Scripting using ImageJ Macro

- a quick 1 hour tutorial -

We use Fiji.

http://fiji.sc

… please download and install.

Kota Miura @ CMCI EMBL

2-1 Why do we write macro?

Aim: Students acquire ImageJ macro programming techniques to ease their work loads with image processing / analysis.

1. Automate Procedures
2. Implement your algorithm
3. Processing using Cluster (fast!)
4. Use from Cell Profiler
5. Limitations
    1. Interactive processing is difficult
    2. Re-Usability

2-2 Basics routine

2-2-1 Hello World! (Fiji)

1. Start-up script editor

   [PlugIns -> Scripting -> Script Editor]

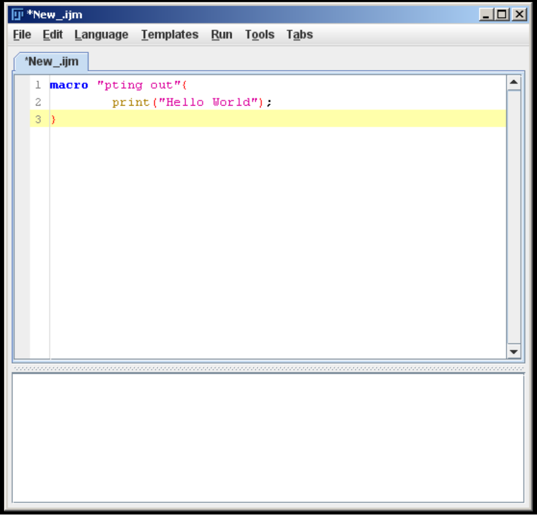
   …then set language

   [Language -> ImageJ macro]

2. And then write as follows (**omit numbers**)

Tabulate!

```
1: macro "print_out" {
2:      print("Hello World!");
3: }
```

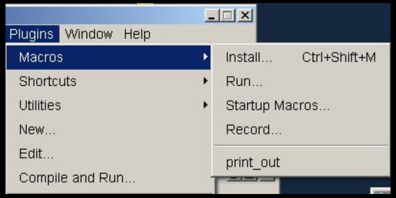


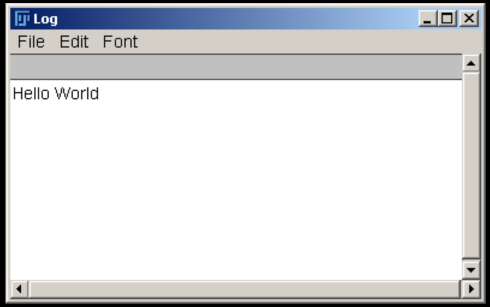
```
macro "pting out"{
        print("Hello World");
}
```

3. Install the macro. In the Editor menu

   [Run -> Install Macro]
   (or command - I)

4. In Fiji menu,
   [Plugins -> Macro -> Print out]





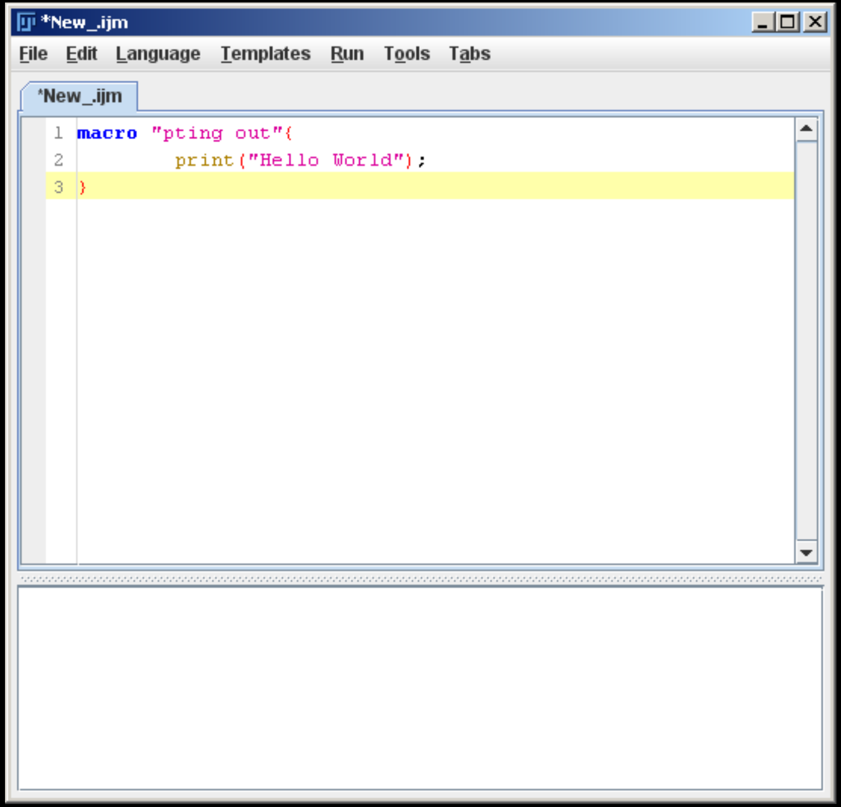
```
Hello World
```

5. Save the macro as "HelloWorld.ijm".

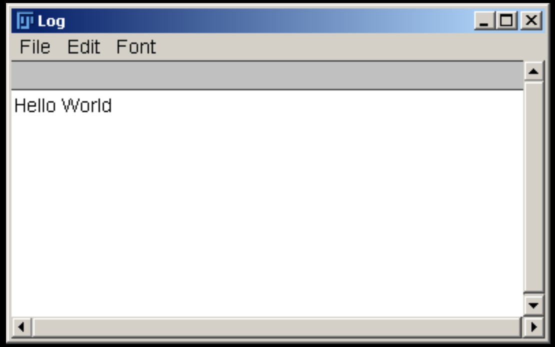
   [File -> Save As…]

Second way to run script

[Run -> Run] , or control-R (win) command-R (mac)



```
macro "pting out"{
	print("Hello World");
}
```



```
Hello World
```

By the way, you could start up script editor by ctrl-{ (win)

2-2 Basics

2-2-1 Hello World!

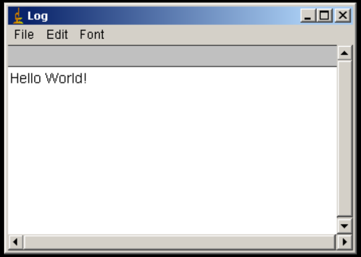
```
macro "print_out" {
    print("Hello World!");
}
```

Semi-colon at the end of command is very important

print() command

Parameter “Hello World”

*** { } → Braces define the boundary of macro.



```
Hello World!
```

2-2 Basics 2-2-1 Hello World!: EXERCISE

Code 1

```
1: macro "print_out" {
2:     print("Hello World!");
3: }
```

Exercise 2-1-1: Try modifying the print out text and check that the text will be printed in the "Log" window.

Exercise 2-1-2:

(1) Add another line “**print("\\Clear");**” after the second line (don’t forget the semi-colon at the end!).

(2) Then test also when you insert the same command in the third line. What happened?

Code 1.5

```
1: macro "print_out" {
2:          print("\\Clear");
3:          print("Hello World!");
4: }
```

Code 1.75

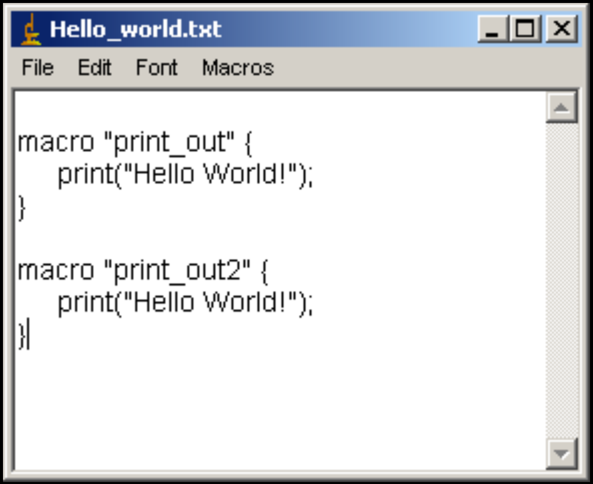
```
1: macro "print_out" {
2:          print("Hello World!");
3:          print("\\Clear");
4: }
```

2-2 Basics

2-2-1 Hello World!: EXERCISE

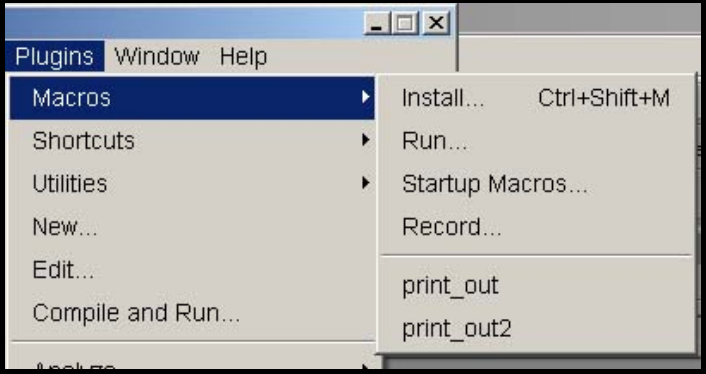
```
macro "print_out" {
    print("Hello World!");
}
```

Exercise 2-1-3: Multiple macros can exist in a single file. We call this “**macro sets**”. Duplicate the code you wrote by copying and pasting under the original. **The second macro should have a different name**. In the example here, the second macro is named “pirnt_out2”.



Hello_world.txt

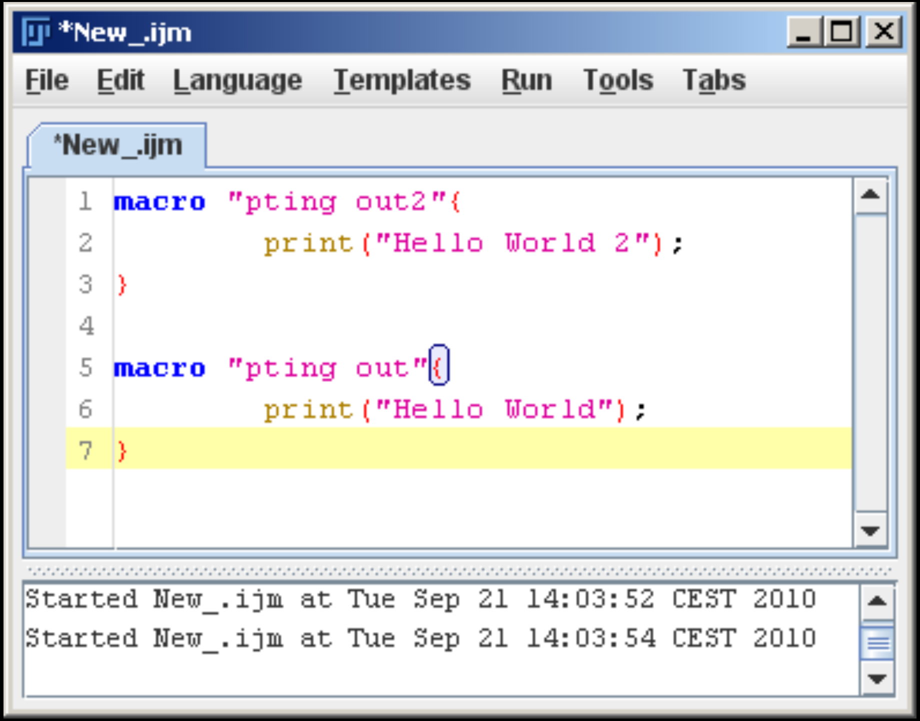
```
macro "print_out" {
    print("Hello World!");
}

macro "print_out2" {
    print("Hello World!");
}
```



2-2 Basics

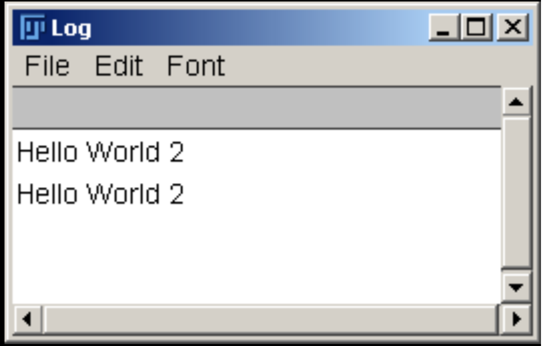
Second way to run script

[Run -> Run] (imited to the first macro)



```
macro "pting out2"{
		print("Hello World 2");
}

macro "pting out"{
		print("Hello World");
}
```

```
Started New_.ijm at Tue Sep 21 14:03:52 CEST 2010
Started New_.ijm at Tue Sep 21 14:03:54 CEST 2010
```

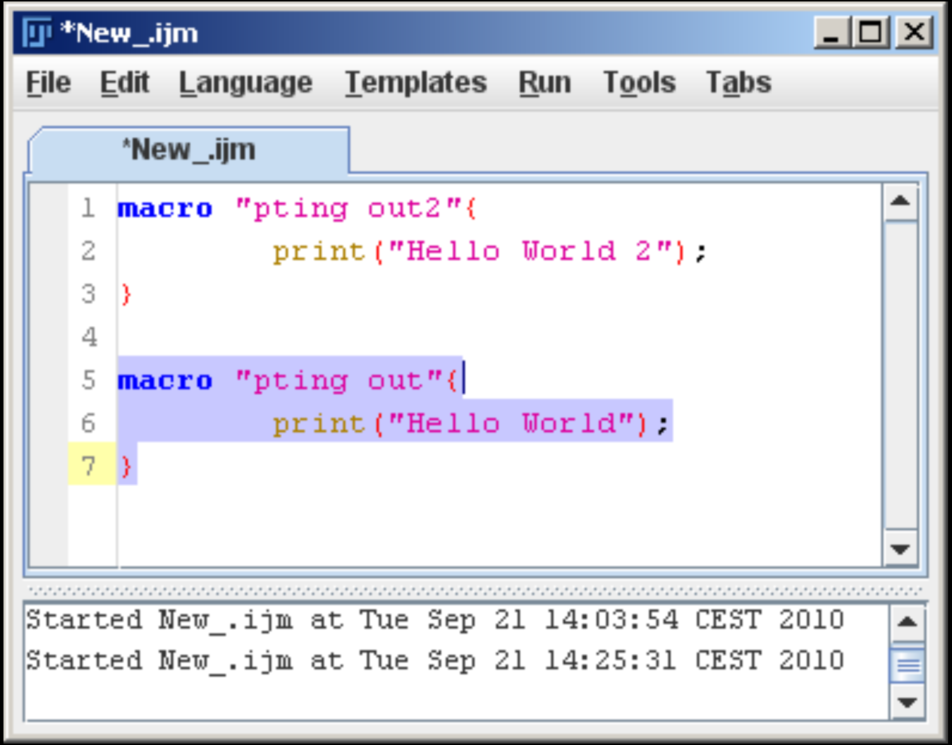


```
Hello World 2
Hello World 2
```

2-2 Basics

Third way to run script

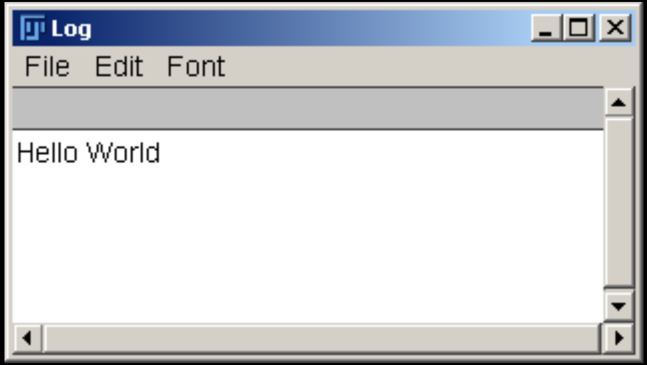
First make a selection (purple)



```
macro "pting out2"{
        print("Hello World 2");
}

macro "pting out"{
        print("Hello World");
}
```

```
Started New_.ijm at Tue Sep 21 14:03:54 CEST 2010
Started New_.ijm at Tue Sep 21 14:25:31 CEST 2010
```

… then **[Run -> Run Selected Code]**



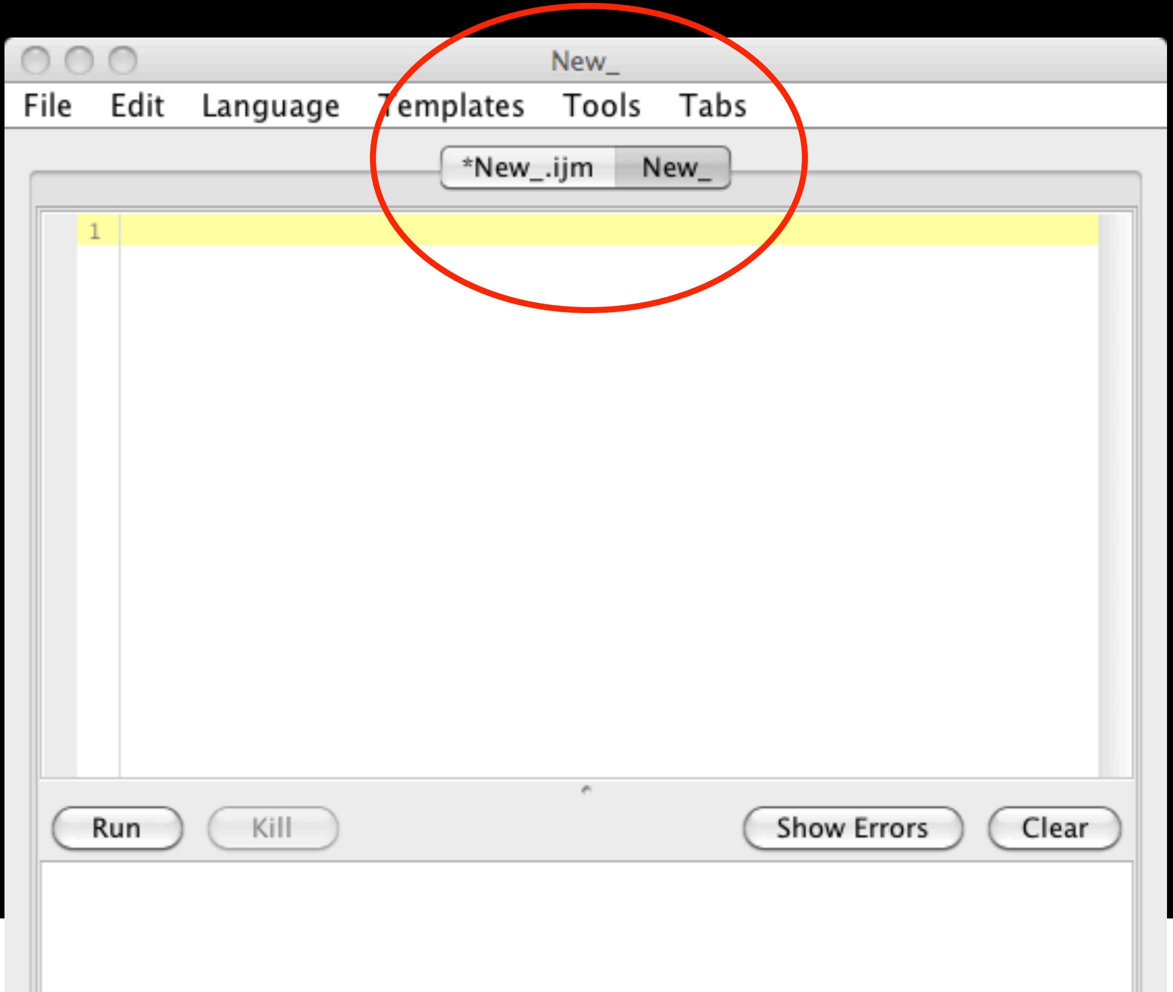
```
Hello World
```

2-2 Basics

By the way,

you could always make a new tab to edit another script while you still have the current one.

try
[File > New]

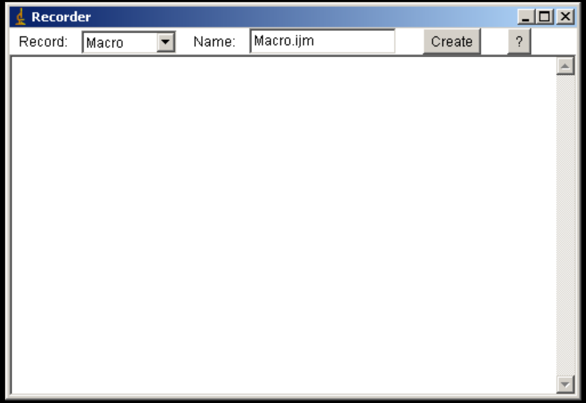


2-2 Basics

2-2-4 Including ImageJ Macro Commands into your macro

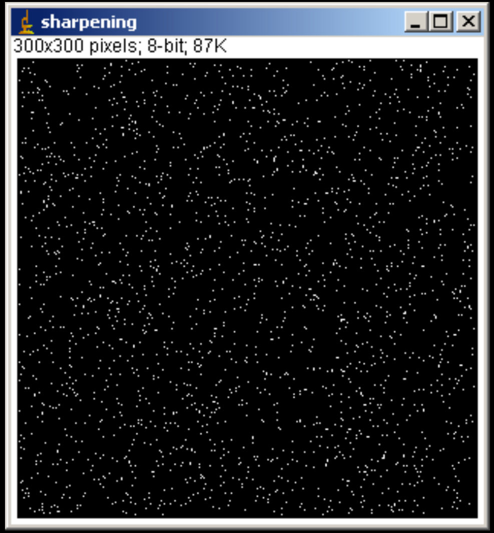
“Macro Recorder” a very powerful tool for macro programming

[PlugIns -> Macros -> Record…]



1. **[File -> New]** (size can be anything)

2. **[Process -> Noise -> Salt and Pepper]**


sharpening
300x300 pixels; 8-bit; 87K

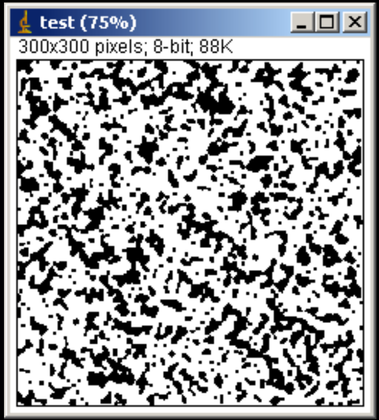
3. **[Process -> Filters -> Gaussian Blur]** (diameter=2)

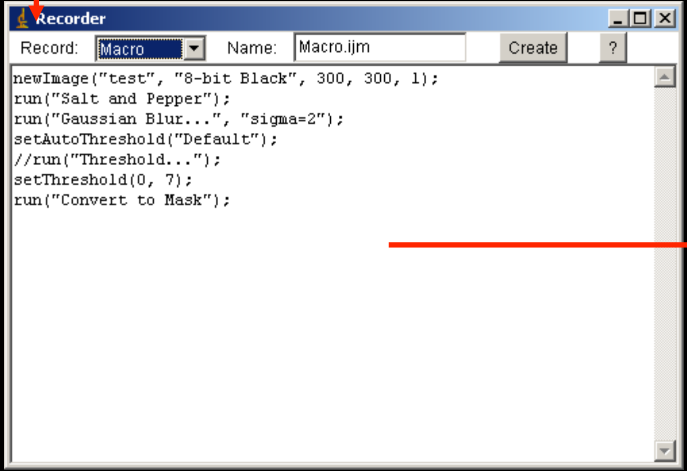
4. **[Image -> Adjust -> Threshold..]** then “Apply”

2-2 Basics

2-2-4 Including ImageJ Macro Commands into your macro

1. **[File -> New]** (size can be anything)
2. **[Process -> Noise -> Salt and Pepper]**
3. **[Process -> Filters -> Gaussian Blur]** (diameter=2)
4. **[Image -> Adjust -> Threshold..]** then “Apply”





```
newImage("test", "8-bit Black", 300, 300, 1);
run("Salt and Pepper");
run("Gaussian Blur...", "sigma=2");
setAutoThreshold("Default");
//run("Threshold...");
setThreshold(0, 7);
run("Convert to Mask");
```

```
macro "GB2_Thr" {
		newImage("test", "8-bit Black", 300, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", “sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

2-2-4 Including ImageJ Macro Commands into your macro

1. **[File -> New]**
2. **[Process -> Noise -> Salt and Pepper]**
3. **[Process -> Filters -> Gaussian Blur]** (diameter=2)
4. **[Image -> Adjust -> Threshold..]** then “Apply”

```
macro "GB2_Thr" {
        newImage("test", "8-bit Black", 300, 300, 1);
        run("Salt and Pepper");
        run("Gaussian Blur...", "sigma=2");
        setThreshold(0, 7);
        run("Convert to Mask");
}
```

2-2-4 Including ImageJ Macro Commands into your macro

1. **[File -> New]**
2. **[Process -> Noise -> Salt and Pepper]**
3. **[Process -> Filters -> Gaussian Blur]** (diameter=2)
4. **[Image -> Adjust -> Threshold..]** then “Apply”

```
macro "GB2_Thr" {
        newImage("test", "8-bit Black", 300, 300, 1);
        run("Salt and Pepper");
        run("Gaussian Blur...", "sigma=2");
        setThreshold(0, 7);
        run("Convert to Mask");
}
```

ImageJ macro reference:
[Help > Macro functions]

From “Build-in Macro Functions”

newImage(title, type, width, height, depth)

Opens a new image or stack using the name *title*. The string *type* should contain"8-bit", "16-bit", "32-bit" or "RGB". In addition, it can contain "white", "black" or "ramp" (the default is "white"). As an example, use "16-bit ramp" to create a 16-bit image containing a grayscale ramp. *Width* and *height* specify the width and height of the image in pixels. *Depth* specifies the number of stack slices.

2-2 Basics

2-2-4 Including ImageJ Macro Commands into your macro

1. **[File -> New]**
2. **[Process -> Noise -> Salt and Pepper]**
3. **[Process -> Filters -> Gaussian Blur]** (diameter=2)
4. **[Image -> Adjust -> Threshold..]** then “Apply”

```
macro "GB2_Thr" {
        newImage("test", "8-bit Black", 300, 300, 1);
        run("Salt and Pepper");
        run("Gaussian Blur...", "sigma=2");
        setThreshold(0, 7);
        run("Convert to Mask");
}
```

From “Build-in Macro Functions”

run("command"[, "options"])
Executes an ImageJ menu command. The optional second argument contains values that are automatically entered into dialog boxes (must be GenericDialog or OpenDialog). Use the Command Recorder (*Plugins>Macros>Record*) to generate run () function calls. Use string concatentation to pass a variable as an argument. With ImageJ 1.43 and later, variables can be passed without using string concatenation by adding "&" to the variable name.

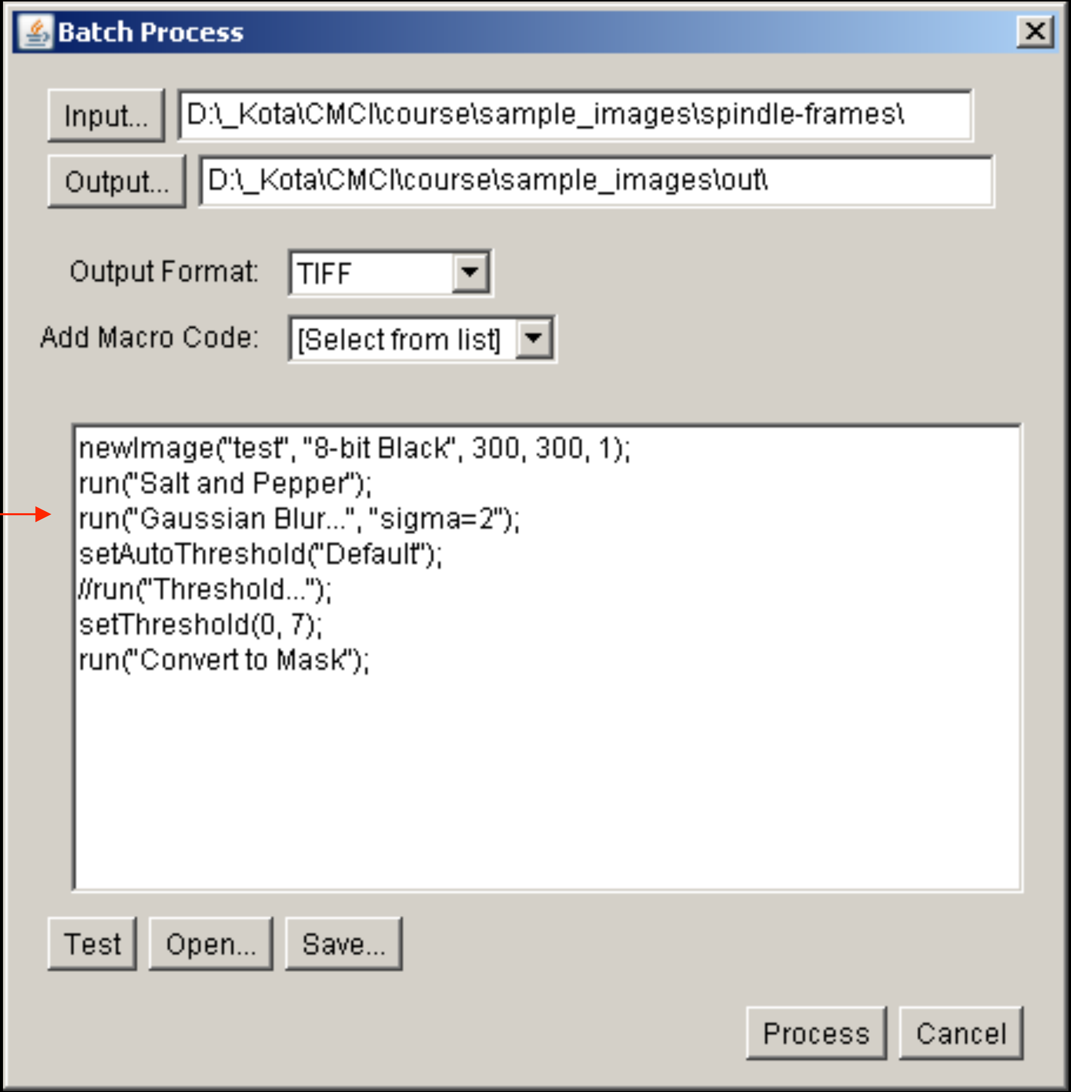
2-2 Basics

2-2-5 Batch Processing using “batch macro” function

… doing the same processing for all files in a folder.

[Process -> Batch -> Macro]

Copy & paste codes.



Input: D:_Kota\CMCI\course\sample_images\spindle-frames\

Output: D:_Kota\CMCI\course\sample_images\out\

Output Format: TIFF

Add Macro Code: [Select from list]

```
newImage("test", "8-bit Black", 300, 300, 1);
run("Salt and Pepper");
run("Gaussian Blur...", "sigma=2");
setAutoThreshold("Default");
//run("Threshold...");
setThreshold(0, 7);
run("Convert to Mask");
```

2-2-4 Including ImageJ Macro Commands into your macro

Let’s modify: more flexibility with image size.

```
macro "GB2_Thr" {
		newImage("test", "8-bit Black", 300, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

```
macro "GB2_Thr" {
		width = 500;
		newImage("test", "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

We call this a “**variable**”

Please run and test!

2-2-4 Including ImageJ Macro Commands into your macro

Let’s modify: more flexibility with image title.

```
macro "GB2_Thr" {
		width = 500;
		newImage("test", "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

```
macro "GB2_Thr" {
		width = 500;
		title = "noise image“;
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

Please run and test!

We call this a “**string**”

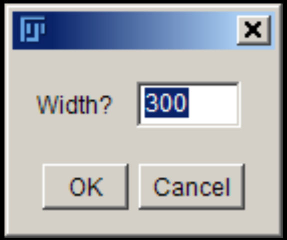
2-2-4 Including ImageJ Macro Commands into your macro

Let’s modify more!: asking user to input

```
macro "GB2_Thr" {
            width = 500;
            title = "noise image“;
            newImage(title, "8-bit Black", width, 300, 1);
            run("Salt and Pepper");
            run("Gaussian Blur...", "sigma=2");
            setThreshold(0, 7);
            run("Convert to Mask");
}
```

```
macro "GB2_Thr" {
            width = getNumber("Width?", 300);
            title = "noise image“;
            newImage(title, "8-bit Black", width, 300, 1);
            run("Salt and Pepper");
            run("Gaussian Blur...", "sigma=2");
            setThreshold(0, 7);
            run("Convert to Mask");
}
```

Please run and test!



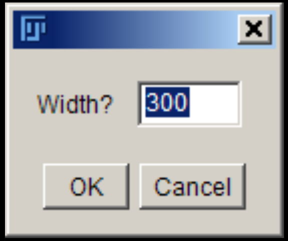
2-2-4 Including ImageJ Macro Commands into your macro

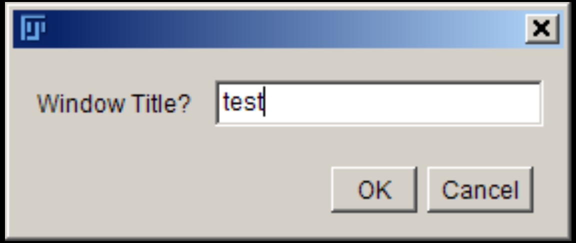
Let’s modify more!: asking user to input

```
macro "GB2_Thr" {
		width = getNumber("Width?", 300);
		title = "noise image“;
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

```
macro "GB2_Thr" {
		width = getNumber("Width?", 300);
		title = getString("Window Title?", "test");
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
}
```

Please run and test!


Width? 300


Window Title? test

```
newImage("test", "8-bit Black", 300, 300, 1);
```

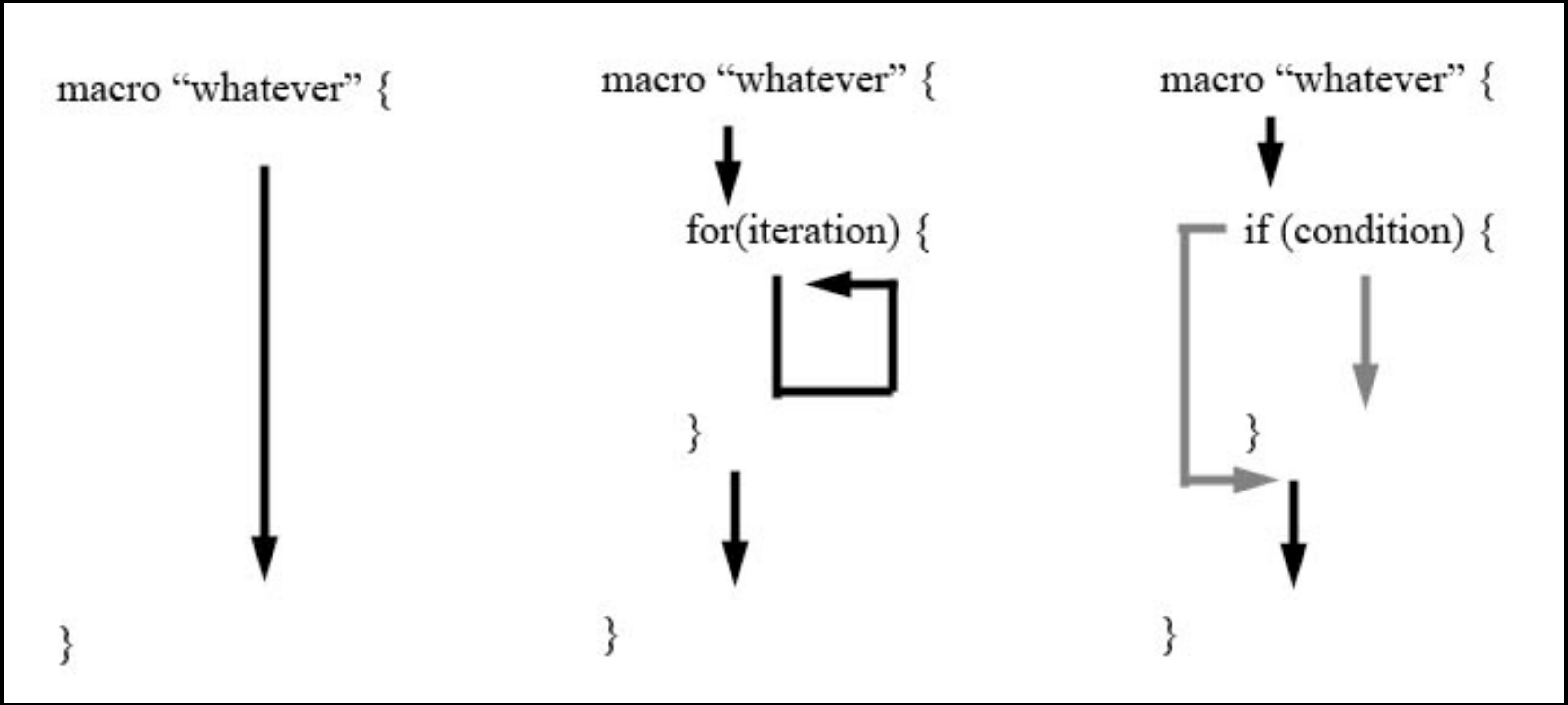
From “Build-in Macro Functions”

newImage(title, type, width, height, depth)

Opens a new image or stack using the name *title*. The string *type* should contain"8-bit", "16-bit", "32-bit" or "RGB". In addition, it can contain "white", "black" or "ramp" (the default is "white"). As an example, use "16-bit ramp" to create a 16-bit image containing a grayscale ramp. *Width* and *height* specify the width and height of the image in pixels. *Depth* specifies the number of stack slices.

ImageJ macro reference:
[Help > Macro functions]

2-3 Condition and Loops



Punk

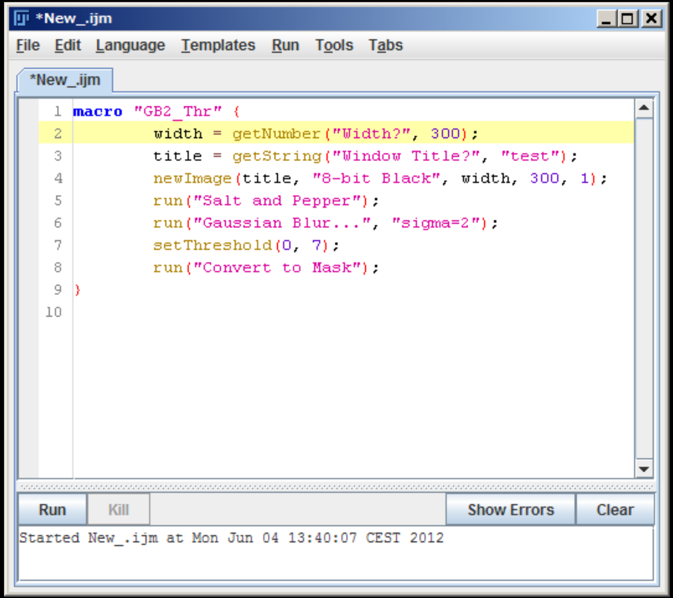
Loop

Condition

2-2 Basics

Adding a loop: an example using erosion.

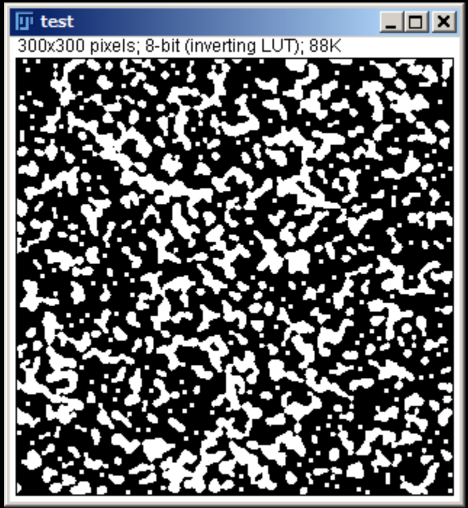
1. Run the macro…



```
macro "GB2_Thr" {
	width = getNumber("Width?", 300);
	title = getString("Window Title?", "test");
	newImage(title, "8-bit Black", width, 300, 1);
	run("Salt and Pepper");
	run("Gaussian Blur...", "sigma=2");
	setThreshold(0, 7);
	run("Convert to Mask");
}
```

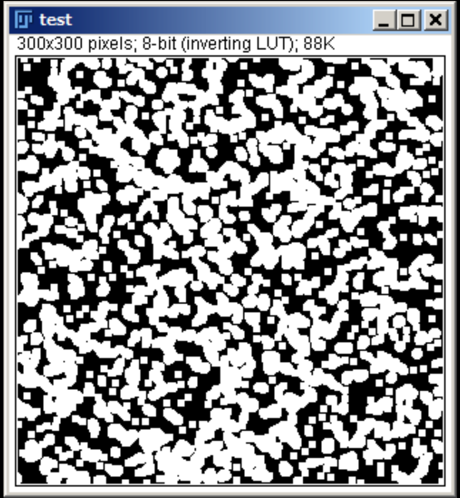
```
Started New_.ijm at Mon Jun 04 13:40:07 CEST 2012
```

2. … to create an image



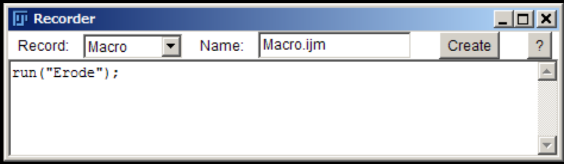
3. Use the macro recorder to record the following command:

[Process -> Binary -> Erode]



4. Append the command ***run("Erode");*** to the current macro.

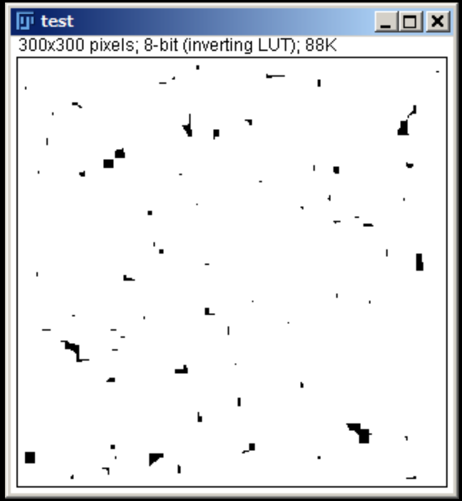
```
macro "GB2_Thr" {
		width = getNumber("Width?", 300);
		title = getString("Window Title?", "test");
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
		run("Erode");
}
```



```
run("Erode");
```

4. Add Looping.

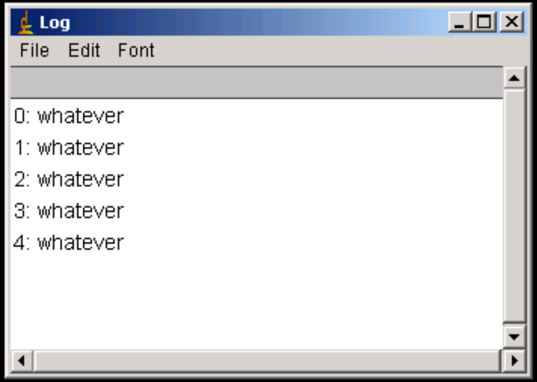
```
macro "GB2_Thr" {
		width = getNumber("Width?", 300);
		title = getString("Window Title?", "test");
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
		for (i = 0; i < 5; i+=1){
				run("Erode");
		}
}
```



2-3-1 Loop by "for"

Make a new tab and test it by yourself!!

```
macro "loop1" {
        txt = “whatever”;
        for( i =0 ; i <5 ; i += 1 ) {
                print(i + “: ” + txt);
        }
}
```



```
0: whatever
1: whatever
2: whatever
3: whatever
4: whatever
```

line 3 *for(i=0; i<5; i+=1)* sets the looping condition.

initialize counter

Condition for exiting the loop

Increment per loop

i++

*** braces are again to define the boundary of “for” looping

2-3-1 Loop: for-statement

```
macro "loop1" {
            txt = “whatever”;
            for( i =0 ; i <5 ; i += 1 ) {
                        print(i + “: ” + txt);
            }
}
```

Exercise 2-3-1-1:

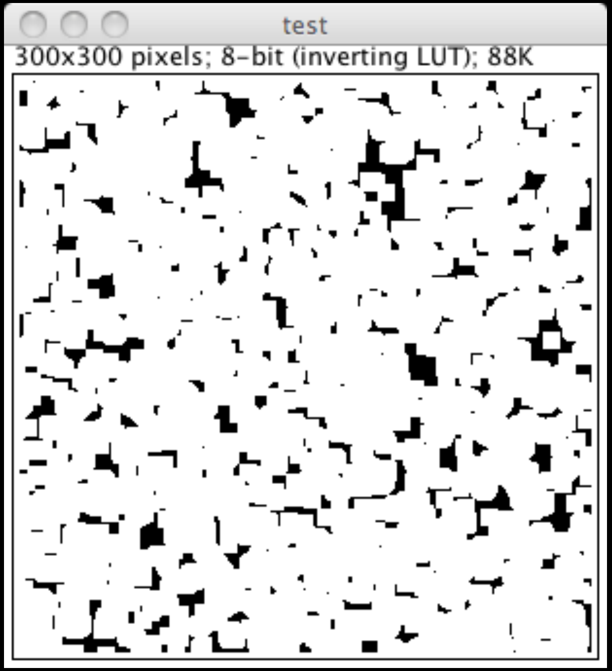
(1) Change the first parameter in *for(**i=0**;i<5;i+=1)* so that the macro prints out only 1 line. → i=4

(2) Change the second parameter in *for(i=0;**i<5**;i+=1)* so that the macro prints out 10 lines. → i<10

(3) Change the third parameter in *for(i=0;i<5;**i+=1**)* so that the macro prints out 10 lines. → i+=0.5

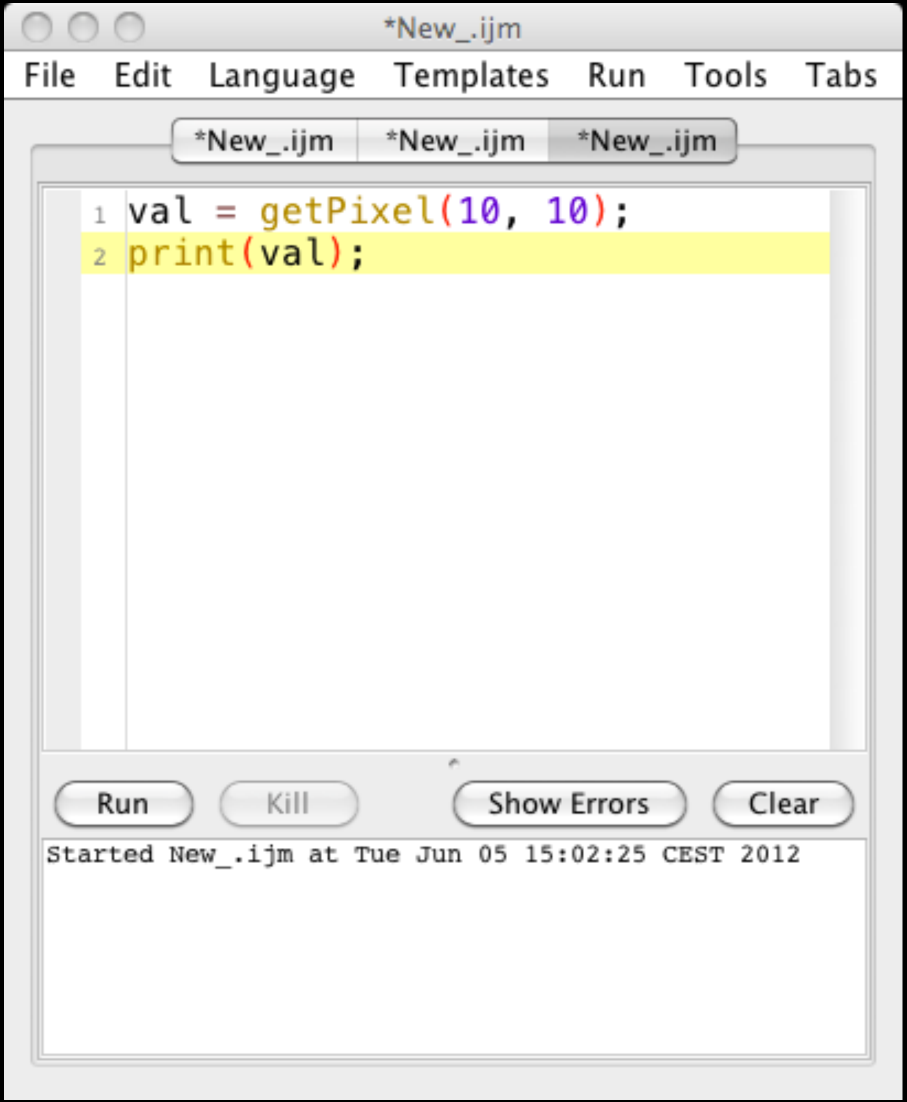
2-2 Basics using if – else

```
macro "GB2_Thr" {
		width = getNumber("Width?", 300);
		title = getString("Window Title?", "test");
		newImage(title, "8-bit Black", width, 300, 1);
		run("Salt and Pepper");
		run("Gaussian Blur...", "sigma=2");
		setThreshold(0, 7);
		run("Convert to Mask");
		for (i = 0; i < 3; i+=1){
			run("Erode");
		}
}
```



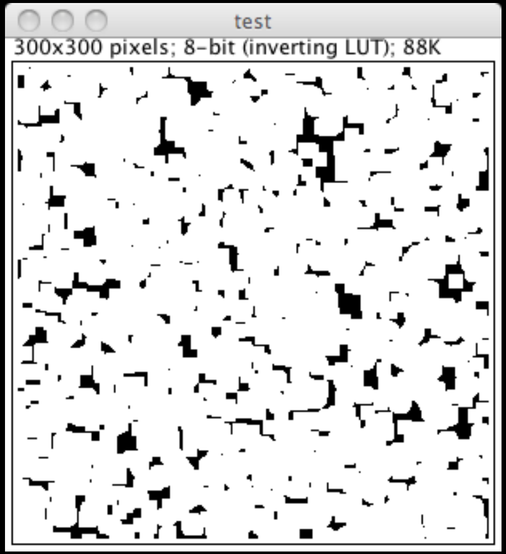
We use this synthetic image to do the following tutorial.
So make a new tab!

write these two lines and run it.

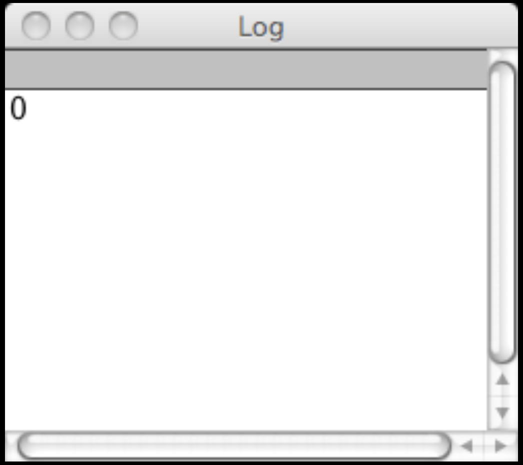


```
val = getPixel(10, 10);
print(val);
```

```
Started New_.ijm at Tue Jun 05 15:02:25 CEST 2012
```



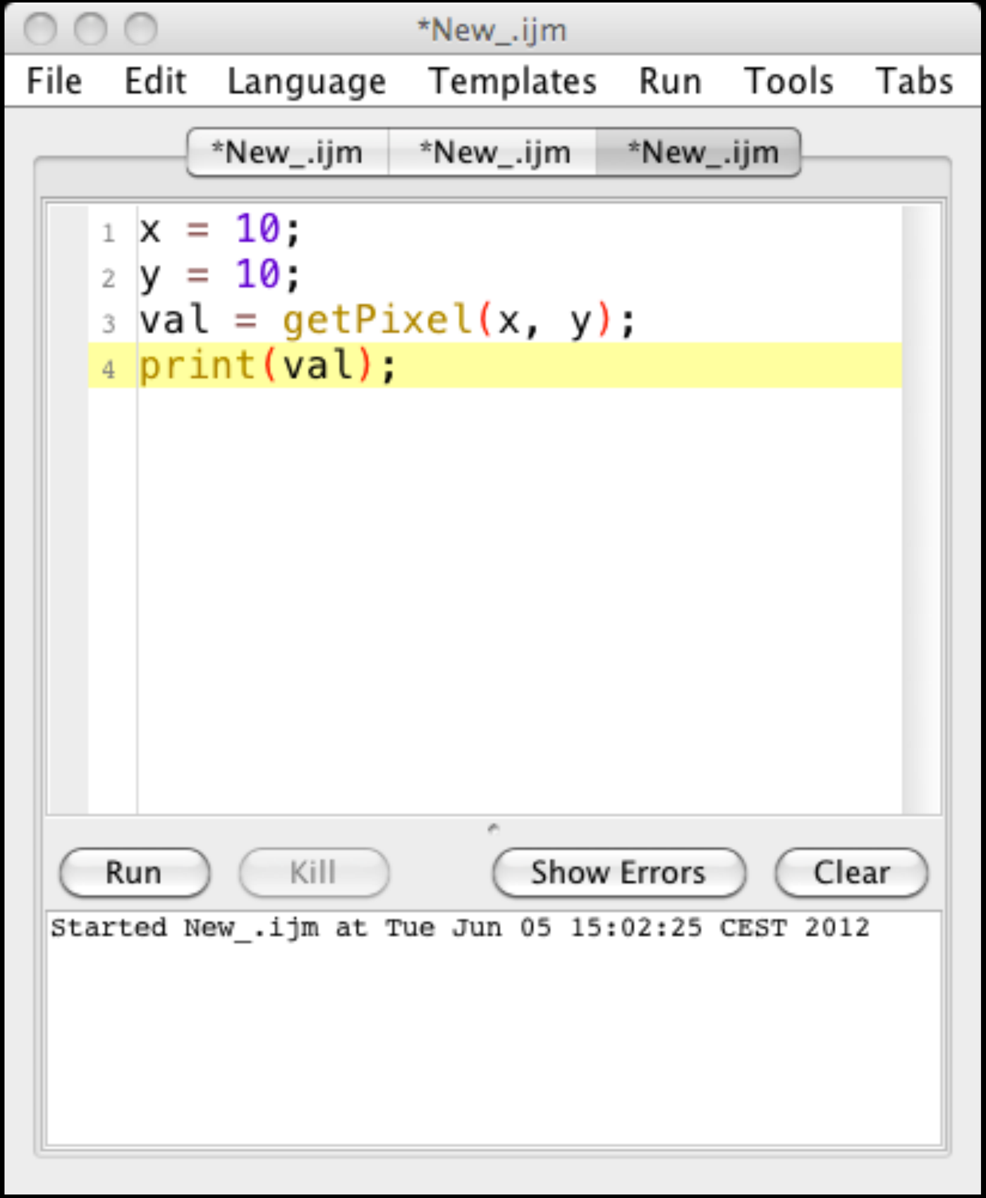
this will print a number in the log window…



```
0
```

… we now further extend this…

rewrite it using variables. Results should be the same.

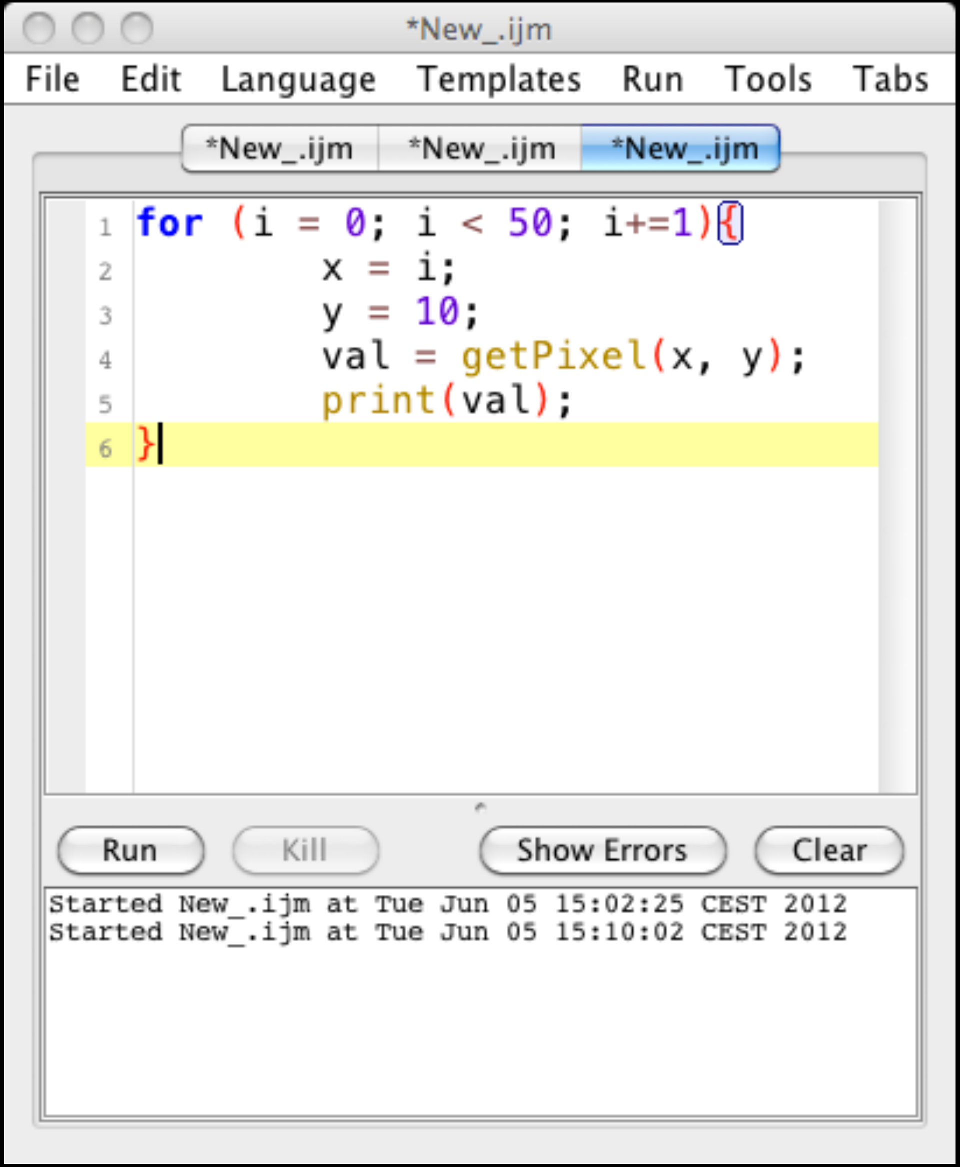

*New_.ijm
File Edit Language Templates Run Tools Tabs
*New_.ijm *New_.ijm *New_.ijm
Run Kill Show Errors Clear

```
x = 10;
y = 10;
val = getPixel(x, y);
print(val);
```

```
Started New_.ijm at Tue Jun 05 15:02:25 CEST 2012
```

… we now further extend this…

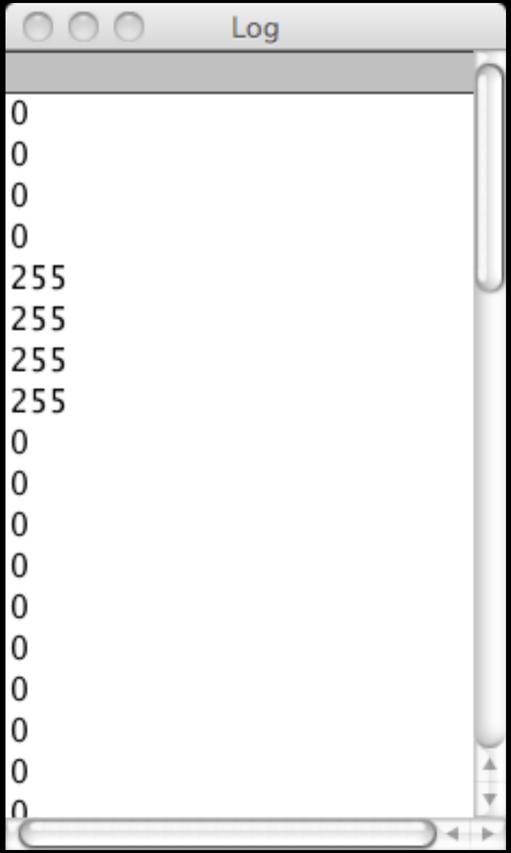
Now, enclose the code within loop.
Then replace the value of x from 10 to *i*



```
for (i = 0; i < 50; i+=1){
        x = i;
        y = 10;
        val = getPixel(x, y);
        print(val);
}
```

```
Started New_.ijm at Tue Jun 05 15:02:25 CEST 2012
Started New_.ijm at Tue Jun 05 15:10:02 CEST 2012
```

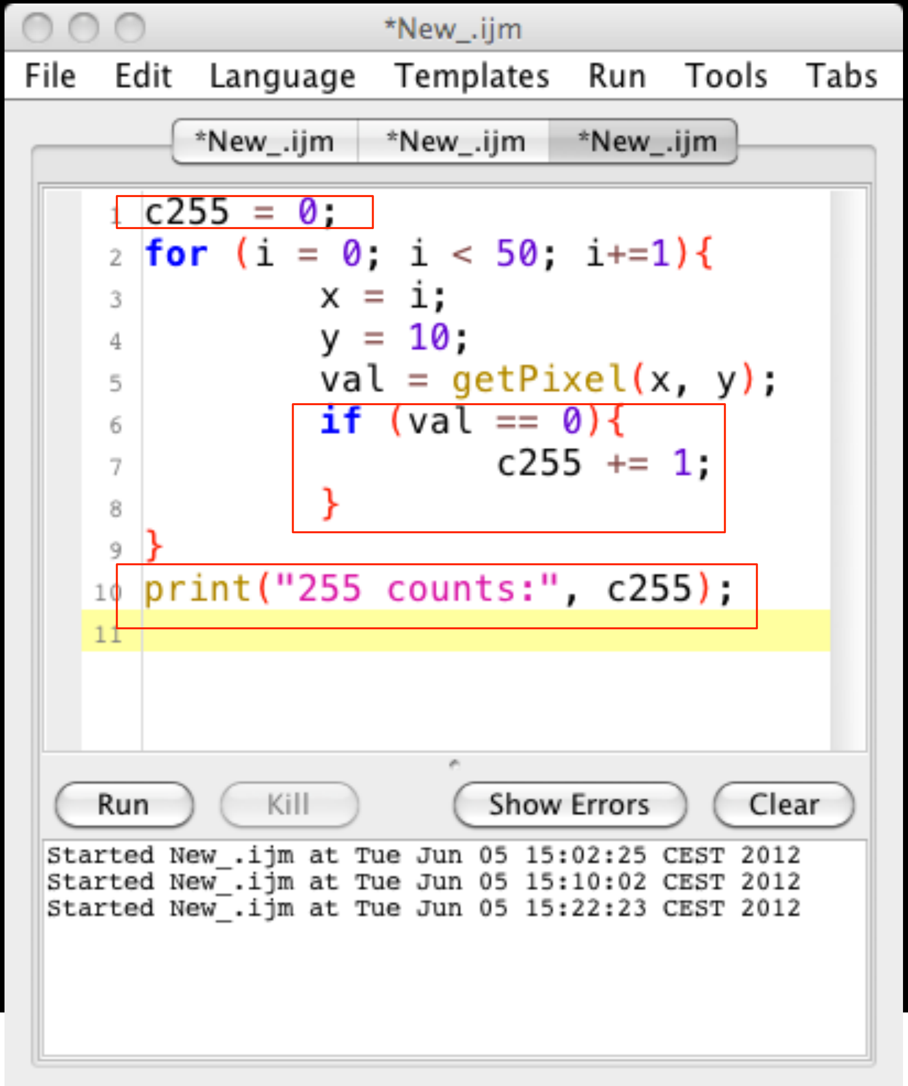
running this script outputs 50 numbers. These are the pixel values from (0, 10) to (49, 10)



```
0
0
0
0
255
255
255
255
0
0
0
0
0
0
0
0
0
```

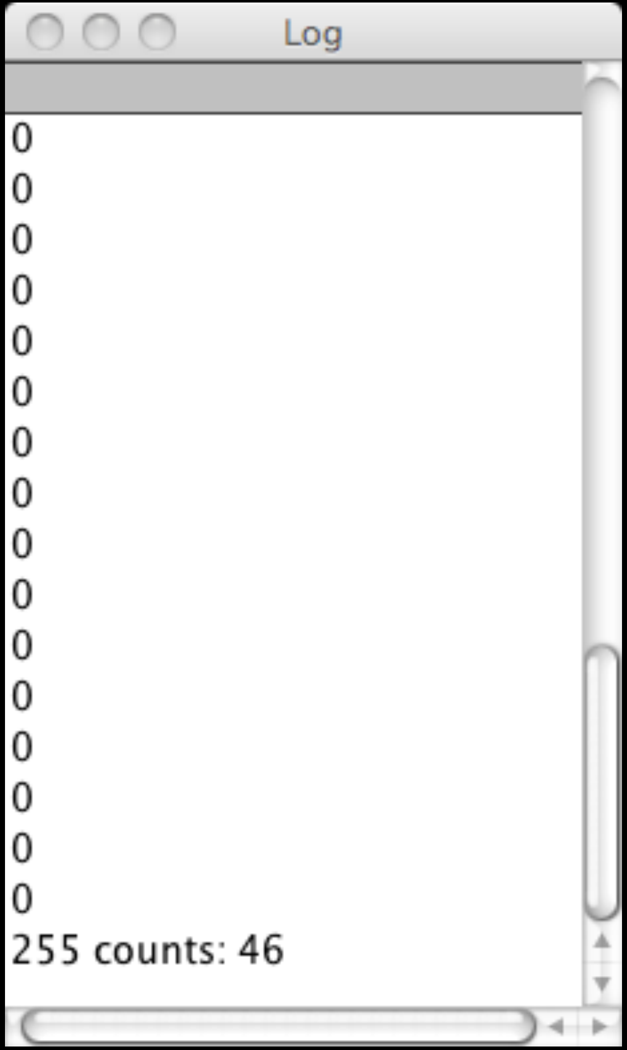
… we now further extend this…

We count the number of pixels those with intensity 255.
- first line initializing a variable “c255”.
-three lines for “if” the pixel vale is 255.
- after looping, prints out the counted number.


*New_.ijm
File Edit Language Templates Run Tools Tabs
*New_.ijm *New_.ijm *New_.ijm
Run Kill Show Errors Clear

```
c255 = 0;
for (i = 0; i < 50; i+=1){
        x = i;
        y = 10;
        val = getPixel(x, y);
        if (val == 0){
                c255 += 1;
        }
}
print("255 counts:", c255);
```

```
Started New_.ijm at Tue Jun 05 15:02:25 CEST 2012
Started New_.ijm at Tue Jun 05 15:10:02 CEST 2012
Started New_.ijm at Tue Jun 05 15:22:23 CEST 2012
```



```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
255 counts: 46
```

We could count the number of pixels with value 255 scanning the full range.

```
c255 = 0;
for (j = 0; j < getHeight(); j+=1){
            for (i = 0; i < getWidth(); i+=1){
                        val = getPixel(i, j);
                        if (val == 0){
                                    c255 += 1;
                        }
            }
}
print("255 counts:", c255);
```

2-3 Condition and Loops

2-3-4 Conditions: if-else-statement

Code 12

```
macro "Condition_if_else 1"{
        a = getNumber("Input a number", 5);
        if (a == 5) {
                print(a + ": The number is 5 ");
        }
}
```

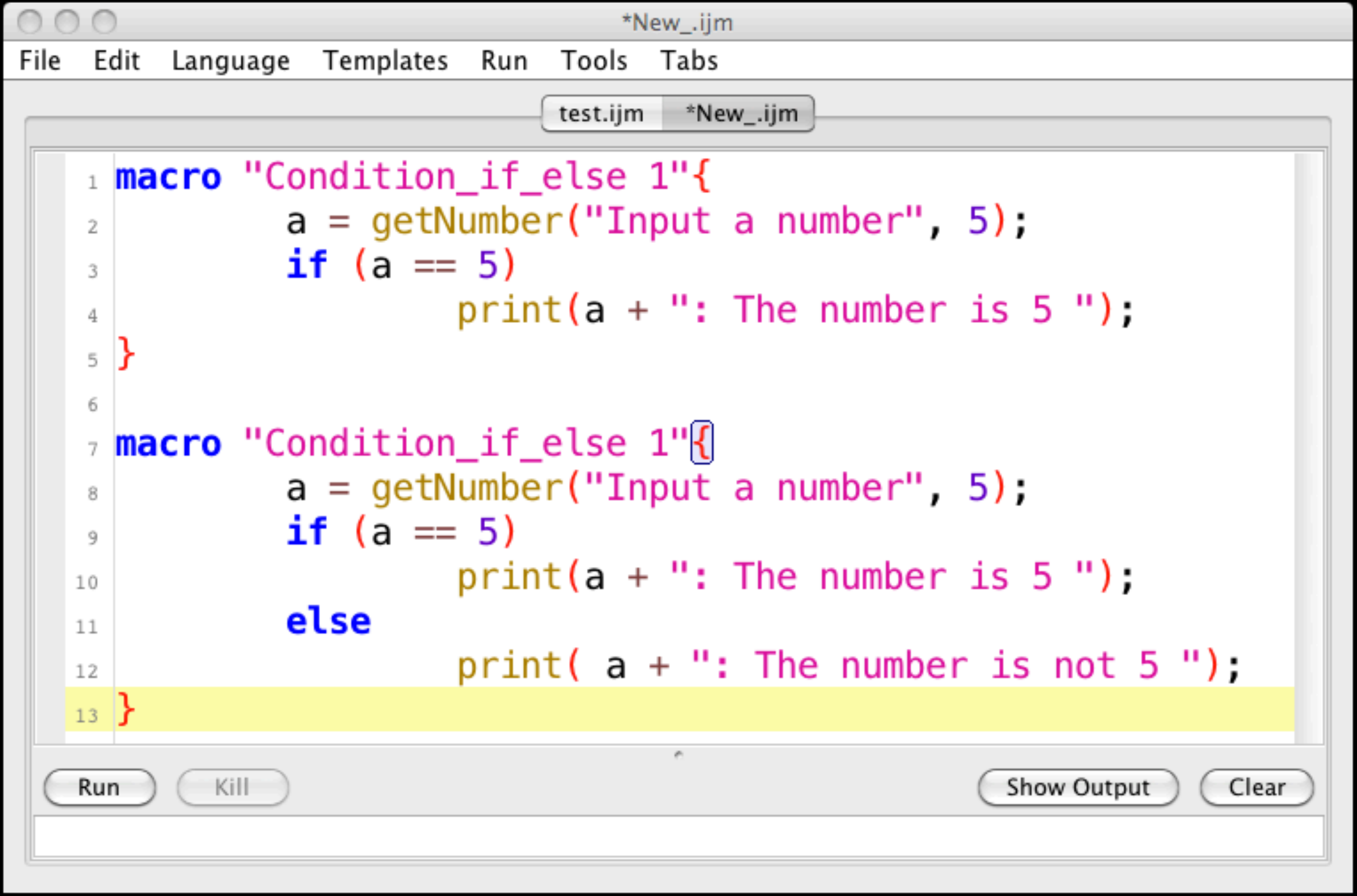
Code 12.5

```
macro "Condition_if_else 2"{
        a = getNumber("Input a number", 5);
        if (a == 5) {
                print( a + ": The number is 5 ");
        } else {
                print( a + ": The number is not 5 ");
        }
        print("--------------");
}
```

Exercise:

Instead of checking if number is 5, modify the code, so that it evaluates if the given number is greater or less than 5.

if there is only one line nested, no {} required!


*New_.ijm
File Edit Language Templates Run Tools Tabs
test.ijm *New_.ijm
Run Kill Show Output Clear

```
macro "Condition_if_else 1"{
        a = getNumber("Input a number", 5);
        if (a == 5)
                print(a + ": The number is 5 ");
}

macro "Condition_if_else 1"{
        a = getNumber("Input a number", 5);
        if (a == 5)
                print(a + ": The number is 5 ");
        else
                print( a + ": The number is not 5 ");
}
```

2-3 Condition and Loops

2-3-2 Stack Management by for-statement.

Code 10

```
macro "Measure Ave Intensity Stack" {
        frames=nSlices;
        run("Set Measurements...", "  mean redirect=None decimal=3");
        run("Clear Results");
        for(i=0; i<frames; i++) {
                currentslice=i+1;
                setSlice(currentslice);
                run("Measure");
        }
}
```

2-3 Condition and Loops

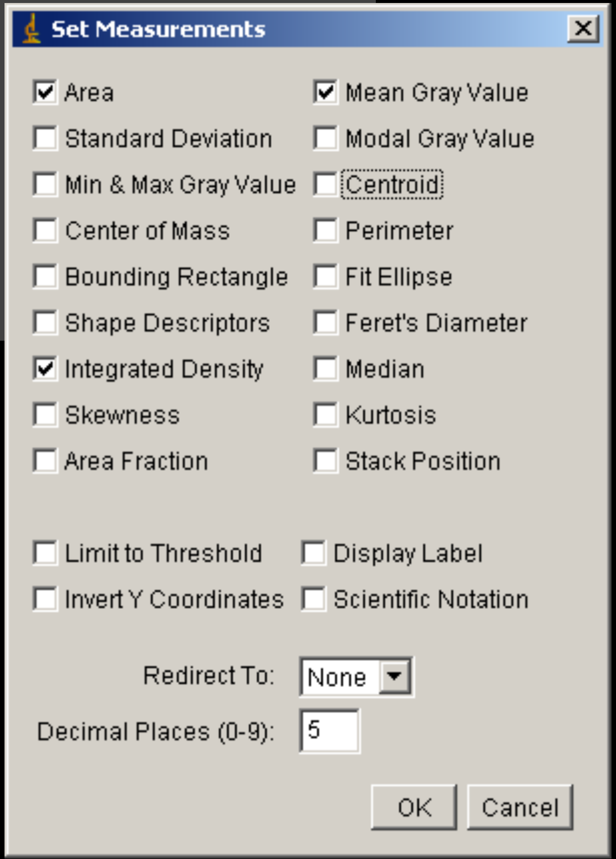
2-3-2 Stack Management by for-statement.

Code 10

```
macro "Measure Ave Intensity Stack" {
        frames=nSlices;
        run("Set Measurements...", "  mean redirect=None decimal=3");
        run("Clear Results");
        for(i=0; i<frames; i++) {
                currentslice=i+1;
                setSlice(currentslice);
                run("Measure");
        }
}
```

nSlices
Returns the number of slices in the current stack.
Returns 1 if the current image is not a stack.

setSlice(n)
Displays the *n*th slice of the active stack. Does nothing if the active image is not a stack.



Set Measurements

✓ Area	✓ Mean Gray Value
☐ Standard Deviation	☐ Modal Gray Value
☐ Min & Max Gray Value	☐ Centroid
☐ Center of Mass	☐ Perimeter
☐ Bounding Rectangle	☐ Fit Ellipse
☐ Shape Descriptors	☐ Feret's Diameter
✓ Integrated Density	☐ Median
☐ Skewness	☐ Kurtosis
☐ Area Fraction	☐ Stack Position
☐ Limit to Threshold	☐ Display Label
☐ Invert Y Coordinates	☐ Scientific Notation

Redirect To: None

Decimal Places (0-9): 5

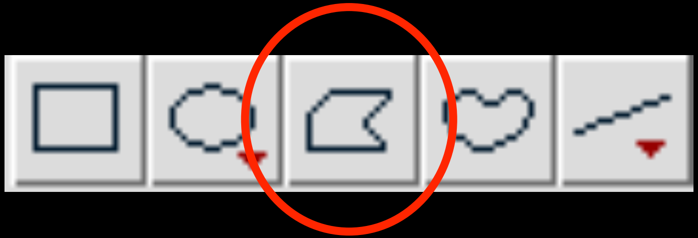
2-3 Condition and Loops

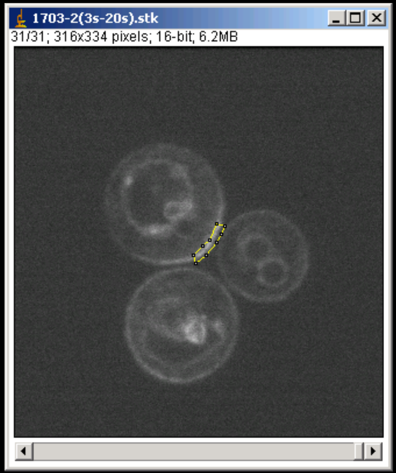
2-3-2 Stack Management by for-statement.

Code 10

```
macro "Measure Ave Intensity Stack" {
        frames=nSlices;
        run("Set Measurements...", "  mean redirect=None decimal=3");
        run("Clear Results");
        for(i=0; i<frames; i++) {
                currentslice=i+1;
                setSlice(currentslice);
                run("Measure");
        }
}
```

Open an example stack **1703-2(3s-20s).stk**. Select FRAPped region by polygon ROI tool. Execute the macro. Results will be printed in the Results window.





2-3 Condition and Loops

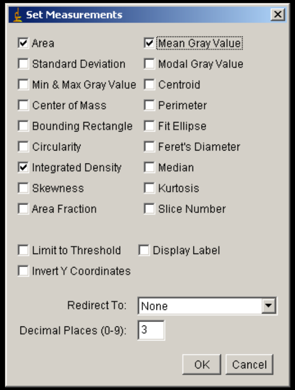
2-3-2 Stack Management by for-statement.

Code 10

```
macro "Measure Ave Intensity Stack" {
        frames=nSlices;
        run("Set Measurements...", "  mean redirect=None decimal=3");
        run("Clear Results");
        for(i=0; i<frames; i++) {
                currentslice=i+1;
                setSlice(currentslice);
                run("Measure");
        }
}
```

Exercise 3-2-1: Modify code 10 to include more measurement parameters (whatever you like), and test the macro. Check the results.

Hint: use



✓ Area	✓ Mean Gray Value
☐ Standard Deviation	☐ Modal Gray Value
☐ Min & Max Gray Value	☐ Centroid
☐ Center of Mass	☐ Perimeter
☐ Bounding Rectangle	☐ Fit Ellipse
☐ Circularity	☐ Feret's Diameter
✓ Integrated Density	☐ Median
☐ Skewness	☐ Kurtosis
☐ Area Fraction	☐ Slice Number
☐ Limit to Threshold	☐ Display Label
☐ Invert Y Coordinates	

Redirect To: None

Decimal Places (0-9): 3

+ Macro Recorder

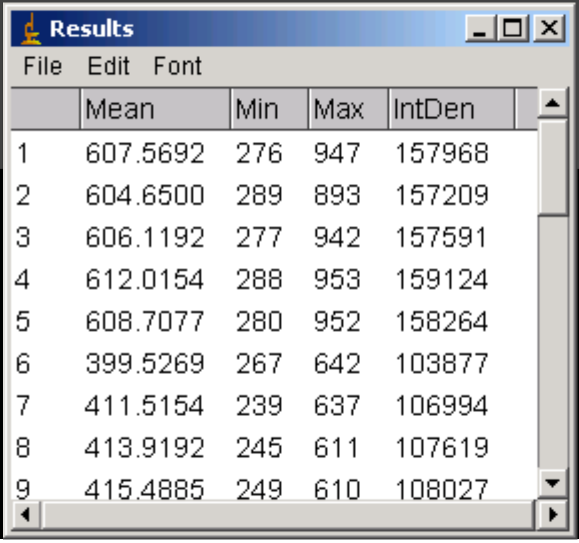
2-3 Condition and Loops

2-3-2 Stack Management by for-statement.

Exercise 3-2-1: Modify code 10 to include more measurement parameters (whatever you like), and test the macro. Check the results.

Code 10

```
macro "Measure Ave Intensity Stack" {
        frames=nSlices;
        run("Set Measurements...", "  mean min integrated redirect=None decimal=3");
        run("Clear Results");
        for(i=0; i<frames; i++) {
                currentslice=i+1;
                setSlice(currentslice);
                run("Measure");
        }
}
```



	Mean	Min	Max	IntDen
1	607.5692	276	947	157968
2	604.6500	289	893	157209
3	606.1192	277	942	157591
4	612.0154	288	953	159124
5	608.7077	280	952	158264
6	399.5269	267	642	103877
7	411.5154	239	637	106994
8	413.9192	245	611	107619
9	415.4885	249	610	108027

2-2 Basics

2-2 Basics

2-2-2 Variables, Strings

String Assignments

Code 2

```
macro "print_out" {
        text = ”Hello World!";
        print(text);
        text = ”Bye World!";
        print(text);
}
```

String Variable

Code 3

```
macro "print_out" {
        text1 = "Hello";
        text2 = " World!";
        text3 = text1 + text2;
        print(text3);
}
```

String concatenation

Exercise 2-2-2-1: Add more string variables and make a longer sentence. Check your macro by running it.

2-2-2 Variables, Strings

Numerical variable Assignments

```
message_text = 256;
```
→ Nummerical

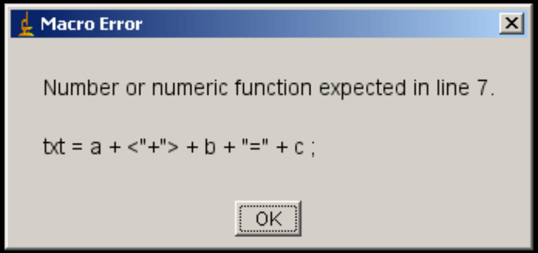
```
message_text = “256”;
```
→ String function

Code 4

```
1: macro "print_out_calc" {
2:          a = 1;
3:          b = 2;
4:          c = a + b;
5:          print("\\Clear");
6:          print(c);
7:          print(a + "+"+ b + "="+c);
8:          txt=""+a + "+"+ b + "="+c;
9:          print(txt);
10: }
```

Numerical variable

```
txt=a + "+"+ b + "="+c;
```


Macro Error

Number or numeric function expected in line 7.

```
txt = a + <"+"> + b + "=" + c ;
```

OK

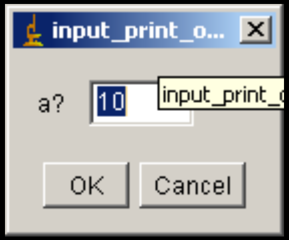
Exercise 2-2-2: Modify the code 4, so that the calculation involves subtraction (-), multiplication (*) and division (/).

2-2 Basics

2-2-3 parameter input by user

Code 5

```
macro "input_print_out_calc" {
        a = getNumber("a?", 10);
        b = getNumber("b?", 5);
        c = a*b;
        print("\\Clear");
        print(c);
}
```



Numerical Function

Clears Log Window

Syntax: **getNumber**(message string, default number)
return value: user input number

2-2 Basics

2-2-3 parameter input by user

Code 6

```
macro "input_print_out_str" {
            a = getString("a?", “hello”);
            b = getString("b?", “ world!”);
            c = a+b;
            print("\\Clear");
            print(c);
}
```

String Function

Syntax: **getString**(message string, default string)
return value: user input string

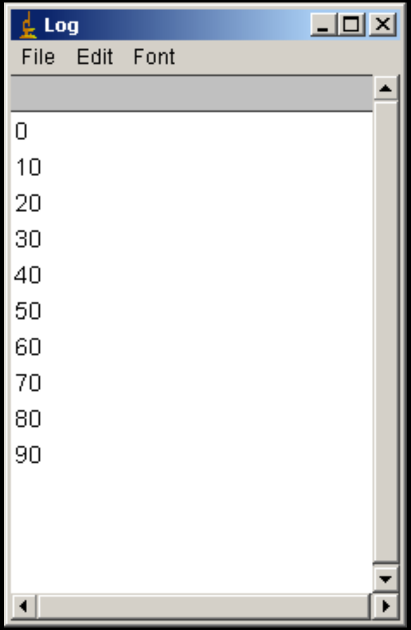
Exercise 2-3-1: Run the code 6 and input 1 for *a* and 2 for *b*. What happened? **Explain the reason**.

2-3 Conditions and Loops

2-3-3 Loop: while-statement

Code11

```
macro "while looping1" {
        counter=0;
        while (counter<=90) {
                print(counter);
                counter = counter + 10;
        }
}
```



```
0
10
20
30
40
50
60
70
80
90
```

Exercise 2-3-3-1-1:

(1) Change code 11 so that it uses "+=" sign.

(2) (2) Change code 11 so that it uses "++" sign, and prints out integers from 0 to 9.

```
counter = counter + 10;  →  counter += 10;
```

```
counter = counter - 10;  →  counter -= 10;
counter = counter * 10;  →  counter *= 10;
counter = counter / 10;  →  counter /= 10;
```

```
counter += 1;    counter ++;
counter -= 1;    counter --;
```

2-3 Condition and Loops

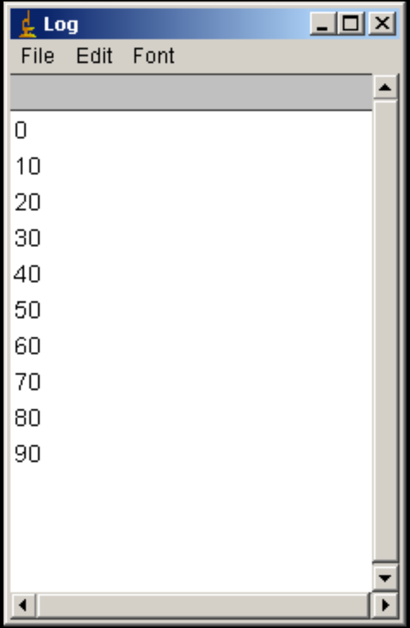
2-3-3 Loop: while-statement

Code11

```
1: macro "while looping1" {
2:         counter=0;
3:         while (counter<=90) {
4:                 print(counter);
5:                 counter += 10;
6:         }
7: }
```

Code 11.5

```
1: macro "while looping2" {
2:         counter=0;
3:         do {
4:                 print(counter);
5:                 counter += 10;
6:         } while (counter<0)
7: }
```



```
0
10
20
30
40
50
60
70
80
90
```

Exercise 2-3-3-1-2: Change the line 3 of code 11 to “while (counter <0)” and check the effect.

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

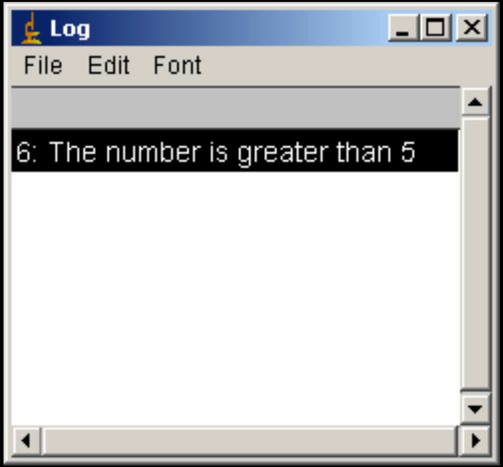
Exercise 2-3-3-1-3: Modify code 11 so that the macro prints out numbers from 200 to 100, with an increment of -10.

2-3-4 Conditions: if-else-statement

Code 12

```
macro "Condition_if_else 1"{
        a = getNumber("Input a number", 5);
        if (a == 5) {
                print( a + ": The number is 5 ");
        }
}
```

Evaluate the condition o
numerical variable
input_num



```
6: The number is greater than 5
```

<u>Exercise:</u>
<u>write the code!</u>
<u>Don’t delete the code, we extend it in the next slide!</u>

2-3-4 Conditions: if-else-statement : complex conditions

Code 12.75

```
1: macro "Condition_if_else 3"{
2:         a = getNumber("Input a number 1", 5);
3:         b = getNumber("Input a number 2", 6);
4:         message0 = ""+ a + ","+ b;                    //use this string four times
5:         if ( (a == 5) && ( b == 6) ) {
6:                 print(message0+ ": The parameter1 is 5 and the parameter2 is 6");
7:         } else {
8:                 if (a != 5) && (b != 6) {
9:                         print(message0 + ": The parameter1 is not 5 and the parameter2 is not 6");
10:                } else {
11:                        if (b == 6) {
12:                                print(message0 + ": The parameter1 is NOT 5 but the parameter2 is 6");
13:                        } else {
14:                                print(message0 + ": The parameter1 is 5 but the parameter2 is NOT 6");
15:                        }
16: }
```

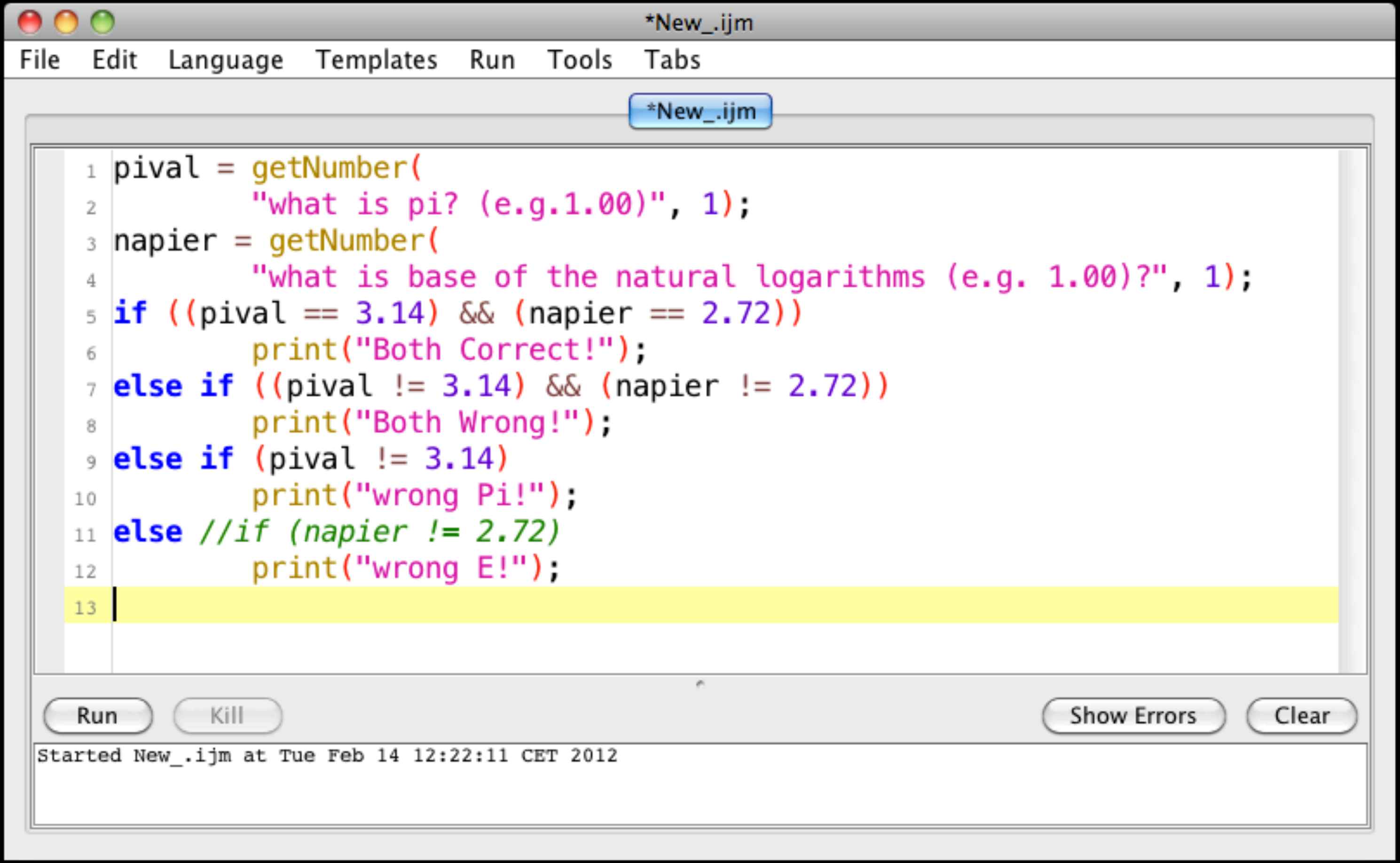
Line 5: **if ((a == 5) && (b == 6))**

Line 8: **if ((a != 5) && (b != 6))**

&&	**boolean AND**
||	**boolean OR**

Exercise

if there is only one line nested, no {} required!



```
pival = getNumber(
        "what is pi? (e.g.1.00)", 1);
napier = getNumber(
        "what is base of the natural logarithms (e.g. 1.00)?", 1);
if ((pival == 3.14) && (napier == 2.72))
        print("Both Correct!");
else if ((pival != 3.14) && (napier != 2.72))
        print("Both Wrong!");
else if (pival != 3.14)
        print("wrong Pi!");
else //if (napier != 2.72)
        print("wrong E!");
```

```
Started New_.ijm at Tue Feb 14 12:22:11 CET 2012
```

2-3-4 Conditions: if-else-statement : Application

(see the actual macro working first)

2-3-4 Conditions: if-else-statement : Application

Code 13

```
macro "Generate Dot Movement back and forth" {
// **** initial values ****
        sizenum=10;             //dot size
        int=255;                //dot intensity
        frames=50;              //frames in stack
        w=200;                  //width of frame
        h=50;                   //height of frame
        x_position = 0;         // x position: changes with speed defined later
        y_position= (h/2)-(sizenum/2);      //y positon of the oval top-left corner: constant
//**** set colors *****
        setForegroundColor(int, int, int);
        setBackgroundColor(0, 0, 0);
//**** ask speed *****
        speed=getNumber("Speed [pix/frame]?",10)
//**** prepare stack ****
        stackname="dotanimation"+speed;
        newImage(stackname, "8-bit Black", w, h, frames);
//**** drawing oval in the stack ****
        for(i=0; i<frames; i++) {
                setSlice(i+1);
                makeOval(x_position, y_position, sizenum, sizenum);
                run("Fill", "slice");
                if ((x_position > (w-sizenum)) || (x_position < 0) ) {
                        speed*=-1;
                }
                x_position += speed;
        }
}
```

Sets initial values

Set Drawing / Background Color

User inputs the speed

Prepare Stack to be drawn

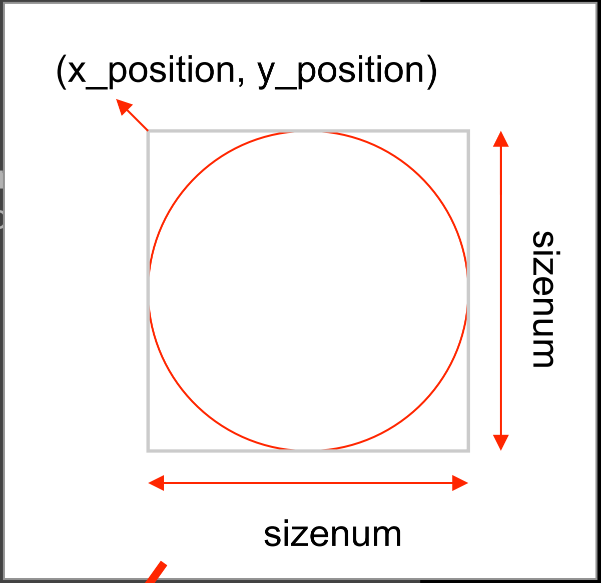
Drawing Dot in each frame (for loop)

“If” statement → switching direction

2-3-4 Conditions: if-else-statement : Application

Code 13

```
macro "Generate Dot Movement back and forth" {
// **** initial values ****
        sizenum=10;                 //dot size
        int=255;                    //dot intensity
        frames=50;                  //frames in stack
        w=200;                      //width of frame
        h=50;                       //height of frame
        x_position = 0;             // x position: changes with speed defi
        y_position= (h/2)-(sizenum/2);          //y positon of the oval top
//**** set colors *****
        setForegroundColor(int, int, int);
        setBackgroundColor(0, 0, 0);
//**** ask speed *****
        speed=getNumber("Speed [pix/frame]?",10)
//**** prepare stack ****
        stackname="dotanimation"+speed;
        newImage(stackname, "8-bit Black", w, h, frames);
//**** drawing oval in the stack ****
        for(i=0; i<frames; i++) {
                setSlice(i+1);
                makeOval(x_position, y_position, sizenum, sizenum);
                run("Fill", "slice");
                x_position += speed;
                if ((x_position > (w-sizenum)
                        speed*=-1;
                }
        }
}
```



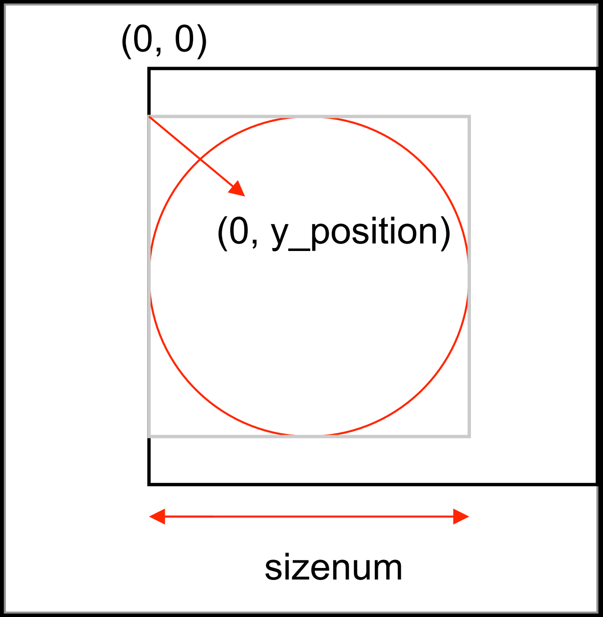
makeOval(x, y, width, height)
Creates an elliptical selection, where (x,y) define the upper left corner of the bounding rectangle of the ellipse.

2-3-4 Conditions: if-else-statement : Application

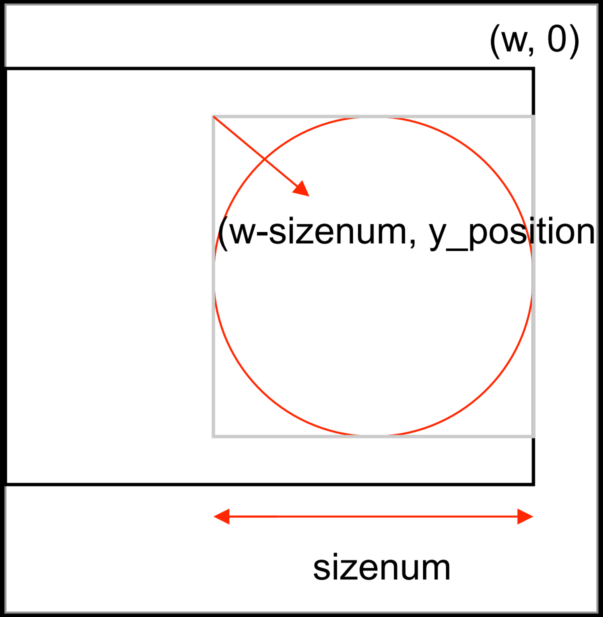
Code 13 (part)

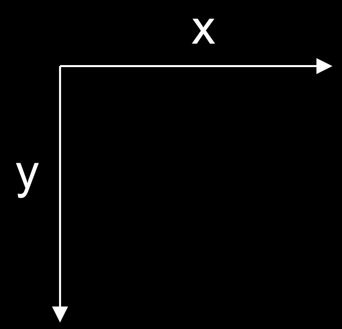
```
x_position += speed;
if ((x_position > (w-sizenum)) || (x_position < 0) ) {
        speed*=-1;
}
```

x_position > (w-sizenum)



x_position < 0





2-3-4 Conditions: if-else-statement : Application

Code 13

Page 20!

```
macro "Generate Dot Movement back and forth" {
// **** initial values ****
          sizenum=10;                    //dot size
          int=255;                       //dot intensity
          frames=50;                     //frames in stack
          w=200;                         //width of frame
          h=50;                          //height of frame
          x_position = 0;                // x position: changes with speed defined later
          y_position= (h/2)-(sizenum/2);           //y positon of the oval top-left corner: constant
//**** set colors *****
          setForegroundColor(int, int, int);
          setBackgroundColor(0, 0, 0);
//**** ask speed *****
          speed=getNumber("Speed [pix/frame]?",10)
//**** prepare stack ****
          stackname="dotanimation"+speed;
          newImage(stackname, "8-bit Black", w, h, frames);
//**** drawing oval in the stack ****
          for(i=0; i<frames; i++) {
                    setSlice(i+1);
                    makeOval(x_position, y_position, sizenum, sizenum);
                    run("Fill", "slice");
                    x_position += speed;
                    if ((x_position > (w-sizenum)) || (x
                              speed*=-1;
                    }
          }
}
```

Exercise 3-4-3-1:

Modify code 13 that the dot moves **up and down vertically**. Change the stack width and height also.

Acknowledgement

Cihan Cetin + Mitocheck Group (EMBL) for providing nucleus dividing sequence.