Initiation interactive à Python : Défis de programmation - 1

Résoudre des défis de programmation est un bon moyen de progresser rapidement. Nous proposons ici de détailler comment résoudre certains des problèmes disponibles sur PyDéfis. Là aussi, il existe beaucoup d'autres sites de ce type, francophones ou non.

Vous pouvez vous inscrire du PyDéfis si vous voulez pouvoir soumettre des solutions. Si vous voulez simplement consulter les énoncés, en revanche, pas besoin d'inscription.

Bien qu'ils ne soient pas très compliqués, les exercices proposés sur PyDéfis nécessitent généralement un peu plus de connaissances que ce que nous avons vu pour le moment. Il n'est donc pas inutile de se munir d'un tutoriel général sur Python, comme le document mentionné plus haut. D'autres explications seront données au fur et à mesure dans ce texte.

Programmer un algorithme

Lien vers l'énoncé : Algorithme du professeur Guique

Ce premier exercice est facile et ne contient rien que nous n'ayons encore vu. Il s'agit simplement de traduire un calcul répétitif donné sous la forme d'un algorithme, en Python. Les variables `a`, `b`, `c`, `i`, et `n` doivent être initialisées à des valeurs d'entrées données comme entrées au problème. Ces valeurs peuvent varier. Supposons qu'elles sont égales à 1,2,4,1 et 0 :

```
a = 1
b = 2
c = 4
...
```

On peut faire plus court, car Python possède un opérateur pour faire «simultanément» plusieurs affectations :

```
a,b,c,i,n = 1,2,4,1,0
```

Il faut ensuite répéter le même bloc d'instructions, tant que i est inférieur à `1000-n`, puis afficher les valeurs qui seront contenues dans `a`, `b` et `c`, ce qui constitue la réponse à la question :

```
while i < 1000 - n :
   BLOC À RÉPÉTER
print(a,b,c)
```

Le bloc à répéter ne contient que des affectation et des opérations élémentaires. Le contenu de ce bloc peut varier. Voici un exemple :

```
            a=b
            b=c+a
```

```
c=(2)*c+(3)*a-b
n=a+b
augmenter i de 1
```

Donnera en Python :

```
a = b
b = c + a
c = 2 * c + 3 * a - b
n = a + b
i = i + 1
```

Pour valider la réponse sur Pydéfis, il faut indiquer les 3 nombres séparés par des virgule.

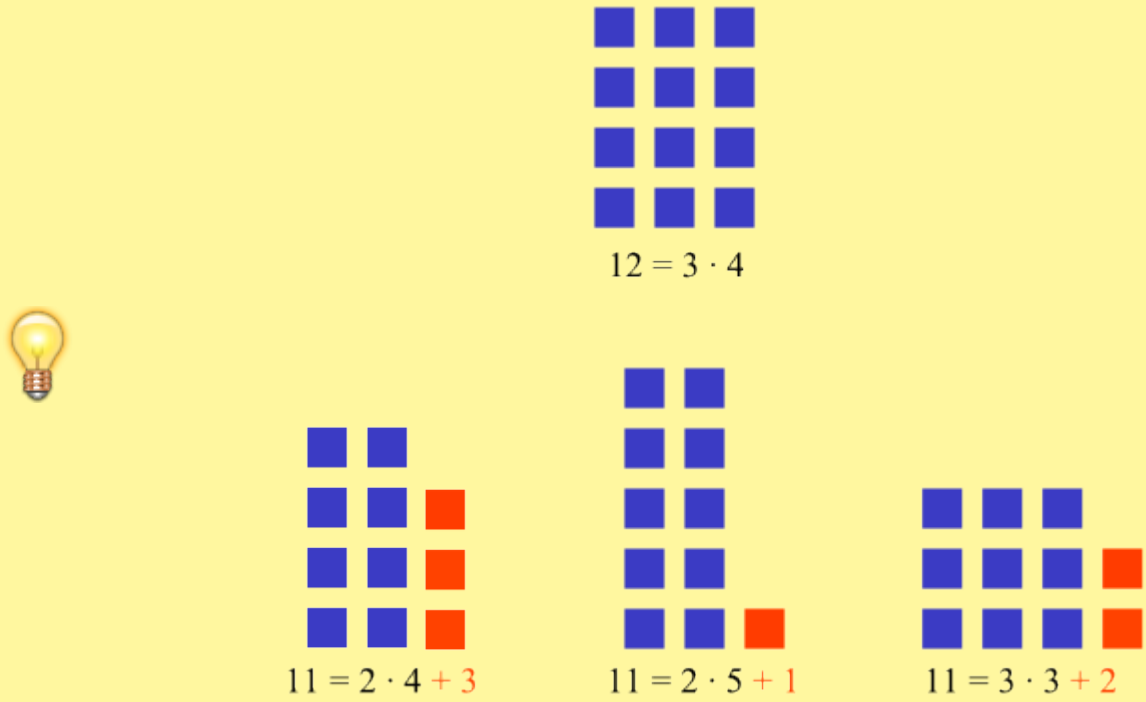
Si vous tentez de valider l'exercice, vérifiez bien votre énoncé : opérations arithmétiques dans le bloc `while` et valeurs de départs. Ils peuvent varier par rapport à ce qui est proposé ici.

Premier particulier

Lien vers l'énoncé : Premier particulier (1)

Dans ce défi, nous devons trouver les nombres de la forme (par exemple) 3k+7 où k est un nombre entier. Ces nombres sont : 3*0+7, 3*1+7, 3*2+7, 3*3+7, 3*4+7 etc... c'est à dire 7, 10, 13, 16, 19... Parmi les nombres obtenus (il y en a une infinité), certains sont des nombres premiers.

> Les nombres premiers sont les nombres qui ne sont divisibles que par 1 et eux-mêmes. Une autre façon de voir les chose est de dire que si vous avez un nombre de morceaux de sucre qui est premier, vous ne pouvez pas les disposer, tous à plat, de manière à faire un rectangle, sauf en les mettant tous en ligne. Ainsi, 12 n'est pas premier, mais 11 est premier.



Vous trouverez des informations complémentaires sur les nombres premiers ici : Nombre Premier.

Pour tester si on nombre est premier, nous pouvons essayer de le diviser successivement par 2, 3, 4, etc... Si une des division, par exemple la division par 6, tombe juste, cela signifie que le nombre est divisible par 6. Or dire qu'une division tombe juste c'est dire que le reste de la division est nul. Et nous avons justement un opérateur qui calcule le reste d'une division en Python : %.

Ainsi, pour savoir si le nombre n est premier, il suffit de calculer les restes des division de n par 2, puis 3, puis 4... jusqu'à n-1. Si un des restes est nul, le nombre n'est pas premier. Si aucun reste n'est nul, le nombre n est premier.

Il est possible d'améliorer cette idée, et arrêtant de calculer les restes plus tôt (on peut s'arrêter à l'arrondi par défaut de $\sqrt{n}$). Si vous savez comment faire, essayez. Sinon, ce n'st pas grave, nous pouvons résoudre le défi sans savoir ça...

Voyons d'abord comment tester si un nombre est premier :

```
n = 10
i = 2
premier = True
while i<n :
    if n % i == 0 :
        premier = False
if premier :
    print("Le nombre",n,"est premier")
```

```
else :
    print("Le nombre",n,"n'est pas premier (il est composé)")
```

Il y a juste deux choses nouvelles dans ce programme, ce sont les valeurs `True` et `False`. Ces deux valeurs sont, en français : *Vrai* et *Faux*. Une variable qui peut valeoir *Vrai* ou *Faux* est une variable *booléenne* (comme il y a des variables entières, qui contiennent des entiers, il y a des variables booléennes, qui contiennent des booléens).

Le test final : `if premier` est une autre façon d'écrire `if premier == True`. Observez attentivement le programme. Voyez comme la valeur de `premier` est initialisée à `True`. Puis on procède à toutes les divisions. Et si une des divisions tombe juste, on passe `premier` à `False`. Après les tests, si `premier` est à `False`, c'est qu'au moins une des division est tombée juste, et donc que le nombre n'est pas premier.

Notre programme marche bien, mais si nous devons maintenant vérifier si plusieurs nombres sont premiers, il n'est pas facile à utiliser. Pour nous simplifier la vie, nous allons écrire notre première *fonction*. Pour l'instant nous pouvons considérer qu'écrire une fonction, c'est rajouter une nouvelle fonctionnalité à Python. Nous allons lui ajouter la possibilité de nous dire si n'importe quel nombre donné est premier. ...

À suivre...