Условные и логические операторы, циклы

Занятие 3

Ход занятия

- Повторение предыдущего занятия
- Условные операторы (if, ?)
- Конструкция switch
- Логические операторы (&&, ||, !)
- Циклы while и for
- Практика

Повтор материала предыдущего занятия

null == undefined	Результат?
null > 0	Результат?
"0" === false	Результат?
undefined == false	Результат?

Повтор материала предыдущего занятия

null == undefined	TRUE // особенность языка JavaScript
null > 0	Результат?
"0" === false	Результат?
undefined == false	Результат?

Повтор материала предыдущего занятия

null == undefined	TRUE // особенность языка JavaScript
null > 0	FALSE // тк 0 не больше 0
"0" === false	Результат?
undefined == false	Результат?

Повтор материала предыдущего занятия

null == undefined	TRUE // особенность языка JavaScript
null > 0	FALSE // тк 0 не больше 0
"0" === false	FALSE // тк это строгое сравнение
undefined == false	Результат?

Повтор материала предыдущего занятия

null == undefined	TRUE // особенность языка JavaScript
null > 0	FALSE // тк 0 не больше 0
"0" === false	FALSE // тк это строгое сравнение
undefined == false	FALSE // особенность языка JavaScript

Условные операторы (if, ?)

...

Инструкция if(*условие*){ *инструкция* }

Инструкция if (...) выполняет инструкцию, если условие выполняется (true).

- При вычислении условия происходит преобразование значения к логическому типу.
- Число 0, пустая строка "", null, undefined и NaN становятся false.
- Все остальные значения приводятся к true.

```
if (true) {
        //выполняемая инструкция
}
```

```
if (false) {
        //пропускаемая инструкция
}
```

Инструкция else{ *инструкция* }

Если условие, заданное в инструкции **if(...)** не выполняется, принимает ложное значение (false). То может быть выполнена другая инструкция, содержащаяся в блоке **else**.

```
if (false) {

        //пропускаемая инструкция
} else {

        //выполняемая инструкция
}
```

Инструкция else if (*условие*) {*инструкция*}

Инструкция типа **else if(...)** предоставляет возможность проверки нескольких условий в единой конструкции.

```
if (false) {

        //пропускаемая инструкция
} else if (false) {

        //пропускаемая инструкция

} else if (false) {

        //пропускаемая инструкция
} else {

        //выполняемая инструкция
}
```

Закрепление материала

Какое сообщение выведет браузер при выполнении представленного фрагмента кода.

```
const age = 20;

if (age  < 18) {

        console.log(Доступ запрещен!);

} else if (age >=18) {

        console.log(Доступ разрешен!);

} else {

        console.log(Укажите свой возраст!);

}
```

Условный оператор “?” (тернарный оператор)

Тернарный оператор **“?”** возвращает значение №1, если условие выполняется (**true**), в противном случае возвращает значение №2.

```
let result = условие ? значение №1 :
значение №2
```

Условный оператор “?” (тернарный оператор)

Тернарный оператор **“?”** возвращает значение №1, если условие выполняется (**true**), в противном случае возвращает значение №2.

```
let result = условие ? значение №1 :
значение №2
```

```
if (условие) {

        значение №1
} else {

        значение №2
}
```

Несколько операторов “?”

```
let result = условие №1 ? значение №1 : условие №2 ? значение №2 : значение №3
```

Большие конструкции сложно читаемы и вложенное использование тернарных операторов внутри других тернарных операторов считается “плохой практикой”.

Конструкция типа “switch”

Конструкция **switch** является аналогом для нескольких блоков **if … else if …** .

В данную конструкцию входит неограниченное количество блоков **case** и один необязательный блок **default**.

```
switch(value){
    case “value1”: // value === value1
        // исполняемая инструкция
        break;
    case “value2”: // value === value2
        // исполняемая инструкция
        break;
    default:
        // исполняемая инструкция
        break;
}
```

Принцип работы “switch”

- Переменная *value* проверяется на **строгое** равенство первому значению *value1*, затем второму *value2* и так далее.
- Если соответствие установлено – *switch* начинает выполняться от соответствующей директивы *case* и далее, до ближайшего *break* (или до конца *switch*).
- Если ни один *case* не совпал – выполняется (если есть) вариант *default*.
- Если *break* нет, то выполнение пойдёт ниже по следующим case, при этом остальные проверки игнорируются.
- Любое выражение может быть аргументом для *switch/case*.
- Несколько вариантов case можно группировать.

Практика

- С помощью условных операторов вывести на экран длину наибольшего отрезка из трех данных (все отрезки разной длины).
- Реализовать систему рекомендаций для пользователя опираясь на значение переменной, хранящей значение температуры в градусах Цельсия.
    - ниже либо равна -30: “Оставайтесь дома!”;
    - от -30 до -10 включительно: “Сегодня холодно”;
    - от -10 до +5 включительно: “Не холодно”;
    - от +5 до +15 включительно: “Тепло”;
    - от +15 до +25 включительно: “Очень тепло”;
    - от +25 до +35: “Жарко”;
    - выше либо равно +35: “Пекло!”;
- С помощью конструкции switch и переменной, хранящей роль пользователя (admin, manager, user ...), выводить на экран информацию о пользователе (информацию любого типа, роль, дату рождения, любимый напиток и тд).

Логические операторы (&&, ||, !)

...

|| (логическое “ИЛИ”)

Для вычисления приводит значение к логическому типу, если оно не является таковым. Используется для проверки условий в инструкции **if ...** .

При выполнении находится первое истинное значение, которое возвращается в исходном виде.

- Вычисляет операнды слева направо.
- При вычислении преобразует значения к логическому типу и возвращает первое **true**.
- Если все значения **false**, то возвращает значение последнего операнда.

```
if (1 || 0) {
        console.log(“Сработало!!”)

}
```

```
const a = 0;

const b = 1;

let result1 = a || b; // result1 = 1

let result2 = a || false || “value” || b; //
result2 = “value”

let result3 = false || “” || 0; // result3 = 0
```

&& (логическое “И”)

Для вычисления приводит значение к логическому типу, если оно не является таковым. Возвращает true, если оба значения истинны.

При выполнении находится первое ложное значение, которое возвращается в исходном виде.

- Вычисляет операнды слева направо.
- При вычислении преобразует значения к логическому типу и возвращает первое **false**.
- Если все значения **true**, то возвращает значение последнего операнда.

```
if (1 && 0) {
        console.log(“Не сработало!!”) //  тк выражение false

}
```

```
const a = 0;

const b = 1;

let result1 = a && b; // result1 = 0

let result2 = b && true && “value”; // result2 = “value”
```

Необходимо запомнить

- Поведение операторов **&& (И)** и **|| (ИЛИ)** похоже между собой. Разница в том, что **&& (И)** возвращает первое **ложное** значение, в то время как **|| (ИЛИ)** возвращает первое **истинное** значение.
- Приоритет оператора **&& (И)** больше, чем у **|| (ИЛИ)**, поэтому он выполняется раньше.

Выражение: a && b || c && d,

можно представить как,

(a && b) || (c && d)

! (логическое “НЕ”)

Оператор принимает один аргумент и выполняет следующие действия:

- Приводит аргумент к логическому типу, если тот не является таковым.
- Возвращает противоположное значение.

Приоритет **! (НЕ)** является наивысшим из всех логических операторов, всегда выполняется перед **&& (И)** и **|| (ИЛИ)**.

```
if (!1) {

        console.log(“Не сработало!!”) //  тк выражение false

} else if (!0) {
        console.log(“Сработало!!”)

} else {

        console.log(“Нет результата!!”)

}
```

Практика

- Что выведет комбинация данных логических операторов, и почему?
  - null || 0 || “” || undefined
  - “яблоко” && true && null && 1
  - 0 || true && “false” || null
  - 0 && true || “false” && null
  - !0 && !!1
  - !(null || !”апельсин” && true)

Циклы while и for

...

Цикл “while”

Применяется для многократного выполнения однотипных задач.

- Каждый шаг цикла называется - итерация.
- Пока условие истинно код из тела цикла будет выполняться.
- Любое выражение или переменная может быть передана как условие цикла: условие while вычисляется и преобразуется в логическое значение.

```
while(true){

    // тело цикла

}

// бесконечный цикл, тк условие всегда истинно
```

Цикл “do while”

Используя специальный синтаксис, **do ... while ...** проверку условия можно расположить под телом цикла. Это позволит выполнить тело цикла хотя бы один раз, даже если условие окажется ложным.

Данная форма редко используется на практике.

```
do{

        // тело цикла

} while(false)

//выполнится одна итерация тк условие ложно
```

Цикл “for”

Является самым распространенным циклом.

- Начало - выполняется один раз при входе в цикл.
- Условие - проверяется перед каждой итерацией. Если условие становится ложным то цикл прекращает свою работу.
- Шаг - выполняется на каждой итерации после тела цикла и перед проверкой условия.

```
for(начало; условие; шаг){

        //тело цикла

}

for(let i = 0; i <= 3; i++){

        console.log(i); //0,1,2,3

}

console.log(i); // выдаст ошибку, переменная i является локальной переменной и доступна только в теле цикла.
```

Цикл “for”

Переменные можно объявлять за телом цикла

```
let i = 0;

for(; i <= 3; i++){

        console.log(i); //0,1,2,3

}

console.log(i); // 4, тк прошло четыре итерации цикла
```

Любая часть цикла может быть пропущена:

```
for(; true; ) // аналог цикла “while”

for(; ;) // бесконечный цикл
```

Прерывание циклов “break”

Обычно цикл заканчивает свою работу при становлении условия ложным.

Но любой из рассмотренных циклов можно принудительно завершить с помощью директивы **break**.

Данная команда полностью прекращает выполнение цикла и передает управление на строку после его тела.

break не может быть использовано в связке с тернарным оператором.

```
while(true){
    break;
}
console.log(i)(“Break!”);
```

Переход к следующей итерации “continue”

При выполнении директивы **continue**, цикл не прерывается, а переходит к следующей итерации, если условие все еще истинно.

continue не может быть использовано в связке с тернарным оператором.

```
for(let i = 0; i <= 3; i++){
    if(i === 2) continue;
    console.log(i); //0,1,3
}
```

Метки для “break” и “continue”

Метки необходимы в случае, если нам необходимо прервать выполнение нескольких уровней цикла.

Метка имеет вид идентификатора с двоеточием перед циклом.

Вызов **break/continue** с меткой в цикле, ищет ближайший внешний цикл с такой меткой и завершает его, передавая управление на следующую строку после его тела.

Редко используемая синтаксическая конструкция.

```
mark: for(....){

    for(...){

        break mark;

    }

}
```

Практика

Решить следующие задачи с использованием циклов while и for.

- Вывести в консоль заданную строку N раз.
- Ежедневно количество доступных автомобилей в салоне уменьшается в два раза. Выяснить, на какой день продаж, количество доступных к покупке авто станет меньше M, если известно, что в первый день продаж всего было N автомобилей.