

Introduction à la programmation : cours 1

A la découverte d'un nouveau monde !

Benoît Frénay - Faculté d'Informatique



UNIVERSITÉ
DE NAMUR









**KEEP
CALM
AND
HAVE
FUN**

Quelques précisions à propos du cours

- apprendre à concevoir des programmes pour résoudre des problèmes
- approche universelle : tout n'est pas en cours de Python

toutes les questions sont les bienvenues (et encore n'est simple)

- interrompez-moi tout moment pour une question
- je vous encourage à demander tout ce que vous voulez
- je suis là pour vous aider à apprendre

Quelques précisions à propos du cours

- ♦ apprendre à concevoir des programmes pour résoudre des problèmes
- ♦ approche universitaire : ceci n'est pas un cours de Python

Toutes les questions sont les bienvenues (et aucune n'est stupide) :

- interrompez-moi dès que vous avez une question
- si vous débutez, demandez-moi / je vous explique
- le but est que vous appreniez à programmer

Quelques précisions à propos du cours

- ♦ apprendre à concevoir des programmes pour résoudre des problèmes
- approche universitaire : ceci n'est pas un cours de Python

toutes les questions sont les bienvenues (et aucune n'est stupide)

- ♦ **interrompez-moi** dès que vous avez une question
- si vous décrochez, **demandez-moi** de (ré)expliquer
- ♦ le but est que **vous** appreniez à programmer

Quelques précisions à propos du cours



Plan du cours

- ✧ un peu d'histoire
- ✧ la programmation est un jeu d'enfant
- ✧ exemple de système programmé
- ✧ notions de base en programmation
- ✧ outils utilisés par le cours
- ✧ organisation du cours
- ✧ quelques références

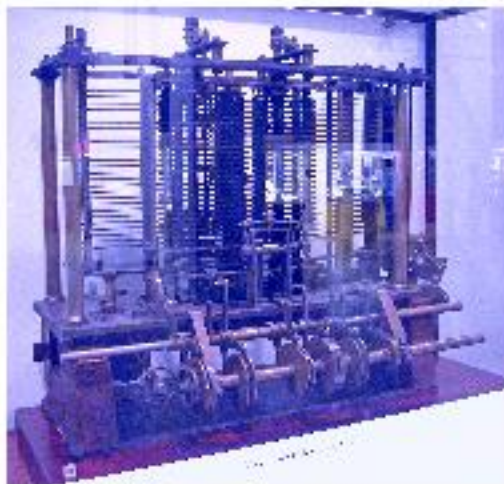
Un peu d'histoire

L'algorithmique : une science vieille comme le monde

- ♦ vient du nom du mathématicien perse Al-Khwarizmi (780–850)
- ♦ premiers algorithmes remontent à Babylone (+ de 4000 ans)
- ♦ algorithme PGCD d'Euclide dans ses Éléments (300 av. J.-C.)



Avant l'ordinateur électronique



Alan Turing (1912-54) : les bases théoriques

- ♦ pose les bases mathématiques de l'informatique en 1936
- ♦ conçoit une "bombe" capable de cracker le code d'Enigma (84.000 messages/mois) et permet de forcer le blocus des U-boats allemands (raccourci la guerre de 2 à 4 ans et sauve de 14 à 21 millions de vies)
- ♦ propose le test de Turing (= "imitation game") en 1950



John von Neumann (1903–57) : le touche-à-tout

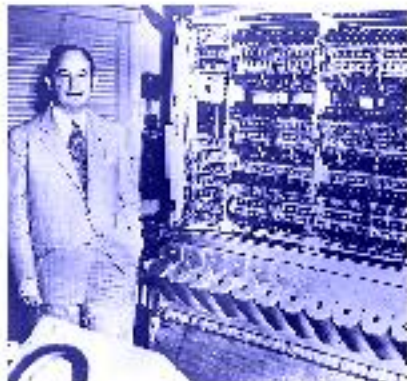
- mathématicien et physicien ayant participé à la création de l'EDVAC (5,5 ko, 1.160 additions/seconde, 6.000 tubes à vides, 12.000 diodes, 56 kW, 45,5 m², 7.850 kg, \$500.000, équipes de trente personnes)
- a proposé l'architecture Von Neumann à la base des ordinateurs actuels



ENIAC, EDVAC, Mark I : les premiers ordinateurs

"I think there is a world market for maybe five computers."

– Thomas Watson, chairman of IBM, 1943.



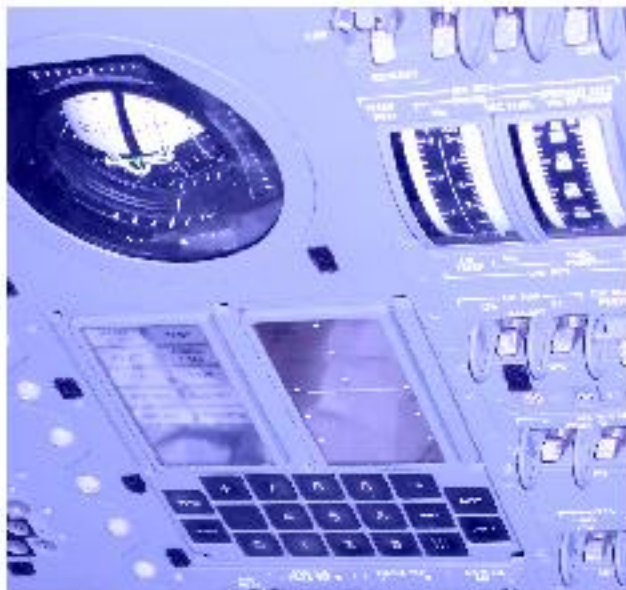
IBM 704 computer at NASA in 1957



Dual IBM 7090s at NASA Mission Control in 1962



Apollo Guidance Computer (64 ko memory, 0.043MHz)



Deep Blue (1996) vs. Samsung Galaxy S5 (2014)



- 1960's : premier compilateur et département d'informatique, ARPAnet
- 1970's : premier microprocesseur et micro-ordinateurs, Unix, CRAY-I
- 1980's : explosion des ordinateurs personnels (5 millions en 1982)
- 1990's : Internet, Windows 3.1, e-mails, HTML 4.0, MP3, JPEG
- 2000's : eBay, Facebook, YouTube, Google, Mozilla, Linux (Android)
- 2010's : smartphones, internet of things, big data, machine learning



La programmation est
un jeu d'enfant



Définition

un programme informatique est une suite d'instructions qui décrivent de façon claire et non-ambigüe à un ordinateur comment effectuer une tâche

- écrit dans un langage spécifique (= langage de programmation)
- constitué de variables, d'instructions, de boucles, de fonctions, etc.
- programmer = écrire un programme pour résoudre un problème

l'exécution d'un programme permet de résoudre le problème pour lequel il a été conçu = exécution = effectuer la suite d'instructions (peut varier)

Définition

un programme informatique est une suite d'instructions qui décrivent de façon claire et non-ambigüe à un ordinateur comment effectuer une tâche

- ↗ écrit dans un langage spécifique (= langage de programmation)
- ↗ constitué de variables, d'instructions, de boucles, de fonctions, etc.
- ↗ programmer = écrire un programme pour résoudre un problème

L'exécution d'un programme permet de résoudre le problème pour lequel il est conçu ⇒ exécution = effectuer la suite d'instructions (peut varier)

Définition

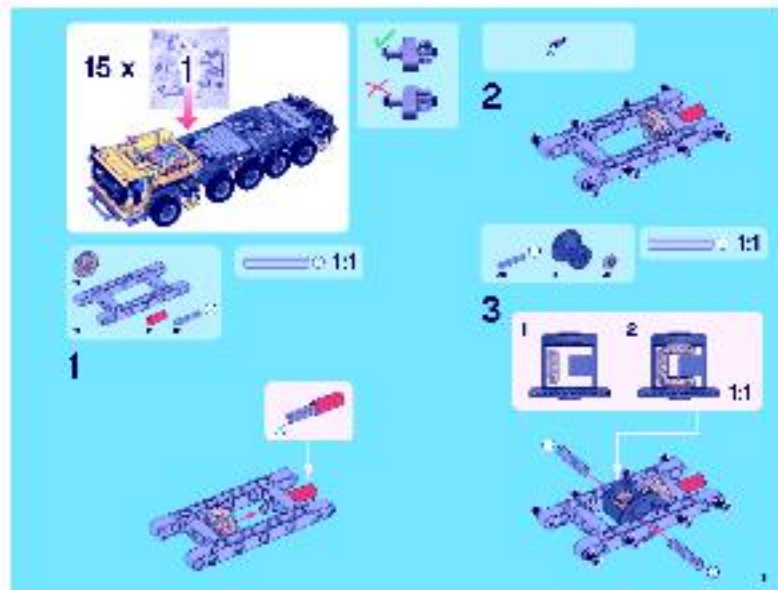
un programme informatique est une suite d'instructions qui décrivent de façon claire et non-ambigüe à un ordinateur comment effectuer une tâche

- ↗ écrit dans un langage spécifique (= langage de programmation)
- ↗ constitué de variables, d'instructions, de boucles, de fonctions, etc.
- ↗ programmer = écrire un programme pour résoudre un problème

l'exécution d'un programme permet de résoudre le problème pour lequel il est conçu \Rightarrow exécution = effectuer la suite d'instructions (peut varier)

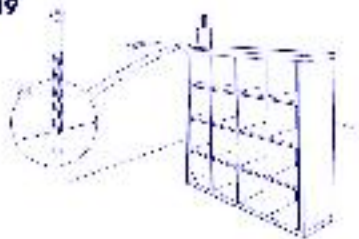
Exemples de "programme" que vous avez un jour exécuté



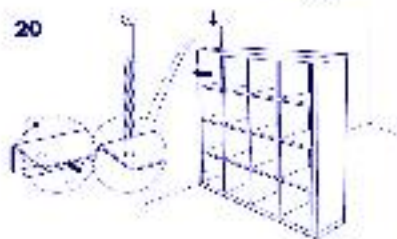


Exemples de "programme" que vous avez un jour exécuté

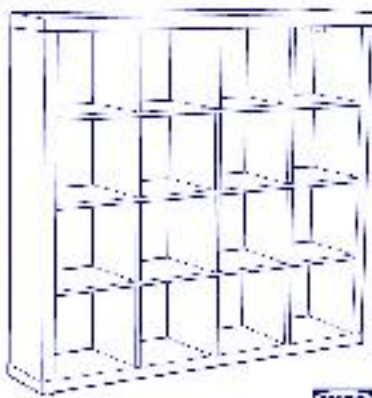
19



20



EXPEDIT



Exemples de "programme" que vous avez un jour exécuté



Comment "cuisiner" avec un ordinateur ?

Le besoin

permet de "stocker" une recette pour le manipuler

- lister les personnes impliquées dans le besoin
- définir les aspects du besoin qui vont avec achats

modifier le comportement du programme pour l'adapter aux besoins

Les recettes

décrivait le programme = "que faire à quel moment ?"

- équivaut à la liste des étapes dans une recette
- écrites dans un langage particulier (pas le français !)
- programme informatique = suite d'instructions

Comment "cuisiner" avec un ordinateur ?

Variable

permet de "stocker" une valeur pour la manipuler

- ↪ nombre de personnes invitées pour le repas
- ↪ poids du paquet de beurre que vous avez acheté

modifie le comportement du programme pour l'adapter aux besoins

Algorithme

décrit le programme = "que faire à quel moment ?"

- ↪ équivaut à la liste des étapes dans une recette
- ↪ écrites dans un langage particulier (pas le français !)
- ↪ programme informatique = suite d'instructions

Comment "cuisiner" avec un ordinateur ?

Variable

permet de "stocker" une valeur pour la manipuler

- nombre de personnes invitées pour le repas
- poids du paquet de beurre que vous avez acheté

modifie le comportement du programme pour l'adapter aux besoins

Instructions

décrivent le programme = "que faire à quel moment ?"

- équivalent à la suite des étapes dans une recette
- écrites dans un langage particulier (pas le français!)
- programme informatique = suite d'instructions

Comment "cuisiner" avec un ordinateur ?

Instructions conditionnelles

permet de choisir les instructions en fonction d'une condition

- exemple : "si je veux du steak à dîner, alors achète 5", alors "achète 5"
- permet une structure hiérarchique de programmes ("enfonçements")

Boucles

regroupe une suite d'instructions souvent répétées ensemble

- permet de former des blocs réutilisables = "bouts à suite"
- exemples : `while`, `do while`, `for`, `do for`
- rend les programmes beaucoup plus simples à écrire, lire, corriger

spécification = contrat d'utilisation ("que lui fais-tu et que fait-elle?")

Comment "cuisiner" avec un ordinateur ?

Instructions conditionnelles

permet de choisir les instructions en fonction d'une condition

- exemple : "si je veux un steak à point, alors cuire 8', sinon cuire 5'"
- permet une structure non-linéaire de programmes ("embranchements")

regroupe une suite d'instructions souvent utilisées ensemble

➤ permet de former des blocs réutilisables = "bouts à suite"

➤ exemples : pelier() ; 5, saisir() ; 3, saisir() ; 2

➤ rend les programmes beaucoup plus simples à écrire, lire, corriger

spécification = contrat d'utilisation ("que lui fais-il et que fait-elle?")

Comment "cuisiner" avec un ordinateur ?

Instructions conditionnelles

permet de choisir les instructions en fonction d'une condition

- exemple : "si je veux un steak à point, alors cuire 8', sinon cuire 5'"
- permet une structure non-linéaire de programmes ("embranchements")

Fonction

regroupe une suite d'instructions souvent utilisées ensemble

- permet de former des blocs réutilisables \Rightarrow "boîte à outils"
- exemples : `peler(...)`, `cuire(...)`, `mélanger(..., ...)`
- rend les programmes beaucoup plus simples à écrire, lire, corriger

spécification = contrat d'utilisation ("que lui faut-il et que fait-elle ?")

Comment "cuisiner" avec un ordinateur ?

Loops et répétition

permet d'exécuter des instructions de manière répétitive

- exemple : "répéter n fois : verser de la pâte et cuire une crêpe"
- permet de choisir à l'exécution le nombre de répétitions
- une boucle peut aussi dépendre d'une condition d'arrêt
- exemple : "tant qu'il y a de la pâte, en verser et cuire une crêpe"

Structures de données et listes

permet de représenter l'information sous forme efficace

- un célèbre : les recettes sont mises dans un classant (type, titre, etc.)
- associées à des algorithmes spécifiques (recherche, tri, etc.)

Comment "cuisiner" avec un ordinateur ?

Boucle et invariant

permet d'exécuter des instructions de manière répétitive

- exemple : "répéter n fois : verser de la pâte et cuire une crêpe"
- permet de choisir à l'exécution le nombre de répétitions
- une boucle peut aussi dépendre d'une condition d'arrêt
- exemple : "tant qu'il y a de la pâte, en verser et cuire une crêpe"

Structures de données abstraites

permet de représenter l'information sous forme efficace

- un ensemble de requêtes sont mises dans un tableau (type, nom, etc.)
- associées à des algorithmes spécifiques (recherche, tri, etc.)

Comment "cuisiner" avec un ordinateur ?

Boucle et invariant

permet d'exécuter des instructions de manière répétitive

- ◆ exemple : "répéter n fois : verser de la pâte et cuire une crêpe"
- permet de choisir à l'exécution le nombre de répétitions
- ◆ une boucle peut aussi dépendre d'une condition d'arrêt
- exemple : "tant qu'il y a de la pâte, en verser et cuire une crêpe"

Structures de données et fichiers

permet de représenter l'information sous forme efficace

- ◆ en cuisine, les recettes sont mises dans un classeur (type, nom, etc.)
- associées à des algorithmes spécifiques (manipulation, tri, etc.)

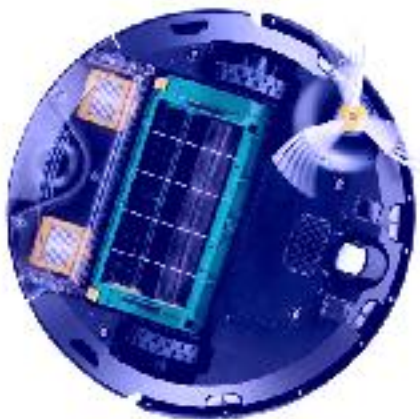
Définition

un programme informatique est une suite d'instructions qui décrivent de façon claire et non-ambigüe à un ordinateur comment effectuer une tâche

Quelques-unes des notions vues au cours

- instructions
- variables
- instructions conditionnelles
- fonctions
- boucle et invariant
- structures de données et fichiers

Exemple de système programmé



iRobot Roomba 560

The Roomba uses a variety of cleaning behaviors to cover a room, using input from its sensors to decide where to go next.

It doesn't know where it has or has not been in the absolute sense, but on average, it will cover each area of a room 3-4 times.



The Roomba uses a variety of cleaning behaviors to cover a room, using input from its sensors to decide where to go next.

It doesn't know where it has or has not been in the absolute sense, but on average, it will cover each area of a room 3-4 times.



The Roomba uses a variety of cleaning behaviors to cover a room, using input from its sensors to decide where to go next.

It doesn't know where it has or has not been in the absolute sense, but on average, it will cover each area of a room 3-4 times.



The Roomba uses a variety of cleaning behaviors to cover a room, using input from its sensors to decide where to go next.

It doesn't know where it has or has not been in the absolute sense, but on average, it will cover each area of a room 3-4 times.





Dans la peau d'un Roomba avec une GoPro

- ◆ <https://www.youtube.com/watch?v=xX9RkWsX1KE>
- <https://www.youtube.com/watch?v=pcw79-onqRw>

en savoir plus sur le iRobot Roomba : <http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum.htm>



- INFOM450 : Internet des objets
- INFOM451 : conception d'applications mobiles
- INFOM453 : laboratoire en informatique ambiante et mobile

Notions de base en programmation

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

éléments d'un langage

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

Éléments d'un langage

- syntaxe : vocabulaire (mots disponibles pour "parler" à l'ordinateur) et grammaire (comment ils peuvent s'enchaîner pour former une phrase)
- sémantique : signification de toute "phrase" syntaxiquement correcte
- pragmatique : à quoi servent les différentes concepts et constructions et surtout comment les utiliser pour résoudre une tâche complexe

attention : l'ordinateur est profondément stupide et psychorigide !

- <https://grisebouille.net/les-ordinateurs-sont-cons>

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

Éléments d'un langage

- ♦ syntaxe : vocabulaire (mots disponibles pour "parler" à l'ordinateur) et grammaire (comment ils peuvent s'enchaîner pour former une phrase)

• sémantique : signification de toute "phrase" syntaxiquement correcte

• pragmatique : à quoi servent les différentes concepts et constructions et surtout comment les utiliser pour résoudre une tâche complexe

attention : l'ordinateur est profondément stupide et psychorigide !

• <https://grisebouille.net/les-ordinateurs-sont-cons>

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

Éléments d'un langage

- ♦ syntaxe : vocabulaire (mots disponibles pour "parler" à l'ordinateur) et grammaire (comment ils peuvent s'enchaîner pour former une phrase)
- ♦ sémantique : signification de toute "phrase" syntaxiquement correcte
- pragmatique : à quoi servent les différentes concepts et constructions et surtout comment les utiliser pour résoudre une tâche complexe

attention : l'ordinateur est profondément stupide et psychorigide !

• <https://grisebouille.net/les-ordinateurs-sont-conn>

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

Éléments d'un langage

- ♦ syntaxe : vocabulaire (mots disponibles pour "parler" à l'ordinateur) et grammaire (comment ils peuvent s'enchaîner pour former une phrase)
- ♦ sémantique : signification de toute "phrase" syntaxiquement correcte
- ♦ pragmatique : à quoi servent les différents concepts et constructions et surtout comment les utiliser pour résoudre une tâche complexe

attention : l'ordinateur est profondément stupide et psychorigide !

→ <https://grisebouille.net/les-ordinateurs-sont-conn>

Définition

un langage de programmation est un langage formel et exécutable par un ordinateur permettant d'expliquer comment effectuer une tâche

Éléments d'un langage

- ♦ syntaxe : vocabulaire (mots disponibles pour "parler" à l'ordinateur) et grammaire (comment ils peuvent s'enchaîner pour former une phrase)
- ♦ sémantique : signification de toute "phrase" syntaxiquement correcte
- ♦ pragmatique : à quoi servent les différents concepts et constructions et surtout comment les utiliser pour résoudre une tâche complexe

attention : l'ordinateur est profondément stupide et psychorigide !

- ♦ <https://grisebouille.net/les-ordinateurs-sont-cons>

Erreurs de programmation

Erreurs d'orthographe

fautes d'orthographe ou de grammaire

- mot qui n'existe pas ou mal orthographié
- mots combinés d'une façon qui n'est pas autorisée

Erreurs d'exécution

syntaxe correcte, mais impossible à exécuter dans certaines situations

- exemple : prendre n objets dans le frigo, alors qu'il est vide
- une erreur d'exécution a pour effet l'arrêt du programme

Erreurs d'exécution, d'orthographe, de terminaison

s'exécute normalement, mais ne produit pas le résultat escompté

- très difficile à détecter puisque le programme se termine
- exemple : faire un cake sans farine ni œufs

Erreurs syntaxiques

fautes d'orthographe ou de grammaire

- ♦ mot qui n'existe pas ou mal orthographié
- ♦ mots combinés d'une façon qui n'est pas autorisée

Erreurs d'exécution

syntaxe correcte, mais impossible à exécuter dans certaines situations

- ♦ exemple : prendre n oeufs dans le frigo, alors qu'il est vide
- ♦ une erreur d'exécution a pour effet l'arrêt du programme

Erreurs d'intention / des logiques de programmation

s'exécute normalement, mais ne produit pas le résultat escompté

- ♦ très difficile à détecter puisque le programme se termine
- ♦ exemple : faire un cake sans farine ni oeufs.

Erreurs de programmation

Erreurs syntaxiques

fautes d'orthographe ou de grammaire

- ♦ mot qui n'existe pas ou mal orthographié
- ♦ mots combinés d'une façon qui n'est pas autorisée

Erreurs d'exécution

syntaxe correcte, mais impossible à exécuter dans certaines situations

- ♦ exemple : prendre n oeufs dans le frigo, alors qu'il est vide
- ♦ une erreur d'exécution a pour effet l'arrêt du programme

Erreurs d'exécution / Erreurs de logique

s'exécute normalement, mais ne produit pas le résultat escompté

- ♦ très difficile à détecter: puisque le programme se termine
- ♦ exemple : faire un cake sans farine ni œufs.

Erreurs de programmation

Erreurs syntaxiques

fautes d'orthographe ou de grammaire

- ❖ mot qui n'existe pas ou mal orthographié
- ❖ mots combinés d'une façon qui n'est pas autorisée

Erreurs d'exécution

syntaxe correcte, mais impossible à exécuter dans certaines situations

- ❖ exemple : prendre n oeufs dans le frigo, alors qu'il est vide
- ❖ une erreur d'exécution a pour effet l'arrêt du programme

Erreurs d'intention / de logique / de sémantique

s'exécute normalement, mais ne produit pas le résultat escompté

- ❖ très difficile à détecter, puisque le programme se termine
- ❖ exemple : faire un cake sans farine ni oeufs

But du cours

apprendre à concevoir des programmes pour résoudre des problèmes

Qu'est-ce que programmer ?

- apprendre à programmer, c'est surtout apprendre la pragmatique
 - suivre la démarche à suivre de façon rigoureuse et méthodique
 - indépendamment du langage utilisé (ou grande partie)

La syntaxe et la sémantique seront découvertes progressivement...

But du cours

apprendre à concevoir des programmes pour résoudre des problèmes

Approche universitaire

apprendre à programmer, c'est surtout apprendre la pragmatique

- ♦ décrire la démarche à suivre de façon rigoureuse et non ambiguë
- ♦ indépendamment du langage utilisé (en grande partie)

la syntaxe et la sémantique seront découverts progressivement...

Étapes de résolution d'une tâche

1. Analyse du problème

- **motus-codes** = énoncé initial français/anglais et symboles
- **code** = instructions (non ambiguës) dans un langage informatique

2. Algorithmique et programmation

- **algorithme** = solution au problème en pseudo-code
- **programme(x)** = implémentation(x) dans un langage spécifique

3. Validation d'un programme

- analyse de la tâche et écriture d'un algorithme la résolvant
- traduction de l'algorithme en langage de programmation
- validation et test du programme obtenu

Étapes de résolution d'une tâche

Code vs. pseudo-code

- pseudo-code = ébauche mêlant français/anglais et symboles
- code = instructions (non ambiguës) dans un langage informatique

Différence entre un algorithme et un programme

- algorithme = solution au problème en pseudo-code
- programme(x) = implémentation(x) dans un langage spécifique

Étapes de la programmation

- analyse de la tâche et écriture d'un algorithme la résolvant
- traduction de l'algorithme en langage de programmation
- validation et test du programme obtenu

Étapes de résolution d'une tâche

Code vs. pseudo-code

- pseudo-code = ébauche mêlant français/anglais et symboles
- code = instructions (non ambiguës) dans un langage informatique

Algorithme vs. programme

- algorithme = solution au problème en pseudo-code
- programme(s) = implémentation(s) dans un langage spécifique

Processus de programmation

- analyse de la tâche et écriture d'un algorithme la résolvant
- traduction de l'algorithme en langage de programmation
- vérification et test du programme obtenu

Étapes de résolution d'une tâche

Code vs. pseudo-code

- pseudo-code = ébauche mêlant français/anglais et symboles
- code = instructions (non ambiguës) dans un langage informatique

Algorithme vs. programme

- algorithme = solution au problème en pseudo-code
- programme(s) = implémentation(s) dans un langage spécifique

Conception d'un programme

- analyse de la tâche et écriture d'un algorithme la résolvant
- traduction de l'algorithme en langage de programmation
- vérification et test du programme obtenu

Pseudo-code

afficher le message "Hello, World !" à l'écran

Implémentation (code) en langage machine

```
10111010 00010000
00000001 10110100
00001001 11001101
00100001 00110000
11100100 11001101
00010110 10111000
00000000 01001100
11001101 00100001
01001000 01100101 (he)
01101100 01101100 (ll)
01101111 00100000 (o )
01010111 01101111 (wo)
01110010 01101100 (rl)
01100100 00100001 (d!)
00100100 ($)
```

Implémentation (code) en assembleur

```
section .data
    helloMsg:    db 'Hello world!',10
    helloSize:   equ $-helloMsg

section .text
    global _start
_start:
    mov eax,4           ; Appel système "write" (sys_write)
    mov ebx,1           ; File descriptor, 1 pour STDOUT
    mov ecx, helloMsg   ; Adresse de la chaîne à afficher
    mov edx, helloSize  ; Taille de la chaîne
    int 80h             ; Exécution de l'appel système
                       ; Sortie du programme

    mov eax,1           ; Appel système "exit"
    mov ebx,0           ; Code de retour
    int 80h
```

Implémentation (code) en C

```
#include <stdio.h>

int main(void)
{
    printf("hello , world\n");
    return 0;
}
```

source : http://fr.wikipedia.org/wiki/Liste_de_programmes>Hello_world

Implémentation (code) en Python

```
print("Hello world!")
```

source : http://fr.wikipedia.org/wiki/Liste_de_programmes>Hello_world

Outils utilisés par le cours



- site web du cours (INFOB131 "Introduction à la programmation")
- tout fichier sur WebCampus peut être considéré comme définitif
 - transparents PDF (= "slides") disponibles en plusieurs formats
 - imprimer Intro_prog_X-handout_notes.pdf (à savoir, files-la !)
 - exemples TFA, liens web, documents supplémentaires, etc.
- communication : annonces, forums (+ e-mails)
- outils standard pour les cours : vous allez l'utiliser pendant 5 ans !



- **site web du cours INFOB131 "Introduction à la programmation"**
- tout fichier sur WebCampus peut être considéré comme définitif
 - transparents PDF (= "slides") disponibles en plusieurs formats
 - imprimer `Intro_prog_X-handout_notes.pdf` (à associer, à télécharger)
 - exemples TFA, liens web, documents supplémentaires, etc.
- communication : annonces, forums (+ e-mails)
- outils standard pour les cours : vous allez l'utiliser pendant 5 ans !



- site web du cours INFOB131 "Introduction à la programmation"
- tout fichier sur WebCampus peut être considéré comme définitif!
 - transparents PDF (= "slides") disponibles en plusieurs formats
 - imprimez `intro_prog_X_handout_notes.pdf` (si soucis, dites-le !)
 - énoncés TP, liens web, documents supplémentaires, etc.
- communication : annonces, forums (+ e-mails)
- outils standard pour les cours : vous allez l'utiliser pendant 5 ans !



- site web du cours INFOB131 "Introduction à la programmation"
- tout fichier sur WebCampus peut être considéré comme définitif!
 - transparents PDF (= "slides") disponibles en plusieurs formats
 - imprimez `intro_prog_X_handout_notes.pdf` (si soucis, dites-le !)
 - énoncés TP, liens web, documents supplémentaires, etc.
- communication : annonces, forums (+ e-mails)
- outils standard pour les cours : vous allez l'utiliser pendant 5 ans !

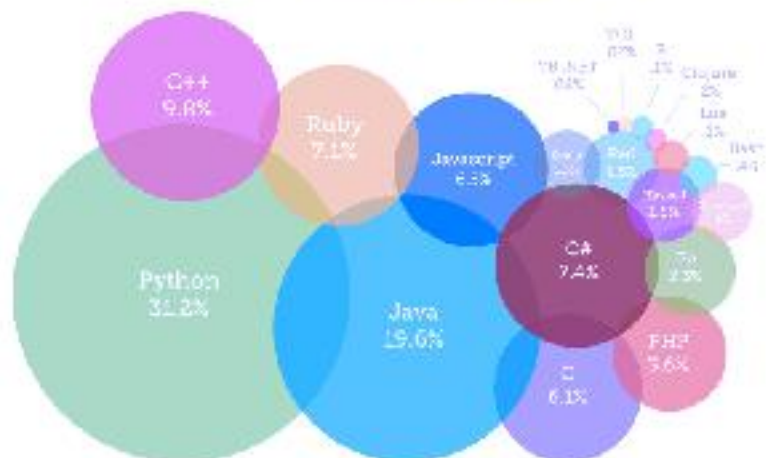


- site web du cours INFOB131 "Introduction à la programmation"
- tout fichier sur WebCampus peut être considéré comme définitif!
 - transparents PDF (= "slides") disponibles en plusieurs formats
 - imprimez `intro_prog_X_handout_notes.pdf` (si soucis, dites-le !)
 - énoncés TP, liens web, documents supplémentaires, etc.
- communication : annonces, forums (+ e-mails)
- outils standard pour les cours : vous allez l'utiliser pendant 5 ans !

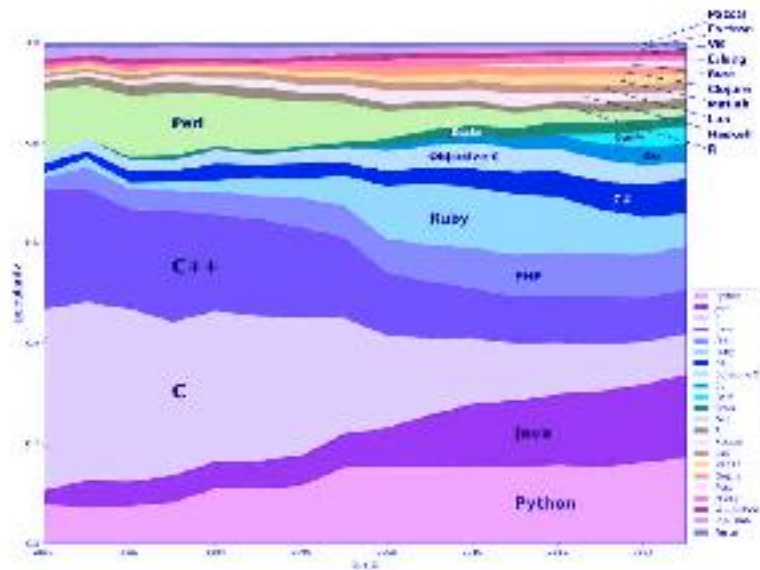


Pourquoi Python ?

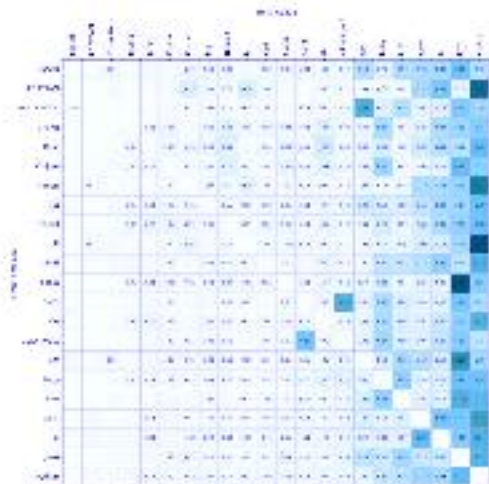
Most Popular Coding Languages of 2015



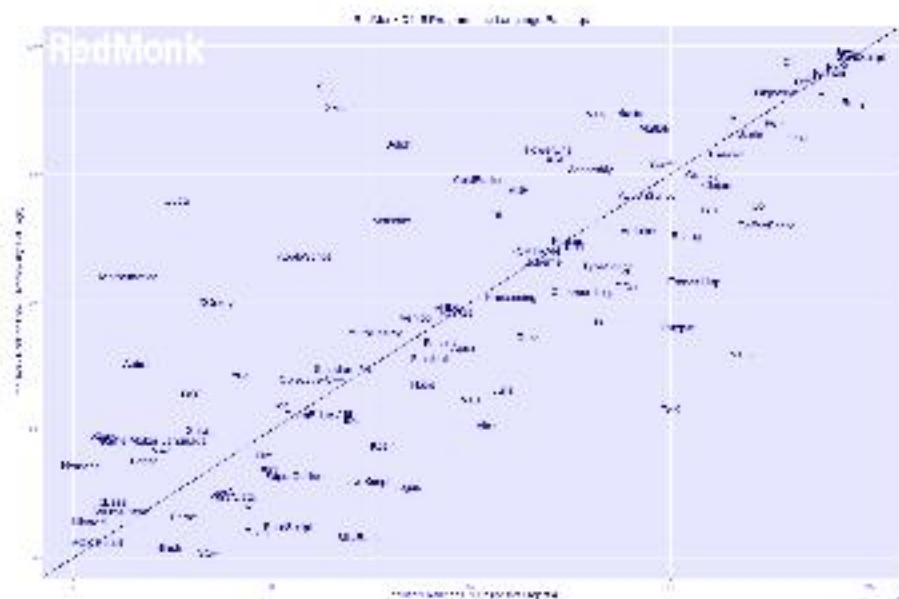
Pourquoi Python ?



Pourquoi Python ?



Pourquoi Python?



Pourquoi Python ?

Avantages

- ✓ (premier) langage simple à prendre en main
- ✓ lisibilité du code (par ex. indentation obligatoire)
- ✓ langage généraliste et de haut niveau (peu de code % à C, Java etc.)
- ✓ possibilité d'exécuter des bouts de code (+ notebooks) + scalable
- ✓ libre, gratuit, open source, disponible sur Windows, Mac OS, Linux
- ✓ communauté active (bibliothèques, documentation, tutoriaux, etc.)
- ✓ structure de données disponibles immédiatement
- ✓ système de modules hiérarchique et lisible

Inconvénients

- plus lent à l'exécution (mais Cython peut aider)
- typage dynamique (erreurs difficiles à détecter)

Pourquoi Python ?

Avantages

- ✓ (premier) langage simple à prendre en main
- ✓ lisibilité du code (par ex. indentation obligatoire)
- ✓ langage généraliste et de haut niveau (peu de code % à C, Java etc.)
- ✓ possibilité d'exécuter des bouts de code (+ notebooks) + scalable
- ✓ libre, gratuit, open source, disponible sur Windows, Mac OS, Linux
- ✓ communauté active (bibliothèques, documentation, tutoriaux, etc.)
- ✓ structure de données disponibles immédiatement
- ✓ système de modules hiérarchique et lisible

Inconvénients

- × plus lent à l'exécution (mais Cython peut aider)
- × typage dynamique (erreurs difficiles à détecter)

Pourquoi Anaconda + PyCharm ?

Pourquoi Python 3 et Anaconda ?

- tous les outils en un seul téléchargement
- look and feel identique sur tous les OS
- centaines de modules disponibles immédiatement
- gestionnaire Jupyter et notebooks intégrés
- disponible pour Python 2.7 et 3.x

À propos des alternatives

- Python est (et restera) libre et gratuit
- Anaconda + PyCharm vous aide à gérer toute les outils
- dans le cadre de ce cours, n'utilisez que PyCharm

Pourquoi Anaconda + PyCharm ?

Distribution Python "self-contained" et gratuite

- tous les outils en un seul téléchargement
- look and feel identique sur tous les OS
- centaines de modules disponibles immédiatement
- gestionnaire, IDE et notebooks intégrés
- disponible pour Python 2.7 et 3.x

À propos des licences

- Python est (et restera) libre et gratuit
- Anaconda + PyCharm nous aide à gérer tous les outils
- dans le cadre de ce cours, n'utilisez que PyCharm

Pourquoi Anaconda + PyCharm ?

Distribution Python "self-contained" et gratuite

- tous les outils en un seul téléchargement
- look and feel identique sur tous les OS
- centaines de modules disponibles immédiatement
- gestionnaire, IDE et notebooks intégrés
- disponible pour Python 2.7 et 3.x

A propos des alternatives

- Python est (et restera) libre et gratuit
- Anaconda + PyCharm vous aide à gérer tous les outils
- dans le cadre de ce cours, n'utilisez que PyCharm



instructions disponible sur le site WebCampus

- n'hésitez pas à demander de l'aide aux assistants
- une install party est prévue très bientôt (détails sur WebCampus)

disponible sur les machines du pool informatique

- réalisé par Fishing Cactus (Mons, Belgique - 1,8 millions DL Shift)
- présenté à la conférence EDEN à Oslo (Norvège) en 2013
- "Meilleur Serious Game" au Serious Game Expo de Lyon 2013
- "Best Learning Game, 2nd" aux 2013 European Serious Game awards



Présentation du système de notebooks

IPYNB Notebook

spectrogram.py 2017-06-06 14:07:14-0400

File Edit View Insert Cell Help Window Help

Python 3.6.1 IPython 5.1.0 JupyterLab 0.34.0

Simple spectral analysis

An audio file is loaded into the notebook

$$A_n = \sum_{k=0}^{N-1} a_k e^{-j\frac{2\pi}{N}kn} \quad n = 0, \dots, N-1$$

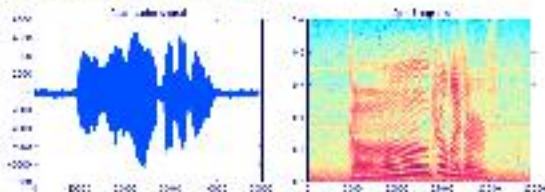
Logarithmic frequency resolution is implemented using a log scale

Complexity is linear, and the memory footprint is linear as well

```
In [1]: from math import log, exp, pi, sqrt
import numpy as np
import matplotlib.pyplot as plt
```

Zero-centered, normalized, and windowed signal is computed using spectrogram

```
In [2]: from math import log, exp, pi, sqrt
import numpy as np
import matplotlib.pyplot as plt
from spectrogram import spectrogram
```

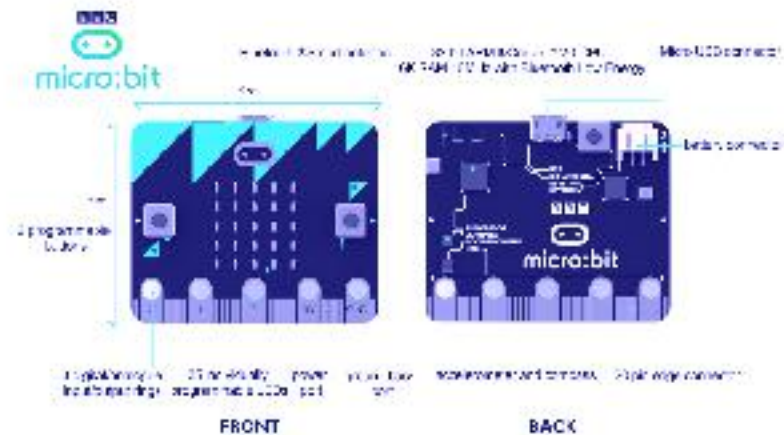


Nouveauté de cette année : le projet micro:PUNCH



Nouveauté de cette année : le projet micro:PUNCH

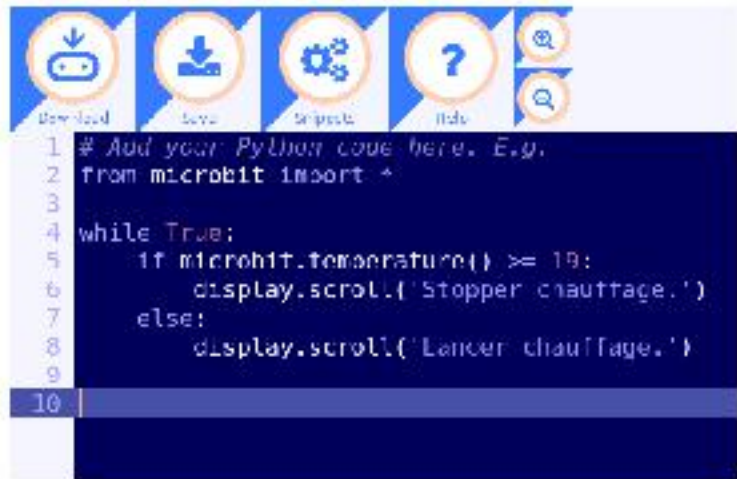




Nouveauté de cette année : le projet micro:PUNCH



Nouveauté de cette année : le projet micro:PUNCH





Nouveauté de cette année : le projet micro:PUNCH

Ressources

micro:bit est un projet de la BBC qui offre plein de ressources

- démarrage rapide : <http://microbit.org/guide/quick>

chaque étudiant recevra son kit avec un BBC micro:bit (contre caution)

Utilisation + encadrement

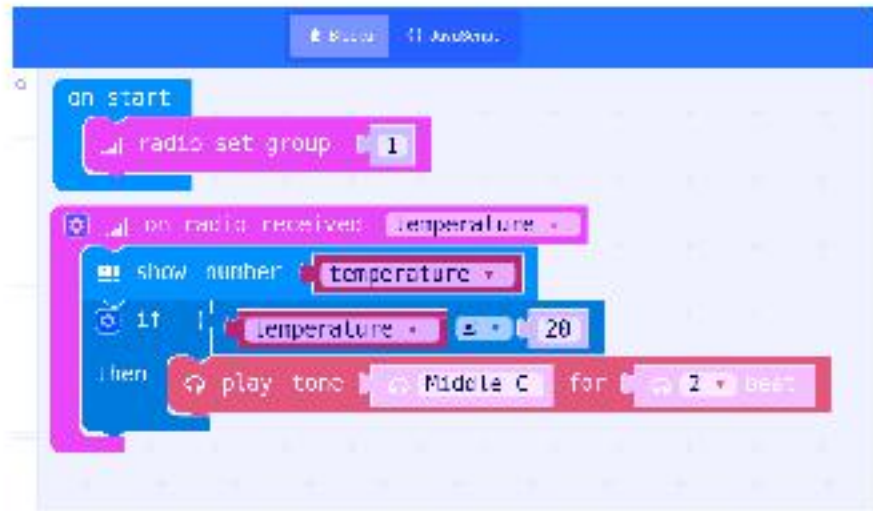
- le BBC micro:bit sera utilisé pour introduire les concepts (cours + TP)
- à chaque TP, vous recevrez des problèmes à résoudre chez vous
- un étudiant-tuteur assure des permanences au local du CSLabs (2ème étage, local 226) chaque jeudi de 13h30 à 15h30 (accès libre)
- vous serez amené à remplir des questionnaires, merci de le faire sérieusement et à chaque fois ⇒ évaluation du projet micro:PUNCH

Disclaimer : the text "BBC" and "micro:bit" are trademarks of the BBC. micro:PUNCH is not endorsed, sponsored nor associated with the Foundation. See <http://microbit.org> for more info on the micro:bit ecosystem

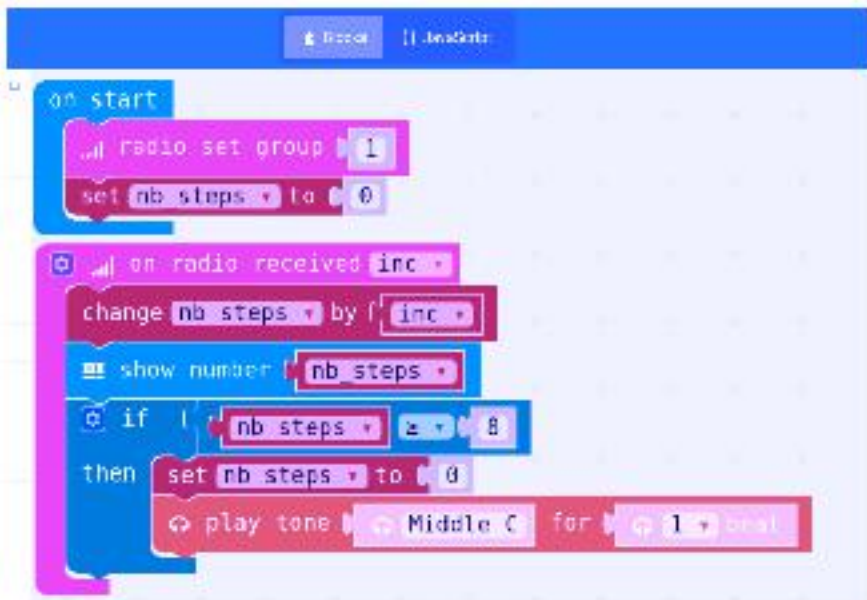
Le BBC micro:bit en action : alarme antigel sonore



Le BBC micro:bit en action : alarme antigel sonore







Le BBC micro:bit en action : versions pour un seul micro:bit



Le BBC micro:bit en action : versions pour un seul micro:bit



Organisation du cours

INFOB131 = 11 crédits - 60h+45h

- ECTS = "European Credits Transfer System"
- 1 ECTS = 30h de travail tout type confondu
- 10/11 ECTS = \pm 300 heures de travail
 - 60h de cours + 45h de TP's (= présence "obligatoire" à celle)
 - reste \pm 200h de travail personnel (recherche, travaux, vos collègues)
 - études : plusieurs cours/TP's : statique 4 heures, points des questions, etc

Comment préparer un cours ?

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \approx 300 heures de travail
 - 60h de cours + 45h de TP's (= présence "obligatoire" à celle)
 - reste \approx 200h de travail personnel (exercices, travaux, vos collègues)
 - études : plusieurs cours/TP's : statistiques 4 heures, points des questions, etc

Comment préparer un cours ?

INFOB131 = 11 crédits - 60h+45h

- ↗ ECTS = "European Credits Transfer System"
- ↗ 1 ECTS = 30h de travail **tout type confondu**
- ↗ 10/11 ECTS = \pm 300 heures de travail
 - ↗ 60h de cours + 45h de TP (= présence "obligatoire" en salle)
 - ↗ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ↗ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours ?

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \pm 300 heures de travail
 - ♦ 60h de cours + 45h de TP (= présence "obligatoire" en salle)
 - ♦ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ♦ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours/TP

- mettre au propre/rajouter des notes \Rightarrow ai-je tout compris ?
- refaire les illustrations (notabook) \Rightarrow suis-je capable de coder cela ?
- exercices complémentaires \Rightarrow permet une vraie interaction en cours !
- noter les questions que j'ai \Rightarrow début du cours, "question time"

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \pm 300 heures de travail
 - ♦ 60h de cours + 45h de TP (= présence "obligatoire" en salle)
 - ♦ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ♦ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours/TP

- ♦ mettre en ordre/relire ses notes \Rightarrow ai-je tout compris?
- ♦ refaire les illustrations (notebook) \Rightarrow suis-je capable de coder cela?
- ♦ exercices complémentaires \Rightarrow permet une vraie interaction au cours!
- ♦ noter les questions que j'ai \Rightarrow début du cours, "question time"

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \pm 300 heures de travail
 - ♦ 60h de cours + 45h de TP (= présence "obligatoire" en salle)
 - ♦ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ♦ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours/TP

- ♦ mettre en ordre/relire ses notes \Rightarrow ai-je tout compris?
- ♦ refaire les illustrations (notebook) \Rightarrow suis-je capable de coder cela?
- ♦ exercices complémentaires \Rightarrow permet une vraie interaction au cours
- ♦ noter les questions que j'ai \Rightarrow début du cours, "question time"

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \pm 300 heures de travail
 - ♦ 60h de cours + 45h de TP (= présence "obligatoire" en salle)
 - ♦ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ♦ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours/TP

- ♦ mettre en ordre/relire ses notes \Rightarrow ai-je tout compris?
- ♦ refaire les illustrations (notebook) \Rightarrow suis-je capable de coder cela?
- ♦ exercices complémentaires \Rightarrow permet une vraie interaction au cours!
- ♦ noter les questions que j'ai au début du cours, "question time"

INFOB131 = 11 crédits - 60h+45h

- ♦ ECTS = "European Credits Transfer System"
- ♦ 1 ECTS = 30h de travail **tout type confondu**
- ♦ 10/11 ECTS = \pm 300 heures de travail
 - ♦ 60h de cours + 45h de TPs (= présence "obligatoire" en salle)
 - ♦ reste \pm 200h de travail personnel (tout seul ou avec vos collègues)
 - ♦ étude, préparer cours/TPs, pratique @ home, poser des questions, etc.

Comment préparer un cours/TP

- ♦ mettre en ordre/relire ses notes \Rightarrow ai-je tout compris ?
- ♦ refaire les illustrations (notebook) \Rightarrow suis-je capable de coder cela ?
- ♦ exercices complémentaires \Rightarrow permet une vraie interaction au cours !
- ♦ noter les questions que j'ai \Rightarrow début du cours, "question time"

- ♦ bloc n°1 : bases de la programmation
 - ♦ à la découverte d'un nouveau monde
 - ♦ valeurs, expressions et variables
 - ♦ instructions conditionnelles et exceptions
 - ♦ fonctions et spécifications
- ♦ bloc n°2 : itération et structures de données
 - ♦ boucles et récursité
 - ♦ structures de données et listes
- ♦ bloc n°3 : notions d'algorithmique
 - ♦ introduction à l'algorithmique
 - ♦ programmation récursive

après chaque bloc, une semaine est consacrée à un mini-projet en groupe

- ♦ travail obligatoire et coté en petit groupe (groupes imposés)
- ♦ occasion de mettre en pratique et de "fixer" ses connaissances du bloc
- ♦ chaque mini-projet comptera pour 10% (absence non-justifiée ⇒ 0/20)

- ❖ bloc n°1 : bases de la programmation
 - à la découverte d'un nouveau monde
 - valeurs, expressions et variables
 - instructions conditionnelles et exceptions
 - fonctions et spécifications
- bloc n°2 : itération et structures de données
 - boucles et invariants
 - structures de données et fichiers
- bloc n°3 : notions d'algorithmique
 - introduction à l'algorithmique
 - programmation récursive

après chaque bloc, une semaine est consacrée à un mini-projet en groupe :

- travail obligatoire et coté en petit groupe (groupes imposés)
- occasion de mettre en pratique et de "fixer" tous les concepts du bloc
- chaque mini-projet comptera pour 10% (absence non-justifiée ⇒ 0/20)

- ❖ bloc n°1 : bases de la programmation
 - à la découverte d'un nouveau monde
 - valeurs, expressions et variables
 - instructions conditionnelles et exceptions
 - fonctions et spécifications
- bloc n°2 : itération et structures de données
 - boucles et invariants
 - structures de données et fichiers
- bloc n°3 : notions d'algorithmique
 - introduction à l'algorithmique
 - programmation récursive

après chaque bloc, une semaine est consacrée à un mini-projet en groupe

• travail obligatoire et coté en petit groupe (groupes imposés)

• occasion de mettre en pratique et de "fixer" tous les concepts du bloc

• chaque mini-projet comptera pour 10% (absence non-justifiée ⇒ 0/20)

- ❖ bloc n°1 : bases de la programmation
 - à la découverte d'un nouveau monde
 - valeurs, expressions et variables
 - instructions conditionnelles et exceptions
 - fonctions et spécifications
- ❖ bloc n°2 : itération et structures de données
 - boucles et invariants
 - structures de données et fichiers
- ❖ bloc n°3 : notions d'algorithmique
 - introduction à l'algorithmique
 - programmation récursive

après chaque bloc, une semaine est consacrée à un mini-projet en groupe

- ❖ travail obligatoire et coté en petit groupe (groupes imposés)
- occasion de mettre en pratique et de "fixer" tous les concepts du bloc
- chaque mini-projet comptera pour 10% (absence non-justifiée = 0/20)

Travaux pratiques (chaque semaine entre les cours)

- TPx papier
- TPx machine

Examinés écrits en cours et ap. TP

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Examen écrit en cours et ap. TP

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Préparation avant et après les TPs

Travaux pratiques (chaque semaine entre les cours)

- ➔ TPs papier
- ➔ TPs machine

Pourquoi venir au cours et au TP ?

- lire les slides n'est pas (du tout) suffisant
- simulation sur les pairs (plus on est de fous...)
- présence du prof, des assistants, matériel disponible...
- interactions (notebook en live, questions, quiz, etc.)
- ça "force" à avancer dans la matière (et on se sent moins seul)
- participation active aux mini-projets visant à découvrir la matière
- vraiment écrit (sur papier) portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
 - simulation sur les pairs (plus on est de fous...)
 - présence du prof, des assistants, matériel disponible...
 - interactions (notebook en live, questions, quiz, etc.)
 - ça "force" à avancer dans la matière (et on se sent moins seul)
 - participation active aux mini-projets nécessaires de connaître la matière
 - vraiment écrit (sur papier !) portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
- émulation par les pairs (plus on est de fous...)
- ◆ présence du prof, des assistants, matériel disponible...
- ◆ interactions (notebook en live, questions, quiz, etc.)
- ◆ ça "force" à avancer dans la matière (et on se sent moins seul)
- ◆ participation active aux mini-projets visant à découvrir la matière
- ◆ vraiment écrit (sur papier !) portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
- émulation par les pairs (plus on est de fous...)
- ◆ présence du prof, des assistants, matériel disponible...
- interactions (notebook en live, questions, quizz, etc.)
- ◆ ça "force" à avancer dans la matière (et on se sent moins seul)
- ◆ participation active aux mini-projets visant à consolider la matière
- ◆ vraiment écrit (sur papier) portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
- ◆ émulation par les pairs (plus on est de fous...)
- ◆ présence du prof, des assistants, matériel disponible...
- ◆ interactions (notebook en live, questions, quizz, etc.)
- ◆ ça "force" à avancer dans la matière (et on se sent moins seul)
- ◆ participatif active aux mini-projets, résultats de consultations, la pratique
- ◆ verbatim écrit (sur papier) portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
- émulation par les pairs (plus on est de fous...)
- ◆ présence du prof, des assistants, matériel disponible...
- interactions (notebook en live, questions, quizz, etc.)
- ◆ ça "force" à avancer dans la matière (et on se sent moins seul)
- participation active aux mini-projets nécessite de connaître la matière

« vraiment écrit (sur papier) » portant sur la théorie **et** les exercices

Travaux pratiques (chaque semaine entre les cours)

- TPs papier
- TPs machine

Pourquoi venir au cours et au TP ?

- ◆ lire les slides n'est pas (du tout) suffisant
- émulation par les pairs (plus on est de fous...)
- ◆ présence du prof, des assistants, matériel disponible...
- interactions (notebook en live, questions, quizz, etc.)
- ◆ ça "force" à avancer dans la matière (et on se sent moins seul)
- participation active aux mini-projets nécessite de connaître la matière
- examen : écrit (sur papier !) portant sur la théorie **et** les exercices

Et après ? Le laboratoire de développement de programme !

Le projet en quelques mots (en 2015-16)

par groupes (libres !) de 3 étudiants, implémenter

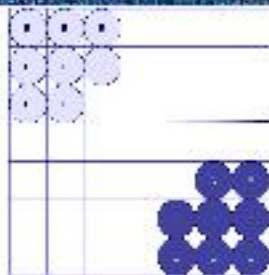
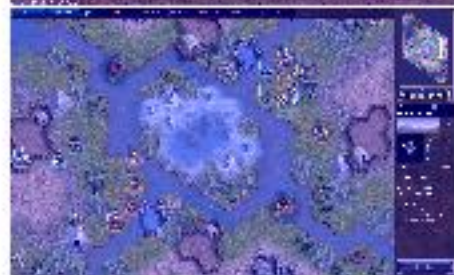
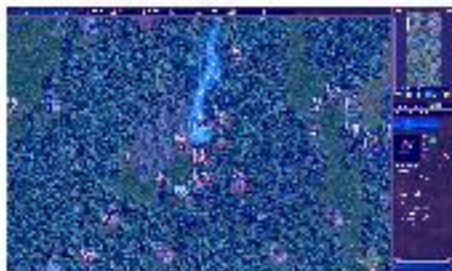
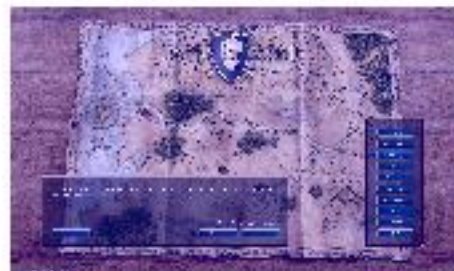
- ♦ un programme permettant de jouer à un jeu raisonnablement complexe
- ♦ une intelligence artificielle (IA) contre laquelle jouer à ce jeu

le jeu est imposé \Rightarrow découverte de l'énoncé en groupe en février

Organisation du quadrimestre

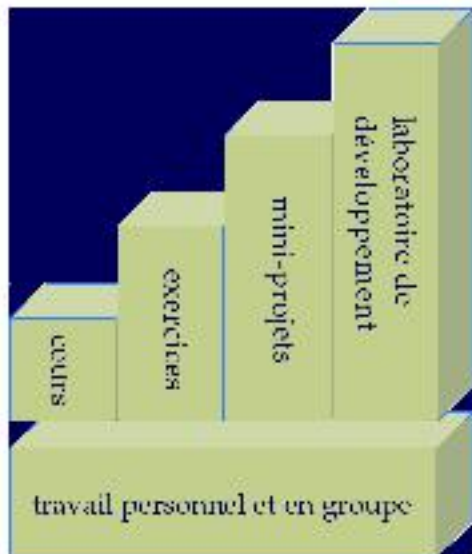
- ♦ le laboratoire est l'aboutissement d'INFOB131 et des mini-projets
- ♦ uniquement des séances en groupe encadrées par les assistants
- ♦ pas assez \Rightarrow cours de 3 ECTS = 90 heures de travail/étudiant
- ♦ les étapes du laboratoire sont soigneusement balisées

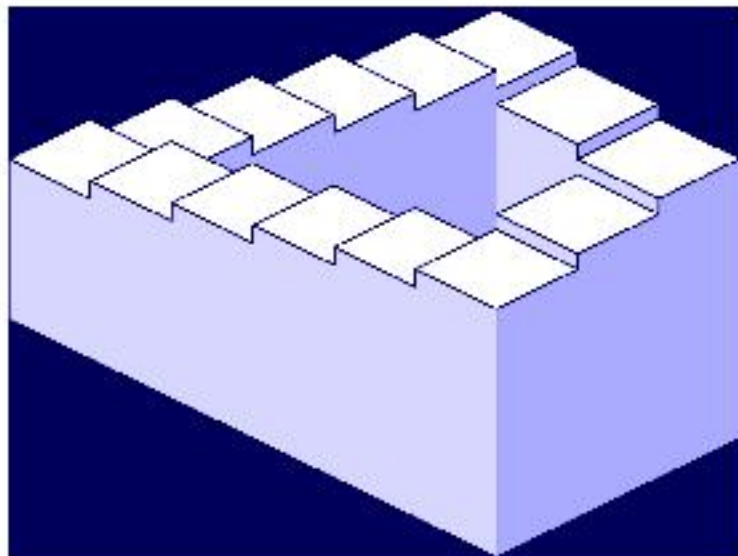
Et après ? Le laboratoire de développement de programme !



Et après ? Le laboratoire de développement de programme !







Être capable de résoudre des problèmes simples à l'aide d'algorithmes (pseudo-code) et de les traduire en langage de programmation (Python).

- manipuler des variables et des valeurs
- concevoir des structures conditionnelles et les conditions associées
- abstraire du code à l'aide de fonctions
- écrire une spécification de fonction et documenter son code
- utiliser des boucles for/while et écrire l'invariant correspondant
- choisir et manipuler une structure de données adaptée au problème
- créer, utiliser et manipuler des fichiers, notamment avec Pickle
- écrire un algorithme simple et caractériser sa complexité
- concevoir un algorithme récursif

Faire preuve d'une compréhension des concepts, c-à-d exprimer avec ses propres mots la théorie et les outils vus au cours et expliquer dans quel contexte ceux-ci sont utiles. Une maîtrise du langage Python est attendue.

L'étudiant sera capable de comprendre du code, de le critiquer et d'y apporter les corrections nécessaires.



"Apprendre à programmer avec Python 3", G. Swinnen, éd. 2012
https://inforef.be/swi/download/apprendre_python3_5.pdf

+ ressources sur le site webcampus du cours

certaines images de ce cours proviennent de <https://commons.wikimedia.org>



Benoît Frénay - bureau n°408
benoit.frenay@unamur.be



Nesrine Noughi - bureau n°435
nesrine.noughi@unamur.be



Julie Henry - bureau n°426
julie.henry@unamur.be



Adrien Bibal - bureau n°435
adrien.bibal@unamur.be

Vous remarquez une erreur ? Vous avez des questions ?

- ◆ contactez un membre de l'équipe pédagogique
- ➔ en particulier, vous pouvez écrire à benoit.frenay@unamur.be
- ◆ l'équipe pédagogique est là pour vous aider !

ce cours est "nouveau", tout feedback/suggestion est plus que le bienvenu !



The most exciting phrase to hear in science,
the one that heralds new discoveries, is not 'Eureka !'
but 'That's funny...' – Isaac Asimov (1920–1992)

The beginning of knowledge is the discovery of something
we do not understand. – Frank Herbert (1920–1986)