R Programming Reference Sheet: Week 5–6

Conditional Statements: if, else, else if

R supports standard conditional statements to control the flow of execution.

Syntax:

```
if (condition) {
  # code if condition is TRUE
} else if (another_condition) {
  # code if another_condition is TRUE
} else {
  # code if all above conditions are FALSE
}
```

Example:

```
x <- 5
if (x > 0) {
  print("Positive")
} else if (x == 0) {
  print("Zero")
} else {
  print("Negative")
}
```

Loops: for, while, repeat, break, next

1. for Loop: Iterates over a sequence

```
for (i in 1:5) {
  print(i)
}
```

2. while Loop: Repeats while condition is TRUE

```
x <- 1
while (x <= 5) {
```

```
  print(x)
  x <- x + 1
}
```

3. repeat Loop: Repeats indefinitely until break

```
x <- 1
repeat {
  print(x)
  x <- x + 1
  if (x > 5) break
}
```

4. break: Exits the loop early

```
for (i in 1:10) {
  if (i == 6) break
  print(i)
}
```

5. next: Skips the current iteration

```
for (i in 1:5) {
  if (i == 3) next
  print(i)
}
```

Vectorized Operations and apply Family

Vectorized Operations:
R supports operations on entire vectors without loops.

Example:

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
z <- x + y  # z is c(5, 7, 9)
```

Apply Family:
Efficient alternatives to loops for applying functions to data structures.

1. apply(): Used on matrices/arrays

```
mat <- matrix(1:9, nrow=3)
apply(mat, 1, sum)  # Row-wise sum
apply(mat, 2, mean) # Column-wise mean
```

2. lapply(): Applies a function to each element of a list and returns a list

```
lst <- list(a=1:3, b=4:6)
lapply(lst, sum)
```

3. sapply(): Same as lapply but tries to simplify the result

```
sapply(lst, sum)
```

4. tapply(): Applies a function to subsets of a vector grouped by another vector

```
ages <- c(21, 22, 23, 21, 22)
group <- c("A", "A", "B", "B", "A")
tapply(ages, group, mean)
```

5. mapply(): Multivariate version of sapply

```
mapply(sum, 1:3, 4:6)  # sum(1,4), sum(2,5), sum(3,6)
```