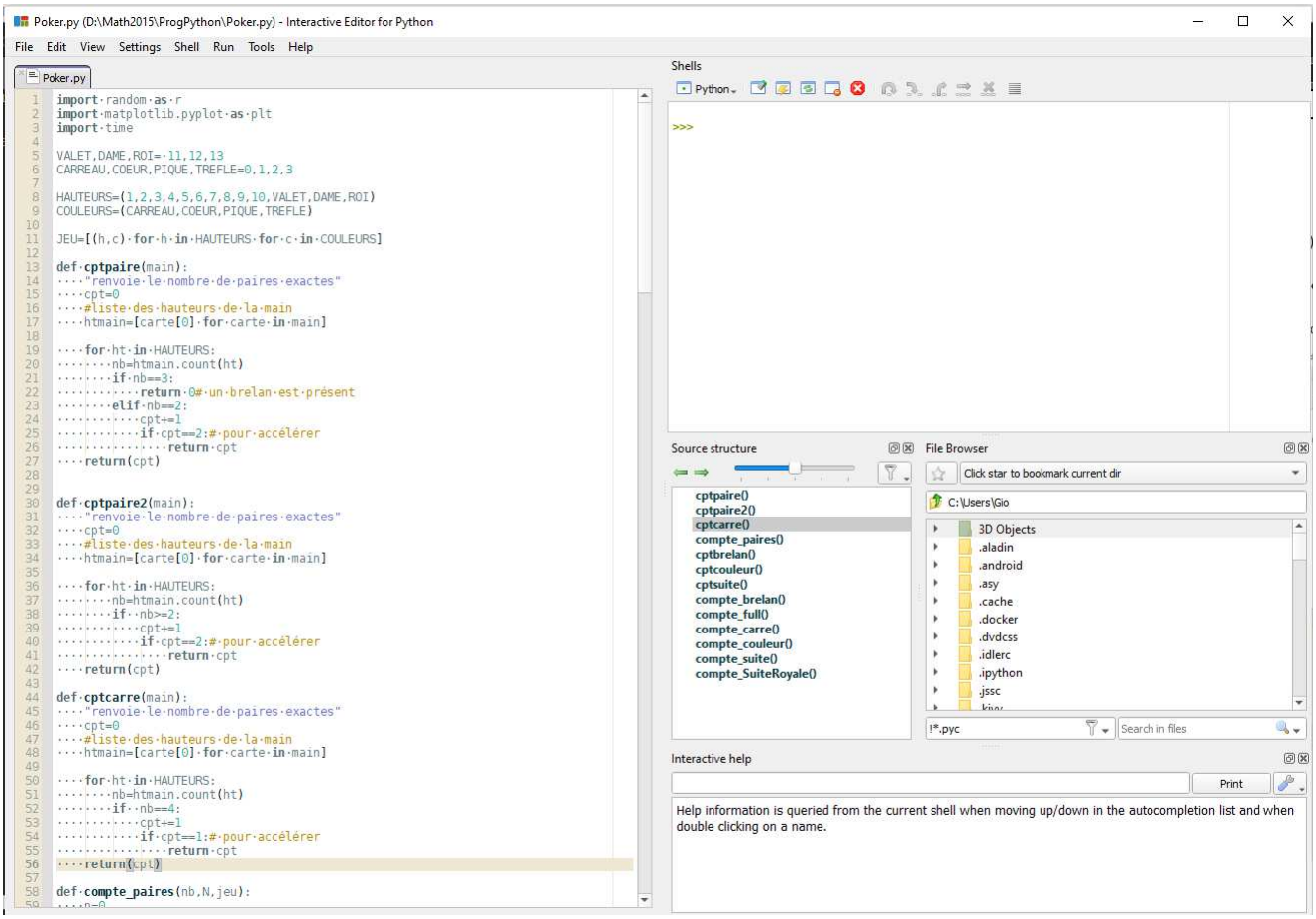
1 Préliminaires

Nous allons utiliser le logiciel Pyzo (gratuit). **Lancer Pyzo**.
L'interface comporte trois fenêtres *a priori* :

- la fenêtre **shell** (console en Français) où l'on peut exécuter directement des instructions (partie haute sur l'image ci-dessous)
- la fenêtre **module de travail** qui permet de créer un fichier où l'on écrit son code de programmation
- un « workspace » (espace de travail) qui fournit une aide éventuelle : on peut choisir ces fenêtres selon nos besoins.



1.1 La console : shell

Dans la partie **shell**, on peut taper directement des instructions en langage Python, comme par exemple des opérations arithmétiques mais aussi des éléments de programme que nous avons défini.

Taper les instructions des sections suivantes dans la console. Indiquer et expliquer s'il y a lieu le résultat dans la colonne de droite.

1.2 Opérations de base.

⋙ **7+8**	
⋙ **2*6**	
⋙ **2**6**	
⋙ **36/5**	

⋙ **36//5** ⋙ **5//2**	
⋙ **36%5** ⋙ **5%2**	
⋙ **int(3.14), int(-3.14)**	
⋙ **round(3.4), round(3.5)**	

Pour la partie entière d'un nombre, il faut d'abord importer le module math avec :

import math (à préférer) ou **from math import ***

L'étoile signifie que l'on importe *toutes* les fonctions de ce module.

Si l'on ne veut importer qu'un nombre limité de fonctions, on écrira par exemple

from math import floor, ceil

⋙ **floor(3.1), floor(2.9)**	
⋙ **ceil(3.1), ceil(2.9)**	

Pour effacer le *shell*, on peut taper **cls** ou utiliser le raccourci **crt+L**.

Mais cela n'efface pas la mémoire vive, les fonctions du module math sont encore accessibles.

Pour vider la mémoire vive, on peut ouvrir une nouvelle console (new shell).

Pour obtenir une description d'une instruction, on peut utiliser l'instruction help() :

par exemple taper après **import math** : help(math) ou dir(math)

⋙ **math.pi**	

Pour ne pas écrire à chaque fois math.fonction, on utilise un alias **import math as m**

⋙ **m.e, "%0.3f" %m.e**	

☞ **S'exercer** **Combien de décimale peut-on obtenir au maximum ?**

✘✘✘ Attention ! Les variables définies dans un module ne sont pas protégées. Par exemple, si on tape pi=3, on perd la valeur de pi. Idem avec des fonctions comme exp, si on tape exp=3, on perd la définition de la fonction exp.

2 Les variables, l'affectation et le type de variables.

⋙ print(x) **⋙ x=x+2**	
⋙ print(x) **⋙ print(X)**	
⋙ x=5 **⋙ print('x')** **⋙ print('x= ',x)**	
⋙ 3x **⋙ 3*x**	

Pour les tableaux suivants, essayer de prévoir le résultat final avant de taper les instructions.

⋙ x=2 **⋙ x=x+28** **⋙ x=x/11** **⋙ x=3**x**	

⋙ x=2 **⋙ y, z=2*x, x+3** **⋙ x, y, z** **⋙ x=y=12** **⋙ x, y** **⋙ y=1** **⋙ x, y**	

Exercice 1

Que font ces instructions ?

1)

$x = 1$
$y = 2.6$
`x=y ; y=x ;`

2)

$x = 1$
$y = 2.6$
$x, y = y, x$
x, y

⋙ '2'*2 **⋙ 'Einstein '*30**	
⋙ type(2) **⋙ type('2')**	

3 Premiers programmes

Les programmes sont des suites plus ou moins longues d'instructions. On crée pour cela un fichier : File → New dans *l'éditeur de script*.

Quelques conseils en vrac avec de commencer :

- On veillera à donner un nom en rapport avec le programme pour pouvoir le retrouver rapidement. Exemple Tp1_Ex1 à l'aide de la commande File → Save
- Sauvegarder assez souvent votre travail pour ne pas risquer de tout perdre suite à une mauvaise manip ou bien une extinction subite de votre machine.
- Créer un répertoire (dossier) sur votre disque dur qui contiendra tous vos fichiers Python pour ne pas perdre de temps à les retrouver.
- On évitera dans les noms de fichiers les accents, les espaces... qui peuvent engendrer des soucis par la suite.

Exercice 2

1) Dans un premier temps sans ordinateur, « Faire tourner ces programmes à la main » et déterminer :
   - Les variables qui interviennent dans ces programmes.
   - Les valeurs de chacune de ces variables à la fin du programme.

2) Créer un fichier Tp1_Ex1, taper ces programmes (un par un) et vérifier vos résultats. Pour éxecuter un programme.
   Lorsque vous passez d'un programme au suivant, bien mettre les premiers en commentaire en mettant # en début de chaque ligne ou sélectionner ce programme et File → Edit → Comment pour que Python n'en tienne pas compte lors de la compilation du programme.

```
a=3
b=5
c=b ;
b=a ;
a=c
```

$$\begin{matrix} a=2 \\ b,a \\ a+6,a+5 \\ b=a-6 \end{matrix} =$$

$$\begin{matrix} a,b=5,1 \\ c=a+b \\ a=c^2 \\ b=c-a^2 \end{matrix}$$

$$\begin{matrix} a=3 \\ b=a^2 \\ c=a+b \\ a=c^3 \\ b=c-a \end{matrix}$$

$$\begin{matrix} n=0 \\ u=0 \\ n=n+1 \\ u=u+2n+1 \\ n=n+1 \\ u=u+2n+1 \\ n=n+1 \\ u=u+2n+1 \\ n=n+1 \\ u=u+2n+1 \end{matrix}$$

Traduire cet algorithme en programme

$x \leftarrow 7$
$y \leftarrow 2$
$z \leftarrow 2x-y$
$y \leftarrow 2y-3z$
$x \leftarrow 5z+y-4x$
Afficher x, y, z

3) Dans le dernier programme, changer les valeurs initiales de x et y.
   Qu'observe-t-on ? Pouvez-vous l'expliquer ?

4 Les types prédéfinis

Nous présentons ici quelques types d'usages les plus courants. Il en existe d'autres que nous verrons par la suite à l'occasion d'exercices.

4.1 Les types numériques

Le type **int** pour les entiers relatifs.

type(2*3)	le résultat d'opérations de base +, -, *, /, //, **
a=int('2')	entre deux variables de type int est de type int
type(a)	

Le type **float**, un ordinateur ne connait pas les nombres réels ! !

type(3.0) **type(5/2)** **type(5//2)**	On approche les nombres réels par un certain type de nombres. appelés nombres **flottants**
a=int(3.0) **type(a)**	L'instruction **int** permet de convertir certains nombres de type flottant en type int.

Exercice 3

1) `Taper dans l'ordre les instructions : 0.1+0.2, puis 0.1+0.2-0.3.`
`Comment expliquez-vous ce résultat ?`

2) `Que donne l'addition d'un nombre de type int et d'un nombre de type float ?`

4.2 Le type booléen

Ce type permet d'effectuer des tests logiques. Il existe deux constants **True** et **False** qui permettent de caractériser si une propriété donnée est vrai ou fausse.

Pour cela, on utilise des opérateurs comme

Opérateurs de comparaison teste →	== l'égalité	< ou > la stricte inégalité	<= ou >= l'inégalité large	!= la différence	
Les connecteurs logiques	and et	or ou	not négation	is égalité	in appartenance à un conteneur

Exercice 4 **Variables booléennes**

1) `Taper les instructions :` $i = 3$; $i == 2$, $i == 3$, $4 * i! = 13$, $5 * i! = 10$.
`Expliquer les résultats.`

2) `Quels seront les résultats de ces instructions ?`
`(On pourra les taper pour vérifier vos réponses dans un second temps)`

$$j = i^2; \quad i == j, \quad j < 3 * i, \quad j - 1 > 2 * i, \quad j <= \texttt{sqrt}\ (99)$$

Exercice 5 — **Opération et/ ou sur les variables booléennes**

1) Compléter les tables logiques suivantes :

ET	V	F
V		
F		

OU	V	F
V		
F		

2) Que vont donner les instructions de deux premiers programmes ?

$a = 3$
$a = -1$
$b = 1$
$a^4 == b^4$
$a^5 == b^5$

$a = 3$
$b = 5$
$c = 2$
$d = b^2 - 4ac$
$d > 0$

Dans quel cas, le programme suivant affiche True ?

```
x=int(input('Entrer x='))
y=int(input('Entrer y='))
z=int(input('Entrer z='))
print(((x<z) and (z<y)) or
(y<=x))
```

Dans quel cas, le programme suivant affiche True ?

```
a=float(input("a= "))
b=float(input("b= "))
c=float(input("c= "))
print(((a<b) and (a<c)) or
(b<c))
```

3) Proposer un programme qui demande deux nombres et qui renvoie **True** si le premier est au moins quatre fois plus grand que le second.

4) Proposer un programme qui demande trois nombres et qui renvoie **True** si les deux premiers sont inférieurs au troisième nombre ou si la somme des deux premiers est inférieur au double du troisième nombre.

4.3 Le type string : les chaine de caractères

Une chaîne de caractères est une suite quelconque de caractères (lettres, chiffres, ponctuation...) délimité par des apostrophes du type 'Python v3.7.1' ou "Python v3.7.1". C'est tout simplement un bout de texte.

Pour Python, cela correspond au type **str** (string ou chaîne).

Plusieurs opérations sont possibles sur les chaînes de caractères. On en donne quelques exemples :

```
x='chien'
type(x)
len(x)
```

```
y="chien"
x is y
```

```
a='Ce' z='aboie',t='plutôt fort !'
a+' '+x+' '+z+' '+t
```

```
print(x[4]+x[2]+x[0]+x[1]+x[3])
n=len(x)
print(x[0].upper()+x[1]+x[n-3]+x[n-1]+x[n-2])
```

Des conversions sont possibles d'un type vers un autre type :

par exemple, str(2) donne la chaîne '2' considérer par Python comme du texte. A l'inverse, int('2') ou float('3.15') convertit ces textes en nombres entiers ou flottants.