PERL

CLASE 1

PERL

PERL - Practical Extraction and Report Language

Perl es un lenguaje pensado para la manipulación de cadenas de caracteres, archivos y procesos.

Perl es un lenguaje intermedio entre los shell scripts y la programación en C.

El lenguaje Perl no es precompilado, pero aún así es más rápido que la mayoría de lenguajes interpretados, en especial que el Bourne Shell.

Esto se debe a que los programas en Perl son analizados, interpretados y compilados por el interprete perl antes de su ejecución.

PERL

Historia de PERL

- Larry Wall es el creador (1987) y único soporte de mantenimiento de Perl.
- Liberó la primera versión a la comunidad de lectores de la revista USENET.
- Decenas y cientos de lectores de la revista respondieron a la convocatoria de Larry Wall.
- Como resultado de tan grande convocatoria Perl fue creciendo de la misma manera que ocurrió con el kernel de Unix,

PERL

Perl es ideal para producir soluciones rápidas a problemas pequeños, o para crear prototipos para testear soluciones potenciales a grandes problemas.

Perl provee todas las ventajas del sed y awk. Perl tambien soporta un sed-to-Perl translator y un awk-to-Perl translator.

Una vez que escribimos el código en Perl, podemos usarlo rápidamente en las mas variadas maquinas: practicamente en todas las UNIX, Windows NT, Windows 95, Macintosh, VMS, OS/2, MS/DOS.

En resumen, Perl es tan poderoso como el C pero tan conveniente como awk, sed, and shell scripts.

PERL

Un programa Perl es un conjunto de sentencias y definiciones dentro de un archivo.

Ese archivo se hace ejecutable cambiando los permisos via chmod + x filename.

Es el mismo concepto que tenemos de un shell script (secuencia de comandos shell puestos en un archivo de texto).

Habitualmente se utiliza la extension .pl , pero no es un standard.

PERL

Ejemplo

```
#!/bin/perl
 print "Hola Mundo\n";
```

Como se puede deducir, con la primera línea indicamos donde está el intérprete de Perl, después de #!.
Esto no haría falta si, al ejecutar el script, lo hacemos usando el intérprete.
Por ejemplo: `/usr/bin/perl hola.pl`

Para separar las instrucciones, como en la mayoría de los lenguajes de programación se utiliza el punto y coma (;) y los comentarios precedidos por #.

PERL

Ejemplo

```
#!/bin/perl
 $inputline = <STDIN>;
```

Cuando el intérprete de Perl analiza la sentencia, la divide en unidades más pequeñas de información.

En este ejemplo, las unidades más pequeñas de información son `$inputline`, `=`, recoger con `<STDIN>`, y `;`.

Cada una de estas unidades más pequeñas de información se denomina un **token**.

Los token pueden estar separados por varios blancos. La idea de la utilización de los mismos es hacer legible el programa Perl.

PERL

Tipos de Datos

Escalares:

Las variables contengan un $ representan un escalar.

```
$x = 0.897;          # un real
$y = 6.23e-24;       # un real
$n = 567;            # un entero
$i = -234;           # un entero
```

PERL

Declaración incorrectas de variables escalares:

```
variable      # debe tener un $
$             # debe tener por lo menos una letra
$47x          # el segundo caracter debe ser una letra
$_var         # ídem caso anterior
$variable!    # no puede contener un !
$new.var      # no puede contener un .
```

Los valores enteros no pueden empezar por cero porque esto permite especificar un entero mediante su codificación octal o hexadecimal.
El código octal se antecede con en cero *0;* el código hexadecimal se antecede con un *0x* o *0X.*
Por ejemplo:

```
$x = 0377;              # equivale a 255
$y = 0xff;              # equivale a 255
```

PERL

Las cadenas de caracteres se especifican literalmente por medio de un sucesión de caracteres delimitada por comillas dobles ("..") o simples ('..').

Cuando van delimitadas por comillas (".."), toda variable referenciada en el interior de la cadena se evalúa y se reemplaza por su valor, además interpreta los caracterer \ (\n newline, \t tab, \b backspace).

Por ejemplo, las instrucciones siguientes:

```
$wld = "mundo";
$str = "¡Hola $wld!";print $str ==> ¡Hola mundo!
$str = '¡Hola $wld!';print $str ==> ¡Hola $wld !
```

El tipo booleano existe, al igual que en C, un número es falso si es igual a cero y verdadero en cualquier otro caso.
Como el cero está asociado a la cadena vacía (""), ésta también equivale al valor falso.

PERL

Operadores de variables escalares :

+ suma,

- resta,

***** multiplicación,

/ división
(Ej: `print "2+5*2+15/3\n"; ==> $var = 17`)

****** potencia (Ej:`$var=2**3;print "$var\n"; ==> 8`)

% resto (Ej:`$var=10%3;print "$var\n"; ==> 1`)

. concatenación
(Ej: `$var1=Hola;$var2=Mundo;`
`print $var1.$var2."\n"; ==> HolaMundo`

x repeticion.
(Ej: `$var1=Hola;print $var1 x 3 . "\n"; ==>HolaHolaHola`)

PERL

Operadores de asignación:

=	asignación solamente	(Ej: `$var=2;`)
+=	suma y asignación	(Ej: `$a+=2; #$a=$a+2;`)
-=	resta y asignación	(Ej: `$a-=2; #$a=$a-2;`)
***=**	producto y asignación	(Ej: `$a*=3; #$a=$a*3;`)
/=	división y asignación	(Ej: `$a/=3; #$a=$a/3;`)
x=	repetición y asignación	(Ej: `$b x=3; #$b=$b x 3`)
.=	concatenación y asignación	(EJ:`$b .="33"; #b=$b."33"`)

Operador autoincremental :

++variable pre	variable++ post

`$var1=1;`
`print ++$var1; ==>2`
`print $var1 ==>2`
	`$var1=1;`
	`print $var1++. "\n"; ==>1`
	`print $var1 ==>2`

PERL

El tipo y el valor de las variables en Perl se determina a partir del contexto.
Así, una cadena de caracteres conteniendo un valor numérico será convertida en variable numérica en un contexto numérico.
Consideremos el código siguiente:

```
$x = 4.1;       # un real
$y = "11";      # una cadena de caracteres
$z = $x + $y;   # adición de dos valores numéricos 15.1
$t = $x . $y;   # concatenación de dos cadenas 4.111
```

Se puede *interpolar* variables dentro de otra:

```
$animal="Gato $color de $edad años";
```

La *interpolación* de variables puede evitarse anponiendo la barra invertida (\) o delimitando la cadena con apóstrofes.
Ejemplo:

```
$adr = "www";
$msg = "La direccion es $adr.fi.uba.ar"; print $msg. "\n";
$msg = "La direccion es \$adr.fi.uba.ar"; print $msg. "\n";
$msg = 'La direccion es $adr.fi.uba.ar'; print $msg. "\n";
```

PERL

Comandos chop() y chomp()

La función chomp() remueve (por lo general) el carácter de newline desde el final de una cadena.
La función elimina cualquier carácter que coincide con el valor actual de $/ (input record separator) cuyo valor default es newline.
Ej.

```
$text = <STDIN>;
chomp($text);
print "Usted Ingreso '$text'\n";
```

La función chop() elimina el último carácter de una cadena independientemente de que sea este.
Ej.

```
$string = 'frog';
$chr = chop($string);
print "String: $string\n"; ==> String: fro
print "Char: $chr\n";          ==> Char: g
```

PERL

Funciones con string

`index(CADENA,SUBCADENA,POSICIÓN);`
Retorna la posición de la primera ocurrencia de SUBCADENA dentro de la CADENA iniciando en POSICIÓN. Si no se informa POSICIÓN la búsqueda inicia al principio de la CADENA, (POSICION de 0 a n).

Por ejemplo:

```
$string = "perlmeme.org";
print index($string,"e",2);   #Imprime 5
print index($string,"L");     #Imprime -1
```

PERL

Funciones con string

`substr(CADENA,DESPLAZAMIENTO,LONGITUD);`

Retorna una porción de la cadena entre DESPLAZAMIENTO y LONGITUD. Si no se específica una LONGITUD se va al final de la CADENA. Se puede poner un DESPLAZAMIENTO negativo para iniciar a la derecha de la CADENA.

Por ejemplo:

```
$cadena="la casa esta en orden";
print substr($cadena,3,4)."\n"; # imprime=> casa
print substr($cadena,3,-5)."\n";# imprime=> casa esta en
print substr($cadena,-5,3)."\n";# imprime=> ord
```

PERL

Funciones con string

`length(CADENA);`
Retorna la longitud de la CADENA. Por ejemplo:

```
$string = "programa";
print length($string); #Imprime 8
```

`lc(CADENA);`
Retorna la CADENA convertida en minúsculas. Por ejemplo:

```
$string = "PROGRAMA";
print lc($string); #Imprime programa
```

`uc(CADENA);`
Retorna la CADENA convertida en mayúsculas. Por ejemplo:

```
$string = "programa";
print uc($string); #Imprime PROGRAMA
```

PERL

Arreglos

Un arreglo es una lista de datos de tipo escalar. Cada elemento de la lista es una variable escalar a la que se le asocia un valor.
Las variables de tipo arreglo se identifican por el prefijo arroba **@**.
Por ejemplo:

```
@numeros = (2, 1, 667, 23, 2.2, 5, 6);
@letras = ("perro", "gato", "león");
@mezcla = ("hola", 23, "adios", 31.234);
@alfabeto = (a..z);
```

Los elementos de un arreglo se referencian mediante índices.
El primer elemento se referencia por el índice 0.
Recordar que cada elemento de un arreglo es una variable escalar.
Por ejemplo:

```
print $numero[4];                    # vale 2.2
print $letras[2];                    # vale "león"
print $mezcla[0];                    # vale "hola"
```

PERL

Ejemplos arreglos:

```
@num1 = @numeros[1..3];     # @num1 = (1, 667, 23)
@str = @letras[0,2];        # @str = ("perro", "león")
($ristra, $num)= @mezcla;   # $ristra = "hola";$num = 23
$long = @arreglo            # longitud del arreglo

@a=(1,2,3);@b=(5,6,7);
@c=(@a,4,@b,8); # (1,2,3,4,5,6,7,8)

@ordenada = sort(@arreglo)
@arreglo2 = reverse(@arreglo) # de atrás para adelante
```

PERL

Comandos para arreglos:

Para sacar/insertar elementos se pueden usar las funciones **pop** y **push**, que sacan o insertan, respectivamente, un elemento al final, es decir, tratan el array como una pila.

También podemos utilizar **shift** y **unshift** para sacar o insertar, respectivamente, un elemento del principio del array.

Para ver la última posición del array **$#array**.

```
Ej.print "El último es ".$arreglo[$#arreglo]."\n";
```

Para ver longitud asignar a una variable escalar el array

```
Ej.$longarray=@array.
```

PERL

Ejemplos comandos para arreglos:

```
# Asignamos unos cuatro valores al array
@matriz=(1,"dos",3,"cuatro");

# Añadimos con Push
push(@matriz, 5, 6, "siete");

# Mostramos el último elemento
print "El último es ".$matriz[$#matriz]."\n";
# Imprime: El último es siete

# Sacamos con Pop
$uno=pop(@matriz);
print "He sacado $uno\n";
# Imprime: He sacado siete
```

PERL

Ejemplos comandos para arreglos:

```
# Añadimos con unshift
unshift(@matriz, "cero" );

# Mostramos el primer elemento
print "El primero es ". $matriz[0]."\n";
# Imprime: El primero es cero

# Sacamos con shift
$uno=shift(@matriz);
print "He sacado $uno\n";
# Imprime: He sacado cero
print "La matriz tiene ".($#matriz + 1)." elementos\n";
# Imprime: La matriz tiene 6 elementos
```

PERL

Ejemplos comandos para arreglos:

```
 #!/usr/local/bin/perl
($file1, $file2) = @ARGV;
print "El primer archivo pasado como parámetro es: $file1\n
El segundo es : $file2\n";
```

PERL

Hashes (listas asociativas)

Una lista asociativa está indexada por cadenas en lugar de por números. Se utiliza % para definir el tipo de lista asociativa y un elemento está indexado por el anterior formando ambos parejas del tipo (clave, valor).

```
%cuota = ("root", 1000, "Juan", 256, "Jose", 4000);
%fruit = (apples => 3,oranges => 6);
print $cuota{root} ." ". $fruit{oranges}."\n"; # 1000 6
```

Esta lista puede completarse añadiendo nuevos valores y asociando a cada clave el valor correspondiente.
Por ejemplo:

```
$cuota{"dave"} = 250;
```

PERL

En cuanto al nombre de las variables están permitidos las letras, dígitos y el carácter underscore (_).

Las letras mayúsculas y minúsculas son diferenciadas en los nombres de variables. Los nombre de las variables siempre deben comenzar por una letra.

Funiones:

defined(VARIABLE). `ej. print "Existe" if defined(@array);`
delete(KEY). `ej .delete($hash{key})`
exists(KEY). `ej. exists($hash{"key"})`
values(HASH). `ej. @valores=values(%hash)`
keys(HASH). `ej. @claves=keys(%hash)`

PERL

Estructuras de Control

if (expresión) {
   instrucción o bloque de intrucciones 1;
}
[else {
   instrucción o bloque de instrucciones 2;
}
Ejemplo:

```
if ($i==5)
 {
 print 'La variable $i es 5'."\n";
 }
 else
 {
 print 'La variable $i no es 5'."\n";
 }
```

PERL

Operadores de Comparación

Numeric	String	Meaning
>	gt	Greater than
>=	ge	Greater than or equal to
<	lt	Less than
<=	le	Less than or equal to
==	eq	Equal to
!=	ne	Not equal to
<=>	cmp	Comparison, with signed result

PERL

while (expresión) {
**   instrucción o bloque de instrucciones;**
}

Ejemplo:

```
print "Teclea \"x\" para salir:\n";
$cadena = "";
while ($cadena ne "x") {
    $cadena = <STDIN> ;
    chop($cadena);
    print "Has escrito $cadena\n";
}
print "Salida.\n"
```

PERL

for (inicial_exp; test_exp; incremento_exp) {
instrucción o bloque de intrucciones;
}

Ejemplo:

```
print "10 Iteraciones\n";
for ($i=0; $i<10; $i++) {
        print "Interación número $i\n";
}
@lista = ("elem1", "elem2", "elem3", "elem4");
for ($i = 0; $i<= $#lista; $i++) {
   print $lista[$i], "\n";
```

PERL

foreach $variable (@lista) {
instrucción o bloque de instrucciones;
}

Ejemplo:

```
%dias=("lunes",L,"martes",M,"miercoles",X,"jueves",J,"
viernes",V,"sabado",S,"domingo",D);
 foreach $clave (keys(%dias))
 {
 print "\%dias{".$clave."}=".$dias{$clave}."\n";
 }
```

PERL

La instrucción **last** interrumpe la ejecución del bucle actual y se ejecuta la instrucción que sigue al bloque.

El ejemplo siguiente permite interrumpir el bucle while cuando la variable i toma el valor 3.

```
$i = 0;
while($i < 6) {
    if($i == 3) {
      last;
    }
    $i++;
}
print "el valor de \$i es $i";
```

PERL

La instrucción **next** es idéntica a la instrucción continue en C. Interrumpe la ejecución del bloque de instrucción actual y prosigue la ejecución en la iteración siguiente Esta instrucción no interrumpe completamente la ejecución del bucle; la expresión que controla el bucle se evalúa. Si el resultado de la expresión es válido, el bucle se ejecuta de nuevo.

```
print "Teclea \"x\" para salir:\n";
print "Si se pulsa la tecla \"s\" no se imprime:\n";
$cadena= "";
while ($cadena ne "x") {
   $cadena=<STDIN>;
        chop($cadena);
   if ($cadena eq "s") {
      next;   }
   print "Has escrito $ristra\n";
}
print "Salida.\n"
```

PERL

Entradas y Salidas

Para leer de teclado se utiliza <STDIN>, que es un manejador de ficheros (Filehandle), que por costumbre se ponen en mayúsculas.

También existe <STDOUT> y <STDERR> para la sálida estándar y la sálida de errores respectivamente.

Lectura de STDIN:

```
#!/usr/bin/perl
print "Hola. ¿Cómo te llamas?\n";
$nombre=<STDIN>;chop($nombre);
if ($nombre eq "")
{print 'Ingresa tu nombre'."\n";}
else
{print "Hola $nombre!\n";};
```

PERL

Entradas y Salidas

En el anterior ejemplos el script de Perl se espera a que el usuario pulse Return (retorno de carro), es decir, lee una línea.
Si en lugar de una variable escalar hubiera utilizado un array, el script hubiese guardado también los retornos de carro hasta que se hubiese pulsado CTRL+D.
Para escribir en la salida estándar se utiliza print como hemos visto, pero si queremos escribir en la salida de errores sería poniendo después de print el manejador de fichero de donde queremos escribir.

PERL

Entradas y Salidas

Para trabajar con un archivo hay que abrirlo, escribir/leer de él y cerrarlo.
Para abrir un fichero se utiliza la función open y para cerrarlo con close.
El formato para abrir un fichero es open (Manejador_de_fichero, Modo_y_NombreFichero).
Lo modos de abrir un fichero se muestran en la siguiente tabla.

Archivo	Abre Archivo para lectura
<Archivo	Abre Archivo para lectura
>Archivo	Abre Archivo para estritura, lo crea si no exixte
>>Archivo	Abre Archivo para appendear
+>Archivo	Abre Archivo para Lectura Escritura

PERL

Entradas y Salidas

```
Ejemplo:
#!/usr/bin/perl
$entrada="entrada.txt";
$salida ="salida.txt";
open (ENTRADA,"<$entrada") || die "ERROR: No puedo
abrir el fichero $entrada\n";
open (SALIDA,">$salida") || die "ERROR: No puedo abrir
el fichero $salida\n";
while ($linea=<ENTRADA>)
{
print SALIDA $linea;
}
close (ENTRADA);
close (SALIDA);
```