Notatki do nauki

--------------------------------------------------------------------------------------------------------------------------------------

- - - JavaScript - - -

--------------------------------------------------------------------------------------------------------------------------------------

**1. Zmienne i stałe oraz różne typy danych:**

* Procedura tworzenia zmiennych:

Aby utworzyć zmienną używamy słowa „let” lub słowa „var” a po nim wstawiamy identyfikator czyli nazwę zmiennej np. let a = 22; Jeżeli chcemy ponownie odnieść się do tej zmiennej i nadpisać inną wartość, to już nie musimy posługiwać się słowem „let”.

Możemy użyć też słowa const przed identyfikatorem uzyskując efekt otrzymania stałej, których wartość nie może zmienić się w naszym programie np. const name = „Anna”. Nie możemy nadpisać stałej inną wartością ponieważ w konsoli pojawi się błąd.

* Tworzenie zmiennych:

- stosujemy konwencję camelCase gdy zmienna ma składać się z więcej niż jednego słowa

- nazwa zmiennej może zaczynać się od znaku „\_” lub „$” np. let \_nazwaZmiennej;

- stosujemy angielskie nazwy zmiennych

* Jak nie można tworzyć nazw zmiennych:

- nazwa zmiennej nie być słowem kluczowym takim jak np. let let; let for; itd.

- nazwa zmiennej nie może mieć w środku kropki ani przecinka np. let nazwa.zmiennej; itp.

- nazwa zmiennej nie może być rozdzielona spacją np. let nazwa zmiennej; itp.

- nazwa zmiennej nie może rozpoczynać się od cyfry np. let 1nazwa;

* Wartości liczbowe zmiennych:

- część ułamkową liczby zapisujemy po kropce np. let pi = 3.14;

- zapis w systemie szesnastkowym następuje po przedrostku 0x w wartości zmiennej

- dla liczb ułamkowych rozpoczynających się od „0” można rozpocząć wartość od „.” czyli: .123

* Łańcuchy znaków string:

String to prosty typ danych będący łańcuchem znaków, zwany również literałem łańcuchowym lub literałem znakowym. Zasady stosowania łańcucha znaków string są następujące:

- możemy stosować dwa typy zapisu: let str = ”Hello World!”; lub let str = ’Hello World!’

- możemy łączyć łańcuchy znaków poprzez operator konkatenacji „+”

- możemy używać polskich znaków ponieważ łańcuch string obsługuje znaki unicode

Przykład:

let str = "Hello World!";

let str2 = 'Hello World';

let msg = str2 + " Hey! ";

console.log(msg);

let strlenght = msg.length;

console.log(strlenght);

* Literał szablonu:

Literał umożliwia pisanie łańcuchów znaków w kilku wierszach. Należy używać backtick zamiast cudzysłowu. Przykład:

let str = `Hello

World

!`;

Inna możliwość łączenia stringów:

let txt = " Początek tekstu ";

let txt2 = `${txt} koniec tekstu`;

* Wartość logiczna boolean:

Boolean to typ danych będący wartością logiczną. Ma tylko dwie możliwe wartości true i false.

Przykład:

let b1 = true;

console.log(b1); // true

let b2 = 23 < 1;

console.log(b2); // false

let b3 = 34 == 32;

console.log(b3); // false

* Wartość liczbowa BigInt:

Jest to typ wartości pozwalający na zapisanie bardzo dużych liczb całkowitych, większych od maksymalnej liczby całkowitej jaką możemy zapisać dla naszych zmiennych lub stałych. Aby posłużyć się BigInt na końcu wartości stosujemy literę „n”, tak jak w przykładzie poniżej:

let numer = 928374982742983479263478264727398n;

* Wartość undefined:

Wartość automatycznie przypisywana do zmiennej, która nie otrzymała żadnej wartości początkowej. Przykładowo:

let data;

console.log(data); // Rezultat wyświetlony w konsoli to undefined

* Słowo kluczowe null:

Słowo kluczowe oznaczające pustą wartość i jest ono stosowane aby pokazać, że dana zmienna

nie ma już wartości. Przykład:

let taskToDo = "task 1";

taskToDo = null;

* Typy wartości zmiennych i stałych:

JavaScript nie ma typów zmiennych, ma typy wartości, które są przypisywane do zmiennych lub stałych. Przykłady:

let str = "Asia"; // string

let num = 23; // numer

let bool = true; // boolean

let bigInt = 975793872938749823749832479832n; // bigint

let data; // undefined

let txt = "Kasia";

txt = null; // null

let uniqueSymbol = Symbol(); // symbol

* Zmienne oraz stałe niezadeklarowane i niezdefiniowane:

Zmienna lub stała niezadeklarowana undeclared to taka, która nie była powołana do życia za pomocą let, var lub const. Zmienna niezdefiniowana undefined to taka, która została utworzona ale przypisanej wartości początkowej.

Zmienna lub stała zadeklarowana jest utworzona z let, var lub const, a gdy ma przypisaną wartość to jest dodatkowo zdefiniowana.

let a; // Zmienna zadeklarowana ale niezdefiniowana

console.log(a); // Konsola zwraca undefined

let b = 4; // Zmienna zadeklarowana i zdefiniowana

console.log(b); // Konsola zwraca 4

c; // Zmienna niezadeklarowana i niezdefiniowana

console.log(c); // Konsola zwraca Uncaught ReferenceError: c is not defined

* Zmienne lokalne i globalne:

Zmienne globalne mają zasięg dostępności w całym programie. Zmienne lokalne mają zasięg dostępności w danym bloku kodu, przykładowo tylko wewnątrz instrukcji if. Wewnątrz bloku kodu zmienna lokalna o nazwie takiej samej jak zmienna globalna, znajdująca się poza tym blokiem dokonuje przesłonięcia zmiennej globalnej. Przykład:

let a = 10;

const name = "Asia";

if (true) {

let b = 100; // Zmienna lokalna wewnątrz bloku instrukcji if

console.log(b); // Jest dostęp do zmiennej lokalnej b z wnętrza bloku

console.log(a); // Jest dostęp do zmiennej globalnej a z wnętrza bloku

// Stała name lokalna przesłania stałą name globalną

const name = "Kasia";

// Odwołanie się do stałej name powoduje wywołanie stałej lokalnej

console.log(name);

}

// console.log(b); // Nie ma dostępu do zmiennej lokalnej b z zewnątrz bloku

/\* Odwołanie się do stałej name poza blokiem if powoduje wywołanie

stałej globalnej name, czyli w konsoli otrzymujemy Asia. \*/

console.log(name);

* Zmienna var:

Zanim w ES2015/ES6 wprowadzono let i const w JavaScript mieliśmy tylko dostępne var do powoływania zmiennych. Zmienna var nie ma zasięgu blokowego. Nieważne czy użyjemy var wewnątrz, czy na zewnątrz bloku kodu, zawsze mamy dostęp do takiej zmiennej zaróno w wnętrza bloku jak i z zewnątrz bloku kodu. Zmienna var wewnątrz bloku nadpisuje wartość zmiennej o tej samej nazwie, gdy taka jest wcześniej powołana do życia poza blokiem. Przykład:

var a = 10;

if (true)

{

console.log(a); // Konsola zwraca wartość 10

var a = 20;

console.log(20); // Konsola zwraca wartość 20

var b = 30;

var a = 40;

}

console.log(b); // Konsola zwraca wartość 30

console.log(a); // Konsola zwraca wartość 40

* Scope zakres zmiennych:

Zakres w większości języków programowania definiowany jest przez blok kodu, ale nie w JavaScript. W JavaScript istnieje zakres globalny oraz lokalny. Zakres tzw. scope to widoczność zmiennych dla poszczegónych treści programu.

Zakres globalny istnieje przez całe życie skryptu, raz zadeklarowana zmienna globalna już tam pozostaje chyba, że uzyjemy na niej delete. Globalna zmienna może być używana wewnątrz oraz poza funkcją, w praktyce gdziekolwiek w bieżącym dokumencie. W przeglądarce window jest obiektem globalnym.

Zakres lokalny istnieje wewnątrz funkcji, w której zmienna zostałą zadeklarowana, również argumenty funkcji. Domyślnie JavaScript nie ma zakresu blokowego z var co powoduje pewne problemy.

let g1 = 10; // Zmienna globalna

/\* Zmienna var umożliwia nadpisanie zmiennej przy dwukrotnym użyciu

słowa kluczowego var, jednak unikamy takiego zapisu \*/

var g2 = 20; // Zmienna globalna

var g2 = 99; // Zmienna globalna

if (1==1) {

var g3 = 100; // Zmienna globalna

var g2 = 111; // Zmienna globalna

window.g4 = 200; // Globalna zmienna, której unikamy

}

for (var i=0 ; i < 5 ; i++) {} // Nawet w for zmienna var nadpisuje się globalnie

console.log(g2);

console.log(g3);

console.log(i);

// Zmienna g3 ma zasięg globalny mimo, że jest wewnątrz instrukcji if.

// Zmienna g2 użyta wewnątrz instrukcji if nadal nadpisuje wartość 99 liczbą 111.

**2. Obiekty i metody:**

* Obiekt:

Obiektem nazywamy typ złożony, który posiada różne dane jako pary właściwości i wartości.

* Notacja literałowa:

Obiekt można tworzyć go na różne sposoby. Jednym z najbardziej popularnych jest literał obiektu, czyli notacja literałowa. Przykładowy literał obiektu:

let user1 = {

name : "Kasia",

city : "Wawa",

age : 30,

favColor : "red" };

// Wywołanie jednej z właściwości obiektu w konsoli

console.log(user1.name);

console.log(user1.city);

// Nadpisanie wewntrznej zmiennej obiektu czyli właściwości

user1.city = "Krk";

console.log(user1.city);

// Wywołanie całej zawartości obiektu w konsoli

console.log(user1);

* Obiekt arguments w funkcji:

Obiekt arguments w funkcji posiada przekazane argumenty przy wywołaniu funkcji. Obiekt ten dostępny jest tylko w zwykłych funkcjach, nie ma go w funkcjach strzałkowych. Nawet jeżeli funkcja nie posiada parametrów, to gdy mimo to przekazujemy jej argumenty wywołując ją, to te argumenty zostają zapamiętywane w obiekcie arguments.

function sumaLiczb()

{

let suma = 0;

for (let i = 0; i < arguments.length; i++)

{

suma += arguments[i];

}

return suma;

}

// Rezultatem jest suma przekazanych argumentów

let wynik = sumaLiczb(1,2,3,4,5,6,7,8,9);

console.log(wynik);

* Metody:

Metody to wewnętrzne funkcje obiektu, które zwykle operują na danych tego obiektu. Metoda odwołuje się do danych swojego obiektu za pomocą this.nazwaZmiennej.

let phone = {

brand: "Sony",

model: "Xperia",

manufactureDate: 2023,

connect: function (message) // Funkcja w obiekcie, czyli metoda.

{

console.log("Łączę się i przekazuje: " + message);

},

getInfo: function ()

{

console.log(this.brand, this.model);

}

}

// Wywołanie metod

phone.connect("Hello!");

phone.getInfo();

* Obiekt przykładowy:

let carFactory = {

name: "Tesla",

city: "Toronto",

employees: [], // Tablica z pracownikami

addEmploee: function (name, surname) { // Wyrażenie funkcyjne dodające osobę

const obj = { // Obiekt w obiekcie

name: name,

surname: surname,

email: `${name.toLowerCase()}.${surname.toLowerCase()}@example.com`

// W emailu stosujemy literał szablonu

};

// Zmienna index zliczająca ilość pracowników w tablicy

let index = this.employees.length;

// Dodawanie pracowników do tablicy employees

this.employees[index] = obj;

},

showEmploee: function () { // Wyrażenie funkcyjne pokazujące danego pracownika

console.log(this.name, "employees");

for (let i = 0 ; i < this.employees.length ; i++) {

const e = this.employees[i];

console.log(e.name, e.surname, e.email);

}

}

};

// Dodawanie pracowników

carFactory.addEmploee("Kasia","Kowalska");

carFactory.addEmploee("Ola","Nowak");

carFactory.addEmploee("Ala","Nowicka");

console.log(carFactory.employees); // Pokazanie tablicy z pracownikami

carFactory.showEmploee(); // Pokazanie listy z pracownikami

* Enumeracja właściwości obiektu:

Enumeracji właściwości obiektu dokonujemy przy użyciu pętli for in.

let obiekt = {

a: 18,

b: "Anna",

info: function() {console.log("Hej");},

tablica: [1,2,3]

};

// Stała "prop" przyjmująca z pętli kolejną właściwość obiektu

// "in" słowo kluczowe dla pętli for in

for (const prop in obiekt) {

if (obiekt.hasOwnProperty(prop)) {

console.log(prop, obiekt[prop]); // Pokazanie właściwości i ich wartości

}

}

Instrukcja warunkowa if sprawdza czy właściwość należy do obiektu. Sprawdzamy to ponieważ każdy obiekt dziedziczy pewne elementy od głównego obiektu JavaScript. Potrzebne jest wywołanie dla obiektu hasOwnProperty(nazwaStałej) żeby sprawdzić, czy właściwość bezpośrednio należy do naszego obiektu.

* Gettery i settery w obiektach:

Jeżeli dodamy setter do obiektu dla danej zmiennej, to zostanie on wywołany automatycznie przy zapisie wartości do tej zmiennej. Podobnie będzie z getterem przy pobieraniu wartości.

const user = {

// Podkreślenie \_ daje informacje innym, że wartość będzie używana

// tylko wewnątrz tego obiektku

\_name: null,

// Setter

set name(newValue) {

this.\_name = newValue;

},

// Getter

get name() {

return this.\_name;

}

};

// Zapisanie wartości w \_name wywołując setter

user.name = "Asia";

// Sprawdzenie czy wstawienie wartości do \_name się odbyło wywołując getter

console.log(user.name);

* Object window i jego właściwości oraz metody:

Object window to ogólnodostępny obiekt w przeglądarce, w której działa

JavaScript, udostępnia nam wiele przydatnych informacji, o aktualnej

karcie lub oknie przeglądarki.

\*/

// Szerokość i wysokość okna przeglądarki

console.log(window.innerWidth);

console.log(window.innerHeight);

/\* Można wywołać polecenie bez window ponieważ aplikacja w przeglądarce

działa w kontekście obiektu window. Dopiero po kliknięciu na alert

w przeglądarce wykonuje się dalszy kod. \*/

alert("Hello!") // Alert wyświetlany w osobnym okienku

// Pobieranie informacji od użytkownika

let userInput = prompt("Podaj swoje imię: ");

console.log("Twoje imię to: ",userInput);

// Pobieranie cyfry i wykonanie dodawania

let userInput2 = prompt("Podaj wiek: ");

let wiek = parseInt(userInput2); // Trzeba parsować ponieważ byłby łańcuch znaków

console.log("Twoj wiek za 2 lata to: ",wiek+2);

// Uruchomienie fragmentu kodu po jakimś czasie, tutaj 1 sekunda

setTimeout(function () {

console.log("Wait...");

}, 1000);

// Ustawienie interwału ciągłego wykonywania się fragmentu kodu

let i = 0;

let interwalPierwszy = setInterval(() => {

console.log(i);

i++;

}, 100);

// Ustawienie zakończenia cyklicznego wywoływania kodu po 4 sekundach

setTimeout(() => {

clearInterval(interwalPierwszy);

}, 4000);

* Zdarzenia obiektu window:

Event handlers to zmienne, pod które możemy podpiąć nasze funkcje i będą wywołane przez np. przeglądarkę gdy zajdzie jakieś zdarzenie. Przykładowo zdarzenie load będzie wywołane, gdy wszystkie elementy witryny zostaną załadowane, czyli kod html, css, obrazki itd. Aby podpiąć się pod zdarzenie load wystarczy użyć polecenia "window.onload", do którego przypiszemy naszą funkcję.

// Po załadowaniu strony konsola komunikkuje nam ten fakt

window.onload = function () {

console.log("Strona załadowana");

};

// W trakcie zmieniania rozmiaru okna konsola podaje jego aktualne wymiary

window.onresize = function () {

console.log(window.innerWidth, window.innerHeight);

}

* Zdarzenia z addEventListener:

Zdarzenia możemy również podpiąć z wykorzystaniem funkcji addEventListener. Określamy rodzaj zdarzenia oraz funkcję, jaka ma być wywołana po pojawieniu się tego zdarzenia.

window.addEventListener("load", function (event) {

console.log("Strona załadowana!");

console.log(event); // Tu informacyjnie obiekt event pokazuje swoją zawartość

});

window.addEventListener("resize", function(){

console.log(window.innerHeight, window.innerWidth);

});

**3. Tablice:**

* Tablice:

Tablice to specjalne obiekty przechowujące różne dane. Do poszczególnych wartości możemy dostać się dzięki indeksom oraz nawiasom kwadratowym. Przykładowo:

let names = ["Asia", "Kasia", "Karol"]; /\* Wpisywanie danych do tablicy \*/

// Wywołanie w konsoli poszczególnych wartości z tablicy

console.log(names[0]);

console.log(names[2]);

console.log(names[names.length-1]);

// Wprowadzanie i nadpisywanie pojedyńczych wartości do tablicy

names[3] = "Kuby";

names[1] = "Ola";

// Wywoływanie w konsoli całej zawartości tablicy

console.log(names);

/\* Przechowanie wartości całkowicie różnych typów

np. liczbowe, logiczne, tekstowe i obiekty \*/

let data = [1, true, "Marek", {a: "test"}];

console.log(data);

* Tablice const:

Nawet jeśli tablica lub obiekt są const, to można nadpisywać pojedyncze wartości, które się wewnątrz nich znajdują. Const ogranicza się do tego, że nie można stałej nadpisać nową całą tablicą lub nowym całym obiektem. Przykładowo:

const primitives = ["string", "number", "bigint", "boolean", "null", "undefined", "symbol"];

console.log(primitives);

primitives[0] = "string !!!";

console.log(primitives);

**4. Instrukcje warunkowe:**

* Instrukcja if:

Instrukcja warunkowa if dodaje obsługę logiki w naszych programach. Sprawdza czy warunek jest spełniony, czy daje wartość true, jeżeli tak to następniewykonywany jest blok kodu w nawiasach po instrukcji if.

if (2 <= 9)

{

console.log("true"); // Ten kod się wykonuje

}

if (2 <= 1)

{

console.log("true"); // Ten kod się nie wykonuje

}

// Jeżeli cała instrukcja mieści się w jednej linii to możemy pominąć nawiasy

if (5 < 21) console.log("true");

* Instrukcja if else:

let data = 6;

if (data > 5)

{console.log("data większe od 5");}

else

{console.log("data mniejsze od 5");}

* Instrukcja if else if:

Kolejne instrukcje else if wykonują się po kolei:

let age = 19;

if (age == 17)

{console.log("niepełnoletni");}

else if (age == 18)

{console.log("pełnoletni 18-stolatek");}

else if (age == 19)

{console.log("pełnoletni 19-stolatek");}

else if (age > 19)

{console.log("pełnoletni");}

else

{console.log("dziecko");}

* Instrukcja switch:

Pozwala na proste porównanie czy dana zmienna posiada jedną z wartości. Przykład:

let num = 10;

switch(num)

{

case 5:

console.log("num jest 5");

break;

case 10:

console.log("num jest 10");

break;

case 12:

console.log("num jest 12");

break;

default:

console.log("default");

}

* Instrukcja switch z wieloma przypadkami:

let num = 4;

switch(num)

{

case 2:

case 4:

case 6:

case 8:

case 10:

console.log("num jest parzyste");

break;

case 1:

case 3:

case 5:

case 7:

case 9:

console.log("num jest nieparzyste");

break;

default:

console.log("default");

}

**5. Pętle:**

* Pętla while:

Pętla pozwalająca na wielokrotne wykonanie bloku kodu gdy warunek jest spełniony. Przykład:

let i = 0;

while (i < 5) {

console.log(i);

i = i + 1; }

let num = 4;

while(num < 10) {

console.log(num);

num++; }

num = 10;

while(num > 0) {

console.log(num);

num--; }

* Iteracja tablicy przy użyciu pętli while wraz z sumowaniem elementów:

const tab = [10,20,30,40];

let i = 0; let sum = 0;

while (i < tab.length)

{

const element = tab[i];

console.log(element);

sum = sum + element;

i++;

}

console.log("Suma wynosi: " + sum);

* Pętla do while:

Pozwala nam na jednokrotne początkowe wykonanie się iteracji zanim zostanie sprawdzony warunek dalszego iterowania się kodu.

Przykład:

let a = 5;

do {

console.log(a);

a--; }

while (a > 0)

* Pętla for:

for (let i = 0 ; i < 3 ; i++)

{

console.log(i);

}

**6. Polecenia i wyrażenia oraz operatory:**

* Polecenie (statement) jest to grupa słów, liczb albo operatorów, które mają na celu wykonanie jakiegoś zadania. Polecenie składa się z jednego lub większej ilości wyrażeń. Przykład polecenia:

let numer = 10;

* Wyrażenie to pewna część kodu, którego wynikiem jest powstała wartość. Jest to składowa polecenia. Przykładowo poleceniem jest poniższy zapis:

let data = 10 + 7;

Natomiast wyrażeniem jest: 10 + 7 lub data. Dokładniej opisując mamy w powyższym poleceniu następujące elementy:

data wyrażenie zmiennej

= operator przypisania wartości

10 wyrażenie w postaci literalnej wartości

10 + 7 wyrażenie matematyczne

Polecenia możemy kończyć średnikiem ale nie musimy. Dobrym zwyczajem jest przyjęcie w całym programie jednolitej konwencji.

* Operator umożliwia operacje na operandach. Operatory mogą operować na jednym, dwóch a czasami trzech operandach. Wyrażenia są stworzone z operatorów, które działają na operandach. Przykładowo:

const name = "Zuza"; // Operator przypisania =

let number = 1;

number++; // Operator inkrementacji

let wynik = 2 + 3;

// 2 i 3 to operandy

// + i = to operatory

// 2 + 3 to wyrażenie

* Operatory przypisania:

let n = 0;

n = n + 2;

n += 2;

console.log(n);

n -= 2;

console.log(n);

n \*= 2;

console.log(n);

n /= 2;

console.log(n);

n \*\*= 3; // Potęgowanie do trzeciej liczby przypisanej do n

console.log(n);

* Operator modulo:

Operator ten pozwala uzyskać resztę z dzielenia. Przykładowo:

let wynik = 5 % 2; // Reszta z dzielenia 5 przez 2 to 1

console.log(wynik);

wynik = 4 % 2; // Reszta z dzielenia 4 przez 2 to 0

console.log(wynik);

let x = 6;

let y = 3;

x %= y; // To polecenie daje ten sam wynik co polecenie x = x % y

console.log(x);

console.log(5 % 3); // Reszta z dzielenia 5 przez 3 to 2

* Operatory inkrementacji i dekrementacji:

Inkrementacja i dekrementacja są to jednoargumentowe operatory, które dzielą się na warianty: postinkrementacja, preinkrementackja, postdekrementacja i predekrementacja.

let n = 10;

n++; // Inkrementacja

console.log(n); // Wynik to 11

n--; // Dekrementacja

console.log(n); // Wynik to 10

console.log(n++); // Wynik to dalej 10 bo jest to postinkrementacja

console.log(n); // Wynik to już 11 bo powyżej była postinkrementacja

console.log(--n); // Wynik to 10 bo jest to predekrementacja

console.log(n); // Wynik to nadal 10 bo powyżej była predekrementacja

* Operator trójargumentowy:

Ternary, czyli operator trójargumentowy zwany również trójelementowym to skrócona instrukcja if else. Przykład:

let wiekOsoby = 16;

if (wiekOsoby >= 18)

{console.log("Pełnoletnia");}

else

{console.log("Niepełnoletnia");}

let wiekJana = (wiekOsoby >= 18) ? "Dorosła" : "Nieletnia";

console.log(wiekJana);

* Operator przecinka:

Operator ten pozwala na zapisanie wielu wyrażeń po sobie. Przykładowo:

// Tworzenie zmiennych jedna po drugiej

let a = 5, b = 10;

console.log(a + " " + b);

// Inkrementacja i dekrementacja jednocześnie dwóch elementów i oraz j w pętli

for (let i = 0, j = 10 ; i < 12 ; i++, j--) {

console.log("i: " + i + " " + "j: " + j); }

* Operatory porównania a obiekty:

// Operatory porównania

console.log(1<2);

console.log(1<=2);

console.log(3>2);

console.log(3>=2);

console.log(3==3);

console.log(3!=2);

Operatory równości w przypadku zmiennych powodują, że przypisując wartość jednej zmiennej (np. a) innej zmiennej (np. b), tworzy się kopia tej wartości w pamięci. Czyli są to dwa oddzielne elementy w pamięci.

let a = 5; // Wartość 5

let b = a; // Wartość 5, kopia

Operatory równości w przypadku sprawdzania obiektów tylko stwierdzają, czy dwa obiekty mają ten sam adres w pamięci. Uchwyty do obiektów mogą być przechowywane w wielu zmiennych i stałych, bo to typ złożony a nie prymityw.

// Obie zmienne obj1 i obj2 wskazują na ten sam obiekt

let obj1 = { data: 20,

age: 30,

tel: 123456789 };

let obj2 = obj1;

// Wartość true ponieważ zmienne obj1 i obj2 wskazują na ten sam obiekt

console.log(obj1 == obj2);

* Ścisłe operatory porównania:

Luźne porównanie ==, w którym następuje automatyczna konwersja na jeden wspólny typ, czyli łańcuch znaków został zamieniony na liczbę. Jest to konwersja wymuszona przez porównanie.

console.log(4 == "4"); // Konsola zwraca wartość true

Ścisłe operatory porówniania === sprawdzają zarówno czy obie wartości są równe oraz czy ich typ jest taki sam. Jeżeli wartości są takie same ze względu na automatyczną konwersję, ale typy się nie zgadzają, to zwrócone zostanie false.

console.log(4 === "4"); // Konsola zwraca wartość false

* Operatory jednoargumentowe:

Unary operators, czyli operatory jednoargumentowe przyjmujące pojedynczy operand.

// Operator ++ oraz --

let a = 10;

a++;

console.log(a);

// Operator delete

let obiekt1 = {age: 19, name: "Anna"};

delete obiekt1.age;

console.log(obiekt1);

// Operator + oraz -

let b = "20" + 10;

console.log(b);

b = +"20" + 10; // + powoduje konwersję łańcucha do liczby

console.log(b);

* Operatory relacyjne:

Relational operators, czyli operatory relacyjne porównują operandy i zwracają true jeżeli spełnione jest porównanie.

// Operator in

let obiekt = { data: "a" };

if ("data" in obiekt) console.log("Data jest w obiekt")

let tablica = [1,2,3,4,5];

if (2 in tablica) console.log("2 jest w tablica");

// Operator instanceof

let date = new Date(2030, 11, 24);

console.log(date);

if (date instanceof Date) console.log("Jest to instancja z Date")

* Operator typeof:

Operator zwracający typ wartości w zmiennej lub stałej. Typ określony jest jako łańcuch znaków.

console.log(typeof 12); // Konsola zwraca "number"

console.log(typeof "str"); // Konsola zwraca "string"

console.log(typeof false); // Konsola zwraca "boolean"

console.log(typeof {a: 12}); // Konsola zwraca "object"

console.log(typeof [1,2,3]); // Konsola zwraca "object"

console.log(typeof undefined); // Konsola zwraca "undefined"

console.log(typeof 10n); // Konsola zwraca "bigint"

console.log(typeof Symbol()); // Konsola zwraca "symbol"

console.log(typeof console.log); // Konsola zwraca "function"

console.log(typeof null); // Konsola zwraca "object"

Gdy nie wiemy jaki jest typ zmiennej lub stałej możemy posłużyć się operatorem typeof.

let data = "Anna";

if (typeof data == "string") console.log("Łańcuch znaków");

* Operatory and or not:

/\* Operator and \*/

console.log(true && true); // Konsola zwróci true

console.log(true && false); // Konsola zwróci false

console.log(false && false); // Konsola zwróci false

console.log(false && true); // Konsola zwróci false

/\* Operator or \*/

console.log(true || true); // Konsola zwróci true

console.log(true || false); // Konsola zwróci true

console.log(false || true); // Konsola zwróci true

console.log(false || false); // Konsola zwróci false

console.log(false || false || false); // Konsola zwróci false

console.log(true || false || false); // Konsola zwróci true

/\* Operator not \*/

console.log(!false); // Konsola zwróci true

console.log(!true); // Konsola zwróci false

// Negacja najpierw wymusza automatyczną konwersję liczby 1 do boolean true

console.log(!1); // Konsola zwróci false

console.log(!!1); // Konsola zwróci true bo podwójna negacja

* Operator rest:

Operator rest w postaci trzech kropek ... zbiera przekazane do funkcji argumenty w tablicę.

function funkcjaTestowa(...zmienna)

{

console.log(zmienna);

}

funkcjaTestowa(1,2,3,4,5,6,7);

Wywołanie funkcji spowodowało zwrócenie przez konsolę tablicy z liczbami 1,2,3,4,5,6,7.

* Operator spread:

Operator spread w postaci trzech kropek ... rozbija tablicę lub obiekt na pojedyncze wartości.

function funkcjaTestowa(a,b,c)

{

console.log(arguments);

}

let tab = [99,100,101];

funkcjaTestowa(...tab); // Przekazano argumenty z rozbitej tablicy tab

let obiektTestowy = {

name: "Jan",

city: "Warszawa"

};

let osoba = {

...obiektTestowy, // Tu przekazano rozbite elementy składowe z obiektTestowy

tel: 123456789,

email: "jan@kowalski.com"

};

console.log(osoba);

**7. Funkcje:**

* Tworzenie i wywoływanie funkcji:

Funkcje pozwalają na wielokrotne wywoływanie jakiegoś kodu. Tworzymy je za pomocą słowa kluczowego function oraz dowolnej nazwy funkcji. Wywołujemy ją za pomocą nazwy i nawiasów okrągłych.

// Deklaracja funkcji z parametrami a,b,c

function sumaTrzechLiczb(a,b,c)

{

let suma = a + b + c;

console.log(suma);

}

// Wywołanie funkcji z przekazanymi argumentami 1,2,3

sumaTrzechLiczb(1,2,3); // Konsola zwraca wynik 6

* Zwracanie wartości z return:

function sumaLiczb(a,b) {

let suma = a + b;

return suma; }

let data = sumaLiczb(2,2);

console.log(data); // Konsola zwraca sumę 4

function osoba(name)

{

if (name == "Asia")

{

return

}

else

{

console.log("Nie jesteś Asią!");

}

}

osoba("Ola"); // Konsola zwraca "Nie jesteś Asią!"

osoba("Asia"); // Konsola nic nie zwraca

* Zwracanie obiektów z funkcji:

Funkcje mogą zawierać również obiekty.

// Funkcja z parametrami tworząca i zwracająca obiekt

function createUser(name, email, city)

{

if (name.lenght < 3) return null;

if (email.lenght < 6) return null;

if (email.indexOf("@") < 0) return null;

if (city.lenght < 3) return null;

let obiekt =

{

name: name,

email: email,

city: city,

type: "employee"

};

return obiekt;

}

// Utworzenie obiektu za pomocą funkcji oraz jego pokazanie w konsoli

let user1 = createUser("Anna", "anna@domenapocztowa.com", "Warszawa");

console.log(user1);

* Funkcja anonimowa:

Wyrażenie funkcyjne, czyli function expression to funkcja anonimowa przypisana do zmiennej lub stałej. Funkcja anonimowa to funkcja bez nazwy.

// Funkcja anonimowa przypisana do zmiennej

let showName = function (name) {

console.log(name);

}

// Wywołanie funkcji przy użyciu zmiennej

showName("Ania");

// Referencje do tej funkcji możemy przekazać do innej zmiennej lub stałej

let test = showName;

test("Kasia");

Funkcja callback:

Funkcja przekazana jako argument do innej funkcji, dzięki czemu może być wewnątrz wywołana jeśli zachodzi taka potrzeba.

function pokazDane(data)

{

console.log(data);

}

function pokazTablice(tab, callback)

{

for(let i = 0; i <= 4; i++)

{

let x = tab[i];

callback(x); // Wywołanie funkcji pokazDane

}

}

pokazTablice([1,2,3,4,5], pokazDane);

* Przekazywanie danych przez wartości lub referencję:

Prymitywy przekazywane są przez kopię, zaś obiekty przez referencję, czyli przez uchwyt do miejsca w pamięci gdzie znajduje się obiekt. Obiekty zatem mogą być zmieniane po przekazaniu do funkcji.

function wiekOsoby(wiek, obiektOsoba)

{

wiek = 26;

console.log(wiek); // Konsola zwraca 26 bo zmienna lokalna czyli kopia

obiektOsoba.imie = "Ola"; // Konsola zwraca Ola ponieważ obiet przekazywany

console.log(obiektOsoba); // poprzez referencje bez tworzenia kopii

}

let iloscLat = 24;

let kobieta = {imie: "Anna"};

wiekOsoby(iloscLat, kobieta);

console.log(iloscLat); // Konsola zwraca 24 ponieważ zmienna jest poza funkcją

console.log(kobieta); // Konsola zwraca Ola

* Funkcja zagnieżdżona:

function funkcjaPoziomu1(liczba)

{

function funkcjaPoziomu2(liczba2)

{

return liczba2 \* 3;

}

let wynik = funkcjaPoziomu2(liczba);

return wynik;

}

console.log(funkcjaPoziomu1(10));

* Funkcja i jej domyślne argumenty:

Domyślne argumenty funkcji dostępne są od momentu pojawienia się EcmaScript2015 ES6.

Przykład:

function createUser(name = "unknown", email = "unknown")

{

return { // Zwracany jest obiekt

name: name,

email: email

};

}

// Tworzony jest obiekt z jednym argumentem zamiast z dwoma.

let user1 = createUser("Ania");

// Brak drugiego argumentu powoduje wyświetlenie wartości domyślnej „unknown”.

console.log(user1);

* Funkcja strzałkowa:

Funkcja strzałkowa, czyli arrow function, to skrócony zapis funkcji dodany od pojawienia się wersji EcmaScript2015/ES6 JavaScript'u.

// Funkcja zwykła

function Sposob1(a,b)

{

return a \* b

};

console.log(Sposob1(2,2)); // Wynik działania 2x2=4

// Funkcja strzałkowa

let Sposob2 = (a,b) => a \* b;

console.log(Sposob2(2,2)); // Wynik działania 2x2=4

// Funkcja strzałkowa z normalnym ciałem funkcji

let Sposob3 = (a,b) => { return a \* b; }

console.log(Sposob3(2,2)); // Wynik działania 2x2=4

Nawiasy w funkcji strzałkowej należy pisać tak jak dla sposob3 ponieważ czasami są sytuacje, że kod nie będzie działał gdy nawiasy są rozstawione inaczej.

* Funkcja strzałkowa a obiekty:

Funkcja strzałkowa nie ma dostępu do obiektu arguments, ale można użyć dla niej operatora rest. Przykłady:

// Zastosowanie operatora rest w funkcji strzałkowej

let pokazArgumenty = (...parametry) => console.log(parametry);

// Wywołanie funkcji strzałkowej

pokazArgumenty(1,2,3,4);

// Sposób nr 1 zwrócenia obiektu z funkcji strzałkowej

let obiekt = (data) => {

return {

data: data

};

}

// Podgląd zwróconego obiektu z zawartością "Anna"

console.log(obiekt("Anna"));

// Sposób nr 2 zwrócenia obiektu z funkcji strzałkowej bez return

let obiekt2 = (data) => ({data: data});

// Podgląd zwróconego obiektu z zawartością "Ula"

console.log(obiekt("Ula"));

* Funkcja natychmiastowa IIFE:

IIFE czyli Immediately Invoked Function Expression to funkcja natychmiastowa, będąca również funkcją anonimową. Ogranicza dostęp zmiennych i funkcji tylko do niej, czyli nie zaśmieca globalnymi danymi naszego programu. Przykład:

// Rozpoczynamy tworzenie funkcji od nawiasu okrągłego

( function (data)

{

console.log(data);

let a = 10;

function test() { console.log(a); }

test();

} ) ("test"); // "Test" w nawiasie okrągłym jest przekazane jako argument data

Konsola zwraca "test" oraz "10".

**8. Konstruktory:**

* Tworzenie konstruktorów:

Obiekty oprócz literału obiektu mogą być tworzone poprzez konstruktor. Konstruktor jest specjalną funkcją pewnego szablonu na bazie którego powstaje obiekt. Stworzenie nowej instancji obiektu, czyli nowego egzemplarza będzie wymagało użycia słowa kluczowego new. Nazwy konstruktorów piszemy z dużej litery. Przykład:

let date = new Date();

console.log(date.getHours()); // Pokazana zostaje w konsoli godzina bez minut

// Tworzenie konstruktora z zapisem nazwy funkcji z dużej litery

function Car(brand, name, color)

{

this.brand = brand; // Wewnątrz konstruktora stosujemy this

this.name = name;

this.color = color;

this.year = 2000;

// Metoda wewnątrz konstruktora

// Zastosowano literał szablonu by przedstawić dane czytelnie z myślnikiem

this.printInfo = function () {console.log(`${this.name} - ${this.color}`);}

}

// Stworzenie nowego obiektu wywołując konstruktor

let car1 = new Car("Ford", "Mustang", "Red");

console.log(car1); // Wyświetlenie obiektu

car1.printInfo(); // Wywołanie metody

* Konstruktor bez new:

Wywołanie konstruktora bez słowa new sprawia, że this będzie obiektem window, czego należy unikać. Przykład:

function Car (brand, model) {

this.brand = brand;

this.model = model;

this.color = "Blue";

this.printInfo = function () {

console.log(this.brand, this.model, this.color);

}

}

// Wywołujemy konstruktor bez new

let car1 = Car("Ford", "Mustang");

console.log(car1);

// Tu widzimy, że elementy this są w obiekcie window

console.log(window.brand); // Konsola zwraca Ford

window.printInfo(); // Konsola zwraca Ford Mustang Blue

Aby zabezpieczyć się przed wywołaniem konstruktora bez new należy zastosować poniższy zabieg z instrukcją if.

function Vehicle (brand, model) {

if (this instanceof Vehicle == false) {

return new Vehicle(brand, model);

}

this.brand = brand;

this.model = model;

this.color = "Orange";

this.printInfo = function () {

console.log(this.brand, this.model, this.color);

}

}

// Wywołujemy konstruktor bez new

let vehicle1 = Vehicle("Fiat", "Tipo");

console.log(vehicle1); // Konsola prawidłowo pokazuje obiekt

// Wywołujemy konstruktor

let vehicle2 = new Vehicle("Fiat", "Panda");

console.log(vehicle2);

**9. Pozostałe:**

* Włączenie konsoli na przykładzie przeglądarki Google Chrome:

Będąc w oknie przeglądarki wchodzimy w narzędzia deweloperskie poprzez naciśnięcie przycisku F12. Następnie w menu wybieramy zakładkę „Console”.

* Wyłączenie alertów o błędach w konsoli, w ten sposób by otrzymywać informacje jedynie z naszych skryptów:

Będąc w narzędziach deweloperskich w menu wybieramy zakładkę „Console”, następnie „Console settings” i zaznaczamy checkbox „Selected context only”.

* Przykład wypisania w konsoli przykładowej wiadomości tekstowej operując w pliku html:

console.log("Hello World!")

* Przykład wywołania wiadomości typu warning w konsoli przeglądarki internetowej:

console.warn("Warning!")

* Wartość NaN:

Wartość czyli not a number to specjalna wartość informująca nas w konsoli, że coś nie jest liczbą. Uwaga! To nie jest typ danych! Przykładowo:

// Sprzeczność matematyczna jako wartość NaN

let data = 0/0;

console.log(data); /\* NaN \*/

data = Math.sqrt(-1);

console.log(data); /\* NaN \*/

// Sprawdzenie czy dany element jest NaN

console.log(isNaN(data)); /\* true \*/

console.log(isNaN(10)); /\* false \*/

* Dynamiczne typowanie JavaScript:

Dynamic typing, czyli dynamiczne typowanie oznacza, że w JavaScript do zmiennej możemy przypisać dowolną wartość np. string, numer, boolean itd. Typowane języki programowania takie jak C++ wymagają określenia danej zmiennej z góry, każdorazowo przed wprowadzeniem wartości zmiennej.

* Konwersje:

Koercja to automatyczna konwersja typu wartości w JavaScript gdy wymaga tego konkretna sytuacja. Przykłady konwersji:

// Konwersje jawne

let str = String(22); // Konwersja z liczby na łańcuch znaków

console.log(typeof str); // Konsola podaje string

let number = parseInt("123") // Konwersja ze string na liczbę

console.log(number); // Konsola zwraca 123

// Konwersje niejawne czyli koercje

let data = "test" + 10; // Następuje konwersja niejawna do string

console.log(data); // Konsola zwraca test10

let x = -"10"; // Następuje konwersja niejawna do liczby

console.log(x); // Konsola zwraca -10

console.log(typeof x); // Konsola zwraca number