2. R Foundations - Lists and Functions

CT5102 - J. Duggan

Recap - R Data Types

	Homogenous	Heterogenous
1d	Atomic Vector	**List**
2d	Matrix	Data Frame/Tibble
nd	Array	

- Lists are different from atomic vectors because their elements can be of any type, including lists.
- list() creates a list, instead of c()

Creating a list

```
x <- list(1:3, "a", c(T,F,T), c(1.2, 1.3, 1.4))
str(x)
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:3] 1.2 1.3 1.4
```

Subsetting Lists

- Works in the same way as subsetting an atomic vector
- Using [will always return a list
- [[and $ pull out the contents of a list
- If list x is a train carrying objects, then x[[5]] is the object in car 5, x[4:6] is a train of cars 4-6" @RLangTip

Example

```
x <- list(1:3, c(T,F,T))
x[1]
```

```
## [[1]]
## [1] 1 2 3
```

```
str(x[1])
```

```
## List of 1
##  $ : int [1:3] 1 2 3
```

```
x[[1]]
```

```
## [1] 1 2 3
```

```
str(x[[1]])
```

```
##  int [1:3] 1 2 3
```

Naming list elements

```
x <- list(el1=1:3, el2=c(T,F,T))
x
```

```
## $el1
## [1] 1 2 3
##
## $el2
## [1]  TRUE FALSE  TRUE
```

The $ operator

- $ is a shorthand operator, where x$y is equivalent to x[[“y”,exact=FALSE]]
- Often used to access variables in a data frame
- $ does partial matching

```
x
```

```
## $el1
## [1] 1 2 3
##
## $el2
## [1]  TRUE FALSE  TRUE
```

```
x$el1
```

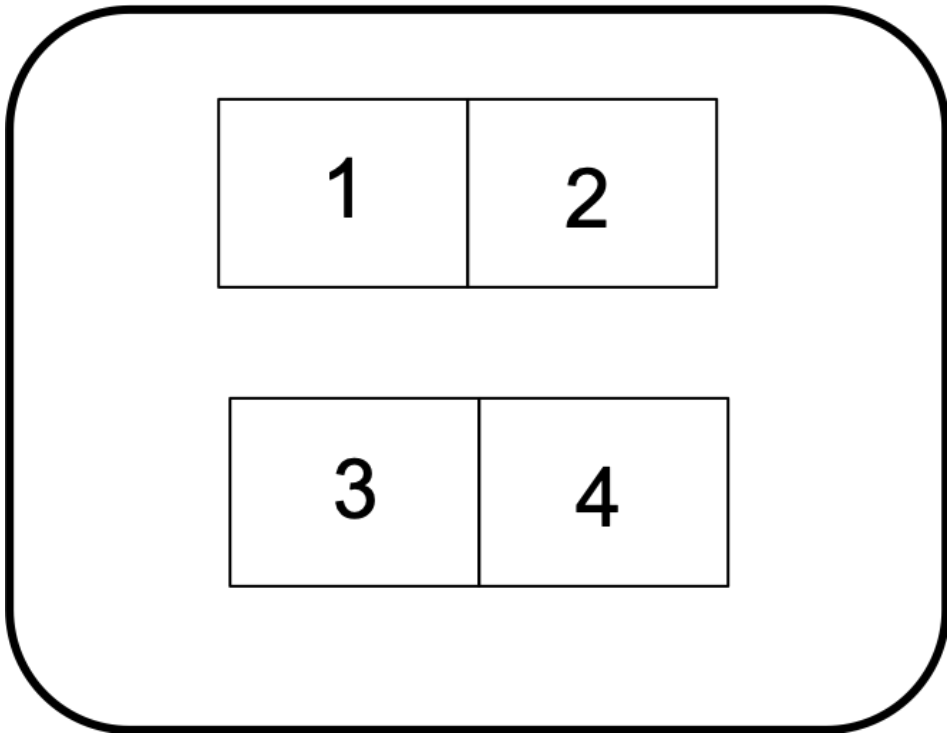
```
## [1] 1 2 3
```

```
x$el2
```

Visualising Lists

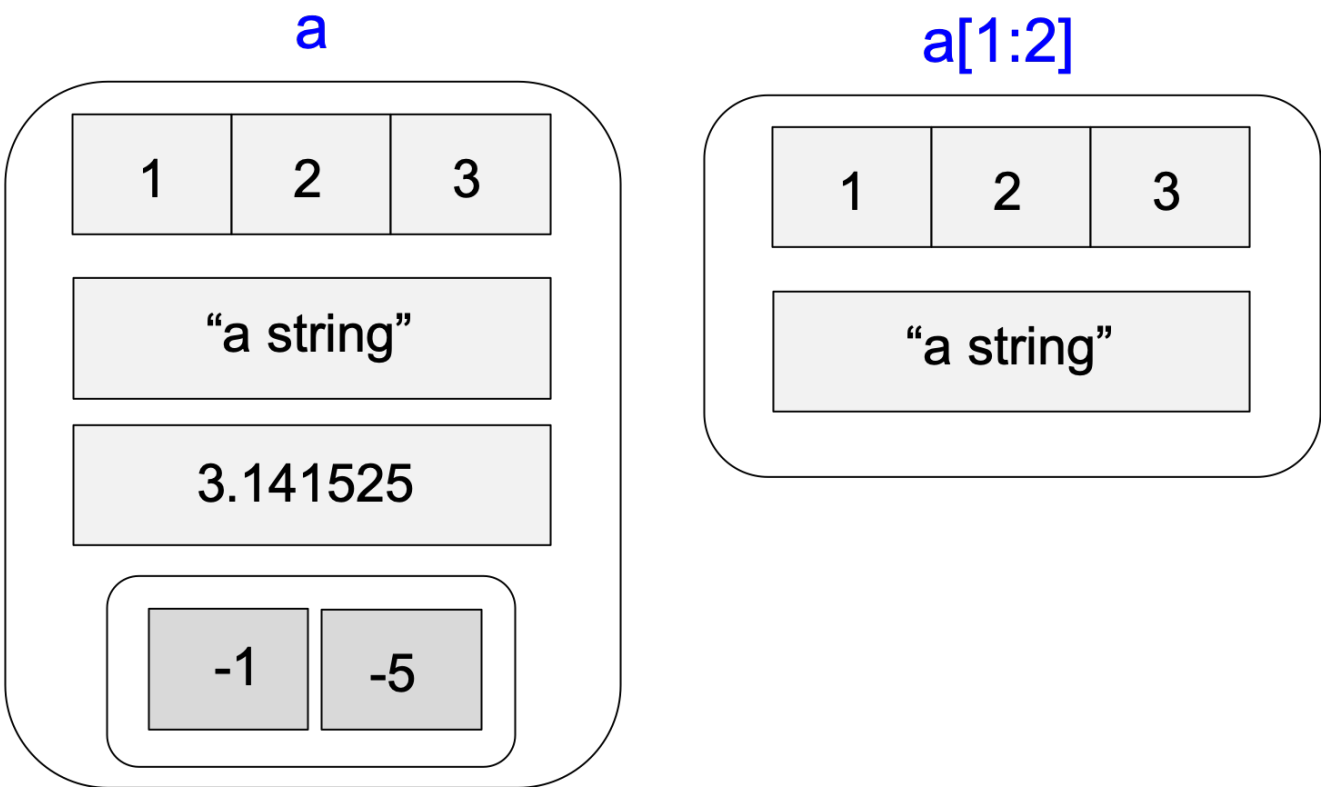
- Lists have rounded corners.
- Atomic vectors have square corners
- Children are drawn inside their parent, and have a slightly darker background

```
y <- list(c(1,2),c(3,4))
```



Further List Example

```
a <- list(el1=1:3,el2="a string",el3=pi,el4=list(-1,-5))
```



Summarising the list structure

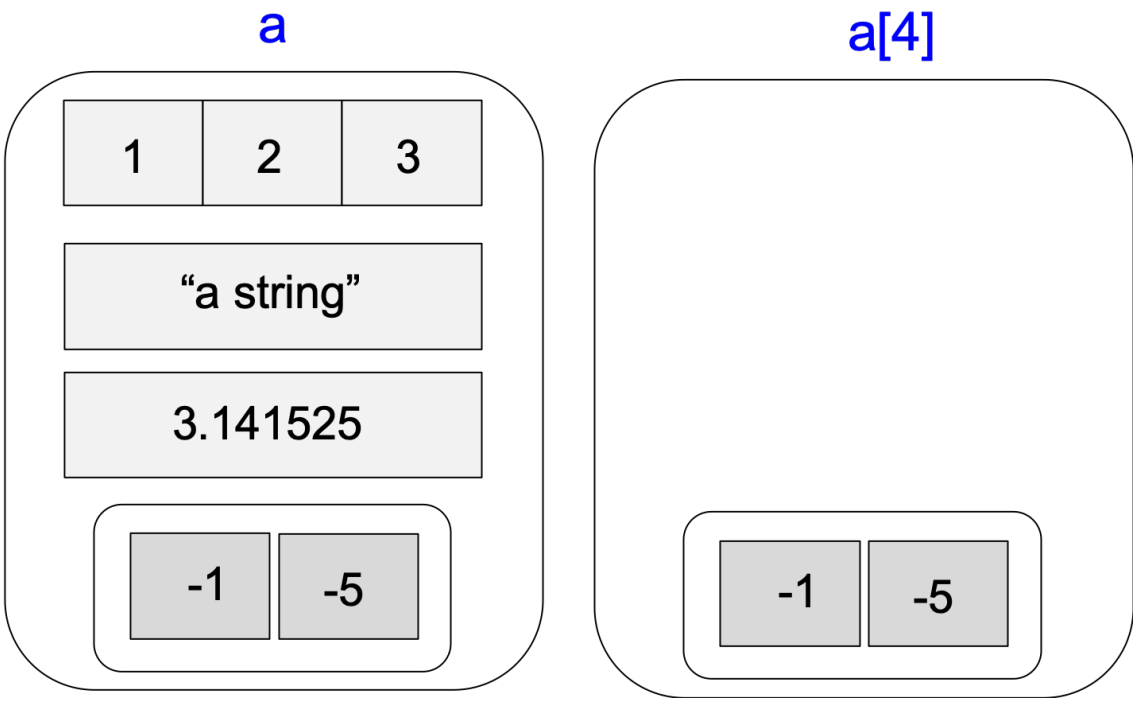
```
str(a)
```

```
## List of 4
##  $ el1: int [1:3] 1 2 3
##  $ el2: chr "a string"
##  $ el3: num 3.14
##  $ el4:List of 2
##   ..$ : num -1
##   ..$ : num -5
```

Exploring element 4

```
str(a[4])
```

```
## List of 1
##  $ el4:List of 2
##   ..$ : num -1
##   ..$ : num -5
```



Exploring the fourth element (a list)

```
str(a[[4]])
```

```
## List of 2
##  $ : num -1
##  $ : num -5
```

```
str(a[[4]][1])
```

```
## List of 1
##  $ : num -1
```

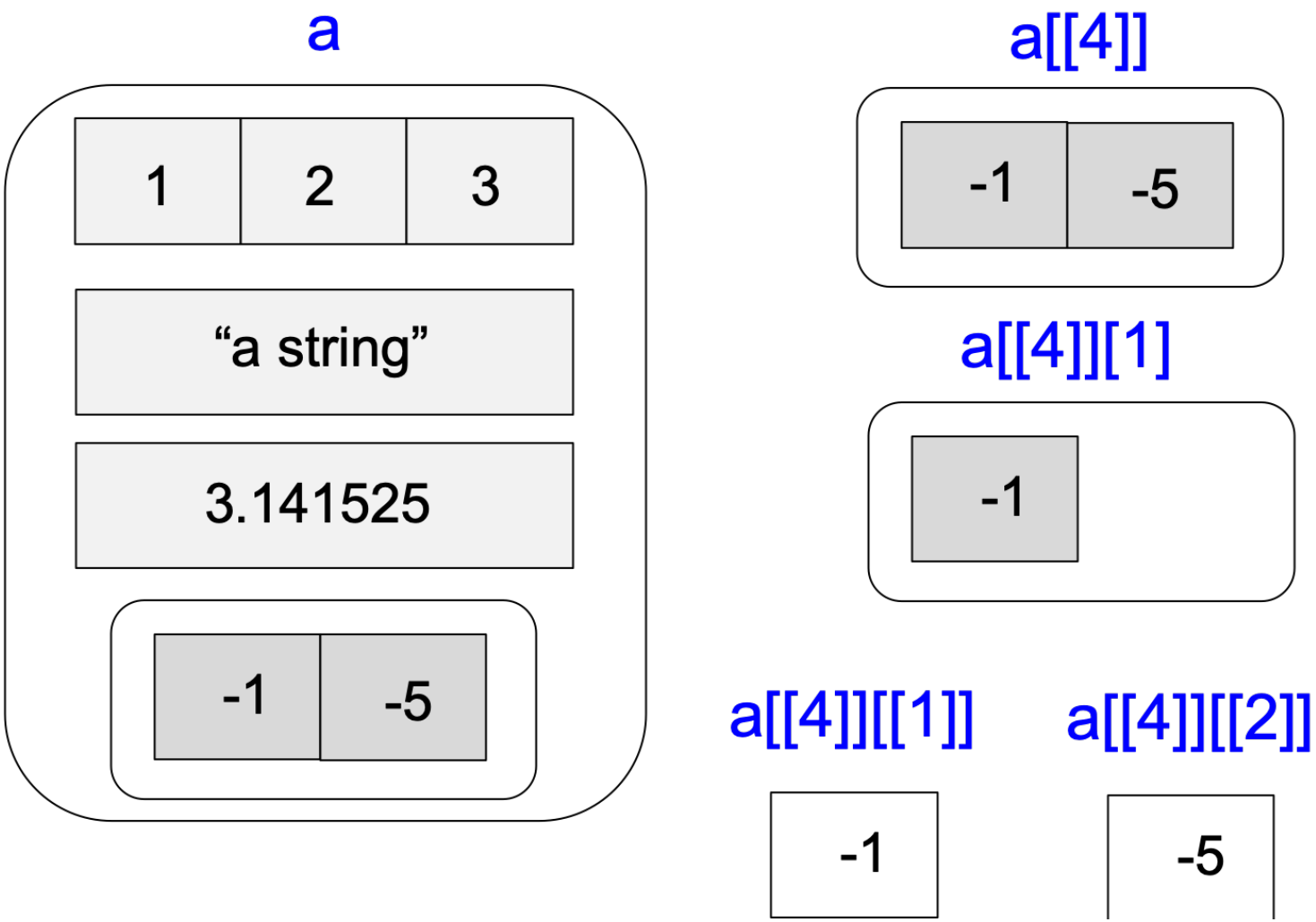
```
str(a[[4]][[1]])
```

```
##  num -1
```

```
str(a[[4]][[2]])
```

```
##  num -5
```

Visualising these operations



Challenge 2.1

```
l1 <- list(1:4,list(2:4,c(T,F,T)),1:10)
```

- What is the structure of l1[[2]]?
- What is the structure of l1[[2]][1]?
- Sum all of the elements of the first element of the embedded list
- Explain the following result

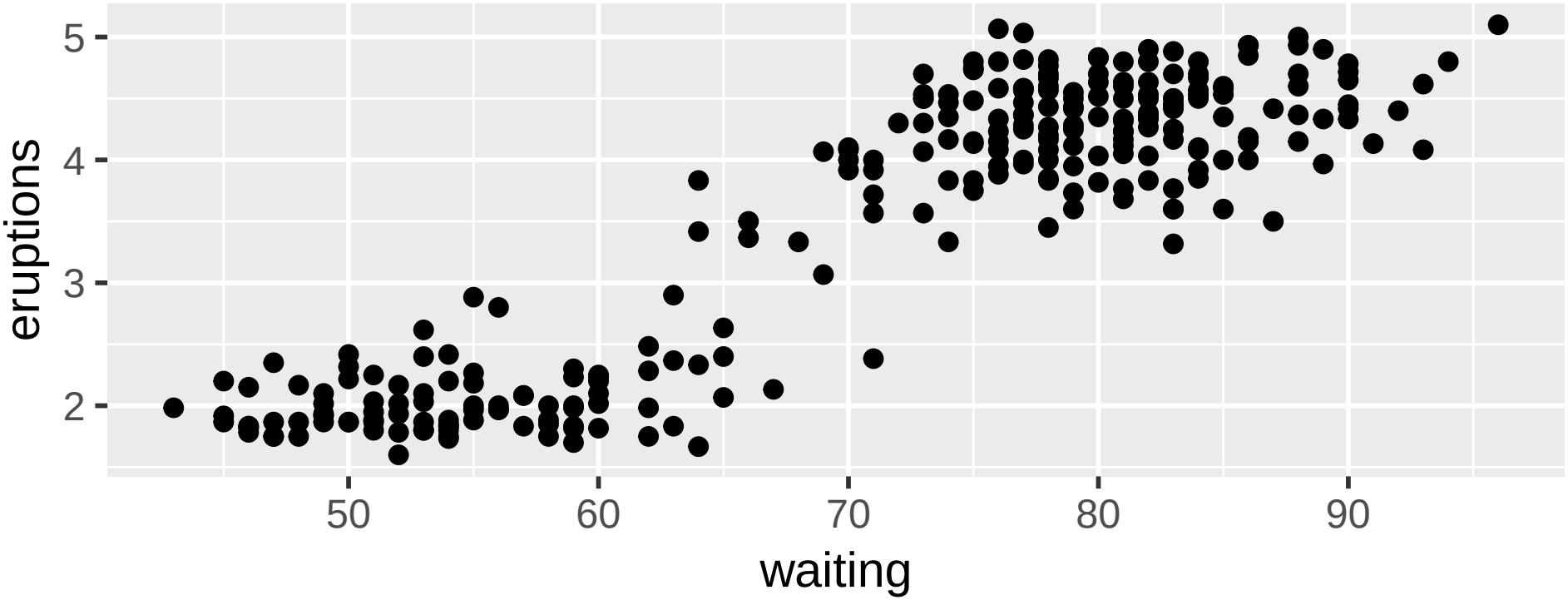
```
unlist(l1[[2]])
```

```
## [1] 2 3 4 1 0 1
```

Using Lists

- Are the basis of many S3 objects that are returned from regression functions (e.g. linear regression)
- The basis for data frames (the $ operator identifies columns)

```
ggplot(data=faithful)+
 geom_point(aes(x=waiting,y=eruptions))
```



lm function - Returns a list of 12

```
mod <- lm(eruptions ~ waiting, data=faithful)
mod$coefficients
```

```
## (Intercept)     waiting
## -1.87401599  0.07562795
```

```
class(mod)
```

```
## [1] "lm"
```

```
coefficients(mod)
```

```
## (Intercept)     waiting
## -1.87401599  0.07562795
```

```
str(coefficients(mod))
```

```
##  Named num [1:2] -1.874 0.0756
##  - attr(*, "names")= chr [1:2] "(Intercept)" "waiting"
```

Summary - Lists

- Lists are vectors, but can store many different types
- [] returns a subset of a list (as a list),
- [[]] returns the list contents
- The function **unlist()** will convert a list to an atomic vector
- The tag ($) operator is very useful (and used in data frames)
- In regression (and other machine learning algorithms in R), result information is returned as a list

Functions

- A function is a group of instructions that:
  - takes input,
  - uses the input to compute other value, and
  - returns a result (Matloff 2009).
- Functions are a fundamental building block of R
- Users of R should adopt the habit of creating simple functions which will make their work more effective and also more trustworthy (Chambers 2008).
- Functions:
  - are declared using the function reserved word
  - are objects

General Form

- **function** (*arguments*) *expression*
- arguments gives the arguments, separated by commas.
- Expression (body of the function) is any legal R expression, usually enclosed in { }
- Last evaluation is returned
- return() can also be used, but usually for exceptions.

```
f <- function(x)x^2 # this function squares a vector
f(1:3)
```

```
## [1] 1 4 9
```

Challenge 2.2

Write an R function (evens) that filters a vector to return all the even numbers. Use the modulus operator %%, and also logical filtering of vectors.

```
x <- 1:6
x
```

```
## [1] 1 2 3 4 5 6
```

```
y <- evens(x)
y
```

```
## [1] 2 4 6
```

Function Arguments

- It is useful to distinguish between formal arguments and the actual arguments
  - Formal arguments are the property of the function
  - Actual arguments can vary each time the function is called.
- When calling functions, arguments can be specified by
  - Complete name
  - Partial name
  - Position
- Guidelines (Wickham 2015)
  - Use positional mapping for the first one or two arguments (most commonly used)
  - Avoid using positional mapping for less commonly used attributes
  - Named arguments should always come after unnamed arguments

Function Arguments - Example

```
f1 <- function(arg1, arg2, arg3) arg1 * arg2  + arg3
f1(2, 3, 4) # positional
```

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

```
f1(2, arg3=4,3) # name for arg3
```

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

```
f1(arg3=4, arg2=3, 2) # name for arg2, arg3
```

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

Default Arguments

- Function arguments in R can have default values
- R function arguments are "lazy" – only evaluated if actually used

```
g <- function(a=1,b=2) c(a,b)

g()
```

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

```
g(10)
```

```
## [1] 10  2
```

```
g(10,20)
```

```
## [1] 10 20
```

Functions are objects

- Functions are first class objects, so they can be passed to other functions
- Provides flexibility, and widely used in R

```
f1 <- function(f,v)f(v) # f is a function object

f1(min,c(2,4,6,7))
```

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

```
f1(max,c(2,4,6,7))
```

```
## [1] 7
```

Challenge 2.3

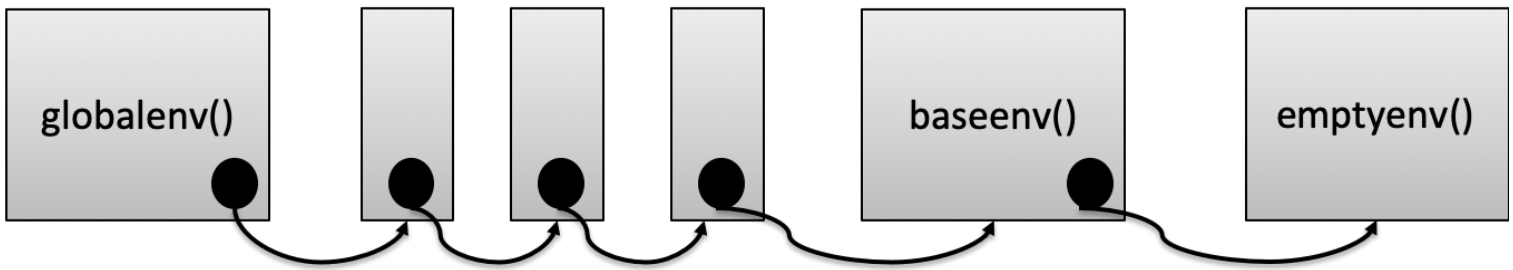
Write a function that takes in a vector and returns a vector with no duplicates. Make use of the R function duplicated().

```
x <- c(1, 2, 3, 4, 5, 1)
duplicated(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Environments

- Environments can be thought of as consisting of two things: a frame, which is a set of symbol-value pairs, and an enclosure, a pointer to an enclosing environment
- Every object (variable or function) in an environment has a unique name
- The working environment is known as the Global Environment
- Environments form a tree structure. The tree of environments is rooted in an empty environment, available through emptyenv(), which has no parent



Using search() to explore the hierarchy

```
search()
```

```
##  [1] ".GlobalEnv"        "package:ggplot2"   "package:stats
##  [4] "package:graphics"  "package:grDevices" "package:utils
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

Functions and Environments

- Functions are first class objects that exist in an environment
- Functions can access all variables contained in their enclosing environment
- If a name isn't defined inside a function, R will look one level up to the enclosing environment

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

g()
```

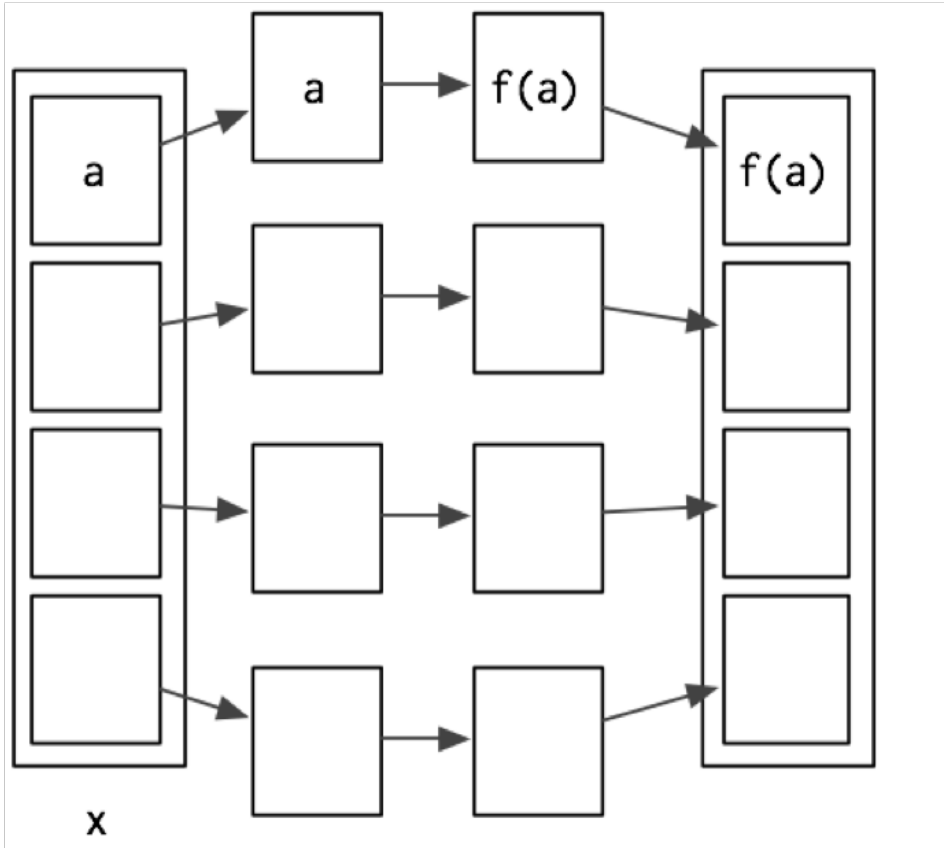
```
## [1] 2 1
```

Functionals

- A functional is a function that takes a function as an input and returns a vector as output
- Commonly used as an alternative for loops
- Common ones
  - sapply()
  - apply()
  - lapply()

Common Pattern (Wickham 2015)

- Create a container for output
- Apply f() to each component of the list
- Fill the container with the results



sapply()

- The general form of the **sapply(x,f,fargs)** function is as follows:
  - **x** is the target vector or list
  - **f** is the function to be called and applied to each element
  - **fargs** are the optional set of arguments that can be applied to the function f.
- sapply() returns a vector

```
x <- 1:3
y <- sapply(x,function(v)v*2)
y
```

```
## [1] 2 4 6
```

apply() - process matrices/data frames

The general form of this function is **apply(m, dimcode, f, fargs)**, where: - m is the target matrix - dimcode identifies whether it's a row or column target. The number 1 applies to rows, whereas 2 applies to columns - f is the function to be called, and fargs are the optional set of arguments that can be applied to the function f.

```
m <- matrix(1:10,nrow = 2)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
apply(m,1,sum) # sum the row
```

```
## [1] 25 30
```

```
apply(m,2,sum) # sum the columns
```

Summary - Functions

- Functions are a fundamental building block of R
- Functions:
  - are declared using the function reserved word
  - are objects
- Functions can access variables within the environment where they are created
- Functionals are functions that takes a function as an input and returns a vector as output (can be used as a looping structure)
- The apply family in R are functionals (**apply**, **sapply**, **lapply**)
- The package **purrr** is now being used instead of **apply**