Cours 1

Christophe Saint-Jean

September 14, 2021

1 Introduction à la programmation

Christophe Saint-Jean

Transparents / Code du cours 2021-2022

1.0.1 Mentions légales

Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International.

2 Cours 1

- Administration du cours
- Généralités sur les langages de programmation
- Généralités sur le langage Python
- Variables et standards
- Sauts conditionnnels

2.1 Administration du cours

2.1.1 L'équipe enseignante

- Bernard Besserer (TP)
- Patrick Franco (TP)
- Mohamed Haddache (TP)
- Laurent Mascarilla (TP)
- **Christophe Saint-Jean** (Cours/TP - Resp.)
- El Hadi Zahzah (TP)

2.1.2 Communication

- Questions pédagogiques : Moodle
  - Appronfondissement/Questions (Forum)
  - Organisation de l'UE/Planning (Messages privés)
- Questions administratives (Secrétariat)
  - Appartenance groupes TD/TP
  - Absences/Justifications

2.1.3 Dispositif horaire

- 5 cours de 1,5 heures (Amphithéâtre)
- 12 TPs de 1,5 heures (Salles de TP)

2.1.4 Evaluation

$$S_1 = \frac{CC_1 + CC_2}{2}$$

$$S_2 = CC_3$$

Les CC se passent en TP (5/6 et 12) sur machine:

- Une partie QCM
- Une partie évaluée par un enseignant
- Attention à la règle sur les absences

2.1.5 Les objectifs de cet enseignement

- Découvrir les bases de la programmation informatique:
  - Savoir ce qu'est un langage de programmation
  - Connaitre les éléments et le vocabulaire de bases
- Développer l'esprit logique par la pratique de la programmation.
- Appprendre un des langages support de votre formation.
- Libérer votre créativité !

2.1.6 Retour sur le questionnnaire de rentrée

2.2 Généralités sur la programmation

2.2.1 Langage humain

- Un certain *vocabulaire*, *une orthographe*, des *règles de grammaire* communes

```
le soir, les chats se promènent sur le mur
```

- Grande expressivité et diversité
- Même si l'on commet des erreurs, nous sommes capables de comprendre "globalement" le message

```
les soir chats, mur la promème sur le
```

2.2.2 Langage informatique

- La machine traite des informations binaires: 100110010101000110 . . . (même si images, sons, *programmes*, . . .)
- Le vocabulaire d'instructions machine est très réduit (arith., logique, mémoire, . . .)
- Pas de tolérance aux erreurs d'instructions
- Parler à une machine (et donc programmer), c'est s'adresser une **entité efficace mais peu compréhensive** . . .

2.2.3 Niveau d'un langage de programmation

Différents niveaux d'abstraction par rapport aux instructions du processeur:

- Langage machine: 100110010101000110 . . . (Quasi-impossible)
- Langages de bas niveau (Ex. Assembleur)
  - Lisible par un humain
  - Equivalent au langage machine (1 <-> 1)

```
MOV r0,r1       (r0 = r1)
ADD r0,r1,r2    (r0 = r1 + r2)
```

  - Spécifique à chaque processeur et son architecture
  - Haute valeur ajoutée
- Langages de bas/haut niveau (Ex. C)
  - bas niveau (proche du langage d'assemblage) Ex. possibilité d'accéder à la mémoire

```
char *a = (char*) 0x1000;
*a = 20;
```

    Langage de prédilection pour le dev. de pilote/système
- Langages de haut niveau (Ex. **Python**, Java, C++, R, . . .)
  - Langage lisible et naturel
  - Indépendent du matériel sur lequel s'exécute le programme
  - Gestion simple de la mémoire (alloc./désalloc.)
  - Variables, structures de contrôle, fil d'exécution, . . .

2.2.4 Programme Source (ou Code)

Un programme source est un ***texte*** qui:

- Dépend d'un *langage de programmation*
- Utilise un certain nombre de *conventions* (nommage, opérations, . . .),
- Obéit aux règles de *syntaxe*, de *grammaire* d'un langage.
- En géneral, sauvegardé puis exécuté depuis un fichier.

Le mode d'exécution du programme est variable (compilation, interprétation, hybride)

Exemples : Hello World

c (CM1_helloworld.c):

```c
#include <stdio.h>
int main() {
    printf("Hello World\n");
}
```

Java (HelloWorld.java):

```java
class HelloWorld {
  static public void main( String args[] ) {
    System.out.println( "Hello World" );
  }
}
```

Python 3 (CM1_helloworld.py):

```python
print("Hello World")
```

Pour aller plus loin Hello World Collection !!

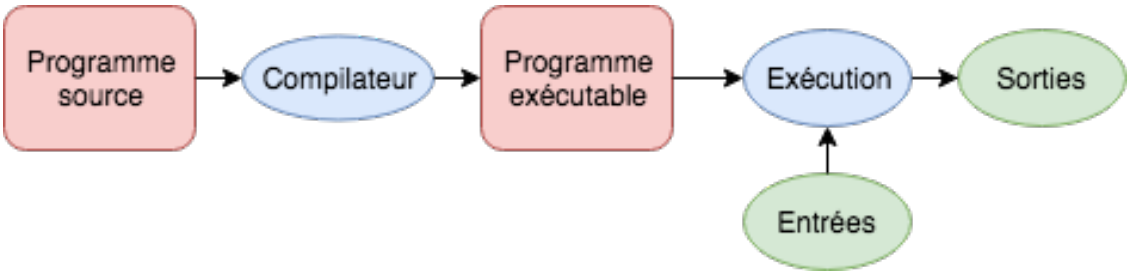
2.2.5 Les différentes sources d'erreur

- Erreurs de syntaxe: Non respect des conventions du langage
  - Oubli de parenthèses
  - Marqueurs de blocs d'instruction (begin..end, *tabulations*, accolades)
  - Nommage des variables
- Erreurs d'exécution (Runtime-Error): Opération non valide lors de l'exécution
  - Types incompatibles pour l'opération demandée (compilé vs interprété)
  - Problèmes d'accès mémoire
- Erreurs logique: Résultat différent de celui désiré

En tant que programmeurs débutants, familiarisez vous avec chaque message d'erreur !

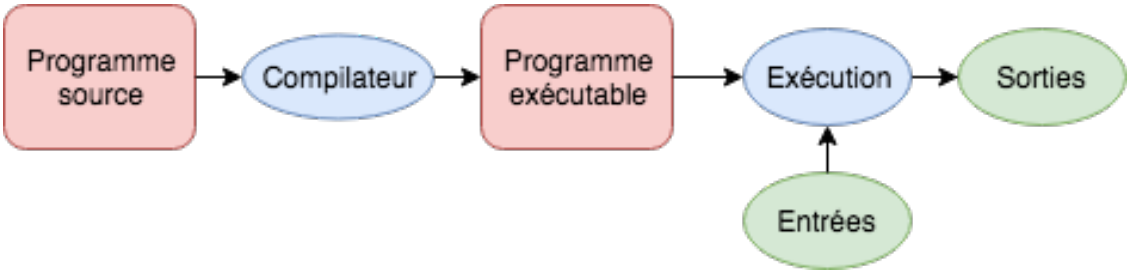
2.2.6 Exécution d'un programme

- La machine n'éxecute que du code machine
- Nécessité de traduire le langage de programmation en langage machine:
  - Compilation
  - Interprétation
  - Hybride



Compilation Compilation et exécution pour le compilateur gcc:

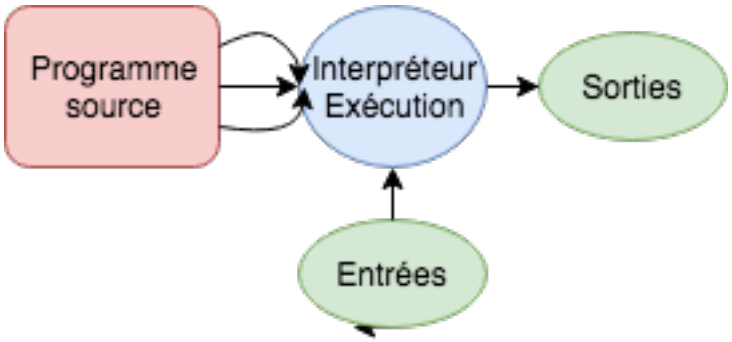
```
> gcc source.c -o prog_executable.exe
> ./prog_executable.exe
```



Analogie: Service de traduction intégrale à distance

Propriétés:

- Cible un type de machine (rapide)
- Vérification de la syntaxe à la compilation (Temps de compilation long

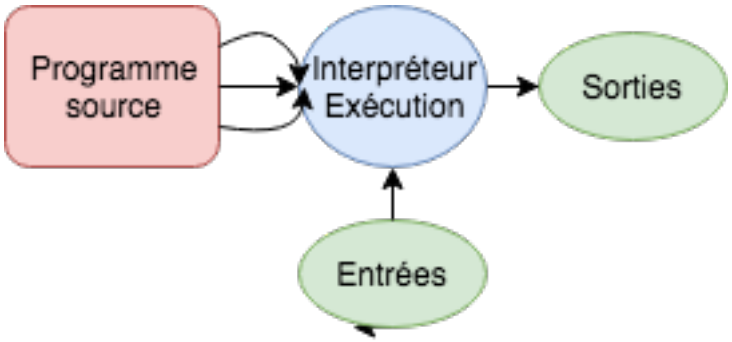


Interprétation Exemple ligne de commande Unix (bash):

```
> echo $((4+5))
9
> echo $nexistepas

> [ -x  3]
-bash: [: missing `]'
```

2.2.7 Exécution d'un programme (Interprétation) 2/2

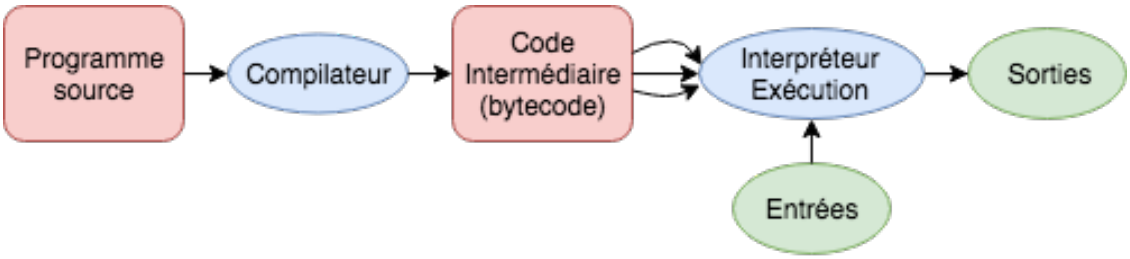


Analogie: Traduction à la volée puis exécution

Propriétés:

- Flexible car pas de cible (c'est l'interpréteur)
- Cycle Vérification + Exécution (plus lent)
- Découverte d'erreurs à l'exécution

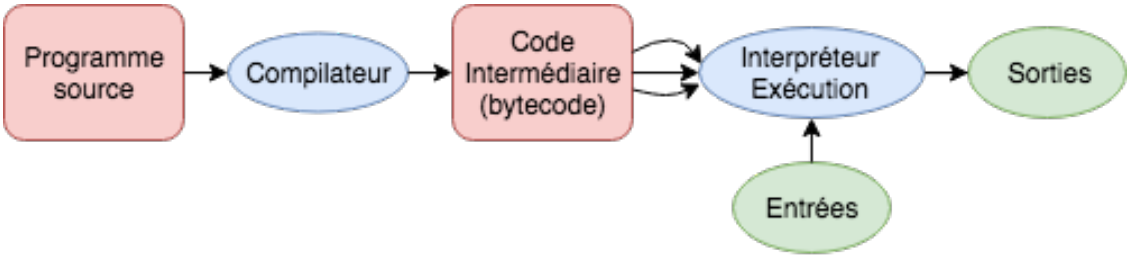
2.2.8 Exécution d'un programme (Hybride) 1/2



Compilation et interprétation d'un code Java:

```
> javac HelloWorld.java
> java HelloWorld
```

2.2.9 Exécution d'un programme (Hybride) 2/2



Le meilleur des deux mondes:

- Flexible
- Relativement rapide
- Multi-cible
- Supprime les erreurs de syntaxe à la compilation

2.2.10 Mode d'exécution - Conclusion

- De nombreux langages disposent à la fois de compilateurs et d'interpréteurs.
- C'est l'usage historique qui a fait penché la balance...
- *Python* s'exécute en mode hybride même s'il pourrait croire qu'il s'agit d'un pur interpréteur:

```
> python HelloWorld.py
```

Arguments techniques - hors cadre de l'UE: dis post

2.2.11 Les outils d'édition du code 1/2

Un "bon" *environnement de développement intégré* est un programme qui:

- Facilite l'édition du code (coloration syntaxique, saisie prédictive, ...)
- Intègre les règles de bonnes pratiques d'un langage (Ex.: PEP8 pour Python)
- Détecte les erreurs de syntaxe lors de l'édition

2.2.12 Les outils d'édition du code 2/2

Un "bon" *environnement de développement intégré* est un programme qui:

- Permet de compiler/d'interpréter un code source par un clic
- Donne accès à des outils de débogage (valeurs des variables, point d'arrêt, ...)
- Gestion de projets, des versions, lancement du code à distance, ...

2.3 Généralités sur les langages de programmation

2.3.1 Quelques paradigmes

On peut catégoriser les langages suivants des propriétés qui les caractérisent:

- Langages impératifs
- Langages procéduraux
- Langages à objets
- Langages déclaratifs

et bien d'autres ...

La plupart des langages sont multi-paradigmes.

2.3.2 Langage impératif 1/2

Un langage est dit impératif lorsque le programme correspond à une succession d'instructions:

```
Instruction 1
Instruction 2
...
Instruction n
```

Chaque instruction tient compte de l'état du système (mémoire, E/S, ...) et peut le modifier.

2.3.3 Langage impératif 2/2

Exemples typiques d'instructions:

- Affectations d'une valeur à une *variable*
- Le saut conditionnel "If"
- Les répétitives "Pour" et "Tant que"
- Optionnel: le saut inconditionnel "Goto"

2.3.4 Langage procédural

Il s'agit simplement de pouvoir regrouper un ensemble d'instructions nommé **procédure** (ou *routine* ou *sous-routine*).

```
Procédure Quelconque:
    Instruction 1
    Instruction 2
    ...
    Instruction n
```

On parle de **programme modulaire** lorsque regroupe thématiquement des procédures dans un *module* (ou *bibliothèque* ou *paquet*)

2.3.5 Langage orienté objet 1/2

Dans ce type de langage, les programmes sont organisés autour de briques logicielles appelées **Objets** qui:

- représente un concept, une entité réelle ou non
- possède une représentation interne (*attributs* ou *slots*)
- disposent de *méthodes* ou *slots*:
  - récupérer/changer sa répresentation interne
  - exécuter des traitements
  - interagir avec d'autres objets

2.3.6 Langage orienté objet 2/2

```
class Ballon {
   double rayon;          // Attribut
   Ballon(double rayon) { // Construction
     this.rayon = rayon;
   }
   void gonfler() {       // Méthode
     this.rayon = this.rayon + 1;
   }
   void collision(Ballon autre){
     double rayon = (this.rayon + autre.rayon)/2;
     this.rayon = autre.rayon = rayon;
   }
 }
```

2.3.7 Aparté : Langage déclaratif

On parle de *programmation déclarative* lorsque l'on décrit le résultat attendu, les objectifs (**quoi**) sans donner la manière de le faire (**comment**). Exemple: page HTML5 minimaliste

```
<!DOCTYPE html>
<html>
<head><title>This is Hello World page</title></head>
<body>
```

```
<h1>Hello World</h1>
</body>
</html>
```

2.4 Généralités sur le langage Python

2.4.1 Caractéristiques du langage

Créé en 1991, Python est un langage multi-paradigme:

- Impératif et procédural, orienté-objet
- Fourni avec un interpréteur (usage dominant) qui:
  - infère le type des variables lors de l'exécution (typage dynamique)
  - gère la mémoire automatiquement (ramasse-miettes)
- qui dispose d'une grande bibliothèque de base (modules)
- Version actuelle: **3.9** (3.10 en cours)

2.4.2 Outils d'édition du code Python

- *Thonny* (cette UE, S2)
- *Visual Studio Code*
- *Spyder* (Calcul scientifique)
- NetBeans, *PyCharm*
- *Jupyter Notebook* (voir S2)

Liste complète

2.4.3 Exemple d'un programme Python

```
print('Hello World!!')
```

Test via Thonny:

- en mode interactif
- en mode script

Par convention, les scripts Python ont pour extension ****.py***

2.5 Les variables

2.5.1 Variable (définition)

Une variable est une zone de la mémoire dans laquelle une valeur est stockée.

Elle est désignée par:

- un **nom** pour le programmeur
- une **adresse** pour l'ordinateur

La fonction *id* renvoie un nombre unique (~ adresse) qui qualifie une variable.

2.5.2 Type d'une variable

Le **type** d'une variable définit les opérations valides pour elle.

Types élémentaires:

- Les nombres entiers
- Les nombres "réels" ou "à virgule"
- Les chaînes de caractères
- Les booléens

La fonction *type* renvoie le type d'une variable.

2.5.3 Déclaration d'un variable

En Python, la **déclaration** et l'**initialisation** d'une variable se fait de manière simultanée.

```
In [1]: a = 2

In [2]: type(a)
Out[2]: int

In [3]: id(a)
Out[3]: 4525733248
```

Remarques sur le typage 1/2

- Le type d'une variable est donné par le type de l'expression à droite du = (*Inférence de type*)
- Toute variable a un type (*Typage fort*) (qui peut changer lors de l'éxécution (*Typage dynamique*))

Remarques sur le typage 2/2

```
In [1]: a = 2

In [2]: type(a)
Out[2]: int

In [3]: a = 3.14159

In [4]: type(a)
Out[4]: float

In [5]: type(3.14159)
Out[5]: float
```

2.6 Les types de base de Python

2.6.1 Les types numériques: *int*

- Entier avec précision arbitraire

- Attention, le type *long* existait dans les versions précédentes

```
In [1]: type(2)
Out[1]: int

In [2]: 2**1024
Out[2]: 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708
```

2.6.2 Les types numériques: *float*

- nombre à virgule avec une précision **fixe** $$[2.26.. * 10^{-308}, 1.79.. * 10^{308}]$$
- Tous les réels **ne sont pas représentables** par le type *float*

```
In [1]: 1.79e308
Out[1]: 1.79e+308

In [2]: 1.79e308*10
Out[2]: inf

In [3]: type(1.79e308*10)
Out[3]: float

In [4]: 1e20 + 1    # voir cours architecture des ordinateurs (1e20 + 1 = 1e20 ?)
Out[4]: 1e+20
```

2.6.3 Autres types numériques

Seront également évoqués si besoin en TP:

- *complex* : les nombres complexes
- *decimal* : les nombres décimaux (précision fixée arbitraire)
- *fraction* : les nombres rationnels (rapport entre deux entiers)

pour mémoire: *float* $\subset$ *decimal* $\subset$ *fraction* $\subset \mathbb{R}$

type *int* := $\mathbb{Z}$

2.6.4 Opérations sur les types numériques

- Arithmétique usuelle: +, -, *, /
- Division entière: a // b (arrondi vers $-\infty$)
- Reste de la division entière: a % b
- Puissance: a**b ou *pow*(a,b)
- Valeur absolue: *abs*(a)

2.6.5 Le type booléen

- Il permet de représenter les valeurs de vérité *True* ou *False*
- Opérations sur les booléens (priorité décroissante):
  - *not*: négation logique

  - *and*: “et” logique
  - *or*: “ou” logique

```
In [1]: type(True)
Out[1]: bool

In [2]: not False and True
Out[2]: True
```

2.6.6 Les chaines de caractères 1/2

Les chaînes de caractères sont délimitées par les simples ou doubles guillemets.

L’opération de **concaténation** de deux chaînes est effectuée par le symbole +.

2.6.7 Les chaines de caractères 2/2

```
In [1]: a = 'abc'

In [2]: b = 'def'

In [3]: a + b
Out[3]: 'abcdef'
```

Une version moderne, les f-strings ($\geq$ 3.6) Permet de construire une chaîne de caractères à partir d’expressions

```
In [1]: nom = "toto"
In [2]: age = 8

In [3]: f"Salut {nom} !"
Out[3]: 'Salut toto !'

In [4]: f"Le double de {age} est {2*age}"
Out[4]: 'Le double de 8 est 16'
```

2.6.8 Le type NoneType

- *None* est une valeur spéciale pour indiquer qu’une variable *existe* mais *n’a pas de contenu connu* (valeur manquante)
- Peut indiquer également “n’a pas de sens”

```
In [1: a
...
NameError: name 'a' is not defined

In [2]: a = None

In [3]: a
```

2.6.9 Conversion implicite entre types numériques

```
In [1]: type(1+2.)
Out[1]: float

In [2]: 2**100000 + 5.
...
OverflowError: int too large to convert to float
```

Ccl: calcul mixte int/float -> float

2.6.10 Conversion implicite entre types 1/2

Par **convention**, le *True* "équivaut" à 1 et *False* à 0 dans les calculs numériques.

```
In [1]: True * 2
Out[1]: 2
```

2.6.11 Conversion implicite entre types 2/2

A l'opposé, *0, 0.* ou *None* sont considérées comme *False*, sinon *True* dans les expressions booléennes.

```
In [1]: not 0
Out[1]: True
```

2.6.12 Conversion explicite entre types

On peut forcer la conversion avec la syntaxe:

```
<type>(<expression>)
```

Exemples:

```
In [1]: int(4.7)
Out[1]: 4

In [2]: bool(0)
Out[2]: False

In [3]: str(4.7)
Out[3]: '4.7'

In [4]: float('4.7')
Out[4]: 4.7
```

2.7 Entrées-Sortie Clavier/Ecran

2.7.1 Affichage à l'écran

Syntaxe sur la fonction *print*:

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

- ... : *print* accepte une suite de valeurs converties en *str*
- sep: le séparateur
- end: le caractère de fin
- file : flux de sortie (sys.stdout est l'écran)
- flush : force à vider le flux immédiatement.

Exemples d'affichage 1/2

```
In [1]: année = 2021

In [2]: print(année)
2021

In [3: print("année", année)
année 2020
```

Exemples d'affichage 2/2

```
In [4]: print("l'année", année, end=""); print(" est un bon cru !!")
l'année  2021 est un bon cru !!

In [5]: print(année, année*2, année / 2, type(année), sep=", ")
2021, 4042, 1010, <class 'int'>
```

On remarque:

- Il n'a pas de "Out" (Output) pour *print*
- *Confusion usuelle*: l'affichage à l'écran n'est pas un retour de la fonction *print*.

2.7.2 Saisie Utilisateur

Fonction *input* :

```
input(prompt=None)
```

- retourne la chaîne de caractères saisie.
- Il est souvent nécessaire de **convertir explicitement dans le type désiré** ensuite.

```
In [1]: a = int(input('Nombre de pas ? '))
Nombre de pas ? 12

In [2]: b = float(input('pi ? '))
pi ? 3,14
...
ValueError: could not convert string to float: '3,14'
```

C'est la convention anglaise pour les *float*.

2.8 Structures conditionnelles

if, else, elseif

2.8.1 Instruction if, else

```
if <condition>:
    <instructions si True>
[else:
    <instructions si False>]
```

- La partie else est optionnelle
- **L'indendation est obligatoire car elle marque le début et la fin d'un bloc d'instructions**

Condition dans un if 1/2 Une condition peut-être:

- une expression booléenne construite par:
  - un opérateur de comparaison: <=, <, >, >=
  - un opérateur d'égalité: ==, !=, <>
  - un opérateur d'identité: *is*, *is not*
  - un opérateur d'appartenance : *in*, *not in*

Condition dans un if 2/2 Une condition peut-être aussi:

- une valeur numérique: 0 ou 0. équivaut à *False* sinon *True*
- une chaine de caractères: '' équivaut à *False* sinon *True*
- *None* équivaut à *False*
- *etc*

Les conditions peuvent être combinées par *not*, *and* et *or*

Exemples if

```
In [1]: a = int(input('Nombre de pas ? '))
Nombre de pas ? 9680

In [2]: if a > 8000:
          print('Vous avez fait le nombre de pas journaliers recommandé.')
        else:
          print('Faire plus d\'exercice.')
Vous avez fait le nombre de pas journaliers recommandé.

In [3]: if a > 8000: print('bravo')
bravo
```

2.8.2 Instruction elif

- *elif* est une contraction de *else if* qui n'oblige pas à une indentation supplémentaire
- C'est le "switch case" de Python.

```
In [1]: a = 7

In [2]: if a < 0:
          print('a est négatif')
        elif a % 2 == 0:
```

```
        print('a est pair')
      elif a % 3 == 0:
        print('a est divisible par 3')
      else:
        print('a est un nombre positif, impair et non divisible par 3')
```
```
a est un nombre positif, impair et non divisible par 3
```

2.8.3 Démonstration !

- Conditions if imbriquées : a $\in$ [0,1]
- Année bissextile