Getting ready for remote pairing

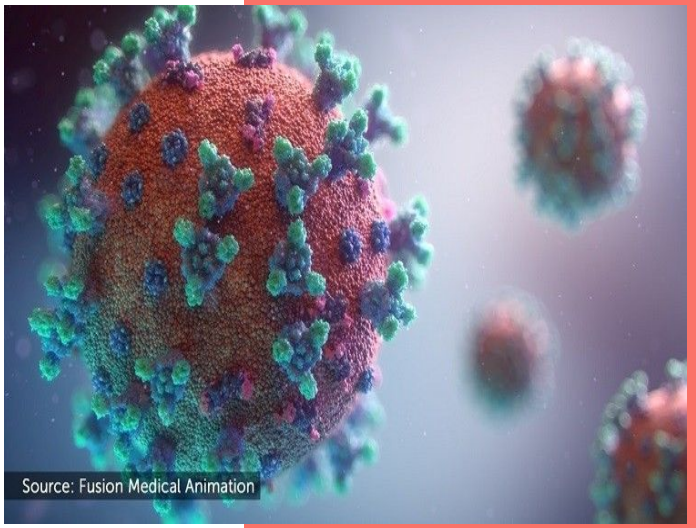
Setup

Some useful tools:

- Zoom client
  - Video Conference software
- Git
  - Distributed Version control system.
    Have it ready!
- NPM
  - Package manager for Javascript
  - Nvm for Npm version management
- IDE: Webstorm/VSCode (ideally the first one)
  - Integrated development environment

Why do we need them?

Since COVID-19 happened, our ways of working have adapted to the “new normal”. In this scenario, pairing remotely could be very difficult and different. For that reason, our suggestion is to make it through zoom breakout rooms, while the handover will be done strictly through git



codurance

Human interaction through Zoom

Zoom is a video conference software, somewhat used in years prior to the COVID pandemic. After the lockdowns and remote working exploded, its usage went to the roof. Right now, it’s the cornerstone of communication for several teams across the globe, and will be our for the following weeks. Get used to it, explore “Annotations” and “Remote Control”. Quite good for pairing

Handing over to your partner: Git

If you've been living in Earth for the last 10 years, and been working on Software development within the last 5 years at least, then you've probably heard or used Git, a distributed version control system for which there are plenty of providers, Github being one of them. We're going to use git to hand over the work to our pairing partner.

SDKMan to the rescue!

SDKMan is a tool for managing parallel versions of multiple Software Development Kits on most Unix based systems.

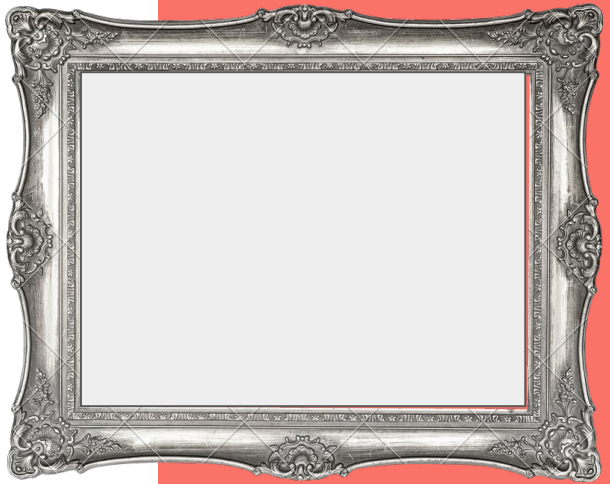
It provides a convenient CLI and API for installing, switching, removing and listing candidates.

It was inspired by tools like RVM or rbenv. Just follow the installation instructions and you're pretty much ready to go.

IDE: Use whatever you feel more productive with

We strongly believe that you should use the best tools available and, at the same time, use those you feel more comfortable and productive working with. What's necessary for this course is a tool that can handle projects, syntax highlight and most importantly, REFACTORS! If your tool does not offer easy refactors, reconsider using one that does it.

Extra tools: Use for silver plating your exercises



We consider that keeping the katas we do to revisit them in the future is a great way to internalize all the concepts. These katas may even be helpful to start conversations in your teams or in the community so we offer this tools to silver plate them:

- www.githistory.xyz
- Red/Green/Refactor functions

```
function gwip {  git add -A; git commit -m🟢🟢🟢🟢 GREEN - $1 - to squash" }

function rwip {  git add -A; git commit -m🔴🔴🔴🔴 RED - $1 - to squash” }

function fwip {  git add -A; git commit -m👍👍👍👍 REFACTOR - $1 - to squash" }

function dwip {  git add -A; git commit -m📖 📖 📖  DOCUMENT - $1 - to squash" }
```

Fizz Buzz kata

Write a function that takes numbers from 1 to 100 and outputs them as a string i.e "4", but for multiples of three returns "Fizz" instead of the number and for the multiples of five returns "Buzz". For numbers which are multiples of both three and five returns "FizzBuzz".

Notes: Get familiar with the IDE and the options it provides if you haven't already. Get together with your partner to explore Zoom annotations, remote control. Alternate with your partner and handover code through Git

Leap year kata

Write a function that returns true or false depending on whether its input integer is a leap year or not.
A leap year is defined as one that is divisible by 4, but is not otherwise divisible by 100 unless it is also divisible by 400.
For example, 2001 is a typical common year and 1996 is a typical leap year, whereas 1900 is an atypical common year and 2000 is an atypical leap year.

Notes: We will try a MOB programming session where we'll be all coding on one station at a time, handing over the code. Let's try to setup https://github.com/remotemobprogramming/mob so we can make this easier