- PHP Expressions and Operators
- Similar to those in C++ / Java / Perl
- Be careful with a few operators
    - / in PHP is always floating point division
        - To get integer division, we must cast to int

        ```
        $x = 15;
        $y = 6;
        echo ($x/$y), (int) ($x/$y), "<BR />";
        ```

            - Output is 2.5 2
    - Mixed operands can produce odd results
        - Values may be cast before comparing

- To compare strings, it is better to use the C-like string comparison function, strcmp()
- Other string functions are listed in Sebesta Table 9.3
- Even the == operator has odd behavior in PHP when operands are mixed
- Ex: Consider comparing a string and an int
    - Any non-numeric PHP string value will “equal” 0
    - Any numeric PHP string will equal the number it represents
    - Ex: Consider comparing a string and a boolean
        - Regular PHP string value will “equal” true
        - “0” string will equal false
    - This behavior is consistent but confusing to the programmer and is probably best avoided

- An additional equality operator and inequality operator are defined

    === returns true only if the variables have the same value and are of the same type

    > If casting occurred to compare, the result is false

    !== returns true if the operands differ in value or in type

- Precedence and associativity are similar to C++/Java
    - See http://us2.php.net/manual/en/language.operators.precedence.php

- PHP Control Structures
- Again, these are similar to those in C++ / Java
  - **if**, **while**, **do**, **for**, **switch** are virtually identical to those in C++ and Java
  - PHP allows for an alternative syntax to designate a block in the if, while, for and switch statements
    - Advantage to this syntax is readability
      - keywords vs. brackets
      - Now instead of seeing a number of close braces, we see different keywords to close different types of control structures

- A nice feature of PHP is that the "control" resulting from a control structure is maintained even when you exit back to html mode
    - Thus, in <?php you can branch / loop etc.
    - You can then exit php ?> and format in straight html
- PHP also has the foreach loop
    - Similar to the Java for loop for Iterable objects
    - We will look at this when we discuss arrays
- See ex6.php

Lecture 3: PHP Arrays

- Arrays in PHP are quite versatile
  - See http://php.net/manual/en/language.types.array.php
- We can use them as we use traditional arrays, indexing on integer values
- We can use them as hashes
  - Associating a key with a value in an arbitrary index of the array
- In either case we access the data via subscripts
  - In the first case the subscript is the integer index
  - In the second case the subscript is the key value
- We can even mix the two if we'd like

- Creating Arrays
- PHP Arrays can be created in a number of ways
    - Explicitly using the array() construct
    - Implicitly by indexing a variable
    - Since PHP has dynamic typing, you cannot identify a variable as an array except by assigning an actual array to it
    - If the variable is already set to a string, indexing will have undesirable results – indexes the string!
    - However, we can unset() it and then index it
    - We can test a variable to see if it is set (isset() and if it is an array (is_array()) among other things
    - Size will increase dynamically as needed

Lecture 3: More on PHP Arrays

- Accessing Arrays – can be done in many ways
- We can use **direct access** to obtain a desired item
    - Good if we are using the array as a hash table or if we need direct access for some other reason
    - We provide the key and retrieve the value
- For **sequential access**, the foreach loop was designed to work with arrays
    - Iterates through the items in two different ways

      foreach ($arrayvar as $key => $value)

        > Gives both the key and value at each iteration

      foreach ($arrayvar as $value)

        > Gives just the next value at each iteration

- How can these both be done efficiently?
  - PHP arrays are not implemented in the traditional way
    - Ex: In Java or C++ the array is a contiguous collection of memory locations
  - PHP arrays more resemble a linked list (see Figure 9.3 in Sebesta text)
    - But wouldn't this not allow direct access?
  - The locations are also hashed
    - The "key" in PHP arrays is actually a hash value
  - So sequential access follows the linked list
  - Direct access accesses via the hash value

- Be careful – iteration via foreach is in the order the data has been generated, not by index order
    - i.e. it follows the linked list
    - Thus, even arrays with identical keys and values can have different orderings
- Items accessed in the arrays using foreach are copies of the data, not references to the data
    - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
    - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

Lecture 3: More on PHP Arrays

- The data in the array is not contiguous, so incrementing a counter for the next access will not work correctly unless the array index values are used in the "traditional" way

```
for (int $i = 0; $i < count($A); $i++):
        echo “$A[$i] <br/>”;
endfor;
```

- We know that there are count($A) items in $A
- What we do NOT know, is under which indices they are being stored
- There is no requirement that they have to start at 0 or even be integers at all
    - See ex7.php

- In addition to foreach, there are other array iterators that we can use
- Ex: Using next to access the array elements
    - The next() function gives us the next value in the array with each call
        - It moves to the next item, then returns it, so we must get the first item with a separate call (ex: use current())

```
$curr = current($a1);
while ($curr):
    echo "\$curr is $curr <br/>\n";
    $curr = next($a1);
endwhile;
```

Lecture 3: More on PHP Arrays

- Ex: Using each to iterate:
  - The each() function returns a pair with each call
  - A key field for the current key
  - A value field for the current value
  - It returns the next (key,value) pair, then moves, so the first item is no longer a special case

```
while ($curr = each($a1)):
    $k = $curr["key"];
    $v = $curr["value"];
    echo "key is $k and value is $v <BR />\n";
endwhile;
```

  - This function may be preferable to next() if it is possible that FALSE or an empty string or 0 could be in the array
    - The loop on the previous slide will stop for any of those values

- Both of these iteration functions operate similar to the **Iterator interface** in Java
  - Iterate through the data in the collection without requiring us to know how that data is actually organized
  - However, **unlike in Java**, if the array is changed during the iteration process, the current iteration is NOT invalidated
    - Since new items are always added at the "end" of the array (from an iterator's point of view) adding a new item during an iteration does not cause any data validity problems
    - However, we need to be careful if doing this – can lead to an infinite iteration

Lecture 3: Sorting PHP Arrays

- There are various predefined sort functions in PHP
- sort (rsort for reverse)
    - Sorts arrays of numbers numerically
    - Sorts arrays of strings alphabetically
    - If mixed, the strings count as 0 compared to numbers
    - Reindexes array so that keys start at 0 and increment from there
- asort
    - Same as sort but retains the original key values (arsort for reverse)

Lecture 3: Sorting PHP Arrays

- PHP uses Quicksort to sort arrays
- This means that PHP sorting is NOT STABLE
- What does it mean for a sort to be STABLE?
    - Given equal keys K_1 and K_2, their relative order before and after the sort will be the same
- Due to data movement during partition, Quicksort is not stable
    - Implications?
    - If we want stability, we will have to do it ourselves
        - See Web for some solutions
- See ex8.php

Lecture 3: Two-dimensional Arrays

- Array values can be any legal PHP type
- This includes the array type, and allows for arbitrary dimensional arrays
- We may think of them as "arrays of arrays"
- It seems odd but once you know the array syntax it follows quite naturally

```
$a[0] = array(1,2,3,4);
$a[1] = array(5,6,7,8);
$a[2] = array(9,10,11,12);
```

Lecture 3: Two-dimensional Arrays

- We can also use "normal" indexing for 2-D PHP arrays
- Keep in mind that the key values are still arbitrary, so we need to be careful
- More general access can be done via iterators or recursive functions – we will see this soon
    - See ex9.php

Lecture 3: CGI and Scripts

- CGI - Common Gateway Interface
  - http://en.wikipedia.org/wiki/Common_Gateway_Interface
  - http://tools.ietf.org/html/rfc3875
- Interface for Web servers that interact with browsers, utilizing scripting languages and the HTTP (HyperText Transfer Protocol)
- Used to allow data interaction between clients and server scripts
  - Ex. Extracting data sent via HTTP requests and passing to scripts
  - Scripts can then use this data as input and act on it

Lecture 3: CGI and Scripts

- Two best known HTTP methods: GET and POST
    - GET
    - appends user input to URL and requests corresponding document
    - server parses URL - first part is a program that it invokes, second part is parameters passed along

Recommended usage for safe and idempotent requests

> I dem WHAT?

> Isn't superman idempotent?

- Safe:

> For retrieval only – has no side effects on the server

- Idempotent:
    - Making N > 1 identical requests has the same effect as making only 1 request

- POST
  - sends data as a stream to script program
  - more suitable for large amounts of data
  - arguments are not shown in address but are still extracted and processed by server
  - Used for requests that may alter / update the server
    - i.e. NOT safe and NOT idempotent
    - Ex: update a database
    - Ex: submit a payment

- GET and POST are often used within HTML **forms**
- User enters data into form and then SUBMITS it
- Browser processes form and passes choices and information to the url specified
- Server invokes appropriate script utilizing requested method, extracting submitted data
    - Most scripting languages (including PHP) have predefined ways to easily extract this data
    - This data is used as input to the script

Lecture 3: CGI and Scripts

- Results are sent back to browser and displayed in the Web browser
- See getpost.html and getpost.php
- Since scripts are executed by the server and can access files on the server
- It is prudent for a webmaster to be cautious about the scripts placed onto the server
- For example many servers will only execute “approved” scripts placed into an approved directory

Lecture 3: Processing Forms with PHP

- Using PHP with forms is fairly simple
- When forms are submitted the server executes the php script, returning the resulting html
    - Remember that some of the file is unchanged, since it may not have an embedded php script within it
    - Server can be set to that the form variables can be accessed directly by simply using the $ sign
    - However, it is better to access the variables from the $_POST array (or the $_GET array)
        - The form element name is the key into the array
        - Discuss and see getpost.php

Lecture 3: Processing Forms with PHP

- We can also use PHP to create forms
    - However, it is really just HTML that we are using
    - We can "interleave" the PHP and html to get the desired overall result, or we can have PHP output the appropriate HTML tags
    - So if you don't know it yet – learn some HTML
        - See Chapter 2 in Sebesta
- See ex12.php, ex12b.php – note many comments!
    - Note how the script interacts with the data file
        - It will show as many rows in the table as there are lines in the file
    - Note how the PHP and html are interleaved

Lecture 3: PHP Files

- Using files in PHP is fairly straightforward
- Can open a file for reading, writing, append, and a couple variations of reading+writing
    - *Note 1: Files are not covered in the Sebesta text*
    - *Note 2: You may have to set some permissions on your file system to allow your server write access to files*
- There are a few different ways to access files in PHP
- Many C file functions are almost identical in PHP
    - Ex: fopen, fseek, fscanf, fgetc, fgets
    - See the manual for complete list

- Opening files
- Typically we use fopen() to open a file for either reading or writing

  $fp = fopen(<filename>, <mode>);

  - Where <filename> is the path/name of a file that is accessible to the server
  - Where <mode> specifies how the file will be accessed
    - Ex: "r" → read only

      "r+" → read/write with pointer at beginning
    - The above modes require the file to already exist

"w" → write only

"w+" → write / read, truncating previous file length to 0

  - For the above modes, the server will attempt to create the file if it does not exist.
  - Also "a" and "a+" for append modes

- Reading from files
- For text files, we can read different amounts per read depending on our requirements
    - Read a single character at a time
    - Read the entire file into a single string
    - Read the lines of the file into an array of strings
- Can also read binary data if necessary
    - Ex: images, audio, etc.

- PHP allows all of these with various functions
    - Look at the options in the manual
    - See: http://php.net/manual/en/ref.filesystem.php

- Writing to files
- Most commonly done with fwrite
- Again see manual for details

- Very Simple Example:
- See readwrite.php

Lecture 3: Debugging Note

- Many situations that produce compilation or run-time errors in Java will not do so in PHP
  - Ex: Accessing a variable that has no value:

    ```
    $count = $count + 1;
    ```

  - Ex: Reading a file that does not exist:

    ```
    $data = file(“nonexistentfile.txt”);
    ```

- However, these situations will produce warnings, which we can elect to see or not see in the returned web page
  - We can determine whether these warnings (and actual errors) are seen or not via .htaccess files

Lecture 3: Debugging Note

- These are configuration files that allow per directory configuration options for the server
- For example the settings:

  php_value display_errors 1

  php_value display_startup_errors 1

  – will send PHP warnings back to the client browser
- And the settings:

  php_value display_errors 0

  php_value display_startup_errors 0

  – will hide the warnings from the user
- *Note: In some installations these cause problems for the server – if these cause an error in your server don't use them*

- Flocking files
    - See http://php.net/manual/en/function.flock.php
- The flock() function is called to restrict access to files (when necessary) to one "user" at a time
    - If each "user" calls flock() prior to accessing a file pointer to the same file, only one will be allowed to access it at a time
- Why do we need this?
    - Multiple users frequently access the same server
    - Server typically spawns a separate process for each user

Lecture 3: PHP Files

- These processes can execute in pseudo-parallel or in actual parallel depending on how the server is configured
- Consider the following scenario for process P1:
  - Read a file into an array
  - Update a value in the array
  - Write the array back to the file
- What if process P2 writes to the file between P1's reading and writing?
- If used correctly, flock() can prevent this problem
  - See flock.php

Lecture 3: Maintaining State

- HTTP is a stateless protocol
- It is simply defines how clients and servers communicate with each other over the Web
- Yet with many Web applications, maintaining state is important
  - Ex: When a customer logs into a site such as Amazon, he/she may go through multiple pages
    - We may want to keep track of the user him / her self (authentication information)
    - We may want to keep track of what he / she has been doing

Lecture 3: Maintaining State

- State can be maintained in various ways and in various places
  - Ex: We can store information on the server or on the client
  - We will examine several of these throughout the rest of the term

- One way of maintaining state is via Cookies
  - http://en.wikipedia.org/wiki/HTTP_cookie

Lecture 3: Cookies

- Cookies – what are they?
- Small pieces of information (up to 4K) initially sent by the server to the client and stored on the **client machine**
- When client next connects to a server, it sends cookies from that server back to it
- Information about the client can then be extracted by the server
    - If no cookie, server can create a new cookie for the client and send it with the response
    - However, browsers can disable cookies
        - Can cause problems if server is dependent upon them

- Cookie format:
  - Name: name of the cookie – typically used to extract / examine the cookie
  - Value: contents of the cookie – seems like a simple value but can be an array if generated correctly
  - Domain: domain of the server that is to receive the cookie – actual domain of server must match domain stored in the cookie
    - Idea is that other servers cannot look at all of your cookies to see what you have
    - If not explicitly set in the cookie, it is the full domain of the server that created the cookie

- Expires: When cookie will expire
  - Timestamp: Very specific format is required, but we can use function calls to make it easier
- Path: Path in server from which cookie can be sent
  - If not specified it is the full path from where cookie was set
- Secure: Does cookie require secure server using https
  - Default is no

Lecture 3: Sending Cookies to Client

- Cookies are sent with the HTTP header of an html file:
  - Set-Cookie: oreo=Count Chocula; domain=.chocolate.com;
path=/cgi/bin;
expires=Thu, 08-Jun-2015, 16:15:00 GMT;
    - Must be set PRIOR to any html tags (since it is sent with the header)
  - If not sent with HTTP header will not be interpreted as a cookie
  - If client does not accept cookies it will just discard them
    - We can send a cookie and test to see if client accepts cookies

Lecture 3: Cookies in PHP

- Cookies in PHP are fairly easy to use:
- setcookie() function is called to create a cookie that will be sent to the client
    - See http://php.net/manual/en/function.setcookie.php
    - As always with cookies, they must be sent with the http header
        - Thus, you should determine and set any cookies in PHP mode prior to using any html (or even simple text)
- $_COOKIE array contains the cookies received back from the client machine
    - Cookies sent to client by server previously
    - Associative array allows access of cookies by name

Lecture 3: Cookies in PHP

- Thus, to maintain state a server can:
  - Send the client a cookie the first time the client connects to the server
  - Receive and update / modify the cookie as client navigates the site
    - Or send additional cookies
  - Use the presence and / or value of cookies to discern information about the client
    - Ex: A repeat customer – time of last visit
    - Ex: A current customer – last request or last page visited