

Creating and using functions

Gordon Erlebacher

Quick Review

- Vector: a container: all elements are same type
- List: a container: elements can be of different type
- Functions: a container that contains a list of instructions to execute
 - a function takes arguments and returns an object (one or more values) to the user

A function that does nothing

```
nop = function() {  
}
```

zero arguments
returns nothing

```
> nop  
  
function() {  
  }  
  
> nop()  
  
NULL
```

Function that returns a number of the user

```
one = function() {  
  return(3)  
}
```

There is a single instruction:
return the number 3 to
the caller of the
function

```
> one
```

```
function() {  
  return(3)  
}
```

```
> one()
```

```
[1] 3
```

Useless function

```
one = function() {  
  return(3)  
}
```

Why useless? Because the function always returns the same number.

To make the function more useful, it could, for example return different numbers?

HOW TO DO THIS?

Functions

- mean of a list
 - return the mean value
- standard deviation of a list
 - return the standard deviation
- return both mean and standard deviation

More functions

- Construction a list of “n” random numbers with mean “m” and standard deviation “s”.
- Square each element of the list, and return the mean value and standard deviation of this new list.

More functions

- Calculate the mean value of all numbers divisible by 7 in the range from n to m . Do this using the function *computeMean7*
- Example usage:
 - *computeMean7(345, 723)*
- Make sure the function works properly
 - test with arguments that can be checked manually
 - *computeMean7(13, 22)*

Default arguments

- Standard deviation biased and unbiased.
- Given “n” numbers, the average is computed by summing them up and dividing by “n”.
- The standard deviation is computed by summing the squares of the difference between the numbers and the mean and dividing by “n” (biased) or “n-1” unbiased.

mean(x, bias=FALSE)

- In most cases, I'd like to divide the standard deviation by "n"
- In some cases, I'd like to divide the standard deviation by "n-1"
- So add a default argument, which won't be used most of the time.

Order of arguments

- `plot(x, y)`
- `plot(x,y,'l')`
- `plot(x,y,type='l')`
- `plot(x,y,type='l',col='red')`
- `plot(type='l',x,y)` *# error*
- `plot(x,y,col='red',type='l')`
- `plot(x,col='red')` *# allowed*
 - see *plot.default* *# ?plot*
- `plot(y,x,col='blue',main='A few random numbers')`

Remarks

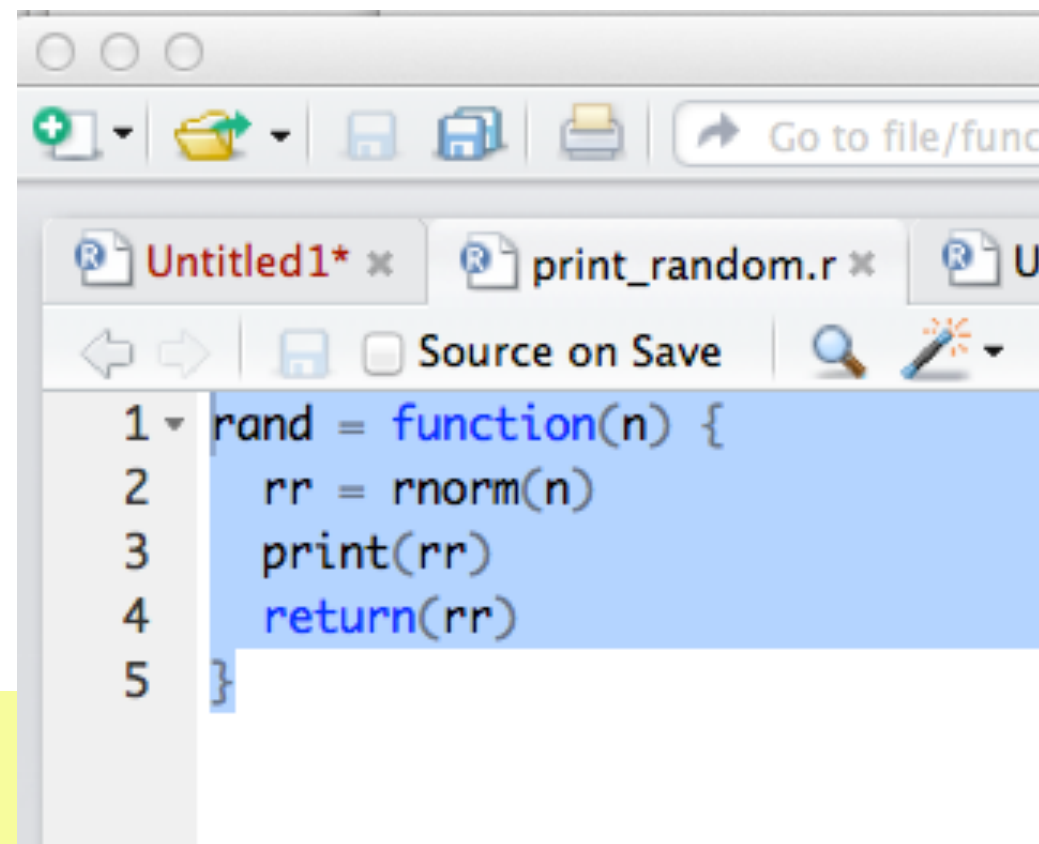
- A function can contain any R instruction
- A function can call other functions, which can call other functions, etc.
- Functions are stored in scripts
- Scripts are executed with the *source("scriptname")* function

Simple script

```
rand = function(n) {  
  rr = rnorm(n)  
  print(rr)  
  return(rr)  
}
```

This function return “n” random numbers, and also prints them to the screen.

```
> getwd()  
[1] "/Users/erlebach"
```



Store the script in a file called `print_random.r`

The script is saved in the folder returned by `getwd()`

source()

- ?source
- Executes the file name used as an argument

```
source(file, local = FALSE, echo = verbose, print.eval =  
echo,  
        verbose = getOption("verbose"),  
        prompt.echo = getOption("prompt"),  
        max.deparse.length = 150, chdir = FALSE,  
        encoding = getOption("encoding"),  
        continue.echo = getOption("continue"),  
        skip.echo = 0, keep.source =  
getOption("keep.source"))
```

source()

file

a [connection](#) or a character string giving the pathname of the file or URL to read from. " " indicates the connection [stdin](#)().

echo

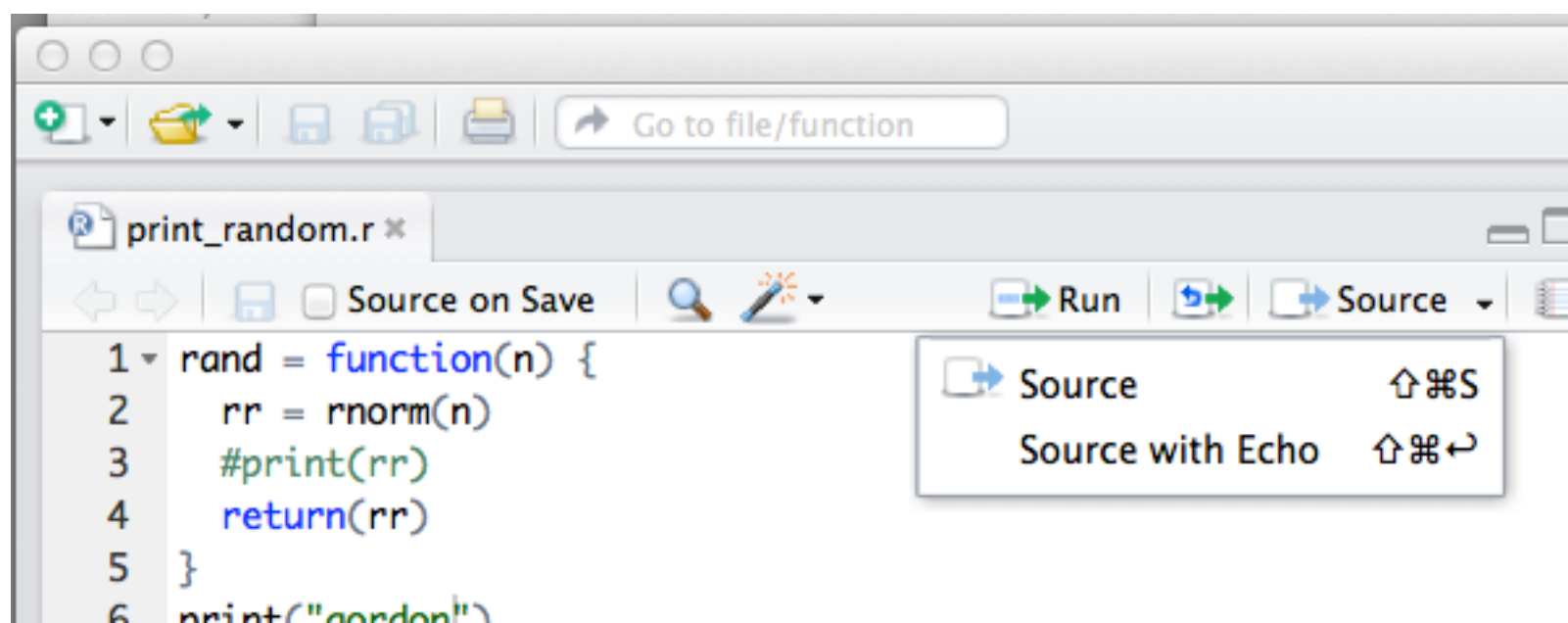
logical; if TRUE, each expression is printed after parsing, before evaluation.

print.eval

logical; if TRUE, the result of `eval(i)` is printed for each expression `i`; defaults to the value of `echo`.

source() function

```
source("print_random.r")  
source("print_random.r",echo=T)  
source("print_random.r", echo=F)
```



Plotting

- Let us play with the “plot” function
- Read the help
- plot lines, symbols, titles, axis labels, etc.
- Create out own functions using plot()

Example functions to create

- Plot a list of “n” random numbers
- Plot a list of “n” random numbers against another list of “n” random numbers
- Create a function “*square*” that computes ‘n’ random numbers between 0 and 1000 and squares these numbers. Create another function “*plot.square()*” that plots these numbers. This new functions should have arguments for the plot title and axis labels.

Internet searching

- In the time left, if any, we will do some internet search with advanced google.