References

References

- Analagous (somewhat) to pointers in C/C++
  - Far less messy, and definitely less dangerous
- You can take a reference to any variable - array, hash, or scalar.
  - The reference itself is a scalar.
- Use the \ to create a reference:
- `my @foo = (1, 2, 3);`
- `my $aref = \@foo;`
- $aref now contains a reference to the array @foo;
  - Changes to @foo will affect array referenced by $aref
  - Changes to the array referenced by $aref will affect @foo

De-Referencing

- De-reference a reference by enclosing it in { } and prepending the appropriate sigil
  - (array Ł @, hash Ł %, scalar Ł $)
- **`$aref = \@foo;`**
- **`@new_array = @{$aref};`**
  - contents of { } a simple scalar can drop them: @$aref
- **`@{$aref}`** can be used like any other array:
  - **`push @{$aref}, 'val1', 'val2';`**
- **`@new_array`** is a different array, which is initialized with the values that the array referenced by $aref contains.
  - same as if you'd done: **`@new_array = @foo;`**
  - Changes to **`@foo`** or **`$aref`** do NOT affect **`@new_array`**
  - Changes to **`@{$aref}`** DO affect **`@foo`**

Referencing Other Types

- You can also take references to other kinds of variables:
- `my %age_of = (Paul=>29, Tom=>20);`
- `my $href = \%age_of;`
- `my $bar = "hello world\n";`
- `my $sref = \$bar;`
- To dereference, `%{$href}` and `${$sref}`

Anonymous References

- A value need not be contained in an existing variable to create a reference.
- To create an anonymous array reference, use square brackets, instead of parens
- `my $arr_ref = [20, 30, 50, "hi!!"];`
- `my @a = @{$arr_ref};`
  - @a Ł (20, 30, 50, "hi!!");
- For hash references, use curly brackets, instead of parens:
- `$h_ref = {sky=>'blue', grass=>'green'};`
- `my %color_of = %{$color_of_ref};`
  - %color_of Ł (sky => 'blue', grass => 'green');
- There is no "clean" way to create an anonymous scalar ref:
- `my $s_ref = do { my $x; \$x; };`

Elements of Referenced Values

- TIMTOWTDI
- `my $a_ref = ['Hi', 'Hiya', 'Hello'];`
- `@{$a_ref} Ł ('Hi', 'Hiya', 'Hello')`
  - `${$a_ref}[2] Ł 'Hello'`
  - `$$a_ref[2] Ł 'Hello' #simple scalar`
  - `$a_ref->[2] Ł 'Hello'`
- `my $h_ref = {foo=>'bar', alpha=>'beta'};`
- `%{$h_ref} Ł (foo=>'bar', alpha=>'beta')`
  - `${$h_ref}{foo} Ł 'bar'`
  - `$$h_ref{foo} Ł 'bar' #simple scalar`
  - `$h_ref->{foo} Ł 'bar'`
- These are all valid and acceptable. The form you choose is whatever looks the best to you.
  - I personally prefer the array notation. It looks cleaner to me.

Uses for References

- To create a 2D array, create an array of array references:
- **`my @two_d = ([1, 2], [3, 4], [5, 6]);`**
- **`$two_d[1]`** is a reference to an array containing (3, 4)
- **`@{$two_d[1]}`** is an array containing (3, 4)
- **`${$two_d[1]}[0]`** is the scalar value 3.
  - note braces cannot be dropped!!! $two_d[1] is not a simple scalar!
- **`$two_d[1]->[0]`** is the same thing
- Any time an arrow separates two brackets ([] or {}), it can be dropped completely
  - **`$two_d[1][0]`** is also the scalar value 3.
  - Other arrows CANNOT be dropped:
    - **`$foo->[1][0]`** is NOT equivalent to **`$foo[1][0]`**

More Complicated

- Using similar methods, you can create arrays of hashes, hashes of arrays, hashes of hashes, arrays of arrays of hashes, hashes of hashes of arrays, arrays of hashes of arrays, ...
- ```
  my %letters_in = (
     lower => ['a' .. 'z'],
     upper => ['A' .. 'Z']
  )
  ```
- **`$letters_in{lower}`** is an array reference;
- **`@{$letters_in{lower}}`** is an array;
- **`${$letters_in{lower}}[1]`** is the scalar value 'b'.
- **`$letters_in{lower}->[1]`** is the scalar value 'b'
- **`$letters_in{lower}[1]`** is the scalar value 'b'

References and copies

- **`my @nums = (1..10);`**
- **`my $ref1 = \@nums;`**
- **`my $ref2 = [ @nums ];`**
- What's the difference?
  - **`$ref1`** contains a reference to **`@nums`**. Changes to **`@nums`** affect **`@{$ref1}`** and vice-versa.
  - **`$ref2`** contains a reference to an anonymous array, that happened to be populated with the values **`@nums`** contained at that moment.
    - Changes to **`@nums`** have no effect on **`@{$ref2}`**, nor vice-versa.
- **`my $ref3 = $ref1;`**
  - **`$ref3`** and **`$ref1`** are two references to the **same** array. Changes to **`@{$ref3}`** also change **`@{$ref1}`** and **`@nums`**
  ```

Schwartzian Transform

- Randal Schwartz devised a method to efficiently sort a list based on a computation-intense calculation
- To sort a list of files based on file size
- `sort {-s $a <=> -s $b} @files;`
- The stat has to be done twice every time two files are compared. That's costly!
-
```
my @sorted =
   map  { $_->[0] }
   sort { $a->[1] <=> $b->[1] }
   map  { [$_, -s $_] } @files;
```
- Now stat is done only once for each filename

Help available

- **`perldoc perlreftut`**
  - tutorial on references
- **`perldoc perllol`**
  - "lists of lists" – very inaccurately named
- **`perldoc perldsc`**
  - Data Structures Cookbook: building complicated structures
- **`perldoc perlref`**
  - Reference reference.