==========flow controls:==========

---

it will control the flow of the program.

1.decision making statements
2.looping statements
3.jumping statemenst

The “if” statement:

The if(...) statement evaluates a condition in parentheses and, if the result is true, executes a block of code.

```
<script>


let year = prompt('In which year was ECMAScript-2015 specification published?', '');

if (year == 2015) alert( 'You are right!' );
</script>
```

if else:

```
<script>


let year = prompt('In which year was the ECMAScript-2015 specification published?', '');

if (year == 2015) {
  alert( 'You guessed it right!' );
} else {
  alert( 'How can you be so wrong?' ); // any value except 2015
}
</script>
```

Several conditions: “else if”:

Sometimes, we’d like to test several variants of a condition. The else if clause lets us do that.

```
<script>


let year = prompt('In which year was the ECMAScript-2015 specification published?', '');

if (year < 2015) {
  alert( 'Too early...' );
```

```
} else if (year > 2015) {
  alert( 'Too late' );
} else {
  alert( 'Exactly!' );
}
</script>
```

Conditional operator '?':

Sometimes, we need to assign a variable depending on a condition.

```
<script>


let accessAllowed;
let age = prompt('How old are you?', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);
</script>
```

eg: let result = condition ? value1 : value2;

eg: let accessAllowed = (age > 18) ? true : false;

Multiple '?':

A sequence of question mark operators ? can return a value that depends on more than one condition.

```
<script>


let age = prompt('age?', 18);

let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hello!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';

alert( message );
</script>
```

Loops:
---

while and for

We often need to repeat actions.

For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10.

Loops are a way to repeat the same code multiple times.

while:

syntax:

```
while (condition) {
  // code
  // so-called "loop body"
}
```

eg:

```
<script>

let i = 0;
while (i < 3) { // shows 0, then 1, then 2
  alert( i );
  i++;
}
</script>
```

```
<script>

let i = 3;
while (i) { // when i becomes 0, the condition becomes falsy, and the loop stops
  alert( i );
  i--;
}
</script>
```

The “do…while” loop:

syntax:

```
do {
  // loop body
} while (condition);
```

```
<script>


let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
</script>
```

The “for” loop:
---

synatx:

```
for (begin; condition; step) {
  // ... loop body ...
}
```

eg:

```
<script>


for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
  alert(i);
}
</script>
```

eg:
--

```
<script>


for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // error, no such variable
</script>
```

eg:
--

```
<script>
```

```
let i = 0;

for (i = 0; i < 3; i++) { // use an existing variable
  alert(i); // 0, 1, 2
}

alert(i); // 3, visible, because declared outside of the loop
</script>
```

Breaking the loop:

Normally, a loop exits when its condition becomes falsy.

But we can force the exit at any time using the special break directive.

For example, the loop below asks the user for a series of numbers, “breaking” when no number is entered:

```
      <script>


let sum = 0;

while (true) {

  let value = +prompt("Enter a number", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Sum: ' + sum );
</script>
```

Continue to the next iteration:

The continue directive is a “lighter version” of break. It doesn’t stop the whole loop. Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).

We can use it if we’re done with the current iteration and would like to move on to the next one.

The loop below uses continue to output only odd values:

```
<script>
```

```
for (let i = 0; i < 10; i++) {

  // if true, skip the remaining part of the body
  if (i % 2 == 0) continue;

  alert(i); // 1, then 3, 5, 7, 9
}
</script>
```