Instruções para a entrega: entregar as respostas na folha de respostas na aula do dia 27/ago.

1 – Na linguagem Javascript a palavra reservada `this` refere-se ao objeto ao qual ele pertence. Para mais detalhes acesse:

https://www.w3schools.com/js/js_this.asp

Analise o trecho de código a seguir e marque a alternativa incorreta.

```
<!DOCTYPE html>
<html>
  <body>
    <p id="saidaa"></p>
    <p id="saidab"></p>
    <p id="saidac"></p>
    <p id="saidad"></p>
    <p id="saidae"></p>
    <button onclick="d(this)">Letra d</button>
    <button id="regular">Função regular</button>
    <button id="arrow">Arrow function</button>
    <script>
      a(); /* letra (a) */
      document.getElementById("saidab")
              .innerHTML = this;  /* letra (b) */
      let carro ={
        nome:'uno',
        print: function(){
           document.getElementById("saidac")
                .innerHTML = this; /* letra (c) */
        }
      };
      carro.print();
      function a(){
        document.getElementById("saidaa")
                .innerHTML = this;
      }

      function d(obj){
        document.getElementById("saidad")
                .innerHTML = obj; /* letra (d) */
      }

      regular = function() {
        document.getElementById("saidae")
                .innerHTML = this;
      };

      arrow = () => {
        document.getElementById("saidae")
                .innerHTML = this;
      };
      /* letra (e) */
      document.getElementById("regular")
              .addEventListener("click", regular);
      document.getElementById("arrow")
              .addEventListener("click", arrow);
    </script>
  </body>
</html>
```

(a) No corpo da função a() this refere-se ao objeto global, isto é, a toda a toda a página.

(b) O objeto this no corpo do Javascript refere-se ao objeto global, isto é, a toda a toda a página.

(c) O objeto this no corpo do método print refere-se ao objeto carro, isto é, o objeto que contém o método.

(d) O objeto this em um manipulador de evento refere-se ao próprio elemento HTML, nesse exemplo, refere-se ao botão.

(e) A chamada dos métodos regular e arrow produz o mesmo resultado.

2 – A linguagem Javascript atualmente está na versão ES6, conhecida por ECMAScript 6 ou ECMAScript 2015 (https://www.w3schools.com/js/js_es6.asp). Essa versão passou a suportar a definição de classes, mas na prática as classes são tipos de function.

As propriedades/atributos da classe são definidas no construtor. Para mais detalhes sobre classes acesse:

https://www.w3schools.com/js/js_classes.asp

Marque a alternativa que possui um código com erro.

(a)
```
class Msg{
   print(){
      alert("Bom dia");
   }
}
```

```
    new Msg().print();
(b) class Carro{
       constructor(modelo, ano){
          this.modelo = modelo;
          this.ano = ano;
       }

       print(){
         alert(this.modelo + " " + this.ano);
       }
    }
    let c = new Carro('gol',2001);
    c.ano += 1;
    c.print();
(c) class Cliente{
      const status = 'gold';
    }
    let c = new Cliente();
    alert(c.status);
(d) class Chefe{
       static print(){
          alert("oi");
       }
    }
    Chefe.print();
(e) class Produto{
       constructor(nome){
         this.nome = nome;
       }

       get _nome(){
         return this.nome;
       }
       set _nome(nome){
         this.nome = nome;
       }
    }
    let p = new Produto('arroz');
    p._nome = 'feijão';
    alert(p._nome);
```

3 – A linguagem Javascript possui suporte para o tratamento de erros em tempo de execução. Veja como exemplo o código a seguir:

```
<!DOCTYPE html>
<html>
<body>
<p id='saidaa'></p>
<p id='saidab'></p>
<script>
try {
  funcao("oi");
}
catch(e) {
  document.getElementById("saidaa").innerHTML    =
e.message;
}

try{
  throw "mensagem de erro";
}
catch(e) {
  document.getElementById("saidab").innerHTML = e;
}
finally{
  document.getElementById("saidab").innerHTML += "
finally";
}
</script>
</body>
</html>
```

Para mais detalhes acesse:

https://www.w3schools.com/js/js_errors.asp

As exceções são objetos do tipo Error na linguagem Javascript e possui as propriedades name e message. Os objetos do tipo Error pode ser de um dos seguintes tipos: EvalError, RangeError, ReferenceError, SyntaxError, TypeError e URIError.

Marque a alternativa cujo erro não é do tipo indicado.

(a) A chamada de uma função que não existe irá lançar uma exceção do tipo ReferenceError.

(b) O uso de uma variável para leitura que não foi declarada irá lançar uma exceção do tipo ReferenceError, por exemplo, x = y * 2;

(c) Acessar uma propriedade ou método de uma variável cujo o conteúdo não possui esse membro gera uma exceção do tipo TypeError, por exemplo, var a = 10; console.log(a.length());

(d) Acessar uma posição inexistente em um array lança uma exceção do tipo RangeError.

(e) Podemos receber códigos de terceiros e estes podem ter erros de digitação que serão percebidos apenas

durante a execução. Este tipo de erro lança a exceção do tipo SyntaxError, por exemplo, eval('2 + *x');

4 – Os eventos estão associados aos elementos HTML e são disparados a partir do navegador ou através da interação do usuário. Para mais detalhes acesse:
https://www.w3schools.com/js/js_events.asp
Marque a alternativa correta com relação aos eventos HTML.

(a) O evento onload é disparado após a página ser carregada, então ele deve estar na última marcação do body.

(b) O evento onload é disparado imediatamente após todas as marcações da página serem interpretadas pelo navegador e ele deverá estar na chamada da marcação <body>.

(c) As propriedades que são eventos são inicializadas com o prefixo "on". O valor da propriedade pode ser código Javascript, CSS ou HTML.

(d) O evento onchange deve estar associado a marcação <body> desta forma qualquer alteração no corpo da página irá disparar esse evento.

(e) Os eventos são marcações associadas a códigos Javascript.

5 – A linguagem Javacript possui alguns métodos para operar sobre array, entre eles, tem-se o método foreach para operar sobre cada elemento do array. Para mais detalhes acesse:
https://www.w3schools.com/js/js_array_iteration.asp
Marque a alternativa que apresenta **erro**, isto é, não faz aquilo que se espera.

(a)
```
var saida = "", v = [3, 8, 2, 5];
v.forEach(print);
alert(saida);
function print(value, index) {
  saida += index +":"+ value +"\n";
}
```

(b)
```
var saida = 0, v = [3, 8, 2, 5];
v.forEach(soma);
alert(saida);
function soma(value) {
  saida += value;
}
```

(c)
```
var min = Number.MAX_SAFE_INTEGER,
    v = [3, 8, 2, 5];
v.forEach(minimo);
alert(min);
function minimo(value, index, array) {
  min = min > value? value : min;
}
```

(d)
```
var aux = [], v = [3, 8, 2, 5];
v.forEach(copy);
alert(aux);
function copy(value, index) {
  aux[index] = value;
}
```

(e)
```
var v = [3, 8, 2, 5];
v.forEach(dobrar);
alert(v);
function dobrar(value, index, array) {
  value = value * 2;
}
```

6 – Assim como o método foreach, os métodos map, filter, reduce, every e some são usados para operar sobre um array. Marque a alternativa que apresenta **erro**.

(a)
```
var v = [3, 8, 2, 5];
var aux = v.every(impar);
alert(aux == true);
function impar(value) {
  return value % 2 == 1;
}
```

(b)
```
var v = [3, 8, 2, 5];
var aux = v.some(impar);
alert(aux == true);
function impar(value) {
  return value % 2 == 1;
}
```

(c)
```
var v = [3, 8, 2, 5];
var aux = v.filter(impar);
alert(aux);
function impar(value) {
  return value % 2 == 1;
}
```

(d)
```
var v = [3, 8, 2, 5];
var aux = v.reduce(somar);
alert(aux);
function somar(total, value) {
  return total + value;
}
```

(e)
```
var v = [3, 8, 2, 5];
var aux = v.map(dobrar);
alert(aux);
```

```
function dobrar(value, index, array) {
  return value * 2;
}
```

7 – Marque a alternativa que não retorna o somatório do IMC (Índice de Massa Corporal) dos elementos do array lista. Para mais detalhes acesse:

```
var lista = [{nome:'Pedro',peso:72,altura:1.7},
             {nome:'Ana',peso:58,altura:1.6}];
```

(a)
```
function imc(a){
  return a.reduce((acc,cur) => acc +
          (cur.peso/Math.pow(cur.altura,2)), 0);
}
console.log( imc(lista) );let a = {
```

(b)
```
function calcImc(total,obj){
  return total + obj.peso/Math.pow(obj.altura,2);
}
console.log( lista.reduce(calcImc, 0) );
```

(c)
```
function somaImc(lista){
 let s = 0;
 for(let i =0; i < lista.length; i++){
   s += lista[i].peso/Math.pow(lista[i].altura,2);
 }
 return s;
}
console.log(somaImc(lista));
```

(d)
```
function sImc(total,obj){
  return total + obj.peso/Math.pow(obj.altura,2);
}
console.log( lista.map(sImc) );
```

(e)
```
function somarImc(obj){
  s += obj.peso/Math.pow(obj.altura,2);
}
var s = 0;
lista.forEach(somarImc);
console.log(s);
```

8 – O código a seguir retorna cada elemento do array com o sufixo Sr.

```
var nomes = ['Pedro','João','Paulo'];
var r = nomes.map(function(nome){
   return `Sr. {nome}`;
});
console.log(r);
```

Observação: template strings são usadas para incorporar expressões que envolvem o uso de variáveis, onde variável fica dentro da string delimitada por backquotes `.

As Arrow Functions são usadas para escrever notações mais concisas e enxutas. Marque a alternativa que produz o mesmo resultado da função anterior usando Arrow Function.

(a)
```
var m = nomes.map((nome) => `Sr. {nome}`);
console.log(m);
```

(b)
```
var n = nomes.map((nome) => return `Sr. {nome}`);
console.log(n);
```

(c)
```
var o = nomes.map((nome) => {`Sr. {nome}`});
console.log(o);
```

(d)
```
var p = nomes.map(function(nome) => `Sr. {nome}`);
console.log(p);
```

(e)
```
var q = nomes.map(function(nome) => {return `Sr.
{nome}`});
console.log(q);
```

9 – Marque a alternativa que faz uso da função reduce e Arrow Function para calcular o somatório dos elementos do array lista.

```
var lista = [3, 8, 2, 5];
```

(a)
```
console.log( v.reduce(function(total,value) =>
total+value) );
```

(b)
```
var total = 0;
console.log( v.reduce((value) => total+value) );
```

(c)
```
var total = 0;
console.log( v.reduce(value => return total+value) );
```

(d)
```
console.log( v.reduce((total,value => return
total+value) );
```

(e)
```
console.log( v.reduce((total,value) => total+value) );
```

10 – Promisse é um objeto usado para realizar processamentos assíncronos - isto é, em requisições no servidor - esse objeto guarda um valor que pode estar disponível agora, no futuro ou nunca (já que a conexão pode apresentar falha). Isso permite o tratamento de eventos ou ações que acontecem de forma assíncrona em casos de sucessos ou falhas. Uma operação no servidor é assíncrona, pois não sabemos quando o servidor irá responder.

Para mais detalhes acesse:

https://blog.matheuscastiglioni.com.br/trabalhando-com-promises-em-javascript/

Marque a alternativa que faz a chamada da função testar exibir numa mensagem de alerta o texto “Acima do limite”.

Observação: a função setTimeout é usada para esperar 2 seg.

```
function testar(nro) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if(nro > 10) {
          reject("Acima do limite");
      }
      resolve("Número válido");
       }, 2000);
    });
}
```

(a) testar(20);

(b) testar(20).catch(console.error);

(c) testar(20).catch(alert);

(d) testar().then(alert).catch(alert);

(e) testar(20).then(console.log).catch(console.error);