

TP 1 Python – Introduction

Nombres et variables

1. Un petit aperçu du monde de l'informatique.

1.A) Mon ordinateur : qu'est-ce qu'il a dans le ventre ?

On peut considérer simplement qu'un ordinateur¹ est composé de trois ensembles :

- le **microprocesseur** – c'est le cœur de l'ordinateur, c'est lui qui effectue les opérations demandées par l'utilisateur. Il n'effectue que des calculs sur des **bits**, c'est à dire des 0 ou 1.
- la **mémoire vive**, qui permet de stocker les données et les programmes utilisés par le microprocesseur.
- les **périphériques** (écran, imprimantes, disque dur, clavier, souris, ...) – tout le reste, qui nous permet de dialoguer avec l'ordinateur et de conserver des informations une fois que celui-ci est éteint.

1.B) Et *Python*, dans tout ça ?

Un ordinateur ne fonctionne qu'à partir de bits de donnée, i.e. de 0 et de 1. Pour nous, c'est absolument incompréhensible... il faut donc un traducteur entre le « langage machine » (des suites de 0 et de 1) et l'utilisateur : c'est ce que l'on appelle l'**interpréteur** ou le compilateur.

Les ordinateurs sont incapables d'initiatives, il faut donc leur donner des instructions précises et sans ambiguïtés. Le langage dans lequel l'utilisateur donne ses instructions à l'interpréteur ne peut donc pas être le langage courant : ainsi viennent les **langages de programmation**, qui utilisent une syntaxe *précise* et *univoque*.

Algorithmes et programmes

Un **algorithme** est une suite finie d'étapes à appliquer à des données dans l'ordre, pour arriver à un résultat souhaité. Il s'écrit en "langage naturel" : le but est d'être compris par ceux à qui il s'adresse !

Un **programme** est la traduction d'un algorithme dans un langage de programmation. Il est à la fois compréhensible par la machine (via un interpréteur ou un compilateur) et par l'homme (via les cours d'informatique!).

Remarque. Une recette de cuisine est un algorithme, tout comme les instructions pour monter un meuble.

Il existe différents langages de programmation. Celui qui a été choisi en BCPST est le même qu'au lycée : *Python*². Nous allons donc (ré)apprendre cette année la syntaxe du Python, les instructions de bases, ainsi que certaines méthodes de modélisation. Mais avant de se lancer, retenez que la syntaxe des langages de programmation est très rigide et devra être respectée **à la ponctuation près** : c'est très facile de faire planter un ordinateur à cause d'une parenthèse mal placée...

Les spécificités du langage Python :

- Python est assez concis, ce qui est très appréciable pour des novices en programmation : moins on doit en écrire, moins on risque de faire une erreur.
- C'est un langage modulaire : la définition du langage est très succincte, mais il existe de nombreux modules (des annexes) disponibles pour une utilisation plus spécifique à partir du moment où on demande à les ouvrir.
- Tout objet utilisé dans le code est identifié dans une catégorie particulière (sa nature), qu'on appelle son **type**. Le sens des opérations effectuées ne sera pas le même selon le type de l'objet auquel on l'applique.
- Il est à syntaxe positionnelle, ce qui signifie que l'**indentation** (alignement effectué avec la touche « Tab ») fait partie du langage. Il faudra donc faire attention à aligner ou décaler les lignes selon ce que l'on veut faire.

1. Ou une tablette, un smartphone, une calculatrice, ...

2. Créé dans les années 90 et baptisé ainsi en hommage aux *Monty Python*

1.C) Pyzo et son interface.

Il existe plusieurs versions de *Python* (c'est le même langage, mais dans les anciennes versions quelques fonctions sont différentes ou n'existent pas encore).

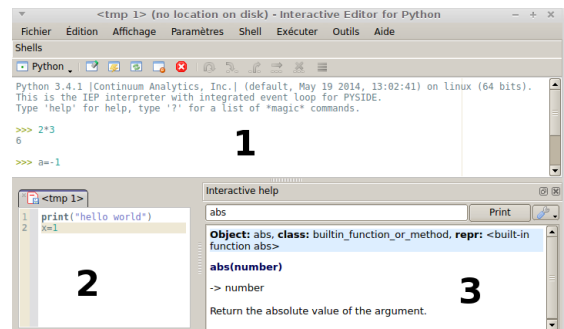
Nous travaillerons avec la version la plus récente (choisie par le concours) : Python 3.

Il existe plusieurs environnements (des logiciels), permettant de travailler avec Python (le langage reste le même, mais la présentation change, de même que vous pouvez naviguer sur internet via Firefox ou Google Chrome). Nous utiliserons l'environnement *Pyzo* : c'est celui qui a été choisi par le concours.

Vous devez donc télécharger le logiciel *Pyzo* sur votre ordinateur personnel. Vous trouverez les instructions pour le faire sur le site de la classe, dans la section « Informatique ».

Une fois Pyzo installé et lancé, vous obtiendrez une fenêtre divisée en plusieurs parties :

1. La **console** (=shell), sur laquelle vous pouvez directement entrer une ligne de commande, puis demander son exécution en tapant sur la touche **Entrée** (comme pour une calculatrice). Elle sera utilisée pour des instructions courtes (calculs simples, tests).
2. L'**éditeur**, sur lequel vous pouvez écrire plusieurs lignes de commande (on parle alors de **script**), les enregistrer, puis demander leur exécution. Les résultats sont alors affichés dans la console. C'est ce que nous utiliserons majoritairement.
3. D'autres fenêtres proposent des options dont nous reparlerons.



2. Les nombres

2.A) Opérations élémentaires

Python peut déjà servir de simple calculatrice. On peut donc effectuer les opérations élémentaires suivantes sur des nombres :

Opération mathématique	Instruction <i>Python</i>
Addition	+
Soustraction	-
Multiplication	*
Puissance	**
Division	/
Quotient de la division euclidienne	//
Reste de la division euclidienne	%

Exemple

```
In [1]: 3+1
Out[1]: 4
In [2]: 5/3
Out[2]: 1.6666666666666667
In [3]: 16//3
Out[3]: 5
In [4]: 16%3
Out[4]: 1
```

```
In [5]: 1/0
```

```
-----
ZeroDivisionError      Traceback (most recent call last)
<ipython-input-7-05c9758a9c21> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

- Remarque.* • Les règles de priorité sont les mêmes que sur vos calculatrices. Il est cependant *très fortement* conseillé de mettre des parenthèses dans les gros calculs pour plus de lisibilité.
- ⚠ De manière générale en informatique, on écrit les nombres décimaux à l'anglaise : avec un point et pas une virgule ! La virgule a un tout autre sens : 1,2 désigne le couple (1; 2).
 - ⚠ L'opération $^$ existe en *Python*, mais ça n'est pas du tout la puissance !

2.B) Les types utilisés pour les nombres

Les nombres en Python, comme tous les objets manipulés, sont toujours identifiés comme faisant partie d'un type particulier. On utilisera les types suivants pour les nombres réels :

- `int` (*integer*) pour les nombres entiers, par exemple
- `float` pour les nombres réels/décimaux³, par exemple

Remarque. • On utilise toujours l'instruction `type(x)` pour connaître le type d'un élément.

- Les opérations numériques sont compatibles entre les types : on peut calculer par exemple `3.5+1`, et le résultat sera `4.5`. De manière générale, le résultat est du type le plus "large" parmi les types présents dans l'expression.
- On peut convertir un nombre d'un type dans l'autre type, à l'aide des instructions `int(x)` et `float(x)`.

⚠ L'instruction `int(x)` se contente de tronquer juste avant la virgule, mais attention, ce n'est pas la fonction partie entière⁴ !

Exemple

- | | |
|--|--|
| • <code>type(3.5)</code> renvoie | • <code>type(-2)</code> renvoie |
| • <code>type(2.)</code> renvoie | • <code>type(0.5 + 2.5)</code> renvoie |
| • <code>float(1)</code> renvoie | • <code>int(-10.000)</code> renvoie |
| • <code>int(2.8)</code> renvoie | • <code>int(-2.8)</code> renvoie |

3. Les variables

3.A) Définition d'une variable

Lorsque l'on veut garder en mémoire des informations, on utilise des variables. Une variable est caractérisée par son **nom**, sa **valeur** et son **type**.

On peut imaginer une variable informatique comme une « boîte » étiquetée avec son nom, et contenant une valeur, que l'on utilisera et/ou remplacera au fur et à mesure des calculs.

⚠ Les variables en informatique diffèrent de celles en mathématiques car elles doivent nécessairement avoir une valeur **explicite**. Python ne peut pas comprendre la phrase « Soit x un entier ».

3.B) Types d'objets couramment utilisés

Une variable peut contenir n'importe quel type d'objet manipulé par Python. Parmi les plus courants et ceux que nous utiliserons majoritairement :

- Des nombres (réels) bien sûr, du type `int` ou `float`.
- Du texte, appelé « chaîne de caractère » et toujours écrit entre guillemets en Python. Il est du type `str` (*string of characters*). Ex : `"Hello world !"`.

3. Un ordinateur a une mémoire finie, il ne peut donc en fait gérer (du moins de manière standard) que des nombres qui ont un nombre fini (et on devrait même plutôt dire petit) de décimales.

4. Dont nous parlerons à la fin de ce TP

- Les valeurs logiques **True** (vrai) et **False** (faux), du type **bool** (pour *booléen*⁵).
- Des listes de données, des nombres complexes, des dictionnaires... que nous verrons plus tard!

3.C) Affectation

On utilise le signe « = » pour affecter une valeur à une variable, par exemple : **A=2**.

Affectation :

- S'il n'existe pas encore de variable appelée A, cette instruction
- Si la variable A existe déjà, cette instruction

Exemple

Remplir le tableau suivant avec le nom, la valeur et le type de chacune des variables définies.

Instruction	Nom	Valeur	Type
<code>âge = 12</code>			
<code>température_celcius = 25.5</code>			
<code>message = "bonjour"</code>			

Remarque. • L'affectation s'écrit usuellement $A \leftarrow 2$ en langage algorithmique.

- ⚠ L'affectation d'une variable n'est pas symétrique! Elle **se lit de droite à gauche** : Python effectue d'abord les calculs à droite du signe « = », puis stocke le résultat dans la variable indiquée à gauche. Ainsi, `2=A` renvoie un message d'erreur!
- Une variable doit être définie pour pouvoir être utilisée.
Par exemple l'instruction `2+x` renvoie le message d'erreur

Exemple

1. On tape les lignes de commande suivante :

`A = 1000`

`A = A * 1.05`

Quelles sont, dans l'ordre, les opérations effectuées par Python?

Quels sont la valeur et le type de la variable A à la fin des instructions?

Peut-on récupérer la valeur initiale de A?

2. On tape dans cet ordre la suite d'instructions :

`a = 1000`

`b = 700`

`B = 500`

`c = a + b`

`a = 1500`

`C = a + B`

Que valent à la fin les variables **a**, **b**, **c**, **B**, **C**?

3. On définit une suite récurrente par $u_0 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \frac{2u_n}{1 + u_n^2}$.

Proposer une suite d'instructions qui permette de calculer u_4 en ne définissant qu'une seule variable.

3.D) Affectation multiple

En Python, il est possible d'affecter des valeurs à plusieurs variables **en même temps**. Par exemple :

L'instruction `a, b, c = 1, 2.5, "blablabla"`

L'instruction `A, B = B, A` permet

5. Du mathématicien, logicien et informaticien du 19^e siècle George Boole.

3.E) Donner un nom à une variable : les règles

Le choix du nom d'une variable doit respecter les règles suivantes :

- Le nom doit commencer par une lettre.
- Le nom ne doit pas comporter d'espace.
- Les accents et les chiffres sont autorisés, mais aucun autre caractère spécial, à part le symbole « `_` » (utilisé pour remplacer l'espace).

Exemple

Dans les exemples suivants, barrer ceux qui ne sont pas licites :

`effectif_de_la_population`, `effectif de la population`, `2eff`, `eff1`, `eff_2`, `eff-1`, `Année`, `Taux_en_%`.

- Remarque.*
- **ATTENTION!** Python est **sensible à la casse**, ce qui signifie qu'il fait la distinction entre
 - Utiliser des noms **significatifs**, comme `age`, `annee`, `pourcentage`... plutôt que des noms comme `x`, `y`, `z`, `t`. Ce sera valable aussi pour les noms des fonctions.
De manière générale, votre code doit être **facilement compréhensible** par un de vos collègues!
 - Certains noms sont réservés, car ils sont déjà utilisés par le langage Python : `and`, `def`, `for`, `if`,...

4. Les modules

Python est un langage modulaire : un certain nombre de constantes et de fonctions sont définies dans le corps de base du logiciel et donc toujours accessibles. De nombreux autres outils existent, définis dans des modules spécifiques (des "bibliothèques spécialisées", si on veut). Il suffit d'importer le contenu de ces modules pour y avoir accès.

Par exemple le module `math` contient toutes les fonctions mathématiques que vous connaissez.

Il existe de nombreux modules qui peuvent être ajoutés à *Python* pour étendre ses fonctionnalités. Vous avez peut-être déjà utilisé `turtle` pour des dessins, mais nous utiliserons aussi `matplotlib` pour tracer des graphes, `numpy` pour gérer des tableaux de nombres, etc.

Pour utiliser les fonctionnalités d'un module, par exemple le module `math`, on utilise au choix les instructions suivantes :

1. `from math import *`
`from math import nom_fonction`
2. `import math`
`import math as nom_raccourci`

On peut alors utiliser les fonctions qui sont définies dans le module. Il faut faire cependant attention, car on n'accède pas à ces fonctions de la même façon selon la méthode d'importation choisie :

- avec la première méthode, le nom de la fonction est reconnu directement, par exemple, on tape `cos(0.3)` pour calculer le cosinus d'un angle ;
- avec la deuxième méthode, il est nécessaire de préciser le nom du module (ou son raccourci) avant la fonction, par exemple il faut taper `math.cos(0.3)` pour accéder à la fonction cosinus.

4.A) Le module `math`

Ce module permet d'avoir accès aux fonctions mathématiques courantes. Voici un tableau récapitulatif des fonctions les plus utilisées.

Fonctions usuelles

Constantes		exp		exp(0) →
		log		log(exp(1)) →
		sqrt		sqrt(4) →
		sin		sin(pi) →
		cos		cos(5*pi/2) →
		tan		tan(3*pi/4) →
pi				
e				

Arrondis

floor		floor(2.7) →	
ceil		ceil(2.4) →	
round		round(2.7) →	round(2.5) →

4.B) Le module `random`

Ce module permet de générer des nombres de façon aléatoire.

random		random() →
randint		randint(2,7) →

5. Exercices de conclusion

5.A) Python comme simple calculatrice.

Exercice 1. En 3000 av. J.C., le roi Belkib promet une récompense fabuleuse à qui lui proposerait une distraction qui le satisferait. Lorsque le sage Sissa lui présenta le jeu d'échecs, le souverain lui demanda ce qu'il souhaitait en échange de ce cadeau extraordinaire.

Sissa demanda au prince de déposer un grain de riz sur la première case, deux sur la deuxième, quatre sur la troisième, et ainsi de suite pour remplir l'échiquier en doublant la quantité de grain à chaque case. Le prince accorda immédiatement cette récompense sans se douter de ce qui allait suivre.

- Combien y a-t-il de grains de riz sur la dernière case de l'échiquier ?
- Et sur tout l'échiquier ? (On va voir si vous vous souvenez de la formule donnant la valeur de $1 + 2 + 2^2 + \dots + 2^{63}$)
- Supposons que le poids moyen d'un grain de riz est de 0,04 g. Quel est le poids total du riz sur l'échiquier ?
- La production mondiale de riz est estimée par le FAO à 699 millions de tonnes en 2010. Combien d'années faudrait-il, au rythme de production de 2010, pour donner sa récompense à Sissa ?

Exercice 2. Quel jour de la semaine serons-nous dans 4560 jours ? Combien de dimanches seront passés ?
On pourra utiliser les commandes % et //.

Exercice 3.

Charger le module `math` et calculer les expressions suivantes. Quels sont les résultats obtenus ? Proposez une explication s'ils paraissent étonnants.

- `sin(pi)`
- `cos(pi/2)`
- `log(e)`
- `sqrt(2)`
- `exp(0)`
- `sqrt(-3)`
- `int(-3.3)`
- `floor(-3.3)`

Exercice 4.

Étant donné le nombre impressionnant de commandes disponibles dans *Python*, il est inconcevable de les connaître toutes par cœur ! Il est donc utile de savoir accéder à la documentation relative à une fonction prédéfinie de *Python*.

Cette documentation peut être affichée dans la console en tapant l'instruction :

`help(nom_fonction)`

Utiliser l'aide pour savoir comment utiliser les fonctions `floor`, `ceil` et `round`. Calculer `floor(-3.5)`, `ceil(-3.5)`, `round(3.5)`, `round(3.557, 1)`, `round(3516.2, -2)`. Expliquer les résultats obtenus.

Exercice 5.

Vous connaissez tous le logarithme népérien $\ln$. Il y a d'autres logarithmes, définis par ce qu'on appelle leur *base* de la manière suivante : Si b est réel strictement positif et x aussi, $\log_b(x) = \frac{\ln(x)}{\ln(b)}$.

Utiliser l'aide pour savoir comment utiliser la fonction `log`.

Calculer $x = \log_2(16)$, $y = \log_4(16)$ et $z = \log_{10}(100)$ puis 2^x , 4^y et 10^z : que remarquez vous ?
Comment calculer $\ln(10)$? A quelle base correspond le logarithme népérien ?

Exercice 6. Chargez le module `random`. On rappelle que l'instruction `random()` de ce module renvoie un nombre flottant aléatoire de l'intervalle $[0; 1[$.

1. À l'aide de la fonction `random`, écrire une commande renvoie une valeur aléatoire de l'intervalle $[0; 70[$.
2. En déduire une commande qui renvoie une valeur aléatoire de l'intervalle $[-20; 50[$.
3. Sur le même modèle, écrire une commande qui renvoie une valeur aléatoire de l'intervalle $[3; 8.5[$.

5.B) Variables

Exercice 7.

Pour chacun des scripts suivants, deviner sans utiliser l'ordinateur, ce que renvoie le script à l'exécution. Vérifier vos prédictions en recopiant ces scripts et en les exécutant.

- `a = 1`
 `print(a)`
 `print(A)`
- `a=1`
 `b = 2`
 `c = a + b`
 `a = 8`
 `b = 4`
 `print(c)`
- `d = "bonjour"`
 `e = bonjour`

Exercice 8.

Au premier janvier 2015, deux populations, notées respectivement 1 et 2, comptaient respectivement 1000 et 700 individus. On considère que le taux d'accroissement de chaque population reste constant dans le temps et vaut 5% par an pour la population 1 et 7% par an pour la population 2.

1. Taper les instructions suivantes dans la console :
 `eff1 = 1000`
 `eff1 = eff1 * 1.05`
 Que venez-vous de calculer ?
2. La population 2 sera-t-elle plus nombreuse que la population 1 avant le premier janvier 2022 ? On pourra utiliser la flèche  du clavier afin de ne pas avoir à retaper à chaque fois tous les calculs.

Exercice 9. Écrire un script qui modélise le lancer de trois dés équilibrés et qui affiche leur somme.

Exercice 10. Pour $a, b \in \mathbb{R}$, on définit :

- Leur moyenne arithmétique, comme étant la moitié de leur somme.
 - Si $a, b \geq 0$, leur moyenne géométrique, comme étant la racine carrée de leur produit.
1. Écrire un script qui affiche la moyenne arithmétique et la moyenne géométrique de deux nombres réels choisis aléatoirement dans l'intervalle $[0; 1000[$.
 2. Exécutez votre script un bon nombre de fois : quelle inégalité pouvez-vous conjecturer ?