INTRODUCTION TO THE JULIA PROGRAMMING LANGUAGE

This text uses Julia code embedded in TeXmacs documents to construct scientific computation models. The TeXmacs menu item Insert->Session->Julia inserts a new Julia session,

```
Julia (1.6.1) session in GNU TeXmacs
```

```
∴
```

Julia usage is introduced by example throughout this course, assuming no prior familiarity with programming. The basic interaction consists of the following steps:

1. Julia instructions are entered into the session, and Enter is pressed. The Julia interpreter reads the instructions.
2. The instructions are evaluated.
3. A result is returned and printed. Julia awaits for new instructions.

The above is known as a read-evaluate-print loop, with an REPL acronym. Here is a simple example

```
∴ 1+2
```

```
3
```

```
∴
```

The expression $1+2$ was introduced, the addition was evaluated, and 3 was printed as the result. A new box awaiting further instructions is inserted in the document.

1. Numbers

The standard arithmetic operations can be carried out in Julia.

```
∴ 2+3
```

```
5
```

```
∴ 2+3*5
```

```
17
```

```
∴ (2+3)*5
```

```
25
```

Evaluation of a number returns that number

```
∴ 1
```

```
1
```

```
∴
```

Julia represents numbers through various types, such as integers (`Int64`) and reals (`Float64`), distinguished by use of the decimal point notation. The type of a particular expression is returned by `typeof()`.

```
∴ typeof(1)
```

```
Int64
```

```
∴ typeof(1.0)
```

```
Float64
```

```
∴
```

Addition of different types yields a result of the more general type

```
∴ typeof(1+1.)
```

```
Float64
```

```
∴
```

2. Variables

A variable is a notation for computer memory locations that contain values that may change during the course of a computation. Variable symbols start with a letter, and distinguish between upper and lower case. If a variable has not yet been define, an informational message appears. If a variable has been defined, evaluation of the variable name gives the current contents of the associated memory locations. Variables also have a type, and the type is inferred by the value given to the variable

```
∴ a
```

```
UndefVarError(:a)
```

```
∴ a=1
```

1

```
∴ typeof(a)
```

```
Int64
```

```
∴ b=1.
```

1.0

```
∴ typeof(b)
```

```
Float64
```

```
∴ A
```

```
UndefVarError(:A)
```

```
∴ a=1.
```

1.0

```
∴ typeof(a)
```

```
Float64
```

```
∴
```

Arithmetic expressions can be built up between variables and numbers

```
∴ 2*a+3*b
```

5.0

```
∴ a-1
```

0.0

```
∴
```

The equal sign is used to denote assigning a value to a function, $a \leftarrow a+1$ becomes

```
∴ a=a+1
```

2.0

```
∴ a
```

2.0

```
∴
```

Variables are defined within a particular scope. The above variables are defined in the general scope. Local scopes can be defined as shown below.

3. Functions

Functions are expressions that receive an input, process it, and return an output. Julia has two ways of defining a function:

1. On one line for simple functions

```
∴ f(x)=x^2-2
```

```
f
```

```
∴ f(0)
```

```
-2
```

```
∴ f(2)
```

```
2
```

```
∴ f(a)
```

```
2.0
```

```
∴ a
```

```
2.0
```

```
∴ g(x,y)=x+y
```

```
g
```

```
∴ g(1,2)
```

```
3
```

```
∴ f(x,y)=x+y
```

```
f
```

```
∴ f(0)
```

```
-2
```

```
∴ f(1,2)
```

```
3
```

```
∴
```

2. Over multiple lines (Shift+Enter gives a new line without evaluation of the expression). The function definition ends with the `end` keyword, and returns the last evaluated expression

```
∴ function g(x,y)
    u=x-y
    z=x+y
    return u
  end
```

```
g
```

```
∴ g(3,2)
```

```
5
```

```
∴ function g(x,y)
    u=x-y
    z=x+y
  end
```

```
g
```

```
∴ g(1,2)
```

−1

```
∴ u
```

```
UndefVarError(:u)
```

```
∴
```

4. Conditionals

Julia evaluates logical expressions

```
∴ 1<2
```

true

```
∴ 2<1
```

false

```
∴ a
```

2.0

```
∴ a<1
```

false

```
∴
```

Logical or is denoted as ||. Logical and is denoted as &&.

```
∴ (1<2) || (2<1)
```

true

```
∴ (1<2) && (2<1)
```

false

```
∴
```

Negation is denoted through !

```
∴ !(1<2)
```

false

```
∴
```

A conditional expression consists of three parts:

1. a condition that is evaluated;
2. an expression that is evaluated if the condition is true;
3. an expression that is evaluated if the condition is false.

```
∴ if (1<2)
      print("1<2\n")
  else
      print("1>2\n")
  end
```

```
1>2
```

```
∴
```

5. Loops

Scientific computation involves repeated execution of instructions. As an example consider the sequence

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}, n = 2, \ldots, x_1 = 1.$$

To repeatedly evaluate the above instruction a number of times known before hand a for loop can be used.

```
∴ x=1.;
```

```
∴ for n=1:5
    global x
    print("n=",n,"␣x=",x,"\n")
    x = x/2 + 1/x
  end
```

```
n=1 x=1.0
n=2 x=1.5
n=3 x=1.4166666666666665
n=4 x=1.4142156862745097
n=5 x=1.4142135623746899
```

```
∴
```

The number of repetitions can be controlled by a condition.

```
∴ x=1.; n=0;
```

```
∴ while (abs(x-sqrt(2.))>0.000001)
    global x, n
    print("n=",n,"␣x=",x,"\n")
    x = x/2 + 1/x; n = n + 1;
  end
```

```
n=0 x=1.0
n=1 x=1.5
n=2 x=1.4166666666666665
n=3 x=1.4142156862745097
```

```
∴
```

6. Ranges

7. Arrays