Lecture 01 - Solutions

Exercise A - (5 min)

1. Why does this code throw an error? Try to fix it.

```
x <- 3; x > 2 & < 9
```

2. Does `(NA & TRUE)` equal `(NA | TRUE)`? Explain.
3. Does `(Inf - Inf)` equal `(Inf - 1)`? Explain.
4. Run the following. What happens? [(further reading)]

```
y <- (1 - 0.8); z <- 0.2
y == z; y < z; all.equal(y, z); identical(y, z)
```

5. Why do I use double quotes here?

```
important_message <- "The harder you try, the more you'll learn."
```

Solution

```
# Part 1
# Need to write x twice to get two complete statements
x <- 3
(x > 2) & (x < 9)
```

```
[1] TRUE
```

```
# Part 2
NA & TRUE # result unknown since AND is only TRUE if both are TRUE
```

```
[1] NA
```

```
NA | TRUE # since one condition is true, OR is true
```

```
[1] TRUE
```

```
# Part 3
Inf - Inf
```

```
[1] NaN
```

```
Inf - 1
```

```
[1] Inf
```

```
# Part 4
y <- (1 - 0.8)
z <- 0.2
y == z
```

```
[1] FALSE
```

```
y < z
```

```
[1] TRUE
```

```
all.equal(y, z)
```

```
[1] TRUE
```

```
identical(y, z)
```

```
[1] FALSE
```

```
# Part 5
# With single quote, apostrophe in "you'll" would cause problems.
```

Exercise B - (1 minute)

Predict the result that you will obtain if you use `typeof()` to find the type of each of the following atomic vectors. Then check to see if you were right!

```
foo <- c('1', '2', '3')
bar <- c('TRUE', 'FALSE')
```

Solution

```
# They're both character vectors
typeof(foo)
```

```
[1] "character"
```

```
typeof(bar)
```

```
[1] "character"
```

Exercise C - (5 minutes)

```
y <- c('Keble', 'LMH', 'Univ', 'Merton')
```

1. Enter the command `y[5]`. What result do you get? Why?
2. I want to extract the second and fourth elements of `y` so I enter `y[2,4]`. What happens? Can you fix it? How?
3. Select `'Keble'` and `'Univ'` two different ways.
4. Below is a vector of sales in $ over several months. Using `[]`, `length()` and `-`, compute the monthly *growth rates* in %

```
sales <- c(100, 120, 90, 110, 105, 130, 140, 135, 125, 145, 150, 160)
```

Solution

```
y <- c('Keble', 'LMH', 'Univ', 'Merton')

# Part 1
# We get an NA since there is no 5th element
y[5]
```

```
[1] NA
```

```
# Part 2
# Need to enclose 2,4 within c()
y[c(2, 4)]
```

```
[1] "LMH"    "Merton"
```

```
# Part 3
y[c(1, 3)]
```

```
[1] "Keble" "Univ"
```

```
y[-c(2, 4)]
```

```
[1] "Keble" "Univ"
```

```
# Part 4
100 * ((sales[-1] / sales[-length(sales)]) - 1)
```

```
 [1]  20.000000 -25.000000  22.222222  -4.545455  23.809524   7.692308
 [7]  -3.571429  -7.407407  16.000000   3.448276   6.666667
```

Exercise D - (3 min)

The probability mass function of a Binomial(n, p) random variable is given by

$$\mathbb{P}(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Use vectorized mathematical operations and the `choose()` function to calculate the pmf of a Binomial $(5, 0.3)$ random variable *in one fell swoop*.

Solution

```
n <- 5
p <- 0.3
x <- 0:n
pmf <- choose(n, x) * p^x * (1 - p)^(n - x)
pmf
```

```
[1] 0.16807 0.36015 0.30870 0.13230 0.02835 0.00243
```

```
# Check that our calculations agree with dbinom()
all.equal(dbinom(x, n, p), pmf)
```

```
[1] TRUE
```

Exercise E - (5 min)

1. Replace all of the `999`s in this vector with `NA`s

```
x <- c(5, 10, 3, 7, 999, 2, 999, 17, 0)
```

2. In a deck of [Italian playing cards], the face cards are *fante* (Knave), *cavallo* (Knight), and *re* (King). In the game [Scopa], *fante* is worth 8, *cavallo* 9, and *re* 10. Convert `cards` to the appropriate numeric values.

```
cards <- c('re', 'cavallo', 're', 'fante', 'cavallo', 'fante', 're')
```

3. This code throws an error. Coerce `y` to make it work.

```
y <- c('1', '2', '3')
sum(y)
```

4. What happens if you run `as.logical(-2:2)`? Can you figure out the coercion rule for numeric to logical?

Solution

```
# Part 1
x[x == 999] <- NA
x
```

```
[1]  5 10  3  7 NA  2 NA 17  0
```

```
# Part 2
# The slickest solution uses a lookup table:
lookup <- c('fante' = 8, 'cavallo' = 9, 're' = 10)
cards <- c('re', 'cavallo', 're', 'fante', 'cavallo', 'fante', 're')
lookup[cards]
```

```
   re cavallo       re fante cavallo    fante       re
   10        9      10       8       9       8      10
```

```
# Part 3
y <- c('1', '2', '3')
sum(as.numeric(y))
```

```
[1] 6
```

```
# Part 4
# Every element becomes TRUE except for 0, which becomes FALSE
as.logical(-2:2)
```

```
[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

Exercise F - (10 min)

1. Call `z_score(w)` where `w <- c(1, 2, NA)`. What happens? See `?mean()`.
2. Test out this function. What happens? Now try adding `return(z)` at the bottom of the function body. Explain your results.

```
bad_z_score <- function(x) {
  z <- (x - mean(x)) / sd(x)
}
```

3. Write a function to compute skewness using `sum()`, `length()`, `mean()` and `sd()`.

$$\text{Skewness} \equiv \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^3.$$

4. Use `sum()`, `length()` and `is.na()` to write a function called `my_var()` that drops `NA`s and then computes the sample variance.
5. Write a function called `summary_stats()` that returns a named vector with two elements: the sample mean and standard deviation.

Solution

```
# Part 1

# Part 2
# The final statement in this function *stores* the result so it doesn't return
# anything. Either drop the assignment or add return()

# Part 3
skewness <- function(x) {
  mean(((x - mean(x)) / sd(x))^3)
}

# Part 4
```

```
my_var <- function(x) {
  x <- x[!is.na(x)]
  n <- length(x)
  sum((x - mean(x))^2) / (n - 1)
}

# Part 5
summary_stats <- function(x) {
  c('mean' = mean(x), 'sd' = sd(x))
}
```

Exercise G - (8 min)

1. What happens if you run the following code? Why?

```
x <- c(TRUE, TRUE)
if(x) {
  print('hello world!')
}
```

2. What happens if you run this code? Try to fix it.

```
if(3 > 5) {
 print('3 is greater than 5')
}
else {
 print('3 is not greater than 5')
}
```

3. Write a function called `mycov()` that calculates the sample covariance between `x` and `y`. Use an early return to print an error message when `x` and `y` have different lengths.
4. Consult `?trunc()`. Then use `trunc()` to write a function called `myround()` that rounds `x` to the nearest integer.

Solution

```
# Part 1
# This code fails: the condition inside of if() must evaluate to
# a *single* logical value, but this is a vector.

# Part 2
# The problem is the line break before else. This runs:
if(3 > 5) {
 print('3 is greater than 5')
} else {
 print('3 is not greater than 5')
}
```

```
[1] "3 is not greater than 5"
```

```
# Part 3
mycov <- function(x, y) {
  if(!identical(length(x), length(y))) {
    return('Error: x and y must have the same length')
  }
  (x - mean(x)) * (y - mean(y))
}

# Part 4
myround <- function(x) {
  integer_part <- trunc(x)
  decimal_part <- x - integer_part
  if(decimal_part <= 0.5) {
    out <- integer_part
  } else {
    out <- integer_part + 1
  }
  out
}
```

Exercise H - (8 min)

1. The Fibonacci Sequence is defined by $F_1 = 1, F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n > 2$. Write a function that uses a `for()` loop to compute first `n` Fibonacci numbers.
2. Come up with a way to generate the same output as `f()` *without* using a loop or `if() ... else`.

```
f <- \(x) {
  for(j in 1:length(x)) {
    if(x[j] > 0) {
      x[j] <- x[j]^3 + x[j]
    } else {
      x[j] <- x[j]^2 - x[j]
    }
  }
  x
}
```

Solution

```
# Part 1
fib <- function(n) {
  out <- vector(length = n)
  out[2] <- out[1] <- 1
  for(i in 3:n) {
    out[i] <- out[i - 1] + out[i - 2]
  }
  out
}
fib(12)
```

```
 [1]   1   1   2   3   5   8  13  21  34  55  89 144
```

```
# Part 2
g <- function(x) {
  (x > 0) * (x^3 + x) + (x <= 0) * (x^2 - x)
}
f(-2:2)
```

```
[1]  6  2  0  2 10
```

```
g(-2:2)
```

```
[1]  6  2  0  2 10
```

Exercise I - (8 min)

1. Create a 5×5 matrix called `A`, each of whose rows contains the elements `1:5`. Hint: see `?rep`.
2. Display all elements of `A` *except* row 3 and column 2.
3. Form a matrix `B` by stacking the (4×4) identity matrix on top of itself.
4. Display the seventh row of `B`.
5. Write a function that uses a `for()` loop to construct the $(n \times n)$ exchange matrix J_n.

Solution

```
# Part 1
A <- matrix(rep(1:5, times = 5), 5, 5, TRUE)

# Part 2
A[-3, -2]
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    4    5
[2,]    1    3    4    5
[3,]    1    3    4    5
[4,]    1    3    4    5
```

```
# Part 3
B <- rbind(diag(nrow = 4), diag(nrow = 4))

# Part 4
B[7, ]
```

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

```
# Part 5
get_exchange <- function(n) {
  out <- matrix(0, n, n)
  for(i in 1:n) {
    out[i, n + 1 - i] <- 1
```

```
  }
  out
}
```

Exercise J - (8 min)

1. Write a function to constructs the $(n \times n)$ exchange matrix J_n *without* using a loop.
2. Compute the *element-wise* product of J_3 with itself, and the *square* of J_3, i.e. the ordinary matrix product $J_3 J_3$.
3. Let X be a Bernoulli(0.2) and Y be a Binomial$(2, 0.5)$ RV. Construct a matrix `p_XY` that represents the *joint* pmf of X and Y, under the assumption that X and Y are independent. Name the rows and columns.
4. Consult `?rowSums()` and `?colSums()`. Then extract the marginal pmfs of X and Y from the matrix `p_XY`.

Solution

```
# Part 1
get_exchange <- function(n) {
  out <- matrix(0, n, n)
  anti_diagonal <- cbind(1:n, n:1)
  out[anti_diagonal] <- 1
  out
}

# An even slicker solution to part 1, suggested by a student:
get_exchange2 <- function(n) {
  diag(1, n)[n:1, ]
}

# Part 2
J3 <- get_exchange(3)
J3 * J3
```

```
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]    1    0    0
```

```
J3 %*% J3
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

```
# Part 3
p_XY <- c(0.2, 0.8) %o% c(0.25, 0.5, 0.25)
rownames(p_XY) <- c('x=0', 'x=1')
colnames(p_XY) <- c('y=0', 'y=1', 'y=2')
```

```
# Part 4
rowSums(p_XY)
```

```
x=0 x=1
0.2 0.8
```

```
colSums(p_XY)
```

```
 y=0  y=1  y=2
0.25 0.50 0.25
```

Exercise K - (7 min)

1. I used `students$name == 'Xerxes'` above. Why didn't I instead use `identical(students$name, 'Xerxes')`?
2. Use the following code chunk to construct the `employees` data frame. Then display it.

```
employees <- data.frame(
  name = c("Alice", "Bob", "Cathy", "David", "Eva",
           "Frank", "Grace", "Hank", "Ivy", "Jack"),
  age = c(25, 31, 28, 40, 35, 23, 30, 45, 33, 29),
  department = c("HR", "IT", "Finance", "IT", "HR",
                 "Finance", "IT", "HR", "Finance", "IT"),
  salary = c(50000, 60000, 55000, 70000, 53000,
             51000, 62000, 71000, 57000, 59000)
)
```

3. Display the `age` column of `employees`.
4. Display the sixth row of `employees`.
5. Display the employee record for `Eva`.
6. Display employee records for everyone in the `IT` department.
7. Repeat the preceding, restricted to people with a salary of at least 60,000.

Solution

```
# Part 1
students <- data.frame('name' = c('Xerxes', 'Xanthippe', 'Xanadu'),
                       'age' = c(19, 23, 21),
                       'grade' = c(65, 70, 68),
                       'favorite_color' = c('blue', 'red', 'orange'))

# identical() returns a *scalar* but we need a vector
students[identical(students$name, 'Xerxes'), ]
```

```
[1] name           age            grade          favorite_color
<0 rows> (or 0-length row.names)
```

```
# Part 2
employees
```

```
    name age department salary
1  Alice  25         HR  50000
2    Bob  31         IT  60000
3  Cathy  28    Finance  55000
4  David  40         IT  70000
5    Eva  35         HR  53000
6  Frank  23    Finance  51000
7  Grace  30         IT  62000
8   Hank  45         HR  71000
9    Ivy  33    Finance  57000
10  Jack  29         IT  59000
```

```
# Part 3
employees$age
```

```
 [1] 25 31 28 40 35 23 30 45 33 29
```

```
# Part 4
employees[6, ]
```

```
   name age department salary
6 Frank  23    Finance  51000
```

```
# Part 5
employees[employees$name == 'Eva', ]
```

```
  name age department salary
5  Eva  35         HR  53000
```

```
# Part 6
is_IT <- employees$department == 'IT'
employees[is_IT, ]
```

```
    name age department salary
2    Bob  31         IT  60000
4  David  40         IT  70000
7  Grace  30         IT  62000
10  Jack  29         IT  59000
```

```
# Part 7
high_salary <- employees$salary >= 60000
employees[is_IT & high_salary, ]
```

```
   name age department salary
2   Bob  31         IT  60000
4 David  40         IT  70000
7 Grace  30         IT  62000
```