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

An introduction to Julia and JuMP for Operations Research

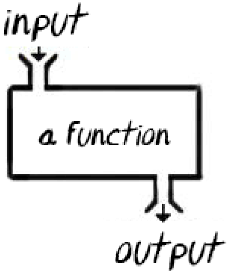
Prof. Dr. Xavier Gandibleux

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

Topic 3

Control flow (Part 1)

Functions



Declaring a function

Julia gives us different ways to write a function:

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

Declaring and calling a single expression function

Function in a single line

Example of declaration:

```
julia> f(x) = x^2 + 7
```

Example of call:

```
julia> f(2)
```

Declaring and calling a single expression function

Function in a single line

Example of declaration:

```
julia> f(x) = x^2 + 7
```

Example of call:

```
julia> f(2)
```

Declaring/calling an anonymous function

No named function

Example over a scalar:

```
julia> map(x -> x^2 + 7 , 2)
```

Example over a vector:

```
julia> map(x -> x^2 + 7 , [2, 7, 4])
```

Example with multiple parameters:

```
julia> map((x,b) -> x^2 + b , 2, 7)
```

Declaring/calling an anonymous function

No named function

Example over a scalar:

```
julia> map(x -> x^2 + 7 , 2)
```

Example over a vector:

```
julia> map(x -> x^2 + 7 , [2, 7, 4])
```

Example with multiple parameters:

```
julia> map((x,b) -> x^2 + b , 2, 7)
```

Declaring/calling an anonymous function

No named function

Example over a scalar:

```
julia> map(x -> x^2 + 7 , 2)
```

Example over a vector:

```
julia> map(x -> x^2 + 7 , [2, 7, 4])
```

Example with multiple parameters:

```
julia> map((x,b) -> x^2 + b , 2, 7)
```

Declaring and calling a general function (1/6)

With a single parameter

Example of declaration:

```
julia> function affinefct(x)
         y = x^2 + 7
         return y
       end
```

Example of call:

```
julia> affinefct(2)
```

Declaring and calling a general function (1/6)

With a single parameter

Example of declaration:

```
julia> function affinefct(x)
          y = x^2 + 7
          return y
       end
```

Example of call:

```
julia> affinefct(2)
```

Declaring and calling a general function (2/6)

With multiple parameters

Example of declaration:

```
julia> function affinefct(x,b)
         y = x^2 + b
         return y
       end
```

Example of call:

```
julia> affinefct(2,7)
```

Declaring and calling a general function (2/6)

With multiple parameters

Example of declaration:

```
julia> function affinefct(x,b)
         y = x^2 + b
         return y
       end
```

Example of call:

```
julia> affinefct(2,7)
```

Declaring and calling a general function (3/6)

When the type of parameters is specified

Example of declaration:

```
julia> function affinefct(x::Int64,b::Int64)
         y = x^2 + b
         return y
       end
```

Example of call:

```
julia> affinefct(2,7)
```

Declaring and calling a general function (3/6)

When the type of parameters is specified

Example of declaration:

```
julia> function affinefct(x::Int64,b::Int64)
         y = x^2 + b
         return y
       end
```

Example of call:

```
julia> affinefct(2,7)
```

Declaring and calling a general function (4/6)

With optional arguments

Example of declaration:

```
julia> function affinefct(x,b=3)
         y = x^2 + b
         return y
       end
```

Examples of call:

```
julia> affinefct(2)
```

```
julia> affinefct(2,7)
```

Declaring and calling a general function (4/6)

With optional arguments

Example of declaration:

```
julia> function affinefct(x,b=3)
         y = x^2 + b
         return y
       end
```

Examples of call:

```
julia> affinefct(2)
```

```
julia> affinefct(2,7)
```

Declaring and calling a general function (5/6)

The return Keyword

- return a single value

```
return expression
```

- return several values

return $expression_1, expression_2 \ldots expression_n$

- return no value

```
return nothing
```

Declaring and calling a general function (6/6)

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**.

Declaring and calling a general function (6/6)

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**.

Review and exercises

(notebook)