Perl for Biologists

Session 5
April 1, 2015

Hashes

Jon Zhang
Review of Session 4

• Array explicit declaration
  ```perl
  @array = (1, 5, "a", 77, "abcd", 99);
  ```

• Array range declaration
  ```perl
  @array = 1..9;
  ```

• Array quoted word declaration
  ```perl
  @array = qw(jon zhang perl 2013);
  @array = qw*jon zhang perl 2013*;
  ```
Review of Session 4

• Array access

@array = (1, 5, "a", 77, "abcd", 99);
print $array[0]; # prints "1"
print $array[4]; # prints "abcd"
print $array[5]; # prints "99"

$i = 3;
print $array[$i]; # prints "77"
Review of Session 4

@array = (1, 5, "a", 77, "abcd", 99);

• push (@array, $value): Appends $value to the end of @array
  $value = 88;
  push (@array, $value);
  print $array[6]; # prints "88"

• $value = pop (@array): Removes last element of @array, sets $value to the removed element
  $value = pop (@array);
  print $value; # prints "88"
@array = (1, 5, "a", 77, "abcd", 99);

- $value = shift (@array): Removes first element of @array, sets $value to the removed element
  $value = shift (@array);
  print $value;  # prints "1"
  print @array[0];  # prints "5"

- unshift (@array, $value): Adds $value to the front of @array, all other elements shifted back one index
  $value = 1;
  unshift (@array, $value);
  print @array[0];  # prints "1"
Review of Session 4

$array = (1, 5, "a", 77, "abcd", 99);

• @reverse_array = reverse @array: Sets @reverse_array as a reverse order @array
  @reverse_array = reverse (@array);
  print $reverse_array[0]; # prints "99"

• @sorted_array = sort @array: Sets @sorted_array as an ASCII sorted @array
  @sorted_array = sort @array;
  print $sorted_array[5]; # prints "abcd"
Review of Session 4

@array = (5, 7, 23, 8, 1, 4);

- @sorted_array = sort {$a <=> $b} @array: Sets @sorted_array as a numeric sorted @array
  @sorted_array = sort {$a <=> $b} @array;
  print $sorted_array[0];  # prints "1"
  print $sorted_array[5];  # prints "23";
@array = (1, 5, "a", 77, "abcd", 99);

• @spliced_array = splice (@array, $start_index):
Removes everything @array starting at $start_index, and returns it to @spliced_array

$start_index = 3;
@spliced_array = splice (@array, $start_index);
print $spliced_array[0]; # prints "abcd";
Review of Session 4

- @spliced_array = splice (@array, $start_index, $length): Removes $length elements from @array starting at $start_index, and returns it to @spliced_array.

- @spliced_array = splice (@array, $start_index, $length, @replacement): Removes $length elements @array starting at $start_index, and returns it to @spliced_array. Replaces removed with @replacement.
Review of Session 4

• @string_array = split /pattern/, $string

```perl
$string = "1-800-123-4567";
@number_parts = split /-/, $string;
print "$number_parts[0]";  # prints "1"
print "$number_parts[3]";  # prints "4567"
```
Review of Session 4

• The Foreach loop

```perl
foreach $element (@number_parts)
{
    print "$element
";
}
```
A. Modify the program from session 3 exercise 3 (random DNA sequence) to produce a random DNA sequence of 5 Mb (originally 4.1kb), store the sequence string in a variable and discard the rest of the program (the part printing it to STDERR).

```perl
for ($i=1; $i<=5_000_000; $i++)
```

B. Take the random DNA string obtained in step 1 and apply in silico restriction enzyme by cutting the DNA at each occurrence of the pattern of “ATGCAT”. The easiest way to do it is to use `split` function with ATGCAT as the splitting pattern, store the DNA fragments in an array.

```perl
@fragments = split /ATGCAT/, $seq;
```
C. Create a new array containing lengths of the strings from the array obtained in step 2 (length($str) function returns the length of a string $str). Unlike the real restriction enzyme, split function removes ATGCAT pattern, to correct for this you need to add 6 to each middle fragment, 1 to first and 5 to the last (simulating cutting A\{cut\}TGCAT).
$n=0;
foreach $frg (@fragments)
{
    if ($n==0)
    {
        $fraglen[$n] = length($frg) + 1;
    }
    elsif ($n==#fragments)
    {
        $fraglen[$n] = length($frg) + 5;
    }
    else
    {
        $fraglen[$n] = length($frg) + 6;
    }
    $n++;
}
D. Sort the lengths array. Remember that `sort` function by default sorts in string context (in alphabetical order i.e. 123 comes before 99), you need to provide sorting function to sort numerically:

```perl
sort { $a <=> $b } @array
```

Print out the sorted fragment lengths.

```perl
@fraglen = sort { $a <=> $b } @fraglen;
foreach $frag (@fraglen)
{
    print "$frag
";
}
```
# What is a Hash?

## Array

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>apple</td>
</tr>
<tr>
<td>1</td>
<td>banana</td>
</tr>
<tr>
<td>2</td>
<td>cranberry</td>
</tr>
<tr>
<td>3</td>
<td>daikon</td>
</tr>
<tr>
<td>4</td>
<td>eggplant</td>
</tr>
<tr>
<td>5</td>
<td>81</td>
</tr>
</tbody>
</table>

## Hash

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>red fruit</td>
<td>apple</td>
</tr>
<tr>
<td>yellow fruit</td>
<td>banana</td>
</tr>
<tr>
<td>red berry</td>
<td>cranberry</td>
</tr>
<tr>
<td>white tuber</td>
<td>daikon</td>
</tr>
<tr>
<td>purple veggie</td>
<td>eggplant</td>
</tr>
<tr>
<td>nine squared</td>
<td>81</td>
</tr>
</tbody>
</table>
What is a Hash?

- Data structure similar to an Array.
  - Use to be known as “Associative Arrays”
- Indexed by an arbitrary unique string, the Key
- Each Key points to an element, the Value
- Each Value is an arbitrary scalar
- Establish relationship between Key and Value
What is a Hash?

- **Keys** must be UNIQUE!
- **Keys** are strings!
- Only one **Value** per **Key**!

**THERE IS NO “ORDER”!**
Hash Syntax

• Declaring a variable to be a Hash, use “%”

```perl
%hash;
```

• Can initialize a hash using an Array

```perl
%hash = ('red fruit', 'apple', 'yellow fruit', 'banana', 'nine squared', 81);
```
Hash Syntax

• Accessing individual Hash element

```
$hash{ 'red fruit' };
```

• Equivalently

```
$key = 'red fruit';
$hash{ $key };
```
Hash Syntax

• Assigning individual Hash element

```perl
$hash{ 'purple veggie' } = 'eggplant';
```

• Can initialize a array using a Hash

```perl
$array = %hash;
```
The Big Arrow

• Also known as “The Fat Comma”
• A way to “spell” a comma
• Simplifies Hash Declaration
The Big Arrow

```perl
%hash = (
    'red fruit' => 'apple',
    'yellow fruit' => 'banana',
    'red berry' => 'cranberry',
    'white tuber' => 'daikon',
    'purple veggie' => 'eggplant',
    'nine squared' => 81,
);
```
Hash Functions

• keys(%hash): returns an Array of the Keys
  
  @hash_keys = keys (%hash);
  @hash_keys = keys %hash;

• values(%hash): returns an Array of the Values
  
  @hash_values = values (%hash);
  @hash_values = values %hash;
Hash Functions

• Order of elements is consistent between arrays returned for Keys and Values

• First element of Keys will be the key to the hash that returns the first element of Values
Hash Functions

• `exists($hash{$key})`: returns true if $key exists in the hash
  ```perl
  exists  ($hash{$key1});
  exists  $hash{$key1};
  ```

• `defined($hash{$key})`: returns true if $key has a defined Value in the hash
  ```perl
  defined  ($hash{$key1});
  defined  $hash{$key1};
  ```
Hash Functions

• `delete($hash{$key})`: deletes $key and associated Value from the hash
  
  ```perl
  $hash{ 'green fruit' } = 'kiwi';
  delete ($hash{ 'green fruit' });
  ```

• `reverse(%hash)`: returns Hash with Keys and Values swapped
  
  ```perl
  %reverse_hash = reverse (%hash);
  ```
Hash Functions

- `each(%hash)`: returns the next Key Value pair as a 2 element array

```perl
@pair = each (%hash);
print "$pair[0] = $pair[1]\n";
```
Hash Functions

• Lets use a while loop to go through the rest of the hash using the Each function

```perl
while (@pair = each (%hash))
{
    print "$pair[0] = $pair[1]\n";
}
```
Hash Foreach

• Just like for an array, you can use a foreach loop to look through a hash

```perl
foreach $key (keys %hash) {
    print "$key = $hash{$key}\n";
}
```
Hash Sort

• What if we want to order a hash? Sort the keys!

```perl
foreach $key (sort keys (%hash)) {
    print "$key = $hash{$key}\n";
}
```
Hash Sort

• More likely we’ll want to sort by value

```perl
foreach $key (sort { $hash{$a} <=> $hash{$b} } keys (%hash))
{
    print "\t$key = $hash{$key}\n";
}
```
Hash Nuances

- **Keys** are strings, perl will convert if otherwise

```perl
%hash = ('red fruit' => 'apple', 'yellow fruit' => 'banana', 'nine squared' => 81, 5/2 => '5 over 2', );

print "->$hash{ '5/2' }<-
"n;
print "$hash{ '2.5' }\n";
```
Hash Nuances

- **Keys** are unique, you will lose data if you try to assign more than one Value to a Key

```perl
$hash{ 'red fruit' } = 'cherry';

print "$hash{ 'red fruit' }\n";
```
Hash Nuances

• Declare a new **Key** and modify it in one line

```perl
$hash->{'test int'} += 1;
print "$hash->{'test int'}\n";

$hash->{'test string'} .= 'Hello World!';
print "$hash->{'test string'}\n";
```
Hash Nuances

• Reverse function can cause you to lose data

```perl
%hash = (
    'red fruit' => 'apple',
    'technology company' => 'apple',
);

%reverse_hash = reverse (%hash);
while (@pair = each (%reverse_hash))
{
    print "$pair[0] = $pair[1]\n";
}
```
The Environment Hash

- Perl runs in a certain environment
- Most cases this will be linux
- %ENV hash contains information about the environment that the perl program is running in
The Environment Hash

```perl
foreach $key (keys %ENV) {
    print "$key\n";
}

print "$ENV{ 'PATH' } \n";
```

• Now lets set a new environment variable and try to access it
Hash Example

• Lets redo part D of last week’s exercise with hashes

• List out the steps that we need to take to make bins to use in Excel to make a histogram
  – Create a hash where Keys are bins and values are bin counts
  – Go through the list of fragment lengths adding one to the correct bin
  – Print out results in order of bin
Hash Example

• Creating a the hash table with correct bin values
• Bin values are multiples of 2000 up to 100,000
$bin = 2000;

while ($bin <= 100000) {
    $bin_counts{$bin} = 0;
    $bin += 2000;
}
Hash Example

• Traverse through the fragment length array, adding to the correct bin

```perl
foreach $element (@fraglen)
{
    $this_bin = 2000 * int($element/2000) + 2000;
    $bin_counts{$this_bin}++;
}
```
Hash Example

• Print the results in order of bin

```perl
foreach $key (sort { $a <=> $b } keys %bin_counts) {
    print "$key\t$bin_counts{$key}\n";
}
```
Exercises

1. Modify the code from session 3 exercise 3 to generate a 9kb long random DNA sequence. Save this sequence to a variable.

2. Create a hash where the keys are unique sequences of 3 base pairs and the values are the counts of how often the key appeared in the randomly generated sequence. Print out/save to a file these keys and values.

- There are numerous ways to accomplish creating the hash from the string, a few hints:
  - Look at the substring function from Session 2
  - Modify the creation of the sequence string, look at the split function from Session 4, and the % (mod) operator
Exercises

• Bonus: Print out/save keys and values sorted by values in decreasing order

• Person with the most number (at least 3) of distinct methods of populating a hash from a string wins 50 FREE CBSU computing hours!