# Code Review

* LGTM – Looks Good to Me
* Use a style guide – for everything – coding standard. not only small things, also design/architectural things. Start an empty style guide and fill it as we progress.
* Look at the style guides I started looking at in amazon.
* **Review all CRs within 1 day**.
* **Start high level and work your way down**.
  + Like: re-designing a class interface or splitting up complex functions
  + Save your low-level notes like naming and comments to next iterations
* **Be generous with code examples**. Give them gifts:
  + You might branch the code to create a larger example for the Author like breaking up a large function or adding a unit test
  + Note: reserve this for clear, uncontroversial improvements. If there is argument on if to do this, you will annoy the author and waste your time.
  + Don’t do this more then 1-2 times per review
* Never say ‘you’
  + Can **we** rename this variable to something more descriptive, **like** seconds\_remaining?
  + How about renaming this.… ? what about….?
* **Frame feedback as requests or questions**, not commands
* **Tie notes to principles, not opinions**
  + Instead of saying, “We should split this class into two,” it’s better to say, “Right now, this class is responsible for both downloading the file and parsing it. We should split it up into a downloader class and parsing class per the [single responsibility principle](https://en.wikipedia.org/wiki/Single_responsibility_principle).”
  + Provide supporting evidence where possible in the form of links. They more authoritative, the better!
* Aim to **bring the code up a letter grade or two**. Don’t try taking a D code to A.
  + The only reason to withhold approval is if the code remains at an F (with bugs or so complex that you can’t tell if there are bugs) after a few rounds of review.
* **Limit feedback on repeated patterns**. Instead say – also true throughout the CR.
* **Respect the scope of the review** – only the lines that were changed in the review, unless the CR causes an issue with the previously existing code.
* **Look for opportunities to split up large reviews**: 400 lines is usually the limit. Don’t just ask for a split, offer a logical way to do it (e.g. independent changes etc)
* **Offer sincere praise**
* **Grant approval when remaining fixes are trivial**
* **Handle stalemates proactively**.
  + Talk it out
  + Consider a design review
  + Concede or Escalate.
    - How bad the code really is – what severity issue can it cause?
    - Offer to assign another reviewer or have the manager/tech leader review.
* **Recovering from a stalemate**. Stalemates usually indicate a personal conflict that needs to be resolved.
  + Consult your manager – maybe we’re doing something wrong that causes this conflict
  + Take a break from each other
  + Study conflict resolution

# Questions are the answer

## Project Management

* What is the problem we are trying to solve?
* How do we know it is a problem?
* What does successfully solving this problem look like?
* What solution are we proposing?
  + Is there a better solution – quicker, more efficient, automatic etc?
    - Is there an existing solution we can (re)use?
    - Why aren't we doing that instead?
  + What does this solution cost? In time/people/money.
  + Can we simplify the solution?
  + Are there any parts of the solution that are not required?
* Is this solution permeant or temporary?
  + If temporary - when can we stop doing it?
* How do we track if the solution was (us)successful?
  + What do we do if it’s unsuccessful?
  + What is our contingency plan?
* How do we track this solution?
* Milestones for this solution

## Interrupting Bias

Interrupting Biases:

* How are personal experiences, beliefs or cultural and social norms shaping my perspective?
* Have I sought out new/different perspective before making this decision?
* Did I allow enough time to gather and consider all data versus making the quickest/easiest decisions?
* Am I taking calculated risks or avoiding loss/conflict?