Exercice 1 Différence entre `if` et `while`

On considère les deux programmes suivants :

```
>>> a = 7.5
>>> if a > 3 :
...     a = a-1
```

```
>>> a = 7.5
>>> while a > 3 :
...     a = a-1
```

1) Dans la colonne de gauche, quelle est la valeur de `a` à la fin du programme ? Tracer le graphe de flux du programme de gauche.

2) Tacer le graphe de flux du programme de droite et simuler son exécution "à la main" en détaillant chaque passage dans la boucle avec la valeur correspondante de `a`. Donner la valeur de `a` à la fin de ce programme.

Exercice 2

On suppose qu'une variable réelle x a déjà été déclarée. Écrire un programme qui calcule

$$\begin{cases} 2x+1 & \text{si } x \geqslant 4 \\ 2x & \text{si } x \in [2,4[\\ 3x-2 & \text{sinon} \end{cases}$$

Exercice 3

La commande `print ('Bonjour')` permet d'afficher à l'écran le message `Bonjour`. Le message doit être délimité par des apostrophes.
Écrire un programme qui affiche en fonction d'une variable `n` précédemment définie l'un des messages suivants :

```
"Ce nombre est pair"
```

```
"Ce nombre n'est pas pair mais est multiple de 3"
```

```
"Ce nombre n'est ni pair ni multiple de 3"
```

Exercice 4

Programme mystère : préciser en fonction de l'entier `n` la valeur de `y` après l'exécution du programme suivant. On précisera les différents cas.

```
if n >= 3:
    y = n*n
    if n <= 4 :
        y = y-1
        y = 2*y
    else :
        y += 1
else :
    if n % 2 == 0 :
        y = 3*n
    else :
        y = n-5
    y = 3*y
```

Exercice 5 Importance de l'indentation

On suppose qu'une variable `a` a été préalablement initialisée.
Expliquer la différence entre les deux programmes suivants et donner une valeur de `a` pour laquelle les deux programmes donneront en fin d'exécution une valeur différente à `b` :

```
b = 1
if a > 3 :
    b = 2 * a
b = b + 1
```

```
b = 1
if a > 3 :
    b = 2 * a
    b = b + 1
```

Exercice 6

On définit une suite (u_n) par $u_0 = 1$ et $\forall n \in \mathbb{N}, u_{n+1} = 2u_n + 1$.

1) En supposant qu'on a déjà initialisé une variable `n` avec une valeur entière positive, écrire une boucle `while` qui permet de calculer u_n.

2) On pourrait démontrer (mais on l'admettra ici) que la suite (u_n) diverge en croissant vers $+\infty$.

   En supposant qu'on a déjà initialisé une variable `d` avec une valeur réelle, écrire une boucle `while` qui permet de calculer le plus petit entier naturel n tel que $u_n > d$.

Exercice 7 Programme mystère

On suppose qu'on a déjà initialisé deux variables `a` et `b` avec des valeurs entières strictement positives.
On considère le programme :

```
while a >= b :
    a -= b
```

1) Dans le cas particulier où la variable `a` a été initialisée avec la valeur 14 et la variable `b` avec la valeur 3, dérouler "à la main" l'exécution de ce programme en précisant à chaque passage dans la boucle la valeur de `a` et de `b`. Donner la valeur finale de `a`.

2) Revenons au cas général. Démontrer l'invariant de boucle : "soient a_i la valeur initiale de `a`, a_c sa valeur courante et b la valeur de `b`, alors $a_i \equiv a_c \mod b$".

3) En fin de programme, comment spécifier mathématiquement la valeur finale de `a` en fonction de a_i et de b ?

Exercice 8 De l'importance des inégalités strictes et larges

Le savant Sunisoc a programmé le programme donné dans le cours pour savoir si `n` est premier ou non. Mais il a apporté une légère modification.

```
k = 2
if n < 2 : resultat = False
else :
    while k * k < n and n % k != 0 :
        k = k+1
    resultat = (k * k >= n)
```

Montrer que son programme peut aboutir à un résultat erroné.

Exercice 9 Une erreur classique

Le savant Sunisoc souhaite calculer le 4–ième terme de la suite de Fibonacci (F_n). Il pense écrire un programme équivalent à celui donné dans le cours :

```
n = 4
a = 0
b = 1
while n > 0 :
    a = b
    b = a+b
    n = n-1
a
```

Surprise : il obtient comme valeur de retour 8, alors que $F_4 = 3$.
Dérouler "à la main" l'exécution de son programme, en détaillant à chaque passage dans la boucle la valeur de la variable `a` et celle de `b`. Expliquer l'erreur conceptuelle de son programme.

Expliquer pourquoi s'il remplace
```
a = b
b = a+b
n = n-1
```
par
```
c = b
b = a+b
a = c
n = n-1
```
son programme sera correct.

Exercice 10 Punition

L'instruction `print "Je dois ranger ma chambre"` provoque l'affichage du message :

```
Je dois ranger ma chambre
```

Écrire un programme avec une boucle qui affiche 50 fois le message `Je dois ranger ma chambre`.

Exercice 11 Chanson traditionnelle bretonne

La séquence d'instructions :

```
n = 10
print "C'est dans %i ans je m'en irai j'entends le loup le renard chanter" % n
```

permet d'afficher le message :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
```

Écrire une boucle `while` qui permet d'afficher :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
C'est dans 9 ans je m'en irai j'entends le loup le renard chanter
C'est dans 8 ans je m'en irai j'entends le loup le renard chanter
...
C'est dans 1 ans je m'en irai j'entends le loup le renard chanter
```

On ne s'occupera pas de la faute d'orthographe de la dernière ligne.

Exercice 12 Importance de l'imbrication

On suppose qu'une variable `n` a été préalablement initialisée avec une valeur entière supérieure ou égale à 2.
On propose deux programmes possibles pour calculer le nombre de diviseurs positifs de `n`, ce nombre de diviseurs étant stocké dans la variable `p` en fin de programme :

```
p = 0
k = 1
while k <= n :
    if n % k == 0 :
        p = p+1
    k = k+1
```

```
p = 0
k = 1
if n % k == 0 :
    while k <= n :
        p = p+1
        k = k+1
```

Lequel de ces deux programmes calcule bien le résultat souhaité ? Que calcule l'autre programme ?

Exercice 13 Tables de multiplications

La séquence d'instructions :

```
i=3
j=4
print i, 'x', j, '=', i*j
```

permet d'afficher le message :

```
3 x 4 = 12
```

Écrire un programme permettant d'afficher toutes les tables de multiplications, depuis `1 x 1 = 1` jusqu'à `10 x 10 = 100`. Il y a 100 messages à afficher.
Indication : le plus simple est probablement d'imbriquer deux boucles `while` l'un à l'intérieur de l'autre.

Exercice 14 Exponentiation rapide, version itérative

Fixons une valeur a réelle, et une valeur $b \in \mathbb{N}^*$.
Le but de cet exercice est de calculer $a^b = \underbrace{a \times a \times \cdots \times a}_{b \text{ fois}}$ en utilisant uniquement des multiplications.

1) Algorithme naïf

   Traduisez en Python le pseudo–code suivant :

   Valeurs d'entrée : $a \in \mathbb{R}$, $b \in \mathbb{N}^*$

   $m \leftarrow 1$

   Tant que $b > 0$ **Faire**

   $m \leftarrow m * a$

   $b \leftarrow b - 1$

   Fin Tant que

   Valeur de retour : m

   Prouvez l'invariant de boucle : "si on note b_i la valeur initiale et b_c la valeur courante de b, alors $a^{b_i} = m.a^{b_c}$".

   En déduire que l'algorithme naïf calcule bien a^{b_i}.

   Préciser en fonction de b_i le nombre de multiplications effectuées.

Dans la suite, nous allons calculer a^b avec l'algorithme suivant (où E désigne la partie entière) :
Valeurs d'entrée : $a \in \mathbb{R}$, $b \in \mathbb{N}^*$
$m \leftarrow 1$
Tant que $b > 1$ **Faire**
Si b impair **alors** $m \leftarrow m \times a$ **Fin Si**
$a \leftarrow a \times a$
$b \leftarrow E(b/2)$
Fin Tant que
Valeur de retour : $m \times a$

2) Dérouler "à la main" l'exécution de cet algorithme lorsque l'on initialise b avec $b \leftarrow 49$ et vérifier que la valeur de retour est bien a^{49}. Combien de multiplications a–t–on effectuées ?

3) Programmer cet algorithme en Python.

4) Prouver l'invariant de boucle : "en notant a_i (resp. b_i) la valeur initiale et a_c (resp. b_c) la valeur courante de a (resp. b), alors $a_i^{b_i} = m.a_c^{b_c}$". En déduire que cet algorithme calcule bien $a_i^{b_i}$.

5) Notons $M(b)$ le nombre de multiplications effectuées par cet algorithme en fonction de b.

   Montrer que $M(b) \leqslant 2 + M(E(b/2))$ (où E désigne la partie entière).

   En considérant p le plus petit entier tel que $p \geqslant \log_2(b)$, montrer que $M(b) \leqslant 2p + 1$ et en déduire que $M(b) \leqslant 2\log_2(b) + 3$, ce qui est beaucoup plus intéressant que l'algorithme naïf lorsque b est "grand".