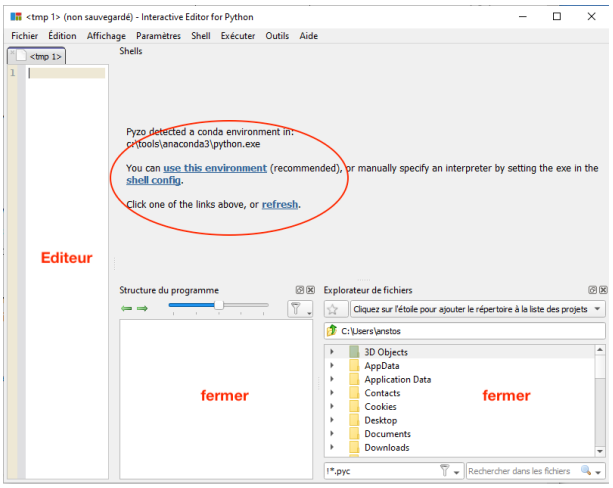
TP 0 : Prise en main

Prise en main de l'environnement de travail

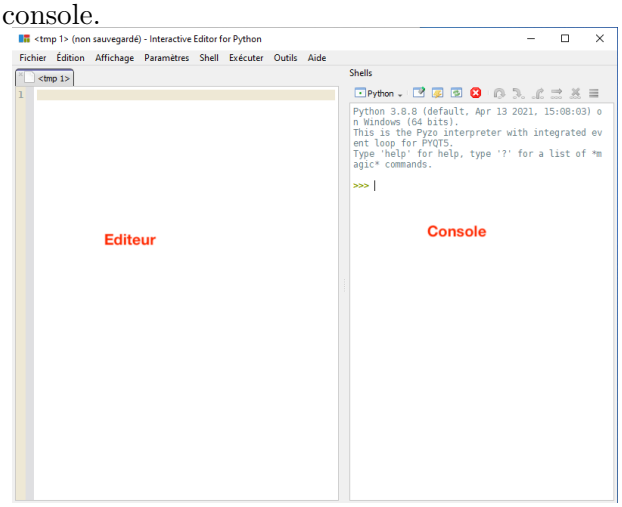
Sur votre machine personnelle ou sur une machine virtuelle (`https://vdiportail.dsi.uca.fr`) effectuez les actions suivantes.

1. Dans le menu `Démarrer` → `Tous les programmes` trouvez et lancez l'application Pyzo.
   Au premier lancement, Pyzo détecte l'installation de Python et demande une confirmation pour l'utiliser.



   Cliquer "use this environement" pour confirmer : une console interactive de Python apparaitra dans ce même cadre.

2. Il est commode de fermer deux petites cadres en bas à droite pour agrandir l'espace de travail utile : l'éditeur et la console.



3. Dans la console, calculez `2+2`.
4. A la différence de calculettes dont vous avez l'habitude, en Python la fonction puissance est notée `**` (deux étoiles). En utilisant cette fonction, calculez dans la console 5 puissance 4.
5. Pour faire des calculs plus complexes et pour les sauvegarder, on crée des *scripts* (programmes) dans l'éditeur. Sur le disque ce sont des fichiers d'extension `.py`.
   Allez dans la fenêtre de l'éditeur (à gauche) et tapez : `print(2+2)`
   Sauvegardez ce script sur le disque (on pourra le nommer `tp0exo0.py`).
   Lancez votre script en tapant `Ctrl-E`. Observez le résultat dans la console.

Prenez l'habitude de sauvegarder souvent votre travail de TP (`Ctrl-S`) !

Attention : Pour ouvrir vos fichiers `*.py` dans une fenêtre d'explorateur de fichiers Windows, un double-click ne fonctionnera pas (l'extension n'est pas forcément reconnue par Windows). On peut toujours ouvrir ses scriptes à partir de l'application Pyzo.

6. Dans un script, un commentaire se met après le signe dièse `#`. Par exemple :

   ```
   1+1 # c'est trop facile!
   ```

   Cela est très utile pour expliquer ce qui se passe dans vos programmes et noter les réponses aux questions posées dans la feuille de TP.

Exercice 1. *Variables et opérations*

On peut penser qu'une *variable* est une boîte qui contient une valeur. Pour affecter à une variable `x` une valeur, on utilise le symbol "`=`", par exemple :

```
x = 5
```

Attention ! Il faut comprendre que ceci ne représente pas une égalité mathématique, ni d'ailleurs une comparaison !

C'est une action effectuée par l'ordinateur (une modification dans la mémoire vive de la machine).

On va lire : *x prends la valeur 5.*

Effectuons maintenant quelques opérations basiques (à taper dans la console) :

```
x = 5
y = 2 * x
print("la somme x + y = ", x + y)
```

Dans la dernière ligne, on remarquera une chaine de caractères entre guillemets.

Python distingue majuscules et minuscules. Tapez :

```
a = 1
A = 2
print(a, A)
```

A présent, on se propose d'augmenter la valeur de `x` de 1. On tape (dans la console) :

```
x = x + 1
print(x)
```

Encore une fois, notez bien la différence entre l'égalité mathématique et l'affectation ci-dessus. Alors que l'équation algébrique $x = x + 1$ n'a pas beaucoup d'intérêt, l'action effectuée ci-dessus est bien courante en programmation :

tout d'abord, la machine calcule la valeur de l'expression à droite, c'est-à-dire de `x + 1`, puis elle affecte cette valeur à la variable qui se trouve à gauche, `x` (en oubliant son ancienne valeur 5).

Par ailleurs, Python propose une élégante syntaxe pour cette opération : au lieu d'écrire `x = x + 1` on peut taper `x += 1` (essayez !). Cela a l'avantage d'éviter une confusion avec une équation algébrique. On devine facilement d'autres opérateurs : `-=` `*=` `/=`

Exercice 2. *Affichage*

Tapez les exemples de la section "*Affichage*" de l'aide mémoire. Observez les résultats. Afficher 1023.98765 arrondi à 2 chiffres après la virgule.

Ensuite, écrivez un code qui affecte une valeur à une variable `x` (p. ex. `x = 10.23`) et qui calcule puis affiche cette valeur élevée au carré sous la forme `"10.23 au carré = ...."`.

Exercice 3. *Test* `if`

1. Étudiez les exemples concernant le test `if` dans l'aide mémoire. Tapez dans la console :
   ```
   0 == 1
   2 > 1
   ```
   Il faut bien distinguer l'affectation "=" de la comparaison "==" !
   Tapez par exemple :
   ```
   a = 10
   a == 20
   print(a)
   ```
2. Créez un script avec le code ci-dessous. Faites attention à respecter le décalage de 4 espaces pour des commandes `print` (sauf la dernière ligne).
   ```
   a = 10
   b = 12
   if a >= b:
       print("a est plus grand")
   else:
       print("b est plus grand")
   print("ceci s'affiche toujours")
   ```
   Lancez ce script plusieurs fois avec différentes valeurs de variables `a` et `b`.

   Observez le rôle du décalage dans ce code.

   On dit que le décalage marque un *bloc* de commandes. C'est une propriété importante de la syntaxe Python qui contribue beaucoup à la lisibilité du code. Attention, ce décalage doit être toujours de la même taille de 4 espaces.
3. On peut utiliser plusieurs conditions en les séparant par une opération logique `and` ("et") ou `or` ("ou"). Par exemple :
   ```
   a = 10
   if a >= 9 and a <= 11:
       print("a est dans l'intervalle [9, 11]")
   ```
   Mettez ce code dans un script, testez-le avec différentes valeurs de la variable `a`.

Exercice 4. *Boucle* `for`

La boucle `for` permet d'effectuer les mêmes commandes plusieurs fois avec la valeur d'un index variant dans une liste. Cette liste est le plus souvent générée à l'aide de la commande `range` qui fait varier l'index entre deux bornes. Par exemple :

— `range(3, 7)` renvoie `3, 4, 5, 6`, la borne supérieure 7 étant par définition exclue.
— `range(4)` est un raccourci pour `range(0, 4)` : cela renvoie donc `0, 1, 2, 3`.

1. Créez un script avec l'exemple d'une boucle `for` dans l'aide mémoire (encore attention au décalage !), exécutez et observez les résultats.
2. Créez un script qui affiche les nombres pairs entre 10 et 40 inclus. On pourra utiliser l'opérateur `%` (le reste de la division euclidienne, cf. l'aide mémoire).