

Functions and data structures

Objects of the game

In R we have objects which are **functions** and objects which are **data**.

Function examples:

- ▶ `sin()`
- ▶ `integrate()`
- ▶ `plot()`
- ▶ `"+"()`
- ▶ `paste()`

Data examples:

- ▶ `42`
- ▶ `1:5`
- ▶ `"R"`
- ▶ `matrix(1:12, nrow=4, ncol=3)`
- ▶ `data.frame(a=1:5, tmt=c("a","b","a","b","a"))`
- ▶ `list(x=2, y="abc", x=1:10)`

Singles

- ▶ Logical, e.g:

```
> TRUE
```

```
[1] TRUE
```

```
> 1==2
```

```
[1] FALSE
```

- ▶ Single numbers, e.g:

```
> 1
```

```
[1] 1
```

```
> 1.2
```

```
[1] 1.2
```

```
> sqrt(as.complex(-1))
```

```
[1] 0+1i
```

- ▶ Character, e.g:

```
> "5"
```

```
[1] "5"
```

```
> "abc"
```

```
[1] "abc"
```

Vectors

- ▶ Vector of numbers, e.g:

```
> c(1, 1.2, pi, exp(1))
```

```
[1] 1.000000 1.200000 3.141593 2.718282
```

- ▶ We can have vectors of other things too, e.g:

```
> c(TRUE, 1==2)
```

```
[1] TRUE FALSE
```

```
> c("a", "ab", "abc")
```

```
[1] "a" "ab" "abc"
```

- ▶ But not combinations, e.g:

```
> c("a", 5, 1==2)
```

```
[1] "a" "5" "FALSE"
```

Notice that **R** just turned everything into characters!

Factor

- ▶ A special kind of vector is called a factor
- ▶ It has a known finite set of levels (options), e.g:

```
> gl(2,10, labels=c("male", "female"))
```

```
[1] male   male   male   male   male   male   male  
[8] male   male   male   female female female female  
[15] female female female female female female  
Levels: male female
```

- ▶ One could also do

```
> as.factor(c(rep("male",10),rep("female",10)))
```

```
[1] male   male   male   male   male   male   male  
[8] male   male   male   female female female female  
[15] female female female female female female  
Levels: female male
```

Matrix, and arrays

- ▶ Similar to vectors we can have matrices of objects of the same type, e.g:

```
> matrix(c(1,2,3,4,5,6)+pi,nrow=2)
```

```
      [,1]      [,2]      [,3]  
[1,] 4.141593 6.141593 8.141593  
[2,] 5.141593 7.141593 9.141593
```

```
> matrix(c(1,2,3,4,5,6)+pi,nrow=2)<6
```

```
      [,1] [,2] [,3]  
[1,] TRUE FALSE FALSE  
[2,] TRUE FALSE FALSE
```

Matrix, and arrays

- We can create higher order arrays, e.g:

```
> array(c(1:24), dim=c(4,3,2))
```

```
, , 1
```

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

```
, , 2
```

	[,1]	[,2]	[,3]
[1,]	13	17	21
[2,]	14	18	22
[3,]	15	19	23
[4,]	16	20	24

Data frame

- ▶ A special data object is called a data frame (`data.frame()`):
- ▶ We can create data frames by reading data in from files;
- ▶ or by using the function `as.data.frame()` on a set of vectors.
- ▶ A data frame is a set of parallel vectors, where the vectors can be of different types, e.g:

```
> data.frame(treatment=c("active","active","placebo"),  
+            bp=c(80,85,90))
```

```
  treatment bp  
1   active 80  
2   active 85  
3 placebo 90
```

- ▶ Compare to a matrix

```
> cbind(treatment=c("active","active","placebo"),  
+        bp=c(80,85,90))
```

```
  treatment bp  
[1,] "active" "80"  
[2,] "active" "85"  
[3,] "placebo" "90"
```


Lists

- ▶ Most general object type
- ▶ Elements can be of different types and lengths, e.g:

```
> list(a=1,b="abc",c=c(1,2,3),d=list(e=matrix(1:4,2), f=function(x)x^2))
```

```
$a
```

```
[1] 1
```

```
$b
```

```
[1] "abc"
```

```
$c
```

```
[1] 1 2 3
```

```
$d
```

```
$d$e
```

```
      [,1] [,2]  
[1,]     1     3  
[2,]     2     4
```

```
$d$f
```

```
function (x)  
x^2
```

- ▶ The objects returned from many of the built-in functions in **R** are fairly complicated lists.

Functions

Let us consider the examples of functions that we listed at the start of the session:

Function examples:

- ▶ `sin()`
- ▶ `integrate()`
- ▶ `plot()`
- ▶ `"+"()`
- ▶ `paste()`

Typical input and output

- ▶ `sine()`: Input number, output number.
- ▶ `integrate()`: Input function and range, output list.
- ▶ `plot()`: Input vector, output NULL.
- ▶ `"+"()`: Input two numbers, output one number.
- ▶ `paste()`: Input character objects, output one character object.

Defining a simple function

The functions on the previous slide are **built-in functions**.

- ▶ it is simple to write your own functions

- ▶ A square function

```
> square<-function(x){  
+   return(x*x)  
+ }  
> square(1:5)  
  
[1] 1 4 9 16 25
```

- ▶ A power function with two arguments

```
> power<-function(x,pow){  
+   return(x^pow)  
+ }  
> power(1:5,3)  
  
[1] 1 8 27 64 125
```

Default arguments

- ▶ A function can have default arguments

```
> power<-function(x,pow=2){  
+   return(x^pow)  
+ }
```

```
> power(1:5)
```

```
[1]  1  4  9 16 25
```

```
> power(1:5,4)
```

```
[1]  1 16 81 256 625
```

Masking functions

Be careful when you write functions.

- ▶ Masking the `c()` function:

- ▶

```
> c<-function(x,y) x*y
```

```
> c(2,3)
```

```
[1] 6
```

```
> rm(c)
```

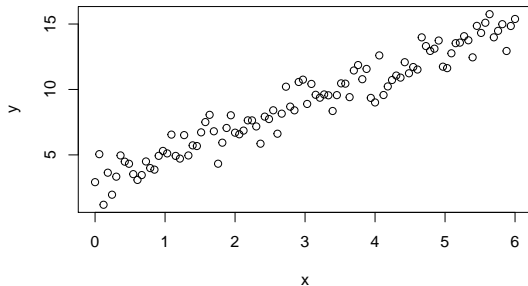
```
> c(2,3)
```

```
[1] 2 3
```

- ▶ If you use a reserved function identifier, you mask the function.

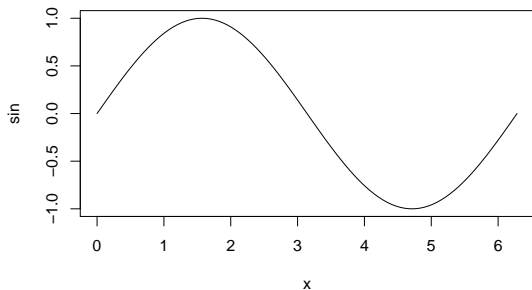
Simple plotting

```
> x<-seq(0,6, length=100)  
> y<-2*x+3+rnorm(100)  
> plot(x,y)
```



Simple plotting

```
> plot(sin,0,2*pi)
```



Editor and the function `source()`

- ▶ If you write programs spanning more than a few lines it is convenient to write them in an editor.
- ▶ A frequently used approach is to write your code in the editor and then paste blocks into R to run it.
- ▶ Once the script is complete, the file is saved, and we can run it all by typing:

```
> source("C:/programdir/script.R")
```
- ▶ Lines starting with “#” are ignored by R and can be used to insert comments in the script.