Eric Pitman Summer Workshop in Computational Science

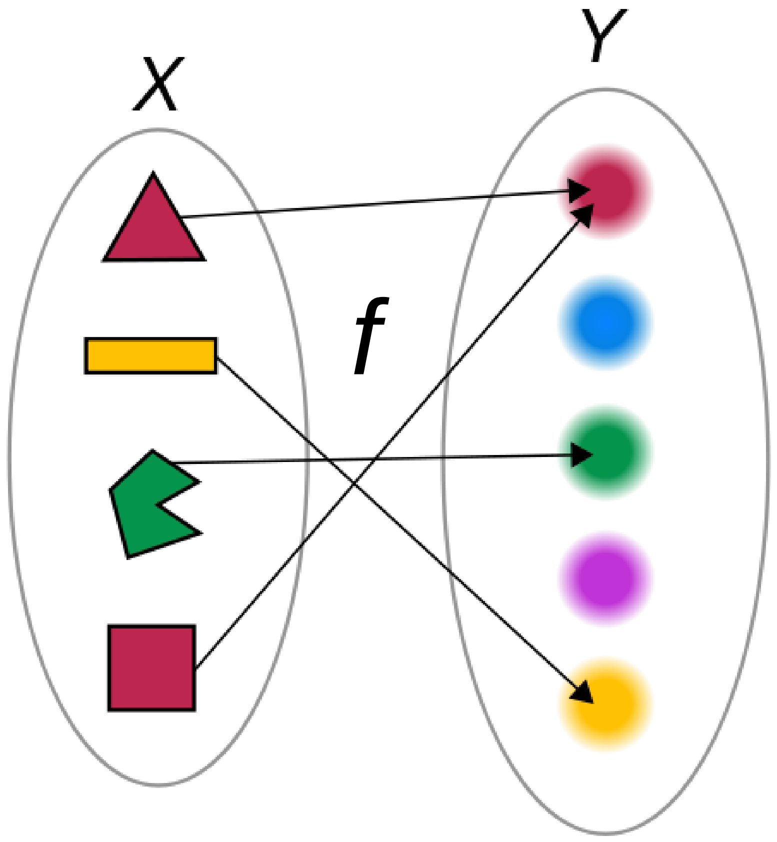
4. Writing Functions

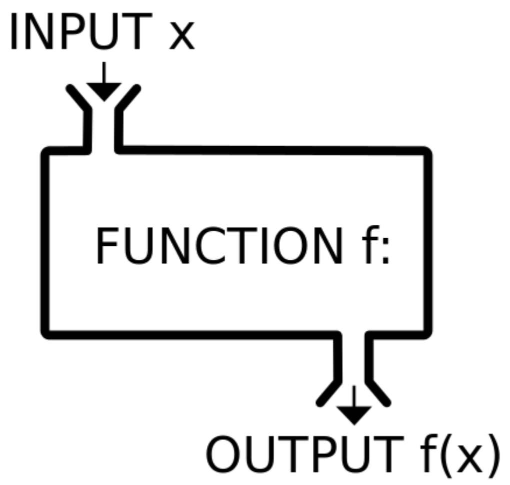
CENTER FOR COMPUTATIONAL RESEARCH

University at Buffalo
The State University of New York

Functions

The function *f* generates an output (Y), given an input (X).



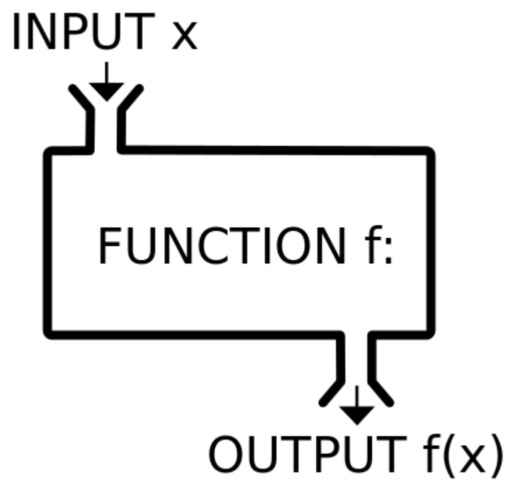


Functions

A piece of code that can be called again and again

To call it, specify:
- Function name
- Input values

It may return an output value



Functions

- We use native R functions all the time! Examples:
  class(), str(), summary()

- You can also write your own.

Function Syntax in R

Name of function

Input parameter(s)

```
functionName = function(inputs) {
  # do something
  # return the result
}
```

Declaration
(start of function)

End of function

Function Syntax in R

Name of function

Input parameter(s)

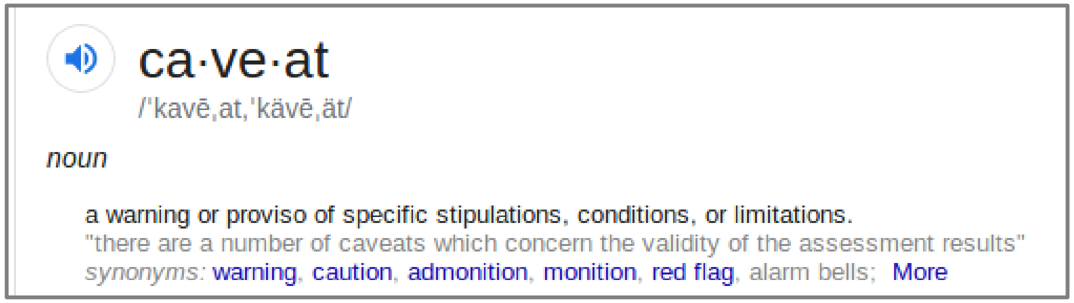
```r
toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32; # calculate
  return(f); # return the result
}
```

Declaration
(start of function)

Output value

End of function

Caveats


ca·ve·at
/ˈkavē,at,ˈkävē,ät/

noun

a warning or proviso of specific stipulations, conditions, or limitations.
"there are a number of caveats which concern the validity of the assessment results"
synonyms: warning, caution, admonition, monition, red flag, alarm bells; More

About return()

```
NotToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  g = 97; # R will return the last computed value
}
```

About return()

```
NotToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  return(f);
  g = 97; # Do we ever execute this line?
}
```

Accepted inputs?

```
ToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  return(f);
}
```

Q: Do I need to write a loop to call the function with multiple values?

Accepted inputs?

```
ToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  return(f);
}
```

Q: Do I need to write a loop to call the function with multiple values?

A: NOPE! These operations accept vectors.

Calling *toFahrenheit()*

```
celsius = c(20:25); # define input temperatures

toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32;  # perform the conversion
  return(f);
}

# call the function to convert temperatures to Fahrenheit:
toFahrenheit(celsius);
[1] 68.0  69.8  71.6  73.4  75.2  77.0
```

Control Structures

Control Structures: if/else

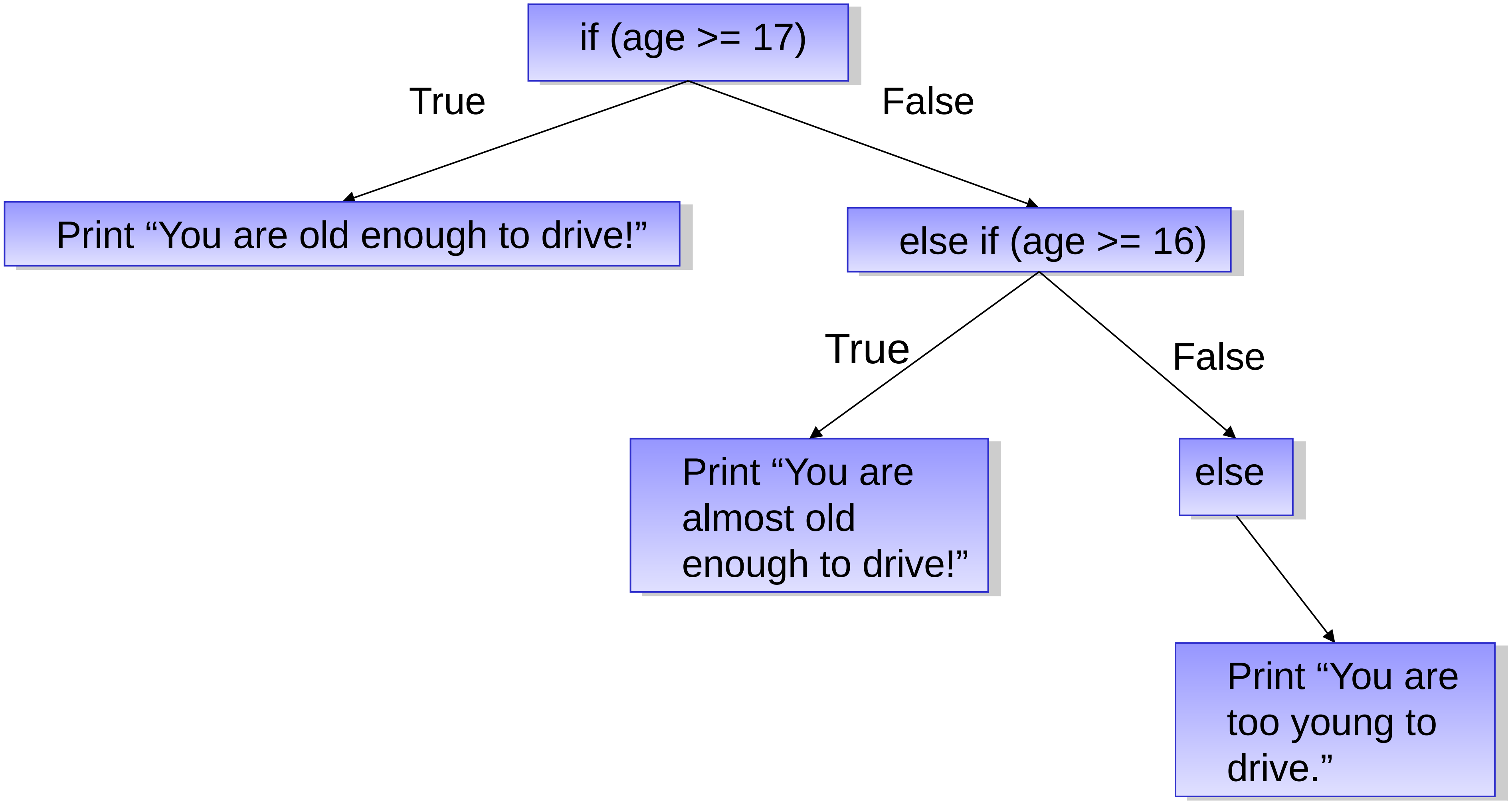
- Make a logical test
- Perform operations based on the outcome

```
if (condition is true)
{
    # do something
}
```

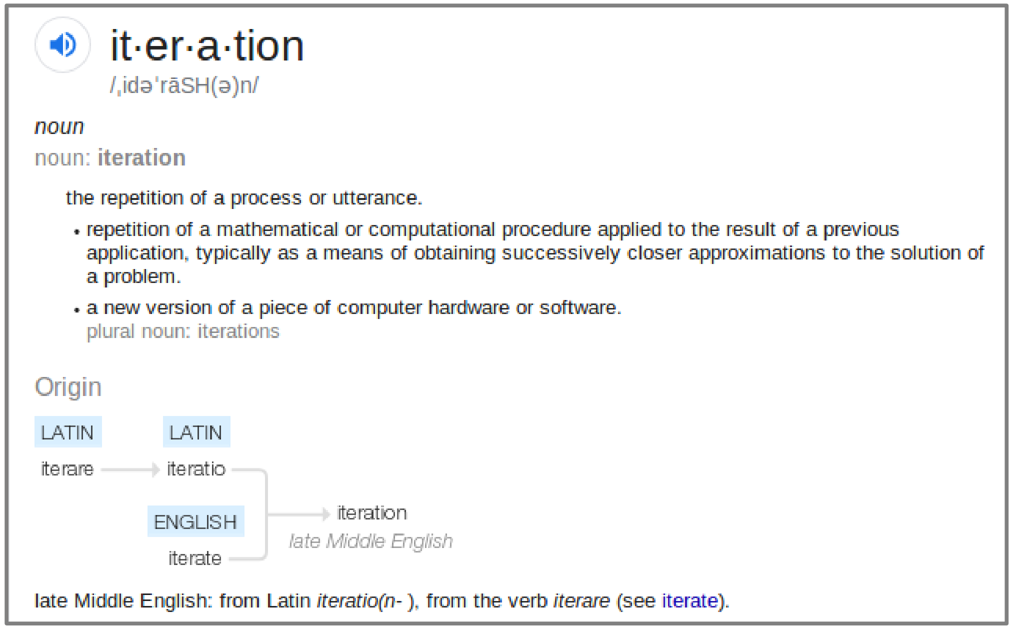
Control Structures: if/else

```
age = 21;

if (age >= 17) {

    print(“You can drive!”);

} else if (age >= 16) {

    print(“You are almost old enough to drive!”);

} else {

    print(“You are not old enough to drive.”);
}
```

if/else Flowchart


if (age >= 17)
True
False
Print “You are old enough to drive!”
else if (age >= 16)
True
False
Print “You are almost old enough to drive!”
else
Print “You are too young to drive.”

Iteration



it·er·a·tion
/ˌidəˈrāSH(ə)n/

noun
noun: **iteration**

the repetition of a process or utterance.

- repetition of a mathematical or computational procedure applied to the result of a previous application, typically as a means of obtaining successively closer approximations to the solution of a problem.
- a new version of a piece of computer hardware or software.
  plural noun: iterations

Origin

late Middle English: from Latin *iteratio(n-)*, from the verb *iterare* (see iterate).

Control Structures: Iteration

- What if we want to call a function over and over?
- Other languages use loops.
- R can do this with a single line of code!
- Use it on native R functions, or functions you wrote yourself.

Definitions

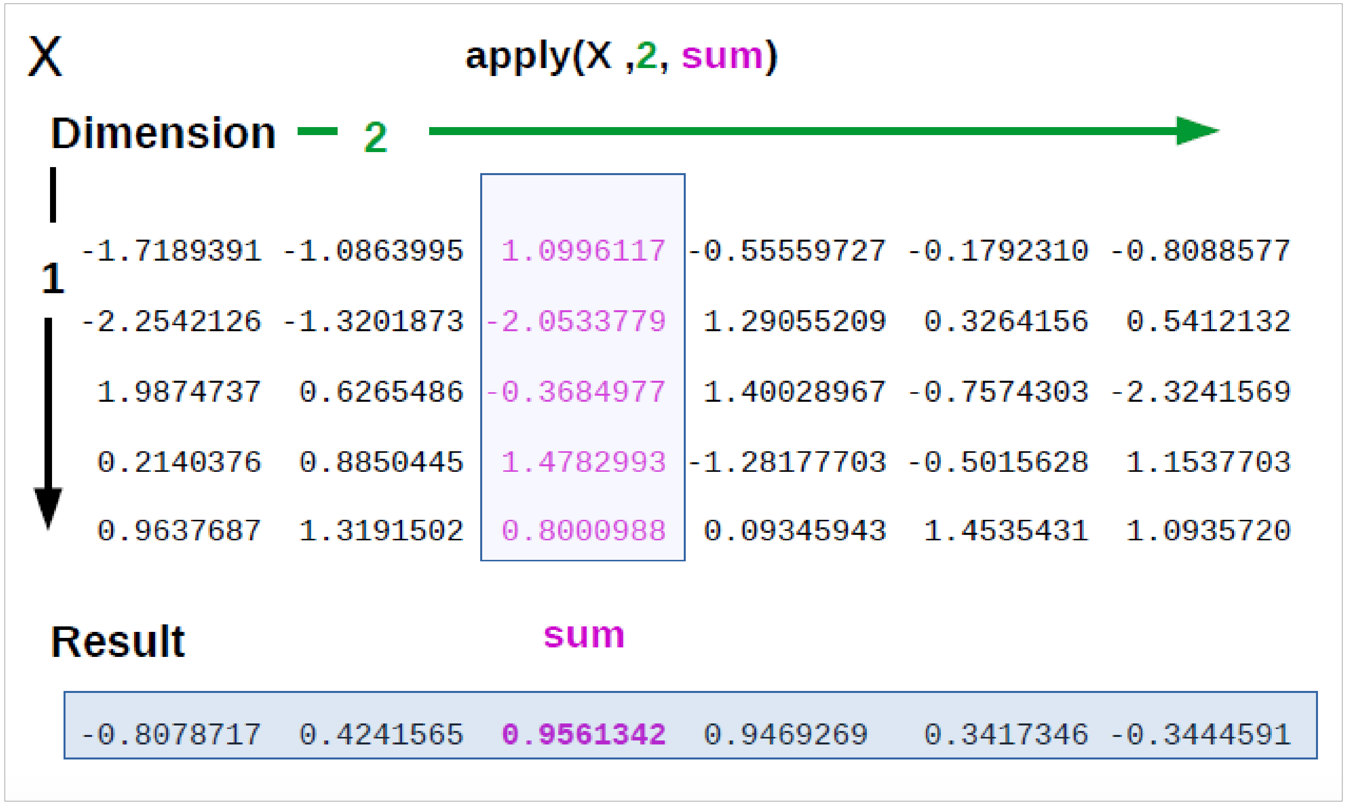
- Functional languages consider functions first class citizens.
  - Functions can be assigned to variables, stored in lists, passed as arguments, and returned from calls to other functions.
- Vectorized operations execute as precompiled C code, hiding their loops. They are fast—even on vectors.

Meet the apply() family

- by(): with factors
- apply(): returns vector
- sapply(): returns vector or matrix
- lapply()
- mapply()
- rapply()
- ...

apply(X, MARGIN, FUN,...)

Returns a vector obtained by applying a function to margins of a matrix:



```
-1.7189391 -1.0863995  1.0996117 -0.55559727 -0.1792310 -0.8088577
-2.2542126 -1.3201873 -2.0533779  1.29055209  0.3264156  0.5412132
 1.9874737  0.6265486 -0.3684977  1.40028967 -0.7574303 -2.3241569
 0.2140376  0.8850445  1.4782993 -1.28177703 -0.5015628  1.1537703
 0.9637687  1.3191502  0.8000988  0.09345943  1.4535431  1.0935720
```

```
-0.8078717  0.4241565  0.9561342  0.9469269   0.3417346 -0.3444591
```

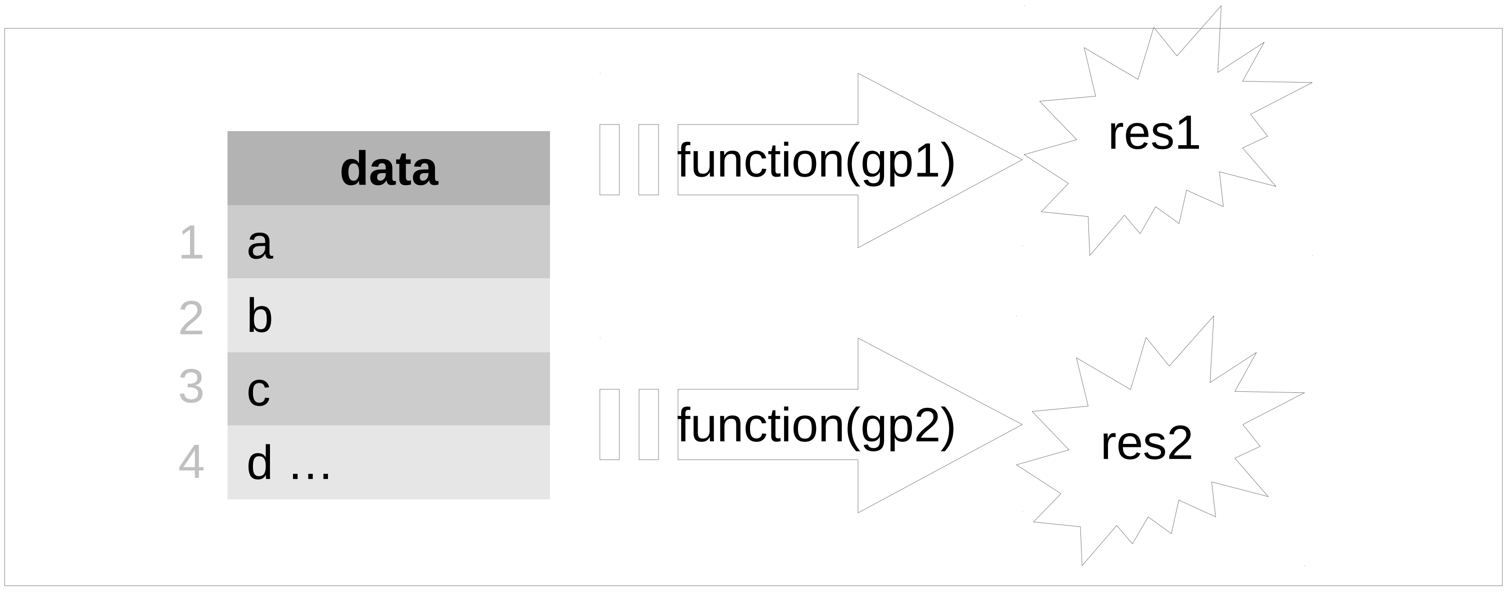
Control Structures: by()

- What if we want to call a function several times, on several *groups* of data?
- We can use a single line of R code:

by(*data*, *group*, *function*)

Group and operate: by()

```
by(data-to-operate-on,
   factor-to-group-by,
   function)
```



Group and operate: by()

```
> by(data$Height, data$Hand, max)
```

Vector to operate on

Factor for grouping

Function

```
     Height Weight Age Hand
1      68        120  16    L
2      75        160  17    R
3      60        118  16    R
```

Group and operate: by()

```
> by(data$Height, data$Hand, max)
 L      R
68    75
```

```
     Height Weight Age Hand
1      68       120  16    L
2      75       160  17    R
3      60       118  16    R
```

iris and by()

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         5.1         3.5          1.4         0.2  setosa
         4.9         3.0          1.4         0.2  setosa
         4.7         3.2          1.3         0.2  setosa
```

Compute summaries and means of data, grouping by Species:

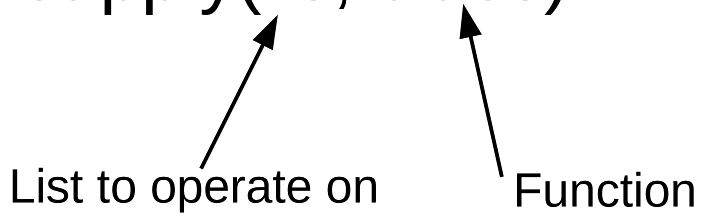
<workshop>/examples/by-example.R

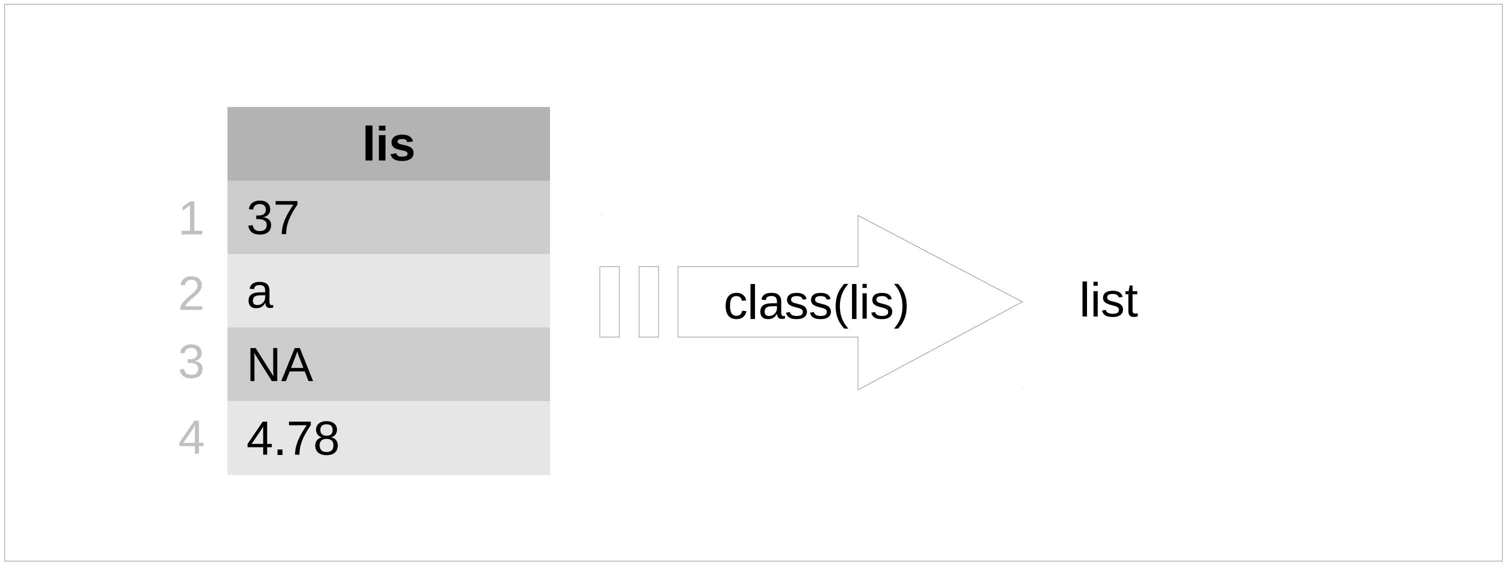
iris and by()

```
> by(iris[,c(1:3)], Species, colMeans)

Species: setosa
Sepal.Length  Sepal.Width Petal.Length 
       5.006        3.428        1.462 
-------------------------------------------------------
Species: versicolor
Sepal.Length  Sepal.Width Petal.Length 
       5.936        2.770        4.260 
-------------------------------------------------------
Species: virginica
Sepal.Length  Sepal.Width Petal.Length 
       6.588        2.974        5.552 
```

Discriminate and operate: sapply()

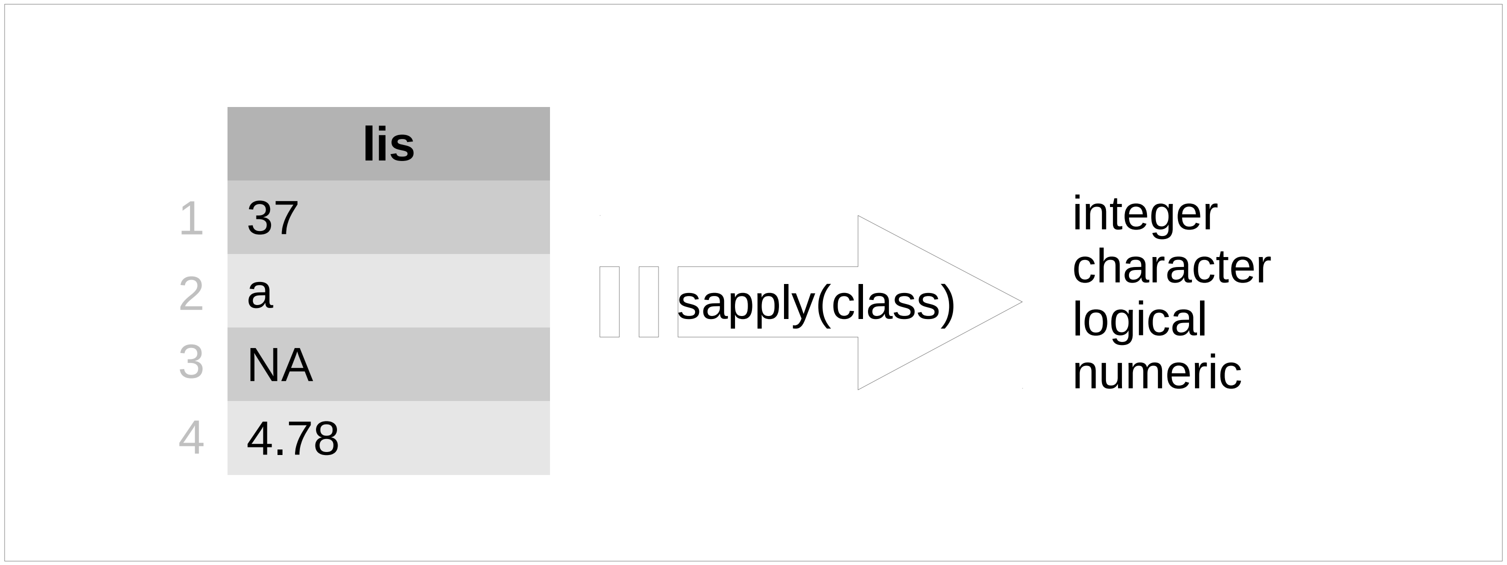
```
> lis = list(37, "a", NA, 4.78)
> sapply(lis, class)
```





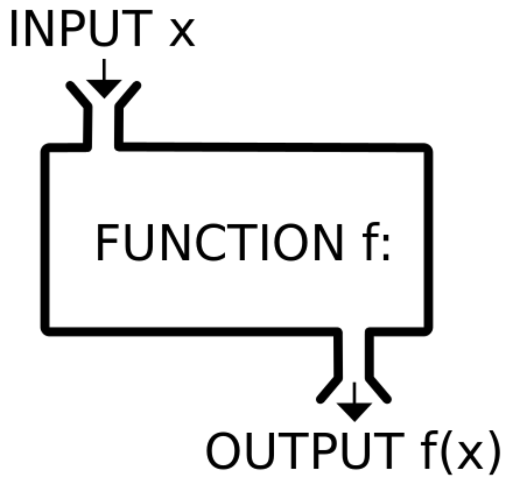
Discriminate and operate: sapply()

```
> lis = list(37, "a", NA, 4.78)
> sapply(lis, class)

"integer" "character" "logical" "numeric"
```



Tips: Writing Functions

- Use an editor window (not the command line) to compose functions
- Try out one line at a time, and test!
- Start with the simplest case and build.
- Comment your function to indicate:
  - input
  - output
  - purpose



Student Dataset Example

Remember our own dataset:

```
firstInitial, lastInitial, school, height, htUnit, age,
handed, gender
```

Let's write functions that:

- Convert heights to a uniform unit
- List initials of students that are old enough to drive

Interlude

Complete function exercises.

Open in the RStudio source editor:

<workshop>/exercises/4-exercises-functions.R

Interlude++

Function reading assignment:

“How to write and debug an R function”:

https://vidia.ccr.buffalo.edu/resources/686