LECTURE 4: DATA STRUCTURES IN R (contd)

STAT598z: Intro. to computing for statistics

Vinayak Rao

Department of Statistics, Purdue University

```
In [ ]: options(repr.plot.width=3, repr.plot.height=3)
```

Data frames

Very common and convenient data structures

Used to store tables:

- Columns are variables and rows are observations

	Age	PhD	GPA
Alice	25	TRUE	3.6
Bob	24	TRUE	3.4
Carol	21	FALSE	3.8

An R data frame is a list of equal length vectors

```
In [ ]: df <- data.frame(age = c(25L,24L,21L),  # Warning: df is an
                         PhD = c( T , T , F ),  #   R function
                         GPA = c(3.6,2.4,2.8))
```

```
In [ ]: print(df)
```

```
In [ ]: typeof(df)
```

```
In [ ]: class(df)
```

Since data frames are lists, we can use list indexing

Can also use matrix indexing (more convenient)

```
In [ ]: print(df[2,'age'])
```

```
In [ ]: print(df[2,])
```

In [ ]:
```
print(df$GPA)
```

In [ ]:
```
nrow(df)*ncol(df)
```

list functions apply as usual

matrix functions are also interpreted intuitively

Useful functions are:

- 'length(), dim(), nrow(), ncol()'
- 'names()' (or 'colnames()')', rownames'
- 'rbind(), cbind()'

In [ ]:
```
rownames(df) <- c("Alice", "Bob", "Carol")
```

In [ ]:
```
df[4,1] <- 30L; print(df)
```

Many R datasets are data frames

In [ ]:
```
library("datasets")
class(mtcars)
```

In [ ]:
```
print(head(mtcars)) # Print part of a large object
```

Tibbles

Tibbles are essentially dataframes, with some convenience features

Interact will with the tidyverse package (later)

In [ ]:
```
library(tidyverse)
t_mtcars <- as_tibble(mtcars)
class(t_mtcars)
```

Tibbles print more nicely that dataframes (but see RStudio's View())

In [ ]:
```
print(t_mtcars)
```

You can reference columns of a tibble as you create it

In [ ]:
```
sin_tb <- tibble(x=seq(-5,5,.1), y=sin(x));
print(sin_tb)
```

Factors

Categorical variables that take on a finite number of values

- **Employee type**: `student/staff/faculty`
- **Grade**: `A/B/C/F`

Useful when variable can take a fixed set of values (unlike character strings)

R implements these internally as integer vectors

Has two attributes to distinguish from regular integers:

`levels()` specifies possible values the factor can take

- E.g. `c("male", "female")`

`class = factor` tells R to check for violations

```
In [ ]: # Character vector for 4 students
        grades_bad <- c("a", "a", "b", "f")
```

```
In [ ]: # Factor vector for 4 students
        grades <- factor(c("a", "a", "b", "f"))
```

```
In [ ]: print(grades);
```

```
In [ ]: typeof(grades)
```

```
In [ ]: class(grades)
```

```
In [ ]: levels(grades) # Not quite what we wanted!
```

```
In [ ]: grades <- factor(c("a", "a", "b", "f"))
        str(grades)
```

```
In [ ]: grades[2] <- "c"
```

```
In [ ]: str(grades)
```

```
In [ ]: grades <- factor(c("a","a","b","a","f"),
                     levels = c("a","b","c","f"))
```

```
In [ ]: str(grades)
```

```
In [ ]: table(grades)   # table also works with other data-types
```

Factors can be ordered:

```
In [ ]: grades <- factor(c("a","a","b","f"),
                    levels = c("f","c","b","a"),
                    ordered = TRUE )
        grades
```

```
In [ ]: grades[1] > grades[3]
```

`gl()`: Generate factors levels

Usage (from the R documentation):

```
gl(n, k, length = n * k, labels = seq_len(n),
   ordered = FALSE )
```

Look at the examples there:

```
In [ ]: # First control, then treatment:
        gl(2, 8, labels = c("Control", "Treat"))
```

```
In [ ]: gl(2, 1, 20) # 20 alternating 1s and 2s
```

```
In [ ]: gl(2, 2, 20) # alternating pairs of 1s and 2s
```