Introduction à la programmation

Christophe Saint-Jean

https://gitpitch.com/christophesaintjean/cours/IntroProgS1

Année 2018-2019

@transition[fade]

Organisation de l'UE

+++

L'équipe enseignante

- Sanae Boutarfass (TP)
- Laurent Mascarilla (TP)
- Matthieu Robert (TP)
- **Christophe Saint-Jean** (Cours/TP - Resp.)
- El Hadi Zahzah (TP)
- *Etc*

+++

Communication

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

+++

Dispositif horaire

@ul

- 5 cours de 1,5 heures (Amphithéâtre)
- 10 TPs de 1,5 heures (Sallles de TP)
- 2 créneaux de 1,5h de TEA (Salles de TP)

@ulend

+++

Evaluation

$S_1 = (CC_1+CC_2)/2$

$S_2 = CC_3$

Les CC se passent en TP sur machine:

- Une partie QCM
- Une partie évaluée par un enseignant
- Attention à la règle sur les absences
- Le TEA sera pris en compte dans l'évaluation

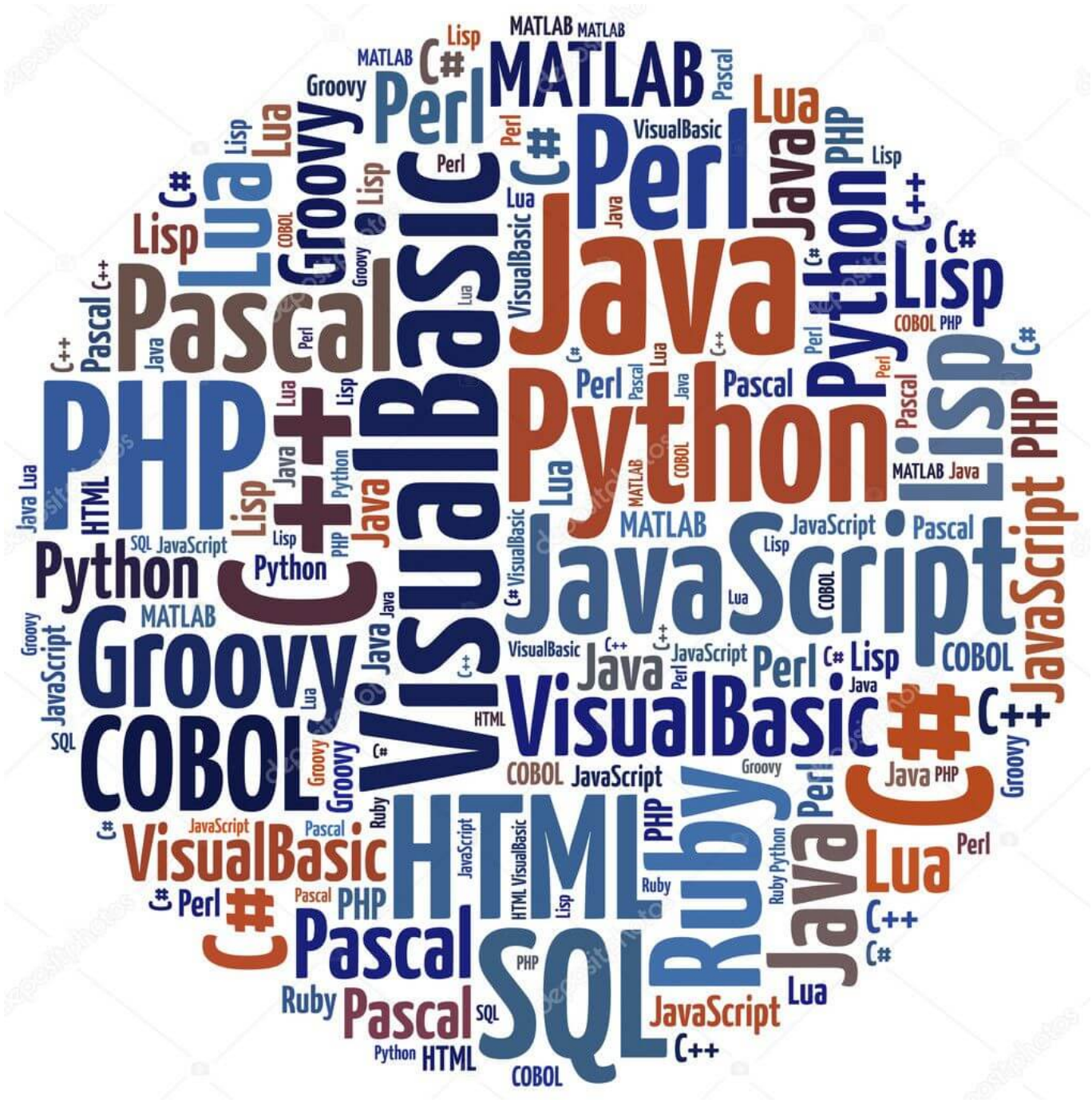
+++

Les objectifs de cet enseignement

- Découvrir les bases de la programmation informatique
- Développer l'esprit logique par la pratique de la programmation
- Appprendre un des langages support de votre formation
- Libérer votre créativité

+++

Sondage !



@transition[fade]

Généralités sur la programmation

Langage humain

- Un certain ***vocabulaire***, ***une orthographe***, des ***règles de grammaire*** communes
- Grande expressivité et diversité
- Même si l'on commet des erreurs, nous sommes capables de comprendre "globalement" le message

+++

Langage informatique

- La machine traite des informations binaires: 1001100101010000110 ... (même si images, sons, *programmes*, ...)
- Le vocabulaire d'instructions machine est très réduit
- Pas ou peu (encore) d'expressivité
- Pas de tolérance aux erreurs d'instructions

Niveau d'un langage

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

- 1001100101010000110 ... (Quasi-impossible)
- Langages de bas niveau (Ex. Assembleur)
- Langages de bas/haut niveau (Ex. C)
- Langages de haut niveau (Ex. Python, Java, C++, R, ...)

+++

Programme Source (ou Code)

Un programme source est un ***texte*** qui dépend du langage de programmation:

- Utilise un certain nombre de ***conventions*** (nommage, opérations, ...),
- Obéit à des règles de *syntaxe*, de *grammaire*
- En géneral, sauvegardé puis exécuté depuis un fichier.

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

+++

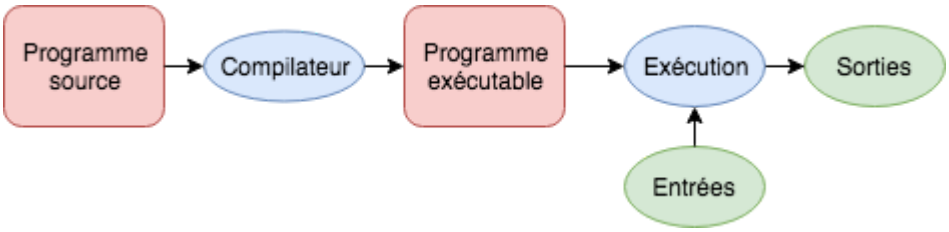
Les différentes sources d'erreur

@ul

- Erreurs de syntaxe: Non respect des conventions du langage
- Erreurs d'exécution (Runtime-Error): Opération non valide lors de l'exécution
- Erreurs sémantiques: Résultat différent de celui désiré

@ulend

Exécution d'un programme (Compilation) 1/2

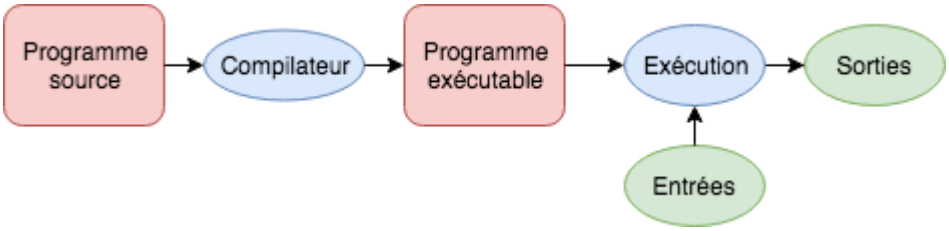


Compilation et exécution pour le compilateur gcc:

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

+++

Exécution d'un programme (Compilation) 2/2



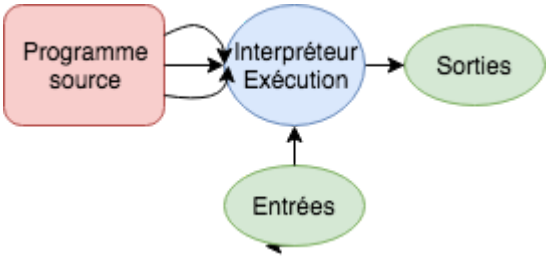
@ul

Analogie: Service de traduction intégrale à distance

- Avantages:
  - Rapidité
  - Vérification de la syntaxe à la compilation
- Inconvénients:
  - Temps de compilation (une fois)
  - Mono-cible

@ulend

Exécution d'un programme (Interprétation) 1/2



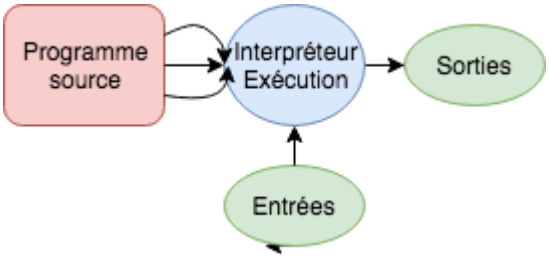
Exemple ligne de commande Unix (bash):

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

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

+++

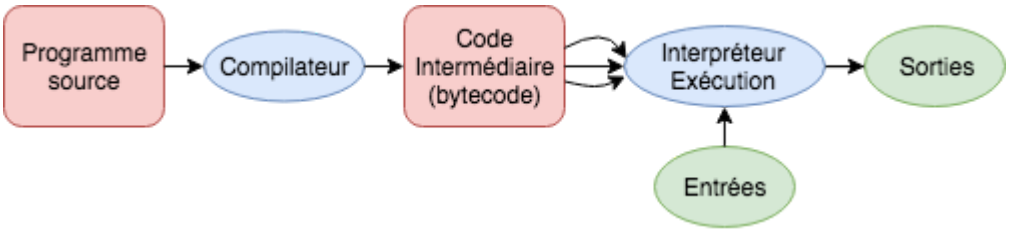
Exécution d'un programme (Interprétation) 2/2



Analogie: Traduction à la volée puis exécution

- Avantages:
  - Flexibilité
  - Multi-cible (lié à l'interpréteur)
- Inconvénients:
  - Lenteur (/compilateur)
  - découverte d'erreurs à l'exécution

Exécution d'un programme (Hybride) 1/2

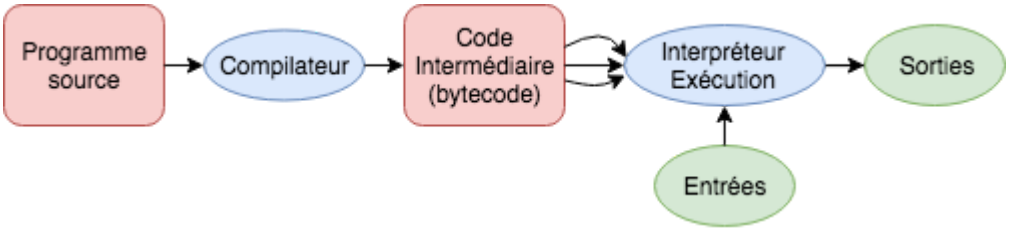


Compilation et interprétation d'un code Java:

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

+++

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

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

Les outils d'édition du code

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
- Permet de compiler/d'interpreter 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, ...

@transition[fade]

Généralités sur les langages de programmation

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.

+++

Langage impératif 1/3

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.

+++

Langage impératif 2/3

On y trouve le même genre d'instructions:

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

+++

Langage impératif 3/3

C'est le type de langage qui domine depuis plus de 30 ans !

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

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 n

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

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

+++

Langage orienté objet 2/2

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

Aparté : Langage déclaratif

On parle de *programmation déclarative* lorsqu'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>
```

@transition[fade]

Généralités sur le langage Python

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: **3.7.0** (ou *2.7.15*)

Outils d'édition du code Python

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

Liste complète

Exemple d'un programme Python

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

Test via Thonny:

- en mode interactif
- en mode script

> Les scripts Python ont pour extension *.py*

@transition[fade]

Les variables

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.

Type d'une variable

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

Types élémentaires:

- Les nombres entiers
- Les nombres réels
- Les chaînes de caractères
- Les booléens

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

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 qui peut changer lors de l'éxécution (*Typage dynamique fort*)

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
```

Les types de base de Python

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]:
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

Les types numériques: *float*

- nombre à virgule avec une précision **fixe** $$[2.2250738585072014*10^{-308}, 1.79769313486231570*10^{308}]$$
- Attention tous les nombres réels ne sont pas représentables...

```
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
Out[4]: 1e+20
```

Autres types numériques

Seront également évoqués si besoin en TP:

- *complex* : les nombres complexes
- *decimal* : les nombres décimaux
- *fraction* : les nombres rationnels

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

int := $\mathbb{N}$

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)

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

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 +.

Les chaines de caractères 2/2

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

In [2]: b = 'def'

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

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 [28]: a
...
NameError: name 'a' is not defined

In [29]: a = None

In [30]: a
```

Conversion implicite entre types numériques

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

In [2]: 2**100000 + 5.
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call
last)
<ipython-input-10-3eb3822c8b2d> in <module>()
----> 1 2**100000 + 5.

OverflowError: int too large to convert to float
```

Ccl: calcul mixte int/float -> float

Conversion implicite entre types

Par **convention**, le *True* "équivaut" à 1 et *False* à 0.

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

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

Conversion explicite entre types

On peut forcer la conversion avec la syntaxe: $$()$$ 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'
```

Ecriture non formatée/formatée

Saisie Utilisateur

Saut conditionnel : If