PERL

CLASE 3

Subrutinas

PERL

Qué es una Subrutina

En Perl, una subrutina es una porción de código separada usada para efectuar una tarea particular.

El programa PERL ejecuta esa porción de código llamando o invocando la subrutina.

Una Subrutina se usa para alguno de estos dos propósitos:

- Para partir un programa en partes pequeñas haciéndolo mas fácil de leer y mas entendible.
- Para permitirnos usar una porción de código para llevar a cabo la misma tarea varias veces eliminando la duplicación.

PERL

Ejemplo

```
$total = 0;
&getnumbers;
foreach $number (@numbers) {
        $total += $number;
}
print ("El total es $total\n");

sub getnumbers
{
       $line = <STDIN>;
       $line =~ s/^\s+|\s*\n$//g;
       @numbers = split(/\s+/, $line);
}
```

PERL

La sintaxis para la definición de una subrutina es:

```
sub subname {
        statement_block
}
```

`subname` es el nombre de la subrutina. Como todos los nombres en Perl, subname consiste en un carácter alfabético seguido de una o más letras, dígitos o guiones (_).
Es aconsejable evitar nombres iguales al de una palabra reservada de perl, y no poner todo el nombre con letras mayúsculas ya que así es como Perl llama a sus subrutinas internas.

`statement_block` es el cuerpo de la subrutina y consiste en una o mas sentencias Perl. Cualquier sentencia que puede aparecer en un main program, puede aparecer en una subrutina.

PERL

NOTA

El Intérprete de Perl nunca confunde un nombre de subrutina con el nombre de una variable escalar o cualquier otro porque puede deducir del contexto cual nombre estamos haciendo referencia. Por lo tanto podemos tener una subrutina y una variable escalar con el mismo nombre.
Por ejemplo:

```
$word = 0;
&word;
```

Aquí, cuando el Intérprete de Perl ve el **&** en la segunda sentencia, asume que esta llamando una subrutina llamada word.

Cuando se definen nombres de subrutinas, es conveniente no usar nombres iguales a las funciones de librería de Perl para evitar errores involuntarios.

PERL

Las subrutinas pueden aparecer en cualquier parte de un programa

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

while (1) {
        &readaline;
        last if ($line eq "");
        sub readaline {
                $line = <STDIN>;
        }
        print ($line);
}
print ("FIN\n");
```

Si bien las subrutinas pueden aparecer en cualquier parte de un programa, es usual definirlas al principio o al final de un programa. Esto lo hace mas “legible”.

PERL

Forward References a Subrutinas

Como vimos, el Intérprete de Perl usa el carácter & para indicar que se trata de una subrutina.
En Perl 5 no es necesario el & antes del nombre en la invocación si la subrutina ya fue definida.
Ejemplo:

```
sub readaline {
        $line = <STDIN>;
}
...
Readaline;
```

Esto es porque el intérprete ya sabe que readaline es una subrutina.

PERL

Forward References a Subrutinas

Si escribimos todas las subrutinas hacia el final de un programa, entonces si debemos preceder su invocación con el carácter & salvo que hagamos al principio una *forward reference* de la misma:

```
sub readaline;   # forward reference
...
readaline;
...
sub readaline {
        $line = <STDIN>;
}
```

El forward reference le dice al Intérprete de Perl que readaline es el nombre de una subrutina. De esta manera no necesitamos adicionarle el & en la invocación.

PERL

Retornando un valor desde una Subrutina:

```
sub getnumbers {
        $line = <STDIN>;
        $line =~ s/^\s+|\s*\n$//g;
        @numbers = split(/\s+/, $line);
}
```

En el caso de la subrutina del ejemplo, tiene una seria limitación: sobrescribe el arreglo **@numbers** (tambien sobrescribe $line).
Esto puede traer problemas… Por ejemplo:

```
@numbers = ("the", "a", "an");
&getnumbers;
print ("El valor de \@numbers es: @numbers\n");
```

Cuando la subrutina getnumbers es invocada, el valor de @numbers es sobrescrito.

PERL

Retornando un valor desde una Subrutina:

Para evitar esto podemos utilizar una propiedad de las subrutinas en Perl: el valor de la última expresión evaluada por la subrutina es considerada automáticamente como el valor de retorno (return value) de la subrutina.

Por ejemplo, en la subrutina getnumbers la última expresión evaluada es:

```
@numbers = split(/\s+/, $temp);
```

El valor de esta expresión es la lista de números obtenida del splitting de la línea de input. De esta manera, la lista de números es el valor de retorno de la subrutina.

PERL

Retornando un valor desde una Subrutina:

```
$total = 0;
@numbers = &getnumbers; # así getnumbers retornará su valor
en @numbers
foreach $number (@numbers) {
         $total += $number;
}
print ("El total es $total\n");
sub getnumbers {
        $line = <STDIN>;
        $line =~ s/^\s+|\s*\n$//g;
        split(/\s+/, $line);     # Este es el return value
}
```

PERL

Retornando un valor desde una Subrutina:

Para hacer mas claro la idea de la última expresión evaluada se puede crear una variable dentro de la subrutina solo para contener el valor de retorno.
Por ejemplo:

```
sub getnumbers {
$line = <STDIN>;
$line =~ s/^\s+|\s*\n$//g;
@retval = split(/\s+/, $temp); # the return value
}
```

Aquí es obvio que el valor de retorno está contenido en @retval.

La desventaja está solo en usar una asignación cuando no es necesaria. Eliminarla hace la subrutina mas eficiente. Usar una variable especial para retornar valores elimina unos cuantos errores que veremos mas adelante.

PERL

Valores de Retorno y Expresiones Condicionales

Como el valor de retorno de una subrutina es siempre la última expresión evaluada, a veces el valor de retorno no devuelve lo que esperamos.

```
$total = &get_total;
print("El total es $total\n");
sub get_total {
        $value = 0;
        $inputline = <STDIN>;
        $inputline =~ s/^\s+|\s*\n$//g;
        @subwords = split(/\s+/, $inputline);
        $index = 0;
        while ($subwords[$index] ne "") {
                $value += $subwords[$index++];
        }
}
```

PERL

Valores de Retorno y Expresiones Condicionales

El problema en la subrutina **get_total** es que la última expresión evaluada no es la que nosotros esperábamos sino la expresión: `$subwords[$index] ne ""`

Cuando el valor de la expresión es cero, el loop termina, y la subrutina también.

Por lo tanto el 0 es lo que retorna y como en un print el 0 se interpreta como string nulo, por lo tanto lo que se imprime es lo siguiente (lo cual no es lo que esperábamos):

```
El total es
```

PERL

Valores de Retorno y Expresiones Condicionales

La solución a este problema sería la siguiente :

```
$total = &get_total;
print("El total es $total.\n");

sub get_total {
        $value = 0;
        $inputline = <STDIN>;
        $inputline =~ s/^\s+|\s*\n$//g;
        @subwords = split(/\s+/, $inputline);
        $index = 0;
        while ($subwords[$index] ne "") {
                $value += $subwords[$index++];
        }
       $retval = $value;
}
```

PERL

Valores de Retorno y Expresiones Condicionales

Con la asignación de **`$retval`** nos aseguramos que esta sea la ultima sentencia evaluada.
Pero en realidad no sería necesario efectuar la asignación, haciendo la siguiente modificación:

```
sub get_total {
        $value = 0;
        $inputline = <STDIN>;
        $inputline =~ s/^\s+|\s*\n$//g;
        @subwords = split(/\s+/, $inputline);
        $index = 0;
        while ($subwords[$index] ne "") {
                $value += $subwords[$index++];
        }
        $value;
}
```

PERL

La Sentencia return

Otra manera de asegurar el valor de retorno de una subrutina es usando la sentencia return. La sintaxis es la siguiente:

```
return (retval);
```

`retval` es el valor que queremos retornar.

Puede ser un valor escalar (incluido el resultado de una expresión) o una lista.

PERL

La Sentencia return

```
$total = &get_total;
if ($total eq "error") {print ("No se ingresó nada.\n");}
else { print("el total es $total.\n");
}
sub get_total {
        $value = 0;
        $inputline = <STDIN>;
        $inputline =~ s/^\s+|\s*\n$//g;
        if ($inputline eq "") {
                return ("error");}
      @subwords = split(/\s+/, $inputline);
      $index = 0;
      while ($subwords[$index] ne "") {
              $value += $subwords[$index++];
      }
      $retval = $value;
}
```

PERL

Usando variables locales en Subrutinas

Si tenemos la certeza que las variables solo serán usadas dentro de la subrutina, podemos decirle a Perl que defina estas variables como locales (local variables).

En Perl 5, existen dos sentencias para definir local variables:

- La sentencia **`my`**, la cual define variables que solo van a existir dentro de la subrutina.
- La sentencia **`local`**, la cual define variables que solo van a existir dentro de la subrutina y cualquier otra subrutina llamada por esta.

PERL

Usando variables locales en Subrutinas

```
$total = 0;
while (1) {
        $linetotal = &get_total;
        last if ($linetotal eq "done");
        print ("Total para esta línea: $linetotal\n");
        $total += $linetotal;}
print ("Total para todas las Líneas: $total\n");
sub get_total {
        my ($total, $inputline, @subwords);
        my ($index, $retval);
        $total = 0;
        $inputline = <STDIN>;
        if ($inputline eq "") {return ("done");}
        $inputline =~ s/^\s+|\s*\n$//g;
        @subwords = split(/\s+/, $inputline);
        $index = 0;
        while ($subwords[$index] ne "") {
                $total += $subwords[$index++];}
        $retval = $total;
}
```

PERL

Usando variables locales en Subrutinas

El programa usa dos copias de la variable escalar **`$total`**.
Una copia de **`$total`** esta definida en el main program y almacena el total de todas las Líneas.
La variable escalar **`$total`** está también definida en la subrutina **`get_total`**; en esta subrutina, **`$total`** referencia al total para una linea en particular.

Como la variable local no es conocida fuera de la subrutina, se destruye cuando la subrutina es completada. Si la subrutina es llamada nuevamente una nueva copia de la variable local es definida.
El siguiente ejemplo muestra como NO funcionan:

```
sub subrutina_count {
my($number_of_calls);
$number_of_calls += 1;
}
```

PERL

Inicialización de variables locales

Las variables locales pueden ser inicializadas.
Ejemplo:

```
sub my_sub {
        my($scalar) = 43;
        my(@array) = ("here's", "a", "list");
        # comandos
}
```

Aqui la variable local `$scalar` toma un valor inicial de 43, y el arreglo `@array` es inicializado con la lista ("here's", "a", "list").

PERL

Pasando valores a una Subrutina

Podemos hacer mas flexibles nuestras subrutinas si permitimos que ellas acepten valores pasados desde el main program.
Estos valores se los llama Argumentos.

```
print ("Ingrese 3 numeros, uno por vez:\n");
$number1 = <STDIN>;chop ($number1);
$number2 = <STDIN>;chop ($number2);
$number3 = <STDIN>;chop ($number3);
&printnum ($number1, $number2, $number3);
sub printnum {
        my($number1, $number2, $number3) = @_;
        my($total);
        print ("Los numeros ingresados son: ");
        print ("$number1 $number2 $number3\n");
        $total = $number1 + $number2 + $number3;
        print ("El total es: $total\n");
}
```

PERL

Pasando valores a una Subrutina

Después que la variable @_ es creada, puede ser usada dentro de la subrutina como cualquier arreglo.
Ejemplo:

```
sub printnum {
my($total);
print ("The numbers you entered: ");
print ("$_[0] $_[1] $_[2]\n");
$total = $_[0] + $_[1] + $_[2];
print ("The total: $total\n");
}
```

PERL

Pasando valores a una Subrutina

Después que la variable @_ es creada, puede ser usada dentro de la subrutina como cualquier arreglo.
Ejemplo:

```
sub printnum {
my($total);
print ("The numbers you entered: ");
print ("$_[0] $_[1] $_[2]\n");
$total = $_[0] + $_[1] + $_[2];
print ("The total: $total\n");
}
```

Usualmente es mejor definir variables locales y asignar @_ a ellas para que así sea mas fácil de entender el código.

PERL

Pasando una lista a una Subrutina

Si queremos pasar una lista a una subrutina veamos el siguiente ejemplo:

```
sub addlist {
        my (@list) = @_;
        $total = 0;
        foreach $item (@list) {
                $total += $item;
        }
        print ("The total is $total\n");
}
```

En cada caso, los valores son juntados en una sola lista, y pasados como una lista simple a addlist.
Como los valores son juntados, entonces solo se debe incluir una lista en los argumentos de una sub y es conveniente ponerla COMO ULTIMO ARGUMENTO.

PERL

Llamando Subrutinas desde otras Subrutinas

En Perl, podemos llamar subrutinas desde otra subrutina. Para hacerlo, se usa la misma sintaxis que vinimos empleando.
Las Subrutinas que son llamadas desde otras subrutinas se las conoce como **nested subrutinas**

PERL

Llamando Subrutinas desde otras Subrutinas

```
($wordcount, $charcount) = &getcounts(3);
print ("Totals for three lines: ");
print ("$wordcount words, $charcount characters\n");
sub getcounts {
        my ($numLines) = @_;
        my ($charpattern, $wordpattern);
        my ($charcount, $wordcount,$line, $linecount,@retval);
        $charpattern = "";
        $wordpattern = "\\s+";
        $linecount = $charcount = $wordcount = 0;
        while (1) {
                $line = <STDIN>;
                last if ($line eq "");
                $linecount++;
                $charcount += &count($line, $charpattern);
                $line =~ s/^\s+|\s+$//g;
                 $wordcount += &count($line, $wordpattern);
        last if ($linecount == $numLines);};
        @retval = ($wordcount, $charcount);
}
```

PERL

Llamando Subrutinas desde otras Subrutinas

```
sub count {
        my ($line, $pattern) = @_;
        my ($count);
        if ($pattern eq "") {
                @items = split (//, $line);
        } else {
                @items = split (/$pattern/, $line);
        }
        $count = @items;
}
```

PERL

Subrutinas Predefinidas

Perl 5 define tres subrutinas especiales que son ejecutadas en momentos específicos.

- BEGIN subrutina, se llama ANTES que cualquier sentencia. Es más, se ejecuta ANTES que los script regulares sean analizados.
- END subrutina, se llama ANTES que termine el script regular, al mismo tiempo que sale del intérprete de perl.
- AUTOLOAD subrutina, se carga si hacemos una llamada a una subrutina inexistente. La variable $AUTOLOAD contiene el nombre de la sub que dio error, la variable @_ contiene la información pasada en la rutina que dio error.

PERL

Subrutinas Predefinidas

Ejemplo

```
BEGIN { print "empezamos ...\n";}
END { print "nos fuimos \n"}
AUTOLOAD { print "la sub $AUTOLOAD no existe\n"}
```