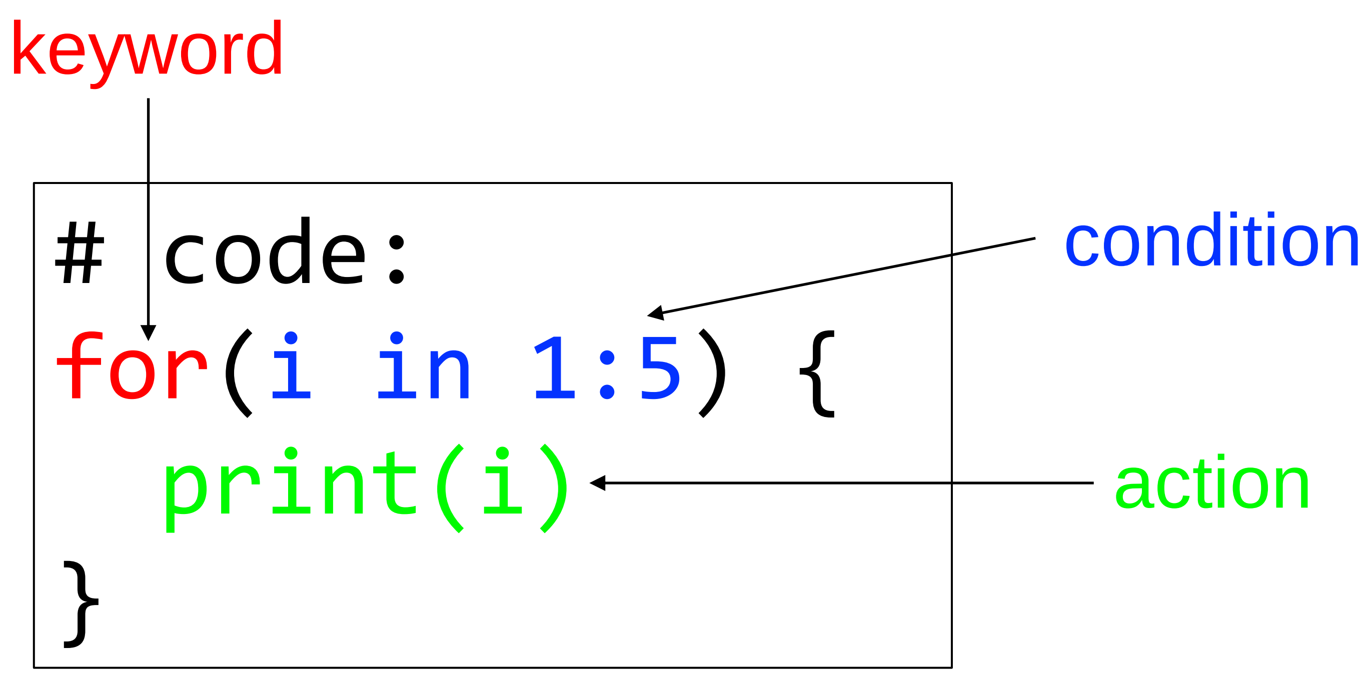
STA101L : Lab 4
For loops, Functions, predict() and RDATA

For Loops in R

```
# code:
for(i in 1:5) {
  print(i)
}
```

```
# output:
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Components of an R for loop:



```
# code:
for(i in 1:5) {
  print(i)
}
```

Note: ( ) around condition and { } around the action are important!

Examples:

```
# code:
for (i in 1.5:5.5) {
  print(i)
}
```

```
# output:
[1] 1.5
[1] 2.5
[1] 3.5
[1] 4.5
[1] 5.5
```

What if we only want to increment by 0.5 each time?

Examples:

```
# code:
for (i in seq(from=1, to=3, by=0.5)) {
  print(i)
}
```

```
# output:
[1] 1
[1] 1.5
[1] 2
[1] 2.5
[1] 3
```

What if we only want to go in reverse order?

Examples:

```
# code:
for (i in seq(from=3, to=1, by=-0.5)) {
  print(i)
}
```

```
# output:
[1] 3
[1] 2.5
[1] 2
[1] 1.5
[1] 1
```

Examples:

```
# code:
for (i in 5:1) {
  print(i)
}
```

```
# output:
[1] 5
[1] 4
[1] 3
[1] 2
[1] 1
```

Examples: Nested For Loops (extra)

```
# code:
for (i in 5:1) {
  for (j in 1:5) {
    cat(j, " ")
  }
  cat("\n")
}
```

```
# output:
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
1  2  3  4  5
```

Alternatives to for loops (extra):

- while loops:

```
i <- 1
while(i <= 3) {
   print(i)
   i <- i + 1
}
```

- lapply function:

```
lapply(1:4, function(a) {a + 1})
```

- sapply function:

```
sapply(1:4, function(a) {a + 1})
```

Functions in R

- Convenient way to group together tasks into a single action
- Can be used to simplify tasks that are repetitive
- Implement flexibility using inputs (arguments)
- Return a meaningful result (return value) computed from the inputs (arguments)

Example: Functions in R

```
df <- tibble::tibble(
  a = rnorm(10), b = rnorm(10),
  c = rnorm(10), d = rnorm(10)
)

df$a_scaled <- (df$a - min(df$a)) / (max(df$a) - min(df$a))
df$b_scaled <- (df$b - min(df$b)) / (max(df$b) - min(df$b))
df$c_scaled <- (df$c - min(df$c)) / (max(df$c) - min(df$c))
df$d_scaled <- (df$d - min(df$d)) / (max(df$d) - min(df$d))
```

Adapted from R for Data Science by Wickham & Grolemund

Very repetitive & difficult to read!

Example: Functions in R

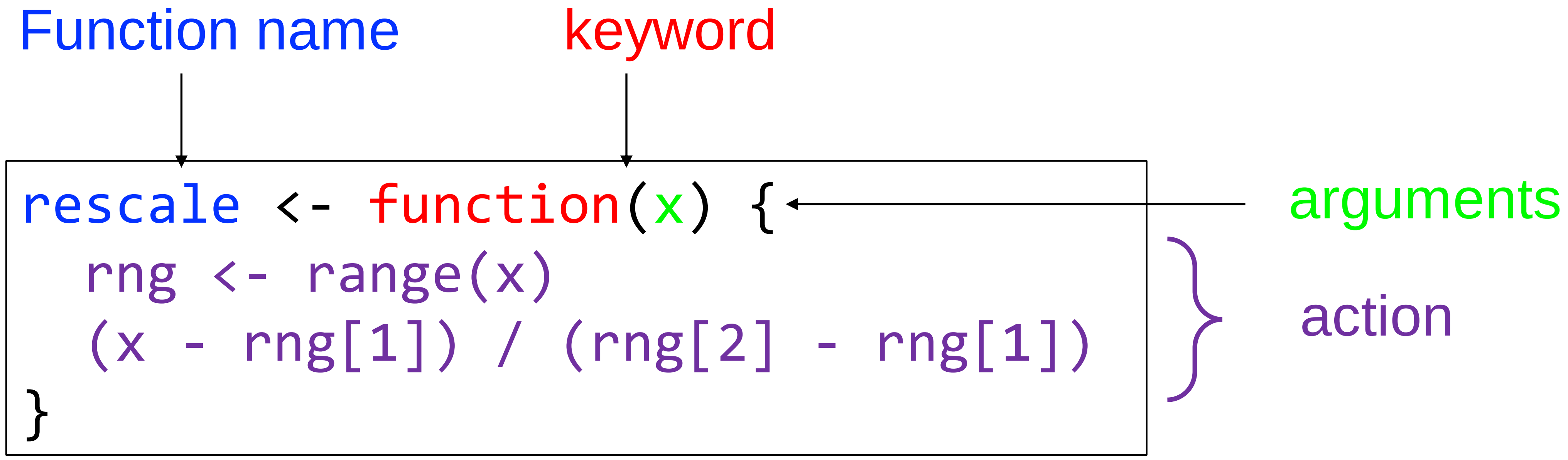
Repetitive code:

```
df$a_scaled <- (df$a - min(df$a)) / (max(df$a) - min(df$a))
```

Rewrite as a function:

```
rescale01 <- function(x) {
  rng <- range(x)
  (x - rng[1]) / (rng[2] - rng[1])
}
```

Components of a function in R



```r
rescale <- function(x) {
  rng <- range(x)
  (x - rng[1]) / (rng[2] - rng[1])
}
```

Note: ( ) around arguments, and { } around the action are important!

Using a function in R

```
df <- tibble::tibble(
  a = rnorm(10), b = rnorm(10),
  c = rnorm(10), d = rnorm(10)
)

df$a_scaled <- rescale(df$a)
df$b_scaled <- rescale(df$b)
df$c_scaled <- rescale(df$c)
df$d_scaled <- rescale(df$d)
```

Much more readable code!

Example of function in R: predict()

```
# Fit a linear model:
m <- lm(salary ~ hours + experience, data = employees)

# Create a test dataset:
test_data <- data.frame(
     hours = c(1, 2, 3), experience = c(10, 25, 11)
)

# Make predictions for test dataset:
test_data$pred_salary <- predict(m, test_data)
```

```
Format of the function: predict(<model>, <test dataset>)
```

R Tutorial + Exercise on everything we learnt + RDATA