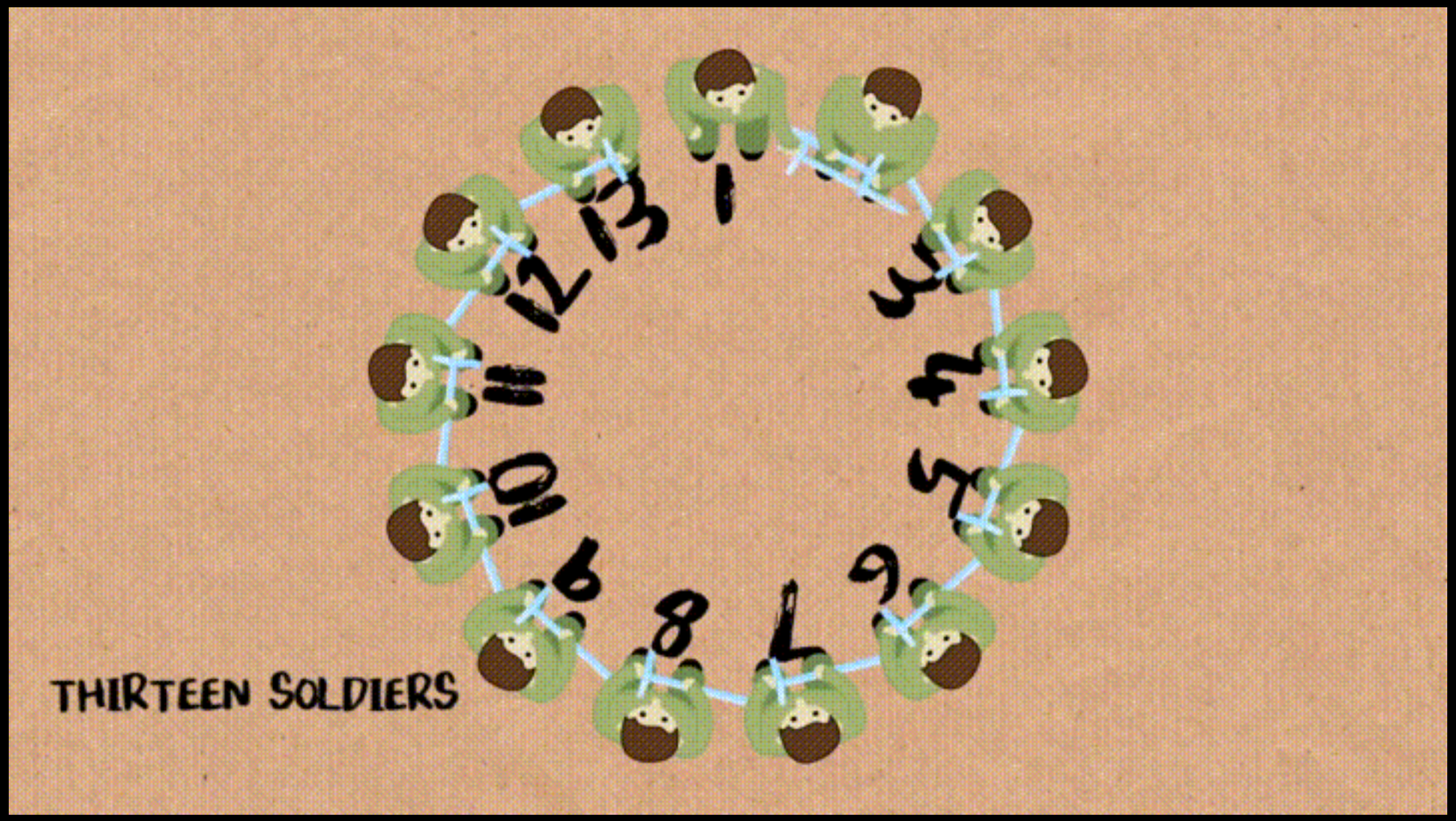
\\W//ORKSHOP

CONDITIONALS & LOOPING

If you need wisdom,
ask our generous God,
and he will give it to you.
--- He will not rebuke you for asking ---

JAMES 1:5

THE JOSEPHUS PROBLEM



WHICH POSITION SURVIVES

when there are N-number of people?

WHAT IS THE FORMULA?

HIGHER ORDER THINKING SKILLS

(HOTs)

HIGHER ORDER THINKING

"Why do I have to eat my vegetables?"

LEVEL 0	*REJECT THE QUESTION*	"Don't ask me." "Because I said so."
LEVEL 1	*RESTATE QUESTION AS RESPONSE*	"Because you have to eat your vegetables."
LEVEL 2	*ADMIT IGNORANCE/ PRESENT INFORMATION*	"I don't know, but that's a good question."
LEVEL 3	*ENCOURAGE TO SEEK RESPONSE THROUGH AUTHORITY*	"Let's google it." "Who might know the ans.?"
LEVEL 4	*ENCOURAGE BRAINSTORMING ALTERNATIVE EXPLENATIONS*	"Let's think of some possible answers. Maybe …, or maybe …, or maybe …. …"
LEVEL 5	*LEVEL 5 + MEANS OF EVALUATING THEM*	Level 5 + "how do we assess these possibilities?"
LEVEL 6	*LEVEL 6 + FOLLOW-THROUGH ON EVALUATIONS*	Level 6 + "Let's research for X days, then back to evaluate together."

HIGHER ORDER THINKING

Categories of the Cognitive Domain

	BLOOM TAXONOMY (1956)	ANDERSON-KRATHWOHL REVISION (2000)
LEVEL 1	KNOWLEDGE	REMEMBERING *using memory to produce*
LEVEL 2	COMPREHENSION	UNDERSTANDING *constructing meaning from different functions*
LEVEL 3	APPLICATION	APPLYING *applying to models/simulation*
LEVEL 4	ANALYSIS	ANALYZING *deconstructing materials to highlight correlations of parts*
LEVEL 5	SYNTHESIS	EVALUATING *judge/check/critique results*
LEVEL 6	EVALUATION	CREATING *synthesize parts to something new*

HIGHER ORDER THINKING

Instruction Keywords

HOT skills

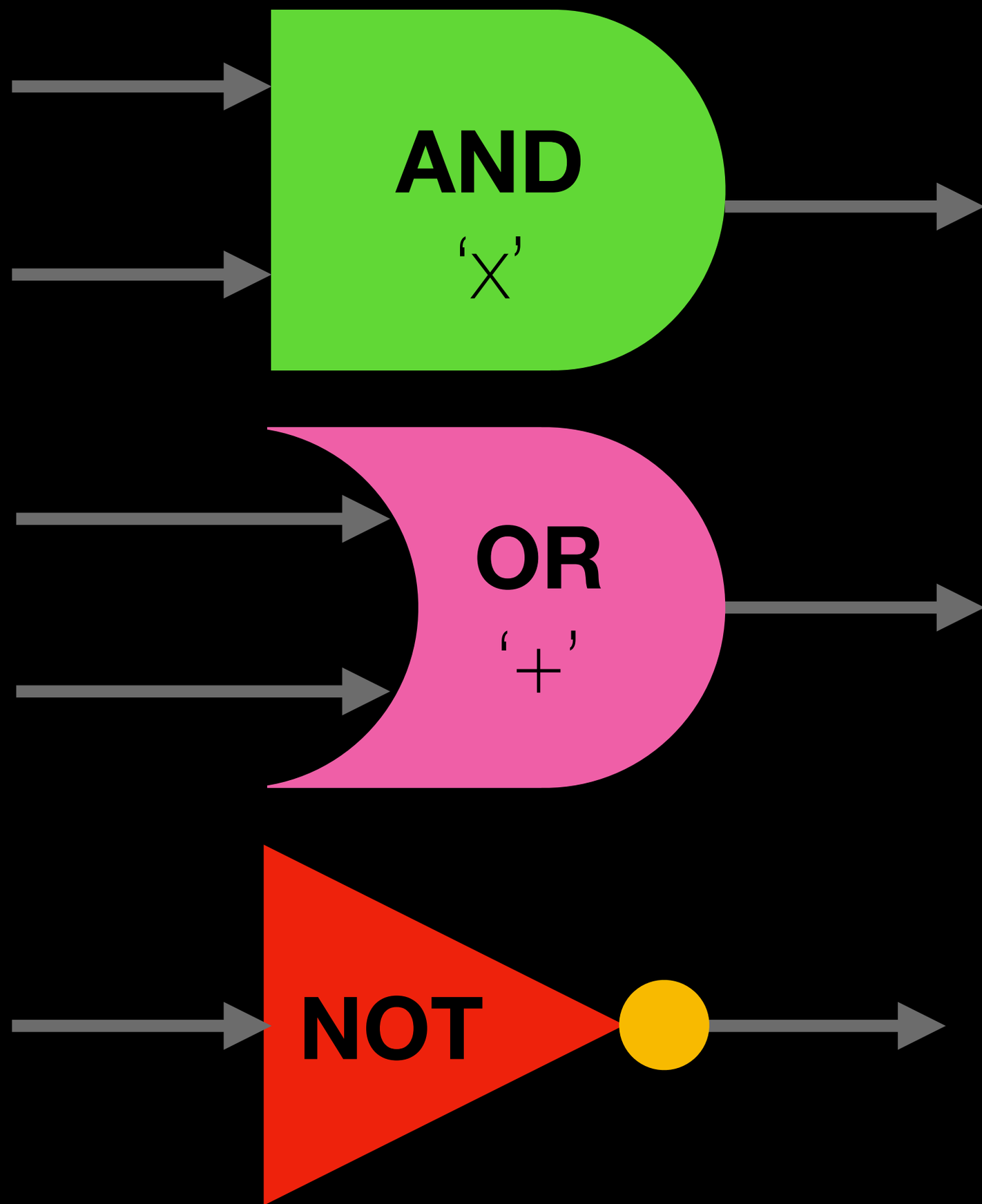
Lower OT / LOT skills

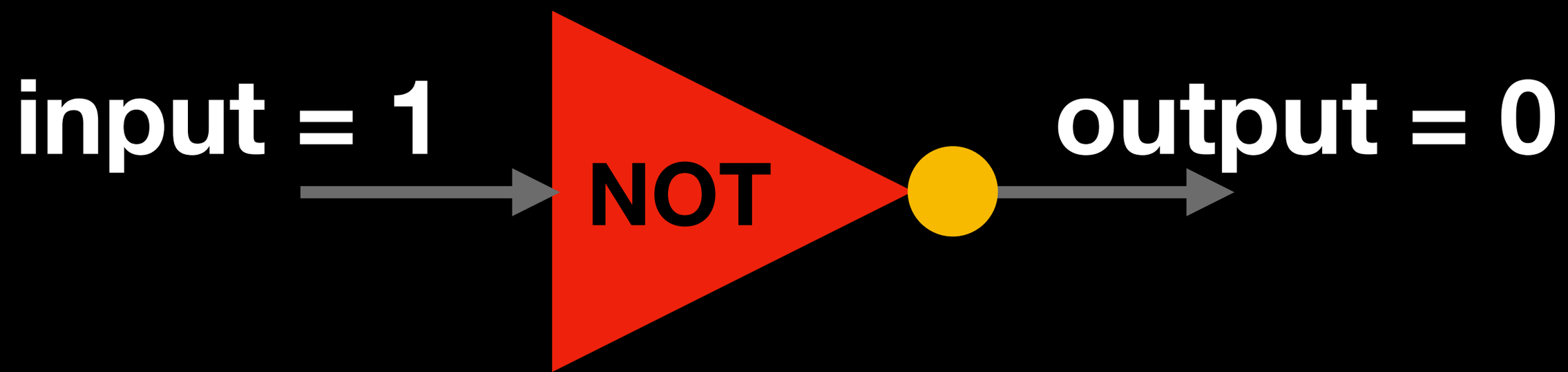
					CREATING
				EVALUATING	adapt, anticipate, arrange, assemble, categorize, collaborate,
			ANALYZING	appraise, argue, assess, choose, compare, conclude, contrast,	collect, combine, comply, compose, construct, create,
		APPLYING	advertise, analyze, appraise, breakdown, calculate, categorize,	consider, convince, critique, debate, decide, defend, describe,	design, develop, devise, explain, express, facilitate, formulate,
	UNDERSTAND-ING	act, administer, articulate, apply, calculate, chart, collect,	classify, compare, conclude, connect, contrast, correlate,	discriminate, distinguish, editorialize, estimate, evaluate,	generate, imagine, infer, intervene, justify, make, manage, negotiate,
REMEMBER -ING	ask, associate, cite, classify, compare, convert, defend,	compute, change, choose, complete, construct, demonstrate,	criticize, debate, deduce, devise, diagram, differentiate,	explain, find errors, grade, interpret, judge, justify, measure, order,	organize, originate, plan, prepare, propose, rearrange, reconstruct,
arrange, define, describe, duplicate, identify, locate, label, list	describe, discuss, distinguish, demonstrate, discover,	discover, dramatize, develop, establish, examine, explain,	discriminate, distinguish, dissect, divide, estimate,	persuade, predict, rank, rate, recommend, reframe, revise, score,	relate, reorganize, revise, rewrite, schematize, set up,
match, memorize, name, order, outline, quote, recognize, relate,	differentiate, estimate, explain, express, extend, examples,	employ, illustrate, interpret, judge, list, manipulate, modify,	evaluate, examine, experiment, explain, focus, identify, illustrate,	select, support, rewrite, set up, summarize, synthesize, tell, value,	simulate, solve, speculate, structure, support, summarize,
recall, record, repeat, reproduce, select, state, tell, underline, vizualize	group, identify, indicate, infer, illustrate, judge, paraphrase, predict,	operate, practice, predict, prepare, produce, relate, record,	infer, inspect, inventory, model, order, organize, outline, plan, point out,	weight, write	synthesize, test, tell, validate
	recognize, restate, rewrite, review, select, summarize, show, tell,	simulate, schedule, show, sketch, solve, teach, transfer, utilize,	prioritize, question, relate, select, separate, subdivide, survey, test		
	translate, trace, transform	use, write			

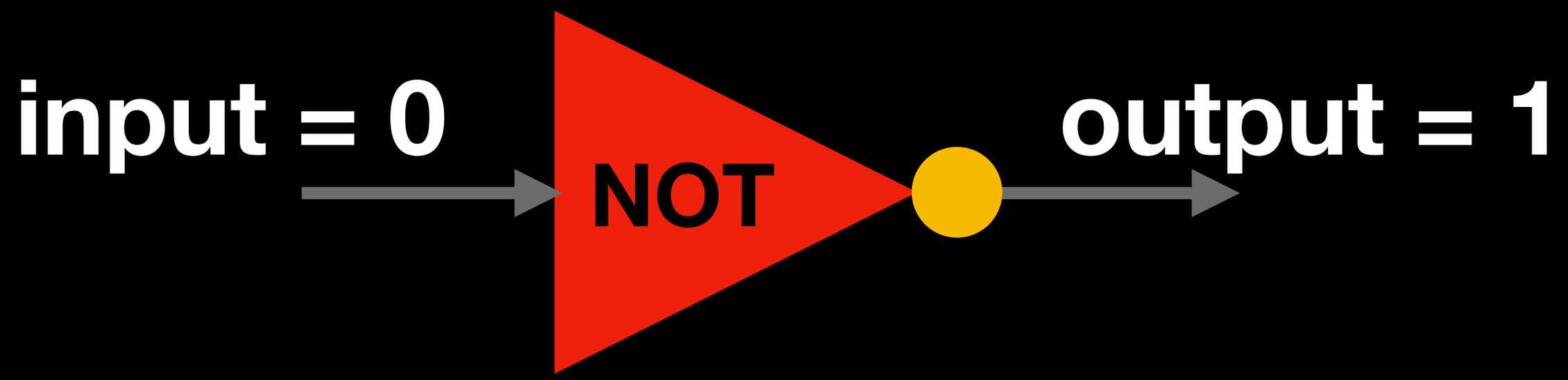
CONDITIONALS

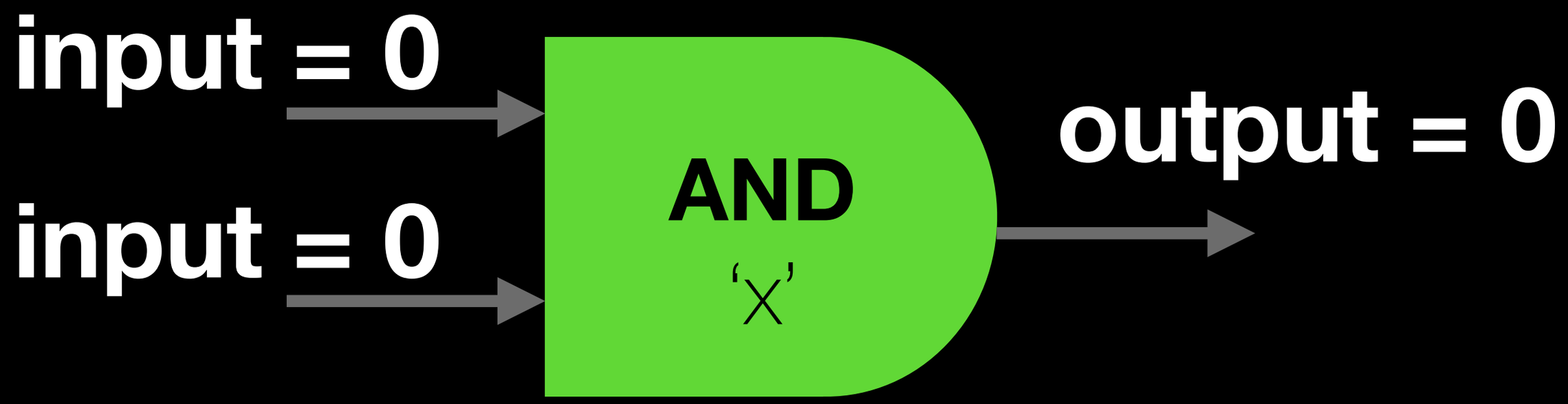
LOGIC GATES

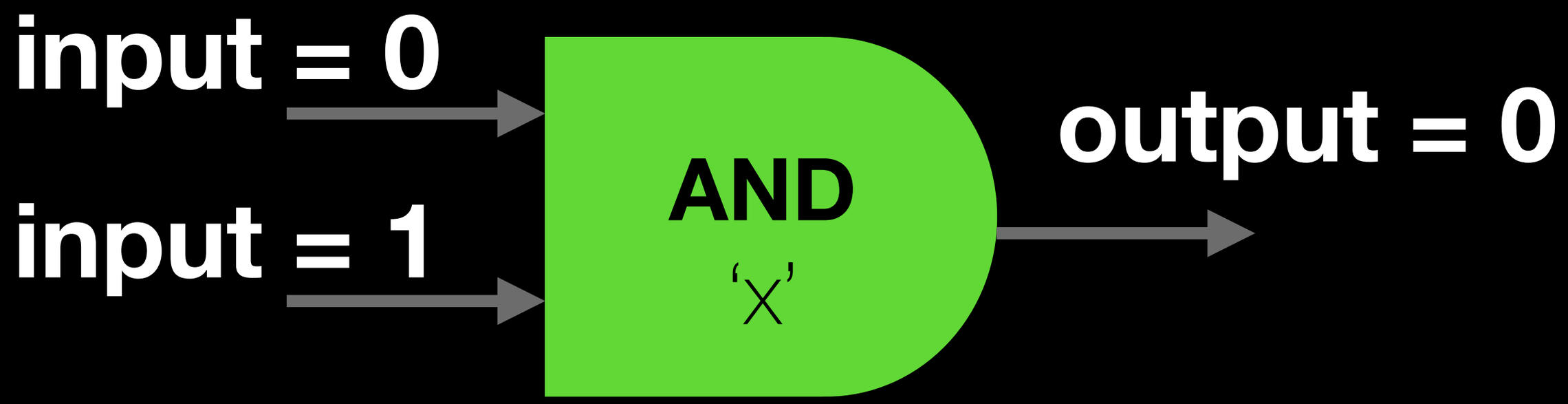
YES	NO
1	0
TRUE	FALSE

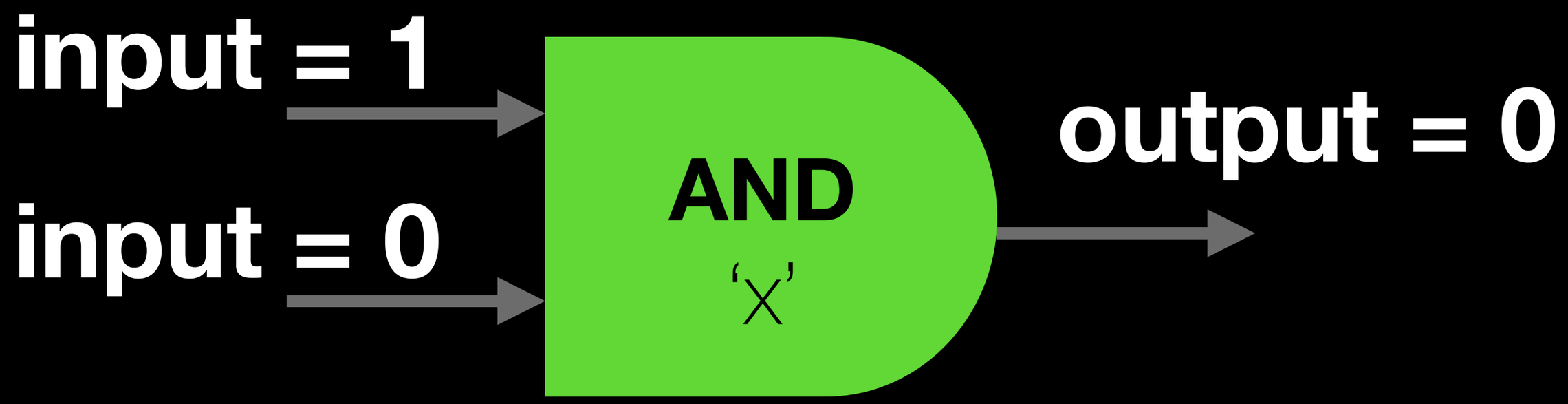

AND
'x'
OR
'+'
NOT

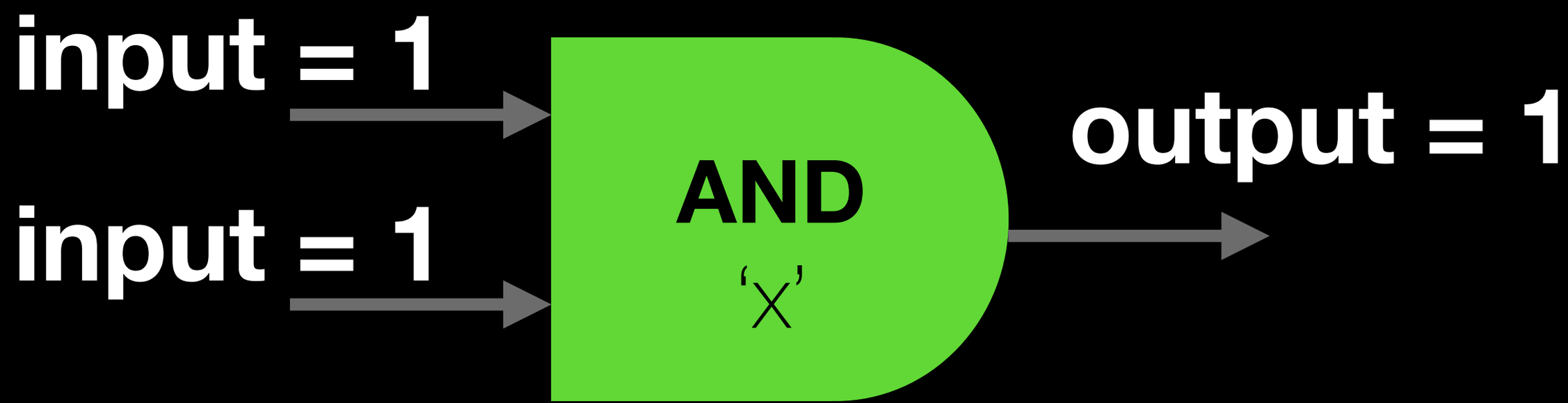

input = 1
NOT
output = 0

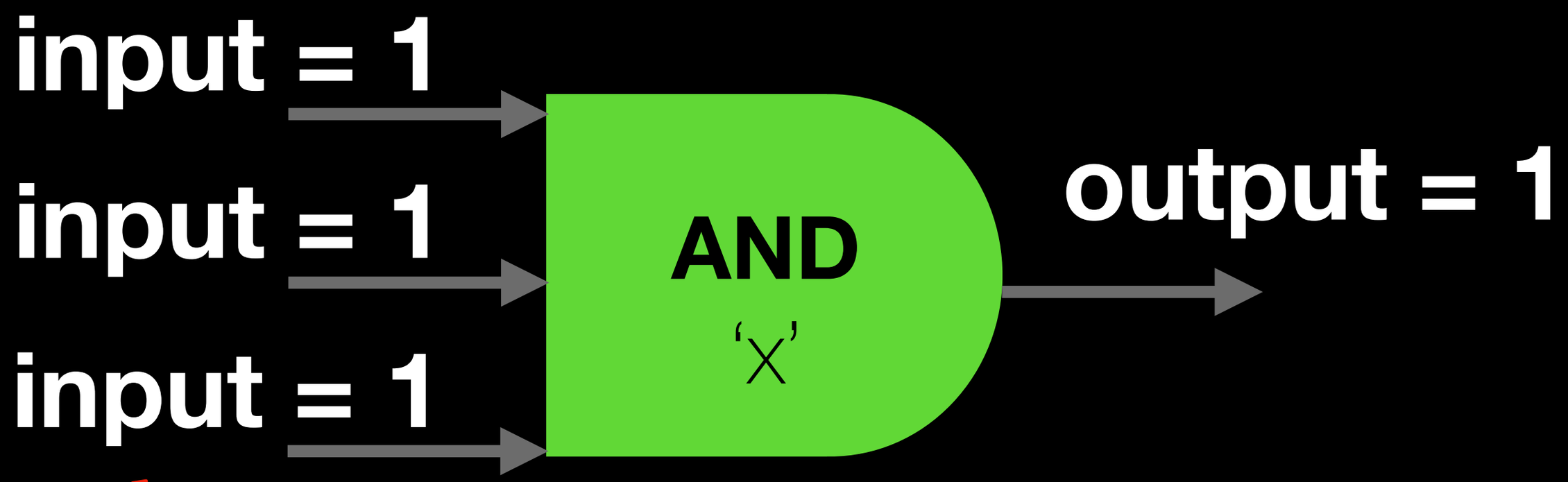

input = 0
NOT
output = 1


input = 0
input = 0
AND
'X'
output = 0

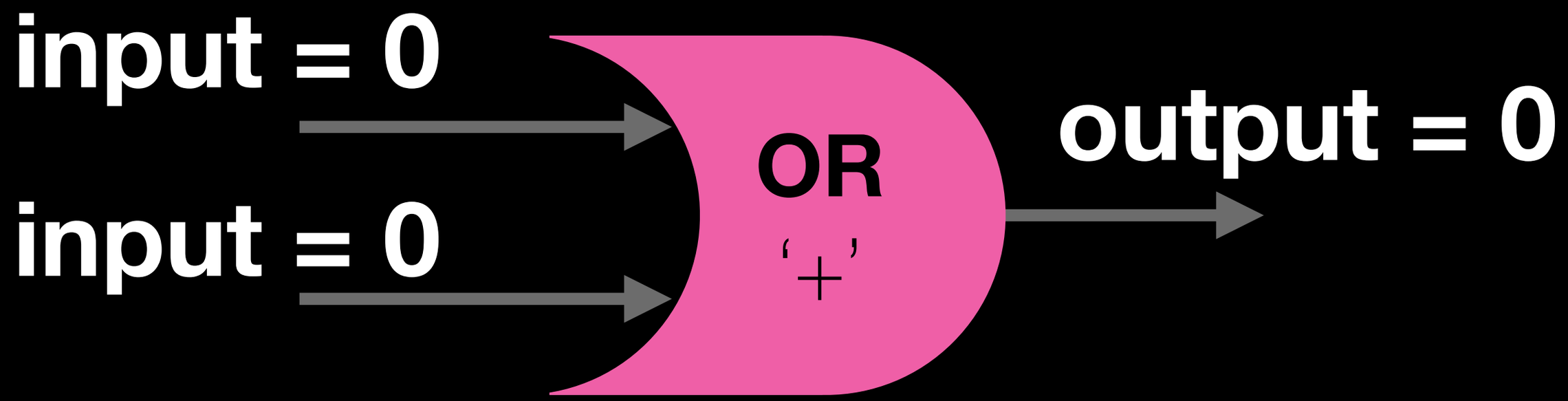

input = 0
input = 1
AND
'X'
output = 0

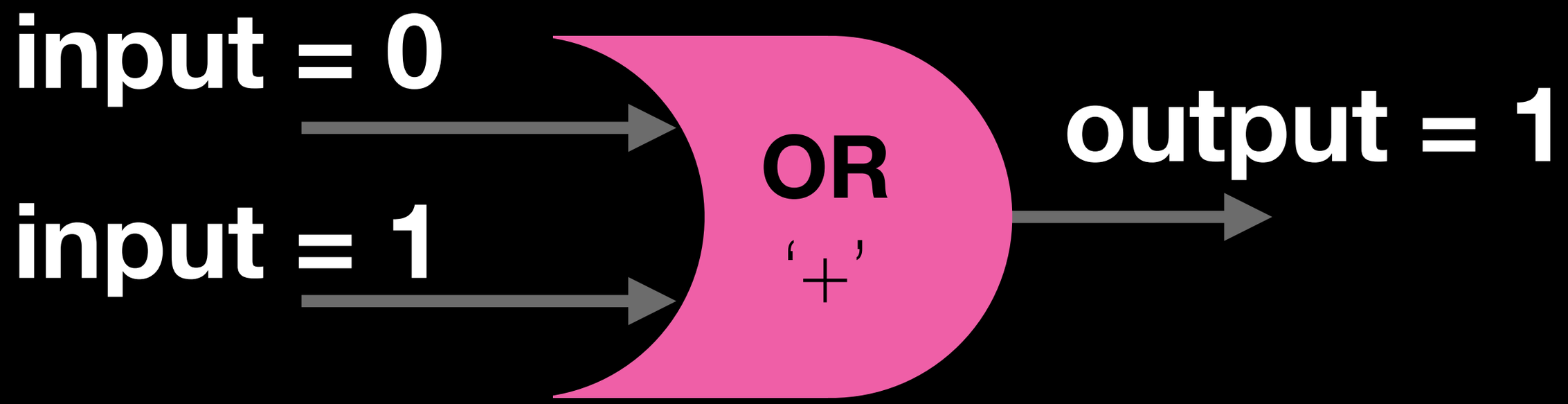

input = 1
input = 0
AND
'X'
output = 0

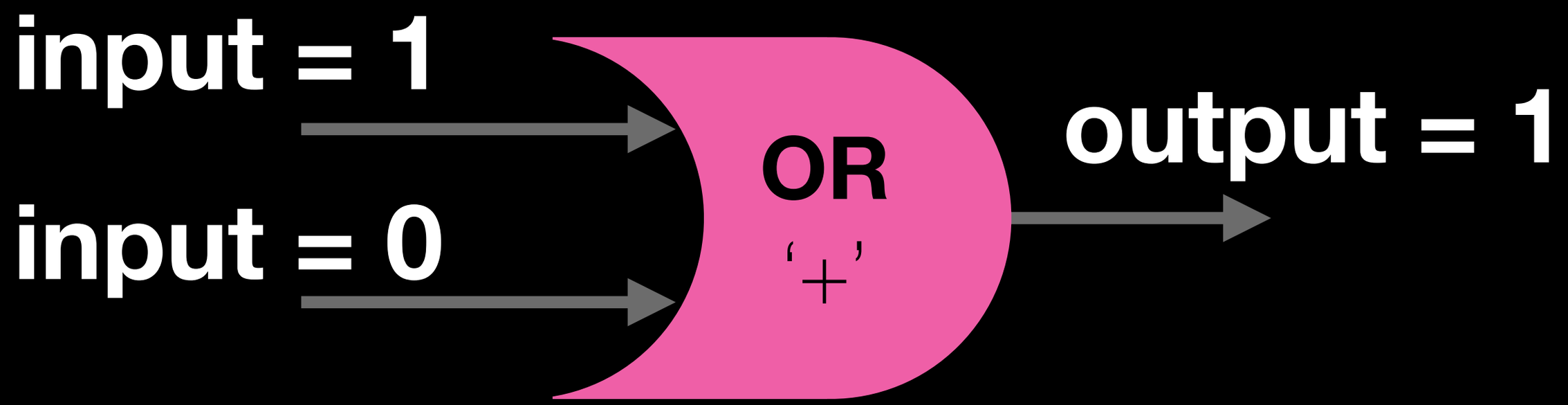

input = 1
input = 1
AND
'X'
output = 1

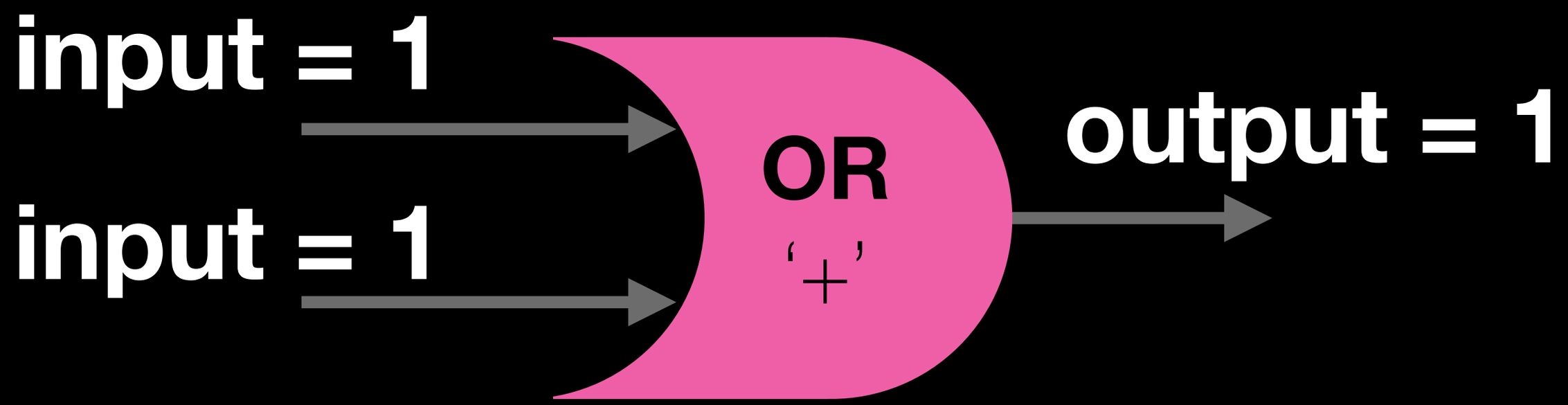


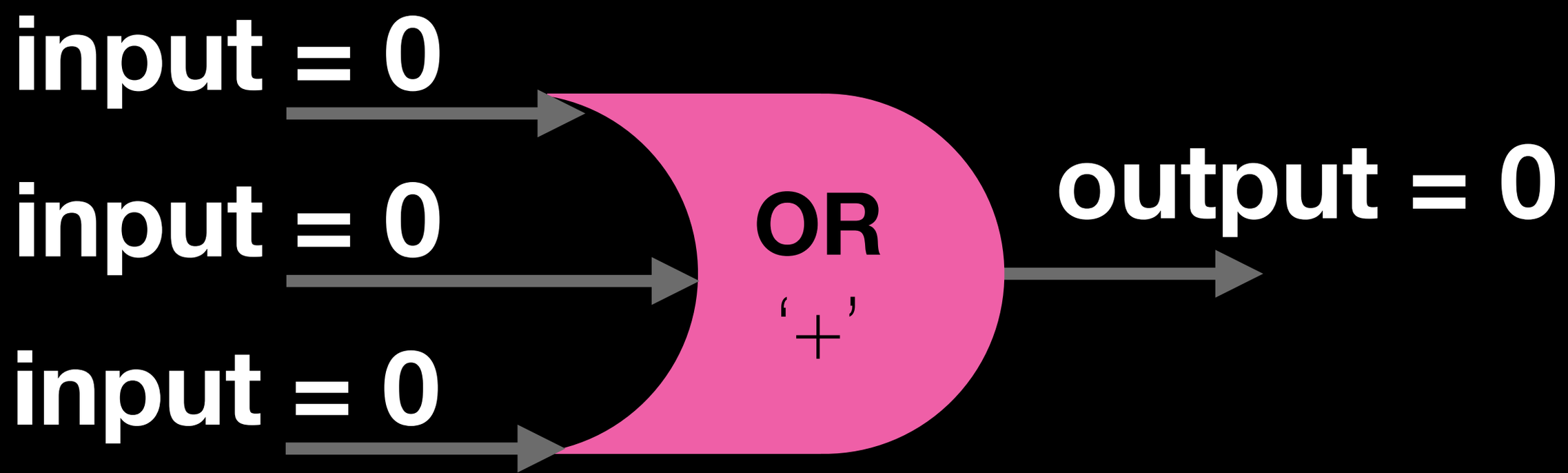
output = 1 ONLY iF ALL input = 1;
ELSE output = 0


input = 0
input = 0
OR
'+'
output = 0


input = 0
input = 1
OR
'+'
output = 1


input = 1
input = 0
OR
'+'
output = 1


input = 1
input = 1
OR
'+'
output = 1



output = 0 ONLY iF ALL input = 0;
ELSE output = 1

IMPORTANT NOTES

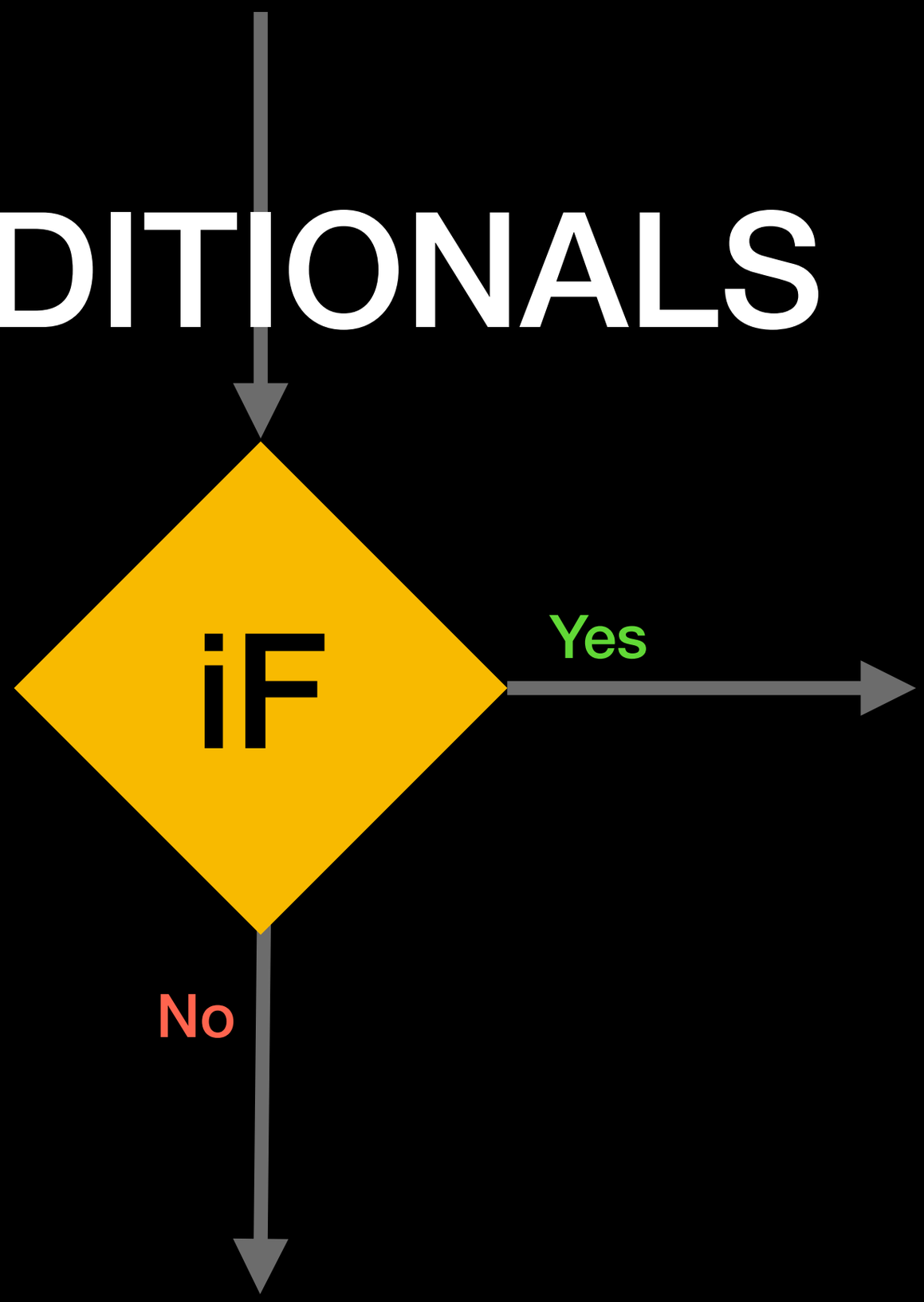
WRONG!! (VAR_A === X && !== Y) WRONG!!

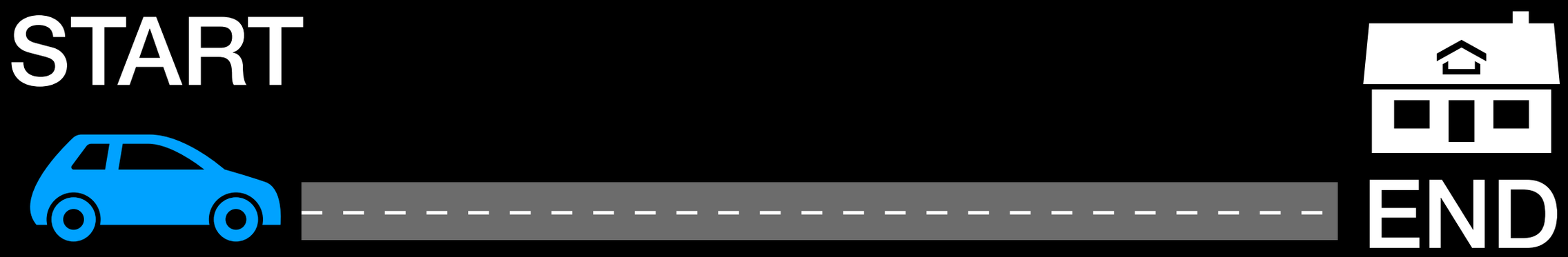
(VAR_A === X && VAR_A !== Y)

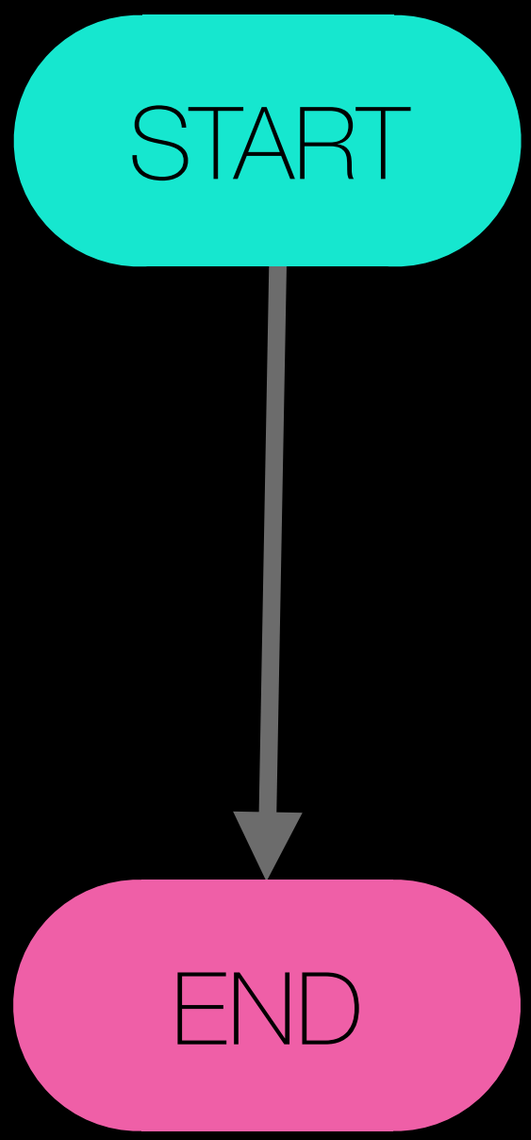
(VAR_B === Y || VAR_B !== X)

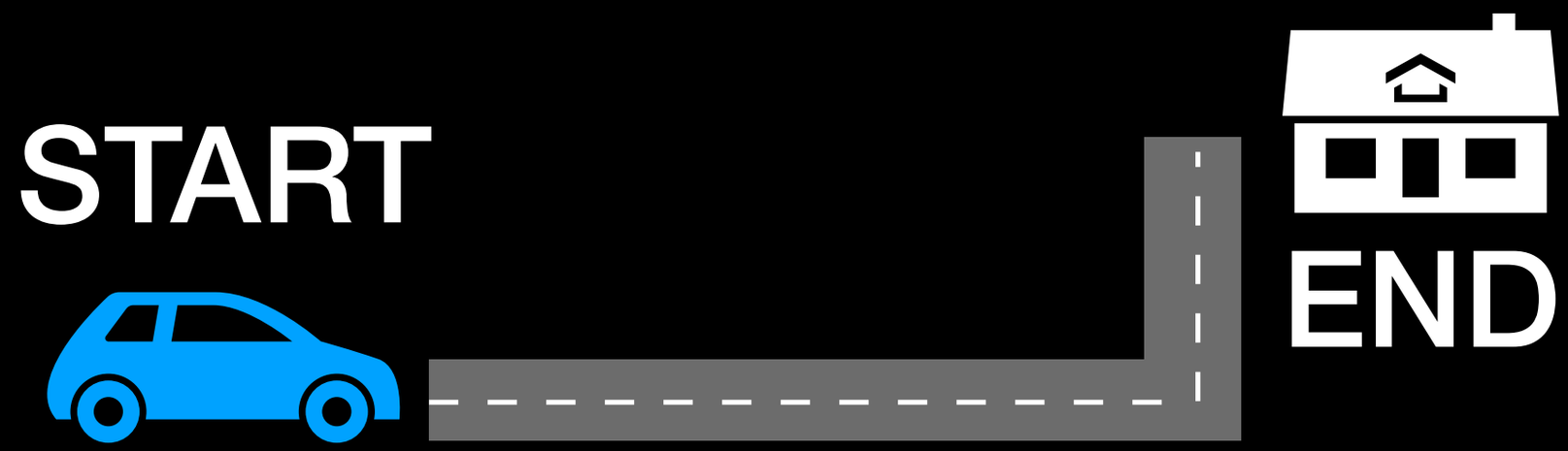
‘===‘ and ‘==‘ are logic comparator!
‘=‘ is used to assign value!

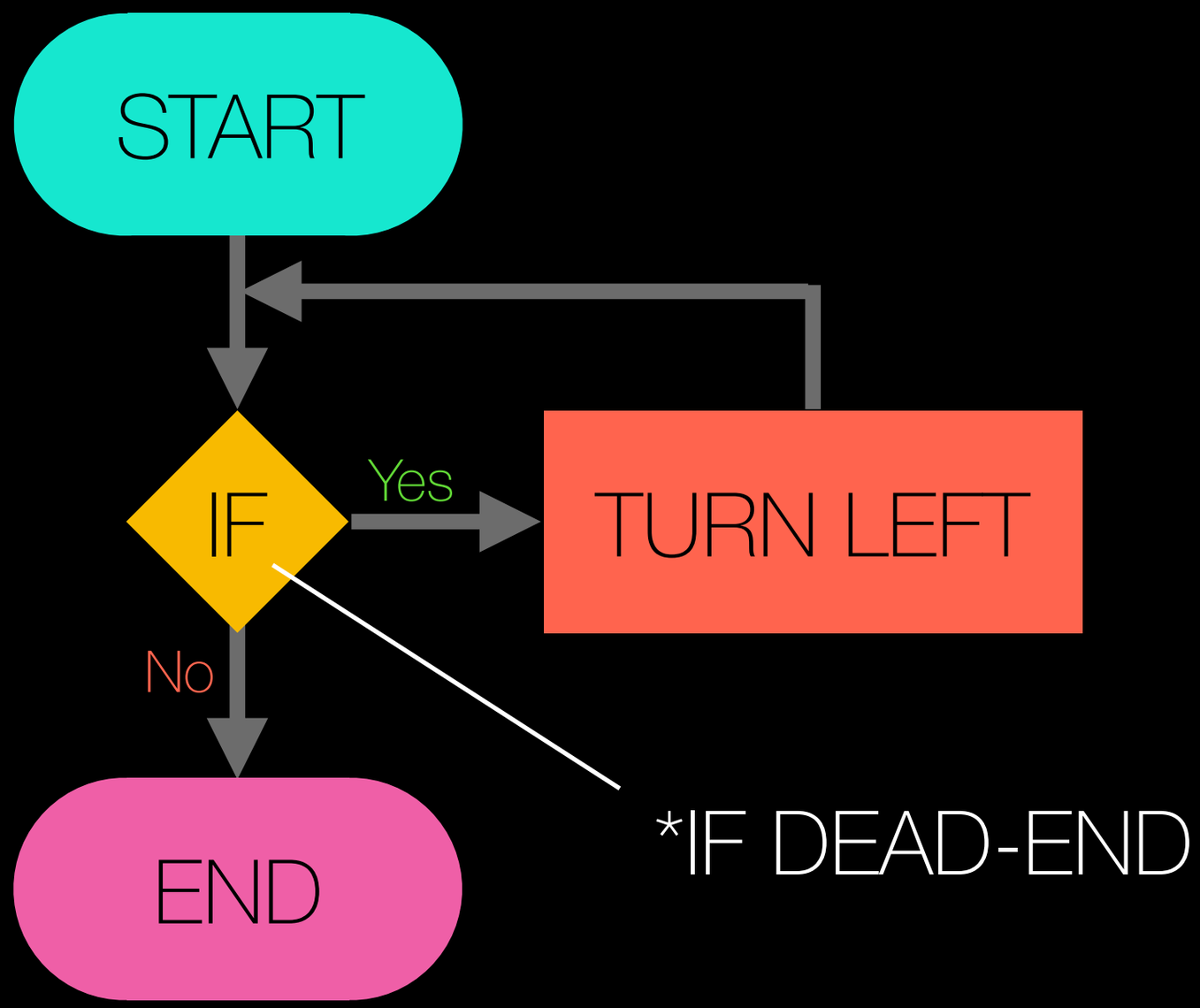
CONDITIONALS

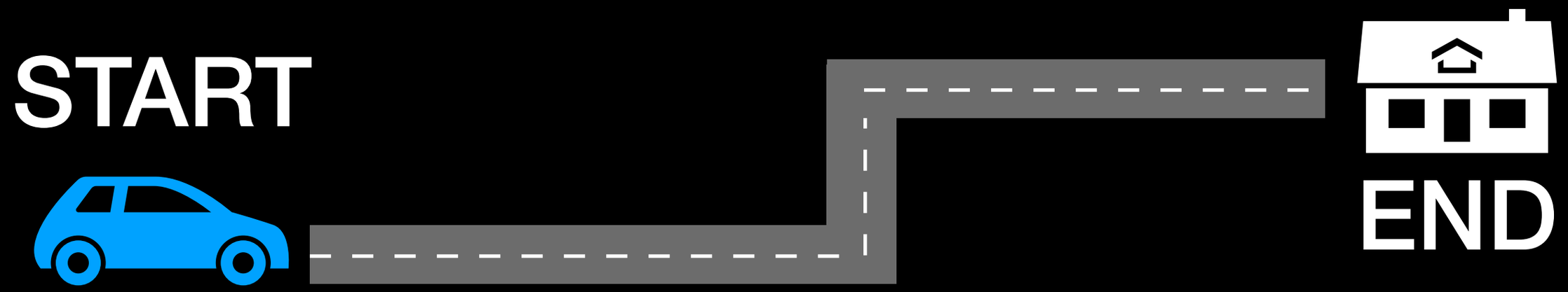


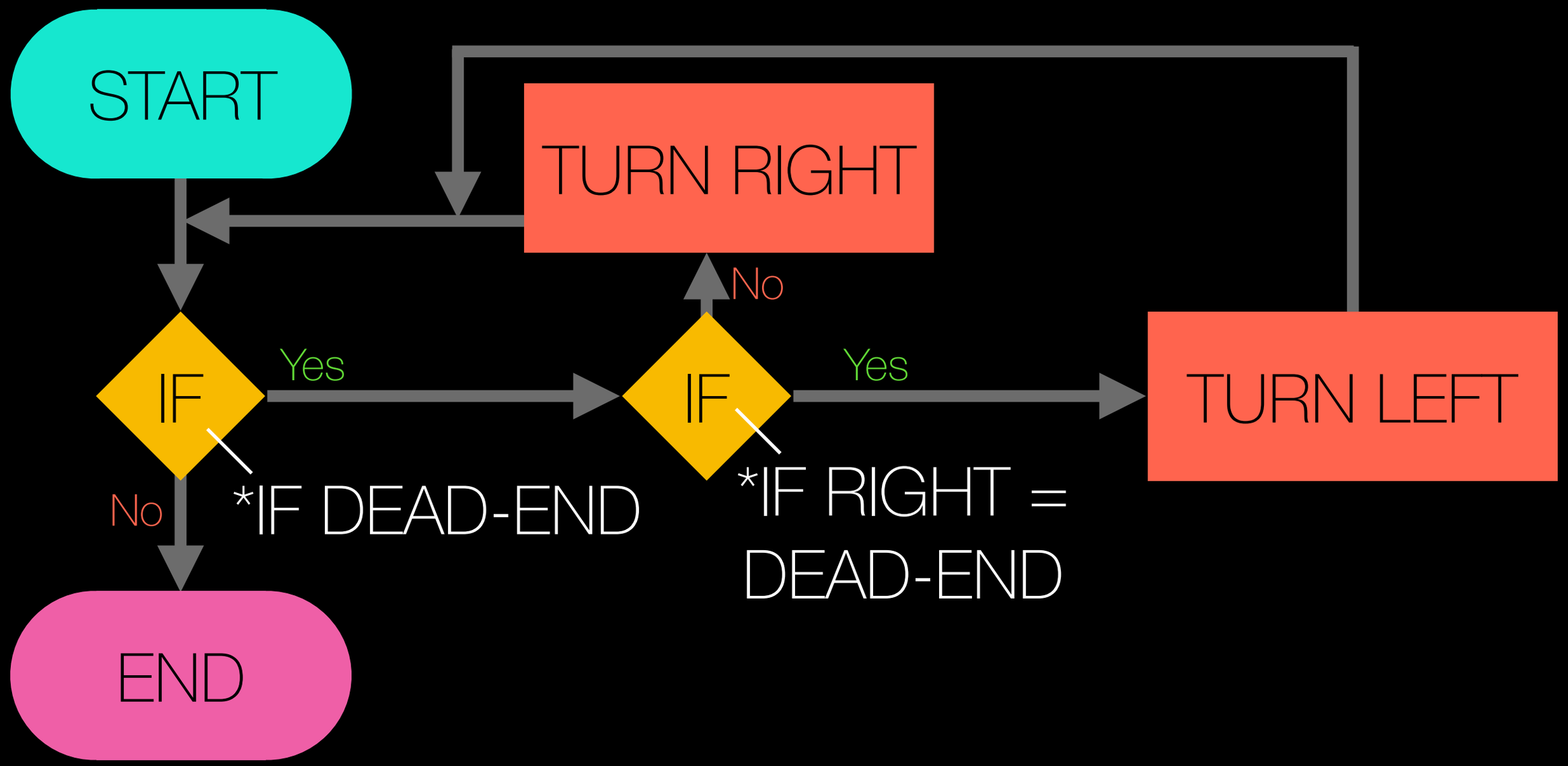

START
END

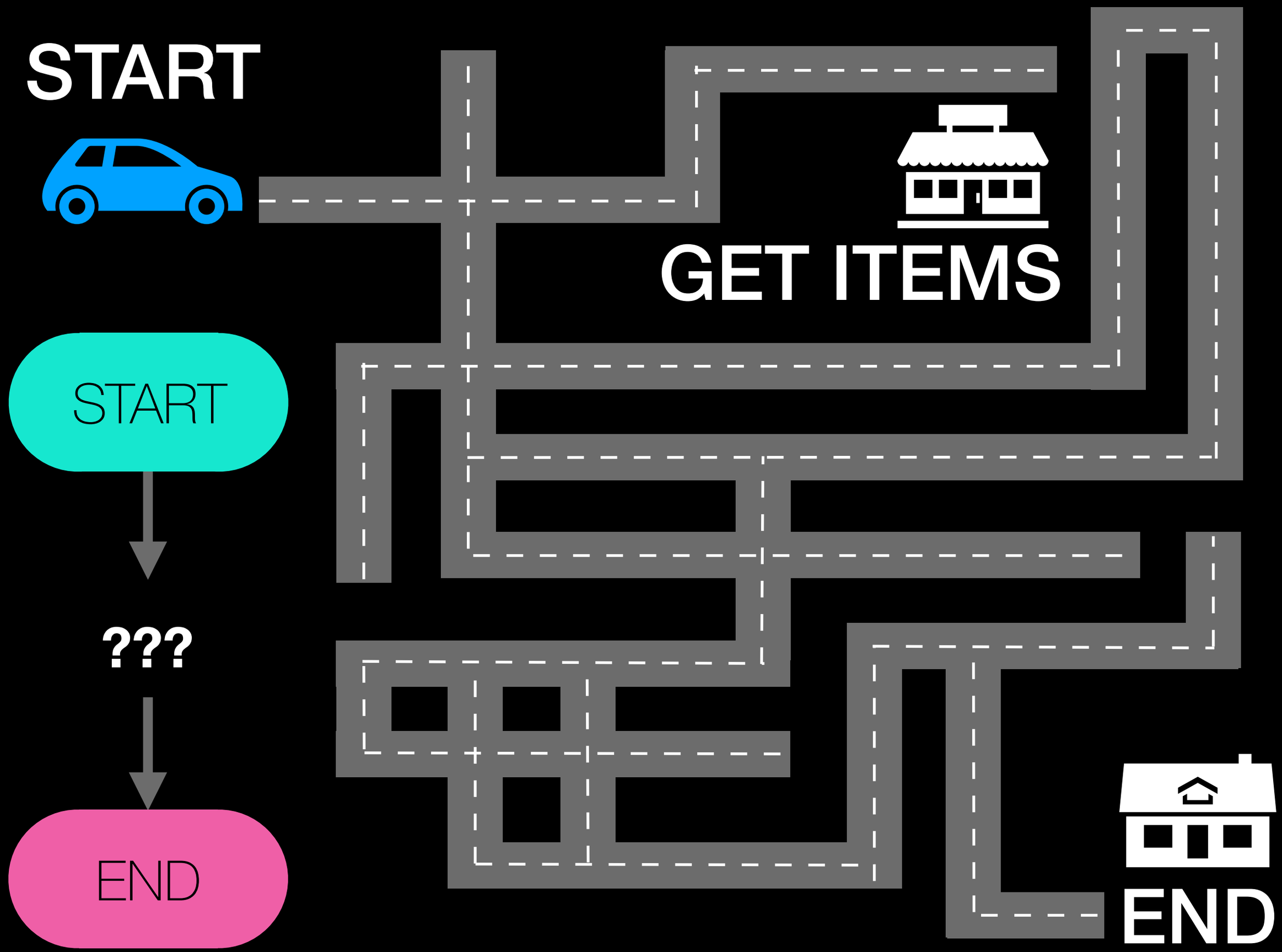

START
END


START
END

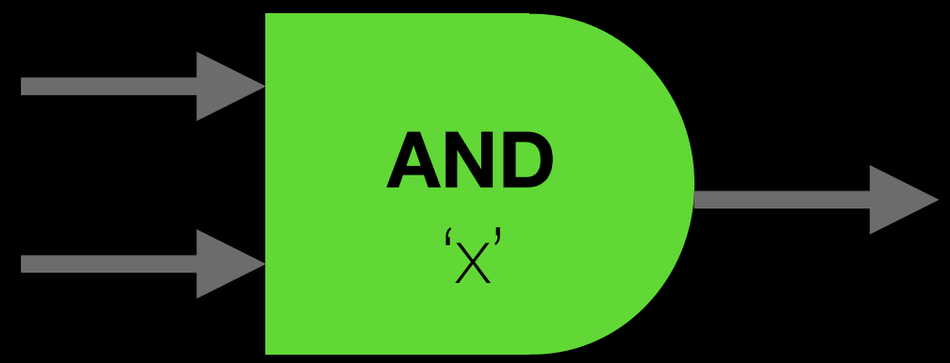

START
IF
Yes
TURN LEFT
No
END
*IF DEAD-END


START
END


START
TURN RIGHT
No
IF
Yes
IF
Yes
TURN LEFT
No
*IF DEAD-END
*IF RIGHT =
DEAD-END
END

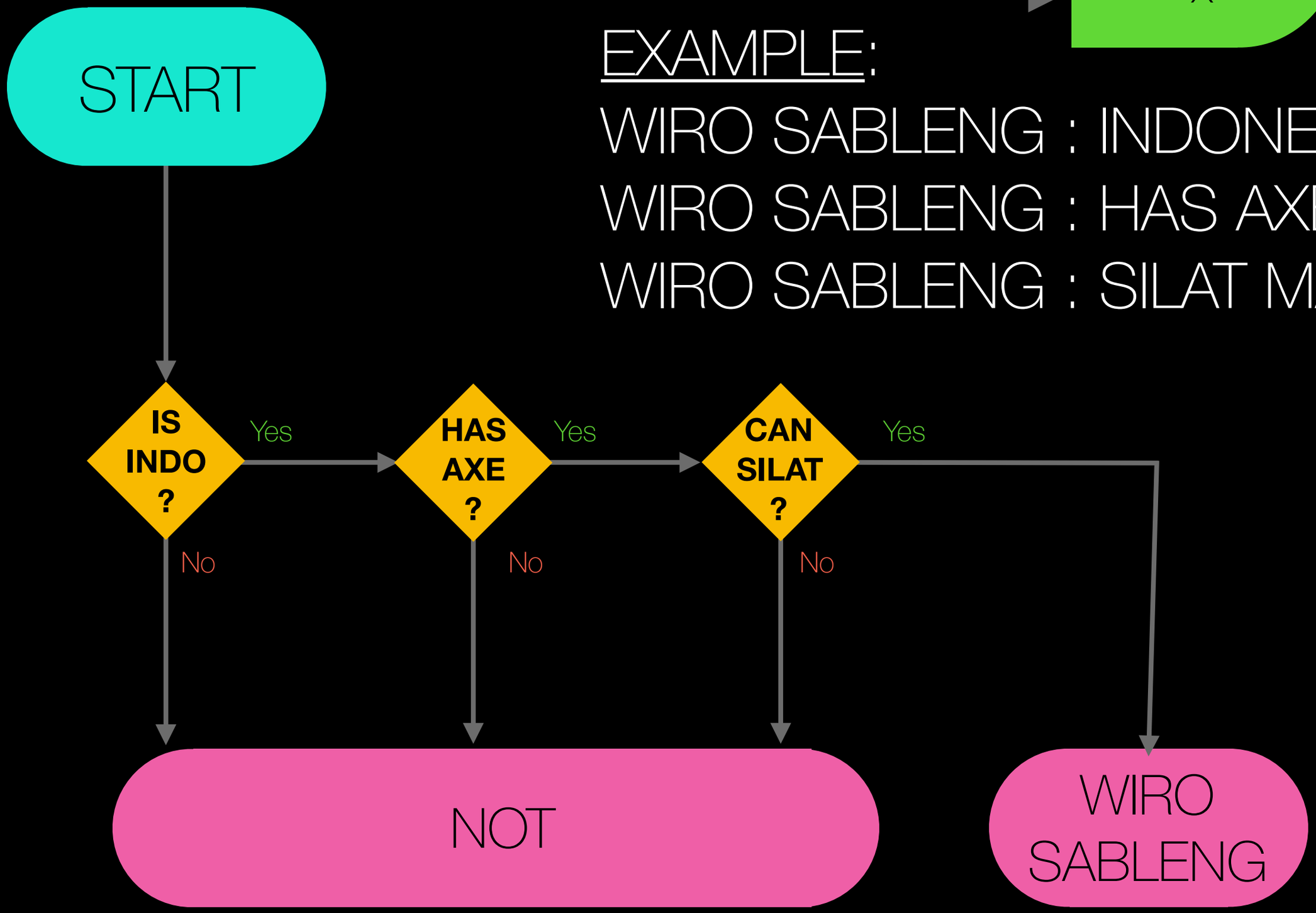

START
GET ITEMS
START
???
END
END

ONE LEVEL iF:

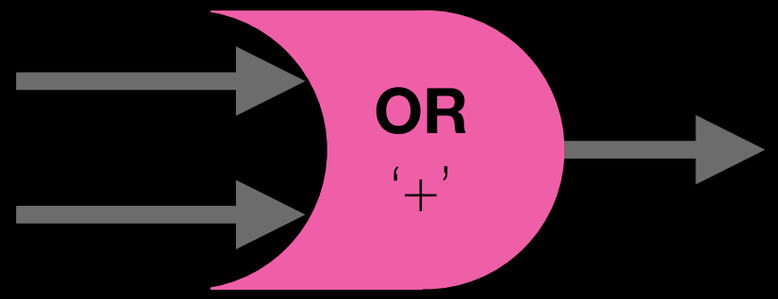


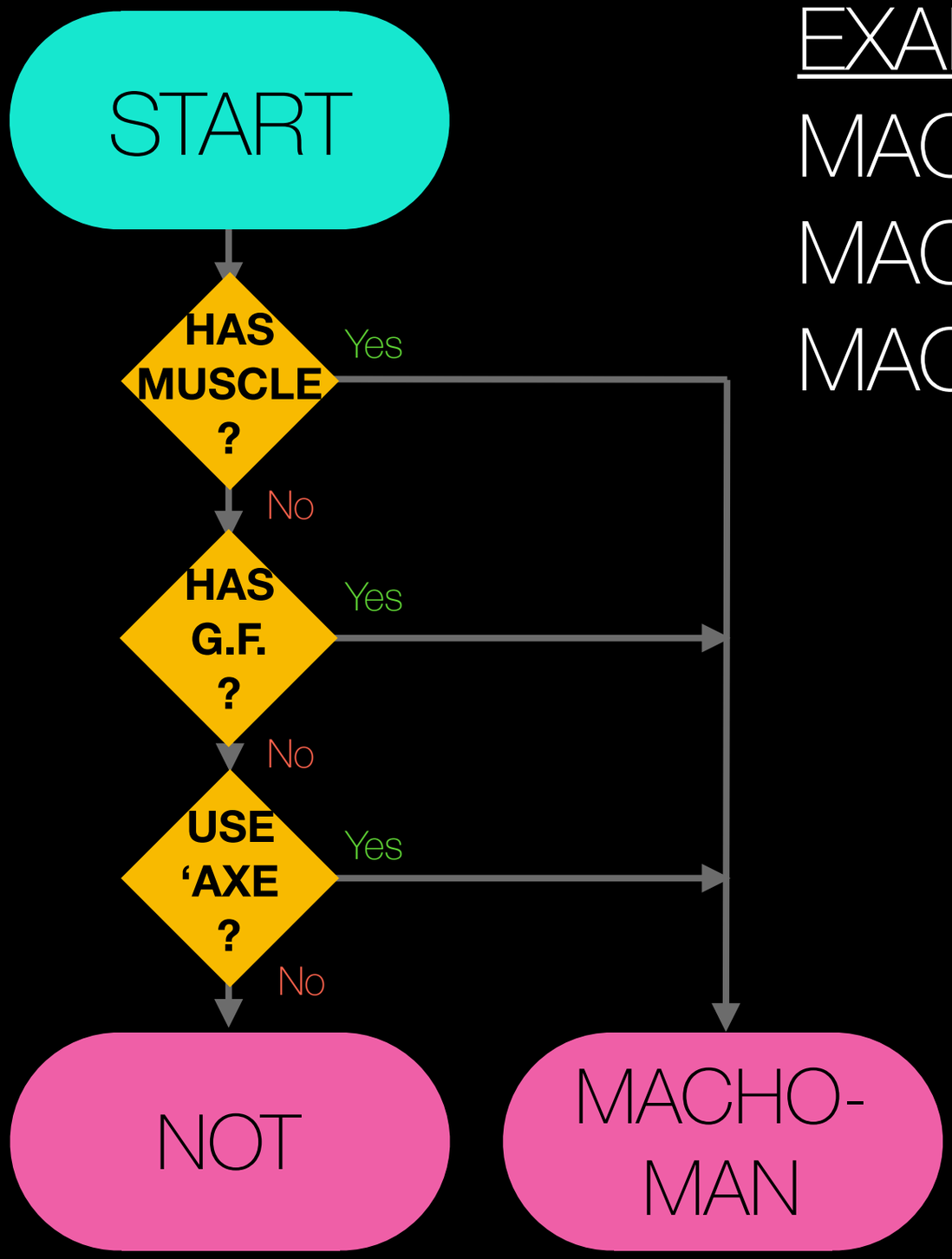
<u>EXAMPLE</u>:

WIRO SABLENG : INDONESIAN &
WIRO SABLENG : HAS AXE &
WIRO SABLENG : SILAT MASTER



ONE LEVEL iF:





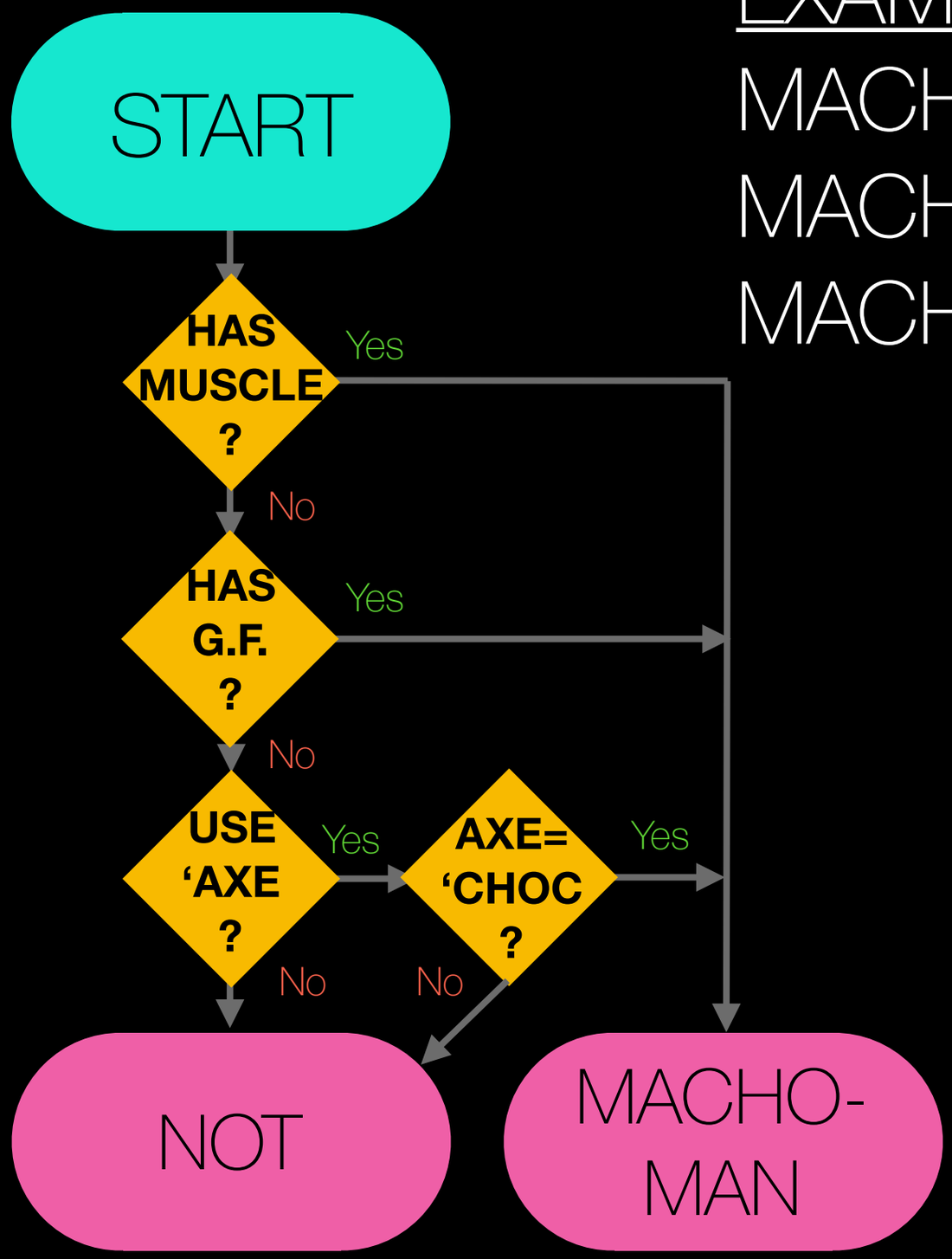
<u>EXAMPLE</u>:

MACHO-MAN : HAS MUSCLES or
MACHO-MAN : HAS GIRLFRIEND or
MACHO-MAN : USES 'AXE'

THIS IS A CONDITION WHERE YOU CAN USE:

SWITCH - CASE

MULTIPLE LEVEL OF iFs:



<u>EXAMPLE</u>:

MACHO-MAN : HAS MUSCLES or
MACHO-MAN : HAS GIRLFRIEND or
MACHO-MAN : USES ('AXE' : CHOC)

AND

IMPORTANT NOTES

IF ($*%#@#$&^@*%#&#&$^@) { }

this DOESN’T return an ‘ERROR’ message!!!

During LIVE CODE or STRESSED,
DON’T PANIC!
Always check inside your iFs first!

e.g. ’===‘ vs ‘=‘; correct variables

LOOPING

IMPORTANT NOTES

When do we use **WHILE** ?

INFINITE LOOP, with <u>at least one</u> CONDITIONAL where a 'TOGGLE' (true/false) variable **<u>changes value</u>** (true —> false, or false —> true);

ELSE …

Always use **FOR !!!**

IMPORTANT NOTES

```
for ( var i = 0; i < x.length; i += 2 ) { }
```

start — until end — increment (freq.)

Direction =

IMPORTANT NOTES

```
for ( var i = x.length - 1; i >= 0; i -- ) { }
```

start

until end

decrement (freq.)

Direction =

IMPORTANT NOTES

The Usual Mistakes: with Direction =

1. for (var i = 0 ; i < arr.**lenght** ; i ++) { } —> TYPO 'Length'
2. for (var i = 0 **,** i < arr.length **,** i ++) { } —> using comma instead of ';'
3. for (var i = 0 ; i **<= arr.length** ; i ++) { } —> array length = right-most index + 1
4. for (var i = 0 ; i < arr.length ; i **--**) { } —> right-to-left but increment (infinite loop!)
5. for (var i = 0 ; **j** < arr.length ; **k** ++) { } —> different variable(s)
6. for (var i = 0 ; i **>** arr.length; i ++) { } —> 'until-end' contradicts 'start' (loop exit!)
7. for (var i = 0 ; i < arr.length ; i ++) {
   for (**var i** = 0 ; **i** < arr.length ; **i** ++) { } } —> using same variable inside the SAME 'SCOPE'
8. for (var i = 0 ; i < arr.length ; i ++) { **...** —> forgetting to close with '}'

IMPORTANT NOTES

The Usual Mistakes: with Direction =

1. for (var i = arr.**lenght** - 1; i >= 0 ; i --) { } –> TYPO 'Length'
2. for (var i = arr.length - 1**,** i >= 0 **,** i --) { } –> using comma instead of ';'
3. for (var i = **arr.length** ; i >= 0 ; i --) { } –> array length = right-most index + 1
4. for (var i = arr.length - 1; i >= 0 ; i **++**) { } –> right-to-left but increment (infinite loop!)
5. for (var i = arr.length - 1; **j** >= 0 ; **k** --) { } –> different variable(s)
6. for (var i = arr.length - 1; **i < 0** ; i --) { } –> 'until-end' contradicts 'start' (loop exit!)
7. for (var i = arr.lenght - 1; i >= 0 ; i --) {
   for (**var i** = arr.lenght - 1; **i** >= 0 ; **i** --) { } } –> using same variable inside the SAME 'SCOPE'
8. for (var i = arr.lenght - 1; i >= 0 ; i --) { **...** –> forgetting to close with '}'

IMPORTANT NOTES

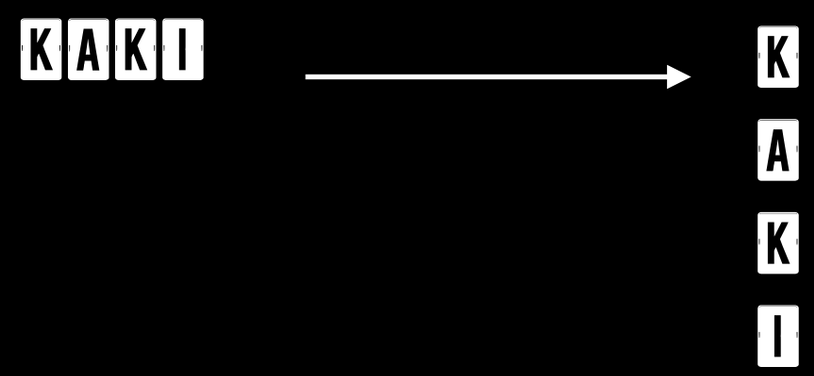
WHEN to ADD a LOOP within LOOP?

1. When we need different loop directions simultaneously.
   (LOOP#1 left-to-right, while LOOP#2 right-to-left)

2. When we want to access '*multi-dimensional*'/ additional-layer-details.
   (accessing array *['a', ['b1' , 'b2'] , 'c']*)

3. When within the loop, we need to save/push our 'answer' to an answer-array.
   (e.g. *Answer_Array.push(i)*)

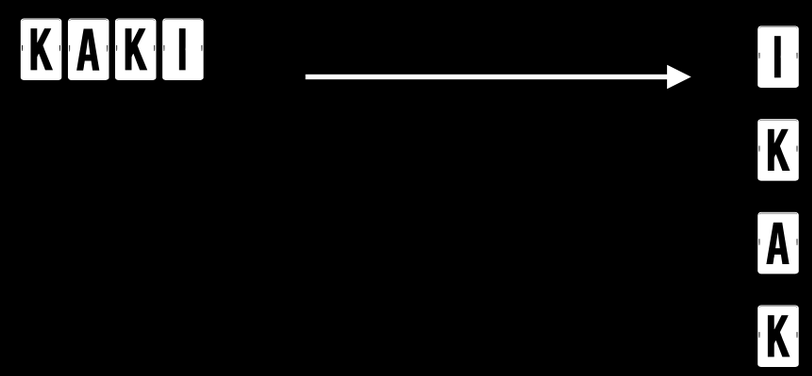
FOR Loop in STRING




```
var kata = ‘kaki’;
for (var j = 0; j  < kata.length ; j ++) {
   console.log(kata[j]);
}
```

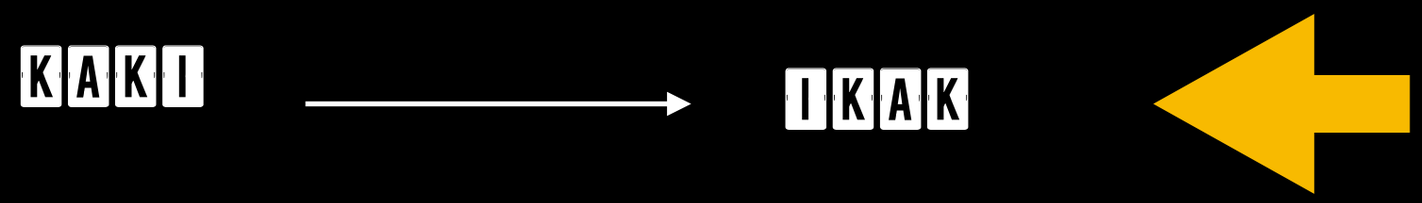
NOTE:
CONSOLE.LOG( ); = **DISPLAY** things inside ‘( )’
then **CHANGE ROW!!**

FOR Loop in STRING




```
var kata = ‘kaki’;
for (var ii = kata.length - 1 ; ii  >= 0 ; ii -- ) {
   console.log(kata[ii]);
}
```

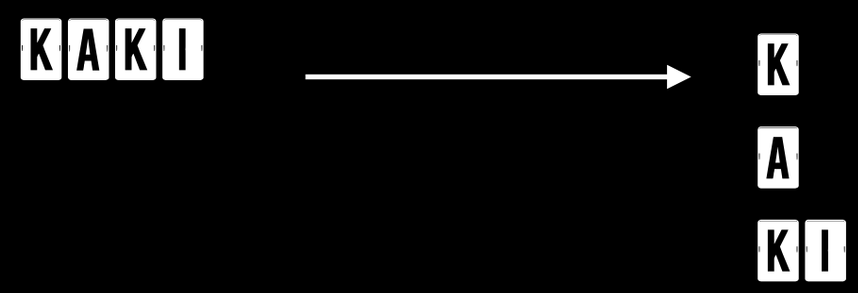
FOR Loop in STRING




```
var kata = ‘kaki’;
var sementara = ‘’;
for ( var k = kata.length-1 ; k >= 0 ; k -- ) {
   sementara += kata[k];
}
console.log(sementara);
```

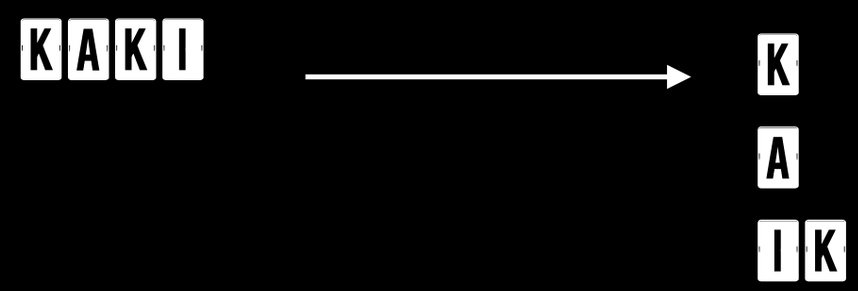
NOTE:
sem += kata[k] is the same with *sem = sem + kata[k]*;
This is a trick to compose letters horizontally
(because console.log changes the row)!!

FOR Loop in STRING



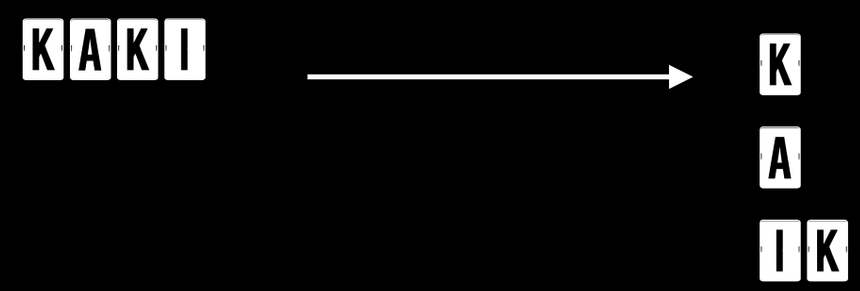

```
var kata = ‘kaki’;
var x = 0;
var sementara = ‘’;
for (x ; x  < 2 ; x ++) {
   console.log(kata[x]);
}   // –> here variable x = 2
for (x ; x < kata.length; x ++) {
   sementara = sementara + kata[x];
}
console.log(sementara);
```

FOR Loop in STRING




```
var kata = ‘kaki’;
var x = 0;
var sementara = ‘’;
var sementara2 = ‘’;
for (x ; x  < 2 ; x ++) {
   console.log(kata[x]);
} // —> here variable x = 2
for (x ; x < kata.length; x ++) {
   sementara = sementara + kata[x];
}
for (var y = sementara.length - 1; y >=0 ; y -- ){
   sementara2 += sementara[y];
}
console.log(sementara2);
```

FOR Loop in STRING

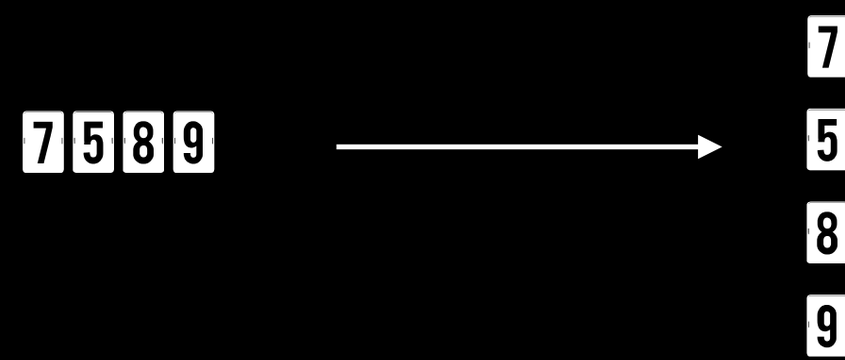



```
var kata = ‘kaki’;
var x = 0;
var sementara = ‘’;
for (x ; x  < 2 ; x ++) {
   console.log(kata[x]);
}  // —> here variable x = 2
for (var y = kata.length -1 ; y >= x ; y -- ){
   sementara += kata[y];
}
console.log(sementara);
```

FOR Loop in **NUMBERS**

7589 → 7 5 8 9

FOR Loop in **NUMBERS**



Numbers **CANNOT** be changed to **STRING** and then put together with STRING += WORD[i];

WHY?

Because in javascript: Numbers in a String Variable **can still be added as if it is a Number**!!

THEN HOW?

Turn the variable **to String** *{ num = n.toString() or String(n) }*, then **Split** *{ string.split(‘’) }*it to become an **ARRAY**; then process it.

FOR Loop in **NUMBERS**

7589 → '7589' → [7,5,8,9] → 7 5 8 9

```
var num = 7589;  //—> input
var arr = String(num).split("");  // —> number to string, string split to array
for (var i = 0 ; i <arr.length ; i++) {
   console.log(arr[i]);
}
```

FOR Loop in **NUMBERS**

Get smallest single digit number

7589 → 5

FOR Loop in **NUMBERS**

Get smallest single digit number

7589 ⟶ 5

```
var num = 7589;  //—> input
var arr = String(num).split("");  —> //  number to string, string split to array
var pembanding = 9; //—>  the biggest single digit number
for (var i = 0 ; i < arr.length ; i++) {
   if (arr[i] < pembanding ) { //—>  assign comparator new value
                                      if current element is smaller than comparator
     pembanding = arr[i];
}
console.log(pembanding);
```

// note: remember! Although technically the elements of the array are string versions of numbers; it can still be compared to a number (… can be added/subtracted/multiplied/ divided)

FOR Loop in **ARRAYS**

[[100, 200, 500, 1000],
[100000, 50000, 20000, 10000, 5000, 2000, 1000]] → 100000

HOW TO TACKLE THIS???!!! :(

FOR Loop in **ARRAYS**

[[100, 200, 500, 1000],
[100000, 50000, 20000, 10000, 5000, 2000, 1000]] ⟶ 100000

HOW TO TACKLE THIS???!!! :(

1. Simplify the Array —> arr = [[coins],[paper]]
   Now we know how many loops we need to access the details
   The first Loop (using var i) is to loop [coins],[paper];
   The 2nd Loop (using var j) is to loop the details inside 1st Loop;
2. Observe what is *constant* & what is *dynamic*
   —> in this case the length of [coins] and [paper] are different;
3. Observe what is asked (output)
   —> in this case is get the biggest number in the array & display
   because of this we need to compare, hence needing:
   —> comparator variable (set to the opposite of biggest)
   —> conditional (if clause) to compare and assign the value
   to the comparator the item is bigger than comparator;

FOR Loop in **ARRAYS**

[[100, 200, 500, 1000],
[100000, 50000, 20000, 10000, 5000, 2000, 1000]] → 100000

var arr = [[100, 200, 500, 1000], [100000, 50000, 20000, 10000, 5000, 2000, 1000]]

```
var pembanding = 0;
for (var i = 0; i < arr.length; i ++) {
   for (var j = 0; j < arr[i].length; j++) {
      if ( arr[i][j] > pembanding ) {
        pembanding = arr[i][j];
      }
   }
}
console.log(pembanding);
```

FOR Loop in **ARRAYS**

```
var arr =  [ [ [ 100, 150, 200, 250 ],
               [ ari, ira, ria, air ],
               [ #*$, $#*, **$, $## ] ],
             [ [ 250, 300, 500, 700 ],
               [ ika, ria, asami, komang],
               [ $#*, *$$$, $#$, **#] ] ]
```

→ 700

HOW TO TACKLE THIS???!!! :(

FOR Loop in **ARRAYS**

```
var arr =  [ [ [ 100, 150, 200, 250 ],
               [ ari, ira, ria, air ],
               [ #*$, $#*, **$, $## ] ],
             [ [ 250, 300, 500, 700 ],
               [ ika, ria, asami, komang],
               [ $#*, *$$$, $#$, **#] ] ]
```

→ 700

HOW TO TACKLE THIS???!!! :(

1. Simplify the Array: arr =[[[num],[nam],[sym]] , [[num],[nam],[sym]]] ; arr = [[group1],[group2]] ; —> 3 loops;
2. Observe what is *constant* & what is *dynamic*:
   [group1] & [group2] are arranged in the same way & same length (4 elements each inside [num], [nam], & [sym])
3. Observe what is asked (output): getting the largest number… only in the [num] —> reduce 1 loop;

FOR Loop in **ARRAYS**

```
var arr =  [ [ [ 100, 150, 200, 250 ],
               [ ari, ira, ria, air ],
               [ #*$, $#*, **$, $## ] ],
             [ [ 250, 300, 500, 700 ],
               [ ika, ria, asami, komang],
               [ $#*, *$$$, $#$, **#] ] ]
```

→ 700

```
var pembanding = 0;
for (var i = 0; i < arr.length; i ++) {
   for (var j = 0; j < arr[i][0].length; j++) {
      if ( arr[i][0][j] > pembanding ) {
        pembanding = arr[i][0][j];
      }
   }
}
console.log(pembanding);
```

—> // arr[i][0] to access inside [num] which always happens to be the first element / row of the [group] array…

QUESTIONS