

# Initiation à la programmation avec Python

Matthieu Falce

Novembre 2020

# Au programme I

Hardware

Software

Python

Matthieu Falce

Hardware

Software

Python

# A propos de moi – Qui suis-je ?

Matthieu Falce

Hardware

Software

Python

- ▶ Qui suis-je ?
  - ▶ Matthieu Falce
  - ▶ habite à Lille
  - ▶ ingénieur en bioinformatique (INSA Lyon)

# A propos de moi – Qui suis-je ?

Matthieu Falce

Hardware

Software

Python

- ▶ Qu'est ce que j'ai fait ?
  - ▶ ingénieur R&D en Interaction Homme-Machine (IHM), Inria Lille, équipe [Mint](#) puis [Mjolnir](#)
  - ▶ développeur *fullstack* / *backend* à [FUN-MOOC](#) (France Université Numérique)

# A propos de moi – Actuellement

Matthieu Falce

Hardware

Software

Python

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations

# A propos de moi – Actuellement

Matthieu Falce

Hardware

Software

Python

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*

# A propos de moi – Actuellement

Matthieu Falce

Hardware

Software

Python

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*
- ▶ cofondateur de l'agence de communication **méninges à trois**
  - ▶ stratégie
  - ▶ développement back-end

# A propos de moi – Actuellement

Matthieu Falce

Hardware

Software

Python

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*
- ▶ cofondateur de l'agence de communication **méninges à trois**
  - ▶ stratégie
  - ▶ développement back-end
- ▶ cofondateur / CTO de **ExcellencePriority** (site de partage exclusif de petites annonces orienté luxe)



# A propos de moi – Actuellement

Matthieu Falce

Hardware

Software

Python

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*
- ▶ cofondateur de l'agence de communication **méninges à trois**
  - ▶ stratégie
  - ▶ développement back-end
- ▶ cofondateur / CTO de **ExcellencePriority** (site de partage exclusif de petites annonces orienté luxe)
- ▶ coorganisateur de meetups à Lille
  - ▶ python
  - ▶ big data et machine learning

# Où me trouver ?

Matthieu Falce

Hardware

Software

Python

- ▶ mail: [matthieu@falce.net](mailto:matthieu@falce.net)
- ▶ github : [ice3](#)
- ▶ twitter : [@matthieufalce](#)
- ▶ site: [falce.net](#)

Hardware

Culture générale

Matériel

Software

Python

# Hardware

# 1- Hardware

## 1.1. Culture générale

Pour faire de l'informatique, on a besoin de deux choses :

- ▶ du logiciel
  - ▶ peut être fait sans matériel (théorique)
  - ▶ peut être fait "facilement"
- ▶ du matériel, pour exécuter le logiciel
  - ▶ pas facile à faire soi-même

Dans ce cours de programmation, nous allons nous concentrer sur le logiciel.

# 1- Hardware

## 1.2. Matériel

En pratique, le code fonctionne sur une machine.

- ▶ culture générale informatique
- ▶ quel est ce matériel ? À quoi sert chaque composant ?
- ▶ on ne va voir que les plus importants pour notre cas d'usage

# Stockage longue durée

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python

Permet de stocker des données sur le "long terme".

C'est là où sont enregistrés

- ▶ vos photos
- ▶ vos musiques
- ▶ vos documents
- ▶ ...

On parle de leur capacité de stockage en octets (de nos jours en Giga ou Tera Octets).

Pour information, 1 To permet de stocker <sup>1</sup>:

- ▶ 250 000 photos à 12 millions de pixels
- ▶ 6.5 millions de pages de documents Office ou Pdf (1 300 armoires d'archivage)

---

1.<https://www.dropbox.com/fr/features/cloud-storage/how-much-is-1tb>



# Stockage longue durée

Exemple :

- ▶ disques durs
- ▶ SSD
- ▶ bandes magnétiques (dans les data centers)



Stockage sur bande // Source : <https://www.lemondeinformatique.fr/actualites/lire-pourquoi-l-archivage-sur-bande-est-toujours-d-actualite-73237.html>

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python

# Stockage longue durée

Exemple :

- ▶ disques durs
- ▶ SSD
- ▶ bandes magnétiques (dans les data centers)



# Stockage longue durée

Exemple :

- ▶ disques durs
- ▶ SSD
- ▶ bandes magnétiques (dans les data centers)



Disque dur fonctionnel 3.5" // Source : [https://www.darty.com/nav/achat/console\\_jeux/composant/disque\\_dur\\_interne/western\\_digital\\_3\\_5\\_2\\_to\\_wdbh2d0020hnc.html](https://www.darty.com/nav/achat/console_jeux/composant/disque_dur_interne/western_digital_3_5_2_to_wdbh2d0020hnc.html)

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python

# Stockage longue durée

Exemple :

- ▶ disques durs
- ▶ SSD
- ▶ bandes magnétiques (dans les data centers)



Un SSD 2.5" // Source : <https://fr.wikipedia.org/wiki/SSD>

Permet de stocker les variables de vos programmes.  
Elle s'efface quand on redémarre le matériel (perte d'électricité). Elle a des temps d'accès beaucoup plus rapide que les disques durs ou SSD.

C'est là où est enregistré :

- ▶ l'état de vos programmes en cours d'exécution

On parle de leur capacité de stockage en octets (de nos jours en Giga Octets), de l'ordre de la dizaine à la centaine.

# RAM

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python



Barrettes de RAM // Source : <https://french.alibaba.com/product-detail/2018-ram-memory-2gb-4gb-8gb-ddr2-ddr3-677mhz-1333mhz-1600mhz-desktop-ram-60480803397.html>

# Processeur

Matthieu Falce

Effectue les opérations permettant d'exécuter le code des programmes.

On parle :

- ▶ de la vitesse de calcul, en Hertz (Hz), de l'ordre du Giga
- ▶ du nombre de coeurs, de l'ordre de 5-10



Un CPU, recto-verso // Source :

<https://www.mensup.fr/hitech/mobilite/operateurs/a,107144,lexique-le-processeur-ou-cpu.html>

Hardware

Culture générale

Matériel

Software

Python

Permet d'afficher du contenu à l'écran, mais pas que.

Utilisation :

- ▶ pour les jeux vidéos / logiciels professionnels
- ▶ pour les tâches massivement parallélisables (GPGPU)
  - ▶ encodage vidéo
  - ▶ minage de crypto monnaies
  - ▶ apprentissage de réseaux de neurones profonds
  - ▶ ...



Permet d'afficher du contenu à l'écran, mais pas que.

Utilisation :

- ▶ pour les jeux vidéos / logiciels professionnels
- ▶ pour les tâches massivement parallélisables (GPGPU)
  - ▶ encodage vidéo
  - ▶ minage de crypto monnaies
  - ▶ apprentissage de réseaux de neurones profonds
  - ▶ ...

Les processeurs récents intègrent un GPU peu puissant, sinon on ne pourrait rien afficher à l'écran à part du texte.

Permet d'afficher du contenu à l'écran, mais pas que.

Utilisation :

- ▶ pour les jeux vidéos / logiciels professionnels
- ▶ pour les tâches massivement parallélisables (GPGPU)
  - ▶ encodage vidéo
  - ▶ minage de crypto monnaies
  - ▶ apprentissage de réseaux de neurones profonds
  - ▶ ...

Les processeurs récents intègrent un GPU peu puissant, sinon on ne pourrait rien afficher à l'écran à part du texte.

C'est une sorte de mini-ordinateur dans l'ordinateur, elle a sa propre RAM, ses propres cœurs de calculs...

# Carte graphique

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python



Un GPU // Source :

<https://www.eurogamer.net/articles/digitalfoundry-2019-09-27-gpu-power-ladder-all-graphics-cards-tested>

# Et pour notre cours ?

Matthieu Falce

Hardware

Culture générale

Matériel

Software

Python

Nous allons utiliser :

- ▶ le disque dur pour stocker nos fichiers
- ▶ la RAM pour contenir nos variables
- ▶ (un seul coeur) CPU pour exécuter du code

Hardware

**Software**

Langages de programmation

Algorithmes

Outils

Intérêt

*no-code*

Python

# Software

## 2- Software

### 2.1. Langages de programmation

Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est composé d'un alphabet, d'un vocabulaire, de règles de grammaire et de significations.

---

[https://fr.wikipedia.org/wiki/Langage\\_de\\_programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation)

Les langages de programmation permettent de décrire d'une part les structures des données qui seront manipulées par l'appareil informatique, et d'autre part d'indiquer comment sont effectuées les manipulations, selon quels algorithmes. Ils servent de moyens de communication par lesquels le programmeur communique avec l'ordinateur, mais aussi avec d'autres programmeurs ; les programmes étant d'ordinaire écrits, lus, compris et modifiés par une équipe de programmeurs.

---

[https://fr.wikipedia.org/wiki/Langage\\_de\\_programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation)



Spécificités d'un langage informatique :

- ▶ comme une langue naturelle, il y a une grammaire et des éléments à respecter

## Spécificités d'un langage informatique :

- ▶ comme une langue naturelle, il y a une grammaire et des éléments à respecter
- ▶ beaucoup plus rigoureux qu'une langue naturelle (pas de place à l'interprétation des instructions)

## Spécificités d'un langage informatique :

- ▶ comme une langue naturelle, il y a une grammaire et des éléments à respecter
- ▶ beaucoup plus rigoureux qu'une langue naturelle (pas de place à l'interprétation des instructions)
- ▶ permet la communication
  - ▶ avec l'ordinateur
    - ▶ en décrivant des algorithmes
    - ▶ en décrivant des structures de données
  - ▶ avec d'autres développeurs (travail en équipe)
  - ▶ avec soi même

- ▶ les premiers langages sont apparus dans les années 1950, en même temps que les ordinateurs
- ▶ il existe une filiation entre les langages, certains s'inspirant d'autres pour leur syntaxe ou leur paradigme

Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ impératif ou procédural : les langages de ce type exécutent des actions étapes par étape
  - ▶ C, Pascal, Fortran, Cobol ...

Certains langages peuvent être multi-paradigmes.

Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ déclaratif fonctionnel : ces langages décrivent des fonctions mathématiques, le code peut être exécuté par des appels successifs de fonctions, aucune valeur n'est assignée de façon indépendante (elle n'est vue que comme le résultat d'une fonction)
  - ▶ Lisp, Haskell, OCaml, ...

Certains langages peuvent être multi-paradigmes.

Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ déclaratif logique : les langages de ce type vont décrire des prédicats (déclarations qui peuvent être vraies ou fausses). Le programme va répondre à une question par des recherches sur des ensembles en utilisant des axiomes, des requêtes et des règles de déduction.
  - ▶ Prolog, ...

Certains langages peuvent être multi-paradigmes.

Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ orienté objet : ce paradigme facilite le découpe de gros programme en les découpant en modules isolé (les objets). Un objet contient des variables et des fonctions en rapport avec un sujet donné.
  - ▶ Smalltalk, Java, C++

Certains langages peuvent être multi-paradigmes.



Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ concurrent : les programmes de ce type permettent d'effectuer plusieurs tâches "en même temps" (soit en basculant très rapidement d'une tâche à l'autre, soit en les exécutant sur plusieurs coeurs)
  - ▶ Java, C++, C, Go

Certains langages peuvent être multi-paradigmes.

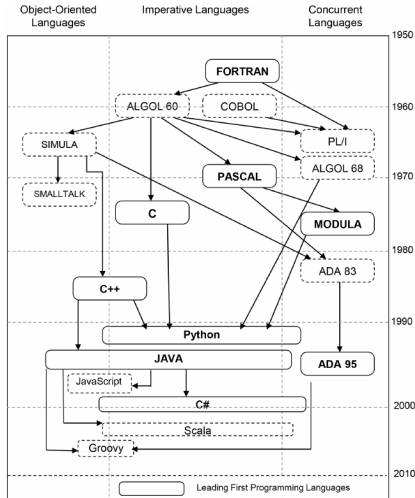
Les langages peuvent se classer dans différentes grosses catégories, selon leur approche (on parle de paradigme de programmation).

- ▶ événementiel : les programmes interactifs sont de ce type de paradigme. La programmation consiste à décrire les actions à avoir en réponse à des événements (clic, déplacement de souris, fin d'un téléchargement, horloge, ...)
  - ▶ Simula, Javascript, ...

Certains langages peuvent être multi-paradigmes.

# Généalogie

Matthieu Falce



Filiation entre les languages

Source : Farooq, & al. (2014). An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages. PloS one. 9. e88941. 10.1371/journal.pone.0088941.

Hardware

Software

Langages de programmation

Algorithmes

Outils

Intérêt

no-code

Python

Nous allons nous intéresser à Python <sup>2</sup>.

- ▶ plutôt ancien (début années 90)
- ▶ multi-paradigme
- ▶ syntaxe légère / facile à manipuler
- ▶ activement utilisé dans l'industrie et l'éducation
- ▶ écosystème très développé

---

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

Nous allons nous intéresser à Python <sup>2</sup>.

- ▶ plutôt ancien (début années 90)
- ▶ multi-paradigme
- ▶ syntaxe légère / facile à manipuler
- ▶ activement utilisé dans l'industrie et l'éducation
- ▶ écosystème très développé

Pour quelles tâches voulez-vous l'utiliser ?

Hardware

Software

Langages de programmation

**Algorithmes**

Outils

Intérêt

*no-code*

Python

## 2- Software

### 2.2. Algorithmes

Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre une classe de problèmes.

Le mot algorithme vient du nom d'un mathématicien perse du IXe siècle, Al-Khwârizmî

---

<https://fr.wikipedia.org/wiki/Algorithme>

Un algorithme est une méthode générale pour résoudre un type de problèmes. Il est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé.

On mesure l'efficacité d'un algorithme notamment par sa durée de calcul, par sa consommation de mémoire vive (en partant du principe que chaque instruction a un temps d'exécution constant), par la précision des résultats obtenus (par exemple avec l'utilisation de méthodes probabilistes), sa scalabilité (son aptitude à être efficacement parallélisé), etc

---

<https://fr.wikipedia.org/wiki/Algorithme>



# Exemples dans la vie courante

Matthieu Falce

Hardware

Software

Langages de programmation

Algorithmes

Outils

Intérêt

no-code

Python

- ▶ recette de cuisine (souvent présentée de façon ambiguë)
- ▶ notice d'un montage de meuble
- ▶ tissage (utilise même des cartes perforées proches de celles des premiers ordinateurs)
- ▶ étapes pour la résolution d'un *Rubik's Cube*
- ▶ protocole d'un diagnostic ou traitement médical
- ▶ ...

- ▶ l'algorithme ne dépend pas du langage de programmation (certains paradigmes vont impacter l'algorithme utilisé)
- ▶ il peut exister plusieurs algorithmes permettant d'obtenir le même résultat
- ▶ il peut être visualisé sous forme textuelle
- ▶ ou sous forme graphique (on parle d'algorigramme ou d'organigramme de programmation)

La représentation doit être la plus normalisée possible pour prévoir tous les cas et être non ambiguë.

En pratique, la finalité de l'algorithme est d'être *implémenté* en code. Cela peut être plus ou moins difficile selon les langages.

# Représentations

Matthieu Falce

Hardware

Software

Langages de programmation

Algorithmes

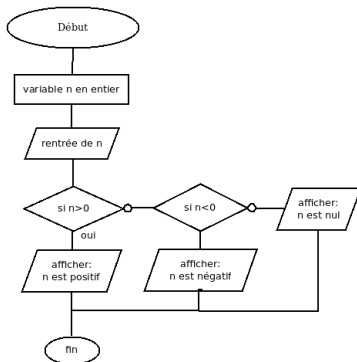
Outils

Intérêt

no-code

Python

```
Variable n en Entier
Début
Ecrire "Entrez un nombre : "
Lire n
Si n > 0 Alors
    Ecrire "Ce nombre est positif"
Sinon
    Si n < 0 Alors
        Ecrire "Ce nombre est négatif"
    Sinon
        Ecrire "Ce nombre est nul"
Finsi
Finsi
Fin
```



Algorithme permettant d'afficher si un nombre est positif ou négatif.

Source : [http://sti2d.ecolelamache.org/partie\\_2\\_cration\\_dalgorithmes\\_ou\\_dalgorigrammes.html](http://sti2d.ecolelamache.org/partie_2_cration_dalgorithmes_ou_dalgorigrammes.html)

# Flowchart

Matthieu Falce

Hardware

Software

Langages de programmation

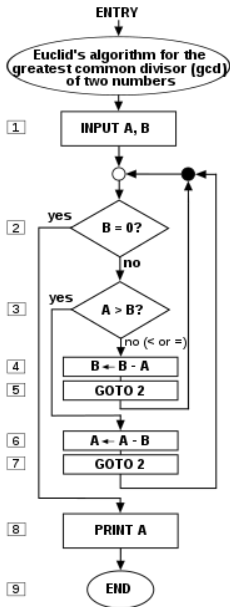
Algorithmes

Outils

Intérêt

no-code

Python



Exemple de *flowchart*.

Source : <https://en.wikipedia.org/wiki/Algorithm>

# Algorithmme – ce que l'on peut faire

Matthieu Falce

Hardware

Software

Langages de programmation

**Algorithmes**

Outils

Intérêt

*no-code*

Python

- ▶ assigner une valeur à une variable (pour la réutiliser plus tard)
- ▶ effectuer une suite d'instructions, linéairement ou pas
- ▶ brancher selon une condition (Si ma valeur vaut ... alors ..., sinon ...)
- ▶ revenir à un point précédent (itération, liée à une condition permettant de sortir de la boucle)

Il existe 2 grandes catégories de métiers touchant au logiciel en informatique :

- ▶ la recherche algorithmique (*computer science* en anglais)
  - ▶ on va chercher à découvrir l'algorithme le plus optimisé pour résoudre un problème donné
  - ▶ plutôt théorique (peut se faire sans coder)
  - ▶ proche des mathématiques

Je pense qu'il faut trouver l'équilibre entre les deux, que tout est question de curseur selon ses préférences personnelles.

Il existe 2 grandes catégories de métiers touchant au logiciel en informatique :

- ▶ la recherche algorithmique (*computer science* en anglais)
  - ▶ on va chercher à découvrir l'algorithme le plus optimisé pour résoudre un problème donné
  - ▶ plutôt théorique (peut se faire sans coder)
  - ▶ proche des mathématiques
- ▶ le développement
  - ▶ on va chercher à développer de nouveaux programmes, ajouter des fonctionnalités
  - ▶ proche de l'artisanat

Je pense qu'il faut trouver l'équilibre entre les deux, que tout est question de curseur selon ses préférences personnelles.

Hardware

Software

Langages de programmation

Algorithmes

**Outils**

Intérêt

*no-code*

Python

## 2- Software

### 2.3. Outils



Jusqu'à présent nous avons eu seulement besoin d'un papier et d'un crayon.

Comment exécuter nos algorithmes ?

Nous allons devoir les implémenter, les coder dans un langage (Python dans notre cas).

Nous allons avoir besoin de 2 choses :

- ▶ un outil pour éditer le code (le texte de notre programme)
- ▶ un outil pour traduire ce texte en *quelque chose* compréhensible par l'ordinateur

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ Word

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)
- ▶ Notepad (le bloc note windows)

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)
- ▶ Notepad (le bloc note windows)
- ▶ des éditeurs de texte (comprenant que nous éditons du code)

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)
- ▶ Notepad (le bloc note windows)
- ▶ des éditeurs de texte (comprenant que nous éditons du code)
- ▶ des IDE (Integrated Development Editor), spécialisées sur un langage

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)
- ▶ Notepad (le bloc note windows)
- ▶ des éditeurs de texte (comprenant que nous éditons du code)
- ▶ des IDE (Integrated Development Editor), spécialisées sur un langage
- ▶ ...

Le code n'est qu'un fichier texte. Pour l'éditer nous pouvons utiliser ce que nous voulons :

- ▶ ~~Word~~ (les fichiers qu'il produit sont formatés et ne contiennent pas *exactement* ce que nous avons tapé)
- ▶ Notepad (le bloc note windows)
- ▶ des éditeurs de texte (comprenant que nous éditons du code)
- ▶ des IDE (Integrated Development Editor), spécialisées sur un langage
- ▶ ...

Dans ce cours, je vous invite à utiliser VSCode <sup>3</sup>  
(multi-plateforme, mode python, très complet)

---

3.<https://code.visualstudio.com/>



Il faut traduire le code pour que l'ordinateur le comprenne.

Il y a deux types d'outils pour cela <sup>4</sup> :

- ▶ les compilateurs : qui prennent des fichiers de code, le traduisent avant son exécution et génèrent un fichier exécutable (un .exe ou une .dll sous windows). Ce fichier est alors compréhensible par l'ordinateur (pour une architecture précise)

.

---

4. Certains langages peuvent être les 2 à la fois, ce n'est pas exclusif

Il faut traduire le code pour que l'ordinateur le comprenne.

Il y a deux types d'outils pour cela <sup>4</sup> :

- ▶ les compilateurs : qui prennent des fichiers de code, le traduisent avant son exécution et génèrent un fichier exécutable (un .exe ou une .dll sous windows). Ce fichier est alors compréhensible par l'ordinateur (pour une architecture précise)
- ▶ les interpréteurs : qui prennent des fichiers de code et les traduisent pendant son exécution. Il n'y a pas de fichier exécutable, il est exécuté directement par l'interpréteur. Le code est alors compréhensible par l'ordinateur dès qu'un interpréteur est disponible.

.

---

4.Certains langages peuvent être les 2 à la fois, ce n'est pas exclusif

Il faut traduire le code pour que l'ordinateur le comprenne.

Il y a deux types d'outils pour cela <sup>4</sup> :

- ▶ les compilateurs : qui prennent des fichiers de code, le traduisent avant son exécution et génèrent un fichier exécutable (un .exe ou une .dll sous windows). Ce fichier est alors compréhensible par l'ordinateur (pour une architecture précise)
- ▶ les interpréteurs : qui prennent des fichiers de code et les traduisent pendant son exécution. Il n'y a pas de fichier exécutable, il est exécuté directement par l'interpréteur. Le code est alors compréhensible par l'ordinateur dès qu'un interpréteur est disponible.

Dans ce cours, nous allons utiliser l'interpréteur Python <sup>5</sup>.

---

4. Certains langages peuvent être les 2 à la fois, ce n'est pas exclusif

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

Hardware

Software

Langages de programmation

Algorithmes

Outils

**Intérêt**

*no-code*

Python

## 2- Software

### 2.4. Intérêt

# Quel intérêt à la programmation

Matthieu Falce

Hardware

Software

Langages de programmation

Algorithmes

Outils

**Intérêt**

*no-code*

Python

- ▶ automatisation
  - ▶ de tâches longues
  - ▶ de tâches complexes
- ▶ diminution des erreurs
- ▶ meilleure vitesse d'exécution que par un humain

Hardware

Software

Langages de programmation

Algorithmes

Outils

Intérêt

*no-code*

Python

## 2- Software

### 2.5. *no-code*

Récemment plusieurs outils cherchent à ne pas utiliser de code pour :

- ▶ créer des applications mobiles
- ▶ créer des sites internet
- ▶ gérer des bases de données
- ▶ ...

Ce n'est pas forcément un problème pour les développeurs :

- ▶ ces outils sont chers
- ▶ ils sont limités (pas de spécialisation métier facile sans code)
- ▶ ils permettent de faciliter la création de prototypes

Hardware

Software

**Python**

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

# Python



Hardware

Software

Python

**Généralités**

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

## 3- Python

### 3.1. Généralités

- ▶ créé au début des années 90 par Guido Van Rossum
- ▶ le nom vient des *Monty Python* et pas du serpent
- ▶ utilisé pour du scripting puis pour de réels programmes
- ▶ l'usage décolle depuis 5-10 ans, porté par l'explosion des *data science*

# Pourquoi Python

Matthieu Falce

Hardware

Software

Python

**Généralités**

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ langage multi-paradigmes
  - ▶ impératif
  - ▶ orienté objet
  - ▶ fonctionnel
- ▶ facile à apprendre
  - ▶ syntaxe simple
  - ▶ initiation aisée aux différents concepts
  - ▶ (typage dynamique, ramasse-miette, exceptions...)
- ▶ simple mais pas simpliste

# Pourquoi Python

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ bibliothèque standard et écosystème très complet
- ▶ très utilisé dans de nombreux domaines
  - ▶ data science
  - ▶ finance
  - ▶ ingénierie
  - ▶ DIY
  - ▶ devops
  - ▶ cinéma
  - ▶ web
  - ▶ ...
- ▶ meilleur nulle part, mais excellent partout
  - ▶ il y a souvent un langage plus adapté à une tâche précise
  - ▶ nos projets complexes mélangent souvent plusieurs domaines
  - ▶ python permet de n'utiliser qu'un seul langage

# Qui l'utilise

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Un peu tout le monde, du développeur indépendant au GAFA.

Parmi les grands noms :

- ▶ Google
- ▶ Dropbox
- ▶ Uber
- ▶ Netflix
- ▶ Instagram
- ▶ NASA
- ▶ ...

Hardware

Software

Python

Généralités

**Syntaxe**

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

## 3- Python

### 3.2. Syntaxe

Nous allons à présent utiliser un langage de programmation.  
Votre interlocuteur (l'ordinateur) sera bien plus strict que moi :)

- ▶ langage *formel*
- ▶ doit être compris par l'ordinateur
- ▶ syntaxe stricte (assez frustrant au départ)
- ▶ comme pour parler une langue humaine, il est nécessaire de maîtriser la syntaxe avant de chercher à communiquer

La syntaxe du langage comporte plusieurs éléments :

- ▶ les déclarations et assignations de variables
- ▶ les déclarations de blocs (fonctions, conditions, boucles, classes)
  - ▶ commencent par un mot clé et la ligne se finie par un :
  - ▶ c'est l'indentation (le décalage) du bloc qui permet de savoir quand il se termine
  - ▶ l'indentation est donc obligatoire en python
- ▶ les commentaires qui servent à donner des informations aux êtres humains manipulant le code
  - ▶ décrire la raison plutôt que juste répéter ce que l'on fait



# Exemple

Matthieu Falce

Hardware

Software

Python

Généralités

**Syntaxe**

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

[Aller plus loin](#)

```
"""Programme permettant de séparer une liste de nombre  
selon s'ils sont plus ou moins grand qu'un pivot  
"""
```

```
# définition d'un pivot
```

```
pivot = 5
```

```
# make two empty lists
```

```
petits = []
```

```
grands = []
```

```
# sépare les nombre dans petits et grands
```

```
for nombre in range(10):
```

```
    if nombre < pivot:
```

```
        petits.append(nombre)
```

```
    else:
```

```
        grands.append(nombre)
```

```
print("petits:", petits)
```

```
print("grands:", grands)
```

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

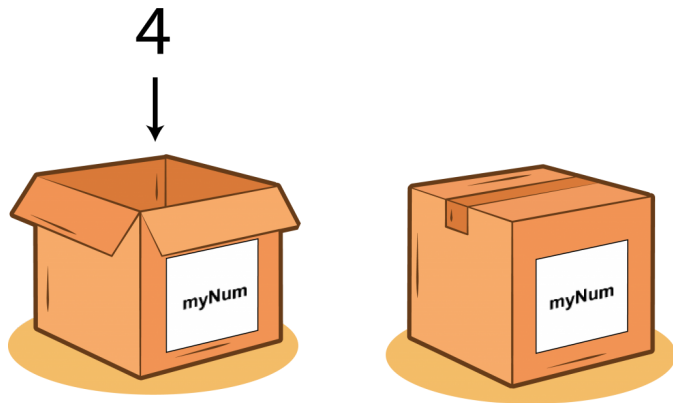
Outils et bonnes pratiques

Aller plus loin

## 3- Python

### 3.3. Variables

# Concept d'une variable



Assignment d'une variable

Source :

<http://digital.academy.free.fr/blog/python-syntax-variables-definition-declaration-data-types-and-scope/>

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

# Concept d'une variable

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

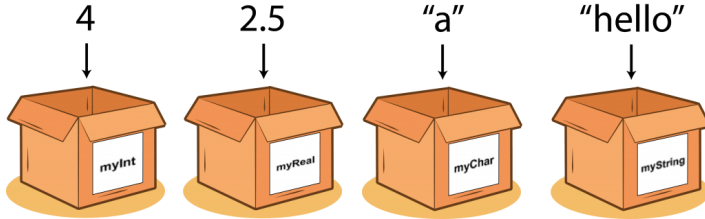
**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin



Variables de différents types

Source :

<http://digital.academy.free.fr/blog/python-syntax-variables-definition-declaration-data-types-and-scope/>

# Assignment d'une variable

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

L'assignation consiste à mettre une valeur dans une variable.

- ▶ l'assignation s'effectue avec le signe égal (=)
- ▶ l'assignation n'est valide qu'à un instant d'un temps

Les variables sont le seul moyen de garder une mémoire dans nos programmes.

Il faut éviter de nommer ses variables avec des mots utilisés dans le langage :

- ▶ int
- ▶ list
- ▶ print
- ▶ range
- ▶ ...

# Différence entre assignation et égalité

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ assignation :
  - ▶ on déclare le nom de la variable et une valeur en même temps
  - ▶ c'est comme si l'on ouvrait une boîte, que l'on mettait une étiquette avec le nom de la variable et le contenu de la variable à l'intérieur
  - ▶ en python, on utilise le signe égal =
  - ▶ on peut assigner une variable à une valeur ou au contenu d'une autre variable

# Différence entre assignation et égalité

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ assignation :
  - ▶ on déclare le nom de la variable et une valeur en même temps
  - ▶ c'est comme si l'on ouvrait une boîte, que l'on mettait une étiquette avec le nom de la variable et le contenu de la variable à l'intérieur
  - ▶ en python, on utilise le signe égal =
  - ▶ on peut assigner une variable à une valeur ou au contenu d'une autre variable
- ▶ égalité :
  - ▶ on cherche à savoir si 2 variables sont égales, si elles contiennent la même chose
  - ▶ on prend 2 boîtes différentes, on les ouvre et on compare ce qu'elles ont à l'intérieur
  - ▶ souvent utilisé pour les exécutions conditionnelles (les if)

# Les différents types de variables

Matthieu Falce

Il existe différents types de variables, chacun ayant ses caractéristiques

- ▶ les scalaires (nombres, caractères et objets arbitraires) : ne contiennent qu'une valeur

Pour connaître le type d'une variable, on utilise `type(ma_variable)`

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin



# Les différents types de variables

Matthieu Falce

Il existe différents types de variables, chacun ayant ses caractéristiques

- ▶ les scalaires (nombres, caractères et objets arbitraires) : ne contiennent qu'une valeur
- ▶ les conteneurs : peuvent contenir de 0 à une infinité d'autres variables
  - ▶ les chaînes : permettent de stocker des textes (caractères)
  - ▶ les listes : permettent de stocker des valeurs arbitraires
  - ▶ les tuples : comme des listes mais ne peuvent plus être modifiées une fois créés
  - ▶ les dictionnaires : permettent d'associer une clé à une valeur
  - ▶ les sets : permettent de stocker un ensemble d'éléments sans doublons

Pour connaître le type d'une variable, on utilise `type(ma_variable)`

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

# Les scalaires

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

```
a = 2
b = 3.5
c = a * (b + a)
a = 4
```

```
print(c)
print(type(a), type(b), type(c))
```

```
booleen = True
booleen2 = 2 == 3
print(type(booleen), type(booleen2))
```

# Les chaines

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

```
une_chaine = "il était une fois,"
```

```
une_autre = "une princesse"
```

```
# opération sur une chaine (concaténation)
```

```
concatenation = une_chaine + " " + une_autre
```

```
print(concatenation, type(concatenation))
```

```
print(une_autre * 3)
```

```
# application de fonctions et méthodes
```

```
print(len(concatenation))
```

```
print(concatenation.split(" "))
```

```
# slicing
```

```
print(concatenation[5])
```

```
print(concatenation[1:5])
```

# Les listes

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

[Aller plus loin](#)

```
# une liste peut contenir un nombre infini d'éléments
# de types différents et imbriqués
une_liste = [1, 2, "des mots", [3, 4], []]

# accéder à un élément
premier = une_liste[0]
dernier = une_liste[-1]
print(premier, type(premier))
print(dernier, type(dernier))

# modifier un élément
une_liste[1] = "modifié"
print(une_liste)

# ajouter un élément
## attention certaines actions modifient la liste "en place"
## d'autres en renvoient une nouvelle
une_liste.append(5)
print(une_liste)

une_autre_liste = une_liste + [1, 2, 3]
print(une_autre_liste)
```

# Les tuples

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

*# un tuple est constant, on ne  
# peut pas le modifier une fois créé*

```
mon_tuple = (1, 2, "a", [3, 4])  
print(mon_tuple[3])
```

*# création d'un tuple à un élément*  
tuple\_un\_element = ("toto",)

# Les tuples

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

*# un tuple est constant, on ne  
# peut pas le modifier une fois créé*

```
mon_tuple = (1, 2, "a", [3, 4])  
print(mon_tuple[3])
```

*# création d'un tuple à un élément*  
tuple\_un\_element = ("toto",)

La différence entre liste et tuple est subtile, c'est un signal qu'envoie le développeur aux autres.

# Les dictionnaires

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

**Variables**

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

```
# définition d'un dictionnaire
num_telephone = {"Matthieu": "06xxxxxx"}

# ajout d'un élément
num_telephone["Thomas"] = "07xxxxxx"

# récupération d'un élément
print(num_telephone["Matthieu"])

# on vérifie si un élément est présent avec `in`
jerome_est_present = "Jerome" in num_telephone
```

# Les dictionnaires

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

```
# définition d'un dictionnaire
num_telephone = {"Matthieu": "06xxxxxx"}

# ajout d'un élément
num_telephone["Thomas"] = "07xxxxxx"

# récupération d'un élément
print(num_telephone["Matthieu"])

# on vérifie si un élément est présent avec `in`
jerome_est_present = "Jerome" in num_telephone
```

Les dictionnaires sont très pratiques pour simplifier des programmes.

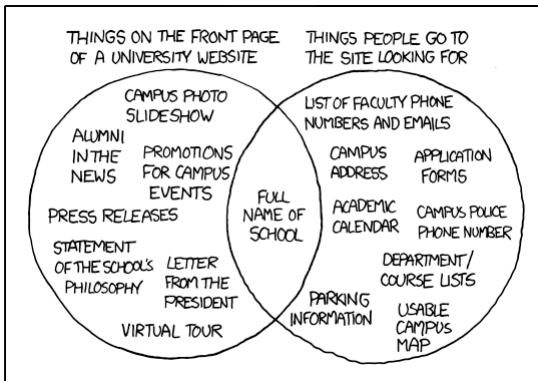
Limite : on ne peut stocker que des objets immutables en clé (nombres, chaînes, tuples).



# Les sets

Matthieu Falce

Les sets sont très utiles pour effectuer des opérations ensemblistes, telles que celles représentées sur des diagrammes de Venn.



Source : <https://xkcd.com/773/>

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Les sets sont très utiles pour effectuer des opérations ensemblistes, telles que celles représentées sur des diagrammes de Venn.

```
premier_ensemble = {1, 2, 3, 4, 5, 3}
print(premier_ensemble) # pas de duplications

second_ensemble = {4, 5, 6, 7, 8}

intersection = premier_ensemble.intersection(second_ensemble)
intersection2 = premier_ensemble & second_ensemble
print(intersection, intersection2)

sous_ensemble = premier_ensemble.issubset(second_ensemble)
sous_ensemble2 = premier_ensemble < second_ensemble

union = premier_ensemble.union(second_ensemble)
union2 = premier_ensemble | second_ensemble
```

Il peut exister une infinité d'autre types (nous pouvons créer les nôtres) :

- ▶ les fichiers
- ▶ les itérateurs
- ▶ ce qui sera retourné par différentes bibliothèques

Il faut aller regarder les documentations des types pour connaître les propriétés des variables.

On ne peut pas faire ce que l'on veut avec les types. Des opérateurs peuvent changer de comportement voire lever des erreurs.

```
>>> 1 + "1"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Hardware

Software

Python

Généralités

Syntaxe

Variables

**Structures de base**

Fonctions

Outils et bonnes pratiques

Aller plus loin

## 3- Python

### 3.4. Structures de base

# Les actions possibles

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ assigner une valeur à une variable
- ▶ comparer des variables et des traitements conditionnels
- ▶ boucler jusqu'à la réalisation d'une condition
- ▶ ces différents blocs peuvent être imbriqués

La traduction des algorithmes est très simple grâce à la légèreté de la syntaxe. Vous pouvez toujours utiliser le site <http://www.pythontutor.com/> pour visualiser l'état des variables à chaque bout d'exécution de votre code.

# Comparaisons

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

En python il est possible d'effectuer des comparaisons entre des variables ou des valeurs. A la fin, vous avez une variable booléenne qui va valoir True ou False

```
a = 1
```

```
b = 2
```

```
c = 3
```

```
egalite = a == b
```

```
difference = a != b
```

```
inferieur = a < b
```

```
superieur = a > b
```

```
appartenance = a < b < c
```

```
l = [1, 2, 3]
```

```
contenance = a in l
```

# Opérateurs de comparaison

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Il est également possible d'effectuer des opérations sur les éléments de comparaisons pour les modifier ou les regrouper.

```
a = 30
```

```
b = "test"
```

```
a_supérieur_18 = a > 18
```

```
b_contient_z = "z" in b
```

```
print(a_supérieur_18 and b_contient_z)
```

```
print(a_supérieur_18 or b_contient_z)
```

```
print(not a_supérieur_18)
```

```
booleans = [a_supérieur_18, b_contient_z]
```

```
print(all(booleans))
```

```
print(any(booleans))
```



# Traitements conditionnels

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Une fois que vous avez fait votre comparaison, vous pouvez avoir des traitements conditionnels. Nous entrerons dans les différents blocs en fonction du résultat de la comparaison.

```
a_trouver = 42  
valeur = 43
```

```
if valeur == a_trouver:  
    print("Bravo, tu as trouvé")  
else:  
    print("eh non ! Essaies encore")
```

# Traitements conditionnels

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Une fois que vous avez fait votre comparaison, vous pouvez avoir des traitements conditionnels. Nous entrerons dans les différents blocs en fonction du résultat de la comparaison.

```
age = 30
```

```
if age > 18:
    print("tu es majeur")
elif age < 0:
    print("tu n'es pas né")
else:
    print("tu es mineur")
```

Il existe deux types de boucles :

- ▶ les boucles `for`
  - ▶ permettent de parcourir un conteneur (liste, set, tuple, dict) élément par élément
  - ▶ à utiliser quand l'on connaît le nombre d'itérations à effectuer
- ▶ les boucles `while`
  - ▶ attendent explicitement la résolution d'une condition pour se terminer
  - ▶ à utiliser quand l'on ne connaît pas le nombre d'itérations à effectuer ou que l'on attend que des conditions soient remplies

# Les boucles – for

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

En python, on itère sur des conteneurs (listes, sets, tuples, dicts, ...) et on récupère directement les valeurs des éléments individuels

```
prenums = ["Matthieu", "Thomas", "Lucie"]
```

```
for prenom in prenums:  
    print("Bonjour", prenom)
```

# Les boucles – while

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

```
cible = 10  
valeur = -1.3
```

```
while valeur < cible:  
    valeur += 0.2
```

```
print("la valeur cible a été atteinte")
```

Il faut faire très attention à la condition d'arrêt pour ne pas avoir une boucle infinie.

Il est possible de modifier le comportement d'une boucle avec les modificateurs suivants

- ▶ `continue` : pour sauter directement à l'élément suivant
- ▶ `break` : pour sortir complètement de la boucle (souvent utilisé avec les boucles `while True`)

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

**Fonctions**

Outils et bonnes pratiques

Aller plus loin

## 3- Python

### 3.5. Fonctions

- ▶ nous en avons déjà utilisé (`print`, `len`...), on parle de fonction `builtin`
- ▶ permettent de réduire la complexité d'un programme
  - ▶ isolation d'un morceau de code qui peut être réutilisé à plusieurs endroits
- ▶ elles sont caractérisées par leurs paramètres et leurs valeurs retournée
- ▶ on peut créer les nôtres grâce au mot clé `def`
- ▶ dans l'idéal, une fonction ne doit faire qu'une seule chose



- ▶ une fonction retourne toujours quelque chose
- ▶ une fonction peut prendre de 0 à un nombre infini de paramètres
- ▶ on ne définit pas le type des paramètres
- ▶ si l'on ne lui passe pas le bon nombre de paramètres, il y a une erreur
- ▶ on peut utiliser des variables comme paramètres

# Définition et appel

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

**Fonctions**

Outils et bonnes pratiques

Aller plus loin

```
def addition(nombre_1, nombre_2):  
    return nombre_1 + nombre_2
```

```
a = 1
```

```
b = 2
```

```
somme = addition(a, b)
```

```
print(somme)
```

Les fonctions renvoient toujours quelque chose :

```
def addition_1(x, y):  
    res = x + y
```

```
def addition_2(x, y):  
    res = x + y  
    return res
```

```
print(addition_1(1, 2))  
print(addition_2(1, 2))
```

Les fonctions peuvent appeler d'autres fonctions. Il est possible d'avoir autant de blocs imbriqués les uns dans les autres que nécessaire.

```
def addition(x, y):  
    return x + y  
  
def calcule(operation, x, y):  
    if operation == "addition":  
        print("on va faire une addition")  
        res = addition(x, y)  
    return res  
  
def main():  
    calcule("addition", 1, 2)  
  
main()
```

# Différentes catégories de variables

Matthieu Falce

Une fonction a accès aux variables qui sont déclarées en dehors de son bloc (variables *globales*). Par contre, les variables déclarées dans les fonctions (variables locales) ne sont pas accessibles en dehors. Les fonctions isolent leurs variables locales.

```
variable = 1
```

```
def acces_variables():  
    print(variable)
```

```
def acces_variables_2():  
    variable = 3  
    print(variable)
```

```
def acces_variables_3(variable):  
    print(variable)
```

```
acces_variables()  
acces_variables_2()  
print(variable)  
  
acces_variables_3(5)
```

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

# Différentes catégories de variables

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

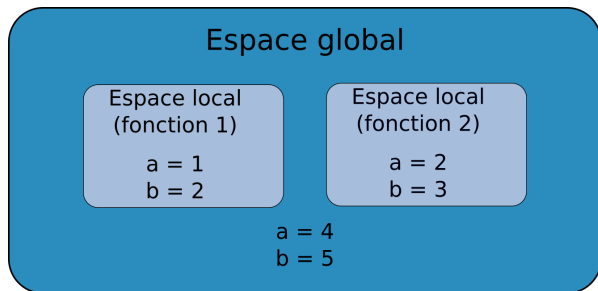
Structures de base

**Fonctions**

Outils et bonnes pratiques

Aller plus loin

Une fonction a accès aux variables qui sont déclarées en dehors de son bloc (variables *globales*). Par contre, les variables déclarées dans les fonctions (variables locales) ne sont pas accessibles en dehors. Les fonctions isolent leurs variables locales.



Les différents espaces mémoires créés par les fonctions

# Exceptions et analyse d'une stacktrace

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Certaines parties de code vont planter, cela peut être dû à une erreur de programmation, c'est normal, ça arrive. Python est assez explicite dans ses messages d'erreurs.

```
def addition(x, y):  
    return x + y  
  
def calcul(operation, x, y):  
    if operation == "addition":  
        print("on va faire une addition")  
        res = addition(x, y)  
        return res  
  
def main():  
    calcul("addition", 1, "2")  
  
main()
```

# Exceptions et analyse d'une stacktrace

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

Certaines parties de code vont planter, cela peut être dû à une erreur de programmation, c'est normal, ça arrive.  
Python est assez explicite dans ses messages d'erreurs.

Traceback (most recent call last):

```
File "python/codes/fr/fonctions/erreur.py", line 16, in <module>
    main()
File "python/codes/fr/fonctions/erreur.py", line 13, in main
    calcule("addition", 1, "2")
File "python/codes/fr/fonctions/erreur.py", line 8, in calcule
    res = addition(x, y)
File "python/codes/fr/fonctions/erreur.py", line 2, in addition
    return x + y
```

**TypeError:** unsupported operand **type(s)** **for** **+**: 'int' and 'str'



En python il est très facile de créer et d'utiliser des bibliothèques. Souvent ces bibliothèques regroupent simplement des fonctions et variables définies etinstanciées dans d'autres fichiers.

- ▶ vous pouvez importer des modules présents dans la bibliothèque standard (liste exhaustive <https://docs.python.org/3/library/>)
- ▶ vous pouvez installer des modules puis les importer (liste des paquets ici : <https://pypi.org/>, pour les installer on utilise la commande `python -m pip install nom_du_paquet`)
- ▶ vous pouvez importer des fichiers python que vous avez créé s'ils sont dans le même dossier

## 3- Python

### 3.6. Outils et bonnes pratiques

- ▶ le développement est un travail d'équipe (des fois avec soi-même)
- ▶ le code est plus souvent lu qu'écrit

Un ensemble de pratique permettent de gérer la complexité des projets.

Nous allons en étudier 3 :

- ▶ le coding style
- ▶ le debugging
- ▶ les tests unitaires

Ne pas écrire comme un cochon.

- ▶ des outils (les *linters*) vous permettent de noter les erreurs classiques (par défaut, VSCode utilise pylint)
- ▶ des outils (les *formatters*) vous permettent de modifier votre code pour qu'il respecte les critères de mise en forme classiques (je recommande black, à installer)

En python, les règles classiques à suivre sont précisées dans la PEP 8 :

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

Le super-pouvoir qui vous permet d'arrêter le temps

- ▶ exécution pas à pas d'un programme
- ▶ visualisation des variables à un instant donné
- ▶ certains permettent la modification des variables
- ▶ c'est un peu l'industrialisation des `print` de toutes les variables

Pour ce faire :

- ▶ en ligne de commande avec `pdb`
- ▶ graphiquement avec VSCode (que je privilégie pour des débutants)

Pour vous éviter de vérifier votre programme à chaque fois pour voir s'il a le bon comportement

- ▶ fonctionnent de préférence avec des fonctions pures (une entrée et une sortie dépendante)
- ▶ permet de tester une partie précise d'un programme
- ▶ on va affirmer (*to assert* en anglais) que le résultat attendu est égal au résultat obtenu
- ▶ on utilise le mot clé `assert` ou les méthodes fournies par le module de test

Pour ce faire on utilise :

- ▶ le module `unittest` (inclus dans Python, assez verbeux, approche orienté objet)
- ▶ le module à installer `pytest` (que je recommande, plus simple pour commencer)

# Les tests unitaires – exemple

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

[Aller plus loin](#)

On veut tester le code des fonctions additions qui sont censées faire une addition de 2 nombres.

```
def addition(x, y):  
    return x + y
```

```
def addition_2(x, y):  
    return x + x
```

```
def test_addition():  
    assert addition(1, 2) == 3
```

```
def test_addition_2():  
    assert addition_2(1, 2) == 3
```

# Les tests unitaires – exemple

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

[Aller plus loin](#)

Après avoir installé pytest, avec `python -m pip install --user pytest` on lance la commande `python -m pytest` et voilà le résultat

```
===== test session starts =====
platform linux -- Python 3.6.12, pytest-5.2.1, py-1.8.0, pluggy-0.13.0
rootdir: ../../bonnes_pratiques
plugins: dash-1.8.0, cov-2.8.1
collected 2 items

test_exemple.py .F

===== FAILURES =====
_____ test_addition_2 _____

    def test_addition_2():
>     assert addition_2(1, 2) == 3
E       assert 2 == 3
E         + where 2 = addition_2(1, 2)

test_exemple.py:14: AssertionError
```



Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

**Aller plus loin**

## 3- Python

### 3.7. Aller plus loin

# Ce n'est que le commencement I

Ce que vous avez vu dans ce cours n'est que le commencement de votre parcours de développeur.

Ce qu'il vous reste encore à voir (entre autre et en vrac) :

- ▶ quand utiliser chaque structure de données
- ▶ le développement orienté objet (OOP)
- ▶ le développement fonctionnel
- ▶ la gestion des exceptions
- ▶ l'installation de modules externes
- ▶ les bases de l'administration système pour déployer son programme
- ▶ la compréhension de l'écosystème des modules
- ▶ gérer la collaboration à plusieurs développeurs
- ▶ ...

Bon courage et n'hésitez pas à me demander des informations sur certains sujets.

Matthieu Falce

Hardware

Software

Python

Généralités

Syntaxe

Variables

Structures de base

Fonctions

Outils et bonnes pratiques

Aller plus loin

- ▶ la référence pour les débutants *Automate the Boring Stuff with Python* <https://automatetheboringstuff.com/>
- ▶ <https://jakevdp.github.io/WhirlwindTourOfPython>
- ▶ [https://koor.fr/Python/Tutorial/python\\_introduction.wp](https://koor.fr/Python/Tutorial/python_introduction.wp)
- ▶ <https://courspython.com>
- ▶ <http://sametmax.com>
- ▶ <https://realpython.com>