Algoritmos: 4

Estructuras de repetición

- Ciclos condicionados al inicio
- Ciclos condicionados al final

Introducción a estructuras de repetición

Es muy común encontrar en la práctica algoritmos cuyas operaciones se deben ejecutar un número repetido de veces. Si bien las instrucciones son las mismas, los datos sobre los que se opera pueden variar.

Tipos de estructuras de repetición

Todo ciclo debe terminar de ejecutarse luego de un número finito de veces, por lo que es necesario en cada iteración del mismo, evaluar las condiciones necesarias para decidir si se debe seguir ejecutando o si debe detenerse. En todo ciclo, siempre debe existir una condición de parada o fin de ciclo la cual puede estar al inicio o al final. Por lo cual se tienen:

- Ciclos condicionados al inicio.
- Ciclos condicionados al final.

Ciclos condicionados al inicio

Las estructuras de repetición condicionadas al inicio tienen la particularidad de preguntar si cumplen con cierta condición antes de entrar en el ciclo. Estas estructuras de repetición se dividen en:

- For
- While

Ciclo For

La estructura de repetición FOR, es la adecuada para utilizar en un ciclo que se ejecutará un número definido de veces. Este tipo de estructura está presente en todos los lenguajes de programa orientados a objetos y estructurados.

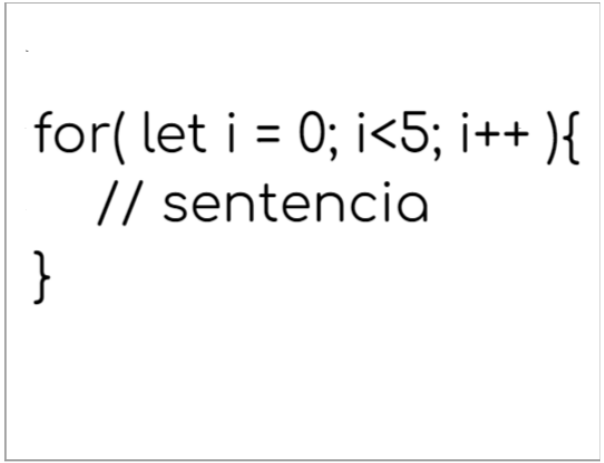

***Tabla 1. Representación del ciclo* For *con* JavaScript**

Sintaxis For

La estructura básica de un ciclo for en JavaScript se compone de tres partes principales:

1. **Inicialización**: Se establece una variable de control (normalmente un contador).
2. **Condición**: Se evalúa antes de cada iteración del ciclo. Si la condición es true, el ciclo continúa; si es false, el ciclo termina.
3. **Actualización**: Se ejecuta al final de cada iteración y generalmente se usa para actualizar la variable de control.

```
for (inicialización; condición; actualización) {
    // Código a ejecutar en cada iteración
}
```

Ejemplo de un problema usando ciclo For

Programe un algoritmo que sume los números del 1 al 1000.

```
let suma = 0;
for (let i = 0; i<1000; i++){
    suma = suma+ i;
}
```

Ejercicio

Programe un algoritmo que sume los números pares del 1 al 700.

Ciclo While

La estructura de repetición While, es la estructura adecuada para utilizar en un ciclo cuando no sabemos el número de veces que éste se ha de repetir.

```
let i = 0
while (i < 5) {
    // sentencia
    i++
}
```

***Tabla 2. Representación del ciclo* While *con* JavaScript**

Sintaxis While

La estructura básica del ciclo while en JavaScript se compone de las siguientes partes:

1. **Inicialización**: La variable de control se inicializa antes del ciclo.
2. **Condición**: Se evalúa antes de cada iteración del ciclo. Si la condición es true, el ciclo continúa; si es false, el ciclo termina.
3. **Actualización**: Se debe asegurar que la variable de control se actualice dentro del cuerpo del ciclo para evitar un bucle infinito.

```
// Inicialización
let variableDeControl = valorInicial;
while (condición) {
    // Código a ejecutar en cada iteración

    // Actualización
    variableDeControl++;
}
```

Ejemplo de un problema usando ciclo While

Programe un algoritmo que sume los 100 números negativos más pequeños.

```
let i = -1;
let suma = 0;
while (i >= -100) {
    suma = suma + i;
    i--;
}
```

Ciclos condicionados al final

Las estructuras de repetición condicionadas al final tienen la particularidad de ejecutar primero las sentencias dentro del ciclo y luego preguntar si cumple la condición para continuar, por lo cual ejecuta siempre mínimo una vez las instrucciones dentro del ciclo. La estructura de repetición que se usa para este tipo de ciclos es:

- Do While

Ciclo Do While

La estructura de repetición Do While, es la estructura adecuada para utilizar en un ciclo cuando sabe que se ejecutara mínimo una vez

```
let i = 0
do {
    // sentencia
    i++
} while (condition);
```

Tabla 3. Representación del ciclo **Do While** ***con*** JavaScript

Sintaxis Do-While

La estructura básica del ciclo Do While en JavaScript se compone de las siguientes partes:

1. **Inicialización**: La variable de control se inicializa antes del ciclo.
2. **Código a ejecutar**: Se ejecuta al menos una vez, independientemente de la condición.
3. **Actualización**: Cualquier cambio a las variables de control que asegure que la condición eventualmente se vuelva falsa.
4. **Condición**: Se evalúa después de cada iteración del ciclo. Si la condición es true, el ciclo continúa; si es false, el ciclo termina.

```
// Inicialización
let variableDeControl = valorInicial;

do {
   // Código a ejecutar en cada iteración

   // Actualización
   variableDeControl++;
} while (condición);
```

Ejemplo de un problema usando ciclo Do While

Programe un algoritmo que lea dos números enteros y devuelva la suma entre los elementos leídos (incluyendolos). Realice la respectiva prueba de escritorio.

```
let min = parseInt(prompt("Ingrese el numero minimo"));
let max = parseInt(prompt("Ingrese el numero maximo"));
let suma = 0;
let i = min;
do {
    suma = suma + i;
    i++;
} while (i <= max);
```

Acumuladoras y Contadores

- **Contador:**
  - Debe inicializarse antes de entrar al ciclo
  - Dentro del ciclo debe haber una instrucción que modifique el valor del contador
    - Ejemplo:

      contador = contador + Incremento
      contador = contador - Decremento

  - El valor a incrementar es cualquier cantidad de tipo numérico y sirve para aumentar o disminuir su valor en cada iteración del ciclo
  - los valores a incrementar o decrementar son valores constantes
- **Acumuladora:**
  - Debe inicializarse antes de entrar al ciclo
  - Dentro del ciclo debe haber una instrucción que modifique el valor
    - Ejemplo:

      acumuladora = acumuladora + Incremento
      acumuladora = acumuladora - Decremento

  - El valor a incrementar es cualquier cantidad de tipo numérico
  - Aumenta o disminuye en cada iteración del ciclo
  - los valores a incrementar o decrementar son valores variables o fijos

Banderas

- Si la variable es de tipo booleno, los únicos valores son true o flase
- Si la variable es de tipo entero, los valores posibles podrían ser 1 o 0, que se pueden interpretar como Verdadero o Falso
- Si la variable es de tipo carácter, puede tomar cualquier valor; los valores más comunes son S o N interpretados como si o no
- Es fundamental darle un valor inicial antes del ciclo, este valor cambiará dependiendo de ciertas condiciones que estarán dadas por la solución del problema

Ejemplos Banderas, Acumuladores y Contadores

Banderas

```
let sigue = true; // Bandera para controlar el ciclo
let i= 0;

while (sigue) {
    i++;
    if (i >= 5) {
        sigue = false; // Cambiar la bandera para salir del ciclo
    }
}
```

Ejemplos Banderas, Acumuladores y Contadores

Acumulador

```
let notas = 0;
for (let i = 1; i <= 5; i++) {
    let nota = parseInt(prompt("Ingrese su nota"));
    notas += nota;
}
```

Ejemplos Banderas, Acumuladores y Contadores

Contador

Ejemplo 1

```
let contador = 0;
while( contador < 10 ){
    contador++
}
```

Ejemplo 2

```
let goles = 0;
for(let i = 0; i < 90; i++ ){
    let gol = prompt("Metieron gol? (S/N)")
    if(gol == "S"){
        goles++
    }
}
```

Ejercicio

Programe un algoritmo que pida un nombre y la nacionalidad y los muestre, todo esto tantas veces como diga el usuario

Haga la respectiva prueba de escritorio

Algoritmos: 4

VankVersity