R CRASH COURSE

R Syntax

R language uses a syntax reminiscent of basic arithmetic and function notation. For example,

`(2*a+3)^5`	is the value of an arithmetic expression $(2a+3)^5$
`sin(2)`	is the sine of 2
`max(1, 2, 3, -4)`	is 3, which is the maximum value given to the function

There are a few differences when compared with the standard algebraic notation: all multiplications are shown explicitly as $*$, and variable names can be many characters long.

For example, `3a` is completely wrong, but `3*a` is a product of 3 and a variable `a`. An expression such as `xy + 5` is a sum of a single variable named `xy` and 5, whereas `x*y + 5` is the product of `x` and `y`, plus 5.

In the following examples, we will use a blue text to highlight things you would enter into R, and green text for things R would print back at you. We also put spaces around some operations, but this is only to improve readability. R mostly ignores spaces in arithmetic expressions.

Storing Data For Later Use

To create a variable and store some value in it, use the assignment operator = like so:

```
foo = 17
```

The equality sign is an instruction to put the value of the expression on the right into the variable with the given name. You can print the value of a variable by entering its name:

```
foo
```

```
[1] 17
```

Besides the value of `foo`, this output includes `[1]`, which annotates the output. The basic data type of R language is not a number, but a vector: a sequence of data values such as numbers or literal strings. The word "vector" is rather technical, so it may be helpful to think of these as data sets, or single columns in a spreadsheet file.

Try creating a data set consisting of integers from 50 to 100 and storing it in a variable `v`:

```
v = 50:100
```

`:` is an operator which creates integer sequences. Here it creates a data set of integers from 50 up to and including 100. Now print the value of `v` with

```
v
```

and you will see something like

```
 [1]  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
[20]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87
[39]  88  89  90  91  92  93  94  95  96  97  98  99 100
```

This is the way R displays data sets. The numbers in brackets on the left side are not a part of the data. They are indices of the leftmost data points, so 50 is the 1st data point, 69 is the 20th, and 88 is the 39th data point in the data set.

Arithmetic

R supports all the standard arithmetic operations:

algebra	**R syntax**
$a+b$	`a+b`
$a-b$	`a-b`
ab	`a*b`
a/b	`a/b`
a^b	`a^b`

Many other mathematical operations are implemented as functions. Here are some of the ones we will find useful:

`sqrt(x)`	$\sqrt{x}$
`abs(x)`	$\lvert x\rvert$
`mean(x)`	arithmetic average of the elements of data set x
`ceiling(x)`	round x up to the closest integer at or above it

Care should be taken as you translate from the standard algebraic notation and into something R can understand. In particular, all multiplications, exponents, and parentheses must be made visible. For example, a fraction such as

$$\frac{3a+b}{c+1}$$

can be written as

```
(3*a + b)/(c + 1)
```

Omitting any of these parentheses would alter the order of operations, producing a wrong value. For example,

```
3*a + b/c + 1
```

is equivalent to $3a+\frac{b}{c}+1$, which is a completely different expression.

Here are a few more examples of familiar algebraic expressions and their R equivalents:

algebraic expression	**R syntax**
$\frac{a-2b}{2}$	`(a-2*b)/2`
$\sqrt{\frac{p(1-p)}{n}}$	`sqrt(p*(1-p)/n)`
$\frac{4(a-b)}{1+\sqrt{\frac{x^2}{n}+\frac{y^2}{m}}}$	`4*(a-b)/(1 + sqrt(x^2/n + y^2/m))`

Data Set Manipulation

Since many statistical functions operate on data sets, R provides many different ways to either input or generate them. Function c will create a data set with given values:

```
c(x, y, z, ...)
```

Try creating a data set and saving it into a variable called d:

```
d = c(-2, 0, 4, -9, -4, -2, 6)
```

Now you can manipulate this sample in a variety of ways. For example, here's a data set consisting of absolute values of the data in d:

```
abs(d)
```

```
[1] 2 0 4 9 4 2 6
```

Here's a data set of squares of the data in d:

```
d^2
```

```
[1]  4  0 16 81 16  4 36
```

The following table lists a few other ways to manipulate the data we may find useful:

function	**result**
`2*d`	a data set where each value is doubled
`length(d)`	number of data values
`sort(d)`	data in ascending order
`sum(d)`	sum of the data
`sqrt(d)`	a data set of square roots of original values
`max(d)`	the greatest data point
`min(d)`	the least data point

Fun With R

R is very good at generating “random” data sets of many different types, which is quite useful for visualizing random processes and testing our statistical hypotheses.

The `sample` function can choose so many random entries out of a data set. In the example below we choose 5 random integers between 60 and 90. Of course, if you try repeating this work, R will very likely produce a different selection every time you try:

```
sample(60:90, 5)
```

```
  [1] 77 80 75 84 72
```

Another way to generate random data sets is via the uniform sampling. The following example will select 5 random rational numbers between 10 and 20. Again, your output will almost certainly be different every time:

```
runif(5, 10, 20)
```

```
  [1] 19.83321 18.40819 19.09421 17.00301 13.86644
```

Another powerful feature of R is how it applies its operations and functions to each element of a data set.

Suppose we want to compute the sum of the first 100 positive integers. A task like this is very easy in R even if you don’t know any algebraic shortcuts.

Recall that the expression `a:b` produces a data set of integers from `a` to `b`:

```
1:100
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

Finding the sum for this data set solves our problem:

```
sum(1:100)
```

```
[1] 5050
```