Лабораторная работа №1. Ветвление и циклы

Цель работы: научиться разрабатывать программы с использованием условного оператора и циклов.

Краткие теоретические сведения по теме лабораторной работы: Оператор **if (…)** вычисляет выражение в скобках **(…)** и преобразует результат к логическому типу.

Число **0**, пустая строка **""**, **null**, **undefined** и **NaN** становятся **false**. Из-за этого их называют «ложными» («falsy») значениями.

Остальные значения становятся **true**, поэтому их называют «правдивыми» («truthy»).

Оператор **if** может содержать необязательный блок «**else**» («**иначе**»). Он выполняется, когда условие ложно.

```
let year = prompt('В каком году появилась спецификация ECMAScript-2015?', '');

if (year == 2015) {
  alert( 'Да вы знаток!' );
} else {
  alert( 'А вот и неправильно!' ); // любое значение, кроме 2015
}
```

Если нужно проверить несколько вариантов условия, используется блок **else if**.

```
let year = prompt('В каком году появилась спецификация ECMAScript-2015?', '');

if (year < 2015) {
  alert( 'Это слишком рано...' );
} else if (year > 2015) {
  alert( 'Это поздновато' );
} else {
  alert( 'Верно!' );
}
```

В приведённом выше коде, JavaScript сначала проверит **year < 2015**. Если это неверно, он переходит к следующему условию **year > 2015**. Если оно тоже ложно, тогда сработает последний **alert**.

Блоков **else if** может быть и больше. Присутствие блока **else** не является обязательным.

Цикл **while** имеет следующий синтаксис:

```
while (condition) {
  // код
  // также называемый "телом цикла"
}
```

Код из тела цикла выполняется, пока условие **condition** истинно. Например, цикл ниже выводит **i**, пока **i < 3**:

```
let i = 0;
while (i < 3) { // выводит 0, затем 1, затем 2
  alert( i );
  i++;
}
```

Проверку условия можно разместить под телом цикла, используя специальный синтаксис **do...while**:

```
do {
  // тело цикла
} while (condition);
```

Цикл сначала выполнит тело, а затем проверит условие **condition**, и пока его значение равно **true**, он будет выполняться снова и снова.

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

Такая форма синтаксиса оправдана, если нужно, чтобы тело цикла выполнилось хотя бы один раз, даже если условие окажется ложным.

Более распространённый вариант цикла – цикл **for**:

```
for (начало; условие; шаг) {
  // ... тело цикла ...
}
```

Цикл ниже выполняет **alert(i)** для **i** от **0** до **3** (***не*** включительно):

```
for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
  alert(i);
}
```

В данном примере в каждой части цикла **for** выполняются следующие операции:

часть		
начало	`i = 0`	Выполняется один раз при входе в цикл
условие	`i < 3`	Проверяется *перед* каждой итерацией цикла. Если оно вычислится в `false`, цикл остановится.
шаг	`i++`	Выполняется *после* тела цикла на каждой итерации *перед* проверкой условия.
тело	`alert(i)`	Выполняется снова и снова, пока условие вычисляется в `true`.

Обычно цикл завершается при вычислении условия в **false**. Но можно выйти из цикла в любой момент с помощью директивы **break**.

Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока пользователь их вводит, а затем – выдаёт ее:

```
let sum = 0;

while (true) {

  let value = +prompt("Введите число", '');

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

  sum += value;

}
alert( 'Сумма: ' + sum );
```

Директива **break** в строке **(*)** полностью прекращает выполнение цикла и передаёт управление на строку за его телом, то есть на **alert**.

Сочетание «бесконечный цикл + **break**» подходит для тех ситуаций, когда условие, по которому нужно прерваться, находится не в начале или конце цикла, а где-то внутри.

Директива **continue** – «облегчённая версия» **break**. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно **true**). **continue** используется, когда выполнение текущей итерации не требуется. Например, воспользуемся **continue**, чтобы вывести только нечётные числа из диапазона **[0, 10)**:

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

  // если true, пропустить оставшуюся часть тела цикла
  if (i % 2 == 0) continue;

  alert(i); // 1, затем 3, 5, 7, 9
}
```

Для чётных значений **i**, директива **continue** прекращает выполнение тела цикла и передаёт управление на следующую итерацию **for** (со следующим числом). Таким образом **alert** вызывается только для нечётных значений.

Задания для самостоятельной работы:

1. Используя условный оператор, определить знак произведения трех чисел.

   Пример работы программы: даны числа **3**, **–7**, **2**. Программа должна вывести на экран знак «**–**».

2. С использованием цикла перебрать все числа от **1** до **100**. Вместо чисел, кратных **3**, вывести на экран строку «**Fizz**». Вместо чисел, кратных **5**, вывести на экран строку «**Buzz**». Вместо чисел, кратных и **3**, и **5**, вывести на экран строку «**FizzBuzz**». Вывод осуществлять с помощью инструкции **console.log**(число/строка).
   Пример работы программы:
   1
   2
   Fizz
   4
   Buzz
   ...
   13
   14
   FizzBuzz
3. С использованием цикла определить все трехзначные числа Армстронга. Число Армстронга – такое число, сумма кубов цифр которого равна этому числу. Например, число **371** является числом Армстронга, т.к. **3**3 + 7**3 + 1**3 = 371**
4. Написать программу, которая определяет наибольший общий делитель двух целых положительных чисел. Проверить работу программы на двух трехзначных числах.
5. Написать программу, которая определяет сумму всех чисел от **1** до **1000** включительно, являющихся кратными либо **3,** либо **5**.
6. С помощью цикла сформировать строку «**-1-2-3-4-5-6-7-8-9-**».

7. С помощью циклов воспроизвести следующий паттерн:

1
22
333
4444
55555
666666
7777777
88888888
999999999

8. Используя цикл, вычислить:

(1+2)*(1+2+3)*...*(1+2+...+10)