



# LOGIQUE DE PROGRAMMATION

**Bastien Gorissen & Thomas Stassin**

# INTRO

Hello !

Team Panoptes: studio de jeu vidéo indépendant, dont nous sommes les 2 programmeurs principaux. Nous faisons aussi du Python, et nous sommes formateurs, principalement chez les GAME à Interface3.

Dans ce cours, nous allons doucement entamer notre découverte de la programmation. Vous allez faire vos premiers pas sur un chemin aussi passionnant que gratifiant!

# PROLOGUE

“Qu’est-ce que la programmation ?”

# LA PROGRAMMATION, CET ART MYSTÉRIEUX...

La programmation est la discipline qui consiste à écrire du code. Code qui sera ensuite exécuté par un ordinateur.

C'est donc "comment dire à un ordinateur ce qu'il doit faire".

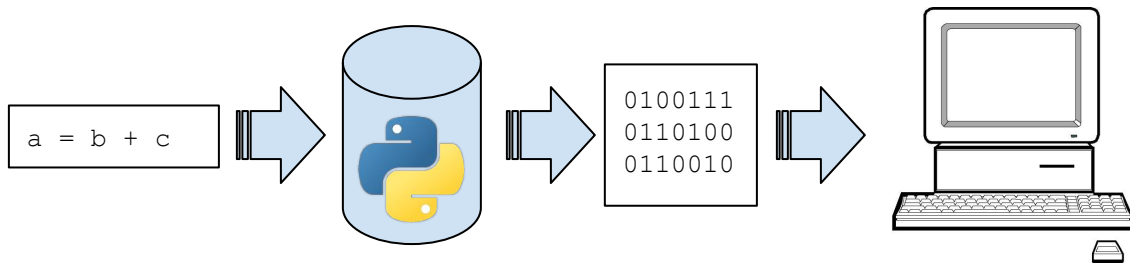
De Windows à Firefox, de Twitter à Fortnite, tout ce qui s'exécute sur un ordinateur quel qu'il soit a dû être programmé.

Et comme l'ordinateur ne comprend pas le Français, nous allons devoir passer par un langage de programmation.

# UN LANGAGE DE PROGRAMMATION, VOUS DITES ?

Il existe plein de conventions différentes pour parler à l'ordinateur : Python, C#, Java, PHP, ...

L'ordinateur ne les comprend pas directement, mais Python (par exemple) va lui rendre ça digeste, sous forme d'instructions binaires.



# LA PROGRAMMATION EST PARTOUT!

De nos jours, les ordinateurs sont omniprésents, et régissent tous les aspects de notre vie quotidienne, directement et indirectement.

Apprendre à programmer, c'est apprendre à maîtriser ces outils incroyables devant lesquels nous passons tous de plus en plus de temps.

C'est une compétence qui vous ouvrira un nombre infini de portes, tant au niveau professionnel que personnel. Que ça soit pour le web, la création d'applications mobiles, l'écriture de simples scripts...

# ET POUR AUJOURD'HUI ?

Pratiquement tous les langages de programmation se basent sur les même principes.

Il est donc utile de nous pencher sur lesdits principes avant de plonger dans l'étude d'un langage en particulier!

Du coup, aujourd'hui nous allons tenter d'apprendre à penser comme des programmeur·se·s !

# VIVE LE PAPIER!

“Go with the flow”



# NOUS ALLONS UTILISER UN OUTIL...

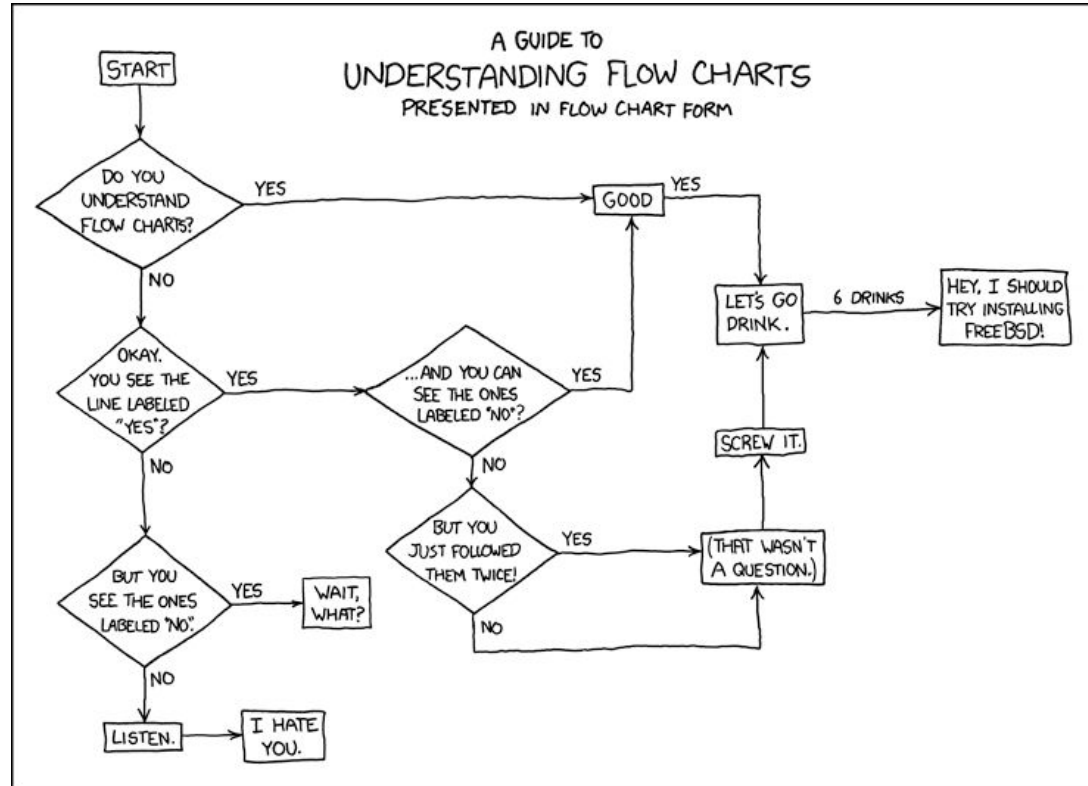
Il existe plusieurs façons de représenter des programmes de façon indépendante d'un langage particulier.

Par exemple, le pseudo-code, une façon d'écrire "en Français" les actions d'un programme de façon semi-structurée.

Mais aussi des outils visuels, comme les flowcharts, ou "diagrammes de flux".

C'est ce que nous allons commencer par utiliser.

# DES FLOWCHARTS !



# CRÉER SES FLOWCHARTS

La façon la plus simple de créer un flowchart est simplement de le dessiner sur papier.

Vous pouvez aussi utiliser un programme de dessin comme Paint, Gimp ou autre.

Ou bien, un outil comme Google Drawings, adapté à ce genre de schémas (<https://docs.google.com/drawings/>).

Nous parlerons aussi brièvement de Flowgorithm, mais malheureusement certains points sont un peu plus compliqués, et donc mieux vaut commencer par les méthodes plus "libres".

# IER PROGRAMME

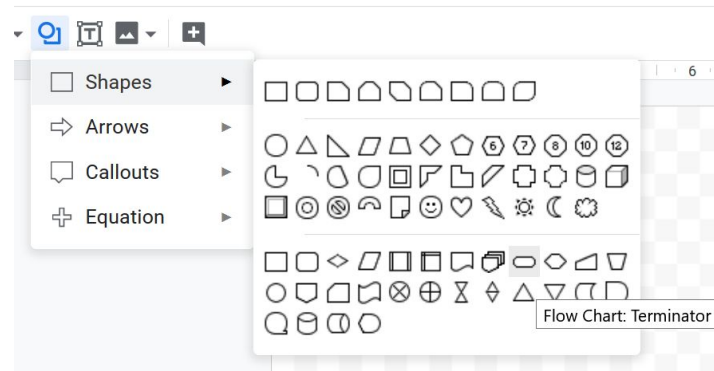
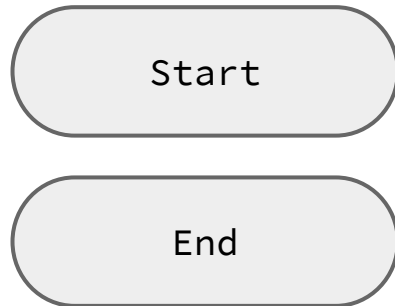
“...ou presque”

# PREMIER(S) BLOC(S)

Pour élaborer nos flowcharts, nous allons avoir besoin de différents blocs.

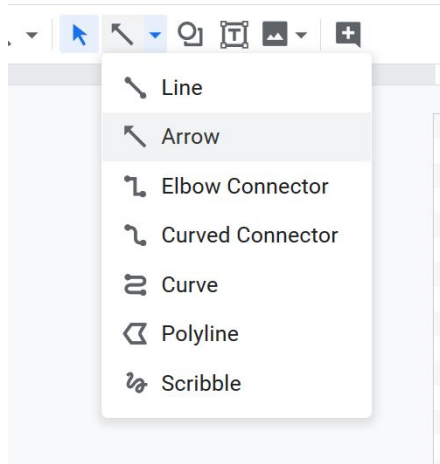
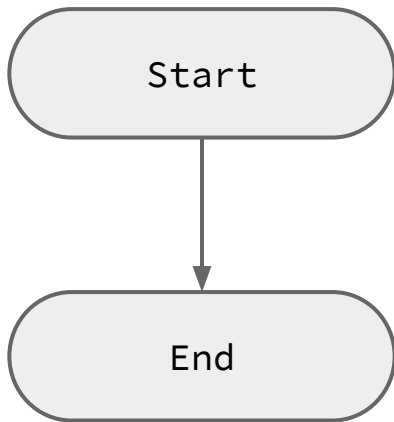
Au minimum, nos programmes devront avoir un début, et une fin (idéalement au moins).

Pour ça, on utilisera les blocs suivants :



# MINIMUM SYNDICAL

Le programme le plus simple que nous pouvons créer avec ces blocs est un programme qui ne fait rien :



Pas super excitant, mais c'est un programme valide! Vous pouvez suivre son comportement de "Start" à "End".

"HELLO, WORLD!"

“On sacrifie à la tradition.”

# INSTRUCTION DE SORTIE

Comme première chose utile, nous allons découvrir ce qu'on appelle une *instruction de sortie*.

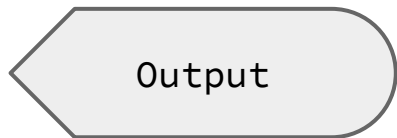
Une instruction de sortie a pour but de faire "sortir" de l'information du programme. Quelques exemples :

- Ecrire un message à l'écran
- Ecrire dans un fichier
- Imprimer via une imprimante
- Envoyer une requête vers un serveur
- etc.

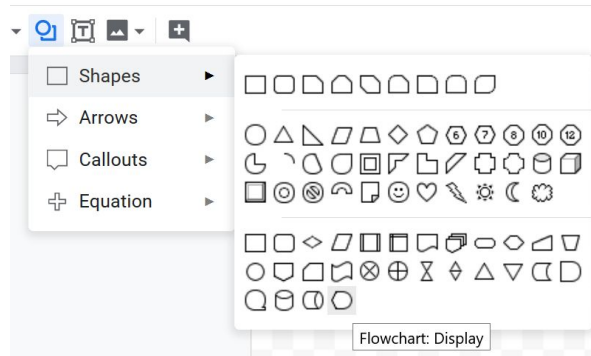
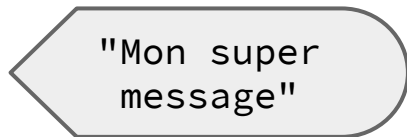


# LES SORTIES EN FLOWCHART

Le bloc représentant une instruction de sortie est censée rappeler un tube cathodique (c'est vieux les flowcharts...).



En pratique, si on veut que le programme affiche un message à l'utilisateur, on le met entre guillemets dans le bloc :



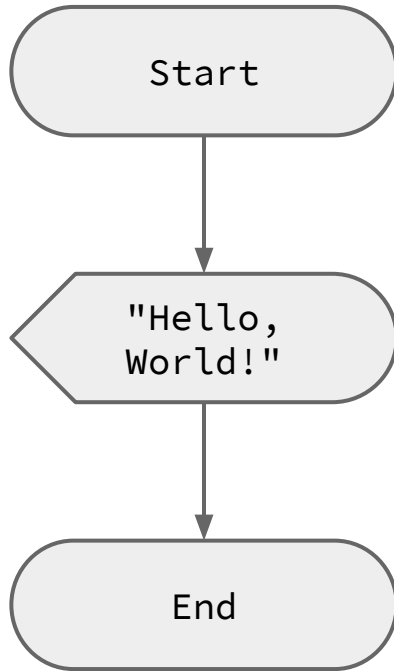
# EXERCICE 1: LE "HELLO, WORLD!"

Votre premier exercice consiste à dessiner un flowchart qui représente un programme qui :

- Commence
- Affiche "Hello, World!"
- Se termine

N'oubliez pas que les flèches indiquent le sens de parcours de votre flowchart.

# EXERCICE 1: LE "HELLO, WORLD!" (SOLUTION)



"TOC TOC"

“Entrée.”

# INSTRUCTION D'ENTRÉE

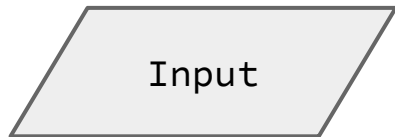
L'*instruction d'entrée* est un peu l'inverse de l'instruction de sortie: elle fait "entrer" de l'information depuis le monde extérieur dans votre programme.

Quelques exemples:

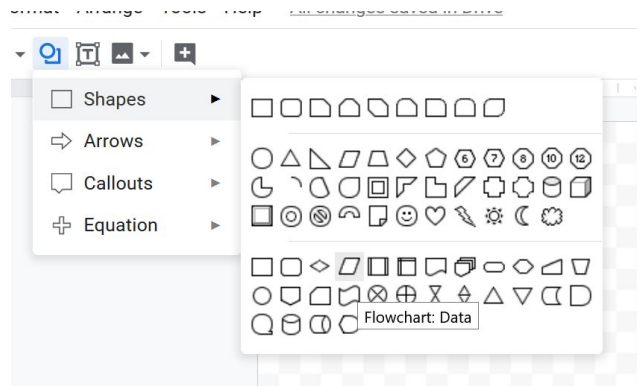
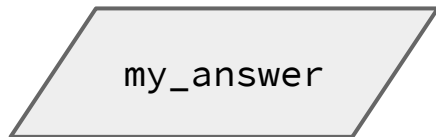
- Demander à l'utilisateur d'entrer quelque chose au clavier
- Recevoir une requête d'un browser
- Lire un fichier
- etc.

# LES ENTRÉES EN FLOWCHART

Le bloc représentant une instruction de sortie représente cette fois-ci un clavier. Avec juste un peu d'imagination.



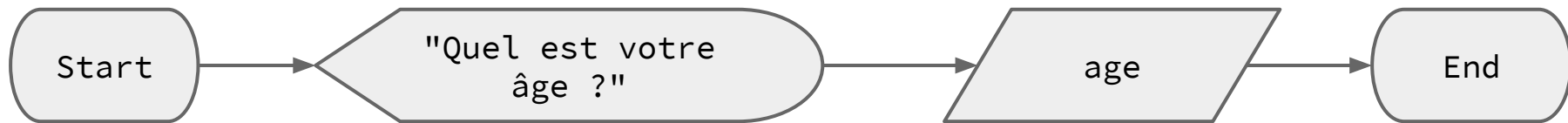
Comme on va recevoir de l'information, mais qu'on ne sait pas exactement quoi (on ne contrôle pas le comportement de l'utilisateur), on met un mot pour représenter les informations reçues.



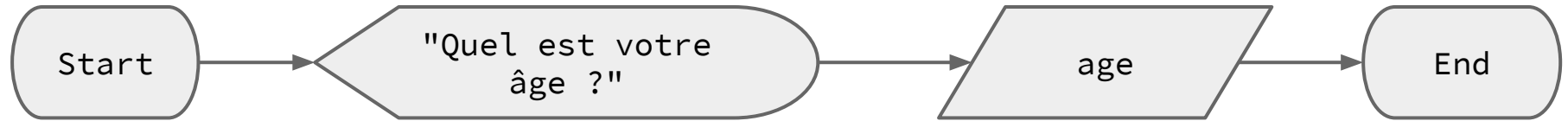
# UN EXEMPLE?

Imaginons qu'on veuille demander l'âge de l'utilisateur.  
Pour avoir un programme cohérent, on va devoir procéder en deux temps:

1. Dire à l'utilisateur ce qu'on veut de lui (son age)
2. Utiliser une instruction d'entrée pour recevoir cette information.



# QUELQUES POINTS IMPORTANTS



- On a besoin des deux instructions pour faire ce qu'on peut considérer comme une seule opération, car on interagit avec un humain qui a besoin d'instructions.
- Chaque instruction est "bête" et ne fait qu'exactly ce qu'elle est prévue pour faire.
- On pourrait utiliser le mot **age** pour représenter la valeur que l'utilisateur va entrer (qui changera donc à chaque fois que le programme est exécuté).



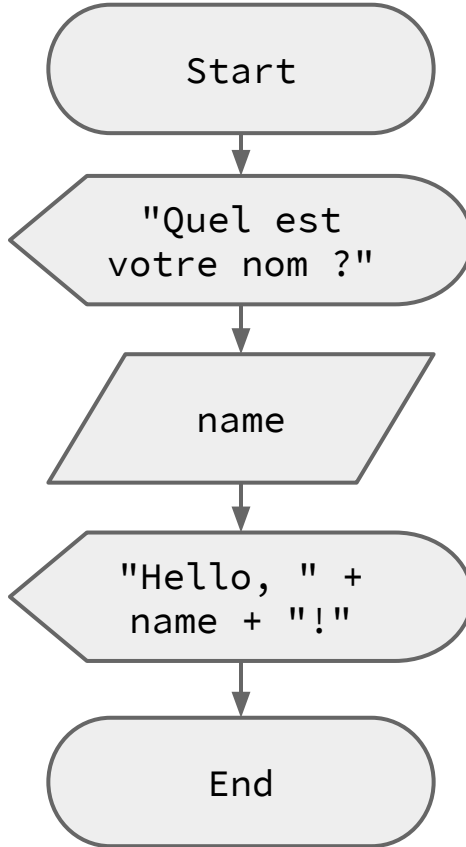
## EXERCICE 2: LE "HELLO, <INSERT YOU>!"

Nous allons faire un programme un peu plus personnalisé.  
Cette fois, dessinez un programme qui:

- Commence
- Demande son nom à l'utilisateur (attention, cf exemple ;)
- Affiche "Hello, ...!", où ... est le nom entré au point précédent.
- Se termine

Décidons que, par convention, si vous vouliez utiliser une valeur comme l'âge de l'exemple précédent dans du texte, vous pouvez écrire : **"Vous avez " + age + " ans"**

## EXERCICE 2: LE "HELLO, <INSERT YOU>!" (SOLUTION)



# DÉCISIONS

“Un monde manichéen”

# RENDRE UN PROGRAMME INTÉRESSANT

Avec les instructions d'entrée et de sortie, vous pouvez faire un programme qui réagit à l'utilisateur, mais toujours de la même façon.

Pas de quoi rendre les choses intéressantes...

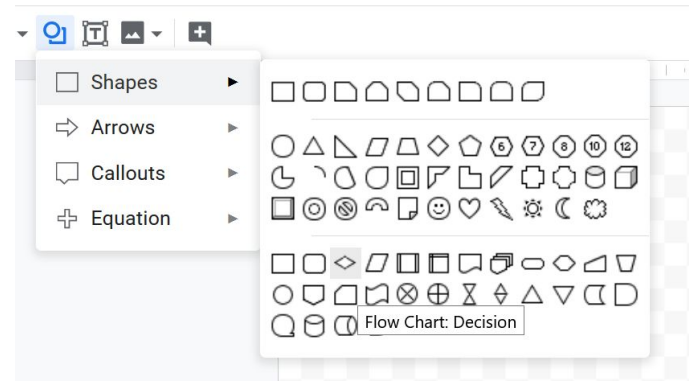
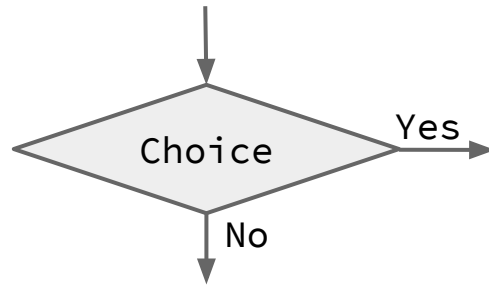
Par contre, et si on pouvait faire en sorte que le programme réagisse différemment selon les entrées fournies par l'utilisateur ?

C'est un mécanisme fondamental en programmation, et est évidemment possible via l'utilisation d'un bloc *décision*.

# INSTRUCTION DE CONTRÔLE DE FLUX

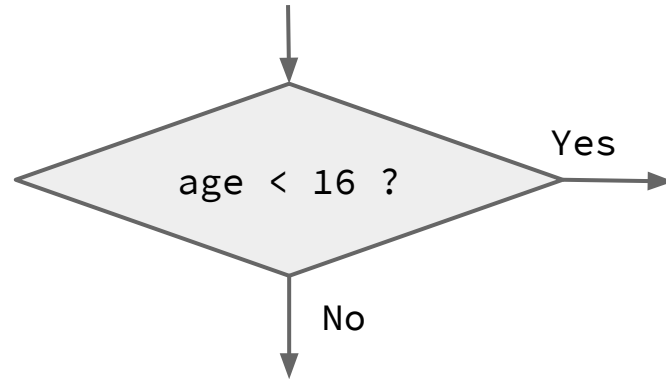
Les ordinateurs affectionnent tout ce qui est binaire. Après tout, ils fonctionnent sur base de 0 et de 1.

Du coup, pour contrôler notre programme, nous allons pouvoir prendre des décisions basées sur des questions "oui ou non".



# REMARQUES

- Le bloc de décision est le seul qui a deux flèches sortantes, une pour "oui", et une pour "non".
- Toutes les questions peuvent à priori se réduire à une série de questions binaires.
- On utilise souvent des raccourcis mathématiques pour écrire la question. Par exemple:



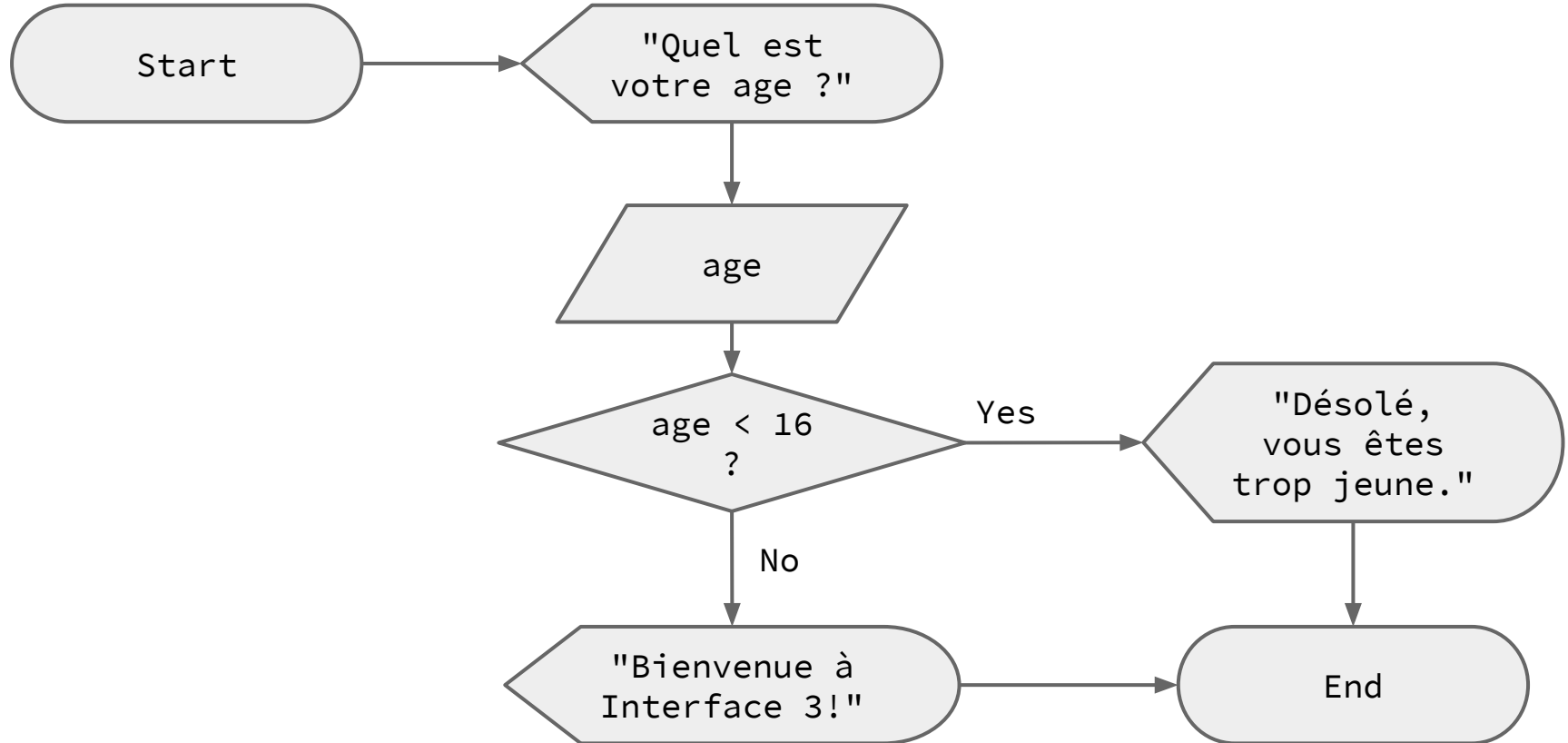
# EXERCICE 3: "WELCOME TO IF3 (OR NOT)"

Cette fois-ci, réagissons à l'entrée utilisateur. Concevez un programme qui:

- Commence
- Demande son âge à l'utilisateur
- Si l'âge est inférieur à 16:
  - Afficher "Désolé, vous êtes trop jeune..."
- Sinon
  - Afficher "Bienvenue à Interface3 !"
- Se termine

En bonus: comment faire pour mettre un âge maximum (disons, 130 ans ?)

# EXERCICE 3: "WELCOME TO IF3 (OR NOT)" (SOLUTION)





# AUTOMATION

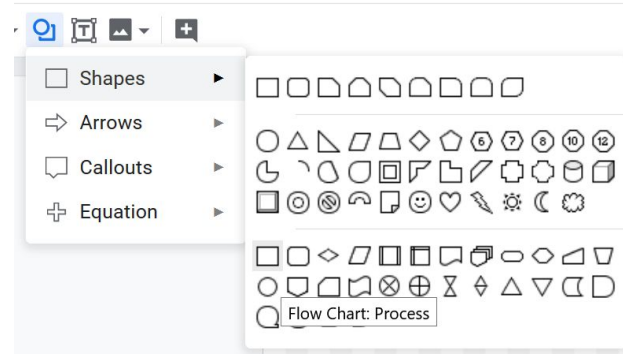
“Contrôlez votre formateur!”

# AUTRE INSTRUCTIONS

Evidemment, on pourrait imaginer beaucoup d'opérations qu'on voudrait pouvoir faire effectuer à un ordinateur. Ou un robot.

Nous allons faire quelques exercices au cours desquels vous allez pouvoir donner des instructions à un "robot" (aka le formateur) pour effectuer diverses tâches. Pour cela, nous allons introduire un nouveau bloc qui servira pour les instructions génériques :

Instruction



# JEU D'INSTRUCTIONS

Pour les exercices, nous allons définir un environnement et une série d'instructions disponibles.

3 boîtes sont présentes : B0, B1, B2

Les boîtes peuvent contenir des objets de différentes couleurs (rouges, verts, bleus, blancs, noirs, ...)

Le "robot" ne peut tenir qu'un seul objet en main, ne peut pas choisir consciemment quel objet il prend dans une boîte (il est choisi au hasard), et peut s'interroger sur l'objet qu'il tient en main pour connaître sa couleur.

# JEU D'INSTRUCTIONS

Prendre un objet  
dans la boîte N

Prendre dans N

Mettre l'objet tenu  
dans la boîte N

Mettre dans N

La boîte N est-elle  
vide ?

Boîte N  
vide ?

L'objet tenu est-il  
rouge ?

Objet  
rouge ?

Version abrégée:

P N

M N

N vide ?

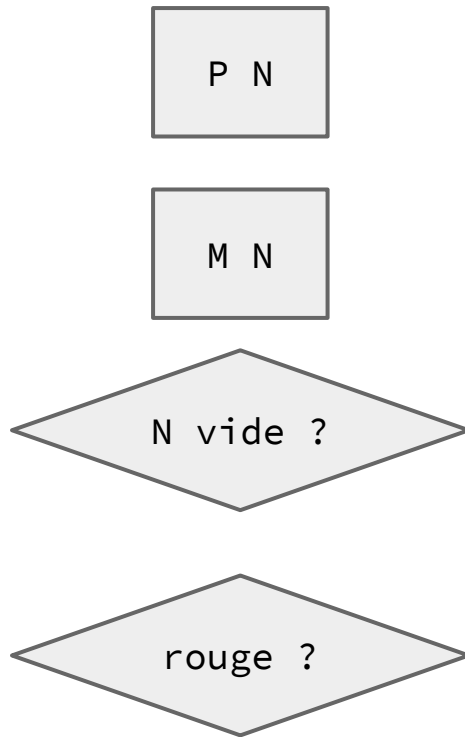
rouge ?

# ATTENTION AUX ERREURS

Le robot est un peu primitif, et ne gère pas très bien les imprévus. Il va bugger et se mettre en erreur si vous lui demandez de:

- prendre un objet dans une boîte vide.
- tester la couleur d'un objet alors qu'il n'en tient aucun.
- mettre un objet dans une boîte et qu'il n'en tient aucun.

Version abrégée:



# EXERCICE 4: "SORT IT OUT"

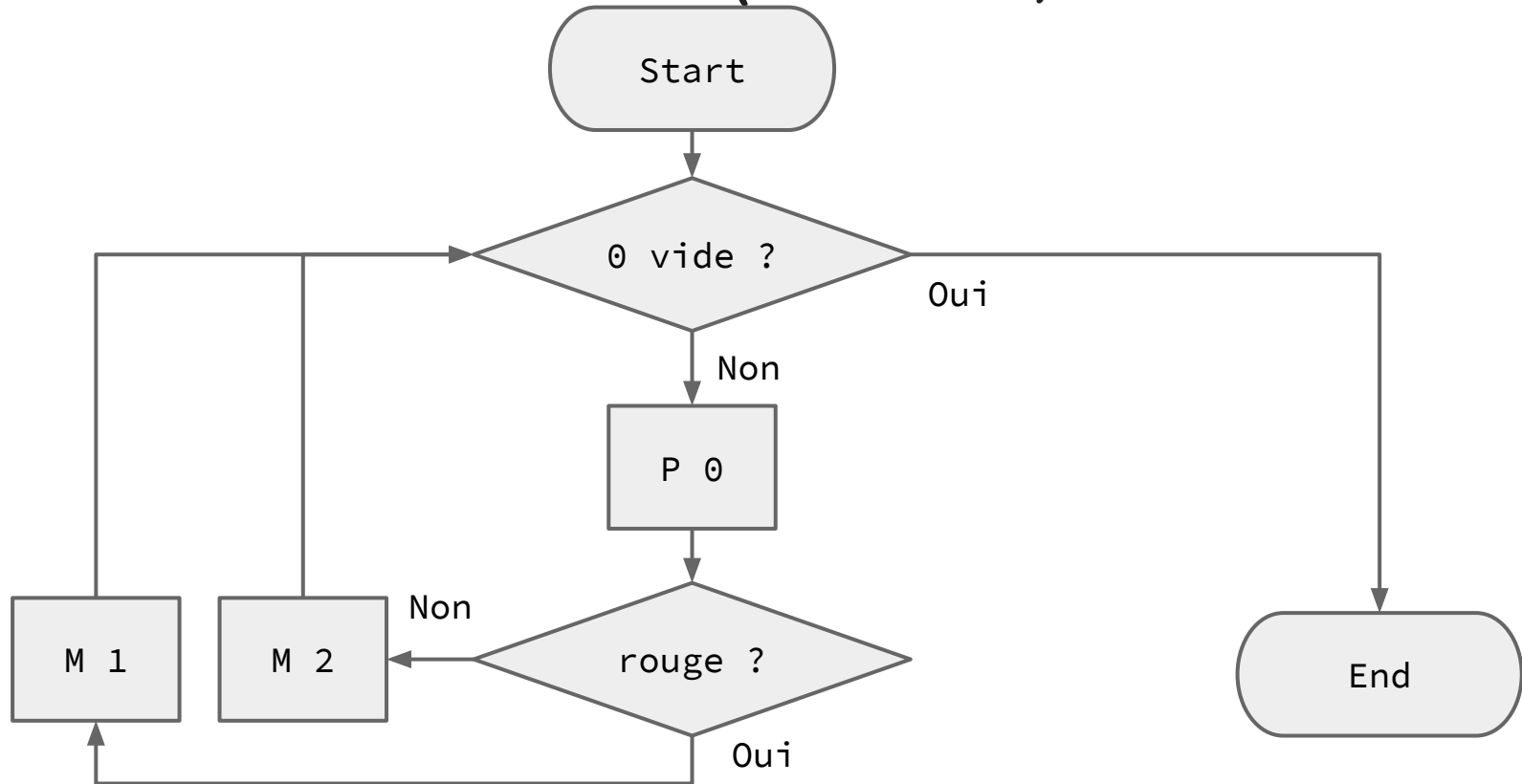
Conditions initiales:

- La boîte 0 contient **entre 0 et 200 objets, tous rouges ou bleus** (aucune autre couleur, mais un nombre inconnu de bleus et de rouges).
- Les boîtes 1 et 2 sont vides.

Résultat recherché:

- La boîte 0 est vide.
- La boîte 1 contient tous les objets rouges.
- La boîte 2 contient tous les objets bleus.

# EXERCICE 4: "SORT IT OUT" (SOLUTION)



# EXERCICE 5: "I LIKE TO... MOVE IT!"

Conditions initiales:

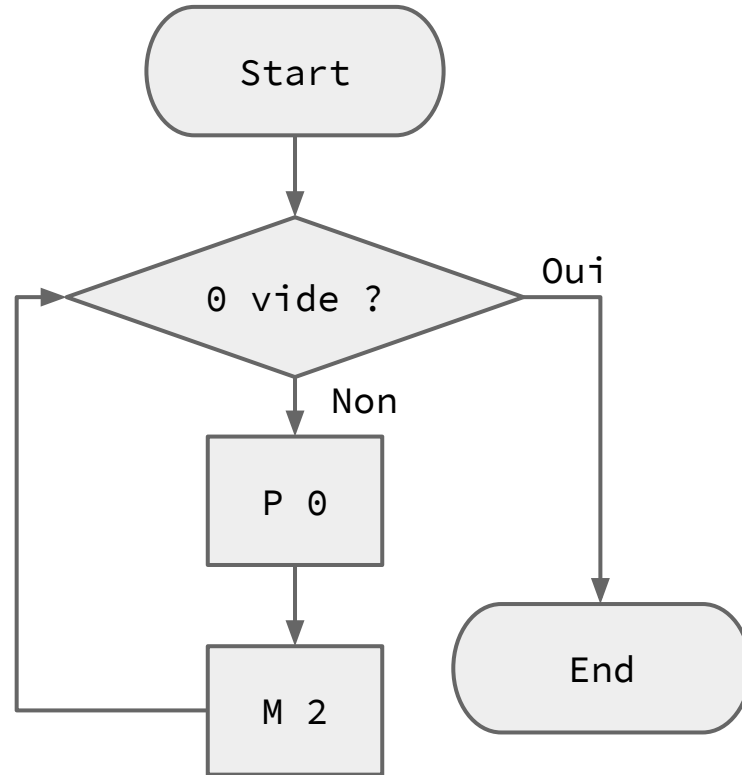
- La boîte 0 contient **entre 0 et 200 objets, de couleurs aléatoires.**
- Les boîtes 1 et 2 sont vides.

Résultat recherché:

- La boîte 0 est vide.
- La boîte 1 est vide.
- La boîte 2 contient tous les objets qui étaient initialement dans la boîte 0.



# EXERCICE 5: "I LIKE TO... MOVE IT!"



# EXERCICE 6: "SWITCHEROO"

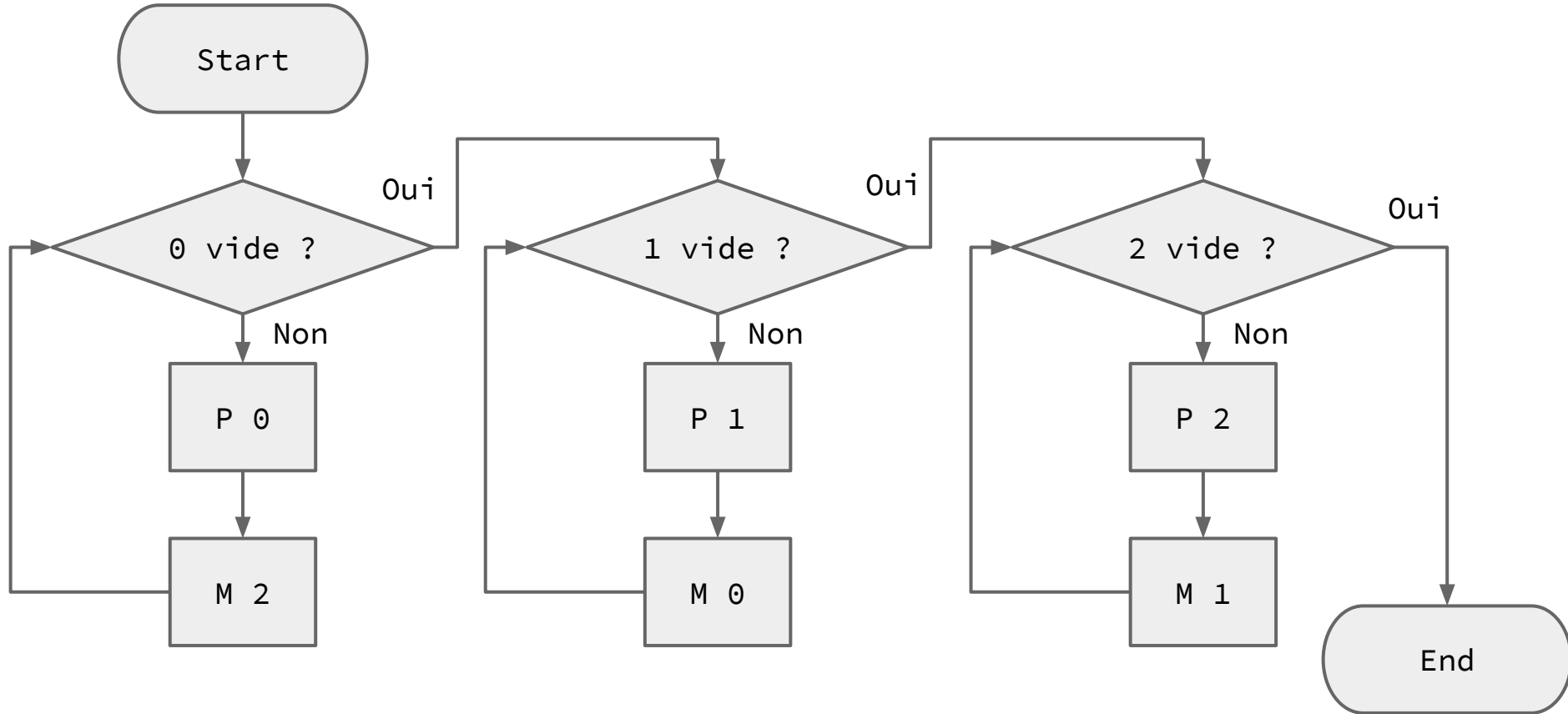
Conditions initiales:

- Les boîtes 0 et 1 contiennent chacune **entre 0 et 200 objets, de couleurs aléatoires.**
- La boîte 2 est vide.

Résultat recherché:

- La boîte 0 contient tous les objets qui étaient dans la boîte 1 initialement.
- La boîte 1 contient tous les objets qui étaient dans la boîte 0 initialement.
- La boîte 2 est vide.

# EXERCICE 6: "SWITCHEROO" (SOLUTION)



# ABSTRACTION

“Ce n'est pas un gros mot !”

# C'EST LA BASE DE TOUT!

Nous avons utilisé 2 aspects de ce qu'on appelle "abstraction".

1. En parlant des instructions d'entrées, nous avons représenté une valeur particulière par un mot (puisque'on ne pouvait pas prévoir la valeur). C'est un concept qui reviendra illico en Python!
2. Chaque instruction du "robot" représente en fait de nombreuses opérations, mais qu'on rassemble en un seul bloc (mouvement du bras, des doigts pour prendre les objets...)

# UNE NOUVELLE INSTRUCTION

L'exercice d'échange de contenu des boîtes nous a demandé une certaine répétition de la solution de l'exercice 5.

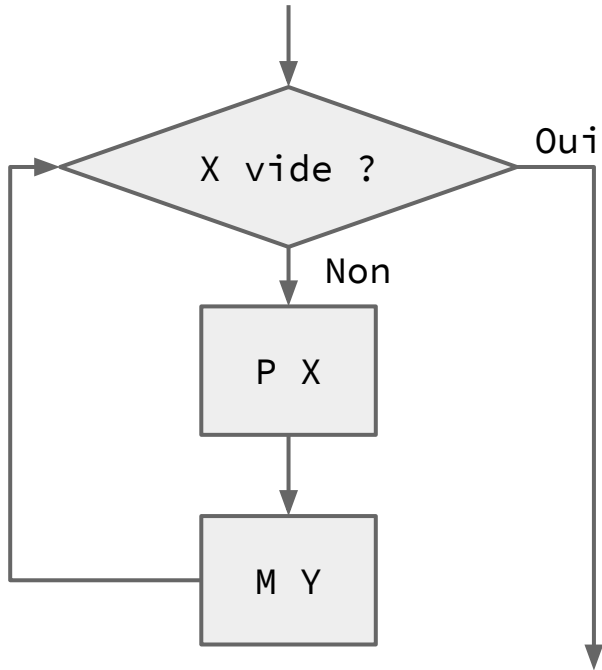
On pourrait décider de créer un nouveau bloc d'instruction pour se simplifier la vie!

Notez qu'un bloc a un "début" (là où la flèche entrante se connecte), et une "fin" (là d'où la flèche sortante part).

C'est comme si c'était un sous-programme !

# UNE NOUVELLE INSTRUCTION: DÉPLACER X DANS Y

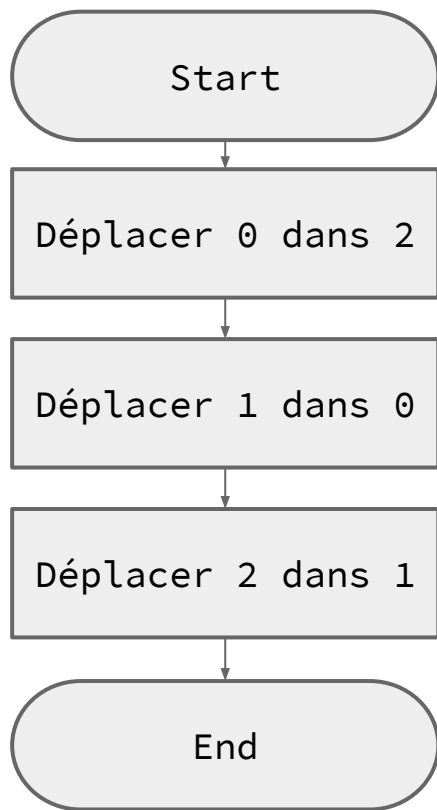
On pourrait donc faire :



devient:

Déplacer X dans Y

# EX. 6B: "SWITCHEROO 2 : ELECTRIC BOOGALOO" (SOLUTION)





# PAS MAL, HEIN ?

Ce matin, nous avons travaillé avec des opérations simples, et sur des problèmes bien définis.

Mais les notions vues vont revenir dans tous vos cours de programmation !

Le point sur lequel nous allons fortement insister est que tout doit rester aussi clair que ces flowcharts.

Quelques détails seront forcément un peu opaques pour éviter de vous surcharger de détails superflus, mais la programmation est un domaine très cartésien.

# FLOWGORITHM

“Un petit bonus”

# UN SOFT POUR FAIRE DES FLOWCHARTS

Flowgorithm est un programme qui vous permet de faire des flowcharts et tester leur comportement.

Vous pouvez le télécharger ici : <http://flowgorithm.org>

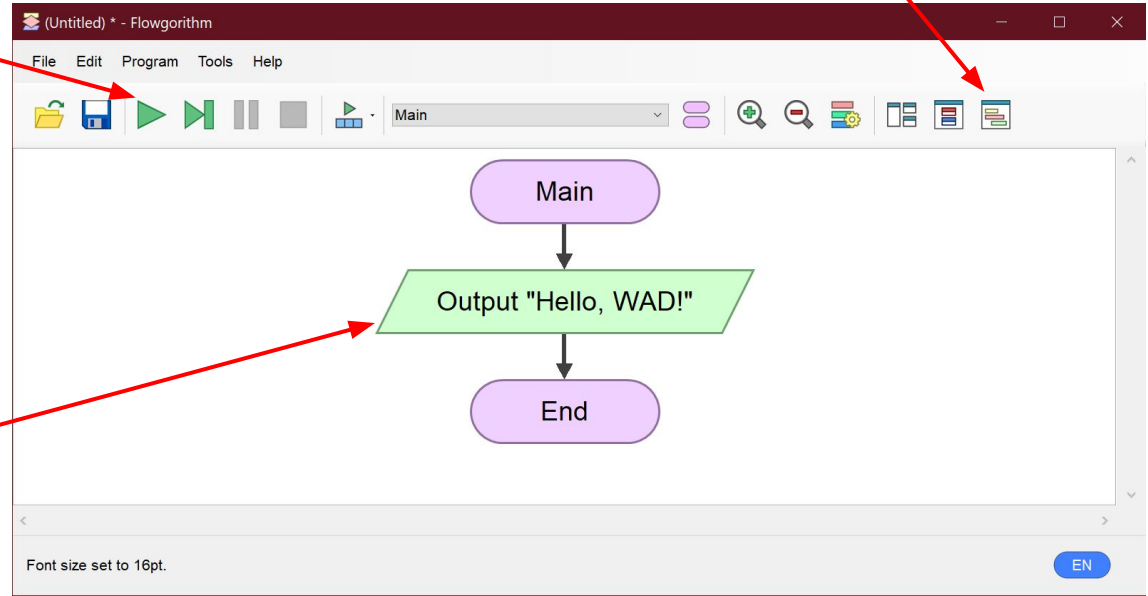
Il y a quelques petites complexités par rapport à notre version très simple des flowcharts, mais rien d'insurmontable.

# LE "HELLO, WAD!" EN FLOWGORITHM

Lancer le programme

Voir le code

Instruction de sortie

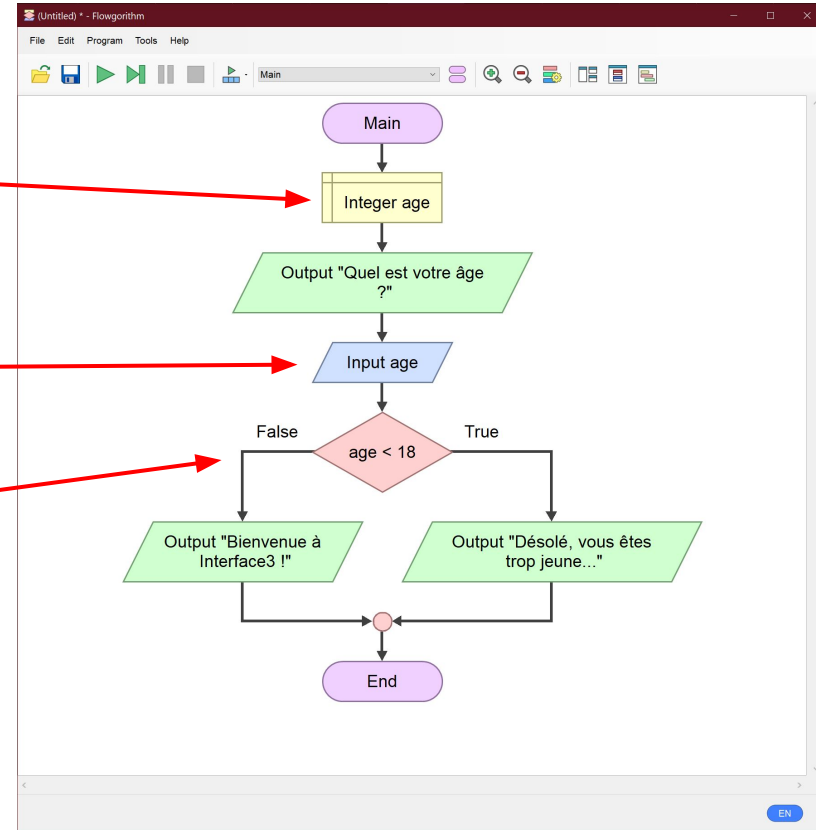


# UN EXERCICE PLUS COMPLEXE

Déclarer une "variable" (un mot représentant une valeur)

Instruction d'entrée

Décision  
(True  $\leftrightarrow$  Oui, False  $\leftrightarrow$  Non)



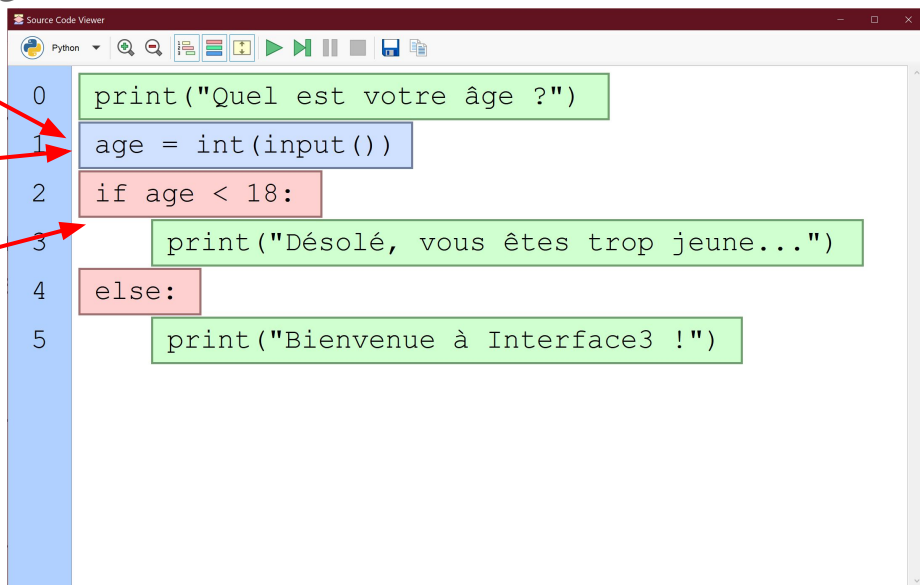
# UN EXERCICE PLUS COMPLEXE (EN PYTHON)

Déclarer une "variable" (un mot représentant une valeur)

Instruction d'entrée

Décision

(If <-> Si, Else <-> Sinon)



```
0 print("Quel est votre âge ?")
1 age = int(input())
2 if age < 18:
3     print("Désolé, vous êtes trop jeune...")
4 else:
5     print("Bienvenue à Interface3 !")
```