

Jason Triche

Total duration: 171:57

Question 1

```
<?php

/**
 * Pub/Sub Challenge
 *
 * FIRST: COPY/PASTE ALL THE CODE IN THE TEXTFIELD BELOW AND EDIT IT THERE
 *
 * 1. The goal is to build a very simple PubSub/event class in PHP.
 *    We will create an EventEmitter object and then we'll subscribe to events and trigger them.
 *    Subscribing to an event simply adds a callback to be run when the event is triggered.
 *    Triggering an event (emit) should run all the attached callbacks.
 * 2. Don't overthink it. The solution should only take a few minutes and a few lines of code.
 *    Build only what you need to get the desired output.
 *
 * Constraints:
 * 1. Although we only use error/success events, please build the class to handle arbitrary events.
 * 2. Events data will always be an associative array.
 * 3. A callback should always be safe to call.
 */

class EventEmitter {

    public function __construct() {}

    public function emit() {}

    public function subscribe() {}

}

$emitter = new EventEmitter;

$error_callback = function($data) {
    echo "Error 1. {$data["message"]}\n";
};

$error_callback2 = function($data) {
    echo "Error 2. {$data["message"]}\n";
};

$success_callback = function($data) {
    echo "SUCCESS! {$data["message"]}\n";
};

$emitter->emit("error", ["message" => "Error one."]);

$emitter->subscribe("error", $error_callback);
$emitter->emit("error", ["message" => "Second error."]);

$emitter->subscribe("error", $error_callback2);
$emitter->emit("error", ["message" => "Yet another error."]);

$emitter->subscribe("success", $success_callback);
$emitter->emit("success", ["message" => "Great success!."]);

// Expected output:

// Error 1. Second error.
// Error 1. Yet another error.
```

```
// Error 2. Yet another error.
// SUCCESS! Great success!
```

```
class EventEmitter {

    public $data;
    public $callback;

    public function __construct() {
        $this->data = array();
        $this->callback = array();
    }

    public function emit($eventtag,$message) {
        $this->data = $message;
        if($this->callback[$eventtag]){
            call_user_func($this->callback[$eventtag],$message);
        }
    }

    public function subscribe($eventtag,$callback_function) {
        $this->callback[$eventtag] = $callback_function;
    }
}

$emitter = new EventEmitter;

$error_callback = function($data) {
    echo "Error 1. {$data["message"]}\n";
};

$error_callback2 = function($data) {
    echo "Error 2. {$data["message"]}\n";
};

$success_callback = function($data) {
    echo "SUCCESS! {$data["message"]}\n";
};

$emitter->emit("error", ["message" => "Error one."]);

$emitter->subscribe("error", $error_callback);
$emitter->emit("error", ["message" => "Second error."]);

$emitter->subscribe("error", $error_callback);
$emitter->emit("error", ["message" => "Yet another error."]);

$emitter->subscribe("error", $error_callback2);
$emitter->emit("error", ["message" => "Yet another error."]);

$emitter->subscribe("success", $success_callback);
$emitter->emit("success", ["message" => "Great success!"]);

// Expected output:

// Error 1. Second error.
// Error 1. Yet another error.
// Error 2. Yet another error.
// SUCCESS! Great success!
```



1x

2x

5x



0:00 / 59:14

Question 2

-- Use the following SQL Dump to write SQL queries for the 5 questions at the bottom:

```
CREATE TABLE Employees
(EmployeeID int auto_increment primary key,
 DepartmentID int,
 BossID int,
```

```

    Name varchar(100),
    Salary int);

CREATE TABLE
Departments
(
    DepartmentID int auto_increment primary key,
    Name varchar(30));

INSERT INTO
Departments
(DepartmentID ,Name)
VALUES
(1, "Sales"),
(2, "Support"),
(3, "Development");

INSERT INTO
Employees
(EmployeeID,
DepartmentID,
BossID,
Name,
Salary)
VALUES
(1, 1, 1, "Jimbo Jones", 200000),
(2, 1, 1, "John Doe", 250000),
(3, 3, 3, "Nerdy Dev", 130000),
(4, 3, 3, "Kevin Mitnick", 40000),
(5, 2, 5, "Janice Smith", 50000),
(6, 2, 5, "Support #2", 45000),
(7, 2, 5, "Support #3", 55000),
(8, 3, 5, "Support Dev", 75000);

-- COPY/PASTE ALL THE FOLLOWING QUESTIONS IN THE TEXTFIELD BELOW AND EDIT IT THERE

-- 1. List employees (names) who have a bigger salary than their boss

-- 2. List departments that have less than 3 people in it

-- 3. List all departments along with the total salary there

-- 4. List employees that don't have a boss in the same department

-- 5. List all departments along with the number of people there

```

```

-- 1. List employees (names) who have a bigger salary than their boss

Select a.`name` FROM Employees AS a
JOIN Employees AS b
ON (a.`BossID` = b.`EmployeeID`)
WHERE a.`Salary` > b.`Salary`;

-- 2. List departments that have less than 3 people in it
Select a.`DepartmentID`, a.`Name`,COUNT(b.`EmployeeID`) AS employee_count FROM Departments AS a
JOIN Employees AS b
ON (a.`DepartmentID` = b.`DepartmentID`)
GROUP BY b.`DepartmentID`
HAVING employee_count < 3;

-- 3. List all departments along with the total salary there
Select a.`DepartmentID`, a.`Name`,SUM(b.`Salary`) AS total_salary FROM Departments AS a
JOIN Employees AS b
ON (a.`DepartmentID` = b.`DepartmentID`)
GROUP BY b.`DepartmentID`;

-- 4. List employees that don't have a boss in the same department
Select a.* FROM Employees AS a
JOIN Employees AS b
ON (a.`BossID` = b.`EmployeeID` AND a.`DepartmentID` <> b.`DepartmentID`);

-- 5. List all departments along with the number of people there
Select a.`DepartmentID`, a.`Name`,COUNT(b.`EmployeeID`) AS employee_count FROM Departments AS a
JOIN Employees AS b

```

```
ON (a.`DepartmentID` = b.`DepartmentID`)  
GROUP BY b.`DepartmentID`;
```



1x

2x

5x



0:00 / 87:21

Question 3

// 1. What do you think the expected output is for the following code? Can you explain why it's not working as expected?
// 2. Rewrite the code to produce the expected output (important: we need to keep the 1000ms interval!)

```
for (var i=1; i<=5; ++i) {  
  setTimeout(() => console.log(i), 1000);  
}
```

```
for (var i=1; i<=5; ++i) {  
  setTimeout(() => console.log(i), 1000);  
}
```

*/*Upon first glance, it would appear that the loop would pause for 1 second then display each integer. However, in javascript
/*the correct approach uses the setTimeout in a recursive fashion, as it "promises" the result to the callback function even*

```
runSetTimeout(0);  
function runSetTimeout(i){  
  setTimeout(() =>  
    {if(i <= 5) {  
      console.log(i);  
      i++;  
      return runSetTimeout(i);  
    }}, 1000);  
}
```



1x

2x

5x



0:00 / 25:17