Introduction to Perl: Part I

Pat Morin

COMP 2405

Carleton UNIVERSITY

Canada's Capital University

Outline

- Literals
- Scalar variables
- File I/O
- Arrays
- Hashes
- Contexts
- Focus is on where Perl differs from Java and C
  - not comprehensive
- Read the Perl 5 Tutorial

Carleton UNIVERSITY
Canada's Capital University

Hello World

```
#!/usr/bin/perl

print("Hello World!\n");
```

Carleton UNIVERSITY
Canada's Capital University

Literals

- Perl has two kinds of basic literals:
  - strings: text strings
  - numbers: including *integers* and *decimals* (floating-point)

Canada's Capital University

String Literals

- String literals in Perl are similar to those in C or Java but can be specified in three different ways:
- Single quoted strings
  - Prints exactly what is contained in the single quotes
- Double quoted strings
  - Like in C or Java (with special escape codes) and variables can be used within the string
- Here docs
  - Like single or double-quoted strings, but for multiline strings

Carleton UNIVERSITY
Canada's Capital University

String Literals

```
print('This is a single-quoted string\n');

print("This is a double-quoted string\n");

print(<<ENDOFSTRING);
This is a heredoc string that spans
multiple lines, including carriage
returns.
ENDOFSTRING

print(<<'FINISH');
This is another heredoc string that
spans multiple lines. But this one
is treated like a single-quoted string
FINISH
```

Carleton
UNIVERSITY
Canada's Capital University

Number Literals

- Numbers are similar to C and can be specified as
  - Decimal (base 10), e.g., 47362
  - Octal (base 8), e.g., 04837
  - Hexadecimal (base 16), e.g., 0x38de
  - Floating-point, e.g., 2.8
  - Scientific notation, e.g., 2.9e12

```
print(47362); print('\n');
print(0437); print('\n');
print(0x38de); print('\n');
print(2.8); print('\n');
print(2.9e12); print('\n');
```

Carleton UNIVERSITY
Canada's Capital University

Data Types

- Perl 5 only distinguishes between two types of data
- *Scalar data* represents a single piece of data
  - literals
  - variables
- *List data* is an aggregation of scalar data
  - arrays
  - hashes (hash tables)

Canada's Capital University

Scalar Variables

- Scalar variables can hold
  - A string
  - A number
  - A reference
- By default variables are global, unless specified otherwise
- We declare variables as local using the `my` keyword
- When we use perl 5 strict, variables must be declared before they are used

Carleton UNIVERSITY
Canada's Capital University

Using Scalar Variables

- In Perl, scalar variables are prefixed with $
- Assignments are done as in most other programming languages
- The assignment operator returns the assigned value

Carleton UNIVERSITY
Canada's Capital University

```
my $aString = "Hello my name is Simon";

my $aSecondString = "and I love to do drawrings.";

my $thisNumber = 42;

my $a = my $b = $thisNumber;
```

Variable Substitution

- An extremely useful feature in Perl is *variable substitution* within strings
- This works with double-quoted strings
  - To avoid variable substitution, use single-quoted strings
- The substitution occurs at the time the string is evaluated (and can occur again)

```
my $name = "Huckleberry Finn";
my $age = 14;

print("His name was $name and his age was $age\n");
```

Carleton
UNIVERSITY
Canada's Capital University

Curly Braces

- Variable names can surrounded with curly braces
- This is sometimes helpful in string substitutions

```
$n = 4;

print("${n}th Edition\n");
```

Carleton UNIVERSITY
Canada's Capital University

Comparing Scalar Variables

- How to compare scalar variables depends on whether they are strings or numbers
- For numbers, we use <, >, <=, >=, and ==, just like in C or Java
- For strings, we use `lt`, `gt`, `le`, ge, eq to get (case-sensitive) lexicographic comparison
- Be careful: This is a common source of errors

```
if ($lastName le 'M') {
   print("First half of alphabet\n");
} else {
   print("Second half of alphabet\n");
}
```

Carleton UNIVERSITY
Canada's Capital University

Basic String Operations

- The `length()` function gets the length of a string
- The `substr()` function is used for extracting and replacing a substring from a string
  - `substr STRING, OFFSET`
  - `substr STRING, OFFSET, LENGTH`
  - `substr STRING, OFFSET, LENGTH, REPLACEMENT`

```
$string = "This is a test string.";
$len = length($string);
print(substr($string, 5));    # is a test string.
print(substr($string, 5, 2)); # is
print(substr($string, 8, 0, "not a "));
  # This is not a test string.
```

- **Note:** The last form is destructive!

Basic String Operations

- The . (dot) operator is used to concatenate strings

```
print("This string is" .
      " concatenated with this string\n");
```

Carleton UNIVERSITY
Canada's Capital University

Carleton UNIVERSITY
Canada's Capital University

Manipulating Text Files

- Files are opened with the open function and closed with the `close` function
- `open(filehandle, mode, filename)`
- Common modes are
  - Reading `"<"`, clobbering `">"`, and appending `">>"`
- open returns true on success and false on failure (and `$!` contains an error message)

```
open(my $ifp, "<", "infile.txt");

open(my $ofp, ">", "outfile.txt");

open(my $afp, ">>", "logfile.txt");
```

Example of open

```
if (!open(my $fp, "<", "infile.txt")) {
  print("Error opening file: $!\n");
  exit(-1);
}
...
close($fp);
```

Carleton UNIVERSITY

Canada's Capital University

```
open(my $fp, "<", "infile.txt")) ||
  die("Error opening file: $!\n");
...
close($fp);
```

Reading from a File

- The <> (diamond) operator is used to read a line from a file
- Returns true on success or false on end-of-file

```
# Open infile.txt and print its contents
my $fp;
open($fp, "<", "infile.txt") || die("Error: $!");
while (my $line = <$fp>) {
  print($line);
}
close($fp);
```

Carleton UNIVERSITY
Canada's Capital University

Writing to a File

- We can write to a file using the `print` command
- `print filehandle (list)`

```
open(my $lfp, ">>", "logfile.txt")
   || die("Error opening logfile: $!\n");
}

print $lfp ("Processed another transaction\n");

close($lfp);
```

Carleton UNIVERSITY
Canada's Capital University

Arrays and Lists

- Perl has arrays that are indexed starting at 0
- Array sizes do not have to specified in advance
  - Perl arrays grow and shrink dynamically (like Vectors in Java)
- Perl arrays are often frequently used like stacks and/or queues
- Perl arrays are also often used as parameter lists to subroutines (functions)

Canada's Capital University

Creating an Array

- Array variables are prefixed with @
- Arrays can be created and populated in different ways

```
@choices = ("yes", "no", "maybe");

#equivalent to
$choices[0] = "yes";
$choices[1] = "no";
$choices[2] = "maybe";
```

Carleton UNIVERSITY
Canada's Capital University

Nested Arrays

- Arrays can be nested
- But this doesn't result in an array of arrays!
- The arrays are *flattened* into a single array

```
@colors = (("bright red", "dark red"),
           ("bright yellow", "dark yellow"));

# equivalent to
@colors = ("bright red", "dark red",
           "bright yellow", "dark yellow");
```

Carleton UNIVERSITY
Canada's Capital University

Merging and Appending to Arrays

- When applied to arrays , (comma) is a merge operator

```
# Merge two arrays into one big array
@bigArray = (@smallArray1, @smallArray2);

# Add a new element to the end of myArray
@myArray = (@myArray, $myNewElement);

# Add a new element to the beginning of myArray
@myArray = ($myNewElement, @myArray);
```

Carleton UNIVERSITY
Canada's Capital University

Getting the Size of an Array

- We can get the size of an array *by converting the array to a scalar!*

```
$nColors = @colors; # conversion to scalar
```

- Or we can get the last index of the array

```
$lastIndex = $#colors;
$nColors = $lastIndex + 1;
```

Carleton UNIVERSITY Canada's Capital University

The Range Operator

- The range operator `..` generates an array of consecutive numbers
  - `@numbers = (100 .. 200);`
- The range must be increasing
- For a decreasing range, use the reverse function
  - `@numbers = reverse(100 .. 200);`

Carleton UNIVERSITY
Canada's Capital University

Array Access and Slices

- We use the [ ] operator to access the elements of an array
  - `$listOfNames[2] = "Mark Twain";`
  - `print("Name: $listOfNames[2]\n");`
- The [ ] operator also lets us take a *slice* of an array

```
@alphabet = ('0' .. '9', 'a' .. 'z', 'A' .. 'Z');

@lowercase = @alphabet[10 .. 35];

@zeroAndLowercase = @alphabet[0, 10 .. 35];
```

Carleton UNIVERSITY
Canada's Capital University

Printing the Contents of an Array

- The `print` function, like many functions, takes a list (array) of parameters
- If we give it an array, the `print` function will print the array items
- A special variable `$,` determines what is printed between the array (list) entries

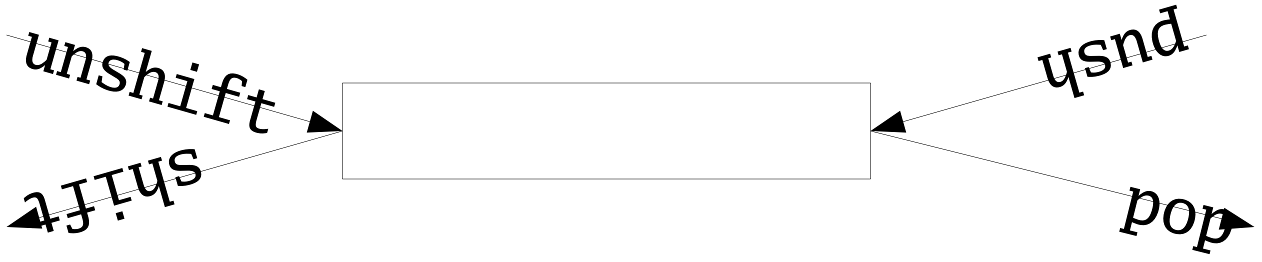
```
@colors = ("red", "green", "blue", "yellow");

$, = " ";
print(@colors, "\n");
```

Carleton UNIVERSITY
Canada's Capital University

Carleton UNIVERSITY
Canada's Capital University

Arrays as Deques

- Arrays can also be treated like stacks in which we push and pop from the end
  - `push` – add an element to the end
  - `pop` – remove an element from the end
- Or like stacks in which we push and pop from the front
  - `unshift` – add an element to the front
  - `shift` – remove an element from the front



Array Splicing

- The splice function can do all the above and more
  - `splice ARRAY, OFFSET`
  - `splice ARRAY, OFFSET, LENGTH`
  - `splice ARRAY, OFFSET, LENGTH, LIST`
- Starting at position `OFFSET`, remove `LENGTH` elements and replace them with `LIST`
  - If no `LIST` is provided then only the deletion is done
  - If no `LENGTH` is specified then all elements from `OFFSET` to the end of the list are removed
- More general than `push`, `pop`, `unshift`, `shift`, etc but harder to read

Carleton UNIVERSITY
Canada's Capital University

Other Array Functions

- `join` – concatenates a list of scalars into a single string
- `reverse` – reverse a list
- `map` – applies an operation to every element in a list and produces a new list containing the results of each operation
- `sort` – sorts a list (lexicographically by default)
- We will touch on `sort` and `map` again later

Canada's Capital University

Hashes

- Perl hashes are associative containers
- They associate a *key* with *data*
- It is very efficient to access the data for a specific key
- Similar to arrays, but we can use anything for indexes
- Hash names are prefixed with %

Canada's Capital University

Initializing a Hash

- A hash can be initialized using an array
- The array entries alternate key/value key/value ...

```
%grades = ('Peruvian', 9.5,
           'Columbian', 9,
           'Canadian', 6,
           'Mexican', 8);
```

- The => operator is (almost) identical to a comma, but easier to read

```
%grades = ('Peruvian' => 9.5,
           'Columbian' => 9,
           'Canadian' => 6,
           'Mexican'=> 8);
```

Carleton UNIVERSITY
Canada's Capital University

Accessing a Hash

- The values in a hash can be accessed by key
- This is the normal way in which hashes are used and is the most efficient
- If you frequently need to access the values some other way, maybe you shouldn't use a hash

Carleton
UNIVERSITY
Canada's Capital University

```
print("Before testing: Peruvian = ",
       $grades{'Peruvian'});

$grades{'Peruvian'} = 10;
print("After testing: Peruvian = ",
       $grades{'Peruvian'});
```

Adding and Removing Elements

- Elements can be added to a hash simply by assigning a value to them
- Elements can be deleted from a hash using the `delete` function
- To delete an entire hash, just assign it to be empty or use the `undef` function

```
$grades{'Brazillian'} = 8.6;  # add new pair
delete $grades{'Canadian'};   # delete pair
%grades = ();                 # clear hash
undef %grades;                # undefine hash
```

Carleton UNIVERSITY
Canada's Capital University

Testing a Hash

- To test if a key is in a hash, use the `exists` function
- To test if a key is in a hash *and it's value is defined*, use the `defined` function

```
if (defined($grades{$name})) {
  print("The Grade of $name is $grades{$name}\n");
}
```

Carleton UNIVERSITY
Canada's Capital University

Enumerating a Hash

- To get all the keys in a hash we use the keys function
- To get all the values in a hash we use the values function
- These functions return an array

```
for $k (keys(%grades)) {
   print("$k => $grades{$k}\n");
}
for $v (values(%grades)) {
   print("$v\n");
}
```

Carleton UNIVERSITY
Canada's Capital University

Contexts in Perl

- Perl is a *context-sensitive* language
- The meaning of a code fragment can depend on the context in which it appears
- This is most common on the right hand side of the assignment = operator
  - Recall: `$numEelements = @colors;`
  - This evaluates `@colors` in the *scalar context*
- We can force a scalar context using the scalar function
  - `print (scalar(@colors));`

Carleton UNIVERSITY
Canada's Capital University

A Context Example

- The following code creates an array
  - `@a = (35, 48, 56);`
- The following code assigns the value 56 to $a
  - `$a = (35, 48, 56);`
- In the array context, the comma acts as a separator for array values
- scalar context, the comma operator evaluates a sequence of expressions and returns the value of the last one

Carleton UNIVERSITY
Canada's Capital University

More Context

- Places where you expect a boolean (true/false) value are treated as scalar contexts
  - 0, the empty string "", and undefined values are treated as false
  - All other values are treated as true
- What does the following code do?

```
if (@colors) {
   # do something
}
```

Carleton UNIVERSITY
Canada's Capital University

Summary

- We have discussed
  - Literals
  - Scalar variables
  - File I/O
  - Arrays
  - Hashes
  - Context
- Chapters 1-3 in the Perl 5 Tutorial

Carleton UNIVERSITY
Canada's Capital University