R Bootcamp Part 3

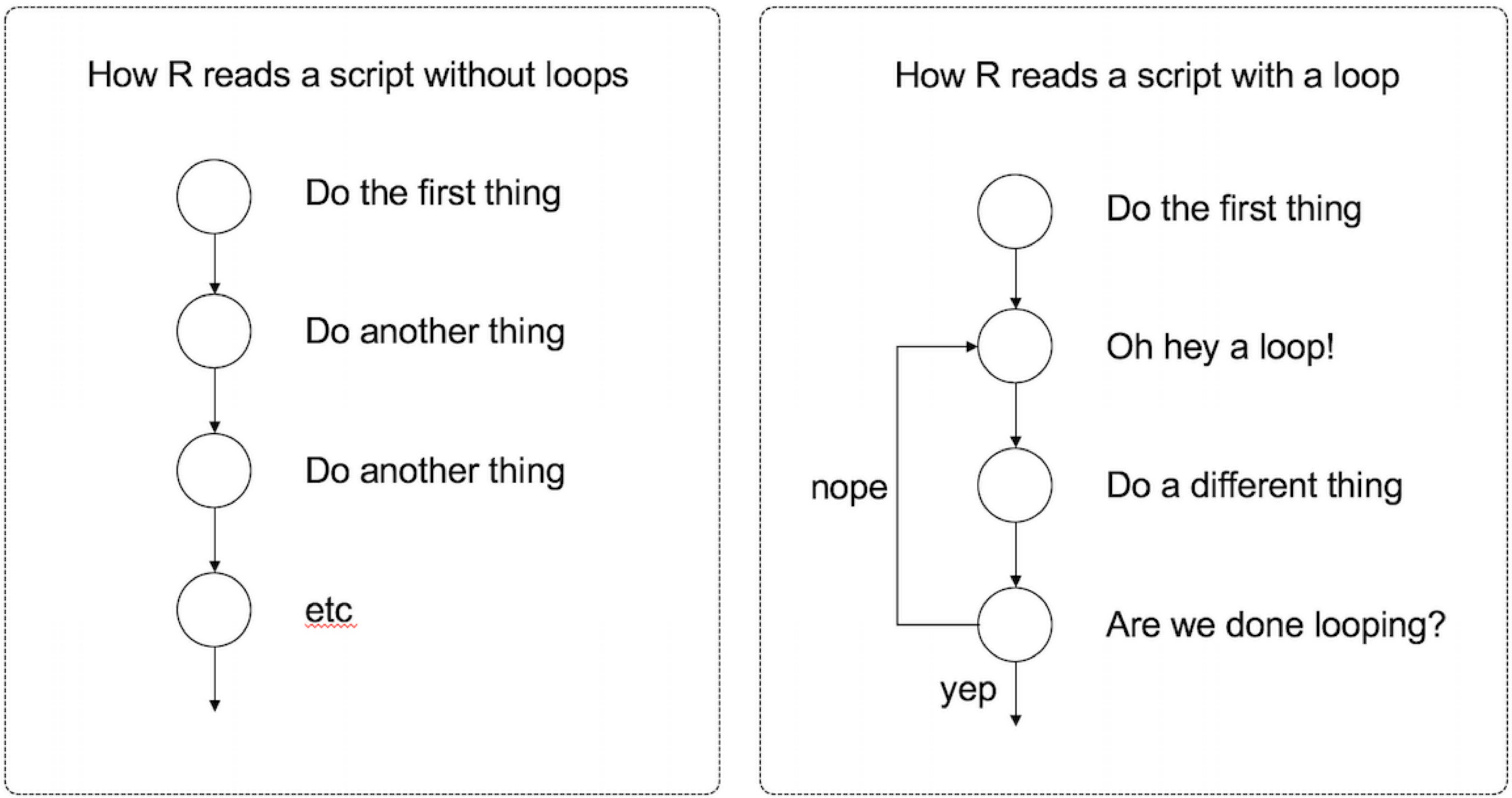
Dani Navarro
Amy Perfors

Today’s Plan

1. Loops
2. Branches
3. Functions
4. Programming
5. File system

Loops

The purpose of a loop



While loops

```
while ( CONDITION ) {
   STATEMENT1
   STATEMENT2
   ETC
}
```

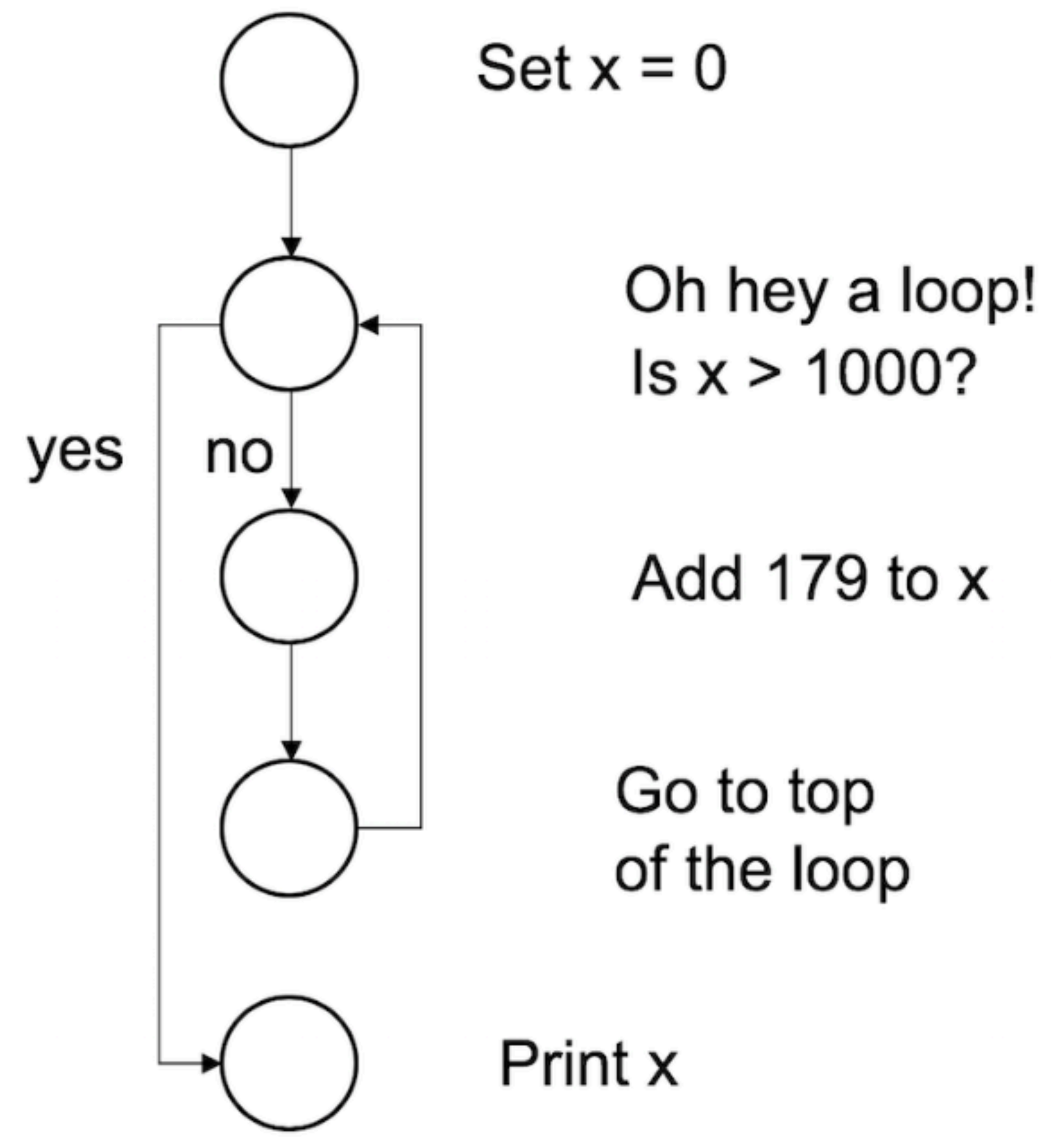
Needs to be a logical (TRUE or FALSE)

Does all of these things as long as the condition is TRUE

While loops

```r
x <- 0
while (x < 1000) {
  x <- x + 179
}
print(x)
```

```
## [1] 1074
```



For loops

```
for ( VAR in VECTOR ) {
   STATEMENT1
   STATEMENT2
   ETC
}
```

Counts through each of the things in the vector

Does all of these things as long as the condition is TRUE

For loops

```
for ( value in 1:10 ) {
  answer <- 137*value
  print(answer)
}
```

```
# 137
# 274
# 411
# 548
# 685
# 822
# 959
# 1096
# 1233
# 1370
```

Looping over vectors

```
words <- c("farewell","cruel","world")
for (thisWord in words) {
   nLetters <- nchar(thisWord)
   blockWord <- toupper(thisWord)
   cat(blockWord,"has",nLetters,"letters\n")
}
```

```
# FAREWELL has 8 letters
# CRUEL has 5 letters
# WORLD has 5 letters
```

Branches

Branches

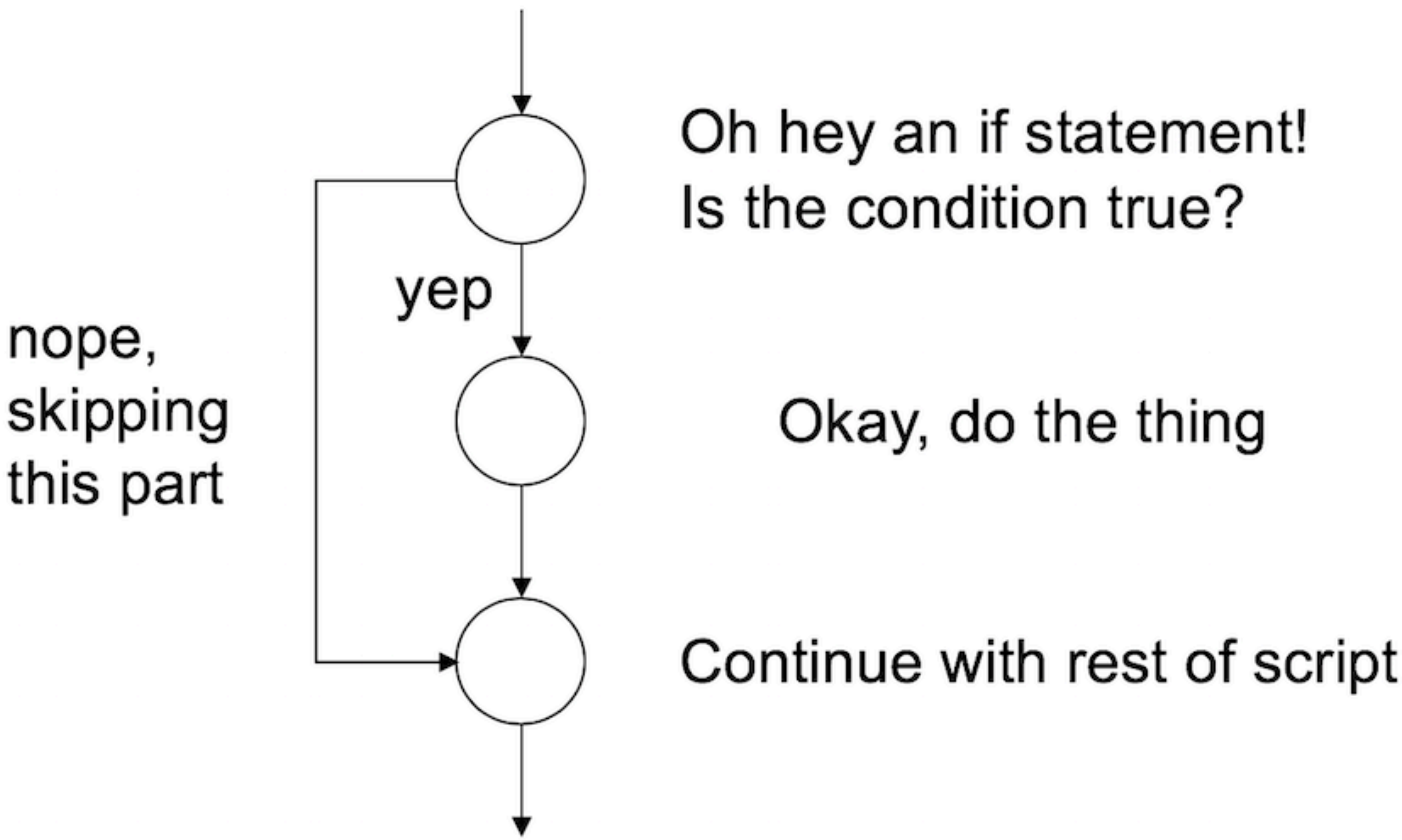
These let you evaluate conditional statements and do different things depending on the outcome

```
If ( CONDITION ) {
   STATEMENT1
   STATEMENT2
   ETC
}
```

Needs to be a logical (TRUE or FALSE)

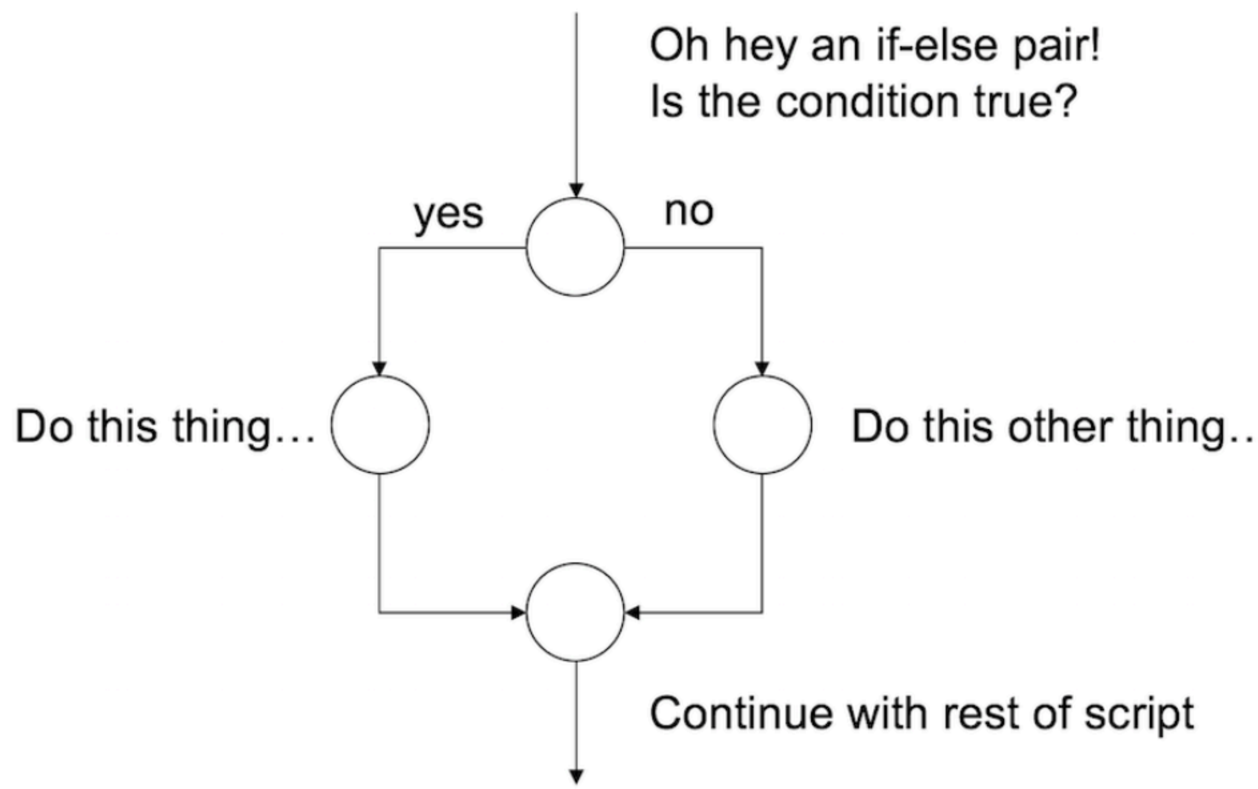
Does all of these things only if the condition is TRUE

Branches


Oh hey an if statement!
Is the condition true?
yep
nope,
skipping
this part
Okay, do the thing
Continue with rest of script

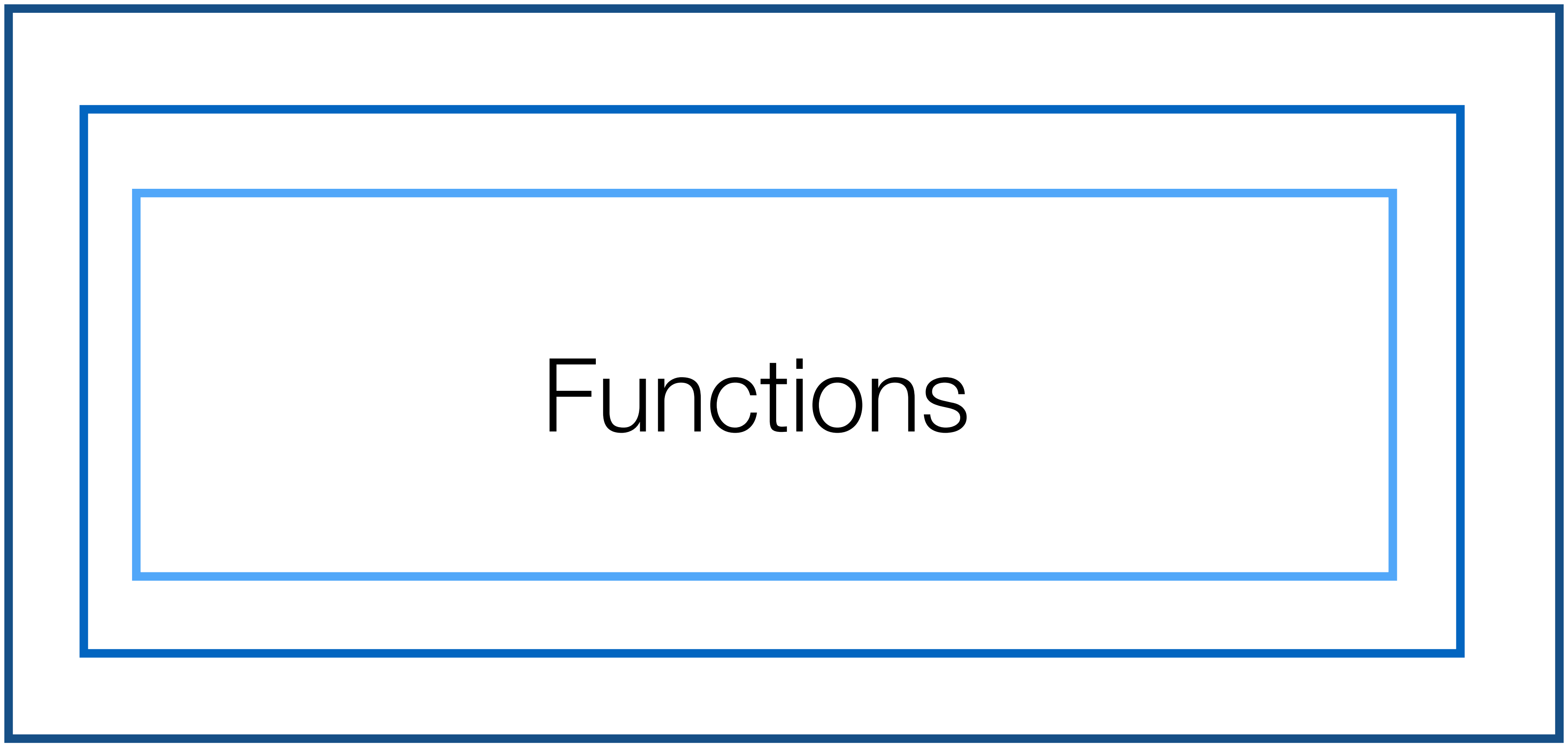
If-Else

```
if ( CONDITION ) {
  STATEMENT1
  STATEMENT2
  ETC
} else {
  STATEMENT3
  STATEMENT4
}
```



Example

```
if (today==“Saturday”) {
   print(“Yay! Weekend!”)
} else if (today==“Sunday”) {
   print(“Uh oh, Monday is coming”)
} else {
   print(“I need coffee.”)
}
```


Functions

Functions

You can actually create your *own* functions with arguments. Whenever it is called R will execute the statements within it. Creating a function means R creates a temporary environment with it while it's in practice, and only "keeps" the value in the `return()` statement.

```
FNAME <- function (ARG1, ARG2, ARG3, ETC) {
   STATEMENT1
   STATEMENT2
   STATEMENT3
   ETC
   return (VALUE)
}
```

Functions

Here’s an example of a function that will square any number.

```
square <- function(x) {
   y <- x*x
   return(y)
}
```

```
> square(4)
# 16
```

Functions

The … argument lets the user enter as many arguments as they would like, as in the example below.

```
doubleMax <- function(...) {
   maxVal <- max(...)
   out <- 2*maxVal
   return(out)
}
```

Bringing it all together

What is all this about?????

Suppose we present a compound stimulus AB, which consists of two things, a tone (A) and a light (B). This compound is presented together with a shock. In associative learning studies, this kind of trial is denoted AB+ to indicate that the outcome (US) was present at the same time as the two stimuli that comprise the CS. According to the Rescorla-Wagner model, the rule for updating the associative strengths v_A and v_B between the originally neutral stimuli and the shock is given by:

$$\begin{aligned} v_A &\leftarrow v_A + \alpha_A \beta_U (\lambda_U - v_{AB}) \\ v_B &\leftarrow v_B + \alpha_B \beta_U (\lambda_U - v_{AB}) \end{aligned}$$

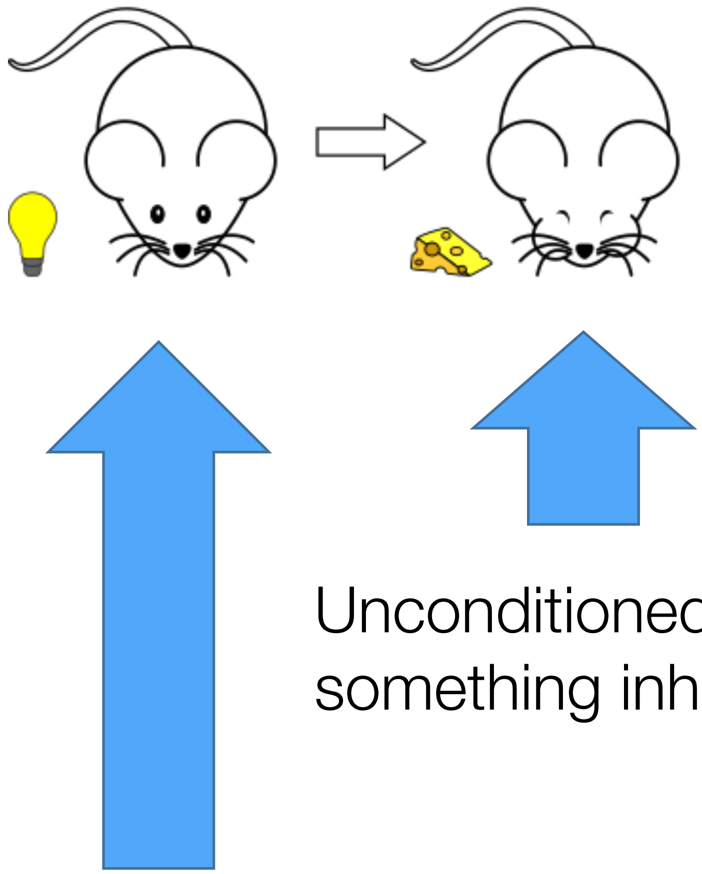
where the associative value v_{AB} of the compound stimulus AB is just the sum of the values of the two items individually. This is expressed as:

$$v_{AB} = v_A + v_B$$

To understand this rule, note that:

- λ_U is a variable that represents the "reward value" (or "punishment value") of the US itself, and as such represents the maximum possible association strength for the CS.
- β_U is a learning rate linked to the US (e.g. how quickly do I learn about shocks?)
- α_A is a learning rate linked to the CS (e.g, how quickly do I learn about tones?)
- α_B is also a learning rate linked to the CS (e.g, how quickly do I learn about lights?)

Associative learning



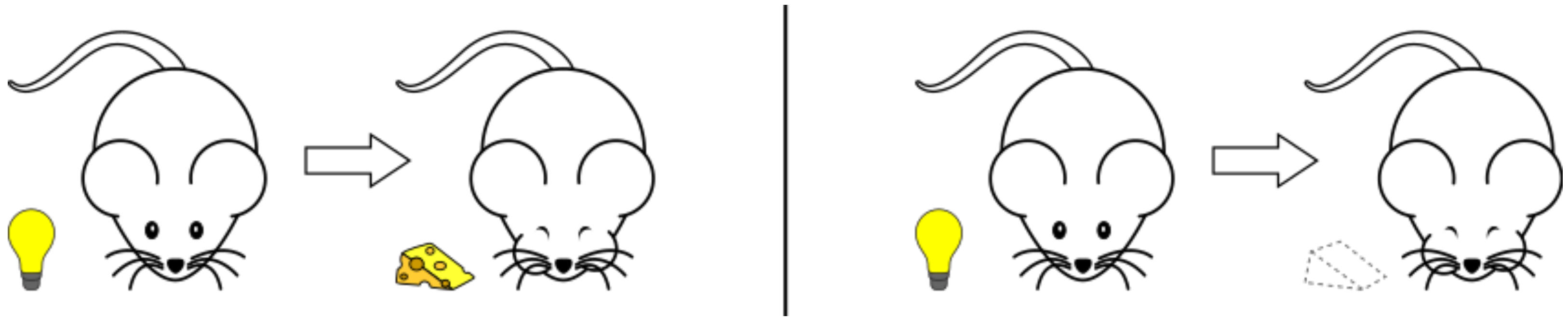
In the simplest design (forward conditioning) the CS is presented slightly before the US, so that it can serve as a signal that reward is coming

Unconditioned stimulus (US) – something inherently rewarding

Conditioned stimulus (CS) – something initially neutral

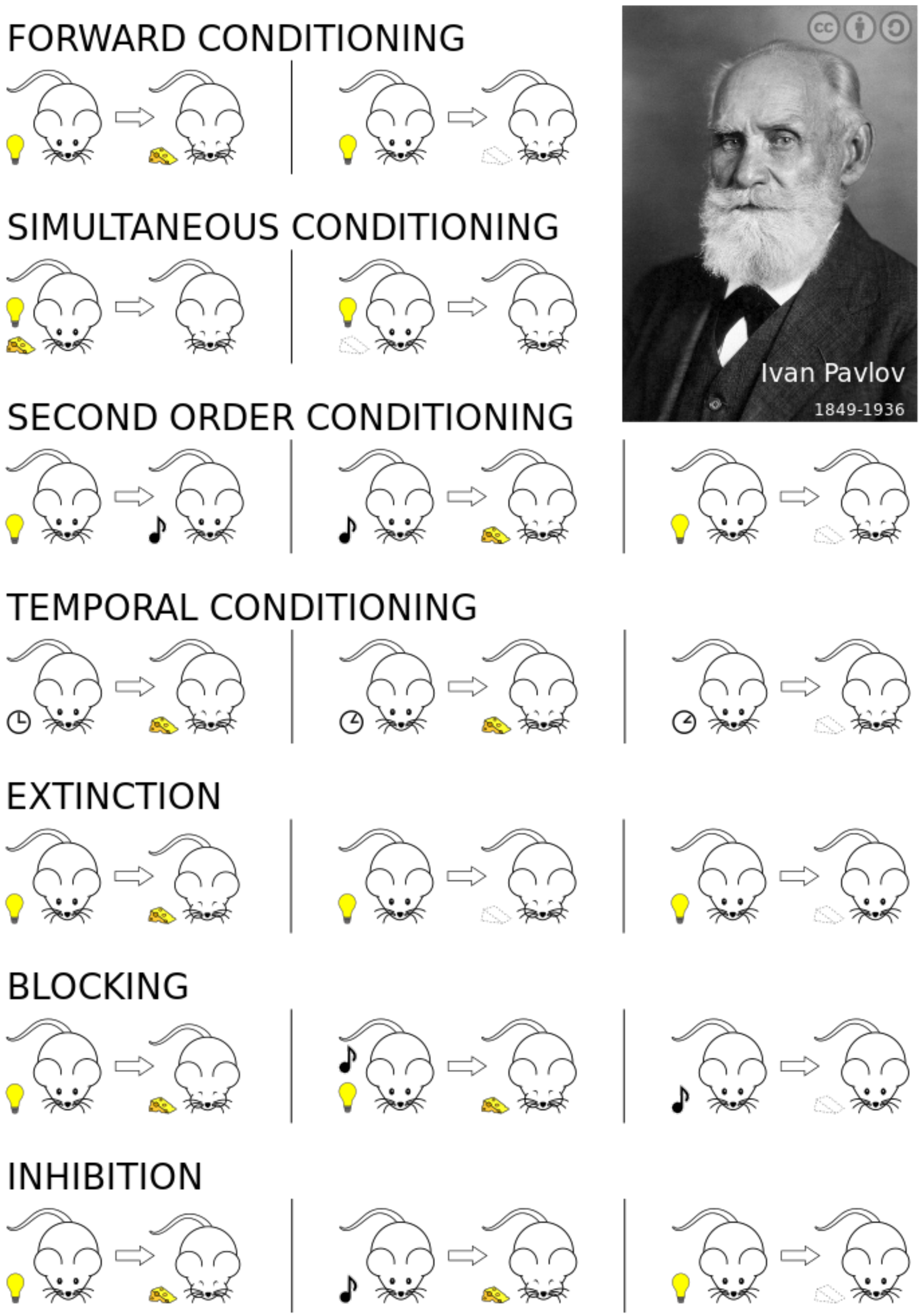
Associative learning

(well, Pavlovian anyway)



After some number of presentations, the learner starts to respond to the CS in the same way they would respond to the US

They have a learned association between the CS and the US



There are many variations on this!

(Long list of empirical effects to account for)

The Rescorla-Wagner model

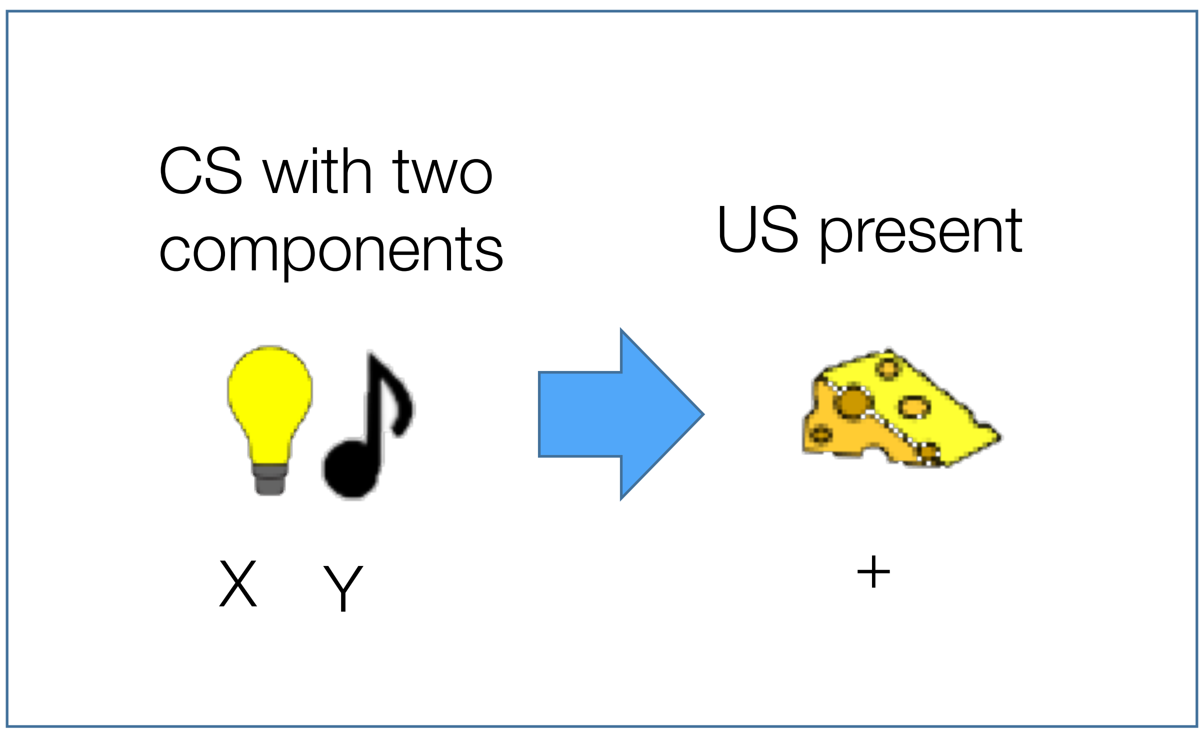
$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

One popular (though flawed & incomplete) account of associative learning is the Rescorla-Wagner model

But what does this strange inscription mean?????

The Rescorla-Wagner model

$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$



An XY+ trial

Consider a design in which there are two features present (X and Y) and the learner needs to predict an outcome that might be present (+) or absent (-)

The Rescorla-Wagner model

$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

The <u>old</u> strength of association for stimulus X

The <u>new</u> strength of association for stimulus X after seeing XY+

The difference between the old and the new. By convention “differences” are denoted “delta”, so we call this “delta-V”, ΔV

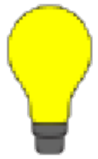
The Rescorla-Wagner model

$$\Delta V_x = \alpha\beta(\lambda - V_{xy})$$

This delta-V describes "how much we learn about X from the current trial/event"

The "alpha" and "beta" terms here are parameters describing learning rates.

- alpha depends on the CS
- beta depends on the US



The Rescorla-Wagner model

$$\Delta V_x = \alpha\beta(\lambda - V_{xy})$$

This difference term here is called
the "reward prediction error"

The Rescorla-Wagner model

$$\Delta V_x = \alpha\beta(\lambda - V_{xy})$$

lambda is represents the “intrinsic” value of the outcome (unconditioned stimulus), sometimes referred to as the “reward”, r

The Rescorla-Wagner model

$$\Delta V_x = \alpha\beta(\lambda - V_{xy})$$

V_{xy} is the “predicted reward”: the amount of reward/punishment that the learner expects to receive upon seeing the compound stimulus XY

In the Rescorla-Wagner model, expectations are additive, which means that:

$$V_{xy} = V_x + V_y$$

(But not all learning models assume additivity)

Error driven learning!

$$\Delta V_x = \alpha\beta(\lambda - V_{xy})$$

reward

expected reward

the difference between outcomes and expectations is the prediction error, and it is this error that “drives” learning

how much does the learner change their beliefs?

learning is gradual, and depends on a learning rate

Let’s do this!

Step 1: Skeleton

```
updateRW <- function(value, alpha, beta, lambda) {
}
```

$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

The design of our R function mirrors the structure of the Rescorla-Wagner model that it implements

Step 1: Skeleton

```
updateRW <- function(value, alpha, beta, lambda) {
}
```

Vector specifying associative strength between US and each element of the CS

Vector with the salience parameters

Single learning rate (since only one US presented)

Single max associability

Reminder:

$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

Step 2: Make a plan

```
updateRW <- function(value, alpha, beta, lambda) {

  # compute the value of the compound stimulus
  # compute the prediction error
  # compute the change in strength
  # update the association value
  # return the new value

}
```

Reminder:

$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

Step 3: Put in the details

```
updateRW <- function(value, alpha, beta, lambda) {

  # compute the value of the compound stimulus
  valueCompound <- sum(value)

  # compute the prediction error
  predictionError <- lambda - valueCompound

  # compute the change in strength
  valueChange <- alpha * beta * predictionError

  # update the association value
  value <- value + valueChange

  # return the new value
  return(value)
}
```

Reminder:

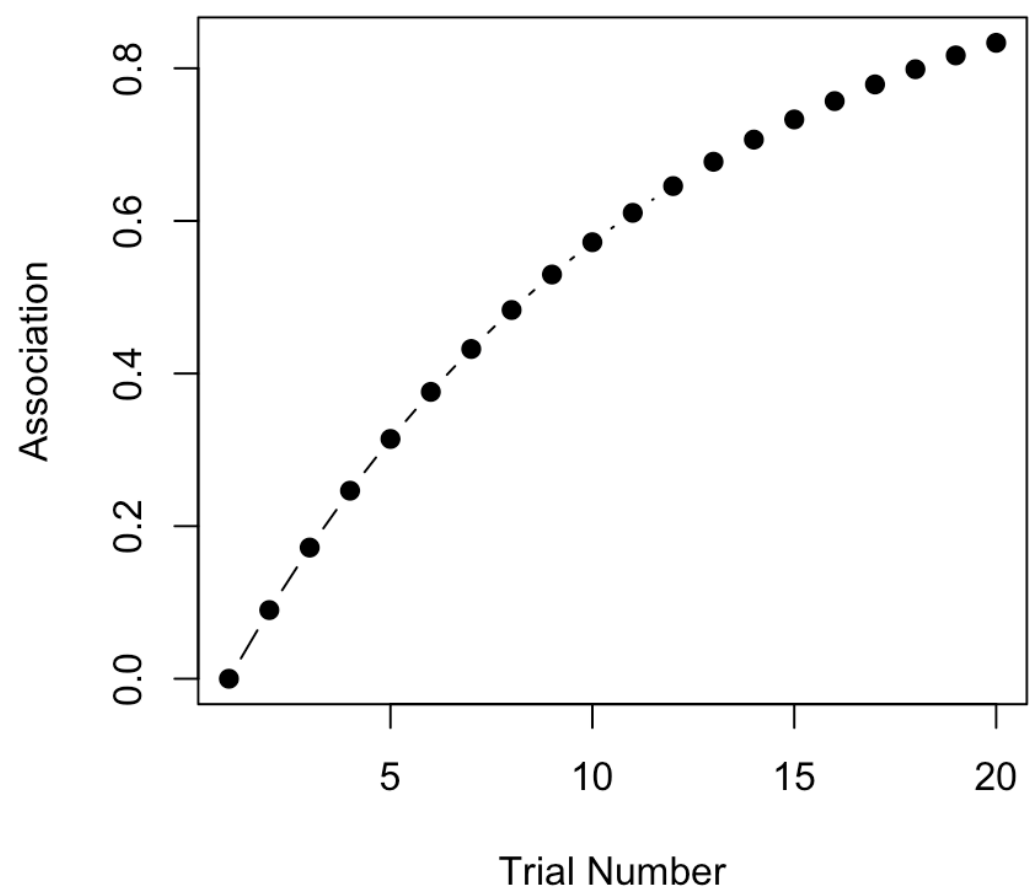
$$V_x \leftarrow V_x + \alpha\beta(\lambda - V_{xy})$$

Step 4: Model predictions

1. Conditioning
2. Extinction
3. Blocking

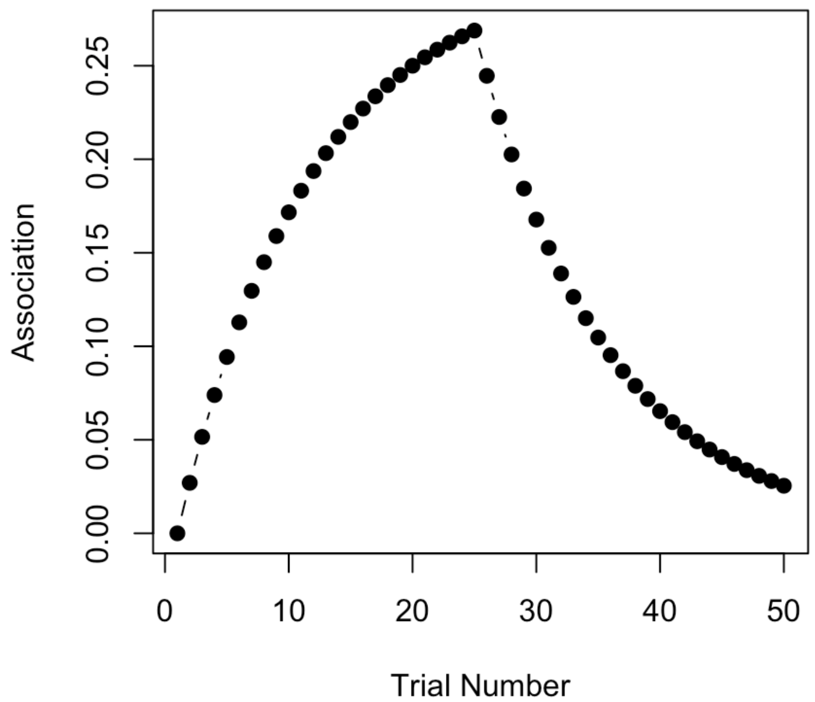
Conditioning

```
nTrials <- 20
strength <- numeric(nTrials)

for (trial in 2:nTrials) {
  strength[trial] <- updateRW(strength[trial-1])
}
```



Extinction

```
nTrials <- 50
strength <- numeric(nTrials)
lambda <- 0.3

for (trial in 2:nTrials) {

   # remove the shock after trial 25
   if (trial>25) {
     lambda <- 0
   }

   # update associative strength on each trial
   strength[trial] <- updateRW(value=strength[trial-1],
                               lambda=lambda)
}
```



Blocking

```
# total number of trials across
# both phases of the task
n_trials <- 50

# vectors of zeros
strength_A <- rep(0,n_trials)
strength_B <- rep(0,n_trials)
```

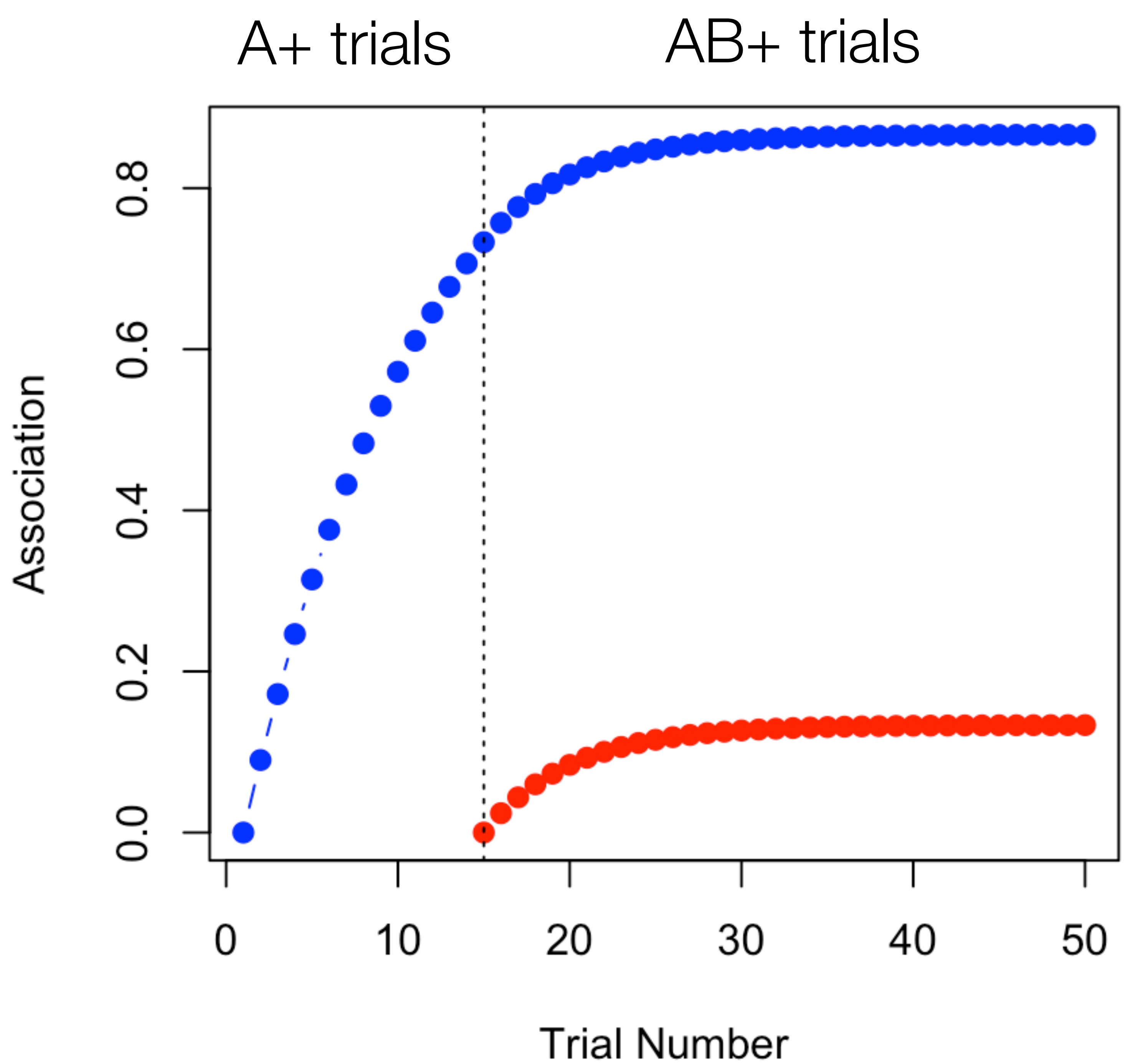
Blocking

```
# total number of trials across
# both phases of the task
n_trials <- 50

# vectors of zeros
strength_A <- rep(0,n_trials)
strength_B <- rep(0,n_trials)

# learning rate for the CS at the
# start of the experiment is .3 for
# A and 0 for B (b/c it's absent)
alpha <- c(.3, 0)
```

Blocking

```
for(trial in 2:n_trials) {

  # after trial 15, both stimuli are present
  if(trial > 15) alpha <- c(.3, .3)

  # vector of current associative strengths
  v_old <- c(strength_A[trial-1], strength_B[trial-1])

  # vector of new associative strengths
  v_new <- update_RW(
    value = v_old,
    alpha = alpha
   )

  # record the new strengths
  strength_A[trial] <- v_new[1]
  strength_B[trial] <- v_new[2]
}
```

Blocking



Strong association to A is formed early and maintained

There is learning to B, but greatly reduced and it asymptotes at a low level

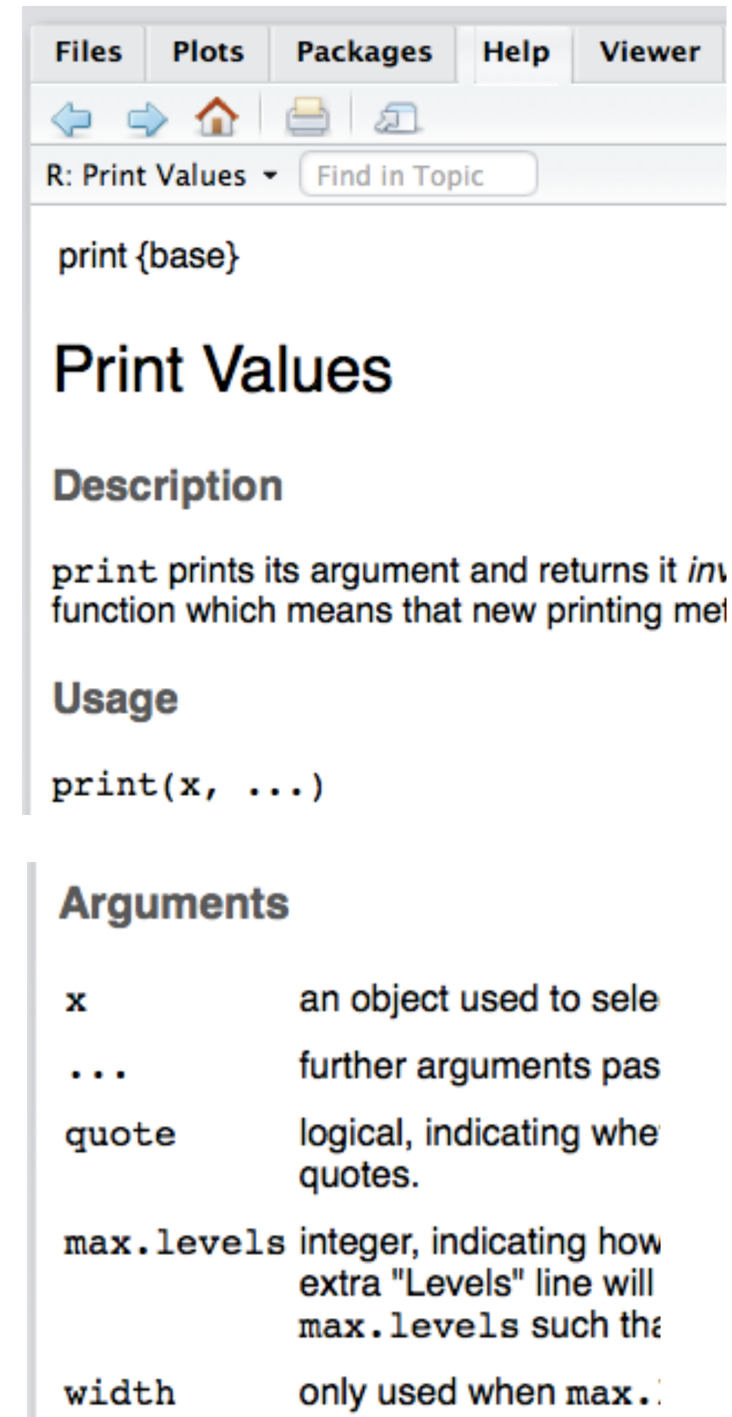
Intro to R cheat sheet

1 Saving and importing

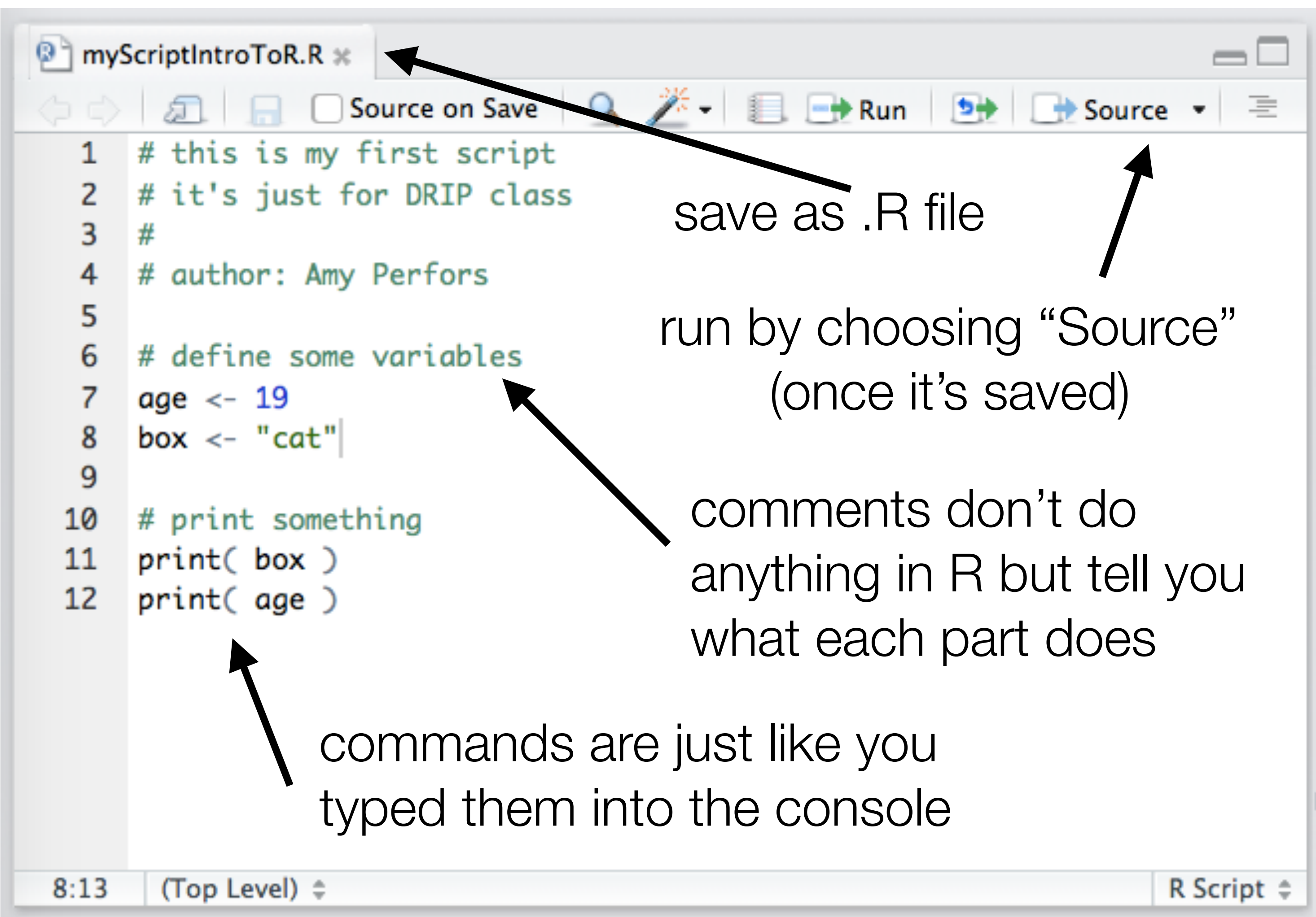
- Save as .RData, using menu or `save.image()`
- Can load .csv, using menu or `read.csv()`

13 `help(functionName)`

e.g. `help(print)`



12 Scripts let you run and save series of commands



save as .R file

run by choosing "Source" (once it's saved)

comments don't do anything in R but tell you what each part does

commands are just like you typed them into the console