* Quokka extension එක start කරගන්න ctrl+shift p :- quokka start.

* In React we are using immutable arrays.
(const දාල හදන array)

Destructuring

* වේගෙ array නම් [ ] object නම් { } / "type of" return වෙන්නෙ මනළ එක object.

* In JS array can contain objects. Array can contain different data types. Including objects.

```
let students = [
  { name: "John", age: 20 },
  { name: "Jane", age: 22 },
  { name: "Alex", age: 21 }
];
```

array
object

Here array contain 3 objects.

* Destructuring :- Get some data out of objects or outputs of an array.

Destructing objects

```
const book = getBook(2);

// const title = book.title;
// const author = book.author;

const { title, author } = book;

console.log(author, title);  Stanislaw Lem The Cyberiad
```

Normal way

Destructuring object

book කියන object එකේ තියෙන title & author key ගෙන් තියෙන value retrive කරගන්නවා.

* මෙතන අලුතින් key name එකෙන් variable create වෙනවා

API වලින් data ගන්න ගේ විදිහ use වෙනවා.

Destructuring array.

```
const { title, author, pages, publicationDate, genres, hasMovieAdaptation } =
  book;

console.log(author, title, genres);  Stanislaw Lem The Cyberiad [ 'science fictio

// const primaryGenre = genres[0];
// const secondaryGenre = genres[1];

const [primaryGenre, secondaryGenre] = genres;

console.log(primaryGenre, secondaryGenre);  science fiction humor
```

} normal way

:- Destructuring array

* Here we are defining variable called primaryGenre and it assign first element's value in array.

Array Destructuring:

- Extracts values from an array and assigns them to variables.
- Uses square brackets `[]` to define the destructuring pattern.
- The order of the variables matches the order of the values in the array.
- You can use default values and rest parameters.

Object Destructuring:

- Extracts values from an object and assigns them to variables.
- Uses curly braces `{}` to define the destructuring pattern.
- The variable names match the property names in the object.
- You can use default values and aliasing.

Arrays:

- Arrays are a type of object in JavaScript that store a collection of data in an ordered sequence.
- The data in an array is stored in indexed positions, starting from 0.
- Arrays can contain elements of different data types, including numbers, strings, and objects.
- Arrays have built-in methods for performing common operations, such as adding or removing elements, sorting, and filtering.

Objects:

- Objects are a type of data structure in JavaScript that store data in key-value pairs.
- The keys in an object are strings, and each key is associated with a value.
- Objects can contain values of different data types, including numbers, strings, and other objects.
- Objects can be used to represent real-world entities, such as people, cars, or products.
- Objects can be nested, meaning that an object can contain other objects as values.

In summary, arrays are used to store collections of data in an ordered sequence, while objects are used to store data in key-value pairs. Arrays are useful when you need to work with a collection of data that needs to be ordered, while objects are useful when you need to represent real-world entities or when you need to store data in a non-sequential manner.

Rest operator.

```
const [primaryGenre, secondaryGenre, ...otherGenres] = genres;

console.log(primaryGenre, secondaryGenre, otherGenres);  sciene
```

Here Create array name as otherGenres and it contains all the elements except primary-Genre & secondaryGenre (rest elements of genres array.

...<name> :- මෙවුට දැම්ම error එකක් එනවා. meaningless ගේ rest of element කියල එක්සේ element එකක් තියෙන එක.

Spred operator. for array

```
const newGenres = [genres, "epic fantasy"];
newGenres;  [ [ 'science fiction', 'humor', 'speculative fiction', 'short stories', 'fantasy' ], 'epic fantasy' ]
```

අපි මෙතන expect කරන්නෙ exsisting එකේ අන්තිමට node එකක් විදියට attach වෙයි කියල. but වෙලා තියෙන්නෙ new element එකක් විදියට add වෙලා.

* අපිට මේක තනි array එකක් විදියට (හරියටම අන්තිමටම එන විදියට) spred operator throw කරගන්න ඕවගේ.

```
const newGenres = ["epic fantasy", ...genres];
newGenres;  [ 'epic fantasy', 'science fiction', 'humor', 'speculative fiction', 'short stories', 'fantasy' ]
```

Spred operator for object.

```
const updatedBook = { book, moviePublicationDate: "2001-12-19" };
```

මුල්ම එන existing book object එකට key value pare එකක් විදියට moviepublicationDate : "2001-12-19" add කරනවා. but මෙහෙම කරාම වෙන්නෙ book කියන object එකයි අනිත් key value pare එකයි property 2 ක් වෙනවා.

```
const updatedBook = { ...book, moviePublicationDate: "2001-12-19", pages: 1210 };
```

මෙහෙම කරාම නේ. මොනාද book object එක ඇතුලෙම element key value pair විදියට වියනවා ∴ we can overide values

```
const updatedBook = {
  ...book,
  // Adding a new property
  moviePublicationDate: "2001-12-19",

  // Overwriting an existing property
  pages: 1210,
};
```

* React වල object, states update කරන්න මේ technique එක use කරනවා.

Template Literals.

වචන සංඛ්‍යාව 24

```
const summary= title + " is a book "+(2+4);
summary   The Lord of the Rings is a book 6
```

```
const summary = `${title} is a book ${2+4}`;
summary;  The Lord of the Rings is a book 6
```

back ticks

within the {} you can do any JS expression. (calling functions)

```
const summary = `${title}, a ${pages}-page long book, was written by ${author} and published in $
{publicationDate}`;
summary;  The Lord of the Rings, a 1216-page long book, was written by J. R. R. Tolkien and published in 1954-07-29
```

```
const summary = `${title}, a ${pages}-page long book, was written by ${author} and published in ${
  publicationDate.split("-")[0]
}`;
summary;  The Lord of the Rings, a 1216-page long book, was written by J. R. R. Tolkien and published in 1954
```

Ternaries operations (Alternative for if else)

```
const pagesRange = pages > 1000 ? "over a thousand" : "less than 1000";
```

condition is true

condition is false

* Ternaries always return a value.

```
const summary = `${title}, a ${pages}-page long book, was written by ${author} and published in ${
  publicationDate.split("-")[0]
}. The book has ${hasMovieAdaptation ? "" : "not"} been adapted as a movie`;
```

value of const summary :-

```
The Cyberiad, a 295-page long book, was written by Stanislaw Lem and published in 1965. The book has not been adapted as a movie
```

Arrow Functions

conventional function :-

assigning parameters which are passing when calling the function

```
function getYear(str) {
  return str.split("-")[0];
}
console.log(getYear(publicationDate));
```

Arrow function :-

Input params

මේ කෑල්ල තමයි arrow function එකා.

```
const getYear = (str) => str.split("-")[0];

console.log(getYear(publicationDate));  1965
```

* මෙහෙදී return කරන්නෙ => right side තියෙන එක. return කියන key word එක use කරන්නෙ නෑ.

* Arrow function එකෙන් return කරන value එක getyear එකට assign කරගන්නවා.

* මේ විදියට arrow function ලියන්න ගුළුවන් single line function වලයි.

* Multiple line arrow functions :- (Declaration block)

```
const getYear = (str) => {
  return str.split("-")[0];
};
```

* Here using return keyword is mandotery. { } use කරනවනම් return යින. Declaration block

* Single line arrow function use වෙන්නෙ call back function වලට.

Short Circutting works with logical operators.

Short Circutting :- Operator return first value without looking second value.

```
console.log(true && "Some string");  Some string
console.log(false && "Some string");  false
```

Short circutting

falsy value in js :- 0, '', null, undefind

Truthy values is non of falsy value.

```
// falsy: 0, '', null, undefined
console.log("jonas" && "Some string");  Some string
console.log(0 && "Some string");  0
```

AND operator

short circutting

```
console.log(true || "Some string");  true
console.log(false || "Some string");  Some string
```

```
const count = book.reviews.librarything.reviewsCount ?? "no data";
count;  0
```

nullish coalescing operator.

* Only return 2nd value when 1st value is null or undifiend.
* 0 or emptystring it returns 0 or empty string

Optional Chaining Operator

```
function getTotalReviewCount(book) {
  const goodreads = book.reviews.goodreads.reviewsCount;
  const librarything = book.reviews.librarything.reviewsCount;  Cannot read properties of undefined (
  return goodreads + librarything;
}

console.log(getTotalReviewCount(book));  Cannot read properties of undefined (reading 'reviewsCount')
```

මේ තොතට librarything reviews defined කරල නෑ. මේ වින error prevent කරගන්න තමයි optional chaining operator එක use කරන්නෙ.

```
function getTotalReviewCount(book) {
  const goodreads = book.reviews.goodreads.reviewsCount;
  const librarything = book.reviews.librarything?.reviewsCount;
  return goodreads + librarything;
}

console.log(getTotalReviewCount(book));  NaN
```

optional chainging operator.

මේ chain එක undefind නම් reviewsCount එක read කරන්නෙ නෑ. කෙලින්ම librarything එකට undefinded assign කරනවා

```
function getTotalReviewCount(book) {
  const goodreads = book.reviews.goodreads.reviewsCount;
  const librarything = book.reviews.librarything?.reviewsCount;
  librarything;  undefined
  return goodreads + librarything;
}

console.log(getTotalReviewCount(book));  NaN
```

Number + undefined

nullish coalescing operator එක දාල

```
function getTotalReviewCount(book) {
  const goodreads = book.reviews.goodreads.reviewsCount;
  const librarything = book.reviews.librarything?.reviewsCount ?? 0;
  librarything;  0
  return goodreads + librarything;
}

console.log(getTotalReviewCount(book));  49701
```

The Arrary map

```
const x = [1, 2, 3, 4, 5].map((el) => el * 2);
console.log(x);  [ 2, 4, 6, 8, 10 ]
```

map එකෙන් වෙන්වෙන array එකේ තියෙන element වලින් වෙනස් එක හැරෙන් arrow function එකේ input parameter වලට pass කරල return කරන value එක දාල const x array එක create කරනවා

```
const titles = books.map((book) => book.title);
titles;  [ 'The Lord of the Rings', 'The Cyberiad', 'Dune', 'Harry Potter and the Philosopher\'s Stone',
```

```
const essentialData = books.map((book) => {
  return {
    title: book.title,
    author: book.author,
  };
});
```

without having return

```
const essentialData = books.map((book) => ({
  title: book.title,
  author: book.author,
}));
```

Filter Method

* Filter elements of array bassed on condition.

```
const longBooks = books.filter((book) => book.pages > 500);
```

If this condition is true then the element is insert in to new array.

```
const longBooks = books
  .filter((book) => book.pages > 500)
  .filter((book) => book.hasMovieAdaptation);
```

has boolean value

this method is returning an array. so next filter method is apply for this returning array.

```
const adventureBooks = books
  .filter((books) => books.genres.includes("adventure"))
  .map((book) => book.title);
adventureBooks;  [ 'The Lord of the Rings', 'Dune', 'Harry Potter and the Philosopher\'s Stone' ]
```

checking weather books.genres array contain "adventure" element.

Reduce Method

```
const pagesAllBooks = books.reduce((acc, book) => acc + book.pages, 0);
```

```
pagesAllBooks;  3227
```
← output.

starting value of acc.

acc is stand for accumilator. But we can use any variable name. but we must mention starting value.

මෙතන වෙන්නෙ array එකේ තියෙන book.pages වල sum එක return කරනවා.

first itteration එකේදී acc = 0 book.pages = 100
2nd itteration එකේදී acc = 100 book.pages = 50
3rd itteration එකේදී acc = 150 book.pages = 1000
⋮ ⋮

Finally acc එකේ final value එකට return කරනවා

* acc value can be array, object, ...

```
const pagesAllBooks = books.reduce((sum, book) => sum + book.pages, 0);
```

acc වෙනුවට sum දැම්

Sort method

```
const arr = [3, 7, 1, 9, 6];
const sorted = arr.sort((a, b) => a - b);
sorted;  [ 1, 3, 6, 7, 9 ]
```

accending

```
const sorted = arr.sort((a, b) => b - a);
sorted;  [ 9, 7, 6, 3, 1 ]
```

deccending

```
const arr = [3, 7, 1, 9, 6];
const sorted = arr.sort((a, b) => b - a);
sorted;  [ 9, 7, 6, 3, 1 ]
arr;  [ 9, 7, 6, 3, 1 ]
```

Here original array is also sorted.

To prevent this mutating data we create a copy of array and sort this array.

```
const arr = [3, 7, 1, 9, 6];
const sorted = arr.slice().sort((a, b) => a - b);
sorted;  [ 1, 3, 6, 7, 9 ]
arr;  [ 3, 7, 1, 9, 6 ]
```

Sorted object

```
const sortedByPages = books.slice().sort((a, b) => b.pages - a.pages);
```

Add, update, delete elements of array without changing original array.

```
// 1) Add book object to array
const newBook = {
  id: 6,
  title: "Harry Potter and the Chamber of Secrets",
  author: "J. K. Rowling",
};
const booksAfterAdd = [...books, newBook];
```

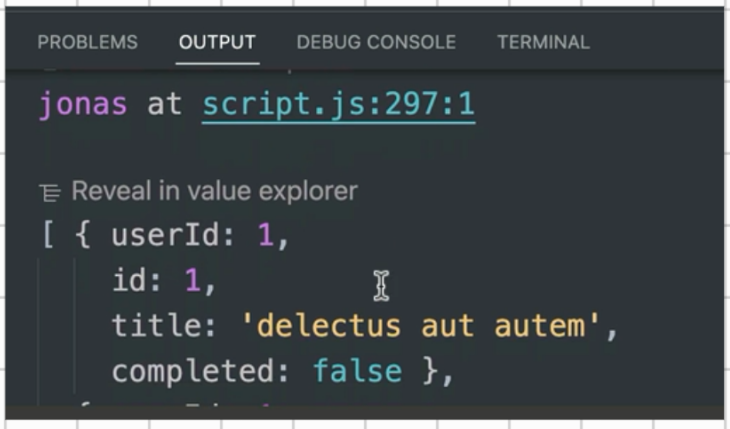
```
// 2) Delete book object from array
const booksAfterDelete = booksAfterAdd.filter((book) => book.id !== 3);
```

```
// 3) Update book object in the array
const booksAfterUpdate = booksAfterDelete.map((book) =>
  book.id === 1 ? { ...book, pages: 1210 } : book
);
```

Asynchronous js promises.

```
fetch("https://jsonplaceholder.typicode.com/todos")
  .then((res) => res.json())
  .then((data) => console.log(data));

console.log("jonas");  jonas
```



* Synchronous behaviour of js. It isn't waiting untill data returning from API. It just keep a reference then exsicute next line.

මේ විදිහට ලියන්නත් පුළුවන්

```
async function getTodos() {
  const res = await fetch("https://jsonplaceholder.typicode.com/todos");
  const data = await res.json();
  console.log(data);  ... am', completed: true }, { userId: 10, id: 200, title:
}

getTodos();
```