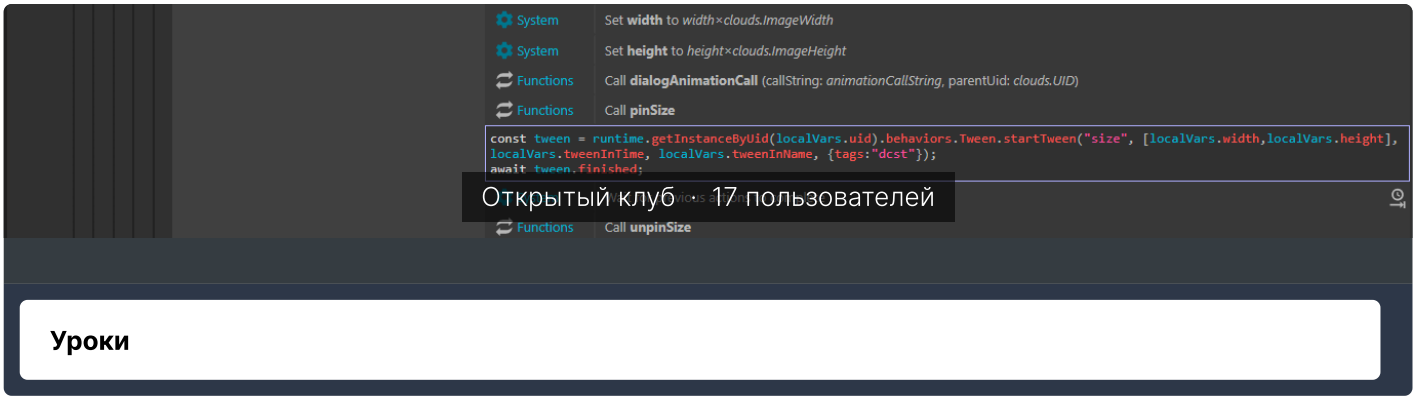


Урок 3 - Расширяем границы

Автор cliva,
11 декабря, 2022 в Уроки

Расширяем границы

Самое время узнать о ещё некоторых конструкциях js кода.

Мы уже знаем что такое цикл и как его использовать, но допустим нам надо сделать так, чтобы цикл состоящий из 6 итераций(повторений) выводил в консоль первые три раза число 000, а последующие три раза число 555, это мы можем сделать с помощью условного ветвления:

```
if (условие) строка выполняемого кода:
if (money < 100) alert("У вас не хватает на проезд")

if (условие){
        строка кода;
        строка кода;
        ...
}:
if (money >= 100){
        money-=100;
        alert("проезд оплачен")
}
```

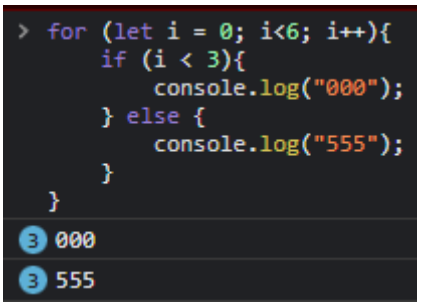
Также к любому если(**if**) можно добавить иначе - **else:**

```
if (money == 111){
        alert("счастливое число")
} else {
        alert("обычное число")
```

```
}

//Также мы можем собирать почти бесконечную цепочку if - else, небольшой глупый пример:
if (num == 1){
        alert("число = 1");
} else if (num == 2){
        alert("число = 2");
} else if (num == 3){
        alert("число = 3");
} else ...
```

Попробуем в консоли написать цикл, который выведет на первых трёх итерациях 000, а на последних трёх - 555:

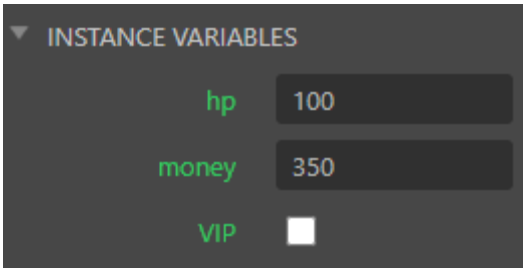


Вот и новая для нас фишка консоли - она собирает сообщения с одинаковым содержимым, которые идут друг за другом в одну кучу, слева число в кружочке отображает сколько раз было выведено это сообщение. Для закрепления предлагаю скормить консоли функцию с параметром, которая бы решала по какому тарифу мы можем проехаться на такси, если в параметре меньше 100 - мы идём пешком, если больше 100, но меньше 180 - тариф "эконом", а если больше или равно 180 - пусть предлагает тариф "комфорт":

Сначала пробуем сами (Показать контент)

Вернёмся к нашим человечкам из прошлого урока, добавим ещё парочку, теперь их 5, представим, что они попали в клуб, и только у некоторых из них есть ВИП-билеты, и по этому билету человек может получить золотую корону, чтобы всем показать что он "особая" персона, так как нашим человечкам мы не давали переменной зависти, то и беспокоиться не о чем - драки не будет, а значит приступаем к выполнению задачи:

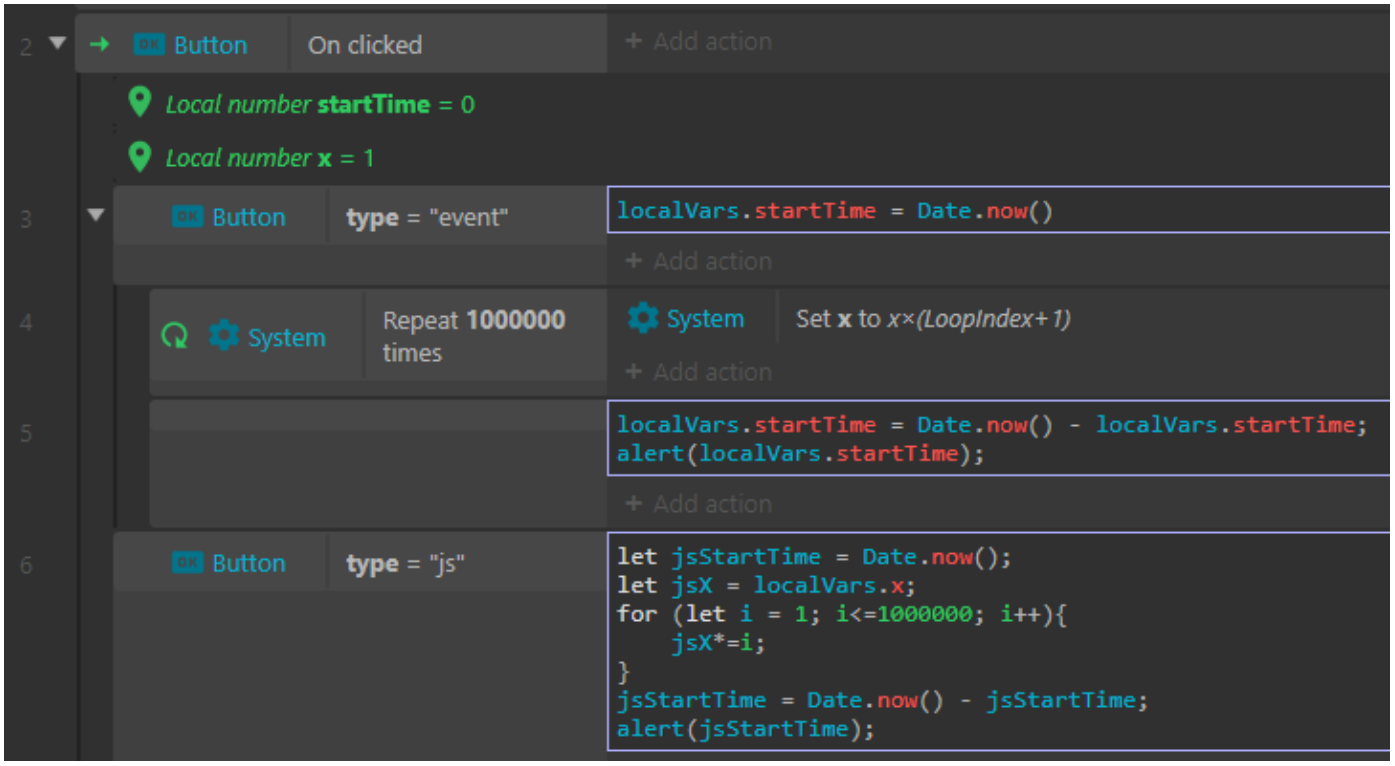
Добавим объекту человека булеву переменную **VIP:**



Нарисуем новую кривую (или не очень) анимацию - человечка с короной, и раздадим ВИП-билеты, я дам их центральному и правому человеку, теперь переходим к списку событий, при старте уровня через цикл нужно проверить если у человека билет, и если он есть - сменим ему анимацию, сменить анимацию спрайта можно с помощью метода setAnimation("Имя анимации"):

Сначала пробуем сами (Показать контент)

Думаю стоит упомянуть, что с помощью js можно упростить сложные/большие вычисления, добавим на макет кнопку и сделаем её копию, добавим кнопке переменную type, первой кнопке в текст напишем **benchmark event,** второй кнопке напишем **benchmark js,** в переменные кнопок type занесём значения event и js соответственно, перейдем в список событий и реализуем двумя способами(ивентами и кодом) подсчёт факториала (1*2*3*4*5*6*...*n) в качестве n возьмём число 1000000 и посмотрим, сколько займёт каждый просчёт, для этого воспользуемся функцией Date.now() - она возвращает кол-во миллисекунд прошедших с полуночи 1 января 1970 года, замерим время до вычисления факториала и после, а разница - как раз время, которое потребовалось на вычисление:



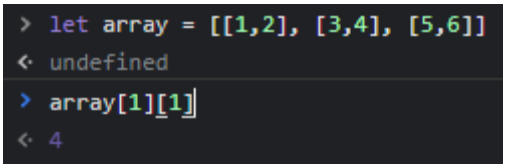
Запускаем, тестируем и замечаем, что реализация на js почти в 10 раз обгоняет реализацию на ивентах! Добавим ка ещё пару ноликов в циклы - реализация кодом снова обгоняет ивенты в заметные 10-15 раз. Почему это происходит? - мы избавляемся от оболочки C3 и работаем напрямую с переменными js, и я подозреваю, что мы сделали срез в адресации - вместо абстрактного **...runtime.globalVars.x** - мы используем просто **x**, что в теории выполняется на сотые миллисекунды быстрее, а благодаря циклу это имеет накопительный эффект

Снова поиграем с консолью, а именно с массивами, из C3 мы знаем, что массивы могут быть одномерные, двумерные и трёхмерные - забудьте, в js нет таких условностей:

Вот "одномерный" массив, из которого я могу достать значение с помощью одной пары квадратных скобок:

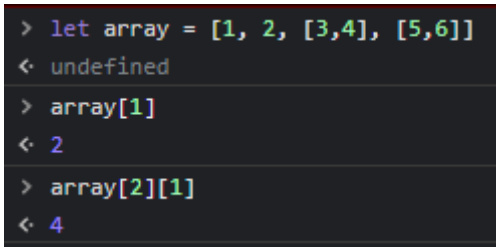
```
> let array = [1,2,3,4,5]
< undefined
> array[2]
< 3
```

Вот "двумерный" массив, из которого я могу достать значение с помощью двух пар квадратных скобок:



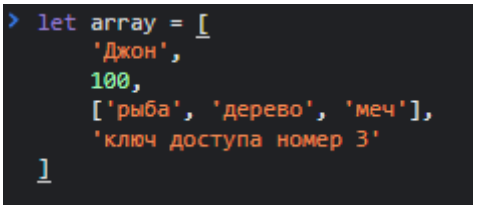
Да, вы правы, "двумерный" массив - это массив "одномерных" массивов

И вот уничтожение устоев, какой это массив?:



Полу-двумерный? :)

На самом деле это самый обычный массив, у которого первые два элемента являются числами, а третий и четвёртый - являются массивами. Запоминаем - понятие "одномерный массив" можем вообще не использовать, говорим просто массив, понятие "двумерный массив" используем когда описываем массив с массивами одинаковой длины, "трёхмерный массив" - массив с массивами, которые содержат в себе массивы одинаковой длины, также для любых n-мерных массивов, но такие требуются крайне редко, и куда более часто встречаются массивы, которые содержат в себе какие-то значения, ещё массивы, те массивы могут также содержать какие-то значения или ещё массивы, пример:



Это своеобразное хранилище всей информации о каком-нибудь персонаже для нашей игры, которого зовут Джон, у него 100 жизней, в инвентаре лежит рыба, дерево и меч, также у него есть ключ доступа номер 3, такую структуру не очень удобно расширять, но в таком компактном виде в принципе приемлемо использовать, к удобствам обратимся чуть позже

Возвращаемся к C3, поработаем с объектом tilemap, для этого добавим его в проект, нарисуем два тайла - один светлый, другой тёмный (я изменил размер изображения на 16x32, нарисовал белый и чёрный квадрат, ширину и высоту тайлов в свойствах объекта выставил 16x16). Сделаем из тайлмапа шахматную доску, для этого понадобится умение запихнуть один цикл в другой (их счётчики должны называться по разному!) и метод для тайлмапа - .setTileAt(x тайла,y тайла, номер тайла):

> **Сначала пробуем сами (Показать контент)**

Наверняка у вас возник вопрос, откуда я беру все эти методы, ответ прост - на **официальном сайте C3** Большинство интересующих нас методов работы с плагинами уже задокументировано, и если вы задались вопросом, какой метод мне использовать для такого-то плагина чтобы сделать с ним что-то - проверьте документацию, скорее всего там это написано.

Думаю, стоит взять небольшую паузу

Домашнее задание:

- Выдайте человечкам разное кол-во денег, раздайте 2-3 ВИП-билета в произвольном порядке, сделайте так, чтобы у самой богатой ВИП-персоны помимо короны появлялась ещё и трость

- Задайте массив из 10 элементов, это могут быть как числа, так и строки, сделайте функцию, которая будет переворачивать этот массив (последний элемент станет первым, предпоследний вторым и тд), задание можно выполнить в консоли

- Кодом заполните тайлмап таким образом, чтобы он по краям был тёмным, а внутри светлым (светлый квадрат, с тёмной обводкой)

-* Нарисуйте на тайлмапе такой смайлик - 

подсказка для последнего задания (Показать контент)

Если какое-то задание не получилось или остались вопросы по материалу - задаём вопросы в этой теме, если имеются вопросы не относящиеся именно к этой теме - переходим в клуб, во вкладке **Общее** есть подходящие темы.

Исходник урока - expTheBound.c3p
Недоступно

Вернуться в основную тему

Изменено 11 декабря, 2022 пользователем cliva

1

bioid.space

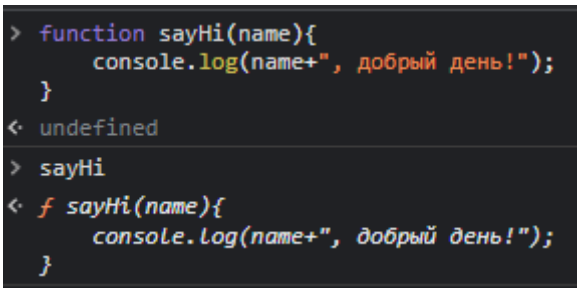
Продолжим расширять границы

Отступление - это продолжение основного урока, мы не будем трогать C3, лишь расширим свой инструментарий для работы с js

Ранее мы уже использовали функции в js - объявляли их, однако существует ещё один синтаксис создания функций, который называется Function Expression (Функциональное Выражение), этот способ позволяет нам создавать новую функцию в середине любого выражения, вот так он выглядит:

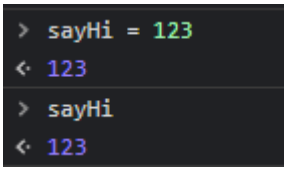
```
let sayHi = function(name) {
  console.log(name+", добрый день!");
};
```

Да, функция это значение переменной и всегда ей была, давайте убедимся в этом, объявим функцию ранее знакомым нам способом и вызовем её без скобок:

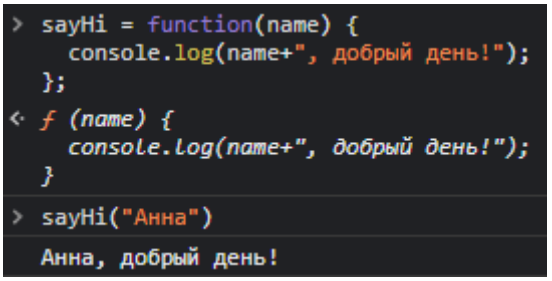


(1)

Как видим, в консоль нам попал текст кода, который объявляет эту функцию, и да - мы можем просто взять и перезаписать функцию каким-либо другим значением:



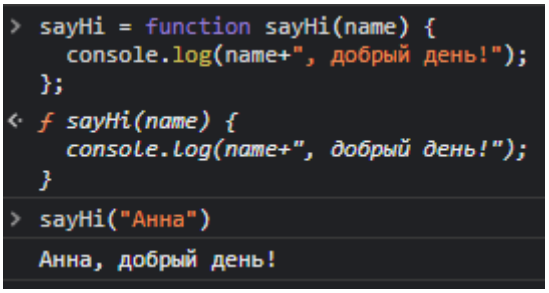
И также можем перезаписать любую переменную функцией и вызывать её по надобности:



(2)

(let опущено, так как переменная уже задавалась в консоли ранее)

Как можно заметить - у функции пропало имя, на самом деле оно никуда не пропало, объявляя функцию способом (1) мы указываем имя после слова function, а в способе (2) именем является переменная, но никто не запрещает сделать так:



Но в моём виденье это выглядит как "мокрая вода", "масло масленное" или "светящийся свет" - тавтология

Подробнее о функциональных выражениях - https://learn.javascript.ru/function-expressions

Также существуют **стрелочные функции** - очень удобный способ записи для однострочных функций, лучше наверное сразу показать на примере

как это

```
> let summ = function(a,b){
      return a+b;
  };
< undefined
> summ(2,4)
< 6
```

превращается в это

```
> let summ = (a, b) => a + b;
< undefined
> summ(2,4)
< 6
```

Стрелка `=>` позволяет опустить слово function и return, и читается это достаточно удобно - в переменную summ подаём параметры a и b, которые затем складываем и возвращаем, стрелочная функция также может быть многострочной:

```
> sayHi = (name) => {
      console.log("Это стрелочная функция");
      console.log(name+", добрый день!")
  };
< (name) => {
      console.log("Это стрелочная функция");
      console.log(name+", добрый день!")
  }
> sayHi("Анна")
  Это стрелочная функция
  Анна, добрый день!
```

(let опущен, так как sayHi уже был задан в консоли ранее)

В многострочной стрелочной функции, если необходимо возвращаемое значение - необходимо использовать return

Подробнее о стрелочных функциях - https://learn.javascript.ru/arrow-functions-basics

Вернёмся у массивам, ранее мы уже делали цикл, который проходится по каждому элементу массива, что-то наподобие такого:

```
> let ary = [3, 2, 1, 2, 3, 4];
  for (let i = 0; i<ary.length; i++){
      console.log( ary[i] )
  }
  3
  2
  1
  2
  3
  4
```

Выглядит громоздко, есть способ короче, воспользуемся методом **.forEach** - выполняет указанную функцию один раз для каждого элемента в массиве :

```
> let ary = [3, 2, 1, 2, 3, 4];
  ary.forEach(i => console.log(i));
  3
  2
  1
  2
  3
  4
```

Выглядит намного короче, особенно с использованием стрелочной функции
Вот ещё пример с [mdn](), без стрелочной функции:

```
const items = ['item1', 'item2', 'item3']
const copy = []

// до
for (let i = 0; i < items.length; i++) {
  copy.push(items[i])
}

// после
items.forEach(function(item){
  copy.push(item)
})
```

Думаю стоит взять небольшую паузу, так как продолжение урока я не планировал сильно большим, возможно о чём-то опять вспомню с утра и будет третья часть этого же урока)

В качестве домашнего задания - вернитесь к предыдущим урокам и выполните домашние задания в них, но с учётом новых знаний

1

bioid.space

1 месяц спустя...

Какие преимущества использовании js дает по сравнению с обычным визуальным кодингом?

@Parahod

Попробуй тоже самое на виз. программировании сделать)

bioid.space

> **В 30.01.2023 в 00:48, cliva сказал:**
>
> @Parahod
>
> Попробуй тоже самое на виз. программировании сделать)

То есть не особо нужен.

Плюс к возможностям, @Wilson делал тесты на циклах и определил, что JS производительнее событи (не помню на сколько).

Сижу с телефона, исходник не посмотрю / не отправлю.

@Parahod я только начал изучать JS, в том числе по этим урокам. И пока вижу одни неудобства, работать с событиями намного приятнее и быстрей получается, но это пока, возможно дальше что-то поменяется.
Насчёт производительности вот здесь в этом уроке есть тест, можешь сам проверить:

> **В 11.12.2022 в 01:29, cliva сказал:**
>
> Думаю стоит упомянуть, что с помощью js можно упростить сложные/большие вычисления,

> **В 11.12.2022 в 01:29, cliva сказал:**
>
> реализуем двумя способами(ивентами и кодом) подсчёт факториала (1*2*3*4*5*6*...*n) в качестве n возьмём число 1000000 и посмотрим, сколько займёт каждый просчёт

> **В 11.12.2022 в 01:29, cliva сказал:**
>
> Запускаем, тестируем и замечаем, что реализация на js почти в 10 раз обгоняет реализацию на ивентах!

Игры: dmitrygalias.itch.io

> **В 30.01.2023 в 07:16, dmitryartist сказал:**
>
> реализация на js почти в 10 раз обгоняет реализацию на ивентах!

А вот это уже интересно

А как через JS отловить клик на копию спрайта и дальше передать её для обработки в другое действие? И сменить там прозрачность

> **В 30.01.2023 в 21:16, Parahod сказал:**
>
> как через JS отловить клик на копию спрайта

Я во 2ом уроке как раз у @cliva это спрашивал. Это можно сделать через .getPickedInstances(), но там не всё так просто.

Игры: dmitrygalias.itch.io

Последние посетители 0 пользователей онлайн

Ни одного зарегистрированного пользователя не просматривает данную страницу