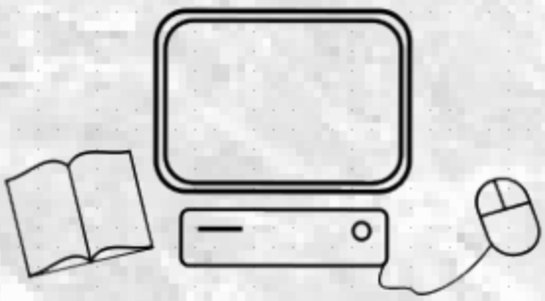
Formation Rapide

Programmer en Scheme

avec Gimp

Exemples de code



Claude Turrier

AP Apprendre et Pratiquer

Du même auteur,

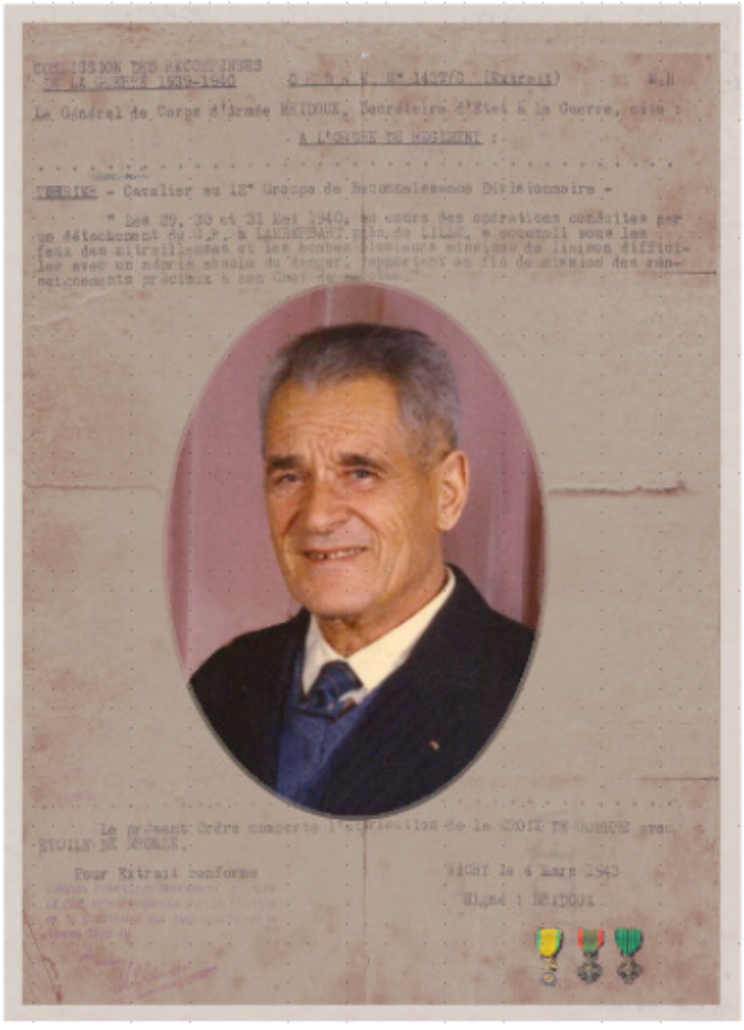
Éditions Ellipses

COLLECTION TECHNOSUP
- Le son - théorie et technologie (2015)
- Photographie numérique (2013)

COLLECTION DE CLIC EN CLIC
- Créer et retoucher des images avec Illustrator (2009)
- Créer et retoucher des images avec Gimp (2008)
- Photos numériques (2008)
- Initiation à la modélisation et la programmation 3D (2007)

Programmer en Scheme avec Gimp
IDCTUR21032021

à max
"ce qui est possible..."



Maximin Turrier (1901-1981)

Avant- Propos

GIMP (GNU Image Manipulation Program) est un logiciel libre de traitement d'images fonctionnant sous divers systèmes d'exploitation et en particulier sous **Linux**, **Windows** et **MacOS**.

Gimp est interfacé avec divers langages de programmation et notamment, de façon native, avec le langage **Scheme** et avec l'interpréteur de langage **TinyScheme**. Cette particularité permet d'étendre les possibilités de Gimp de façon quasi-infinie, à travers l'écriture de plugins.

Ce livre s'adresse à toutes les personnes qui souhaitent apprendre à écrire, installer et utiliser des scripts en langage Scheme pour Gimp.

Son objectif est de permettre à chacun de se former rapidement et efficacement, à l'aide d'une présentation simple et concrète accompagnée de nombreux exemples de code.

Ce livre comprend 4 chapitres et un index :

Le premier chapitre constitue une formation rapide et opérationnelle aux notions essentielles du langage Scheme ;

Le second chapitre montre comment créer, installer puis utiliser des scripts en langage Scheme, pour Gimp ;

Le troisième chapitre montre comment réaliser des scripts de traitement par lot, permettant d'automatiser la répétition d'un traitement donné sur un grand nombre d'images ;

Le quatrième chapitre compare et met en relief le double mode de fonctionnement de Gimp : le mode interactif (mode manuel classique, en utilisant les fenêtres et les menus de Gimp) et le mode non interactif (mode programmatique, à l'aide d'écriture de scripts).

L'index permet de retrouver rapidement les pages du livre dans lesquelles un mot-clé donné est évoqué.

Les exemples de code inclus dans ce livre on été testés

✓	Sous **Linux** en utilisant Gimp 2.8.22, TinyScheme 1.41 (mini interpréteur Scheme), SCM version 5f2 (interpréteur Scheme) et le Terminal de commandes Gnome 3.28.2 de Ubuntu 18.04
✓	Sous **Windows 10** en utilisant Gimp 2.10.22

Les notions et les exemples présentés ci-après sont valables sous **Linux**, **Windows** ou **MacOS**, à condition d'utiliser, pour les chemins complets de fichiers, la syntaxe adaptée à l'OS concerné (par exemple "/" et "/*.jpg" sous Linux deviennent "\\" et "*.jpg" sous Windows...)

▸ ***Exemple***

```
Linux :  "/usr/share/gimp/2.0/scripts/"
Windows: "C:\\Programmes\\GIMP2\\share\\gimp\2.0\scripts"
```

Tous les scripts utilisés dans ce livre sont regroupés en annexe, en format **Linux** et **Windows** afin de pouvoir les recopier plus facilement.

Tout au long du livre, on utilise les signes suivants :

- ▸ Pour signaler un exemple
- ⚑ Pour indiquer une remarque
- ↯ Pour signaler les lignes de code qui se trouvent coupées uniquement en raison de la largeur de la page du livre

Les exemples et les remarques sont écrits avec une police de caractères plus petite et sont légèrement décalés par rapport à la marge gauche du document.

▸ ***Exemple***

```
ts> (* 3 2)
```

⚑ ***Remarque***

L'utilisation de la fonction car, pour récupérer uniquement le nom du fichier, est rendue nécessaire par le fait que la fonction gimp-image-get-name retourne le nom sous forme de liste.

Une ligne d'instruction coupée en deux pour les besoins de la mise en page doit en réalité être considérée comme une seule ligne.

```
(gimp-drawable-set-pixel moncalque ↯
   3 3 4 monpixel)

⇨(gimp-drawable-set-pixel moncalque 3 3 4 monpixel)
```

Enfin, les noms des scripts utilisés dans ce livre sont volontairement précédés des lettres ct (initiales de l'auteur) afin de pouvoir les retrouver plus facilement dans le répertoire des scripts de Gimp. Naturellement chacun peut utiliser la méthode de son choix pour nommer ses scripts.

Je vous souhaite une bonne lecture...

Sommaire

Chapitre I
Débuter en Scheme

Pour s'entraîner à programmer en langage **Scheme**, on peut se limiter à utiliser uniquement la console de scripts intégrée au logiciel **Gimp**. Cependant, on peut également utiliser, en complément, les interpréteurs **TinyScheme** et **scm** depuis le Terminal de commande.

1.1) Utiliser la console Script-Fu de Gimp

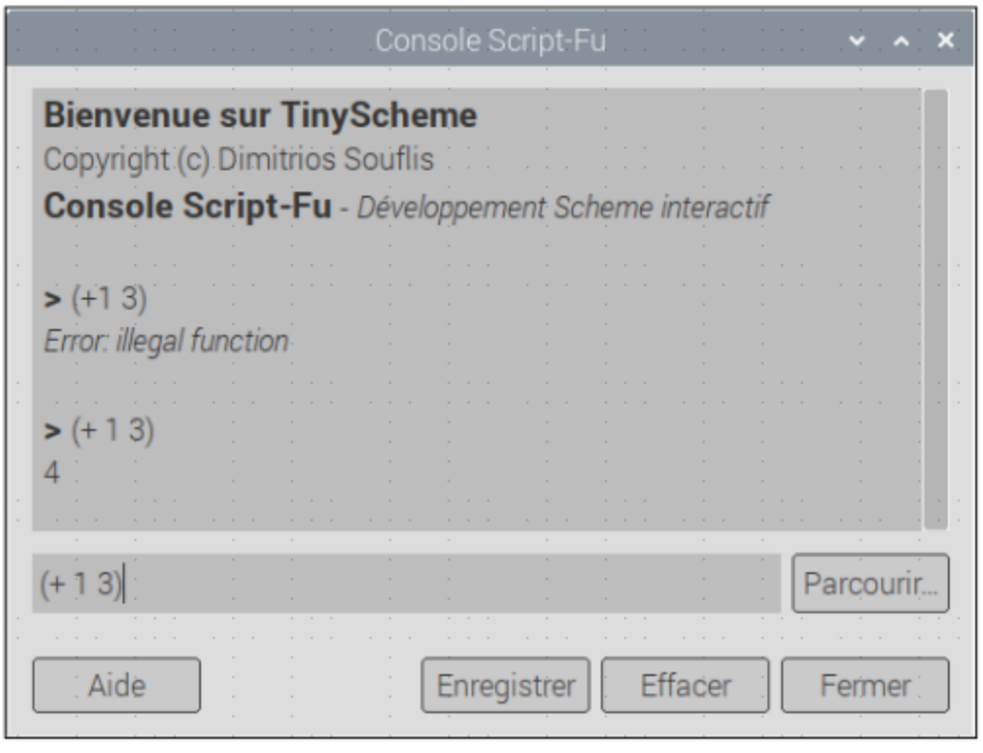
Gimp contient à la fois un interpréteur de commande Scheme et une console, appelée **console Script-Fu**, qui permettent d'exécuter, une par une ou regroupées, des lignes de script écrites en langage Scheme.

Une fois Gimp installé, la console Script-Fu est accessible depuis le menu suivant de Gimp :

Filtres > Script-Fu > Console

▸ ***Exemple***

On ouvre la Console Script-Fu de Gimp. On saisit l'instruction (+1 3), dans la zone de saisie, puis on appuie la touche Entrée. Le nombre 4 est automatiquement calculé et retourné dans la fenêtre de la console. Si on saisit (+1 3) au lieu de (+ 1 3), un message *Error: illegal function* est retourné car la syntaxe de Scheme impose de laisser un espace entre le + et le 1 ainsi qu'un espace entre le 1 et le 3.



1.2) Utiliser l'interpréteur TinyScheme

La console Script-Fu de Gimp permet de tester des lignes de code en langage Scheme, une par une ou regroupées en plusieurs lignes. Elle ne permet cependant pas de tester un fichier scheme **.scm** si celui-ci n'est pas préalablement enregistré en tant que script pour Gimp.

Pour tester un fichier scheme d'extension **.scm** en dehors de Gimp, il suffit d'utiliser un interpréteur de langage Scheme (**TinyScheme** ou **scm** par exemple) de façon autonome.

Sous **Linux**, on installe les interpréteurs **tinyScheme** et **scm**, en exécutant les commandes classiques suivantes depuis le Terminal de commandes (on peut remarquer que le paquet **slib** est installé en même temps que **scm**)

```
sudo apt update
sudo apt install tinyscheme
sudo apt install scm
```

Une fois tinyscheme et scm installés, à l'aide des commandes pré-citées, on vérifie qu'ils sont bien pris en compte par Linux. Pour cela on saisit la commande suivante depuis le Terminal de commandes : `dpkg-query -l`

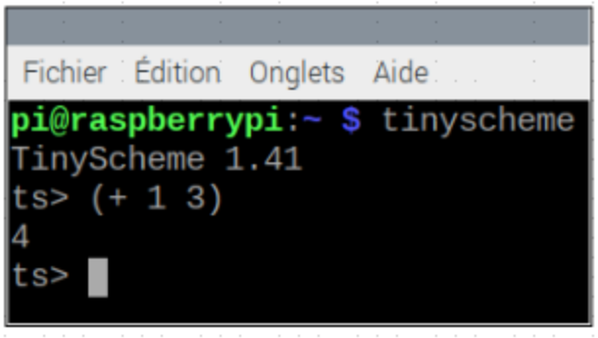
Cette commande retourne la liste des paquets installés sur l'ordinateur. Dans cette liste, on doit trouver les lignes suivantes.

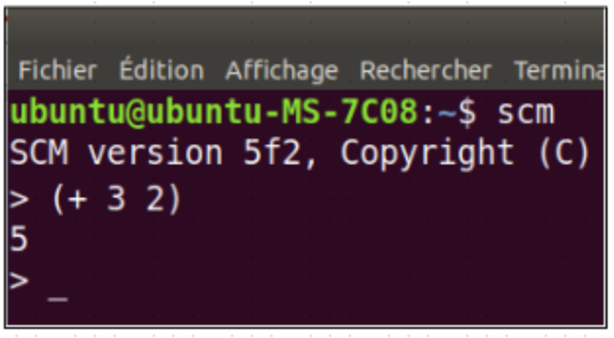
```
ii scm  5f2-2  amd64   Scheme language interpreter
ii slib 3b1-5  all     Portable Scheme library
ii tinyscheme 1.41.svn.2016.03.21-1 armhf small scheme implementation
```

On utilise TinyScheme et scm comme la console Script-Fu de Gimp.

▸ ***Exemple***

On démarre tinyscheme ou scm à l'aide de la commande tinyscheme ou scm. On reçoit alors l'invite de commande ts> ou >. L'instruction (+ 1 3), par exemple, retourne le résultat 4, identique à celui obtenu avec la console script-fu de Gimp.





Sous **Windows**, on peut trouver ces outils, ou des outils équivalents, sur divers sites web.

1.3) Calculer des expressions numériques

Pour calculer des expressions numériques, le langage **Scheme** utilise la notation préfixée. Les expressions sont interprétées en appliquant l'opérateur situé à gauche (+, -, *, /) aux valeurs des opérandes situés à sa droite:
Ainsi (* x y) est interprété par "j'effectue la multiplication de x par y", comme en langage naturel.

Expression numérique **Syntaxe**: (opérateur x1 x2)
opérateur: +, -, *, / x1, x2: opérandes (valeurs numériques, variables)

▸ **Exemple**

```
ts> (* 3 2)
6
ts>
```

On peut naturellement utiliser des nombres décimaux.

▸ **Exemple**

```
ts> (* 3 0.5)
1.5
ts>
```

Lorsque les instructions sont imbriquées, les valeurs internes sont calculées en premier et les résultats de ces calculs sont ensuite utilisés comme opérandes.

▸ **Exemple**

```
ts> (+ (+ 1 2) (+ 3 4))
10
ts>
```

Le nombre de parenthèses et leurs positions dépendent des opérateurs utilisés.

▸ **Exemple**

```
ts> (* 1 (* 2 (* 3 (* 4 5))))
120
ts>
```

⚑ **Remarques**

Pour effacer l'écran du Terminal on utilise le menu **Terminal > Réinitialiser** et effacer.

Pour quitter TS, depuis le Terminal de commande, on utilise la commande **(quit)**.

Pour effacer le contenu de la fenêtre de la console script-fu de Gimp, il suffit de cliquer le bouton **Effacer** sité en bas de la console

On peut également manipuler des chaines de caractères (strings). Celle-ci doivent être précédées du caractère ' ou du mot clef quote sinon l'interpréteur Scheme retourne le message *Error: eval: unbound variable:*

▶ **Exemple**
```
ts> 'abc
abc
ts> (quote abc)
abc
ts> abc
Error: eval: unbound variable: abc
```

1.4) Utiliser des variables

On distingue deux types de variables : les variables globales dont les valeurs sont visibles depuis tout le programme et les variables locales dont les valeurs ne sont visibles que dans la portée de création de ces variables.

1.4.1) Variables globales

Pour déclarer une variable globale, on utilise l'expression **define**.

Définition d'une variable globale **Syntaxe**: (define var expr)
var: nom de la variable expr: expression définissant la valeur de la variable

Si la variable est un nombre on écrit directement le nombre.
Si la variable est une chaine de caractères on précède la valeur de cette chaine par une apostrophe.

▶ **Exemple**
```
ts> (define x 3.75)
x
ts> x
3.75
ts> (define x '3.75jean)
x
ts> x
3.75jean
ts>
```

On peut modifier la valeur d'une variable à l'aide de l'instruction set!

Modification d'une variable **Syntaxe**: (set! var (expr))
var: nom de la variable expr: expression permettant de modifier la valeur de la variable

▶ **Exemple**

```
ts> (define x 3.0)
x
ts> (set! x (+ x 3.5))
6.5
ts> (define x 'jean)
x
ts> (set! x 'marc)
marc
ts>
```

1.4.2) Variables locales

Bien qu'il existe plusieurs méthode pour déclarer une variable locale, la déclaration d'une ou plusieurs variables locales est souvent effectuée à l'aide de l'expression **let***.

Définition d'une variable locale **Syntaxe**: (let* ((var1 val1) (var2 val2)…) (instr))
var1, var2,...: noms des variables val1, val2,...: valeurs des variables instr: instructions (éventuelles) agissant sur les variables

▶ **Exemple**

```
ts> (let* ((x 1.5) (y 3.2)) (+ x y))
4.7
ts> (let* ((x 1.5)) (set! x (+ x 3.5)))
5.0
ts>
```

Les variables locales sont conservées uniquement dans leur portée, c'est la raison pour laquelle, dans l'exemple suivant, on obtient un message d'erreur car on essaye de rappeler la variable x en dehors de sa portée de création.

▶ **Exemple**

```
TinyScheme 1.41
ts> (let* ( (x 1.5)) (set! x (+ x 3.5)))
5.0
ts> x
Error: eval: unbound variable: x
ts>
```

On peut facilement vérifier qu'avec une variable globale, le rappel en dehors de la portée fonctionne.

▶ **Exemple**

```
ts> (define x 3.5)
x
ts> (set! x (+ x 3.5))
7.0
ts>
```

1.5) Utiliser des listes

Une liste est une suite d'éléments (nombres, chaînes de caractères, variables) rangés dans l'ordre.

```
Définition d'une liste
Syntaxe: '(element1 element2 ...)
```

```
  ': caractère apostrophe
  element1, element2,...: éléments de la liste
```

▶ **Exemple**

```
TinyScheme 1.41
ts> '(1 2 3.5)
(1 2 3.5)
ts> '(jean michel rene)
(jean michel rene)
ts> '(x y z)
(x y z)
ts> '(jean 1.5  x)
(jean 1.5 x)
```

Il existe diverses fonctions permettant de manipuler les listes. Les fonctions **car**, **cdr**, **cons** , **cadr** et **list** sont souvent utilisées.

```
Fonction car : retourne le premier élément d'une liste
Syntaxe: (car '(el1 el2 el3...))
```

```
  ': caractère apostrophe
  el1, el2,...: éléments de la liste
```

▶ **Exemple**

```
TinyScheme 1.41
ts> (car '(jean 1.5 x 3.5))
jean
```

```
Fonction cdr : retourne, sous forme de liste, tous les éléments qui
suivent le premier élément d'une liste
Syntaxe: (cdr '(el1 el2 el3...))
```

```
  ': caractère apostrophe
  el1, el2,...: éléments de la liste
```

▶ **Exemple**

```
TinyScheme 1.41
ts> (cdr '(jean 1.5 x 3.5))
(1.5 x 3.5)
```

```
Fonction cons : ajoute un élément au début d'une liste
Syntaxe: (cons el0 '(el1 el2 ...))
```

```
  ': caractère apostrophe
  el0: élément à ajouter au début de la liste
  el1, el2,...: éléments de la liste
```

▶ **Exemple**

```
ts> (cons 'claude '(jean 1.5 x 3.5))
(claude jean 1.5 x 3.5)
ts> (cons 7.2 '(jean 1.5 x 3.5))
(7.2 jean 1.5 x 3.5)
ts>
```

Fonction cadr : retourne le second élément d'une liste **Syntaxe**: (cadr '(el1 el2 ...))
': caractère apostrophe el1, el2,...: éléments de la liste

▶ **Exemple**

```
> (cadr '(1 2 3))
2
```

Les listes créées ci-dessus sont constituées de nombres et de chaînes de caractères. Pour créer des listes contenant des variables on utilise la fonction list. Il faut obligatoirement que les variables utilisées par la fonction list soient définies préalablement.

Fonction list : crée une liste pouvant contenir des variables **Syntaxe**: (list el1 el2 el3 ...)
': caractère apostrophe el1, el2, el3...: éléments de la liste (si l'élément est une variable, celle-ci doit avoir été préalablement définie)

▶ **Exemple**

```
TinyScheme 1.41
ts> (let* ((x 5)) (list 2.5 "jean" x))
(2.5 "jean" 5)
ts>
```

1.6) Créer de nouvelles fonctions

On définit une nouvelle fonction, en utilisant la syntaxe suivante, puis on l'appelle en passant son nom et ses paramètres entre parenthèses.

Définition d'une fonction **Syntaxe**: (define (nom param1 param2 ...) expr)
nom: nom de la fonction param1, param2...: paramètres de la fonction, expr: expression définissant ce que fait la fonction

▶ **Exemple**

```
ts> (define (multiplie x y) (* x y))
multiplie
ts> (multiplie 3 4)
12
ts>
```

Notation lambda

En mathématiques, pour définir une fonction on utilise la notation :

```
x --> f(x)
```

En langage Scheme, on utilise la notation suivante, appelée notation **lambda**. Une fois la fonction définie, on l'utilise en mettant son nom et ses paramètres entre parenthèses

Définition d'une fonction avec la notation lambda **Syntaxe**: (define nomfonc (lambda (x y ...) expr)
nomfonc: nom de la fonction x, y...: variables de la fonction, expr: expression définissant ce que fait la fonction

▸ **Exemple**

Soit la fonction mathématique x --> f(x)= x^2+1

```
TinyScheme 1.41
ts> (define f (lambda (x) (+ (* x x) 1)))
f
ts> (f 2)
5
ts>
```

Soit la fonction mathématique x --> f(x,y)= x+y

```
ts> (define f (lambda (x y) (+ x y)))
f
ts> (f 3 5)
8
ts>
```

La notation lambda peut être utilisée pour créer des fonctions non uniquement mathématiques.

▸ **Exemple**

La notation lambda suivante crée une fonction liste1 qui permet de créer une liste constituée d'un seul élément (celui passé en paramètre à la fonction).

```
ts> (define liste1 (lambda (s) (cons s '())))
liste1
ts> (liste1 'jean)
(jean)
ts>
```

Une fois la fonction liste1 créée, on l'appelle en plaçant son nom et son paramètre entre parenthèses.
Dans le cas où le paramètre représente une chaine de caractère, on le précède d'une apostrophe.

1.7) Utiliser des fonctions intégrées

Le langage Scheme met à la disposition du programmeur de nombreuses fonctions intégrées au langage. Les fonctions suivantes sont souvent utilisées.

1.7.1) Fonctions mathématiques

Nom	Calcul effectué	Nbre de paramètres
abs	valeur absolue	1
acos	arc cosinus	1
asin	arc sinus	1
atan	arc tangente	1
ceiling	partie entière par excès	1
cos	cosinus	1
exp	exponentielle	1
expt	puissance	2
floor	partie entière par défaut	1
log	logarithme	1
modulo	modulo d'une division	2
quotient	division	2
remainder	reste d'une division	2
round	arrondi	1
sin	sinus	1
sqrt	racine carrée	1
tan	tangente	1

On utilise ces fonctions en mettant son nom suivi de son (ou de ses deux) paramètre(s) entre parenthèses. Il est à noter que Scheme exprime les angles en radians.

▶ **Exemple**

```
TinyScheme 1.41
ts> (abs -1)
1
ts> (sin (/ 3.14159 2))
1.0
ts> (define pi 3.14159)
pi
ts> (cos pi)
-1.0
ts> (cos 3.14159)
-1.0
ts>
```

1.7.2) Fonctions sur les chaînes de caractères

La fonction **string-append** permet d'ajouter une chaîne de caractères à une chaîne de caractères existante.

Ajout d'une chaîne de caractères à une chaîne de caractères existante **Syntaxe**: (string-append ch1 ch2)
ch1: chaîne de caractères existante ch2: chaîne de caractères ajoutée à la chaîne existante

▶ **Exemple**

```
ubuntu@ubuntu-MS-7C08:~$ tinyscheme
TinyScheme 1.41
ts> (define x "jean")
x
ts> (string-append x "dupond")
"jeandupond"
ts>
```

La fonction **substring** permet d'extraire une partie d'une chaîne de caractères.

Extraction d'une partie d'une chaîne de caractères **Syntaxe**: (substring ch start end)
ch: chaîne de caractères start: n° du caractère de départ pour l'extraction (en partant de 0) end: n° du caractère de fin (exclu) pour l'extraction (en partant de 0)

▶ **Exemple**

```
ubuntu@ubuntu-MS-7C08:~$ tinyscheme
TinyScheme 1.41
ts> (define x "bonjour jean!")
x
ts> (substring x 3 6)
"jou"
ts>
```

Le mot **string?** teste si un objet est une chaîne de caractères. Le mot **string=?** teste si deux chaînes de caractères sont égales.

▶ **Exemple**

```
ubuntu@ubuntu-MS-7C08:~$ tinyscheme
TinyScheme 1.41
ts> (define x "bonjour")
x
ts> (string? x)
#t
ts> (define y "bonjour")
y
ts> (string=? x y)
#t
ts>
```

1.8) Utiliser des expressions conditionnelles

1.8.1) Expressions booléennes

En langage Scheme, les valeurs booléennes true et false s'écrivent respectivement **#t** et **#f**.

Pour tester des nombres, on utilise les symboles **>**, **<**, **<=**, **>=**, **=** et **zero?**

▶ **Exemple**

```
TinyScheme 1.41
ts> (> 5 3)
#t
ts> (> 2 5)
#f
ts> (= 2 2)
#t
ts> (= 4 5)
#f
ts> (zero? (- 2 2))
#t
ts> (zero? (- 2 3))
#f
```

Pour tester des chaînes de caractères et des listes on utilise les symboles **eqv?** et **null?**

▶ **Exemple**

```
TinyScheme 1.41
ts> (eqv? 'max 'max)
#t
ts> (eqv? 'max 'jean)
#f
ts> (null? '())
#t
ts> (null? '(jean))
#f
ts>
```

Pour tester si une expression est un nombre ou une chaîne de caractères, on utilise les symboles **number?** et **string?**

▶ ***Exemple***

```
TinyScheme 1.41
ts> (define a "jean")
a
ts> (string? a)
#t
ts> (number? a)
#f
ts>
```

▸ **Exemple**

```
TinyScheme 1.41
ts> (define a (+ 3 2))
a
ts> a
5
ts> (string? a)
#f
ts> (number? a)
#t
ts>
```

1.8.2) Expression if

Pour exécuter une expression conditionnelle à choix simple, on utilise le mot clef **if**.

Exécution d'une instruction conditionnelle à choix simple **Syntaxe**: (if expr0 expr1 expr2)
expr0: expression indiquant le test effectué expr1: expression exécutée si le résultat du test est #t (true) expr2: expression exécutée si le résultat du test est #f (false)

Si l'expression expr0 est vraie **alors** l'expression expr1 est éxécutée **sinon** l'expression expr2 est exécutée.

▸ **Exemple**

```
ts> (define a 3)
a
ts> (define b 2)
b
ts> (if (>= a b) (- a b) (- b a))
1
ts> (define a 2)
a
ts> (define b 3)
b
ts> (if (>= a b) (- a b) (- b a))
1
ts>
```

Les symboles **and** et **or** permettent de créer des expressions de test plus complexes.

▸ **Exemple**

```
ts>(define a 2)
a
ts>(if (or (= a 2) (= a 3)) (+ a a))
4
ts>(define a 3)
a
ts>(if (or (= a 2) (= a 3)) (+ a a))
6
```

1.8.3) Expression cond

Pour exécuter une expression conditionnelle à choix multiples, on utilise le mot clef **cond**.

Exécution d'une instruction conditionnelle à choix multiples **Syntaxe**: (cond (t1 a1) (t2 a3) (t3 a3)... else a)
t1, t2, t3,...: expressions indiquant les tests effectués a1, a2, a3,...: expression exécutée si le résultat du test correspondant t1, t2, t3,… est #t (true) a: expression exécutée par défaut si aucun des résultats des tests précédents est #t (true)

▶ **Exemple**

```
ts> (define a 2)
a
ts> (define b 3)
b
ts> (cond ((< a b)(- b a)) ((> a b)(- a b)) ((= a 0)(+ a 1)) (else 1))
1
ts> (define a 3)
a
ts> (define b 2)
b
ts> (cond ((< a b)(- b a)) ((> a b)(- a b)) ((= a 0)(+ a 1)) (else 1))
1
ts> (define a 0)
a
ts> (define b 0)
b
ts> (cond ((< a b)(- b a)) ((> a b)(- a b)) ((= a 0)(+ a 1)) (else 1))
1
ts> (define a 2)
a
ts> (define b 2)
b
ts> (cond ((< a b)(- b a)) ((> a b)(- a b)) ((= a 0)(+ a 1)) (else 1))
1
```

⚑ **Remarque**

Pour écrire une expression qui constitue une séquence ordonnée d'autres expressions, on peut utiliser le mot **begin**.

▶ **Exemple**

On écrit puis on exécute, à l'aide du Terminal de commandes, le fichier programme "test.scm" suivant

```
(define x 1)
(if (= x 1) (begin (display "x =1 ") ↯
             (newline)(display (+ x 1))(newline) ))

ubuntu@ubuntu-MS-7C08:~$ scm -f test1.scm
x =1
2
ubuntu@ubuntu-MS-7C08:~$
```

1.9) Utiliser des entrées et sorties

1.9.1) Sortie d'information

Pour afficher des informations dans la **fenêtre de la console**, on utilise l'instruction **display**. Après l'affichage l'interpréteur affiche #t, à la suite de l'affichage, si celui-ci s'est bien passé.

Affichage d'information dans la fenêtre de la console **Syntaxe**: (display inf)
inf: information afichée (variable, valeur numérique, chaine de caractères)

▶ **Exemple**

```
TinyScheme 1.41
ts> (define x 5)
x
ts> (display x)
5#t
ts> (define s "jean")
s
ts> (display s)
jean#t
ts>
```

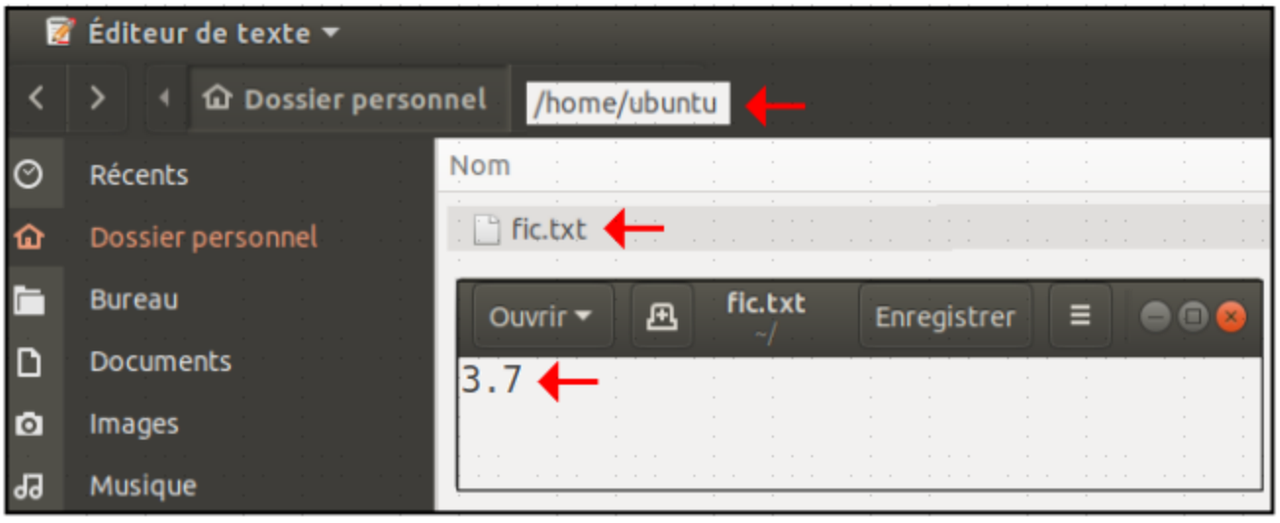
Pour afficher des informations dans un **fichier texte**, on commence par ouvrir le fichier, l'aide de l'instruction **open-output-file**, puis on utilise l'instruction **display**, en la dirigeant vers le fichier ouvert, et enfin on ferme le fichier avec l'instruction **close-port**.

Ces différentes instructions s'enchainent de la façon suivante et, sous Ubuntu, "fic.txt" est enregistré dans "/home/ubuntu".

```
(define x 3.7)
(define out (open-output-file "fic.txt"))
(display x out)
(close-port out)
```

▶ **Exemple**

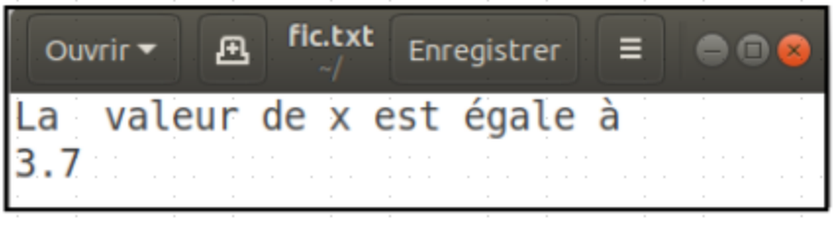
```
TinyScheme 1.41
ts> (define x 3.7)
x
ts> (define out (open-output-file "fic.txt"))
out
ts> (display x out)
#t
ts> (close-port out)
#t
ts>
```



On peut écrire plusieurs instructions display, les unes à la suite des autres, et faire des retours à la ligne, à l'aide de l'instruction **newline**.

▶ **Exemple**

```
ts> (define x 3.7)
x
ts> (define out (open-output-file "fic.txt"))
out
ts> (display "La  valeur de x est égale à " out)
#t
ts> (newline out)
#t
ts> (display x out)
#t
ts> (close-port out)
#t
ts>
```



1.9.2) Entrée d'information

Pour pouvoir entrer des information depuis le **clavier**, on peut utiliser l'instruction **read** (avec tinyscheme depuis le Terminal de commandes).

Entrée de la valeur d'une variable à l'aide du clavier **Syntaxe**: (define x (read))
x: variable dans laquelle est placée la valeur lue au clavier

▶ **Exemple**

```
TinyScheme
ts> (define x (read))
7.4  (valeur saisie au clavier)
x
ts> (display x)
7.4#t
ts>
ts> (define x (read))
jean
x
ts> (display x)
jean#t
ts>
```

Pour lire des informations depuis un **fichier texte**, on peut utiliser l'instruction **read** (avec tinyscheme depuis le Terminal de commandes ou avec la console script-fu).

Lecture dans un fichier texte **Syntaxe**: (define x (read in))
in: flux d'entrée du fichier lu, défini à l'aide de l'instruction open-input-file

On commence par définir un flux d'entrée pour le fichier, l'aide de l'instruction **open-output-file**, puis on utilise l'instruction r**ead** pour lire séquentiellement dans le fichier puis, à la fin, on ferme le fichier à l'aide de l'instruction **close-input-port**.

▶ **Exemple**

Le présent exemple lit, puis affiche dans la console, les valeurs 1.1, 2.2 et 3.3 contenues au début des lignes 1,2 et 3 du fichier fic.txt.

```
TinyScheme
ts> (define in (open-input-file "fic.txt"))
in
ts> (define x (read in))
x
ts> (display x)
1.1#t
ts> (define y (read in))
y
ts> (display y)
2.2#t
ts> (define z (read in))
z
ts> (display z)
3.3#t
ts> (close-input-port in)
#t
ts>
```

1.10) Créer des programmes Scheme

Jusqu'à présent, on a exécuté des instructions Scheme dans la fenêtre de la **console Script-Fu** de Gimp ou dans la fenêtre du **Terminal de commandes** (sous l'invite scm de scm ou ts de tinyscheme).

Si on n'utilise pas de fonctions spécifique à **Gimp**, on peut regrouper les instructions **scheme** dans un fichier programme doté de l'extension scm ("test.scm" par exemple). On peut ensuite exécuter ce programme avec **tinyscheme** ou avec **scm**, depuis le Terminal de commande. Pour cela, on utilise l'une des commandes suivantes, selon l'interpréteur utilisé.

Interpréteur tinyscheme : `tinyscheme test.scm`

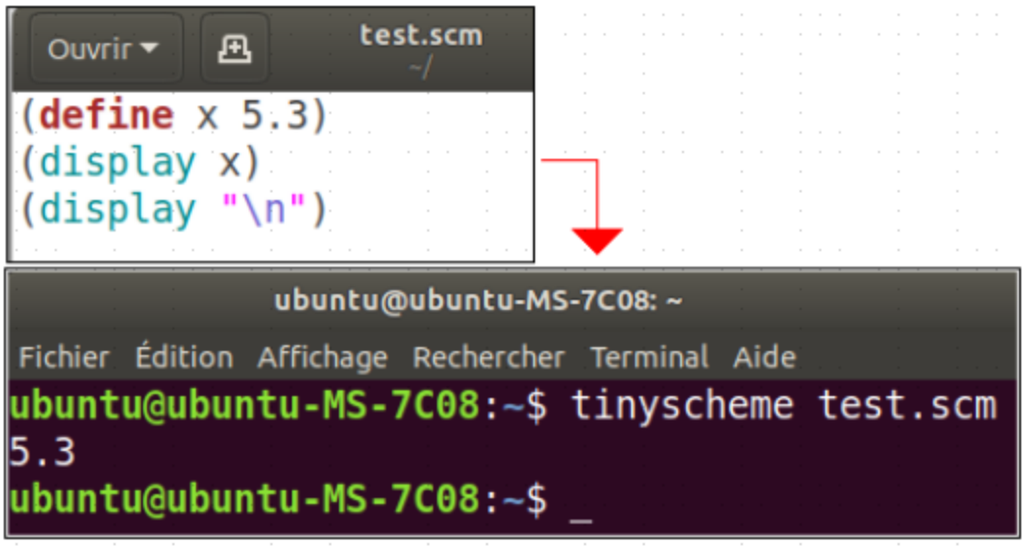
Interpréteur scm : `scm -f test.scm`

▶ **Exemple 1**

test.scm

```
(define x 5.3)
(display x)
(display "\n")
```

```
ubuntu@ubuntu-MS-7C08:~$ tinyscheme test.scm
5.3
```



▶ **Exemple 2**

test.scm

```
(display "Saisissez un nombre x -> ")
(define x (read))
(display "x = ")
(display x)
(newline)
```

```
ubuntu@ubuntu-MS-7C08:~$ scm -f test.scm
Saisissez un nombre x -> 4 (valeur saisie au clavier)
x = 4
ubuntu@ubuntu-MS-7C08:~$
```

▶ **Exemple 3**

Ce programme effectue l'adition de deux nombres x et y saisis au clavier et affiche le résultat dans la fenêtre du Terminal.
La première ligne est une ligne de commentaire. Elle n'est pas prise en compte par l'interpréteur scm car elle commence par le caractère ;

test.scm

```
;-------------- Addition-------------
(display "Saisissez un nombre x -> ")
(define x (read))
(display "Saisissez un nombre y -> ")
(define y (read))
(define z (+ x y))
(display "x+y = ")
(display z)
(newline)
```

```
ubuntu@ubuntu-MS-7C08:~$ scm -f test.scm
Saisissez un nombre x -> 4
Saisissez un nombre y -> 5
x+y = 9
ubuntu@ubuntu-MS-7C08:~$ ⚑
```

⚑ **Remarque**

En langage Scheme, les lignes de commentaire doivent commencer par un point virgule.

```
; ceci est un commentaire
```

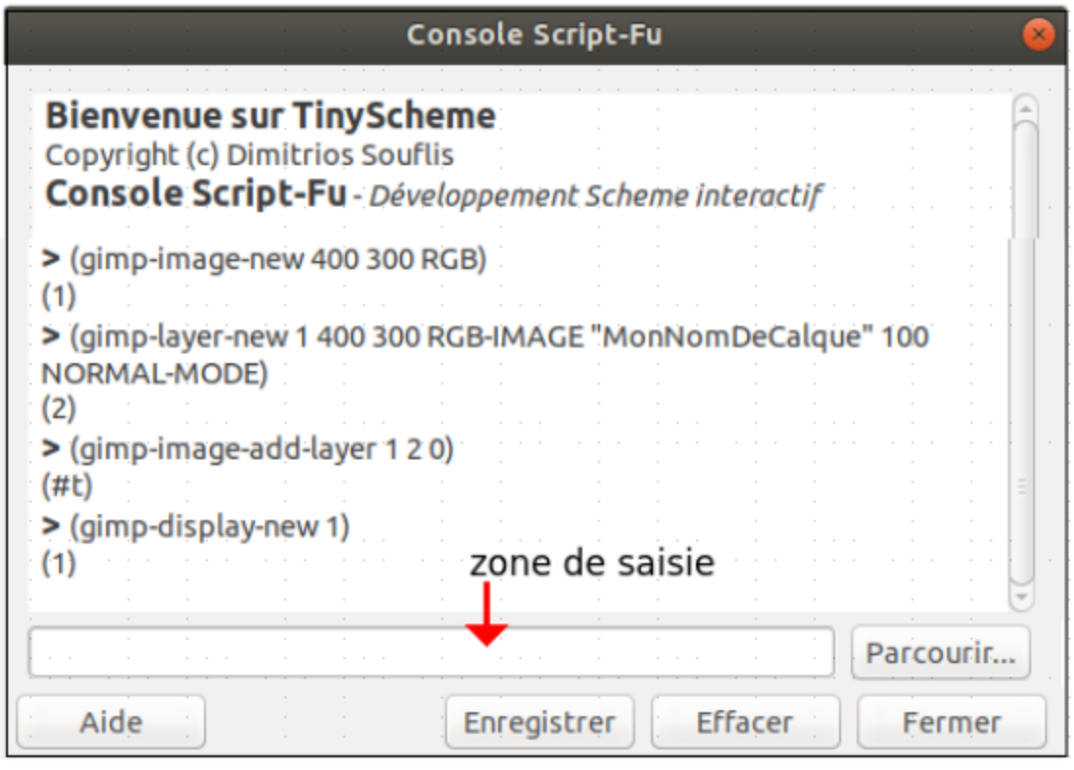
Chapitre II
Créer des scripts Gimp

Pour créer des scripts en **Gimp**, on utilise le langage **Script-Fu**. C'est un langage de programmation interprété basé sur le langage **TinyScheme** (langage dérivé du langage **Scheme** qui est lui même dérivé du langage **Lisp** - List Processing). Le langage Script-Fu fait partie du noyau de Gimp. Comme le langage Lisp, il dispose d'une syntaxe en forme de listes encadrées par des parenthèses.

Un script Gimp, également appelé Script-Fu, est un simple fichier texte doté de l'extension **scm**. Ce fichier peut être créé avec n'importe quel éditeur de texte. Il est composé d'instructions dont le nombre varie selon la nature du traitement à effectuer. Chacune d'elles peut être testée séparément, ou regroupée avec d'autres, à l'aide de la console Script-Fu de Gimp. Cette possibilité permet de repérer facilement les erreurs de syntaxe présentes au sein d'un script.

2.1) Tester avec la console Script-Fu de Gimp

On lance Gimp, on ouvre la console Script-Fu depuis le menu **Filtres > Script-Fu > Console** de Gimp puis on saisit les instructions suivantes, une par une, dans la zone de saisie de la console Script-Fu.

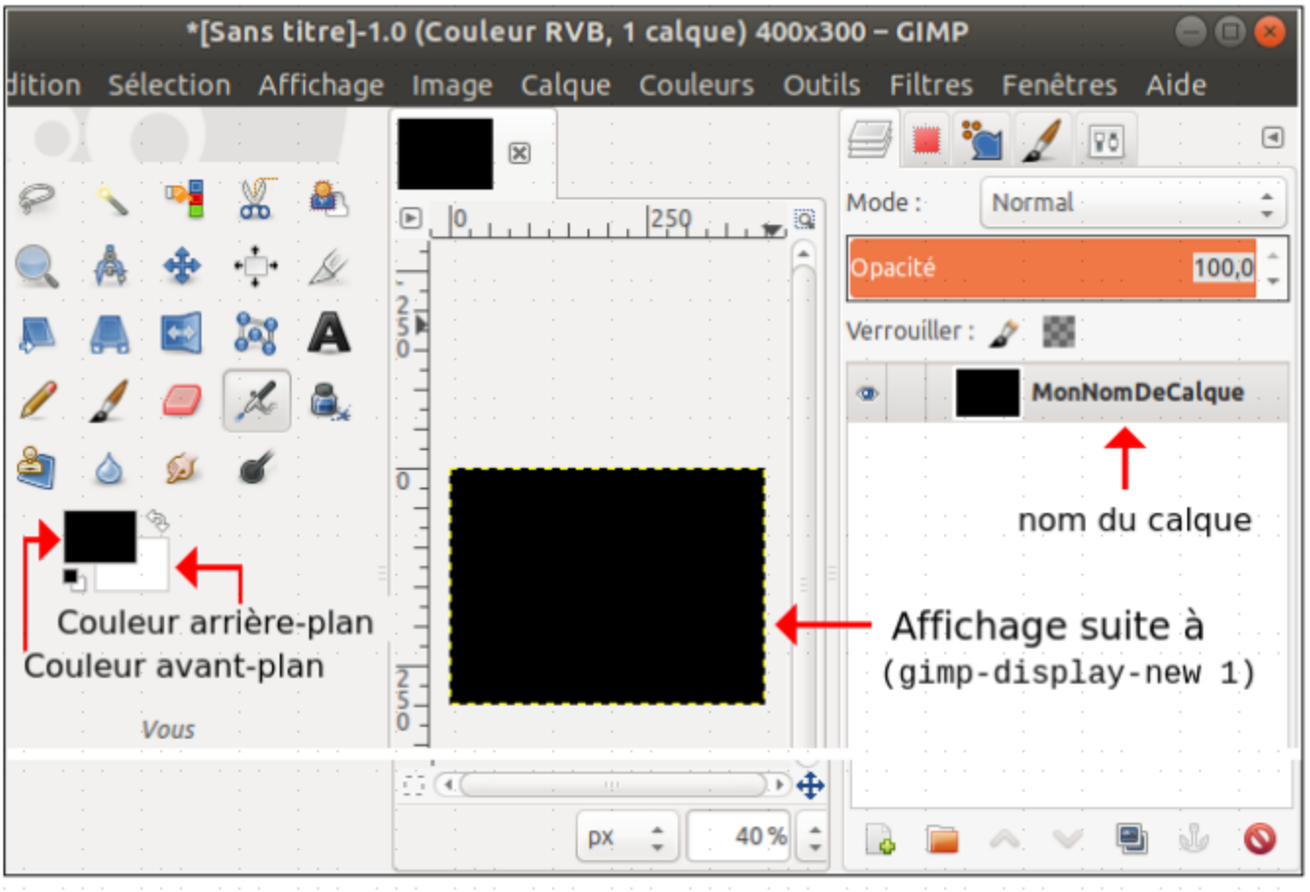


```
(gimp-image-new 400 300 RGB)
```
Cette instruction crée une nouvelle image de taille 400x300 pixels en mémoire. La valeur 1, retournée par la console script-fu, représente le numéro d'identification (id) de cette image en mémoire qui, à ce stade, n'est pas visible à l'écran.

```
(gimp-layer-new 1 400 300 RGB-IMAGE "MonNomDeCalque" 100 NORMAL-MODE)
```
Cette instruction crée un calque de taille 400x300 pixels en mémoire et lui attribue le nom «MonNomDeCalque ». Ce calque est nécessaire car l'image doit être associée à un calque. La valeur 2, retournée par la console script-fu, représente le numéro d'identification de ce calque en mémoire qui, à ce stade, n'est pas non plus visible à l'écran. Le chiffre 100 réprésente le degré d'opacité souhaité pour le calque.

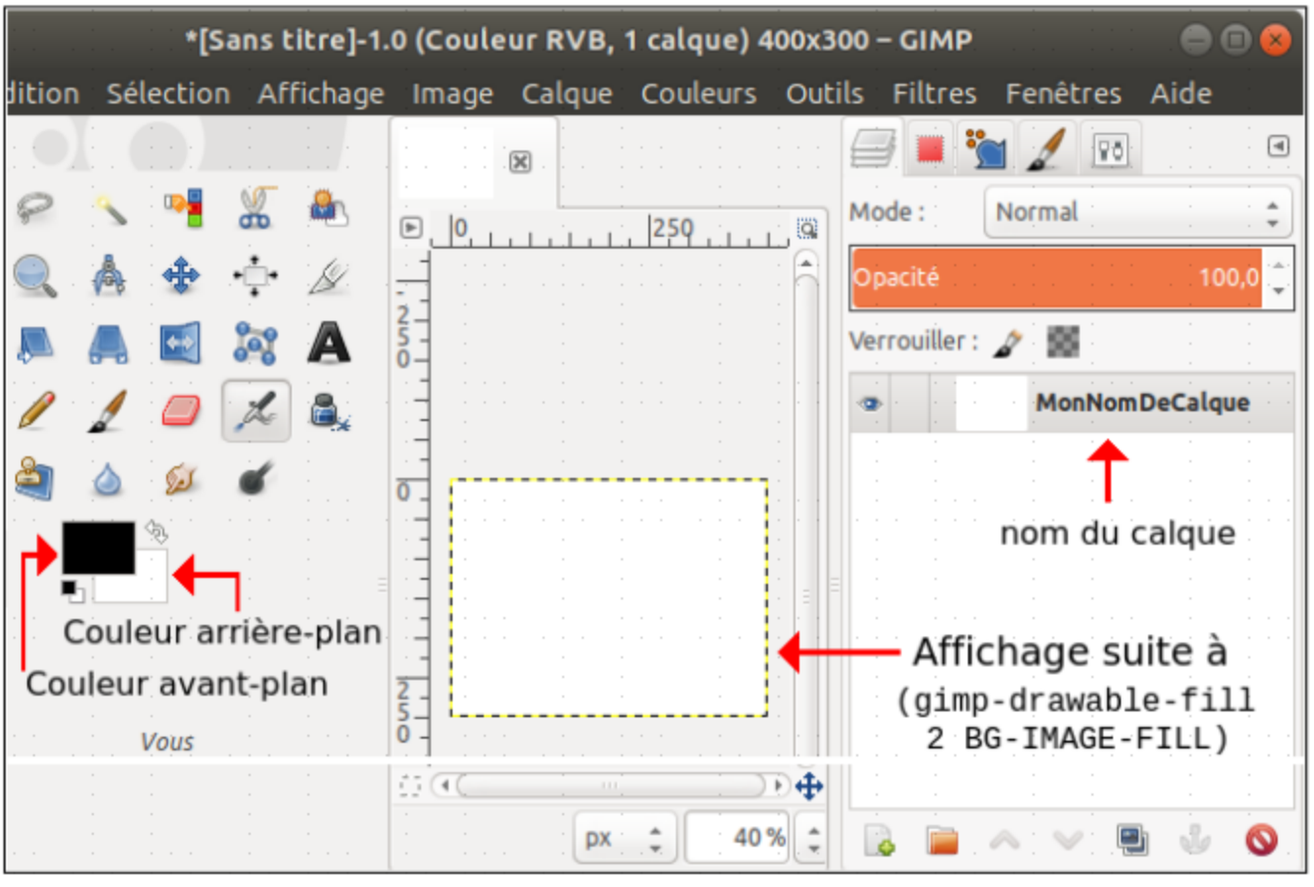
```
(gimp-image-add-layer 1 2 0)
```
Cette instruction place le calque d'id 2 dans l'image d'id 1, en mémoire. La valeur #t, retournée par la console script fu, signifie "true" (vrai). Cela signifie que l'opération s'est déroulée avec succès. Á ce stade, l'image et le calque existent en mémoire mais ne sont toujours pas visibles à l'écran. Le chiffre 0 indique la position du calque dans la pile de calques.

```
(gimp-display-new 1)
```
Cette instruction affiche l'image d'id 1 dans la fenêtre principale de Gimp, avec la couleur d'avant plan en cours pour le calque associé. La console script-fu retourne l'id de l'image en mémoire, c'est à dire 1.



```
(gimp-drawable-fill 2 BG-IMAGE-FILL)
```

Cette instruction remplit le calque d'id 2 avec la couleur d'arrière plan en cours (blanc dans notre exemple). La valeur #t retournée par la console script-fu signifie signifie que l'opération s'est déroulée avec succès.



2.2) Consulter la documentation intégrée

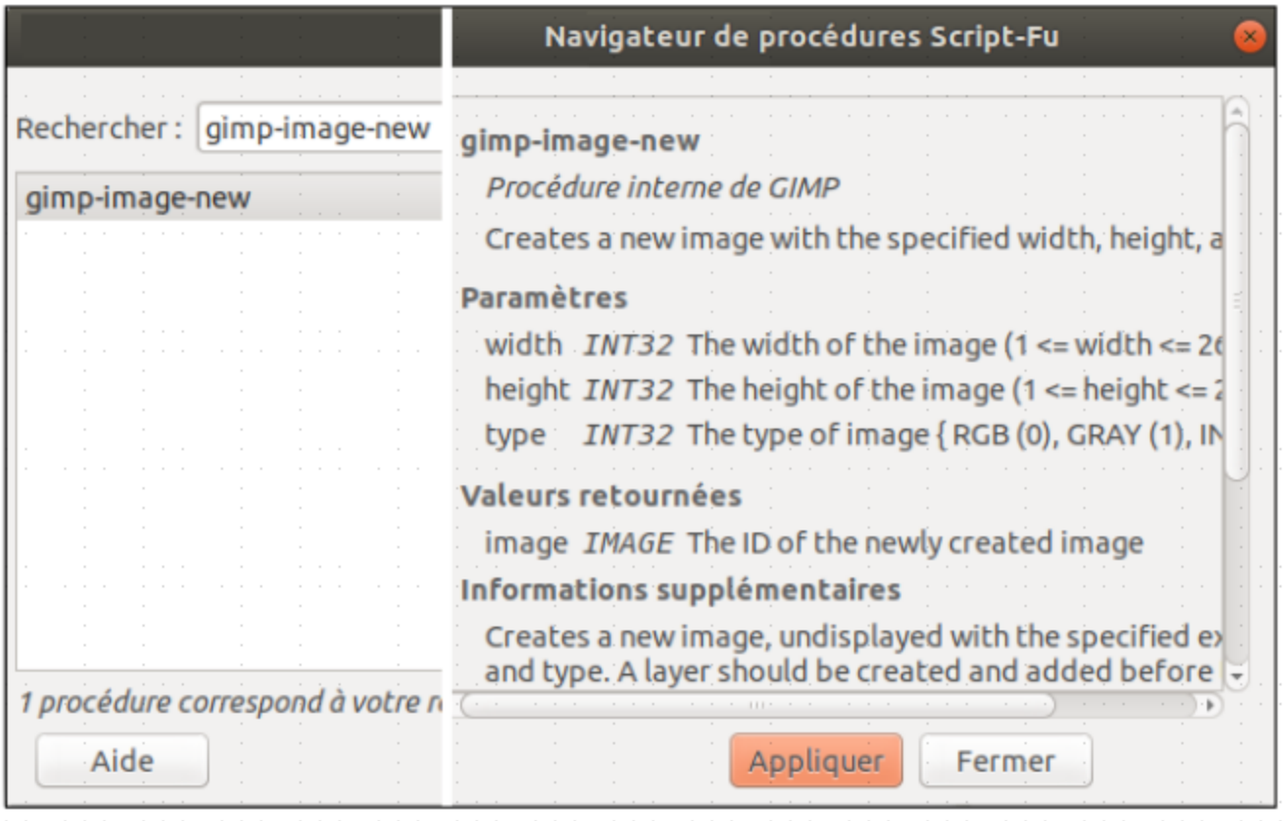
La documentation intégrée, permet d'approfondir toutes les instruc-tions Script-Fu disponibles ainsi que leur syntaxe. Elle est accesssible en cliquant le bouton **Parcourir...** de la Console Script-Fu.
On peut ainsi obtenir des informations précieuses sur toutes les fonctions utilisées par Gimp.

Par exemple, l'instruction **img gimp-image-new** permet de créer une nouvelle image en mémoire.

```
Crée une nouvelle image en mémoire
Syntaxe: (img gimp-image-new w h t)

Paramètres
w INT32: Largeur de l'image en pixels (comprise entre 1 et 262144)
h INT32: Hauteur de l'image en pixels (comprise entre 1 et 262144)
t INT32: type de l'image (0 pour RGB, 1 pour nuances de gris et 3
         pour indexée)
```

Valeur retournée
img IMAGE: identificateur (id) de la nouvelle image, créée en mémoire

Une fois créée en mémoire, l'image n'est pas affichée à l'écran. Un calque devra être créé puis atribué à cette image pour que celle-ci puisse être affichée sinon les appels suivants à "gimp-display-new", avec cette image comme argument, échoueront. Les calques peuvent être créés en utilisant la fonction "gimp-layer-new". Ils peuvent être attribués à une image en utilisant la fonction "gimp-image-insert-layer" ("gimp-image-add-layer" devient une fonction obsolète).



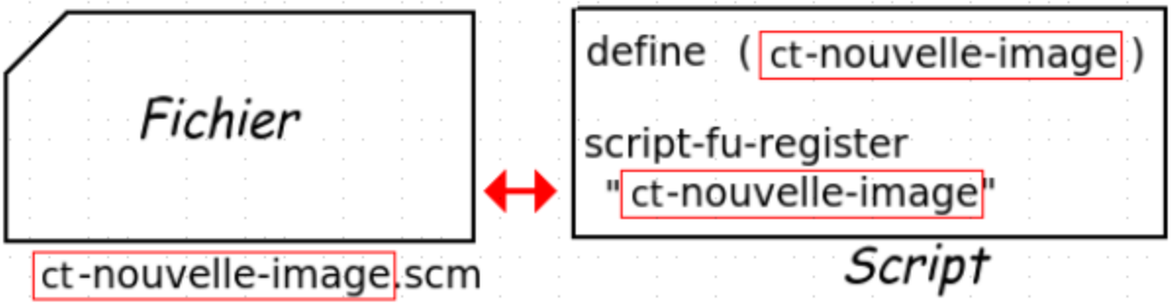
2.3) Créer un premier script simple pour Gimp

Il est possible et naturel de regrouper dans un script l'ensemble des instructions décrites précédemment. Ce script peut ensuite être éxécuté automatiquement autant de fois que l'on souhaite sans avoir à reécrire les instructions une par une.

On commence par écrire le premier script simple pour Gimp suivant, à l'aide de n'importe quel éditeur de texte. Puis on l'enregistre avec le nom "ma-nouvelle-image.scm" (par exemple). On verra en 2.4) comment installer ce fichier afin que le script correspondant puisse être utilisé et fonctionner depuis Gimp.

⚑ **Remarque**
Pour permettre la prise en compte d'un script par GIMP, dans sa base de données de scripts (Procedure Data Base – notée PDB), le nom du script doit obligatoirement être identique au nom du fichier.

Si le nom du fichier est "ct-nouvelle-image.scm" par exemple, le nom du script, dans le programme, doit être "ct-nouvelle-image".



▸ ***Exemple***

Ce script simple est enregistré dans le fichier "ct-nouvelle-image.scm". Il permet de créer et afficher puis enregistrer, avec le nom "sauv.jpg", une nouvelle image de couleur blanche, de taille 400x300 pixels.

Les fonctions utilisées seront examinées plus en détail par la suite. Ici on regarde uniquement comment écrire, installer et exécuter un script scm pour gimp.

ct-nouvelle-image.scm

```
(
define (ct-nouvelle-image)
(let* ( (img (car (gimp-image-new 400 300 RGB )))
(c (car (gimp-layer-new img 400 300 RGB-IMAGE "c1" 100 NORMAL-MODE))))
(gimp-context-set-background  '(255 255 255))
(gimp-image-insert-layer img c 0 0)
(gimp-drawable-fill c BACKGROUND-FILL)
(gimp-display-new img)
(gimp-file-save  RUN-NONINTERACTIVE img c ↯
          "/home/ubuntu/essai/sauv.jpg" "")
))
(
script-fu-register
"ct-nouvelle-image"
"<Image>/MesScripts/ct-nouvelle-image"
"Mon commentaire"
"Essai de script Gimp"
"Claude Turrier"
"2016"
"----------"
)
```

⚑ **Installation sous Linux**

```
sudo cp ct-nouvelle-image.scm /usr/share/gimp/2.0/scripts/
```

La première partie du script, placée entre parenthèses, commence par une instruction **define** qui définit le nom du script (celui-ci doit correspondre au nom du fichier .scm du script). Puis suit l'ensemble des instructions qui constituent le script à exécuter.

La seconde partie du script, elle aussi entre parenthèses, permet d'enregistrer le script afin que Gimp puisse le prendre en compte.

`script-fu-register`
Cette instruction est nécessaire pour que Gimp enregistre le script.

`"ct-nouvelle-image"`
Cette ligne sert à enregistrer le nom du script dans la base de données de scripts de Gimp (Procedure Data Base - PDB)

`<Image>/MesScripts/ct-nouvelle-image`
Cette instruction fait apparaître une option de menu "MesScripts/ct-nouvelle-image" dans la barre de menu de Gimp. La balise <image> est nécessaire.

Puis viennent 5 lignes de commentaires dont la présence est obligatoire pour l'enregistrement du script.

```
"Mon commentaire"
"Essai de script Gimp"
"Claude Turrier"
"2016"
"-----------"
```

Une fois le script créé et enregistré dans un fichier doté de l'extension scm, il ne reste plus qu'à le copier dans le répertoire des scripts de Gimp (cf 2.4)

2.4) Installer un script pour Gimp

Pour installer un script pour Gimp (fichier "ma-nouvelle-image.scm", par exemple), il suffit de copier ce fichier dans le répertoire des scripts de Gimp.

• Installation sous Windows
Sous Windows, le répertoire des scripts de Gimp se trouve généralement à l'emplacement suivant (pour le chemin d'accès, on utilise le caractère antislash "\", barre oblique inversée):

```
C:\Program Files\GIMP2\lib\gimp\2.0\share\gimp\2.0\scripts
```

Sous Windows, on peut installer le fichier script dans ce répertoire à l'aide d'un simple copier-coller. On peut le supprimer en le sélectionnant puis en appuyant la touche clavier Suppr, depuis l'Explorateur de fichiers.

• Installation sous Linux

Pour installer un script pour Gimp, il suffit de l'installer dans le répertoire de Gimp dédiés aux scripts.

Sous **Linux**, le répertoire des scripts se trouve généralement à l'emplacement suivant (pour le chemin d'accès, on utilise le caractère slash "/", barre oblique) :

```
/usr/share/gimp/2.0/scripts/
```

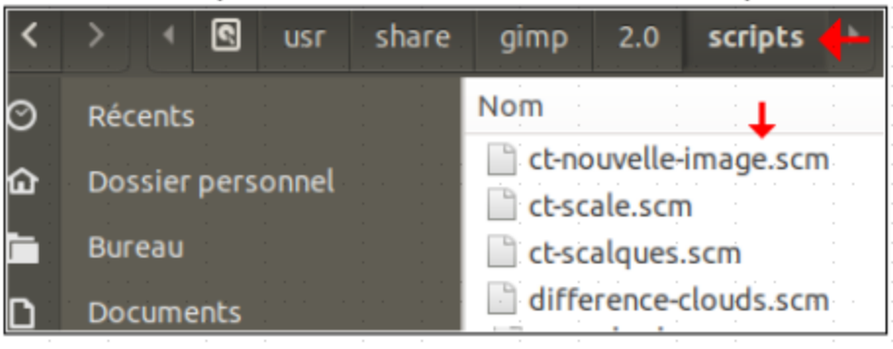
Linux est particulièrement bien sécurisé. On ne peut pas installer un script à l'aide d'un simple copier-coller comme sous Windows. Il faut saisir la commande suivante, depuis le Terminal de commandes ouvert dans le répertoire où se trouve le fichier script à installer.

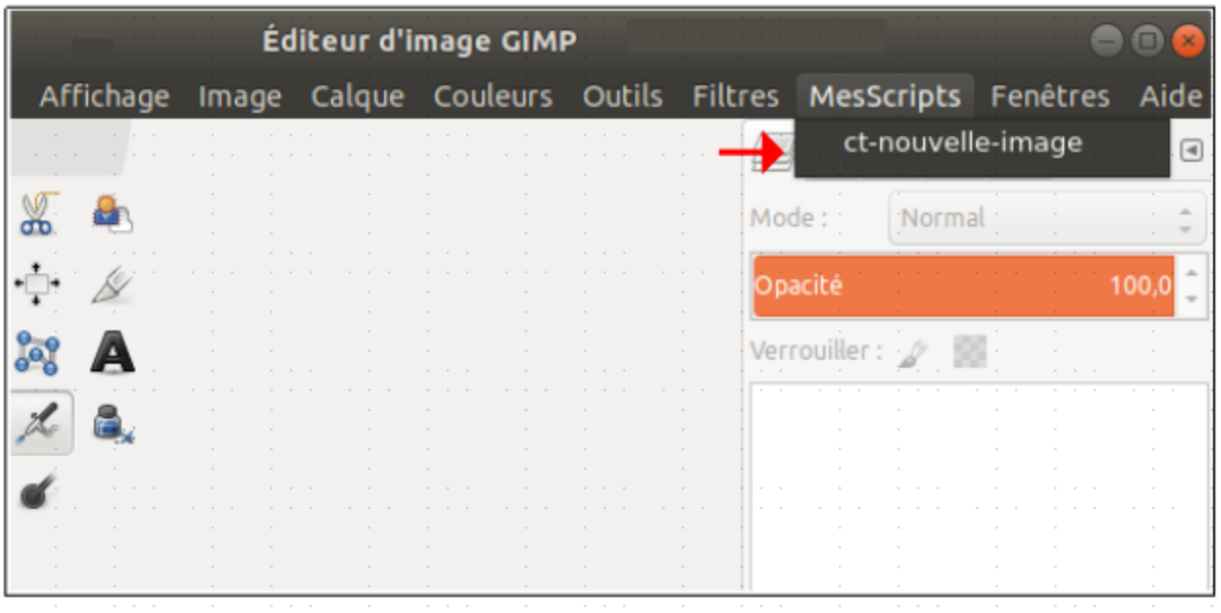
```
sudo cp ct-nouvelle-image.scm /usr/share/gimp/2.0/scripts/
```

⚑ **Remarque**
Sous Linux, on a parfois besoin de rendre un fichier exécutable. Pour cela on utilise la commande suivante qui n'est pas nécessaire pour les scripts scm Gimp: sudo chmod +x /home/ubuntu/essai/monfichier

Sous **Windows**, on installe simplement le fichier script à l'aide d'un simple copier-coller, depuis l'Explorateur de fichiers.

Une fois le script copié dans le répertoire des scripts de Gimp on peut l'utiliser directement depuis la barre de menu de Gimp.





• Suppression sous **Linux**

Sous Linux, pour supprimer un script (le script "ma-nouvelle-image.scm", par exemple), il suffit d'ouvrir le Terminal de commandes dans le répertoire des scripts de Gimp puis de saisir la commande suivante :

```
sudo rm  ct-nouvelle-image.scm
```

2.5) Améliorer un script Gimp

Le script précédent permet de créer une nouvelle image uniquement avec une taille fixe 400x300. Il serait utile de le modifier afin de pouvoir créer des images de tailles variables.

Cela peut se faire à l'aide d'une boîte de dialogue permettant de saisir les valeurs à prendre en compte pour la largeur et pour la hauteur de l'image à créer.

Pour cela, il suffit de généraliser la procédure de script précédente en plaçant, après son nom, deux paramètres (w et h par exemple) associés à la largeur et à la hauteur de l'image.

Ces paramètres, placés après le nom de la procédure de script, conduisent à l'ouverture automatique d'une boîte de dialogue, au moment de l'éxécution du script. Cette boîte de dialogue permet de saisir les valeurs de ces paramètres.

```
define (ct-nouvelle-image w h)
```

On peut ensuite utiliser ces paramètres comme on le souhaite dans le script mais il faut obligatoirement les enregistrer, dans la fonction register, à l'aide du mot clef **SF-VALUE** (qui accepte les nombres), en leur donnant une valeur qui sera proposée par défaut.

```
SF-VALUE "w" "400"
SF-VALUE "h" "300"
```

▶ **Exemple**

ct-nouvelle-image.scm

```
(
define (ct-nouvelle-image w h)
(let* ( (img (car (gimp-image-new w h RGB )))
(c (car (gimp-layer-new img w h RGB-IMAGE "c1" 100 NORMAL-MODE))))
(gimp-context-set-background  '(255 255 255))
(gimp-image-insert-layer img c 0 0)
(gimp-drawable-fill c BACKGROUND-FILL)
(gimp-display-new img)
(gimp-file-save  RUN-NONINTERACTIVE img c  ↯
        "/home/ubuntu/essai/sauv.jpg" "")
))
```

```
(
script-fu-register
"ct-nouvelle-image"
"<Image>/MesScripts/ct-nouvelle-image"
"Mon commentaire"
"Essai de script Gimp"
"Claude Turrier"
"2016"
"-----------"
SF-VALUE "w" "400"
SF-VALUE "h" "300"
)
```

⚑ **Installation sous Linux**

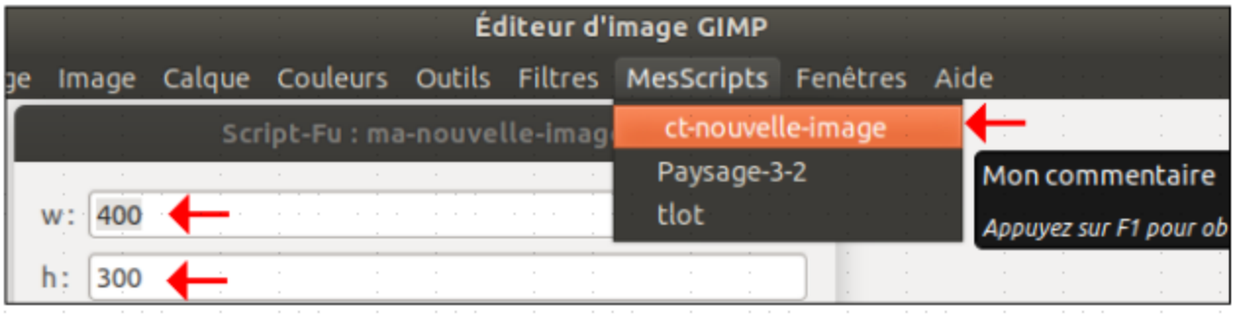
```
sudo cp ct-nouvelle-image.scm /usr/share/gimp/2.0/scripts/
```

Les modifications effectuées au sein de ce script, par rapport à la version précédente, sont les suivantes :

1) Insertion des paramètres w et h dans l'instruction «define» qui définit le nom du script (cela va conduire, à l'ouverture automatique d'une boîte de dialogue permettant de saisir leurs valeurs) ;
2) Insertion de ces paramètres dans les instructions gimp-image-new et gimp-layer-new qui les utilisent ;
3) Insertion, à la fin de la partie script-fu-register du script, des indications SF-VALUE "w" "400" et SF-VALUE "h" "300" (valeurs présentées par défaut dans la boîte de dialogue).

L'insertion, à la fin de la partie script-fu-register du script, des indications SF-VALUE "maLargeur" "400" et SF-VALUE "maHauteur" "300" permet :

1) l'enregistrement dans l'ordre des paramètres w et h déclarés dans l'instruction «define» ;
2) la prise en compte de ces paramètres en tant que valeurs numériques
3) l'ouverture automatique, lors du lancement du script, d'une boîte de dialogue qui permet de saisir les valeurs de largeur h et de hauteur w utilisées par le script pour l'image ;
4) l'initialisation de la boîte de dialogue avec les valeur 400 et 300 pour ces paramètres.



2.5.1) gimp-layer-new

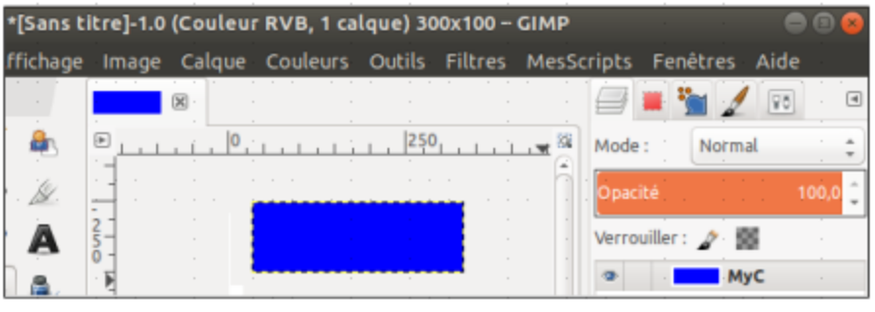
L'instruction **gimp-layer-new** crée un nouveau calque en mémoire. Une fois ce calque créé en mémoire, il devra ensuite être explicitement ajouté à une image présente en mémoire (avec l'instruction gimp-image-insert-layer) car cet ajout n'est pas automatique.

```
Crée un nouveau calque
Syntaxe: (img gimp-layer-new img w h t n o m)
```

```
Paramètres

img IMAGE: identifiant de l'image à laquelle on ajoute le calque
w INT32: Largeur du calque (comprise entre 1 et 262144)
h INT32: Hauteur du calque (comprise entre 1 et 262144)
t INT32: type du calque { RGB-IMAGE (0), RGBA-IMAGE (1), GRAY-IMAGE (2),
         GRAYA-IMAGE (3), INDEXED-IMAGE (4), INDEXEDA-IMAGE (5) }
n STRING: nom du calque
o FLOAT: opacité du calque  (0 <= opacité <= 100)
m INT32: mode de combinaison du calque
  { NORMAL-MODE (0), DISSOLVE-MODE (1), BEHIND-MODE (2), MULTIPLY-MODE
  (3), SCREEN-MODE (4), OVERLAY-MODE (5), DIFFERENCE-MODE (6), ADDITION-
  MODE (7), SUBTRACT-MODE (8), DARKEN-ONLY-MODE (9), LIGHTEN-ONLY-MODE
  (10), HUE-MODE (11), SATURATION-MODE (12), COLOR-MODE (13), VALUE-MODE
  (14), DIVIDE-MODE (15), DODGE-MODE (16), BURN-MODE (17), HARDLIGHT-
  MODE (18), SOFTLIGHT-MODE (19), GRAIN-EXTRACT-MODE (20), GRAIN-MERGE-
  MODE (21), COLOR-ERASE-MODE (22), ERASE-MODE (23), REPLACE-MODE (24),
  ANTI-ERASE-MODE (25) }

Valeur retournée
c LAYER: identifiant du calque créé
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(define img (car (gimp-image-new 300 100 RGB )))
(define c (car (gimp-layer-new img 300 100 RGB-IMAGE ↯
          "MyC" 100 NORMAL-MODE)))
(gimp-context-set-background  '(0 0 255))
(gimp-image-insert-layer img c 0 0)
(gimp-drawable-fill c BG-IMAGE-FILL)
(gimp-display-new img)
(1)
```



⚑ **Remarque**
L'instruction de création d'une nouvelle image en mémoire **gimp-image-new** a été détaillée ci-dessus, en 2.2.

2.5.3) gimp-context-set-background

La fonction **gimp-context-set-background** fixe la couleur d'arrière-plan courante.

Fixer la couleur d'arrière-plan courante **Syntaxe**: (img gimp-context-set-background b)
Paramètres b COLOR: couleur d'arrière plan '(r g b) avec r, g et b compris entre 0 et 255 inclus

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (gimp-context-set-background  '(0 0 255))
(#t)#<EOF>
```

2.5.4) gimp-image-insert-layer

La fonction **gimp-image-insert-layer** ajoute un calque à l'image dont l'identifiant est passé en paramètre.

Ajouter un calque à une image **Syntaxe**: (gimp-image-insert-layer img c pa po)
Paramètres img IMAGE: identifiant de l'image à laquelle le calque est ajouté c LAYER: identifiant du calque ajouté à l'image pa LAYER:identifiant du calque parent au calque ajouté po INT32: position du calque ajouté

⚑ **Remarque**
Si le parent est 0, le calque est ajouté à l'intérieur de la pile des calques. Le paramètre po spécifie l'emplacement du calque à l'intérieur de cette pile, en commençant par le haut (0) et en augmentant.

▶ ***Exemple***

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(let* ( (img (car (gimp-image-new 300 100 RGB )))
(c (car (gimp-layer-new img 300 100 RGB-IMAGE ↯
   "MyC" 100 NORMAL-MODE))))
```

```
  (gimp-context-set-background  '(0 0 255))
  (gimp-image-insert-layer img c 0 0)
  (gimp-drawable-fill c BG-IMAGE-FILL)
  (gimp-display-new img)
  )
```

2.5.4) gimp-drawable-fill

L'instruction **gimp-drawable-fill** remplit un calque dessinable avec une couleur.

Remplir un calque dessinable avec une couleur **Syntaxe**: (img gimp-drawable-fill d tr)
Paramètres d DRAWABLE: identifiant d'un calque sur lequel on peut dessiner tr INT32: type de remplissage du calque { FOREGROUND-FILL (0), BACKGROUND-FILL (1), WHITE-FILL (2), TRANSPARENT-FILL (3), PATTERN-FILL (4), NO-FILL (5) }

⚑ **Remarque**

Cette fonction remplit un calque dessinable. Si le mode de remplissage est "premier plan", la couleur de premier plan courante est utilisée. Si le mode de remplissage est "arrière-plan", la couleur d'arrière-plan courante est utilisée. Si le type de remplissage est "blanc", le blanc est utilisé. Le remplissage "transparent" n'affecte que les calques avec un canal alpha. Si le dessinable n'a pas de canal alpha, il est rempli en blanc.

▶ **Exemple**

```
  Bienvenue sur TinyScheme
  Copyright (c) Dimitrios Souflis
  Console Script-Fu - Développement Scheme interactif

  > (let* ( (img (car (gimp-image-new 300 100 RGB )))
    (c (car (gimp-layer-new img 300 100 RGB-IMAGE ↯
     "MyC" 100 NORMAL-MODE))))
  (gimp-context-set-background  '(0 0 255))
  (gimp-image-insert-layer img c 0 0)
  (gimp-drawable-fill c BG-IMAGE-FILL)
  (gimp-display-new img)
  )
```

2.5.5) gimp-display-new

L'instruction **gimp-display-new** affiche une image à l'écran.

Afficher une image à l'écran **Syntaxe**: (img gimp-display-new img)
Paramètres img IMAGE: identifiant de l'image à afficher Valeur retournée d DISPLAY: identifiant du contexte d'affichage

⚑ **Remarque**
L'affichage créé peut ensuite être détruit avec un appel à la fonction gimp-display-delete. Cette procédure n'a de sens que pour une utilisation avec l'interface utilisateur de GIMP. Elle entraînera une erreur d'exécution si elle est appelée alors que GIMP n'a pas d'interface utilisateur ouverte.

2.5.6) gimp-jpeg-save

La fonction **file-jpeg-save** permet d'enregistrer une image dans un fichier au format jpeg.

Enregistrer une image chargée en mémoire dans un fichier jpg
Syntaxe: (file-jpeg-save rm img c out out q n o p c s b m)

Paramètres
```
rm INT32: run-mode - mode d'éxécution de la fonction
                  { RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1) }
img IMAGE: identificateur de l'image mémoire à enregistrer
c DRAWABLE: Calque associé à l'image et dont le contenu
                    est à enregistrer
out STRING: chemin complet du fichier à enregistrer
out STRING: chemin complet du fichier à enregistrer
q FLOAT: qualité d'enregistrement de l'image jpg (0<=q<=1)
n FLOAT: niveau de lissage (smooth) de l'image (0<=s<=1)
o INT32: codage de Huffman optimisé (1) ou non (0)
p INT32: codage de type image progressive (1) ou non(0)
c STRING:commentaire
s INT32: type de sous échantillonnage
           0 == 4:2:0 (chroma quartered),
           1 == 4:2:2 Horizontal (chroma halved),
           2 == 4:4:4 (best quality),
           3 == 4:2:2 Vertical (chroma halved)
b INT32: création d'une baseline (0) ou non (1)
m INT32:  marqueurs pour redémarrage en cas d'erreur (1) ou non (0)
d INT32: méthode DCT utilisée { INTEGER (0), FIXED (1), FLOAT (2) }
```

▶ **Exemple**
```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (define out "/home/ubuntu/essai2/001.jpg")
out
> (define c (car (gimp-image-get-active-drawable img)))
c
> (file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0 0)
(#t)
```

2.6) Mettre une image au format 3:2

2.6.1) Utiliser la fonction gimp-image-resize

Pour mettre une image de format quelconque au format 3:2 sans la déformer, on copie en mémoire la largeur w et la hauteur h initiales de cette image.

Ensuite on place cette image dans un canevas au format 3:2 et on complète la surface non utilisée de ce canevas avec une couleur de fond.

Pour cela on utilise la fonction **gimp-image-resize** qui permet d'effectuer deux tâches:

1. Redimensionner avec une nouvelle largeur et une nouvelle hauteur, le canevas qui contient une image chargée en mémoire .
2. Fixer la position de cette image à l'intérieur du canevas redimensionné.

Redimensionner le canevas qui contient une image chargée en mémoire et positionner l'image à l'intérieur du canevas redimensionné. **Syntaxe**: (gimp-image-resize img nw nh ox oy)
Paramètres img IMAGE: identifiant de l'image chargée en mémoire nw INT32: nouvelle largeur du canevas (en pixels) nh INT32: nouvelle hauteur du canevas (en pixels) ox INT32: décalage (offset) en x de l'image dans le nouveau canevas oy INT32: décalage (offset) en *y* de l'image dans le nouveau canevas ox et oy représentent les valeurs du décalage, horizontal et vertical en abscisse et en ordonnée (en pixels), de l'image par rapport à l'abscisse et à l'ordonnée (0,0) du canevas redimensionné.

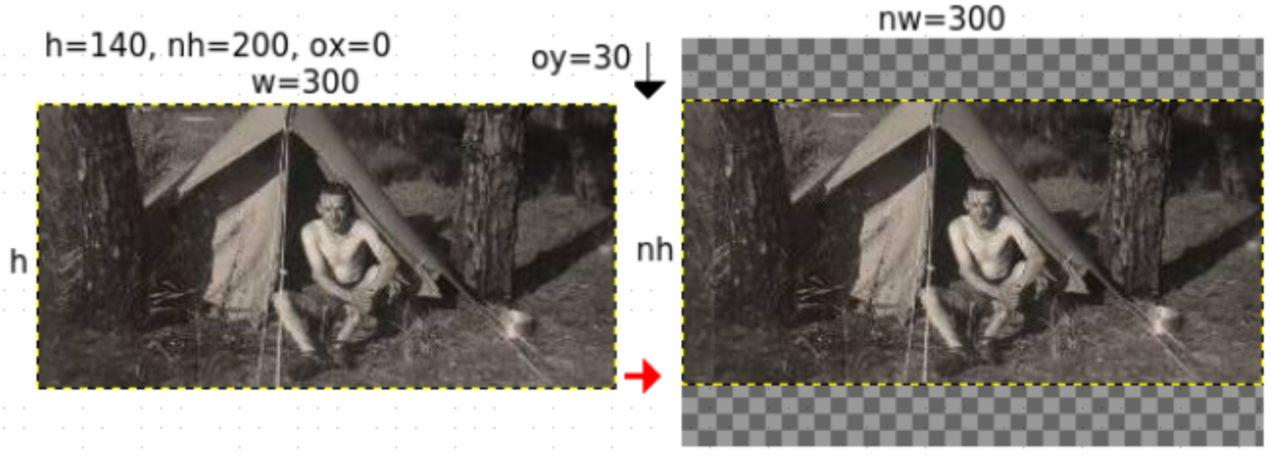
▶ **Exemple**

Soit l'image "/home/ubuntu/essai/test.jpg". On ouvre gimp puis la console script-fu et on saisit les lignes suivantes:

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> img
1
> (gimp-display-new 1)
(1)
 > (gimp-image-resize img 300 200 0 30)
 (#t)
```

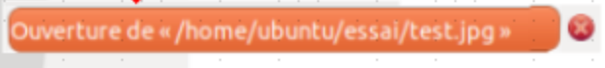
Les fonctions **gimp-file-load** et **gimp-display-new** seront examinées plus en détail au chapitre 3. Elles permettent de charger une image en mémoire puis de l'afficher dans la fenêtre de Gimp.

Dans le présent exemple, la fonction **gimp-image-resize** est utilisée pour passer en format 3:2 c'est à dire en taille 300x200 une image de taille initiale 300x140.
Dans ce cas, l'offset en x est égal à 0 et l'offset en y est égal à 30 car on veut que la nouvelle image soit centrée dans son canevas.




```
(gimp-image-resize img 300 200 0 30)
```

```
> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> img
1
```



```
> (gimp-display-new 1)
(1)
```

```
> (gimp-image-resize img 300 200 0 30)
(#t)
```

2.6.1) Identifier les types de formats possibles

Une image donnée peut présenter un nombre considérable de tailles et de formats différents. Cependant, pour ce qui concerne la mise au format 3:2 de cette image, on peut se limiter à considérer les cinq cas de figure suivants :

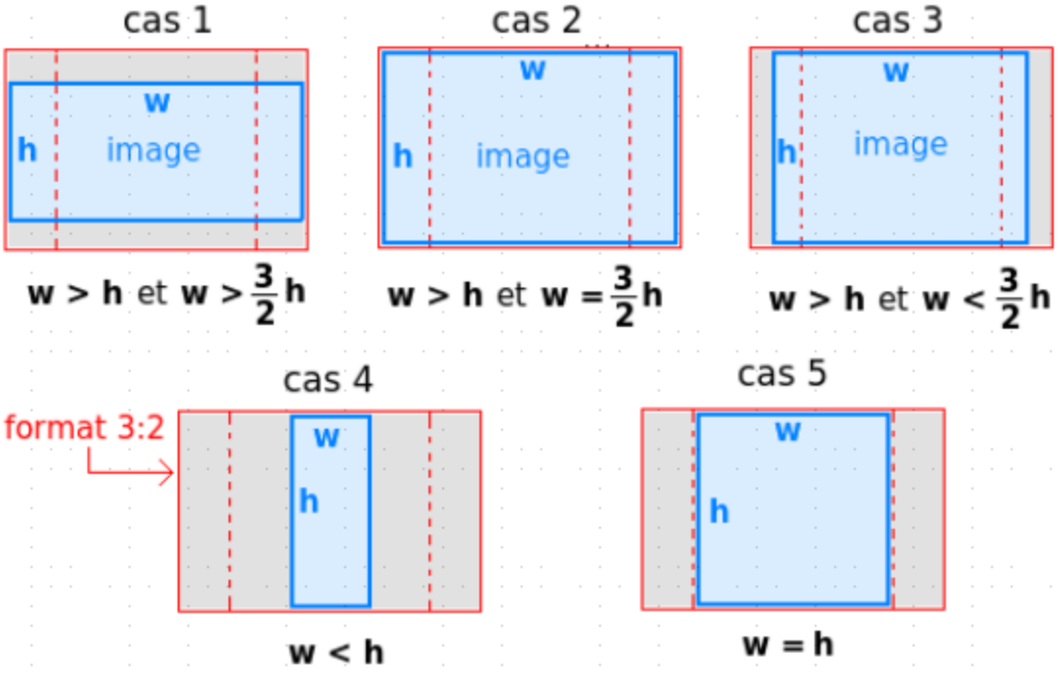
Cas 1 - Si la largeur w de l'image est à la fois supérieure à sa hauteur h et supérieure à 1,5 fois sa hauteur, il faut la compléter avec une couleur de fond au dessus et au dessous ;

Cas 2 - Si la largeur w de l'image est à la fois supérieure à sa hauteur h et égale à 1,5 fois sa hauteur, il n'y a rien à faire car l'image est déjà au format 3:2 ;

Cas 3 - Si la largeur w de l'image est à la fois supérieure à sa hauteur h et inférieure à 1,5 fois sa hauteur, il faut la compléter avec une couleur de fond à gauche et à droite

Cas 4 - Si la largeur w de l'image est inférieure à sa hauteur h, il faut la compléter avec une couleur de fond à gauche et à droite

Cas 5 - Si la largeur w de l'image est égale à sa hauteur h, il faut également la compléter avec une couleur de fond à gauche et à droite



Dans ces cinq cas le script suivant place l'image de départ dans un cadre paysage 3:2 le plus petit possible, en remplissant les parties non occupées par cette image avec la couleur de fond choisie (c'est à dire avec couleur d'arrière-plan sélectionnée, dans la fenêtre de Gimp).

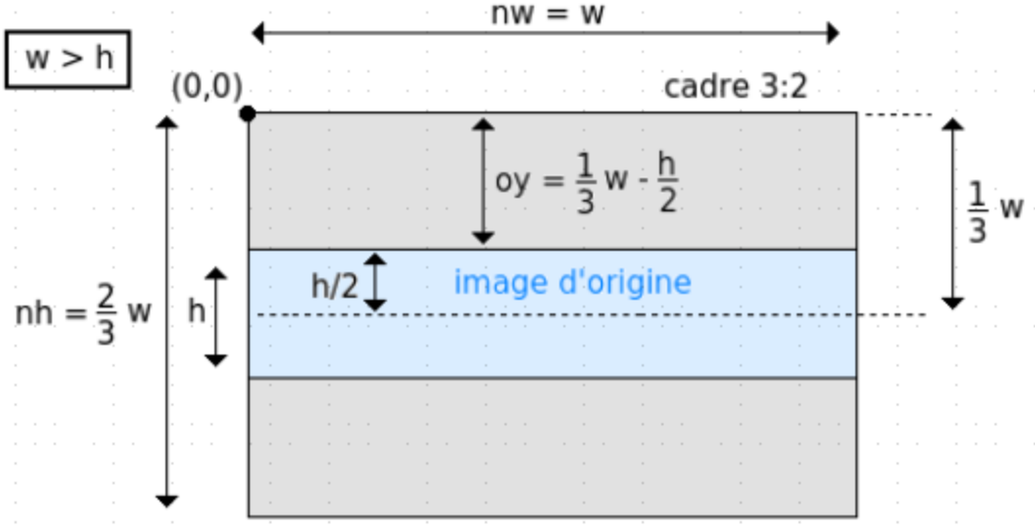
Pour modifier la couleur d'arrière-plan courante, il suffit de cliquer le rectangle correspondant, situé sous les outils. Cela ouvre une boîte de dialogue qui permet de modifier cette couleur.

2.6.3) Préparer le programme

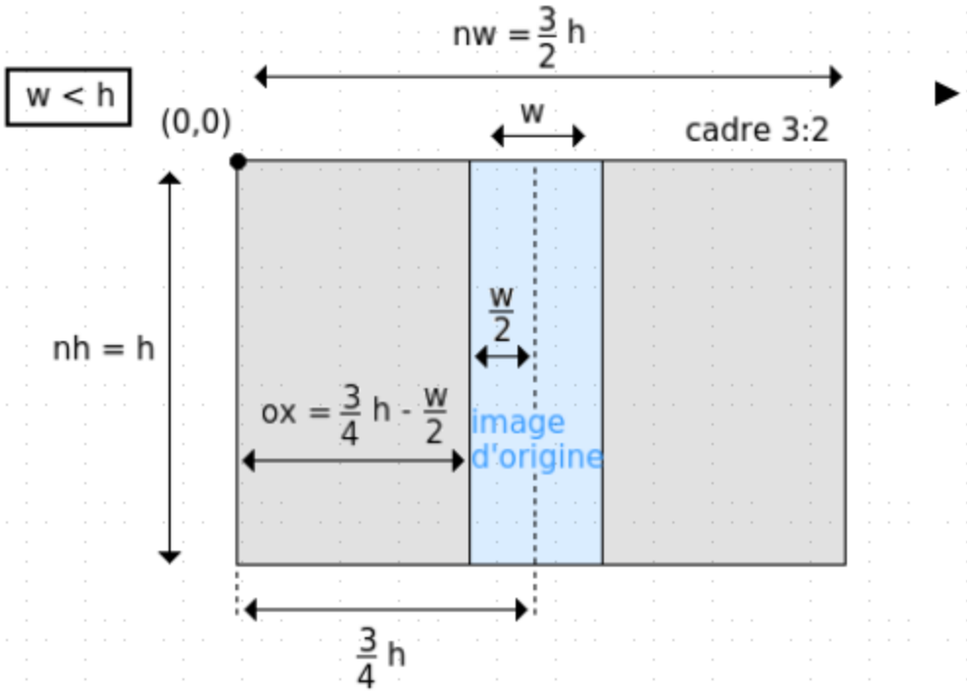
Pour ce script, on utilise les variables suivantes :

w et h	Largeur et hauteur de l'image de départ ouverte dans GIMP ;
u et v	Copie de la largeur et de la hauteur de l'image de départ ouverte dans GIMP ;
nw et nh	Nouvelle largeur et nouvelle hauteur de l'image après passage au format paysage 3/2 ;
ox et oy	Décalage horizontal et vertical de l'ancienne image, par rapport à l'abscisse et à l'ordonnée (0,0) du cadre au format 3:2 délimitant la nouvelle image ;
i et j	Variables permettant de faciliter les calculs intermédiaires lors des calculs de nw, nh, ox et oy ;
k	pour rendre plus lisibles certaines conditions placées entre parenthèses dans les instructions conditionnelles if ( ).

Le schéma suivant illustre les relations entre ces variables lorsque w>h



Le schéma suivant illustre les relations entre ces variables lorsque w<h



2.6.4) Écrire le script

On écrit le script suivant avec un éditeur de texte, on enregistre le fichier sous le nom "ct-paysage-3-2.scm" puis on le place dans le répertoire des scripts de Gimp.

⚑ On rappelle que l'utilisation de ce nom est obligatoire car il doit correspondre à celui indiqué dans le script afin que GIMP puisse l'enregistrer correctement.

Sous Windows

```
C:\Program Files\GIMP\lib\gimp\2.0\share\gimp\2.0\scripts
```

Sous Linux

```
/usr/share/gimp/2.0/scripts/
sudo cp ct-paysage-3-2.scm /usr/share/gimp/2.0/scripts/
```

La première partie du script consiste à déclarer les variables qui sont utilisées dans le script. Le paramètre **img** passé à la fonction **define** est l'identificateur de l'image qui va être traitée par le script.

Un paramètre **SF-IMAGE** est utilisé lorsque le script opère sur une image déjà ouverte dans Gimp. Associé au paramètre **img**, il doit être placé en fin de script, suivi d'une valeur 0 qui correspond à l'image courante.

C'est par exemple le cas lorsque le script est appliqué à une image ayant été ouverte dans Gimp par l'interrmédiaire du menu **Fichier > Ouvrir**.

ct-paysage-3-2.scm

```
;-----------------------------------------------------------------------
; Ce script met au format paysage 3:2 une image quelconque ouverte
; dans GIMP, sans la déformer, en utilisant une couleur de remplissage
; Auteur : Claude Turrier - 28 mai 2016
; Installation sous Linux :
; sudo cp ct-paysage-3-2.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------------
;
(define (ct-paysage-3-2 img)
;-déclarer les variables utilisées par le script
(let* ((nw 0) (nh 0) (ox 0) (oy 0) (u 0) (v 0) (i 0) ( j 0) (k 0)
        (h (car (gimp-image-height img))) (w (car (gimp-image-width
img))))
;-fixer la couleur de remplissage à noir
(gimp-context-set-background '(0 0 0))
;-placer la hauteur et la largeur de l'image ouverte dans u et v
(set! u  w)  (set! v  h)  (set! k   (* (/ 3 2) v) )
;-quel que soit le format de l'image ouverte on le transforme  3/2
;-si la hauteur de l'image est inférieure à sa largeur,
;-on la complète avec du noir en haut et en bas
;-sinon on la complète avec du noir à gauche et à droite
;
;-cas n°1 : w > h et w > 1,5 h
(if (> u  v)
(if (> u  k)
   (begin
   (set! i (/ 2 3)) (set! nh (* w i))
   (set! i (/ 1 3)) (set! j (* w i)) (set! i (/ h 2)) (set! oy (- j i ))
   (gimp-image-resize img w nh 0 oy)
)));end begin  ;end if  ;end if
;
;-cas n°3 : w > h et w < 1,5 h
(if (> u  v)
(if (< u  k )
   (begin
   (set! i (/ 3 2)) (set! nw (* h i))
  (set! i (/ 3 4)) (set! j (* h i)) (set! i (/ w 2)) (set! ox (- j i))
  (gimp-image-resize img nw h ox 0)
)));end begin  ;end if  ;end if
;
;-cas n° 4 et 5 : w < ou = h
(if (<= u  v)
  (begin
  (set! i (/ 3 2)) (set! nw (* h i))
  (set! i (/ 3 4)) (set! j (* h i)) (set! i (/ w 2)) (set! ox (- j i))
  (gimp-image-resize img nw h ox 0)
));end begin ;end if
;
;-aplatir l'image
(gimp-image-flatten img)
;-afficher le résultat à l'écran
(gimp-display-new img)
)) ; fin de let* ; fin de define
;
```

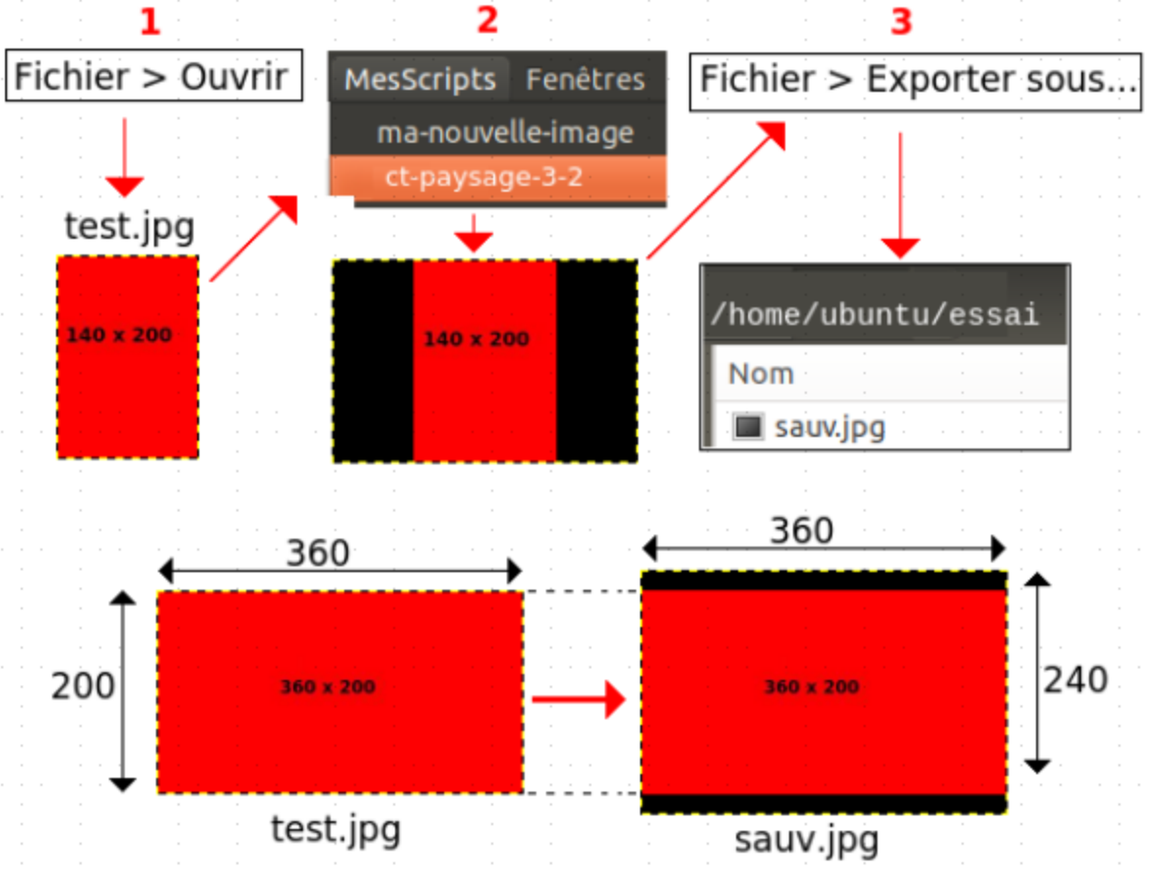
```
;--Enregistrer le script
(script-fu-register "ct-paysage-3-2"
"<Image>/MesScripts/ct-paysage-3-2"
"------------------------------"
"Met les images au format 3/2"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0
)
```

2.6.5) Utiliser le script

On utilise le script après avoir ouvert une image dans Gimp depuis le menu **Fichier > Ouvrir**. On procède de la façon suivante:

1) On démarre **Gimp** ;
2) On ouvre une image ("test.jpg" par ex.) par **Fichier > Ouvrir** ;
3) On applique clique le menu **MesScripts > ct-paysage3-2** ;
4) On enregistre l'image modifiée ("sauv.jpg" par exemple) depuis le menu **Fichier > Exporter sous...** ;
5) On ferme l'image depuis le menu **Fichier > Fermer tout**.

L'image est complétée avec des bandes noires (couleur d'arrière-plan).



2.6.6) gimp-image-height

La fonction **gimp-image-height** retourne la hauteur (en pixels) d'une image.

```
Récupérer la hauteur, en pixels, d'une image
Syntaxe: (gimp-image-height img)
```

```
Paramètres
img IMAGE: image dont on souhaite obtenir la hauteur
Valeur retournée
h INT32: hauteur de l'image en pixels

Cette valeur est associée à la hauteur du canevas contenant l' image (et non pas à la hauteur d'un des calques)
```

⚑ **Remarque**
gimp-image-width est analogue mais retourne la largeur w de l'image.

▶ **Exemple**
Soit l'image test.jpg de wxh= 300x140 pixels située dans le répertoire suivant sous Linux:
"/home/ ubuntu/essai/"

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (gimp-display-new img)
(3)
> (gimp-image-height img)
(140)
> (gimp-image-width img)
(300)
```

2.6.7) gimp-image-flatten

La fonction **gimp-image-flatten** aplatit tous les calques visibles en un seul calque, sans prendre en compte les calques invisibles. L'image résultante est dépouillée de son canal alpha.

```
Applatir une image
Syntaxe: (gimp-image-flatten img)
```

```
Paramètres
img IMAGE: image dont on souhaite aplatir les calques
 Valeur retournée
c LAYER: calque résultant de l'applatissage
```

▶ **Exemple**
Soit l'image "test.jpg" de dimensions w.h= 300x140 pixels, située dans le répertoire suivant de Linux:

```
"/home/ ubuntu/essai/"

 Bienvenue sur TinyScheme
 Copyright (c) Dimitrios Souflis
 Console Script-Fu - Développement Scheme interactif

 > (define in "/home/ubuntu/essai/test.jpg")
 in
 > (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
 img
 > (gimp-display-new img)
 (1)
 > (gimp-image-flatten img)
 (3)
```

2.7) Modifier l'échelle d'une image

La fonction **gimp-image-scale** permet de modifier l'échelle d'une image en utilisant la méthode d'interpolation par défaut.

Modifier l'échelle d'une image **Syntaxe**: (gimp-image-scale img nw nh)
Paramètres img IMAGE: image que l'on souhaite redimensionner nw INT32: nouvelle largeur de l'image (1 <= nw <= 262144) nh INT32: nouvelle hauteur de l'image (1 <= nh <= 262144) Cette fonction met l'image à l'échelle afin que ses nouvelles largeur et hauteur soient égales aux paramètres fournis. Tous les calques et canaux de l'image sont mis à l'échelle en fonction des paramètres spécifiés. La méthode d'interpolation utilisée peut être définie avec la fonction **gimp-context-set-interpolation**.

Soit une image ouverte dans Gimp à l'aide du le menu **Fichier > Ouvrir**.

L'appel du script "ct-scale.scm" suivant, depuis le menu "**MesScripts**" ouvre une boite de dialogue permettant de saisir la largeur de la nouvelle image puis redimensionne automatiquement l'image avec la nouvelle largeur en préservant le ratio initial **w/h** de cette image

ct-scale.scm

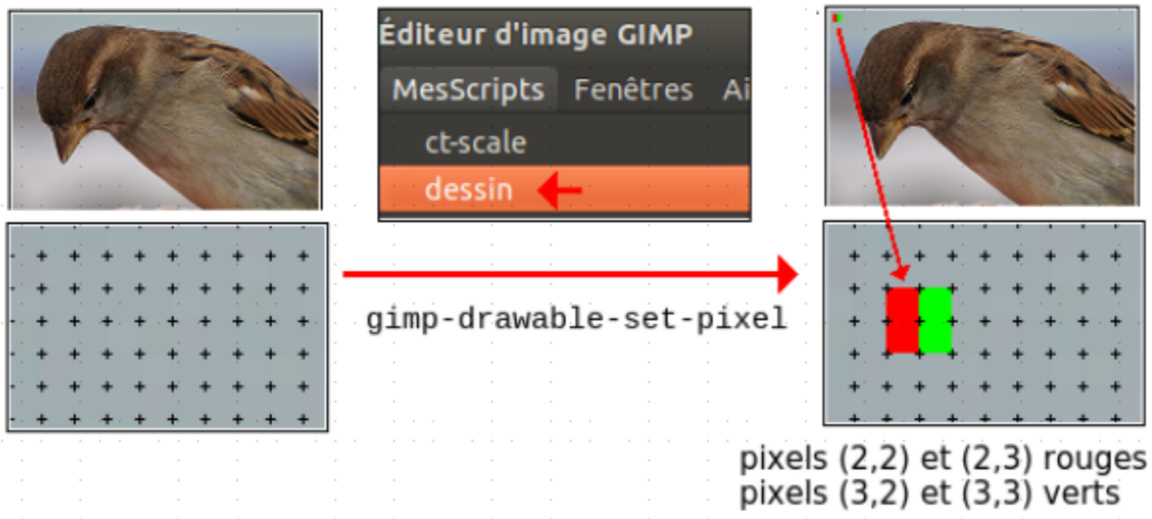
```
;-----------------------------------------------------------------------
;
; Ce script redimensionne une image quelconque ouverte
; dans GIMP, sans la déformer (ratio k=w/h conservé)
; Auteur : Claude Turrier - 28 mai 2021
; Installation sous Linux :
; sudo cp ct-scale.scm /usr/share/gimp/2.0/scripts/
;
;-----------------------------------------------------------------------
;
```

```
(define (ct-scale img nw)
(let* ((nh 0) (k 0) (h (car (gimp-image-height img))) ↯
       (w (car (gimp-image-width img))))
(set! k (/ w h))
(set! nh (/ nw k))
(gimp-image-scale img nw nh)
(gimp-image-flatten img)
(gimp-display-new img)
)
)
(script-fu-register "ct-scale"
"<Image>/MesScripts/ct-scale"
"------------------------------"
"Redimensionne une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;->img
SF-VALUE "nw" "100"    ;->nw
)
```

2.8) Dessiner des pixels sur une image

Il est possible de dessiner automatiquement des pixels sur une image à l'aide d'un script. Cette image peut être une image nouvelle, créée avec GIMP, ou une image existante, enregistrée dans un fichier de type quelconque (jpg, png, tiff ou bmp...) et ouverte dans GIMP.

L'exemple ci-après est facilement généralisable. Il montre comment dessiner automatiquement deux pixels rouges aux points de coordonnées x,y égales à (2,2) et (2,3) et deux pixels verts aux points de coordonnées (3,2) et (3,3) sur une image (jpg, png...) ouverte dans GIMP.



ct-dessin.scm

```
;-------------------------------------------------
; Ce script permet de dessiner des pixels sur une image ouverte dans GIMP
; Auteur : Claude Turrier - 28 mai 2016
; Installation sous Linux:
; sudo cp ct-dessin.scm /usr/share/gimp/2.0/scripts/
;-------------------------------------------------
;
(define (ct-dessin img )
;--------Déclarer les variables---
(let* ( (moncalque 0) (monpixel (cons-array 4 'byte)))
;-Créer un nouveau calque transparent et l'ajouter à l'image en cours
(set! moncalque (car (gimp-layer-new img 200 200 RGBA-IMAGE ↯
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img moncalque 0 0)
;--- Dessiner les pixels sur le calque transparent ajouté à l'image
(aset monpixel 0 255) ;rouge=255
(aset monpixel 1 0) ;vert=0
(aset monpixel 2 0) ;bleu=0
(aset monpixel 3 255) ;alpha=255
(gimp-drawable-set-pixel moncalque 2 2 4 monpixel)
(gimp-drawable-set-pixel moncalque 2 3 4 monpixel)
(aset monpixel 0 0) (aset monpixel 1 255) (aset monpixel 2 0)
(aset monpixel 3 255) ;rgba=(0,255,0,255)
  (gimp-drawable-set-pixel moncalque 3 2 4 monpixel)
(gimp-drawable-set-pixel moncalque 3 3 4 monpixel)
;------ Aplatir l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistrer le script par
(script-fu-register "ct-dessin"
"<Image>/MesScripts/ct-dessin"
"------------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```

La fonction **cons-array** (aujourd'hui dépréciée) permet de créer un tableau à une dimension, de 4 valeurs de type **byte** dans le cas présent.

Créer un tableau à une dimension **Syntaxe**: (cons-array n type)
Paramètres n : nombre d'éléments du tableau type : type d'éléments (byte, double, lisp)

▶ **Exemple**

```
(cons-array 4 'byte)))
```

La fonction **aset** (aujourd'hui dépréciée) permet de ranger, dans un tableau, une valeur donnée à un emplacement donné.

```
Ranger une valeur dans un tableau de n éléments
Syntaxe: (aset t i v)

Paramètres

t : tableau créé avec la fonction cons-array
i : index du tableau concerné (0 à n-1)
v: valeur à ranger dans la case d'index i du tableau
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define monpixel (cons-array 4 'byte))
monpixel
> (aset monpixel 0 255)
#(255 0 0 0)
> (aset monpixel 1 0)
#(255 0 0 0)
> (aset monpixel 2 0)
#(255 0 0 0)
> (aset monpixel 3 255)
#(255 0 0 255)#<EOF>
> monpixel
#(255 0 0 255)
```

Il est à noter que jusqu'à la version **2.4**, Gimp utilisait l'interpréteur scheme **SIOD** (Scheme In One Defun - Scheme en un seul coup).

Depuis la version 2.4, Gimp n'utilise plus cet interpréteur mais utilise l'interpréteur **TinyScheme**.

La syntaxe des instructions en langage Scheme reste la même mais certaines instructions sont aujourd'hui dépréciées bien qu'elles continuent de fonctionner. Elles sont remplacées par de nouvelles instructions dont l'usage est conseillé (pour assurer la continuité de fonctionnement des scripts avec les futures versions de Gimp).

C'est notamment le cas des instructions **cons-array** et **aset** qui sont respectivement remplacées par **make-vector** et **vector-set!**

Le listing précédent sera donc avantageusement remplacé par le suivant. Celui-ci fonctionne comme le précédent mais utilise les nouvelles instructions, ce qui garantie son fonctionnement dans les futures versions de gimp

ct-dessin.scm

```
;-----------------------------------------------
; Ce script permet de dessiner des pixels sur une image ouverte dans GIMP
; Auteur : Claude Turrier - 28 mai 2016
; Installation sous Linux:
; sudo cp ct-dessin.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------
;
(define (ct-dessin img )
;--------Déclarer les variables---
(let* ( (moncalque 0) (monpixel (make-vector 4 'byte)))
;-Créer un nouveau calque transparent et l'ajouter à l'image en cours
(set! moncalque (car (gimp-layer-new img 200 200 RGBA-IMAGE ↯
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img moncalque 0 0)
;--- Dessiner les pixels sur le calque transparent ajouté à l'image
(vector-set! monpixel 0 255) ;rouge=255
(vector-set! monpixel 1 0) ;vert=0
(vector-set! monpixel 2 0) ;bleu=0
(vector-set! monpixel 3 255) ;alpha=255
(gimp-drawable-set-pixel moncalque 2 2 4 monpixel)
(gimp-drawable-set-pixel moncalque 2 3 4 monpixel)
(vector-set! monpixel 0 0) (aset monpixel 1 255) (aset monpixel 2 0)
(vector-set! monpixel 3 255) ;rgba=(0,255,0,255)
(gimp-drawable-set-pixel moncalque 3 2 4 monpixel)
(gimp-drawable-set-pixel moncalque 3 3 4 monpixel)
;------ Aplatir l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistrer le script par
(script-fu-register "ct-dessin"
"<Image>/MesScripts/ct-dessin"
"------------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```

⚑ Fonctions SIOD, dépréciées

SIOD	TinyScheme
aref	vector-ref
aset	vector-set!
cons-array	make-vector
fopen	open-input-file
mapcar	map
nil	'()
nreverse	reverse
pow	expt

⚑ Fonctions SIOD, dépréciées	
SIOD	**TinyScheme**
print	write
string-lessp	string<?
symbol-bound?	defined?
trcat	String-append

Le script utilise la fonction **gimp-drawable-set-pixel** qui permet de dessiner, sur un calque dessinable, avec une couleur donnée, un pixel situé à un emplacement donné.

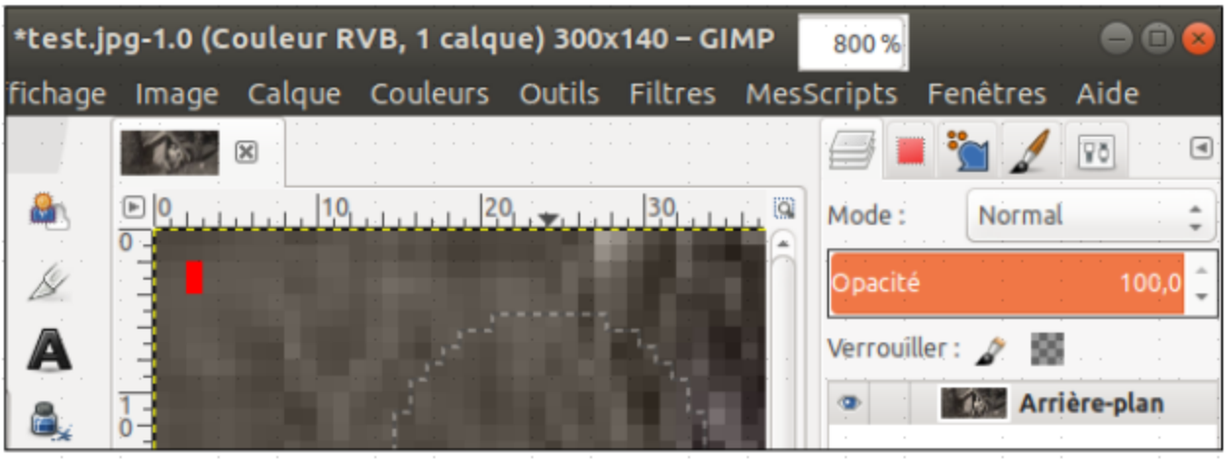
```
Dessiner un pixel sur un calque dessinable
Syntaxe: (gimp-drawable-set-pixel c x y n pixel)

Paramètres

c DRAWABLE: identifiant du calque dessinable sur lequel on va dessiner
            le pixel
x INT32: abscisse du pixel (x>=0)
y INT32: ordonnée du pixel (y>=0)
n INT32: nombre de canaux pour le pixel (RGB->3 et RGBA->4)
pixel INT8ARRAY: valeur du pixel (tableau de bytes)
```

▶ **Exemple**
Soit une image de dimension 300x140 pixels, enregistrée dans le fichier "test.jpg" dans le répertoire "/home/ubuntu/essai"

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (define monpixel (make-vector 4 'byte))
monpixel
> (define c (car (gimp-layer-new img 300 140 RGBA-IMAGE ↯
  "MonCalque" 100 NORMAL-MODE)))
c
> (gimp-image-insert-layer img c 0 0)
(#t)
> (vector-set! monpixel 0 255)
#(255 byte byte byte)
> (vector-set! monpixel 1 0)
#(255 0 byte byte)
> (vector-set! monpixel 2 0)
#(255 0 0 byte)
> (vector-set! monpixel 3 255)
#(255 0 0 255)
> (gimp-drawable-set-pixel c 2 2 4 monpixel)
```

```
(#t)
> (gimp-drawable-set-pixel c 2 3 4 monpixel)
(#t)
> (gimp-image-flatten img)
(4)
> (gimp-display-new img)
(1)
```

Lorsqu'on saisit les instructions précédentes une par une, dans la console script-fu, l'image avec les deux pixels rouges dessinés s'affichent dans la fenêtre de Gimp seulement lorsque la dernière instruction (gimp-display-new img) est exécutée.



Les instructions du programme **ct-dessin.scm** assurent les tâches séquentielles suivantes :

1	Création d'un calque transparant ayant l'identifiant "moncalque", associé à l'image en cours d'identifiant "img", ouverte dans gimp depuis le menu Fichier > Ouvrir
2	Insertion de ce calque, au dessus de l'image, dans la pile des calques
3	Dessin de pixels sur ce calque
4	Applatissement de l'image
5	Affichage de l'image finale contenant les pixels colorés

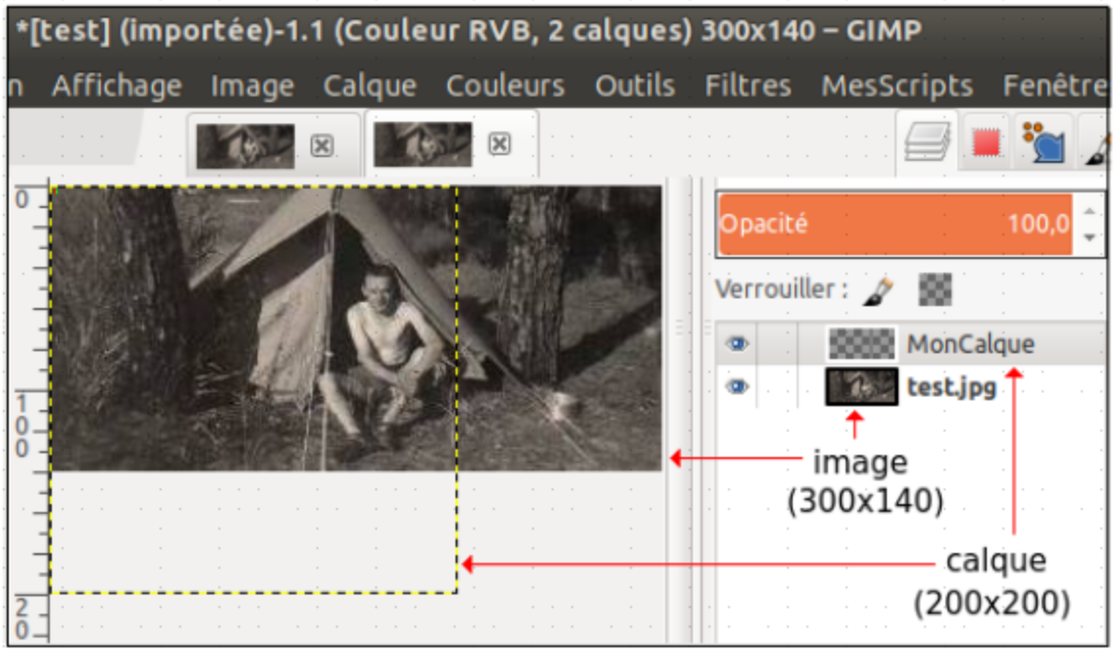
Si on charge une image de dimension 300x140 pixels par exemple, le calque ayant une dimension de 200x200 pixels, sa surface ne correspond pas à celle de l'image. Cependant il prend la dimension de l'image quand on applatit l'image.

Ce programme est utilisé ici uniquement pour illustrer le fonctionnement des instructions utilisées. En réalité, il faudrait donner au calque non pas une dimension fixe mais lui donner les dimensions de l'image qu'on

pourrait récupérer à l'aide des instructions gimp-image-height et gimp-image-width vues en 2.6

Si on ouvre dans gimp une image de 300x140 pixels, par exemple, et qu'on supprime l'instruction "gimp-image-flatten img" dans le script, l'affichage dans la fenêtre de Gimp fait clairement apparaître ces différences de surfaces entre le calque et le canevas de l'image.

```
(gimp-drawable-set-pixel moncalque 3 3 4 monpixel)
;------ Aplatir l'image ---------
;(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
```



Compte tenu de ces éléments le script amélioré est le suivant:

ct-dessin.scm

```
;-----------------------------------------------------
; Ce script permet de dessiner des pixels sur une image ouverte dans GIMP
; Auteur : Claude Turrier - 28 mai 2016
; Installation sous Linux:
; sudo cp ct-dessin.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------
;
(define (ct-dessin img )
;--------Déclarer les variables---
(let* ( (c 0) (monpixel (make-vector 4 'byte))
(h (car (gimp-image-height img))) (w (car (gimp-image-width img))))
;-Créer un nouveau calque transparent et l'ajouter à l'image en cours
(set! c (car (gimp-layer-new img w h RGBA-IMAGE ↯
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img c 0 0)
;--- Dessiner les pixels sur le calque transparent ajouté à l'image
```

```
(vector-set! monpixel 0 255) ;rouge=255
(vector-set! monpixel 1 0) ;vert=0
(vector-set! monpixel 2 0) ;bleu=0
(vector-set! monpixel 3 255) ;alpha=255
(gimp-drawable-set-pixel c 2 2 4 monpixel)
(gimp-drawable-set-pixel c 2 3 4 monpixel)
(vector-set! monpixel 0 0) (aset monpixel 1 255) (aset monpixel 2 0)
(vector-set! monpixel 3 255) ;rgba=(0,255,0,255)
(gimp-drawable-set-pixel c 3 2 4 monpixel)
(gimp-drawable-set-pixel c 3 3 4 monpixel)
;------ Aplatir l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistrer le script par
(script-fu-register "ct-dessin"
"<Image>/MesScripts/ct-dessin"
"------------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```

2.9) Ajouter un cadre à une image

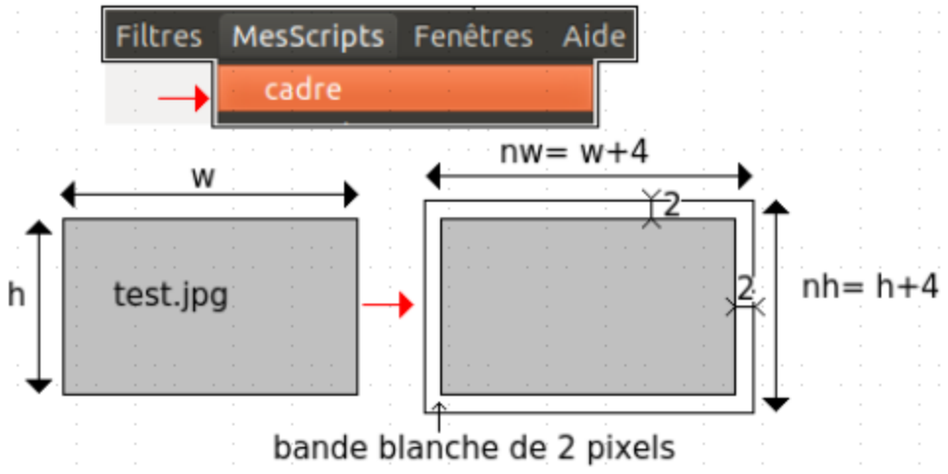
2.9.1) Ajouter une bordure blanche à une image

Le script suivant montre comment ajouter une bordure blanche extérieure de 2 pixels à une image.
Pour cela, on utilise la fonction **gimp-image-resize** vue en 2.6.
Si l'image de départ possède une largeur w et une hauteur h (en pixels), la nouvelle image possède une largeur nw=w+4 et une hauteur nh=h+4. L'ancienne image est centrée dans ce nouveau cadre.

ct-cadre.scm

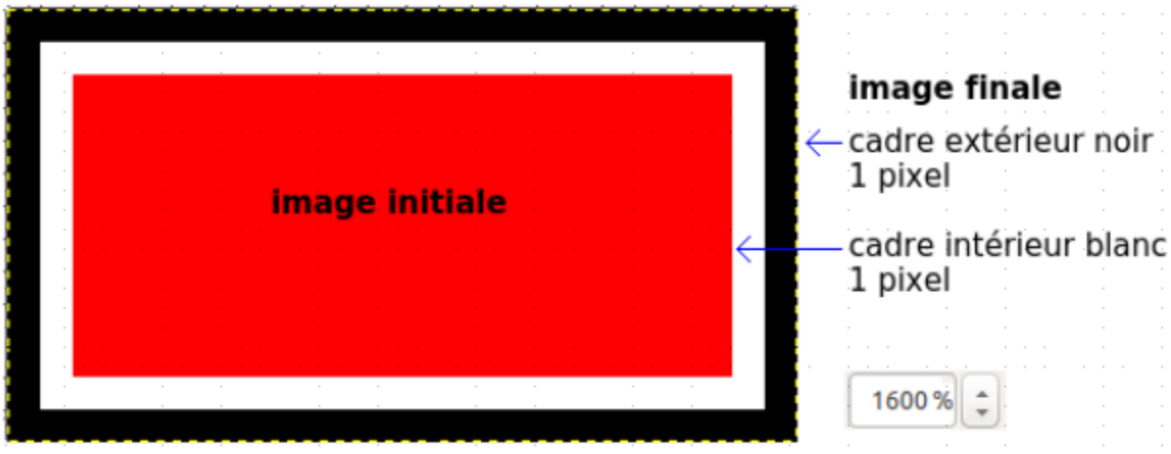
```
;-------------------------------------------------
; Ce script permet d'ajouter une ligne blanche  de 2 pixels
; à la périphérie d'une image ouverte dans GIMP
; Auteur : Claude Turrier - 5 mars 2021
; Installation sous Linux:
; sudo cp ct-cadre.scm /usr/share/gimp/2.0/scripts/
;-------------------------------------------------
;
(define (ct-cadre img )
;--------Déclare les variables---
(let* ( (c 0)
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(nw (+ w 4)) (nh (+ h 4)) )
;-Crée un nouveau calque transparent et l'ajoute à l'image en cours
(gimp-image-resize img nw nh 2 2)
```

```
(set! c (car (gimp-layer-new img nw nh RGBA-IMAGE
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img c 0 0)
;--------------------------------
;------ Aplatit l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistre le script
(script-fu-register "ct-cadre"
"<Image>/MesScripts/cadre"
"-test-----------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"-------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```



2.9.2) Dessiner un cadre blanc-noir

Il ne reste plus qu'à ajouter, avant le commentaire ";applatit l'image", le code qui trace le cadre blanc-noir à l'extérieur de l'image.



Le script final est le suivant

ct-cadre.scm

```
;-------------------------------------------------
; Ce script ajoute un cadre noir et blanc de 2 pixels
; à la périphérie d'une image ouverte dans GIMP
; Auteur : Claude Turrier - 5 mars 2021
; Installation sous Linux:
; sudo cp ct-cadre.scm /usr/share/gimp/2.0/scripts/
;-------------------------------------------------
;
(define (ct-cadre img )
;--------Déclare les variables---
(let* ( (c 0)
(couleur_ext '(0 0 0)) ; couleur de la partie extérieure du cadre
(couleur_int '(255 255 255)) ; couleur de la partie intérieure du cadre
(n 10) ; nombre de coordonnées des points constituant le tracé
(moncadre (make-vector 10 'double))
;-------------------
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(u w) (v h) (nw (+ w 4)) (nh (+ h 4)) )
;-Crée un nouveau calque transparent et l'ajoute à l'image en cours
(gimp-image-resize img nw nh 2 2)
(set! c (car (gimp-layer-new img nw nh RGBA-IMAGE
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img c 0 0)
;-----Dessin du double cadre--------------
(gimp-brush-new "mabrosse")
(gimp-brushes-refresh)
(gimp-context-set-brush "mabrosse")
(gimp-context-set-brush-aspect-ratio 0)
;--cadre extérieur noir de 1 pixel d'épaisseur
(gimp-context-set-foreground couleur_ext) ;noir
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 1))
(set! v (- nh 1))
(vector-set! moncadre 0 0) ; point 1 (x)
(vector-set! moncadre 1 0) ; point 1 (y)
(vector-set! moncadre 2 u) ; point 2 (x)
(vector-set! moncadre 3 0) ; point 2 (y)
(vector-set! moncadre 4 u) ; point 3 (x)
(vector-set! moncadre 5 v) ; point 3 (y)
(vector-set! moncadre 6 0) ; point 4 (x)
(vector-set! moncadre 7 v) ; point 4 (y)
(vector-set! moncadre 8 0) ; point 5 (x)
(vector-set! moncadre 9 0) ; point 5 (y)
(gimp-pencil c n moncadre ); tracé dans l'ordre des points
;--cadre intérieur blanc de 1 pixel d'épaisseur
(gimp-context-set-foreground couleur_int) ; blanc
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 2))
(set! v (- nh 2))
(vector-set! moncadre 0 1)
(vector-set! moncadre 1 1)
(vector-set! moncadre 2 u)
(vector-set! moncadre 3 1)
(vector-set! moncadre 4 u)
```

```
(vector-set! moncadre 5 v)
(vector-set! moncadre 6 1)
(vector-set! moncadre 7 v)
(vector-set! moncadre 8 1)
(vector-set! moncadre 9 1)
(gimp-pencil c n moncadre )
;------ Aplatit l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistre le script
(script-fu-register "ct-cadre"
"<Image>/MesScripts/cadre"
"-test-----------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"-------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```

Avec ce script, on découvre les nouvelles fonctions suivantes
- gimp-brush-new ;
- gimp-brushes-refresh ;
- gimp-context-set-brush ;;
- gimp-context-set-brush-aspect-ratio ;
- gimp-context-set-foreground ;
- gimp-context-set-brush-size ;
- gimp-pencil.

2.9.3) gimp-brush-new

La fonction **gimp-brush-new** crée un nouveau pinceau.

```
Créer un nouveau pinceau
Syntaxe: (gimp-brush-new nom)
```

```
Paramètres
nom STRING: nom attribué au nouveau pinceau

Valeur retournées
name STRING: nom du pinceau
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (gimp-brush-new "mabrosse")
("mabrosse #5")
```

2.9.4) gimp-brushes-refresh

La fonction **gimp-brushes-refresh** récupère tous les pinceaux utilisables par Gimp et met à jour les boîtes de dialogue correspondantes.

```
Récupérer l'ensemble des pinceaux utilisables par Gimp
Syntaxe: (gimp-brushes-refresh)
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (gimp-brushes-refresh)
(#t))
```

2.9.5) gimp-context-set-brush

La fonction **gimp-context-set-brush** spécifie un pinceau comme étant le pinceau courant.

```
Spécifier un pinceau comme étant le pinceau courant
Syntaxe: (gimp-brush-new nom)
```

```
Paramètres
nom STRING: nom du pinceau

Le pinceau spécifié devient actif et sera utilisé dans toutes les
opérations de peinture suivantes.
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (gimp-brush-new "mabrosse")
("mabrosse #6")
> (gimp-brushes-refresh)
(#t)
> (gimp-context-set-brush "mabrosse")
(#t)
```

2.9.6) gimp-context-set-brush-aspect-ratio

La fonction **gimp-context-set-brush-aspect-ratio** fixe un ratio d'aspect (rapport hauteur/largeur) pour les pinceaux de Gimp.

```
Fixer le ratio d'aspect des pinceaux de Gimp
Syntaxe: (gimp-context-set-brush-aspect-ratio ar)
```

```
Paramètres
ar FLOAT: ratio d'aspect w/h qu'on fixe pour les pinceaux de Gimp
          (-20 <= ar <= 20)
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (gimp-context-set-brush-aspect-ratio 0)
(#t)
```

2.9.7) gimp-context-set-foreground

La fonction **gimp-context-set-foreground** spécifie la couleur de premier plan (couleur de dessin) courante.

Spécifier la couleur de premier plan courante **Syntaxe**: (gimp-context-set-foreground f)
Paramètres f COLOR: couleur de premier plan Une fois ce paramètre défini, les opérations qui utilisent le premier plan telles que les outils de peinture, le mélange et le remplissage utiliseront cette couleur.

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define macouleur '(255 0 0))
macouleur
> (gimp-context-set-foreground macouleur)
(#t)
```

2.9.8) gimp-context-set-brush-size

La fonction **gimp-context-set-brush-size** fixe la dimension (en pixels) du pinceau courant.

Fixer la dimension (en pixels) du pinceau courant **Syntaxe**: (gimp-context-set-brush-size d)
Paramètres d FLOAT: dimension du pinceau courant en pixels (d>=0)

▶ **Exemple**

```
> (gimp-context-set-brush-size 1)
(#t)
```

⚑**Remarque**

En faisant des essais, on constate que les pinceaux de tailles impaires (1, 3, 5, 7…) tracent exactement avec la taille spécifiée. En revanche, les pinceaux de tailles paires effectuent des tracés dont la taille est impaire et supérieure (ou inférieure) de 1 pixel à la taille spécifiée. D'autre part, dans le trait effectué par un pinceau, la ligne du milieu est celle qui correspond à l'abscisse et à l'ordonnée prise en compte lors de l'exécution de l'instruction gimp-pencil.

2.9.9) gimp-pencil

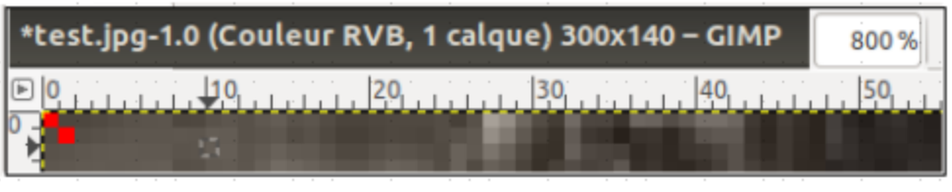
La fonction **gimp-pencil** réalise un tracé avec le pinceau courant.

```
Réaliser un tracé avec le pinceau courant
Syntaxe: (gimp-context-set-brush-size d)
```

```
Paramètres
d DRAWABLE: identifiant du calque sur lequel sera réalisé le tracé
nc INT32: nombre de coordonnées définissant le tracé
          nc= (2 coordonnées, x et y, par point) x nombre de points
np FLOATARRAY: tableau contenant les 2n coordonnées (x et y) des n
               points qui définissent le tracé
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define macouleur '(255 0 0))
macouleur
> (gimp-context-set-foreground macouleur)
(#t)
> (gimp-context-set-brush-size 1)
(#t)
>
(define in "/home/ubuntu/essai/test.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (define c (car (gimp-layer-new img 300 140 RGBA-IMAGE ↯
          "MonCalque" 100 NORMAL-MODE)))
c
> (gimp-image-insert-layer img c 0 0)
(#t)
> (define maligne (make-vector 4 'double))
maligne
> (vector-set! maligne 0 0)
#(0 double double double)
> (vector-set! maligne 1 0)
#(0 0 double double)
> (vector-set! maligne 2 1)
#(0 0 1 double)
> (vector-set! maligne 3 1)
#(0 0 1 1)
> (gimp-pencil c 4 maligne)
(#t)
> (gimp-image-flatten img)
(4)
> (gimp-display-new img)
(1)
```



2.10) Ajouter un masque à une image

2.10.1) Créer le script

Le script suivant montre comment placer automatiquement un masque sur image ouverte dans GIMP, sans modifier les dimensions de cette image. Ce masque est une image transparente quelconque, enregistrée dans un fichier png. Elle est choisie au début de l'exécution du script par l'intermédiaire de la boîte de dialogue associée à ce script. Cette méthode peut être utilisée en particulier pour placer automatiquement une signature ou un copyright personnalisé sur une image.

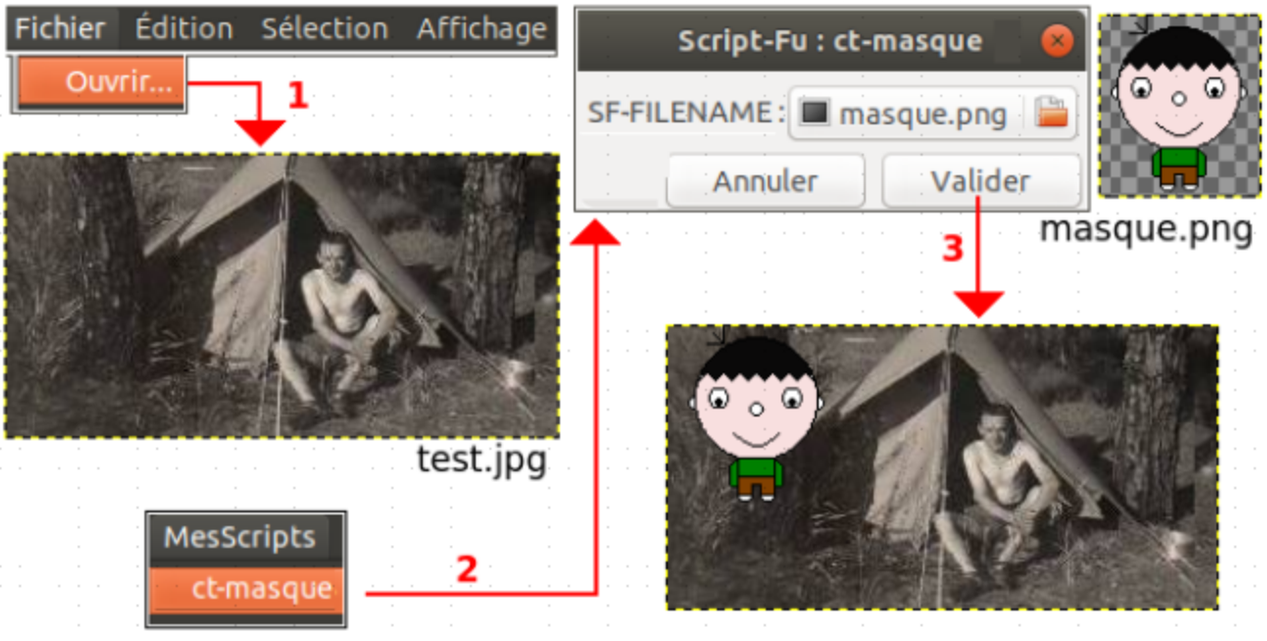
ct-masque.scm

```
;-----------------------------------------------------
; Ce script place un masque sur une image ouverte dans GIMP
; sans modifier les dimensions de cette  image
; ce masque est une image transparante  enregistrée dans un
; fichier .png. Elle est choisie au début de l'exécution du script
; par l'intermédiaire de la boîte de dialogue associée à ce script.
; Auteur : Claude Turrier - 5 mars 2021
; Installation sous Linux:
; sudo cp ct-masque.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------
(define (ct-masque img masq )
(let*
(
(c (car (gimp-file-load-layer 1 img masq)))
)
(gimp-image-add-layer img c 0)
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;-----enregistrer le script -----
(script-fu-register "ct-masque"
"<Image>/MesScripts/ct-masque"
"------------------------------"
"Ajoute un masque au dessus d'une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "img" 0
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/essai/masque.png" ;-> masq
)
```

On suppose ici que le dessin à appliquer se trouve dans le fichier "masque.png", situé dans le répertoire "home/ubuntu/essai".

Le paramètre **SF-IMAGE** et le paramètre **img** associé indiquent que l'image utilisée sera ouverte non pas automatiquement par le script mais manuellement par l'utilisateur depuis le menu Fichier > Ouvrir de Gimp.

Le paramètre **SF-FILENAME** et le paramètre **masq** associé conduisent à l'ouverture automatique, lors de l'exécution du script, d'une boîte de dialogue qui permet de choisir le fichier "masque.png".
La chaine de caractères "/home/ubuntu/essai/masque.png" permet de pré-renseigner directement cette boîte de dialogue avec le chemin complet qui permet de sélectionner directement le fichier.



2.10.2) Fonction gimp-file-load-layer

La fonction **gimp-file-load-layer** permet de charger une image en tant que calque qui vient se placer au dessus d'une image existante.

```
Charger une image en tant que calque
Syntaxe: (gimp-file-load-layer r img f)

Paramètres
m INT32: mode d'exécution {RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1)}
img IMAGE: image au dessus de laquelle va venir se placer le calque
f STRING: chemin complet du fichier à charger en tant que calque

Valeur retournée
c LAYER: identifiant du calque créé contenant l'image chargée

Cette procédure se comporte comme la procédure de chargement de fichier
mais ouvre l'image spécifiée en tant que calque pour une image exis-
```

```
tante. Le calque renvoyé peut être ajouté à l'image existante avec
"gimp-image-insert-layer" ou avec "gimp-image-flatten img"
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in1 "/home/ubuntu/essai/test.jpg")
in1
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
img
> (define in2  "/home/ubuntu/essai/masque.png")
in2
> (define c (car (gimp-file-load-layer 1 img in2)))
c
> (gimp-image-add-layer img c 0)
(#t)
> (gimp-image-flatten img)
(14)
> (gimp-display-new img)
(2)
```

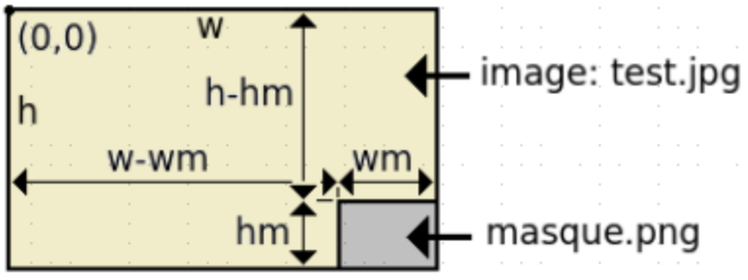
2.10.3) Améliorer le script

On constate que la fonction **gimp-file-load-layer** positionne le masque sur l'image en faisant coincider les coins haut et gauche (0,0) de l'image et du calque. Cependant on peut avoir besoin d'ajouter, comme masque sur l'image, une indication ou une signature qui serait située dans le coin bas et droit par exemple.

Dans ce cas, la solution consiste à ajuster la dimension du calque à celle de l'image, à l'aide de la fonction **gimp-layer-resize-to-image-size**, puis à translater le calque, à l'aide de la fonction **gimp-layer-translate** , avant d'aplatir l'image.

Soient w et h la largeur et la hauteur de l'image (en pixels) et wm et hm la largeur et la hauteur initiales du masque (avec wm<=w et hm<=h).

Pour que le masque viennent se placer exactement dans le coin bas et droit de l'image, il suffit de translater le calque correspondant de w-wm dans le sens des x et de h-hm dans le sens des y.



Le script amélioré est alors le suivant

```
;-----------------------------------------------------
; Ce script place un masque sur une image ouverte dans GIMP
; sans modifier les dimensions de cette  image
; ce masque est une image transparante  enregistrée dans un
; fichier .png. Elle est choisie au début de l'exécution du script
; par l'intermédiaire de la boîte de dialogue associée à ce script.
; Auteur : Claude Turrier - 5 mars 2021
; Installation sous Linux:
; sudo cp ct-masque.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------
(define (ct-masque img masq )
(let*
(
(wm 90) (hm 90)
(c (car (gimp-file-load-layer 1 img masq)))
(w (car (gimp-image-width img))) (h (car (gimp-image-height img)))
)
(gimp-image-add-layer img c 0)
(gimp-layer-resize-to-image-size c)
(gimp-layer-translate c (- w wm) (- h hm))
(gimp-layer-resize-to-image-size c)
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;-----enregistrer le script -----
(script-fu-register "ct-masque"
"<Image>/MesScripts/ct-masque"
"------------------------------"
"Ajoute un masque au dessus d'une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "img" 0
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/essai/masque.png"
)
```

2.10.4) Fonction gimp-layer-resize-to-image-size

La fonction **gimp-layer-resize-to-image-size** redimensionne un calque de façon à ce que ses dimensions soient égales aux dimensions de l'image au-dessus de laquelle il est placé.

Dimensionner un calque aux dimensions de l'image **Syntaxe**: (gimp-layer-resize-to-image-size c)
Paramètres c LAYER: identifiant du calque à dimensionner aux dimensions de l'image Cette procédure redimensionne le calque afin que sa nouvelle largeur et hauteur soient égales à la largeur et à la hauteur de son conteneur d'image.

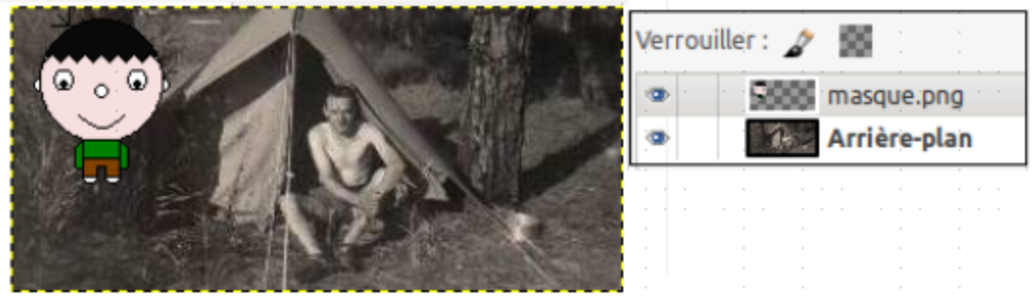
▶ **Exemple**

Soient par exemple les images "test.jpg" et "masque.png" de dimensions 300x140 et 90x90 pixels.

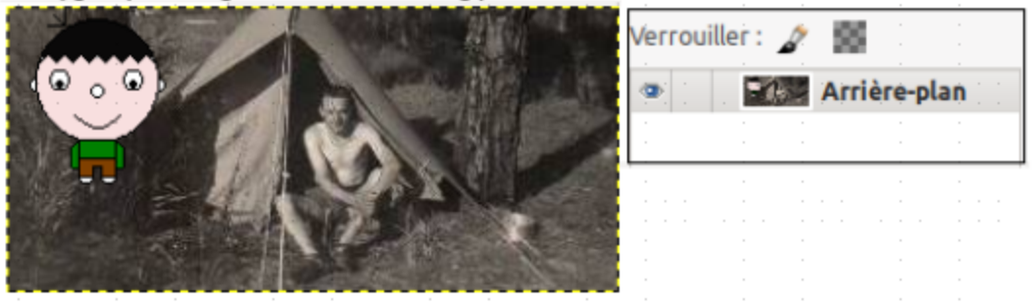
```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in1 "/home/ubuntu/essai/test.jpg")
in1
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
img
> (define in2  "/home/ubuntu/essai/masque.png")
in2
> (define c (car (gimp-file-load-layer 1 img in2)))
c
> (gimp-image-add-layer img c 0)
(#t)
> (gimp-display-new img)
(1)
> (gimp-layer-resize-to-image-size c)
(#t)
> (gimp-image-flatten img)
(8)
```

```
> (gimp-image-add-layer img c 0)
> (gimp-display-new img)
```


```
> (gimp-layer-resize-to-image-size c)
```




```
> (gimp-image-flatten img)
```



2.10.5) Fonction gimp-layer-translate

La fonction **gimp-layer-translate** translate un calque placé au-dessus d'une image.

```
Translater un calque au dessus d'une image
Syntaxe: (gimp-layer-translate c dx dy)
```

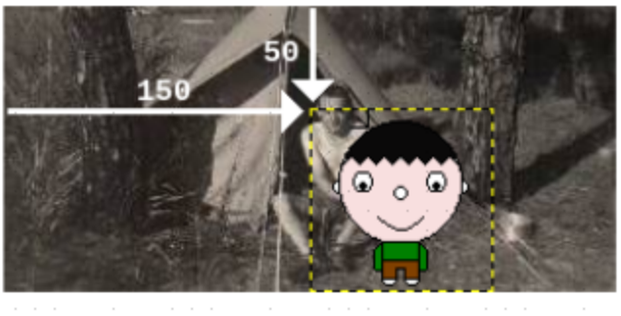
```
Paramètres
c LAYER: identifiant du calque à translater
dx INT32: décalage du calque dans la direction des x
dy INT32: décalage du calque dans la direction des y

Cette commande ne fonctionne que si le calque a été ajouté à une image.
L'origine (0,0) des axes est le coin haut et gauche d'image, l'axe des x
est l'axe horizontal et l'axe des y est l'axe vertical.
```

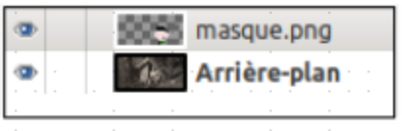
▶ **Exemple**

Soient par exemple "test.jpg" et "masque.png" de dimensions 300x140 et 90x90 pixels.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in1 "/home/ubuntu/essai/test.jpg")
in1
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
img
> (define in2  "/home/ubuntu/essai/masque.png")
in2
> (define c (car (gimp-file-load-layer 1 img in2)))
c
> (gimp-image-add-layer img c 0)
(#t)
> (gimp-layer-translate c 150 50)
(#t)
> (gimp-display-new img)
(1)
> (gimp-layer-resize-to-image-size c)
(#t)
> (gimp-image-flatten img)
(6)
```

```
> (gimp-layer-translate c 150 50)
```





2.10.6) Finaliser le script

Le problème posé par le script ci-dessus est que le masque apposé sur l'image a une taille fixe. Ce masque occupe donc une proportion de surface plus ou moins importante sur l'image, selon la taille de l'image.

Pour que le masque occupe des proportions choisies px et py de la surface d'une image quelles que soient les dimensions w et h de cette image, il faut que le script prennent en compte quelques variables supplémentaires.

On définit donc les variables suivantes mesurées en pixels:

w	largeur de l'image "test.jpg"
h	hauteur de l'image "test.jpg"
wm	largeur du masque "masque.png"
hm	hauteur du masque "masque.png"
r	ratio de la hauteur du masque par rapport à sa largeur r= hm/whm
px	Proportion souhaitée du masque, en largeur, par rapport à la largeur de l'image.
nw	nouvelle largeur du masque nw = px.w
nh	nouvelle hauteur du masque nh = nw.r
dx	Décalage du calque sur l'axe des x dx = w- nw (si on décale le masque en bas et à droite de l'image)
dy	Décalage du calque sur l'axe des y dy = h- nh (si on décale le masque en bas et à droite de l'image)

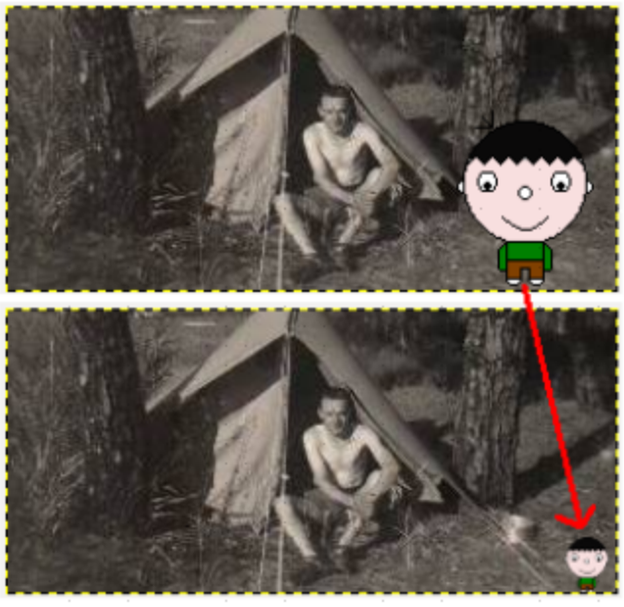
On suppose ci-après que le masque utilisé présente une taille de 90x90 pixels et qu'on souhaite qu'il occupe une surface égale à 10 % de la surface de l'image.
Dans ce cas, si notre image "masque.png" présente une taille de 90x90 pixels et qu'on souhaite qu'elle n'occupe de 10 % de la surface de l'image en largeur, on initialise :

- px=0.1, wm=90 et hm=90

Le script final est alors le suivant :

```
;-----------------------------------------------------
; Ce script place un masque sur une image ouverte dans GIMP
; sans modifier les dimensions de cette  image
; ce masque est une image transparante  enregistrée dans un
; fichier .png. Elle est choisie au début de l'exécution du script
; par l'intermédiaire de la boîte de dialogue associée à ce script.
; Auteur : Claude Turrier - 5 mars 2021
; Installation sous Linux:
; sudo cp ct-masque.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------
```

```
(define (ct-masque img masq )
(let*
(
(px 0.1) (wm 90) (hm 90)
(w (car (gimp-image-width img))) (h (car (gimp-image-height img)))
(r (/ hm wm))
(nw (* px w)) (nh (* nw r))
(dx (- w nw)) (dy (- h nh))
(c (car (gimp-file-load-layer 1 img masq)))
)
(gimp-image-add-layer img c 0)
(gimp-layer-scale c nw nh 0)
(gimp-layer-resize-to-image-size c)
(gimp-layer-translate c dx dy)
(gimp-layer-resize-to-image-size c)
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;-----enregistrer le script -----
(script-fu-register "ct-masque"
"<Image>/MesScripts/ct-masque"
"------------------------------"
"Ajoute un masque au dessus d'une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "img" 0
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/essai/masque.png"
)
```



On peut utiliser une image initiale "masque.png" de n'importe quelle taille, il suffit de préciser cette taille, au début du script ainsi que le pourcentage de largeur relative px qu'on souhaite lui voir occuper en largeur par à l'image réceptrice.

2.11) Exporter tous les calques d'une image

2.11.1) Créer le script

Le script suivant enregistre automatiquement, dans des fichiers images séparés, tous les calques constituant une image source. L'image source (img0) peut être une image existante, ouverte dans gimp puis modifiée ou non, ou une image entièrement créée dans gimp.

```
;-----------------------------------------------------
; ct-scalques.scm
; Ce script enregistre tous les calques d'une image
; Auteur : C.Turrier - mars 2021
; Fonctionnement:
; On ouvre une image dans gimp, par l'intermédiaire du menu
; Fichier > Nouveau ou Fichier > Ouvrir:
; Le script enregistre les calques dans le répertoire
; "/home/ubuntu/essai/"
; lc: liste des calques consituant l'image source
; cc: calque courant (calque  en cours de traitement, dans la liste)
; imgsauv: id de l'image associée au calque courant enregistré
; out: chemin+nom de fichier pour le calque courant à enregistrer
; Installation sous Linux:
; sudo cp ct-scalques.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------
(
define (ct-scalques img0)
;-->début let1
(let* ((chemin  "/home/ubuntu/essai/image_"))
;-->début letloop
(let loop ((lc (vector->list (cadr (gimp-image-get-layers img0)))))
;-->début unless
(unless (null? lc)
(gimp-edit-copy (car lc))
;-->début let2
(let* (
(nom (car (gimp-item-get-name (car lc))))
(out (string-append chemin nom ".png"))
(imgsauv  (car (gimp-edit-paste-as-new)))
(cc (aref (cadr (gimp-image-get-layers imgsauv)) 0)))
(file-png-save 1 imgsauv cc out out 0 0 0 0 0 0 0)
);<--fin let2
(loop (cdr lc)); retour vers  début unless
))));<--fin unless, letloop, let1, define
;
(script-fu-register "ct-scalques"
"<Image>/MesScripts/ct-scalques"
"-----aa-------------------------"
"Enregistre tous les calques"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "Image" 0 ;img0
)
```

Le script fonctionne de la façon suivante.

On établit une liste **lc** des identifiants des calques constituant l'image source (l'image source possède l'identifiant **img0**).

Avec l'instruction **unless**, on parcourt cette liste et, pour chacun des calques rencontrés dans la liste, on crée une image courante (d'identifiant **imgsauv**) à laquelle on associe un calque courant (d'identifiant **cc**) qui possède le même contenu que le calque rencontré.

Le fonctionnement des différentes fonctions utilisées dans le script est détaillé ci-après.

2.11.5) Structure let loop

La structure **let loop** permet de répéter un ensemble d'instructions en boucle, en faisant varier, en fin de boucle, une variable qu'on réinjecte au début de la première instruction de la boucle (juste après la partie définition des variables).

```
Répéter en boucle des instructions.
Syntaxe:
(let loop ((var valeur)...)(expr1)(expr2)...(exprn) (loop (x))
```

```
Paramètres
((var valeur)...): variables et valeurs associées
(expr1): instruction 1
…
(exprn): instruction n
(loop (x)): retour en début d'insruction1 en injectant une valeur x
```

```
(
let loop
;---définitions
((var1 val1) (var2 val2)... )
;
;---instructions
(unless (condition); instruction 1
(instruction 2)
...
(instruction n)
(loop (vari vali))
; retour au début de l'instruction 1
) ; fin unless
) ; fin letloop
```

2.11.4) Fonction vector->list

La fonction **vector->list** convertit un vecteur v (c'est à dire un tableau) en une liste qui contient les même valeurs que ce tableau.

```
Convertir un vecteur v (c'est à dire un tableau) en une liste qui
contient les mêmes valeurs que ce tableau
Syntaxe: (vector->list v)
```

```
Paramètres
v VECTOR: tableau d'éléments

Valeurs retournées (dans une liste)
li LIST: liste contenant les éléments du tableau
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (vector->list #(1 2 3))
(1 2 3)
> (vector 0 1 2 3 4)
#(0 1 2 3 4)
> (define v (vector 1 2 3 4))
v
> v
#(1 2 3 4)
> (vector->list  v)
(1 2 3 4)
```

2.11.8) Fonction gimp-image-get-layers

La fonction **gimp-image-get-layers** retourne une liste contenant le nombre des calques constituant l'image en cours ainsi que les identifiants de ces calques

```
Retourner un tableau contenant le nombre et des identifiants des calques
constituant l'image en cours
Syntaxe: (gimp-image-get-layers img)
```

```
Paramètres
img IMAGE: identifiant de l'image en cours

Valeurs retournées (dans une liste)
n INT32: nombre de calques constituant l'image
ids INT32ARRAY: tableau des identifiants de calques constituant l'image

L'ordre des calques, dans le tableau, va du haut vers le bas.
```

▶ **Exemple**

Soit l'image "/home/ubuntu/essai/test.jpg" présentant une dimension de 300 x140 pixels.
On ouvre cette image dans gimp et on lui ajoute deux calques c1 et c2.
On constate que gimp va attribuer l'identifiant 1 à l'image img0 et les identifiants 2, 3 et 4 aux trois calques associés à l'image.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img0 (car (gimp-file-load 1 in in)))
img0
> (define c1 (car (gimp-layer-new img0 300 140 1 "c1" 100 0)))
c1
> (gimp-image-insert-layer img0 c1 0 0)
(#t)
> (define c2 (car (gimp-layer-new img0 300 140 1 "c2" 100 0)))
c2
> (gimp-image-insert-layer img0 c2 0 0)
(#t)
> (gimp-display-new img0)
(1)
> (define calques (gimp-image-get-layers img0))
calques
> calques
(3 #(4 3 2))
> (define lc (vector->list (cadr calques)))
lc
> lc
(4 3 2)
> (car lc)
4
> (gimp-edit-copy (car lc))
(1)
> (define imgsauv  (car (gimp-edit-paste-as-new)))
imgsauv
> (define cc (aref (cadr (gimp-image-get-layers imgsauv)) 0))
cc
> cc
6
```

```
> (gimp-display-new img)
```

⚑ **Remarque**
On peut noter au passage que plutôt que de saisir les instructions une par une dans la console TinyScheme, comme on l'a fait jusqu'à présent, on peut, pour aller plus vite, entrer un bloc complet d'instructions comme ci-après puis examiner ensuite le contenu des variables.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
(define img0 (car (gimp-file-load 1 in in)))
(define c1 (car (gimp-layer-new img0 300 140 1 "c1" 100 0)))
(gimp-image-insert-layer img0 c1 0 0)
(define c2 (car (gimp-layer-new img0 300 140 1 "c2" 100 0)))
(gimp-image-insert-layer img0 c2 0 0)
(gimp-display-new img0)
(define calques (gimp-image-get-layers img0))
(define lc (vector->list (cadr calques)))
(gimp-edit-copy (car lc))
(define imgsauv  (car (gimp-edit-paste-as-new)))
(define cc (aref (cadr (gimp-image-get-layers imgsauv)) 0))

inimg0c1(#t)c2(#t)(1)calqueslc(1)imgsauvcc
> imgsauv
2
> lc
(4 3 2)
> cc
6
> img0
1
```

2.11.6) Mots when et unless

En scheme, on utilise le mot **when** lorsque on souhaite utiliser un **if** sans le faire suivre d'un **else** et qu'on a besoin d'inclure plusieurs instructions à l'intérieur de ce **if**.

On utilise le mot **unless** de la même façon que **when** mais lorsque la condition utilisée est de type non (if not).

▶ **Exemple**
```
(unless (null? s)...)    tant que s n'est pas égale à ""
```

2.11.7) Condition string=?

La condition **string=?** permet de tester des chaînes de caractères.

`(string=? x)`	retourne true si x est une chaîne de caractères (string)
`(string=? x "toto")`	retourne true si x est une chaîne de caractères égale à "toto"

▶ **Exemple**

```
  Bienvenue sur TinyScheme

> (define x "toto")
x
> (string=? x "toto")
#t
```

2.11.9) Fonction gimp-edit-copy

La fonction **gimp-edit-copy** copie, dans le presse-papiers, le contenu du calque dessinable spécifié.

Copier le contenu d'un calque dans le presse-papiers **Syntaxe**: (gimp-edit-copy id)
Paramètres id DRAWABLE: identifiant du calque dessinable dont on veut copier le contenu dans le presse-papiers Valeur retournée n INT32: TRUE si la copie a réussi et False si le calque ne contenait rien à copier (calque vide) Le contenu copié peut ensuite être récupéré en utilisant la commande **gimp-edit-paste**. La fonction échouera si la zone sélectionnée se trouve complètement en dehors des limites du dessin courant et qu'il n'y a rien à copier.

▶ **Exemple**

On utilise l'image "test.jpg" de 300x140px, citée précédemment, qui se trouve dans le répertoire "/home/ubuntu/essai".

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
in
> (define img0 (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img0
> (define c1 (car (gimp-layer-new img0 300 140 RGBA-IMAGE "c1" 100
0)))
c1
> (gimp-image-insert-layer img0 c1 0 0)
(#t)
> (define c2 (car (gimp-layer-new img0 300 140 RGBA-IMAGE "c2" 100
0)))
c2
> (gimp-image-insert-layer img0 c2 0 0)
(#t)
> (gimp-display-new img0)
(1)
> (define calques (gimp-image-get-layers img0))
```

```
calques
> calques
(3 #(4 3 2))
> (define c (vector->list (cadr (gimp-image-get-layers img0))))
c
> c
(4 3 2)
> (gimp-edit-copy (car c))
(1)
```

Si on dessine sur le calque du dessus, avec le pinceau, avant l'instruction (gimp-edit-copy (car c)), puis qu'on efface le contenu de ce calque, un copier-coller depuis le menu de Gimp le restituera.

2.11.11) Fonction gimp-item-get-name

La fonction **gimp-item-get-name** retourne le nom de l'objet dont l'identifiant est passé en paramètre. Cette fonction remplace la fonction **gimp-item-get-name** devenue obsolète.

Obtenir le nom d'un objet, sous la forme d'une chaîne de caractères. **Syntaxe**: (gimp-item-get-name)
Paramètres item ITEM: objet Valeurs retournées nom STRING: nom de l'objet

▶ **Exemple**

On utilise l'image "test.jpg" citée précédemment.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
(define img0 (car (gimp-file-load 1 in in)))
(define c1 (car (gimp-layer-new img0 300 140 1 "c1" 100 0)))
(gimp-image-insert-layer img0 c1 0 0)
(define c2 (car (gimp-layer-new img0 300 140 1 "c2" 100 0)))
(gimp-image-insert-layer img0 c2 0 0)
(gimp-display-new img0)
(define calques (gimp-image-get-layers img0))
(define c (vector->list (cadr (gimp-image-get-layers img0))))
(define nom (car (gimp-item-get-name (car c))))

inimg0c1(#t)c2(#t)(1)calquescnom#<EOF>
> calques
(3 #(4 3 2))
> c
(4 3 2)
> nom
"c2"
```

```
L'image et les calques s'affichent dans la fenêtre de gimp aussitôt
après l'instruction (gimp-display-new img0)
```

2.11.12) Fonction gimp-edit-paste-as-new

La fonction **gimp-edit-paste-as-new** copie, le contenu du presse-papiers dans une nouvelle image.

Copier le contenu du presse-papiers dans une nouvelle image **Syntaxe**: (gimp-edit-paste-as-new)
Valeur retournée img IMAGE: identifiant de la nouvelle image Cette procédure colle une copie du contenu du presse-papiers interne de GIMP dans une nouvelle image.

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/test.jpg")
(define img0 (car (gimp-file-load 1 in in)))
(define c1 (car (gimp-layer-new img0 300 140 1 "c1" 100 0)))
(gimp-image-insert-layer img0 c1 0 0)
(define c2 (car (gimp-layer-new img0 300 140 1 "c2" 100 0)))
(gimp-image-insert-layer img0 c2 0 0)
(gimp-display-new img0)
(define calques (gimp-image-get-layers img0))
(define lc (vector->list (cadr calques)))
(gimp-edit-copy (car lc))
(define imgsauv  (car (gimp-edit-paste-as-new)))
(define cc (aref (cadr (gimp-image-get-layers imgsauv)) 0))

inimg0c1(#t)c2(#t)(1)calqueslc(1)imgsauvcc

> imgsauv
2
> cc
6
```

2.11.13) Fonction file-png-save

La fonction file-png-save permet d'enregistrer une image dans un fichier au format png.

Enregistrer une image chargée en mémoire dans un fichier png
Syntaxe:
```
(file-png-save rm img c out out i c b g o p t  )
```

Paramètres

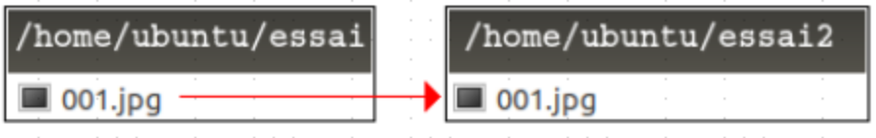
```
rm INT32: run-mode - mode d'éxécution de la fonction
          { RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1) }
img IMAGE: identificateur de l'image mémoire à enregistrer
c DRAWABLE: Calque associé à l'image et dont le contenu est
            à enregistrer
out STRING: chemin complet du fichier à enregistrer
out STRING: chemin complet du fichier à enregistrer
i INT32: interlacement Adam7 {oui (1) non(0))
c INT32: niveau compression de l'image (0<=c<=9)
b INT32: écriture d'un bloc bkgd {oui (1) non(0))
g INT32: écriture d'un bloc gama {oui (1) non(0))
o INT32: écriture d'un bloc offs {oui (1) non(0))
p INT32: écriture d'un bloc phys {oui (1) non(0))
t INT32: écriture d'un bloc time {oui (1) non(0))t
```

⚑ **Remarque**

L'entrelacement n'est pas nécessaire. C'est une technique qui permet aux navigateurs Web d'afficher rapidement une version basse qualité de l'image. Celle-ci devient progressivement plus claire au fur et à mesure de son chargement complet. Les paramètres suivants peuvent être laissés à 0.

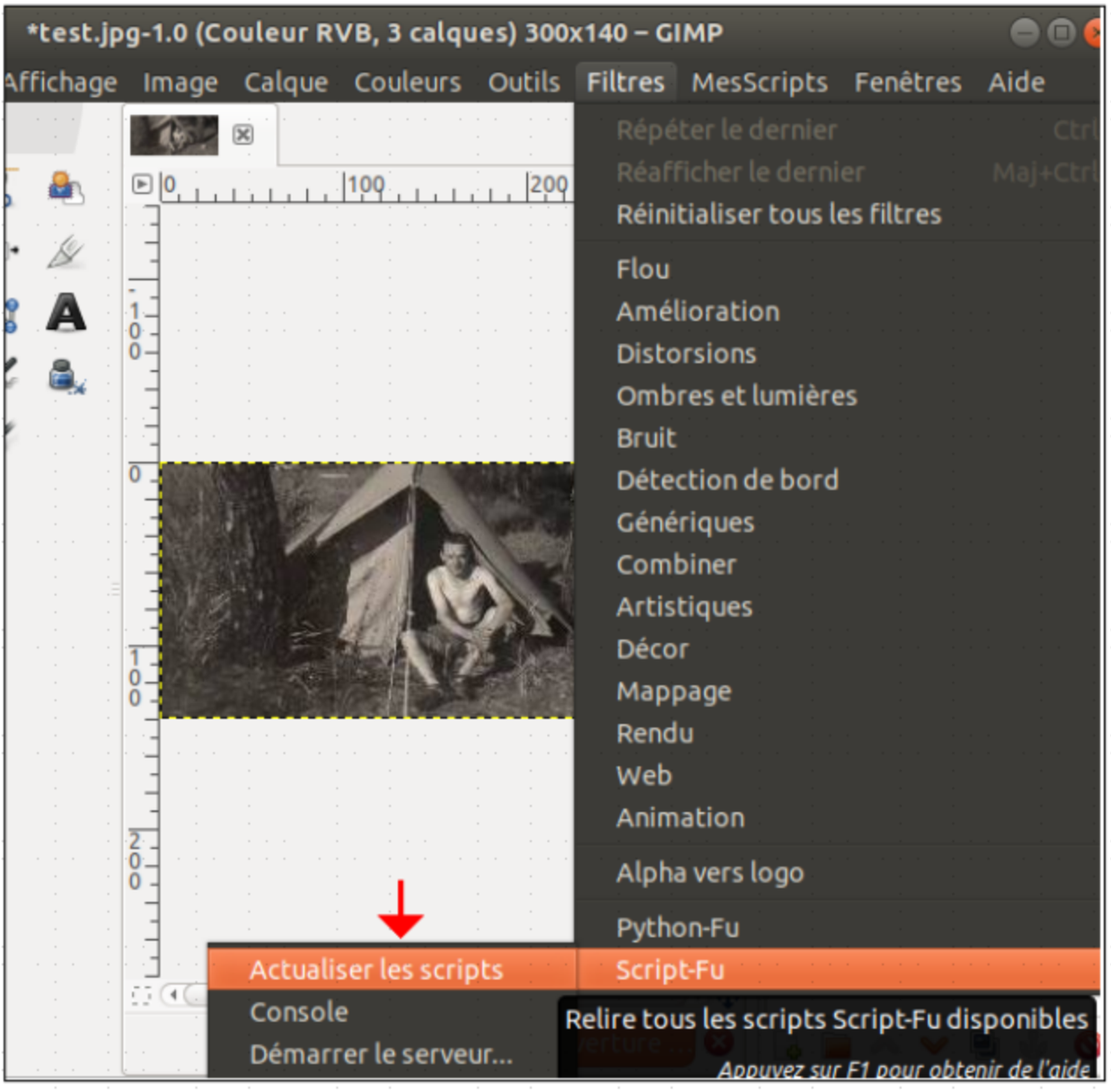
▶ ***Exemple***

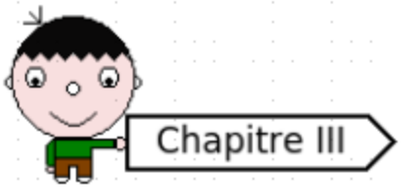
```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/001.png")
in
> (define img (car (gimp-file-load 1 in in)))
img
> (define out "/home/ubuntu/essai2/new001.png")
out
> (define c (car (gimp-image-get-active-drawable img)))
c
> (file-png-save 1 img c out out 0 0 0 0 0 0 0)
(#t)
```



⚑ **Remarque**

Lorsqu'on modifie un script ("monscript.scm" par exemple) puis qu'on l'enregistre à nouveau dans le répertoire des scripts de Gimp, ce script est naturellement pris en compte par Gimp si on ferme puis qu'on redémarre Gimp.
Il existe cependant une méthode plus rapide. Il suffit de cliquer le menu **Filtres > Script-Fu > Actualiser les scripts** et Gimp met aussi-tôt à jour sa base de scripts. On peut alors immédiatement utiliser le script modifié sans avoir à quitter Gimp.





Chapitre III
Traiter des lots d'images

Le traitement par lot consiste à appliquer un traitement non pas à une seule image mais à un ensemble d'images contenues dans un répertoire.

3.1) Utiliser file-glob

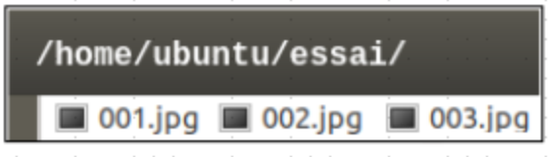
Pour programmer des traitements par lot, Gimp met à la disposition du programmeur une fonction très intéressante : la fonction **file-glob**.
Cette fonction peut être avantageusement utilisée dans les scripts qui effectuent des traitements sur des lots d'images mais elle ne peut gérer que des modèles simples comme "/home/toto/essai/*.jpg".

```
Retourner le nombre et la liste des fichiers qui correspondent au motif qui est passé en paramètre
Syntaxe: (file-glob pattern encoding)
```

```
Paramètres
pattern STRING : Motif global du chemin complet des images (codé UTF-8)
encoding INT32 : Encodage des noms renvoyés:
                 {UTF-8 (0), filename encoding (1) }
Valeurs retournées
num-files INT32 : Nombre de noms renvoyés
files STRINGARRAY : Liste des noms qui correspondent
```

▶ **Exemple**
Supposons qu'on ait placé 3 fichiers images 001.jpg, 002.jpg et 003.jpg dans le répertoire "/home/ubuntu/essai". Les lignes suivantes, saisies dans la console Script-Fu de Gimp retournent le nombre et la liste de ces fichiers.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define pattern "/home/ubuntu/essai/*.jpg")
pattern#<EOF>
> (display (file-glob pattern 1) )
(3 (/home/ubuntu/essai/002.jpg /home/ubuntu/essai/003.jpg
/home/ubuntu/essai/001.jpg))#t
> (display (file-glob pattern 0) )
(3 (/home/ubuntu/essai/002.jpg /home/ubuntu/essai/003.jpg
/home/ubuntu/essai/001.jpg))#t
```



3.2) Utiliser gimp-file-load

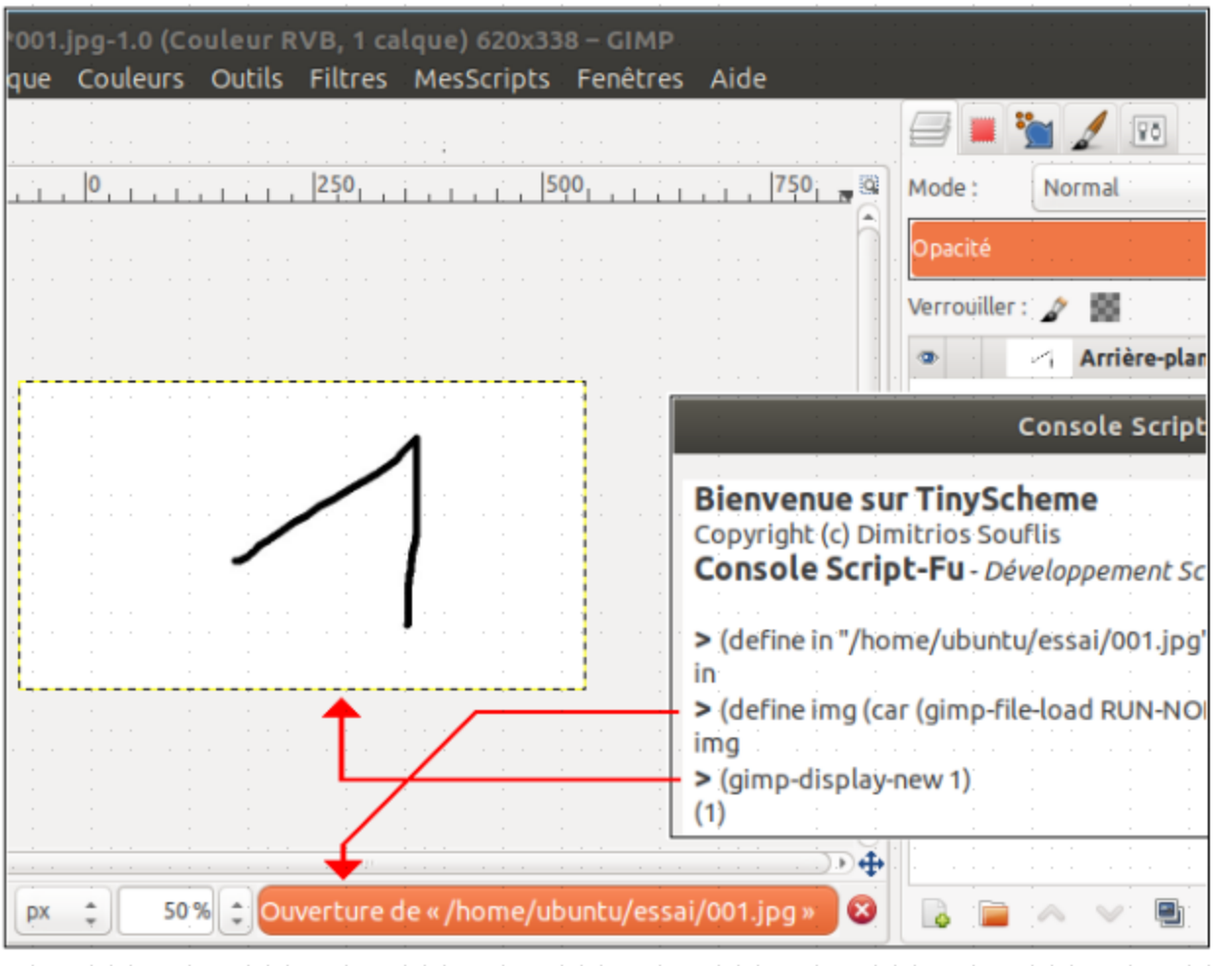
La fonction **gimp-file-load** permet de charger une image en mémoire. Le paramètre est le chemin complet d'accès à ce fichier.

```
Charger en mémoire l'image contenue dans un fichier sur le disque dur
Syntaxe: (gimp-file-load run-mode filename raw-filename)
```

```
Paramètres
run-mode INT32 : mode d'exécution de la fonction
                 {RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1)
filename STRING: chemin complet du fichier image
raw-filename STRING: chemin complet du fichier image
Valeurs retournées
image IMAGE : identificateur de l'image en mémoire
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (gimp-display-new 1)
(1)
```



3.3) Utiliser gimp-image-get-name

La fonction **gimp-image-get-name** permet d'obtenir le nom de fichier d'une image chargée en mémoire. La valeur retournée par cette fonction n'est pas le chemin complet d'accès à ce fichier mais seulement le nom de ce fichier.

Retourner, dans une liste, le nom de fichier d'une image en mémoire **Syntaxe**: (gimp-image-get-name image)
Paramètres image IMAGE : identificateur de l'image chargée en mémoire Valeurs retournées name STRING : nom de fichier de l'image en mémoire

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (gimp-display-new 1)
(1)
> (define nom (car (gimp-image-get-name img)))
nom
> (display nom)
001.jpg#t
```

⚑ **Remarque**

L'utilisation de la fonction **car**, pour récupérer uniquement le nom du fichier, est rendue nécessaire par le fait que la fonction **gimp-image-get-name** retourne le nom sous forme de liste. C'est pourquoi, si on utilise uniquement la fonction **gimp-image-get-name** (sans utiliser la fonction car), pour récupérer le nom du fichier, on obtient celui-ci entre parenthèses, ce qui n'est pas le résultat recherché.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load 1 in in)))
img
> (define nom (gimp-image-get-name img))
nom
> (display nom)
(001.jpg)#t
```

⚑ **Rappel**

dans gimp-file-load: {RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1)

3.4) Utiliser gimp-image-get-active-drawable

La fonction **gimp-image-get-active-drawable** permet de récupérer l'identifiant (simple numéro) du calque actif d'une image chargée en mémoire

Récupérer, l'identifiant du calque actif d'une image chargée en mémoire **Syntaxe**: (gimp-image-get-active-drawable image)
Paramètres image IMAGE : identificateur de l'image chargée en mémoire Valeurs retournées drawable DRAWABLE : identificateur du calque actif de l'image

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (gimp-display-new 1)
(1)
> (define nom (car (gimp-image-get-name img)))
nom
> (display nom)
001.jpg#t
> (define c (car (gimp-image-get-active-drawable img)))
c
> (display c)
2#t
```

3.5) Utiliser gimp-image-delete

La fonction **gimp-image-delete** permet de supprimer une image en mémoire. Cette image sera supprimée en mémoire même si elle n'est pas, ou n'a pas été, affichée à l'écran.

Supprimer une image en mémoire **Syntaxe**: (gimp-image-delete image)
Paramètres image IMAGE : identifiant de l'image, chargée en mémoire, à supprimer

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (gimp-image-delete img)
(#t)
```

3.6) Utiliser file-jpeg-save

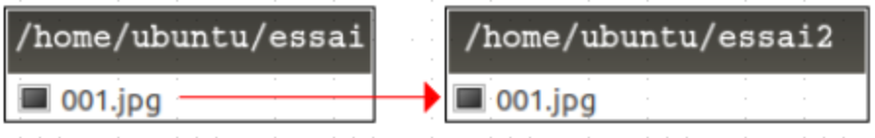
La fonction **file-jpeg-save** permet d'enregistrer, dans un fichier au format jpg, une image se trouvant en mémoire.

```
Enregistrer une image chargée en mémoire dans un fichier jpg
Syntaxe:
(file-jpeg-save rm img c out out q n o p c s b m  )
```

```
Paramètres
rm INT32: run-mode - mode d'éxécution de la fonction
                  { RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1) }
img IMAGE: identificateur de l'image mémoire à enregistrer
c DRAWABLE: Calque associé à l'image dont le contenu est à sauver
out STRING: chemin complet du fichier à enregistrer
out STRING: chemin complet du fichier à enregistrer
q FLOAT: qualité d'enregistrement de l'image jpg (0<=q<=1)
n FLOAT: niveau de lissage (smooth) de l'image (0<=s<=1)
o INT32: codage de Huffman optimisé (1) ou non (0)
p INT32: codage de type image progressive (1) ou non(0)
c STRING:commentaire
s INT32: type de sous échantillonnage
           0 == 4:2:0 (chroma quartered),
           1 == 4:2:2 Horizontal (chroma halved),
           2 == 4:4:4 (best quality),
           3 == 4:2:2 Vertical (chroma halved)
b INT32: création d'une baseline (0) ou non (1)
m INT32: avec marqueurs pour redémarrage en cas d'erreur (1) ou non (0)
d INT32: méthode DCT utilisée {INTEGER (0), FIXED (1), FLOAT (2)}
```

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
> (define in "/home/ubuntu/essai/001.jpg")
in
> (define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
img
> (define out "/home/ubuntu/essai2/001.jpg")
out
> (define c (car (gimp-image-get-active-drawable img)))
c
> (file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0
0)
(#t)
```



3.7) Utiliser file-png-save

Rappel (cf 2.11.13) - La fonction **file-png-save** permet d'enregistrer, dans un fichier au format png, une image se trouvant en mémoire.

```
Enregistrer une image chargée en mémoire dans un fichier png
Syntaxe:
(file-png-save rm img c out out i c b g o p t  )
```

Paramètres

```
rm INT32: run-mode - mode d'éxécution de la fonction
                  {RUN-INTERACTIVE (0), RUN-NONINTERACTIVE (1)}
img IMAGE: identificateur de l'image mémoire à enregistrer
c DRAWABLE: Calque associé à l'image dont le contenu est à sauver
out STRING: chemin complet du fichier à enregistrer
out STRING: chemin complet du fichier à enregistrer
i INT32: interlacement Adam7 {oui (1) non(0))
c INT32: niveau compression de l'image (0<=c<=9)
b INT32: écriture d'un bloc bkgd {oui (1) non(0))
g INT32: écriture d'un bloc gama {oui (1) non(0))
o INT32: écriture d'un bloc offs {oui (1) non(0))
p INT32: écriture d'un bloc phys {oui (1) non(0))
t INT32: écriture d'un bloc time {oui (1) non(0))
```

⚑ **Remarque**

L'entrelacement est une technique qui permet aux navigateurs Web d'afficher rapidement une version basse qualité de l'image qui devient progressivement plus claire au fur et à mesure de son chargement. Les paramètres suivants peuvent être laissés à 0.

▶ **Exemple**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in "/home/ubuntu/essai/001.png")
in
> (define img (car (gimp-file-load 1 in in)))
img
> (define out "/home/ubuntu/essai2/new001.png")
out
> (define c (car (gimp-image-get-active-drawable img)))
c
> (file-png-save 1 img c out out 0 0 0 0 0 0 0)
(#t)
```

3.8) Créer une ossature de traitement par lot

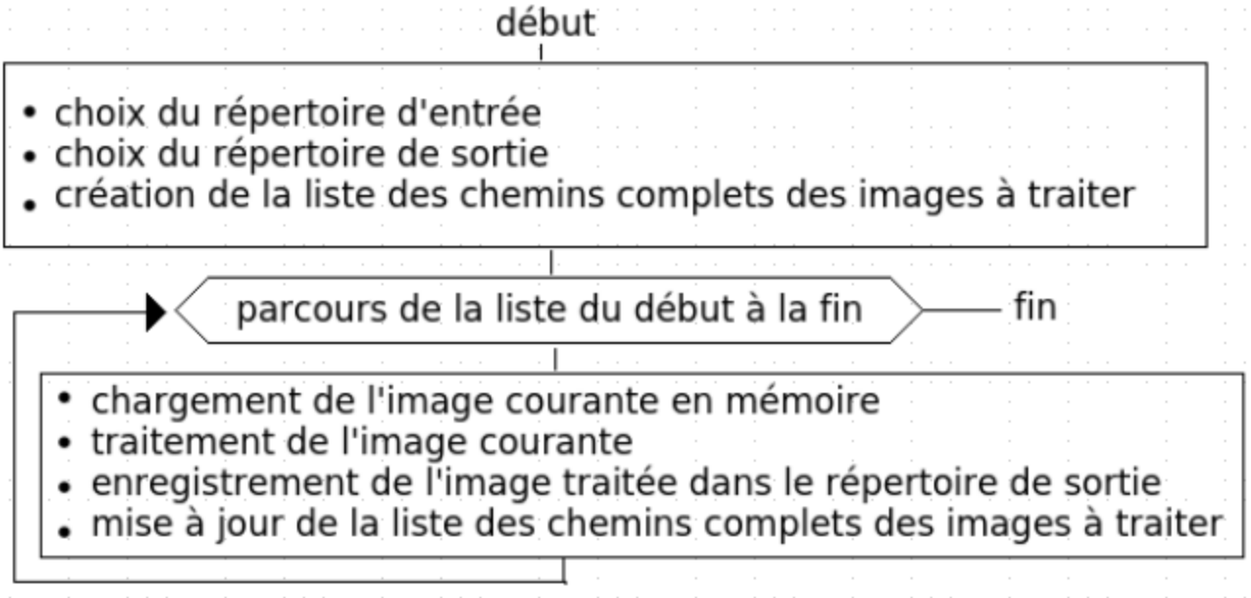
Il est intéressant de commencer par créer une ossature de traitement d'images par lot. Cette ossature de code pourra être ensuite être utilisée à l'identique, pour n'importe quel lot d'images, quel que soit le traitement d'image particulier mis en oeuvre.

Le script suivant forme une ossature de traitement pouvant s'appliquer à un lot d'images. Il assure les tâches suivantes:

- Choisir le répertoire d'entrée où se trouve les images à traiter ;
- Choisir le répertoire de sortie, pour enregistrer les images traitées ;
- Etablir la liste des chemins complets des images à traiter ;
- Parcourir cette liste du début à la fin afin de :
  - > charger l'image courante en mémoire ;
  - > traiter cette image ;
  - > enregistrer l'image traitée dans le répertoire de sortie ;
  - > mettre à jour la liste des chemins complets des images à traiter.

Dans le cas présent, le traitement effectué sur chaque image est réduit à sa plus simple expression. Il effectue simplement une rotation de 90° (soit 1,5708 radians) de l'image.

Il appartient au programmeur de placer, au sein de l'ossature, à l'emplacement indiqué en commentaires dans le code, les lignes de code qui correspondent au traitement de son choix. Celui-ci sera alors automatiquement appliqué au lot d'images considéré.



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Fichier: tlot.scm
;; Action:  traitement d'un lot d'images contenues dans un dossier r1
;;          et copie des images traitées dans un dossier r2
;; Install: sudo cp tlot.scm /usr/share/gimp/2.0/scripts/
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;============================================================
;; CHARGER L'IMAGE COURANTE EN MEMOIRE
;;============================================================
;; tlot:nom du script | r1 et r2: répertoires d'entrée et de sortie
(define (tlot r1  r2)
;; liste:liste des chemins complets des images à traiter
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))))
;; parcours de la liste, du début à la fin
(while (not (null? liste))
;; in:chemin complet de l'image courante à traiter
(let* ((in (car liste))
;; img:image courante à traiter, chargée en mémoire
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
;; out:chemin complet de l'image courante à enregistrer
(out (string-append r2 "/" (car (gimp-image-get-name img))))
;; c:calque actif courant
(c (car (gimp-image-get-active-drawable img))))
;;================================================================
;; TRAITER L'IMAGE COURANTE
;; placer ici le code du traitement à appliquer
;; à chacune des images
;;================================================================
(gimp-rotate c 1 1.5708 )
;;================================================================
;; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;;================================================================
;; enregistrement de l'image courante après traitement
(file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0 0)
;; fermeture de l'image courante
(gimp-image-delete img)
)
;; mise à jour de la liste des images à traiter
(set! liste (cdr liste))
)))
;;================================================================
;; ENREGISTREMENT DU SCRIPT
;;================================================================
(
script-fu-register "tlot"
"<Image>/MesScripts/tlot"
"Traitement par lot"
"*********"
"CT"
"2021"
""  ;types d'images
SF-DIRNAME  "Répertoire d'entrée"   ""  ;r1
SF-DIRNAME  "Répertoire de sortie"  "" ;r2
)
```

La partie utile du code (hors commentaire) est la suivante

```
(define (tlot r1  r2)
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))))
(while (not (null? liste))
(let* ((in (car liste))
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
)
(gimp-rotate c 1 1.5708 )
(file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
)
(set! liste (cdr liste))
)))
```

⚑ **Remarque**

Il faut vérifier, au niveau du code, que:

- Le nombre de parenthèses fermantes est égal au nombre de parenthèses ouvrantes;
- pour chaque instruction, la syntaxe et le nombre de parenthèses associées sont bien respectés.

```
❶ (define ()                ;-> début define
❷ (let*((liste(()))))       ;-> début let1
❸ (while(( ))               ;-> début while
❹   (let* ((in()) (img(())) (out((()))))(c(()))) ;->debut let2 )
    );-> fin let2
❺   (gimp-rotate c 1 1.5708 )
❻   (file-jpeg-save)
❼   (gimp-image-delete))
❽   (set! liste())
❾))) ;-> fin while, fin let1, fin define
```

1- Définit la procédure **tlot** et les paramètres **r1** et **r2** associés.
2- La seconde instruction définit la variable **liste** ainsi que le traitement à effectuer. Ce traitement commence à la ligne suivante et se termine à la parenthèse **fin let1**.
3- La troisième instruction est une boucle **while** qui teste une condition et effectue le traitement associé tant que cette condition est satisfaite. Ce traitement commence à la ligne suivante et se termine à la parenthèse **fin while**.
4- La quatrième instruction permet de définiir les variables **in**, **img**, **out** et **c** associées à l'image courante de la liste.
5- La cinquième instruction fait pivoter l'image courante de 90°.
6- La sixième instruction enregistre, dans le répertoire **r2**, l'image courante pivotée de 90°.
7- La septième instruction supprime l'image courante en mémoire.
8- La huitième instruction met à jour la liste des chemins complets des images restant à traiter et, tant que cette liste n'est pas vide, revient à l'instruction trois.

3.9) Redimensionner un lot d'images

On utilise l'ossature de traitement par lot vue en **3.8** et le script de modification de l'échelle d'une image vue en **2.7** pour créer le script suivant qui redimensionne un lot d'images.

```
;-----------------------------------------------------------------
; Fichier: ct-lotredim.scm
; Action:  Redimensionne un lot d'images contenues dans un dossier r1
;          et copie des images redimensionnées dans un dossier r2
; Install: sudo cp ct-lotredim.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotredim r1  r2);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(nh 0) (k 0)(nw 100)
(h (car (gimp-image-height img)))
(w (car (gimp-image-width img)))
);fin defvar let2
;-----------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------
(set! k (/ w h))
(set! nh (/ nw k))
(gimp-image-scale img nw nh)
;-----------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;-----------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-----------------------------------------------------------
(
script-fu-register "ct-lotredim"
"<Image>/MesScripts/ct-lotredim"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
)
```

Le script effectue les tâches suivantes :

- Il établit la liste des chemins complets de fichiers qui se trouvent dans le répertoire r1
- Il parcourt cette liste et pour chaque fichier de cette liste :
  > il charge l'image correspondante en mémoire ;
  > il la redimensionne ;
  > il l'enregistre dans le répertoire r2.

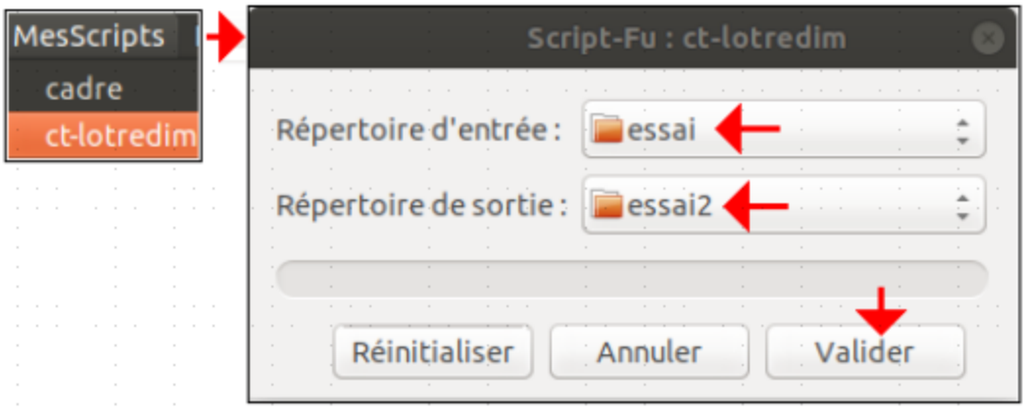
Les variables utilisées sont les suivantes :

r1	: répertoire où se trouvent les fichiers images sources
r2	: répertoire où sont enregsitrées les images redimensionnées
liste	: liste des chemins complets des images à traiter
in	: chemin complet de l'image courante à traiter
img	: Identifiant de l'image courante à traiter, chargée en mémoire
out	: chemin complet de l'image courante à enregistrer
c	: identifiant du calque actif courant
w	: largeur de l'image courante (px)
h	: hauteur de l'image courante (px)
nh	: nouvelle hauteur de l'image redimensionnée
nw	: nouvelle largeur de l'image redimensionnée
k	: ratio de l'image courante k = w/h = nw/nh

Le fonctionnement des différentes instructions et fonctions a été détaillé en 2.7) et en 3.8).

Les instructions suivantes positionnent (par défaut, dans la boîte de dialogue automatique correspondante associée au script) les répertoires r1 et r2 en "/home/ubuntu/essai" et "/home/ubuntu/essai".

```
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
```



Ce script fixe la largeur des images redimensionnées à nw=100 et le niveau de qualité du codage jpg à 1.

En pratique, il serait intéressant de pouvoir choisir ces deux paramètres de façon interactive en début d'exécution du script.

On va donc modifier le script en lui passant en paramètres les variables suivantes :

```
nw INT32: largeur en pixels de l'image redimensionnée
q FLOAT : niveau de qualité du fichier jpg  (0<= q <=1)
```

Pour cela on associe le type de paramètre SF-VALUE à la variable nw et le type de paramètre SF-ADJUSTMENT à la variable q.
Ces paramètres conduisent à la création automatique des zones de dialogue correspondantes, dans la boîte de dialogue associée au script.

Le paramètre SF-VALUE permet de saisir directement un nombre et le paramètre SF-ADJUSTMENT permet de saisir un nombre à l'aide d'un widget de type glissière.

Syntaxe
```
SF-VALUE "info" "n"
```

```
"info": Texte affiché devant la zone de dialogue.
n : nombre affiché par défaut
```

Syntaxe
```
SF-ADJUSTMENT "info" '(def min max i pi d t)
```

```
"info": Texte affiché devant la zone de dialogue.
def : valeur affichée par défaut
min : valeur minimum
max : Valeur maximum
i : valeur d'incrément
pi: valeur d'incrément ( touches Page)
d : nombre de chiffres décimaux après le point
t : type SF-SLIDER (0)  ou SF-SPINNER (1)
```

► **Exemple**

```
SF-VALUE "Nouvelle largeur" "100"
SF-ADJUSTMENT  "Qualité"     '(1 0 1 0.1 1 1 1)
```

Le script final obtenu est le suivant. Il permet de choisir de façon interactive, à l'aide de la boîte de dialogue qui s'ouvre automatiquement

au début du script, la largeur nw des images redimensionnées ainsi que le niveau de qualité q des fichiers jpg correspondants enregistrés.

```
;-----------------------------------------------------------------
; Fichier: ct-lotredim.scm
; Action:  Redimensionne un lot d'images contenues dans un dossier r1
;          et copie des images redimensionnées dans un dossier r2
; Install: sudo cp ct-lotredim.scm /usr/share/gimp/2.0/scripts/
;
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotredim r1  r2 nw q);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(nh 0) (k 0) ;(nw 100)
(h (car (gimp-image-height img)))
(w (car (gimp-image-width img)))
);fin defvar let2
;
;-----------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------
(set! k (/ w h))
(set! nh (/ nw k))
(gimp-image-scale img nw nh)
;-----------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;-------------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-------------------------------------------------------------
(
script-fu-register "ct-lotredim"
"<Image>/MesScripts/ct-lotredim"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-VALUE    "Nouvelle largeur" "100"  ;nw
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
)
```

Le schéma suivant illustre ces différents éléments.

3.10) Encadrer un lot d'images

On utilise l'ossature de traitement par lot, vue en **3.8**, et le script qui ajoute un cadre à une image, vu en **2.9,** pour créer le script suivant qui ajoute un cadre à chacune des images d'un lot d'images.

Le script effectue les tâches suivantes :

- Il établit la liste des chemins complets de fichiers qui se trouvent dans le répertoire r1
- Il parcourt cette liste et pour chaque fichier de cette liste :
  > il charge l'image correspondante en mémoire ;
  > il l'encadre avec un cadre blanc et noir simple ;
  > il l'enregistre dans le répertoire r2.

Les variables utilisées sont les suivantes :

r1	: répertoire où se trouvent les fichiers images sources
r2	: répertoire où sont enregsitrées les images encadrées
liste	: liste des chemins complets des images à encadrer
in	: chemin complet de l'image courante à encadrer
img	: Identifiant de l'image courante à encadrer, chargée en mémoire
out	: chemin complet de l'image courante encadrée à enregistrer
c	: identifiant du calque actif courant
w	: largeur de l'image source (px)
h	: hauteur de l'image source (px)

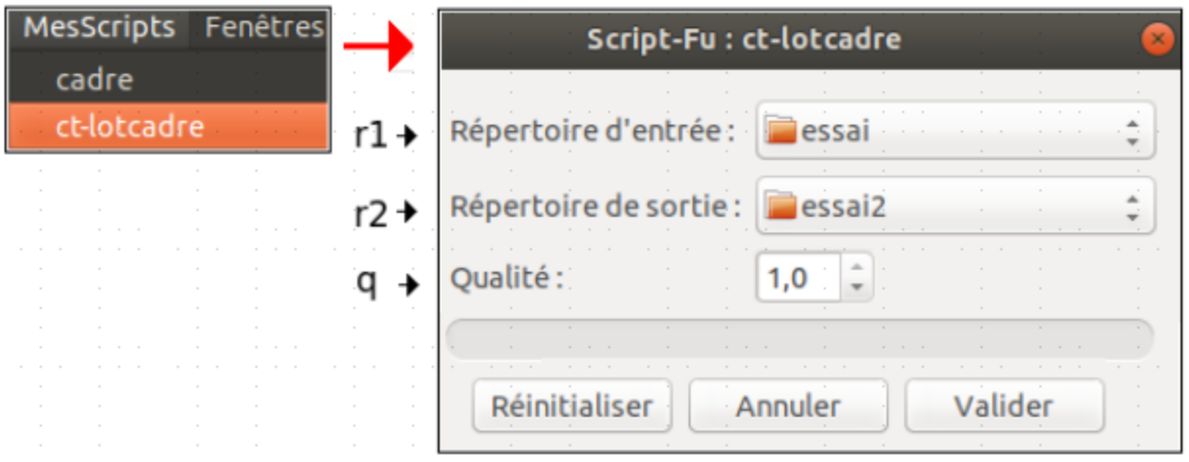
nh	: nouvelle hauteur de l'image encadrée nh = h+4
nw	: nouvelle largeur de l'image encadrée nw = w+4
couex	: couleur de la partie extérieure du cadre '(0 0 0)
couin	: couleur de la partie intérieure du cadre '(255 255 255)
n	: nombre de coordonnées des points constituant le tracé 4 pts x 2 + 2 pour le retour sur le premier point = 10
tc	: tableau des 10 coordonnées des 4 points
u	: variable utilisée pour les calculs de largeur
v	: variable utilisée pour les calculs de hauteur

Le fonctionnement des différentes instructions et fonctions a été détaillé en 2.9) et en 3.8).

```
;-----------------------------------------------------------------------
; Fichier: ct-lotcadre.scm
; Action:  Encadre chacune des images contenues dans un dossier r1
;          et copie les images encadrées dans un dossier r2
; Install: sudo cp ct-lotcadre.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------------
(define (ct-lotcadre r1  r2 q);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(couex '(0 0 0)) ; couleur de la partie extérieure du cadre
(couin '(255 255 255)) ; couleur de la partie intérieure du cadre
(n 10) ; nombre de coordonnées des points constituant le tracé
(tc (make-vector 10 'double))
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(u w) (v h) (nw (+ w 4)) (nh (+ h 4))
);fin defvar let2
;-----------------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------------
(gimp-image-resize img nw nh 2 2)
(set! c (car (gimp-layer-new img nw nh 1 "myC" 100 0)))
(gimp-image-insert-layer img c 0 0)
(gimp-brush-new "mabrosse")
(gimp-brushes-refresh)
(gimp-context-set-brush "mabrosse")
(gimp-context-set-brush-aspect-ratio 0)
;
;--cadre extérieur noir de 1 pixel d'épaisseur
(gimp-context-set-foreground couex) ;noir
(gimp-context-set-brush-size 1) ;1 pixel
(set! u (- nw 1))
(set! v (- nh 1))
```

```
(vector-set! tc 0 0) ; point 1 (x)
(vector-set! tc 1 0) ; point 1 (y)
(vector-set! tc 2 u) ; point 2 (x)
(vector-set! tc 3 0) ; point 2 (y)
(vector-set! tc 4 u) ; point 3 (x)
(vector-set! tc 5 v) ; point 3 (y)
(vector-set! tc 6 0) ; point 4 (x)
(vector-set! tc 7 v) ; point 4 (y)
(vector-set! tc 8 0) ; point 5 (x)
(vector-set! tc 9 0) ; point 5 (y)
(gimp-pencil c n tc) ;tracé dans l'ordre des points
;
;--cadre intérieur blanc de 1 pixel d'épaisseur
(gimp-context-set-foreground couin) ;blanc
(gimp-context-set-brush-size 1) ;1 pixel
(set! u (- nw 2))
(set! v (- nh 2))
(vector-set! tc 0 1)
(vector-set! tc 1 1)
(vector-set! tc 2 u)
(vector-set! tc 3 1)
(vector-set! tc 4 u)
(vector-set! tc 5 v)
(vector-set! tc 6 1)
(vector-set! tc 7 v)
(vector-set! tc 8 1)
(vector-set! tc 9 1)
(gimp-pencil c n tc )
;
(gimp-image-flatten img)
(set! c (car (gimp-image-get-active-drawable img)))
;-----------------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;
;-------------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-------------------------------------------------------------
(
script-fu-register "ct-lotcadre"
"<Image>/MesScripts/ct-lotcadre"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
)
```

Le schéma suivant illustre ces différents éléments.



Ce script place autour de chaque image un cadre blanc-noir (variables couin et couex) de 2 pixels de large (1 pixel blanc + 1 pixel noir)

Cependant, il serait intéressant de pouvoir choisir ces deux couleurs de façon interactive, en début d'exécution du script.

On va donc modifier le script en lui passant en paramètres les variables suivantes :

couin COLOR: couleur de la partie intérieure du cadre '(255 255 255)
couex COLOR: couleur de la partie extérieure du cadre '(0 0 0)

Pour cela on associe le type de paramètre SF-COLOR aux variables couin et couex.

Ces paramètres créent, en mode interactif, des zones de dialogue dans la boîte de dialogue du script-fu permettant de saisir les couleurs correspondantes.

Syntaxe

```
SF-COLOR "info" '(r g b)
```

```
"info": Texte affiché devant la zone de dialogue.
'(r g b) : couleur par défaut (r, g et b compris entre 0 et 1 inclus)
```

▶ **Exemple**

```
SF-COLOR   "Couleur intérieure"   '(255 255 255) ;couex
SF-COLOR   "Couleur extérieure"   '(0 0 0)       ;couin
```

On obtient alors le script final suivant.

```
;-----------------------------------------------------------------------
; Fichier: ct-lotcadre.scm
; Action:  Encadre chacune des images contenues dans un dossier r1
;          et copie les images encadrées dans un dossier r2
; Install: sudo cp ct-lotcadre.scm /usr/share/gimp/2.0/scripts/
;
;-----------------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------------
(define (ct-lotcadre r1  r2 q couex couin);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
;(couex '(0 0 0)) ; couleur de la partie extérieure du cadre
;(couin '(255 255 255)) ; couleur de la partie intérieure du cadre
(n 10) ; nombre de coordonnées des points constituant le tracé
(tc (make-vector 10 'double))
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(u w) (v h) (nw (+ w 4)) (nh (+ h 4))
);fin defvar let2
;
;-----------------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------------
(gimp-image-resize img nw nh 2 2)
(set! c (car (gimp-layer-new img nw nh 1 "myC" 100 0)))
(gimp-image-insert-layer img c 0 0)
(gimp-brush-new "mabrosse")
(gimp-brushes-refresh)
(gimp-context-set-brush "mabrosse")
(gimp-context-set-brush-aspect-ratio 0)
;
;--cadre extérieur noir de 1 pixel d'épaisseur
(gimp-context-set-foreground couex) ;couleur extérieure choisie
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 1))
(set! v (- nh 1))
(vector-set! tc 0 0) ; point 1 (x)
(vector-set! tc 1 0) ; point 1 (y)
(vector-set! tc 2 u) ; point 2 (x)
(vector-set! tc 3 0) ; point 2 (y)
(vector-set! tc 4 u) ; point 3 (x)
(vector-set! tc 5 v) ; point 3 (y)
(vector-set! tc 6 0) ; point 4 (x)
(vector-set! tc 7 v) ; point 4 (y)
(vector-set! tc 8 0) ; point 5 (x)
(vector-set! tc 9 0) ; point 5 (y)
(gimp-pencil c n tc) ;tracé dans l'ordre des points
;
```

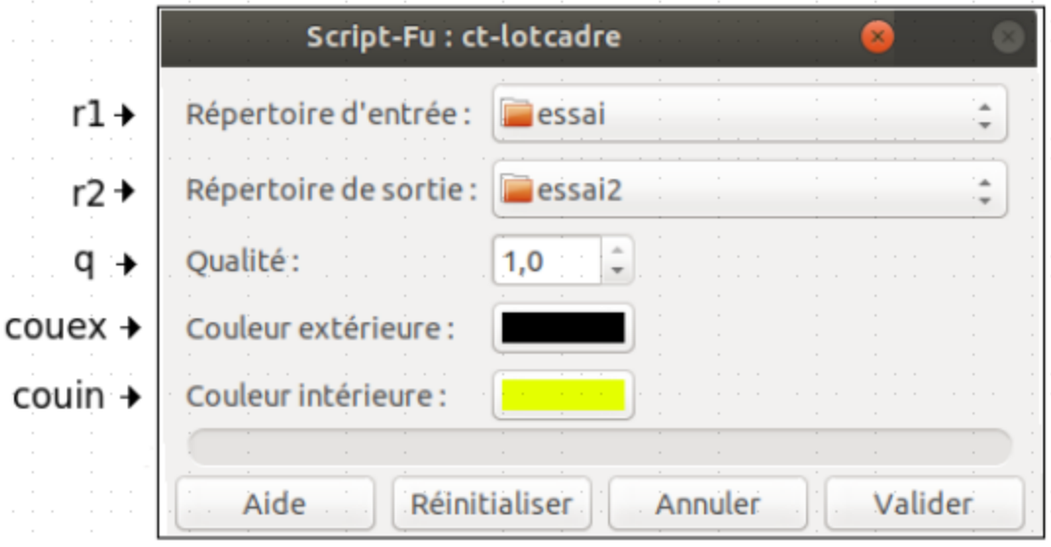
```
;--cadre intérieur blanc de 1 pixel d'épaisseur
(gimp-context-set-foreground couin) ;couleur intérieure choisie
(gimp-context-set-brush-size 1) ;1 pixel
(set! u (- nw 2))
(set! v (- nh 2))
(vector-set! tc 0 1)
(vector-set! tc 1 1)
(vector-set! tc 2 u)
(vector-set! tc 3 1)
(vector-set! tc 4 u)
(vector-set! tc 5 v)
(vector-set! tc 6 1)
(vector-set! tc 7 v)
(vector-set! tc 8 1)
(vector-set! tc 9 1)
(gimp-pencil c n tc )
;
(gimp-image-flatten img)
(set! c (car (gimp-image-get-active-drawable img)))
;-----------------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;
;-------------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-------------------------------------------------------------
(
script-fu-register "ct-lotcadre"
"<Image>/MesScripts/ct-lotcadre"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
SF-COLOR   "Couleur extérieure"      '(255 255 255) ;couex
SF-COLOR   "Couleur intérieure"     '(0 0 0)  ;couin
)
```

Désormais on peut choisir la couleur extérieure et la couleur intérieure de l'encadrement mais chaque couleur n'occupe qu'un pixel de large.

Si on choisit la même couleur intérieure et extérieure, cela revient à encadrer l'image d'une ligne de 2 pixels d'épaisseur.

Pour pouvoir obtenir d'autres épaisseurs de cadre, il faudrait modifier le script en conséquence.

Le schéma suivant montre la nouvelle boîte de dialogue obtenue.



3.11) Ajouter un logo à un lot d'images

On utilise l'ossature de traitement par lot, vue en **3.8**, et le script qui ajoute un masque à une image, vu en **2.10,** pour créer le script suivant qui ajoute un logo à chacune des images d'un lot d'images.

Le script effectue les tâches suivantes :

- Il établit la liste des chemins complets de fichiers qui se trouvent dans le répertoire r1
- Il parcourt cette liste et pour chaque fichier de cette liste :
  > il charge l'image correspondante en mémoire ;
  > il ajoute un logo à cette image, dans le coin bas-droit ;
  > il l'enregistre l'image dotée du logo dans le répertoire r2.

Les variables utilisées sont les suivantes :

`r1`	`: répertoire où se trouvent les fichiers images sources`
`r2`	`: répertoire où sont enregistrées les images encadrées`
`q`	`: niveau de qualité enregistrement jpg (0<=q<=1)`
`logo`	`: chemin complet du fichier logo`
`liste`	`: liste des chemins complets des images à encadrer`
`imgl`	`: identifiant d'image associé au chemin complet logo`
`hl`	`: hauteur du logo en pixels`
`wl`	`: largeur du logo en pixels`
`in`	`: chemin complet de l'image courante à encadrer`

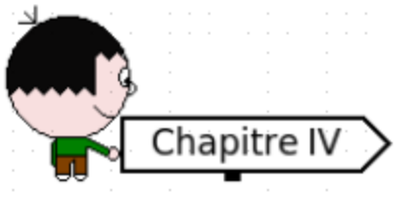
img	: Identifiant de l'image courante à encadrer, chargée en mémoire
out	: chemin complet de l'image courante encadrée à enregistrer
c	: identifiant du calque actif courant (image courante)
w	: largeur de l'image source (px)
h	: hauteur de l'image source (px)
nh	: nouvelle hauteur de l'image encadrée nh = h+4
nw	: nouvelle largeur de l'image encadrée nw = w+4
px	: pourcentage de largeur d'image occupée par le logo
r	: ratio du logo r=hl/Wl
nwl	: nouvelle largeur du logo
nhl	: ouvelle hauteur du logo
dx	: décalage en x du logo par rapport à l'image courante
dy	: décalage en y du logo par rapport à l'image courante
cl	: identifiant du calque copntenant le logo

Le fonctionnement des différentes instructions et fonctions a été détaillé en 2.9) et en 3.8).

Le script obtenu est le suivant

```
;-----------------------------------------------------------------
; Fichier: ct-lotlogo.scm
; Action:  Place un logo sur chaque image d'un lot d'images
;          contenues dans un dossier r1
;          et copie des images redimensionnées dans un dossier r2
; Install: sudo cp ct-lotlogo.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotlogo r1  r2 q logo);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1)));let1
(imgl (car (file-png-load 1 logo logo)))
(hl (car (gimp-image-height imgl)))
(wl (car (gimp-image-width imgl)))
); fin defvar let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(h (car (gimp-image-height img)))
(w (car (gimp-image-width img)))
(px 0.1) ;(wl 128) (hl 128)
(r (/ hl wl))
(nwl (* px w)) (nhl (* nwl r))
(dx (- w nwl)) (dy (- h nhl))
(cl (car (gimp-file-load-layer 1 img logo)))
);fin defvar let2
```

```
;-----------------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------------
(gimp-image-add-layer img cl 0)
(gimp-layer-scale cl nwl nhl 0)
(gimp-layer-resize-to-image-size cl)
(gimp-layer-translate cl dx dy)
(gimp-layer-resize-to-image-size cl)
(gimp-image-flatten img)
;
(set! c (car (gimp-image-get-active-drawable img)))
;-----------------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;-------------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-----------------------------------------------------------------------
(
script-fu-register "ct-lotlogo"
"<Image>/MesScripts/ct-lotlogo"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/Images/logo.png";logo
)
```



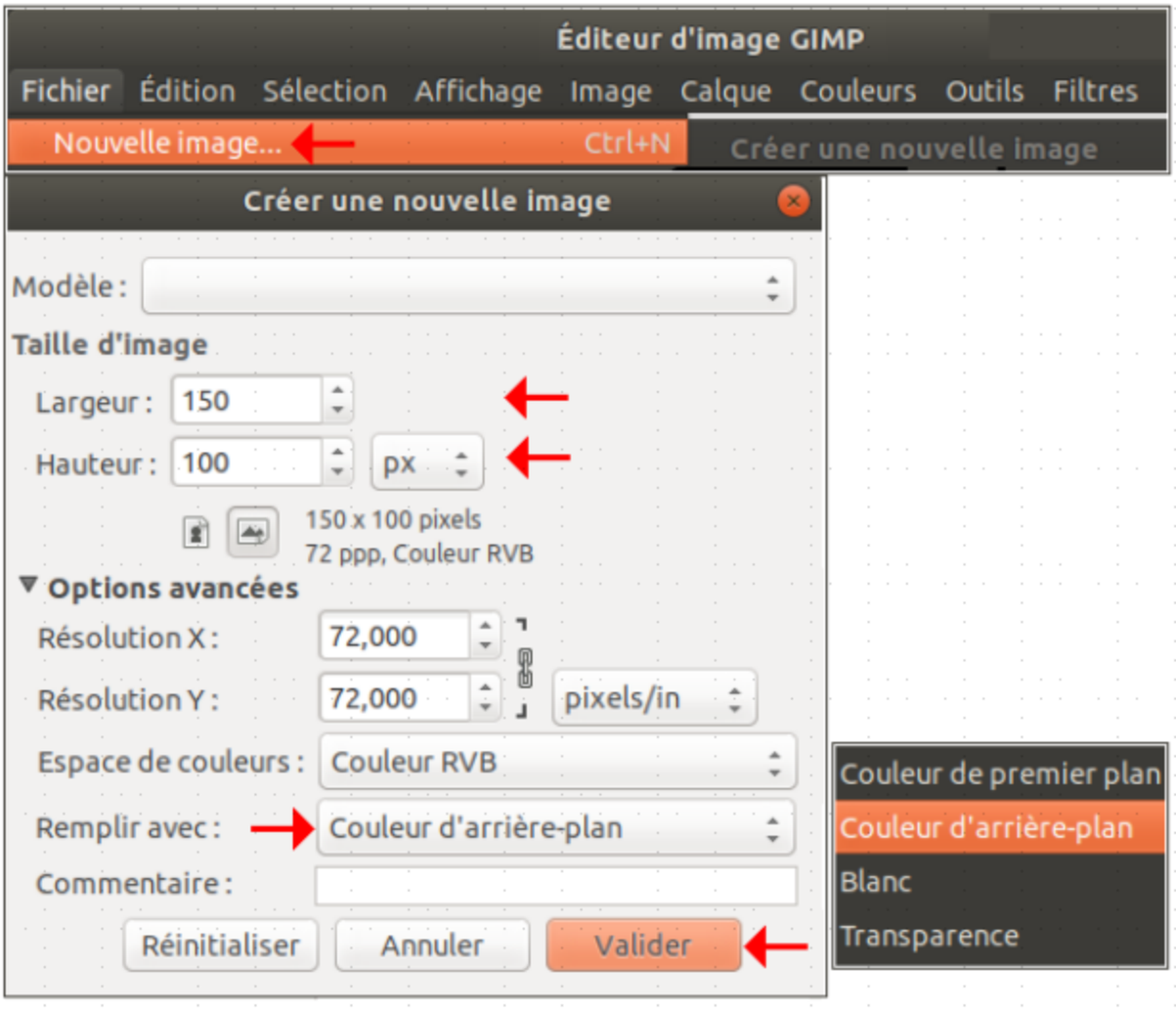
Chapitre IV
Tester les fonctions Gimp

On examine ci-après le fonctionnement de quelques fonctions importantes de gImp en terme de correspondance de leur action par rapport à ce que l'on fait lorsqu'on manipule les images en mode interactif.

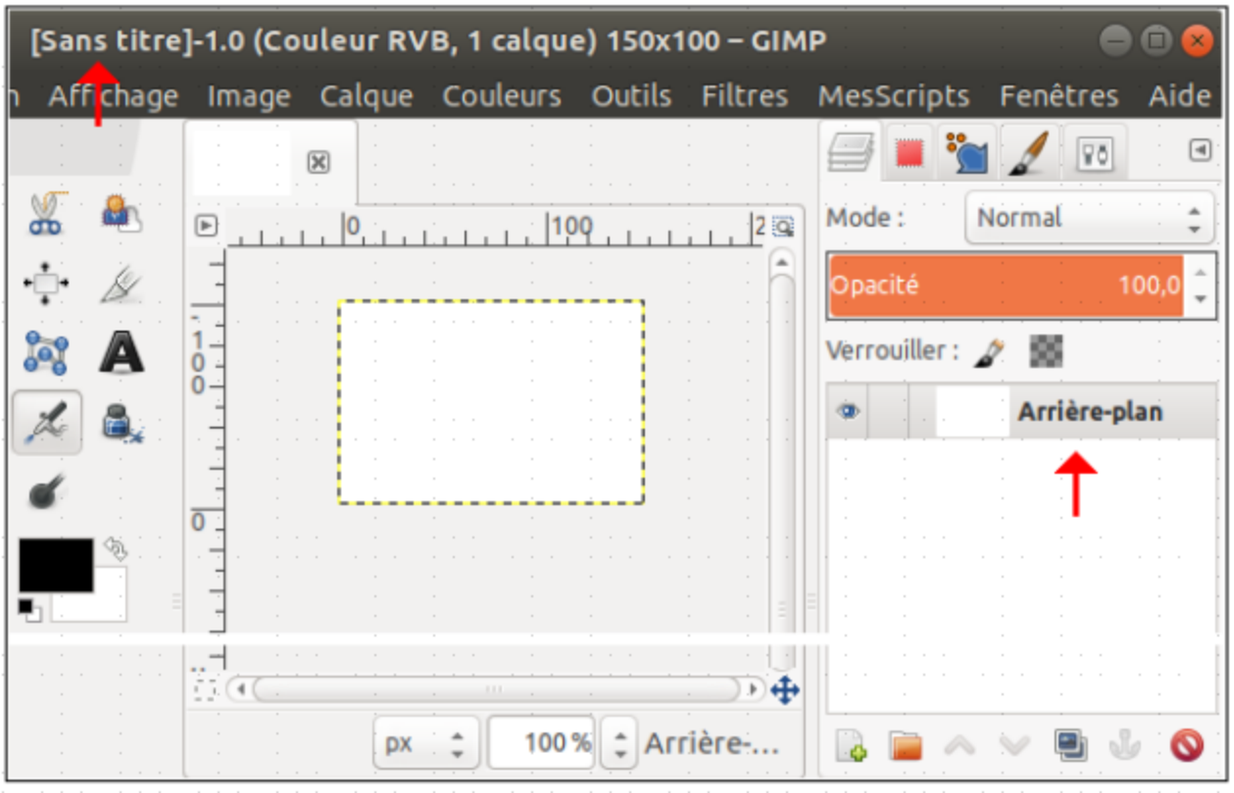
4.1) Créer une nouvelle image

4.1.1 Mode interactif

Pour créer une nouvelle image en mode interactif, on utilise le menu **Fichier > Nouvelle image** puis, dans la boîte de dialogue qui s'ouvre, on précise la taille de l'image et sa couleur de remplissage (couleur de premier plan, d'arrière plan, blanc ou transparence. Puis on clique Valider.



La nouvelle image ainsi créée s'affiche alors aussitôt dans la fenêtre de Gimp qui attribue automatiquement le nom **Sans titre** à l'image et le nom **Arrière-plan** au calque.



4.1.2 Mode non interactif

Pour effectuer la même tâche, c'est à dire créer et afficher une nouvelle image, en mode non interactif on saisit les instructions suivantes dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(define img (car (gimp-image-new 150 100 RGB )))
(define c1 (car (gimp-layer-new img 150 100 1 "Arrière-plan" 100 0)))
(gimp-image-insert-layer img c1 0 0)
(gimp-drawable-fill c1 BG-IMAGE-FILL)
(gimp-display-new img)

Réponse de la console
imgc1(#t)(#t)(1)
```

La nouvelle image ainsi créée s'affiche alors aussitôt dans la fenêtre de Gimp, de façon identique à l'image précédente.
En mode interactif, pour obtenir une image transparente, non remplie de blanc, il suffit de choisir l'option de menu transparence. En mode non interactif, il suffit de supprimer l'instruction suivante.

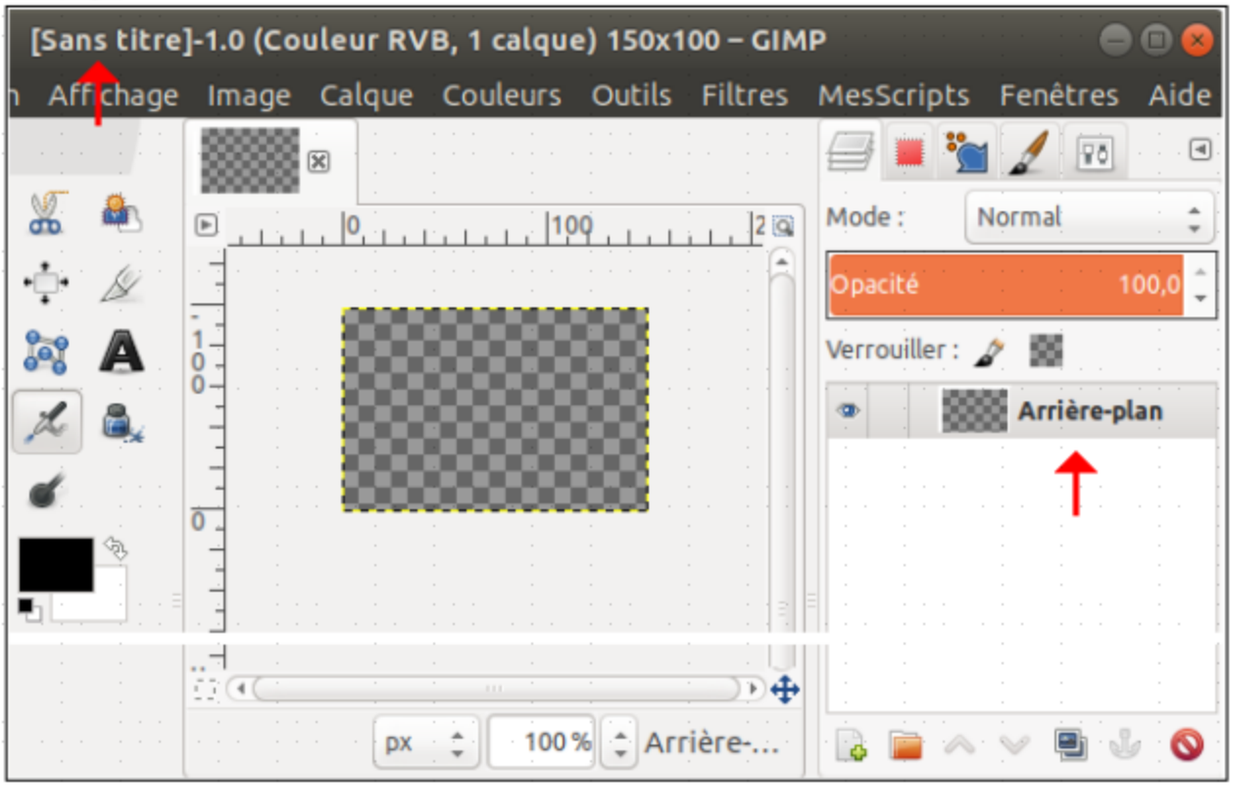
```
(gimp-drawable-fill c1 BG-IMAGE-FILL)
```

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(define img (car (gimp-image-new 150 100 RGB )))
(define c1 (car (gimp-layer-new img 150 100 1 "Arrière-plan" 100 0)))
(gimp-image-insert-layer img c1 0 0)
(gimp-display-new img)
```

Réponse de la console

```
imgc1(#t)(1)
```

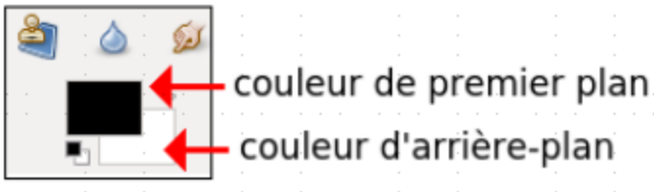
Affichage



4.2) Modifier la couleur d'arrière plan

4.2.1 Mode interactif

En mode interactif, pour modifier la couleur d'arrière plan utilisée par Gimp (blanc par défaut), on clique le carré correspondant, situé sous les outils. Dans la boîte de dialogue qui s'ouvre on choisit une couleur puis on clique Valider.

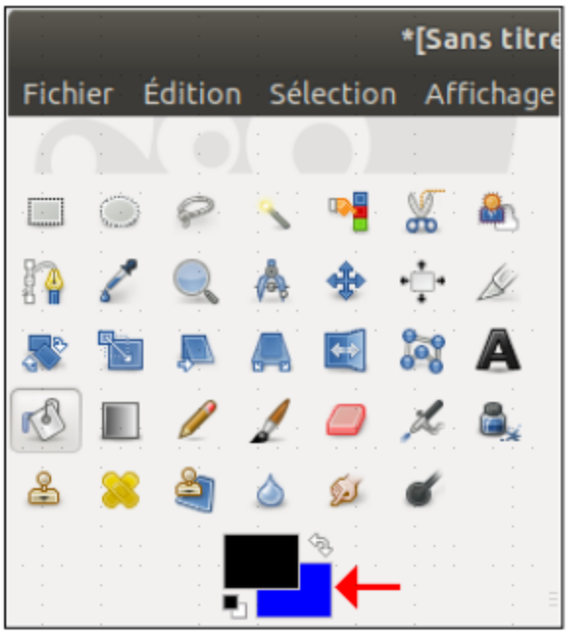


4.2.2 Mode non interactif

En mode non interactif, pour modifier la couleur d'arrière plan utilisée par Gimp on saisit l'instruction suivante dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (gimp-context-set-background  '(0 0 255))

Réponse de la console
(#t)
```

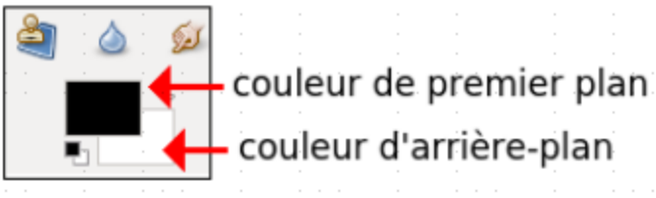
La nouvelle couleur d'arrière-plan active (bleu avec l'instruction précédente) s'affiche alors aussitôt dans le carré correspondant, sous la boîte à outils .



4.3) Modifier la couleur de premier plan

4.3.1 Mode interactif

En mode interactif, pour modifier la couleur de premier plan utilisée par Gimp (noir par défaut), on clique le carré correspondant, situé sous les outils. Dans la boîte de dialogue qui s'ouvre on choisit une couleur puis on clique valider.

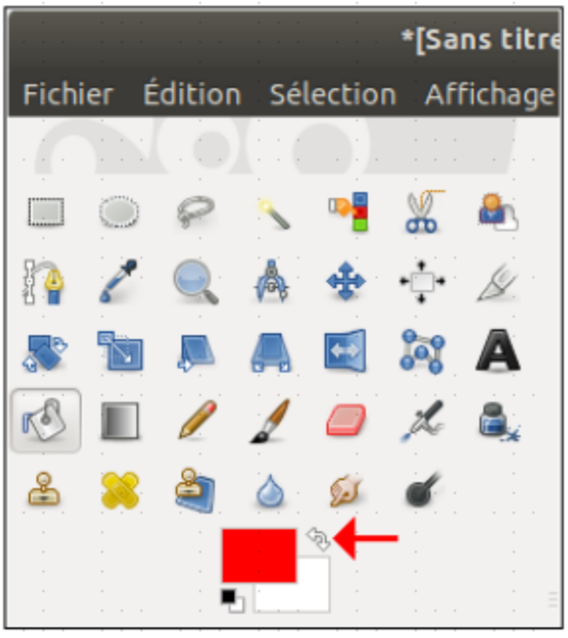


4.3.2 Mode non interactif

En mode non interactif, pour modifier la couleur de premier plan utilisée par Gimp on saisit l'instruction suivante dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (gimp-context-set-foreground  '(255 0 0))

Réponse de la console
(#t)
```

La nouvelle couleur de premier plan (rouge avec l'instruction précédente) active s'affiche alors aussitôt dans le carré correspondant, sous la boîte à outils .



4.4) Ouvrir une image

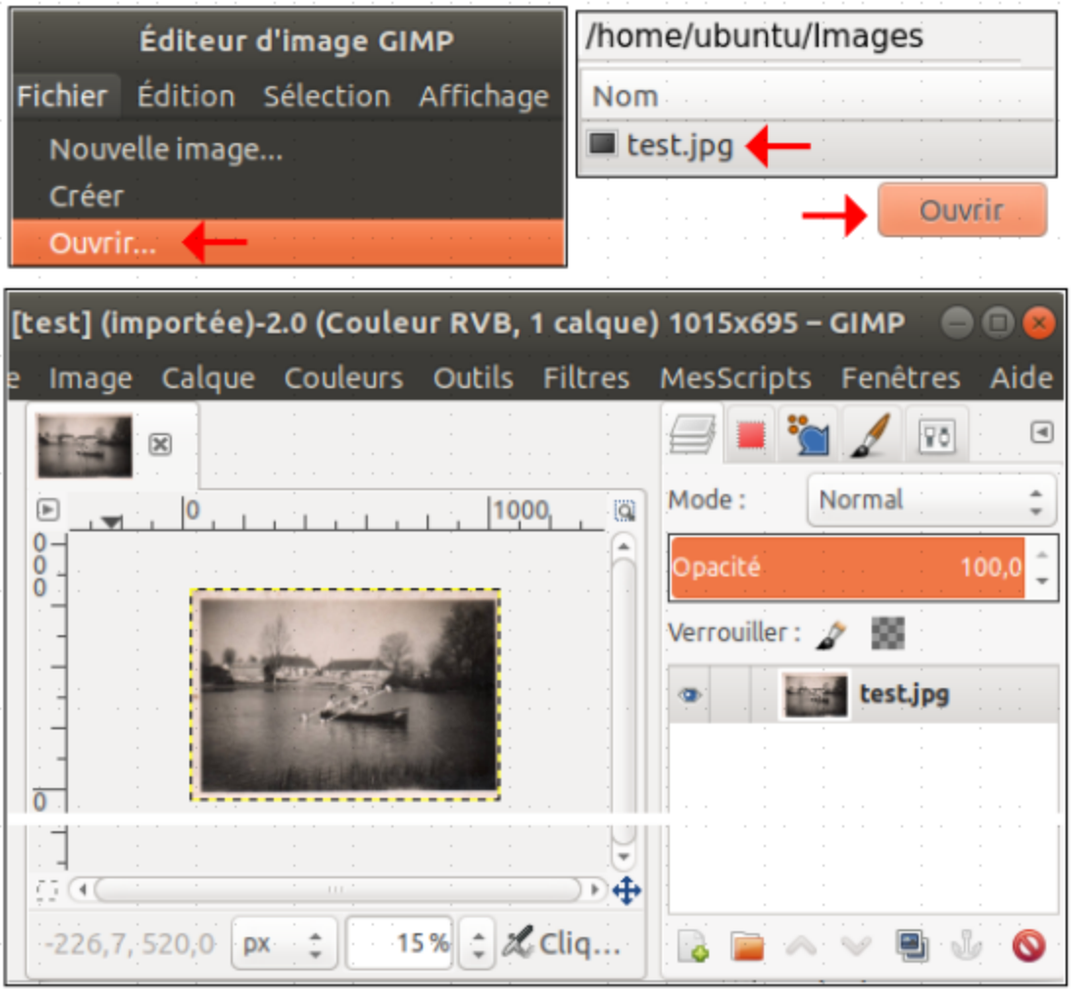
4.4.1 Mode interactif

Pour ouvrir une image en mode interactif, on clique le menu correspondant de Gimp **Fichier > Ouvrir....**

Dans la boîte de dialogue qui s'ouvre, on choisit un fichier image (le fichier "test.jpg" dans le répertoire "/home/ubuntu/Images" par exemple) et on clique le bouton **Ouvrir**.

L'image correspondante s'affiche aussitôt dans la fenêtre de Gimp.

Gimp attribue automatiquement le nom du fichier (ici "test.jpg") au calque qui contient l'image affichée.



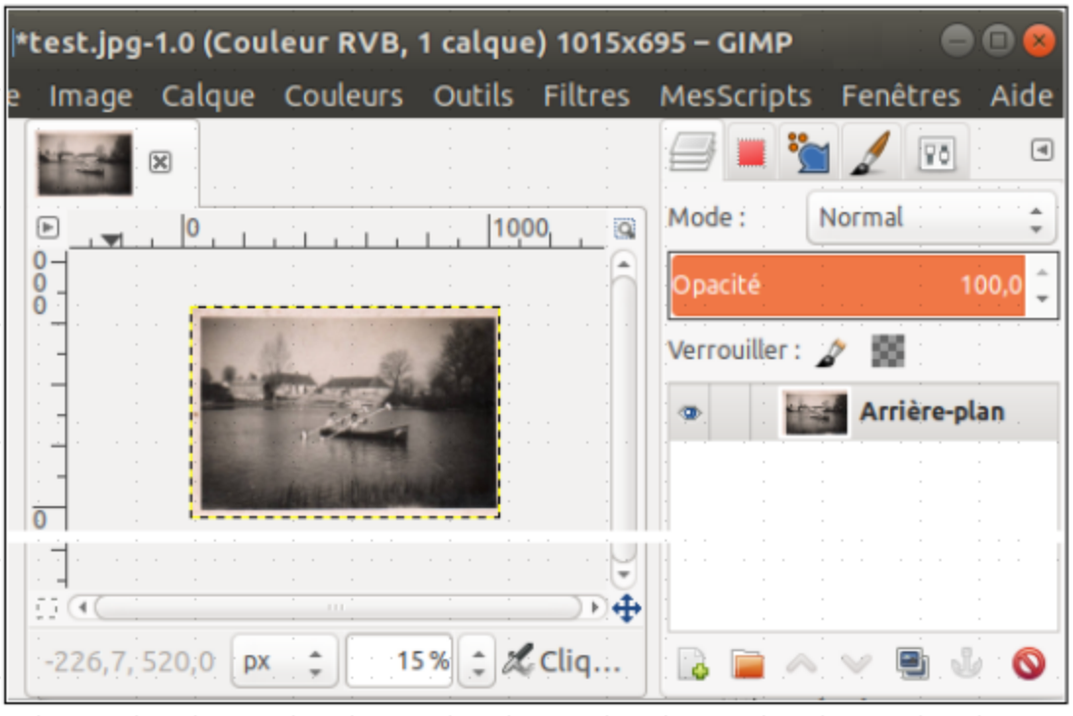
4.4.2 Mode non interactif

Pour ouvrir et afficher une image, en mode non interactif, on saisit les instructions suivantes dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

>
(define in "/home/ubuntu/Images/test.jpg")
(define img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(gimp-display-new img)

Réponse de la console
inimg(1)
```

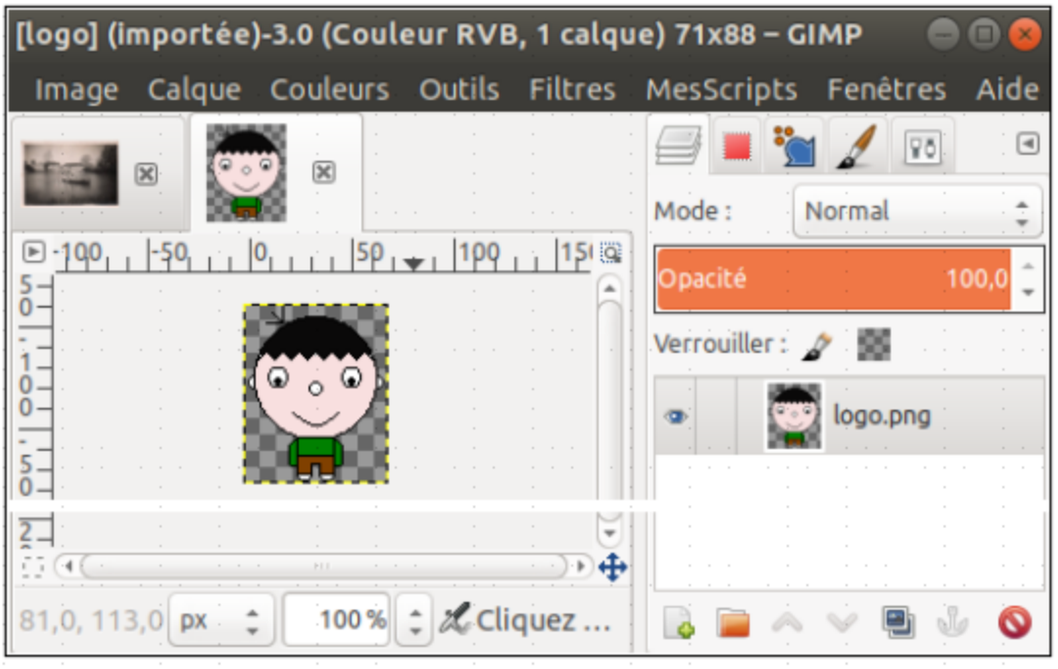
L'image correspondante s'affiche alors aussitôt dans la fenêtre de Gimp, comme en mode interactif.

Le résultat obtenu est le même, à la seule petite différence que Gimp attribue automatiquement le nom **Arrière-plan** au calque qui contient l'image affichée.



4.5) Ouvrir une seconde image

4.5.1 Mode interactif



Pour ouvrir une seconde image, ayant déjà ouvert une première image, on clique à nouveau le menu **Fichier > Ouvrir...** et on procède comme indiqué en 4.4.

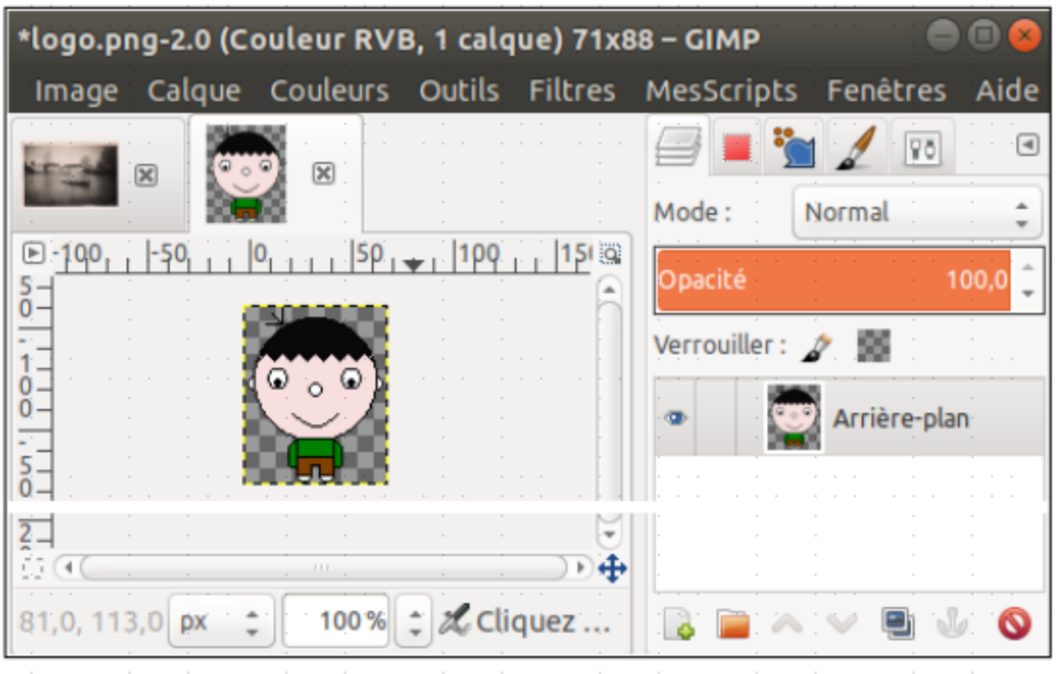
On ouvre par exemple l'image "logo.png", dotée d'un canal de transparence et située dans le répertoire "/home/ubuntu/Images".
La seconde image s'affiche dans la fenêtre de Gimp qui attribue automatiquement le nom du fichier (ici "logo.png") au calque qui contient l'image affichée.

4.5.2 Mode non interactif

En mode non interactif, pour ouvrir et afficher une seconde image, suite à une première image déjà ouverte, on saisit les instructions suivantes dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

>
(define in1 "/home/ubuntu/Images/test.jpg")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(gimp-display-new img1)
(define in2 "/home/ubuntu/Images/logo.png")
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
(gimp-display-new img2)

Réponse de la console
in1img1(1)in2img2(2)
```

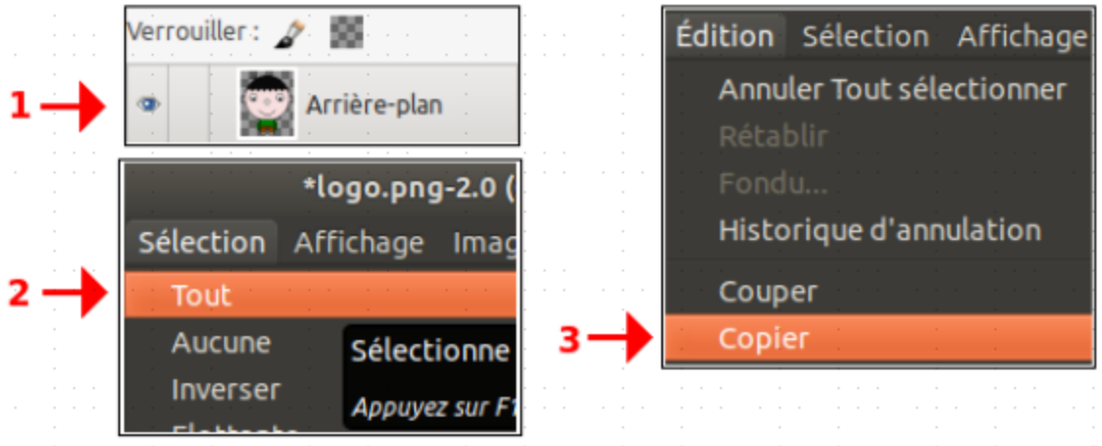
Le résultat obtenu est le même, à la seule petite différence que Gimp attribue automatiquement le nom **Arrière-plan** au calque qui contient l'image affichée.



4.6) Copier dans le Presse-papiers

4.6.1 Mode interactif

En mode interactif, pour copier dans le presse-papiers le contenu d'un calque :
1) On sélectionne le calque de cette image ;
2) On clique le menu **Sélection > Tout ;**
3) On clique le menu **Edition > Copier**.



Il suffit alors d'ouvrir un logiciel graphique quelconque (**kolourpaint** par exemple) puis de cliquer **Edition > Coller** pour vérifier que le contenu du calque a bien été copié dans **Gimp** et peut ainsi être par la suite directement collé dans la fenêtre de Kolourpaint.

4.6.2 Mode non interactif

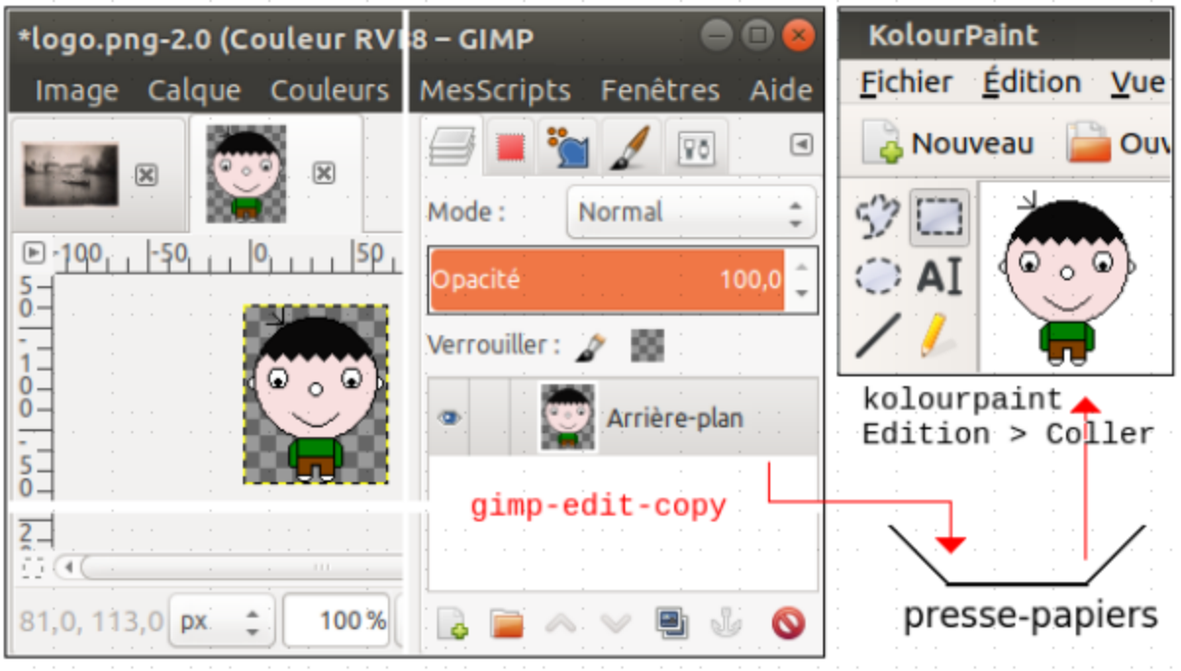
En mode non interactif, pour copier le contenu d'un calque dans le presse-papiers, on saisit les instructions suivantes dans la console Script Fu de Gimp.

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(define in1 "/home/ubuntu/Images/test.jpg")
(define in2 "/home/ubuntu/Images/logo.png")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
(gimp-display-new img1)
(gimp-display-new img2)
(define tcimg2 (gimp-image-get-layers img2))
(define lcimg2 (vector->list (cadr tcimg2)))
(gimp-edit-copy (car lcimg2))

Réponse de la console
in1in2img1img2(1)(2)tcimg2lcimg2(1)
```

Dans ces instructions, on établit la liste des identifiants des calques associés à l'image d'identifiant img2 puis on utilise le premier des identifiants de cette liste (c'est à dire **car** lclimg2) pour copier le contenu du calque correspondant dans le presse-papiers.

```
> in1
"/home/ubuntu/Images/test.jpg"
> in2
"/home/ubuntu/Images/logo.png"
> img1
1
> img2
2
> tcimg2
(1 #(4))
> lcimg2
(4)
> (car lcimg2)
4
```

`in1`	chemin complet de l'image test.jpg
`in2`	chemin complet de l'image logo.png
`img1`	identifiant associé à l'image "test.jpg"
`img2`	identifiant associé à l'image "logo.png"
`tcimg2`	tableau contenant le nombre et les identifiants des calques associés à l'image d'identifiant img2
`lcimg2`	Liste contenant les identifiants des calques associés à l'image d'identifiant img2
`(car lcimg2)`	identifiant du premier des calques associés à l'image d'identifiant img2

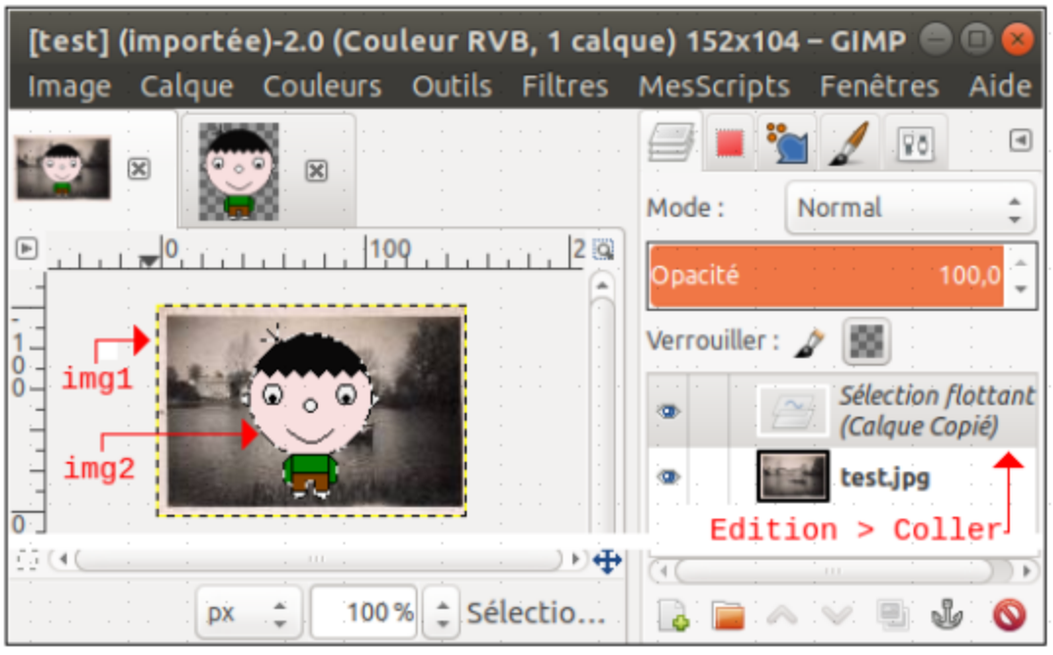


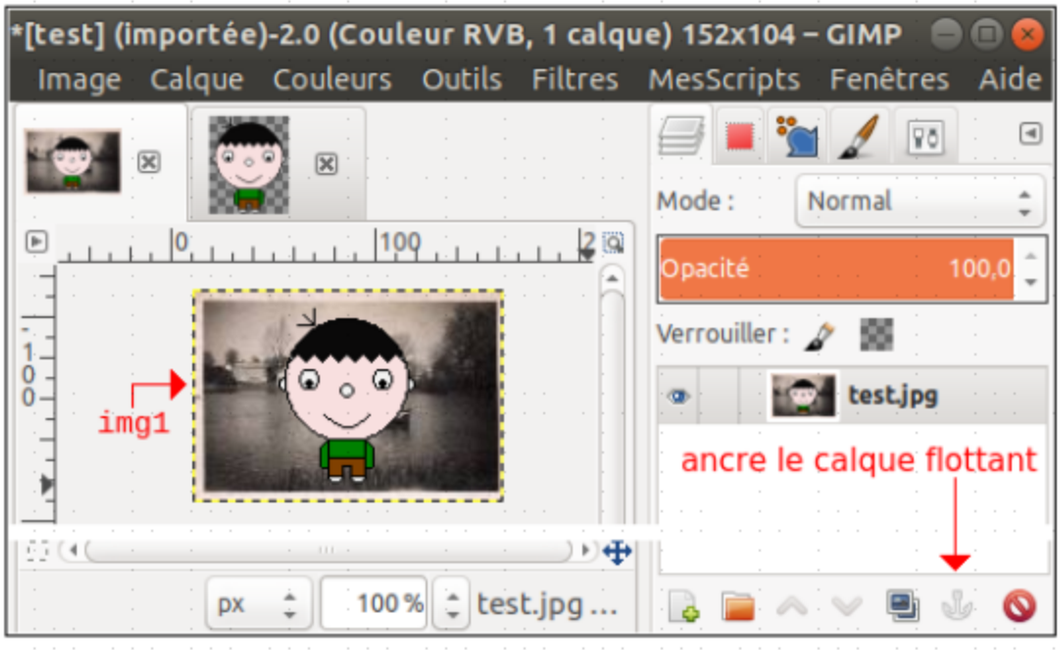
4.7) Coller depuis le Presse-papiers

4.7.1 Mode interactif

En mode interactif, pour coller le contenu du presse-papiers au dessus d'une image (au dessus de l'image "test.jpg" par exemple) :
1) On sélectionne le calque de cette image ("test.jpg"))
2) On clique le menu **Edition > Coller**

Cela colle le contenu du presse-papiers dans un calque flottant, juste au dessus du calque de l'image. On colle le contenu de ce calque sur l'image en cliquant l'**ancre** située en bas de la fenêtre des calques.



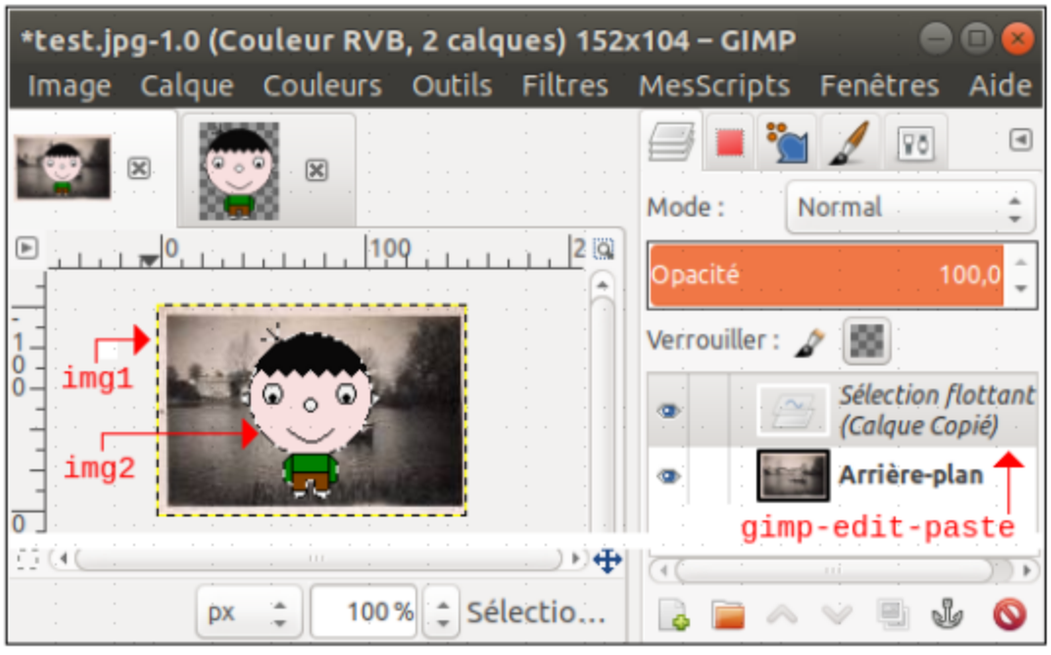


4.7.2 Mode non interactif

En mode non interactif, pour coller le contenu du presse-papiers au dessus d'une image (au dessus de l'image "test.jpg" par exemple) on utilise l'instruction **gimp-edit-paste**.

Coller le contenu du presse-papiers dans un calque dessinable **Syntaxe**: (gimp-edit-paste d clear)
Paramètres d DRAWABLE: calque où on veut copier le contenu du presse-papiers option INT32: (0) ou (1) en option si utilisation de masques. Valeur retournée calqueflottant LAYER:calque flottant, placé au dessus de l'image, dans lequel se trouve collé le contenu du presse-papiers

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

>
(define in1 "/home/ubuntu/Images/test.jpg")
(define in2 "/home/ubuntu/Images/logo.png")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
(gimp-display-new img1)
(gimp-display-new img2)
(define tcimg2 (gimp-image-get-layers img2))
(define lcimg2 (vector->list (cadr tcimg2)))
(gimp-edit-copy (car lcimg2))
(define tcimg1 (gimp-image-get-layers img1))
(define lcimg1 (vector->list (cadr tcimg1)))
(define c1img1 (car lcimg1))
(define sf (gimp-edit-paste c1img1 0))
```



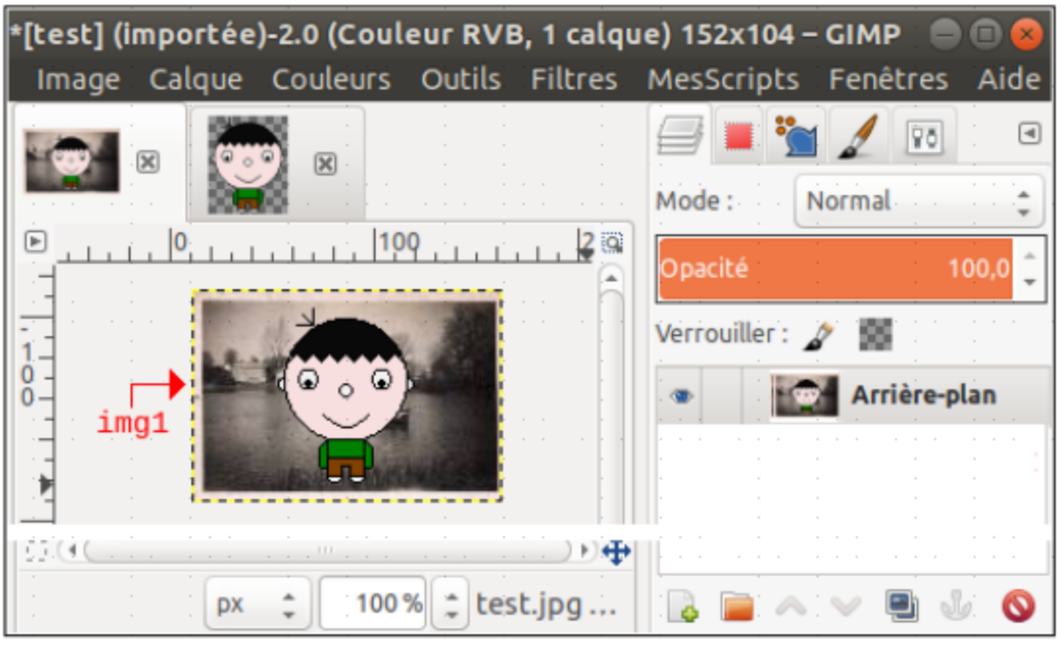
Réponse de la console
```
in1in2img1img2(1)(2)tcimg2lcimg2(1)tcimg1lcimg1c1img1sf
```

Pour ancrer le calque flottant sur l'image, on ajoute l'instruction **gimp-floating-sel-anchor**

img2	identifiant associé à l'image "logo.png"
tcimg1	tableau contenant le nombre et les id. des calques associés à img1
lcimg1	Liste contenant les identifiants des calques associés à 1 img1
c1img1	identifiant du premier des calques associés à l'image d'identifiant img1
sf	Sélection flottante

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif
>
(define in1 "/home/ubuntu/Images/test.jpg")
(define in2 "/home/ubuntu/Images/logo.png")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
(gimp-display-new img1)
(gimp-display-new img2)
(define tcimg2 (gimp-image-get-layers img2))
(define lcimg2 (vector->list (cadr tcimg2)))
(gimp-edit-copy (car lcimg2))
(define tcimg1 (gimp-image-get-layers img1))
(define lcimg1 (vector->list (cadr tcimg1)))
(define c1img1 (car lcimg1))
(define sf (gimp-edit-paste c1img1 0))
(gimp-floating-sel-anchor (car sf))
```

Réponse de la console
```
in1in2img1img2(1)(2)tcimg2lcimg2(1)tcimg1lcimg1c1img1sf(#t)
```

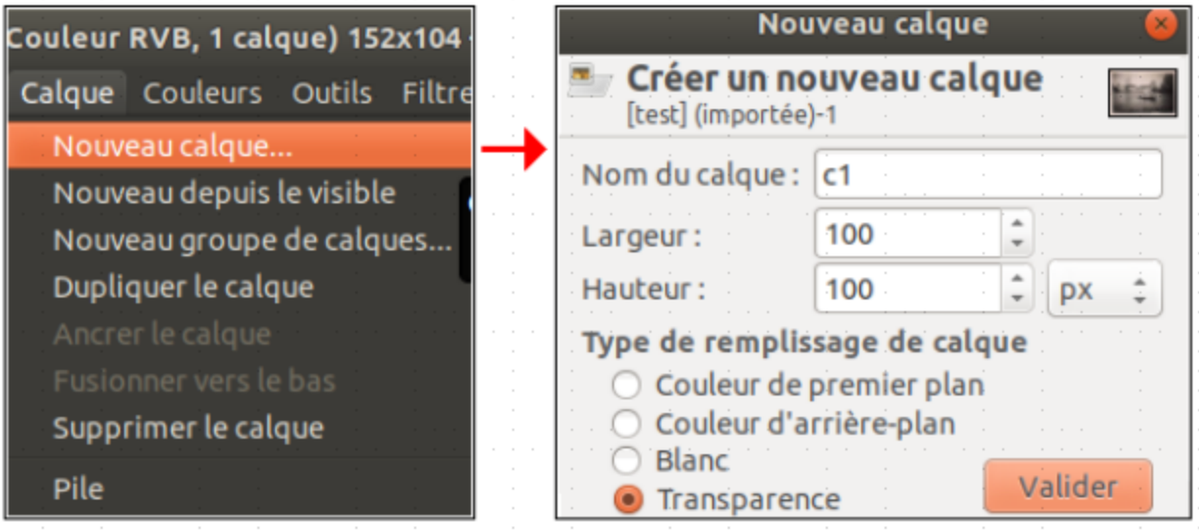


4.8) Ajouter des calques sur une image

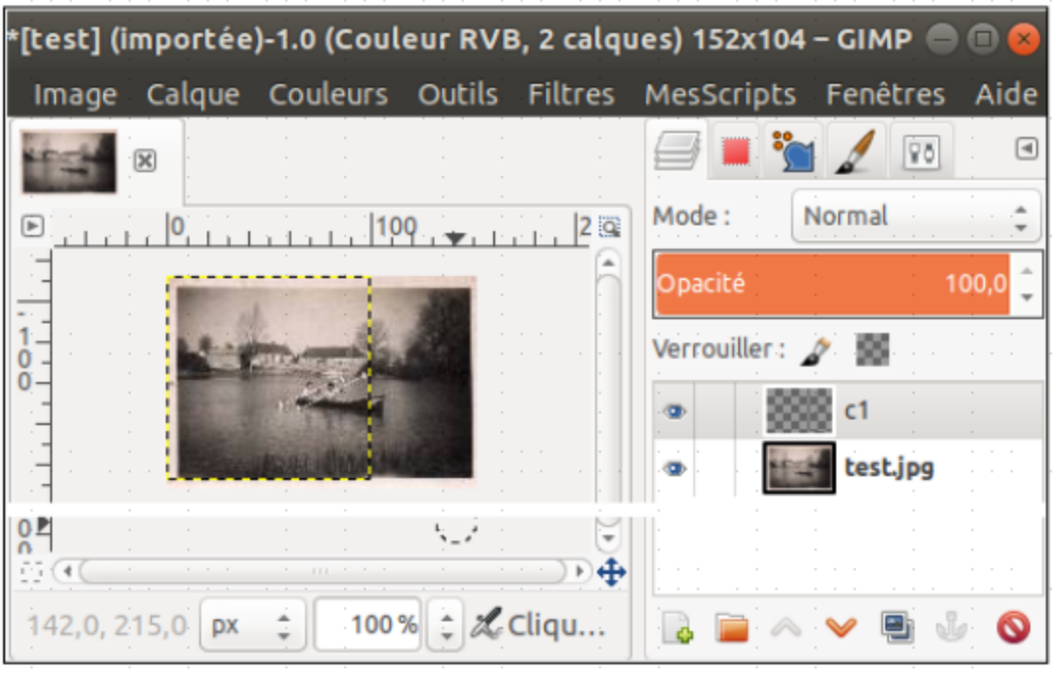
4.8.1 Mode interactif

Pour ajouter un nouveau calque au dessus d'une image ouverte dans Gimp, on sélectionne l'image au dessus de laquelle on veut ajouter un nouveau calque puis on clique le menu **Calque > Nouveau Calque...**

Dans la boîte de dialogue qui s'ouvre, on indique le nom du calque, la taille du calque (largeur et hauteur en pixels) et le mode de remplissage (premier-plan, arrière plan, blanc ou transparent) puis on clique Valider.



Le nouveau calque apparaît alors aussitôt dans la fenêtre des calques, au dessus de la pile des calques qui étaient déjà existants



4.8.2 Mode non interactif

En mode non interactif, on utilise les instructions suivantes pour ajouter un calque c1 au dessus d'une image. Le calque c1 est rempli avec la couleur de premier plan (noir par défaut).

```
(define in1 "/home/ubuntu/Images/test.jpg")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(gimp-display-new img1)
(define c1 (car (gimp-layer-new img1 100 100 RGB-IMAGE "c1" 100 1)))
(gimp-image-insert-layer img1 c1 0 0)
```

Pour que le calque c1 ne soit pas rempli de noir mais soit transparent on remplace RGB-IMAGE par RGBA-IMAGE.

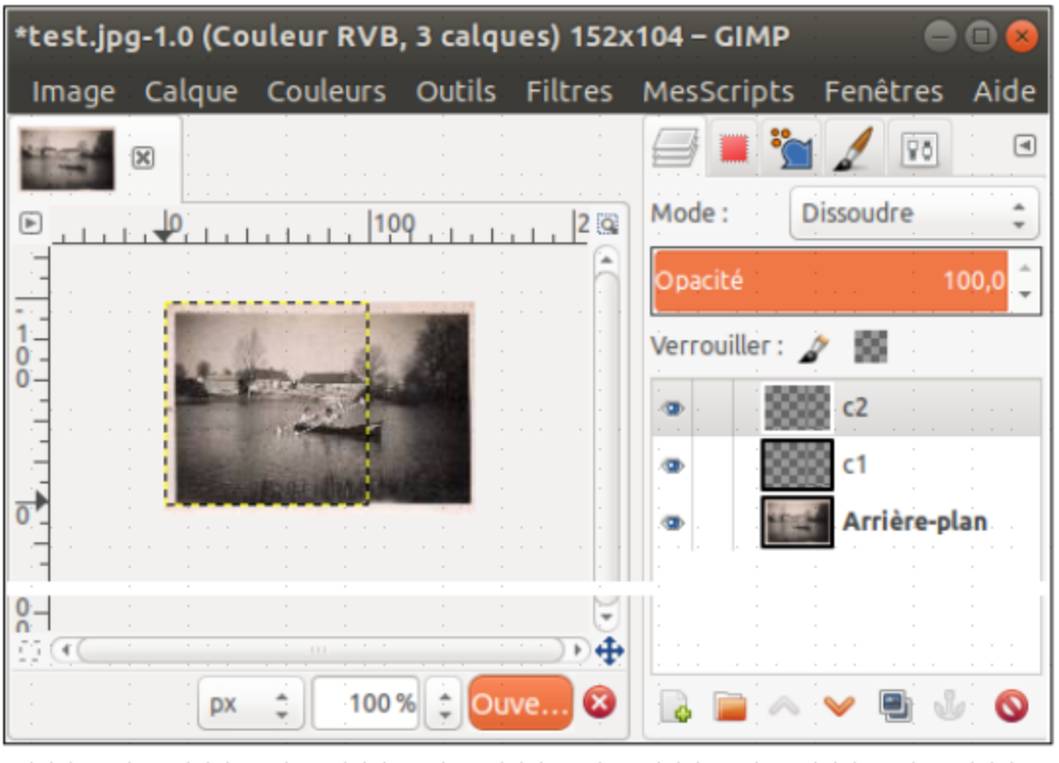
```
(define c1 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c1" 100 1)))
(gimp-image-insert-layer img1 c1 0 0)
```

Pour ajouter deux calques, c1 au dessus de l'image et c2 au dessus de c1, on utilise les instructions suivantes.

```
(define in1 "/home/ubuntu/Images/test.jpg")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(gimp-display-new img1)
(define c1 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c1" 100 1)))
(gimp-image-insert-layer img1 c1 0 0)
(define c2 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c2" 100 1)))
(gimp-image-insert-layer img1 c2 0 0)
```

Réponse de la console

```
in1img1(1)c1(#t)c2(#t)
```



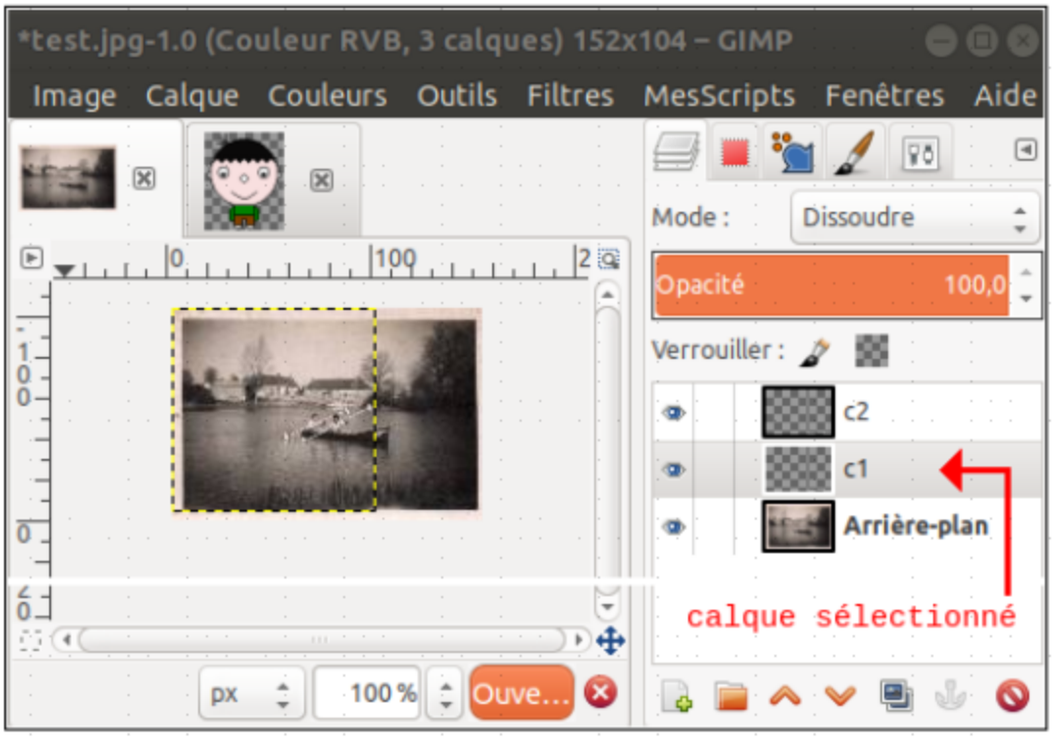
4.9) Coller dans un calque

On a vu en 4.7) comment on procède pour coller le contenu du presse-papiers dans le calque associé à une image dans le cas où cette image ne comporte qu'un seul calque (elle-même).
Dans le cas où l'image possède un (ou plusieurs) calque(s) au dessus d'elle, il est nécessaire de choisir le calque dans lequel on veut que soit copié le contenu du presse-papiers

4.9.1 Mode interactif

En mode interactif, pour coller le contenu du presse-papiers dans un des calques situés au dessus d'une image, on sélectionne ce calque à l'aide de la souris, puis on clique le menu **Edition > Coller.**

Ensuite on ancre le calque flotttant, en cliquant le bouton en forme d'ancre, comme indiqué en 4.7.1



4.9.2 Mode non interactif

En mode non interactif, pour coller le contenu du presse-papiers dans un des calques associés à une image:

- on précise l'identifiant du calque dans l'instruction **gimp-edit-paste** ;

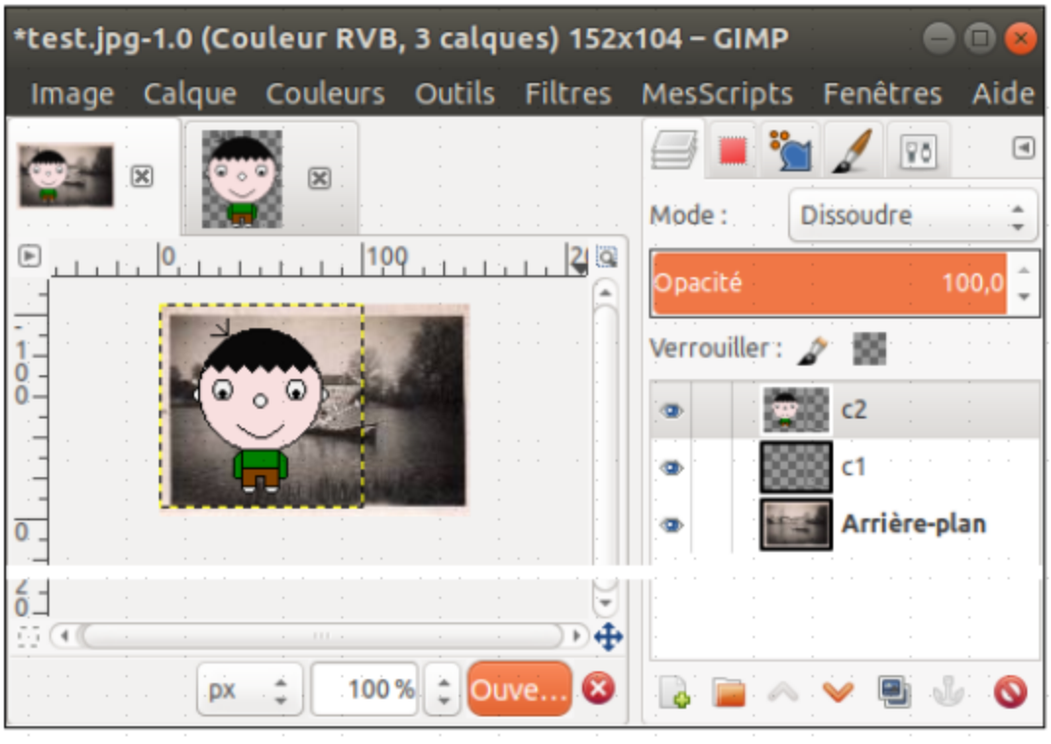
- on ancre le cadre flottant à l'aide de la fonction suivante.
  gimp-floating-sel-anchor

Avec les instructions suivantes, on copie l'image "logo.png" dans le presse-papiers puis on colle son contenu dans le calque c2 associé à l'image "test.jpg"

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

>
(define in1 "/home/ubuntu/Images/test.jpg")
(define in2 "/home/ubuntu/Images/logo.png")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
(define c1 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c1" 100 1)))
(define c2 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c2" 100 1)))
(gimp-image-insert-layer img1 c1 0 0)
(gimp-image-insert-layer img1 c2 0 0)
(gimp-display-new img1)
(gimp-display-new img2)
(define tcimg2 (gimp-image-get-layers img2))
(define lcimg2 (vector->list (cadr tcimg2)))
(gimp-edit-copy (car lcimg2))
(define tcimg1 (gimp-image-get-layers img1))
(define lcimg1 (vector->list (cadr tcimg1)))
(define c1img1 (car lcimg1))
(define sf (gimp-edit-paste c1img1 0))
(gimp-floating-sel-anchor (car sf))
```

Réponse de la console

```
in1in2img1img2c1c2(#t)(#t)(1)(2)tcimg2lcimg2(1)tcimg1lcimg1c1img1sf
```



Ces différentes instructions assurent les tâches suivantes :

Définition des variables in1 et in2 pour contenir le chemin complet des fichiers "test.jpg" et "logo.png"

```
(define in1 "/home/ubuntu/Images/test.jpg")
(define in2 "/home/ubuntu/Images/logo.png")
```

Définitin des identifiants des images "test.jpg" et "logo.png", une fois celles-ci chargées en mémoire.

```
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define img2 (car (gimp-file-load RUN-NONINTERACTIVE in2 in2)))
```

Définition des identifiants des calques c1 et c2 associés à l'image "test.jpg" en mémoire (identifiant img1)

```
(define c1 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c1" 100 1)))
(define c2 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c2" 100 1)))
```

Insertion des calques c1 et c2, dans la pile des calques associés à l'image 1 (fichier "test.jpg", identifiant img1)

```
(gimp-image-insert-layer img1 c1 0 0)
(gimp-image-insert-layer img1 c2 0 0)
```

Affichage des images 1 et 2 ("test.jpg", "logo.png") dans la fenêtre Gimp.

```
(gimp-display-new img1)
(gimp-display-new img2)
```

Définition du tableau des calques de l'image 2 ("logo.png")

```
(define tcimg2 (gimp-image-get-layers img2))
```

Définition de la liste des calques de l'image 2 ("logo.png")

```
(define lcimg2 (vector->list (cadr tcimg2)))
```

Copie dans le presse-papiers du premier calque de la liste lcimg2

```
(gimp-edit-copy (car lcimg2))
```

Définition du tableau des calques de l'image 1 ("test.jpg")

```
(define tcimg1 (gimp-image-get-layers img1))
```

Définition de la liste des calques de l'image l'image 1 ("test.jpg")

```
(define lcimg1 (vector->list (cadr tcimg1)))
```

Définition de l'identifiant du premier calque de la liste (c'est le calque c2 car le début de la liste correspond au haut de la pile de calques)

```
(define c1img1 (car lcimg1))
```

Définition de l'identifiant de la sélection flottante et collage, dedans, du contenu du presse-papiers.

```
(define sf (gimp-edit-paste c1img1 0))
```

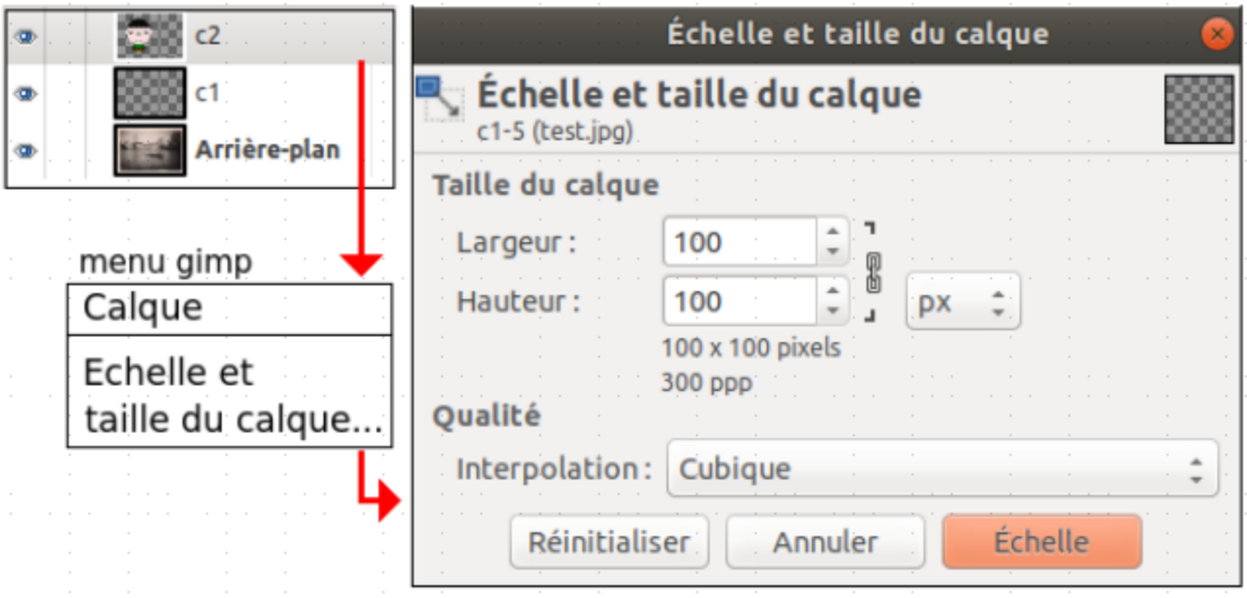
Ancrage de la sélection flottante
```
(gimp-floating-sel-anchor (car sf))
```

4.10) Obtenir les dimensions d'un calque

4.10.1 Mode interactif

En mode interactif, pour connaître les dimensions d'un calque (en pixels), on sélectionne ce calque, dans la pile des calques, puis on clique le menu **Calque > Echelle et taille du calque...**

Dans la boîte de dialogue qui s'ouvre on voit apparaître la largeur et la hauteur de ce calque.



4.10.2 Mode non interactif

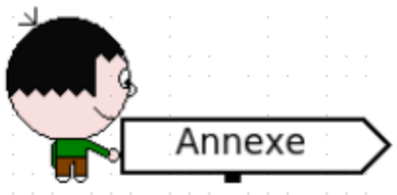
En mode non interactif, pour connaître les dimensions d'un calque (en pixels) on utilise les fonctions suivantes :
gimp-drawable-width et **gimp-drawable-height.**

```
Bienvenue sur TinyScheme
Copyright (c) Dimitrios Souflis
Console Script-Fu - Développement Scheme interactif

> (define in1 "/home/ubuntu/Images/test.jpg")
(define img1 (car (gimp-file-load RUN-NONINTERACTIVE in1 in1)))
(define c1 (car (gimp-layer-new img1 100 100 RGBA-IMAGE "c1" 100 1)))
(gimp-image-insert-layer img1 c1 0 0)
(gimp-display-new img1)
```

```
(define tcimg1 (gimp-image-get-layers img1))
(define lcimg1 (vector->list (cadr tcimg1)))
(define c1img1 (car lcimg1))
(define wc1img1 (gimp-drawable-width c1img1))
(define hc1img1 (gimp-drawable-height c1img1))
wc1img1
hc1img1
```

Réponse de la console

```
in1img1c1(#t)(1)tcimg1lcimg1c1img1wc1img1hc1img1(100)(100)

> wc1img1
(100)
> hc1img1
(100)
```



Annexe

On regroupe, dans les pages suivantes, les listings des principaux scripts réalisés dans le livre.

Le script **1** crée une nouvelle image vide, dans la fenêtre de Gimp.

Les scripts **2** à **7** sont des scripts qu'on applique à une seule image, ouverte ou créée en mode interactif à partir des menus de Gimp. Une fois modifiée par l'un de ces scripts, l'image résultante doit être enregistrée de façon interactive à l'aide du menu **Fichier > Exporter** de Gimp.

Les scripts **8** à **11** sont des scripts qu'on applique à un lot d'images qui se trouvent dans un répertoire d'entrée **r1** choisi. Une fois modifiées par l'un de ces scripts, les images résultantes sont automatiquement enregistrées dans un répertoire de sortie **r2** et avec un niveau de qualité jpg choisis.

Scripts appliqués à une image

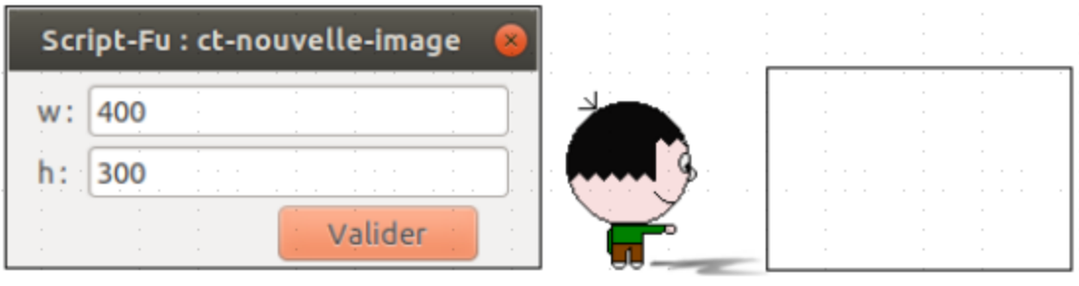
Scripts appliqués à un lot d'images

1) ct-nouvelle-image.scm (chapitre 2.5 – page 34)

Sous **Windows**, le chemin **"/home/ubuntu/essai/sauv.jpg"** devient par exemple **"F:\\essai\\sauv.jpg"** (F étant ici une clef USB...).

D'autre part, la console **Gimp 2.10** sous **Windows 10** propose d'utiliser le paramètre FILL-BACKGROUND (au lieu de BACKGROUND-FILL avec Gimp 2.8 sous Ubuntu 18.04) mais les deux paramètres fonctionnent.

```
;-----------------------------------------------------------------------
; ct-nouvelle-image.scm - version Linux
; sudo cp ct-nouvelle-image.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------------
;
(
define (ct-nouvelle-image w h)
(let* ( (img (car (gimp-image-new w h RGB )))
(c (car (gimp-layer-new img w h RGB-IMAGE "c1" 100 NORMAL-MODE))))
(gimp-context-set-background  '(255 255 255))
(gimp-image-insert-layer img c 0 0)
(gimp-drawable-fill c BACKGROUND-FILL)
(gimp-display-new img)
(gimp-file-save 1 img c "/home/ubuntu/essai/sauv.jpg" "")
))
(
script-fu-register
"ct-nouvelle-image"
"<Image>/MesScripts/ct-nouvelle-image"
"Mon commentaire"
"Essai de script Gimp"
"Claude Turrier"
"2016"
"-----------"
SF-VALUE "w" "400"
SF-VALUE "h" "300"
)
```

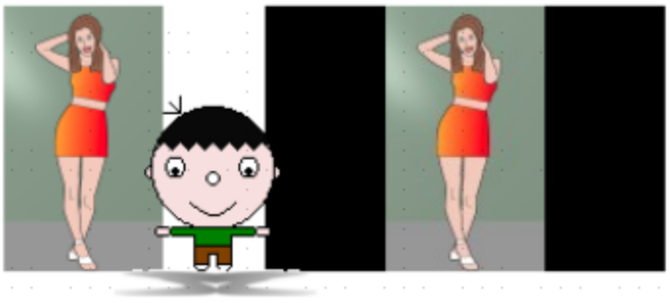


2) ct-paysage-3-2.scm (chapitre 2.6 page 45)

Les versions Linux et Windows sont identiques.

```
;-----------------------------------------------------------------------
; ct-paysage-3-2.scm
; version Linux
; sudo cp ct-paysage-3-2.scm /usr/share/gimp/2.0/scripts/
; Version Windows 10
; Copier dans "C:\Programmes\GIMP2\share\gimp\2.0\scripts"
;-----------------------------------------------------------------------
;
(define (ct-paysage-3-2 img)
;-déclarer les variables utilisées par le script
(let* ((nw 0) (nh 0) (ox 0) (oy 0) (u 0) (v 0) (i 0) ( j 0) (k 0)
(h (car (gimp-image-height img))) (w (car (gimp-image-width img))))
;-fixer la couleur de remplissage à noir
(gimp-context-set-background '(0 0 0))
;-placer la hauteur et la largeur de l'image ouverte dans u et v
(set! u  w)  (set! v  h)  (set! k   (* (/ 3 2) v) )
;-quel que soit le format de l'image ouverte on le transforme  3/2
;-si la hauteur de l'image est inférieure à sa largeur,
;-on la complète avec du noir en haut et en bas
;-sinon on la complète avec du noir à gauche et à droite
;
;-cas n°1 : w > h et w > 1,5 h
(if (> u  v)
(if (> u  k)
   (begin
   (set! i (/ 2 3)) (set! nh (* w i))
   (set! i (/ 1 3)) (set! j (* w i)) (set! i (/ h 2)) (set! oy (- j i ))
   (gimp-image-resize img w nh 0 oy)
)));end begin  ;end if  ;end if
;
;-cas n°3 : w > h et w < 1,5 h
(if (> u  v)
(if (< u  k )
   (begin
   (set! i (/ 3 2)) (set! nw (* h i))
  (set! i (/ 3 4)) (set! j (* h i)) (set! i (/ w 2)) (set! ox (- j i))
  (gimp-image-resize img nw h ox 0)
)));end begin  ;end if  ;end if
;
;-cas n° 4 et 5 : w < ou = h
(if (<= u  v)
  (begin
  (set! i (/ 3 2)) (set! nw (* h i))
  (set! i (/ 3 4)) (set! j (* h i)) (set! i (/ w 2)) (set! ox (- j i))
  (gimp-image-resize img nw h ox 0)
));end begin ;end if
;
;-aplatir l'image
(gimp-image-flatten img)
;-afficher le résultat à l'écran
(gimp-display-new img)
```

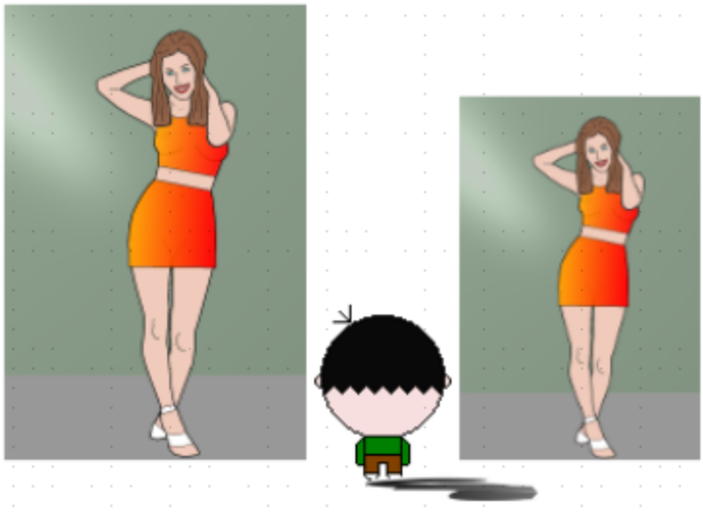
```
)) ; fin de let* ; fin de define
;
;--Enregistrer le script
(script-fu-register "ct-paysage-3-2"
"<Image>/MesScripts/ct-paysage-3-2"
"------------------------------"
"Met les images au format 3/2"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0
)
```



3) ct-scale.scm (chapitre 2.7 – page 48)

Les versions Linux et Windows sont identiques.

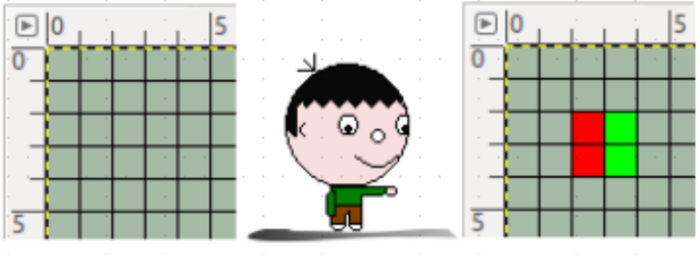
```
;-----------------------------------------------------------------------
; ct-scale.scm
; version Linux
; sudo cp ct-scale.scm /usr/share/gimp/2.0/scripts/
; Version Windows
; Copier dans "C:\Programmes\GIMP2\share\gimp\2.0\scripts"
;-----------------------------------------------------------------------
;
(define (ct-scale img nw)
(let* ((nh 0) (k 0) (h (car (gimp-image-height img)))
       (w (car (gimp-image-width img))))
(set! k (/ w h))
(set! nh (/ nw k))
(gimp-image-scale img nw nh)
(gimp-image-flatten img)
(gimp-display-new img)
)
)
(script-fu-register "ct-scale"
"<Image>/MesScripts/ct-scale"
"------------------------------"
"Redimensionne une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;->img
SF-VALUE "nw" "100"    ;->nw
)
```



4) ct-dessin.scm (cf 2.8 - page 55)

Les versions Linux et Windows sont identiques.

```
;-----------------------------------------------------------------------
; ct-dessin.scm
; version Linux
; sudo cp ct-dessin.scm /usr/share/gimp/2.0/scripts/
; Version Windows
; Copier dans "C:\Programmes\GIMP2\share\gimp\2.0\scripts"
;-----------------------------------------------------------------------
;
(define (ct-dessin img )
;--------Déclarer les variables---
(let* ( (c 0) (monpixel (make-vector 4 'byte))
(h (car (gimp-image-height img))) (w (car (gimp-image-width img))))
;-Créer un nouveau calque transparent et l'ajouter à l'image en cours
(set! c (car (gimp-layer-new img w h RGBA-IMAGE "c1" 100 NORMAL-MODE)))
(gimp-image-insert-layer img c 0 0)
;--- Dessiner les pixels sur le calque transparent ajouté à l'image
(vector-set! monpixel 0 255) ;rouge=255
(vector-set! monpixel 1 0) ;vert=0
(vector-set! monpixel 2 0) ;bleu=0
(vector-set! monpixel 3 255) ;alpha=255
(gimp-drawable-set-pixel c 2 2 4 monpixel)
(gimp-drawable-set-pixel c 2 3 4 monpixel)
(vector-set! monpixel 0 0) (aset monpixel 1 255) (aset monpixel 2 0)
(vector-set! monpixel 3 255) ;rgba=(0,255,0,255)
(gimp-drawable-set-pixel c 3 2 4 monpixel)
(gimp-drawable-set-pixel c 3 3 4 monpixel)
;------ Aplatir l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistrer le script par
(script-fu-register "ct-dessin"
"<Image>/MesScripts/ct-dessin"
"------------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```

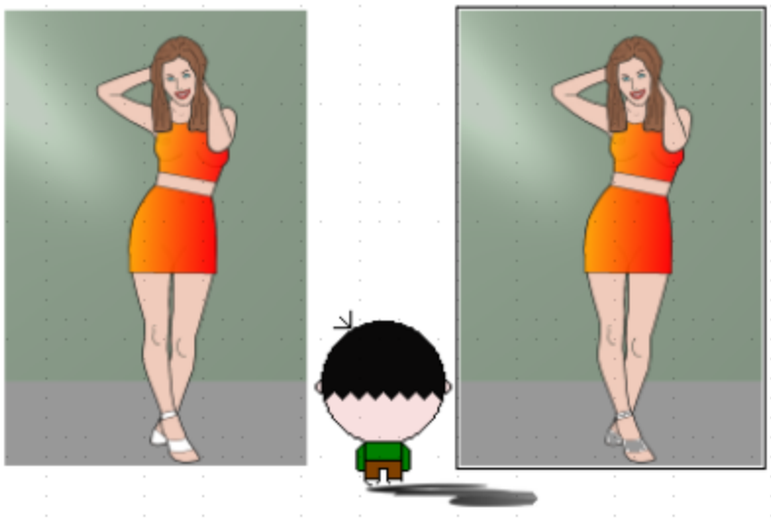


5) ct-cadre.scm (cf 2.9 page 58)

Les versions Linux et Windows sont identiques.

```
;-----------------------------------------------------------------------
; ct-cadre.scm
; version Linux
; sudo cp ct-cadre.scm /usr/share/gimp/2.0/scripts/
; Version Windows
; Copier dans "C:\Programmes\GIMP2\share\gimp\2.0\scripts"
;-----------------------------------------------------------------------
;
(define (ct-cadre img )
;--------Déclare les variables---
(let* ( (c 0)
(couleur_ext '(0 0 0)) ; couleur de la partie extérieure du cadre
(couleur_int '(255 255 255)) ; couleur de la partie intérieure du cadre
(n 10) ; nombre de coordonnées des points constituant le tracé
(moncadre (make-vector 10 'double))
;-------------------
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(u w) (v h) (nw (+ w 4)) (nh (+ h 4)) )
;-Crée un nouveau calque transparent et l'ajoute à l'image en cours
(gimp-image-resize img nw nh 2 2)
(set! c (car (gimp-layer-new img nw nh RGBA-IMAGE
                "MonCalque" 100 NORMAL-MODE)))
(gimp-image-insert-layer img c 0 0)
;-----Dessin du double cadre-------------
(gimp-brush-new "mabrosse")
(gimp-brushes-refresh)
(gimp-context-set-brush "mabrosse")
(gimp-context-set-brush-aspect-ratio 0)
;--cadre extérieur noir de 1 pixel d'épaisseur
(gimp-context-set-foreground couleur_ext) ;noir
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 1))
(set! v (- nh 1))
(vector-set! moncadre 0 0) ; point 1 (x)
(vector-set! moncadre 1 0) ; point 1 (y)
(vector-set! moncadre 2 u) ; point 2 (x)
(vector-set! moncadre 3 0) ; point 2 (y)
(vector-set! moncadre 4 u) ; point 3 (x)
(vector-set! moncadre 5 v) ; point 3 (y)
(vector-set! moncadre 6 0) ; point 4 (x)
(vector-set! moncadre 7 v) ; point 4 (y)
(vector-set! moncadre 8 0) ; point 5 (x)
(vector-set! moncadre 9 0) ; point 5 (y)
(gimp-pencil c n moncadre ); tracé dans l'ordre des points
;--cadre intérieur blanc de 1 pixel d'épaisseur
(gimp-context-set-foreground couleur_int) ; blanc
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 2))
(set! v (- nh 2))
(vector-set! moncadre 0 1)
(vector-set! moncadre 1 1)
```

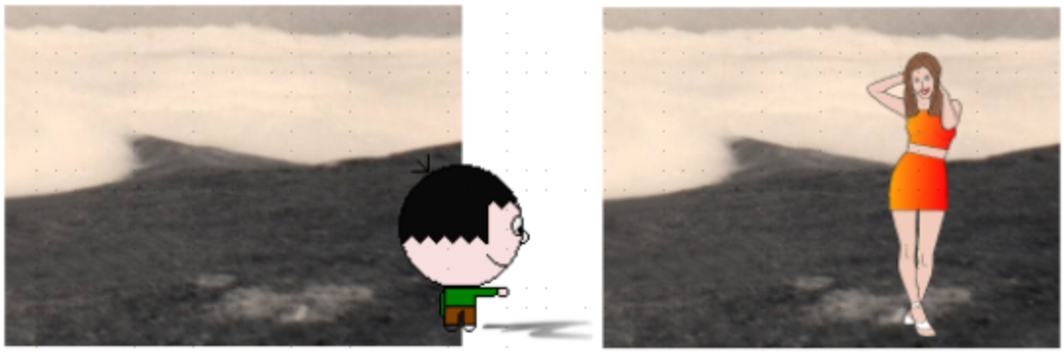
```
(vector-set! moncadre 2 u)
(vector-set! moncadre 3 1)
(vector-set! moncadre 4 u)
(vector-set! moncadre 5 v)
(vector-set! moncadre 6 1)
(vector-set! moncadre 7 v)
(vector-set! moncadre 8 1)
(vector-set! moncadre 9 1)
(gimp-pencil c n moncadre )
;------ Aplatit l'image ---------
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;------Enregistre le script
(script-fu-register "ct-cadre"
"<Image>/MesScripts/ct-cadre"
"------------------------------"
"Dessine des pixels sur une image"
"Claude Turrier"
"2016"
"------------------------------"
SF-IMAGE "Mon Image" 0 ;-> img
)
```



6) ct-masque.scm (cf 2.10 - page 69)

Les versions Linux et Windows sont identiques.

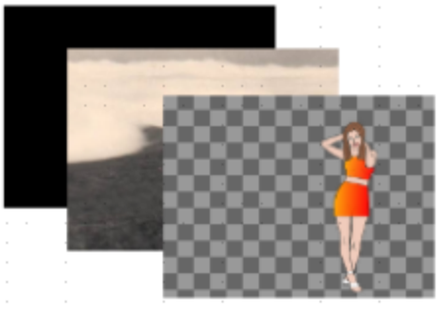
```
;-------------------------------------------------------------------
; ct-masque.scm
; version Linux
; sudo cp ct-masque.scm /usr/share/gimp/2.0/scripts/
; Version Windows
; Copier dans "C:\Programmes\GIMP2\share\gimp\2.0\scripts"
;-------------------------------------------------------------------
(define (ct-masque img masq )
(let*
(
(px 0.1) (wm 90) (hm 90)
(w (car (gimp-image-width img))) (h (car (gimp-image-height img)))
(r (/ hm wm))
(nw (* px w)) (nh (* nw r))
(dx (- w nw)) (dy (- h nh))
(c (car (gimp-file-load-layer 1 img masq)))
)
(gimp-image-add-layer img c 0)
(gimp-layer-scale c nw nh 0)
(gimp-layer-resize-to-image-size c)
(gimp-layer-translate c dx dy)
(gimp-layer-resize-to-image-size c)
(gimp-image-flatten img)
(gimp-display-new img)
) ; fin de let*
) ; fin de define
;-----enregistrer le script -----
(script-fu-register "ct-masque"
"<Image>/MesScripts/ct-masque"
"------------------------------"
"Ajoute un masque au dessus d'une image"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "img" 0
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/essai/masque.png"
)
```



7) ct-scalques.scm (cf 2.11 - page 71)

Sous **Windows**, le chemin **"/home/ubuntu/essai/image_"** devient par exemple **"F:\\essai\\image_"** (F étant ici une clef USB...).

```
;-----------------------------------------------------------------------
; ct-scalques.scm
; version Linux
; sudo cp ct-scalques.scm /usr/share/gimp/2.0/scripts/
; -----------------------------------------------------------------------
(
define (ct-scalques img0)
;-->début let1
(let* ((chemin  "/home/ubuntu/essai/image_"))
;-->début letloop
(let loop ((lc (vector->list (cadr (gimp-image-get-layers img0)))))
;-->début unless
(unless (null? lc)
(gimp-edit-copy (car lc))
;-->début let2
(let* (
(nom (car (gimp-item-get-name (car lc))))
(out (string-append chemin nom ".png"))
(imgsauv  (car (gimp-edit-paste-as-new)))
(cc (aref (cadr (gimp-image-get-layers imgsauv)) 0)))
(file-png-save 1 imgsauv cc out out 0 0 0 0 0 0 0)
);<--fin let2
(loop (cdr lc)); retour vers  début unless
))));<--fin unless, letloop, let1, define
;
(script-fu-register "ct-scalques"
"<Image>/MesScripts/ct-scalques"
"------------------------------"
"Enregistre tous les calques"
"Claude Turrier"
"2021"
"------------------------------"
SF-IMAGE "Image" 0 ;img0
)
```



8) ct-rotate.scm (cf 3.8 - page 87)

Sous **Windows**, le chemin "/*.jpg" devient "*.jpg").

```
;-----------------------------------------------------------------------
; ct-rotate.scm
; version Linux
; sudo cp ct-rotate.scm /usr/share/gimp/2.0/scripts/
;;=============================================================
;; CHARGER L'IMAGE COURANTE EN MEMOIRE
;;=============================================================
;; tlot:nom du script | r1 et r2: répertoires d'entrée et de sortie
(define (ct-rotate r1  r2)
;; liste:liste des chemins complets des images à traiter
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))))
;; parcours de la liste, du début à la fin
(while (not (null? liste))
;; in:chemin complet de l'image courante à traiter
(let* ((in (car liste))
;; img:image courante à traiter, chargée en mémoire
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
;; out:chemin complet de l'image courante à enregistrer
(out (string-append r2 "/" (car (gimp-image-get-name img))))
;; c:calque actif courant
(c (car (gimp-image-get-active-drawable img))))
;;=====================================================================
;; TRAITER L'IMAGE COURANTE
;; placer ici le code du traitement à appliquer
;; à chacune des images
;;=====================================================================
(gimp-rotate c 1 1.5708 )
;;=====================================================================
;; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;;=====================================================================
;; enregistrement de l'image courante après traitement
(file-jpeg-save RUN-NONINTERACTIVE img c out out 1 0 1 0 "" 0 1 0 0)
;; fermeture de l'image courante
(gimp-image-delete img)
)
;; mise à jour de la liste des images à traiter
(set! liste (cdr liste))
)))
;;=====================================================================
;; ENREGISTREMENT DU SCRIPT
;;=====================================================================
(
script-fu-register "ct-rotate"
"<Image>/MesScripts/ct-rotate"
"Traitement par lot"
"*********"
"CT"
"2021"
""  ;types d'images
SF-DIRNAME  "Répertoire d'entrée"   ""  ;r1
SF-DIRNAME  "Répertoire de sortie"  "" ;r2
)
```

9) ct-lotredim.scm (cf 3.9 - page 93)

Sous **Windows**, les adresses doivent être modifiées:
"/*.jpg" devient "*.jpg").
"/home/ubuntu/essai" devient "C:\\Users\\ct\\Documents\\essai"
"/home/ubuntu/essai2" devient "C:\\Users\\ct\\Documents\\essai2"

```
;-----------------------------------------------------------------
; ct-lotredim.scm
; version Linux
; Install: sudo cp ct-lotredim.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotredim r1  r2 nw q);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(nh 0) (k 0) ;(nw 100)
(h (car (gimp-image-height img)))
(w (car (gimp-image-width img)))
);fin defvar let2
;-----------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------
(set! k (/ w h))
(set! nh (/ nw k))
(gimp-image-scale img nw nh)
;-----------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;-----------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-----------------------------------------------------------
(
script-fu-register "ct-lotredim"
"<Image>/MesScripts/ct-lotredim"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-VALUE    "Nouvelle largeur" "100"  ;nw
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
)
```

10) ct-lotcadre.scm (cf 3.10 - page 98)

Sous **Windows**, les adresses doivent être modifiées comme précédemment.
"/*.jpg" devient **"*.jpg"**)
et, par exemple,
"/home/ubuntu/essai" devient **"C:\\Users\\ct\\Documents\\essai"**
"/home/ubuntu/essai2" devient **"C:\\Users\\ct\\Documents\\essai2"**

```
;-----------------------------------------------------------------
; ct-lotcadre.scm
; version Linux
; Install: sudo cp ct-lotcadre.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotcadre r1  r2 q couex couin);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1))));let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
;(couex '(0 0 0)) ; couleur de la partie extérieure du cadre
;(couin '(255 255 255)) ; couleur de la partie intérieure du cadre
(n 10) ; nombre de coordonnées des points constituant le tracé
(tc (make-vector 10 'double))
(h (car (gimp-image-height img))) (w (car (gimp-image-width img)))
(u w) (v h) (nw (+ w 4)) (nh (+ h 4))
);fin defvar let2
;
;-----------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------
(gimp-image-resize img nw nh 2 2)
(set! c (car (gimp-layer-new img nw nh 1 "myC" 100 0)))
(gimp-image-insert-layer img c 0 0)
(gimp-brush-new "mabrosse")
(gimp-brushes-refresh)
(gimp-context-set-brush "mabrosse")
(gimp-context-set-brush-aspect-ratio 0)
;
;--cadre extérieur noir de 1 pixel d'épaisseur
(gimp-context-set-foreground couex) ;couleur extérieure choisie
(gimp-context-set-brush-size 1); 1 pixel
(set! u (- nw 1))
(set! v (- nh 1))
(vector-set! tc 0 0) ; point 1 (x)
(vector-set! tc 1 0) ; point 1 (y)
(vector-set! tc 2 u) ; point 2 (x)
(vector-set! tc 3 0) ; point 2 (y)
(vector-set! tc 4 u) ; point 3 (x)
(vector-set! tc 5 v) ; point 3 (y)
(vector-set! tc 6 0) ; point 4 (x)
(vector-set! tc 7 v) ; point 4 (y)
(vector-set! tc 8 0) ; point 5 (x)
```

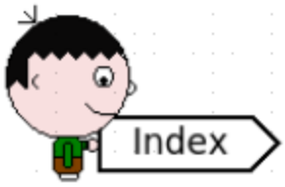
```
(vector-set! tc 9 0) ; point 5 (y)
(gimp-pencil c n tc) ;tracé dans l'ordre des points
;
;--cadre intérieur blanc de 1 pixel d'épaisseur
(gimp-context-set-foreground couin) ;couleur intérieure choisie
(gimp-context-set-brush-size 1) ;1 pixel
(set! u (- nw 2))
(set! v (- nh 2))
(vector-set! tc 0 1)
(vector-set! tc 1 1)
(vector-set! tc 2 u)
(vector-set! tc 3 1)
(vector-set! tc 4 u)
(vector-set! tc 5 v)
(vector-set! tc 6 1)
(vector-set! tc 7 v)
(vector-set! tc 8 1)
(vector-set! tc 9 1)
(gimp-pencil c n tc )
;
(gimp-image-flatten img)
(set! c (car (gimp-image-get-active-drawable img)))
;-----------------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;
;-------------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-------------------------------------------------------------
(
script-fu-register "ct-lotcadre"
"<Image>/MesScripts/ct-lotcadre"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
SF-COLOR   "Couleur extérieure"      '(0 0 0) ;couex
SF-COLOR   "Couleur intérieure"     '(255 255 255)  ;couin
)
```

11) ct-lotlogo.scm (cf 3.11 - page 100)

Sous **Windows**, les adresses doivent être modifiées comme précédemment.

```
;-----------------------------------------------------------------
; ct-lotlogo.scm
; version Linux
; Install: sudo cp ct-lotlogo.scm /usr/share/gimp/2.0/scripts/
;-----------------------------------------------------------------
;-----------------------------------------------------------------
; CHARGER L'IMAGE COURANTE EN MEMOIRE
;-----------------------------------------------------------------
(define (ct-lotlogo r1  r2 q logo);define
(let* ((liste (cadr (file-glob (string-append r1 "/*.jpg") 1)));let1
(imgl (car (file-png-load 1 logo logo)))
(hl (car (gimp-image-height imgl)))
(wl (car (gimp-image-width imgl)))
); fin defvar let1
(while (not (null? liste));while
(let* ((in (car liste)) ;let2
(img (car (gimp-file-load RUN-NONINTERACTIVE in in)))
(out (string-append r2 "/" (car (gimp-image-get-name img))))
(c (car (gimp-image-get-active-drawable img)))
(h (car (gimp-image-height img)))
(w (car (gimp-image-width img)))
(px 0.1) ;(wl 128) (hl 128)
(r (/ hl wl))
(nwl (* px w)) (nhl (* nwl r))
(dx (- w nwl)) (dy (- h nhl))
(cl (car (gimp-file-load-layer 1 img logo)))
);fin defvar let2
;-----------------------------------------------------------------
; TRAITER L'IMAGE COURANTE
;-----------------------------------------------------------------
(gimp-image-add-layer img cl 0)
(gimp-layer-scale cl nwl nhl 0)
(gimp-layer-resize-to-image-size cl)
(gimp-layer-translate cl dx dy)
(gimp-layer-resize-to-image-size cl)
(gimp-image-flatten img)
;
(set! c (car (gimp-image-get-active-drawable img)))
;-----------------------------------------------------------------
; PREPARER LE PASSAGE A l'IMAGE SUIVANTE
;-----------------------------------------------------------------
(file-jpeg-save RUN-NONINTERACTIVE img c out out q 0 1 0 "" 0 1 0 0)
(gimp-image-delete img)
); fin let2
(set! liste (cdr liste))
)))  ;fin while, let1, define
;-----------------------------------------------------------
; ENREGISTRER LE SCRIPT
;-----------------------------------------------------------
(
```

```
script-fu-register "ct-lotlogo"
"<Image>/MesScripts/ct-lotlogo"
"------------------------------"
"traitement par lot"
"Claude Turrier"
"2021"
"------------------------------"
SF-DIRNAME  "Répertoire d'entrée"   "/home/ubuntu/essai"  ;r1
SF-DIRNAME  "Répertoire de sortie"  "/home/ubuntu/essai2" ;r2
SF-ADJUSTMENT  "Qualité"   '(1 0 1 0.1 1 1 1) ;q
SF-FILENAME   "SF-FILENAME" "/home/ubuntu/Images/logo.png";logo
)
```



Index

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z