Fonctions

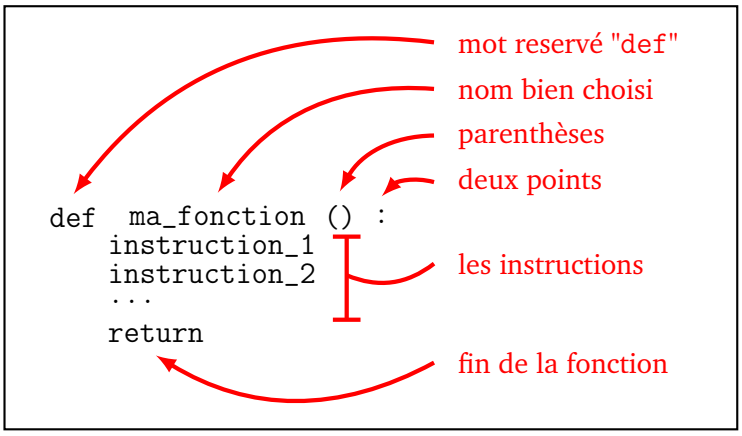
Écrire une fonction, c'est la façon la plus simple de regrouper du code pour une tâche bien particulière, dans le but de l'exécuter une ou plusieurs fois par la suite.

Cours 1 (Fonction (début))**.**
Une fonction en informatique est une portion réalisant un tâche bien précise et qui pourra être utilisée une ou plusieurs fois dans la suite du programme. Pour définir une fonction avec `Python`, c'est très simple. Voici deux exemples :

```
def dit_bonjour():
    print("Bonjour le monde !")
    return
```

```
def affiche_carres():
    for i in range(20):
        print(i**2)
    return
```

Les instructions sont regroupées dans un bloc indenté. Le mot `return` (optionnel) indique la fin de la fonction. Ces instructions ne sont exécutées que si j'appelle la fonction. Par exemple, chaque fois que j'exécute la commande `dit_bonjour()`, `Python` affiche la phrase « Bonjour le monde ! ». Chaque fois que j'exécute la commande `affiche_carres()`, `Python` affiche $0, 1, 4, 9, 16, \ldots$, c'est-à-dire les nombres i^2 pour $i = 0, \ldots, 19$.



Cours 2 (Fonction (suite))**.**
Les fonctions informatiques acquièrent tout leur potentiel avec :

- une ***entrée***, qui regroupent des variables qui servent de ***paramètres***,
- une ***sortie***, qui est un résultat renvoyé par la fonction (et qui souvent dépendra des paramètres d'entrée).

Voici deux exemples :

```
def affiche_mois(numero):
    if numero == 1:
        print("Nous sommes en janvier.")
    if numero == 2:
        print("Nous sommes en février.")
    if numero == 3:
        print("Nous sommes en mars.")
    # etc.
    return
```

Lorsqu'elle est appelée cette fonction affiche le nom du mois en fonction du nombre fourni en entrée. Par exemple `affiche_mois(3)` va afficher `"Nous sommes en mars."`.

```
def calcule_cube(a):
    cube = a * a * a    # ou bien a**3
    return cube
```

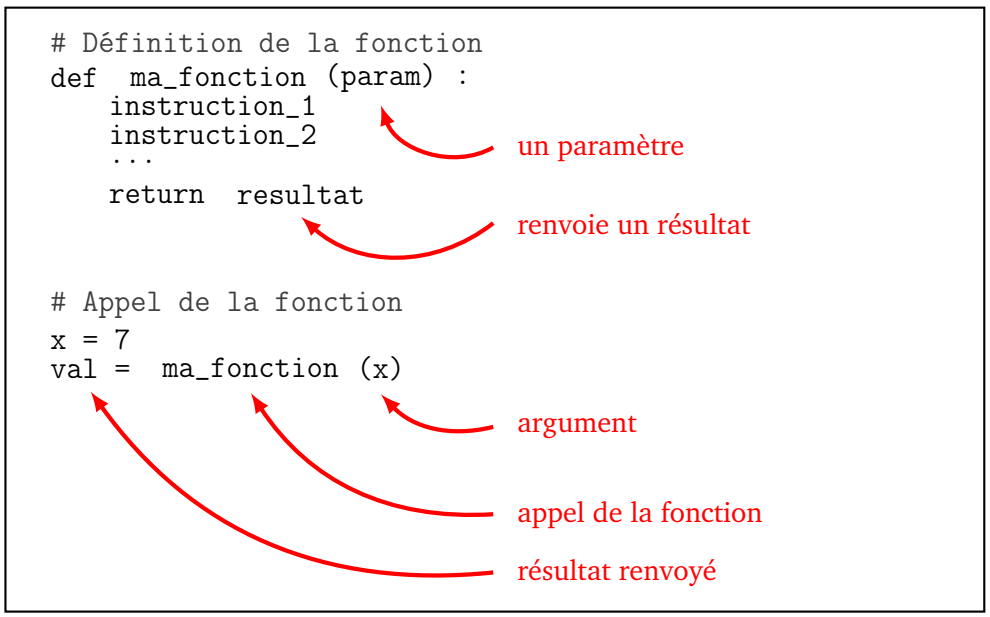
Cette fonction calcule le cube d'un nombre, par exemple `calcule_cube(2)` n'affiche rien mais renvoie la valeur 8. Cette valeur peut être utilisée ailleurs dans le programme. Par exemple, que font les instructions suivantes ?

```
x = 3
y = 4
z = calcule_cube(x) + calcule_cube(y)
print(z)
```

En terme mathématiques, on pose $x = 3$, $y = 4$, puis on calcule le cube de x, le cube de y et on les additionne :

$$z = x^3 + y^3 = 3^3 + 4^3 = 27 + 64 = 91$$

Ainsi le programme affiche 91.



Les avantages de la programmation utilisant des fonctions sont les suivants :

- on écrit le code d'une fonction une seule fois, mais on peut appeler la fonction plusieurs fois ;
- en divisant notre programme en petits blocs ayant chacun leur utilité propre, le programme est plus facile à écrire, à lire, à corriger et à modifier ;
- on peut utiliser une fonction (écrite par quelqu'un d'autre, comme par exemple la fonction `sqrt()`) sans connaître tous les détails internes de sa programmation.

Activité 1 (Premières fonctions)**.**

Objectifs : écrire des fonctions très simples.

1. **Fonction sans paramètre ni sortie.**
   (a) Programme une fonction appelée `affiche_table_de_7()` qui affiche la table de multiplication par 7 : $1 \times 7 = 7$, $2 \times 7 = 14\ldots$
   (b) Programme une fonction appelée `affiche_bonjour()` qui demande à l'utilisateur son prénom et affiche ensuite « Bonjour » suivi du prénom de l'utilisateur.
   Indication. Utilise `input()`.
2. **Fonction avec un paramètre et sans sortie.**
   (a) Programme une fonction appelée `affiche_une_table(n)` qui dépend d'un paramètre `n` et qui affiche la table de multiplication par l'entier n. Par exemple, la commande `affiche_une_table(5)` doit afficher : $1 \times 5 = 5$, $2 \times 5 = 10\ldots$
   (b) Programme une fonction appelée `affiche_salutation(formule)` qui dépend d'un paramètre `formule`. Cette fonction demande le prénom de l'utilisateur et affiche une formule de salutation suivi du prénom. Par exemple `affiche_salutation("Coucou")` afficherait « Coucou » suivi du prénom donné par l'utilisateur.
3. **Fonction sans paramètre et avec sortie.**
   Programme une fonction appelée `demande_prenom_nom()` qui demande d'abord le prénom de l'utilisateur, puis son nom et renvoie comme résultat l'identité complète avec le nom en majuscule. Par exemple, si l'utilisateur saisi « Dark » puis « Vador », la fonction renvoie la chaîne `"Dark VADOR"` (la fonction n'affiche rien).
   Indications.
   - Si `chaine` est une chaîne de caractères, alors `chaine.upper()` est la chaîne transformée avec les caractères en majuscules. Exemple : si `chaine = "Vador"` alors `chaine.upper()` renvoie `"VADOR"`.
   - On peut fusionner deux chaînes en utilisant le signe « + ». Exemple : `"Dark" + "Vador"` vaut `"DarkVador"`. Autre exemple : si `chaine1 = "Dark"` et `chaine2 = "Vador"` alors `chaine1 + " " + chaine2` vaut `"Dark Vador"`.

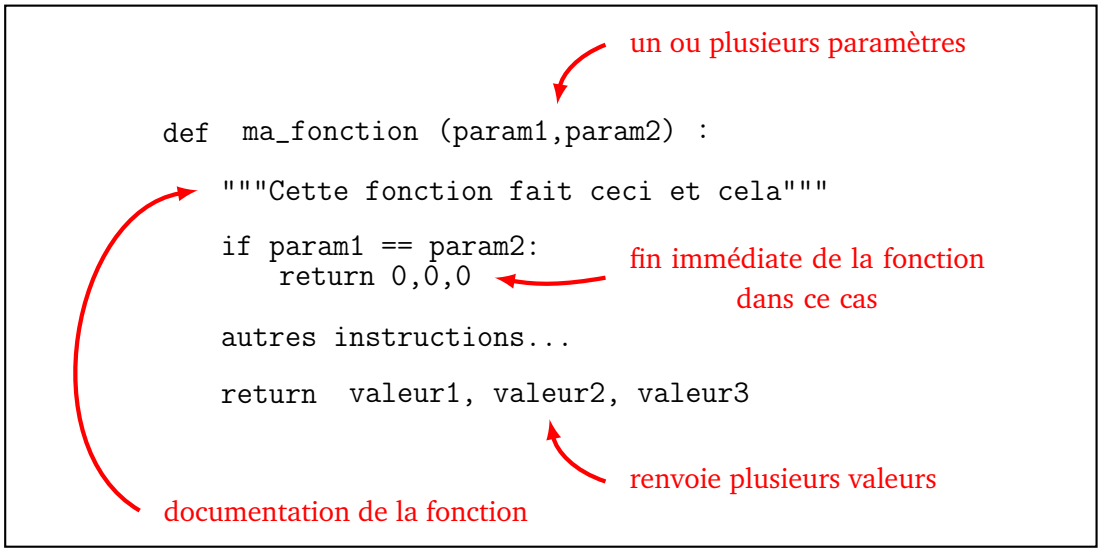
Cours 3 (Fonction (suite et fin pour l'instant))**.**
Une fonction peut avoir plusieurs paramètres et renvoyer plusieurs résultats. Par exemple, voici une fonction qui calcule et renvoie la somme et le produit de deux nombres donnés en entrée.

```
def somme_produit(a,b):
    """Calcule la somme et le produit de deux nombres"""
    s = a + b
```

```
    p = a * b
    return s, p

som, pro = somme_produit(6,7)
```

La dernière ligne appelle la fonction avec les arguments 6 (pour le paramètre `a`) et 7 (pour le paramètre `b`). Cette fonction renvoie deux valeurs, la première est affectée à `som` (qui vaut donc ici 13) et la seconde à `pro` (qui vaut donc 42).



Retenons donc :

- Il peut y avoir plusieurs paramètres en entrée.
- Il peut y avoir plusieurs résultats en sortie.
- Très important ! Il ne faut pas confondre afficher et renvoyer une valeur. L'affichage (par la commande `print()`) affiche juste quelque chose à l'écran. La plupart des fonctions n'affichent rien, mais renvoient une valeur (ou plusieurs). C'est beaucoup plus utile car cette valeur peut être utilisée ailleurs dans le programme.
- Dès que le programme rencontre l'instruction `return`, la fonction s'arrête et renvoie le résultat. Il peut y avoir plusieurs fois l'instruction `return` dans une fonction mais une seule sera exécutée. On peut aussi ne pas mettre d'instruction `return` si la fonction ne renvoie rien.
- Dans les instructions d'une fonction, on peut bien sûr faire appel à d'autres fonctions !
- Il est important de bien commenter tes programmes. Pour documenter une fonction, tu peux décrire ce qu'elle fait en commençant par un *docstring*, c'est-à-dire une description (en français) entourée par trois guillemets : `""" Ma fonction fait ceci et cela. """` à placer juste après l'entête.
- Lorsque l'on définit une fonction, les variables qui apparaissent entre les parenthèses sont appelées les ***paramètres*** ; par contre, lorsque l'on appelle la fonction, les valeurs entre les parenthèses sont appelées les ***arguments***. Il y a bien sûr une correspondance entre les deux.

Activité 2 (Encore des fonctions)**.**

Objectifs : construire des fonctions avec différents types d'entrée et de sortie.

1. **Trinômes.**

   (a) Écris une fonction `trinome_1(x)` qui dépend d'un paramètre `x` et qui renvoie la valeur du

trinôme $3x^2 - 7x + 4$. Par exemple `trinome_1(7)` renvoie 102.

(b) Écris une fonction `trinome_2(a,b,c,x)` qui dépend de quatre paramètres `a`, `b`, `c` et `x` et qui renvoie la valeur du trinôme $ax^2 + bx + c$. Par exemple `trinome_2(2,-1,0,6)` renvoie 66.

2. **Devises.**

(a) Écris une fonction `conversion_euros_vers_dollars(montant)` qui dépend d'un paramètre et qui pour une somme d'argent `montant`, exprimée en euros, renvoie sa valeur en dollars (tu prendras par exemple 1 euro = 1,15 dollar).

(b) Écris une fonction `conversion_euros(montant,devise)` qui dépend d'un paramètre `montant` et d'une monnaie `devise` et qui convertit la somme `montant` donnée en euros, dans la devise souhaitée. Exemples de devises : 1 euro = 1,15 dollar ; 1 euro = 0,81 livre ; 1 euro = 130 yens. Par exemple `conversion_euros(100,"livre")` renvoie 81.

Prends soin de donner un nom intelligible à tes fonctions ainsi qu'aux variables. N'oublie pas de documenter chaque fonction en ajoutant un petit texte explicatif entre triples guillemets au tout début de ta fonction.

3. **Volumes.**

Construis des fonctions qui calculent et renvoient des volumes :

- le volume d'un cube en fonction de la longueur d'un côté,
- le volume d'une boule en fonction de son rayon,
- le volume d'un cylindre en fonction du rayon de sa base et de sa hauteur,
- le volume d'une boîte parallélépipède rectangle en fonction de ses trois dimensions.

Pour la valeur de π, tu prendras soit la valeur approchée 3.14, soit la valeur approchée fournie par la constante `pi` du module `math`.

4. **Périmètres et aires.**

(a) Écris une fonction dont l'usage est `perimetre_aire_rectangle(a,b)` et qui renvoie en sortie le périmètre et l'aire d'un rectangle de dimensions a et b.

(b) Même question avec `perimetre_aire_disque(r)` pour le périmètre et l'aire d'un disque de rayon r.

(c) Utilise ta fonction précédente pour conjecturer à partir de quel rayon, l'aire d'un disque est plus grande que le périmètre de ce disque.

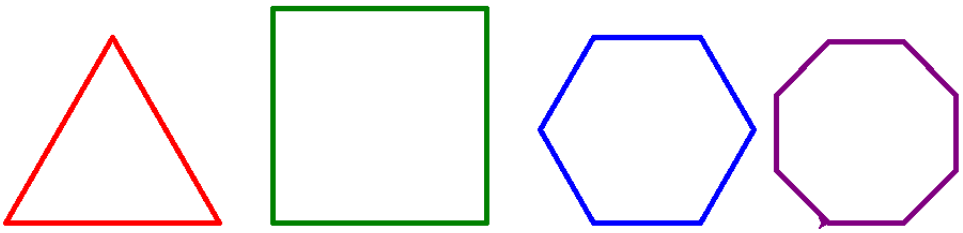
Indication. Si tu veux balayer les rayons en incrémentant la valeur de 0.1 à chaque fois, tu peux construire une boucle ainsi :

```
for rayon in range(0,30):
```

puis faire un appel à la fonction par `perimetre_aire_disque(rayon/10)`.

Activité 3 (Tortue)**.**

Objectifs : définir quelques fonctions qui dessinent des figures géométriques. Créer une fonction est similaire à créer un bloc avec Scratch*.*



1. Programme une fonction `triangle()` qui dessine un triangle (en rouge, chaque côté mesurant 200).
2. Programme une fonction `carre()` qui dessine un carré (en vert, chaque côté mesurant 200). Utilise une boucle « pour » afin de ne pas avoir à réécrire les mêmes instructions plusieurs fois.
3. Programme une fonction `hexagone(longueur)` qui trace un hexagone (en bleu) d'une longueur de côté donnée (l'angle pour tourner est de 60 degrés).
4. Programme une fonction `polygone(n,longueur)` qui trace un polygone régulier à n côtés et d'une longueur de côté donnée (l'angle pour tourner est alors de $360/n$ degrés).

Activité 4 (Toujours des fonctions)**.**

Objectifs : créer de nouvelles fonctions.

1. (a) Voici la réduction pour le prix d'un billet de train en fonction de l'âge du voyageur :
   - réduction de 50% pour les moins de 10 ans ;
   - réduction de 30% pour les 10 à 18 ans ;
   - réduction de 20% pour les 60 ans et plus.

   Écris une fonction `reduction()` qui renvoie la réduction en fonction de l'âge et dont les propriétés sont rappelées dans le cadre ci-dessous :

   `reduction()`

   Usage : `reduction(age)`
   Entrée : un entier correspondant à l'âge
   Sortie : un entier correspondant à la réduction

   Exemples :
   - `reduction(17)` renvoie 30.
   - `reduction(23)` renvoie 0.

   (b) Déduis-en une fonction `montant()` qui calcule le montant à payer en fonction du tarif normal et de l'âge du voyageur.

   `montant()`

   Usage : `montant(tarif_normal,age)`
   Entrée : un nombre `tarif_normal` correspondant au prix sans réduction et `age` (un entier)
   Sortie : un nombre correspondant au montant à payer après réduction
   Remarque : utilise la fonction `reduction()`

   Exemple : `montant(100,17)` renvoie 70.

Une famille achète des billets pour différents trajets, voici le tarif normal de chaque trajet et les âges des voyageurs :
- tarif normal 30 euros, enfant de 9 ans ;
- tarif normal 20 euros, pour chacun des jumeaux de 16 ans ;
- tarif normal 35 euros, pour chacun des parents de 40 ans.

Quel est le montant total payé par la famille ?

2. On souhaite programmer un petit quiz sur les tables de multiplication.

(a) Programme une fonction `calcul_est_exact()` qui décide si la réponse donnée à une multiplication est juste ou pas.

`calcul_est_exact()`

Usage : `calcul_est_exact(a,b,reponse)`
Entrée : trois entiers, `reponse` étant la réponse proposée au calcul de $a \times b$.
Sortie : « vrai » ou « faux », selon que la réponse est correcte ou pas

Exemples :
- `calcul_est_exact(6,7,35)` renvoie `False`.
- `calcul_est_exact(6,7,42)` renvoie `True`.

(b) Programme une fonction qui affiche une multiplication, demande une réponse et affiche une petite phrase de conclusion. Tout cela en français ou en anglais !

`test_multiplication()`

Usage : `test_multiplication(a,b,lang)`
Entrée : deux entiers, la langue choisie (parmi `"francais"` ou `"anglais"`)
Sortie : rien
Remarque : utilise la fonction `calcul_est_exact()`

Exemple : `test_multiplication(6,7,"anglais")` demande, en anglais, la réponse au calcul 6×7 et répond si c'est correct ou pas.

Bonus. Améliore ton programme afin que l'ordinateur propose tout seul des opérations aléatoires au joueur. (Utilise la fonction `randint()` du module `random`.)

Activité 5 (Égalité expérimentale)**.**

Objectifs : utiliser l'ordinateur pour expérimenter des égalités de fonctions.

1. (a) Construis une fonction `valeur_absolue(x)` qui renvoie la valeur absolue d'un nombre (sans utiliser la fonction `abs()` de `Python` !).

(b) Construis une fonction `racine_du_carre(x)` qui correspond au calcul de $\sqrt{x^2}$.

(c) On dit que deux fonctions (d'une variable) f et g sont ***expérimentalement égales*** si $f(i) = g(i)$ pour $i = -100, -99, \ldots, 0, 1, 2, \ldots, 100$. Vérifie par ordinateur que les deux fonctions définies par

$$|x| \qquad \text{et} \qquad \sqrt{x^2}$$

sont expérimentalement égales.

2. (a) Construis une fonction à deux paramètres `F1(a,b)` qui renvoie $(a+b)^2$. Même chose avec `F2(a,b)` qui renvoie $a^2 + 2ab + b^2$.

(b) On dit que deux fonctions de deux variables F et G sont ***expérimentalement égales*** si $F(i,j) = G(i,j)$ pour tout $i = -100, -99, \ldots, 100$ et pour tout $j = -100, -99, \ldots, 100$. Vérifie par ordinateur que les fonctions définies par $(a+b)^2$ et $a^2 + 2ab + b^2$ sont expérimentalement égales.

(c) Je sais que l'une des deux égalités suivantes est vraie :

$$(a-b)^3 = a^3 - 3a^2b - 3ab^2 + b^3 \qquad \text{ou} \qquad (a-b)^3 = a^3 - 3a^2b + 3ab^2 - b^3.$$

Aide-toi de l'ordinateur pour décider laquelle est-ce !

3. (a) Construis une fonction `sincos(x)` qui renvoie $(\sin(x))^2 + (\cos(x))^2$ et une autre `un(x)` qui renvoie toujours 1. Ces deux fonctions sont-elles expérimentalement égales (au sens de la première question) ? Cherche quelle peut être la cause de cette réponse.

(b) On pose $\epsilon = 0,00001$. On dit que deux fonctions (d'une variable) f et g sont ***expérimentalement approximativement égales*** si $|f(i) - g(i)| \leqslant \epsilon$ pour $i = -100, -99, \ldots, 100$. Est-ce que maintenant les deux fonctions définies par `sincos(x)` et `un(x)` vérifient ce critère ?

(c) Vérifie de façon expérimentale et approchée les égalités :

$$\sin(2x) = 2\sin(x)\cos(x), \qquad \cos\left(\frac{\pi}{2} - x\right) = \sin(x).$$

(d) **Bonus. Un contre-exemple.** Montre que les fonctions définies par $g_1(x) = \sin(\pi x)$ et $g_2(x) = 0$ sont expérimentalement égales (avec notre définition donnée plus haut). Mais montre aussi que l'on n'a pas $g_1(x) = g_2(x)$ pour tout $x \in \mathbb{R}$.

Cours 4 (Variable locale)**.**
Voici une fonction toute simple qui prend en entrée un nombre et renvoie le nombre augmenté de un.

```
def ma_fonction(x):
    x = x + 1
    return x
```

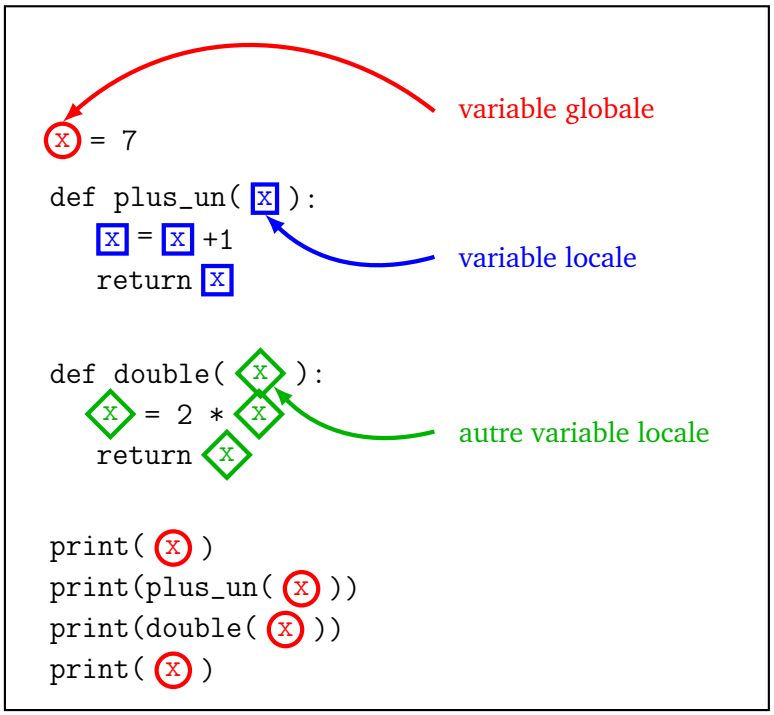
- Bien évidemment `ma_fonction(3)` renvoie 4.
- Si je définis une variable par `y = 5` alors `ma_fonction(y)` renvoie 6. Et la valeur de `y` n'a pas changé, elle vaut toujours 5.
- Voici la situation délicate qu'il faut bien comprendre :

```
x = 7
print(ma_fonction(x))
print(x)
```

  — La variable `x` est initialisée à 7.
  — L'appel de la fonction `ma_fonction(x)` est donc la même chose que `ma_fonction(7)` et renvoie logiquement 8.
  — Que vaut la variable `x` à la fin ? La variable `x` est inchangée et vaut toujours 7 ! Même s'il y a eu entre temps une instruction `x = x + 1`. Cette instruction a changé le `x` à l'intérieur de la fonction, mais pas le `x` en dehors de la fonction.

- Les variables définies à l'intérieur d'une fonction sont appelées ***variables locales***. Elles n'existent pas en dehors de la fonction.
- S'il existe une variable dans une fonction qui porte le même nom qu'une variable dans le programme (comme le `x` dans l'exemple ci-dessus), c'est comme si il y avait deux variables distinctes ; la variable locale n'existant que dans la fonction.

Pour bien comprendre la portée des variables, tu peux colorier les variables globales d'une fonction en rouge, et les variables locales avec une couleur par fonction. Le petit programme suivant définit une fonction qui ajoute un et une autre qui calcule le double.



Le programme affiche d'abord la valeur de `x`, donc `7`, puis il l'augmente de `1`, il affiche donc `8`, puis il affiche le double de `x`, donc `14`. La variable globale `x` n'a jamais changé, le dernier affichage de `x` est donc encore `7`.

Il est tout de même possible de forcer la main à `Python` et de modifier une variable globale dans une fonction à l'aide du mot clé `global`. Voir la fiche « Calculatrice polonaise – Pile ».