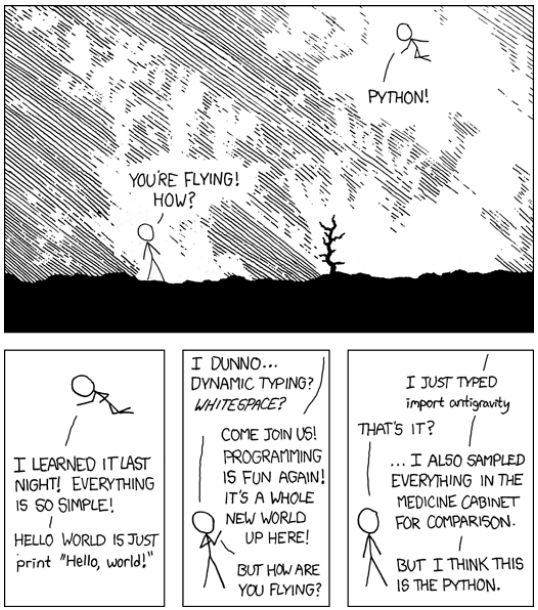
TP Info : Fonctions et procédures



1 Fonctions

1. On définit la fonction suivante :

```
def f(x):
    return x + 1
```

Pour appeler cette fonction, il suffit d'écrire `f(1)` ou `f(5)`.

(a) Essayez d'appeler `f` avec des valeurs différentes. Essayez en utilisant des valeurs de types différents.

(b) Quels sont les types de `x` pour lesquels l'évaluation de `f(x)` ne soulève pas d'erreur ?

(c) Le type de `f(x)` dépend-il uniquement du type de `x` ou aussi de sa valeur ?

2. On définit la fonction

```
def est_pythagoricien(a, b, c):
    alpha = a*a + b*b
    gamma = c*c
    return alpha == gamma
```

que l'on souhaite appeler pour des valeurs entières de a, b et c.

(a) Quel est le type de la valeur renvoyée par cette fonction ?

(b) Testez la fonction avec `est_pythagoricien(3, 4, 5)` ? Trouvez d'autres triplets pythagoriciens.

(c) Pourquoi est-ce une très mauvaise idée d'utiliser cette fonction avec des nombres flottants ? En s'inspirant de l'exemple précédent, trouvez des nombres flottants pour lesquels cette fonction renvoie un résultat très surprenant.

De manière générale, à chaque fois qu'on veut résoudre un problème qui dépend d'un ou plusieurs paramètres, on peut utiliser une fonction. La syntaxe

```
def nom_de_fonction(param_1, param_2, ..., param_n):
    # Bloc d'instructions
    return res
```

permet de définir une fonction appelée `nom_de_fonction`. L'appel de cette fonction permettra d'exécuter la suite d'instructions et renverra la valeur contenue dans `res`. Une fonction ne peut renvoyer qu'une valeur, mais cette valeur peut être un `tuple`. Une fois qu'une fonction a été définie, on peut l'appeler dans d'autres parties du code, de la même manière que l'on utilise les fonctions déjà définies en Python.

3. Créer une fonction `difference(a, b)` qui, pour deux valeurs a et b, renvoie $a - b$. Quel est le type de la valeur renvoyée par `difference(32.5, 41)` ?
4. Écrire une fonction `est_un_triangle(a, b, c)` qui renvoie `True` si a, b, c sont les côtés d'un triangle et `False` sinon. On rappelle que a, b, c sont les côtés d'un triangle si et seulement si ils vérifient les 3 inégalités triangulaires.
5. Écrire une fonction `perimetre_aire(a, b, c)` qui prend en arguments trois valeurs a, b et c. Si a, b, c sont les côtés d'un triangle, la fonction renverra le tuple formé du périmètre et de l'aire de ce dernier. Si ce n'est pas le cas, la fonction renverra `None`. On rappelle la formule de Héron donnant l'aire d'un triangle en fonction des longueurs de ses côtés :

$$\mathcal{A} = \sqrt{s(s-a)(s-b)(s-c)} \quad \text{avec} \quad s = \frac{a+b+c}{2}$$

2 Boucles for

La syntaxe

```
for i in range(a, b):
    # Bloc d'instructions (i)
```

permet de demander à l'ordinateur d'exécuter le bloc d'instructions, pour les valeurs $a, a+1, \dots, b-1$, lorsque a et b sont des entiers tels que $a \leqslant b$.

1. (a) Compléter les instructions suivantes de manière à définir la fonction `factorielle(n)` qui renvoie $n!$.

```
def factorielle(n):
    res = ___
    for i in range(1, ____):
        res = res * i
    return res
```

Une fonction peut fonctionner par effet de bord. Dans ce cas, elle ne renvoie rien et ne comporte pas de `return`. Par contre, elle peut afficher des choses à l'écran avec l'instruction `print`. Dans ce cas, on dit que la fonction est une procédure.

(b) Écrire une procédure qui affiche les valeurs des factorielles des nombres de 1 à n.

(c) Testez cette procédure sur quelques exemples de votre choix. Vérifier la cohérence de cette fonction avec la fonction `factorial` de la bibliothèque `math`.

2. Écrire une fonction `zeta3(n)` qui renvoie la valeur de

$$\sum_{k=1}^{n} \frac{1}{k^3}$$

lorsque $n \in \mathbb{N}^*$.

3. Écrire une procédure qui affiche la liste des 52 premiers chiffres (en base 2) après la virgule d'un nombre $x \in [1, 2[$.

3 Boucles while

Le programmeur ne peut utiliser de boucle `for` que lorsqu'il est capable de déterminer à l'avance le nombre de passages dans la boucle qu'il souhaite effectuer. Lorsque l'on veut interrompre les itérations au moment où une condition cesse d'être vérifiée, on utilise une boucle `while`. La syntaxe :

```
while condition:
    # Bloc d'instructions
```

permet de répéter le bloc d'instructions tant que la `condition` est vérifiée.

1. On admettra que

$$\lim_{n\to+\infty} \sum_{k=1}^{n} \frac{1}{\sqrt{k}} = +\infty$$

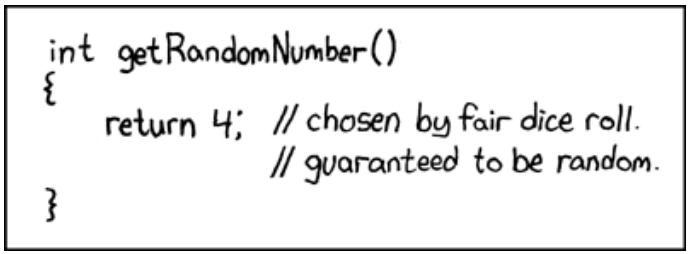
Soit $x \in \mathbb{R}$. Compléter la fonction suivante pour qu'elle renvoie le plus petit entier $n \in \mathbb{N}^*$ tel que :

$$\sum_{k=1}^{n} \frac{1}{\sqrt{k}} \geqslant x.$$

```
def depasse(x):
    somme = 0
    k = ____
    while somme ___ x:
        k = ___
        somme = somme + 1 / np.sqrt(k)
    return ___
```

2. On admet que la fonction `est_pile` renvoie un booléen de valeur `True` avec la probabilité 1/2.

```
def est_pile():
    return rd.random() <= 0.5
```



(a) Écrire une fonction qui simule une expérience aléatoire où un individu lancerait une pièce équilibrée et chercherait le nombre de lancers nécessaires pour obtenir le premier Pile.

(b) Estimer le nombre moyen de lancers nécessaires pour obtenir pile.

3. Programmez l'algorithme de division euclidienne par différences successives.

4. Écrire une procédure qui affiche tous les chiffres de la décomposition en base 2 d'un nombre entier naturel.

5. Écrire une fonction qui, lorsque a et b sont des entiers naturels non nuls tels que $a < b$, renvoie la période de l'écriture décimale de a/b.

4 Suites récurrentes

4.1 Suites récurrentes simples

1. On considère la suite u définie par $u_0 = 25$ et la relation de récurrence

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \sqrt{2u_n + 3}.$$

(a) Écrire une fonction `u(n)` qui renvoie la valeur de u_n. On utilisera une boucle for.

(b) Écrire une fonction remplissant la même fonction mais utilisant une boucle `while`.

(c) Que valent u_{100} ? u_{1000} ? u_{10000} ?

(d) Que peut-on conjecturer sur le comportement asymptotique de cette suite ?

(e) Donner en fonction de n le nombre d'appels à `sqrt` utilisés par `u(n)`.

2. Écrire une fonction `v(a, n)` qui renvoie le terme d'indice n de la suite (v_n) définie par la relation de récurrence :

$$v_0 = a \quad \text{et} \quad \forall n \in \mathbb{N}, \quad v_{n+1} = \sqrt{2v_n + 3}$$

(a) Tester cette fonction pour plusieurs valeurs de a.

(b) Que peut-on conjecturer du comportement asymptotique de cette suite suivant les valeurs de a ?

3. Écrire une fonction `w(a, n, f)` qui renvoie le terme d'indice n de la suite (w_n) de premier terme a, et vérifiant la relation récurrence

$$\forall n \in \mathbb{N}, \quad w_{n+1} = f(w_n).$$

On passera comme paramètre effectif de `f` le nom d'une fonction déjà définie. Testez cette fonction sur plusieurs exemples de votre choix.

4.2 Suites récurrentes croisées

On choisit deux réels a et b tels que $0 \leqslant a \leqslant b$. On définit les suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ en posant :

$$u_0 = a, \quad v_0 = b, \quad \forall n \in \mathbb{N},\ u_{n+1} = \sqrt{u_n v_n} \quad \text{et} \quad v_{n+1} = \frac{u_n + v_n}{2}\,.$$

1. Complétez la fonction suivante pour qu'elle renvoie la valeur (u_n, v_n) lorsque les réels a et b et l'entier n sont passés en paramètre.

```
def arithmetico_geometrique(a, b, n):
    u, v = ___
    for k in range(___):
        u, v = ___
    return u, v
```

2. Étudier la convergence de chacune de ces suites pour différentes valeurs de a et b.
3. On peut montrer que le suites (u_n) et (v_n) convergent vers une même limite. Écrire une fonction `rang(a, b, eps)` qui renvoie la première valeur de n telle que $|u_n - v_n| \leqslant \varepsilon$, lorsque ε est un réel strictement positif.
4. Que conjecturer sur les variations de u et v ?
5. Écrire une fonction `verifie_monotonie(a, b, n)` renvoyant `True` si la conjecture précédente se vérifie jusqu'au rang n et `False` sinon.

5 Quelques calculs de sommes et produits

1. On pose, pour tout $n \in \mathbb{N}$

$$M_n = \sum_{i=1}^{n} \sum_{j=1}^{2i} \max(i, j).$$

   (a) Écrire une fonction qui renvoie M_n lorsque n est passé en paramètre.

   (b) Donner en fonction de n le nombre d'appels effectués à la fonction `max` lors du calcul de M_n.

   (c) Écrire une fonction renvoyant

$$A_n = \sum_{i=1}^{n} \max(2i, n) + \sum_{j=1}^{n} \max(3j, n).$$

   Combien d'appels à `max` cette fonction utilise-t-elle ?

   (d) Écrire une fonction calculant

$$G_n = \sum_{k=1}^{n} \left(\prod_{j=1}^{n} (1 + 3kj)^2 \right).$$

   Que valent G_{10} ? et G_{100} ?