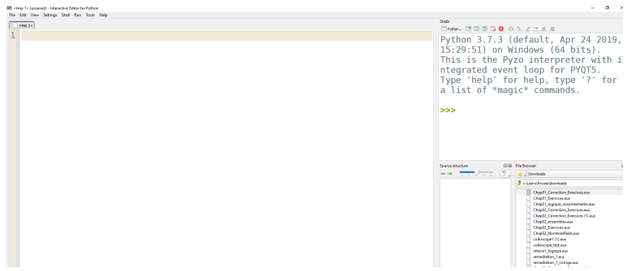
TP d'informatique n° 1

— Introduction à Python —

Nous travaillerons cette année avec **Pyzo**, qui est l'environnement retenu par les concours.
C'est un logiciel libre et gratuit, téléchargeable à l'adresse `http://www.pyzo.org/`).
Il permet d'exécuter des commandes écrites en **langage Python**.

1 Présentation de Pyzo

Le logiciel est présent dans le dossier informatique, sur le bureau de votre ordinateur, avec comme logo  .
Lorsqu'on lance **Pyzo**, voici ce qu'on obtient.



1.1 Le shell Python

Le **Shell** correspond au cadre en haut à droite. Chaque ligne commence par trois chevrons `>>>` appelés le **prompt**. On y tape des commandes qui sont exécutées immédiatement, un peu à la manière d'une calculatrice.

Exercice 1. Opérations de base
Taper les instructions suivantes et observer :

```
>>> 3 + 5
>>> 3 - 5
>>> 3 * 5
>>> 2^5
>>> 2**5
>>> 2e3
>>> 60/7
```

1. Que représente l'opérateur ** ?
2. Quel symbole représente le séparateur décimal ?
3. Comment est notée l'écriture scientifique de $a.10^n$?

1.2 Enrichir la calculatrice

Si on essaye des calculs un peu plus poussés, on se rend compte que l'on est rapidement limité.
Par exemple,

```
>>> sin(pi)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'sin' is not defined
```

Python râle car il ne connaît pas la fonction sinus (c'est plutôt limité comme calculatrice…).
On peut facilement enrichir les outils disponibles en faisant appel à des **bibliothèques ou modules**.

Nous utiliserons les modules suivants :

- Le module `math` : contient les fonctions mathématiques usuelles ainsi les constantes comme pi ou e.
- Le module `random` : utile pour générer des nombres aléatoires.

Pour utiliser ces bibliothèques, on tape l'instruction suivante :

```
>>> from nom_du_module import *
```

Revenons à l'exemple.

```
>>> from math import *
>>> sin(pi)
1.2246467991473532e-16
>>> cos(pi/2)
6.123233995736766e-17
```

Les résultats obtenus sont-ils logiques ?
Chacune des fonctions possède une aide intégrée au langage Python qui explique comment utiliser la fonction.
Par exemple,

```
>>> help(sin)
Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

Exercice 2. Différents passages d'un réel à un entier
En utilisant l'aide et en faisant des essais, expliquer l'usage des fonctions :

`floor`, `ceil`, `round` et `trunc`.

Exercice 3. Aide à la frappe
En utilisant l'aide à la frappe (quand on tape le début du nom d'une fonction connue de Python, il propose le nom complet), déterminer :

1. quelle fonction permet de calculer la racine carrée d'un réel ?
2. que devez-vous taper pour calculer (42)! ?
3. quelle est la commande qui correspond au logarithme népérien ? et au logarithme décimal ?

2 Manipulation des variables

2.1 Dénomination des variables

Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

- un nom de variable est une séquence de lettres (a-z, A-Z) et de chiffres (0-9), qui doit toujours commencer par une lettre.
- seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que `$`, `#`, … sont interdits, à l'exception du caractère `_` (tiret du 8 = underscore).
- la casse est significative (les caractères majuscules et minuscules sont distingués).
- il y a 33 « mots réservés » ci-dessous qui sont utilisés par le langage Python et ne peuvent pas servir de nom de variable :

```
and     as      array    break    class    continue   def    del
elif    else    except   False    finally  for        from   global
if      import  in is    lambda   None     nonlocal   not    or
pass    raise   return   True     try      while      with   yield
```

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules. Il s'agit d'une simple convention, mais elle est largement respectée. Pour aider à la lisibilité du code informatique, il est conseillé de donner des noms significatifs aux variables, par exemple : `score`, `nb_colonnes`, `Liste_notes` (ou encore `ListeNotes`).

2.2 Affectation d'une valeur à une variable

L'affectation d'une valeur dans une variable se fait avec le symbole =.
La variable est écrite à gauche et reçoit la valeur écrite à droite. Par exemple :

```
>>> score = 0
```

La valeur de droite peut être le résultat d'un calcul et/ou utiliser la valeur d'une autre variable.

```
>>> y = 5 + 4*score
```

Le symbole = n'a donc pas la même signification qu'en mathématique.
En particulier, la commutativité n'est pas possible.

```
>>> 0=score
  File "<console>", line 1
SyntaxError: can't assign to literal
```

En revanche, l'affectation permet l'opération d'**incrémentation** :

```
>>> score = score + 1

>>> print(score)
1
>>> score = score + 1

>>> print(score)
2
```

Cela signifie que Python évalue (calcule) la valeur de droite à partir de l'actuelle valeur de la variable `score` et en déduit la nouvelle valeur de cette variable.
Attention, l'incrémentation d'une variable doit toujours être précédée d'une **initialisation** de cette variable. Sinon le message d'erreur est le suivant :

```
>>> z = z + 1
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'z' is not defined
```

Enfin l'affectation peut être **multiple** en séparant les variables et les valeurs par des virgules :

```
>>> a, b = 0, 5
```

L'affectation multiple (qui n'existe pas dans tous les langages) permet d'écrire très facilement l'échange de deux valeurs :

```
>>> print(a,b)
0  5
>>> b, a = a, b

>>> print(a,b)
5  0
```

3 Les différents types de variables

En mémoire, toutes les variables sont stockées en binaire, c'est-à-dire sous la forme d'une chaine de **0** et de **1**.
Cependant, suivant ce que représente la variable, elle ne sera pas codée de la même manière dans la mémoire de l'ordinateur, n'occupera pas la même place en mémoire et les opérations possibles avec celle-ci seront différentes. Nous allons présenter ici quelques types de variables et leurs propriétés. Nous en verrons d'autres lors des prochains TP.
L'instruction `type` renvoie le type d'une variable.

3.1 Les entiers

Les nombres entiers relatifs sont stockés sous le format appelé **integer** noté `int` par Python.
Pour les entiers Python réserve la place en mémoire nécessaire pour enregistrer la valeur exacte de l'entier : il n'y a pas d'arrondis sur les entiers.

```
>>> 2**64
18446744073709551616
>>> factorial(42)
1405006117752879898543142606244511569936384000000000
```

Les opérations possibles sur les entiers sont :

- les opérations usuelles : +, −, *, **
- la division euclidienne : $a//b$ donne le quotient de la division euclidienne de a par b
  $a\%b$ donne le reste de la division euclidienne de a par b

```
>>> a = 5

>>> type(a)
<class 'int'>
>>> 60//7
8
>>> 60%7
4
```

Exercice 4. Chiffre des centaines

1. Proposer une instruction qui donne la valeur du nombre s'écrivant en base dix sous la forme abc où a, b et c sont trois entiers strictement inférieurs à 10.
2. Proposer un algorithme (suite d'instructions) qui permet de déterminer le chiffre des centaines d'un entier n. Le taper en une ligne dans le shell.

3.2 Les flottants

Les nombres réels sont stockés sous le format appelé **flottants** noté `float` par Python. Ils sont enregistrés sous leur écriture scientifique ($mantisse \times 10^{exposant}$) et la place en mémoire qui leur est attribuée est toujours de 64 bits.

La conséquence immédiate est que les valeurs des réels sont des valeurs approchées. Python réalise un arrondi sur le dernier bit de mémoire qui correspond à la 16 ème décimale de la mantisse.
On retient que tous les calculs sur les réels se font avec une précision limitée. Cela peut donner des résultats surprenants. Nous avons déjà rencontré le cas de `pi` :

```
>>> sin(pi)
1.2246467991473532e-16
```

Et voici une autre résultat intéressant :

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Les flottants peuvent correspondre à des valeurs entières mais ils sont alors affichés avec une partie décimale nulle.

```
>>> type(2)
<class 'int'>
>>> type(2.0)
<class 'float'>
```

Les opérations usuelles sur les réels sont : +, −, *, **, /, exp, log, log10, ...
Attention : la division donne toujours un réel. Tester en demandant le type de 10/5. Rappel : si on est sûr que le résultat est un entier on utilise : 10//5.

3.3 Les booléens

Une variable booléenne ne peut prendre que deux valeurs : `True` ou `False` (sans guillemets et avec une majuscule). Le type booléen est noté `bool` par Python.
Ce sont des variables logiques et les opérateurs logiques usuels s'appliquent :

- l'opérateur ET via la commande `and`,
- opérateur OU via la commande `or`,
- opérateur NON via la commande `not`.

Par exemple,

```
>>> a, b = True, False

>>> a and b
False
>>> a or b
True
>>> not a
False
>>> type(a)
< class :'bool'>
>>> type(a or b)
< class :'bool'>
```

3.4 Autres types de variables

Voici d'autres types de variable que nous découvrirons en détail dans les prochains TP.

- **Les chaînes de caractères (type : 'str')** pour manipuler des lettres, des mots, des phrases...
- **Les listes (type : 'list')** pour stocker plus d'informations dans une seule variable.
- **Les tableaux (type : 'array')** pour travailler sur les images.
- **Les dictionnaires (type : 'dict')** pour associer des clés et des valeurs.

4 Fonctions

4.1 Éditeur et environnement de développement intégré

Le shell Python est très vite limité si l'on souhaite écrire un programme un peu évolué (et qui dépasse quelques lignes) :

- si on a fait une faute, pour la corriger, il faut tout reprendre ;
- il est difficile de contrôler l'exécution d'un programme ;
- il n'y a pas d'aide « en direct » pour comprendre comment utiliser une fonction...

La première amélioration est d'utiliser un éditeur externe pour taper le programme. Quelques avantages :

- on tape le programme, puis on l'exécute. Si on veut le modifier, il suffit de changer le bout de code souhaité sans toucher au reste. C'est très très utile dès que la taille du programme dépasse quelques lignes ;
- on peut sauvegarder le programme pour y revenir plus tard ou pour le partager ;
- on peut ajouter des commentaires au code qui n'auront aucune influence sur l'exécution du programme, mais qui faciliteront la relecture ;
- la coloration syntaxique facilite la lecture en changeant l'aspect des mots-clefs du code ;
- la complétion automatique permet de compléter les commandes ou les noms de variables après en avoir tapé les premières lettres...

Pyzo est un éditeur simple. Dans la fenêtre, on a, par défaut :

- ◇ un shell ;
- ◇ un éditeur (avec des onglets pour ouvrir plusieurs fichiers en même temps) ;
- ◇ un explorateur de fichiers (File Browser) ;
- ◇ un affichge de la struture des programmes (Source structure).

4.2 Syntaxe

La syntaxe générale d'une fonction en Python est la suivante :

```
1 def nom_de_la_fonction(paramètre1, paramètre2, ...):
2
3     ...
4     bloc de commandes
5     ...
6
7     return sortie1, sortie2, ...
```

Remarques :

- On remarque le décalage absolument indispensable qui permet d'indiquer au langage quelles sont les commandes qui appartiennent au corps de la fonction (en informatique, on parle d'**indentation**). Lorsque la fonction est finie, on reviendra simplement au début de la ligne.
- Si la fonction possède plusieurs paramètres d'entrée ou de sortie, on séparera le nom des variables par une virgule.
- Le corps de la fonction peut être aussi compliqué que nécessaire : il peut contenir des branchements conditionnels, des boucles `for` ou `while`, faire appelle à d'autres fonctions...

4.3 Exemple de fonction

Voici l'exemple d'une fonction qui calcule le discriminant d'un ax^2+bx+c :

```
def discriminant(a,b,c):
    d = b**2-4*a*c;
    return d
```

On peut ensuite faire appel à la fonction.

```
>>> discriminant(1,2,3)
-8
```

On peut aussi créer une variable dont la valeur est calculée par une fonction.

```
>>> delta = discriminant(1,2,3) # la variable delta reçoit la valeur renvoyée par la
    fonction discriminant

>>> print(delta) # pour faire afficher la variable delta
-8
```

Exercice 5. Fonction inconnue

```
def fonction_inconnue(m,n):
    a=(m+n)/2
    return a
```

1. Que va rendre l'instruction `fonction_inconnue(3,5)` ?
2. Proposer une fonction qui calcule l'aire d'un rectangle de côtés m et n.
3. Et pour un carré ?

Exercice 6. Produits

1. Écrire une fonction `prod2` qui reçoit en entrée a et b et qui renvoie le produit ab.
2. En utilisant la fonction `prod2`, écrire une fonction `prod3` qui reçoit en entrée a, b et c et qui renvoie le produit abc.

5 Bibliothèque Turtle

Exercice 7. Dessiner avec Turtle

1. Importer le module turtle. Il permet de réaliser des dessins.
   Voici quelques fonctions disponibles

Commandes	Alias	Explications	Commentaires
forward(nbre)	fd(nbre)	avance de nbre pixels	nbre est un entier
backward(nbre)	bk(nbre)	recule de nbre pixels	nbre est un entier
left(angle)	lt(angle)	tourne de angle degré	sens trigo
right(angle)	tr(angle)	tourne de angle degrés	sens rétrograde
penup()	pu()	monte le stylo	
pendown()	pd()	descend le stylo	
speed(n)		change la vitesse du tracé	de 0(lent) à 10(rapide)

2. Écrire une fonction `rectangle` qui dessine un rectangle en 4 instructions.