Wednesday session notes

Summary

We will talk with roughly equal importance about both function-writing and lifetables.

Functions

Anything in `R` with `()` after its name is a function.

```
c(1,2)
```

```
## [1] 1 2
```

```
sum()
```

```
## [1] 0
```

Anatomy of a function. First we assign to a name, which will become the name of our function. We use `function()` to make it. Inside the round parentheses we define argument names. These can be anything, and not need to refer to actual objects sitting in your `R` session. These will only be used internally in the body of the function. The body of the function is everything that happens between the curly braces. Whatever you do in the curly braces *should* only refer to things explicitly *passed in* from the defined arguments.

```
fun_1 <- function(arg1, arg2){
  arg1 ^ 2 + arg2
}
# fun_1(arg1 = , arg2 = )
fun_1(arg1 = 2, arg2 = 10)
```

```
## [1] 14
```

```
fun_1(2, 10)
```

```
## [1] 14
```

```
fun_1(arg2 = 10, arg1 = 2)
```

```
## [1] 14
```

```
fun_1(10, 2)
```

```
## [1] 102
```

Note, when using a function you should probably give its arguments by name, and not merely by order. If you don't name the arguments when calling the function then they will be interpreted in order.

```
fun_2 <- function(x, y){
  x2 <- x^2 - x
  x2 * y
}
fun_2(x = 5, y = 3)
```

```
## [1] 60
```

Observe: neither `x`, nor `y` nor `x2` were ever created in our environment, nor were they ever available to us in an interactive way. They only existed temporarily inside the function, which *only* returns a result. Note in the above two examples, we return the result as the last thing evaluated. The `x2` that was assigned, is just a temporary line, and is not returned.

We can be explicit about returning things from the function by using `return()`.

```
fun_3 <- function(x, y){
  step1 <- fun2(x = x, y = y)
  out   <- step1 / x + y
  return(out)
}
```

Also, we are free to use other functions inside of our functions, as long as they are somehow available. Her ewe make `readr` function available to us withg `library()`

```
library(readr)
fun_4 <- function(x){
  x <- parse_number(x)
  x ^ 2
}
fun_4(x = "x4")
```

```
## [1] 16
```

But you can also grab a function from a package without needing to load the package, a nice thing to understand. Just use `::`

```
fun_5 <- function(x){
  x <- readr::parse_number(x)
  x ^ 2
}
fun_5("x4")
```

```
## [1] 16
```

You can return more than one thing, or something with an interesting dimension:

```
fun_6 <- function(DF){
  DF$z <- DF$x * DF$y
  DF
}
my_DF <- data.frame(x = rnorm(10),
                    y = runif(10, min = -5, max = 5))
my_DF
```

```
##              x           y
## 1   0.12973015  1.77904699
## 2   0.36075547 -1.87328478
## 3   0.41900103  0.77277771
## 4  -0.16735556 -0.03258761
## 5  -1.92288570  4.84223049
## 6  -0.91058866  0.88471906
## 7   0.02249537 -1.47047894
## 8   0.50340262  4.80722124
## 9   1.48179751 -3.16342945
## 10  0.34070526  0.95856762
```

```
fun_6(my_DF)
```

```
##              x           y            z
## 1   0.12973015  1.77904699  0.230796033
## 2   0.36075547 -1.87328478 -0.675797732
## 3   0.41900103  0.77277771  0.323794656
## 4  -0.16735556 -0.03258761  0.005453717
## 5  -1.92288570  4.84223049 -9.311055737
## 6  -0.91058866  0.88471906 -0.805615152
## 7   0.02249537 -1.47047894 -0.033078963
## 8   0.50340262  4.80722124  2.419967766
## 9   1.48179751 -3.16342945 -4.687561868
## 10  0.34070526  0.95856762  0.326589031
```

We could redo **fun_6()** using tidyverse instead of base:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v dplyr   1.0.7
## v tibble  3.1.3     v stringr 1.4.0
## v tidyr   1.1.3     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
fun_7 <- function(DF){
  DF %>%
    mutate(z = x * y)
}
fun_7(my_DF)
```

```
##              x           y            z
## 1   0.12973015  1.77904699  0.230796033
## 2   0.36075547 -1.87328478 -0.675797732
## 3   0.41900103  0.77277771  0.323794656
## 4  -0.16735556 -0.03258761  0.005453717
## 5  -1.92288570  4.84223049 -9.311055737
## 6  -0.91058866  0.88471906 -0.805615152
## 7   0.02249537 -1.47047894 -0.033078963
## 8   0.50340262  4.80722124  2.419967766
## 9   1.48179751 -3.16342945 -4.687561868
## 10  0.34070526  0.95856762  0.326589031
```

Now it's time for lifetables

```
library(tidyverse)
library(readr)

path <- "https://raw.githubusercontent.com/timriffe/KOSTAT_Workshop1/master/Data/LT_inputs.csv"
LT <- read_csv(path)
```

```
## Rows: 13395 Columns: 7
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Country, ISO3, Sex
## dbl (4): Year, Age, nMx, nAx

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(LT)
```

```
## Rows: 13,395
## Columns: 7
## $ Country <chr> "Algeria", "Algeria", "Algeria", "Algeria", "Algeria", "Algeri~
## $ ISO3    <chr> "DZA", "DZA", "DZA", "DZA", "DZA", "DZA", "DZA", "DZA", "DZA",~
## $ Year    <dbl> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 20~
## $ Sex     <chr> "f", "f", "f", "f", "f", "f", "f", "f", "f", "f", "f", "f", "f~
## $ Age     <dbl> 0, 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 7~
## $ nMx     <dbl> 0.03443, 0.00145, 0.00070, 0.00046, 0.00065, 0.00078, 0.00094,~
## $ nAx     <dbl> 0.09535144, 2.80888571, 2.50000000, 2.50000000, 2.50000000, 2.~
```

Lifetable transformations as functions

Death probabilities between age x and $x + n$ ${}_nq_x$

$$ {}_nq_x = \frac{n * {}_nM_x}{1 + (n - {}_nA_x){}_nM_x} $$

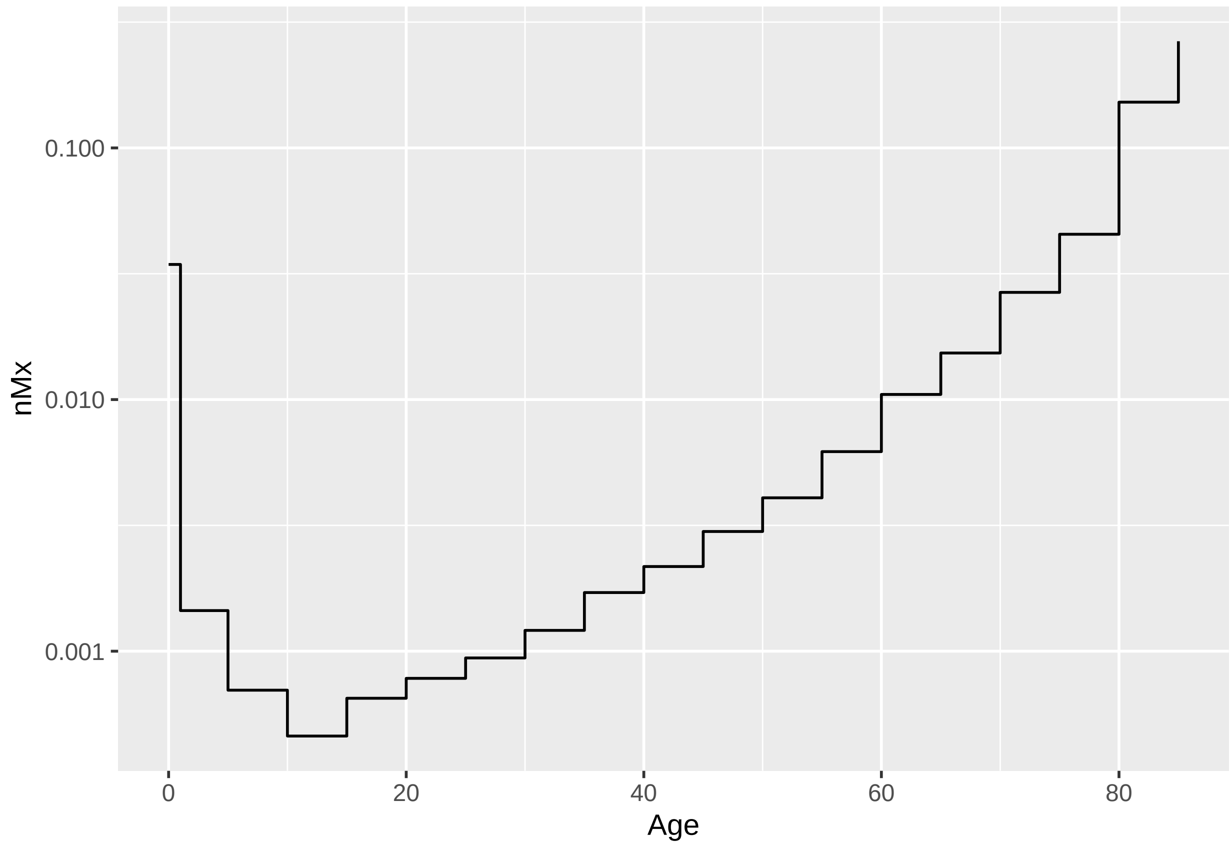
where ${}_nA_x$ is the average number of person-years lived in the interval by those dying in the interval and n is the width of the age-interval.

```
calc_nqx <- function(nMx, nAx, n){
  nqx <- (n * nMx) / (1 + (n - nAx) * nMx)
  nqx[nqx > 1] <- 1
  nqx[nqx < 0] <- 0
  nqx
}

head(LT)
```

```
## # A tibble: 6 x 7
##   Country ISO3   Year Sex     Age     nMx    nAx
##   <chr>   <chr> <dbl> <chr> <dbl>   <dbl>  <dbl>
## 1 Algeria DZA    2000 f         0 0.0344  0.0954
## 2 Algeria DZA    2000 f         1 0.00145 2.81
## 3 Algeria DZA    2000 f         5 0.0007  2.5
## 4 Algeria DZA    2000 f        10 0.00046 2.5
## 5 Algeria DZA    2000 f        15 0.00065 2.5
## 6 Algeria DZA    2000 f        20 0.00078 2.5
```

```
DZA <-
  LT %>%
  filter(Country == "Algeria",
       Sex == "f",
       Year == 2000)

DZA %>%
ggplot(aes(x = Age, y = nMx)) +
```

```
  geom_step() +
  scale_y_log10()
```




```
nMx <- DZA$nMx
nAx <- DZA$nAx
n   <- c(1,4,rep(5,17))

nqx <- calc_nqx(nMx, nAx, n)
```

Survival probabilities between age x and $x + n$, ${}_np_x$

$$ {}_np_x = 1 - {}_nq_x $$

Survival probabilities to age x, l_x

$$ l_{x+n} = r \prod_{y=0}^{x} {}_np_y $$

where $r = {}_nl_0$ is the radix.

```
calc_lx <- function(nqx, radix = 1){
  lx <- c( 1, cumprod(1 - nqx))
  lx <- lx[-length(lx)]
  radix * lx
}
lx <- calc_lx(nqx)
```

Death distribution, ${}_nd_x$

$$ {}_nd_x = {}_nq_x * l_x $$

```
calc_ndx <-function(nqx, lx){
  nqx * lx
}
ndx <- calc_ndx(nqx, lx)
ndx
```

```
##  [1] 0.033390000 0.005596672 0.003357671 0.002200078 0.003100193 0.003706957
##  [7] 0.004439675 0.005684137 0.007977459 0.010049909 0.013632564 0.018246117
## [13] 0.027165070 0.043985816 0.060271892 0.094664618 0.135003977 0.290540878
## [19] 0.233701135
```

Person-years lived between age x and $x + n$, ${}_nL_x$

$$ {}_nL_x = \frac{{}_nd_x}{{}_nM_x} $$

You can think of this identity as relating to occurrence exposure rates, however, you might get a 0 in the denominator, which would need to be handled, so instead we might prefer something that isn't a ratio:

$$ {}_nL_x = n(l_x - {}_nd_x) + {}_nA_x * {}_nd_x $$

And be sure to close out the final value with:

$$ L_{85} = l_{85} + A_{85} $$

```
calc_nLx <- function(ndx, lx, nAx, n){
 n * (lx - ndx) + nAx * ndx
}

nLx <- calc_nLx(ndx, lx, nAx, n)
nLx
```

```
##  [1] 0.9697938 3.8597737 4.7966725 4.7827781 4.7695274 4.7525095 4.7230581
##  [8] 4.6976336 4.6651808 4.6312946 4.5593860 4.4830755 4.3744073 4.1971198
## [15] 3.9367663 3.5494795 2.9749664 1.9113274 0.8803961
```

Person-years lived above age x T_x

$$ T_x = \sum_{y=x}^{\infty} {}_nL_y $$

```
calc_Tx <- function(nLx){
  nLx %>% rev() %>% cumsum() %>% rev()
}
Tx <- calc_Tx(nLx)
```

Life expectancy e_x

$$ e_x = \frac{T_x}{l_x} $$

```
calc_ex <- function(Tx, lx){
  Tx / lx
}
calc_ex(Tx, lx)
```

```
##  [1] 73.515147 75.051316 71.472036 66.713861 61.861723 57.054963 52.268143
##  [8] 47.511775 42.794189 38.147684 33.532881 29.009332 24.559990 20.256526
## [15] 16.212183 12.303514  8.704028  5.292094  3.714966
```

bring it all together

```
LT %>%
  group_by(Country, ISO3, Sex, Year) %>%
  mutate(
    n = case_when(Age == 0 ~ 1,
                  Age == 1 ~ 4,
                  TRUE ~ 5),
    nqx = calc_nqx(nMx = nMx, nAx = nAx, n = n),
    lx = calc_lx(nqx = nqx, radix = 1e5),
    ndx = calc_ndx(nqx = nqx, lx = lx),
    nLx = calc_nLx(ndx = ndx, lx = lx, nAx = nAx, n = n),
    Tx = calc_Tx(nLx),
    ex = calc_ex(Tx = Tx, lx = lx))
```

```
## # A tibble: 13,395 x 14
## # Groups:   Country, ISO3, Sex, Year [705]
##    Country ISO3   Year Sex     Age     nMx    nAx     n     nqx      lx   ndx
##    <chr>   <chr> <dbl> <chr> <dbl>   <dbl>  <dbl> <dbl>   <dbl>   <dbl> <dbl>
##  1 Algeria DZA    2000 f         0 0.0344  0.0954     1 0.0334  100000  3339
##  2 Algeria DZA    2000 f         1 0.00145 2.81       4 0.00579  96661   560.
##  3 Algeria DZA    2000 f         5 0.0007  2.5        5 0.00349  96101.  336.
##  4 Algeria DZA    2000 f        10 0.00046 2.5        5 0.00230  95766.  220.
##  5 Algeria DZA    2000 f        15 0.00065 2.5        5 0.00324  95546.  310.
##  6 Algeria DZA    2000 f        20 0.00078 2.5        5 0.00389  95236.  371.
##  7 Algeria DZA    2000 f        25 0.00094 0.454      5 0.00468  94865.  444.
##  8 Algeria DZA    2000 f        30 0.00121 0.881      5 0.00602  94421.  568.
##  9 Algeria DZA    2000 f        35 0.00171 1.56       5 0.0085   93852.  798.
## 10 Algeria DZA    2000 f        40 0.00217 2.87       5 0.0108   93055. 1005.
## # ... with 13,385 more rows, and 3 more variables: nLx <dbl>, Tx <dbl>,
## #   ex <dbl>
```