

JAVASCRIPT クイックリファレンス

ES6+, DOM, イベント, Fetch API, async/await

基礎

変数

```
let name = "Alice"; // reassignable
const PI = 3.14;    // constant
var old = "avoid"; // function-scoped (legacy)
```

データ型

string 文字列: "hello" または `hello`
number 整数または浮動小数点: `42`, `3.14`
boolean true / false
null 意図的な空値
undefined 宣言済みだが未代入
object キーと値のペア: {a:1}
array 順序付きリスト: [1,2,3]
symbol 一意な識別子

型チェック & 変換

```
typeof "hello" // "string"
typeof 42      // "number"
Number("42")  // 42
String(100)   // "100"
parseInt("3.9") // 3
parseFloat("3.14") // 3.14
```

文字列

テンプレートリテラル

```
const name = "Alice";
`Hello, ${name}!`; // Hello, Alice!
`Total: ${2 + 3}` // Total: 5
`Multi
line string`
```

文字列メソッド

s.length 文字数
s.toUpperCase() 大文字のコピー
s.toLowerCase() 小文字のコピー
s.trim() 前後の空白を削除
s.split(",") 配列に分割
s.includes("x") 含まれるかチェック → bool
s.indexOf("x") 最初のインデックス (なければ -1)
s.slice(1, 4) インデックスで部分文字列
s.replace(a, b) 最初のマッチを置換
s.replaceAll(a, b) 全マッチを置換
s.startsWith(x) 前方一致チェック → bool
s.endsWith(x) 後方一致チェック → bool
s.padStart(n, c) 長さ n になるよう先頭にパディング

配列

作成 & アクセス

```
const fruits = ["apple", "banana", "cherry"];
fruits[0] // "apple"
fruits.length // 3
fruits.at(-1) // "cherry"
```

破壊的メソッド

arr.push(x) 末尾に追加
arr.pop() 末尾を削除して返す
arr.unshift(x) 先頭に追加
arr.shift() 先頭を削除して返す
arr.splice(i, n) インデックス i から n 個を削除
arr.sort() インプレースでソート (辞書順)
arr.reverse() インプレースで逆順

非破壊的メソッド

arr.map(fn) 各要素を変換
arr.filter(fn) fn が true の要素を残す
arr.reduce(fn, init) 最初の値に集約
arr.find(fn) 最初のマッチまたは undefined
arr.findIndex(fn) 最初のマッチのインデックス (-1)
arr.includes(x) 含まれるかチェック → bool
arr.slice(a, b) 一部の浅いコピー
arr.join(" ") 文字列に結合
arr.forEach(fn) イテレーション (戻り値なし)
[...a, ...b] 配列の連結 (スプレッド)

オブジェクト

作成 & アクセス

```
const user = { name: "Alice", age: 20 };
user.name // "Alice"
user["age"] // 20
user.age = 3.85; // add/update
```

分割代入 & スプレッド

```
const { name, age } = user;
const copy = { ...user, age: 21 };
```

オブジェクトメソッド

Object.keys(o) キーの配列
Object.values(o) 値の配列
Object.entries(o) [キー, 値] ペアの配列
Object.assign(t, s) プロパティを s へとコピー
[k in obj] キーの存在チェック → bool
delete obj.k プロパティを削除
Object.freeze(o) 不変にする (浅いコピー)

制御フロー

if / else if / else

```
if (score >= 90) {
  grade = "A";
} else if (score >= 80) {
  grade = "B";
} else {
  grade = "C";
}
```

三項演算子 & Null 合体

```
const status = score >= 60 ? "pass" : "fail";
const name = user.name ?? "Anonymous";
```

switch

```
switch (color) {
  case "red": stop(); break;
  case "green": go(); break;
  default: wait();
}
```

ループ

for / for...of / for...in

```
for (let i = 0; i < 5; i++) { }
for (const item of ["a", "b"]) { } // arrays
for (const key in obj) { } // object keys
```

while / do...while

```
while (count < 10) { count++; }
```

```
do { count++; } while (count < 10);
```

break & continue

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break; // stop loop
  if (i % 2 === 0) continue; // skip
}
```

関数

関数宣言 & アロー関数

```
function greet(name) {
  return `Hello, ${name}!`;
}
const greet = (name) => `Hello, ${name}!`;
const square = x => x * x; // single param
```

デフォルト引数 & rest

```
function greet(name = "World") { }
function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
```

コールバック

```
[1, 2, 3].map(x => x * 2); // [2, 4, 6]
[1, 2, 3].filter(x => x > 1); // [2, 3]
setTimeout(() => console.log("done"), 1000);
```

クラス

```
class Dog {
  constructor(name, breed) {
    this.name = name;
    this.breed = breed;
  }
  bark() { return `${this.name} says Woof!`; }
}
class Puppy extends Dog {
  constructor(name, breed, toy) {
    super(name, breed);
    this.toy = toy;
  }
}
```

エラーハンドリング

```
try {
  JSON.parse("bad json");
} catch (err) {
  console.error(err.message);
} finally {
  console.log("always runs");
}
```

エラーのスロー

```
throw new Error("Something went wrong");
```

DOM

要素の取得

```
document.querySelector(".cls") // first match
document.querySelectorAll("li") // all matches
document.getElementById("main")
```

要素の変更

```
e1.textContent = "new text";
e1.innerHTML = "<b>bold</b>";
e1.style.color = "red";
e1.classList.add("active");
e1.classList.toggle("hidden");
e1.setAttribute("data-id", "42");
```

イベント

```
btn.addEventListener("click", (e) => {
  console.log(e.target);
});
```

要素の作成

```
const li = document.createElement("li");
li.textContent = "New item";
ul.appendChild(li);
ul.removeChild(li);
el.remove(); // remove element
```

Fetch API

GET リクエスト

```
fetch("https://api.example.com/data")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

POST リクエスト

```
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ key: "value" }),
});
```

Async / Await

```
async function loadData() {
  try {
    const res = await fetch(url);
    const data = await res.json();
    return data;
  } catch (err) {
    console.error(err);
  }
}
```

並列リクエスト

```
const [users, posts] = await Promise.all([
  fetch("/users").then(r => r.json()),
  fetch("/posts").then(r => r.json()),
]);
```

モジュール

名前付きエクスポート

```
// math.js
export const PI = 3.14;
export function add(a, b) { return a + b; }
```

```
// main.js
import { PI, add } from "./math.js";
```

デフォルトエクスポート

```
// logger.js
export default function log(msg) { }
```

```
// main.js
import log from "./logger.js";
```