TP 1. Prise en main du langage Python

Cette année nous travaillerons avec le langage **Python** version 3.2 ; nous utiliserons l'environnement de développement **IDLE**.

Étape 0. Dans votre espace personnel, créer un dossier *infoPcsi*. Dans ce dossier créer un sous-dossier *tp1* : c'est là que vous enregistrerez les fichiers relatifs à ce TP.

1 Python en interactif

Lorsque vous ouvrez l'environnment IDLE vous obtenez une fenêtre appelée *interpréteur* (ou *console*, ou *terminal*, ou *shell*). À partir de là nous pouvons utiliser le langage Python en mode interactif.
Les expressions sont rédigées à la suite des chevrons >>> puis évaluées avec la commande *Entrée*.

```
Python 3.2.5 (default, May 15 2013, 23 :06 :03) [MSC
v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more infor-
mation.
>>>
```

Exercice 1. Calculer avec Python

1. Ouvrir IDLE puis réaliser les opérations suivantes dans l'interpréteur.

$$50 + 3 \times (12,5 - 31) = \ldots\ldots\ldots\ldots \qquad \frac{3}{2} = \ldots\ldots\ldots\ldots \qquad \frac{2}{3} = \ldots\ldots\ldots\ldots \qquad 2^5 = \ldots\ldots\ldots\ldots$$

2. Évaluer les expressions suivantes : `23//3` et `23 % 3` ; `abs(1.3)` et `abs(-1.3)`.

Compléter le tableau ci-dessous :

Opération	**Opérateur**	**Opération**	**Opérateur**
Somme		Produit	
Différence		Division numérique	
	`//`		`%`
Puissance		Valeur absolue	

Les fonctions mathématiques usuelles ne sont pas disponibles immédiatement à l'ouverture d'IDLE ; il faut charger une librairie (ou module) complémentaire : le module `math`. Nous verrons d'autres modules importants au cours de l'année.

Exercice 2. Exécuter les commandes ci-dessous.

```
>>> sin(pi)
>>> from math import *
>>> sin(pi)
```

La commande `import` donne accès à une nouvelle librairie ; le symbole * signifie que toutes les nouvelles commandes issues du module importé sont désormais accessibles.

Exercice 3. Calculer (des valeurs approchées) des expressions suivantes.

$$\ln(10) = \ldots\ldots\ldots \qquad \cos\left(\frac{\pi}{8}\right) = \ldots\ldots\ldots \qquad \frac{\sqrt{2+\sqrt{2}}}{2} = \ldots\ldots\ldots$$

2 Variables et affectations

Pour manipuler des données, il peut être utile de manipuler des *variables* qui représentent ces données.

Les objets manipulés par Python, au moment où ils sont créés, sont stockés dans la mémoire de l'ordinateur. Un *nom de variable* est une suite de caractères qui renvoie à une adresse mémoire où a été créé un objet.

Pour affecter (ou assigner) une valeur à une variable on utilise le signe *égal*.

La commande ci-contre signifie que désormais le caractère `x` renvoie à la valeur `1`.
Le signe *égal* doit être distingé de l'égalité mathématique.
On peut écrire en *pseudo-code* : $x \longleftarrow 1$

```
>>> x=1
>>> x
1
```

Exercice 4. Examiner la série de commandes ci-dessous. Prédire le résultat puis confirmer-le à l'aide d'IDLE.

```
>>> x=1
>>> y=2
>>> x=x+y
>>> y=x**y
>>> y
?
```

Règle. Les noms de variables autorisés sont des séquences de lettres (majuscules ou miniscules) et de chiffres qui débute toujours par une lettre. La *casse* est significative. Il existe quelques mots *réservés* ne pouvant faire office de nom de variable (`def`, `if`, `while`, `True`, `False`...).

La fonction `print()` permet également d'afficher la valeur d'une ou plusieurs variables séparées par des virgules.

```
>>> x=1
>>> y=1.5
>>> print(x,y)
1  1.5
```

Affectations multiples. Le langage Pyhton offre la possibilité de réaliser en une seule instruction plusieurs affectations.

Affectations successives. La commande ci-contre signifie que `x` représente la valeur `1` et `y` représente la valeur `x` (donc `1`).

```
>>> y=x=1
>>> y
1
>>> x
1
```

Affectations parallèle. La commande ci-contre signifie que `a` représente la valeur `1` et `b` représente la valeur `2`.

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

Remarque. La commande ci-dessous permet d'échanger deux noms de variables.

```
>>> x,y=y,x
```

Exercice 5. Examiner la série de commandes ci-dessous. Prédire le résultat puis confirmer-le à l'aide d'IDLE.

```
>>> var1,var2=1,2
>>> var1,var2=var2,var1+var2
>>> var3=var1**var2
>>> print(var1,var2,var3)
?
```

3 Premier programme Python

L'utilisation de Python en ligne de commande dans l'interpréteur ne permet pas de sauvegarder vos lignes de calcul : ce n'est pas très pratique si vous souhaitez relancer une même série de calculs avec des valeurs différentes pour les variables.

Pour conserver nos programmes on va donc sauvegarder les lignes de code dans un fichier texte : on parlera d'un *script* Python. À partir d'une console IDLE la procédure est la suivante :

1. Ouvrir une nouvelle fenêtre : File → New Window.
2. Écrire et enregistrer le script. Le nom du fichier doit se terminer par l'extension **.py**.
3. Exécuter le script : Run → Run Module ou F5. Le résultat est donné dans l'interpréteur.

Exercice 6. On souhaite rédiger un programme élémentaire qui calcule le prix d'une commande de livres. Les trois valeurs suivantes sont représentées par les variables :

`nbr`	=	entier désignant le nombre de livres commandés
`prix`	=	prix unitaire d'un livre
`reduc`	=	coefficient (entre 0 et 1) représentant la réduction dont bénéficie le client

Le programme affiche le montant de la facture `m = nbr * prix * reduc`.

1. Ouvrir un nouveau fichier que vous sauvegardez sous le nom de `factureLivres.py`.
2. Affecter les variables comme indiqué ci-dessus dans le cas d'une commande de 27 livres dont le prix unitaire est de 30,5 € pour un client bénéficiant de 5 % de réduction.
3. Afficher le résultat dans l'interpréteur. Pour afficher un résultat il faut faire appel à la fonction `print`. Pour afficher une phrase on peut utiliser par exemple la syntaxe : `print('Le montant de la commande est de',m,'euros')`. Pour aller plus loin : vous afficherez la valeur tronquée de `m` à deux chiffres après la virgule.

Remarques.

1. Vous pouvez afficher des commentaires dans votre script en les rédigeant précédés d'un symbole # ; ils ne seront pas exécutés.
2. La fonction `input()` permet d'interagir avec l'utilisateur : elle interrompt le programme jusqu'à ce que l'utilisateur rentre une valeur et appuie sur *Entrée*. Cela nous impose d'en apprendre un peu plus sur les *types* des variables...

4 Premiers types simples

Dans les parties précédentes nous avons manipulé des objets de nature numérique (entiers, décimaux). Pour désigner la nature d'un objet manipulé par Python, on parle de **type**. Voici les types les plus simples :

1. `int` : les entiers relatifs. *Cf chapitre 2.*

2. `float` : les nombres en virgule flottante. *Cf chapitre 2.*

3. `str` : les chaines de caractères.

   Une *chaine de caractères* est une séquence de caractères les uns à la suite des autres ; pour les définir il faut les écrire entre *quotes* (simples ou doubles). Les éléments d'une chaine sont numérotés à partir de 0. Pour extraire un caractère, il suffit d'accoler à la chaine (ou la variable qui la représente) son indice entre crochet.

   ```
   >>> message='bonjour'
   >>> print(message)
   bonjour
   >>> message[1]
   'o'
   ```

   Si cela a un sens, les commandes `int` et `float` convertissent une chaine de caractères en une donnée numérique.

Voici une liste non exhaustive d'opérations autorisées sur les chaines de caractères.

Opération	**Opérateur**	**Opération**	**Opérateur**
Concaténation	+	Répétition	*
Extraction de fragment	*Chaine*`[n:m]`	Longueur	`len`(*Chaine*)

4. `bool` : les booléens. *Cf TP 2. Structures conditionnelles.* Il s'agit d'expressions dont la valeur est `True` ou `False`.

```
>>> 1>0
True
>>> 1==0
False
```

Les booléens peuvent être construits avec les opérateurs suivants :

Opération	**Opérateur**	**Opération**	**Opérateur**
Égal	==	Différent	!=
Strictement supérieur	>	Strictement inférieur	<
Supérieur ou égal	>=	Inférieur ou égal	<=
ou	`or`	et	`and`
non	`not`		

La commande `bool()` évalue l'argument comme un booléen selon la règle suivante : si la donnée est booléenne elle retourne sa valeur, si la donnée est 1 elle retourne `True` sinon elle retourne `False`

Exercice 7. Prédire la valeur des booléens suivants et confirmer votre résultat avec l'interpréteur Python.
A = `not(1>0 and 0>1)` ..... B = `1!=0 or 1==1` .....

5. `tuples` : les *tuples*. Il s'agit de séquences d'objet séparés par des virgules. Pour extraire une opérande, il suffit d'accoler à la chaine (ou la variable qui la représente) son indice entre crochet (on commence à 0).

```
>>> x='coucou',1,1>0
>>> x[2]
True
```

La commande `type` permet d'obtenir le type d'un objet.

Sous Python, il n'est pas nécessaire de définir le type d'une variable avant de l'utiliser : on dit que le langage est à *typage dynamique*.

Exercice 8. Reprendre l'exercice 6 en utilisant la fonction `input` pour interroger l'utilisateur et lui retourner le montant de la facture.

Indication. La commande `input` récupère les entrées de l'utilisateur sous forme de *chaine de caractère*.

```
>>> X=input('Quel âge avez-vous ? ')
Quel âge avez-vous ? 17
>>> X,type(X)
('17', <class 'str'>)
```

Exercice 9. Qu'est-ce q'un bon mot de passe ?

Vu sur le site securite-informatique.gouv.fr :

« Un mot de passe est bon quand il exploite au maximum les possibilités de choix laissées par le mécanisme de déverrouillage pour qu'il soit plus difficile à retrouver, soit directement, soit par « ingénierie sociale », soit à l'aide d'outils automatisés. Un mot de passe est d'autant plus faible qu'il est court ou qu'il est composé à partir d'un alphabet réduit.

Sur un simple PC un outil de « craquage par force brute »mettrait environ 1 heure pour craquer un mot de passe de 8 caractères alphabétiques [A-Z]. Mais il faudrait environ 1 mois à ce même PC pour craquer un mot de passe de 10 caractères alphabétiques ou un mot de passe de 8 caractères alphanumériques [A-Za-z0-9].

Avec les capacités techniques actuelles, la taille d'un mot de passe doit être d'au moins 10 caractères non signifiants, composés de lettres majuscules, minuscules, de chiffres et si possible de caractères spéciaux. »

En Python la fonction `randrange()` permet de générer des entiers de façon (pseudo)-aléatoire. Elle doit être importée du module `random`.

Réaliser un programme `mdp.py` dont l'exécution affiche **aléatoirement** un mot de 10 caractères *composés de lettres majuscules, minuscules, de chiffres et si possible de caractères spéciaux* .

Exercice 10. Réaliser un programme `horaire.py` qui demande à l'utilisateur : l'horaire de départ de son train, l'horaire d'arrivée de son train et qui affiche la durée du trajet en minutes.

On suppose que l'utilisateur rentre les horaires sous le format : `XXhYY` (exemples : 19h51, 08h01).

Tester votre programme pour les entrées : départ 17h51 ; arrivée 19h38. Dans l'interpréteur votre résultat ressemble à ça :

```
>>>
Horaire de départ ? 17h51
Horaire d'arrivée ? 19h38
Votre trajet dure 107 minutes.
```

Quels sont les inconvénients de votre programme ?