Algorithmique et programmation en Python

Présentation des Notebooks

Les notebooks, comme celui-ci que vous venez d'ouvrir, sont des pages web interactives dans lesquelles on peut taper du texte courant, du texte mathématiques, ainsi que du code informatique.

Faites une copie du notebook avec la commande ci dessus "File>Make a Copy...", et renommez-le en cliquant sur le titre (à côté de jupyter TD1...), par exemple en rajoutant votre nom.

On peut entrer des expressions (opérations, calculs) ou des instructions (des commandes) dans tous les champs ci-dessous qui commencent par In[..]. Pour taper plusieurs lignes de code à la suite, on appuie sur la touche entrée entre deux lignes (rien de spécial donc).

Le résultat s'obtient avec :

- ctrl + entrée, dans ce cas on reste dans la cellule courante;
- majusucule + entrée, dans ce cas on passe à la cellule suivante.
- Après exécution, le résultat d'un programme est souvent précédé de Out[ ].
  Exemple : tapez ctrl entrée après avoir cliqué dans la cellule ci-dessous

In [ ]:
```
nom = input("Comment vous appelez-vous ? ")
print("Salut ", nom)
```

En cas de plantage : Si lors d'une tentative d'éxécution de programme, rien ne se passe, c'est probablement que vous avez tapé un programme qui boucle infiniment. Dans ce cas, commencez par sauver votre travail. Puis deux possibilités:

- essayez d'abord de redémarrer le noyau Python (restart dans le menu kernel en haut de la page)
- sinon, allez dans l'onglet "Home", et fermez le TD : cocher la case correspondante, "shutdown" en haut de la page. Fermez ensuite l'onglet du TD, et relancez-le à partir de "Home".
  Si vous voulez reprendre ce TD chez vous, le plus simple est d'installer le navigateur Anaconda (puis cliquer sur launch jupyter notebook etc.).

Exemple d'un algorithme du bac et sa traduction en Python, boucle "pour"

Calcul d'un terme de rang donné d'une suite
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} u_0 = 4 \\ u_{n+1} = 3 - \frac{4}{u_n+1} \end{cases}$$

Calculer u_{10}.

Solution avec une boucle "pour"

Algorithme en langage "naturel":

Algorithme de calcul d'un terme de rang donné d'une suite
entrée : rien
sortie : affiche le terme de rang 10 de la suite définie par $u_0 = 4$ et $u_{n+1} = 3 - \frac{4}{u_n+1}$
Début

> *u* ← 4
> **Pour** *i* **de** 1 **à** 10 **faire**:
>
> > *u* ← 3 − 4÷(u + 1)
>
> **Fin pour**
> **afficher** *i* # n'est pas demandé mais permet une vérification
> **afficher** *u*

Fin

Programme Python correspondant :

In [ ]:
```
u=4
for i in range(1,11):   # attention fin a 11, range(1,11) corres
pond a [1,11[ soit [1,10]
    u=3-4/(u+1)
    print("rang ",i, "terme ",u)    # non demandé, permet l'affi
chage des variables lors
                                    # de l'execution
print("le terme u",i,"vaut ", u)
```

Exercice

Recopier et modifier l'algorithme précédent pour qu'il calcule le terme de rang 42 de la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = 1,2 \times u_n + 1 \end{cases}$$

Le programmer en Python ci-dessous, ne pas afficher les termes non demandés (cela risque de faire planter jupyter).

Remarques :

- on pourra copier/coller le programme donné en exemple. Les raccourcis clavier sont ctrl C pour copier , ctrl V pour coller
- il faut faire figurer l'opération de multiplication, comme sur la calculatrice.
- la notation des nombres décimaux est anglo-saxonne : on tape 1.2

In [ ]:
```

```

Exemple d'un algorithme du bac avec une boucle "tant que"

Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par :
$\begin{cases} u_0 = 1 \\ u_{n+1} = 1,2 \times u_n + 1 \end{cases}$
On peut conjecturer grâce à l'exercice précédent que cette suite tend vers $+\infty$.
Trouver le plus petit rang n tel que $u_n \geqslant 1000000$.

Solution

Algorithme en langage naturel

Algorithme de calcul d'un rang pour lequel une suite dépasse une certaine valeur
entrée : rien
sortie : affiche le rang du terme qui dépasse 1 000 000, pour la suite définie par $u_0 = 1$ et $u_{n+1} = 1,2 \times u_n + 1$,

Début

> $u \leftarrow 1$
> $i \leftarrow 0$
> **Tant que** $u < 1000000$: # attention à < ou $\leqslant$, vérifier qu'on obtient le bon indice, et le bon terme
>
> > $u \leftarrow 1,2 \times u + 1$
> > $i \leftarrow i + 1$
>
> **Fin tant que**
> **afficher** i # n'est pas demandé mais permet une vérification
> **afficher** u

Fin

Programme Python correspondant :

In [ ]:
```
u=1
i=0
while u < 1000000:
    u=1.2*u + 1
    i = i+ 1

print("le terme u(",i,") vaut", u," et est le premier à dépasser
1 000 000")
```

Modifier l'exemple ci dessus pour trouver le plus petit rang n tel que $u_n \geqslant 10^{35}$
Remarque : 10^{35} s'écrit `10**35` en python.

Complément : calcul d'un terme de rang donné avec une boucle "tant que"

On reprend le premier exemple
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par :
$\begin{cases} u_0 = 4 \\ u_{n+1} = 3 - \frac{4}{u_n+1} \end{cases}$
Calculer u_{10}.

Algorithme en langage "naturel":

Algorithme de calcul d'un terme de rang donné d'une suite
entrée : rien
sortie : affiche le terme de rang 10 de la suite définie par $u_0 = 4$ et $u_{n+1} = 3 - \frac{4}{u_n+1}$

Début

$u \leftarrow 4$
$n \leftarrow 0$
Tant que $n < 10$: # attention à < ou $\leqslant$, vérifier qu'on obtient le bon indice, et le bon terme

$u \leftarrow 3 - 4 \div (u + 1)$
$n \leftarrow n + 1$

Fin tant que
afficher n # n'est pas demandé mais permet une vérification
afficher u

Fin

Programme Python correspondant :

In [ ]:
```
u=4
n=0
while n < 10:
    u=3-4/(u+1)
    n = n + 1
    print("rang",n, "terme",u)    # non demandé, permet l'affich
age des variables lors
                                  # de l'execution

print("le terme u",n,"vaut", u)
```

Les instructions en algorithmique...

Les instructions autorisées pour écrire un algorithme sont :

- Affectation de variables (symbolisées par ←)
- Instruction itérative (répétitive, boucle):

```
Tant que condition faire
…
fin tant que.
```

  Rq : C'est la boucle la plus générale.
- Instructions itératives complémentaires:

```
Pour variable de valeur de début à valeur de fin faire
…
fin pour

Répéter
…
jusqu'à condition
```

- Instruction conditionnelle (test):

```
Si condition alors
…
(sinon
…)
fin si
```

- Entrée/sortie :

```
Lire au clavier
Afficher/imprimer à l'écran
```

Pour chaque conditionnelle, chaque répétitive, les instructions doivent être décalées, et un trait doit limiter la séquence d'instructions à effectuer à l'intérieur de cette conditionnelle/répétitive.

Chaque algorithme doit être accompagné de spécifications. Celles ci précisent les données en entrée de l'algorithme, le résultat de celui-ci. Quelques commentaires peuvent (dans les cas compliqués doivent) éclaircir les instructions délicates dans le corps de l'algorithme. Vous pouvez les signaler avec le #, puisque c'est le symbole que l'on utilise en Python.

... et leur traduction en Python

Opérations : +, - ,*, /, // (division entière), ** (puissance), % (reste dans la division entière)

Affectation d'une variable : a prend la valeur 2 s'écrit `a = 2`

Lire la valeur d'une variable au clavier :

- Si la variable est un entier : `n = int(input("Entrer un entier : "))`
- Si la variable est un réel : `x = float(input("Entrer un réel : "))`
- Si la variable est une chaine de caracteres : `nom = input("Entrer votre nom : ")`

Afficher un texte et/ou une variable : `print("texte ",n)`

si *condition* **alors** *instruction* (**sinon** *instruction*) :

`if` *condition* :

```
    instruction 1
```

`(else :)`

```
    instruction 2
```

suite du programme

opérateurs de comparaison dans les tests (`if` et `while`):

- `x == y` # x est égal à y
- `x != y` # x est différent de y
- `x > y` # x est plus grand que y
- `x < y` # x est plus petit que y
- `x >= y` # x est plus grand que, ou égal à y
- `x <= y` # x est plus petit que, ou égal à y

Boucle tant que :
`while` *condition* :

```
    instruction
```

suite du programme

Boucle pour :
`for` *variable* `in` *séquence*:

```
    instruction
```

suite du programme

Exercices

Les exercices suivants sont basés sur des suites très simples, souvent arithmétiques ou géométriques, ceci pour que vous puissiez travailler l'agorithmique sans vous soucier de difficultés techniques de nature mathématique. En conséquence, dans l'optique de travailler des questions de type bac -qui porteront sur des suites plus compliquées-, vous ne devez utiliser que les formules du cours les plus simples. C'est à dire uniquement $u_{n+1} = u_n + r$ pour les suites arithmétiques ; et $u_{n+1} = q \times u_n$ pour les suites géométriques. Toute autre formule utilisée ne vous permettra pas de travailler correctement.

Pour **tous les exercices suivants, écrire l'algorithme en langage naturel** (en conserver la trace dans votre cahier d'exercices). Vous pouvez écrire les algorithmes avant ou après avoir écrit le code en python, et tester le code dans la cellule en dessous de l'énoncé. N'hésitez pas à comparer les résultats avec vos voisins, cela peut être très instructif. Ceux qui n'écrivent pas les algorithmes auront leur moyenne divisée par 5.

Une des questions centrales à se poser, en ce qui concerne les algorithmes du bac, est de savoir si l'on utilise plutôt une boucle "pour" ou une boucle "tant que". Pour y répondre, on peut remarquer que l'on utilise les boucles "pour" lorsque l'on sait quand on doit s'arreter. Alors qu'une boucle "tant que" est utilisée lorsque l'on ne sait pas à quelle itération on doit sortir de la boucle.

Exercice 1

Arthur possède 1000 € sur son compte et en rajoute 25 euros tous les mois. Écrire un algorithme qui donne la somme totale sur son compte au bout de n mois, n étant rentré au clavier. Compléter le programme ci-dessous correspondant à cet algorithme.

Rappel : interdit d'utiliser une formule de sommation, l'idée est d'utiliser une boucle pour sommer mois par mois.

In [ ]:
```
n = int(input("Donner le nombre de mois : "))    # pour rentrer
un nombre au clavier
somme = 1000
for i in range():
```

Exercice 2

Soit (u_n) la suite géométrique de premier terme $u_0 = 5$ et de raison $0,8$. Écrire un algorithme qui donne la somme des 10 premiers termes. Le programmer en Python.

Rappel : interdit d'utiliser une formule de sommation, l'idée est d'utiliser une boucle pour sommer terme par terme.

In [ ]:
```

```

Exercice 3.1

Soit (u_n) la suite définie par $u_0 = 1$ et $u_{n+1} = 2 \cdot u_n - 2$.
Écrire un algorithme qui calcule le terme de rang *N*, *N* étant rentré au clavier. Le programmer en Python.

In [ ]:

Exercice 3.2

Soit (u_n) la suite définie par $u_0 = 1$ et $u_{n+1} = 2 \cdot u_n + 3 \cdot n - 2$.
Écrire un algorithme qui calcule le terme de rang *N*, *N* étant rentré au clavier. Le programmer en Python.

In [ ]:

Exercice 4

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_n = \frac{n^2\sqrt{n}}{n+1}$.
On admet que cette suite diverge vers $+\infty$.
Soit *A* un réel fixé. Ecrire un algorithme qui calcule le plus petit indice *n* tel que $u_n > A$. Le programmer en python.
Tester avec *A* = 800, *A* = 10000, $A = 5 \cdot 10^6$.

In [ ]:
```
from math import *      # a laisser en debut de programme pour p
ouvoir calculer les racines carrées.
                        # racine carré de 2 s'écrit sqrt(2) ; co
mme square root
```

Exercice 5

Même exercice que le précédent avec la suite u_n définie pour tout entier naturel n par
$\begin{cases} u_0 = 1 \\ u_{n+1} = 1,2 \cdot u_n + 5 \end{cases}$

In [ ]:

Exercice 6

On considère la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par $u_n = 1 + \frac{1}{n^2}$.
On admet que cette suite est décroissante et converge vers 1.
Ecrire un algorithme qui détermine le plus petit indice n tel que $u_n - 1 < 10^{-5}$. Le programmer

In [ ]:

Exercice 7

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par :
$\begin{cases} u_0 = 1 \\ u_{n+1} = \frac{u_n}{2} + 3 \end{cases}$

On admet que cette suite converge vers 6.
Ecrire une algorithme qui détermine le plus petit indice n tel que $u_n \in \left]6 - 10^{-5}; 6 + 10^{-5}\right[$.
Remarque : En algorithmique, et dans la majorité des langages de programmation, on ne peut pas faire de test avec une double inégalité (comme $6 - 10^{-5} < u < 6 + 10^{-5}$).
On utilise alors le "et" logique, en Python cela donne : `(u > 6 - 10**-5) and (u < 6 + 10**-5)`

In [ ]:

Exercice 8

Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par :
$\begin{cases} u_0 = -1 \\ u_1 = 0,5 \\ u_{n+2} = u_{n+1} - 0,25 \cdot u_n \end{cases}$
Ecrire un algorithme qui donne le terme de rang *N*, *N* étant rentré au clavier.

In [ ]:

Frederic Mandon (mailto:frederic.mandon@ac-montpellier.fr), Lycée Jean Jaurès - Saint Clément de Rivière - France (2015)