

#### Functions & for loops

#### Hadley Wickham

#### 1. Review

#### 2. Complete\* prize strategy

- 3. Function structure and calling conventions
- 4. Practice writing functions

#### 5. For loops



## What are three ways to determine if all the elements in a vector are the same?

What's the difference between && and &?





#### All bars

How can we determine if all of the windows are B, BB, or BBB?

#### All bars

How can we determine if all of the windows are B, BB, or BBB?

Take 1 minute to brainstorm possible solutions

windows[1] %in% c("B", "BB", "BBB")
windows %in% c("B", "BB", "BBB")

allbars <- windows %in% c("B", "BB", "BBB")
allbars[1] & allbars[2] & allbars[3]
all(allbars)</pre>

# See also ?any for the complement

#### Your turn

Complete the previous code so that the correct value of prize is set if all the windows are the same, or they are all bars

payoffs <- c("DD" = 800, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)

same <- length(unique(windows)) == 1
allbars <- all(windows %in% c("B", "BB", "BBB"))</pre>

```
if (same) {
   prize <- payoffs[windows[1]]
} else if (allbars) {
   prize <- 5
}</pre>
```

#### Cherries

Need numbers of cherries, and numbers of diamonds (hint: use sum)

Then need to look up values (like for the first case) and multiply together

cherries <- sum(windows == "C")
diamonds <- sum(windows == "DD")</pre>

c(0, 2, 5)[cherries + 1] \*
 c(1, 2, 4)[diamonds + 1]

payoffs <- c("DD" = 800, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)

```
same <- length(unique(windows)) == 1
allbars <- all(windows %in% c("B", "BB", "BBB"))</pre>
```

```
if (same) {
  prize <- payoffs[windows[1]]</pre>
} else if (allbars) {
  prize <- 5
} else {
  cherries <- sum(windows == "C")
  diamonds <- sum(windows == "DD")
  prize <- c(0, 2, 5)[cherries + 1] *
    c(1, 2, 4)[diamonds + 1]
}
```

## Writing a function

Now we need to wrap up this code into a reusable tool. We need a function.

We've used functions a lot, and now it's time to learn how to write one.

## Functions

# What we want

calculate\_prize(c("DD", "DD", "DD"))
calculate\_prize(c("B", "BBB", "BB"))
calculate\_prize(c("B", "7", "C"))

```
calculate_prize <- function(windows) {</pre>
  payoffs <-c("DD" = 800, "7" = 80, "BBB" = 40,
    "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
  same <- length(unique(windows)) == 1</pre>
  allbars <- all(windows %in% c("B", "BB", "BBB"))</pre>
  if (same) {
    prize <- payoffs[windows[1]]</pre>
  } else if (allbars) {
    prize <- 5
  } else {
    cherries <- sum(windows == "C")</pre>
    diamonds <- sum(windows == "DD")</pre>
    prize <- c(0, 2, 5)[cherries + 1] *</pre>
      c(1, 2, 4)[diamonds + 1]
  }
  prize
}
```

```
same <- length(unique(windows)) == 1</pre>
   allbars <- all(windows %in% c("B", "BB", "BBB"))
   if (same) {
     prize <- payoffs[windows[1]]</pre>
   } else if (allbars) {
     prize <- 5
   <u>∧else {</u>
always indent (- sum(windows == "C")
                - sum(windows == "DD")
  inside {}!
     prize <- c(0, 2, 5)[cherries + 1] *</pre>
       c(1, 2, 4)[diamonds + 1]
   }
   prize
   last value in function is result
```

#### Basic structure

- Name
- Input arguments
  - Names/positions
  - Defaults
- Body (the code)
- Output (final result)

```
# Default values
```

```
mean <- function(x) {
   sum(x) / length(x)
}
mean(1:10)</pre>
```

```
mean <- function(x, na.rm = FALSE) {
    if (na.rm) x <- x[!is.na(x)]
    sum(x) / length(x)
}
mean(c(NA, 1:9))
mean(c(NA, 1:9), na.rm = FALSE)
mean(c(NA, 1:9), na.rm = TRUE)</pre>
```

```
# Default values
```

```
mean <- function(x) {</pre>
  sum(x) / length(x)
}
mean(1:10)
                                default value
mean <- function(x, na.rm = FALSE) {</pre>
  if (na.rm) x <- x[!is.na(x)]
  sum(x) / length(x)
}
mean(c(NA, 1:9))
mean(c(NA, 1:9), na.rm = FALSE)
mean(c(NA, 1:9), na.rm = TRUE)
```

#### Your turn

Write a function to calculate the sample variance of a vector.

Start by calculating the variance of vector of values called x. Make sure you can do the computation before writing the function.

Hint: you'll need to use length, sum and mean.

## Strategy

Always want to start simple: start with test values and get the body of the function working first.

Check each step as you go.

Don't try and do too much at once!

"Wrap it up" as a function only once everything works.

```
x <- runif(100)</pre>
var(x)
n <- length(x)
xbar <- mean(x)</pre>
x – xbar
(x - xbar) ^ 2
sum(x - xbar)^{2}
sum((x - xbar)^{2})
sum((x - xbar) ^ 2) / n - 1
sum((x - xbar)^{2}) / (n - 1)
```

```
my_var <- function(x) {
    n <- length(x)
    xbar <- mean(x)
    sum((x - xbar) ^ 2) / (n - 1)
}</pre>
```

## Testing

Always a good idea to test your code.

We have a prebuilt set of test cases: the prize column in slots.csv

So **for** each row in slots.csv, we need to calculate the prize and compare it to the actual. (Hopefully they will be same!)

# For loops

#### For loops

print(1) print(2) print(3) print(4) print(5) print(6) print(7) print(8) print(9) print(10)

for(value in 1:10) {
 print(value)
}

#### For loops

```
cuts <- levels(diamonds$cut)
for(cut in cuts) {
   selected <- diamonds$price[diamonds$cut == cut]
   print(cut)
   print(mean(selected))
}
# Have to do something with output!</pre>
```

# Common pattern: create object for output, # then fill with results

```
cuts <- levels(diamonds$cut)
means <- rep(NA, length(cuts))</pre>
```

```
for(i in seq_along(cuts)) {
   sub <- diamonds[diamonds$cut == cuts[i], ]
   means[i] <- mean(sub$price)
}</pre>
```

# We will learn more sophisticated ways to do this # later on, but this is the most explicit # Common pattern: create object for output, # then fill with results

```
cuts <- levels(diamonds$cut)
means <- rep(NA, length(cuts))</pre>
```

```
for(i in seq_along(cuts)) { Why use i and not cut?
   sub <- diamonds[diamonds$cut == cuts[i], ]
   means[i] <- mean(sub$price)
}</pre>
```

# We will learn more sophisticated ways to do this # later on, but this is the most explicit

```
1:5
seq_len(5)
1:10
seq_len(10)
1:0
seq_len(0)
seq_along(1:10)
1:10 * 2
seq_along(1:10 * 2)
```

#### Your turn

### For each diamond colour, calculate the median price and carat size

```
colours <- levels(diamonds$color)
n <- length(colours)
mprice <- rep(NA, n)
mcarat <- rep(NA, n)</pre>
```

```
for(i in seq_len(n)) {
   set <- diamonds[diamonds$color == colours[i], ]
   mprice[i] <- median(set$price)
   mcarat[i] <- median(set$carat)
}</pre>
```

results <- data.frame(colours, mprice, mcarat)</pre>

#### Back to slots...

For each row, calculate the prize and save it, then compare calculated prize to actual prize

Question: given a row, how can we extract the slots in the right form for the function?

```
slots <- read.csv("slots.csv")</pre>
i <- 334
slots[i, ]
slots[i, 1:3]
str(slots[i, 1:3])
as.character(slots[i, 1:3])
c(as.character(slots[i, 1]), as.character(slots[i, 2]),
as.character(slots[i, 3]))
slots <- read.csv("slots.csv", stringsAsFactors = F)</pre>
str(slots[i, 1:3])
as.character(slots[i, 1:3])
```

calculate\_prize(as.character(slots[i, 1:3]))

# Create space to put the results
slots\$check <- NA</pre>

```
# For each row, calculate the prize
for(i in seq_len(nrow(slots))) {
   w <- as.character(slots[i, 1:3])
   slots$check[i] <- calculate_prize(w)
}</pre>
```

```
# Check with known answers
subset(slots, prize != check)
# Uh oh!
```

# Create space to put the results
slots\$check <- NA</pre>

```
# For each row, calculate the prize
for(i in seq_len(nrow(slots))) {
   w <- as.character(slots[i, 1:3])
   slots$check[i] <- calculate_prize(w)
}</pre>
```

```
# Check with known answers
subset(slots, prize != check)
# Uh oh!
```

What is the problem? Think about the most general case

DD	DD	DD	800
7	7	7	80
BBB	BBB	BBB	40
BB	BB	BB	25
В	В	В	10
С	С	С	10
Any bar	Any bar	Any bar	5
С	С	*	5
С	*	С	5
С	*	*	2
*	С	*	2
*	*	С	2

windows <- c("DD", "C", "C")
# How can we calculate the
# payoff?</pre>

DD doubles any winning combination. Two DD quadruples. DD is wild