**Zad 1**

* **Silnik JavaScript:** program interpretujący i wykonujący kod JavaScript
* **Kontekst wykonania:** środowisko, w którym wykonywany jest kod JavaScript
* **Środowisko JavaScript:** szersze pojęcie obejmujące silnik JavaScript i kontekst wykonania
* **API:** zestaw funkcji i obiektów dostępnych dla programisty
* **Obiekt globalny:** obiekt dostępny we wszystkich kontekstach wykonania
* **Interpretacja JavaScript:** program interpretujący kod JavaScript i tłumaczący go na język maszynowy
* **Kompilator JavaScript:** program kompilujący kod JavaScript do kodu maszynowego
* **Wtyczka JavaScript:** program rozszerzający funkcjonalność środowiska JavaScript

**Zad 2**

**Obiekt navigator w JS**

Obiekt navigator udostępnia informacje o środowisku użytkownika: przeglądarce, systemie i sieci.

**Właściwości:**

* appName: Nazwa przeglądarki (np. "Chrome").
* appVersion: Wersja przeglądarki (np. "87.0.4280.141").
* userAgent: Pełny identyfikator przeglądarki i systemu (np. "Mozilla/5.0 ...").
* platform: Nazwa platformy systemowej (np. "Windows").
* language: Język przeglądarki (np. "pl-PL").
* onLine: Czy użytkownik jest online (true/false).
* geolocation: Dostęp do API Geolocation (współrzędne geograficzne).

**Przykłady:**

* Pobranie nazwy przeglądarki:

const browserName = navigator.appName;

console.log(browserName); // "Chrome"

* Sprawdzenie, czy użytkownik jest online:

const isOnline = navigator.onLine;

console.log(isOnline); // true

* Pobranie współrzędnych geograficznych:

navigator.geolocation.getCurrentPosition((position) => {

const latitude = position.coords.latitude;

const longitude = position.coords.longitude;

console.log(latitude, longitude); // 52.2370014 21.0175322

});

**Zad 3**

**Obiekt history w JS:**

**Właściwości:**

* length: Liczba stron w historii.
* state: Obiekt JSON przechowywany w historii.

**Metody:**

* back(): Przejście do poprzedniej strony.
* forward(): Przejście do następnej strony.
* go(delta): Przejście do strony o delta pozycji od bieżącej.
* pushState(state, title, url): Dodanie nowej strony.
* replaceState(state, title, url): Zastąpienie bieżącej strony.

**Przykłady:**

* Nawigacja wstecz: history.back()
* Przejście o 2 pozycje dalej: history.go(2)
* Dodanie nowej strony:

const state = {page: "about"};

const title = "About Page";

const url = "/about";

history.pushState(state, title, url);

* Zastąpienie bieżącej strony:

const state = {page: "contact"};

const title = "Contact Page";

const url = "/contact";

history.replaceState(state, title, url);

**Zad 4**

**Obiekt location w JS:**

**Właściwości:**

* href: Pełny adres URL strony.
* protocol: Protokół strony (np. "https").
* host: Nazwa hosta (np. "[www.example.com](https://www.example.com/)").
* hostname: Nazwa domeny (np. "example.com").
* port: Numer portu (np. "80").
* pathname: Ścieżka do pliku (np. "/index.html").
* search: Ciąg zapytania (np. "?q=javascript").
* hash: Fragment adresu URL (np. "#about").

**Metody:**

* reload(): Odświeżenie strony.
* replace(url): Zmiana adresu URL strony.
* assign(url): Zmiana adresu URL strony i przejście do niej.

**Przykłady:**

* Pobranie pełnego adresu URL strony:

const url = location.href;

console.log(url); // https://www.example.com/index.html

* Przeładowanie strony:

location.reload();

* Przejście do strony o adresie https://www.google.com:

location.assign("https://www.google.com");

**Zad 5**

**Zdarzenia dokumentu HTML**

Zdarzenia dokumentu HTML to akcje wykonywane przez użytkownika na stronie internetowej, które wywołują określone skrypty. Oto 3 przykłady:

**1. Zdarzenie click:**

* Opis: Występuje po kliknięciu elementu HTML.
* Przykład:

<button onclick="alert('Kliknięto przycisk!')">Kliknij mnie</button>

**2. Zdarzenie change:**

* Opis: Występuje po zmianie wartości elementu formularza.
* Przykład:

<input type="text" onchange="alert('Wartość została zmieniona!')">

**3. Zdarzenie keydown:**

* Opis: Występuje po naciśnięciu klawisza na klawiaturze.
* Przykład:

<body onkeydown="alert('Naciśnięto klawisz: ' + event.key)">

</body>

**Zad 6**

W języku JavaScript dziedziczenie klas odbywa się za pomocą prototypów. Każdy obiekt w JavaScript ma prototyp, który może być innym obiektem lub null. Kiedy próbujesz uzyskać dostęp do właściwości obiektu, które nie istnieją w tym obiekcie, silnik JavaScript będzie szukał tych właściwości w jego prototypie, a następnie w prototypie prototypu, i tak dalej, aż znajdzie właściwość lub dotrze do prototypu null.

Klasy w JavaScript mogą być tworzone za pomocą składni class. Dziedziczenie w klasach jest osiągane poprzez używanie słowa kluczowego extends, co pozwala na tworzenie hierarchii klas.

Przykład:

class Zwierze {

constructor(imie) {

this.imie = imie;

}

dajGlos() {

console.log(`${this.imie} wydaje dźwięk.`);

}

}

class Pies extends Zwierze {

constructor(imie, rasa) {

super(imie);

this.rasa = rasa;

}

szczekaj() {

console.log(`${this.imie} rasy ${this.rasa} szczeka.`);

}

}

const pies = new Pies('Burek', 'Owczarek niemiecki');

pies.dajGlos(); // Wyświetli: Burek wydaje dźwięk.

pies.szczekaj(); // Wyświetli: Burek rasy Owczarek niemiecki szczeka.

W tym przykładzie klasa `Pies` dziedziczy z klasy `Zwierze`, używając słowa kluczowego `extends`. Dzięki temu klasa `Pies` ma dostęp do wszystkich metod i właściwości klasy `Zwierze` za pomocą metody `super()`.

**Zad 6**

Dziedziczenie prototypowe w JavaScript polega na tym, że każdy obiekt ma referencję do innego obiektu, zwanej prototypem. Gdy właściwość nie zostanie znaleziona w obiekcie, JavaScript automatycznie szuka jej w prototypie. Jeśli tam również nie zostanie znaleziona, szuka dalej, aż dotrze do prototypu null.

Przykład 1:  
  
javascript  
Copy code  
// Definiujemy prototyp Zwierze  
var ***Zwierze*** = {  
 dajGlos: function() {  
 ***console***.log(this.imie + ' wydaje dźwięk.');  
 }  
};  
  
// Tworzymy obiekt Pies z prototypem Zwierze  
var ***Pies*** = ***Object***.create(***Zwierze***);  
***Pies***.szczekaj = function() {  
 ***console***.log(this.imie + ' szczeka.');  
};  
  
var ***burek*** = ***Object***.create(***Pies***);  
***burek***.imie = 'Burek';  
  
***burek***.dajGlos(); // Wyświetli: Burek wydaje dźwięk.  
***burek***.szczekaj(); // Wyświetli: Burek szczeka.  
Przykład 2:  
  
javascript  
Copy code  
function Zwierze(imie) {  
 this.imie = imie;  
}  
  
Zwierze.prototype.dajGlos = function() {  
 ***console***.log(this.imie + ' wydaje dźwięk.');  
};  
  
function Pies(imie, rasa) {  
 Zwierze.call(this, imie);  
 this.rasa = rasa;  
}  
  
Pies.prototype = ***Object***.create(Zwierze.prototype);  
Pies.prototype.constructor = Pies;  
  
Pies.prototype.szczekaj = function() {  
 ***console***.log(this.imie + ' rasy ' + this.rasa + ' szczeka.');  
};  
  
var ***burek*** = new Pies('Burek', 'Owczarek niemiecki');  
  
***burek***.dajGlos(); // Wyświetli: Burek wydaje dźwięk.  
***burek***.szczekaj(); // Wyświetli: Burek rasy Owczarek niemiecki szczeka.  
Przykład 3:  
  
javascript  
Copy code  
class Zwierze {  
 constructor(imie) {  
 this.imie = imie;  
 }  
  
 dajGlos() {  
 ***console***.log(`${this.imie} wydaje dźwięk.`);  
 }  
}  
  
class Pies extends Zwierze {  
 constructor(imie, rasa) {  
 super(imie);  
 this.rasa = rasa;  
 }  
  
 szczekaj() {  
 ***console***.log(`${this.imie} rasy ${this.rasa} szczeka.`);  
 }  
}  
  
const ***burek*** = new Pies('Burek', 'Owczarek niemiecki');  
  
***burek***.dajGlos(); // Wyświetli: Burek wydaje dźwięk.  
***burek***.szczekaj(); // Wyświetli: Burek rasy Owczarek niemiecki szczeka.

| **Aspekt** | **Dziedziczenie prototypowe** | **Klasy ES6** |
| --- | --- | --- |
| Składnia | Wykorzystuje obiekty i **Object.create()** | Wykorzystuje składnię **class** i dziedziczenie za pomocą **extends** |
| Definicja konstruktora | Funkcja konstruktora i prototypy | Konstruktor klasy i metody klasy |
| Dziedziczenie | Poprzez prototypy | Poprzez klauzulę **extends** |
| Definiowanie metod | Prototypy obiektów lub metody zdefiniowane w konstruktorze | Metody klasy zdefiniowane w ciele klasy |
| Dostęp do super | Brak prostego sposobu, często wymaga jawnej manipulacji prototypem | Używając **super** w konstruktorze i metodach klasy |
| Popularność | Tradycyjne podejście, zwykle stosowane w starszym kodzie JavaScript | Nowoczesne podejście, preferowane w nowych aplikacjach JavaScript |

Dziedziczenie prototypowe było pierwotnym sposobem dziedziczenia w JavaScript, a klasy ES6 zostały wprowadzone, aby uprościć tworzenie hierarchii dziedziczenia i uczynić kod bardziej czytelnym i łatwiejszym do zrozumienia.