Exercice 1

Il faut déterminer le plus petit entier $n \in \mathbb{N}$ tel que $10^{n+1} > x$, c'est-à-dire le premier entier pour lequel cette condition est vérifiée. Pour cela, on va utiliser une boucle TANT QUE.

1) Voici l'algorithme en français :

$n = 0$
TANT QUE $10^{n+1} \leqslant x$
$\quad n = n + 1$
Afficher n

2) En `Python`, cela se traduit simplement :

```
n=0
while 10**(n+1) <= x:
    n=n+1
print(n)
```

Exercice 2

1) (a) Comme $x = 10 > 1$, on effectue le bloc d'instructions après ALORS.
On incrémente un entier N jusqu'à ce que $10^N \geqslant 10^6$, puis on l'affiche.
Autrement dit, on renvoie le premier entier N tel que $10^N \geqslant 10^6$.

Ainsi, L'algorithme affiche 6.

(b) Là, $x \leqslant 1$ donc on effectue le bloc d'instructions après SINON.

Ainsi, L'algorithme affiche `"x doit dépasser 1"`.

(c) Comme $x = 20 > 1$, on effectue le bloc d'instructions après ALORS comme dans le premier cas, qui renvoie le premier entier N tel que $20^N \geqslant 10^6$, soit N = 5.

Ainsi, L'algorithme affiche 5.

2) Cet algorithme affiche le plus petit (le premier) entier naturel $N \in \mathbb{N}$ tel que $x^N \geqslant 10^6$ si $x > 1$, et renvoie le message `"x doit dépasser 1"` sinon.

Il permet tout simplement de résoudre l'inéquation $x^N \geqslant 10^6$.

3) Voilà le travail :

```
x=eval(input("Donnez un réel x>1 : "))
if x>1:
    N=0
    while x**N<10**6:
        N=N+1
    print(N)
else:
    print("x doit dépasser 1")
```

Exercice 3

Le plus simple est de commencer par attribuer une valeur négative ou nulle à la variable dans laquelle on va stocker le nombre testé. Cela permet d'éviter de dupliquer les demandes et les tests. Ensuite, on utilise une boucle `while` afin de réitérer les demandes.

```
from math import log
x=0
while x<=0:
    x=eval(input("Entrez un réel strictement positif : "))
print(log(x))
```

Exercice 4

1) Il manque une parenthèse fermante à la fin de la première ligne.

   De plus, la variable `total` n'a pas été créée (via une affectation) avant d'être utilisée la première fois dans le calcul à l'avant-dernière ligne.
   Pour y remédier, il faut affecter la variable `total` avant l'entrée dans la boucle.

```
n=int(input())
total=0
for i in range(n):
    if i%2==0:
        carre=i**2
    else:
        carre=0
    total=total+carre
print(total)
```

2) Ce programme demande un entier n à l'utilisateur, puis il affiche la somme des carrés des entiers pairs compris entre 0 et $n-1$ (donc 0 si $n < 1$).

Exercice 5

1) Voici l'état des variables c et p au cours de l'exécution de la fonction `f` :

c	3	2	1	0
p	1	2	4	8

   Quand on tape `f(3)`, on obtient donc 8.

2) Si l'on tape `f(-4)`, la variable `c` va d'abord prendre la valeur -4 puis diminuer de 1 à chaque itération. Comme elle reste ainsi strictement négative, elle n'atteindra jamais la valeur 0 si bien que la boucle `while` ne s'arrête pas.

   On obtient donc une boucle infinie lorsque l'on tape `f(-4)` ... avec un carré rouge dans la console, et si tout se passe bien un blocage de la machine qui nous oblige à redémarrer le noyau !

3) Il y a plusieurs façons de procéder. En voici deux, qui utilisent le fait que si $n < 0$,

$$2^n = \frac{1}{2^{-n}} \qquad \text{et} \qquad 2^n = \left(\frac{1}{2}\right)^{-n}$$

❶ La fonction `f` de l'énoncé fait très bien le travail pour les entiers positifs. On l'utilise pour définir une nouvelle fonction qui, elle, sait aussi gérer les entiers négatifs.

```
def g(n):
    if n>=0:
        return f(n)
    else:
        return 1/f(-n)
```

❷ On modifie `f` afin de calculer des puissances de 1/2 lorsque n est négatif.

```
def f(n):
    p=1
    if n>=0:
        c=n
        x=2
    else:
        c=-n
        x=1/2
    while c!=0:
        p=p*x
        c=c-1
    return p
```

Remarque : une variante consiste, lorsque $n < 0$, à augmenter c de 1 dans la boucle et à diviser p par 2 à chaque étape.

```
def f(n):
    p=1
    c=n
    if c>=0:
        while c!=0:
            p=p*2
            c=c-1
    else:
        while c!=0:
            p=p/2
            c=c+1
    return p
```