Examen PW Enero 2025

TIPO TEST

1. ¿Cuál de las siguientes afirmaciones es cierta acerca de las peticiones HTTP?

   a) El método GET es idempotente.

   b) El método POST no devuelve un cuerpo.

   c) El método OPTIONS devuelve el cuerpo del recurso.

   d) Ninguna de las anteriores es correcta.

   Explicación: El método GET es idempotente porque realizar la misma solicitud varias veces produce el mismo resultado sin modificar el estado del servidor. En cambio, POST no es idempotente porque puede modificar el estado en cada solicitud.

2. ¿Cuál es una ventaja de los "Servicios REST" en comparación con los "Servicios SOAP"?

   a) Ofrecen una semántica mejor definida con operaciones fuertemente tipadas.

   b) Son adecuados para aplicaciones que requieren un alto desempeño.

   c) Comunican múltiples formatos y tipos de medios debido a su estructura flexible de mensajes.

   d) Ninguna de las anteriores es correcta.

   Explicación: REST es más flexible que SOAP ya que permite transmitir datos en formatos como JSON, XML, HTML, etc. SOAP, en cambio, usa XML exclusivamente y es más rígido.

3. En un patrón de microservicios, ¿cuál es el principal desafío al implementar una transacción que abarque múltiples servicios?

   a) La gestión de la consistencia de datos entre servicios.

   b) La sincronización de los distintos lenguajes en los que se implementa.

   c) La autenticación entre servicios.

   d) Ninguna de las anteriores es correcta.

   Explicación: En microservicios, la consistencia de datos es un desafío porque cada servicio maneja su propia base de datos. Para transacciones, se usan patrones como Saga para mantener la coherencia sin afectar la escalabilidad.

4. En el contexto de los web frameworks, ¿cómo beneficia la inversión de control (IoC) al desarrollo de aplicaciones web?

   a) Permite un mayor control sobre el protocolo de red utilizado.

   b) Simplifica el manejo de dependencias y reduce el acoplamiento entre componentes.

   c) Incrementa la velocidad de procesamiento del servidor web.

   d) Ninguna de las anteriores es correcta.

   Explicación: IoC permite que el framework controle la creación y gestión de dependencias, lo que mejora la modularidad y facilita pruebas unitarias.

5. ¿Qué significa el principio de robustez de Jon Postel en el diseño de software?

   a) Que los programas deben requerir entradas estrictamente definidas mientras producen una variedad de salidas.

   b) Que los programas deben aceptar la mayor variedad de entradas posible y limitar la salida a formatos bien definidos.

   c) Que los programas deben ser flexibles tanto en las entradas como en las salidas.

   d) Ninguna de las anteriores es correcta.

   Explicación: Este principio dice que un sistema debe ser tolerante a errores en la entrada pero estricto en la salida, mejorando la interoperabilidad.

6. ¿Qué define un tipo de medios?

   a) El formato válido de un elemento multimedia.

   b) La naturaleza y formato de un recurso.

   c) Valida el tipo de extensión del fichero recibido desde el servidor.

   d) Ninguna de las anteriores es correcta.

   Explicación: El tipo de medios (MIME type) describe el formato de un recurso, como `text/html`, `image/png` o `application/json`.

7. ¿Cuál de los siguientes puertos se considera un puerto de sistema?

   a) 10433

   b) 3306

   c) 80

   d) Ninguna de las anteriores es correcta.

   Explicación: Los puertos de sistema van de 0 a 1023 y están reservados para servicios estándar. El puerto 80 es el predeterminado para HTTP.

8. ¿Cuál de los siguientes crees que es un desafío común al implementar Single Page Applications (SPA)?

   a) Manejo de la navegación y el historial del navegador.

   b) Uso de múltiples hojas de estilo.

   c) Simplificación del back-end.

   d) Ninguna de las anteriores es correcta.

   Explicación: Las SPA manejan la navegación sin recargar la página, por lo que gestionar el historial del navegador correctamente es un reto.

9. ¿Qué imprime este código?

```
const objeto = {
    valor: 42,
    metodo: () => {
        return this.valor;
    }
};
console.log(objeto.metodo());
```

   a) 42
   b) `undefined`
   c) Error: `this.valor` no está definido
   d) Ninguna de las anteriores es correcta.

   Explicación: Las funciones flecha no tienen su propio `this`, por lo que `this.valor` se evalúa como `undefined`.

10. ¿Qué imprime este código?

```
function sumar(a, ...resto) {
    return a + resto.length;
};
console.log(sumar(1,2,3,4));
```

   a) 4
   b) 5
   c) Error: `resto` no es un Array.
   d) Ninguna de las anteriores es correcta.

   Explicación:

   - `a` toma el valor 1.
   - `resto` toma los valores `[2,3,4]`, ya que `...resto` almacena el resto de los argumentos en un array.
   - `resto.length` devuelve 3 (porque hay tres elementos en resto).
   - `a + resto.length = 1 + 3 = 4`.

11. ¿Qué imprime este código?

```
const objeto = {
    x: 10,
    metodo: () => {
        const sumar = function () {
        return this.x + 5;
        };
        return sumar();
    }
};
console.log(objeto.metodo());
```

   a) `NaN`
   b) 15
   c) Error: `this.x` no está definido
   d) Ninguna de las anteriores es correcta.

   Explicación: La función `sumar` usa un `this` no definido en el contexto de objeto, lo que da como resultado `NaN`.

12. ¿Qué imprime este código?

```
const saludar = (nombre = "Invitado") =>  `Hola, ${nombre}!`;
console.log(saludar());
```

a) Hola, Invitado!
b) Hola, `undefined`!
c) Error: Falta un argumento en la función
d) Ninguna de las anteriores es correcta.

Explicación:

- **Definición de la función**:
  - `saludar` es una función flecha que recibe un parámetro `nombre`.
  - Si no se proporciona un valor, **se usará el valor por defecto "Invitado"**.
- **Llamada a la función `saludar()`:**
  - Se ejecuta sin pasar ningún argumento, por lo que `nombre` toma el valor "Invitado".
  - La función retorna la cadena:

```
"Hola, Invitado!"
```

- **Impresión con `console.log(saludar());`:**
  - Se muestra en la consola el valor recibido previamente al haber llamado a la función.
  - Vista de consola:

```
Hola, Invitado!
```

13. ¿Qué imprime este código?

```
const objeto = {
    a: 5,
    b: 10,
    calcular: () => objeto.a + objeto.b;
};
console.log(objeto.calcular());
```

a) `NaN`
b) 15
c) Error: objeto no está definido en la función flecha
d) Ninguna de las anteriores es correcta.

Explicación: Las funciones flecha no tienen su propio `this`, por lo que `objeto` dentro de `calcular` puede no estar bien referenciado.

14. Tiene el siguiente array de objetos que representa productos con sus cantidades y precios:

```
const productos = [
{nombre: "Manzana", cantidad: 2, precio: 1.5},
{nombre: "Naranja", cantidad: 3, precio: 2.0},
{nombre: "Platano", cantidad: 5, precio: 1.2}
];
```

¿Cuál de las siguientes opciones de código calcula y almacena el costo total de los productos ('cantidad * precio' para cada producto) en una variable llamada 'total', utilizando 'forEach'?

a)
```
let total = 0;
productos.forEach(producto =>total += producto.cantidad * producto.precio);
```
b) `const total = productos.forEach(producto =>producto.cantidad * producto.precio);`
c) No se puede obtener el resultado utilizando forEach.
d) Ninguna de las anteriores es correcta.

Explicación: `forEach` ejecuta una función en cada elemento del array, y la opción a usa una variable externa `total` para acumular el resultado correctamente.

15. ¿Cuál de las siguientes afirmaciones describe correctamente cómo se determina el valor de `this` en una función normal?

    a) Depende del contexto léxico donde fue definida la función.

    b) Depende de cómo se invoca la función (modo estricto, objeto invocante, etc.).

    c) Es siempre una referencia al objeto global o 'undefined' en modo de funcionamiento normal.

    d) Ninguna de las anteriores es correcta.

    Explicación: El valor de `this` en una función normal depende de cómo se llama: puede ser el objeto global, `undefined` (modo estricto), o un objeto específico.

16. En JavaScript, ¿cuál es el alcance de una variable declarada con `var` dentro de una función?

    a) Tiene alcance global.

    b) Tiene alcance de bloque dentro de la función.

    c) Tiene alcance de función, independientemente de los bloques donde esté definida.

    d) Ninguna de las anteriores es correcta.

    Explicación: Las variables declaradas con `var` tienen alcance de función, no de bloque (a diferencia de `let` y `const`).

17. ¿Cuál de las siguientes afirmaciones es verdadera sobre el `hoisting` en JavaScript?

    a) Tanto las declaraciones como las inicializaciones de variables y funciones se elevan al inicio del contexto de ejecución.

    b) Solo las declaraciones de funciones y variables se elevan; las inicializaciones no se elevan.

    c) Solo las funciones declaradas se elevan; las variables permanecen en su lugar.

    d) Ninguna de las anteriores es correcta.

    Explicación: El `hoisting` mueve las declaraciones al inicio del ámbito, pero las inicializaciones no se mueven.

18. ¿Qué imprime el siguiente código al hacer clic en '<hijo>'?

```
document.querySelector("#padre").addEventListener("click", () => {
    console.log("Padre");
}, true);

document.querySelector("#hijo").addEventListener("click", () => {
    console.log("Hijo");
});
```

    a) Imprime "Hijo" y luego "Padre".

    b) Imprime "Padre" y luego "Hijo".

    c) No imprime nada, ya que no se especificaron manejadores de eventos.

    d) Ninguna de las anteriores es correcta.

Explicación: El evento en `#padre` tiene `true`, activando `capturing`, lo que hace que se ejecute primero antes del evento en `#hijo`.

CORTAS/CÓDIGO

1. Defina cada una de las tres propiedades que caracterizan a los métodos HTTP: seguro, idempotente y cacheable. A continuación, indique las diferencias entre los métodos POST y PUT respecto a estas propiedades, así como su objetivo y cualquier otra restricción en cuanto a su sintaxis y/o funcionalidad.

   - **Seguro**: Un método HTTP es seguro si **no altera el estado del servidor**. Los métodos seguros son **GET, OPTIONS y HEAD**, ya que solo recuperan información sin modificarla.
   - **Idempotente**: Un método es idempotente si **realizar múltiples peticiones idénticas produce el mismo resultado**. Ejemplos de métodos idempotentes son **GET, PUT y DELETE**.
   - **Cacheable**: Un método es cacheable si **las respuestas pueden almacenarse en caché para reutilizarse en futuras solicitudes**. **GET** es el método más comúnmente cacheable.

   Diferencias entre `POST` y `PUT`

   - **Objetivo**:
     - `POST` se usa para **crear un nuevo recurso** en el servidor.
     - `PUT` se usa para **actualizar o reemplazar un recurso existente** en una ubicación específica.
   - **Propiedades**:
     - `POST` **NO es idempotente** (múltiples solicitudes pueden crear múltiples recursos).
     - `PUT` **es idempotente** (múltiples solicitudes generan el mismo resultado).
     - `POST` **NO es seguro**, mientras que `PUT` tampoco lo es.
     - `POST` **NO es cacheable**, mientras que `PUT` sí lo puede ser bajo ciertas condiciones.
   - **Restricciones sintácticas**:
     - `POST` envía datos en el **cuerpo** de la petición y puede contener diferentes formatos de datos.
     - `PUT` también envía datos en el **cuerpo**, pero requiere una URL específica donde se actualizará el recurso.

2. Sobrescribe el método `toString` del prototipo de `Array` para que devuelva una cadena con los elementos del array separados por | (carácter barra horizontal) en lugar de la coma por defecto. Demuestra el resultado con el array `[10, 20, 30]`

```
// Se debe entender que los metodos publicos de un objeto ('Array' en este caso) estan en el 'prototype'
// Dicho de otra forma: todos los objetos basados en el prototipo comparten metodos. Por tanto, la funcion 'toString' se debe redefinir en el prototipo de todos los objetos creados a partir de 'Array'

Array.prototype.toString = function () {
// La funcion 'join' combina todos los elementos de un array en una cadena utilizando el caracter entre parentesis. En esta ocasion, 'this' hace referencia al array que ha invocado el metodo
    return this.join('|');
};
// Se prueba con el array propuesto
const num = [10, 20, 30];
console.log(num.toString()); // "10|20|30"
```

3. Desarrolle una función `mapaDeHuecos` que reciba un array disperso y devuelva un objeto donde las claves sean los índices de inicio de los huecos consecutivos y los valores, la longitud de cada secuencia de huecos.

   Por ejemplo, para el siguiente array:

   $$[1, \_, \_, 4, \_, \_, \_, 5, 6, \_]$$

   Debe retornar:

   $$\{1 : 2, 4 : 3, 9 : 1\}$$

Solución en JavaScript

```javascript
function mapaDeHuecos(arr) {
    // Se necesita un objeto que contenga los pares <indice: num_huecos>
    const mapa = {};
    // Inicio de una secuencia de huecos
    let inicio = null;
    // Registro de la longitud de la secuencia
    let longitud = 0;

    // Se recorre el array en toda su longitud
    for (let i = 0; i < arr.length; i++) {
        // Se comprueba que el hueco esta vacio, evitando comparaciones con
        // 'undefined', 'null' u otros valores que podrian ser validos
        if (!(i in arr)) {
            // Si inicio es 'null', indica que es el comienzo de una nueva
                secuencia
            if (inicio === null) inicio = i;
            // La longitud se incrementa
            longitud++;
        } else {
            // Si se estaba contabilizando un hueco consecutivo...
            if (inicio !== null) {
                // Se registra en el objeto
                mapa[inicio] = longitud;
                // Se reinician los valores
                inicio = null;
                longitud = 0;
            }
        }
    }

    // Se revisa si al final quedo un hueco sin registrar
    if (inicio !== null) {
        mapa[inicio] = longitud;
    }

    return mapa;
}

// Se comprueba el funcionamiento
const arr = [1, , , 4, , , , 5, 6, ,];
console.log(mapaDeHuecos(arr)); // {1: 2, 4: 3, 9: 1}
```

4. Cree una función llamada crearObjeto que tome como argumentos un nombre y una edad, y devuelva un objeto con:

   a) Una propiedad `nombre` con el valor pasado como argumento.

   b) Una propiedad `edad` con el valor pasado como argumento.

   c) Un método `actualizarEdad` que reciba un número y actualice la propiedad edad.

   Demuestre su uso actualizando la edad de un objeto creado.

Solución en JS

Para realizar este ejercicio, hay que tener en cuenta que se está pidiendo que el método devuelva un objeto literal, por lo que no se debe caer en implementar una función constructora, sino una función convencional que devuelve un objeto ..

```
// La funcion recibe dos parametros que representan las propiedades del objeto.
function crearObjeto(nombre, edad) {
// Se devuelve un objeto literal, que contiene las propiedades de los argumentos (nombre, edad) y un metodo que recibe la nueva edad, n.
    return {
        nombre,
        edad,
        actualizarEdad(n) {
        // Para actualizar la propiedad 'edad', se debe acceder al contexto del objeto, por lo que es necesario el uso de 'this', que se refiere al objeto actual (el devuelto por crearObjeto)
            this.edad = n;
        },
    };
}
// Para probar la creacion del objeto, es importante considerar que NO es una funcion constructura (no se utiliza 'new') y, por tanto, se invocara como una funcion convencional
const persona = crearObjeto("Raul", 10);
// Para posteriormente invocar a actualizarEdad
persona.actualizarEdad(15);
console.log(persona);
```

Salida por consola:

```
Console
#(3) {nombre: "Raul", edad: 15, actualiza...}
    nombre: "Raul"
    edad: 15
    actualizar_Edad: f actualizarEdad()
        length: 1
        name: "actualizarEdad"
        [[Prototype]]: f ()
    [[Prototype]]: {}
```