TP 3 : Structures répétitives et conditionnelles

Dans ce TP, on utilisera les structures conditionnelle et on commencera à utiliser les structures itératives.

I Un éditeur en ligne

Exercice 1 *Code en ligne*

Cette partie a juste pour but de vous donner une ressource pour vous entrainer à produire du coder Python via une interface en ligne sur Internet.

1. Ouvrir un navigateur Internet et aller sur la page `https://repl.it/languages/python3`. La partie droite de l'écran est l'éditeur de texte, la partie gauche la console. La case « Run » remplace la touche F5 de compilation, le reste étant tout à fait analogue. Les `>>>` de la console sont remplacés par  .
2. Dans cet éditeur, définir une fonction `g(x)` qui renvoie $\dfrac{1}{1+x^2}$ puis compiler et tester votre fonction.

Vous pourrez découvrir que vous pouvez créer un compte en ligne pour sauvegarder votre travail (via Facebook ou Gmail par exemple) et qu'il existe des raccourcis clavier `Ctrl+Enter` par exemple pour remplacer F5. Retenez que vous pouvez utiliser ceci pour vous entraîner à Python. Néanmoins, les fonctions de représentations peuvent ne pas être fonctionnelles. La suite du TP se passera comme d'habitude.

Exercice 2 *Pour rappel*

Démarrer le logiciel Spyder puis, dans le menu « Fichier » créer un nouveau fichier puis le sauvegarder avec un nom sous la forme TP3_votrenom. Dans la suite, on sauvegardera régulièrement son travail.

II Quelques exemples de structures conditionnelles

Dans chaque cas, on testera la fonction.

Exercice 3 *Tests de base*

1. Écrire une fonction Python `est_pair(n)` qui renvoie des chaînes de caractères : `l entier est pair` si n est pair et `l entier est impair` sinon. On pourra utiliser `//`.
2. Écrire une fonction Python `est_pair(n)` qui renvoie `l entier est pair est positif` si n est pair et positif et `l entier est impair` sinon.

Exercice 4 *Identifiant et mot de passe*

1. Créer une fonction `utilisateur(c)` qui renvoie la chaîne de caractère `'vous etes c'` où `c` est la chaîne de caractères de votre prénom (en minuscules et sans accents) lorsque `c` est effectivement votre prénom et `'je ne vous connait pas'` sinon.
2. Créer une fonction `utilisateur()` qui fait la même chose que précédemment mais en interagissant avec l'utilisateur avec la commande `input` (celle-ci a été vue au TP précédent et vous pourrez la revoir).
3. Créer une fonction `connexion()` qui interagit avec l'utilisateur en lui demandant son nom et son mot de passe et qui renverra `'connexion autorisee'` lorsque vous indiquez votre prénom puis un mot de passe (qui vous pouvez créer fictivement) et `'connexion refusee'` sinon.
4. *Question facultative :* Créer une fonction qui autoriserait plusieurs utilisateurs/mots de passe.

III Fonctions et structures conditionnelles

Exercice 5 *Une fonction*

1. Créer une Python `g(x)` qui est définie de la manière suivante : lorsque $x \in \mathbb{R}$,

$$g(x) = \sqrt{x} \text{ lorsque } x \geq 1 \text{ et } g(x) = -x + 2 \text{ lorsque } x < 1.$$

2. Représenter graphiquement la fonction sur $[-4, 4]$ (on pourra ouvrir les TP précédents et utiliser des modules comme `matplotlib.pyplot`).

Exercice 6 *Domaine de définition*

1. Recopier la fonction du cours `racines_reelles(a,b,c)` et la tester.
2. On considère la fonction $\varphi : x \mapsto ax^2 + bx + c$ avec $a, b, c \in \mathbb{R}$ et $a \neq 0$. Construire une fonction `signetrinome(a,b,c)` qui selon a, b, c donnés, affiche l'ensemble sur lequel le polynôme est positif. On pourra s'appuyer sur la fonction `racines_reelles(a,b,c)` et on doit obtenir des chaînes de caractères de la forme

```
'la fonction f est positive sur [-0.12645232,0.122336456]'
```

ou

```
'la fonction f est positive sur ]-infini, 0.2564556] U [0.122336456,+infini['
```

3. On souhaite désormais construire un programme interactif donnant le domaine de définition de la fonction $f : x \mapsto \sqrt{ax^2 + bx + c}$. Construire alors une fonction `domaine()` qui ne prend pas d'argument, mais qui :
   (a) demande à l'utilisateur les valeurs de a, b, c ;
   (b) renvoie le domaine de définition de f.

IV Structures itératives

Exercice 7 *Somme*

Pour représenter une suite, il suffira de prendre des entiers pour les abscisses (par exemple `range(1,4)` crée une liste avec les entiers de 1 à 4) et pour la représentation, on pourra utiliser `pypl.plot(...,...,'o')` pour avoir seulement des points (c'est le role du `'o'`).

1. Créer une fonction Python `liste_carres(n)` qui retourne la liste des carrées des n premiers entiers. Par exemple, si $n = 4$, la liste retournée est `[0,1,4,9,16]`.
2. En utilisant la fonction `sum` de Python (on pourra utiliser l'aide), écrire une fonction Python `somme_carres(n)` qui retourne

$$u_n = \sum_{k=0}^{n} k^2.$$

3. Modifier la fonction précédente pour qu'elle renvoie 0 si n n'est pas un entier ou si n est un entier strictement négatif.
4. Créer une fonction `repres(n)` qui représente graphiquement les termes $u_1, \cdots, u_n$.

Exercice 8 *Coefficients binomiaux*

1. Créer la fonction Python `facto(n)` donnée par

```
def facto(n):
    P=1
    for i in range(1,n+1):
         P = P*i
    return P
```

Que retourne-t-elle ?

2. On rappelle la formule pour $n \in \mathbb{N}$ et $k \in [\![0, n]\!]$:
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$
Créer une fonction Python `binome(n,k)` qui renvoie $\binom{n}{k}$ pour $k \in [\![0, n]\!]$ et 0 sinon. On pourra utiliser la fonction `facto`.
3. Créer une fonction Python `somme_coeff_binomiaux(n)` qui retourne la somme
$$\sum_{k=0}^{n} \binom{n}{k}.$$
Que peut-on conjecturer ? Le démontrer mathématiquement.

V Compléments

Exercice 9 *Affichage des nombres premiers*

1. Écrire une fonction `est_premier(p)` qui détermine si un nombre p est premier. Un nombre est premier s'il n'est divisible que par 1 et lui-même et 1 n'est pas un nombre premier. On pourra tester quels nombres divisent p avec la fonction `%`.
2. Écrire une fonction `liste_premiers(n)` qui renvoie la liste des nombres premiers inférieurs ou égaux à n.

Exercice 10 *Recherche dans une liste de façon linéaire*

1. Chercher dans l'aide l'utilité de la fonction `randint`.
2. Avec la question précédente, créer une fonction Python `liste_alea(n,m)` qui renvoie une liste de taille n d'entiers inférieurs ou égaux à m.
3. Écrire une fonction Python `presence(k,L)` qui retourne `True` si k est dans la liste L et `False` sinon.
4. Écrire une fonction Python `positions(k,L)` qui pour une liste d'entiers renvoie les positions de k dans cette liste. De façon plus précise, cette fonction doit retourner :

```
>>> positions(42, [12, 17, 42, 5])
[2]
>>> positions(24, [12, 17, 42, 5])
[ ]
>>> positions(42, [42, 12, 17, 42, 5])
[0, 3]
```

Exercice 11 *Triangle de Sierpinski*

Dans le cours, on a vu le triangle de Pascal. Si on écrit chaque coefficient binomial dans un tableau, on obtient le triangle de Sierpinski en coloriant chaque case contenant un nombre impair en noir.
En Python, un tableau est une liste de liste. Par exemple, un tableau avec $0, 1$ sur la première ligne et $1, 1$ sur la seconde s'écrit `tableau = [[0,1],[1,1]]`. Un élément peut être récupéré grâce à la commande `tableau[0][1]`.

1. Créer un tableau simple et tester la récupération d'éléments.
2. Créer une fonction Python `sierpinski(n)` donnant un tableau à n lignes et n colonnes comportant un 1 en position (i, j) si et seulement si $\binom{i}{j}$ est impair.
3. Pour créer une image blanche en Python, on peut utiliser

```
from PIL import Image
im = Image.new('RGB', (500,500),(255,255,255))
```

Tester ensuite la commande

```
for i in range(500):
    im.putpixel((i,100),(0,0,0))
im.show()
```

Expliquer ces différentes lignes de commande.
4. Créer une image du triangle de Sierpinski.
