**Examen de conocimientos para desarrollar en Cells**

**Número de preguntas:** 25

**Tiempo:** 2 horas

Sección teórica:

1. Programación Orientada a Objetos 5 Preguntas - Fallos permitidos: 1
2. JavaScript 10 Preguntas - Fallos permitidos: 2
3. Polymer 2 10 Preguntas - Fallos permitidos: 3

90 min.

Sección practica – 1 ejercicio

30min.

**Tipo de examen:** Opción múltiple

Para obtener una nota aprobatoria de todo el examen debes aprobar cada una de las secciones, lo que implica que si una de las secciones rebasa su número máximo permitido de fallos automáticamente habrás reprobado el examen independientemente del hecho de que las secciones restantes no tengan error alguno.

El examen se ejecuta en pantalla completa de la cual no debes salir hasta terminar el examen, los cambios de pestaña o ventana son registrados en la plataforma así como la apertura de la consola de desarrollador, si dichos eventos son registrados el examen será anulado y por ende no acreditado.

# **Sección I - POO**

**1. ¿Qué principio de diseño favorece la Alta Cohesión?**

Elige 1 respuesta:

1. Single Responsibility
2. Composition over inheritance
3. Interface Segregation
4. Modularity

**2. ¿A qué se refiere SOLID?**

Elige 1 respuesta:

1. Se refiere a 5 buenas prácticas de desarrollo: Single Responsibility, Open Closed, Low coupling, Interface Segregation, Dependency Inversion
2. Se refiere a 5 principios de diseño: Segregation of Interfaces, Open Closed, Liskov Substitution, Interface programming, Dependency Inversion
3. Se refiere a 5 patrones de diseño: Singleton, Observer, Layers Pattern, Inversion of control, Data transfer object
4. Se refiere a 5 principios de diseño: Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation, Dependency Inversion

**3. ¿A qué se refiere el Principio de Segregación de Interfaces?**

Elige 1 respuesta:

1. A crear interfaces con un solo método
2. A no agregar métodos nuevos a una interface existente que ya se este usando
3. A forzar a los clientes de una interfaz a depender de ésta con métodos que no necesita
4. A que los métodos de la interfaz tengan un solo objetivo

**4. ¿Qué afirmaciones son correctas?**

Elige 3 respuestas:

1. Composition over Inheritance recomienda crear componentes de estructura compleja usando Composición en lugar de herencia.
2. El High Coupling y Low Cohesion es Malo
3. El High Coupling y Low Cohesion es Bueno
4. Encapsular un componentes es solo cambiar sus propiedades a privadas y métodos a públicos
5. Encapsular un componentes es ocultar detalles de implementación y solo permitir la interacción con el mediante métodos públicos

**5. ¿Qué Principio de Diseño asegura que todas las clases de implementación cumplirán su objetivo funcional aunque con algunas variantes de forma?**

Elige 1 respuesta:

1. Desarrollo basado en Interfaces
2. Principio de Liskov
3. Herencia de Clases
4. Pruebas unitarias

# **Sección II - JavaScript**

1. **¿Qué líneas permiten la creación de un objeto?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var o1 = {};  var o2 = Object.create({});  var o3 = new Object();  var o4 = {prop1: ‘value’, prop2: ‘value2’};  class MiClase{}  var o5 = new MiClase();  function MiFunConstructor () {}  var o6 = new MiFunConstructor(); |

Elige 1 respuesta:

1. 1, 2, 3, 4;
2. 1, 4, 6, 8
3. 1, 2, 3, 4, 6, 8
4. 3, 4, 6, 8
5. **¿Cómo se crea una función constructor?**

Elige 2 respuestas:

1. Se crea una función sin argumentos con el mismo nombre de la clase
2. Se crea una función con el nombre constructor dentro de una Clase
3. Se crea una función con la primera letra en mayúscula y al invocarse se antepone new
4. Con Object.create( { } );
5. **Al ejecutar las siguientes líneas de código ¿Qué resultado obtendremos?**

let nums = [30, 400, 100, 10, 1];

nums.sort();

console.log(nums);

Elige 1 respuesta:

1. [1, 10, 30, 100, 400]
2. [1, 10, 100, 30, 400]
3. [30, 400, 100, 10, 1]
4. **Que expresión regular permite validar un match exacto?**

Elige 1 respuesta:

1. Regexp.exactMatch(expression\_regular, cadena\_a\_probar)
2. /expression\_regular/i
3. /expression\_regular/gi
4. /^expression\_regular$/gi
5. **Si tenemos el siguiente fragmento de código:**

let fsumar = (a, b) => a + b;

console.log(fsumar);

console.log(typeof fsumar);

console.log(fsumar(1,2));

console.log(typeof fsumar(1,2));

Que se imprime en consola?

Elige 1 respuesta:

1. La definición de la función

“function”

3

“number”

1. La definición de la función

“arrow function”

3

“number”

1. Error de ejecución
2. **Dado el siguiente fragmento de código**

var o1 = { "emp":{ "name":"John", "age":30, "city":"NY" }};

var o2 = { emp:{ name:"John", age:30, city:"NY" }};

var o3 = "{\"emp\":{ \"name\":\"John\", \"age\":30, \"city\":\"NY\"}}";

console.log(typeof o1);

console.log(typeof o2);

console.log(typeof o3);

Elige 1 respuesta:

1. json, object, string
2. object, object, string
3. object, object, object
4. Error en la declaración de o3
5. **¿Cuál es la salida de la siguiente Promesa?**

(function() {

console.log("1. Inicio");

var promise = new Promise((resolver, rechazar) => {

console.log('2. Dentro de Promesa');

resolver();

});

promise.then(() => {

throw new Error('4. Algo falló');

console.log('5. Despues del Error');

}).catch(() => {

console.log('6. Manejando el error!');

}).then(() => {

console.log('7. Despues de la Tempestad');

});

let resolver = () => {

console.log("3. Procesando resultados");

}

})();

Elige 1 respuesta:

1. Error, rechazar no está definido
2. “1. Inicio”, “2. Dentro de promesa”, “3. Procesando Resultados”, “6. Manejando el error!”, “7. Despues de la Tempestad”
3. “1. Inicio”, “2. Dentro de promesa”, “3. Procesando Resultados”, “6. Manejando el error!”
4. “1. Inicio”, “2. Dentro de promesa”, “6. Manejando el error!”, “7. Despues de la Tempestad”
5. “1. Inicio”, “2. Dentro de promesa”, “3. Procesando Resultados”, “6. Manejando el error!”, “5. Despues del Error”, “7. Despues de la Tempestad”
6. ¿Cual es la salida del siguiente fragmento de código?

function x(value) {

var a = value || "Homero";

console.log("a: " + a);

}

x(); //? a: Homero

x(null); //? a: Homero

x("a"); //? a: a

x(false); //? a: Homero

Elige 1 respuesta:

1. undefined, “Homero”, “a”, “Homero”
2. null, “Homero”, “a”, “Homero”
3. “Homero”, “Homero”, “a”, “Homero”
4. Error en la llamada a x(false)
5. Cuál es la salida del siguiente fragmento de código?

‘use strict’;

fun();

varfun();

var varfun = function(){ console.log("OK"); };

function fun(){ console.log("OK"); }

fun ();

varfun();

Elige 1 respuesta:

1. OK OK OK OK
2. OK Error: varfun is not a function
3. OK Error: varfun is not a function OK OK
4. ¿ Cuál es la salida del siguiente fragmento de código?

//Funcion constructor de Person

function Person(name, age) { ... }

var persons = [];

persons.push(new Person("Ed", 31));

persons.push(new Person("John", 22));

persons.push(new Person("Peter", 44));

persons.push(new Person("Joseph", 12));

persons.push(new Person("Richard", 18));

persons.sort( (pa, pb) => { return pa.age > pb.age ? -1 : 1; });

var res = persons.filter( p => { return p.age > 18;});

console.log(res);

Elige 1 respuesta:

0: Person {name: "Peter", age: 44}

1: Person {name: "Ed", age: 31}

2: Person {name: "John", age: 22}

0: Person {name: "Peter", age: 44}

1: Person {name: "Ed", age: 31}

2: Person {name: "John", age: 22}

3: Person {name: "Richard", age: 18}

4: Person {name: "Joseph", age: 12}

0: Person {name: "John", age: 22}

1: Person {name: "Ed", age: 31}

2: Person {name: "Peter", age: 44}

0: Person {name: "Joseph", age: 12}

1: Person {name: "Richard", age: 18}

2: Person {name: "John", age: 22}

3: Person {name: "Ed", age: 31}

4: Person {name: "Peter", age: 44}

# **Sección III- Polymer**

1. Se ha creado un componente ContactList extends Polymer.Element{} que contiene los métodos:

constructor(), ready(),

attributeChangedCallback(), connectedCallbackEvent()

El componente se ha incluido correctamente a una página mediante

<contact-list></contact-list>

La secuencia de ejecución de los métodos al cargar la página será:

1. constructor, ready y posiblemente connectedCallbackEvent
2. constructor, ready y posiblemente attributeChangedCallback
3. constructor, ready
4. constructor, ready, connectedCallbackEvent, attributeChangedCallback
5. ¿Qué afirmaciones son ciertas en Polymer?
6. Attributes goes down, Events goes up
7. El shadow dom es accessible desde fuera del componente solo si se usa javascript
8. Los atributos se inicializan desde el constructor y las propiedades se desde el Tag
9. Los nombres de componentes deben llevar al menos un ‘-‘ para evitar colisiones con tags de HTML
10. ¿ Qué líneas modifican a stkarray de forma que se actualice cualquier data binding que apunte al arreglo?

this.stkarray = [];

let s = new SofttekianDM('Benito', "BEN1", "benito@softtek.com");

1. this.stkarray.push(s);
2. this.stkarray.unshift(s);
3. this.pop();
4. this.push(‘stkarray’, s);
5. this.stkarray[this.stkarray.length] = s;
6. Que cambios se necesitan para que cuando se modifique arreglo genere un evento arreglo-changed y pueda ser escuchado desde document?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | static get properties() {  return {  arreglo: {  type: Array,  value: [],  readOnly : false,  notify: true,  bubbles : false,  composed: false  }  }; |

1. El código funciona tal como esta
2. Cambiar el valor de las líneas 8, 9. Y con document crear un listener para el evento.
3. Crear un CustomEvent pasándole el nombre del evento y los datos en detail y dispararlo
4. No es posible crear un evento arreglo-changed que pueda llegar hasta document
5. Acerca de Data bindings ¿Cual de las siguientes configuraciones puede usa {{}} para recuperar y actualizar valores?

|  |  |  |
| --- | --- | --- |
|  |  |  |
| A | notify: false,  readOnly: true |  |
| B | notify: false,  readOnly: false |  |
| C | notify: true,  readOnly: true |  |
| D | notify: true,  readOnly: false |  |

1. Dado el siguiente fragmento de código

<template>

<iron-input id=”txtNombre”><input value=”nombre” /></iron-input>

</template>

¿Que línea permite tener referencia al elemento txtNombre?

1. let c = this.$.txtNombre;
2. let c = Polymer.dom(this.root).querySelector("#txtNombre");
3. let c = this.shadowRoot.querySelector('txtNombre);
4. let c = document.getElementById("txtNombre");

# **Sección IV- Práctica**

Realizar el siguiente ejercicio en Plunker <https://plnkr.co/edit/?p=catalogue>

Tiempo: 30 minutos

Entregar 2 archivos

* index.html
  + Link a Polymer:

<script src="https://polygit.org/polymer+^v2.6.0/webcomponentsjs+^v1.3.0/components/webcomponentsjs/webcomponents-loader.js"></script>

* parser-tuis.html
  + Link a Componentes de Polymer:

<link rel="import" href="https://polygit.org/polymer+^v2.6.0/webcomponentsjs+^v1.3.0/components/polymer/polymer-element.html">

<link rel="import" href="https://polygit.org/polymer+^v2.6.0/webcomponentsjs+^v1.3.0/iron-\*+polymerelements+^v2.0.0/components/iron-input/iron-input.html">

Esta pantalla recibe una caden Json, la parsea a un objeto javascript y la imprime en consola

 <parser-jlil>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | **JSON PARSER** | |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Cadena JSON |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | Parsear |  | Limpiar |  |
|  |  |  |  |  |  |  |

Ejemplo:

Cadena JSON:

{ "employee":{ "name":"John", "age":30, "city":"New York" }};

        En consola se imprime:

        { employee:{ name:"John", age:30, city:"New York" }}