ImageJ Macro Programming

January – February 2020

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

Where to find the workshop documentation ?

- USB Keys
- *Online* https://github.com/bvernay/Macro-_Scripting_Workshop

Where to find help after the workshop ?

- ImageJ Website https://imagej.net/
- Fiji Website https://fiji.sc/
- Forum https://forum.image.sc
- Microscopy Facility groupe-mic-photon@igbmc.fr
- Your colleagues

Workshop rules

Ask many questions,

…

And ask more questions

Why writing macros?

- **Automate** repetitive tasks to make your life easier
- **Document** your image processing and analysis complex pipeline
- **Reproducible** data analysis
- **Share** procedures with colleagues
- **Add tools** to the ImageJ toolbar
- **Add keyboard shortcuts** in ImageJ
- …

ImageJ macro scripting language

PROS

- Scripting is intuitive and easy to learn
- No need to know the general ImageJ Java API
- Multiple macros in one file for complex processing tasks

CONS

- Limited extendibility: only copy and paste, no library. Cannot be used from others programs
- Slow processing speed
- No efficient implementation of real-time user input
- Coupled to GIU -> image have to be opened

Learning Java will open infinite possibilities for processing/analysis

Macro recording & editing

- Easiest way to create a macro: Plugins>Macros>Record…
- Create a new macro from scratch: Plugins>New>Macro
- Open the editor window: File>New>Script… (ctrl+shift+n or press “[”) and copy/paste an existing code
    - Run the macro: ctrl+r
    - Run a single line: ctrl+y
    - Run the selected code only: shift+ctrl+r
- Save the macro as *.ijm or *.txt

Text editor for ImageJ macro programming

- It is recommended to use the Fiji script editor:
  - language specific syntax highlights (Select Language>IJ1 Macro)
  - run the script from the editor

- Other text editors: Notepad++ (Windows), TextWrangler (Mac OSX), gedit (Linux), …

- Do not use a word processor (Microsoft Word, LibreOffice Writer)
  Quotation mark: “Hello” versus "Hello"

Exercice

1- Open the script editor
File>New>Script…

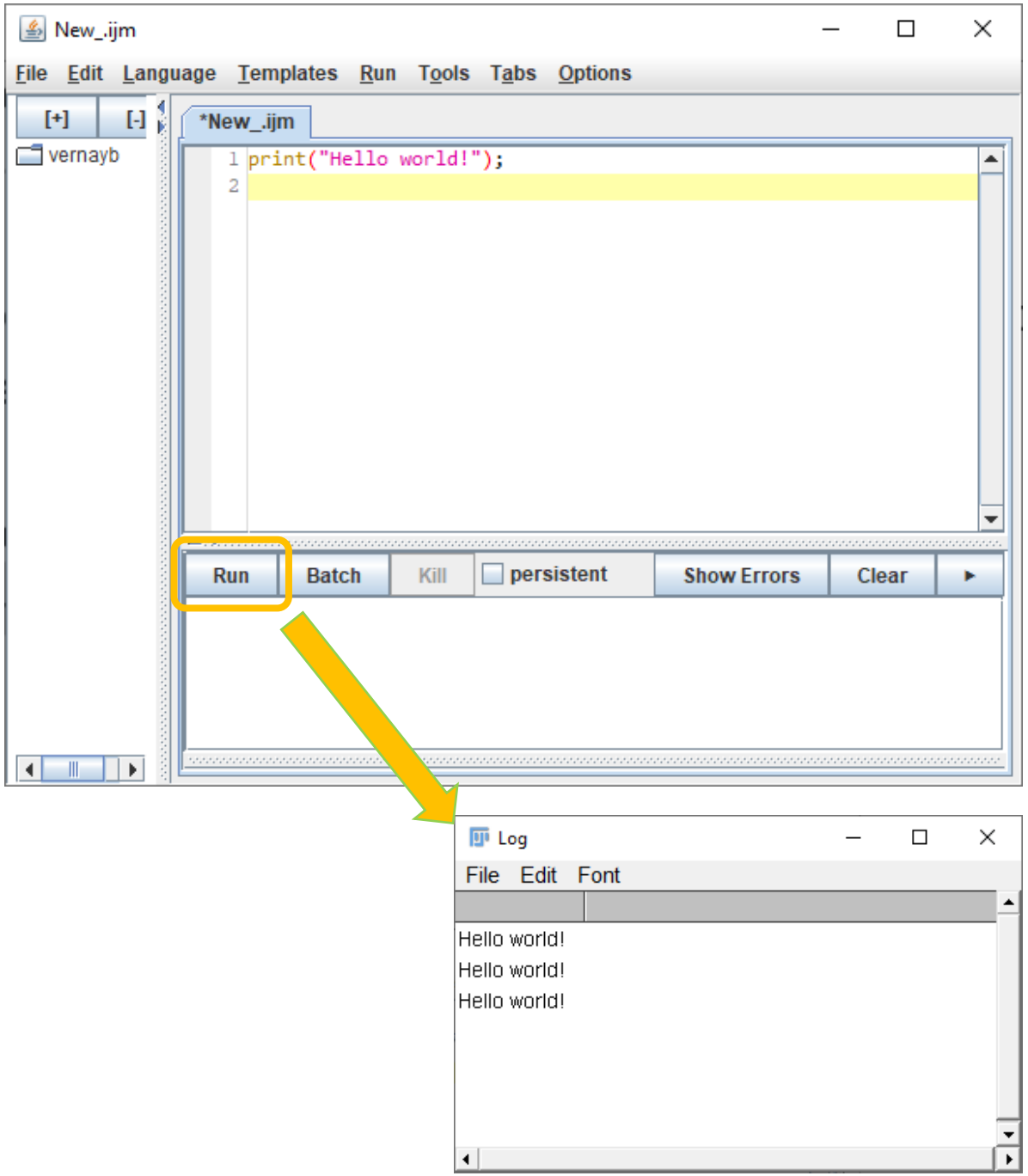
2- Select *Language>IJ1 Macro* for the syntax highlighter and autocomplete

3- In the script window type:

```
print("Hello world!");
```

4- Press Run or crtl+R

Hello world! is displayed in the Log windows each time you run the script



Anatomy of a statement

Literal string passed as the argument of the print function

```
print("Hello World!");
```

Built-in Macro function

Semicolon to separate commands in the macro

the brackets separate the name of the function from the argument(s) passed to the function

Anatomy of a statement

argument of the print function

```
print(argument);
```

Anatomy of a built-in function

print(string)
Outputs a string to the "Log" window. Numeric arguments are automatically converted to strings. The print() function accepts multiple arguments. For example, you can use *print(x,y,width, height)* instead of *print(x+" "+y+" "+width+" "+height)*. If the first argument is a file handle returned by File.open(path), then the second is saved in the refered file (see SaveTextFileDemo).

Numeric expressions are automatically converted to strings using four decimal places, or use the d2s function to specify the decimal places. For example, *print(2/3)* outputs "0.6667" but *print(d2s(2/3,1))* outputs "0.7".

The print() function accepts commands such as *"\\Clear"*, *"\\Update:<text>"* and *"\\Update<n>:<text>"* (for nprint("\\Clear") erases the Log window, *print("\\Update:new text")* replaces the last line with "new text" and *print("\\Update8:new 8th line")* replaces the 8th line with "new 8th line". Refer to the LogWindowTricks macro for an example.

The second argument to print(arg1, arg2) is appended to a text window or table if the first argument is a window title in brackets, for example *print("[My Window]", "Hello, world")*. With text windows, newline characters ("\n") are not automatically appended and text that starts with *"\\Update:"* replaces the entire contents of the window. Refer to the PrintToTextWindow, Clock and ProgressBar macros for examples.

The second argument to print(arg1, arg2) is appended to a table (e.g., ResultsTable) if the first argument is the title of the table in brackets. Use the *Plugins>New* command to open a blank table. Any command that can be sent to the "Log" window (*"\\Clear"*, *"\\Update:<text>"* , etc.) can also be sent to a table. Refer to the SineCosineTable2 and TableTricks macros for examples.

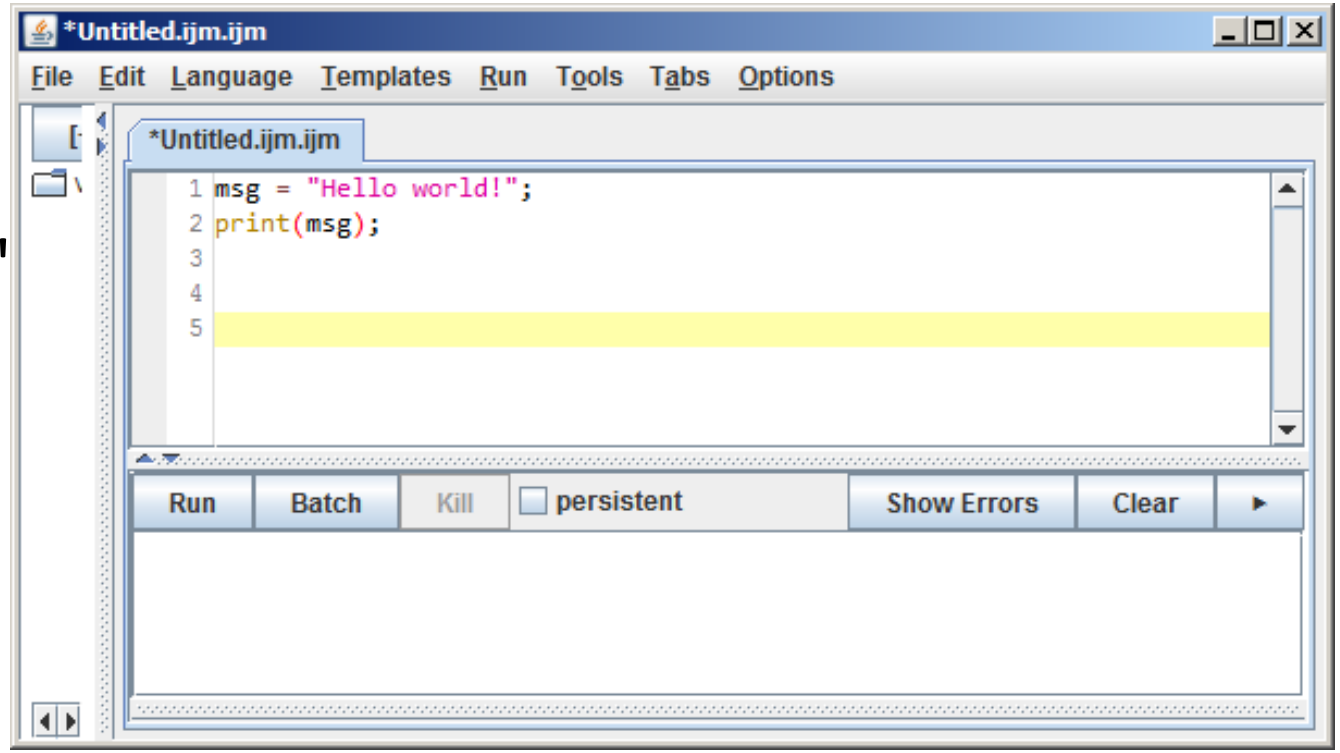
https://imagej.nih.gov/ij/developer/macro/functions.html#P

Exercice

1- Run this code:

```
msg = "Hello world!"
print(msg);
```

2- What happened ?

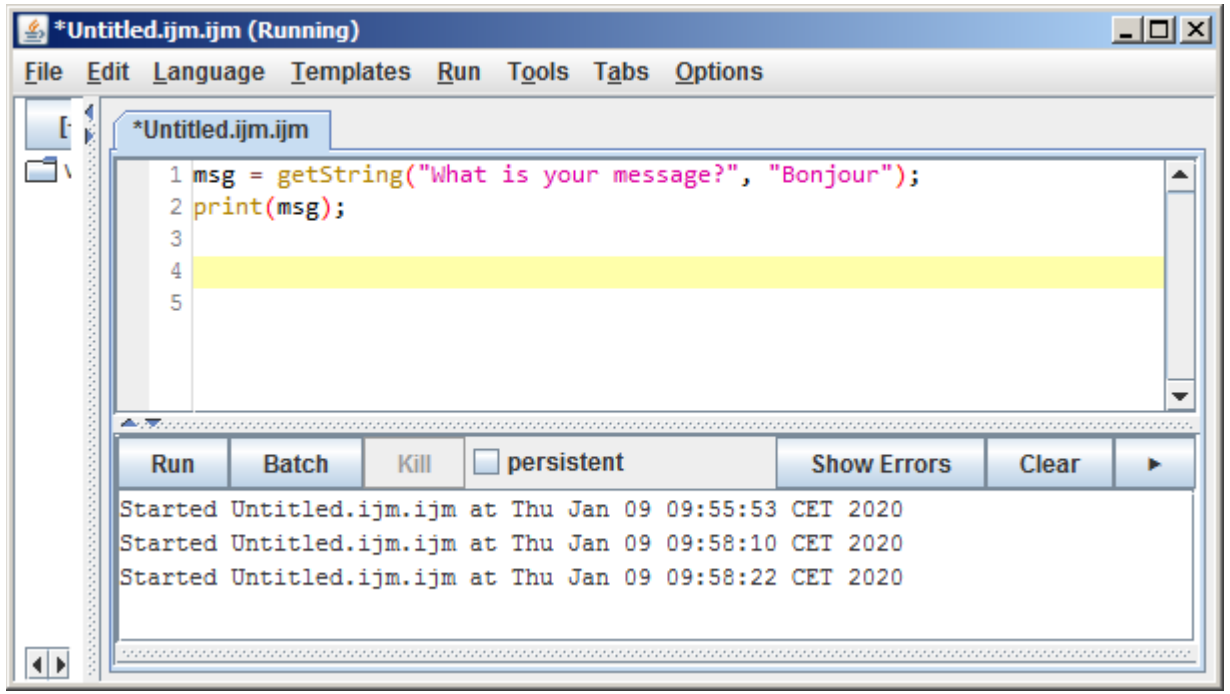


Exercice

1- Run this code:

```
msg = getString("What is your message?",
"Bonjour");
print(msg);
```

2- What happened ?



Literals:
Numbers, Strings and Booleans

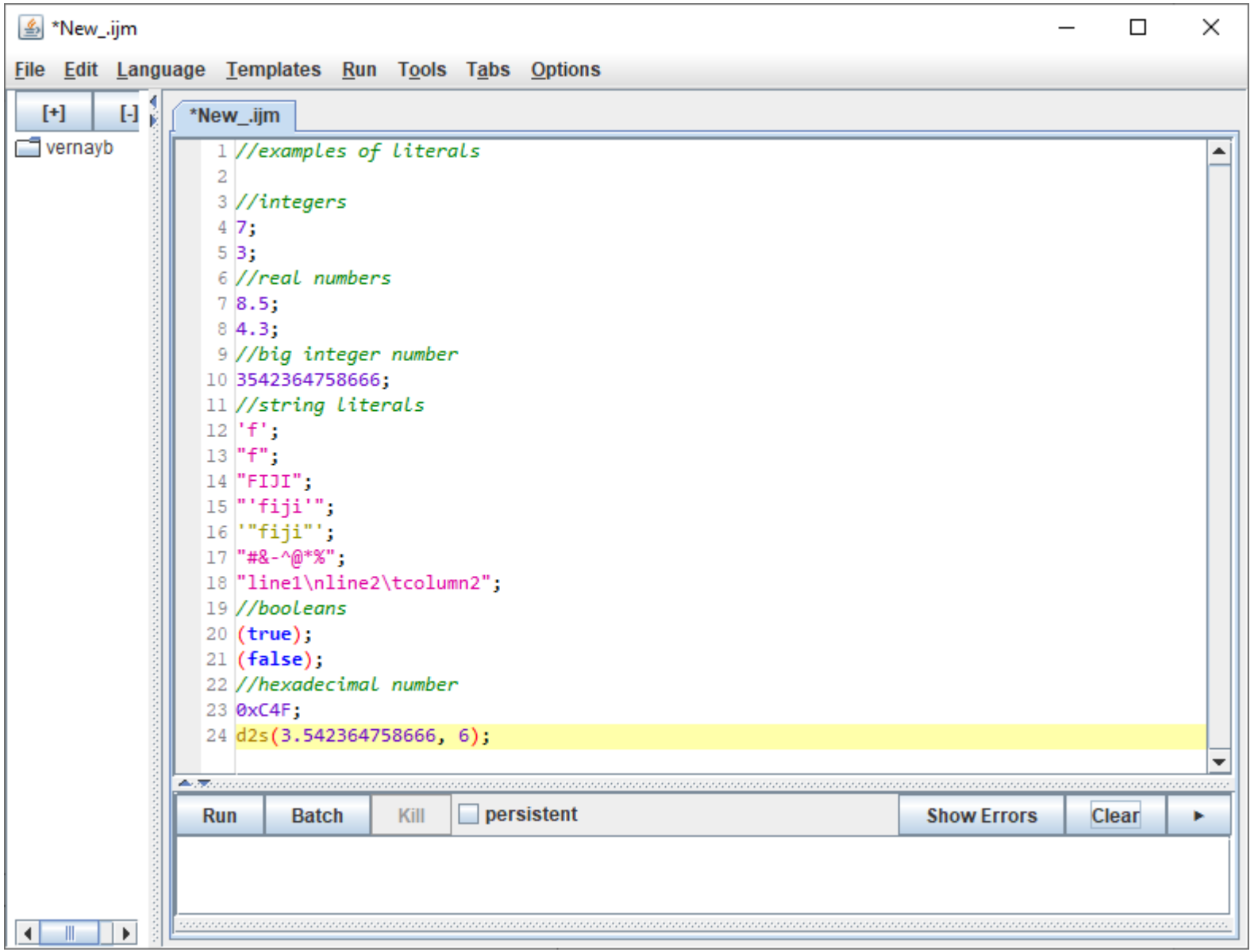
Literals

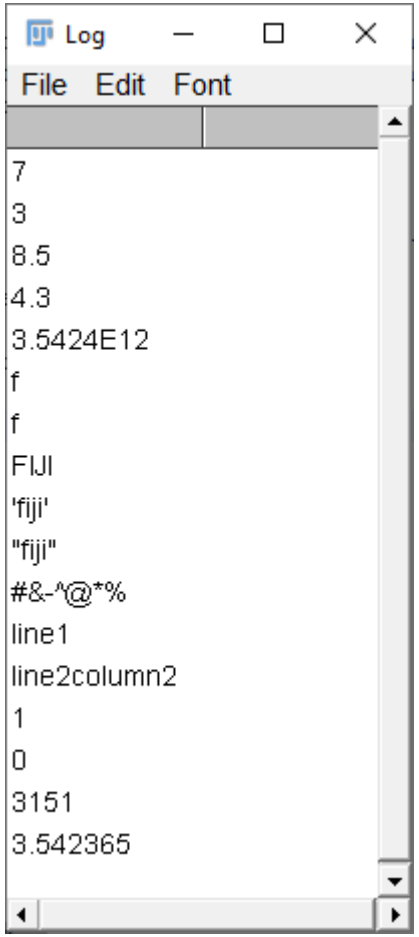
- A value written exactly as it is meant to be interpreted

- 3 types of literals
  - Numbers: ***integer*** or ***real***
  - Text called ***strings***
  - Truth values or ***boolean***

Exercice

```
//examples of literals
//integers
7;
3;
//real numbers
8.5;
4.3;
//big integer number
3542364758666;
//string literals
'f';
"f";
"FIJI";
"'fiji'";
'"fiji"';
"#&-^@*%";
"line1\nline2\tcolumn2";
//booleans
(true);
(false);
//hexadecimal number
0xC4F;
d2s(3.542364758666, 6);
```





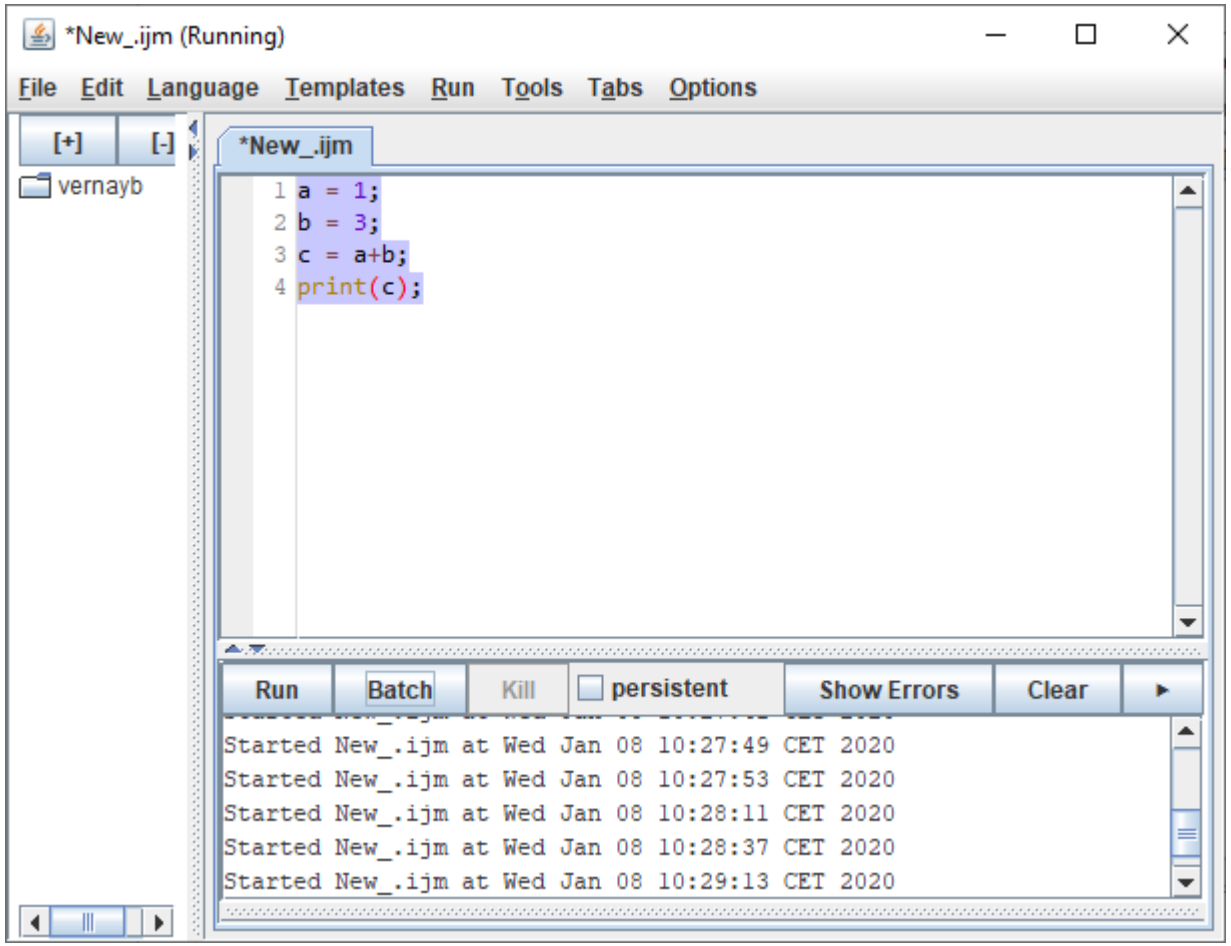
Variables:
Numeric, String and Array

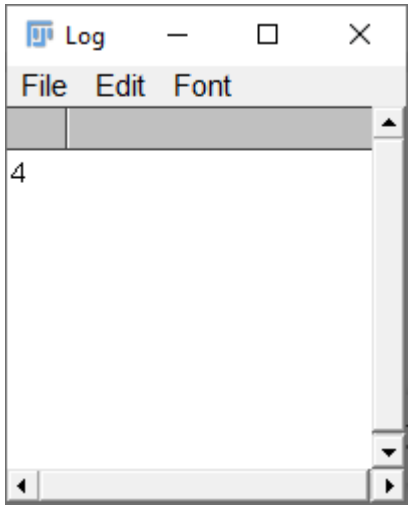
Exercice

1- Run this code:

```
a = 1;
b = 3;
c = a + b;
print(c);
```

2- What happened ?





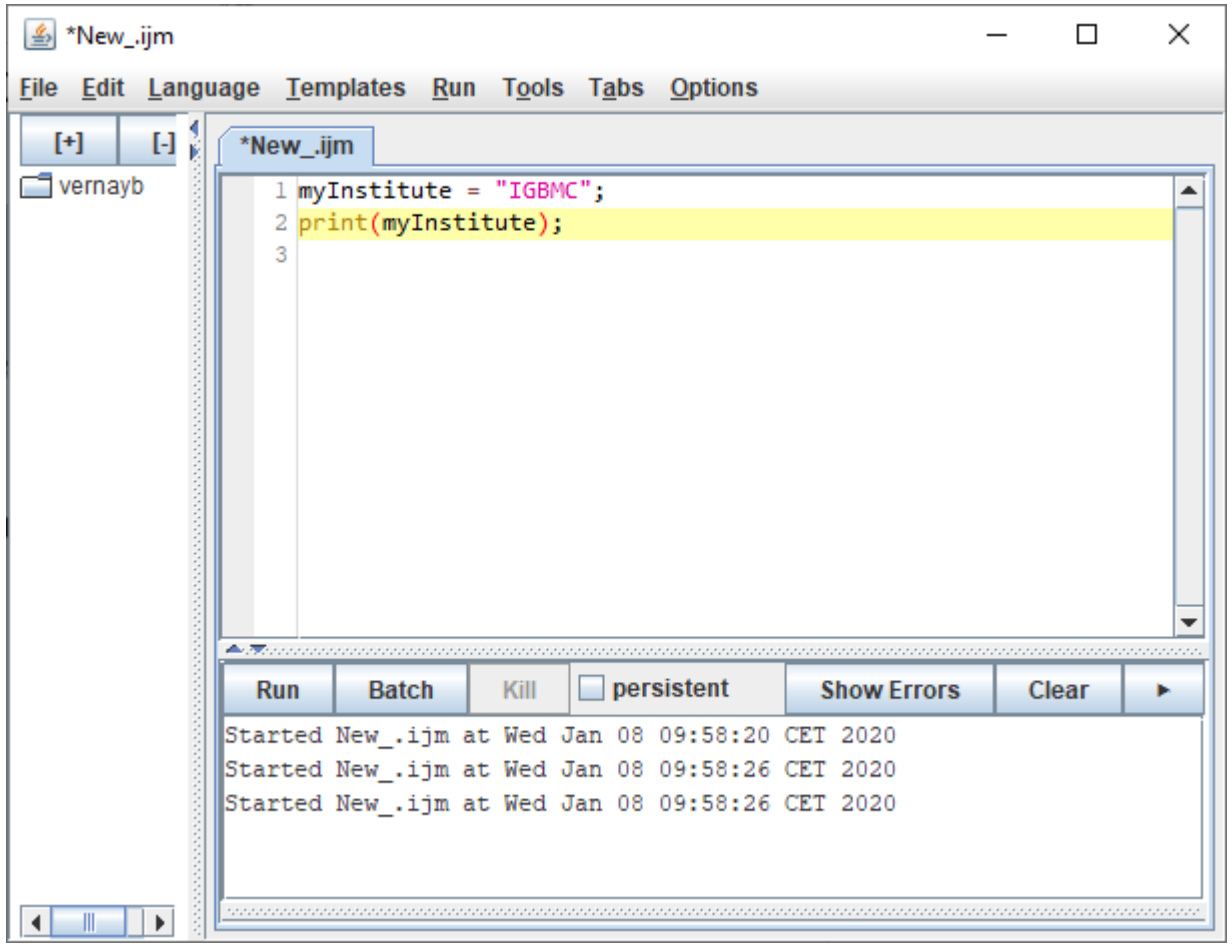
3- Why using this code versus the shorter print(1+3); ?

Exercice

1- Run this code:

```
myInstitute = "IGBMC";
print(myInstitute);
```

2- What happened ?



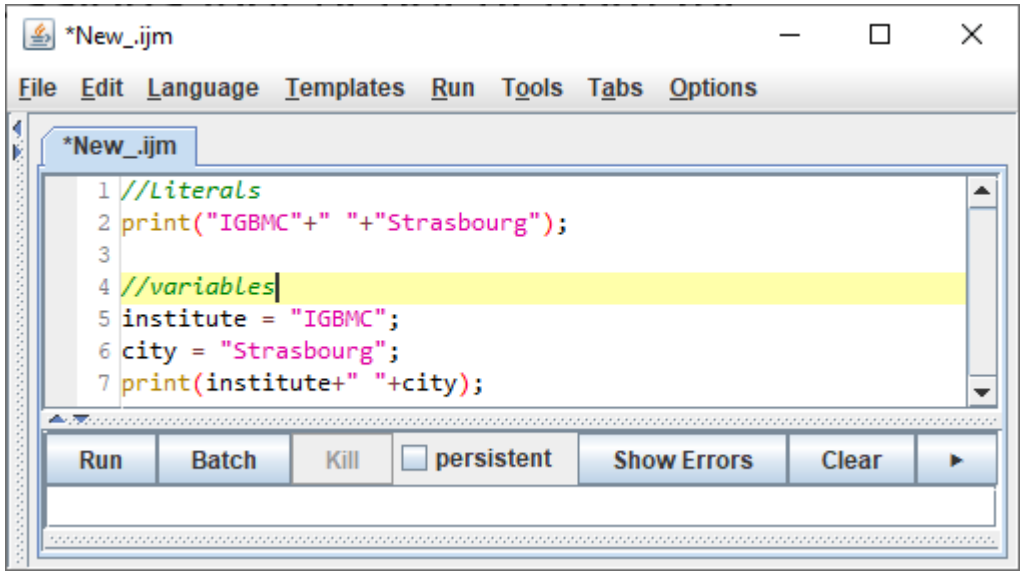
Variables

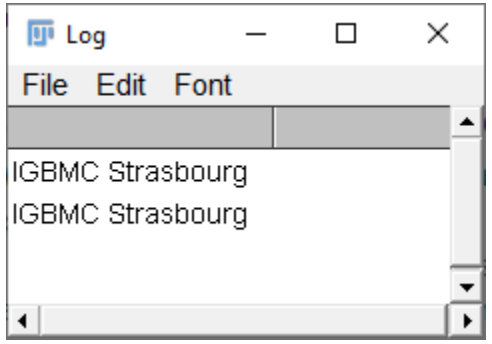
- A variable is placeholder for a changing entity. It has a *name or identifier* (<u>case sensitive</u>) and a *value.*

- Types of variables
  - ***Number***
  - ***String***
  - ***Boolean***
  - List of values or ***Array***

Variables

- Variables can be used in expressions
- Variables allow to generalise and write expressions independent from the values for which they will be evaluated





Variable names

- Variable name must only contain letters, numbers and underscores
- Variable name is case sensitive: **myvar**, **Myvar**, **myVar** are all distinct variables
- Cannot use word already built in the macro language such as **print** or **true**
- Variable names are arbitrary, you can pick whatever you like

- What makes a good variable name?
    - It should give a clue to what the variable refers to:

      title1 versus GreenChannel ?
    - camelCase notation help to read long variable name:

      imagenamegreenchannel versus imageNameGreenChannel

      newstack versus newStack

Assigning a variable

variable name = assigned value ;

Assignment rules:

- Assignment is done from right to left
- The equal sign = assigns a value to the variable (assignment operator)
- Semi-colon ; tells ImageJ that the assignment is done
- String are assigned between two straight quotation marks "
- Array are assigned by using the newArray() function

Assigning a variable

```
//Numeric
numberAttendees= 9;
//String
messageToClass = "Hello Class"; // "Hello Class"  as text
iAmAString = "12345";                  // "12345" as text
//Array
        Method #1
        attendeesNames = newArray("Grandgirard", "Boeglin", "Guiot", "Vernay");

        Method #2
        attendeesNames = newArray(4);
        attendeesNames[0] = " Grandgirard ";
```

Assigning a variable

Numeric

v = 1.23;	assign the value 1.23 to an arbitrary named variable v
print(v);	output (i.e print) the value of v in the log window

String

v = "a string";	assign the text " a string " to an arbitrary named variable v
print(v);	output (i.e print) the value of v in the log window

Assigning a variable

Numeric

v = 1.23;	assign the value 1.23 to an arbitrary named variable v
print(v);	output (i.e print) the value of v in the log window
v = 8;	assign a new value to the variable v
print(v);	print the new value of v

Array (numerical list)

Array

v = 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(v);//throw an error!

print(v[0]); //print element at position 0 of the array v -> 1

print(v[1]); //print element at 1 of the array v -> 2

print(v[2]); //print element at 2 of the array v -> 3

print(v[3]); //print element at 3 of the array v -> 4

print(v[4]); //print element at 4 of the array v -> 5

print(v[5]); //no element at position 5 -> throw an error!

Array (string list)

Array

v = newArray("SP5", "SP8", "Spinning"); //create a list of text 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(v);       //throw an error!
print(v[0]);    //print position 0 of the array v -> SP5
print(v[1]);    //print position 0 of the array v -> SP8
print(v[2]);    //print position 0 of the array v -> Spinning
print(v[3]);    //throw an error!
```

Array (string list)

Array

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

Expressions & Operations

Expressions and Operators

- Values can be combined to expressions with the help of operators

- Values can be written in literal form or as variables

Numerical operators

- Arithmetic operations

	code	results
- negation	(-1);	-1
+ addition	3+7;	10
- substraction	3-7;	-4
* multiplication	3*7;	21
/ division	3/7;	0.4286
	d2s(3/7, 9);	0.428571429
% remainder	3%7	3
	7%3	1

Numerical operators precedence

Arithmetic operations	Precedence
- negation	1
* multiplication	2
/ division	2
% remainder	2
+ addition	3
- substraction	3

Use brackets to change the precedence

2 + 3 * 5;	17
(2 + 3) * 5;	25

Strings concatenation

- String represents text
- Strings can be concatenated with the **+** operator
- One of the most important operation in Macro ImageJ use for building file names and file paths

String + string

"Hello" + " " + "World!" give the string "Hello World!".

String + numeric

"Hello " + 67 give the string "Hello 67", note that the numerical part becomes a string

Concatenate strings examples

```
firstName = "Bertrand";
Name = "Vernay";
fullName = firstName + " " +Name;
print(fullName);

msg = "Welcome to the Macro programming class " + fullName;
print(msg);
```

Self-referencing assignments

- When assigning a variable the right-hand side is evaluated first, and only then the assignment is performed:

```
counter=1;
counter = counter + 1;     //counter = 2
counter++;                 //counter = 3

counter = counter - 1;     //counter =2
counter--;                 //counter =1

counter = counter * 2;     //counter = 2
```

Self-referencing assignments

- When assigning a variable the right-hand side is evaluated first, and only then the assignment is performed:

```
counter+=3;        // add 3
counter-=7;        // subtract 7
counter*=5;        // multiply by 5
counter/=2;        // divide by 2
```

Macro language Comparison operators

For all basic datatypes: numbers, booleans and strings

Precedence 4

==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

Numerical comparison

Example

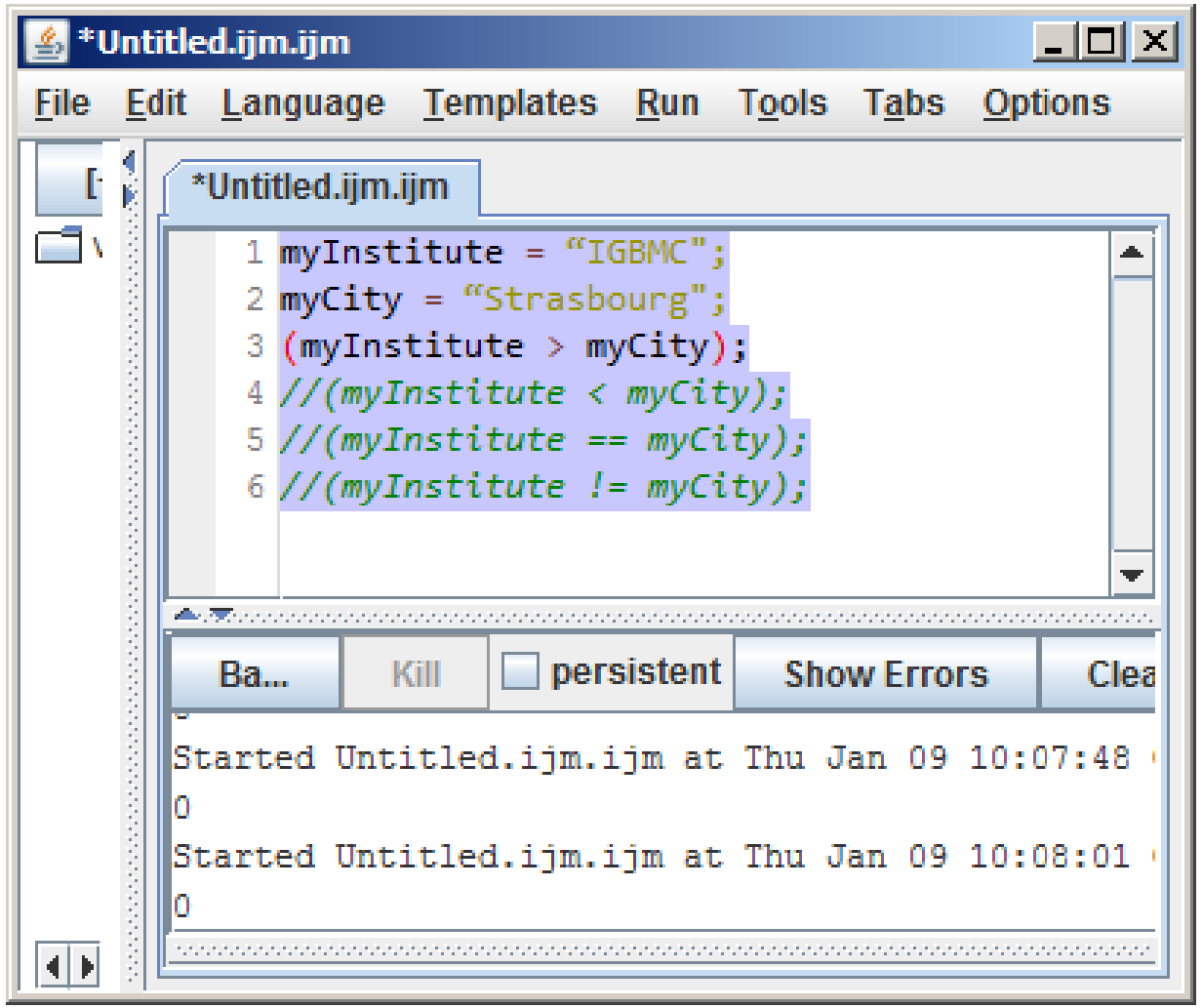
```
x=5<7;
print ("X= "+x);

(3.14159265359 == PI);

d2s(PI, 9);
```

Comparison operators strings

```
myInstitute = "IGBMC";
myCity = "Strasbourg";
(myInstitute > myCity);
//(myInstitute < myCity);
//(myInstitute == myCity);
//(myInstitute != myCity);
```



Strings comparison

Strings comparison is case-insensitive

```
("yes" == "Yes");          // is true and will return 1
print("yes" == "Yes");     // is true and will return 1
 ("yes" == "no");          // is false and will return 0
print("yes" == "no");      // is false and will return 0
```

Comparison operators booleans

- Logical operators:

!	NOT
&&	AND
||	OR

  In ImageJ *true* is 1 and *false* is 0

```
a=5;
b=7;
c=4;
results= ((a==5) && (b==7));          //testing for a=5 AND b=7
print(results);
results= ((a==5) || (c==7));          //testing for a=5 OR c=7
print(results);
```

Commenting the code

- To help debugging the code (commented-out code)
- To explain your code to yourself (after the weekend or 6 months later …) and to your colleagues
- Never loose the documentation with comments being part of the source code
- Make it easier to read old code
- Remember: good code need few explanations

```
// everything after the 2 forward slashes up to the end of the line is a comment
```

or

```
/* multi-line comments
* are enclosed
* in multi-lines blocks
*/
```

Built-in Functions

Functions

- Functions implement the actions you want to execute.
- Functions have *names* and parameters called *arguments* that you can pass to the functions
- ImageJ comes with pre-defined or built-in functions (http://imagej.net/developer/macro/functions.html)
- You can create your own functions: myMagicFunction(arg1, arg2)

Functions

- write(" Hello world ");

The semicolon signifies the end of the statement

The function name is *write*, and has only *one argument* enclosed in ()

- newImage("My image", "8-bit black", 640, 480, 1);

The function name is *newImage*, and there are *5 arguments* separated by commas. The order of the arguments is important, this the way the function knows what each parameter means.

Exercice #1

Write the code to create a new image with the following parameters:

- title = My image
- bit depth = 8
- fill with = black
- width = 512
- height = 256
- slice =1

Tip: use the recorder with File>New<Image…

Exercice #2

Write the code to create a new image with the previous parameters

- define variable for each parameter
- pass the variables as arguments

Exercice #3

Write the code to create a new image with the previous parameters

- define a variable for each parameter
- use user's input to define width and height
- pass the variables as arguments

Tip: getNumber()

Exercice #4

Write the code to create a new image with the previous parameters

- define variable for each parameter
- user input to define each parameter
- pass the variables as arguments

Tips: getNumber(), getString()

Exercice #5

Change the image title

Tips: macro recorder, getString()