“5” – 1 → 5 – 1 → 4

“5” == 5 → true

“5” + 1 → “51”

5 + “1” → “51”

“5” + 1 - 1 → “51” -1 → 51 – 1 → 50
“5” – 1 + 1 → 5 -1 +1 → 4 + 1 → 5
“5” + (-1 + 1) → “5” + 0 → “50”
“5” + (-2 + 1) → “5”+ (-1) → “5-1”

```
x = “5”;
y = 5;
```

x==y → true

(x+1) == (y+1) → “51” == 6 → false

false == 0 → true
false == “” → true
“” == 0 → true

```
x = false;
if(x) { do A;}
else {do B; } → do B;
```

```
if(x==false) { do A;} → do A;
else {do B; }
```

```
x = 0;
if(x) { do A;}
else {do B; } → do B;
```

```
x = “”;
if(x) { do A;}
else {do B; } → do B;
```

“0” == 0 → true
0 == “” → true
“0” == “” → false

“0” == 0 → true
0 == false → true
“0” == false → true

x = “0”;
if(x) { do A;} → do A;
else {do B; }

Rule 1: Anything that can be converted to false must be treated as false in a condition.

Rule 2: if a string is used as condition, then it is treated as true if the string is non-empty and it is treated as false if it is the empty string

Those rules conflict in the case of the string “0”. Javascript uses then rule 2.

0 == false → true
(!0) == true → true

“0” == false → true
(!“0”) == false → true

5 == false → false
5 == true → false

x = 5;
if(x) { do A;} → do A;
else {do B; }

```
null  === undefined → false
null == undefined → true


null == false → false
null == true  → false

x = null;
if(x) { do A;}
else {do B; } → do B;

undefined == false → false
undefined == true  → false

x = undefined;
if(x) { do A;}
else {do B; } → do B;
```

“five” + 1 → “five1”

“five” – 1 → NaN

typeof NaN → number

NaN === NaN → false

NaN == NaN → false

X=NaN;

```
if (x==NaN) { do A; }
else { do B; }   → always do B; It does not work.
```

```
if (isNaN(x)) { do A; }  → do A; I does test for NaN
else { do B; }
```

```
if (!(x===x)) { do A; }  → do A; I does test for NaN
else { do B; }
```

```
if (!(x==x)) { do A; }   → do A; I does test for NaN
else { do B; }
```

```
if (x!==x) { do A; }     → do A; I does test for NaN
else { do B; }
```

```
if (x!=x) { do A; }      → do A; I does test for NaN
else { do B; }
```

Considering the piece of code in C#:

```
if (x==x) { do B; }
else { do A; }
```

we can say that this code will always execute B.

(X) True
( ) False

Considering the piece of code in Javascript:

```
if (x==x) { do B; }
else { do A; }
```

we can say that this code will always execute B.

( ) True
(X) False (because x may be NaN)

Questions

The piece of code:

```
if (x) { A; }
else { B; }
```

is equivalent (hold the same result) to

```
if (x==true) {  A; }
else { B; }
```

( ) true
(x) false (ex: x=5 → A in the first code and B in the second one)

==========

The piece of code:

```
if (!x) { A; }
else { B; }
```

is equivalent (hold the same result) to

```
if (x==false) {  A; }
else { B; }
```

( ) true
(X) false (x=null → A in the first code, B in the second code
x= “0” → B in the first code, A in the second code)

==========

The piece of code:

```
if (x==true) { A; }
else { B; }
```

is equivalent (hold the same result) to

```
if ((!x)==false) { A; }
else { B; }
```

( ) true
(x) false (x=5 → B in the first code, A in the second code
X= “0” → B in the first code, A in the second code)

==========

The piece of code:

```
if (x==false) { A; }
else { B; }
```

is equivalent (hold the same result) to

```
if ((!x)==true) { A; }
else { B; }
```

( ) true
(x) false (x=null → B in the first code, A in the second code
x= “0” → A in the first code, B in the second code

==========

The piece of code:

```
if (x==false) { A; }
else { B; }
```

is equivalent (hold the same result) to

```
if ((!x)==false) { B; }
else { A; }
```

( ) true
(x) false (x=null → B in the first code, A in the second code
x= “0” → A in the first code, B in the second code

==========

Considering the piece of code below:

```
if (x == x ) { B; }
else { A; }
```

we can say that it will always execute B.

( ) true
(x) false (x=NaN)

The lines below are equivalent:

```
if ((x==0) || isNaN(x)) { x = -1;}

x = ((x==0) || isNaN(x))? -1 : x

x= x || -1;
```

==========

The lines below are equivalent:

```
if (x== “”) { x = “!?”;}

x = (x == “”)? “!?” : x

x= x || “!?”;
```

==========

document.write(…) → it writes to the **input** of the browse, without changing line.

document.writeln(…) → it writes to the **input** of the browse, and change line.

==========

$$\lambda(x, y)(x + y)$$

```
(x,y) => { return x+y;  }

(x,y) => x+y

(x) => 2*x

x => 2*x
```