Les variables et les types

Capacités attendues

✓ Mettre en évidence un corpus de constructions élémentaires (ici déclaration/affectation de variables).

1 Les variables

Définition

Une variable informatique est la désignation par un **nom** d'un emplacement en mémoire où est stockée une **valeur** d'un certain type.

Dans un programme informatique,

- la création d'une variable s'appelle la déclaration ;
- l'instruction qui consiste à donner une valeur à une variable s'appelle l'affectation .

Le nom d'une variable :

- commence en général par une lettre miniuscule ;
- ne doit contenir que des lettres, chiffres ou tirets bas (pas d'espace, pas de symbole) ;
- doit être choisi pour illustrer son rôle dans le programme (compteur, indice, somme, *etc.*).

En Python :

- la déclaration est implicite et est réalisée avec la première affectation ;
- l'affectation s'effectue à l'aide du signe '=' selon la syntaxe :

```
nom_de_la_variable = valeur
```

- la fonction `print` permet d'afficher dans la console une description de la valeur.

Exemple :

Pour stocker le nombre de pommes qu'on a pour faire une tarte, on peut utiliser une variable dont le **nom** est `nb_pommes`. Avant de commencer la recette, elle contient la **valeur** `4`.

Exercice 1 – *(sur papier)*

Cocher les noms de variables invalides et expliquer pourquoi sur les pointillés. Préciser aussi sur les pointillés les noms de variables qui sont valides mais qui ne sont pas une bonne pratique de Python.

☐ `age` .....

☐ `user_age` .....

☐ `123abc` .....

☐ `_name` .....

☐ `first name` .....

☐ `favoriteColor` .....

☐ `last-name` .....

☐ `total$` .....

☐ `3dogs` .....

☐ `car_model` .....

2 Les types de variables

Définition

En informatique, un type de variable est une caractéristique commune à une classe d'objets.
En particulier, la façon dont est stockée en mémoire une variable dépend de son type (selon que la variable représente un entier signé, un entier non signé, un flottant, un booléen, *etc.*).

Exemple :

On considère une variable stockée en mémoire avec les octets suivants.

`1010 1001 0110 1110 0111 0011 0110 1001`

Selon le type de la variable, sa valeur est :

- le nombre 2842588009 si la variable est un entier non signé ;
- le nombre -1452379287 si la variable est un entier signé ;
- le nombre $-5,2946718325863965 \times 10^{-14}$ si la variable est un flottant ;
- la chaîne de caractères `'©nsi'` si la variable est une chaîne de caractères.

Principaux types de variables en Python

- `int` : les entiers (de taille arbitraire, signés ou non signés) ;
- `float` : les nombres flottants (codés sur 64 bits, avec un point pour la virgule : 2.13 pour $2,13$) ;
- `str` : les caractères et chaînes de caractères (entre guillemets doubles anglais "") ;
- `bool` : les booléens (True ou False, sans guillemets, avec majuscule) ;
- `list`: les tableaux / listes ;
- `tuple` : les *p*-uplets, ou tuples ;
- `dict` : les *p*-uplets nommés, ou dictionnaires.

Bonnes pratiques

- On ne change pas le type d'une variable : si on affecte une valeur d'un certain type à une variable, on lui affectera que des valeurs de ce type par la suite ;
- Si une fonction comporte différents renvois de valeurs, ces valeurs doivent être du même type.

Contre-exemple ! À NE PAS FAIRE !

```
s = 0
print(s)
s = " message "
```

Fonctions de conversion (*casting* ou transtypage)

Lorsque c'est possible, des fonctions permettent de convertir une valeur d'un type dans un autre type. En Python, elles portent le nom du type vers lequel on veut convertir la valeur (`int`, `str`, `float`, *etc.*).

La fonction `print`

Pour afficher la valeur d'une variable, on utilise la fonction `print` issue de la bibliothèque standard de Python.

Exemple :

```
1 ma_variable = 11
2 print(ma_variable)
```

Ce script affichera : `11`

La fonction `type`

Pour connaître le type d'une variable, on utilise la fonction `type` issue de la bibliothèque standard de Python.

Exemple :

```
num_carac = "4"
num = int(num_carac)

print(num_carac)            # affiche : 4
print(type(num_carac))      # affiche : <class 'str'>
print(num)                  # affiche : 4
print(type(num))            # affiche : <class 'int'>
```

Exercice 2 – *(sur papier puis vérification sur ordinateur)*

Pour chacune des valeurs proposées, indiquer le type de base utilisé **et** son nom en Python.

- `42` : un entier, int
- `3.14` : un nombre décimal, float
- `"Hello, world!"` : une chaîne de caractères, str
- `True` : un booléen, bool
- `-7` : un entier, int
- `"123"` : une chaîne de caractères, str
- `0.5` : un nombre décimal, float
- `False` : un booléen, bool
- `"True"` : une chaîne de caractères, str

3 L'affectation et les opérateurs

Affectation

Affecter une valeur à une variable consiste à stocker une valeur (qui a un type) dans l'espace de mémoire de la variable. On utilise l'opérateur = en Python, et la syntaxe :

`nom_de_la_variable = valeur.`

→ Cette valeur peut être changée dans la suite du programme. Une *variable*, par définition, n'est pas fixe !

Exemple :

```
1 nb_pommes = 4
2 # puis on découpe une pomme pour faire la tarte
3 nb_pommes = 3
```

Opérateurs

Les opérateurs permettent de modifier les valeurs des variables au fur et à mesure d'un programme.

On peut effectuer toute sorte d'opérations sur ces variables, avec différents opérateurs.

Symbole	Opérations	Opérations possibles	Erreurs

+	addition, concaténation	`int + int -> int`
		`int + float -> float`
		`float + float -> float`
		`str + str -> str`
		etc.
			`int + str -> ERROR`
			`list + str -> ERROR`
			etc.

−	soustraction	`int + int -> int`
		`int - float -> float`
		`float - float -> float`
		etc.
			`int - str -> ERROR`
			`list - str -> ERROR`
			`str - str -> ERROR !!!`
			etc.

*	multiplication	`int*int->int`
		`int*float->float`
		`int*str->str`
		etc.	

**	puissance	sur `int` ou `float`	
/	division	sur `int` ou `float`	
//	quotient de la division euclidienne	sur `int` ou `float`	
%	reste de la division euclidienne (modulo)	sur `int` ou `float`	

Exercice 3 – *(sur papier puis vérification sur ordinateur)*

Que contient la variable `nb_pommes` après exécution de ces instructions ? 9

```
1  nb_pommes = 4
2  nb_pommes = nb_pommes - 1
3  nb_pommes = nb_pommes * 3
```

→ On peut représenter l'évolution des valeurs des variables d'un programme avec un **tableau de trace**.

N° de ligne	Contenu de `nb_pommes`
1	4
2	3
3	9

Exercice 4 – *(sur papier puis vérification sur ordinateur)* Prévoir les affichages réalisés par le script ci-dessous.
10 8 13

```
1  a = 5
2  b = a - 2
3  a = a * 2
4  c = a + b
5  b = c - 5
6  print(a, b, c)
```

> **Ne pas confondre !**
> Il ne faut pas confondre l'égalité mathématique avec l'affectation bien que le caractère soit identique.

Exemple :

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

La ligne 2 n'aurait aucun sens en mathématiques, il s'agit ici d'affecter une nouvelle valeur à x : la valeur de x précédant l'affectation à laquelle on ajoute trois.
Une variable ne contient qu'une valeur à la fois, les précédentes valeurs sont écrasées au fur et à mesure des affectations.

Exercice 5 – *(sur papier puis vérification sur ordinateur)*

Cocher les opérations qui génèrent des erreurs. Vérifier dans la console Python et préciser l'essentiel de l'erreur sur les pointillés.

- ☐ `2 + 3` .....
- ☐ `"NSI" + 2024` X .....
- ☐ `"NSI" * 2024` .....
- ☐ `2.8 + 3` .....
- ☐ `"toto"**3` X .....
- ☐ `360 % 3` .....
- ☐ `"toto" / 2` X .....

Exercice 6 – *(sur papier puis vérification sur ordinateur)*

Quelle est la valeur et le type du résultat de chaque expression (vérifier sur ordinateur en cas de doute) ?

Opération	Valeur	Type
1 - 6	-5	int
10.1 + 0.9	11.0	float
68 / 2.0	34.0	float
5 * 4	20	int
5**4	625	int
7 % 10	7	int
10**2	100	int
13 // 3	4	int
13 % 3	1	int
"Hello" + "world!"	"Helloworld!"	str
"Salut " + "to"*2 + " !"	"Salut toto !"	str
int("25") + int("5")	30	int
"25" + "5"	"255"	str

Exercices en vrac

Exercice 7

1. Que vaut `x` après exécution de ce programme ? 7

```
x = 1
x = 5
x = x + 2
```

2. Que valent `x` et `y` après exécution de ce programme ? 1 et 3 respectivement

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

Exercice 8

On donne le programme suivant. Remplir le tableau de trace avec la valeur de chaque variable à chaque ligne du programme, et en déduire ce que fait ce programme.

```
1  a = 9
2  b = 24
3  t = a
4  a = b
5  b = t
```

N° de ligne	a	b	t
1			
2			
3			
4			
5			

Exercice 9

On donne le programme suivant :

```
x = 5
print(x)
x = x + 3
print(x)
```

1. Qu'est-ce qui va s'afficher à l'exécution ?
2. Dans votre éditeur Python, exécuter ce programme. Aviez-vous trouvé la bonne réponse ?

Exercice 10

Quels sont les affichages produits par le script ci-dessous ? Pourquoi ?

```
x = 2020*2
print(x, type(x))

x = 2020.
print(x, type(x))

x = 1+1<=2
print(x, type(x))

x = 2019/2
print(x, type(x))

x = 2019//2
print(x, type(x))

x = 10 % 3
print(x, type(x))

x = "nsi"
print(x, type(x))

x = [2, 4]
print(x, type(x))

x = {"nsi1": 24, "nsi2": 24}
print(x, type(x))

x = (2022, 2023)
print(x, type(x))
```

Exercice 11

Quels sont les affichages produits par le script ci-dessous ? Pour répondre à cette question, vous devez d'abord dérouler le code à la main, sur papier, et donner votre réponse, puis utiliser votre éditeur Python pour l'exécuter et vérifier le résultat précédent.

```
x = int(1.999)
print(x)
s = str(3.14)
t = s+s
print(t)
```

Exercice 12

1. Dans une console Python, taper les lignes suivantes et observer leurs résultats :

   ```
   >>> 5 - 3 - 2
   >>> 1 / 2 / 2
   ```

   Que peut-on en déduire sur le fonctionnement de la soustraction et de la division en Python ?

2. Réécrire les opérations suivantes en explicitant toutes les parenthèses (le résultat est inchangé avant et après) et vérifier les résultats dans votre console :
   (a) `1 + 2 * 3 - 4`
   (b) `1+2 * 4*3`
   (c) `1-a+a*a/2-a*a*a/6+a*a*a*a/24`
3. Réécrire les expressions suivantes en utilisant aussi peu de parenthèses que possible sans changer le résultat :
   (a) `1+(2*(3-4))`
   (b) `(1+2)+((5*3)+4)`
   (c) `(1-((2-3)+4))+(((5-6)+((7-8)/2)))`

Exercice 13

1. Que fait le programme suivant ?

```
a = 5
b = 6
a = b
b = a
```

2. Que fait le programme suivant ?

```
a = 7
b = 8
tmp = a
a = b
b = tmp
```

Exercice 14

On rappelle que la division euclidienne (ou division entière) entre deux nombres entiers naturels `a` et `b` donne un reste `r` et un quotient `q` est telle que : `a = (b * q) + r` (avec `r < b`).

On veut écrire un programme qui calcule le nombre de boîtes de 6 œufs nécessaire au transport d'un certain nombre d'œufs.

1. Questions préalables :
   (a) Quel opérateur permet d'obtenir le quotient d'une division euclidienne (entière) en Python ?
   (b) Quel opérateur permet d'obtenir le reste d'une division euclidienne (entière) en Python ?
   (c) Quel opérateur permet de faire une division décimale en Python ?
2. Un élève propose la solution suivante : `nb_boites = nb_oeufs // 6`
   (a) Pour quelles valeurs de `nb_oeufs` ce programme est-il correct ?
   (b) Un autre élève suggère d'écrire plutôt : `nb_boites = nb_oeufs // 6 + 1` Est-ce correct ? Pourquoi ?
   (c) Proposer une solution correcte.