Travaux pratiques

Instructions itératives

Préambule : la fonction print

Lors de la première séance de travaux pratiques, nous avons réalisé de la programmation par *effet* : les différentes commandes utilisées (`forward`, `left`, ...) pour diriger la tortue *modifiaient* le contenu de la fenêtre graphique, autrement dit avaient un effet sur l'environnement.
La fonction `print` réalise elle aussi un effet, mais cette fois-ci directement dans la fenêtre du shell, en affichant les uns à la suite des autres ses différents arguments. Par exemple :

```
In [1]: print(1, 2, 3)
1 2 3

In [2]:
```

On peut observer que chacun des arguments a été affiché séparé des autres par un espace. En outre, même si cela est moins visible, un passage à la ligne a été ajouté à la fin de l'affichage. Ces arguments se nomment `sep` (comme *separator*) et `end` ; ce sont des variables qui contiennent des chaînes de caractères et qui possèdent les valeurs par défaut : `sep=' '` et `end='\n'` (le caractère d'échappement \ sert à coder au sein d'une chaîne de caractères des valeurs particulières, ici `\n` pour un passage à la ligne).
Il est possible de modifier les valeurs de ces deux arguments ; par exemple pour attribuer au paramètre `sep` la valeur `'\n'` et au paramètre `end` la valeur `'*'` il suffit d'écrire :

```
In [2]: print(1, 2, 3, sep='\n', end='*')
1
2
3*
In [3]:
```

On observe que cette fois les trois arguments sont séparés par un passage à la ligne, et qu'à la fin de l'affichage une étoile a été ajoutée.

Énumérations

Nous avons déjà rencontré la notion d'énumération lors de la séance précédente : on produit une énumération en utilisant la fonction `for` dans la structure suivante :

```
for ... in ...:
    bloc ..........................
    ..............................
    d'instructions ................
```

Immédiatement après le mot-clé `for` doit figurer le nom d'une variable, qui va prendre les différentes valeurs de l'énumération qui figure après le mot-clé `in`. Pour chacune de ces valeurs, le bloc d'instructions qui suit sera exécuté.
La fonction `range` permet d'énumérer des entiers suivant une progression arithmétique ; elle peut prendre entre 1 et 3 arguments :

- `range(b)` énumère les entiers $0, 1, 2, \ldots, b-1$;
- `range(a, b)` énumère les entiers $a, a+1, a+2, \ldots, b-1$;
- `range(a, b, c)` énumère les entiers $a, a+c, a+2c \ldots, a+nc$ où n est le plus grand entier vérifiant $a+nc < b$.

Exercice 1. Premières énumérations

a. Définir une fonction baptisée `entiers` prenant comme arguments deux entiers i et j (avec $i \leqslant j$) et qui affiche sur une même ligne les entiers de l'intervalle $[i, j]$ séparés par le caractère –.

b. Modifier votre fonction pour qu'elle affiche désormais les entiers de l'intervalle $[\![i, j]\!]$ qui ne sont pas des multiples de 7.

```
In [1]: entiers(7, 21)
7–8–9–10–11–12–13–14–15–16–17–18–19–20–21

In [2]: entiers(7, 21)
8–9–10–11–12–13–15–16–17–18–19–20
```

Figure 1 – Un exemple d'utilisation des deux versions successives de la fonction `entiers`.

Exercice 2. Graphisme en console

Rappelons que les chaînes de caractères Python sont délimitées par le caractère `'` ou `"` et possèdent deux opérateurs :

– l'opérateur de concaténation `+` :

```
In [1]: 'abc' + 'def'

Out[1]: 'abcdef'
```

– l'opérateur de duplication `*` :

```
In [2]: 'abc' * 3

Out[2]: 'abcabcabc'
```

a. Définir une fonction `triangle1` à un argument entier n qui dessine dans le shell un triangle sur n lignes.

b. Définir une fonction `triangle2` qui dessine ce même triangle mais dans l'autre sens.

c. Définir une fonction `pyramide1` qui dessine une pyramide sur $2n-1$ lignes.

d. Définir une fonction `pyramide2` qui dessine une pyramide sur n lignes.

Un exemple pour $n = 5$ de chacune de ces quatre fonctions est présenté figure 2.

```
In [1]: triangle1(5)
*
**
***
****
*****
```

```
In [2]: triangle2(5)
*****
****
***
**
*
```

```
In [3]: pyramide1(5)
*
**
***
****
*****
****
***
**
*
```

```
In [4]: pyramide2(5)
    *
   * *
  * * *
 * * * *
* * * * *
```

Figure 2 – Les quatre fonctions demandées pour $n = 5$.

Formatage d'une chaîne de caractères

La méthode `format` appliquée à une chaîne de caractères permet d'insérer une ou plusieurs valeurs numériques en son sein. Dans la chaîne initiale, les emplacements que vont occuper les valeurs sont représentées par `{}` ; par exemple,

l'instruction : `'{} plus {} égal {}'.format(1, 2, 3)` produit la chaîne de caractères : `'1 plus 2 égal 3'`.

Pour modifier l'ordre de substitution ou utiliser à plusieurs reprises la même valeur on peut numéroter les accolades en partant de 0. Par exemple,

`'{0:} plus {0:} égal {1:}'.format(1, 2)` produit la chaîne : `'1 plus 1 égal 2'`.

Les valeurs numériques sont transformées en chaînes de caractères avant d'être insérées au sein de la chaîne cible, mais l'intérêt de cette notion réside surtout dans le fait qu'il est possible d'*ajouter une option de formatage lors de cette conversion*. Par exemple,

`{:.4f}` force l'écriture en virgule flottante en fixant le nombre de décimales (ici 4) ;

`{:.3e}` force l'écriture en notation scientifique en fixant le nombre de décimales (ici 3).

Ainsi, `'{0:.6e} est approximativement égal à {0:.2f}'.format(100/7)` produit la chaîne :

`'1.428571e+01 est approximativement égal à 14.29'`.

Il existe de nombreuses options de formatage ; l'exercice qui suit peut être résolu élégamment en utilisant les options de formatage suivantes :

`{:<10}` fixe la longueur de la chaîne (ici 10) et justifie à gauche ;

`{:>10}` fixe la longueur de la chaîne (ici 10) et justifie à droite ;

`{:^10}` fixe la longueur de la chaîne (ici 10) et justifie au centre.

Exercice 3. Le Talkhys

Le *Talkhys* est un traité d'arithmétique d'Ibn Albanna, mathématicien marocain de la première moitié du XIII^{e} siècle. On y trouve un certain nombre d'identités remarquables telles celles présentées figure 3.

```
       1 x 1         =         1
      11 x 11        =       121
     111 x 111       =     12321
    1111 x 1111      =    1234321
   11111 x 11111     =   123454321
  111111 x 111111    =  12345654321
 1111111 x 1111111   = 1234567654321
11111111 x 11111111  = 123456787654321
111111111 x 111111111 = 12345678987654321
```

```
8 x 1          + 1 = 9
8 x 12         + 2 = 98
8 x 123        + 3 = 987
8 x 1234       + 4 = 9876
8 x 12345      + 5 = 98765
8 x 123456     + 6 = 987654
8 x 1234567    + 7 = 9876543
8 x 12345678   + 8 = 98765432
8 x 123456789  + 9 = 987654321
```

```
9 x 1          +  2 = 11
9 x 12         +  3 = 111
9 x 123        +  4 = 1111
9 x 1234       +  5 = 11111
9 x 12345      +  6 = 111111
9 x 123456     +  7 = 1111111
9 x 1234567    +  8 = 11111111
9 x 12345678   +  9 = 111111111
9 x 123456789  + 10 = 1111111111
```

```
9 x 9          + 7 = 88
9 x 98         + 6 = 888
9 x 987        + 5 = 8888
9 x 9876       + 4 = 88888
9 x 98765      + 3 = 888888
9 x 987654     + 2 = 8888888
9 x 9876543    + 1 = 88888888
9 x 98765432   + 0 = 888888888
```

Figure 3 – Des identités remarquables issues du Talkhys.

Choisir une ou deux tables de votre choix parmi les quatre présentées et les reproduire dans la console.

Retour d'une fonction

Les quelques fonctions que nous avons écrites jusqu'à présent réalisent un effet (dans la fenêtre graphique du module Turtle ou dans la console par l'intermédiaire de la fonction `print`) mais ne retournent aucun résultat [1]. Pour qu'une fonction retourne un résultat, il faut utiliser l'instruction `return` suivie du contenu que l'on souhaite voir retourner par la fonction. Par exemple, à la fonction $f : \mathbb{N} \to \mathbb{N}$ définie ci-dessous correspondra la fonction Python :

$$\forall n \in \mathbb{N},\ f(n) = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 3n+1 & \text{si } n \text{ est impair} \end{cases}$$

```
def f(n):
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1
```

L'intérêt majeur de cette notion est de permettre la composition des fonctions, puisque un résultat retourné par une fonction peut servir d'argument à une autre fonction, ce que ne permet pas la programmation par effet. Ainsi, pour calculer $f \circ f \circ f(17)$ il suffira d'écrire :

```
In [1]: f(f(f(17)))
Out[1]: 13
```

1. En réalité si, ces fonctions retournent une valeur particulière appelée None qui reste en général masquée.

Exercice 4. calcul de durée
Rédiger une fonction `heure_to_sec(h, m, s)` qui prend en arguments trois entiers représentant une durée exprimée en heures/minutes/secondes et qui *retourne* cette durée exprimée en secondes.
Rédiger une fonction `sec_to_heure(s)` qui prend en argument un nombre entier de secondes et qui *affiche* cette durée au format `hh:mm:ss`.
En déduire une fonction `duree(h1, m1, s1, h2, m2, s2)` qui prend en arguments six entiers représentant deux dates d_1 et d_2 et qui affiche la durée de $d_2 - d_1$ au format `hh:mm:ss` (on supposera $d_1 < d_2$).

Indexation d'une chaîne de caractères

Python offre une méthode simple pour accéder aux caractères contenus dans une chaîne : chaque caractère est accessible directement par son rang dans la chaîne (numéroté à partir de 0), encadré par des crochets. Par exemple, si `s = 'abcdefghi'` alors `s[0] = 'a'`, `s[1] = 'b'`, ..., `s[8] = 'i'`. À l'inverse, la méthode `index` appliquée à une chaîne de caractères `s` retourne le rang de la première apparition de ce caractère dans `s`. Par exemple, `s.index('c') = 2`, `s.index('g') = 6`, mais `s.index('k')` déclenche l'exception `ValueError` puisque le caractère `'k'` n'est pas présent dans `s`.

Exercice 5. ROT13
Le ROT13 (*rotate by 13 places*) est un cas particulier du chiffre de César ; comme son nom l'indique, il s'agit d'un décalage de 13 caractères de chaque lettre du texte à chiffrer : A devient N, B devient O, C devient P, etc. Son principal aspect pratique est que le codage et le décodage se réalisent exactement de la même manière, puisque notre alphabet contient 26 lettres.
ROT13 est parfois utilisé dans les forums en ligne comme un moyen de masquer la réponse à une énigme, un *spoiler* ou encore une expression grossière, et on trouve sur le net de nombreux sites se proposant de coder/décoder une phrase en suivant ce principe.

Définir une fonction nommée `rot13` qui prend en argument une chaîne de caractères et retourne cette même chaîne codée en ROT13. On conviendra que :

- les caractères a, b, c, ..., y, z seront transformés en n, o, p, ..., l, m ;
- tous les autres caractères (espaces, ponctuation, chiffres, caractères accentués, ...) ne seront pas modifiés.

Indication. Définir une chaîne de caractères `alph = 'abcdefghijklmnopqrstuvwxyz'` et utiliser le fait que si `c` est un caractère et `s` une chaîne de caractères, l'instruction `c in s` renvoie un booléen caractérisant la présence ou non de `c` dans `s`.

Utilisez votre fonction pour connaître la réponse à l'énigme suivante :

> Quelle est la différence entre un informaticien et une personne normale ?
> *har crefbaar abeznyr crafr dh'ha xvyb-bpgrg rfg étny à 1000 bpgrgf, ha vasbezngvpvra rfg pbainvaph dh'ha xvybzèger rfg étny à 1024 zègerf.*

Boucles conditionnelles

Une boucle conditionnelle applique une suite d'instructions tant qu'une certaine condition est réalisée ; elle peut donc tout aussi bien ne jamais réaliser cette suite d'instructions (lorsque la condition n'est pas réalisée au départ) que de les réaliser un nombre infini de fois (lorsque la condition reste éternellement vérifiée). La syntaxe d'une boucle conditionnelle est la suivante :

```
while condition:
    bloc ..........................
    ..............................
    d'instructions ................
```

La condition doit être une expression à valeurs booléennes.

Exercice 6. racine carrée entière
La racine carrée entière d'un entier $n \in \mathbb{N}$ est l'unique entier p vérifiant $p^2 \leqslant n < (p+1)^2$.

a. Rédiger une fonction baptisée `isqrt` qui calcule la racine carrée entière d'un entier passé en paramètre.
b. Écrire une deuxième version de cette fonction en ne s'autorisant cette fois que des additions.

Exercice 7. Nombres-univers

On appelle *nombre-univers* (en base 10) un nombre réel dont la partie décimale contient n'importe quelle succession de chiffres de longueur finie. Un exemple simple de nombre-univers en base 10 est la constante de Champernowne $0,123\,456\,789\,101\,112\,131\,415\,161\,718\cdots$ On pense que π est un nombre univers mais ceci n'a pas encore été démontré.
De même, on appelle *suite-univers* (en base 10) une suite de nombres entiers telle que n'importe quelle succession de chiffres de longueur finie se trouve dans l'un des termes de cette suite. Il a été prouvé que la suite des puissances de 2 est une suite-univers (et donc que le nombre $0,124\,816\,326\,412\,825\,651\,210\,242\,048\cdots$ est un nombre univers).
Déterminer quelle est la plus petite puissance de 2 qui contient votre date de naissance (écrite au format *jjmmaa*).

Exercice 8. Nombres premiers

Le but de cet exercice est de déterminer les mille plus petits nombres premiers. Pour déterminer si un entier est premier, on utilise le critère suivant :

> un entier p est premier lorsque $p \geqslant 2$ et lorsqu'il n'est divisible par aucun entier $k \geqslant 2$ vérifiant $k^2 \leqslant p$.

a. Écrire une fonction nommée `premier`, prenant un paramètre entier p, et qui retourne le booléen True lorsque p est un nombre premier, et le booléen False dans le cas contraire.
b. Utiliser cette fonction pour afficher les mille plus petits nombres premiers.
c. La conjecture de Goldbach postule que tout entier pair supérieur à 3 peut s'écrire comme somme de deux nombres premiers (éventuellement égaux). Vérifier cette conjecture pour tout entier inférieur ou égal à 1000.
d. A contrario, montrer que la conjecture suivante est fausse : *Tout nombre impair est la somme d'une puissance de 2 et d'un nombre premier*.

Et pour les plus rapides...

Exercice 9. Suite de Conway

Les premiers termes de la suite de Conway sont : 1, 11, 21, 1211, 111221, ..., chaque terme étant obtenu en lisant à haute voix le terme précédent (c'est pourquoi Conway avait baptisé cette suite *Look and say*). Par exemple, le terme 1211 se lit « un 1, un 2, deux 1 » donc le terme suivant est 111221.
Rédiger en Python une fonction `lookandsay(n)` qui retourne le terme qui suit l'entier n dans cette énumération.

Remarque. Pour rédiger cette fonction, il est plus facile de manipuler les entiers sous la forme de chaînes de caractères. Par exemple, `lookandsay('1332')` retournera la chaîne `'112312'`.

Afficher à l'aide de cette fonction les 20 premiers termes de la suite de Conway.

Il a été démontré que si on note u_n le nombre de chiffres du n^{e} terme de cette suite alors le rapport $\dfrac{u_{n+1}}{u_n}$ tend vers une constante, appelée *constante de* Conway. Calculez-en une valeur approchée.

Une autre propriété remarquable de cette constante est qu'*elle ne dépend pas de la valeur initiale* (à l'exception de 22 qui produit une suite constante). Le vérifier expérimentalement en testant différentes valeurs initiales.

Démontrez enfin que dans la suite de Conway ne peuvent apparaître que les chiffres 1, 2 et 3.

Exercice 10. Compter avec les Shadoks

Les Shadoks possèdent pour tout vocabulaire quatre mots : « ga, bu, zo, meu », qui leur servent aussi à compter jusqu'à trois :

$$\text{ga} = 0, \quad \text{bu} = 1, \quad \text{zo} = 2, \quad \text{meu} = 3.$$

Le professeur Shadoko, conscient des limitations de ce système, inventa alors la façon de compter suivante : « quand on a meu shadoks, et qu'on en ajoute bu, il n'y a plus de place. On les mets alors dans une poubelle. Il y a donc bu poubelle et ga shadok à côté, ce qui donne bu-ga. » Ainsi,

$$\text{bu-ga} = 4, \quad \text{bu-bu} = 5, \quad \text{bu-zo} = 6, \quad \text{bu-meu} = 7, \quad \text{zo-ga} = 8, \quad \text{zo-bu} = 9, \quad \text{etc.}$$

Rédiger une fonction `shadok(n)` qui convertit un entier n en numérotation Shadok, puis écrire la fonction réciproque.