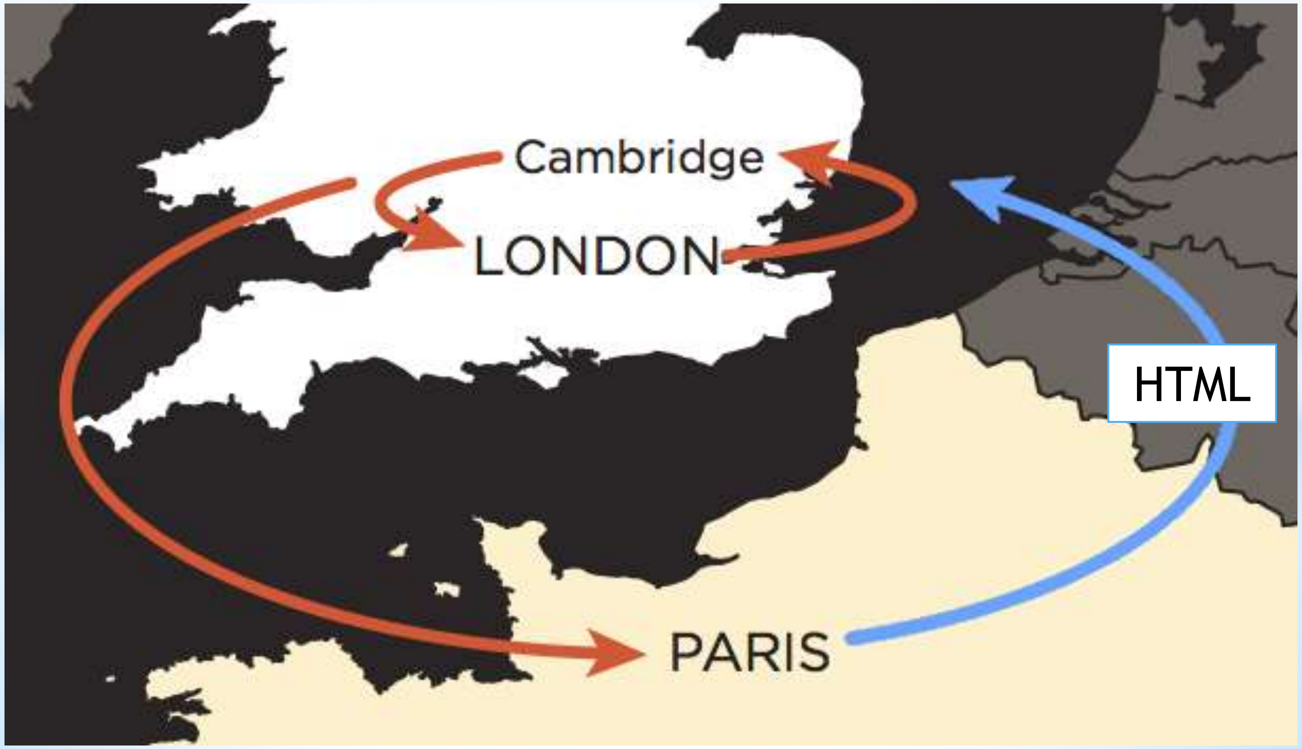
Programming 1 Team Solo

The basics of web programming:
Lecture #3: Introduction to PHP

CLICKER CHANNEL: 82

Recap: How the web works

- The browser queries a DNS server to find the IP address of the destination web server
- The browser connects to the destination web server
- The destination web server sends back (among other things) HTML describing the structure and content of the page



The story so far

- HTML describes the structure and content of a web page
- HTML is static – it never changes
  - If you reload a page a million times, you'll see the same page a million times
- However, the modern web doesn't (often) work like that

The need for dynamic pages

- Consider if you wanted a page to simulate the throw of a dice
  - Say you had six images of a dice, and you wanted to show a random different one each time the page loaded

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <img src="http://paulneve.com/dice1.png"/>
  </body>
</html>
```

The need for dynamic pages

- No matter how many times you request the page, the server would always send back the same HTML, i.e.

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <img src="http://paulneve.com/dice1.png"/>
  </body>
</html>
```

- So every time you requested the page, you'd get the dice face image for 1
- But what about if the server could send back (slightly) different HTML markup each time?

Introducing server side scripting

- Server side scripting allows us to write programs that run on the web server – this is called *server side*
- There are many different server side scripting languages and platforms
  - PHP
  - Perl
  - Java
  - ASP.NET
  - Python
  - Ruby
- Regardless of language, the principle is the same
  - A program runs on the web server
  - The program generates an HTML page
    - The HTML that the program generates can be different each time
    - The HTML that the program generates can vary, depending on (for example)
      - user input
      - external data sources (e.g. databases, web services)
  - The server sends back the generated HTML to the browser

An example...

- Let's pretend that we could write some Banana code that would run on a web server
- Our hypothetical code is going to generate the HTML to produce a random dice throw:

```
set dicenum =  random(5) +1
display "<html>"
display "<head>"
display "<title>Dice page!</title>"
display "</head>"
display "<body>"
display "<p>You have thrown a</p>"
display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
display "</body>"
display "</html>"
```

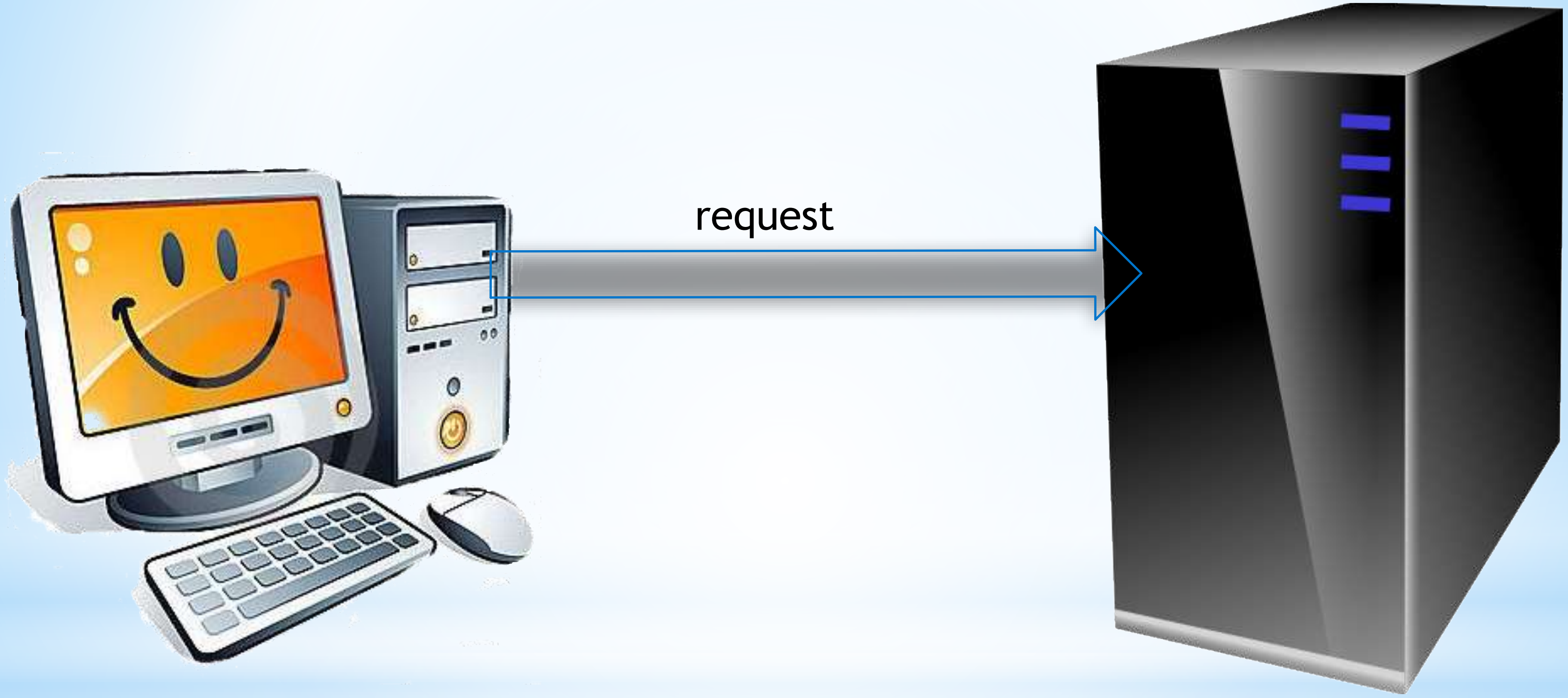
Note sneaky new Banana command, random – this gives a random number between zero and the number specified between the brackets

An example…

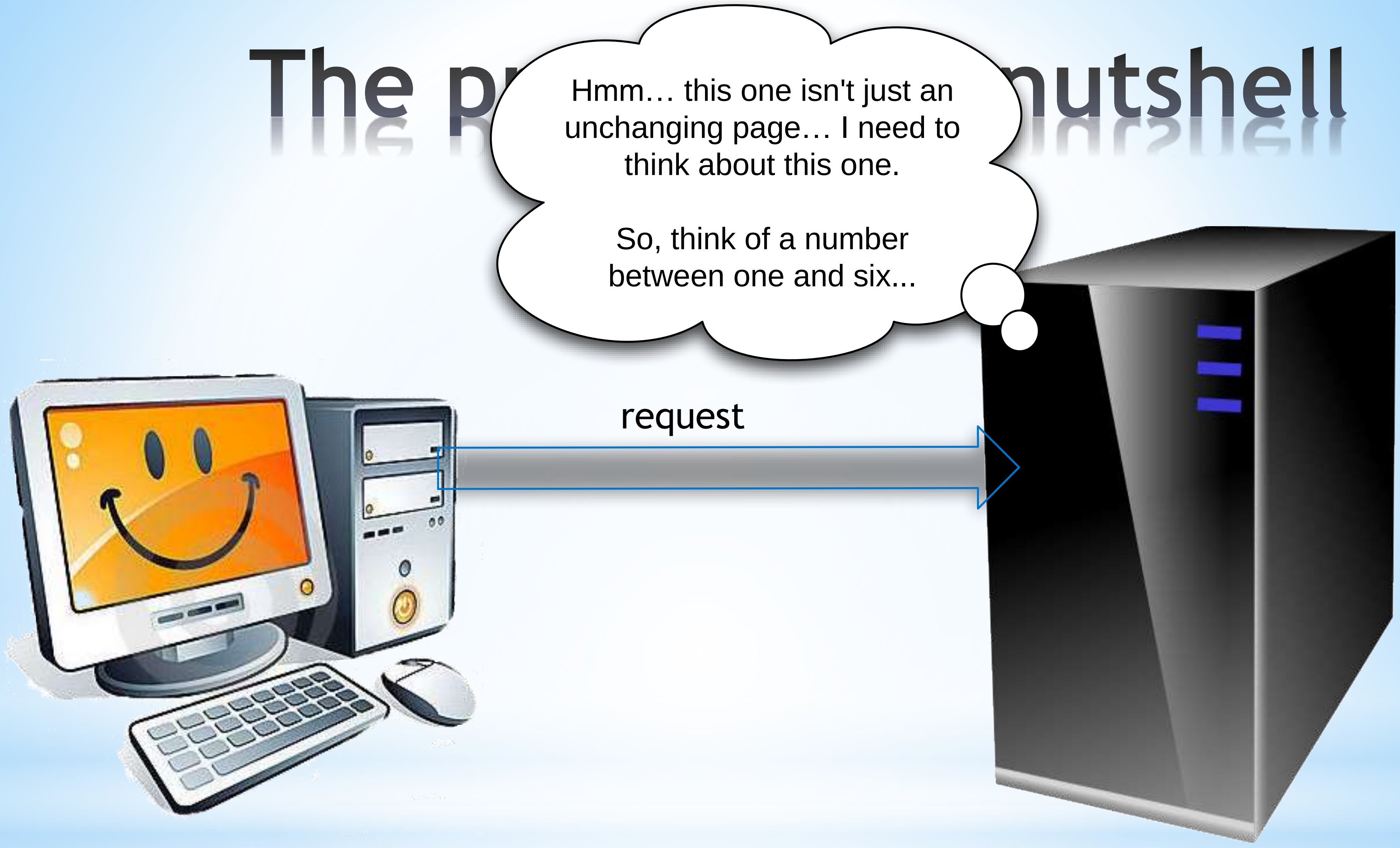
```
set dicenum =  random(5) +1
display "<html>"
display "<head>"
display "<title>Dice page!</title>"
display "</head>"
display "<body>"
display "<p>You have thrown a</p>"
display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
display "</body>"
display "</html>"
```

- Now, think about what would happen if
  - This code ran on our server whenever a web browser requested a particular URL
  - Instead of the output appearing in the NoobLab console window, it was sent back to the browser
  - What would the user viewing the browser page see? What would their experience be?
  - What would happen when they visited the page a second, third, fourth time?

The process in a nutshell

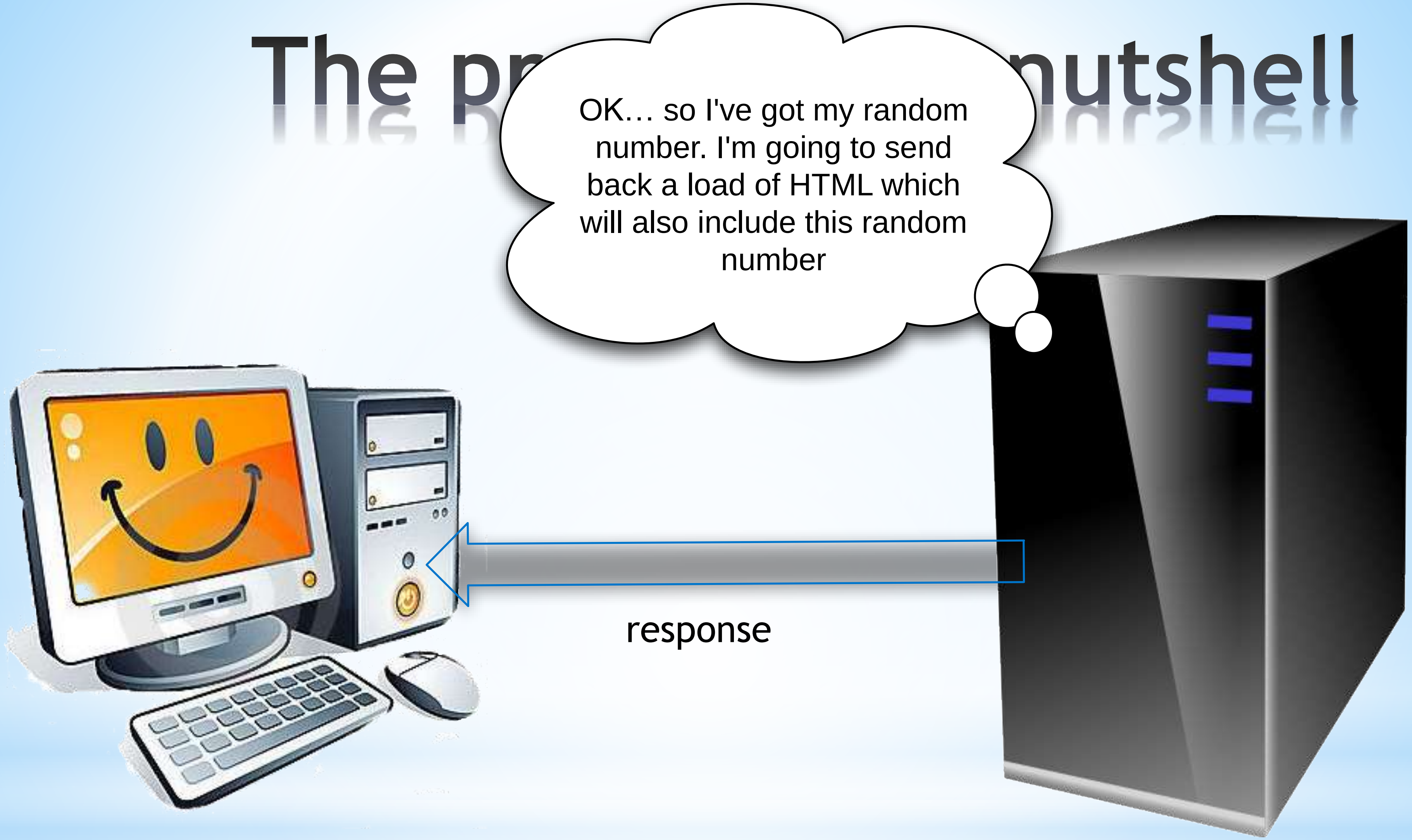


The p… nutshell

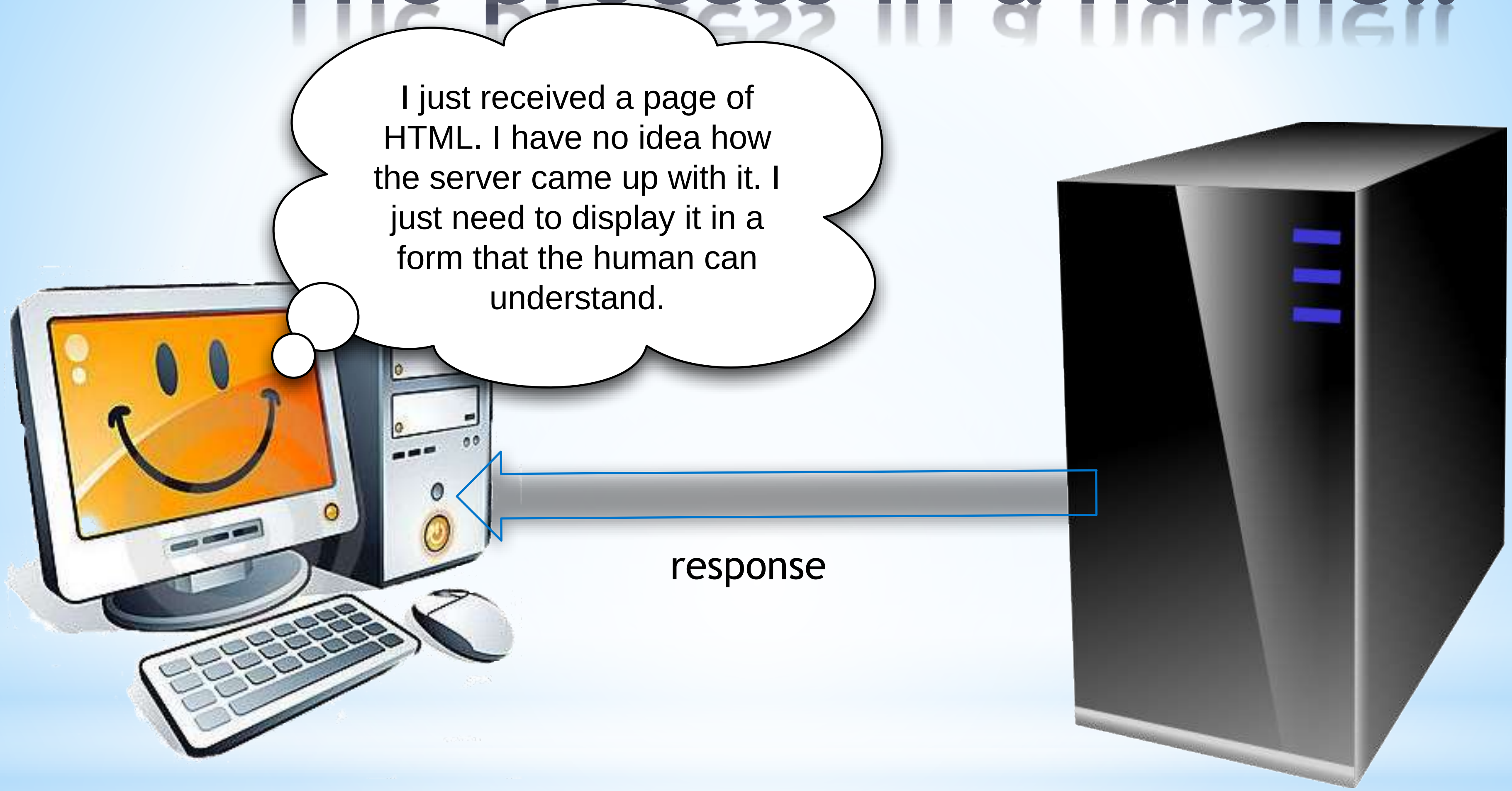

Hmm… this one isn't just an unchanging page… I need to think about this one.

So, think of a number between one and six...
request

The pr nutshell


OK… so I've got my random number. I'm going to send back a load of HTML which will also include this random number
response

The process in a nutshell


I just received a page of HTML. I have no idea how the server came up with it. I just need to display it in a form that the human can understand.
response

A bit of a mouthful

- In our example, most of the HTML we send back from the server is unchanging – only one character changes

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <img src="http://paulneve.com/dice1.png"/>
  </body>
</html>
```

- It's a bit of a drag having to have repeated display instructions with string constants, isn't it?
- Wouldn't it be easier if we could just have a mixture of static HTML and then just bring in Banana code at the point where we needed the bits that change?

Mixing code and HTML

- What about if we could do this?

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <bananaCode>
      set diceNum = random(5)+1
      display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
    </bananaCode>
  </body>
</html>
```

Mixing code and HTML

- What about if we could do this?

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <bananaCode>
      set diceNum = random(5)+1
      display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
    </bananaCode>
  </body>
</html>
```

- Anything not within the **<bananaCode>** tags just goes to the sever as static HTML
- Anything within the **<bananaCode>** tags gets run as code
  - Any display commands get inserted into the HTML at the point in which they occur
- So, the author of the page can just create standard HTML for the majority of it
- Only the bits that need to be dynamic – i.e. that change – need to have actual code

Mixing code and HTML

- The line of code
  `display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"`
  is still a bit convoluted
- What about if we could do this?

```
<bananaCode>
   set diceNum = random(5)+1
</bananaCode>
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <img src='http://paulneve.com/dice<bananaVar>dicenum</bananaVar>.png'/>
  </body>
</html>
```

Mixing code and HTML

- The line of code
  `display "<img src='http://paulneve.com/dice" + dicenum + "'.png'/>"`
  is still a bit convoluted
- What about if we could do this?

```
<bananaCode>
   set dicenum = random(5)+1
</bananaCode>
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
   <p>You have thrown a</p>
   <img src='http://paulneve.com/dice <bananaVar>dicenum</bananaVar>
.png'/>
  </body>
</html>
```

Do our server-side "thinking" as the first thing, up front

Use another server-side tag to specify that we just want to insert the value of a Banana variable into the HTML here

Bad news, good news

- The bad news is that Banana is NOT a server-side language or platform for the web – you can't write your web apps in Banana
  - (yet! Maybe next year! ☺)
- The good news is that PHP (and many of the real platforms and/or languages) works in precisely this way

Introducing PHP

- PHP stands for PHP: Hypertext Processor
  - (yes, this is a recursive acronym!)
- PHP is by far the most popular web application platform currently in use
- It is used by big names such as Facebook, Wikipedia and Twitter to name a few
- PHP is popular because novices can quickly learn it and use it to create dynamic web sites and applications…
- …but PHP is also criticised because this ease of use often leads to sites with really ugly (and often buggy code)

My first PHP "program"

```
<?php
    echo '<h1>Hello world</h1>';
?>
```

My first PHP "program"

```
<?php
     echo '<h1>Hello world</h1>';
?>
```

The text **`<?php`** means that everything that follows is interpreted at the server side as PHP code.

The text **`?>`** means that the PHP code ends at this point

Note: Think of these as being PHP's equivalent to our fictional **`<bananaCode>`** and **`</bananaCode>`** start and end tags…

My first PHP "program"

```
<?php
    echo '<h1>Hello world</h1>';
?>
```

Anything between **`<?php`** and **`?>`** is interpreted and run by the server as PHP code

Only the output from running this code gets sent to the browser (or *client*). The browser has no sight of any of this code – it just sees the HTML that results from the server running the code.

My first PHP "program"

- One problem: our previous example would result in invalid HTML being sent to the browser, so actually we should have done something more like this

```
<html>
  <head>
    <title>PHP Hello World</title>
  </head>
  <body>
    <?php
       echo "<h1>Hello world</h1>";
    ?>
  </body>
</html>
```

My first PHP "program"

- One problem: our previous example would result in invalid HTML being sent to the browser, so actually we should have done something more like this

```
<html>
  <head>
    <title>PHP Hello World</title>
  </head>
  <body>
   <?php
      echo "<h1>Hello world</h1>";
   ?>
  </body>
</html>
```

As with our fictional Banana example earlier on, PHP is "slotted" into any HTML code at the point that it occurs. So the result of running this echo statement will be placed into the HTML between the body start and end tags.

A quick lowdown on the grammar and syntax of PHP

- PHP is (mostly) a C-like language
- Each individual statement in PHP ends with a semi-colon
  - Think of the semi-colon as being like a full stop in English
- Just as is the case with all programming languages, think of the text-based code as still being block based
- Blocks start with a curly bracket { and end with the corresponding curly bracket }
- Variables are a bit weird in PHP – they must start with a dollar sign
  - After the dollar sign common variable name rules apply, including
    - Names cannot start with a number (although you can have a number in the variable name after the first letter)
    - Names cannot contain spaces or punctuation apart from the underscore character _

An example - our dice program in PHP

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <?php
      $dicenum = rand(1,6);
      echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
    ?>
  </body>
</html>
```

An example - our dice program in PHP

```
<html>
  <head>
    <title>Dice throw</title>
  </head>
  <body>
    <p>You have thrown a</p>
    <?php
      $dicenum = rand(1,6);
      echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
    ?>
  </body>
</html>
```

Remember: Only the code within the PHP start and end markers gets run by the server as PHP code. The rest gets passed along to the browser as is.

Banana v PHP

```
<bananaCode>
  set diceNum = random(5)+1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>

<?php
  $dicenum = rand(1,6);
  echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
?>
```

Banana v PHP

```
<bananaCode>
  set diceNum = random(5)+1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>


<?php
  $dicenum = rand(1,6);
  echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
?>
```

When assigning a value to variables in PHP, you don't need a command like set. Just put a dollar, followed by the variable name and then a single equals.

Banana v PHP

```
<bananaCode>
  set diceNum = random(5) +1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>


<?php
  $dicenum = rand(1,6) ;
  echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
?>
```

PHP's random number command, **`rand`**, is a bit nicer than the one we have in Banana. You specify the minimum and maximum value (so no messing around with +1)

Banana v PHP

```
<bananaCode>
  set diceNum = random(5) +1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>

<?php
  $dicenum = rand(1,6);
  echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>';
?>
```

The `display` command in Banana maps directly on to the `echo` command in PHP.

Banana v PHP

```
<bananaCode>
   set diceNum = random(5) +1
   display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>


<?php
   $dicenum = rand(1,6);
   echo '<img src="http://paulneve.com/dice'.$dicenum.'.png"/>` ;
?>
```

In both PHP and in HTML itself, you can use single quotes ‘ and double quotes “ interchangeably. This simplifies things when you need to send some HTML from your server side process that includes quotes.

In the Banana example, we have to encase our string literal in double quotes. Thus the HTML quote inside the quotes uses single quotes. In the PHP example, we encase our string literal in single quotes, meaning the HTML inside can use double quote.

Banana v PHP

```
<bananaCode>
  set diceNum = random(5) +1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>


<?php
  $dicenum = rand(1,6);
  echo '<img src="http://paulneve.com/dice' .$dicenum. '.png"/>' ;
?>
```

In both examples, we build a URL by concatenating strings with the value of our **`dicenum`** variable. Our target is a URL that looks like the following

`http://paulneve.com/diceX.png`

where the red X is the value of our **`dicenum`** variable.

Note that in PHP (irritatingly) the operator for concatenation (i.e. joining strings together) is the full stop . and not the plus + sign!

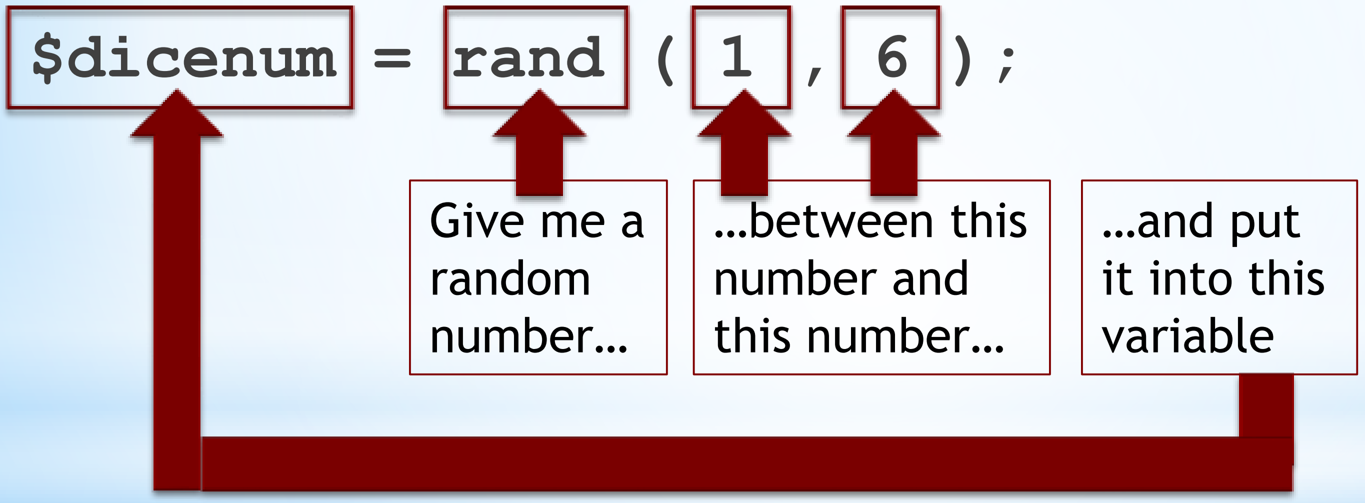
Banana v PHP

```
<bananaCode>
  set diceNum = random(5) +1
  display "<img src='http://paulneve.com/dice" + dicenum + ".png'/>"
</bananaCode>

<?php
  $dicenum = rand(1,6) ;
  echo '<img src="http://paulneve.com/dice' .$dicenum. '.png"/>' ;
?>
```

Don’t forget, in PHP (and in all C-like languages each statement MUST end with a semi-colon – just like a full stop in English.

Protip

- Do make a point of observing and making a note of the random number function
- You will be using it a LOT in this week's exercises ☺



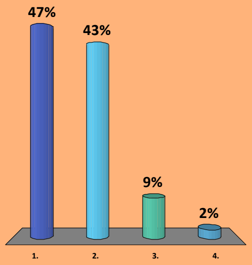
```
$dicenum = rand ( 1 , 6 ) ;
```

Give me a random number...
...between this number and this number...
...and put it into this variable

1.
2.
3.
4.

What would be the correct output of the following code?

```
<html>
  …head element goes here…
  <body>
    <h1>Hello
    <?php
       echo 'There';
    ?></h1>
  </body>
</html>
```



1 **Hello There**

2 **Hello**

There

3 **Hello**

There

4 **Hello** There

PHP is injected at the exact point the code occurs

```
<html>
  …head element goes here…
  <body>
    <h1>Hello
    <?php
       echo 'There';
    ?></h1>
  </body>
</html>
```

```
<html>
  …head element goes here…
  <body>
    <h1>Hello
    There</h1>
  </body>
</html>
```

Hello There

Basic constructs: IF statement

- Let's assume that in both cases, age was a variable that contained a number:

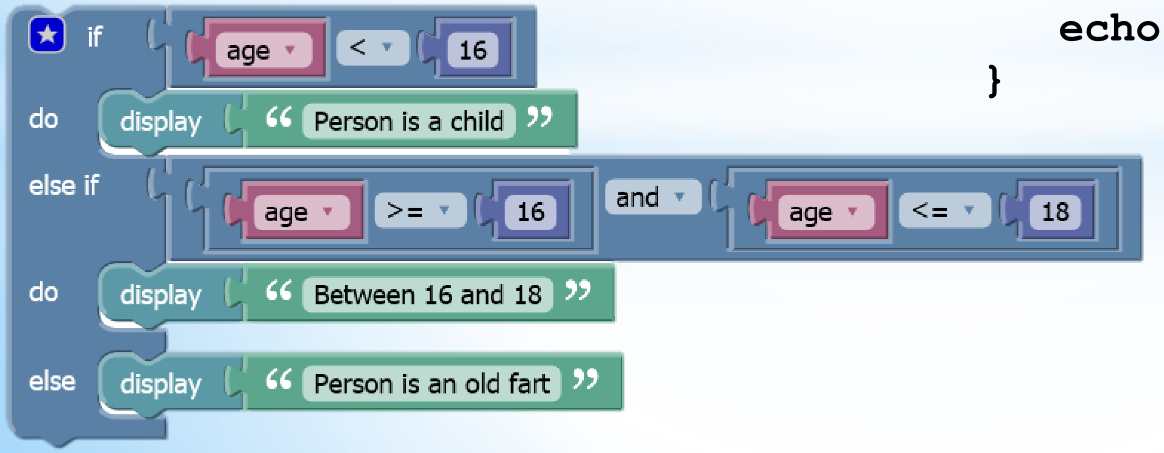
Banana

```
if age < 16
   display "Person is a child"
elseif age >= 16 and age <= 18
   display "Between 16 and 18"
else
   display "Person is an old fart"
endif
```

PHP

```
if ($age < 16)
{
   echo "Person is a child";
}
else if ($age >= 16 && $age <= 18)
{
   echo "Between 16 and 18";
}
else
{
   echo "Person is an old fart"
}
```

Blocks



```
if      age < 16
do      display "Person is a child"
else if age >= 16 and age <= 18
do      display "Between 16 and 18"
else    display "Person is an old fart"
```

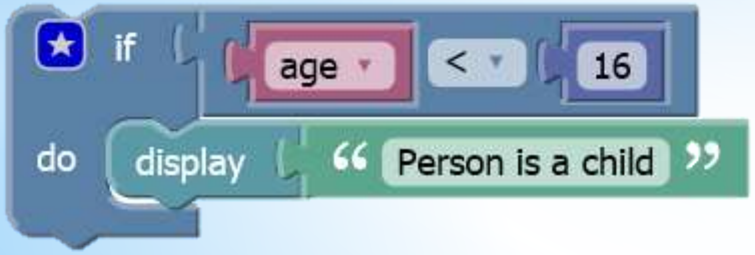
Think in Code Blocks

- In PHP (and in all C-like languages), code blocks are indicated by curly brackets:

```
if ($age < 16)
{
   echo "Person is a child";
}
```

In this PHP IF statement, the curly brackets show the start and end of the encasing IF block.

The opening curly bracket shows where the inside of the IF starts. The closing curly bracket is equivalent to the ENDIF in Banana.



```
if age < 16
   display "Person is a child"
endif
```

Basic constructs: IF statement

- Let’s assume that in both cases, age was a variable that contained a number:

Banana

```
if age < 16
   display "Person is a child"
elseif age >= 16 and age <= 18
   display "Between 16 and 18"
else
   display "Person is an old fart"
endif
```

PHP

```
if ($age < 16)
{
   echo "Person is a child";
}
else if ($age >= 16 && $age <= 18)
{
   echo "Between 16 and 18";
}
else
{
   echo "Person is an old fart"
}
```

Conditions in PHP must be enclosed in brackets. Also note that you cannot use the simple English version of `and` – you must use the symbol based logical operators e.g. `&&` for and, `||` for or.

Basic constructs: IF statement

- Let’s assume that in both cases, age was a variable that contained a number:

Banana

```
if age < 16
   display "Person is a child"
elseif age >= 16 and age <= 18
   display "Between 16 and 18"
else
   display "Person is an old fart"
endif
```

PHP

```
if ($age < 16)
{
   echo "Person is a child";
}
else if ($age >= 16 && $age <= 18)
{
   echo "Between 16 and 18";
}
else
{
   echo "Person is an old fart"
}
```

PHP can use either `elseif` or `else if`, i.e. two words separated by a space. Get in the habit of using the latter. It will stand you in good stead in other C-like languages which do not allow `elseif` with no space.

Basic constructs: IF statement

- Let’s assume that in both cases, age was a variable that contained a number:

Banana

```
if age < 16
   display "Person is a child"
elseif age >= 16 and age <= 18
   display "Between 16 and 18"
else
   display "Person is an old fart"
endif
```

PHP

```
if ($age < 16)
{
   echo "Person is a child";
}
else if ($age >= 16 && $age <= 18)
{
   echo "Between 16 and 18";
}
else
{
   echo "Person is an old fart"
}
```

Note how the curly brackets wrap each block of code that belongs with each if/else condition. You must always have a matching closing bracket for each opening bracket!

A common mistake

- The semi-colon in C-like languages indicates the end of a statement
- So what would be wrong with this picture?

```
if ($age < 16);
{
   echo "Person is a child";
}
```

A common mistake

- The semi-colon in C-like languages indicates the end of a statement
- So what would be wrong with this picture?

```
if ($age < 16)~~;~~
{
   echo "Person is a child";
}
```

In this case, the if statement ends here – at the end of the code block!

If you put a semi-colon at the end of the if line itself, you are essentially saying "if the condition is true do what's between the condition and the end of the statement" – i.e. nothing!

Relational operators

Operator	Meaning	Example
>	greater than	`if ($number > 40)`
<	less than	`if ($height < 1.5)`
==	equals	`if ($counter == 0)`
!=	not equals	`if ($records != 1)`
>=	greater than or equal to	`if ($students >= 10)`
<=	less than or equal to	`if ($result <= -5)`

Relational operators and boolean values

- Relational operators result in a **boolean** value
  - A boolean value has two possible states – TRUE or FALSE
- If a variable `value` contains 7

```
$value < 5          false
$value > 5          true
$value == 5         false
$value != 5         true
$value == 7         true
```

The grammar of relational operators and boolean values

- An **expression** involving relational operators will result in a single value, just as with mathematical operators
- The single value that results from an expression with relational operators can only be TRUE or FALSE – i.e. a boolean
- So our IF statements, grammatically speaking, expect something that will end up as a single boolean value

The grammar of relational operators and boolean values

```
if (an expression that results in true or false)
{
    …do something
}
```

```
$value = 7;
if ($value == 7)
{
   $value = 0;
}
```
✓

```
$value = 7;
if ($value > 5)
{
   $value = 0;
}
```
✓

```
$value = 7;
if ($value + 5)
{
   $value = 0;
}
```
✗

```
$value = 7;
if ($value + 5 == 13)
{
   $value = 0;
}
```
✓

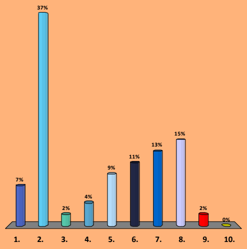
Logical operators

Operator	Name	Description
||	OR	If ANY of the conditions are true, this operator will return TRUE. If ALL of the conditions are false, it will return FALSE
&&	AND	If ALL of the conditions are true, this operator will return TRUE. If ANY of the conditions are false, it will return FALSE.
!	NOT	Reverses a condition – so if the original condition was true, this will return FALSE. If the original condition was false, this will return TRUE

What would be the output of the following PHP code?

```
$woot = 17;
if ($woot > 21 && $woot < 30)
{
   echo "Apricot";
}
if ($woot == 17)
{
   echo "Banana";
}
if ($woot < 12)
{
   echo "Orange";
}
else
{
   echo "Apple";
}
```

1. Apricot
2. Banana
3. Orange
4. Apple
5. ApricotBanana
6. BananaOrange
7. OrangeApple
8. BananaApple
9. ApricotApple
10. ApricotBanana



IF statements are not implicitly related to each other, even if they come after each other…

```
$woot = 17;
```

```
if ($woot > 21 && $woot < 30)
{
  echo "Apricot";
}
```

```
if ($woot == 17)
{
  echo "Banana";
}
```

```
if ($woot < 12)
{
  echo "Orange";
}
else
{
  echo "Apple";
}
```

- Just because an IF statement comes straight after another IF statement, doesn't mean they are related
- Each IF statement is evaluated in isolation and independently of others – even if they come after each other
- The only exception is when you use ELSE (or ELSE IF) – in which case, if one of the conditions is true, the other conditions are not evaluated

Introducing the FOR loop in PHP

```
for ( $count = 0 ; $count < 5 ; $count++)
{
   echo "PHP is awesome";
}
```

Declares and initialises the counter variable – in this case, the counter variable is called *count* and it starts at zero.

Specifies the *condition* under which the loop continues to repeat – in this case it keeps going while it's less than 5… i.e. it does it UNTIL it gets to 4…

Introducing the FOR loop in PHP

Declares and initialises the counter variable – in this case, the counter is *i* and it starts at zero.

```
for ( $count = 0 ; $count <= 4 ; $count++ )
{
   echo "PHP is awesome";
}
```

equivalent to the Banana code

```
FOR I = 0 TO 4 STEP 1
DISPLAY "PHP is awesome"
ENDFOR
```

Specifies the *condition* under which the loop continues to repeat – in this case it keeps going UNTIL it gets to 5.

What happens to the counter variable every time we repeat the loop - in this case, it is increased by 1.

Introducing the FOR loop in PHP

```
for ( $count = 0 ; $count <= 4 ; $count++ )
{
   echo "PHP is awesome";
}
```

equivalent to the Banana code

```
FOR I = 0 TO 4 STEP 1
DISPLAY "PHP is awesome"
ENDFOR
```

This is a mathematical operation – basically, we are saying "increase **count** by 1"

We could equally have said **$count = $count + 1** here…

Counting down

Declares and initialises the counter variable – in this case, the counter is *i* and it starts at ten.

```
for ( $count = 10 ; $count > 0 ; $count-- )
{
   echo "PHP is awesome";
}
```

Specifies the *condition* under which the loop continues to repeat – in this case it keeps going while it's more than zero

equivalent to the Banana code

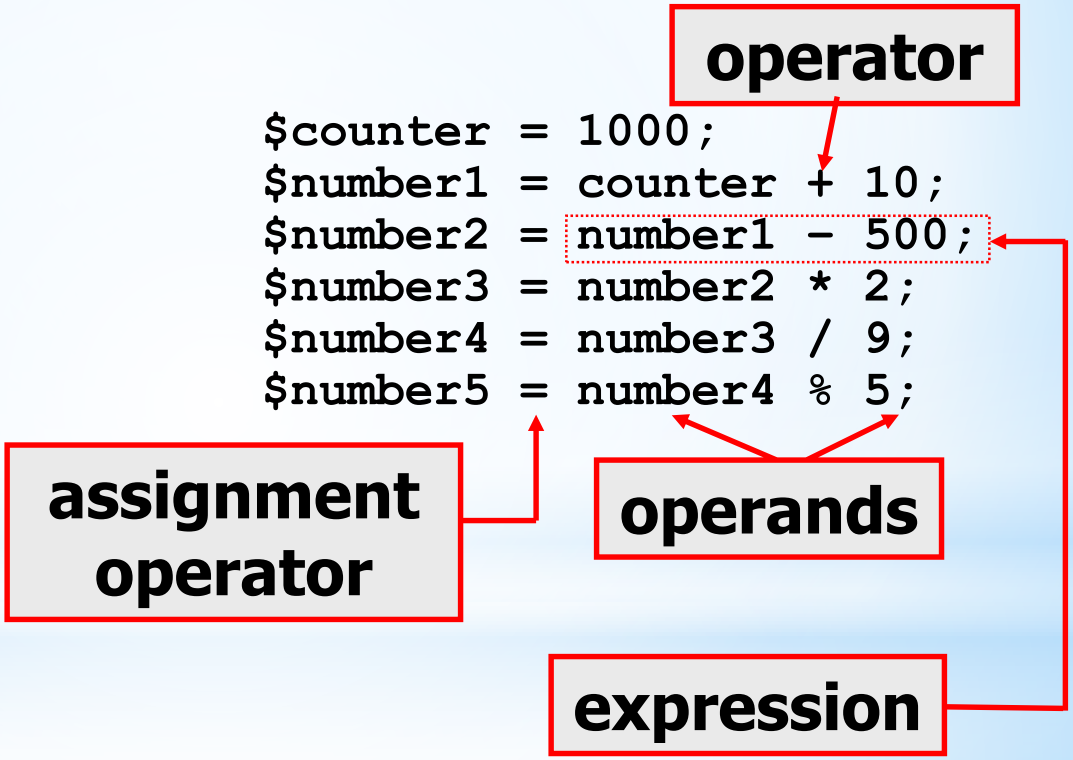
```
FOR I = 10 TO 1 STEP -1
DISPLAY "PHP is awesome"
ENDFOR
```

What happens to the counter variable every time we repeat the loop - in this case, it is decreased by 1.

We could also have used **`$count = $count - 1`**

A tangent: PHP Arithmetic Operators

+	add
-	Subtract
*	multiply
/	divide
%	remainder



```
$counter = 1000;
$number1 = counter + 10;
$number2 = number1 – 500;
$number3 = number2 * 2;
$number4 = number3 / 9;
$number5 = number4 % 5;
```

Shorthand

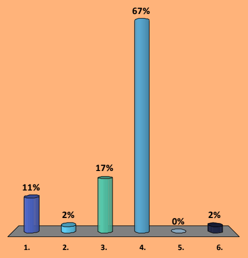
```
$myNumber = 10;                 $myNumber = 10;
$myNumber = $myNumber + 5;      $myNumber += 5;


$myNumber = 10;                 $myNumber = 10;
$myNumber = $myNumber + 1;      $myNumber++;
```

- Also for operators other than plus, e.g.

```
$myNumber = 10;
$myNumber = $myNumber--;    // myNumber becomes 9
$myNumber *= 2;             // myNumber becomes 18
```

What would this code display in the browser?

```
(rest of HTML document assumed)
<body>
  <p>
    <?php
      for ($count = 10; $count > 5; $count--)
      {
        echo $count;
      }
    ?>
  </p>
</body>
```



1. 5678910
2. 1098765
3. 678910
4. 109876
5. 12345678910
6. 10987654321

The WHILE loop

```
$count = 0;
$end = rand(0,10);
while ($count < $end)
{
   echo "<p>PHP is repeatedly awesome</p>";
   $count++;
}
```

equivalent to the Banana code

```
SET count = 0
SET end = random(10)
WHILE count < end
  DISPLAY "<p>Banana is repeatedly awesome</p>"
  count = count + 1;
ENDWHILE
```

DO-WHILE

```
$count = 0
$end = rand(0,10);
do
{
    echo "<p>PHP rocks</p>";
    $count++;
} while ($count != $end);
```

← **note the semicolon**

equivalent to the pseudocode

```
SET count = 0
SET end = random(10)
REPEAT
  DISPLAY "<p>Banana is repeatedly awesome"
  count = count + 1
UNTIL count == end
```

DO-WHILE

The **condition** specifies the *condition* under which the loop continues to repeat – in this case it keeps going WHILE the variable **`count`** is not equal to the variable **`end`**.

```
$count = 0
$end = rand(0,10);
do
{
   echo "<p>PHP rocks</p>";
   $count++;
} while ($count != $end) ;
```

<- note the semicolon

equivalent to the pseudocode

```
SET count = 0
SET end = random(10)
REPEAT
  DISPLAY "<p>PHP rocks</p>"
  count = count + 1
UNTIL count == 5
```

While the **condition** evaluates to true, these statements while repeatedly run.

IMPORTANT: note the fundamental difference between the REPEAT/UNTIL construct you saw in Banana and PHP's DO/WHILE.

- REPEAT/UNTIL repeats **until** the final condition is TRUE.
  - So when it is true it stops!
- DO/WHILE in PHP repeats **while** the condition is true
  - So when it is true it repeats!

WHILE v DO/WHILE in a nutshell

```
$count = 0;
$end = rand(0,10);
while ($count < $end)
{
   echo "<p>PHP rocks</p>";
   $count++;
}

$count = 0
$end = rand(0,10);
do
{
   echo "<p>PHP rocks</p>";
   $count++;
} while ($count < $end);
```

- What would happen in both cases if the random number chosen was 0?

The moral of the story: deciding which loop to use and when

- `for` loop - when the number of repetitions can be determined *before the loop is entered*
- `while` loop - if the number of repetitions cannot be determined before the loop is entered
- `do-while` loop - same as a `while` loop, but *the statements are executed at least once* (same as a repeat/until in Banana)

Functions

```
function diceThrow($sides)
{
   $result = rand(1,$sides);
   return $result;
}
```

```
function diceThrow(sides)
   set result = random(sides-1)+1
   return result
endfunction
```

Functions

```
function diceThrow( $sides )
{
   $result = rand(1,$sides);
   return result;
}
```

As with Banana, function parameters go between brackets. If there's more than one function separate them with commas.

Function parameters in PHP need to have dollar signs $ in front of them just like any other variable.

```
function diceThrow( sides )
   set result = random(sides-1)+1
   return result
endfunction
```

Functions

```
function greeting()
{
   echo "Hello there!"
}

function greeting
   display "Hello there!"
endfunction
```

Functions

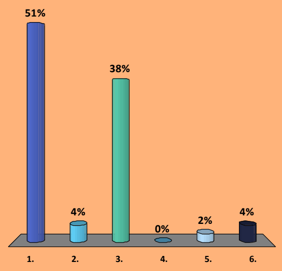
```
function greeting ()
{
   echo "Hello there!"
}

function greeting
   display "Hello there!"
endfunction
```

UNLIKE Banana, if a function doesn't have any parameters, you must still include the brackets after the function name. If there are no parameters just put nothing between the brackets.

What would be the output of the following PHP code?

```
function foo()
{
  echo "<p>Kevin</p>";
}

function bar()
{
  echo "<p>Jane</p>";
}

function whee($stuff)
{
  if ($stuff > 10)
  {
    echo "<p>Abdul</p>";
  }
  else
  {
    echo "<p>Clara</p>";
  }
}

bar();
whee(12);
foo();
```

1. Kevin/Jane/Abdul
2. Kevin/Jane/Clara
3. Jane/Abdul/Kevin
4. Jane/Clara/Kevin
5. Jane/Kevin
6. Something else



Single lines of PHP

- If your PHP code is short (e.g. you're just injecting a variable into some HTML) you can put the PHP start and end tag on a single line around the PHP code itself
- For example, you could do something like this:

```
<?php $dicenum = rand(1,6); ?>
<html>
  <head>
    <title>An example</title>
  </head>
  <body>
    <p>The dice rolled is: <?php echo $dicenum ?></p>
  </body>
</html>
```

Single lines of PHP

- If your PHP code is short (e.g. you're just injecting a variable into some HTML) you can put the PHP start and end tag on a single line around the PHP code itself
- For example, you could do something like this:

```
<?php $dicenum = rand(1,6); ?>
<html>
  <head>
    <title>An example</title>
  </head>
  <body>
    <p>The dice rolled is: <?php echo $dicenum ?> </p>
  </body>
</html>
```

Red: PHP start tags

Blue: PHP end tags

Green: PHP code

Even shorter shorthand…

- You can also do

```
<?php $dicenum = rand(1,6); ?>
<html>
  <head>
    <title>An example</title>
  </head>
  <body>
    <p>The dice rolled is: <?= $dicenum ?></p>
  </body>
</html>
```

- Note the <?= start tag rather than <?php
  - This is a shorthand for <?php echo
  - In a nutshell - it means that whatever value is between the start and end tags will be directly injected into the HTML output
- WARNING: THIS ONLY WORKS ON NEWER VERSIONS OF PHP, OR ON OLDER VERSIONS THAT HAVE THE PARAMETER **`short_open_tag`** ENABLED
  - translation: if your hosting company uses an old version of PHP, you will need to bug them to make a parameter change ☺

PHP file naming conventions

- Static HTML pages usually have the extension .htm or .html
- PHP pages have (surprisingly) the extension .php
- If you do not have this extension at the end of your filename, any PHP code in it will not get run by the server
- In NoobLab, the default page is index.html
  - Either click on its tab to rename it, or create a new page called index.php and delete the index.html

Summary

- HTML pages are static - they never change even if you reload them a million times
- Server-side scripting platforms like PHP give us a way to create dynamic HTML pages that can change each time
- The programmer creates a program that runs on the server and generates HTML as its output
- The program runs on the server and sends back this output
- The browser (or client) has no idea that the HTML it receives was generated by a program, or indeed, that it might change the next time it's accessed - it just receives the HTML from the server and renders it in a form the user can read

Summary

- PHP is (mostly) C-like
- Use curly brackets to denote code blocks (as opposed to END keywords like ENDIF, ENDFOR as found in Banana)
- PHP variables are prefixed with dollar signs
- The usual programming constructs exist in PHP, e.g. IF, FOR, WHILE, DO/WHILE and functions
  - DO/WHILE is analogous to REPEAT/UNTIL in Banana, but the condition is reversed