JavaScript

wprowadzenie, historia, komentarze, wartości, zmienne, operatory, funkcje, ciągi znaków, wyrażenia regularne, wyrażenia warunkowe, tablice, pętle, obiekty, this, Promise, fetch

Plan zajęć

1. Wprowadzenie
2. Historia
3. Komentarze
4. Wartości
5. Zmienne
6. Operatory
7. Funkcje
8. Ciągi znaków
9. Wyrażenia regularne
10. Wyrażenia warunkowe
11. Tablice
12. Pętle
13. Obiekty
14. This
15. Promise
16. Fetch

Wprowadzenie

JavaScript to **język** wspomagający rozwój stron oraz aplikacji internetowych.

Stworzony przez pracownika Netscape Communications, **Brendana Eicha**.

Wydany wraz z **Netscape Navigator 2.0** w roku 1995.

Historia

1995

Netscape wprowadza **JavaSript** w przeglądarce **Navigator 2.0**

1996

Microsoft wprowadza **JScript** w przeglądarce **Internet Explorer 3.0**

1996

Netscape standaryzuje **JavaSript** w **ECMA International**

Historia

1998 **ECMA Script** wersji **2**

1999 **ECMA Script** wersji **3**

2000 ~~**ECMA Script** wersji **4**~~

2009 **ECMA Script** wersji **5**

2009 **Ryan Dahl** prezentuje **NodeJS**

Komentarze

Komentarze to fragmenty kodu, które w żaden sposób nie wpływają na jego wykonanie. Pełnią rolę dokumentacyjną, a ich stosowanie jest opcjonalne.

Komentarze / Przykłady

```
// To jest komentarz jednolinijkowy


/* To jest komentarz wielolinijkowy

        Lorem ipsum dolor sit amet,
        fugit putent animal in has,
        diam quando at eum.

   Per congue deseruisse ex,
   decore omittantur ex vis.
*/
```

Wartości

W języku JavaScript można posługiwać się pewnymi jasno określonymi typami danych. Przykładowo, wartości typu `Number` to liczby, wartości typu `String` to ciągi znaków, a wartości typu `Boolean` to `true` oraz `false`, itd.

Wartości / Przykłady

`“John Smith”`	ciągi znaków
`25678.99`	liczby
`true`	prawda
`false`	fałsz
`NaN`	Not a Number, np. dzielenie zera przez zero
`null`	“pusta” wartość
`undefined`	brak wartości, np. niezdefiniowana zmienna

Zmienne

Zmienne pozwalają przechowywać rozmaite wartości w pamięci operacyjnej komputera. W języku JavaScript do zmiennych można przypisywać, np. ciągi znaków, liczby, funkcje, wartości boolowskie, tablice, obiekty, itp.

Zmienne / Dozwolone nazwy

1. Nazwa zmiennej może zawierać:

   _ $ liczby małe litery WIELKIE LITERY

2. Nazwa zmiennej nie może zaczynać się liczbą lub operatorem.

3. Nazwa zmiennej nie może być słowem zarezerwowanym:

   if do class with case else itd.

Zmienne / Przykłady

```
var _name = “James”;

var $lastName = “Bond”;

var time1 = 12654345360;

var time2 = 25654365373;

var PI = 3.1415;

var yes = true;
```

Ćwiczenie 1 / Wizytówka

Zadeklaruj zmienne `name`, `lastName` oraz `job`, a następnie przypisz odpowiednio swoje imię, nazwisko i zawód. Wypisz powyższe zmienne do konsoli.

Operatory

Operatory to znaki specjalne, które pozwalają zapisywać i grupować rozmaite działania.

Najczęściej stosuje się je w obliczeniach i wyrażeniach warunkowych.

Operatory

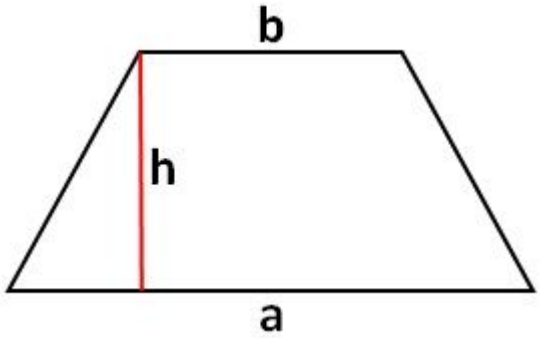
`+`	dodawanie
`++`	inkrementacja
`-`	odejmowanie
`--`	dekrementacja
`*`	mnożenie
`/`	dzielenie
`%`	modulo
`( )`	grupowanie
`==`	równość
`===`	tożsamość
`!=`	różność
`!==`	nietożsamość
`>`	większe
`<`	mniejsze
`>=`	większe lub równe
`<=`	mniejsze lub równe
`? :`	trójkowy

Operatory / Przykłady

```
var value = 27;

// Zmienna total przyjmie wartość 64
var total = (value + 5) * 2;

var woman = true;

// Zmienna title przyjmie wartość “Pani”
var title = woman ? “Pani” : “Pan”;
```

Ćwiczenie 2 / Pole prostokąta

Zadeklaruj zmienne a, b, h oraz `area`. Przypisz do a i b długości boków trapezu. Natomiast do h jego wysokość. Do `area` przypisz równanie na pole tego trapezu, wykorzystując zmienne a, b i h. Wypisz wynik, czyli wartość `area`, do konsoli.



Funkcje

Funkcje to fragmenty kodu, do których można się odwołać, podobnie jak do zmiennych. Dzięki funkcjom, unika się wtórnego pisania tego samego kodu, co z kolei znacząco zwiększa przejrzystość oraz wydajność programu.

Funkcje / Przykłady

```
function sayHi(name) {
    return “Hi, ” + name + “!”;
}

// Zmienna greeting przyjmie wartość “Hi, Jason!”
var greeting = sayHi(“Jason”);
```

Funkcje / Przykłady

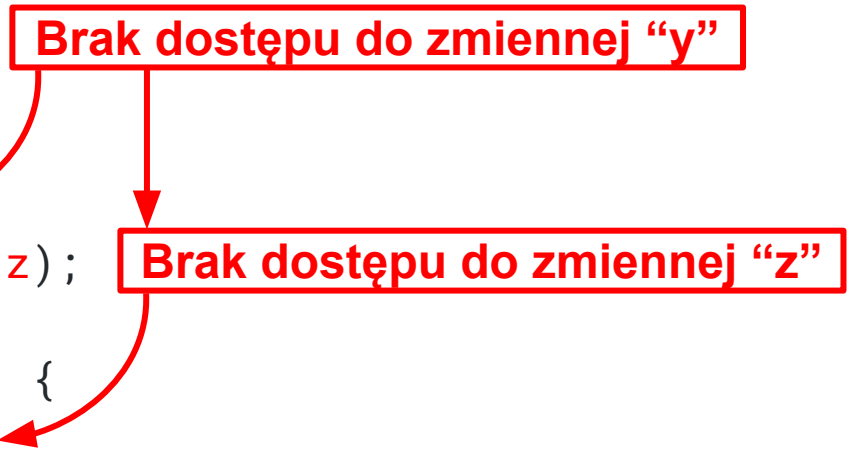
```
function sayHi(name) {
    console.log(“Hi, ” + name + “!”);
}

// Wypisuje “Hi, Jason!” do konsoli
sayHi(“Jason”);
```

Funkcje / Zakres

```
var x = 1111;
console.log(x, y, z);

function foo(name) {
    var y = 2222;
    console.log(x, y, z);

    function bar(name) {
        var z = 3333;
        console.log(x, y, z);
    }
}
```

Funkcje / Zakres

```
var x = 1111;
console.log(x, y, z);

function foo(name) {
    var y = 2222;
    console.log(x, y, z);

    function bar(name) {
        var z = 3333;
        console.log(x, y, z);
    }
}
```



Ćwiczenie 3 / Temperatura

Napisz funkcję `toFahrenheit`, która przyjmuje liczbę °C, a następnie zwraca przeliczony na °F odpowiednik:

```
toFahrenheit(-2) → 28.4
toFahrenheit(25) → 77
toFahrenheit(41) → 105.8
```

Ciągi znaków

Ciągi znaków to najpowszechniej wykorzystywany typ danych w JavaScript.

Umiejętność obróbki tekstu to podstawa w pracy programisty.

Ciągi znaków / Metody

`charAt`	zwraca znak spod wybranego indeksu
`charCodeAt`	zwraca kod znaku spod wybranego indeksu
`concat`	łączy dwa lub więcej ciągów znaków
`includes`	sprawdza obecność wybranego znaku w ciągu
`indexOf`	zwraca pierwszy indeks poszukiwanego znaku
`lastIndexOf`	zwraca ostatni indeks poszukiwanego znaku
`startsWith`	sprawdza obecność wybranych znaków na początku ciągu
`endsWith`	sprawdza obecność wybranych znaków na końcu ciągu

Ciągi znaków / Metody

`replace`	zastępuje wybrane znaki
`search`	wyszukuje wybrane znaki
`slice`	zwraca fragment ciągu znaków
`split`	dzieli ciąg znaków na tablicę
`substr`	zwraca wybrany fragment ciągu znaków
`substring`	zwraca wybrany fragment ciągu znaków
`toLowerCase`	zamienia wielkie litery na małe litery
`toUpperCase`	zamienia małe litery na wielkie litery

Ciągi znaków / Długość ciągu

Właściwość `length` przechowuje aktualną długość danego ciągu znaków.

```
var code = “0123456789”;

// Zmienna characters przyjmie wartość 10
var characters = code.length;

// Zmienna characters przyjmie wartość 4
characters = code.substring(0, 4).length;
```

Ciągi znaków / Przykłady

```
var player1 = “Bill”;

var player2 = ‘Doug’;

var paragraph = “Lorem ipsum\ndolor sit amet”;

var newLine = “The \\n character represents a new line”;

var quote = ‘She said “goodbye” too many times’;
```

Ćwiczenie 4 / Wielokropek

Napisz funkcję `shortQuote`, która przymuje dowolny ciąg znaków, skraca go o połowę, dodaje wielokropek, a następnie otacza cudzysłowami i wypisuje go do konsoli:

```
shortQuote("At vero eos et accusamus") → ""At vero eos ..."
shortQuote("0.123456789") → ""0.123..."
shortQuote("") → ""..."
```

Wyrażenia regularne

Wyrażenia regularne to wzorce, które pozwalają opisywać łańcuchy znaków. Dzięki nim można zweryfikować siłę hasła, poprawność adresu e-mail, albo wyszukać konkretne informacje w długich tekstach.

Wyrażenia regularne / Znaki specjalne

`.`	wyszukaj dowolny znak
`\s`	wyszukaj spację
`\S`	wyszukaj nie-spację
`\d`	wyszukaj cyfrę
`\D`	wyszukaj nie-cyfrę
`N?`	wyszukaj zero lub jedno wystąpienie *N*
`N+`	wyszukaj jedno lub więcej wystąpień *N*
`N*`	wyszukaj zero lub więcej wystąpień *N*

Wyrażenia regularne / Znaki specjalne

`{X}`	wyszukaj dokładnie *X* wystąpień
`{X,Y}`	wyszukaj od *X* do *Y* wystąpień
`^abc`	wyszukaj wystąpienie zaczynające się od *abc*
`abc$`	wyszukaj wystąpienie kończące się na *abc*
`(X|Y)`	wyszukaj wystąpienie *X* lub *Y*
`[abc]`	wyszukaj wymienione znaki
`[^abc]`	wyszukaj niewymienione znaki
`[A-z]`	wyszukaj wymieniony przedział znaków

Wyrażenia regularne / Metody

`exec`	przeszukuje ciąg znaków wyrażeniem regularnym
`test`	sprawdza ciąg znaków wyrażeniem regularnym

Wyrażenia regularne / Przykłady

```
// Opisuje ciągi znaków typu “Apollo 8”, “Apollo 13”, itp.
var apollo = /Apollo\s\d+/;

// Zwraca true
apollo.test(“Apollo 13”);

// Zwraca false
apollo.test(“apollo T50”);
```

Wyrażenia regularne / Przykłady

```
// Opisuje ciągi znaków typu “09:30 AM”, “11:00 PM”, itp.
var time = /\d{2}:\d{2}\s(AM|PM)/;

// Zwraca true
time.test(“11:00 PM”);

// Zwraca false
time.test(“7:30 AM”);
```

Ćwiczenie 5 / Liczba trafień

Napisz funkcję `countHits`, która przymuje ciąg znaków obrazujący poszukiwaną frazę oraz ciąg znaków, który będzie przeszukiwany, a następnie zwraca liczbę trafień:

```
countHits("cobra", "The cat is the problem") → 0
countHits("cat", "The cat is the problem") → 1
countHits("the", "The cat is the problem") → 2
```

Wyrażenia warunkowe

Wyrażenia wrunkowe pozwalają wpływać na porządek wykonywania się programu. Umożliwiają wykonywanie lub pomijanie wybranych operacji w zależności od określonych przez programistę warunków.

Wyrażenia warunkowe / Konstrukcja if

```
if (warunek1) {
    // Wykonaj, gdy warunek1 jest prawdziwy
} else if (warunek2) {
    // Wykonaj, gdy warunek2 jest prawdziwy
} else if (warunek3) {
    // Wykonaj, gdy warunek3 jest prawdziwy
} else {
    // Wykonaj, gdy żaden z powyższych warunków nie jest prawdziwy
}
```

Wyrażenia warunkowe / Konstrukcja switch

```
switch (wartość) {
    case “foo”:
        // Wykonaj, gdy wartość jest równa “foo”
        break;
    case 25000:
        // Wykonaj, gdy wartość jest równa 25000
        break;
    default:
        // Wykonaj, gdy żaden z powyższych przypadków nie wystąpił
}
```

Wyrażenia warunkowe / Operatory logiczne

Do tworzenia bardziej złożonych warunków wykorzystuje się operatory logiczne.

```
&&      koniunkcja (AND)

||      alternatywa (OR)

!       negacja (NOT)
```

Wyrażenia warunkowe / Przykłady

```
if (day === “sobota” || day === “niedziela”) {
    console.log(“ZAMKNIĘTE”);
} else {
    console.log(“OTWARTE”);
}

if (!active && userReputation < 200) {
    console.log(“UŻYTKOWNIK NIEAKTYWNY”);
}
```

Ćwiczenie 6 / Sprawdzanie wieku

Napisz funkcję `isAdult`, która przyjmuje wiek, a następnie zwraca true, gdy wiek wynosi 18 lub więcej oraz false, gdy tak nie jest. Jeśli wartość przekazana do funkcji nie jest liczbą, należy zwrócić false i wypisać do konsoli "Oops!”:

```
isAdult(42) → true
isAdult(15) → false
isAdult(“baz”) → false
```

Tablice

Tablica to jedna z powszechnie używanych struktur danych w JavaScript.

Pojedyncza tablica stanowi zbiór indeksów oraz wartości do nich przypisanych.

W JavaScript tablice zapisuje się przy pomocy nawiasów kwadratowych.

Tablice / Metody

`concat`	łączy dwie lub więcej tablic
`includes`	sprawdza obecność wybranego elementu w tablicy
`indexOf`	zwraca pierwszy indeks poszukiwanego elementu
`lastIndexOf`	zwraca ostatni indeks poszukiwanego elementu
`forEach`	iteruje po tablicy
`filter`	filtruje elementy tablicy
`map`	mapuje elementy tablicy
`reduce`	redukuje elementy tablicy

Tablice / Metody

`join`	łączy elementy tablicy w ciąg znaków
`pop`	usuwa element z końca tablicy
`push`	dodaje element do końca tablicy
`shift`	usuwa element z początku tablicy
`unshift`	dodaje element do początku tablicy
`sort`	sortuje elementy tablicy
`reverse`	odwraca kolejność elementów tablicy
`slice`	zwraca fragment tablicy
`splice`	wstawia elementy pod wybrany indeks tablicy

Tablice / Długość tablicy

Właściwość `length` przechowuje aktualną długość danej tablicy.

```
var numbers = [0, 1, 2, 3, 4, 5, 6];

// Zmienna size przyjmie wartość 7
var size = numbers.length;

// Zmienna size przyjmie wartość 4
size = numbers.slice(0, 4).length;
```

Ćwiczenie 7 / Pomijanie elementów

Napisz funkcję `skip`, która przyjmuje liczbę `n` oraz tablicę, a następnie zwraca przekazaną tablicę, pomijając jej `n` pierwszych elementów:

```
skip(0, ["foo", "bar"]) → ["foo", "bar"]
skip(2, [10, 20, 30, 40]) → [30, 40]
```

Pętle

Pętla to konstrukcja pozwalająca na wielokrotne wykonywanie pewnego fragmentu kodu do czasu jej przerwania. Najczęściej, obieg pętli zatrzymuje się przy pomocy wyrażeń warunkowych.

Pętle / Konstrukcja for

```
for (inicjalizacja; warunek; działanie) {
    // Wykonaj, jeśli warunek jest prawdziwy
}
```

Pętle / Przykłady

```
var array = [“foo”, “bar”, “baz”];

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

for (var i in array) {
    console.log(array[i]);
}
```

Pętle / Konstrukcja while

```
while (warunek) {
    // Wykonaj, jeśli warunek jest prawdziwy
}
```

Pętle / Przykłady

```
var array = [“foo”, “bar”, “baz”];
var i = 0;

while (i < array.length) {
    console.log(array[i]);
    ++i;
}
```

Ćwiczenie 8 / Obliczanie sumy

Napisz funkcję `addAll`, która przyjmuje tablicę liczb, a następnie zwraca ich sumę:

```
addAll([]) → 0
addAll([2, 5, 10]) → 17
```

Obiekty

Obiekt to jedna z powszechnie używanych struktur danych w JavaScript.

Pojedynczy obiekt stanowi zbiór kluczy oraz wartości do nich przypisanych.

W JavaScript obiekty zapisuje się przy pomocy nawiasów klamrowych.

Obiekty / Przykłady

```
var book = {
    author: “Orson Scott Card”,
    title: “Ender’s Game”,
    pages: 400,
    isNew: false,
    tags: [“computer”, “fiction”, “war”],
    toString() {
        return “Book: Ender’s Game”;
    }
};
```

Obiekty / Dostęp do wartości

```
// Wypisuje “Orson Scott Card” do konsoli
console.log(book.author);

// Wypisuje “Ender’s Game” do konsoli
console.log(book[“title”]);

var property = “pages”;

// Wypisuje 400 do konsoli
console.log(book[property]);
```

Obiekty / Pętla for

```
var object = { name: “John”, age: 20 };

for (var key in object) {
    console.log(key);
    console.log(object[key]);
}
```

Obiekty / Date

`getTime`/`setTime`	pobiera/ustawia milisekundy od 01.01.1970
`getSeconds`/`setSeconds`	pobiera/ustawia sekundy (0-59)
`getMinutes`/`setMinutes`	pobiera/ustawia minuty (0-59)
`getHours`/`setHours`	pobiera/ustawia godziny (0-23)
`getDate`/`setDate`	pobiera/ustawia dzień (1-31)
`getMonth`/`setMonth`	pobiera/ustawia miesiąc (0-11)
`getFullYear`/`setFullYear`	pobiera/ustawia rok (yyyy)

Obiekty / Przykłady

```
var today = new Date();

var yesterday = new Date();

// Zmienna yesterday reprezentuje poprzedni dzień
yesterday.setDate(today.getDate() - 1);

// Zwraca wartość zmiennej yesterday jako ciąg znaków
yesterday.toISOString();
```

Obiekty / Math

`Math.E`	zwraca stałą E
`Math.PI`	zwraca stałą PI
`Math.abs`	zwraca wartość absolutną wybranej liczby
`Math.min`	zwraca najmniejszą z wybranych liczb
`Math.max`	zwraca największą z wybranych liczb
`Math.random`	zwraca losową liczbę z przedziału 0-1

Obiekty / Math

`Math.floor`	zwraca podłogę wybranej liczby
`Math.ceil`	zwraca sufit wybranej liczby
`Math.round`	zwraca zaokrąglenie wybranej liczby
`Math.pow`	zwraca potęgę wybranej liczby
`Math.sqrt`	zwraca pierwiastek kwadratowy wybranej liczby
`Math.sin`	zwraca sinus wybranej liczby
`Math.cos`	zwraca cosinus wybranej liczby

Obiekty / Przykłady

```
var value = -12;

// Zmienna radius przyjmie wartość 12
var radius = Math.abs(value);

var circleArea = Math.PI * Math.pow(radius, 2);

// Zmienna circleArea przyjmie wartość 452.39
var fixedCircleArea = circleArea.toFixed(2);
```

Ćwiczenie 9 / Klucz i wartość

Napisz funkcję `toPairs`, która przyjmuje obiekt i zwraca tablicę tablic, które przechowują pary kluczy i wartości tego obiektu:

```
toPairs({ foo: 10, bar: 20 }) → [ ["foo", 10], ["bar", 20] ]
```

This

This to słowo kluczowe, a jego wartość zależy od kontekstu w którym zostało wywołane.

Istnieje wiele przypadków użycia this, przykładowo w funkcjach-konstruktorach.

This / Przykłady

```
var person = {
    name: “John”,
    greet: function greet() {
        console.log(“Hello, I’m ” + this.name);
    }
};

// Wypisuje “Hello, I’m John” do konsoli
person.greet();
```

This / Przykłady

```
var person = {
    name: “John”,
    greet: function greet() {
        console.log(“Hello, I’m ” + this.name);
    }
};

// Wypisuje “Hello, I’m John” do konsoli
person.greet();
```

Obiekt *person* jest kontekstem dla *this*

This / Wymuszanie kontekstu

```
var person = { . . . };
var monster = {
    name: “Zorg”
};

// Wypisuje “Hello, I’m Zorg” do konsoli
person.greet.call(monster);
person.greet.apply(monster);
```

This / Wiązanie kontekstu

```
var person = { . . . };
var monster = { . . . };

// Tworzy nową funkcję z “przywiązanym” this
var monsterGreeting = person.greet.bind(monster);

// Wypisuje “Hello, I’m Zorg” do konsoli
monsterGreeting();
```

This / Funkcje-konstruktory

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

// Tworzy obiekt Person { name: “Adam”, age: 15 }
var adam = new Person(“Adam”, 15);
```

This / Funkcje-konstruktory

```
// Dodaje metodę do wszystkich obiektów Person
Person.prototype.isAdult = function () {
    return this.age >= 18;
}

// Zwraca false
adam.isAdult();
```

Ćwiczenie 10 / Poprawny wynik

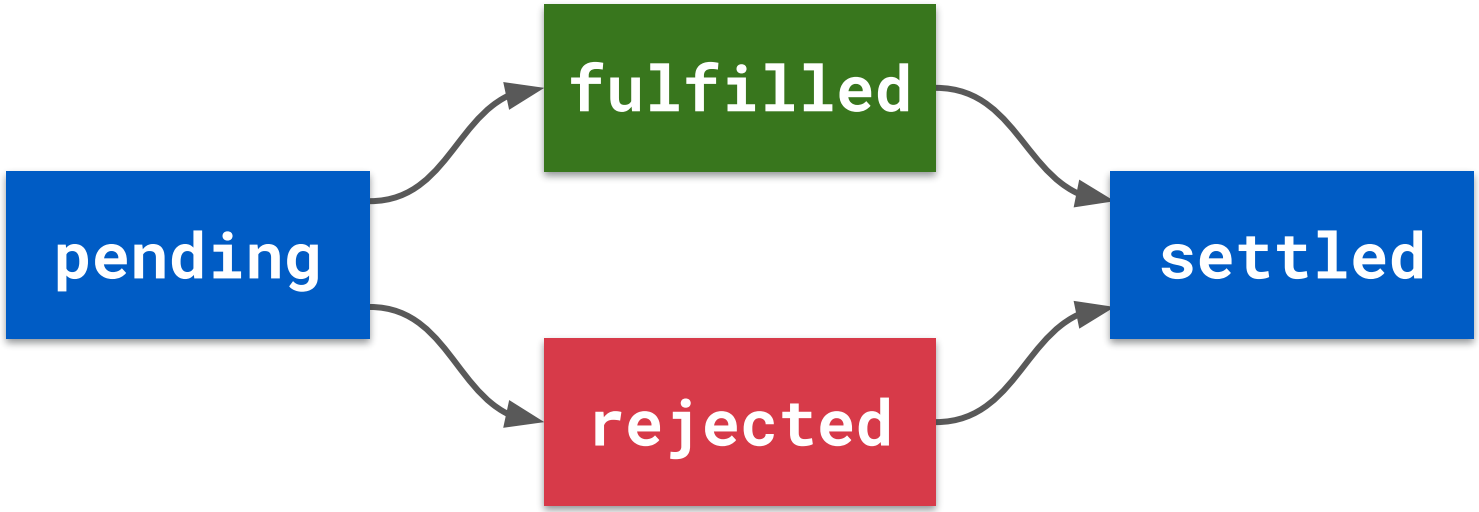
Popraw poniższy kod tak, aby zmienna `total` przyjęła wartość 25; przepisanie funkcji `num` lub podstawienie odpowiednich wartości do zmiennej `total` jest niedozwolone:

```
function num() {
    return this;
}

var total = 20 + num();
```

Promise

Promise to obiekt reprezentujący pewne działanie, które może zakończyć się sukcesem, zwracając pożądaną wartość, lub niepowodzeniem, zwracając komunikat błędu.

Promise / Cykl życia



Promise / Konstrukcja Promise

```
function executor(resolve, reject) {
    // Wywołuje funkcję resolve w przypadku sukcesu
    // Wywołuje funkcję reject w przypadku niepowodzenia
}

var promise = new Promise(executor);

// Konsumuje sukces lub niepowodzenie
promise.then(success).catch(error);
```

Promise / Przykłady

```
function burger(resolve, reject) {
    resolve(“BigMac”);
}

var ticket = new Promise(burger);

// Wypisuje “BigMac” do konsoli
ticket.then(function (result) {
    console.log(result);
});
```

Ćwiczenie 11 / Chaining

Napisz dwie funkcje wykorzystujące `Promise` tak, aby można było wykorzystać chaining. Funkcja `upperCaseAll` przyjmuje tablicę wyrazów i zamienia wszystkie ich litery na wielkie. Funkcja `alphabetize` przyjmuje tablicę wyrazów i sortuje je alfabetycznie:

```
var words = [“c”, “a”, “b”];

upperCaseAll(words)
    .then(alphabetize)
    .then(console.log); → [“A”, “B”, “C”]
```

Fetch

Fetch to funkcja dostępna w przeglądarkach, która pozwala wykonywać zapytania HTTP.

W przeciwieństwie do starszej alternatywy, XMLHttpRequest, fetch zwraca Promise.

Fetch / JSON

JavaScript Object Notation to format wymiany danych autorstwa Douglasa Crockforda, ustandaryzowany przez ECMA International w 2013 roku.

```
{
    “title”: “Ender’s Game”,
    “pages”: 400,
    “isNew”: false,
    “tags”: [“computer”, “fiction”, “war”],
    “reviews”: { “likes”: 123 }
}
```

Fetch / Obiekt JSON

Obiekt JSON udostępnia dwie metody niezbędne do zamiany ciągu znaków na obiekt lub zamiany obiektu na ciąg znaków reprezentujący ‘poprawny’ JSON.

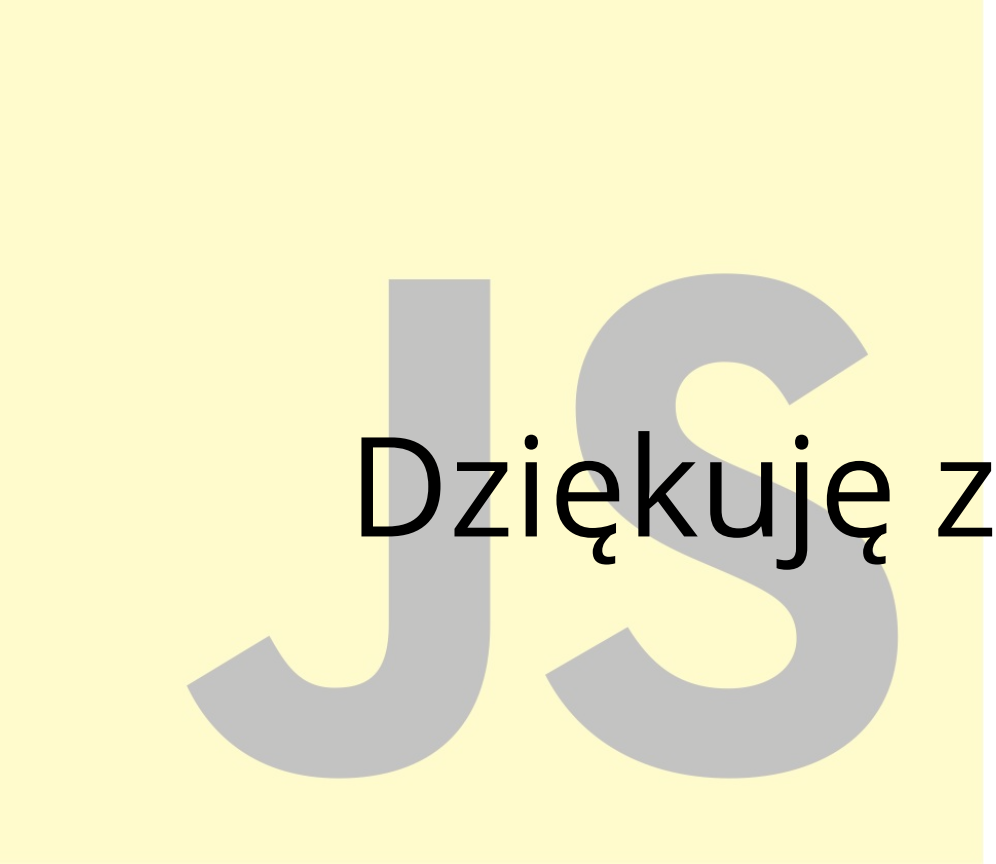
```
// Zwraca ciąg znaków “{“name”:“John”,“age”:21,“active”:true}”
JSON.stringify({ name: “John”, age: 21, active: true });

// Zwraca obiekt { name: “John”, age: 21, active: true }
JSON.parse(‘{“name”:“John”,“age”:21,“active”:true}’);
```

Fetch / Przykłady

```
fetch("https://jsonplaceholder.typicode.com/users")
    .then(function (response) {
        return response.json();
    })
    .then(function (users) {
        console.log(users);
    })
    .catch(function (error) {
        console.log(error);
    });
```

Ćwiczenie 12 / Integracja z API

Napisz funkcję `getPosts`, która przy pomocy `fetch` wykonuje zapytanie HTTP pod adres `https://jsonplaceholder.typicode.com/posts`, a następnie wypisuje wartość pola `title` pierwszych dwudziestu postów do konsoli.



Dziękuję za uwagę.