Spatial Simulation

Winter Semester 2023 / 24, MSc Applied Geoinformatics

Principle of coding (with GAMA)

GAMA - Help!
..a resource collection

The GAMA website

http://gama-platform.org/ use the “Search” field top right to look for functionalities and commands

basic skeleton of a GAMA model

https://gama-platform.github.io/wiki/ModelOrganization

Documentation index (!!!)

https://gama-platform.org/wiki/Exhaustive-list-of-GAMA-Keywords

GitHub Wiki

https://github.com/gama-platform/gama/wiki

Errors in your model? Code Verification helps

https://github.com/gama-platform/gama/wiki/ValidationOfModels

Structure your code!

Curly brackets

- a section always starts with { and ends with }
- If you place the cursor right after a bracket, the other bracket is highlighted

Indentation

- each sub-section is shifted (indented) further to the right
- a line break after a bracket will automatically be indented
- use the tabulator to indent manually
- to indent a selected block of lines, use CapsLock+Tab (right shift) or Shift+Tab (left shift)
- If you want: colourise your code sections

Code folding

- Each section can be folded and unfolded

Structure outline

- Explore the structural overview in the GUI

Commenting of code

Code commenting is useful for two things

1. Comment to explain what you do

```
//Number of lion-agents
int number_of_lions <- 5;
```

2. If you want to exclude a code part from execution comment out

```
//Number of lion-agents
//int number_of_lions <- 5;
```

To toggle commenting for a selected code part on and off, use Ctrl + 7

Functions

= specify the processes to be executed, i.e. the behaviour of a model, a species, or a grid cell.

There are three main types

1. `init {}`
   Initialisation. Executed only once, at the beginning

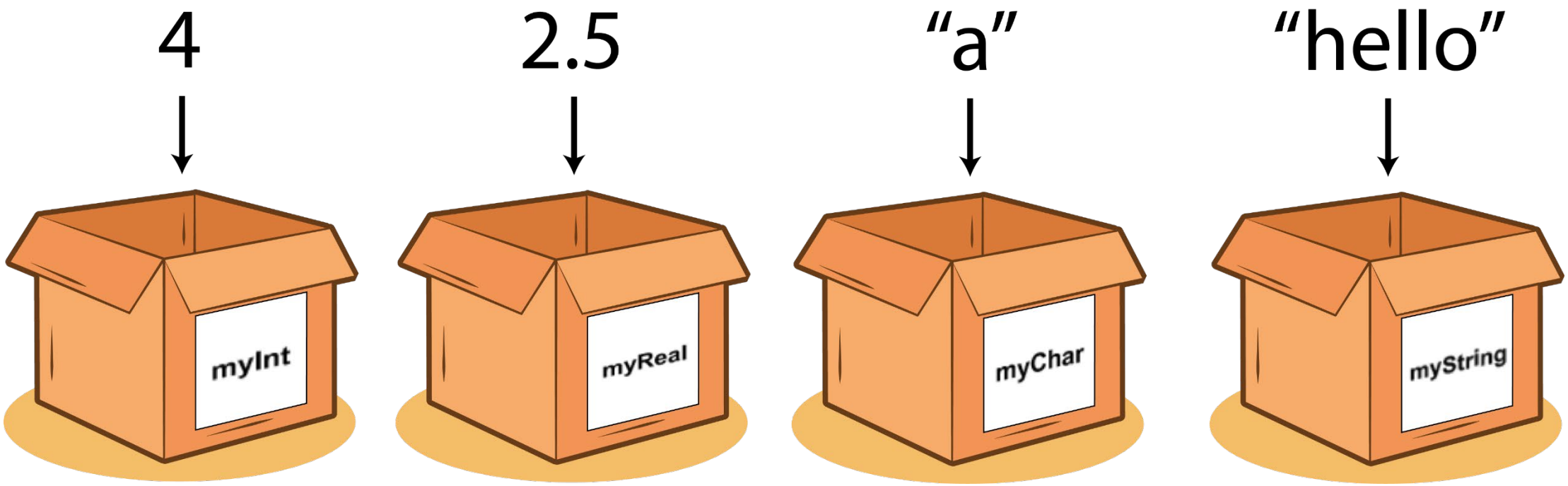
2. `reflex name_of_function {}`
   Automatic execution at every time step;

3. `action name_of_function {}`
   Execution everytime the action is called
   to call the function: `do name_of_function`

For example

```
reflex write_message {
    write “Hello World";
}
```

Variables



int float string

Variables

= property of the model, an agent, or a grid cell

The definition of a variable has 2 steps:

1) Declaration (prepare the „box“)

   Brings the variable into existence; defines the type of a variable, and its name:

```
int age;
```

2) Assignment of a value (put a value into the box)

```
age <- 3;
```

The two steps can be shortened into one:

```
int age <- 3;
```

Variables – primitive types

Declaration in GAMA	Type	Example
int	Integer (whole numbers)	1, 5, 239
float	Decimal numbers	4.927, 1.1
bool	Boolean	true, false
string	Characters	abc, Hello World!

Global variables

= properties of the model

There is only one value for this property in the entire model, e.g. the maximum age of an agent.

1) Declaration

   All agent variables are declared right in the beginning of the global section

```
int max_age;
```

2) Assignment of a value

```
max_age <- 30;
```

Agent variables

= property of an agent (in GAMA: „species“).

Each agent has the same property (e.g. age), but this variable can have different values (e.g. 2 years, 15 years, etc).

1) Declaration

   All agent variables are declared right in the beginning of the species section

```
int age;
```

2) Assignment of a value

```
age <- 3;
```

Cell variables

= property of a cell in a raster (in GAMA: „grid cell“).

Each cell has the same property (e.g. elevation), but this variable can have different values (e.g. 500m, 505m, 508m, etc).

1) Declaration

   All agent variables are declared right in the beginning of the species section

```
float elev;
```

2) Assignment of a value

```
elev <- 300.0;
```

Parameters

= input variables. An existing global variable, which accepts user-defined input.

Defined as a parameter at the beginning of an experiment.

```
parameter "What’s your age: " var: age;
```

Calculate with variables

To calculate a new value for an integer variable, use basic maths:

```
age <- 1 + 1;
```
(result: age = 2)

```
age <- rnd(10);
```
(result: age = a random number between 0 and 9)

```
age <- age + 1;
```
(result: age = current age +1)

Update a variable‘s value each time step with the update facet:

```
age <- 1 update: age + 1;
```