Домашнее задание к лекции 4.2 «Генераторы»

Перед началом работы

1.Активируйте строгий режим соответствия.

2.Добавьте код в редактор:

```
function hslToRgb(h, s, l) {
  let r, g, b;

  if(s == 0) {
    r = g = b = l; // achromatic
  } else {
    const hue2rgb = function (p, q, t) {
      if(t < 0) t += 1;
      if(t > 1) t -= 1;
      if(t < 1/6) return p + (q - p) * 6 * t;
      if(t < 1/2) return q;
      if(t < 2/3) return p + (q - p) * (2/3 - t) * 6;
      return p;
    };

    const q = l < 0.5 ? l * (1 + s) : l + s - l * s;
    const p = 2 * l - q;
    r = hue2rgb(p, q, h + 1/3);
    g = hue2rgb(p, q, h);
    b = hue2rgb(p, q, h - 1/3);
  }

  function colorToHex(color) {
    let hex = Math.round(color * 255).toString(16);
    return hex.length < 2 ? `0${hex}` : hex;
  }

  const color = [r, g, b].map(colorToHex).join('');
  return `#${color}`;
}
```

```
class Order {
  constructor(id, weight) {
    this.id = id;
    this.weight = weight;
  }
}

class Truck extends Array {
  constructor(number, weightLimit) {
    super();
    this.number = number;
    this.weightLimit = weightLimit;
  }

  add(order) {
    if (!this.isFit(order)) {
      return false;
    }
    this.push(order);
    return true;
  }

  isFit(order) {
    return this.weightLimit >= (this.totalWeight + order.weight);
  }

  get totalWeight() {
    return this.reduce((total, order) => total + order.weight, 0);
  }

  show() {
    console.log(`Машина №${this.number} (общий вес груза $
{this.totalWeight}кг):`);
    this.forEach(order => console.log(`\tЗаказ #${order.id} $
{order.weight}кг`));
  }
}
```

Задача № 1. Палитра для сайта

Мы хотим для каждого пользователя выдавать наш сайт в уникальных цветах. И первое, что для этого нужно сделать — это реализовать генератор цветовой палитры с аналогичными цветами.

Для этого вам потребуется реализовать генератор `palette`, который принимает количество цветов и создает итератор цветов.

Описание генератора

Создает итератор сгенерированных цветов. Принимает один аргумент:

1. `amount` — количество цветов, которое нужно сгенерировать, число.

Описание итератора

Количество итераций равно числу, которое передано в генератор `palette`. В каждой итерации должен вернуть цвет в шестнадцатеричном представлении формата RGB, например `#f8dff2`, который обычно применяется в CSS.
Для генерации аналогичных цветов удобнее всего использовать формат HSL. Потому что в нём цвет состоит из тона (hue), насыщенности (saturation) и светлоты (lightness). У аналогичных цветов насыщенность и светлота будут одинаковыми, а тон будет отличаться на равные значения. Тон обычно представляют в виде цветового круга. Тон, равный `0` (0˚), и тон, равный `1`(360˚), совпадают.
И, если нам нужно получить 3 цвета, мы можем взять на круге 3 точки через каждые 120˚ (`360 / 3`): например, 33˚, 153˚ и 273˚, что соответствует тонам `33/360`, `153/360` и `273/360`.
Сгенерировав случайный тон, насыщенность и светлоту и получив нужное количество тонов, останется только перевести полученный цвет в RGB-формат. Для этого воспользуйтесь функцией `hslToRgb`.
Описание функции `hslToRgb`
Функция принимает три числа, соответствующие трем компонентам HSL представления цвета: тону, насыщенности и светлоте:

1. hue — тон, число от `0` до `1`;
2. `saturation` — насыщенность, число от `0` до `1`;
3. `lightness` — светлота, число от `0` до `1`.

Функция возвращает строку в формате RGB.

Пример использования

```
for (const color of palette(3)) {
  console.log(color);
}
```

Если все реализовано верно, вы получите три случайных аналогичных цвета:

```
#817a8d
#8d817a
#7a8d81
```

Процесс реализации

1.Создайте генератор `palette`.
2.Сгенерируйте случайный тон, насыщенность и светлоту.

3.Определите шаг изменения тона, исходя из количества, которое нужно сгенерировать.

4.Меняйте тон в цикле на полученный шаг.

5.Если значение тона превысило 1, просто уменьшите его на 1.
6.В каждой итерации возвращайте в итератор RGB-цвет для полученных насыщенности, светлоты и текущего тона.

7.Количество итераций должно быть равно количеству цветов.

Задача № 2. Конкурс «Угадай скидку»

Для нашей новой акции с суперскидками мы хотим реализовать простую игру «угадай число». Наш сайт будет загадывать скидку, и пользователь должен будет её угадать.

Ваша задача — реализовать генератор `numberQuiz`, который бы смог контролировать процесс игры. В качестве аргумента он принимает загаданное число и выдает подсказки «больше» или «меньше» до тех пор, пока пользователь не угадает это число.

Описание генератора

Создает итератор, который контролирует ход игры. Принимает один аргумент:

1.`number` — число, которое необходимо угадать, число.

Описание итератора

Созданный итератор должен вернуть строку «Назовите число:» и ожидать последующих действий со стороны вызывающего кода.

Вызывающий код должен передавать во все последующие вызовы метода `next` число, которое называет пользователь. Если число совпало с загаданным, итератор должен завершиться с сообщением «Вы угадали, это 5». Иначе он должен вернуть подсказку «Меньше, чем 7!» или «Больше, чем 4!» и ожидать следующего хода.

Пример использования

В данном примере `attempts` — числа в том порядке, в котором их будет называть пользователь.

```
const attempts = [7, 4, 6, 5];

const quiz = numberQuiz(5);
```

```
let attempt, result;
do {
  result = quiz.next(attempt);
  console.log(result.value);
  attempt = attempts.shift();
} while (!result.done);
```

Если все реализовано верно, вы получите такой вывод:

```
Назовите число:
Меньше, чем 7!
Больше, чем 4!
Меньше, чем 6!
Вы угадали, это 5
```

Процесс реализации

1. Создайте генератор `numberQuiz`.
2. Верните сообщение «Назовите число:» с помощью `yield`.
3. Получите значение из итератора.
4. Сравните значение из итератора с тем, что было передано в генератор.
5. Если числа совпадают, завершите итератор сообщением «Вы угадали, это 5».
6. Если нет, верните с помощью `yield` подсказку.
7. И так, пока пользователь не угадает число.

Проверьте работу вашего кода на примере использования. Также протестируйте его, используя свои примеры.

Задача № 3. Планировщик машин

Для нашего отдела логистики потребовалось реализовать функционал, который бы показывал, какой заказ в какую машину грузить.

Реализовать простой класс `TruckPlanner`, который бы позволял в простом виде получить распределение заказов по машинам. На данном этапе не требуется решать задачу оптимального распределения заказов с целью сокращения количества машин.

Описание класса

Класс `TruckPlanner` должен принимать в конструктор следующие аргументы:

1. `weightLimit` — лимит по весу для всех машин, число.

Методы

У экземпляра класса `TruckPlanner` должен быть доступен один метод:
add — добавляет заказ в планировщик, принимает в качестве аргумента заказ `Order`.

Мета-программирование

У экземпляра класса `TruckPlanner` должен быть реализован итератор, который возвращает машины `Truck` с распределенными по ним заказами `Order`.
Для реализации итератора должен обязательно быть использован генератор. Уточним еще раз, что заказы нужно распределить по машинам просто последовательно, с учетом лимита, оптимизировать распределение не требуется.

Класс `Order`

Принимает в конструктор два аргумента:

1. `id` — идентификатор, число;
2. `weight` — вес заказа, число.

Класс `Truck`

Принимает в конструктор два аргумента:

1. `number` — порядковый номер машины, число;
2. `weightLimit` — лимит по весу, число.

Имеет следующие свойства и методы:

`totalWeight` — свойство - общий вес заказов, число, только для чтения.
`add` — метод - помещает заказ в машину, принимает заказ Order в качестве аргумента. Вернет истину, если заказ влезает в машину, иначе вернет ложь.
`isFit` — метод - проверит, поместится ли заказ в машину, принимает заказ `Order` в качестве аргумента. Вернет истину, если заказ влезает в машину, иначе вернет ложь.

Пример использования

```
const planner = new TruckPlanner(10);
planner.add(new Order(1, 2));
planner.add(new Order(2, 5));
planner.add(new Order(3, 4));
planner.add(new Order(4, 4));
planner.add(new Order(5, 1));
planner.add(new Order(6, 2));

for (const truck of planner) {
  truck.show();
}
```

Если все реализовано верно, вы получите такой вывод:

```
Машина №1 (общий вес груза 7кг):
  Заказ #1 2кг
  Заказ #2 5кг
Машина №2 (общий вес груза 9кг):
  Заказ #3 4кг
  Заказ #4 4кг
```

```
  Заказ #5 1кг
Машина №3 (общий вес груза 2кг):
  Заказ #6 2кг
```

Процесс реализации

1.Создайте класс `TruckPlanner`.
2.Реализуйте конструктор.

3.Реализуйте метод `add`.
4.Добавьте символьное свойство `Symbol.iterator`, сделав его генератором.
5.Перебирайте заказы, помещая их в машину `Truck`.
6.Как только заказы перестанут помещаться в машину, верните в итератор текущую машину и создайте новую.

7.И так до тех пор, пока заказы не закончатся.