ÖGOR Summer-Workshop for PhD-candidates and Post-Docs

An introduction to Julia and JuMP for Operations Research

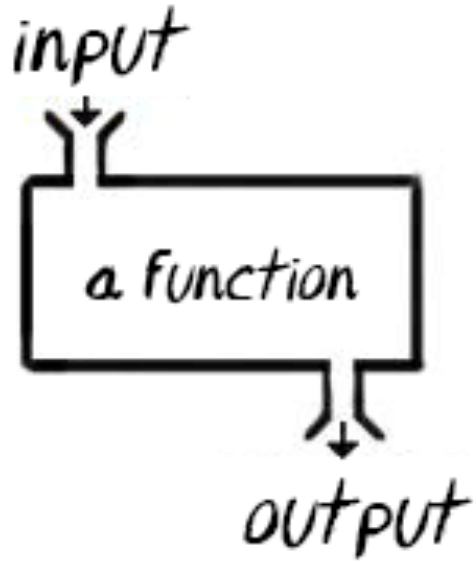
Prof. Dr. Xavier Gandibleux

Nantes Université – France
Département Informatique – Faculté des Sciences et Techniques

Topic 9

Control flow (part 2)

Functions (Con't)



Declaring a function (reminder)

Julia gives us different ways to write a function:

- A single expression function
- An anonymous function
- A general function

Declaring and calling a general function (reminder)

Mutable and immutable objects (see `ismutable` function)

The following type are immutable:

- integer
- float
- boolean
- character
- tuple

If a function has a parameter with this type, modifying the variable inside the function didn't modify the value outside the function.

The following type are mutable:

- array

If a function has a parameter with this type, modifying the variable inside the function changes the value outside of the function.

By convention, **functions followed by ! alter their contents**.

Broadcast

```
julia> function add1(x::Int64)
         return x+1
       end
```

```
julia> s=1
julia> add1(s)
```

```
julia> a=[1,2,3]
julia> add1(a)                                  # error
```

Broadcast

```
julia> function add1(x::Int64)
         return x+1
       end
```

```
julia> s=1
julia> add1(s)
```

```
julia> a=[1,2,3]
julia> add1(a)                                  # error
```

Broadcast

"broadcast" a function to work over each elements of an array:

1)

```
broadcast(funcName,elements)
```

```
julia> broadcast(add1,a)
```

2)

```
funcName.(elements)
```

```
julia> add1.(a)
```

Broadcast

"broadcast" a function to work over each elements of an array:

1)

```
broadcast(funcName,elements)
```

```
julia> broadcast(add1,a)
```

2)

```
funcName.(elements)
```

```
julia> add1.(a)
```

Broadcast

"broadcast" a function to work over each elements of an array:

1)

```
broadcast(funcName,elements)
```

```
julia> broadcast(add1,a)
```

2)

```
funcName.(elements)
```

```
julia> add1.(a)
```

Broadcast

"broadcast" a function to work over each elements of an array:

1)

```
broadcast(funcName,elements)
```

```
julia> broadcast(add1,a)
```

2)

```
funcName.(elements)
```

```
julia> add1.(a)
```

Multiple-dispatch (1/2)

- The same function can be defined with different number and type of parameters; example:

```
julia> brol(x::Int64)= println("Int:  ",x)
julia> brol(x::Float64)= println("Flt:  ",x)
julia> brol(x::Bool)= println("Bool:  ",x)
julia> brol(x)= println("Others:  ",x)
```

- These different versions are named methods in Julia
- Inspect the methods of a function with

```
methods(functName)
```

- When calling such functions, Julia will pick up the correct one depending from the parameters in the call
(by default the stricter version).

Multiple-dispatch (1/2)

- The same function can be defined with different number and type of parameters; example:

```
julia> brol(x::Int64)= println("Int:   ",x)
julia> brol(x::Float64)= println("Flt:   ",x)
julia> brol(x::Bool)= println("Bool:   ",x)
julia> brol(x)= println("Others:   ",x)
```

- These different versions are named methods in Julia
- Inspect the methods of a function with

```
methods(functName)
```

- When calling such functions, Julia will pick up the correct one depending from the parameters in the call
(by default the stricter version).

Multiple-dispatch (1/2)

Example:

```
julia> brol(8.3)
julia> brol('c')
julia> brol(3)
julia> brol(true)
julia> brol("hello")
```

```
julia> methods(brol)
```

Multiple-dispatch (1/2)

Example:

```
julia> brol(8.3)
julia> brol('c')
julia> brol(3)
julia> brol(true)
julia> brol("hello")
```

```
julia> methods(brol)
```