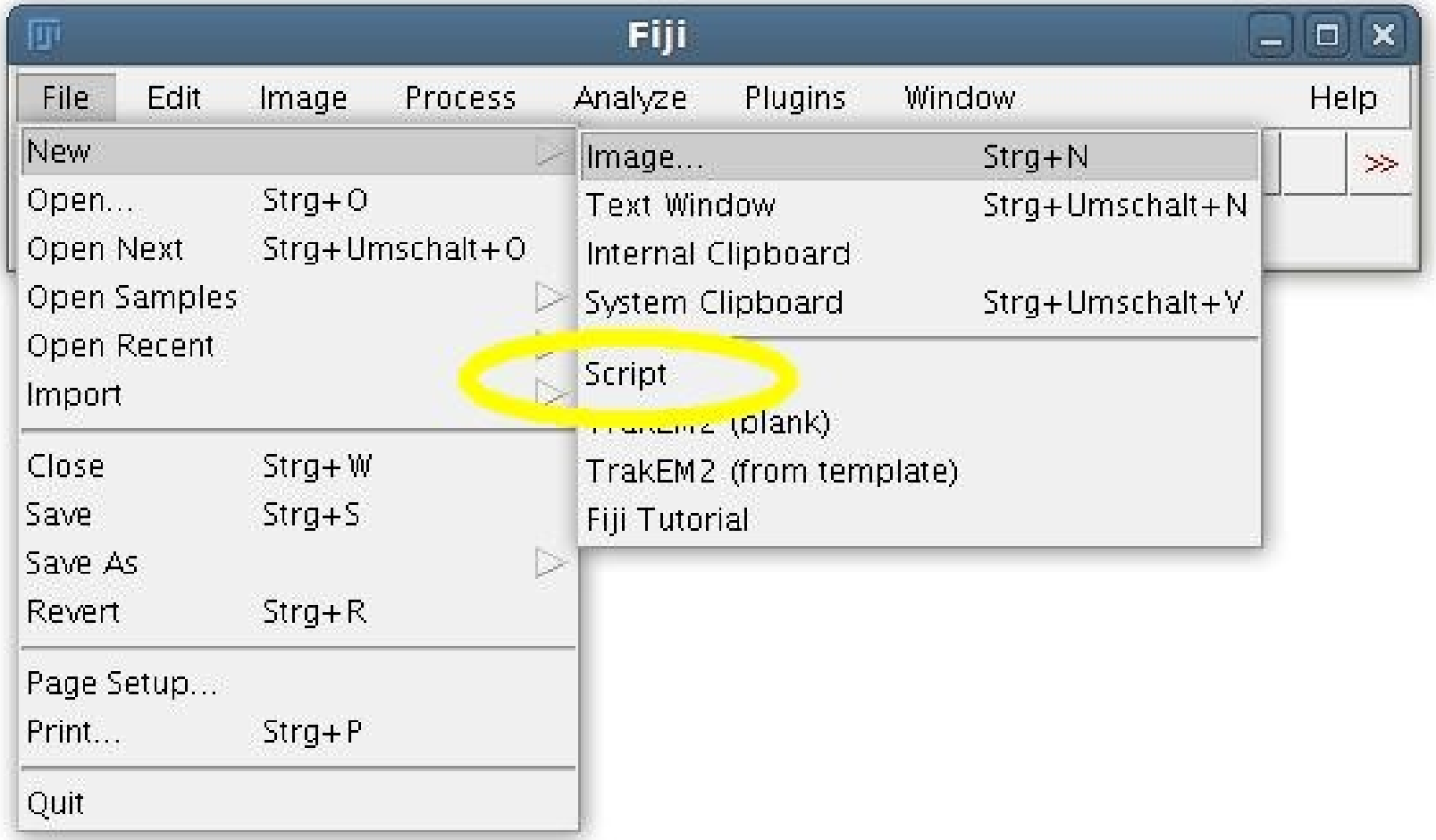
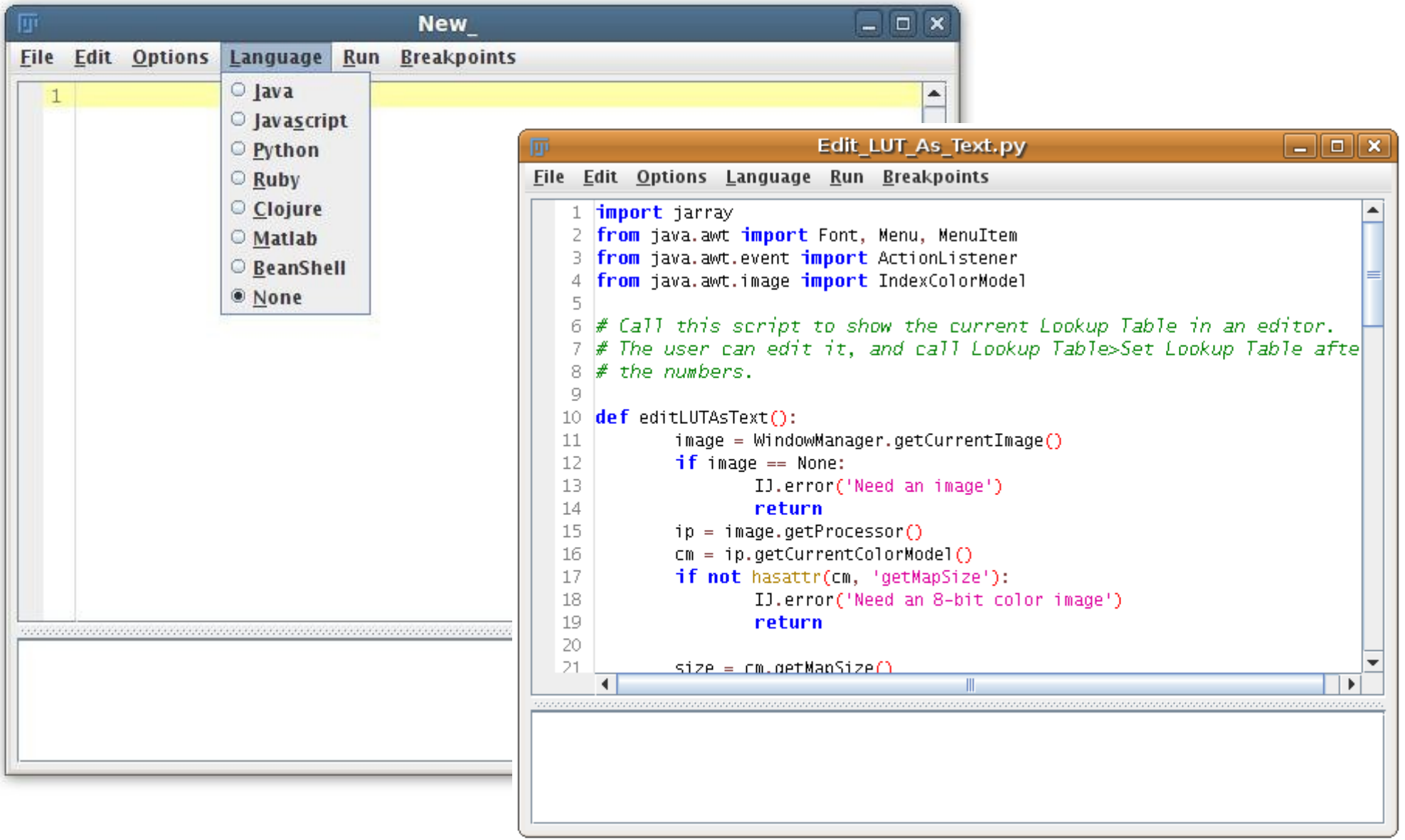
Fiji

Script Editor



Script Editor



Scripting languages

- 

- 

- 

- Javascript 

- BeanShell 

Scripts:

Purpose

- Automation
- Rapid prototyping
- Reproducible workflows
- Share workflows with collaborators
- Develop workflows

Scripts vs Macros

- Scripting languages are open standards (Ruby, Python, Scheme, Javascript, Beanshell)
- Typically, scripting languages run faster than macros
- Scripting languags can use the complete set of functionality like a Java plugin could
- Scripts can run in parallel
- Scripts usually only show results, not intermediate images/result tables
- Some functions available to macros are not available to scripts (or for that matter, plugins)

Scripting in Beanshell

- Beanshell is very similar to the macro language
- Beanshell is “interpreted Java” (but simpler!)
- The builtin functions of the macro language are **not** available in scripting languages!
- However, for many macro functions, there are equivalents of the form *IJ.name()* (e.g. IJ.makeLine())

Beanshell:

Basic concept: Variables

- A variable is a placeholder for a changing entity
- Each variable has a name
- Each variable has a value
- Values can be **any** data type (numeric, text, etc)
- Variables can be assigned new values

Variables:

Setting variables

```
value = 2;

intensity = 255;

title = “Hello, World!”;

text = “title”;

text = title;
```

Variables:

Using variables

```
x = y;

y = x;

x = y * y - 2 * y + 3;

intensity = intensity * 2;
```

Beanshell:

Comments

```
// This is a comment trying to help you to
// remember what you meant to do here:
a = Math.exp(x * Math.sin(y))
   + Math.atan(x * y - a);

// Code can be disabled by commenting it out
// x = y * 2;

/*
 Multi-line comments can be started by a slash
 followed by a star, and their end is marked
 by a star followed by a slash:
*/
```

Beanshell:

Built-in functions

```
// print into the output area of the script editor
print("The title reads " + title);

// mathematical functions are prefixed with Math.
print(Math.exp(1));

// Some functionality provided by the macro
// functions is available via the prefix ij.IJ.
ij.IJ.newImage("My image", "RGB", 640, 480, 1);

// Likewise, the run() macro function:
ij.IJ.run("Gaussian Blur...", "radius=5");
```

Variables:

String concatenation: what is it?
And why do I need it?

```
radius = 3;

print("The radius is radius");
print("The radius is " + number);

// does not work
ij.IJ.run("Gaussian Blur...",
   "radius=radius");

// does work
ij.IJ.run("Gaussian Blur...",
   "radius=" + radius);
```

Classes (first part):

What is ij.IJ?

```
/*
"ij.IJ" is a class name (the prefix "ij" is
called package)

This class provides a large number of
functions similar (but not identical) to the
built-in functions of the macro language.

You can find out what functions, and what they
do, via the Tools>Open Help for Class... menu
entry.
*/

// Importing a class
import ij.IJ;
IJ.run("Gaussian Blur...");
```

Beanshell:

Some useful functions

```
import ij.IJ;

// output to the Log window
IJ.log("Hello, world!”);

// show a message in a dialog window
IJ.showMessage(“Click OK to continue”);

// ask the user for a number
radius = IJ.getNumber(“Radius”);

// ask the user for a text (“string”)
Name = IJ.getString(“Your name”, “Bugs Bunny”);
```

Beanshell:

Basic concept: User-defined functions

```
// Define a function for a recurring task
newBlackImage(title, width, height) {
   // The function body is usually indented for
   // clarity
   IJ.newImage(title, "RGB black",
      width, height, 1);
}

newBlackImage("Tiny", 10, 10);
newBlackImage("Huge", 8000, 8000);
```

Classes (second part):

What are classes?

Classes are “bags of things”. For example, an image consists of numbers for all pixel coordinates and metadata, such as title, dimensions, calibration, etc. These “things” are called “fields”.

Classes can “do a bunch of things”. An image can be hidden, for example. These “things” are called “methods”.

(Functions are special methods: they do not need the bag of things, the so-called *class instance.*)

Most important classes

```
// The most important class is ij.ImagePlus,
// which wraps an image.
image = IJ.getImage();

image.setTitle("New title");

// Selections are instances of the class
// ij.gui.Roi
roi = image.getRoi();
print("The length of the ROI is " +
   roi.getLength());
```

Classes (third part):

Subclasses

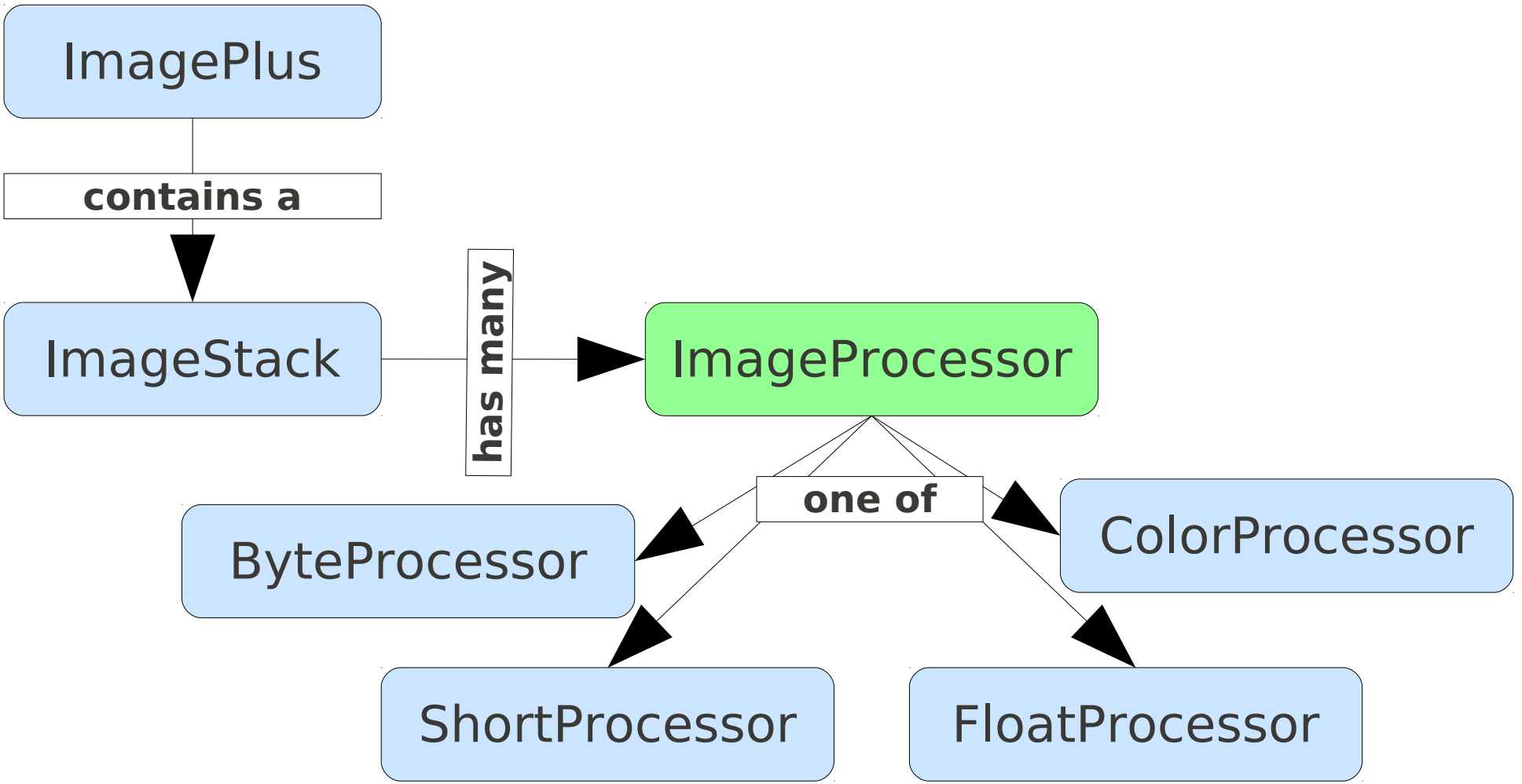
Classes can have *subclasses* which inherit all the fields and methods of the subclassed classes.

Example: all of ij.gui.Line, ij.gui.TextRoi, etc inherit all fields and methods from ij.gui.Roi

“A TextRoi is a Roi” means that a *TextRoi* is a subclass of the class *Roi*.

ImageJ API

The image hierarchy (graphical)



Beanshell:

Advanced example: Hello, World!

```
import ij.IJ;
import ij.gui.TextRoi;

// This creates a new 640x480 color image
image = IJ.createImage("Hi", "RGB black",
   640, 480, 1);
image.show();

// make a selection in the image
text = new TextRoi(50, 300, "Hello, World!");
image.setRoi(text);

// draw it
IJ.run("Draw");
```

Beanshell:

Basic concept: Conditional code blocks

```
// If the image is not binary, abort
if (image.getProcessor().isBinary()) {
   IJ.showMessage("This slice is binary!");
}

// If the code block consists of only one
// statement, the curly braces can be dropped:
if (image.getProcessor().isBinary())
   IJ.showMessage("This slice is binary!");
```

Beanshell:

Basic concept: Loops

```
// Write "Hello, World!" ten times
for (i = 0; i < 10; i++)
   print("Hello, World!");

// As before, if the code block (or "loop body")
// consists of more than one statement, curly
// braces need to be added
for (i = 0; i < 10; i++) {
   // show the progress bar
   IJ.showProgress(i, 10);
   IJ.run("Gaussian Blur...", "radius=" + i);
}
```

Beanshell:

Strict typing

```
// loose typing
i = 0;

// strict typing
int i = 0;

// caveat: this does not work
i = 0;
i = “Hello”;

// however, this will...
j = 0;
j = “Hello”;
```

ImageJ API

The current image

```
// get the current image
image = WindowManager.getCurrentImage();

// get the current slice
ip = image.getProcessor();

// duplicate the slice
ip2 = ip.duplicate();


// show the copied slice
new ImagePlus("copy", ip2).show();
```

ImageJ API

Beyond 3D: hyperstacks

```
// get the n'th slice (1 <= n <= N!)
stack = image.getStack();
size = stack.getSize();
ip = stack.getProcessor(size);

// get the ImageProcessor for a given
// (channel, slice, frame) triple
index = image.getStackIndex(channel,
    slice, frame);
ip = stack.getProcessor(index);
```

ImageJ API

Getting at the pixels

```
// get one pixel's value (slow)
value = ip.getf(0, 0);

// get all type-specific pixels (fast)
// in this example, a ByteProcessor
pixels = ip.getPixels();
w = ip.getWidth();
h = ip.getHeight();
for (j = 0; j < h; j++)
    for (i = 0; i < w; i++)
        handle(pixels[i + w * j] & 0xff);
```

ImageJ API

Getting at the pixels: color

```
// get all pixels of a ColorProcessor
pixels = ip.getPixels();
w = ip.getWidth();
h = ip.getHeight();
for (j = 0; j < h; j++)
    for (i = 0; i < w; i++) {
        value = pixels[i + w * j];
        // value is a bit-packed RGB value
        red = value & 0xff;
        green = (value >> 8) & 0xff;
        blue = (value >> 16) & 0xff;
}
```

ImageJ API

Making a new processor

```
gradient(angle, w, h) {
    c = (float)Math.cos(angle);
    s = (float)Math.sin(angle);
    p = new float[w * h];
    for (j = 0; j < h; j++)
        for (i = 0; i < w; i++)
            p[i + w *j] = (i – w / 2) * c
                + (j – h / 2) * s;
    return new FloatProcessor(w, h, p,
        null);
}
```

ImageJ API
Making a new image

```
// make a gradient
w = 64;
h = 64;
stack = new ImageStack(w, h);
for (i = 0; i < 180; i += 45)
    stack.addSlice("", gradient(i / 180f
        * 2 * Math.PI, w, h));
image = new ImagePlus("stack", stack);
image.show();
```

ImageJ API
Showing the progress

```
// show a progress bar
for (i = 0; i < 100; i++) {
    // do something
    IJ.wait(500);
    IJ.showProgress(i + 1, 100);
}

// show something in the status bar
IJ.showStatus("Hello, world!");
```

ImageJ API
ImageProcessor functions

```
smooth();
sharpen();
findEdges();
```
etc

Tip: use the Script Editor's functions in the *Tools* menu:

- **“Open Help for Class...”**
- **“Open *.java* file for class...”**
- **“Open *.java* file for menu item...”**

ImageJ API
The WindowManager

```
import ij.WindowManager;

ids = WindowManager.getIDList();
for (int id : ids) {
   image = WindowManager.getImage(id);
   print("Image " + id + " has title "
      + image.getTitle());
}

// get non-image windows
frames =
   WindowManager.getNonImageWindows();
```

ImageJ API
Input/Output

```
import ij.IJ;

// open an image
image = IJ.openImage("Hello.tif");

// save an image
IJ.saveAs(image, "JPEG", "Hello.jpg");
```

ImageJ API
Regions of interest (1/2)

```
// rectangular ROI
roi = new Roi(10, 10, 90, 90);
image.setRoi(roi);

// oval ROI
roi = new OvalRoi(10, 10, 90, 90);
image.setRoi(roi);

// testing ROI type
if (roi != null &&
        roi.getType() == Roi.POLYGON)
    IJ.log("This is a polygon!");
```

ImageJ API

Regions of interest (2/2)

```
// get ROI coordinates
showCoordinates(polygon) {
     x = polygon.getXCoordinates();
     y = polygon.getYCoordinates();
     // x, y are relative to the bounds
     bounds = polygon.getBounds();
     for (i = 0; i < x.length; i++)
          IJ.log("point " + i + ": "
               + (x[i] + bounds.x) + ", "
               + (y[i] + bounds.y));
}
```

ImageJ API
Overlays

```
import ij.IJ;
import ij.gui.Overlay;
import ij.gui.TextRoi;

// get the current image
image = IJ.getImage();

// make a new text ROI and an overlay
roi = new TextRoi(10, 50, “Hello...”);
overlay = new Overlay(roi);

// activate the overlay
image.setOverlay(overlay);
```

ImageJ API
The results table

```
rt = Analyzer.getResultsTable();
if (rt == null) {
    rt = new ResultsTable();
    Analyzer.setResultsTable(rt);
}
for (i = 1; i <= 10; i++) {
    rt.incrementCounter();
    rt.addValue("i", i);
    rt.addValue("log", Math.log(i));
}
rt.show("Results");
```

ImageJ API
Plots (1/2)

```
plot(values) {
     x = new double[values.length];
     for (i = 0; i < x.length; i++)
          x[i] = i;
     plot = new Plot(“Plot window”,
          “x”, “values”, x, values);
     plot.show();
}
```

ImageJ API
Plots (2/2)

```
plot(values, v2) {
     x = new double[values.length];
     for (i = 0; i < x.length; i++)
          x[i] = i;
     plot = new Plot("Plot window",
          "x", "values", x, values);
     plot.setColor(Color.RED);
     plot.draw();
     plot.addPoints(x, v2, Plot.LINE);
     plot.show();
}
```

Scripting comparisons
Beanshell

```
import ij.IJ;
rotate(image, angle) {
   IJ.run(image, "Rotate... ", "angle="
     + angle + " interpolation=Bilinear");
}
image = IJ.createImage("Beanshell Example",
   "8-bit Ramp", 100, 100, 1);
image.show();
for (i = 0; i < 360; i+= 45) {
   rotate(image, 45);
   IJ.wait(100);
}
```

Scripting comparisons
ImageJ Macro language

```
function rotate(angle) {
   run("Rotate... ", "angle=" + angle
     + " interpolation=Bilinear");
}

newImage("Macro Example", "8-bit Ramp",
   100, 100, 1);
for (i = 0; i < 360; i+= 45) {
   rotate(45);
   wait(100);
}
```

Scripting comparisons

Javascript

```
importClass(Packages.ij.IJ);
rotate = function(image, angle) {
  IJ.run(image, "Rotate... ", "angle="
    + angle + " interpolation=Bilinear");
}
image = IJ.createImage("Javascript Example",
  "8-bit Ramp", 100, 100, 1);
image.show();
for (i = 0; i < 360; i+= 45) {
  rotate(image, 45);
  IJ.wait(100);
}
```

Scripting comparisons
Jython

```
from ij import IJ
from time import sleep


def rotate(image, angle):
   IJ.run(image, "Rotate... ", "angle="
      + str(angle)
      + " interpolation=Bilinear")
image = IJ.createImage("Jython Example",
   "8-bit Ramp", 100, 100, 1)
image.show()
for i in range(0, 360, 45):
   rotate(image, 45)
   sleep(0.1)
```

Scripting comparisons
Jython

```
include_class 'ij.IJ'


def rotate(image, angle)
  ij.IJ.run(image, "Rotate... ",
  "angle=#{angle} interpolation=Bilinear")
end
image = ij.IJ.createImage("JRuby Example",
  "8-bit Ramp", 100, 100, 1)
image.show()
0.step(360, 45) do |angle|
  rotate(image, 45)
  sleep(0.1)
end
```

Scripting comparisons
Clojure (Scheme)

```
(import '(ij.IJ))
(defn rotate [image, angle]
  (ij.IJ/run image "Rotate... "
   (str "angle=" angle
    "interpolation=Bilinear")))
(let [image (ij.IJ/createImage
  "Clojure Example" "8-bit Ramp" 100 100 1)]
    (.show image)
    (dotimes [angle (/ 360 45)]
      (rotate image 45)
      (ij.IJ/wait 100)))
```

Fiji community:

Mailing lists:

Fiji User list <fiji-user@googlegroups.com>

ImageJ mailing list <imagej@list.nih.gov>

Fiji developer list <fiji-devel@googlegroups.com>

IRC (internet chat):

#fiji-devel on irc.freenode.net

See also http://pacific.mpi-cbg.de/IRC

Documentation, information, tutorials:

http://pacific.mpi-cbg.de/

Thanks!

Max Planck Institute CBG, Dresden
http://www.mpi-cbg.de/

Janelia Farm, Ashburn VA http://janelia.hhmi.org

INI, Zürich http://www.ini.uzh.ch/

EMBL, Heidelberg http://www.embl.de/

LBNL, Berkeley http://www.lbl.gov/

Wayne Rasband, Curtis Rueden, Grant Harris
(ImageJ)

The Fiji Team:

http://pacific.mpi-cbg.de