Introduction à la pensée algorithmique avec Python (1)

📌 Ecole du Louvre, Master Documentation et Huménités Numériques, 2020

Alix Chagué

📫 alix.chague@inria.fr (mailto:alix.chague@inria.fr)

💼 *Ingénieure Recherche et Développement* @ Inria

Crédits

> Librement inspiré des supports de cours (https://github.com/gguibon/essec-python-1) de Gaël Guibon
> Librement inspiré des supports de cours (https://goo.gl/UFqu2U) de Julien Pilla
>
> Retrouvez l'ensemble du cours sur 👉 **github.com/alix-tz/enc-intro-algo** (https://github.com/alix-tz/enc-intro-algo)👈

Syllabus

Plan du cours

- Syllabus
- Pogrammation et algorithmique
- Présentation de Python
- Valeurs et variables
- Opérations arithmétiques
- Exercices logiques
- Exercices pratiques

Objectifs du cours

🏆 Connaître le vocabulaire de la programmation

🏆 Savoir lire et concevoir un algorithme

🏆 Se familiariser avec l'environnement de développement de Python

🏆 Connaître les bonnes pratiques de développement

🏆 Ecrire un programme simple avec Python

Liens utiles

👉 Simulateur d'environnement Python en mode pseudo-IDE : https://repl.it/languages/python3 (https://repl.it/languages/python3)

👉 Simulateur d'environnement Python en mode console : https://www.python.org/shell/ (https://www.python.org/shell/)

👉 Documation officielle de Python : https://docs.python.org/3/ (https://docs.python.org/3/)

👉 Visualisateur d'execution de code Python : http://pythontutor.com/ (http://pythontutor.com/)

Quelques ressources pour continuer à se former

👉 "Automate the Boring Stuff with Python" (`en`) : https://automatetheboringstuff.com/ (https://automatetheboringstuff.com/)

👉 Leçons dédiées à Python sur *Programming Historian* (`en` , `fr` ou `es`) : https://programminghistorian.org/en/lessons/introduction-and-installation (https://programminghistorian.org/en/lessons/introduction-and-installation)

👉 "Apprendre à coder avec Python", MOOC de l'Université Libre de Bruxelles (`fr`) : https://www.fun-mooc.fr/courses/course-v1:ulb+44013+session04/about (https://www.fun-mooc.fr/courses/course-v1:ulb+44013+session04/about)

👉 "Apprenez à programmer en Python", cours en ligne sur OpenClassroom (`fr`) : https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python (https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python)

👉 "Introduction à Python" pour le Master Ingénierie Multilingue de l'Inalco, Loïc Grobol et Yoann Dupont (`fr`) : https://loicgrobol.github.io/python-im/m2-2018/ (https://loicgrobol.github.io/python-im/m2-2018/)

Programmation et algorithmique

A quoi ça sert de programmer ?

- Automatiser des actions simples et répétitives
- Réduire le temps passé sur ces tâches
- Réduire les erreurs humaines (dues à la fatigue ou à l'ennui par exemple) et faciliter leur détection

Exemple :

- Comment faites-vous pour renommer 390 fichiers (*ex: retirer "_copie" ou les renommer en ajoutant un numéro d'ordre*) ? Combien de temps cela prend-il ? Quelles chances avez-vous de commettre des erreurs en faisant cela ?
- Vous avez une base de données pleine de liens URL vers des ressources et vous voulez vérifier que ces liens sont corrects (toujours actifs et liés à la bonne ressource). Comment faites-vous ? Combien de temps cela va prendre ?

> Dans un univers professionnel de plus en plus "numérique", avoir des bases en programmation est une compétence utile et valrisée.
>
> Pour pouvoir mener à bien des projets numériques d'ampleur, il est nécessaire de démistifier le travail des développeur·ses avec qui vous collaborerez et de pouvoir comprendre les contraintes de la programmation.

Notions élémentaires

Par `algorithmique` (ou *algorithmie*), on désigne généralement les paradigmes qui guident la logique programmatique et qui existent quel que soit le langage de programmation utilisé. Construire un algorithme, c'est donc **imaginer un scénario** permettant d'obtenir un résultat étant donné un ensemble de contraintes formelles.

Un `programme` est la mise en application d'un algorithme. C'est une **suite d'instructions** fournies à la machine pour qu'elle les exécute afin de produire un résultat (`output`). Un programme peut en plus faire appel à des données extérieures (`input`).

Ces instructions sont exécutées dans un ordre précis, un algorithme a donc un début et une fin.

Elles sont construites à partir d'éléments de bases communs à tous les langages :

- `valeurs`
- `fonctions`
- `variables`
- `opérateurs`
- `structures conditionnelles` (si... alors)
- `boucles` ou `structures répététives`
- `commentaires`

Langages de programmation

Il existe une multitude de langages pour exprimer ces instructions, par exemple :

C++	Java	Python	R	php
				

🤷 **langages de haut niveau :** Python, Java, etc.

🤖 **langages de bas niveau :** C, langage d'assemblage, code machine, etc.

Les langages de haut niveau ont besoin d'être `compilés` avant d'être exécutés : ils sont traduits en un code binaire (suite de 0 et de 1) compréhensible par la machine.

Starter pack

Pas besoin d'être un **robot** 🤖, un-e **magicien-ne** 🧙 ou un-e **geek** 🤓 pour faire de l'algo !

Il faut simplement...

- de la **logique**
- de la **rigueur :**
  - vérifier tout, chaque caractère, chaque valeur, chaque ponctuation
  - mettre les instructions dans le bon ordre
- de la **patience :**
  - vérifier/comprendre les messages d'erreur
  - procéder pas à pas

Logique

Pseudo code

Le pseudo code est une façon d'exprimer une suite d'instructions de manière abstraite, sans suivre la grammaire d'un langage en particulier.

En pseudo code, `1 ligne = 1 instruction` !

```
DEBUT

Instruction 1
Instruction 2
Instruction 3
etc...

FIN
```

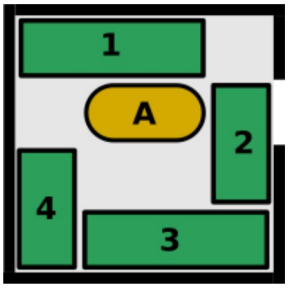
Instructions

Quelques exemples d'instructions :

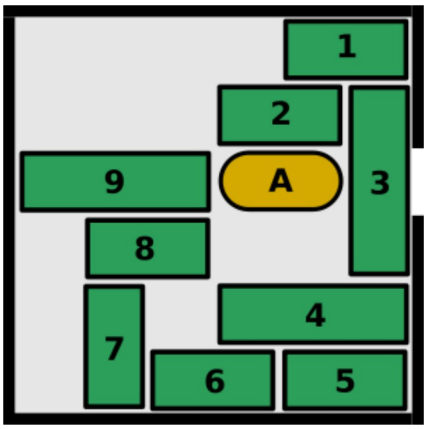
```
// commentaire

variable = "valeur" // assignation

faisQuelqueChose()  // appel de fonction

1 + 1               // opération arithmétique

1 == 1              // comparaison de valeurs

// etc...
```

Problèmes

1. Comment intervertir le contenu de deux verres ?
2. Comment vérifier la validité d'une série de liens URL ?
3. Comment renommer une série de fichiers ?
4.



Pseudo-code : exercice de logique



Instructions possibles

- sélectionner un bloc
- déplacer dans un direction

Python

👉 https://repl.it/languages/python3 (https://repl.it/languages/python3)

Python est un langage de programmation de haut niveau sous licence libre. Il a été créé par Guido Van Rossum mais son développement est pris en charge depuis 2001 par la Python Software Foundation.

Le langague Python continue d'évoluer depuis par le biais des `PEP` (*Python Enhancement Proposals*)

Python 2.x vs. 3.x

Entre 2008 et 2019, cohabitaient `Python 2` et `Python 3`, deux versions de Python dont la compatibilité n'est pas totale. Depuis le 31 décembre 2019, Python 2 est officiellement "abandonné" et seul regne Python 3.

Premier Script

👉 A tester sur https://www.python.org/shell/ (https://www.python.org/shell/)

```
print("hello, world")
```

👉 `print()` est une fonction dite `built-in` et qui affiche des valeurs qui lui sont fournies

👉 `"hello, world"` est une `chaîne de caractères` fournie à la fonction `print()`

👉 une fonction est un bloc d'instructions permettant d'obtenir un résultat précis

👉 anatomie d'une fonction : `nom_de_la_fonction(argument, argument, etc)`

Les instructions sont executées dans l'ordre où elles sont écrites

```
In [2]: print('hey') # première instruction
        print('you') # deuxième instruction
        print('!') # troisième instruction
```

```
hey
you
!
```

Gestion des erreurs

Un programme **plante** s'il contient des erreurs. Il affiche alors un `message d'erreur` indiquant la raison pour laquelle il s'est interrompu.

Ce n'est **pas grave** de faire des erreurs. L'important c'est de savoir lire le message d'erreur pour trouver l'origine du problème et le résoudre.

Savoir lire un message d'erreur

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

```
File "<stdin>", line 1
  print("hello world)
                    ^
SyntaxError: EOL while scanning string literal
```

```
Traceback (most recent call last):
File "{nom du script}", line {numéro de ligne}, in <module>:`
  print("hello world) {<- citation de la ligne de code fautive}
                   ^ {<- pointeur}
{type d'erreur}: {description}
```

- `numéro de ligne` : numéro de la ligne fautive dans le script
- `pointeur` : indique non pas l'emplacement de l'erreur mais l'emplacement à partir duquel l'erreur fait planter le programme. Il faut parfois remonter quelques caractères ou quelques lignes plus haut pour trouver l'origine de l'erreur.
- `{type d'erreur}` : NameError, SyntaxError, TypeError, ValueError, etc...
- `description` : parfois suivi d'une suggestion de correction. Notez que dans certains cas une erreur dans le code conduit le programme a mal interpréter les instructions et donne une description de l'erreur inexacte.

Messages d'erreurs fréquents

Erreur n° 1 "NameError"

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

> On a fait référence à une variable qui n'existe pas.
>
> - soit parce qu'on a oublié les " autour d'une chaine de caractères
> - soit parce qu'on a fait une faute en tapant le nom de la variable
> - soit parce qu'on fait référence à une variable trop tôt dans le script (avant qu'elle ne soit créée)

Erreur n° 2 "SyntaxError"

```
print "hello world"
  File "<stdin>", line 1
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

> Le code ne respecte pas la syntaxe de Python
>
> - "EOL while scanning string literal" : on a oublié de fermé une chaîne de caractère (EOL : end of line)
> - "Missing parentheses in ..." : on a mal refermé un bloc entre parenthèse
> - "can't assign to literal" : on essaie d'assigner une valeur à un nombre
> - "invalid syntax" : message générique

Accéder à la documentation

En programmation, lire la documentation est essentiel.

- soit la documentation officielle de Python : https://docs.python.org/fr/3/ (https://docs.python.org/fr/3/)
- soit en affichant un extrait de documentation grâce à la fonction `help()`

```
In [1]: help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Valeurs et variables

Il existe plusieurs types de valeurs possibles, les principales sont :

- `string`, `str` (chaîne de caractères) : du texte écrit 'entre guillemets simples', "doubles" ou """triples""". *Ex : "21"*
- `integer`, `int` (entiers) : des nombres entiers. *Ex : 21*
- `float`, `fl` (décimaux) : des nombre décimaux, Attention les décimaux sont après un point `.` et non une virgule (`,`). *Ex : 21.0*
- `boolean`, `bool` (Booléen) : soit True, soit False

> Attention !
>
> - "3" et 3 ne sont pas identiques !
> - 4 et 4.0 sont identiques
> - alors qu'on aura : 1, 14, 19, 40, 150
> - on aura : "1", "14", "150", "19", "40"

Les variables sont des conteneurs permettant de sauvegarder temporairement des valeurs.

👉 une variable porte un `nom`, contient une `valeur` et possède un `type`

👉 la valeur et le type d'une variable peuvent changer mais pas son nom.

Assigner une valeur à une variable

```
nom_de_variable = "valeur de la variable"
```

In [6]:
```
print("hello, world 1")  # affichage sans variable

var = "hello, world 2"  # création de la variable "var" contenant une chaine de caractères

print(var)  # affichage de la valeur contenu dans la variable "var"
```

```
hello, world 1
hello, world 2
```

In [5]:
```
varA = "hello"  # première assignation de valeur
varA = "world"  # réassignation d'une nouvelle valeur

print(varA)
```

```
world
```

Nommer une variable

👉 le nom de variable doit respecter les règles suivantes :

- être une suite de caractères alphanumériques ou "_" (underscore) en majuscule ou minuscule
- ne pas commencer par un chiffre
- ne pas contenir de "-" (tiret) ou d'espace
- ne pas être un `mot réservé` (35 mots-clefs + noms de fonctions built-in)
- éviter les caractères non-ASCII (caractès accentués, emoji, etc)

In [1]:
```
#Pour savoir quels sont les "mots réservés"

import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def
', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', '
lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

In [2]:
```
# Pourquoi il ne faut pas utiliser le nom d'une fonction built-in pour créer une variable :
# On définit une variable nommée "help" contenant l'entier 666
help = 666
# résultat : le mot "help" renvoie désormais à la variable et non plus à la fonction help()
help(print)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-11b52f5eadb2> in <module>
      3 help = 666
      4 # résultat : le mot "help" renvoie désormais à la variable et non plus à la fonction help()
----> 5 help(print)

TypeError: 'int' object is not callable
```

In [3]:
```
# On ne peut pas commencer le nom d'une variable par un chiffre
1_does_not_work = 'ça ne marche pas'
```

```
  File "<ipython-input-3-6497af45ade4>", line 2
    1_does_not_work = 'ça ne marche pas'
     ^
SyntaxError: invalid decimal literal
```

In [3]:
```
# Le nom d'une variable ne peut pas contenir de tiret (-)
ma-variable = "oups"
```

```
  File "<ipython-input-3-47797ba742ac>", line 2
    ma-variable = "oups"
    ^
SyntaxError: cannot assign to operator
```

In [4]:
```
# ne pas contenir d'espace
ma variable = "Toujours pas"
```

```
  File "<ipython-input-4-53cd3fcbc277>", line 2
    ma variable = "Toujours pas"
       ^
SyntaxError: invalid syntax
```

In [8]:
```
lambda = "ceci est un mot réservé"
```

```
  File "<ipython-input-8-40194925f324>", line 1
    lambda = "ceci est un mot réservé"
           ^
SyntaxError: invalid syntax
```

Convention et bonnes pratiques de nommage

Il existe deux manières conventionnelles de former des noms de variables contenant plusieurs mots :

In [10]:
```
# le camelCase
maVariable = "Une bosse ou deux bosses ?"

# le snake_case
ma_variable = "DJ Snake"
```

Il est recommandé d'utiliser :

- des noms de variables clairs et explicites (`mon_document` plutôt que `var1`)
- des noms de variables non ambigus (`varl` et `var1` se ressemblent)

👉 https://www.python.org/dev/peps/pep-0008/#naming-conventions (https://www.python.org/dev/peps/pep-0008/#naming-conventions)

Typage dynamique

Le type d'une variable fait référence au type d'objet / de valeur qu'elle contient. En Python, le `typage est dynamique` , contrairement à d'autres programmes, comme C ou Javascript, où on doit annoncer le type d'une variable au moment de sa création.

```
// en c
int nombre_d_eleves = 20

# en python
nombdre_d_eleves = 20
```

En Python, si on change la valeur d'une variable par une valeur d'un type différent, le type de la variable change automatiquement.

```
mon_chiffre = 7  # variable de type "int"
mon_chiffre = "sept"  # variable de type "str"
```

Pour connaître le type d'une variable, on utilise la fonction built-in `type()`

In [11]:
```
mon_chiffre = 7
print(mon_chiffre, " = ", type(mon_chiffre))

mon_chiffre = "sept"
print(mon_chiffre, " = ", type(mon_chiffre))
```

```
7  =  <class 'int'>
sept  =  <class 'str'>
```

Opérations arithmétiques

Les opérations arithmétiques sont l'une des premières manipulations qu'il est possible de faire avec des valeurs et des variables.

On peut additionner (`+`) des valeurs, les soustraire (`-`), les multiplier (`*`), les diviser selon plusieurs méthodes (`/` , `//`) ou récupérer le reste d'une division entière (`%`), etc.

```
a = 4
b = 2 + a
c = b * a
```

Addition : +

In [27]:
```
addition_int = 1 + 1  # addition de 2 entiers
addition_float = 1.0 + 2  # addition d'un décimal et d'un entier
addition_str = "a" + "b"  # addition de deux chaines de caractères

print(addition_int, addition_float, addition_str, sep="\n")
```

```
2
3.0
ab
```

Soustraction : -

In [30]:
```
substraction_int = 2 - 4  # soustraction de 2 entiers
substraction_float = 12.6 - 8.4  # soustraction d'un décimal et d'un entier

print(substraction_int, substraction_float, sep="\n")
```

```
-2
4.199999999999999
```

In [31]:
```
substraction_str = "hello" - "world"  # on ne peut pas soustraire deux chaines de caractères
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-d1c41d5bcf59> in <module>
----> 1 substraction_str = "hello" - "world"  # on ne peut pas soustraire deux chaines de caractères

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Multiplication : *

In [37]:
```
multiplication = 2 * 10.0  # multiplication d'un entier et un décimal

multiplication_str = "multi" * 3  # multiplication d'une chaine de caractère et d'un entier

print(multiplication, multiplication_str, sep="\n")
```

```
20.0
multimultimulti
```

Division et reste : `/`, `%` et `//`

In [16]:
```
division = 36 / 7  # la division de 2 entiers produit toujours un décimal
division_tronquée = 36 // 7  # une division tronquée produit toujours un entier
reste_decimal = 36 % 7.0
reste_entier = 36 % 7

print(division, division_tronquée, reste_decimal, reste_entier, sep="\n")
```

```
5.142857142857143
5
1.0
1
```

In [20]:
```
division_string = "blabla" // 2  # on ne peut pas diviser une chaine de caractères
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-0e24fe7ac070> in <module>
----> 1 division_string = "blabla" // 2

TypeError: unsupported operand type(s) for //: 'str' and 'int'
```

Exposant : `**`

In [21]:
```
10 ** 4  # 10^4
```

Out[21]:
```
10000
```

Priorités

Comme en mathématiques, les parenthèses `()` permettent de définir les priorités :

In [22]:
```
sans_parenthese = 36 / 2 * 3
avec_parenthese = 36 / (2 * 3)
print(sans_parenthese, avec_parenthese, sep="\n")
```

```
54.0
6.0
```

(In)compatibilités

On ne peut pas `concaténer` (additionner) certains types :

In [25]:
```
"hello" + 42
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-3baee1551631> in <module>
----> 1 "hello" + 42

TypeError: can only concatenate str (not "int") to str
```

Tandis que d'autres types de données sont plus souples :

In [28]:
```
15 + 3.21
```

Out[28]:
```
18.21
```

Convertir des objets

Certaines fonctions permettent de changer le type d'un objet, à condition que la valeur de cet objet puisse être transférée vers un nouveau type. Ces fonctions sont : `str()`, `int()`, `float()`, `bool()`. Il en existe d'autres pour d'autres types de valeurs que nous verrons plus tard.

- `str(21)` donnera `"21"`
- `int(41.69)` donnera `41`
- `bool(1)` donnera `True`
- `float("4")` donnera `4.0`
- par contre on ne peut pas faire `int("ma chaine de caractères")`

In [1]:
```
var_int = 4
var_float = 18.21
var_bool = False
var_str = "soixante"
```

str()

In [59]:
```
# str() : transformer en chaîne de caractères
print(type(var_int), type(str(var_int)), sep = " ==> ")
print(type(var_bool), type(str(var_bool)), sep = " ==> ")
print(type(56.5), type(str(56.5)), sep = " ==> ")
```

```
<class 'int'> ==> <class 'str'>
<class 'bool'> ==> <class 'str'>
<class 'float'> ==> <class 'str'>
```

int()

In [60]:
```
# int() : transformer en entier
print(type(var_float), type(int(var_float)), sep = " ==> ")
print(type(56.5), type(int(56.5)), sep = " ==> ")
print(type("42"), type(int("42")), sep = " ==> ")
```

```
<class 'float'> ==> <class 'int'>
<class 'float'> ==> <class 'int'>
<class 'str'> ==> <class 'int'>
```

In [61]:
```
# spécificité quand on transforme un booléen en entier
print(type(var_bool), type(int(var_bool)), sep = " ==> ")
print(var_bool, int(var_bool), sep = " ==> ")
```

```
<class 'bool'> ==> <class 'int'>
False ==> 0
```

In [63]:
```
# toutes les chaines de caractères ne sont pas convertibles en entiers
print(type(var_str), type(int(var_str)), sep = " ==> ")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-63-3a384075590f> in <module>
      1 # toutes les chaines de caractères ne sont pas convertibles en entiers
----> 2 print(type(var_str), type(int(var_str)), sep = " ==> ")

ValueError: invalid literal for int() with base 10: 'soixante'
```

float()

In [71]:
```
# float() : transformer en décimal
print(type(var_int), type(float(var_int)), sep=" ==> ")
print(type("42"), type(float("42")), sep = " ==> ")
```

```
<class 'int'> ==> <class 'float'>
<class 'str'> ==> <class 'float'>
```

In [72]:
```
# spécificité quand on transforme un booléen en décimal
print(type(var_bool), type(float(var_bool)), sep = " ==> ")
print(var_bool, float(var_bool), sep = " ==> ")
```

```
<class 'bool'> ==> <class 'float'>
False ==> 0.0
```

In [73]:
```
print(type(var_str), type(float(var_str)), sep = " ==> ")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-73-340baff32a83> in <module>
----> 1 print(type(var_str), type(float(var_str)), sep = " ==> ")

ValueError: could not convert string to float: 'soixante'
```

bool()

In [82]:
```
# bool() : transformer en booléen
print(type(var_float), type(bool(var_float)), sep=" ==> ")

print(type(var_int), type(bool(var_int)), sep=" ==> ")

print(type(var_str), type(bool(var_str)), sep = " ==> ")
```

```
<class 'float'> ==> <class 'bool'>
<class 'int'> ==> <class 'bool'>
<class 'str'> ==> <class 'bool'>
```

In [85]:
```
# Mais quel booléen ?
print(type(var_float), type(bool(var_float)), sep=" ==> ")
print(var_float, bool(var_float), sep=" ==> ")

print(type(var_int), type(bool(var_int)), sep=" ==> ")
print(var_int, bool(var_int), sep=" ==> ")

print(type(var_str), type(bool(var_str)), sep = " ==> ")
print(var_str, bool(var_str), sep = " ==> ")
```

```
<class 'float'> ==> <class 'bool'>
18.21 ==> True
<class 'int'> ==> <class 'bool'>
4 ==> True
<class 'str'> ==> <class 'bool'>
soixante ==> True
```

In [84]:
```
# A quel moments obtient-on "False"
print(type("False"), type(bool("False")), sep = " ==> ")
print("False", bool("False"), sep = " ==> ")

print(type(""), type(bool("")), sep = " ==> ")
print("", bool(""), sep = " ==> ")

print(type(0), type(bool(0)), sep = " ==> ")
print(0, bool(0), sep = " ==> ")

print(type(None), type(bool(None)), sep = " ==> ")
print(None, bool(None), sep = " ==> ")
```

```
<class 'str'> ==> <class 'bool'>
False ==> True
<class 'str'> ==> <class 'bool'>
 ==> False
<class 'int'> ==> <class 'bool'>
0 ==> False
<class 'NoneType'> ==> <class 'bool'>
None ==> False
```

Exercices logiques

Exercice 1

En utilisant des opérateurs arithmétiques et une variable, écrivez un algorithme permettant d'obtenir l'affichage suivant :

```
3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
```

Exercice 2

Ecrivez un script Python appliquant l'algorithme de l'exercice 1.

Exercice 3

A. Ecrire un algorithme pour un programme permettant de créer automatiquement un pseudo à partir du nom, du prénom et de la longueur de ces deux éléments.

Vous pouvez également varier en utilisant la date de naissance de l'utilisateur-rice.

B. Rédiger ce programme en Python, étant donné que :

- `input()` permet de demander à l'utilisateur-rice d'entrée un valeur
- `len()` permet de connaître la taille d'une valeur

Exercices pratiques

Exercice 1

Réaliser les 6 exercices du fichier `variables.py`

👉 **https://repl.it/@AlixChagu/ENCintroalgo#basics/variables.py (https://repl.it/@AlixChagu/ENCintroalgo#basics/variables.py)**