ImageJ Macro Programming

January – February 2020

Trainers: Marcel Boeglin, Erwan Grandgirard, Elvire Guiot & Bertrand Vernay
Imaging center – IGBMC
groupe-mic-photon@igbmc.fr

Array (sequence of numbers)

Array

```
listOfNumbers = newArray(1, 2, 3, 4, 5); //create a list of numerical elements
```

In an array with n elements, each element is stored in a defined **position starting a index i=0 and last position i=n-1**

```
print(listOfNumbers);          //throw an error!
print(listOfNumbers[0]);       //print element at position 0 of the array -> 1
print(listOfNumbers[1]);       //print  element at 1 of the array -> 2
print(listOfNumbers[2]);       //print  element at 2 of the array -> 3
print(listOfNumbers[3]);       //print  element at 3 of the array -> 4
print(listOfNumbers[4]);       //print  element at 4 of the array -> 5
print(listOfNumbers[5]);       //no element at position -> throw an error!
```

Array (sequence of strings)

Array

```
igbmcMicro = newArray("SP5", "SP8", "DLS");          //create a list of strings
```

In an array with n elements, each element is stored in a defined **position starting a index i=0 and last position i=n-1**

```
print(igbmcMicro);          //throw an error!
print(igbmcMicro[0]);       //print position 1 of the array -> SP5
print(igbmcMicro[1]);       //print position 2 of the array -> SP8
print(igbmcMicro[2]);       //print position 3 of the array -> DLS
print(igbmcMicro[3]);       //throw an error!
```

Array: example

Array

```
micPhotonNames = newArray(5);             //create a list of for elements
micPhotonNames[0] = "Grangirard";         //assign the string "Grangirard" to element index=0
micPhotonNames[1] = "Boeglin";            //assign the string "Boeglin" to element index=1
micPhotonNames[2] = "Guiot";              //assign the string "Guiot" to element index=2
micPhotonNames[3] = "Vernay";             //assign the string "Vernay" to element index=3
micPhotonNames[4] = “Lutz";               //assign the string “Lutz" to element index=4
print(micPhotonNames[0]);
print(micPhotonNames[1]);
print(micPhotonNames[2]);
print(micPhotonNames[3]);
print(micPhotonNames[4]);
print(micPhotonNames[5]);
```

Array functions

Displays the content of an array

-> **Array.show(array)**

Prints the array on a single line

-> **Array.print(array)**

Create a new array by joining 2 arrays or more or values

-> **Array.concat(array1,array2)**

Return the length of an array

-> ***array*.length** *array* is the name of your array

https://imagej.nih.gov/ij/developer/macro/functions.html#A

Exercice #1

1 - Create an array containing the name of 5 fruits

2 - Print the length of your current list of fruits

3 - Print the name of the 3rd fruit in your list

4 - Add another fruit at the start of your list of fruits

5 – Print the length of your current list of fruits

6 - Add another fruit at the end of your list of fruits

7 - Get the length of your current list of fruits

Exercice #2

1 - Create an array containing all the files of the "Data\Images" folder

2 - Get the length of this array

3 - Print all the files name contained in that folder

Tips: use the built-in functions ***getDirectory***, ***getFileList*** and ***Array.show()***

Reading and Writing Files

Opening files: built-in functions

open(*path*) opens and displays tiff, dicom, fits, pgm, jpeg, bmp, gif, lut, roi, or text file specified by ***path***

open() displays a file open dialog box if ***path*** is an empty string or if there is no argument

File.openDialog(*title*) displays a file open dialog and returns the path to the file chosen by the user

Saving files: built-in functions

save(*path*)
Saves an image, lookup table, selection or text window to the specified file path. The path must end in ".tif", ".jpg", ".gif", ".zip", ".raw", ".avi", ".bmp", ".fits", ".png", ".pgm", ".lut", ".roi" or ".txt".

saveAs(*format*, *path*)
Saves the active image, lookup table, selection, measurement results, selection XY coordinates or text window to the specified file path.
The *format* argument must be "tiff", "jpeg", "gif", "zip", "raw", "avi", "bmp", "fits", "png", "pgm", "text image", "lut", "selection", "results", "xy Coordinates" or "text". Use **saveAs(*format*)** to have a "Save As" dialog displayed.

Exercice #3

1 - Create an array containing all the files of the "Data\Images" folder

2 - Print the length of this array

3 - Print all the files name contained in that folder

4 – Open all the images files of that folder

Tip #1: use the built-in functions ***getDirectory***, ***getFileList, open()***

Tip #2: the path to each file is ***directory+file name***

File separator

Argument *path* is a string

Windows full path

something like this open("C:\Users\Bertrand\Desktop\myFolder\myfile.tif“)

Linux full path

something like this open("/home/bertrand/desktop/myfolder/myfile.tif“)

Mac OSX full path

something like this open("/Users/bertrand/Desktop/myFolder/myfile.tif“)

The **File.separator** function returns the file name separator character ("/" or "\").

All operating systems path

folder+File.separator+folder2+File.separator+folder3+File.separator+myFile.tif

File functions

Make a directory

-> File.makeDirectory(path)

Deletes the specified file or directory

-> File.delete(path)

The name of the last file opened with the extension removed

-> File.nameWithoutExtension

https://imagej.nih.gov/ij/developer/macro/functions.html#F

Conditional Tests

Conditional execution *if*

```
if (condition){                         //the condition is evaluated
        list of statements 1;           //if the condition is true statements 1 are executed
        }
statement2;                             //the statement2 is always executed
```

{ } are used to group together multiple statement as one

Example

```
        if(1 == 1){
                print("It is true!");
                }
        print("end");
```

Conditional execution, *if* … *else* statements

```
if (condition) {                        //the condition is evaluated
          statement 1;                  //if the condition is true statement1 is executed
          }
else {
          statement 2;                  //if the condition is false statement2 is executed
          }
```

Conditional execution, *if … else* statements

```
newImage("My image", "8-bit black", 640, 480, 1);
x = getNumber("Enter value to add", 5);
if (x == 0) {
        showMessage("Adding zero would have no effect");
        }
else {
        print("Adding " + x + " to all pixel values. ");
        run("Add...", "value=" + x);
}
```

if, if … else   curly brackets or no curly brackets

```
newImage("My image", "8-bit black", 640, 480, 1);
x = getNumber("Enter value to add", 5);
if (x == 0)                          //{} are not required when using single statement
        showMessage("Adding zero would have no effect");
else                                 //{} are not required when using single statement
        run("Add...", "value=" + x);
```

No advisable = may lead to error when modifying the code later

if, if … else: testing for file type

```
path = File.openDialog("Select a file");
if (endsWith(path,".tif")){
        open(path);
        }
else {
        print(path+ " is not a tif file");
        }
```

Exercice #4a

1 - Create an array containing all the files of the "Data\Images" folder

2 - Print the length of this array

3 - Print all the files name contained in that folder

4 - Open the images files of that folder only if they are .gif

Exercice #4b

5 - Open the images files of that folder only if they are .gif

Tip #1: use the built-in functions ***getDirectory***, ***getFileList, open(), endsWith()***

Tip #2: the path to each file is ***directory+file name***

Repetition-with-variation: Looping

Why using a loop

Script to open 4 images from the same folder

```
inputDir = getDirectory("Choose a Directory");
myList = getFileList(inputDir);
open(inputDir + myList[0]);
open(inputDir + myList[1]);
open(inputDir + myList[2]);
open(inputDir + myList[3]);
```

Why using a loop

Script to open n files from the same folder

```
inputDir = getDirectory("Select the folder");
myList = getFileList(inputDir);
for(i = 0; i<myList.length; i++){
        open(inputDir + myList[i]);
}
```

For loop

- A *for* loop can repeat a code block a given number of times

- *for* (*initialisation*; *condition*; *increment*) {

  list of statements

  }

The *initialization* is a statement that runs once at the beginning of the loop.

The *condition* is before each iteration of the loop and the loop terminates when it evaluates to false.

The *increment* is a statement that is executed after each iteration of the loop.

For loops: example #1

```
for (i = 0; i <= 100; i++){
        print(i);
        }
```

Initialisation	run only once at the first loop iteration i = 0
Condition	go through the loop only if i is less than 100
Increment	at the end of the loop iteration add 1 to i value

The initialisation value and the condition define the range of the loop

For loops: example #2

```
//Open an image in Fiji before running the script
k = getNumber("Repeat how many times", 1);
for (i = 1; i <= k; i++){
        print("start of iteration = " + i);
        run("Median...", "radius=2");
        print("end of iteration = " + i);
        print("\n");
}
```

For loops: example #3 using a list

Using an array to store a list of elements

```
//Define a list called myList
myList = newArray(1,2,3,4,5);

//Access the list element
print(myList[0]);       //print the first element of the list
print(myList[4]);       //print the last element of the list
```

For loops: example #3 using a list

```
//Define a list called myList
myList = newArray(1,2,3,4,5);

//Loop through the 5 elements of the array
for (i = 0; i < myList.length; i++){
        print(" array index = " + i + ", value = " + myList[i]);
}
```

For loops: example #4 using a list to open all files in a folder

```
//Select the folder
inputDir = getDirectory("Select the folder");
myListOfFiles = getFileList(inputDir);

//Loop through the 4 elements of the array
for (i = 0; i < myListOfFiles.length; i++){
		print("Processing file = " + myListOfFiles[i]);
		inputPath = inputDir + myListOfFiles[i];
		open(inputPath);
}
```

Exercice #5

1 – Use a for loop to process the 4 images in the « Data\Images » folder

Process>Noise>Add Noise

Tip #1: use the macro recorder to get Process>Noise>Add Noise code

Tip #2: you only need to add a single line of code to the previous code

Exercice #6

1 – Use a for loop to process the 4 images in the « Data\Images » folder

for the gif image -> Process>Noise>Add Noise

for the jpg image -> Median filter of radius 5

Exercice #8

1 – Use a for loop to process the 4 images in the « Data\Images » folder

1.1- for the gif image -> Process>Noise>Add Noise, the user determine how many time the «Add Noise» function is applied

1.2- for the jpg image -> Median filter with a radius determined by user input

Exercice #7

1 – Use a for loop to process the 4 images in the « Data\Images » folder

for the gif image -> Process>Noise>Add Noise

for the jpg image -> Median filter with a radius determined by user input

Loops: *while*

```
 while (condition){                    //test for condition
         list of statements            //executes statement as long condition is true
}

 while (nImages()>0) {
         selectImage(nImages());
         close();
 }
```

While loop preferably used when the number iteration is not known at the beginning

Loops: *do* … *while*

```
do
        statement(s)
while (condition);
```

A *do … while* loop execute the body at least once