TP 1
Prise en main

I Le mode interactif

Dans le mode interactif, on écrit les commandes **une ligne à la fois**. À chaque ligne, Python affiche le résultat du calcul, et enregistre au fur et à mesure les variables.
Les morceaux de programme de ce document sont donc à recopier et à essayer ligne par ligne. **Ne pas hésiter à prendre des initiatives** et tester avec d'autres valeurs que celles proposées ici !
Les espaces sont optionnels, mais rendent le code plus lisible. La différence entre minuscules et majuscule est importante.

I.1 Utilisation comme calculatrice

Dans un premier temps, on peut utiliser Python comme une simple calculatrice avec les nombres entiers et les opérations `+`, `-`, `*`, `/`, et les parenthèses.

```
3 + 4
8 - 10
1 + 3 * 5
(1 + 3) * 5
5 + 1 / 2
```

L'opération puissance s'écrit `**` : ainsi 10^3 est

```
10 ** 3
```

I.2 Les types

Les expressions Python ont un **type**, qui indique quel type d'objet on manipule et comment l'ordinateur se les représente. Le type d'une expression peut-être obtenu avec la fonction `type` :

```
type(3)
type(3.5)
```

Les principaux types à connaitre sont les suivants :

Le type entier `int` C'est le type des nombres entiers comme `5` ou bien `-2`.
Un fait remarquable est que Python est capable de gérer des nombres entiers très grands !

```
2 ** 10
2 ** 100
2 ** 1000
```

Le type flottant `float` C'est le type pour les nombres « à virgule » (voir ci-dessous). En informatique, on ne peut pas représenter tous les nombres réels avec une infinité de décimales après la virgule, car l'espace pour stocker le nombre n'est pas infini...
Ces nombres peuvent s'écrire avec un point `.` pour la virgule, avec un signe, mais aussi en notation scientifique où la lettre `e` désigne la puissance de 10 :

```
3.57e+4
2e-5
12.35e+50
```

Remarque 1. On parle de *nombre à virgule flottante* car dans son fonctionnement interne l'ordinateur les représente en notation scientifique : un espace fixe pour stocker l'exposant, et un espace fixe pour stocker les décimales. Ainsi la précision d'un nombre est toujours limitée à une quinzaine de chiffres, mais l'exposant peut varier d'au moins `e-308` à `e+308`. C'est la plupart du temps bien suffisant pour les sciences !

L'opération de division `/` donne toujours un nombre flottant, même entre nombres entiers :

```
10 / 5
type(10 / 5)
```

Pour travailler uniquement avec des nombres entiers on utilise la division euclidienne `//`, celle qui donne un quotient et un reste :

```
10 // 5
11 // 5
17 // 5
```

Le reste lui est obtenu par l'opération `%`

```
15 % 5
16 % 5
17 % 5
```

Le type des chaines de caractères `str` En informatique, un texte à manipuler s'appelle une *chaine de caractères*, en anglais *string*. En effet l'ordinateur les considère comme une liste de caractères les uns à la suite des autres (que le texte ait du sens ou pas...). Ce sont des types d'objets que l'on peut très bien utiliser en Python, elles sont encadrées par le guillemet double `"`.

```
"Bonjour"
type("Bonjour")
```

L'opération `+` sur les chaines de caractère crée une nouvelle chaine en plaçant les deux bout à bout et s'appelle en informatique la **concaténation**.

```
"machin" + "truc"
```

Il existe aussi une opération de multiplication entre une chaine et un entier. À votre avis, que fait-elle ?

```
"ha" * 5
```

La syntaxe crochets `[n]` permet d'accéder au n-ième caractère da la chaine :

```
"Bonjour"[1]
"Bonjour"[2]
```

... en fait, elle est numérotée à partir de 0. Et avec des indices négatifs, on compte en partant de la fin !

```
"Bonjour"[0]
"Bonjour"[-1]
"Bonjour"[-2]
```

Le type booléen `bool` C'est un type qui représente une valeur vraie ou faux, en Python `True`, `False`. Le nom provient du mathématicien anglais George Boole.

```
type(True)
type(False)
```

On peut utiliser dessus les opérations
- et : `and`
- ou : `or`
- non : `not`

... et vérifier toutes les propriétés que nous avons vu en cours !

```
True and False
True and True
True or True
not True
```

Bien entendu cela se combine avec des parenthèses.
Les opérations de comparaisons entre nombres donnent une valeur booléenne. Ce sont :
- L'égalité `==`, la différence `!=`,
- Les comparaisons strictes `<`, `>`,

— Les comparaisons larges $\leqslant$, $\geqslant$ s'écrivent en Python `<=`, `>=`.

```
5 > 8
5 + 3 <= 8
1 == 2
```

Les opérations logiques ne sont pas tout à fait commutatives comme en mathématiques...

```
1/0==5 or True
True or 1/0==5
```

I.3 Les conversions de type

Comparer et expliquer :

```
12 + 34
"12" + "34"
```

Comme on l'a vu, le nombre `3` n'est pas la même chose que `3.0` car le premier est `int` et le second est `float`. Et la chaine de caractères `"12"` n'est pas la même chose que le nombre `12`.
Aux types correspondent aussi des fonctions de conversion de type `int()`, `float()`, `str()`, `bool()` :

```
float(5)
int(12.7)
int(-12.7)
int("12") + int("34")
str(12) + str(34)
```

Via `bool()`, n'importe quel nombre est converti à `True`, sauf `0`. N'importe quelle chaine est convertie en `True`... sauf la chaine vide `""`. Essayer !

I.4 Les variables

En informatique, une variable est une case de la mémoire capable d'enregistrer une valeur ou le résultat d'un calcul. Pour la manipuler il faut lui donner un nom (qui peut être composé de plusieurs lettres), elle a alors un type et un contenu.

```
x = 3
x
type(x)
```

L'opération `x = 3` s'appelle **affectation**, elle dit de mettre la valeur `3` dans le nombre `x`, ce qu'on note parfois en mathématiques $x \leftarrow 3$. Ce n'est pas vraiment la même chose que de tester l'égalité `x == 3`.
Les types découverts ci-dessus peuvent tous être contenus dans une variable ! Avec des chaines :

```
nom = "Louis"
"Bonjour  " + nom
```

Ou avec des booléens :

```
t = True
not t
```

En mode interactif, le programme enregistre les variables au fur et à mesure jusqu'à ce qu'on lui demande de redémarer. Si vous avez écrit les codes ci-dessus alors la variable `x` est restée déclarée et égale à `3`.

II Le mode script

II.1 Principes de base

En mode script, on peut écrire plusieurs lignes à la suite et l'envoyer d'un coup à l'interpréteur Python. Les instructions sont obligatoirement séparées par un retour à la ligne. Le programme continue à enregistrer les variables au fur et à mesure, mais **il n'affiche rien si on ne le lui demande pas** ! Il faut alors utiliser la commande `print()` pour afficher quelque chose.

```
x = 5
y = 8
x = y + 8
print(x)
```

On peut aussi se servir de `print` pour afficher plusieurs variables, ou nombres ou chaines, à la suite sur une même ligne.

```
x = 5
y = 12
print("x vaut", x, "et y vaut", y)
```

Toute ligne commencée par le symbole `#` est ignorée : c'est un commentaire, qui sert à expliquer ce que fait le programme. Dans Pyzo, une ligne `##` permet de séparer le code en *cellules* pour garder tout le code sur une même page, mais ne demander à n'exécuter qu'une seule cellule à la fois (sinon, on rajoute du code au fur et à mesure, et à chaque fois, tout est exécuté depuis le début !) Une bonne idée est de l'utiliser pour séparer chaque question des exercices.
Enfin la dernière notion utile pour entamer le mode script est la fonction `input()` qui permet de demander une information à l'utilisateur.

```
x = input("Entrez quelque chose : ")
print("Vous avez entré", x)
```

La valeur donnée par `input()` est toujours de type `str` ! Si on veut demander un nombre (entier) à l'utilisateur, on écrit en général directement `int(input())` :

```
x = int(input("Entrez un nombre entier : "))
print("Son double est :", 2 * x)
```

II.2 Des exercices

Dans les exercices suivants, on demande d'écrire des petits bouts de programme : quelques lignes de code dans le mode interactif, séparées en cellules, de façon à pouvoir les exécuter séparément. Pour que les programmes soient intéressants, ils utilisent largement `input` pour que l'utilisateur puisse varier un paramètre.

Exercice 1. Écrire un programme qui demande à l'utilisateur son nom, puis affiche `"Bienvenue"` suivi de son nom.

Exercice 2. Écrire un programme qui demande à l'utilisateur son année de naissance, puis affiche son âge en 2022.

Exercice 3. Écrire un programme qui demande à l'utilisateur d'entrer deux nombres `a` et `b` et en fait la somme.

Exercice 4. Écrire un programme qui demande à l'utilisateur d'entrer deux nombres `a` et `b`, les affiche, puis les échange, et les affiche encore.

Exercice 5. On peut utiliser `print` directement sur une expression booléenne pour afficher simplement `True` or `False`.
Écrire des programes qui demandent à l'utilisateur des nombres et testent, en affichant le résultat, les conditions suivantes :

1. Le nombre `n` est pair.
2. Les nombres entiers `n` et `m` sont de même signe.
3. Les nombres entiers `n`, `m`, `p` sont de même signe.
4. Les nombres entiers `n`, `m`, `p` sont deux à deux distincts.