Les fonctions définies par `def`

Evidemment, même si la fonction peut s'écrire sur une seule ligne, il est possible d'utiliser `def` plutôt qu'une λ-fonction. Nous reprenons donc les exemples précédents avec cette nouvelle syntaxe.

- A nouveau l'indentation est synthaxique donc **obligatoire**.
- La fonction peut prendre autant d'arguments que nécessaire (cela peut être 0 ou bien n'importe quel entier aussi grand que voulu - il n'y a plus de limites depuis python 3.6).
- Le mot-clef `return` permet d'indiquer le (les) résultat(s) que renvoie la fonction. Elle s'arrête aussitôt après, même s'il reste des instructions. La fonction peut n'avoir aucun `return`, c'est-à-dire ne renvoyer aucune valeur : on peut dans ce cas l'appeler procédure.

In [18]:
```
def carre(x):
    return x**2
x = 2
print(f"Evaluation de la fonction carre sur x={x} : {carre(x)}")
```

```
Evaluation de la fonction carre sur x=2 : 4
```

In [20]:
```
def echange(x, y):
    return y, x
x, y = 1, 2
z, t = echange(x, y)
print(f"On échange {x} et {y} : {z}, {t}")
```

```
On échange 1 et 2 : 2, 1
```

In [21]:
```
def inverse(x):
    if x == 0:
        return None
    return 1./x
x, y = 2, 0
print(f"1/{x} = {inverse(x)}")
print(f"1/{y} = {inverse(y)}")
```

```
1/2 = 0.5
1/0 = None
```

Notez dans le dernier exemple ci-dessous qu'il y a deux `return`. Si le test `x == 0` est vrai alors le premier `return` est utilisé : la fonction retourne `None` et s'arrête. Si le test est faux alors la fonction descend au deuxième `return`.

Il est également possible d'utiliser les fonctions de manière récursive. Par exemple pour calculer $n!$.

In [25]:
```
def factorielle(n):
    if n != int(n):
        print(f"{n} n'est pas un entier !")
        return
    if n <= 0:
        print(f"{n} est négatif !")
        return
    if n == 1:
        return 1
    return n * factorielle(n-1)

print(factorielle(17.1))
print(factorielle(-3))
print(factorielle(3))
print(factorielle(1000))
```

```
17.1 n'est pas un entier !
None
-3 est négatif !
None
6
40238726007709377354370243392300398571937486421071463254379991042
93851239862902059204420848696940480047998861019719605863166687299
40855890132382966994459099742450408707375991882362772718873251977
95059509952761208749754624970436014182780946464962910563938874378
48733711918104582578364784997701247663288983595573543251318532395
46307555740911426241747434934755342864657661166779739666882029120
37914385371958824980812686783837455973174613608537953452422158659
20192809087829730843139284440328123155861103697680135730421616874
60967587134831202547858932076716913244842623613141250878020800026
68315102734182797770478463586817016436502415369139828126481021309
76124489635992870511496497541990934222156683257208082133318611681
55361583654698404670897560290095053761647584772842188967964624494
16076535340819890138544248798495995331910172335555660213945039973
28075013783761530712776192684903435262520001588853514733161170210
96817592151090778801939317811419454525722386554146106289218796022
83897147608850627686296714667469756291123408243920816015378088989
96451826324367161676217916890977991190375403127462228998800519544
41428201218736174599264295658174662830295557029902432415318161721
46583203678690611726015878352075151628422554026517048330422614397
28693306169089796848259012545832716822645806652676995865268227280
70757813918581788896522081643483448259932660433676601769996128318
78838615027946595513115655203609398818061213855860030143569452722
20634463179746059468257310379008402443243846565724501440282188525
47093519062092902313649327349756551395872055965422874977401141334
96271542284586237738753823048386568897646192738381490014076731044
64025989949022222176590433990188601856652648506179970235619389701
86004081188972991831102117122984590164192106888438712185564612496
79872290851929681937238864261483965738229112312502418664935314397
13742853192664987533721894069428143411852015801412334482801505139
69429015348307764456909907315243327828826986460278986432113908350
21709500259738986355427719674282224875758676575234422020757363056
49882508796892816275384886339690995982628095612145099487170124451
46126037902930912088908694202851064018215439945715680594187274899
09425474217358240106367740459574178516082923013535808184009699637
52423056085590370062427124341690900415369010593398383577793941097
02775347200000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
```

Exercice

Proposez une fonction récursive `mon_pgcd` qui calcule le pgcd de 2 entiers à l'aide de l'algorithme d'Euclide :

- si $b = 0$, $PGCD(a, b) = |a|$,
- si $b \neq 0$, $PGCD(a, b) = PGCD(b, r)$, où r est le reste de la division euclidienne de a par b.

In [25]:
```
def mon_pgcd(a, b):
    if type(a) != int or type(b) != int:
        print(f"Dans mon_pgcd a={a} et b={b} doivent être entiers")
        return
    if b == 0:
        return abs(a)
    return mon_pgcd(b, a % b)

a, b = 153*17, 153*15
print(f"Le PGCD de {a} et de {b} vaut {mon_pgcd(a, b)}")
```

```
Le PGCD de 2601 et de 2295 vaut 153
```