```
(
// steady state: a study in taxation (2019)
// written in SuperCollider version 3.9.0

~calculateWealthGap = {
	// ~~~~~~~~~input variables~~~~~~~~~~~~~~
	// number of economy participants
	// population matters, the more participants the more normalized the results
	arg populationSize = 100,

	// type of init wealth distribution
	// this is a sensitive initial condition
	// options:
	//   equal  =  everybody starts the same financially
	//   linear =  wealth is distributed from poor to rich linearly
	//   random =  everybody is given a random amount of money
	initWealthDist = 'equal',

	// number of iterations / tax years
	taxYears = 500,

	// number of economic transactions per tax year
	tradesPerYear = 500,

	// a debt limit above 0 does not allow debt, below zero debt is allowed
	debtLimit = 0,

	// a difference between the minimum tax (be it income or wealth) and the corresponding maximum tax is equivalent to a progressive tax
	// that progresses from the minimum to the maximum according to the corresponding tax curve
	minWealthTax = 0, maxWealthTax = 0, wealthTaxCurve = 0,

	// a negative tax curve of 0 means that the collected taxes are redistributed equally
	// values below 0 will result in the poor getting relatively more
	negativeIncomeTaxCurve = 0, negativeWealthTaxCurve = 0;

	// ~~~~~~~~~model variables~~~~~~~~~~~~~~
	var wealthPerCapita, incomePerCapita,
	trader1, trader2,
	incomeTaxPerCapita, wealthTaxPerCapita,
	collectedIncomeTaxes, collectedWealthTaxes;

	// initialize the net worth of the population in an array
	wealthPerCapita = populationSize.collect({arg i;
		switch(initWealthDist, 'equal', {100}, 'linear', {i}, 'random', {100.rand})});

	// run the model
	taxYears.do({
		// reset income to 0 at the beginning of every iteration
		incomePerCapita = populationSize.collect({0});


		// ~~~~~~~~~toy model of an economy~~~~~~~~~~~~~~
		// completely based on fair trade and fixed resources (no inflation)
		// e.g. pay for a good what it costs to make that good, which is a best case scenario
		// the richer are more likely to make transactions
		tradesPerYear.do({
			trader1 = (0..populationSize).wchoose(
				wealthPerCapita.thresh(debtLimit).normalizeSum);
			trader2 = (0..populationSize).wchoose(
				wealthPerCapita.copy.put(trader1, 0).thresh(debtLimit).normalizeSum);
			wealthPerCapita[trader1] = wealthPerCapita[trader1] + 1;
			incomePerCapita[trader1] = incomePerCapita[trader1] + 1;
			wealthPerCapita[trader2] = wealthPerCapita[trader2] - 1;
			incomePerCapita[trader2] = incomePerCapita[trader2] - 1;
		});


		// ~~~~~~~~~toy model of income taxation~~~~~~~~~~~~~~
		// calculate the income tax for each individual
		incomeTaxPerCapita = incomePerCapita.thresh(0) * pow(
			incomePerCapita.thresh(0), incomeTaxCurve).normalize(minIncomeTax, maxIncomeTax);
		// levy the tax
		wealthPerCapita = wealthPerCapita - incomeTaxPerCapita;
		// sum the taxes
		collectedIncomeTaxes = incomeTaxPerCapita.sum;


		// ~~~~~~~~~toy model of wealth taxation~~~~~~~~~~~~~~
		// calculate the wealth tax for each individual
		// note that this is per tax year
		// a more sophisticated model could track this from generation to generation
		// the results should be similar
		wealthTaxPerCapita = wealthPerCapita.thresh(0) * pow(
			wealthPerCapita.thresh(0), wealthTaxCurve).normalize(minWealthTax, maxWealthTax);
		// levy the tax
		wealthPerCapita = wealthPerCapita - wealthTaxPerCapita;
		// sum the taxes
		collectedWealthTaxes = wealthTaxPerCapita.sum;

```

```
90
91          // ~~~~~~~~~redistribute the taxes~~~~~~~~~~~~~~
92          wealthPerCapita = wealthPerCapita + (collectedIncomeTaxes * pow(
93               incomePerCapita.thresh(0) + 1, negativeIncomeTaxCurve).normalizeSum);
94          wealthPerCapita = wealthPerCapita + (collectedWealthTaxes * pow(
95               wealthPerCapita.thresh(0) + 1, negativeWealthTaxCurve).normalizeSum);
96     });
97
98     // ~~~~~~~~~plot and return results~~~~~~~~~~~~~~
99     // plot resulting wealth gap by sorting the population from poorest to richest
100    // then show a sample of the curve from low to high numerically
101    wealthPerCapita = wealthPerCapita.sort;
102    wealthPerCapita.plot;
103    10.collect({arg i; wealthPerCapita[(wealthPerCapita.size/10) * i].trunc}) ++ [wealthPerCapita.last.trunc];
104}
105)
106
107// a few examples
108// excecute the code block above (delimited by the outermost parenthesis)
109// define and execute the taxation models in the code blocks below to run the model and see particular results
110(
111// no taxes whatsoever
112~calculateWealthGap.value(
113     initWealthDist: 'equal',
114     minIncomeTax: 0, maxIncomeTax: 0,
115     minWealthTax: 0, maxWealthTax: 0)
116)
117// example result:
118// -> [ 0, 11, 20, 38, 56, 72, 96, 122, 165, 241, 460 ]
119
120(
121// non-progressive income tax only; flat 25% rate
122~calculateWealthGap.value(
123     initWealthDist: 'equal',
124     minIncomeTax: 0.25, maxIncomeTax: 0.25)
125)
126// example result:
127// -> [ 31, 51, 59, 71, 85, 95, 105, 118, 133, 155, 273 ]
128
129(
130// progressive income tax; linear 25% to 50% rate
131~calculateWealthGap.value(
132     initWealthDist: 'equal',
133     minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 1)
134)
135// example result:
136// -> [ 44, 62, 73, 84, 89, 94, 100, 114, 127, 148, 182 ]
137
138(
139// progressive income tax with a negative income tax curve benefitting the poor
140~calculateWealthGap.value(
141     initWealthDist: 'equal',
142     minIncomeTax: 0.25, maxIncomeTax: 0.5,
143     negativeIncomeTaxCurve: -1)
144)
145// example result:
146// -> [ 46, 62, 75, 88, 93, 98, 103, 112, 121, 135, 199 ]
147
148(
149// very progressive income tax with with a negative income tax curve strongly benefiting the poor
150// starting from a wealth gap to show that it is not self-correcting
151~calculateWealthGap.value(
152     initWealthDist: 'linear',
153     minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 3,
154     negativeIncomeTaxCurve: -2)
155)
156// example result:
157// -> [ 15, 23, 29, 33, 39, 43, 51, 58, 66, 81, 139 ]
158
159// ~~~~~~~~~conclusion~~~~~~~~~~~~~~
160/*
161regardless of any income tax structure, a steady state is reached with a wealth gap.
162at best, progressive taxes and negative taxes (basic income) minimize the wealth gap.
163none of these models based only on income taxes are self-correcting.
164
165response:
166) level the playing field with a one-time total wealth redistribution
167) levy a wealth tax such that individuals can primarily accrue wealth in their lifetime
168) maintain a progressive income tax and basic income to limit the wealth gap within a generation
169*/
170(
171// progressive income and wealth tax with negative tax (basic income)
172~calculateWealthGap.value(
173     initWealthDist: 'linear',
174     minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 1,
175     minWealthTax: 0.25, maxWealthTax: 0.5, wealthTaxCurve: 1,
176     negativeIncomeTaxCurve: -1)
177)
178// example result:
179// -> [ 46, 48, 48, 49, 49, 49, 49, 50, 50, 50, 50 ]
180
```

```
// ~~~~~~~~~~~~~~~~~~~~~ performance notes / instructions ~~~~~~~~~~~~~~~~~~~~~~~~~~
// This code was translated to dutsch by Samuel Vriezen for publication in vol. 38 of nY literary review
// and subsequently performed with Dante Boon.
// For a performance, inspired by Vriezen with Dante Boon, the code can be simply read aloud
// (potentially with sparse musical accompaniment / interpretations).
```

steady state: a study in taxation

```
(
// steady state: a study in taxation (2019)
// escrito en SuperCollider versión 3.9.0

~calculateWealthGap = {
     // ~~~~~~~~~variables de entrada~~~~~~~~~~~~~~~
     // número de participantes en la economía
     // la población importa, cuantos más participantes, más normalizados serán los resultados
     arg populationSize = 100,

     // tipo de distribución inicial de la riqueza
     // ésta es una condición inicial sensible
     // opciones:
     //    equal significa que todos comienzan igual financieramente
     //    linear significa que la riqueza se distribuye de pobre a rico linealmente
     //    random significa que todos reciben una cantidad aleatoria de dinero
     initWealthDist = 'equal',

     // número de iteraciones / ejercicios fiscales
     taxYears = 500,

     // número de transacciones económicas por ejercicio fiscal
     tradesPerYear = 500,

     // un límite de deuda superior a cero no permite el endeudamiento;
     // los valores inferiores a cero permiten el endeudamiento
     debtLimit = 0,

     // una diferencia entre el impuesto mínimo (ya sea sobre la renta o sobre la riqueza)
     // y el impuesto máximo correspondiente equivale a un impuesto progresivo
     // que progresa del mínimo al máximo según la curva impositiva correspondiente
     minIncomeTax = 0.25, maxIncomeTax = 0.5, incomeTaxCurve = 1,
     minWealthTax = 0, maxWealthTax = 0, wealthTaxCurve = 0,

     // una curva de impuestos negativa de 0 significa que los impuestos recaudados se redistribuyen equitativamente
     // valores inferiores a 0 harán que los pobres reciban relativamente más
     negativeIncomeTaxCurve = 0, negativeWealthTaxCurve = 0;

     // ~~~~~~~~~variables del modelo~~~~~~~~~~~~~~~
     var wealthPerCapita, incomePerCapita,
     trader1, trader2,
     incomeTaxPerCapita, wealthTaxPerCapita,
     collectedIncomeTaxes, collectedWealthTaxes;

     // inicializar la riqueza neta de la población en una matriz
     wealthPerCapita = populationSize.collect({arg i;
          switch(initWealthDist, 'equal', {100}, 'linear', {i}, 'random', {100.rand})});

     // ejecute el modelo
     taxYears.do({
          // restablecer el ingreso a 0 al principio de cada iteración
          incomePerCapita = populationSize.collect({0});


          // ~~~~~~~~~modelo de juguete de una economía~~~~~~~~~~~~~~~
          // totalmente basado en el comercio justo y los recursos fijos (sin inflación)
          // por ejemplo, pagar por un producto lo que cuesta fabricar ese producto, es lo mejor que se puede hacer
          // los más ricos tienen más probabilidad de hacer transacciones
          tradesPerYear.do({
               trader1 = (0..populationSize).wchoose(
                    wealthPerCapita.thresh(debtLimit).normalizeSum);
               trader2 = (0..populationSize).wchoose(
                    wealthPerCapita.copy.put(trader1, 0).thresh(debtLimit).normalizeSum);
               wealthPerCapita[trader1] = wealthPerCapita[trader1] + 1;
               incomePerCapita[trader1] = incomePerCapita[trader1] + 1;
               wealthPerCapita[trader2] = wealthPerCapita[trader2] - 1;
               incomePerCapita[trader2] = incomePerCapita[trader2] - 1;
          });


          // ~~~~~~~~~modelo de juguete del impuesto sobre la renta~~~~~~~~~~~~~~~
          // calcular el impuesto sobre la renta de cada individuo
          incomeTaxPerCapita = incomePerCapita.thresh(0) * pow(
               incomePerCapita.thresh(0), incomeTaxCurve).normalize(minIncomeTax, maxIncomeTax);
          // recaudar el impuesto
          wealthPerCapita = wealthPerCapita - incomeTaxPerCapita;
          // sumar los impuestos
          collectedIncomeTaxes = incomeTaxPerCapita.sum;


          // ~~~~~~~~~modelo de juguete del impuesto sobre la riqueza~~~~~~~~~~~~~~~
          // calcular el impuesto sobre la riqueza para cada individuo
          // teniendo en cuenta que esto es por año fiscal
          // un modelo más sofisticado podría seguir esto de generación en generación
          // los resultados deberían ser similares
```

```
            wealthTaxPerCapita = wealthPerCapita.thresh(0) * pow(
                wealthPerCapita.thresh(0), wealthTaxCurve).normalize(minWealthTax, maxWealthTax);
            // recaudar el impuesto
            wealthPerCapita = wealthPerCapita - wealthTaxPerCapita;
            // sumar los impuestos
            collectedWealthTaxes = wealthTaxPerCapita.sum;


            // ~~~~~~~~~redistribuir los impuestos~~~~~~~~~~~~~~~
            wealthPerCapita = wealthPerCapita + (collectedIncomeTaxes * pow(
                incomePerCapita.thresh(0) + 1, negativeIncomeTaxCurve).normalizeSum);
            wealthPerCapita = wealthPerCapita + (collectedWealthTaxes * pow(
                wealthPerCapita.thresh(0) + 1, negativeWealthTaxCurve).normalizeSum);
        });

        // ~~~~~~~~~graficar y devolver resultados~~~~~~~~~~~~~~~
        // trazar la brecha de riqueza resultante ordenando la población de la más pobre a la más rica
        // a continuación, muestre una porción de la curva de menor a mayor numéricamente
        wealthPerCapita = wealthPerCapita.sort;
        wealthPerCapita.plot;
        10.collect({arg i; wealthPerCapita[(wealthPerCapita.size/10) * i].trunc}) ++ [wealthPerCapita.last.trunc];
})

// algunos ejemplos
// ejecute el bloque de código anterior (delimitado por el paréntesis exterior)
// defina y ejecute los modelos de imposición en los bloques de código siguientes
// para ejecutar el modelo y ver los resultados particulares
(
// sin impuestos de ningún tipo
~calculateWealthGap.value(
        initWealthDist: 'equal',
        minIncomeTax: 0, maxIncomeTax: 0,
        minWealthTax: 0, maxWealthTax: 0)
)
// resultado de ejemplo:
// -> [ 0, 11, 20, 38, 56, 72, 96, 122, 165, 241, 460 ]

(
// impuesto solo sobre la renta no progresivo; tipo fijo del 25%
~calculateWealthGap.value(
        initWealthDist: 'equal',
        minIncomeTax: 0.25, maxIncomeTax: 0.25)
)
// resultado de ejemplo:
// -> [ 31, 51, 59, 71, 85, 95, 105, 118, 133, 155, 273 ]

(
// impuesto solo sobre la renta progresivo; tipo lineal del 25% al 50%.
~calculateWealthGap.value(
        initWealthDist: 'equal',
        minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 1)
)
// resultado de ejemplo:
// -> [ 44, 62, 73, 84, 89, 94, 100, 114, 127, 148, 182 ]

(
// Impuesto sobre la renta progresivo con renta negativa para beneficiar a los pobres
~calculateWealthGap.value(
        initWealthDist: 'equal',
        minIncomeTax: 0.25, maxIncomeTax: 0.5,
        negativeIncomeTaxCurve: -1)
)
// resultado de ejemplo:
// -> [ 46, 62, 75, 88, 93, 98, 103, 112, 121, 135, 199 ]

(
// impuesto sobre la renta muy progresivo con una curva negativa del impuesto sobre la renta
// que beneficia fuertemente a los pobres partiendo de una brecha de riqueza para demostrar que no se autocorrige
~calculateWealthGap.value(
        initWealthDist: 'linear',
        minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 3,
        negativeIncomeTaxCurve: -2)
)
// resultado de ejemplo:
// -> [ 15, 23, 29, 33, 39, 43, 51, 58, 66, 81, 139 ]

// ~~~~~~~~~conclusión~~~~~~~~~~~~~~~
/*
independientemente de cualquier estructura del impuesto sobre la renta,
se alcanza un estado estacionario con una brecha de riqueza. en el mejor de los casos,
los impuestos progresivos y los impuestos negativos (renta básica) minimizan la brecha de riqueza.
ninguno de estos modelos basados únicamente en el impuesto sobre la renta se autocorrige.

respuesta:
1) nivelar el campo de juego con una redistribución total de la riqueza de una sola vez
2) aplicar un impuesto sobre la riqueza para que los individuos puedan acumular riqueza
   principalmente a lo largo de su vida
```

```
3) mantener un impuesto progresivo sobre la renta y una renta básica para limitar
   la brecha de riqueza en el plazo de una generación
*/
(
// impuesto progresivo sobre la renta y la riqueza con impuesto negativo (renta básica)
~calculateWealthGap.value(
     initWealthDist: 'linear',
     minIncomeTax: 0.25, maxIncomeTax: 0.5, incomeTaxCurve: 1,
     minWealthTax: 0.25, maxWealthTax: 0.5, wealthTaxCurve: 1,
     negativeIncomeTaxCurve: -1)
)
// resultado de ejemplo:
// -> [ 46, 48, 48, 49, 49, 49, 49, 50, 50, 50, 50 ]

// ~~~~~~~~~~~~~~~~~~~~~ performance notes / instructions ~~~~~~~~~~~~~~~~~~~~~~~~~~
// Este código fue traducido al dutsch por Samuel Vriezen para su publicación en el vol. 38 de nY literary review
// y posteriormente interpretado con Dante Boon.
// Para una performance, inspirada por Vriezen con Dante Boon, el código puede ser simplemente leído en voz alta
// (potencialmente con escaso acompañamiento musical / interpretaciones).
```