Урок #5 – Условные операторы (if-else)

Все языки программирования содержат условную конструкцию. В уроке мы научимся использовать оператор if-else и научимся создавать условия. Также в видео мы сделаем мини игру на основе условных операторов.

Условные операторы очень важны, ведь они позволяют выполнять разный код в зависимости от исхода условия. В языке C# существует три конструкции позволяющих осуществлять проверку

Конструкция if - else

- За счёт if else можно проверить одно или несколько условий и в случае их успешной проверки будет выполнен один, иначе другой.

- int a = 2, b = 10;
- if (a == b) { // Если a будет равным b, тогда будет выполнен код // Здесь код который будет выполнен // Если все одна строка кода, то фигурные скобки {} // можно не ставить }
- else if (a <= b) { // Если a будет меньшим или равным b // Если предыдущее условие не будет выполнено, // а здесь условие окажется верным, // то будет выполнен этот код }
- else { // Этот код сработает, если другие условия не будет выполнено }

- Вы можете прописать структуру лишь с одним условием if, а можете дописать в неё сколько-угодно вложенных условий else if.
- Внутри каждого из условий можно прописывать другие конструкции if else, что будут проверять новые выражения.
- Если необходимо проверить несколько условий в одном операторе, то можно воспользоваться логическим «и» или же логическим «или»:
- if (a != b && a > b) { // Код будет выполнен, если и первое, и второе условие // окажутся верными }
- if (a < b || a == b) { // Код будет выполнен, если или первое, или второе условие // окажется верным }

Оператор «Switch-case»

- Продолжаем тему условных конструкций. В ходе урока мы рассмотрим пример работы конструкции switch-case. За счет этого операторы мы можем выполнять проверку одной переменной на несколько возможных значений.

Конструкция switch

- Конструкция case обладает более удобным форматом для проверки множественных условий на совпадение значения. В конструкцию записывается переменная, что проверяется, а также значения на которые происходит проверка.
- Пример оператора:

- int x = 23; switch (x) { // Проверяем переменную x
- case 1: // Если переменная будет равна 1, то здесь сработает код // Может быть множество строк, а не только одна
- Console.WriteLine("Переменная равна 1");
- break; // Указываем конец для кода для этой проверки case 56: // Если переменная будет равна 56, то здесь сработает код // Может быть множество строк, а не только одна Console.WriteLine("Переменная равна 56");
- break; // Указываем конец для кода для этой проверки // По аналогии таких проверок может быть множество // Также можно добавить проверку, которая сработает в случае // если все остальные проверки не сработают default: Console.WriteLine("Что-то другое"); break; // Можно и не ставить, так как это последние условие }

Цикл for, while и do while. Операторы циклов

- Для выполнения кода несколько раз подряд необходимо использовать всевозможные циклы. В уроке мы научимся работать с циклами for, while и do while. Также мы создадим мини программу и познакомимся с операторами «break» и «continue» в циклах.

- В языке **C#**, как и в большинстве других языков, существует 3 вида циклов. Каждый из них выполняет одну и ту же роль, но записывается по-разному. Рассмотрим все три цикла.
- **Цикл For**
- В цикле for все условия записываются в одном месте, что очень удобно во многих случаях. Стандартная запись такого цикла выглядит следующим образом:
- for (int i = 0; i < 10; i++) Console.WriteLine(i);В объявлении цикла записывается следующее: переменная цикла, её начальное значение и диапазон. В примере выше будут выведены числа от 0 до 10.
- Если в теле цикла всего одна строка кода, то фигурные скобки можно пропустить и не записывать их.

Цикл While

- Суть цикла while не особо отличается от цикла for. Единственное отличие заключается в способе записи цикла. В while необходимо прописать лишь условие, а все остальные параметры записываются вне цикла:
- int i = 1; // Создание переменной while (i <= 10) { // Здесь только условие Console.WriteLine(i); i++; // Увеличение переменной }

Цикл Do While

- Цикл схож с циклом while по форме написания, но при этом работает немного по-другому. Цикл **do..while** будет выполнен один раз сто процентов, а дальше проверит условие и если оно верно, то цикл будет выполняться дальше:
- int x = 13; do { x--; Console.WriteLine(x); } while (x > 10);Как видно из примера, цикл изначально неверный, но это не мешает ему сработать один раз.

Операторы для работы в циклах

- Существует три основных оператора для работы в циклах:
- Оператор break - служит для выхода из цикла полностью;
- Оператор return - выполняет ту же функцию, что и оператор break, но для циклов лучше использовать break;
- Оператор continue - пропускает лишь одну итерацию и не выходит из цикла.

Массивы данных. Одномерные и многомерные

- Для хранения большого объема данных можно использовать массивы. За урок мы изучим концепцию использования массивов и научимся создавать одномерные и многомерные массивы данных.
- Массивы позволяют хранить большой объем информации в одном месте. В языке **C#** можно найти несколько основных типов массивов.

Одномерный массив

- Чтобы создать массив необходимо указать тип данных, поставить квадратные скобки и назвать массив. Это очень схоже с созданием обычных переменных, но здесь после типа данных идут еще квадратные скобки.
-
- В массивах отсчет начинается с 0, поэтому первый элемент по индексу будет равен 0, второй - 1 и так далее.
-
- Примеры создания массива:
- char[] stroka = new char[2]; // Создание пустого массива int numbers[]; // Будет считаться ошибкой int[] nums = new int[4]; nums[0] = 1; // Добавление элементов в массив из 4 элементов int[] nums2 = new int[] { 1, 2, 3, 5 }; // Присвоение всех значений сразуРаботать с элементами массива можно точно как с переменными. Мы можем их выводить или же устанавливать для них новые значения.

- Для массивов существует несколько дополнительных методов, что позволяют проводить действия над массивом. Рассмотрим несколько из них:
- Length - возвращает количество элементов в массиве. К примеру, создадим массив arr и укажем для него 3 элемента. При вызове функции будет выдано число 3: arr.Length;
- Arrays.Clear - очищает массив и устанавливает в качестве новых значений параметр. Пример:

- Пример:
- int[] arr = new int[] { 1, 2, 3, 5 }; Array.Clear(arr, 0, arr.Length); // Установит повсюду 0 Console.Write(arr[1]); // Выведет значение 0Arrays.CopyTo - копирует один массив в другой.
- Пример:
- int[] arr = new int[] { 1, 2, 3, 5 }; // Основной массив int[] newOne = new int [4]; // Новый массив arr.CopyTo(newOne, 0); // Копирование начиная с элемента под индексом 0 Console.Write(newOne[2]); // Выведет 3
-

Многомерный массив

- Многомерный массив – массив, в котором каждый элемент является другим массивом. На практике очень редко используются массивы с более чем третим уровнем вложенности. То есть массивы, в которых все элементы являются другими массивами и в котором все элементы также другие массивы.

- Мы не будем изучать подобные массивы, так как принцип их построения точно такой же как при работе с двумерными массивами.

- Для создания двумерных массивов необходимо использовать двойные квадратные скобки после типа данных. Выглядит такой массив как матрица, а записывается следующим образом:

- int[,] x = { { 0, 34, 2 }, { 3, 4, 5 } }; Console.Write(x[0, 1]); // Выведет 34 // Можно их сразу не присваивать char [,] symbols = new char[5,5]; symbols [0, 1] = 'A‘;
- Мы видим, что элементы первого массива являются другими массивами. Чтобы выбрать какой-либо объект используйте ту же структуру что и для одномерных массивов, вот только теперь указывайте индекс как первого массива, так и второго:
- x[0, 1] = 1; // Вместо 34 теперь будет 1