Cours 1 : Premiers pas en ingénierie informatique

POLYTECH MARSEILLE Aix-Marseille Université

Ing-info — Ingénierie informatique

Christophe Gonzales

Plan du cours n° 1

Objectifs du module

Objectif principal

Introduction ≪ ludique ≫ à la programmation.

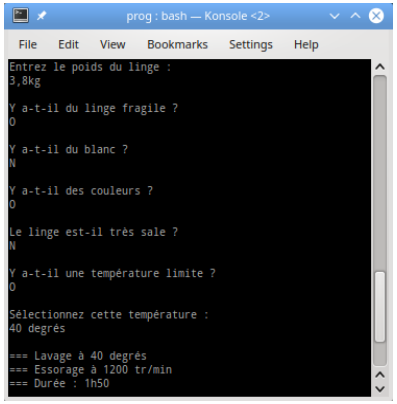
Compétences attendues

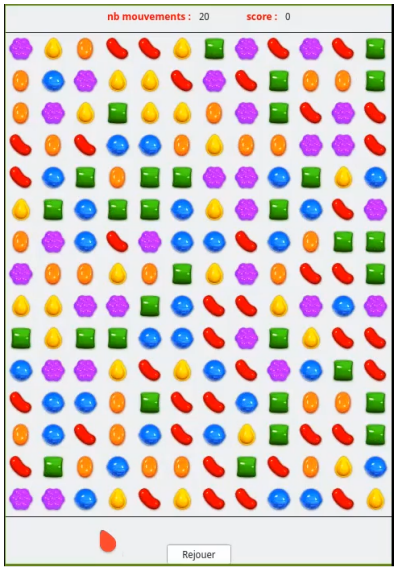
- Comprendre ce qu'est un programme
- Savoir écrire de petits programmes
- Acquérir de bonnes habitudes de programmation
- Savoir débugguer un programme

⚠ Accent mis sur la mise en œuvre pratique

Déroulement du module

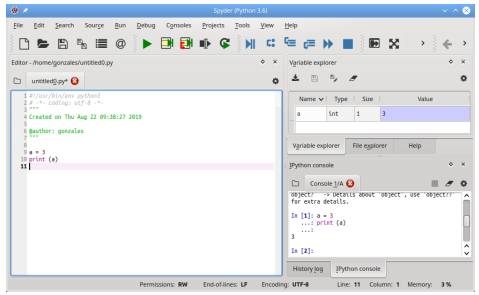
- 4 cours
- 6 TP $\Longrightarrow$ un projet

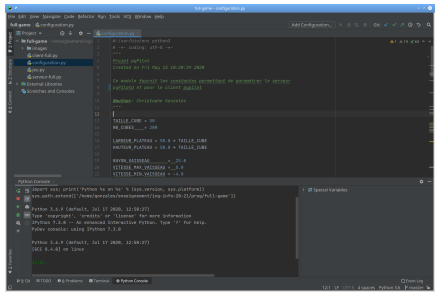




TPs en pratique (1/2)

- 2 éditeurs : spyder et pyCharm





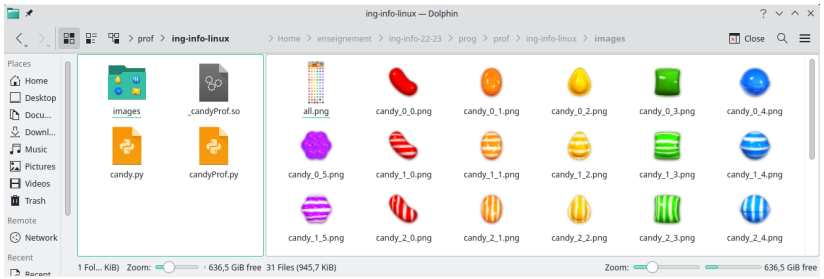
- Fichier source : `nom.py`
- Nom de l'interpréteur : `python3` ou `python`
  $\Longrightarrow$ exécution d'un programme : `python3 nom.py`
- Conventions graphiques pour la suite :

```
Code python
```

```
Résultat de l'exécution
```


TPs en pratique (2/2)

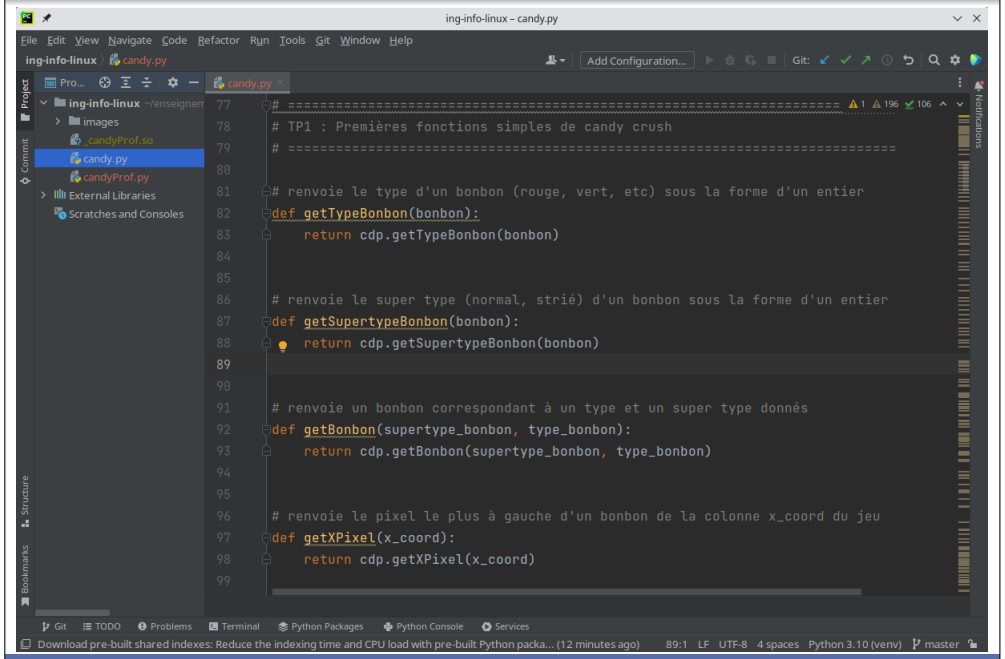
- TP n°1 : installer les fichiers



- Après installation, le jeu (candy.py) fonctionne
- Principe : réécrire le jeu

⚠ Toujours travailler dans le fichier candy.py

Développement de Candy Crush en TP



Évaluation

- Aucun contrôle continu (CC)
- Note finale = examen final
- Aucun document autorisé à l'examen

2 Généralités sur la programmation

Un programme ?

Ensemble d'instructions exécutées par un ou des processeur(s)

Programme C	assembleur	code machine
`#include <stdio.h>`	`printf@plt :`	
	`jmp QWORD PTR [rip+0x200bf2]`	`FF 25 F2 0B 20 00`
`int main () {`	`push 0x0`	`68 00 00 00 00`
`  printf ("hello") ;`	`jmp 400410 <.plt>`	`E9 E0 FF FF FF`
`}`	`main :`	
	`sub rsp,0x8`	`48 83 EC 08`
programme source	`mov edi,0x4005e4`	`BF E4 05 40 00`
	`xor eax,eax`	`31 C0`
	`call 400420 <printf@plt>`	`E8 E0 FF FF FF`
	`xor eax,eax`	`31 C0`
	`add rsp,0x8`	`48 83 C4 08`
	`ret`	`C3`
	`nop WORD PTR [rax+rax*1+0x0]`	`66 0F 1F 84 00 00`

Haut niveau → Bas niveau

Du source à l'exécutable

Programme source

- Ensemble d'instructions à exécuter
- Langage de haut niveau compréhensible par des humains
- Traduit *in fine* en langage machine pour être exécuté

Mécanismes de traduction en langage machine

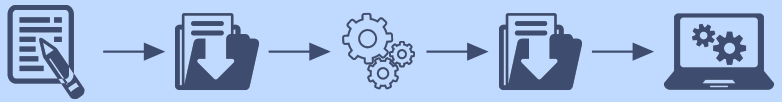
- Compilateur
- Interpréteur
- Interpréteur de bytecode

⚠ Les mécanismes dépendent du langage de programmation (source) !

Les compilateurs

Compilateur

- Processus :



- Création un fichier exécutable (code machine)
- Exécution du programme = exécution du fichier exécutable

avantages	inconvénients
▶ exécution rapide	▶ exécutable non portable (linux,MacOS,windows)
▶ vérification avant exécution	▶ les compilations peuvent être lentes

- Langages : C, C++, COBOL, Pascal, *etc.*

Les interpréteurs

Interpréteur

- Processus :

- Pas de création de fichier exécutable
- L'interpréteur traduit le *fichier source* en langage machine **et** exécute en même temps ce qu'il traduit

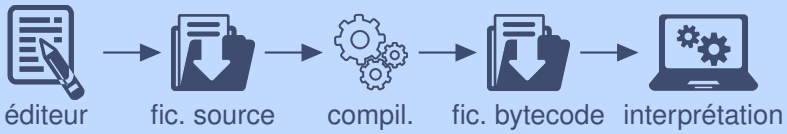
avantages	inconvénients
▶ programme portable	▶ exécution plus lente
▶ flexible (typage dynamique)	▶ pas de vérification avant exécution

- Langages : python, JavaScript, PHP, *etc.*

Les interpréteurs de bytecode

Interpréteur de bytecode

- Processus :



- Compil. en un langage indep. de la machine (bytecode)
- Exec. du programme = Exec. d'un *interpréteur de bytecode*
- L'interpréteur traduit le *bytecode* en langage machine **et** exécute en même temps ce qu'il traduit

avantages	inconvénients
▶ programme portable	▶ les compilations peuvent être lentes
▶ exécution assez rapide	

- Langages : Java, Ocaml, python, *etc.*

Choix du langage de programmation

Paradigmes de programmation

Différentes manières de « penser » le code source :

- Programmation impérative (C, JavaScript, COBOL, *etc.*) : instructions $\Longrightarrow$ changent l'état de la mémoire de l'ordi
- Programmation fonctionnelle (Caml, Scheme, *etc.*) : résultat de l'application de fonctions « mathématiques »
- Programmation objet (C++, Java, *etc.*) : objet = « concept », avec des propriétés
- Programmation par événement (librairies Qt, Gtk, *etc.*) : associations d'instructions à des événements (clic souris, redimensionnement de fenêtre, *etc.*)

- Projet : graphique : événements + objets (Qt)
  jeu : prog impérative

3 Programmation de Candy Crush

Structuration d'un programme

Structuration : approche hiérarchique

▶ **Candy crush :**

```
1  créer la fenêtre de jeu
2  initialiser le jeu
3  tant que le jeu n'est pas terminé faire
4    attendre les actions du joueur
5    faire les mises à jour du jeu et affichages
6  fait
```


Structuration (détails des étapes 4 et 5)

▶ **Actions du joueur :**

- ▶ Appuyer sur un bouton de souris
- ▶ Déplacer la souris
- ▶ Relâcher un bouton de souris

```
1  créer la fenêtre de jeu
2  initialiser le jeu
3  tant que le jeu n'est pas terminé faire
4    si le joueur clique sur un bonbon alors
5      griser le bonbon
6    sinon si le joueur relâche le bonbon alors
7      calculer le déplacement du bonbon
8      si le déplacement est invalide alors
9        animation d'un échange avorté de bonbons
10     sinon
11       mettre à jour le jeu
12     finsi
13   finsi
14 fait
```


Structuration (détails de l'étape 11)

```
1  créer la fenêtre de jeu
2  initialiser le jeu
3  tant que le jeu n'est pas terminé faire
4    si le joueur clique sur un bonbon alors
5      griser le bonbon
6    sinon si le joueur relâche le bonbon alors
7      calculer le déplacement du bonbon
8      si le déplacement est invalide alors
9        animation d'un échange avorté de bonbons
10     sinon
11       tant que il existe des match3 faire
12         supprimer les bonbons des match3
13         faire descendre les bonbons situés au dessus
14         rajouter des bonbons dans les cases vides
15         calculer les nouveaux match3 possibles
16       fait
17     finsi
18   finsi
19 fait
```


Structuration (détails de l'étape 2)

▶ **Initialisation :**

- ▶ remplir l'espace de jeu de bonbons
- ▶ de telle sorte qu'il n'y ait aucun match3

▶ **Déterminer s'il y a un match3 :**

- ▶ déterminer s'il y a un match3 horizontal
- ▶ déterminer s'il y a un match3 vertical

TP n°2 !

▶ **Algorithme d'initialisation :**

```
1  tant que l'espace de jeu n'est pas rempli faire
2    bonbon_choisi ← False
3    tant que bonbon_choisi = False faire
4      choisir un bonbon au hasard
5      si il ne fait pas partie d'un match3 alors
6        bonbon_choisi ← True
7      finsi
8    fait
9  fait
```

Structuration du programme

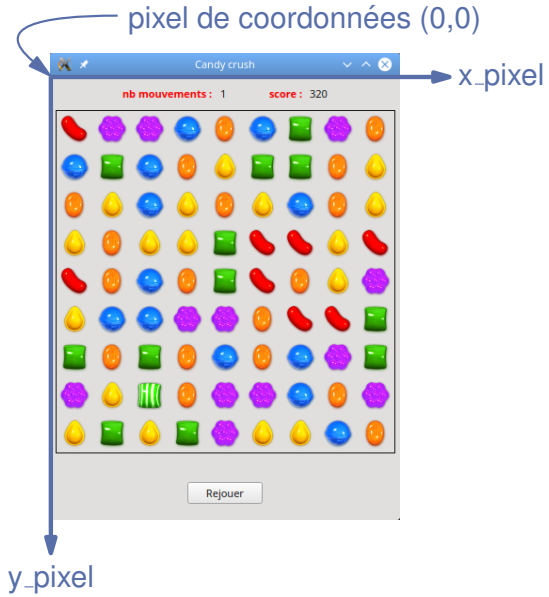
Conclusion

- Structuration hiérarchique
- Haut de la hiérarchie : le programme = quelques idées
- Tant qu'une idée est trop compliquée à programmer : affiner/décomposer celle-ci
  $\Longrightarrow$ on descend dans la hiérarchie
- En bas de la hiérarchie : code de chacune des fonctions

- ***Avantages de l'approche :***
  - Chaque étape est facile à réaliser
  - Permet d'écrire des programmes très complexes
  - Avant de coder une fonction, on sait précisément ce qu'elle doit faire

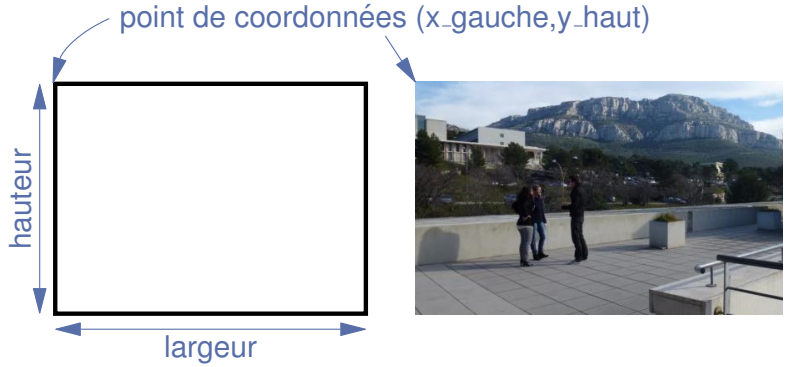
4 Affichages graphiques : pyQt5

Fenêtre et pixels

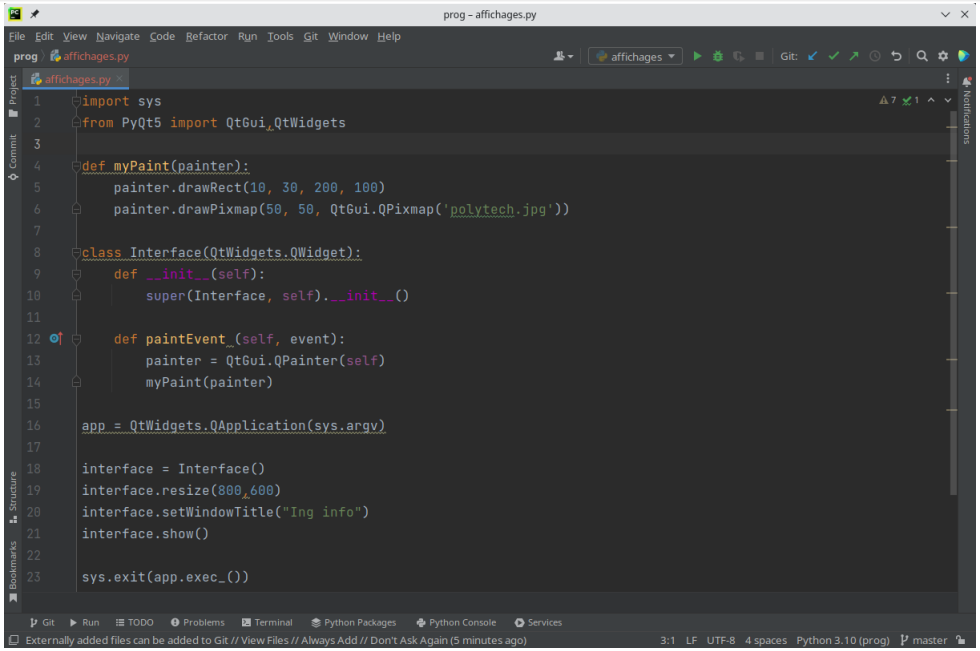


Joli dessin (Qt drawing)

- Affichages $\Longrightarrow$ utilisation de Qt (pyQt5)
- Fenêtre $\Longrightarrow$ Widget Qt $\Longrightarrow$ Painter
  - painter.drawRect(x_gauche, y_haut, largeur, hauteur)
  - painter.drawPixmap(x_gauche, y_haut, image)

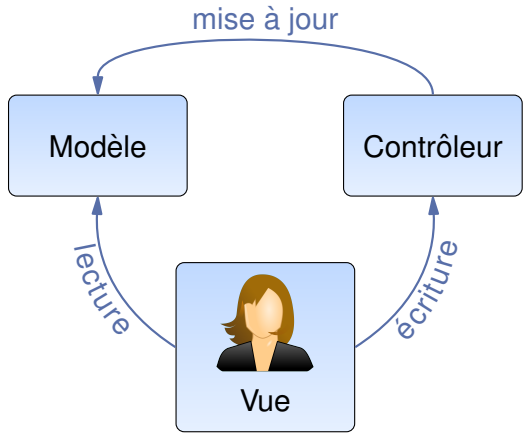


Affichages en pratique



5 Modèle-Vue-Contrôleur

« design pattern » MVC



- **Modèle :** contient les données du programme
- **Vue :** présentation des données à l'utilisateur
- **Contrôleur :** prise en compte des actions de l'utilisateur

Le modèle de Candy Crush (1/3)

Vue

1	1	4	3	2	1	5	3	0
0	3	5	3	0	5	1	5	0
1	5	0	2	2	4	4	0	4
5	1	3	3	5	0	4	0	5
4	3	0	3	5	5	3	4	5
4	0	1	2	0	2	4	4	1
0	5	2	5	3	5	4	5	4
1	5	2	23	4	1	0	5	4
2	1	3	0	3	2	1	0	5

Modèle

Modèle : espace de jeu = tableau de nombres

Le modèle de Candy Crush (2/3)

`tab` =

	0	1	2	3
0	*A*	*B*	*C*	*D*
1	*E*	*F*	*G*	*H*
2	*I*	*J*	*K*	*L*

- `tab` = tableau 2D
- Accès aux éléments : `tab[indice1,indice2]`

⚠ Accès tableau 2D : indice1 = ligne, indice2 = colonne

⚠ Les indices débutent à 0 !!!

- **Exemple :** `tab[1,2] = G`

$\Longrightarrow$ si `x` = abscisse, `y` = ordonnée du bonbon en (x, y) alors bonbon = `tab[y,x]` !!!

Le modèle de Candy Crush (3/3)

Modèle : espace de jeu = tableau **transposé** de nombres

$\Longrightarrow$ bonbon en (x, y) = nombre dans `tab[x,y]`

⚠ Transposée

1	0	1	5	4	4	0	1	2
1	3	5	1	3	0	5	5	1
4	5	0	3	0	1	2	2	3
3	3	2	3	3	2	5	23	0
2	0	2	5	5	0	3	4	3
1	5	4	0	5	2	5	1	2
5	1	4	4	3	4	4	0	1
3	5	0	0	4	4	5	5	0
0	0	4	5	5	1	4	4	5

- **Debug :** affichage de tab transposé : `candyPrintTableau(tab)`

Encodage des bonbons dans le modèle

Modèle : bonbon = entier

- Type des bonbons :

0 1 2 3 4 5

- Supertype des bonbons :

0 1 2

- Bonbon = Type + 10 $\times$ Supertype

$\Longrightarrow$  = 13

conséquences du « pattern » MVC

Modèle :
- classe `Jeu`
- fonctions sans affichage $\Longrightarrow$ méthodes de `Jeu`

Vue :
- classe `InterfaceJeu`
- méthodes « `affiche...` »
- la fonction `main` qui construit la fenêtre de jeu

Contrôleur :
- méthodes `mousePressEvent` et `mouseReleaseEvent` de la classe `InterfaceJeu`

6 Premiers pas en python

1er programme

- Instruction pour afficher « hello Polytech » :

```
print("hello Polytech")
```

```
hello Polytech
```

- Plusieurs instructions :

```
print("hello Polytech")
print("hello Marseille")
```

```
hello Polytech
hello Marseille
```

une seule instruction par ligne !

Manipulation de chaîne de caractères

Chaîne de caractères

- texte entouré de guillemets (")
  Exemple : `"toto"`
- caractère " dans une chaîne entourée de guillemets :
  utiliser `\"`. Exemple : `"toto\"titi"`
- caractère \ : utiliser `\\`. Exemple : `"toto\\titi"`
- autres caractères spéciaux utiles :
  `\n` : passer à la ligne suivante de l'écran
  `\t` : tabulation

```
print("12\tHello\nX\tPolytech\\Marseille")
```

```
12  hello
X   Polytech\Marseille
```


Vers un 2ème programme (1/2)

Problème : afficher un texte saisi au clavier par l'utilisateur ?

$\Longrightarrow$ sauvegarder le texte saisi dans la mémoire de l'ordinateur !

$\Longrightarrow$ utilisation de variables

Variable

- Variable = espace mémoire utilisé pour contenir des infos
- Toute variable est caractérisée par :
  - un **nom** : avec lequel on la manipule
  - une **valeur** : qui est l'info stockée
  - un **type** : qui définit comment stocker cette info en mémoire

Manipulation de variables

Que faire avec des variables

1. créer de nouvelles variables
2. leur affecter une valeur
3. modifier leur valeur
4. accéder à leur valeur

1 Créations, 2 affectations, 3 modifications

- 1 seule instruction : `nom_variable = valeur`
- Si variable non créée : opérations 1 et 2, Sinon : opération 3
- 2 et 3 : valeur est évaluée et *ensuite* affectée à la variable

⚠ rappel : 1 instruction par ligne de code

4 Accéder à la valeur d'une variable

- indiquer son nom (excepté à gauche d'un signe =)
- peut être utilisé dans une instruction complexe (`print(nom)`)

Exemples de variables (1/2)

- Programme Python :

```
1 a = "toto"
2 print(a)
3 a = "titi"
4 print(a)
```

- Effets des instructions :
  1. `a` n'existe pas encore $\Longrightarrow$ ▶ création de la variable `a` ▶ affectation de `"toto"` à `a`
  2. pas de « = » $\Longrightarrow$ on accède à la valeur de `a` $\Longrightarrow$ équivalent à `print("toto")`
  3. « = » mais `a` existe $\Longrightarrow$ modification de sa valeur $\Longrightarrow$ maintenant, `a` vaut `"titi"`
  4. équivalent à `print("titi")`

⚠ Accéder au contenu d'une variable : pas de guillemets

Exemples de variables (2/2)

- Programme Python :

```
1 a = "Marseille"
2 a = "hello"
3 b = a
4 print(b)
5 a = b
```

- Effets des instructions :
  1. Création de `a`. Maintenant `a` vaut `"Marseille"`
  2. Modification de `a` : `a` vaut `"hello"`
  3. `nom = valeur` $\Longrightarrow$ ▶ on crée `b` ▶ on évalue `a` : valeur = `"hello"` ▶ on affecte `"hello"` à `b`
  4. Affichage de la valeur de `b` : `"hello"`
  5. Modification de `a` : ▶ on évalue `b` : valeur = `"hello"` ▶ on modifie la valeur de `a` : `a` vaut `"hello"`

Noms des variables

Règles de nommage

- Nom débute par une lettre ou un underscore (_), suivi de lettres, de nombres ou d'underscores
  Exemples : `ma_variable`, `X33`
- Python différencie majuscules et minuscules : `X33` $\neq$ `x33`

Conseils pour nommer vos variables :

- **Important que les noms soient lisibles :**
  `masupervar`, `maSuperVar` ou `ma_super_var` ?
- **Important que les noms décrivent ce qu'ils contiennent :**
  `masupervar` ou `saisie_utilisateur` ?
- **Noms à éviter absolument :**
  L minuscule : `l` ressemble au nombre `1`
  O majuscule : `O` ressemble au nombre `0`
  $\Longrightarrow$ très difficile de débugguer

Fonctions mystères...

Quels sont les buts des fonctions ci-dessous ?

```
def a (b, c):
  d,e = b.shape
  f = np.full(d,0)
  for g in range(d):
      for h in range(e):
          f[g] += b[g,h] * c[h]
  return f
```

```
def prod_matrice_vecteur (mat, vect):
  dim1,dim_vect = mat.shape
  vect_produit = np.full(dim1,0)
  for i in range(dim1):
      for j in range(dim_vect):
          vect_produit[i] += mat[i,j] * vect[j]
  return vect_produit
```

$\Longrightarrow$ noms des variables et fonctions importants pour la lisibilité !

Vers un 2ème programme (2/2)

Problème : afficher un texte saisi au clavier par l'utilisateur ?

```
texte_saisi = input("saisir un texte : ")
print(texte_saisi)
```

```
saisir un texte : hello Marseille
hello Marseille
```

Syntaxe pour demander une saisie au clavier :
nom_variable = input (message demandant la saisie)