LOOPS

Loops

- Special type of branching statement

- Types of loops in Perl
  - while
  - until
  - for
  - foreach

while loop

- Executes while cond is still true
- Tests for true cond expression
- Syntax

```
while (cond)
{
            stmts;
}
```

until loop

- Executes until cond is true
- Test for false cond expression
- Syntax

```
until (cond)
{
        stmts;
}
```

do statement

- Not a loop, but used with while/ until loops
- Allows to execute block of stmts in a loop at least once
- Syntax

```
do
{
        stmts;
} while(cond);

do
{
        stmts;
} until(cond);
```

Example

```
while($ct <3)
{
     print "Inside while loop count is $ct\n";
     $ct++;
}
print "count increment one more time in while
loop, now count is $ct\n";
until ($ct > 6)
{
     $ct++;
     print "Inside until loop count is $ct\n";
}
print "count is not increment in until loop, so
count is $ct\n";
```

Example (cont..)

```
do
{
        print “do stmt is always executed at
                least once. Count is $ct\n”;
        $ct++;
} until ($ct >6);
do
{
        print “do stmt is always executed at
                least once. Count is $ct\n”;
        $ct++;
} while ($ct < 3);
```

```
do
        {
                print “do stmt can act as loop.
                Count is $ct\n”;
                $ct++;
        } until ($ct <10);
```

Example (cont..)

- Result

```
Inside while loop count is
Inside while loop count is 1
Inside while loop count is 2
count increment one more time in while loop, now count is 3
Inside until loop count is 4
Inside until loop count is 5
Inside until loop count is 6
Inside until loop count is 7
count is not increment in until loop, so count is 7
do stmt is always executed at least once. Count is 7
do stmt is always executed at least once. Count is 7
do stmt can act as loop. Count is 8
do stmt can act as loop. Count is 9
```

for loop – 1/2

- Iterate discrete no of times
- Syntax

```
for (initialization; condition; increment)
{
        //stmts;
}
```

- Eg

```
for ($i=0; $i < 10; $i++)
{
      print "$i \n";
}
```

for loop – 2/2

- Infinite loop

```
for (;;)
{
        //stmts;
}
```

foreach loop – 1/4

- Used for processing arrays and hashes
- Syntax

```
foreach $var (list)
{
        //stmts;
}
```

- $var is optional

```
foreach (list)
{
        //stmts;
}
```

foreach loop – 2

- Eg
- Print each element explicitly

```
foreach $i (1..10)
{
        print “$i \n”;
}
```

- Print each element implicitly

```
foreach (1..10)
{
        print;
}
```

foreach loop – 3/4

- When $var is omitted, the special default variable $_ is assigned the value of each element in the list

- The list variable’s scope is local to foreach stmts

- Does not modify variables of same name whose scope is exterior to foreach block of stmts

foreach loop – 4/4

- Eg

```
$n = 30;
foreach $n (1..10)
{
        print "$n \t";
}
print "$n";
```

- Result

```
1  2  3  4  5  6  7  8  9  10  30
```

Array processing with foreach loop – 1/2

- Syntax

```
        foreach $arrayelement (@array)
        {
                //stmts;
        }
```

- If $arrayelement is omitted, default special variable $_ will be set to value of each element of array

Array processing with foreach loop – 2

- Eg

```
@digits = (1..10);
foreach $no (@digits)
{
        print "$no \t";
        $no = $no + 10;
}
print "\n @digits";
```

- When we change the $arrayelement inside foreach loop, it is also changing the corresponding value in array

Hash processing with foreach loop

- Syntax

```
foreach $index (keys %hashname)
{
        print "$hashname{$index}\n";
}

foreach $value (values %hashname)
{
        print "$value \n";
}
```

last

- Jumps out of stmt block
- Syntax

```
last;

or,

last LABEL;
```

- With LABEL, it exits block of stmts associated with LABEL;
- w/o label, it exits current block of stmts

next

- Works only with loop block of stmts
- Skip rest of stmt block and continues with next iteration of loop
- Syntax

```
next;
next LABEL;
```

- w/o label, returns execution to enclosing block of stmts – for, foreach, while, until
- With LABEL, exits to loop associated with accompanying label

Redo – 1/2

- Restart the stmt block
- Not often use
- Syntax

```
redo;
redo LABEL;
```

- w/o label, redo jumps to 1st stmt of enclosing block of stmts
- With label, redo jumps to 1st stmt of block of stmts associated with label

Redo – 2/2

- Works within any enclosing block of stmts
- Creates its own loop syntax

```
{
        block of stmts;
        redo if cond exp;
}
```

- This block executes like while/until loops except it execute at least once before encountering redo stmt

Standard file handles

- STDIN
  - Reads program input.
  - Typically this is the computer's keyboard.
- STDOUT
  - Displays program output.
  - This is usually the computer's monitor.
- STDERR
  - Displays program errors.
  - Most of the time, it is equivalent to STDOUT, which means the error messages will be displayed on the computer's monitor.

Using STDIN – 1/2

- Read from Standard Input Until an End-of-file Character Is Found

```
while (<STDIN>)
{
        print();
}
```

- The <> characters, when used together, are called the *diamond* operator.
- They tell Perl to read a line of input from the file handle inside the operators.

Using STDIN – 2/2

- The diamond operator assigned the value of the input string to $_
- Then, the print() function was called with no parameters, which tells print() to use $_ as the default parameter

```
while ($inputLine = <STDIN>)
{
      print($inputLine);
}
```

- When we pressed Ctrl+Z or Ctrl+D, we told Perl that the input file was finished.