JavaScript

A JavaScript cheat sheet with the most important concepts, functions, methods, and more. A complete quick reference for beginners.

Getting Started

Introduction

JavaScript is a lightweight, interpreted programming language.

- JSON cheatsheet (cheatsheets.zip)
- Regex in JavaScript (cheatsheets.zip)

Console

```
// => Hello world!
console.log("Hello world!");

// => Hello CheatSheets.zip
console.warn("hello %s", "CheatSheets.zip");

// Prints error message to stderr
console.error(new Error("Oops!"));
```

Numbers

```
let amount = 6;
let price = 4.99;
```

Variables

```
let x = null;
let name = "Tammy";
```

```
const found = false;

// => Tammy, false, null
console.log(name, found, x);

var a;
console.log(a); // => undefined
```

Strings

```
let single = "Wheres my bandit hat?";
let double = "Wheres my bandit hat?";

// => 21
console.log(single.length);
```

Arithmetic Operators

```
5 + 5 = 10      // Addition
10 - 5 = 5      // Subtraction
5 * 10 = 50     // Multiplication
10 / 5 = 2      // Division
10 % 5 = 0      // Modulo
```

Comments

```
// This line will denote a comment

/*
The below configuration must be
changed before deployment.
*/
```

Assignment Operators

```
let number = 100;

// Both statements will add 10
number = number + 10;
number += 10;

console.log(number);
// => 120
```

String Interpolation

```
let age = 7;

// String concatenation
"Tommy is " + age + " years old.";

// String interpolation
`Tommy is ${age} years old.`;
```

let Keyword

```
let count;
console.log(count); // => undefined
count = 10;
console.log(count); // => 10
```

const Keyword

```
const numberOfColumns = 4;

// TypeError: Assignment to constant...
numberOfColumns = 8;
```

JavaScript Conditionals

if Statement

```
const isMailSent = true;

if (isMailSent) {
  console.log("Mail sent to recipient");
}
```

Ternary Operator

```
var x = 1;

// => true
result = x == 1 ? true : false;
```

Operators

```
true || false; // true
10 > 5 || 10 > 20; // true
false || false; // false
10 > 100 || 10 > 20; // false
```

Logical Operator &&

```
true && true; // true
1 > 2 && 2 > 1; // false
true && false; // false
4 === 4 && 3 > 1; // true
```

Comparison Operators

```
1 > 3; // false
3 > 1; // true
250 >= 250; // true
1 === 1; // true
1 === 2; // false
1 === "1"; // false
```

Logical Operator !

```
let lateToWork = true;
let oppositeValue = !lateToWork;

// => false
console.log(oppositeValue);
```

Nullish coalescing operator ??

```
null ?? "I win"; //  'I win'
undefined ?? "Me too"; //  'Me too'

false ?? "I lose"; //  false
0 ?? "I lose again"; //  0
"" ?? "Damn it"; //  ''
```

else if

```
const size = 10;

if (size > 100) {
  console.log("Big");
```

```
} else if (size > 20) {
  console.log("Medium");
} else if (size > 4) {
  console.log("Small");
} else {
  console.log("Tiny");
}
// Print: Small
```

switch Statement

```
const food = "salad";

switch (food) {
  case "oyster":
    console.log("The taste of the sea");
    break;
  case "pizza":
    console.log("A delicious pie");
    break;
  default:
    console.log("Enjoy your meal");
}
```

== vs ===

```
0 == false; // true
0 === false; // false, different type
1 == "1"; // true,  automatic type conversion
1 === "1"; // false, different type
null == undefined; // true
null === undefined; // false
"0" == false; // true
"0" === false; // false
```

The == just check the value, === check both the value and the type.

JavaScript Functions

Functions

```
// Defining the function:
function sum(num1, num2) {
  return num1 + num2;
}

// Calling the function:
sum(3, 6); // 9
```

Anonymous Functions

```
// Named function
function rocketToMars() {
  return "BOOM!";
}

// Anonymous function
const rocketToMars = function () {
  return "BOOM!";
};
```

Arrow Functions (ES6)

With two arguments

```
const sum = (param1, param2) => {
  return param1 + param2;
};
console.log(sum(2, 5)); // => 7
```

With no arguments

```
const printHello = () => {
  console.log("hello");
};
printHello(); // => hello
```

With a single argument

```
const checkWeight = (weight) => {
  console.log(`Weight : ${weight}`);
};
checkWeight(25); // => Weight : 25
```

Concise arrow functions

```
const multiply = (a, b) => a * b;
```

```
// => 60
console.log(multiply(2, 30));
```

Arrow function available starting ES2015

return Keyword

```
// With return
function sum(num1, num2) {
  return num1 + num2;
}

// The function doesn't output the sum
function sum(num1, num2) {
  num1 + num2;
}
```

Calling Functions

```
// Defining the function
function sum(num1, num2) {
  return num1 + num2;
}

// Calling the function
sum(2, 4); // 6
```

Function Expressions

```
const dog = function () {
  return "Woof!";
};
```

Function Parameters

```
// The parameter is name
function sayHello(name) {
  return `Hello, ${name}!`;
}
```

Function Declaration

```
function add(num1, num2) {
  return num1 + num2;
```

```
}
```

JavaScript Scope

Scope

```
function myFunction() {
  var pizzaName = "Margarita";
  // Code here can use pizzaName
}

// Code here can't use pizzaName
```

Block Scoped Variables

```
const isLoggedIn = true;

if (isLoggedIn == true) {
  const statusMessage = "Logged in.";
}

// Uncaught ReferenceError...
console.log(statusMessage);
```

Global Variables

```
// Variable declared globally
const color = "blue";

function printColor() {
  console.log(color);
}

printColor(); // => blue
```

let vs var

```
for (let i = 0; i < 3; i++) {
  // This is the Max Scope for 'let'
  // i accessible ✔
}
```

```
// i not accessible ❌
```

```
for (var i = 0; i < 3; i++) {
  // i accessible ✔
}
// i accessible ✔
```

`var` is scoped to the nearest function block, and `let` is scoped to the nearest enclosing block.

Loops with closures

```
// Prints 3 thrice, not what we meant.
for (var i = 0; i < 3; i++) {
  setTimeout(_ => console.log(i), 10);
}
```

```
// Prints 0, 1 and 2, as expected.
for (let j = 0; j < 3; j++) {
  setTimeout(_ => console.log(j), 10);
}
```

The variable has its own copy using `let`, and the variable has shared copy using `var`.

JavaScript Arrays

Arrays

```
const fruits = ["apple", "orange", "banana"];

// Different data types
const data = [1, "chicken", false];
```

Property .length

```
const numbers = [1, 2, 3, 4];

numbers.length; // 4
```

Index

```
// Accessing an array element
const myArray = [100, 200, 300];

console.log(myArray[0]); // 100
console.log(myArray[1]); // 200
```

Mutable chart

	add	remove	start	end
push	✔			✔
pop		✔		✔
unshift	✔		✔	
shift		✔	✔	

Array.push()

```
// Adding a single element:
const cart = ["apple", "orange"];
cart.push("pear");

// Adding multiple elements:
const numbers = [1, 2];
numbers.push(3, 4, 5);
```

Add items to the end and returns the new array length.

Array.pop()

```
const fruits = ["apple", "orange", "banana"];

const fruit = fruits.pop(); // 'banana'
console.log(fruits); // ["apple", "orange"]
```

Remove an item from the end and returns the removed item.

Array.shift()

```
let cats = ["Bob", "Willy", "Mini"];

cats.shift(); // ['Willy', 'Mini']
```

Remove an item from the beginning and returns the removed item.

Array.unshift()

```
let cats = ["Bob"];

// => ['Willy', 'Bob']
cats.unshift("Willy");

// => ['Puff', 'George', 'Willy', 'Bob']
cats.unshift("Puff", "George");
```

Add items to the beginning and returns the new array length.

Array.concat()

```
const numbers = [3, 2, 1];
const newFirstNumber = 4;

// => [ 4, 3, 2, 1 ]
[newFirstNumber].concat(numbers);

// => [ 3, 2, 1, 4 ]
numbers.concat(newFirstNumber);
```

If you want to avoid mutating your original array, you can use concat.

JavaScipt Set

Create Set

```
// Empty Set Object
const emptySet = new Set();

// Set Object with values
const setObj = new Set([1, true, "hi"]);
```

Add

```
const emptySet = new Set();
```

```
// add values
emptySet.add("a"); // 'a'
emptySet.add(1); // 'a', 1
emptySet.add(true); // 'a', 1, true
emptySet.add("a"); // 'a', 1, true
```

Delete

```
const emptySet = new Set([1, true, "a"]);

// delete values
emptySet.delete("a"); // 1, true
emptySet.delete(true); // 1
emptySet.delete(1); //
```

Has

```
const setObj = new Set([1, true, "a"]);

// returns true or false
setObj.has("a"); // true
setObj.has(1); // true
setObj.has(false); // false
```

Clear

```
const setObj = new Set([1, true, "a"]);

// clears the set
console.log(setObj); // 1, true, 'a'
setObj.clear(); //
```

Size

```
const setObj = new Set([1, true, "a"]);

consoloe.log(setObj.size); // 3
```

ForEach

```
const setObj = new Set([1, true, "a"]);

setObj.forEach(function (value) {
  console.log(value);
```

```
});

// 1
// true
// 'a'
```

JavaScript Loops

While Loop

```
while (condition) {
  // code block to be executed
}

let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

Reverse Loop

```
const fruits = ["apple", "orange", "banana"];

for (let i = fruits.length - 1; i >= 0; i--) {
  console.log(`${i}. ${fruits[i]}`);
}

// => 2. banana
// => 1. orange
// => 0. apple
```

Do...While Statement

```
x = 0;
i = 0;

do {
  x = x + i;
  console.log(x);
  i++;
```

```
} while (i < 5);
// => 0 1 3 6 10
```

For Loop

```
for (let i = 0; i < 4; i += 1) {
  console.log(i);
}

// => 0, 1, 2, 3
```

Looping Through Arrays

```
for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}

// => Every item in the array
```

Break

```
for (let i = 0; i < 99; i += 1) {
  if (i > 5) {
    break;
  }
  console.log(i);
}
// => 0 1 2 3 4 5
```

Continue

```
for (i = 0; i < 10; i++) {
  if (i === 3) {
    continue;
  }
  text += "The number is " + i + "<br>";
}
```

Nested

```
for (let i = 0; i < 2; i += 1) {
  for (let j = 0; j < 3; j += 1) {
    console.log(`${i}-${j}`);
  }
```

```
}
```

for...in loop

```
const fruits = ["apple", "orange", "banana"];

for (let index in fruits) {
  console.log(index);
}
// => 0
// => 1
// => 2
```

for...of loop

```
const fruits = ["apple", "orange", "banana"];

for (let fruit of fruits) {
  console.log(fruit);
}
// => apple
// => orange
// => banana
```

JavaScript Iterators

Functions Assigned to Variables

```
let plusFive = (number) => {
  return number + 5;
};
// f is assigned the value of plusFive
let f = plusFive;

plusFive(3); // 8
// Since f has a function value, it can be invoked.
f(9); // 14
```

Callback Functions

```
const isEven = (n) => {
```

```
  return n % 2 == 0;
};

let printMsg = (evenFunc, num) => {
  const isNumEven = evenFunc(num);
  console.log(`${num} is an even number: ${isNumEven}.`);
};

// Pass in isEven as the callback function
printMsg(isEven, 4);
// => The number 4 is an even number: True.
```

Array.reduce()

```
const numbers = [1, 2, 3, 4];

const sum = numbers.reduce((accumulator, curVal) => {
  return accumulator + curVal;
});

console.log(sum); // 10
```

Array.map()

```
const members = ["Taylor", "Donald", "Don", "Natasha", "Bobby"];

const announcements = members.map((member) => {
  return member + " joined the contest.";
});

console.log(announcements);
```

Array.forEach()

```
const numbers = [28, 77, 45, 99, 27];

numbers.forEach((number) => {
  console.log(number);
});
```

Array.filter()

```
const randomNumbers = [4, 11, 42, 14, 39];
const filteredArray = randomNumbers.filter((n) => {
  return n > 5;
```

```
});
```

JavaScript Objects

Accessing Properties

```
const apple = {
  color: "Green",
  price: { bulk: "$3/kg", smallQty: "$4/kg" },
};
console.log(apple.color); // => Green
console.log(apple.price.bulk); // => $3/kg
```

Naming Properties

```
// Example of invalid key names
const trainSchedule = {
  // Invalid because of the space between words.
  platform num: 10,
  // Expressions cannot be keys.
  40 - 10 + 2: 30,
  // A + sign is invalid unless it is enclosed in quotations.
  +compartment: 'C'
}
```

Non-existent properties

```
const classElection = {
  date: "January 12",
};

console.log(classElection.place); // undefined
```

Mutable

```
const student = {
  name: "Sheldon",
  score: 100,
  grade: "A",
};
```

```
console.log(student);
// { name: 'Sheldon', score: 100, grade: 'A' }

delete student.score;
student.grade = "F";
console.log(student);
// { name: 'Sheldon', grade: 'F' }

student = {};
// TypeError: Assignment to constant variable.
```

Assignment shorthand syntax

```
const person = {
  name: "Tom",
  age: "22",
};
const { name, age } = person;
console.log(name); // 'Tom'
console.log(age); // '22'
```

Delete operator

```
const person = {
  firstName: "Matilda",
  age: 27,
  hobby: "knitting",
  goal: "learning JavaScript",
};

delete person.hobby; // or delete person[hobby];

console.log(person);
/*
{
  firstName: "Matilda"
  age: 27
  goal: "learning JavaScript"
}
*/
```

Objects as arguments

```
const origNum = 8;
```

```
const origObj = { color: "blue" };

const changeItUp = (num, obj) => {
  num = 7;
  obj.color = "red";
};

changeItUp(origNum, origObj);

// Will output 8 since integers are passed by value.
console.log(origNum);

// Will output 'red' since objects are passed
// by reference and are therefore mutable.
console.log(origObj.color);
```

Shorthand object creation

```
const activity = "Surfing";
const beach = { activity };
console.log(beach); // { activity: 'Surfing' }
```

this Keyword

```
const cat = {
  name: "Pipey",
  age: 8,
  whatName() {
    return this.name;
  },
};
console.log(cat.whatName()); // => Pipey
```

Factory functions

```
// A factory function that accepts 'name',
// 'age', and 'breed' parameters to return
// a customized dog object.
const dogFactory = (name, age, breed) => {
  return {
    name: name,
    age: age,
    breed: breed,
    bark() {
      console.log("Woof!");
```

```
    },
  };
};
```

Object methods

```
const engine = {
  // method shorthand, with one argument
  start(adverb) {
    console.log(`The engine starts up ${adverb}...`);
  },
  // anonymous arrow function expression with no arguments
  sputter: () => {
    console.log("The engine sputters...");
  },
};

engine.start("noisily");
engine.sputter();
```

Getters and setters

```
const myCat = {
  _name: "Dottie",
  get name() {
    return this._name;
  },
  set name(newName) {
    this._name = newName;
  },
};

// Reference invokes the getter
console.log(myCat.name);

// Assignment invokes the setter
myCat.name = "Yankee";
```

JavaScript Classes

Static Methods

```
class Dog {
  constructor(name) {
    this._name = name;
  }

  introduce() {
    console.log("This is " + this._name + " !");
  }

  // A static method
  static bark() {
    console.log("Woof!");
  }
}

const myDog = new Dog("Buster");
myDog.introduce();

// Calling the static method
Dog.bark();
```

Class

```
class Song {
  constructor() {
    this.title;
    this.author;
  }

  play() {
    console.log("Song playing!");
  }
}

const mySong = new Song();
mySong.play();
```

Class Constructor

```
class Song {
  constructor(title, artist) {
    this.title = title;
    this.artist = artist;
  }
```

```
}

const mySong = new Song("Bohemian Rhapsody", "Queen");
console.log(mySong.title);
```

Class Methods

```
class Song {
  play() {
    console.log("Playing!");
  }

  stop() {
    console.log("Stopping!");
  }
}
```

extends

```
// Parent class
class Media {
  constructor(info) {
    this.publishDate = info.publishDate;
    this.name = info.name;
  }
}

// Child class
class Song extends Media {
  constructor(songData) {
    super(songData);
    this.artist = songData.artist;
  }
}

const mySong = new Song({
  artist: "Queen",
  name: "Bohemian Rhapsody",
  publishDate: 1975,
});
```

JavaScript Modules

Export

```
// myMath.js

// Default export
export default function add(x, y) {
  return x + y;
}

// Normal export
export function subtract(x, y) {
  return x - y;
}

// Multiple exports
function multiply(x, y) {
  return x * y;
}
function duplicate(x) {
  return x * 2;
}
export { multiply, duplicate };
```

Import

```
// main.js
import add, { subtract, multiply, duplicate } from './myMath.js';

console.log(add(6, 2)); // 8
console.log(subtract(6, 2)) // 4
console.log(multiply(6, 2)); // 12
console.log(duplicate(5)) // 10

// index.html
<script type="module" src="main.js"></script>
```

Export Module

```
// myMath.js

function add(x, y) {
  return x + y;
}
```

```
function subtract(x, y) {
  return x - y;
}
function multiply(x, y) {
  return x * y;
}
function duplicate(x) {
  return x * 2;
}

// Multiple exports in node.js
module.exports = {
  add,
  subtract,
  multiply,
  duplicate,
};
```

Require Module

```
// main.js
const myMath = require("./myMath.js");

console.log(myMath.add(6, 2)); // 8
console.log(myMath.subtract(6, 2)); // 4
console.log(myMath.multiply(6, 2)); // 12
console.log(myMath.duplicate(5)); // 10
```

JavaScript Promises

Promise states

```
const promise = new Promise((resolve, reject) => {
  const res = true;
  // An asynchronous operation.
  if (res) {
    resolve("Resolved!");
  } else {
    reject(Error("Error"));
  }
});
```

```
promise.then(
  (res) => console.log(res),
  (err) => console.error(err),
);
```

Executor function

```
const executorFn = (resolve, reject) => {
  resolve("Resolved!");
};

const promise = new Promise(executorFn);
```

setTimeout()

```
const loginAlert = () => {
  console.log("Login");
};

setTimeout(loginAlert, 6000);
```

.then() method

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Result");
  }, 200);
});

promise.then(
  (res) => {
    console.log(res);
  },
  (err) => {
    console.error(err);
  },
);
```

Promise.catch()

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(Error("Promise Rejected Unconditionally."));
  }, 1000);
```

```
});

promise.then((res) => {
  console.log(value);
});

promise.catch((err) => {
  console.error(err);
});
```

Promise.all()

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(3);
  }, 300);
});
const promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(2);
  }, 200);
});

Promise.all([promise1, promise2]).then((res) => {
  console.log(res[0]);
  console.log(res[1]);
});
```

Avoiding nested Promise and .then()

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("*");
  }, 1000);
});

const twoStars = (star) => {
  return star + star;
};

const oneDot = (star) => {
  return star + ".";
};

const print = (val) => {
```

```
  console.log(val);
};

// Chaining them all together
promise.then(twoStars).then(oneDot).then(print);
```

Creating

```
const executorFn = (resolve, reject) => {
  console.log("The executor function of the promise!");
};

const promise = new Promise(executorFn);
```

Chaining multiple .then()

```
const promise = new Promise((resolve) => setTimeout(() => resolve("dAlan"), 100));

promise
  .then((res) => {
    return res === "Alan" ? Promise.resolve("Hey Alan!") : Promise.reject("Who are
  })
  .then(
    (res) => {
      console.log(res);
    },
    (err) => {
      console.error(err);
    },
  );
```

Fake http Request with Promise

```
const mock = (success, timeout = 1000) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve({ status: 200, data: {} });
      } else {
        reject({ message: "Error" });
      }
    }, timeout);
  });
};
```

```
const someEvent = async () => {
  try {
    await mock(true, 1000);
  } catch (e) {
    console.log(e.message);
  }
};
```

JavaScript Async-Await

Asynchronous

```
function helloWorld() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Hello World!");
    }, 2000);
  });
}

const msg = async function () {
  //Async Function Expression
  const msg = await helloWorld();
  console.log("Message:", msg);
};

const msg1 = async () => {
  //Async Arrow Function
  const msg = await helloWorld();
  console.log("Message:", msg);
};

msg(); // Message: Hello World! <-- after 2 seconds
msg1(); // Message: Hello World! <-- after 2 seconds
```

Resolving Promises

```
let pro1 = Promise.resolve(5);
let pro2 = 44;
let pro3 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 100, "foo");
```

```
});

Promise.all([pro1, pro2, pro3]).then(function (values) {
  console.log(values);
});
// expected => Array [5, 44, "foo"]
```

Async Await Promises

```
function helloWorld() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Hello World!");
    }, 2000);
  });
}

async function msg() {
  const msg = await helloWorld();
  console.log("Message:", msg);
}

msg(); // Message: Hello World! <-- after 2 seconds
```

Error Handling

```
let json = '{ "age": 30 }'; // incomplete data

try {
  let user = JSON.parse(json); // <-- no errors
  console.log(user.name); // no name!
} catch (e) {
  console.error("Invalid JSON data!");
}
```

Aysnc await operator

```
function helloWorld() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Hello World!");
    }, 2000);
  });
}
```

```
async function msg() {
  const msg = await helloWorld();
  console.log("Message:", msg);
}

msg(); // Message: Hello World! <-- after 2 seconds
```

JavaScript Requests

JSON

```
const jsonObj = {
  "name": "Rick",
  "id": "11A",
  "level": 4
};
```

Also see: JSON cheatsheet

XMLHttpRequest

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "mysite.com/getjson");
```

`XMLHttpRequest` is a browser-level API that enables the client to script data transfers via JavaScript, NOT part of the JavaScript language.

GET

```
const req = new XMLHttpRequest();
req.responseType = "json";
req.open("GET", "/getdata?id=65");
req.onload = () => {
  console.log(xhr.response);
};

req.send();
```

POST

```
const data = {
```

```
  fish: "Salmon",
  weight: "1.5 KG",
  units: 5,
};
const xhr = new XMLHttpRequest();
xhr.open("POST", "/inventory/add");
xhr.responseType = "json";
xhr.send(JSON.stringify(data));

xhr.onload = () => {
  console.log(xhr.response);
};
```

fetch api

```
fetch(url, {
    method: 'POST',
    headers: {
      'Content-type': 'application/json',
      'apikey': apiKey
    },
    body: data
  }).then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Request failed!');
  }, networkError => {
    console.log(networkError.message)
  })
}
```

JSON Formatted

```
fetch("url-that-returns-JSON")
  .then((response) => response.json())
  .then((jsonResponse) => {
    console.log(jsonResponse);
  });
```

promise url parameter fetch api

```
fetch('url')
.then(
  response  => {
```

```
    console.log(response);
  },
 rejection => {
    console.error(rejection.message);
);
```

Fetch API Function

```
fetch("https://api-xxx.com/endpoint", {
  method: "POST",
  body: JSON.stringify({ id: "200" }),
})
  .then(
    (response) => {
      if (response.ok) {
        return response.json();
      }
      throw new Error("Request failed!");
    },
    (networkError) => {
      console.log(networkError.message);
    },
  )
  .then((jsonResponse) => {
    console.log(jsonResponse);
  });
```

async await syntax

```
const getSuggestions = async () => {
  const wordQuery = inputField.value;
  const endpoint = `${url}${queryParams}${wordQuery}`;
  try {
    const response = await fetch(endpoint, { cache: "no-cache" });
    if (response.ok) {
      const jsonResponse = await response.json();
    }
  } catch (error) {
    console.log(error);
  }
};
```

Related Cheatsheet

- jQuery Cheatsheet
  Quick Reference
- HTML Canvas Cheatsheet
  Quick Reference
- CSS 3 Cheatsheet
  Quick Reference
- Django Cheatsheet
  Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet
  Quick Reference
- Arduino Programming Cheatsheet
  Quick Reference
- HTML Canvas Cheatsheet
  Quick Reference
- TypeScript Cheatsheet
  Quick Reference

ES6

A quick reference cheatsheet of what's new in JavaScript for ES2015, ES2016, ES2017, ES2018 and beyond

Getting Started

Block-scoped

Let

```
function fn () {
  let x = 0
  if (true) {
    let x = 1 // only inside this `if`
  }
}
```

Const

```
const a = 1;
```

`let` is the new `var`. Constants (`const`) work just like `let`, but cannot be reassigned. See: Let and const

Template Strings

Interpolation

```
const message = `Hello ${name}`;
```

Multi-line string

```
const str = `
hello
the world
`;
```

Templates and multiline strings. See: template strings

Binary and octal literals

```
let bin = 0b1010010;
let oct = 0o755;
```

See: Binary and Octal Literals

Exponential Operator

```
const byte = 2 ** 8;
```

Same as: Math.pow(2, 8)

New library additions

New string methods

```
"hello".repeat(3);
"hello".includes("ll");
"hello".startsWith("he");
"hello".padStart(8); // "hello"
"hello".padEnd(8); // "hello"
"hello".padEnd(8, "!"); // hello!!!
"\u1E9B\u0323".normalize("NFC");
```

New Number Methods

```
Number.EPSILON;
Number.isInteger(Infinity); // false
Number.isNaN("NaN"); // false
```

New Math methods

```
Math.acosh(3); // 1.762747174039086
Math.hypot(3, 4); // 5
Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2); // 2
```

New Array methods

```
//return a real array
Array.from(document.querySelectorAll("*"));
//similar to new Array(...), but without the special single-argument behavior
Array.of(1, 2, 3);
```

See: New library additions

kind

```
class Circle extends Shape {
```

Constructor

```
constructor (radius) {
  this.radius = radius
}
```

method

```
getArea () {
  return Math.PI *2 *this.radius
}
```

Call the superclass method

```
expand(n) {
  return super.expand(n) *Math.PI
}
```

Static methods

```
static createFromDiameter(diameter) {
  return new Circle(diameter /2)
}
```

Syntactic sugar for prototypes. See: classes

Private class

The javascript default field is public (`public`), if you need to indicate private, you can use (`#`)

```
class Dog {
  #name;
  constructor(name) {
    this.#name = name;
  }
  printName() {
    // Only private fields can be called inside the class
    console.log(`Your name is ${this.#name}`);
  }
}
```

```
const dog = new Dog("putty");
//console.log(this.#name)
//Private identifiers are not allowed outside class bodies.
dog.printName();
```

Static private class

```
class ClassWithPrivate {
  static #privateStaticField;
  static #privateStaticFieldWithInitializer = 42;

  static #privateStaticMethod() {
    // …
  }
}
```

Promises

make the commitment

```
new Promise((resolve, reject) => {
  if (ok) {
    resolve(result);
  } else {
    reject(error);
  }
});
```

for asynchronous programming. See: Promises

Using Promises

```
promise
  .then((result) => { ··· })
  .catch((error) => { ··· })
```

Using Promises in finally

```
promise
  .then((result) => { ··· })
```

```
  .catch((error) => { ··· })
  .finally(() => {
    /*logic independent of success/error */
  })
```

The handler is called when the promise is fulfilled or rejected

Promise function

```
Promise.all(···)
Promise.race(···)
Promise.reject(···)
Promise.resolve(···)
```

Async-await

```
async function run () {
  const user = await getUser()
  const tweets = await getTweets(user)
  return [user, tweets]
}
```

`async` functions are another way to use functions. See: Async Function

Destructuring

Destructuring assignment

Arrays

```
const [first, last] = ["Nikola", "Tesla"];
```

Objects

```
let { title, author } = {
  title: "The Silkworm",
  author: "R. Galbraith",
};
```

Supports matching arrays and objects. See: Destructuring

Defaults

```
const scores = [22, 33];
const [math = 50, sci = 50, arts = 50] = scores;
```

```
//Result:
//math === 22, sci === 33, arts === 50
```

A default value can be assigned when destructuring an array or object

Function parameters

```
function greet({ name, greeting }) {
  console.log(`${greeting}, ${name}!`);
}
```

```
greet({ name: "Larry", greeting: "Ahoy" });
```

Destructuring of objects and arrays can also be done in function parameters

Defaults

```
function greet({ name = "Rauno" } = {}) {
  console.log(`Hi ${name}!`);
}
```

```
greet(); // Hi Rauno!
greet({ name: "Larry" }); // Hi Larry!
```

Reassign keys

```
function printCoordinates({ left: x, top: y }) {
  console.log(`x: ${x}, y: ${y}`);
}
```

```
printCoordinates({ left: 25, top: 90 });
```

This example assigns `x` to the value of the `left` key

Loop

```
for (let {title, artist} of songs) {
  ...
```

```
}
```

Assignment expressions also work in loops

Object Deconstruction

```
const { id, ...detail } = song;
```

Use the `rest(...)` operator to extract some keys individually and the rest of the keys in the object

Spread Operator

Object Extensions

with object extensions

```
const options = {
  ...defaults,
  visible: true,
};
```

No object extension

```
const options = Object.assign({}, defaults, { visible: true });
```

The object spread operator allows you to build new objects from other objects. See: Object Spread

Array Expansion

with array extension

```
const users = [
  ...admins,
  ...editors,
  'rstacruz'
]
```

No array expansion

```
const users = admins.concat(editors).concat(["rstacruz"]);
```

The spread operator allows you to build new arrays in the same way. See: Spread operator

Functions

Function parameters

Default parameters

```
function greet(name = "Jerry") {
  return `Hello ${name}`;
}
```

Rest parameters

```
function fn(x, ...y) {
  // y is an array
  return x * y.length;
}
```

Extensions

```
fn(...[1, 2, 3]);
//same as fn(1, 2, 3)
```

Default (default), rest, spread (extension). See: function parameters

Arrow function

Arrow functions

```
setTimeout(() => {
  ...
})
```

with parameters

```
readFile('text.txt', (err, data) => {
  ...
})
```

implicit return

```
arr.map(n => n*2)
//no curly braces = implicit return
//Same as: arr.map(function (n) { return n*2 })
```

```
arr.map(n => ({
  result: n*2
}))
//Implicitly returning an object requires parentheses around the object
```

Like a function, but preserves `this`. See: Arrow functions

Parameter setting default value

```
function log(x, y = "World") {
  console.log(x, y);
}

log("Hello"); // Hello World
log("Hello", "China"); // Hello China
log("Hello", ""); // Hello
```

Used in conjunction with destructuring assignment defaults

```
function foo({ x, y = 5 } = {}) {
  console.log(x, y);
}

foo(); // undefined 5
```

name attribute

```
function foo() {}
foo.name; // "foo"
```

length property

```
function foo(a, b) {}
foo.length; // 2
```

Objects

Shorthand Syntax

```
module.exports = { hello, bye };
```

same below:

```
module.exports = {
  hello: hello,
  bye: bye,
};
```

See: Object Literals Enhanced

method

```
const App = {
  start() {
    console.log("running");
  },
};
//Same as: App = { start: function () {···} }
```

See: Object Literals Enhanced

Getters and setters

```
const App = {
  get closed () {
    return this.status === 'closed'
  },
  set closed (value) {
    this.status = value ? 'closed' : 'open'
  }
}
```

See: Object Literals Enhanced

Computed property name

```
let event = "click";
let handlers = {
  [`on${event}`]: true,
};
//Same as: handlers = { 'onclick': true }
```

See: Object Literals Enhanced

Extract value

```
const fatherJS = { age: 57, name: "Zhang San" }
Object.values(fatherJS)
//[57, "Zhang San"]
Object.entries(fatherJS)
//[["age", 57], ["name", "Zhang San"]]
```

Modules module

Imports import

```
import "helpers";
//aka: require('···')
```

```
import Express from "express";
//aka: const Express = require('···').default || require('···')
```

```
import { indent } from "helpers";
//aka: const indent = require('···').indent
```

```
import * as Helpers from "helpers";
//aka: const Helpers = require('···')
```

```
import { indentSpaces as indent } from "helpers";
//aka: const indent = require('···').indentSpaces
```

`import` is the new `require()`. See: Module imports

Exports export

```
export default function () { ··· }
//aka: module.exports.default = ···
```

```
export function mymethod () { ··· }
//aka: module.exports.mymethod = ···
```

```
export const pi = 3.14159;
//aka: module.exports.pi = ···
```

```
const firstName = "Michael";
const lastName = "Jackson";
const year = 1958;
export { firstName, lastName, year };
```

```
export * from "lib/math";
```

`export` is the new `module.exports`. See: Module exports

`as` keyword renaming

```
import {
  lastName as surname // import rename
} from './profile.js';

function v1() { ... }
function v2() { ... }

export { v1 as default };
//Equivalent to export default v1;

export {
  v1 as streamV1, // export rename
  v2 as streamV2, // export rename
  v2 as streamLatestVersion // export rename
};
```

Dynamically load modules

```
button.addEventListener("click", (event) => {
  import("./dialogBox.js")
    .then((dialogBox) => {
      dialogBox.open();
    })
    .catch((error) => {
      /*Error handling */
    });
});
```

ES2020 Proposal introduce `import()` function

import() allows module paths to be dynamically generated

```
const main = document.querySelector("main");

import(`./modules/${someVariable}.js`)
  .then((module) => {
    module.loadPageInto(main);
  })
  .catch((err) => {
    main.textContent = err.message;
  });
```

import.meta

ES2020 Added a meta property `import.meta` to the `import` command, which returns the meta information of the current module

```
new URL("data.txt", import.meta.url);
```

In the Node.js environment, `import.meta.url` always returns a local path, that is, a string of the `file:URL` protocol, such as `file:/// home/user/foo.js`

Import Assertions

static import

```
import json from "./package.json" assert { type: "json" };
//Import all objects in the json file
```

Dynamic Import

```
const json = await import("./package.json", { assert: { type: "json" } });
```

Generators

Generator function

```
function* idMaker() {
  let id = 0;
  while (true) {
    yield id++;
  }
}
```

```
let gen = idMaker();
gen.next().value; // → 0
gen.next().value; // → 1
gen.next().value; // → 2
```

it's complicated. See: Generators

For..of + iterator

```
let fibonacci = {
  [Symbol.iterator]() {
    let pre = 0,
      cur = 1;
    return {
      next() {
        [pre, cur] = [cur, pre + cur];
        return { done: false, value: cur };
      },
    };
  },
};

for (var n of fibonacci) {
  // truncate sequence at 1000
  if (n > 1000) break;
  console.log(n);
}
```

For iterating over generators and arrays. See: For..of iteration

Relationship with Iterator interface

```
var gen = {};
gen[Symbol.iterator] = function* () {
  yield 1;
  yield 2;
  yield 3;
};

[...gen]; // => [1, 2, 3]
```

The `Generator` function is assigned to the `Symbol.iterator` property, so that the `gen` object has the `Iterator` interface, which can be traversed by the `...` operator

Symbol.iterator property

```
function* gen() {
  /*some code */
}
var g = gen();

g[Symbol.iterator]() === g; // true
```

`gen` is a `Generator` function, calling it will generate a traverser object `g`. Its `Symbol.iterator` property, which is also an iterator object generation function, returns itself after execution

see also

Learn ES2015(babeljs.io)
ECMAScript 6 Features Overview (github.com)

Related Cheatsheet

- Express Cheatsheet — Quick Reference
- JSON Cheatsheet — Quick Reference
- Kubernetes Cheatsheet — Quick Reference
- TOML Cheatsheet — Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet — Quick Reference
- Arduino Programming Cheatsheet — Quick Reference
- HTML Canvas Cheatsheet — Quick Reference
- TypeScript Cheatsheet — Quick Reference

TypeScript

A TypeScript cheat sheet with the most important concepts, functions, methods, and more. A complete quick reference for beginners.

Getting Started

Installing the Compiler

```
npm install typescript --save-dev
npm tsc
```

Basic DataTypes

basic types

```
let isDone: boolean = false;
let age: number = 30;
let userName: string = "John";
let list: number[] = [1, 2, 3];
let tuple: [string, number] = ["hello", 10];
let notSure: any = 4;
notSure = "maybe a string instead";
```

enums

```
enum Color {
  Red,
  Green,
  Blue,
}
```

```
let c: Color = Color.Green;
```

interface

```
interface Person {
  firstName: string;
  lastName: string;
  age?: number; // Optional property
}

function greet(person: Person) {
  return "Hello, " + person.firstName + " " + person.lastName;
}
```

Functions

```
function add(x: number, y: number): number {
  return x + y;
}

let myAdd = function (x: number, y: number): number {
  return x + y;
};

let myArrowAdd = (x: number, y: number): number => x + y;

function buildName(firstName: string, lastName = "Smith") {
  return firstName + " " + lastName;
}

function buildFullName(firstName: string, ...restOfName: string[]) {
  return firstName + " " + restOfName.join(" ");
}
```

Classes

```
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

```
let greeter = new Greeter("world");
```

Inheritance

```
class Animal {
  move(distance: number = 0) {
    console.log(`Animal moved ${distance} meters.`);
  }
}

class Dog extends Animal {
  bark() {
    console.log("Woof! Woof!");
  }
}

const dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

Generics

```
function identity<T>(arg: T): T {
  return arg;
}

let output1 = identity<string>("myString");
let output2 = identity<number>(42);
```

Type Assertions

```
let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;
// or
let strLength2: number = (someValue as string).length;
```

Modules

Export

```
export interface StringValidator {
  isAcceptable(s: string): boolean;
}

export class ZipCodeValidator implements StringValidator {
  isAcceptable(s: string) {
    return s.length === 5;
  }
}
```

Import

```
import { ZipCodeValidator } from "./ZipCodeValidator";

let myValidator = new ZipCodeValidator();
```

Namespaces

```
namespace Validation {
  export interface StringValidator {
    isAcceptable(s: string): boolean;
  }

  export class LettersOnlyValidator implements StringValidator {
    isAcceptable(s: string) {
      return /^[A-Za-z]+$/.test(s);
    }
  }
}

let validator = new Validation.LettersOnlyValidator();
```

Union and Intersection Types

Union Types

```
function padLeft(value: string, padding: string | number) {
  if (typeof padding === "number") {
    return Array(padding + 1).join(" ") + value;
```

```
  }
  if (typeof padding === "string") {
    return padding + value;
  }
  throw new Error(`Expected string or number, got '${padding}'.`);
}
```

Intersection Types

```
interface ErrorHandling {
  success: boolean;
  error?: { message: string };
}

interface ArtworksData {
  artworks: { title: string }[];
}

type ArtworksResponse = ArtworksData & ErrorHandling;

const response: ArtworksResponse = {
  success: true,
  artworks: [{ title: "Mona Lisa" }],
};
```

Utility Types

Partial

```
interface User {
  id: number;
  name: string;
  age: number;
}

let partialUser: Partial<User> = {
  name: "Alice",
};
```

Readonly

```
let readonlyUser: Readonly<User> = {
  id: 1,
  name: "Bob",
  age: 25,
};

// readonlyUser.age = 26; // Error: cannot reassign a readonly property
```

Pick

```
type UserName = Pick<User, "name">;

let userName: UserName = {
  name: "Charlie",
};
```

Omit

```
type UserWithoutAge = Omit<User, "age">;

let userWithoutAge: UserWithoutAge = {
  id: 2,
  name: "Dave",
};
```

Decorators

Class Decorator

```
function sealed(constructor: Function) {
  Object.seal(constructor);
  Object.seal(constructor.prototype);
}

@sealed
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
```

```
    return "Hello, " + this.greeting;
  }
}
```

Method Decorator

```
function enumerable(value: boolean) {
  return function (
    target: any,
    propertyKey: string,
    descriptor: PropertyDescriptor
  ) {
    descriptor.enumerable = value;
  };
}

class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }

  @enumerable(false)
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

Async/Await

```
async function fetchData(url: string) {
  let response = await fetch(url);
  let data = await response.json();
  return data;
}
```

Also read

TypeScript

Related Cheatsheet

HTML Canvas Cheatsheet
Quick Reference

CSS 3 Cheatsheet
Quick Reference

Django Cheatsheet
Quick Reference

HTML Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

HTML

This HTML quick reference cheat sheet lists the common HTML and HTML5 tags in readable layout.

Getting Started

hello.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>HTML5 Boilerplate</title>
  </head>
  <body>
    <h1>Hello world, hello CheatSheets.zip!</h1>
  </body>
</html>
```

Or try it out in the jsfiddle

Comment

```
<!-- this is a comment -->

<!--
    Or you can comment out a
    large number of lines.
-->
```

Paragraph

```
<p>I'm from CheatSheets.zip</p>
<p>Share quick reference cheat sheet.</p>
```

See: The Paragraph element

HTML link

```
<a href="https://cheatsheets.zip">CheatSheets</a>
<a href="mailto:[email protected]">Email</a>
<a href="tel:+12345678">Call</a>
<a href="sms:+12345678&body=ha%20ha">Msg</a>
```

`href`	The URL that the hyperlink points to
`rel`	Relationship of the linked URL
`target`	Link target location: `_self`, `_blank`, `_top`, `_parent`

See: The <a> Attributes

Image Tag

```
<img loading="lazy" src="https://xxx.png" alt="Describe image here" width="400"
height="400" />
```

`src`	Required, Image location (URL | Path)
`alt`	Describe of the image
`width`	Width of the image
`height`	Height of the image
`loading`	How the browser should load

See: The Image Embed element

Text Formatting Tags

```
<b>Bold Text</b>
<strong>This text is important</strong>
<i>Italic Text</i>
<em>This text is emphasized</em>
<u>Underline Text</u>
<pre>Pre-formatted Text</pre>
```

```
<code>Source code</code>
<del>Deleted text</del>
<mark>Highlighted text (HTML5)</mark>
<ins>Inserted text</ins>
<sup>Makes text superscripted</sup>
<sub>Makes text subscripted</sub>
<small>Makes text smaller</small>
<kbd>Ctrl</kbd>
<blockquote>Text Block Quote</blockquote>
```

Headings

```
<h1>This is Heading 1</h1>
<h2>This is Heading 2</h2>
<h3>This is Heading 3</h3>
<h4>This is Heading 4</h4>
<h5>This is Heading 5</h5>
<h6>This is Heading 6</h6>
```

You should only have one h1 on your page

Section Divisions

`<div></div>`	Division or Section of Page Content
`<span></span>`	Section of text within other content
`<p></p>`	Paragraph of Text
` `	Line Break
`<hr>`	Basic Horizontal Line

These are the tags used to divide your page up into sections

Inline Frame

```
<iframe
  title="New York"
  width="342"
  height="306"
  id="gmap_canvas"
  src="https://maps.google.com/maps?
q=2880%20Broadway,%20New%20York&t=&z=13&ie=UTF8&iwloc=&output=embed"
  scrolling="no"
>
```

```
</iframe>
```

↓ Preview

See: The Inline Frame element

JavaScript in HTML

```
<script type="text/javascript">
  let text = "Hello CheatSheets.zip";
  alert(text);
</script>
```

External JavaScript

```
<body>
  ...

  <script src="app.js"></script>
</body>
```

CSS in HTML

```
<style type="text/css">
  h1 {
    color: purple;
  }
</style>
```

External stylesheet

```
<head>
```

```
  ...
  <link rel="stylesheet" href="style.css" />
</head>
```

HTML5 Tags

Document

```
<body>
  <header>
    <nav>...</nav>
  </header>
  <main>
    <h1>CheatSheets.zip</h1>
  </main>
  <footer>
    <p>©2023 CheatSheets.zip</p>
  </footer>
</body>
```

Header Navigation

```
<header>
  <nav>
    <ul>
      <li><a href="#">Edit Page</a></li>
      <li><a href="#">Twitter</a></li>
      <li><a href="#">Facebook</a></li>
    </ul>
  </nav>
</header>
```

HTML5 Tags

article	Content that’s independent
aside	Secondary content
audio	Embeds a sound, or an audio stream
bdi	The Bidirectional Isolate element

canvas	Draw graphics via JavaScript
data	Machine readable content
datalist	A set of pre-defined options
details	Additional information
dialog	A dialog box or sub-window
embed	Embeds external application
figcaption	A caption or legend for a figure
figure	A figure illustrated
footer	Footer or least important
header	Masthead or important information
main	The main content of the document
mark	Text highlighted
meter	A scalar value within a known range
nav	A section of navigation links
output	The result of a calculation
picture	A container for multiple image sources
progress	The completion progress of a task
rp	Provides fall-back parenthesis
rt	Defines the pronunciation of character
ruby	Represents a ruby annotation
section	A group in a series of related content
source	Resources for the media elements
summary	A summary for the <details> element
template	Defines the fragments of HTML
time	A time or date

track	Text tracks for the media elements
video	Embeds video
wbr	A line break opportunity

HTML5 Video

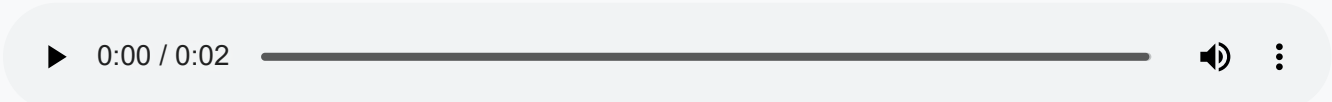
```
<video controls="" width="100%">
  <source src="https://interactive-examples.mdn.mozilla.net/media/cc0-
videos/flower.mp4" type="video/mp4" />
  Sorry, your browser doesn't support embedded videos.
</video>
```

↓ Preview

HTML5 Audio

```
<audio controls src="https://interactive-examples.mdn.mozilla.net/media/cc0-
audio/t-rex-roar.mp3">
  Your browser does not support the audio element.
</audio>
```

↓ Preview



HTML5 Ruby

```
<ruby> 汉 <rp>(</rp><rt>hàn</rt><rp>)</rp> 字 <rp>(</rp><rt>zì</rt><rp>)</rp>
</ruby>
```

↓ Preview

<ruby> 汉 <rp>(</rp><rt>hàn</rt><rp>)</rp> 字 <rp>(</rp><rt>zì</rt><rp>)</rp>
</ruby>

HTML5 kdi

```
<ul>
  <li>User <bdi>hrefs</bdi>: 60 points</li>
  <li>User <bdi>jdoe</bdi>: 80 points</li>
  <li>User <bdi>إيـان</bdi>: 90 points</li>
</ul>
```

↓ Preview

- User hrefs: 60 points
- User jdoe: 80 points
- User إيان: 90 points

HTML5 progress

```
<progress value="50" max="100"></progress>
```

<progress value="50" max="100"></progress>

HTML5 mark

```
<p>I Love <mark>CheatSheets.zip</mark></p>
```

I Love CheatSheets.zip

HTML Tables

Table Example

```
<table>
  <thead>
    <tr>
      <td>name</td>
      <td>age</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Roberta</td>
      <td>39</td>
    </tr>
    <tr>
      <td>Oliver</td>
      <td>25</td>
    </tr>
  </tbody>
</table>
```

HTML Table Tags

<table>	Defines a table
<th>	Defines a header cell in a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<caption>	Defines a table caption
<colgroup>	Defines a group of columns
<col>	Defines a column within a table
<thead>	Groups the header content
<tbody>	Groups the body content
<tfoot>	Groups the footer content

<td> Attributes

colspan	Number of columns a cell should span
headers	One or more header cells a cell is related to

`rowspan`	Number of rows a cell should span

See: td#Attributes

<th> Attributes

`colspan`	Number of columns a cell should span
`headers`	One or more header cells a cell is related to
`rowspan`	Number of rows a cell should span
`abbr`	Description of the cell's content
scope	The header element relates to

See: th#Attributes

HTML Lists

Unordered list

```
<ul>
  <li>I'm an item</li>
  <li>I'm another item</li>
  <li>I'm another item</li>
</ul>
```

See: The Unordered List element

Ordered list

```
<ol>
  <li>I'm the first item</li>
  <li>I'm the second item</li>
  <li>I'm the third item</li>
</ol>
```

See: The Ordered List element

Definition list

```
<dl>
  <dt>A Term</dt>
  <dd>Definition of a term</dd>
  <dt>Another Term</dt>
  <dd>Definition of another term</dd>
</dl>
```

See: The Description List element

HTML Forms

Form tags

```
<form method="POST" action="api/login">
  <label for="mail">Email: </label>
  <input type="email" id="mail" name="mail" />
  <br />
  <label for="pw">Password: </label>
  <input type="password" id="pw" name="pw" />
  <br />
  <input type="submit" value="Login" />
  <br />
  <input type="checkbox" id="ck" name="ck" />
  <label for="ck">Remember me</label>
</form>
```

↓ Preview

Email:

Password:

Login

Remember me

The HTML `<form>` element is used to collect and send information to an external source.

Form Attribute

`name`	Name of form for scripting
`action`	URL of form script

`method`	HTTP method, `POST` / `GET` (default)
`enctype`	Media type, See enctype
`onsubmit`	Runs when the form was submit
`onreset`	Runs when the form was reset

Label tags

```
<!-- Nested label -->
<label
  >Click me
  <input type="text" id="user" name="name" />
</label>
```

```
<!-- 'for' attribute -->
<label for="user">Click me</label>
<input id="user" type="text" name="name" />
```

`for` in a label references an input's `id` attribute

Input tags

```
<label for="Name">Name:</label> <input type="text" name="Name" id="" />
```

↓ Preview

Username:

See: HTML input Tags

Textarea tags

```
<textarea rows="2" cols="30" name="address" id="address"></textarea>
```

↓ Preview

Textarea is a multiple-line text input control

Radio Buttons

```
<input type="radio" name="gender" id="m" />
<label for="m">Male</label>
<input type="radio" name="gender" id="f" />
<label for="f">Female</label>
```

↓ Preview

○ Male ○ Female

Radio buttons are used to let the user select exactly one

Checkboxes

```
<input type="checkbox" name="s" id="soc" />
<label for="soc">Soccer</label>
<input type="checkbox" name="s" id="bas" />
<label for="bas">Baseball</label>
```

↓ Preview

☐ Soccer ☐ Baseball

Checkboxes allows the user to select one or more

Select tags

```
<label for="city">City:</label>
<select name="city" id="city">
  <option value="1">Sydney</option>
  <option value="2">Melbourne</option>
  <option value="3">Cromwell</option>
</select>
```

↓ Preview

City: Sydney

A select box is a dropdown list of options

Fieldset tags

```
<fieldset>
  <legend>Your favorite monster</legend>
```

```
  <input type="radio" id="kra" name="m" />
  <label for="kraken">Kraken</label><br />
  <input type="radio" id="sas" name="m" />
  <label for="sas">Sasquatch</label>
</fieldset>
```

↓ Preview

Your favorite monster

○ Kraken
○ Sasquatch

Datalist tags(HTML5)

```
<label for="b">Choose a browser: </label>
<input list="list" id="b" name="browser" />
<datalist id="list">
  <option value="Chrome"></option>
  <option value="Firefox"></option>
  <option value="Internet Explorer"></option>
  <option value="Opera"></option>
  <option value="Safari"></option>
  <option value="Microsoft Edge"></option>
</datalist>
```

↓ Preview

Choose a browser:

Submit and Reset Buttons

```
<form action="register.php" method="post">
  <label for="foo">Name:</label>
  <input type="text" name="name" id="foo" />
  <input type="submit" value="Submit" />
  <input type="reset" value="Reset" />
</form>
```

↓ Preview

Name: Submit Reset

`Submit` the data to server; `Reset` to default values

HTML input Tags

Input Attributes

The input tag is an empty element, identifying the particular type of field information to obtain from a user.

```
<input type="text" name="?" value="?" minlength="6" required />
```

`type="…"`	The type of data that is being input
`value="…"`	Default value
`name="…"`	Used to describe this data in the HTTP request
`id="…"`	Unique identifier that other HTML elements
`readonly`	Stops the user from modifying
`disabled`	Stops any interaction
`checked`	The radio or checkbox select or not
`required`	Being compulsory, See required
`placeholder="…"`	Adds a temporary, See ::placeholder
`autocomplete="off"`	Disable auto completion
`autocapitalize="none"`	Disable auto capitalization
`inputmode="…"`	Display a specific keyboard, See inputmode
`list="…"`	The id of an associated datalist
`maxlength="…"`	Maximum number of characters
`minlength="…"`	Minimum number of characters
`min="…"`	Minimum numerical value on range & number
`max="…"`	Maximum numerical value on range & number
`step="…"`	How the number will increment in range & number

`pattern="…"`	Specifies a Regular expression, See pattern
`autofocus`	Be focused
`spellcheck`	Perform spell checking
`multiple`	Whether to allow multiple values
`accept=""`	Expected file type in file upload controls

Also see: Attributes for the <input> element

Input types

`type="checkbox"`	☐
`type="radio"`	○
`type="file"`	Choose File No file chosen
`type="hidden"`	
`type="text"`	
`type="password"`	
`type="image"`	Login
`type="reset"`	Reset
`type="button"`	button
`type="submit"`	Submit

New Input Types in HTML5

`type="color"`	
`type="date"`	mm/dd/yyyy
`type="time"`	--:-- --
`type="month"`	--------- ----
`type="datetime-local"`	mm/dd/yyyy --:-- --
`type="week"`	Week --, ----

`type="email"`	
`type="tel"`	
`type="url"`	
`type="number"`	
`type="search"`	
`type="range"`	

Input CSS selectors

`input:focus`	When its keyboard focused

See: Input pseudo classes

HTML meta Tags

Meta tags

The meta tag describes meta data within an HTML document. It explains additional material about the HTML.

```
<meta charset="utf-8" />
```

```
<!-- title -->
<title>···</title>
<meta property="og:title" content="···" />
<meta name="twitter:title" content="···" />
```

```
<!-- url -->
<link rel="canonical" href="https://···" />
<meta property="og:url" content="https://···" />
<meta name="twitter:url" content="https://···" />
```

```
<!-- description -->
<meta name="description" content="···" />
<meta property="og:description" content="···" />
```

```
<meta name="twitter:description" content="···" />
```

```
<!-- image -->
<meta property="og:image" content="https://···" />
<meta name="twitter:image" content="https://···" />
```

```
<!-- ua -->
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
```

```
<!-- viewport -->
<meta name="viewport" content="width=device-width" />
<meta name="viewport" content="width=1024" />
```

Open Graph

```
<meta property="og:type" content="website" />
<meta property="og:locale" content="en_CA" />
<meta property="og:title" content="HTML cheatsheet" />
<meta property="og:url" content="https://cheatsheets.zip/html" />
<meta property="og:image" content="https://xxx.com/image.jpg" />
<meta property="og:site_name" content="Name of your website" />
<meta property="og:description" content="Description of this page" />
```

Used by Facebook, Instagram, Pinterest, LinkedIn, etc.

Twitter Cards

```
<meta name="twitter:card" content="summary" />
<meta name="twitter:site" content="@FechinLi" />
<meta name="twitter:title" content="HTML cheatsheet" />
<meta name="twitter:url" content="https://cheatsheets.zip/html" />
<meta name="twitter:description" content="Description of this page" />
<meta name="twitter:image" content="https://xxx.com/image.jpg" />
```

See: Twitter Card Documentation

Geotagging

```
<meta name="ICBM" content="45.416667,-75.7" />
<meta name="geo.position" content="45.416667;-75.7" />
<meta name="geo.region" content="ca-on" />
<meta name="geo.placename" content="Ottawa" />
```

See: Geotagging

Also see

HTML 4.01 Specification (w3.org)

Related Cheatsheet

HTML Canvas Cheatsheet
Quick Reference

CSS 3 Cheatsheet
Quick Reference

Django Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

CSS 3

This is a quick reference cheat sheet for CSS goodness, listing selector syntax, properties, units and other useful bits of information.

Getting Started

Introduction

CSS is rich in capabilities and is more than simply laying out pages.

External stylesheet

```
<link href="./path/to/stylesheet/style.css" rel="stylesheet" type="text/css" />
```

Internal stylesheet

```
<style>
  body {
    background-color: linen;
  }
</style>
```

Inline styles

```
<h2 style="text-align: center;">Centered text</h2>

<p style="color: blue; font-size: 18px;">Blue, 18-point text</p>
```

Add class

```
<div class="classname"></div>
<div class="class1 ... classn"></div>
```

Support multiple classes on one element.

!important

```
.post-title {
  color: blue !important;
}
```

Overrides all previous styling rules.

Selector

```
h1 { }
#job-title { }
div.hero { }
div > p { }
```

See: Selectors

Text color

```
color: #2a2aff;
color: green;
color: rgb(34, 12, 64, 0.6);
color: hsla(30 100% 50% / 0.6);
```

See: Colors

Background

```
background-color: blue;
background-image: url("nyan-cat.gif");
background-image: url("../image.png");
```

See: Backgrounds

Font

```
.page-title {
  font-weight: bold;
  font-size: 30px;
  font-family: "Courier New";
}
```

See: Fonts

Position

```
.box {
  position: relative;
  top: 20px;
  left: 20px;
}
```

See also: Position

Animation

```
animation: 300ms linear 0s infinite;

animation: bounce 300ms linear infinite;
```

See: Animation

Comment

```
/* This is a single line comment */

/* This is a
   multi-line comment */
```

Flex layout

```
div {
  display: flex;
  justify-content: center;
}
div {
  display: flex;
  justify-content: flex-start;
}
```

See: Flexbox | Flex Tricks

Grid layout

```
#container {
  display: grid;
  grid: repeat(2, 60px) / auto-flow 80px;
}
```

```
#container > div {
  background-color: #8ca0ff;
  width: 50px;
  height: 50px;
}
```

See: Grid Layout

Variable & Counter

```
counter-set: subsection;
counter-increment: subsection;
counter-reset: subsection 0;

:root {
  --bg-color: brown;
}
element {
  background-color: var(--bg-color);
}
```

See: Dynamic content

CSS Selectors

Examples

Groups Selector

```
h1,
h2 {
  color: red;
}
```

Chaining Selector

```
h3.section-heading {
  color: blue;
}
```

Attribute Selector

```
div[attribute="SomeValue"] {
  background-color: red;
}
```

First Child Selector

```
p:first-child {
  font-weight: bold;
}
```

No Children Selector

```
.box:empty {
  background: lime;
  height: 80px;
  width: 80px;
}
```

Basic

`*`	All elements
`div`	All div tags
`.classname`	All elements with class
`#idname`	Element with ID
`div,p`	All divs and paragraphs
`#idname *`	All elements inside #idname

See also: Type / Class / ID / Universal selectors

Combinators

`div.classname`	Div with certain classname
`div#idname`	Div with certain ID
`div p`	Paragraphs inside divs
`div > p`	All p tags one level deep in div
`div + p`	P tags immediately after div
`div ~ p`	P tags preceded by div

See also: Adjacent / Sibling / Child selectors

Attribute selectors

`a[target]`	With a target attribute
`a[target="_blank"]`	Open in new tab
`a[href^="/index"]`	Starts with /index
`[class | ="chair"]`	Starts with chair
`[class*="chair"]`	containing chair
`[title~="chair"]`	Contains the word chair
`a[href$=".doc"]`	Ends with .doc
`[type="button"]`	Specified type

See also: Attribute selectors

User action pseudo classes

`a:link`	Link in normal state
`a:active`	Link in clicked state
`a:hover`	Link with mouse over it
`a:visited`	Visited link

Pseudo classes

`p::after`	Add content after p
`p::before`	Add content before p
`p::first-letter`	First letter in p
`p::first-line`	First line in p
`::selection`	Selected by user
`::placeholder`	Placeholder attribute
`:root`	Documents root element
`:target`	Highlight active anchor

`div:empty`	Element with no children
`p:lang(en)`	P with en language attribute
`:not(span)`	Element that's not a span

Input pseudo classes

`input:checked`	Checked inputs
`input:disabled`	Disabled inputs
`input:enabled`	Enabled inputs
`input:focus`	Input has focus
`input:in-range`	Value in range
`input:out-of-range`	Input value out of range
`input:valid`	Input with valid value
`input:invalid`	Input with invalid value
`input:optional`	No required attribute
`input:required`	Input with required attribute
`input:read-only`	With readonly attribute
`input:read-write`	No readonly attribute
`input:indeterminate`	With indeterminate state

Structural pseudo classes

`p:first-child`	First child
`p:last-child`	Last child
`p:first-of-type`	First of some type
`p:last-of-type`	Last of some type
`p:nth-child(2)`	Second child of its parent
`p:nth-child(3n42)`	Nth-child (an + b) formula
`p:nth-last-child(2)`	Second child from behind

`p:nth-of-type(2)`	Second p of its parent
`p:nth-last-of-type(2)`	...from behind
`p:only-of-type`	Unique of its parent
`p:only-child`	Only child of its parent

CSS Fonts

Properties

`font-family:`	<font>
`font-size:`	<size>
`letter-spacing:`	<size>
`line-height:`	<number>
`font-weight:`	<number> / bold / normal
`font-style:`	italic / normal
`text-decoration:`	underline / none
`text-align:`	left / right center / justify
`text-transform:`	capitalize / uppercase / lowercase

See also: Font

Shorthand

`font:`	`italic`	`400`	`14px`	`/`	`1.5`	`sans-serif`
	style	weight	size (required)		line-height	family (required)

Example

```
font-family: Arial, sans-serif;
font-size: 12pt;
letter-spacing: 0.02em;
```

Case

```
/* Hello */
text-transform: capitalize;

/* HELLO */
text-transform: uppercase;

/* hello */
text-transform: lowercase;
```

@font-face

```
@font-face {
  font-family: "Glegoo";
  src: url("../Glegoo.woff");
}
```

CSS Colors

Named color

```
color: red;
color: orange;
color: tan;
color: rebeccapurple;
```

Hexadecimal color

```
color: #090;
color: #009900;
color: #090a;
color: #009900aa;
```

rgb() Colors

```
color: rgb(34, 12, 64, 0.6);
color: rgba(34, 12, 64, 0.6);
color: rgb(34 12 64 / 0.6);
color: rgba(34 12 64 / 0.3);
```

```
color: rgb(34 12 64 / 60%);
color: rgba(34.6 12 64 / 30%);
```

HSL Colors

```
color: hsl(30, 100%, 50%, 0.6);
color: hsla(30, 100%, 50%, 0.6);
color: hsl(30 100% 50% / 0.6);
color: hsla(30 100% 50% / 0.6);
color: hsl(30 100% 50% / 60%);
color: hsla(30.2 100% 50% / 60%);
```

Other

```
color: inherit;
color: initial;
color: unset;
color: transparent;

color: currentcolor; /* keyword */
```

CSS Backgrounds

Properties

background:	(Shorthand)
background-color:	See: Colors
background-image:	url(...)
background-position:	left/center/right top/center/bottom
background-size:	cover X Y
background-clip:	border-box padding-box content-box
background-repeat:	no-repeat repeat-x repeat-y

background-attachment:	scroll/fixed/local

Shorthand

background:	#ff0	url(a.jpg)	left	top	/	100px auto	no-repeat	fixed;
background:	#abc	url(b.png)	center	center	/	cover	repeat-x	local;
	color	image	posX	posY		size	repeat	attach..

Examples

```
background: url(img_man.jpg) no-repeat center;

background: url(img_flwr.gif) right bottom no-repeat, url(paper.gif) left top
    repeat;

background: rgb(2, 0, 36);
background: linear-gradient(
  90deg,
  rgba(2, 0, 36, 1) 0%,
  rgba(13, 232, 230, 1) 35%,
  rgba(0, 212, 255, 1) 100%
);
```

CSS The Box Model

Maximums/Minimums

```
.column {
  max-width: 200px;
  width: 500px;
}
```

See also: max-width / min-width / max-height / min-height

Margin / Padding

```
.block-one {
  margin: 20px;
  padding: 10px;
}
```

See also: Margin / Padding

Box-sizing

```
.container {
  box-sizing: border-box;
}
```

See also: Box-sizing

Visibility

```
.invisible-elements {
  visibility: hidden;
}
```

See also: Visibility

Auto keyword

```
div {
  margin: auto;
}
```

See also: Margin

Overflow

```
.small-block {
  overflow: scroll;
}
```

See also: Overflow

CSS Animation

Shorthand

`animation:`	`bounce`	`300ms`	`linear`	`100ms`	`infinite`	`alternate-reverse`	`both`
	name	duration	timing-function	delay	count	direction	fill-mode

Properties

`animation:`	(shorthand)
`animation-name:`	<name>
`animation-duration:`	<time>ms
`animation-timing-function:`	ease / linear / ease-in / ease-out / ease-in-out
`animation-delay:`	<time>ms
`animation-iteration-count:`	infinite / <number>
`animation-direction:`	normal / reverse / alternate / alternate-reverse
`animation-fill-mode:`	none / forwards / backwards / both / initial / inherit
`animation-play-state:`	normal / reverse / alternate / alternate-reverse

See also: Animation

Example

```
/* @keyframes duration | timing-function | delay |
   iteration-count | direction | fill-mode | play-state | name */
animation: 3s ease-in 1s 2 reverse both paused slidein;

/* @keyframes duration | timing-function | delay | name */
animation: 3s linear 1s slidein;

/* @keyframes duration | name */
animation: 3s slidein;

animation: 4s linear 0s infinite alternate move_eye;
animation: bounce 300ms linear 0s infinite normal;
animation: bounce 300ms linear infinite;
animation: bounce 300ms linear infinite alternate-reverse;
```

```
animation: bounce 300ms linear 2s infinite alternate-reverse forwards normal;
```

Javascript Event

```
.one('webkitAnimationEnd oanimationend msAnimationEnd animationend')
```

CSS Flexbox

Simple example

```
.container {
  display: flex;
}

.container > div {
  flex: 1 1 auto;
}
```

Container

.container {

```
display: flex;
display: inline-flex;

flex-direction: row; /* ltr - default */
flex-direction: row-reverse; /* rtl */
flex-direction: column; /* top-bottom */
flex-direction: column-reverse; /* bottom-top */

flex-wrap: nowrap; /* one-line */
flex-wrap: wrap; /* multi-line */

align-items: flex-start; /* vertical-align to top */
align-items: flex-end; /* vertical-align to bottom */
align-items: center; /* vertical-align to center */
align-items: stretch; /* same height on all (default) */

justify-content: flex-start; /* [xxx        ] */
```

```
justify-content: center; /* [    xxx    ] */
justify-content: flex-end; /* [        xxx] */
justify-content: space-between; /* [x    x    x] */
justify-content: space-around; /* [ x   x   x ] */
justify-content: space-evenly; /* [  x  x  x  ] */
```

}

Child

.container > div {

```
/* This: */
flex: 1 0 auto;

/* Is equivalent to this: */
flex-grow: 1;
flex-shrink: 0;
flex-basis: auto;


order: 1;


align-self: flex-start; /* left */
margin-left: auto; /* right */
```

}

CSS Flexbox Tricks

Vertical center

```
.container {
  display: flex;
}

.container > div {
  width: 100px;
  height: 100px;
  margin: auto;
}
```

Vertical center (2)

```
.container {
  display: flex;

  /* vertical */
  align-items: center;

  /* horizontal */
  justify-content: center;
}
```

Reordering

```
.container > .top {
  order: 1;
}

.container > .bottom {
  order: 2;
}
```

Mobile layout

```
.container {
  display: flex;
  flex-direction: column;
}

.container > .top {
  flex: 0 0 100px;
}

.container > .content {
  flex: 1 0 auto;
}
```

A fixed-height top bar and a dynamic-height content area.

Table-like

```
.container {
  display: flex;
}
```

```
/* the 'px' values here are just suggested percentages */
.container > .checkbox {
  flex: 1 0 20px;
}
.container > .subject {
  flex: 1 0 400px;
}
.container > .date {
  flex: 1 0 120px;
}
```

This creates columns that have different widths, but size accordingly according to the circumstances.

Vertical

```
.container {
  align-items: center;
}
```

Vertically-center all items.

Left and right

```
.menu > .left {
  align-self: flex-start;
}
.menu > .right {
  align-self: flex-end;
}
```

CSS Grid Layout

Grid Template Columns

```
#grid-container {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 20% 60%;
}
```

fr Relative Unit

```
.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 1fr 60px 1fr;
}
```

Grid Gap

```
/*The distance between rows is 20px*/
/*The distance between columns is 10px*/
#grid-container {
  display: grid;
  grid-gap: 20px 10px;
}
```

CSS Block Level Grid

```
#grid-container {
  display: block;
}
```

CSS grid-row

CSS syntax:

- grid-row: grid-row-start / grid-row-end;

Example

```
.item {
  grid-row: 1 / span 2;
}
```

CSS Inline Level Grid

```
#grid-container {
  display: inline-grid;
}
```

minmax() Function

```
.grid {
```

```
  display: grid;
  grid-template-columns: 100px minmax(100px, 500px) 100px;
}
```

grid-row-start & grid-row-end

CSS syntax:

- grid-row-start: auto|row-line;
- grid-row-end: auto|row-line|span n;

Example

```
grid-row-start: 2;
grid-row-end: span 2;
```

CSS grid-row-gap

```
grid-row-gap: length;
```

Any legal length value, like px or %. 0 is the default value

CSS grid-area

```
.item1 {
  grid-area: 2 / 1 / span 2 / span 3;
}
```

Justify Items

```
#container {
  display: grid;
  justify-items: center;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

CSS grid-template-areas

```
.item {
  grid-area: nav;
}
.grid-container {
```

```
  display: grid;
  grid-template-areas:
    "nav nav . ."
    "nav nav . .";
}
```

Justify Self

```
#grid-container {
  display: grid;
  justify-items: start;
}

.grid-items {
  justify-self: end;
}
```

The grid items are positioned to the right (end) of the row.

Align Items

```
#container {
  display: grid;
  align-items: start;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

CSS Dynamic Content

Variable

Define CSS Variable

```
:root {
  --first-color: #16f;
  --second-color: #ff7;
}
```

Variable Usage

```
#firstParagraph {
  background-color: var(--first-color);
  color: var(--second-color);
}
```

See also: CSS Variable

Counter

```
/* Set "my-counter" to 0 */
counter-set: my-counter;

/* Increment "my-counter" by 1 */
counter-increment: my-counter;

/* Decrement "my-counter" by 1 */
counter-increment: my-counter -1;

/* Reset "my-counter" to 0 */
counter-reset: my-counter;
```

See also: Counter set

Using counters

```
body {
  counter-reset: section;
}

h3::before {
  counter-increment: section;
  content: "Section." counter(section);
}


ol {
  counter-reset: section;
  list-marker-type: none;
}

li::before {
  counter-increment: section;
  content: counters(section, ".") " ";
```

```
}
```

Css 3 tricks

Scrollbar smooth

```
html {
  scroll-behavior: smooth;
}
```

Click me, the page will scroll smoothly to Getting started

Modern CSS

container queries(size)

```
.element-wrap {
  container: element / inline-size;
}
@container element (min-inline-size: 300px) {
  .element {
    display: flex;
    gap: 1rem;
  }
}
```

container qeries(style)

```
.container {
  --variant: 1;

  &.variant2 {
    --variant: 2;
  }
}

@container style(--variant: 1) {
```

```
  button {
  } /* You can't style .container, but can select inside it */
  .other-things {
  }
}

@container style(--variant: 2) {
  button {
  }
  .whatever {
  }
}
```

container units

- The units are cqw (“container query width”),
- cqh (“container query height”),
- cqi (“container query inline”),
- cqb (“container query block”),
- cqmin (smaller of cqi and cqb),
- and cqmax (larger of cqi and cqb)

```
.card {
  padding: 5cqi;
  font-size: 4cqi;
  border: 1cqi solid brown;
  height: 100%;
}

h2 {
  font-size: 10cqi;
  margin-block: 0 3cqi;
}
```

the :has() pseudo selector

```
figure:has(figcaption) {
  border: 1px solid black;
  padding: 0.5rem;
}
```

nesting

```
.cards {
  .card {
    & .card-description {
      color: blue;
    }
    & .card-title {
      color: red;
    }
  }
}
```

scoping

```
@scope {
  :scope {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 1rem;
    padding: 1rem;
    border: 1px solid black;
  }
  .card {
    padding: 1rem;
    border: 1px solid black;
    background: lightgray;
    h2 {
      margin: 0 0 1rem 0;
    }
  }
}
```

cascade layers

```
/* Specify the order to apply styles in cascade */
@layer legacyCard, newCard;

/* Imagine you have a lot of styles */
@layer newCard {
  .card {
    background-color: red;
  }
```

```
}
@layer legacyCard {
  .card {
    background-color: green;
  }
}
```

logical properties

```
button {
  background-color: #4caf50;
  border: none;
  color: white;
  padding: 0.5rem 1.5rem;
  text-decoration: none;
  font: inherit;
  border-radius: 4px;
  .icon {
    position: relative;
    top: 0.125em;
    fill: white;
    width: 1em;
    aspect-ratio: 1;
    margin-inline-end: 0.25rem;
  }
}
```

p3 colors

```
<div class="container">
  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-color: color(display-p3 1 0.5 0);
        }
      }
    </style>
  </div>
  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-color: oklch(61.88% 0.286 342.4);
        }
```

```
      }
    </style>
  </div>
  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-color: oklab(0.73 0.15 0.16);
        }
      }
    </style>
  </div>

  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-image: linear-gradient(to right in oklch, red, blue);
        }
      }
    </style>
  </div>

  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-image: linear-gradient(to right in oklab, red, blue);
        }
      }
    </style>
  </div>

  <div class="swatch">
    <style contenteditable>
      @scope {
        :scope {
          background-image: linear-gradient(to right in srgb, red, blue);
        }
      }
    </style>
  </div>
</div>
```

color mixing

```
.swatch {
  color: white;
  width: 100px;
  aspect-ratio: 1;
  display: grid;
  place-items: center;
  text-align: center;

  &:nth-child(1) {
    background-color: var(--bg);
  }
  &:nth-child(2) {
    background-color: color-mix(in oklch, var(--bg), black 30%);
  }
  &:nth-child(3) {
    background-color: color-mix(in oklch, var(--bg), white 30%);
  }
}
```

margin trim

```
.container {
  /* prevent "extra" margin at the end of the element */
  margin-trim: block-end;

  /* an element like this might be the culprit, but it could be anything */
  > p {
    margin-block-end: 1rem;
  }
}
```

text wrapping

```
.balance {
  text-wrap: balance;
}
.pretty {
  text-wrap: pretty;
}

html {
  font-family: system-ui, sans-serif;
}

main {
```

```
  max-inline-size: 60ch;
  margin-inline: auto;
}
```

subgrid

```
.grid {
  display: grid;
  grid-template-columns: repeat(9, 1fr);
  grid-template-rows: repeat(4, minmax(100px, auto));
}

.item {
  display: grid;
  grid-column: 2 / 7;
  grid-row: 2 / 4;
  grid-template-columns: subgrid;
  grid-template-rows: repeat(3, 80px);
}

.subitem {
  grid-column: 3 / 6;
  grid-row: 1 / 3;
}
```

Also see

frontendmasters.com
CSS selectors cheatsheet (frontend30.com)
MDN: Using CSS flexbox
Ultimate flexbox cheatsheet
GRID: A simple visual cheatsheet
CSS Tricks: A Complete Guide to Grid
Browser support

Related Cheatsheet

HTML Canvas Cheatsheet

Django Cheatsheet

Quick Reference

Quick Reference

HTML Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

React.js

A React.js cheat sheet with the most important concepts, functions, methods, and more. A complete quick reference for beginners.

Getting Started

JSX

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file.

```
let element = <h1>Hello, world!</h1>;

let emptyHeading = <h1 />;
```

JSX Expressions

```
let name = "Josh Perez";
let element = <h1>Hello, {name}</h1>;

function fullName(firstName, lastName) {
  return firstName + " " + lastName;
}
let element = <h1>Hello, {fullName("Julie", "Johnson")}</h1>;
```

JSX Attributes

```
const element = <img src={user.avatarUrl} />;
const element = <button className="btn">Click me</button>;
```

JSX Functions

```
name() {
  return "Julie";
}
```

```
return (
  <h1>
    Hi {name()}!
  </h1>
)
```

JSX Conditional Rendering

```
import React from "react";
export default function Weather(props) {
  if (props.temperature >= 20) {
    return (
      <p>
        It is {props.temperature}°C (Warm) in {props.city}
      </p>
    );
  } else {
    return (
      <p>
        It is {props.temperature}°C in {props.city}
      </p>
    );
  }
}
```

Note: A component must always return something.

Components

Functional Component

```
import React from "react";

export default function UserProfile() {
  return (
    <div className="UserProfile">
      <div>Hello</div>
      <div>World</div>
    </div>
  );
```

```
}
```

Note: Every component needs one root element

Embed an internal Component

```
import React from "react";
import UserAvatar from "./UserAvatar";

export default function UserProfile() {
  return (
    <div className="UserProfile">
      <UserAvatar />
      <UserAvatar />
    </div>
  );
}
```

Note: Assuming UserAvatar is declared in UserAvatar.js

Embed an external Component

```
import React from "react";
import ComponentName from "component-name";

export default function UserProfile() {
  return (
    <div className="UserProfile">
      <ComponentName />
    </div>
  );
}
```

Note: External components are found on npmjs.com and need to be imported first.

Advanced Functional Components

```
import React from "react";

export default function Hello(props) {
  function fullName() {
    return `${props.firstName} ${props.lastName}`;
  }
  return <p>{fullName()}</p>;
}
```

```
<Hello firstName="Matt" lastName="Delac" />;
```

Properties

Passing Properties to a Component

```
<Student
  firstName="Julie"
  lastName="Johnson"
  age={23}
  pro={true}
/>
```

Assigning the Properties from a Component

```
import React from "react";

export default function Student(props) {
  return (
    <h1>
      {props.firstName} {props.lastName} is {props.age}.
    </h1>
  );
}
```

States

React State

```
import React, { useState } from "react";

export default function Hello(props) {
  let [name, setName] = useState("Julie");
  function updateName() {
    let newName = prompt("What is your name?");
    setName(newName);
```

```
  }

  return (
    <div>
      <h1>{name}</h1>
      <button onClick={updateName}>Update name</button>
    </div>
  );
}
```

Events

Event Listener

```
import React from "react";

export default function Hello() {
  function handleClick(event) {
    event.preventDefault();
    alert("Hello World");
  }

  return (
    <a href="/" onClick={handleClick}>
      Say Hi
    </a>
  );
}
```

Note: The most common event listeners are onClick for links/buttons and onSubmit for forms.

Loops

Looping through an Array

```
let elements = ["one", "two", "three"];
```

```
return (
  <ul>
    {elements.map(function (value, index) {
      return <li key={index}>{value}</li>;
    })}
  </ul>
);
```

Note: Each list item inside a map loop needs a key attribute with a unique value which is generally the index.

Looping through an Array of Objects

```
let elements = [
  {
    name: "one",
    value: 1,
  },
  {
    name: "two",
    value: 2,
  },
  {
    name: "three",
    value: 3,
  },
];
return (
  <ul>
    {elements.map(function (element, index) {
      return (
        <li key={index}>
          The value for {element.name} is {element.value}
        </li>
      );
    })}
  </ul>
);
```

Note: Each list item inside a map loop needs a key attribute with a unique value which is generally the index.

Forms

React Forms

```
import React, { useState } from "react";

export default function LoginForm() {
  let [username, setUsername] = useState("");
  let [password, setPassword] = useState("");

  function handleSubmit(event) {
    event.preventDefault();
    alert(`Loging in with ${username} and ${password}`);
  }

  function updateUsername(event) {
    setUsername(event.target.value);
  }

  function updatePassword(event) {
    setPassword(event.target.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" placeholder="Username" onChange={updateUsername} />
      <input type="password" placeholder="Password" onChange={updatePassword} />
      <input type="submit" value="Login" />
    </form>
  );
}
```

CSS

CSS in a React Component

```
import React from "react";
import "./Student.css";

export default function Student() {
  return <div className="Student">Julie Johnson</div>;
```

```
}
```

Note: You'll then have to crate a css file called Student.css

AJAX

AJAX Request with Axios

```
import React from "react";
import axios from "axios";

export default function Weather(props) {
  function handleResponse(response) {
    console.log(response);
  }

  if (notifications) {
    return <p>notifications</p>;
  } else {
    let url = `https://notifications.com`;
    axios.get(url).then(handleResponse);
    return <p>Loading notifications..</p>;
  }
}
```

Note: Make sure to import Axios first to your project.

Hooks

useState Hook

```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
```

```
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

Note: The useState Hook is a built-in React Hook that allows functional components to manage local state. It provides a way to declare state variables and update them within a functional component. Example code illustrating how to use it

Multiple State Variable Declaration

```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("");
  const [isCompleted, setIsCompleted] = useState(false);

  const handleIncrement = () => {
    setCount(count + 1);
  };

  const handleNameChange = (event) => {
    setName(event.target.value);
  };

  const toggleCompletion = () => {
    setIsCompleted(!isCompleted);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleIncrement}>Increment</button>

      <input type="text" value={name} onChange={handleNameChange} placeholder="Ent

      <label>
        <input type="checkbox" checked={isCompleted} onChange={toggleCompletion} /
        Completed
      </label>
```

```
    </div>
  );
}

export default Counter;
```

Note: You can declare multiple state variables using the useState Hook by calling it multiple times in a functional component. Each call to useState manages a separate piece of state.

Input State Management

```
import { useState } from "react";

function FormExample() {
  const [formData, setFormData] = useState({ name: "", email: "", message: "" });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData((prevFormData) => ({ ...prevFormData, [name]: value }));
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Name: ${formData.name}, Email: ${formData.email}, Message: ${formData.m
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="name">Name:</label>
      <input type="text" id="name" name="name" value={formData.name} onChange={har

      <label htmlFor="email">Email:</label>
      <input type="email" id="email" name="email" value={formData.email} onChange=

      <label htmlFor="message">Message:</label>
      <textarea id="message" name="message" value={formData.message} onChange={har

      <button type="submit">Submit</button>
    </form>
  );
}

export default FormExample;
```

useEffect Hook

```
import React, { useState, useEffect } from "react";

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prevSeconds) => prevSeconds + 1);
    }, 1000);

    return () => clearInterval(interval);
  }, []);

  return <div>Seconds: {seconds}</div>;
}

export default Timer;
```

Note: The useEffect Hook in React is used for performing side effects in functional components. It allows you to execute code based on component lifecycle events like mounting, updating, and unmounting.

Fetch API using useEffect

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function UserList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/users")
      .then((response) => {
        setUsers(response.data);
      })
      .catch((error) => {
        console.error("Error fetching users:", error);
      });
  }, []);

  return (
    <div>
```

```
      <h2>User List</h2>
      <ul>
        {users.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default UserList;
```

Note: Make sure to import Axios first to your project.

Custom Hook creation useLocalStorage

```
import { useState, useEffect } from "react";

function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    const storedValue = localStorage.getItem(key);
    return storedValue !== null ? JSON.parse(storedValue) : initialValue;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}

export default useLocalStorage;
```

Note: Custom Hooks are reusable functions in React that contain logic shared across multiple components. They allow you to extract stateful logic from components into standalone functions.

Related Cheatsheet

HTML Canvas Cheatsheet
Quick Reference

CSS 3 Cheatsheet
Quick Reference

Django Cheatsheet
Quick Reference

HTML Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Express

A quick reference cheatsheet for Express, a flexible and streamlined web framework for Node.js

Getting Started

Hello World

1 "Create project, add `package.json` configuration

```
$ mkdir myapp # create directory
$ cd myapp    # enter the directory
$ npm init -y # Initialize a configuration
```

2 Install dependencies

```
$ npm install express
```

3 Entry file `index.js` add code:

```
const express = require("express");
const app = express();
const port = 3000;
app.get("/", (req, res) => {
  res.send("Hello World!");
});
app.listen(port, () => {
  console.log(`Listening port on ${port}`);
});
```

4 Run the application using the following command

```
$ node index.js
```

express -h

```
Usage: express [options] [dir]
```

```
Options:
  -h, --help output usage information
      --version output version number
  -e, --ejs add ejs engine support
      --hbs add hbs engine support
      --pug add pug engine support
  -H, --hogan add hogan.js engine support
      --no-view No view engine generated
  -v, --view <engine> add view <engine> support (ejs|hbs|hjs|jade|pug|twig|vash) (
  -c, --css <engine> add stylesheet <engine> support (less|stylus|compass|sass) (d
      --git add .gitignore
  -f, --force force non-empty directories
```

Create a `myapp` project

```
$ express --view=pug myapp
# run the application
$ DEBUG=myapp:*npm start
```

express()

`express.json()`	#
`express.raw()`	#
`express.Router()`	#
`express.static()`	#
`express.text()`	#
`express.urlencoded()`	#

Router

`router.all()`	#
`router.METHOD()`	#
`router.param()`	#
`router.route()`	#
`router.use()`	#

Application

```
var express = require("express");
var app = express();

console.dir(app.locals.title);
//=> 'My App'
console.dir(app.locals.email);
//=> '[email protected]'
```

Attribute	
app.locals	Local variables in the application #
app.mountpath	Path pattern for mounting sub-apps #

Events	
mount	The child application is mounted on the parent application, and the event is triggered on the child application #

Method	
app.all()	#
app.delete()	#
app.disable()	#
app.disabled()	#
app.enable()	#
app.enabled()	#
app.engine()	#
app.get(name)	#
app.get(path, callback)	#
app.listen()	#
app.METHOD()	#
app.param()	#
app.path()	#
app.post()	#

app.put()	#
app.render()	#
app.route()	#
app.set()	#
app.use()	#

Request

Attribute	
req.app	#
req.baseUrl	#
req.body	#
req.cookies	#
req.fresh	#
req.hostname	#
req.ip	#
req.ips	#
req.method	#
req.originalUrl	#
req.params	#
req.path	#
req.protocol	#
req.query	#
req.route	#
req.secure	#
req.signedCookies	#
req.stale	#

req.subdomains	#
req.xhr	#

Method	
req.accepts()	#
req.acceptsCharsets()	#
req.acceptsEncodings()	#
req.acceptsLanguages()	#
req.get()	Get HTTP request header fields #
req.is()	#
req.param()	#
req.range()	#

Response

```
app.get("/", function (req, res) {
  console.dir(res.headersSent); //false
  res.send("OK");
  console.dir(res.headersSent); //true
});
```

Attribute	
res.app	#
res.headersSent	#
res.locals	#

Method	
res.append()	#
res.attachment()	#
res.cookie()	#
res.clearCookie()	#
res.download()	Prompt for files to download #

`res.end()`	end the response process #
`res.format()`	#
`res.get()`	#
`res.json()`	Send JSON response #
`res.jsonp()`	Send a response with JSONP support #
`res.links()`	#
`res.location()`	#
`res.redirect()`	Redirect request #
`res.render()`	render view template #
`res.send()`	Send various types of responses #
`res.sendFile()`	Send a file as an octet stream #
`res.sendStatus()`	#
`res.set()`	#
`res.status()`	#
`res.type()`	#
`res.vary()`	#

Example

Router

Called for any request passed to this router

```
router.use(function (req, res, next) {
  //.. some logic here .. like any other middleware
  next();
});
```

will handle any request ending in `/events`

```
//depends on where the router "use()"
router.get("/events", (req, res, next) => {
  //..
});
```

Response

The `res` object represents the HTTP response sent by the `Express` application when it receives an HTTP request

```
app.get("/user/:id", (req, res) => {
  res.send("user" + req.params.id);
});
```

Request

A `req` object represents an `HTTP` request and has properties for the request query string, parameters, body, HTTP headers, etc.

```
app.get("/user/:id", (req, res) => {
  res.send("user" + req.params.id);
});
```

res. end()

```
res.end();
res.status(404).end();
```

End the response process. This method actually comes from the Node core, specifically the `response.end()` method of `http.ServerResponse`

res.json([body])

```
res.json(null);
res.json({ user: "tobi" });
res.status(500).json({ error: "message" });
```

app.all

```
app.all("/secret", function (req, res, next) {
  console.log("access secret section...");
  next(); // Pass control to the next handler
});
```

app.delete

```
app.delete("/", function (req, res) {
  res.send("DELETE request to homepage");
});
```

app.disable(name)

```
app.disable("trust proxy");
app.get("trust proxy");
// => false
```

app.disabled(name)

```
app.disabled("trust proxy");
// => true

app.enable("trust proxy");
app.disabled("trust proxy");
// => false
```

app.engine(ext, callback)

```
var engines = require("consolidate");

app.engine("haml", engines.haml);
app.engine("html", engines.hogan);
```

app.listen([port[, host[, backlog]]][, callback])

```
var express = require("express");

var app = express();
app.listen(3000);
```

Routing

```
const express = require("express");
const app = express();

//Respond to "hello world" when making a GET request to the homepage
app.get("/", (req, res) => {
  res.send("hello world");
});
```

```
// GET method routing
app.get("/", (req, res) => {
  res.send("GET request to the homepage");
});

// POST method routing
app.post("/", (req, res) => {
  res.send("POST request to the homepage");
});
```

Middleware

```
function logOriginalUrl(req, res, next) {
  console.log("ReqURL:", req.originalUrl);
  next();
}

function logMethod(req, res, next) {
  console.log("Request Type:", req.method);
  next();
}

const log = [logOriginalUrl, logMethod];

app.get("/user/:id", log, (req, res, next) => {
  res.send("User Info");
});
```

Using templates

```
app.set("view engine", "pug");
```

Create a `Pug` template file named `index.pug` in the `views` directory with the following content

```
html
  the head
    title= title
  the body
    h1=message
```

Create a route to render the `index.pug` file. If the view engine property is not set, the extension of the view file must be specified

```
app.get("/", (req, res) => {
  res.render("index", {
    title: "Hey",
    message: "Hello there!",
  });
});
```

Related Cheatsheet

- ES6 Cheatsheet
  Quick Reference
- JSON Cheatsheet
  Quick Reference
- Kubernetes Cheatsheet
  Quick Reference
- TOML Cheatsheet
  Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet
  Quick Reference
- Arduino Programming Cheatsheet
  Quick Reference
- HTML Canvas Cheatsheet
  Quick Reference
- TypeScript Cheatsheet
  Quick Reference

GraphQL

This quick reference cheat sheet provides a brief overview of GraphQL.

Getting Started

Overview

- An alternative approach to RESTful APIs
- GraphQL is a query language for APIs
- Easily describe the shape of the GraphQL API using clear shared terms.
- Clients issue queries/mutations to read and update data
- GraphQL syntax can express complex entity relations
- Libraries to implement GraphQL in different languages

→ GraphQL

Schema

`schema`	GraphQL schema definition
`query`	Read and traverse data
`mutation`	Modify data or trigger an action
`subscription`	Run a query when an event occurs

Built-in Scalar Types

`Int`	Signed 32-bit integer
`Float`	Signed double-precision floating-point value

`String`	UTF-8 character sequence
`Boolean`	true or false
`ID`	A Unique identifier

Type Definitions

`scalar`	Scalar Type
`type`	Object Type
`interface`	Interface Type
`union`	Union Type
`enum`	Enum Type
`input`	Input Object Type

Type Modifiers

`String`	Nullable String
`String!`	Non-null String
`[String]`	List of nullable Strings
`[String]!`	Non-null list of nullable Strings
`[String!]!`	Non-null list of non-null Strings

Input Arguments

Basic Input

```
type Query {
    users(limit: Int): [User]
}
```

Input with default value

```
type Query {
    users(limit: Int = 10): [User]
}
```

Input with multiple arguments

```
type Query {
    users(limit: Int, sort: String): [User]
}
```

Input with multiple arguments and default values

```
type Query {
    users(limit: Int = 10, sort: String): [User]
}
type Query {
    users(limit: Int, sort: String = "asc"): [User]
}
type Query {
    users(limit: Int = 10, sort: String = "asc"): [User]
}
```

Input Types

```
input ListUsersInput {
    limit: Int
    since_id: ID
}

type Mutation {
    users(params: ListUsersInput): [User]!
}
```

Custom Scalars

```
scalar Url
type User {
    name: String
    homepage: Url
}
```

Interfaces

```
interface Foo {
    is_foo: Boolean
}
interface Goo {
    is_goo: Boolean
}
type Bar implements Foo {
    is_foo: Boolean
```

```
    is_bar: Boolean
}
type Baz implements Foo, Goo {
    is_foo: Boolean
    is_goo: Boolean
    is_baz: Boolean
}
```

Object implementing one or more Interfaces

Unions

```
type Foo {
    name: String
}
type Bar {
    is_bar: String
}
union SingleUnion = Foo
union MultipleUnion = Foo | Bar
type Root {
    single: SingleUnion
    multiple: MultipleUnion
}
```

Union of one or more Objects

Enums

```
enum USER_STATE {
    NOT_FOUND
    ACTIVE
    INACTIVE
    SUSPENDED
}
type Root {
    stateForUser(userID: ID!): USER_STATE!
    users(state: USER_STATE, limit: Int = 10): [User]
}
```

Also see

GraphQL Schema Language Cheat Sheet (github.com)

Top Cheatsheet

Python Cheatsheet
Quick Reference

Vim Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Bash Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Python

The Python cheat sheet is a one-page reference sheet for the Python 3 programming language.

Getting Started

Introduction

- Python (python.org)
- Python Document (docs.python.org)
- Learn X in Y minutes (learnxinyminutes.com)
- Regex in python (cheatsheets.zip)

Hello World

```
>>> print("Hello, World!")
Hello, World!
```

The famous "Hello World" program in Python

Variables

```
age = 18      # age is of type int
name = "John" # name is now of type str
print(name)
```

Python can't declare a variable without assignment.

Data Types

`str`	Text
`int`, `float`, `complex`	Numeric

`list`, `tuple`, `range`	Sequence
`dict`	Mapping
`set`, `frozenset`	Set
`bool`	Boolean
`bytes`, `bytearray`, `memoryview`	Binary

See: Data Types

Slicing String

```
>>> msg = "Hello, World!"
>>> print(msg[2:5])
llo
```

See: Strings

Lists

```
mylist = []
mylist.append(1)
mylist.append(2)
for item in mylist:
    print(item) # prints out 1,2
```

See: Lists

If Else

```
num = 200
if num > 0:
    print("num is greater than 0")
else:
    print("num is not greater than 0")
```

See: Flow control

Loops

```
for item in range(6):
    if item == 3: break
    print(item)
```

```
else:
    print("Finally finished!")
```

See: Loops

Functions

```
>>> def my_function():
...     print("Hello from a function")
...
>>> my_function()
Hello from a function
```

See: Functions

File Handling

```
with open("myfile.txt", "r", encoding='utf8') as file:
    for line in file:
        print(line)
```

See: File Handling

Arithmetic

```
result = 10 + 30 # => 40
result = 40 - 10 # => 30
result = 50 * 5  # => 250
result = 16 / 4  # => 4.0 (Float Division)
result = 16 // 4 # => 4 (Integer Division)
result = 25 % 2  # => 1
result = 5 ** 3  # => 125
```

The / means quotient of x and y, and the // means floored quotient of x and y, also see StackOverflow

Plus-Equals

```
counter = 0
counter += 10           # => 10
counter = 0
counter = counter + 10  # => 10

message = "Part 1."
```

```
# => Part 1.Part 2.
message += "Part 2."
```

f-Strings (Python 3.6+)

```
>>> website = 'Quickref.ME'
>>> f"Hello, {website}"
"Hello, Quickref.ME"

>>> num = 10
>>> f'{num} + 10 = {num + 10}'
'10 + 10 = 20'
```

See: Python F-Strings

Python Built-in Data Types

Strings

```
hello = "Hello World"
hello = 'Hello World'

multi_string = """Multiline Strings
Lorem ipsum dolor sit amet,
consectetur adipiscing elit """
```

See: Strings

Numbers

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

>>> print(type(x))
<class 'int'>
```

Booleans

```
my_bool = True
```

```
my_bool = False

bool(0)     # => False
bool(1)     # => True
```

Lists

```
list1 = ["apple", "banana", "cherry"]
list2 = [True, False, False]
list3 = [1, 5, 7, 9, 3]
list4 = list((1, 5, 7, 9, 3))
```

See: Lists

Tuple

```
my_tuple = (1, 2, 3)
my_tuple = tuple((1, 2, 3))
```

Similar to List but immutable

Set

```
set1 = {"a", "b", "c"}
set2 = set(("a", "b", "c"))
```

Set of unique items/objects

Dictionary

```
>>> empty_dict = {}
>>> a = {"one": 1, "two": 2, "three": 3}
>>> a["one"]
1
>>> a.keys()
dict_keys(['one', 'two', 'three'])
>>> a.values()
dict_values([1, 2, 3])
>>> a.update({"four": 4})
>>> a.keys()
dict_keys(['one', 'two', 'three', 'four'])
>>> a['four']
4
```

Key: Value pair, JSON like object

Casting

Integers

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Floats

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Strings

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Python Advanced Data Types

Heaps

```
import heapq

myList = [9, 5, 4, 1, 3, 2]
heapq.heapify(myList) # turn myList into a Min Heap
print(myList)    # => [1, 3, 2, 5, 9, 4]
print(myList[0]) # first value is always the smallest in the heap

heapq.heappush(myList, 10) # insert 10
x = heapq.heappop(myList)  # pop and return smallest item
print(x)                   # => 1
```

Negate all values to use Min Heap as Max Heap

```
myList = [9, 5, 4, 1, 3, 2]
myList = [-val for val in myList] # multiply by -1 to negate
heapq.heapify(myList)
```

```
x = heapq.heappop(myList)
print(-x) # => 9 (making sure to multiply by -1 again)
```

Heaps are binary trees for which every parent node has a value less than or equal to any of its children. Useful for accessing min/max value quickly. Time complexity: O(n) for heapify, O(log n) push and pop. See: Heapq

Stacks and Queues

```
from collections import deque

q = deque()          # empty
q = deque([1, 2, 3]) # with values

q.append(4)     # append to right side
q.appendleft(0) # append to left side
print(q)    # => deque([0, 1, 2, 3, 4])

x = q.pop() # remove & return from right
y = q.popleft() # remove & return from left
print(x)    # => 4
print(y)    # => 0
print(q)    # => deque([1, 2, 3])

q.rotate(1) # rotate 1 step to the right
print(q)    # => deque([3, 1, 2])
```

Deque is a double-ended queue with O(1) time for append/pop operations from both sides. Used as stacks and queues. See: Deque

Python Strings

Array-like

```
>>> hello = "Hello, World"
>>> print(hello[1])
e
>>> print(hello[-1])
d
```

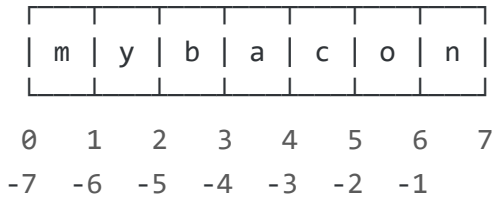
Get the character at position 1 or last

Looping

```
>>> for char in "foo":
...     print(char)
f
o
o
```

Loop through the letters in the word "foo"

Slicing string




```
>>> s = 'mybacon'
>>> s[2:5]
'bac'
>>> s[0:2]
'my'


>>> s = 'mybacon'
>>> s[:2]
'my'
>>> s[2:]
'bacon'
>>> s[:2] + s[2:]
'mybacon'
>>> s[:]
'mybacon'


>>> s = 'mybacon'
>>> s[-5:-1]
'baco'
>>> s[2:6]
'baco'
```

With a stride

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
>>> s[::-5]
'55555'
>>> s[::-1]
'5432154321543215432154321'
```

String Length

```
>>> hello = "Hello, World!"
>>> print(len(hello))
13
```

The len() function returns the length of a string

Multiple copies

```
>>> s = '===+'
>>> n = 8
>>> s * n
'===+===+===+===+===+===+===+===+'
```

Check String

```
>>> s = 'spam'
>>> s in 'I saw spamalot!'
True
>>> s not in 'I saw The Holy Grail!'
True
```

Concatenates

```
>>> s = 'spam'
>>> t = 'egg'
>>> s + t
'spamegg'
>>> 'spam' 'egg'
'spamegg'
```

Formatting

```
name = "John"
print("Hello, %s!" % name)


name = "John"
age = 23
print("%s is %d years old." % (name, age))
```

format() Method

```
txt1 = "My name is {fname}, I'm {age}".format(fname="John", age=36)
txt2 = "My name is {0}, I'm {1}".format("John", 36)
txt3 = "My name is {}, I'm {}".format("John", 36)
```

Input

```
>>> name = input("Enter your name: ")
Enter your name: Tom
>>> name
'Tom'
```

Get input data from console

Join

```
>>> "#".join(["John", "Peter", "Vicky"])
'John#Peter#Vicky'
```

Endswith

```
>>> "Hello, world!".endswith("!")
True
```

Python F-Strings (Since Python 3.6+)

f-Strings usage

```
>>> website = 'Quickref.ME'
>>> f"Hello, {website}"
```

```
"Hello, Quickref.ME"

>>> num = 10
>>> f'{num} + 10 = {num + 10}'
'10 + 10 = 20'

>>> f"""He said {"I'm John"}"""
"He said I'm John"

>>> f'5 {"{stars}"}'
'5 {stars}'
>>> f'{{5}} {"stars"}'
'{5} stars'

>>> name = 'Eric'
>>> age = 27
>>> f"""Hello!
...     I'm {name}.
...     I'm {age}."""
"Hello!\n    I'm Eric.\n    I'm 27."
```

it is available since Python 3.6, also see: Formatted string literals

f-Strings Fill Align

```
>>> f'{"text":10}'     # [width]
'text      '
>>> f'{"test":*>10}'   # fill left
'******test'
>>> f'{"test":*<10}'   # fill right
'test******'
>>> f'{"test":*^10}'   # fill center
'***test***'
>>> f'{12345:0>10}'    # fill with numbers
'0000012345'
```

f-Strings Type

```
>>> f'{10:b}'        # binary type
'1010'
>>> f'{10:o}'        # octal type
'12'
>>> f'{200:x}'       # hexadecimal type
'c8'
>>> f'{200:X}'
```

```
'C8'
>>> f'{345600000000:e}' # scientific notation
'3.456000e+11'
>>> f'{65:c}'       # character type
'A'
>>> f'{10:#b}'      # [type] with notation (base)
'0b1010'
>>> f'{10:#o}'
'0o12'
>>> f'{10:#x}'
'0xa'
```

F-Strings Others

```
>>> f'{-12345:0=10}'  # negative numbers
'-000012345'
>>> f'{12345:010}'    # [0] shortcut (no align)
'0000012345'
>>> f'{-12345:010}'
'-000012345'
>>> import math       # [.precision]
>>> math.pi
3.141592653589793
>>> f'{math.pi:.2f}'
'3.14'
>>> f'{1000000:,.2f}' # [grouping_option]
'1,000,000.00'
>>> f'{1000000:_.2f}'
'1_000_000.00'
>>> f'{0.25:0%}'      # percentage
'25.000000%'
>>> f'{0.25:.0%}'
'25%'
```

F-Strings Sign

```
>>> f'{12345:+}'      # [sign] (+/-)
'+12345'
>>> f'{-12345:+}'
'-12345'
>>> f'{-12345:+10}'
'    -12345'
>>> f'{-12345:+010}'
'-000012345'
```

Python Lists

Defining

```
>>> li1 = []
>>> li1
[]
>>> li2 = [4, 5, 6]
>>> li2
[4, 5, 6]
>>> li3 = list((1, 2, 3))
>>> li3
[1, 2, 3]
>>> li4 = list(range(1, 11))
>>> li4
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Generate

```
>>> list(filter(lambda x : x % 2 == 1, range(1, 20)))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

>>> [x ** 2 for x in range (1, 11) if  x % 2 == 1]
[1, 9, 25, 49, 81]

>>> [x for x in [3, 4, 5, 6, 7] if x > 5]
[6, 7]

>>> list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))
[6, 7]
```

Append

```
>>> li = []
>>> li.append(1)
>>> li
[1]
>>> li.append(2)
>>> li
[1, 2]
>>> li.append(4)
```

```
>>> li
[1, 2, 4]
>>> li.append(3)
>>> li
[1, 2, 4, 3]
```

List Slicing

Syntax of list slicing:

```
a_list[start:end]
a_list[start:end:step]
```

Slicing

```
>>> a = ['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[2:5]
['bacon', 'tomato', 'ham']
>>> a[-5:-2]
['egg', 'bacon', 'tomato']
>>> a[1:4]
['egg', 'bacon', 'tomato']
```

Omitting index

```
>>> a[:4]
['spam', 'egg', 'bacon', 'tomato']
>>> a[0:4]
['spam', 'egg', 'bacon', 'tomato']
>>> a[2:]
['bacon', 'tomato', 'ham', 'lobster']
>>> a[2:len(a)]
['bacon', 'tomato', 'ham', 'lobster']
>>> a
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[:]
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
```

With a stride

```
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[0:6:2]
['spam', 'bacon', 'ham']
>>> a[1:6:2]
['egg', 'tomato', 'lobster']
>>> a[6:0:-2]
```

```
['lobster', 'tomato', 'egg']
>>> a
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a[::-1]
['lobster', 'ham', 'tomato', 'bacon', 'egg', 'spam']
```

Remove

```
>>> li = ['bread', 'butter', 'milk']
>>> li.pop()
'milk'
>>> li
['bread', 'butter']
>>> del li[0]
>>> li
['butter']
```

Access

```
>>> li = ['a', 'b', 'c', 'd']
>>> li[0]
'a'
>>> li[-1]
'd'
>>> li[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Concatenating

```
>>> odd = [1, 3, 5]
>>> odd.extend([9, 11, 13])
>>> odd
[1, 3, 5, 9, 11, 13]
>>> odd = [1, 3, 5]
>>> odd + [9, 11, 13]
[1, 3, 5, 9, 11, 13]
```

Sort & Reverse

```
>>> li = [3, 1, 3, 2, 5]
>>> li.sort()
>>> li
[1, 2, 3, 3, 5]
```

```
>>> li.reverse()
>>> li
[5, 3, 3, 2, 1]
```

Count

```
>>> li = [3, 1, 3, 2, 5]
>>> li.count(3)
2
```

Repeating

```
>>> li = ["re"] * 3
>>> li
['re', 're', 're']
```

Python Flow control

Basic

```
num = 5
if num > 10:
    print("num is totally bigger than 10.")
elif num < 10:
    print("num is smaller than 10.")
else:
    print("num is indeed 10.")
```

One line

```
>>> a = 330
>>> b = 200
>>> r = "a" if a > b else "b"
>>> print(r)
a
```

else if

```
value = True
if not value:
```

```
    print("Value is False")
elif value is None:
    print("Value is None")
else:
    print("Value is True")
```

match case

```
x = 1
match x:
  case 0:
    print("zero")
  case 1:
    print("one")
  case _:
    print("multiple")
```

Python Loops

Basic

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

Prints: 2 3 5 7

With index

```
animals = ["dog", "cat", "mouse"]
# enumerate() adds counter to an iterable
for i, value in enumerate(animals):
    print(i, value)
```

Prints: 0 dog 1 cat 2 mouse

While

```
x = 0
while x < 4:
    print(x)
```

```
    x += 1  # Shorthand for x = x + 1
```

Prints: 0 1 2 3

Break

```
x = 0
for index in range(10):
    x = index * 10
    if index == 5:
        break
    print(x)
```

Prints: 0 10 20 30 40

Continue

```
for index in range(3, 8):
    x = index * 10
    if index == 5:
        continue
    print(x)
```

Prints: 30 40 60 70

Range

```
for i in range(4):
    print(i) # Prints: 0 1 2 3

for i in range(4, 8):
    print(i) # Prints: 4 5 6 7

for i in range(4, 10, 2):
    print(i) # Prints: 4 6 8
```

With zip()

```
words = ['Mon', 'Tue', 'Wed']
nums = [1, 2, 3]
# Use zip to pack into a tuple list
for w, n in zip(words, nums):
    print('%d:%s, ' %(n, w))
```

Prints: 1:Mon, 2:Tue, 3:Wed,

for/else

```
nums = [60, 70, 30, 110, 90]
for n in nums:
    if n > 100:
        print("%d is bigger than 100" %n)
        break
else:
    print("Not found!")
```

Also see: Python Tips

Python Functions

Basic

```
def hello_world():
    print('Hello, World!')
```

Return

```
def add(x, y):
    print("x is %s, y is %s" %(x, y))
    return x + y

add(5, 6)    # => 11
```

Positional arguments

```
def varargs(*args):
    return args

varargs(1, 2, 3)  # => (1, 2, 3)
```

Type of "args" is tuple.

Keyword arguments

```
def keyword_args(**kwargs):
```

```
    return kwargs

# => {"big": "foot", "loch": "ness"}
keyword_args(big="foot", loch="ness")
```

Type of "kwargs" is dict.

Returning multiple

```
def swap(x, y):
    return y, x

x = 1
y = 2
x, y = swap(x, y)  # => x = 2, y = 1
```

Default Value

```
def add(x, y=10):
    return x + y

add(5)      # => 15
add(5, 20)  # => 25
```

Anonymous functions

```
# => True
(lambda x: x > 2)(3)

# => 5
(lambda x, y: x ** 2 + y ** 2)(2, 1)
```

Python Modules

Import modules

```
import math
print(math.sqrt(16))  # => 4.0
```

From a module

```
from math import ceil, floor
print(ceil(3.7))   # => 4.0
print(floor(3.7))  # => 3.0
```

Import all

```
from math import *
```

Shorten module

```
import math as m

# => True
math.sqrt(16) == m.sqrt(16)
```

Functions and attributes

```
import math
dir(math)
```

Python File Handling

Read file

Line by line

```
with open("myfile.txt") as file:
    for line in file:
        print(line)
```

With line number

```
file = open('myfile.txt', 'r')
for i, line in enumerate(file, start=1):
    print("Number %s: %s" % (i, line))
```

String

Write a string

```
contents = {"aa": 12, "bb": 21}
```

```
with open("myfile1.txt", "w+") as file:
    file.write(str(contents))
```

Read a string

```
with open('myfile1.txt', "r+") as file:
    contents = file.read()
print(contents)
```

Object

Write an object

```
contents = {"aa": 12, "bb": 21}
with open("myfile2.txt", "w+") as file:
    file.write(json.dumps(contents))
```

Read an object

```
with open('myfile2.txt', "r+") as file:
    contents = json.load(file)
print(contents)
```

Delete a File

```
import os
os.remove("myfile.txt")
```

Check and Delete

```
import os
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
else:
    print("The file does not exist")
```

Delete Folder

```
import os
os.rmdir("myfolder")
```

Python Classes & Inheritance

Defining

```
class MyNewClass:
    pass

# Class Instantiation
my = MyNewClass()
```

Constructors

```
class Animal:
    def __init__(self, voice):
        self.voice = voice

cat = Animal('Meow')
print(cat.voice)    # => Meow

dog = Animal('Woof')
print(dog.voice)    # => Woof
```

Method

```
class Dog:

    # Method of the class
    def bark(self):
        print("Ham-Ham")

charlie = Dog()
charlie.bark()   # => "Ham-Ham"
```

Class Variables

```
class MyClass:
    class_variable = "A class variable!"

# => A class variable!
print(MyClass.class_variable)

x = MyClass()

# => A class variable!
print(x.class_variable)
```

Super() Function

```
class ParentClass:
    def print_test(self):
        print("Parent Method")

class ChildClass(ParentClass):
    def print_test(self):
        print("Child Method")
        # Calls the parent's print_test()
        super().print_test()
```

```
>>> child_instance = ChildClass()
>>> child_instance.print_test()
Child Method
Parent Method
```

repr() method

```
class Employee:
    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return self.name

john = Employee('John')
print(john)  # => John
```

User-defined exceptions

```
class CustomError(Exception):
    pass
```

Polymorphism

```
class ParentClass:
    def print_self(self):
        print('A')

class ChildClass(ParentClass):
    def print_self(self):
        print('B')
```

```
obj_A = ParentClass()
obj_B = ChildClass()

obj_A.print_self() # => A
obj_B.print_self() # => B
```

Overriding

```
class ParentClass:
    def print_self(self):
        print("Parent")

class ChildClass(ParentClass):
    def print_self(self):
        print("Child")

child_instance = ChildClass()
child_instance.print_self() # => Child
```

Inheritance

```
class Animal:
    def __init__(self, name, legs):
        self.name = name
        self.legs = legs

class Dog(Animal):
    def sound(self):
        print("Woof!")

Yoki = Dog("Yoki", 4)
print(Yoki.name) # => YOKI
print(Yoki.legs) # => 4
Yoki.sound()     # => Woof!
```

Python Type Hints (Since Python 3.5)

Variable & Parameter

```
string: str = "ha"
```

```
times: int = 3


# wrong hit, but run correctly
result: str = 1 + 2
print(result)  # => 3


def say(name: str, start: str = "Hi"):
    return start + ", " + name

print(say("Python"))  # => Hi, Python
```

Built-in date type

```
from typing import Dict, Tuple, List

bill: Dict[str, float] = {
    "apple": 3.14,
    "watermelon": 15.92,
    "pineapple": 6.53,
}
completed: Tuple[str] = ("DONE",)
succeeded: Tuple[int, str] = (1, "SUCCESS")
statuses: Tuple[str, ...] = (
    "DONE", "SUCCESS", "FAILED", "ERROR",
)
codes: List[int] = (0, 1, -1, -2)
```

Built-in date type (3.10+)

```
bill: dict[str, float] = {
    "apple": 3.14,
    "watermelon": 15.92,
    "pineapple": 6.53,
}
completed: tuple[str] = ("DONE",)
succeeded: tuple[int, str] = (1, "SUCCESS")
statuses: tuple[str, ...] = (
    "DONE", "SUCCESS", "FAILED", "ERROR",
)
codes: list[int] = (0, 1, -1, -2)
```

Positional argument

```
def calc_summary(*args: int):
    return sum(args)

print(calc_summary(3, 1, 4))  # => 8
```

Indicate all arguments' type is int.

Returned

```
def say_hello(name) -> str:
    return "Hello, " + name

var = "Python"
print(say_hello(var))  # => Hello, Python
```

Union returned

```
from typing import Union

def resp200(meaningful) -> Union[int, str]:
    return "OK" if meaningful else 200
```

Means returned value type may be int or str.

Keyword argument

```
def calc_summary(**kwargs: int):
    return sum(kwargs.values())

print(calc_summary(a=1, b=2))  # => 3
```

Indicate all parameters' value type is int.

Multiple returns

```
def resp200() -> (int, str):
    return 200, "OK"

returns = resp200()
print(returns)  # => (200, 'OK')
print(type(returns))  # tuple
```

Union returned (3.10+)

```
def resp200(meaningful) -> int | str:
    return "OK" if meaningful else 200
```

Since Python 3.10

Property

```
class Employee:
    name: str
    age: int

    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.graduated: bool = False
```

Self instance

```
class Employee:
    name: str

    def set_name(self, name) -> "Employee":
        self.name = name
        return self

    def copy(self) -> 'Employee':
        return type(self)(self.name)
```

Self instance (3.11+)

```
from typing import Self

class Employee:
    name: str
    age: int

    def set_name(self: Self, name) -> Self:
        self.name = name
        return self
```

Type & Generic

```
from typing import TypeVar, Type

T = TypeVar("T")
```

```
# "mapper" is a type, like int, str, MyClass and so on.
# "default" is an instance of type T, such as 314, "string", MyClass() and so on.
# returned is an instance of type T too.
def converter(raw, mapper: Type[T], default: T) -> T:
    try:
        return mapper(raw)
    except:
        return default

raw: str = input("Enter an integer: ")
result: int = converter(raw, mapper=int, default=0)
```

Function

```
from typing import TypeVar, Callable, Any

T = TypeVar("T")

def converter(raw, mapper: Callable[[Any], T], default: T) -> T:
    try:
        return mapper(raw)
    except:
        return default

# Callable[[Any], ReturnType] means a function declare like:
# def func(arg: Any) -> ReturnType:
#     pass

# Callable[[str, int], ReturnType] means a function declare like:
# def func(string: str, times: int) -> ReturnType:
#     pass

# Callable[..., ReturnType] means a function declare like:
# def func(*args, **kwargs) -> ReturnType:
#     pass

def is_success(value) -> bool:
    return value in (0, "OK", True, "success")

resp = dict(code=0, message="OK", data=[])
successed: bool = converter(resp["message"], mapper=is_success, default=False)
```

Miscellaneous

Comments

```
# This is a single line comments.


""" Multiline strings can be written
    using three "s, and are often used
    as documentation.
"""


''' Multiline strings can be written
    using three 's, and are often used
    as documentation.
'''
```

Generators

```
def double_numbers(iterable):
    for i in iterable:
        yield i + i
```

Generators help you make lazy code.

Generator to list

```
values = (-x for x in [1,2,3,4,5])
gen_to_list = list(values)

# => [-1, -2, -3, -4, -5]
print(gen_to_list)
```

Handle exceptions

```
try:
    # Use "raise" to raise an error
    raise IndexError("This is an index error")
except IndexError as e:
    pass                # Pass is just a no-op. Usually you would do recovery her
except (TypeError, NameError):
    pass                # Multiple exceptions can be handled together, if require
else:                   # Optional clause to the try/except block. Must follow al
```

```
    print("All good!")   # Runs only if the code in try raises no exceptions
finally:                 # Execute under all circumstances
    print("We can clean up resources here")
```

Related Cheatsheet

- Hook Cheatsheet
  Quick Reference
- Ruby Cheatsheet
  Quick Reference
- Awk Cheatsheet
  Quick Reference
- Bash Cheatsheet
  Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet
  Quick Reference
- Arduino Programming Cheatsheet
  Quick Reference
- HTML Canvas Cheatsheet
  Quick Reference
- TypeScript Cheatsheet
  Quick Reference

Numpy

NumPy is the fundamental package for scientific computing with Python. This cheat sheet is a quick reference for NumPy beginners.

Getting Started

Introduction

You'll also need to import numpy to get started:

```
import numpy as np
```

Importing/exporting

`np.loadtxt('file.txt')`	From a text file
`np.genfromtxt('file.csv',delimiter=',')`	From a CSV file
`np.savetxt('file.txt',arr,delimiter=' ')`	Writes to a text file
`np.savetxt('file.csv',arr,delimiter=',')`	Writes to a CSV file

Creating Arrays

`np.array([1,2,3])`	One dimensional array
`np.array([(1,2,3),(4,5,6)])`	Two dimensional array
`np.zeros(3)`	1D array of length 3 all values 0
`np.ones((3,4))`	3x4 array with all values 1
`np.eye(5)`	5x5 array of 0 with 1 on diagonal (Identity matrix)
`np.linspace(0,100,6)`	Array of 6 evenly divided values from 0 to 100

`np.arange(0,10,3)`	Array of values from 0 to less than 10 with step 3 (eg [0,3,6,9])
`np.full((2,3),8)`	2x3 array with all values 8
`np.random.rand(4,5)`	4x5 array of random floats between 0–1
`np.random.rand(6,7)*100`	6x7 array of random floats between 0–100
`np.random.randint(5,size=(2,3))`	2x3 array with random ints between 0–4

Inspecting Properties

`arr.size`	Returns number of elements in arr
`arr.shape`	Returns dimensions of arr (rows,columns)
`arr.dtype`	Returns type of elements in arr
`arr.astype(dtype)`	Convert arr elements to type dtype
`arr.tolist()`	Convert arr to a Python list
`np.info(np.eye)`	View documentation for np.eye

Copying/sorting/reshaping

`np.copy(arr)`	Copies arr to new memory
`arr.view(dtype)`	Creates view of arr elements with type dtype
`arr.sort()`	Sorts arr
`arr.sort(axis=0)`	Sorts specific axis of arr
`two_d_arr.flatten()`	Flattens 2D array two_d_arr to 1D
`arr.T`	Transposes arr (rows become columns and vice versa)
`arr.reshape(3,4)`	Reshapes arr to 3 rows, 4 columns without changing data
`arr.resize((5,6))`	Changes arr shape to 5x6 and fills new values with 0

Adding/removing Elements

`np.append(arr,values)`	Appends values to end of arr
`np.insert(arr,2,values)`	Inserts values into arr before index 2

`np.delete(arr,3,axis=0)`	Deletes row on index 3 of arr
`np.delete(arr,4,axis=1)`	Deletes column on index 4 of arr

Combining/splitting

`np.concatenate((arr1,arr2),axis=0)`	Adds arr2 as rows to the end of arr1
`np.concatenate((arr1,arr2),axis=1)`	Adds arr2 as columns to end of arr1
`np.split(arr,3)`	Splits arr into 3 sub-arrays
`np.hsplit(arr,5)`	Splits arr horizontally on the 5th index

Indexing/slicing/subsetting

`arr[5]`	Returns the element at index 5
`arr[2,5]`	Returns the 2D array element on index [2][5]
`arr[1]=4`	Assigns array element on index 1 the value 4
`arr[1,3]=10`	Assigns array element on index [1][3] the value 10
`arr[0:3]`	Returns the elements at indices 0,1,2 (On a 2D array: returns rows 0,1,2)
`arr[0:3,4]`	Returns the elements on rows 0,1,2 at column 4
`arr[:2]`	Returns the elements at indices 0,1 (On a 2D array: returns rows 0,1)
`arr[:,1]`	Returns the elements at index 1 on all rows
`arr<5`	Returns an array with boolean values
`(arr1<3) & (arr2>5)`	Returns an array with boolean values
`~arr`	Inverts a boolean array
`arr[arr<5]`	Returns array elements smaller than 5

Vector Math

`np.add(arr1,arr2)`	Elementwise add arr2 to arr1
`np.subtract(arr1,arr2)`	Elementwise subtract arr2 from arr1
`np.multiply(arr1,arr2)`	Elementwise multiply arr1 by arr2
`np.divide(arr1,arr2)`	Elementwise divide arr1 by arr2

`np.power(arr1,arr2)`	Elementwise raise arr1 raised to the power of arr2
`np.array_equal(arr1,arr2)`	Returns True if the arrays have the same elements and shape
`np.sqrt(arr)`	Square root of each element in the array
`np.sin(arr)`	Sine of each element in the array
`np.log(arr)`	Natural log of each element in the array
`np.abs(arr)`	Absolute value of each element in the array
`np.ceil(arr)`	Rounds up to the nearest int
`np.floor(arr)`	Rounds down to the nearest int
`np.round(arr)`	Rounds to the nearest int

Scalar Math

`np.add(arr,1)`	Add 1 to each array element
`np.subtract(arr,2)`	Subtract 2 from each array element
`np.multiply(arr,3)`	Multiply each array element by 3
`np.divide(arr,4)`	Divide each array element by 4 (returns np.nan for division by zero)
`np.power(arr,5)`	Raise each array element to the 5th power

Statistics

`np.mean(arr,axis=0)`	Returns mean along specific axis
`arr.sum()`	Returns sum of arr
`arr.min()`	Returns minimum value of arr
`arr.max(axis=0)`	Returns maximum value of specific axis
`np.var(arr)`	Returns the variance of array
`np.std(arr,axis=1)`	Returns the standard deviation of specific axis
`arr.corrcoef()`	Returns correlation coefficient of array

Top Cheatsheet

- Python Cheatsheet
  Quick Reference
- Vim Cheatsheet
  Quick Reference
- JavaScript Cheatsheet
  Quick Reference
- Bash Cheatsheet
  Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet
  Quick Reference
- Arduino Programming Cheatsheet
  Quick Reference
- HTML Canvas Cheatsheet
  Quick Reference
- TypeScript Cheatsheet
  Quick Reference

HTTP Status Code

The HTTP status codes cheat sheet. A quick reference to every HTTP status code.

HTTP Status code

Means
• 1xx: Informational
• 2xx: Success
• 3xx: Redirection
• 4xx: Client Error
• 5xx: Server Error

2xx. Successful
• 200: OK
• 201: Created
• 202: Accepted
• 203: Non-Authoritative Information
• 204: No Content
• 205: Reset Content
• 206: Partial Content
• 207: Multi-Status
• 208: Already Reported

- 226: IM Used

4xx. Client Error

- 400: Bad Request
- 401: Unauthorized
- 402: Payment Required
- 403: Forbidden
- 404: Not Found
- 405: Method Not Allowed
- 406: Not Acceptable
- 407: Proxy Authentication Required
- 408: Request Timeout
- 409: Conflict
- 410: Gone
- 411: Length Required
- 412: Precondition Failed
- 413: Payload Too Large
- 414: URI Too Long
- 415: Unsupported Media Type
- 416: Range Not Satisfiable
- 417: Expectation Failed
- 421: Misdirected Request
- 426: Upgrade Required
- 428: Precondition Required
- 429: Too Many Requests
- 431: Request Header Fields Too Large

- 451: Unavailable For Legal Reasons

1xx. Information

- 100: Continue
- 101: Switching Protocols
- 102: Processing
- 103: Early Hints

3xx. Redirection

- 300: Multiple Choices
- 301: Moved Permanently
- 302: Found
- 303: See Other
- 304: Not Modified
- 305: Use Proxy
- 306: Unused
- 307: Temporary Redirect
- 308: Permanent Redirect

5xx. Server Error

- 500: Internal Server Error
- 501: Not Implemented
- 502: Bad Gateway
- 503: Service Unavailable
- 504: Gateway Timeout
- 505: HTTP Version Not Supported
- 506: Variant Also Negotiates

- 507: Insufficient Storage
- 508: Loop Detected
- 510: Not Extended
- 511: Network Authentication Required

Top Cheatsheet

Python Cheatsheet
Quick Reference

Vim Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Bash Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

ASCII Code

This cheatsheet is a complete list of ASCII Code Table with their numbers and names.

ASCII Code Table

ASCII Code Table

Symbol	Dec	Oct	Hex	Bin
NUL (Null)	0	0	0	0
SOH (Start of Heading)	1	1	1	1
STX (Start of Text)	2	2	2	10
ETX (End of Text)	3	3	3	11
EOT (End of Transmission)	4	4	4	100
ENQ (Enquiry)	5	5	5	101
ACK (Acknowledgment)	6	6	6	110
BEL (Bell)	7	7	7	111
BS (Back Space)	8	10	8	1000
HT (Horizontal Tab)	9	11	9	1001
LF (Line Feed)	10	12	0A	1010
VT (Vertical Tab)	11	13	0B	1011
FF (Form Feed)	12	14	0C	1100
CR (Carriage Return)	13	15	0D	1101
SO (Shift Out / X-On)	14	16	0E	1110

Symbol	Dec	Oct	Hex	Bin
SI (Shift In / X-Off)	15	17	0F	1111
DLE (Data Line Escape)	16	20	10	10000
DC1 (Device Control 1 / oft. XON)	17	21	11	10001
DC2 (Device Control 2)	18	22	12	10010
DC3 (Device Control 3 / oft. XOFF)	19	23	13	10011
DC4 (Device Control 4)	20	24	14	10100
NAK (Negative Acknowledgement)	21	25	15	10101
SYN (Synchronous Idle)	22	26	16	10110
ETB (End of Transmit Block)	23	27	17	10111
CAN (Cancel)	24	30	18	11000
EM (End of Medium)	25	31	19	11001
SUB (Substitute)	26	32	1A	11010
ESC (Escape)	27	33	1B	11011
FS (File Separator)	28	34	1C	11100
GS (Group Separator)	29	35	1D	11101
RS (Record Separator)	30	36	1E	11110
US (Unit Separator)	31	37	1F	11111
(Space)	32	40	20	100000
!	33	41	21	100001
"	34	42	22	100010
#	35	43	23	100011
$	36	44	24	100100
%	37	45	25	100101
&	38	46	26	100110

Symbol	Dec	Oct	Hex	Bin
'	39	47	27	100111
(	40	50	28	101000
)	41	51	29	101001
*	42	52	2A	101010
+	43	53	2B	101011
,	44	54	2C	101100
-	45	55	2D	101101
.	46	56	2E	101110
/	47	57	2F	101111
0	48	60	30	110000
1	49	61	31	110001
2	50	62	32	110010
3	51	63	33	110011
4	52	64	34	110100
5	53	65	35	110101
6	54	66	36	110110
7	55	67	37	110111
8	56	70	38	111000
9	57	71	39	111001
:	58	72	3A	111010
;	59	73	3B	111011
<	60	74	3C	111100
=	61	75	3D	111101
>	62	76	3E	111110

Symbol	Dec	Oct	Hex	Bin
?	63	77	3F	111111
@	64	100	40	1000000
A	65	101	41	1000001
B	66	102	42	1000010
C	67	103	43	1000011
D	68	104	44	1000100
E	69	105	45	1000101
F	70	106	46	1000110
G	71	107	47	1000111
H	72	110	48	1001000
I	73	111	49	1001001
J	74	112	4A	1001010
K	75	113	4B	1001011
L	76	114	4C	1001100
M	77	115	4D	1001101
N	78	116	4E	1001110
O	79	117	4F	1001111
P	80	120	50	1010000
Q	81	121	51	1010001
R	82	122	52	1010010
S	83	123	53	1010011
T	84	124	54	1010100
U	85	125	55	1010101
V	86	126	56	1010110

Symbol	Dec	Oct	Hex	Bin
W	87	127	57	1010111
X	88	130	58	1011000
Y	89	131	59	1011001
Z	90	132	5A	1011010
[	91	133	5B	1011011
\	92	134	5C	1011100
]	93	135	5D	1011101
^	94	136	5E	1011110
_	95	137	5F	1011111
`	96	140	60	1100000
a	97	141	61	1100001
b	98	142	62	1100010
c	99	143	63	1100011
d	100	144	64	1100100
e	101	145	65	1100101
f	102	146	66	1100110
g	103	147	67	1100111
h	104	150	68	1101000
i	105	151	69	1101001
j	106	152	6A	1101010
k	107	153	6B	1101011
l	108	154	6C	1101100
m	109	155	6D	1101101
n	110	156	6E	1101110

Symbol	Dec	Oct	Hex	Bin
o	111	157	6F	1101111
p	112	160	70	1110000
q	113	161	71	1110001
r	114	162	72	1110010
s	115	163	73	1110011
t	116	164	74	1110100
u	117	165	75	1110101
v	118	166	76	1110110
w	119	167	77	1110111
x	120	170	78	1111000
y	121	171	79	1111001
z	122	172	7A	1111010
{	123	173	7B	1111011
	124	174	7C	1111100
}	125	175	7D	1111101
~	126	176	7E	1111110
(Delete)	127	177	7F	1111111
€	128	200	80	10000000
	129	201	81	10000001
‚	130	202	82	10000010
ƒ	131	203	83	10000011
„	132	204	84	10000100
…	133	205	85	10000101
†	134	206	86	10000110

Symbol	Dec	Oct	Hex	Bin
‡	135	207	87	10000111
ˆ	136	210	88	10001000
‰	137	211	89	10001001
Š	138	212	8A	10001010
‹	139	213	8B	10001011
Œ	140	214	8C	10001100
	141	215	8D	10001101
Ž	142	216	8E	10001110
	143	217	8F	10001111
	144	220	90	10010000
‘	145	221	91	10010001
’	146	222	92	10010010
“	147	223	93	10010011
”	148	224	94	10010100
•	149	225	95	10010101
–	150	226	96	10010110
—	151	227	97	10010111
˜	152	230	98	10011000
™	153	231	99	10011001
š	154	232	9A	10011010
›	155	233	9B	10011011
œ	156	234	9C	10011100
	157	235	9D	10011101
ž	158	236	9E	10011110

Symbol	Dec	Oct	Hex	Bin
Ÿ	159	237	9F	10011111
(Non-breaking space)	160	240	A0	10100000
¡	161	241	A1	10100001
¢	162	242	A2	10100010
£	163	243	A3	10100011
¤	164	244	A4	10100100
¥	165	245	A5	10100101
¦	166	246	A6	10100110
§	167	247	A7	10100111
¨	168	250	A8	10101000
©	169	251	A9	10101001
ª	170	252	AA	10101010
«	171	253	AB	10101011
¬	172	254	AC	10101100
(Soft hyphen)	173	255	AD	10101101
®	174	256	AE	10101110
¯	175	257	AF	10101111
°	176	260	B0	10110000
±	177	261	B1	10110001
²	178	262	B2	10110010
³	179	263	B3	10110011
´	180	264	B4	10110100
µ	181	265	B5	10110101
¶	182	266	B6	10110110

Symbol	Dec	Oct	Hex	Bin
·	183	267	B7	10110111
¸	184	270	B8	10111000
¹	185	271	B9	10111001
º	186	272	BA	10111010
»	187	273	BB	10111011
¼	188	274	BC	10111100
½	189	275	BD	10111101
¾	190	276	BE	10111110
¿	191	277	BF	10111111
À	192	300	C0	11000000
Á	193	301	C1	11000001
Â	194	302	C2	11000010
Ã	195	303	C3	11000011
Ä	196	304	C4	11000100
Å	197	305	C5	11000101
Æ	198	306	C6	11000110
Ç	199	307	C7	11000111
È	200	310	C8	11001000
É	201	311	C9	11001001
Ê	202	312	CA	11001010
Ë	203	313	CB	11001011
Ì	204	314	CC	11001100
Í	205	315	CD	11001101
Î	206	316	CE	11001110

Symbol	Dec	Oct	Hex	Bin
Ï	207	317	CF	11001111
Ð	208	320	D0	11010000
Ñ	209	321	D1	11010001
Ò	210	322	D2	11010010
Ó	211	323	D3	11010011
Ô	212	324	D4	11010100
Õ	213	325	D5	11010101
Ö	214	326	D6	11010110
×	215	327	D7	11010111
Ø	216	330	D8	11011000
Ù	217	331	D9	11011001
Ú	218	332	DA	11011010
Û	219	333	DB	11011011
Ü	220	334	DC	11011100
Ý	221	335	DD	11011101
Þ	222	336	DE	11011110
ß	223	337	DF	11011111
à	224	340	E0	11100000
á	225	341	E1	11100001
â	226	342	E2	11100010
ã	227	343	E3	11100011
ä	228	344	E4	11100100
å	229	345	E5	11100101
æ	230	346	E6	11100110

Symbol	Dec	Oct	Hex	Bin
ç	231	347	E7	11100111
è	232	350	E8	11101000
é	233	351	E9	11101001
ê	234	352	EA	11101010
ë	235	353	EB	11101011
ì	236	354	EC	11101100
í	237	355	ED	11101101
î	238	356	EE	11101110
ï	239	357	EF	11101111
ð	240	360	F0	11110000
ñ	241	361	F1	11110001
ò	242	362	F2	11110010
ó	243	363	F3	11110011
ô	244	364	F4	11110100
õ	245	365	F5	11110101
ö	246	366	F6	11110110
÷	247	367	F7	11110111
ø	248	370	F8	11111000
ù	249	371	F9	11111001
ú	250	372	FA	11111010
û	251	373	FB	11111011
ü	252	374	FC	11111100
ý	253	375	FD	11111101
þ	254	376	FE	11111110

Symbol	Dec	Oct	Hex	Bin
ÿ	255	377	FF	11111111

Related Cheatsheet

HTML Characters Entities Cheatsheet
Quick Reference

ISO 639-1 Language Code Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Google Search

This quick reference cheat sheet lists of Google advanced search operators.

Getting Started

Google Advanced Search Operators		
""	Allows searching for a specific phrase - exact match search. Individual word prevents synonyms	Basic,Mail
OR/AND	Boolean search function for OR searches as Google defaults to AND between words - must be all caps	Basic,Mail
\	Implements OR	Basic
()	Allows grouping of operators and helps dictate order	Basic,Mail
-	Excludes a word from results	Basic,Mail
*	Acts as a wildcard and will match any word or phrase	Basic
#..#	# represents a number in this instance. Use to find numbers in a series.	Basic
$	Allows for search of USD	Basic
€	Allows for search of Euro	Basic
in	Allows searches for unit conversion (currency, unit or measure)	Basic
~	Prefix - Include synonyms (potentially defunct)	Basic
+	Prefix - Force exact match on single phrase	Basic,Mail
AROUND(X)	This is sandwiched between two words and the X declares how many words they must be mentioned between. I.e. if it's (4) then the two keywords must be mentioned within 4 words of each other.	Advanced
_	Acts as wildcard for autocomplete	Advanced

Search with url

`inurl:`	Only returns results where the queried keyword(s) is present in the URL	Advanced
`allinurl:`	As above but only containing all of the specified words in the URL	Advanced
`blogurl:`	Find blog URLs under a specific domain. This was used in Google blog search, but I've found it does return some results in regular search.	Advanced
`site:`	Limit results to those from one site	Advanced
`related:`	Find similar domains to the queried domain	Advanced

Search with dates

`daterange:`	Return results in a specified range (requires julian dates)	Advanced
`after:`	Allows you to search drive or mail for files modified or mail sent/received anytime after a set date	Drive,Mail
`before:`	Allows you to search drive or mail for files modified or mail sent/received before a certain date	Drive,Mail
`older:`	Search for messages older than a certain date	Mail
`newer:`	Search for messages newer than a certain date	Mail

Search files

`filename:`	Search for messages with a particular type of file attached, or the exact name of a file	Mail
`type:`	Allows you to search drive by file type	Drive
`owner:`	Allows you to search drive by owner of file or folder	Drive
`to:`	Allows you to search drive for files shared with a specific person	Drive
`title:`	Searches drive for files with the keyword in their title alone	Drive
`source:domain`	Allows you to search for files or folders shared with everyone in your business	Drive
`filetype:`	Returns only files of a particular type associated with the keyword searched	Advanced
`ext:`	As above, based on extension	Advanced

`after:`	Allows you to search drive or mail for files modified or mail sent/received anytime after a set date	Drive,Mail
`before:`	Allows you to search drive or mail for files modified or mail sent/received before a certain date	Drive,Mail
`is:trashed`	Searches for the item in the Drive bin	Drive

Search with page content

`link:`	Find pages that link to the target domain	Advanced
`inanchor:`	Find pages linked to with the specified anchor text/ phrase. Data is heavily sampled.	Advanced
`allinanchor:`	Find pages with all individual terms after "inanchor:" in the inbound anchor text.	Advanced
`intitle:`	Returns pages based on the searched query appearing in their title	Advanced
`allintitle:`	Similar to intitle: but only returns titles where all the words in the title match	Advanced
`inposttile:`	Finds pages with keywords in their post titles (i.e. for researching blogs)	
`intext:`	Finds pages where the keyword(s) are mentioned within the page content.	Advanced
`allintext:`	Similar to “intext,” but only results containing all of the specified words somewhere on the page will be returned.	Advanced

Keywords

`Business`	type E.g. cafe, restaurant, bar etc will return a selection of appropriate businesses in the are	Maps
`Petrol/Charging`	Station EV near me or perol station near me returns	Maps
`Search`	for a message with a google sheet attached	Mail
`Search`	for a message with a google presentation attached	Mail

Search on emails

`+`	Prefix - Force exact match on single phrase	Basic,Mail
`()`	Allows grouping of operators and helps dictate order	Basic,Mail

-	Excludes a word from results	Basic,Mail
""	Allows searching for a specific phrase - exact match search. Individual word prevents synonyms	Basic,Mail
OR/AND	Boolean search function for OR searches as Google defaults to AND between words - must be all caps	Basic,Mail
after:	Allows you to search drive or mail for files modified or mail sent/received anytime after a set date	Drive,Mail
before:	Allows you to search drive or mail for files modified or mail sent/received before a certain date	Drive,Mail
is:starred	Searches only items that have been starred in drive	Drive,Mail
from:	Specify the sender in google mail	Mail
to:	Specify the recipient in google mail	Mail
cc:	Search by a recipient that was copied into an email	Mail
bcc:	Search by a recipient that was blind copied into an email	Mail
older:	Search for messages older than a certain date	Mail
newer:	Search for messages newer than a certain date	Mail
Search	for a message with a google sheet attached	Mail
Search	for a message with a google presentation attached	Mail
AROUND	Similar to the normal google search function, allows you to search for keywords near each other.	Mail
subject:	Search by keywords featured in the subject line	Mail
{}	Use for OR in mail instead of the OR function	Mail
label:	Search for messages that have a certain label	Mail
has:attachment	Search for messages that have an item attached	Mail
has:drive	Search for messages with a google drive attached	Mail
has:document	Search for messages with a google doc attached	Mail
has:youtube	Search for a message containing a youtube video	Mail
list:	Search for all messages from a particular mailing list	Mail

`in:anywhere`	Includes all folders in your search, including spam and bin	Mail
`is:important`	Search for messages that have been marked as important	Mail
`label:important`	Same as is:important	Mail
`is:snoozed`	Searches for messages that have been snoozed	Mail
`is:unread`	Searches for unread messages	Mail
`is:read`	searches for read messages only	Mail
`has:yellow-star`	Searches for messages with coloured star icon	Mail
`has:blue-info`	Searches for messages with colourd icon	Mail
`is:chat`	Searches for messagse from chat	Mail
`deliveredto:`	Search by email address for delivered messages	Mail
`category:`	Searches by messages based on category. Follow the colon with the categoy name, i.e. category:primary	Mail
`size:`	Messages larger than a certain size in bytes	Mail
`larger:`	Messages larger than a certain size in bytes	Mail
`smaller:`	Messages smaller than a certain size in bytes	Mail
`has:userlabels`	Search for messages that have custom user labels	Mail
	Search for messages that have no custom user labels	Mail

Some other useful search operators

`define:`	Pulls a card response from Google displaying the dictionary definition of the word or phrase	Advanced
`cache:`	Returns the most up to date cache of an indexed web page	Advanced
`weather:`	Brings up the featured snipped for weather for that location	Advanced
`stocks:`	Returns stock information for the specified ticker	Advanced
`map:`	Force google map results for a particular query	Advanced
`movie:`	Find information for the specified movie (particularly useful when that movie has an ambiguous name). If the movie is still in theatres it'll also return screen times	Advanced

`source:`	Use in google news, returns results from the specified source	Advanced
`loc:`	Returns results for a specific location	Advanced
`location:`	As above but with Google news	Advanced
`info:`	Returns information related to a domain (pages with domain text, similar on-site pages, cache etc)	Advanced
`near`	Part of the google maps lazy searches e.g. book shops near work	Maps

Also see

Google Search Operators Cheat Sheet (static.semrush.com)

IntelliJ IDEA

IntelliJ IDEA is a very good Java IDE, most of its commands have shortcuts to keep your hands from leaving the keyboard

IDEA Windows & Linux Keymap

Editing

Ctrl Space	Basic code completion
Ctrl Shift Space	Smart code completion
Ctrl Shift Enter	Complete statement
Ctrl P	Parameter info
Ctrl Q	Quick documentation lookup
Shift F1	External Doc
Ctrl hover	Brief Info
Ctrl F1	Error or warning at caret
Alt Insert	Generate code
Ctrl O	Override methods
Ctrl I	Implement methods
Ctrl Alt T	Surround with
Ctrl /	Comment or uncomment line
Ctrl Shift /	Comment or uncomment block

Ctrl W	Select successively increasing code blocks
Ctrl Shift W	Decrease current selection to previous state
Alt Q	Context info
Alt Enter	Show intention actions and quick-fixes
Ctrl Alt L	Reformat code
Ctrl Alt O	Optimize imports
Ctrl Alt I	Auto-indent line(s)
Tab	Indent selected lines
Shift Tab	Unindent selected lines
Ctrl X	Cut current line or selected block to clipboard
Ctrl C	Copy current line or selected block to clipboard
Ctrl V	Paste from clipboard
Ctrl Shift V	Paste from recent buffers
Ctrl D	Duplicate current line or selected block
Ctrl Y	Delete line at caret
Ctrl Shift J	Smart line join
Ctrl Enter	Smart line split
Shift Enter	Start new line
Ctrl Shift U	Toggle case for word at caret or selected block
Ctrl Shift] / [	Select till code block end/start
Ctrl Backspace	Delete to word end/start
Ctrl + / -	Expand/collapse code block
Ctrl Shift +	Expand all
Ctrl Shift -	Collapse all

Ctrl F4	Close active editor tab

Usage Search

Alt F7 / Ctrl F7	Find usages/Find usages in file
Ctrl Shift F7	Highlight usages in file
Ctrl Alt F7	Show usages

Navigation

Ctrl N	Go to class
Ctrl Shift N	Go to file
Ctrl Alt Shift N	Go to symbol
Alt Right / Left	Go to next / previous editor tab
F12	Go back to previous tool window
Esc	Go to editor
Shift Esc	Hide active or last active window
Ctrl Shift F4	Close active run, messages...
Ctrl G	Go to line
Ctrl E	Recent files popup
Ctrl Alt Left / Right	Navigate back / forward
Ctrl Shift Backspace	Navigate to last edit location
Alt F1	Select current file or symbol in any view
Ctrl B / Ctrl Click	Go to declaration
Ctrl Alt B	Go to implementation(s)
Ctrl Shift I	Open quick definition lookup
Ctrl Shift B	Go to type declaration
Ctrl U	Go to super-method / super-class

Alt Up / Down	Go to previous / next method
Ctrl]/[	Move to code block end/start
Ctrl F12	File structure popup
Ctrl H	Type hierarchy
Ctrl Shift H	Method hierarchy
Ctrl Alt H	Call hierarchy
F2 / Shift F2	Next/previous highlighted error
F4 / Ctrl Enter	Edit source / View source
Alt Home	Show navigation bar
F11	Toggle bookmark
Ctrl F11	Toggle bookmark with mnemonic
Ctrl 0...9	Go to numbered bookmark
Shift F11	Show bookmarks

Search/Replace

Double Shift	Search everywhere
Ctrl F	Find
F3 Shift F3	Find next / Find previous
Ctrl R	Replace
Ctrl Shift F	Find in path
Ctrl Shift R	Replace in path

Live Templates

Ctrl Alt J	Surround with Live Template
Ctrl J	Insert Live Template

iter	Iteration according to Java SDK 1.5 style
inst	Check object type with instanceof and downcast it
itco	Iterate elements of java.util.Collection
itit	Iterate elements of java.util.Iterator
itli	Iterate elements of java.util.List
psf	public static final
thr	throw new

Refactoring

F5	Copy
F6	Move
Alt Delete	Safe Delete
Shift F6	Rename
Ctrl F6	Change Signature
Ctrl Alt N	Inline
Ctrl Alt M	Extract Method
Ctrl Alt V	Extract Variable
Ctrl Alt F	Extract Field
Ctrl Alt C	Extract Constant
Ctrl Alt P	Extract Parameter

Debugging

F8/F7	Step over/Step into
Shift F7 / Shift F8	Smart step into/Step out
Alt F9	Run to cursor
Alt F8	Evaluate expression

F9	Resume program
Ctrl F8	Toggle breakpoint
Ctrl Shift F8	View breakpoints

Compile and Run

Ctrl F9	Make project
Ctrl Shift F9	Compile selected file, package or module
Alt Shift F10 / F9	Select configuration and run/and debug
Shift F10 / F9	Run/Debug
Ctrl Shift F10	Run context configuration from editor

VCS/Local History

Ctrl K	Commit project to VCS
Ctrl T	Update from VCS
Alt Shift C	View recent changes
Alt `	VCS Operations Popup

General

Alt 0...9	Open corresponding tool window
Ctrl S	Save all
Ctrl Alt Y	Synchronize
Ctrl Shift F12	Toggle maximizing editor
Alt Shift F	Add to Favorites
Alt Shift I	Inspect current file
Ctrl `	Quick switch current scheme
Ctrl Alt S	Open Settings dialog

Ctrl Alt Shift S	Open Project Structure dialog
Ctrl Shift A	Find Action
Ctrl Tab	Switch between tool and tabs

Related Cheatsheet

- WebStorm Cheatsheet Quick Reference
- Emacs Cheatsheet Quick Reference
- Vim Cheatsheet Quick Reference
- VSCode Cheatsheet Quick Reference

Recent Cheatsheet

- Gnome Desktop Cheatsheet Quick Reference
- Arduino Programming Cheatsheet Quick Reference
- HTML Canvas Cheatsheet Quick Reference
- TypeScript Cheatsheet Quick Reference