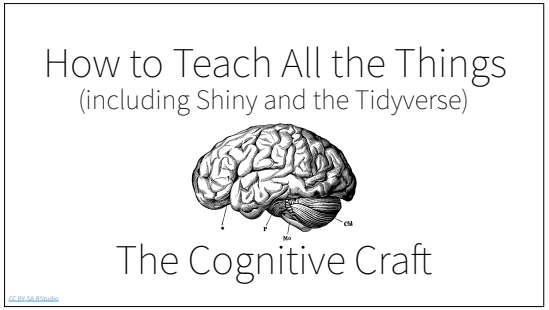
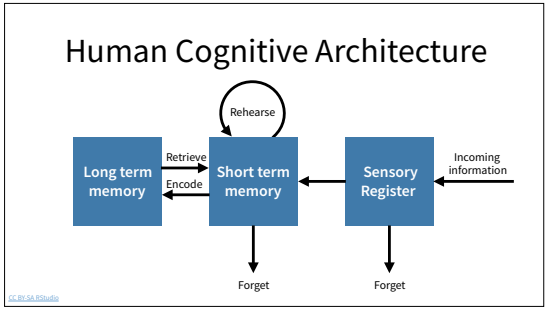


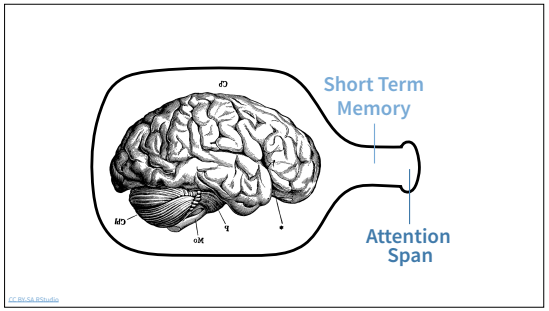
1



2

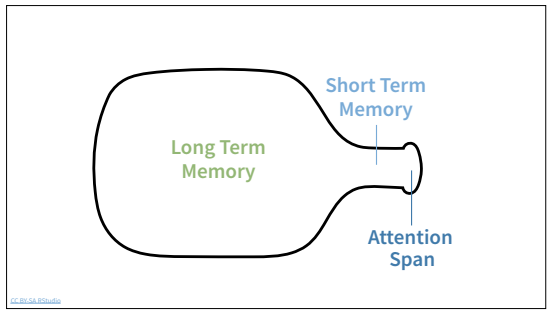
Let's come back to this simplified model of how the mind works.

- Information goes through the sensory register to short-term memory.
- Once information is in short term memory, we either rehearse it or forget it.
- If we keep it in short term memory long enough, it is **encoded** for storage, and we can (hopefully) **retrieve** it later.



3

One of the central challenges of teaching is getting information through the bottleneck of short term memory and into permanent storage.

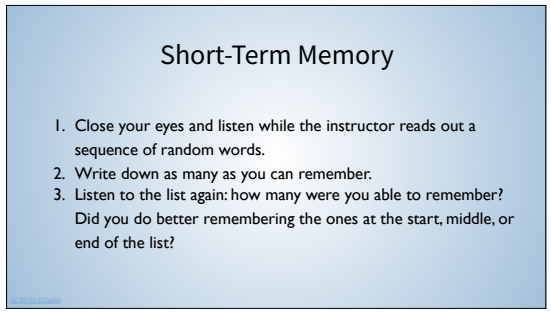


4

Your attention span acts like a filter that can screen out information. Things happen around us all of the time that we simply do not notice because we do not pay attention to them. If you've daydreamed during any part of class today then you know what I'm talking about.

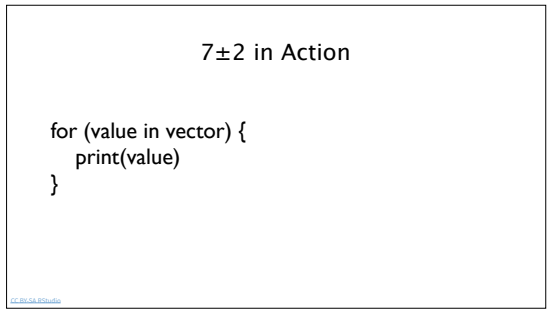
What is more important is how limited short term memory is. Initial estimates in the 1950s were that the average human being could hold 7±2 items in short term memory for a few seconds. Modern estimates put the number closer to 4±1; it only seems larger because of a phenomenon called **chunking**, whereby which things that frequently occur together (like the digits in an area code) are

---



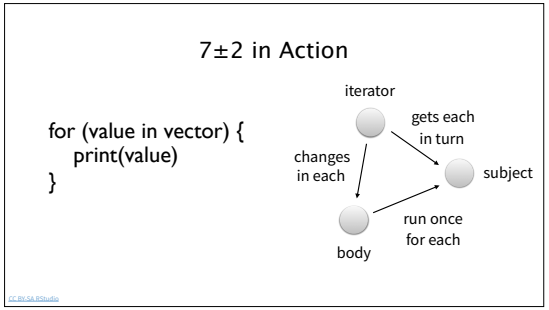
5

1. Think of a small pattern you frequently use when programming.
2. Write down an example of it.
3. Identify the related parts that you would fade when creating a set of similar examples.



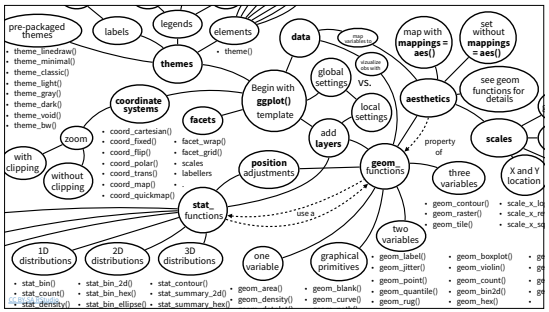
6

How does this apply to teaching? Imagine we're introducing for loops.



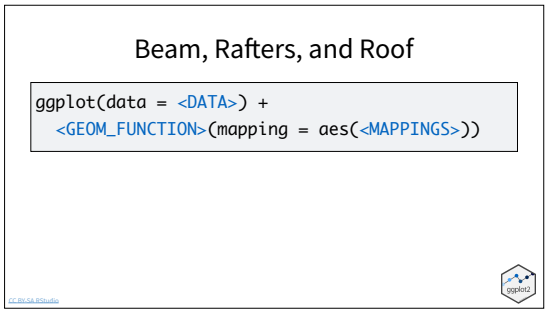
7

The concept map for this basic loop has three concepts and three links. OK, that will fit into short term memory, so we can probably teach this in a single episode. But what if we add break and next?



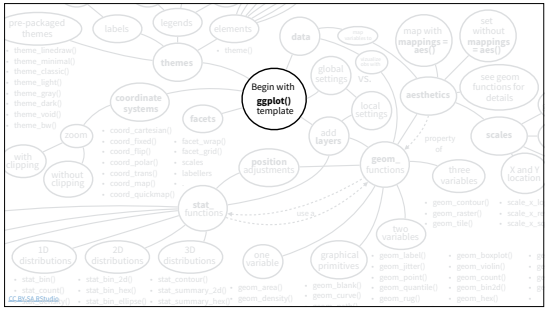
8

Or what if we try to do all of this in one pass? It will fit into our mind **because it's already there**, but it won't all fit into our learners'.



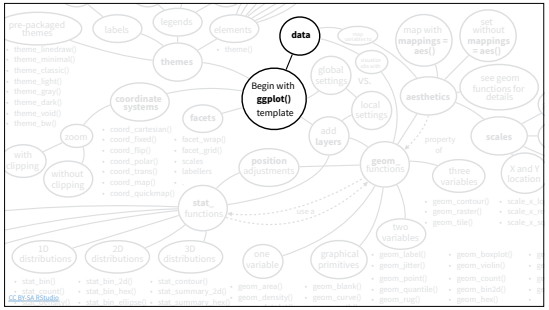
9

This is the template for ggplot2. It just about fits into short term memory.



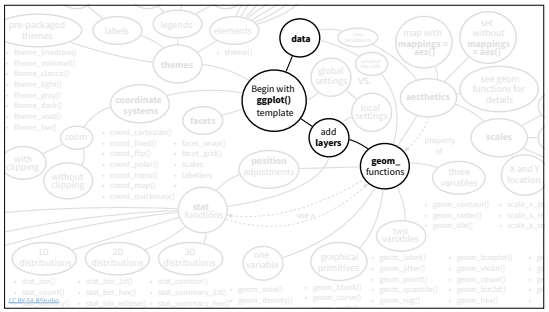
10

So when we are teaching, we start with that…



11

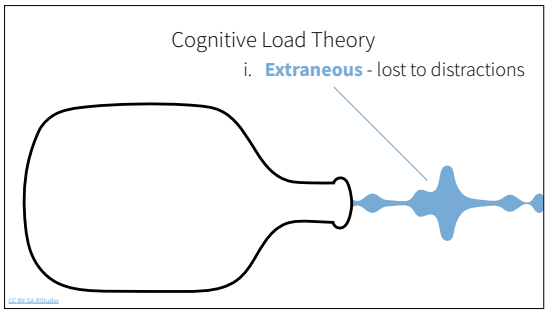
…and then attach something to it…



12

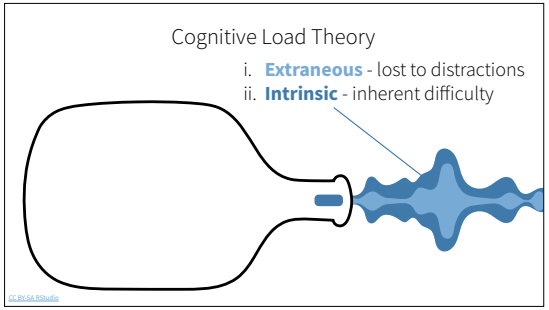
…and then attach a few more things, adding just a couple of links and nodes at a time (because learners will need to keep at least part of the concept map so far in short term memory to attach to).

There are clearly a lot of different orders in which we could do this – we'll look at some strategies for making the choice when we talk about motivation – but first we need to explore another implication of the small size of short term memory.



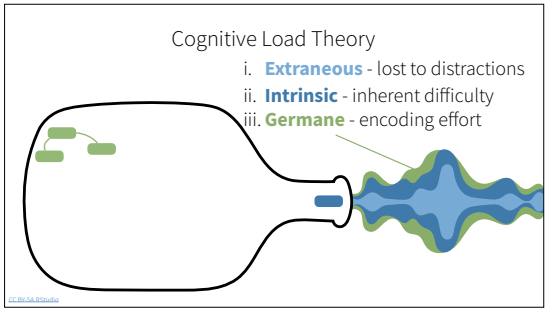
13

Cognitive load theory holds that there are three kinds of load on short term memory and processing power while we're learning. The first is **extraneous load**: what's lost to distractions and irrelevancies.



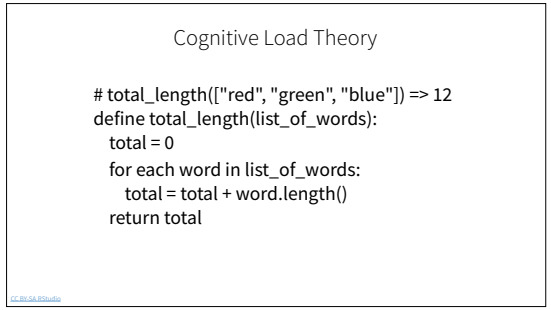
14

The second is **intrinsic load**: the inherent difficulty of the new material.



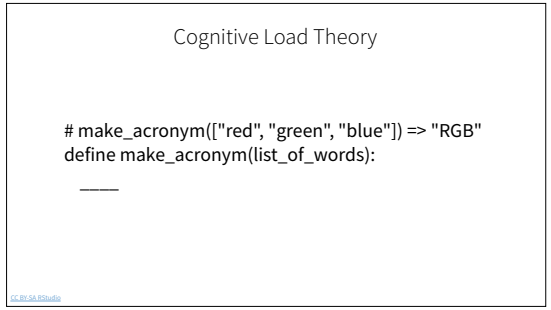
15

The third is **germane load**: what we do to encode new information and connect it with what we already know. Cognitive load theory predicts that we learn fastest when we eliminate extraneous load (no surprise) and reduce germane load as much as possible; some more recent refinements of the theory get rid of this category altogether, since encoding and making connections is as intrinsic to learning as storing new facts.



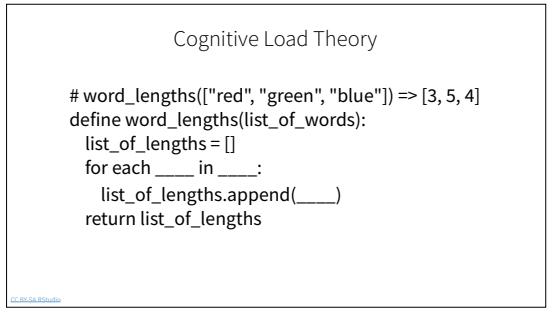
16

Here's the experiment. Everyone in a class is walked through this short piece of code, which calculates the total length of a list of words by updating an accumulator variable in a loop.



17

Half the class – the control group – is then asked to solve this problem.



18

The other half – the treatment group – is first asked to solve this problem, which fades out the controls in the loop…

Cognitive Load Theory

```
# join_all(["red", "green", "blue"]) => "redgreenblue"
define join_all(list_of_words):
  joined_words = ____
  for each ____ in ____:
    ____
  return joined_words
```


19

…then this one, which fades out the update as well, and **then** asked to solve the acronym problem. In other words, the treatment group is doing three times as many exercises as the control group. The surprising finding is that the treatment group will actually get correct solutions to the final problem faster, i.e., they'll do more total work in less total time. The reason is that this sequence of **faded examples** reduces germane load by focusing their attention on one aspect of the problem at a time. This is a generalizable finding: where there are patterns to solutions (and there almost always are), faded examples allow learners to pick up the solution strategy a step at a time, rather than trying to simultaneously figure out what to do

---

Parsons Problems

```
# data columns are (est1970, hi1970, lo1970, est1971, hi1971, lo1971, ...)
# Use this function to produce (year, est, hi, lo) for one year given the index
# of the first column.

strip <- function(data, start, year) {
  rename(estimate = 1, hi = 2, lo = 3)
  select(data, start:(start + 2))
  select(year, everything())
  mutate(year = rep(year, num_rows))
}
```


20

Parsons Problems are another application of cognitive load theory: given a jumbled set statements needed to solve a problem, put them in the right order. This is like giving someone the words to answer a sense in a foreign language – it separates vocabulary from grammar so that the learner can focus on the latter.

Create a Faded Example

1. Think of a small pattern you frequently use when programming.
2. Write down an example of it.
3. Identify the related parts that you would fade when creating a set of similar examples.

21

1. Think of a small pattern you frequently use when programming.
2. Write down an example of it.
3. Identify the related parts that you would fade when creating a set of similar examples.