

# Initiation à la programmation avec Python (v3)

## Cours n°1

Copyright (C) 2015 - 2019

Jean-Vincent Loddó

Licence Creative Commons Paternité  
Partage à l'Identique 3.0 non transposé  
(CC BY-SA 3.0)

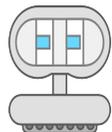
## Sommaire du cours n°1

- Notion n°1 : programmer = **automatiser un service**
- Notion n°2 : les **valeurs**
- Notion n°3 : les **variables**
- Notion n°4 : la **conditionnelle** (prise de décision)

2

## Notion n°1 : programmer = automatiser un service

- On peut imaginer un **programme** comme un **robot**
- Même si il n'a pas un corps

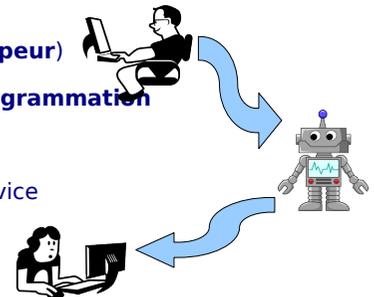


- Et même si son ectoplasme est emprisonné dans une fenêtre graphique ou textuelle (**terminal**) d'un ordinateur
- Comme un robot : il fait un travail, il questionne l'utilisateur, il réagit aux réponses et autres stimuli (clavier, souris, réseau, etc)
- Comme un robot : quelqu'un le construit, quelqu'un l'utilise

3

## Développeur → Programme → Utilisateur

- Comme un robot : quelqu'un le construit, quelqu'un l'utilise
- Qui le construit ?
  - C'est le **programmeur** (ou **développeur**)
  - Comment : avec un **langage de programmation**
  - Combien de fois : une fois !
  - Pourquoi : parce qu'**il rendra** un service
- Qui l'utilise ?
  - C'est l'**utilisateur**
  - Comment : avec une interface (graphique ou textuelle)
  - Combien de fois : autant qu'il le souhaite
  - Pourquoi : parce qu'**il rend** un service

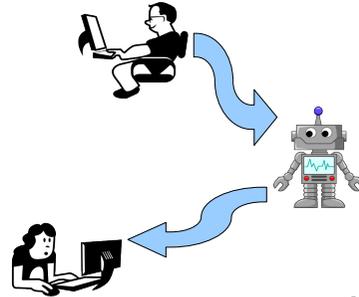


4

## Développeur → Programme → Utilisateur

- Mais alors, apprendre à programmer c'est apprendre à **automatiser un service** ?

# OUI



5

## Développeur → Programme → Utilisateur

- Est-ce que le développeur peut être aussi l'utilisateur ?
- Ce n'est pas sa vocation et ce n'est pas souvent le cas, mais c'est possible
- Sauf... lorsque le développeur **teste** (avant de le livrer) si le programme rend effectivement le service qu'on attend de lui
- Un peu de terminologie à propos :
  - si le comportement est erroné on dit que le programme a un **bug** (ou **bogue**)
  - l'activité qui consiste à corriger un programme s'appelle **debugging** (ou **debogage**)

6

## Premier programme (1)

- **Service à rendre** : afficher des citations ou proverbes qui stimulent la réflexion

```
#!/usr/bin/python3
# coding: utf-8
print("Lorsqu'on se cogne la tête contre un pot et que cela sonne creux,")
print("ce n'est pas forcément le pot qui est vide.")
print("Confucius, philosophe.")
print("---")
print("Qui veut faire quelque chose trouve un moyen,")
print("qui ne veut rien faire trouve une excuse.")
print("Proverbe Arabe.")
```

7

## Premier programme (2)

- **Service à rendre** : afficher des citations ou proverbes qui stimulent la réflexion

```
#!/usr/bin/python3
# coding: utf-8
print("Lorsqu'on se cogne la tête contre un pot et que cela sonne creux,")
print("ce n'est pas forcément le pot qui est vide.")
print("Confucius, philosophe.")
print("---")
print("Qui veut faire quelque chose trouve un moyen,")
print("qui ne veut rien faire trouve une excuse.")
print("Proverbe Arabe.")
```

*shebang* : en-tête du programme (ou "script").  
Ne pas oublier non plus de rendre le fichier exécutable (`chmod +x`)

pour utiliser des caractères accentués

**Remarque :**  
ce programme fait appel à l'outil (fonction) **print** sept fois :  
le service rendu est donc un **assemblage** de sous-services rendus par des outils (sous-programmes) pré-existants

8

## Premier programme (3)

- **Service à rendre** : afficher des citations ou proverbes qui stimulent la réflexion
- Nous avons rangé la liste des instructions dans le fichier `citations.py` et nous l'avons rendu exécutable (`chmod +x`).  
Exécution :
- ```
$ ./citations.py
```

Lorsqu'on se cogne la tête contre un pot et que cela sonne creux,  
ce n'est pas forcément le pot qui est vide.  
Confucius, philosophe.  
---  
Qui veut faire quelque chose trouve un moyen,  
qui ne veut rien faire trouve une excuse.  
Proverbe Arabe.

9

## Premier programme (4)

- **Service à rendre** : afficher des citations ou proverbes qui stimulent la réflexion
- Nous avons rangé la liste des instructions dans le fichier `citations.py` et nous l'avons rendu exécutable (`chmod +x`).  
Exécution :

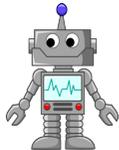
- ```
$ ./citations.py
```

Lorsqu'on se cogne la tête contre un pot et que cela sonne creux,	print n°1
ce n'est pas forcément le pot qui est vide.	print n°2
Confucius, philosophe.	print n°3
---	print n°4
Qui veut faire quelque chose trouve un moyen,	print n°5
qui ne veut rien faire trouve une excuse.	print n°6
Proverbe Arabe.	print n°7

10

## Notion n°1 : programmer = automatiser un service

- On peut imaginer un **programme** comme un **robot**



- Ok, ok, mais emprisonné dans la fenêtre d'un ordinateur il ne pourra pas faire le **ménage** ! Ni **repasser le linge** !
- Alors, que peut-il rendre comme service intéressant ?
  - récupérer des informations (p.e. sur Internet, dans des fichiers, ou données par l'utilisateur), présenter des informations (p.e. des proverbes, la date), calculer des informations
- Autrement dit : un programme **élabore des informations**

11

## Notion n°1 : programmer = automatiser un service

- Admettons : un programme **élabore des informations**



- Quels types d'information sont traitées ?
  - Textes ? Nombres ? Dates ? Noms de fichiers ? Adresses Internet ? Autre chose ?
  - C'est la notion de **valeur**
  - Et ça dépend du langage de programmation...

12

## Notion n°2, les informations ou valeurs élaborées

- Les **valeurs** d'un langage de programmation sont les informations que les programmes sont capables de manipuler
- Il y en a de plusieurs **types** :
  - Nombres entiers (0 42 -16 100)
  - Nombres flottants (3.14159 2.71828)
  - Booléens (True False)
  - Caractères (a z A Z 0 9 #)
  - Chaînes de caractères (salut HeLLo Confucius, philosophe.)
  - Tableaux, listes, tuples, arbres, dictionnaires, fonctions, objets, ...

13

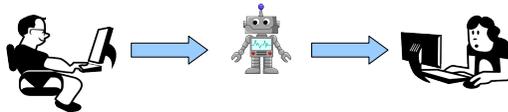
## Les valeurs en Python

- Les **valeurs** d'un langage de programmation sont les informations que les programmes sont capables de manipuler
- Il y en a de plusieurs **types** :
  - Nombres entiers (0 42 -16 100) ← type « int »
  - Nombres flottants (3.14159 2.71828) ← type « float »
  - Booléens (True False) ← type « bool », attention à la syntaxe : première lettre de True et False en majuscule
  - ~~Caractères (a z A Z 0 9 #)~~ ← ce type de base n'existe pas en Python
  - Chaînes de caractères (salut HeLLo Confucius, philosophe.) ← type « string »
  - Tableaux, listes, tuples, arbres, dictionnaires, fonctions, objets, ... ← tout y est en Python !

14

## Résumé pour l'instant et prochaine question...

- Un programme est comme un robot sauf que son job est de manipuler des informations, par exemple des chaînes de caractères
- Un programmeur le construit, un utilisateur l'utilise

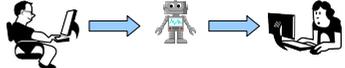


- Parce que cela rend service
- Très bien mais on entrevoit un petit problème :
  - comment programmer le comportement du robot **sans savoir ni quand**, ni dans **quelle circonstance**, ni par **qui**, ni **comment** il sera utilisé ?

15

## Programmer c'est aussi prévoir sans tout savoir

- Question posée :

Comment programmer le comportement du robot **sans savoir ni quand**, ni dans **quelle circonstance**, ni par **qui**, ni **comment** il sera utilisé ?
- Exemples :
  - Programmer un **navigateur Internet** : quelles pages seront visitées ? Combien en même temps ? À quelles dates ? Avec quel OS ? Avec quelles technologies (Html, Javascript, Java, ...) ? ...
  - Programmer un **traitement de texte** : quel texte ? Quelle langue ? Quels caractères ? Combien de pages ? Quels effets (gras, italique, souligné,...) ? ...
  - (plus simple) Programmer un robot qui **calcule le double** : de quel nombre ?
  - (plus simple) Programmer l'**affichage** d'une gentillesse du style « C'est génial mon cher Emmanuel d'avoir 42 ans » : quel sera le nom (à la place de Emmanuel) ? Quel sera l'âge (à la place de 42) ? Et si c'était Brigitte ?

## Notion n°3, les variables

- Pour traiter l'information que le programmeur connaît mais surtout celle qu'il **ne connaît pas**, les langages de programmation proposent les « variables »
- Les variables sont des boîtes qui ont un **nom** et un **contenu**



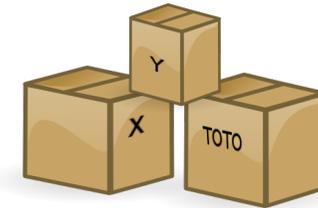
- Le contenu est une **valeur**, c'est-à-dire une information traitée par le langage de programmation
- Comment on stocke une information (valeur) dans une boîte ?
- Comment on la récupère ?

17

## Variables : comment on stocke une information dans une variable ?

- Par **affectation** du contenu :

```
TOTO = "salut le monde"
Y = 16
X = 3.14159
```



- Par lecture des caractères saisis au clavier par l'utilisateur. En C ou Java cela se fait avec **scanf**, en Bash avec **read**, en Python (v3) cela se fait avec **input** et toujours par l'affectation :

```
X = input("Votre nom ? ")
Y = input("Votre age ? ")
```

18

## Variables : comment on récupère l'information stockée dans une variable ?

- Dans certains langages le contenu de la boîte **X** est indiqué par **\$X** (Bash, PHP,...) :
- En Python, comme en math et comme dans beaucoup d'autres langages, le **nom** de la boîte peut indiquer aussi le **contenu**, ça dépend du contexte :

```
- TOTO = "La réponse"
  X = 12
  Y = 30 + X
  print(TOTO,"à la question ultime est",Y)
```

- Ce programme affiche :

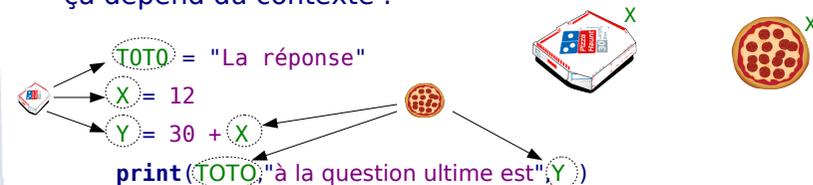
La réponse à la question ultime est 42



19

## Variables : comment on récupère l'information stockée dans une variable ?

- Dans certains langages le contenu de la boîte **X** est indiqué par **\$X** (Bash, PHP,...) :
- En Python, comme en math et dans beaucoup d'autres langages, le **nom** de la boîte peut indiquer aussi le **contenu**, ça dépend du contexte :



- Ce programme affiche :

La réponse à la question ultime est 42

20

## Variables : remarque sur les informations manipulées

- Le **développeur** programme (écrit) cette ligne en **1999** : 

```
X = input("Votre nom ? ")
```
- Un **robot** (une instance du programme) est exécuté en **2017** 
- L'**utilisateur** saisit une chaîne de caractères en **2017** 
- Le **robot** stocke cette chaîne dans la variable **X** en **2017**
- Le **développeur**, toujours en **1999**, ne connaît pas la chaîne saisie
  - mais il sait qu'elle se trouve dans **X**
  - il peut donc y accéder en écrivant **X** dans la suite du programme
- Le développeur planifie le traitement des informations qu'il **connaît** et des informations qu'il **ne connaît pas** 

21

## Le petit robot bien gentil

- Service à rendre** : le robot doit demander le nom et l'âge de l'utilisateur et lui écrire ensuite une gentillesse du style « Cher ... c'est génial d'avoir ... ans »
  - Comment le programmeur peut « remplir » les pointillés, c'est-à-dire manipuler l'information qu'il ne connaît pas ?
  - C'est simple, il peut citer cette information sans la connaître :

```
NOM = input("Votre nom ?")
AGE = input("Votre âge ?")
print("Cher", NOM, "c'est génial d'avoir", AGE, "ans")
```



- Le programmeur sait qu'une certaine information est dans une certaine boîte. Cela suffit pour faire des calculs, pour afficher ou pour prendre des décisions !

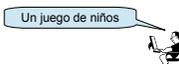
22

## Un robot pour tout âge

- Supposons à présent de vouloir écrire une phrase différente en fonction de l'âge. **Service à rendre** :
  - Si l'utilisateur a moins de 13 ans le robot devra écrire « Alors vous jouez à Call of Duty »
  - Sinon il écrira « Alors vous jouez à Pokemon »

### Comment faire ?

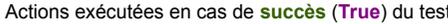
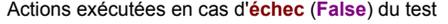
- Avec la célèbre construction **if-then-else** ! (en Python **if-elif-else**) :
- ```
AGE = input("Votre âge ? ") # AGE a type str
AGE = int(AGE) # AGE a type int
if (AGE <= 13):
    print("Alors vous jouez à Call of Duty")
else:
    print("Alors vous jouez à Pokemon")
```



23

## Notion n°4, la « conditionnelle »

- Il est possible de planifier des actions **conditionnelles** en utilisant un test, qui est une **expression** dont le résultat est **booléen** (vrai ou faux) :
  - Si le test a **succès** (résultat **True**), le robot exécutera certaines actions
  - Si le test **échoue** (résultat **False**), le robot exécutera d'autres actions

- if** **EXPR**: 
  -  **ACTIONS<sub>1</sub>** 
  - else**:
  -  **ACTIONS<sub>2</sub>** 
- Tabulation ou espaces (blancs) de la même longueur  
Pour délimiter les sous-blocs d'instructions (**spécificité Python** !)

24

## Un robot pour tout âge

- Syntaxe

```
if EXPR:  
    ACTIONS1  
else:  
    ACTIONS2
```

Remarque :  
on compose les deux fonctions  
`input` et `int` pour avoir directement  
un entier dans `AGE`

- Par rapport à notre exemple

```
AGE = int(input("Votre âge ?"))  
if (AGE <= 13):  
    print("Alors vous jouez à Call of Duty")  
else:  
    print("Alors vous jouez à Pokemon")
```

Diagramme illustrant la correspondance entre la syntaxe générale et l'exemple :

- EXPR : `(AGE <= 13)`
- ACTIONS<sub>1</sub> : `print("Alors vous jouez à Call of Duty")`
- ACTIONS<sub>2</sub> : `print("Alors vous jouez à Pokemon")`

25

## L'indentation en Python

- En informatique** : l'indentation consiste en l'ajout de tabulations ou d'espaces (blancs) dans un fichier, pour une meilleure lecture et compréhension du code (def. Wikipedia)

```
if (AGE <= 13):  
    print("Alors vous jouez à Call of Duty")  
else:  
    print("Alors vous jouez à Pokemon")
```

- En Python** : l'indentation fait partie de la **syntaxe** du langage pour délimiter les **blocs** de code. Par exemple, pour délimiter le bloc « then » (alors) et le bloc « else » (sinon) :

```
if EXPR:  
    ACTIONS1  
else:  
    ACTIONS2
```

26

## Un robot pour tout âge Le mot de la fin (du 1<sup>er</sup> cours)

```
AGE = int(input("Votre âge ?"))  
if (AGE <= 13):  
    print("Alors vous jouez à Call of Duty")  
else:  
    print("Alors vous jouez à Pokemon")
```

Non seulement on peut stocker (dans une variable) une information **qu'on ne connaît pas**, mais on peut aussi **traiter** cette information en adaptant le comportement du robot aux différentes possibilités par la construction **conditionnelle**



27

## Adresse des images utilisées

- Boite fermée <https://openclipart.org/detail/15872/closed-box-by-mcol>
- Robot sympa <https://openclipart.org/detail/170101/cartoon-robot-by-sirrobo1>
- Robot chenille <https://openclipart.org/detail/168755/cartoon-robot-by-qubodup>
- Laptop <https://openclipart.org/detail/24817/-by--24817>
- Développeur [https://openclipart.org/detail/37129/personnage\\_ordinateur-by-antoine](https://openclipart.org/detail/37129/personnage_ordinateur-by-antoine)
- Utilisateur [https://openclipart.org/detail/37135/personnage\\_ordinateur-by-antoine-37135](https://openclipart.org/detail/37135/personnage_ordinateur-by-antoine-37135)
- Pizza box <https://openclipart.org/detail/171767/pizza-haunt-by-jakororiginal-171767>
- Pizza <https://openclipart.org/detail/189439/pepperoni-pizza-by-toons4biz-189439>

28