PIC 40A

Lecture 18: PHP Arrays

PHP Arrays

- Arrays have names that begin with $.
- Its elements consists of key-value pairs.
- Can have elements with positive integer keys and/or string keys at the same time.
- Elements of an array need not be of the same type.

Declaring an array

2 ways to create an array:

1. Assign a value to an element of an array that does not exist

```
$my_array[2] = 2006;
```

2. use the `array` construct

```
$my_array = array(); # creates empty array
```

or

```
$my_array = array("Hello",2,TRUE);
```

Example

```
$primes[0] = 2;
$primes[1] = 3;
$primes[] = 5; # 5 is stored in index 2.
```

Note: even if the script had a scalar variable called `$primes` before the assignment `$primes[0]=2;` `$primes` becomes an array.

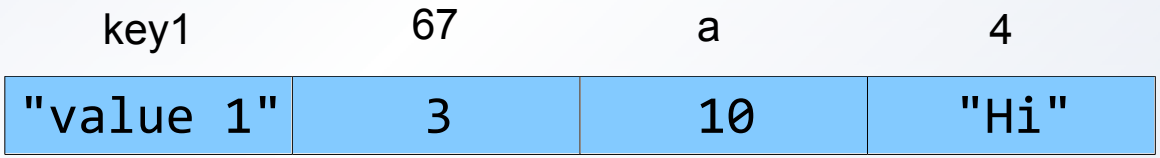
Using keys as indices

Arrays may be indexed either by ordinary non-negative integers or by “keys”.

Example:

```
$ar = array("key1"=>"value 1", 67=>3, "a"=>10, 4=>"Hi");
```

We now have an array where instead of usual integer indices (0,1,...) We have keys indexing the slots.



By default ordinary indices are used:

```
$my_array = array(1,2,3,4); #keys 0,1,2,3
```

Accessing PHP Arrays

- String key `'Joe'`

```
$password['Joe']="Hello!";
$pw = $password['Joe'];
```

- Number Key 2

```
$password[2]="Hello!";
```

Warning: "2" and 2 are the same key!

Printing array elements.

You can interpolate array element variables, but you have to be careful!

```
//These do not work!
print("Passwd:$password[‘Joe’].");
print "Passwd:$password["Joe"].";

//This is right.
print "Passwd:$password[Joe]";
```

Getting Array Keys and Values as Arrays with Integer Keys

- Use the `array_keys` and `array_values` functions

```
$table=array("Fry"=>"slurm", "Leela"=>"PlanEx3011");

$usernames = array_keys($table);
$passwds= array_values($table);
```

Example continued

```
$usernames[0] is "Fry"
$usernames[1] is "Leela"
$passwds[0] is "Slurm"
$passwds[1] is "PlanEx3011"
```

PHP Array functions

- `unset`

Deletes the contents of a scalar variable, an array entry, or an entire array.

- `count`

Returns the number of elements in an array.

- `sizeof`

This is same as `count`.

- `is_array`

Returns `TRUE` if the argument is an array.

Example

```
$list = array(1,2,3,4); #keys 0,1,2,3

unset($list[2]);
```

Now `$list` has keys `0,1,3` and elements `1,2,4`

```
unset($list);
```

Now `$list` is `NULL`

```
$a = 5;
unset($a);
```

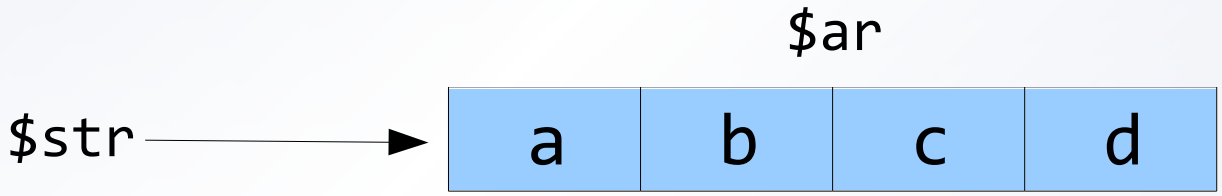
Now `$a` is `NULL`

explode

`explode` is JS equivalent of split. It takes an argument of a string and a delimiter and returns an array consisting of substrings of the string.

Example:

```
$str = "a:b:c:d";
$ar = explode(":",$str);
```



implode

implode does the inverse. It takes an array and returns a string where the entries are appended together using a delimiter.

Example:

```
$str = implode("^",$ar");
```

$str has the value: "a^b^c^d".

Accessing array elements in sequential order

- Every array has an internal pointer which points to the "current" element of the array.
- The "current" pointer is initialized to the first element of the array when the array is created
- To access and dereference the "current" pointer of an array, use the `current` function

Current Pointer Example

```
$snacks = array("chips", "pretzels", "candy");
$snack = current($snacks);
print("The first snack is $snack");
```

Output: The first snack is chips

Accessing array elements in sequential order

The `next` function moves an array's "current" pointer to the next element in the array and then dereference it.

The `prev` function moves the "current" pointer to the previous element in the array and then dereference it.

Moving Current Pointer Example

```
$snacks = array("chips", "pretzels", "candy");

$snack = next($snacks);
print("The 2nd snack is $snack");

$snack = prev($snacks);
print("The 1st snack is $snack");

# The 2nd snack is pretzels
# The 1st snack is chips
```

More about next

Current pointer is moved with the `next` function, which moves the pointer to the next array element and returns the value of that element. If the current pointer is pointing to the end of the array the function returns `FALSE`.

Example:

```
$snacks = array("chips", "pretzels", "candy");
$snack = current($snacks);
print ("$snack <br/>");
while($snack = next($snacks))
print("$snack <br/>");
```

Warning about `next`

If an element of the array has a value `FALSE` the loop in the preceding example would end before we reached the end of the array.

One may avoid this problem by using the `each` function.

More Array Traversing

`reset` function

- Moves an array's "current" pointer to the first element in the array and then dereferences it.

`end` function

- moves "current" to the last element in the array
and then dereferences it.

Moving Current Pointer Example

```
$snacks = array("chips", "pretzels", "candy");

$first = reset($snacks);
print("The first snack is $first");

$last = end($snacks);
print("The last snack is $last");

# The first snack is chips
# The last snack is candy
```

`each` function

- Returns a two element array consisting of the key and the value of the "current" element in the array.
- The keys of the two element array are "key" and "value".
- It then moves the "current" pointer to the next element of the array or `FALSE` if there isn't one.

each Example

```
$snacks = array("abe" => "chips", "abby" => "pretzels",
"darin" => "candy");

while($order= each($snacks))
{
    $partier = $order["key"];
    $snack = $order["value"];
    print("$partier wants $snack");
}

# abe wants chips
# abby wants pretzels
# darin wants candy
```

Other PHP Array Functions

- `key($array)`
  - returns the key of the "current" element of the array

- `in_array($value, $array, $strict)`
  - returns TRUE if the given `$value` exists in the array `$array`. If `$strict` is `TRUE`, then the type of `$value` must also match the type of the element in the array.

Using an array as a stack

- `array_push($array, $e1, $e2,..)`

Places the elements `$e1,$e2` at the end of the array `$array` and returns the new number of elements in the
array.

- `array_pop($array)`

Removes and returns the last element of the array `$array`, or `NULL` if the array is empty.

More PHP control structures

- `foreach` loop

Used to process all the elements of an array. There are two forms of `foreach`:

```
1.foreach($array as $value) {}
2.foreach($array as $key => $value){}
```

foreach Example

```
$snacks = array("abe" => "chips", "abby" => "pretzels",
                "darin" => "candy");

foreach($snacks as $val)
{
   print("$val<br/>");
}

# Output is:
# chips
# pretzels
# candy
```

foreach Example

```
$snacks = array("abe" => "chips", "abby" => "pretzels",
                "darin" => "candy");

foreach($snacks as $partier => $snack)
{
   print("$partier wants $snack<br/>");
}

# Output is:
# abe wants chips
# abby wants pretzels
# darin wants candy
```

Sorting PHP Arrays

```
sort($array)
```

- Sorts the values in `$array`, but replaces the keys by 0,1,2,3, etc...

- String values appear first in alphabetical order followed by the numeric values in ascending order.

Sorting PHP Arrays

`asort($array)`
Same as `sort($array)`, but key-value associations are preserved.

`ksort($array)`
Sorts `$array` by its keys, rather than by its values. Key-value associations are preserved.

Sorting PHP Arrays

- `rsort($array)`
- `arsort($array)`
- `krsort($array)`

These functions behave the same as `sort`, `asort`, and `ksort` respectively, but sort in the reverse order.