What We Expect You to Know

- Construct with (`as.`)`data.table()` or `setDT()`
- `dt[i, j]`
- Both `i` and `j` are evaluated inside the data.table
  - exceptions: `i` with single symbols, `j` with constants
- `i` selects rows `logical()` or `numeric()` values
- `j` creates a new data.table out of lists (use `.()` as shortcut) or returns atomics
  - use `with = FALSE` to select columns directly
- Use `:=` in `j` to create or change columns
  - use `` `:=`() `` to assign multiple columns at once
  - use `(...) := ...` to assign to columns dynamically; possibly multiple at once
- Give both `i` and `j` to have the `j`-expression evaluated on a subset of the original table
- This all "adds up to normality" most of the time
- But you can get very elaborate in your expressions!
- For more useful printing, set

```
options(datatable.print.class = TRUE,
        datatable.print.keys = TRUE,
        datatable.print.trunc.cols = TRUE)
```

What We Expect You to Know

`data.table`: know how to...

- tell if an object is a `data.table` or just a `data.frame`
- change how data.table objects are printed
  - by giving `datatable.print.xxx` arguments to `print()` as described in `?print.data.table`
  - by using `options(datatable.print.xxx = OPTION)` to set the option globally
- get a `DT` from `data.frame`s, matrices, `list`s of rows: `as.data.table`, `setDT()`, `rbindlist()`
- get individual rows, columns, elements from a `DT`
- subset a DT: specific columns, specific rows, rows by a condition
- modify or add new columns based on calculations done on old columns using `` `:=` `` or `set()`
  - single col at a time & `[, (<col group>) := .(<value list>)]`
- handle in-place functions (that start with set... in data.table) as well as the `` `:=` `` operator in `[ ]`), know about reference-semantics and use `copy()` if needed
- use `fread()`, `fwrite()` for fast reading / writing of large files
- do aggregation with `` `by =` ``
  - count subgroup sizes with `.N`
  - calculate aggregate values for subgroups
  - advanced aggregation control with `.SD` and `.SDcols`
  - other special values: `.BY`, `.GRP`, `.NGRP`, `.I`, `.EACHI`
- work with list columns that may contain different kinds of data on different rows
- merge / join data.tables
  - both with `merge()` as well as with `X[Y, ...]` (with `X`, `Y` both being a `DT`)
  - understand the difference between inner, left/right, outer, anti join and how to do them in `DT`
- reshape `DT`s between "wide" format and "long" format using `dcast()` and `melt()`
- use keys
  - what are keys useful for?
    - automatic sorting
    - fast row subsetting
    - row selection using X[<value>]
  - `key()`, `indices()`, `haskey()`
  - difference between `setkey()` and `setindex()`
  - difference between `setkey()`/`setindex()` and `setkeyv()`/`setindexv()`

What We Expect You to Know

`data.table`: know about... (grey: not that important at our level)

- using `[]` as a suffix to print `data.table` in-place operation results even when they are "invisible"
- functions that treat `DT`s like sets of rows to do set operations and sorting on them
  - `fintersect()`, `fsetdiff()`, `fsetequal()`, `funion()`: set-operations that treat data table rows as sets
  - `duplicated()`, `unique()`, `anyDuplicated()`: find duplicate rows / restrict to unique rows
    - `uniqueN()`: short for `nrow(unique(x))`
    - also note these have a "`by`" argument
  - `frank()`, `frankv()`: `rank()` on data.table
  - `split()`: split data.table into list of smaller tables (but it is usually better do do aggregate operations with '`by`' in `[ ]`.)
  - `na.omit()`: exclude rows with NAs
- Further `set...()` functions
  - `setattr()`, `setnames()` -- change attributes by reference
  - `setcolorder()` -- reorder columns
  - `setorder()`, `setorderv()` -- reorder rows, similar to `setkey()`/`setkeyv()`, but without setting a key
- helpful operators for the `i` (i.e. row-selector) argument
  - `between()`, `%between%` -- between to values
  - `inrange()`, `%inrange%` -- in any of multiple ranges
  - `like()`, `%like%`, `%flike%`, `%ilike%`: faster `grepl()`; `%flike%`: fixed (not regex), `%ilike%`: ignores case
- general helper functions
  - `first()`, `last()` -- like `head()`/`tail()`, but get just one item
  - `shift()` -- lead or lag a vector
  - `transpose()` -- transpose lists, `data.frames`, `data.tables`
  - `tstrsplit()` -- `transpose()` of `strsplit()`
  - `fcoalesce()`: vectorized: give first non-NA value
  - `nafill()`, `setnafill()` -- fill missing values
  - `CJ()` -- cross product DT
- System info functions and global settings
  - `address()` -- address of an object
  - `setDTthreads()`, `getDTthreads()` -- change cpu parallelization threads
  - `tables()` -- summarize metadata of all 'data.table' objects in memory
  - `getNumericRounding()`, `setNumericRounding()` -- rounding mode for equality checks
  - `timetaken()` -- time difference to result of call `proc.time()`

What We Don't Really Expect You To Know, But Include Here for Completeness Sake

- fast version of R function, optimized for character vectors
  - chgroup(): like order(), but only groups together duplicates instead of sorting
  - chmatch(): character version for match()
  - chorder(): character version of order()
  - %chin%: character version of %in%
- other fast / more robust versions of R functions
  - fifelse(): ifelse(), preserves attributes
  - frank(), frankv(): faster rank(), but also ranks lists, data.frames and data.tables
- Helpers for aggregation and joining
  - groupingsets(), rollup(), cube() -- aggregate by different columns
  - Id column generators
    - rowid(), rowidv(): unique rowid
    - rleid(), rleidv(): run-length encoding
    - SJ(), CJ(): Join helpers
- Experimental (usage of these functions might change)
  - foverlaps(): fast overlap join
  - truelength(), alloc.col(), setalloccol(): over-allocation of column memory
  - frollmean(), frollsum(), frollapply(): rolling window aggregates
  - fsort(): faster sort through multicore
  - (Experimental) date/time class -- mostly a wrapper for POSIXct and Date
    - IDate, ITime: classes
    - as.IDate(), as.ITime(), IDateTime(): conversion
    - year(), quarter(), month(), week(), isoweek(), yday(), mday(), wday(), hour(), minute(), second(): get specific aspect from object