



# INTRODUCTION À LA PROGRAMMATION

*1re Technologie de l'informatique*

*1re Sécurité des systèmes*



# CONTACTS

- THÉORIE ET ATELIERS

- [bastien.bodart@henallux.be](mailto:bastien.bodart@henallux.be)
- [cecile.pirotte@henallux.be](mailto:cecile.pirotte@henallux.be)

- EXERCICES

- [bastien.bodart@henallux.be](mailto:bastien.bodart@henallux.be)
- [mohamed.bouraada@henallux.be](mailto:mohamed.bouraada@henallux.be)
- [adrien.huygens@henallux.be](mailto:adrien.huygens@henallux.be)
- [cecile.pirotte@henallux.be](mailto:cecile.pirotte@henallux.be)
- [anne.smal@henallux.be](mailto:anne.smal@henallux.be)
- [didier.valentin@henallux.be](mailto:didier.valentin@henallux.be)

# CRITÈRES DE RÉUSSITE

## Écrire des programmes « qui tournent »

RÉUSSITE  $\Rightarrow$  programmes tournent  
programmes tournent  $\nRightarrow$  RÉUSSITE

Il faut aussi...

- que les programmes soient **propres**,
- que les programmes soient **efficaces**,
- **comprendre** pourquoi ils tournent !

# PROGRAMMES PROPRES

## Écrire du code propre (« clean code »)

Il doit être **facile à lire et à modifier**

- **compréhensible**,
- bien **structuré**,
- avec des **noms explicites**,
- etc.

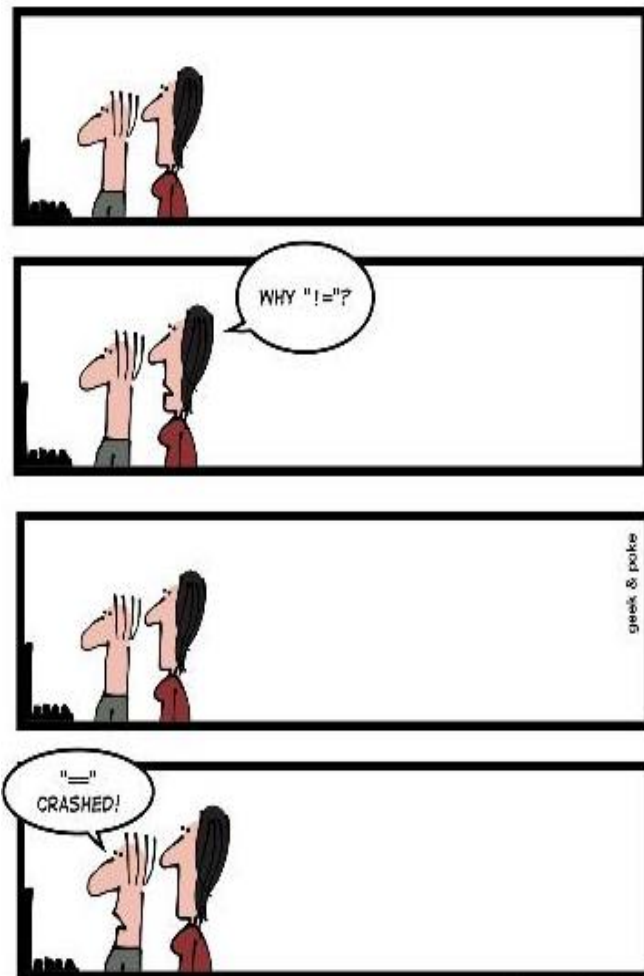


# PROGRAMMES EFFICACES

## Réfléchir avant de coder

Il est important de

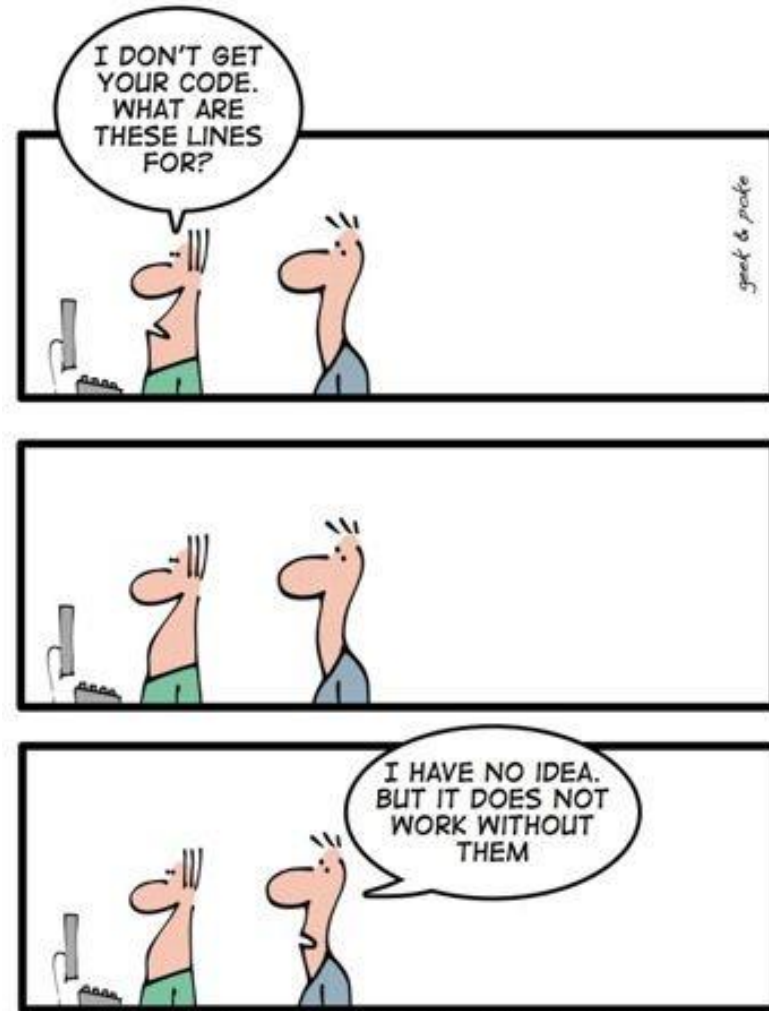
- réfléchir à **l'algorithmique**,
- utiliser les **instructions les plus adéquates**,
- choisir des **structures de données appropriées**,
- etc.



# COMPRENDRE

## Comprendre ce que vous lisez/écrivez

- Ne pas coder par **essais/erreurs**...
- Éviter les **rustines**,
- Comprendre le « **pourquoi** » des erreurs et en déduire la correction,
- Etc.



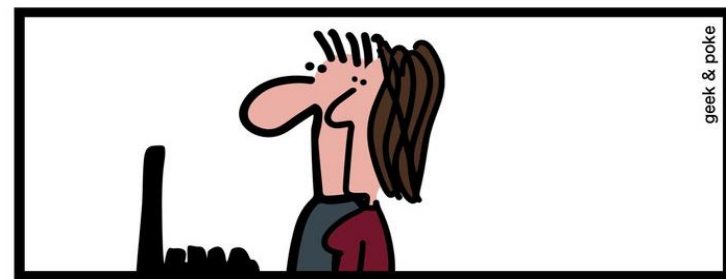
# RESTEZ TOUJOURS CRITIQUES !

## Ne recopiez pas bêtement le code d'internet !

La plupart des bouts de code qui trainent sur le web sont truffés de fautes !

Lisez-le avant de le copier,

- s'il est incompréhensible, **oubliez-le**,
- s'il est compréhensible mais mal écrit, **améliorez-le**,
- etc.



GOOD QUESTIONS

# GÉRER VOTRE TEMPS

- Pour réaliser les ateliers et les exercices,
  - vous devez travailler en dehors/en plus des séances d'exercices !
- vous pouvez
  - en discuter entre vous et poser des questions aux profs lors de la séance,
  - en discuter entre vous et échanger vos idées sur Discord ou autres,
  - poser des questions aux profs via Discord ou par mail.



# PLAN

- **Module 0 : Introduction et notions de base**
- Module 1 : Variable, opérateurs et entrées/sorties
- Module 2 : Fonctions sans paramètres
- Module 3 : Fonctions avec paramètres et retour
- Module 4 : Structures de contrôle
- Module 5 : Séquence
- Module 6 : Dictionnaires et déballage

# ORGANISATION

3 phases par module :

- des **ateliers**

- ↪ présentation d'éléments théoriques au travers de petits exercices dirigés à réaliser en autonomie

- une **séance de mise en commun**

- ↪ révision des éléments vus lors de l'atelier au travers d'explications et d'exercices

- des **séance d'exercices**

- ↪ exercices moins dirigés permettant d'approfondir sa compréhension de la matière

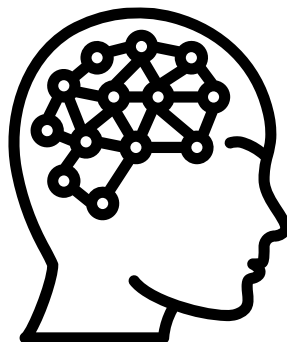
# ORGANISATION

At. 0 - Introduction (algorithmique et Python + cours)	
Ex. 0 - IDLE et Console	
At. 1 - Variables, opérateurs, entrées et sorties	
At. 1 - Variables, opérateurs, entrées et sorties	
Th. 1 - Variables, opérateurs et entrées/sorties	
At. 2 - Structures de contrôle	Ex. 1 - Bases
At. 2 - Structures de contrôle	
Th. 2 -Structures de contrôle	
At. 3 - Fonctions avec paramètres et avec retour	Ex.2 - Structures de contrôle
At. 3 - Fonctions avec paramètres et avec retour	Ex.2 -Structures de contrôle
Th. 3 - Fonctions avec paramètres et retour	
Ex.3 - Fonctions avec paramètres et retour	
Ex.3 - Fonctions et pile d'appels	
Interro	
At. 4 -Séquences et for	
At. 4 - Séquences et for	
Th. 4 - Séquences et for	
At. 5 - Dictionnaires	Ex. 4 - Séquences et for
At. 5 - Dictionnaires	Ex. 4 - Séquences et for
Th. 5 - Dictionnaires	
At. 6 - Dictionnaires et déballage	Ex. 5 - Dictionnaires
At. 6 - Dictionnaires et déballage	Ex. 5 - Dictionnaires
Th. 6 - Dictionnaires et déballage	
Ex. 6 - Dictionnaires et déballage	
Ex. 6 - Dictionnaires et déballage	
Ex. fun - micro:bit	
Ex. fun - micro:bit	
Ex. sup - Ratrapage	

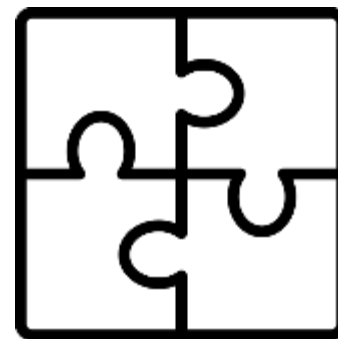
# JOUEZ LE JEU !



Atelier



Théorie



Exercices

Découverte

**PDF + IDLE**

Explications

**PDF + Wooclap**

Applications

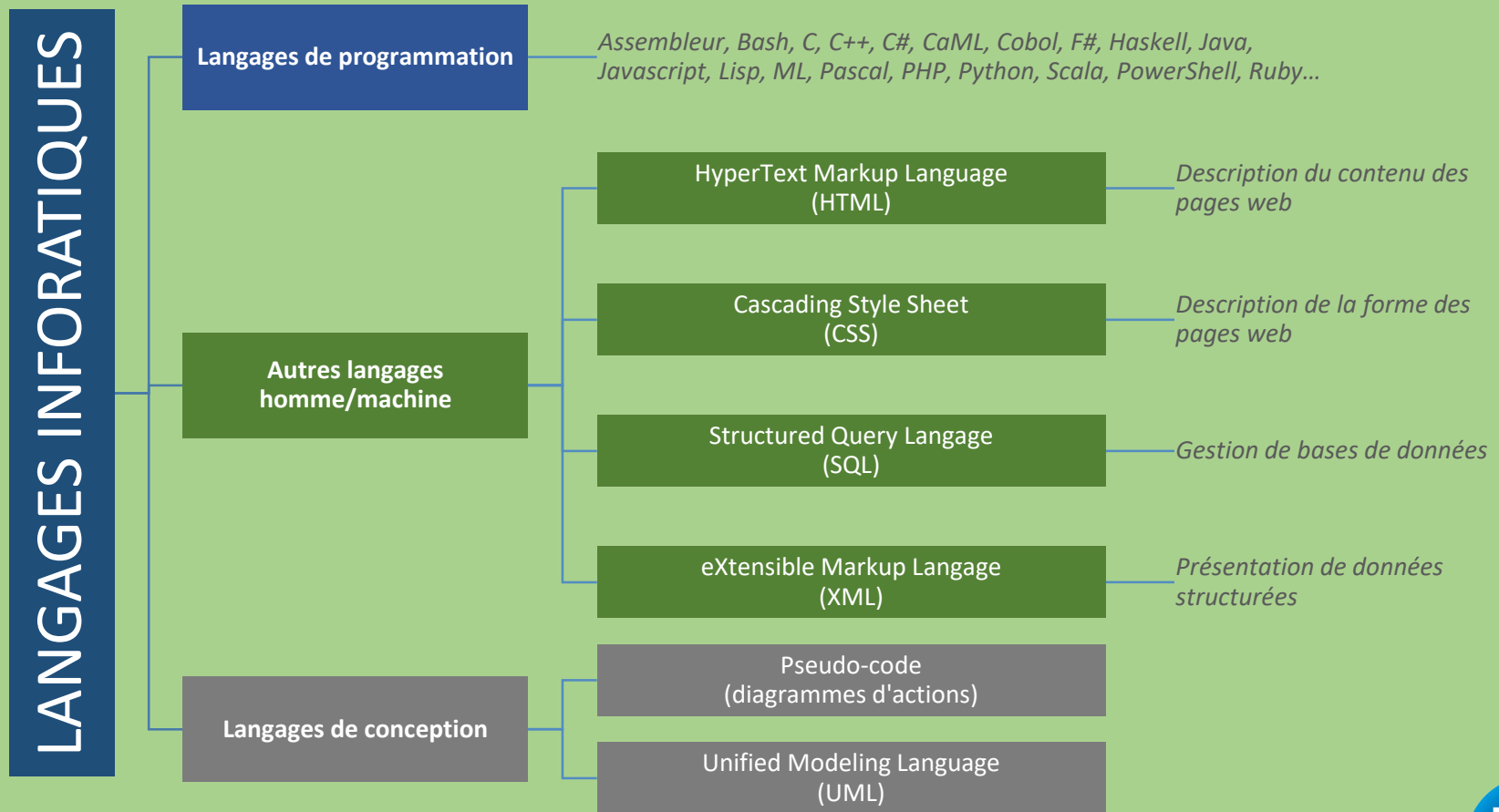
**PDF + IDLE**

**Quizz → Moodle**

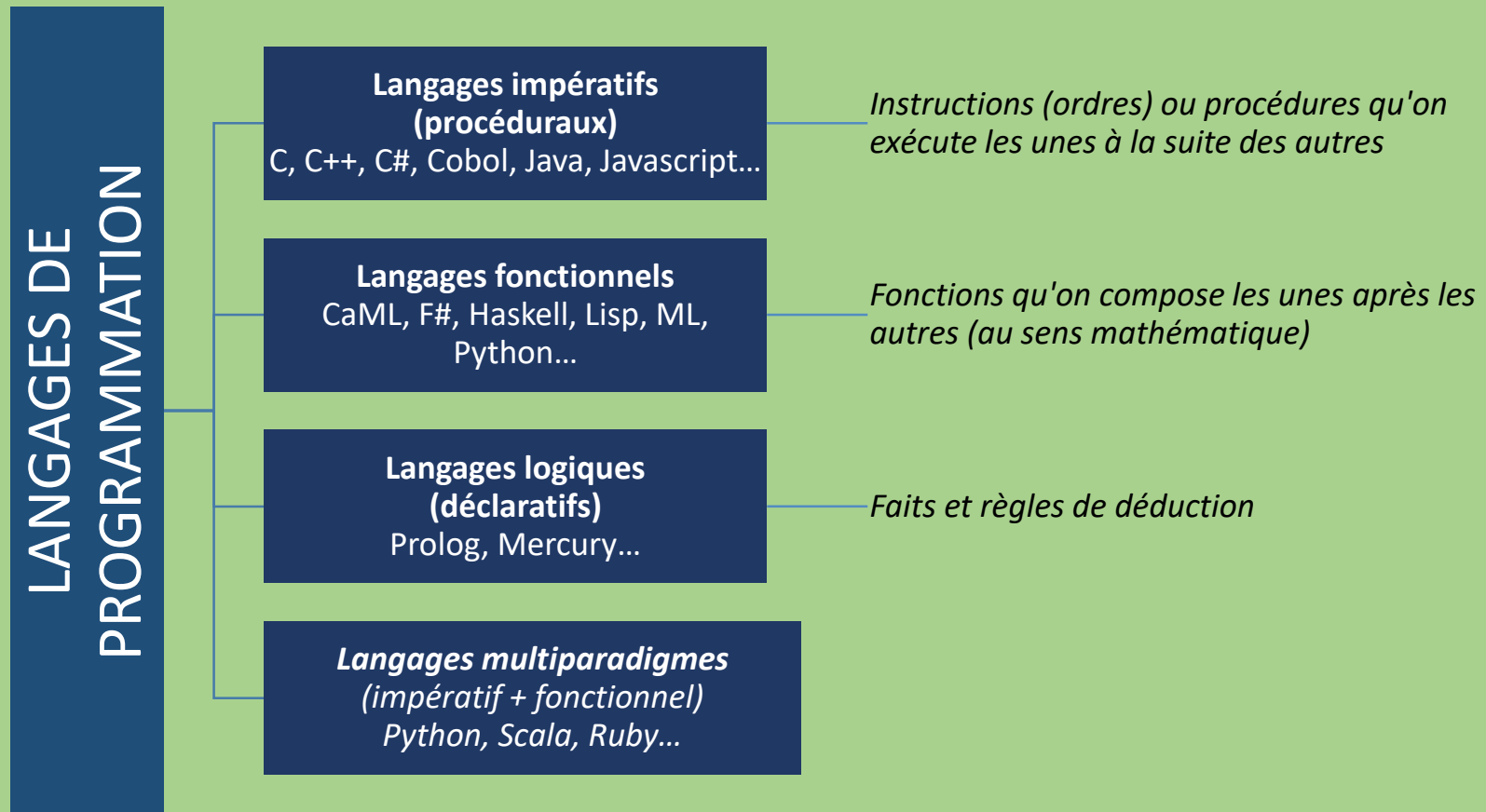
# MODULE 0

- Une séances de présentation en auditoire
  - Découvrir l'environnement de travail
  - Apprendre quelques notions de base
- Une séance d'exercices pour découvrir IDLE...

# LES LANGAGES DE L'INFORMATIQUE



# LES LANGAGES DE L'INFORMATIQUE



À lire : [https://fr.wikipedia.org/wiki/Langage\\_de\\_programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation)

# QUELQUES QUESTIONS...

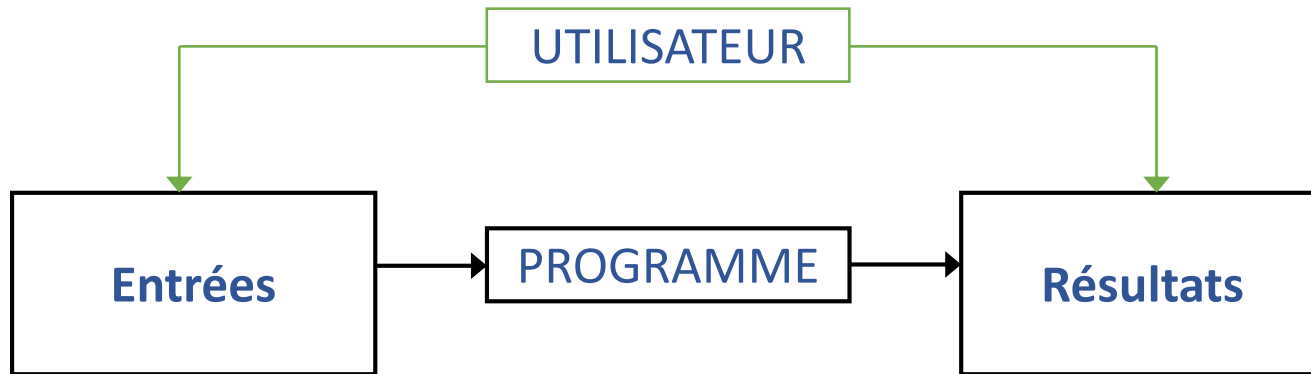
- Utilisateur ou programmeur ?
- Qu'est-ce qu'un algorithme ?
- Qu'est-ce que l'algorithmique ?
- Qu'est-ce qu'un programme ?
- Qu'est-ce que la programmation ?
- Comment le programme est-il exécuté ?



# Utilisateur ou programmeur ?

## ■ Utilisateur

Il doit connaître le mode d'emploi du programme pour savoir ce qu'il fait et ce qu'il doit lui donner pour avoir les résultats attendus.



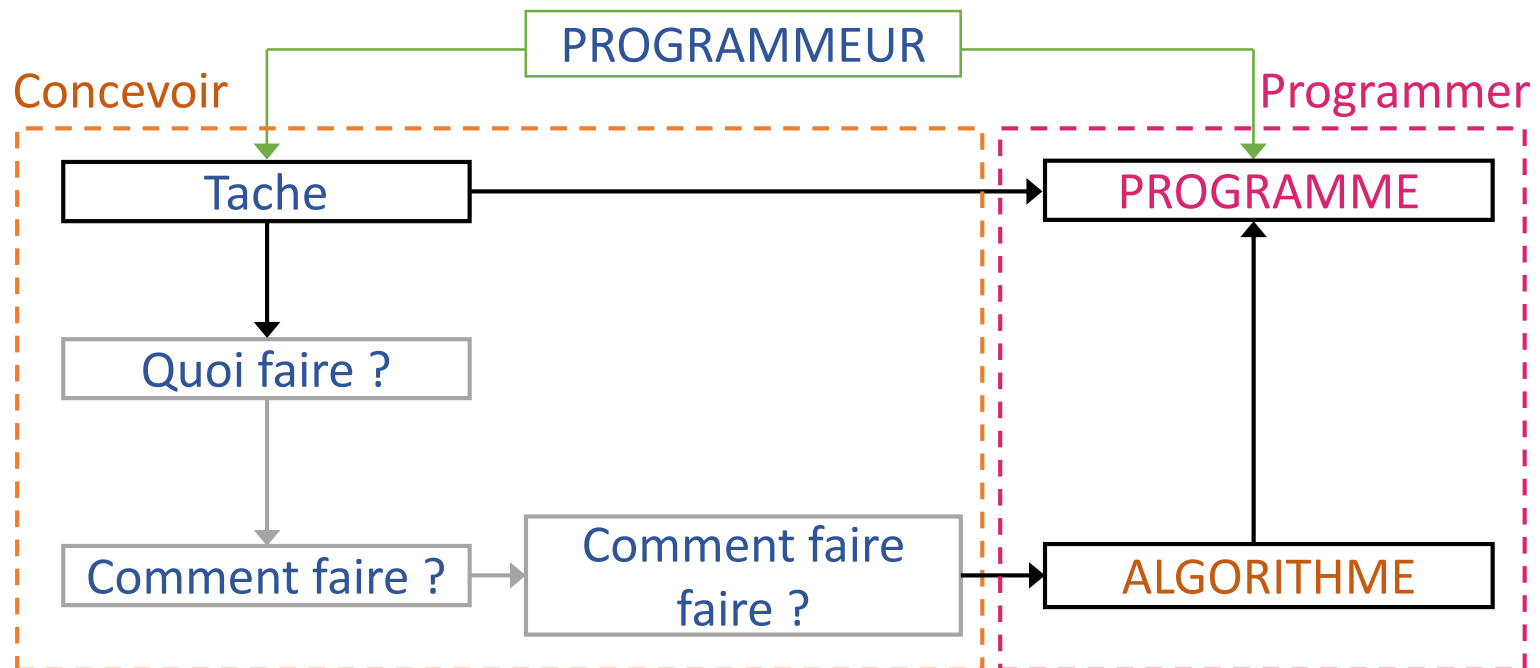
Identifiant  
Mot de passe → Accès à Moodle

Points par UE → Admis ou non

# Utilisateur ou programmeur ?

## ■ Programmeur

Il doit développer un **algorithme** et le transformer en **programme** que l'ordinateur puisse exécuter de façon à répondre aux attentes de l'utilisateur.



# Qu'est-ce qu'un algorithme ?

**Suite d'instructions qui permet d'obtenir un résultat.**

- algorithme exact  $\Rightarrow$  le résultat est celui attendu
- algorithme inexact  $\Rightarrow$  le résultat est indéfini...

## Exemples

- Recette de cuisine
- Notice de montage
- Résolution du Rubik's Cube
- Calcul du  $n^{\text{ième}}$  nombre de la suite de Fibonacci

# Exemple

## Les fêtes de Wallonie !

Gérer une commande de boissons (en très simplifié).

Il faut, en respectant l'ordre...

- Proposer les boissons disponibles : peket ou coca
- Demander quelle boisson le client désire
- Demander la quantité
- Calculer le prix de la commande ( $\text{prix} * \text{quantité}$ )
- Demander la somme due
- Si la somme est supérieure
  - Compter combien on a reçu en trop
  - Rendre la monnaie
- Donner la/les boisson(s)

# Qu'est-ce que l'algorithmique ?

Science des algorithmes, qui inclus entre autre l'étude de leur conception et de leur complexité.

Ce cours porte **entre-autre sur la conception d'algorithmes** : vous devez imaginer une suite d'instructions ou d'opérations qui permettent d'aboutir au résultat voulu, une solution (correcte) à un problème donné.

Aucune notation particulière n'est imposée pour cette étape... lors de vos réflexions, vous pouvez noter vos idées en français, sous la forme de dessins ou en Python.

# Qu'est-ce qu'un programme ?

**Séquence d'instructions exécutables par l'ordinateur.**

Une fois l'algorithme établi, il peut alors être programmé au moyen d'un langage de programmation. Ensuite il est traduit en langage machine et exécuté par un ordinateur.

## **Langage de programmation**

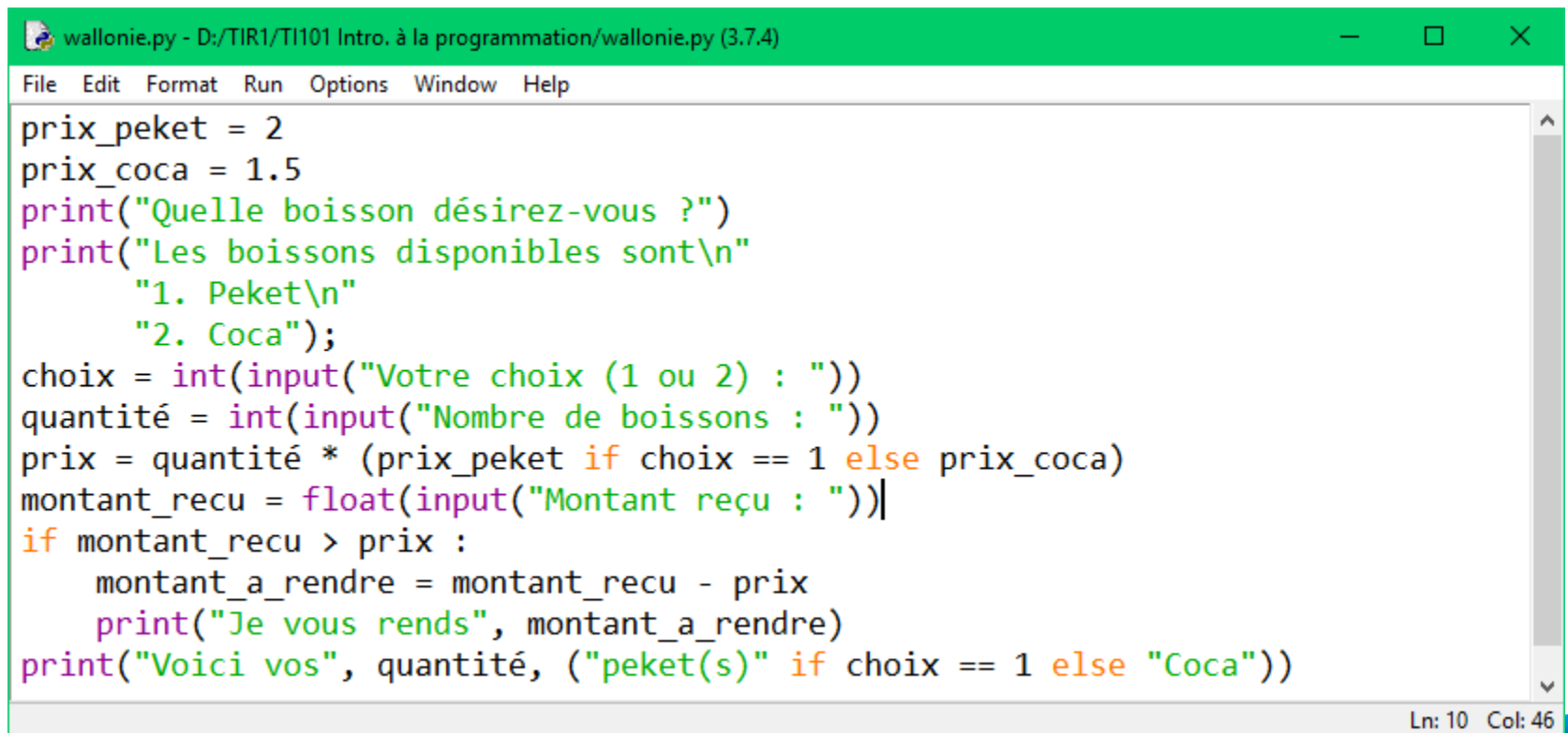
Intermédiaire entre l'humain et la machine.

↳ Permet d'écrire dans un langage proche de la machine, mais intelligible par l'humain, les opérations que l'ordinateur doit effectuer.

# Exemple

## Les fêtes de Wallonie !

Gérer une commande de boissons (en très simplifié).



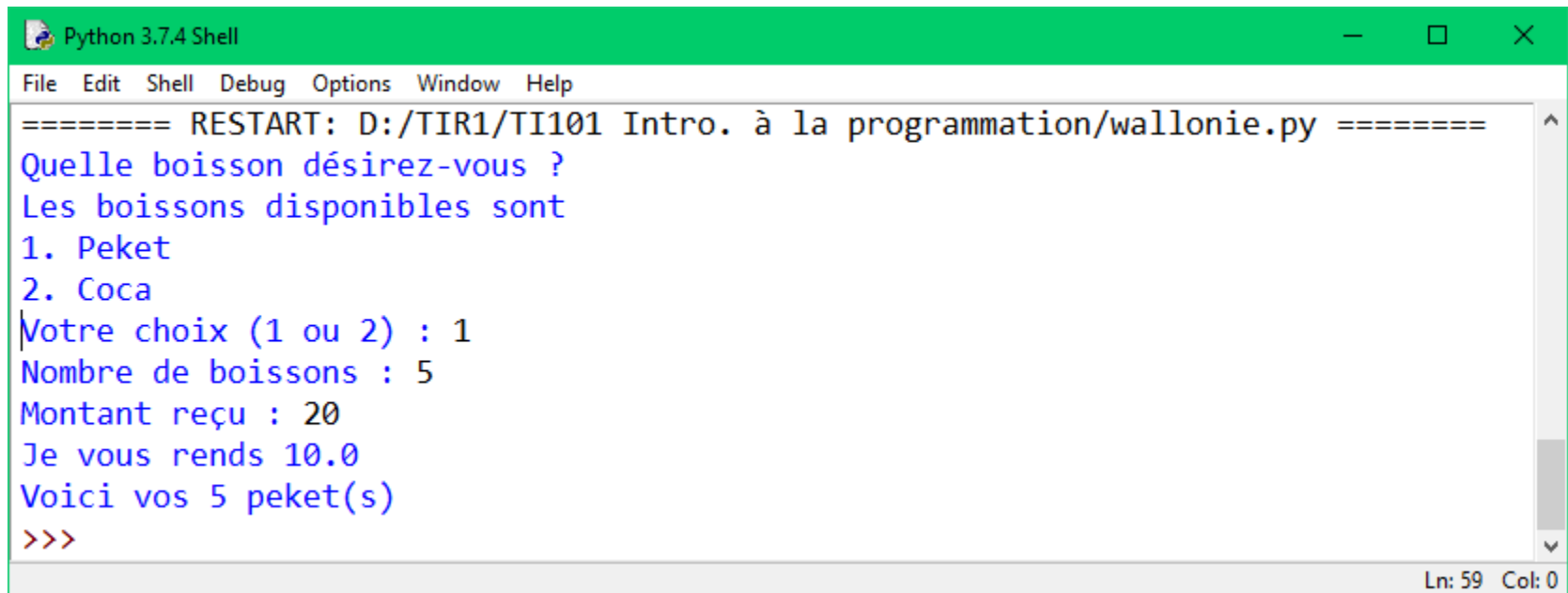
```
wallonie.py - D:/TIR1/TI101 Intro. à la programmation/wallonie.py (3.7.4)
File Edit Format Run Options Window Help
prix_peket = 2
prix_coca = 1.5
print("Quelle boisson désirez-vous ?")
print("Les boissons disponibles sont\n"
      "1. Peket\n"
      "2. Coca");
choix = int(input("Votre choix (1 ou 2) : "))
quantité = int(input("Nombre de boissons : "))
prix = quantité * (prix_peket if choix == 1 else prix_coca)
montant_recu = float(input("Montant reçu : "))
if montant_recu > prix :
    montant_a_rendre = montant_recu - prix
    print("Je vous rends", montant_a_rendre)
print("Voici vos", quantité, ("peket(s)" if choix == 1 else "Coca"))
```

Ln: 10 Col: 46

# Exemple

## Les fêtes de Wallonie !

Gérer une commande de boissons (en très simplifié).



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
===== RESTART: D:/TIR1/TI101 Intro. à la programmation/wallonie.py =====
Quelle boisson désirez-vous ?
Les boissons disponibles sont
1. Peket
2. Coca
Votre choix (1 ou 2) : 1
Nombre de boissons : 5
Montant reçu : 20
Je vous rends 10.0
Voici vos 5 peket(s)
>>>
Ln: 59 Col: 0
```



# Qu'est-ce que la programmation ?

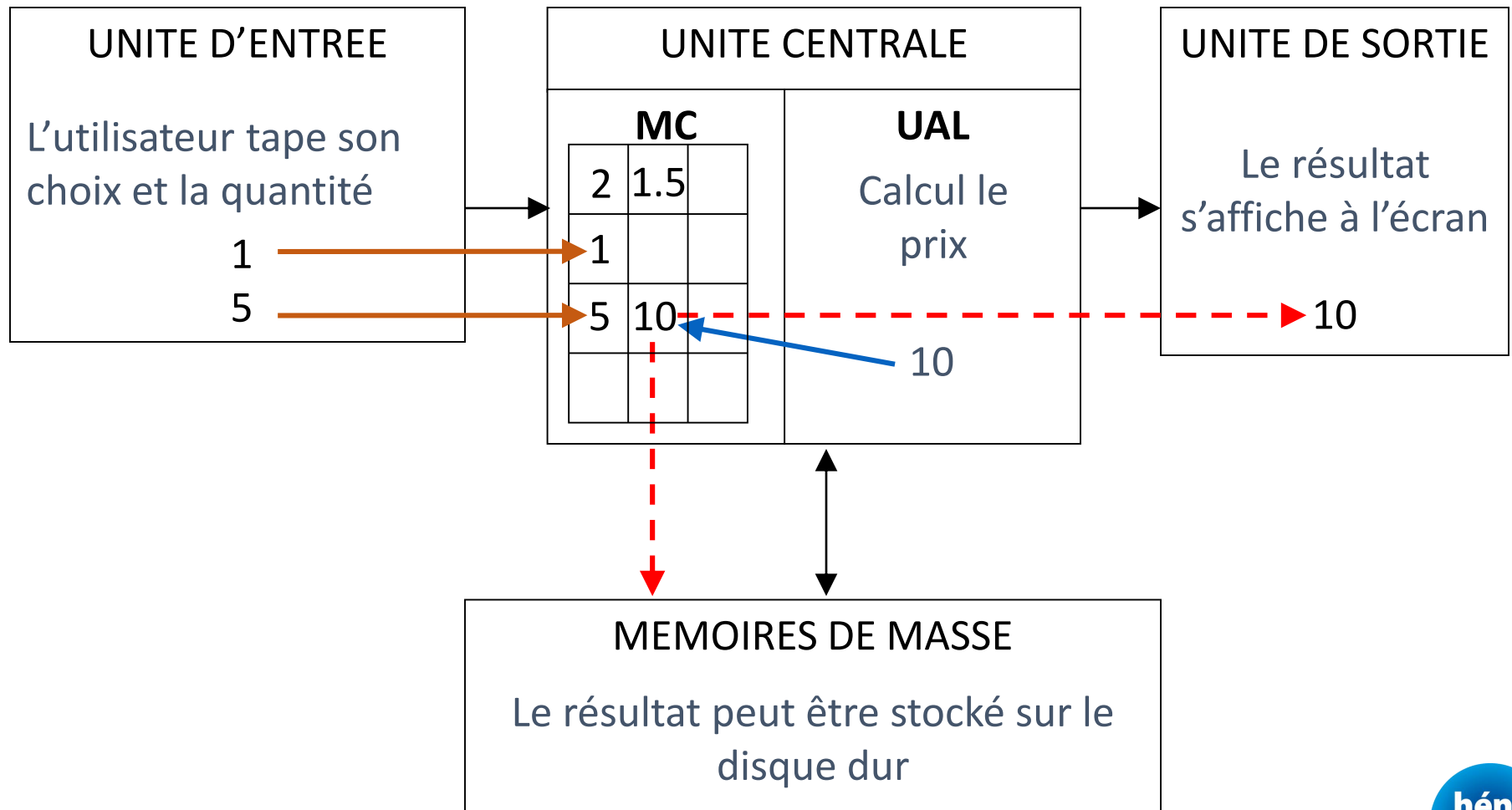
Concevoir et écrire des programmes... ou des scripts.

La notion de programmation englobe le fait de concevoir et de programmer !

D'où les consignes d'évaluations exposées ci-avant :

- les programmes doivent être **propres**,
- les programmes doivent être **efficaces**,
- vous devez **comprendre** pourquoi ils tournent !

# Comment le programme est-il exécuté ?



# COMPILER OU INTERPRÉTER

- Deux façons de traduire **UN** code source en langage machine :

- Compilation



- Interprétation



- Python : mélange les deux... comme C#, Java et Javascript...



# COMPILETION

Le compilateur traduit toutes les instructions en code machine.

Ensuite le code généré peut être exécuté.



Le programme peut être exécuté autant de fois que nécessaire sans devoir le recompiler.

On peut même (si le code est "portable") l'exécuter sur des machines différentes...

# INTERPRÉTATION

L'interpréteur traduit les instructions les unes après les autres en les exécutant directement.



À chaque fois que vous devez exécuter le programme, il doit être réinterprété.

# INTERPRÉTÉ VS COMPILÉ

## Interprété

- Avantages
  - Facilité de tests (à la volée)
  - Les modifications sont prises en compte sans devoir recompiler
- Inconvénients
  - Lenteur à l'exécution...
  - Détection des erreurs seulement à l'exécution du programme
  - Le code source est nécessaire pour exécuter le programme

## Compilé

- Avantages
  - Rapidité d'exécution
  - Détection des erreurs à la compilation
  - L'exécutable est nécessaire pour exécuter le programme (pas le code source)
- Inconvénients
  - Il faut recompiler à chaque modification

# ÉVOLUTION ET NORMES

- **1989** Invention du Python (Monty Python) par Guido van Rossum sur base du langage ABC (utilisé au CWI)
- ...
- **2000** Création de BeOpen.com et PythonLabs → 2.0
- **2001** Création de la Python Software Foundation → 2.1
- ...
- **2008** Python 3.0
- **2010** Dernière version de la branche Python 2.x → 2.7
- ...
- **2019** **Python 3.7.4**

# IMPLÉMENTATIONS

Il existe différentes implémentations de Python

- **CPython** implémentation de référence, basée sur le C (CPythonVM)
- **Jython** implémentation permettant de générer du code objet Java (JVM)
- **ronPython** implémentation pour Microsoft .NET permettant de le combiner à du C# (CLR/.Net)
- **PyPy** implémentation en Python
- **RubyPython** implémentation en Ruby (RubyVM)
- **Brython** implémentation permettant de le combiner à du Javascript (Javascript Interpreter V8)



# DE LA BONNE UTILISATION DU WEB

Il est fréquent de consulter divers sites pour trouver de l'inspiration... Quelques conseils de bonne pratique sont donc nécessaires :

- Veillez au choix des sources.
- Validez l'information en la croisant avec celle d'autres sites.
- Gardez toujours un esprit critique.

[https://www.lemonde.fr/campus/article/2016/04/26/conseils-pour-faire-une-recherche-sur-internet\\_4909157\\_4401467.html](https://www.lemonde.fr/campus/article/2016/04/26/conseils-pour-faire-une-recherche-sur-internet_4909157_4401467.html)

Il existe d'autres moteurs de recherches que Google... par exemple : <https://www.diigo.com/list/urfistrennes/moteurs-scientifiques>

# RÉFÉRENCES

Le tutoriel Python

<https://docs.python.org/fr/3/tutorial/index.html>

PEP 8 -- Style Guide for Python Code

<https://www.python.org/dev/peps/pep-0008/>

...