# STAT 408: Week 6

Simulation, Loops, and If/Else Statements

2/22/2022

# Simulation in R

# What is Simulation

A few questions about simulation:

- 1. What does statistical simulation mean to you?
- 2. Describe a setting where simulation can be used.

# Simulation of Roulette

Consider the casino game Roulette.



We can use simulation to evaluate gambling strategies.

#### **Roulette Simulation in R**

#### kable(RouletteSpin(2), row.names=FALSE)

number	color
10	black
8	black

# Exercise: Probability of Red, Green, and Black

- 1. Calculate/derive the probability of landing on green, red, and black.
- 2. How can the **RouletteSpin()** function be used to compute or approximate these probabilities?

#### Solution: Probability of Red, Green, and Black

In this situation, it is easy to compute the probabilities of each color analytically. However, consider simulating this process many times to estimate these probabilities.

```
num.sims <- 1000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims</pre>
```

Analytically  $P[red] = \frac{18}{38} = 0.4737$ , this is estimated as 0.47. Similarly,  $P[black] = \frac{18}{38} = 0.4737$ , this is estimated as 0.472 and  $P[green] = \frac{2}{38} = 0.0526$ , this is estimated as 0.058

# Exercise: Simulation Questions – Part 2

Now what happens if we:

- 1. run the simulation again with the same number of trials?
- 2. run the simulation with more trials, say 1 million?

# Solution: Simulation Questions – Part 2

Run the simulation again with the same number of trials:

```
num.sims <- 1000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims</pre>
```

The simulated results are different Analytically  $P[red] = \frac{18}{38} = 0.4737$ , this is estimated as 0.458. Similarly,  $P[black] = \frac{18}{38} = 0.4737$ , this is estimated as 0.488 and  $P[green] = \frac{2}{38} = 0.0526$ , this is estimated as 0.054

#### Solution: Simulation Questions – Part 2

Run the simulation with more trials, say 1 million:

```
num.sims <- 1000000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims</pre>
```

Analytically  $P[red] = \frac{18}{38} = 0.4737$ , this is estimated as 0.4742. Similarly,  $P[black] = \frac{18}{38} = 0.4737$ , this is estimated as 0.4727 and  $P[green] = \frac{2}{38} = 0.0526$ , this is estimated as 0.0531

11/48

# **Conditional Expressions in R**

#### Exercise: Conditions in R

We have touched on many of these before, but here are some examples of expressions (conditions) in R. Evaluate these expressions:

```
pi > 3 & pi < 3.5
c(1,3,5,7) %in% 1:3
1:3 %in% c(1,3,5,7)
rand.uniform <- runif(n = 1, min = 0, max = 1)
rand.uniform < .5</pre>
```

#### Solutions: Conditions in R

```
pi > 3 & pi < 3.5
## [1] TRUE
c(1,3,5,7) %in% 1:3
## [1] TRUE TRUE FALSE FALSE
1:3 %in% c(1,3,5,7)
## [1] TRUE FALSE TRUE
rand.uniform <- runif(n = 1, min = 0, max = 1); rand.uniform
## [1] 0.6718267
rand.uniform < .5
## [1] FALSE</pre>
```

# **Conditional Expression: If and Else**

print(rand.uniform)

## [1] 0.6718267

Now what does this return?

```
if (rand.uniform < .5){
   print('value less than 1/2')
} else {
   print('value greater than or equal to 1/2')
}</pre>
```

15/48

# **Conditional Expression: If and Else**

```
print(rand.uniform)
## [1] 0.6718267
if (rand.uniform < .5){
    print('value less than 1/2')
} else {
    print('value greater than or equal to 1/2')
}</pre>
```

```
## [1] "value greater than or equal to 1/2"
```

### **Conditional Expression: Vectorized?**

Does this function accept a vector as an input?

```
rand.uniform2 <- runif(2)
print(rand.uniform2)
if (rand.uniform2 < .5){
    print('value less than 1/2')
} else {
    print('value greater than or equal to 1/2')
}</pre>
```

17/48

# **Conditional Expression: Vectorized?**

```
rand.uniform2 <- runif(2)
print(rand.uniform2)
## [1] 0.8362597 0.8058175
if (rand.uniform2 < .5){
   'value less than 1/2'
} else {
      'value greater than 1/2'
}
## [1] "value greater than 1/2"</pre>
```

### Conditional Expression: ifelse()

print(rand.uniform2)

## [1] "greater than 1/2" "greater than 1/2"

The **ifelse()** function is vectorized and generally preferred with a single set of if/else statements.

19/48

### **Exercise: Conditional Expression**

Write a conditional statement that takes a playing card with two arguments, number (A, 2,..., 10, J, Q, K) and suit (C, D, H, S), and prints **Yes** if the card is a red face card and **No** otherwise.

- 4 of clubs: card.number <- '4' and card.suit <- 'C'</pre>
- King of hearts: card.number <-'K' and card.suit <- 'H'</li>

Verify this works using the following inputs:

- card.number <- 'J' and card.suit <- 'D'</pre>
- card.number <- 'Q' and card.suit <- 'S'</pre>

## Solution: Conditional Expression

## [1] "No"

21/48

# Loops

#### for loops

When you want to do the same thing more than once:

```
output <- vector("numeric", 100) # Set up empty object
for(i in LOOP_OVER_THIS_SEQUENCE) {
    # Repeat this code on each item in the sequence
    # Store in output vector
}
```

# for loops

What will each of these two loops print?

```
rand.uniform2
## [1] 0.8362597 0.8058175
for (i in seq_along(rand.uniform2)){
   print(i)
}
and
for (i in rand.uniform2){
   print(i)
}
```

# for loops

We can loop through a sequence or a vector.

```
for (i in seq_along(rand.uniform2)){
   print(i)
}
### [1] 1
### [1] 2
for (i in rand.uniform2){
   print(i)
}
### [1] 0.8362597
## [1] 0.8058175
```

# While loops

An alternative to for loops is to use the while statement.

#### **Exercise:** Loops

Assume you plan to wager \$1 on red for ten roulette spins. If the outcome is red you win a dollar and otherwise you lose a dollar. Write a loop that simulates ten rolls and determines your net profit or loss.

#hint: to get color from a single spin use
RouletteSpin(1)[2]

## color ## 38 red

27/48

### Solution: Loops

Assume you plan to wager \$1 on red for ten roulette spins. If the outcome is red you win a dollar and otherwise you lose a dollar. Write a loop that simulates ten rolls and determines your net profit or loss.

# Why not loops?

Some of you have seen or heard that loops in R should be avoided.

- Why: it has to do with how code is compiled in R. In simple terms, vectorized operations are much more efficient than loops.
- **How**: we have seen some solutions to this, explicitly using the apply class of functions. We can also write vectorized functions, consider the earlier roulette example where number of spins was an argument for the function.

# **Monte Carlo Procedures**

### **Motivation for Monte Carlo procedures**

Some probabilities can easily be calculated either intuitively or using pen and paper; however, as we have seen we can also simulate procedures to get an approximate answer.

Consider poker, where players are dealt a hand of five cards from a deck of 52 cards. What is the probability of being dealt a full house?

31/48

# Full House Probability Calculation

Could we analytically compute this probability? **Yes** Is it an easy calculation? not necessarily. So consider a (Monte Carlo) simulation.

DealPoker <- function(){
 # Function returns a hand of 5 cards</pre>

#### **Full House Probability Calculation**

Could we analytically compute this probability? **Yes** Is it an easy calculation, not necessarily. So consider a (Monte Carlo) simulation.

33/48

#### **Full House Probability Calculation**

Next write another function to check if the hand is a full house.

```
IsFullHouse <- function(hand){
    #determines whether a hand of 5 cards is a full house
    #ARGS: data frame of 5 cards
    #RETURNS: TRUE or FALSE
    cards <- hand[,2]
    num.unique <- length(unique(cards))
    ifelse(num.unique == 2, return(TRUE), return(FALSE))
}</pre>
```

Does this work? If not, what is the problem and how do we fix it?

#### **Full House Probability Calculation**

35/48

### **Full House Probability Calculation**

num.sims <- 1e5 sim.hands <- replicate(num.sims, DealPoker(), simplify=FALSE) results <- lapply(sim.hands, IsFullHouse) prob.full.house <- sum(unlist(results))/num.sims</pre>

Analytically the probability of getting a full house *can* be calculated as approximately 0.00144, with our simulation procedure we get 0.00141.

# **Closing Thoughts on Monte Carlo**

A few facts about Monte Carlo procedures:

- They return a random result due to randomness in the sampling procedures.
- The run-time (or number of iterations) is fixed and typically specified.
- Mathematically, Monte Carlo procedures are computing an integral or enumerating a sum.
- They take advantage of the law of large numbers.
- They were given the code name Monte Carlo in reference to the Monte Carlo Casino in Monaco.

# Advanced Debugging

#### Overview

We can often fix bugs using the ideas sketched out previously and this becomes *easier* with more experience coding in R. Trial and error can be very effective and strategic use of print() function help to identify where bugs are occuring.

However, R does also have advanced tools to help with debugging code.

- ・ traceback()
- "Rerun with debug"
- browser()

#### traceback()

Consider the following code:

```
f <- function(a) g(a)
g <- function(b) h(b)
h <- function(c) i(c)
i <- function(d) "a" + d
f(10)</pre>
```

## Error in "a" + d: non-numeric argument to binary operator

#### traceback()

Consider the traceback() function. Which identifies which functions have been executed (along with the row number of the function).

> traceback()
4: i(c) at #1
3: h(b) at #1
2: g(a) at #1
1: f(10)

Note: due to the way that R Markdown is compiled, traceback() needs to be run directly in R, not R Markdown.

#### Browsing on an error

Another option (in R Studio) is to browse on the error. This gives you an interactive way to move through the function calls to identify the problem of the location. This can also be called explicitly using debug().



#### browser()

The browser function can also be used to interactively step through a function.

```
SS <- function(mu, x) {
    browser()
    d <- x - mu
    d2 <- d^2
    ss <- sum(d2)
    ss
}</pre>
```

### browser() step 1

Function: SS (.GlobalEnv)	(Read-only) SSQ •
Debug location is approximate because the source is not available.	× Values
1 - function(mu, x) { * 2 browser() 3 d <- x - mu 4 d2 <- d^2 5 ss <- sum(d2) 6 ss	mu 2 x 4
7 }	Traceback Show + SS(2, 4)
	Files     Plots     Packages     Holp     Viewer       Image: State of the state of th
Console R Markdown x	v (tese) P Documentation ply a Function over a List Uector
~/ 🔗	
Stop	cription
<pre>&gt; SS &lt;- function(mu, x) { + browser() + d &lt;- x - mu</pre>	ly returns a list of the same length as X, each int of which is the result of appying FUX to the sponding element of X.
+ d2 <- d^2 + ss <- sum(d2) + ss + }	<pre>ly is a user-friendly version and vergoper of ly by befault unkning a vector, maintor, or if lify = "arcey", an array if appropriate, by ing simplify/arcey(), a seption (y, et a), lify = FALSE, USE. NAMES = FALSE ) is one as lapping/(x, f.).</pre>
> > SS(2, 4) Colled from: SS(2, 4)	ly is similar to eapply, but has a pre-specified of return value, so it can be safer (and sometimes ) to use.
Browse[1]>	icate is a wrapper for the common use of 1y for repeated evaluation of an expression n will usually involve random number ation).
	lify2array() is the utility called from

# browser() step 2

🚵 Debug location is approximate because the source is not available.	Values			
1- function(mu, x) {	mu	2		
2 browser()	×	4		
◆ 3 d <- x - mu				
4 d2 <- $d^2$				
5 ss <- sum(d2)				
6 SS				
	Traceba	c k		Show -
	+ SS(2, 4	.)		
		·		
	Files Plots	Packages	Help Viewe	
	🗢 🤿 🏠	a 🔒	Q,	ି । ଔ
	R: Apply a Fund	tion over a Lis	t or Vector +	Find in Topic
	y {base}		R Documer	itation
	ply a Fur	nction o	ver a Li	st
Console R Markdown ×	ply a Fur Vector	nction o	ver a Li	st
Console & Markdown ×	ply a Fur Vector	nction o	ver a Li	st
Console R Markdown x	ply a Fur Vector	nction o	iver a Li	st
Console  # Markdown ×	ply a Fur Vector cription	CTION O	Iver a Li	st
Console & Markdown ×	ply a Fur Vector cription	t of the same	length as X, en	sch the
Console R Markdown × √ / ∞ 4 = Next   ↑  4 =   > Continue Stop > SS <- function(mu, x) { + browser() + d <- x - mu	ply a Fur Vector cription ly returns a list ant of which is th sponding eleme	t of the same he result of ap ent of X.	length as X, explying FUN to	ach the
	ply a Fur Vector cription ly returns a list ant of which is th sponding eleme ly by default re	t of the same he result of ap ant of x. andly version :	length as X, er pplying FUN to and wrapper o	ach the f
A Markdown ×	ply a Fur Vector cription ly returns a list int of which is ti sponding eleme ly by default re ly by default re lify = "arr	t of the same he result of as not of x. andly version sturning a vec ay", an array	length as X, explying FUN to and wrapper or tor, matrix or, i if appropriate,	ach the f by
Console # Markdown × ~/ A 4= Mark [P]   q=   > Continue ■ Stop > SS <- function(mu, x) { + browser() + d <- x - mu + d2 <- dr2 + ss <- sum(d2) + ss	ply a Fur vector cription ly returns a list nt of which is the sponding eleme ly by default re ly by default re lify = "arr: ing simplify2 lify = PALS!	t of the same he result of ag ant of X. andly version : sturning a vec ay", an array array (). as c, USE.NAM	length as X, explying FUN to and wrapper of tor, matrix or, i if appropriate, apply (x, f, f); S = FALS;	ach the f by
Console R Markdown × -/ ∞ -/ ∞	ply a Fur Vector cription ly returns a list sponding eleme ly by default re lify = "arring simplify" lify = Play lify = Play lify = stapply	t of the same he result of agont of X. andly version : sturning a vec ay", an array array(). az E, USE.Nat r(X, f).	length as X, explying FUN to and wrapper of tor, matrix or, i if appropriate, apply(x, f, ES = FALSE;	ach the đ by is
<pre>cnote # Markdown =</pre>	ply a Fur Vector cription ly returns a list sponding eleme ly by default re lify = "arring simplify? lify = PAUS lify = PAUS lify = simplify?	t of the same he result of agont of X. andly version : sturning a vec ay", an array array().set F, USE.Nat (x, f). sapply, but	length as X, explying FUN to and wrapper or tor, matrix or, i if appropriate, apply(x, f, ES = FALSE; has a pre-spec	ach the đ , by is
	ply a Fur Vector cription ly returns a list nt of which is the sponding eleme ly by default re lify g "arr, ing simplify? lify = PALS; ume as lapply ly is similar to r feturn value, to use.	t of the same he result of ap ont of X. andly version : sturning a vec array(). az array(). az c, USE.NAM ((X, f). sapply, but so it can be s	length as X, explying FUN to and wrapper of tor, matrix or, i appropriate appropriate apply (x, f, ES = PALSE) has a pre-spece afer (and some	ach the đ đ , by v is iffied stimes
Console # Markdown ×     □       ~/∞     */∞       %=maxt     %1 φ=1 > Continue       > SSS << fortunation(mu, x) {       +     browser()       +     d2 < < d+2       +     ss <- sum(d2)       +     ss       > SS( < fortunation(mu, x)         SS << sum(d2)       +     ss       > SS <       > SS( < fortunation(mu, x)        (all of from: SS(2, 4)	ply a Fur Vector cription ly returns a list sponding element ly is a user-fine ing simplify? lify = "atr; ime as lapply ly is similar to st return value, ) to use.	nction c tof the same he result of ag and of x. andly version : sturning a vec ay", an array tarray(), so z, USE.NAM (r(x, f). sapply, but so it can be s	Ver a Li length as x, explying FUN to and wrapper of tor, matrix or, i, if appropriate upp1y(x, f, ES = FALSE; has a pre-spee afer (and som	ach the đ , by i is ::fied stimes
<pre>Console # Markdown *</pre>	Dly a Fur Vector cription ly returns a list nt of which is t y by default re lify = "azri ing simplify: lify = Ptyl lify = Ptyl return value, ) to use. icate is a wra ly for repeated	ACTION O to the same he result of ag nt of X. andly version : turning a vec sy", an array turning to the so the same to the same s, USE.NAM (x, f). sapply, but so it can be so it can be poper for the c	Ver a Li length as X, ex phyling FUN to the if appropriate, apply(x, f, ES = FALSE; has a pre-spet after (and som ommon use of an expression	st ach the d f j by is ::find stimes
Console # Markdown × →/▷ →/▷ →/▷ →/▷ →/▷ →/▷ →/▷ → b Continue ■ Stop → Sto < function(mu, x) { + browser() + d <- x - mu + ss + } > > SS(2, 4) Called from: SS(2, 4) Browser[1]> debug at #3: d <- x - mu Browser[2]= 1	Dly a Fur Vector cription Ly returns a list sponding eleme to the which is the sponding eleme ty by default re lify = "art; ing simplify? Lify = "art; ing simplify? Lify = PALS ume as Lapply Ly is similar to of return value, ) to use.	Action of to the same he result of ag nt of X. andly version : turning a vec agr", an array array (), a: g, USE.NAM r(x, f). asply, but so it can be s pper for the o revaluation of volve random	Ver a Li length as X, et phyling FUN to and wrapper of tor, matrix or, if if appropriate. ppp1/xr, f, ES = FALSE; has a pre-spet afer (and som ommon use of an expression number	ach the đ đ j, by j is sified stimes
Console # Markdown × -//∞ 4≣ Markdown × SS <- function(mu, x) { + browser() + d2 <- d-2 + ss <- sum(d2) + ss + } > > SS(2, 4) Called from: SS(2, 4) Browse[1]> d <- x - mu Browse[2]>	ply a Fur Vector cription ly returns a list proding identified in the proding identified in the proding identified in the proding identified in the proding identified in the ing simplify = PALS ing sing sim	t of the same he result of an int of X. andly version. turning a vec ay", an array array (), as ay, USE.NAM r(x, f). aspp1y, but so it can be s opper for the c i evaluation of volve random	length as X, et pplying FUN to and wrapper of tor, matrix or, i, thapropriate pp1y(x, f, ES = FALSE; has a pre-spea after (and som ommon use of an expression number called from	ach the d d d, by is sified stimes

# browser() step 3

Function: SS (.GlobalEnv)	(Read-only)	🚺 SSO -	Q,
A Debug location is approximate because the source is not available.	×	Values	
1 · function(mu, x) {		d 2	
2 browser()		mu 2	
3 d <- x - mu		× 1	
$\Rightarrow 4$ d2 $<-$ d <sup>2</sup>		^ T	
5 ss <- sum(d2)			
6 ss			
7 }			
		Traceback	Show =
		♦ SS(2, 4)	
		Files Plots Packages	Help Viewer
		🍥 🤿 🏠 🗎 🕰	Q,
		R: Apply a Function over a L	ist or Vector • Find in Topic
		v (base)	B Documentation
		, (0000)	
		oly a Function	over a List
Console R Markdown ×		Vector	
		VECIOI	
S Next P1 (C ) Continue Stop		cription	
+ browserO			
+ d <- x - mu		ly returns a list of the sam	e length as X, each applying FUN to the
+ d2 <- d^2		sponding element of X.	
+ ss <- sum(d2)		lv is a user-friendly version	n and wrapper of
+ SS		ly by default returning a ve	ector, matrix or, if
+ }		ing simplify2array().	sapply(x, f,
		lify = FALSE, USE.NA	MES = FALSE) is
> \$\$(2, 4)		tile as tappiy(x, t).	
Called from: SS(2, 4)		ly is similar to sapply, bu of return value, so it can be	t has a pre-specified safer (and sometimes
Browse[1]>		) to use.	
debug at #3: d <- x - mu		icate is a wrapper for the	common use of
Browse[2]>		1y for repeated evaluation	of an expression
debug at #4: d2 <- d^2		ation).	n number
Browse[2]>		1 (fu?arrau() is the utilit	v called from
		iiiyzaiiay() is the util	y called north

# browser() step 4

Function: SS (.GlobalErv) (Read-only)	SS0 - Q
🚵 Debug location is approximate because the source is not available. 🔹 🖈	Values
1 - function(mu, x) {	d 2
2 browser()	d2 4
3 d <- x - mu	2
4 d2 <- d^2	mu z
◆ 5 ss <- sum(d2)	x 4
6 ss	
7 }	
	Traceback Show
	• SS(2, 4)
	Files Plots Packages Help Viewer
	R: Apply a Function over a List or Vector + Find in Topic
	(here) B Documentation
	y (base) H Documentation
	olv a Function over a List
Console R Markdown ×	Vector
-1 @	100101
e ≡ Next [ <sup>3</sup> [ ] <i>φ</i> = ] ▶ Continue   ■ Stop	cription
+ d2 <- d^2	The returns a list of the same length as X, each
+ ss <- sum(d2)	ant of which is the result of applying FUN to the
+ SS	sponding element of x.
+ }	ly is a user-friendly version and wrapper of
>	lify = "array", an array if appropriate, by
> SS(2, 4)	ing simplify2array().sapply(x, f,
Called from: SS(2, 4)	ime as lapply(x, f).
Browse[1]>	ly is similar to samply, but has a pre-specified
debug at #3: d <- x - mu	of return value, so it can be safer (and sometimes
Browse[2]>	) to use.
debug at #4: d2 <- d^2	icate is a wrapper for the common use of
Browse[2]>	n will usually involve random number
debug at #5: ss <- sum(d2)	ation).
Browse[2]>	lify2array() is the utility called from

# browser() step 5

💠 🗇 📶 🔒 🗆 Source on Save 🛛 Save 🖉 🖌 📃	📑 Run 📑 📑 Source 🔹 🚖 🔂 📑 Import Dataset 🔹 🎻 📃 List 🔹 🎯
1 SS <- function(mu, x) {	🐴 Global Environment 👻 🔍
2 browser()	Functions
3 d <- x - mu	f function (a) 🖾
4 d2 <- $d^2$	g function (b) 🖾
5 ss <- sum(d2)	h function (c)
6 ss	i function (d)
7 }	MakeErr function (in val)
8	SS function (mu x)
9 55(2, 4)	55 Function (ma, x)
	Files       Plots       Packages       Help       Viewer       Image: Comparison of the comparison
	y (base) R Documentation
10.1 (Top Level) :	v(base) R Documentation R Script 2 ply a Function over a List
101 (Top Level) : Console R Markdown ×	(Posse) R Documentation R Script 1 ply a Function over a List Vector
10.1 (Top Level) : Console R Markdown x -/ / 20	(pase) R Documentation     (pase) y a Function over a List     Vector
10.1 (Top Level) : Console    Musrkdown ×   -/ / 0 + }	(¢ase) R Documentation (¢ase) P Documentation ply a Function over a List Vector pription
101 (Top Level) : Console R Markdown ≥ -/ ∞ + } > \$55(2, 4)	(base) R Documentation     (construction over a List     Vector     ription     Jordan a List of the same length as X, each     Jordan a List of the same length as X, each     Jordan a List of the same length as X, each
10.1 (Top Level) : Console A Markdown x -/↔ + } > S(2, 4) Called from: SS(2, 4)	(pase) R Documentation     (pase)     (
101 (Top Level) : Console R Markdown × +/↔ + } > SS(2, 4) (Called from: SS(2, 4) Browse[1]>	( \$\phi_{abs}\)     R Documentation     ( \$\phi_{abs}\)     R Documentation     ( \$\phi_{abs}\)     Ply a Function over a List     Vector     Pription     Ly returns a for of the same length as x, each     nrd of which is the result of applying TXN to the     sponding element of x     Ly is a user/finedly vention and wrapper of
101 (Top Level): Console (R Markdown × → / ∞ + } > 55(2, 4) Called from: 55(2, 4) Browse[1]> debug at #3: d <- x - mu	(base) R Documentation     (compared by the same length as X, each     tription     yreturns a fist of the same length as X, each     tription     yreturns a fist of the same length as X, each     tription     yreturns a fist of the same length as X, each     yreturns a fist of the same len
10.1 (Top Level) : Console R Markdown x -/∞ + } > S(2, 4) Called from: SS(2, 4) Browse[1]> debug at #3: d <- x - mu Browse[2]>	(case) R Documentation     (case)     (
101 (Top Level) : Console    Markdown ×	(\$ease) R Documentation     (\$ease)     (\$Poson     (\$Poson
10.1 (Top Level) : Console R Markdown × -/-⊘ + } > S(2, 4) Colled from: SS(2, 4) Browse[1]> debug at #3: d <- x - mu Browse[2]> debug at #4: d2 <- dA2 Browse[2]>	(case) R Documentation     (case)     (
<pre>10.1 (Top Level) : Console</pre>	(case)     R Documentation     (case)
101 (Top Level) : Console    Markdown ≠	(\$ease)     R Documentation     (* Cose)     (* Cose
10.1       (Top Level) :         Console       A Markdown is         -/->       + }         > S       (Console [Constrainty])         Colled from:       SS(2, 4)         Browse[1]>       debug at #3: d <- x - mu         Browse[2]>       debug at #4: d2 <- d^2         Browse[2]>       debug at #5: ss <- sum(d2)         Browse[2]>       debug at #6: ss	(rease) R Documentation     (rease)
<pre>10.1 (Top Leve) : Console R Markdown # -/~ + } &gt; &gt; SS(2, 4) Called from: SS(2, 4) Browse[1]&gt; debug at #3: d &lt;- x - mu Browse[2]&gt; debug at #4: d2 &lt;- d^2 Browse[2]&gt; debug at #5: ss &lt;- sum(d2) Browse[2]&gt; debug at #6: ss Browse[2]&gt;</pre>	(case)     R Documentation     (case)
10.1 (Top Level) : Console R Markdown × -/-> + } > S(2, 4) Colled from: SS(2, 4) Browse[1]> debug at #3: d <- x - mu Browse[2]> debug at #4: d2 <- dA2 Browse[2]> debug at #5: ss <- sum(d2) Browse[2]> debug at #5: ss <- sum(d2) Browse[2]> debug at #5: ss <- sum(d2) Browse[2]> debug at #5: ss <	(rease)     Rocumentation     (rease)