4. Perl

4.1 Scalar data type

A scalar is the simplest kind of data that Perl manipulates. A scalar is either a number (like 4. 3.25e20) or string of characters (like hello).

```
#!/usr/local/bin/perl  -w
# Assign the variables.
my $name="Gizmo";
my $age=3;
my $height=4 . 5;

# Print out the contents of each variable.
print "Original Name        : $name\n";
print "Original Age         : $age\n";
print "Original Height      : $height \n";

# Perform 'incorrect' operations.
my $height = $height . 5;
my $name = $name +  0.9;
# Print out the contents of each variable.
print "---\n";
print "Altered Name         : $name\n";
print "Altered Height       : $height\n";
```

4.2 Arrays

Perl' s arrays can be uses as a simple list, a stack, or even the skeleton of a complex data structure.

4.2.1 Using arrays as an indexed list

The following example demonstrates directly indexing an array's contents:

```
#!/usr/local/bin/perl  -w
# Define variables.
my @months = qw (JUNK Jan Feb March April  May June July Aug Sept Oct Nov
Dec);
my $x = 0;

# Print out the contents of the array.
for ($x=0; $x <= $#months; $x++)
{
      print "Index[$x] = $months[$x]\n";
}
```

4.3 Associative arrays

Associative arrays are arrays that are indexed by string value instead of by integer index value.

4.3.1 Extracting information from an associative array

The following script lists the contents of an associative array:

```
#1/usr/local/bin/perl -w
# Define variables.
my %myList = ( "XXX" => 111, “yyy" => 222, “ZZZ" => 333) ;
my $key1;
# Print out the contents of the hash.
for $key1 (keys %myList)
{
    print "Key: $key1 Value: $myList{$key1} \n";
}
```

If you're only interested in the values of the associative array, then you could use the values function instead.

The following script lists the contents of an associative array using the values function.

```
#!/usr/local/bin/perl -w
# Define variables.
my %myList = (“XXX" => 111, “yyy" => 222, “ZZZ" => 333);
my $value1;
# Print out the contents of the hash.
for $value1 (values %myList)
{
    print "Value: $value1 \n";
}
```

4.4 References

A reference is more a generic entity that can point on any given data type, native or generated.

4.4.1 Scalar reference

A scalar reference is created by using the backslash operator on an existing scalar variable.
The following example creates a reference to a scalar, then prints out the contents of the reference.

```
 #!/usr/local/bin/perl -w
 # Create the scalar variable
 my $scalarVariable = "Gizmo was here.";
 # Create the scalar reference
 my $scalarRef = \$scalarVariable;
 # Print out the contents of the scalar variable
 print "Var: $scalarVariable\n";

 # Print out the contents of the scalar reference.
 # Note the double  $$
 print "Ref:  " . $$scalarRef  . "\n";
```

4.4.2 Array references

The following script creates a reference to an array variable named @months, then prints out the contents of the array reference.

```
#! /usr/local/bin/perl -w
# Create the array
my @months = qw (Jan Feb March April May June July Aug Sept Oct Nov Dec);
# Create the array reference.
my $arrayRef = \@months;
my $month;
# Print out the contents of the array reference.
for $month (@$arrayRef)
{
     print "Month: $month \n";
}
```

4.4.3 Hash references

The following script creates a reference to a hash variable named %who and then prints out the contents of the hash reference:

```
#!/usr/local/bin/perl -w
# Create the associative array
my %who = ('Name'  =>  'Gizmo',  'Age' => 3,  'Height'  =>  '10 cm', 'Weight'  =>
'10 gm' );
# Create the hash reference
my $hashRef = \%who;
my $key;
# Print out the contents of the associative array.
for $key (sort keys %$hashRef)
{
      my $value = $$hashRef{$key};
      printf "Key: %10s  Value: %-40s\n",  $key,  $value;
}
```

4.4.4 Code references

The following script creates a reference to a subroutine named callBack and dereferences the reference to call the subroutine.

```
#!/usr/local/bin/perl  -w
# Define the callback function.
sub callBack
{
      my $mesg = shift;
      print "$mesg\n";
}
# Create the code reference
my $codeRef =\&callBack;
# Call the callback function with different parameters.
&$codeRef ("Hi  Mike!");
&$codeRef ("How Are You?") ;
```