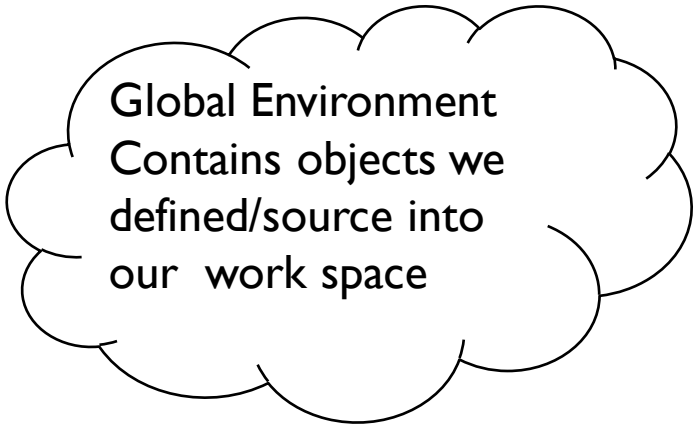
Environments, Scope, and Lazy Evaluation

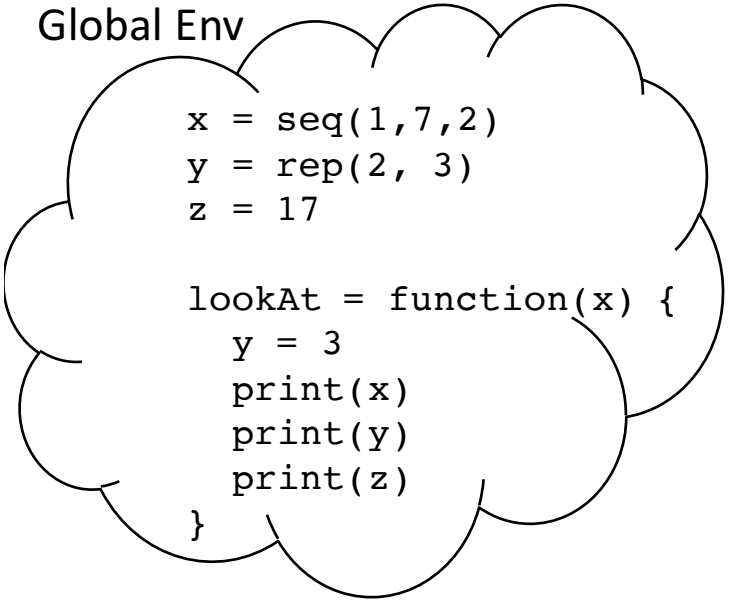
Environments and variable scope

R has a special mechanism for allowing you to use the same name in different places in your code and have it refer to different objects.

For example, you want to be able to create new variables in your functions and not worry if there are variables with the same name already in the workspace.

The solution relies on *environments* and the *variable scoping rules*.





Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

When we call a function, a new workspace, AKA a frame, is created

```
lookAt(x = c(0,100))
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

```
lookAt(x = c(0,100))
```

This call frame has Global Env as its parent environment

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

In the frame, variables are created for the arguments in the function call

```
lookAt(x = c(0,100))
```

```
x = c(0, 100)
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

As the code in the function is evaluated, new objects are added to the call frame

```
lookAt(x = c(0,100))
```

```
x = c(0, 100)
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

```
lookAt =
function(x) {
  y = 3
  print(x)
  print(y)
  print(z)
}
```

```
lookAt(x = c(0,100))
```

```
x = c(0, 100)
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

```
lookAt =
function(x) {
  y = 3
  print(x)
  print(y)
  print(z)
}
```

```
lookAt(x = c(0,100))
```

```
x = c(0, 100)
y is vector 3
```

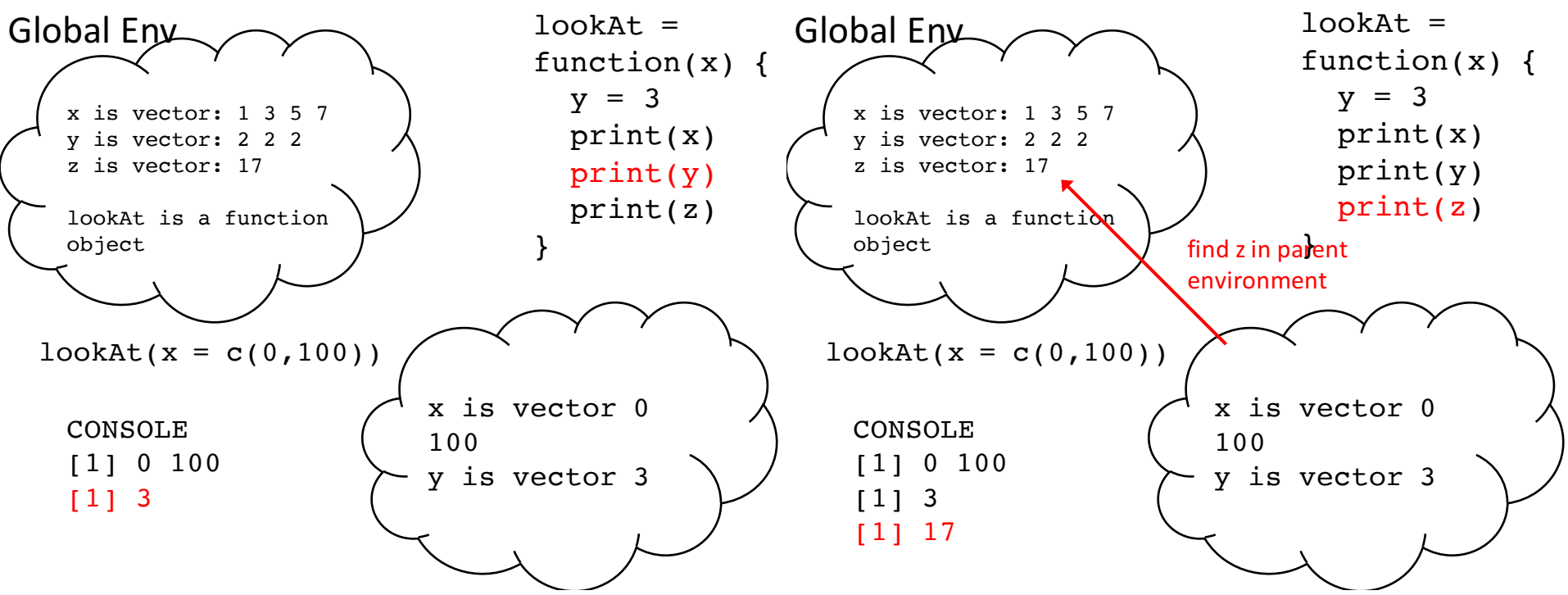
Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt is a function
object
```

```
lookAt =
function(x) {
  y = 3
  print(x)
  print(y)
  print(z)
}
```

```
lookAt(x = c(0,100))

CONSOLE
[1] 0 100
```

```
x is vector 0
100
y is vector 3
```



What is happening is that R is looking for variables with that name in a sequence of environments. An *environment* is just a frame (collection of variables) plus a pointer to the next environment to look in.

In our example, R does not find **z** in the environment defined by lookAt, so it went on to the next one. In this case, this is our main workspace, which is the *Global Environment*.

The "next environment to look in" is called the parent environment.

Finding Objects

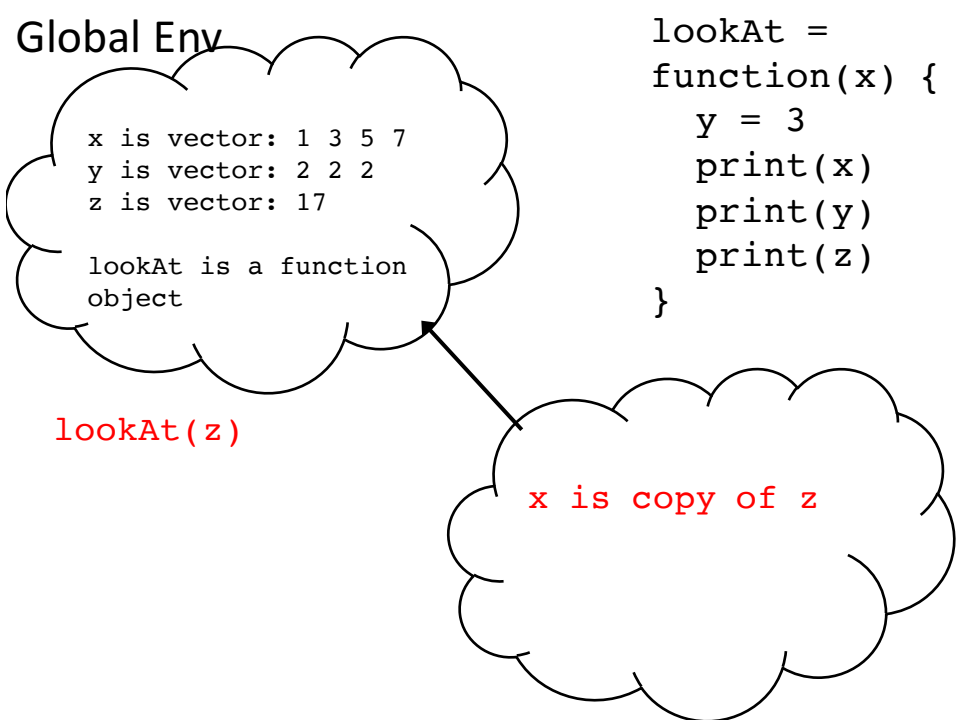
We can ask R to find objects for us:

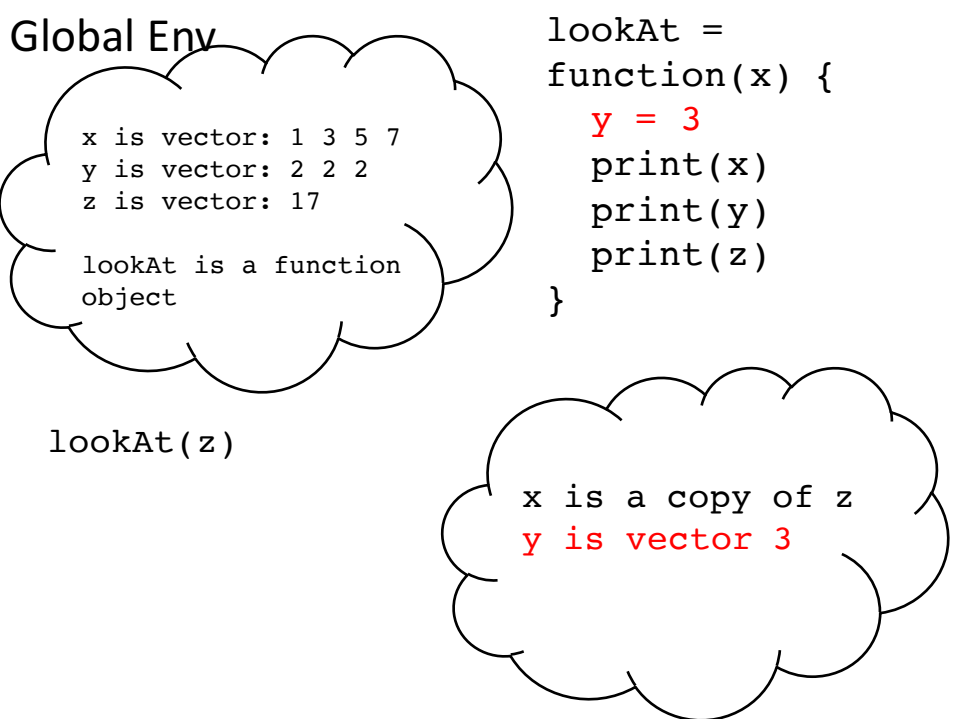
```
> find("pi")
[1] "package:base"

> find("z")
[1] ".GlobalEnv"

> find("ggplot")
[1] "package:ggplot2"
```

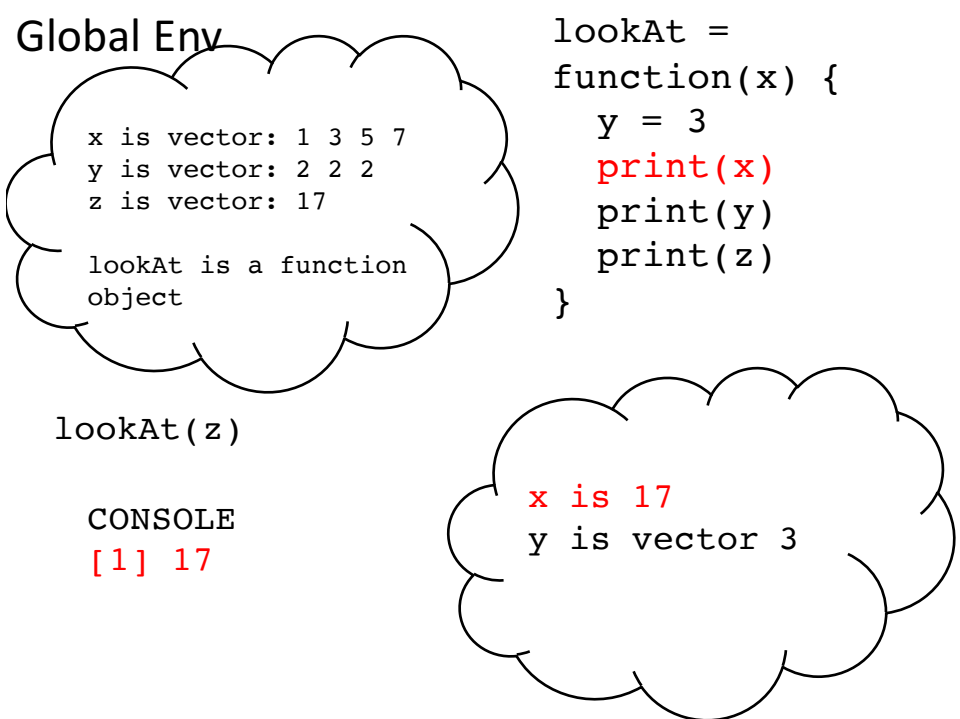
Finding Objects

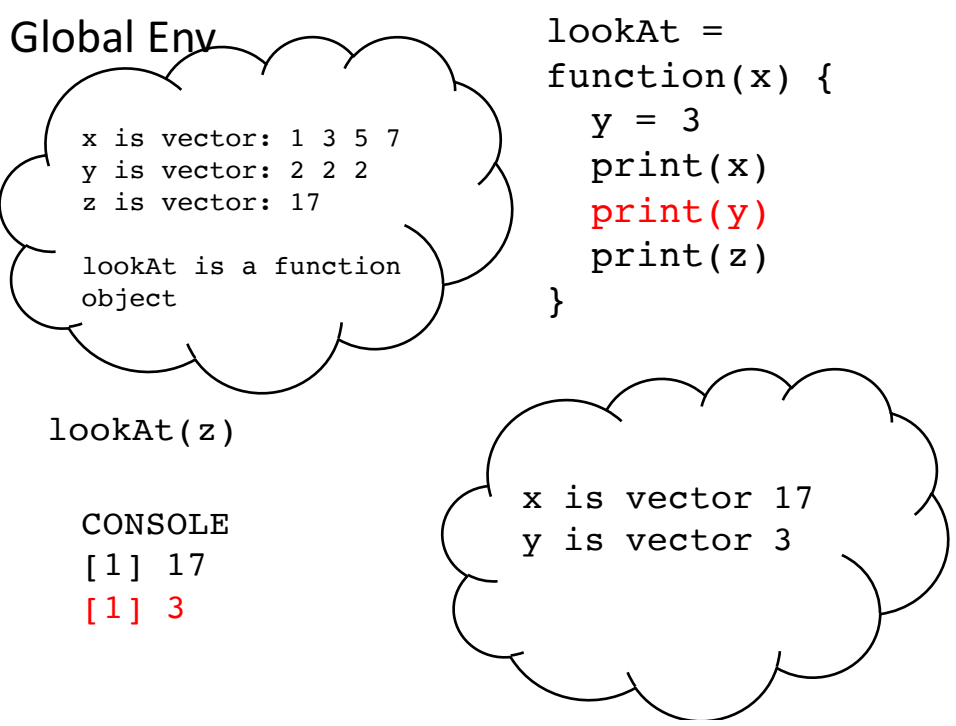
The package codetools helps us find global variables in our functions:

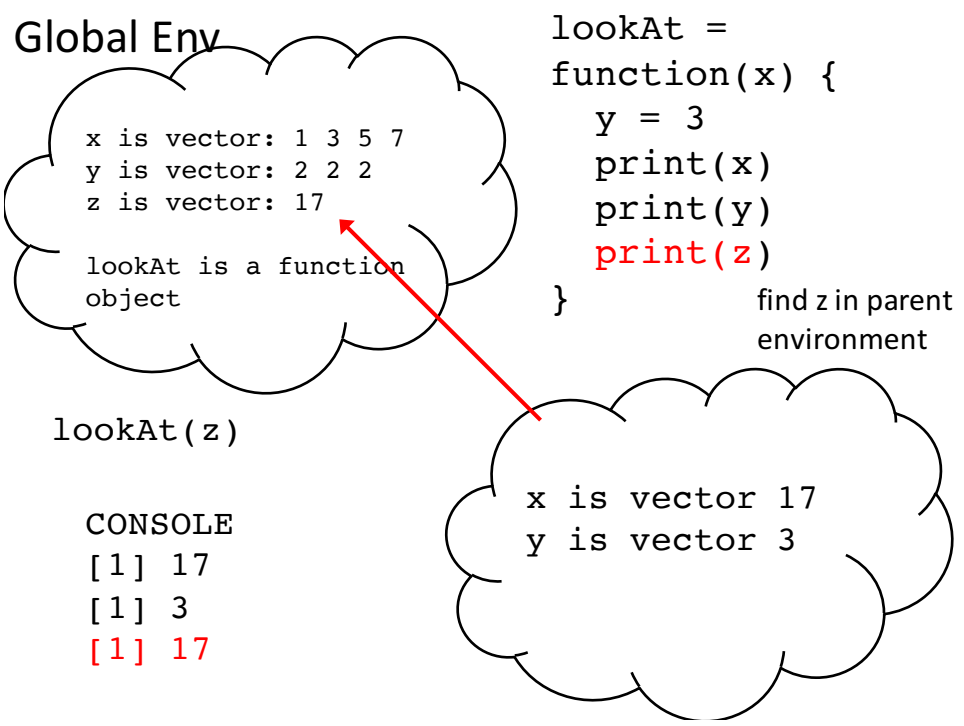
```
> library(codetools)
> findGlobals(lookAt)

[1] "-"     ":"     "{"     "="     ">"
[6] "if"    "print" "z"
```











Lazy Evaluation

The inputs for a function call are not evaluated until they are needed

R sets up a call frame for the function with the input arguments as variables BUT these are only associated with an expression. When the variable is references in the code then this expression is evaluated

This is called *lazy evaluation*.

```
funnyMean =
  function(x, y = mean(x))
{
  set.seed(1234)
  return(y)
}

> set.seed(1234)
> mean(runif(2))
[1] 0.3680014

> set.seed(6789)
> mean(runif(2))
[1] 0.4268003
```

What does

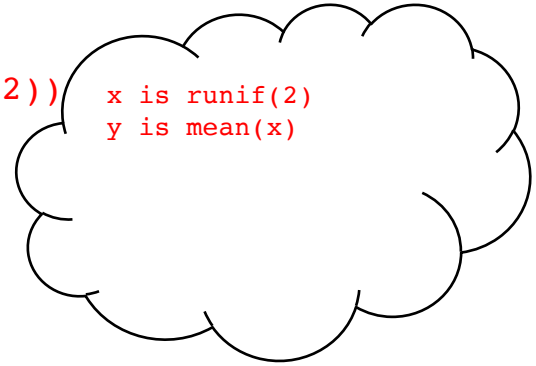
```
> set.seed(6789)
> funnyMean(runif(2))
```

Return?

A. 0.3680014
B. 0.4268003
C. Some other value

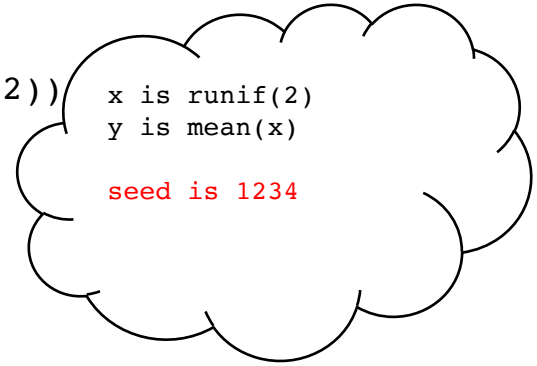
```
funnyMean =
  function(x, y = mean(x))
{
  set.seed(1234)
  return(y)
}
```

```
set.seed(1234)
funnyMean(runif(2))
```



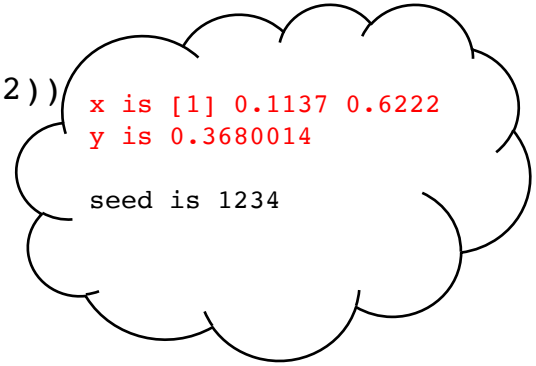

```
funnyMean =
  function(x, y = mean(x))
{
  set.seed(1234)
  return(y)
}
```

```
set.seed(1234)
funnyMean(runif(2))
```




```
funnyMean =
  function(x, y = mean(x))
{
  set.seed(1234)
  return(y)
}
```

```
set.seed(1234)
funnyMean(runif(2))

[1] 0.3680014
```



The parameter x has not been specified in the function call so the frame is set up with x as a variable with no contents

This doesn't cause a problem.

The first time that x is referred to it needs to have a value.

```
funnyMean2 =
  function(x, y = mean(x))
{
  if (any(is.na(x)) {
   warning("Founds NA")
  }
  set.seed(1234)
  return(y)
}
```

```
> set.seed(1234)
> mean(runif(2))
[1] 0.3680014

> set.seed(6789)
> mean(runif(2))
[1] 0.4268003
```

What does

```
> set.seed(6789)
> funnyMean2(runif(2))
```

Return?

A. 0.3680014
B. 0.4268003
C. Some other value

It's not a good idea to set the seed inside a function, unless expressly asked to via an input parameter

```
> set.seed(6789)
> funnyMean2(runif(2))
1] 0.4268003

> funnyMean2(runif(2))
[1] 0.3680014
> funnyMean2(runif(2))
[1] 0.3680014
```

CAN you figure out what's happening?

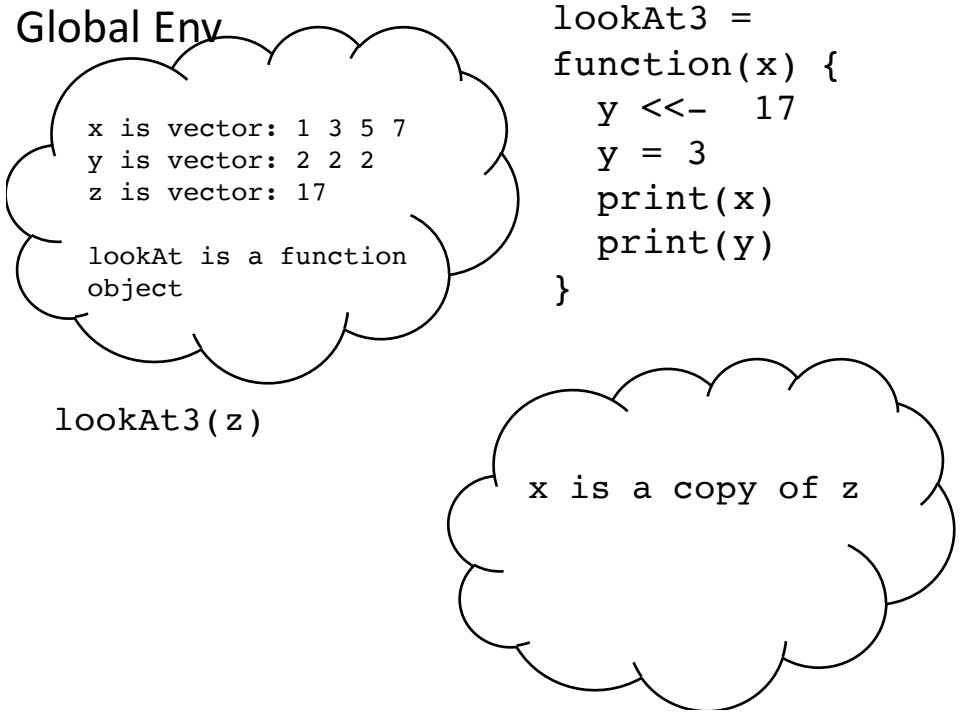
`funnyMean2` keeps resetting the seed to 1234. It's an example of what can go wrong when we set a seed

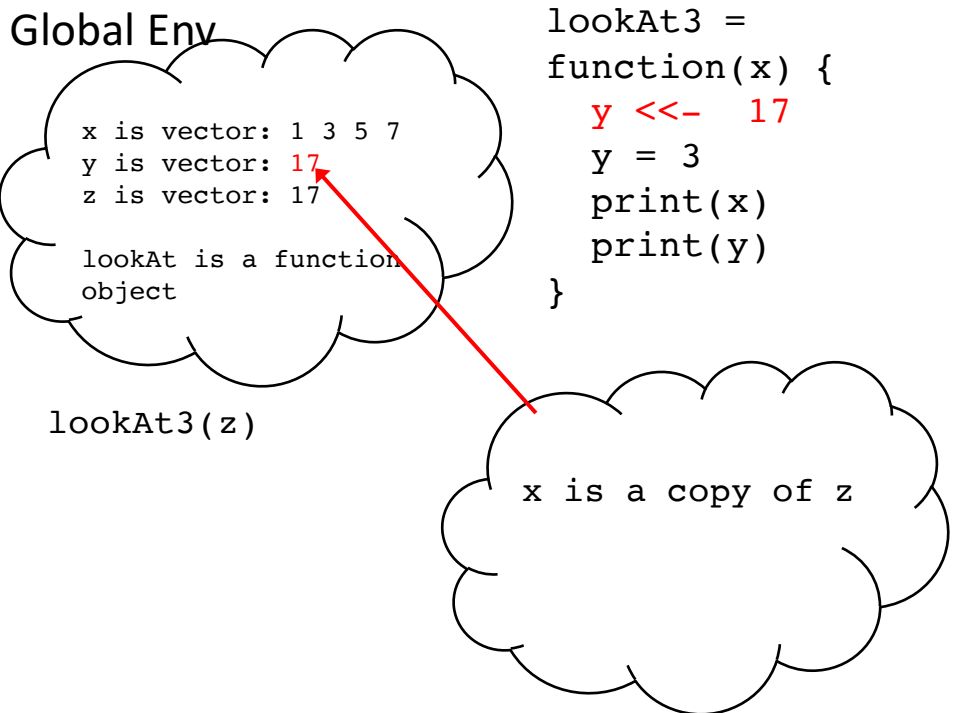
If R reaches the Global Environment and still can't find the variable, it looks in something called the *search path*. This is a list of additional environments, which is used for packages of functions and user attached data. You can see the search path by typing `search()`.

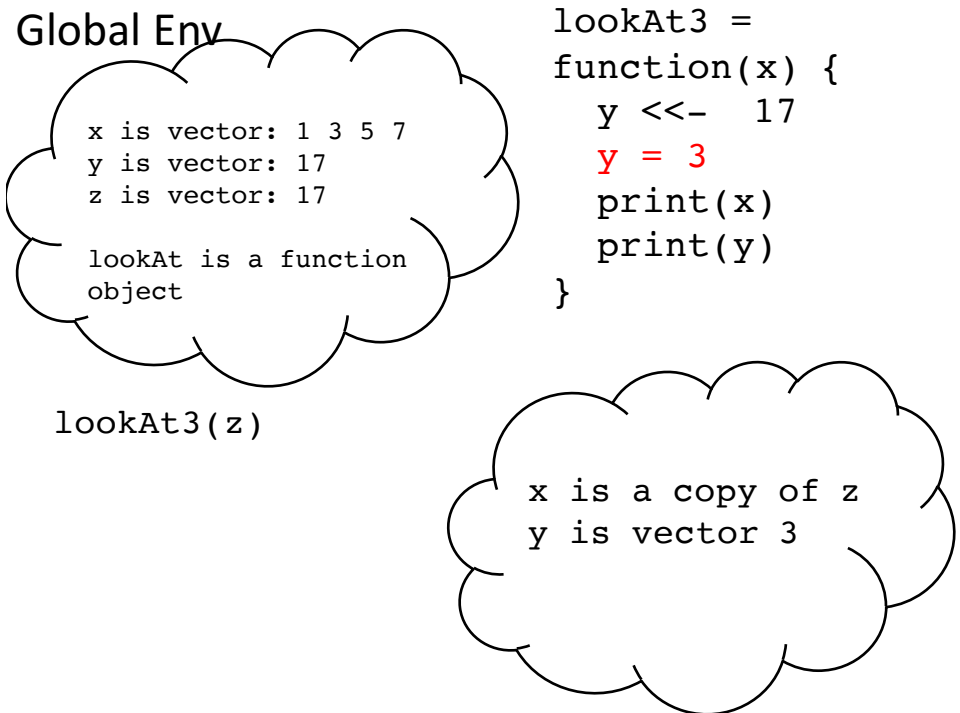
This helps explain why we can write over built-in objects in R. What we're really doing is creating that object in the Global Environment, and then when we refer to it by name, R finds it here before it finds the predefined one.

```
> help(pi)
> pi = 3
> pi
> rm(pi)
> pi
```

Assignment in Parent Environment







Global Env

```
x is vector: 1 3 5 7
y is vector: 17
z is vector: 17

lookAt is a function
object
```

```
lookAt3 =
function(x) {
  y <<-  17
  y = 3
  print(x)
  print(y)
}
```

```
lookAt3(z)

  CONSOLE
  [1] 17
```

```
x is vector 17
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 17
z is vector: 17

lookAt is a function
object
```

```
lookAt3 =
function(x) {
  y <<-  17
  y = 3
  print(x)
  print(y)
}
```

```
lookAt3(z)

  CONSOLE
  [1] 17
  [1] 3
```

```
x is vector 17
y is vector 3
```

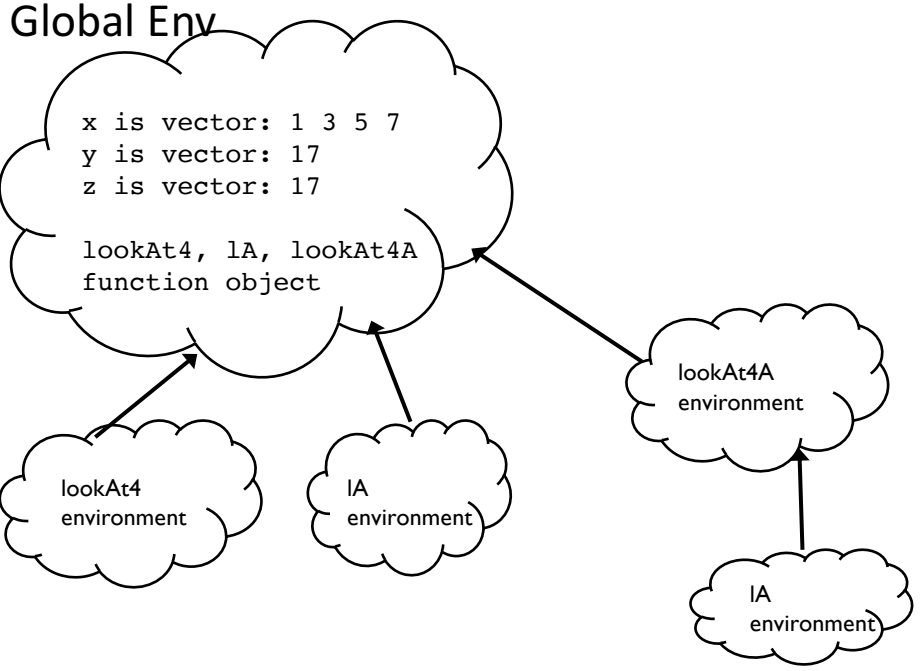
Call Frames for Function Calls within Function

```
lA = function() {
    print(x)
}

lookAt4 = function()
{
  x = 3
  lA()
  print(x)
}
```

When the lA function is defined in the Global Env, it's parent environment is Global Env

```
lookAt4A = function()
{
  lA = function() {
    print(x)
  }

  x = 3
  lA()
  print(x)
}
```

When the lA function is defined in the body of lookAt4E, it's parent environment is lookAt4E's frame




```
lA = function() {
    print(x)
}

lookAt4 = function()
{
  x = 3
  lA()
  print(x)
}
```

```
Global env
x is 1 3 5 7
y is 2 2 2
z is 35
lA, lookat4,
and lookAt4A
```

```
lookAt4A = function()
{
  lA = function() {
    print(x)
  }

  x = 3
  lA()
  print(x)
}
```

```
> lookAt4()
> lookAt4A()
```

```
A.Same
B.Different
```

The function definition for `lA is in the Global environment.`

When `lA` is called in the code in `lookAt4()`, a call frame is set up for `lA()`.

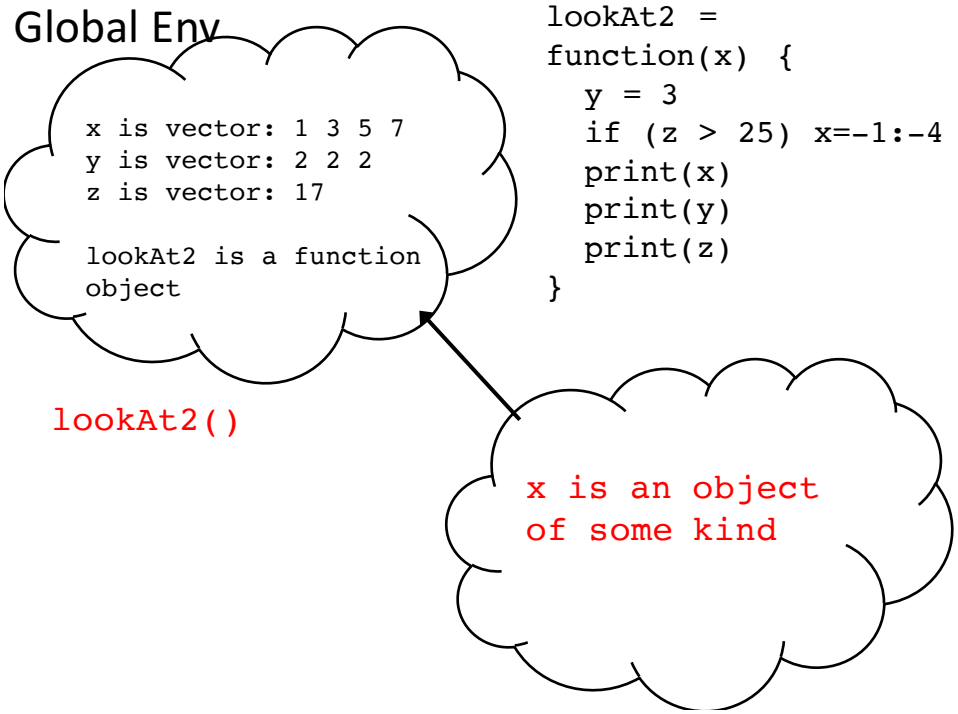
It's parent environment is Global Env

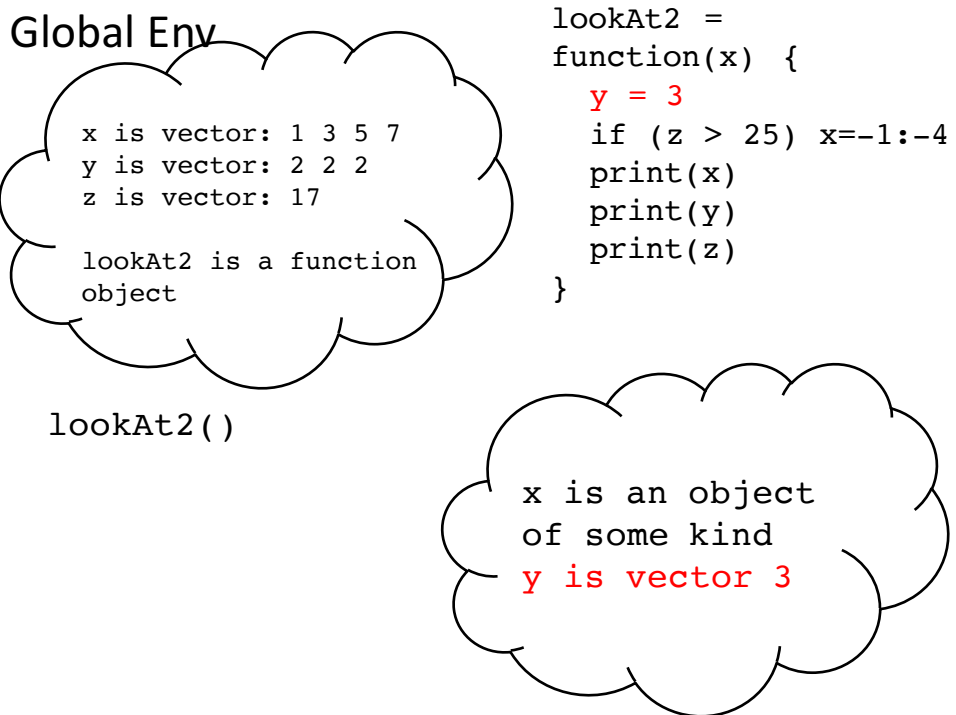
The function definition for `lA is in the Global environment.`

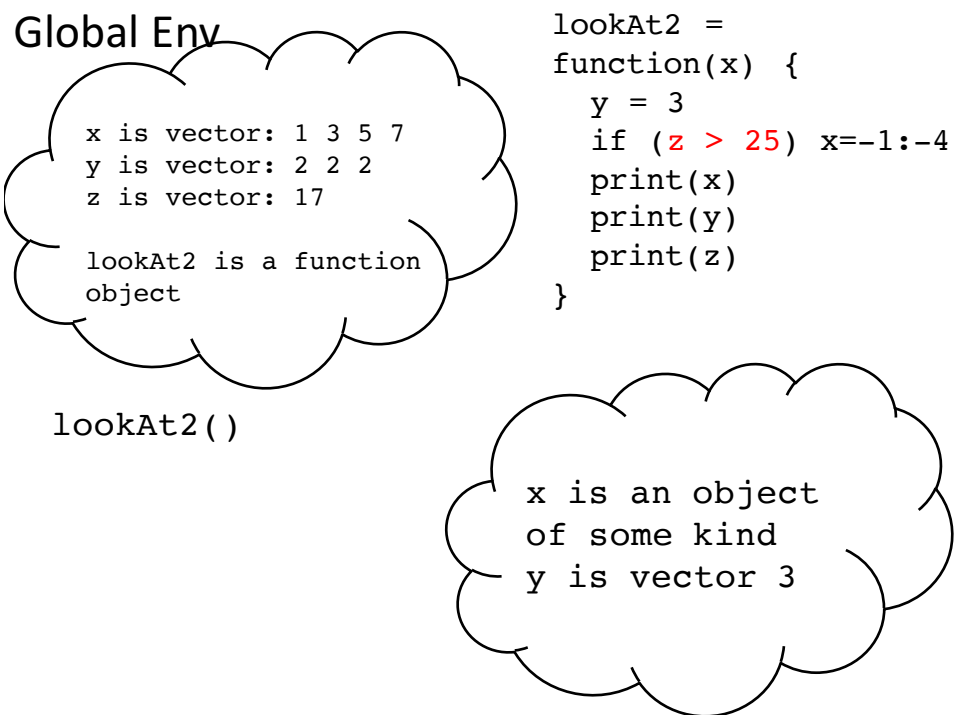
When `lA` is called in the code in `lookAt4A()`, a call frame is set up for `lA()`.

It's parent environment is the lookAt4A frame.

Additional Examples







Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
lookAt2()
```

CONSOLE

Error in print(x) : argument "x" is missing, with no default

```
x is an object
of some kind
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 35

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()
```

```
x is an object
of some kind
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 17

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()
```

```
x is an object
of some kind
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 35

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()
```

```
x is vector -1 -2 -3 -4
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 35

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()
CONSOLE
[1] -1 -2 -3 -4
```

```
x is vector -1 -2 -3 -4
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 35

lookAt2 is a function
object
```

```
lookAt2 =
function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()
CONSOLE
[1] -1 -2 -3 -4
[1] 3
```

```
x is vector -1 -2 -3 -4
y is vector 3
```

Global Env

```
x is vector: 1 3 5 7
y is vector: 2 2 2
z is vector: 35

lookAt2 is a function
object
```

```
lookAt2 = function(x) {
  y = 3
  if (z > 25) x=-1:-4
  print(x)
  print(y)
  print(z)
}
```

```
z = 35
lookAt2()

CONSOLE
[1] -1 -2 -3 -4
[1] 3
[1] 35
```

```
x is vector -1 -2 -3 -4
y is vector 3
```

```
lookAt2 =
 function(x) {
  y = 3
  if (z > 25) {
     x=-1:-4
}
  print(x)
  print(y)
  print(z)
}

Global env
x is 1 3 5 7
y is 2 2 2
z is 17

lookAt2(4)
```

A.
```
[1] -1 -2 -3 -4
[1] 3
[1] 35
```

B.
```
[1] 4
[1] 3
[1] 35
```

C. Error in print(x) : argument "x" is missing, with no default

```
lookAt2 =
 function(x) {
  y = 3
  if (z > 25) {
    x=-1:-4
  }
  print(x)
  print(y)
  print(z)
}
```

Global env
x is 1 3 5 7
y is 2 2 2
z is 35

```
lookAt2(4)
```

A.
```
[1] -1 -2 -3 -4
[1] 3
[1] 35
```

B.
```
[1] 4
[1] 3
[1] 35
```

C. Error in print(x)
: argument "x" is
missing, with no
default