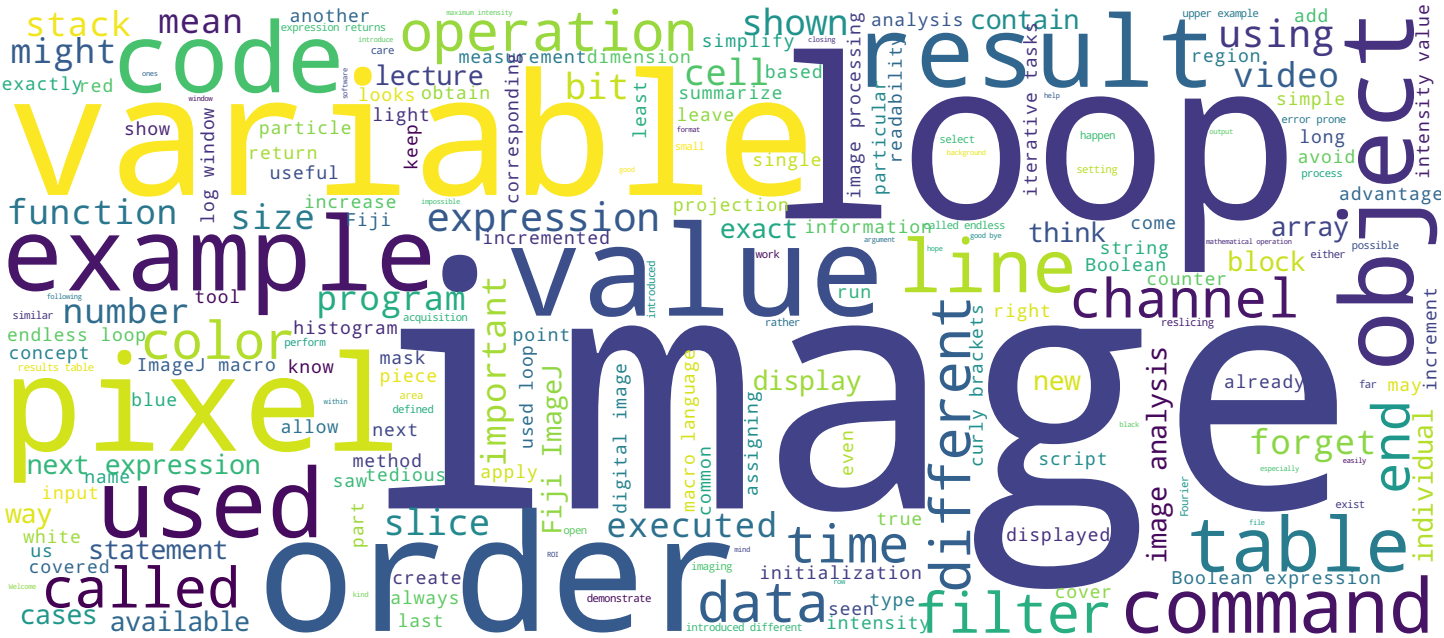
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Loops

Image Processing & Analysis for Life Scientists

Olivier Burri, Romain Guiet & Arne Seitz



Search MOOC

Video

EPFL

Outlook

- Loops
  - For …next statement
  - While statement
  - Do…while statement
- Implementation ImageJ macro language

Welcome to this lecture, in which I will introduce loops and how they are implemented in the Fiji/ImageJ macro language. Loops are extremely helpful for iterative tasks. In image processing, they can be used in order to do a certain mathematical operation on each and every pixel of an image. I think it is obvious that it would be much too tedious to do this by addressing the pixels individually.

Notes

Summary

0m 05s

Loops

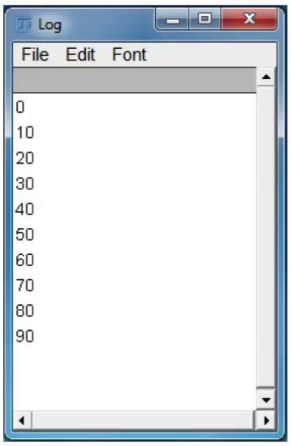
For…next statement

for (initialization; condition; increment) { statement(s) }

- Repetitive tasks
  - For …next statement

```
for (i=0; i<10; i++) {
      j = 10*i;
      print(j);
}
```

Increment: i++ i=i+1 i+=1



The most commonly used loop are the so called 'for next statement'. It consists of a block which contains the initialization of a variable, a condition in most of the cases so called 'stop condition' and an increment expression. In our case the variable i is set to 0. The second expression returns a Boolean. The loop is executed as long as this expression is true. Last but not least, we define how the variable i is incremented. The cryptic expression i++ means that the variable i is incremented with 1. Alternatively, you could also use the expression i=i+1. However, this expression drives every mathematician crazy, and programmers prefer short expressions. Therefore it is rarely used. Sometimes you may also find i+=1. All of them are accepted in the Fiji/ImageJ macro language and they are all doing the same: incrementing the variable i by one. The code to be executed is typically wrapped by curly brackets. This is useful as it allows loops with more than one line. It is also common to indent these lines. This increases the readability of the code. So let's have a look at what these 3 lines return when executed. Inside the loop the variable j is calculated by multiplying the variable i with ten.

Notes

Summary

0m 33s

Loops

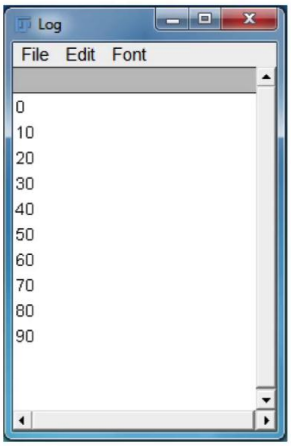
For…next statement

for (initialization; condition; increment) { statement(s) }

- Repetitive tasks
  - For …next statement

```
for (i=0; i<10; i++) {
      j = 10*i;
      print(j);
}
```

Increment: i++ i=i+1 i+=1



The variable j is then displayed in the log-window by the print command. So we are obtaining 10 different numbers from 0 to 90. Once i = 10, the Boolean expression returns the value false, and the loop is no longer executed.

Notes

Summary

2m 15s

Loops

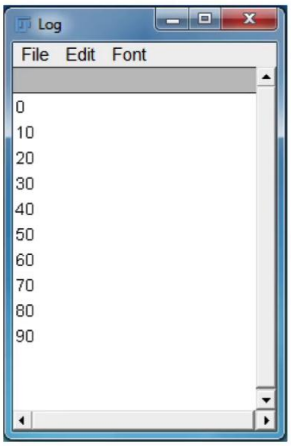
For…next statement

for (initialization; condition; increment) { statement(s) }

- Repetitive tasks
  - For …next statement

```
for (i=0; i<10; i++) {
      j = 10*i;
      print(j);
}
```

Increment: i++ i=i+1 i+=1



I leave it up to you to think about a modification of the expression in order to obtain numbers from 10 to 100.

Notes

Summary

2m 32s

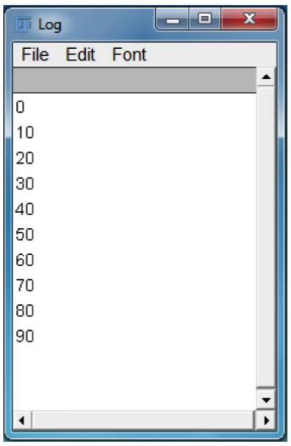
Loops

While... statement

while (condition) { statement(s) }

- Repetitive tasks
  - For …next statement
  - While…statement

```
i=0;
while (i<10) {
      j = 10*i;
      print(j);
      i++;
}
```



The previously introduced 'for next expression' is mainly found in programs and scripts. However in some cases you might also stumble over the while and/or the 'do while' statement. Therefore I will introduced it as well. The While expression consists of a Boolean expression followed by a block of commands. As long as the expression is TRUE, the block of commands is executed. The increment of the variable i is here part of the expression inside the curly brackets. The here shown while expression is returning the exact same result as the for next expression shown in the previous example. The danger of the while expression is that the initialization is mandatory. If we forget it we might end up with strange results, in particular in longer programs where the variable i might have been used before.

Notes

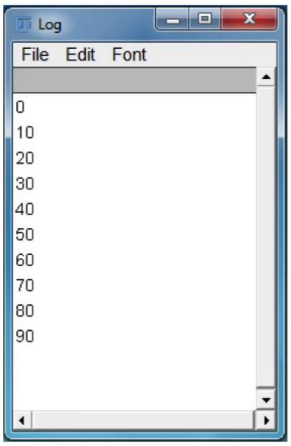
Summary

2m 42s

Do…While statement

do { statement(s) } while (condition);

```
i=0;
do {
      j = 10*i;
      print(j);
      i++;
} while (i<10);
```



- Repetitive tasks
  - For …next statement
  - While…statement
  - Do…while statement

Last but not least, there is the do while statement. It is very similar to the previously discussed 'While statement'. Here the Boolean expression is put at the end of the loop. There are cases where this might lead to different results. However, in our case it returns the exact same results. As in the case of the previous loops, the numbers in the log window are ranging form 0 to 90.

Notes

Summary

3m 39s

Loops: comparison

For...next statement

for (initialization; condition; increment) { statement(s) }

While... statement

while (condition) { statement(s) }

Do...While statement

do { statement(s) } while (condition);

Attention: endless loops

```
i=11;
do {
        j = 10*i;
        print(j);
        i-=2;
} while (i!=0);
```

So I have introduced 3 different loops. The for next expression, the while expression and the do while expression. All of them can be used in order to quickly perform iterative tasks. The for next expression has the advantage that the compiler will complain in case you forget to initialize your variable or to increment it. Both can be possible with the other expressions. So especially for newcomers, it is advisable to use the for next expression. Otherwise, there is a high risk that you program so called endless loops. An example for an endless loop is given here. Feel free to execute it and see what happens.

Notes

Summary

4m 06s

Loops: Code simplification

```
Stack.setChannel(1);          //Set Channel 1
run("Cyan");                  // set LUT
setMinAndMax(0, 128);         // set B&C
Stack.setChannel(2);          //Set Channel 2
run("Magenta");               // set LUT
setMinAndMax(0, 128);         // set B&C
Stack.setChannel(3);          //Set Channel 3
run("Yellow");                // set LUTs
setMinAndMax(0, 128);         // set B&C
```

```
luts = newArray("Cyan", "Magenta", "Yellow");
for(i=0 ; i<3 ; i++) {
    Stack.setChannel(i+1);        //Set ith channel
    run(luts[i]);                 // set LUT
    setMinAndMax(0, 128);         // set B&C
}
```

However, you should be prepared that closing and restarting Fiji/ImageJ is the only solution to leave such a loop. Toward the end of the lecture, I want to demonstrate that loops can also be used in order to simplify the code and increase its readability. On my left hand side you find two pieces of code which are doing exactly the same task. They are assigning a so called look up table to one slice of a stack, and setting brightness and contrast settings. For these 3 channels, we need 9 lines of code as shown in the upper example. However, we need to type the same command three times. Even if we use copy and paste, we end up with a piece of code which looks a bit messy. We can simplify the code by using a loop. Then it becomes obvious that three commands are executed. The first one is setting the slice of the stack based on the counter of the loop. The second one is assigning the look-up table. In order to avoid that the exact same look-up-table is used, an array with the individual look-up-tables is defined at the beginning of the program. We can now address the names of the individual look-up-tables by its position in the array. The last command is identical for all slices the min and max values are set to 0 and 128.

Notes

Summary

4m 50s

Loops: Code simplification

```
Stack.setChannel(1);          //Set Channel 1
run("Cyan");                  // set LUT
setMinAndMax(0, 128);         // set B&C
Stack.setChannel(2);          //Set Channel 2
run("Magenta");               // set LUT
setMinAndMax(0, 128);         // set B&C
Stack.setChannel(3);          //Set Channel 3
run("Yellow");                // set LUTs
setMinAndMax(0, 128);         // set B&C
```

```
luts = newArray("Cyan", "Magenta", "Yellow");
for(i=0 ; i<3 ; i++) {
    Stack.setChannel(i+1);        //Set ith channel
    run(luts[i]);                 // set LUT
    setMinAndMax(0, 128);         // set B&C
}
```

The solution using the loop has the advantage that it can be easily extended. We could, for example, normalize each channel to its maximum intensity by adding the necessary commands only once. In the upper example we would need to add it for each channel. This is tedious and also error prone.

Notes

Summary

6m 27s

Summary

- Loops
  - For …next statement
  - While statement
  - Do…while statement
- Implementation ImageJ macro language
- Attention: endless loops

Thus loops are ideally suited to simplify code and make it more reliable. It is time to summarize! I have introduced 3 different kind of loops: The for next statement, the while statement and the do while statement. All of them are well suited in order to program repetitive tasks. The mostly used loop is the for next loop. There it is impossible to forget the initialization of the counter or forget to increment is during the loop. Therefore it is less error prone. For all loops, it is essential to avoid programming so called endless loops. As such loops often required to shut down the program it is recommended to save the macro, or script, before executing it. In particular if it contains a lot of loops. Take care and good bye.

Notes

Summary

6m 49s