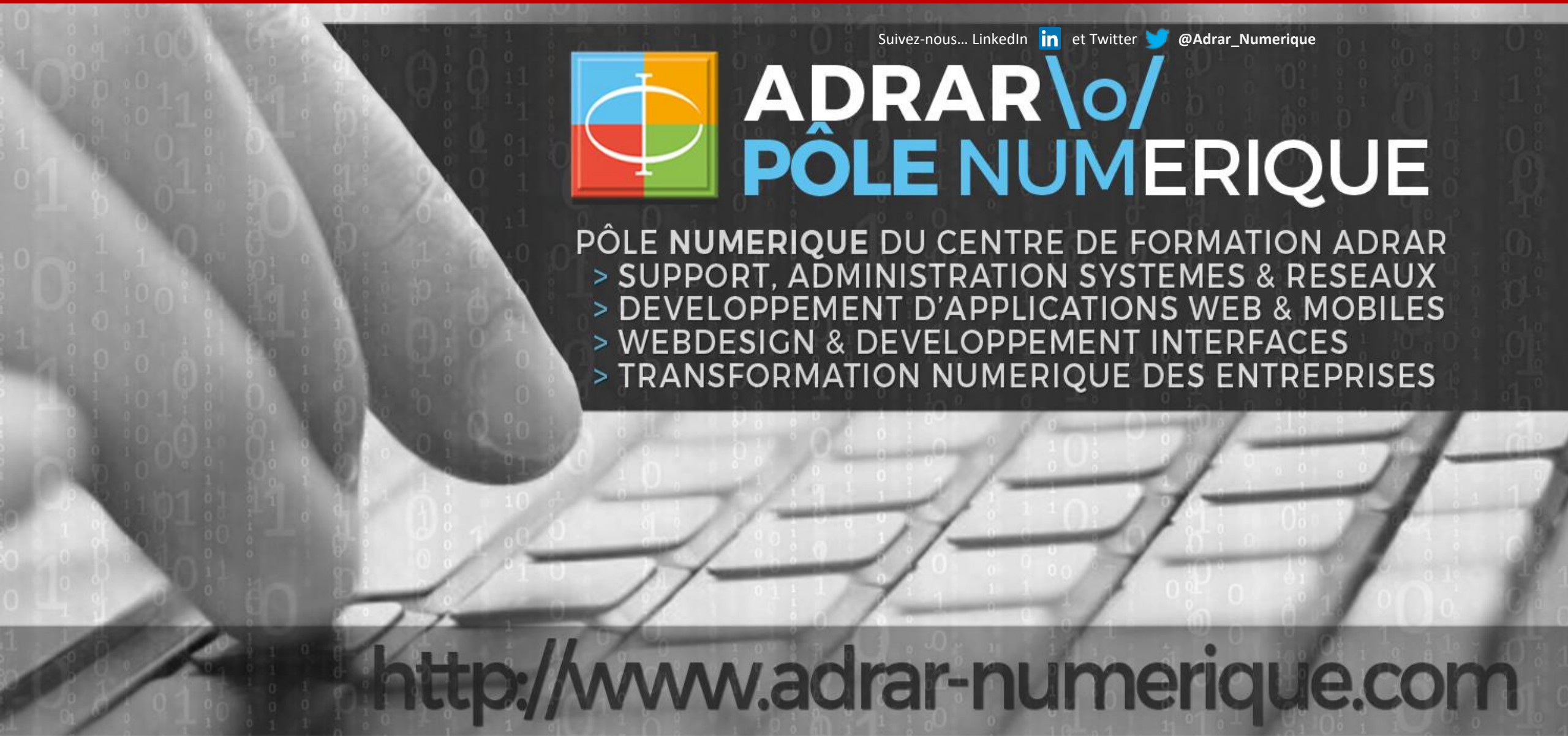
# Présentation ADRAR Pôle Numérique



python™

# LES FONCTIONS

Afin de structurer son code et également de l'optimiser on va l'organiser sous forme de fonctions. Les fonctions vont nous permettre de nous y retrouver plus facilement et également de factoriser nos éléments.
Une fonction n'est rien de plus qu'un bloc d'instruction qui ne sera exécuté uniquement quand on l'appellera.
La norme veut que le corps d'un code correctement constitué ne fasse que des appels de fonctions et des affichages.

Une fonction est composée de 5 éléments:

- Le mot clé « def » permettant de dire qu'on définit la fonction
- Le nom de la fonction (en l'occurrence Carre)
- Les paramètres de la fonction, entre parenthèse suivit de « : »
- Le corps la fonctions (les actions à effectuer)
- Ce que la fonction renvoie, si elle renvoie quelque chose, par le mot clé « return »

```python
def Carre(nombre):
    #Corp de la fonction
    return nombre**2


a = Carre(8)

print(a)
```

Pour appeler une fonction il suffit d'utiliser son nom suivi de parenthèses avec à l'intérieur autant de valeur que la fonction a de paramètres

```python
def HelloWorld():
    #Corp de la fonction
    print("Hello World")


HelloWorld()
```

```python
def Multiplier(nbr1, nbr2):
    #Corp de la fonction

    return nbr1*nbr2


a = Multiplier(8, 7)
```

Nous avons la possibilité d'appeler une fonction dans une fonction. Nous pouvons utiliser le même nom de variable dans les deux fonctions sans qu'il y ai de confusion car une variable n'existe que dans une fonction (en dehors de la fonction la variable n'existe pas. Son nom peut donc être ré utilisé sans problème)

```
def carre(nbr):
    return nbr**2


def cube(nbr):
    return carre(nbr)*nbr
```

Pour stocker le résultat d'une fonction dans une variable il faut que celle-ci dispose du mot clé « return ». Ce mot clé indique quelle valeur la fonction de retourner. En l'occurrence si je déclare une variable et que je l'initialise avec la fonction cube() on stockera dedans le résultat de la fonction.

resultat = cube(3)

La variable « resultat » vaudra 27, (3**2)*3.

# EXERCICES

- Faire une fonction qui prend en paramètre une année de naissance et qui retourne un âge.
- Faire une fonction qui prend en paramètre deux nombres et qui retourne la multiplication de ces nombres.
- Faire une fonction qui prend en paramètre une chaine de caractère et qui affiche « OK » si c'est une adresse email valide ou « NOK » si elle n'est pas valide (une adresse email valide se compose de 4 lettres/chiffres, puis d'un @, puis de 6 lettres/chiffres, puis d'un point, puis de 3 lettres.

# *args et **kwargs

Dans une fonction il est possible de ne pas fixer le nombre d'argument que l'on veut passer en paramètre. Ou bien laisser la possibilité à l'appel de la fonction de gérer un nombre différent de paramètres.
En python nous avons la possibilité d'indiquer qu'une fonction va prendre un tuple de paramètre (de taille variable) ou un dictionnaire de paramètres. C'est ce qu'on va appeler les *args et les **kwargs.

```python
def addition(*arguments):
    somme = 0
    for i in arguments:
        somme += i

    return somme

print(addition(2, 4, 6))
print(addition(3, 6, 9, 12, 15, 18))
```

```
12
63
```

```python
def moyenne_matiere(**arguments):
    somme = 0
    for i in arguments:
        somme += arguments[i]

    return somme / len(arguments)


print(moyenne_matiere(francais=8, maths=12, histoire=18))
```

```
12.666666666666666
```

# LAMBDA ET INLINE

Il existe deux façons de simplifier l'écriture d'une fonction simple. Les fonctions lambda et les fonctions inline.

Syntaxe d'une fonction lamdba : **lambda argument : résultat**

Par exemple la fonction :

```
def plusUn(nbre):
    return nbre+1

a = plusUn(8)
```

Peut s'écrire :

```
plusUn = lambda nbre:nbre+1

a = plusUn(8)
```

Ou bien:

```
a = (lambda nbre:nbre+1)(8)
```

L'un des intérêt de lambda et des inline va être de pouvoir utiliser des fonctions facilement en tant qu'argument d'autre fonctions ou bien de faire des structures conditionnelles en une ligne.

```
a = [('quatre',4),("deux",2),("huit",8),("un",1)]

print(sorted(a, key=lambda x:x[1]))
```

```
[('un', 1), ('deux', 2), ('quatre', 4), ('huit', 8)]
```

```
a = [('quatre',4),("deux",2),("huit",8),("un",1)]

print(max([x[1] for x in a]))
```

```
8
```