



Éléments de programmation

OÙ ON NE PARLE PAS SEULEMENT DU LANGAGE DE PROGRAMMATION...

La programmation comme spécification

- ▶ Indépendamment du langage de programmation choisi (C, C++, Python, Java, JavaScript, etc...) le but est **d'expliquer** à un ordinateur ce que l'on voudrait qu'il effectue **automatiquement**.
- ▶ Nous on a en tête une **vision** de ce que l'on voudrait obtenir, et les actions à effectuer:
 - ▶ Piloter un moteur
 - ▶ Effectuer des mesures physiques
 - ▶ Enseigner un robot
 - ▶ Actionner des actionneurs
 - ▶ ...

La vraie vie est différente de l'ordinateur

- ▶ Donc on va effectuer une **traduction** de ce que l'on a en tête vers un langage qu'un ordinateur comprend
- ▶ On doit expliquer:
 - ▶ Les concepts réels (les actionneurs, les commandes, capteurs, les valeurs)
 - ▶ Les séquences (les algorithmes, les règles, les conditions)
 - ▶ Les contextes, les objectifs,
 - ▶ Les interfaces avec les opérateurs (humains ou non humains)

La modélisation

- ▶ La vraie vie avec ses contraintes, ses réalités ne correspond pas (*jamais*) exactement à ce que le matériel (les ordinateurs, les capteurs, les gammes de valeur accessibles, les capacités matérielles des actionneurs) accepte !!!
- ▶ On doit sélectionner un sous-ensemble **réalisable** de nos objectifs
- ▶ Cette sélection est la **modélisation**
- ▶ Les langages de programmation **traduisent** ce modèle en quelque chose de réalisable par le **système** considéré
- ▶ Le système regroupe l'ensemble des éléments actifs chargés de réaliser nos objectifs (ordinateur, base de données, capteurs, écrans, etc...)

La conception

- ▶ On doit commencer par écrire en langage **naturel** notre système et l'objectif que l'on **souhaite** obtenir.
- ▶ Exemples
 - ▶ *Le **véhicule** est **motorisé** par **deux moteurs**. La **commande** permettra de **sélectionner** la **vitesse** du **véhicule**, les **directions**, la **détection** des **obstacles**, la **réactivité automatique** à la **détection** d'obstacles.*
- ▶ On identifie alors les éléments de cette description
 - ▶ Les **substantifs** qui correspondent aux objets manipulés
 - ▶ Les **verbes** qui correspondent aux actions
 - ▶ Les **qualificatifs**, les **conditions**, qui caractérisent les algorithmes

Le langage de programmation

- ▶ Les substantifs vont correspondre aux **objets**, les **valeurs**, les **variables**, les **constantes**
- ▶ Les actions vont correspondre aux **fonctions**
- ▶ Les conditions correspondent aux structures **logiques** et **décisionnelles**
- ▶ Ces termes trouvent leur traduction quel que soit le langage de programmation choisi, avec plus ou moins de flexibilité, de richesse dans les moyens d'exprimer les détails, les précisions

Des règles liées à la qualité

- ▶ Quand on écrit du code, on change peu ce que l'on écrit (si on a bien spécifié !!) mais par contre on va relire plein de fois ce que l'on a écrit:
 - ▶ Pour expliquer aux collègues
 - ▶ Pour comprendre soi-même notre code par rapport à ce que l'on écrit plus tard
 - ▶ Pour comprendre ce que l'on a écrit un mois plus tard
 - ▶ Pour comprendre pourquoi parfois cela ne marche pas
- ▶ Donc... il faut *toujours* écrire les mots complets, clairs, signifiants, précis
 - ▶ Les abréviations sont interdites
 - ▶ Même si ça prend un peu plus de temps à écrire

règles... suite

- ▶ Préférez de sections de code **courtes**
- ▶ Une section de code en double est toujours en trop => **créez des fonctions**
- ▶ Les commentaires !!! *Aaah les commentaires*
 - ▶ Certains pourraient dire qu'un bon programme bien écrit n'a pas besoin de commentaires ???
 - ▶ C'est **presque** vrai : la vrai défaut des commentaires c'est que très souvent ils ne sont pas **à jour** par rapport aux évolutions du code
 - ▶ Ils ne faut jamais mettre un commentaire qui est en double par rapport à la ligne de code.
 - ▶ Un commentaire sert à **expliquer** pourquoi on a choisi tel ou tel algorithme.

Langage interprété ou compilé ?

- ▶ Python, JavaScript sont **interprétés**
 - ▶ Avantages: très **facile** à faire évoluer
 - ▶ Défaut: non **typé**, c'est-à-dire que les variables ne sont pas vérifiées vis-à-vis des valeurs utilisées
- ▶ C, C++, Java, Fortran sont **compilés**
 - ▶ Avantages et inconvénients inverses des langages interprétés
 - ▶ Beaucoup plus **fiables** à cause du typage des variables
 - ▶ Nécessitent une phase de compilation avant l'exécution
 - ▶ Plus **rapides** à l'exécution

Python et autres docs

- ▶ La documentation officielle est la plus efficace

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

- ▶ Micropython

<https://docs.micropython.org/en/latest/esp32/tutorial/index.html>

- ▶ ESP32 series

<https://www.espressif.com/en/products/modules>

- ▶ Micropython et ESP-NOW

<https://docs.micropython.org/en/latest/library/espnw.html#module-espnw>

Types

- ▶ Valeurs numériques
 - ▶ Entiers, flottants, doubles
- ▶ booléens
- ▶ Chaînes de caractères
- ▶ Tableaux
- ▶ Objets
- ▶ Pointeurs

```
valeur_entière = 1244
valeur_flottante = 45.777
valeur_double = 23.666
valeur_bouléenne = Vrai
valeur_chaîne = "bonjour tout le monde"
valeur_tableau = [1, 4, 9]
valeur_objet = {1244, 45.777, Vrai, "bonjour", [1, 4, 9]}
```

Les guillemets **simples**
ou **doubles** sont
équivalentes

Print

```
print("bonjour tout le monde")  
  
x = 12.33  
y = 44  
t = [1, 2, 3]  
  
print("x=", x, "y=", y)  
  
print(f"x={x} y={y} ")  
  
print(f"le tableau t vaut {t} ")
```

Sortie minimale
à l'écran

Enchaîner des sorties
individuelles

Formatter l'impression

Les fonctions

```
def piloter(arg1, arg2, ...):  
    code  
    code
```

```
def calculer(arg1, arg2, ...):  
    code  
    code  
    return resultat
```

Structure générale:

```
def nom (arguments) :  
    <indentation>Bloc de code  
    <indentation>Bloc de code  
    ...
```

ESP32 C3 mini

LED interne

