Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Perl

Lesson Objectives

- After completing this lesson you should be able to understand
  - Arrays
  - Associative Arrays
  - Subroutines in Perl

Copyright © Capgemini 2016. All Rights Reserved

Arrays

- It is a variable that stores an ordered list of scalar values
- Array variable is preceded by @ symbol
- To refer to single element symbol $ is used with variable name followed by index of the element in square brackets
- They can shrink and grow dynamically as elements are added or deleted in the list

Copyright © Capgemini 2016. All Rights Reserved

Array - Syntax

- Array creation
  - my @products=(‘Product1’,’Product2’,’Product3’)
  - my @age=(10,20,30)
  - my @country=(1,"India",2,"USA",3,"UK",4);
- Access elements of array
  - print(“\n$products[0]”)
  - print(“\n$age[0]”)
- Array creation using qw operator
  - my @month=qw/Jan Feb Mar Apr/
- Access elements of array string
  - print(“\n$month[0]”)

Copyright © Capgemini 2016. All Rights Reserved

Array - Size

- Array creation
  - my @products=(‘Product1’,’Product2’,’Product3’)
- Size of array products is
  - my $size=@products
  - print "\nSize of array products is $size";

Copyright © Capgemini 2016. All Rights Reserved

Array – Creating copy of array

- Array creation
  - my @products=(‘Product1’,’Product2’,’Product3’)
- Assigning array product to dupproducts
  - my @dupproducts=@products
  - my $size=@dupproducts
  - print "\nSize of array dupproducts is $size";

Copyright © Capgemini 2016. All Rights Reserved

Sequential Number Arrays using range operator (..)

- Array creation
  - my @numbers=(1..10)
  - my @characters=(‘a’..’z’)
- Print elements of array
  - print @numbers #will print all numbers from 1 to 10
  - print @characters #will print all characters from a to z

Copyright © Capgemini 2016. All Rights Reserved

Iterating the array

- foreach loop
- while loop
- for loop

Copyright © Capgemini 2016. All Rights Reserved

Iterating Array

```
my @numbers = qw ( one two three four);
print "The Array contains : \n";

foreach my $number (@numbers){
        print "$number\n";
}
```


Copyright © Capgemini 2016. All Rights Reserved

Iterating Array

```
my @numbers = qw ( one two three four);
print "The Array contains : \n";
my $n = 0;

while ($numbers[$n]){
        print "\n $numbers[$n] \n";
        $n++;
}
```


Copyright © Capgemini 2016. All Rights Reserved

Iterating Array

```
my @numbers = qw ( one two three four);
print "The Array contains : \n";

for my $number (@numbers){
        print "\n $number \n";
}
```


Copyright © Capgemini 2016. All Rights Reserved

Array Functions

S.No.	Types & Description
1	**push @ARRAY, LIST** Pushes the values of the list onto the end of the array.
2	**pop @ARRAY** Pops off and returns the last value of the array.
3	**shift @ARRAY** Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down.
4	**unshift @ARRAY, LIST** Prepends list to the front of the array, and returns the number of elements in the new array.

Copyright © Capgemini 2016. All Rights Reserved

Array Functions

```
my @num=(1,2,3);
print "\nnum contains @num";
push(@num,(4,5,6));  #apppends the list to the number array
print "\nnum contains @num";
shift(@num); #removes the first element in the number array
print "\nnum contains @num";
unshift(@num,(9,10,11));  #prepends the number array with
9,10,11
print "\nnum contains @num";
pop(@num); #removes one element from last of the array
print "\nnum contains @num";
```


Copyright © Capgemini 2016. All Rights Reserved

Slicing Arrays

- Create new array from existing
  - my @num=(1,2,3,4,5,6)
  - my @slicednum=@num[2,3,4] # here [2,3,4] are subscript of num array
  - print("\nslicenum contains @slicednum") #will print 3 4 5

Copyright © Capgemini 2016. All Rights Reserved

Replacing elements in the Array

- Array- @originalnumbers
  - my @originalnumbers=(1..10);
  - print "\nOriginal numbers contains @originalnumbers";
  - splice(@originalnumbers,8,5,19..23);
    - #first parameter is the name of the array
    - #second parameter is the position of subscript from where the number replacement would start
    - #third parameter is the total number to be replaced
    - #fourth parameter is the new range
  - print "\nspliced array :@originalnumbers "; #will replace numbers 9 and 10 with the new numbers ie 19 to 23

Copyright © Capgemini 2016. All Rights Reserved

Sorting Arrays

- Create new array from existing and sort the array
  - my @foods=qw(pizza steak chicken burgers)
  - foods=sort(@foods)
  - print(“\n After sorting foods array contains: @foods”)

Copyright © Capgemini 2016. All Rights Reserved

Special Variable : $[

- To change the index of array variable
- By default $[will always point to zero subscript
- To reset the first index of the array
  - $[= 1

```
# define an array
@foods = qw(pizza steak chicken burgers);
print "Foods: @foods\n"; # Let's reset first index of all the arrays.
$[ = 1;
print "Food at \@foods[1]: $foods[1]\n";
print "Food at \@foods[2]: $foods[2]\n";
```


Copyright © Capgemini 2016. All Rights Reserved

Merging Arrays

- To join two arrays

```
my @odd = (1,3,5);
my @even = (2, 4, 6);
my @numbers = (@odd, @even);
print "numbers = @numbers\n";
```


Copyright © Capgemini 2016. All Rights Reserved

Hash Variable or Associative Array

- It is a set of key value pair
- Hash variables are preceded by % symbol
- Keys are always unique, but data or value can be duplicate.
- Order of data is not guaranteed as in case of arrays.
- Hash variables can be created by one of the following ways
  - my %account=('Ravi',2000,'Sunil',3000)
  - print "\n$account{'Ravi'}

    OR
  - my %account=('Ravi' => 2000,'Sunil'=>3000)
  - print "\n$account{'Ravi'}

Copyright © Capgemini 2016. All Rights Reserved

Slicing Hash Variable

- my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
- print("\n Account num : $accounts{'account no'}");

- my @extractedaccount=@accounts{'account no‘,'balance'};
- print("\n Extracted Account : @extractedaccount"); #this will not have type key

Copyright © Capgemini 2016. All Rights Reserved

Extract Keys and Values from Hash Variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
print("\n Account num : $accounts{'account no'}");
my @acckeys= keys %accounts;
my @accvalues=values  %accounts;

print "\nKeys : $acckeys[0]";
print "\nKeys : $acckeys[1]";
print "\nKeys : $acckeys[2]";

print "\nValues: $accvalues[0]";
print "\nValues: $accvalues[1]";
print "\nValues: $accvalues[2]";
```


Copyright © Capgemini 2016. All Rights Reserved

Iterate Hash Variable – for construct

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');

foreach(keys %accounts){
        print(“\n$accounts{$_}”)
}

OR

for(keys %accounts){
        print(“\n$accounts{$_}”)
}
```


Copyright © Capgemini 2016. All Rights Reserved

Iterate Hash variable – each construct

- The **each** construct returns a two element lists:
  - One list of key
  - One list of its value
- Every time **each** is called in the **while** loop, it returns another key/value pair (that is, the next key/value pair in the iteration).

Copyright © Capgemini 2016. All Rights Reserved

Iterate Hash Variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
while ((my $key, my $value) = each(%accounts)){
    print $key.",". $value."\n";
}
```


Copyright © Capgemini 2016. All Rights Reserved

Find size of Hash variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
$size=keys %accounts;
print("\nsize of accounts hash is $size");
```


Copyright © Capgemini 2016. All Rights Reserved

Functions used in Associative Array

Copyright © Capgemini 2016. All Rights Reserved

Functions

- exists
- delete
- undef

Copyright © Capgemini 2016. All Rights Reserved

Checking for existence of Keys in Hash Variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');

if( exists($accounts{'account no'}) ){
        print "\nAccount $accounts{'account no'} has $accounts{'balance'}
balance";
}else{
        print "\naccount no does not exist";
}
```


Copyright © Capgemini 2016. All Rights Reserved

Add and Remove key-value from Hash variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
my $size=keys %accounts;
print("\nsize of accounts hash is $size"); #will print 3

$accounts{'intrate'}=4.5; #Adding new key named intrate
$size=keys %accounts;
print("\nsize of accounts hash is $size"); #will print 4

delete $accounts{'intrate'}; #Removing the key intrate
$size=keys %accounts;
print("\nsize of accounts hash is $size"); #will print 3
```


Copyright © Capgemini 2016. All Rights Reserved

undef function

- This function deletes an associative array

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
undef(%accounts);

if(%accounts){
        print "Defined";
}else{
        print "Not Defined";
}
```


Copyright © Capgemini 2016. All Rights Reserved

Subroutine

Copyright © Capgemini 2016. All Rights Reserved

subroutine

- Subroutine is a name given to a section of code
- It is similar to a user-defined function in C
- It is mainly created to:
  - Reuse code
  - Manage code
- It can be placed anywhere in the program (at the beginning or the end)

Copyright © Capgemini 2016. All Rights Reserved

subroutine

- There are three sections in the declaration of the subroutine
  - The sub keyword
  - Name of the subroutine
  - Block of code
- The @_ list array variable is the special variable in PERL that gets created for every subroutine and holds the arguments passed to the subroutine. Thus the first argument to the function is in $_[0], the second is in $_[1], and so on.

Copyright © Capgemini 2016. All Rights Reserved

subroutine

- syntax

```
sub subname
{
        code block
}
```


Copyright © Capgemini 2016. All Rights Reserved

subroutine

- Code snippet

```
sub fun
{
        print “Hello World";
}
fun;
```

```
sub greet{
        @names = @_;
        foreach my $name (@names)
        {
                print “Hello , $name!\n";
        }
}
print greet( “john", “Harry", “Maggie");
```


Copyright © Capgemini 2016. All Rights Reserved

Reference to a subroutine

```
sub displayhash{
   my (%hash) = @_;

   foreach my $item (%hash){
      print "Item : $item\n";
   }
}

my %hash = ('name' => ‘Rahul', 'age' => 29);

# Create a reference to above function.
my $cref = \& displayhash;

# Function call using reference.
&$cref(%hash);
```


Copyright © Capgemini 2016. All Rights Reserved

Returning value from a subroutine

```
sub addnumbers{
	my @numbers=@_;
	my $n=scalar(@_); #get total number of arguments passed
	my $sum=0;
	foreach my $num (@numbers){
		$sum=$sum+$num;
	}
	print ("\n Sum of numbers is $sum");
	my $average=$sum/$n;
	return $average;
}

my $avg1=addnumbers(1,2,3);
print "\nAverage is $avg1";
```


Copyright © Capgemini 2016. All Rights Reserved

References in Perl

Copyright © Capgemini 2016. All Rights Reserved

References

- References are the addresses of data items in memory.
- These are scalar values.
- Extracting the value referred to by the reference variable is called dereferencing.

Copyright © Capgemini 2016. All Rights Reserved

References

- The backslash(\) operator is used to create a reference.
  - Creating references to a named data variable:
    - Reference to a scalar variable:

      For example,

```
$scalar =10;
$scalar_ref=\$scalar;
```

    - Reference to an array variable:

      For example,

```
@array =(1,2,3,4,5);
$array_ref=\@array
```


Copyright © Capgemini 2016. All Rights Reserved

References

- Reference to a hash variable:
  For example,

```
%hash =(Java => 1000 , PERL => 200 , C => 1500 );
 $hash_ref=\%hash;
```


Copyright © Capgemini 2016. All Rights Reserved

Dereference

- Dereferencing returns the value from a reference point to the location
- Dereferencing can be done in two ways
  - Prefix
  - Infix
- Prefix
  - To dereference a reference simply use $, @ or % as prefix of the reference variable depending on whether the reference is pointing to a scalar, array, or hash
- Inflix
  - Use -> operator

Copyright © Capgemini 2016. All Rights Reserved

Dereference- Prefix

- Referencing & Dereferencing scalar variable

```
my $data=10;
my $dataref=\$data;
print "\ndataref contains- value :: $$dataref\n";
```

- Referencing & Dereferencing array

```
my @month=('Jan','Feb','Mar');
my $monthref=\@month;
print "\nMonth array contains:: @$monthref\n";
```


Copyright © Capgemini 2016. All Rights Reserved

Dereference- Prefix

- Referencing & Dereferencing hash variable

```
my %accounts=('account no'=>'A001','balance'=>456.56,'type'=>'savings');
my $accountref=\%accounts;
print "\nhash account contains =",%$accountref,"\n";
```


Copyright © Capgemini 2016. All Rights Reserved

Reference and Dereferences

Ref Var Name	Variable Name	Value
$num_ref	$num	100
$scores_ref	@scores	10,20,30
$skill_ref	%skillhash	basic=>2,medium=>3

Access Values from variables

Scalar variable	Array Variable	Hash Variable
$$num_ref	@$score_ref	$skill_ref->{‘basic’}

Copyright © Capgemini 2016. All Rights Reserved

Assignment

- Create a reference which points to a hash variable (which stores name of the months). Iterate through the reference to display the month names

Copyright © Capgemini 2016. All Rights Reserved

Summary

- In this lesson you learnt
  - Arrays
  - Associative Arrays
  - References

Copyright © Capgemini 2016. All Rights Reserved