Table of Contents

1 These Sushi Cards will help you learn to build webpages using PHP, a very popular programming language that's used to build websites like **Facebook** and **Wikipedia**. You can combine it with **HTML** (and check out the HTML Sushi Cards if you don't know how to code HTML yet!).

2 First, you're going to need somewhere to code! PHP needs a few different pieces in place to make it work and, if you're going to release a website later, you'll eventually need to learn how to set that up. For now, though, let's just get going quickly by using Cloud 9, an online editor that will do all the setup for you! Check it out at dojo.soy/php-edit.

3 You can use your GitHub or Bitbucket account to sign in if you have one. Otherwise, sign up with your email address.

4 Choose "Create a new workspace" and fill it in as follows:

- **Workspace name:** beginner-php
- **Description:** My first PHP website
- Make sure that **Hosted workspace** is selected
- Choose a **Private** workspace
- Skip over the **Clone from Git or Mercurial URL**—you don't need to clone anything
- Choose the **PHP, Apache** template

5 Now just click on **Create workspace** and you're all set!

6 Once your workspace is up and running, you'll notice you've been given a few files, which you can see in the sidebar.

7 You're going to need to create a new file to put all your code in. You can do this by choosing **File > New File** in the menu.

8 Once the file is opened, you can save it by choosing **File > Save As** and giving it a name. Save this one now as **index.php**

9 You're ready to make your first PHP webpage!

1 Time to start coding your first real PHP webpage! You're going to start with one of the classics, saying "Hello!" to your user and, by the end of these cards, you'll be making a number guessing game for the user to play!

2 So, to begin with, every PHP webpage will have at least a little bit of HTML in it. Start by making a really basic one by typing this code into your **index.php** and saving it.

```
<!DOCTYPE html>
<html>
<head>
<title>My PHP webpage</title>
</head>

<body>
This is just HTML text, it is not clever like your PHP text will be!
</body>

</html>
```

The things in the angle brackets (`< >`) are called **HTML tags** and you'll be coming across a few of them in these cards. There are lots more, though, and you can learn about them in the HTML Sushi Cards and in loads of other places online!

3 Run that page and you'll see a basic HTML page. To begin with, your PHP page is going to look pretty similar, but very quickly you'll be using the power of PHP to do things that HTML never could alone! To get started, replace that HTML text with some special PHP code.

```
<body>
    <?php
        echo "Hello everyone! This is my first PHP program!";
    ?>
</body>
```

Notice that PHP code always appears inside `<?php` and `?>`. This is so the computer can tell which parts are PHP and which parts are HTML. Using these, you can put PHP code anywhere inside the HTML.

Also, notice that the PHP code always ends each line with a semicolon (`;`). This is so PHP knows to end this command and start another. If you forget to do this, PHP can get very confused.

4 Now save and run your PHP file. Congratulations! If it all worked, you've just written your first PHP webpage!

1 Now it's time to start working on your game! The first thing you're going to need is to teach the player the rules.

You might want to change things like the smallest and biggest numbers, or the number of guesses you're going to give the player. If you wrote out the rules with plain HTML, you'd have to go back and re-write them every time you changed any of those things. You don't need to do that, though. You can use PHP and **variables** to include the numbers as part of your text!

2 First, you're going to need to to update your text, so change the PHP code on your page to this:

```
echo "I've picked a number between 1 and 9<br />";
echo "You will have 5 chances to try to guess my number!";
```

The `<br />` is a piece of HTML that tells the browser to start a new line after it. You can put any HTML inside your PHP and it will be treated exactly as if it had been written as HTML.

Notice that you're using double quotes (`"`) around your text instead of single quotes (`'`), for two reasons:

- PHP will get confused if you use an apostrophe inside of single quotes. Try it and see! What happens, and why?
- PHP will do something special inside of double quotes that it won't inside of single quotes, which you'll see below.

3 Now it's time for you to start adding **variables**! These are labels that you can use to store values, like a **string** of text or a number. PHP remembers those values and lets you use the values later by using their label. This lets you set a value once and use it loads of times in your program.

In PHP, all variable names start with a dollar sign (`$`) and are usually written in **camelCase**, where the first word starts with a small (lowercase) letter, there are no spaces, and any later words start with capital (uppercase) letter.

Put this line in, inside the `<?php` but before the `echo` lines

```
$minValue = 1;

$maxValue = 9;

$guesses = 5;
```

All of these **variables** are numbers, but you'll be working with text variables later.

4 Now, update your two `echo` lines so they look like this:

```
echo "I've picked a number between {$minValue} and {$maxValue}<br />";

echo "You will have {$guesses} chances to try to guess my number!";
```

Notice the curly braces (`{` and `}`) around the variable names to tell PHP not to treat them like regular text!

5 Run your program and see what happens. Then try changing the values of some of the variables and run it again. Just make sure to set everything back to the way you've got it here before moving on!

1 Ok, so you can get information from a variable and show it to the player with `echo`, but how do you get information from the player? After all, this is a guessing game, so you need some way to collect their guesses!

2 You're going to use a combination of HTML and PHP to do this. The HTML you'll be using is going to be new, even if you've already done the HTML Sushi Cards, since they don't use many **forms** and your PHP programs will probably use a lot of them. You can see them on almost every website you use the internet and the code for them is pretty easy!

```
<form method="get">
 Your guess: <input type="text" name="guess"/>
</form>
```

A HTML form is a simple idea: controls—like text boxes, drop-down menus, check boxes or buttons—are used to collect information from users and send that information to your PHP program. Sometimes, responses are sent to the users. You'll be sending responses in your games.

3 Now, time to add a form to your page! So you can use PHP variables and other code in creating your form, you're going to use `echo` statements to create it, instead of just typing the HTML into the file. So, add the following below your two existing `echo` statements:

```
echo "<form method="get">";
echo "Your guess: <input type="text" name="guess"/>";
echo "</form>";
```

Run this code and see what happens!

4 Well, that didn't quite work right, did it? Any idea why? It's because of the double quotes in `<form method="get">`. PHP isn't smart enough to recognise that they are part of the HTML tag; it reads them as the end of the **string** of text that `echo` is trying to insert into the page. After that, it's looking for a semicolon (`;`) but instead it finds `g` and gets very confused! This kind of problem can come up quite often either because you've forgotten which kind of quotes to use or you're copying from another of your programs and used quotes differently there. Luckily, it's very easy to fix. Just put a backslash (`\`) in front of your all your double quotes (don't forget the ones in the **input** tag!) like this, to **escape** the normal rule of ending the **string**!

```
echo "<form method=\"get\">";
```

5 Run the code again! Now you've got somewhere for your user to put their text! Type something in and press the enter key. Watch the page URL in the browser and notice what changes!

1 The text that appeared on the end of the URL is called a **query parameter** and you can have loads of them on the same URL. The form automatically added one (because you set `method="get"`), but you could just as easily type them in, separated by semicolons. PHP reads the end of the URL for the page it's loaded on and turns all the query parameters into something called a **key, value array**. You don't need to know exactly what that is right now, we'll go into it in a later Sushi Card series. What you *do* need to know is how to get values out of it.

2 All you need to do to get a value out of the array is pass in the **key** of a **value** that's in there. Since your form adds the `guess` field, that's the one you'll be looking for. You'll want to **assign** that value to a `playerGuess` variable, so add this line just before all your `echo` lines:

```
$playerGuess = $_GET['guess'];
```

3 Now, remind the player what their last guess was by adding another `echo` just before the input form:

```
echo "Your last guess was {$playerGuess}.<br />";
```

Run the code. Now you're taking input from your player and giving it back to them. Very cool!

4 Try removing all the query parameters from the URL and reloading the page. Notice that you now get "Your last guess was ."

That's not ideal. What you want to happen is:

- Check **if** there's a value (maybe they haven't made a guess yet)
  - if so, show the message about the last guess

5 PHP can figure all this out and do it for you! You just need to use an `if` statement.

An `if` statement uses a test (in brackets), that has an answer that's either **true** or **false** and, if it's **true**, a piece of code to run. You can do this to check if your variable contains anything like this:

```
if(!empty($playerGuess)){
    echo  "Your last guess was {$playerGuess}.";
}
```

Here `empty()` is a special piece of PHP that checks whether the variable inside its brackets has a value and answers either **true** if it doesn't or **false** if it does. The `!` before it reverses this answer, turning a **true** into a **false** and a **false** into a **true**. So what this code says is: "If there is not no value in $playerGuess, then print out *Your last guess was [the value of player guess].*"

Replace `echo "Your last guess was {$playerGuess}.";` with the code above and test it with and without answers to see it working.

1 Now that you know how to use `if` statements, you can start writing the code to get your game to run! You'll deal with using a random number in a later card, but for now just add another variable up at the top with all the others to set your "secret" number, like this:

```
$secretNumber = 5;
```

2 What you want to do now is compare the player's guess with the secret number, and tell them if they guessed correctly. To compare one value to another and get a **true** or **false** result, you use two equals signs (`==`). If the values on either side are the same, then the result is **true**; otherwise, it's **false**.

Here's how you'd check the player's answer in your PHP:

```
if($secretNumber==$playerGuess){
    echo("<br />That's right! I was thinking of {$secretNumber}!");
}
```

Add this code in (after those variables' values are set!) and then run the program. Try guessing correctly (i.e. 5) and incorrectly.

3

Have you noticed a few issues with what's happening here? For one thing, the player is still asked to pick a number even when they've won the game! You can fix that, though, by using `else` statements, after your `if` statements. You can't use `else` on its own, it only runs the code inside it if the test on the `if` statement just before it was **false**.

So, to only show the form for the next guess if the player has not yet guessed the secret number, you need to add an `else` on to your `if` from above and move all the form code into it, like this:

```
if($secretNumber==$playerGuess){
echo("<br />That's right! I was thinking of {$secretNumber}!");
}
else{
    echo  "<form method='get'>";
    echo  "Your guess: <input type='text' name='guess'/>";
    echo  "</form>";
}
```

1 It doesn't make sense to give the player the instructions on how to play every single turn. However, it would be useful to tell them how many guesses they have left. To do this, you'll need to create another variable, `guessesLeft` and add it to your file:

```
$guessesLeft = $guesses;
```

Note that because you are setting this variable to the value of another, it must be declared *after* that variable.

2 Next, take the rules `echo` code and stick it into an `if` that checks if the number of guesses the player has left is equal to the number they started with. If it is, then you know it's the first turn and you can show them the rules. If it's not, then tell them how many guesses they have left.

```
if($guessesLeft==$guesses){
    echo "I've picked a number between {$minValue} and {$maxValue}<br />";
    echo "You will have {$guesses} chances to try to guess my number!<br />";
}
else{
    echo "You have {$guessesLeft} guesses left.<br />";
}
```

3 If you run this code and play through it a few times, you'll notice that the number of guesses never goes down! There are two reasons of that, and you'll solve them one at a time. The first is that you never decrease the value of `guessesLeft` ! There are two things you need to do to make this happen and you'll go through both of them on this card. First, you need to check if the player made a guess and, if so, set `guessesLeft` to one less than the current value.

You already have the test for making a guess, since you check for a guess before deciding whether to show "Your last guess was...". You can just update that code to include a second line that decreases guessesLeft, like this:

```
if(!empty($playerGuess)){
    echo  "Your last guess was {$playerGuess}.";
    $guessesLeft = $guessesLeft - 1;
}
```

4 Now it's important to keep an eye on the order of your code. If you run this code as written, you'll notice the count of guesses still doesn't drop! This is because the `echo` code that prints the value of guessesLeft runs before the code that changes that value! All you need to do is move (cut and paste) the code so it comes just after all your variable declarations at the start and things should work fine. Do that now and try running it again.

1 You'll notice that now the count of guesses does go down to 4, but it never goes any lower! Any idea why? It's because of the way PHP works: Every time the page loads, the program runs again from the start, with no memory of the last time it ran! That means it always resets the value of `guessesLeft` to 5 and then, if a guess was made, subtracts 1. So, what you need to do here is store that value somewhere and pass it into the program the next time it runs. Well, you already have something like that: the form on the web page! You can write the current value of `guessesLeft` into the form and read it back out from `$_GET` later. Start with the writing, by updating your form creation code to look like this:

```
echo "<form method='get'>";
echo "Your guess: <input type='text' name='guess'/>";
echo "<input type='hidden' name='guessesLeft' value='{$guessesLeft}'/>";
echo "</form>";
```

The type of the field is "hidden", so it won't be visible, but will be sent in on the URL. The value is set to the current value of 'guessesLeft'.

2 Now you need to pick up the value you've saved and write it into guessesLeft. This gets a little complicated, since there won't always be a value in the URL (the first time the game is run, for example). So you need to replace the current line `$guessesLeft = $guesses` with this:

```
$guessesLeft = $_GET['guessesLeft'];
if(!isset($guessesLeft)){
    $guessesLeft = $guesses;
}
```

This code tries to get the value from the URL and, if it doesn't find it, leaves `guessesLeft` empty, which causes the code in the `if` statement to run, filling it with the value of guesses. The reason we didn't use `empty` here is because empty will give a **false** if the value is 0, which is going to happen here, when the player runs out of guesses. Try running the program again and you'll see that it all works now!

3 You may have noticed one last problem: The game doesn't end when you run out of guesses! We'll look at how to fix that, and how to use a secret number that actually changes, on the next card!

1 First, let's look at making the player lose the game. This needs to happen when they've used their last guess up, so when `$guessesLeft == 0`. When this happens, keep showing them their last guess, and that they have no guesses left, but not the form or the rules. To make that happen, you'll need to use a new bit of code: `elseif`.

2 As you might have guessed `elseif` is a combination of the `else` and `if` statements. Like `else`, it only happens if the condition in an `if` statement is **false**, but like `if` it has its own condition.

You can use as many `elseif` statements as you want, but only the first one that has a **true** condition will run. If none of them are true, the `else` statement will run, just like with a regular `if`.

To get your game to tell the player they've lost, you just want to add an `elseif` to the code that checks their answer and shows them the for to, if they have now guesses left, show them a different message instead, like this:

```
if($secretNumber==$playerGuess){
    echo("<br />That's right! I was thinking of {$secretNumber}!");
}
elseif($guessesLeft==0){
    echo("<br />Game over! You lose!");
}
else{
    echo  "<form method='get'>";
    echo  "Your guess: <input type='text' name='guess'/>";
    echo  "<input type='hidden' name='guessesLeft' value='{$guessesLeft}'/>";
    echo  "</form>";
}
```

3 Ok, now you have a game where the player can guess a number, get a certain number of tries and be told if they win or lose. Very cool! However, right now, that number is *always* 5... which is less cool. PHP is pretty good at coming up with random numbers, though. To use a random number instead you just need to change the code for `secretNumber` like this:

```
$secretNumber = $_GET['secretNumber'];
if(!isset($secretNumber)){
    $secretNumber = rand($minValue, $maxValue);
}
```

The `rand` is a **function** that takes two numbers and gives you back a random number between them. You're using `minValue` and `maxValue` here so you only have to change those numbers in one place and they'll change everywhere in your code!

4 Now you need to make sure that it's the *same* random number throughout the game. It wouldn't be fair to keep changing it on your player! You already know how to do this one though: a hidden form field. Just add this to your `echo` code for the guessing form (you won't need to keep storing it if the player wins or loses!).

```
echo  "<input type='hidden' name='secretNumber' value='{$secretNumber}'/>";
```

5 Now, try to play your game!

6 How else could you use this code? You've got all the pieces here to make a quiz or an interactive story, where you keep score, ask different questions on each page and even send the player in different directions based on their answers!