Les λ-fonctions

Nous commençons par détailler la structure des λ-fonctions. Un exemple sera plus parlant. Construisons la fonction $x \mapsto x^2$.

In [4]:
```
carre = lambda x: x**2
x = 2.1
print(f"Evaluation de la fonction carre sur x={x} : {carre(x)}")
```

```
Evaluation de la fonction carre sur x=2.1 : 4.41
```

Il est possible d'avoir plusieurs variables et plusieurs sorties :

In [5]:
```
echange = lambda x, y: (y, x)
x, y = 1, 2
print(f"On échange {x} et {y} : {echange(x, y)}")
```

```
On échange 1 et 2 : (2, 1)
```

On peut faire des choses plus compliquées avec l'opérateur ternaire :

In [6]:
```
inverse = lambda x: 1./x if x != 0 else None
x, y = 2, 0
print(f"1/{x} = {inverse(x)}")
print(f"1/{y} = {inverse(y)}")
```

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

In [ ]:
```

```

Exercice

- Créez une λ-fonction qui prend une liste l en paramètre et qui retourne la liste composée des carrés de la liste l.
- Evaluez la fonction sur la liste des entiers de 1 à 10.

In [11]:
```
liste_carres = lambda l: [lk*lk for lk in l]
print(liste_carres([k for k in range(1, 11)]))
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Attention : on ne doit pas faire de récursif avec les λ-fonctions même si c'est théoriquement possible. La raison est que cela peut vite devenir gourmant en mémoire et faire planter l'ordinateur.

Voici un exemple quand même :

In [13]:
```
factorielle = lambda n: n*factorielle(n-1) if n > 1 else 1
print(factorielle(3))
print(factorielle(1000))
```

```
6
402387260077093773543702433923003985719374864210714632543799910429
938512398629020592044208486969404800479988610197196058631666872994
808558901323829669944590997424504087073759918823627727188732519779
505950995276120874975462497043601418278094646496291056393887437886
487337119181045825783647849977012476632889835955735432513185323958
463075557409114262417474349347553428646576611667797396668820291207
379143853719588249808126867838374559731746136085379534524221586593
201928090878297308431392844403281231558611036976801357304216168747
609675871348312025478589320767169132448426236131412508780208000261
683151027341827977704784635868170164365024153691398281264810213092
761244896359928705114964975419909342221566832572080821333186116811
553615836546984046708975602900950537616475847728421889679646244945
160765353408198901385442487984959953319101723355556602139450399736
280750137837615307127761926849034352625200015888535147331611702103
968175921510907788019393178114194545257223865541461062892187960223
838971476088506276862967146674697562911234082439208160153780889893
964518263243671616762179168909779911903754031274622289988005195444
414282012187361745992642956581746628302955570299024324153181617210
465832036786906117260158783520751516284225540265170483304226143974
286933061690897968482590125458327168226458066526769958652682272807
075781391858178889652208164348344825993266043367660176999612831860
788386150279465955131156552036093988180612138558600301435694527224
206344631797460594682573103790084024432438465657245014402821885252
470935190620929023136493273497565513958720559654228749774011413346
962715422845862377387538230483865688976461927383814900140767310446
640259899490222221765904339901886018566526485061799702356193897017
860040811889729918311021171229845901641921068884387121855646124960
798722908519296819372388642614839657382291123125024186649353143970
137428531926649875337218940694281434118520158014123344828015051399
694290153483077644569099073152433278288269864602789864321139083506
217095002597389863554277196742822248757586765752344220207573630569
498825087968928162753848863396909959826280956121450994871701244516
461260379029309120889086942028510640182154399457156805941872748998
094254742173582401063677404595741785160829230135358081840096996372
524230560855903700624271243416909004153690105933983835777939410970
027753472000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
```

Exercice

1. Créez la fonction $x \mapsto x + 1$ appelée `right_shift`
2. Créez la fonction $f \mapsto fof$ appelée `compose` . Cette fonction prend donc en paramètre une fonction et retourne une fonction...
3. Evaluez `compose(right_shift)(0)` qui devrait vous retourner `2` !

In [14]:
```
right_shift = lambda x: x+1
compose = lambda f: lambda x: f(f(x))
print(compose(right_shift)(0))
```

```
2
```