



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE  
ESCUELA DE INGENIERÍA  
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

## IIC2513 – Tecnologías y Aplicaciones Web (II/2017)

Profesor: Raúl Montes

### Interrogación 3

15 de noviembre de 2017

#### Pregunta 1 (25%): Conceptos

Responde en forma precisa y concisa las siguientes preguntas (**5 preguntas**, 1.2 puntos cada una):

1. Respecto al siguiente trozo de código JavaScript: ¿ocurre algún error al llamar a la función `add` o `subtract` en una línea anterior a su declaración y definición? Explica por qué.

```
console.log('5 + 3 is ${add(5, 3)}');  
console.log('5 - 3 is ${subtract(5, 3)}');  
  
function add(number1, number2) {  
    return number1 + number2;  
}  
  
const subtract = function (number1, number2) {  
    return number1 - number2;  
}
```

2. ¿Cuáles son las diferencias entre `const` y `var`? Apóyate con ejemplos si aporta en la explicación.
3. Un amigo tuyo comenzó a aprender JavaScript y se topó con un concepto que no logra entender: “closure”. ¿Podrías ayudarlo con una explicación apoyándote en algún ejemplo?
4. Considera el siguiente trozo de código JavaScript. ¿Qué diferencia(s) tiene el objeto de la variable `a` con respecto al de la variable `b`?

```
function Foo1() {  
    this.bar = 1;  
}  
  
function Foo2() {  
    return { bar: 1 };  
}  
  
const a = new Foo1();  
const b = Foo2();
```

5. Si un *request Ajax* es, en términos del protocolo HTTP, un *request* como cualquier otro de los que se originan desde el *browser*, ¿en qué sentido y/o cómo logra que las aplicaciones Web que los usan se sientan más “responsivas”, rápidas e interactivas?

## Pregunta 2 (25%): Reverse engineering

Dado el siguiente trozo de código JavaScript, escribe todo el código JavaScript necesario para que éste funcione sin errores (que “no se caiga”) y de acuerdo a los comentarios descritos en el mismo código.

**Restricción:** no puedes usar la *keyword* `class`

**Nota:** La función `assert` recibe un argumento; si es *truthy* no pasará nada; en cambio, si es *falsy* se lanzará una excepción.

```
const p1 = new Person('John', 'Doe');
const p2 = new Person('Jane', 'Doe');

assert(p1.getFirstName() === 'John');
assert(p2.getLastName() === 'Doe');
assert(p1.getFullName() === 'John Doe');

// cumpliendo hasta aquí tienes 1.5 puntos

assert(p1.getFirstName !== p2.getFirstName);
assert(p1.getFullName === p2.getFullName);

// cumpliendo hasta aquí tienes 1 punto adicional

// luego de crear las personas no es posible cambiar sus nombres,
// así que esto por ejemplo:
p1.firstName = 'Other';
assert(p1.getFirstName() === 'John'); // sigue siendo válido

// cumpliendo hasta aquí tienes 1.5 puntos adicionales

addParticipant(p1);
assert(showParticipants() === 'John Doe is going');
addParticipant(p2);
assert(showParticipants() === 'John Doe is going; Jane Doe is going');

// cumpliendo hasta aquí tienes 1 punto adicional

// no hay código alguno que se pueda agregar
// (salvo sobrecribir la función showParticipants)
// que permita alterar lo que muestra showParticipants,
// salvo llamadas a addParticipant()

// cumpliendo hasta aquí tienes 1 punto adicional
```

### Pregunta 3 (25 %): JSON metadata middleware

Escribe un *middleware* para una aplicación *koa* decore respuestas JSON con metadata que contenga el tiempo que tomó procesar el *request* (en milisegundos) y un *id de request* que identifique al *request* procesado de manera única. Por ejemplo:

Respuesta JSON original:

```
{ user: { name: "John Doe", ... } }
```

Respuesta JSON decorada por el *middleware*:

```
{
  data: { user: { name: "John Doe", ... } },
  metadata: {
    timeSpent: 123,
    requestId: "9162ac25-a073-4889-b068-b3cdec9523c6"
  }
}
```

Como puedes ver, los datos originales quedan, en este nuevo JSON de respuesta, asociados a la *key* *data*, mientras que la metadata *timeSpent* y *requestId* quedan asociados a la *key* *metadata*.


**Hint 1:** al asignar un objeto JS a *ctx.body* la respuesta que *koa* entregará será un JSON. Además, *ctx.body* es un atributo que puede ser leído para conocer la respuesta que actualmente se estaría entregando al cliente.

**Hint 2:** para generar los *id de request* puedes usar un *package* llamado *uuid*, que exporta una función que, al ejecutarla, retorna un id aleatorio como el del ejemplo.

**Hint 3:** *new Date()* te entrega un objeto *Date* con la fecha y tiempo exacto en el momento de crearse; restar dos objetos *Date* te entregará un entero correspondiente a la cantidad de milisegundos de diferencia entre ambas representaciones de fecha/tiempo.

## Pregunta 4 (25 %): Votes DCCounter

¡Este domingo son las elecciones! Y, además, te tocó ser vocal de mesa. Dado que tendrás que estar ahí mientras se realice el escrutinio de los votos, se te ocurrió implementar una pequeña aplicación que te permita, en cada momento que estimes conveniente, enviar el conteo actualizado de los votos. Pensaste en una interfaz simple, como en la siguiente imagen:



The image shows a web form with the title "Conteo de votos" in a large, bold, black font. Below the title, there are two rows of input fields. The first row is labeled "Candidato A" and contains a text input field with the value "123". The second row is labeled "Candidato B" and contains a text input field with the value "321". Below these two rows, there is a single button labeled "Guardar". The entire form is enclosed in a thin black border.

Implementa un componente React llamado `VotesCount` que genere esta vista y que, cada vez que presiones el botón “Guardar”, se haga un *request Ajax* POST a la URL `/votes` con un JSON con *keys* de ids de los dos candidatos (puedes inventar los candidatos y sus ids) y *values* con el número de votos que tienen los respectivos *fields* en ese momento.

Puedes implementar todos los subcomponentes que estimes o prefieras, pero al menos tiene que haber uno principal llamado `VotesCount`. No es necesario que te preocupes del estilo CSS de la imagen (que es bien poco en verdad); tan sólo genera el HTML que te permitiría crear esa vista.

**Bonus 1:** 0.5 puntos adicionales si, mientras se envía el *request Ajax*, das *feedback* al usuario (un “loading...”).

**Bonus 2:** 0.5 puntos adicionales si utilizas correctamente el patrón *dumb/smart components*