Unidad 5: Sentencias Alternativas

Cátedra Introducción a la Programación
Tecnicatura en Desarrollo de Aplicaciones Web
Facultad de Informática
Buenos Aires 1400 (8300) Neuquén
Universidad Nacional del Comahue, Argentina

---

1. SENTENCIAS ALTERNATIVAS

Las alternativas son partes de nuestra vida cotidiana, continuamente lidiamos con ellas en nuestras actividades. Por ej una alternativa simple como: "si llueve voy al cine, si no llueve voy a la plaza", o una alternativa mas trascendental "¿que carrera voy a seguir?".

Las estructuras alternativas son una estructura de control útil cuando nos encontramos con problemas que requieren ejecutar una o más instrucciones según diferentes casos. En otras palabras, las estructuras alternativas bifurcan o dirigen la ejecución de un programa hacia un grupo de sentencias u otro dependiendo de una condición. Las distintas condiciones que incluye una alternativa pueden estar explícitas en el mismo problema que debemos resolver, por ej: "dado un entero ingresado por el usuario si el valor del entero es par incrementarlo, si es impar decrementarlo", o pueden ser parte de la solución matemática o lógica de un problema. Veremos a continuación un ejemplo introductorio a las estructuras alternativas.

Imaginemos que se nos plantea el siguiente problema:

Problema 4.1:. Debemos leer un número y, si el número es positivo, debemos escribir en pantalla el cartel *"Numero positivo"*.

Solución. Especificamos nuestra solución: se deberá leer un número x. Si x $>$ 0 se escribe el mensaje "Número positivo".

Diseñamos nuestra solución:.

ALGORITMO principal
  ESCRIBIR("Ingrese un numero:")
  LEER(num)
  SI (num $>$ 0) **ENTONCES**
    ESCRIBIR (num, " es Numero Positivo")
  FIN SI
FIN ALGORITMO principal

Es claro que la primera línea se puede traducir como
echo (" "Ingrese un numero: "); = *trim(fgets(STDIN));*

Sin embargo, con las instrucciones que vimos hasta ahora no podemos tomar el tipo de decisiones que nos planteamos en la segunda línea de este diseño, sin utilizar el operador ternario.

Para resolver este problema introducimos una nueva instrucción que llamaremos *condicional* que tiene la siguiente forma:

```
1 if (condición) {
2         <hacer algo si se da la condición>
3 }
```

Donde ***if*** es una palabra reservada.

¿Qué es la condición que aparece luego de la palabra reservada ***if***? Antes de seguir adelante con la construcción debemos introducir un nuevo tipo de expresión que nos indicará si se da una cierta situación o no. Hasta

ahora las expresiones con las que trabajamos fueron de tipo numérica y de tipo texto. Pero ahora la respuesta que buscamos es de tipo sí o no.

1.1. Expresiones booleanas

Además de números y de textos, PHP introduce las constantes *true* y *false* para representar los valores de verdad *verdadero* y *falso* respectivamente.

Vimos que una expresión es una porción de código PHP que produce o calcula un valor (resultado). Una *expresión booleana* o *expresión lógica* es una expresión que tiene como resultado o *true* o *false*.

1.1.1. Expresiones de comparación

En el ejemplo que queremos resolver, la condición que queremos ver si se cumple o no es que x sea mayor que cero. PHP provee las llamadas *expresiones de comparación* que sirven para comparar valores entre sí, y que por lo tanto permiten codificar ese tipo de pregunta. En particular la pregunta de si x es mayor que cero, se codifica en PHP como $x > 0$.

De esta forma, $2 > 3$ es una expresión booleana cuyo valor es *false*, y $4 > 2$ también es una expresión booleana, pero su valor es *true*.

En PHP el resultado de las siguientes expresiones podemos visualizarlos en el siguiente script:

Fig. 1. Visualizando valores expresiones booleanas en PHP

Para PHP el resultado verdadero se imprime con el valor **1** , y el valor de falso imprime con vacío.

El manual de PHP (`http://php.net/manual/es/language.types.boolean.php`) explica el tipo de datos booleano:
Un boolean expresa un valor que indica verdad. Puede ser TRUE (verdadero) o FALSE (falso). Para especificar un literal de tipo boolean se emplean las constantes **TRUE** o **FALSE**, siendo ambas constantes no son susceptibles a mayúsculas y minúsculas.
Por ser un lenguaje dinámico PHP realiza conversiones de tipos y si el contexto requiere convertir un tipo de dato a boolean lo hará de la siguiente manera:

Los siguientes valores se consideran FALSE:
el boolean **FALSE** mismo
el integer **0 (cero)**
el float **0.0 (cero)**
el valor **string vacío**, y el string **"0"**
un array con cero elementos

Cualquier otro valor se considera como TRUE (incluso el valor -1 es tomado como TRUE).

Las expresiones booleanas de comparación que provee PHP son las que se detallan en la tabla I

1.1.2. Operadores lógicos

De la misma manera que se puede operar entre números mediante las operaciones de suma, resta, etc., también existen tres operadores lógicos para combinar expresiones booleanas: ***and*** (y), ***or*** (o) y ***not*** (no).

El significado de estos operadores es igual al del castellano, pero vale la pena recordarlo en II:

Expresión	Significado
a == b	*a* es igual a *b*
a <> b	*a* es distinto de *b*
a < b	*a* es menor que *b*
a <= b	*a* es menor o igual que *b*
a > b	*a* es mayor que *b*
a >= b	*a* es mayor o igual que *b*

Table I. Expresiones Booleanas en PHP

Expresión	Significado
a **&** **&** b	El resultado es *Verdadero* solamente si **a** es *Verdadero* y **b** es *Verdadero* de lo contrario el resultado es *Falso*
a || b	El resultado es *Verdadero* si **a** es *Verdadero* o **b** es *Verdadero* de lo contrario el resultado es *Falso*
! a	El resultado es *Verdadero* si **a** es *Falso* de lo contrario el resultado es *Verdadero*

Table II. Operadores lógicos

—(a > b **and** a > c) es verdadero si *a* es simultáneamente mayor que *b* y que *c*.

```
1  <?php
2  echo "\n\nEste es el resultado de la expresion:(5>2 and 5>3) >> ".(5>2 and 5>3);
3  echo "\nEste es el resultado de la expresion:(5>2 and 5>6) >> ".(5>2 and 5>6);
4  ?>
```

—a > b **or** a > c es verdadero si *a* es mayor que *b* o *a* es mayor que *c*.

```
1  <?php
2  echo "\n\nEste es el resultado de la expresion: (5>2 or 5>3) >>> ".(5>2 or 5>3);
3  echo "\nEste es el resultado de la expresion: (5>2 or 5>6) >>> ". (5>2 or 5>6);
4  echo "\nEste es el resultado de la expresion: (5>8 or 5>6) >>> ".(5>8 or 5>6) ;
5  ?>
```

—**!** (a > b) es verdadero si *a* > *b* es falso (o sea si *a* <= *b* es verdadero).

```
1  <?php
2  echo "\n\nEste es el resultado de la expresion: (5>8) >>> " .(5>8);
3  echo "\nEste es el resultado de la expresion: !(5>8) >>> " .!(5>8);
4  echo "\nEste es el resultado de la expresion: (5>2) >>> " .(5>2);
5  echo "\nEste es el resultado de la expresion: !(5>2) >>> " . !(5>2);
6  ?>
```

1.2. Comparaciones simples

Volvemos al problema que nos planteamos al inicio: Debemos leer un número y, si el número es positivo, debemos escribir en pantalla el mensaje *"Numero positivo"*.

Utilizando la instrucción **if** que acabamos de introducir y que sirve para tomar decisiones simples. Dijimos que su formato más sencillo es:

```
1  if (condición){
2          <hacer algo si se da la condición>
3  }
```

cuyo significado es el siguiente: se evalúa *<condición>* y si el resultado es *=1* (verdadero) se ejecutan las acciones indicadas en *<hacer algo si se da la condición>*.

Como ahora ya sabemos cómo construir condiciones de comparación, estamos en condiciones de implementar nuestra solución. Escribimos la función es_positivo($valor) que hace lo pedido:

```
1  <?php
2  function esPositivo($valor){
3          if ($valor>0)
4      echo  "El numero es positivo";
5  }
```

y lo probamos:

```
function probando_esPositivo(){
        echo "Ingrese un valor";
        $numero = trim(fgets(STDIN));
        esPositivo($numero);
}
probando_esPositivo();
```

Problema 1.2.. En la etapa de mantenimiento nos dicen que, en realidad, también se necesitaría un mensaje "*Numero no positivo*" cuando no se cumple la condición.

Modificamos la especificación consistentemente y modificamos el diseño:

MODULO esPositivo(Number valor) **RETORNO** ∅
  SI (valor > 0) **ENTONCES**
    ESCRIBIR (valor,“ es Numero Positivo”)
  FIN SI
  SI not(valor > 0) **ENTONCES**
    ESCRIBIR (valor,“ es Numero No Positivo”)
  FIN SI
FIN MODULO

La negación de $x > 0$ es $\neg(x > 0)$ que se traduce en PHP como **!** $(x > 0)$, por lo que implementamos nuestra solución en PHP como:

```
function esPositivo($valor){
        if ($valor>0)
                echo "El numero es positivo";
        if (!($valor>0))
                echo "El numero NO es positivo";
}
```

Sin embargo hay algo que nos preocupa: si ya averiguamos una vez, en la segunda línea del cuerpo, si $x > 0$, ¿Es realmente necesario volver a preguntarlo en la cuarta?.

Existe una construcción alternativa para la estructura de decisión:

Si se da la condición C, hacer S, de lo contrario, hacer T. Esta estructura tiene la forma:

```
if (condición){
        <hacer algo si se da la condición>
}
else{
        <hacer otra cosa si no se da la condición>
}
```

Donde **if** y **else** son palabras reservadas.

Su significado es el siguiente: se evalúa *<condición>*, si el resultado es *=1* (verdadero) se ejecutan las acciones indicadas en *<hacer algo si se da la condición>*, y si el resultado es *<>1* (falso) se ejecutan las acciones indicadas en *<hacer otra cosa si no se da la condición>*.

Volvemos a nuestro diseño:

MODULO esPositivoElse(Number valor) **RETORNO** ∅
  SI (valor > 0) **ENTONCES**
    ESCRIBIR (valor,“ es Numero Positivo”)
  SINO
    ESCRIBIR (valor,“ es Numero No Positivo”)
  FIN SI
FIN MODULO

Este diseño se implementa como:

```
1  function esPositivoElse($valor){
2          if ($valor>0)
3                  echo "El numero es positivo";
4          else
5                  echo "El numero NO es positivo";
6  }
```

y lo probamos:

```
1  function probando_esPositivoElse(){
2          echo "Ingrese un valor";
3          $numero = trim(fgets(STDIN));
4          esPositivoElse($numero);
5  }
```

Es importante destacar que, en general, negar la condición del **if** y poner **else** no son intercambiable, no necesariamente producen el mismo efecto en el programa. Notar qué sucede en los dos programas que se transcriben a continuación. ¿Por qué se dan estos resultados?:

```
1  <?php function probando($valor){
2          if ($valor>0){
3                  echo "El numero es positivo";
4                  $valor = -$valor;
5          }
6          if (!($valor>0))
7                  echo "El numero NO es positivo";
8  }
9  probando(25);
10 ?>
```

```
1  <?php
2  function probando2($valor){
3          if ($valor>0){
4                  echo "El numero es positivo";
5                  $valor = -$valor;
6          }else
7                  echo "El numero NO es positivo";
8  }
9  probando2(25);
10 ?>
```

1.3. Múltiples decisiones consecutivas

La decisión de incluir una alternativa en un programa, parte de una lectura cuidadosa de la especificación. En nuestro caso la especificación nos decía:

Si el número es positivo escribir un mensaje "*Numero positivo*", de lo contrario escribir un mensaje "*Numero no positivo*".

Veamos qué se puede hacer cuando se presentan tres o más alternativas.

Problema 1.3.. Si el número es positivo escribir el mensaje "*Numero positivo*", si el número es igual a 0 el mensaje "*Igual a 0*", y si el número es negativo escribir el mensaje "*Numero negativo*".

Una posibilidad es considerar que se trata de una estructura con dos casos como antes, sólo que el segundo caso es complejo (es nuevamente una alternativa):

Este diseño se implementa como:

```
1  function alternativas_anidadas($valor){
2          if ($valor> 0)
3                  echo "Numero positivo";
4          else{
5                  if ($valor == 0)
6                          echo "Igual a 0";
7                  else
```

MODULO alternativas$_{A}nidadas(Numbervalor)$ **RETORNO** $\emptyset$
SI (valor >0) **ENTONCES**
ESCRIBIR(valor, “es positivo”)
SINO
SI(valor = 0) **ENTONCES**
ESCRIBIR(valor, “es igual a cero”)
SINO
ESCRIBIR(valor, “es No positivo”)
FIN SI
FIN SI
FIN MODULO

```
8                               echo "Numero negativo";
9           }
10  }
```

Esta estructura se conoce como *alternativas anidadas* ya que dentro de una de las ramas de la alternativa (en este caso la rama del **else**) se anida otra alternativa.

```
1  if (condición_1){
2          <hacer algo_1 si se da la condición_1>
3  } elseif (condición_2){
4          <hacer algo_2 si se da la condición_2>
5  } elseif (condición_n){
6          <hacer algo_n si se da la condición_n>
7  } else {
8          <hacer otra cosa si no se da ninguna de las condiciones anteriores>
9  }
```

MODULO probando_elif(Number valor) **RETORNO** $\emptyset$
SI (valor >0) **ENTONCES**
ESCRIBIR(valor, “es positivo”)
OTRO-SI (valor = 0) **ENTONCES**
ESCRIBIR(valor, “es igual a cero”)
SINO
ESCRIBIR(valor, “es No positivo”)
FIN SI
FIN MODULO

Donde **if, elif** y **else** son palabras reservadas.
En nuestro ejemplo:

```
1  function  probando_elseif($valor){
2          if ($valor > 0){
3                  echo  $valor." Numero positivo \n";
4          }elseif ($valor == 0){
5                  echo $valor. " Igual a 0 \n";
6          }else{
7                  echo  $valor." Numero negativo \n";
8          }
9  }
```

Se evalúa la primera alternativa, si es verdadera se ejecuta su cuerpo. De lo contrario se evalúa la segunda alternativa, si es verdadera se ejecuta su cuerpo, etc. Finalmente, si todas las alternativas anteriores fallaron, se ejecuta el cuerpo del *else*.

SABIAS QUE...

No sólo con los operadores vistos (como > o =) es posible obtener expresiones booleanas. En PHP, se consideran verdaderos los valores numéricos distintos de 0, las cadenas de caracteres que no son vacías, y en general cualquier valor que no sea 0 o vacío. Mientras que los valores 0 o vacíos se consideran falsos.
Así, el ejemplo anterior también podría escribirse de la siguiente manera:

```
1 function probando_elseif2($valor){
2         if ($valor > 0){
3                 echo $valor." Numero positivo \n";
4         } elseif ($valor == 0){
5                 echo $valor. " Igual a 0 \n";
6         }else{
7                 echo $valor." Numero negativo \n";
8         }
9 }
```

1.4. Ejercicios

Ejercicio 1.1.. El usuario del tarifador nos pide ahora una modificación, ya que no es lo mismo la tarifa por segundo de las llamadas cortas que la tarifa por segundo de las llamadas largas. Al inicio del programa se informará la duración máxima de una llamada corta, la tarifa de las llamadas cortas y la de las largas. Se deberá facturar con alguno de los dos valores de acuerdo a la duración de la comunicación.

Ejercicio 1.2.. Mantenimiento del tarifador:

1. Al nuevo programa que cuenta con llamadas cortas y largas, agregarle los adicionales, de modo que:

   —Los montos se escriban como pesos y centavos.

   —Se informe además cuál fue el total facturado en la corrida.

2. Modificar el programa para que sólo informe cantidad de llamadas cortas, valor total de llamadas cortas facturadas, cantidad de llamadas largas, valor total de llamadas largas facturadas, y total facturado. Al llegar a este punto debería ser evidente que es conveniente separar los cálculos en funciones aparte.

Ejercicio 1.3.. Dados tres puntos en el plano expresados como coordenadas (x, y) informar cuál es el que se encuentra más lejos del centro de coordenadas.

1.5. Resumen

—Para poder tomar decisiones en los programas y ejecutar una acción u otra, es necesario contar con una **estructura condicional**.

—Las **condiciones** son expresiones booleanas, es decir, cuyos valores pueden ser *verdadero* o *falso*, y se las confecciona mediante operadores entre distintos valores.

—Mediante **expresiones lógicas** es posible modificar o combinar expresiones booleanas.

—La estructura condicional puede contar, opcionalmente, con un bloque de código que se ejecuta si no se cumplió la condición.

—Es posible ***anidar*** estructuras condicionales, colocando una dentro de otra.

—También es posible ***encadenar*** las condiciones, es decir, colocar una lista de posibles condiciones, de las cuales se ejecuta la primera que sea verdadera.

REFERENCIA DEL LENGUAJE PHP

if (condición):

Bloque condicional. Las acciones a ejecutar si la condición es verdadera.

```
1 if (condición){
2         acciones a ejecutar si condición es verdadera
3 }
```

else:

Un bloque que se ejecuta cuando no se cumple la condición correspondiente al if. Sólo se puede utilizar else si hay un if correspondiente.

```
1 if (condición){
2         acciones a ejecutar si condición es verdadera
3 } else{
4         acciones a ejecutar si condición es falsa
5 }
```

elseif (condición)

Bloque que se ejecuta si no se cumplieron las condiciones anteriores pero sí se cumple la condición especificada. Sólo se puede utilizar elseif si hay un if correspondiente, y las acciones a ejecutar deben escribirse en un bloque de indentación mayor. Puede haber tantos elseif como se quiera, todos al mismo nivel**.**

```
1 if (condición1){
2         acciones a ejecutar si condición1 es verdadera
3 }elseif (condición2){
4         acciones a ejecutar si condición2 es verdadera
5 }else{
6         acciones a ejecutar si ninguna condición fue verdadera
7 }
```

Operadores de comparación

Son los que forman las expresiones booleanas.

Expresión	Significado
a == b	*a* es igual a *b*
a <> b	*a* es distinto de *b*
a < b	*a* es menor que *b*
a <= b	*a* es menor o igual que *b*
a > b	*a* es mayor que *b*
a >= b	*a* es mayor o igual que *b*

Operadores lógicos

Son los utilizados para concatenar o negar distintas expresiones booleanas.

Expresión	Significado
a **&** **&** b	El resultado es *Verdadero* solamente si **a** es *Verdadero* y **b** es *Verdadero* de lo contrario el resultado es *Falso*
a || b	El resultado es *Verdadero* si **a** es *Verdadero* o **b** es *Verdadero* de lo contrario el resultado es *Falso*
! a	El resultado es *Verdadero* si **a** es *Falso* de lo contrario el resultado es *Falso*