Помещение значения в переменную называется "**присвоением**" значения, для этого используется операция "=". (Ее не следует путать с операцией сравнения, для этого в JavaScript используется операция "==".)

Let & var

**1. Область видимости.**

В отличие от переменной, объявленной через ключевое слово **var** и видимой внутри всей функции, в которой произошло объявление, переменная, объявленная словом **let**, видна только внутри блока { ... }, в котором она объявлена.

**2. Видимость по времени.**

Переменная, объявленная ключевым словом **let**, видна только после объявления, а переменная, объявленная ключевым словом **var,** может быть доступна в коде и до того места, в котором она объявлена, конечно, в рамках общих правил видимости переменных.

3. При использовании переменной в качестве счетчика цикла объявленная через слово **var** переменная живет в течение всего выполнения цикла и доступна даже после его завершения. Если же использовать объявление через **let**, каждой итерации цикла будет соответствовать своя независимая переменная. Этот вариант использования мы рассмотрим в третьем модуле, при изучении **замыканий**.

Если переменную объявить без использования ключевого слова "var", то она автоматически объявляется глобальной, даже если объявление произведено внутри функции.

**Есть три типа переменных** на данном этапе: локальный var (доступен только в функции), локальный let (доступна только в функции) и глобальная переменная, зачастую пишется вне функции с использованием var (доступна везде). Внутри функции можно объявить глобальную переменную если не упомянуть var или let. **Надеюсь поняли!**

**Тип переменной String**получается если значение, присвоенное переменной, заключить в одинарные или двойные кавычки.

**Логический тип данных - boolean**: это всего лишь два варианта значения переменной - **true**(правда или логическая 1) и **false** (ложь или логический 0). Этот тип данных используется при применении операторов сравнения, логических операций и операторов ветвления.

**Тип данных undefined**переменная имеет в тот момент, когда она объявлена, но еще не инициализирована, то есть ее создали, а значение еще не присвоили.

Более сложные **Арифметические операторы,**начальное значение **y = 5:**

**Остаток от деления (он же - деление по модулю) "%"**выражение x = y % 2 даст результат x = 1; y = 5;  
[Тут](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) вы можете почитать подробнее про остаток от деления.

**Инкремент "++"**Эта операция производит увеличение аргумента на единицу, т.е. выражение x++ будет эквивалентно выражению x = x + 1.  
Порядок применения инкремента в javascript имеет значение, например:

* выражение x = ++y даст результат x = 6, y = 6, так как **вначале** увеличивается значение переменной y на единицу и затем переменной x присваивается значение переменной y;
* а выражение x = y++ даст результат x = 5, y = 6, поскольку в данном случае операция инкрементирования (увеличения) происходит ПОСЛЕ того как переменной х присвоили значение из переменной у.

**Декремент "--"**А эта операция производит уменьшение значения переменной на единицу, т.е. х-- эквивалентно x = х - 1. Порядок применения декремента в javascript также имеет значение, например:

* выражение x = --y даст результат x = 4, y = 4 - тут мы вначале уменьшаем на единицу переменную y, а затем присваиваем переменной x значение переменной y;
* а выражение x = y-- даст результат x = 5, y = 4 -  так как операция декремент была выполнена ПОСЛЕ того как переменной х присвоили значение из переменной у.

<script>  
 i = 5;  
 i = ++i + ++i;  
 alert(i); // 13  
</script>

1) присваимваем i значение 5

2)разбиваем строку на три шага: первый шаг: префиксная операция ++i увеличивает i на единицу и присвает новый результат переменной, т.е. 6. Второй шаг: снова префиксная операция увеличивает переменную i(а она уже равна 6), и получаем уже 7. Третий шаг: сложение 6+7. Получаем 13 и присваиваем это переменной i.

Отличие ++i от i++ в том, каким станет i после операции. В одном случае i+1, во втором старое значение i.

Иными словами:

y=++x даёт нам у=х+1, а х=х+1.

У=х++ даёт нам у=х+1, а х=х.

Т.е. в таких простых выражениях игреку пофигу пост или префикс. А вот для икса это имеет значение. В сложных же выражениях уже есть разница.

**Операторы присваивания бывают такие:**

В примерах будем рассматривать x = 10; y = 5;

**Оператор =**  это обычный оператор присваивания.  
Выполнение x = y приведет к x = 5.

Некоторые арифметические операторы можно использовать вместе с оператором присваивания:

**Оператор +=**  Это присваивание со сложением  
Выполнение x += y приведет к x = 15, эквивалентно x = x + y

**Оператор -=**  Это присваивание с вычитанием  
Выполнение x -= y приведет к x = 5, эквивалентно x = x - y

**Оператор \*=**  Присваивание с умножением  
Выполнение x \*= y приведет к x = 50, эквивалентно x = x \* y

**Оператор /=**  Присваивание и деление.  
Выполнение x /= y приведет к x = 2, эквивалентно x = x / y

**Оператор %=**  Присваивание с операцией "остаток от деления"

Выполнение x %= y приведет к x = 0, эквивалентно x = x % y

Все следующие примеры рассматриваем при x = 5

Оператор **равно ==**  
Эта операция выполняется с преобразованием типов, т.е. если один из операндов - число три, а второй - символ тройки, то выражение 3 == '3' будет истинным, поскольку второй операнд будет приведен к типу первого, если это допускается.  
(x == 8 будет false, x == y будет true при y = '5' )

Оператор **строго равно ===**  
Это сравнение будет истинным без учета преобразования типа  
( x === 5 будет true, x === '5' будет false)

Оператор **не равно !=** (x != 8 будет true)

Оператор **строго не равно !==**(возвращает **true** если операнды строго не равны или имеют разный тип.)

Оператор **больше чем >** (x > 8 это false)  
Оператор **меньше чем <** (x < 8 это true)

Оператор **больше или равно >=**  
(x >= 8 это false)  
Оператор **меньше или равно <=**  
(x <= 8 это true)

И наконец **логические операторы**, их немного.  
  
В данных примерах x = 6; y = 3;

**Оператор &&** (Логическое И)  
x < 10 && y > 1 это true

**Оператор ||** (Логическое ИЛИ)  
x == 5 || y == 5 это false

**Оператор !** (Логическое НЕ)  
!(x == y) это true

***Приоритетность***

Ну и наконец, перед тем как закончить обзор основных операторов, необходимо осветить вопрос **приоритета** их выполнения. Все операторы делятся на группы, у которых различный приоритет. Те, у которых приоритет выше, выполняются первыми.

**1 - Самым высоким приоритетом обладают операторы группировки. В данном случае это скобки "(" и ")"**

i = (a + b) \* c

/\*

В данном случае группировка имеет более высокий приоритет чем умножение,

поэтому сначала выполнятся все операции внутри скобок

\*/

2 - За ними следуют операторы **постфиксного** (указанного ПОСЛЕ переменной) инкремента и декремента

a-- //Постфиксный декремент

b++ //Постфиксный инкремент

3 - Следующими по приоритету идут **префиксные** инкремент и декремент а также логическое отрицание

--x //Префиксный декремент

++y //Префиксный инкремент

!z //Логическое отрицание

4 - За ними идет **возведение в степень**

x = y \*\* z //Возведение в степень

5 - Следующая группа: **умножение, деление и остаток от деления**

x = y \* z //Умножение

x = y / z //Деление

x = y % z //Остаток от деления

6 - Еще меньшим приоритетом обладают **операции сложения и вычитания**

x = y + z //Сложение

x = y - z //Вычитание

7 - Следом идут **операции сравнения**

> //Больше

>= //Больше или равно

< //Меньше

<= //Меньше или равно

8 - Затем **операции равенства**

== //Равно

!= //Не равно

=== //Строго равно

!== //Строго не равно

9 - **Логические операции (и/или)**

& //Побитовое И

| //Побитовое ИЛИ (побитовые операции мы не рассматривали в данном уроке)

&& //Логическое И

|| //Логическое ИЛИ

10 - И наконец последними выполняются **операции присваивания**

= //Присваивание

+= //Присваивание со сложением

-= //Присваивание с вычитанием

\***\*= //Присваивание с возведением в степень**

**\***= //Присваивание с умножением

/= //Присваивание с делением

%= //Присваивание с остатком от деления

**Ветвление**

В данном уроке мы рассмотрим варианты изменения поведения программы в зависимости от значений заданного условия. Эти конструкции так и называются - **условные операторы**.

В JavaScript имеется два варианта условных операторов: конструкция if ... else и конструкция switch

Первая из них используется для выполнения некоторого кода, если указанное условие истинное.

if (условие) {

// код для выполнения если условие истинно

}

Давайте рассмотрим пример:

var x = 10;

var y = 50;

if (x > 11) {

x += y;

}

В данном примере оператор проверяет условие и если х больше 11, то переменная x увеличивается на величину y.

В JavaScript также имеется так называемый **условный оператор**(его также иногда называют ***тернарный* оператор**), он присваивает значение переменной на основе некоторого условия.  Синтаксис условного оператора выглядит так:

***имя\_переменной = (условие) ? значение1 : значение2;***

Это означает, что в результате выполнения выражения в ***переменную*** будет записано либо ***значение1*** либо ***значение2***, в зависимости от результата проверки истинности ***условия.***

Например, выражение

hiJack = (friend == "Jack") ? "Hi, Jack!" : "Hi";

console.log(hiJack);

приведет к выводу "**Hi, Jack!**" если переменная **friend** будет иметь значение **"Jack"** или к выводу **"Hi"** при любом другом значении.

Существует еще один вариант использования данного оператора - **когда можно выполнять некоторый код если условие истинно и другой код, если условие ложно.** Синтаксис выглядит следующим образом:

if (условие) {

  // код для выполнения если условие истинно

} else {

  // код для выполнения если условие ложно

}

В качестве примера рассмотрим такой вариант:

var x = 10;

var y = 50;

if (x > 11) {

x += y;

} else {

x -= y;

}

В данном случае если x > 11 то он увеличивается на величину y, а если меньше 11 - то уменьшается на эту же величину.

Также нужно отметить что конструкции if ... else могут продолжать друг друга. Например:

if (x > 11) {

  x += y;

} else if (x == 4) {

  x -= y;

} else {

  x = y;

}

В данном примере х увеличит значение на величину y если он больше 11, уменьшит значение на величину y если равен 4 и примет значение y во всех остальных случаях, т.е. если *x*⩽11 и не равен 4.

получается чтобы проверить работоспособность кода нужно объявить переменные **х, у;**  
let x = 3;  
let y = 12;  
if (x > 11) {  
  x+=y;  
} else if (x==4)  {  
  x-=y;  
} else {  
   x=y;  
}

**Второй вариант управления ходом выполнения программы это конструкция switch.**

Данная конструкция позволяет выбирать один из многих вариантов кода на выполнение в зависимости от условия (обычно это значение одиночной переменной). Синтаксис команды выглядит следующим образом:

switch(n) {

case 1:

// блок кода 1;

break;

case 2:

// блок кода 2;

break;

// .......

// другие варианты case

// .......

default:

// блок кода если не подошло ни одно условие;

}

Выражение условия (то, что находится в скобках после switch) вычисляется один раз, затем результат выражения сравнивается с каждым значением case в данной конструкции (т.е. в данном случае содержимое переменной n сравнивается со значениями 1, 2 и т.д.).  
Если условие совпадает, то выполняется блок кода, связанный с соответствующим case. Если ни одно условие не подошло, то выполняется код, указанный в блоке **default**, если он есть. Для выхода из конструкции используется команда **break**. Если ее не указывать, автоматически выполнится блок кода в следующем case и т.д.

**Давайте рассмотрим пример использования конструкции switch:**

var a = 3; //объявляем переменную по которой будем делать выборку

var x = ""; //объявляем переменную в которую будем записывать результат

switch (a) {

case 1: // если a равна единице то записываем

x = "Переменная a равна единице!"; // в переменную x фразу "Переменная a равна единице!"

break;

case 2:

x = "Переменная a равна двум!";

break;

case 3:

x = "Переменная a равна трем!";

break;

default:

x = "Переменная a не равна 1, 2 или 3!";

}

// В итоге результат выполнения switch оказывается в переменной x