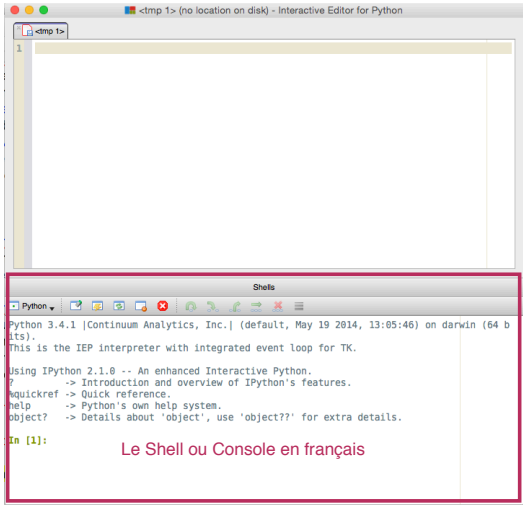
Partie 1 : Calcul dans la console : types de nombres et opérations arithmétiques

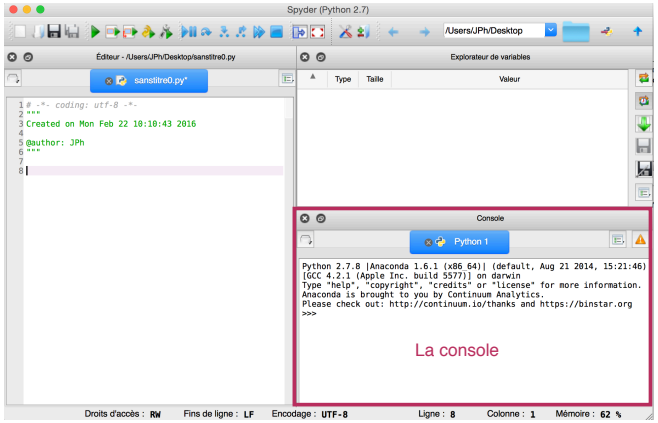
1 La console (ou *shell* en anglais)

La console est une fenêtre dans l'environnement de développement (ou E.D.I.).

• La console dans l'E.D.I. PYZO :



• La console dans l'E.D.I. SPYDER :



La console est essentielle ; elle doit toujours rester ouverte. Elle permet l'intéraction avec un programme et aussi de saisir directement des commandes.

1. Dans l'E.D.I. ouvert, fermer et déplacer des fenêtres pour obtenir la même apparence que ci-contre.

Dans la console, les commandes sont saisies **au prompt**.
Le prompt est représenté par 3 flèches (console python) : `>>>`
ou par un numéro de commande entre crochets (console ipython) : `In [1]`.

Une fois saisie une commande au prompt, appuyer sur la touche ENTRÉE pour l'exécuter. Si un résultat est renvoyé, il l'est à la ligne suivante, dans la console ; une nouveau prompt apparaît à la ligne suivante.

Exemple.

```
>>> 1+1
2

>>>
```

ou

```
In [1]: 1+1
Out[1]: 2

In [2]:
```

2. Exécuter dans la console la commande précédente.

2 opérateurs arithmétiques

Pour effectuer des calculs, on utilise les opérateurs arithmétiques :

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Puissance

Les règles de priorité des opérations sont, par priorité décroissante :

`**` (puissance) `*` et `/` (produit et quotient) `+` et − (addition et soustraction)

Pour imposer les priorités des opérations, on utilise les deux parenthèses `(` et `)` :

```
>>> (1-3) * 15
-30
```

Remarque : Entre un opérateur et ses deux opérandes, on peut insérer aucun ou plusieurs espaces :

```
>>> ( 1 - 3 )  *  15
-30
```

Seules les parenthèses `(` et `)` sont autorisées ! On ne peut pas utiliser des corchets `[, ]` ou des accolades {, }.

3. Taper la dernière commande en remplaçant les parenthèses par des crochets et par des accolades. Que se passe-t-il ?

4. Obtenir de deux façons différentes le résultat du calcul de 2/3.

3 Types entier et flottant

• Les entiers relatifs, s'écrivent naturellement : `0`, `1`, `2`, ..., −`1`, −`2`. Ces nombres sont de type entier.

• Les nombres à virgules, s'écrivent à l'aide du séparateur décimal, qui est le point `.`, ou à l'aide de la notation scientifique `1e10` ou `1E10`. Ces nombres sont de type flottant (ou nombres à virgule flottante).

S'il n'y a que des zéros avant ou après le séparateur décimal, ces zéros peuvent être ignorés.

5. Écrire de 3 façons différentes le nombre 0,5 (sans utiliser aucune opération arithmétique.)

6.a) Taper `>>> 2 ** 0.5` pour obtenir l'écriture décimale (approchée) de $\sqrt{2}$.
b) Élever le nombre obtenu au carré. Que constate-t-on ?
Comment expliquer ce résultat ?

7.a) Soit $a = \frac{1+\sqrt{5}}{2}$. Calculer dans la console a^2 et $a+1$. Que constate-t-on ?
b) Sauriez-vous expliquer le résultat obtenu ?
c) Déterminer tous les nombres vérifiant la même propriété.

4 D'autres opérateurs arithmétiques

Les nombres de type entier, admettent deux autres opérateurs :

Opérateur	**Opération : à déterminer**
`//`	
`%`	

8. Effectuer quelques essais, pour en déduire les opérations qu'effectuent les opérateurs `//` et `%` avec des opérandes de type entier, et compléter le tableau ci-dessus.

9. Appliquer l'algorithme d'Euclide pour calculer $PGCD(918, 221)$.

10. Comment à l'aide de ces opérateurs obtenir le chiffre des centaine d'un entier ? Le chiffre des milliers ? Des dizaines de milliers ?

Partie 2 : Fonctions mathématiques ; notion de variable

5 Le module `math`

Par défaut python ne connaît, en dehors des opérateurs arithmétiques, aucune fonction ou constante mathématiques : sans bibliothèque l'appel de `cos(1)` ou `pi` produit une erreur.

```
>>> # Sans bibliothèque python est ignorant en math :
... cos(1)
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: name 'cos' is not defined
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
```

Il suffit de faire appel à la bibliothèque '`math`' de fonctions mathématiques prédéfinies. C'est un *module* :

```
>>> from math import *          # importation des fonctions de la biliothèque
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
>>> acos(-1)
3.141592653589793
```

```
>>> sqrt(2)                     # racine carrée
1.4142135623730951
>>> e
2.718281828459045
>>> log(e)                      # logarithme neperien
1.0
>>> exp(1)                      # exponentielle
2.718281828459045
>>> log(256,2)                  # logarithme en base 2
8.0
>>> log(1000,10)                # logarithme en base 10
2.9999999999999996
```

```
>>> # lui préférer :
... log10(1000)                 # logarithme base 10 plus précis
3.0
```

```
>>> fabs(-3)                    # valeur abolue
3.0
>>> floor(pi)                   # partie entière
3.0
>>> floor(-pi)
-4.0
```

11) En utilisant les fonctions du module `math` :

1. Obtenir la partie entière de $e^{\pi} \times \ln(1/2)$.
2. Quel est le plus grand entier n pour lequel $2^n \leqslant 10^6$?

6 Variables

6.1 Notion de variable

• La notion de variable est essentielle en programmation.

• Elle permet de stocker en mémoire des valeurs, et de les utiliser et modifier à volonté au sein d'un programme.

• La valeur d'une variable évolue au cours de l'exécution d'un programme, en fonction du déroulement du programme et selon ses instructions.

Une **variable** a :

– un **identifiant** : pour nous c'est son nom, qui permet de manipuler la variable au sein d'un programme ou d'une instruction (en mode console). Le nom est composé de lettres et de chiffres (et du symbôle '_'), et ne doit pas débuter par un chiffre. Son nom doit être clair pour faciliter la relecture du programme.

– Un **contenu**, c'est sa valeur.

Elle est stockée dans la mémoire sous forme d'un nombre en écriture binaire, pour l'interpréter correctement, chaque valeur est munie d'un **type** : entier (relatif), flottant (réel...), complexe, chaîne de caractère, etc...

6.2 Affectation

En python la définition (déclaration) d'une variable se fait à l'aide d'une affectation : `variable = valeur`.

```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
>>>
```

L'opération principale pour une variable est l'affectation représentée par le symbôle `=`. Elle permet de déclarer (c'est à dire définir) une variable, en lui affectant une valeur. Elle permet aussi d'en modifier la valeur.

Dans la console, il suffit de saisir le nom d'une variable, puis d'appuyer sur entrée pour obtenir sa valeur.

```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
```

Lors d'une affectation l'expression à droite du symbôle `=` est évaluée, avant d'être affecté à la variable dont le nom figure à gauche du symbôle `=`.

Le membre de gauche de `=` ne peut être que le nom d'une variable. Si elle n'existe pas encore la variable sera créée lors de l'affectation (= déclaration).

Si un même nom de variable figure des 2 côtés de `=`, la valeur de celle de droite est celle AVANT l'affectation, celle de gauche est celle APRES l'affectation.

L'affectation `x = x + 1` n'a rien à voir avec l'équation mathématiques impossible $x = x + 1$.

12) Pour chacun des cas suivants, déclarer une variable `a = 1` et répéter cinq fois l'affectation décrite, avant de vérifier quelle est la nouvelle valeur de la variable `a` :

1. `a = a + 2`
2. `a = a/2`
3. `a = 2*a + 1`

Dans chaque cas, quelle valeur devrait prendre `a` si l'on répète 100 fois l'affectation ? Justifier. En obtenir une valeur approchée dans la console.

```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
```

L'instruction :

`variable †= expression`

est équivalente à l'instruction :

`variable = variable † expression`

où † désigne n'importe quelle opération : `+, -, *, /, //, %, **, etc....`

13) Refaire l'exercice 2.1) et 2.2) en utilisant cette fonctionnalité.

14) (Algorithme de Babylone)

1. Déclarer la variable `a = 2`. Puis appliquer plusieurs fois l'affectation `a = a/2 + 1/a`. Qu'obtient-on après quelques itérations ? Le vérifier.

6.3 Échange du contenu de deux variables

Soient `a` et `b` deux variables. Une opération courante consiste à échanger les valeurs des deux variables `a` et `b`.

15) Proposer une suite d'affectation qui permet d'échanger le contenu de deux variables quelconques `a` et `b`.

16) Déclarer deux variables `a` et `b` à valeurs numériques (on prendra deux valeurs différentes). Puis saisir les affectation suivantes :

```
a = a + b
b = a - b
a = a - b
```

Que sont devenues les valeurs de `a` et `b` ?
Expliquer.

6.4 Une particularité de l'affectation de variables sous python

Python permet en une seule instruction d'affectation ('=') d'affecter plusieurs variables :

```
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

Les variables, à gauche de l'instruction '`=`' d'affectation, sont séparées par des virgules, de même que les valeurs, à droite de '`=`', qui doivent être en même nombre, et sont affectées de gauche à droite.

Bien noter que durant une affectation multiple les valeurs (à droite du `=`) sont celles avant l'appel de l'instruction. Ainsi l'instruction `a,b = b,a` échange les valeurs de 2 variables `a` et `b` :

```
>>> a = 1 ; b = 2
>>> a, b = b, a
>>> a
2
>>> b
1
```

17) Suite de Fibonacci.

La suite de Fibonacci décrit l'évolution d'une population (idéale) de lapin en l'absence de prédateurs. C'est la suite dont le n-ième terme correspond au nombre de paires de lapins au n-ième mois. Dans cette population (idéale), on suppose que :

- au (début du) premier mois, il y a juste un couple de lapereaux ;
- Il leur faut un mois pour atteindre leur maturité sexuelle, et la gestation dure 1 autre mois : les lapereaux ne procréent donc qu'à partir du (début du) troisième mois ; chaque (début de) mois, toute paire susceptible de procréer engendre effectivement une nouvelle paire de lapereaux ;
- les lapins ne meurent jamais (la population est idéale) donc la suite de Fibonacci est croissante).

Notons F_n le nombre de couples de lapins au début du mois n. Jusqu'à la fin du deuxième mois, la population se limite à un couple (ce qu'on note :

$$F_1 = F_2 = 1$$

Dès le début du troisième mois, le couple de lapins a deux mois et il a engendré un autre couple de lapins ; on note alors

$$F_3 = 2$$

Plaçons-nous maintenant au mois n n et exprimons le nombre de couples deux mois plus tard, au mois $n+2$: F_{n+2} désigne la somme des couples de lapins au mois $n+1$ et des couples nouvellement engendrés.
Or, n'engendrent au mois $n+2$ que les couples pubères, c'est-à-dire ceux nés au moins deux mois auparavant. On a donc, pour tout entier n :

$$F_{n+2} = F_{n+1} + F_n$$

On pose alors $F_0 = 0$, de manière que cette équation soit encore vérifiée pour $n = 0$.

1. Déclarer deux variables `u = 0` et `v = 1`. Quelles instructions permettent, lorsque `u` et `v` ont pour valeurs F_n et F_{n+1}, de changer leurs valeurs respectivement en F_{n+1} et F_{n+2}.
2. Obtenir dans la console le nombre de couples après 5 mois, après 1 an.
3. Vérifier sur les valeurs obtenues que :

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

4. Sauriez-vous le démontrer ?