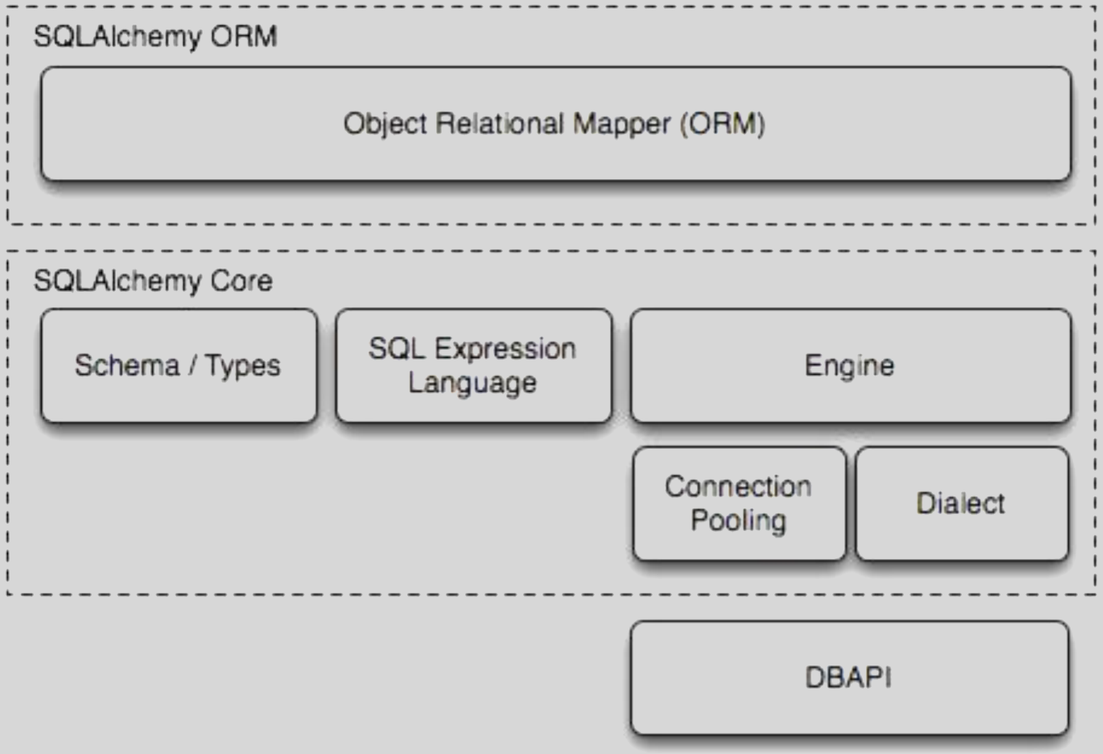
Principes d'un ORM 2/2

Exemple d'un ORM en Python : SQLAlchemy

https://www.sqlalchemy.org/

```
$ pip install SQLAlchemy
```

Schéma d'un ORM basé sur l'exemple de SQLAlchemy



Exemple SQLAlchemy

```
# Exemple incomplet, télécharger le fichier source pour l'ensemble du code fonctionnel


class (Base):
    __tablename__ = 'products'
    id = Column(Integer, primary_key=True)
    title = Column('title', String(32))
    in_stock = Column('in_stock', Boolean)
    quantity = Column('quantity', Integer)
    price = Column('price', Numeric)

    def __str__(self):
        return self.title


product = Product(title='user', in_stock=True, quantity=50, price=5.99)
s.add(product)
s.commit()
```

Atelier n°8

Base de données et ORM

- Reprendre l'atelier n°7
- Réaliser le même travail avec SQLAlchemy