

Perl:  
-Variables  
-Variables Especiales  
-Operadores

## Introducción

- Perl se considera como una mezcla entre C y shell
- Es un lenguaje potente en manipulación de texto, programas CGI y tareas de administración de sistemas
- Lo que se puede hacer en C se puede hacer en Perl y probablemente mucho más fácil
- Perl funciona independiente de la plataforma
- Es un lenguaje muy útil para desarrollar aplicaciones en forma rápida, con manejo de archivos y acceso a las funciones del sistema

## Introducción

- Uno de los lenguajes preferidos para desarrollar aplicaciones en el web
- Donde acudir:

<http://www.perl.com/>

<http://www.perl.com/CPAN/>

<http://www.perlmonks.org/>

## Introducción

- Un programa simple: hello.pl
- Los programas se pueden ejecutar desde la línea de comandos con el “comando” perl
- Para utilizarlo como “shell script” basta con agregar al principio del programa la línea:

```
#!/usr/bin/perl
```

- Dependiendo de donde se encuentre instalado el intérprete puede variar el path “/usr/bin”

## Variables

- En Perl existen 3 tipos de variables: escalares, arreglos y hash, no es necesario declararlas
- Las variables escalares son los “strings” y números:

```
$dato = "Hello World\n";
print $dato;
$dato = 26;
```

- También existe la función “printf”, que funciona igual que en C

## Variables

- No es necesario pedir o liberar espacio para las variables, Perl se preocupa de ello
- Los arreglos se denotan por “@”:

```
@arreglo = (1, 2, 3, 4);
```

- Para acceder al primer elemento del arreglo se hace:

```
$arreglo[0]
```

## Variables

- Arreglos:

```
@a = ("zero", "one", "two", "three", "four");
@b = @a[0, 2..3]; #@b es ("zero", "two", "three")
```

- En una instrucción:

```
@b = ("zero", "one", "two", "three", "four")[0, 2..3];
```

- Asignación desde un arreglo:

```
($cero, $dos, $tres) = @b;
```

## Variables

- Arreglos:

```
@half = ("zero", "one", "two");
@half2 = ("three", "four");
@a = (@half, @half2);
```

- Intercambio de variables, pensando en C:

```
$temp = $a;
$a = $b;
$b = $temp;
```

## Variables

- Intercambio en Perl utilizando arreglos:

```
($a, $b) = ($b, $a);
```

- Hash: arreglos asociativos
  - Permiten almacenar y recuperar datos por medio de una clave
  - Se utiliza el símbolo “%” para identificar un hash
  - Muy útiles para representar relaciones entre datos
  - Las claves de un hash son “case sensitive”

## Variables

- Ejemplo de hash:

```
%fonos = ( "Juan" => "247305",
           "Sara"  => "123322",
           "Luis"  => "987652");
```

- Acceder a los elementos del hash:

```
print "Número de Luis: ", $fonos{"Luis"}, "\n";
print "Número de Luis: ", $fonos{Luis}, "\n";
```

## Variables

- Las variables se expanden al estar con comillas dobles (cuidado con las comillas de las claves del hash):

```
print "Número de Luis: $fonos{Luis}\n";
```

- Perl convierte entre strings y números:

```
print "4 peras" + 2; # imprime "6"
print "8" + "10"; #imprime "18"
```

## Variables Especiales

- Perl tiene una gran cantidad de variables especiales, la principal es: “\$”
  - Variable escalar por omisión para las operaciones y funciones
  - Al llamar a la función “print”, sin argumentos, imprimirá el valor de “\$”
- Por ejemplo:

```
while(<>){
    print unless /^#/;
}
```

## Variables Especiales

- En el ejemplo anterior la variable `$_` se utiliza 3 veces:
  - Tomar una línea de la entrada estándar
  - Como argumento para “print”
  - Para testear si comienza con “#”
- Es recomendable no omitir la variable `$_` en el código
- El arreglo `@ARGV` contiene los argumentos entregados al programa
  - A diferencia de C, `$ARGV[0]` es el primer argumento
  - El nombre del programa se obtiene con `$0`

## Variables

- El hash `%ENV` contiene las variables del ambiente de ejecución:
  - Imprimir el PATH, EDITOR:

```
print $ENV{PATH};
print $ENV{EDITOR};
```

- Modificar el editor:

```
$ENV{EDITOR} = "joe";
```

## Operadores

- Operadores Numéricos:
  - +, -, /, \*, %, \*\*
  - sin, cos, sqrt, log (ver `perlop` y `perlfunc`)
  - El operador % (resto), se debe utilizar con números positivos
  - Si se tiene: `$a % -$b`, el resultado es: `($a % $b) - $b`
  - Incremento, pre y post: `++$a`, `$a++`
- Operadores de Strings:
  - Concatenar “.”

```
$a = "hola"." juan"; # $a es "hola juan"
```

## Operadores

- Concatenar repitiendo el string varias veces:

```
$a = "ba".("na"x2); # "banana"
$a = 1x3; # 111
```

- No confundir los operadores “x” y “\*”
- Borrar el último carácter de un string: `chop`
- “chomp” borra cualquier carácter de retorno o fin de string (`\n`)
- Si los string son completamente alfabéticos, se incrementarán de acuerdo al valor ASCII

## Operadores

- Si los strings comienzan con un número, Perl lo convierte a número y el resto se pierde:

```
$a = "abc"; print ++$a;      #imprime "abd"
$a = "azz"; print ++$a;      #imprime "baa"
$a = "3 peras"; print ++$a; #imprime 4
```

- Se puede separar un string con split:

```
$_ = "uno dos tres";
@a = split; #@a = ("uno", "dos", "tres")
```

## Operadores

- Split:

```
$p = "jose:x:500:500:JoseUrzua:/home/jose:/bin/tcsh";
@p = split ":", $p;
($uid, $gid) = @p[2,3];
```

- En una instrucción:

```
($uid, $gid) = (split ":", $p)[2,3];
```

## Operadores

- Para llevar una lista a un string, se puede utilizar join:

```
@list = ("uno", "dos", "tres");
$numeros = join "-", @list; #"uno-dos-tres"
```

- Para obtener partes de un string:

```
$str = "hola cc31a";
print substr($str, 5); # "cc31a"
print substr($str, -3); # "31a"
print substr($str, 5, 2); # "cc"
```

## Operadores

- Largo de un string: "length"
- Con "reverse" se puede invertir un arreglo
  - Si se aplica a un escalar, lo tomará como un arreglo de 1 elemento
  - Para que se aplique en "contexto" escalar, se debe anteponer la palabra "scalar"

```
$a = "Lenguaje Perl";
print length $a;          # 13
print reverse $a;          # "Lenguaje Perl"
print scalar reverse $a; # "lreP ejaugneL"
```

## Operadores

- Para convertir entre mayúsculas y minúsculas:
  - uc: upper case
  - ucfirst: upper case para el primer carácter
  - lc: lower case
  - lcfirst: lower case para el primer carácter
- Operadores lógicos y de bits:
  - &, |, ^, ~, >>, <<
  - Se utiliza 0x y 0 para los prefijos de hexadecimal y octal
  - Los operadores lógicos &&, || y ! funcionan igual que en C
  - También se pueden utilizar la palabras “and”, “or” y “not”

## Operadores

- Lógicos:

```
funcionCompleja()
  and print "funcionó OK\n"
  or print "No funcionó\n";
```

```
funcionCompleja()
  && print "funcionó OK\n"
  || print "No funcionó\n";
```

## Operadores

- De la misma manera se puede utilizar “if” y “unless”
 

```
print "funcionó OK" if funcionCompleja();
$a = "Valor base" unless $a;
```
- Valores 0 y “undefined” son falsos, el resto verdadero
- Para comparar strings:
  - lt (less than), gt (greater than), eq (equal), ne (not equal)
- Para comparar números:
  - <, >, ==, !=

## Operadores

- Arreglos:

- Para saber el número de elementos se utiliza “scalar @a”
- Para saber el índice del último elemento: \$#a
- Para sacar elementos de un arreglo existe “shift” y “pop”

```
@a = ("1", "2", "3", "4");
print scalar @a; # imprime 4
print $#a;      # imprime 3
print shift @a; # imprime "1"
                # @a es ("2", "3", "4");
print pop @a;   # imprime "4"
                # @a es ("2", "3");
```

## Operadores

- Para agregar elementos a un arreglo existe push y unshift:

```
@a = ();
push @a, "2"; # @a es ("2");
unshift @a, "1"; # @a es ("1", "2");
push @a, "3", "4"; # @a = ("1", "2", "3", "4");
unshift @a, "-1", "0";
print join ", ", @a;

#imprime: -1, 0, 1, 2, 3, 4
```

## Operadores

- Para ordenar un arreglo en ascii-order:

```
@a = ("z", "b", "c", "a");
@b = sort @a; # "a", "b", "c", "z"
```

- Ordenar numéricamente:

```
@a = (5, 8, 3, 0);
@b = sort {$a <=> $b} @a; # 0, 3, 5, 8
```

## Operadores

- Para un hash podemos intercambiar las claves por los valores:

```
%fonos = ("Juan" => "213433",
          "Sara" => "998766",
          "Jose" => "555421");
%numeros = reverse %fonos;
print $numeros{"998766"} # Sara
```

- Si dos valores son iguales, al convertirlos a clave se perderá uno

## Operadores

- Se puede extraer una lista de claves y valores de un hash:

```
@nombres = keys %fonos;
@numeros = values %fonos;
```

- Obtener los valores por pares:

```
while (($key, $value) = each %fonos){
    print "$key tiene el telefono: $value\n";
}
```

- El orden en que están los datos en el hash no es necesariamente el orden en que ingresaron