ALGO 01 Instructions de base et types

▸ Compétences

- Comprendre la notion d'algorithme, de `pseudo-code` et de langage python.
- Déclarer et affecter des variables en utilisant les types de base.
- Utiliser les opérateurs arithmétiques.
- Afficher des résultats avec `print()` en utilisant notamment les *f-strings*.
- Réaliser des saisies utilisateur avec `input()` et transtyper si nécessaire.

▸ Fil rouge *coloration drone*

Un système embarqué sur drone collecte en temps réel des données de vol : altitude (entier), vitesse (réel), identifiant du pilote (chaîne de caractères), statut de décollage (booléen).
Comment stocker, manipuler et afficher ces informations dans un premier programme informatique ?

I. Introduction

I.1. Algorithmes

▸ Intro

Le mot « algorithme » vient du nom du mathématicien et astronome Al Khuwarizmi (latinisé au Moyen Âge en *Algoritmi*), qui, au IX^{e} siècle écrivit le premier ouvrage systématique sur la solution des équations linéaires et quadratiques.

▸ Définition 1.1

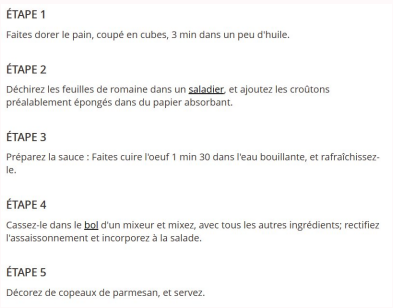
Un algorithme est une suite finie d'opérations ou d'instructions à appliquer dans un ordre déterminé à un nombre fini de données pour résoudre un problème en un nombre fini d'étapes.

▸ Exemples 1.1

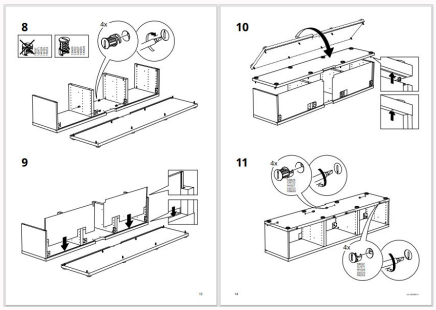
On peut trouver des exemples d'algorithmes dans la vie courante :

- recette de cuisine,
- notice de montage de meuble.

▸ Illustrations


ÉTAPE 1
Faites dorer le pain, coupé en cubes, 3 min dans un peu d'huile.
ÉTAPE 2
Déchirez les feuilles de romaine dans un saladier, et ajoutez les croûtons préalablement épongés dans du papier absorbant.
ÉTAPE 3
Préparez la sauce : Faites cuire l'oeuf 1 min 30 dans l'eau bouillante, et rafraîchissez-le.
ÉTAPE 4
Cassez-le dans le bol d'un mixeur et mixez, avec tous les autres ingrédients; rectifiez l'assaisonnement et incorporez à la salade.
ÉTAPE 5
Décorez de copeaux de parmesan, et servez.

Recette du site `Marmiton.org`©



Montage d'un meuble `IKEA`©

▸ Remarque 1.1

Les premiers algorithmes mathématiques sont apparus bien avant les premiers ordinateurs et étaient exécutés à la main. Dans ce cours nous allons apprendre à écrire les algorithmes en *langage de programmation* de manière à ce que l'ordinateur les exécute

▸ Définition 1.2

Un algorithme écrit en langage de programmation s'appelle un programme.

I.2. Langages informatiques

▸ **Intro**

Le langage de l'ordinateur est le langage **binaire**, qui est composé uniquement de 0 et de 1.

▸ **Exemple 1.2**

On peut par exemple traduire le mot « Bonjour » (codage `UTF-8`) par le code binaire :
« 01000010011011110110111001101010011011110111010101110010 ».

▸ **Idée**

Pour éviter de devoir écrire nos algorithmes en binaire, on a recours à ce que l'on appelle un *langage de programmation* (comme par exemple Python, C, C++, etc).
Un programme, appelé *interpréteur* ou *compilateur* selon les cas, se charge ensuite de traduire ces algorithmes en langage binaire compréhensible par la machine.

▸ **Cadre**

Durant l'année, nous allons utiliser deux langages pour nos algorithmes :

- le ALGO `pseudo-code` : un langage naturel, exprimé en français, que nous utiliserons pour élaborer nos algorithmes sur papier ;
- ALGO python : un langage informatique multi-plateforme placé sous licence libre, créé en 1989 par le programmateur Guido van Rossum.

▸ **Point histoire**

Guido van Rossum (1956 –, 🇳🇱) est un développeur connu pour être le créateur et leader du projet du langage de programmation ALGO python. Il est également l'auteur du navigateur web Grail entièrement programmé en ALGO python ; il n'a pas été actualisé depuis 1999. Le nom fait référence au film *Monty Python and the Holy Grail*.

II. Environnement de développement

II.1. IDE

▸ **Cadre**

Durant l'année, nous allons utiliser la version 3 de ALGO python. Pour faciliter l'écriture d'algorithmes, nous allons travailler dans un environnement de développement intégré (en anglais IDE, pour Integrated Development Environment).

▸ **Logiciel**

Au cours de l'année, nous allons utiliser le logiciel Thonny.
Thonny est un environnement de développement léger, muni d'un débogueur permettant de visualiser le contenu de chaque variable. Il est portable et peut s'installer sans droits d'administration. Il est disponible à l'adresse : `https://thonny.org/`.

▸ **Remarques**

À noter qu'il existe d'autres possibilités :

- Idle, un IDE installé avec toute distribution ALGO python (`https://www.python.org/downloads/`) ;
- Spyder, un peu lourd et lent mais assez convivial, qui s'installe avec le pack Anaconda (`http://continuum.io/downloads`) ou avec le pack Miniconda (`https://conda.io/miniconda.html`) puis taper dans la console conda : SHELL `conda install spyder` ;
- Pythontutor qui est un environnement de développement en ligne qui permet de visualiser le contenu de la mémoire au fur et à mesure (`http://www.pythontutor.com/visualize.html#mode=edit`).

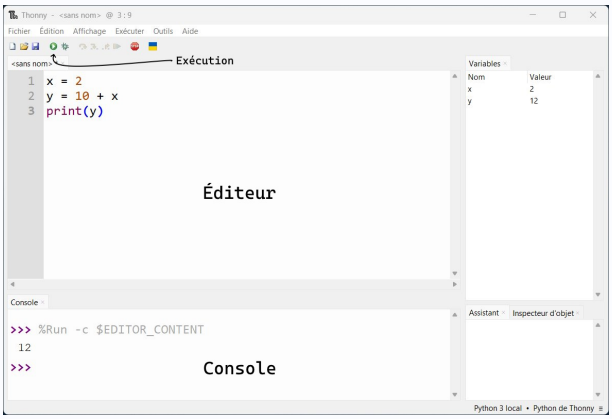
II.2. Présentation de Thonny

▸ Logiciel

Les environnements de développement sont essentiellement constitués de deux *blocs* : l'*éditeur de texte* et la *console de l'interpréteur*.

L'éditeur de texte

L'éditeur de texte est la zone du programmeur. En effet, les programmes sont tapés dans des fichiers textes, que l'on peut sauvegarder, modifier et distribuer à loisir.
Dans Thonny, on peut exécuter le programme contenu dans le fichier texte à l'aide du bouton d'exécution.



La console de l'interpréteur

La console est la zone de l'utilisateur. Elle se repère aux trois chevrons (PY >>>) qu'elle contient.
En général, on se sert de cette zone essentiellement pour visualiser le résultat de l'exécution de nos programmes, et pour interagir avec le programme (saisie). Cependant, on peut aussi se servir de la console pour tester l'effet d'une commande.

▸ Cadre 2.1

Tous les exemples du module sont à taper dans l'éditeur de texte, sauf si il est explicitement écrit de tester les exemples dans la console.

▸ Remarques 2.1

- Le symbole PY >>> dans la console s'appelle une *invite de commande*, il signifie que l'interpréteur est prêt à recevoir une instruction.
- Les fichiers créés par l'éditeur de texte portent l'extension SHELL .py, et il est important de sauvegarder son travail régulièrement.
- Pour ajouter des commentaires, on utilise le symbole PY #; le texte suivant le symbole PY # (sur une seule ligne) n'est pas pris en compte par l'interpréteur.
- En ALGO python, les espaces en début de ligne (on parle d'*indentation*) ont un sens. Sauf précision contraire, toutes les instructions doivent donc être alignées complètement à gauche.

▸ Logiciel

Dans Thonny, on peut mettre en *commentaire* une portion du texte en sélectionnant celle-ci, en allant dans le menu Édition > Commenter.
On peut également réaliser l'opération inverse en allant dans le menu Édition > Décommenter.

▸ Attention

Les résultats des instructions tapées dans la console sont affichés automatiquement.
Mais pour afficher le résultat d'une instruction tapée dans l'éditeur de texte, il faut utiliser PY `print`, qui est la procédure permettant d'afficher.

- dans la console, on peut taper PY `1+2`;
- dans l'éditeur de texte, on tape PY `print(1+2)`.

II.3. Règles de nommage des fichiers et de sauvegarde

▸ **Attention**

Les ordinateurs du lycée peuvent être réinitialisés à tout moment, toute donnée stockée sur l'un d'eux étant alors perdue. Il est recommandé de sauvegarder son travail :

- sur l'espace partagé du serveur ;
- sur une clé USB (attention toutefois avec les virus) ;
- sur un espace de stockage en ligne (drive, etc).

Dans tous les cas, tout travail effectué en classe doit être sauvegardé.

▸ **Remarque**

Par ailleurs, tout fichier rendu devra être nommé de la façon suivante : `nom_devoir_date.py`.
Par exemple, un élève dont le nom est Azerty, qui rend un fichier pour le DS1 le 15/09/2042 devra nommer son fichier `azerty_ds1_15_09_2042.py`.

De plus, pour une meilleure organisation, il est par ailleurs recommandé de créer un *dossier dédié*.

III. Variables

III.1. Ce qu'est une variable

▸ **Définition 3.1**

Une variable correspond à un emplacement dans la mémoire de l'ordinateur auquel on donne :

- **un nom** : qui doit commencer par une lettre et ne contenir que des lettres non accentuées, des chiffres et du caractère « _ » ;
- **une valeur** : qui est modifiable ;
- **un type** : entier, réel, etc, que l'on peut obtenir en python avec la fonction `type()`.

▸ **Remarques 3.1**

- python est sensible à la casse, autrement dit la variable `nom_de_variable` et la variable `Nom_De_Variable` ne sont pas les mêmes.
- Il est très important de donner des noms de variable qui permettent de retrouver ce que la variable contient. Par exemple si je veux créer une variable qui contient un âge, je ne l'appelle pas `toto`, ou `x`, mais `age`.

Les évaluations pourront tenir compte de choix cohérents pour les noms de variables.

III.2. Affichage

▸ **Algorithme 3.1**

Pour afficher la valeur d'une variable, une valeur ou du texte dans la console, on utilise :

Pseudo-Code
```
Afficher : <expression à afficher>
```

Code Python
```
print(<expression à afficher>)
```

▸ **Exemple 3.1**

En `pseudo-code` :

Pseudo-Code
```
a ← 10
Afficher : a, 5, "Bonjour"
```

En python :

Code Python
```
1 a = 10
2 print(a, 5, "Bonjour")
```

▸ Python 3.1

La console permet – par exemple – de vérifier le bon fonctionnement des commandes :

```
🐍 Début de la console Python 🐍
>>> a = 10
>>> print(a, 5, "Bonjour")
10 5 Bonjour
🐍 Fin de la console Python 🐍
```

▸ Remarques importantes

- Attention à bien distinguer une variable d'une chaîne de caractères :
  - la chaîne de caractères est un texte écrit entre guillemets et qui sera affiché tel quel ;
  - lorsqu'on met un nom de variable (sans guillemets), c'est la valeur de cette variable qui sera affichée.
- En python, les différents éléments à afficher sont mis entre parenthèses et séparés par des virgules, comme par exemple `print(42, "Bob", "1+1 =", 1+1)`.

III.3. Affectation d'une valeur à une variable

▸ Définition 3.2

L'**affectation** est une opération permettant de définir la valeur initiale d'une variable (on parle alors d'*initialisation*), puis de modifier la valeur de cette variable.

En pseudo-code, on note :

Pseudo-Code

```
nom_variable ← valeur
```

En python, on note :

Code Python

```
nom_variable = valeur
```

▸ Exemple 3.2

En pseudo-code :

Pseudo-Code

```
1 annee ← 2021
2 Afficher : annee
3 annee_prochaine ← annee + 1
4 Afficher : annee_prochaine
```

En python :

Code Python

```
1 annee = 2021
2 print(annee)
3 annee_prochaine = annee + 1
4 print(annee_prochaine)
```

▸ Python 3.2

La console permet – par exemple – de vérifier le bon fonctionnement des commandes :

```
🐍 Début de la console Python 🐍
>>> annee = 2021
>>> print(annee)
2021
>>> annee_prochaine = annee + 1
>>> print(annee_prochaine)
2022
🐍 Fin de la console Python 🐍
```

▸ Remarques 3.2

- On peut affecter à une variable la valeur d'une autre variable au moment de l'affectation.

```
Début de la console Python
>>> a = 42
>>> b = a
>>> a = 10
>>> print(a, b)
10 42
Fin de la console Python
```

- Une même variable peut apparaître à droite et à gauche de l'affectation.

```
Début de la console Python
>>> a = 10
>>> a = a * 4
>>> print(a)
40
Fin de la console Python
```

- On fera toujours très attention à mettre la variable dont on veut changer la valeur à gauche, et la valeur que l'on veut donner à cette variable à droite !

```
Début de la console Python
>>> 10 = a
  File "<input>", line 1
    10 = a
    ^^
SyntaxError: cannot assign to literal here. Maybe you meant '==' instead of
'='?
Fin de la console Python
```

▸ Définition 3.3

On appelle **incrémentation** l'opération (*très courante*) consistant à ajouter 1 ou une valeur entière fixe à une variable.

- Par exemple `a ← a + 1` en `pseudo-code`.
- Par exemple `a = a + 1` en `python`.

III.4. Saisie

▸ Définition 3.4

Pour rendre un programme *interactif*, on peut, pendant l'exécution de celui-ci, demander à l'utilisateur de saisir un texte ou une valeur grâce à une *instruction de saisie*.

Dès qu'arrive une instruction de saisie, le programme se met en pause. L'utilisateur doit alors saisir quelque chose dans la console et appuyer sur la touche ⏎ pour valider.
Pour expliciter ce qu'il faut saisir, une instruction de saisie est en général accompagnée par l'affichage d'un message explicatif.

La valeur saisie doit par ailleurs être stockée dans une variable de manière à pouvoir être réutilisée dans la suite du programme.

Pour demander de saisir une valeur et la stocker dans une variable nommée `a` (ou `a`), on utilise les instructions données ci-après.

En `python` il est important de comprendre que `input` stocke la saisie utilisateur dans une **chaîne de caractères**, il est donc nécessaire de réfléchir au type de la variable à saisir, et de proposer le cas échéant un *transtypage*, comme explicité ultérieurement.

▸ Algorithme 3.2

En `pseudo-code` :

Pseudo-Code

```
Afficher : <message>
Saisir a
```

En `python` :

– pour saisir un texte :

Code Python

```
1    a = input("message")
```

– pour saisir un nombre :

Code Python

```
1    a = eval(input("message"))
```

▸ Remarque 3.3

- Dans les instruction précédentes, le nom de variable `a` (ou `a`) doit bien sûr être remplacé par un nom de variable bien choisi.
- Dans les instruction précédentes, le `"message"` (ou `"message"`) doit être remplacé par un message expliquant à l'utilisateur ce qu'il doit saisir.

▸ Exemple 3.3

- En `pseudo-code` :

Pseudo-Code

```
Afficher : "Veuillez saisir votre prénom : "
Saisir : nom
Afficher : "Veuillez saisir votre âge : "
Saisir : age
Afficher : "Bonjour ", nom , ", vous avez ", age , " ans."
```

- En `python` :

Code Python

```
1 nom = input("Veuillez saisir votre prénom : ")
2 age = eval(input("Veuillez saisir votre âge : "))
3 #Affichage classique
4 print("Bonjour ", nom , ", vous avez ", age , " ans.")
5 #Affichage formaté
6 print(f"Bonjour {nom}, vous avez {age} ans.")
```

▸ Python 3.3

La console permet – par exemple – de vérifier le bon fonctionnement des commandes (les saisies ne sont pas ici présentées *comme en vrai*) :

```
🐍 Début de la console Python 🐍
>>> nom = "Maurice"
>>> age = 75
>>> #Affichage classique
>>> print("Bonjour ", nom , ", vous avez ", age , " ans.")
Bonjour  Maurice , vous avez  75  ans.
>>> #Affichage formaté
>>> print(f"Bonjour {nom}, vous avez {age} ans.")
Bonjour Maurice, vous avez 75 ans.
🐍 Fin de la console Python 🐍
```

▸ Remarque importante !

La valeur retournée par la fonction `input()` est toujours un texte (du type chaîne de caractère, `str` en python) et non un nombre !

💣 Or python ne gère pas les chaînes de caractères et les nombres de la même façon.

```
🐍 Début de la console Python 🐍
>>> age = "75"
>>> print(type(age))
<class 'str'>
>>> print(age * 2)
7575
🐍 Fin de la console Python 🐍
```

```
🐍 Début de la console Python 🐍
>>> age = 75
>>> print(type(age))
<class 'int'>
>>> print(age * 2)
150
🐍 Fin de la console Python 🐍
```

III.5. Opérations sur les variables, variables numériques

▸ Méthode 3.1

Sous python, le type de la variable est donné par le *type* de la valeur qu'on lui affecte.
On peut changer de type (on parle de `transtypage`) à l'aide de fonctions :

- Type **entier** en python : `int()`.
- Type **réel** en python : `float()` (Le séparateur décimal est un point !).
- Type **chaîne** en python : `str()`.
- Type **booléen** en python : `bool()`.

▸ Python 3.4

python permet d'effectuer les opérations classiques sur les nombres. En plus de celles-ci, deux opérations importantes sont à connaître :

- la **division entière**, qui donne la partie entière de la division (ou le quotient de la division euclidienne pour des entiers) ; elle s'obtient par le symbole `//` ;
- le *modulo*, qui est le reste de la division entière ; il s'obtient par le symbole `%`.

Le tableau ci-dessous récapitule les symboles permettant d'effectuer les différentes opérations en python :

addition	soustraction	multiplication	division	puissance	div. entière	modulo
`+`	`-`	`*`	`/`	`**`	`//`	`%`

```
🐍 Début de la console Python 🐍
>>> 1+1
2
>>> 10-3
7
>>> 6*7
42
>>> 12/4
3.0
>>> 10**3
1000
>>> 13//3  # retourne 4 car 13/3 vaut environ 4,33 dont la partie entière est
4
4
>>> 13%3   # retourne 1 car 13 = 3 * 4 + 1
1
🐍 Fin de la console Python 🐍
```

▸ Remarque 3.4

Attention, les calculs avec des nombres réels (flottants) donnent parfois des résultats approximatifs à cause de la façon dont sont stockés ces nombres ! Il faut essayer de privilégier les calculs entiers, et manier les réels avec une extrême prudence (et ce quelque soit le langage) !

🐍 Début de la console Python 🐍

```
>>> 0.1+0.1
0.2
>>> 0.1+0.2
0.30000000000000004
```

🐍 Fin de la console Python 🐍

IV. Exemple commenté

▸ Exemple : fiche de vol d'un drone

On souhaite écrire un programme qui :

- saisit les informations d'un drone (identifiant, altitude de croisière, niveau de batterie) ;
- calcule la distance maximale que le drone peut parcourir avant de devoir recharger ;
- détermine si le drone est autorisé à décoller (batterie supérieure à 20 %) ;
- affiche une fiche de vol complète.

</> Code Python

```python
# === Saisies ===
identifiant   = input("Identifiant du drone : ")               # str
altitude      = eval(input("Altitude de croisière (en m) : ")) # int
batterie      = eval(input("Niveau de batterie (en %) : "))    # float
# === Calculs ===
conso         = 0.1                  # float : consommation en % par mètre
autonomie_max = batterie // conso    # int   : distance max en mètres
autorisation  = (batterie > 20)      # bool  : décollage autorisé ?
# === Affichage ===
print("=== Fiche de vol ===")
print(f"Drone         : {identifiant}")
print(f"Altitude      : {altitude} m")
print(f"Batterie      : {batterie} %")
print(f"Autonomie max : {autonomie_max} m")
print(f"Décollage OK  : {autorisation}")
```

🐍 Début de la console Python 🐍

```
>>> identifiant   = "DRONE-42"
>>> altitude      = 80
>>> batterie      = 65.0
>>> conso         = 0.1
>>> autonomie_max = batterie // conso
>>> autorisation  = (batterie > 20)
>>> print("=== Fiche de vol ===")
=== Fiche de vol ===
>>> print(f"Drone         : {identifiant}")
Drone         : DRONE-42
>>> print(f"Altitude      : {altitude} m")
Altitude      : 80 m
>>> print(f"Batterie      : {batterie} %")
Batterie      : 65.0 %
>>> print(f"Autonomie max : {autonomie_max} m")
Autonomie max : 649.0 m
>>> print(f"Décollage OK  : {autorisation}")
Décollage OK  : True
```

🐍 Fin de la console Python 🐍

NB : ici les saisies dans la console (les `input()`) ont été remplacés par des affectations directes.