BASICS IN PROGRAMMING

INTERFACES

Created by Beat Rossmy
SCOPE

1. Variables and Data Types
   1. Variables
   2. Datatypes
   3. Boolean(-operators)
   4. Colors

2. Control Structures
   1. If and Else
   2. Loops
   3. Functions

3. Arrays
   1. Arrays
   2. Iterate

4. Classes
   1. Class
   2. Inheritance
   3. Interfaces
VARIABLES AND DATA TYPES
### VARIABLES

**Declaration:** give values a keyword/name (variable) to make them "memorize-able".

```plaintext
int x;
```

**Initialization:** give these variables initial values.

```plaintext
void setup () {
  size(600,600);
  x = 100;
}
```

**Usage:** use variables instead of static values (e.g arguments). PC looks up the values of variables during execution.

```plaintext
void draw () {
  background(0);
  rect(x,200,200,200);
}
```
**VARIABLES**

<table>
<thead>
<tr>
<th>Datatype:</th>
<th>Name:</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables can be of different types.</td>
<td>names can be single letters but also words. Always start with lowercase.</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
int x;
```
**DATATYPES**

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

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td><code>int i = 10;</code></td>
</tr>
<tr>
<td>Float</td>
<td><code>float f = 3.33;</code></td>
</tr>
<tr>
<td>String</td>
<td><code>String s = &quot;hello world!&quot;;</code></td>
</tr>
<tr>
<td>Character</td>
<td><code>char c = 'a';</code></td>
</tr>
<tr>
<td>Boolean</td>
<td><code>boolean b = false;</code></td>
</tr>
</tbody>
</table>
BOOLEAN(-OPERATORS)

Statements generate boolean values.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>greater than</td>
<td>( x &gt; 100 )</td>
</tr>
<tr>
<td>greater or equal</td>
<td>( x \geq 100 )</td>
</tr>
<tr>
<td>equal</td>
<td>( x == 100 )</td>
</tr>
<tr>
<td>smaller or equal</td>
<td>( x \leq 100 )</td>
</tr>
<tr>
<td>smaller than</td>
<td>( x &lt; 100 )</td>
</tr>
<tr>
<td>unequal</td>
<td>( x \neq 100 )</td>
</tr>
</tbody>
</table>
**BOOLEAN(-OPERATORS)**

Booleans can be **combined or manipulated** to new boolean values.

- **and**
  
  - `true && true == true`
  - `true && false == false`
  - `false && true == false`
  - `false && false == false`

- **or**
  
  - `true || true == true`
  - `true || false == true`
  - `false || true == true`
  - `false || false == false`

- **not**
  
  - `!true == false`
  - `!false == true`
COLORS

- Colors are either entered as gray values or RGB values.
- The number of arguments specifies the color type.
- Each color channel can take values from 0-255.

<table>
<thead>
<tr>
<th>Gray</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>background(0); fill(123); stroke(255);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RGB</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>background(255,0,0); fill(0,255,0); stroke(0,0,255);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transparent colors.</th>
<th>0-255</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill(r,g,b,a); fill(g,a);</td>
<td></td>
</tr>
</tbody>
</table>
IF AND ELSE

- Based on a condition we can execute specific code sections.
- if the condition is true execute {...}. else execute {***}

```java
void draw () {
    background(0);
    x = x+1;

    if (x>100) {...}
    else {***}

    rect(x,200,200,200);
}
```
IF AND ELSE

Keyword

Condition:
a statement that describes a certain state. A statement is either true or false.

Consequence:
if the condition is true the included commands are performed and otherwise skipped.

```python
if (x>0)
...
```
LOOPS

- Loops help us to solve recurring patterns.
- The three instructions in the () define the execution.

```java
for (int i=0; i<10; i = i+1) {
    ellipse(300, 300, 200-10*i, 200-10*i);
}
```
<table>
<thead>
<tr>
<th>for</th>
<th>(int i=0; i&lt;100; i=i+1)</th>
<th>{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start:</strong></td>
<td>initial value of the counter.</td>
<td></td>
</tr>
<tr>
<td><strong>End:</strong></td>
<td>what is the maximum value of the counter?</td>
<td></td>
</tr>
<tr>
<td><strong>Steps:</strong></td>
<td>how to increment after each loop.</td>
<td></td>
</tr>
<tr>
<td><strong>Body:</strong></td>
<td>commands to be performed.</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTIONS

- Functions allow us to use **generalized** sets of instructions.
- A set of commands is performed when calling the function, using the **attributes** from the attribute list.
- Functions can return values of datatypes or do not return any value (**void**).

```c
void printRandomCharacters () {
    for (int i=0; i<100; i = i+1) {
        print(char((int)random(255)));
    }
}

int double (int v) {
    return 2*v;
}
```
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.

```java
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]);
}
```
ITERATE

○ Use loops to automatically **iterate** over all elements of an array.

○ Use this technique for: initialization or handling all elements.

```java
int [] a;
void setup () {
    size(600,600);
    a = new int [3];
    for (int i = 0; i<3; i = 1+1) {
        a[i] = (int)random(255);
    }
}

void draw () {
    background(a[0],a[1],a[2]);
}
```
CLASSES
A `class` allows us to define data structures.

The class (`Ball`) is the abstract description and the objects (`b`) are instances of that class.

Each object contains its own set of variables defined in the class as fields.
<table>
<thead>
<tr>
<th>keyword</th>
<th>class + classname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>float x;</td>
</tr>
<tr>
<td></td>
<td>float y;</td>
</tr>
<tr>
<td></td>
<td>float d;</td>
</tr>
</tbody>
</table>
| Constructor: classname + arguments | public Ball (int x, int y, int d) {
|         | this.x = x;       |
|         | this.y = y;       |
|         | this.d = d;       |
| Methods: | void move () {
|         | x = x+1;          |
|         | y = y+1;          |
| End of class. | } |
Inheritance enables us to define functionalities and datastructures once that are shared by different classes.

The child class (B) inherits all the fields and methods of its parent class (A).

A class can have multiple children but only one parent.
### INHERITANCE

<table>
<thead>
<tr>
<th>keyword</th>
<th>extends + parentname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td></td>
</tr>
<tr>
<td>Constructor: call parent constructor</td>
<td></td>
</tr>
<tr>
<td>Methods:</td>
<td></td>
</tr>
</tbody>
</table>

**class B extends A {**

```java
float x;
float y;
// fields of A are inherited
```

```java
public B (int x, int y) {
    super(...); // call A constructor
    this.x = x;
    this.y = y;
}
```

```java
void doSomething () { ... }
// methods of A are inherited
```

**End of class.**

```java
}
```
INTERFACES

- Interfaces define methods that **have to** be implemented by the classes implementing this interface.

- The interface specifies the **return type**, the **name** and the **arguments**, but **not the commands** that are performed.

- A class can implement **multiple** interfaces.

```java
interface Doable {
    void doSomething ();
    void doSomethingElse (int x);
}

class A implements Doable {
    ...
    void doSomething () {...}
    void doSomethingElse (int x) {...}
}

class Z {
    ...
    void doSomething () {...}
    void doSomethingElse (int x) {...}
}
REFERENCES