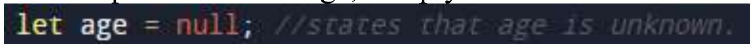
DATA TYPES & CONDITIONALS

Before we start digging deep, read through this overview of the most common data types in JavaScript: https://javascript.info/types

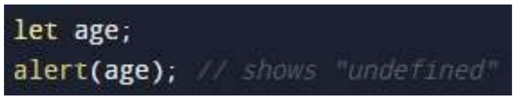
A. Data Types

- Null : represents "nothing", "empty" or "value unknown"



```
let age = null; //states that age is unknown.
```

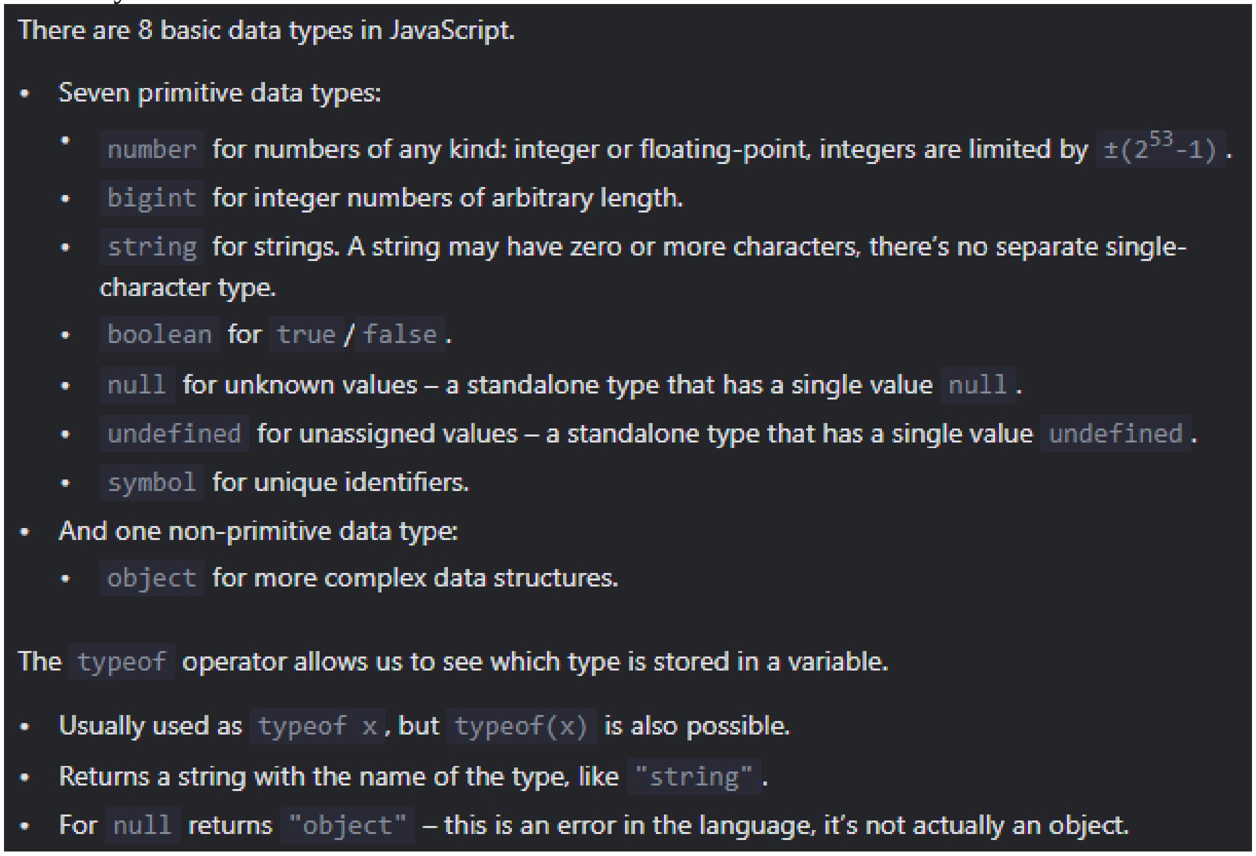
- Undefined : The meaning of undefined is "value is not assigned"



```
let age;
alert(age); // shows "undefined"
```

- Object : to store collections of data and more complex entities
- Symbol : to create unique identifiers for objects

Summary:



There are 8 basic data types in JavaScript.

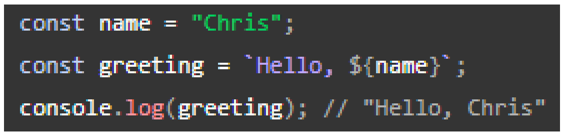
- Seven primitive data types:
  - `number` for numbers of any kind: integer or floating-point, integers are limited by `±(2^53-1)`.
  - `bigint` for integer numbers of arbitrary length.
  - `string` for strings. A string may have zero or more characters, there's no separate single-character type.
  - `boolean` for `true` / `false`.
  - `null` for unknown values – a standalone type that has a single value `null`.
  - `undefined` for unassigned values – a standalone type that has a single value `undefined`.
  - `symbol` for unique identifiers.
- And one non-primitive data type:
  - `object` for more complex data structures.

The `typeof` operator allows us to see which type is stored in a variable.

- Usually used as `typeof x`, but `typeof(x)` is also possible.
- Returns a string with the name of the type, like `"string"`.
- For `null` returns `"object"` – this is an error in the language, it's not actually an object.

B. String

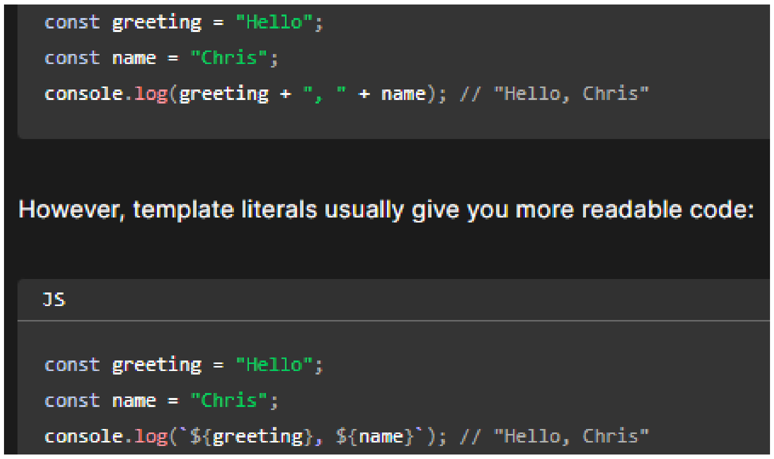
In JS, you can choose single quotes ('), double quotes ("), or backticks (`) to wrap your strings in. Strings declared using backticks are a special kind of string called a template literal. Inside a template literal, you can wrap JavaScript variables or expressions inside ' ${ } ', and the result will be included in the string:



```
const name = "Chris";
const greeting = `Hello, ${name}`;
console.log(greeting); // "Hello, Chris"
```

➢ Concatenation using "+"

You can use ${} only with template literals, not normal strings. You can concatenate normal strings using the + operator:



```
const greeting = "Hello";
const name = "Chris";
console.log(greeting + ", " + name); // "Hello, Chris"
```

However, template literals usually give you more readable code:

JS

```
const greeting = "Hello";
const name = "Chris";
console.log(`${greeting}, ${name}`); // "Hello, Chris"
```

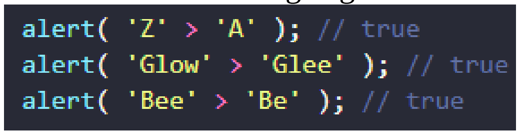
- ➢ Template literals
  - Including expressions in strings.
  - Allow to write Multiline strings(without \n).
  - Including quotes in strings. We also can use single quotes for wrapping strings that including (double) quotes. We also can use backslash to make sure the quote recognized as text. Ex: ‘I\'ve got no right to take my place…'

1. String Method
   Source:
   - https://www.w3schools.com/js/js_string_methods.asp
   - https://www.w3schools.com/jsref/jsref_obj_string.asp

   Docs: (You are not expected to memorize these but the documentation will be a very useful reference to revisit, so bookmark it!)
   - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

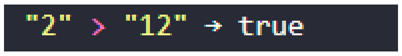
C. Conditionals

1. Comparison
   - ➢ String comparison
     To see whether a string is greater than another, JS uses unicode order.



```
alert( 'Z' > 'A' ); // true
alert( 'Glow' > 'Glee' ); // true
alert( 'Bee' > 'Be' ); // true
```

- the longer string is greater.
- “a” is greater than “A”, Because the lowercase character has a greater index in the unicode.

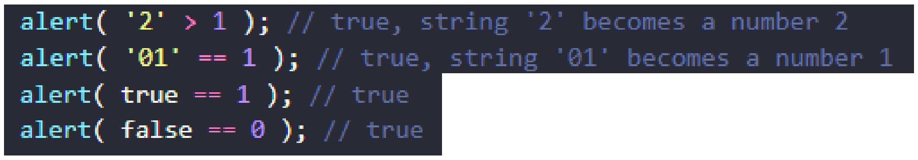


```
"2" > "12" → true
```

- unicode comparison, first char "2" is greater than the first char "1".

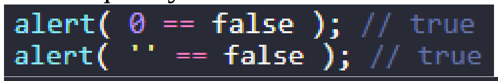
   - ➢ Different types comparison

When comparing values of different types, JavaScript converts the values to numbers.

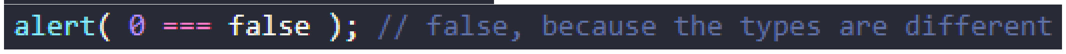


```
alert( '2' > 1 ); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
alert( true == 1 ); // true
alert( false == 0 ); // true
```

- ➢ Strict equality

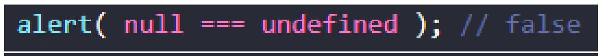


```
alert( 0 == false ); // true
alert( '' == false ); // true
```

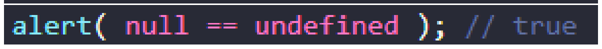


```
alert( 0 === false ); // false, because the types are different
```

- ➢ Null & Undefined comparison

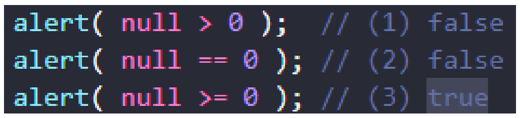


```
alert( null === undefined ); // false
```



```
alert( null == undefined ); // true
```

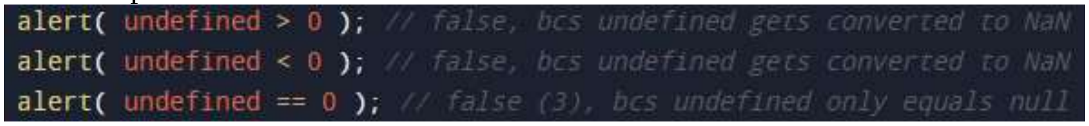
Strange result



```
alert( null > 0 );  // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

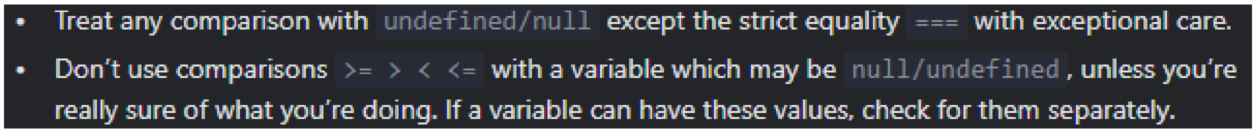
The reason is that an equality check '==' and comparisons '> < >= <=' work differently. Comparisons convert null to a number, treating it as 0. That's why (3) null >= 0 is true and (1) null > 0 is false.

- ➢ An incomparable undefined



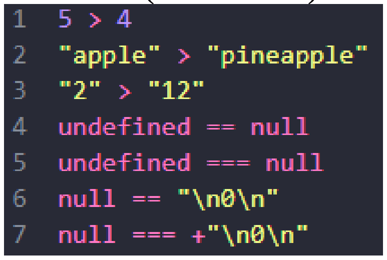
```
alert( undefined > 0 ); // false, bcs undefined gets converted to NaN
alert( undefined < 0 ); // false, bcs undefined gets converted to NaN
alert( undefined == 0 ); // false (3), bcs undefined only equals null
```

Avoid Problem:

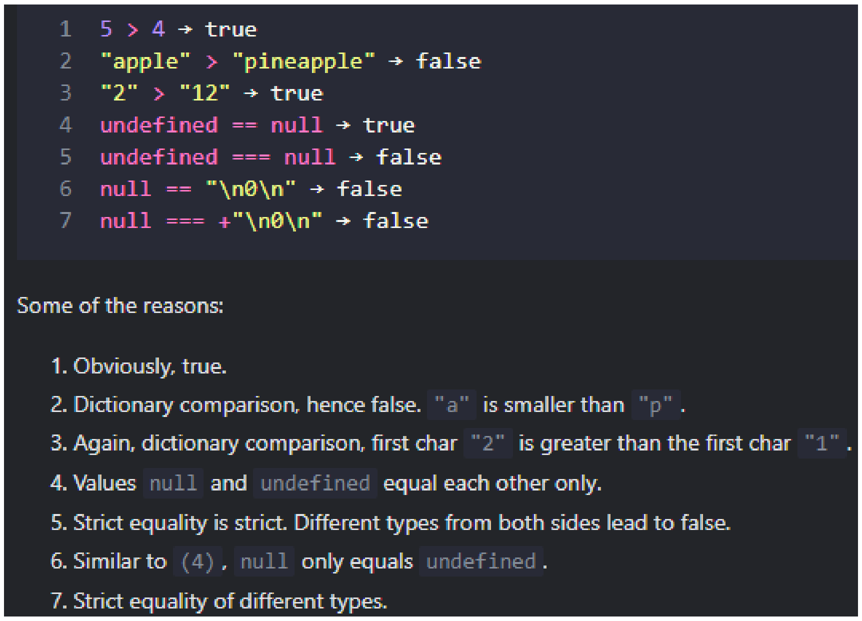


- Treat any comparison with `undefined/null` except the strict equality `===` with exceptional care.
- Don't use comparisons `>= > < <=` with a variable which may be `null/undefined`, unless you're really sure of what you're doing. If a variable can have these values, check for them separately.

Exercises: (true or false)



```
1  5 > 4
2  "apple" > "pineapple"
3  "2" > "12"
4  undefined == null
5  undefined === null
6  null == "\n0\n"
7  null === +"\n0\n"
```



```
5 > 4 → true
"apple" > "pineapple" → false
"2" > "12" → true
undefined == null → true
undefined === null → false
null == "\n0\n" → false
null === +"\n0\n" → false
```

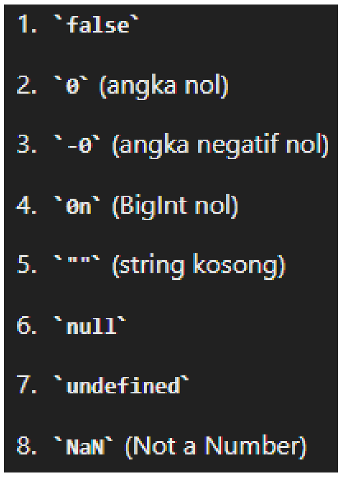
Some of the reasons:

1. Obviously, true.
2. Dictionary comparison, hence false. `"a"` is smaller than `"p"`.
3. Again, dictionary comparison, first char `"2"` is greater than the first char `"1"`.
4. Values `null` and `undefined` equal each other only.
5. Strict equality is strict. Different types from both sides lead to false.
6. Similar to `(4)`, `null` only equals `undefined`.
7. Strict equality of different types.

2. Logical Operator (CEK TASK paling bawah https://javascript.info/logical-operators)
   Penjelasan detail mudah dipahami: https://chatgpt.com/share/7952c32f-801f-4674-8743-be6a686bbc8c
   a. OR (||)
      Operator OR mengevaluasi setiap operand dari kiri ke kanan dan mengembalikan nilai pertama yang "truthy". Jika semua operand "falsy", maka ia mengembalikan nilai yg terakhir.

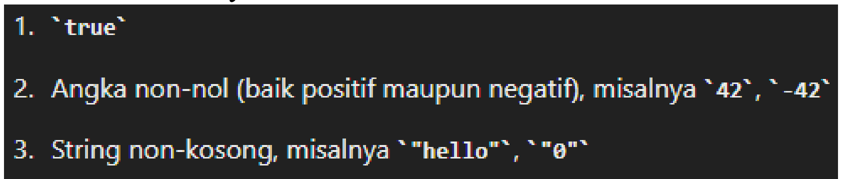
   b. AND (&&)
      Operator AND mengevaluasi setiap operand dari kiri ke kanan dan mengembalikan nilai pertama yang "falsy". Jika semua operand "truthy", maka ia mengembalikan nilai terakhir.

   c. Nilai-nilai "falsy"
      Nilai-nilai berikut akan dievaluasi sebagai false dalam konteks Boolean:

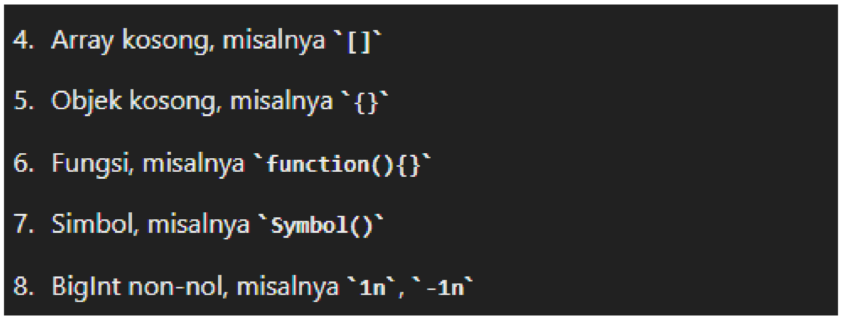


1. `false`
2. `0` (angka nol)
3. `-0` (angka negatif nol)
4. `0n` (BigInt nol)
5. `""` (string kosong)
6. `null`
7. `undefined`
8. `NaN` (Not a Number)

   d. Nilai-nilai "Truthy"



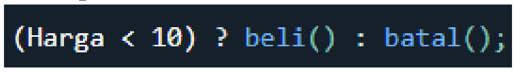
1. `true`
2. Angka non-nol (baik positif maupun negatif), misalnya `42`, `-42`
3. String non-kosong, misalnya `"hello"`, `"0"`


4. Array kosong, misalnya `[]`
5. Objek kosong, misalnya `{}`
6. Fungsi, misalnya `function(){}`
7. Simbol, misalnya `Symbol()`
8. BigInt non-nol, misalnya `1n`, `-1n`

3. Syntax
   - Ternary operator

     Syntax: condition ? <expression if true> : <expression if false>

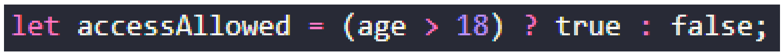
     Example:



```
(Harga < 10) ? beli() : batal();
```

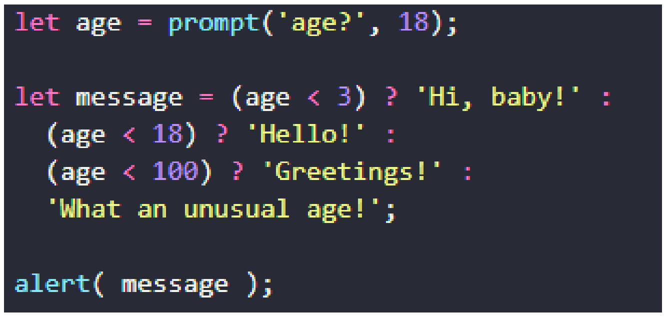
     Penjelasan: if (harga < 10) then beli() else batal();

     Other example:



```
let accessAllowed = (age > 18) ? true : false;
```

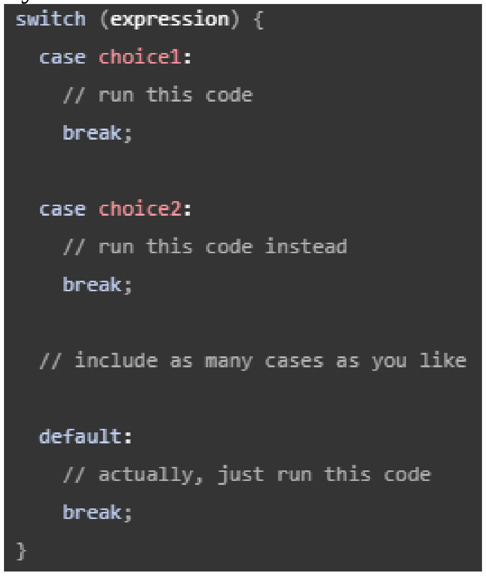
     Other example:



```
let age = prompt('age?', 18);

let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hello!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';

alert( message );
```

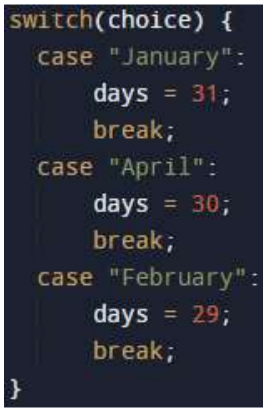
   - Switch

     Syntax:



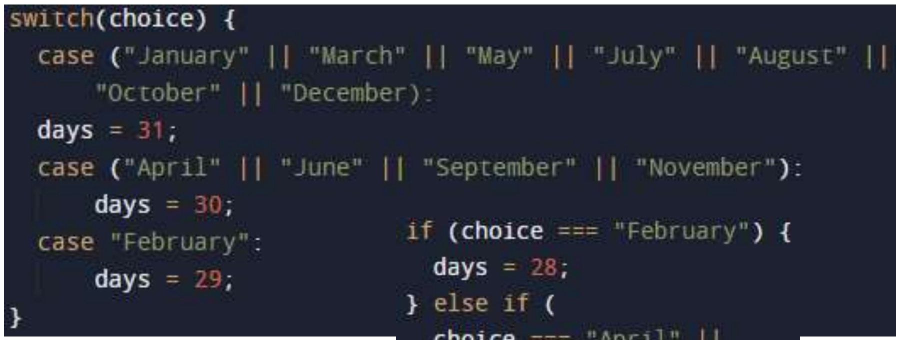
```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice2:
    // run this code instead
    break;

  // include as many cases as you like

  default:
    // actually, just run this code
    break;
}
```

Ex:



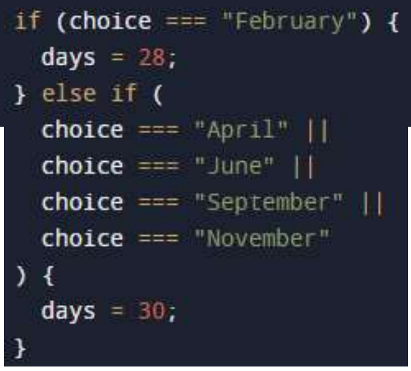
```
switch(choice) {
   case "January":
        days = 31;
        break;
   case "April":
        days = 30;
        break;
   case "February":
        days = 29;
        break;
}
```

Note: Tidak bisa seperti ini



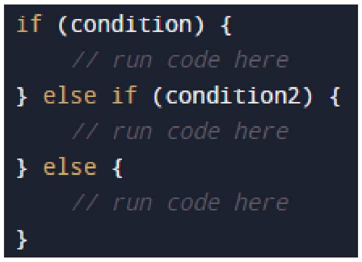
```
switch(choice) {
  case ("January" || "March" || "May" || "July" || "August" ||
      "October" || "December"):
  days = 31;
  case ("April" || "June" || "September" || "November"):
      days = 30;
  case "February":
      days = 29;
}
```

Solusinya:



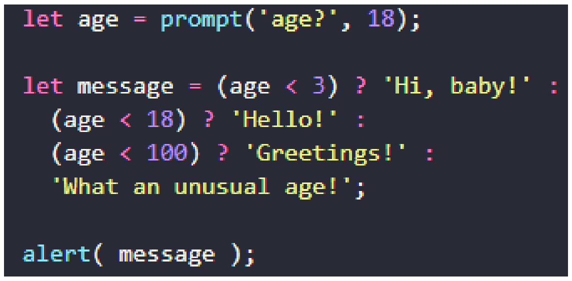
```
if (choice === "February") {
  days = 28;
} else if (
  choice === "April" ||
  choice === "June" ||
  choice === "September" ||
  choice === "November"
) {
  days = 30;
}
```

- If-else
  Syntax:



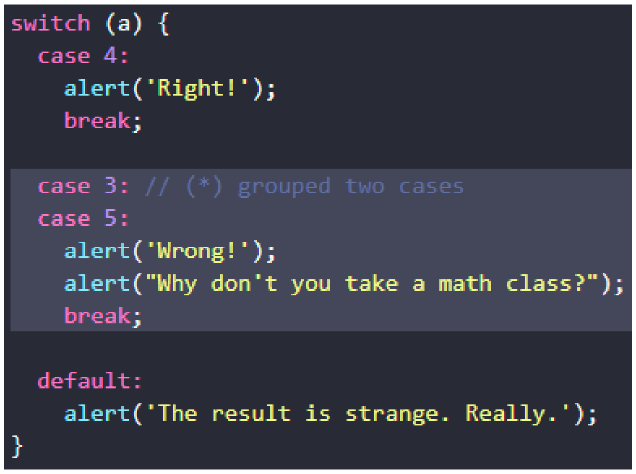
```
if (condition) {
    // run code here
} else if (condition2) {
    // run code here
} else {
    // run code here
}
```

- Multiple ternary operator
  The conditional similar with 'else if'



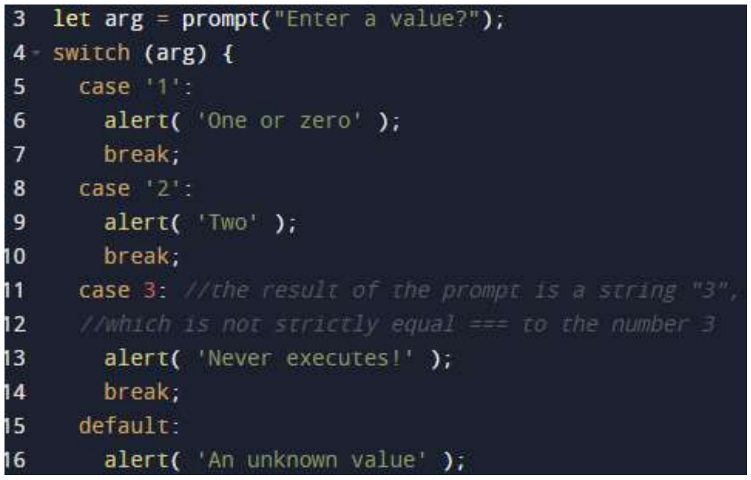
```
let age = prompt('age?', 18);

let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hello!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';

alert( message );
```

- Multiple switch 'case'



```
switch (a) {
  case 4:
    alert('Right!');
    break;

  case 3: // (*) grouped two cases
  case 5:
    alert('Wrong!');
    alert("Why don't you take a math class?");
    break;

  default:
    alert('The result is strange. Really.');
}
```

Note: Type Matters. Let's emphasize that the equality check is always strict. The case values must be the same type. Example:



```
let arg = prompt("Enter a value?");
switch (arg) {
  case '1':
    alert( 'One or zero' );
    break;
  case '2':
    alert( 'Two' );
    break;
  case 3: //the result of the prompt is a string "3",
  //which is not strictly equal === to the number 3
    alert( 'Never executes!' );
    break;
  default:
    alert( 'An unknown value' );
```