Numeric

Decimal values are called numeric in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, the class of x will be numeric.

```
> x = 10.5        # assign a decimal value
> x               # print the value of x
[1] 10.5

> class(x)        # print the class name of x
[1] "numeric"
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
> k = 1
> k               # print the value of k
[1] 1

> class(k)        # print the class name of k
[1] "numeric"
```

The fact that k is not an integer can be confirmed with the is.integer function. This is a consequence of the lack of class inheritance relationship between the numeric and integer data types.

```
> is.integer(k)   # is k an integer?
[1] FALSE
```

Integer

In order to create an integer variable in R, we invoke the as.integer function. We can be assured that y is indeed an integer by applying the is.integer function.

```
> y = as.integer(3)
> y                       # print the value of y
[1] 3

> class(y)                # print the class name of y
[1] "integer"

> is.integer(y)           # is y an integer?
[1] TRUE
```

Incidentally, we can coerce a numeric value into an integer with the same as.integer function.

```
> as.integer(3.14)      # coerce a numeric value
[1] 3
```

And we can parse a string for decimal values in much the same way.

```
> as.integer("5.27")  # coerce a decimal string
[1] 5
```

On the other hand, it is erroneous trying to parse a non-decimal string.

```
> as.integer("Joe")    # coerce an non-decimal string
[1] NA

Warning message:
NAs introduced by coercion
```

Often, it is useful to perform arithmetic on logical values. Like the C language, TRUE has the value 1, while FALSE has value 0.

```
> as.integer(TRUE)    # the numeric value of TRUE
[1] 1

> as.integer(FALSE)   # the numeric value of FALSE
[1] 0
```

Complex

A complex value in R is defined via the pure imaginary value i.

```
> z = 1 + 2i     # create a complex number
> z              # print the value of z
[1] 1+2i

> class(z)       # print the class name of z
[1] "complex"
```

The following gives an error as -1 is not a complex value.

```
> sqrt(-1)       # square root of -1
[1] NaN

Warning message:
In sqrt(-1) : NaNs produced
```

Instead, we have to use the complex value -1 + 0i.

```
> sqrt(-1+0i)    # square root of -1+0i
[1] 0+1i
```

An alternative is to coerce -1 into a complex value.

```
> sqrt(as.complex(-1))
[1] 0+1i
```

Logical

A logical value is often created via comparison between variables.

```
> x = 1; y = 2    # sample values
> z = x > y       # is x larger than y?
> z               # print the logical value
[1] FALSE

> class(z)        # print the class name of z
[1] "logical"
```

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

```
> u = TRUE; v = FALSE
> u & v          # u AND v
[1] FALSE

> u | v          # u OR v
[1] TRUE

> !u             # negation of u
[1] FALSE
```

Further details and related logical operations can be found in the R documentation.

```
> help("&")
```

Character

A character object is used to represent string values in R. We convert objects into character values with the as.character() function:

```
> x = as.character(3.14)
> x                    # print the character string
[1] "3.14"

> class(x)             # print the class name of x
[1] "character"
```

Two character values can be concatenated with the paste function.

```
> fname = "Joe"; lname ="Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

However, it is often more convenient to create a readable string with the sprintf function, which has a C language syntax.

```
> sprintf("%s has %d dollars", "Sam", 100)
[1] "Sam has 100 dollars"
```

To extract a substring, we apply the substr function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

```
> substr("Mary has a little lamb.", start=3, stop=12)
[1] "ry has a l"
```

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

```
> sub("little", "big", "Mary has a little lamb.")
[1] "Mary has a big lamb."
```

More functions for string manipulation can be found in the R documentation.

```
> help("sub")
```