

Capítulo 2: web dinámica



Perl



- Perl
 - **P**ractical **E**xtraction and **R**eport **L**anguage
 - Creado e implementado por Larry Wall en 1987
 - Es un lenguaje interpretado
 - Combina características de
 - Shell, sed, awk, de C y también BASIC
- Muy útil para desarrollar de forma rápida
 - Manejo de archivos
 - Funciones de sistema
- Lenguaje preferidos para el desarrollo de aplicaciones WEB
 - perlmonks.org

Perl



- Ejecutar un programa perl
`perl programa.pl`
- También se puede agregar a la primera línea del programa (path de perl):

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

- Se debe dar permisos de ejecución al programa
- Más información, ejecute:

```
perldoc
```

```
perldoc perldoc
```

```
perldoc perlrun
```



- Perl es muy permisivo
 - Deja mucha responsabilidad en el programador
 - Para dar robustez a los programas se debe utilizar

```
#!/usr/bin/perl  
use strict;  
use warnings;
```

- Con “use strict” se capturan problemas potenciales
 - Se detiene la ejecución apenas se encuentra
- Con “use warnings” se generan advertencias pero sigue la ejecución
- Para el curso **siempre** deben usar estas líneas



- Sintaxis
 - Serie de sentencias
 - No es necesario un “main()”
 - Las sentencias terminan con ; comentarios comienzan con #

```
# Este es un comentario  
print "Hello, world";
```

- Con “ se interpolan variables

```
print "Hello, $name\n";  
print 'Hello, $name\n';      # imprime $name literal
```



- Variables

- Escalares

```
my $animal = "camel";  
my $answer = 42;  
print $animal;  
print "The animal is $animal\n";  
print "The square of $answer is ", $answer *  
$answer, "\n";
```

- Existen variables especiales:

```
print; # imprime el valor de $_ (variable default)
```

- Otras variables especiales: perldoc perlvar

- Alcance de las variables

- Revisar scope.pl



- Variables

- Arreglos: representa una lista de valores

- ```
my @animals = ("camel", "llama", "owl");
my @numbers = (23, 42, 69);
my @mixed = ("camel", 42, 1.23);
```

```
print $animals[0]; # prints "camel"
print $animals[1]; # prints "llama"
print $mixed[$#mixed];
```

```
if (@animals < 5) { ... } # verdadero si tiene menos
 # 5 elementos

@animals[0,1]; # retorna ("camel", "llama");
@animals[0..2]; # retorna ("camel", "llama", "owl");
@animals[1..$#animals]; # retorna todos excepto el
 # primero
```



- Variables: arreglos

- Funciones útiles:

```
my @sorted = sort @animals;
my @backwards = reverse @numbers;
```

- Arreglos especiales:

- @ARGV: argumentos que se reciben por línea de comandos
    - @\_ : argumentos que recibe una subrutina
    - Mas información: perldoc perlvar





- Variables: Hashes
  - Conjunto de pares llave → valor

```
my %fruit_color = ("apple", "red", "banana",
"yellow");
my %fruit_color = (
 apple => "red",
 banana => "yellow",
);
$fruit_color{"apple"}; # retorna "red"

my @fruits = keys %fruit_colors;
my @colors = values %fruit_colors;
```

- Hash especial: %ENV (variables.pl)



- Sentencias condicionales

```
if (condition) {
 ..
} elsif (other condition) {
 ..
} else {
 ..
}
```

```
if ($zippy) {
 print "Yow!";
}
```

```
print "Yow!" if $zippy;
print "We have no bananas" unless $bananas;
```



- Ciclos

```
while (condition) {
 ...
}
```

```
until (condition) {
 ...
}
```

```
print "LA LA LA\n" while 1; # Que hace?
```

```
for ($i = 0; $i <= $max; $i++) {
 ...
}
```

```
foreach (@array) {
 print "This element is $_\n";
}
```

```
print $list[$_] foreach 0 .. $max;
```



- Operadores y funciones

## Aritméticos

+ addition  
- subtraction  
\* multiplication  
/ division

## Comparación Numérica

== equality  
!= inequality  
< less than  
> greater than  
<= less than or equal  
>= greater than or equal

## Comparaciones Strings:

eq equality  
ne inequality  
lt less than  
gt greater than  
le less than or equal  
ge greater than or equal

## Operadores Lógicos

&& and  
|| or  
! not



- Archivos

```
open(my $in, "<", "input.txt") or die "Can't open input.txt: $!";
open(my $out, ">", "output.txt") or die "Can't open output.txt: $!";
open(my $log, ">>", "my.log") or die "Can't open my.log: $!";
```

```
my $line = <$in>; # lee una linea del archivo
my @lines = <$in>; # lee todas las líneas y las deja en el arreglo
```

```
while (<$in>) { # asigna cada línea a $_
 print "leyendo la línea: $_";
}
```

```
print STDERR "This is your final warning.\n";
print $out $record;
print $log $logmessage;
```

```
close $in or die "$in: $!";
```



- Expresiones regulares
  - Una de las grandes potencialidades de Perl

```
if (/foo/) { ... } # true if $_ contains "foo"
if ($a =~ /foo/) { ... } # true if $a contains "foo"
```

```
s/foo/bar/; # replaces foo with bar in $_
$a =~ s/foo/bar/; # replaces foo with bar in $a
$a =~ s/foo/bar/g; # replaces ALL INSTANCES
```



- Expresiones regulares

|               |                                                   |
|---------------|---------------------------------------------------|
| .             | a single character                                |
| \s            | a whitespace character (space, tab, newline, ...) |
| \S            | non-whitespace character                          |
| \d            | a digit (0-9)                                     |
| \D            | a non-digit                                       |
| \w            | a word character (a-z, A-Z, 0-9, _)               |
| \W            | a non-word character                              |
| [aeiou]       | matches a single character in the given set       |
| [^aeiou]      | matches a single character outside the given set  |
| (foo bar baz) | matches any of the alternatives specified         |
| ^             | start of string                                   |
| \$            | end of string                                     |



## • Expresiones Regulares

```
* zero or more of the previous thing
+ one or more of the previous thing
? zero or one of the previous thing
{3} matches exactly 3 of the previous thing
{3,6} matches between 3 and 6 of the previous thing
{3,} matches 3 or more of the previous thing

/^\d+/ string starts with one or more digits
/^\$/ nothing in the string (start and end are adjacent)
/(\d\s){3}/ a three digits, each followed by a whitespace
 character (eg "3 4 5 ")
/((a.))+/ matches a string in which every odd-numbered
 letter is a (eg "abacadaf")

ciclo que lee desde la entrada estandar e imprime líneas
que no están en blanco:
while (<>) {
 next if /^\$/; print;
}
```





- Expresiones regulares
  - Captura de datos con paréntesis

```
if ($email =~ /([^\@]+)@(.+)/) {
 print "Username is $1\n";
 print "Hostname is $2\n";
}
```

- Ejemplo: hora.pl