|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385314) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385314) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385314) |
|  | |
| 3. JavaScript Control Structures  3.1. Conditional Statements  *Copyrighted Material - subject to fair use exception*  **if Statement:**  The general syntax of an if statement is:  if (condition) {      Statements to be executed if condition is truthy  }  The then block is executed if the expression is truthy.  **The parentheses around the condition are required.**  The indentation is just for readability.  It does not affect the correctness of the code.  **The curly braces delimit the block of code to be executed.**  It is recommended to always delimit the block of code with curly braces { } even if it is just one statement.  It is just more readable and less error prone.  Because of the implicit semicolon insertion, it is also recommended to **start the curly braces on the same line as the condition.**  Example:  var balance, amount;  if  (balance >= amount) {      balance = balance - amount;  }  **if else Statement:**  if  (condition) {      Statements to be executed if condition is truthy  } else {       Statements to be executed if condition is not truthy  }  Example:  var number, absolute;  if (number >= 0) {      // The absolute value of a positive number is the number itself      absolute = number;  } else {      // The absolute value of a negative number is the opposite of that number      absolute = -number;  }  **Conditional Operator:**  The conditional operator ? : may sometimes be used to achieve the same result as an if statement.  The general syntax is as follows:  condition ?  operand1  : operand2  The above expression evaluates to operand1 if the condition is truthy and to operand2 if the condition is falsy.  Example:  number >=  0 ? number : -number  // return the absolute value of number  We can try the above expression in the Firebug console as follows:  >>> var number = 5;  >>> number >= 0 ? number : -number  // Here the condition is true so number (5) is returned  5  >>> number = - 3;  >>> number >= 0 ? number : -number  // Here the condition is false so -number (- - 3) is returned  3  **Multiple if else if Statements:**  Sometimes there is more than one condition that we need to check.  We can use multiple if … else if… statements as follows:  if (condition 1) {       // Execute code block #1  } else if (condition 2) {       // Execute code block #2  } else if (condition 3) {       // Execute code block #3  } else {       // If all else fails, execute block #4   }  Example:  var grade = 87, letterGrade;  if (grade >= 90) {      letterGrade = 'A';  } else if (grade >= 80) {      letterGrade = 'B';  } else if (grade >= 70) {      letterGrade = 'C';  } else if (grade >= 60) {      letterGrade = 'D';  } else {      letterGrade = 'F';  }  console.log('Letter Grade: ', letterGrade);    Letter Grade: B    3.2. The Switch Statement  *Copyrighted Material - subject to fair use exception*  The switch statement is used to perform a multiway branch based on the value of a given expression.  The expression can produce a number or a string. The expression is compared to the values specified in each case clause.  If there is a match, execution starts at the block of code associated with that case. If there is no match, the optional default statements are executed.  **We use break to prevent the code from falling through into the next case.**When using **switch inside a function, we use a return** statement instead of a break statement.    **Syntax:**  switch (expression) {  case  value1:      execute code block 1      break;  case value2:      execute code block 2      break;  default:      code to be executed if the expression is different from value1 and value2  }  **Example:**  var grade = 'B', gpa;  switch (**grade**) {  case 'A':    // if grade === 'A'      gpa = 4;      break;  case 'B':  // if grade === 'B'      gpa = 3;      break;  case 'C':  // if grade === 'C'      gpa = 2;      break;  case 'D':  // if grade === 'D'      gpa = 1;      break;  default:      gpa = 0;  }  console.log(gpa);    3    3.3. While Loops  *Copyrighted Material - subject to fair use exception*  The general syntax of a while statement is as follows:  while (expression) {      block of code to execute  }  The expression is first evaluated.  If it is falsy, then the block of code that follows is skipped completely. If the expression is truthy, then the block of code is executed repeatedly as long as the expression is truthy.  Example 1:  while (true) {      console.log('Ha');  }  console.log('Bye');  **Do not try that.** This is an example of an infinite loop.  The expression always evaluates to true. The loop does not terminate.  It never gets to print ‘Bye’.  Example 2:  while (false) {      console.log('This will never get printed');  }  console.log('Bye');  Bye  In this case the statement inside the loop never gets executed because the condition is always false.  Example 3:  var counter = 1;  while (counter < 10) {      console.log(counter)  }  This is another infinite loop.  What is missing? counter is always 1.  We need to update counter inside the loop.  This is a very common mistake. **Remember to update the variable used in the while condition.**  Example 4:  var counter = 1;  while (counter <= 10) {      console.log(counter++); //  this is the increment operator.  More details next.  }  1  2  3  4  5  6  7  8  9  10  3.4. Increment and Decrement Operators  *Copyrighted Material - subject to fair use exception*  The increment operator is frequently used in loops.  In general, ++ is equivalent to adding 1 to the given variable.  However there is a distinction between the post-increment and pre-increment operators.  Let’s illustrate that difference with an example.  You can follow along by entering the code below and running it in Scratchpad.  var counter = 1;  var result = counter++;  console.log('counter:', counter);  console.log('result:', result);  Remember that the console.log output will appear in the Firebug console.  counter: 2  result: 1  With post-increment (counter++), result is assigned the current value of counter, 1, then counter is incremented.  result = counter++; is equivalent to:  result = counter;  //assign first  counter = counter + 1;  // then increment  So result is 1 while counter is 2.  Let's try the pre-increment operator next:  var counter = 1;  var result = ++counter ;  console.log('counter:', counter);  console.log('result:', result);    The corresponding output in the Firebug console:  counter: 2  result: 2  With pre-increment (++counter), counter is incremented first then result is assigned the incremented value, 2.  result = ++ counter; is equivalent to:  counter = counter + 1;  // increment first  result = counter;  // then assign  The decrement operator is similar.  Here are two examples with post-decrement and pre-decrement:  var counter = 1;  var result = counter--;  console.log('counter:', counter);  console.log('result:', result);  counter: 0  result: 1  var counter = 1;  var result = --counter;  console.log('counter:', counter);  console.log('result:', result);  counter: 0  result: 0  In a while loop, using the pre-increment or post-increment operators yield different results:  var counter = 1;  while (counter <= 3) {      console.log(counter++);  }  1  2  3  var counter = 1;  while (counter <= 3) {      console.log(++counter);  }  2  3  4  **More Shorthand Operators:**  JavaScript has some more shorthand operators that you’ll also see in loops:  \*=, /=, %=, += and -=.  var counter = 10;  counter -= 2;  // shorthand for counter = counter - 2  console.log(counter);  8  var counter = 10;  counter += 2;  // shorthand for counter = counter + 2  console.log(counter);  12  var counter = 10;  counter \*= 2; // shorthand for counter = counter \* 2  console.log(counter);  20  var counter = 10;  counter /= 2 ;  // shorthand for counter = counter / 2  console.log(counter);  5  var counter = 10;  counter %= 2;   // shorthand for counter = counter % 2  console.log(counter);  0    3.5. do ... while ... Statements  *Copyrighted Material - subject to fair use exception*  The general syntax of a do … while... statement is as follows:  do {      block of code to execute  } while (expression)**;**  The do … while loop is similar to the while loop, except that expression is tested at the bottom of the loop rather than at the top. This means that**the body of the loop is executed at least once.**  Note that **the do loop must always be terminated with a semicolon.**  Examples :  var counter = 4;  do {     console.log(counter++);  } while (counter < 3);  4  The body of the loop is executed even though the condition is false from the beginning.  var counter = 1;  do {      console.log(counter++);  } while (counter < 3);  1  2  3.6. for Statements  *Copyrighted Material - subject to fair use exception*  A for loop allows us to put the initialization, the test, and the update of the loop variable on a single line.  The general syntax is as follows:  for ( initialize ; test ; update) {      Block of code to be executed  }  initialize is executed before the loop starts.  test defines the condition for running the loop.  update is executed each time after the loop (the code block) has been executed.  The advantage of a for loop over the equivalent while loop is that it prevents bugs due to forgetting to initialize or increment the loop variable.  Example:  var counter;  // remember to declare the loop variable  for ( counter  =  1; counter  < 5; counter++) {      console.log(counter);  }  1  2  3  4  **We should always declare the loop variable**.  It is a common practice to move the declaration inside the for loop as follows:  for (**var**  counter  =  1; counter  < 5; counter++) {      console.log(counter);  }  Note that **the parentheses after the for should always contain two semicolons,** even if we omit the initialization, test or update.  So if counter is initialized in previous statements, it would be OK to write:  for ( ; counter  < 5; counter++) {      console.log(counter);  }  Note that **omitting the test would result in an infinite loop unless you include a break statement** inside the loop.   **The break statement causes the innermost enclosing loop to exit immediately.** |  |