

Question about a function

[Subscribe for email updates.](#)



Sort replies by: [Oldest first](#) [Newest first](#) [Most popular](#)

[assignments](#) × [function](#) × [assignment2](#) × [+ Add Tag](#)



[Mick van Hulst](#) [Signature Track](#) · [5 days ago](#)

I just finished assignment 2 but there is still something that I do not understand.

To give an example when I declare a function I do something like this:

```
get <- function(x)
```

However in this assignment I actually saw something else:

```
get <- function() x
```

Why is x not within the function brackets? Could anyone explain the difference to me?

Thanks in advance! :)

4 · [flag](#)

[Ivan Medvid](#) [Signature Track](#) · [5 days ago](#)

Ok, I'll try. To be honest I googled and played a lot with this stuff till I got it a bit.

So, functions have 3 types of parameters(variables):

1st) arguments of the function.

When you type `f<- function(x) {some transformation on x, or actually its not necessary, then return some result}`. Everything is easy here, when you run e.g. `f(5)` then your function just "eats" 5 and gives you results based on it. x is an argument in this example.

2nd) local parameters.

```
f<-function(x) {  
  t<-x^2  
  return(t+x)  
}
```

In this example `t` is local parameter which exists only in the environment of this function. After you run the function it disappears. Why? Because you run function `f` in Global environment and `t` was never defined there. If you want it not to "die" you can put

```
t<-x^2
```

Next time you run e.g `f(5)`, `t` will be defined in Global environment. It is very useful when you have some important local parameters you don't want to lose them.

3rd) free variables.

Here we have your example `f<- function() x`. statement `x` is equivalent to `print(x)`. So when you run your function `f()`, R will try to print `x`. But what is `x`?

Another example. You could wrote function `f<-function(t) t+x`. Then when you write `f(5)`, R will try print `5+x`, but first it(R) has to find this `x`.

So `x` in those examples is a free variable. When R tries to print `x` or `5+x` it starts looking for it. And the rule is: it looks for it in the environment where the function `f` was defined. In our example `f` is defined in Global environment, so R looks for `x` there.

If you had defined `x` previously in global environment (e.g. you run there `x<-3`), then you'll get `5+3` and `f(5)` returns 8. If you hadn't defined `x` before you'll get "object 'x' not found"

Now, a little bit more complicated stuff which was in Assignment 2.

a) Your function `get<- function() x` is defined inside another function (`makeCacheMatrix`), so every time your "get" function is run R will try to find `x` in the environment where function "get" was defined.

b) This specific environment is created only when `makeCacheMatrix` function was run first time.
`matrix<-makeCacheMatrix()`

c) What is `matrix`? `matrix` is what `makeCacheMatrix()` returned, and the last returned list of functions. Now when you have this list in Global environment, you can run "get" in Global environment by typing `matrix$get` in prompt.

d) You can check what is the environment of `matrix$get` by typing `environment(matrix$get)`
 You'll get something like `<environment: 0x09049690>` -- it is the environment which was created by `makeCacheMatrix`.

`ls(environment(matrix$get))` -- will list all objects which exist in this environment

```
[1] "get" "getinv" "minv" "set" "setinv" "x"
```

Please pay attention that these 6 objects are not present in Global environment (on the right top corner in RStudio).

e) When you run `matrix$get()`, you are actually running your code mentioned in your comment. So R has to print `x` and it looks for this `x` in environment created by `makeCacheMatrix` (0x09049690). You'll get

```
[,1]
```

```
[1,] NA
```

empty matrix. Why? Because default value of `x` is empty matrix 1x1.

You can try to get this x by yourself

```
get("x", environment(matrix$get))
[1,]
[1,] NA
```

f) Let's put some not empty matrix into this environment (which we call CACHE) by running `matrix$set(matrix(1:4,2,2))`.

Again let's run get function

```
> matrix$get()
[1,] [2,]
[1,] 1 3
[2,] 2 4
```

The same if we run

```
> get("x", environment(matrix$get))
[1,] [2,]
[1,] 1 3
[2,] 2 4
```

Now this 2x2 matrix exists, but again only in this specific (CACHE) environment.

g) When you call

```
> cacheSolve(matrix)
```

you will in particular run `data <- x$get()` which is equivalent to `data <- matrix$get()`, that's how it gets matrix from cache.

↑ 5 ↓ · flag

[+ Comment](#)

Erica DePasquale · 4 days ago 🗨

`get<- function()` x is the same as:

```
get<- function() {
  x
}
```

You are not passing x as an argument like `get<- function(x)` would.

In short, this is just shortened code to make it cleaner, though I feel it loses some readability personally.

↑ 4 ↓ · flag

[+ Comment](#)**Mick van Hulst**

Signature Track

· 3 days ago

Thank you all for your answers! I now understand what they mean when calling a function like the one mentioned above :)!

0 · flag

[+ Comment](#)**Caroline Wilkinson** · a day ago

I don't remember this being explained in any of the lecture, so thank you.

0 · flag

[+ Comment](#)

New post

To ensure a positive and productive discussion, please read our [forum posting policies](#) before posting.

B	<i>I</i>			Link	<code>	Pic	Math		Edit: Rich ▼	Preview
<div></div>										

☐ Make this post anonymous to other students☒ Subscribe to this thread at the same time

Add post