Lab 1 Class Notes

1. Installation

2. Basic syntax

2.1 Installing Packages (Once in your workstation)

- install.packages("packagename")

2.2 Loading Packages (for every session)

- library(packagename)

Basic Data Types

- Basic data types in R can be divided into the following types:
- numeric - (10.5, 55, 787)
- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- complex - (9 + 3i, where "i" is the imaginary part)
- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- logical (a.k.a. boolean) - (TRUE or FALSE)

Type Conversion

- as.numeric()
- as.integer()
- as.complex()

Example

```
x <- 1L # integer
y <- 2 # numeric
# convert from integer to numeric:
a <- as.numeric(x)
# convert from numeric to integer:
b <- as.integer(y)
# print values of x and y
x
y
# print the class name of a and b
class(a)
class(b)
```

For instance we can assign the value 3 to the variable x using the <- assignment operator.

Assignment

```
 x <- 3
x = value
x <<- value
```

Rightward assignment

```
value -> x
value ->> x
```

The original assignment operator in R was <- and has continued to be the preferred among R users. The = assignment operator was added in 2001 primarily because it is the accepted assignment operator in many other languages and beginners to R coming from other languages were so prone to use it. The operators <<- is normally only used in functions which we will not get into the details.

Case Sensitivity

Lastly, note that R is a case sensitive programming language. Meaning all variables, functions, and objects must be called by their exact spelling:

```
x <- 1
y <- 3
z <- 4
x * y * z
```

```
## [1] 12
```

```
x * Y * z
```

```
## Error in eval(expr, envir, enclos): object 'Y' not found
```

There are many types of R-objects. The frequently used ones are −

Vectors, Lists, Matrices, Arrays, Factors, Data Frames

These will be discussed later.

Basic Arithmetic

At its most basic function R can be used as a calculator. When applying basic arithmetic, thePEMDAS order of operations applies: parentheses first followed by exponentiation, multiplication and division, and final addition and subtraction.

```
8 + 9 / 5 ^ 2
## [1] 8.36
8 + 9 / (5 ^ 2)
## [1] 8.36
8 + (9 / 5) ^ 2
## [1] 11.24
(8 + 9) / 5 ^ 2
## [1] 0.68
```

By default R will display seven digits but this can be changed using options () as previously outlined.

```
1 / 7
## [1] 0.1428571
options (digits = 3)
1 / 7
## [1] 0.143
pi
## [1] 3.141592654
options(digits = 22)
pi
## [1] 3.141592653589793115998+
```

We can also perform integer divide (%/%) and modulo (%%) functions. The integer dividefunction will give the integer part of a fraction while the modulo will provide the remainder.

```
42 / 4 # regular division
## [1] 10.5
42 %/% 4 # integer division
## [1] 10
42 %% 4 # modulo (remainder)
## [1] 2
```

Control Statements in R Programming

```
if(expression)
{
statements
   ....
   ....
}
```

Example:

```
x <- 100
if(x > 10)
{
        print(paste(x, "is greater than 10"))
}
```

Example:

```
x <- 5
# Check value is less than or greater than 10
if(x > 10)
{
        print(paste(x, "is greater than 10"))
} else
{
        print(paste(x, "is less than 10"))
}
```

Functions:

You can write your own functions in order to make repetitive operations using a single command. Let's start by defining your function "my_function" and the input parameter(s) that the user will feed to the function. Afterwards you will define the operation that you desire to program in the body of the function within curly braces ({}). Finally, you need to assign the result (or output) of your function in the return statement.

Example 1:

```
fahrenheit_to_celsius<- function(temp_F)
{
        temp_C<- (temp_F - 32) * 5 / 9
        return(temp_C)
}
```

Let's try running our function. Calling our own function is no different from calling any other function:

```
# Freezing point of water
fahrenheit_to_celsius(32)
```

Example 2:

```
func<- function(x){
if(x > 0)
{
        return("Positive")
}
else if(x < 0)
        {
        return("Negative")
        }
        else
        {
        return("Zero")
        }
}
func(1)
func(0)
func(-1)
```

For loop in R

It is an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

For loop in R Syntax:

```
for (var in vector)
{
        statement(s)
}
```

Example 1

```
for (i in 1: 4)
{
        print(i ^ 2)
}
```

Example 2

```
for (i in c(-8, 9, 11, 45))
{
        print(i)
}
```

Break statements

```
# R Program to demonstrate the use of
# break in for loop

for (i in c(3, 6, 23, 19, 0, 21))
{
        if (i == 0)
        {
                break
        }
        print(i)
}
print("Outside Loop")
```

Next Statement

It discontinues a particular iteration and jumps to the next iteration. So when next is encountered, that iteration is discarded and the condition is checked again. If true, the next iteration is executed. Hence, the next statement is used to skip a particular iteration in the loop.

Example

```
for (i in c(3, 6, 23, 19, 0, 21))
{
        if (i == 0)
        {
                next
        }
        print(i)
}
print('Outside Loop')
```

R – while loops

```
i <- 1
while (i< 6) {
        print(i)
        i<- i + 1
}
```

Syntax to take input from the user

```
var <- readline("Enter the number: ")


var <- as.numeric(readline("Enter the number: "))
```