M1 Imageurs, Poitiers 2021

Introduction aux macros ImageJ

Fabrice P. Cordelières

26-28 avril 2021

Matériel de cours
disponible sur GitHub

Corrigés en pièces jointes au pdf

Table des matières

Chapitre 1: Les bases du langage macro ImageJ

Ce premier chapitre est destiné à vous montrer les points communs, mais surtout les différences entre les langages de programmation avancés que vous pouvez connaître et le langage de scripting propre à ImageJ. Le langage macro, quoique moins puissant et moins formel qu'un vrai langage de programmation, permet une automatisation rapide de tâches, soit pour une utilisation directe, soit pour le prototypage de tâches complexes qui pourront faire ultérieurement l'objet de modules spécifiques, codés en Java.

L'outil de base

Le Macro Recorder

Le `Macro Recorder` permet d'enregistrer (presque) toutes les opérations réalisées par l'utilisateur. Il suffit de le lancer pour voir se créer une série d'instructions permettant de reproduire le protocole de traitement et d'analyse sur une autre image. Il se trouve dans le menu `Plugins>Macros>Record` et fonctionne de la même manière sous ImageJ et Fiji.

Éditer une séquence et la sauvegarder : .txt ou .ijm ?

Bien que permettant l'édition des instructions, le `Macro Recorder` n'est pas l'éditeur à proprement parler des macros ImageJ. Pour le faire apparaître, il suffit de cliquer sur le bouton `Create`.

Un éditeur plus complet s'affiche, différent suivant qu'on utilise ImageJ ou Fiji. Sous ImageJ, l'éditeur est un simple éditeur de texte, sans coloration syntaxique. En revanche, il embarque quelques fonctionnalités intéressantes :

- Un outil de déboggage, accessible via le menu `Debug`. Il permet notamment l'exécution pas-à-pas de la macro.
- Le `Function Finder`, accessible via le menu `Macros>Functions Finder...` : il permet de visualiser et de réaliser une recherche dans l'ensemble des fonctions macros. Le menu `Help>Macro Functions...` renvoie vers la **page de référence du site d'ImageJ.**

Sous Fiji, l'éditeur de macros offre une coloration syntaxique. En revanche, le `Function Finder` n'est pas disponible : il est remplacé par l'auto-complétion et une aide contextuelle. Les outils de deboggage/d'exécution pas-à-pas ne sont pas disponibles. Une astuce permet néanmoins de faire apparaître l'éditeur de macros ImageJ sous Fiji : il suffit pour d'ouvrir le fichier macro après en avoir ôté l'extension.

Justement, quelle extension pour nos fichiers macros ? Deux conventions sont possibles : au final, le fichier est un simple fichier texte. Peu importe, on pourra utiliser au choix txt ou ijm (Image J Macro).

Exécuter une macro ou la stopper

Sous ImageJ, pour exécuter une macro, depuis la fenêtre de l'éditeur, aller dans le menu `Macros>Run Macro` (raccourci clavier `CTRL` ou `CMD+R`). Pour en stopper l'exécution, presser la touche `Esc`.

Pour exécuter une macro sous Fiji, utiliser le bouton `Run` en bas de la fenêtre de l'éditeur ou le menu `Macros>Run Macro` (raccourci clavier `CTRL` ou `CMD+R`). Pour en stopper l'exécution, presser la touche `Esc` ou le bouton `Kill` en bas de la fenêtre de l'éditeur.

Les fonctions intégrées

Structure des fonctions

Plusieurs structures de commandes co-existent :

- `run("Nom_De_Commande", "Arguments")` : mot-clé run prenant 2 arguments 1-le nom de la fonction, 2-une chaîne de caractères contenant les arguments, séparés par des espaces ;
- `nomDeCommande("Nom_De_Commande")` : nom de la fonction, suivi d'une chaîne de caractères contenant les arguments.

Ce ne sont que 2 des formes des commandes macros : il n'y a pas de consensus sur la structure des fonctions. La raison est historique : les fonctions en "run" correspondent aux premières fonctions implémentées à l'origine du logiciel, les fonctions plus récentes reprenant une syntaxe plus proche du Java.

Trois logiques différentes

Les instructions peuvent adopter trois types de fonctionnement différents qu'il est important de distinguer :

- **Les méthodes :** ce sont des fonctions qui exécutent une opération ex : redimensionner une image, ajouter une coupe à une pile etc ;
- **Les fonctions :** elles renvoient un résultat qui peut être stocké dans une variable ;
- **Les méthodes de collecte :** c'est le cas par exemple de la fonction `getStatistics(area, mean, min, max, std, histogram)` dont l'appel stocke dans les variables fournies en arguments les informations demandées.

NB : Tout comme en Java, le ; est obligatoire, utilisé pour marquer la fin d'une instruction.

Éléments de syntaxe

Trouver ses marques avec ce nouveau langage

A l'évidence, pour des tâches simples, le `Macro Recorder` peut suffire. En revanche, dès lors qu'il faudra modifier dynamiquement un paramètre, boucler un processus sur un ensemble d'éléments ou adapter le comportement de la macro en fonction d'une entrée, il deviendra nécessaire de connaître la syntaxe d'un ensemble d'éléments de base que sont les variables, les boucles et les structures de choix. Comme dans tout langage de programmation, il sera également important de documenter son code : il est nécessaire de connaître les balises à utiliser.

Commenter son code : quelles balises ?

Les balises de commentaires sont identiques aux balises utilisées en Java :

- **Les commentaires courts :** ils sont délimités par une seule balise, le `//` ;
- **Les commentaires longs :** la balise ouvrante est `/*`, la balise fermante est `*/`.

Les variables : types, opérateurs et portée

Types de variables

Dans le langage Macros ImageJ, les variables ne sont pas typées. En revanche, trois catégories de variables sont utilisables :

- **Les variables simples :** elles sont déclarées par leur nom. L'attribution d'une valeur se fait au moyen du signe égal. Les guillemets permettent de définir la nature "chaîne de caractère" du contenu. ex : `maVariable=3;` ou `monAutreVariable="chaîne de caractères";;`
- **Les variables de type tableau :** elles sont déclarées par leur nom et l'attribution (=) suivi du mot-clé `newArray()`. Cette instruction peut prendre trois types d'arguments (voir ci-après). *Attention : le langage Macros ImageJ ne permet pas de créer de tableaux multi-dimensionnels.*
  - `monTableau=newArray();` : le tableau est initialisé mais n'a pas de dimension ;
  - `monTableau=newArray(n);` : le tableau est initialisé sans contenu mais à une dimension de "n" cases ;
  - `monTableau=newArray(1, 2, 3);` : le tableau est initialisé et dimensionné en fonction du contenu. Ses cases sont initialisée telles que `monTableau[0]` contient la valeur 1, `monTableau[1]` contient la valeur 2 etc. On peut déterminer la dimension du tableau en ayant recours à l'instruction `monTableau.length`.
- **La liste de couples clé-valeur :** On peut utiliser *une unique liste de clés-valeurs*. Voici une liste des fonctions les plus utiles :
  - `List.clear()` : efface le contenu de la liste actuelle ;
  - `List.size` : renvoie le nombre d'éléments dans la liste actuelle ;
  - `List.set(cle, valeur)` : ajoute ou modifie la paire clé-valeur dans la liste ;
  - `List.get(cle)` : renvoie la valeur associée à la clé en tant que chaîne de caractères, ou une chaîne vide si la clé n'existe pas dans la liste ;
  - `List.getValue(cle)` : renvoie la valeur numérique associée à la clé. Si elle est absente de la liste ou n'est pas une valeur numérique, l'exécution de la macro est stoppée.

Opérateurs

Les opérateurs applicables aux variables sont résumés dans le tableau pages suivante. Certaines opérations entre variables de nature différentes aboutissent à un transtypage implicite en chaînes de caractères au moment où l'opération est réalisée.

```
myVariable1 = 1
myVariable2 = " variable de type chaîne"
myVariable3 = myVariable1 + myVariable2

print(myVariable3)
```

L'exécution de la macro renvoie "1 variable de type chaîne" dans la fenêtre `Log`

Opérateur	Priorité	Description
++	1	Pré ou post incrément
−−	1	Pré ou post décrément
−	1	Soustraction bit à bit
!	1	Complémentaire du booléen
~	1	Bit complémentaire
*	2	Multiplication
/	2	Division
%	2	Reste entier de la division
|	2	Ou logique bitwise (OR, opération sur les bits)
^	2	Ou exclusif logique (XOR, opération sur les bits)
<<, >>	2	Rotation de bit (vers la gauche ou la droite)
+	3	Addition arithmétique ou concaténation de chaînes de caractères
−	3	Soustraction arithmétique
<, <=	4	Comparaisons : plus petit que, plus petit ou égal à
>, >=	4	Comparaisons : plus grand que, plus grand ou égal à
==, !=	4	Comparaisons : égal à, différent de
&&	5	Et logique (AND, opération sur les booléens)
||	5	Ou logique bitwise (OR, opération sur les booléens)
=	6	Attribution
+=, -=, *=, /=	6	Attribution combinée à une opération

Portée des variables

Le langage Macros ImageJ permet de créer ses propres groupes fonctionnels (voir le Chapitre 3 - Groupes fonctionnels). Cette possibilité permet de mieux organiser son code et d'en réutiliser certaines parties dans plusieurs macros sans pour autant avoir à tout re-coder. Ces groupes peuvent nécessiter la déclaration de variables en leur coeur : leur contenu ne sera alors pas disponible à l'extérieur de la fonction à moins qu'un retour explicite soit implémenté. De même, une variable déclarée en dehors de la fonction définie par l'utilisateur n'y sera pas directement utilisable : il faudra alors passer son contenu en argument, au moment de l'appel.

Une autre stratégie est envisageable : modifier la portée des variables. L'utilisation du mot-clé `var` devant une variable placée dans le corps de la macro (plutôt en en-tête, pour plus de lisibilité) permet de la transformer en **variable globale**, accessible depuis n'importe quel point du code.

Les boucles

Boucle définie à priori : for

Sa structure est identique à celle utilisée en Java, à la différence qu'il n'est pas possible d'utiliser un itérateur sur une liste :

```
for(initialisation; condition de poursuite; itération) {
    //Instructions
}
```

Boucle indéfinie à priori : while

Sa structure est identique à celle utilisée en Java :

```
while(condition) {
    //Instructions
}
```

Boucle indéfinie à posteriori : do/while

Cette boucle est exécutée au moins une fois, l'évaluation de la condition s'effectuant à la fin du bloc. Il se déclare comme suit :

```
do {
    //Instructions
} while (condition);
```

Structure de choix

Une seule structure de choix est disponible : la structure if/then/else - else if :

```
if(condition1) {
    //Instructions 1
}else{
    //Instructions 2
}else if(condition2){
    //Instructions 3
}
```

NB : Inutile d'essayer, la structure `switch/case` n'est malheureusement pas implémentée...

Chapitre 2: Premiers pas avec le langage macro ImageJ

Au cours de cette première session nous allons explorer le langage macro et découvrir les outils à disposition pour résoudre la problématique posée (voir l'encart "problématique").

Par où commencer ?

Ayant pris connaissance de la problématique, deux étapes devront être réalisées :

1. Identifier les étapes du protocole de traitement et d'analyse ;
2. Identifier les instructions clés à utiliser.

Les étapes du protocole de traitement et d'analyse

L'utilisateur a fourni un protocole détaillé. Le plus simple est donc de reprendre chaque étape proposée et d'initier la macro en utilisant chaque item comme un commentaire. L'espace laissé libre entre chaque commentaire sera progressivement complété avec les instructions adéquates.

Étape 01 : à vous de jouer !

A faire :
- Créer une nouvelle macro : `Plugins>New>Macro`
- En utilisant la syntaxe décrite au "Chapitre 1 - Commenter son code", créer un commentaire par étape de traitement

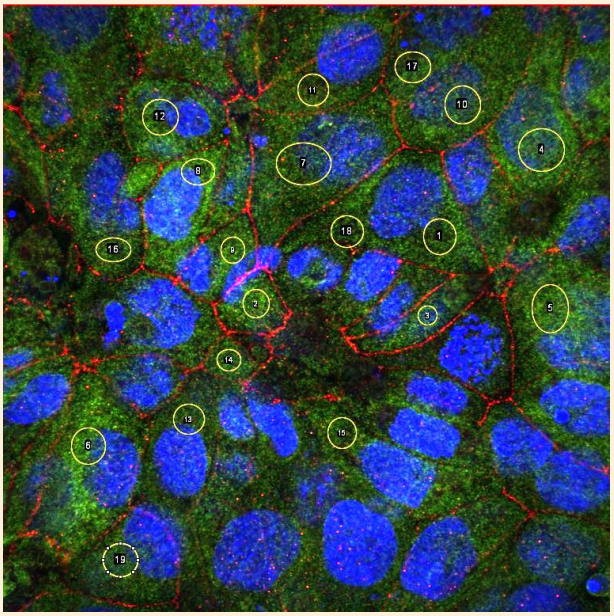
Identifier les instructions clés à utiliser

ImageJ dispose d'une fonction permettant d'enregistrer les opérations réalisées par l'utilisateur sous la forme d'un script : c'est le `Macro Recorder`, activé au moyen du menu `Plugins>Macros>Record...`.

Certaines fonctions ne sont pas enregistrables, mais peuvent néanmoins être utilisées. En complément au `Macro Recorder` il existe une *page de référence* listant l'ensemble des fonctions accessibles en scripting. En outre, nombre de fonctions documentées incluent des exemples d'utilisation.

D'ores et déjà, au moment de l'enregistrement des instructions, il est important d'anticiper deux points en se posant deux questions. *De quelle situation partons-nous ?* En d'autres termes, une analyse précédente aura-t-elle une incidence sur la nouvelle analyse, en laissant, par exemple, des éléments pouvant compromettre l'analyse courante. *L'analyse actuelle est-elle généralisable ?* Les dénominations des éléments manipulés sont-elles fixes ? Que se passe-t-il si le nom de l'image est différent ? Il est important de prêter attention aux arguments des fonctions utilisées et à la manière dont on pourra les manipuler ultérieurement.

La problématique



Si cela te convient, je voudrais cette année proposer un projet assez simple (quoi que...) mais dont l'automatisation presque complète me permettrait de gagner beaucoup de temps.

Sur une acquisition multicouleurs et 3D :

- *empiler tout ou partie du stack (MIP) en laissant le choix à l'utilisateur (option MIP par défaut)*
- *laisser la main à l'utilisateur pour sélectionner autant de régions d'intérêt qu'il le souhaite (20 max) sur cette image*
- *sauver automatiquement ces régions (Roiset.zip) au nom du fichier et l'image avec les ROI incrustées*
- *utiliser ces ROI pour mesurer l'intensité : soit en 3D, soit sur la projection et pour les channels requis (toutes options proposées, sélectionnées par l'utilisateur et si possible conservées par défaut pour l'analyse suivante)*
- *sauvegarder les résultats en format csv (pareil, si possible mémoriser le chemin et le reproposer par défaut).*

Ça te semble jouable ??

Étape 02 : à vous de jouer !

A faire :

- Activer le `Macro Recorder` `Plugins>Macros>Record...`
- Réaliser manuellement les étapes de traitement de l'image, pour le moment en 2D :
  1. Ouvrir une des deux images exemple : `File>Open...`
  2. Réaliser une projection de l'image courante : `Image>Stacks>Z Project...`
  3. Dessiner les régions d'intérêt (ROI) à analyser : utiliser l'outil de sélection approprié
  4. Pousser tour à tour chaque ROI vers le `Roi Manager` : `Edit>Selection>Add to Manager`
  5. Sauvegarder les ROIs dans un fichier : depuis le `ROI Manager`, cliquer sur `More...>Save...`
  6. Paramétrer les grandeurs à mesurer : `Analyze>Set Measurements...`
  7. Mesurer ces grandeurs pour chaque région : depuis le `ROI Manager`, s'assurer qu'aucune région n'est sélectionner en cliquant sur `Deselect` puis mesurer toutes les ROIs en cliquant sur `More...>Multi-Measure` en ne laissant coché que "Measure all 3 slices"
  8. Sauvegarder le tableau de résultats : avec la `Results table` active, faire `File>Save`
- Faire le tri des instructions et les insérer entre les commentaires par simple copier/coller

Tester la macro

La première version de la macro étant prête, il est possible de la tester en la lançant depuis l'éditeur, soit au moyen du bouton `Run` soit du menu `Run>Run`. A noter, il est possible de ne sélectionner qu'une partie du code et de n'exécuter que les lignes concernées au moyen du menu `Run>Run selected code`.

En cas de problème, quelques points à vérifier :

- Manque-t-il un point-virgule pour terminer la ligne précédent la ligne donnée en erreur ?
- Le nom des éléments à manipuler correspondent-ils aux éléments effectivement présents à l'écran ? Vérifier, par exemple, le nom des images à l'écran et les confronter aux noms des images attendus par la macro
- Pour les tables de résultats et le ROI Manager, en cas d'éléments surnuméraires : dans quel état était la ResultsTable/ROI Manager au démarrage de la macro ?
- L'éditeur de macro permet de vérifier les parenthèses, accolades et crochets : chaque balise ouvrante est-elle accompagnée d'une balise fermante ?
- L'instruction qui pose problème est-elle bien orthographiée ? On peut vérifier en enregistrant l'instruction ou en se référant à *cette page*.
- La syntaxe de l'instruction qui pose problème est-elle correcte ? Manque-t-il des arguments ? La nature des arguments est-elle la bonne ? On peut vérifier en enregistrant l'instruction ou en se référant à *cette page*.
- De quelle couleur apparaît l'instruction ? Les arguments ? L'éditeur utilise une coloration basée sur la syntaxe : les commentaires apparaissent en vert, les instructions en orange, les chaînes de caractères en rose et les nombres en violet. Si la couleur n'est pas la couleur attendue, il y a sans doute un problème.

Initialisation et interactions

Initialiser la macro

Au démarrage de la macro, il se peut que plusieurs images soient déjà ouvertes au sein du logiciel. ImageJ travaille systématiquement sur l'image active, qu'elle soit présente au démarrage ou qu'elle vienne juste d'être générée, par exemple par la macro. Afin d'éviter toute confusion, notamment due à la présence de deux images portant un même titre, une pratique possible consiste à fermer au démarrage de la macro toutes les images hormis l'image active.

Fermer une image/fenêtre *(source: doc. en ligne)* :

close()

Closes the active image. This function has the advantage of not closing the "Log" or "Results" window when you meant to close the active image. Use ***run("Close")*** to close non-image windows.

close(pattern)

Closes windows whose title matches 'pattern', which can contain the wildcard characters '*' (matches any character sequence) and '?' (matches single character). For example, close("Histo*") could be used to dispose all histogram windows. Non-image windows like "Roi Manager" have to be specified without wildcards. For text windows, wildcards are allowed if 'pattern' ends with ".txt", ".ijm", ".js" etc. Use close("*") to close all image windows. Use ***close(pattern, "keep")*** to not close text or image windows with changes. If 'pattern' is "\\Others", all images except the front image are closed. The most recent macro window is never closed.

close("*")

Closes all image windows.

close("\\Others")

Closes all images except for the front image.

L'exécution courante de la macro peut faire suite à son exécution sur une autre image. Il est fort probable qu'une liste de ROIs soit déjà présente dans le ROI Manager et qu'un tableau de résultats contienne déjà des données chiffrées. Si on relance en l'état une analyse, les résultats seront sans doute faussés par la présence de données issues des instances précédentes. Lors de l'initialisation, il faudra veiller à vider le tableau de résultats, par

exemple en fermant la fenêtre du tableau ou en utilisant la fonction `Analyse>Clear Results`. Il n'y a pas de fonction directement enregistrable pour fermer le ROI Manager, mais plusieurs fonctions, dont la fonction `reset` permettent de le réinitialiser depuis une macro.

Manipuler le ROI Manager *(source: doc. en ligne)* :

roiManager("count")
Returns the number of ROIs in the ROI Manager list. See also : RoiManager.size and RoiManager.selected.

roiManager("delete")
Deletes the selected ROIs from the list, or deletes all ROIs if none are selected.

roiManager("deselect")
Deselects all ROIs in the list. When ROIs are deselected, subsequent ROI Manager commands are applied to all ROIs.

roiManager("measure")
Measures the selected ROIs, or if none is selected, all ROIs on the list.

roiManager("reset")
Deletes all ROIs on the list.

roiManager("select", index)
Selects an item in the ROI Manager list, where index must be greater than or equal zero and less than the value returned by roiManager("count"). Note that macros that use this function sometimes run orders of magnitude faster in batch mode. Use roiManager("deselect") to deselect all items on the list. For an example, refer to the ROI Manager Stack Demo macro.

Gérer les sorties

Dans le cadre de cette macro, le parti est pris d'avoir une image ouverte au démarrage. En revanche, il est souhaité que les fichiers de résultats soient stockés dans un répertoire défini par l'utilisateur. Une interaction est requise : il faudra trouver un moyen d'afficher une boite de dialogue pour que le répertoire soit sélectionné et que le chemin puisse être stocké dans une variable.

La macro étant supposée faciliter la tâche de l'utilisateur pour l'analyse de fichiers multiples, un soin particulier devra être apporté aux conventions de nommage des fichiers de sortie. Le plus simple serait de les construire en incluant une référence au titre de l'image à laquelle ils se rapportent.

Les instructions de sauvegarde des fichiers ROIs et tableau font apparaître leur chemin sous la forme d'une chaîne de caractères : il sera donc relativement simple de les construire par concaténation.

Gestion des sorties *(source: doc. en ligne)* :

getDirectory(string) or getDir(string)
Displays a "choose directory" dialog and returns the selected directory, or returns the path to a specified directory, such as "plugins", "home", etc. The returned path ends with the file separator ("/"). Returns an empty string if the specified directory is not found or aborts the macro if the user cancels the "choose directory" dialog box. For examples, see the GetDirectoryDemo and ListFilesRecursively macros. See also : getFileList and the File functions.

getTitle()
Returns the title of the current image.

Mettre l'exécution en pause

La macro doit permettre à l'utilisateur de dessiner par lui-même les ROIS sur l'image. Une fois lancée, mis à part si l'on place des boites de dialogue, son exécution se poursuit sans interruption. Il faudra donc trouver un moyen de mettre en pause le script afin de redonner la main à l'utilisateur pour la création et le stockage des ROIs, en somme,d'"attendre une entrée utilisateur".

Un raffinement supplémentaire pourrait être de n'exécuter le reste de la macro qu'à la condition d'avoir une ou plusieurs ROIs à analyser. Les ROIs sont stockées dans le `ROI Manager` auquel s'applique une gamme de fonctions permettant de le manipuler et d'en extraire certaines grandeurs caractéristiques, comme le nombre d'éléments qu'il contient. Une boucle conditionnelle indéfinie à priori (voir le Chapitre 1 - Boucle while) pourrait faire l'affaire, pourvu qu'elle soit bien placée et que sa condition soit bien définie...

Attendre une entrée utilisateur *(source: doc. en ligne)* :

waitForUser(string)
Halts the macro and displays string in a dialog box. The macro proceeds when the user clicks "OK". Unlike showMessage, the dialog box is not modal, so the user can, for example, create a selection or adjust the threshold while the dialog is open. To display a multi-line message, add newline characters ("\n") to string. This function is based on Michael Schmid's Wait_For_User plugin. Example : WaitForUserDemo.

waitForUser(title, message)
This is a two argument version of waitForUser, where title is the dialog box title and message is the text displayed in the dialog.

waitForUser
This is a no argument version of waitForUser that displays "Click OK to continue" in the dialog box.

Exercice de fin de chapitre

Vous avez tous les éléments en mains pour modifier la macro existante et la rendre plus robuste en y ajoutant l'initialisation, la gestion des sorties et de l'interaction avec l'utilisateur.

Étape 03 : à vous de jouer !

A faire :

- Pour chaque point suivant, en s'aidant du `Macro Recorder` et/ou de *cette page*, ajouter un commentaire et les instructions relatives aux étapes suivantes :
  1. Initialisation :
     . Fermer toutes les images présentes sauf l'image courante
     . Réinitialiser le ROI Manager
     . Réinitialiser la table de résultats
  2. Répertoire de sortie :
     . Demander à l'utilisateur de pointer vers un répertoire de sortie
     . Stocker le chemin dans une variable
     . Utiliser la variable pour générer les chemins de sauvegarde des deux fichiers ROIs et résultats. Attention : on ne peut pas utiliser un nom générique pour les fichiers de sortie, mieux vaudra les nommer en fonction, par exemple, du nom de l'image courante
  3. Mettre l'exécution en pause :
     . Afficher une boite de dialogue invitant l'utilisateur à dessiner les ROIs et les stocker dans le ROI Manager au moyen de la touche 't' du clavier
     . A l'issue de cette étape (ou de tout autre étape adaptée), vérifier le contenu du `ROI Manager` : s'il est vide, afficher de nouveau le message
     . Sauvegarder le contenu du `Roi Manager` : `More>Save...`
  4. Réaliser l'analyse 2D de l'image et sauvegarder le tableau de résultats :
     . Utiliser la fonction `More>Multi Measure` du `Roi Manager`
     . Sauvegarder le tableau de résultats à l'aide du menu `File>Save as...`
- Tester la macro : la lancer, vérifier son comportement et corriger les éventuels bugs

Chapitre 3: Fonctionnalisation de la macro et mesures 3D

Le mot-clé de la seconde session sera "`function`". Nous allons tirer profit des possibilités de fonctionnalisation qu'offre le code macro. Au-delà du gain en lisibilité du code, la création de fonctions personnalisées permettra une réutilisation simplifiée de pans entiers d'instructions entre macros.

Groupes fonctionnels

Utilité des fonctions

Les groupes fonctionnels sont ce que les chapitres sont à un livre : une manière d'organiser les instructions par thématique. Le corps du code apparaît alors comme une table des matières : elle référence l'ensemble des chapitres et permet un accès direct aux sections d'intérêt.

Pour un langage aussi simple que le langage macro ImageJ, outre leur avantage structurant, les fonctions personnalisées généralistes sont utiles dans le cadre d'une réutilisation de blocs, diminuant le temps nécessaire au développement, pourvu qu'elles aient été bien réfléchies en amont.

Syntaxe des fonctions

Une fonction est déclarée par le mot-clé `function` suivi du nom de la fonction. Ce nom doit être suffisamment explicite. Par convention, on utilise la notation chameau employée en Java (les mots sont collés les uns aux autres, leur première lettre est capitalisée à l'exception de la lettre initiale). Il doit obligatoirement commencé par une lettre mais peut comporter des chiffres.

Le nom est suivi de deux parenthèses, ouvrante et fermante. Entre les parenthèses peuvent être déclarés des **arguments**, facultatifs : une ou plusieurs variables sont alors fournies entre parenthèses. La portée de ces variables est locale : elles ne sont utilisables qu'à l'intérieur de la fonction.

Les **instructions** définissant les opérations réalisées par la fonction sont placées entre **accolades**. Une fonction particulière, facultative, peut également y être présente : le mot-clé `return` permet de renvoyer le contenu d'une unique variable à l'extérieur de la fonction suite à son appel.

L'appel à la fonction se fait simplement en utilisant son nom et fournissant les arguments nécessaires. Si la fonction renvoie un résultat, il peut être attribué à une variable pour utilisation ultérieure.

```
//Appel à la fonction
resultat=maFonction(1, "arg2");

//Déclaration de la fonction
function maFonction(arg1, arg2, ...){
   //Instruction 1
   //Instruction 2
   ...
   return valeurDeSortie;
}
```

Différents types de fonctions

Les fonctions disposent de deux lots d'éléments facultatifs : les arguments et le retour. Sur la base de ce dernier, on distinguera deux grands types de fonctions : les **méthodes** qui ne disposent pas de l'élément "`return`" (le retour est `null`) et les **fonctions** qui l'utilisent pour renvoyer une unique variable.

Attention : le fait que le mot-clé ne renvoie le contenu que d'une variable ne signifie pas pour autant qu'on ne puisse renvoyer qu'une valeur. On peut, par exemple utiliser une variable de type tableau pour renvoyer un ensemble d'éléments.

Ma première fonction

Le code de l'**étape 03** génère deux sorties : le fichiers de ROIs et le tableau de résultats. Tous deux sont nommés sur la base du nom de l'image. Une image dont le titre est "16HBE SERCA NT_0001.tif" donnera lieu aux deux fichiers "16HBE SERCA NT_0001.tif_Results.csv" et "16HBE SERCA NT_0001.tif_RoiSet.zip". On constate que le nom de l'image est utilisé en entier, y compris l'extension.

Pour ce premier exemple, nous allons définir la fonction `removeExtension` permettant de raccourcir le nom utilisé pour les sorties en supprimant son extension. Avant de définir la fonction, il est nécessaire d'identifier ses paramètres et sorties. La fonction prendra en entrée le nom de l'image et renverra une chaîne de caractères.

Reste à identifier les instructions ImageJ permettant de manipuler une chaîne de caractères.

Manipuler une chaîne *(source: doc. en ligne)* :

endsWith(string, suffix)
Returns true (1) if string ends with suffix.

indexOf(string, substring)
Returns the index within string of the first occurrence of substring.

indexOf(string, substring, fromIndex)
Returns the index within string of the first occurrence of substring, with the search starting at fromIndex.

lastIndexOf(string, substring)
Returns the index within string of the rightmost occurrence of substring.

matches(string, regex)
Returns true if string matches the specified regular expression. Can be replaced with string.matches(regex) in ImageJ 1.52t or later.

startsWith(string, prefix)
Returns true (1) if string starts with prefix.

substring(string, index1, index2)
Returns a new string that is a substring of string. The substring begins at index1 and extends to the character at index2 - 1. Can be replaced with str.substring(i1,i2) in ImageJ 1.52t and later.

substring(string, index)
Returns a substring of string that begins at index and extends to the end of string. Can be replaced with str.substring(i) in ImageJ 1.52t and later.

replace(string, old, new)
Returns a string that results from replacing all occurrences of old in string with new, where old is a single character string. If old is longer than one character, each substring of string that matches the regular expression old is replaced with new. When doing a simple string replacement, and old contains regular expression metacharacters ('.', '[', ']', '^', '$', etc.), you must escape them with a "\\". For example, to replace "[xx]" with "yy", use string=replace(string,"\\[xx\\]","yy"). Can be replaced with string.replace(old,new) in ImageJ 1.52t or later.

Plusieurs stratégies sont possibles : on pourra tenter d'identifier la dernière instance du point et créer une sous-chaîne. Le problème de cette méthode est que certains fichiers pourraient ne pas disposer d'une extension tout en utilisant le point comme séparateur de champs. On optera plutôt pour un remplacement de chaîne. Pour que la création d'une fonction ait un sens, on gérera les deux cas, l'extension pouvant être en majuscule, ou en minuscules.

Une implémentation possible et un exemple d'utilisation sont proposés dans le cartouche suivant.

```
testTxt="16HBE SERCA NT_0001.tif";
withoutExt=removeExtension(testTxt);

print("Avant : "+testTxt);
print("Après : "+withoutExt);

function removeExtension(input){
   output=replace(input, ".tif", "");
   output=replace(output, ".TIF", "");

   return output;
   }
```

Fonctionnalisation de la macro

Du point de vue de la macro, la fonctionnalisation du code démarrera par une relecture des instructions et l'identification de blocs concourant à un même but. A ces fins, les commentaires sont des éléments qui peuvent aider. On pourra identifier quatre étapes, chacune pouvant être encapsulée en une fonction :

- **Fonction "init" :** L'initialisation
- **Fonction "getROIs" :** L'interaction utilisateur pour la définition des ROIs et leur enregistrement
- **Fonction "analyze2D" :** L'analyse 2D et la sauvegarde des résultats
- **Fonction "analyze3D" :** L'analyse 3D et la sauvegarde des résultats (non encore implémenté)

Afin de préparer les différentes fonctions, il sera nécessaire d'identifier les paramètres nécessaires à son fonctionnement et la sortie ou retour que chacune devra fournir :

Fonction	Paramètre(s)	Sortie
init	Aucun	Aucune
getRois	Répertoire et image	Aucune
analyze2D	Répertoire et image	Aucune
analyze3D	Répertoire et image	Aucune

Étape 04 : à vous de jouer !

A faire :
- Utiliser la fonction précédente pour modifier le code de la macro créée à l'étape 03
- Adapter le code de manière à créer les 3 fonctions suivantes :
  - . Fonction "init"
  - . Fonction "getROIs"
  - . Fonction "analyze2D"

Implémentation de l'analyse 3D

Réaliser les mesures en 3D

L'une des caractéristiques des mesures natives proposées par ImageJ est leur limite à la 2D. Le logiciel de base peut être complémenté par des modules programmés en Java : les plugins. Dans le cadre de ce cours, nous n'utiliserons pas les plugins. Il faudra donc trouver des méthodes pour contourner ces limitations.

Une première stratégie pourrait consister à rapatrier les données 2D et les combiner avant de les renvoyer vers le tableau de résultats. La deuxième stratégie consiste à utiliser certaines fonctions disponibles sur les piles d'images.

Manipuler des images 3D *(source: doc. en ligne)* :

Stack (hyperstack) Functions (extrait)

These functions allow you to get and set the position (channel, slice and frame) of a hyperstack (a 4D or 5D stack). The HyperStackDemo demonstrates how to create a hyperstack and how to work with it using these functions

Stack.getDimensions(width, height, channels, slices, frames) Returns the dimensions of the current image.

Stack.setChannel(n) - Displays channel n.

Stack.setSlice(n) - Displays slice n.

Stack.setPosition(channel, slice, frame) - Displays the specified channel, slice and frame.

C'est cette dernière option que nous allons utiliser. Attention : notre image de départ comporte plusieurs canaux. En préalable à toute mesure, il faudra veiller à obtenir des canaux indépendants au moyen du menu `Image>Colors>Split channels` : on obtient 3 images, nommées d'après le titre de l'image originale, et portant C1-, C2- ou C3 pour préfixe. La génération des noms d'images étant standardisée il sera simple d'activer l'image à analyser au besoin.

L'analyse des 3 canaux, pour toutes les régions nécessitera que l'on puisse répéter la mesure : il faudra utiliser une structure répétitive, comme celles décrites au "Chapitre 1 - Structure de choix", et bâtir le nom de l'image à analyser par concaténation.

Créer un tableau de résultats

Les instructions utilisées pour réaliser les mesures renvoient dans des variables les résultats : contrairement à l'analyse 2D, le tableau de résultats n'est pas généré automatiquement. Il faudra donc construire le tableau à partir des instructions macro disponibles.

Manipuler un tableau de résultats *(source: doc. en ligne)* :

Table Functions

These functions, added in ImageJ 1.52a, work with tables. They operate on the current (frontmost) table or, with most of the functions, an optional title argument can be provided (must be last argument). Examples : Sine/Cosine Tables, Rearrange Table and Access Tables.

Table.size - The number of rows in the current table.

Table.set(columnName, rowIndex, value) - Assigns a numeric or string value to the cell at the specified column and row.

Table.update - Updates the window displaying the current table.

Table.save(filePath) - Saves a table.

Table.open(filePath) - Opens a table.

nResults

Returns the current measurement counter value. The parentheses "()" are optional. See also : getValue("results.count").

setResult("Column", row, value)

Adds an entry to the ImageJ results table or modifies an existing entry. The first argument specifies a column in the table. If the specified column does not exist, it is added. The second argument specifies the row, where 0<=row<=nResults. (nResults is a predefined variable.) A row is added to the table if row=nResults. The third argument is the value to be added or modified. With v1.47o or later, it can be a string. Call setResult("Label", row, string) to set the row label. Call updateResults() to display the updated table in the "Results" window. For examples, see the SineCosineTable and ConvexitySolidarity macros.

updateResults()

Call this function to update the "Results" window after the results table has been modified by calls to the setResult() function.

Toute la difficulté de l'utilisation des tables de résultats réside dans l'adressage des lignes du tableau. La variable `nResults` stocke en permanence le nombre de lignes du tableau, numérotées de 0 à nResults-1.

Pour ajouter une ligne au tableau, il suffit de pousser un résultat sur la ligne nResults, soit une ligne qui n'existe pas encore. Autre point important, une fois la ligne créée, si l'on souhaite y ajouter de nouveaux résultats, il faudra adresser la ligne nResuts-1.

Quant aux colonnes, pas de soucis pour les créer : il suffit d'adresser les résultats en nommant la colonne à créer.

Exercice de fin de chapitre

Vous avez à présent tous les éléments en mains pour implémenter l'analyse 3D au sein de la macro existante.

Étape 05 : à vous de jouer !

A faire :

- Créer une nouvelle macro permettant de réaliser l'analyse 3D
- Identifier les étapes à réaliser et les instructions à utiliser
  1. Sélectionner l'image d'origine (le stack)
  2. Stocker les dimensions du stack
  3. Séparer les canaux
  4. Pour chaque canal :
     . Sélectionner l'image du canal
     * Pour chaque ROI :
     + Activer la n-ième ROI
     + Réaliser les mesures
     + Pousser les mesures et le numéro du canal sur une même ligne du tableau
  5. Sauvegarder le tableau de résultats
- Intégrer la fonction `analyze3D` à la macro principale
- Remplacer l'appel à la fonction `analyze2D` par un appel à la fonction `analyze3D`
- Tester la nouvelle macro

Chapitre 4: Amélioration de l'expérience utilisateur

Pour cette dernière session, nous allons améliorer l'expérience utilisateur. Nous avons implémenté deux types d'analyses, 2D et 3D mais n'avons pas la possibilité de sélectionner simplement laquelle réaliser : une boite de dialogue serait la bienvenue pour basculer de l'une à l'autre. Lors d'analyses en série, il peut être fastidieux d'avoir à préciser de nouveau certains paramètres : nous allons voir comment stocker et rappeler les préférences utilisateur.

Interface utilisateur : GUI

Créer une interface

La création d'interface (**Graphical User Interface** ou GUI en anglais), quoique rudimentaire en langage macro, est relativement simple et puissante. Certaines fonctionnalités ont été récemment implémentées sous ImageJ pour augmenter les capacités du logiciel. Notamment, une nouvelle instruction permet d'afficher une boite de dialogue afin que l'utilisateur choisisse un chemin : c'est la version intégrée du `getDirectory` que nous avons utilisé et qu'il faudra remplacer. **Attention :** pour pouvoir utiliser les fonctions les plus récentes des boites de dialogue, il est impératif de procéder à la mise à jour du coeur ImageJ utilisé par Fiji. Pour cela, aller dans le menu `Help>Update ImageJ` puis redémarrer le logiciel.

Les paramètres à collecter sont les suivants :

- **Le mode d'analyse, 2D ou 3D** : l'affichage de boutons radio semble tout indiqué
- **Le mode de projection 3D** : compte-tenu du nombre de modes possibles, une liste déroulante sera appropriée
- **Le répertoire de sortie** : l'intégration d'une fenêtre de choix du répertoire centralisera l'entrée des paramètres dans une unique fenêtre

Une fois la boite de dialogue affichée, il faudra rapatrier l'ensemble des entrées utilisateur. Les éléments de la GUI ne sont pas stockés dans des variables indépendantes : on ne les adresse pas directement. Le mécanisme de collection se fait dans l'ordre où les éléments ont été ajoutés à l'affichage, au moyen de getters (fonctions permettant d'obtenir un résultat) spécifiques.

Créer une GUI *(source: doc. en ligne)* :

Dialog.create("Title")
Creates a modal dialog box with the specified title, or use Dialog.createNonBlocking("Title") to create a non-modal dialog. Call Dialog.addString(), Dialog.addNumber(), etc. to add components to the dialog. Call Dialog.show() to display the dialog and Dialog.getString(), Dialog.getNumber(), etc. to retrieve the values entered by the user. Refer to the DialogDemo macro for an example.
Dialog.addRadioButtonGroup(label, items, rows, columns, default) - Adds a group of radio buttons to the dialog, where 'label' is the group label, 'items' is an array containing the button labels, 'rows' and 'columns' specify the grid size, and 'default' is the label of the default button. (example).
Dialog.addChoice(label, items) - Adds a popup menu to the dialog, where items is a string array containing the menu items.
Dialog.addChoice(label, items, default) - Adds a popup menu, where items is a string array containing the choices and default is the default choice.
Dialog.addDirectory(label, defaultPath) - Adds a directory field and "Browse" button. The field width is determined by the length of 'defaultPath', with a minimum of 25 columns. Use Dialog getString to retrieve the directory path. Requires 1.53d.
Dialog.show() - Displays the dialog and waits until the user clicks "OK" or "Cancel". The macro terminates if the user clicks "Cancel".
Dialog.getString() - Returns a string containing the contents of the next text, directory or file field.
Dialog.getChoice() - Returns the selected item (a string) from the next popup menu.
Dialog.getRadioButton() - Returns the selected item (a string) from the next radio button group.

Bien évidemment, afin de structurer au mieux le code, l'idéal serait de créer une **fonction GUI** en charge de l'affichage et du rapatriement des paramètres.

Obtenir les paramètres : portée des variables

La fonction GUI dispose d'un certain nombre de variables pour stocker les différentes paramètres. Comme indiqué au "Chapitre 1 - Portée des variables", les variables à l'intérieur des fonctions sont locales et ne sont pas accessibles depuis l'extérieur.

L'utilisation de variables globales est généralement déconseillé. En effet, si l'on veut réutiliser un bloc fonctionnel, un simple copier/coller entre macros devrait suffire. Si une fonction utilise des variables globales, en

plus du bloc fonctionnel, il faudra penser à créer les variables globales qui s'y rapportent, ce qui peut compliquer la tâche. Dans le cas de la fonction GUI, non seulement elle est spécifique de cette macro, mais en plus, elle comporte de nombreuses sorties qu'il ne sera pas pratique de stocker, par exemple, dans une unique sortie de type tableau. Nous allons donc employer des variables globales.

Étape 06 : à vous de jouer !

A faire :

- Créer une nouvelle macro permettant de générer l'interface graphique
- Identifier les étapes à réaliser et les instructions à utiliser
  1. Créer une variable globale stockant le chemin de sortie
  2. Créer une variable globale stockant le type d'analyse à réaliser (2D ou 3D)
  3. Créer une variable globale de type tableau (voir le Chapitre 1 - Types de variables) contenant les différents modes de projection 3D
  4. Créer une variable globale stockant le mode de projection 3D retenu
  5. Créer une interface graphique
  6. Ajouter le champ de sélection du chemin de sortie
  7. Ajouter le champ de sélection du type d'analyse à réaliser
  8. Ajouter le champ de sélection du mode de projection
  9. Afficher la boite de dialogue
  10. Stocker dans les variables appropriées les différentes entrées utilisateur
  11. Encapsuler l'ensemble des instructions sous la forme d'une fonction `GUI`
- Intégrer la fonction `GUI` à la macro principale
- Créer l'appel à la fonction `GUI`
- En utilisant la structure de choix (voir Chapitre 1 - Structure de choix), adapter le comportement de la macro en fonction du type d'analyse souhaité
- Tester la nouvelle macro
- Raffinement possible : on peut envisager de différencier, sur la base du nom de la sortie, les résultats de l'analyse 2D vs 3D

Sauvegarder les préférences

Le problème actuel de notre macro est qu'à chaque lancement, les paramètres sont réinitialisés. Pour le traitement d'une série d'images, l'utilisateur pourrait apprécier que les paramètres soient stockés à l'acquittement de l'interface puis rappelés à l'exécution suivante.

Le stockage de préférences est disponible lorsque l'on crée une extension Java d'ImageJ sous la forme d'un plugin : il est disponible au travers de la classe `ij.Prefs` via les fonctions `set` et `get`. Le langage macro n'expose pas directement cette classe, mais permet néanmoins un appel au travers de la fonction `call`.

Travailler avec les préférences (fonction non documentée) :

call("ij.Prefs.set", "macroPoitiers.param1", 1)

Crée une préférence dans le fichier `IJ_prefs.txt` (ce fichier est stocké dans ~/Library/Preferences/ sous Mac OS, ~/.imagej/ sous Linux ou Windows, ~ étant le répertoire utilisateur). Le second argument désigne la clé sous laquelle stockée la préférence, la clé devant obligatoirement être sous la forme "préfixe.suffixe".

maPref = call("ij.Prefs.get", "macroPoitiers.param1", "0")

Rappelle une préférence et la stocke dans la variable. Le second paramètre désigne la clé dont la valeur associée doit être rappelée. Le troisième argument est une valeur par défaut, renvoyée lorsque la clé n'est pas trouvée.

Exercice de fin de chapitre

Nous allons terminer cette session en simplifiant la tâche de l'utilisateur : ses préférences seront stockées au cours de l'exécution de la macro et rappelées à chaque appel.

Étape 07 : à vous de jouer !

A faire :

- Au moyen des instructions présentées en encart, modifier la fonction `GUI` de sorte à ce qu'elle stocke les préférences utilisateur
- Modifier l'initialisation des variables globales afin qu'elle puisse lire, si elles existent, les préférences utilisateur
- Tester la nouvelle macro