introduction to

SCRIPTING, DATABASES, SYSTEM ARCHITECTURE

PHP: input validation & regular expressions

Claus Brabrand

(((brabrand@itu.dk)))

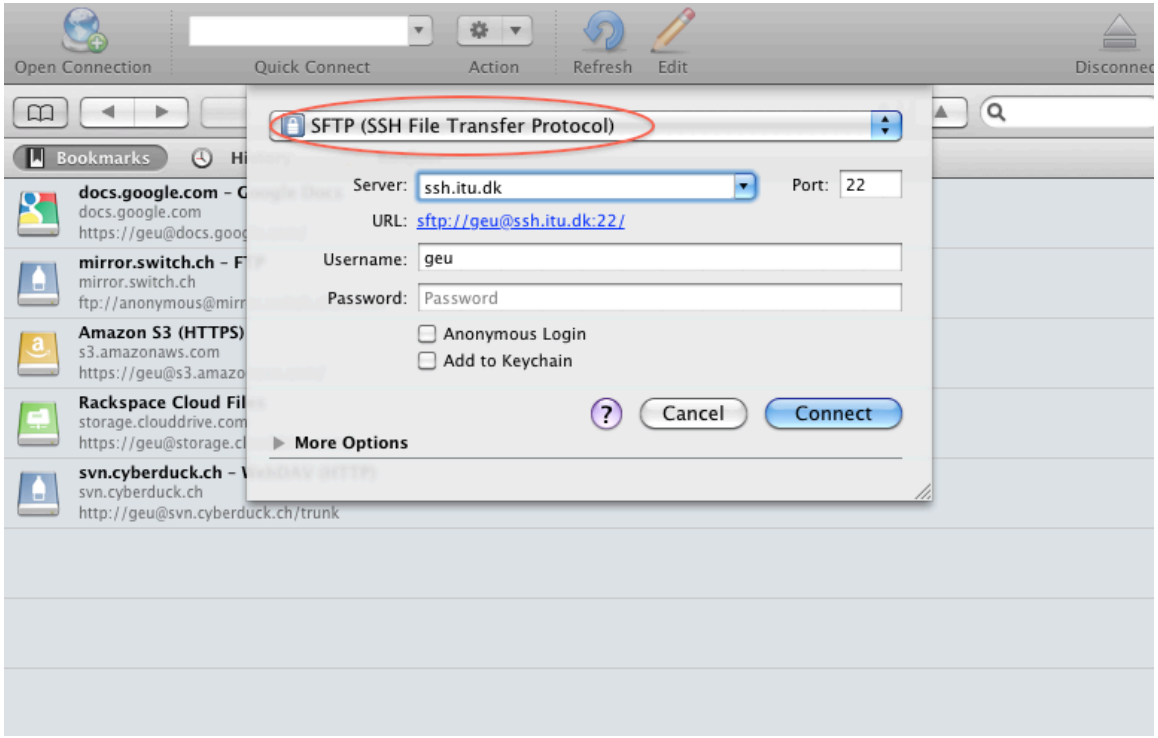
Associate Professor, Ph.D.

(((Programming, Logic, and Semantics)))

IT University of Copenhagen

Message from IT dept.

- Please use **SFTP** instead of **FTP**:



- ...it combines **SSH** (secure protocol) with **FTP** (File Transfer Protocol)

Agenda

- 1) RECAP (arrays & assoc. arrays)
- 2) SINGLE PAGE PHP
- 3) FORM PROCESSING
- 4) INPUT VALIDATION
- 5) REGULAR EXPRESSIONS (regexps)

Array vs Associative Array

- **Array** (maps an ***index/number*** to a value):

```
$things[0] = "Car" ;
$things[1] = "Bike
$things[2] = "House" ;

for ( $i = 0 ; $i < sizeof($things) ; $i++ ) {
   echo "<li> $things[$i] </li>" ;
}
```

- Car
- Bike
- House

i ↓

index	value
0	Car
1	Bike
2	House

- **Assoc. Array** (maps a ***name/text*** to a value):

```
$capitals["Denmark"] = "Copenhagen" ;
$capitals["England"] = "London" ;
$capitals["France"]  = "Paris" ;

foreach ( $capitals as $key => $value) {
   echo "<li> $value, $key </li>" ;
}
```

key	value
Denmark	Copenhagen
England	London
France	Paris

- Copenhagen, Denmark
- London, England
- Paris, France

Alt. Syntax for Assoc. Arrays

- **Array** (maps an ***index/number*** to a value):

```
$things[0] = "Car" ;
$things[1] = "Bike
$things[2] = "House" ;

for ( $i = 0 ; $i < sizeof($things) ; $i++ ) {
   echo "<li> $things[$i] </li>" ;
}
```

- Car
- Bike
- House

index	value
0	Car
1	Bike
2	House

i ↓

- **Assoc. Array** (maps a ***name/text*** to a value):

```
$capitals = array("Denmark" => "Copenhagen",
                  "England" => "London"
                  "France"  => "Paris") ;

foreach ( $capitals as $key => $value) {
   echo "<li> $value, $key </li>" ;
}
```

key	value
Denmark	Copenhagen
England	London
France	Paris

- Copenhagen, Denmark
- London, England
- Paris, France

'foreach' on (normal) Arrays

- **Array** (maps an ***index/number*** to a value):

```
$things[0] = "Car" ;
$things[1] = "Bike
$things[2] = "House" ;

foreach ( $things as $index => $value ) {
   echo "<li>Thing number $index is a $value</li>" ;
}
```

- Thing number 0 is a Car
- Thing number 1 is a Bike
- Thing number 2 is a House

index	value
0	Car
1	Bike
2	House

- **Assoc. Array** (maps a ***name/text*** to a value):

```
$capitals = array("Denmark" => "Copenhagen",
                  "England" => "London"
                  "France"  => "Paris") ;

foreach ( $capitals as $key => $value) {
   echo "<li> $value, $key </li>" ;
}
```

key	value
Denmark	Copenhagen
England	London
France	Paris

- Copenhagen, Denmark
- London, England
- Paris, France

Let's send a bunch of emails

- Example:

```
$emails = array("John Doe" => "john_doe@notmail.com",
                "Jane Doe" => "jane_doe@notmail.com",
                "Barack Obama" => "obama@whitehouse.gov",
                "Helle Thorning" => "helle@folketinget.dk");

foreach ( $emails as $key => $value ) {
    $message = "Dear $key, I have a business opportunity ..." ;
    $message = strtoupper($message) ;
    mail($value, "Great Opportunity!", $message) ;
}
```

key	value
John Doe	john_doe@notmail.com
Jane Doe	jane_doe@notmail.com
Barack Obama	obama@whitehouse.gov
Helle Thorning	helle@folketinget.dk

More Examples...

```php
$unis = array("ITU" => "http://www.itu.dk/en/",
              "KU"  => "http://www.ku.dk/english/",
              "CBS" => "http://www.cbs.dk/en/",
              "AU"  => "http://www.au.dk/en/");

foreach ( $unis as $key => $value ) {
   echo "<li> <a href='$value'>$key</a> </li>" ;
}
```

- ITU
- KU
- CBS
- AU

```php
$menu = array("Home" => "index.php",
              "About us" => "about.php",
              "List of products" => "prod.php",
              "Contact us" => "contact.php");

foreach ( $menu as $key => $value ) {
   echo "<a href='$value'>$key</a> <br/>" ;
}
```

Home
About us
List of products
Contact us

Array errors?

- Let's try to abuse the arrays to see what happens (and if we can get an error):

```php
<?php

      $a[10] = 42;       // set value at index 10
      $size = sizeof($a) ;
      echo "size is: ($size)." ;
      echo "<p/>" ;
      $value = $a[20] ; // read value at index 20 (which wasn't set)!
      echo "value is: ($value)." ;
      echo "<p/>" ;
      var_dump($a);

?>
```

```
size is: (1).

value is: ().

array(1) { [10] => int(42) }
```

- **Note: no errors!**
  - **à la Assoc. Array: we associated *key* 10 with *value* 42**

Agenda

- 1) RECAP (arrays & assoc. arrays)
- 2) SINGLE PAGE PHP
- 3) FORM PROCESSING
- 4) INPUT VALIDATION
- 5) REGULAR EXPRESSIONS (regexps)

Form Submission (GET)

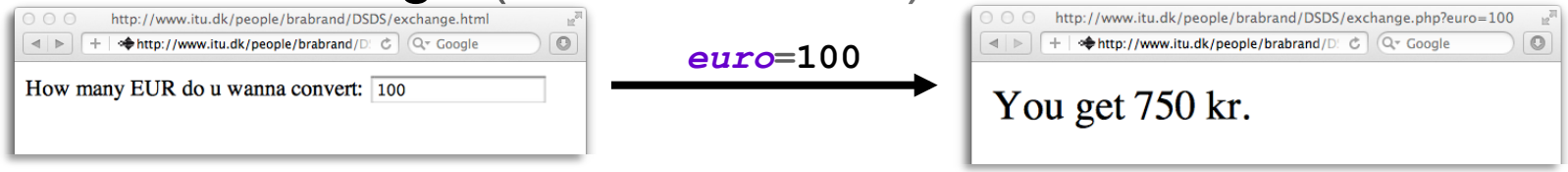
```
<html>
  <body>
    <form action="exchange.php">
      How many EUR do u wanna convert:
      <input type="text" name="euro"/>
    </form>
  </body>
</html>
```

exchange.html

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

exchange.php

- Conventional usage (HTML → PHP):



- Enter URL directly in browser:

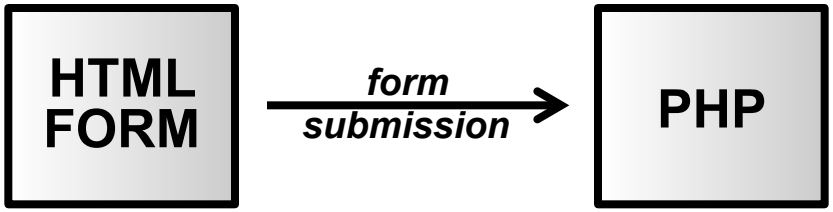
```
http://www.itu.dk/people/brabrand/
DSDS/exchange.php?euro=100
```

- Make direct link in HTML:

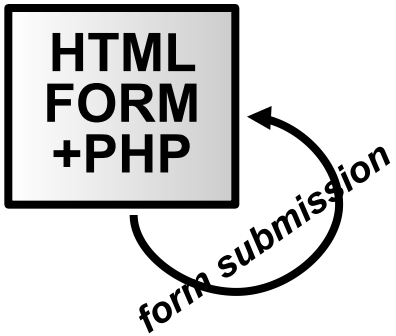
```
<a href=
  "http://www.itu.dk/people/brabrand/
   DSDS/exchange.php?euro=100">
click here</a>
```

Single Page PHP

- So far, we have used two files:



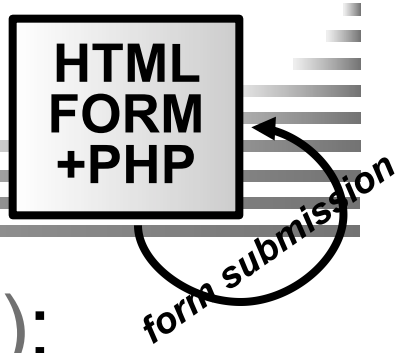
- We can also use ***one single PHP*** file (the same file to ***display and handle*** form):




```
<html><body>
    <form action="">
        ...display HTML form...
    </form>
    <?php
        if ( ...form has been submitted... ) {
            ...process data and display result...
        }
    ?>
</body></html>
```

will submit to ***itself!***

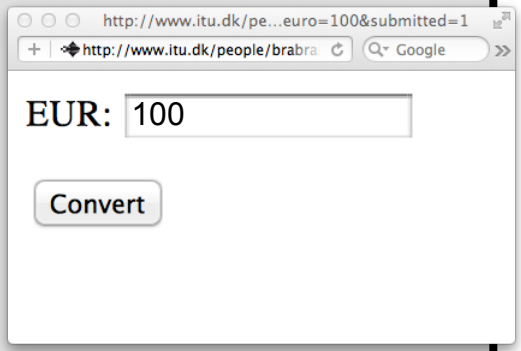
Single Page PHP



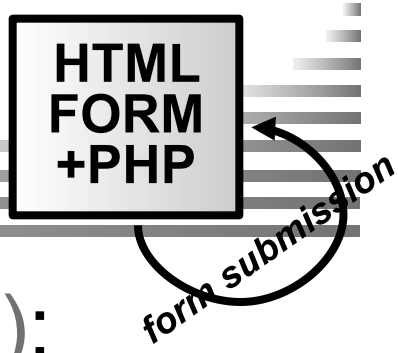
- Single Page PHP (not: HTML→ PHP):

```
<html><body>
    <form action="">
        <input type="text" name="euro" />
        <p />
        <input type="submit" value="Convert" />
    </form>

    <?php
        if ( isset($_REQUEST['euro']) ) { // if form was submitted
            $eur = $_REQUEST['euro'] ;              // then process
            $dkk = $eur * 7.5 ;                     // data submitted
            echo "You get $dkk kr. for $eur EUR" ;  // and display!
        }
    ?>
</body></html>
```



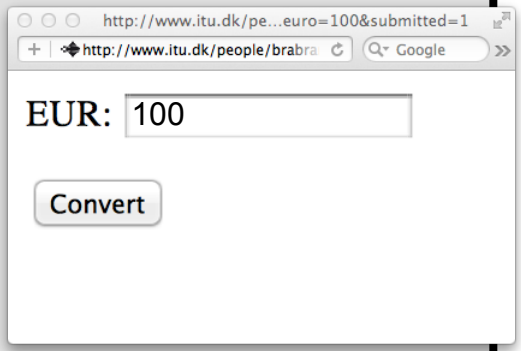
Single Page PHP



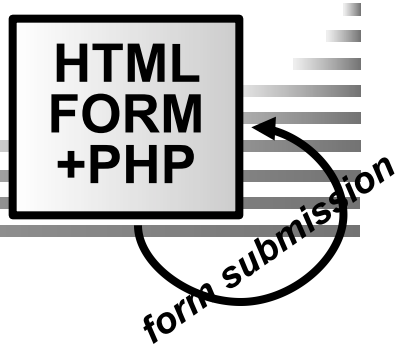
- Single Page PHP (not: HTML→ PHP):

```
<html><body>
    <form action="">
        <input type="text" name="euro" />
        <p />
        <input type="submit" value="Convert" />
    </form>

    <?php
        if ( isset($_REQUEST['euro']) ) { // if form was submitted
            $eur = $_REQUEST['euro'] ;                // then process
            $dkk = $eur * 7.5 ;                       // data submitted
            echo "You get $dkk kr. for $eur EUR" ;    // and display!
        }
    ?>
</body></html>
```



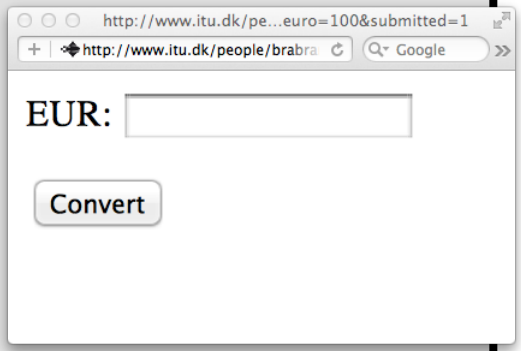
Single Page PHP



- Improved version: "Sticky Form":

```
<html><body>
    <form action="">
        <input type="text" name="euro"
 value="<?php echo $_REQUEST['euro']; ?>" /> <p />
        <input type="submit" value="Convert" />
    </form>

    <?php
        if ( isset($_REQUEST['euro']) ) { // if form was submitted
            $eur = $_REQUEST['euro'] ;              // then process
            $dkk = $eur * 7.5 ;                     // data submitted
            echo "You get $dkk kr. for $eur EUR" ;  // and display!
        }
    ?>
</body></html>
```



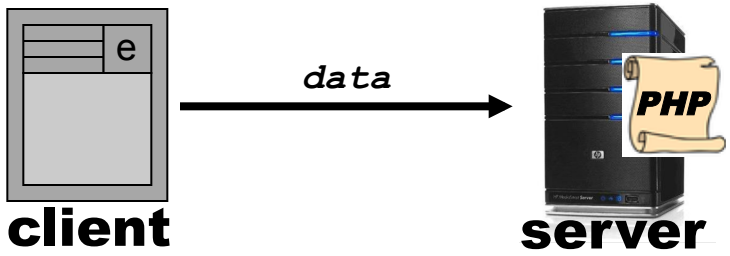
Agenda

- 1) RECAP (arrays & assoc. arrays)
- 2) SINGLE PAGE PHP
- 3) FORM PROCESSING
- 4) INPUT VALIDATION
- 5) REGULAR EXPRESSIONS (regexps)

Processing text input fields

```
Enter a text:
<br/>
<input type="text" name="val"/>
```

```
<?php
    $value = $_REQUEST['val'] ;
    echo "You entered: <br/>" ;
    echo "<b>$value</b>" ;
?>
```



Enter a text:

Hello world!

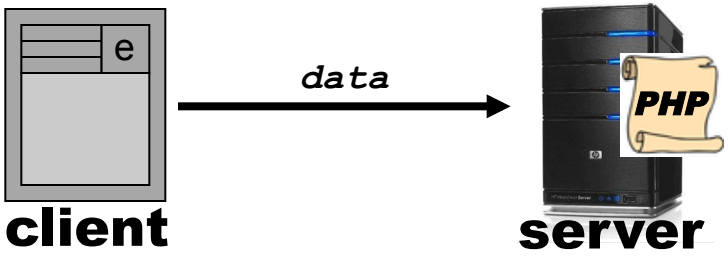
val=Hello+world%22

You entered:
Hello world!

Processing password input fields

```
Enter a text:
<br/>
<input type="password" name="val"/>
```

```
<?php
    $value = $_REQUEST['val'] ;
    echo "You entered: <br/>" ;
    echo "<b>$value</b>" ;
?>
```



Enter a text:

●●●●●●●●●●●●

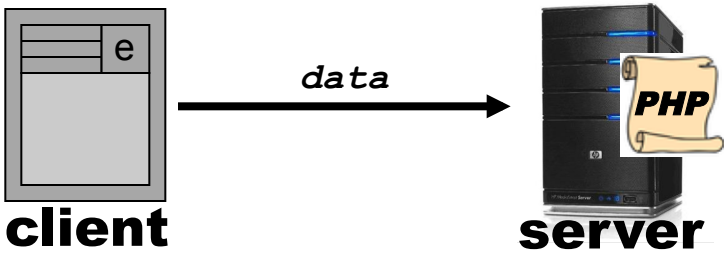
val=Hello+world%22

You entered:
Hello world!

Processing textarea input fields

```
Enter a text:
<br/>
<textarea name="val">...</textarea>
```

```
<?php
    $value = $_REQUEST['val'] ;
    echo "You entered: <br/>" ;
    echo "<b>$value</b>" ;
?>
```



Enter a text:

```
Once upon a time,
there lived a great
King with his ...
```

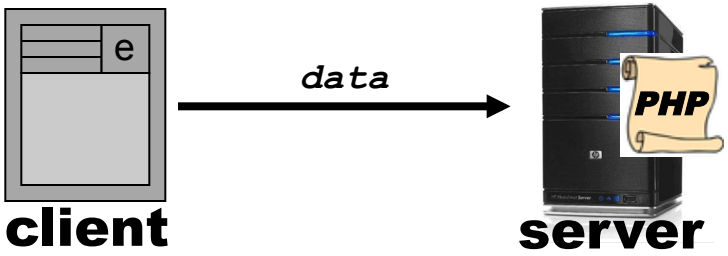
val=Once+upon+a...

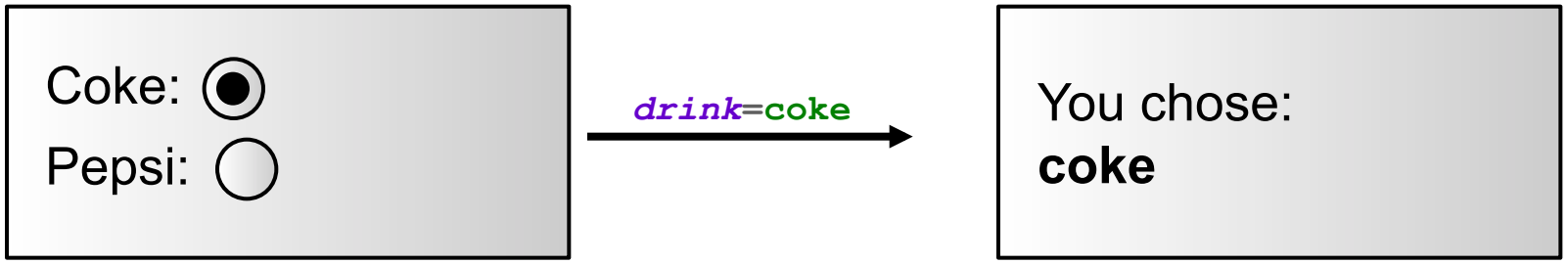
You entered:
Once upon a time, there lived a great King with his ...

Processing radio buttons

```
Coke:
<input type="radio" name="drink"
       value="coke"/>
<br/> Pepsi:
<input type="radio" name="drink"
       value="pepsi"/>
```

```
<?php
    $value = $_REQUEST['drink'] ;
    echo "You chose: <br/>" ;
    echo "<b>$value</b>" ;
?>
```

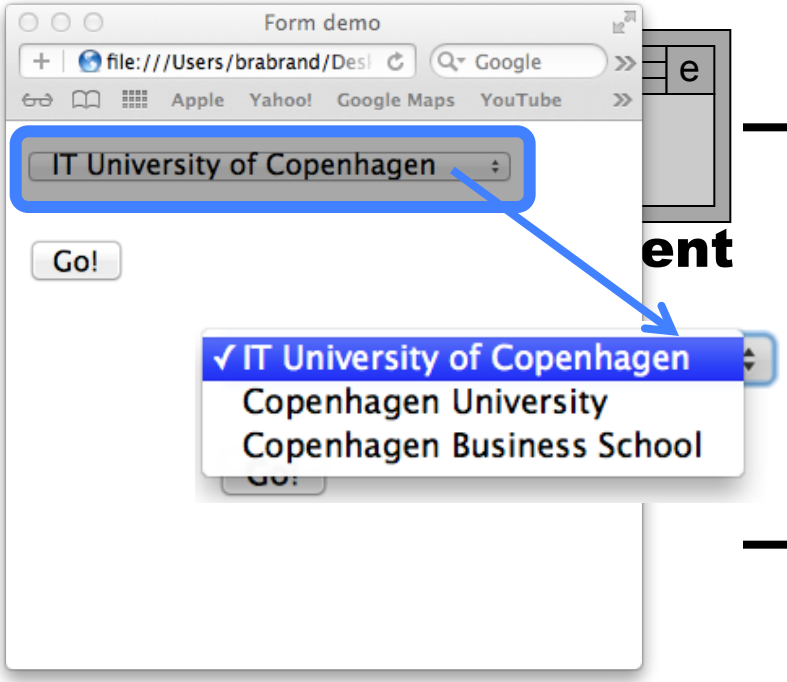


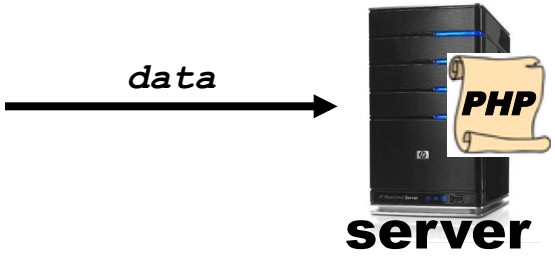


Processing select input fields

```
<select name="uni">
  <option value="ITU">IT Universi..</>
  <option value="KU">Copenhagen U..</>
  <option value="CBS">Copenhagen B.</>
</select>
```

```
<?php
    $value = $_REQUEST['uni'] ;
    echo "You chose: <br/>" ;
    echo "<b>$value</b>" ;
?>
```





uni=ITU

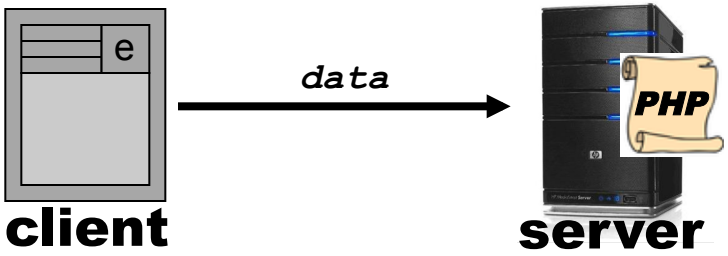
You chose:
ITU

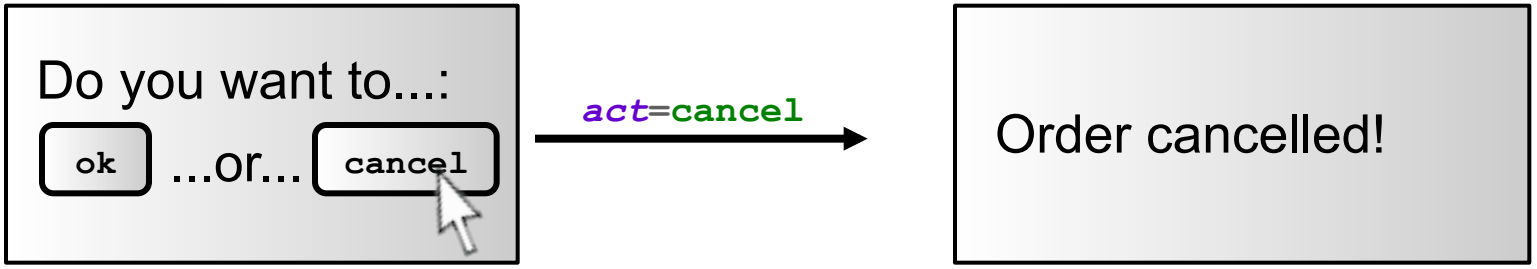
Note: a **select** is essentially the same as a **radio button** group !

Processing submit buttons

```
Do you want to...:
<input type="submit" name="act"
       value="ok"/>
...or...
<input type="submit" name="act"
       value="cancel"/>
```

```
<?php  $value = $_REQUEST['act'] ;
    if (value == "ok") {
        echo "Order confirmed!" ;
    } else {
        echo "Order cancelled!" ;
    }                                  ?>
```

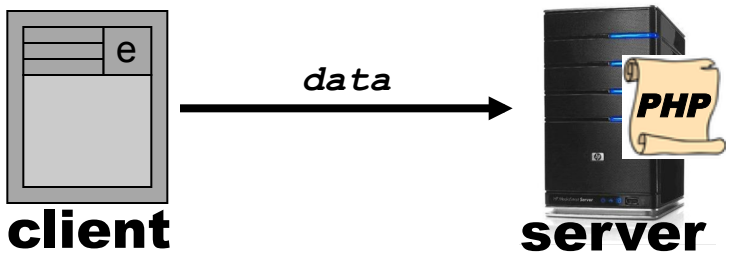




Processing checkbox buttons

```
Cheese:
<input type="checkbox" name="ingr[]"
       value="cheese"/>
... Onion:
<input type="checkbox" name="ingr[]"
       value="onion"/>
```

```
<?php  $a = $_REQUEST['ingr'] ;
       // 'a' is an array! :-)
  echo "You selected:" ;
  for ($i=0; $i<sizeof($a); $i++) {
    echo "<li> $a[$i] </li>" ;
  }                                  ?>
```



Cheese: [X]

Tomato: [X]

Onion: [ ]

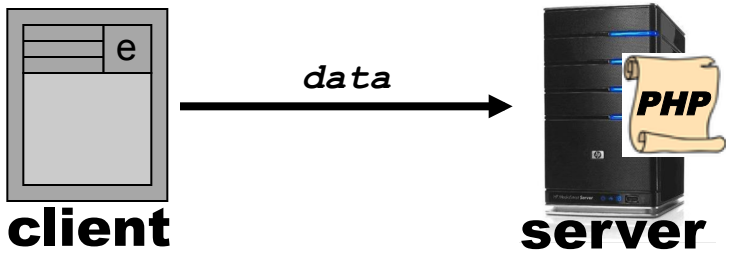
ingr[]=cheese&*ingr[]*=tomato

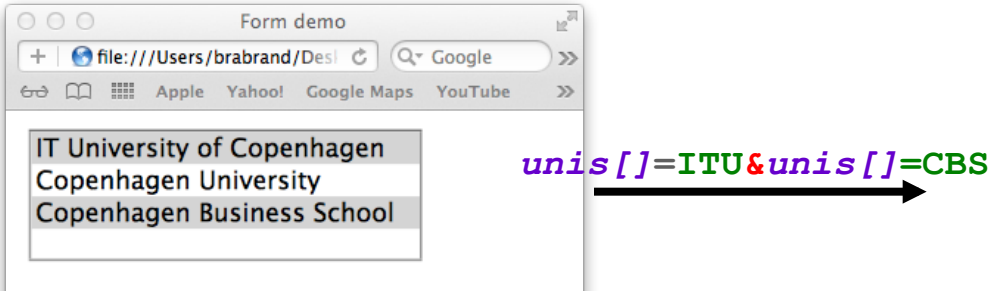
You selected:
- cheese
- tomato

Processing select-multiple

```
<select name="unis[]"
        multiple="multiple">
  <option value="ITU">IT Universi..</>
  <option value="KU">Copenhagen U..</>
  <option value="CBS">Copenhagen B.</>
</select>
```

```
<?php  $a = $_REQUEST['unis'] ;
       // 'a' is an array! :-)
  echo "You selected:" ;
  for ($i=0; $i<sizeof($a); $i++) {
    echo "<li> $a[$i] </li>" ;
  }                                  ?>
```





unis[]=ITU&unis[]=CBS

You selected:
- ITU
- CBS

Note: a **select-multiple** is essentially the same as a **checkbox** group **!**

Agenda

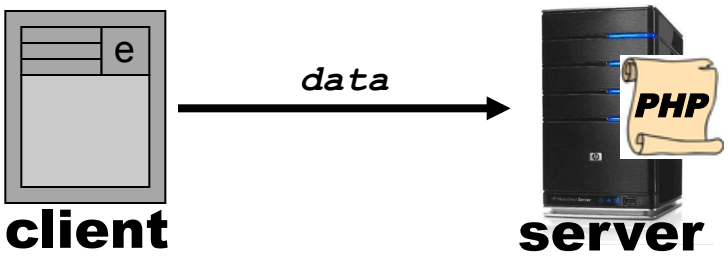
- 1) RECAP (arrays & assoc. arrays)
- 2) SINGLE PAGE PHP
- 3) FORM PROCESSING
- 4) INPUT VALIDATION
- 5) REGULAR EXPRESSIONS (regexps)

Input Validation Example

```
<!-- currency conversion example -->

How many euros:
<br/>
<input type="text" name="euro"/>
```

```
<?php
    $eur = $_REQUEST['euro'] ;
    $dkk = $eur * 7.5 ;
    echo "You get $dkk kr.
          for $eur euros." ;     ?>
```



How many euros:

grwlkwkjl

euro=grwlkwkjl

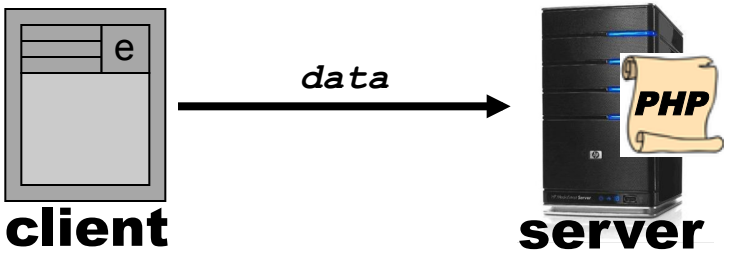
You get 0 kr. for grwlkwkjl euros.

Input Validation Example

```
<!-- currency conversion example -->

How many euros:
<br/>
<input type="text" name="euro"/>
```

```
<?php
    $eur = $_REQUEST['euro'] ;
    $dkk = $eur * 7.5 ;
    echo "You get $dkk kr.
          for $eur euros." ;     ?>
```



user-injected content

How many euros:

```
<h1>:-)</h1>
```

euro=...

You get 0 kr. for

:-)

euros.

Note: this ***lack of validation*** can be exploited for ***cross site scripting*** (XXS) attacks!

Solution: Validate Input

```
<!-- currency conversion example -->

How many euros:
<br/>
<input type="text" name="euro"/>
```

```
<?php
    $eur = $_REQUEST['euro'] ;
    if (...check $eur valid...) {
        $dkk = $eur * 7.5 ;
        echo "You get $dkk kr. " ;
        echo "for $eur euros." ;
    } else {
        // give suitable error msg
        // and ask user to input
        // data again.
    }
?>
```

How many euros:

grwlkwkjl

euro=grwlkwkjl

You bastard!, that was not a f***in' number. Enter again!
How many euros:

Note: ***validation*** means that ***cross site scripting*** attacks are no longer possible!

Many different input formats

numbers	
two digit numbers	09
decimal numbers	3.1415926
zip codes	2300
phone numbers	72185000
ISBN numbers	978-1-4503-0665-2
CPR numbers	141011-9999
names of people	John X. Doe
legal dates	14-10-2011
finite set of choices	'coke' or 'pepsi' 'cheese', 'tomato', or 'onion' 'jan', 'feb', .., or 'dec'
email addresses	john_doe@notmail.com
web URLs	http://www.itu.dk/da/Presse
...	

What bad stuff can happen?

- Various errors can occur when invalid data finds its way into the PHP script:
  - ***Cosmetic errors:*** the PHP script doesn't create well-formed HTML that the browser doesn't render well.
  - ***Semantic errors:*** the PHP script processes invalid data and produces wrong results.
  - ***Security threats:*** hackers can exploit lack of input validation to delete or modify data in the service.

- By ***validating*** the client input data, we can avoid a lot of errors and security threats.

Agenda

- 1) RECAP (arrays & assoc. arrays)
- 2) SINGLE PAGE PHP
- 3) FORM PROCESSING
- 4) INPUT VALIDATION
- 5) REGULAR EXPRESSIONS (regexps)


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Input Validation Example

```
How many euros:
<br/>
<input type="text"
       name="euro"/>
```

```
<?php
    $eur = $ REQUEST['euro'] ;
    if ( preg_match('/^[0-9]+$/', $eur) ) {
        $dkk = $eur * 7.5 ;
        echo "You get $dkk kr. " ;
        echo "for $eur euros." ;
    } else {
        // give suitable error msg
        // and ask user to input
        // data again.
    }
?>
```

How many euros:

grwlkwkjl

euro=grwlkwkjl →

That wasn't a valid number.
Please enter data again:
How many euros:

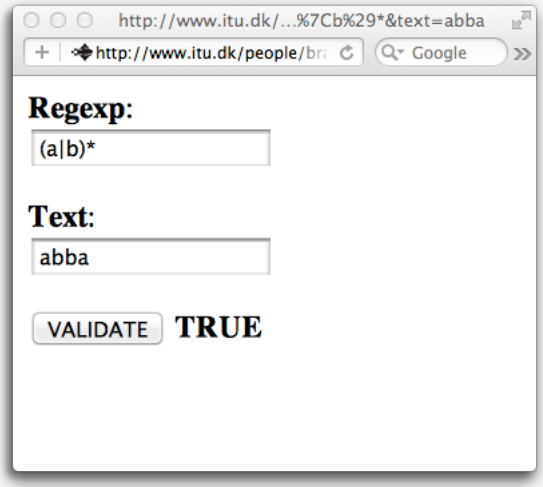
Note: ***validation*** means that ***cross site scripting*** attacks are no longer possible!

```
<html>
  <body>
     <?php
        $regexp = $_REQUEST['regexp'] ;
        $text = $_REQUEST['text'] ;
     ?>

    <form action="">
      <b>Regexp</b>:<br/>
      <input type="text" name="regexp"/ value="<?php echo $regexp; ?>">
      <p/>
      <b>Text</b>:<br/>
      <input type="text" name="text" value="<?php echo $text; ?>">
      <p/>
      <input type="submit" value="VALIDATE"/>

      <?php
         if ( isset($regexp) ) { // values submitted
           // validate!
           $pattern = "/^" . $regexp . "$/" ;
           if ( preg_match($pattern, $text) ) {
             echo "<b>TRUE</b>" ;
           } else {
             echo "<b>FALSE</b>" ;
           }
         }
      ?>
    </form>
  </body>
</html>
```

Simple Regexp Validator:



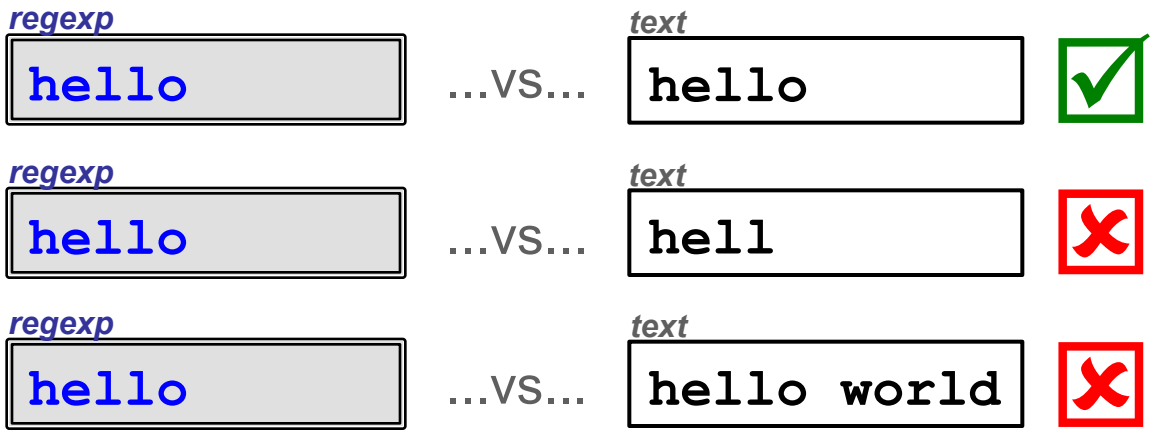
Regexp Tutorial

step by step

mixed with mini exercises

Constant Text

- ***Constant text:***
  - matches only itself (i.e., 'hello' matches only 'hello')

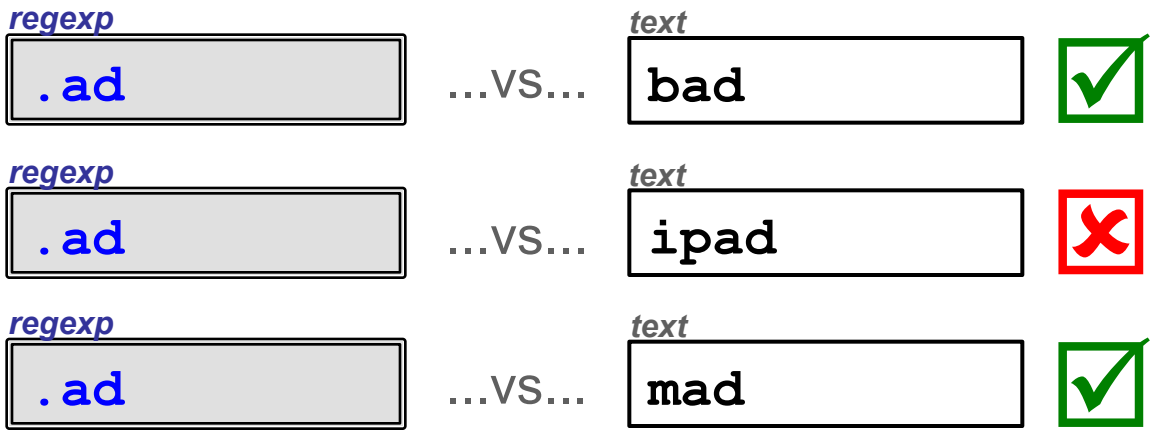


PHP

```
preg_match('/^hello$/', "hello")        == true
preg_match('/^hello$/', "hell")         == false
preg_match('/^hello$/', "hello world")  == false
```

Wild Card: '.' (dot)

- ***Wild card:*** '.' (dot)
  - matches *any* one character (except newline '\n')

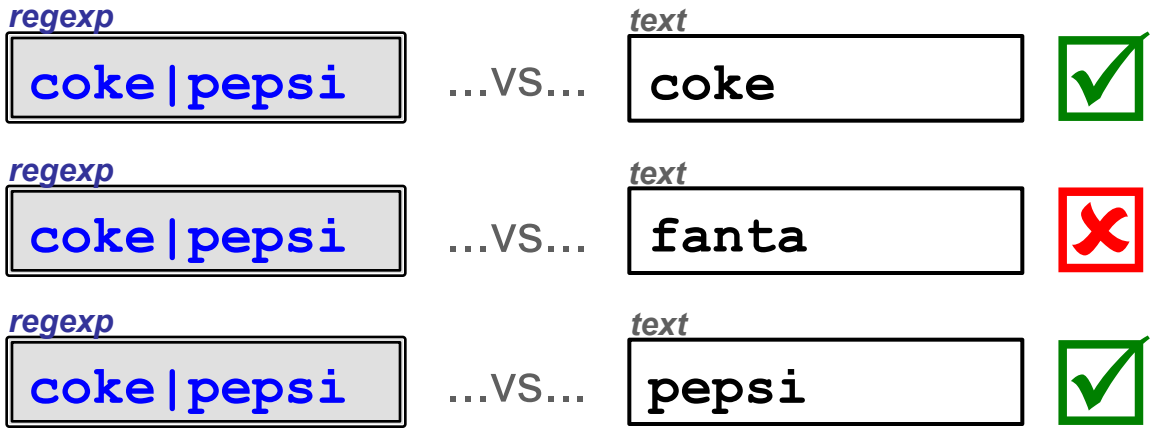


PHP

- `preg_match('/^.ad$/', "bad")` == *true*
- `preg_match('/^.ad$/', "ipad")` == *false*
- `preg_match('/^.ad$/', "mad")` == *true*

Alternative: '|' (vertical bar)

- ***Alternative:*** '|' (vertical bar)
  - regexp '`X|Y`' matches either regexp '`X`' or regexp '`Y`'

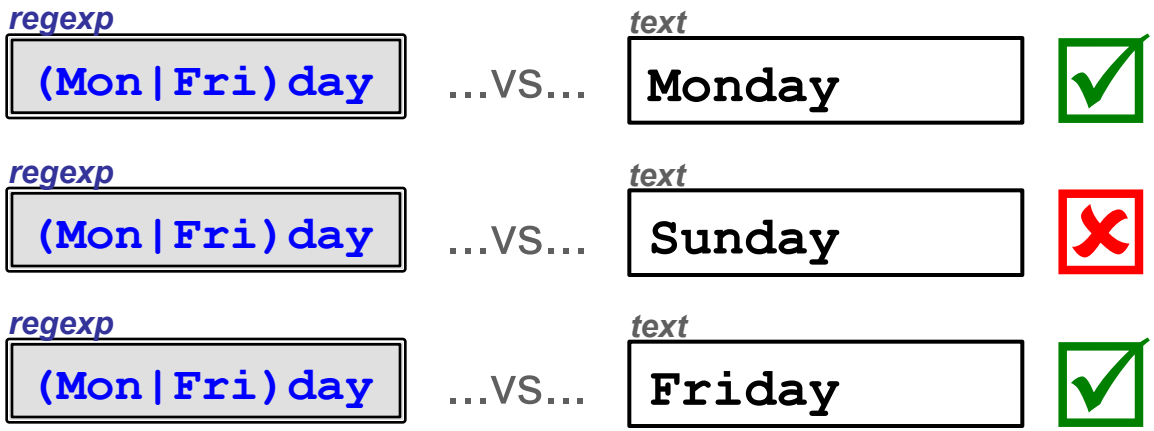


PHP

- `preg_match('/^coke|pepsi$/', "coke")` == *true*
- `preg_match('/^coke|pepsi$/', "fanta")` == *false*
- `preg_match('/^coke|pepsi$/', "pepsi")` == *true*

Grouping: '(...)' (parentheses)

- ***Grouping:*** '(...)' (parentheses)
  - You can use parentheses for grouping regexps

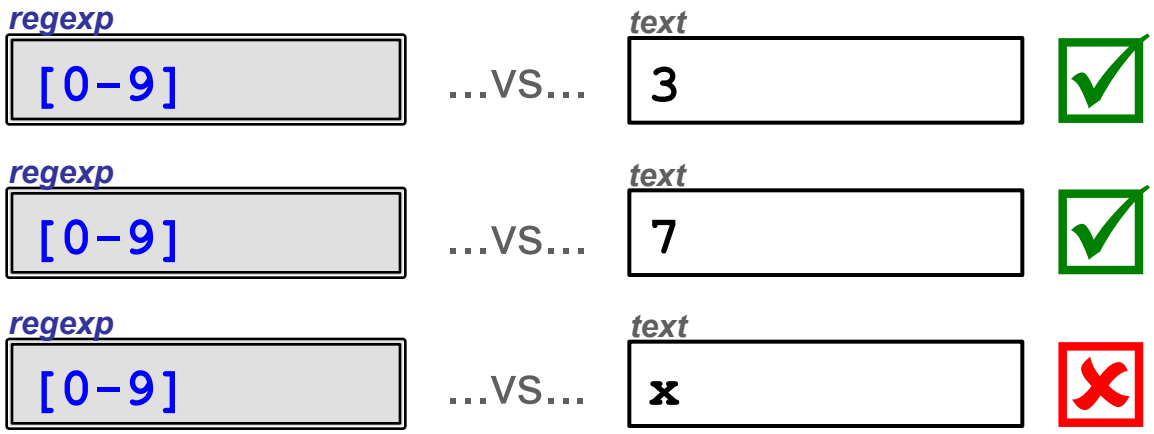


PHP

- `preg_match('/^(Mon|Fri)day$/', "Monday") == true`
- `preg_match('/^(Mon|Fri)day$/', "Sunday") == false`
- `preg_match('/^(Mon|Fri)day$/', "Friday") == true`

Character Class: '[0-9]'

- ***Character class:*** '[0-9]' (interval)
  - matches any one character in the interval given

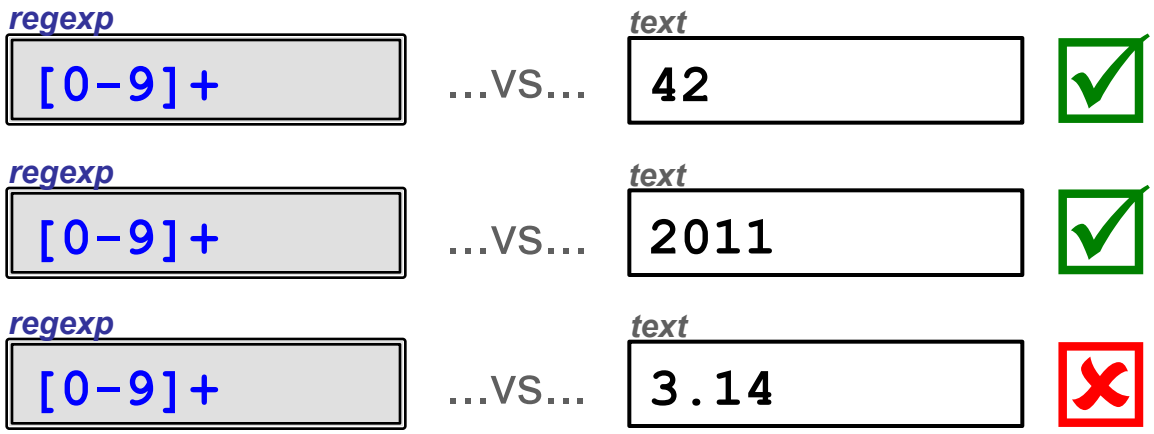


PHP

- `preg_match('/^[0-9]$/', "3")` == *true*
- `preg_match('/^[0-9]$/', "7")` == *true*
- `preg_match('/^[0-9]$/', "x")` == *false*

One-or-More: '[0-9]+'

- *One-or-More:* '[0-9]+'
  - regexp '**R+**' matches *one-or-more* times regexp '**R**'

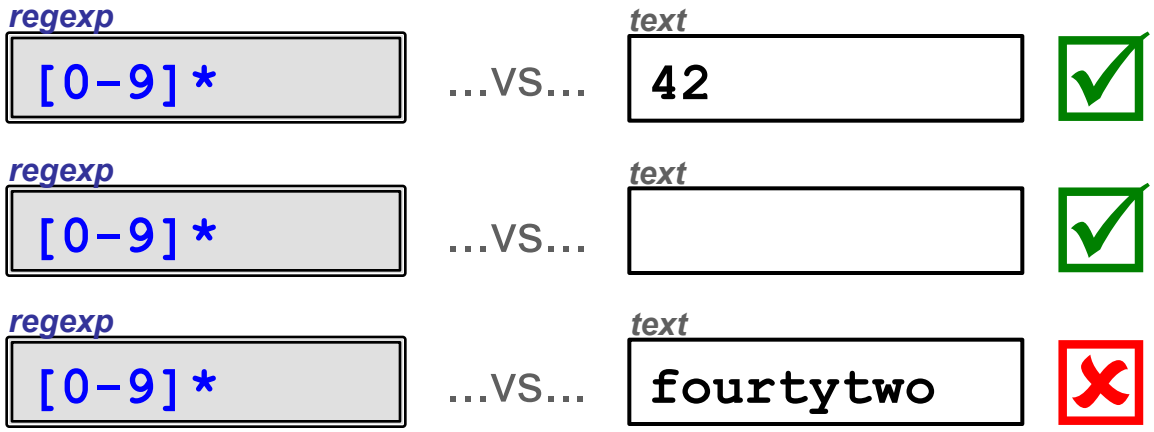


PHP

- `preg_match('/^[0-9]+$/', "42")` == *true*
- `preg_match('/^[0-9]+$/', "2011")` == *true*
- `preg_match('/^[0-9]+$/', "3.14")` == *false*

Zero-or-More: '[0-9]*'

- ***Zero-or-More:*** **'[0-9]*'**
  - regexp '**R***' matches *zero-or-more* times regexp '**R**'



PHP

- `preg_match('/^[0-9]*$/', "42")` == *true*
- `preg_match('/^[0-9]*$/', "")` == *true*
- `preg_match('/^[0-9]*$/', "fourtytwo")` == *false*

Repetitions: '[0-9]{4}'

- `x{4}` exactly four times
- `x{4,}` four-or-more
- `x{2,5}` two to five times

- ***Zero-or-More:*** '[0-9]{4}'
  - regexp '`R{4}`' matches regexp '`R`' four times

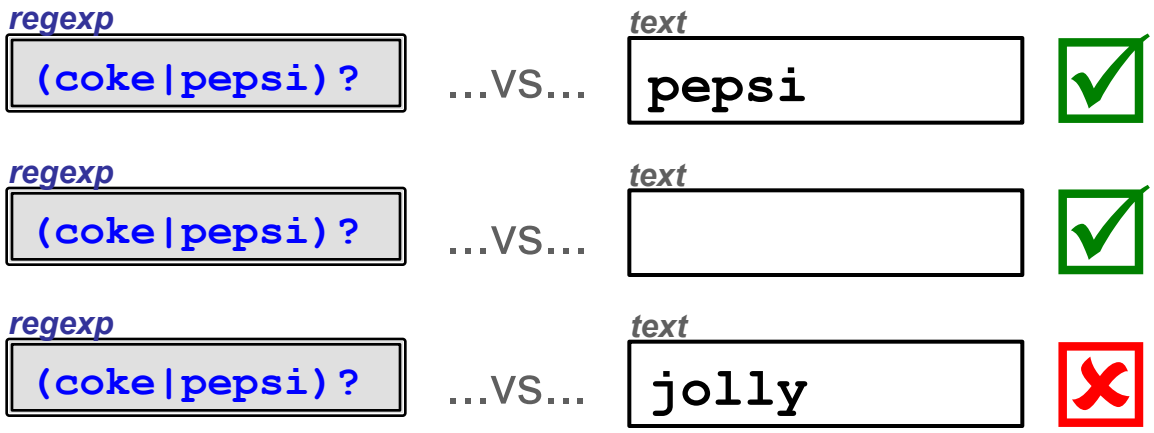
regexp		*text*	
`[0-9]{4}`	...vs...	2300	✓
`[0-9]{4}`	...vs...	123	✗
`[0-9]{4}`	...vs...	12345	✗

PHP

- `preg_match('/^[0-9]{4}$/', "2300")` == *true*
- `preg_match('/^[0-9]{4}$/', "123")` == *false*
- `preg_match('/^[0-9]{4}$/', "12345")` == *false*

Optional: '(coke|pepsi)?'

- ***Optional:*** '(coke|pepsi)?'
  - regexp 'R?' matches regexp 'R' or nothing (empty text)

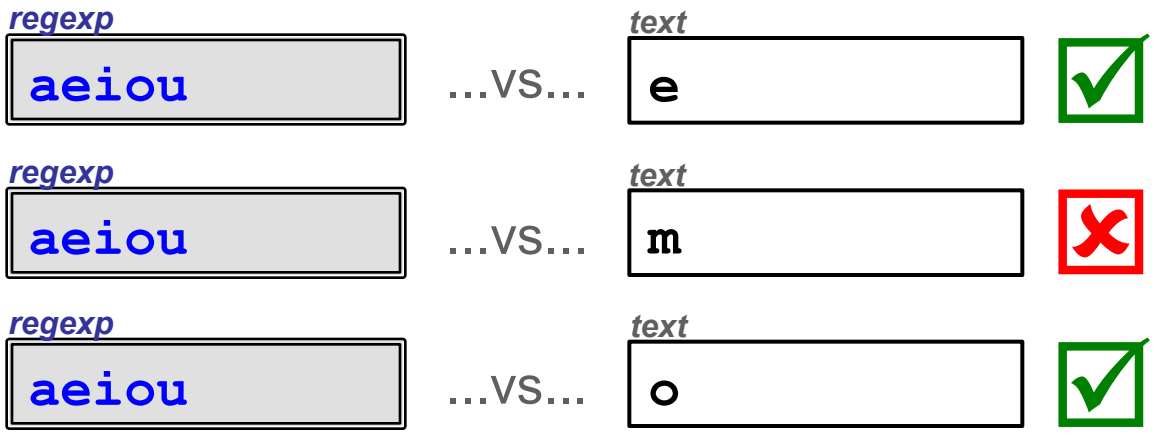


PHP

- `preg_match('/^(coke|pepsi)?$/', "pepsi") == true`
- `preg_match('/^(coke|pepsi)?$/', "")      == true`
- `preg_match('/^(coke|pepsi)?$/', "jolly") == false`

Character Class: '[aeiou]'

- ***Character class:*** `'[aeiou]'`
  - matches any one of the given characters

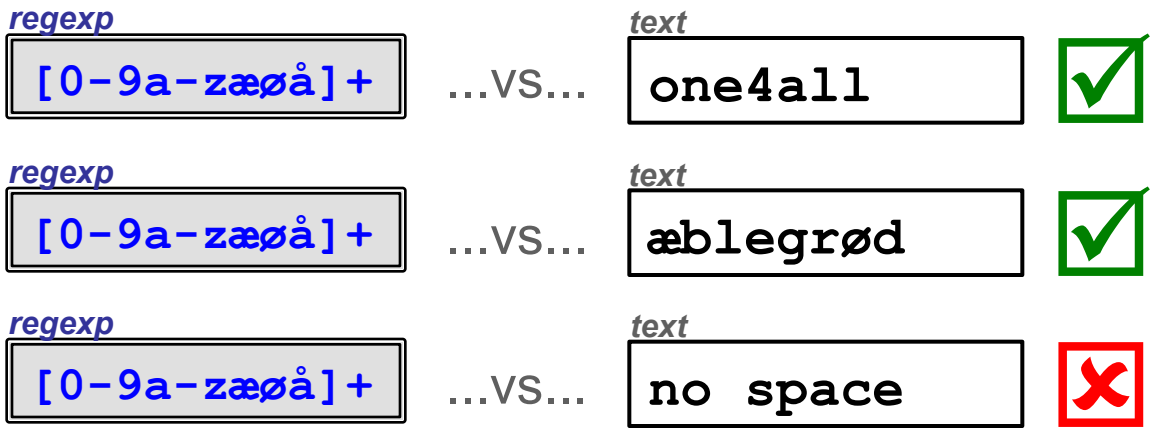


PHP

- `preg_match('/^[aeiou]$/', "e")` == *true*
- `preg_match('/^[aeiou]$/', "m")` == *false*
- `preg_match('/^[aeiou]$/', "o")` == *true*

Character Class: '[0-9a-zæøå]+'

- ***Character class:*** '[0-9a-zæøå]+' (intervals)
  - matches any one character in the intervals given

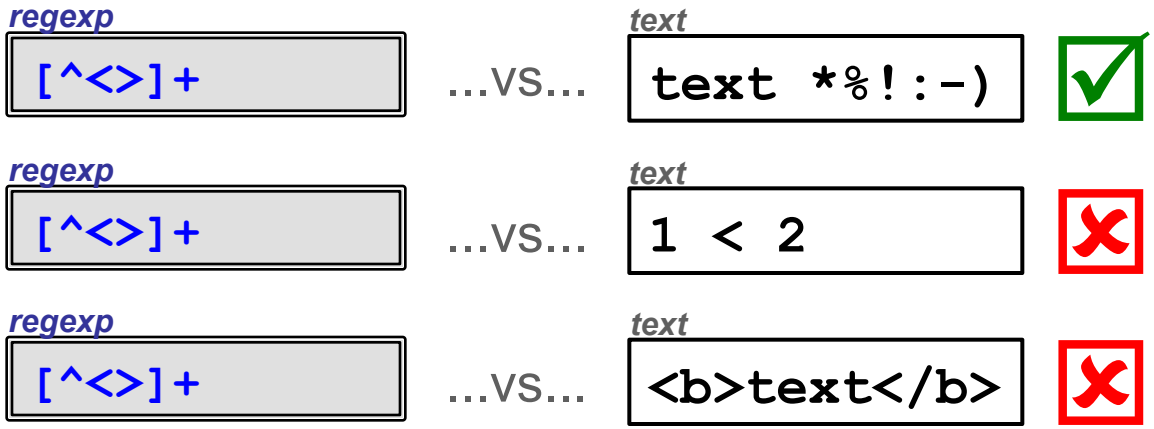


PHP

- `preg_match('/^[0-9a-zæøå]+$/', "one4all") == true`
- `preg_match('/^[0-9a-zæøå]+$/', "æblegrød")== true`
- `preg_match('/^[0-9a-zæøå]+$/', "no space")== false`

Negated Char Class: '[^<>]+'

- ***Negated character class:*** `'[^<>]+'`
  - matches any character that is ***NOT*** one of the given

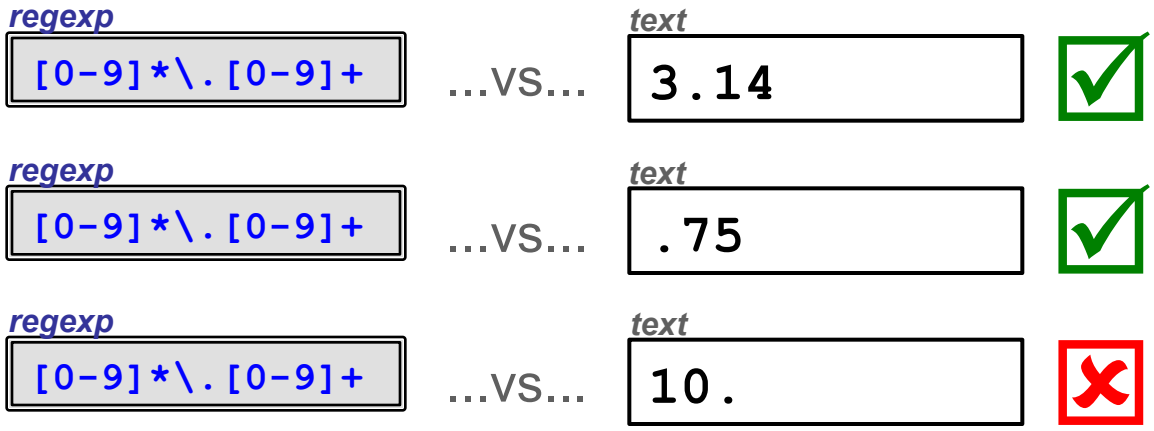


PHP

- `preg_match('/^[^<>]+$/', "text *%!:-)")` == *true*
- `preg_match('/^[^<>]+$/', "1 < 2")` == *false*
- `preg_match('/^[^<>]+$/', "<b>text</b>")` == *false*

Escaping: '[0-9]*\.[0-9]+)'

- ***Character escaping:*** '[0-9]*\.[0-9]+'
  - the regexp '\c' matches the character 'c'



PHP

- `preg_match('/^[0-9]*\.[0-9]+$/', "3.14") == true`
- `preg_match('/^[0-9]*\.[0-9]+$/', ".75") == false`
- `preg_match('/^[0-9]*\.[0-9]+$/', "10.") == false`

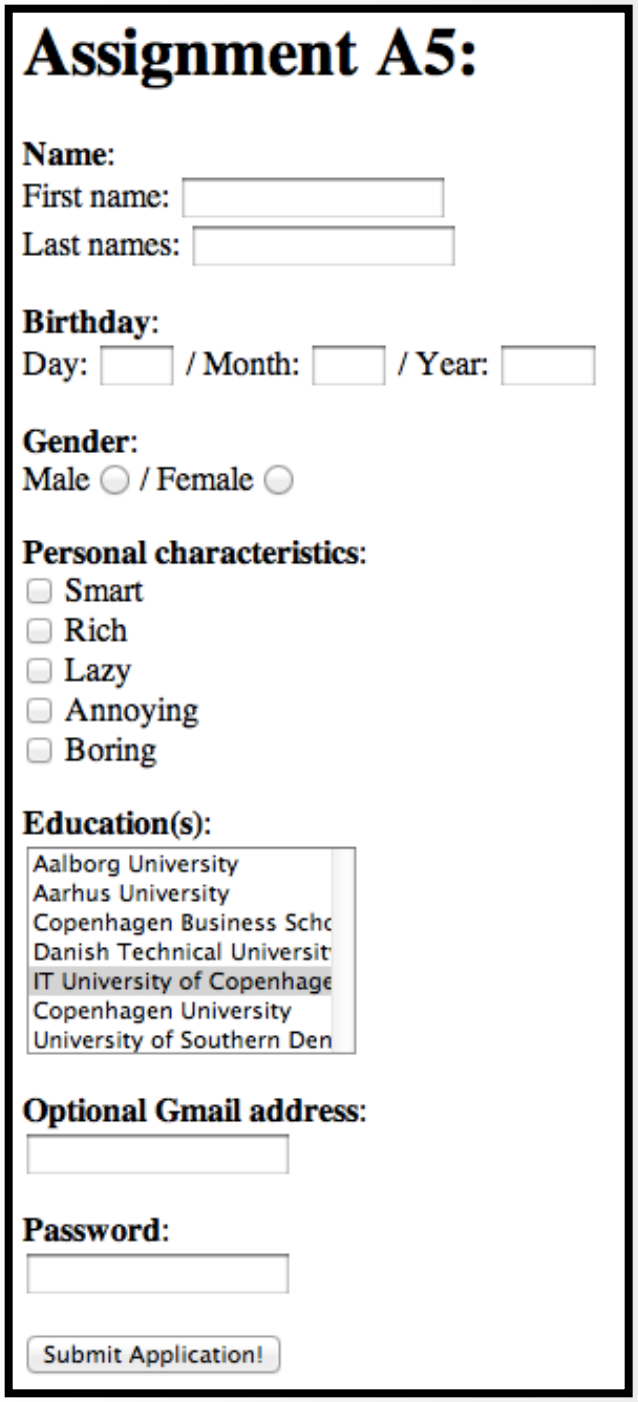
For specifying characters that have a regexp meaning: '*', '+', '?', '|', '[', ']', ...

Regular Expressions (regexp)

```
R   :   x              // matches text 'x' itself
    :   \c             // escaping (matches char 'c')
    :   '.' (dot)      // matches one character
    :   [aeiou]        // character class: one of chars
    :   [a-z]          // any character in interval
    :   [^...]         // negated char class
    :   R1 R2          // sequence (matches R1 then R2)
    :   R1|R2          // alternative (either R1 or R2)
    :   R+             // one-or-more repetitions of R
    :   R*             // zero-or-more repetitions of R
    :   R?             // optional R (empty text or R)
    :   R{n}           // n times repetitions of R
    :   R{n,}          // at least n repetitions of R
    :   R{n,m}         // n to m repetitions of R
    :  ( R )           // parentheses for grouping
```

Regexp (other uses)

- **Client-side Validation**:
  - Also used for validation in the browser (via JavaScript)
- **Web Scraping**:
  - Also used for extracting data from other external web services (e.g., currency rates, weather forecasts, ...)
- **Searching data**:
  - Also used for searching for certain formats in large data files (e.g., extract all email addresses in a file)
- **Search and Replace**:
  - (As above, except it also replaces with something)



--- ASSIGNMENT 5 ---

1) program a PHP script that reads the form input and simply writes it out again.

2) add input validation (see next slide for requirements on allowed input).

3) **OPTIONAL:** make the form submit to itself and reinsert what the client entered and selected in the input fields (so that input data isn't lost if the page didn't validate).

Assignment A5:

Name:
First name:
Last names:

Birthday:
Day: / Month: / Year:

Gender:
Male / Female

Personal characteristics:
- Smart
- Rich
- Lazy
- Annoying
- Boring

Education(s):
- Aalborg University
- Aarhus University
- Copenhagen Business Schc
- Danish Technical Universit
- IT University of Copenhage
- Copenhagen University
- University of Southern Den

Optional Gmail address:

Password:

Submit Application!

--- ASSIGNMENT 5 ---

Name: the first name should be a name (one of more alphanumeric characters). The last names should be one or more names (separated by one space character). The last name may also contain initials (e.g. 'R. R. Tolkien').

Birthday: the month is fine as 1 to 31 (i.e., we don't care that for instance "November 31" doesn't exist). Feel free to ignore leap years (which are a bit more complicated).

Gender: should be either 'male' or 'female'.

Personal chars: only 'smart', 'rich', 'lazy', 'annoying', and 'boring' are allowed (and you have to pick 1-3 of these [note: this should be done in normal PHP, not via regexps]).

Education: only 'ITU', 'AU', ..., 'KU' are allowed.

Optional Gmail address: if present, it must be a text that ends with '@gmail.com' (note that it should allow 'john.doe@gmail.com' and 'john.x.doe@gmail.com', but not 'john...doe@gmail.com' or '.doe@gmail.com')

Password: should comply with the following requirements:
- 1) should be minimum 4 characters;
- 2) should be maximum 8 characters;
- 3) should contain at least one letter character; and
- 4) should contain at least one numeric character.

[note: these requirements 1)-4) should give different error messages, so they should each be tested by a regexp]

Any questions?