

## Lecture I:

# Variables, data types, operators and assignments

Mark Huiskes

LIACS, Leiden University

*Introduction to Programming, Media Technology MSc*



# Lecture Schedule

Today	Lecture I	Variables, data types, operators & assignments
22 Sep	Lecture II	Functions & mouse interaction
29 Sep		no class
6 Oct	Lecture III	Conditions & loops
13 Oct	Lecture IV	Arrays
20 Oct	Lecture V	Classes & designing complex programs
27 Oct	Lecture VI	Recursion, data structures & sorting
3 Nov		no class
10 Nov	Lecture VII	Images and libraries
17 Nov	Lecture VIII	Max/MSP/Jitter (Edwin van der Heide)

# Admin

- Lectures (structure, duration)
- Lab sessions led by Amalia Kallergi
- Assignment system
- "vrijstelling"?
- Books and laptops
- (written) Exam: **Tuesday December 1, at 10:30 AM**

# Introduction to Programming

- Mix of student backgrounds

## Main goals

- Learn basics of programming: data structures (arrays, stacks, queues etc), techniques (sorting, recursion etc)
- Learn 'Processing'
- Understand important general programming principles (various forms of abstraction)
  - Dealing with complexity: interfaces to hide implementation details (classes/objects, functions)

# Today

- A first look at Processing
- Statements
- Variables and data types, memory
- Assignments, operators and expressions
- Processing programming modes
- Conversion/casting

# Processing

- **Open source** project initiated by **Casey Reas** and **Ben Fry** (MIT Media Lab)
- Processing is “a programming language (as well as an IDE) for the **electronic arts** and **visual design** communities”
- Aims at teaching the basics of computer programming in a **visual context**
- Language builds on the graphical capabilities of **Java**: simplifying some features, and making it easy to make graphical web applets
- We will discuss some other ways of using Processing later, e.g. through **Javascript** (ported by **John Resig**), or the **Eclipse** IDE.



[http://en.wikibooks.org/wiki/Transwiki:List\\_of\\_hello\\_world\\_programs](http://en.wikibooks.org/wiki/Transwiki:List_of_hello_world_programs)

## Integrated Development Environment (IDE)

Application for software development; consists of a source code editor, a compiler/interpreter, build automation tools, and (usually) a debugger.

## Compiler

Program that translates text written in a computer language (the *source language*) into another computer language (the *target language*), usually from a high-level language to a lower-level one (e.g. machine language or assembly).

# Example I.1

## Statement

A complete instruction to the computer. The primary building blocks of a program. A statement can define a variable, assign a variable, run a function, construct an object etc. A statement always has a semi-colon at the end.

# Example 1.2

## Variable

A data element that is referenced with a name. Every variable has a data type and a value.

A variable needs to be declared before it is used. The **declaration** states the data type and variable name. For example:

```
int x;
```

Using the **assignment operator** we can give it a value

```
x = 5;
```

## Operators

For now we look at the arithmetic,  
increment and decrement  
and assignment operators

+, -, \*, /, %

++, --

+=, -=, \*=, /=, %=

% is the modulo operator, e.g.  $13 \% 5$  is equal to 3

The add assign operator += combines addition and assignment.  
Similarly there's a -=, \*= etc.

## Expression

A combination of operators, variables and values. A (valid) expression has a value, determined by these elements.

An expression can be as simple as a single number or can contain a long sequence of elements.

$x = \sin(y) / z + 15.2 - 3 * x + \text{sqrt}(5);$

# The increment and decrement operators

## Examples

```
int x = 1;  
println(x); // -> 1  
x++;  
println(x); // -> 2
```

```
int x = 1;  
println(x++); // -> 1  
println(x); // -> 2
```

```
int x = 1;  
println(++x); // -> 2  
println(x); // -> 2
```

To update the value before the expression is evaluated, place the operator in front of the variable. When it's put after the variable (postfix) the variable is incremented after the expression is evaluated.

# Operator Precedence

Example:

```
x = 4 + 5 * 6;
```

- () (function call operator)
- ++ -- ! (Increment, decrement, Logical NOT)
- Multiplicative: \* / %
- Additive: + -
- Relational: < > <= >=
- Equality: == !=
- Logical AND: &&
- Logical OR: ||
- Assignment: = += -= \*= /= %=

# Data type of a variable

- Tells the compiler what kind of values the variable can take and thus how much storage it should allocate for it, i.e. how much memory is needed for it.
- Allows compiler to check which operations may be performed on it.
- Useful for error checking.
- Various built-in types: int, float, char, boolean
- Later: defining your own, usually composite, types

Memory

# Built-in/primitive types in Processing

Type	Value Range	Size
byte	-128 to 127	1 byte
int	-2,147,483,648 to 2,147,483,647	4 bytes
float	-3.40282347E+38 to 3.40282347E+38	4 bytes
boolean	true, false	'1 bit'
char	(65536) Unicode characters	2 bytes
color	16777216 colors	4 bytes

# Example 1.3

Floats and conversions

# Operations, data types and conversion

- Operators can act differently depending on type. For example:

```
int x = 10; int y = 3; int z;  
z = x / y; // gives 3
```

```
float x = 10; float y = 3; float z;  
z = x / y; // gives 3.333...
```

If you want to get the the 'correct' fractional result out of two integers you have to make one becomes a float first:

```
int x = 10; int y = 3; float z;  
z = x / y; // gives 3.0 !!!  
z = float(x) / y; // gives 3.333...
```

# Operations, data types and conversion

- when assigned to floats, int's are automatically converted

```
int x = 3;  
float z = x; // is ok, z is 3.0 now
```

- the other way around would too often lead to information loss

```
float z = 3; // z is 3.0  
int x = z; // ERROR: can't automatically convert a float to an int
```

- if you want this/don't mind, you can convert explicitly

```
float z = 3.9;  
int x = int(z); // x is 3; the 0.9 is gone
```

# Processing Programming Modes

Processing allows programming at three levels of complexity

- **Basic**

This mode is used for drawing static images

- **Continuous**

This mode provides a `setup()` structure that is run once when the program begins and a `draw()` structure which by default continually loops through the code inside. It allows writing custom functions and classes and using keyboard and mouse events

- **Java**

Allows use of most of Java language. For advanced users.

# Example 1.4

Processing programming modes