Modul praktikum - Minggu 09 - *Arrays*

Dosen pengampu: **Henokh Lugo Hariyanto**
Asisten mata kuliah: **Jein Ananda - (10221031)**; **Muhammad Aulia Rahman - (10221055)**

Tujuan:

- Mampu memahami terminologi dasar terkait *arrays*.
- Mampu menuliskan penyusunan *arrays*.
- Mampu melakukan *indexing* pada *arrays*.
- Mampu menggunakan *arrays* dengan *control structure* `for/of` dan *spread operators*.
- Mampu menggunakan beberapa *methods* yang tersedia di dalam *arrays*

> Tips belajar bahasa pemrograman adalah mengetik ulang perintah yang kita temukan di buku atau di internet, lalu kita ubah-ubah untuk menguji pemahaman kita sudah tepat atau belum. Faktor bermain-main dan eksplorasi sangat diperlukan untuk memahami setiap perintah bahasa pemrograman yang kita pelajari. Setiap potongan kode di bawah dapat ditulis dalam berkas `.js` lalu dapat di-*running* dengan Node.js.

Arrays adalah suatu *type* yang berupa kumpulan beberapa *expression*. Tiap-tiap *expression* ini dipisahkan oleh tanda koma, dan *expression* tersebut disebut *element*. Kumpulan *expression* ini diapit oleh tanda kurung siku. Pembentukan *array* semacam ini disebut *array literals*. Urutan *expression* itu muncul dalam *array* sangatlah penting. Berbeda urutan element, berarti menunjukkan array yang berbeda pula. Urutan dalam *array* ini disebut *index*. *Index* atau posisi numerik dari array ini sangat penting diketahui untuk mengakses element yang kita ingin tampilkan (proses *querying*). Di dalam JavaScript *index* berupa bilanga bulat dimulai dari angka 0 hingga maksimum dari bilangan integer positif yang mampu diwakili oleh komputer. Di dalam mesin yang saya miliki berada di kisaran $2^{1024} \approx 10^{308}$ bandingkan dengan jumlah butiran pasir di seluruh dunia di sekitar 10^{27} (du Sautoy, 2012). Tetapi di dalam prakteknya kita tidak pernah menggunakan element array berukuran mencapai 10^{308} karena keterbatasan memory dan CPU yang kita miliki.

Beberapa sifat-sifat *array* yang perlu dipahami adalah

- *untyped*: tipe-tipe data yang dimiliki oleh tiap element di dalam *array dapat berbeda-beda
- *dynamics*: ukuran (jumlah element) array dapat berubah-ubah sesuai kebutuhan

Pada bagian berikutnya kita akan membahas hal berikut:

- Cara membuat array
- Membaca dan menuliskan elemen suatu array
- Membuat *sparse array*
- Mengetahui ukuran *array*
- Menambahkan dan menghapus element suatu array
- Melakukan iterasi pada element array
- Array multidimensi (*nested array*)
- Beberapa *methods* di dalam *arrays*
- Object yang memiliki sifat seperti array
- Melihat *string* sebagai *array*

Creating array

Ada beberapa cara untuk membuat array

- Menggunaan *array literals*

  create-arr-literal.js

```
let empty = [];                 // Array tanpa element
let primes = [2, 3, 5, 7, 11];  // Array dengen element berupa bilangan
prima
let various = [1, 'apple', true, 3.14]    // Array dengan elemen berbagi
`type`

// Array dengan elemen berupa `expression`
let base_address = 'Perum. ';
let addresses = [
  base_address + 'Bangun Reksa',
  base_address + 'Graha Indah',
  base_address + 'Grand City'];
console.log(addresses);

// Array yang memuat dua *arrays* dan masing-masing array tersebut
// memuat satu bilangan integer dan satu object yang memiliki
// dua properties bernama `x` dan `y`
let mixed_arr = [
  [1, {x: 1, y: 2}],
  [2, {x: 3, y: 4}]
]
console.log(mixed_arr);

// Array dengan ukuran 4. Tanda koma setelah koma dianggap
// element `undefined`
let arr_with_undefined = ['a', ,'c', ,'e', , ,];
console.log(arr_with_undefined)
console.log(arr_with_undefined.length);
console.log(typeof arr_with_undefined[1]);

// Array dengan ukuran 2. Tanda koma setelah
// koma dianggap element `undefined`
let non_zero_length = [ , , , ];
console.log(non_zero_length.length);
```

- Menggunakan spread operator

  create-arr-spread-op.js

```
// Menyisipkan array a ke array b
let arr1 = ['a', 'b', 'c'];
let arr2 = ['z', ...arr2, 'x'];
console.log(arr2);
```

```
// Spread operator dapat digunakan untuk menggandakan array
let original = ['a', 'b', 'c'];
let copy = [...original];
copy[0] = 0;      // Mengubah element index 0 array `copy` tidak mengubah
                  // element index 0 array `original`
console.log(original[0])


// Menggunakan spread operator untuk string akan mengubahnya menjadi array
let digits = [..."0123456789ABCDEF"];
console.log(digits);
```

- Menggunakan object `Array()`

  create-arr-arr-obj.js

```
let a = new Array();
console.log(a);   // Array tanpa element, setara dengan []

let b = new Array(10);
console.log(b);   // Array kosong namun dengan ukuran 10

// Mendata semua elemen, meskipun tidak efisien dibandingkang
// menggunakan array literal
let c = new Array('e', 'd', 'c', 'b', 'a', 'hello world');
console.log(c);
```

- Menggunakan `Array.of()`

  create-arr-arr-of-and-from.js

```
let arr1 = Array.of();       // array kosong tanpa element
let arr2 = Array.of(10);     // membuat array dengan satu element: [10]
let arr3 = Array.of('a', 'b', 'c');

let sourceArr = ['a', 'b', 'c'];
let targetArr = Array.from(sourceArr);
targetArr[0] = 'z';
console.log(sourceArr[0]);
```

Melakukan pembacaan dan penulisan *elements* pada *array*

Pembacaan dan penulisan *elements* di *array* sangat mirip dengan *querying and setting property* di *object* yang sudah dibahas pada pertemuan sebelumnya

read-and-write-elements.js

```
let a = ['world'];    // Definisikan array dengan satu element string 'world'
let value = a[0];     // Membaca element di index 0 dan memberikan nilainya
                      // ke variable `value`
console.log(value)

a[1] = 3.14;          // Menuliskan element di index 1
console.log(a);

// Indexing dapat dilakukan dengan bilangan tidak bulat (non-integer)
// dan negatif yang akan menghasilkan suatu element berupa object
a[-1.23] = true;      // akan menambahkan object dengan property '-1.23': true
console.log(a);

// Index dengan string yang bernilai bilangan bulat akan diproses
// sebagai index bilangan bulat
a["100"] = 0;
console.log(a);

// Index dengan bilangan desimal (namun angka dibelakang koma adala nol)
// akan diproses sebagai index bilangan bulat
a[1.000] = 1;
console.log(a);

a = [true, false];
// Membaca element yang tidak terdapat di suatu array akan memberikan `undefined`
console.log(a[2]);

// Berbeda dengan bahasa pemrograman Python, kita tidak dapat menggunakan index
// bilangan bulat negatif untuk membaca element di index terakhir
console.log(a[-1]);   // => undefined
```

Perlu diperhatikan perbedaan antara *index* suatu *array* dan *property name* suatu *object*. Meskipun keduanya sangat mirip, namun keduanya memiliki perbedaan yang mendasar. Semua *indexes* (bilangan integer dari 0 hingga `Number.MAX_VALUE`) dapat digunakan sebagai `property name`. Namun tidak semua *property names* dapat digunakan sebagai `index`.

Membuat *sparse array*

Sparse array adalah suatu array yang mayoritas elementnya belum didefinisikan. Contoh element yang ada nilainya hanya ada di index = 0 dan index = 100. Index selain kedua index tersebut (index = 1 hingga index = 99) akan didefinisikan nilainya sebagai `undefined`.

sparse-arr.js

```
let a = new Array(5);     // membuat *sparse array* yang berukuran 5

// Mengubah ukuran array `a` dengan menuliskan elemen ke-1001.
a[1000] = 0;
console.log(a);
```

```
// Ketiga array berikut meskipun memiliki nilai element yang sama (`undefined`)
// ketika di print out menggunakan console.log(), namun mereka memiliki
// karakteristik yang berbeda
let arr1 = [];            // Array tanpa element dengan ukuran 0
let arr2 = [,];           // Array tanpa element dengan ukuran 1
let arr3 = [undefined];   // Array dengan element `undefined` dengan ukuran 1
```

Mengetahui ukuran *array*

Untuk mengetahui ukuran suatu *array* dapat digunakan perintah `.length`; Melakukan setting ke property `.length` akan memodifikasi ukuran array (array akan terpotong atau diperpanjang).

arr-length.js

```
console.log([].length);     // => 0; array [] tidak memiliki element

// memiliki ukuran (jumlah element) 3
console.log(['a', 'b', 'c'].length); s // => 3

// Pemotongan dan pemanjangan ukuran array
let arr = ['a', 'b', 'c', 'd', 'e'];
arr.length = 3;
console.log(arr);   // => ['a', 'b', 'c']

arr.length = 0;     // hapus semua elements
console.log(arr);   // => []

arr.length = 5;     // => array tanpa elements namun berukuran 5
console.log(arr);
```

Menambahkan dan menghapus element suatu array

Ada dua cara untuk menambahkan element yaitu menggunakan *index* dan *push*

Untuk menghapus suatu element dapat menggunakan operator delete dan diikuti pembacaan array untuk *index* yang ingin dihapus

add-and-delete-element.js

```
let arr1 = ['a', 'b', 'c'];

// Penambahan dilakukan dimulai dari ukuran element,
// Jika indeks merupakan indeks element yang sebelumnya sudah terdapat element
// maka yang terjadi bukan penambahan melainkan mengganti nilainya
arr1[3] = 'zero';
arr1[4] = 'one';
console.log(arr1);
```

```
let arr2 = ['a', 'b', 'c'];
arr2.push('zero');
arr2.push('one')
```

Pada bagian selanjutnya akan di bahas menggunakan cara yang lebih elegan menggunakan *splice()*. *splice method* ini mampu melakukan penyisipan penghapusan, dan penggantian element.

Melakukan iterasi pada element array

Untuk dapat mencacah satu per satu element dalam *array* dapat digunakan

- *control structure* `for/of` loop,
- *control structure* `for` loop,
- *control structure* `for/of` loop dan `.entries()`

iterate-arr.js

```
let letters, string;
letters = [..."Hello world"];

// Menggunakan for/of loop
string = "";
for (let letter of letters) {
  string += letter;
}
console.log(string);      // Didapatkan hasil yang serupa dengan `letters`


// Menggunakan `for`
string = "";
for (let i = 0; i < letters.length; i++) {
  string += letters[i];
}
console.log(string);


// Menggunakan for/of loop dan .entries()
string = "";
for (let [index, letter] of letters.entries()) {
  string += letters[i];
}
console.log(string);
```

Array multidimensi (*nested array*)

Element dari suatu *array* dapat juga berupa *array*. Proses ini terus dapat diulangi tergantung dengan jumlah dimensi *array* yang ingin kita buat

multidim-arr.js

```
// Contoh data dengan 2D array
let expenseAndPrice = [
  ["Breakfast at locak food stall", 20_000],
  ["Transportation to work", 10_000],
  ["Lunch with colleagues", 50_000],
  ["Snacks from a convenience store", 15_000],
  ["Dinner at home", 30_000]
];
console.log(expenseAndPrice);
console.log();

// Contoh data dengen 3D array
let weeklyExpenses = [
  ["Monday, March 21", [
      ["Breakfast at locak food stall", 20_000],
      ["Transportation to work", 10_000],
      ["Lunch with colleagues", 50_000],
      ["Snacks from a convenience store", 15_000],
      ["Dinner at home", 30_000]
    ]
  ],
  ["Tuesday, March 22", [
      ["Breakfast at home", 10_000],
      ["Coffee from a cafe", 25_000],
      ["Groceries for the week", 150_000],
      ["Lunch from a food delivery app", 35_000],
      ['Snack from a street vendor', 10_000]
    ]
  ]
]
console.log(weeklyExpenses);
```

Beberapa *methods* di dalam *arrays*

Methods dalam *array* sangatlah banyak karena luasnya penerapan struktur data atau *type* ini. Kita dapat kelompokkan menjadi 4 grup *methods* yang terdapat di dalam *array*:

1. *Methods* yang berfungsi untuk melakukan iterasi
2. *Methods* yang berfungsi untuk pembentukan struktur data *stack* (LIFO: Last-In First Out) dan *queue* (FIFO: First-In First Out).
3. *Methods* yang bekerja pada sebagian elements *array* (*subarray*) untuk melakukan pengambilan, penghapusan, penyisipan, pengisian, dan pengggandaan *elements*.
4. *Methods* yang digunakan untuk mencari posisi *index* suatu *element* dan pengurutan *elements* suatu *array*

Array iterator methods

- `forEach()`
  Melakukan iterasi tanpa *control structures" `for` loop. Teknik ini banyak digunakan dalam paradigma pemrograman fungsional (di bahas lebih lanjut di perkuliahan *web pemrograman* tingkat lanjut). Beberapa contoh berikut banyak menggunakan input berupa *arrow function* yang akan kita bahas di pertemuan berikutnya tentang fungsi.

  for-each-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];
let concat = '';       // untuk menyimpan hasil concatenation

// Argument dari .foreach merupakan suatu fungsi
// Fungsi akan kita bahas di pertemuan praktikum berikutnya
data.forEach(value => { concat += value; });
console.log(concat);
```

- `map()`
  Merupakan *method* yang ringkas untuk mengaplikasikan fungsi ke setiap *element* di dalam *array*, tanpa *control structure* `for` loop.

  map-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];
let result = data.map(x => `item_${x.toUpperCase()}`);
console.log(result);
```

- `filter()`
  Digunakan untuk menyeleksi menggunakan nilai Boolean, *elements* yang akan di ambil dari suatu *array*

  filter-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];
let result = data.filter(x => {x >= 'a' && x <= 'c';});
console.log(result);
```

- `find()` dan `findIndex()`
  Digunakan untuk untuk mencari *element* di suatu *array*. `find()` akan menghasilkan semua *elements* yang ditemukan. Sedangkan `findIndex()` akan memberikan posisi dari semua *element** yang ditemukan

  find-and-find-index-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];
let result;
```

```
result = data.find(x => {x >= 'a' && x <= 'c';});
console.log(result);


result = data.findIndex(x => {x >= 'a' && x <= 'c';});
console.log(result);
```

- `every()` dan `some()`
  Digunakan untuk memeriksa apakah semua *elements* di dalam suatu *array* telah memenuhi kondisi Boolean yang diberikan.

  every-and-some-demo.js

  ```
  let data = ['a', 'b', 'c', 'd', 'e'];
  let result;

  result = data.every(x => {x >= 'a' && x <= 'c';});
  console.log(result);

  result = data.some(x => {x => 'a' && x <= 'c';});
  console.log(result);
  ```

- `reduce()` dan `reduceRight()`
  Digunakan untuk mereduksi *elements* di suatu *array* menjadi satu nilai tunggal. Maksud mereduksi disini adalah kita dapat mendefinisikan bagaimana dua *elements* di dalam *array* berinteraksi (dioperasikan/digabungkan/dikombninasikan). Untuk `reduceRight()` adalah variasi dari `reduce()` namun operasi reduksi dimulai dari dua pasangan terakhir *element* paling kanan.

  reduce-and-reduce-right-demo.js

  ```
  let data = ['a', 'b', 'c', 'd', 'e'];
  let result;

  result = data.reduce((x, y) => x + '+' + y);
  console.log(result);

  result = data.reduceRight((x, y) => x + '+' + y);
  console.log(result);
  ```

Methods for creating stacks and queues

Pada bagian ini ada 4 *methods* yaitu:

- `push()`: digunakan untuk menambahkan satu atau lebih *element* di ujung kanan *array* (di *index* terakhir)
- `pop()`: digunakan untuk menghapus satu *element* di ujung kanan *array* (di *index* terakhir)

- `unshift()`: digunakan untuk menambahkan satu atau lebih *element* di ujung kiri *array* (di *index* pertama*);
- `shift()`: digunakan untuk menghapus satu *element* di ujung kiri *array* (di *index* pertama);

stack-end-and-start-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];

stack_end = Array.from(data);     // menggandakan array `data`
stack_end.push('f', 'g'); console.log(stack_end);

// Menghapus satu per satu element terakhir
stack_end.pop(); console.log(stack_end);
stack_end.pop(); console.log(stack_end);
stack_end.pop(); console.log(stack_end);


console.log();  // untuk memberi spasi dari print out sebelumnya
stack_start = Array.from(data);     // menggandakan array `data`
stack_start.unshift(7, 8, 9); console.log(stack_start);

// Menghapus satu per satu element pertama
stack_start.shift(); console.log(stack_start);
stack_start.shift(); console.log(stack_start);
stack_start.shift(); console.log(stack_start);
```

Methods on subarrays

- `slice()`
  Digunakan untuk merndapatkan *subarray* dari suatu *array*.

  slice-demo.js

  ```
  let data = ['a', 'b', 'c', 'd', 'e'];
  let result;

  // Lakukan pengambilan subarray dari index 0 hingga index 3 - 1;
  result = data.slice(0, 3);
  console.log(result);

  // Lakukan pengambilan subarray dari index 3
  result = data.slice(3);
  console.log(result);

  // Lakukan pengambilan subarray dari index 1, hingga index terakhir (-1)
  result = data.slice(1, -1);
  console.log(result);

  // Lakukan pengambilan subarray dari index -3 (tiga terakhir dari kanan)
  ```

```
// hingga index -2 (dua terakhir dari kanan)
result = data.slice(-3, -2);
console.log(result);
```

- `splice()`
  Merupakan *array method* yang memiliki kegunaa beragam, seperti yang telah disinggung di paragraf sebelumnya

  splice-demo.js

```
let data, result;

// Melakukan pemotongan array di indeks 4
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];
result = data.splice(4);
console.log(result);
console.log(data);
console.log();

// Melakukan pemotongan array di indeks 1 hingga indeks 2
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];
result = data.splice(1, 2);
console.log(result);
console.log(data);
console.log();

// Melakukan pemotongan array di indeks 1 hingga indeks 1
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];
result = data.splice(1, 1);
console.log(result);
console.log(data);
console.log();

// Melakukan penghapusan dimulai dari index 2 sebanyak 0 element
// kemudian disisipkan dua elements `fill01` dan `fill02`
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];
result = data.splice(2, 0, 'fill01', 'fill02');
console.log(result);
console.log(data);
console.log();

// Melakukan penghapusan dimulai dari index 2 sebannyak 2 elements
// kemudian disisipkan dua elements [1, 2] dan 5
data = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'];
result = data.splice(2, 2, [1, 2], 5);
console.log(result);
console.log(data);
```

- `fill()`
  Digunakan untuk pengisian *elements* pada *array*.

fill-demo.js

```
let data;

// Mengisi semua element dengan angka 0
data = ['a', 'b', 'c', 'd', 'e'];
data.fill(0);
console.log(data);

// Mengisi array dengan angka 9 dimulai dari indeks 1 hingga akhir
data = ['a', 'b', 'c', 'd', 'e'];
data.fill(9, 1);
console.log(data);

// Mengisi array dengan angka 8 dimulai dari indeks 2 hingga indeks
// terakhir dikurangi 1 (indeks terakhir kedua)
data = ['a', 'b', 'c', 'd', 'e'];
data.fill(8, 2, -1);
console.log(data);
```

- `copyWithin()`
  Digunakan untuk menggandakan *subarray* di dalam *array* tersebut ke posisi yang baru.

copyWithin()

```
let data;

// Menggandakan array data ditempatkan mulai dari indeks 1.
// Element pada posisi indeks 1 hingga akhir akan di-overwrite.
data = ['a', 'b', 'c', 'd', 'e'];
data.copyWithin(1);
console.log(data);

// Menggandakan array data ditempatkan mulai dari indeks 2
// dari array data yang dimulai dari indeks 3 hingga element ke 5
// (indeks ke = 5-1 = 4) =  ['d', 'e']
// Element pada indeks 3 dan 4 akan di-overwrite dengan ['d','e']
data = ['a', 'b', 'c', 'd', 'e'];
data.copyWithin(2, 3, 5);
console.log(data);

// Menggandakan array data ditempatkan mulai dari indeks 0
// dari array data terakhir kedua (indeks -2) hingga element terakhir
// = ['d', 'e']
// Element pada indeks 1 dan 2 akan di-overwrite dengan ['d', 'e']
data = ['a', 'b', 'c', 'd', 'e'];
data.copyWithin(0, -2);
console.log(data);
```

Array searching and sorting methods

- `indexOf()` dan `lastIndexOf()`
  Berbeda dengan *methods* sebelumnya (`find()` dan `findIndex()`), *methods* ini hanya akan memberikan *element index* yang pertama kali ditemukan. Serta argument yang digunakan berupa nilai *element*. Untuk `lastIndexOf()` pencarian dimulai dari *index* terakhir.

  index-of-and-last-index-of.js

  ```
  let data = ['a', 'b', 'c', 'b', 'c'];
  let result

  // Memberikan index untuk element 'c' yang pertama kali
  // ditemukan
  result = data.indexOf('c'); console.log(result);

  // Melakukan pencarian dari element terakhir dan
  // menghasilkan index untuk element 'b' yang pertama kali
  // ditemukan
  result = data.lastIndexOf('b'); console.log(result);


  // Melakukan pencarian untuk element yang tidak ada di
  // dalam array akan menghasilkan nilai -1
  result = data.indexOf('f'); console.log(result);
  ```

- `includes()`
  Digunakan untuk memeriksa apakah suatu *element* terdapat dalam suatu *array*

  includes-demo.js

  ```
  let data = ['a', 'b', 'c', 'd', 'e'];
  let result;

  result = data.includes('c'); console.log(result);

  result = data.includes(2); console.log(result);
  ```

- `sort()`
  Digunakan untuk mengurutkan *elements* pada suatu *array* berdasarkan **alphabetical order**. Jadi perlu hati-hati menggunakan *methods* ini. Untuk mengurutkan angka maka perlu mendefinisikan fungsi *sorting* tersendiri.

  sort-demo.js

  ```
  let arr = ['banana', 'cherry', 'apple'];
  arr.sort(); console.log(arr);
  ```

```
let numericArr;

// Secara default .sort mengurutkan angka berdasarkan alphabet
// Semua angka akan diubah ke dalam string
numericArr = [33, 4, 1111, 222];
numericArr.sort();
console.log(numericArr);

// Mengurutkan angka berdasarkan nilainya make perlu mendefinisikan
// ulang fungsi sorting
numericArr = [33, 4, 1111, 222];
numericArr.sort((x, y) => { return x - y; });
console.log(numericArr)
```

- `reverse()`
  Digunakan untuk membalik urutan *element* yang ada dalam suatu *array*

  reverse-demo.js

```
let data = ['a', 'b', 'c', 'd', 'e'];
data.reverse();
console.log(data);
```

Object yang memiliki sifat seperti array

Kita dapat membentuk suatu object menjadi suatu array. Sehingga dapat dilakukan iterasi layaknya iterasi pada array.

Berikut adalah contoh penggunaan *object* yang dapat dipandang sebagai suatu *array* sehingga dapat dilakukan proses *looping*

array-like-obj.js

```
let arrLikeObj = {};     // Mendeklarasikan object tanpa properties
const objLength = 10;

// Menambahkan property ke `arrLikeObj`
for (let i = 0; i < objLength; i++) {
  arrLikeObj[i] = `item${(i+1).toString().padStart(2, '0')}`;
}
console.log(arrLikeObj);
arrLikeObj.length = objLength;



// Sekarang kita mencoba untuk melakukan iterasi ke `arrLikeObj`
for (let i = 0; i < arrLikeObj.length;i++) {
```

```
  console.log(arrLikeObj[i]);
}
```

Melihat *string* sebagai *array*

Setiap karakter di dalam *string* dapat diakses layaknya *elements* di dalam suatu *array*.

str-as-arr.js

```
let strExample = "Parallel World";
console.log(strExample.charAt(5));  // => l
console.log(strExample[5]);         // => l
```

Tugas (Exercise - 05)

> Laporan harus ditulis dan dikumpulkan dalam bentuk berkas *markdown* atau berkas berekstensi `.md`. Apabila laporan memuat lebih dari satu berkas, misal memuat berkas gambar `.png` atau `.jpg`, maka berkas disatukan menjadi berkas `.zip`.
> **PASTIKAN** berkas `md` sudah dilakukan *preview*, sehingga kode *markdown* bisa di-*preview* dengan benar.
> Format penamaan file: `NIM_NAMA.md` atau `NIM_NAMA.zip` (boleh nama lengkap atau nama panggilan).
>
> **Contoh format laporan atau jawaban (`NIM_NAMA.md`)**
> Nama: [NAMA LENGKAP]
> NIM: [NIM]
>
> 1. (Jawaban nomor 1)
> 2. (Jawaban nomor 2)

1. [30 poin] Pelajari terkait implementasi LIFO (Last-In First Out) dan FIFO (First-In First-Out) menggunakan *array*. Carilah kasus nyata penggunaan LIFO dan FIFO dan ceritakan dalam bentuk contoh penggunaan.

2. [70 poin] Dalam bidang *business analytics* dikenal suatu indikator bernama ROI (Return on Investment). Rumus sederhana untuk menghitung ROI selama setahun diberikan oleh: $$ROI = \frac{\textrm{hasil investasi} - \textrm{besar investasi}}{\textrm{besar investasi}} \times 100%$$

   Diberikan tabel besar investasi dan hasil investasi selama setahun untuk 5 buah saham, carilah saham manakah yang paling menguntungkan untuk dilakukan investasi.

Nama Saham	Besar Investasi	Hasil Investasi
BBCA	18,500	22,200
UNVR	25,200	29,500
TLKM	10,000	11,800
PGAS	5,500	7,400

Nama Saham	Besar Investasi	Hasil Investasi
ASII	28,000	32,900
WSKT	16,700	19,300

Selesaikan pencarian saham dengan hasil investasi terbaik menggunakan konsep 2D *array*, *control structure* looping, dan *array method* `.sort()`.