Corrigé

Instructions itératives

Boucles énumérées

Exercice 1.
La première version de la fonction `entiers` peut être définie ainsi :

```
def entiers(i, j):
    for k in range(i, j):
        print(k, end='-')
    print(j)
```

Pour déterminer si un entier est divisible par 7, on calcule son reste modulo 7 :

```
def entiers(i, j):
    if j % 7 == 0:
        j -= 1
    for k in range(i, j):
        if k % 7 != 0:
            print(k, end='-')
    print(j)
```

Exercice 2.
Les quatre fonctions demandées se définissent par exemple ainsi :

```
def triangle1(n):
    for k in range(1, n+1):
        print('*'*k)
```

```
def triangle2(n):
    for k in range(n, 0, -1):
        print('*'*k)
```

```
def pyramide1(n):
    triangle1(n)
    triangle2(n-1)
```

```
def pyramide2(n):
    for k in range(1, n+1):
        print(' '*(n-k), '* '*k, sep='')
```

Exercice 3.
Les fonctions qui permettent de générer les quatre tables présentées peuvent se définir ainsi :

```
def talkhys1():
    n = 0
    for k in range(9):
        n = 10 * n + 1
        print('{0:>9} x {0:<9} = {1:^17}'.format(n, n*n))
```

```
def talkhys2():
    n = 0
    for k in range(1, 10):
        n = 10 * n + k
        print('8 x {:<9} + {} = {:<9}'.format(n, k, 8*n+k))
```

```
def talkhys3():
    n = 0
    for k in range(1, 10):
        n = 10 * n + k
        print('9 x {:<9} + {:>2} = {:<10}'.format(n, k+1, 9*n+k+1))
```

```
def talkhys4():
    n = 0
    for k in range(7, -1, -1):
        n = 10 * n + k + 2
        print('9 x {:<8} + {} = {:<9}'.format(n, k, 9*n+k))
```

Exercice 4.
On définit successivement :

```
def heure_to_sec(h, m, s):
    return h * 3600 + m * 60 + s
```

```
def sec_to_heure(s):
    m, s = s // 60, s % 60
    h, m = m // 60, m % 60
    print('{:>2}:{:>2}:{:>2}'.format(h, m, s))
```

```
def duree(h1, m1, s1, h2, m2, s2):
    s = heure_to_sec(h2, m2, s2) - heure_to_sec(h1, m1, s1)
    sec_to_heure(s)
```

Exercice 5.
Si un caractère appartient à la chaîne `alph`, son index k permet de calculer son codage : c'est la lettre d'index $(k + 13) \bmod 26$.

```
def rot13(s):
    alph = 'abcdefghijklmnopqrstuvwxyz'
    t = ''
    for c in s:
        if c in alph:
            k = alph.index(c)
            t += alph[(k+13) % 26]
        else:
            t += c
    return t
```

Cette fonction nous permet de connaître la réponse à l'énigme posée :

```
In [1]: rot13("har crefbaar abeznyr crafr dh'ha xvyb-bpgrg rfg étny à 1000 bpgrgf, ha
        vasbezngvpvra rfg pbainvaph dh'ha xvybzèger rfg étny à 1024 zègerf.")

Out[1]:"une personne normale pense qu'un kilo-octet est égal à 1000 octets, un informaticien
        est convaincu qu'un kilomètre est égal à 1024 mètres."
```

Boucles conditionnelles

Exercice 6.
Une première démarche consiste à itérer p jusqu'à trouver un carré qui dépasse n :

```
def isqrt(n):
    p = 0
    while (p+1)*(p+1) <= n:
        p += 1
    return p
```

Si on ne s'autorise que des additions, il faut observer que $(p+1)^2 = p^2 + 2p + 1$; autrement dit, on peut calculer le carré suivant à partir du précédent en ajoutant $2p + 1$. Ceci conduit à la version suivante :

```
def isqrt(n):
    p, c = 0, 1
    while c <= n:
        p += 1
        c += p+p+1
    return p
```

Exercice 7.
Il suffit de calculer les puissances de 2 successives, les convertir en chaînes de caractères, puis déterminer si la date recherchés s'y trouve. Cette date sera introduite en paramètre de la fonction directement sous forme de chaîne de caractère de manière à gérer convenablement les jours de naissance débutant par un 0.

```
def recherche(m):
    if type(m) != str:
        m = str(m)
    n = 0
    x = 1
    while not m in str(x):
        n += 1
        x *= 2
    print(n, x)
```

Cette fonction affiche l'entier n et la valeur de 2^n correspondante.

```
In [1]: recherche("090712")   # date de naissance de Champernowne
2231
396205341587359809071201960508281034633001234024671700517209433879099732138957955180125254618
598508416974054152536692099314093967249376650832906940867725494706212246635191331892246242982
580293911065600056593786718320880090753316789515706244357732722400544210231905358445910562340
916116286229592117977677380999735068008335378515121153286811076631762748819111499033174373018
152918040459530123666820773609637573714011383498294888100647448720820157033291592429215077500
117664196054818061553319120280760325129186038284332809825664447726718475106331994860932207451
664122117089765442229417456964138638342181972499473377303626672450292905529945961911445207008
052819661294716059648
```

Exercice 8.
Le critère de primalité proposé peut se définir ainsi :

```
def premier(p):
    if p < 2:
        return False
    k = 2
    while k * k <= p:
        if p % k == 0:
            return False
        k += 1
    return True
```

(Rappelons que l'instruction `return` interrompt immédiatement la fonction pour retourner le résultat indiqué.)
Les mille plus petits nombres premiers s'obtiennent alors en exécutant le script :

```
p, n = 0, 0
while n < 1000:
    while not premier(p):
        p += 1
    print(p, end=' ')
    n += 1
    p += 1
```

Pour vérifier la conjecture de Goldbach, on commence par définir une fonction qui recherche un nombre premier p tel que $n - p$ soit premier :

```
def goldbach(n):
    for p in range(2, n-1):
        if premier(p) and premier(n-p):
            return True
    return False
```

Le script permettant de vérifier la conjecture jusqu'à 10 000 s'écrit alors :

```
  for n in range(4, 10001, 2):
      if not goldbach(n):
          print("Conjecture mise en défaut pour {}".format(n))
          break
  if n == 10000:
      print("Conjecture vérifiée")
```

On suit la même démarche pour étudier la conjecture suivante :

```
def conjecture(n):
    k = 1
    while k < n:
        if premier(n-k):
            return True
        k *= 2
    return False
```

sauf que cette fois le script suivant montre que la conjecture est mise en défaut pour $n = 127$.

```
  for n in range(3, 10001, 2):
      if not conjecture(n):
          print("Conjecture mise en défaut pour {}".format(n))
          break
  if n == 10000:
      print("Conjecture vérifiée")
```

Exercice 9.
On définit la fonction :

```
def lookandsay(s):
    i = 0
    t = ''
    while i < len(s):
        k = 1
        j = i + 1
        while j < len(s) and s[j] == s[i]:
            k += 1
            j += 1
        t = t + str(k) + s[i]
        i = j
    return t
```

Le script suivant affiche les 20 premiers termes de la suite de Conway :

```
s = '1'
print(s)
for _ in range(19):
    s = lookandsay(s)
    print(s)
```

On obtient une valeur approchée de la constante de Conway en l'approchant par $\frac{u_{n+1}}{u_n}$ pour une valeur de n assez grande, quotient que l'on calcule à l'aide de la fonction :

```
def eval(s, n):
    u, v = s, lookandsay(s)
    for _ in range(n):
        u, v = v, lookandsay(v)
    return len(v) / len(u)
```

Cependant, la croissance exponentielle du temps de calcul de u_n nous force à ne pas dépasser $n = 40$:

```
In [1]: eval('1', 40)
Out[1]: 1.3031208257437765
```

On peut néanmoins constater que cette constante est indépendante de la valeur initiale choisie et vaut environ $1,30$:

```
In [2]: eval('66', 40)
Out[2]: 1.3052921299324176

In [3]: eval('11', 40)
Out[3]: 1.3044580289250036
```

Enfin, montrons que seuls les chiffres 1, 2 et 3 peuvent apparaître dans la suite de Conway.
Supposons par exemple que le chiffre 4 apparaisse pour la première fois à l'étape p. Puisqu'il n'est pas présent à l'étape précédente, c'est qu'il existe à l'étape $p-1$ un motif de la forme $xxxx$, où x est un certain chiffre.
Ce motif ne peut être apparu sous la forme $(xx)(xx)$; pour cela il faudrait que l'étape précédente contienne $(\underbrace{xx\cdots x}_{x \text{ fois}})(\underbrace{xx\cdots x}_{x \text{ fois}})$, mais ce motif serait lu « $(2x)$ fois x » et non pas « x fois x, x fois x ».
Et il ne peut pas non plus être apparu sous la forme $(yx)(xx)(xz)$ car pour cela l'étape précédente devrait contenir le motif $(\underbrace{xx\cdots x}_{y \text{ fois}})(\underbrace{xx\cdots x}_{x \text{ fois}})(\underbrace{xx\cdots z}_{x \text{ fois}})$ qui serait lu « $(x+y)$ fois x, x fois z ».
Le même raisonnement s'applique à tout entier $n \geqslant 4$.

Exercice 10.
Pour convertir un entier en numérotation Shadok, on définit la fonction :

```
def shadok(n):
    c = ['ga', 'bu', 'zo', 'meu']
    s = c[n % 4]
    n //= 4
    while n > 0:
        s = c[n % 4] + '-' + s
        n = n // 4
    return s
```

La fonction réciproque utilise la méthode `split` qui, appliquée à une chaîne de caractères, retourne la liste des sous-chaînes que séparent l'argument `sep` de cette méthode.

```
def kodahs(s):
    t = s.split(sep='-')
    c = ['ga', 'bu', 'zo', 'meu']
    n = 0
    for x in t:
        n = n * 4 + c.index(x)
    return n
```

Par exemple :

```
In [1]: shadok(1968)
Out[1]: 'bu-meu-zo-meu-ga-ga'

In [2]: kodahs('bu-meu-meu-bu-meu-meu')
Out[2]: 2015
```