Scalar data

Scalar data – 1/3

- A variable is a modifiable memory location used to keep info for later use
- **Scalar means one**
- **Scalar variable can store only one data item at a time**

Scalar data – 2/3

- Three rules
  - Scalar variable can contain only one item at a time
  - Can contain either numeric or string data
  - Based on context, can determine whether it contains string or numeric data

Scalar data – 3/3

- **The operators, type of data and variable itself determine the context**
- Operators like **==** and **eq** determine the data will be treated as **numeric** or **string** value

Scalar variable names – 1/2

- **All scalar variable names are preceded by $**

- Rules
  - Names cannot contain spaces
  - May contain any printable char but special rules apply to nonalphanumeric chars
  - **Variables that begin with underscore or alphabet → any practical length**
  - **Characters that follow variable names that begin with an underscore or alphabet → may be digits, underscores, and alphabet**

Scalar variable names – 2/2

- **If variable name begins with digit - contain only digits**
- **If variable name begins with nonalphanumeric char – only be one char long**

- **Literal**
  - Constant value in the code that cannot be modified
  - Eg. 10 is a numeric literal, John is a string literal

Strings and character data

- **Anything surrounded by double quotation marks is considered a string**
- **Single char in single quotation mark is a character**

```
$name="Eric Hermann";
$gender='M';
```

Quotation marks

- Different styles of quotation marks
  - Double
  - Single
  - Back
- **They always come in pairs**
- **Use double and single to define string literals**

Single quotes – 1/2

- **Tells interpreter to stop interpreting any char that follows the single quote**
- This effect continues until it finds the next single quote
- **All chars between the pair of single quotes are treated as single string literal**

Single quotes – 2/2

```
#!/usr/bin/local/bin/perl
$name='Eric Hermann';
print 'The name is $name \n";
print "The names is $name \n";
```

output

The name is $name \n

The name is Eric Hermann

Double quotes and variable interpolation

- With double quotes, **Perl looks at each char of the string literal to see if it happens to have a special meaning**
- This is called **variable interpolation**
- **Double quotes allow variable interpolation**

Back quotes

- The backward quotation mark (`) tells Perl to **interpret the string between the quotation marks as an operating system command**

```
$name = 'Eric Hermann';
$d = `dir/w`;

print "Scalar context Directory listing \n $d\n";
```

Quote operators – 1/5

- Four operators
  - **q**
  - **qq**
  - **qx**
  - **qw**
- Each of these **works on any characters between delimiters**
- Delimiter for a quote operator is the first nonalphanumeric char that follows the operator

Quote operators – 2/5

- **Space does not count as delimiter**
- **Opening and closing delimiter should be same**

Quote operators – 3/5

```
$single= q("This is a test." 'He exclaimed' "Quotes don't count");
print "$single\n";

$a=10;
$b=20;
$sum=$a+$b;

$double=qq[double quotes allow variable interpolation. Here, sum of $a and $b is $sum.];

print "$double\n";
```

Quote operators – 4/5

```
$dir=qx!dir/w!;
print “qx is like back quote. Directory
          is $dir\n\n”;

$total=10;
@words=qw(Each of the $total words is
placed into array words);

print “No of words are $#words.\n\n”;
```

Quote operators – 5/5

- Rules for string literals
  - **Single quoted strings** are delimited by paired forward quotation marks or q operator. **Not interpolated**
  - **Double quoted strings** are delimited by paired double quotation marks or **qq** operator. **Interpolated**
  - **Back quoted strings** are delimited by paired backward quotation marks or **qx** operator. Treated as **system commands**
  - The **qw** operator **returns a list of non interpolated words**

Numeric literals

- **Numeric formats**

Type	Notation	Example
Integer	NN	12
Floating pt	NN.NN	123.23
Scientific	NN.NNENN	24.02E-5
Big number	NN_NNN_NNN	6_000_000
Hex	OxNNNN	OxFFD2
Octal	ONNN	O233

Fixed point number solutions – 1/3

- **Perl does not have any in built way to define rounding**

- Three ways we can deal
  - Use **sprintf()** to round result
  - Use **regular expression** to truncate result
  - Create **own rounding function** to generate rounded value that we can control

Fixed point number solutions – 2/3

```
$total=2.333+3.253+3.433;
print “value is $total\n”;
#use sprintf fn
$roundvalue=sprintf “%0.2f”, $total;
print “rounded value is $roundvalue\n”;
#use regular exp
$total =~ /(\d*\.\d\d)/;
$trunvalue = $1;
print “Truncate value is $trunvalue\n”;
```

Fixed point number solutions – 3/3

Output

value is 9.019

rounded value is 9.02

Truncate value is 9.01

sprintf() – 1/2

- Format data for printing and other display purposes
- Stands for **string print format**
- **Rounds up** to format requested
- Takes **two input parameters**
- 1st parameter → data format parameter
- 2nd parameter → data needs to be formatted

$roundvalue = sprintf “%0.2f”, $total;

sprintf() – 2/2

- Parameter tells Perl to format data as follows
  - The **f** tells the format should be **floating pt**
  - **.2** tells that **two digits after decimal pt**
  - **0** before decimal pt tells Perl to fill in any missing digits with zeros. This is called **zero fill**. Eg. 2 becomes 2.00

Regular expression – 1/2

$total =~ /(\d*\.\d\d)/;

- Regular expression consists of the \d, * and . between parentheses
- Tells Perl to **search the variable on left side of pattern binding operator (=~**) for any number of digits (\d*), followed by a period (.), followed by two digits (\d\d)
- If it founds, it saves it to a back **reference var.** it is named **$1**

Regular expression – 2/2

- Back reference variable contains chars of input string ($total) that match corresponding portion of the regular expression surrounded by parentheses
- The back reference variable is saved into the scalar variable **$1runvalue**

Boolean values

- Three ways var can be false
  - Value is zero
  - Value is null
  - Variable is undefined
- Everything else evaluates to true

Variables and namespaces – 1/2

- **A namespace is a long list or table of all the names that Perl interpreter must keep track of**
- One of those is **main::**
- Every Perl program has **main:: namespace** that **contains all variable names that are part of main package**
- When we declare variable it becomes part of namespace table

Variables and namespaces – 2/2

- **This namespace is also called symbol table**
- **Declared variable** has a place in the table but **does not have any contents or data** associated with it
- **Declared value** has **null value not zero**
- **Defined variable** has been assigned **some value**, even if that value is zero

my and local

- **Declare a variable without defining it by using my or local**
- A variable **declared but not initialized** is set to **null**

```
my $a;
local $a;

my ($a, $b, $c);
local ($a, $b, $c);
```