



Concepts d'IA, Python et GitHub

Dr. NSENGE MPIA HERITIER, Ph.D

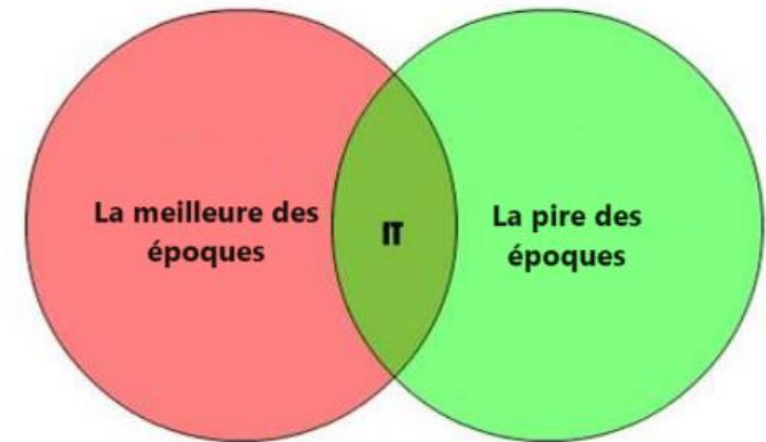
Plan de la leçon



- Concepts fondamentaux
- Représentation des connaissances
- Bases de la logique
 - Logique syllogistique
- Aperçu du raisonnement
 - Déductif, inductif, abductif
- Apprendre à coder en Python
- Aide au codage
- Git et GitHub



Selon Charles Dickens





Les concepts fondamentaux

Vue d'ensemble

- L'une des **caractéristiques de l'intelligence** est la **capacité à raisonner**
- Le **raisonnement** nous permet de **résoudre des problèmes**
- Si nous voulons construire des machines intelligentes, nous devons pouvoir **automatiser le raisonnement**
- La **logique** est l'étude de la manière dont nous raisonnons (correctement)
- Objectifs de la logique :
 - **Représenter** la **connaissance** du monde
 - **Raisonner** à partir de ces **connaissances**

Connaissance

- Définition de la connaissance (débat permanent entre les philosophes)
- En ce qui nous concerne, la définition est la suivante :
 - Des propositions : **Faits, informations, descriptions**
 - Considérées comme **vraies/correctes** ou au moins hautement probables.

- Exemple:

- Un billet de 5000 FC vaut mieux qu'un billet de 1000 FC
 - Un billet de 1000 FC est meilleure qu'un billet de 1 FC

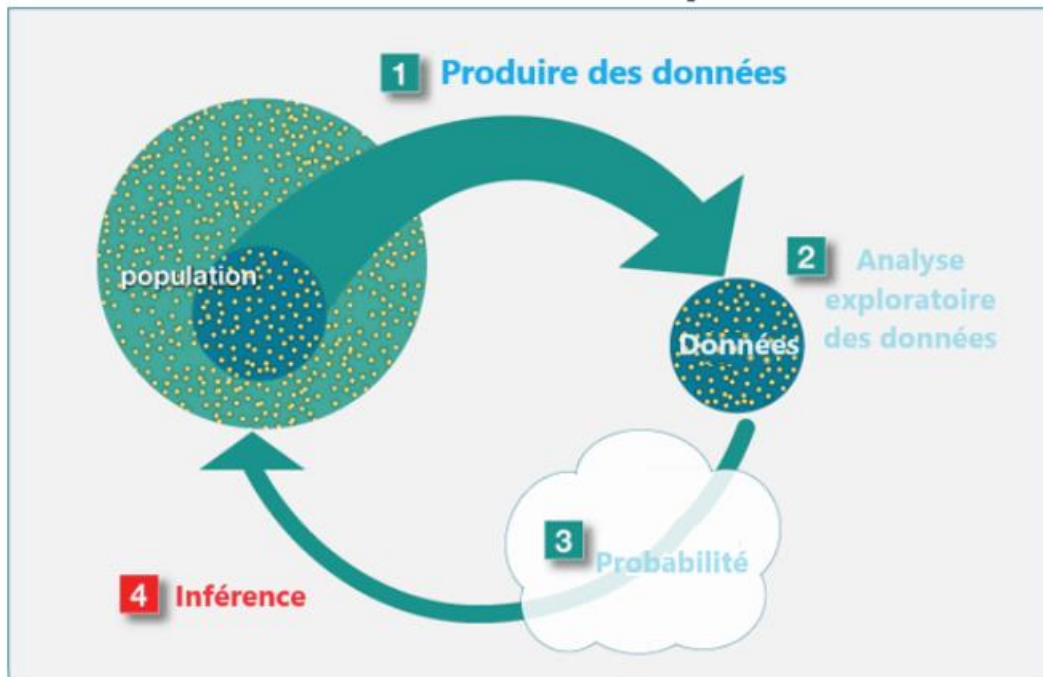


- Connaissances acquises par l'expérience ou l'éducation
 - Percevoir, découvrir ou apprendre
- C'est assez simple, mais.. :
 - Communiquer ?
 - Manipuler les connaissances pour tirer des conclusions et obtenir de nouvelles connaissances ?

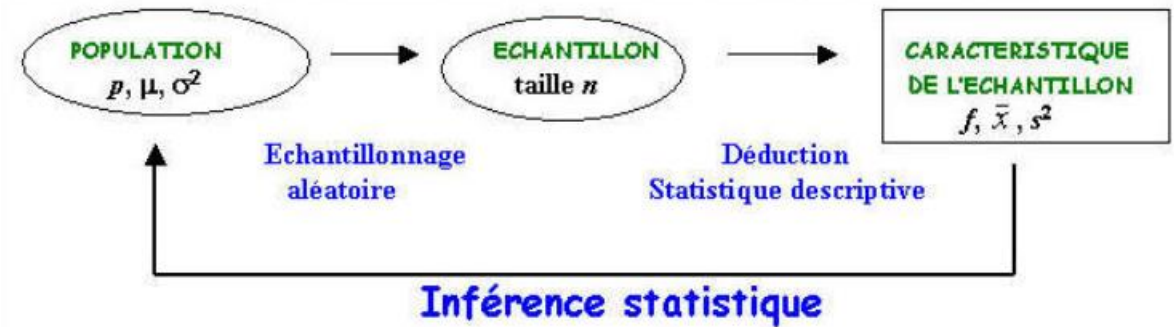
Inférence

Une **conclusion** tirée sur la base de preuves et d'un raisonnement

Inférence statistique



Les **statistiques inférentielles** ou inductives peuvent se résumer par le schéma suivant :



Raisonnement

Conclure (déductions/inférences) à partir de connaissances connues ou supposées.

Conclusions →

Un billet de 5000 FC vaut mieux qu'un billet de 1000 FC

Un billet de 1000 FC est meilleure qu'un billet de 1 FC

} **Connaissance**

Donc, un billet de 5000 FC est meilleur qu'un billet de 1 FC

} **Raisonnement**

- Il existe de nombreuses façons de raisonner
- Voici un exemple de raisonnement logique déductif ci-dessus
 - c'est-à-dire le **syllogisme**



Représentation des connaissances

Raisonner avec le « langage naturel »

- Problématique:



Conclusion invalide →

Un billet de 1 FC vaut mieux que rien

Rien ne vaut la paix dans le monde

} **Connaissance**

Donc, un billet de 1 FC vaut mieux que la paix dans le monde

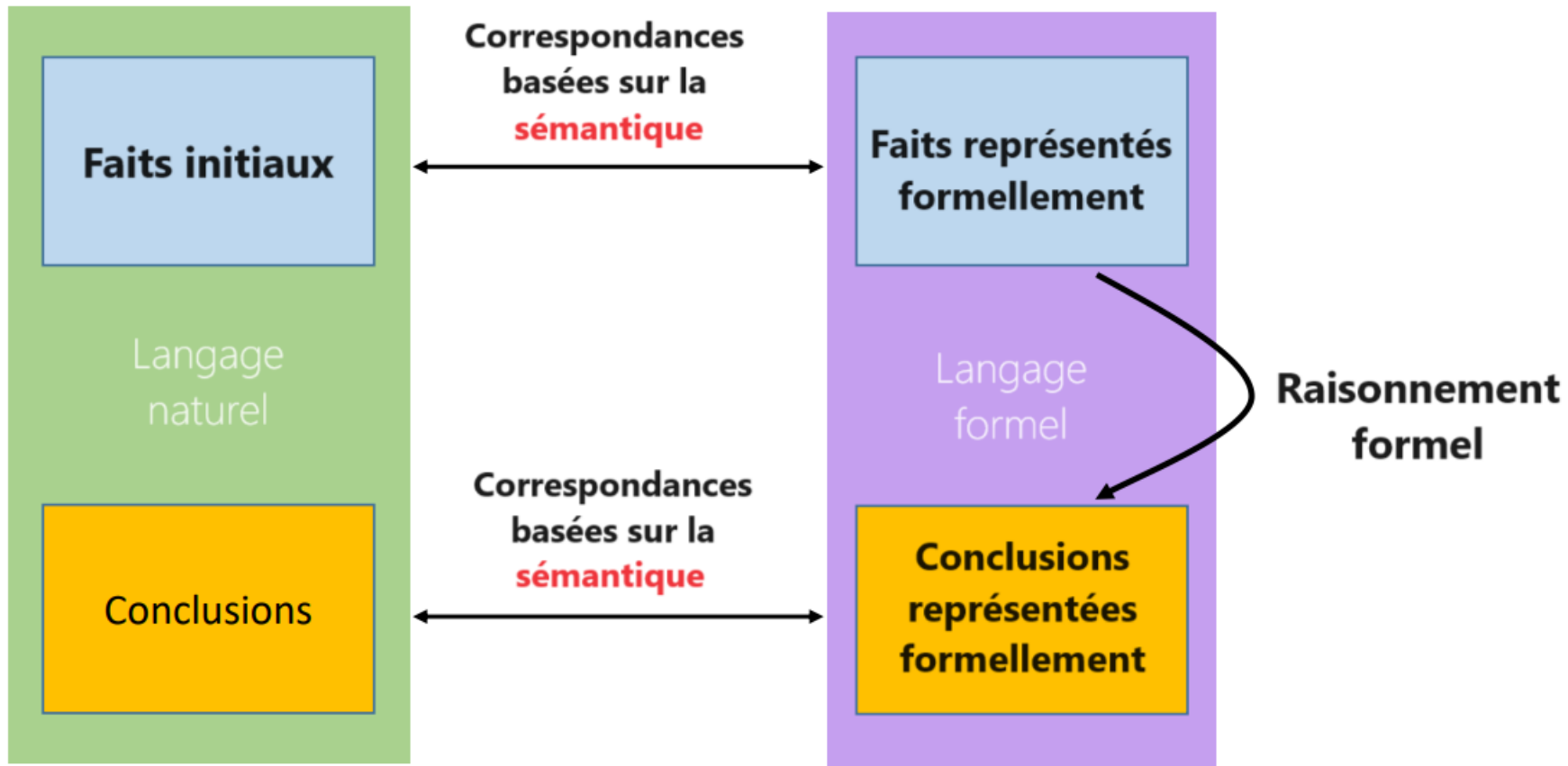
- Le langage naturel est délicat (souvent ambigu).
 - Une base de connaissances excessivement vaste peut être nécessaire pour comprendre le **contexte**.
- Nous avons besoin d'une méthode plus cohérente et contrôlée pour représenter les connaissances afin de garantir :
 - la validité de nos conclusions
 - Un système informatique peut utiliser efficacement les connaissances

- La résolution de problèmes dans un **domaine** particulier nécessite souvent une **connaissance des objets** et de la **manière de raisonner** dans ce domaine.
- **Représentation formelle** :
 - **Normalise** les objets et le raisonnement
 - Exemple :
 - Logique
 - Langages de programmation
 - Ontologies
- Une mise en correspondance (**Mapping**) est nécessaire pour relier le naturel au formel
- Conséquence :
 - Moins de flexibilité

« Le rôle prévu de la représentation des connaissances dans l'IA est de réduire les problèmes d'action intelligente à des **problèmes de recherche** ».

– **Ginsberg (1993)**

Représentation et correspondances (**Mapping**)



Représentation : Structures des symboles

- Les connaissances doivent être représentées :
 - **efficacement**, de manière **cohérente** et **significative**
- Structures de symboles (représentant des éléments de connaissance)
 - Exemple :
 - Le symbole "**bleu**" désigne une couleur particulière.
 - Structure de symboles **bleu(Voiture-Nsenge)** pour indiquer que ma voiture est bleue.
 - Il peut s'agir de n'importe quel support physique
 - Les ordinateurs facilitent les choses en les rendant faciles
 - **Faits** : structures de données (liste, dictionnaire, tableau, cadre de données)
 - **Raisonnement** : code de programme



Structure du symbole de la carte perforée



Langages de programmation

Exigences en matière de représentation des connaissances

- **Adéquation représentationnelle :**
 - Représentation de toutes les connaissances nécessaires au raisonnement
- **Adéquation déductive/inférentielle :**
 - Permettre de déduire de nouvelles connaissances à partir d'un ensemble de faits de base.
 - Ne pas énumérer les points négatifs qui pourraient être couverts par un point positif
- **Efficacité inférentielle :**
 - Les déductions doivent être faites facilement/efficacement
- **Syntaxe et sémantique claires :**
 - Connaître les expressions autorisées de la langue et ce qu'elles signifient.
- **Naturel :**
 - Raisonnement naturel et facile à utiliser

Syntaxe vs sémantique



Syntaxe : La **forme**, la **grammaire** ou la **structure** des expressions, des énoncés et des unités de programme (c'est-à-dire les expressions autorisées).

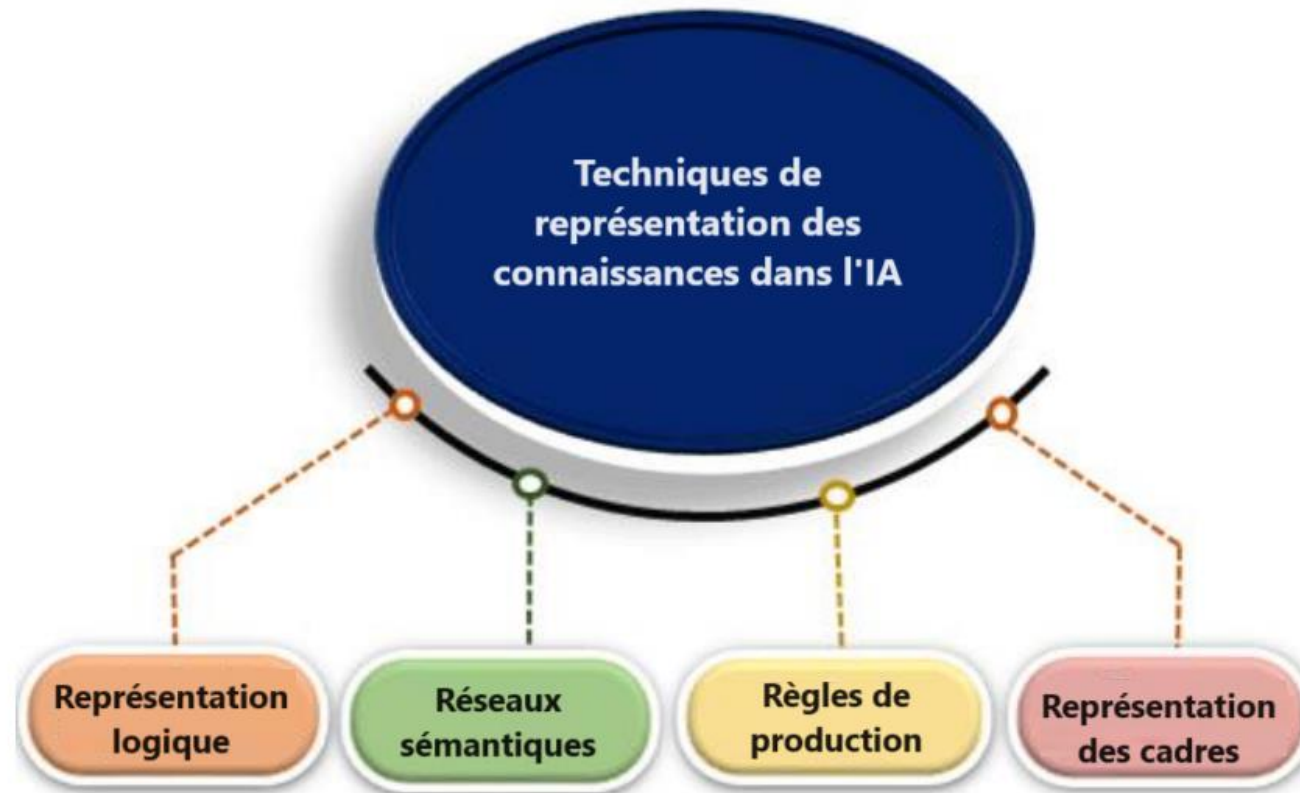
Sémantique : La **signification** des expressions, des énoncés et des unités de programme

- Exemple de syntaxe :
 - **bleu(Voiture-Nsenge)** est OK, mais **Voiture-Nsenge(bleu & noir)** n'est pas OK
- Exemple sémantique :
 - **blue(Voiture-Nsenge)** signifie « **La voiture de Nsenge est bleue** »
 - Mais ne signifie pas (une instruction de) "**Peindre la voiture de Nsenge en bleu**".
 - Très important pour tirer de nouvelles conclusions

Représentation naturelle

- Choisissez des noms significatifs pour les symboles
- **Fait naturel** :
 - "Si quelqu'un a mal à la tête, il doit prendre de l'aspirine".
- Représentation 1 :
 - $Si(X, m, a)$
- Représentation 2 :
 - Si symptôme(X, mal de tête) ALORS médicament(X, aspirine)
- La deuxième représentation est plus lisible et plus facile à traiter.

Principales approches en matière de représentation





Les bases de la logique

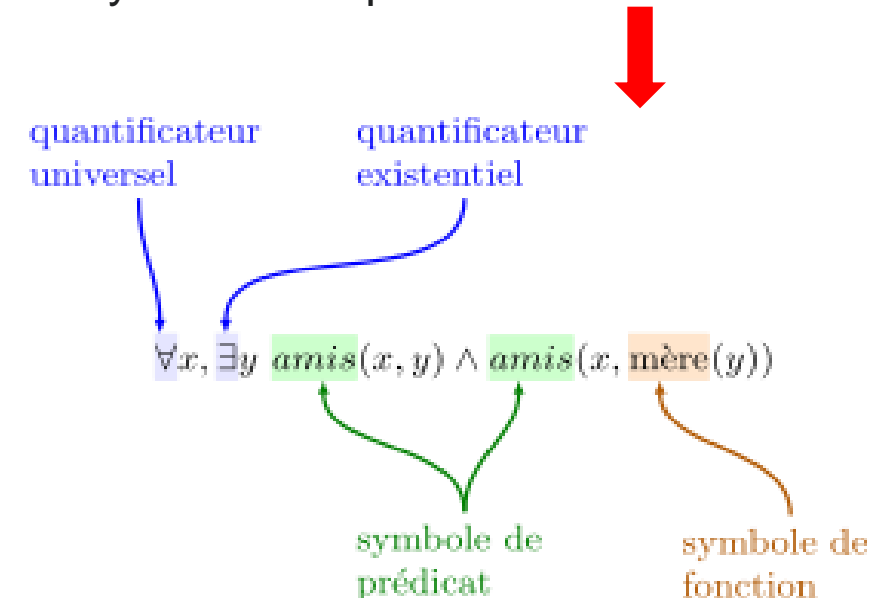
Introduction à la logique

- Objectifs de la logique :
 - **Représenter** la connaissance du monde
 - **Raisonner** à partir de ces connaissances
- La logique est sans doute l'approche la plus importante en matière de représentation de la connaissance
- Une logique (presque par définition)
 - possède une syntaxe et une sémantique bien définies
 - Se préoccupe de l'inférence préservant la vérité
- La logique étudie la façon dont les humains raisonnent formellement à travers l'**argumentation**
- La "logique" se réfère généralement à la **logique propositionnelle** ou à la **logique des prédicats du premier ordre**.

Introduction à la logique (Cont.)

- Alors que la **logique propositionnelle** traite des propositions déclaratives simples, la **logique du premier ordre** couvre en plus les prédicats et la quantification.
- Un **prédicat** prend en entrée une ou plusieurs entités du domaine du discours et en sortie, il est soit **Vrai**, soit **Faux**.
- Considérons les deux phrases « Socrate **est un philosophe** » et « Platon **est un philosophe** ».
- En logique propositionnelle, ces phrases sont considérées comme non liées et peuvent être désignées, par exemple, par des variables telles que **p** et **q**.
- Le prédicat « **est un philosophe** » apparaît dans les deux phrases, dont la structure commune est « X est un philosophe ».
- La variable **X** est instanciée en tant que « Socrate » dans la première phrase, et est instanciée en tant que « Platon » dans la deuxième phrase.
- Alors que la logique du premier ordre permet l'utilisation de prédicats, tels que « est un philosophe » dans cet exemple, la logique propositionnelle ne le permet pas.

Exemple d'une formule de la logique du premier ordre. Le schéma montre les quantificateurs, les occurrences des symboles de fonctions et des symboles de prédicats.



Logique : Argument

- **Argument logique** :
 - Un ensemble (fini) d'énoncés comprenant des **prémisses** (c'est-à-dire des faits, des connaissances, des preuves à l'appui) et une **conclusion**
 - Formule le raisonnement en mots
 - La conclusion est identifiée par le symbole "**donc** " (\therefore)
- Exemple

Si tu dors trop, tu seras en retard.

Tu n'es pas en retard.

\therefore Tu n'as pas trop dormi

- Comment décrire la qualité d'un argument ?

Arguments valides



Argument valide :

Il est impossible que les prémisses soient toutes vraies et que la conclusion soit fausse.

Si tu dors trop, tu seras en retard.

Tu n'es pas en retard.

∴ Tu n'as pas trop dormi

Valide

Si tu dors trop, tu seras en retard.

Tu n'as pas trop dormi.

∴ Tu n'es pas en retard.

Invalide

Si tu es en RDC, tu es en Afrique.

Tu n'es pas en Afrique.

∴ Tu n'es pas en RDC.

Valide

Si tu es en RDC, tu es en Afrique.

Tu n'es pas en RDC.

∴ Tu n'es pas en Afrique.

Invalide

Arguments solides (bons)

Argument solide :

- Est **valide** et **chaque prémisse est vraie**

- Non solide (fondé) (2 façons) :

1. Il repose sur une prémisse erronée
2. La conclusion peut ne pas découler des prémisses

Si tu lis ceci, tu n'es pas analphabète.

Tu lis ceci.

∴ Tu n'es pas analphabète.

Valide et solide

Tous les informaticiens sont des millionnaires.

Nsenge est un informaticien.

∴ Nsenge est millionnaire.

Valide mais non solide

Tous les millionnaires mangent bien.

Nsenge mange bien.

∴ Nsenge est millionnaire.

Invalidé donc non solide

Logique syllogistique

- Inventé par **Aristote**
- Langage syllogistique :
 - Majuscules pour les **catégories générales**
 - Minuscules pour les **éléments/individus spécifiques**

Tous les **informaticiens** sont des **millionnaires**.
Nsenge est un **informaticien**.
 \therefore **Nsenge** est **millionnaire**.



tout **I** est **M**
n est **I**
 \therefore **n** est **M**

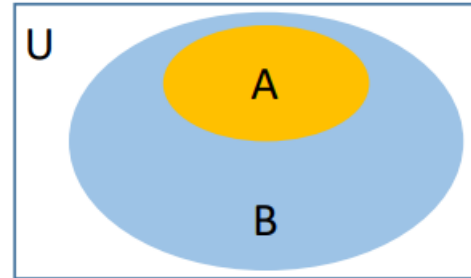
- **Formules bien formées (fbf) :**
 1. Syntaxe (**objets acceptables**) de la logique
 2. Utilise cinq mots : **tout, n'est pas, certain, est, aucun**
 3. Les *fbf* ont l'une de ces **8** formes :

tout A est B	certain A est B	x est A	x est y
Aucun A est B	certain A n'est pas B	x n'est pas A	x n'est pas y

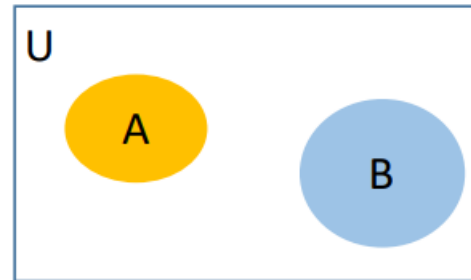
fbf avec diagrammes de Venn

- **Diagramme de Venn** : Un diagramme composé de plusieurs figures qui se chevauchent et qui sont contenues dans un rectangle appelé l'univers (**U**).

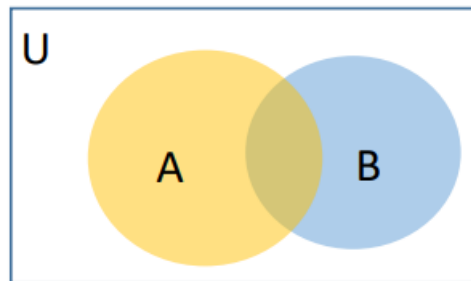
- Tout A est B (Si A, alors B.)



- Aucun A n'est B



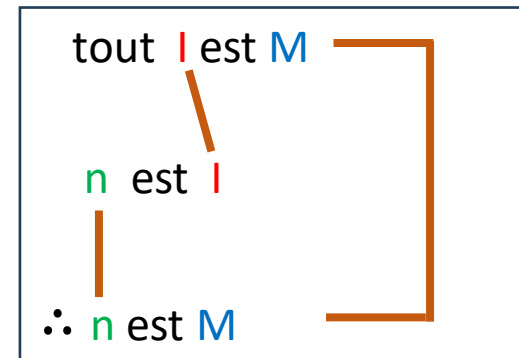
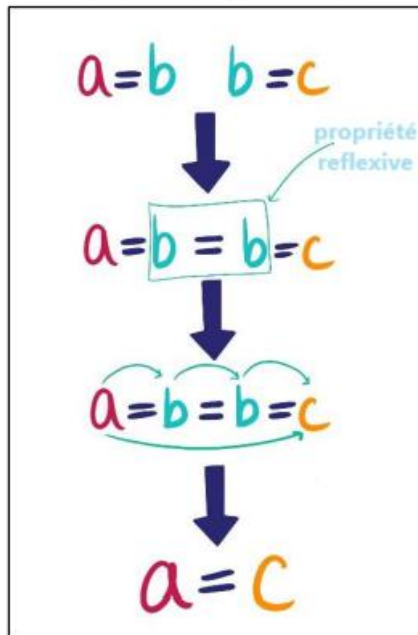
- Certains A sont B



Syllogisme

- Arguments de deux ou plusieurs **fbf** dans lesquels chaque lettre apparaît deux fois et où les lettres "**forment une chaîne**".
- Chaque **fbf** a *au moins une lettre en commun avec* le **fbf** juste en dessous, et le premier **fbf** a au moins une lettre en commun avec le dernier **fbf**.

Propriété transitive



Prémisse majeure

Prémisse mineure

Conclusion

Le test de la star

- Astuce pour tester la validité d'un syllogisme:

- Identifier les lettres "**distribuées**" :

- Apparaît juste après "tous" ou n'importe où après "aucun" ou "ne pas".
- La première lettre après "tout", mais pas la seconde
- Les deux lettres après "aucun".
- N'importe quelle lettre après "ne pas".

tout A est B

certain A est B

x est A

x est y

Aucun A est B

certain A n'est pas B

x n'est pas A

x is not y

- Test de la star:

- Mets les '*' sur les lettres des prémisses qui sont distribuées et lettres de la conclusion qui ne le sont pas
- Le syllogisme est valide si:
 - chaque lettre majuscule est étoilée exactement une fois
 - Il y a exactement une étoile sur le côté droit

tout I* est M

n est I

∴ n est M*

Valide

aucun A* est B*

aucun C* est A*

∴ aucun C est B

Invalide

aucun P* est B*

certain C est B

∴ certain C* n'est pas P

Valide

Éthymème



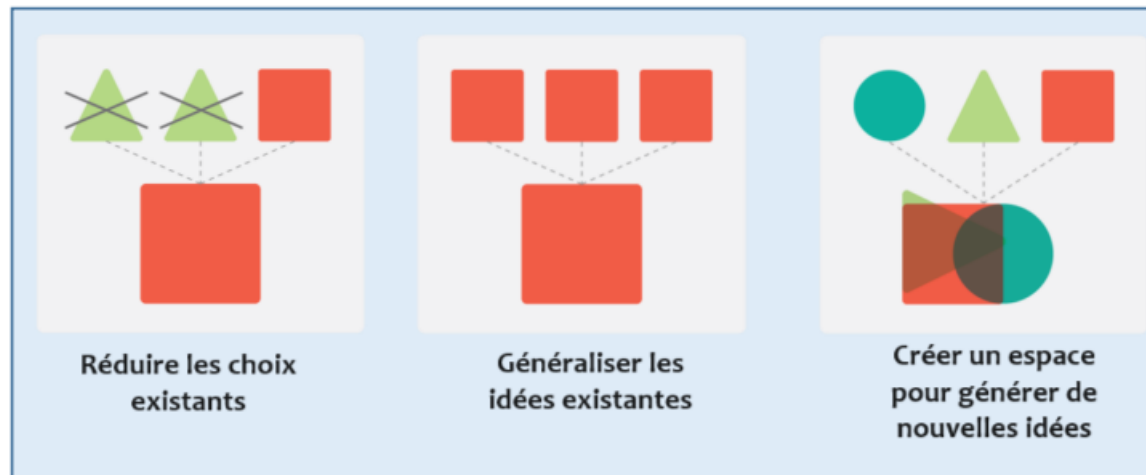
- Argument dans lequel la **prémisse majeure** n'est pas énoncée.
 - Souvent une conclusion soutenue par une seule prémisse
- Exemples:
 - Elle doit être une bonne étudiante puisqu'elle figure sur la liste du Prof
[Conclusion] [Prémisse mineure]
 - Elle doit être une bonne étudiante puisqu'elle figure sur la liste du Prof
[Implique prémisse majeure]
Tous les bons étudiants figurent sur la liste du Prof



Aperçu du raisonnement

Types de raisonnement

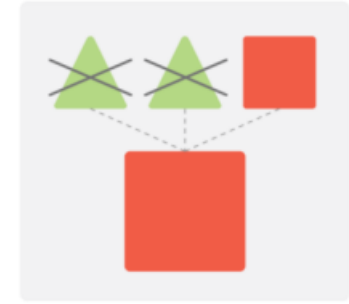
Déduction	Induction	Abduction
Tous les hommes sont mortels	Socrate est un homme	Tous les hommes sont mortels
Socrate est un homme	Socrate est mortel	Socrate est mortel
∴ Socrate est mortel	∴ Tous les hommes sont mortels	∴ Socrate est un homme



Général → Spécifique

Spécifique → Général

Raisonnement déductif



- Général \rightarrow Spécifique c'est-à-dire "Logique descendante"
- **Raisonnement à partir d'une ou plusieurs affirmations pour parvenir logiquement à *certaine conclusion***
- Le raisonnement va dans le même sens que celui des conditionnels
 - L'inverse est vrai pour le *raisonnement abductif*
- Une conclusion est obtenue de manière *réductrice* :
 - Appliquer des règles générales qui s'appliquent à l'ensemble d'un domaine fermé, en réduisant l'éventail considéré jusqu'à ce qu'il ne reste que la (les) conclusion(s).
- Les *sylogismes* sont une forme courante de raisonnement déductif.
- Caractéristique des *systèmes experts*

Raisonnement inductif



- Spécifique → général, c'est-à-dire "logique ascendante".
- Raisonner à partir de **preuves** concernant certains membres d'une **classe** afin de formuler une conclusion concernant tous les membres d'une classe.
- Une conclusion tirée par un raisonnement inductif est appelée "**hypothèse**".
- Toujours moins sûre que la preuve elle-même
 - c'est-à-dire que la conclusion est **probable mais incertaine**
- Le raisonnement inductif n'est pas ici le même que l'induction dans les preuves mathématiques
 - L'induction mathématique est en fait une forme de **raisonnement déductif**.
- **L'apprentissage automatique (ML)** repose sur le raisonnement inductif
 - Utilisation d'un ensemble de cas spécifiques pour trouver des généralisations utiles

Raisonnement inductif (Cont.)

- **Syllogisme statistique :**
 - **Lien correct entre les prémisses et la conclusion : Fort ou faible**
 - **Prémisses vraies : Convaincant/fiable vs Non convaincant/non fiable**

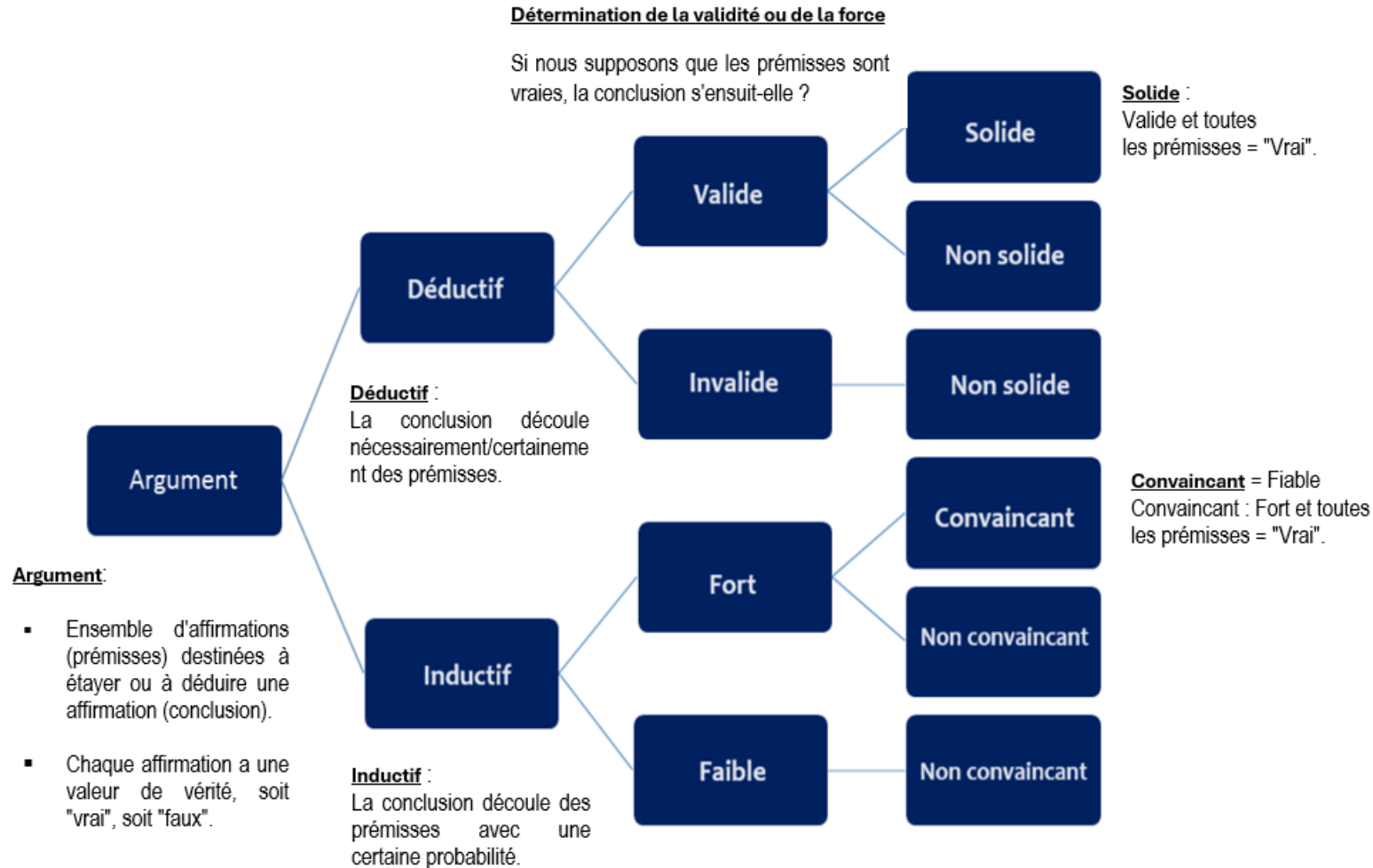
90 % des villes congolaises n'ont pas d'eau potable.

Kinshasa est une ville congolaise.

C'est tout ce que nous savons à ce sujet.

∴ Il est probable à 90% que Kinshasa n'ait pas d'eau potable

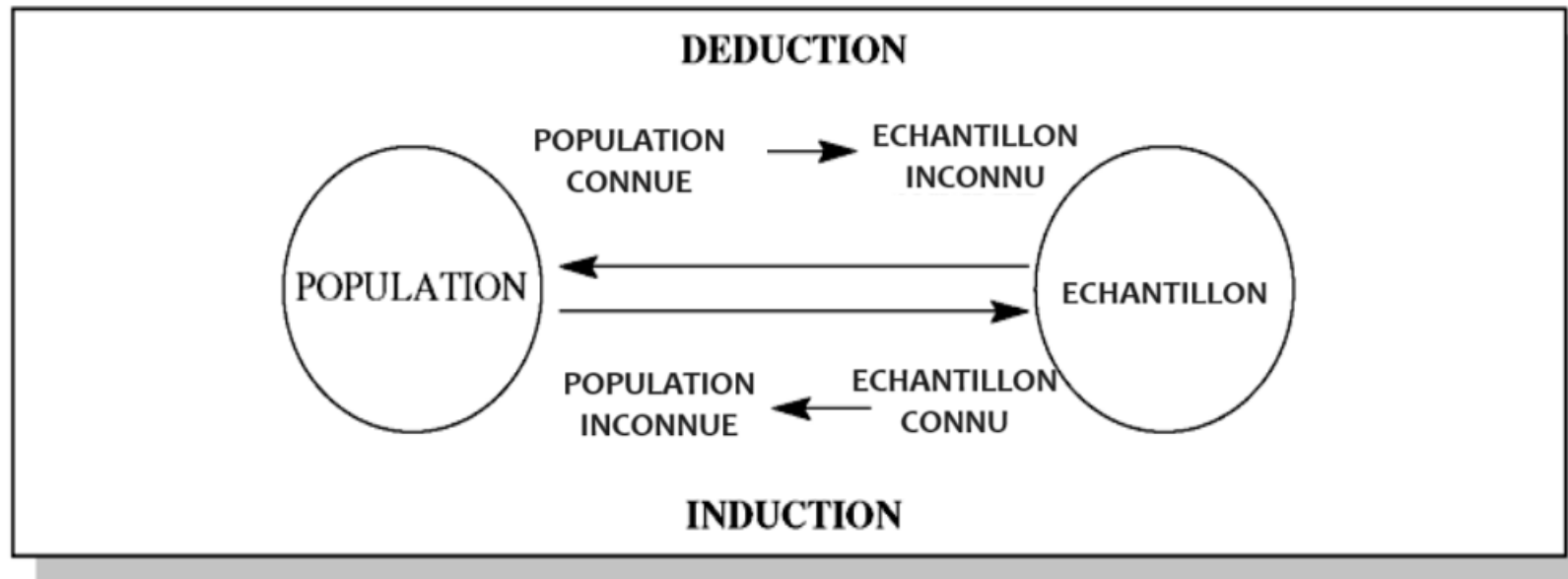
Qualité des arguments de raisonnement



Source de l'image: Patrick J. Hurley, « A concise introduction to Logic », 12^e Ed.

Déduction et induction

- Dans le contexte des populations et des échantillons



Apprentissage déductif ou inductif

- Apprentissage/instruction déductif :
 - L'enseignant présente une généralisation/règle
 - L'apprenant l'applique à des exemples ou activités spécifiques
- Apprentissage/instruction inductif :
 - Les apprenants détectent ou remarquent des modèles et élaborent une "règle" pour eux-mêmes.

Déduction

Généralisation (ou Règle) —————> Exemples ou activités spécifiques

Induction

Exemples ou activités spécifiques —————> Généralisation (ou Règle)



Raisonnement abductif

- Déduction de la meilleure explication
- Part d'une observation ou d'un ensemble d'observations, puis cherche à **trouver l'explication la plus simple et la plus probable pour les observations**
- Permet d'expliquer un **phénomène** ou une observation à partir de certains **faits**.
- C'est la recherche des causes, ou d'une hypothèse explicative.
- Permet d'aboutir à une conclusion plausible mais ne la vérifie pas de façon certaine.
 - " **Meilleure information disponible**" ou " **la plus probable**".
- Parfois utilisé par **les systèmes experts de diagnostic**



Apprendre à coder en Python

Programmation en Python : Editeurs et IDE

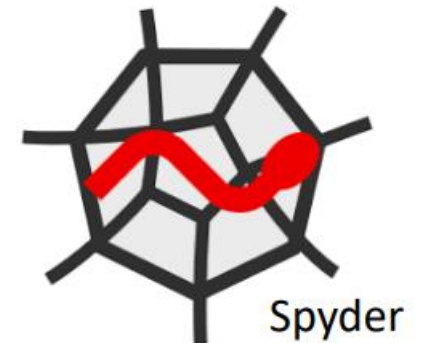
- Éditeur (c'est-à-dire éditeur de texte) :
 - Il s'agit simplement d'un endroit où le texte et le code peuvent être écrits et modifiés.
- Environnement de développement intégré (IDE) :
 - Une application logicielle qui fournit des facilités complètes aux programmeurs informatiques pour le développement de logiciels.
 - Éditeur de code source
 - Outils d'automatisation de la construction
 - Débogueur.

• Éditeurs généraux et IDE avec prise en charge de Python

- Eclipse + PyDev
- Sublime Text
- Atom
- GNU Emacs
- Vi / Vim
- Visual Studio
- Visual Studio Code
- Jupyter Notebook

• IDEs spécifiques de Python :

- PyCharm
- Spyder
- Thonny
- IDLE



Tutoriels d'introduction au codage Python

- Les "bases" autoguidées :
 - <https://docs.python.org/fr/3/tutorial/>
- Vidéo (notions de base) :
 - Lien : <https://www.youtube.com/watch?v=oUJolR5bX6g>
 - Pandas (science des données) :
 - Lien : <https://www.python-simple.com/python-pandas/panda-intro.php>
- Scikit-Learn (apprentissage automatique) :
 - Lien: <https://www.data-transitionnumerique.com/scikit-learn-python/>
- Cours :Introduction à la programmation Python (BAC1)

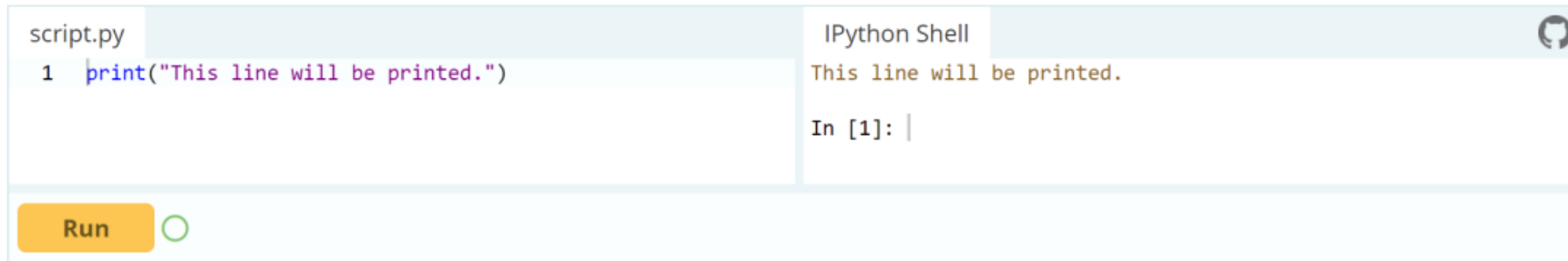


Programmation en python: Leçons introductives

- [Variables et types](#)
- [Listes](#)
- [Les opérateurs basiques](#)
- [Formatage des chaînes des caractères](#)
- [Opérations basiques sur les chaînes des caractères](#)
- [Les conditions](#)
- [Les boucles « while »](#)
- [Les fonctions](#)
- [Les classes et les objets](#)
- [Les dictionnaires](#)
- [Les modules et les packages](#)

Print et indentation

- Print (en Python 3)



The screenshot shows a Jupyter Notebook interface. On the left, a code editor window titled 'script.py' contains the following code:

```
1 print("This line will be printed.")
```

Below the code editor is a yellow 'Run' button with a green circle icon. On the right, the 'IPython Shell' window displays the output of the code:

```
This line will be printed.
```

Below the output, the prompt 'In [1]: |' is visible.

- Indentation :

- Les "blocs" de code sont délimités par des indentations (c'est-à-dire des tabulations), plutôt qu'avec des accolades "{ }" (comme dans R).



The screenshot shows a code editor window titled 'script.py' containing the following code:

```
1 x = 1
2 if x == 1:
3     # indented four spaces
4     print("x is 1.")
```

Variables et affectation

- Variables

- Python est entièrement orienté objet et n'est pas "statiquement typé"
- Il n'est pas nécessaire de déclarer les variables avant de les utiliser, ni de déclarer leur type (comme en Java ou C++)
- Chaque variable en Python est un objet.
- Les variables sont définies par l'affectation d'une valeur initiale
- Les variables peuvent avoir (presque) n'importe quel nom
 - Lien : [Tutoriel sur l'affectation des variables \(règles et conventions\)](#)
 - Règles :
 - Les noms de variables doivent commencer par une lettre ou par "_"
 - Les noms de variables sont sensibles à la casse
 - Conventions :
 - Les noms doivent maximiser la lisibilité
 - Par exemple, my_variable vs myvariable

- Affectation

- Les variables sont définies en leur attribuant une valeur (initiale)
- Par exemple : `ma_variable = 0`, `ma_variable = autre_variable`

Types et casting

- Types
 - Nombres
 - Integer - (int) (par exemple 3, 38, 404049)
 - Flottant - (float) (par exemple 7.140)
 - String (c'est-à-dire du texte plutôt que des valeurs)
 - Spécifiées avec des guillemets simples (' ') ou doubles (" ") (par exemple, "hello")
- Casting: Convertit une variable d'un type à un autre
 - `int(5.7) → 5`
 - `str(5.7) → '5.7'`
 - `float(5) → 5.0`

Listes

- Les listes sont très similaires aux tableaux
 - Les listes sont des conteneurs pour des éléments ayant des **types de données différents**, alors que les tableaux sont utilisés comme conteneurs pour des éléments ayant le même type de données
 - Lien : [Différences entre les listes et les tableaux](#)
- Elles peuvent contenir n'importe quel type de variable, et elles peuvent contenir autant de variables autant de variables que vous le souhaitez
- Les listes peuvent également être itérées d'une manière très simple.

script.py	IPython Shell
<pre>1 mylist = [] 2 mylist.append(1) 3 mylist.append(2) 4 mylist.append(3) 5 print(mylist[0]) # prints 1 6 print(mylist[1]) # prints 2 7 print(mylist[2]) # prints 3 8 9 # prints out 1,2,3 10 for x in mylist: 11 print(x)</pre>	<pre>1 2 3 1 2 3 In [1]: </pre>

Opérateurs arithmétiques de base



script.py	IPython Shell
<pre>1 number = 1 + 2 * 3 / 4.0 2 print(number)</pre>	<pre>2.5</pre>

script.py	IPython Shell
<pre>1 remainder = 11 % 3 2 print(remainder)</pre>	<pre>2</pre>

script.py	IPython Shell
<pre>1 squared = 7 ** 2 2 cubed = 2 ** 3 3 print(squared) 4 print(cubed)</pre>	<pre>49 8 In [1]:</pre>

Opérateurs avec les chaînes et les listes



script.py

```
1 helloworld = "hello" + " " + "world"
2 print(helloworld)
```

IPython Shell

hello world

In [1]: |

script.py

```
1 lotsofhellos = "hello" * 10
2 print(lotsofhellos)
```

IPython Shell

hellohellohellohellohellohellohellohellohello

In [1]: |

script.py

```
1 even_numbers = [2,4,6,8]
2 odd_numbers = [1,3,5,7]
3 all_numbers = odd_numbers + even_numbers
4 print(all_numbers)
```

IPython Shell

[1, 3, 5, 7, 2, 4, 6, 8]

In [1]: |

Formatage des chaînes de caractères

- L'opérateur "%" est utilisé pour *formater un ensemble de variables incluses dans un "tuple"* (une liste de taille fixe) ainsi qu'une chaîne de format, qui contient du texte normal ainsi que des "spécificateurs d'arguments", des symboles spéciaux comme "%s" et "%d".

script.py

```
1 # This prints out "Hello, John!"  
2 name = "John"  
3 print("Hello, %s!" % name)
```

IPython Shell

Hello, John!

In [1]: |

script.py

```
1 # This prints out "John is 23 years old."  
2 name = "John"  
3 age = 23  
4 print("%s is %d years old." % (name, age))
```

IPython Shell

John is 23 years old.

In [1]: |

script.py

```
1 # This prints out: A list: [1, 2, 3]  
2 mylist = [1,2,3]  
3 print("A list: %s" % mylist)
```

IPython Shell

A list: [1, 2, 3]

In [1]: |

Autres opérateurs de chaînes de caractères

len()

Python utilise base
zéro indexation

index()

Count()

[start, stop, step]

Chaîne inversée : L'indice
-1 fait référence au dernier
élément d'une liste

```
script.py
1 astring = "Hello world!"
2 print("single quotes are ' '")
3
4 print(len(astring))
```

```
IPython Shell
single quotes are ' '
12
single quotes are ' '
12
```

```
script.py
1 astring = "Hello world!"
2 print(astring.index("o"))
```

```
IPython Shell
4
In [1]: |
```

```
script.py
1 astring = "Hello world!"
2 print(astring.count("l"))
```

```
IPython Shell
3
In [1]: |
```

```
script.py
1 astring = "Hello world!"
2 print(astring[3:7])
3 print(astring[3:7:1])
```

```
IPython Shell
lo w
lo w
In [1]: |
```

```
script.py
1 astring = "Hello world!"
2 print(astring[::-1])
```

```
IPython Shell
!dlrow olleH
In [1]: |
```

```
script.py
1 astring = "Hello world!"
2 print(astring.upper())
3 print(astring.lower())
```

```
IPython Shell
HELLO WORLD!
hello world!
In [1]: |
```


Conditions (if, else)

- Python utilise des variables booléennes pour évaluer les conditions. Les valeurs booléennes **True** et **False** sont renvoyées lorsqu'une expression est comparée ou évaluée.

```
script.py
1 x = 2
2 print(x == 2) # prints out True
3 print(x == 3) # prints out False
4 print(x < 3) # prints out True
```

```
IPython Shell
True
False
True
In [1]: |
```

```
script.py
1 name = "John"
2 age = 23
3 if name == "John" and age == 23:
4     print("Your name is John, and you are also 23 years
5     old.")
6 if name == "John" or name == "Rick":
7     print("Your name is either John or Rick.")
```

```
IPython Shell
Your name is John, and you are also 23 years old.
Your name is either John or Rick.
In [1]: |
```

```
script.py
1 statement = False
2 another_statement = True
3 if statement is True:
4     # do something
5     pass
6 elif another_statement is True: # else if
7     # do something else
8     pass
9 else:
10    # do another thing
11    pass
```

```
script.py
1 x = 2
2 if x == 2:
3     print("x equals two!")
4 else:
5     print("x does not equal to two.")
```

```
IPython Shell
x equals two!
In [1]: |
```

Conditions (in, is, not)



in

```
script.py
1 name = "John"
2 if name in ["John", "Rick"]:
3     print("Your name is either John or Rick.")
```

IPython Shell

Your name is either John or Rick.

In [1]: |

- Contrairement à l'opérateur d'égalité double "==", l'opérateur "is" ne correspond pas aux valeurs des variables, mais aux instances elles-mêmes.

is

```
script.py
1 x = [1,2,3]
2 y = [1,2,3]
3 print(x == y) # Prints out True
4 print(x is y) # Prints out False
```

IPython Shell

True

False

In [1]: |

not

```
script.py
1 print(not False) # Prints out True
2 print((not False) == (False)) # Prints out False
```

IPython Shell

True

False

In [1]: |

Boucle For

- Les boucles For itèrent sur une séquence donnée

script.py

```
1 primes = [2, 3, 5, 7]
2 for prime in primes:
3     print(prime)
```

IPython Shell

```
2
3
5
7
```

range()

(Start, stop, step)

script.py

```
1 # Prints out the numbers 0,1,2,3,4
2 for x in range(5):
3     print(x)
4
5 # Prints out 3,4,5
6 for x in range(3, 6):
7     print(x)
8
9 # Prints out 3,5,7
10 for x in range(3, 8, 2):
11     print(x)
```

IPython Shell

```
0
1
2
3
4
3
4
5
3
5
7
```

In [1]: |

Boucle While

- Les boucles While se répètent tant qu'une certaine condition booléenne est remplie.

script.py	IPython Shell
1 # Prints out 0,1,2,3,4	0
2	1
3 count = 0	2
4 while count < 5:	3
5 print(count)	4
6 count += 1 # This is the same as count = count + 1	
	In [1]:

Break et Continue

- **Break** est utilisé pour sortir d'une boucle "for" ou d'une boucle "while".
- **Continue** est utilisé pour sauter le bloc en cours et revenir à l'instruction "for" ou "while".

script.py

```
1  # Prints out 0,1,2,3,4
2
3  count = 0
4  while True:
5      print(count)
6      count += 1
7      if count >= 5:
8          break
9
10 # Prints out only odd numbers - 1,3,5,7,9
11 for x in range(10):
12     # Check if x is even
13     if x % 2 == 0:
14         continue
15     print(x)
```

Fonctions



- Un moyen pratique de diviser votre code en blocs utiles :
 - Ordonner notre code
 - le rendre plus lisible
 - Le réutiliser et gagner du temps
- Un moyen essentiel de définir des interfaces pour que les programmeurs puissent partager leur code.

script.py	IPython Shell
<pre>1 # Define our 3 functions 2 def my_function(): 3 print("Hello From My Function!") 4 5 def my_function_with_args(username, greeting): 6 print("Hello, %s , From My Function!, I wish you %s"% 7 (username, greeting)) 8 9 def sum_two_numbers(a, b): 10 return a + b 11 12 # print(a simple greeting) 13 my_function() 14 15 #prints - "Hello, John Doe, From My Function!, I wish you 16 a great year!" 17 my_function_with_args("John Doe", "a great year!") 18 19 # after this line x will hold the value 3! 20 x = sum_two_numbers(1,2)</pre>	<pre>Hello From My Function! Hello, John Doe , From My Function!, I wish you a great year! In [1]: </pre>

Classes et objets



- Les objets sont une encapsulation de variables et de fonctions en une seule entité
- Les objets obtiennent leurs variables et leurs fonctions à partir des classes
- Les classes sont essentiellement un modèle pour créer vos objets.

Accès à la variable d'un objet

script.py	IPython Shell
<pre>1 class MyClass: 2 variable = "blah" 3 4 def function(self): 5 print("This is a message inside the class.") 6 7 myobjectx = MyClass() 8 myobjecty = MyClass() 9 10 myobjecty.variable = "yackity" 11 12 # Then print out both values 13 print(myobjectx.variable) 14 print(myobjecty.variable)</pre>	<pre>blah yackity In [1]: </pre>

Accès à la fonction d'un objet

script.py	IPython Shell
<pre>1 class MyClass: 2 variable = "blah" 3 4 def function(self): 5 print("This is a message inside the class.") 6 7 myobjectx = MyClass() 8 9 myobjectx.function()</pre>	<pre>In [1]: </pre>

Dictionnaires



- Un type de données similaire aux tableaux, mais qui fonctionne avec des clés et des valeurs au lieu d'index.
- Chaque valeur stockée dans un dictionnaire est accessible à l'aide d'une clé, qui est un type d'objet quelconque (une chaîne de caractères, un nombre, une liste, etc.), au lieu d'utiliser son index pour l'adresser.

script.py	IPython Shell
<pre>1 phonebook = {} 2 phonebook["John"] = 938477566 3 phonebook["Jack"] = 938377264 4 phonebook["Jill"] = 947662781 5 print(phonebook)</pre>	<pre>{'John': 938477566, 'Jill': 947662781, 'Jack': 938377264} In [1]: </pre>

script.py	IPython Shell
<pre>1 phonebook = { 2 "John" : 938477566, 3 "Jack" : 938377264, 4 "Jill" : 947662781 5 } 6 print(phonebook)</pre>	<pre>{'John': 938477566, 'Jill': 947662781, 'Jack': 938377264} In [1]: </pre>

script.py	IPython Shell
<pre>1 phonebook = {"John" : 938477566, "Jack" : 938377264, "Jill" : 947662781} 2 for name, number in phonebook.items(): 3 print("Phone number of %s is %d" % (name, number))</pre>	<pre>Phone number of John is 938477566 Phone number of Jill is 947662781 Phone number of Jack is 938377264</pre>

Dictionnaire : Suppression d'une valeur



del

script.py

```
1 ▾ phonebook = {  
2     "John" : 938477566,  
3     "Jack" : 938377264,  
4     "Jill" : 947662781  
5 }  
6 del phonebook["John"]  
7 print(phonebook)
```

IPython Shell

```
{'Jill': 947662781, 'Jack': 938377264}
```

In [1]: |

pop()

script.py

```
1 ▾ phonebook = {  
2     "John" : 938477566,  
3     "Jack" : 938377264,  
4     "Jill" : 947662781  
5 }  
6 phonebook.pop("John")  
7 print(phonebook)
```

IPython Shell

```
{'Jill': 947662781, 'Jack': 938377264}
```

In [1]: |

Modules

- Un morceau d'une application doté d'une fonctionnalité spécifique
- Chaque module est un fichier différent, qui peut être édité séparément.
- En Python, les *modules sont simplement des fichiers Python portant l'extension .py*
- Le nom du module sera le nom du fichier.
- Un module Python peut avoir un ensemble de fonctions, de classes ou de variables définies et mises en œuvre
- Les modules sont importés d'autres modules à l'aide de la commande *import* pour importer un module.

```
script.py
1  # game.py
2  # import the draw module
3  import draw
4
5  def play_game():
6      ...
7
8  def main():
9      result = play_game()
10     draw.draw_game(result)
11
12  # this means that if this script is executed, then
13  # main() will be executed
14  if __name__ == '__main__':
15      main()
```

Modules (importation)

Importer un objet du module

```
script.py
1 # game.py
2 # import the draw module
3 from draw import draw_game
4
5 def main():
6     result = play_game()
7     draw_game(result)
```

Importer tous les objets du module

```
script.py
1 # game.py
2 # import the draw module
3 from draw import *
4
5 def main():
6     result = play_game()
7     draw_game(result)
```

Personnalisé le nom de
le l'objet importé

```
script.py
1 # game.py
2 # import the draw module
3 if visual_mode:
4     # in visual mode, we draw using graphics
5     import draw_visual as draw
6 else:
7     # in textual mode, we print out text
8     import draw_textual as draw
9
10 def main():
11     result = play_game()
12     # this can either be visual or textual depending on
    visual_mode
13     draw.draw_game(result)
```

Packages

- Les **paquets** (packages) sont des espaces de noms qui contiennent plusieurs paquets et modules. Il s'agit simplement de répertoires, mais avec une **particularité**.
- Chaque paquetage en Python est un répertoire qui doit contenir un fichier spécial appelé **`__init__.py`**. Ce fichier peut être vide, et il indique que le répertoire qu'il contient est un paquetage Python, il peut donc être importé de la même manière qu'un module.
- Si nous créons un répertoire appelé **`foo`**, qui indique le nom du paquetage, nous pouvons alors créer un module à l'intérieur de ce paquetage appelé **`bar`**. Il ne faut pas non plus oublier d'ajouter le fichier **`__init__.py`** dans le répertoire **`foo`**.

Importation de paquets

Répertoire = **`foo`**
Module = **`bar`**

```
script.py  
1 import foo.bar
```

```
script.py  
1 from foo import bar
```

Paquets couramment utilisés

- **NumPy**: pour effectuer des opérations numériques sur les données.
- **Pandas**: pour explorer et manipuler les données.
- **Matplotlib**: pour créer des visualisations de vos résultats.
- **scikit-learn**: également appelé sklearn, pour construire et analyser des modèles d'apprentissage automatique.

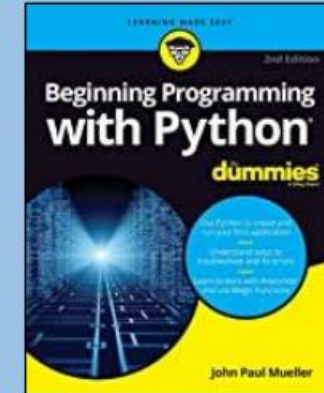
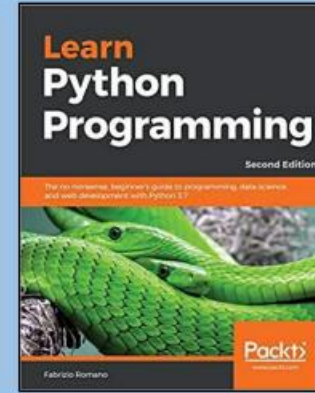
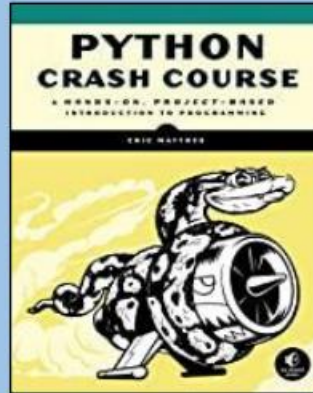
Pandas et NumPy

- NumPy :
 - Une bibliothèque pour le langage de programmation Python, ajoutant la prise en charge des tableaux et matrices multidimensionnels de grande taille ainsi qu'une collection de fonctions mathématiques de haut niveau permettant d'opérer sur ces tableaux
 - Les tableaux Numpy sont d'excellentes alternatives aux listes Python
 - Les principaux avantages des tableaux Numpy sont: rapides, faciles à utiliser et permettent aux utilisateurs d'effectuer des calculs sur des tableaux entiers.
- Pandas :
 - Un outil de manipulation de données de haut niveau.
 - Il s'appuie sur le package Numpy et sa structure de données clé est appelée DataFrame
 - Les DataFrames permettent de stocker et de manipuler des données tabulaires en lignes d'observations et colonnes de variables

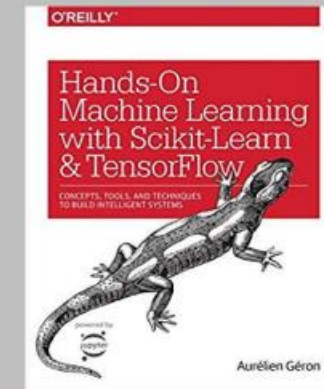
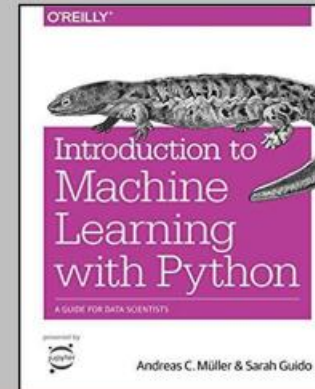
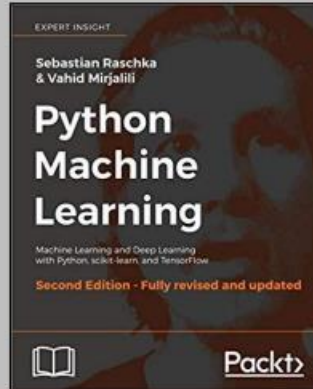
Livres sur Python



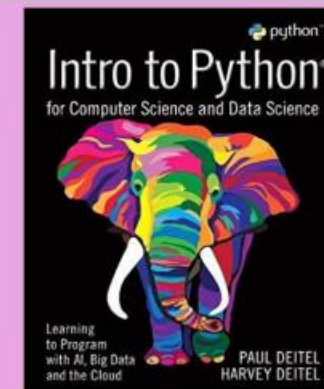
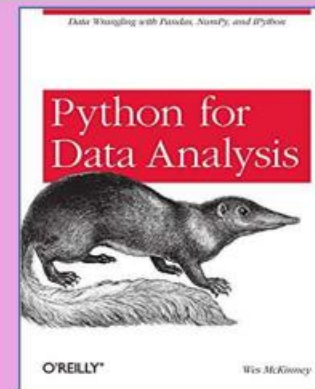
- Python Basics:



- Python Machine Learning:



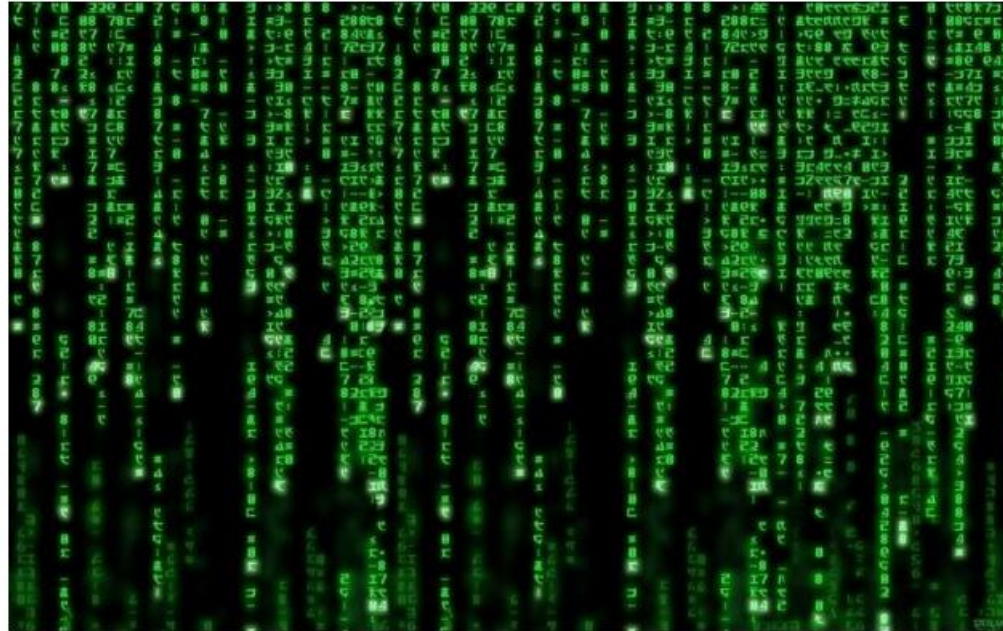
- Python Data Science:





Aide au codage

Obtenir de l'aide pour coder



- Être un data scientist, c'est aussi être **capable de chercher des réponses par soi-même**
 - Faire des efforts
 - Ne pas avoir peur des données/erreurs
 - Admettre que l'on ne sait pas quelque chose n'est pas grave
- La plupart des questions que vous vous posez ont déjà trouvé une réponse sur le web.

Où trouver de l'aide



- La partie la plus difficile peut être de trouver ce qu'il faut chercher

Où d'autre trouver de l'aide pour coder/déboguer ?

- <https://stackoverflow.com>
- <https://www.geeksforgeeks.org>
- <https://python-forum.io/index.php>
- <https://fr.quora.com>





Git et Github

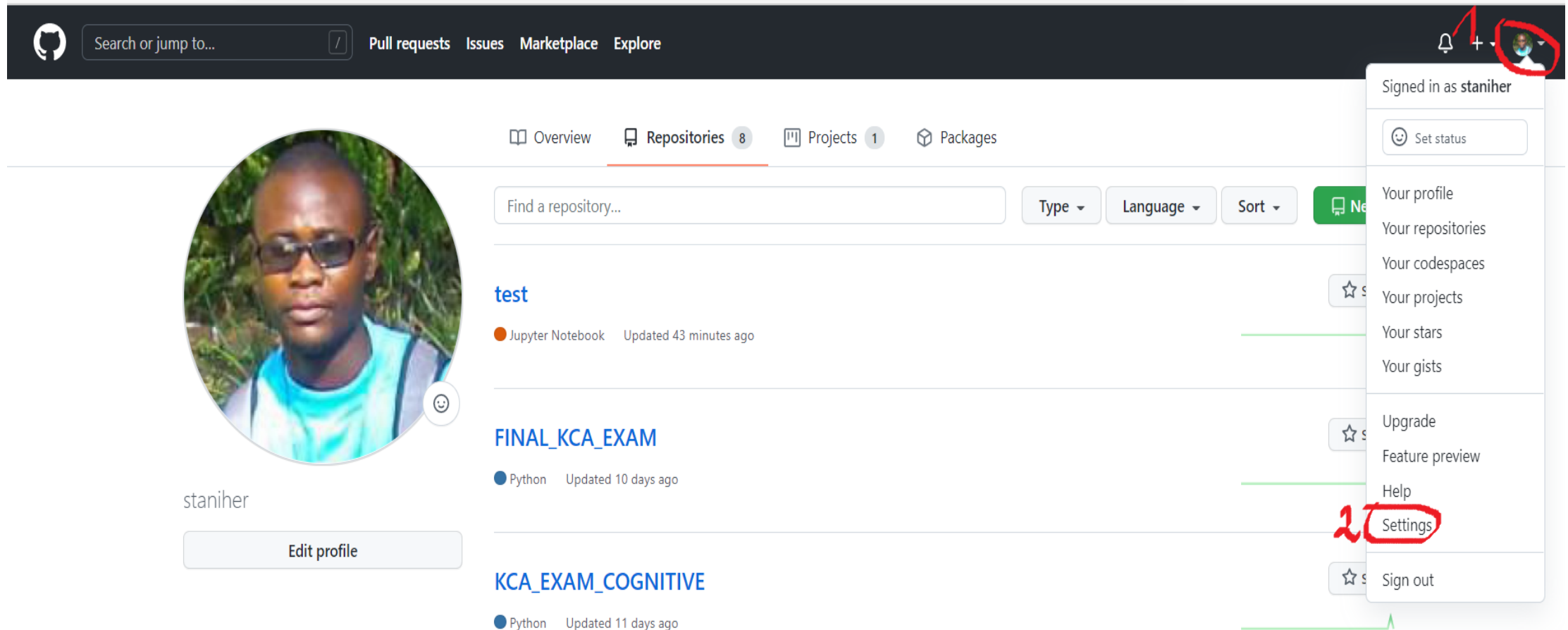
Git

- Git est de loin le système de contrôle de version le plus largement utilisé aujourd'hui. Git est un projet open source avancé, qui est activement maintenu.
- À l'origine, il a été développé en 2005 par Linus Torvalds, le créateur bien connu du noyau du système d'exploitation Linux.
- De plus en plus de projets logiciels reposent sur Git pour le contrôle de version, y compris des projets commerciaux et en open source.
- Les développeurs qui travaillent avec Git sont bien représentés dans le pool de talents disponible, et la solution fonctionne bien sur une vaste gamme de systèmes d'exploitation et d'environnements de développement intégrés (IDE).
- Par sa structure décentralisée, Git illustre parfaitement ce qu'est un système de contrôle de version décentralisé (DVCS).
- Plutôt que de consacrer un seul emplacement pour l'historique complet des versions du logiciel comme c'était souvent le cas dans les systèmes de contrôle de version ayant fait leur temps, comme CVS et Subversion (également connu sous le nom de SVN), dans Git, chaque copie de travail du code est également un dépôt qui contient l'historique complet de tous les changements.
- En plus d'être décentralisé, Git a été conçu pour répondre à trois objectifs : performances, sécurité et flexibilité.
- On va l'installer (le télécharger [ici](#)) pour pouvoir pousser nos projets dans Github

Github

- On crée d'abord un compte Github
- On génère un Token.
- Ce token vous permettra de vous connecter dans votre compte github pour uploader les projets.
- N.B: Vous devez garder ce token soigneusement car à chaque fois que vous chercherez à vous connectez dans github pour y uploader votre projet, vous devez le mettre.
- Ensuite, on peut créer le repository dans lequel on veut uploader un projet ou un dossier

Processus pour générer un Token



The screenshot shows the GitHub profile page for user 'staniher'. The profile picture is a circular image of a man with glasses and a blue shirt. Below the picture is the username 'staniher' and an 'Edit profile' button. The page header includes the GitHub logo, a search bar, and navigation links: 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main navigation bar shows 'Overview', 'Repositories' (with 8 items), 'Projects' (with 1 item), and 'Packages'. A search bar for repositories is present, along with filters for 'Type', 'Language', and 'Sort'. The repository list includes 'test' (Jupyter Notebook, updated 43 minutes ago), 'FINAL_KCA_EXAM' (Python, updated 10 days ago), and 'KCA_EXAM_COGNITIVE' (Python, updated 11 days ago). The user's avatar in the top right corner is circled in red with a '1' next to it. The dropdown menu is open, showing options like 'Signed in as staniher', 'Set status', 'Your profile', 'Your repositories', 'Your codespaces', 'Your projects', 'Your stars', 'Your gists', 'Upgrade', 'Feature preview', 'Help', 'Settings' (circled in red with a '2' next to it), and 'Sign out'.

Search or jump to... / Pull requests Issues Marketplace Explore

Overview Repositories 8 Projects 1 Packages

Find a repository... Type Language Sort

test
Jupyter Notebook Updated 43 minutes ago

FINAL_KCA_EXAM
Python Updated 10 days ago

KCA_EXAM_COGNITIVE
Python Updated 11 days ago

staniher
Edit profile

Signed in as staniher

Set status

Your profile
Your repositories
Your codespaces
Your projects
Your stars
Your gists

Upgrade
Feature preview
Help
Settings
Sign out

Processus pour générer un Token (suite)



staniher

Your personal account

[Go to your personal profile](#)

Account settings

Profile

Account

Appearance

Account security

Billing & plans

Security log

Security & analysis

Emails

Notifications

SSH and GPG keys

Repositories

Packages

Organizations

Saved replies

Applications

Developer settings

3

Public profile

Name

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

Select a verified email to display

You have set your email address to private. To toggle email privacy, go to [email settings](#) and uncheck "Keep my email address private."

Bio

Tell us a little bit about yourself

You can @mention other users and organizations to link to them.

URL

Twitter username

Company

You can @mention your company's GitHub organization to link it.

Profile picture



Edit

Processus pour générer un Token (suite)

[Settings](#) / Developer settings

4

GitHub Apps

OAuth Apps

Personal access tokens

5

Generate new token

Revoke all

Personal access tokens

Tokens you have generated that can be used to access the [GitHub API](#).

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Processus pour générer un Token (suite)

- Ici vous cochez quel niveau d'accessibilité ce token peut avoir tout en nommant votre Note:



GitHub Apps

OAuth Apps

Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS and to [authenticate to the API over Basic Authentication](#).

Note

test_uac

What's new

Expiration *

60 days

This token will expire on Thu, Oct 21 2021

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> commit_status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo_invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input type="checkbox"/> write:repo_hook	Write repository hooks
<input type="checkbox"/> read:repo_hook	Read repository hooks
<input type="checkbox"/> admin:org_hook	Full control of organization hooks
<input type="checkbox"/> gist	Create gists
<input type="checkbox"/> notifications	Access notifications
<input type="checkbox"/> user	Update ALL user data
<input type="checkbox"/> read:user	Read ALL user profile data
<input type="checkbox"/> user:email	Access user email addresses (read-only)
<input type="checkbox"/> user:follow	Follow and unfollow users
<input type="checkbox"/> delete:repo	Delete repositories
<input type="checkbox"/> write:discussion	Read and write team discussions
<input type="checkbox"/> read:discussion	Read team discussions
<input type="checkbox"/> admin:enterprise	Full control of enterprises
<input type="checkbox"/> manage_billing:enterprise	Read and write enterprise billing data
<input type="checkbox"/> read:enterprise	Read enterprise profile data
<input type="checkbox"/> admin:gpg_key	Full control of public user GPG keys (Developer Preview)
<input type="checkbox"/> write:gpg_key	Write public user GPG keys
<input type="checkbox"/> read:gpg_key	Read public user GPG keys


Generate token


Cancel

9

Votre token généré est en bas, il faut le sauvegarder quelque part et ne pas le perdre. Pour notre cas, nous avons généré un token qui va durer 60 jours. Après ces jours, le token va expirer et nous serons obligés de générer un autre, en suivant la même procédure.



 Search or jump to... / Pull requests Issues Marketplace Explore

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved. 

[Settings](#) / Developer settings

GitHub Apps

OAuth Apps

Personal access tokens


Personal access tokens

Generate new token Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

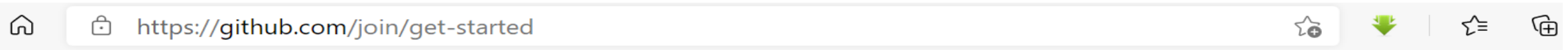
10

✓ ghp_LBrTGwbm1hXrRCWdmkMkXNvXRinmVy0hnJzr 

Delete

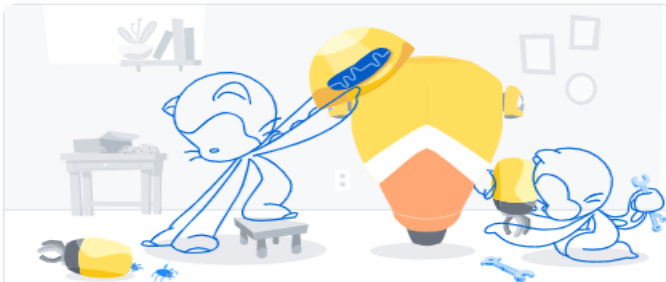
Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

On crée le repository dans lequel on va uploader le projet



What do you want to do first?

Every developer needs to configure their environment, so let's get your GitHub experience optimized for you.



Start a new project

Start a new repository or bring over an existing repository to keep contributing to it.

Create a repository



Collaborate with your team

Improve the way your team works together and get access to more features with an organization.

Create an organization

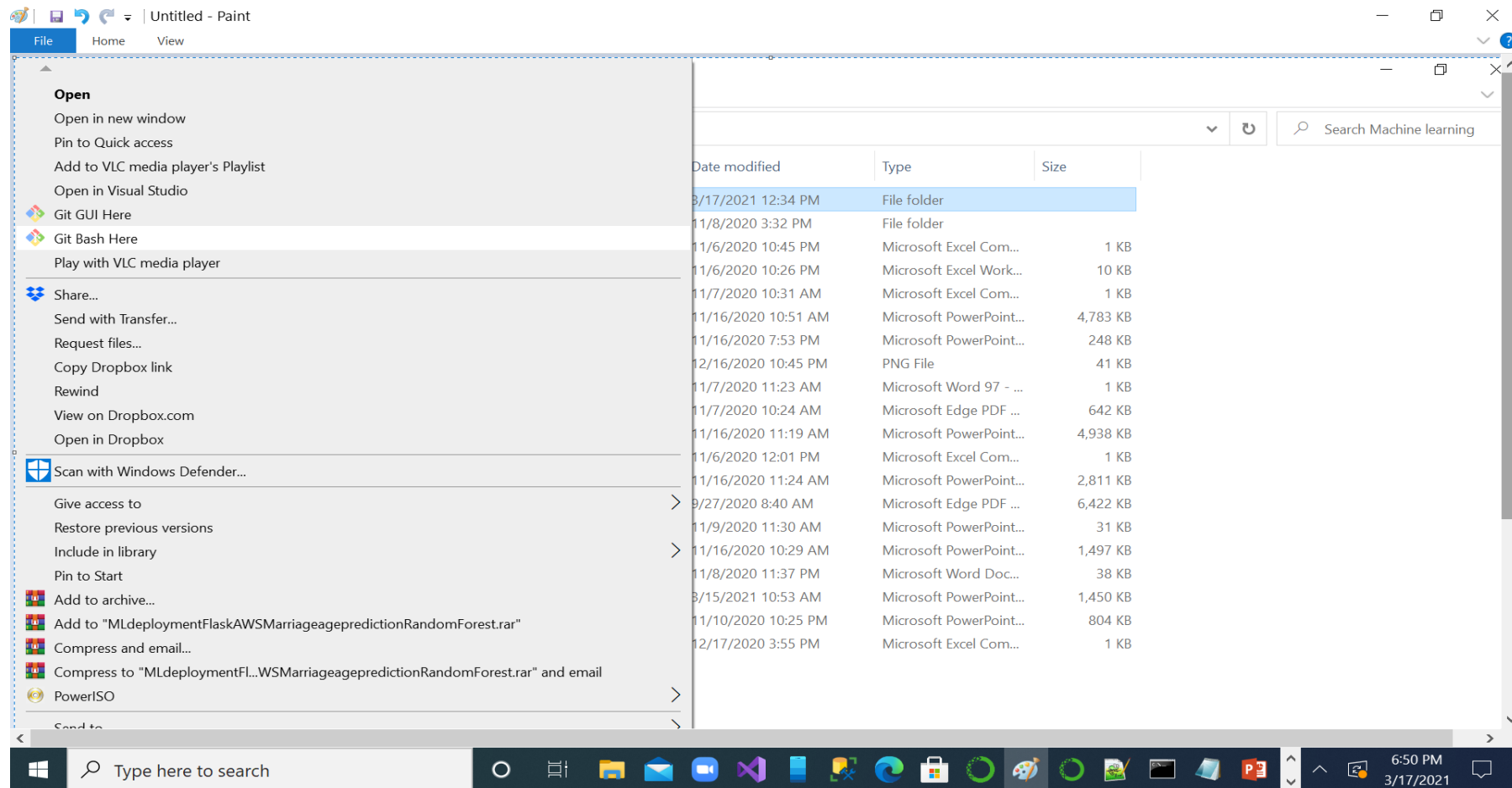


Learn how to use GitHub

Get started with an "Introduction to GitHub" course in our Learning Lab.

Start Learning

On va faire click droit sur le dossier de son projet qu'on veut amener dans Github=>Et puis on clique sur Git Bash Here



MINGW64: c:/Users/user/Dropbox/Cours/DataMining/TD_Covid

```
user@STANIER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (main)
$ git init
Initialized empty Git repository in C:/Users/user/Dropbox/Cours/DataMining/TD_Covid/.git/
```

```
user@STANIER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (master)
$ git add .
warning: LF will be replaced by CRLF in Covid_19_data_analysis_project_deploy.ipynb.
The file will have its original line endings in your working directory
```

```
user@STANIER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (master)
$ git commit -m "First commit"
[master (root commit) fe65ca9] First commit
1 file changed, 28569 insertions(+)
create mode 100644 Covid_19_data_analysis_project_deploy.ipynb
```

Dans la quatrième commande ci-dessous, on doit saisir ceci:

git remote add origin https://ton_username:ton_Token@github.com/ton_username/nom_de_ton_repository.git



```
user@STANIHER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (master)
$ git remote add origin https://staniher:ghp_L6Qx18B3D000rI0panAWe3HnxRXQ8u7ZpNz
r@github.com/staniher/test.git

user@STANIHER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.08 MiB | 62.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/staniher/test.git
 * [new branch]      master -> master

user@STANIHER MINGW64 ~/Dropbox/Cours/DataMining/TD_Covid (master)
$
```

Ton username

Ton token que tu as genere dans la partie Developer setting

Ton username

Le nom de ton repository dans lequel tu veux uploader ton projet

4

5

Travail Pratique 1: Rappel

- **Objectifs :**

- Configurer votre ordinateur (et l'équiper) avec les outils nécessaires à l'utilisation de Python et des packages disponibles.
- Comprendre la différence entre Anaconda, MiniConda, et Conda
- Comprendre comment accéder à Python via votre terminal (MacOS) ou la ligne de commande (Windows ou Linux)
- Apprendre à utiliser Git et GitHub
- Apprendre les bases de la programmation Python



Votre raisonnement est excellent,
seules vos hypothèses de base
sont erronées

— *Ashleigh Brilliant* —