Pagina que parei no livro *Get Programming With JavaScript (2016)* { 231 }. O livro em questão usa um exemplo de um jogo feito em JS como projeto final e vai utilizando ele ao longo do livro.

Em JS, diferente de algumas outras linguagens é preciso abrir o console em que os comandas apareceram em um outro local, para essa opção temos o ‘JS Bin’ ou o ’Visual Studio Code’.

Para usar os comandos no JS bin basta abrir o site e habilitar a opção de ver o console

No Visual Studio Code temos que gerar uma página da web e rodar o script de lá. Aqui vou tentar utilizar o Visual Studio Code por parecer melhor de organizar os arquivos no computador.

Para fazer isso basta criar um arquivo html (no caso criamos um home.html) e digitamos o seguinte código:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Teste em JS</title>

    <script src="./aula.js"></script>

</head>

<script src="./scripts.js"></script>

<body>

    <h1>'Olá Mundo'</h1>

</body>

</html>

E mandamos executar com a extensão live server que deve estar previamente baixada através do VS Code. Isso abrirá uma pagina, de lá se aperta f12 para abrir o console da pagina.

Criamos então um arquivo chamado scripts.js na mesma pasta que o home e nele escrevemos os códigos.

PARTE UM:

Olá Mundo:

Como de costume, abra o console e digite o seguinte código:

let nome;

let local;

nome = "Olá";

local = "Mundo";

console.log(nome + local);

Pronto, aqui está o nosso primeiro Olá Mundo!

Valores de objetos:

As variáveis em JS podem ser objetos, os objetos são variáveis com mais de uma propriedade, cada parâmetro dentro de um objeto pode ser acessado utilizando a notação de ‘.’.

No exemplo seguinte vemos um exemplo disso para uma variável dentro de um jogo que pode receber vários parâmetros e cada parâmetro podendo conter um valor.

var jogador;

jogador = {

   nome : 'Leo',

   vida : 50,

   score : 100,

  };

console.log('Parabens ' + jogador.nome + ', voce fez ' + jogador.score + ' pontos! voce terminou com ' + jogador.vida + ' de HP.');

Ainda nesse programa, podemos alterar ou acrescentar o valor de um parâmetro dentro do objeto apensar delegando um novo para ele como no exemplo a seguira:

jogador.peso = 150;

console.log(jogador.nome);

console.log(jogador.peso);

jogador.peso = jogador.peso + 20

Aqui ainda é importante notar que o JS sempre lê o que está do lado direito do sinal de ‘=’ antes, portanto é possível delegar um valor a mais ao que já existia, como resultado no exemplo o valor da variável ‘peso’ no obj ‘jogador’ (jogador.peso) vale 170 no fim do código.

var filmes;

var indice;

filmes = {

    nome : ["A volta dos que nao se foram", "Tranças de um careca", "Poeira em alto mar"],

    nota : ["10","9","4"],

    diretor : ["Zezinho Perneta","Cuca Beludo","Joselito Boça"]

};

indice = 0;

console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

indice = 1;

console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

indice = 2;

console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

Aqui está um exemplo de como um objeto pode ser usado para guardar listas em seus parâmetros e organizar ainda mais o código. Dessa maneira fica mais fácil de se ler o código. Neste exemplo temos uma listagem de filmes que tem as informações de Nome, Diretor e Nota.

**“One of the main themes of Get Programming with JavaScript is managing complexity through good organization.”**

OBS: CASO PRECISE USAR APOSTROFE NO TEXTO BASTA UTILIZAR A SEGUINTE NOTACAO: \’ .

Funções:

Se objetos podem ser definidos como um conjunto e de propriedades, uma função pode ser definida como um conjunto de instruções. Sendo assim, são usadas para condensar uma parte de código que pode ser utilizada da mesma maneira mais de uma vez.

No exemplo utilizado anteriormente tivemos que repetir 3 vezes o mesmo bloco de texto para exibir as informações do filme, para facilitar a vida podemos criar uma função que quando chamada fara todo aquele escrito ser mostrado sem necessidade de ficar copiando o código tantas vezes.

Para criar uma função basta fazer como no exemplo seguinte: var filmes;

var indice;

var mostraDados;

filmes = {

    nome : ["A volta dos que nao se foram", "Tranças de um careca", "Poeira em alto mar"],

    nota : ["10","9","4"],

    diretor : ["Zezinho Perneta","Cuca Beludo","Joselito Boça"]

};

mostraDados = function () {

    console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

};

indice = 0;

mostraDados();

indice = 1;

mostraDados();

indice = 2;

mostraDados();

O resultado do console é o mesmoq eu o programa anterior, mas ele em si fica muito mais elegante!

Outra maneira mais elegante ainda de declarar uma função é fazer ela sem ter que declarar uma variável, observe como seria para o exemplo anterior:

function mostraDados () {

    console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

};

As funções também podem ser declaradas usado paramentros, que são basicamente variáveis que executam apenas dentro de uma função, para fazer isso basta definir quais vao ser os nome destas “variaveis” que serão executadas dentro do ‘()’, como é mostrado a seguir:

var filmes;

filmes = {

    nome : ["A volta dos que nao se foram", "Tranças de um careca", "Poeira em alto mar"],

    nota : ["10","9","4"],

    diretor : ["Zezinho Perneta","Cuca Beludo","Joselito Boça"]

};

function mostraDados (indice) {

    console.log("Nome: " + filmes.nome[indice] + ". Diretor: "+ filmes.diretor[indice]   + ". Nota: " + filmes.nota[indice]);

};

mostraDados(0);

mostraDados(1);

mostraDados(2);

Veja como este código fica ainda mais elegante que os dois anteriores, pois neste caso conseguimos eliminar 4 linhas de código! Aqui utiliza-se o parâmetro ‘indice’ ao declarar a função e ele é usado dentro dela, e ao chamarmos a função e no ‘()’ dela colocamos o valor que a variável ‘indice’ terá ela acessa mais elegantemente os números declarados nas listas definidas no objeto ‘filmes’.

OBS: VARIAVEIS CRIADAS DENTRO DE UMA FUNCAO PODEM SER UTILIZADAS FORA DELA SE FEITAS COMO DA MANEIRA ACIMA. O EXEMPLO ESTA ABAIXO:

function total (a, b) {

    resultado = a + b;

    console.log (resultado);

};

total(10, 10);

console.log (resultado);

Costuma ser uma boa prática criar e atribuir os valores das variáveis que uma função precisa para ser executada dentro da própria função para evitar que variáveis estejam faltando, com nome incorreto ou com valor diferente do desejado, em suma basta deixar as variáveis que a função precisar para funcionar perto da função a ser criada, sempre que possível e para sso servem os parâmetros da função.

Os parâmetros(chamados de argumentos) de uma função podem variar em quantidade, veja:

var showMessage;

message = "oi meu nome é bruce!";

showMessage = function (message, message2, message3) {

    console.log("A mensagem no.1 é: " + message);

    console.log("A mensagem no.2 é: " + message2);

    console.log("A mensagem no.3 é: " + message3);

};

showMessage("testando 1, 2...", "Aparentemente funcionou", "Sou um genio da computacao");

console.log(message);

Aqui temos um exemplo com 3 parametros, eles sempre são indicado no ‘()’ quando definimos a função e são separados por ‘,’. O parâmetro usado na função não influencia a variável criada fora dela e vice-versa.

Só para treinar: o JS tem uma função buit-in chamada Math.sqrt que faz a raiz quadrada do numero, para testa-la montamos o seguinte código:

sqrt = function (numberToSqrt) {

    console.log("SqRt of " + numberToSqrt + " is " + Math.sqrt(numberToSqrt));

}

cube = function (numberToCube){

    var result;

    result = numberToCube \* numberToCube \* numberToCube;

    console.log(numberToCube + " \* " + numberToCube + " \* " + numberToCube + " = " + result);

}

square = function (numberToSquare) {

  var result;

  result = numberToSquare \* numberToSquare;

  console.log(numberToSquare + " \* " + numberToSquare + " = " + result);

};

sqrt(9);

sqrt(999);

sqrt(7482);

console.log("------------------------------------")

cube(3);

cube(999);

cube(45);

console.log("------------------------------------")

square(10);

square(-2);

square(1111);

Diferença entre parâmetro e argumento: O parâmetro é o nome dado as variáveis inseridas dentro do ‘()’ quando criando a função e argumento é o nome dado ao valor inserido dentro do ‘()’ quando a função é chamada(usada).

Perceba como organizar o código em funções fica muita mais legível, flexível e fácil de acompanhar com o exemplo da função showPlayerInfo() usado no projeto do jogo, já com o exercício do livro de “circular” o nome do personagem resolvido usando a função built-in ‘substring’:

var showPlayerName = function (playerName) {

    console.log("\* " + playerName + " \*");

};

var showPlayerHealth = function (playerName, playerHealth) {

    console.log(playerName + " has health " + playerHealth);

};

var showPlayerPlace = function (playerName, playerPlace) {

    console.log(playerName + " is in " + playerPlace);

};

var showLine = function (playerName) {

    var line = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

    line = line.substring(0, playerName.length + 4);

    console.log(line);

};

var showPlayerInfo = function (playerName, playerPlace, playerHealth) {

    console.log("");

    showLine(playerName);

    showPlayerName(playerName);

    showLine(playerName);

    console.log("----------------------------");

    showPlayerPlace(playerName, playerPlace);

    showPlayerHealth(playerName, playerHealth);

    console.log("----------------------------");

    console.log("");

};

var player1 = {

    name: "Kandra",

    place: "The Dungeon of Doom",

    health: 50

};

var player2 = {

    name: "Dax",

    place: "The Old Library",

    health: 40

};

showPlayerInfo(player1.name, player1.place, player1.health);

showPlayerInfo(player2.name, player2.place, player2.health);

Retorno:

Por vezes é útil que uma função retorne algum valor para o código após ser executada, para isso existe a palavra ‘return’, ao colocar ela no código dentro da função ela retornara qualquer valor/dado inserido após o ‘return’ e será o valor que substituirá a chamada da função no código.

var getMyMessage;

getMyMessage = function (message){

    return message

}

Neste código em particular o console exibira qualquer coisa que tenha sido usada como argumento da função getMyMessage() quando ela for chamada no console.

OBS: APÓS UM PROGRAMA SER EXECUTADO O CONSOLE PODERA ACESSAR AS FUNCOES CRIADAS CHAMANDO ELAS E DEFININDO SEUS ARGUMENTOS (SE NECESSÁRIO);

Aqui está mais um exemplo de como o ‘return’ funciona, sendo que neste caso o console mostra o resultado de 3 numeros dados como argumentos da função ‘add’.

var add;

var sum;

add = function (number1, number2, number3) {

    var total = number1 + number2 + number3;

    return "The sum of " + number1 + " and " + number2 + " and " + number3 + " is " + total;

};

sum = add(23,50,34);

console.log(sum)

Aqui vale dizer que para ‘tirar’ um valor de uma função é necessário delegar à uma variável a função, desta maneira o ‘return’ da função ficara logada no valor da variável; no exemplo anterior por exemplo, caso você queira acessar o valor da variável ‘sum’ no console ela retornara o valor de 107 que foi o valor do ‘return’ da função ‘add’.

Aqui continua com o exemplo de mostrar a informação do jogador conforme o projeto proposto no livro e de acordo com as diretrizes dele.

var player1 = {

    name : "Leo",

    place : "Tower of Doom",

    HP : 50,

    items: "A bottle of wine, one dirty sock"

};

var player2 = {

    name : "Nicolly",

    place : "Sisters of Fate",

    HP : 50,

    items: "One pearl necklace, one sticky finger"

};

var getLine = function () {

    return "==============================";

};

var getName = function (name) {

    return name;

};

var getLocation = function (name, location) {

    return name + " is in the " + location;

};

var getHP = function (name, HP) {

    return name + " has " + HP + " HitPoints";

};

var getItems = function (name, items) {

    return name + " has these items: " + items;

};

var playerInfo = function (name, place, HP, items) {

    var playerInfo = name;

    playerInfo += "\n" + getLine();

    playerInfo += "\n" + getLocation(name, place);

    playerInfo += "\n" + getHP(name,HP);

    playerInfo += "\n" + getItems(name, items);

    playerInfo += "\n" + getLine();

    playerInfo += "\n";

    return playerInfo;

};

console.log(playerInfo(player1.name, player1.place, player1.HP, player1.items));

console.log(playerInfo(player2.name, player2.place, player2.HP, player2.items));

E aqui está a minha sugestão, retirando uma variável de jogador e colocando os dados de cada um em uma lista sendo que cada um é para um jogador, dessa forma reduz-se o numero de variáveis para cada jogador e armazena-se todos os jogadores em uma variável só; talvez isso não seja o mais interessante mas para mérito de aprendizagem é bem interessante.

var player = {

    name : ["Leo","Nicolly"],

    place : ["Tower of Doom","Sisters of Fate"],

    HP : [50,70],

    items: ["A bottle of wine, one dirty sock","One pearl necklace, one sticky finger"],

};

var getLine = function () {

    return "==============================";

};

var getName = function (name) {

    return name;

};

var getLocation = function (name, location) {

    return name + " is in the " + location;

};

var getHP = function (name, HP) {

    return name + " has " + HP + " HitPoints";

};

var getItems = function (name, items) {

    return name + " has these items: " + items;

};

var playerInfo = function (name, place, HP, items) {

    var playerInfo = name;

    playerInfo += "\n" + getLine();

    playerInfo += "\n" + getLocation(name, place);

    playerInfo += "\n" + getHP(name,HP);

    playerInfo += "\n" + getItems(name, items);

    playerInfo += "\n" + getLine();

    playerInfo += "\n";

    return playerInfo;

};

console.log(playerInfo(player.name[0], player.place[0], player.HP[0], player.items[0]));

console.log(playerInfo(player.name[1], player.place[1], player.HP[1], player.items[1]));

O exemplo acima mostra uma boa organização dos dados do código mas ainda há uma maneira de otimiza-lo:

Funções e objetos:

Também pode-se acessar as propriedades de um objeto dentro de uma função, veja o código abaixo:

var planet1;

var getPlanetInfo;

var planet2;

planet1 = {

    name: "Jupiter",

    position: 5,

    type: "Gas Giant",

    radius: 69911,

    sizeRank: 1

};

planet2 = {

    name: "Xuxa",

    position: 666,

    type: "Fun house",

    radius: 666000,

    sizeRank: 69

};

getPlanetInfo = function (planet) {

    return planet.name + ": planet number " + planet.position + ", " + planet.type + ", " + planet.radius + "km of radius, size Rank no. " + planet.sizeRank;

};

console.log(getPlanetInfo(planet1));

console.log(getPlanetInfo(planet2));

Veja como é possível acessar as diferentes variáveis do objeto ‘planet#’ apenas utilizando a notação de ‘.’ na função para se acessar os parâmetros dele como se o fizesse com o objeto em si.

Aqui também vale relembrar que é possível passar novos parâmetros para os objetos atribuindo valores a eles:

var planet1 = { name: "Jupiter", radius: 69911 };

var calculateSizes = function (planet) {

    var r = planet.radius;

    planet.diameter = 2 \* r;

    planet.area = 4 \* 3.142 \* r \* r;

    planet.volume = 4 \* 3.142 \* r \* r \* r / 3;

};

var displaySizes = function (planet) {

    console.log(planet.name);

    console.log("diameter = " + planet.diameter + " km");

    console.log("surface area = " + planet.area + " square km");

    console.log("volume = " + planet.volume + " cubic km");

};

calculateSizes(planet1);

displaySizes(planet1);

ou alterar valores já existentes:

var point1;

var point2;

var move;

var showPoint;

var reflectX;

var rotate;

rotate = function (point) {

    spam = {

        x : -point.y,

        y : point.x

    };

   return point = {

        x : spam.x,

        y : spam.y

    };

};

reflectX = function (point) {

    return {

        x : -point.x,

        y : point.y

    };

};

move = function (point, change) {

    return {

        x: point.x + change.x,

        y: point.y + change.y

    };

};

showPoint = function (point) {

    console.log("( " + point.x + " , " + point.y + " )");

};

point1 = { x : 2,

          y : 5

         };

point2 = move(point1, { x : 7, y : -6 });

showPoint(point1);

console.log("Move 7 across and 6 down: ");

showPoint(point2);

console.log("Reflecting the second point in the x-axis: ");

point2 = reflectX(point2);

showPoint(point2);

console.log("Rotating the reflected point by 90 degrees: ");

point2 = rotate(point2);

showPoint(point2);

Métodos:

Por vezes se faz interessante juntar de funções que trabalham com um tema em comum em algum objeto, a essas funções damos o nome de ‘métodos’. Para cria-los basta lembrar que podemos atribuir funções a variáveis, sendo assim:

var espaco;

espaco.newLine = function (){

    return "\n";

};

espaco.blank = function(){

    return "";

};

             // ou pode-se fazer deste outro jeito mais direto

espaco = {

    newLine : function () {return "\n";},

    blank : function () {return "";},

};

            // Para demonstrar:

console.log("Linha 1");

espaco.newLine();

espaco.blank();

console.log("Linha 2");

Um dos métodos built-ins do JS é o Math, ele lida com operações numéricas, veja um exemplo de como criar uma caixa para display de texto utilizando as funções min e max deste método:

var line = function (lineLength) {

  var line = "========================================";

  lineLength = Math.max(0, lineLength);

  lineLength = Math.min(40, lineLength);

  return line.substr(0, lineLength);

};

var spaceLine = function (lineLength) {

  var line = "                                        ";

  lineLength = Math.max(0, lineLength);

  lineLength = Math.min(40, lineLength);

  return line.substr(0, lineLength);

};

var emptyBox = function (boxLength) {

  var line = "========================================";

  var space = "                                        ";

  var num = 2;

  boxLength = Math.max(0, boxLength);

  boxLength = Math.min(40, boxLength);

  return line.substr(0, boxLength) + "\n=" + space.substr(0, boxLength - num) + "=\n=" + space.substr(0, boxLength - num) + "=\n=" +  space.substr(0, boxLength - num) + "=\n=" + line.substr(0, boxLength);

};

console.log(line(30));

console.log(line(40));

console.log(spaceLine(50) + "home");

console.log(emptyBox(15));

Também existem métodos que trabalham com strings, como é o caso do toUpperCase e do toLowerCase:

var planet = "Jupiter";

var bigPlanet = planet.toUpperCase();

var getBig = function (text){

    text = text.toUpperCase();

    return text;

};

var getSmall = function (text){

    return console.log(text + " becomes " + text.toLowerCase());

};

console.log(planet + " becomes " + bigPlanet);

console.log(getBig("leonardo"));

getSmall("SUPIMPA");

Seguindo com o exemplo do jogo, utilizando algumas funções criadas para formatar o texto e tendo em vista orientação a objetos aqui temos um exemplo de código que mostra as informações do jogador:

var spacer = {

    blank: function () {

      return "";

    },

    newLine: function () {

      return "\n";

    },

    line: function (length, character) {

      var longString = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*".repeat(2);

      longString += "----------------------------------------".repeat(2);

      longString += "========================================".repeat(2);

      longString += "++++++++++++++++++++++++++++++++++++++++".repeat(2);

      longString += "                                        ".repeat(2);

      length = Math.max(0, length);

      length = Math.min(80, length);

      return longString.substr(longString.indexOf(character), length);

    },

    wrap : function (text, length, character) {

      var padLength = length - text.length - 3;

      var wrapText = character + " " + text;

      wrapText += spacer.line(padLength, " ");

      wrapText += character;

      return wrapText;

    },

    box: function (text, length, character) {

      var boxText = spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      boxText += spacer.wrap(text, length, character) + spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      return boxText;

    }

  };

  var getPlayerName = function (player) {

    return player.name;

  };

  var getPlayerHealth = function (player) {

    return player.name + " está com " + player.HP + " HP.";

  };

  var getPlayerPlace = function (player) {

    return player.name + " está em " + player.place;

  };

  var getPlayerClasse = function (player) {

      return player.name + " é um(a) " + player.classe;

  };

  var getPlayerItems = function (player) {

      return player.name + " possui esses items: " + player.items;

  };

  var getPlayerInfo = function (player, character) {

    var place = getPlayerPlace(player);

    var health = getPlayerHealth(player);

    var classe = getPlayerClasse(player);

    var items = getPlayerItems(player);

    var longest = Math.max(place.length, items.length) + 4;

    var info = spacer.box(getPlayerName(player), longest, character);

    info += spacer.wrap(place, longest, character);

    info += spacer.newLine() + spacer.wrap(health, longest, character);

    info += spacer.newLine() + spacer.wrap(classe, longest, character);

    info += spacer.newLine() + spacer.wrap(items, longest, character);

    info += spacer.newLine() + spacer.line(longest, character);

    return info;

  };

  var player1 = {

      name: "Leo",

      classe: "Feiticeiro",

      place: "Torre do desespero",

      items: "celular, computador e oculos",

      HP: 100

  };

  var player2 = {

      name: "Nicolly",

      classe: "Barbara",

      place: "Casa dos elfos caolhos",

      items: "espatula, panela e celular",

      HP: 80

  };

  console.log(getPlayerInfo(player1, "="));

  console.log(getPlayerInfo(player2, "+"));

ARRAYS:

Um ‘array’ é um tipo de objeto que contém vários valores, ordenada, em outras linguagens também é conhecido como lista, cada item dentro deste ‘array’ é chamado de ‘elemento’. É utilizado para conseguir organizar informações, isso é importante pois um bom código é um código simples e organizado.

Para criar u array basta fazer como o que segue:

// exemplo de array (listas ordenadas)

var array1 = ["gato","cachorro","periquito"];

var array2 = ["elefante","girafa","refrigerante"];

console.log(array1);

console.log(array2)

Veja um exemplo de como é possível inserir dados uteis dentro de arrays e como é possível organiza-los:

// Organização: este codigo organiza lugares para visitar utilizando arrays

var lugar1 = {nome: "Belo Horizonte", estado: "Minas Gerais"};

var lugar2 = {nome: "Catanduva", estado: "Sao Paulo"};

var lugar3 = {nome: "Ilha das Couves", estado: "Sao Paulo"};

var esseAno = [lugar1, lugar2];

var proximoAno = [lugar3];

console.log(esseAno);

console.log(proximoAno);

Para acessar seus elementos individualmente basta utilizar seus índices, no exemplo a seguir vemos como podemos criar funções que interagem com esses arrays e os índices deles para que possamos criar scripts mais complexos.

var getVisitorReport = function (visitorArray, dayInWeek) {

    var days = ["Segunda-Feira",

                "Terça-Feira",

                "Quarta-Feira",

                "Quinta-Feira",

                "Sexta-Feira"

               ];

    var index = dayInWeek - 1;

    var visitorReport; // Perceba que aqui é criada a variavel para depois montar, dessa forma o codigo fica mais organizado.

    visitorReport = "Houveram ";

    visitorReport += visitorArray[index];

    visitorReport += " acessos na ";

    visitorReport += days[index];

    return visitorReport;

};

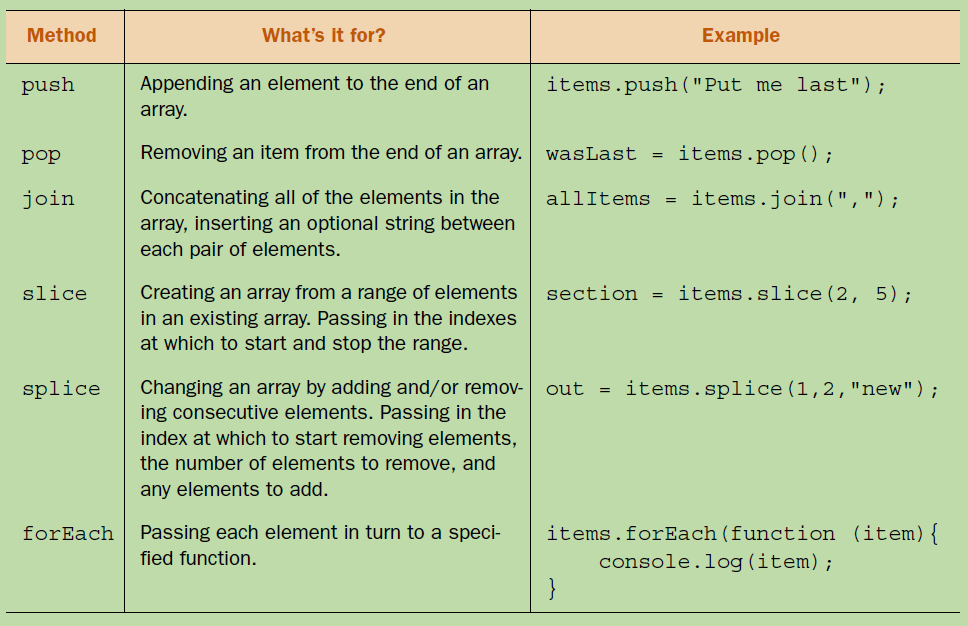
var visitors = [111,222,333,444,555]; //Aqui está representado a quantidade de visitantes por dia.

var day = 2; //Terça-Feira.

var report = getVisitorReport(visitors, day);

console.log(report);

Por array se tratar de um tipo de objeto ele possui métodos já built-in em JavaScript, aqui está uma lista de alguns métodos interessante para se ter em mente.



Veja um exemplo dos métodos pop, push e join sendo utilizados:

// Exemplo de métodos em arrays

var items = [];

var item = "The Pyramids";

var removed;

items.push(item);

items.push("The Grand Canyon");

items.push("Bondi Beach");

items.push("Ubatuba","Pindamonhangaba","Itaquaquecetuba");

console.log("Aqui está a lista de items depois dos 3 primeiros pushes: " + items);

items[0] = "Fernando de Noronha"

removed = items.pop();

console.log("Poping " + removed + " and adding " + items[0]);

console.log("Mostrando o restante da lista usando o 'join': " + items.join(" and "));

ForEach:

O forEach executa uma função passada como argumento para cada item dentro de um array, veja:

// Exemplo de ForEach()

var items;

var showInfo;

items = [

  "The Pyramids",

  "The Grand Canyon",

  "Bondi Beach",

  "Ubatuba"

];

showInfo = function (itemToShow) {

    var num = items.length;

    var letters = 0;

    for (var i = 0 ; i < num; i++){

        console.log(itemToShow[i]);

        console.log("There are " + itemToShow[i].length + " letters in this item");

        console.log("----------------------------------");

        letters += itemToShow[i].length;

    };

    console.log("There are " + letters + " letters in total")

};

showInfo(items)

O forEach possui três parâmetros, o item a ser usado, o índice do item que esta usando no array e o array todo, veja como fazer o exemplo anterior contar o total das letras armazenando seu valor na variável ‘total’ usando o forEach:

// contando o total de letra usando o forEach

var items;

var showInfo;

var total = 0;

items = [

  "The Pyramids",

  "The Grand Canyon",

  "Bondi Beach"

];

showInfo = function (item, index, array) {

    total += array[index].length;

    console.log(item);

    return total;

};

items.forEach(showInfo);

console.log(total);

o forEach SEMPRE é passado com a função ‘callback’ que passa os três argumentos citados acima para a função, você pode ou não acessa-los.

Isso é importante pois podemos acessar o elemento, o índice e o array quando executando uma função, veja no próximo exemplo como declarando apenas um parâmetro de execução para a função declarada em ‘itemsList.forEach’ usa apenas o argumento de elemento passado pelo ‘forEach’:

var getTotalBill = function (itemList) {

    var total = 0;

// a seguir ele vai pegar a lista que estiver como argumento para executar o que segue.

    itemList.forEach(function (item) { // usando apenas um parametro na funcao o forEach vai pegar apenas seu primeiro argumento que é o de elemento, no caso ele vai pegar um dos elementos de cada vez

        total += item.cost \* item.number; // para cada elemento ele vai multiplicar o '.cost' pelo '.number' e somar na variavel 'total'

    });

    return total; // e vai devolver ao programa o valor da variavel 'total'

};

var items = [{cost:1 ,number:10},

             {cost:2 ,number:5},

             {cost:3 ,number:3},

             {cost:4 ,number:2},

             {cost:5 ,number:1}]

console.log('Soma dos custos: ' + getTotalBill(items));

Aqui está um exemplo mais básico de como utilizar estes princípios:

[ "Dax", "Jahver", "Kandra" ].forEach(function (item, index, wholeArray) {

    console.log("Item: " + item);

    console.log("Index: " + index);

    console.log("Array: " + wholeArray);

  });

Veja como é possível utilizar uma função dentro de um ‘forEach’ dentro de uma função em um ‘forEach’:

var displayQuestion = function (questionArray){

    var options = [ "A", "B", "C", "D", "E" ];

    questionArray.forEach(function (item,i){ //Aqui o programa vai correr por cada questao no array de questoes passado e primeiramente imprimir qual é a pergunta

        console.log('Question No.' + (i+1))

        console.log(item.question)

        item.answers.forEach(function (answer,j){ // Aqui o programa ira correr a lista de respostas que fica armazenada em '.answers' de cada objeto e imprimir a opcao correspondetne baseada no indice de cada uma e a resposta correspondente.

            console.log(options[j] + ' - ' + answer)

        })

        console.log('\n')

    })

}

var question1 = {

    question : "What is the capital of France?",

    answers : [

      "Bordeaux",

      "F",

      "Paris",

      "Brussels"

    ],

    correctAnswer : "Paris"

  };

var question2 = {

    question : "What is the capital of Russia?",

    answers : [

      "Bordeaux",

      "F",

      "Paris",

      "Moscow"

    ],

    correctAnswer : "Moscow"

    };

var questions = [question1, question2];

displayQuestion(questions);

E para terminar o capitulo aqui esta o código para apresentar todas as informações do personagem do game agora aplicando o forEach para mostar um array de items que o personagem possui:

var spacer = {

    blank: function () {

      return "";

    },

    newLine: function () {

      return "\n";

    },

    line: function (length, character) {

      var longString = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

      longString += "----------------------------------------";

      longString += "========================================";

      longString += "++++++++++++++++++++++++++++++++++++++++";

      longString += "                                        ";

      length = Math.max(0, length);

      length = Math.min(40, length);

      return longString.substr(longString.indexOf(character), length);

    },

    wrap : function (text, length, character) {

      var padLength = length - text.length - 3;

      var wrapText = character + " " + text;

      wrapText += spacer.line(padLength, " ");

      wrapText += character;

      return wrapText;

    },

    box: function (text, length, character) {

      var boxText = spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      boxText += spacer.wrap(text, length, character) + spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      return boxText;

    }

  };

  // Player display code

  var getPlayerName = function (player) {

    return player.name;

  };

  var getPlayerHealth = function (player) {

    return player.name + " has health " + player.health;

  };

  var getPlayerPlace = function (player) {

    return player.name + " is in " + player.place;

  };

  var getPlayerItems = function (player) {

    var itemsString = "Items:" + spacer.newLine();

    player.items.forEach(function (item,i) {

      itemsString += "   - " + "(" +(i+1) + ") " + item + spacer.newLine();

    });

    return itemsString;

  };

  var getPlayerInfo = function (player, character) {

    var place = getPlayerPlace(player);

    var health = getPlayerHealth(player);

    var longest = Math.max(place.length, health.length) + 4;

    var info = spacer.box(getPlayerName(player), longest, character);

    info += spacer.wrap(place, longest, character);

    info += spacer.newLine() + spacer.wrap(health, longest, character);

    info += spacer.newLine() + spacer.line(longest, character);

    info += spacer.newLine();

    info += "  " + getPlayerItems(player);

    info += spacer.newLine();

    info += spacer.line(longest, character);

    info += spacer.newLine();

    return info;

  };

  var showPlayerInfo = function (player, character) {

    console.log(getPlayerInfo(player, character));

  };

  var showItem = function (player, itemNumber){

      return (player.items[itemNumber])

  };

  var addItem = function (player, item){

      player.items.push(item);

      return "Item added to inventory!"

  };

  // Create a player

  var player1 = {

    name: "Kandra",

    place: "The Dungeon of Doom",

    health: 50,

    items : ["a trusty lamp"]

  };

  showPlayerInfo(player1, "=");

  player1.items.push("a rusty key");

  showPlayerInfo(player1, "\*");

Building Functions:

Em JS você pode criar funções que criam objetos, isto é interessante pois é uma maneira mais fácil e mais organizada que digitar valor por valor de um objeto.

É possível atribuir métodos a objetos para que eles possam executar certas funções, veja no exemplo a seguir como o método ‘.showPlanet()’ é um método do objeto do tipo planeta:

// métodos em objetos:

var buildPlanet = function (name, position, type) {

    var planet = {};

    planet.name = name;

    planet.position = position;

    planet.type = type;

    planet.showPlanet = function () {

        var info = planet.name;

        info += ": planet " + planet.position;

        info += " - " + planet.type;

        console.log(info);

    };

    return planet;

};

var planet1 = buildPlanet(

    "Jupiter",

    5,

    "Gas Giant"

);

planet1.showPlanet();

De maneira geral este tipo de função funciona da seguinte maneira:

// Ideia geral de como fazer uma funcao que cria objetos:

var buildPlanet = function (name, position, type) {

    var planet = {}; // You create an empty object

    planet.name = name; // Assign properties

    planet.position = position; // Assign properties

    planet.type = type; // Assign properties

    return planet; // You return the object

    };

Funções Construtoras:

Na seção anterior definimos um objeto vazio dentro da função porem o Java Script tem um jeito de fazer isso automaticamente usando o ‘new’ e o ‘this’.

O ‘new’ deve ser utilizado ao chamar a função que criara o objeto, isso fará com que o Java script entenda que deve criar um novo objeto vazio dentro da função; já o ‘this’ é utilizado no momento em que estamos escrevendo a função para que o Java script entenda que deve atribuir os valores novos ao objeto que foi criado usando o ‘new’. Também não se faz necessário usar o ‘return’ quando se cria um objeto dessa maneira pois o JS já entende que é para retornar o objeto.

Resumindo: ‘new’ na chamada e ‘this’ na criação. Veja:

OBS: por convenção, para funções que criam objetos costuma-se // Criando um novo objeto utilizando uma funcao com 'new' e 'this'

var Item = function(name, quantity){

    this.name = name; //cria-se usando o this para atribuir os valores ao objeto vazio criado pelo 'new'

    this.quantity = quantity;

    this.showItem = function(){

        console.log('Item: ' + name + '; quantidade: ' + quantity);

    };

};

var itemTeste = new Item('Garrafas de água',5); //Usa-se o new para que o JS entenda que estamos querendo criar um objeto vazio novo para a variavel em questao.

itemTeste.showItem(); // Exemplo de acesso da funcao 'showItem()' criada pela funcao 'Item()'

//Mais um exemplo de como modificar e incluir parametros em objetos:

var Item = function(name, quantity){

    this.name = name;

    this.quantity = quantity;

    this.raridade = [];

    this.showItem = function(){

        console.log('Item: ' + name + '; quantidade: ' + quantity);

    };

    this.addRar = function (rar){

        this.raridade.unshift(rar);

    }

};

var itemTeste = new Item('Garrafas de água',5);

itemTeste.showItem();

inteanceof():

é um método de qualquer tipo de objeto que retorna True se ele foi criado a partir de uma função criadora ou False caso não tenha sido.

var item1 = new Planet("Jupiter", 5, "Gas Giant");

console.log(item1 instanceof Planet);

Nesse caso é printado no console True, já que a variável ‘item1’ foi um objeto criado a partir da função ‘Planet()’.

Aqui está o exemplo de um código que cria u pequeno quis, é utilizado uma função criadora para criar o quis e usa-se o ‘.push()’ para inserir as respotas na lista criada para as mesmas dentro do objeto criado com o ‘Quis()’

var Quiz = function (pergunta, resposta) {

    this.pergunta = pergunta;

    this.resposta = resposta;

    this.opcoes = [];

    this.addOpcao = function (opcao) {

        this.opcoes.push(opcao);

    };

    this.mostrarQuiz = function () {

        console.log(this.pergunta);

        this.opcoes.forEach( function (opcao, i) {

            console.log("(" + (i+1) + ") " + opcao);

        }) ;

    };

};

var questao1 = new Quiz("Qual o animal que come com o rabo?",

                        "Todos.");

questao1.addOpcao("Todos.");

questao1.addOpcao("Cachorro.");

questao1.addOpcao("Gato.");

questao1.addOpcao("Sapato.");

questao1.mostrarQuiz();

Em seguida está um exemplo do criador de lugares que fiz para o jogo baseado no exemplo do livro que também estará logo em seguida, procurei fazer o meu antes como base para não ficar muito dependente do exemplo do livro.

//Criador de lugares para o jogo, exemplo próprio

//Criador de Lugar

var Lugar = function (nome, descricao) {

    this.nome = nome;

    this.descricao = descricao;

    this.items = [];

    this.saidas = [];

    this.informacoes = function () {

        var informacoes = 'Local: ' + this.nome + '.\n';

        informacoes += 'Descricao: ' + this.descricao + '.\n';

        informacoes += 'Itens: ' + this.items.join('; \n       ') + '.\n';

        informacoes += 'Saidas: ' + this.saidas + '.\n';

        console.log(informacoes);

    };

    this.addItem = function(item) {

        this.items.push(item);

    };

    this.addSaida = function(saida) {

        this.saidas.push(saida);

    };

};

var salao = new Lugar ('Salao', 'Um salao cheio de estatuas e quadros. Poeira se amontoa nos cantos, parece que ninguem vive aqui há anos.');

salao.addItem('Lanterna');

salao.addItem('Tocha');

salao.addSaida('Cozinha');

salao.informacoes();

Segue o exemplo do livro:

//Criador de lugares para o jogo, exemplo do livro utilizando funcoes já criadas para formatacao de textos.

// The spacer namespace

var spacer = {

    blank: function () {

      return "";

    },

    newLine: function () {

      return "\n";

    },

    line: function (length, character) {

      var longString = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

      longString += "----------------------------------------";

      longString += "========================================";

      longString += "++++++++++++++++++++++++++++++++++++++++";

      longString += "                                        ";

      length = Math.max(0, length);

      length = Math.min(40, length);

      return longString.substr(longString.indexOf(character), length);

    },

    wrap : function (text, length, character) {

      var padLength = length - text.length - 3;

      var wrapText = character + " " + text;

      wrapText += spacer.line(padLength, " ");

      wrapText += character;

      return wrapText;

    },

    box: function (text, length, character) {

      var boxText = spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      boxText += spacer.wrap(text, length, character) + spacer.newLine();

      boxText += spacer.line(length, character) + spacer.newLine();

      return boxText;

    }

  };

  // The Place constructor

  var Place = function (title, description) {

      var newLine = spacer.newLine();

      this.title = title;

      this.description = description;

      this.items = [];

      this.exits = [];

      this.getItemsInfo = function () {

          var itemsString = "Items: " + newLine;

          this.items.forEach(function (item) {

              itemsString += "   - " + item;

              itemsString += newLine;

          });

          return itemsString;

      };

      this.getExitsInfo = function () {

          var exitsString = "Exits from " + this.title;

          exitsString += ":" + newLine;

          this.exits.forEach(function (exit) {

              exitsString += "   - " + exit.title;

              exitsString += newLine;

          });

          return exitsString;

      };

      this.getTitleInfo = function () {

          return spacer.box(

              this.title,

              this.title.length + 4,

              "="

          );

      };

      this.getInfo = function () {

          var infoString = this.getTitleInfo();

          infoString += this.description;

          infoString += newLine + newLine;

          infoString += this.getItemsInfo() + newLine;

          infoString += this.getExitsInfo();

          infoString += spacer.line(40, "=") + newLine;

          return infoString;

      };

      this.showInfo = function () {

          console.log(this.getInfo());

      };

      this.addItem = function (item) {

          this.items.push(item);

      };

      this.addExit = function (exit) {

          this.exits.push(exit);

      };

  };

  // Test the Place constructor

  var library = new Place(

      "The Old Library",

      "You are in a library. Dusty books line the walls."

  );

  var kitchen = new Place(

      "The Kitchen",

      "You are in the kitchen. There is a disturbing smell."

  );

  var hall = new Place(

      "The Main Hall",

      "You are in a large hall. It is strangely empty."

  );

  library.addItem("a rusty key");

  library.addExit(kitchen);

  library.addExit(hall);

  library.showInfo();

Métodos:

Agora com a habilidade de criar objetos usando funções que os criam podemos inserir funções como propriedades do objeto, a isto damos o nome de método. Anteriormente para acessa-las precisávamos passar o nome do objeto ao qual estava atuando a função como argumento, agora já que essas funções são métodos do objeto em questão basta usar a notação de ponto, ‘’player1.addItem(‘bola’)’.

Essa habilidade torna o código mais organizado e consequentemente mais legível, uma propriedade muito importante no meio da programação.

Null:

Null é um tipo de dado que representa que um objeto/variável ainda não possui valor, normalmente identifica que você cirou dada variável para usa-la posteriormente.

BRACKET NOTATION: FLEXIBLE PROPERTY NAMES

Até agora usamos a notação de ponto (dot notation) para acessar Keys (palavras chaves que acessam valores dentro de um objeto) em uma chamada, porem o java Script nos oferece uma maneira alternativa de fazer isso através do uso dos colchetes e dentro dele a key envolta por aspas. Desta maneira podemos criar uma key que é uma string, isto pode ser útil em algumas situações diferentes como quando por exemplo queremos que uma key tenha um valor com espaço ou caractere especial, veja um exemplo:

//Demonstraçao de keys sendo strings

var objeto = {};

objeto["minas gerais"] = "Deu certo!";

objeto["espaço 1"] = "Também deu certo!";

console.log(objeto);

Aqui está um exemplo de como acessar os valores de um objeto usando ‘forEach’:

// Como acessar valores de objetos

var ages = {

    "Kandra Smith" : 56,

    "Dax Aniaku"   : 21,

    "Blinky"       : 36,

    "Leo"          : 25,

    };

var keys = Object.keys(ages);

ages["Nicolly"] = 24;

ages.gato = 11;

keys.forEach(function (key) {

  console.log(key + ", " + ages[key] + ", Name Lenght: " + key.length);

  console.log("----------");

});

Sendo o ‘forEach’ um método que itera uma função sobre todos os valores dentro de um objeto, podemos utiliza-lo para criar um contador de letras (ou ate mesmo palavras) para alguns textos; no exemplo a seguir usamos vários tweets e contaremos quantas vezes cada caractere aparece nele.

//Contador de letras

var letters = {};

var tweetText = tweets.join(""); //Or just use " " o separate between words.

tweetText = tweetText..toLowerCase();

var tweetLetters = tweetText.split(""); //Or just use " " o separate between words.

tweetLetters.forEach(function (letter) { //First we assign every letter/word the value 0

    letters[letter] = 0;

});

tweetLetters.forEach(function (letter) { //Then when we find it again we add 1 to its property

    letters[letter] += 1;

});

console.log(letters);

A seguir está um exemplo no jogo de como organizar as saídas em objetos, utilizando a notação recém aprendida, para criar uma camada a mais de mistério ao apresentar as saidas para o jogador que agora invés de lugares definidos para ir, terá opções mais ‘obscuras’; veja:

//Criando e Acessando lugares do jogo com bracket notation

var Place = function (title) {

    this.title = title;

};

var kitchen = new Place("The Kitchen");

var dungeon = new Place("The Dungeon");

var hall = new Place("The entrance hall");

var basement = new Place("The Basement");

var exits = {};

exits["north"] = kitchen;

exits["the trapdoor"] = dungeon;

exits["south"] = hall;

exits["west"] = basement;

var keys = Object.keys(exits);

keys.forEach(function (key) {

    console.log(key + " goes to " + exits[key].title);

});

Uma ideia mais completa deste criador de saidas seria criar ainda a função de adicionar saidas e de mostras saidas existentes, veja:

//Exemplo de criacao de saidas para lugares

var Place = function (title, description) {

    this.title = title;

    this.exits = {};

    this.addExit = function (direction, exit) {

        this.exits[direction] = exit;

    };

    this.showExits = function () {

        console.log("Exits from " + this.title + ":");

        Object.keys(this.exits).forEach(function (key) {

            console.log(key);

        });

    };

};

var library = new Place("The Old Library");

var kitchen = new Place("The Kitchen");

var garden = new Place("The Kitchen Garden");

kitchen.addExit("south", library);

kitchen.addExit("west", garden);

library.addExit("north",kitchen);

garden.addExit("east",kitchen);

kitchen.showExits();

PARTE DOIS: organizando o programa

ESCOPO:

Existem dois escopos, o local e o global. O Global é o escopo que é criado ao iniciar o programa, as variáveis criadas nele podem ser acessadas em qualquer lugar do código; já o escopo local é criado cada vez que uma função ou objeto/classe é criado, as variáveis utilizadas nele podem ser acessadas somente dentro dos locais em que foram criadas (a menos que tenham recebido um comando dizendo o contrario).

Aqui faz-se importante incluir a diference entre os 3 tipos de declaração de variáveis:

* var = sofre hoisting ao ser declarada porem apenas a variável, o valor associado não, gerando um ‘undefined’ caso tente chamar ela antes da atribuição; fica presa ao escopo criada (function scope), porem seu valor pode ser alterado; pode ser redeclarada.
* let = sofre hoisting porem não carrega o valor undefined, se fazendo necessário atribuir ao criar; blocked scope, isto é, a variável não pode sair do escopo em que foi criada, nem seu valores associados, por exemplo, variáveis criadas com o let dentro de condicionais ou loops são “destruidas” ao sair do loop sem deixar vestígios; não é possível redeclarar.
* const = sofre hoisting mas o valor não é inicializado (undefined); tem blocked scope; não pode ser redeclarada; O VALOR NÃO PODE SER ALTERADO, se ela começar apontando para um valor, ela sempre apontara para tal valor.

Aqui também se faz interessante mostrar a diferença entre *implementação* e *interface*. Interface são as propriedades e as funções que você quer que o usuário acesse. Implementação é como o código foi criado e como ele funciona por tras dos panos.

Escondendo informações do usuário:

Podemos observar então que é importante manter as variáveis “sensíveis” dentro de funções, para que o usuário não possa alterar elas mas ainda consiga acessar as funcionalidades (interface) do programa.

Veja o exemplo de um código de um contador não seguro, ouse já, o usuário pode alterar o valor do contador a vontade:

// Contador nao seguro

var counter = 0; //Sendo a variavel global, o usuario pode acessa-la e altera-la

var count = function () {

  counter = counter + 1;

  return counter;

};

Agora veja um exemplo de um contador seguro, estando encapsulado em uma função:

//Contador seguro

var getCounter = function () {

    var counter = 0; //Agora a variavel count está dentro de uma funcao e nao pode ser acessada no escopo global, nao podendo ser alterada pelo usuario

    var countUpBy1 =  function () {

      counter = counter + 1;

      return counter;

    };

    return countUpBy1;

  };

  var count = getCounter();

Neste exemplo, é como se fosse criado um mini-escopo global (escopo local) para se trabalhar a variável ‘counter’, deixando assim o código mais seguro!

A ideia geral então é sempre criar um escopo local quando se quer esconder informações do usuário que, se acessadas, podem ser usadas de maneira incorreta, como por exemplo no código a seguir, que caso estivessem no escopo global ou mesmo dentro de uma variável, o usuário poderia definir as respostas para as perguntas, veja:

//Escondendo propriedades em escopos locais

var getQuiz = function () {

    var qIndex = 0;

    var questions = [

        {

            question: "What is the highest mountain in the world?",

            answer: "Everest"

        },

        {

            question: "What is the highest mountain in Scotland?",

            answer: "Ben Nevis"

        },

        {

            question: "How many munros are in Scotland?",

            answer: "284"

        }

    ];

    return {

        quizMe : function () {

            return questions[qIndex].question;

        },

        showMe : function () {

            return questions[qIndex].answer;

        },

        next : function () {

            qIndex = qIndex + 1;

            return "Ok";

        },

        showIndex : function () {

            console.log("You're in question number: " + qIndex)

    }

    };

};

var quiz = getQuiz();

Uma vez que a variável ‘quiz’ é apenas uma variável que aponta para o valor de retorna da função ‘getQuiz()’ ela não possui as propriedades ‘questions’ ou mesmo ‘qIndex’, o que impede o usuário de alterar seus valores, é só quando a função em questão é chamada, através da variável ‘quiz’ que podemos executar os métodos pertinentes à função.

Sendo assim, podemos dizer que quando uma variável é atrelada à chamada de uma função, o usuário tem acesso apenas aos métodos mas não as variáveis locais utilizadas por estes métodos inerentes a função atrelada à variável.

Agora observe o código a seguir e tente descobrir a diferença:

// The Player constructor – user can manipulate player info

var Player = function (name, health) {

    var newLine = spacer.newLine();

    this.name = name;

    this.health = health;

    this.items = [];

    this.place = null;

    this.addItem = function (item) {

      this.items.push(item);

    };

    this.getNameInfo = function () {

      return this.name;

    };

    this.getHealthInfo = function () {

      return this.name + " has health " + this.health;

    };

    this.getPlaceInfo = function () {

      return this.name + " is in " + this.place.title;

    };

    this.getItemsInfo = function () {

      var itemsString = "Items:" + newLine;

      this.items.forEach(function (item, i) {

        itemsString += "   - " + item + newLine;

      });

      return itemsString;

    };

    this.getInfo = function (character) {

      var place = this.getPlaceInfo();

      var health = this.getHealthInfo();

      var longest = Math.max(place.length, health.length) + 4;

      var info = spacer.box(this.getNameInfo(), longest, character);

      info += spacer.wrap(place, longest, character);

      info += newLine + spacer.wrap(health, longest, character);

      info += newLine + spacer.line(longest, character);

      info += newLine;

      info += "  " + this.getItemsInfo();

      info += newLine;

      info += spacer.line(longest, character);

      info += newLine;

      return info;

    };

    this.showInfo = function (character) {

      console.log(this.getInfo(character));

    };

  };

E agora compare com este código:

// The Player constructor - user info hidden

var Player = function (name, health) {

    var newLine = spacer.newLine();

    var items = [];

    var place = null;

    var getNameInfo = function () {

        return name;

    };

    var getHealthInfo = function () {

        return "(" + health + ")";

    };

    var getItemsInfo = function () {

        var itemsString = "Items:" + newLine;

        items.forEach(function (item, i) {

            itemsString += "   - " + item + newLine;

        });

        return itemsString;

    };

    var getTitleInfo = function () {

        return getNameInfo() + " " + getHealthInfo();

    };

    var getInfo = function () {

        var info = spacer.box(getTitleInfo(), 40, "\*");

        info += "  " + getItemsInfo();

        info += spacer.line(40, "\*");

        info += newLine;

        return info;

    };

    this.addItem = function (item) {

        items.push(item);

    };

    this.setPlace = function (destination) {

        place = destination;

    };

    this.getPlace = function () {

        return place;

    };

    this.showInfo = function (character) {

        console.log(getInfo(character));

    };

};

Percebe-se que setar as variáveis com o ‘this.variavel’ elas ficarão acessíveis para o usuário agora se você setar as variáveis com o ‘this.’ O usuário pode acessar. Isso acontece porque as variáveis criadas dentro da função usando var/let/const não podem sair do escopo dela agora as variáveis criadas com o ‘this.’ Se tornam “métodos” do objeto e aí o usuário pode acessar.

No código a seguir vemos um exemplo mais direto da situação demonstrada anteriormente, onde é possível acessar a propriedade ‘name’ e ‘age’ mas não o ‘pinNumber’, o qual só é acessível mediante a função ‘showPin()’

//segunraça de dados - parametros dentro de funcoes

var Person = function(name, age, pin = 123456) {

    this.name = name;

    this.age = age;

    var pinNumber = pin;

    this.showPin = function() {

      return 'pin: ' + pinNumber;

    }

  }

  var leo = new Person('leo', 25);

CONDIÇÕES:

Até agora os programas escritos como exemplos no livro tinham apenas um caminho a seguir, a partir de agora adicionaremos caminhos utilizando condições, os chamados ‘if’s statements.

O ‘if’ indica que o programa deve comparar um ou mais valores a uma condição pre-estabelecida a partir daí o programa retorna e/ou executa um seguimento de código dependendo se a comparação foi verdadeira ou falsa.

//Condicoes

let hello = 'Hello'

if (hello === 'Hello'){

    console.log('Hello World!')

}

No exemplo, se a condição estabelecida, no caso se a variável ‘hello’ aponta para o valor ‘Hello’, for verdadeira o trecho entre {} é executado.

Isto sempre funcionará desta forma, ou seja, se a condição for verdadeira o código identado é executado.

//Condicoes

let hello = 'Helo'

if (hello === 'Hello'){

    console.log('Hello World!')

} else {

    console.log('Not hello to you, sir')

}

Ainda existe o comando ‘else’, ele executa caso o(s) ‘if’ não execute. No exemplo acima, já que a variável hello não é igual a ‘Hello’ ele escreve no console a frase ‘Not hello to you, sir’.

Math.random():

Este método é responsável por gerar valores aleatórios entre zero e um. Veja um exemplo de como usar o Math.random() e um condição ’if’ e ainda usando o Math.floor, que é usado para arredondar para baixo o numero gerado pelo Math.random():

//Adivinhe numero

var guessTheNumber = function() {

    let rightNumber = Math.floor(Math.random() \* 10);

  //   console.log(rightNumber);

    return function(guessedNumber) {

      if (guessedNumber === rightNumber) {

        console.log('You guessed correctly!');

      } else {

        console.log('Wrong number, try again.');

      }

    };

  };

  var guess = guessTheNumber();

Estruturando o código da maneira como aprendemos anteriormente, é possível esconder a variável com o resultado dentro da função ‘guessTheNumber()’, desta maneira o usuário não tem acesso ao valor correto, então associamos uma função que escreve no console caso o usuário acerte o valor determinado pelo código ao ‘return’ desta função.

Este jogo ainda pode ser melhorado utilizando mais clausulas ‘if’s’ para que o usuário tenha um feedback caso tenha tentado advinhar um numer muito alto ou muito baixo.

//Adivinhe numero - com feedback

var guessTheNumber = function() {

    let rightNumber = Math.floor(Math.random() \* 10);

  //   console.log(rightNumber);

    return function(guessedNumber) {

      if (guessedNumber === rightNumber) {

        console.log('You guessed correctly!');

      } else if (guessedNumber > rightNumber) {

        console.log('Wrong. Number too high.');

      } else if (guessedNumber < rightNumber) {

        console.log('Wrong. Number too low.');

      }

    };

  };

  var guess = guessTheNumber();

Mudando o foco um pouco do ‘if/else’, perceba que no exemplo acima escrevemos uma função que tem apenas a funcionalidade de gerar um numero aleatório de 0 a 10 e então no seu valor de retorno escrevemos a função que verifica um numero digitado. Isto é interessante porque nos abre a possibilidade de enxergar o bloco return como uma extensão do bloco da função.

Diferenca entre === e ==:

O primeiro ‘===’ diz respeito a estritamente igual, isto é, 7 === 7 é igual a true, porem, 7 === “7” é false. Agora com == 7 é igual a “7”.

Voltando às condições e às variáveis escondidas nas funções:

//no cheating quiz

var getQuiz = function() {

    var score = 0,

      qIndex = 0,

      inPlay = true,

      questions,

      next,

      getQuestion,

      checkAnswer,

      getHint,

      submit;

    questions = [{

        question: "What is the highest mountain in the world?",

        answer: "Everest",

        hint: "\_\_\_rest"

      },

      {

        question: "What is the highest mountain in Scotland?",

        answer: "Ben Nevis",

        hint: "\_en \_\_vis"

      }

    ];

    next = function() {

      qIndex = qIndex + 1;

      if (qIndex >= questions.length) {

        inPlay = false;

        console.log("You have finished the quiz.");

      }

    };

    getQuestion = function() {

      if (inPlay) {

        return questions[qIndex].question;

      } else {

        return "You have finished the quiz.";

      }

    };

    checkAnswer = function(userAnswer) {

      if (userAnswer === questions[qIndex].answer) {

        console.log("Correct!");

        score = score + 1;

      } else {

        console.log("No, the answer is " + questions[qIndex].answer);

      }

    };

    getHint = function() {

      return console.log(questions[qIndex].hint);

    };

    submit = function(userAnswer) {

      var message = "You have finished the quiz.";

      if (inPlay) {

        checkAnswer(userAnswer);

        next();

        message = "Your score is " + score + " out of " + qIndex;

      }

      return message;

    };

    return {

      quizMe: getQuestion,

      submit: submit,

      helpMe: getHint

    };

  };

  var quiz = getQuiz()

Características de um bom código:

Perceba mais uma vez que agora neste exemplo do quiz, temos todas as variáveis (que ficam invisíveis ao usuário) declaradas no inicio da função com um único ‘var’, após isso seguem os códigos para as respectivas variáveis, então as variáveis que deseja-se que sejam utilizadas pelo usuário são passadas a uma interface para o usuário pelo bloco ‘return’ (interface de objeto).

Além disso, temos uma variável ‘inPLay’ que diz ao programa se ainda é possível jogar, baseado em qual questão o jogador está, se estiver na ultima (qIndex >= questions.length(); inPlay = false) o jogo emitira escreverá na tela que o jogo já acabou.

Ele também usa a variável ‘qIndex’ para saber qual questão/resposta/dica do jogo acessar, e consequentemente, se o jogo acabou ou não.

Perceba que em todos os momentos usamos um paradigma muito importante de se comtemplar em linguagens orientadas a objetos, cada função faz exatamente apenas UMA coisa; o método ‘submit’, por exemplo, foi criado especificamente para utilizar os métodos ‘checkAnswer’ e o ‘next’, que por sua vez foram criados apenas para checar a resposta e para ir para a próxima pergunta, respectivamente. Isto é importante para poder mandar o fluxo de leitura do cód1igo mais simples e mais “acompanhável”.

Continuando com o jogo – interface do usuário: ‘go’ e ‘get’:

Agora inicia-se a implementação dos dois métodos iniciais usados no jogo, estes dois métodos serão responsáveis por fazer o jogador se mover para uma direção e o outro para pegar os itens de uma área. Para isso, inicialmente criamos o seguinte código dentro do bloco return do programa:

return {

    go: function(direction) {

      var place = player.getPlace();

      var destination = place.getExit(direction);

      player.setPlace(destination);

      render();

      return "";

    },

    get: function() {

      var place = player.getPlace();

      var item = place.getLastItem();

      player.addItem(item);

      render();

      return "";

    }

  };

Ele parece funcionar corretamente porém se o jogador digitar alguma direção inválida dentro do console ele irá quebrar e assinalar o valor ‘undefined’ para a variável ‘place’ e tudo irá por água abaixo.

Para contornar esta situação utilizamos o bloco IF que aprendemos:

go: function (direction) {

    var place = player.getPlace();

    var destination = place.getExit(direction);

    if (destination !== undefined) {

    player.setPlace(destination);

    render();

    return "";

    } else {

    return "\*\*\* There is no exit in that direction \*\*\*";

    }

    },

Deste modo o programa só fará a alteração do a variável ‘place’ caso o valor seja diferente de ‘undefined’.

O mesmo é válido para o método ‘get’.

MÓDULOS:

Até agora todo o código do jogo estava em apenas um único arquivo, mas conforme os códigos crescem em tamanho é interessante fazê-los ficarem em diferentes arquivos, para que seja melhor organizado.

Para poder importar outros módulos em um determinado script primeiro deve-se editar o arquivo HTML para que ele contenha na tag do script que está sendo carregado o código [type=”module”]:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script type="module" src="./getProgrammingWithJS.js"></script>

    <title>Aprendendo a programar.</title>

</head>

</html>

OBS: apenas o arquivo que servira de “HUB” de importação de módulos deve ser citado.

Então agora temos algumas opções de como realizar a importação [<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/import>] mas antes de importarmos qualquer coisa de um módulo, precisamos declarar que queremos exportar uma determinada função, por isso, no arquivo que desejamos pegar a função devemos fazer o seguinte:

//preparando a função 'foo' para ser exportada:

function foo (){

    return ('fooncionou');

}

function bar (n){

    return n \* 2;

}

export {foo}; //usamos o export {objeto-a-ser-exportado} para declaramos que ele será exportado

Então temos duas opções para importar:

* Importa-se todas as funções do modulo desejado que tem a tag ‘export’:

import \* as utils from "./utils.js"; //sintaxe para importar tudo que veio do outro modulo.

var a = utils.foo(); //acesso a função usando o nameSpace criado acima com dotNotation.

var b = utils.bar(2);

console.log(a); // 'fooncionou'

console.log(b); // 4

* Importa-se apenas uma função especifica:

import { foo } from "./utils.js"; //sintaxe para importar um objeto especifico de outro modulo

var a = foo(); //temos acesso direto a função.

console.log(a); // 'fooncionou'

Agora é possível utilizar vários módulos para organizar melhor os scripts criados. Faremos isso então para um programa que roda um Quiz, veja:

No arquivo utils.js temos:

function random (lower, higher){

    let range = higher - lower;

    return lower + Math.floor(Math.random() \* range);

}

export {random};

e no arquivo quiz.js, temos:

import { random } from "./utils.js";

function getQuiz (){

    let i = 0;

    let counter = 0;

    let score = 0;

    const questions = [

    { question: "7 x 8", answer: "56" },

    { question: "12 x 12", answer: "144" },

    { question: "5 x 6", answer: "30" },

    { question: "9 x 3", answer: "27" }];

    const checkAnswer = function (userAnswer){

        if (counter >= questions.length){

            return 'The quiz is already over, pal.';

        }else{

            if (userAnswer == questions[i].answer){

                score++;

                return 'Congratulations! Your answer was correct, your score now is ' + score;

            }else{

                return 'Shame on you! Your input was the incorrect one, TRY HARDER NEXT TIME!';

            }

        }

    };

    const getQuestion = function (){

        counter++;

        i = random (0, questions.length);

        console.log('Question number: ' + (i + 1));

        return questions[i].question;

    };

    const getScore = function (){

        return 'Your score is: ' + score + '. You still have ' + (questions.length - counter) + ' tries';

    };

    return{

        quizMe: getQuestion(),

        submit: checkAnswer,

        myScore: getScore()

    };

};

window.quiz = getQuiz(); // fazemos isto para criar uma variavel 'quiz' no escopo global par apdoer ser acessada no console para podermos executar os comandos da função