



Explorer les variables, les types et les opérateurs

Découvrir expérimentalement le rôle de l'interpréteur, suivre l'évolution des variables, distinguer les principaux types simples et construire des expressions arithmétiques et booléennes fiables.

Situation. Un système embarqué reçoit une mesure brute provenant d'un capteur. Avant de l'exploiter, il doit la convertir, vérifier sa cohérence et décider si elle signale une situation anormale. Ce TP conduit progressivement à l'écriture de ce traitement.

Méthode de travail. Pour chaque expérience :

1. prévoir le résultat sans lancer Python ;
2. exécuter le code et relever le résultat exact ;
3. expliquer les éventuels écarts et ne pas hésiter à m'appeler ;
4. conserver les notes et éventuels coded pour plus tard.

1 | Découvrir la console et le script pas à pas

1.1 Repérer les zones de Spyder

Spyder est l'environnement de développement (IDE) que nous allons utiliser. On y distingue trois zones :

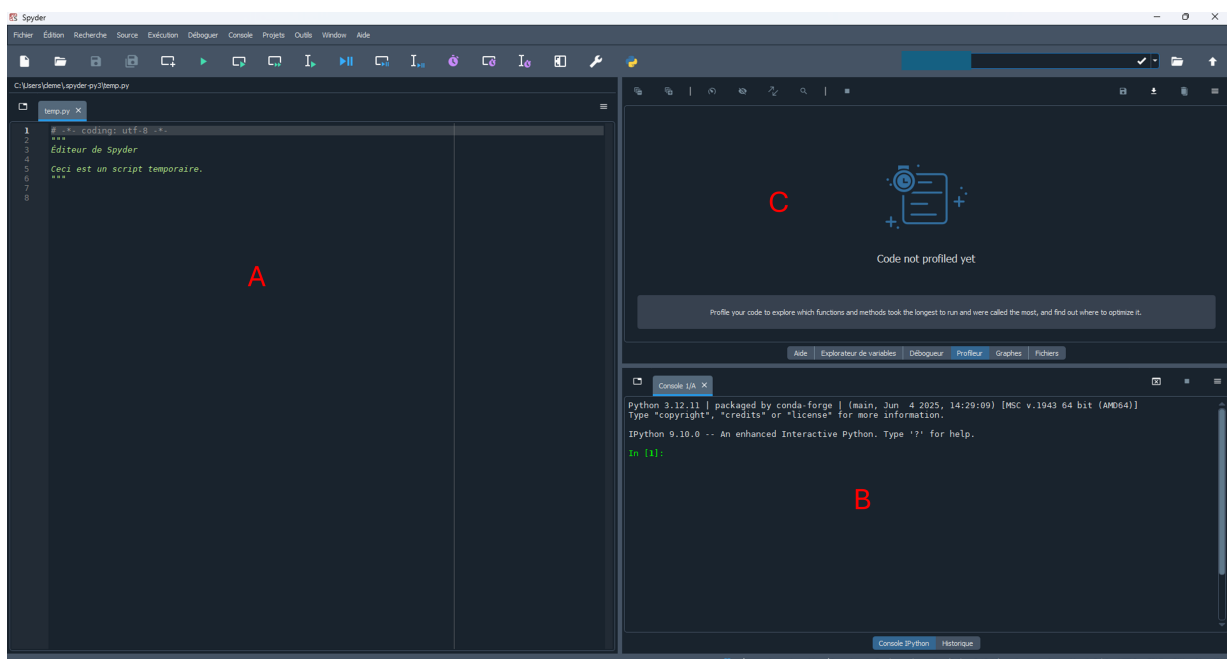


FIGURE 1 – Les principales zones de Spyder

- **Éditeur :** cette zone permet de rédiger et d'enregistrer un script dans un fichier portant l'extension `.py`.

- **Console Python** : cette zone permet d'exécuter immédiatement une instruction, d'afficher les résultats et d'échanger avec un script en cours d'exécution.
- **Panneaux complémentaires** : les onglets donnent notamment accès à l'explorateur de variables, aux graphiques, aux fichiers et à l'aide.

Question 1 : Repérer ces zones dans Spyder (figure 1) et faire la correspondance entre les zones et les lettres.

Réponse 1 : La zone A est l'éditeur, la zone B est la console Python et la zone C regroupe les panneaux complémentaires, dont l'explorateur de variables.

1.2 Premiers calculs dans la console

Dans la console Python, cliquer après l'invite In [...] : et saisir les instructions suivantes une par une. Valider chaque ligne avec la touche Entrée.

```
largeur = 8
hauteur = 5
largeur * hauteur
```

Question 2 : Déterminer le moment où chaque instruction est exécutée, puis distinguer l'effet d'une affectation de celui d'une expression saisie seule dans la console.

Réponse 2 : La console exécute chaque instruction dès sa validation. Les affectations ne produisent pas d'affichage, tandis que la valeur d'une expression saisie seule est affichée par la console. Ici, la dernière ligne affiche 40.

1.3 Créer, enregistrer et exécuter un script

Effectuer les opérations suivantes dans l'ordre indiqué.

1. Créer un dossier TP1 dans votre répertoire d'informatique (ITC).
2. Dans Spyder, choisir *Fichier > Nouveau fichier*.
3. Enregistrer immédiatement le fichier dans TP1 sous le nom TP1_C1.py.
4. Saisir le programme ci-dessous dans l'éditeur.
5. Enregistrer avec Ctrl+S, puis exécuter avec F5 ou le bouton *Exécuter le fichier* (flèche verte 'start').

```
largeur = 8
hauteur = 5
aire = largeur * hauteur
aire
```

Question 3 : Prévoir puis relever les lignes affichées ou pas dans la console. Expliquer l'absence d'affichage produite par la ligne contenant seulement aire dans le script.

Réponse 3 : Dans un script, une expression seule est évaluée mais son résultat n'est pas affiché. Il faut utiliser la fonction `print(...)` pour envoyer une valeur vers la console.

Question 4 : A partir de l'explorateur de variable (zone C), déterminer le nom et la valeur des variables largeur, hauteur et aire.

Réponse 4 : L'explorateur de variable donne les valeurs 8, 5 et 40 respectivement pour largeur, hauteur et aire.

1.4 Échanger avec le script grâce à print et input

La fonction `print(...)` affiche dans la console les objets placés entre parenthèses. A l'inverse, la fonction `input(...)` affiche un message, suspend l'exécution du script et attend une saisie au clavier dans la console.

Remarque : Elle renvoie toujours le texte saisi, donc une variable de type `str`. Nous en parlerons plus tard.

Remplacer le script précédent par les lignes suivantes, enregistrer, puis exécuter de nouveau le fichier.

```
print("Debut du script")
prenom = input("Entrer le prenom : ")
age = input("Entrer l'age : ")
print("Bonjour", prenom)
print("Votre age est :", age)
```

1.5 Retrouver dans la console les objets créés par le script

Après l'exécution du script, revenir à l'invite de la console sans relancer le fichier et saisir ligne à ligne :

```
prenom
age
aire
aire / 2
```

Observer également l'*explorateur de variables* de Spyder.

Question 5 : Relever les objets encore accessibles après la fin du script et montrer que la console permet de poursuivre leur manipulation.

Réponse 5 : Les variables `prenom`, `age`, `aire` et les autres objets créés par le script restent présents dans le contexte de l'interpréteur. La console permet de consulter leur valeur et d'évaluer de nouvelles expressions, par exemple `aire / 2`, sans modifier le script.

À retenir. La console exécute immédiatement une instruction saisie. Un script est un fichier enregistré qui permet de conserver et de réexécuter plusieurs instructions dans l'ordre. `print` envoie des informations vers la console ; `input` récupère une saisie effectuée dans la console. Après l'exécution, les objets créés par le script restent accessibles dans le contexte de l'interpréteur.

2 | Suivre les variables comme un programme

Question 6 : Sans exécuter le code, compléter mentalement le tableau d'évolution associé.

```
a = 7
b = a
a = a + 3
c = (a == b)
b += 1
```

Après l'instruction	Valeur de a	Valeur de b	Valeur de c
<code>a = 7</code>	7		
<code>b = a</code>	7	7	
<code>a = a + 3</code>	10	7	
<code>c = (a == b)</code>	10	7	False
<code>b += 1</code>	10	8	False

Question 7 : Exécuter ensuite le code ligne par ligne en consultant l'explorateur de variables. Corriger le tableau et donner les valeurs finales de `a`, `b` et `c`.

Réponse 7 : Les états successifs sont $(7, -, -)$, $(7, 7, -)$, $(10, 7, -)$, $(10, 7, \text{False})$, puis $(10, 8, \text{False})$. Les valeurs finales sont donc $a = 10$, $b = 8$ et $c = \text{False}$.

Question 8 : Expliquer la validité de l'instruction $a = a + 3$ en Python malgré l'impossibilité de l'égalité mathématique $a = a + 3$, puis justifier que la modification de a ne modifie pas automatiquement b .

Réponse 8 : Le symbole $=$ est une affectation : Python évalue d'abord le membre de droite avec l'ancienne valeur de a , puis range le résultat sous le nom a . L'instruction $b = a$ affecte à b la valeur courante de a ; elle ne crée pas une égalité mathématique permanente entre les deux noms. C'est la notion de code séquentielle.

À retenir. Une variable est caractérisée par un nom, une valeur et un type. Une affectation modifie uniquement la variable placée à gauche du symbole $=$.

3 | Les types

Question 9 : Pour le script suivant, associer chaque valeur à son type Python, puis identifier les objets qui représentent un nombre.

```
n = 42
x = 42.0
z = 1 + 2j
test = n > 0
texte = "monTexte"

print(type(n), type(x), type(z))
print(type(test), type(texte))
```

Réponse 9 : n est un `int`, x un `float`, z un `complex` et `test` un `bool`. `texte` est un `str`, c'est à dire une chaîne de caractère, nous en parlerons plus tard. Les trois premiers représentent des nombres. Même si en Python les booléens `0` et `1` sont en réalité des `int`...

Question 10 : Exécuter le script (à la suite ou dans un nouveau `.py`) puis déterminer ce que fait la fonction `type(...)`.

Réponse 10 : Elle renvoie le type de la variable.

3.1 Influence du nom sur le type

Exécuter successivement les instructions suivantes dans la console.

```
mesure = 12
print(mesure, type(mesure))
mesure = 12.0
print(mesure, type(mesure))
mesure = "12"
print(mesure, type(mesure))
```

Question 11 : Déterminer si le nom `mesure` impose un type, puis indiquer ce qui fixe le type de la variable à chaque étape.

Réponse 11 : Le nom n'impose aucun type. La variable est successivement de type `int`, `float`, puis `str`, selon la valeur qui lui est affectée.

3.2 Conversions explicites

Pour chacune des expressions suivantes, prévoir la valeur et le type du résultat, puis vérifier dans la console.

Expression	Valeur prévue	Type prévu
<code>float(7)</code>		
<code>int(7.9)</code>		
<code>complex(3)</code>		
<code>bool(0)</code>		
<code>bool(-2)</code>		
<code>int("12") + 3</code>		
<code>"12" + "3"</code>		

Question 12 : Expliquer en particulier la différence entre les deux dernières expressions et l'effet de `int(7.9)`.

Réponse 12 : Les résultats sont 7.0 (`float`), 7 (`int`), (3+0j) (`complex`), False (`bool`), True (`bool`), 15 (`int`) et "123" (`str`). La conversion `int(7.9)` tronque la partie décimale : elle n'arrondit pas. Avec deux chaînes, l'opérateur + réalise une concaténation, nous verrons cela en cours plus tard.

À retenir. En Python, le type appartient à l'objet manipulé. Les fonctions `int`, `float`, `complex`, `bool` et `str` permettent de demander explicitement certaines conversions.

4 | Choisir le bon opérateur

4.1 Retrouver une division euclidienne

Un enregistrement audio dure 7 384 secondes.

Question 13 : Sans utiliser de fonction de conversion, calculer avec `//` et le modulo le nombre d'heures, de minutes et de secondes correspondant. Vérifier que le programme affiche "2 h 3 min 4 s".

Réponse 13 :

```
duree = 7384
heures = duree // 3600
reste = duree % 3600
minutes = reste // 60
secondes = reste % 60
print(heures, "h", minutes, "min", secondes, "s")
```

Question 14 : Écrire une unique expression booléenne qui vérifie la recombinaison de la durée à partir de heures, minutes et secondes.

Réponse 14 :

```
duree == 3600*heures + 60*minutes + secondes
```

Cette expression vaut `True`.

4.2 Précision des nombres flottants

Question 15 : Exécuter, une à une, les instructions suivantes. Prévoir les résultats, exécuter, comparer puis conclure.

```
somme = 0.1 + 0.2
print(somme)
print(somme == 0.3)
print(abs(somme - 0.3))
```

Réponse 15 : Python affiche une valeur voisine de 0.30000000000000004 et le test d'égalité exacte vaut donc False. Ceci est dû à l'approximation des nombres en fonction de comment ils sont codés, on y reviendra plus tard dans l'année.

À retenir. Un float est une représentation approchée. Pour comparer deux résultats calculés, on teste généralement si leur écart est inférieur à une tolérance choisie.

5 | Construire et vérifier des expressions booléennes

5.1 Table de vérité expérimentale

Question 16 : Pour chaque couple de valeurs de a et b, exécuter les expressions not a, a and b et a or b, puis compléter le tableau. Déduire de l'expérience la règle qui rend a and b vraie, puis celle qui rend a or b vraie.

a	b	not a	a and b	a or b
False	False			
False	True			
True	False			
True	True			

Réponse 16 : Pour les quatre lignes, not a vaut successivement True, True, False, False; a and b vaut False, False, False, True; et a or b vaut False, True, True, True.

Pour que le résultat d'un and soit True il faut que les deux opérandes soient True. Pour que le résultat d'un or soit True il suffit que l'un des deux opérandes soient True.

5.2 Traduire une spécification

Une température T est considérée comme plausible si elle appartient à l'intervalle $[-20;80]$. Une alerte est déclenchée lorsqu'une température plausible atteint ou dépasse 30°C.

Question 17 : Écrire les expressions booléennes plausible et alerte, puis les tester successivement pour T = -30, T = 22.5, T = 30 et T = 95.

Réponse 17 :

```
plausible = -20 <= T <= 80
alerte = plausible and T >= 30
```

Les couples obtenus sont respectivement :

(False, False), (True, False),
(True, True) et (False, False).

Question 18 : Écrire une expression fonctionnement_normal vraie uniquement pour une mesure plausible sans alerte active.

Réponse 18 :

```
fonctionnement_normal = plausible and not alerte
```

6 | Défi de synthèse : qualifier une mesure de capteur

Créer un nouveau fichier .py (un nouveau script) pour cette partie.

Un convertisseur fournit un entier brut compris entre 0 et 1023. La tension associée est donnée par

$$U = 5 \frac{\text{brut}}{1023},$$

et le capteur délivre une température en degrés Celsius selon

$$T = 100(U - 0,5).$$

Question 19 : Écrire un programme qui, à partir de `brut = 164`, calcule `U` et `T`, puis affiche leur valeur et leur type.

Réponse 19 :

```
brut = 164
U = 5 * brut / 1023
T = 100 * (U - 0.5)
print(U, type(U))
print(T, type(T))
```

Les variables `U` et `T` sont des flottants. On obtient environ $U = 0,8016 \text{ V}$ et $T = 30,16^\circ\text{C}$.

Question 20 : Ajouter les booléens `code_valide`, `temperature_plausible`, `alerte` et `fonctionnement_normal`. Ils doivent traduire respectivement : $0 \leq \text{brut} \leq 1023$, $-20 \leq T \leq 80$, une température plausible supérieure ou égale à 30°C , puis une situation plausible sans alerte.

Réponse 20 :

```
code_valide = 0 <= brut <= 1023
temperature_plausible = -20 <= T <= 80
alerte = code_valide and temperature_plausible and T >= 30
fonctionnement_normal = code_valide and temperature_plausible and not alerte
```

Avec `brut = 164`, le code et la température sont plausibles, l'alerte est vraie et le fonctionnement normal est faux.

Question 21 : Tester le programme avec `brut = -1`, `brut = 100`, `brut = 164` et `brut = 1100`. Présenter les résultats dans un tableau et justifier chaque booléen obtenu.

brut	T	Code valide	T plausible	Alerte	Normal	
-1						
100						
164						
1100						

Réponse 21 : Pour `-1`, le code est invalide. Pour `100`, le code est valide, la température vaut environ $-1,12^\circ\text{C}$ et le fonctionnement est normal. Pour `164`, l'alerte est active. Pour `1100`, le code et la température calculée sont hors domaine, donc les indicateurs de bon fonctionnement sont faux.

Question 22 : Rédiger en trois phrases un bilan distinguant clairement affectation, comparaison et opération booléenne.

Réponse 22 : Une affectation évalue une expression puis associe le résultat à un nom de variable avec `=`. Une comparaison, par exemple avec `==` ou `<=`, produit une valeur booléenne. Les opérateurs `not`, `and` et `or` permettent enfin de combiner ces valeurs pour traduire un critère plus complexe.