THÉMATIQUE 3

Programmation en Python

CHAPITRE 01

Un monde de variables

Programmation en Python	Chap 01	Un monde de variables

1.1 Activité : le dessin dicté

Objectifs

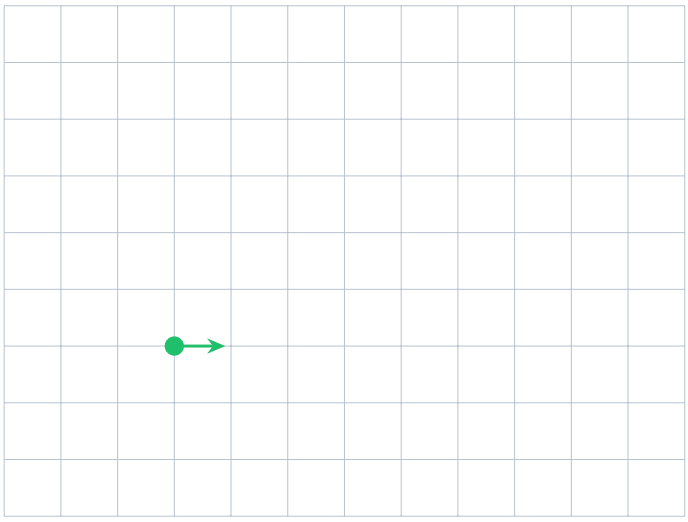
Faire reproduire un dessin à votre binôme, uniquement avec des instructions écrites.

Les règles du jeu

- Chacun reçoit une carte avec un **dessin secret**, différent de celui de son binôme. On ne montre pas sa carte !
- Le dessin commence au **point de départ**, et la **flèche** indique la direction dans laquelle on part.
- La personne qui suit les instructions les applique **à la lettre**, sans deviner, et **sans poser de question** : un ordinateur ne vous demande jamais ce que vous avez voulu dire.

Étape 1. Écrire en français les instructions qui permettent de reproduire votre dessin, à partir du point de départ.

Étape 2. Échanger vos polycopiés. Tracer sur la grille le dessin décrit par les instructions de votre binôme, en les suivant à la lettre.



Étape 3. Récupérer votre polycopié et comparer le dessin obtenu avec votre carte. Noter les instructions qui ont posé problème, et pourquoi.

Bilan de la classe

Qu'est-ce qui rend une instruction claire, pour qu'elle soit suivie exactement comme on le veut ?

1.2 Programmes et langages

Une machine ne devine rien

Dans la vie de tous les jours, on donne souvent des instructions vagues : "monte un peu", "fais un carré", "va à droite" (faut-il tourner, ou se décaler ?). En tant qu'humain, on comble les trous : on devine ce qu'on a voulu dire, ou on pose une question. Un ordinateur, lui, ne devine rien et ne pose pas de question : il fait exactement ce qu'on lui demande, ni plus ni moins. Pour lui parler, il faut donc un langage **précis**, dont chaque mot a un seul sens possible.

Définition 1 . Instruction

Un ordre élémentaire donné à l'ordinateur est appelé une **instruction**.

Définition 2 . Programme

Une suite d'instructions est appelée un **programme**. **Programmer**, c'est écrire un programme : la liste des tâches qu'un ordinateur doit réaliser.

Définition 3 . Langage de programmation

Un **langage de programmation** est un langage précis qui permet d'écrire des instructions compréhensibles par un ordinateur.

Définition 4 . Exécuter un programme

Exécuter un programme, c'est demander à l'ordinateur de lire et de réaliser chaque instruction du programme, l'une après l'autre, dans l'ordre.

En SNT, vous allez utiliser le langage **Python**, qui est très utile pour la majorité des usages. **Python** est dit "de haut niveau", non pas parce qu'il est difficile, mais parce qu'il est proche de notre langage, et loin des 0 et des 1 que comprend la machine (et que vous comprenez maintenant !).

Remarque 1. Dans ce cours, on rencontrera deux façons d'écrire du code.

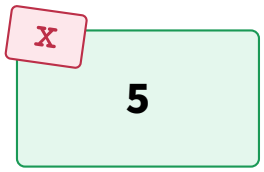
- Un bloc de lignes numérotées est un **programme** : Python exécute toutes ses instructions, dans l'ordre.
- La notation >>> désigne l'utilisation de Python en **ligne de commande** : on tape une instruction après >>>, et Python affiche le résultat à la ligne suivante. Vous la découvrirez pendant votre premier TP.

1.3 Les variables

Un ordinateur peut être vu comme une machine à calculer et à retenir des informations. Une des premières notions à apprendre pour programmer est donc la mémoire. Il est bien utile de garder des informations en mémoire quand on programme : votre nombre de points de vie dans un jeu vidéo, le pseudo de votre personnage, etc. Pour cela, on utilise des variables.

Définition 5 . Variable

Une **variable** est l'association de trois éléments : un **espace mémoire**, un **nom**, et une **valeur**. On peut la voir comme une boîte (l'espace mémoire), qui porte une étiquette (le nom) et qui contient une donnée (la valeur).



La variable `x` : une boîte qui porte l'étiquette `x` et qui contient la valeur 5.

Définition 6 . Affectation

Associer une valeur à une variable s'appelle une **affectation** et utilise le signe **=**. Si la variable contenait déjà une valeur, la nouvelle affectation la **remplace** : l'ancienne valeur est effacée.

Exemple 1 . Affectations

```
1 x = 5
2 prenom = "Aragorn"
```

— `x = 5` range la valeur 5 dans la variable `x` ;
— `prenom = "Aragorn"` range le texte `"Aragorn"` dans la variable `prenom`.

Python sait calculer

Comme en mathématiques, Python sait additionner (+), soustraire (−) et multiplier. Attention : la multiplication s'écrit avec une étoile, `*`. Ainsi, `prix = 3 * 4` range 12 dans la variable `prix`. On verra d'autres opérations au chapitre 2.

Le signe = n'est pas une égalité

Le signe = ne veut pas dire "égal" comme en mathématiques, mais "**prend la valeur**". Python calcule d'abord ce qui est à **droite** du signe, puis range le résultat dans la variable de **gauche**. En Python, `x = x + 1` a donc un sens : la variable `x` voit sa valeur augmenter de 1.

1.4 Conventions de nommage

Propriété 1 . Règles de nommage

Un **nom de variable** doit respecter certaines règles :

- il ne contient que des lettres, des chiffres et le caractère _ (pas d'espace, pas de tiret, pas de symbole comme $) ;
- il commence toujours par une lettre ou par _.

Remarque 2. Python accepte les lettres accentuées (`âge`), mais on les **évite**, pour que le code puisse être lu et partagé partout (avec un clavier anglais, par exemple).

Remarque 3. Par convention, en Python, on utilise souvent le style dit **snake_case** : les mots sont en minuscules et séparés par des tirets bas (_). Exemple : `age_eleve`, `note_math`, `prix_total`.

Exemple 2 . Noms de variables

Corrects : `compteur`, `nombre_eleves`, `a2`.
Incorrects : `2x` (commence par un chiffre), `prix total` (espace)

Exercice 1 . Correct ou pas ?

Pour chaque nom, indiquer s'il peut être utilisé comme nom de variable. S'il est incorrect, expliquer pourquoi.

Nom	Correct ?	Pourquoi ?
`patate`		
`2notes`		
`note_finale`		
`nom eleve`		
`porte-monnaie`		
`note2`		
`$hello$`		
`_total`		
`nb.eleves`		

1.5 Une première fonction, l'affichage !

Définition 7 . Fonction

Une **fonction** est un outil qui réalise une action précise lorsqu'on l'appelle par son nom, suivi de parenthèses.

Définition 8 . La fonction `print`

`print` est la fonction qui permet d'**afficher une valeur** à l'écran, pour l'utilisateur.ice.

Exemple 3 . Afficher un texte

```
>>> print("Patate")
Patate
```

Elle affiche à l'écran le texte `Patate` : les guillemets ne sont pas affichés.

Afficher un élément, c'est bien, mais parfois on veut en afficher plusieurs : on peut alors les séparer par une **virgule**.

Exemple 4 . Afficher plusieurs éléments

```
>>> age = 16
>>> print("la personne a", age, "ans")
la personne a 16 ans
```

Exercice 2 . Que s'affiche-t-il ?

Pour chaque programme, écrire ce qui s'affiche. Si Python affiche une erreur, écrire **ERREUR**.

Programme	Affichage

`a = 3`
`b = 4`
`print(a + b)`	

`a = 3`
`b = 4`
`print("a + b")`	

`x = 12` `print("x vaut", x)`	
`print(Patate)`	

1.6 Suivre un programme : le tableau d'état

Méthode : le tableau d'état

Pour savoir ce que fait un programme, on note la valeur de chaque variable après chaque ligne, dans un **tableau d'état** : une colonne par variable, une ligne par instruction.

— Une case **vide** signifie que la variable n'existe pas encore.

— Quand une variable ne change pas, on **recopie** sa valeur.

Exemple 5 . Un tableau d'état

```
1 x = 4
2 y = x + 2
3 x = x * 3
```

Ligne	x	y
1	4	
2	4	6
3	12	6

À la ligne 3, x change, mais y reste à 6 : y a été calculée une seule fois, à la ligne 2.

Exercice 3 . Premier tableau d'état

Compléter le tableau d'état du programme suivant.

```
1 x = 5
2 y = 3
3 x = x + y + 2
```

Ligne	x	y
1		
2		
3		

Exercice 4 . Deuxième tableau d'état

Compléter le tableau d'état du programme suivant.

```
1 a = 10
2 b = 2
3 a = a - b
4 b = a * b
```

Ligne	a	b
1		
2		
3		
4		

Exercice 5 . Plus difficile

Compléter le tableau d'état du programme suivant. Attention aux priorités de calcul.

```
1 x = 2
2 y = x * 3
3 z = x + y
4 x = z - y
5 y = y + x * 2
6 z = z + z
```

Ligne	x	y	z
1			
2			
3			
4			
5			
6			

Exercice 6 . Le défi de l'échange

On dispose de deux variables x et y.

1. Affecter à x la valeur 5 et à y la valeur 12.
2. Échanger ensuite les valeurs de x et y, sans redonner directement les nombres 5 et 12.
3. Vérifier votre programme avec un tableau d'état.

Indice

Pour échanger le contenu de deux verres, il faut un troisième verre ! On a le droit de créer une nouvelle variable.

CHAPITRE 02

Types et calculs

Programmation en Python	Chap 02	**Types et calculs**

2.1 Les types de valeurs

Au chapitre 1, on a rangé dans des variables des nombres et du texte. Mais on ne peut pas tout faire avec n'importe quelle valeur : additionner deux nombres a un sens, additionner deux textes… c'est moins clair. C'est le type de la valeur qui décide.

Définition 1 . Type

En informatique, chaque valeur a un **type**, qui indique de quelle sorte de donnée il s'agit (nombre, texte, etc.). Une variable a le type de la valeur qu'elle contient. Une variable peut changer de type au cours d'un programme.

Propriété 1 . Le type décide

Le **type** d'une valeur détermine ce que l'on peut faire avec elle.

Les types de base

En Python, les types les plus simples sont :

- **int** : les nombres entiers (ex. `12`, `-27`, `2027`) ;
- **float** : les nombres décimaux (ex. `3.14`, `-0.5`) (⚠ on utilise le point, pas la virgule !) ;
- **str** : les chaînes de caractères, c'est-à-dire du texte, entouré de guillemets (ex. `"bonjour"`, `"7"`) ;
- **bool** : les valeurs de vérité `True` (vrai) ou `False` (faux), que l'on verra au chapitre 3.

Exercice 1 . Quel type ?

Donner le type de chacune des variables suivantes.

```
1 a = 14
2 b = 5.1
3 c = "patate"
4 d = "27"
5 e = 3.0
6 f = "12 + 40"
```

Variable	Type
a	
b	
c	
d	
e	
f	

2.2 Les opérations

Au chapitre 1, on a vu que Python sait additionner, soustraire et multiplier. Il connaît d'autres opérations, un peu moins habituelles.

Retrouver le sens des opérations

Voici des exemples de calculs réalisés par Python. Pour chaque opérateur, retrouver l'opération qu'il effectue.

Opérateur	Exemples	Opération effectuée
`+`	`7 + 2` donne 9	
`-`	`12 - 27` donne −15	
`*`	`7 * 2` donne 14	
`/`	`7 / 2` donne `3.5` `10 / 5` donne `2.0`	
`//`	`7 // 2` donne 3 `36 // 5` donne 7 `27 // 12` donne 2	
`%`	`7 % 2` donne 1 `36 % 5` donne 1 `27 % 12` donne 3	
`**`	`7 ** 2` donne 49 `2 ** 3` donne 8	

Exercice 2 . Calculs en tableau d'état

Compléter le tableau d'état du programme suivant, puis donner le type de `b` à la fin du programme.

```
1  a = 17
2  b = a // 5
3  c = a % 5
4  a = b ** 2 + c
5  b = a / 2
```

Type de `b` à la fin :

Ligne	a	b	c
1			
2			
3			
4			
5			

2.3 Opérations et types

Propriété 2 . Le + et le * sur les textes

Une même opération peut avoir un sens différent selon le type :

```
>>> print(3 + 4)
7
>>> print("hello" + "vous")
hellovous
>>> print("ha" * 3)
hahaha
```

Sur des chaînes de caractères, le + colle les deux chaînes bout à bout, sans ajouter d'espace : pour en avoir un, il faut l'écrire dans la chaîne, `"hello " + "vous"`. Le $*$ répète la chaîne.

Exercice 3 . Coller et répéter

Pour chaque programme, écrire ce qui s'affiche.

Programme	**Affichage**
`print("pa" + "tate")`	
`print("12" + "27")`	
`print("ha" * 4)`	
`print("3" * 2)`	
`mot = "bla"` `print(mot + " " + mot)`	
`print("bon" + "jour" * 2)`	

Exercice 4 . Fabriquer des textes

Écrire une instruction qui affiche :

1. `Bonjour Arwen !`
   en utilisant + et la variable `prenom = "Arwen";`

   ---

2. une ligne de 30 étoiles `******************************`
   en utilisant `*`;

   ---

3. `hip hip hip hourra !`
   en utilisant `*` et +.

Exercice 5 . Un nombre plus un texte ?

Que va-t-il se passer si on exécute `7 + "7"`, selon vous ?

Propriété 3

2.4 Demander une valeur à l'utilisateur.ice

Définition 2 . La fonction `input`

Pour demander une information à la personne qui exécute le programme, on utilise la fonction `input`. Elle affiche le message donné entre parenthèses, attend une réponse au clavier, puis la **renvoie** : on la range en général dans une variable.

```
prenom = input("Comment t'appelles-tu ? ")
print("Bonjour", prenom)
```

À l'écran

```
Comment t'appelles-tu ? Aragorn
Bonjour Aragorn
```

L'utilisateur.ice a tapé **Aragorn** (en gras) puis appuyé sur Entrée : la variable `prenom` contient alors `"Aragorn"`.

input renvoie toujours du texte

Même si l'utilisateur.ice tape un nombre, `input` renvoie du texte : `"15"` et non `15`. On ne peut donc pas faire de calcul directement avec la réponse : il faut d'abord changer son type.

Définition 3 . Changer de type : `int` et `str`

- `int` transforme un texte qui contient un nombre entier en **entier** : `int("15")` donne l'entier `15`, avec lequel on peut calculer.
- `str` transforme un nombre en **texte** : `str(15)` donne la chaîne `"15"`, que l'on peut coller à un autre texte avec `+`.

```
>>> print(int("15") + 1)
16
>>> print(str(15) + "1")
151
>>> print("Tu as " + str(16) + " ans")
Tu as 16 ans
```

Si le texte ne contient pas un nombre entier, `int` provoque une erreur : `int("patate")` donne une `ValueError`.

Exercice 6 . Avec int et str

Pour chaque instruction, écrire ce qui s'affiche, ou **ERREUR** si Python affiche une erreur.

Instruction	Affichage
`print(int("12") + 2)`	
`print(str(12) + "2")`	
`print(str(12) + 2)`	
`print(int("12") * 3)`	
`print(str(12) * 3)`	
`print(int("douze") + 2)`	

Exemple 1 . Calculer avec la réponse

```
1  age = int(input("Quel âge as-tu ? "))
2  print("Dans 10 ans, tu auras " + str(age + 10) + " ans")
```

À l'écran

```
Quel âge as-tu ? 15
Dans 10 ans, tu auras 25 ans
```

À la ligne 1, Python exécute d'abord `input`, qui renvoie `"15"`, puis `int` le transforme en l'entier 15. À la ligne 2, `str` retransforme le résultat 25 en texte pour le coller au reste de la phrase.

Exercice 7 . L'utilisateur.ice tape 12

À chaque fois, **l'utilisateur.ice tape** 12. Écrire ce qu'affiche chaque programme, ou **ERREUR** si Python affiche une erreur.

Programme	Affichage
`n = input("Un nombre ? ")` `print(n + 1)`	
`n = int(input("Un nombre ? "))` `print(n + 1)`	
`n = input("Un nombre ? ")` `print(n + n)`	
`n = int(input("Un nombre ? "))` `print("Le double est " + str(2 * n))`	