R_Functions

Mr. Sachin B.

Simple Function

```
#f() for "Hello"

f<- function(x)
{
  x
}
f("Hello")
```

```
## [1] "Hello"
```

```
# addnum() to add 2 numbers and return result
addnum <- function(x,y)
          {
            x+y
          }

addnum(2,3)
```

```
## [1] 5
```

```
# selectabove() Function to return a numbers above threshold

selectabove <- function(n,key)
{
  use <- n>key
  n[use]
}

n <- 1:20
selectabove(n,15)
```

```
## [1] 16 17 18 19 20
```

```


addnum
```

To see source code of function

```
## function(x,y)
##           {
##             x+y
##           }
```

```
args(addnum)
```

To know argument of function

```
## function (x, y)
## NULL
```

Argument Order

```
# argorder() will assign value to the argument in a order

argorder <- function(first,second, third)
          {
            cat("First=",first,"Second=",second,"Third=",third,"\n")
          }

argorder(1,2,3)
```

```
## First= 1 Second= 2 Third= 3
```

```
argorder(2,3,1)
```

```
## First= 2 Second= 3 Third= 1
```

```
argorder(3,1,2)
```

```
## First= 3 Second= 1 Third= 2
```

Value to Specific Argument

```
# specificarg() will assign value to the argument in a order

specificarg <- function(first,second, third)
          {
            cat("First=",first,"Second=",second,"Third=",third,"\n")
          }

specificarg(1,2,3)
```

```
## First= 1 Second= 2 Third= 3
```

```
specificarg(second = 2, third = 3,first=1)
```

```
## First= 1 Second= 2 Third= 3
```

```
specificarg(third=3,first=1,second=2)
```

```
## First= 1 Second= 2 Third= 3
```

Default Value

```
# defaultval() will assign default value to the argument if not specified

defaultval <- function(first=1000,second=2000, third=3000)
          {
            cat("First=",first,"Second=",second,"Third=",third,"\n")
          }

# only single value provided
defaultval(1)
```

```
## First= 1 Second= 2000 Third= 3000
```

```
# Two values provided
defaultval(1,2)
```

```
## First= 1 Second= 2 Third= 3000
```

```
# All values provided
defaultval(1,2,3)
```

```
## First= 1 Second= 2 Third= 3
```

```
# Single value provided by specifying argument name
defaultval(third = 333)
```

```
## First= 1000 Second= 2000 Third= 333
```

```
# Two values provided by specifying and not specifying argument name
defaultval(2, third = 33)
```

```
## First= 2 Second= 2000 Third= 33
```

```
defaultval(third = 33, 2)
```

```
## First= 2 Second= 2000 Third= 33
```

```
# All values provided by specifying and not specifying argument name
defaultval(third=3,first=11,222)
```

```
## First= 11 Second= 222 Third= 3
```

Default Value (Practice)

```
# defaultseq() will use default sequence for range

defaultseq <- function(key,n=1:10)
{
  use <- n>key
  n[use]
}

n <- 1:20
defaultseq(15,n)
```

```
## [1] 16 17 18 19 20
```

```
defaultseq(8)
```

```
## [1]  9 10
```

Default Value and Argument order (Practice)

```
mydata <- rnorm(100)

# ?sd

# Arguments of Inbuilt function: sd()

sd(mydata)
```

```
## [1] 1.042374
```

```
sd(x=mydata)
```

```
## [1] 1.042374
```

```
sd(x=mydata,na.rm = FALSE)
```

```
## [1] 1.042374
```

```r
sd(na.rm = FALSE, x=mydata)
```

```
## [1] 1.042374
```

```r
sd(na.rm=FALSE,mydata)
```

```
## [1] 1.042374
```

Lazy Evaluation of Function

```r
# Arguments are evaluated as needed

f <- function(x,y)
          {
            x*5
          }
f(2)
```

```
## [1] 10
```

```r

# What if default value is not available and argument value not given

f <- function(x,y)
          {
            print(x)
            print(y)
          }
f(2)

# [1] 2
# Error in print(y) : argument "y" is missing, with no default
```

Error Case

Function as Argument to another Function

```r
addnum <- function(x,y)
          {
            x+y
          }

sample <- function(func)
          {
            func(2,4)
          }

sample(addnum)
```

```
## [1] 6
```

anonymous Function

```
# This does not call the anonymous function.
# (Note that "3" is not a valid function.)

function(x) 3()
```

```
## function(x) 3()
```

```
# With appropriate parenthesis, the function is called:

# (function(x) 3)()

# OR

# (
#   function(x)
#    3
# )

# ()

(function(x) 3)()
```

```
## [1] 3
```

```
# So this anonymous function syntax

# (function(x) x + 3)(10)

#  OR

# (
#   function(x)
#     x + 3
# )

# (10)

(function(x) x + 3)(10)
```

```
## [1] 13
```

```
# behaves exactly the same as
f <- function(x) x + 3
f(10)
```

```
## [1] 13
```

ellipses or ...

```
msg <- function(...)
{
  paste('Start',...,'Stop')
}

msg("Hi","bye")
```

```
## [1] "Start Hi bye Stop"
```

```
#Example
paste
```

```
## function (..., sep = " ", collapse = NULL, recycle0 = FALSE)
## {
##     if (isTRUE(recycle0))
##         .Internal(paste(list(...), sep, collapse, recycle0))
##     else .Internal(paste(list(...), sep, collapse))
## }
## <bytecode: 0x0000000012af0fa8>
## <environment: namespace:base>
```

```
mean
```

```
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x000000001528e000>
## <environment: namespace:base>
```

```
story <- function(...)
{
  a<- list(...)
  place<- a['place']]
  adjective<- a$adjective
  noun<- a[["noun"]]

  paste("News from", place,"today where", adjective, "student took to",noun,"for Fun")

}

story(place="Mumbai",adjective="Sad", noun="Mall")
```

unpacking arguments from ellipses

```
## [1] "News from Mumbai today where Sad student took to Mall for Fun"
```

Laxical Scoping

The basic principle of lexical scoping is that names defined inside a function mask names defined outside a function. This is illustrated in the following example.

```
x <- 10
y <- 20
g02 <- function() {
  x <- 1
  y <- 2
  c(x, y)
}
g02()
```

```
## [1] 1 2
```

```
x <- 2
g03 <- function() {
  y <- 1
  c(x, y)
}
g03()
```

If a name isn't defined inside a function, R looks one level up.

```
## [1] 2 1
```

```
y
```

And this doesn't change the previous value of y

```
## [1] 20
```

Creating new binary operator

```
"%P%" <- function(l,r)
{
  l*r+1
}

4%P%5
```

```
## [1] 21
```