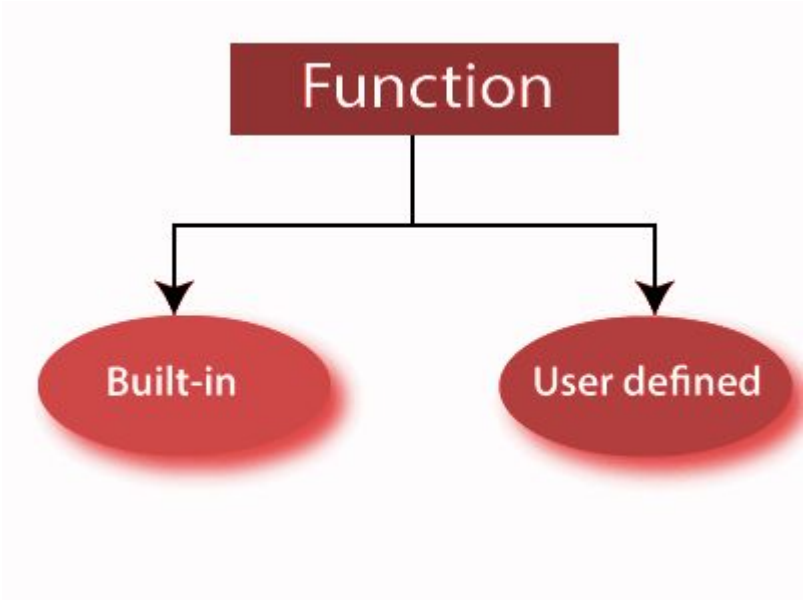
Module 1

Functions in R

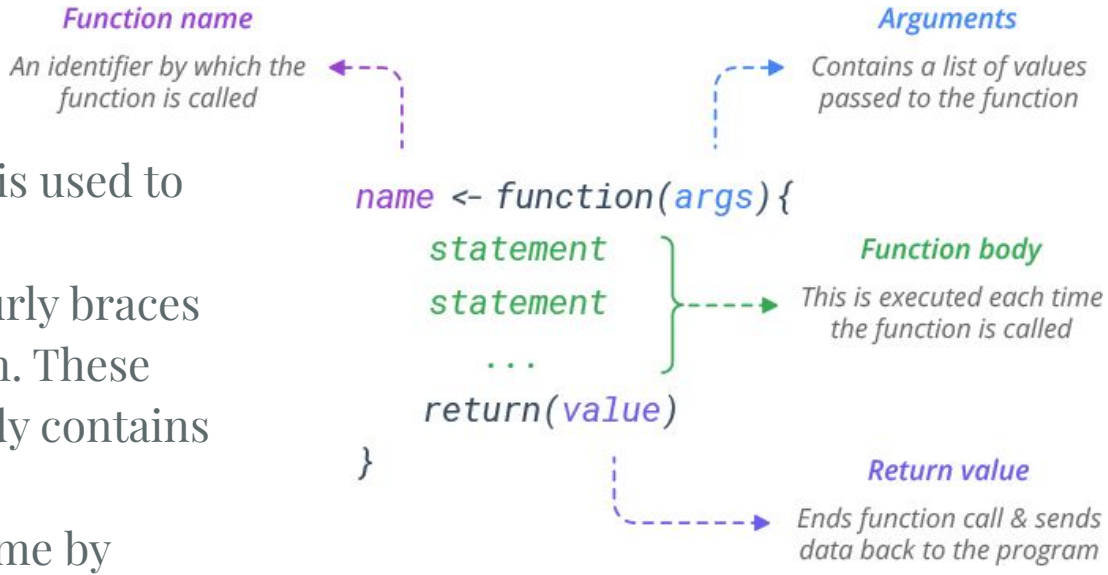
Function

A collection of statements structured together for carrying out a definite task



User defined function

- The reserved word **function** is used to declare 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.
- Function object is given a name by assigning it to a variable,*(here it is **name**)*



An example

```r
# Creating a function with name fun_print
fun_print <- function(){
  print("Welcome to R functions")
}
# Calling the function fun-print()
fun_print()
```

```
R 4.3.1 · ~/
> # Creating a function with name fun_print
> fun_print <- function(){
+   print("Welcome to R functions")
+ }
> # Calling the function fun-print()
> fun_print()
[1] "Welcome to R functions"
>
```

With arguments

```r
### Function with arguments
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))
}

pow(2,5)
```

```
[1] "2 raised to the power 5 is 32"
```

With named arguments

If you pass arguments to a function by name, you can put those arguments in any order.

```r
### Function with arguments
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))
}

pow(2,3)

# using argument names
pow(x=2, y=3)

# changing the order
pow(y=3, x=2)
```

```
> pow(2,3)
[1] "2 raised to the power 3 is 8"
>
> # using argument names
> pow(x=2, y=3)
[1] "2 raised to the power 3 is 8"
>
> # changing the order
> pow(y=3, x=2)
[1] "2 raised to the power 3 is 8"
```

Default Values for Arguments

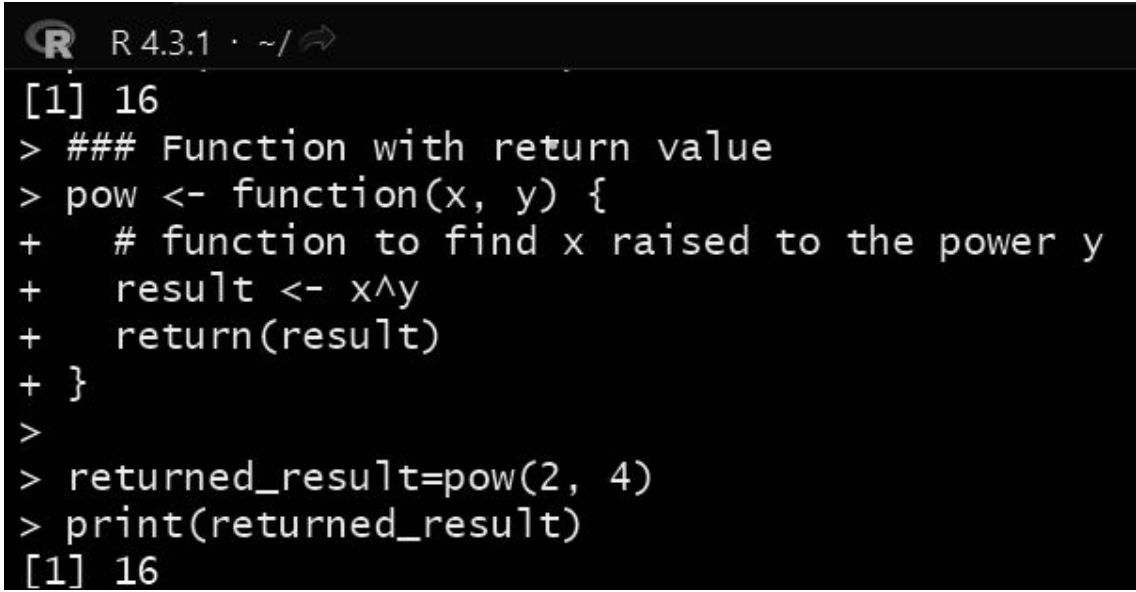
```r
### Function with default arguments
pow <- function(x, y=3) {
  # function to print x raised to the power y
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
# function will use default y value
pow(2)
# specifying a different y value
pow(2, 4)
```

```
> pow(2)
[1] "2 raised to the power 3 is 8"
> # specifying a different y value
> pow(2, 4)
[1] "2 raised to the power 4 is 16"
```

With return value

```r
### Function with return value
pow <- function(x, y) {
  # function to find x raised to the power y
  result <- x^y
  return(result)
}

returned_result=pow(2, 4)
print(returned_result)
```



Return multiple values

```r
#return multiple values
calculator <- function(x, y) {
  add <- x + y
  sub <- x - y
  mul <- x * y
  div <- x / y
  return(c(addition = add, subtraction = sub,
    multiplication = mul, division = div))
}
result=calculator(10,5)
print(result)
```

```
> result=calculator(10,5)
> print(result)
      addition    subtraction multiplication       division
            15              5             50              2
```

Lazy Evaluation

```
###### Lazy Evaluation ###########
myfunc <- function(x, y) {
  if(!x){
    return(y)
  }
  else{
    return(x)
  }
}
# y is not evaluated so not including it causes no harm
myfunc(6)

# y is evaluated so not including it raises error
myfunc(0)
```

- R functions perform lazy evaluation that dramatically extends the expressive power of functions.
- It is the technique of not evaluating arguments unless and until they are needed in the function.

```
> # y is not evaluated so not including it causes no harm
> myfunc(6)
[1] 6
>
> # y is evaluated so not including it raises error
> myfunc(0)
Error in myfunc(0) : argument "y" is missing, with no default
```

Variable Length Argument

- specify an ellipsis (...) in the arguments when defining a function.
- accepts variable number of arguments

```
####Variable length arguments

d <- function(...){
  x <- list(...) # THIS WILL BE A LIST STORING EVERYTHING:
  sum(...)       # Example of inbuilt function
}

d(1,2,3,4,5)
d(30)
d(4,7,6,3)
```

```
> d(1,2,3,4,5)
[1] 15
> d(30)
[1] 30
> d(4,7,6,3)
[1] 20
```

Scope of R functions

- An environment is the collection of all the variables and objects.
- The top-level of the environment is the global environment.
- When we create a function, it creates a local environment that exists in the global environment.
- The function operates inside the local environment.
- Variables and objects created inside a function, exist only inside the function's local environment.
- If an object does not exist inside the function's local environment, then the interpreter tries to find it in the global environment.

Recursive functions

- call a function that calls itself

```
###### Recursive Functions
factorial <- function(x){
  if(x==0)
    return(1)
  else
    return(x*factorial(x-1))
}

factorial(5)
```

```
> factorial(5)
[1] 120
```