1. **Чему будет равен результат**

1.1. `alert( 2 ** 3 );`
1.2. `alert( 1 + 3 ** 2 );`
1.3. `alert( 25 ** (1/2) - 1);`
1.4. `alert( Math.floor(8/2));`
1.5. `alert( Math.floor(25/10));`
1.6. `alert( Math.floor(16/3));`
1.7. `alert( Math.round(25/10));`
1.8. `alert( Math.round(8/2));`
1.9. `alert( Math.round(16/3) - 1);`
1.10. `alert( Math.round(16/3) - 1);`

# Оператор

- с одним операндом называется **унарным**,
- с двумя – **бинарным**,
- с тремя – **тернарным (** условный оператор.**)**.

**Бинарная операция** использует два операнда, один перед оператором и другой за ним:

***operand1* operator *operand2***

Например:

```
3+4
x*y
d = 'Привет'
f1 % f2
```

# Оператор присваивания как часть выражения

Оператор присваивания возвращает значение и его можно использовать как часть выражения

Например

```
x = 1;

y = 5 - (x = x + 2);


alert( x ); // 3

alert( y ); // 2
```

```
let a = 1;
let b = 2;
let c = 3 - (a = b + 1);
alert( a );  //?
alert( c );  //?
```

## Сложение строк при помощи бинарного +

Бинарный оператор `'+'` объединяет строки.

```
let s = "Hello" + "World";

alert(s); // HelloWorld
```

Если один операнд является **строкой**, то второй будет также **преобразован в строку**.
Сложение и преобразование строк — это особенность бинарного плюса +

```
alert( '1' + 2 ); // "12"

alert( 2 + '1' ); // "21"
```

```
alert(2 + 2 + '1' ); // будет "41", а не "221"
```

```
alert('2' + 2 + 1 ); // будет  "221"
```

операторы работают один за другим!!!

Другие арифметические операторы работают только с числами и всегда преобразуют операнды в числа.

```
alert( 6 - '2' ); // 4, '2' приводится к числу

alert( '6' / '2' ); // 3, оба операнда приводятся к числам
```

**Унарная операция** использует один операнд, перед или после оператора:

| *operator* **operand** | **operand** *operator* |
| --- | --- |

Например:

```
x++
 ++x
x- -
+prompt(“10”)
=*x
```

Например

```
x = 1;

y = 5 - (x = x + 2);


alert( x ); // 3

alert( y ); // 2
```

Например

```
x = 1;

y = 5 - (x += 2);


alert( x ); // 3

alert( y ); // 2
```

## 1. Чему будут равны переменные

1.1.
```
num = 15.21;
x=num.toFixed(1) + 5;
```

1.2.
```
num = 15.21;
z=+num.toFixed(1) + 5;
```

1.1.
```
n = 7;
n -= 5;
n *= 2;
```

1.2.
```
a = 2;
x = 1 + (a *= 2);
```

1.3.
```
a = b = c = 2 + 2;
```

1.4.
```
a=+'2' + 2 + 1 ;
```

# Инкремент/декремент

**Инкремент** ++ увеличивает **переменную** на 1:

1) **«Префиксная форма» — это когда оператор идёт перед переменной** `++counter;`

Префиксная форма возвращает новое значение

```
counter = counter  + 1;    =>      ++counter;
```

# Префиксная форма инкремента ++counter

```
counter = 2;

a = ++counter;

alert(a); // 3

alert(counter); // 3
```

```
counter = 2;

counter = counter  + 1;

a = counter;

alert(a); // 3

alert(counter); // 3
```

counter меняется на +1, затем переменной a присваивается значение counter

# Постфиксная форма инкремента counter++

«Постфиксная форма» - оператор идёт после переменной — это : **counter++.**

постфиксная форма возвращает старое (до изменения числа).

```
let counter = 2;

let a = counter++;

alert(a); // 2

alert(counter); // 3
```

```
counter = 2;

a = counter;

counter = counter + 1;

alert(a); // 2

alert(counter); // 3
```

переменной a присваивается значение counter, а затем counter меняется на +1

Пример использования:

```
let x = 1;

alert( 3 * ++x );  //6
```

Пример использования:

```
let x = 1;

alert( 3 * x++ );  //3
```

# Декремент

**Декремент** -- уменьшает переменную на 1:

1) Префиксная форма

```
let counter = 2;

counter--;        // работает как counter = counter - 1, просто запись короче

alert( counter ); // 1
```

2) Постфиксная форма

```
let counter = 2;

--counter;        // работает как counter = counter - 1, просто запись короче

alert( counter ); // 1
```

# Задачи

1)
```
a = 1, b = 1;

c = ++a;

d = b++;


alert( ++a );

alert( b++ );


alert( a );

alert( b );
```

Какой результат будет у выражений ниже?

```
6 / "3"

"2" * "3"

4 + 5 + "px"

"$" + 4 + 5

"4" - 2

"4px" - 2

"-9" + 5

"9" - 5

16 + 4 + "Volvo"

"Volvo" + 16 + 4;
```