1 Objectifs d'algorithmique

a) Objectifs du bac

- **Savoir exécuter un algorithme** : un algorithme est donné, et l'énoncé demande de détailler son exécution, en détaillant les calculs ou éventuellement en complétant un tableau.
- **Savoir comprendre un algorithme**
  - L'énoncé indique ce que doit faire l'algorithme, et plusieurs algorithmes sont proposés, il faut indiquer lequel convient et pourquoi les autres ne conviennent pas.
  - Un algorithme est donné et on demande ce que fait cet algorithme.
- **Savoir exécuter et comprendre un algorithme :** un algorithme est donné, et l'énoncé demande ce que va afficher cet algorithme.
- **Savoir compléter un algorithme :** un algorithme incomplet est donné, et l'énoncé indique ce que doit faire l'algorithme, et il faut le compléter.

b) Objectifs supplémentaires des TP de programmation en Python 3

- **Savoir traduire un programme en algorithme ou un algorithme en programme** L'énoncé donne un programme en Python ou en langage de la calculatrice, et il faut le traduire en algorithme, ou inversement.
- **Savoir concevoir un algorithme :** l'énoncé indique ce que doit faire l'algorithme et il faut créer un algorithme (ou un programme en Python).

2 Variables et affectations

a) Un exemple, avec la calculatrice

On saisit à la calculatrice : `5→A`, ce qui veut dire « stocker 5 dans A ».

On peut vérifier la valeur de `A` en saisissant simplement `A`.

Saisissons maintenant : `A+1→A`, et la valeur de `A` est maintenant 6.

```
5→A
                    5
A
                    5
A+1→A
                    6
```

Ce qui s'est passé est tout simple, mais précisons bien ce qui s'est passé :

- quand on stocke 5 dans la variable `A` on peut considérer qu'il y a une « boîte » avec l'étiquette « `A` » où on a stocké 5 : 

;
- quand on exécute ensuite l'instruction `A+1→A`,
  - on prend le contenu de `A` et on ajoute 1 : $5 + 1 = 6$;
  - on efface le contenu de la variable `A` ;
  - on remplace ce contenu par 6, ce qui donne : 

.

b) Vocabulaire

Définition 1.1 (Variable et affectation)

Dans l'exemple ci-dessus,
- on appelle `A` une **variable**,
- et le fait de stocker un nombre dans `A` est appelé une **affectation**.

c) Différentes façons d'exprimer l'affectation

Calculatrice	`5→A`	Stocker 5 dans `A`
Algorithme	$A \leftarrow 5$	A prend la valeur 5
Python 3	`A=5`	A prend la valeur 5

3 Fonction en Python 3

Exemple

	Algorithme	**Fonction en Python 3**
1		`def f(x):`
2	$y \leftarrow 4x - 7$	`    y=4*x-7`
3		`    return y`

Remarques

- Ligne 1 : dans l'instruction `def f(x):` bien noter la présence des deux points en fin de commande.
- Ligne 2 : remarquer le décalage de l'instruction du `y=4*x-7` par rapport à l'instruction `def f(x):` On appelle cela l'**indentation**, elle est indispensable.

Exécution de la fonction dans EduPython (Windows)

Quand on a saisi la fonction ci-dessus dans EduPython, on exécute le fichier en cliquant sur le bouton avec un triangle vert.

Dans la console, on voit le message ci-dessous.

```
*** Console de processus distant Réinitialisée ***.
```

Pour exécuter la fonction `f`, on saisit par exemple `f(10)` dans la console, et on valide (appuyer sur la touche Entrée). On voit :

```
>>> f(10)
33
```

Cette réponse correspond bien au calcul de l'image de 10 par la fonction f : $f(10) = 4 \times 10 - 7 = 33$

Exécution de la fonction dans Spyder 3

Quand on a saisi la fonction ci-dessus dans Spyder 3 on exécute le fichier en cliquant sur le bouton avec un triangle vert, il faut alors enregistrer ce fichier.

Dans la console, on voit quelque chose qui ressemble à ceci :

```
In[1]: runfile('/home/nom-dossier/nom-fichier.py', wdir='/home/nom-dossier')
In[2]:
```

Pour exécuter la fonction `f`, on saisit par exemple `f(10)` dans la console, et on valide (appuyer sur la touche Entrée). On voit alors ce qui est en dessous :

```
In[2]: f(10)
Out[2]: 33
```

Cette réponse correspond bien au calcul de l'image de 10 par la fonction f : $f(10) = 4 \times 10 - 7 = 33$

Vocabulaire : renvoyer une valeur

Quand on saisit à la console `f(10)` et que le résultat est 33, on dit que la fonction `f` a **renvoyé** la valeur 33.

4 Variables, affectations et opérations en Python 3

Exemple :

Dans l'algorithme et dans la fonction Python ci-dessous, x et y sont des nombres réels.

	Algorithme	**Fonction en Python 3**
1		`def f(x):`
2	$y \leftarrow \dfrac{x^2+3x-6}{10}$	`    y=(x**2+3*x-6)/10`
3		`    return y`

a) Variables

- Les variables de cet algorithme et de cette fonction sont x et y.
- **Déclaration des variables : on ne déclare pas les variables en Python** alors que c'est obligatoire dans d'autres langages comme le *Java* ou le *C*.

b) Affectation

L'instruction $y \leftarrow \dfrac{x^2+3x-6}{10}$ signifie y prend la valeur $\dfrac{x^2+3x-6}{10}$.

En Python, cette instruction s'écrit : `y=(x**2+3*x-6)/10`

Le signe = en Python est un signe d'affectation, il n'a donc pas la même signification que le signe = en mathématiques.

c) Opérations en python3

Les quatre signes d'opérations s'écrivent : `+  -  *  /`

Attention, en Python, le signe multiplié n'est pas sous entendu, par exemple $3a$ s'écrit `3*a`.

Les puissances s'écrivent avec le signe `**`, par exemple : 3^4 s'écrit `3**4`

5 Écriture des algorithmes au baccalauréat

Depuis le baccalauréat 2018,

- les algorithmes sont écrits **sans les entrées et sorties** ;
- les variables sont indiquées dans l'énoncé ;
- mais les algorithmes sont écrits **sans la déclaration des variables** ;
- pour l'affectation, on utilise le signe $\leftarrow$, par exemple *a prend la valeur 5* s'écrit $a \leftarrow 5$.

6 Instruction conditionnelle (si ... alors ...)

Exemple :

Supposons qu'un tarif de location de vélo soit de 3 euros par heure pour moins de 5 heures, et qu'à partir de 5 heures de location, on paie un prix fixe de 15 euros.

On définit alors ci-dessous la fonction `tarif`.

Dans l'algorithme et dans la fonction Python ci-dessous, x est le nombre d'heures et y est le prix payé. x et y sont des nombres réels.

	Algorithme	**Fonction en Python 3**
1		`def tarif(x):`
2	Si $x < 5$	`    if(x<5):`
3	alors $y \leftarrow 3x$	`        y=3*x`
4	sinon	`    else:`
5	$y \leftarrow 15$	`        y=15`
6	Fin du Si	
7		`    return y`

À retenir

- Les lignes 2 à 6 constituent ce qu'on appelle une **instruction conditionnelle** :

  Si (*condition*)
  alors *faire ceci*
  sinon *faire cela*
  Fin du Si

- Dans une instruction conditionnelle, la condition comporte un signe comme $=$ ou $<$ ou $\leqslant$, etc. Voici la traduction de ces signes en python :

Signe	$=$	$<$	$\leqslant$	$>$	$\geqslant$
Python3	`==`	`<`	`<=`	`>`	`>=`

- Ligne 2 : dans la commande `if(x<5):` bien noter la présence des deux points en fin de commande.
- Ligne 3 : en Python, on n'écrit pas le *alors*.
- Ligne 3 : sur cette ligne, l'instruction `y=3*x` est indentée par rapport au `if`.
- Ligne 4 : comme à la ligne 2, il y a deux points après `else`
- Ligne 5 : comme à la ligne 3, la commande `y=15` est indentée
- Ligne 6 : dans le programme Python cette ligne est vide parce qu'en Python, on indique pas la fin du Si, elle est indiquée par l'indentation.

7 Boucle bornée (boucle Pour)

Exemple :

Dans l'algorithme et dans la fonction Python ci-dessous, u est un nombre réel et k et n sont des entiers naturels.

	Algorithme	**Fonction en Python 3**
1		`def calcul(n):`
2	$u \leftarrow 1$	`    u=1`
3	Pour $k = 0$ jusqu'à $k = n - 1$	`    for k in range(n):`
4	$u \leftarrow 3u + 4$	`        u=3*u+4`
5	Fin du Pour	
6		`    return u`

Exemple d'exécution de cette fonction `calcul`

Fixons la valeur de n à 5.

Quand on exécute cette fonction en saisissant `calcul(5)` le résultat est 727.

Détaillons l'exécution pas à pas.

Au début (ligne 2), u prend la valeur 1.

La boucle Pour des lignes 3, 4 et 5 indique que l'instruction $u \leftarrow 3u + 4$ est répétée pour $k = 0$, puis $k = 1$, puis $k = 2$, puis $k = 3$, puis $k = 4$, c'est à dire 5 fois.

Après cela l'instruction `return u` est exécutée.

Détaillons tout cela :

Début :	$u \leftarrow 1$
$k = 0$	$u \leftarrow 3 \times 1 + 4 = 7$
$k = 1$	$u \leftarrow 3 \times 7 + 4 = 25$
$k = 2$	$u \leftarrow 3 \times 25 + 4 = 79$
$k = 3$	$u \leftarrow 3 \times 79 + 4 = 241$
$k = 4$	$u \leftarrow 3 \times 241 + 4 = 727$
Fin :	$u = 727$

La variable u a donc pris successivement les valeurs 1, puis 7, puis 25, puis 79, puis 241, puis 727, et la valeur retournée qui est 727 est donc la dernière valeur prise par la variable u.

À retenir

- Les lignes 3 à 5 constituent une **boucle bornée** ou une **boucle Pour** :

  Pour $k = p$ jusqu'à $k = n$
  faire ceci
  faire cela
  Fin du Pour

- En Python, pour un entier naturel n,
  l'instruction `for k in range(n):` signifie *Pour* $k = 0$ *à* $k = n - 1$.
- En Python,
  pour obtenir l'instruction *Pour* $k = 1$ *à* $k = n$, il faut saisir : `for k in range(1,n+1):`
- Plus généralement, en Python,
  l'instruction *Pour* $k = p$ *jusqu'à* $k = n$ se traduit par : `for k in range(p,n+1):`
- Dans le programme ci-dessus, bien noter :
  - la présence des deux points à la fin de l'instruction `for k in range(n):`
  - l'indentation à la ligne 4 ;
  - le fait qu'aucune commande n'indique la fin du Pour, parce que c'est l'indentation qui l'indique.

8 Boucle non bornée (boucle Tant que)

Exemple

Dans l'algorithme et dans la fonction Python ci-dessous, k est un nombre entier naturel, u et A sont des nombres réels.

	Algorithme	**Programme en python3**
1		`def seuil(A)`
2	$k \leftarrow 0$	`    k=0`
3	$u \leftarrow 0$	`    u=0`
4	Tant que $u < A$	`    while(u<A):`
5	$k \leftarrow k + 1$	`        k=k+1`
6	$u \leftarrow k^3 + k^2$	`        u=k**3+k**2`
7	Fin du Tant que	
8		`    return k`

Exemple d'exécution de cette fonction `seuil`

Fixons la valeur de A à 50. On saisit `seuil(50)` et la valeur renvoyée est 4.

Détaillons l'exécution pas à pas.

Au début (lignes 2 et 3), k et u prennent la valeur 0.

La boucle « Tant que » des lignes 4, 5, 6, et 7 indique que les instructions :
$k \leftarrow k + 1$
$u \leftarrow k^3 + k^2$
sont répétées tant que $u < 50$, et s'arrête donc dès que $u \geqslant 50$.

Après cela la valeur de k est renvoyée (`return k`).

Détaillons tout cela :

Début :	$k \leftarrow 0$	$u \leftarrow 0$	
Boucle Tant que	$k \leftarrow 0 + 1 = 1$	$u \leftarrow 1^3 + 1^2 = 2 < 50$	
	$k \leftarrow 1 + 1 = 2$	$u \leftarrow 2^3 + 2^2 = 12 < 50$	
	$k \leftarrow 2 + 1 = 3$	$u \leftarrow 3^3 + 3^2 = 36 < 50$	
	$k \leftarrow 3 + 1 = 4$	$u \leftarrow 4^3 + 4^2 = 80 \geqslant 50$	arrêt de la boucle Tant que
Fin :	$k = 4$		

À retenir

Les lignes 4 à 7 de l'algorithme ci-dessus constitue une **boucle non bornée** ou **boucle Tant que**.

Tant que (*condition*)
    faire ceci
    faire cela
Fin du Tant que

La boucle Tant que continue à s'exécuter tant que la condition est réalisée et par conséquent s'arrête dès que la condition n'est plus réalisée, par exemple, ci-dessus, la boucle Tant que continue à s'exécuter tant que $u < A$ et s'arrête dès que $u \geqslant A$.

9 Algorithmes à connaître

a) Calculer une somme avec un algorithme :

Une suite est définie sous la forme $u_n = f(n)$ et on veut calculer avec un algorithme la somme : $S = u_1 + u_2 + \cdots + u_n$.
On commence par affecter 0 à la variable S, puis on utilise une boucle Pour qui, à chaque étape :

- calcule u_k pour chaque valeur de k
- ajoute à chaque fois la valeur de u_k dans S.

Cela donne l'algorithme ci-dessous, où les variables n, k sont des entiers naturels, et les variables u, S des réels.

Algorithme de calcul d'une somme
$S \leftarrow 0$
Pour $k = 1$ jusqu'à $k = n$
$\quad u \leftarrow f(k)$
$\quad S \leftarrow S + u$
Fin du Pour

b) Suite croissante à limite infinie (programme de TS)

Le programme de TS précise :
dans le cas d'une limite infinie, étant donné une suite croissante (u_n) et un nombre réel A, déterminer à l'aide d'un algorithme un rang n à partir duquel $u_n \geqslant A$.
Pour une suite (u_n) définie par $u_n = f(n)$, l'algorithme ci-dessous détermine **le rang n à partir duquel $u_n \geqslant A$.**
La variable k est un entier naturel, les variables u et A sont des réels.

Algorithme
$k \leftarrow 0$
$u \leftarrow 0$
Tant que $u < A$
$\quad k \leftarrow k + 1$
$\quad u \leftarrow f(k)$
Fin du Tant que

c) Le compteur

Prenons un exemple : on choisit au hasard un nombre entre 1 et 100, et on vérifie si ce nombre est inférieur ou égal à 10.
On répète n fois cette épreuve aléatoire, et on compte le nombre d'apparitions d'un nombre inférieur ou égal à 10.
On appelle c la variable « compteur ».
Au départ c prend la valeur zéro, puis chaque fois qu'apparaît un nombre inférieur ou égal à 10, on augmente le compteur de 1, ce qui se traduit par : $c \leftarrow c + 1$.
On obtient ainsi l'algorithme ci-contre.

Algorithme
$c \leftarrow 0$
Pour $k = 1$ jusqu'à $k = n$
$\quad a \leftarrow$ nombre aléatoire entre 1 et 100
$\quad$ Si $a \leqslant 10$
$\quad\quad$ alors $c \leftarrow c + 1$
Fin du Pour

À retenir – Le principe du compteur

- Au départ la variable « compteur » prend la valeur zéro ;
- puis chaque fois qu'une condition est respectée on augmente le compteur de 1.

On obtient ainsi un algorithme du type ci-dessous.

Algorithme
$c \leftarrow 0$
Pour $k = 1$ jusqu'à $k = n$
instruction
Si *condition*
alors $c \leftarrow c + 1$
Fin du Pour

d) Une boucle Pour dans une boucle Pour

Exemple

	Algorithme
1	On crée une liste `ListSol` qui est vide
2	Pour $x = 1$ jusqu'à $x = n$
3	Pour $y = 1$ jusqu'à $y = p$
4	Si $3x + 5y - 19 = 0$
5	alors ajouter $(x\ ;\ y)$ à la liste `ListSol`
6	Fin du Pour
7	Fin du Pour

Exécution de cet algorithme pour $n = 4$ et $p = 3$

Au début, la liste `ListSol` est vide

$x = 1$

$y = 1$ $\quad 3x + 5y - 19 = 3 \times 1 + 5 \times 1 - 19 = -11 \neq 0$

$y = 2$ $\quad 3x + 5y - 19 = 3 \times 1 + 5 \times 2 - 19 = -6 \neq 0$

$y = 3$ $\quad 3x + 5y - 19 = 3 \times 1 + 5 \times 3 - 19 = -1 \neq 0$

$x = 2$

$y = 1$ $\quad 3x + 5y - 19 = 3 \times 2 + 5 \times 1 - 19 = -8 \neq 0$

$y = 2$ $\quad 3x + 5y - 19 = 3 \times 2 + 5 \times 2 - 19 = -3 \neq 0$

$y = 3$ $\quad 3x + 5y - 19 = 3 \times 2 + 5 \times 3 - 19 = 2$

$x = 3$

$y = 1$ $\quad 3x + 5y - 19 = 3 \times 3 + 5 \times 1 - 19 = -5 \neq 0$

$y = 2$ $\quad 3x + 5y - 19 = 3 \times 3 + 5 \times 2 - 19 = 0$ On ajoute $(3\ ;\ 2)$ à la liste `ListSol`

$y = 3$ $\quad 3x + 5y - 19 = 3 \times 3 + 5 \times 3 - 19 = 5 \neq 0$

$x = 4$

$y = 1$ $\quad 3x + 5y - 19 = 3 \times 4 + 5 \times 1 - 19 = -2 \neq 0$

$y = 2$ $\quad 3x + 5y - 19 = 3 \times 4 + 5 \times 2 - 19 = 3 \neq 0$

$y = 3$ $\quad 3x + 5y - 19 = 3 \times 4 + 5 \times 3 - 19 = 8 \neq 0$

À la fin, la liste `ListSol` contient le couple $(3\ ;\ 2)$.

Que fait cet algorithme ?

Au début, la liste `ListSol` est vide

Ensuite, pour tous les entiers x et y tels que $1 \leqslant x \leqslant n$ et $1 \leqslant y \leqslant p$, il vérifie si $3x + 5y - 19$ est égal à zéro ou non.

Chaque fois que $3x + 5y - 19$ est égal à zéro il ajoute $(x\ ;\ y)$ à la liste des solutions.

Que contient la liste `ListSol` à la fin ?

À la fin, la liste `ListSol` contient tous les couples $(x\ ;\ y)$ solutions de l'équation $3x + 5y - 19 = 0$ pour tous les entiers x et y tels que $1 \leqslant x \leqslant n$ et $1 \leqslant y \leqslant p$.

À retenir – Une boucle Pour dans une boucle Pour

Dans l'algorithme ci-dessous,

- La boucle *Pour* $x = 1$ *jusqu'à* $x = n$ va de la ligne 1 à la ligne 5,
  et elle **contient** la boucle *Pour* $y = 1$ *jusqu'à* $y = p$ qui va de la ligne 2 à la ligne 4.
- On a donc dans l'ordre :
  - $x = 1$, puis y prend les valeurs de $y = 1$ jusqu'à $y = p$;
  - $x = 2$, puis y prend les valeurs de $y = 1$ jusqu'à $y = p$;
  - et ainsi de suite ...

	Algorithme
1	Pour $x = 1$ jusqu'à $x = n$
2	Pour $y = 1$ jusqu'à $y = p$
3	*Instruction*
4	Fin du Pour
5	Fin du Pour