FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

How to purrr safely()

Auriel Fournier
Instructor

safely()

```
a <- list("unknown", 10) %>%
  map(safely(function(x)
  x * 10,
  otherwise = NA_real_))
```

```
[[1]]
[[1]]$result
[1] NA
[[1]]$error
<simpleError in x * 10: non-numeric
argument to binary operator>

[[2]]
[[2]]$result
[1] 100
[[2]]$error
NULL
```

Reordering

```
a <- list("unknown",10) %>%
 map(safely(function(x)
              x * 10,
        otherwise = NA_real_)) %>%
 transpose()

$result
$result[[1]]
[1] NA
$result[[2]]
[1] 100


$error
$error[[1]]
<simpleError in x * 10:
non-numeric argument to
binary operator>
$error[[2]]
NULL
...
```

FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

Let's purrr-actice!

FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

Another way to possibly()
purrr

Auriel Fournier
Instructor

safely() then possibly()

```
a <- list(-10, "unknown", 10) %>%
  map(safely(function(x)
  x * 10,
  otherwise = NA_real_))

a
```

```
[[1]]
[[1]]$result
[1] -100
[[1]]$error
NULL

[[2]]
[[2]]$result
[1] NA
[[2]]$error
<simpleError in x * 10: non-numeric
argument to binary operator>

[[3]]
[[3]]$result
[1] 100
[[3]]$error
NULL
```

possibly()

```
a <- list(-10, "unknown", 10) %>%
     map(possibly(function(x)
         x * 10,
         otherwise = NA_real_))

a
```

```
[[1]]
[1] -100

[[2]]
[1] NA

[[3]]
[1] 100
```

FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

Let's purrr-actice!

FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

purrr is a walk() in the park

Auriel Fournier
Instructor

Why walk()?

```
short_list <- list(-10, 1, 10)

short_list
```

```
[[1]]
[1] -10

[[2]]
[1] 1

[[3]]
[1] 10
```

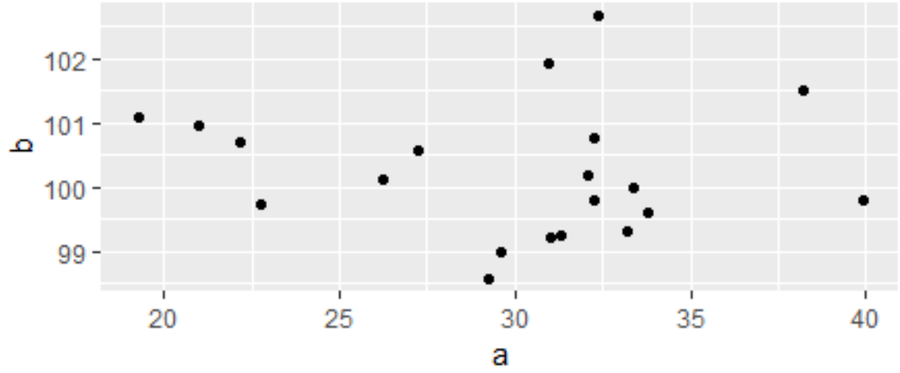
```
walk(short_list, print)
```

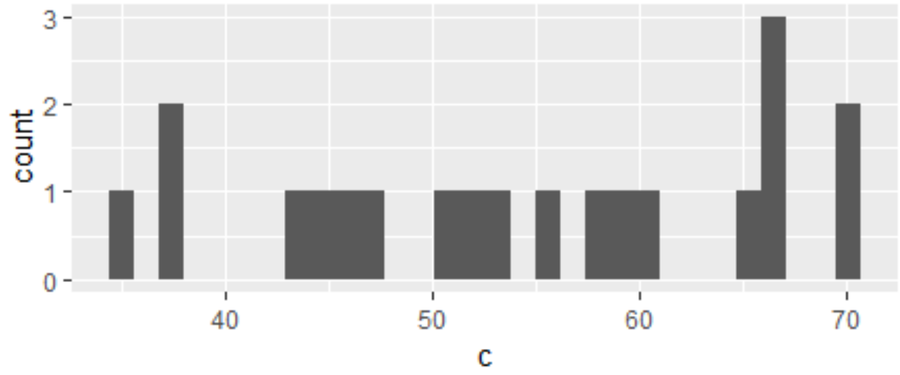
```
[1] -10
[1] 1
[1] 10
```

Plots, the normal way

```
plist
```

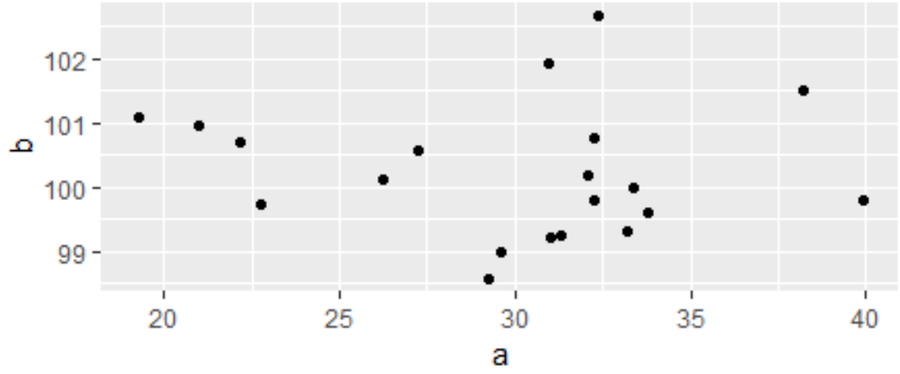
```
[[1]]

[[2]]
```

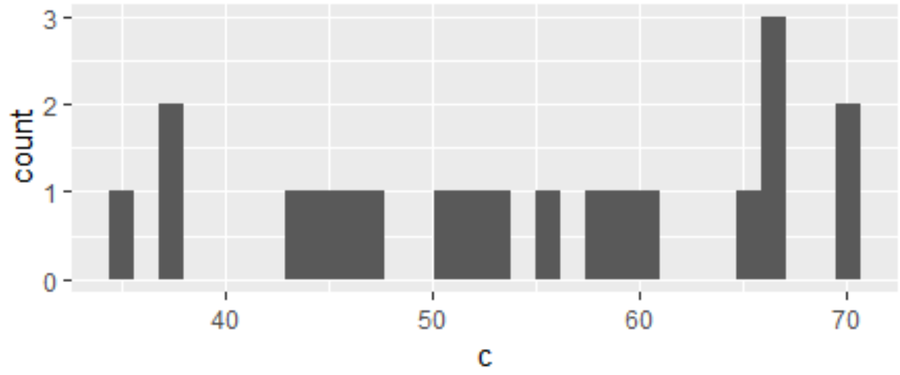




walk() with plots

```
walk(plist, print)
```





FOUNDATIONS OF FUNCTIONAL PROGRAMMING WITH PURRR

Let's purrr-actice!