EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

A VARIABLE IS A CONTAINER IN WHICH A VALUE IS STORED

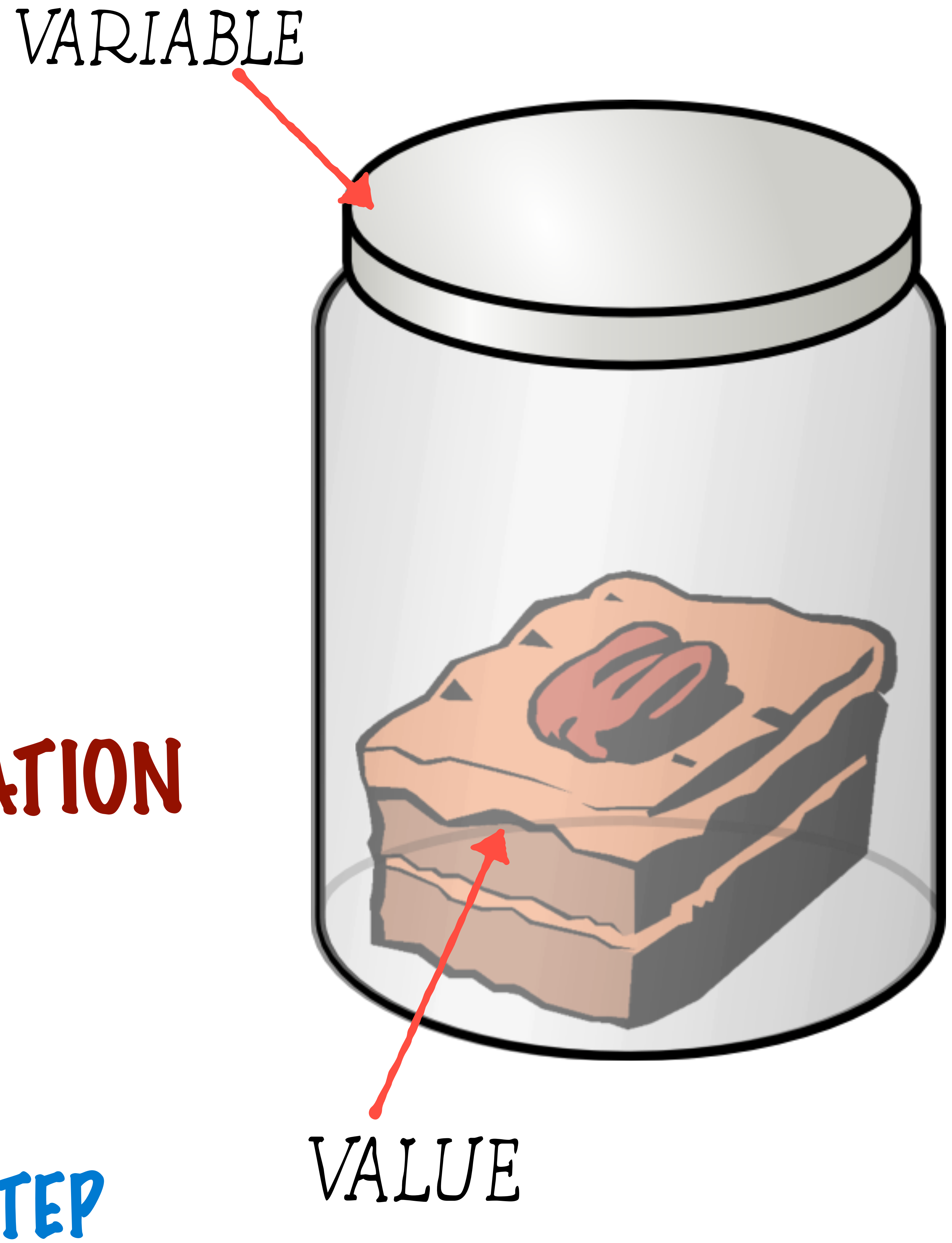
WHEN YOU ARE PROGRAMMING, YOU WANT TO STORE SOME VALUES FOR LATER

AN INPUT RECEIVED FROM A USER (OR)

THE RESULT OF A COMPLEX CALCULATION

IF YOU ASSIGN THE VALUE TO A VARIABLE

THAT VALUE IS AVAILABLE FOR LATER USE, OTHERWISE IT'S LOST AFTER THE CURRENT STEP



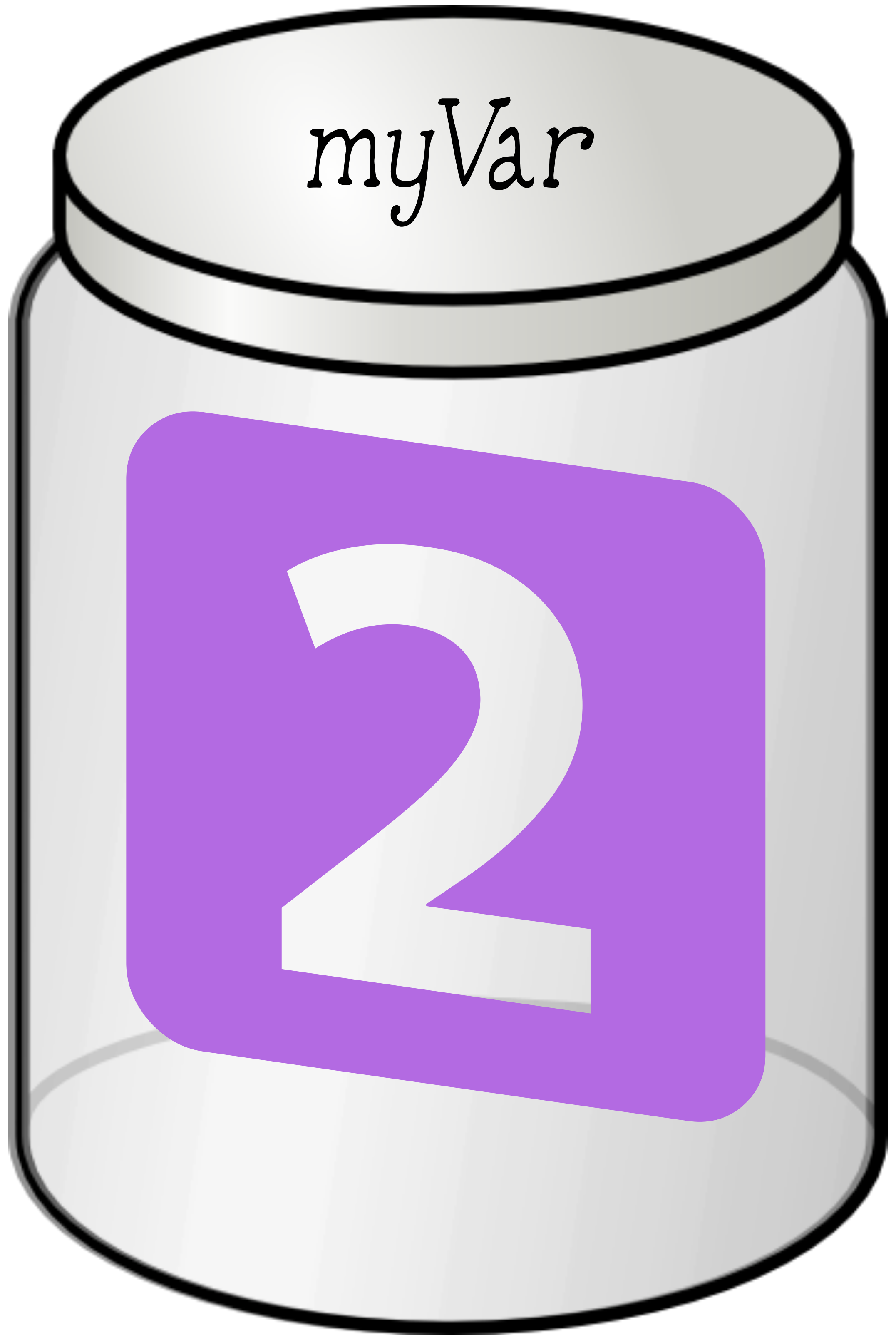
EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

WHEN YOU ASSIGN A VALUE TO A VARIABLE, YOU

1. CREATE A NEW CONTAINER

2. GIVE THE CONTAINER A NAME

3. STORE A VALUE IN THAT CONTAINER

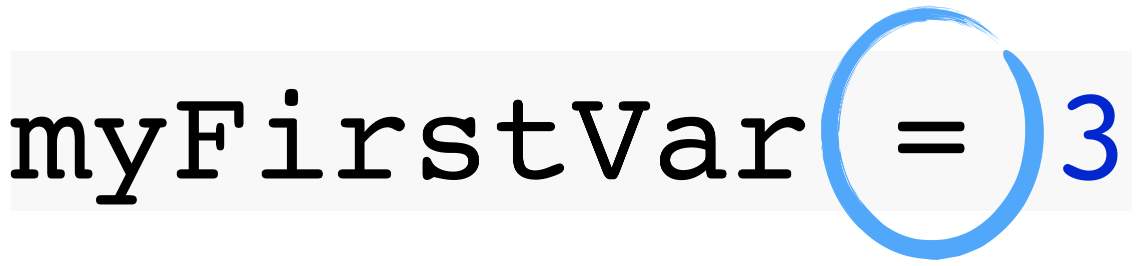


EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

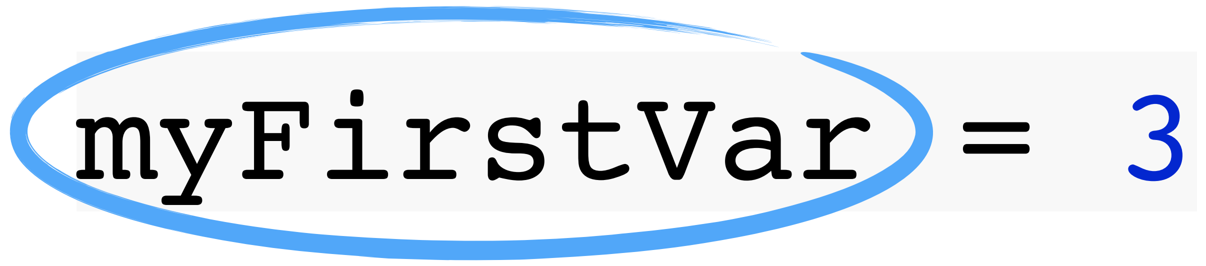
EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
myFirstVar = 3
```

LIKE WITH MOST PROGRAMMING LANGUAGES, YOU CAN ASSIGN A VALUE TO A VARIABLE USING THE = OPERATOR

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

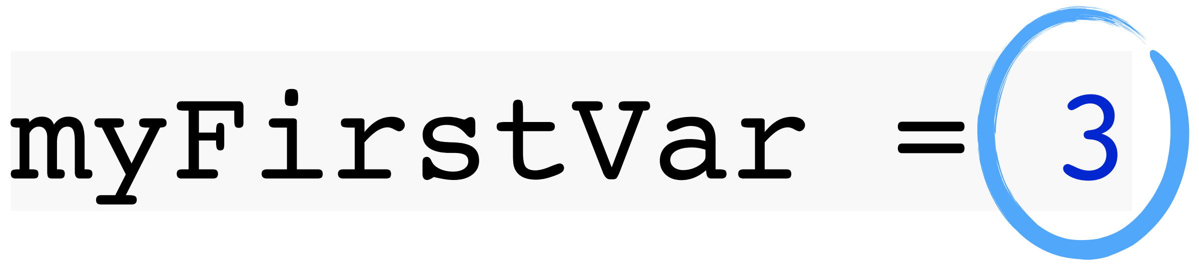


```
myFirstVar = 3
```

LIKE WITH MOST PROGRAMMING LANGUAGES, YOU CAN ASSIGN A VALUE TO A VARIABLE USING THE = OPERATOR

myFirstVar IS THE NAME OF THE VARIABLE

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



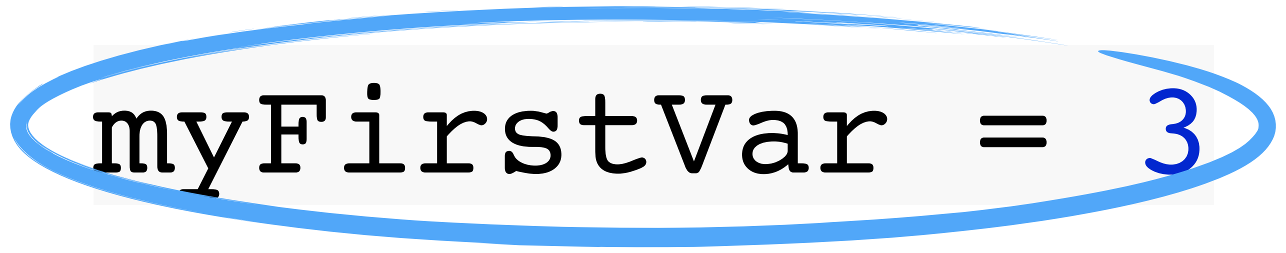
```
myFirstVar = 3
```

LIKE WITH MOST PROGRAMMING LANGUAGES, YOU CAN ASSIGN A VALUE TO A VARIABLE USING THE = OPERATOR

myFirstVar IS THE NAME OF THE VARIABLE

THE VALUE 3 IS ASSIGNED TO myFirstVar

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
myFirstVar = 3
```

LIKE WITH MOST PROGRAMMING LANGUAGES, YOU CAN ASSIGN A VALUE TO A VARIABLE USING THE = OPERATOR

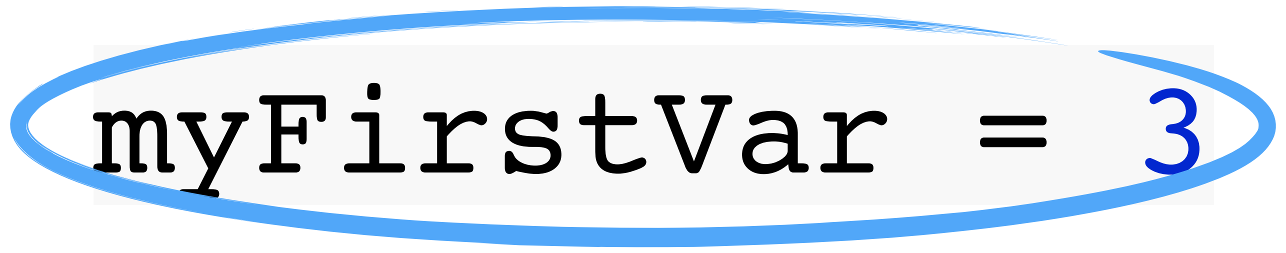
myFirstVar IS THE NAME OF THE VARIABLE

THE VALUE 3 IS ASSIGNED TO myFirstVar

IF myFirstVar DOESN'T EXIST BEFORE THIS, IT IS CREATED NOW

IF myFirstVar DOES EXIST IT'S OLD VALUE IS DISCARDED AND THE NEW VALUE 3 IS ASSIGNED

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
myFirstVar = 3
```

LIKE WITH MOST PROGRAMMING LANGUAGES, YOU CAN ASSIGN A VALUE TO A VARIABLE USING THE = OPERATOR

IF myFirstVar DOESN'T EXIST BEFORE THIS, IT IS CREATED NOW

IF myFirstVar DOES EXIST IT'S OLD VALUE IS DISCARDED AND THE NEW VALUE 3 IS ASSIGNED

myFirstVar IS THE NAME OF THE VARIABLE

THE VALUE 3 IS ASSIGNED TO myFirstVar

ONCE CREATED, A VARIABLE WILL BE AVAILABLE FOR USE, UNTIL THE R SESSION ENDS (OR) UNTIL IT'S DESTROYED BY THE PROGRAMMER

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

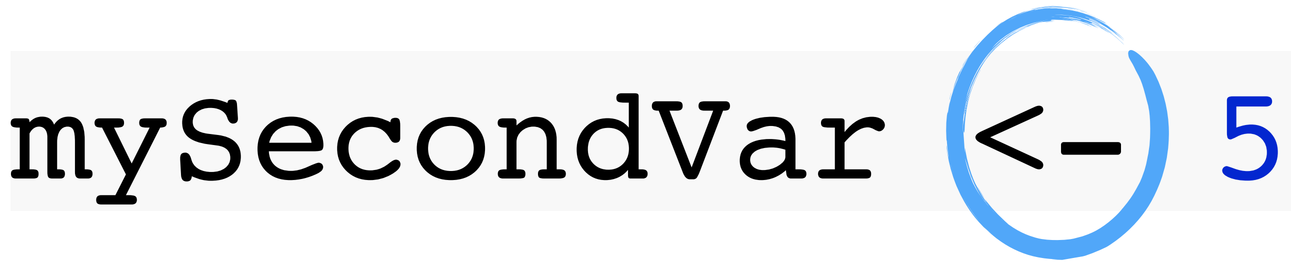
IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

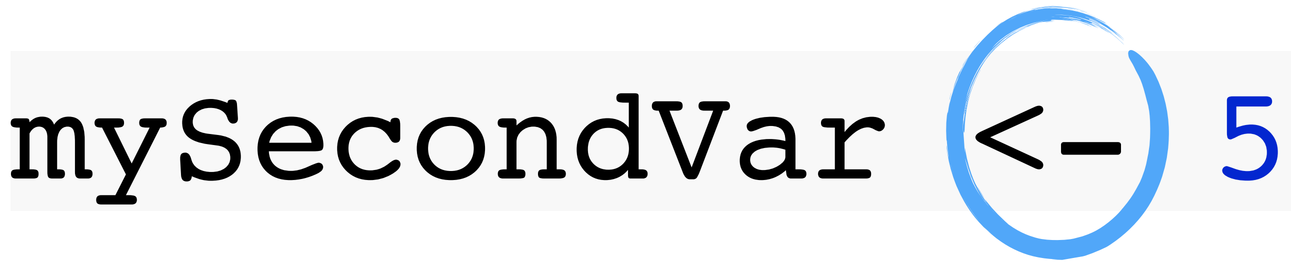
EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
mySecondVar <- 5
```

MANY R PROGRAMMERS **PREFER TO USE THE <- OPERATOR** FOR VARIABLE ASSIGNMENT

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

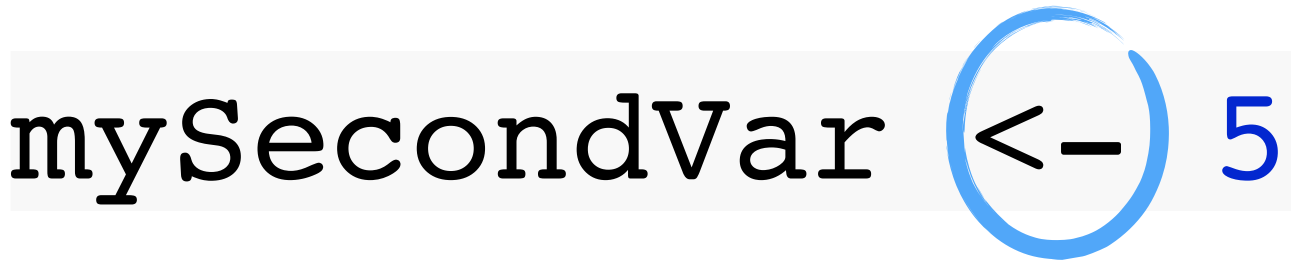


```
mySecondVar <- 5
```

MANY R PROGRAMMERS
PREFER TO USE THE <- OPERATOR
FOR VARIABLE ASSIGNMENT

WHEN R WAS FIRST WRITTEN, ASSIGNMENT COULD BE DONE ONLY USING THE <- OPERATOR (ARROW)

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



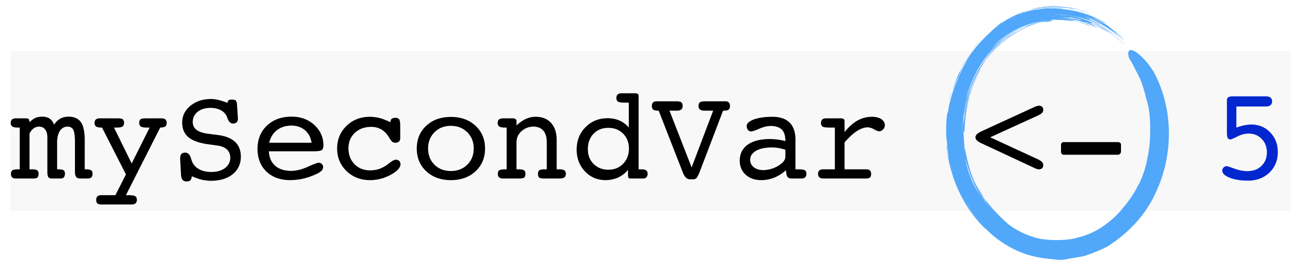
```
mySecondVar <- 5
```

MANY R PROGRAMMERS PREFER TO USE THE <- OPERATOR FOR VARIABLE ASSIGNMENT

WHEN R WAS FIRST WRITTEN, ASSIGNMENT COULD BE DONE ONLY USING THE <- OPERATOR (ARROW)

LATER, AS R STARTED BEING USED MORE WIDELY, THE = OPERATOR WAS ADDED, TO MAKE IT EASIER FOR PROGRAMMERS IN OTHER LANGUAGES LIKE C, PYTHON, JAVA ETC

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
mySecondVar <- 5
```

MANY R PROGRAMMERS PREFER TO USE THE <- OPERATOR FOR VARIABLE ASSIGNMENT

LATER, AS R STARTED BEING USED MORE WIDELY, THE = OPERATOR WAS ADDED, TO MAKE IT EASIER FOR PROGRAMMERS IN OTHER LANGUAGES LIKE C, PYTHON, JAVA ETC

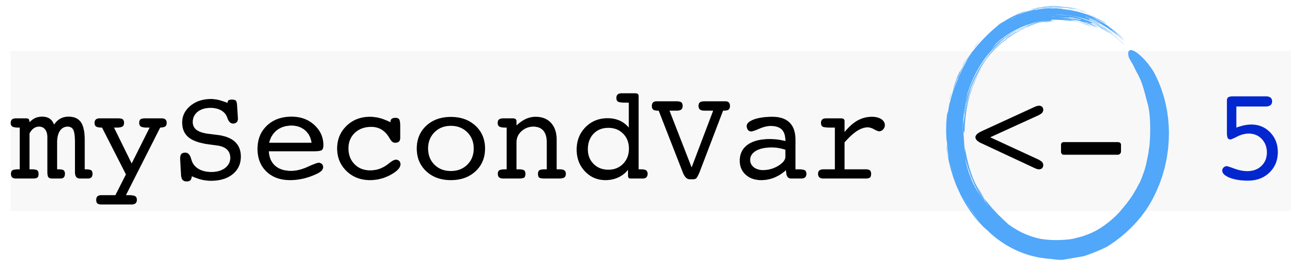
WHEN R WAS FIRST WRITTEN, ASSIGNMENT COULD BE DONE ONLY USING THE <- OPERATOR

THERE ARE SOME SPECIAL CASES WHERE USING THE = OPERATOR FOR ASSIGNMENT WON'T WORK AS INTENDED

WE WON'T WORRY ABOUT THEM RIGHT NOW..

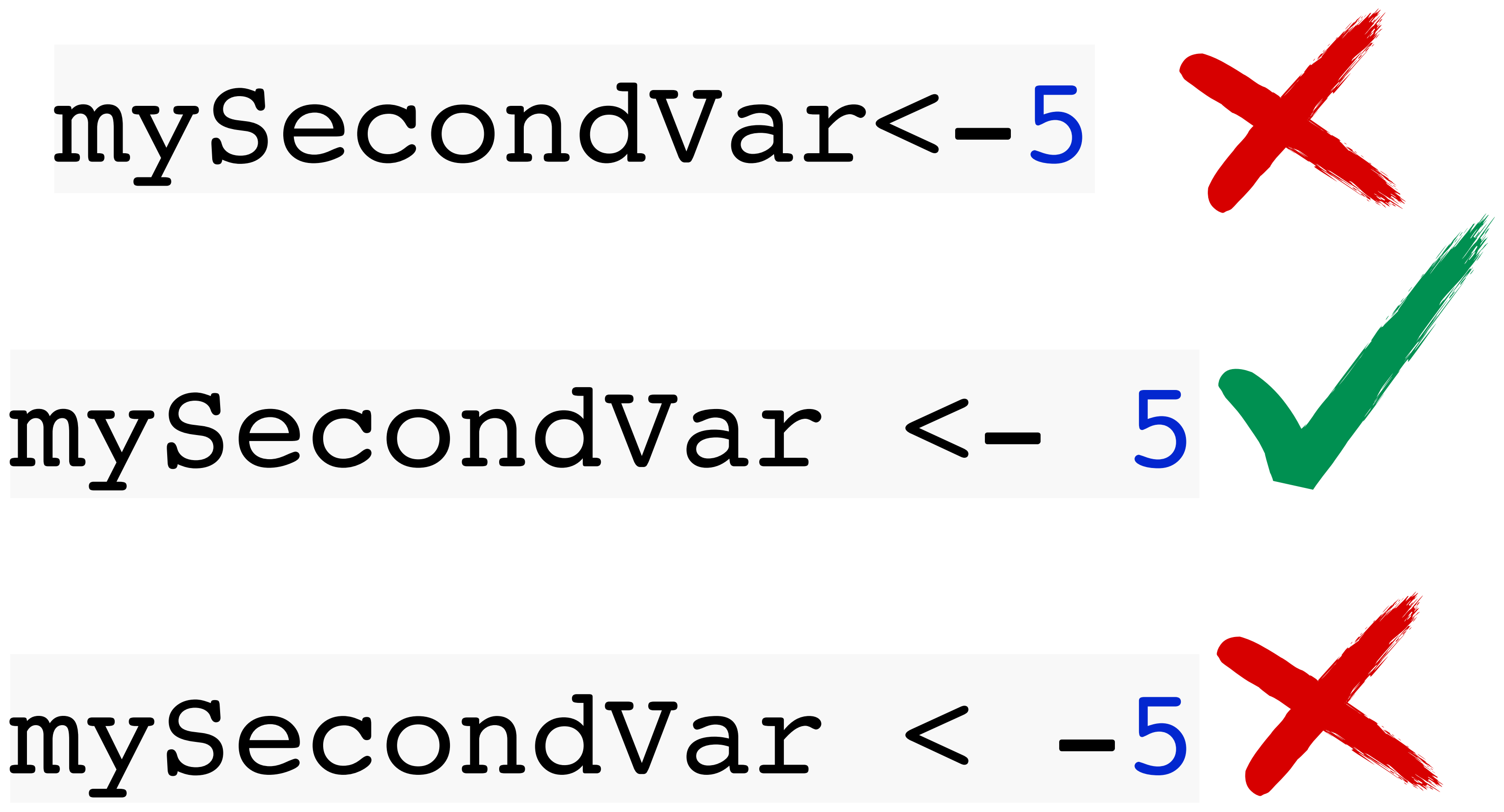
EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

MANY R PROGRAMMERS
PREFER TO USE THE <- OPERATOR
FOR VARIABLE ASSIGNMENT



```
mySecondVar <- 5
```

NOTE TO SELF: TAKE CARE WITH WHITESPACE WHEN USING THE <- OPERATOR



```
mySecondVar<-5
```

✗

```
mySecondVar <- 5
```

✓

```
mySecondVar < -5
```

✗

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

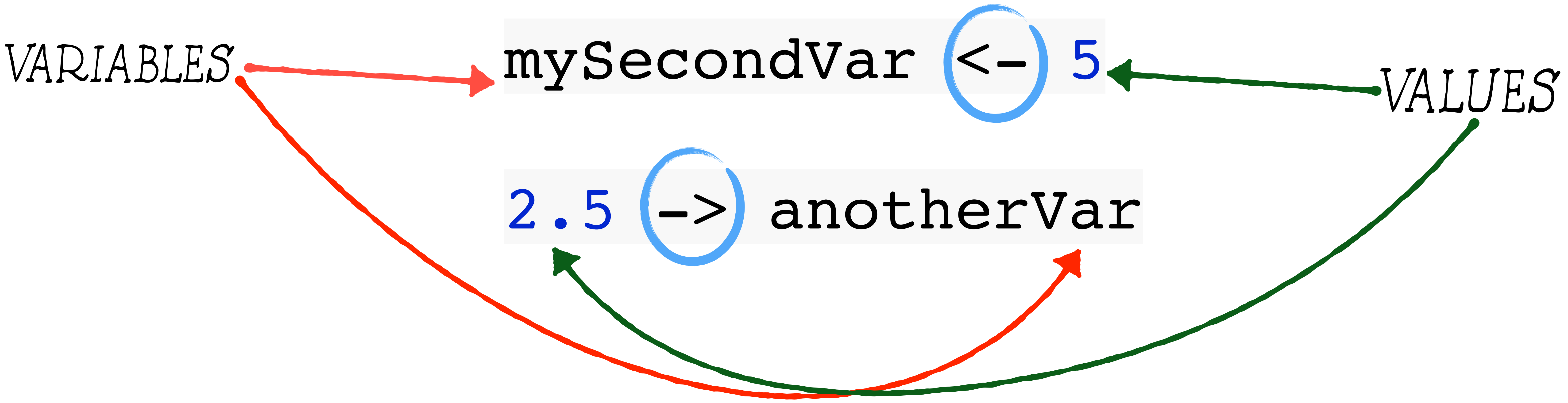
IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```r
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
mySecondVar <- 5
2.5 -> anotherVar
```

THE ARROW OPERATOR CAN BE USED **IN EITHER DIRECTION**

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```r
myFirstVar = 3
```

```r
mySecondVar <- 5

2.5 -> anotherVar
```

```r
this.Silly.Var <- that.Silly.Var <- “silly"
```

```r
assign("funnyWayToAssignVar", 20)
```

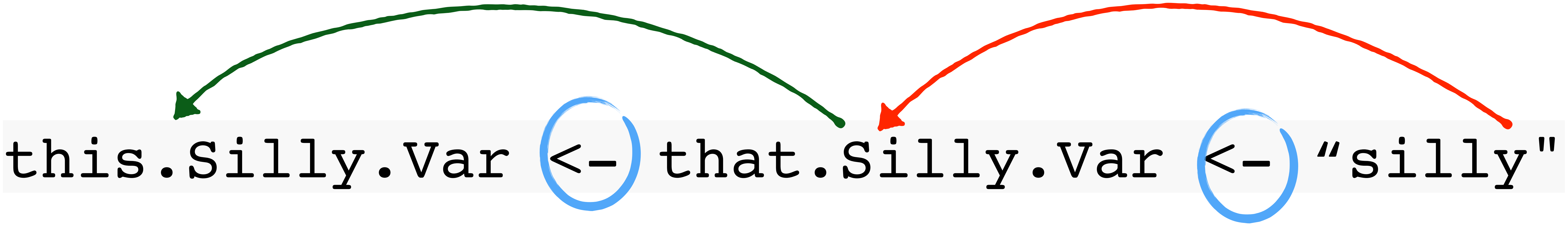
IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES



```
this.Silly.Var <- that.Silly.Var <- “silly"
```

YOU CAN **ASSIGN A VALUE** TO TWO **DIFFERENT VARIABLES** AT THE SAME TIME

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

EXAMPLE 1: ASSIGNING VALUES TO VARIABLES

IN R, VARIABLES CAN BE ASSIGNED IN A BUNCH OF DIFFERENT WAYS

```
myFirstVar = 3

mySecondVar <- 5

2.5 -> anotherVar

this.Silly.Var <- that.Silly.Var <- “silly"

assign("funnyWayToAssignVar", 20)
```

THIS IS A PRETTY FUNNY WAY TO ASSIGN VALUES TO VARIABLES...

...BUT HERE IT IS IN CASE YOU EVER HAVE A WILD HAIR TO USE IT

EXAMPLE 2: PRINTING AN OUTPUT

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS
TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

YOU CAN PRINT WITHOUT USING
ANY EXPLICIT FUNCTION

```
print(mySecondVar)

show(myFirstVar)

this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")

sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
```

USING THE PRINT() FUNCTION

```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
```

```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

USING THE SHOW() FUNCTION

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")

sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

THE CAT() FUNCTION

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS
TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
message(sillyMessage)
silly , silly are both the same
```

LET'S GO THROUGH
THEM ONE BY ONE

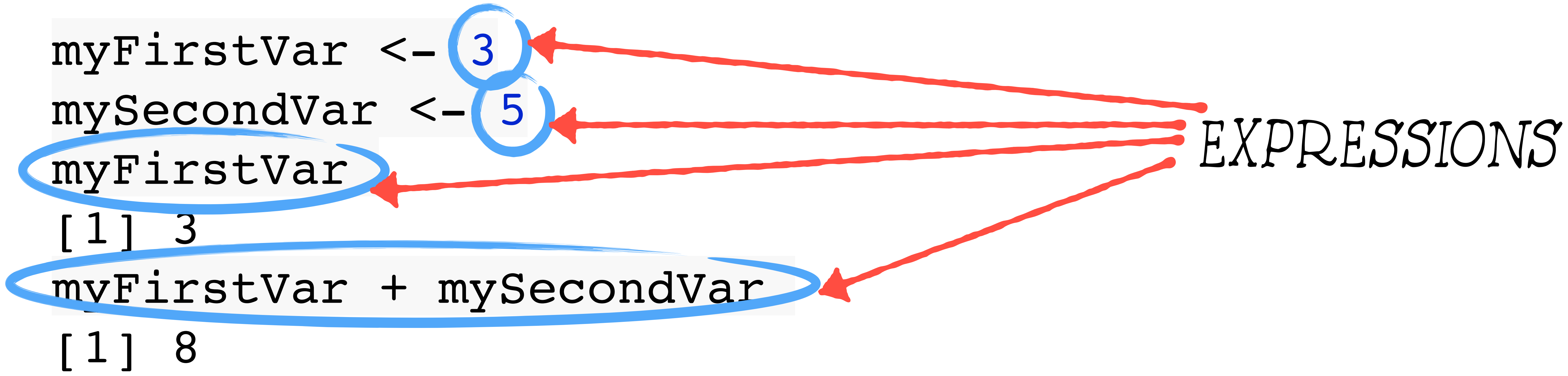
THE MESSAGE() FUNCTION

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS
TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT

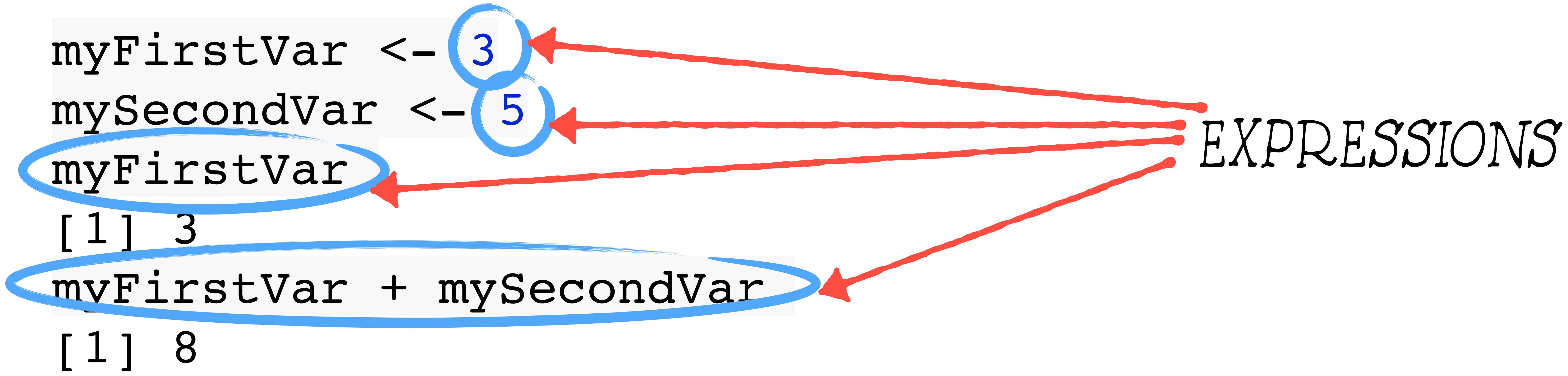


```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

A COMPUTATION IN R IS CALLED AN **EXPRESSION**

EXAMPLE 2: PRINTING AN OUTPUT

A COMPUTATION IN R IS CALLED AN EXPRESSION

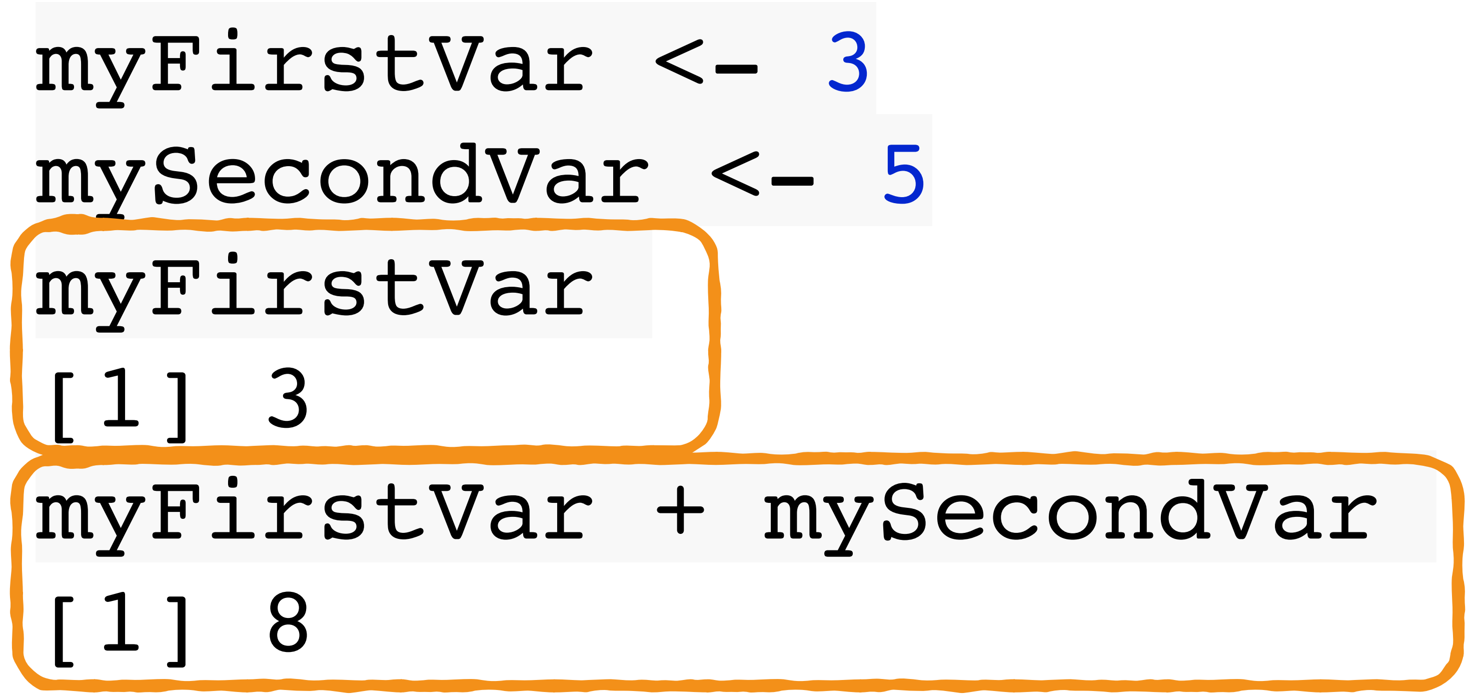


```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

WHENEVER R SEES AN EXPRESSION IT WILL EVALUATE IT IE COMPUTE A RESULT

EXAMPLE 2: PRINTING AN OUTPUT

A COMPUTATION IN R IS CALLED AN EXPRESSION

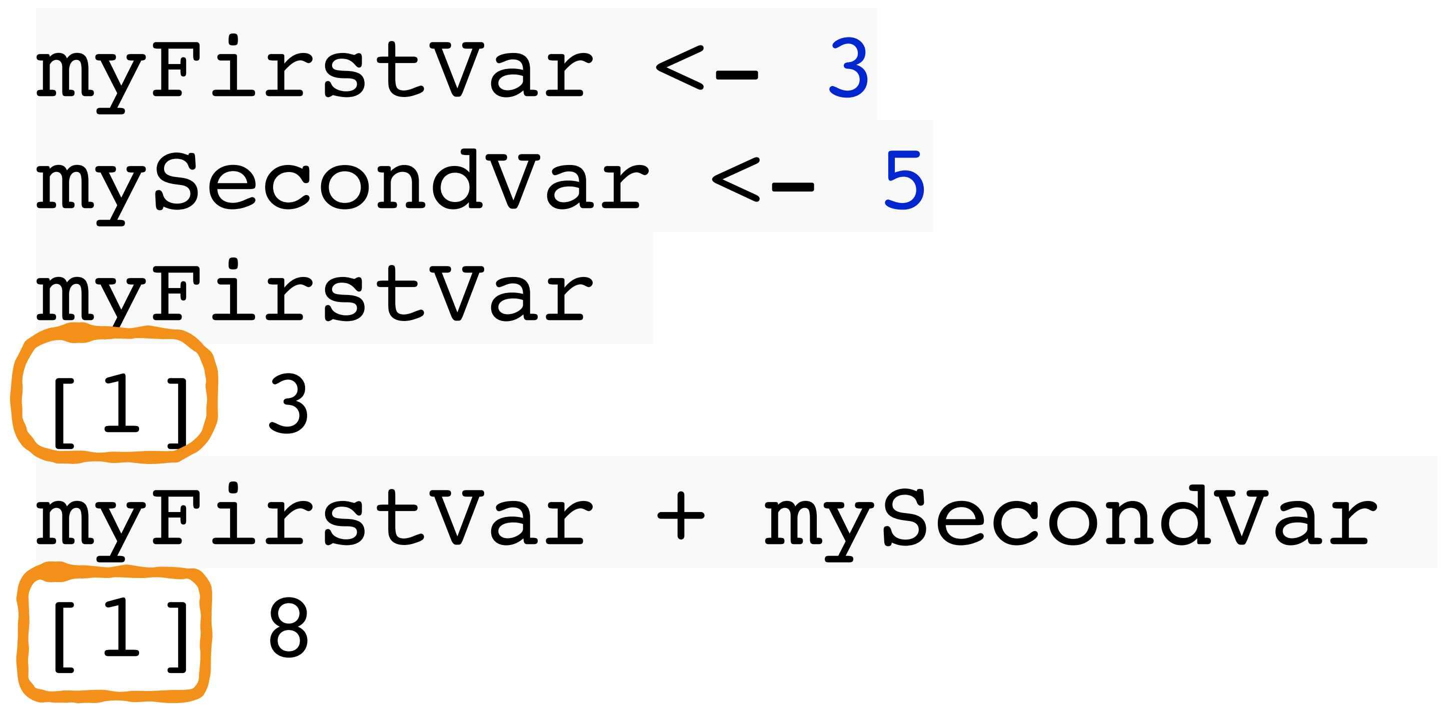


```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

WHENEVER R SEES AN EXPRESSION IT WILL EVALUATE IT I.E. COMPUTE A RESULT

IF THE RESULT IS NOT ASSIGNED TO A VARIABLE, IT WILL PRINT IT TO SCREEN

EXAMPLE 2: PRINTING AN OUTPUT



```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

EVERYTHING IN R IS BY DEFAULT TREATED AS A VECTOR (A KIND OF LIST)

EVEN IF THE RESULT IS JUST ONE THING, IT'S TREATED AS THE FIRST ELEMENT IN A VECTOR WITH 1 ELEMENT

UNLESS OTHERWISE SPECIFIED THE VECTOR IS INDEXED FROM 1 (UNLIKE IN OTHER PROGRAMMING LANGUAGES)

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5

show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same

sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT

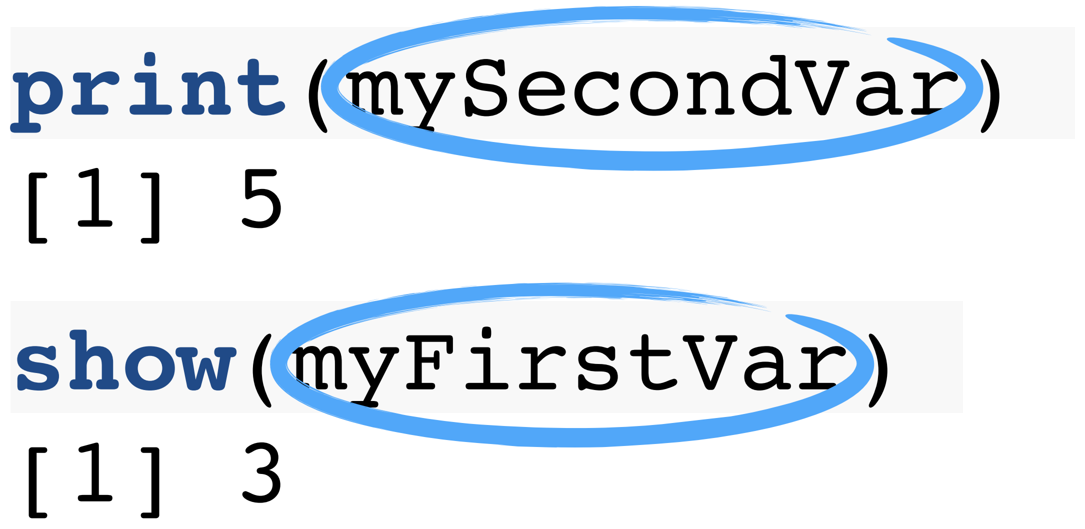
```
print(mySecondVar)
[1] 5
```

```
show(myFirstVar)
[1] 3
```

PRINT() AND SHOW() ARE VERY SIMILAR

EXAMPLE 2: PRINTING AN OUTPUT

PRINT() AND SHOW() ARE VERY SIMILAR



```
print(mySecondVar)
[1] 5
```

```
show(myFirstVar)
[1] 3
```

BOTH OF THEM PRINT A SINGLE RESULT

THAT SINGLE RESULT COULD BE A VARIABLE..

```
show(myFirstVar)
```

..OR AN EXPRESSION

```
print(myFirstVar+mySecondVar)
```

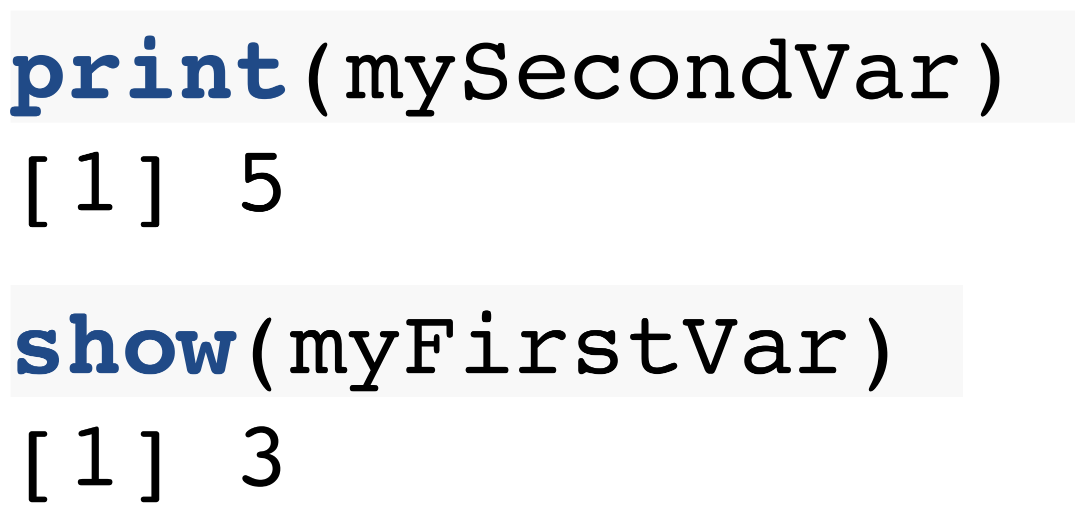
YOU CAN'T PRINT 2 OR MORE THINGS AT THE SAME TIME

```
print(myFirstVar, mySecondVar)
```

```
show(myFirstVar, "is a variable")
```

EXAMPLE 2: PRINTING AN OUTPUT

PRINT() AND SHOW() ARE VERY SIMILAR



```
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
```

BOTH OF THEM PRINT A SINGLE RESULT

YOU CAN'T PRINT 2 OR MORE THINGS AT THE SAME TIME

SHOW() IS ACTUALLY AN EXTENSION OF PRINT()

PRINT() UNDERSTANDS THE TYPE OF THE RESULT (STRING, NUMBER, VECTOR, LIST ETC) AND THEN PRINTS IT ACCORDINGLY

SHOW() CAN DISPLAY ON SCREEN EVERYTHING PRINT() DOES

IN ADDITION, IT CAN DISPLAY GRAPHS, PLOTS, TABLES ETC

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

```
print(mySecondVar)
[1] 5

show(myFirstVar)
[1] 3
```

```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same

sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

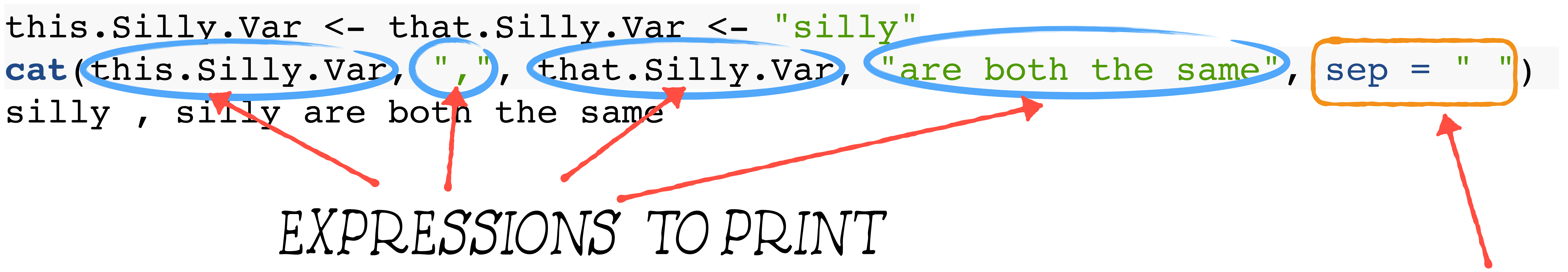
```
print(mySecondVar)
[1] 5
```

```
show(myFirstVar)
[1] 3
```

```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT



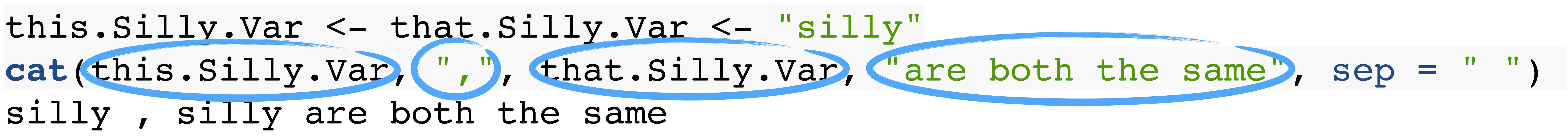
```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

EXPRESSIONS TO PRINT
(AS MANY AS YOU LIKE)

A CHARACTER TO SEPARATE
THE MULTIPLE RESULTS

USE CAT() WHEN YOU WANT TO PRINT MULTIPLE RESULTS

EXAMPLE 2: PRINTING AN OUTPUT



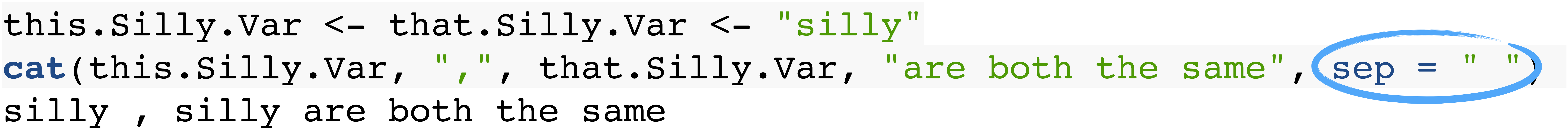
```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

USE CAT() WHEN YOU WANT TO PRINT MULTIPLE RESULTS

CAT() WILL

1. CONVERT EACH VARIABLE/EXPRESSION TO A STRING (IF THEY ARE NOT ALREADY)

EXAMPLE 2: PRINTING AN OUTPUT



```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

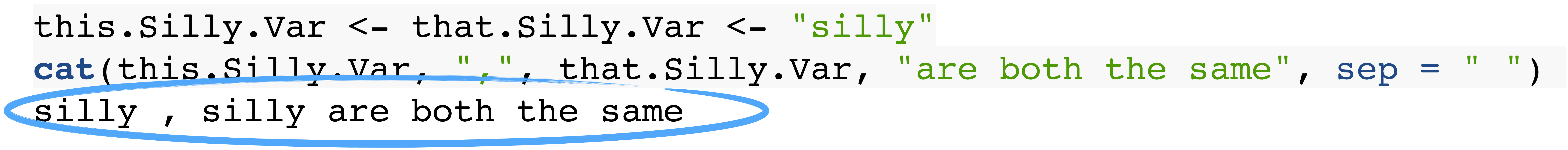
USE CAT() WHEN YOU WANT TO PRINT MULTIPLE RESULTS

CAT() WILL

1. CONVERT EACH VARIABLE/EXPRESSION TO A STRING (IF THEY ARE NOT ALREADY)

2. CONCATENATE ALL THE STRINGS USING THE SPECIFIED DELIMITER

EXAMPLE 2: PRINTING AN OUTPUT



```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

USE CAT() WHEN YOU WANT TO PRINT MULTIPLE RESULTS

CAT() WILL

1. CONVERT EACH VARIABLE/EXPRESSION TO A STRING (IF THEY ARE NOT ALREADY)

2. CONCATENATE ALL THE STRINGS USING THE SPECIFIED DELIMITER

3. PRINT THE CONCATENATED RESULT TO SCREEN

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
```

```
print(mySecondVar)
[1] 5
```

```
show(myFirstVar)
[1] 3
```

```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
```

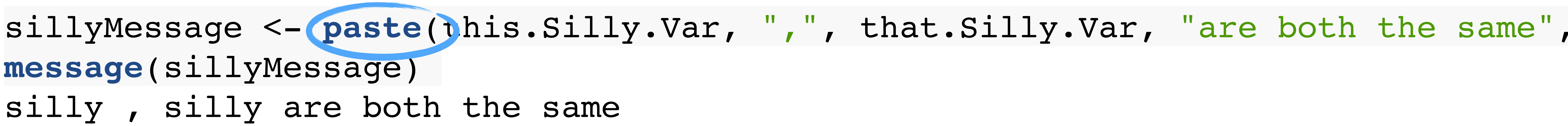
```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8

print(mySecondVar)
[1] 5

show(myFirstVar)
[1] 3

this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same

sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 2: PRINTING AN OUTPUT



```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same",
message(sillyMessage)
silly , silly are both the same
```

PASTE() IS VERY SIMILAR TO CAT()

BOTH WILL

1. CONVERT EACH VARIABLE/EXPRESSION TO A STRING (IF THEY ARE NOT ALREADY)
2. CONCATENATE ALL THE STRINGS USING THE SPECIFIED DELIMITER

USE CAT() TO PRINT THE RESULTING STRING TO SCREEN

USE PASTE() IF YOU WANT TO STORE THE RESULTING STRING FOR LATER

EXAMPLE 2: PRINTING AN OUTPUT

```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same",
message(sillyMessage)
silly , silly are both the same
```

PASTE() IS VERY SIMILAR TO CAT()

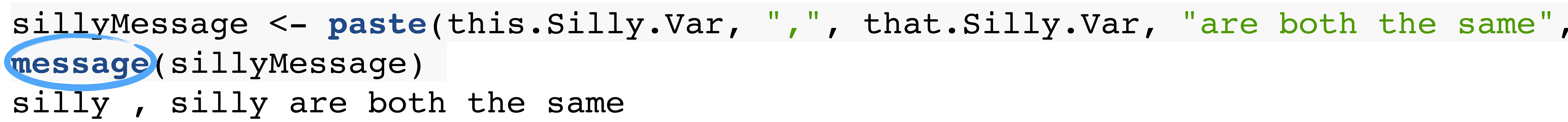
USE CAT() TO PRINT THE RESULTING STRING TO SCREEN

USE PASTE() IF YOU WANT TO STORE THE RESULTING STRING FOR LATER

THE RESULT OF CAT() CANNOT BE STORED IN A VARIABLE

```
sillyMessage <- cat(this.Silly.Var, that.Silly.Var,sep = “ “)

silly , silly are both the same

print(sillyMessage)

Error in print(sillyMessage) : object 'sillyMessage' not found
```

EXAMPLE 2: PRINTING AN OUTPUT



```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same",
message(sillyMessage)
silly , silly are both the same
```

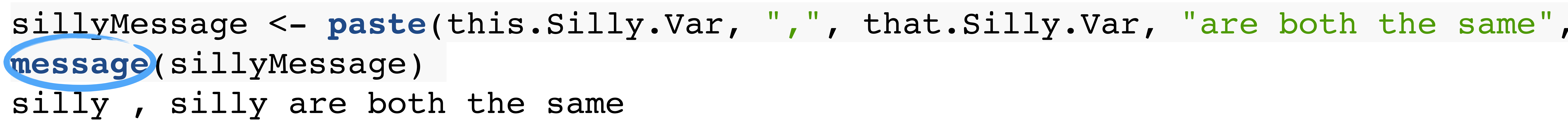
PASTE() IS VERY SIMILAR TO CAT()

USE CAT() TO PRINT THE RESULTING STRING TO SCREEN

USE PASTE() IF YOU WANT TO STORE THE RESULTING STRING FOR LATER

MESSAGE() CAN BE USED TO PRINT A SINGLE RESULT TO THE SCREEN

EXAMPLE 2: PRINTING AN OUTPUT



```
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same",
message(sillyMessage)
silly , silly are both the same
```

PASTE() IS VERY SIMILAR TO CAT()

USE CAT() TO PRINT THE RESULTING STRING TO SCREEN

USE PASTE() IF YOU WANT TO STORE THE RESULTING STRING FOR LATER

MESSAGE() CAN BE USED TO PRINT A SINGLE RESULT TO THE SCREEN

MESSAGE() WILL CONVERT THE OUTPUT TO A STRING AND ADD A NEWLINE (\n) AT THE END OF THE STRING

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS
TO PRINT AN OUTPUT IN R

```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
message(sillyMessage)
silly , silly are both the same
```

PRINT() AND SHOW() WILL TREAT THE
RESULT AS A VECTOR WITH 1 ELEMENT

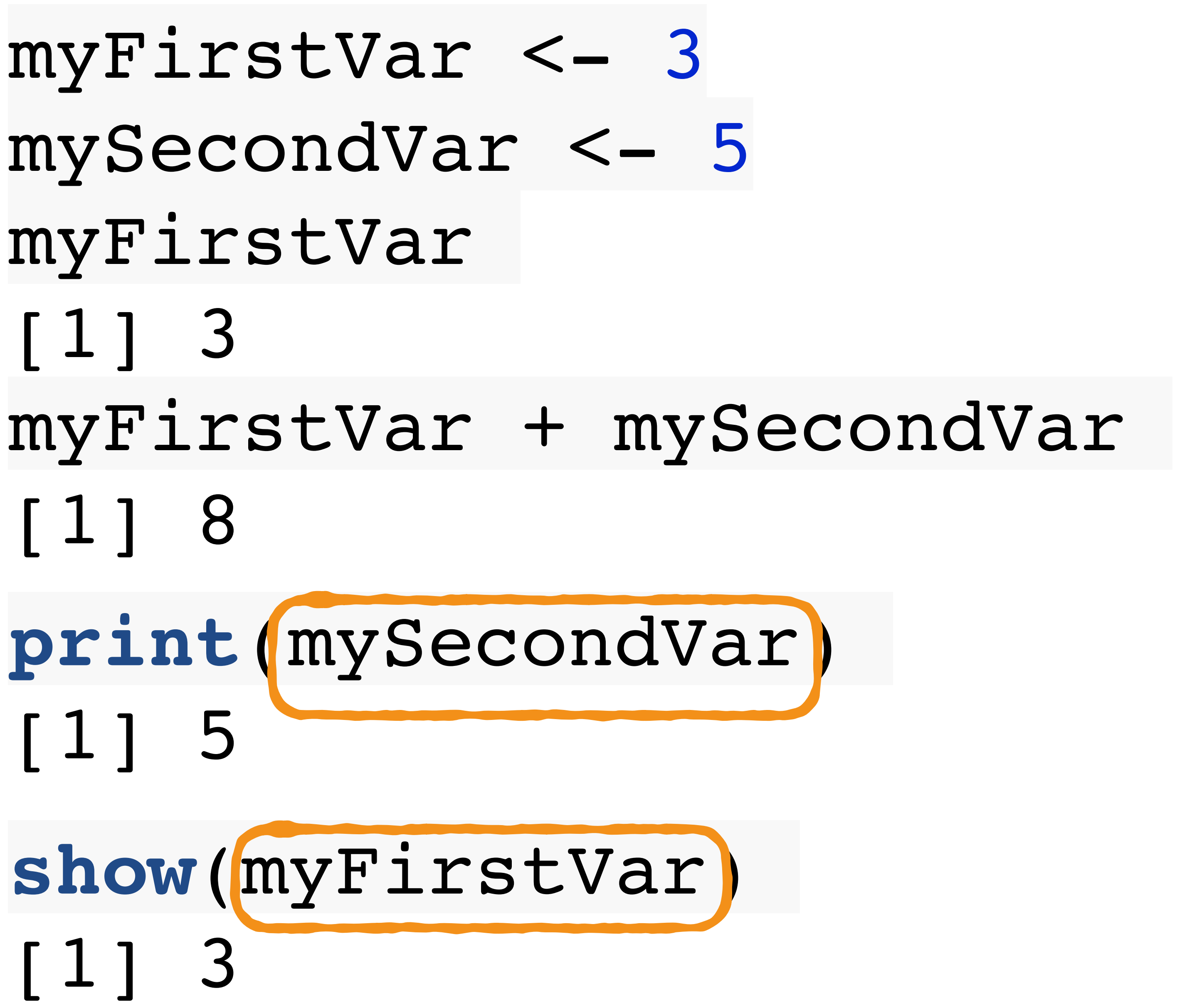
CAT() AND MESSAGE() WILL
TREAT THE RESULT AS A STRING

EXAMPLE 2: PRINTING AN OUTPUT

HERE ARE A FEW DIFFERENT WAYS TO PRINT AN OUTPUT IN R

PRINT(), SHOW() AND MESSAGE() WILL TAKE ONLY 1 INPUT

CAT() CAN TAKE MULTIPLE INPUTS



```
myFirstVar <- 3
mySecondVar <- 5
myFirstVar
[1] 3
myFirstVar + mySecondVar
[1] 8
print(mySecondVar)
[1] 5
show(myFirstVar)
[1] 3
```

```
this.Silly.Var <- that.Silly.Var <- "silly"
cat(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " ")
silly , silly are both the same
sillyMessage <- paste(this.Silly.Var, ",", that.Silly.Var, "are both the same", sep = " "
message(sillyMessage)
silly , silly are both the same
```

EXAMPLE 3: NUMBERS IN R

EXAMPLE 3: NUMBERS IN R

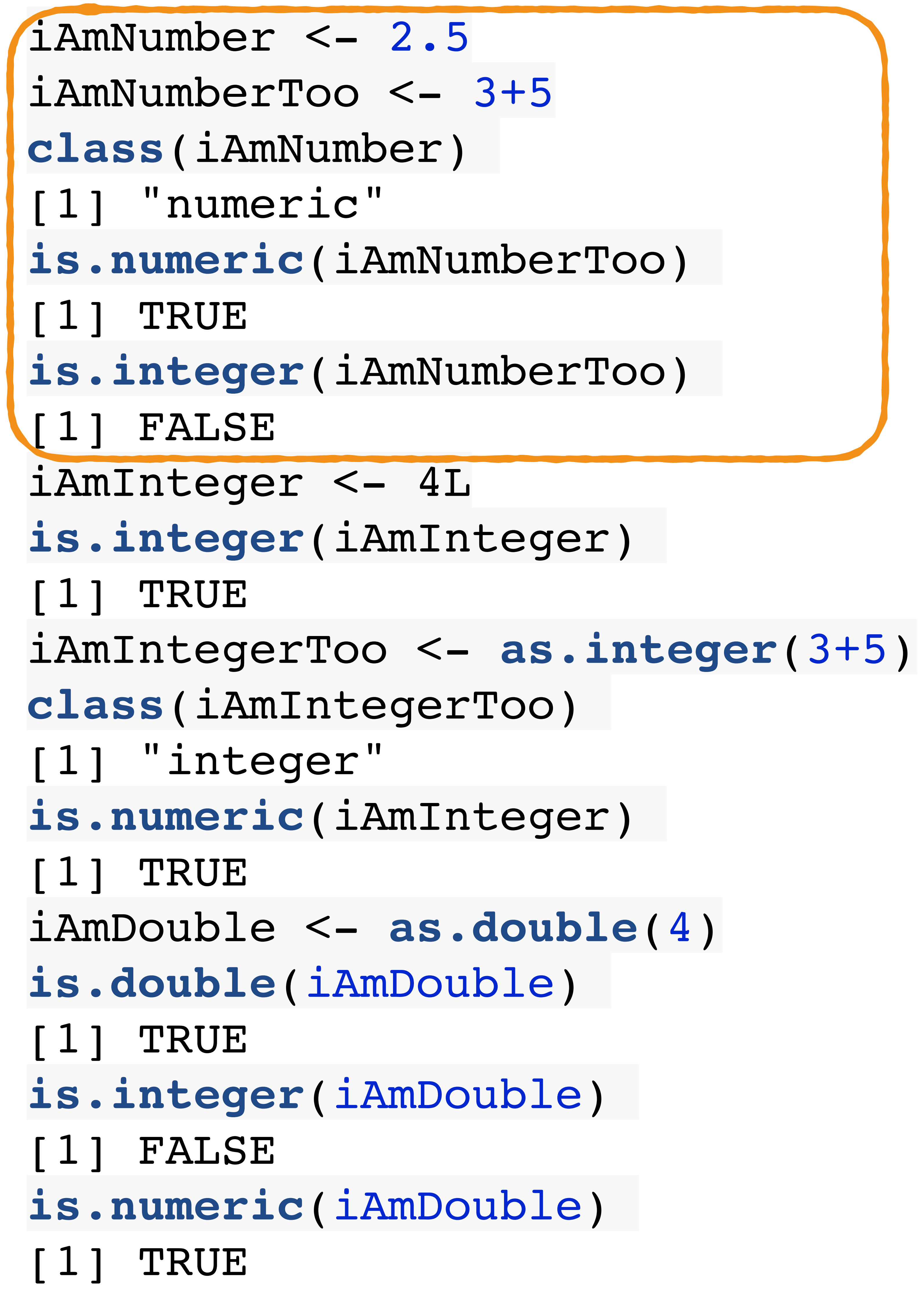
VARIABLES WHICH ARE NUMBERS ARE OF THE DATATYPE

NUMERIC

NUMERIC COVERS ALL KINDS OF NUMBERS - INTEGERS, FLOATS/DOUBLES ETC

YOU CAN EXPLICITLY MAKE A VARIABLE INTEGER OR DOUBLE - BUT IT WOULD STILL BE NUMERIC TOO

EXAMPLE 3: NUMBERS IN R



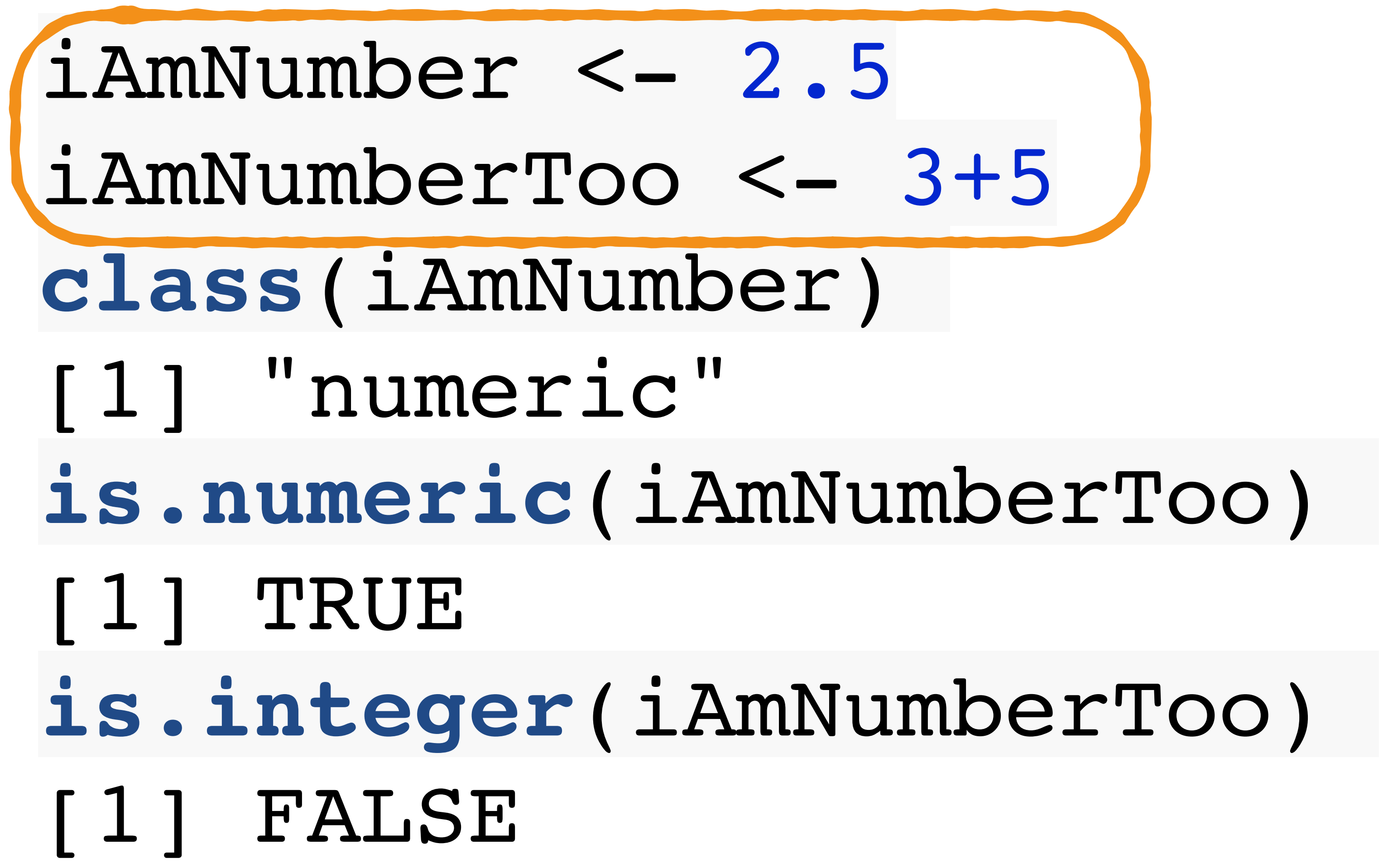
```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

VARIABLES WHICH ARE NUMBERS
ARE OF THE DATATYPE NUMERIC

NUMERIC COVERS ALL KINDS OF NUMBERS -
INTEGERS, FLOATS/DOUBLES ETC

YOU CAN EXPLICITLY MAKE A VARIABLE INTEGER
OR DOUBLE - BUT IT WOULD STILL BE NUMERIC TOO

EXAMPLE 3: NUMBERS IN R



```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
```

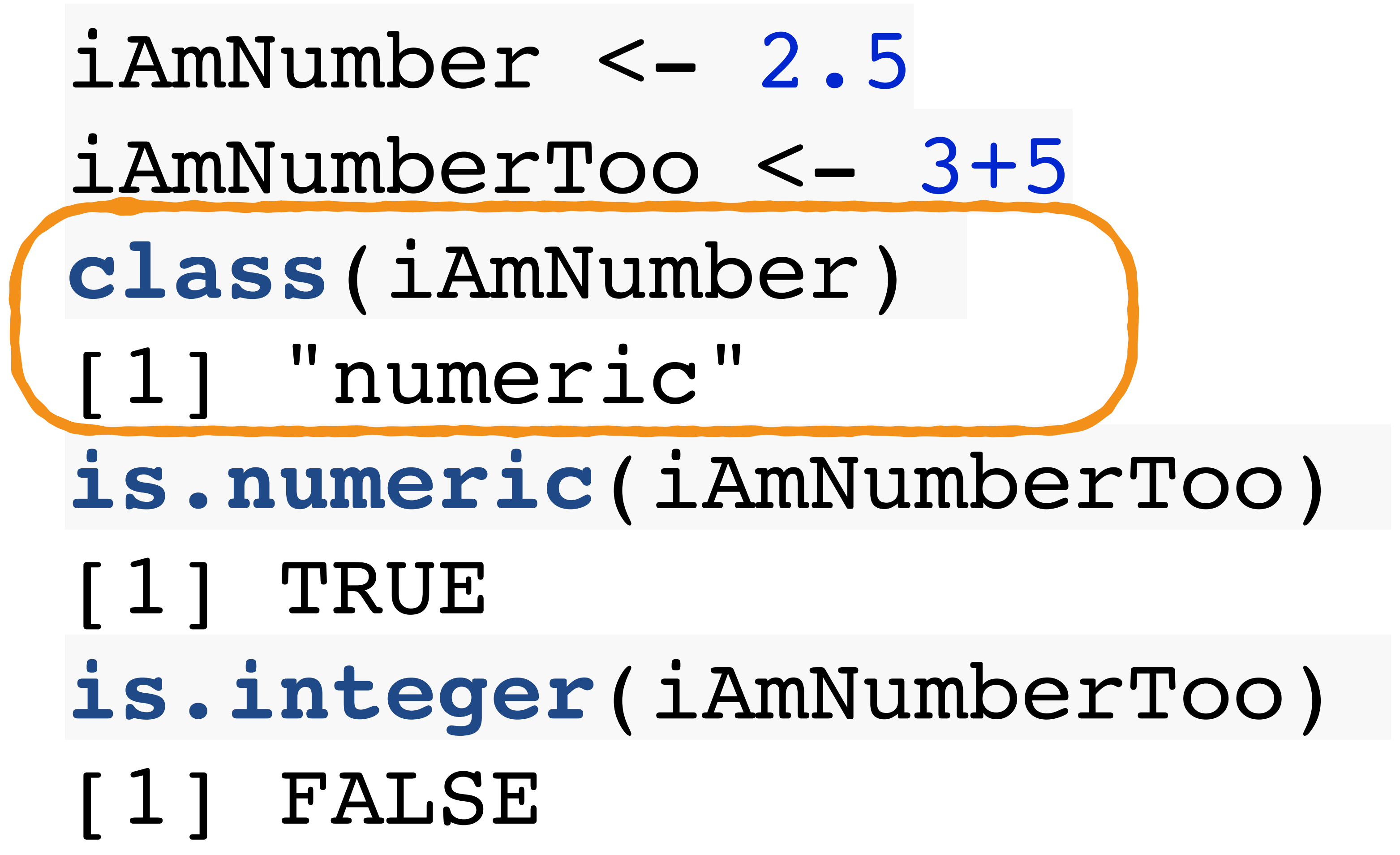
WHENEVER YOU ASSIGN A VALUE TO A VARIABLE, R WILL AUTOMATICALLY IDENTIFY THE DATATYPE

THE DATATYPE OF A VARIABLE NEED NOT BE DECLARED BEFOREHAND (LIKE YOU WOULD IN C/C++/JAVA)

EXAMPLE 3: NUMBERS IN R

WHENEVER YOU ASSIGN A VALUE TO A VARIABLE, R WILL AUTOMATICALLY IDENTIFY THE DATATYPE

THE DATATYPE OF A VARIABLE NEED NOT BE DECLARED BEFOREHAND (LIKE YOU WOULD IN C/C++/JAVA)



```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
```

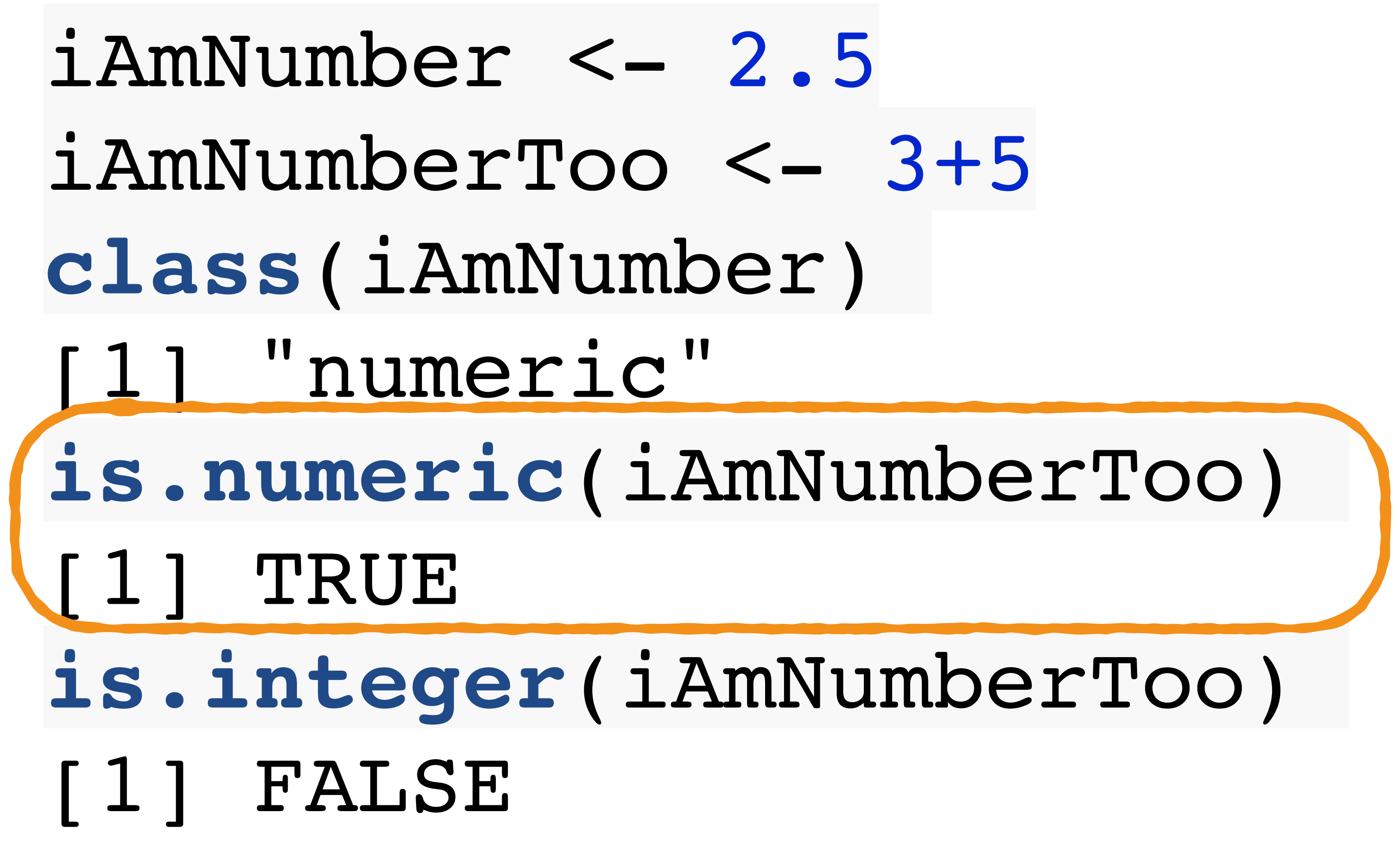
THE CLASS() FUNCTION WILL PRINT THE DATATYPE OF A VARIABLE

WHEN A NUMBER IS ASSIGNED TO A VARIABLE, IT AUTOMATICALLY BECOMES OF TYPE "NUMERIC"

EXAMPLE 3: NUMBERS IN R

WHENEVER YOU ASSIGN A VALUE TO A VARIABLE, R WILL AUTOMATICALLY IDENTIFY THE DATATYPE

THE DATATYPE OF A VARIABLE NEED NOT BE DECLARED BEFOREHAND (LIKE YOU WOULD IN C/C++/JAVA)



```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
```

YOU CAN ALSO CHECK WHETHER A VARIABLE IS OF A CERTAIN TYPE

WHEN A NUMBER IS ASSIGNED TO A VARIABLE, IT AUTOMATICALLY BECOMES OF TYPE “NUMERIC”

EXAMPLE 3: NUMBERS IN R

WHENEVER YOU ASSIGN A VALUE TO A VARIABLE, R WILL AUTOMATICALLY IDENTIFY THE DATATYPE

THE DATATYPE OF A VARIABLE NEED NOT BE DECLARED BEFOREHAND (LIKE YOU WOULD IN C/C++/JAVA)

WHEN A NUMBER IS ASSIGNED TO A VARIABLE, IT AUTOMATICALLY BECOMES OF TYPE "NUMERIC"

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
```

INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 3: NUMBERS IN R

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 3: NUMBERS IN R

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 3: NUMBERS IN R

```
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
```

IF YOU APPEND L TO THE NUMBER, R WILL CONSIDER THAT IT'S OF DATATYPE INTEGER

EXAMPLE 3: NUMBERS IN R

IF YOU APPEND L TO THE NUMBER, R WILL CONSIDER THAT IT'S OF DATATYPE INTEGER

```
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
```

AS.INTEGER() WILL CONVERT A NUMBER TO AN INTEGER

EXAMPLE 3: NUMBERS IN R

IF YOU APPEND L TO THE NUMBER, R WILL CONSIDER THAT IT'S OF DATATYPE INTEGER

AS.INTEGER() WILL CONVERT A NUMBER TO AN INTEGER

```
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
```

AN INTEGER IS ALSO A NUMERIC

EXAMPLE 3: NUMBERS IN R

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

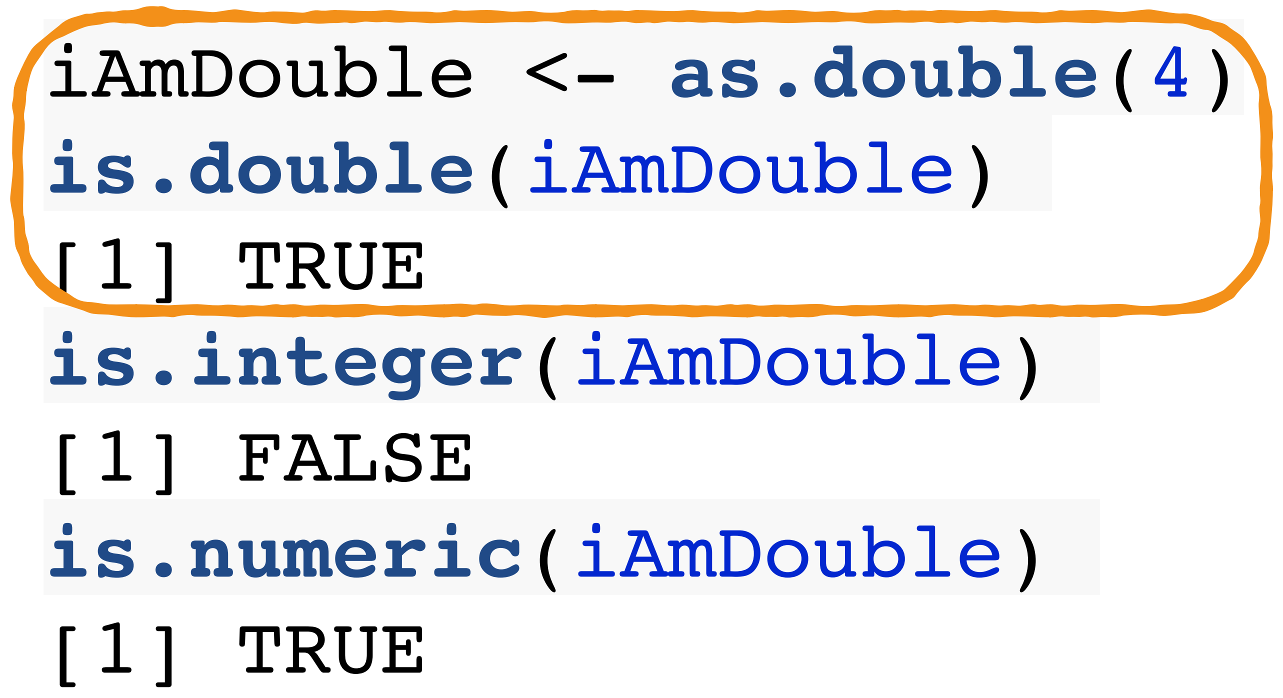
INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 3: NUMBERS IN R

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 3: NUMBERS IN R

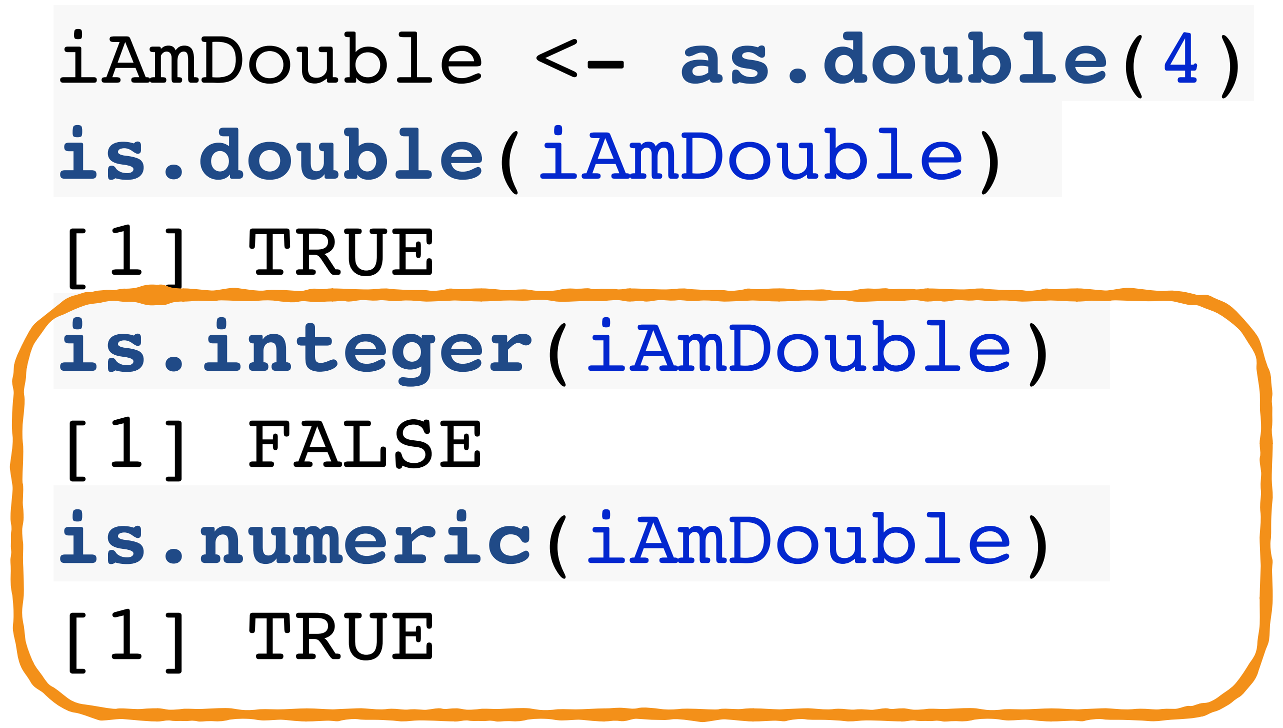


```
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

AS.DOUBLE() WILL CONVERT A NUMBER TO DOUBLE

EXAMPLE 3: NUMBERS IN R

AS.DOUBLE() WILL CONVERT A NUMBER TO DOUBLE



```
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

A DOUBLE IS NOT AN INTEGER

BUT A DOUBLE IS A NUMERIC

EXAMPLE 3: NUMBERS IN R

```
iAmNumber <- 2.5
iAmNumberToo <- 3+5
class(iAmNumber)
[1] "numeric"
is.numeric(iAmNumberToo)
[1] TRUE
is.integer(iAmNumberToo)
[1] FALSE
iAmInteger <- 4L
is.integer(iAmInteger)
[1] TRUE
iAmIntegerToo <- as.integer(3+5)
class(iAmIntegerToo)
[1] "integer"
is.numeric(iAmInteger)
[1] TRUE
iAmDouble <- as.double(4)
is.double(iAmDouble)
[1] TRUE
is.integer(iAmDouble)
[1] FALSE
is.numeric(iAmDouble)
[1] TRUE
```

INTEGER AND DOUBLE ARE ALSO AVAILABLE AS DATATYPES, BUT THESE HAVE TO BE EXPLICITLY SPECIFIED

EXAMPLE 4: CHARACTERS AND DATES

EXAMPLE 4: CHARACTERS AND DATES

VARIABLES WHICH ARE STRINGS ARE OF THE DATATYPE

CHARACTER

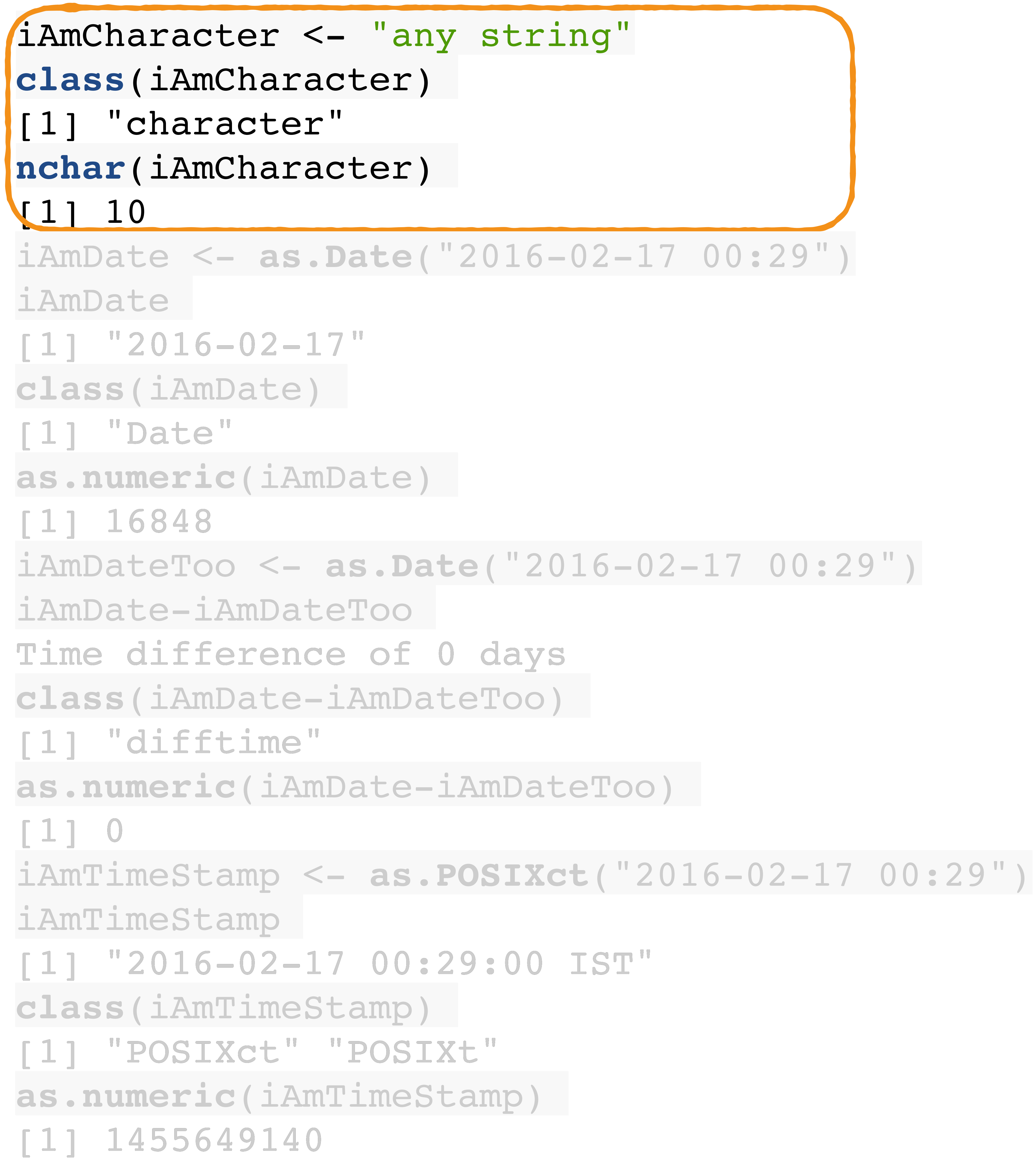
THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

POSIXCT

EXAMPLE 4: CHARACTERS AND DATES



```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-17 00:29")
iAmDate-iAmDateToo
Time difference of 0 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 0
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

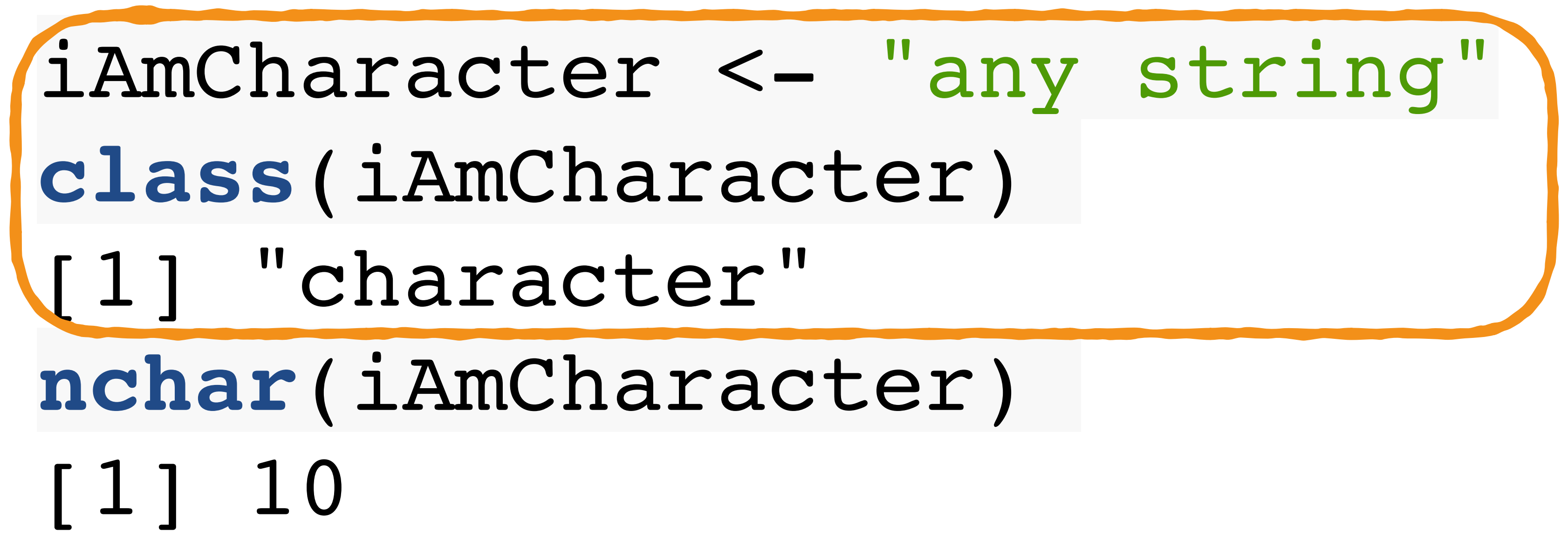
THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

POSIXCT

EXAMPLE 4: CHARACTERS AND DATES

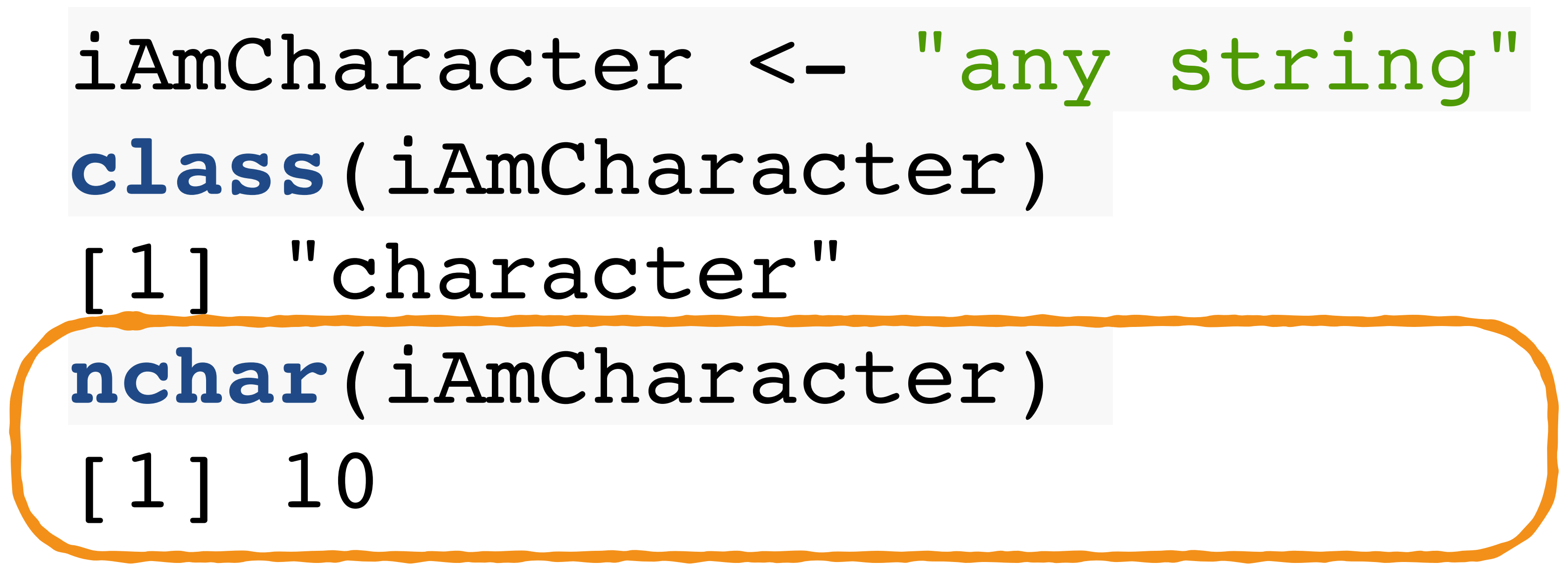


```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
```

ALL STRINGS HAVE THE DATATYPE “CHARACTER”

EXAMPLE 4: CHARACTERS AND DATES

ALL STRINGS HAVE THE DATATYPE "CHARACTER"



```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
```

NCHAR() WILL PRINT THE LENGTH OF A STRING

I.E. THE NUMBER OF CHARACTERS IN THE STRING

EXAMPLE 4: CHARACTERS AND DATES

```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-17 00:29")
iAmDate-iAmDateToo
Time difference of 0 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 0
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

THERE IS A SPECIAL DATATYPE FOR DATES
DATE
AND FOR TIMESTAMPS
POSIXCT

EXAMPLE 4: CHARACTERS AND DATES

```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

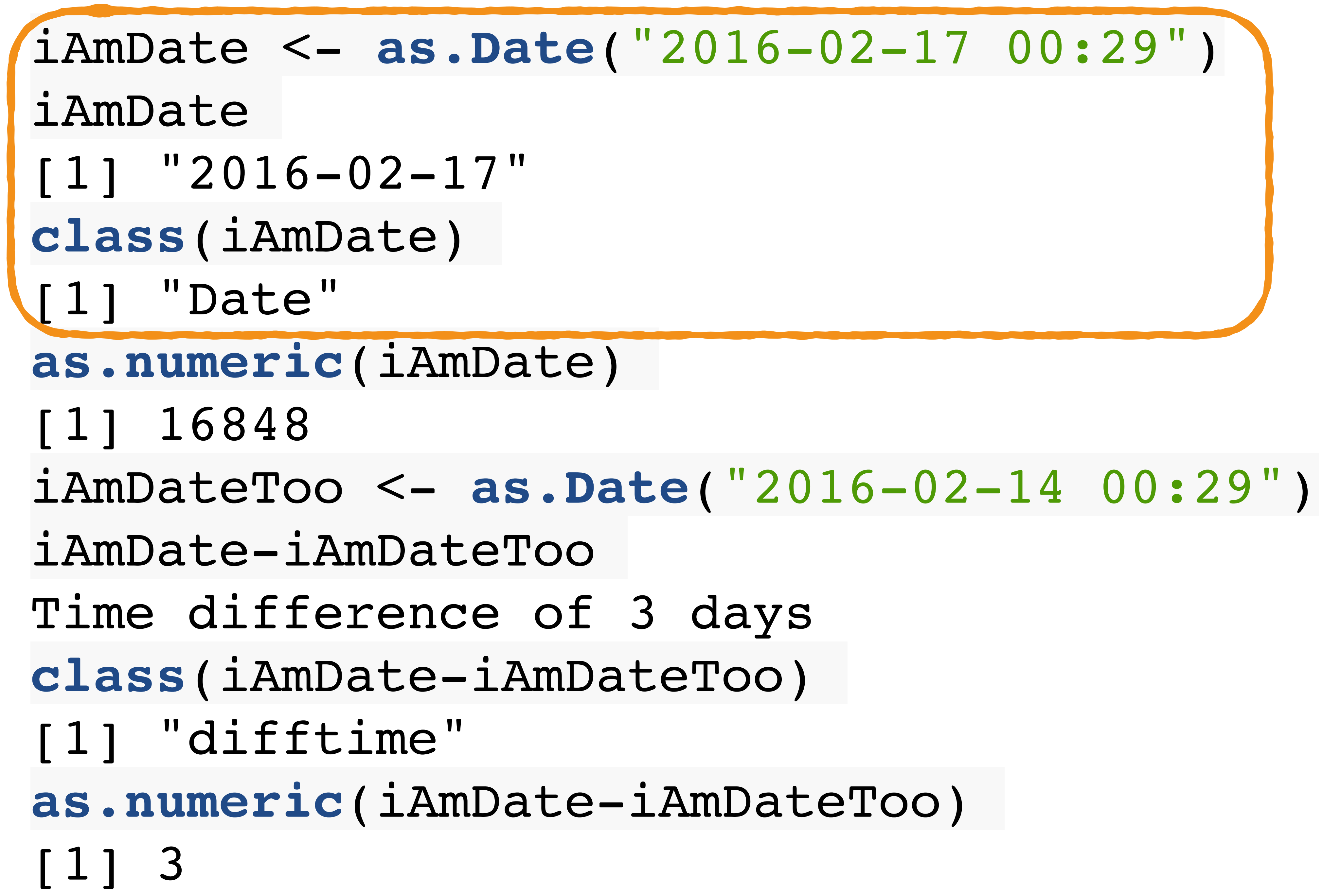
THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

POSIXCT

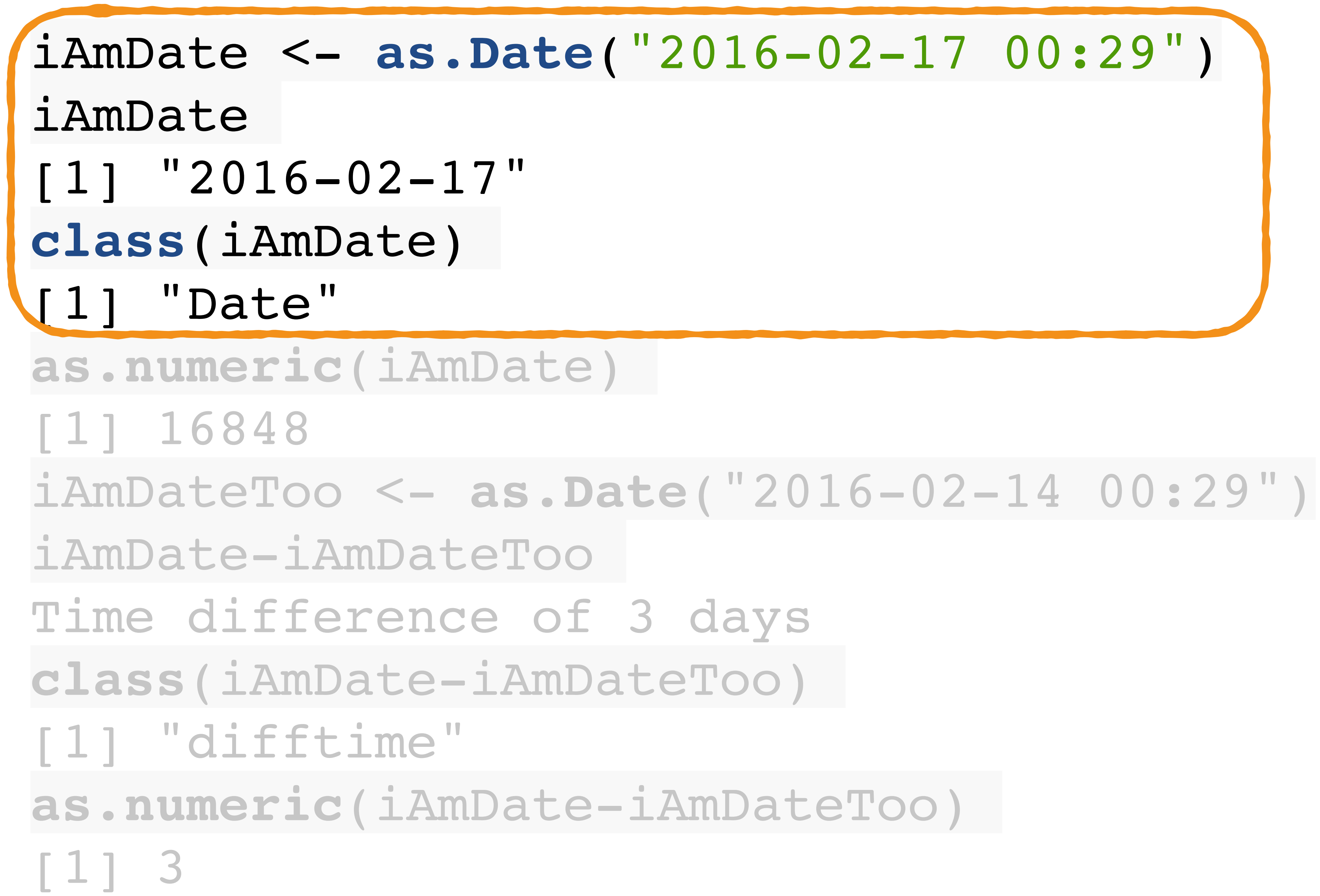
EXAMPLE 4: CHARACTERS AND DATES



```
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
```

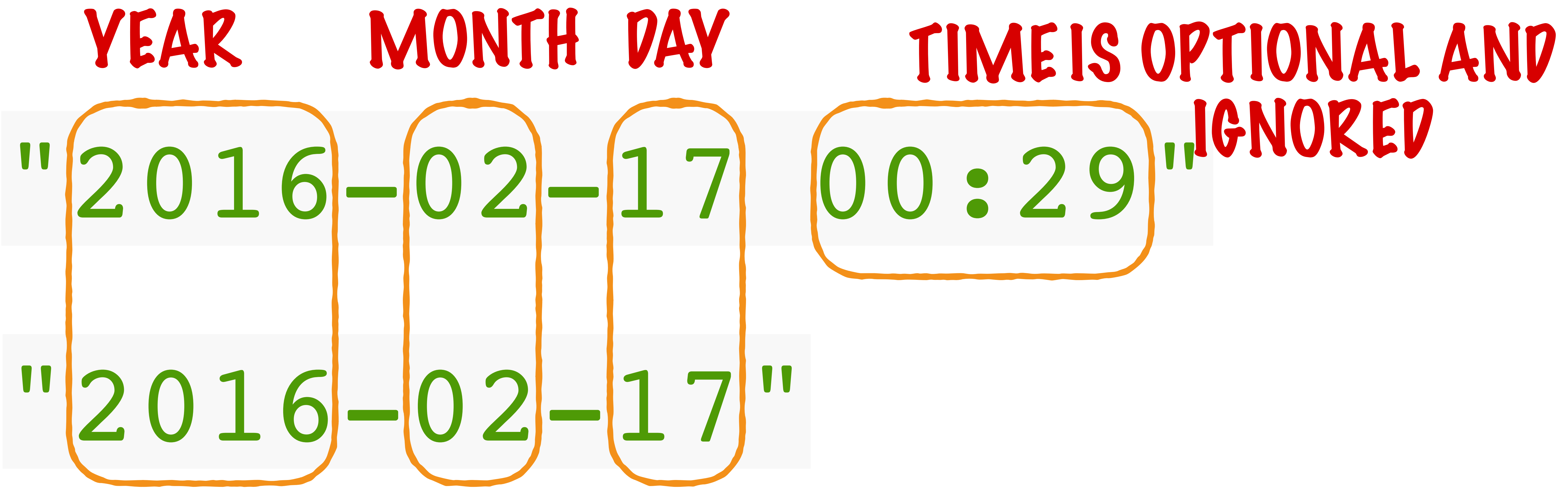
A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "DATE"

EXAMPLE 4: CHARACTERS AND DATES



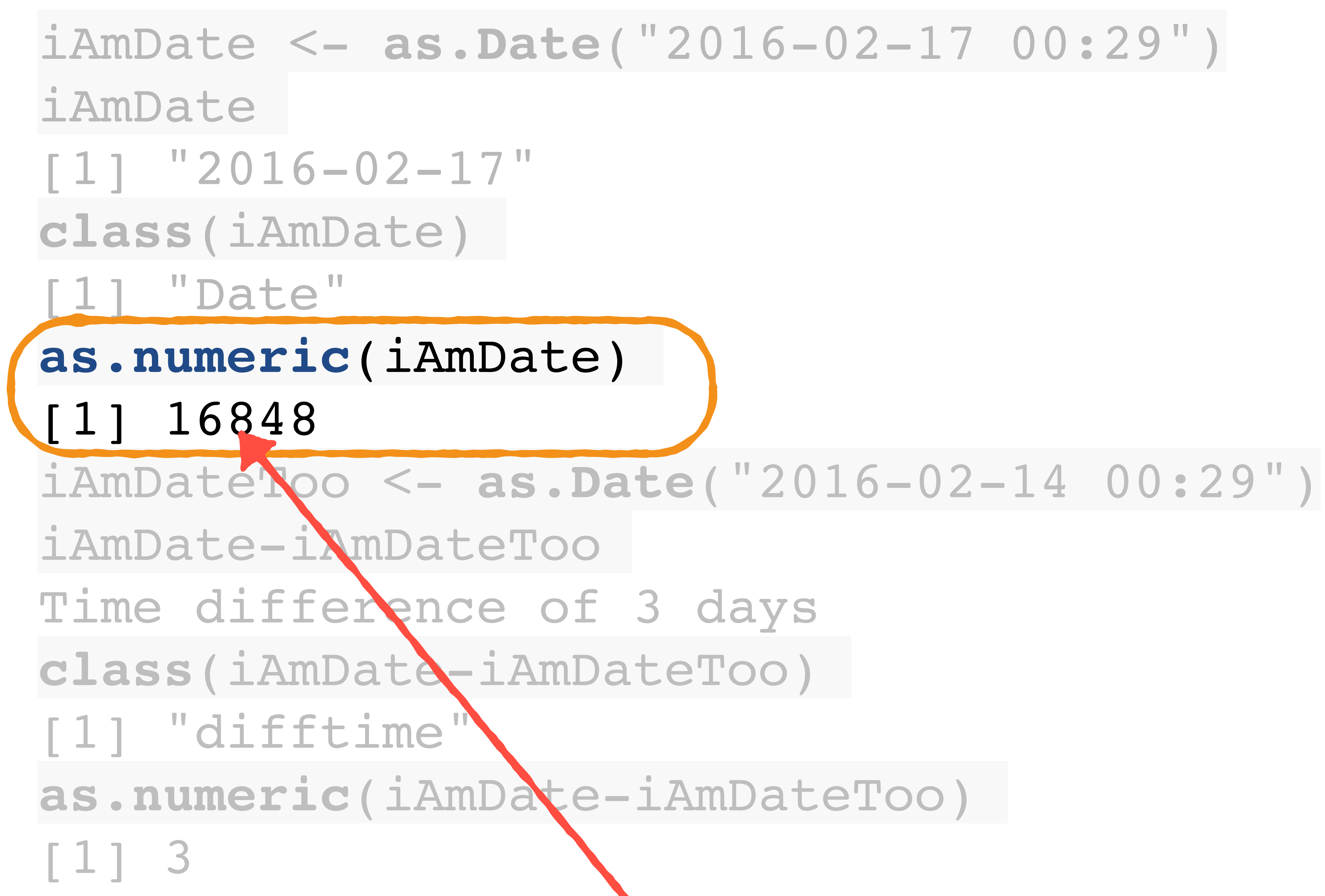
```
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
```

A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "DATE"



EXAMPLE 4: CHARACTERS AND DATES

A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "DATE"



```
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
```

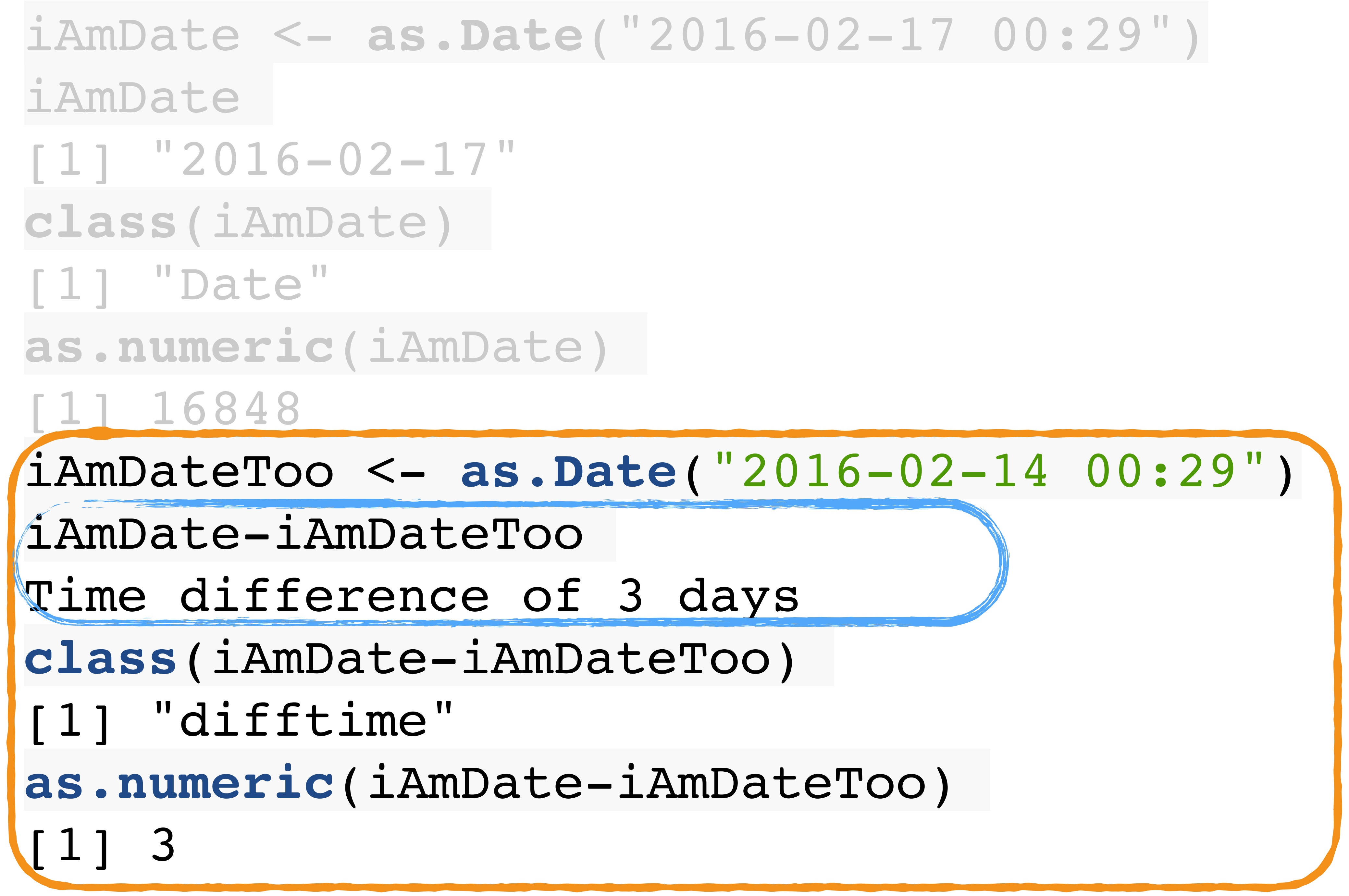
ANY "DATE" CAN BE CONVERTED TO A NUMBER

WHICH IS THE NUMBER OF DAYS SINCE JAN 1, 1970

EXAMPLE 4: CHARACTERS AND DATES

A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "DATE"

ANY "DATE" CAN BE CONVERTED TO A NUMBER WHICH IS THE NUMBER OF DAYS SINCE JAN 1, 1970



```
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
```

YOU CAN FIND THE DIFFERENCE BETWEEN 2 DATES

THE DIFFERENCE WILL BE STORED IN A SPECIAL OBJECT OF TYPE "DIFFTIME"

YOU CAN CONVERT THE DIFFERENCE TO A NUMBER IF YOU NEED TO

EXAMPLE 4: CHARACTERS AND DATES

```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-14 00:29")
iAmDate-iAmDateToo
Time difference of 3 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 3
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

POSIXCT

EXAMPLE 4: CHARACTERS AND DATES

```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-17 00:29")
iAmDate-iAmDateToo
Time difference of 0 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 0
```

```
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

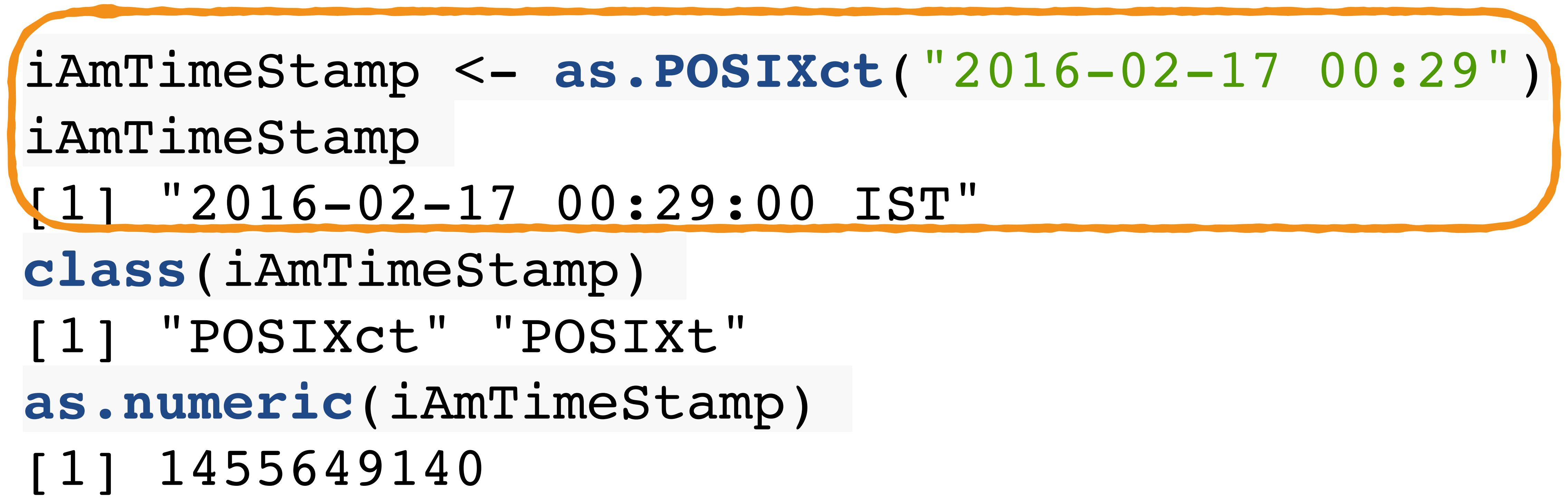
THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

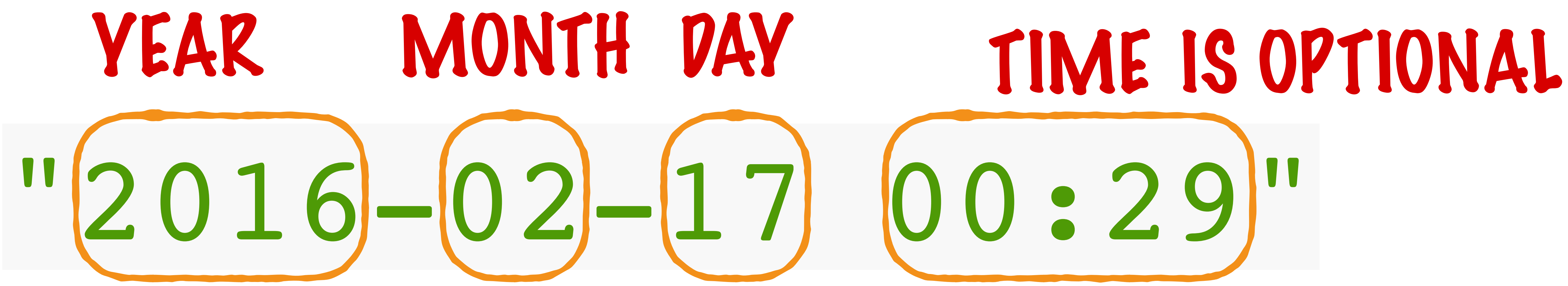
POSIXCT

EXAMPLE 4: CHARACTERS AND DATES



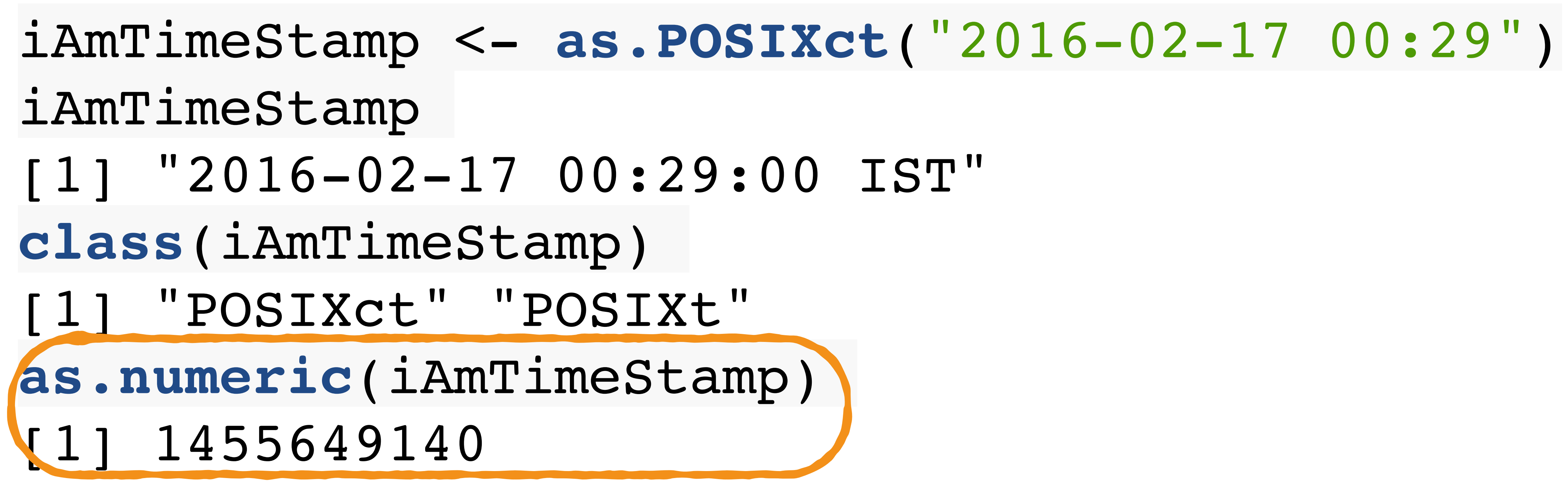
```
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "POSIXCT" WHICH IS A TIMESTAMP



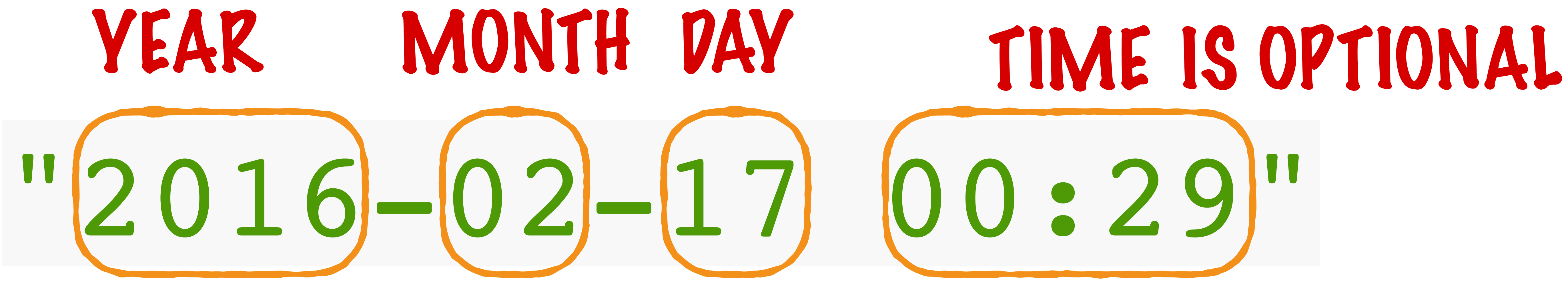
EXAMPLE 4: CHARACTERS AND DATES

A STRING IN A PARTICULAR FORMAT CAN BE CAST AS DATATYPE "POSIXCT" WHICH IS A TIMESTAMP



```
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

A TIMESTAMP CAN BE CONVERTED TO A NUMBER WHICH IS THE NUMBER OF SECONDS SINCE JAN 1, 1970



EXAMPLE 4: CHARACTERS AND DATES

```
iAmCharacter <- "any string"
class(iAmCharacter)
[1] "character"
nchar(iAmCharacter)
[1] 10
iAmDate <- as.Date("2016-02-17 00:29")
iAmDate
[1] "2016-02-17"
class(iAmDate)
[1] "Date"
as.numeric(iAmDate)
[1] 16848
iAmDateToo <- as.Date("2016-02-17 00:29")
iAmDate-iAmDateToo
Time difference of 0 days
class(iAmDate-iAmDateToo)
[1] "difftime"
as.numeric(iAmDate-iAmDateToo)
[1] 0
iAmTimeStamp <- as.POSIXct("2016-02-17 00:29")
iAmTimeStamp
[1] "2016-02-17 00:29:00 IST"
class(iAmTimeStamp)
[1] "POSIXct" "POSIXt"
as.numeric(iAmTimeStamp)
[1] 1455649140
```

VARIABLES WHICH ARE STRINGS
ARE OF THE DATATYPE CHARACTER

THERE IS A SPECIAL DATATYPE FOR DATES

DATE

AND FOR TIMESTAMPS

POSIXCT

EXAMPLE 5: LOGICALS

EXAMPLE 5: LOGICALS

VARIABLES WITH DATATYPE

LOGICAL

CAN TAKE ONLY 2 VALUES

TRUE OR FALSE

EXAMPLE 5: LOGICALS

```
iAmTrue <- TRUE
class(iAmTrue)
[1] "logical"
iAmFalse <- FALSE
class(iAmFalse)
[1] "logical"
iAmNumber <- 5
iAmFalse * iAmNumber
[1] 0
iAmTrue * iAmNumber
[1] 5
iAmLogical <- 2 == 3
iAmLogical
[1] FALSE
iAmLogicalToo <- 2 != 3
iAmLogicalToo
[1] TRUE
iCompareCharacters <- "Red" > "Blue"
iCompareCharacters
[1] FALSE
```

VARIABLES WITH DATATYPE

LOGICAL

CAN TAKE ONLY 2 VALUES

TRUE

OR

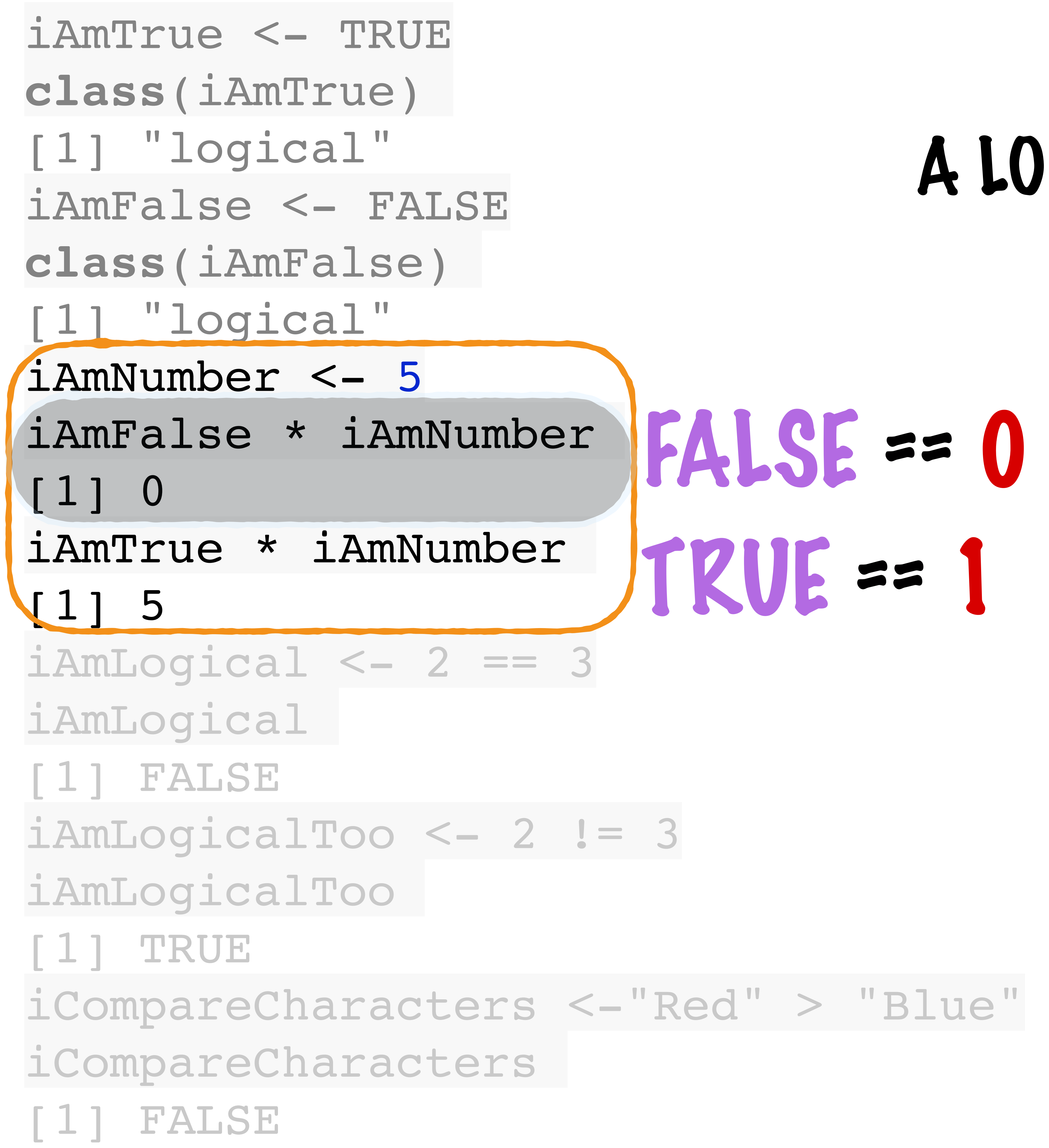
FALSE

EXAMPLE 5: LOGICALS

```
iAmTrue <- TRUE
class(iAmTrue)
[1] "logical"
iAmFalse <- FALSE
class(iAmFalse)
[1] "logical"
iAmNumber <- 5
iAmFalse * iAmNumber
[1] 0
iAmTrue * iAmNumber
[1] 5
iAmLogical <- 2 == 3
iAmLogical
[1] FALSE
iAmLogicalToo <- 2 != 3
iAmLogicalToo
[1] TRUE
iCompareCharacters <- "Red" > "Blue"
iCompareCharacters
[1] FALSE
```

A LOGICAL VARIABLE CAN HAVE THE VALUE **TRUE** OR **FALSE**

EXAMPLE 5: LOGICALS



```
iAmTrue <- TRUE
class(iAmTrue)
[1] "logical"
iAmFalse <- FALSE
class(iAmFalse)
[1] "logical"
iAmNumber <- 5
iAmFalse * iAmNumber
[1] 0
iAmTrue * iAmNumber
[1] 5
iAmLogical <- 2 == 3
iAmLogical
[1] FALSE
iAmLogicalToo <- 2 != 3
iAmLogicalToo
[1] TRUE
iCompareCharacters <-"Red" > "Blue"
iCompareCharacters
[1] FALSE
```

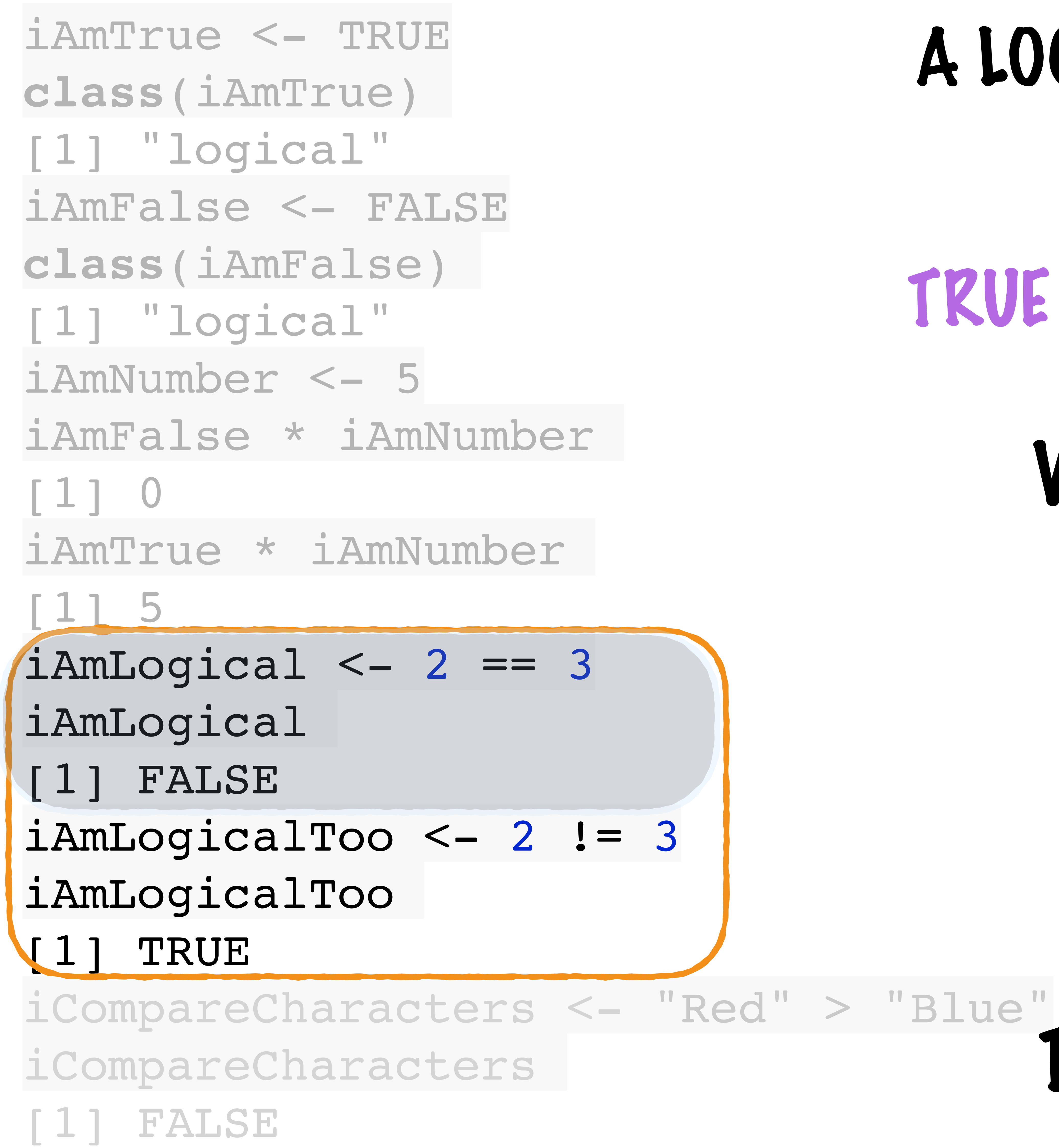
FALSE == 0

TRUE == 1

A LOGICAL VARIABLE CAN HAVE THE VALUE TRUE OR FALSE

TRUE AND FALSE ACT LIKE THEY ARE NUMBERS

EXAMPLE 5: LOGICALS



```
iAmTrue <- TRUE
class(iAmTrue)
[1] "logical"
iAmFalse <- FALSE
class(iAmFalse)
[1] "logical"
iAmNumber <- 5
iAmFalse * iAmNumber
[1] 0
iAmTrue * iAmNumber
[1] 5
iAmLogical <- 2 == 3
iAmLogical
[1] FALSE
iAmLogicalToo <- 2 != 3
iAmLogicalToo
[1] TRUE
iCompareCharacters <- "Red" > "Blue"
iCompareCharacters
[1] FALSE
```

A LOGICAL VARIABLE CAN HAVE THE VALUE
TRUE OR FALSE

TRUE AND FALSE ACT LIKE THEY ARE
NUMBERS FALSE == 0 TRUE == 1

WHEN YOU COMPARE TWO NUMBERS,
THE RESULT IS A LOGICAL

```
2 == 3
```

TESTS WHETHER 2 IS EQUAL TO 3

```
2 != 3
```

TESTS WHETHER 2 IS NOT EQUAL TO 3

EXAMPLE 5: LOGICALS

```
iAmTrue <- TRUE
class(iAmTrue)
[1] "logical"
iAmFalse <- FALSE
class(iAmFalse)
[1] "logical"
iAmNumber <- 5
iAmFalse * iAmNumber
[1] 0
iAmTrue * iAmNumber
[1] 5
iAmLogical <- 2 == 3
iAmLogical
[1] FALSE
iAmLogicalToo <- 2 != 3
iAmLogicalToo
[1] TRUE
iCompareCharacters <- "Red" > "Blue"
iCompareCharacters
[1] FALSE
```

A LOGICAL VARIABLE CAN HAVE THE VALUE TRUE OR FALSE

TRUE AND FALSE ACT LIKE THEY ARE NUMBERS FALSE == 0 TRUE == 1

WHEN YOU COMPARE TWO NUMBERS, THE RESULT IS A LOGICAL

WHEN YOU COMPARE TWO STRINGS, THE RESULT IS A LOGICAL

```
"Red" > "Blue"
```

TESTS IF "RED" IS AFTER "BLUE" ALPHABETICALLY