



Metode Array, Regex, dan Asymptotic Notation Silver- Chapter 1 - Topic 3

**Selamat datang di Chapter 1 Topic 3 pada course
React Native dari Binar Academy!**





Halloo, Binarian! 🙌

Kali ini kita masuk di sesi terakhir **Chapter 1 Topic 3** nih!

Setelah kemarin kita bahas seputar Fungsi dan Objek JavaScript, kali ini kita bakal belajar tentang **Metode Array, Regex, dan Asymptotic Notation** di JavaScript. Keren ya kedengerannya? Emang keren sih kalau kamu sudah menguasai ini.

Jadi, yuk langsung aja kita pelajari hal-hal itu. Semangat!





Detailnya, kita bakal bahas hal-hal berikut ini:

- Array dan Array Methods
- Regex
- Asymptotic Notation





Array & Array Methods





Bayangan...

Bayangan sekarang kamu lagi bikin aplikasi mobile. Kamu pengen nampilin daftar nama-nama produk pada aplikasi tersebut, lalu kamu bikin kodennya kayak gambar di samping.

Apakah bisa? Apakah boleh?

Bisa dan boleh saja sih kamu bikin kodennya kayak gitu, tapi..



```
const produk1 = 'Modem';
const produk2 = 'Hardisk';
const produk3 = 'Flashdisk';

console.log(produk1);
console.log(produk2);
console.log(produk3);
```



Tapi penulisan kode kayak di samping ini kurang efisien, Binarian.

Lho kok bisa nggak efisien?



```
const produk1 = 'Modem';
const produk2 = 'Hardisk';
const produk3 = 'Flashdisk';

console.log(produk1);
console.log(produk2);
console.log(produk3);
```

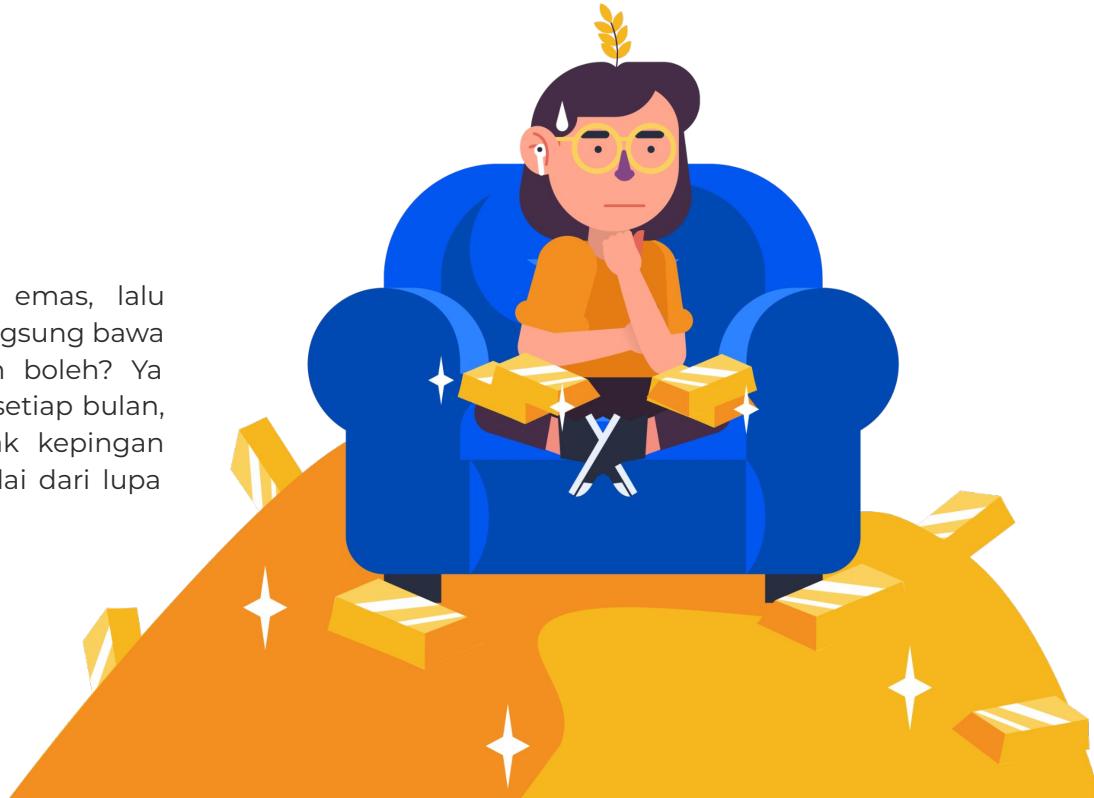


Kenapa?

Kita analogiin dulu deh.

Misalnya kamu mau mulai belajar investasi emas, lalu kamupun mulai beli emas di gerai Antam dan langsung bawa pulang kepingan emas itu ke rumah. Apakah boleh? Ya pastinya boleh dong. Tapi kalau kamu rutin beli setiap bulan, apakah kamu akan menyimpan begitu banyak kepingan emas di rumah? Tentu risikonya banyak ya, mulai dari lupa naro, tercecer, bahkan hilang.

Nah, kira-kira gitu.





Bikin deret kode kayak di halaman sebelumnya boleh saja sih, tapi gimana kalo ada 250 produk yang harus kamu tulis? Apakah kamu akan bikin 250 variabel dan melakukan console.log sebanyak 250 kali?

Capek juga yaaa..

Karena itu, untuk efisiensi kamu perlu pakai **Array**.



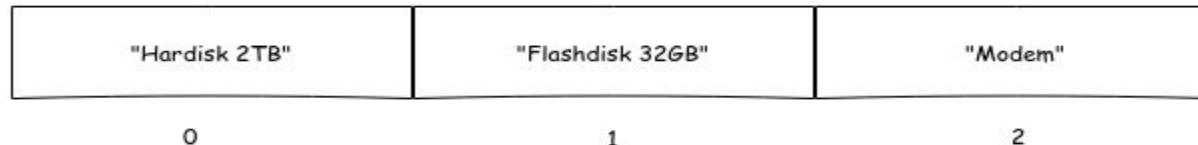


Jadi apakah Array itu?

Oke, sebelum kita bahas Array, kita bahas dulu **struktur data**. Struktur data merupakan cara-cara atau metode yang dipakai untuk menyimpan data di dalam memori komputer.

Hmm, terus apa kaitannya sama Array ya?

Nah ini, salah satu struktur data yang sering dipakai dalam pemrograman adalah **Array**. Array dipakai untuk **nyimpan sekumpulan data** dalam satu tempat dan setiap data dalam Array punya indeks sehingga kamu bisa gampang memprosesnya.





Gimana cara bikin Array di JavaScript?

Di JavaScript, kamu bisa bikin array dengan tanda kurung siku ([...]).



```
const produk = [];
```

Maka variabel produk akan berisi sebuah array kosong.



Kamu bisa isi data ke dalam array, lalu setiap data dipisah dengan tanda koma (,).



```
const produks = ["Flashdisk", "SDD", "Monitor"];
```

Karena JavaScript adalah bahasa pemrograman yang dynamic typing, maka kamu bisa menyimpan dan mencampur tipe data apapun di dalam array.



```
const produks = [12, 2.1, true, 'C', "Petanikode"];
```



Cara ngambil Data dari Array

Seperti yang kita tahu Array akan nyimpan sekumpulan data dan ngasih nomer indeks supaya data gampang diakses.

Indeks array selalu dimauli dari nol 0.



```
const makanan = ["Nasi Goreng", "Mie Ayam", "Mie Gelas"];
```

Oke, dengan data kayak di atas, gimana caranya kamu ngambil nilai "Mie Ayam"?

Ada yang bisa jawab?



```
const makanan = ["Nasi Goreng", "Mie Ayam", "Mie Gelas"];  
  
// cara mengambil nilai "Mie Ayam"  
console.log(makanan[1]) // => "Mie Ayam"
```

Yaps! Sama kayak contoh sebelumnya, kalau mau dapatin nilai lain selain Mie Ayam, kamu juga bisa pakai **makanan[0]** untuk dapat value “Nasi Goreng” dan **makanan[2]** untuk dapat value “Mie Gelas”.

Ingatya, indeks array selalu dimulai dari nol.



Balik lagi masalah penulisan kode.

Pada case sebelumnya yaitu saat kamu sedang bikin aplikasi mobile, lalu pengen nampilin daftar nama-nama produk pada aplikasi tersebut.

Kamu bisa saja pakai array untuk menampilkan list produk-produk tersebut dengan bikin kode seperti gambar disamping.



```
const products = ["Moden", "Hardisk", "Flashdisk"];  
  
// console.log semua produk  
console.log(products[0])  
console.log(products[1])  
console.log(products[2])
```



Tapi..

Gimana nanti kalo item di array ada 300?

Tentu saja kita nggak akan nulis 300 baris kode untuk nampilin isi semua array.

Solusinya adalah kita pakai pengulangan.

```
const products = ["Moden", "Hardisk", "Flashdisk"];  
  
// console.log semua produk  
console.log(products[0])  
console.log(products[1])  
console.log(products[2])
```



Pakai Array Method saja!

Yaps! Kamu bisa pakai **array method perulangan dengan method forEach**. Method forEach akan melakukan perulangan pada setiap item di array, kemudian dalam proses perulangan tersebut, kode di dalam blok forEach akan dieksekusi.

```
const products = ["Moden", "Hardisk", "Flashdisk"];  
  
// Method forEach dengan arrow function  
products.forEach((namaProduk) =>{  
    console.log(namaProduk);  
})  
  
// Method forEach dengan anonymous function  
products.forEach(function(namaProduk){  
    console.log(namaProduk);  
})
```



Array Method adalah..

Sebuah metode yang cuma bisa dipakai pada sebuah variabel yang menyimpan value-nilai berbentuk Array untuk melakukan manipulasi tertentu pada data array tersebut.

Jadi kalau tipe data dari variabel yang kamu bikin bukan Array, kamu nggak bakalan bisa pakai Array Method.

```
● ● ●

const minuman = ['Kopi susu', 'Teh Manis', 'Jahe Anget'];

// Method .push(itemBaru) adalah method yang akan menambahkan
// item pada akhir dari sebuah array
minuman.push('Air Mineral');
console.log(minuman); // => const minuman = ['Kopi susu', 'Teh Manis', 'Jahe Anget'];


const makanan = 'Ayam Goyeng';

// Variabel makanan tidak menyimpan value berbentuk Array.
// Karena value makanan bukan array, maka tidak bisa memanggil method push
// method .push(item) hanya bisa dipanggil pada variabel dengan value berbentuk array
makanan.push('Tempe goyeng');
console.log(makanan); // => Akan TypeError: makanan.push is not a function
```



Berikut beberapa array method yang sering digunakan:

```
● ● ●

// Method .push() => berfungsi menambahkan sebuah item pada akhir array
const array = [1, 2, 3, 4];
array.push(20) // => [1, 2, 3, 4, 20]

// Method .pop() => berfungsi menghapus item terakhir pada sebuah array
const array = [1, 2, 3, 4];
array.pop() // => [1, 2, 3]

// Method .unshift(70) => berfungsi menambahkan sebuah item pada awal array
const array = [1, 2, 3, 4];
array.unshift(70) // => [70, 1, 2, 3, 4]

// Method .shift() => berfungsi menghapus item pertama pada sebuah array
const array = [1, 2, 3, 4];
array.shift() // => [2, 3, 4]
```



Method .filter()

Method filter() berfungsi untuk **menyaring data** dari sebuah array.

Parameter yang harus diberikan pada method filter() sama seperti method forEach() sebelumnya, yaitu callback function.



```
const angka = [30, 28, 10, 23, 56, 31, 40, 100];

// Kita ingin filter array pada variabel angka, kita pengen
// ambil item yang angkanya lebih besar dari 30
const filteredArray = angka.filter((item) => {return item > 30});

console.log(filteredArray) // -> [ 56, 31, 40, 100 ]
```



```
const angka = [30, 28, 10, 23, 56, 31, 40, 100];

// Pake arrow function bisa tanpa return
const filteredArray = angka.filter((item) => item > 30);

console.log(filteredArray) // -> [ 56, 31, 40, 100 ]
```



Method .sort()

Method `sort()` berfungsi untuk **mengurutkan item** pada sebuah array.

```
● ● ●

const namaAnak = [
  'Andi',
  'Candrawana',
  'Budi',
  'Citra',
  'Yanti',
  'Siska',
  'Andika',
];
const sortedNamaAnak = namaAnak.sort();
console.log(sortedNamaAnak);
// Result => [ 'Andi', 'Andika', 'Budi', 'Candrawana', 'Citra', 'Siska', 'Yanti' ]
```



Explore more..

- | | | |
|----------------------------------|-----------------------------------|--------------------------------|
| 1. concat() | 11. includes() | 21. reduce() |
| 2. copyWithin() | 12. indexOf() | 22. reverse() |
| 3. entries() | 13. isArray() | 23. slice() |
| 4. every() | 14. join() | 24. some() |
| 5. fill() | 15. lastIndexOf() | 25. sort() |
| 6. filter() | 16. map() | 26. splice() |
| 7. find() | 17. push() | 27. toString() |
| 8. findIndex() | 18. pop() | 28. values() |
| 9. forEach() | 19. shift() | |
| 10. Array.from() | 20. unshift() | |

Masih ada banyak array method yang wajib kamu pahami dan kuasai., karena itu kamu harus nyoba dan bereksperimen sendiri pada metode-metode tersebut.

Yuk, eksplor array method lain pada beberapa tautan berikut ini:

<https://www.tutorialstonight.com/js/javascript-array-methods.php>

https://www.w3schools.com/js/js_array_methods.asp

https://www.w3schools.com/js/js_array_sort.asp

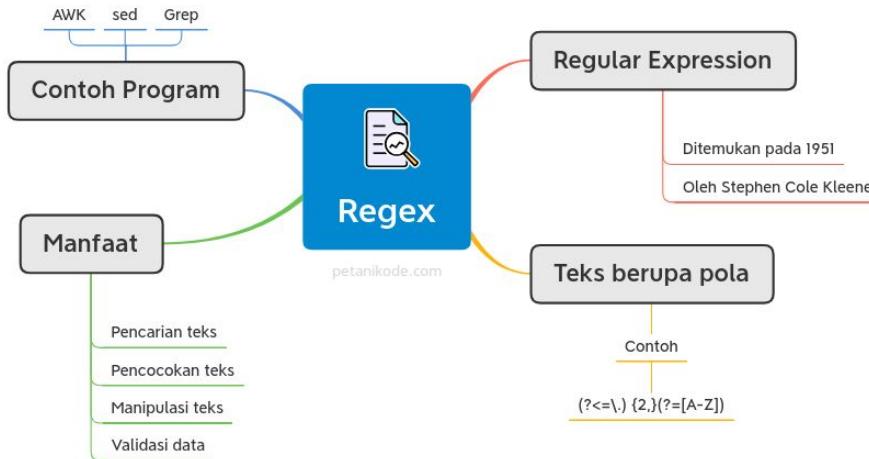
https://www.w3schools.com/js/js_array_iteration.asp



Regex Basic



Apa itu Regex?



Regular Expression atau disingkat **RegExp** atau **RE** adalah mekanisme pencocokan pola (*pattern matching*) yang dibikin pakai karakter-karakter khusus. Fungsinya beragam, mulai dari meriksa apakah sebuah inputan sudah sesuai atau belum (*test*), sampai bikin fitur pencarian (*search*) atau penggantian string (*replace*).



Contoh pendekripsi pola kalimat menggunakan Regex



```
const polaRegex = /[abcde]../;

console.log(polaRegex.test("abaa")); // true
console.log(polaRegex.test("fba")); // false
console.log(polaRegex.test("1dd")); // false
console.log(polaRegex.test("add")); // true
console.log(polaRegex.test(" dd")); // false
console.log(polaRegex.test("belajar")); // true
```

Pola/[abcde]../ pada syntax kode di samping berarti digit pertama cuma bisa diisi oleh salah satu dari huruf a, b, c, d atau e. Lalu salah satu huruf tersebut diikuti setidaknya oleh 2 karakter lain (bebas mau karakter apa saja).

Dengan begitu, string yang memenuhi syarat adalah: "abaa", "add" dan "belajar". String "fba" jadi false karena digit pertama di awali huruf f. String "1dd" juga false karena karakter pertama berupa angka, sedangkan string " dd" hasilnya false karena karakter pertama berisi spasi. Pola /[abcde]../ bisa juga ditulis jadi: /[a-e]../.



Asymptotic Notation/ Big O Notation

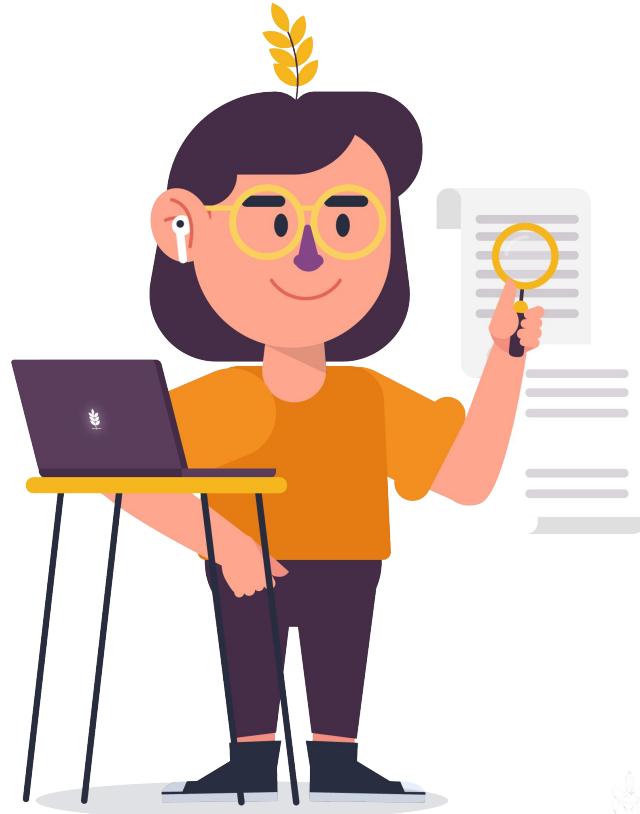




Big O Notation adalah salah satu konsep paling penting dalam Computer Science yang harus dimengerti oleh setiap programmer. Karena kalo kamu nggak ngerti dan pahami konsep ini, kemungkinan besar kamu akan nulis kode yang yang berdampak jelek pada performa aplikasi.

Jangan sampai deehh..

Makanya, penting banget buat kamu untuk bener-bener memahami materi ini ya, Binarian :)





Setiap programmer yang baik akan pakai cara yang paling efektif dan efisien dalam menyelesaikan suatu masalah. Untuk bisa melakukannya, kamu **harus bisa meminimalisir kerumitan atau kompleksitas dari algoritma yang kamu pakai.**

Kompleksitas suatu algoritma dibagi jadi 2, yaitu:

1. **Time Complexity** adalah seberapa lama waktu yang diperlukan untuk jalankan suatu algoritma.
2. **Space Complexity** adalah seberapa besar memori yang dipakai untuk jalankan suatu algoritma.

Kali ini, kita hanya akan membahas tentang **Time Complexity**.





Balik sebentar ke Big O Notation

Big O Notation dipakai dalam computer science untuk **menganalisis/mengukur performa dari sebuah algoritma** (ssttt, kalau kamu belum tahu, sebenarnya algortima adalah sebutan lain dari function pada JavaScript lho).

Big O Notation adalah konsep perhitungan tenaga yang dibutuhin buat jalanin sebuah algoritma atau sebuah function. Maksud ‘tenaga’ disini adalah **waktu** dan **memori komputer** yang dibutuhin untuk jalanin function.



Constant Time Complexity/Big O(1)

Coba tebak, pada syntax kode di bawah ini, kira-kira mana proses yang butuh perhitungan lebih lama?

2 dikali 5 atau 2 dikali 1397?

```
● ● ●

function perkalianAngkaDua(num) {
    return 2 * num;
}
let result = perkalianAngkaDua(5); // 10
let result2 = perkalianAngkaDua(1397); // 2794
```



Oke jangan main nebak-nebak aja. Begini jawabannya, Sob..

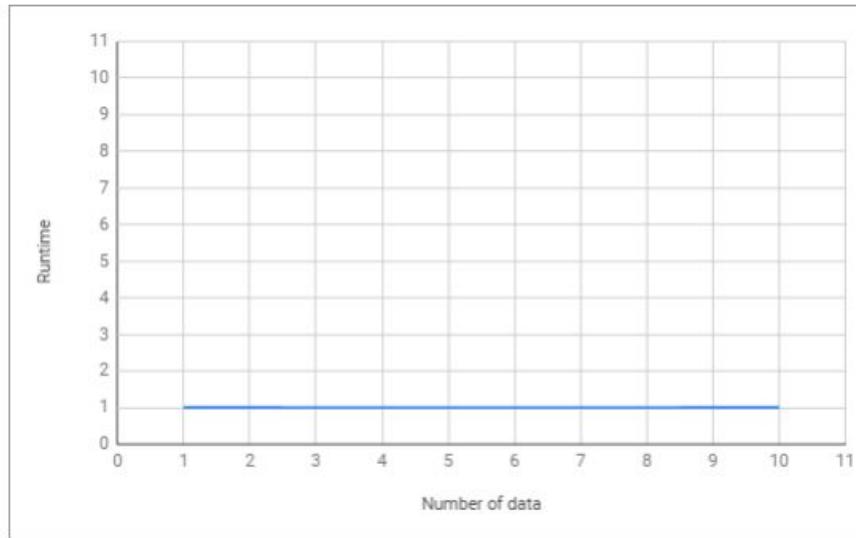
Kalau kamu minta seseorang untuk ngitung antara **2 dikali 5** dan **2 dikali 1397**, pasti orang itu akan butuh waktu lebih lama untuk menghitung hasil dari **2 dikali 1397** dibandingkan menghitung hasil dari **2 dikali 5**. Iya nggak?

Nah beda halnya dengan komputer. Komputer nggak butuh perbedaan waktu untuk ngitung kedua perhitungan tersebut. Waktu yang dibutuhkan untuk menghitung 2 dikali 5 dan **2 dikali 1397** sama, nggak ada bedanya sama sekali.

Kok bisa?.

Karena komputer menganggap itu hanyalah satu operasi perkalian yang sama, sehingga **nggak peduli berapapun angkanya**, selama itu proses operasi perkalian, maka **waktu yang dibutuhkan untuk memproses perhitungannya akan tetap sama**.

```
● ● ●  
function perkalianAngkaDua(num) {  
    return 2 * num;  
}  
let result = perkalianAngkaDua(5); // 10  
let result2 = perkalianAngkaDua(1397); // 2794
```



Nah, jadi berdasarkan contoh case di atas, bisa kita tarik benang merah kesimpulan bahwa **Constant Time** atau banyaknya input yang diberikan kepada sebuah algoritma, ngak akan mempengaruhi waktu proses (*runtime*) dari algoritma tersebut.



Logarithmic Time / O(n)

Pada syntax kode di bawah ini, kira-kira mana yang butuh waktu proses lebih lama untuk menghasilkan output?

reversedArray1 atau **reversedArray2**?



```
function reverseArray(arr) {
  let newArr = []
  for (let i = arr.length - 1; i >= 0; i--) {
    newArr.push(arr[i])
  }
  return newArr
}

const reversedArray1 = reverseArray([1, 2, 3]) // Output => [3, 2, 1]
const reversedArray2 = reverseArray([1, 2, 3, 4, 5, 6]) // Output => [6, 5, 4, 3, 2, 1]
```



```
function reverseArray(arr) {
    let newArr = []
    for (let i = arr.length - 1; i >= 0; i--) {
        newArr.push(arr[i])
    }
    return newArr
}

const reversedArray1 = reverseArray([1, 2, 3]) // Output => [3, 2, 1]
const reversedArray2 = reverseArray([1, 2, 3, 4, 5, 6]) // Output => [6, 5, 4, 3, 2, 1]
```

Jawabannya adalah saat memproses **reversedArray2** komputer akan butuh waktu lebih lama dilama dibanding saat memproses **reversedArray1**. Kenapa? Karena pada function tersebut kamu melakukan perulangan pada setiap item di dalam array itu.

Pada **reversedArray1** kamu melakukan 3 kali operasi, yaitu melakukan perulangan sebanyak 3 kali karena array yang ada pada reversedArray1 cuma sebanyak 3 item, beda halnya pada **reversedArray2** yang butuh 6 operasi perulangan.



Jadi berdasarkan contoh case di atas, kita bisa ditarik sebuah benang merah kesimpulan bahwa..

Logarithmic Time artinya saat kita ngasih input sebesar n terhadap sebuah function, jumlah tahapan yang dilakukan oleh function tersebut bergantung dengan jumlah input data yang akan dimasukan sebagai parameter. Sederhananya, **semakin banyak jumlah data input, maka semakin banyak pula waktu yang dibutuhkan function tersebut.**



Saatnya kita Quiz!





1. Apa output yang dihasilkan, apabila syntax kode berikut dieksekusi?

```
● ● ●  
let num = 1;  
const listHewan = ['Kucing', 'Beruang', 'Panda', 'Sapi'];  
  
console.log(listHewan[(num += 1)]);
```

- A. Panda
- B. Beruang
- C. Syntax Error



1. Apa output yang dihasilkan, apabila syntax kode berikut dieksekusi?

```
● ● ●  
let num = 1;  
const listHewan = ['Kucing', 'Beruang', 'Panda', 'Sapi'];  
  
console.log(listHewan[(num += 1)]);
```

- A. Panda
- B. Beruang
- C. Syntax Error

num += 1 adalah 2, maka index ke 2 dari array listHewan



2. Apa output yang dihasilkan, kalau syntax kode di atas dieksekusi?

```
const listHewan = [ 'Kucing', 'Beruang', 'Panda', 'Sapi' ];

listHewan.slice(0, 1);
listHewan.splice(0, 1);
listHewan.unshift('Gajah');

console.log(listHewan);
```

- A. ['Gajah', 'Beruang', 'Kucing', 'Sapi']
- B. ['Gajah', 'Beruang', 'Kucing', 'Panda', 'Sapi']
- C. ['Gajah', 'Beruang', 'Panda', 'Sapi']



2. Apa output yang dihasilkan, kalau syntax kode di atas dieksekusi?

```
const listHewan = ['Kucing', 'Beruang', 'Panda', 'Sapi'];

listHewan.slice(0, 1);
listHewan.splice(0, 1);
listHewan.unshift('Gajah');

console.log(listHewan);
```

- A. ['Gajah', 'Beruang', 'Kucing', 'Sapi']
- B. ['Gajah', 'Beruang', 'Kucing', 'Panda', 'Sapi']
- C. ['Gajah', 'Beruang', 'Panda', 'Sapi']

.splice(0,1) akan membuat 'Kucing' dari array, kemudian .unshift('Gajah') akan menambahkan item 'Gajah' pada urutan item pertama array



3. Di bawah ini yang bukan syntax kode untuk menghasilkan data pada Gambar B, berdasarkan array pada Gambar A adalah..

- A. listHewanVaksin.filter((item) => !item.sudahVaksin)
- B. listHewanVaksin.filter((item) => !false)
- C. listHewanVaksin.filter((item) => item.sudahVaksin === false)



// GAMBAR A

```
const listHewanVaksin = [
  {
    namaHewan: 'Kucing',
    sudahVaksin: false,
  },
  {
    namaHewan: 'Sapi',
    sudahVaksin: true,
  },
  {
    namaHewan: 'Gajah',
    sudahVaksin: false,
  },
];
```



// GAMBAR B

```
// dapatkan hewan-hewan yang belum vaksin
[
  { namaHewan: 'Kucing', sudahVaksin: false },
  { namaHewan: 'Gajah', sudahVaksin: false }
]
```



3. Di bawah ini yang bukan syntax kode untuk menghasilkan data pada Gambar B, berdasarkan array pada Gambar A adalah..

- A. listHewanVaksin.filter((item) => !item.sudahVaksin)
- B. listHewanVaksin.filter((item) => !false)
- C. listHewanVaksin.filter((item) => item.sudahVaksin === false)

! adalah kebalikan atau lawan kata, jadi kalau !false artinya true



// GAMBAR A

```
const listHewanVaksin = [
  {
    namaHewan: 'Kucing',
    sudahVaksin: false,
  },
  {
    namaHewan: 'Sapi',
    sudahVaksin: true,
  },
  {
    namaHewan: 'Gajah',
    sudahVaksin: false,
  },
];
```



// GAMBAR B

// dapatkan hewan-hewan yang belum vaksin

```
[
  { namaHewan: 'Kucing', sudahVaksin: false },
  { namaHewan: 'Gajah', sudahVaksin: false }
]
```



Penutup

Nah, akhirnya selesai juga pembahasan kita di Chapter 1 Topic 3 ini. Kita akan belajar hal baru lain di Topik 4 pada sesi berikutnya.

See youuu!!



Terima Kasih!



Next Topic

loading...