Java Script

NEXT

GENIUS space

План

Функції

Стрілочні функції

Лексичне оточення

Контексти і виклики

Функції

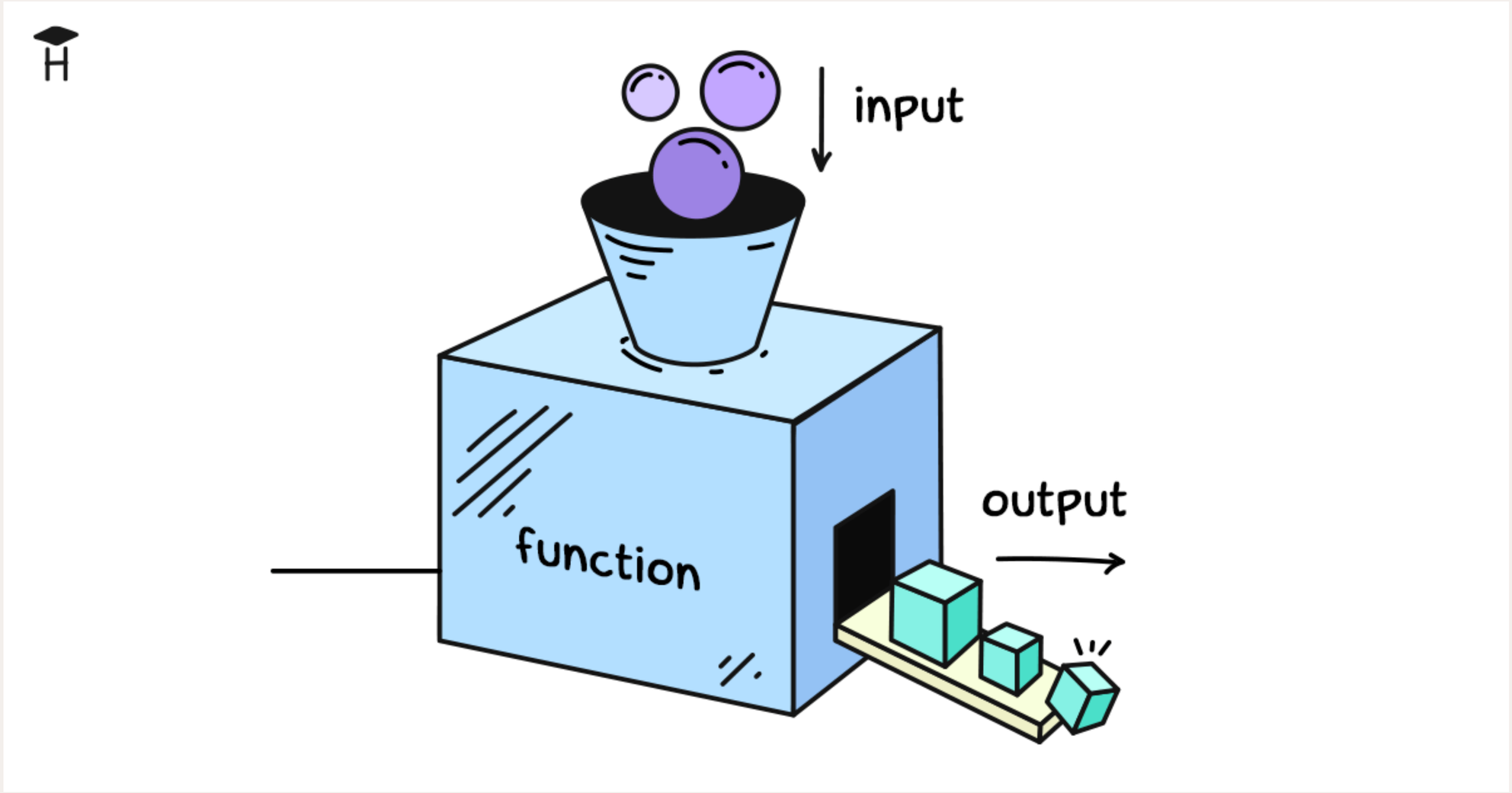
Досить часто нам потрібно виконати однакову дію в декількох місцях програми.

Наприклад, нам треба показати якесь повідомлення, коли користувач входить або виходить з системи і може ще десь.

Функції — це головні “будівельні блоки” програми. Вони дозволяють робити однакові дії багато разів без повторення коду.

Функції можна уявити як чорну скриньку, вони отримують щось на вході (дані), і віддають щось на виході (результат виконання коду всередині функції).

Функції є інструментом для структурування великих програм, зменшення повторень та ізолювання коду.



Параметри

Ми можемо передати в функцію довільні дані використовуючи параметри.

В наступному прикладі, функція має два параметри: from і text.

```
function showMessage(from, text) { // параметри: from, text
  alert(from + ': ' + text);
}
showMessage('Анна', 'Привіт!'); // Анна: Привіт! (*)
```

Якщо викликати функцію без аргументів, тоді відповідні значення стануть undefined.

Функціональний вираз

Функціональний вираз (function expression) - звичайне оголошення змінної, значення якої буде функцією.

Оголосимо змінну add, і надамо їй функцію приймаючу 3 значення і повертає результат складання цих значень.

```
const add = function(a, b, c) {
  return a + b + c;
};
```

Ім'я функції це дія, дієслово, що починається з маленької літери, що відповідає на питання 'Що зробити?'. Наприклад: findSmallesNumber, fetchUserInfo, validateInput.

Визначення функції починається з ключового слова function, за яким може йти необов'язкове ім'я функції.
У круглих дужках йдуть параметри - перерахування даних, які функція буде отримувати з-за. Параметрів може бути кілька або взагалі їх не бути, тоді записуються просто порожні круглі дужки ().

Далі йде тіло функції, укладене у фігурні дужки {}, що містить інструкції, які необхідно виконати при виклику функції. Тіло функції завжди укладають у фігурні дужки, навіть якщо воно складається з однієї інструкції.

Оператор return визначає значення, що повертається. Коли інтерпретатор доходить до return, він відразу ж виходить з функції, і повертає це значення в місце коду, де функція була викликана.

Оператор return без виразу повертає значення undefined. При відсутності обороту в тілі функції вона все одно поверне значення undefined.

Потім, коли необхідно, функція викликається за допомогою імені та круглих дужок, усередині яких можуть бути передані аргументи.

Термін аргументи використовується під час виклику функції, коли ми передаємо значення у функцію.

Термін параметри використовується при оголошенні функції, це ті локальні змінні всередині функції, які будуть записані значення аргументів під час її виклику.

JavaScript важливий порядок оголошення параметрів функції. Немає ніякого іншого механізму пояснити інтерпретатору як значення аргументів функції при виклику пов'язані з параметрами.
Порядок оголошення параметрів відповідає порядку передачі аргументів під час виклику функції: значення першого аргументу буде присвоєно першому параметру, другого аргументу другому параметру і т. д. Якщо параметрів буде менше аргументів, то параметрам без значень буде присвоєно undefined.

Стрілкові функції

Існує ще один простий та короткий синтаксис для створення функцій, який часто доцільніше використовувати замість Функціонального Виразу.

Це так звані “стрілкові функції”, а виглядають вони ось так:

```
let func = (arg1, arg2, ..., argN) => expression;
```

Цей код створить функцію func з аргументами arg1..argN, що обчислює expression з правого боку (використовуючи ці аргументи) та повертає його результат.

Іншими словами, це приблизно те ж саме, що і:

```
let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```

Колбеки (функції зворотного виклику)

Розглянемо інші приклади передачі функції як значення та використання Функціональних Виразів.

Для цього напишемо функцію ask(question, yes, no) з трьома параметрами:

question
Текст запитання

yes
Функція, що буде викликатись, якщо відповідь “Так”

no
Функція, що буде викликатись, якщо відповідь “Ні”

Функція повинна поставити запитання question і, залежно від відповіді користувача, викликати yes() або no():

```
function ask(question, yes, no) {
if (confirm(question)) yes()
else no();
}

function showOk() {
alert( "Ви погодились." );
}

function showCancel() {
alert( "Ви скасували виконання." );
}

// використання: функції showOk, showCancel передаються як аргументи для ask
ask("Ви згодні?", showOk, showCancel);
```

Лексичне оточення (LexicalEnvironment)

Офіційна специфікація ES6 визначає цей термін як:
Lexical Environment – це тип специфікації, який використовується для дозволу імен ідентифікаторів при пошуку конкретних змінних та функцій на основі лексичної структури вкладеності коду ECMAScript. Лексичне оточення (Lexical Environment) складається із запису середовища і, можливо, нульового посилання на зовнішнє Лексичне середовище.
Розберемося докладніше.

Я уявлю собі лексичне оточення як структуру, яка зберігає зв'язок ідентифікаторів контексту зі своїми значенням. Це свого роду сховище змінних, функцій, класів, оголошених у сфері видимості цього контексту.

Технічно ЛО є об'єкт з двома властивостями:

- запис оточення (саме тут зберігаються всі оголошення)
- посилання на ЛО породжуючого контексту.

Через посилання на контекст-батька поточного контексту ми можемо у разі потреби отримати посилання на «контекст-дідусь» контексту-батька і так далі до глобального контексту, посилання на батько якого буде null.
З цього визначення випливає, що Лексичне оточення - це зв'язок сутності з контекстами, що її породили.

```
let x = 10;
let y = 20;
const foo = z => {
  let x = 100;
  return x + y + z;
}

foo(30);
```

поверне 150. ЛО для foo буде виглядати так {record: {z: 30, x: 100}, parent: __parent};
// __parent буде вказувати на глобальне ЛО
{record: {x: 10, y: 20}, parent: null}

Технічно процес дозволу імен ідентифікаторів відбуватиметься так - послідовне опитування об'єктів у ланцюзі ЛВ дoти, доки не буде знайдено потрібний ідентифікатор. Якщо ідентифікатор не знайдено, ReferenceError.

Лексичне оточення створюється та наповнюється на етапі створення контексту. Коли поточний контекст закінчує своє виконання, він видаляється зі стека викликів

Лексичне оточення може продовжувати жити до тих пір, поки на нього є хоч одне посилання. Це одна з переваг сучасного підходу до проектування мов програмування.

Контексти і виклики

При виклику функції всередині її тіла можуть викликатися інші функції, а в них інші і т. д. JavaScript однопоточна (в основному) мова, тобто в одну одиницю часу може виконуватися тільки одна операція.

Це означає, що вже викликані функції, які закінчили своє виконання, повинні чекати виконання функцій викликаних у собі, щоб продовжити своє виконання.

Відповідно необхідний механізм зберігання списку функцій, які були викликані, але ще не закінчили своє виконання, і управління порядком виконання цих функцій, і саме за це відповідає стек контекстів виконання.

Стек – структура даних, яка працює за принципом останнім прийшов – першим вийшов (LIFO – Last In, First Out). Останнє, що ви додали в стек, буде видалено з нього першим, тобто можна додати або видалити елементи лише з верхівки стека.

Контекст виконання (execution context) - внутрішня конструкція мови для відстеження виконання функції, що містить метаінформацію про її виклик.

Глобальний контекст виконання (global execution context) – це контекст є за замовчуванням, сам файл скрипта – це функція яка запускається на виконання. Контекст виконання функції (functional execution context) - створюється щоразу під час виклику функції.

Стек викликів (Execution Context stack, call stack) – внутрішня констуркція двигуна, що містить усі контексти виконання.

Stack frame (кадр стека, запис стека) - структура, яка додається на стек при виклику функції. Зберігає деяку метаінформацію: ім'я функції, аргументи які були передані під час виклику та номер рядка в якому відбувся виклик.

Додаткові матеріали

- https://uk.javascript.info/function-expressions
- https://uk.javascript.info/function-basics
- https://uk.javascript.info/arrow-functions-basics
- https://hackernoon.com/execution-context-in-javascript-319dd72e8e2c
- https://codeguida.com/post/2662
- http://xn--80adth0aefm3i.xn--j1amh/function
- https://webdoky.org/uk/docs/Web/JavaScript/Guide/Functions/
- https://understandinges6.denysdovhan.com/manuscript/03-Functions.html
- https://angularlessons.gitbook.io/javascriptiseasy/rozdil-4.-funkciya

Домашнє завдання

---1---

Перепишіть функцію, використовуючи '?' або '||'

```
function checkAge(age) {
if (age > 18) {
 return true;
} else {
return confirm('Батьки дозволили?');
}
}
```

---2---

Напишіть функцію min(a, b), яка повертає менше з двох чисел a та b.

---3---

Перепишіть з використанням стрілкових функцій
Замініть Функціональні Вирази на стрілкові функції у коді нижче:

```
function ask(question, yes, no) {
if (confirm(question)) yes();
else no();
}

ask(
"Ви згодні?",
function() { alert("Ви погодились."); },
function() { alert("Ви скасували виконання."); }
);
```