introduction to

SCRIPTING, DATABASES, SYSTEM ARCHITECTURE

PHP II: while loops, for loops, functions

Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

IT University of Copenhagen

Example PHP Programs

- PHP Program examples:
  - `( http://www.itu.dk/people/brabrand/DSDS/ )`
- Note you can't see the PHP source code (because it gets executed by the server), so you can use the "**`show.php`**" script:
  - `( show.php?file=bmi.php )`
- Example:

(http://www.itu.dk/people/brabrand/DSDS/show.php?file=bmi.php)

Agenda

- 1) RECAP (variable transfer)
- 2) GET vs POST (form submission)
- 3) LOOPS (the 'while' loop)
- 4) LOOPS (the 'for' loop)
- 5) FUNCTIONS (introduction)

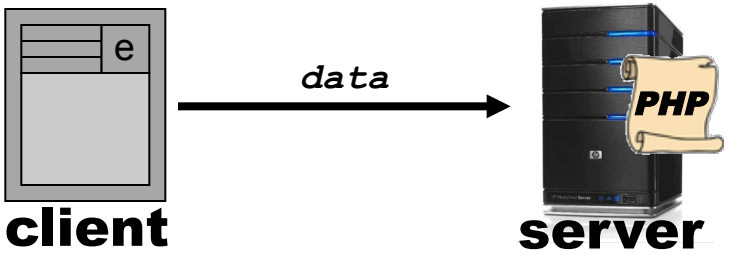
Variable Transfer

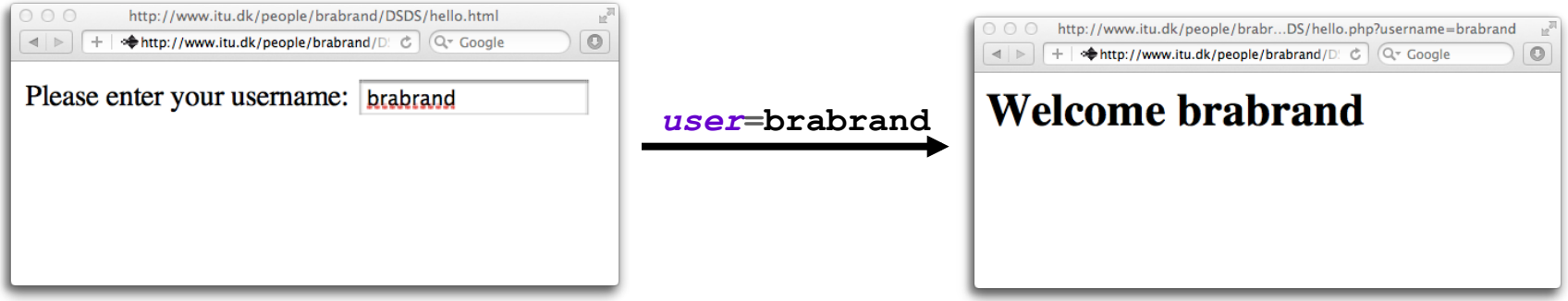
```
<html>
  <body>
    <form action="hello.php">
      Please enter your username:
      <input type="text" name="user"/>
    </form>
  </body>
</html>
```

hello.html

```
<html>
  <body>
    <?php
      $u = $_REQUEST['user'] ;
      echo "<h1>Welcome $u</h1>" ;
    ?>
  </body>
</html>
```

hello.php





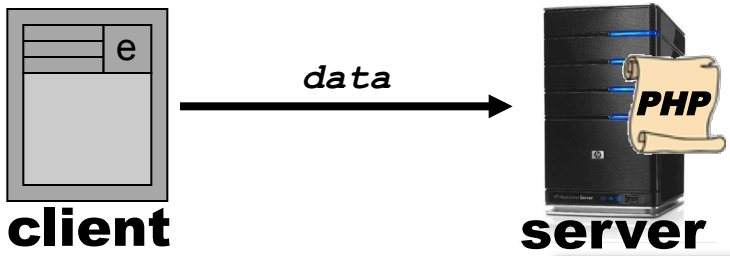
Currency Conversion (arith)

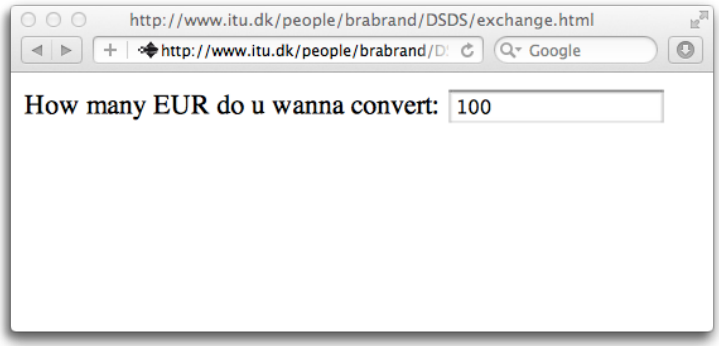
```
<html>
  <body>
    <form action="exchange.php">
      How many EUR do u wanna convert:
      <input type="text" name="euro"/>
    </form>
  </body>
</html>
```

exchange.html

```
<html>
  <body>
    <?php
      $eur = $_REQUEST['euro'] ;
      $dkk = ($eur * 7.5); // arith
      echo "You get $dkk kr." ; ?>
  </body>
</html>
```

exchange.php





euro=100

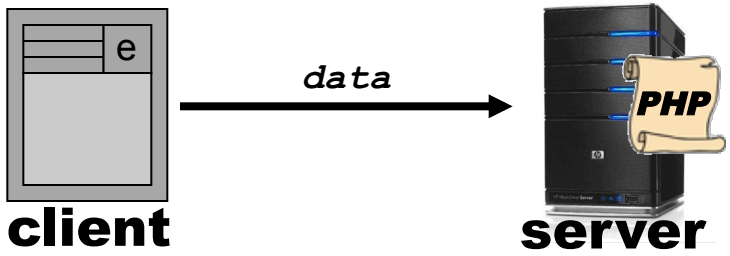
Calculator (multiple numbers)

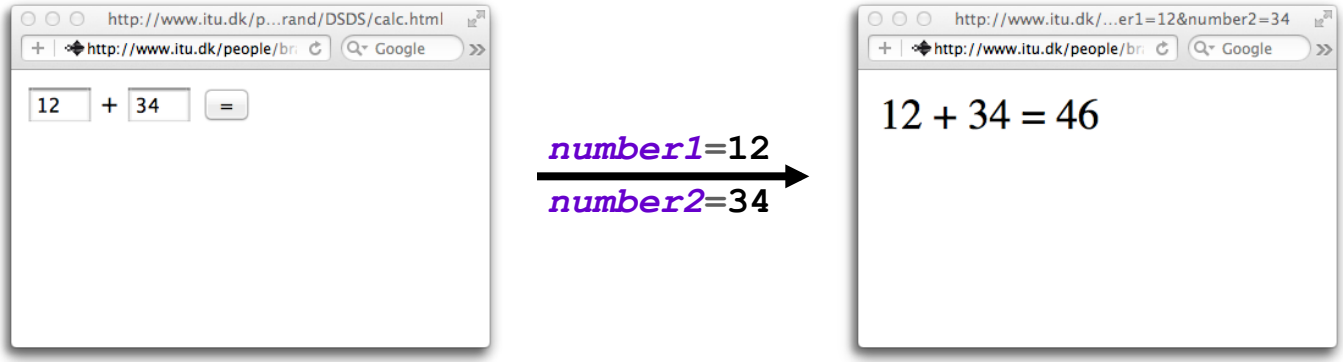
```
<form action="calc.php">
  <input type="text" name="number1"
         size="2"/>
  +
  <input type="text" name="number2"
         size="2"/>
  <input type="submit" value="="/>
</form>
```

calc.html

```
<?php
  $x = $_REQUEST['number1'] ;
  $y = $_REQUEST['number2'] ;

  $res = ($x + $y) ; // addition

  echo "$x + $y = $res " ;
?>
```

calc.php





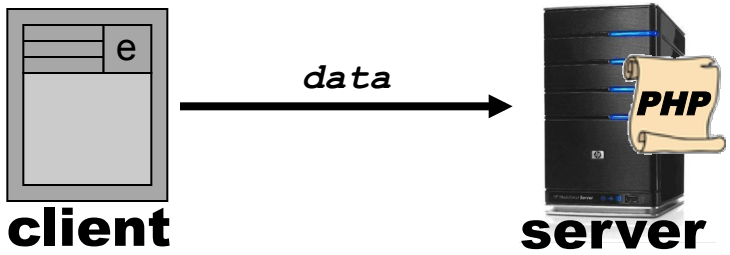
Welcome (concatenation)

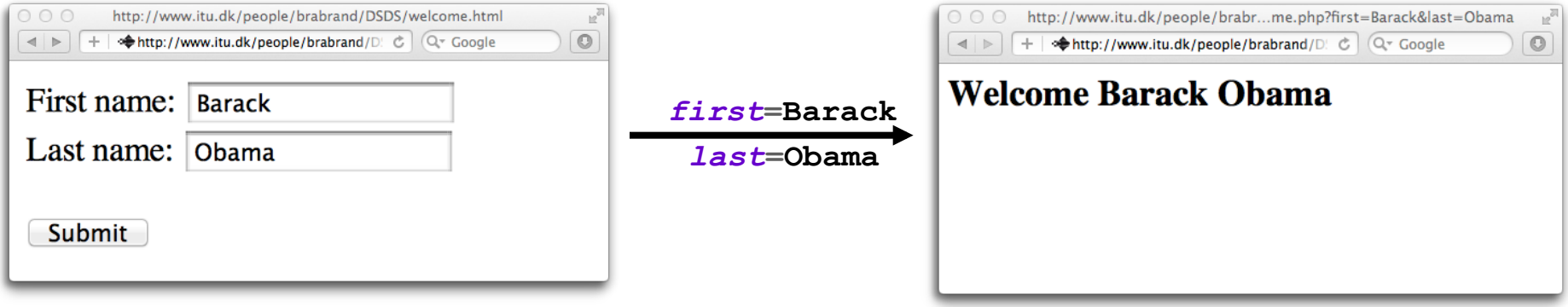
```
<form action="welcome.php">
  First name:
  <input type="text" name="first"/>
  <br/>
  Last name:
  <input type="text" name="last"/>
  <p/><input type="submit"/>
</form>
```

welcome.html

```
<?php
  $x = $_REQUEST['first'] ;
  $y = $_REQUEST['last'] ;

  $name = $x . " " . $y ;//concat

  echo "<h1>Welcome $name</h1>" ;
?>
```

welcome.php





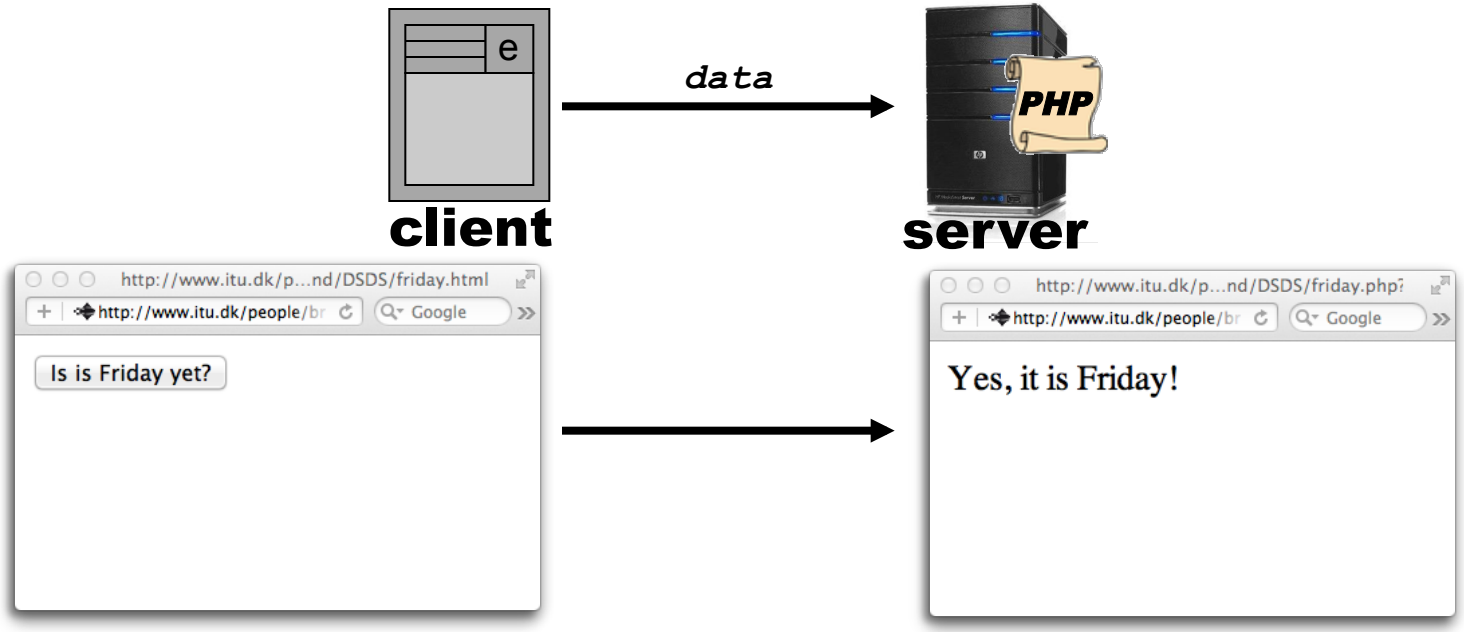
Is-it-Friday-yet? (if-else)

```
<html>
  <body>
    <form action="friday.php">
      <input type="submit"
             value="Is is Friday?"/>
    </form>
  </body>
</html>
```

friday.html

```
<?php
  $day = date("w"); // weekday
  if ($day == 5) {  // if-else
    echo "Yes, it is Friday!" ;
  } else {
    echo "No." ;
  }
?>
```

friday.php

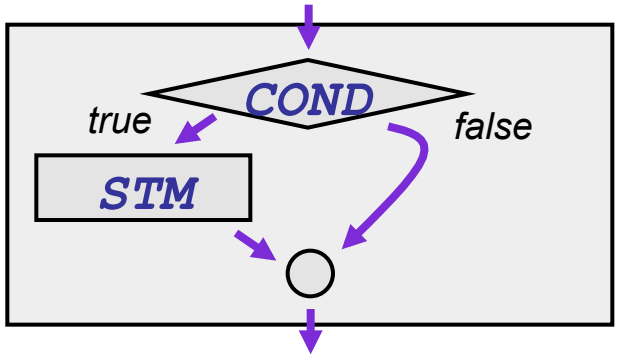


The 'if' Statement

- Syntax:

```
if ( COND ) {
    STM
}
```

- Semantics:



*If the condition (**COND**) evaluates to **true**, the given statement (**STM**) is executed, otherwise not.*

- Example:

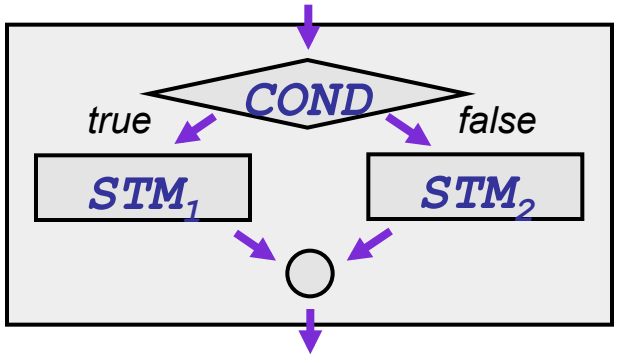
```
$day = date("w");
if ($day == 5) {
    echo "Yes, it is Friday!" ;
}
```

The 'if-else' Statement

- Syntax:

```
if ( COND ) {
    STM1
} else {
    STM2
}
```

- Semantics:



*If the condition (**COND**) evaluates to **true**, statement (**STM**$_1$) is executed, otherwise statement (**STM**$_2$) is executed.*

- Example:

```
$day = date("w");
if ($day == 5) {
    echo "Yes, it is Friday!" ;
} else {
    echo "No." ;
}
```

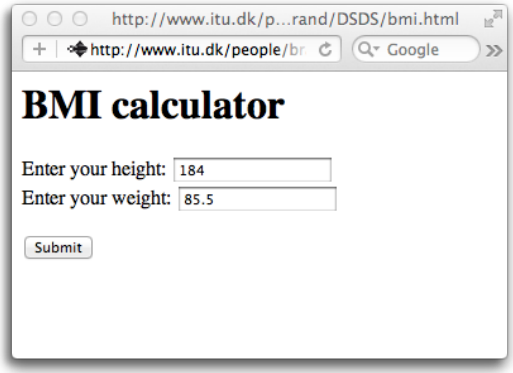
The BMI Calculator (if-elseif-else)

```html
<html>
<body>
<h1>BMI calculator</h1>

<form action="bmi.php">
Enter your height:
<input type="text" name="height" />
<br/>Enter your weight:
<input type="text" name="weight" />
<p/><input type="submit" />
</form>
</body>
</html>
```

bmi.html

```php
<?php
  $h = $_REQUEST['height'] ;
  $w = $_REQUEST['weight'] ;
  $bmi = $w/(($h/100)*($h/100)) ;
  echo "Your BMI: <b>$bmi</b> " ;
  if ( $bmi < 20.0 ) {
    echo "which is too low!" ;
  } elseif ( $bmi > 25.0 ) {
    echo "which is too high!" ;
  } else {
    echo "which is normal." ;
  }
?>
```

bmi.php



height=184
weight=85.5

Agenda

- 1) RECAP (variable transfer)
- 2) GET vs POST (form submission)
- 3) LOOPS (the 'while' loop)
- 4) LOOPS (the 'for' loop)
- 5) FUNCTIONS (introduction)

Form Submission (GET)

```
<html><body>
<form method="get"
      action="get_example.php">
  Please enter your username:
  <input type="text" name="username"/>
</form>
</body></html>
```

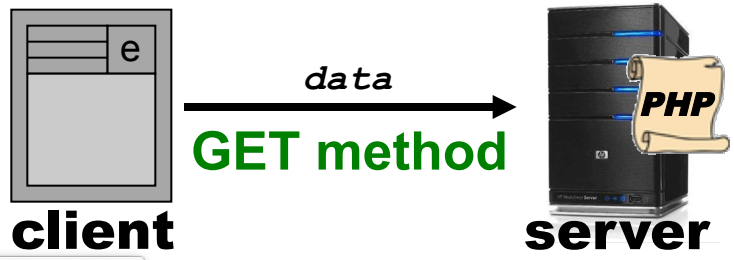
get_example.html

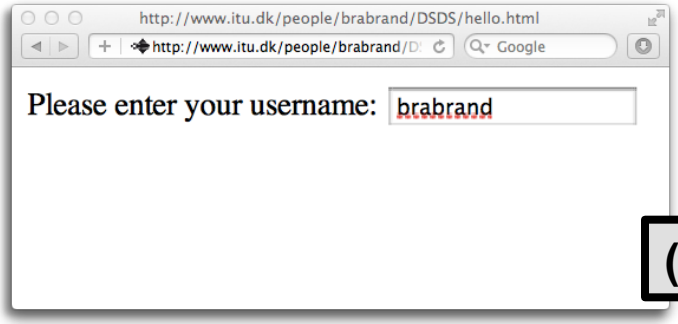
Note: "get" is default

```
<html><body>
  <?php
    $u = $_GET['username'] ;

    echo "<h1>Welcome $u</h1>" ;
  ?>
</body></html>
```

get_example.php

Note: "$_REQUEST['...']" also works as well as $_GET['...'] here





username=brabrand

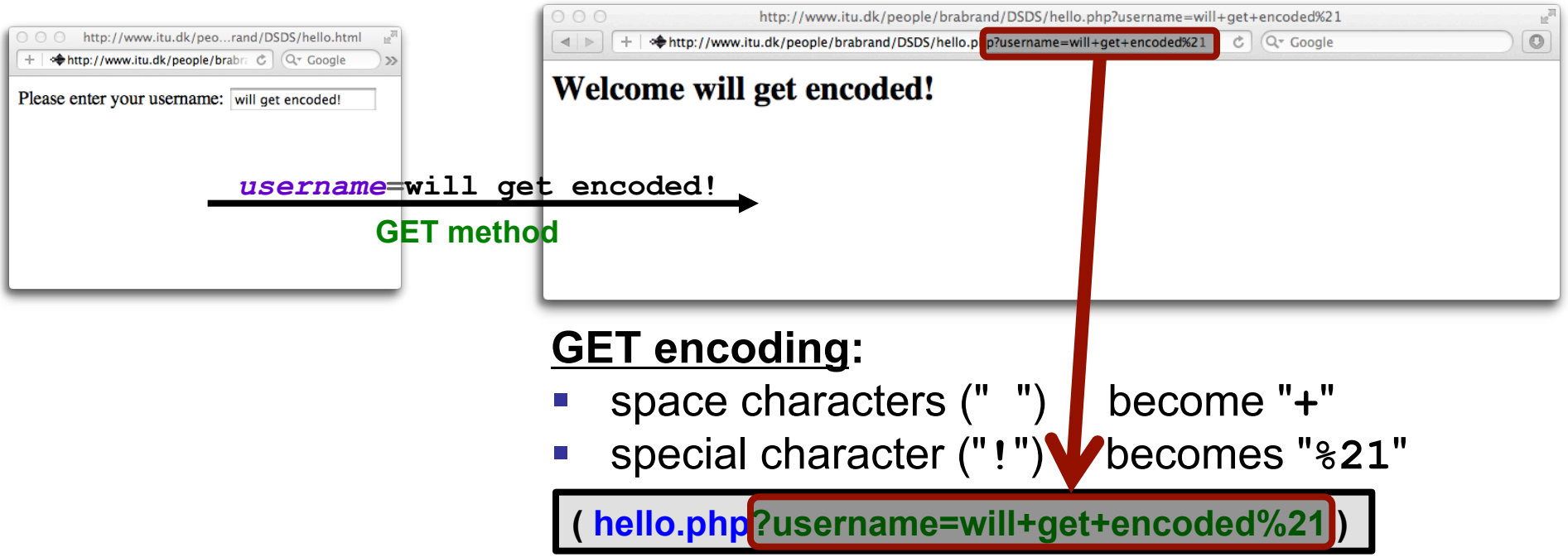
(http://.../get_example.php?username=brabrand)

"GET encoded"

Form Submission (GET)

```
<html><body>
<form method="get"
      action="get_example.php">
  Please enter your username:
  <input type="text" name="username"/>
</form>
</body></html>
```
hello.html

```
<html><body>
  <?php
    $u = $_GET['username'] ;

    echo "<h1>Welcome $u</h1>" ;
  ?>
</body></html>
```
hello.php



GET encoding:

- space characters (" ") become "+"
- special character ("!") becomes "%21"

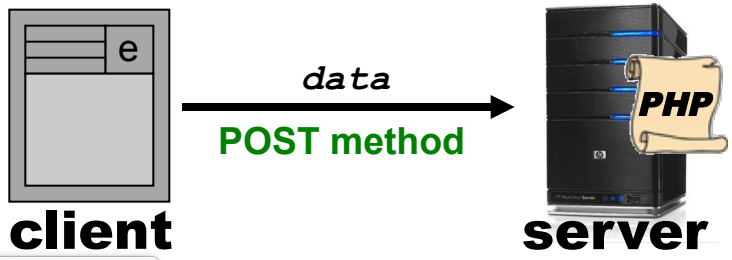
(hello.php?username=will+get+encoded%21)

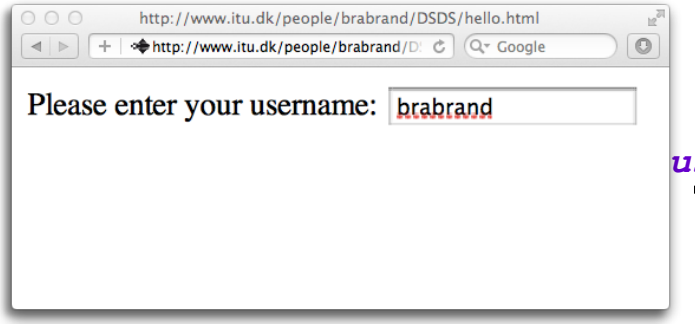
Form Submission (POST)

```
<html><body>
<form method="post"
      action="post_example.php">
  Please enter your username:
  <input type="text" name="username"/>
</form>
</body></html>
```
post_example.html

```
<html><body>
  <?php
    $u = $_POST['username'] ;

    echo "<h1>Welcome $u</h1>" ;
  ?>
</body></html>
```
post_example.php

Note: "$_REQUEST['...']" works for both get and post



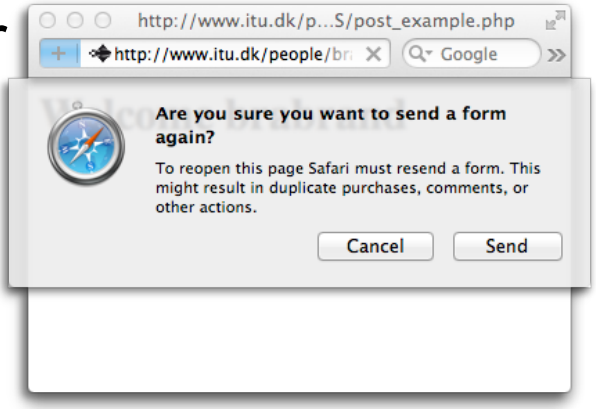


username=brabrand
POST method

(http://.../post_example.php)

GET vs POST?

- **GET** requests should ideally be ***idempotent:***
  - (means: accessing multiple times = accessing once)
  - **Example:** useful for retrieving data, not for submitting orders to an online shop
  - **GET:** safe to reload page in browser
  - **POST:** browser asks:
    - `"repost form data?"`
  - **GET** may be cached by the browser



- **Note: GET** has limits on URI length
- **Note: POST** allows for other encodings
  - In particular, file uploading is permitted

Agenda

- 1) RECAP (variable transfer)
- 2) GET vs POST (form submission)
- 3) LOOPS (the 'while' loop)
- 4) LOOPS (the 'for' loop)
- 5) FUNCTIONS (introduction)

Loops

- A ***loop*** is a piece of code that is repeatedly executed many times:

```php
<?php
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo "<li> I will not use Facebook in class again! </li>" ;
  echo
?>
```

vs

```php
$count = 0;                                          // initialization
while ( $count < 10 ) {
  echo "<li> I will not use Facebook in class again! </li>" ;
  $count = $count + 1 ;                              // incrementation
}
```

Loops (another example)

- A ***loop*** is a piece of code that is repeatedly executed many times:

```php
<?php
  echo "<li> 0 </li>" ;
  echo "<li> 1 </li>" ;
  echo "<li> 2 </li>" ;
  echo "<li> 3 </li>" ;
  echo "<li> 4 </li>" ;
  echo "<li> 5 </li>" ;
  echo "<li> 6 </li>" ;
  echo "<li> 7 </li>" ;
  echo "<li> 8 </li>" ;
  echo "<li> 9 </li>" ;
?>
```

vs

```php
$count = 0;
while ( $count < 10 ) {
  echo "<li> $count </li>" ;
  $count++ ;
}
```

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

The while loop

- The '**while**' loop is used to execute code repeatedly (until a condition becomes false):

```
while ( condition ) {
    // do something repeatedly...
}
```

- Example:

```
<?php /* calculate square numbers */
  $count = 0 ;
  while ( $count < 10 ) {
    $squared = $count * $count ;
    echo "<li> $count squared is: $squared </li>" ;
    $count++;
  }
?>
```

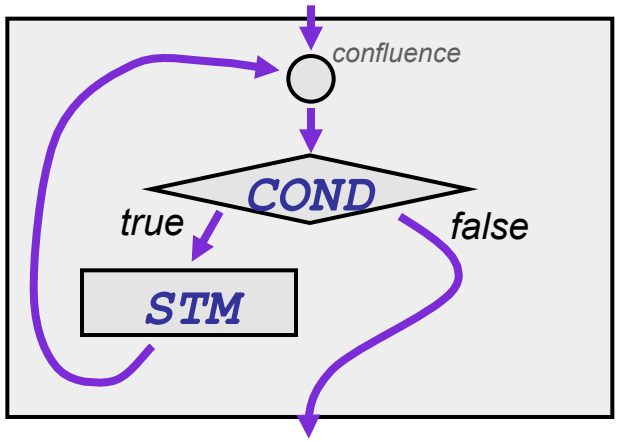
- 0 squared is: 0
- 1 squared is: 1
- 2 squared is: 4
- 3 squared is: 9
- 4 squared is: 16
- 5 squared is: 25
- 6 squared is: 36
- 7 squared is: 49
- 8 squared is: 64
- 9 squared is: 81

The while loop

- syntax:

```
while ( COND ) {
    STM
}
```

- semantics:



*If the condition (`COND`) evaluates to **false**, the given statement (`STM`) is skipped. Otherwise (if the condition was **true**), the statement (`STM`) is executed and afterwards the condition is evaluated again. If it is still **true**, `STM` is executed again... This continues until the condition evaluates to **false**.*

EXERCISE: while loop quizzzz

```php
<?php                          // Program 1
  $i = 10 ;
  while ( $i > 0 ) {
    echo "<li> $i </li>" ;
    $i--;
  }
?>
```

→ Result?

```php
<?php                          // Program 2
  $i = 1 ;
  while ( $i <= 10 ) {
    echo "<li> $i </li>" ;
    $i = $i + 2;
  }
?>
```

→ Result?

```php
<?php                          // Program 3
  $i = 1 ;
  while ( $i < 100 ) {
    echo "<li> $i </li>" ;
    $i = $i * 2;
  }
?>
```

→ Result?

EXERCISE: while loop quizzzz

```php
<?php                          // Program 1
  $i = 0;
  while ( $count <= 15 ) {
    echo "<li> $i </li>" ;
    $i = $i + 3 ;
  }
?>
```

- 0
- 3
- 6
- 9
- 12
- 15

```php
<?php                          // Program 2
  $i = 64 ;
  while (   $i >= 2    ) {
    echo "<li> $i </li>" ;
    $i = $i / 2;
  }
?>
```

- 64
- 32
- 16
- 8
- 4
- 2

```php
<?php                          // Program 3
  $i = 1 ;
  while (  $i <= 10000 ) {
    echo "<li> $i </li>" ;
    $i = $i * 10;
  }
?>
```

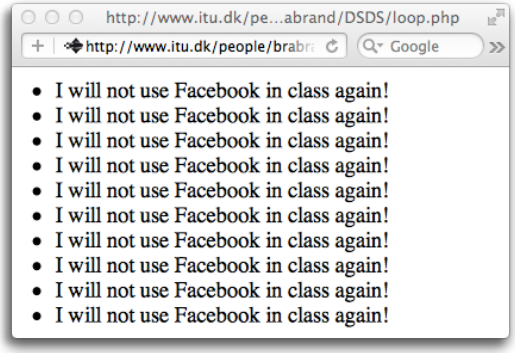
- 1
- 10
- 100
- 1000
- 10000

Agenda

- 1) RECAP (variable transfer)
- 2) GET vs POST (form submission)
- 3) LOOPS (the 'while' loop)
- 4) LOOPS (the 'for' loop)
- 5) FUNCTIONS (introduction)

The 'for' loop

- The **for** loop can do pretty much the same as the **while** loop, but it can be written more compactly:



initialization *condition*

```
$count = 0;
while ( $count < 10 ) {
    echo "<li> I will not use Facebook in class again! </li>" ;
    $count++;
}
```

increment

vs

```
for ( $count = 0 ; $count < 10 ; $count++ ) {
    echo "<li> I will not use Facebook in class again! </li>" ;
}
```

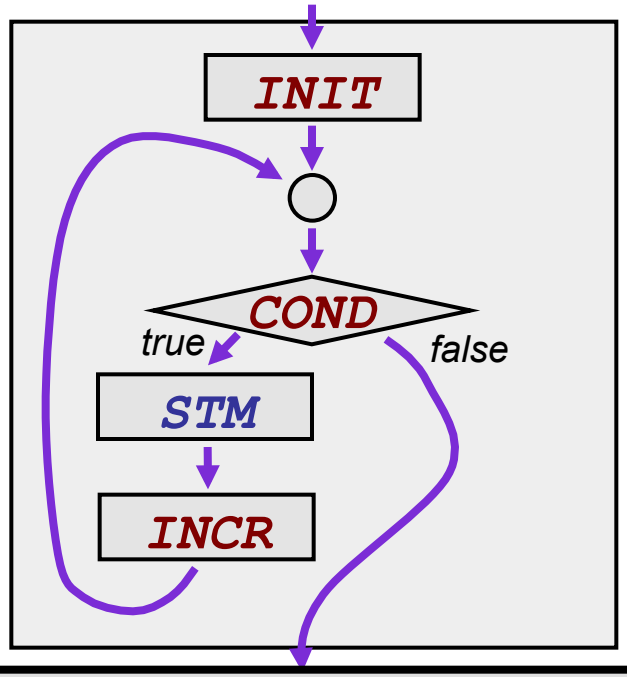
initialization ; condition ; increment

The 'for' loop

- Syntax:

```
for ( INIT ; COND ; INCR ) {
    STM
}
```

- Semantics:



- Example:

```
for ( $count = 0 ; $count < 10 ; $count++ ) {
    echo "<li> I will not use Facebook in class again! </li>" ;
}
```

EXERCISE: for loop quizzzz

```
for ($i = 10; $i >= 0; $i=$i-2) {
   echo "<li> $i </li>" ;
}
```

→ Result?

```
for ($i = 1; $i < 10; $i = $i+3) {
   echo "<li> $i </li>" ;
}
```

→ Result?

```
for ($i = 1; $i < 10; $i = $i*$i) {
   echo "<li> $i </li>" ;
}
```

→ Result?

EXERCISE: for loop quizzzz

```
for ( [redacted] ) {
    echo "<li> $i </li>" ;
}
```

- 0
- 3
- 6
- 9
- 12
- 15

```
for ( [redacted] ) {
    echo "<li> $i </li>" ;
}
```

- 64
- 32
- 16
- 8
- 4
- 2

```
for ( [redacted] ) {
    echo "<li> $i </li>" ;
}
```

- 1
- 10
- 100
- 1000
- 10000

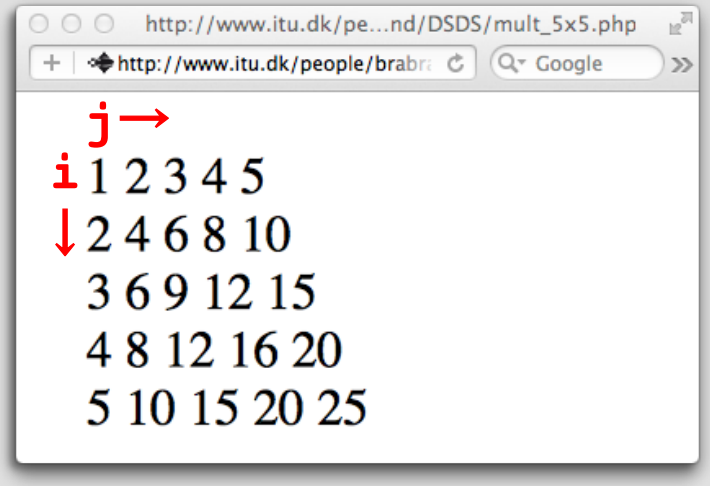
Note on 'for'

- for loops are more common than while loops
  - More compactly written
  - Easier to read (once you get used to it)
  - Higher level of abstraction
  - In while loops it's easy to forget to update the counter
- People often use short counter var names:
  - i, j, k, ...
  - a, b, c, ...
  - x, y, z, ...

Nested Loops

- Multiplication table (of size 5x5):

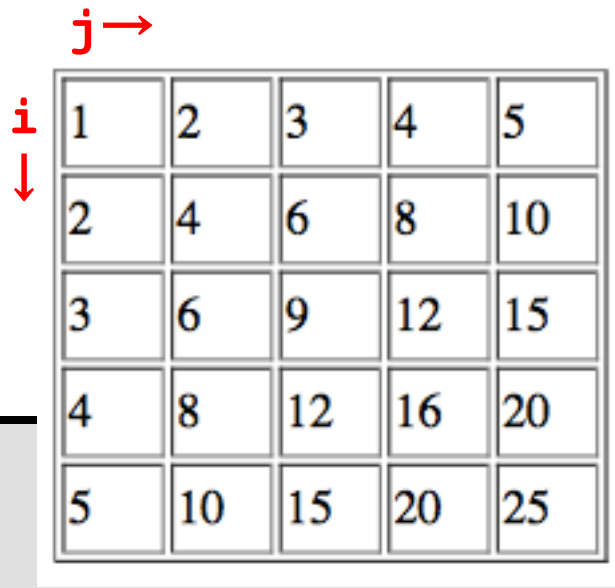
```
/* multiplication table (of size 5x5) */

for ( $i=1 ; $i <= 5 ; $i++ ) {
    for ( $j=1 ; $j <= 5 ; $j++ ) {
        $mult = $i * $j ;
        echo "$mult " ;
    }
    echo "<br/>" ;
}
```



  - The outer loop runs 5 times (i = 1, 2, 3, 4, 5)
  - The inner loop runs 5 times (j = 1, 2, 3, 4, 5), for each **i**
  - Indentation makes code (a lot) easier to read !

EXERCISE: nicer mult-table

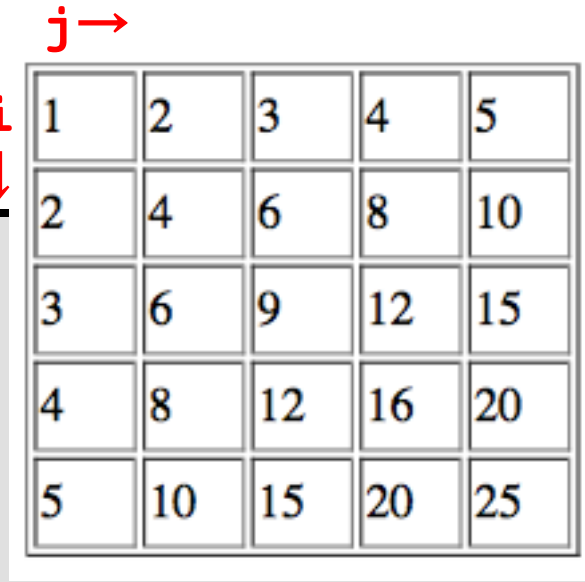
- Let's make it look nicer (as an HTML <table>) so that it looks like this:




```
/* (5x5) multiplication table */

for ( $i=1 ; $i <= 5 ; $i++ ) {
    for ( $j=1 ; $j <= 5 ; $j++ ) {
        $mult = $i * $j ;
        echo "$mult " ;
    }
    echo "<br/>" ;
}
```

EXERCISE: nicer mult-table

- One (of many) solutions:




```
<table border="1">
<?php
   for ( $i=1 ; $i <= 5 ; $i++ ) {
      echo "<tr>" ;
      for ( $j=1 ; $j <= 5 ; $j++ ) {
         $mult = $i * $j ;
         echo "<td width='30' height='30'>$mult</td>" ;
      }
      echo "</tr>" ;
   }
</table>
```

Agenda

- 1) RECAP (variable transfer)
- 2) GET vs POST (form submission)
- 3) LOOPS (the 'while' loop)
- 4) LOOPS (the 'for' loop)
- 5) FUNCTIONS (introduction)

Functions

- A ***function*** is a piece of code we give a name:

```
<?php

    function printMyName() {        // function declaration
      echo "<b>John Doe</b>" ;      // function body
    }
```

- We can then ***use*** this code in other parts of the program:

```
    echo "Hello, my name is " ;
    printMyName();                  // function invocation
?>
```

Hello, my name is **John Doe**

- This is known as ***function invocation*** (or ***function call***) (we ***call*** the function `printMyName`)

Functions (code reuse)

```
<?php
    function printMyName() {        // function declaration
      echo "<b>John Doe</b>" ;      // function body
    }
?>
```

Functions allow us to ***reuse*** code:

```
<?php
    echo "Hello, my name is " ;
    printMyName();                  // function invoked

    echo "<p/>" ;
    printMyName();                  // function invoked again!
    echo " is indeed my name." ;
?>
```

Hello, my name is **John Doe**

John Doe is indeed my name.

Functions (changes are easy)

```
<?php
    function printMyName() {
      echo "<b>Jane Doe</b>" ; // we only need to modify here!
    }
?>
```

We don't have to change the rest of the program:

```
<?php
    echo "Hello, my name is " ;
    printMyName();                    // function invoked

    echo "<p/>" ;
    printMyName();                    // function invoked again!
    echo "is indeed my name." ;
?>
```

Hello, my name is **Jane Doe**

Jane Doe is indeed my name.

Functions (with arguments)

- ***Functions*** can take arguments:

```
<?php

  function printName($name) {
    echo "Hello, my name is <b>$name</b>" ;
  }
```

- which gives us a chance to ***parameterize*** the code (to use the function differently in different contexts):

```
  printName("John Doe");
  echo "<p/>" ;
  printName("Barack Obama");
?>
```

Hello, my name is **John Doe**

Hello, my name is **Barack Obama**

Functions (multiple arguments)

- ***Functions*** can take multiple arguments:

```php
<?php

  function printName2($first, $last) {
    echo "Hello, my name is $last, <em>$first $last</em>." ;
  }

  printName2("James", "Bond");
```

Hello, my name is Bond, *James Bond*.

```php
  printName2("John", "Doe");

?>
```

Hello, my name is Doe, *John Doe*.

Functions (can return a value)

- Functions are allowed to *"return"* a value:

```
function mult($x, $y) {
    $result = $x * $y ;
    return $result ;
}

$val = mult(5,7) ;                          // function invocation
echo "The result of 5*7 is: <b>$val</b>" ;
```

The result of 5*7 is: **35**

```
function emphasize($text) {
    $emph = "<em><b><font color='red'>$text</font></b></em>" ;
    return $emph ;
}

$x = emphasize("never") ;                   // function invocation
echo "I will $x use Facebook in class again!" ;
```

I will ***never*** use Facebook in class again!

Any questions?

We can use built-in functions

We can use ***built-in functions:***

- ***date(x)***

```
$today = date("d-m-Y") ;
echo "Today is: " ;
echo $today ;
```

Today is: 30-9-2011

- ***round*(x)**

```
$pi = 3.1415926 ;
$pi_ish = round($pi) ; // function call
echo "Pi (&pi;) is approximately: $pi_ish" ;
```

Pi (π) is approximately: 3

- ***strlen*(string)**

```
$str = "Hello World" ;
$len = strlen($str) ;
echo "The length of '$str' is: <b>$len</b>" ;
```

The length of 'Hello World' is: **11**

Function optional arguments

You don't need to supply all arguments:

- ***number_format*(num)**

```
$number = 3.1415926 ;
echo "Here's a number: " ;
echo number_format($number) ;
```

Here's a number: 3

- ***number_format*(num,dec)**

```
$number = 3.1415926 ;
echo "Here's a number: " ;
echo number_format($number,2) ;
```

Here's a number: 3.14

- ***number_format*(num,dec,dot,mil)**

```
$number = 123456789.1234;
echo "Here's a number: " ;
echo number_format($number,2,","," ") ;
```

Here's a number: 123 456 789,12

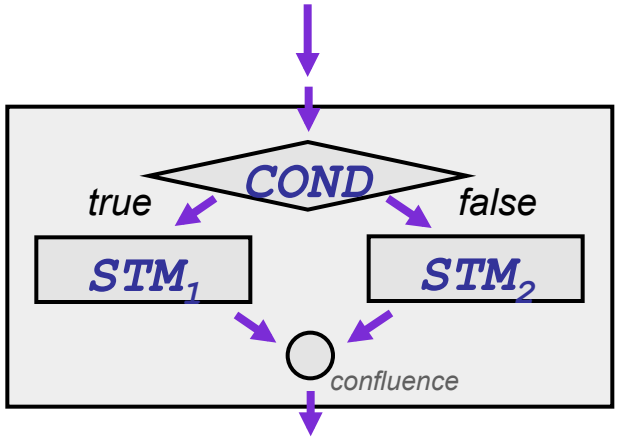
Control Structures

- ***Control Structures*:**
  - ***Statements** (or Expr's)* that ***affect** "**flow of control**":*

- **`if-else`**:

[syntax] `if ( COND ) { STM1 } else { STM2 }`

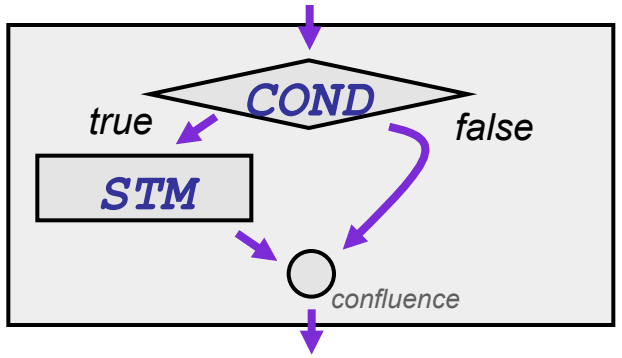
[semantics] *If the condition (`COND`) evaluates to **true**, statement (`STM1`) is executed, otherwise statement (`STM2`) is executed.*



- **`if`**:

[syntax] `if ( COND ) { STM }`

[semantics] *If the condition (`COND`) evaluates to **true**, the given statement (`STM`) is executed, otherwise not.*

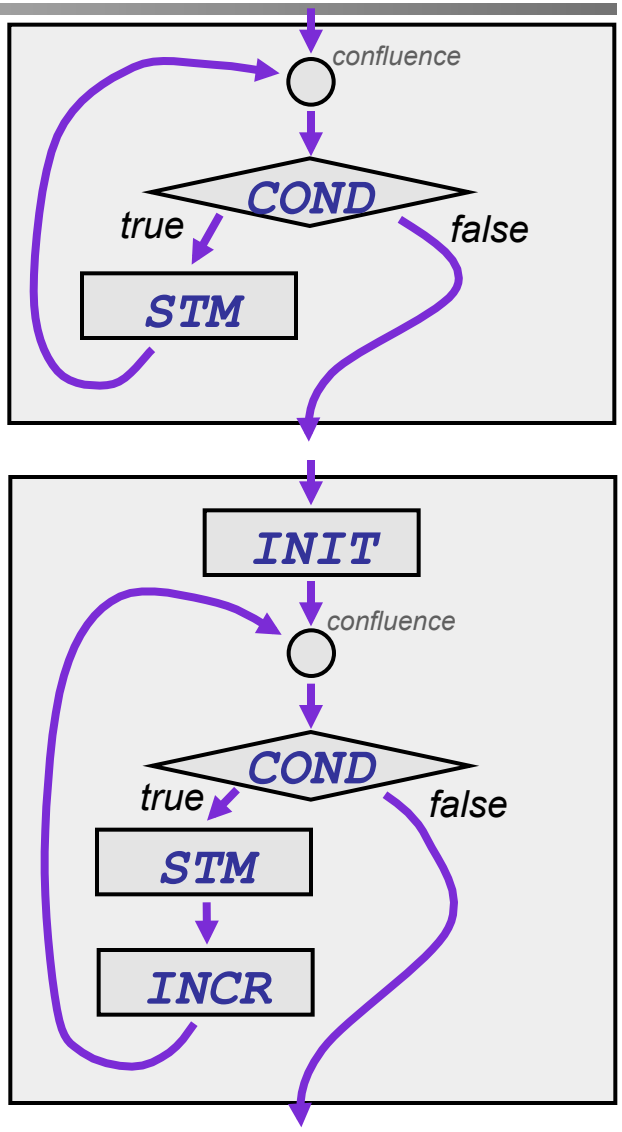


Control Structures (cont'd)

- **`while`**:

[syntax] ■ `while ( COND ) { STM }`

[semantics] ■ *If the condition (`COND`) evaluates to **false**, the given statement (`STM`) is skipped. Otherwise (if the condition was **true**), the statement (`STM`) is executed and afterwards the condition is evaluated again. If it is still **true**, `STM` is executed again... This continues until the condition evaluates to **false**.*



- **`for`**:

[syntax] ■ `for (INIT; COND; INCR) { STM }`

[semantics] ■ *Equivalent to:*

```
{ INIT;
  while ( COND ) { STM  AFTER; }
}
```