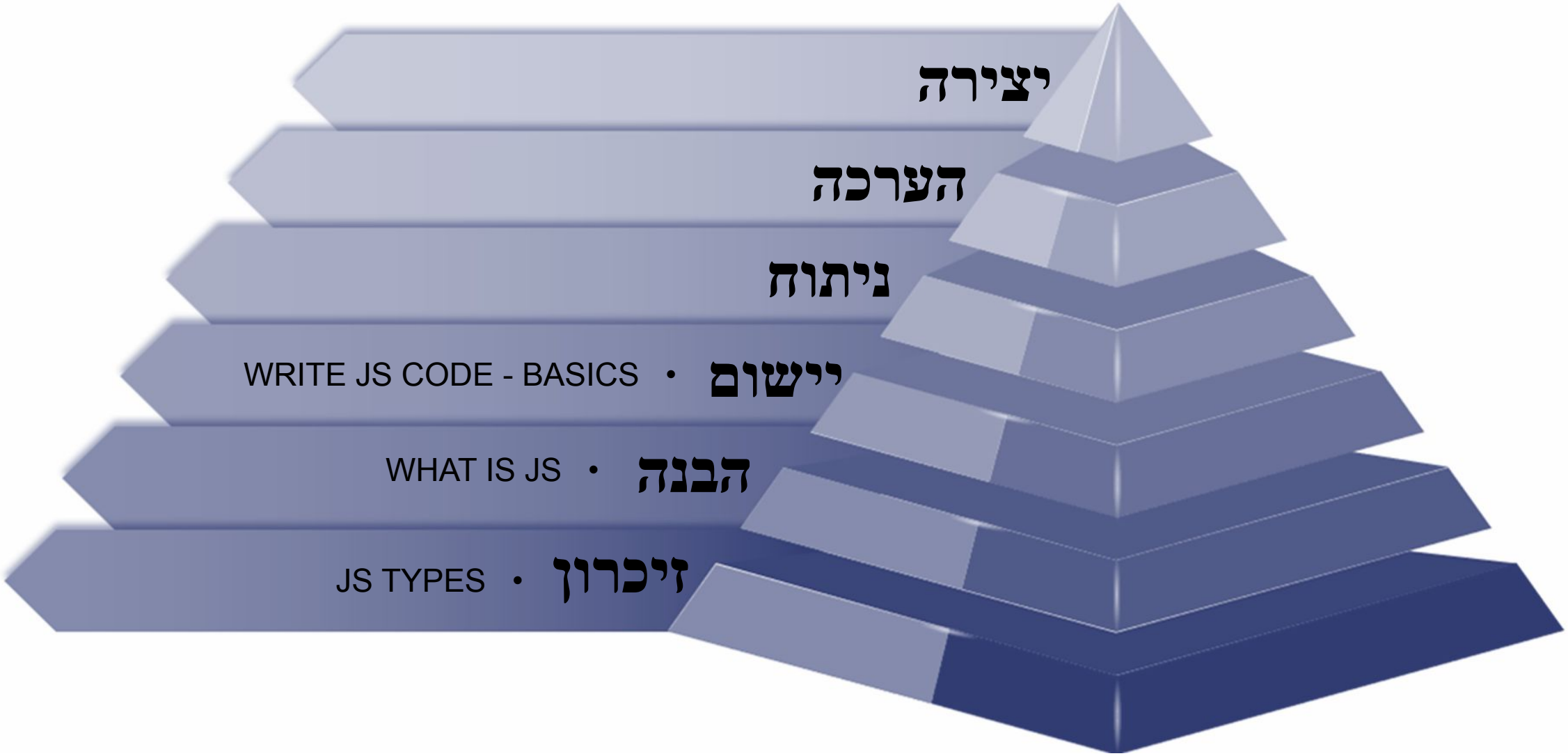
KOKCODE

JS BASICS
conditions, loops

תוכן העניינים

- Goal: understand js conditions and loops
- Topics:
  - operators
  - conditions
  - loops

מטרות השיעור



KOKCODE

js operators

js operators

- We saw some operators in the previous lecture.

- Like “+” - which when put between number produces a number
- We talked about mathematical operators.
- Today we will talk about logical ones.

js operators

- What does “logical operators”?

- It mainly means that **an operator that will ALWAYS result in a boolean.**

js operators

- We have mathematical comparisons:
    - Bigger than: <
    - Smaller than: >
    - Is equal to: = = or = = = (not assignment!)
    - Not equal: !== or ! =
    - Bigger or equal: < =
    - etc

js operators

- For example:

```
let age = 35;

console.log(age < 55) // true
```

- Remember: the result of a logical operator will be a boolean - true or false.

js operators

- What if we want to do multiple operators together?

- For example: if I have age and height, I want to check if the age is bigger than 18 and height is larger than 110 in order to see if you can go on a rollercoaster.

js operators

- For that we need 2 more operators: **and** and **or**

- In js its written like so:
  - And: &&
  - Or: ||
- We have another operator ! (NOT) - that turns true to false and false to true
  Like in: !(55<50) // true

js operators

- **And**: will return true ONLY if the 2 logical operations are true.

- I.e.: (operation 1) && (operation 2)
  Will be true only if 1 and 2 are true.
  Any other case will be false.

- In js:
  35 >= 25 && 110 = = = 110 // true
  55 <=10 && 22 % 3 = = = 0 // false

js operators

- **OR**: will return true if AT LEAST 1 of the 2 logical operations are true.

- I.e.: (operation 1) && (operation 2)
  Will be true if 1 or 2 are true.
  Any other case will be false.

- In js:
  35 >= 25 || 110 = = = 110 // true
  12 % 5 = = = 0 || 15 > 40 // false

js operators

-

AND truth table

condition 1	condition 2	result
T	T	T
T	F	F
F	T	F
F	F	F

js operators

-

OR truth table

condition 1	condition 2	result
T	T	T
T	F	T
F	T	T
F	F	F

js operators

- Recap:

  What will be the result of:

  - 8 >= 8
  - "hello" ! = = ‘Hello’
  - 10 > 2 || 4 = = = 2;
  - (7 ! = = 7) || (8 <= 10 && 3 > 1);

js operators

- I showed that in order to check if something is equal we have 2 ways in JS
  - Double equal: = =
  - Triple equal: = = =
- What is the diff?

js operators

- What happens if we use logical operators between 2 diff

  data types?

- For example:
  35 > “yishai”
  ||
  33 = = false || 55 == true
- What will happen?

js operators

- Similar to mathematical operators - JS will try to do **type conversion**,

  or **type coercion**.

- In math we saw it:
  5 + “yishai” // 5yishai - 5 was converted to a String
- 45 * “yishai” // NaN - yishai was converted to a number, and it’s not a number

js operators

- In mathematical comparisons JS will try to convert any none number into a number.
- If it can be - like in “5” or in true/false (1/0) - it will.
- If it cannot - it will be NaN, and any comparison will always evaluate to be **false**.

js operators

- Mathematical operators type conversions:
  - 5 < undefined // false
  - 10 >= true // true
  - “Yishai” >= false // false

js operators

- The only logical operator that does not do type conversion is

strict equal - triple equal.

- This is why we will always use triple equal unless we have to use double equal.
- console.log(35 = =”35”) // true
  console.log(35 = = = “35”) // false

js operators

- In logical operators it’s diff.

- If we use && || ! - JS will try to convert the values into booleans.
- And value has either a **truthy** or a **falsy** value. I.e - will it be turned into true or false when converting to a boolean.

js operators

- Falsy values are:
  - Empty string: “”
  - 0
  - false
  - null
  - undefined
- Any other value is truthy

js operators

- For example:

```
0 || 5<10 // true

“ ” && (undefined > “yishai”) // false
```

js operators

- Exe - what will be the result of:

  - 5 < "6" && ("10" > 9 || false = = 0)
  - !(0 || "0") && "true" = = true || 1
  - "15" >= 10 || (0 = = "" && !false)
  - null == 0 || !undefined && "5" <= 5

Analiza

Intelligence - Cyber - Data

KOKCODE

IF

js if

- We can use logical operators for assigning values to variables or printing, but we can use them for something bigger:
- Flow control!

js if

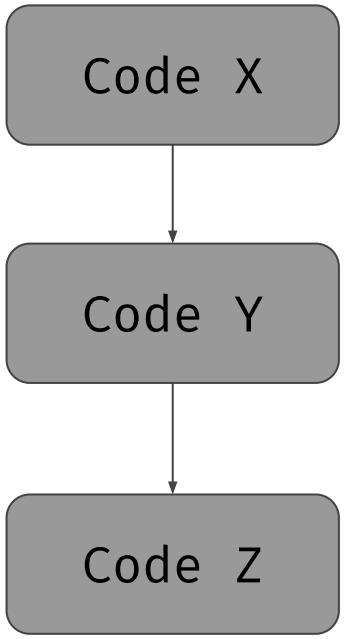
- Until now our JS executed every line of code we wrote.
- This is not that we want, many times.
- Sometimes we want to run our code **conditionally**. I.e - if a condition is met, run code X. if not - run code Y

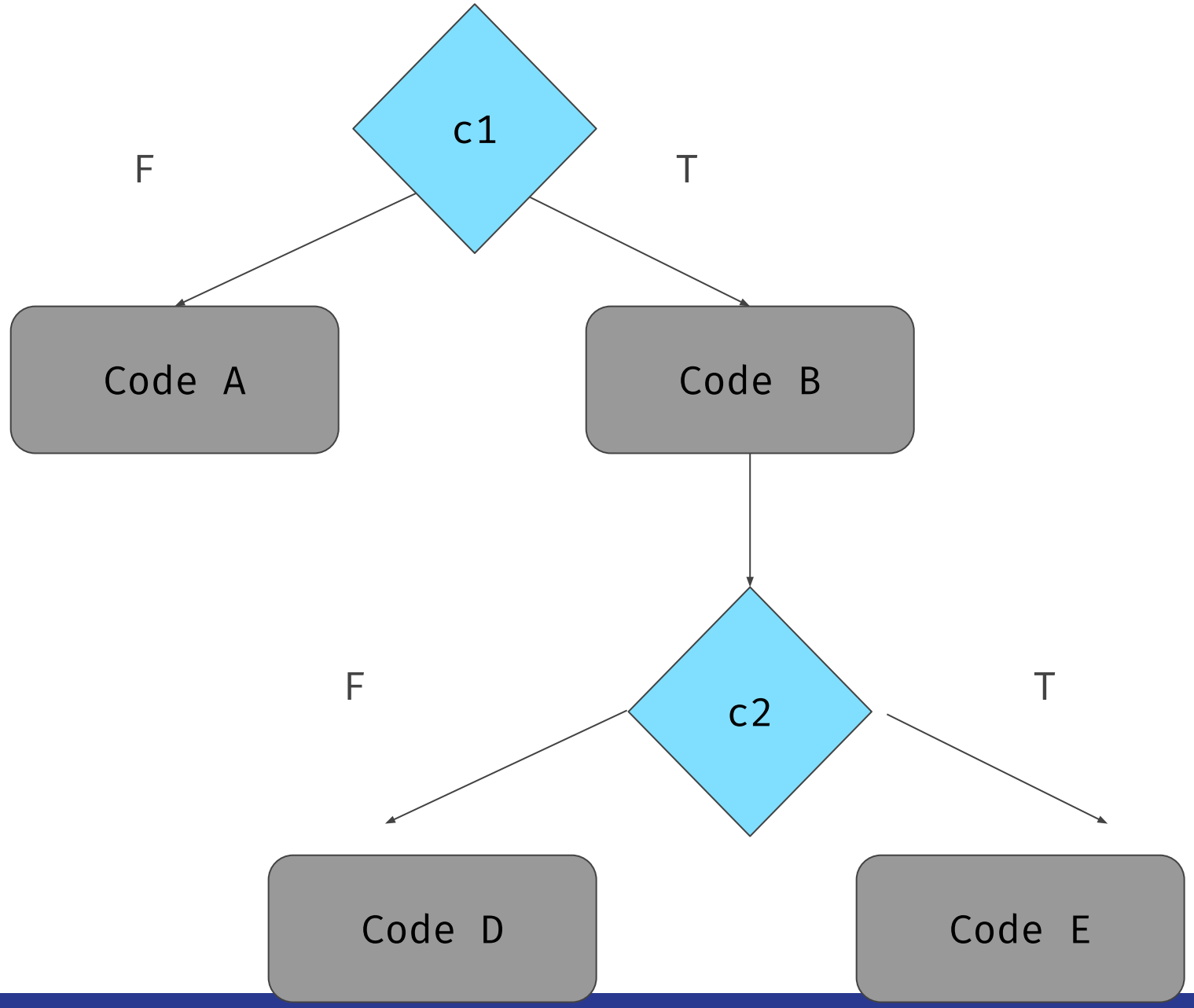
js if

- We can think about it like a flow chart:

js if

Regular code





js if

- The basic way to write a condition is with an “if statement”.
- Its syntax is like so:

```
if(condition 1){
    // code that runs if condition = true
} else if(condition 2){
    // code if condition 1 is false and condition 2 is true
} else {
    // code if con 1 & 2 are false
}
```

js if

- For example:

```
const age = 19;

if(age>18){
    console.log(‘you can vote’);
}
```

This will print, because age>18 = true.

js if

- On the other hand:

```
const age = 17;

if(age>18){
    console.log('you can vote');
}
```

This will not print, because age>18 = false

js if

- Anything inside a condition will results in a boolean.
- If its a mathematical comparison - its will type corese and resolve.
- If its a logical operator - it will convert to truthy or falsy value.

js if

- For example:
  const age = 0;
  if(age){
  console.log(‘you are bigger than 0!’);
  }

  This will not print, because age is 0, and 0 is **falsy**, i.e - it turns into the boolean false by the JS engine.

js if

- On the other hand:

  const name = ‘yishai’
  if(name){console.log(‘you have a name!’)};

  This will print, because an none empty string is **truthy**, and it will be evaluated as true.

js if

- If we have a simple else if, sometimes the full syntax can be annoying, like in:

  ```
  if(age>10){console.log(“small”)}
  else {console.log(“big”)}
  ```

- We can write it in a shorter way, using the **ternary** operator.

js if

- the ternary operator (the combination of ? and :)
- The structure is like so:

  Condition ? code if true : code if false

js if

- For example:
  const age = 17;
  let msg;

  msg = age > 18 ? ‘you can vote’ : ‘no vote’;
  console.log(msg);

  Will print ‘no vote’, because age > 18 is false, so the part after : is selected.

js if

- The structure is:

```
switch(variable){
    case value1:
        // code if variable = value1
        break;
    case value2:
        // code if variable = value2
        break;
}
```

js if

- Task:
  - Check age and height
  - If age < 15 - print cannot enter
  - If age > 15 and height < 110 - print go on kids ride
  - If age > 15 and he igt > 110 - print go on rollercoaster
  - If at any point sonOfManager = true, the kid can go on whatever he wants

KOKCODE

js loops

js loops

- While loops syntax:

```
while(CONDITION){
    // code here
}
```

- As long as the CONDITION is evaluated to be true - the code inside will run.

js loops

- Less used syntax is do… while

```
do {
// code here
} while(CONDITION)
```

js loops

- So using a while loop, how would you print all numbers from 0 to 100?
- Try it!
- Remember that this is not the main use case - while loops are mainly for loops without specific number of iterations.

js loops

- For loop - General syntax:

```
for(INITIALIZATION; CONDITION; AFTERTHOUGHT){
    // code
}
```

js loops

- A for loop gives use a variable. We **initialize it** at the first section of the for loop.
- This is the value the variable will have when the loops runs for the first time.
- The value will be init **before** the code runs.
- For example:

```
for(let i = 0; CONDITION; AFTERTHOUGHT){
    // code
}
```

js loops

- Then we add the condition.
- Remember - only if the condition is true the code will run.
- In order to control the amount of times the code will run - **we will usually use the variable inside the condition.**

js loops

- For example:

```
for(let i = 0; i < 10; AFTERTHOUGHT){
    // code
}
```

js loops

- Just like a while loop - we have to make sure that the condition will be false, eventually (“terminate the loop”).
- So we change our variable in the **afterthought - in a way the will make the condition false**.
- The afterthought will run when the code is done running.

js loops

- For example:

```
for(let i = 0; i <= 10; i++){
    console.log(i);
}
```

- How many times the code will run?

js loops

- After the afterthought the condition is evaluated again, and if it’s true - the code inside will run again.
- Then the afterthought will run again…
- This is the loop.

js loops

- Notice that you can init the variables however you like!

  I = 0, i = 10, name=”yishai” etc
- You can do the afterthought however you want!
  i - -, i + 50, i+” nachaliel”
- Just make sure the variable and afterthought can terminate the condition!

js loops

- How many times the code will run?
  - for (let i = 0; i < 5; i++){}
  - for (let x = 10; x > 5; x--) {}
  - for (let j = 11; j < 10; j += 2){}
  - for (let a = 3; a < 15; k+=2) {}
  - for (let c = 0; c >= 0; c+=5) {}

js loops

- Exe:
  - Use for loop to print all numbers that are divided fully by 3 between 0-50.