1. Construction de matrices

Il y a beaucoup de solutions possibles, par exemple:

```
  np.diag(np.arange(7, dtype=np.float))[1:, 2:]
  np.diag(np.arange(2,7, dtype=np.float), -1)
  np.concatenate( (np.zeros((5,))[np.newaxis, :]  , np.diag(np.arange(2,7)) )  )
```

On peut aussi procéder en remplaçant des éléments:

```
  a = np.zeros((6, 5))
  a[1:, :] = np.diag(np.arange(2, 7))
```

Une autre approche, sans np.diag():

```
  a = np.zeros((30,))
  a[5::6] = np.arange(2, 7)
  a.shape = (6,5)
```

2. Elements positifs

Avec

```
   x = np.range(10)-5
```

les expressions suivantes donnent la bonne réponse:

```
   np.repeat(x, x>0)
   x.repeat(x>0)
   np.take(x, np.where(x > 0))
   x.take(np.where(x > 0))
```

Pour ceux qui aiment bien vivre dangéreusement, il y a d'autres options:

```
   x[x>0]
   x[np.where(x>0)]
```

Dans le premier cas, l'indéxation est équivalente à x.repeat(x>0), dans le deuxième cas à x.take (np.where(x > 0)). Les règles précises pour savoir quelle opération est appliquée sont assez complexes.

3. Table de multiplication

Avec

```
  a = np.arange(3)
  b = np.array([-1., 1., 2.])
```

on obtient le résultat recherché par

```
  a[:, np.newaxis]*b[np.newaxis, :]
```

ou

```
  np.outer(a, b)
```

L'intérêt de la première version est qu'elle peut être généralisé à d'autres opération que la multiplication.

4. Différences

Avec

```
   x = np.array([1., 2., -3., 0.])
```

la solution la plus compacte est

```
x[1:]-x[:-1]
```