Data Types in R

Creating objects

R stores both data and output from data analysis (as well as everything else) in objects.

Creating a new object is as easy as typing the object's name and assigning a value to it. There are multiple ways to assign values to objects in R.

As in many computer languages you can use the equal sign (=) as an assignment operator. Copy the following code into your script, and run it.

```
#assign the value of the square root of 100 to
#the object, variable 'result'
result = sqrt(100)
```

You are more likely, however, to see <- used to assign values to objects:

```
#assign the value of the square root of 100 to
#the object, variable 'result'
result <- sqrt(100)
```

Printing objects

Now we have created an object called result. It currently has the value of 10. To verify that, we can type the name of the object, and R will print the value stored for it:

```
result
## [1] 10
```

You can think of an object in R as a shoebox (object) with a name. Here, the name of the shoebox is "result." Imagine that it is holding a scrap of paper with the number 10 on it.

Naming objects

You can name your object anything you want, with a few exceptions.
You can store any value in it, with a few exceptions.
For instance we could have named it `ScoobyDoo` if we wanted, but that's not a very informative name.

EVERYTHING IN R IS AN OBJECT.

R has six basic data types:

- character: `"a"`, `"swc"`

R has six basic data types:

- character: `"a"`, `"swc"`
- numeric: `2`, `15.5`

R has six basic data types:

- character: `"a"`, `"swc"`
- numeric: `2`, `15.5`
- integer: `2L` (the `L` tells R to store this as an integer)

R has six basic data types:

- character: `"a"`, `"swc"`
- numeric: `2`, `15.5`
- integer: `2L` (the `L` tells R to store this as an integer)
- logical: `TRUE`, `FALSE`

R has six basic data types:

- character: `"a"`, `"swc"`
- numeric: `2`, `15.5`
- integer: `2L` (the `L` tells R to store this as an integer)
- logical: `TRUE`, `FALSE`
- complex: `1+4i` (complex numbers with real and imaginary parts)

Inspecting an object's data type

R provides many functions to examine features of objects. Two important ones:

- `class()` - what kind of object is it (high-level)?

Inspecting an object's data type

R provides many functions to examine features of objects. Two important ones:

- `class()` - what kind of object is it (high-level)?
- `length()` - how long is it?

For example, we can get some information about the `result` object that we just created:

```
#find class of object 'result'
class(result)
## [1] "numeric"

#find length of object 'result'
length(result)
## [1] 1
```

DRILLS

1. Pass the square root of 100 to the object `sqrt100`. 1a. What is the class of this object?

DRILLS

1. Pass the square root of 100 to the object `sqrt100`. 1a. What is the class of this object?
2. `R` comes loaded with an object called `precip`. What is the class and length of this object?

Data Structures in R

There are several different data structures in R. If you take a longer introduction to R you'll learn about arrays, lists, matrices, etc.

We will skip to the two most common data types: vectors and data frames.

Vectors

Guess what, *you've been using them already!*

Vectors are the building block of R. Simply, they are one-dimensional collections of values of the same type (integer, character, logical, etc.)

Let's build a few vectors from scratch using the `c()` function...

```
#A vector of numbers
x<-c(5,9,11)
x
## [1]  5  9 11
```

```r
#A vector of logical values
y<-c(TRUE,FALSE,TRUE,FALSE,FALSE)
y
## [1]  TRUE FALSE  TRUE FALSE FALSE
```

```r
#A vector of character strings
people<-c("Jack","Jill","Jim","June")
people
## [1] "Jack" "Jill" "Jim"  "June"
```

```r
#A vector with one value -- a "scalar"
v<-c(15)
v
## [1] 15
```

The elements of a vector must **all have the same mode, or data type.**

You can have a vector consisting of three character strings, or three integer elements, but not a vector with one integer element and two character string elements.

```
k <- c(1,2,3,"BOO!")
k
## [1] "1"    "2"    "3"    "BOO!"
```

Accessing values of a vector

Once your vector is defined, you will want to access values from it. For example, you may want to get the fourth value in the vector, or the second to last, or so forth.

The syntax for accessing values is `vectorname[index]` where vectorname is the name of the vector, and index is the index number of the element you are looking for.

Using the `people` vector we created earlier, here are some examples.

```r
#print the people vector just for reference
people
## [1] "Jack" "Jill" "Jim"  "June"
```

```r
#get the first value from the vector
people[1]
## [1] "Jack"
```

```r
#R will evaluate any math within the brackets...
people[1+2]
## [1] "Jim"
people[length(people)]
## [1] "June"
```

```r
#print the 1st and 3rd values in different order
people[c(1,3)]
## [1] "Jack" "Jim"
people[c(3,1)]
## [1] "Jim"  "Jack"
```

DRILLS

1. Create a vector `names` containing the names of five people sitting near you. (If you don't know their names, make them up.) 1a. Access the 1st and 4th values of this vector.

DRILLS

1. Create a vector `names` containing the names of five people sitting near you. (If you don't know their names, make them up.) 1a. Access the 1st and 4th values of this vector.
2. Create two vectors `x` and `y` of length 4; one containing numeric and the other containing logical values. Multiply them together and pass this result to `z`. What is the result?

Data Frames

Datasets for statistical analysis are typically stored in data frames in R.

Data frames are rectangular, where the columns are variables and the rows are observations of those variables. This is very similar to a table in Excel or in a database.

Columns of data frames are. . . **vectors**!

And, like vectors, data frame columns can be of several different data types, but all entries in the same vector must be the same type.

Also, all columns in a data frame must be of the same equal length.

Creating with data.frame()

Data frames can be manually created with `data.frame()`.
The elements of a data frame are almost always named.

```
mydata <- data.frame(diabetic = c(TRUE, FALSE, TRUE, FALSE),
                     height = c(65, 69, 71, 73))
mydata
##   diabetic height
## 1     TRUE     65
## 2    FALSE     69
## 3     TRUE     71
## 4    FALSE     73
```

Subsetting dataframes

Data frames, unlike vectors, are two-dimensional structures. We subset them using the formula [rows, columns].

```
# row 3 column 2
mydata[3,2]
## [1] 71
```

```r
# using column name
mydata[1:2, "height"]
## [1] 65 69
```

```r
# all rows of column "height"
mydata[,"diabetic"]
## [1]  TRUE FALSE  TRUE FALSE
```

DRILLS

1. Continuing on the `mydata` dataframe, access all values in the `height` column.

DRILLS

1. Continuing on the `mydata` dataframe, access all values in the `height` column.
2. Access all values in the 2nd and 4th rows of `mydata`.

Subsetting entire columns

To subset an entire column of a dataframe, we use $ followed by that column name.

```
# subsetting creates a numeric vector
mydata$height
## [1] 65 69 71 73
```

This function will be *extremely important* when we come to do operations on the data.

Naming dataframe columns

`colnames(data_frame)` returns the column names of the data frame.
`colnames(data_frame) <- c("some", "names")` will assign column names to the data frame.

```r
# get column names
colnames(mydata)
## [1] "diabetic" "height"
```

```r
# assign column names
colnames(mydata) <- c("Diabetic", "Height")
colnames(mydata)
## [1] "Diabetic" "Height"
```

To change one column name, just use indexing.

```
colnames(mydata)[1] <- "Diabetes"
colnames(mydata)
## [1] "Diabetes" "Height"
```

DRILLS

1. Rename the column names of `mydata` to `D` and `H`.

Questions?