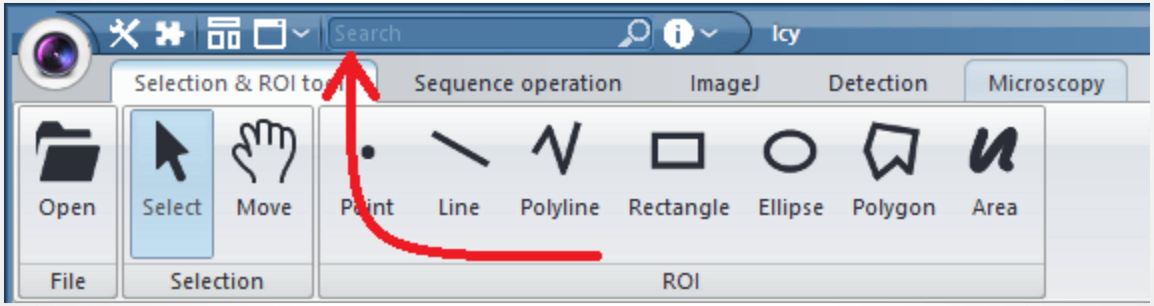
Scripting in Icy

Scripts: the basics

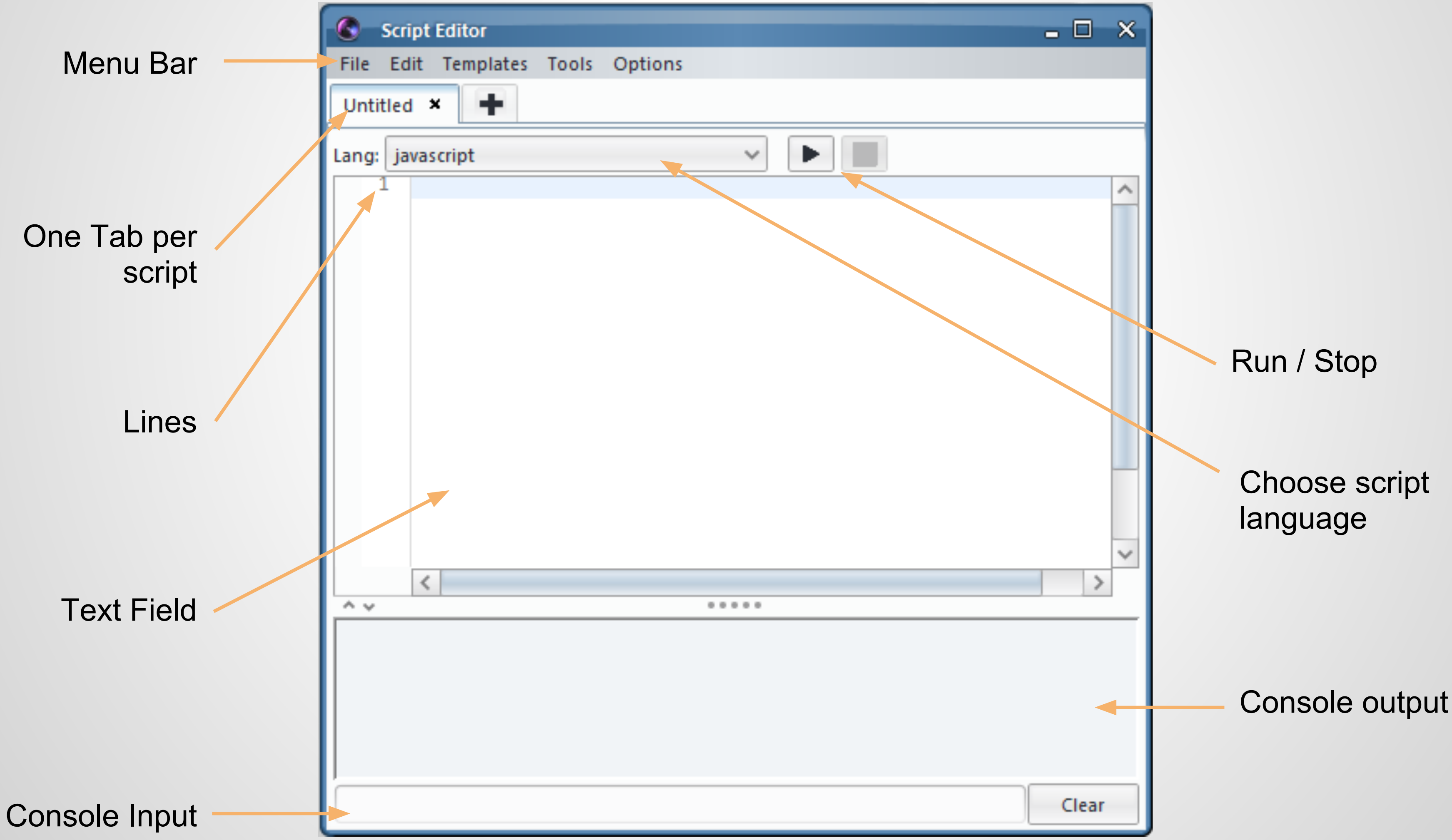
- What is a script?
  - *"A script is a program written for a software environment that automates the execution of tasks. The script benefits from a high level interface, much more accessible."*
  - Scripts have a very simple goal: making research simple and reproducible.
  - Scripts/Macros usually use a specific language. In Icy, we use standard languages such as JavaScript and Python.
  - This lesson will be with JavaScript only.

First Script

- Open the Script Editor plugin:
  - Search it with the SearchBar
  - (Install and) Run it by clicking on it



First Script



First Script

- Write the following line:

```
println("I love cells!")
```

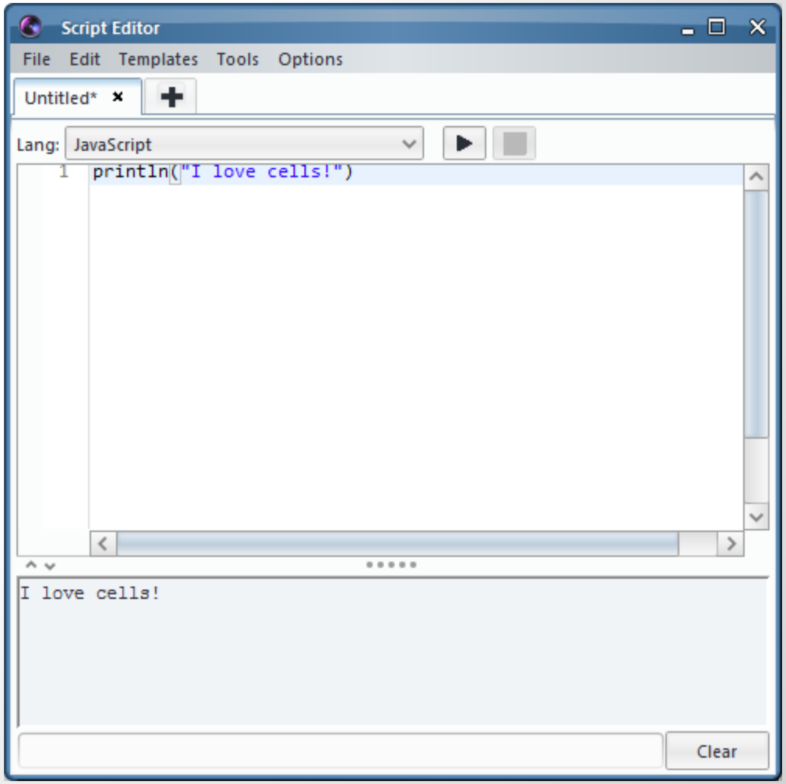
- Hit the  button.

First Script

- Write the following line:

```
println("I love cells!")
```

- Hit the  button.



Predefined features

- Open the sequence: hela-cells.tif

- Predefined methods:
  - Get the current Sequence:
    `seq = getSequence()`

  - Get the current Image:
    `img = getImage()`

- Predefined variables:
  - `gui` **:** represents the interface of Icy

    `gui.addSequence(seq)`

Auto-complete

- How can one discover all methods in Sequence?
- Write :

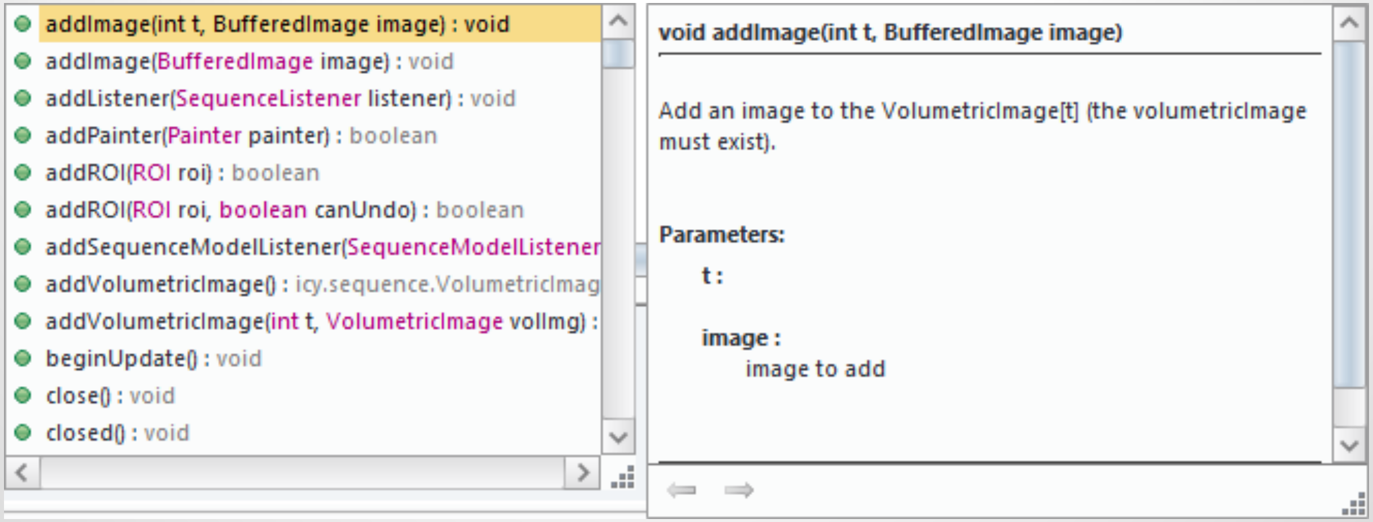
  `seq = getSequence()`
- Then:

  `seq.`
- What happens?

Auto-complete

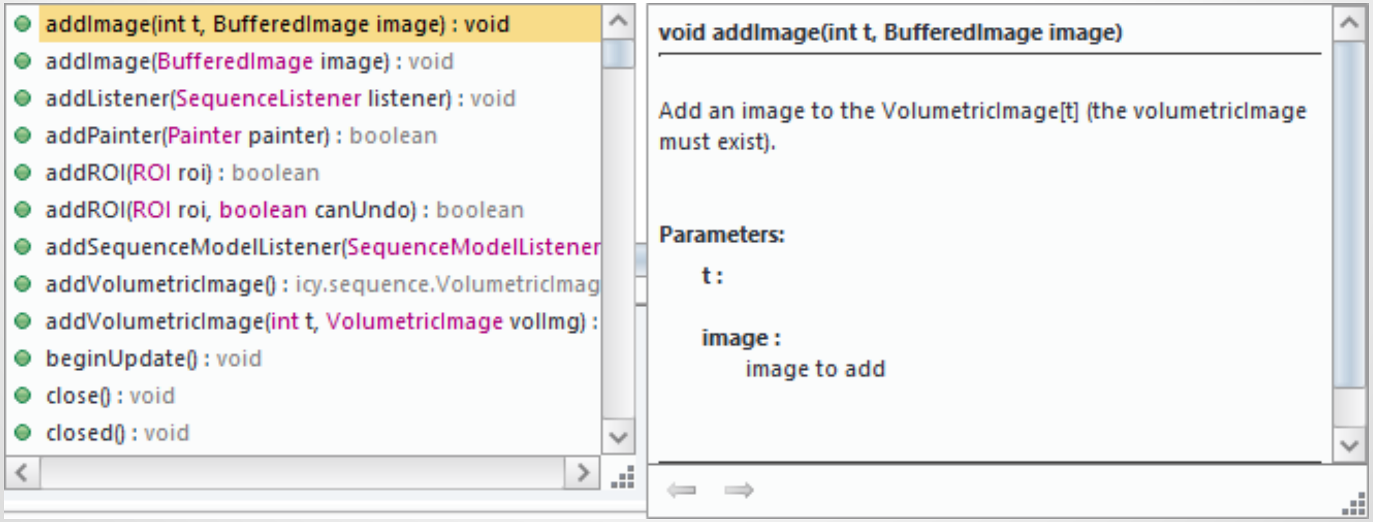
- Look at addImage, there are the two methods we can use:
  - `addImage(int t, BufferedImage image)`
  - `addImage(BufferedImage img)`

- On the right panel, you have more information about the method.



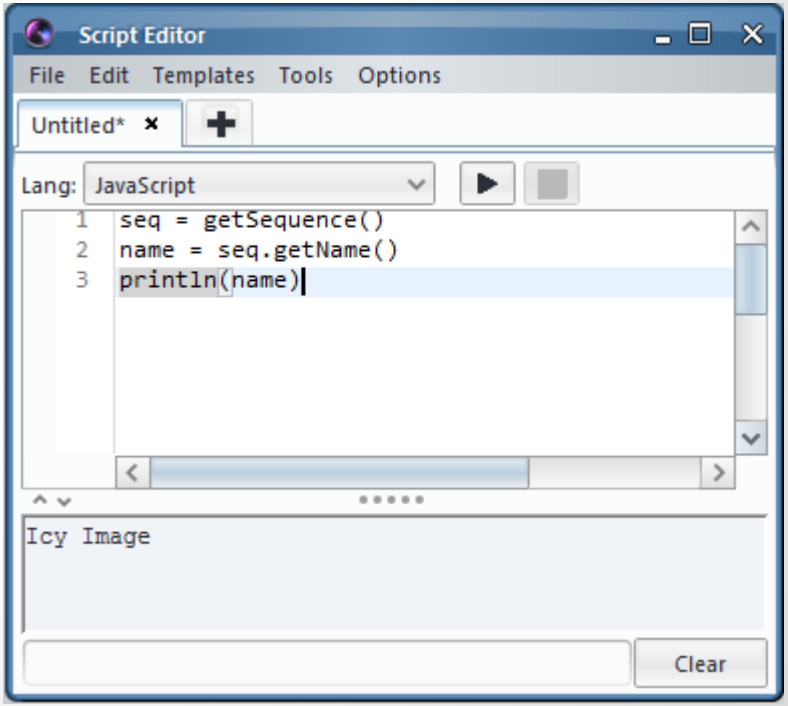
Auto-complete

- Auto-Completion:
  - Know all the methods in a type
  - Get info on the type
  - Get info on the method
  - Get info on the parameters of the method



Auto-complete

- ```
  seq = getSequence()
  name = seq.getName()
  println(name)
  ```

- Displays the name of the sequence.
  ```

Auto-complete

- ```
  seq = getSequence()
  name = seq.getName()
  println(name)
  ```

- Displays the name of the sequence.


  ```

Simple Operations (accessDimensions.js)

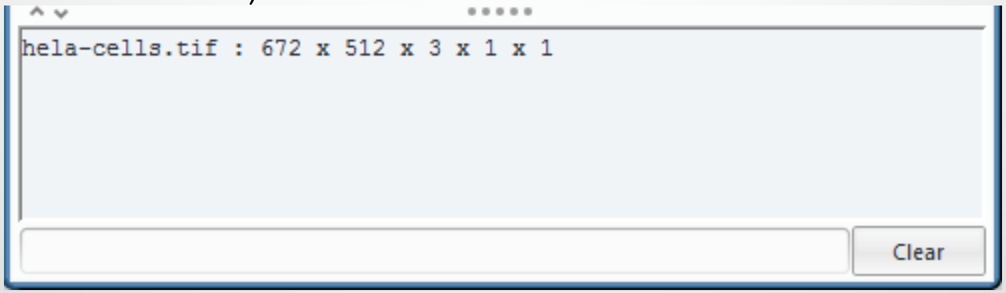
- Get the dimensions of your sequence:

```
seq  = getSequence()

name = seq.getName()
w = seq.getWidth()
h = seq.getHeight()
c = seq.getSizeC()
z = seq.getSizeZ()
t = seq.getSizeT()

println(name + " : " + w + " x " + h + " x " + c + " x
" + z + " x " + t)
```

Simple Operations (accessDimensions.js)

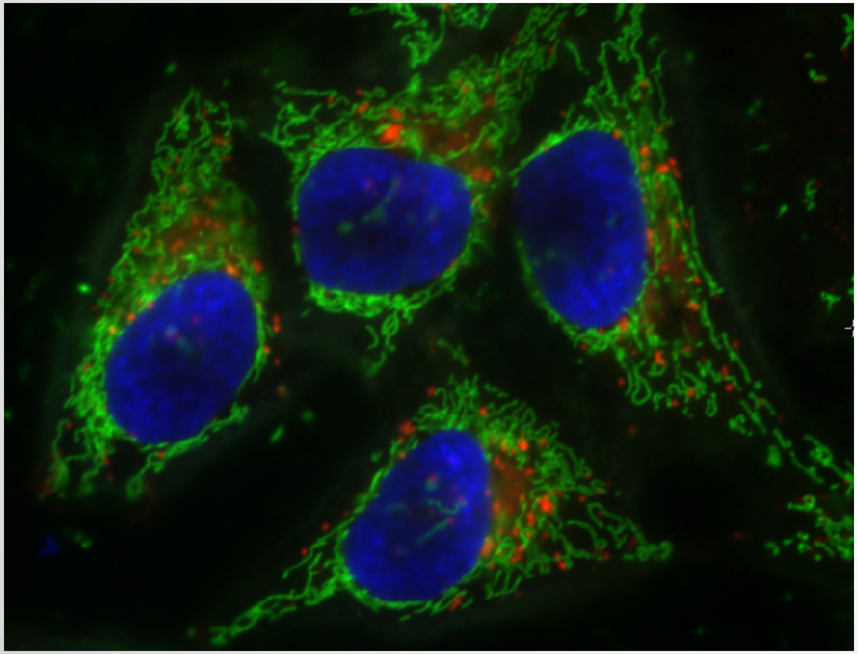
- Get the dimensions of your sequence:

```
seq  = getSequence()

name = seq.getName()
w = seq.getWidth()
h = seq.getHeight()
c = seq.getSizeC()
z = seq.getSizeZ()
t = seq.getSizeT()

println(name + " : " + w + " x " + h + " x " + c + " x " + z + " x " + t)
```



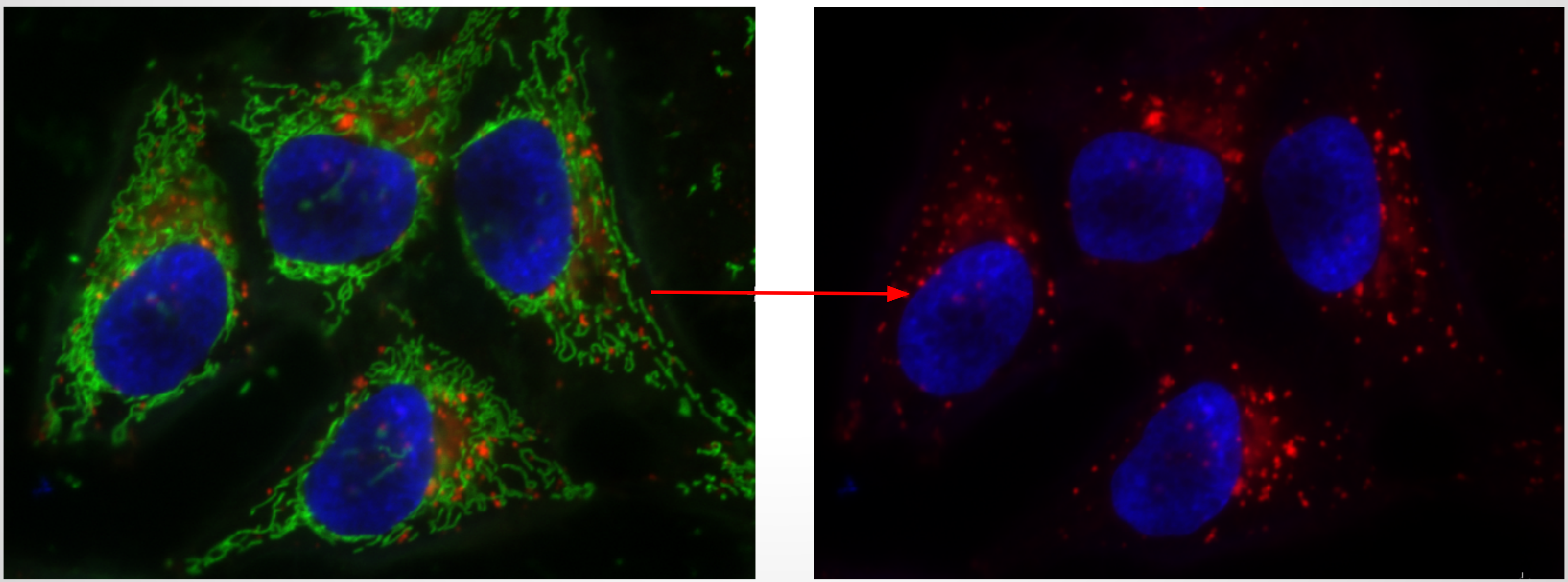
Using Icy tools

- How to remove the green channel?



Using Icy tools

- How to remove the green channel?



Using Icy tools (removeChannel.js)

- All interesting methods for Sequence Operations are stored in the SequenceUtil.

```
seq  = getSequence()

SequenceUtil.removeChannel(seq, 1)
```

- Note: the index always starts at "zero" and not "one".

Creating ROIs (generateROIs.js)

- ROI2Ds coordinates are based on Point2Ds
- Point2D is not an Icy or a plugin type, it is from Java. Thus, we do not provide auto-import (yet!).
- You have to import it manually:
  `importClass(Packages.java.awt.geom.Point2D)`

Creating ROIs (generateROIs.js)

- Ellipse ROI Creation:

```
importClass(Packages.java.awt.geom.Point2D)
importClass(Packages.icy.roi.ROI2DEllipse)

seq = getSequence()

topLeft = new Point2D.Double(100, 100)
bottomRight = new Point2D.Double(200, 200)

roi = new ROI2DEllipse(topLeft, bottomRight)
seq.addROI(roi)
```

Creating ROIs (generateROIs.js)

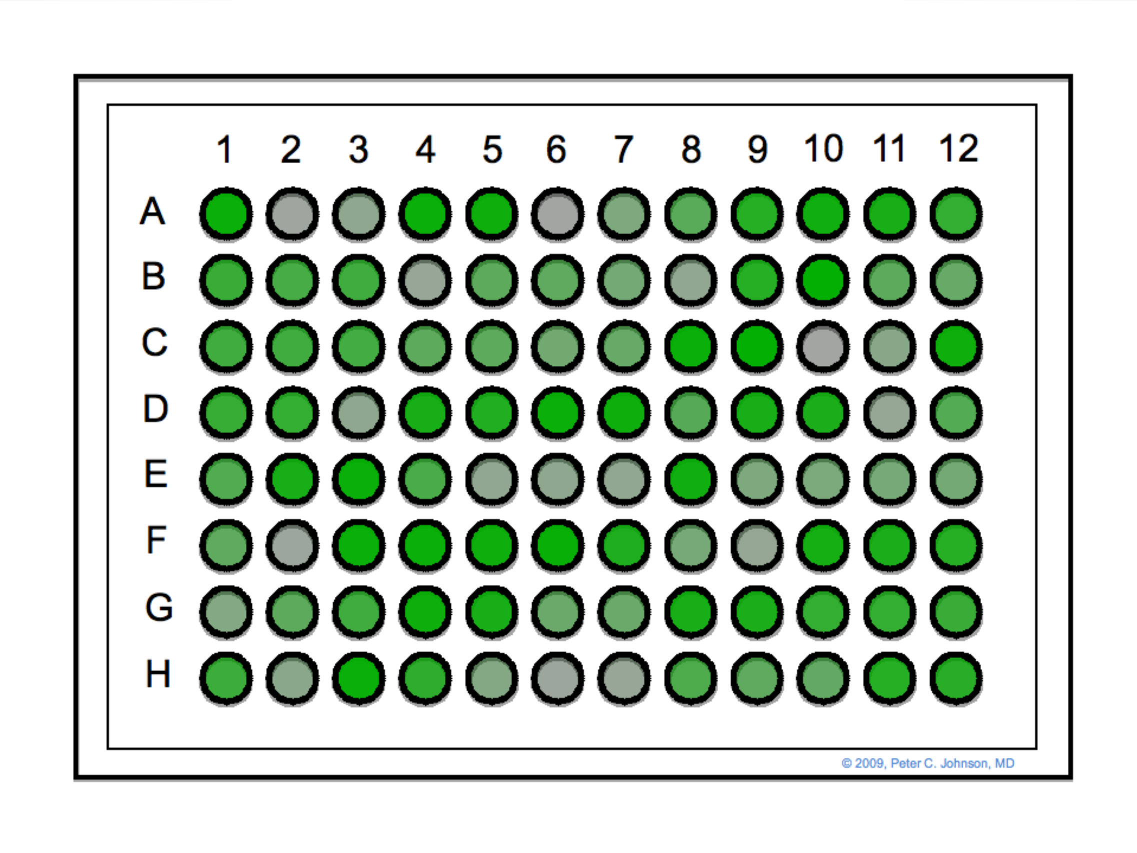
- Ellipse ROI Creation:

```
importClass(Packages.java.awt.geom.Point2D)
importClass(Packages.icy.roi.ROI2DEllipse)

seq = getSequence()

topLeft = new Point2D.Double(100, 100)
bottomRight = new Point2D.Double(200, 200)

roi = new ROI2DEllipse(topLeft, bottomRight)
seq.addROI(roi)
```

- `"new"` means creation

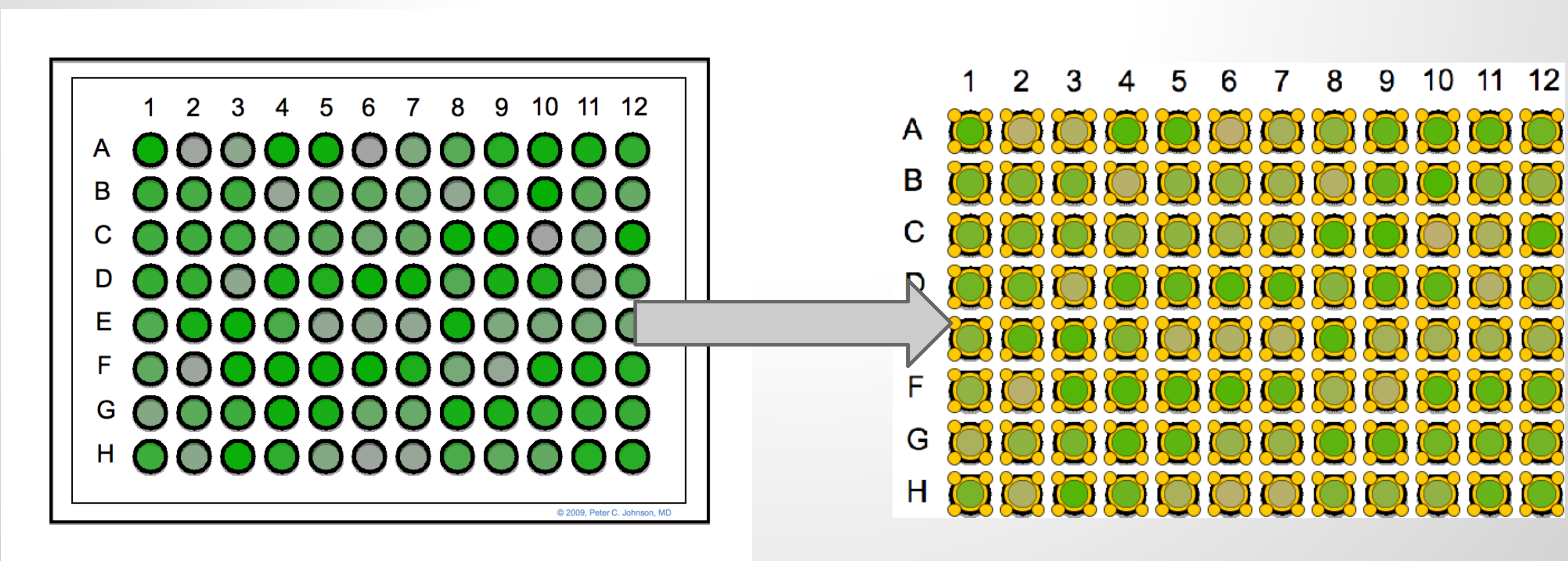
Calculating mean intensity in wells

- Open ElisaRedux_finals.png



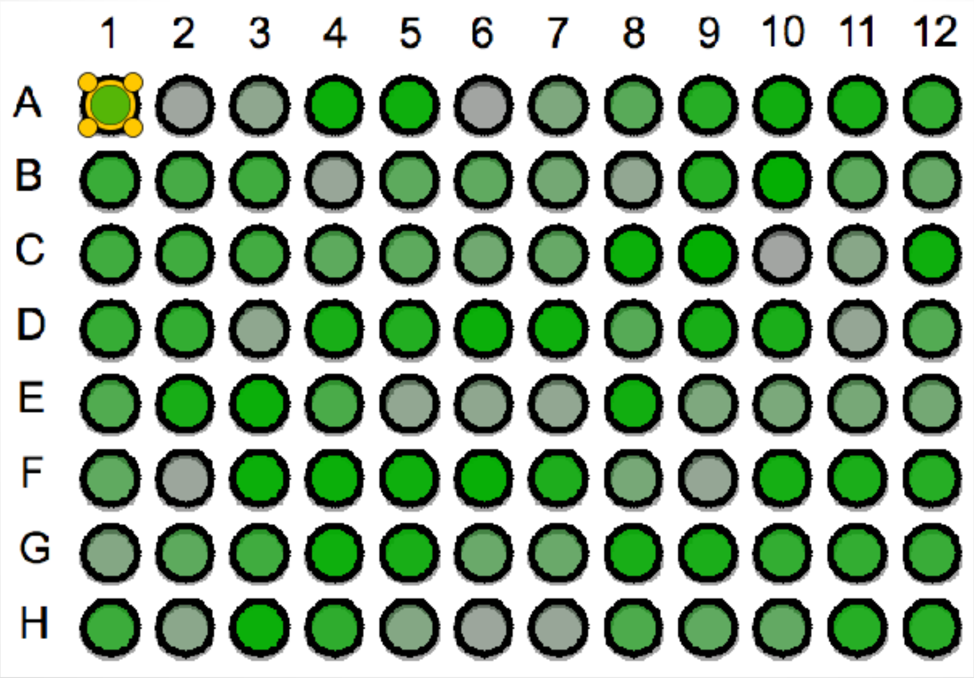
Calculating mean intensity in wells

- Goal:
  a. Represent each well with an ROI ellipse
     i. Create one ellipse on the topleft
     ii. Create an ellipse for each well
  b. Computes the mean intensity per ROI and display it



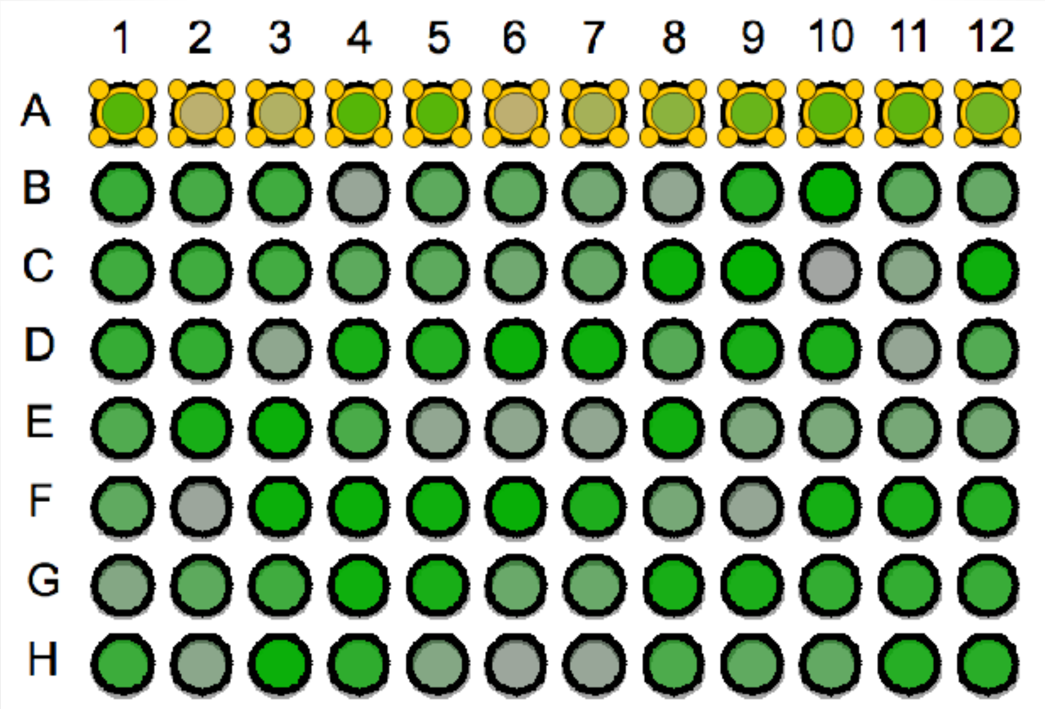
Calculating mean intensity in wells
(ROI_Oval_Grid.js)

1. Create an Ellipse on the first well
   a. Look at the position x / y of the center of the first well (bottom of the viewer)
   b. Use the ruler helper to find the the size of a well...
   c. ...and the space between two wells.
   d. Add ROI to the sequence



Calculating mean intensity in wells
(ROI_Oval_Grid2.js)

1. Create an Ellipse on the first well
2. Create ellipses over the whole first line
   a. Define xa,ya and xb,yb, points of topLeft and topRight of the first circle
   b. Create a loop going from 0 to 11



A loop?

- Creating a second Ellipse:
  a. Copy the code of the first one
  b. Change a few parameters: topLeft & bottomRight

A loop?

- Creating a second Ellipse:
    a. Copy the code of the first one
    b. Change a few parameters: topLeft & bottomRight

- Create a third Ellipse:
    a. Copy the code of the first/second one
    b. Change a few parameters: topLeft & bottomRight

A loop?

- Creating a second Ellipse:
    a. Copy the code of the first one
    b. Change a few parameters: topLeft & bottomRight

- Create a third Ellipse:
    a. Copy the code of the first/second one
    b. Change a few parameters: topLeft & bottomRight

- Create 96 Ellipses: loose a huge amount of time.

A loop?

- Creating a second Ellipse:
  a. Copy the code of the first one
  b. Change a few parameters: topLeft & bottomRight

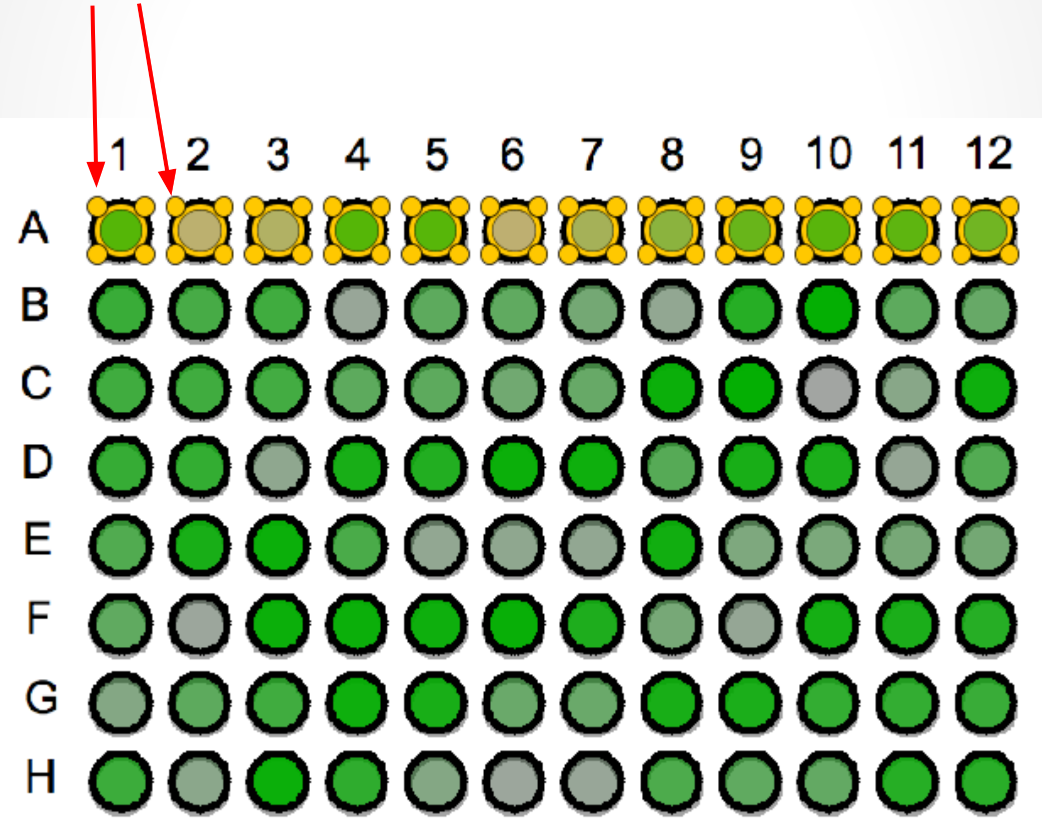
- Create a third Ellipse:
  a. Copy the code of the first/second one
  b. Change a few parameters: topLeft & bottomRight

- Create 96 Ellipses: lose a huge amount of time.

- A loop is a tool that:
  a. Repeats code
  b. Change one variable/parameter at a time

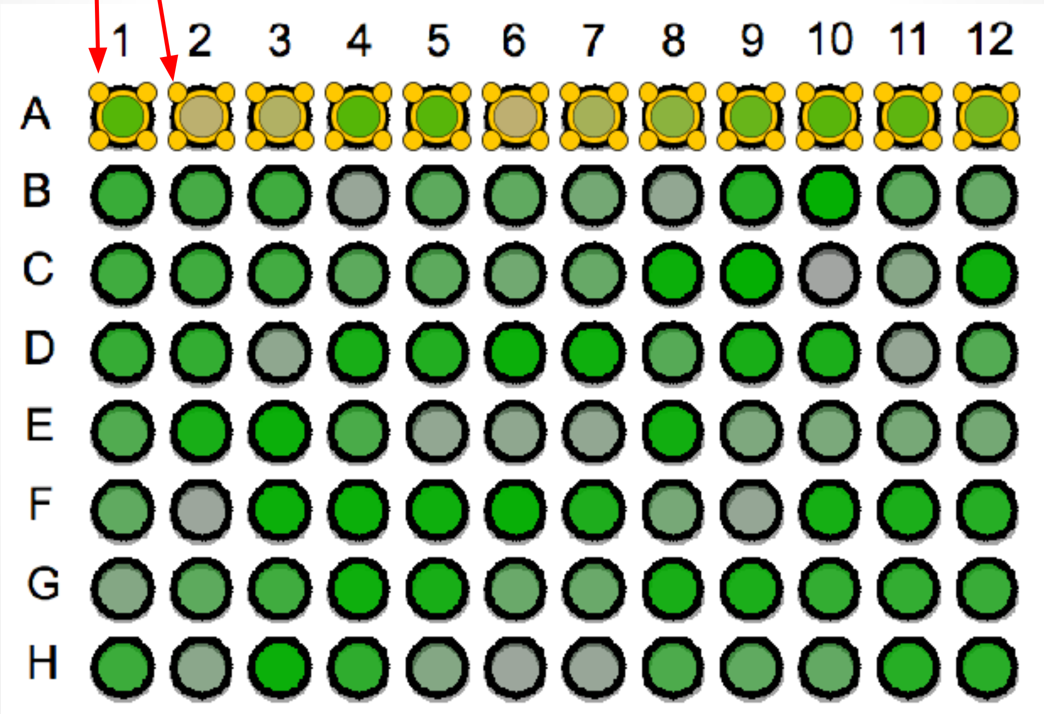
A loop?

1. Analyze the changing parameter(s)
    a. Only x changes, not the size, nor the y



A loop?

1. Analyze the changing parameter(s)
    a. Only x changes, not the size, nor the y
    b. Multiplying the x by the "space" should do it:



A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
i = 0 // well position, starting at zero
while(i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
i = 0 // well position, starting at zero
while(i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

- How many loops are we going to do?

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
i = 0 // well position, starting at zero
while(i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

- How many loops are we going to do? 12.
- When you know this number, there is another tool called ***for loop.***

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

```
i = 0 // well position, starting at zero
while(i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

initialization: only once

```
i = 0 // well position, starting at zero
while(i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

Stop condition: tested every time the block is over. When false, stops.

```
i = 0 // well position, starting at zero
while (i < 12) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

Afterwards: do something every time the block is over

```
i = 0 // well position, starting at zero
while(i <= 11) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
    i = i + 1
}
```

A loop?

1. Analyze the changing parameter(s)
   a. Only x changes
   b. Multiplying the x by the "space" should do it:

```
for (i = 0; i < 12; i = i + 1) {
    xi = x + space * i
    println("i: " + i + " = " + xi)
}
```

   c. Usually:
      i. initialize : creation of the variable with a value
      ii. stop condition : variable < size
      iii. Increase value : variable = variable + 1

A loop!

1. Analyze the changing parameter(s)

2. Create xa/ya and xb/yb, where:
   a. A is the topLeft point
   b. B is the bottomRight point

```
xa = 191 - size / 2
ya = 180 - size / 2
xb = 191 + size / 2
yb = 180 + size / 2

for (i = 0; i < 12; i = i + 1) {

}
```

A loop!

1. Analyze the changing parameter(s)

2. Create xa/ya and xb/yb

3. Create xai/yai and xbi/ybi, where:
    a. Ai is the topLeft point of the ellipse of index i
    b. Bi is the bottomRight point of the ellipse of index i

A loop!

1. Analyze the changing parameter(s)

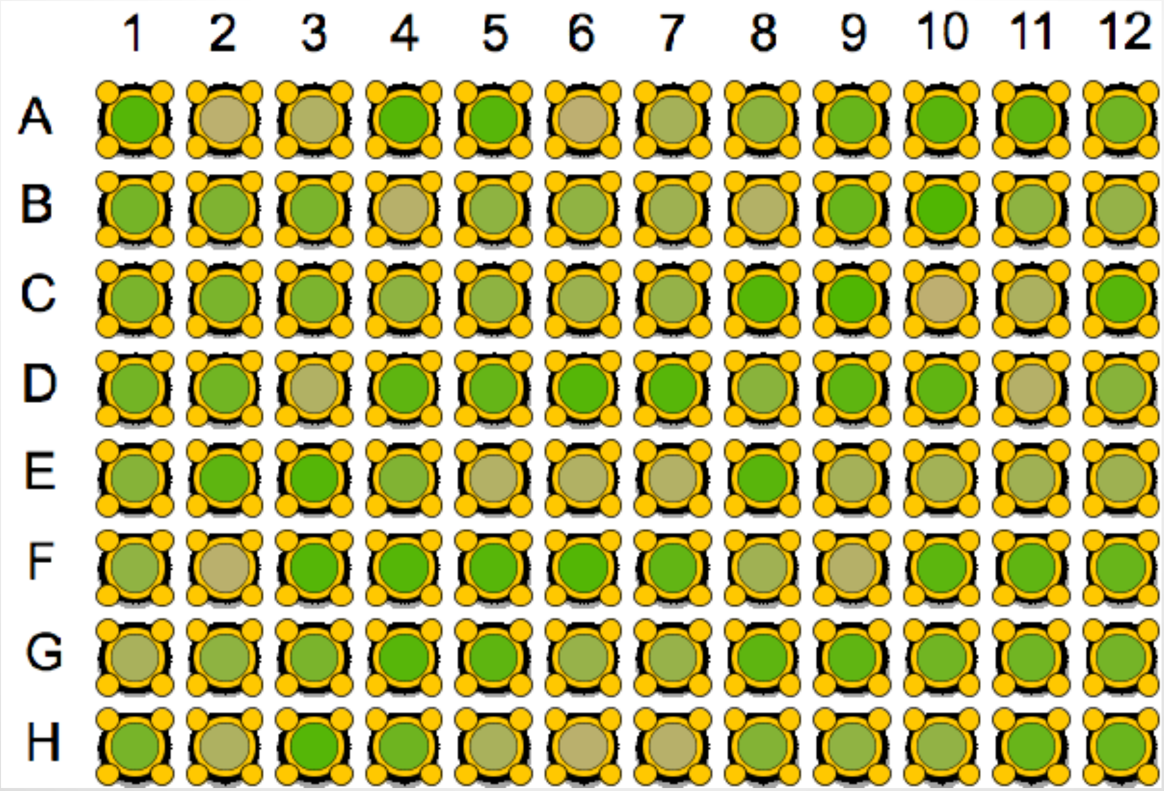
2. Create xa/ya and xb/yb

3. Create xai/yai and xbi/ybi, where:
   a. Ai is the topLeft point of the ellipse of index i
   b. Bi is the bottomRight point of the ellipse of index i

```
for (i = 0; i < 12; i = i + 1) {
    xai = xa + space * i
    yai = ya
    xbi = xb + space * i
    ybi = yb
}
```

Calculating mean intensity in wells
(ROI_Oval_Grid3.js)

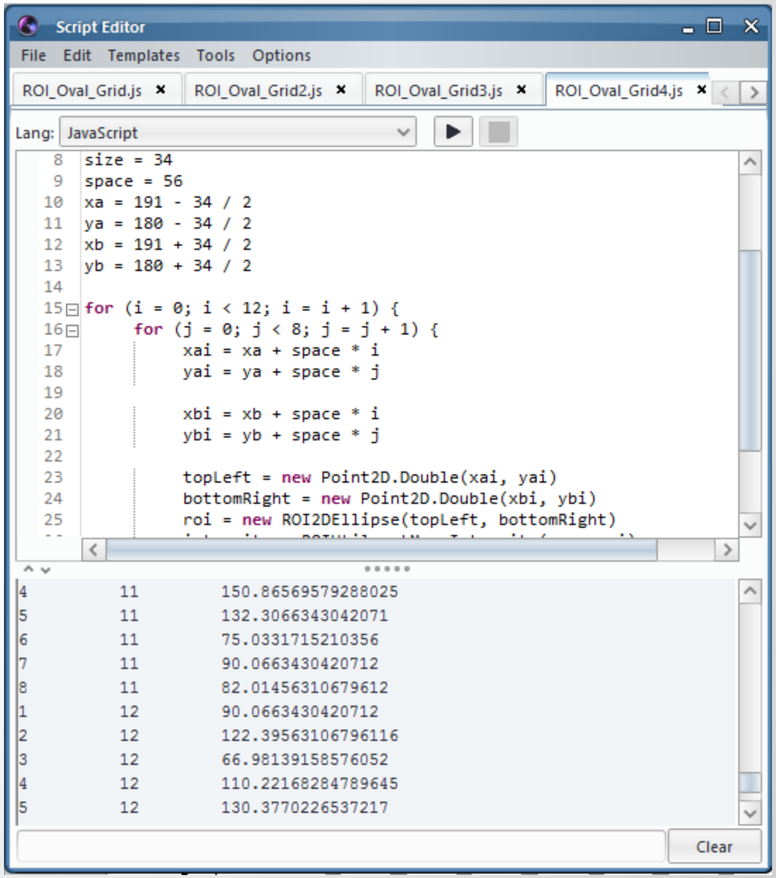
1. Create an Ellipse on the first well
2. Create ellipses over the whole first line
3. Create ellipses over the whole well plate



Calculating mean intensity in wells

(ROI_Oval_Grid4.js)

1. Create an Ellipse on the first well
2. Create ellipses over the whole first line
3. Create ellipses over the whole well plate
4. Computes the mean intensity per well
   a. Use ROIUtil
   b. Display the result



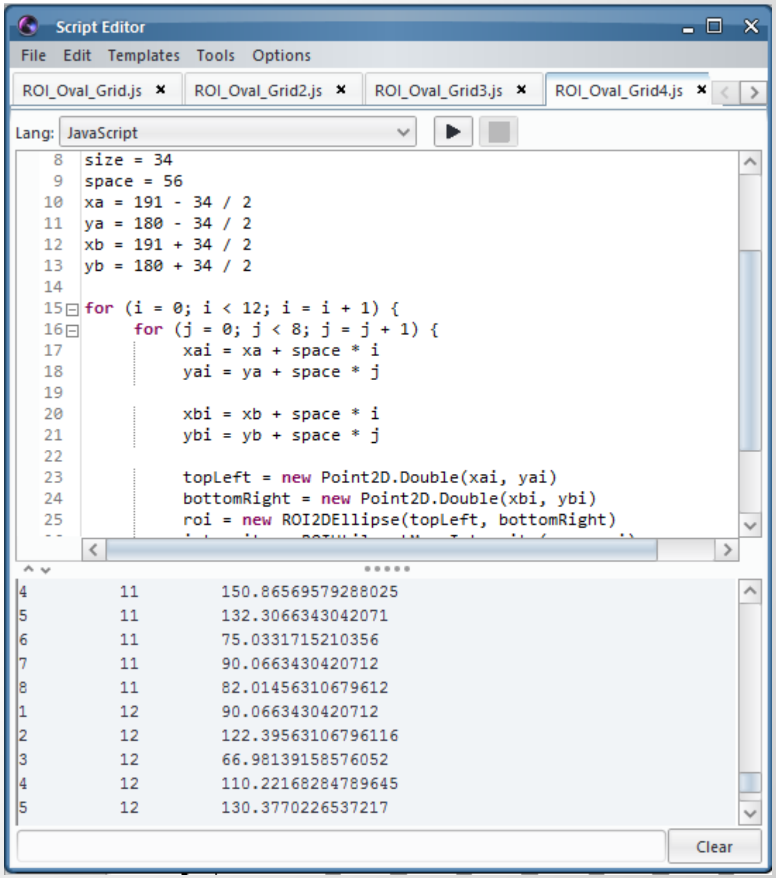
Calculating mean intensity in wells

(ROI_Oval_Grid4.js)

1. Create an Ellipse on the first well
2. Create ellipses over the whole first line
3. Create ellipses over the whole well plate
4. Computes the mean intensity per well
   a. Use ROIUtil

      ```
      RoiUtil.getMeanIntensity(seq,roi)
      ```

   b. Display the result



Calculating mean intensity in wells

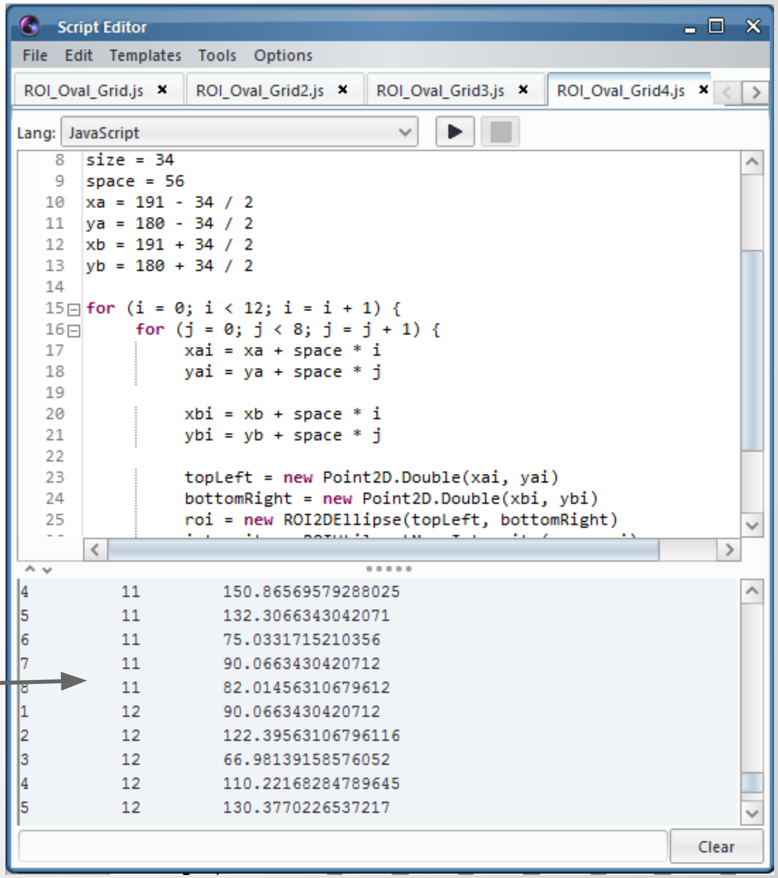
(ROI_Oval_Grid4.js)

1. Create an Ellipse on the first well
2. Create ellipses over the whole first line
3. Create ellipses over the whole well plate
4. Computes the mean intensity per well
   a. Use ROIUtil

      `RoiUtil.getMeanIntensity(seq,roi)`

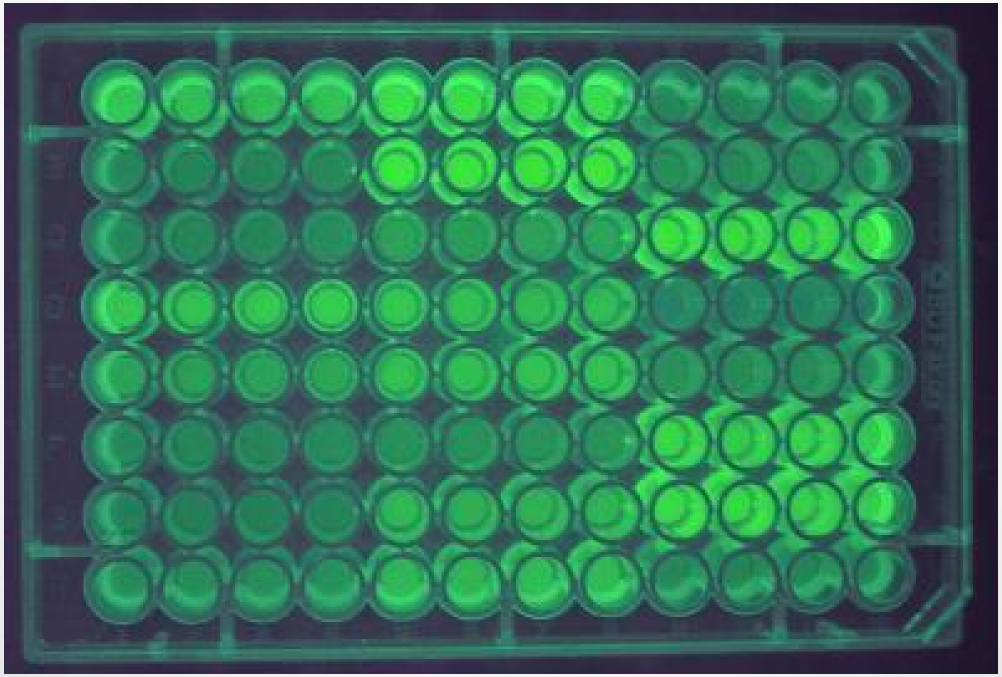
   b. Display the result

Usable in Excel!



Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"



Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image:

```
size = 22
space = 28
xa = 46 - size / 2
ya = 39 - size / 2
xb = 46 + size / 2
yb = 39 + size / 2
```

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image

3. Add a test on the intensity:
   "If my mean intensity is lower than 100, create an 'X' overlay over the image on the well."

```
if (intensity < 100) {

}
```

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image

3. Add a test on the intensity:
   a. If the intensity of a well is lower than 100, display an "X" overlay over the image on the well
   b. Then display the same previous text than before, with "Discarded"
   c. If the well is normal, simply display the previous text

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image

3. Add a test on the intensity:
"If my mean intensity is lower than 100, create an 'X' overlay over the image on the well."

```
if (intensity < 100) {
    font = new Font("Arial", Font.BOLD, 14)
    Note.createNote("X", seq, xai + 5, yai, font, Color.RED)
}
```

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image

3. Add a test on the intensity:
   "If my mean intensity is lower than 100, create an 'X' overlay over the image on the well."

```
if (intensity < 100) {
    font = new Font("Arial", Font.BOLD, 14)
    Note.createNote("X", seq, xai + 5, yai, font, Color.RED)
    println(....... + "\tDiscarded")
}
```

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"

2. Adapt the script for this image

3. Add a test on the intensity:
   "If my mean intensity is lower than 100, create an 'X' on the image on the well."

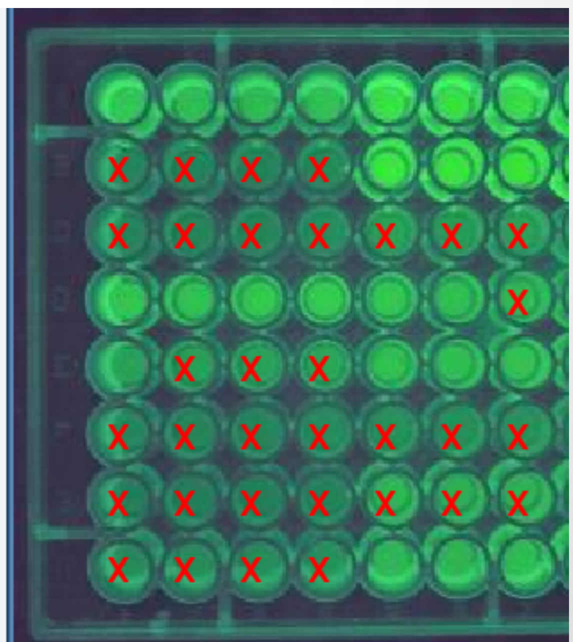
4. Do something else when test did not succeed:
   "Else, normally display the result"

```
else {
   println("" + (j + 1) + "\t" + (i + 1) + "\t" +  intensity)
}
```

Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"
2. Adapt the script for this image
3. Add a test on the intensity
4. Do something else when test did not succeed
5. Result:

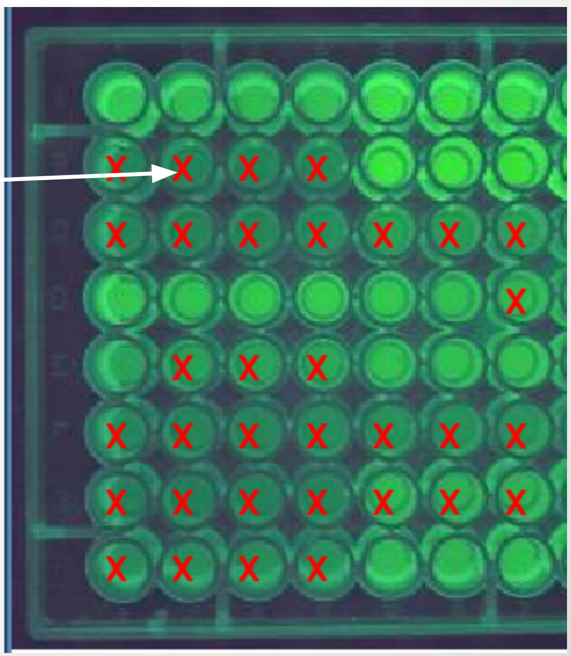
1	9	93.38253012048193	Discarded
1	10	91.74899598393574	Discarded
1	11	90.04819277108433	Discarded
1	12	85.50200803212851	Discarded
2	1	94.32329317269077	Discarded
2	2	87.0160642570281	Discarded
2	3	86.61244979919678	Discarded
2	4	88.9066265060241	Discarded
2	5	110.4578313253012	
2	6	108.62048192771084	
2	7	108.13755020080322	
2	8	109.09136546184739	
2	9	89.71184738955823	Discarded
2	10	88.06124497991968	Discarded
2	11	88.15763052208835	Discarded
2	12	88.41265060240964	Discarded
3	1	95.26807228915662	Discarded
3	2	87.67168674698796	Discarded
3	3	87.74397590361446	Discarded
3	4	91.44578313253012	Discarded
3	5	95	Discarded



Going further... (ROI_Oval_Grid5.js)

1. Open "well96real.jpg"
2. Adapt the script for this image
3. Add a test on the intensity
4. Do something else when test did not succeed
5. Result:

1	9	93.38253012048193	Discarded
1	10	91.74899598393574	Discarded
1	11	90.04819277108433	Discarded
1	12	85.50200803212851	Discarded
2	1	94.32329317269077	Discarded
2	2	87.0160642570281	Discarded
2	3	86.61244979919678	Discarded
2	4	88.9066265060241	Discarded
2	5	110.4578313253012	
2	6	108.62048192771084	
2	7	108.13755020080322	
2	8	109.09136546184739	
2	9	89.71184738955823	Discarded
2	10	88.06124497991968	Discarded
2	11	88.15763052208835	Discarded
2	12	88.41265060240964	Discarded
3	1	95.26807228915662	Discarded
3	2	87.67168674698796	Discarded
3	3	87.74397590361446	Discarded
3	4	91.44578313253012	Discarded
3	5	95	Discarded



Use ImageJ macros (callIJMacro.js)

- Open a file containing the macro: `FileDialog.open()`
- Convert the sequence into an ImagePlus:
  `imPlus = ImageJUtil.convertToImageJImage(seq, null)`
- Run the macro:
  `IJ.runMacroFile(file)`
- Get the result back to Icy
  `seqResult = ImageJUtil.convertToIcySequence(imPlus, null)`

Find scripts

- Scripts are available on the website: http://icy.bioimageanalysis.org/script/list
- You can download them:
  - Directly from the website
  - By using the Search Bar (coming soon!)
- Add your future scripts on the website, and share them!

Programming notions:

- A variable is a container, it associates a name with a value.

  E.g.: The variable ***myvariable*** contains ***10.2***

- This value can be anything: a number, a sequence, an image, etc. To set a value, use the operator **=**

  E.g.: myvariable = 10.2

- This value can change during the execution of the script.

  E.g.:
  myvariable = 10.2
  myvariable = 4

Programming notions:

- `=` is different from `==`
  - `=` : assignment
  - `==` : equality test

- More tests:
  - `!=` : non equal / different
  - `>` : superior
  - `>=` : superior or equal
  - `<` : inferior
  - `<=` : inferior or equal

- `null` : keyword used for non existence:

```
if (seq == null) {

}
```

Programming notions:

- `=` is different from `==`
  - `=` : assignment
  - `==` : equality test

- More tests: Equal is always behind
  - `!=` : non equal / different
  - `>` : superior
  - `>=` : superior or equal
  - `<` : inferior
  - `<=` : inferior or equal

- `null` : keyword used for non existence:

```
if (seq == null) {

}
```

Programming notions:

- `throw` : stops the script. Usually used after an "if" to avoid bugs.

```
if (seq == null) {
    throw "No sequence opened, please open one first."
}
```

- Boolean type:

```
isComputing = true
```

- Tests on booleans:
  - `if (isComputing == true) { ... }`
  - `if (isComputing) { ... }`

  Same: tests if true

  - `if (isComputing == false) { ... }`
  - `if (!isComputing) { ... }`

  Same: tests if false

Programming notions:

- Increasing the value of a variable (Incrementation):
  - `i = i + 1`
  - `i += 1`
  - `i++`

Always increase by 1 the value of `i` !

Programming notions:

- Creating your own function:

```
function hello(a, b) {
    println("Hello " + a + " and " + b + "!")
}
```

- Created functions are displayed in the Autocomplete.