Palindrom

A palindrome is a word that reads the same backward or forward.

Write a function that checks if a given word is a palindrome. Character case should be ignored.

For example, *isPalindrome("Deleveled")* should return *true* as character case should be ignored, resulting in "deleveled", which is a palindrome since it reads the same backward and forward.

```php
<?php
class Palindrome
{
    public static function isPalindrome($word)
    {



    }
}

echo Palindrome::isPalindrome('Deleveled');
```

GroupByOwner

Implement a *groupByOwners* function that:

- Accepts an associative array containing the file owner name for each file name.
- Returns an associative array containing an array of file names for each owner name, in any order.

For example, for associative array *["Input.txt" => "Randy", "Code.py" => "Stan", "Output.txt" => "Randy"]* the *groupByOwners* function should return *["Randy" => ["Input.txt", "Output.txt"], "Stan" => ["Code.py"]]*.

```php
<?php
class FileOwners
{
    public static function groupByOwners($files)
    {




    }
}


$files = array
(
    "Input.txt" => "Randy",
    "Code.py" => "Stan",
    "Output.txt" => "Randy"
);
var_dump(FileOwners::groupByOwners($files));
```

Thesaurus

A *thesaurus* contains words and synonyms for each word. Below is an example of a data structure that defines a thesaurus:

```
array("buy" => array("purchase"), "big" => array("great", "large"))
```

Implement the function *getSynonyms*, which accepts a word as a string and returns all synonyms for that word in JSON format, as in the example below.

For example, the call *$thesaurus->getSynonyms("big")* should return:

```
'{"word":"big","synonyms":["great", "large"]}'
```

while a call with a word that doesn't have synonyms, like *$thesaurus->getSynonyms("agelast")* should return:

```
'{"word":"agelast","synonyms":[]}'
<?php
class Thesaurus
{
    private $thesaurus;

    function Thesaurus($thesaurus)
    {
        $this->thesaurus = $thesaurus;
    }

    public function getSynonyms($word)
    {













    }
}

$thesaurus = new Thesaurus(
    array
        (
            "buy" => array("purchase"),
            "big" => array("great", "large")
        ));

echo $thesaurus->getSynonyms("big");
echo "\n";
echo $thesaurus->getSynonyms("agelast");
```

OddNumbers

In an array having positive integers, except for one number which occurs odd number of times, all other numbers occur even number of times. Find the number.

Try to solve the problem with the lowest effort.

```
<?php

/**
 * get the number which occurs odd times in the source array
 *
 * @param $arrValues array
 * @return int
 */
function getOddNumber($srcValues) {







}


class OddNumberTest extends \PHPUnit_Framework_TestCase
{
    public function testOddNumber()
    {
        $result = getOddNumber([2,5,9,1,5,1,8,2,8]);
        $this->assertEquals(9,$result);
        $result = getOddNumber([2,3,4,3,1,4,5,1,4,2,5]);
        $this->assertEquals(4,$result);
    }
}
```

Invert Binary Tree

The task is to invert a binary tree, so the leaf’s order is inverted.
Example

```
       1         Inverts to         1
     /   \                        /   \
    2     3                      3     2
   / \   / \                    / \   / \
  4   5 6   7                  7   6 5   4
```

```php
<?php
// Tree data structure
class BinaryNode
{
    public $value = null; // node value
    public $left = null; // left child
    public $right = null; // right child
    public function __construct($value) {
        $this->value = $value;
    }
}

/**
*  invertTree function goes here
*/


class InvertBinaryTreeTest extends \PHPUnit_Framework_TestCase
{
    public function testInvert()
    {
        $root = new BinaryNode(1);
        $rootLeftChild = new BinaryNode(2);
        $rootRightChild = new BinaryNode(3);
        $rootLeftChildLeftChild = new BinaryNode(4);
        $rootLeftChildRightNode = new BinaryNode(5);
        $rootRightChildLeftChild = new BinaryNode(6);
        $rootRightChildRightNode = new BinaryNode(7);
        $rootLeftChild->left = $rootLeftChildLeftChild;
        $rootLeftChild->right = $rootLeftChildRightNode;
        $rootRightChild->left = $rootRightChildLeftChild;
        $rootRightChild->right = $rootRightChildRightNode;
        $root->left = $rootLeftChild;
        $root->right = $rootRightChild;
        $invertedTree = invertTree($root);
        $this->assertEquals($invertedTree->value, 1);
        $this->assertEquals($invertedTree->left->value, 3);
        $this->assertEquals($invertedTree->right->value, 2);
        $this->assertEquals($invertedTree->left->left->value, 7);
        $this->assertEquals($invertedTree->left->right->value, 6);
        $this->assertEquals($invertedTree->right->left->value, 5);
        $this->assertEquals($invertedTree->right->right->value, 4);
    }
}
```