Lab 2. Some Logic

Review

Sampling a Vector

```r
# Vector to select from
a <- c(1, 2, 3, 4)
# Without replacement & Equal Probabilities
sample(a, size = 2, replace = FALSE)
```

```
[1] 4 3
```

```r
# With Replacement \& Equal Probabilities
sample(a, size = 12, replace = TRUE)
```

```
 [1] 1 3 2 4 3 3 3 2 1 4 4 1
```

```r
# With Replacement \& UNEQUAL Probabilities
sample(a, size = 12, replace = TRUE, prob = c(0.05, 0.05, 0.1, 0.7))
```

```
 [1] 4 4 4 4 4 4 4 4 4 4 1 4
```

Help Menu

Type ? before the function's/command's name ### Logical Vectors

```r
# Don't name a variable TRUE or FALSE
A <- c(TRUE, TRUE, FALSE, FALSE)  #must be in caps
B <- c(T, F, T, F)  #must be in caps
```

Logical Operators - Vectorized

```r
# Using the Vector A and B from above
!A  #NOT, NEGATION - NOT TRUE is FALSE, NOT FALSE is TRUE
```

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

```r
A && B  #Scalar-AND -> First elements Both TRUE for TRUE, otherwise FALSE
```

```
[1] TRUE
```

```r
A & B  #Vector-AND - Both TRUE for TRUE, otherwise FALSE
```

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

```r
A || B  #Scalar-OR -> First elements At least one TRUE for TRUE, otherwise FALSE
```

```
[1] TRUE
```

```r
A | B  #Vector-OR - At least one TRUE for TRUE, otherwise FALSE
```

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

Comparisons

```r
# Element by Element comparisons
x <- c(1, 2, 3, 4, 5)
y <- c(5, 4, 3, 2, 1)
x == y  # equality

[1] FALSE FALSE  TRUE FALSE FALSE

x != y  # inequality

[1]  TRUE  TRUE FALSE  TRUE  TRUE

x <= y

[1]  TRUE  TRUE  TRUE FALSE FALSE

x < y

[1]  TRUE  TRUE FALSE FALSE FALSE

x >= y

[1] FALSE FALSE  TRUE  TRUE  TRUE

x > y

[1] FALSE FALSE FALSE  TRUE  TRUE
```

Any & All

```r
x <- c(TRUE, FALSE)
y <- c(TRUE, TRUE)
z <- c(FALSE, FALSE)

# Are any TRUE?
any(x)

[1] TRUE

any(y)

[1] TRUE

# Are all TRUE?
all(x)

[1] FALSE

all(y)

[1] TRUE
```

Filtering with Logic

```
x <- c(1, 2, 3, 4)
bool <- c(FALSE, TRUE, FALSE, TRUE)
x[bool]

[1] 2 4

x[x > 3]

[1] 4

x[x >= 3]

[1] 3 4
```

ifelse

```
x <- c(1, 2, 3, 4)
y <- c(2, 2, 2, 2)
# if the corresponding values are equal, produce 0, otherwise produce 1
ifelse(x == y, 0, 1)

[1] 1 0 1 1
```

if…else…

```
x <- c(1, 2, 3, 4)
# if a condition is true, complete these steps. Else Complete alternate steps if first value is 1, output the second,
# else output the third
y <- sample(x, size = 3, replace = FALSE)
y

[1] 1 4 2

if (y[1] == 1) {
    y[2]
} else if (y[1] == 2) {
    y[3]
} else {
    y[1]
}

[1] 4
```

Examples

1. Using the *sample*( ) function, you will use *R* to simulate selecting a numbered ball from a bucket. The contents of the bucket are: One ball marked 1. Two balls marked 2. Three balls marked 3. Four balls marked 4.

a. Create two vectors: **unique.values.on.balls** and **proportions.of.each.value**. The values in the second vector should correspond the the values in the first.

b. Use the sample function to simulate selecting a single ball from the bucket. Run this in the console, just to see the output.

c. Create one vector: **expected.number.of.each.value**. Its elements will represent the number of times you would expect to see a 1,2,3,or 4, if you repeated the selection 1,000 times. I believe there should be about 100 1s, 200 2s, 300 3s and 400 4s. You should think about that.

d. Create one vector: **simulated.observations**. Use the *sample*( ) function to repeat the process of selecting one ball one thousand times.

e. Create one vector: **observed.number.of.each.value**. Use the *table*( ) function to count the number of 1s,2s,3s, and 4s that appeared in **simulated.observations**.

f. Create one vector: **absolute.difference**. Compute the absolute value of the difference between **observed.number.of.each.value** and **expected.number.of.each.value**.

g. Create one vector: **absolute.difference.greater.30**. Use a comparison that checks each element in **absolute.difference** to see if it is larger than 30.

h. Create one vector: **any.greater.than.30**. It should be a logical vector that indicates TRUE if any values in **absolute.difference.greater.30** were greater than 30.

i. Report whether or not there where any occasions where **observed.number.of.each.value** had a difference of more than 30 from **expected.number.of.each.value** Use inline code.

j. Run this code several times, your report should update with new values each time.

Problems

Use the *sample*( ) function to simulate rolling a fair six-side die 50 times.

2. Create two vectors: **unique.number.of.pips** and **proportions.of.number.of.pips**. The values in the second vector should correspond the the values in the first.

3. Use the *sample*( ) function to simulate rolling a single die one time.

4. Create one vector: **expected.number.of.each.side**. Its elements will represent the number of times you would expect to see each number of pips, if you rolled a die 50 times. (Don't round you value.)

5. Create one vector: **simulated.rolls**. Use the *sample*( ) function to repeat the process of rolling a die 50 times.

6. Create one vector: **observed.number.of.each.side**. Use the *table*( ) function to count the number of 1s,2s,3s, and 4s that appeared in **simulated.rolls**.

7. Create one vector: **absolute.difference**. Compute the absolute value of the difference between **observed.number.of.each.side** and **expected.number.of.each.side**.

8. Create one vector: **absolute.difference.greater.5**. Use a comparison that checks each element in **absolute.difference** to see if it is larger than 5.

9. Create one vector: **any.greater.than.5**. It should be a logical vector that indicates TRUE if any values in **absolute.difference.greater.5** were greater than 5.

10. Report whether or not there where any occasions where **observed.number.of.each.side** had a difference of more than 5 from **expected.number.of.each.side** Use inline code.

11. Run this code several times, your report should update with new values each time.