LEARN JAVASCRIPT

BEGINNERS EDITION

A Complete Beginner's Guide to Learn JavaScript

Suman Kunwar

सामग्री तालिका

अभिस्वीकृति

"Learn JavaScript" पुस्तकको सामग्रीलाई नेपालीमा अनुवाद गर्नमा महत्वपूर्ण भूमिका निर्वाह गर्नुहुने सबै योगदानकर्ताहरूलाई म हार्दिक धन्यवाद र कृतज्ञता व्यक्त गर्न चाहन्छु। तपाईंको प्रयास र समर्पणले यस बहुमूल्य स्रोतलाई फराकिलो दर्शकहरूको लागि पहुँचयोग्य बनाउनमा महत्त्वपूर्ण परिवर्तन ल्याएको छ। प्रोग्रामिंगको संसारलाई थप समावेशी बनाउनको लागि तपाईंको उत्कृष्ट योगदान र प्रतिबद्धताको लागि धन्यवाद।

योगदानकर्ताहरू

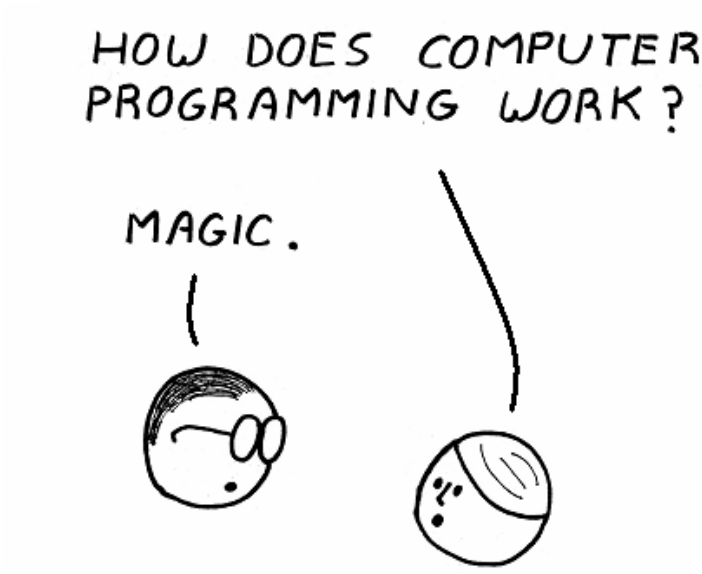
- अनुज खड्का
- रुपेश भण्डारी
- साहिल खड्का

परिचय

कम्प्युटर, आजको समयमा सबैले हरेक दिन सुनिरहने र प्रयोग गरी रहने साधन हो। यो विशेष गरी छिटो र सही रुपमा काम गर्न सक्ने क्षमताको करण प्रसिद्ध छ । कतिपय मनिसहरुको लागि त झन कम्प्युटर जीवन जिउने माध्यम नै बनिसकेको छ। यस्को बास्तविक परिचय त त्यति बेला हुन्छ जब हामी यसलाई हामीले सोच्न सक्ने सम्म हरेक ठाउँमा पाउछौँ। चाहे त्यो अस्पतालको गम्भीर कामको लागी होस या त घरमा मनोरञ्जनात्मक कार्यको लागी होस, यसको जाटिल कार्य र ठूला डाटा प्रक्रिया सेकेन्डमा नै गर्न सक्ने क्षमताले आज कम्प्यूटर हरेक गोजीहरुमा अटाएका छन्। तर यो कम्प्युटर आफैमा कसरी काम गर्छ? के तपाई यो सोच्दै हुनुहुन्छ? यो बुझ्नाको लागी कम्प्यूटरलाई हामी हाम्रो दिमागसँग तुलना गर्न सक्छौ।

कम्प्यूटर भनेको हाम्रो खोपडी जस्तै नै हो, र यसमा प्रयोग हुने विशेष चिप, “Central Chip” भनेको हाम्रो दिमाग जस्तै। यस चिप भित्र बिभिन्न कुराहरु सिकाइको हुन्छ। अझ स्पष्ट रूप भन्नु पर्दा कुनै पनि कामलाई step wise step हल गर्न ट्रेन गरियेको हुन्छ, हाम्रो दिमग जस्तै । हामीलाई पनि त सानो छदा खाना खान सिकाईन्थ्यो, लेख्न सिकाईन्थ्यो र अलि अलि सिकाइएको भरमा नै हामी आफैँ सिक्न खोज्थेम र अझै त्यो कार्य गर्न सिकदै जानथेम। उदाहरणका लागी खाना पकाउने बेला हामीलाई एउटा निश्चित निर्देशन दिएको हुन्छ कि पैला कुकर लिने, अनि त्यसमा पुग्ने गरी चामल हल्ने र त्यस्मा अडकलेर पानी थाप्ने, त्यस पछि ग्यास बालेर कुकर बसाल्ने र पाकी नसकेसम्म सीटी बजाउन दिनुहोस्। तर यसलाइ गर्ने अर्को नी तारिका हुन सक्छ, चमल र पानी ठिक गरेर इलेक्ट्रिक राइस कुकरमा हल्ने र पावर अन गर्दिने। हो येस्तै केही काम गर्ने set of instructions लाई नै हामी कम्प्यूटरको भाषामा प्रोग्राम (Program) भन्छौ ।

यो मध्धे कुनै पनि स्टेप बिग्रियो भने हामीले सोचेको जस्तै मिठो खाना नपक्न, सक्छ। त्यसै गरि program लेख्दा नी कुनै पनि ठाँउमा गल्ती भयो भने हामीले सोचेको जस्तो output न आउन सक्छ । र खानालाइ पनि प्रेसर कुकर र राइस कुकरमा पकाए त्यस्लाई अटोमेटिक या स्वादिलो बनाउन सकिने जस्तै कुनै पनि कम्प्यूटर प्रोग्रामलाई बिभिन्न तारिकाले लेखेर त्यस्को चल्ने गतिलाई बढाउने र त्यो प्रोग्राम चलौने मान्छेलाई अझ सजिलो बनाउने गर्न सकिन्छ। हो खाना पकाउ लग्ने सबै अनुभव कम्प्यूटर प्रोग्राम लेखनमा नि लाग्छ। सानो सानो काम गर्न अल्छी लगने, पकाउदै गर्दा स्वाद बिग्रिए रिस उठ्ने र स्वाद मिलाउने अर्को झनझट हुन तर साही ढंगाले गरे पाकी सक्दा स्वाद को आनन्द लिन छुटै नै मजा भए जस्तै program लेख्दा नी errorहरु फेला पार्न एकदम कठिन र ठकौ लाग्ने काम हुन सक्छ तर जब त्यो program एक छोटी पुरा भएर चल्न थाल्छ, त्यसको आनन्द नै छुटै हुन्छ।



प्रोग्रामलाई एउटा सफ्टवेयरले कम्प्युटरसँग अन्तर्क्रिया गर्ने तरिकाको रुपमा पनि लिन सकिन्छ। जसरी मानिसहरूसँग सञ्चार गर्न धेरै भाषाहरू छन्, कम्प्युटरसँग पनि आफ्नै सञ्चारको लागि विभिन्न भाषाहरू छन्। यस कृत्रिम भाषा जुन कम्प्युटरहरूलाई निर्देशन दिन प्रयोग गरिन्छ त्यसलाई Programming Language भनिन्छ। उदाहरण को लागी BASIC, FORTRAN, Javascript, Python, C, C++, Dart आदि। यी programming language धेरै लामो समयदेखि विकसित हुँदै आएको छ जसले कम्प्यूटर लगाएत मानिसहरूलाई पनि त्यो विशेष भाषा बुझ्न सजिलो बनाइरहेको छ। अहिले त पुराना भाषालाई निकै नै नयाँले प्रतिस्थापन गरिदिएको छ, जुन सिक्न सजिलो हुनुको साथ साथै विभिन्न क्षेत्रमा उपयोग गर्न सकिन्छ। यी सबै पुराना भाषाहरू मध्ये, जाभास्क्रिप्ट (Javascript) एक पुरानो तर प्रभावशाली प्रोग्रामिङ भाषा हो जुन टेक्नोलोजीको लगभग हरेक भागमा प्रयोग गरिन्छ।

विगतमा कम्प्युटरसँग अन्तर्क्रियाको प्राथमिक मार्ग बेसिक र डस प्रोम्प्टहरू जस्ता भाषा-आधारित ईन्टरफेसहरू मार्फत थियो। यी धेरैजसो दृश्य इन्टरफेस द्वारा प्रतिस्थापित गरिएको छ, जुन सिक्न सजिलो हुन्छ तर कम लचिलोपन प्रस्ताव गर्दछ। तथापि, जाभास्क्रिप्ट जस्ता कम्प्युटर भाषाहरू अझै पनि प्रयोगमा छन् र आधुनिक वेब ब्राउजरहरू र अधिकांश उपकरणहरूमा फेला पार्न सकिन्छ।

जाभास्क्रिप्ट (JS) प्रोग्रामिंग भाषा हो जुन वेबपृष्ठहरू, खेलहरू, अनुप्रयोगहरू र सर्भरहरू विकास गर्दा गतिशील अन्तरक्रिया सिर्जना गर्न प्रयोग गरिन्छ। जाभास्क्रिप्ट नेटस्केपमा सुरु भएको थियो । नेटस्केप जुन १९९० को दशकमा विकसित वेब ब्राउजर हो, र आज जाभास्क्रिप्ट सबैभन्दा प्रसिद्ध र प्रयोग गरिएको प्रोग्रामिंग भाषाहरू मध्ये एक हो।

सुरुमा, यो वेबनिर्देशनहरू जीवित बनाउनका लागि सिर्जना गरिएको थियो र ब्राउजरमा मात्र चलाउन सक्षम थियो। अब, यो कुनै पनि उपकरणमा चल्छ जुन जाभास्क्रिप्ट इन्जिनलाई समर्थन गर्दछ। मानक वस्तुहरू जस्तै `Array` , `Date` , र `Math` उपलब्ध छ। साथै अपरेटरहरू, संरचना र बयानहरू नियन्त्रण पनि यसमा गर्नसकिन्छ। *क्लाइन्ट-साइड जाभास्क्रिप्ट* र *सर्भर-साइड जाभास्क्रिप्ट*, मूल जाभास्क्रिप्टको विस्तारित संस्करणहरू हुन्।

- *क्लाइन्ट-साइड जाभास्क्रिप्टले* वेब पृष्ठहरू र ब्राउजरहरूको बृद्धि र हेरफेर सक्षम गर्दछ। प्रयोगकर्ता घटनाहरूको प्रतिक्रियाहरू जस्तै माउस क्लिकहरू, फारम इनपुट, र पृष्ठ नेभिगेसन यसका केही उदाहरणहरू हुन्।
- *सर्भर-साइड जाभास्क्रिप्टले* सर्भर, डाटाबेस, र फाइल प्रणालीमा पहुँच सक्षम गर्दछ।

जाभास्क्रिप्ट एक व्याख्या गरिएको भाषा हो। जाभास्क्रिप्ट चलाउँदा एक दुभाषियाले प्रत्येक रेखाको व्याख्या गर्दछ र यसलाई चलाउँदछ। आधुनिक ब्राउजरले संकलनका लागि जस्ट इन टाइम `(JIT)` टेक्नोलोजी प्रयोग गर्दछ, जसले जाभास्क्रिप्टलाई कार्यान्वयन योग्य बाइटकोडमा संकलन गर्दछ।

> "लाइभस्क्रिप्ट" जाभास्क्रिप्टलाई दिइएको प्रारम्भिक नाम थियो।

यस पुस्तकलाई तीन भागमा विभाजन गरिएको छ। पहिलो १४ अध्यायहरूले जाभास्क्रिप्ट भाषालाई समेट्छ। निम्न तीन अध्यायहरूले वेब ब्राउजरहरू प्रोग्राम गर्न जाभास्क्रिप्ट कसरी प्रयोग गरिन्छ भनेर छलफल गर्दछ। अन्तिम दुई अध्यायहरू विविध र अभ्यास छन्। जाभास्क्रिप्ट प्रोग्रामिंगसँग सम्बन्धित विभिन्न महत्त्वपूर्ण विषयहरू र केसहरू विविध अध्यायमा वर्णन गरिएको छ, जुन अभ्यासहरू पछ्याउँदछ।

कोड, र यसको साथ के गर्ने

कोड लिखित निर्देशन हो जसले प्रोग्राम बनाउँदछ। यस पुस्तकका धेरै अध्यायहरूमा धेरै कोडहरू छन्, र कसरी प्रोग्राम गर्ने भनेर सिक्ने भागको रूपमा कोड पढ्न र लेख्न महत्त्वपूर्ण छ। तपाईंले उदाहरणहरू तुरुन्तै स्क्यान गर्नुहुँदैन - तिनीहरूलाई ध्यानपूर्वक पढ्नुहोस् र तिनीहरूलाई बुझ्ने प्रयास गर्नुहोस्। यो सुरुमा गाह्रो हुन सक्छ, तर अभ्यासको साथ, तपाईं सुधार हुनेछ। गृहकार्यको लागि पनि त्यस्तै हुन्छ - सुनिश्चित गर्नुहोस् कि तपाईं वास्तवमा तिनीहरूलाई बुझ्नु अघि समाधान लेख्ने प्रयास गर्नुहुन्छ। यो जाभास्क्रिप्ट दुभाषियामा अभ्यासहरूको लागि तपाईंको समाधानहरू चलाउन प्रयास गर्न पनि उपयोगी छ, किनकि यसले तपाईंलाई तपाईंको कोडले सही तरिकाले काम गरिरहेको छ कि छैन भनेर हेर्न अनुमति दिन्छ र तपाईंलाई प्रयोग गर्न र अभ्यासहरू भन्दा बाहिर जान प्रोत्साहित गर्न सक्छ।

टाइपोग्राफिक कन्भेन्सनहरू

यस पुस्तकमा एक मोनोस्पेड फन्टमा लेखिएका पाठले प्रोग्राम को तत्वहरू प्रतिनिधित्व गर्दछ। यो एक स्व-निहित अंश वा नजिकैको प्रोग्रामको अंशको सन्दर्भ हुन सक्छ। एउटा उदाहरण तल देखाइएको छ।

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
    txt += numbers[x];
}
```

कहिलेकाँही, प्रोग्राम को अपेक्षित आउटपुट यो पछि लेखिएको हुन्छ, दुई स्ल्याश द्वारा, जस्तै:

```
console.log(txt);

// Result: txt = '45491625'
```

शुभकामना! 🍀

आधारभूत कुराहरू

यस पहिलो अध्यायमा हामी प्रोग्रामिंग र जाभास्क्रिप्ट भाषाको आधारभूत कुरा सिक्नेछौं।

प्रोग्रामिंगको अर्थ कोड लेख्ने हो। अध्याय अध्याय, अनुच्छेद, वाक्य, वाक्यांशहरू, र अन्तमा विराम चिह्नहरू, र अन्तमा विराम चिह्नहरू मिलेर बनेको छ, त्यस्तै प्रोग्राम लाई साना र साना कम्पोनेन्टहरूमा बिच्छेद गर्न सकिन्छ। अहिलेको लागि, सबैभन्दा महत्त्वपूर्ण एक बयान हो। एक बयाई एक पुस्तक मा एक वाक्य को एक वाक्य को रूपैंगी छ। आफ्नै आफ्नै संरचना र उद्देश्य छ, तर वरपर अन्य कथनहरूको सन्दर्भ बिना यो अर्थपूर्ण छैन।

एक कथन अधिक अनौपचारिक (र सामान्य रूपमा) को 'को लागी एक _ लाईको रूपमा परिचित छ। त्यो किनभने बयान व्यक्तिगत लाइनहरूमा लेखिएको हुन्छ। त्यस्तै, प्रोग्राम हरू माथिबाट तलबाट तल पढिन्छ, बाँयामा दायाँ। तपाईं कुन कोडमा सोच्दै हुनुहुन्छ (स्रोत कोड पनि भनिन्छ) हो। त्यो एक फराकिलो अवधि हुन सक्छ जुन सम्पूर्ण प्रोग्राम वा सब भन्दा सानो भागमा सन्दर्भ गर्न सक्दछ। त्यसकारण, कोडको लाइन मात्र तपाईंको प्रोग्रामको लाइन हो।

यहाँ एक साधारण उदाहरण छ:

```
let hello = "Hello";
let world = "World";

// Message equals "Hello World"
let message = hello + " " + world;
```

यो कोड अर्को प्रोग्राम द्वारा कार्यान्वयन गर्न सकिन्छ एक *इन्टरप्रिटर* भनिन्छ जुन कोड पढ्नेछ, र सही क्रममा सबै कथनहरू कार्यान्वयन गर्दछ।

टिप्पणीहरू

टिप्पणीहरू (comments) बयानहरू हुन् जसले अन्य प्रोग्रामरहरू वा के कोडले के गर्दछ भनेर सहज बनाउँदछन्।

जाभास्क्रिप्टमा, टिप्पणीहरू २ तरीकाले लेख्न सकिन्छ:

- *एकल रेखा टिप्पणीहरू*: यो दुई फरवार्ड स्ल्याशहरू (`//`) बाट सुरु हुन्छ र रेखाको अन्त्यसम्म जारी रहन्छ। स्ल्याशहरू पछिको कुनै पनि कुरा जाभास्क्रिप्ट इन्तेर्प्रेटरद्वारा उपेक्षा गरिएको छ। उदाहरणका लागि:

```
// This is a comment, it will be ignored by the interpreter
let a = "this is a variable defined in a statement";
```

- बहु-रेखा टिप्पणीहरू: अगाडिको स्लेश र तारांकन (`/*`) बाट सुरु हुन्छ र तारांकन र अगाडिको स्लेश (`*/`) सँग समाप्त हुन्छ। उद्घाटन र समापन मार्करहरू बीचको कुनै पनि कुरा जाभास्क्रिप्ट इन्तेर्प्रेटरद्वारा उपेक्षा गरिएको छ। उदाहरणका लागि:

```
/*
This is a multi-line comment,
it will be ignored by the interpreter
*/
let a = "this is a variable defined in a statement";
```

चरहरू

वास्तवमा प्रोग्रामिंग बुझ्नको लागि पहिलो कदम बीजगणितमा फर्केर हेर्नु हो। यदि तपाईं यसलाई स्कूलबाट सम्झनुहुन्छ भने, बीजगणित निम्न जस्ता शब्दहरू लेख्दै सुरु हुन्छ।

```
3 + 5 = 8
```

तपाईंले अज्ञात परिचय दिंदा गणना हरू गर्न थाल्नुहुन्छ, उदाहरणका लागि, तल एक्स:

```
3 + x = 8
```

तपाईं वरिपरि ती स्थानान्तरण एक्स निर्धारण गर्न सक्नुहुन्छ:

```
x = 8 - 3
-> x = 5
```

जब तपाईं एक भन्दा बढी परिचय दिनुहुन्छ तपाईं आफ्नो सर्तहरू अधिक लचिलो बनाउनुहुन्छ - तपाईं चरहरू प्रयोग गर्दै हुनुहुन्छ:

```
x + y = 8
```

तपाईंले x र y को मान परिवर्तन गर्न सक्नुहुन्छ र सूत्र अझै पनि सत्य हुन सक्छ:

```
x = 4
y = 4
```

अथवा

```
x = 3
y = 5
```

प्रोग्रामिंग भाषाहरूको लागि पनि यही कुरा लागू हुन्छ। प्रोग्रामिंगमा, चरहरू परिवर्तन हुने मानहरूको लागि कन्टेनरहरू हुन्। चरहरूले सबै प्रकारका मानहरू र गणनाहरूको परिणामहरू पनि राख्न सक्दछन्। चलहरूको नाम र मान बराबर चिन्ह (=) द्वारा छुट्याइएको छ। परिवर्तनशील नामहरू कुनै पनि अक्षर वा शब्द हुन सक्छन् तर मनमा राख्नुहोस् कि तपाईंले प्रयोग गर्न सक्नुहुने भाषाबाट भाषामा प्रतिबन्धहरू छन्, किनकि केही शब्दहरू अन्य कार्यक्षमताका लागि आरक्षित छन्।

जाभास्क्रिप्टमा यसले कसरी काम गर्दछ भनेर जाँच गरौं, निम्न कोडले दुई चरहरू परिभाषित गर्दछ, दुई थप्ने परिणाम गणना गर्दछ, र यो परिणामलाई तेस्रो चरको मानको रूपमा परिभाषित गर्दछ।

```
let x = 5;
let y = 6;
let result = x + y;
```

Exercise

२० बराबर चर `x` परिभाषित गर्नुहोस् ।

```
let x =
```

ईएस ६ संस्करण

ईसीएमएस्क्रिप्ट 2015 वा ईएस 2015 लाई ई 6 पनि भनिन्छ 2009 देखि जाभास्क्रिप्ट प्रोग्रामिंग भाषामा एक महत्वपूर्ण अद्यावधिक हो। ईएस 6 मा हामीसँग चरहरू घोषणा गर्ने तीन तरिकाहरू छन्। &# X20;

```
var x = 5;
const y = 'Test';
let z = true;
```

घोषणाका प्रकारहरू दायरामा निर्भर गर्दछ। `var` कीवर्डको विपरीत, जसले ब्लक स्कोपको परवाह नगरी विश्वव्यापी वा स्थानीय रूपमा सम्पूर्ण प्रकार्यमा चर परिभाषित गर्दछ, `let` ले तपाईंलाई ब्लक, स्टेटमेन्ट, वा अभिव्यक्तिको दायरामा सीमित चरहरू घोषणा गर्न अनुमति दिन्छ जसमा तिनीहरू प्रयोग गरिन्छ। उदाहरणका लागी।

```
function varTest(){
    var x=1;
    if(true){
        var x=2; // same variable
        console.log(x); //2
    }
    console.log(x); //2
}

function letTest(){
    let x=1;
    if(true){
        let x=2;
        console.log(x); // 2
    }
    console.log(x); // 1
}
```

कन्स्ट (`const`) भेरिएबलहरू अपरिवर्तनीय छन् - तिनीहरूलाई पुन: असाइन गर्न अनुमति छैन।

```
const x = "hi!";
x = "bye"; // this will occurs an error
```

प्रकारहरू

कम्प्युटरहरू परिष्कृत छन् र केवल संख्याहरू भन्दा बढी जटिल चरहरूको प्रयोग गर्न सक्दछन्। यो हो जहाँ चर प्रकारहरू आउँछन्। भेरिएबलहरू धेरै प्रकारहरूमा आउँछन् र विभिन्न भाषाहरूले विभिन्न प्रकारहरूलाई समर्थन गर्दछ।

सामान्य प्रकार चरहरू निम्न अनुसारका छन्:

- **नम्बर**: संख्याहरू पूर्णांकहरू (E.g., `1` , `-5` , `100`) वा फ्लोटिंग-पोइन्ट मानहरू (E.g. `3.14` , `-2.5` , `0.01`) हुन सक्छन्। जाभास्क्रिप्टमा पूर्णांक र फ्लोटिंग-पोइन्ट मानहरूको लागि छुट्टै प्रकार छैन; यसले ती दुवैलाई संख्याको रूपमा व्यवहार गर्दछ।
- **स्ट्रिंग**: स्ट्रिङहरू क्यारेक्टरहरूको अनुक्रम हरू हुन्, जुन एकल उद्धरणहरू (उदाहरणका लागि, `"हेलो"`) वा डबल उद्धरणहरू (उदाहरणका लागि, `"संसार"`) द्वारा प्रतिनिधित्व गरिन्छ।
- **बुलियन**: बुलियनहरूले साँचो वा गलत मूल्यको प्रतिनिधित्व गर्दछ। तिनीहरूलाई 'सत्य' वा 'गलत' (उद्धरण बिना) को रूपमा लेख्न सकिन्छ।
- **Null**: NULL प्रकारको शून्य मानको प्रतिनिधित्व गर्दछ, जसको अर्थ "कुनै मान" हुँदैन। यो `null` (उद्धरण बिना) को रूपमा लेख्न सकिन्छ।
- **अपरिभाषित**: अपरिभाषित (Undefined) प्रकारले सेट नगरिएको मानलाई प्रतिनिधित्व गर्दछ । यदि चर घोषणा गरिएको छ, तर मान निर्दिष्ट गरिएको छैन भने, यो अपरिभाषित छ।
- **वस्तु**: वस्तु (object) गुणहरूको संग्रह हो, जसमध्ये प्रत्येकको नाम र मान छ। तपाईंले घुँघराले ब्रेसेस (`{}`) प्रयोग गरेर वस्तु सिर्जना गर्न सक्नुहुन्छ र नाम-मान जोडी प्रयोग गरेर यसमा गुण हरू निर्दिष्ट गर्न सक्नुहुन्छ ।
- **एरे**: एरे एक विशेष प्रकारको वस्तु हो जसले वस्तुहरूको संग्रह राख्न सक्छ। तपाईंले वर्ग कोष्ठक (`[]`) प्रयोग गरेर र यसमा मानहरूको सूची निर्दिष्ट गरेर सरणी सिर्जना गर्न सक्नुहुन्छ ।
- **प्रकार्य**: प्रकार्य (function) कोडको ब्लक हो जुन परिभाषित गर्न सकिन्छ र त्यसपछि नामद्वारा कल गर्न सकिन्छ। प्रकार्यहरूले तर्कहरू (इनपुटहरू) स्वीकार गर्न र मान (निर्गत) फर्काउन सक्छ। तपाईंले प्रकार्य `function` कुञ्जीशब्द प्रयोग गरेर प्रकार्य सिर्जना गर्न सक्नुहुन्छ ।

जाभास्क्रिप्ट एक `_लूज्ली टाइप_` भाषा हो, जसको अर्थ यो हो कि तपाईले स्पष्ट रूपमा घोषणा गर्नु पर्दैन कि चरहरू कुन प्रकारको डेटा हो। तपाईंले भेरिएबल घोषणा गर्दै हुनुहुन्छ भन्ने सङ्केत गर्न तपाईंले 'भर्' कुञ्जीशब्द प्रयोग गर्न आवश्यक छ, र दुभाषियाले तपाईंले सन्दर्भबाट कुन डेटा प्रकार प्रयोग गर्दै हुनुहुन्छ, र उद्धरणहरूको प्रयोग गर्नुहुन्छ भनेर काम गर्नेछ।

Exercise

तीन चरहरू घोषणा गर्नुहोस् र तिनीहरूलाई निम्न मानहरूसँग सुरु गर्नुहोस्: संख्याको रूपमा `age`, स्ट्रिङको रूपमा `name` र बुलियनको रूपमा `isMarried` ।

```
let age =
let name =
let isMarried =
```

`typeof` अपरेटर चरको डेटाटाइप जाँच गर्न प्रयोग गरीन्छ।

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object
```

जाभास्क्रिप्टमा प्रयोग गरिएका डेटा प्रकारहरू समावेश मानहरूको आधारमा दुई भागमा बिभाजन गर्न सकिन्छ।

डाटा प्रकार जसले मानहरू समावेश गर्न सक्दछ:

- `string`
- `number`

- `boolean`
- `object`
- `function`

`Object` , `Date` , `Array` , `String` , `Number` , र `Boolean` जाभास्क्रिप्टमा उपलब्ध वस्तुहरूको प्रकार हो।

डाटा प्रकार जसले मानहरू समावेश गर्न सक्दैन:

- `null`
- `undefined`

एक आदिम डेटा मान कुनै अतिरिक्त गुण र विधिहरूको साथ एक साधारण डेटा मान हो र यो वस्तु होइन। तिनीहरू अपरिवर्तनीय छन्, यसको अर्थ तिनीहरूलाई परिवर्तन गर्न सकिँदैन। त्यहाँ ७ आदिम डेटा प्रकारहरू छन्:

- string
- number
- bigint
- boolean
- undefined
- symbol
- null

Exercise

`person` भनिने चल घोषणा गर्नुहोस् र यसलाई निम्न गुणहरू समावेश गर्ने वस्तुसँग सुरुआत गर्नुहोस्: संख्याको रूपमा `age`, स्ट्रिङको रूपमा `name` र बुलियनको रूपमा `isMarried`।

```
let person =
```

समानता

प्रोग्रामरहरूमा भ्यारीएबलहरूको सम्बन्धमा समानता निर्धारण गर्नुपर्दछ। यो समानता निर्धारण गर्न समानता अपरेटर प्रयोग गरेर गरिन्छ।

सब भन्दा आधारभूत अपरेटर समानता अपरेटर (`==`) हो। यस अपरेटरले यो निर्धारण गर्दछ कि दुई भ्यारीएबल बराबर छ कि छैन।

उदाहरणका लागि, मान्नुहोस्:

```
let foo = 42;
let bar = 42;
let baz = "42";
let qux = "life";
```

`foo == bar` को मूल्यांकन गर्दा `साँचो` र `baz == qux` को लागी `झूटा` को मूल्यांकन हुनेछ। `foo` र `Baz` बिभिन्न प्रकारका बावजुद `foo == baz` को मूल्यांकन `साँचो` हुनेछ। दृश्यहरूको पछाडि `==` समानता अपरेटरले आफ्नो समानता निर्धारण गर्नु अघि यसको प्रकारलाई अपरेट गर्दछ। यो `===` समानता अपरेटरको विपरित हो।

इकुअलिटी अपरेटरले (`===`) दुई भ्यारीएबल बराबर र समान प्रकारका छन् भन्ने निर्धारण गर्दछ ।

Exercise

समानता अपरेटर (`==` र `===`) प्रयोग गरेर `str` र `str2` को तुलना गर्न प्रयोग गर्नुहोस्।

```
let str1 = "hello";
let str2 = "HELLO";
let bool1 = true;
let bool2 = 1;
// compare using ==
let stringResult1 =
let boolResult1 =
// compare using ===
let stringResult1 =
let boolResult2 =
```

संख्याहरू

जाभास्क्रिप्टमा **केवल एक प्रकारको नम्बर** - 64-बिट फ्लोट पोइन्ट छ। यो जाभाको `डबल` जस्तै हो। अधिकांश अन्य प्रोग्रामिंग भाषाहरूको विपरीत, त्यहाँ कुनै अलग पूर्णांक प्रकार छैन, त्यसैले १ र १.० समान मान हो। संख्या सिर्जना गर्न सजिलो छ, यो `var (वर)` कुञ्जीशब्द प्रयोग गरेर कुनै पनि अन्य चर प्रकारको लागि जस्तै गर्न सकिन्छ।

संख्याहरू स्थिर मानबाट सिर्जना गर्न सकिन्छ।

```
// This is a float:
let a = 1.2;

// This is an integer:
let b = 10;
```

वा अर्को भ्यारीएबलको मानबाट।

```
let a = 2;
let b = a;
```

पूर्णांकको परिशुद्धता १५ अंकसम्म सही हुन्छ र अधिकतम संख्या १७ हुन्छ ।

```
let x = 999999999999999;   // x will be 999999999999999
let y = 9999999999999999;  // y will be 10000000000000000
```

यसले संख्यात्मक स्थिरांकलाई हेक्साडेसिमलको रूपमा व्याख्या गर्दछ यदि तिनीहरू `0x` भन्दा पहिले छन्।

```
let z = 0xFF; // 255
```

गणित

`गणित (Math)` वस्तुले जाभास्क्रिप्टमा गणितीय सञ्चालनहरू प्रदर्शन गर्न अनुमति दिन्छ। गणित वस्तु स्थिर छ र कन्स्ट्रक्टर छैन। पहिले गणित वस्तु सिर्जना नगरी गणित वस्तुको विधि र गुणहरू प्रयोग गर्न सकिन्छ। यसको सम्पत्ति पहुँचका लागि *Math.property* प्रयोग गर्न सकिन्छ। गणितका केही गुणहरू तल वर्णन गरिएका छन्:

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

केही गणित विधिहरू

```
Math.pow(8, 2); // 64
Math.round(4.6); // 5
Math.ceil(4.9); // 5
Math.floor(4.9); // 4
Math.trunc(4.9); // 4
Math.sign(-4); // -1
Math.sqrt(64); // 8
Math.abs(-4.7); // 4.7
Math.sin(90 * Math.PI / 180);  // 1 (the sine of 90 degrees)
Math.cos(0 * Math.PI / 180); // 1 (the cos of 0 degrees)
Math.min(0, 150, 30, 20, -8, -200); // -200
Math.max(0, 150, 30, 20, -8, -200); // 150
Math.random(); // 0.44763808380924375
Math.log(2); // 0.6931471805599453
Math.log2(8); // 3
Math.log10(1000); // 3
```

यसको विधि पहुँच गर्न, एक आवश्यक तर्कको साथ यसको विधिहरू कल गर्न सक्दछ।

विद्युतन	वर्णन
`abs(x)`	`x` को निरपेक्ष मान फर्काउँछ
`acos(x)`	रेडियनमा `x` को आर्ककोसाइन फर्काउँछ
`acosh(x)`	`x` को हाइपरबोलिक आर्ककोसाइन फर्काउँछ
`asin(x)`	रेडियनमा `x` को आर्कसिन फर्काउँछ
`asinh(x)`	`x` को हाइपरबोलिक आर्कसिन फर्काउँछ
`atan(x)`	`-PI/२` र `-PI/२` रेडियनबीच सङ्ख्यात्मक मानको रूपमा `x` को आर्कट्यान्जेन्ट फर्काउँछ
`atan2(y,x)`	यसको तर्कको भागफलको आर्कट्यान्जेन्ट फर्काउँछ
`atanh(x)`	`x` को हाइपरबोलिक आर्कट्यान्जेन्ट फर्काउँछ
`crbt(x)`	`x` को घन मूल फर्काउँछ
`ceil(x)`	`x` को निकटतम पूर्णांकमा माथितिर गोलाकार फर्काउँछ
`cos(x)`	रेडियनमा `x` को कन्साइन फर्काउँछ
`cosh(x)`	`x` को हाइपरबोलिक कोसाइन फर्काउँछ
`exp(x)`	`x` को घातीय मान फर्काउँछ
`floor(x)`	`x` को नजिकको पूर्व पूर्णांकमा राउन्ड तल फर्काउँछ
`log(x)`	`x` को प्राकृतिक लघुगणक फर्काउँछ
`max(x,y,z,... n)`	उच्चतम मानसँग नम्बर फर्काउँछ
`min(x,y,z,... n)`	सबैभन्दा कम मान भएको सङ्ख्या फर्काउँछ
`pow(x,y)`	`y` को शक्तिमा `x` को मान फर्काउँछ
random()	० र १ बीचको सङ्ख्या फर्काउँछ
round(x)	नजिकको पूर्व एक्समा राउन्ड नम्बर
sign(x)	यदि x ऋणात्मक, 'शून्य' वा धनात्मक छ भने फर्काउँछ (-१,०,१) फर्काउँछ
sin(x)	रेडियनमा, `x` को साइन फर्काउँछ
sinh(x)	`x` को हाइपरबोलिक साइन फर्काउँछ
sqrt(x)	`x` को वर्गमूल फर्काउँछ
tan(x)	कोणको स्पर्शरेखा फर्काउँछ
tanh(x)	`x` को हाइपरबोलिक स्पर्शरेखा फर्काउँछ
trunc(x)	सङ्ख्याको पूर्णांक भाग फर्काउँछ (x)

आधारभूत अपरेटरहरू

गणितीय अपरेशनहरू नम्बरहरू केहि आधारभूत अपरेटरहरू प्रयोग गरेर गर्न सकिन्छ:

- **अतिरिक्त अपरेटर (`+`)**: यअतिरिक्त अपरेटरले दुई नम्बरहरू एकसाथ थप्दछ। उदाहरणका लागि:

```
console.log(1 + 2); // 3
console.log(1 + (-2)); // -1
```

- **घटाउ अपरेटर (`-`)**: घटाउ अपरेटरले एउटा नम्बरलाई अर्कोबाट घटाउँदछ। उदाहरणका लागि:

```
console.log(3 - 2); // 1
console.log(3 - (-2)); // 5
```

- **गुणन अपरेटर (`*`)**: गुणन अपरेटरलाई दुई नम्बर गुणा गर्दछ। उदाहरण को लागी:

```
console.log(2 * 3); // 6
console.log(2 * (-3)); // -6
```

- **डिभिजन अपरेटर (`/`)**: डिभिजन अपरेटरले एक नम्बरलाई अर्कोले विभाजन गर्दछ। उदाहरणका लागि:

```
console.log(6 / 2); // 3
console.log(6 / (-2)); // -3
```

- **बाँकी अपरेटर (`%`)**: बाँकी अपरेटरले डिभिजन अपरेशनका बाँकी भाग फिर्ता गर्दछ। उदाहरण को लागी:

```
console.log(10 % 3); // 1
console.log(11 % 3); // 2
console.log(12 % 3); // 0
```

जाभास्क्रिप्ट दोभाषे बायाँबाट दायाँ काम गर्दछ। एकजनाले माईन्ट्स भन्दा अलग र ग्रुप अभिव्यक्तिहरू जस्तै कोष्ठकहरू प्रयोग गर्न सक्दछ: `c = (a / b) + d`

> जाभास्क्रिप्टले दुबै थपिएको र अनुरूपको लागि `+` अपरेटर प्रयोग गर्दछ। नम्बरहरू थपिन्छन् जबकि तारहरू अनुगमन गरिन्छ।

`NaN` एक नम्बर कानूनी संख्या होइन, यो कानूनी संख्या होइन, यो उठ्छ जब हामी एक गैर-संख्यात्मक स्ट्रिङको साथ अंकगणित गर्छौं परिणाम `NaN` (Not a Number)।

```
let x = 100 / "10";
```

पार्सन्ट्र्यान्ट `parseInt` विधिले स्ट्रिंगको रूपमा मान पार्स गर्दछ र पहिलो पूर्णांक फर्कउँछ।

```
parseInt("10"); // 10
parseInt("10.00"); // 10
parseInt("10.33"); // 10
parseInt("34 45 66"); // 34
parseInt(" 60 "); // 60
parseInt("40 years"); //40
parseInt("He was 40"); //NaN
```

जाभास्क्रिप्टमा, यदि हामी सबैभन्दा ठूलो सम्भावित संख्या बाहिर नम्बर गणना गर्दछौं यो `अनन्तको" फिर्ता गर्दछ।

```
let x =  2 / 0; // Infinity
let y = -2 / 0; // -Infinity
```

Exercise

गणित अपरेटरहरू +, -, *,*,, *, / र% प्रयोग गर्नुहोस् `num1` र `num2`" मा निम्न अपरेशनहरू प्रदर्शन गर्न।

```
let num1 = 10;
let num2 = 5;

// Add num1 and num2.
let addResult =
// Subtract num2 from num1.
let subtractResult =
// Multiply num1 and num2.
let multiplyResult =
// Divide num1 by num2.
let divideResult =
// Find the remainder num1 is divided by num2.
let reminderResult =
```

उन्नत अपरेटरहरू

जब अपरेटरहरू कोष्ठक बिना एकसाथ राखिन्छ, तिनीहरूलाई लागू गर्ने क्रम अपरेटरहरूको *प्राथमिकता* द्वारा निर्धारित गरिन्छ। गुणन `(*)` र विभाजन `(/)` को अतिरिक्त `(+)` र घटाउ `(-)` भन्दा बढी प्राथमिकता छ।

```
// multiplication is done first, which is then followed by addition
let x = 100 + 50 * 3; // 250
// with parenthesis operations inside the parenthesis are computed first
let y = (100 + 50) * 3; // 450
// operations with the same precedences are computed from left to right
let z = 100 / 50 * 3;
```

धेरै उन्नत गणित अपरेटरहरू हाम्रो कोडमा प्रयोग गर्न सकिन्छ। यहाँ केही मुख्य उन्नत गणित अपरेटरहरूको सूची छ:

- **मोड्युलो अपरेटर (`%`)**: मोडुलो अपरेटरले विभाजन सञ्चालनको बाँकी भाग फर्काउँछ। उदाहरणका लागि:

```
console.log(10 % 3); // 1
console.log(11 % 3); // 2
console.log(12 % 3); // 0
```

- **एक्सपेक्टेड अपरेटरेटर (* *)**: एक्सपोनेन्टेसन अपरेटरले अर्को नम्बरको शक्तिमा एक नम्बर उठाउँछ। यो एक नयाँ अपरेटर हो र सबै ब्राउजरहरूमा समर्थित छैन, त्यसैले तपाईले यसको सट्टामा `Math.pow` प्रकार्य प्रयोग गर्न आवश्यक हुन सक्छ। उदाहरणका लागि:

```
console.log(2 ** 3); // 8
console.log(3 ** 2); // 9
console.log(4 ** 3); // 64
```

- **इन्क्रिमेन्ट अपरेटर (`++`)**: इन्क्रिमेन्ट अपरेटरले एक-एक गरेर संख्या बढाउँछ। यसलाई उपसर्गको रूपमा प्रयोग गर्न सकिन्छ (ओपरेन्ड भन्दा पहिले) वा पोस्टफिक्स (ओपरेन्ड पछि)। उदाहरणका लागि:

```
let x = 1;
x++; // x is now 2
++x; // x is now 3
```

- **घट्दो अपरेटर (`--`)**: घट्दो अपरेटरले एक-एक गरेर संख्या घटाउँछ। यसलाई उपसर्गको रूपमा प्रयोग गर्न सकिन्छ (ओपरेन्ड भन्दा पहिले) वा पोस्टफिक्स (ओपरेन्ड पछि)। उदाहरणका लागि::

```
let y = 3;
y--; // y is now 2
--y; // y is now 1
```

- **गणित वस्तु**: `म्याथ` वस्तु जाभास्क्रिप्टमा निर्मित वस्तु हो जसले गणितीय प्रकार्यहरू र स्थिरांकहरू प्रदान गर्दछ। तपाईं उन्नत गणित सञ्चालनहरू प्रदर्शन गर्न 'गणित' वस्तुको विधिहरू प्रयोग गर्न सक्नुहुन्छ, जस्तै संख्याको वर्गमूल पत्ता लगाउने, संख्याको साइन गणना गर्ने, वा अनियमित संख्या सिर्जना गर्ने। उदाहरणका लागि:

```
console.log(Math.sqrt(9)); // 3
console.log(Math.sin(0)); // 0
console.log(Math.random()); // a random number between 0 and 1
```

यी जाभास्क्रिप्टमा उपलब्ध उन्नत गणित अपरेटरहरू र प्रकार्यहरूको केही उदाहरणहरू मात्र हुन्। त्यहाँ धेरै धेरै छन् जुन तपाईं आफ्नो कोडमा उन्नत गणित सञ्चालनहरू प्रदर्शन गर्न प्रयोग गर्न सक्नुहुनेछ।

Exercise

`num1` र `num2` मा सञ्चालन गर्न निम्न उन्नत अपरेटरहरू प्रयोग गर्नुहोस्।

```
let num1 = 10;
let num2 = 5;

// ++ operator to increment the value of num1.
const result1 =
// -- operator to decrement the value of num2.
const result2 =
//  += operator to add num2 to num1.
const result3 =
// -= operator to subtract num2 from num1.
const result4 =
```

नलीश कोलेसिंग अपरेटर '??'

नलीश कोलेसिंग अपरेटरले पहिलो तर्क फर्काउँछ यदि यो 'शून्य / अपरिभाषित' छैन भने, अन्यथा दोस्रो । यो दुई प्रश्न चिन्ह `??` को रूपमा लेखिएको छ । `x ?? y` को परिणाम निम्न हुनसकछन् :

- if `x` is defined, then `x`,
- यदि `x` परिभाषित छ भने `x`,
- if `y` isn't defined, then `y`.
- यदि `y` परिभाषित छैन भने `y` ।

> यो भाषामा भर्खरको थप हो र पुरानो ब्राउजरहरू समर्थन गर्न पोलिफिलको आवश्यकता हुन सक्छ।

स्ट्रिंग्स

जाभास्क्रिप्ट स्ट्रिङहरूले अन्य उच्च-स्तरीय भाषाहरूबाट स्ट्रिङ कार्यान्वयनहरूसँग धेरै समानताहरू साझेदारी गर्दछ। तिनीहरूले पाठ-आधारित सन्देशहरू र डेटा प्रतिनिधित्व गर्छन्।

यस कोर्समा, हामी आधारभूत कुराहरू समावेश गर्नेछौं। नयाँ स्ट्रिङहरू कसरी सिर्जना गर्ने र तिनीहरूमा सामान्य सञ्चालनहरू कसरी प्रदर्शन गर्ने।

यहाँ एक स्ट्रिंगको उदाहरण छ:

```
"Hello World"
```

स्ट्रिङ अनुक्रमणिकाहरू शून्य-आधारित हुन्छन्, जसको अर्थ `०` मा पहिलो क्यारेक्टरको प्रारम्भिक स्थिति र त्यसपछि वृद्धिशील क्रममा अरूहरू हुन्छन्।

विभिन्न विधिहरू स्ट्रिंग द्वारा समर्थित छन् र नयाँ मान फिर्ता गर्नुहोस्। यी विधिहरू तल वर्णन गरिएको छ।

नाम	वर्णन
`charAt()`	निर्दिष्ट गरिएको अनुक्रमणिकामा क्यारेक्टर फर्काउँछ
`charCodeAt()`	निर्दिष्ट गरिएको अनुक्रमणिकामा युनिकोड क्यारेक्टर फर्काउँछ
`concat()`	दुई वा दुईभन्दा बढी संयुक्त स्ट्रिङ फर्काउँछ
`constructor`	स्ट्रिङको कन्स्ट्रक्टर प्रकार्य फर्काउँछ
`endsWith()`	निर्दिष्ट गरिएको मानसँग स्ट्रिङ समाप्त भएमा जाँच गर्दछ
`fromCharCode()`	क्यारेक्टरको रूपमा युनिकोड मान फर्काउँछ
`includes()`	स्ट्रिङले निर्दिष्ट गरिएको मानसँग समावेश गरेको छ कि छैन जाँच गर्दछ
`indexOf()`	यसको पहिलो घटनाको अनुक्रमणिका फर्काउँछ
`lastIndexOf()`	यसको अन्तिम घटनाको अनुक्रमणिका फर्काउँछ
`length`	स्ट्रिङको लम्बाइ फर्काउँछ
`localeCompare()`	लोकेलसँग दुई स्ट्रिङ तुलना गर्दछ
`match()`	मान वा नियमित अभिव्यक्तिसँग स्ट्रिङ मिलान गर्दछ
`prototype`	वस्तुको गुण र विधि थप्न प्रयोग गरियो
`repeat()`	निर्दिष्ट गरिएका प्रतिलिपिहरूको सङ्ख्यासँग नयाँ स्ट्रिङ फर्काउँछ
`replace()`	नियमित अभिव्यक्ति वा मानसँग स्ट्रिङले प्रतिस्थापन गरेको मानसँग स्ट्रिङ फर्काउँछ
`search()`	मान वा नियमित अभिव्यक्तिविरुद्ध स्ट्रिङको मिलानमा आधारित अनुक्रमणिका फर्काउँछ
`slice()`	स्ट्रिङको भाग समाविष्ट स्ट्रिङ फर्काउँछ
`split()`	सबस्ट्रिङहरूको सरणीमा स्ट्रिङ विभाजन गर्दछ
`startsWith()`	निर्दिष्ट क्यारेक्टरविरुद्ध सुरु हुने स्ट्रिङ जाँच गर्दछ
`substr()`	सुरुआत अनुक्रमणिकाबाट स्ट्रिङको भाग निकाल्छ
`substring()`	स्ट्रिङको भाग निकाल्छ, दुई सूचकको बीचमा
`toLocalLowerCase()`	होस्टको लोकेल प्रयोग गरेर लोअरकेस क्यारेक्टरसँग स्ट्रिङ फर्काउँछ
`toLocalUpperCase()`	होस्टको लोकेल प्रयोग गरेर अपरकेस क्यारेक्टरसँग स्ट्रिङ फर्काउँछ
`toLowerCase()`	सानो अक्षर क्यारेक्टरसँग स्ट्रिङ फर्काउँछ
`toString()`	स्ट्रिङ वा स्ट्रिङ वस्तुलाई स्ट्रिङको रूपमा फर्काउँछ
`toUpperCase()`	ठूलो अक्षरका साथ स्ट्रिङ फर्काउँछ
`trim()`	हटाइएका सेतो खाली स्थानसँग स्ट्रिङ फर्काउँछ
`trimEnd()`	अन्त्यबाट हटाइएका सेतो खाली स्थानसँग स्ट्रिङ फर्काउँछ
`trimStart()`	सुरुदेखि हटाइएका सेतो खाली स्थानसँग स्ट्रिङ फर्काउँछ
`valueOf()`	स्ट्रिङ वा स्ट्रिङ वस्तुको आदिम मान फर्काउँछ

सिर्जना

तपाईं एकल उद्धरण वा डबल उद्धरणमा पाठ संलग्न गरेर जाभास्क्रिप्टमा स्ट्रिङहरू परिभाषित गर्न सक्नुहुन्छ।

```
// Single quotes can be used
let str = "Our lovely string";

// Double quotes as well
let otherStr = "Another nice string";
```

जाभास्क्रिप्टमा, स्ट्रिङले यूटीएफ-८ क्यारेक्टरहरू समावेश गर्न सक्दछ।

```
"中文 español English हिन्दी العربية português বাংলা русский 日本語 ਪੰਜਾਬੀ 한국어";
```

तपाईंले स्ट्रिङ वस्तु सिर्जना गर्न 'स्ट्रिङ' कन्स्ट्रक्टर पनि प्रयोग गर्न सक्नुहुन्छ ।

```
const stringObject = new String('This is a string');
```

तथापि, यो सामान्यतया स्ट्रिङहरू सिर्जना गर्न 'स्ट्रिंग' कन्स्ट्रक्टर प्रयोग गर्न सिफारिस गरिएको छैन, किनकि यसले स्ट्रिंग आदिम र स्ट्रिङ वस्तुहरू बीच भ्रम पैदा गर्न सक्छ। यो सामान्यतया स्ट्रिंग सिर्जना गर्न स्ट्रिंग शाब्दिक प्रयोग गर्न राम्रो छ।

तपाईंले स्ट्रिङहरू सिर्जना गर्न टेम्प्लेट शाब्दिक हरू पनि प्रयोग गर्न सक्नुहुन्छ। टेम्प्लेट शाब्दिक स्ट्रिङहरू हुन् जुन ब्याकटिक (``) मा संलग्न हुन्छन् र मानहरूका लागि प्लेसहोल्डरहरू समावेश गर्न सक्दछन्। प्लेसहोल्डरहरूलाई `${}` वाक्यरचनाको साथ निरूपित गरिन्छ।

```
const name = 'John';
const message = `Hello, ${name}!`;
```

टेम्प्लेट शाब्दिकले धेरै लाइनहरू पनि समावेश गर्न सक्दछ र प्लेसहोल्डरहरू भित्र कुनै पनि अभिव्यक्ति समावेश गर्न सक्दछ।

स्ट्रिङहरू घटाउन, गुणा गर्न वा विभाजन गर्न सकिँदैन।

Exercise

`name` र `age` का मानहरू समावेश गर्ने स्ट्रिङ सिर्जना गर्न टेम्प्लेट शाब्दिक प्रयोग गर्नुहोस् । स्ट्रिङमा निम्न ढाँचा हुनुपर्दछ: "My name is John and I am 25 years old."।

```
let name = "John";
let age = 25;

// My name is John and I am 25 years old.
let result =
```

प्रतिस्थापन

`प्रतिस्थापन` विधिले हामीलाई क्यारेक्टर, शब्द वा वाक्यलाई स्ट्रिङले प्रतिस्थापन गर्न अनुमति दिन्छ। उदाहरणका लागि:

```
let str = "Hello World!";
let new_str = str.replace("Hello", "Hi");

console.log(new_str);

// Result: Hi World!
```

> परिमार्जकसँग नियमित अभिव्यक्तिको सबै उदाहरणहरूमा मान प्रतिस्थापन गर्न सेट गरिएको छ।

यसले मान वा नियमित अभिव्यक्तिका लागि स्ट्रिङ खोजी गर्दछ र मान(हरू) प्रतिस्थापन गरिएको नयाँ स्ट्रिङ फर्काउँछ । यसले मूल स्ट्रिङ परिवर्तन गर्दैन। आउनुहोस् विश्वव्यापी केस-असंवेदनशील प्रतिस्थापन उदाहरण हेरौं।

```
let text = "Mr Blue has a blue house and a blue car";
let result = text.replace(/blue/gi, "red");

console.log(result);
//Result: Mr red has a red house and a red car
```

लम्बाइ

जाभास्क्रिप्टमा गुण `.length` प्रयोग गरेर स्ट्रिङमा कति क्यारेक्टरहरू छन् भनेर जान्न सजिलो छ। `लम्बाइ` गुणले स्ट्रिङमा क्यारेक्टरहरूको संख्या फर्काउँछ, खाली स्थान हरू र विशेष क्यारेक्टरहरू सहित।

```
let size = "Our lovely string".length;
console.log(size);
// size: 17

let emptyStringSize = "".length
console.log(emptyStringSize);
// emptyStringSize: 0
```

खाली स्ट्रिङको लम्बाइ गुण `०` हो ।

`लम्बाइ` गुण पढ्नका लागि मात्र प्रयोग गरिने गुण हो, त्यसैले तपाईंले यसमा नयाँ मान निर्दिष्ट गर्न सक्नुहुन्न ।

संयोजन

संयोजनमा दुई वा दुई भन्दा बढी स्ट्रिङहरू एकसाथ थप्नु समावेश छ, ती मूल स्ट्रिङहरूको संयुक्त डेटा समावेश गर्ने ठूलो स्ट्रिङ सिर्जना गर्दछ। स्ट्रिङको संयोजनले एक वा एकभन्दा बढी स्ट्रिङलाई अर्को स्ट्रिङमा जोड्दछ। यो जाभास्क्रिप्टमा निम्न तरिकाहरू प्रयोग गरेर गरिन्छ।

- **'+'** अपरेटर प्रयोग गर्दै
- `concat()` विधि प्रयोग गर्दै
- सरणी `join()` विधि प्रयोग गर्दै
- टेम्प्लेट शाब्दिक प्रयोग गर्दै (ईएस 6 मा प्रस्तुत)

स्ट्रिङ `concat()` विधिले स्ट्रिङको सूचीलाई प्यारामिटरको रूपमा स्वीकार गर्दछ र संयोजन पछि नयाँ स्ट्रिङ फर्काउँछ अर्थात् सबै स्ट्रिङहरूको संयोजन। जबकि एरे `join()` विधि एरेमा उपस्थित सबै तत्वहरूलाई एकल स्ट्रिङमा रूपान्तरण गरेर संयोजन गर्न प्रयोग गरिन्छ।

टेम्प्लेट शाब्दिक रूपमा ब्याकटिक `(``)` प्रयोग गर्दछ र मल्टिलाइन स्ट्रिङहरू सिर्जना गर्न र स्ट्रिङ इन्टरपोलेसन प्रदर्शन गर्न सजिलो तरिका प्रदान गर्दछ। `$` चिन्ह र घुंघराले ब्रेसेस `${अभिव्यक्ति}` प्रयोग गरेर ब्याकटिक भित्र अभिव्यक्ति प्रयोग गर्न सकिन्छ।

```
const icon = '👋';
// using template Strings
`hi ${icon}`;

// using join() Method
['hi', icon].join(' ');

// using concat() Method
''.concat('hi ', icon);

//  using + operator
'hi ' + icon;
// hi 👋
```

सशर्त तर्क

अवस्था भनेको कुनै कुराको परीक्षण हो । प्रोग्राम हरूको लागि अवस्थाहरू धेरै तरिकामा धेरै महत्त्वपूर्ण छन्।

सबै भन्दा पहिले, शर्तहरू तपाईंको प्रोग्राम ले काम गर्दछ भनेर सुनिश्चित गर्न प्रयोग गर्न सकिन्छ, चाहे तपाईले प्रक्रियाको लागि कुन डेटा फ्याँक्नुहुन्छ। यदि तपाईं अन्धाधुन्ध डेटामा विश्वास गर्नुहुन्छ भने, तपाईं समस्यामा पर्नुहुनेछ र तपाईंको प्रोग्राम हरू असफल हुनेछन्। यदि तपाईं परीक्षण गर्नुहुन्छ कि तपाईले गर्न चाहानुभएको कुरा सम्भव छ र सही ढाँचामा सबै आवश्यक जानकारी छ भने, त्यो हुनेछैन, र तपाईंको प्रोग्राम धेरै स्थिर हुनेछ। यस्तो सावधानी अपनाउनुलाई रक्षात्मक रूपमा प्रोग्रामिंग पनि भनिन्छ।

तपाईंको लागि अवस्थाहरूले गर्न सक्ने अर्को कुरा शाखाहरूको लागि अनुमति दिनु हो। उदाहरणका लागि, फारम भर्दा तपाईंले पहिले नै शाखाचित्रहरू सामना गर्नुभएको हुन सक्छ। मूलतः, यसले कोडको विभिन्न "शाखाहरू" (भागहरू) कार्यान्वयन गर्न बुझाउँछ, यदि शर्त पूरा भएको छ वा छैन भन्ने मा निर्भर गर्दछ।

यस अध्यायमा, हामी जाभास्क्रिप्टमा सशर्त तर्कको आधार सिक्नेछौं।

यदि

सबै भन्दा सजिलो शर्त एक यदि कथन हो र यसको वाक्य रचना `if (शर्त){ यो गर्नुहोस् ... }` . घुँघराले ब्रेसेस भित्रको कोड कार्यान्वयन गर्नका लागि शर्त सही हुनुपर्दछ। उदाहरणका लागि तपाईंले स्ट्रिङ परीक्षण गर्न सक्नुहुन्छ र यसको मानमा निर्भर अर्को स्ट्रिङको मान सेट गर्न सक्नुहुन्छ।

```
let country = "France";
let weather;
let food;
let currency;

if (country === "England") {
  weather = "horrible";
  food = "filling";
  currency = "pound sterling";
}

if (country === "France") {
  weather = "nice";
  food = "stunning, but hardly ever vegetarian";
  currency = "funny, small and colourful";
}

if (country === "Germany") {
  weather = "average";
  food = "wurst thing ever";
  currency = "funny, small and colourful";
}

let message =
  "this is " +
  country +
  ", the weather is " +
  weather +
  ", the food is " +
  food +
  " and the " +
  "currency is " +
  currency;

console.log(message);
// 'this is France, the weather is nice, the food is stunning, but hardly ever vegetarian and the currency is funny, small and
```

अवस्थाहरू पनि नेस्टेड गर्न सकिन्छ।

अन्य

त्यहाँ एक `अन्य` खण्ड पनि छ जुन लागू हुनेछ जब पहिलो शर्त सत्य छैन। यदि तपाईं कुनै पनि मूल्यमा प्रतिक्रिया गर्न चाहनुहुन्छ भने यो धेरै शक्तिशाली छ, तर विशेष उपचारको लागि विशेष गरी एक एकल:

```
let umbrellaMandatory;

if (country === "England") {
  umbrellaMandatory = true;
} else {
  umbrellaMandatory = false;
}
```

`अन्य` खण्ड अर्को `यदि` सँग जोड्न सकिन्छ। अघिल्लो लेखको उदाहरण लाई रिमेक गरौं।

```
if (country === "England") {
  ...
} else if (country === "France") {
  ...
} else if (country === "Germany") {
  ...
}
```

Exercise

निम्न मानहरूबाट एक सशर्त कथन लेख्नुहोस् जसले `num1` `num2` भन्दा ठूलो छ कि छैन भनेर जाँच गर्दछ। यदि यो छ भने, "परिणाम" चरमा `num1 num2 भन्दा ठूलो छ` असाइन गर्नुहोस्। यदि यो छैन भने, `num1 num2 भन्दा कम वा बराबर छ` असाइन गर्नुहोस्।

```
let num1 = 10;
let num2 = 5;
let result;

// check if num1 is greater than num2
if( condition ) {

}else {

}
```

स्विच

'स्विच' एक सशर्त कथन हो जुन विभिन्न अवस्थाहरूमा आधारित कार्यहरू गर्दछ। यसले शर्तहरू मिलान गर्न कडा ('==') तुलना प्रयोग गर्दछ र मिलान गरिएको अवस्थाको कोड ब्लकहरू कार्यान्वयन गर्दछ। 'स्विच' अभिव्यक्तिको वाक्यविन्यास तल देखाइएको छ ।

```
switch (expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
    // code block
}
```

अभिव्यक्ति एक पटक मूल्यांकन गरिएको छ र प्रत्येक मामला संग तुलना गरिएको छ। यदि म्याच फेला पर्यो भने, यदि 'पूर्वनिर्धारित' कोड ब्लक कार्यान्वयन गरिएको छैन भने सम्बन्धित कोड ब्लक कार्यान्वयन गरिन्छ। 'ब्रेक' कीवर्डले कार्यान्वयन रोक्छ र कहीं पनि राख्न सकिन्छ। यसको अनुपस्थितिमा, अर्को अवस्थाको मूल्यांकन गरिन्छ यदि शर्तहरू मेल खाँदैनन् भने पनि।

स्विच अवस्थाको आधारमा हप्ताको दिन को नाम प्राप्त गर्ने एक उदाहरण तल देखाइएको छ।

```
switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
```

बहुविध मिल्दो अवस्थामा, **पहिलो** मिल्दो मान चयन गरिन्छ, यदि पूर्वनिर्धारित मान चयन गरिएको छैन भने। पूर्वनिर्धारित र कुनै मिल्दो केसको अनुपस्थितिमा, प्रोग्राम स्विच सर्तहरू पछि अर्को कथन(हरू) मा जारी रहन्छ।

Exercise

निम्न मानहरूबाट 'स्विच' कथन लेख्नुहोस् जसले दिनको हप्ता को मान जाँच गर्दछ। यदि दिनअफवीक "सोमवार", "मंगलवार", "बुधवार", "बिहीबार", वा "शुक्रवार" हो भने, परिणाम चरमा "यो एक हप्ताको दिन हो" असाइन गर्नुहोस्। यदि 'डेअफवीक' "शनिबार" वा "आइतबार" हो भने, परिणाममा "यो सप्ताहांत हो" असाइन गर्नुहोस्।

```
let dayOfWeek = "Monday";
let result;
// check if it's a weekday or the weekend
switch(expression) {
case x:
// code block
break;
case y:
// code block
break;
default:
// code block
}
```

तुलनात्मक

अब सशर्त भागमा ध्यान केन्द्रित गरौं।

```
if (country === "France") {
    ...
}
```

सशर्त भाग चर 'देश' हो जुन तीन बराबर चिन्हहरू (`===`) द्वारा पीछा गरिन्छ। यदि चर `देश` सँग सही मान (`फ्रान्स`) र सही प्रकार (`स्ट्रिङ`) दुवै छ भने तीन बराबर चिन्हहरूले परीक्षण गर्दछ। तपाईं डबल बराबर चिन्हहरूको साथ शर्तहरू पनि परीक्षण गर्न सक्नुहुन्छ, तथापि सशर्त जस्तै 'यदि (एक्स == 5)' त्यसपछि `var x = 5;` र `var x = "5";` दुवैको लागि सत्य फर्कन्छ। तपाईंको प्रोग्राम ले के गरिरहेको छ भन्ने आधारमा, यसले धेरै फरक पार्न सक्छ। यो एक उत्तम अभ्यासको रूपमा अत्यधिक सिफारिस गरिएको छ कि तपाईं सँधै समानतालाई दुई ('==' र '!=') को सट्टा तीन बराबर चिन्हहरू ('==' र '!=') सँग तुलना गर्नुहोस्।

अन्य ससर्त परीक्षणहरू:

- `x > a` : के x a भन्दा ठूलो छ?
- `x < a` : के x a भन्दा कम छ?
- `x <= a` : के x a भन्दा कम वा बराबर छ?
- `x >=a` : के x a भन्दा ठूलो वा बराबर छ?
- `x != a` : के x a होइन?

तार्किक तुलना

यदि-अन्य झंझटबाट बच्नको लागि, सरल तार्किक तुलनाहरू प्रयोग गर्न सकिन्छ।

```
let topper = marks > 85 ? "YES" : "NO";
```

माथिको उदाहरणमा, `?` एक तार्किक अपरेटर हो। कोडले भन्छ कि यदि अंकको मान 85 भन्दा बढी छ अर्थात् `अंक > 85` , तब `टपर = हो` ; अन्यथा `टपर = होइन` । मूलतः, यदि तुलना अवस्था सत्य साबित हुन्छ भने, पहिलो तर्क पहुँच गरिएको छ र यदि तुलना शर्त गलत छ भने, दोस्रो तर्क पहुँच गरिएको छ।

संयोजन

यसबाहेक, तपाईं `"वा"` वा `"र"` कथनहरूसँग विभिन्न अवस्थाहरू संयोजन गर्न सक्नुहुन्छ, क्रमशः कथन सत्य छ कि छैन, वा दुवै सत्य छन् कि छैन भनेर परीक्षण गर्न।

जाभास्क्रिप्टमा "वा" लाई `||` र " र " लाई `&&` को रूपमा लेखिएको छ।

एक्सको मान १० र २० को बीचमा छ कि छैन भनेर परीक्षण गर्न चाहनुहुन्छ — तपाईंले एउटा शर्तका साथ त्यसो गर्न सक्नुहुन्छ:

```
if (x > 10 && x < 20) {
    ...
}
```

यदि तपाईं यो सुनिश्चित गर्न चाहनुहुन्छ कि देश "इङ्गल्याण्ड" वा "जर्मनी" हो भने तपाईंले प्रयोग गर्नुहुन्छ:

```
if (country === "England" || country === "Germany") {
    ...
}
```

नोट: संख्याहरूमा सञ्चालनहरू जस्तै, अवस्थाहरू कोष्ठक प्रयोग गरेर समूहीकृत गर्न सकिन्छ, उदाहरण: `यदि ( नाम === "जोन" || नाम === "जेनिफर") && देश === "फ्रान्स")` ।

एरे

एरेहरू प्रोग्रामिंगको एक मौलिक भाग हो। एरे डेटाको सूची हो। हामी एक चरमा धेरै डेटा भण्डारण गर्न सक्दछौं, जसले हाम्रो कोडलाई अधिक पठनीय र बुझ्न सजिलो बनाउँदछ। यसले सम्बन्धित डेटामा प्रकार्यहरू प्रदर्शन गर्न धेरै सजिलो बनाउँदछ।

एरेमा डेटालाई **तत्व** भनिन्छ।

यहाँ एक साधारण एर्रेको उदाहरण छ।

```
// 1, 1, 2, 3, 5, and 8 are the elements in this array
let numbers = [1, 1, 2, 3, 5, 8];
```

एरेहरू एरे शाब्दिक प्रयोग गरेर वा `नयाँ` कुञ्जीशब्दको साथ सजिलै सिर्जना गर्न सकिन्छ।

```
const cars = ["Saab", "Volvo", "BMW"]; // using array literals
const cars = new Array("Saab", "Volvo", "BMW"); // using the new keyword
```

अनुक्रमणिका नम्बर एरेको मानमा पहुँच गर्न प्रयोग गरिन्छ । एरेमा पहिलो तत्वको अनुक्रमणिका जहिले पनि '०' हुन्छ किनकि एरे अनुक्रमणिकाहरू '०' बाट सुरु हुन्छ। अनुक्रमणिका नम्बर पनि एरेको तत्वहरू परिवर्तन गर्न प्रयोग गर्न सकिन्छ।

```
const cars = ["Saab", "Volvo", "BMW"];
console.log(cars[0]);
// Result: Saab

cars[0] = "Opel"; // changing the first element of an array
console.log(cars);
// Result: ['Opel', 'Volvo', 'BMW']
```

> एरेहरू एक विशेष प्रकारको वस्तु हो। एरेमा वस्तुहरू हुन सक्छन्।

एरेको 'लम्बाइ' गुणले संख्या तत्वहरूको गणना फर्काउँछ। एरेद्वारा समर्थित विधिहरू तल देखाइएको छ:

नाम	वर्णन
`concat()`	दुई वा दुईभन्दा बढी संयुक्त एरे फर्काउँछ
`join()`	एरेका सबै तत्वलाई स्ट्रिङमा जोड्दछ
`push()`	एरेको अन्त्यमा एक वा बढी तत्व थप्दछ र लम्बाइ फर्काउँछ
`pop()`	एरेको अन्तिम तत्व हटाउँदछ र त्यो तत्व फर्काउँछ
`shift()`	एरेको पहिलो तत्व हटाउँदछ र त्यो तत्व फर्काउँछ
`unshift()`	एरेको अगाडि एक वा बढी तत्व थप्दछ र लम्बाइ फर्काउँछ
`slice()`	एरेको खण्ड निकाल्छ र नयाँ एरे फर्काउँछ
`at()`	निर्दिष्ट गरिएको अनुक्रमणिका वा `अपरिभाषित (undefined)` मा तत्व फर्काउँछ
`splice()`	एरेबाट तत्वहरू हटाउँदछ र (वैकल्पिक रूपमा) तिनीहरूलाई प्रतिस्थापन गर्दछ, र एरे फर्काउँछ
`reverse()`	एरेका तत्वहरू स्थानान्तरण गर्दछ र एरेमा सन्दर्भ फर्काउँछ
`flat()`	निर्दिष्ट गरिएको गहिराइसम्म पुनरावर्ती रूपमा यसमा समाहित गरिएका सबै उप-एरे तत्वहरूसँग नयाँ एरे फर्काउँछ
`sort()`	एरेका तत्वहरूलाई ठाउँमा क्रमबद्ध गर्दछ, र एरेमा सन्दर्भ फर्काउँछ
`indexOf()`	खोजी तत्वको पहिलो मिलानको अनुक्रमणिका फर्काउँछ
`lastIndexOf()`	खोजी तत्वको अन्तिम मिलानको अनुक्रमणिका फर्काउँछ
`forEach()`	एरेको प्रत्येक तत्वमा `callback` कार्यान्वयन गर्दछ र अपरिभाषित फर्काउँछ
`map()`	प्रत्येक एरे वस्तुमा `callback` कार्यान्वयन गर्दा प्रतिफल मानसहितको नयाँ एरे फर्काउँछ ।
`flatMap()`	गहिराइको `map()` पछि `flat()` चलाउँदछ 1
`filter()`	`callback` ले `सत्य` फर्काएको वस्तुसमावेश गर्ने नयाँ एरे फर्काउँछ
`find()`	`callback` ले `सत्य` फर्काएको पहिलो वस्तु फर्काउँछ
`findLast()`	अन्तिम वस्तु फर्काउँछ जसका लागि `कलब्याक` ले `सत्य` फर्कायो
`findIndex()`	पहिलो वस्तुको अनुक्रमणिका फर्काउँछ जसका लागि `कलब्याक` ले `सत्य` फर्कायो
`findLastIndex()`	अन्तिम वस्तुको अनुक्रमणिका फर्काउँछ जसका लागि `कलब्याक` ले `सत्य` फर्कायो
`every()`	यदि `कलब्याक` ले एरेको प्रत्येक वस्तुका लागि `सत्य` फर्काउँछ भने `सत्य` फर्काउँछ
`some()`	यदि `कलब्याक` ले एरेमा कम्तिमा एउटा वस्तुका लागि `सत्य` फर्काउँछ भने `सत्य` फर्काउँछ
`reduce()`	उद्देश्य घटाउन `कलब्याक(संचयकर्ता, हालको मान, हालको अनुक्रमणिका, एरे)` प्रयोग गर्दछ र `कलब्याक` प्रकार्यद्वारा फर्काएको अन्तिम मान फर्काउँछ ।
`reduceRight()`	त्यसरी नै काम गर्दछ `reduce()` तर अन्तिम तत्वबाट सुरु हुन्छ

अनशिफ्ट

`अनशिफ्ट` विधिले एरेको सुरुमा, वा अगाडि क्रमिक रूपमा नयाँ तत्वहरू थप्दछ। यसले मूल सरणीलाई परिमार्जन गर्दछ र सरणीको नयाँ लम्बाइ फर्कउँछ। उदाहरणका लागि.

```
let array = [0, 5, 10];
array.unshift(-5);  // 4

// RESULT: array = [-5 , 0, 5, 10];
```

`unshift()` विधिले मूल सरणीलाई अधिलेखन गर्दछ।

`अनशिफ्ट` विधिले एक वा बढी तर्कहरू लिन्छ, जसले सरणीको सुरुमा थपिने तत्वहरूको प्रतिनिधित्व गर्दछ। यसले तत्वहरू प्रदान गरिएको क्रममा थप्दछ, त्यसैले पहिलो तत्व सरणीको पहिलो तत्व हुनेछ।

एरेमा बहुविध तत्वहरू थप्न `अनशिफ्ट` प्रयोग गर्ने उदाहरण यहाँ छ:

```
const numbers = [1, 2, 3];
const newLength = numbers.unshift(-1, 0);
console.log(numbers); // [-1, 0, 1, 2, 3]
console.log(newLength); // 5
```

म्याप

`Array.prototype.map()` विधिले एरेमा पुनरावृत्ति गर्दछ र कलब्याक प्रकार्य प्रयोग गरेर यसको तत्वहरू परिमार्जन गर्दछ। कलब्याक प्रकार्य त्यसपछि सरणीको प्रत्येक तत्वमा लागू हुनेछ।

यहाँ `म्याप` प्रयोग गर्नका लागि वाक्यरचना छ:

```
let newArray = oldArray.map(function(element, index, array) {
  // element: current element being processed in the array
  // index: index of the current element being processed in the array
  // array: the array map was called upon
  // Return element to be added to newArray
});
```

उदाहरणका लागि, मानौँ कि तपाईंसँग संख्याहरूको सरणी छ र तपाईं नयाँ सरणी सिर्जना गर्न चाहनुहुन्छ जसले मूल सरणीमा संख्याहरूको मानहरू डबल गर्दछ। तपाईं यो `म्याप` प्रयोग गरेर यो गर्न सक्नुहुन्छ:

```
const numbers = [2, 4, 6, 8];

const doubledNumbers = numbers.map(number => number * 2);

console.log(doubledNumbers);

// Result: [4, 8, 12, 16]
```

तपाईंले `map()` मा पास गरिएको प्रकार्य परिभाषित गर्न बाण प्रकार्य वाक्यविन्यास पनि प्रयोग गर्न सक्नुहुन्छ ।

```
let doubledNumbers = numbers.map((number) => {
  return number * 2;
});
```

वा

```
let doubledNumbers = numbers.map(number => number * 2);
```

`map()` विधिले खाली तत्वहरूका लागि प्रकार्य कार्यान्वयन गर्दैन र मौलिक एरे परिवर्तन गर्दैन ।

फैलने

स्प्रेड अपरेटर `(...)` प्रयोग गरेर एरे वा वस्तुलाई अर्को सरणी वा वस्तुमा छिटो प्रतिलिपि गर्न सकिन्छ । यसले पुनरावृत्ति योग्य जस्तै सरणीलाई शून्य वा बढी तर्कहरू (प्रकार्य कलहरूका लागि) वा तत्वहरू (एरे शाब्दिकहरूका लागि) अपेक्षित स्थानहरूमा विस्तार गर्न अनुमति दिन्छ, वा वस्तु अभिव्यक्ति शून्य वा अधिक कुञ्जी-मान जोडीहरू (वस्तु शाब्दिकहरूका लागि) अपेक्षित स्थानहरूमा विस्तार गर्न अनुमति दिन्छ।

यहाँ केहि उदाहरणहरू छन्:

```
let arr = [1, 2, 3, 4, 5];

console.log(...arr);
// Result: 1 2 3 4 5

let arr1;
arr1 = [...arr]; //copies the arr into arr1

console.log(arr1);    //Result: [1, 2, 3, 4, 5]

arr1 = [6,7];
arr = [...arr,...arr1];

console.log(arr);   //Result: [1, 2, 3, 4, 5, 6, 7]
```

स्प्रेड अपरेटरले केवल आधुनिक ब्राउजरहरूमा काम गर्दछ जुन सुविधालाई समर्थन गर्दछ। यदि तपाईंलाई पुरानो ब्राउजरहरू समर्थन गर्न आवश्यक छ भने, तपाईंले स्प्रेड अपरेटर वाक्यविन्यासलाई समकक्ष ईएस 5 कोडमा रूपान्तरण गर्न बाबेल जस्तो ट्रान्सपिलर प्रयोग गर्न आवश्यक हुनेछ।

सिफ्ट

`सिफ्ट` ले त्यो एरेको पहिलो अनुक्रमणिका मेट्दछ र सबै अनुक्रमणिकाहरू बायाँतिर सार्दछ। यसले मूल सरणी परिवर्तन गर्दछ। यहाँ 'शिफ्ट' प्रयोग गर्नका लागि वाक्यरचना छ:

```
array.shift();
```

उदाहरणका लागि:

```
let array = [1, 2, 3];
array.shift();

// Result: array = [2,3]
```

एर्रेबाट सबै तत्वहरू हटाउन तपाईं लूपको संयोजनमा `सिफ्ट` विधि पनि प्रयोग गर्न सक्नुहुन्छ। तपाईं यो कसरी गर्न सक्नुहुन्छ को एक उदाहरण यहाँ छ:

```
while (array.length > 0) {
  array.shift();
}

console.log(array); // Result: []
```

`shift` विधिले एरेहरूमा मात्र काम गर्दछ, न कि अन्य वस्तुहरूमा जुन एरेहरू जस्तै तर्क वस्तुहरू वा नोडलिस्ट वस्तुहरूसँग मिल्दोजुल्दो छ। यदि तपाईंलाई यी प्रकारका वस्तुहरूमध्ये कुनै एकबाट तत्वहरू स्थानान्तरण गर्न आवश्यक छ भने, तपाईंले यसलाई पहिले `Array.prototype.slice()` विधि प्रयोग गरेर एरेमा रूपान्तरण गर्न आवश्यक छ।

पप

'पप' विधिले एरेबाट अन्तिम तत्व हटाउँदछ र त्यो तत्व फर्काउँछ। यो विधिले सरणीको लम्बाइ परिवर्तन गर्दछ ।

यहाँ 'पप (`pop`)' प्रयोग गर्नका लागि वाक्यरचना छ:

```
array.pop();
```

उदाहरणको लागी:

```
let arr = ["one", "two", "three", "four", "five"];
arr.pop();

console.log(arr);

// Result: ['one', 'two', 'three', 'four']
```

तपाईं एरेबाट सबै तत्वहरू हटाउन लूपको संयोजनमा 'पप' विधि पनि प्रयोग गर्न सक्नुहुन्छ। तपाईं यो कसरी गर्न सक्नुहुन्छ को एक उदाहरण यहाँ छ:

```
while (array.length > 0) {
  array.pop();
}

console.log(array); // Result: []
```

`पप` विधिले एरेहरूमा मात्र काम गर्दछ, न कि अन्य वस्तुहरूमा जुन एरेहरू जस्तै तर्क वस्तुहरू वा नोडलिस्ट वस्तुहरूसँग मिल्दोजुल्दो छ। यदि तपाईंलाई यी प्रकारका वस्तुहरूमध्ये कुनै एकबाट तत्वहरू पप गर्न आवश्यक छ भने, तपाईंले यसलाई पहिले `Array.prototype.slice()` विधि प्रयोग गरेर सरणीमा रूपान्तरण गर्न आवश्यक छ।

सामेल

`सामेल` विधि, एक सरणी एक स्ट्रिंग मा बारी बनाउँछ र यो सबै एक साथ जोडिन्छ। यसले मूल सरणी परिवर्तन गर्दैन । यहाँ 'सामेल' प्रयोग गर्नका लागि वाक्यरचना छ:

```
array.join([separator]);
```

`विभाजक (separator)` तर्क वैकल्पिक छ र परिणामी स्ट्रिङमा तत्वहरू अलग गर्न प्रयोग गरिने क्यारेक्टर निर्दिष्ट गर्दछ। यदि छोडियो भने, सरणी तत्वहरू अल्पविराम `(',')` सँग अलग हुन्छन्।

उदाहरणको लागी:

```
let array = ["one", "two", "three", "four"];

console.log(array.join(" "));

// Result: one two three four
```

> कुनै पनि विभाजक निर्दिष्ट गर्न सकिन्छ तर पूर्वनिर्धारित अल्पविराम `(,)` हो।

माथिको उदाहरणमा, एक स्थान विभाजकको रूपमा प्रयोग गरिन्छ। तपाईंले एरे जस्तो वस्तु (जस्तै तर्क वस्तु वा नोडसूची वस्तु) लाई स्ट्रिङमा रूपान्तरण गर्न `सामेल` पनि प्रयोग गर्न सक्नुहुन्छ, यसलाई `Array.prototype.slice()` विधि प्रयोग गरेर सरणीमा रूपान्तरण गरेर:

```
function printArguments() {
  console.log(Array.prototype.slice.call(arguments).join(', '));
}

printArguments('a', 'b', 'c'); // Result: "a, b, c"
```

लम्बाइ

एरेसँग `लम्बाइ (length)` भनिने गुण छ, र यो एरेको लम्बाइ हो।

```
let array = [1, 2, 3];

let l = array.length;

// Result: l = 3
```

लम्बाइ गुणले एरेमा तत्वहरूको संख्या पनि सेट गर्दछ। उदाहरणका लागि:

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length = 2;

console.log(fruits);
// Result: ['Banana', 'Orange']
```

तपाईंले अनुक्रमणिकाको रूपमा प्रयोग गरेर सरणीको अन्तिम तत्व प्राप्त गर्न `लम्बाइ` गुण पनि प्रयोग गर्न सक्नुहुन्छ। उदाहरणका लागि:

```
console.log(fruits[fruits.length - 1]); // Result: Orange
```

सरणी (एरे) को अन्त्यमा तत्वहरू थप्न तपाईंले `लम्बाइ` गुण पनि प्रयोग गर्न सक्नुहुन्छ । उदाहरणका लागि:

```
fruits[fruits.length] = "Pineapple";
console.log(fruits); // Result: ['Banana', 'Orange', 'Pineapple']
```

> सरणीबाट तत्वहरू थप्दा वा हटाउँदा `लम्बाइ` गुण स्वचालित रूपमा अद्यावधिक हुन्छ।

यो पनि टिप्पण योग्य छ कि `लम्बाइ` गुण एक विधि होइन, त्यसैले तपाईं यसलाई पहुँच गर्दा कोष्ठक प्रयोग गर्न आवश्यक छैन।

पुश

अन्तिम अनुक्रमणिकालाई नयाँ थपिएको वस्तु बनाउने सरणीमा निश्चित वस्तुहरू धकेल्न सकिन्छ। यसले एरेको लम्बाइ परिवर्तन गर्दछ र नयाँ लम्बाइ फर्काउँछ।

यहाँ 'पुश' प्रयोग गर्नका लागि वाक्यरचना छ:

```
array.push(element1[, ...[, elementN]]);
```

`element1, ..., elementN` तर्कहरूले एरेको अन्त्यमा थपिने तत्वहरूको प्रतिनिधित्व गर्दछ।

उदाहरण को लागी:

```
let array = [1, 2, 3];
array.push(4);

console.log(array);

// Result: array = [1, 2, 3, 4]
```

तपाईंले एरे जस्तो वस्तु (जस्तै तर्क वस्तु वा नोडलिस्ट वस्तु) को अन्त्यमा तत्वहरू थप्न `push` पनि प्रयोग गर्न सक्नुहुन्छ, यसलाई `Array.prototype.slice()` विधि प्रयोग गरेर सरणीमा रूपान्तरण गरेर:

```
function printArguments() {
  let args = Array.prototype.slice.call(arguments);
  args.push('d', 'e', 'f');
  console.log(args);
}

printArguments('a', 'b', 'c'); // Result: ["a", "b", "c", "d", "e", "f"]
```

नोट गर्नुहोस् कि `पुश` विधि विधिले मूल एर्रे परिमार्जन गर्दछ। यसले नयाँ एरे सिर्जना गर्दैन।

प्रत्येकको लागि

`फोरएच` विधिले प्रत्येक सरणी तत्वका लागि एक पटक प्रदान गरिएको प्रकार्य कार्यान्वयन गर्दछ । यहाँ 'फोरएच' प्रयोग गर्नका लागि वाक्यरचना छ।

```
array.forEach(function(element, index, array) {
  // element: current element being processed in the array
  // index: index of the current element being processed in the array
  // array: the array forEach was called upon
});
```

उदाहरणका लागि, मानौँ कि तपाईंसँग संख्याहरूको सरणी छ र तपाईं कन्सोलमा प्रत्येक नम्बरको डबल मुद्रण गर्न चाहनुहुन्छ। तपाईं 'फोरएच' प्रयोग गरेर यो गर्न सक्नुहुनेछ:

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(number) {
  console.log(number * 2);
});
```

तपाईंले `फोरएच` मा पास गरिएको प्रकार्य परिभाषित गर्न बाण प्रकार्य वाक्यरचना पनि प्रयोग गर्न सक्नुहुन्छ ।

```
numbers.forEach((number) => {
  console.log(number * 2);
});
```

or

अथवा

```
numbers.forEach(number => console.log(number * 2));
```

`फोरएच` ले मूल एर्रे परिमार्जन गर्दैन । यो केवल एर्रेको तत्वहरूमा पुनरावृत्ति गर्दछ र प्रत्येक तत्वको लागि प्रदान गरिएको प्रकार्य कार्यान्वयन गर्दछ।

> खाली कथनका लागि `forEach()` विधि कार्यान्वयन गरिएको छैन ।

क्रमबद्ध

`क्रमबद्ध` विधिले एरेको वस्तुहरूलाई एक विशिष्ट क्रम (आरोही वा अवरोही) मा क्रमबद्ध गर्दछ।

यहाँ 'क्रमबद्ध' प्रयोग गर्नका लागि वाक्यरचना छ:

```
array.sort([compareFunction]);
```

`compareFunction` तर्क वैकल्पिक छ र एक प्रकार्य निर्दिष्ट गर्दछ जसले क्रमबद्ध क्रम परिभाषित गर्दछ। यदि छोडियो भने, तत्वहरू उनीहरूको स्ट्रिङ प्रतिनिधित्व अनुसार आरोही क्रममा क्रमबद्ध हुन्छन्।

उदाहरण को लागी:

```
let city = ["California", "Barcelona", "Paris", "Kathmandu"];
let sortedCity = city.sort();

console.log(sortedCity);

// Result: ['Barcelona', 'California', 'Kathmandu', 'Paris']
```

संख्याहरू क्रमबद्ध गर्दा गलत तरिकाले क्रमबद्ध गर्न सकिन्छ। उदाहरणका लागि, "35" "100" भन्दा ठूलो छ, किनभने "3" "1" भन्दा ठूलो छ।

संख्यामा क्रमबद्ध समस्या समाधान गर्न, तुलना प्रकार्यहरू प्रयोग गरिन्छ। तुलना प्रकार्यहरू क्रमबद्ध आदेशहरू परिभाषित गर्दछ र तर्कमा आधारित **नकारात्मक**, **शून्य**, वा **धनात्मक** मान फर्काउनुहोस्, जस्तै:

- `b` भन्दा पहिले `a` क्रमबद्ध गर्नुपर्छ भने ऋणात्मक मान
- यदि `a` लाई `b` पछि क्रमबद्ध गर्नु पर्दछ भने एक सकारात्मक मान
- 0 यदि `a` र `b` बराबर छन् र तिनीहरूको क्रमले कुनै फरक पार्दैन

```
const points = [40, 100, 1, 5, 25, 10];
points.sort((a, b) => {return a-b});

// Result: [1, 5, 10, 25, 40, 100]
```

`sort()` विधिले मूल सरणीलाई ओभरराइड गर्दछ ।

सूचकांक

तपाइँसँग डेटा तत्वहरूको तपाईंको एरे छ, तर यदि तपाईं एक विशिष्ट तत्व पहुँच गर्न चाहनुहुन्छ भने के हुन्छ? यहींबाट इन्डेक्स आउँछ । एक **अनुक्रमणिका** ले सरणीमा एक स्थानलाई बुझाउँछ। सूचकांकहरू तार्किक रूपमा एक-एक गरेर प्रगति गर्छन्, तर यो ध्यान दिनुपर्दछ कि एरेमा पहिलो अनुक्रमणिका ० हो, किनकि यो अधिकांश भाषाहरूमा छ। तपाईंले सरणीको अनुक्रमणिकालाई सङ्केत गर्दै हुनुहुन्छ भन्ने सङ्केत गर्न कोष्ठक `[]` प्रयोग गरिन्छ ।

```
// This is an array of strings
let fruits = ["apple", "banana", "pineapple", "strawberry"];

// We set the variable banana to the value of the second element of
// the fruits array. Remember that indices start at 0, so 1 is the
// second element. Result: banana = "banana"
let banana = fruits[1];
```

तपाईंले सरणीमा तत्वको मान सेट गर्न सरणी अनुक्रमणिका पनि प्रयोग गर्न सक्नुहुन्छ ।

```
let array = ['a', 'b', 'c', 'd', 'e'];
//  indices:  0    1    2    3    4
array[4] = 'f';
console.log(array); // Result: ['a', 'b', 'c', 'd', 'f']
```

याद गर्नुहोस् कि यदि तपाईंले सरणीको सीमा भन्दा बाहिरको अनुक्रमणिका प्रयोग गरेर तत्व पहुँच गर्न वा सेट गर्न प्रयास गर्नुभयो भने (जस्तै, ० भन्दा कम वा सरणीको लम्बाइभन्दा बढी वा बराबर अनुक्रमणिका), तपाईंले `अपरिभाषित` मान प्राप्त गर्नुहुनेछ।

```
console.log(array[5]); // Output: undefined
array[5] = 'g';
console.log(array); // Result: ['a', 'b', 'c', 'd', 'f', undefined, 'g']
```

लूप

लूपहरू दोहोरिने अवस्थाहरू हुन् जहाँ लूपमा एक चर परिवर्तन हुन्छ। लुपहरू सजिलो छन्, यदि तपाईं एउटै कोड पटक-पटक चलाउन चाहनुहुन्छ भने, प्रत्येक पटक फरक मानको साथ।

लेख्नुको सट्टा:

```
doThing(cars[0]);
doThing(cars[1]);
doThing(cars[2]);
doThing(cars[3]);
doThing(cars[4]);
```

तपाईंले यसरी लेख्न सक्नुहुन्छ:

```
for (var i = 0; i < cars.length; i++) {
  doThing(cars[i]);
}
```

फर

लुपको सब भन्दा सजिलो रूप फर हो।

```
for (condition; end condition; change) {
    // do it, do it now
}
```

हामीले `फर` लूप प्रयोग गरेर दस पटक एउटै कोड कसरी कार्यान्वयन गर्छौं, हेर्नौं ।

```
for (let i = 0; i < 10; i = i + 1) {
  // do this code ten-times
}
```

> ***नोट***: `i = i + 1` लागि यसरि लेख्न सकिन्छ `i++` ।

एक वस्तु वा एरे `फर इन` लूपको गुणहरू मार्फत लूप गर्न पनि प्रयोग गर्न सकिन्छ।

```
for (key in object) {
  // code block to be executed
}
```

वस्तु र एरेका लागि फर इन` लूपका उदाहरणहरू तल देखाइएको छ।

```
const person = {fname:"John", lname:"Doe", age:25};
let info = "";
for (let x in person) {
  info += person[x];
}

// Result: info = "JohnDoe25"

const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}

// Result: txt = '45491625'
```

The value of iterable objects such as `Arrays` , `Strings` , `Maps` , `NodeLists` can be looped using `for of` statement.

पुनरावृत्ति योग्य वस्तुहरूको मान जस्तै `एरे` , `स्ट्रिङ्स` , `म्याप्स` , `नोडलिस्ट्स` को लागि कथन प्रयोग गरेर लुप गर्न सकिन्छ।

```
let language = "JavaScript";
let text = "";
for (let x of language) {
text += x;
}
```

वहाइल

वहाइल लूपहरूले निर्दिष्ट गरिएको अवस्था सत्य भएसम्म कोडको ब्लक पुनरावृत्ति रूपमा कार्यान्वयन गर्दछ।

```
while (condition) {
  // do it as long as condition is true
}
```

उदाहरणका लागि, वहाइल लूपले यसको कोडको ब्लकलाई पुनरावृत्ति रूपमा कार्यान्वयन गर्नेछ जबसम्म चर म ५ भन्दा कम छ:

```
var i = 0,
  x = "";
while (i < 5) {
  x = x + "The number is " + i;
  i++;
}
```

यदि अवस्था सधैं साँचो छ भने अनन्त लूपिङबाट जोगिन होसियार हुनुहोस्!

डु ... वहाइल

डु ... वहाइल कथनले लूप सिर्जना गर्दछ । यसले तबसम्म निर्दिष्ट कथन कार्यान्वयन गर्दछ जबसम्म परीक्षण अवस्था गलत हुन को लागी मूल्यांकन गर्दैन। बयान कार्यान्वयन पछि अवस्था मूल्यांकन गरिन्छ।

```
do {
  // statement
} while (expression);
```

उदाहरणका लागि हेरौं कसरी १० भन्दा कम नम्बरहरू मुद्रण गर्ने ।

```
var i = 0;
do {
  document.write(i + " ");
  i++; // incrementing i by 1
} while (i < 10);
```

> ***नोट***: `i = i + 1` लाई `i++` लेख्न सकिन्छ.

प्रकार्यहरू

प्रकार्यहरू प्रोग्रामिंगमा सबैभन्दा शक्तिशाली र आवश्यक धारणाहरू मध्ये एक हो। गणितीय प्रकार्यहरू जस्ता प्रकार्यहरूले रूपान्तरणहरू प्रदर्शन गर्दछ, उनीहरूले इनपुट मानहरू लिन्छन् जसलाई **तर्क** र **रिटर्न** आउटपुट मान भनिन्छ।

प्रकार्यहरू दुई तरिकामा सिर्जना गर्न सकिन्छ: `प्रकार्य घोषणा` वा `प्रकार्य अभिव्यक्त` प्रयोग गरेर। *function name* एफ 'अनकसन अभिव्यक्ति' मा छोड्न सकिन्छ जसले यसलाई 'अज्ञात प्रकार्य' बनाउँदछ। प्रकार्यहरू, चरहरू जस्तै, घोषणा गर्नुपर्दछ। आउनुहोस् एउटा प्रकार्यलाई 'डबल' घोषणा गरौं जसले `x` भनिने *तर्क* स्वीकार गर्दछ र एक्सको डबल फकाउँछ:

```
// an example of a function declaration
function double(x) {
  return 2 * x;
}
```

> *नोट:* माथिको प्रकार्य * **परिभाषित हुनुभन्दा पहिले** सन्दर्भित हुन सक्छ।

प्रकार्यहरू जाभास्क्रिप्टमा पनि मानहरू हुन्; तिनीहरू चरमा भण्डारण गर्न सकिन्छ (जस्तै संख्याहरू, स्ट्रिङहरू, आदि ...) र तर्कको रूपमा अन्य प्रकार्यहरूमा दिइन्छ:

```
// an example of a function expression
let double = function (x) {
  return 2 * x;
};
```

> *नोट:* माथिको प्रकार्य * **लाई परिभाषित गर्नु अघि** सन्दर्भित नहुन सक्छ, कुनै पनि अन्य चर जस्तै।

कलब्याक एक प्रकार्य हो जुन अर्को प्रकार्यको तर्कको रूपमा पारित गरिएको छ।

एक एरो प्रकार्य पारंपरिक प्रकार्यहरूको लागि एक कम्प्याक्ट विकल्प हो जुन केहि सीमाहरूको साथ केहि अर्थपूर्ण भिन्नताहरू छन्। यी प्रकार्यहरूको `this`, `arguments` र `super` सँग उनीहरूको आफ्नै बाध्यकारी छैन, र कन्स्ट्रक्टरको रूपमा प्रयोग गर्न सकिँदैन। एरो प्रकार्यको एक उदाहरण।

```
const double =  (x) =>  2 * x;
```

एरो प्रकार्यमा `this` कुञ्जीशब्दले तीर प्रकार्य परिभाषित गर्ने वस्तुलाई प्रतिनिधित्व गर्दछ ।

उच्च अर्डर

"उच्च क्रम प्रकार्यहरू" प्रकार्यहरू हुन् जुन अन्य प्रकार्यहरू हेरफेर गर्दछ। उदाहरणका लागि, एक प्रकार्यले अन्य प्रकार्यहरू तर्कको रूपमा लिन सक्छ र / वा यसको रिटर्न मानको रूपमा प्रकार्य उत्पादन गर्न सक्छ। यस्तो *fancy* कार्यात्मक प्रविधिहरू जाभास्क्रिप्ट र अन्य उच्च-स्तरीय भाषाहरू जस्तै पाइथन, लिस्प, आदिमा तपाईंको लागि उपलब्ध शक्तिशाली संरचनाहरू हुन्।

अब हामी दुई सरल प्रकार्यहरू सिर्जना गर्नेछौं, `add_2` र `add_2` , र `map` भनिने एक उच्च क्रम प्रकार्य। `map` ले दुई तर्कहरू स्वीकार गर्दछ, `func` र `list` (यसको घोषणाले यसैले `map(func,list)` सुरु गर्नेछ), र एक एरे फर्काउँछ। `func` (पहिलो तर्क) एक प्रकार्य हुनेछ जुन एरे `list` (दोस्रो तर्क) मा प्रत्येक तत्वमा लागू हुनेछ।

```
// Define two simple functions
let add_2 = function (x) {
  return x + 2;
};
let double = function (x) {
  return 2 * x;
};

// map is cool function that accepts 2 arguments:
//  func    the function to call
//  list    a array of values to call func on
let map = function (func, list) {
  let output = []; // output list
  for (idx in list) {
    output.push(func(list[idx]));
  }
  return output;
};

// We use map to apply a function to an entire list
// of inputs to "map" them to a list of corresponding outputs
map(add_2, [5, 6, 7]); // => [7, 8, 9]
map(double, [5, 6, 7]); // => [10, 12, 14]
```

माथिको उदाहरणमा प्रकार्यहरू सरल छन्। तथापि, जब अन्य प्रकार्यहरूमा तर्कहरूको रूपमा पारित गरिन्छ, तिनीहरूलाई अधिक जटिल प्रकार्यहरू निर्माण गर्न अप्रत्याशित तरिकामा रचना गर्न सकिन्छ।

उदाहरणका लागि, यदि हामी ध्यान दिन्छौं कि हामी आह्वानहरू `map(add_2, ...)` प्रयोग गर्दछौं। र `map(double, ...)` प्राय हाम्रो कोडमा, हामी निर्णय गर्न सक्छौं कि हामी दुई विशेष-उद्देश्य सूची प्रोसेसिंग प्रकार्यहरू सिर्जना गर्न चाहन्छौं जुन तिनीहरूमा इच्छित अपरेशन बेक गरिएको छ। प्रकार्य संरचना प्रयोग गरेर, हामी यसलाई निम्नानुसार गर्न सक्छौं:

```
process_add_2 = function (list) {
  return map(add_2, list);
};
process_double = function (list) {
  return map(double, list);
};
process_add_2([5, 6, 7]); // => [7, 8, 9]
process_double([5, 6, 7]); // => [10, 12, 14]
```

अब `buildProcessor` नामक प्रकार्य सिर्जना गरौं जसले प्रकार्य `func` लाई इनपुटको रूपमा लिन्छ र `func` -प्रोसेसर फर्काउँछ, अर्थात्, एक प्रकार्य जसले सूचीमा प्रत्येक इनपुटमा `func` लागू गर्दछ।

```
// a function that generates a list processor that performs
let buildProcessor = function (func) {
  let process_func = function (list) {
    return map(func, list);
  };
  return process_func;
};
// calling buildProcessor returns a function which is called with a list input

// using buildProcessor we could generate the add_2 and double list processors as follows:
process_add_2 = buildProcessor(add_2);
process_double = buildProcessor(double);

process_add_2([5, 6, 7]); // => [7, 8, 9]
process_double([5, 6, 7]); // => [10, 12, 14]
```

अर्को उदाहरण हेरौं। हामी `buildMultiplier` नामक प्रकार्य सिर्जना गर्नेछौं जसले संख्या `x` लाई इनपुटको रूपमा लिन्छ र प्रकार्य फर्काउँछ जसले यसको तर्कलाई `x` ले गुणा गर्दछ:

```
let buildMultiplier = function (x) {
  return function (y) {
    return x * y;
  };
};

let double = buildMultiplier(2);
let triple = buildMultiplier(3);

double(3); // => 6
triple(3); // => 9
```

वस्तुहरू

जाभास्क्रिप्टमा वस्तुहरू **उत्परिवर्तनीय** छन् किनकि हामी सन्दर्भ वस्तुद्वारा इंगित मानहरू परिवर्तन गर्दछौं, यसको सट्टा, जब हामी एक आदिम मान परिवर्तन गर्दछौं हामी यसको सन्दर्भ परिवर्तन गर्दैछौं जुन अब नयाँ मानलाई इंगित गर्दैछ र त्यसैले आदिम **अपरिवर्तनीय** हो। जाभास्क्रिप्टका आदिम प्रकारहरू 'सत्य', 'गलत', संख्याहरू, स्ट्रिङहरू, 'शून्य' र 'अपरिभाषित' हुन्। प्रत्येक अन्य मान एक 'वस्तु' हो। वस्तुहरूले 'गुणनाम' समावेश गर्दछ: 'गुणभ्यालु' जोडी । जाभास्क्रिप्टमा 'वस्तु' सिर्जना गर्ने तीन तरिकाहरू छन्:

1. literal (शाब्दिक)

```
let object = {};
// Yes, simply a pair of curly braces!
```

> ***नोट:*** यो **सिफारिस गरिएको** तरिका हो।

2. object-oriented (वस्तु उन्मुख)

```
let object = new Object();
```

> ***नोट:*** यो लगभग जाभा जस्तै छ।

3. and using `object.create`

```
let object = Object.create(proto[, propertiesObject]);
```

> ***नोट:*** यसले निर्दिष्ट प्रोटोटाइप वस्तु र गुणहरूको साथ नयाँ वस्तु सिर्जना गर्दछ।

गुणहरू

वस्तुको प्रोपर्टी `propertyName` हो: `propertyValue` जोडी, जहाँ **गुण नाम स्ट्रिङ** मात्र हुन सक्छ । यदि यो स्ट्रिङ होइन भने, यो एक स्ट्रिंगमा फ्याँकिन्छ। वस्तु **वा पछि** सिर्जना गर्दा तपाईंले गुण निर्दिष्ट गर्न सक्नुहुन्छ । अल्पविरामद्वारा छुट्याइएको शून्य वा बढी गुणहरू हुन सक्छन्।

```
let language = {
  name: "JavaScript",
  isSupportedByBrowsers: true,
  createdIn: 1995,
  author: {
    firstName: "Brendan",
    lastName: "Eich",
  },
  // Yes, objects can be nested!
  getAuthorFullName: function () {
    return this.author.firstName + " " + this.author.lastName;
  },
  // Yes, functions can be values too!
};
```

निम्न कोडले कसरी सम्पत्तिको मूल्य प्राप्त गर्ने भनेर देखाउँछ।

```
let variable = language.name;
// variable now contains "JavaScript" string.
variable = language["name"];
// The lines above do the same thing. The difference is that the second one lets you use litteraly any string as a property na
variable = language.newProperty;
// variable is now undefined, because we have not assigned this property yet.
```

निम्न उदाहरणले कसरी ***नयाँ गुण*** थप्ने वा अवस्थित गुण परिवर्तन गर्ने भनेर देखाउँछ।

```
language.newProperty = "new value";
// Now the object has a new property. If the property already exists, its value will be replaced.
language["newProperty"] = "changed value";
// Once again, you can access properties both ways. The first one (dot notation) is recomended.
```

परिवर्तनीय

वस्तुहरू र आदिम मानहरू बीचको भिन्नता यो हो कि **हामी वस्तुहरू** परिवर्तन गर्न सक्छौं, जबकि आदिम मानहरू **अपरिवर्तनीय** हुन्।

उदाहरणको लागी:

```
let myPrimitive = "first value";
myPrimitive = "another value";
// myPrimitive now points to another string.
let myObject = { key: "first value" };
myObject.key = "another value";
// myObject points to the same object.
```

तपाईंले डट नोटेसन वा स्क्वायर कोष्ठक नोटेसन प्रयोग गरेर वस्तुको गुण थप्न, परिमार्जन गर्न वा मेट्न सक्नुहुन्छ ।

```
let object = {};
object.foo = 'bar'; // Add property 'foo'
object['baz'] = 'qux'; // Add property 'baz'
object.foo = 'quux'; // Modify property 'foo'
delete object.baz; // Delete property 'baz'
```

आदिम मानहरू (जस्तै संख्या र स्ट्रिङहरू) अपरिवर्तनीय छन्, जबकि वस्तुहरू (जस्तै एरे र वस्तुहरू) परिवर्तनशील छन्।

सन्दर्भ

वस्तुहरू प्रतिलिपि गर्न सकिँदैन । तिनीहरू सन्दर्भ द्वारा वरिपरि पारित छन्। वस्तु सन्दर्भ एउटा मान हो जसले वस्तुलाई बुझाउँछ। जब तपाईं `new` अपरेटर वा वस्तु शाब्दिक वाक्यरचना प्रयोग गरेर वस्तु सिर्जना गर्नुहुन्छ, जाभास्क्रिप्टले वस्तु सिर्जना गर्दछ र यसलाई सन्दर्भ प्रदान गर्दछ।

वस्तुको शाब्दिक वाक्यविन्यास प्रयोग गरेर वस्तु सिर्जना गर्ने उदाहरण यहाँ छ:

```
var object = {
  foo: 'bar'
};
```

यहाँ `new` अपरेटर प्रयोग गरेर वस्तु सिर्जना गर्ने उदाहरण छ:

```
var object = new Object();
object.foo = 'bar';
```

जब तपाईंले एउटा चललाई वस्तु सन्दर्भ निर्दिष्ट गर्नुहुन्छ, चरले वस्तुको सन्दर्भ मात्र राख्छ, वस्तुलाई होइन। यसको अर्थ यो हो कि यदि तपाईंले वस्तु सन्दर्भलाई अर्को चरमा निर्दिष्ट गर्नुभयो भने, दुवै चलहरूले एउटै वस्तुलाई इंगित गर्नेछन्।

उदाहरणको लागी:

```
var object1 = {
  foo: 'bar'
};

var object2 = object1;

console.log(object1 === object2); // Output: true
```

माथिको उदाहरणमा, `object1` र `object2` दुवै चरहरू हुन् जसले एउटै वस्तुको सन्दर्भ राख्छन्। `===` अपरेटर सन्दर्भहरू तुलना गर्न प्रयोग गरिन्छ, वस्तुहरू आफैलाई होइन, र यसले `सत्य` फर्कन्छ किनकि दुवै चरहरूले एउटै वस्तुमा सन्दर्भहरू राख्छन्।

> तपाईंले अवस्थित वस्तुको प्रतिलिपि भएको नयाँ वस्तु सिर्जना गर्न `Object.assign()` विधि प्रयोग गर्न सक्नुहुन्छ ।

सन्दर्भद्वारा वस्तुको उदाहरण तल दिइएको छ ।

```
// Imagine I had a pizza
let myPizza = { slices: 5 };
// And I shared it with You
let yourPizza = myPizza;
// I eat another slice
myPizza.slices = myPizza.slices - 1;
let numberOfSlicesLeft = yourPizza.slices;
// Now We have 4 slices because myPizza and yourPizza
// reference to the same pizza object.
let a = {},
  b = {},
  c = {};
// a, b, and c each refer to a
// different empty object
a = b = c = {};
// a, b, and c all refer to
// the same empty object
```

प्रोटोटाइप

प्रत्येक वस्तु एक प्रोटोटाइप वस्तुसँग जोडिएको छ जसबाट यसले गुणहरू प्राप्त गर्दछ।

वस्तु शाब्दिक (`{}`) बाट सिर्जना गरिएका सबै वस्तुहरू स्वचालित रूपमा `Object.prototype` सँग लिङ्क हुन्छन्, जुन जाभास्क्रिप्टको साथ मानक आउने वस्तु हो।

जब जाभास्क्रिप्ट दुभाषिया (तपाईंको ब्राउजरमा एक मोड्युल) ले एउटा गुण फेला पार्ने प्रयास गर्दछ, जुन तपाईं पुन: प्राप्त गर्न चाहनुहुन्छ, जस्तै निम्न कोडमा:

```
let adult = { age: 26 },
  retrievedProperty = adult.age;
// The line above
```

पहिलो, दुभाषियाले वस्तुसँग भएका हरेक सम्पत्तिलाई हेर्छ। उदाहरणका लागि, `adult` को आफ्नै सम्पत्ति छ- `age` । तर त्यो बाहेक, यसमा वास्तवमा केहि अन्य गुणहरू छन्, जुन `Object.prototype` बाट विरासतमा प्राप्त भएको थियो।

```
let stringRepresentation = adult.toString();
// the variable has value of '[object Object]'
```

`toString` एक `Object.prototype's` प्रोपर्टी हो, जुन विरासतमा प्राप्त भएको थियो। यसमा प्रकार्यको मान छ, जसले वस्तुको स्ट्रिङ प्रतिनिधित्व फर्काउँछ। यदि तपाईं यसलाई अझ अर्थपूर्ण प्रतिनिधित्व फिर्ता गर्न चाहनुहुन्छ भने, तपाईं यसलाई ओभरराइड गर्न सक्नुहुन्छ। बस वयस्क वस्तुमा नयाँ गुण थप्नुहोस्।

```
adult.toString = function () {
  return "I'm " + this.age;
};
```

यदि तपाईंले अहिले `toString` प्रकार्यलाई कल गर्नुभयो भने, दुभाषियाले वस्तुमा नै नयाँ गुण फेला पार्नेछ र रोक्नेछ।

यसैले दुभाषियाले पहिलो सम्पत्ति पुन: प्राप्त गर्दछ जुन यसले वस्तुबाट नै बाटोमा फेला पार्नेछ र यसको प्रोटोटाइपमार्फत थप।

पूर्वनिर्धारित वस्तु.प्रोटोटाइपको सट्टाप्रोटोटाइपको रूपमा तपाईंको आफ्नै वस्तु सेट गर्न, तपाईंले निम्नानुसार `Object.create` आह्वान गर्न सक्नुहुन्छ:

```
let child = Object.create(adult);
/* This way of creating objects lets us easily replace the default Object.prototype with the one we want. In this case, the ch
child.age = 8;
/* Previously, child didn't have its own age property, and the interpreter had to look further to the child's prototype to fin
 Now, when we set the child's own age, the interpreter will not go further.
 Note: adult's age is still 26. */
let stringRepresentation = child.toString();
// The value is "I'm 8".
/* Note: we have not overridden the child's toString property, thus the adult's method will be invoked. If adult did not have
```

'बच्चाको' प्रोटोटाइप 'वयस्क' हो, जसको प्रोटोटाइप 'वस्तु.प्रोटोटाइप' हो। प्रोटोटाइपको यस अनुक्रमलाई **प्रोटोटाइप चेन** भनिन्छ।

डिलिट

'डिलिट(मेट्नुहोस्)' गुण वस्तुबाट **गुण** हटाउन प्रयोग गर्न सकिन्छ। जब कुनै गुण मेटिन्छ, यसलाई वस्तुबाट हटाइन्छ र पहुँच गर्न वा गणना गर्न सकिँदैन (जस्तै, यो फोर-इन लूपमा देखा पर्दैन)।

यहाँ 'मेट्नुहोस्' प्रयोग गर्नका लागि वाक्यरचना छ:

```
delete object.property;
```

उदाहरणको लागी:

```
let adult = { age: 26 },
  child = Object.create(adult);

child.age = 8;

delete child.age;

/* Remove age property from child, revealing the age of the prototype, because then it is not overriden. */

let prototypeAge = child.age;
// 26, because child does not have its own age property.
```

'मेट्नुहोस्' अपरेटरले वस्तुको आफ्नै गुणहरूमा मात्र काम गर्दछ, र वंशानुगत गुणहरूमा होइन। यसले 'कन्फिगरेसन योग्य' विशेषता 'गलत' मा सेट गरिएको गुणहरूमा पनि काम गर्दैन।

'मेट्नुहोस्' अपरेटरले वस्तुको प्रोटोटाइप श्रृंखला परिमार्जन गर्दैन। यसले केवल वस्तुबाट निर्दिष्ट गुण हटाउँदछ र यसले वास्तवमा वस्तु वा यसको गुणहरू नष्ट गर्दैन। यसले केवल गुणहरू पहुँच योग्य बनाउँदछ। यदि तपाईंलाई कुनै वस्तुलाई नष्ट गर्न र यसको मेमोरी छोड्न आवश्यक छ भने, तपाईं वस्तुलाई 'शून्य' मा सेट गर्न सक्नुहुन्छ वा मेमोरी पुनः प्राप्त गर्न फोहोर संकलक प्रयोग गर्न सक्नुहुन्छ।

गणना[सम्पादन गर्ने]

गणनाले वस्तुको गुणहरू माथि पुनरावृत्ति गर्ने र प्रत्येक गुणको लागि निश्चित कार्य गर्ने प्रक्रियालाई बुझाउँछ। जाभास्क्रिप्टमा वस्तुको गुणहरू गणना गर्ने धेरै तरिकाहरू छन्।

वस्तुको गुणहरू गणना गर्ने एउटा तरिका भनेको 'फोर-इन' लूप प्रयोग गर्नु हो। 'फर-इन' लूपले वस्तुको गणना योग्य गुणहरू माथि स्वेच्छाचारी क्रममा पुनरावृत्ति गर्दछ, र प्रत्येक गुणको लागि यसले कोडको दिइएको ब्लक कार्यान्वयन गर्दछ।

'फोर-इनका लागि' कथनले वस्तुमा सबै सम्पत्ति नामहरू लुप गर्न सक्छ। गणनामा प्रकार्यहरू र प्रोटोटाइप गुणहरू समावेश हुनेछन्।

```
let fruit = {
    apple: 2,
    orange: 5,
    pear: 1,
  },
  sentence = "I have ",
  quantity;
for (kind in fruit) {
  quantity = fruit[kind];
  sentence += quantity + " " + kind + (quantity === 1 ? "" : "s") + ", ";
}
// The following line removes the trailing comma.
sentence = sentence.substr(0, sentence.length - 2) + ".";
// I have 2 apples, 5 oranges, 1 pear.
```

वस्तुको गुणहरू गणना गर्ने अर्को तरिका `Object.keys()` विधि प्रयोग गर्नु हो, जसले वस्तुको आफ्नै गणना योग्य गुण नामहरूको सरणी फर्कउँछ।

उदाहरणको लागी:

```
let object = {
  foo: 'bar',
  baz: 'qux'
};

let properties = Object.keys(object);
properties.forEach(function(property) {
  console.log(property + ': ' + object[property]);
});

// foo: bar
// baz: qux
```

मिति र समय

`मिति` वस्तुले मिति र समय भण्डारण गर्दछ र यसलाई व्यवस्थापन गर्ने विधिहरू प्रदान गर्दछ। मिति वस्तुहरू स्थिर हुन्छन् र पूर्ण-पाठ स्ट्रिङको रूपमा मिति प्रदर्शन गर्न ब्राउजरको पूर्वनिर्धारित समयक्षेत्र प्रयोग गर्नुहोस् ।

`मिति` सिर्जना गर्न हामी `new Date()` कन्स्ट्रक्टर प्रयोग गर्दछौं र निम्न तरिकामा सिर्जना गर्न सकिन्छ।

```
new Date()
new Date(date string)
new Date(year,month)
new Date(year,month,day)
new Date(year,month,day,hours)
new Date(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)
new Date(milliseconds)
```

महिनाहरू `०` देखि `११` सम्म निर्दिष्ट गर्न सकिन्छ, त्यो भन्दा बढी अर्को वर्षको लागि ओभरफ्लोमा परिणत हुनेछ।

विधिहरू र मितिहरू द्वारा समर्थित विधिहरू तल वर्णन गरिएको छ:

नाम	वर्णन
`constructor`	प्रकार्य फिर्ता जुन मिति वस्तुको प्रोटोटाइप सिर्जना गर्दछ
`getDate ()`	एक महिनाको दिन (१-३१) फर्काउँछ
`getDay()`	हप्ताको दिन (०-६) हप्तामा फर्काउँछ
`getFullYear()`	वर्ष फर्काउँछ (अंकमा)
`getHours()`	घण्टा फर्काउँछ (०-२३)
`getMilliseconds()`	मिलिसेकेन्ड्स फर्काउँछ (०-९९९)
`getMinutes()`	मिनेट फर्काउँछ (०-५९)
`getMonth()`	महिना (०-११) फर्काउँछ
`getSeconds()`	सेकेन्ड फर्काउँछ (०-५९)
`getTime()`	१ जनवरी १९७० को मध्यरातदेखि मिलिसेकेन्डमा निर्दिष्ट गरिएको मितिको सङ्ख्यात्मक मान फर्काउँछ
`getTimezoneOffset()`	मिनेटमा टाइमजोन अफसेट फर्काउँछ
`getUTCDate()`	विश्वव्यापी समयअनुसार महिनाको दिन (१-३१) फर्काउँछ
`getUTCDay()`	युनिभर्सल टाइम अनुसार दिन (०-६) फर्काउँछ
`getUTCFullYear()`	सार्वभौमिक समय अनुसार वर्ष (४-अङ्क) फर्काउँछ
`getUTCHours()`	विश्वव्यापी समय अनुसार घण्टा (०-२३) फर्काउँछ
`getUTCMilliseconds()`	युनिभर्सल टाइम अनुसार मिलिसेकेन्ड(०-९९९) फर्काउँछ
`getUTCMinutes()`	युनिभर्सल टाइम अनुसार मिनेट(०-५९) फर्काउँछ
`getUTCMonth()`	विश्वव्यापी समयअनुसार महिना (०-११) फर्काउँछ
`getUTCSeconds()`	सार्वभौमिक समय अनुसार सेकेन्ड (०-५९) फर्काउँछ
`now()`	जनवरी १, १९७० को मध्यरातदेखि मिलिसेकेन्डमा सङ्ख्यात्मक मान फर्काउँछ
`parse()`	मिति स्ट्रिङ पद वर्णन गर्दछ र जनवरी १, १९७० को मध्यरातदेखि मिलिसेकेन्डमा सङ्ख्यात्मक मान फर्काउँछ
`prototype`	गुण थप्न अनुमति दिन्छ
`setDate()`	एक महिनाको दिन सेट गर्दछ
`setFullYear()`	वर्ष सेट गर्दछ
`setHours()`	घण्टा सेट गर्दछ
`setMilliseconds()`	मिलिसेकेन्ड सेट गर्दछ
`setMinutes()`	मिनेट सेट गर्दछ
`setMonth()`	महिना सेट गर्दछ
`setSeconds()`	सेकेन्ड सेट गर्दछ
`setTime()`	समय सेट गर्दछ

नाम	वर्णन
`setUTCDate()`	निभर्सल समय अनुसार महिनाको दिन सेट गर्दछ
`setUTCFullYear()`	युनिभर्सल समय अनुसार वर्ष सेट गर्दछ
`setUTCHours()`	युनिभर्सल समय अनुसार घण्टा सेट गर्दछ
`setUTCMilliseconds()`	युनिभर्सल समय अनुसार मिलिसेकेन्ड सेट गर्दछ
`setUTCMinutes()`	विश्वव्यापी समय अनुसार मिनेट सेट गर्दछ
`setUTCMonth()`	विश्वव्यापी समय अनुसार महिना सेट गर्दछ
`setUTCSeconds()`	विश्वव्यापी अनुसार सेकेन्ड सेट गर्दछ
`toDateString()`	मानव पठन योग्य ढाँचामा मिति फर्काउँछ
`toISOString()`	ISO ढाँचा अनुसार मिति फर्काउँछ
`toJSON()`	जेएसओएन मितिको रूपमा ढाँचाबद्ध गरिएको स्ट्रिङमा मिति फर्काउँछ
`toLocaleDateString()`	लोकेल कन्भेन्सन प्रयोग गरेर स्ट्रिङमा मिति फर्काउँछ
`toLocaleTimeString()`	लोकेल कन्भेन्सन प्रयोग गरेर स्ट्रिङमा समय फर्काउँछ
`toLocaleString()`	लोकेल कन्भेन्सन प्रयोग गरेर मिति फर्काउँछ
`toString()`	निर्दिष्ट गरिएको मितिको स्ट्रिङ प्रतिनिधित्व फर्काउँछ
`toTimeString()`	मानव-पठनीय ढाँचामा *समय* भाग फर्काउँछ
`toUTCString()`	विश्वव्यापी ढाँचा अनुसार मितिलाई स्ट्रिङमा रूपान्तरण गर्दछ
`toUTC()`	युटिसी ढाँचामा जनवरी १, १९७० को मध्यरातदेखि मिलिसेकेन्ड फर्काउँछ
`valueOf()`	'मिति' को आदिम मान फर्काउँछ

जेसन (JSON)

जाभास्क्रिप्ट वस्तु नोटेशन (जेएसओएन) डेटा भण्डारण र परिवहनको लागि पाठ-आधारित ढाँचा हो। जाभास्क्रिप्ट वस्तुहरू सजिलैसँग जेएसओएनमा रूपान्तरण गर्न सकिन्छ र यसको विपरीत पनि। उदाहरणका लागि:

```
//  a JavaScript object
let myObj = { name:"Ryan", age:30, city:"Austin" };

// converted into JSON:
let myJSON = JSON.stringify(myObj);
console.log(myJSON);
// Result: '{"name":"Ryan","age":30,"city":"Austin"}'

//converted back to JavaScript object
let originalJSON = JSON.parse(myJSON);
console.log(originalJSON);

// Result: {name: 'Ryan', age: 30, city: 'Austin'}
```

'स्ट्रिङिफाई' र 'पार्स' जेएसओएनद्वारा समर्थित दुई विधिहरू हुन्।

विधि	वर्णन
`parse()`	पद वर्णन गरिएको जेसन स्ट्रिङबाट जाभास्क्रिप्ट वस्तु फर्काउँछ
`stringify()`	जाभास्क्रिप्ट वस्तुबाट जेसन स्ट्रिङ फर्काउँछ

निम्न डाटा प्रकारहरू JSON द्वारा समर्थित छन्।

- string
- number
- array
- boolean
- object with valid JSON values
- `null`

यो 'प्रकार्य', 'मिति' वा 'अपरिभाषित' हुन सक्दैन।

प्रयास ... क्याच

प्रोग्रामिंग त्रुटिहरूमा विभिन्न कारणहरूको लागि हुन्छ, केहि कोड त्रुटिहरूबाट हुन्छन्, केहि गलत इनपुटको कारण, र अन्य अप्रत्याशित चीजहरूको कारण। जब एउटा त्रुटि हुन्छ, कोड रोकिन्छ र सामान्यतया कन्सोलमा देखिने त्रुटि सन्देश उत्पन्न गर्दछ ।

कोड कार्यान्वयन रोक्नुको सट्टा, हामी `प्रयास गर्नुहोस् ... क्याच` निर्माण जसले स्क्रिप्टको मृत्यु बिना त्रुटिहरू समात्न अनुमति दिन्छ। 'कोशिश... क्याच 'कन्स्ट्रक्टमा दुई मुख्य ब्लकहरू छन्; 'कोशिश' और फिर 'कैच'।

```
try {
  // code...
} catch (err) {
  // error handling
}
```

सुरुमा, `प्रयास` ब्लकमा कोड कार्यान्वयन गरिन्छ। यदि कुनै त्रुटिहरू देखा परेनन् भने यसले `क्याच` ब्लक छोड्छ। यदि कुनै त्रुटि उत्पन्न हुन्छ भने `प्रयास` कार्यान्वयन रोकिन्छ, नियन्त्रण अनुक्रमलाई `क्याच` ब्लकमा सार्दछ। त्रुटिको कारण `त्रुटि` चरमा कब्जा गरिएको छ।

```
try {
  // code...
  alert('Welcome to Learn JavaScript');
  asdk; // error asdk variable is not defined
} catch (err) {
  console.log("Error has occurred");
}
```

`प्रयास ... क्याच` रनटाइम त्रुटिहरूको लागि काम गर्दछ जसको अर्थ कोड रनेबल र सिंक्रोनस हुनुपर्दछ।

अनुकूल त्रुटि फ्याँक्न, 'थ्रो' कथन प्रयोग गर्न सकिन्छ। त्रुटि वस्तु, जुन त्रुटिहरू द्वारा उत्पन्न हुन्छ, दुई मुख्य गुणहरू छन्।

- **नाम**: त्रुटि नाम
- **सन्देश**: त्रुटिको बारेमा विवरण

यदि हामीलाई 'त्रुटि' सन्देश आवश्यक छैन भने क्याचले यसलाई छोड्न सक्छ।

प्रयास ... क्याच ... अन्तमा

हामी एक अर्को एक अर्डर थप्न सक्दछौं `प्रयास गर्नुहोस् ... क्याच" जसलाई 'अन्तमा' भनिन्छ, यो कोड सबै अवस्थामा कार्यान्वयन गर्दछ। अर्थात् 'प्रयास' पछि जब त्यहाँ कुनै त्रुटि छैन र त्रुटिको मामलामा 'क्याच' पछि। 'प्रयास गर्नुहोस् ... पकड़ो... अन्तमा' तल देखाइएको छ।

```
try {
   // try to execute the code
} catch (err) {
    // handle errors
} finally {
   // execute always
}
```

उदाहरणका लागि।

```
try {
  alert( 'try' );
} catch (err) {
  alert( 'catch' );
} finally {
  alert( 'finally' );
}
```

माथिको उदाहरणमा, `प्रयास ब्लकलाई कार्यान्वयन गरिएको छ जुन त्यसपछि अन्तमा" अन्ततः "कुनै त्रुटिहरू छैन भनेर अनुसरण गरिएको छ।

Exercise

एउटा प्रकार्य `divideNumbers()` लेख्नुहोस् जसले दुई तर्कहरू अंश र भाजक लिन्छ र निम्न सेटिङहरू प्रयोग गरेर भाजकद्वारा अंश विभाजन को परिणाम फर्काउँछ।

```
function divideNumbers(numerator, denominator) {
    try {
      // try statement to divide numerator by denominator.
    } catch (error) {
      // print error message
    } finally {
      // print execution has finished
    }
   // return result
  }
  let answer = divideNumbers(10, 2);
```

नियमित अभिव्यक्ति

एक नियमित अभिव्यक्ति एक वस्तु हो जुन या त `रेगएक्स (RegEx)` कन्स्ट्रक्टरको साथ निर्माण गर्न सकिन्छ वा अगाडिको स्लेश `(/)` क्यारेक्टरमा ढाँचा संलग्न गरेर शाब्दिक मानको रूपमा लेख्न सकिन्छ। नियमित अभिव्यक्ति सिर्जना गर्नका लागि वाक्यविन्यासहरू तल देखाइएको छ।

```
// using regular expression constructor
new RegExp(pattern[, flags]);

// using literals
/pattern/modifiers
```

शाब्दिक प्रयोग गरेर नियमित अभिव्यक्ति सिर्जना गर्दा फ्ल्यागहरू वैकल्पिक हुन्छन्। माथि उल्लिखित विधि प्रयोग गरेर समान नियमित सिर्जना गर्ने उदाहरण निम्नानुसार छ।

```
let re1 = new RegExp("xyz");
let re2 = /xyz/;
```

दुबै तरिकाहरूले रेगेक्स वस्तु सिर्जना गर्नेछ र समान विधिहरू र गुणहरू छन्। त्यहाँ केसहरू छन् जहाँ हामीलाई नियमित अभिव्यक्ति सिर्जना गर्न गतिशील मानहरू आवश्यक हुन सक्छ, त्यस अवस्थामा, शाब्दिकहरूले काम गर्दैन र कन्स्ट्रक्टरसँग जानुपर्छ।

> यदि हामी नियमित अभिव्यक्तिको भाग हुनको लागि अगाडिको स्लेश राख्न चाहन्छौं भने, हामीले ब्याकस्ल्याश `()` को साथ अगाडिको स्लेश `(/)` बाट बच्नु पर्दछ।

केस-असंवेदनशील खोजीहरू प्रदर्शन गर्न प्रयोग गरिने विभिन्न परिमार्जकहरू हुन्:

- `g` - ग्लोबल खोज (पहिलो खेल पछि रोकिनुको सट्टा सबै खेलहरू फेला पार्दछ)
- `i` - केस असंवेदनशील खोज
- `m` - मल्टिलाइन मिलान

क्यारेक्टरहरूको दायरा फेला पार्न नियमित अभिव्यक्तिमा कोष्ठकहरू प्रयोग गरिन्छ । तीमध्ये केही तल उल्लेख गरिएको छ।

- `[abc]` - कोष्ठकको बीचमा कुनै पनि क्यारेक्टर फेला पार्नुहोस्
- `[^abc]` - कुनै पनि क्यारेक्टर फेला पार्नुहोस्, कोष्ठकको बीचमा होइन
- `[0-9]` - कोष्ठकको बीचमा कुनै पनि अङ्क फेला पार्नुहोस्
- `[^0-9]` - कुनै पनि क्यारेक्टर फेला पार्नुहोस्, कोष्ठकको बीचमा होइन (गैर-अंक)
- `(x|y)` - द्वारा अलग गरिएको विकल्पहरू मध्ये कुनै पनि फेला पार्नुहोस् |

मेटाक्यारेक्टरहरू विशेष क्यारेक्टर हरू हुन् जसको नियमित अभिव्यक्तिमा विशेष अर्थ हुन्छ। यी क्यारेक्टरहरू तल थप वर्णन गरिएको छ:

पत्रकहारा	वर्णन
`.`	एकल क्यारेक्टर एक्स्पेट नयाँलाइन वा टर्मिनेटर मिलाउनुहोस्
`\w`	शब्द क्यारेक्टर मिलाउनुहोस् (अल्फान्यूमेरिक क्यारेक्टर `[a-zA-Z0-9_]`)
`\W`	एउटा गैर-शब्द क्यारेक्टर मिलाउनुहोस् (जस्तै `[^a-zA-Z0-9_]`)
`\d`	कुनै पनि अङ्कको क्यारेक्टर मिलाउनुहोस् (जस्तै `[0-9]`)
`\D`	कुनै पनि गैर-डिजिटी क्यारेक्टर मिलाउनुहोस्
`\s`	सेतो स्थान क्यारेक्टर मिलाउनुहोस् (खाली स्थान, ट्याब आदि)
`\S`	सेतो खाली स्थान नभएको क्यारेक्टर मिलाउनुहोस्
`\b`	शब्दको सुरु/अन्त्यमा मिलान गर्नुहोस्
`\B`	मिलान गर्नुहोस् तर शब्दको सुरु / अन्त्यमा होइन
`\0`	`शून्य (NULL)` क्यारेक्टर मिलाउनुहोस्
`\n`	नयाँ रेखा क्यारेक्टर मिलाउनुहोस्
`\f`	फारम फिड क्यारेक्टर मिलाउनुहोस्
`\r`	क्यारिज रिटर्न क्यारेक्टर मिलाउनुहोस्
`\t`	Match a tab character
`\v`	ट्याब ठाडो क्यारेक्टर मिलाउनुहोस्
`\xxx`	अक्टल नम्बरले निर्दिष्ट गरेको क्यारेक्टर मिलाउनुहोस् `xxx`
`\xdd`	हेक्साडेसिमल नम्बर `dd` द्वारा निर्दिष्ट क्यारेक्टर मिलाउनुहोस्
`\udddd`	हेक्साडेसिमल सङ्ख्या `dddd` द्वारा निर्दिष्ट गरिएको युनिकोड क्यारेक्टर मिलाउनुहोस्

रेजेक्स द्वारा समर्थित विधिहरू तल सूचीबद्ध छन्।

नाम	वर्णन
`constructor`	रेगएक्सप वस्तुको प्रोटाइप सिर्जना गर्ने प्रकार्य फर्काउँछ
`global`	`g` परिमार्जक सेट गरिएको छ कि छैन जाँच गर्दछ
`ignoreCase`	`i` परिमार्जक सेट गरिएको छ कि छैन जाँच गर्दछ
`lastIndex`	अर्को मिलान सुरु गर्ने अनुक्रमणिका निर्दिष्ट गर्दछ
`multiline`	यदि `m` परिमार्जक सेट गरिएको छ भने जाँच गर्दछ
`source`	स्ट्रिङको पाठ फर्काउँछ
`exec()`	म्याचका लागि टेस्ट गर्नुहोस् र पहिलो म्याच फर्काउँछ, यदि म्याच भएन भने यसले `शून्य (null)` फर्काउँछ
`test()`	म्याचको लागि परीक्षण गर्नुहोस् र `सत्य` वा `गलत` फर्काउँछ
`toString()`	नियमित एक्स्प्रेसनको स्ट्रिङ मान फर्काउँछ

एक `complie()` विधि नियमित अभिव्यक्ति को अनुपालन गर्दछ र अवमूल्यन गरिएको छ।

नियमित अभिव्यक्तिका केही उदाहरणहरू यहाँ देखाइएको छ।

```
let text = "The best things in life are free";
let result = /e/.exec(text); // looks for a match  of e in a string
// result: e


let helloWorldText = "Hello world!";
// Look for "Hello"
let pattern1 = /Hello/g;
let result1 = pattern1.test(helloWorldText);
// result1: true

let pattern1String = pattern1.toString();
// pattern1String : '/Hello/g'
```

मोड्युलहरू

वास्तविक संसारमा, एक प्रोग्राम नयाँ कार्यक्षमताको आवश्यकताहरूको सामना गर्न व्यवस्थित रूपमा बढ्छ। बढ्दो कोडबेस संरचना र कोड को रखरखाव को साथ अतिरिक्त काम को आवश्यकता छ। यद्यपि यसले भविष्यमा भुक्तानी गर्नेछ, यसलाई बेवास्ता गर्न र प्रोग्राम हरूलाई गहिरो रूपमा पेचिलो हुन अनुमति दिन यो लोभलाग्दो छ। वास्तविकतामा, यसले अनुप्रयोगको जटिलता बढाउँछ, किनकि एक प्रणालीको समग्र समझ निर्माण गर्न बाध्य हुन्छ र अलगावमा कुनै पनि टुक्रा हेर्न गाह्रो हुन्छ। दोस्रो, यसको कार्यक्षमता को उपयोग गर्न को लागी एक अनटैंगलिंग मा अधिक समय लगानी गर्नु पर्छ।

मोड्युलहरू यी समस्याहरूबाट बच्न आउँदछन्। एक 'मोड्युल' ले निर्दिष्ट गर्दछ कि यो कोडको कुन टुक्राहरूमा निर्भर गर्दछ, साथै यसले अन्य मोड्युलहरू प्रयोग गर्नका लागि कुन कार्यक्षमता प्रदान गर्दछ। मोड्युलहरू जुन अन्य मोड्युलमा निर्भर छन् तिनीहरूलाई _निर्भरताहरू' भनिन्छ। विभिन्न मोड्युल पुस्तकालयहरू मोड्युलहरूमा कोड व्यवस्थित गर्न र मागमा लोड गर्न त्यहाँ छन्।

- AMD - सबैभन्दा पुरानो मोड्युल प्रणाली मध्ये एक हो। सुरुमा यो require.js प्रयोग गरियो।
- CommonJS - मोड्युल प्रणाली नोड ट्रान्स सर्भर को लागी सिर्जना गरिएको हो।।
- UMD - मोड्युल प्रणाली जुन AMD र CommonJS सँग उपयुक्त छ।

मोड्युलहरूले एक अर्कालाई लोड गर्न सक्दछन्, र कार्यक्षमता आदानप्रदान गर्न विशेष निर्देशनहरू आयात र निर्यात प्रयोग गर्न सक्दछन्, र एक अर्काको कार्यहरू कल गर्न सक्दछन्।

- `निर्यात (export)` ले लेबलहरू कार्यहरू र चरहरू जुन हालको मोड्युल बाहिरबाट पहुँच योग्य हुनुपर्दछ।
- `आयात (import)` - बाहिर मोड्युलबाट कार्यक्षमता आयात गर्दछ।

मोड्युलमा 'आयात' र 'निर्यात' संयन्त्र हेरौं। हामीसँग `sayHi` प्रकार्य `sayHi.js` फाइलबाट निर्यात गरिएको छ।

```
// 🗏 sayHi.js
export const sayHi = (user) => {
    alert(`Hello, ${user}!`);
};
```

'आयात' निर्देशनको सहायताले `main.js` फाइलमा `sayHi` प्रकार्य खपत हुन्छ।

```
// 🗏 main.js
import { sayHi } from "./sayHi.js";

alert(sayHi); // function...
sayHi("Kelvin"); // Hello, Kelvin!
```

यहाँ, आयात निर्देशिकाले सापेक्ष पथ आयात गरेर मोड्युल लोड गर्दछ र `sayHi` चर प्रदान गर्दछ।

मोड्युलहरू दुई तरिकामा निर्यात गर्न सकिन्छ: **नाम** र **पूर्वनिर्धारित** । यसबाहेक, नामित निर्यातहरू इनलाइन वा व्यक्तिगत रूपमा तोक्न सकिन्छ।

```
// 🗏 person.js

// inlined named exports
export const name = "Kelvin";
export const age = 30;

// at once
const name = "Kelvin";
const age = 30;
export { name, age };
```

एउटा फाइलमा एउटा मात्र पूर्वनिर्धारित 'निर्यात' हुन सक्छ ।

```
// 📁 message.js
const message = (name, age) => {
    return `${name} is ${age} years old.`;
};
export default message;
```

निर्यातको प्रकारका आधारमा हामी यसलाई दुई तरिकाले आयात गर्न सक्छौं। नामित निर्यात घुंघराले ब्रेसेस प्रयोग गरेर निर्माण गरिएको छ जबकि, पूर्वनिर्धारित निर्यात हरू छैनन्।

```
import { name, age } from "./person.js"; // named export import
import message from "./message.js"; // default export import
```

मोड्युलहरू निर्दिष्ट गर्दा, हामीले *circular dependency* बच्नुपर्छ। सर्कुलर निर्भरता एक स्थिति हो जहाँ मोड्युल ए बी मा निर्भर गर्दछ, र बी पनि प्रत्यक्ष वा अप्रत्यक्ष रूपमा ए मा निर्भर गर्दछ।

वर्ग

वर्गहरू (classes) वस्तु सिर्जना गर्न टेम्प्लेटहरू हुन्। यसले डेटाको साथ काम गर्न कोडको साथ डेटा लाई समाहित गर्दछ। कुञ्जीशब्द 'वर्ग' वर्ग सिर्जना गर्न प्रयोग गरिन्छ। र 'कन्स्ट्रक्टर' नामक एक विशिष्ट विधि 'वर्ग' को साथ सिर्जना गरिएको वस्तु सिर्जना गर्न र सुरु गर्न प्रयोग गरिन्छ। कार वर्गको एक उदाहरण तल देखाइएको छ।

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}

let myCar = new Car("Toyota", 2021);
console.log(myCar.age()) // 1
```

वर्ग यसको प्रयोग गर्नु अघि परिभाषित गर्नु पर्दछ।

वर्ग शरीरमा, विधिहरू वा कन्स्ट्रक्टरहरू परिभाषित गरिन्छ र `strict mode` मा कार्यान्वयन गरिन्छ। वाक्यविन्यासले कडा मोडको पालना नगर्दा त्रुटि हुन्छ ।

स्थिर

'स्थिर' कीवर्डले वर्गको लागि स्थिर विधिहरू वा गुणहरू परिभाषित गर्दछ। यी विधिहरू र गुणहरू वर्गमा नै भनिन्छ।

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
let myCar = new Car("Toyota");

console.log(myCar.hello()); // This will throw an error
console.log(Car.hello(myCar));
// Result: Hello Toyota
```

`this` कीवर्ड प्रयोग गरेर उही वर्गको अर्को स्थिर विधिको स्थिर विधि वा गुण पहुँच गर्न सक्छ।

उत्तराधिकार

उत्तराधिकार कोड पुन: प्रयोज्य उद्देश्यहरूको लागि उपयोगी छ किनकि यसले एक वर्गको अवस्थित गुणहरू र विधिहरू विस्तार गर्दछ। वर्ग वंशानुक्रम सिर्जना गर्न `extends` कीवर्ड प्रयोग गरिन्छ।

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Toyota", "Camry");
console.log(myCar.show()); // I have a Camry, it is a Toyota.
```

> अभिभावक वर्गको प्रोटोटाइप 'वस्तु' वा 'शून्य' हुनुपर्दछ।

'सुपर' विधि कन्स्ट्रक्टर भित्र प्रयोग गरिन्छ र मूल वर्गलाई बुझाउँछ। यसको साथ, एक अभिभावक वर्ग गुणहरू र विधिहरू पहुँच गर्न सक्दछ।

पहुँच परिमार्जनकर्ताहरू

`public` , `private` , र `protected` तीन पहुँच परिमार्जकहरू हुन् जुन कक्षामा बाहिरबाट यसको पहुँच नियन्त्रण गर्न प्रयोग गरिन्छ। पूर्वनिर्धारित रूपमा, सबै सदस्यहरू (गुणहरू, क्षेत्रहरू, विधिहरू, वा प्रकार्यहरू) वर्ग बाहिरबाट सार्वजनिक रूपमा पहुँच योग्य छन्।

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
let myCar = new Car("Toyota");
console.log(Car.hello(myCar)); // Hello Toyota
```

`private` सदस्यहरूले कक्षाभित्र मात्र आन्तरिक रूपमा पहुँच गर्न सक्दछन् र बाहिरबाट पहुँच योग्य हुन सक्दैनन्। प्राइभेट `#` बाट सुरु गर्नुपर्छ ।

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
  #present(carname) {
    return 'I have a ' + this.carname;
  }
}
let myCar = new Car("Toyota");
console.log(myCar.#present("Camry")); // Error
console.log(Car.hello(myCar)); // Hello Toyota
```

`protected` क्षेत्रहरू वर्ग भित्र र यसलाई विस्तार गर्नेहरूबाट मात्र पहुँचयोग्य छन्। यी आन्तरिक इन्टरफेसको लागि उपयोगी छन् किनकि उत्तराधिकारी वर्गले अभिभावक वर्गमा पनि पहुँच प्राप्त गर्दछ। `_` सँग संरक्षित फाँटहरू ।

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  _present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this._present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Toyota", "Camry");
console.log(myCar.show()) // I have a Toyota, it is a Camry
```

ब्राउजर वस्तु मोडेल (BOM)

ब्राउजर वस्तु मोडेलले हामीलाई ब्राउजर सञ्झ्यालसँग अन्तरक्रिया गर्न दिन्छ। `सञ्झ्याल (window)` वस्तुले ब्राउजरको सञ्झ्यालको प्रतिनिधित्व गर्दछ र सबै ब्राउजरहरू द्वारा समर्थित छ।

वस्तु `सञ्झ्याल` ब्राउजरको लागि पूर्वनिर्धारित वस्तु हो, त्यसैले हामी `सञ्झ्याल` निर्दिष्ट गर्न सक्दछौं वा सबै प्रकार्यहरू सीधै कल गर्न सक्दछौं।

```
window.alert("Welcome to Learn JavaScript");

alert("Welcome to Learn JavaScript")
```

त्यस्तै तरिकामा, हामी सञ्झ्याल वस्तुको तल अन्य गुणहरू जस्तै इतिहास, स्क्रिन, नेभिगेटर, स्थान, र यतिमा कल गर्न सक्दछौं।

सञ्झ्याल

`सञ्झ्याल (window)` वस्तुले ब्राउजर सञ्झ्यालको प्रतिनिधित्व गर्दछ र ब्राउजरहरू द्वारा समर्थित छ। विश्वव्यापी चलहरू, वस्तुहरू, र प्रकार्यहरू पनि सञ्झ्याल वस्तुको भाग हुन्।

विश्वव्यापी **चर गुणहरू** हुन् र **प्रकार्यहरू** सञ्झ्याल वस्तुको **विधिहरू** हुन्।

आउनुहोस् स्क्रिन गुणहरूको उदाहरण लिनुहोस्। यो ब्राउजर सञ्झ्यालको साइज निर्धारण गर्न प्रयोग गरिन्छ र पिक्सेलमा मापन गरिन्छ।

- `window.innerHeight` - ब्राउजर सञ्झ्यालको भित्री उचाइ
- `window.innerWidth` - ब्राउजर सञ्झ्यालको भित्री चौडाइ

> ***Note***: `window.document` `document.location` जस्तै हो किनकि कागजात वस्तु मोडेल(डोम) सञ्झ्याल वस्तुको भाग हो।

सञ्झ्याल विधिका केही उदाहरणहरू

- `window.open()` - नयाँ सञ्झ्याल खोल्नुहोस्
- `window.close()` - हालको सञ्झ्याल बन्द गर्नुहोस्
- `window.moveTo()` - हालको सञ्झ्याल सार्नुहोस्
- `window.resizeTo()` - हालको सञ्झ्याल पुन: साइज गर्नुहोस्

पपअप

पपअपहरू जानकारी देखाउन, प्रयोगकर्ता पुष्टिकरण लिन, वा थप कागजातहरूबाट प्रयोगकर्ता इनपुट लिनको लागि एक अतिरिक्त तरिका हो। पपअपले नयाँ यूआरएलमा नेभिगेट गर्न सक्दछ र ओपनर सञ्झ्यालमा सूचना पठाउन सक्छ। **चेतावनी बाकस**, **पुष्टि बाकस**, र **प्रॉम्प्ट बक्स** विश्वव्यापी प्रकार्यहरू हुन् जहाँ हामी पपअप जानकारी देखाउन सक्छौं।

1. **alert()**: यसले प्रयोगकर्तालाई जानकारी प्रदर्शित गर्दछ र अगाडि बढ्नका लागि "**OK**" बटन छ।

```
alert("Alert message example");
```

2. **confirm()**: कुनै कुरा पुष्टि गर्न वा स्वीकार गर्न संवाद बाकसको रूपमा प्रयोग गर्नुहोस् । यसमा "**OK**" र "**Cancel**" अगाडि बढ्नको लागि छ। यदि प्रयोगकर्ताले "**OK**" क्लिक गर्दछ भने यसले 'सत्य' फर्काउँछ, यदि "**Cancel**" क्लिक गर्दा यसले 'गलत' फर्काउँछ।

```
let txt;
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

3. **prompt()**: "**OK**" र "**Cancel**" बटनसँग प्रयोगकर्ता आगत मान लिन्छ। यदि प्रयोगकर्ताले कुनै आगत मान प्रदान गरेन भने यसले 'नल' फर्काउँछ ।

```
//syntax
//window.prompt("sometext","defaultText");

let person = prompt("Please enter your name", "Harry Potter");

if (person == null || person == "") {
  txt = "User cancelled the prompt.";
} else {
  txt = "Hello " + person + "! How are you today?";
}
```

स्क्रिन

`स्क्रिन (screen)` वस्तुले हालको सञ्झ्याल रेन्डर गरिएको पर्दाको बारेमा जानकारी समावेश गर्दछ । 'स्क्रिन' वस्तु पहुँच गर्न हामी `सञ्झ्याल` वस्तुको 'स्क्रिन' गुण प्रयोग गर्न सक्छौं।

```
window.screen
//or
screen
```

`window.screen` वस्तुसँग विभिन्न गुणहरू छन्, तीमध्ये केही यहाँ सूचीबद्ध छन्:

गुण	विवरण
`height`	स्क्रिनको पिक्सेल उचाइ प्रतिनिधित्व गर्दछ
`left`	हालको स्क्रिनको बायाँ पट्टिको पिक्सेल दूरी प्रतिनिधित्व गर्दछ
`pixelDepth`	पढ्ने मात्र गुण जसले पर्दाको बिट गहिराइ फर्कउँछ
`top`	हालको स्क्रिनको माथिल्लो भागको पिक्सेल दूरी प्रतिनिधित्व गर्दछ
`width`	स्क्रिनको पिक्सेल चौडाइ प्रतिनिधित्व गर्दछ
`orientation`	पर्दा अभिमुखीकरण एपीआईमा निर्दिष्ट गरिए अनुसार पर्दा अभिमुखीकरण फर्कउँछ
`availTop`	पढ्ने मात्र गुण जसले माथिबाट पहिलो पिक्सेल फर्कउँछ जुन प्रणाली तत्वहरू द्वारा लिइएको छैन
`availWidth`	पढ्ने मात्र गुण जसले प्रणाली तत्वहरू बाहेक स्क्रिनको पिक्सेल चौडाइ फर्कउँछ
`colorDepth`	पढ्ने मात्र गुण जसले रङहरू प्रतिनिधित्व गर्न प्रयोग गरिने बिटहरूको सङ्ख्या फर्कउँछ
`height`	स्क्रिनको पिक्सेल उचाइ प्रतिनिधित्व गर्दछ
`left`	हालको स्क्रिनको बायाँ पट्टिको पिक्सेल दूरी प्रतिनिधित्व गर्दछ
`pixelDepth`	पढ्ने मात्र जसले पर्दाको बिट गहिराइ फर्कउँछ
`top`	हालको स्क्रिनको माथिल्लो भागको पिक्सेल दूरी प्रतिनिधित्व गर्दछ
`width`	स्क्रिनको पिक्सेल चौडाइ प्रतिनिधित्व गर्दछ
`orientation`	पर्दा अभिमुखीकरण एपीआईमा निर्दिष्ट गरिए अनुसार पर्दा अभिमुखीकरण फर्कउँछ

नेभिगेटर

`window.navigator` वा `navigator` एक **पढ्न-मात्र** गुण हो र यसले ब्राउजरसँग सम्बन्धित विभिन्न विधिहरू र प्रकार्यहरू समावेश गर्दछ।

आउनुहोस् नेभिगेसनका केही उदाहरणहरू हेर्रौं।

1. **navigator.appName**: यसले ब्राउजर एप्लिकेसनको नाम दिन्छ।

```
navigator.appName;
// "Netscape"
```

> ***नोट:*** "नेटस्केप" आईई ११, क्रोम, फायरफक्स र सफारीको लागि अनुप्रयोग नाम हो।

2. **navigator.cookieEnabled**: ब्राउजरमा कुकी मानमा आधारित बुलियन मान फर्काउँछ ।

```
navigator.cookieEnabled;
//true
```

3. **navigator.platform**: ब्राउजर अपरेटिङ सिस्टमको बारेमा जानकारी प्रदान गर्दछ ।

```
navigator.patform;
"MacIntel"
```

कुकीहरू

कुकीहरू जानकारीका टुक्राहरू हुन् जुन कम्प्युटरमा भण्डारण गरिन्छ र ब्राउजरद्वारा पहुँच गर्न सकिन्छ।

वेब ब्राउजर र सर्भरबीचको सञ्चार स्टेटलेस अर्थ हो कि यसले प्रत्येक अनुरोधलाई स्वतन्त्र रूपमा व्यवहार गर्दछ। त्यहाँ केसहरू छन् जहाँ हामीले प्रयोगकर्ता जानकारी भण्डारण गर्न र ब्राउजरमा त्यो जानकारी उपलब्ध गराउन आवश्यक छ। कुकीजको साथ, आवश्यक पर्दा कम्प्युटरबाट जानकारी प्राप्त गर्न सकिन्छ।

कुकीहरू नाम-मान जोडीमा बचत गरिन्छ।

```
book = Learn Javascript
```

कुकीहरू सिर्जना गर्न, पढ्न र मेट्न `document.cookie` गुण प्रयोग गरिन्छ। कुकी सिर्जना गर्न धेरै सजिलो छ जुन तपाईंलाई नाम र मान प्रदान गर्न आवश्यक छ।

```
document.cookie = "book=Learn Javacript";
```

पूर्वनिर्धारित रूपमा, ब्राउजर बन्द हुँदा कुकी मेटिन्छ। यसलाई निरन्तर बनाउन, हामीले समाप्ति मिति (यूटीसी समयमा) निर्दिष्ट गर्न आवश्यक छ।

```
document.cookie = "book=Learn Javacript; expires=Fri, 08 Jan 2022 12:00:00 UTC";
```

कुकी कुन मार्गसँग सम्बन्धित छ भनेर बताउन हामी एक प्यारामिटर थप्न सक्छौं। पूर्वनिर्धारित रूपमा, कुकी हालको पृष्ठसँग सम्बन्धित छ।

```
document.cookie = "book=Learn Javacript; expires=Fri, 08 Jan 2022 12:00:00 UTC; path=/";
```

यहाँ एक कुकी को एक सरल उदाहरण दिइएको छ।

```
var cookies = document.cookie;
// a simple way to reterive all cookie.

document.cookie = "book=Learn Javacript; expires=Fri, 08 Jan 2022 12:00:00 UTC; path=/";
// setting up a cookie
```

इतिहास

जब हामी वेब ब्राउजर खोल्छौं र वेब पृष्ठ सर्फ गर्छौं यसले इतिहास स्ट्याकमा नयाँ प्रविष्टि सिर्जना गर्दछ। जब हामी विभिन्न पृष्ठहरूमा नेभिगेट गरिरहन्छौं नयाँ प्रविष्टिहरू स्ट्याकमा धकेलिन्छन्।

हामीले प्रयोग गर्न सक्ने इतिहास वस्तुमा पहुँच गर्न

```
window.history
// or
history
```

विभिन्न इतिहास स्ट्याकबीच नेभिगेट गर्न हामी **इतिहास** वस्तुको `go()` , `foward()` र `back()` विधिहरू प्रयोग गर्न सक्छौं।

1. **go()**: यो इतिहास स्ट्याकको विशिष्ट यूआरएल नेभिगेट गर्न प्रयोग गरिन्छ।

   ```
   history.go(-1); // moves page backward
   history.go(0);  // refreshes the current page
   history.go(); // refreshes the current page
   history.go(1) // moves page forward
   ```

   > ***Note:*** इतिहास स्ट्याकमा हालको पृष्ठ स्थिति **0** हो।

2. **back()** : पृष्ठ पछाडि नेभिगेट गर्न हामी `back()` विधि प्रयोग गर्दछौं।

   ```
   history.back();
   ```

3. **forward()**: यसले ब्राउजर इतिहासमा पछिल्लो सूची लोड गर्दछ । यो ब्राउजरमा फरवार्ड बटन क्लिक गर्नु जस्तै हो ।

   ```
   history.forward();
   ```

स्थान

'स्थान' वस्तु कागजातको हालको स्थान (यूआरएल) पुन: प्राप्त गर्न प्रयोग गरिन्छ र कागजात स्थान हेरफेर गर्न विभिन्न विधिहरू प्रदान गर्दछ। एक द्वारा हालको स्थान पहुँच गर्न सकिन्छ

```
window.location
//or
document.location
//or
location
```

> *नोट*: `window.location` र `document.location` ले एउटै स्थान वस्तुलाई सन्दर्भ गर्दछ।

निम्न यूआरएलको उदाहरण लिऔं र 'स्थान' को विभिन्न गुणहरू अन्वेषण गरौं

`http://localhost:3000/js/index.html?type=listing&page=2#title`

```
location.href // हालको कागजात यूआरएल मुद्रण गर्दछ
location.protocol // http: वा https: जस्तै प्रोटोकल मुद्रण गर्दछ
location.host // लोकलहोस्ट वा लोकलहोस्ट जस्तै पोर्टसँग होस्टनाम मुद्रण गर्दछ:३०००
location.hostname // स्थानीय होस्ट वा www.example.com जस्तै होस्टनाम मुद्रण गर्दछ
location.port // ३००० जस्तै पोर्ट नम्बर मुद्रण गर्दछ
location.pathname // जस्तै /js/index.html जस्तै पाथनाम मुद्रण गर्दछ
location.search // क्वेरी स्ट्रिङ मुद्रण गर्दछ जस्तै ?प्रकार=सूचीकरण र पृष्ठ=२
location.hash // #title जस्तै खण्ड पहिचानकर्ता मुद्रण गर्दछ
```

घटनाहरू

प्रोग्रामिंगमा, घटनाहरू प्रणालीमा कार्यहरू वा घटनाहरू हुन् जुन प्रणालीले तपाईंलाई सूचित गर्दछ ताकि तपाईं तिनीहरूलाई प्रतिक्रिया दिन सक्नुहुनेछ। उदाहरणका लागि, जब तपाईं रिसेट बटन क्लिक गर्नुहुन्छ यसले इनपुट खाली गर्दछ।

कुञ्जीपाटीबाट (keyboard) अन्तर्क्रियाहरू जस्तै किप्रेसहरू फेरि जारी गर्नु अघि कुञ्जीको अवस्था समात्न निरन्तर पढ्न आवश्यक छ। अन्य समय-गहन गणनाहरू प्रदर्शन गर्दा तपाईंलाई कुञ्जी प्रेस छुट्न सक्छ। यो केही आदिम मेशिनहरूको इनपुट ह्यान्डलिङ संयन्त्र थियो। अर्को कदम एक लाइन प्रयोग गर्न छ, अर्थात् एक प्रोग्राम जुन समय-समयमा नयाँ घटनाहरूको लागि लाइन जाँच गर्दछ र यसमा प्रतिक्रिया गर्दछ। यो दृष्टिकोणलाई *polling* भनिन्छ।

यस दृष्टिकोणको मुख्य दोष यो हो कि यसले हरेक समय लाइनमा हेर्नुपर्छ, जब एक घटना ट्रिगर हुन्छ तब व्यवधान उत्पन्न हुन्छ। यसको लागि राम्रो संयन्त्र भनेको घटना घट्दा कोडलाई सूचित गर्नु हो। यो आधुनिक ब्राउजरहरूले हामीलाई विशिष्ट घटनाहरूको लागि *ह्यान्डलरहरूको* रूपमा प्रकार्यहरू दर्ता गर्न अनुमति दिएर के गर्दछ।

```
<p>Click me to activate the handler.</p>
<p>ह्यान्डलर सक्रिय पार्न मलाई क्लिक गर्नुहोस् ।</p>
<script>
  window.addEventListener("click", () => {
    console.log("clicked");
  });
</script>
```

यहाँ, `एडइभेन्ट लिस्टनर (addEventListener)` लाई `सञ्झ्याल (window)` वस्तु (ब्राउजरद्वारा प्रदान गरिएको बिल्ट-इन वस्तु) मा सम्पूर्ण `सञ्झ्याल` का लागि ह्यान्डलर दर्ता गर्न भनिन्छ। यसको `एडइभेन्ट लिस्टनर` विधिलाई कल गर्दा यसको पहिलो तर्कद्वारा वर्णन गरिएको घटना हुँदा दोस्रो तर्क लाई भनिन्छ।

> घटना श्रोताहरूलाई तब मात्र बोलाइन्छ जब घटना उनीहरूले दर्ता गरेको वस्तुको सन्दर्भमा हुन्छ।

केहि सामान्य HTML घटनाहरू उल्लेख गरिएका छन्।

Event	Description
`onchange`	When the user changes or modifies the value of form input
`onclick`	When the user clicks on the element
`onmouseover`	When cursor of the mouse comes over the element
`onmouseout`	When cursor of the mouse comes leaves the element
`onkeydown`	When the user press and then releases the key
`onload`	When the browser has finished the loading

घटना	विवरण
`onchange`	जब प्रयोगकर्ताले फारम आगतको मान परिवर्तन वा परिमार्जन गर्दछ
`onclick`	जब प्रयोगकर्ताले तत्वमा क्लिक गर्दछ
`onmouseover`	जब माउसको कर्सर तत्वमाथि आउँछ
`onmouseout`	जब माउसको कर्सर आउँछ तत्व छोड्छ
`onkeydown`	जब प्रयोगकर्ताले प्रेस गर्दछ र त्यसपछि कुञ्जी जारी गर्दछ
`onload`	जब ब्राउजरले लोडिङ समाप्त गरेको छ

नोड्समा दर्ता गरिएका ह्यान्डलरहरूले पनि बच्चाहरूबाट घटनाहरू प्राप्त गर्न सक्छन। उदाहरणका लागि, यदि अनुच्छेद भित्रको बटन क्लिक गरिएको छ भने, अनुच्छेदमा दर्ता गरिएका ह्यान्डलरहरूले पनि क्लिक घटना प्राप्त गर्नेछन्। दुबैमा ह्यान्डलरहरूको उपस्थितिको मामलामा, तलको एक पहिले जान्छ। यो घटनालाई सुरुआत नोडदेखि यसको मूल नोडसम्म र कागजातको मूलमा बाहिरी रूपमा प्रचार गर्न भनिएको छ।

इभेन्ट ह्यान्डलरले ह्यान्डलरहरूलाई घटना प्राप्त गर्नबाट रोक्नका लागि घटना वस्तुमा `stopPropagation` विधिलाई कल गर्न सक्दछ। यो मामलाहरूमा उपयोगी छ जस्तै, तपाइँसँग क्लिक गर्न योग्य तत्व भित्र एक बटन छ र तपाईं बटन क्लिकबाट बाह्य तत्वको क्लिक योग्य व्यवहार ट्रिगर गर्न चाहनुहुन्न।

```
<p>A paragraph with a <button>button</button>.</p>
<script>
  let para = document.querySelector("p"),
      button = document.querySelector("button");
  para.addEventListener("mousedown", () => {
    console.log("Paragraph handler.");
  });
  button.addEventListener("mousedown", event => {
    console.log("Button handler.");
    event.stopPropagation();
  });
</script>
```

यहाँ, " `माउसडाउन` " ह्यान्डलरहरू अनुच्छेद र बटन दुवैद्वारा दर्ता गरिन्छ। बटन क्लिक गरेपछि, बटनको लागि ह्यान्डलरले `स्टपप्रोपेगेसन` कल गर्दछ, जसले अनुच्छेदमा ह्यान्डलरलाई चल्नबाट रोक्नेछ।

घटनाहरू पूर्वनिर्धारित व्यवहार हुन सक्छन्। उदाहरणका लागि, लिङ्कहरू क्लिक मा लिङ्कको लक्ष्यमा नेभिगेट गर्नुहोस्, तपाईं तल तीर क्लिक गरेपछि पृष्ठको तल नेभिगेट गर्नुहुन्छ, र यति मा। यी पूर्वनिर्धारित व्यवहारहरू घटना वस्तुमा `preventDefault` विधिकल गरेर रोक्न सकिन्छ।

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  let link = document.querySelector("a");
  link.addEventListener("click", event => {
    console.log("Nope.");
    event.preventDefault();
  });
</script>
```

यहाँ, क्लिक मा लिङ्क को पूर्वनिर्धारित व्यवहार रोकिन्छ, अर्थात् लिङ्क 'लक्ष्य तिर नेभिगेट।

प्रतिज्ञा, असिंक/प्रतीक्षा

कल्पना गर्नुहोस् कि तपाईं एक लोकप्रिय पुस्तक लेखक हुनुहुन्छ, र तपाईं एक निश्चित दिनमा नयाँ पुस्तक जारी गर्ने योजना बनाउँदै हुनुहुन्छ। यस पुस्तकमा रुचि राख्ने पाठकहरूले यस पुस्तकलाई आफ्नो इच्छासूचीमा थपिरहेका छन् र प्रकाशित हुँदा वा प्रकाशन को दिन पनि स्थगित भएको अवस्थामा सूचित गरिन्छ। रिलीजको दिन, सबैलाई सूचित गरिन्छ र सबै पक्षलाई खुसी पार्ने पुस्तक किन्न सकिन्छ। यो एक वास्तविक जीवन सादृश्य हो जुन प्रोग्रामिंगमा हुन्छ।

1. "*producing code*" भनेको समय लाग्ने र कुनै कुरा पूरा गर्ने कुरा हो। यहाँ यो एक पुस्तक लेखक हो।
2. "*consuming code*" ले "उत्पादन कोड" एक पटक तयार भएपछि उपभोग गर्दछ। यस मामला मा, यो एक "पाठक" हो।
3. "*producing code*" र "*consuming code*" बीचको सम्बन्धलाई *प्रतिज्ञा* भन्न सकिन्छ किनकि यसले "*producing code*" बाट "*consuming code*" मा परिणामहरू प्राप्त गर्ने आश्वासन दिन्छ।

हामीले बनाएको सादृश्य जाभास्क्रिप्ट 'प्रतिज्ञा' वस्तुको लागि पनि सत्य छ। 'प्रतिज्ञा' वस्तुका लागि कन्स्ट्रक्टर वाक्यविन्यास हो।

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code, "writer")
});
```

यहाँ, एउटा प्रकार्य 'नयाँ प्रतिज्ञा' मा पास गरिएको छ जसलाई *निष्पादक* पनि भनिन्छ, र सिर्जनामा स्वचालित रूपमा चल्छ। यसले उत्पादन कोड समावेश गर्दछ जुन परिणाम दिन्छ। 'रिजोल्भ' र 'रिजेक्ट' जाभास्क्रिप्टद्वारा नै प्रदान गरिएका तर्कहरू हुन् र परिणामहरूमा यीमध्ये एक भनिन्छ।

- `resolve(value):` एक कलब्याक प्रकार्य जसले परिणाममा 'मान' फर्काउँछ
- `reject(error)` : कलब्याक प्रकार्य जसले त्रुटिमा 'त्रुटि' फर्काउँछ, यसले त्रुटि वस्तु फर्काउँछ

'नयाँ प्रतिज्ञा (`new Promise`)' कन्स्ट्रक्टरद्वारा फिर्ता गरिएको 'प्रतिज्ञा' वस्तुको आन्तरिक गुणहरू निम्नानुसार छन्:

- result - initially `undefined` , then changes to `value` upon `resolve` or `error` when `reject` is called (सुरुमा 'अपरिभाषित', त्यसपछि 'रिजेक्ट' गर्दा 'रिजोल्युसन' वा 'त्रुटि' मा 'मान' मा परिवर्तन हुन्छ जब 'अस्वीकार' भनिन्छ)

Promise with resolve and reject callbacks

एक प्रतिज्ञा गुणहरू पहुँच गर्न सक्दैन: `स्टेट` र `परिणाम` । प्रतिज्ञाहरू सम्हाल्न प्रतिज्ञा विधिहरू आवश्यक छन्।

Example of a promise.

प्रतिज्ञा को उदाहरण।

```
let promiseOne = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1-second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
})

let promiseTwo = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1-second signal that the job is done with the result "error"
  setTimeout(() => reject(new Error("Whoops!")), 1000);
})
```

यहाँ, `promiseOne` एक "*fulfilled promise*" को एक उदाहरण हो किनकि यसले सफलतापूर्वक मानहरू समाधान गर्दछ, जबकि `promiseTwo` एक "*rejected promise*" हो किनकि यो अस्वीकार हुन्छ। अस्वीकृत वा समाधान गरिएको प्रतिज्ञालाई सुरुमा पेन्डिङ गरिएको प्रतिज्ञाको विपरीत *सेटल* प्रतिज्ञा भनिन्छ। प्रतिज्ञाबाट प्रकार्य उपभोग गर्दा `.then` र `.catch` विधिहरू प्रयोग गरेर दर्ता गर्न सकिन्छ। हामी पहिलेका विधिहरू पूरा भएपछि सफाई वा अन्तिम रूप दिनका लागि `.finally` विधि पनि थप्न सक्छौं।

```
let promiseOne = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promiseOne.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);

let promiseTwo = new Promise(function(resolve, reject) {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject runs the second function in .then
promiseTwo.then(
  result => alert(result), // doesn't run
  error => alert(error) // shows "Error: Whoops!" after 1 second
);

let promiseThree = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// .catch(f) is the same as promise.then(null, f)
promiseThree.catch(alert); // shows "Error: Whoops!" after 1 second
```

`Promise.then()` विधिमा, दुबै कलब्याक तर्कहरू वैकल्पिक छन्।

असिंक/प्रतिक्षा

प्रतिज्ञाहरूको साथ, एक `एसिंक` कीवर्ड प्रयोग गर्न सक्दछ एक एसिंक्रोनस प्रकार्य घोषणा गर्न जसले प्रतिज्ञा फर्काउँछ जबकि `प्रतीक्षा` वाक्यरचनाले जाभास्क्रिप्टलाई प्रतीक्षा गर्दछ जबसम्म त्यो प्रतिज्ञा व्यवस्थित हुँदैन र यसको मान फर्काउँछ। यी कुञ्जीशब्दहरूले प्रतिज्ञाहरू लेख्न सजिलो बनाउँदछ। एसिंकको एक उदाहरण तल देखाइएको छ।

```
//async function f
async function f() {
  return 1;
}
// promise being resolved
f().then(alert); // 1
```

माथिको उदाहरण निम्नको रूपमा लेख्न सकिन्छ:

```
function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

`एसिंक` ले यो सुनिश्चित गर्दछ कि प्रकार्यले एक प्रतिज्ञा फर्काउँछ, र यसमा गैर-प्रतिज्ञाहरू लपेट्दछ। `प्रतीक्षा` को साथ, हामी जाभास्क्रिप्टलाई प्रतिक्षा गर्न सक्दछौं जबसम्म प्रतिज्ञा यसको मूल्य फिर्ता को साथ व्यवस्थित हुँदैन।

```
async function f() {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Welcome to Learn JavaScript!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)
  alert(result); // "Welcome to Learn JavaScript!"
}

f();
```

`प्रतिक्षा` कुञ्जीशब्द केवल `एसिंक` प्रकार्य भित्र प्रयोग गर्न सकिन्छ।

टेम्प्लेट शाब्दिक

टेम्प्लेट शाब्दिक हरू ब्याकटिक (``) को साथ डिलेमिनेटेड शाब्दिकहरू हुन् र स्ट्रिंगहरूमा चर र अभिव्यक्ति प्रक्षेपमा प्रयोग गरिन्छ।

```
let text = `Hello World!`;
// template literals with both single and double code inside a single string
let text = `He's often called "Johnny"`;
// template literals with multiline strings
let text =
`The quick
brown fox
jumps over
the lazy dog`;

// template literals with variable interpolation
const firstName = "John";
const lastName = "Doe";

const welcomeText = `Welcome ${firstName}, ${lastName}!`;

// template literals with expression interpolation
const price = 10;
const VAT = 0.25;

const total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
```

होइस्टिंग

होस्टिङ जाभास्क्रिप्टमा पूर्वनिर्धारित व्यवहार हो जुन शीर्षमा घोषणाहरू सार्ने छ। एक कोड कार्यान्वयन गर्दा, यसले विश्वव्यापी कार्यान्वयन सन्दर्भ सिर्जना गर्दछ: सिर्जना र कार्यान्वयन। सिर्जना चरणमा, जाभास्क्रिप्टले चर र प्रकार्य घोषणालाई पृष्ठको शीर्षमा सार्दछ, जुन उत्थानको रूपमा चिनिन्छ।

```
// variable hoisting
console.log(counter);
let counter = 1; // throws ReferenceError: Cannot access 'counter' before initialization
```

यद्यपि 'काउन्टर' ढेर मेमोरीमा अवस्थित छ तर सुरु गरिएको छैन, यसले त्रुटि फ्याँक्छ। यो फहराउने कारण हुन्छ, 'काउन्टर' चर यहाँ फहराइन्छ।;

```
// function hoisting
const x = 20,
    y = 10;

let result = add(x,y); // ❌ Uncaught ReferenceError: add is not defined
console.log(result);

let add = (x, y) => x + y;
```

यहाँ, `add` प्रकार्य निस्कियो र विस्तृत जानकारीको रूपमा` अपरिभाइनको साथ आरम्भ गरिएको छ। यसैले, त्रुटि फ्याँक्दै।

करिङ

'करिङ' कार्यात्मक प्रोग्रामिंगमा एक उन्नत प्रविधि हो जुन धेरै तर्कहरूको साथ एक प्रकार्यलाई नेस्टिंग प्रकार्यहरूको अनुक्रममा रूपान्तरण गर्दछ। यसले प्रकार्यलाई `f(a,b,c)` बाट `f(a)(b)(c)` को रूपमा कल गर्न योग्यमा रूपान्तरण गर्दछ। यसले प्रकार्यलाई कल गर्दैन बरु यसलाई रूपान्तरण गर्दछ।

करिङको राम्रो समझ प्राप्त गर्न एक साधारण `add` प्रकार्य थप्नुहोस् सिर्जना गरौं जुन तीन तर्कहरू लिन्छ र तिनीहरूको योगफल फर्काउँछ। त्यसपछि, हामी एक `addCurry` प्रकार्य सिर्जना गर्दछौं जसले एकल इनपुट लिन्छ र यसको योगफलको साथ प्रकार्यहरूको श्रृंखला फर्काउँछ।

```
// Noncurried version
const add = (a, b, c)=>{
    return a+ b + c
}
console.log(add(2, 3, 5)) // 10

// Curried version
const addCurry = (a) => {
    return (b)=>{
        return (c)=>{
            return a+b+c
        }
    }
}
console.log(addCurry(2)(3)(5)) // 10
```

यहाँ, हामी देख्न सक्छौं कि दुवै करीड र गैर-करी संस्करणहरूले समान परिणाम फर्काए। करी गर्नु धेरै कारणहरूको लागि लाभदायक हुन सक्छ, जसमध्ये केही यहाँ उल्लेख गरिएको छ।

- यसले एउटै चरलाई बारम्बार पास गर्नबाट जोगिन मद्दत गर्दछ।
- यसले प्रकार्यलाई एकल जिम्मेवारीको साथ सानो खण्डहरूमा विभाजन गर्दछ, जसले प्रकार्यलाई कम त्रुटि-प्रवण बनाउँदछ।
- यो एक उच्च-क्रम प्रकार्य सिर्जना गर्न कार्यात्मक प्रोग्रामिंगमा प्रयोग गरिन्छ।

पोलिफिल र ट्रान्सपाइलरहरू

जाभास्क्रिप्टमा नियमित रूपमा, नयाँ भाषा प्रस्तावहरू प्रस्तुत गरिन्छ, विश्लेषण गरिन्छ, र https://tc39.github.io/ecma262/ मा थपिन्छ र त्यसपछि स्पेसिफिकेसनमा समावेश गरिन्छ। ब्राउजरको आधारमा जाभास्क्रिप्ट इन्जिनमा यसलाई कसरी लागू गरिन्छ भन्ने मा भिन्नता हुन सक्छ। कोही-कोहीले ड्राफ्ट प्रस्तावहरू लागू गर्न सक्छन्, जबकि अरूले सम्पूर्ण स्पेसिफिकेसन जारी नभएसम्म पर्खन्छन्। पश्चगामी अनुकूलता मुद्दाहरू नयाँ चीजहरू प्रस्तुत हुँदा उत्पन्न हुन्छन्।

पुरानो ब्राउजरमा आधुनिक कोडसमर्थन गर्न हामी दुई उपकरणहरू प्रयोग गर्दछौं: 'ट्रान्सपाइलर' र 'पोलिफिल'

ट्रान्सपाइलर

यो एक प्रोग्राम हो जसले आधुनिक कोडअनुवाद गर्दछ र पुरानो वाक्य संरचना निर्माणहरू प्रयोग गरेर यसलाई पुनर्लेखन गर्दछ, ताकि पुरानो इन्जिनले यसलाई बुझ्न सकोस्। उदाहरणका लागि, " `nullish` कोलेसिङ अपरेटर" '?? ' 2020 मा प्रस्तुत गरिएको थियो, र पुरानो ब्राउजरहरूले यसलाई बुझ्न सक्दैनन्

अब, यो ट्रान्सपाइलरको काम हो कि 'शून्य' कोलेसिंग अपरेटर " '?? ' पुरानो ब्राउजरलाई बुझ्न सकिने

```
// before running the transpiler
height = height ?? 200;

// after running the transpiler
height = height !== undefined && height !== null ? height : 200;
```

> बाबेल सबैभन्दा प्रमुख ट्रान्सपिलरहरू मध्ये एक हो। विकास प्रक्रियामा, हामी वेबप्याक वा पार्सल जस्ता निर्माण उपकरणहरू प्रयोग गर्न सक्दछौं।

पोलिफिल

पुरानो ब्राउजर इन्जिनहरूमा नयाँ कार्यक्षमता उपलब्ध छैन। यस अवस्थामा, नयाँ कार्यक्षमता प्रयोग गर्ने कोडले काम गर्दैन। खाली स्थानहरू भर्न, हामी हराएको कार्यक्षमता थप्छौं जसलाई " `पोलिफिल` भनिन्छ। उदाहरणका लागि, `filter()` विधि ईएस 5 मा पेश गरिएको थियो र केहि पुरानो ब्राउजरहरूमा समर्थित छैन। यो विधिले प्रकार्य स्वीकार गर्दछ र मूल सरणीको मान मात्र भएको सरणी फर्काउँछ जसका लागि प्रकार्यले 'सत्य' फर्काउँछ ।

```
const arr = [1, 2, 3, 4, 5, 6];
const filtered = arr.filter((e) => e % 2 === 0); // filter outs the even number
console.log(filtered);

// [2, 4, 6]
```

फिल्टरको लागि पोलिफिल:

```
Array.prototype.filter = function (callback) {
    // Store the new array
    const result = [];
    for (let i = 0; i < this.length; i++) {
        // call the callback with the current element, index, and context.
        //if it passes the text then add the element in the new array.
        if (callback(this[i], i, this)) {
            result.push(this[i]);
        }
    }
    //return the array
    return result;
};
```

> caniuse ले विभिन्न ब्राउजर इन्जिनहरू द्वारा समर्थित अद्यावधिक कार्यक्षमता र वाक्य रचना देखाउँदछ।

लिङ्क गरिएको सूची

यो सबै प्रोग्रामिंग भाषाहरूमा पाइने एक सामान्य डेटा संरचना हो। लिङ्क गरिएको सूची जाभास्क्रिप्टमा सामान्य सरणीसँग धेरै मिल्दोजुल्दो छ, यसले थोरै फरक तरिकाले कार्य गर्दछ।

यहाँ सूचीको प्रत्येक तत्व एक अलग वस्तु हो जसमा लिङ्क वा अर्कोमा सूचक हुन्छ। जाभास्क्रिप्टमा लिङ्क गरिएको सूचीहरूको लागि कुनै बिल्ट-इन विधि वा प्रकार्य छैन त्यसैले यसलाई कार्यान्वयन गर्नु पर्दछ। लिङ्क गरिएको सूचीको उदाहरण तल देखाइएको छ।

```
["one", "two", "three", "four"]
```

लिंक गरिएको सूचीको प्रकार

त्यहाँ तीन प्रकारका लिङ्क गरिएका सूचीहरू छन्:

1. **एकल लिङ्क सूचीहरू:** प्रत्येक नोडले अर्को नोडमा केवल एक सूचक समावेश गर्दछ।
2. **डबल लिङ्क गरिएको सूचीहरू:** प्रत्येक नोडमा दुई पोइन्टरहरू छन्, अर्को नोडमा एक र अघिल्लो नोडमा एक।
3. **सर्कुलर लिङ्क गरिएको सूचीहरू:** एक परिपत्र लिङ्क गरिएको सूचीले अन्तिम नोडलाई पहिलो नोड वा यसको अगाडि कुनै अन्य नोडलाई इंगित गरेर लूप बनाउँदछ।

जोड्नु

लिङ्क गरिएको सूचीमा मान थप्न यहाँ `add` विधि सिर्जना गरिएको छ ।

```
class Node {
    constructor(data) {
        this.data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this.head = head
    }
    append = (value) => {
        const newNode = new Node(value)
        let current = this.head
        if (!this.head) {
            this.head = newNode
            return
        }
        while (current.next) {
            current = current.next
        }
        current.next = newNode
    }
}
```

पप

यहाँ, लिङ्क गरिएको सूचीबाट मान हटाउन 'पप' विधि सिर्जना गरिएको छ।

```
class Node {
    constructor(data) {
        this.data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this.head = head
    }
    pop = () => {
        let current = this.head
        while (current.next.next) {
            current = current.next
        }
        current.next = current.next.next
    }
}
```

प्रिपेन्ड

यहाँ, लिङ्क गरिएको सूचीको पहिलो बच्चा भन्दा पहिले मान थप्न 'प्रिपेन्ड' विधि सिर्जना गरिएको छ।

```
class Node {
    constructor(data) {
        this.data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this.head = head
    }
    prepend = (value) => {
        const newNode = new Node(value)
        if (!this.head) {
            this.head = newNode
        }
        else {
            newNode.next = this.head
            this.head = newNode
        }
    }
}
```

सिफ्ट

यहाँ, लिङ्क गरिएको सूचीको पहिलो तत्व हटाउन 'सिफ्ट' विधि सिर्जना गरिएको छ।

```
class Node {
    constructor(data) {
        this.data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this.head = head
    }
    shift = () => {
        this.head = this.head.next
    }
}
```

ग्लोबल पदचिह्न

यदि तपाईं मोड्युल विकास गर्दै हुनुहुन्छ, जुन वेब पृष्ठमा चलिरहेको हुन सक्छ, जसले अन्य मोड्युलहरू पनि चलाउँदछ, त्यसपछि तपाईंले चर नाम ओभरल्यापिङको बारेमा सावधान हुनुपर्दछ।

मानौं हामी काउन्टर मोड्युल विकास गर्दैछौं:

```
let myCounter = {
  number: 0,
  plusPlus: function () {
    this.number = this.number + 1;
  },
  isGreaterThanTen: function () {
    return this.number > 10;
  },
};
```

> ***नोट:*** आन्तरिक अवस्थालाई बाहिरबाट अपरिवर्तनीय बनाउन यो प्रविधि प्रायः बन्द गरेर प्रयोग गरिन्छ।

मोड्युलले अब केवल एक चर नाम लिन्छ - `myCounter` । यदि पृष्ठमा कुनै अन्य मोड्युलले 'नम्बर' वा `isGreaterThanTen` जस्ता नामहरू प्रयोग गर्दछ भने यो पूर्ण रूपमा सुरक्षित छ किनकि हामी एक अर्काको मानहरू ओभरराइड गर्नेछैनौं।

डिबगिंग

प्रोग्रामिंगमा, कोड लेख्दा त्रुटिहरू हुन सक्छ। यो वाक्यात्मक वा तार्किक त्रुटिहरूको कारण हुन सक्छ। त्रुटिहरू फेला पार्ने प्रक्रिया समय-खपत र पेचिलो हुन सक्छ र यसलाई कोड डिबगिंग भनिन्छ।

सौभाग्यवश, अधिकांश आधुनिक ब्राउजरहरू बिल्ट-इन डिबगरहरूसँग आउँछन्। यी डिबगरहरू स्विच अन र अफ गर्न सकिन्छ, जसले त्रुटिहरू रिपोर्ट गर्न बाध्य पार्छ। कार्यान्वयन रोक्न र चरहरूको जाँच गर्न कोडको कार्यान्वयनको क्रममा ब्रेकपोइन्टहरू सेट अप गर्न पनि सम्भव छ। यसको लागि डिबगिङ विन्डो खोल्नुपर्छ र जाभास्क्रिप्ट कोडमा 'डिबगर' कीवर्ड राख्नुपर्छ। कोड कार्यान्वयन प्रत्येक ब्रेकपोइन्टमा रोकिन्छ, विकासकर्ताहरूलाई जाभास्क्रिप्ट मानहरू जाँच गर्न र कोडको कार्यान्वयन पुनः सुरु गर्न अनुमति दिन्छ।

डिबगर सञ्झ्यालमा जाभास्क्रिप्ट मान मुद्रण गर्न `console.log()` विधि पनि प्रयोग गर्न सकिन्छ ।

```
const a = 5, b = 6;
const c = a + b;
console.log(c);
// Result : c = 11;
```

कन्सोल

जाभास्क्रिप्टमा, हामी कन्सोलमा सन्देश (चरको सामग्री, दिइएको स्ट्रिङ, आदि) लेख्न `console.log()` प्रयोग गर्दछौं। यो मुख्यतया डिबगिंग उद्देश्यहरूको लागि प्रयोग गरिन्छ, आदर्श रूपमा प्रोग्राम को कार्यान्वयनको क्रममा चरहरूको सामग्रीको ट्रेस छोड्नको लागि।

उदाहरण:

```
console.log("Welcome to Learn JavaScript Beginners Edition");
let age = 30;
console.log(age);
```

📝 कार्यहरू:

- [ ] Write a program to print `Hello World` on the console. Feel free to try other things as well!
- [ ] कन्सोलमा 'हेलो वर्ल्ड' प्रिन्ट गर्न प्रोग्राम लेख्नुहोस् । अन्य कुराहरू पनि प्रयास गर्न स्वतन्त्र महसुस गर्नुहोस्!
- [ ] 'कन्सोल' मा चल मुद्रण गर्न प्रोग्राम लेख्नुहोस् ।
    1. एक चर `animal` घोषणा गर्नुहोस् र यसलाई ड्रैगन मान निर्दिष्ट गर्नुहोस्।
    2. `कन्सोल` मा `animal` चर मुद्रण गर्नुहोस् ।

💡 सङ्केतहरू:

- चरहरूको बारेमा थप बुझ्न चर अध्यायमा जानुहोस्।

गुणन

जाभास्क्रिप्टमा, हामी तारांकन `(*)` अंकगणितीय अपरेटरहरू प्रयोग गरेर दुई संख्याहरूको गुणन प्रदर्शन गर्न सक्दछौं।

उदाहरण:

```
let resultingValue = 3 * 2;
```

यहाँ, हामीले `3 * 2` को उत्पादनलाई `resultingValue` चरमा भण्डारण गर्यौं।

📝 कार्यहरू:

- [ ] '23' पटक '41' को उत्पादन भण्डारण गर्न र यसको मान मुद्रण गर्न एक प्रोग्राम लेख्नुहोस्।

💡 सङ्केतहरू:

- गणितीय सञ्चालनहरू बुझ्न आधारभूत अपरेटरहरू अध्यायमा जानुहोस्।

प्रयोगकर्ताहरूबाट इनपुट

जाभास्क्रिप्टमा, हामी प्रयोगकर्ताहरूबाट इनपुट लिन सक्छौं र यसलाई चरको रूपमा प्रयोग गर्न सक्दछौं। तिनीहरूसँग काम गर्न उनीहरूको मूल्य जान्न आवश्यक छैन।

📝 कार्यहरू:

- [ ] प्रयोगकर्ताबाट आगत लिन र त्यसमा `१०` थप्न प्रोग्राम लेख्नुहोस्, र यसको परिणाम मुद्रण गर्नुहोस् ।

💡 सङ्केतहरू:

- चरको सामग्री प्रयोगकर्ताको इनपुटद्वारा निर्धारित गरिन्छ। `prompt()` विधिले स्ट्रिङको रूपमा आगत मान बचत गर्दछ ।
- तपाईंले स्ट्रिङ मान गणनाका लागि पूर्णांकमा रूपान्तरण भएको सुनिश्चित गर्न आवश्यक छ ।
- स्ट्रिङलाई इन्टमा रूपान्तरण गर्ने प्रकारका लागि आधारभूत अपरेटर अध्याय मा जानुहोस् ।

स्थिरहरू

स्थिरहरू ईएस ६ (2015) मा प्रस्तुत गरिएको थियो, र `कन्स्ट` कीवर्ड प्रयोग गर्दछ। `कन्स्ट` सँग घोषणा गरिएका भेरिएबलहरू पुन: असाइन गर्न वा पुन: परिभाषित गर्न सकिँदैन।

उदाहरण

```
const VERSION = "1.2";
```

📝 कार्यहरू:

- [ ] तल उल्लिखित प्रोग्राम चलाउनुहोस् र तपाईंले कन्सोलमा देख्नुभएको त्रुटि ठीक पार्नुहोस् । निश्चित गर्नुहोस् कि कोड परिणाम `०.९` हो जब यो कन्सोलमा फिक्स गरिएको छ।

```
const VERSION = "0.7";
VERSION = "0.9";
console.log(VERSION);
```

💡 सङ्केतहरू:

- कन्स्टको बारेमा थप जानकारीको लागि भेरिएबल्स अध्यायमा जानुहोस् र फिक्स सिक्नका लागि खोजी इन्जिनमा "स्थिर चरमा टाइपइरर असाइनमेन्ट" पनि हेर्नुहोस्।

कन्काटेनेसन

कुनै पनि प्रोग्रामिंग भाषामा, स्ट्रिङ संयोजनको अर्थ केवल एक वा बढी स्ट्रिङहरू अर्को स्ट्रिङमा जोड्नु हो। उदाहरणका लागि, जब स्ट्रिङहरू "*World*" र "*Good Afternoon*" स्ट्रिङ "*Hello*" सँग संयोजन गरिन्छ, तिनीहरूले स्ट्रिङ "*Hello World, Good Afternoon*" बनाउँदछन्। हामी जाभास्क्रिप्टमा धेरै तरिकामा स्ट्रिङ संयोजन गर्न सक्दछौं।

उदाहरण:

```
const icon = "👋";

// using template Strings
`hi ${icon}`;

// using join() Method
["hi", icon].join(" ");

// using concat() Method
"".concat("hi ", icon);

//  using + operator
"hi " + icon;

// RESULT
// hi 👋
```

📝 कार्यहरू:

- [ ] `str1` र `str2` का लागि मानहरू सेट गर्न प्रोग्राम लेख्नुहोस् ताकि कोडले कन्सोलमा '*Hello World*' मुद्रण गर्दछ।

💡 सङ्केतहरू:

- स्ट्रिङ संयोजनको बारेमा थप जानकारीको लागि स्ट्रिङको संयोजन अध्यायमा जानुहोस्।

प्रकार्य

प्रकार्य एक विशिष्ट कार्य प्रदर्शन गर्न डिजाइन गरिएको कोडको ब्लक हो र "केहि" ले यसलाई आह्वान गर्दा निष्पादित हुन्छ। प्रकार्यहरूको बारेमा थप जानकारी प्रकार्यहरू अध्यायमा फेला पार्न सकिन्छ।

📝 कार्यहरू:

- [ ] ४५३४५ नम्बरलाई तर्कको रूपमा पास गर्ने र सङ्ख्या विषम छ वा छैन भनेर निर्धारण गर्ने `isOdd` नामको प्रकार्य सिर्जना गर्न प्रोग्राम लेख्नुहोस् ।
- [ ] परिणाम प्राप्त गर्न यो प्रकार्यलाई कल गर्नुहोस् । परिणाम बुलियन ढाँचामा हुनुपर्दछ र कन्सोलमा सही फर्कनुपर्छ।

💡 सङ्केतहरू:

- प्रकार्यहरू बुझ्न र तिनीहरूलाई कसरी सिर्जना गर्ने भनेर प्रकार्यहरू अध्यायमा जानुहोस्।

सशर्त तर्क

सशर्त तर्क प्रोग्रामिंगमा महत्त्वपूर्ण छ किनकि यसले यो सुनिश्चित गर्दछ कि प्रोग्रामले काम गर्दछ चाहे तपाईले कुन डेटा फ्याँक्नुहुन्छ र शाखाहरू पनि अनुमति दिनुहुन्छ। यो अभ्यास वास्तविक जीवन समस्याहरूमा विभिन्न सशर्त तर्कको कार्यान्वयनको बारेमा हो।

📝 कार्यहरू:

- [ ] `कति किलोमिटर जान बाँकी छ?` प्रोम्प्ट सिर्जना गर्न प्रोग्राम लेख्नुहोस् र प्रयोगकर्ता र निम्न सर्तहरूको आधारमा, कन्सोलमा परिणामहरू मुद्रण गर्नुहोस्।
  - यदि त्यहाँ जान 100 किलोमिटर भन्दा बढी बाँकी छ भने, मुद्रण गर्नुहोस्: `तपाईंसँग अझै पनि ड्राइभिंगको एक बिट बाँकी छ ।`
  - यदि त्यहाँ 50 किलोमिटर भन्दा बढी छ, तर 100 किलोमिटर भन्दा कम वा बराबर छ भने, मुद्रण गर्नुहोस्: `म 5 मिनेटमा त्यहाँ हुनेछु ।`
  - यदि त्यहाँ 50 किलोमिटर भन्दा कम वा बराबर छ भने, मुद्रण गर्नुहोस्: `म पार्किंग गर्दैछु। म तपाईंलाई अहिले भेट्छु।`

💡 सङ्केतहरू:

- सशर्त तर्क र सशर्त कथनहरू कसरी प्रयोग गर्ने भनेर बुझ्न सशर्त तर्क अध्यायमा जानुहोस्।

वस्तु

वस्तुहरू `कुञ्जी` , `मान` जोडीहरूको संग्रह हो र कुञ्जी-मानको प्रत्येक जोडीलाई सम्पत्तिको रूपमा चिनिन्छ। यहाँ, कुञ्जीको गुण `स्ट्रिङ` हुन सक्छ जबकि यसको `मान` कुनै पनि मूल्यको हुन सक्छ।

📝 कार्यहरू:

एक डो परिवार दिइएको छ जसमा दुई-सदस्य समावेश छन्, जहाँ प्रत्येक सदस्यको जानकारी वस्तुको रूपमा प्रदान गरिएको छ।

```
let person = {
    name: "John",                    //String
    lastName: "Doe",
    age: 35,                         //Number
    gender: "male",
    luckyNumbers: [ 7, 11, 13, 17], //Array
    significantOther: person2       //Object,
};

let person2 = {
    name: "Jane",
    lastName: "Doe",
    age: 38,
    gender: "female",
    luckyNumbers: [ 2, 4, 6, 8],
    significantOther: person
};

let family = {
    lastName: "Doe",
    members: [person, person2]       //Array of objects
};
```

- [ ] `कन्सोल` मा डो परिवारको पहिलो सदस्यको नाम मुद्रण गर्ने तरिका फेला पार्नुहोस्।
- [ ] डो परिवारको दोस्रो सदस्यको चौथो `लक्की नम्बर` लाई `३३` मा परिवर्तन गर्नुहोस्।
- [ ] नयाँ व्यक्ति (`जिमी`, `डो` , `१३` , `पुरुष` , `[१, २, ३, ४]` , `शून्य`) सिर्जना गरेर परिवारमा नयाँ सदस्य थप्नुहोस् र सदस्य सूची अद्यावधिक गर्नुहोस्।
- [ ] `कन्सोल` मा डो परिवारको भाग्यशाली नम्बरको `SUM` मुद्रण गर्नुहोस्

💡 सङ्केतहरू:

- वस्तुले कसरी काम गर्छ भनेर बुझ्न वस्तुहरूको अध्यायमा जानुहोस्।
- तपाईं परिवारको वस्तु भित्र प्रत्येक व्यक्ति वस्तुबाट `luckyNumbers` प्राप्त गर्न सक्नुहुनेछ।
- एक पटक जब तपाईं प्रत्येक सरणी प्राप्त गर्नुहुन्छ बस यसलाई प्रत्येक तत्व थप्दै लूप गर्नुहोस् र त्यसपछि ३ परिवारका सदस्यहरूको प्रत्येक योग थप्नुहोस्।

फिजबज समस्या

फिजबज समस्या सामान्यतया सोधिने प्रश्नहरू मध्ये एक हो, यहाँ केहि शर्तहरूमा फिज र बज प्रिन्ट गर्नु पर्दछ।

📝 कार्यहरू

- [ ] १ देखि १०० को बीचमा सबै नम्बरहरू यसरी मुद्रण गर्न एक प्रोग्राम लेख्नुहोस् कि निम्न सर्तहरू पूरा हुन्छन्।
  - ३ को गुणकको लागि, नम्बरको सट्टा, `फिज` मुद्रण गर्नुहोस्।
  - ५ को गुणकको लागि, `बज` मुद्रण गर्नुहोस्।
  - ३ र ५ दुवैको गुणक भएका नम्बरका लागि `फिजबज` प्रिन्ट गर्नुहोस्।

```
/
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
....
....
98
Fizz
Buzz
/
```

💡 सङ्केतहरू:

- लूपले कसरी काम गर्दछ भनेर बुझ्न लूप अध्यायमा जानुहोस्।

टाइल प्राप्त गर्नुहोस्

टाइल प्राप्त गर्नुहोस्! समस्या एक रोचक समस्या हो जहाँ हामीले पुस्तकहरूको सूचीबाट शीर्षक प्राप्त गर्नुपर्दछ। यो एरे र वस्तुहरूको कार्यान्वयनको लागि एक राम्रो अभ्यास हो।

📝 कार्यहरू:

एक लेखकसंग पुस्तकहरू प्रतिनिधित्व गर्ने वस्तुहरूको एक एर्रे दिइएको छ।

```
const books = [
    {
        title: "Eloquent JavaScript, Third Edition",
        author: "Marijn Haverbeke",
    },
    {
        title: "Practical Modern JavaScript",
        author: "Nicolás Bevacqua",
    },
];
```

- [ ] प्रकार्य `getTheTitles` सिर्जना गर्न प्रोग्राम लेख्नुहोस् जसले एरे लिन्छ र शीर्षकको सरणी फर्काउँछ र `कन्सोल` मा यसको मान मुद्रण गर्दछ।

💡 सङ्केतहरू

- एरे र वस्तुले कसरी काम गर्दछ भनेर बुझ्न एरे र वस्तुहरू अध्यायमा जानुहोस्।