

Lecture II:

Functions + mouse interaction

Mark Huiskes

LIACS, Leiden University

Introduction to Programming, Media Technology MSc



Lecture Schedule

15 Sep	Lecture I	Variables, data types, operators & assignments
Today	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)

Today

1. Tips + tricks
2. Summary sketch
3. Assignments last week
4. Function basics
5. Variable scope
6. Mouse interaction

Tips & Tricks



- Shortcuts: CTRL-T, CTRL-], CTRL-R, CTRL-SHIFT-R, CTRL-S
- Export application
- Examples
- In browser: F5

Summary Sketch

Names in Processing

- The first character of an identifier (i.e. a name) must be a letter, an underscore(_), or a dollar sign(\$)
- The rest of the characters in the identifier can be a letter, underscore, dollar sign, or digit. Spaces are NOT allowed in identifiers
- Identifiers are case-sensitive: *age* and *Age* are different names
- Avoid Processing's reserved words
- Stick to a consistent style

Recommended Naming Style

- variables: start lower-case
- in Processing, roomTemp is more common than room_temp
- function names: start lowercase
- Class names should be/are capitalized: String, MyCar
- Common to put constants in all-uppercase, e.g.
`int SCREEN_WIDTH = 200;`

Assignments last week

- Often useful to think about data structures first: which variables do you need; with what data types
- Start with a working program and make small incremental changes: easier to find errors.
- Use temporary variables to hold intermediate values so you can output and check them.

Function Basics

We have already seen quite a few *function calls*

function-name(arguments)

Examples: `size(200, 200);`

`background(100);`

`x = r * cos(angle);`

Today we discuss how to define your own functions

```
return-type function-name(parameter declarations) {  
    statements and declarations  
}
```

Example:

```
float circleArea(float r) {  
    float area = PI * r * r;  
    return area;  
}
```

- Why functions?
 - Reduces code duplication
 - Decomposition into simpler parts
 - Makes code re-use across programs easier
 - Clarity, improved readability
 - Abstraction
 - hiding details of implementation, so can be used without needing to understand these details
 - top-down programming: break down task into smaller tasks
- The main reason to run a function can be to compute a return-value, or to achieve some 'side effects' (e.g. draw a figure); often it's both...

Eyes example

Function Lingo

- **Parameters**

Variables local to a function whose initial values are supplied when the function is called.

- **Arguments**

Values supplied when calling a function. These values are used to initialize the corresponding parameters in the same way that variables of the same type are initialized.

- **Function body**

Block that defines the actions of a function.

- A function *returns* a value

- Variables declared inside the function are called *local variables*

- Related terms: **subroutine**, **procedure**, **method**

How to define a function

```
return-type function-name(parameter declarations) {  
    statements and declarations  
}
```

- Determine what you want the function to do
- Determine required input parameters (and their data types, separated by commas)
- Determine the return type (`void` if you don't return anything)
- Write the function body

What happens when a function is called?

- Argument expressions are evaluated
- Their results need to be of the proper type (or automatic conversion must be possible)
- The expression values are copied into the local parameter variables
- Note that for object and array arguments this is a bit more subtle (later lectures).
- The function body is executed
- If the return type is not void, the function execution ends with a return statement:

return expression;

- The program continues execution where it left off, possibly using the return value

Example II.1

Variable scope

Scope of a variable: part of the program where you can access the variable

- Variables declared inside any block can be accessed only inside their own block and inside any blocks enclosed within their block
 - Local/internal variables have scope limited to the function (or other construct) in which they are declared
 - Global/external variables (declared on same level as `setup()` and `draw()`) have 'global scope': can be accessed anywhere in the program
- Scope can be smaller than a function: inside statement blocks (`{ .. }`) or inside iteration constructs (next lecture)
- When a local variable (or parameter) has the same name as a global variable, then the local variable (or parameter) is used

Example

```
int x = 5; // global variable
```

```
void setup() {
```

```
    int y = 6; // local variable
```

```
    int x = 7; // local variable that 'masks' global variable x
```

```
    println(x); // prints 7
```

```
}
```

```
void draw() {
```

```
    println(x); // prints the global variable, so 5
```

```
    println(y); // ERROR: attempt to access y outside scope
```

```
}
```

- Function parameters work as local variables; the argument values are copied into the parameters ("pass-by-value")
- The original variables used in the argument remain unaffected

Question II.1 sketch

Function Overloading

You may have more than one function of the same name as long as their **parameter types** are different

The compiler makes sure that the right function is called based on the arguments you supply

Example II.2

Mouse Interaction I

- Processing has global variables for the current mouse position: `mouseX` and `mouseY`
- The position of the mouse at the time that the previous frame was drawn is also still available:
in `pmouseX` and `pmouseY`
- All four variables are integer coordinates (so relative to top-left corner)

Example II.3

Mouse Interaction II

- `mousePressed()`: gets called each time a mouse button is pressed
- `mouseMoved()`: gets called each time the mouse is moved
- `mouseDragged()`: gets called each the mouse is dragged

Timing (Processing specific!):

during the execution of the `draw()` function all 'events' are collected. After `draw()` is finished the corresponding functions are executed.

Example II.4

More Functions

Relation to mathematical functions:

- Side-effects
- Effect of global variables, randomness
- Functional programming

Looking up function declarations

<http://dev.processing.org/reference/core/>

Some more built-in functions

- `int/float abs(x1)` – computes absolute value of int/float x1
e.g. `abs(-10.5)` returns 10.5
- `max(x1, x2)` or `max(x1, x2, x3)` – determines largest value, preserves ints and floats
e.g. `max(10.6, 5)` returns 10.6; `max(1, 2)` returns an int value 2
- `min(x1, x2)` or `min(x1, x2, x3)` – smallest value
- `float sqrt(float x1)` – square root of x1
e.g. `sqrt(10)` returns 3.162277...
- `float sq(float x1)` – square of x1
`sq(2.1)` returns 4.41
- `float pow(float num, float exponent)` – power: returns $\text{num}^{\text{exponent}}$
e.g. `pow(3.5, 4)` returns $(3.5)^4 = 150.0625$

- `int ceil(float x1)` – rounds up
e.g. `ceil(3.4)` returns 4
- `int floor(float x1)` – rounds down
- `int round(float x1)` – rounds to nearest integer
e.g. `round(9.5)` returns the integer value 10
- `float dist(float x1, float y1, float x2, float y2)` – distance between points (x1, y1) and (x2, y2)

computes $\text{sqrt}(\text{sq}(x1-x2) + \text{sq}(y1-y2))$

(also works for points with 3 coordinates)

Random()

- `float random(float high)` returns a floating point number between zero and high, i.e.
 $0 \leq \text{random}(\text{high}) < \text{high}.$
- `float random(float low, float high)` returns a floating point number between low and high, i.e.
 $\text{low} \leq \text{random}(\text{low}, \text{high}) < \text{high}$

```
int nEyes = (int)random(6) + 1;
```

Dice demo