GKTCS INNOVATIONS

R Functions

R Functions

- ❑ Functions are used to logically break our code into simpler parts which become easy to maintain and understand.

- ❑ It’s pretty straightforward to create your own function in R programming.

Syntax

```
func_name <function(argument){
statement
}
```

R Functions

- The reserved word **function** is used to declare a function in R.

- The statements within the curly braces form the **body of the function.**

- These braces are optional if the body contains only a single expression.

- Finally, this function object is given a name by assigning it to a variable, **func_name.**

R Functions

Example

```
pow <- function(x, y) {
# function to print x raised to the
power y  result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

- ❑ Here, we created a function called **pow().**
- ❑ It takes two arguments, finds the first argument raised to the power of second argument and prints the result in appropriate format.
- ❑ We have used a built-in function **paste()** which is used to concatenate strings.

R Functions

GKTCS INNOVATIONS

How to call a function?

We can call the above function as follows.

```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
>pow(2, 8)
"2 raised to the power 8 is 256”
```

- ❑ **Here, the arguments used in the function declaration (x and y) are called formal arguments and those used while calling the function are called actual arguments.**
- ❑ **In the above function calls, the argument matching of formal argument to the actual arguments takes place in positional order.**
- ❑ **This means that, in the call pow(8,2), the formal arguments x and y are assigned 8 and 2 respectively.**

R Functions

Named Arguments

We can also call the function using named arguments.
When calling a function in this way, the order of the actual arguments doesn’t matter.
For example, all of the function calls given below are equivalent.

```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
>pow(x = 8, y = 2)
[1] "8 raised to the power 2 is 64"
>pow(y = 2, x = 8)
[1] "8 raised to the power 2 is 64"
```

R Functions

we can use named and unnamed arguments in a single call.
In such case, all the named arguments are matched first and then the remaining unnamed arguments are matched in a positional order.

```
>pow(x=8, 2)
[1] "8 raised to the power 2 is 64"
>pow(2, x=8)
[1] "8 raised to the power 2 is 64”
```

In all the examples above, x gets the value 8 and y gets the value 2.

R Functions

GKTCS INNOVATIONS

Default Values for Arguments

We can assign default values to arguments in a function in R. This is done by providing an appropriate value to the formal argument in the function declaration.

```
pow <- function(x, y = 2) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

R Functions

The use of default value to an argument makes it optional when calling the function.

```
>pow(3)
[1] "3 raised to the power 2 is 9"
>pow(3,1)
[1] "3 raised to the power 1 is 3"
```

R Functions

GKTCS INNOVATIONS

Basic components of a function

- The body(), the code inside the function.
- The formals(), the "formal" argument list, which controls how you can call the function.
- The `environment()`` which determines how variables referred to inside the function are found.
- args() to list arguments.

```
f <- function(x) x
f
formals(f)
environment(f)
```

R Functions

More on environments

Variables defined inside functions exist in a different environment than the global environment. However, if a variable is not defined inside a function, it will look one level above.

```
x <- 2
g <-
  function() {
  y <- 1
  c(x, y)
}
g()
#[1] 2 1
```

R Functions

Same rule applies for nested functions

A first useful function.

```
first <- function(x, y) {
  z <- x + y
  return(z)
}
add <- function(a, b) {
  return(a + b)
}
vector <- c(3, 4, 5, 6)
sapply(vector, add, 1)
```

R Functions

GKTCS INNOVATIONS

What does this function return?

```
x <- 5
f <- function() {  y <- 10
   c(x = x, y = y)
}
f()
```

R Functions

What does this function return?

```
x <- 5
g <- function() {  x <- 20
   y <- 10
   c(x = x, y = y)
}
g()
```

R Functions

What does this function return?

```
x <- 5
h <- function() {  y <- 10
   i <- function() {  z <- 20
      c(x = x, y = y, z = z)
   }
   i()
}
h()
```

R Functions

Functions with pre defined values

```
temp <- function(a = 1, b = 2) {
        return(a + b)
}
```

Functions usually return the last value it computed

```
f <- function(x) {
    if (x < 10) {
  0
  } else {
  10
  }
}
f(5)
f(15)
```

R Functions

Commonly Used R functions

append()	Add elements to a vector
c()	Compactly Values into a Vector or List
identical()	Test if 2 objects are exactly equal.
length()	Returns length of R object.
is()	List objects in current environment.
range(x)	Returns minimum and maximum of vector.
rep(x,n)	Repeat the number x, n times
rev(x)	Reversed version of its argument.

R Functions

GKTCS INNOVATIONS

Seq(x,y,n)	Generate regular sequences from x to y, spaced by n
unique(x)	Remove duplicate entries from vector
summary(x)	Returns Object Summaries
str()	Compactly Display the Structure of an Arbitrary R Object
glimpse(x)	Compactly Display the Structure of an Arbitrary R Object(dplyr package)
class(x)	Return
mode(x)	Get or set the type or storage mode of an object.
tolower()	Convert string to lower case letters
toupper()	Convert string to upper case letters
grep()	Used for regular expressions

Run Script

Running R Script

The source() function instructs R reads the next file and execute its contents.
source("myScript.R")

Optional parameter echo=TRUE will echo the script lines before they are executed
source("myScript.R", echo=TRUE)

Run Script

```
>?source
>source("ex1.R")
[1] "Welcome to R Programming"
[1] "hello" "hi"      "good"
[1] "x is less than 1"
[1] "x is between 8 and 15"
[1] "Second"
[1] "Fourth"
>
```

Run Script

```
>source("ex1.R",echo=TRUE)
Echo the Script lines before execution
>print("Welcome to R Programming")
[1] "Welcome to R Programming"
>?apply
>s1=c("hello","hi","good")
>print(s1)
[1] "hello" "hi" "good"
```

Run Script

Running a Batch Script

R CMD BATCH command will help to run code in batch mode.

```
$ R CMD BATCH myscript.R outputfile
```

In case if you want the output sent to stdout or if you need to pass command-line arguments to the script then Rscript command can be used.

```
$ Rscript myScript.R arg1 arg2
```

Run Script

```
Surendras-MacBook-Pro:rprog SurendraMac$ R CMD BATCH hello.R  outhello
Surendras-MacBook-Pro:rprog SurendraMac$ ls

R_Notebook1.Rmd         ex1.R       hello1.R    outhello
R_Notebook1.nb.html     hello.R     ifex.R      readintegerex.R


Surendras-MacBook-Pro:rprog SurendraMac$ Rscript hello.R
[1] "Welcome to R Programming“
Surendras-MacBook-Pro:rprog SurendraMac$ Rscript hello1.R
[1] "Test R”
```