PHP
!!!metode de caching

PHP is a programming language that can do all sorts of things: evaluate form data sent from a browser, build custom web content to serve the browser, talk to a database, and even send and receive **cookies (little packets of data that your browser uses to remember things**, like if you're logged in to Codecademy).

```
 echo 5 * 7;
<?php
 $myArray = array("do", "re", "mi");
 print $myArray{2}; //or $myArray[2];
 // prints "mi";
unset($myArray[2]); // removes the 3th element from the array

         foreach ($myArray as $lang) {
             echo "<li>$lang</li>";
         }
 unset($myArray) //delete the whole array
?>
```
___

do not write infinite loops!

```
        $loopCond = true;
        while (){
                //Echo your message that the loop is running below

                $loopCond = false;
        }
        echo "<p>And now it's done.</p>";
```

```
while(cond):
  // looped statements go here
endwhile;
```

difference is that the code inside a **while** loop can be **bypassed** entirely whereas the code inside a **do/while loop** will execute **at least once**.

```
$i = 0;
do {
    echo $i;
} while ($i > 0);
```
___

strlen();
substr($stringul, $start, $charsno);
strtoupper();
strtolower(); makes lowercase of that string
strpos($haystack, $needle); gaseste pozitia acului in capita de fan

---

// Round pi down from 3.1416...
//round your number to an integer, or round off complex floating point numbers to a specific number of decimal places
$round = round(M_PI);
print $round; // prints 3

// This time, round pi to 4 places
$round_decimal = round(M_PI, 4);
print $round_decimal; // prints 3.1416

max poz of $string char **= stlen($string) -1**;

$fav = **array()**;
array_push() takes two arguments: an array, and an element to add to the end of that array.
array_push($fav,”primal elem pus la sfarsitul arrayului”);
print **count**($fav);

$array = array(5, 3, 7, 1);
sort($array);
print join(“:”, $array);
// prints “1:3: 5: 7"
PHP also has the opposite function: rsort().
rsort($array);
print join(“>”, $array);
// prints “7,5,3,1”

---

bundle = a **collection** of things, or a quantity of material, **tied or wrapped up together**.

methods are functions bundled into objects and have the syntax:
public function funcname($optionalParameter) {
  // Do something
}
 a special method is:
public function **__construct(**$prop1, $prop2) {
  $this->prop1 = $prop1; // when we access property in a class we use $ only with $this, and not with the property name

```
  $this->prop2 = $prop2;
}
```

calling a method: **$obj1 -> meth1();**

```
<html>
  <head>
    <title>Class and Object Methods</title>
  </head>
  <body>
    <p>
      <?php
        class ClassName {
          public $variable = true;

          function __construct($property_name) {
              $this->property_name = $property_name;
          }

          public function method_name() {
            return "I'm dancing!";
          }
        }

        $obj = new ClassName("Shane");
        if (is_a($obj, "ClassName")) {
          echo "I'm a person, ";
        }
        if (property_exists($obj, "property_name")) {
          echo "I have a name, ";
        }
        if (method_exists($obj, "method_name")) {
          echo "and I know how to dance!";
        }
      ?>
    </p>
  </body>
</html>
```

outputs: I'm a person, I have a name, and I know how to dance!

```
class Person {
  public static $isAlive = "Yep!"
  public static function greet() {
    echo "Hello there!";
  }
```

}

echo **Person::$isAlive**; // **access static methods or properties of a class without instantiating the object**
// prints "Yep!"
Person::greet();

____
Associative Arrays and Arrays of Maps
array('key'=> 'value'…)

```
 $associativeMultidimensionalArray = array(
                 'level1' => array('year' => 2012,
                                          'colour' => 'blue',
                                          'doors' => 5,
                                          'make'=> 'BMW'),
               'level2' => array('year' => 2012,
                                          'colour' => 'blue',
                                          'doors' => 5,
                                          'make'=> 'BMW')
);
```

- the value is easy to use because of the sugestive key

e.g.: echo "The colour of my car is".$associativeArray['colour'];

```
$len = count($myarr);
foreach ($associativeArray as $level){
          foreach ($level as $keya=> $valoarea){
            echo $keya."are valoarea ".$valoarea;
           }
}
```

____
Fatal error: Call to private MyObject::__construct() from invalid context

creating a new object will always call the constructor.
Your MyObject class has a protected or private constructor, which means that the class cannot be instantiated. __construct() functions are always called when an object is instantiated, so trying to do something like $x = new MyObject() will cause a fatal error with a private construction function. (If you do not specifically declare a __construct() function, the parent constructor will be called).
Private constructors are often used in Singleton classes to prevent direct instantiation of an object. If it's not a class that you built, it might have a getInstance() function available (or something similar) to return an instance of

itself.
 I think I've seen people declaring private constructors before. Is there any reason to do this?
The only reason to do it is if you don't wish the class to be instantiated for some reason. As I mentioned in the answer, the Singleton pattern is a popular reason you might do this (en.wikipedia.org/wiki/Singleton_pattern)
Private constructors are often used by PHP implementations of the Singleton pattern, and are sometimes used with Factories. For example, a factory static method of the class can search for locally cached instances of objects and return a reference instead of a new object. Making the constructor private will prevent any accidental bypasses of the factory method.
So, What is the solution?
Singleton is not the only reason to have a private constructor. Sometimes it is nice to be able to disable the constructor by making it private and create one or more *named constructors* which you do by creating a public static method which intializes an instance and returns it

---