Les fonctions

TP

Chapitre 2 - Fonctions v1

Lycée Jean Zay – 21 rue Jean Zay – 63300 Thiers – Académie de Clermont-Ferrand

1 Quelques exemples pour pratiquer

Exercice 1 *On considère la fonction* **mystere** *definie par :*

```
1 def mystere (k,n) :
2     r=1
3     for i in range(n):
4         r=k**r
5     return r
```

Que retourne la fonction **mystere** *?*

Que retourne l'appel de **mystere** *(2,4) ?*

La variable r apparaissant dans la fonction **mystere** n'a de sens que dans la définition de cette fonction. On dit que la *portée* de la variable r est limitée au corps de la fonction et on dit que r est une variable *locale*.

À l'inverse, une variable dont la portée s'étend à l'ensemble du programme est dite *globale*.

Exercice 2 *Écrire une fonction* **fact** *telle que* **fact(n)** *retourne la valeur de* $n!$.

Écrire une fonction **binom** *prenant deux entiers naturels* n *et* k *comme arguments et retournant la valeur du coefficient binomial* $\binom{n}{k}$.

Écrire une fonction **somme** *telle que* **somme(n)** *retourne la valeur de* $\sum_{k=0}^{n}(-1)^k\binom{n}{k}$. *Quelle valeur doit-on logiquement obtenir ?*

On peut bien entendu appeler une fonction dans une autre fonction. Tester !

Exercice 3 *Écrire une fonction* **max3** *prenant en entrée trois réels, et renvoyant leur maximum.*

2 Fonctions de bibliothèques

La fonction **fact** définie précédemment est en réalité déjà connue de Python dans le sens où elle a déjà été écrite puis intégrée dans la distribution du langage sous le nom de **factorial**. L'ensemble de ces fonctions déjà écrites sont regroupées dans ce que l'on appelle la *bibliothèque standard*.

Ces différentes fonctions sont organisées en *modules* parmi lesquels on peut citer :

- **math** qui contient les fonctions habituellement utilisées en mathématiques,
- **array** pour manipuler des tableaux,
- **matplotlib** pour le tracer de courbe,
- **numpy** pour le calcul scientifique avec également sa sous-bibliothèque **linalg**...

Pour utiliser la fonction **factorial** de la bibliothèque **math**, on peut écrire :

```
>>> import math
>>> math.factorial(5)
120
```

```
>>> from math import *
>>> factorial(5)
120
```

```
>>> from math import factorial
>>> factorial(5)
120
```

Une documentation officielle en ligne sur les différents modules et leur usage en Python est disponible à l'adresse :

https ://docs.python.org

3 Un peu plus de complexité

Exercice 4

a) Pour tout réel a, calculer $\sum_{k=6}^{31} a^k$.

b) Écrire une fonction **geometrique** *prenant en entrée un réel a et retournant* $\sum_{k=6}^{31} a^k$.

Exercice 5

a) Écrire une fonction **somme** *prenant en entrée deux entiers a et b avec $a \leqslant b$ et renvoyant* $\sum_{n=a}^{b} n^3$.

b) Question culturelle : exprimer $\sum_{n=a}^{b} n^3$ *en fonction de a et de b.*

Dans l'exemple suivant, on voit qu'une fonction est en fait un objet comme un autre : il peut être donné en paramètre à une autre fonction !

Exercice 6 *Écrire une fonction* **somme2** *prenant en entrée une fonction f, deux entiers a et b avec $a \leqslant b$ et renvoyant* $\sum_{n=a}^{b} f(n)$.

Exemple : on veut calculer $\sum_{k=831}^{944} k^{10}$.

```
1 def f0(x):
2     return x**10
```

```
>>> somme1(f0,831,944)
367241911503651005721610202208 25
```

Exercice 7

a) Justifier que $2^{2^{2^2}} = 65536$.

b) Écrire une fonction **puissance_iteree** *prenant en entrée deux entiers naturels k et n et retournant la $n^{ième}$ puissance itérée de k, ie le nombre* $k^{k^{\cdot^{\cdot^{\cdot^{k}}}}}$ *formé de n exemplaires de k.*

c) Vérifier vos résultat en exécutant puissance_iteree(2,4).