# LEETCODE QUESTION IN PHP

## 1.TWO SUM

**Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.**

**You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.**

**You can return the answer in any order.**

**SOLUTION**

```
class Solution {
    function twoSum($nums, $target) {
        $numMap = [];
        foreach ($nums as $i => $num) {
            $complement = $target - $num;
            if (isset($numMap[$complement])) {
                return [$numMap[$complement], $i];
            }
            $numMap[$num] = $i;
        }
        return [];
    }
}
```

## <u>2.CONTAINS DUBLICATE</u>

Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

## <u>SOLUTION</u>

```
class Solution {
    /**
     * @param Integer[] $nums
     * @return Boolean
     */
    function containsDuplicate($nums) {
        $flipped =array_flip($nums);
        if(count($flipped)<count($nums))
            return true;
      return false;
    }
}
```

## 3<u>.SUM OF LEFT LEAVES</u>

Given the root of a binary tree, return *the sum of all left leaves.*

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

**SOLUTION**

```
class Solution {
    /**
     * @param TreeNode $root
     * @return Integer
     */
    function sumOfLeftLeaves($root, $start=true, $left=false) {
    if ($start) {
        if ($root===null) return 0;
        $this->sum = 0;
    }
    $leave = $root->left===null && $root->right===null;
    if ($leave) {
        if ($left) $this->sum += $root->val;
    } else {
        if ($root->left!==null)
            $this->sumOfLeftLeaves($root->left, false, true);
        if ($root->right!==null)
            $this->sumOfLeftLeaves($root->right, false, false);
    }
    if ($start)
        return $this->sum;
}
}
```

## 4. REFORMAT DATE

Given a date string in the form Day Month Year, where:

- Day is in the set {"1st", "2nd", "3rd", "4th", ..., "30th", "31st"}.
- Month is in the set {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"}.
- Year is in the range [1900, 2100].

Convert the date string to the format YYYY-MM-DD, where:

- YYYY denotes the 4 digit year.
- MM denotes the 2 digit month.
- DD denotes the 2 digit day.

SOLUTION

```
class Solution {
    /**
     * @param String $date
     * @return String
     */
    function reformatDate($date) {
        return date("Y-m-d",strtotime($date));
    }
}
```

5.HAPPY NUMBER

Write an algorithm to determine if a number n is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return true *if* n *is a happy number, and* false *if not*.

SOLUTION

```
class Solution {
   /**
    * @param Integer $n
    * @return Boolean
    */
   function isHappy($n) {
     $seen = [];
      while ($n != 1) {
         // If we've seen this number before, it's a cycle
         if (isset($seen[$n])) {
             return false;
         }

         // Mark the current number as seen
         $seen[$n] = true;

         // Calculate the sum of the squares of its digits
```

```
            $n = $this->sumOfSquares($n);
        }

        return true;
    }

    // calculate the sum of the squares of digits
    private function sumOfSquares($n) {
        $sum = 0;

        while ($n > 0) {
            $digit = $n % 10;
            $sum += $digit * $digit;
            $n = (int)($n / 10);
        }

        return $sum;
    }
}
```

6.REPLACE ELEMENT WITH GREATEST ELEMENT OF RIGHT SIDE

Given an array arr, replace every element in that array with the greatest element among the elements to its right, and replace the last element with -1.

After doing so, return the array.

SOLUTION

```
class Solution
{

    /**
     * @param Integer[] $arr
     * @return Integer[]
     */
    function replaceElements(array $arr): array
    {
        for ($i = count($arr) - 1, $max = -1; $i >= 0; $i--) {
            $current = $arr[$i];
            $arr[$i] = $max;
            $max = max($max, $current);
        }

        return $arr;
    }
}
```

7.DISTRIBUTE CANDIES

Alice has n candies, where the $i^{th}$ candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array candyType of length n,
return *the **maximum** number of different types of candies she can eat if she only eats* n / 2 *of them*

SOLUTION

```
class Solution {


    /**
     * @param Integer[] $candyType
     * @return Integer
     */
    function distributeCandies($candyType) {
        $getCandy = count($candyType) / 2;
        $uniqueCandyType = array_unique($candyType);
        return $getCandy <= count($uniqueCandyType) ? $getCandy :
count($uniqueCandyType);
    }
}
```

8.THIRD MAXIMUM NUMBER

Given an integer array nums, return *the **third distinct maximum** number in this array. If the third maximum does not exist, return the **maximum** number*.

SOLUTION

```
class Solution {
    function thirdMax($nums) {
        $distinct = [];
        $max = [PHP_INT_MIN, PHP_INT_MIN, PHP_INT_MIN];
        foreach ($nums as $n) {
            if (isset($distinct[$n])) {
                continue;
            }
            if ($n > $max[0]) {
                $max[2] = $max[1];
                $max[1] = $max[0];
                $max[0] = $n;
            } elseif ($n > $max[1]) {
                $max[2] = $max[1];
                $max[1] = $n;
            } elseif ($n > $max[2]) {
                $max[2] = $n;
            }
            $distinct[$n] = $n;
        }
        return ($max[2] > PHP_INT_MIN) ? $max[2] : $max[0];
    }
}
```

## 9.REVERSE VOWEL OF A STRING

Given a string s, reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.

SOLUTION

```
class Solution {
    function reverseVowels($s) {
        $vowels = ['a', 'e', 'i', 'o', 'u'];
        $vowelsStr = [];
        $str = str_split($s);
        foreach ($str as $k => $value) {
            if (in_array(strtolower($value), $vowels, true)) {
                $vowelsStr[$k] = $value;
            }
        }
        for ($i = 0, $iMax = count($vowelsStr); $i <= $iMax; $i++) {
            $first = array_key_first($vowelsStr);
            $last = array_key_last($vowelsStr);
            [$str[$first], $str[$last]] = [$vowelsStr[$last], $vowelsStr[$first]];
            unset($vowelsStr[$first]);
            unset($vowelsStr[$last]);
        }

        return implode($str);
    }
}
```

10.PRIME NUMBER OF SET BITS IN BINARY REPRESENTATION

Given two integers left and right, return *the **count** of numbers in the **inclusive** range* [left, right] *having a **prime number of set bits** in their binary representation*.

Recall that the **number of set bits** an integer has is the number of 1's present when written in binary.

- For example, 21 written in binary is 10101, which has 3 set bits.

SOLUTION

```
/**
 * @param Integer $left * @param Integer $right
 * @return Integer
 */
function countPrimeSetBits($left, $right) {
   $answer = 0; for($i = $left; $i <= $right; $i++) {
      $bin = decbin($i);


      $number = substr_count($bin, '1', 0);


               //harcoded primes :(
      if($number == 2 or $number == 3 or $number == 5 or $number ==
7 or $number == 11 or $number == 13 or $number == 17 or $number ==
19) {
         $answer++; } } return $answer;
}
```