Fakultät für Betriebswirtschaft
Munich School of Management

Basics in Programming for MMT

Session 4 – Functions

www.mmt.bwl.lmu.de

Scope of the Session

1. Repetition

- Datatypes
- Loops
- Arrays

2. Theory

- Operations to Commands
- Functions
- void
- Return-Types
- Setup and Draw
- keyPressed() and keyReleased()

3. Next

- Session 5

4. Project

Repetition

Repetition

Datatypes

- If we declare variables, we have to specify their types.
- Different datatypes require different space in the working memory.

String	`String s = "hello world!";`
Character	`char c = 'a';`

Repetition

Loops

- Loos help us to solve recurring patterns.
- The tree instructions in the `()` define the execution.

```
for (int i=0; i<10; i = i+1) {
   ellipse(300,300,200-10*i,200-10*i);
}
```

Repetition

Arrays

- In an array you can **store** multiple values of one datatype.
- You can **access** these values by referencing the array and the specific index.

```
int [] a;
void setup () {
    size(600,600);
    a = new int [3];
    a [0] = 255;
    a [1] = 100;
    a [2] = 30;
}

void draw () {
    background(a[0],a[1],a[2]);
}
```

Theory

Theory

Operations to Commands (1/2)

- Sometimes we recognize **patterns** in our code structure.
- Can we define our own **command** and reuse it in a generalized way?

```
void setup () {...}

void draw () {
    stroke(255,0,0);
    fill(255,0,255);
    ellipse(33,44,55,55);

    stroke(255,0,0);
    fill(0,255,255);
    ellipse(22,77,33,33);

    stroke(0,0,255);
    fill(255,255,0);
    ellipse(55,11,22,22);
}
```

Theory

Operations to Commands (2/2)

- How can we define our own commands?

```
void setup () {...}

void draw () {
    coCircle(33,33,55,10,0,0,73,0,45);

    coCircle(22,77,33,0,14,0,0,14,83);

    coCircle(55,11,22,0,0,73,53,42,0);
}
```

Theory

Functions (1/2)

- Outside of `setup` and `draw` we can define **functions**.
- The function's name `coCircle` specifies the commands defined in `{}`.
- In `{}` you can use the arguments passed in the `()`.

```
void setup () {...}

void draw () {...}

void coCircle (int x, int y, int d, int sR, int sG, int sB, int fR, int fG,
    int fB)
    stroke(sR, sG, sB);
    fill(fR, fG, fB);
    ellipse(x,y,d,d);
}
```

Theory

Functions (2/2)

`void`	`func`	`(int i, ..)`	`{...}`
Type	**Name:** Allows to call the function and perform the included instructions	**Arguments:** The set of arguments that is passed to the function during execution	**Body:** Contains the commands that are executed if called You can use the passed arguments here.

Theory

Void (1/2)

- What is this `void` all about?
- Functions can be called in different locations.
- Also after the assignment operator

```
drawACircle(25,25,25);

int x = theDoubleOf(3);
```

Theory

Void (2/2)

- That means that functions have to be able to generate values that later can be processed or assigned to variables!
- `circle()` does not create a value, instead it creates visual output.
- `void` marks the functions as returning no value.

```
coCircle(25,25,25, ...);

int x = theDoubleOf(3);

void drawACircle (int x, int y, int
ellipse(x,y,2*r);
}
```

Theory

Return-Types (1/3)

- Using a datatype instead of `void` defines the value the function has to return.
- Therefore, we have to use the `return ...;` command to pass a value.

```
drawACircle(25,25,25);

int x = theDoubleOf(3);

void drawACircle(int x, int y, int
  ellipse(x,y,2*r);
}

int theDoubleOf (int n) {
    return n*2;
}
```

Theory

Return-Types (2/3)

`int`	`func`	`(int i)`	`{return 2*i;}`
Return-Type	**Name**	**Arguments**	**Body:** The last command to execute in the body is the return statement, which passes a value of the specified return-type to the outside.

Theory

Return-Types (3/3)

- We can use functions that return values instead of: arguments, to assign values, …
- We have already done it!

```
float x = random(width);
float y = random(height);
```

Theory

Setup and Draw

- `setup` and `draw` are functions that are predefined, but we specify the commands to be executed if called.

```
void setup () {

}

void draw () {

}
```

Theory

KeyPressed() and KeyReleased()

- You can react to the keyboard by using `keyPressed()` and `keyReleased()`.
- These functions are called each time a key is pressed/released.
- The predefined `key` variable stores the character of the last key being pressed.

```
void setup () {}

void draw () {}

void keyPressed () {
    if (key == 'w') {}
    if (key == 's') {}
}

void keyReleased () {}
```

Next

Next

Session 5

- How to define new datatypes?
- How to use multiple instances of one thing?

Project

Project

1. Introduce variables that describe the position of the two batches.
2. Make them move by reacting to key presses.
3. Make them stay inside borders. Do they move simultaneously?