Review: Machine language & high level programming languages
- Machine language vs. high level programming languages
- Compiler vs. interpreter
- Perl

Review: A Simple Perl Program
```perl
#!/usr/local/bin/perl
print "My First Perl Program!";
```

Review: Variables and Values
- Store numbers or strings
- Legal names for variables in Perl
- The value stored in a variable can be retrieved and changed at any time, and we can perform operations on it.

Review: Arithmetic Operators:
<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>7 + 7</td>
<td>= 14</td>
<td>Addition</td>
</tr>
<tr>
<td><code>-</code></td>
<td>7 - 7</td>
<td>= 0</td>
<td>Subtraction</td>
</tr>
<tr>
<td><code>*</code></td>
<td>7 * 7</td>
<td>= 49</td>
<td>Multiplication</td>
</tr>
<tr>
<td><code>/</code></td>
<td>7 / 7</td>
<td>= 1</td>
<td>Division</td>
</tr>
<tr>
<td><code>**</code></td>
<td>7 ** 7</td>
<td>= 823543</td>
<td>Exponents</td>
</tr>
<tr>
<td><code>%</code></td>
<td>7 % 7</td>
<td>= 0</td>
<td>Modulus</td>
</tr>
</tbody>
</table>

Review: Assignment Operators: perform an arithmetic operation and then assign the value to the existing variable.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td>Addition</td>
<td>($x += 10)</td>
</tr>
<tr>
<td><code>-=</code></td>
<td>Subtraction</td>
<td>($x -= 10)</td>
</tr>
<tr>
<td><code>*=</code></td>
<td>Multiplication</td>
<td>($x *= 10)</td>
</tr>
<tr>
<td><code>/=</code></td>
<td>Division</td>
<td>($x /= 10)</td>
</tr>
<tr>
<td><code>%=</code></td>
<td>Modulus</td>
<td>($x %= 10)</td>
</tr>
<tr>
<td><code>**=</code></td>
<td>Exponent</td>
<td>($x **= 10)</td>
</tr>
</tbody>
</table>
Examples: a Perl script uses a single variable to store several types of data

```perl
my $variable = 7;
print("my variable contains the value: $variable\n");
my $variable = 7.3217;
print("my variable contains the value: $variable\n");
my $variable = "apple";
print("my variable contains the value: $variable\n");
```

The output:
- my variable contains the value: 7
- my variable contains the value: 7.3217
- my variable contains the value: apple

String operations

- Concatenation
  ```perl
  $full_name = $first_name . " " . $last_name;
  ```

- Repetition
  ```perl
  $david_cubed = $first_name x 3;
  ```

Review: Input from keyboard

```perl
#!/usr/local/bin/perl
print "Please enter your name\n";
#accept keyboard input, read it into $name
$name=<STDIN>;
print "Hello $name";
```

Printf: formatted output

```perl
my $variable = 12.3456789;
printf("The value of my variable is %d", $variable);
    # The value of my variable is 12
    # (Only keep the integer part.)
print("The value of my variable is %s", $variable);
    # The value of my variable is 12.3456789
    # (Treat the variable as a string)

my $variable = 12.3456789;
printf("The value of my variable is %f", $variable);
    # The value of my variable is 12.35
    # (Treat the variable as a real number, keep 6 digits after the radix point.
    # Add 0's if less than 6 digits)
printf("The value of my variable is %10.2f", $variable);
    # The value of my variable is 12.35
    # (at least 10 columns wide in total, with two figures after the decimal place.)
```

Arrays

- List of scalars
  ```perl
  @names = ("David", "Mary", "Jim");
  @numbers = (12, 2, 4, 5, 8);
  ```

```
**Array Operations:**

- Accessing an element
  - Index starts at 0;
  - print "$name[2] \n";

- Get the array length
  - $#<array name> gives the last index of the array.
  - $#names + 1 is the length of the array names.

**Hashes:**

- Also know as dictionaries
- Contain keys and values (1:1 correspondence)
- Begin with %.
- Assign values
  - %student_grades = ('David' => 'A', 'Mary' => 'B', 'Jim' => 'C')
  - $student_grades('David') = 'A';

**Hash Operations:**

- Accessing hash elements
  - %<hash>[<key>]
  - print "David's grade is $student_grades('David') \n";

- Accessing keys
  - @students = keys %student_grades;

**Simple Perl Commands:**

- print
  - Prints list of arguments that it is passed.
- printf
  - Writes formatted output.
- <STDIN>
  - STDIN is the input stream associated with standard input. Standard input by default comes from your keyboard.
  - $name = <STDIN>; This statement reads from standard input into a scalar variable named name.

**Simple Perl Commands:**

- chop
  - The chop function is used to indiscriminately remove the last character from a variable.
  - chop($name); This statement removes the last character in the variable $name

- chomp
  - removes 'newline' ('\n') from the end of a scalar variable.
  - Chomp($name);
Announcements:

- Lab 3 this week
- Next lecture: Perl: flow control