Zadanie 1

1. Liczby całkowite (integer):

let liczbaCal = 42;

console.log(liczbaCal); // Wyświetli 42

2. Liczby zmiennoprzecinkowe (floating-point):

let liczbaZmienn = 3.14;

console.log(liczbaZmienn); // Wyświetli 3.14

3. Liczby z notacją naukową:

let liczbaNaukowa = 2.5e3; // 2.5 \* 10^3

console.log(liczbaNaukowa); // Wyświetli 2500

4. Ciągi znaków (string):

let tekst = "To jest przykład tekstu.";

console.log(tekst); // Wyświetli "To jest przykład tekstu."

5. Prawda/fałsz (boolean):

let prawdaFałsz = true;

console.log(prawdaFałsz); // Wyświetli true

6. Wartość null:

let pustaWartość = null;

console.log(pustaWartość); // Wyświetli null

7. Wartość undefined:

let niezdefiniowanaWartość;

console.log(niezdefiniowanaWartość); // Wyświetli undefined

8. Symbol (ES6):

const symbol = Symbol('opisSymbolu');

console.log(symbol); // Wyświetli unikalny symbol

9. BigInt (ES11):

const bigInt = 1234567890123456789012345678901234567890n;

console.log(bigInt); // Wyświetli dużą liczbę BigInt

Zadanie 2

1. Tablice (Arrays): Tablice to uporządkowane kolekcje elementów. Mogą zawierać różne typy danych.

let mojaTablica = [1, 2, 3, "cztery", "pięć"];

2. Obiekty (Objects): Obiekty pozwalają na przechowywanie właściwości i metod w jednej zmiennej.

let osoba = {

    imie: "Jan",

    nazwisko: "Kowalski",

    wiek: 30

  };

3. Funkcje (Functions): Funkcje w mogą być przechowywane jako zmienne i przekazywane jako argumenty do innych funkcji.

function dodaj(a, b) {

    return a + b;

  }

4. Zagnieżdżone tablice i obiekty: Możesz tworzyć złożone struktury danych, które zawierają inne tablice, obiekty lub funkcje.

let zagniezdzonaTablica = [[1, 2], [3, 4], [5, 6]];

let zagniezdzonyObiekt = {

  daneOsoby: {

    imie: "Anna",

    nazwisko: "Nowak"

  },

  wiek: 25

};

5. Funkcje zwrotne (Callbacks): Funkcje mogą być przekazywane jako argumenty do innych funkcji i wywoływane w określonym momencie.

function wykonajCos(callback) {

    console.log("Wykonywanie czegoś...");

    callback();

  }

  function przykladowaFunkcja() {

    console.log("To jest funkcja zwrotna.");

  }

  wykonajCos(przykladowaFunkcja);

6. Obiekty JSON ( Object Notation): JSON to format danych, który jest szeroko używany w komunikacji między serwerem a przeglądarką.

let jsonDane = '{"imie": "Maria", "nazwisko": "Wiśniewska"}';

let obiektDane = JSON.parse(jsonDane);

7. Mapy: Mapy są strukturami danych, które pozwalają na przechowywanie kluczy i wartości.

let mapa = new Map();

mapa.set("klucz1", "wartosc1");

mapa.set("klucz2", "wartosc2");

8. Zbiory: Zbiory pozwalają na przechowywanie unikalnych wartości.

let zbior = new Set();

zbior.add(1);

zbior.add(2);

zbior.add(1); // Ten element zostanie zignorowany, ponieważ zbiór przechowuje tylko unikalne wartości

Zadanie 3

1. Operator przypisania: Służy do przypisywania wartości do zmiennych.

let x = 5; // Operator przypisania "=" przypisuje wartość 5 do zmiennej x.

2. Operatory arytmetyczne: Służą do wykonywania operacji matematycznych.

let a = 10;

let b = 5;

let suma = a + b; // Operator dodawania "+"

let roznica = a - b; // Operator odejmowania "-"

let iloczyn = a \* b; // Operator mnożenia "\*"

let iloraz = a / b; // Operator dzielenia "/"

let modulo = a % b; // Operator modulo "%"

3. Operatory porównania: Służą do porównywania wartości.

let x = 5;

let y = 10;

let czyRowne = x === y; // Operator równości "===" (wartość i typ)

let czyNieRowne = x !== y; // Operator nierówności "!==" (wartość i typ)

let czyWieksze = x > y; // Operator większości ">"

let czyMniejsze = x < y; // Operator mniejszości "<"

let czyWiekszeLubRowne = x >= y; // Operator większości lub równości ">="

let czyMniejszeLubRowne = x <= y; // Operator mniejszości lub równości "<="

4. Operatory logiczne: Służą do wykonywania operacji logicznych.

let prawda = true;

let falsz = false;

let wynikAND = prawda && falsz; // Operator logiczny "&&" (I)

let wynikOR = prawda || falsz; // Operator logiczny "||" (LUB)

let wynikNOT = !prawda; // Operator logiczny "!" (NEGACJA)

5. Operator trójargumentowy (warunek ? wartość1 : wartość2): Pozwala na skrócenie zapisu instrukcji warunkowej if-else.

let wiek = 18;

let pelnoletni = wiek >= 18 ? "Tak" : "Nie";

6. Operatory inkrementacji i dekrementacji: Służą do zwiększania lub zmniejszania wartości zmiennych o 1.

let x = 5;

x++; // Operator inkrementacji (x staje się 6)

x--; // Operator dekrementacji (x staje się 5)

7. Operatory konkatenacji: Służą do łączenia ciągów znaków.

let imie = "John";

let nazwisko = "Doe";

let pelneImie = imie + " " + nazwisko; // Operator konkatenacji "+"

8. Operatory bitowe: Służą do wykonywania operacji bitowych na liczbach.

let a = 5; // Binarnie: 0101

let b = 3; // Binarnie: 0011

let wynikAND = a & b; // Operator bitowy AND "&" (wynik: 0001)

let wynikOR = a | b; // Operator bitowy OR "|" (wynik: 0111)

Zadanie 4

Oto operatory dotyczące ciągów tekstowych (operatory konkatenacji) i porównania w :

Operatory ciągów tekstowych (konkatenacji):

1. `+`: Operator konkatenacji, służy do łączenia dwóch ciągów tekstowych.

let imie = "John";

let nazwisko = "Doe";

let pelneImie = imie + " " + nazwisko; // Operator konkatenacji "+"

Operatory porównania:

2. `===`: Operator równości, porównuje wartości i typy dwóch operandów.

let x = 5;

let y = "5";

let czyRowne = x === y; // Operator równości "==="

3. `!==`: Operator nierówności, sprawdza, czy dwie wartości są różne pod względem wartości lub typu.

let x = 5;

let y = "5";

let czyNieRowne = x !== y; // Operator nierówności "!=="

4. `==`: Operator równości luźnej, porównuje wartości dwóch operandów bez uwzględniania typów.

let x = 5;

let y = "5";

let czyRowneLuźnie = x == y; // Operator równości luźnej "=="

5. `!=`: Operator nierówności luźnej, sprawdza, czy dwie wartości są różne bez uwzględniania typów

let x = 5;

let y = "5";

let czyNieRowneLuźnie = x != y; // Operator nierówności luźnej "!="

6. `>`: Operator większości, porównuje, czy pierwszy operand jest większy niż drugi.

let a = 10;

let b = 5;

let czyMniejsze = a > b; // Operator mniejszości "<"

7. `<`: Operator mniejszości, porównuje, czy pierwszy operand jest mniejszy niż drugi.

let a = 10;

let b = 5;

let czyMniejsze = a < b; // Operator mniejszości "<"

8. `>=`: Operator większości lub równości, porównuje, czy pierwszy operand jest większy lub równy drugiemu.

let a = 10;

let b = 10;

let czyWiekszeLubRowne = a >= b; // Operator większości lub równości ">="

9. `<=`: Operator mniejszości lub równości, porównuje, czy pierwszy operand jest mniejszy lub równy drugiemu.

let a = 5;

let b = 10;

let czyMniejszeLubRowne = a <= b; // Operator mniejszości lub równości "<="

Zadanie 5

Oto przykłady różnych operatorów przypisania w języku :

1. Operator przypisania (`=`): Służy do przypisywania wartości do zmiennej.

let x = 5; // Przypisanie wartości 5 do zmiennej x

2. Operator przypisania z dodawaniem (`+=`): Dodaje wartość do zmiennej i przypisuje wynik do tej samej zmiennej.

let x = 10;

x += 5; // Dodaje 5 do x, wynik 15 zostaje przypisany do x

3. Operator przypisania z odejmowaniem (`-=`): Odejmuje wartość od zmiennej i przypisuje wynik do tej samej zmiennej.

let x = 10;

x -= 3; // Odejmuje 3 od x, wynik 7 zostaje przypisany do x

4. Operator przypisania z mnożeniem (`\*=`): Mnoży zmienną przez wartość i przypisuje wynik do tej samej zmiennej.

let x = 5;

x \*= 4; // Pomnóż x przez 4, wynik 20 zostaje przypisany do x

5. Operator przypisania z dzieleniem (`/=`): Dzieli zmienną przez wartość i przypisuje wynik do tej samej zmiennej.

let x = 20;

x /= 4; // Podziel x przez 4, wynik 5 zostaje przypisany do x

6. Operator przypisania modulo (`%=`): Oblicza resztę z dzielenia zmiennej przez wartość i przypisuje wynik do tej samej zmiennej.

let x = 17;

x %= 5; // Oblicza resztę z dzielenia x przez 5, wynik 2 zostaje przypisany do x

7. Operator przypisania konkatenacji (`+=`): Służy do łączenia ciągów znaków i przypisywania wyniku do zmiennej.

let tekst = "Hello";

tekst += " World"; // Łączy "Hello" i " World" i przypisuje wynik do zmiennej tekst

8. Operator przypisania bitowego (`&=`, `|=`): Wykonuje operacje bitowe i przypisuje wynik do zmiennej.

let a = 5; // Binarnie: 0101

a &= 3;    // Binarnie: 0011 (operator bitowy AND), wynik: 1

let b = 3; // Binarnie: 0011

b |= 4;    // Binarnie: 0100 (operator bitowy OR), wynik: 4

Zadanie 6

Programowanie obiektowe (PO) w języku JavaScript (JS) opiera się na kilku kluczowych cechach:

1. Obiekty i Klasy: W JS można tworzyć obiekty i klasy. Klasy są szablonami, na podstawie których można tworzyć obiekty. Obiekty są instancjami klas lub strukturami zawierającymi dane i metody.
2. Enkapsulacja: Enkapsulacja oznacza ukrywanie pewnych informacji i operacji wewnątrz obiektu. W JS można wykorzystać zamknięcia (closures) i techniki takie jak wywoływanie zwrotne (callbacks) do enkapsulacji danych i zachowań.
3. Dziedziczenie: Dziedziczenie pozwala na tworzenie hierarchii klas, w której nowe klasy mogą dziedziczyć właściwości i metody po innych klasach. W JS dziedziczenie jest oparte na prototypach, co oznacza, że obiekty mogą dziedziczyć od innych obiektów.
4. Polimorfizm: Polimorfizm pozwala na tworzenie wielu obiektów o różnych typach, które mogą wykonywać te same operacje w sposób odpowiedni dla swojego typu. W JS polimorfizm jest często osiągany poprzez przeciążanie (overloading) i przesłanianie (overriding) metod.
5. Abstrakcja: Abstrakcja polega na tworzeniu modelu obiektów, które odzwierciedlają rzeczywiste obiekty lub koncepty. W JS abstrakcja może być osiągnięta poprzez tworzenie klas i obiektów, które reprezentują rzeczywiste obiekty i operacje.
6. Kapsułkowanie: Kapsułkowanie polega na ukrywaniu wewnętrznych szczegółów obiektów i udostępnianiu tylko niezbędnych interfejsów. W JS można osiągnąć kapsułkowanie za pomocą zamknięć (closures) i właściwości dostępu (getters i setters).
7. Hermetyzacja: Hermetyzacja to kontrolowanie dostępu do danych i metod obiektu. W JS można stosować hermetyzację, używając zasięgu zmiennych (scope) i właściwości dostępu.

Zadanie 7

// Przykładowy ciąg tekstowy

let tekst = "To jest przykładowy tekst";

// 1. Metoda length - Zwraca długość ciągu tekstowego.

let dlugosc = tekst.length; // Wynik: 25

// 2. Metoda toUpperCase - Zamienia wszystkie litery na duże.

let duzeLitery = tekst.toUpperCase(); // Wynik: "TO JEST PRZYKŁADOWY TEKST"

// 3. Metoda toLowerCase - Zamienia wszystkie litery na małe.

let maleLitery = tekst.toLowerCase(); // Wynik: "to jest przykładowy tekst"

// 4. Metoda slice - Wycina fragment ciągu tekstowego na podstawie indeksów.

let fragment = tekst.slice(3, 10); // Wynik: "jest pr"

// 5. Metoda indexOf - Znajduje pierwsze wystąpienie określonego tekstu.

let pozycja = tekst.indexOf("jest"); // Wynik: 3

// 6. Metoda replace - Zastępuje pierwsze wystąpienie tekstu innym tekstem.

let nowyTekst = tekst.replace("przykładowy", "nowy"); // Wynik: "To jest nowy tekst"

// 7. Metoda split - Dzieli ciąg tekstowy na tablicę elementów na podstawie określonego separatora.

let podzial = tekst.split(" "); // Wynik: ["To", "jest", "przykładowy", "tekst"]

// 8. Metoda concat - Łączy dwa ciągi tekstowe w jeden.

let dodatkowyTekst = "Dodatkowy tekst";

let polaczenie = tekst.concat(" ", dodatkowyTekst); // Wynik: "To jest przykładowy tekst Dodatkowy tekst"

// 9. Metoda charAt - Zwraca znak na określonej pozycji w ciągu.

let znak = tekst.charAt(0); // Wynik: "T"

// 10. Metoda trim - Usuwa białe znaki (spacje, tabulatory itp.) z początku i końca ciągu.

let tekstZBialymiZnakami = "   Tekst z białymi znakami   ";

let oczyszczonyTekst = tekstZBialymiZnakami.trim(); // Wynik: "Tekst z białymi znakami"

// Wyświetlenie wyników w konsoli

console.log("Długość ciągu:", dlugosc);

console.log("Duże litery:", duzeLitery);

console.log("Małe litery:", maleLitery);

console.log("Wycięty fragment:", fragment);

console.log("Pozycja 'jest':", pozycja);

console.log("Zamieniony tekst:", nowyTekst);

console.log("Podział na słowa:", podzial);

console.log("Połączony tekst:", polaczenie);

console.log("Pierwszy znak:", znak);

console.log("Oczyszczony tekst:", oczyszczonyTekst);

Zadanie 8

// Zdefiniuj ciąg tekstowy, w którym będziemy szukać.

let tekst = "To jest przykładowy tekst zawierający kilka słów.";

// Określ szukany ciąg tekstowy.

let szukanyCiąg = "przykładowy";

// Inicjalizacja zmiennej, która będzie przechowywać pozycję znalezionego ciągu.

let pozycja = -1;

// Rozpocznij pętlę do przeszukiwania ciągu tekstowego.

for (let i = 0; i < tekst.length - szukanyCiąg.length + 1; i++) {

  // Sprawdź, czy ciąg tekstowy na danej pozycji zgadza się z szukanym ciągiem.

  let podciąg = tekst.slice(i, i + szukanyCiąg.length);

  if (podciąg === szukanyCiąg) {

    // Jeśli znaleziono szukany ciąg, zaktualizuj pozycję i przerwij pętlę.

    pozycja = i;

    break;

  }

}

// Sprawdź, czy znaleziono ciąg, i wyświetl odpowiedni komunikat.

if (pozycja !== -1) {

  console.log(`Znaleziono ciąg "${szukanyCiąg}" na pozycji ${pozycja}.`);

} else {

  console.log(`Ciąg "${szukanyCiąg}" nie został znaleziony.`);

}

Opis Algorytmu:

1. Zdefiniuj ciąg tekstowy, w którym będziemy szukać.
2. Określ ciąg tekstowy, który chcemy znaleźć (szukanyCiąg).
3. Inicjalizuj zmienną pozycja na wartość -1, która będzie przechowywać pozycję znalezionego ciągu (jeśli zostanie znaleziony).
4. Rozpocznij pętlę for, aby przeszukać ciąg tekstowy. Iteruj od 0 do (tekst.length - szukanyCiąg.length + 1) - to ograniczenie pozwala uniknąć niepotrzebnych iteracji, gdy długość szukanego ciągu jest większa od pozostałego miejsca w tekście.
5. W każdej iteracji pętli użyj metody slice do wyodrębnienia fragmentu tekstu o długości równą długości szukanyCiąg od bieżącej pozycji i.
6. Porównaj wyodrębniony podciąg z szukanyCiąg przy użyciu operatora === (porównanie dokładne).
7. Jeśli podciąg jest równy szukanyCiąg, zaktualizuj zmienną pozycja na bieżącą pozycję i i przerwij pętlę.
8. Po zakończeniu pętli sprawdź wartość zmiennej pozycja. Jeśli nie jest równa -1, to znaczy, że ciąg został znaleziony i wyświetl komunikat z jego pozycją. W przeciwnym razie wyświetl komunikat o braku znalezienia ciągu.

Schemat blokowy czesc 1

Rozpocznij

1. Wyświetl komunikat: "Podaj liczbę oczek na kostce:"
2. Wprowadź liczbę oczek od użytkownika i zapisz ją jako "LiczbaOczek"
3. Wygeneruj losową liczbę całkowitą od 1 do "LiczbaOczek" i zapisz ją jako "WynikLosowania"
4. Wyświetl komunikat: "Wynik losowania to: [WynikLosowania]"
5. Zakończ

Schemat blokowy cz 2

1. Wyświetl komunikat: "Podaj liczbę kostek do rzutu:"
2. Wprowadź liczbę kostek od użytkownika i zapisz ją jako "LiczbaKostek"
3. Wyświetl komunikat: "Podaj liczbę oczek na każdej kostce:"
4. Wprowadź liczbę oczek na kostce od użytkownika i zapisz ją jako "LiczbaOczek"
5. Zainicjuj zmienną "WynikSumaryczny" na 0
6. Dla każdej kostki od 1 do "LiczbaKostek" wykonuj:
   1. Wygeneruj losową liczbę całkowitą od 1 do "LiczbaOczek" i zapisz ją jako "WynikLosowania"
   2. Dodaj "WynikLosowania" do "WynikSumaryczny"
7. Wyświetl komunikat: "Wynik sumaryczny rzutu to: [WynikSumaryczny]"
8. Zakończ