Writing and reading files

A file has to be created or opened before working with it
- a file then becomes associated with an IOStream object

```
f=open(“file.dat”)
```

or

```
filename=“file.txt”
f = open(filename)
```

f is now the IOStream object
- used to refer to the file

This way of opening allows only to read the file
- the file must exist already

`f = open(filename,”r”)`	open for reading (“r” optional)
`f = open(filename,”w”)`	creates file or destroyes existing file
`f = open(filename,”a”)`	for writing, appends existing file

A file should be closed after it has been used

```
close(f)
```

The standard input and output streams are always open

`stdin`	normally the keyboard	(optional to include)
`stdout`	normally the screen	

Examples of reading from a file:

`data = parse(Float64,readline(f))` item on a line or last item on line

`data = parse(Float64,readuntil(f,str))` item followed by the string str

`data = readline(f)` a line of binary data

Examples of writing

`print(f,a,”  “,b,”  “)`

`println(f,a,”  “,b)`

println() is print() with a newline character following after whatever is printed

- next print will be on the next line
- with print(), next output will be on same line

Colored output with

`printstyled(f,a,color=:blue)`

Formatted output best done with @printf (macro) - see Julia doc

Binary output/input

Large data sets should be written in binary form (more compact)

`write(f,data)` ‘data’ could be a big array (you will not be able too “see” it)

Read in binary data this way

`read!(f,data)` the next item in the file must match the size of ‘data’

Examples of files, writing, reading online in write.jl and read.jl

Scope of variables

Scope = part of code where a variable is visible

Scopes are **nested**

- Inner scopes can access variables only in outer scopes

There can be more than one global scope

- each module is its own global scope

Local scope blocks (examples)

- functions, loops (for, while), macros

Role of scopes

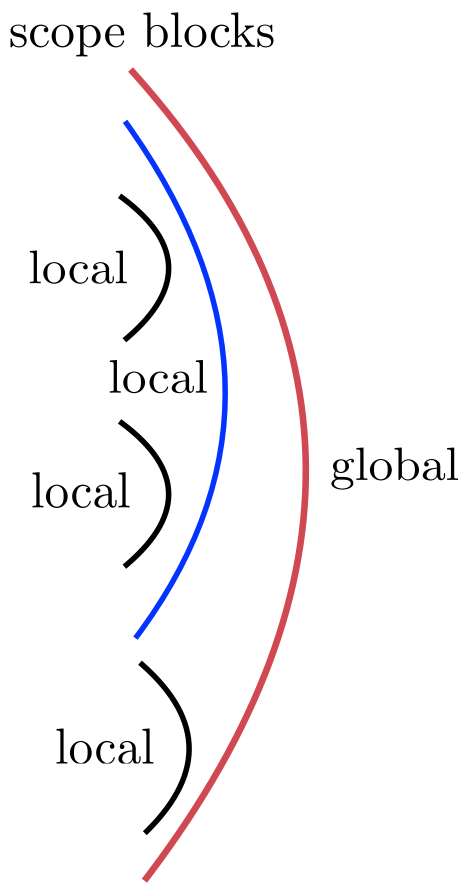
- avoid naming conflicts
  (same names in different scopes ok)
- run-time optimization by compiler

There are two types of local scopes

- hard and soft (functions are hard, loops are soft)

Different rules for how a local variable is assigned if there is already a global one with the same name

Illustrated in scope.jl and scoperror.jl; see also Julia doc



Some differences between the REPL and running files

Composite types

The constructor ‘struct’ for creating a composite type named System

```
struct System
    size::Int
    temp::Float64
    conf::Array{Int,1}
end
```

These are the **fields** of System

An object of type system can now be created, e.g.,

```
sys=System(a,b,c)
```

where a,b,c must match the field types of System

The fields are accessed as: `sys.size, sys.temp, sys.conf`

There is a function fieldnames() that returns the field names

sys can be passed as an argument to a function like any object

A struct is an unmutable object

- but in sys the array field is still mutable (can be changed in a function)

There is also `mutable struct`

Example in struct.jl