The joy of functional programming

June 2019

Hadley Wickham
@hadleywickham
Chief Scientist, RStudio
Joy of Cooking
Import → Tidy → Transform → Visualise → Model → Communicate → Program
Motivation
Imagine we want to read in a bunch of CSV files

```r
# Find all the CSV files in the current directory
paths <- dir(pattern = "\.csv$")

# And read them in as data frames
data <- vector("list", length(paths))
for (i in seq_along(paths)) {
  data[[i]] <- read.csv(paths[[i]])
}
```
Imagine we want to read in a bunch of csv files

```r
# Find all the csv files in the current directory
paths <- dir(pattern = "\\.csv$")

# And read them in as data frames
data <- vector("list", length(paths))
for (i in seq_along(paths)) {
  data[[i]] <- read.csv(paths[[i]])
}
```

R uses <- for assignment
A loop always has three components

data <- vector("list", length(paths))
for (i in seq_along(paths)) {
  data[[i]] <- read.csv(paths[[i]])
}
Create a new list of the correct size

```r
data <- vector("list", length(paths))
for (i in seq_along(paths)) {
  data[[i]] <- read.csv(paths[[i]])
}
```
2. A vector to iterate over

```r
data <- vector("list", length(paths))
for (i in seq_along(paths)) {
  data[[i]] <- read.csv(paths[[i]])
}
```

Avoid `1:length(paths)` because it fails in an unhappy way if `paths` has length 0.
3. Code that’s run for every iteration

```r
data <- vector("list", length(paths))
for (i in seq_along(paths)) {
    data[[i]] <- read.csv(paths[[i]])
}
```

- Use `[[` whenever you get or set a single element.
- Extract element `i` from `paths`.
There’s nothing wrong with using a loop

```r
library(purrr)

# But the FP equivalent is much shorter
data <- map(paths, read.csv)

# And has convenient extensions
data <- map_dfr(paths, read.csv, id = "path")
```
Why not for loops?
Vanilla cupcakes

1 cup flour
a scant ¾ cup sugar
1 ½ t baking powder
3 T unsalted butter
½ cup whole milk
1 egg
¼ t pure vanilla extract

Preheat oven to 350°F.
Put the flour, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.
Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.
Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.
Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Chocolate cupcakes

3/4 cup + 2T flour
2 1/2 T cocoa powder
a scant 3/4 cup sugar
1 1/2 t baking powder
3 T unsalted butter
1/2 cup whole milk
1 egg
1/4 t pure vanilla extract

Preheat oven to 350°F.

Put the flour, cocoa, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.

Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
### Chocolate cupcakes

3/4 cup + 2T flour
2 1/2 T cocoa powder
a scant 3/4 cup sugar
1 1/2 t baking powder
3 T unsalted butter
1/2 cup whole milk
1 egg
1/4 t pure vanilla extract

Preheat oven to 350°F.

Put the flour, cocoa, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps.

Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Vanilla cupcakes

Preheat oven to 350°F.

Put the flour, sugar, baking powder, salt, and butter in a freestanding electric mixer with a paddle attachment and beat on slow speed until you get a sandy consistency and everything is combined.

Whisk the milk, egg, and vanilla together in a pitcher, then slowly pour about half into the flour mixture, beat to combine, and turn the mixer up to high speed to get rid of any lumps. Turn the mixer down to a slower speed and slowly pour in the remaining milk mixture. Continue mixing for a couple of more minutes until the batter is smooth but do not overmix.

Spoon the batter into paper cases until 2/3 full and bake in the preheated oven for 20-25 minutes, or until the cake bounces back when touched.
Vanilla cupcakes

120g flour  
140g sugar  
1.5 t baking powder  
40g butter  
120ml milk  
1 egg  
0.25 t vanilla

Beat flour, sugar, baking powder, salt, and butter until sandy. Whisk milk, egg, and vanilla. Mix half into flour mixture until smooth (use high speed). Beat in remaining half. Mix until smooth.

Bake 20-25 min at 170°C.
Vanilla cupcakes

120g flour
140g sugar
1.5 t baking powder
40g butter
120ml milk
1 egg
0.25 t vanilla

Beat dry ingredients + butter until sandy.
Whisk together wet ingredients. Mix half into dry until smooth
(use high speed). Beat in remaining half. Mix until smooth.
Bake 20-25 min at 170°C.
# Cupcakes

**Vanilla**

- 120g flour
- 140g sugar
- 1.5t baking powder
- 40g butter
- 120ml milk
- 1 egg
- 0.25 t vanilla

**Chocolate**

- 100g flour
- 20g cocoa
- 140g sugar
- 1.5t baking powder
- 40g butter
- 120ml milk
- 1 egg
- 0.25 t vanilla

Beat dry ingredients + butter until sandy.
Whisk together wet ingredients. Mix half into dry until smooth (use high speed). Beat in remaining half. Mix until smooth. Bake 20-25 min at 170°C.
Cupcakes

Vanilla
- 120g flour
- 140g sugar
- 1.5t baking powder
- 40g butter
- 120ml milk
- 1 egg
- 0.25 t vanilla

Chocolate
- 100g flour
- 20g cocoa
- 140g sugar
- 1.5t baking powder
- 40g butter
- 120ml milk
- 1 egg
- 0.25 t vanilla

Espresso
- 120g flour
- 140g sugar
- 1.5t baking powder
- 40g butter
- 120ml milk + 10g espresso powder
- 1 egg

Beat dry ingredients + butter until sandy.
Whisk together wet ingredients. Mix half into dry until smooth (use high speed). Beat in remaining half. Mix until smooth.
Bake 20-25 min at 170°C.
What do these for loops do?

```r
out1 <- vector("double", ncol(mtcars))
for(i in seq_along(mtcars)) {
  out1[[i]] <- mean(mtcars[[i]], na.rm = TRUE)
}
```

```r
out2 <- vector("double", ncol(mtcars))
for(i in seq_along(mtcars)) {
  out2[[i]] <- median(mtcars[[i]], na.rm = TRUE)
}
```

Extraction column i
For loops emphasise the objects

\[
\text{out1} \leftarrow \text{vector}("\text{double}", \text{ncol(mtcars)})
\]
\[
\text{for}(i \in \text{seq\_along(mtcars)}) \{ \\
\quad \text{out1}[i] \leftarrow \text{mean(mtcars}[i], \text{na.rm} = \text{TRUE})
\}
\]

\[
\text{out2} \leftarrow \text{vector}("\text{double}", \text{ncol(mtcars)})
\]
\[
\text{for}(i \in \text{seq\_along(mtcars)}) \{ \\
\quad \text{out2}[i] \leftarrow \text{median(mtcars}[i], \text{na.rm} = \text{TRUE})
\}
\]
out1 <- vector("double", ncol(mtcars))
for(i in seq_along(mtcars)) {
    out1[[i]] <- mean(mtcars[[i]], na.rm = TRUE)
}

out2 <- vector("double", ncol(mtcars))
for(i in seq_along(mtcars)) {
    out2[[i]] <- median(mtcars[[i]], na.rm = TRUE)
}
Functional programming weights action and object equally

```r
out1 <- map_dbl(mtcars, mean, na.rm = TRUE)
out2 <- map_dbl(mtcars, median, na.rm = TRUE)
```
And combines well with the pipe

```r
out1 <- mtcars %>% map_dbl(mean, na.rm = TRUE)
out2 <- mtcars %>% map_dbl(median, na.rm = TRUE)
```
Which is particularly important for harder problems

diamonds %>%
  split_by(diamonds$color) %>%
  map(~ lm(log(price) ~ log(carat), .x)) %>%
  map_dfr(broom::tidy, .id = "color")
Of course someone has to write loops. It doesn’t have to be you.

— Jenny Bryan
Getting data

<table>
<thead>
<tr>
<th>Description of Food or Drink</th>
<th>Total Weight</th>
<th>Description of Food or Drink</th>
<th>Total Cost</th>
<th>Interviewer Use Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vita Pint (Blue Carton semi-skimmed milk)</td>
<td>1pt</td>
<td></td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>2 Sunblest sliced white loaves @ 800gms</td>
<td>1600g</td>
<td></td>
<td>1 20</td>
<td></td>
</tr>
<tr>
<td>Weetabix – Family Pack x 24 biscuits</td>
<td>450g</td>
<td></td>
<td>1 39</td>
<td></td>
</tr>
<tr>
<td>Frying steak – Fresh</td>
<td>1 lb</td>
<td></td>
<td>5 25</td>
<td></td>
</tr>
<tr>
<td>Pork chops – on bone – fresh</td>
<td>1 lb</td>
<td></td>
<td>2 83</td>
<td></td>
</tr>
<tr>
<td>1 pack danish bacon, streaky, pre-packed</td>
<td>1 lb</td>
<td></td>
<td>2 74</td>
<td></td>
</tr>
<tr>
<td>6 Doughnuts @ 20g – fresh from bakers</td>
<td>12 oz</td>
<td></td>
<td>1 80</td>
<td></td>
</tr>
<tr>
<td>2 tins Heinz baked beans @ 420gms</td>
<td>840g</td>
<td></td>
<td>2 84</td>
<td></td>
</tr>
<tr>
<td>Birds Eye frozen cod steaks in natural crumbs</td>
<td>200g</td>
<td></td>
<td>2 69</td>
<td></td>
</tr>
<tr>
<td>Walls beef sausages – frozen</td>
<td>1 lb</td>
<td></td>
<td>1 32</td>
<td></td>
</tr>
<tr>
<td>McCain Deep Pizza, frozen - pepperoni</td>
<td>493g</td>
<td></td>
<td>2 29</td>
<td></td>
</tr>
<tr>
<td>Ski fruit yogurt, low fat with pieces of fruit</td>
<td>150g</td>
<td></td>
<td>1 12</td>
<td></td>
</tr>
<tr>
<td>2 packets Krona Reduced fat 2 @ 250gms</td>
<td>500g</td>
<td></td>
<td>1 14</td>
<td></td>
</tr>
<tr>
<td>1 packet Kerrygold butter – Irish</td>
<td>250g</td>
<td></td>
<td>1 13</td>
<td></td>
</tr>
<tr>
<td>“Old” Potatoes, fresh, prepacked</td>
<td>5Kg</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Cauliflower – fresh</td>
<td>1 lb 4 oz</td>
<td></td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Eating apples – fresh</td>
<td>1 1/4 lb</td>
<td></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Robinson’s orange squash – low cal</td>
<td>1 litre</td>
<td></td>
<td>1 19</td>
<td></td>
</tr>
</tbody>
</table>
map( , f)
purrrr::map (cupcakes, frost, vanilla)
Demo
Generating reports
map([1, 2, 3], f)
map(, f, ) →
purr:=map2(cupcakes, frost, C('vanilla', 'Chocolate', 'strawberry'))
Demo
Conclusion
For loops aren’t bad; but duplicated code can conceal important differences, and why do more work than you have to?

https://adv-r.hadley.nz/functionals.html

https://r4ds.had.co.nz/iteration.html
With big thanks to Allison Horst!

https://github.com/allisonhorst