PERL Constants

Mostly like C

```
12345
-12345
18.75
6.02E23
0b011011     (binary)
0377         (octal)
0xff         (hex)
0xdead_beef  (more hex)
4_294_967
"string"   (subject to variable and
            backslash interpretation)
'string'    (no interpretation)
(1,2,3)         list
```

PERL0010

University of Idaho

Variables – 3 types

Scalars – preceded by $

```
$name
$age = 26;
```

Arrays – preceded by @

```
@numbers = (1,2,3);
@date = (2, 17, 2006);
```

Hash (or associative array) – preceded by %

```
%fruit = ('apples', 3, 'oranges', 6);
%fruit = (
        apples  => 3,
        oranges => 6
          );
```

PERL0020

University of Idaho

Scalars - a Closer Look

No "type" in the usual sense

```
$a = "123";
$b = "456";
$c = $a + $b;
print "The value of c is $c\n";
print "$a + $b is $c\n";
```

PERL0025

University of Idaho

More Arrays

```
@a1 = (1);   #array of 1 element
@a2 = (1,2,3,4,5);  #5 elements
@a3 = (1..10);  #10 elements

print @a1," ",@a2," ",@a3, "\n";

print @a1[0]," ",@a2[1]," ",@a3[2],"\n";
```

Produces:

```
1 12345 12345678910
1 2 3
```

PERL0030

University of Idaho

Hashes - Associative Arrays

```
%fruit = ('apples', 3, 'oranges', 6);
print $fruit{'apples'};
```

Note the scalar "prefix" and braces.

Hashes are like non-positional arrays, where both the "subscript" (called the "key") and "value" are specified.

PERL0040

University of Idaho

More Hashes

```
%month = (January => 1,   February => 2,
          March => 3,     April => 4,
          May => 5,       June => 6,
          July => 7,      August => 8,
          September => 9, October => 10,
          November => 11, December => 12);


@monthname = keys(%month);
@mnumbers = values %month;
```

keys and values are built-in functions.

Perl0050

University of Idaho

Contexts

Perl "converts" values according to type of variable

```
$a = (2,4,6,8);
print $a "\n";    # prints 8

@arr = (1..10);
$b = @arr;     # scalar b is size of array

($sec,$min,$hour,$mday,$mon,$year,
           $wday,$yday,$isdst) = localtime();

@time = localtime();
$now = localtime();
```

PERL0060

University of Idaho

Scoping Rules

All variables are globally scoped unless 'my' is used

```
my $a = "foo";
if ($some_condition)
  {
    my $b = "bar";
    print $a    # prints "foo"
    print $b;   # prints "bar"
  }
print $a;       # prints "foo"
print $b;       # prints nothing
                #       (out of scope)
```

PERL0070

University of Idaho

PERL Control Statements

```
if ( condition )
  {
    ...
  } elsif ( other condition )
  {
    ...
  } else
  {
    ...
  }
```

Braces are always required, even with only one statement

PERL Control Statements

Alternate form of IF

```
if( $zippy ) {
   print "Yow!";
  }


print "Yow!" if $zippy;
```

PERL Control Statements

Negative form of IF

```
unless ( condition )
  {
     ...
  }
```

is equivalent to

```
if ( !condition )
  {
     ...
  }
```

PERL0100

University of Idaho

PERL Control Statements

Conditional Comparisons

Op	Numeric	string
Equal	==	eq
Not Equal	!=	ne
Less than	<	lt
Greater than	>	gt
Less or equal	<=	le
Greater or equal	>=	ge

```
if($str == 0 && $str ne "0") {
   warn "That doesn't look like a number\n";
  }
```

PERL0110

University of Idaho

PERL Control Statements

while / until

```
while ( condition )
  {
     ...
  }

until ( condition )
  {
     ...
  }
```

PERL Control Statements

for / foreach

```
for($i=0; $i < $max; $i++)
  {
     print "This value is $i\n";
  }


foreach $i (0..$max)
  {
     print "This value is $i\n";
  }
```

PERL Control Statements

Examples of foreach

```
foreach $n ( 1..15 ) {
   print $n, " ";
  }
print "\n";
```

```
foreach my $val ( keys %hashvar ) {
   print "The value of $val is $hashvar{$val}\n";
  }
```

```
foreach ( @array ) {
    print "This element is $_\n";
  }
```

```
foreach ( @array ) {
    print;
  }
```

PERL Built-In Variables

Not an exhaustive list!

`$_`	Default input/pattern-search
`$0`	Name of program itself
`$1,$2,etc.`	values of arguments
`$.`	input line number
`$$`	process id
`$!`	Error number returned from last subroutine call
`@ARGV`	array of command-line argument
`%ENV`	hash containing current environment

PERL Operator Precedence

Associativity	*Operator*	
Left	`->`	Method call, dereference
	`++ --`	Increment/decrement
Right	`**`	Exponentiation
Right	`! ~ \ + -`	Unary operators
Left	`=~ !~`	Matches, doesn't match
Left	`* / % x`	Mult, div, mod, string replicate
Left	`+ - .`	Add, Subtract, string concat
Left	`<< >>`	left/right bit shift
		File-test operators
	`< > <= >= lt gt le ge`	Comparisons
	`== != <=> eq ne cmp`	
Left	`&`	bit-and
Left	`| ^`	bit-or/xor
Left	`&&`	logical-AND
Left	`||`	Logical-OR
	`.. ...`	Range
Right	`?:`	ternary conditional
Right	`= op=`	Assignment
Left	`, =>`	comma, arrow
		List operators
Right	`not`	Logical NOT
Left	`and`	Logical AND
Left	`or`	Logical OR

Files in PERL

stdin, stdout and stderr are automatically open when Perl starts:

```
@line = <stdin>    # slurps the entire file!
foreach $i (@line)  {
   print "->", $i;
 }
```

```
while ($line = <stdin>)  {  # one line at a time
   print "->", $line;
 }
```

```
while (<stdin>)  {  # also one line at a time
   print stdout "->", $_;
 }
```

Opening Files

```
$INFILE = "input.txt";
open(INFILE);

open(INFILE, "input.txt") or die "Can't open file\n";


while(<INFILE>) {
  print;      # or print $_;
```

University of Idaho

Opening Files

```
open(OUTFILE, ">out.txt");   #create out.txt
open(OUTFILE, ">", "out.txt");

open(LOGFILE, ">>log.txt");  # append to log.txt
open(LOGFILE, ">>", log.txt);


print OUTFILE "This goes into a file\n";

print LOGFILE "This gets appended to the file\n";


close(OUTFILE);
close(LOGFILE);
```

University of Idaho

PERL Strings

Concatenate

```
$filename = $base . ".dat";
```

"Contains" or "Matches"

```
if($a =~ /foo/)  { ... }
```

Substitution

```
$a =~ s/foo/bar/; # replace foo with bar
 $a =~ s/foo/bar/g; # replace all
```

PERL0200

University of Idaho

PERL Regular Expressions

.	a single character
\s	whitespace character
\S	not whitespace
\d	digit (same as [0-9])
\D	not a digit (same as [^0-9])
\w	"word" (alphanumeric) char [a-zA-Z_0-9]
\W	not a word char
^	start of string
$	end of string
*	match zero or more of previous thing
+	match one or more of previous thing
?	match zero or one of previous thing
(3)	match exactly 3 of previous thing

PERL0210

University of Idaho

Regular Expression Examples

`/^\d+/`	string starts with one or more digits
`/^$/`	null string
`/(\d\s)(3)/`	three digits, each followed by whitespace
`/(a.)+/`	every other char is letter a

```
warn "has nondigits" if /\D/;
warn "not an integer" unless /^[+-]?\d+$/
```

```
while (<>)  {
   next if /^$/;
   print;
 }
```

```
LINE: while(<STDIN>)  {
         next LINE if /^#/;
          #do something with line
       }
```

PERL Subroutines

```
sub max {
    my $maxval = @_[0];
    foreach $val (@_)  {
       $maxval = $val if $val > $maxval;
    }
   return $maxval;
}
```

- Subroutines are defined with 'sub' keyword
- No formal arguments: arguments passed in array @_
- Two ways to call:

```
max($a, $b, $c);
max $a $b $c;  #only if defined before call!
```

PERL Subroutine Examples

```
sub max;    # prototype

sub max {
   my $maxval = $_[0];
   foreach $i (1..$#_)  {
      $maxval = $_[$i] if $_[$i] > $maxval;
   }
  return $maxval;
}


sub max {
   my $max = shift(@_);
   foreach my $val (@_)  {
      $max = $val if $val > $max;
   }
    return $max;
}
```

PERL "Built-In" Functions

File test Functions – return "true" if file:

`-r file`	*is readable*
`-w file`	*is writeable*
`-x file`	*is executable*
`-o file`	*is owned by user*
`-e file`	*exists*
`-z file`	*is zero bytes in length*
`-f file`	*is a regular file*
`-d file`	*is a directory*
`-l file`	*is a symlink*

PERL "Built-In" Functions - Part 2

Exit Functions

`exit` *`val`*	*return to shell, returning val*
`die` *`message`*	*print message and exit, returning $!*
`warn` *`message`*	*just print message*

Array Functions

`shift` *`@arr`*	*return first element, shift others down*

Hash Functions

`keys` *`%hash`*	*return array of all key values*
`values` *`%hash`*	*return array of all values of keys*

PERL0260

University of Idaho

PERL "Built-In" Functions - Part 3

String Functions

`chomp` *`string`*
Remove input separator ($/, usually newline) from string

`index` *`@str,substr`*
Return position of substr within array str

`join` *`char, list`*
Return single string consisting of list separated by char

`length` *`string`*
Return no of characters in string

`q/`*`string`*`/`
`q//`*`string`*`/`
Alternate form of 'string' and "string"

`split` *`/pattern/,string`*
Return list of substrings within string separated by pattern

PERL0270

University of Idaho