Why Use Perl?

1. Perl is *easier* to learn.
   - C++ vs. Perl
2. Perl is *powerful*.
   - Quickly process text (like the letters A,C,G,T!)
3. Perl is *free*!
   - Runs on *any* computer!
4. Peer Pressure: “Everybody’s doin’ it.”
   - Ensembl
   - NCBI

* Individual results may vary.
The Basic Container of Perl: Scalars

How do you store stuff in Perl?

- Single pieces of data are stored in variables.
- Variables in Perl are called ‘scalar’ variables.
- You can always tell a scalar because it starts with $
- Examples:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$blip;</td>
<td>This is a scalar variable</td>
</tr>
<tr>
<td>$blip = 3.14;</td>
<td>‘$blip’ has been assigned a value.</td>
</tr>
<tr>
<td>$blip = “Swordmonkey”;</td>
<td>‘$blip’ has been assigned a different value.</td>
</tr>
</tbody>
</table>

-A scalar variable can be assigned just about anything, including:
- A numeric value (0, 0.3, 343232, 0.99301, etc…)
- A single character (‘a’, ‘3’, ‘+’, etc…)
- A series of characters, aka: as string (‘Swordmonkey’, ‘Damian Fermin’)
- Another variable! $varA = 4.0;
  $varB = $varA;
- Other data types in Perl (will deal with this later.)
Assigning Values to a Scalar

$var1 = 0.034;
$var2 = “Damian Fermin”;
$var3 = ‘I hate this $%#* class!’;
$var4;

What does $var4 have?

How to print the contents of a variable to the screen

‘print’ command allows you to print stuff to the screen.*

print “var1 contains $var1\n”;
print “var2 contains $var2\n”;
print “var3 contains $var3\n”;  
print “var4 contains $var4\n”;  

‘\n’ is the “new line” character
You put it where ever you want to start a new line

‘;’ is the “end-of-line”
You put it at the end of every complete line of perl code.

* Always start and end your print statements with quotes.
Example Perl Script Using Scalar Variables

#!/usr/bin/perl -w

## Example script using Scalar Variables

$var1 = 6;
$var2 = "Damian";
$money = 1.43;

print "Hello my name is: $var1\n";
print "I am $var2 feet tall.\n";
print "I paid $money dollars for this haircut.\n";

exit;

What you get when you run this script. (Screen Output)

Hello my name is Damian.
I am 6 feet tall.
I paid 1.43 dollars for this haircut.
Basic Structure of a Perl Script

<table>
<thead>
<tr>
<th>Line Num.</th>
<th>Perl Code</th>
<th>What the line is</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#!/usr/bin/perl</td>
<td>Shebang Line</td>
</tr>
<tr>
<td>2</td>
<td>## A simple script.</td>
<td>A comment</td>
</tr>
<tr>
<td>3</td>
<td>$name = &quot;Damian&quot;;</td>
<td>Assignment of a variable</td>
</tr>
<tr>
<td>4</td>
<td>$height = 6;</td>
<td>Assignment of a variable</td>
</tr>
<tr>
<td>5</td>
<td>print &quot;Hello my name is $name\n&quot;;</td>
<td>Printing text an embedded variable.</td>
</tr>
<tr>
<td>6</td>
<td>print &quot;I am $height feet tall\n&quot;;</td>
<td>Printing text and embedded variable.</td>
</tr>
<tr>
<td>7</td>
<td>exit;</td>
<td>End of Program</td>
</tr>
</tbody>
</table>

Screen Output

Hello my name is Damian.
I am 6 feet tall.
Practice at using Scalars in Perl

#!/usr/bin/perl -w
## ex1.pl
## Example script using Scalar Variables
$name = "Damian";
$areaCode = 734;
$airSpeed = "I don't know!";

print "\n\nMy name is:  $name\n";
print "My area code is: $areaCode\n";
print "Maximum air speed velocity of an African Swallow: $airSpeed\n\n";
exit;

What’s the output?: My name is: Damian
My area code is: 734
Maximum air speed velocity of an African Swallow: I don't know!
Array: The Sequential Storage Container

- An array is a collection of scalar variables stored sequentially
- Syntax: @arrayName
- Order is important in an array
- An array can hold any type of scalar variable data.
- Each piece of data is stored in a specific memory location called an element
- Array elements have addresses that **ALWAYS** start at 0.
- Each element address is called an index

Example of an Array:  `@ary1 = (55, 0.42, 10023.39, “apple”, 993, “Y”, “toga party”);`

How the data of ‘@ary1’ is stored:

<table>
<thead>
<tr>
<th>Element Number (index)</th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
<th>[6]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Contents</td>
<td>55</td>
<td>0.42</td>
<td>10023.39</td>
<td>apple</td>
<td>993</td>
<td>Y</td>
<td>toga party</td>
</tr>
</tbody>
</table>

• You can access individual elements of an array.
• To do this you need to specify the exact element number you want:

| Prints this to the screen | ary[6] contains: toga party |
Practice Using Arrays

#!/usr/bin/perl

## ex2.pl
## Example using Array.

@ary1 = (55, 0.42, 10023.39, "apple", 993, "Y", "toga party");

print "\n\nContents of ary1:\n"

print "ary1[0] = \$ary1[0]\n";
print "ary1[1] = \$ary1[1]\n";
print "ary1[2] = \$ary1[2]\n";
print "ary1[3] = \$ary1[3]\n";

exit;

Output?

Contents of ary1:
ary1[0] = 55
ary1[1] = 0.42
ary1[2] = 10023.39
ary1[3] = apple
push(): Function used to add something to the END of an array
- Syntax:  `push(@ary1, $scalar1);`

Example:

```perl
$var1 = 2;
$var2 = "wet burrito"
@ary1 = (1.002, 3.220, 4.005);
@newAry = ();

push(@newAry, $var1);
push(@newAry, $var2);
push(@newAry, @ary1);
push(@ary1, @var2);
```

What’s in ‘@newAry’?
```
2
wet burrito
1.002
3.22
4.005
```

What’s in ‘@ary1’ now?
```
1.002
3.22
4.005
wet burrito
```
- `pop()`: Function to *‘pop off’* an element from the END of an array
- Element that is removed can be immediately assigned to another variable

- Syntax: `pop(@arrayName);`

Example:

```perl
$var1 = 2;
$var2 = "wet burrito"
@ary1 = (1.002, 3.220, 4.005);
@newAry = ();  
$temp = 0;

push(@newAry, $var1);
push(@newAry, $var2);
push(@newAry, @ary1);

$temp = pop(@newAry);
```

Current Contents of `newAry`

```
2
wet burrito
1.002
3.22
4.005
```

What does `temp` contain now?
- `temp = 4.005`

What does `newAry` contain now?
- To pop data off the front use `shift()` command.
Associative Arrays (aka: Hashes)

- Do NOT store data in a sequential manner
- Order is unimportant
- Data is stored with a key-> value format
- Each key is associated with exactly one value.
- Keys can be any scalar data (letters, numbers, words, etc…)
- Syntax: %hashName

Two ways to initialize a hash with data:

```perl
%myHash = (key => value,
           key => value,
           ...,
           key' => value);
```

```
%myHash = ( key, value, key, value);
```

Real life example: Internal Revenue Service
“You’re a number, not a name.”
Example:

```perl
%myHash = (
    name => 'Damian',
    year => 2000,
    dept => 'Basket Weaving',
    fruit => 'tomato'
);
```

- You can access any ‘value’ using its ‘key’
- Syntax for assigning/retrieving an individual value using a key:

Retrieve:  `$myHash{"dept"}`; Returns  “Basket Weaving”

Assign:  `$myHash{"dept"} = "Bioinformatics"`;  Assigns Bioinformatics to the key ‘dept’

Special keywords for Hashes

- You can retrieve just the keys of a hash, or just the values of a hash using the keywords **keys**, and **values** respectively. (More on this later)
Difference Between Array and a Hash

<table>
<thead>
<tr>
<th>How a Hash Stores Data</th>
<th>How an Array Stores Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Data is <strong>not</strong> stored sequentially</td>
<td>- Data is stored sequentially</td>
</tr>
<tr>
<td>- You can only get to data using a key</td>
<td>- You can get to data by knowing its index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>element0</td>
</tr>
<tr>
<td>[1]</td>
<td>element1</td>
</tr>
<tr>
<td>[2]</td>
<td>element2</td>
</tr>
</tbody>
</table>
Practice Using Hashes

#!/usr/bin/perl -w

## ex3.pl
## Example using Hashes

%myHash = (
    name => "Damian",
    dept => "Basket Weaving",
    zipCode => 48108
);

print "\n\nName: \$myHash\{name\}\n";
print "Department: \$myHash\{dept\}\n";
print "Zip Code: \$myHash\{zipCode\}\n";

exit;

Output?
Name: Damian
Department: Basket Weaving
Zip Code: 48108
## Math Operators

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>Addition</td>
<td>$2 + 3 = 5$</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Subtraction</td>
<td>$4 - 2 = 2$</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Division</td>
<td>$10 / 5 = 2$</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Multiplication</td>
<td>$3 * 3 = 9$</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Modulus (returns remainder</td>
<td>$10 % 3 = 1$ (when 10 is divided by 3 the remainder is 1)</td>
</tr>
</tbody>
</table>
## Other Math Operators

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
</table>
| `+=`   | Add to variable on left, the amount on the right | `$var1 = 10; $
$var1 += 3; $
‘$var1’ now holds 13 |
| `-=`   | Subtract from variable on left, the amount on the right | `$var1 = 10; $
$var1 -= 3; $
‘$var1’ now holds 7 |
| `*=`   | Multiply variable on left, by the amount on the right | `$var1 = 10; $
$var1 *= 3; $
‘$var1’ now holds 30 |
| `/=`   | Divide variable on left, by the amount on the right | `$var1 = 10; $
$var1 /= 3; $
‘$var1’ now holds 3.333 |
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
</table>
| ** ** | Exponential ( ie: $x^y$) | $2**3 = 2 \times 2 \times 2 = 8$  
Raise 2 to the third power |
| ++     | Increment value on left by 1 | `$var1 = 1;`  
`$var1++;`  
`'var1' now holds 2` |
| --     | Decrement value on left by 1 | `$var1 = 7;`  
`$var1--;`  
`'var1' now holds 6` |
Common Math Rules Still Apply

```perl
#!/usr/bin/perl -w

$a = 2 * 3 + 10;  # Multiplication before Addition
print "a equals $a\n";

$b = 2 * 3 * 10;  # Multiplication moves from left to right
print "b equals $b\n";

$c = 2 * 10 / 5;  # Multiplication before Division
print "c equals $c\n";

$d = 2 ** 2 * 5 + 10;  # Exponential comes before multiplication
print "d equals $d\n";

exit;
```

Output?

- a equals 16
- b equals 60
- c equals 4
- d equals 30
#!/usr/bin/perl -w
# ex4.pl

$num1 = 4.03;
$num2 = 0.97;
$sum = 0;
$product = 0;
$quotient = 0;
$remainder = 0;

print "\n\nnum1 = $num1\n";
print "num2 = $num2\n\n";

$sum = $num1 + $num2;
print "$num1 + $num2 = $sum\n\n";

$product = 5 * $sum;
print "5 * $sum = $product\n\n";

$quotient = $product / 2;
print "$product / 2 = $quotient\n\n";

$remainder = $product % 3;
print "$product % 3 = $remainder\n\n";

exit;

Output?

num1 = 4.03
num2 = 0.97
4.03 + 0.97 = 5
5 * 5 = 25
25 / 2 = 12.5
25 % 3 = 1