Informatique PCSI
Corrigé suite TP 4

Exercice 1

La valeur initiale de `p` est 1, la valeur initiale de `n` est 9, donc on entre dans la boucle.

1er passage : `n` est impair donc `p` prend la valeur 2, `x` prend la valeur 4, `n` prend la valeur 4.

2e passage : `n` est pair donc `p` garde la valeur 2, `x` prend la valeur 16, `n` prend la valeur 2.

3e passage : `n` est pair donc `p` garde la valeur 2, `x` prend la valeur 256, `n` prend la valeur 1.

4e passage : `n` est impair donc `p` prend la valeur 512, `x` prend la valeur 65536, et enfin `n` prend la valeur 0, c'est la sortie de boucle.

La valeur de `p` ($512 = 2^9$) est renvoyée.

Exercice 2

Il s'agit de l'algorithme de multiplication à la russe. On n'utilise que des additions et des divisions par 2.

Par exemple : $5 \times 8 = 8 + 4 \times 8 = 8 + 2 \times 16 = 8 + 1 \times 32 = 40$.

```
def mult_russe(x, n):
    s = 0
    while n != 0:
        if n % 2:
            s = s + x
        x =  x + x
        n = n // 2
    return s
```

Un autre exemple : 16×7 avec 16 additions ou alors avec 4 divisions par 2 et 4 multiplications par 2 : $16 \times 7 = 8 \times 14 = 4 \times 28 = 2 \times 56 = 1 \times 112$.

Avec l'exponentiation rapide, on calcule $x \times x \times \ldots \times x$ avec n facteurs. Pour la multiplication à la russe, on calcule $x + x + \ldots + x$ avec n termes. C'est le même algorithme où on remplace l'opérateur $\times$ par l'opérateur $+$.

Exercice 3

La valeur initiale de `s` est 0, la valeur initiale de `n` est 7, donc on entre dans la boucle.

1er passage : `n` est impair donc `s` prend la valeur 3, `x` prend la valeur 6, `n` prend la valeur 3.

2e passage : `n` est impair donc `s` prend la valeur 9, `x` prend la valeur 12, `n` prend la valeur 1.

3e passage : `n` est impair donc `s` prend la valeur 21, `x` prend la valeur 24, `n` prend la valeur 0 et c'est la sortie de boucle.

La valeur de `s` ($21 = 3 \times 7$) est renvoyée.

Exercice 4

1. On appelle la fonction `loga2`, pour logarithme approché en base 2.

```
def loga2(n):
    s = 0
    while n != 1:  # ou n > 1 (n > 2 - 1)
        s = s + 1
        n = n // 2
    return s, s+1
```

2. Les valeurs successives de n constituent une suite strictement décroissante d'entiers naturels qui converge donc vers 1 en un nombre fini d'étapes. Ceci prouve la terminaison.

L'expression `n` constitue un variant de boucle.

Il existe un entier p tel que la propriété $2^p \leq 2^s \times n < 2^{p+1}$ est vraie après chaque passage dans la boucle. (p est la partie entière de $\log_2(n)$). Soit $s1$ et $n1$ les valeurs de `s` et `n` avant un passage dans la boucle, $s2$ et $n2$ les valeurs de `s` après le passage. Si $2^p \leq 2^{s1} \times n1 < 2^{p+1}$, alors $2^{p-s1} \leq n1 < 2^{p+1-s1}$. Donc $2^{p-s1-1} \leq n1/2 < 2^{p+1-s1-1}$, d'où $2^{p-s2} \leq n2 < 2^{p+1-s2}$, donc $2^p \leq 2^{s2} \times n2 < 2^{p+1}$. À la fin de la boucle, la valeur de `n` est 1, donc $2^p \leq 2^s < 2^{p+1}$ et $s = p$.

3. Sur le même modèle que `loga2`, la fonction `loga10`.

```
def loga10(n):
    s = 0
    while n > 9:  # n > 10 - 1
        s = s + 1
        n = n // 10
    return s, s+1
```