From Notebooks to Packages: Object Oriented Programming

An introduction to writing well-formed modular code to facilitate collaborative programming

What makes good code?

- Easy to read/follow/understand
  - "Encapsulation": bundle code into discrete units with clear scope
- Generalized for broader applications
  - "Abstraction"
- Not redundant, functions only defined in one place
  - "Inheritance" don't re-invent the wheel
- Customizable, can override behaviours to suit our needs
  - "polymorphism"

Object Oriented Programming (OOP)

OOP → Modularises code into chunks or "objects" which are data field that has unique attributes & behaviours

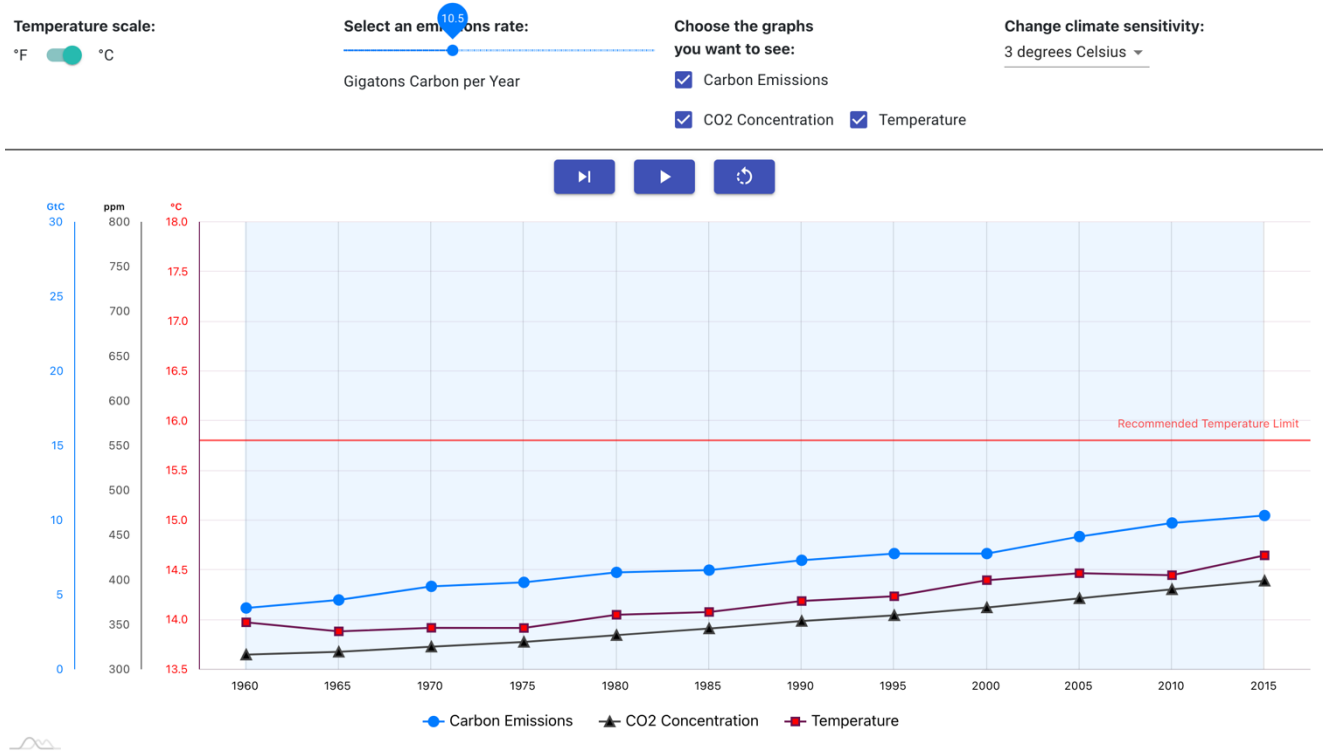
A very simple climate model

https://scied.ucar.edu/interactive/simple-climate-model

$$T = T_0 + S \log_2\left(\frac{C}{C_0}\right)$$

- T: new/current temp
- T_0: known temp at some reference time
- S: climate sensitivity factor
- C: current CO_2 concentration
- C_0: known CO_2 concentration at reference year

S = temperature rise as a result of CO_2 doubling



Class: VerySimpleClimateModel

Has

- reference year
- reference temp
- reference $[CO_2]$
- climate sensitivity
- emission scheme

Does

- run
- display / plot

CO_2 emission scheme

Constant emission rate

Has
- c_0
- t_0
- scheme = "constant"

Does
- get ppm at year
- print

SSP Scenario

has
- scheme
- year
- $[CO_2]$

does
- read from csv
- get ppm at year
- print

↑ polymorphism

Aside: How VCSM handles CO_2 emissions

VCSM

- every 2.3 GtC ↑ atm $[CO_2]$ by 1 ppm
- assumes 0.1% loss / year & 55% absorbed by ocean

$$\Rightarrow C_n = k C_{n-1} + a$$

$$= k^n C_0 + a\left(\frac{1-k^n}{1-k}\right), \quad k = 0.999$$

$$a = \frac{\text{rate}(0.45)}{2.3}$$

BaseCO2

Has

Does

Emission scheme → separate class = "Encapsulation"

BaseCO2 → "Abstraction"

has - scheme type

does: gets [CO2] at given year

↙ constant emissions

↘ SSP "Inheritance"