So what're we learning about?

DAY 1 - Binary, binary addition, parity, the difference between error correction and detection

DAY 2 - Simple binary error detecting codes, the Luhn algorithm

DAY 3 - 2-dimensional parity check, simple set notation and finite fields

DAY 4 - Introduction to vectors and vector spaces, and why we're using them

DAY 5 - Vector subspaces, Hamming distances, and Hamming codes

DAY 6 - Q&A about all class content (or, a continuation of Day 5 content if we haven't covered it already)

Class schedule

~45 minutes: going through content on slides, asking/answering questions

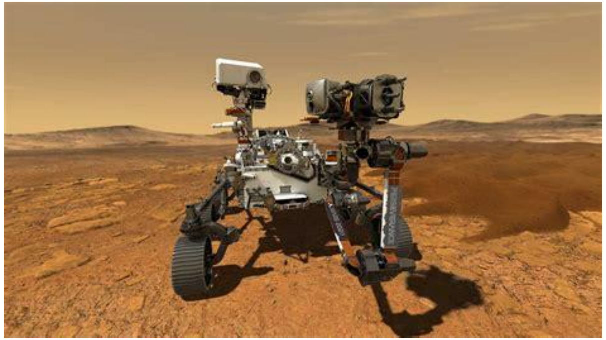
~ 30 minutes: breakout rooms with the volunteers, solving problems together

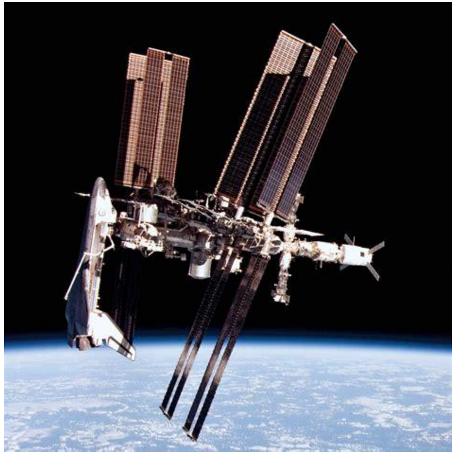
~15 minutes: short recap, answer any more questions

Office hours after class: answer questions about the class homework (which you don’t have to do, by the way), work through problems on demand

Why do we need to find / correct errors?

- sometimes transmission errors occur, and we want to be sure that we're receiving the correct message
- it's resource-intensive to send a transmission back asking for the message to be transmitted again, so using error correcting codes can be more efficient
- and why is efficiency important? when you're sending messages over a long distance it can be difficult (maybe impossible!) to resend them, or it can take a really long time





(image credits: https://mars.nasa.gov/mars2020/ and https://www.britannica.com/topic/International-Space-Station)

Error detection /correction

- **Error detection:** knowing whether there's been one or more errors, but not how many there are, where in the message they are, or how to correct them
- **Error correction:** knowing (**to some level of certainty**) how many errors have occurred, where they are, and how to correct them

"skdfjbue fbjsf ssdbje"

"ypu can' readthis"

What are binary (base-2) numbers?

16	8	4	2	1
1	1	0	1	0

(in base 2) = 16+8+2 = 26 (in base 10)

$1 + 0 = 1$

$1 + 1 = 10$

$101 + 111 = ?$

$110 + 1011 = ?$

What are binary numbers?

binary numbers are typically written with a subscript 2 (ex. 101101_2) to differentiate them from base-10 numbers.

(we'll be mostly working in base-2, though, so I won't bother with the subscripted 2s)

What's the parity of a binary number?

When referring a binary number, the **parity** describes the number of "1" bits in the number. A sequence has an odd parity if the number of "1" bits is odd, and even if the number of "1"s is even.

1011101 → 4 "1"s → even parity
1001101 → 3 "1"s → odd parity