MODUŁ 4:
Język JavaScript- wprowadzenie

Cel modułu

Moduł stanowi przygotowanie do rozpoczęcia nauki programowania aplikacji WWW w języku JavaScript po stronie serwerowej oraz po stronie klienckiej. Kontekst architektoniczno-protokołowy poznaliśmy w poprzednim module, natomiast w niniejszym zapoznamy się z najistotniejszymi z punktu widzenia naszych potrzeb cechami języka JavaScript. Ze względu na popularność tego języka i liczbę dostępnych na jego temat materiałów elektronicznych, zwłaszcza w języku angielskim, moduł ten będzie szczególnie dobrą okazją do przećwiczenia umiejętności samodzielnego poszukiwania informacji uzupełniających związanych z tematem.

Efekty kształcenia

Niniejsza lekcja służy osiągnięciu następujących efektów kształcenia.

Wiedza

Po ukończonym kursie student / studentka:

- Charakteryzuje system typów, model obiektowy oraz właściwości funkcyjne języka JavaScript,
- Omawia najbardziej charakterystyczne rozwiązania składniowe języka JavaScript,
- Wskazuje przykłady obiektów standardowych języka JavaScript oraz ich metod.

Umiejętności

Po ukończonym kursie student / studentka:

- Programuje obiektowo w języku JavaScript,
- Przetwarza dane typów prostych oraz obiekty stosując typ tablicowy i związane z nim metody,
- Wykorzystuje samodzielnie wyselekcjonowane materiały celem zapoznania się z nowymi językami i ramami programistycznymi.

Uzasadnienie

Język JavaScript zyskał w minionych latach szczególną popularność ze względu na rosnące wyrafinowane warstwy klienckiej aplikacji WWW, w których realizacji pełni rolę jako powszechna i praktycznie jedyna skutecznie ustandaryzowana platforma. W zastosowaniach tych zapewnia m.in. możliwość realizacji interakcji z użytkownikiem, obliczeń po stronie klienta, odpowiedniej modyfikacji struktury i wyglądu prezentowanej strony oraz realizowanie żądań kierowanych do serwera.

Drugim z motywów rosnącej popularności JavaScript jest dostępność technologii programowania aplikacji po stronie serwera. Również część technologii nierelacyjnych baz danych traktuje JavaScript jako domyślny język dla odpytywania i manipulacji przechowywanymi strukturami danych. Technologie te otwierają możliwość realizacji całej aplikacji WWW w jednym języku programowania oraz uniknięcia konwersji typów danych przy przekazywaniu danych z serwera do klienta i vice versa.

Oprogramowanie i podstawowe ustawienia

Środowisko programowania w języku JavaScript możemy sobie zapewnić na różne sposoby. Należą do nich m.in.:

- mechanizmy wykonawcze dostępne w przeglądarkach oraz interakcyjne i wsadowe interfejsy JavaScript dla uruchamiania wprowadzonego kodu, dostępne powszechnie w grupie narzędzi programistycznych przeglądarki[1];
- wsadowe i interakcyjne przetwarzanie kodu w środowisku Node.js;
- narzędzia uruchomieniowe JavaScript dostępne w zintegrowanych środowiskach programistycznych[2];
- liczne strony internetowe udostępniające edycję i uruchamianie kodu.

Pochodzenie i rozwój języka

Historia języka sięga początków systemu WWW i towarzyszącej im walki o prymat pomiędzy producentami przeglądarek. Ówczesna wizja aplikacji WWW odbiegała od obecnego ich kształtu. Również zakładana rola języka skryptowego po stronie przeglądarki ewoluowała. W początkowych założeniach przyjmowano, że język ten powinien, podobnie jak sam HTML, zapewniać łatwość nauki i użycia czyniącą go dostępnym dla nieprofesjonalistów. Początkowe zastosowania, zmierzające do uatrakcyjnienia wyglądu i interakcyjności stron, nie zapowiadały, że ostatecznie technologia ta stanie się krytycznym elementem funkcjonowania całego WWW niezbędnym do zrealizowania niektórych rodzajów aplikacji.

Początkowo znacznie większe nadzieje na stworzenie powszechnie wspieranej, silnie ustandaryzowanej technologii strony klienta wiązano z apletami języka Java. Pojawienie się słowa Java również w nazwie interesującego nas języka skryptowego przypisuje się zwykle względom marketingowym (wykorzystanie popularności szybko zdobywającego rynek języka Java). Jednakże można też wskazać na inne powiązanie. Przed ustandaryzowaniem interfejsu DOM (zob. dalej), aplety języka Java nie miały bezpośredniego dostępu do struktury dokumentu HTML, toteż JavaScript mógł pełnić tutaj rolę pośrednika.

Mówiąc o nazewnictwie, należy wspomnieć jeszcze następujące nazwy: *LiveScript* (stosowana przed wprowadzeniem nazwy JavaScript), *JScript* – dialekt języka rozwijany przez firmę Microsoft, *ECMAScript* – oficjalny standard języka zatwierdzony przez ECMA w 1997 roku (później również jako standard ISO). W momencie tworzenia tego tekstu najświeższą jest 9th Edition = ECMAScript 2018.

Dla zapewnienia zgodności kodu wykorzystującego nowe konstrukcje JS z mniej aktualnymi platformami – są wykorzystywane mechanizmy translacji – tzw. *transpilers* zob. https://babeljs.io/ .

Dla zwięzłości, odwołując się w dalszej części tekstu do języka JavaScript, będziemy posługiwać się akronimem JS.

Obserwując rozwój JavaScript można wskazać na następujące trendy:

- Stopniowe redukowanie różnic we wsparciu języka i jego środowiska przez poszczególne przeglądarki,
- Wzrost wyrafinowania i złożoności oprogramowania działającego po stronie klienta opartego na JS,
- Stymulowane przez powyższe – powstawanie narzędzi deweloperskich, bibliotek, wzorców projektowych i ram programistycznych,

[1] Zob. np. Developer Tools w przeglądarce Chrome i opcja Sources / Snippets dostępna tamże
[2] Możemy stosować m.in. dostępny na warunkach licencji akademickiej.JetBrains WebStorm

- Doskonalenie technologii wykonania kodu, owocujące znacznym przyspieszeniem działania oprogramowania pisanego w JS,
- Wzbogacanie interfejsów programistycznych wspieranych po stronie przeglądarki – powodujące zmniejszanie zapotrzebowania na inne technologie klienckie specyficzne dla danego producenta.

Kontrola bieżąca	[ ]	*komentarz odpowiedzi*
Zaznacz poprawne odpowiedzi na temat JavaScript:		
Ma podobieństwa składniowe z Java	x	Tak | Odpowiedź niepełna
Wymaga maszyny wirtualnej języka Java		Nie.
Był przewidziany do współdziałania z apletami Javy	x	Tak | Odpowiedź niepełna
Projektowano go z myślą o użyciu go przez programistów Javy		Nie. Historie tych języków nie są tak mocno powiązane ze sobą. JS był ponadto pomyślany jako prosty język dla szerszego kręgu użytkowników.

Charakterystyka języka

Próbując dopatrzyć się dziś uzasadnienia dla słowa Java w nazwie tego języka, możemy wskazać chyba tylko na pewne podobieństwa składniowe. Istotnie bowiem, składnię szeregu instrukcji (w tym iteracyjnych i warunkowych, składnię komentarzy i szereg innych) zapożyczono z Java (czy ogólniej – z języków rodziny C). Jedną z odziedziczonych w ten sposób cech języka jest czułość jego składni na wielkość znaków (*case sensitivity*). Podobnie znajomy dla osób posługujących się tymi językami okaże się sposób, w jaki JS przetwarza tablice.

JS uchodzi za język o luźnej kontroli typów. Wyróżnia następujące typy danych: `number`, `string`, `boolean`, `function`, `object`, `undefined`. Tablica jest specjalnym rodzajem typu `object`. Możliwość sprawdzenia typu zapewnia funkcja: `string typeof(`*`zmienna_lub_stała`*`)`

Istnieją metody zapewniające możliwość konwersji pomiędzy tymi typami. Ponadto, podobnie jak w Java – konkatenacja wartości innego typu danych (np. liczbowego) z łańcuchem tekstowym nie wymaga uprzedniej jawnej konwersji typu. Innym przejawem takiej elastyczności jest interpretowanie różnych typów danych jako wartości logicznej (m.in. liczba różna od zera daje *true*, wartość *null* daje *false* itp.).

Warto zwrócić uwagę na typ `function` i związaną z nim możliwość przekazywania funkcji jako parametrów.

JS jest ponadto zaliczany do języków dynamicznych. Oznacza to m.in., że mamy możliwość swobodnego zmieniania, w tym uzupełniania, zestawu właściwości danego obiektu – np.

```
var x = new Object();

x.tekst = "Ala ma kota";

x.liczba = 100.0;

x.wartoscLogiczna = true;
```

Istotną różnicą, którą zauważy programista przywykły do Javy lub podobnego jej języka jest brak jawnych wskazań typów w deklaracjach zmiennych czy parametrów. Z kolei deklaracje zmiennych wymagają poprzedzenia słowem kluczowym `var` .

Powyższe zastrzeżenie jest istotne, gdyż pominięcie tego słowa kluczowego sprawi, że zamiast lokalnej zmiennej utworzymy lub zaktualizujemy atrybut o wskazanej nazwie w środowisku globalnym, co może zakłócić funkcjonowanie aplikacji.

Kontrola bieżąca	[ ]	*komentarz odpowiedzi*
Deklaracje zmiennych w JS różni od Javy:		
obecność słowa kluczowego `var`	x	Tak | Odpowiedź niepełna
konieczność rozpoczynania nazwy zmiennej od symbolu `$`		Nie. Symbol $ jest dopuszczalny, ale nie ma tu takiego wymogu.
brak wskazania nazwy typu	x	Tak | Odpowiedź niepełna

Cechy charakterystyczne i ważne pojęcia

Hoisting

Deklaracje zmiennych i funkcji są wirtualnie przesuwane na początek kodu, toteż wyrażenia odwołujące się do nich mogą zostać podane przed tymi deklaracjami. Pamiętajmy jednak, że dotyczy to deklaracji zmiennej a nie inicjalizacji. Zapoznaj się w tym kontekście z tym z różnicą pomiędzy wyrażeniem funkcyjnym przypisanym do zmiennej a deklaracją funkcji.

Deklaracje zmiennych i stałych

Dostępne są następujące słowa kluczowe:

- var – zmienna o zasięgu danej funkcji
  - Hoisting z wartością undefined
- let – zmienna o zasięgu danego bloku
  - Brak inicjowania wartością undefined
  - Nie pozwala na redeklarację
- const – stała o zasięgu danego bloku
  - Brak inicjowania wartością undefined
  - Nie pozwala na zmianę przypisania
  - Pozwala jednak modyfikować zawartość – np. dodanie property albo .push() dla tablic

Rozważ przykład dwóch instrukcji iteracyjnych i różnic w ich działaniu. Czy potrafisz wyjaśnić przyczynę różnic?

```
for(var i=0; i <5; i++){
    setTimeout(function(){
        console.log(i);
    }, 10);
}
for(let i=0; i <5; i++){
    setTimeout(function(){
        console.log(i);
    }, 10);
}
```

Inicjowanie i dostęp do właściwości obiektu

Utworzenie obiektu nie wymaga operatora new. Można użyć składni tzw. object initializer i tym sposobem skonstruować parametr wywołania funkcji albo zainicjować zmienną:

```
let samochod = {
      kolor:"czerwony",
      kola:4,
      silnik:{
            cylindry:4,
            poj:2.0
      }
};
```

Do tak utworzonych obiektów możemy się odwoływać za pomocą tradycyjnych wyrażeń ścieżkowych:

```
samochod.silnik.poj
```

albo w konwencji tablic asocjacyjnych, gdzie nazwy właściwości pełnią role indeksów:

```
samochod["silnik"]["poj"]
```

Operator new a funkcje

Funkcja w JS jest szczególnego rodzaju obiektem (i posiada właściwości; jedną z nich jest `prototype` – zob. następna sekcja).

Gdy wywołujemy funkcję w ramach operatora **`new`** :

- jest tworzony nowy obiekt
- jest on kojarzony w własnością `this` przed wykonaniem ciała funkcji
- przyłączany jest też obiekt prototypu – zob. dalej
- po wykonaniu zwracana jest (implicite) wartość `this`

Koercje

Zmieniany jest typ wyrażenia. Mogą być wykonywane explicite:

np. `Number('44')`

lub implicite – np.

```
1==null
10+''
if(2){...}
1 || 'abc'
```

Operatory logiczne jak || czy && konwertują wewnętrznie na potrzebę ewaluacji na typ boolean, jednak zwracają ostatecznie wartość w postaci oryginalnej.

Wartości specjalne

W ramach występujących w JS typów: object, undefined, number, boolean, string, wyróżniono następujące predefiniowane wartości specjalne:

`null` (typ `object`)

`undefined` (typ `undefined`)

- Badając rezultaty wołanych wyrażeń zwróćmy uwagę na rozróżnienie pomiędzy `undefined` od `ReferenceError` !

`Infinity` (typ `number`)

większa niż dowolna wartość (poza NaN)

- użyteczne np. przy porównaniach, wartościach domyślnych itp.

`-Infinity` (typ `number`)

`NaN` (typ `number`)

Wartości *falsy* i *truthy*

Ponieważ JS posiada dość luźny system typów, ewaluacja wartości logicznych nie musi opierać się na wartościach typu boolean. Wprowadzono następujące reguły ewaluacji.

A) Wartości *falsy* – ewaluowane jako false:
- `false`
- `undefined` // tzw. nonvalue
- `null` // tzw. nonvalue
- `0`
- `NaN`
- `""`

B) Wartości *truthy* – ewaluowane jako true: wszystkie pozostałe

Tworzenie i przekazywanie funkcji

Mogą być powoływane poprzez nazwane (jak niżej) lub nienazwane wyrażenie funkcyjne.

Mogą być przypisywane do zmiennych, przekazywane jako parametry, zwracane jako rezultaty funkcji – np.

```
function changeCase(val) {
  return val.toUpperCase();
}
function demofunc(a, passfunction) {
  console.log(passfunction(a));
}
demofunc("smallcase", changeCase);
```

Obiektowość oparta na prototypach

Chociaż sięganie do właściwości obiektu, a nawet – jego tworzenie instrukcją `new` – wyglądają bardzo podobnie jak w języku Java, to model obiektowy tego języka jest znacząco różny. JS nie stosuje pojęcia klasy – zamiast tego niezmienniki pewnej grupy obiektów mogą być grupowane za pomocą tzw. obiektów-prototypów. Mówimy zatem, że obiektowość JS jest oparta nie na klasach, ale na prototypach.

Warto podkreślić, że nie występuje tu też dedykowana konstrukcja zwana konstruktorem. Rolę konstruktora może pełnić dowolna funkcja, która, jeśli zostanie wywołana w ramach instrukcji `new`, uruchomiona będzie na rzecz tego nowo utworzonego obiektu. Tym samym zgodnie z naszymi oczekiwaniami zadziałają umieszczone w takiej funkcji instrukcje przypisania w postaci

`this.`*`nazwaWłaściowości = wartość`*`;`

Pamiętajmy, że w analogiczny sposób możemy przypisywać nie tylko wartości atrybutów, ale też funkcje, wyposażając tym samym obiekt w metody. To jednak samo w sobie nie daje nam jeszcze dogodnej możliwości gromadzenia niezmienników (wspólnych dla grupy obiektów wartości oraz metod) podobnie, tak, jak ma to miejsce w językach opartych na klasach.

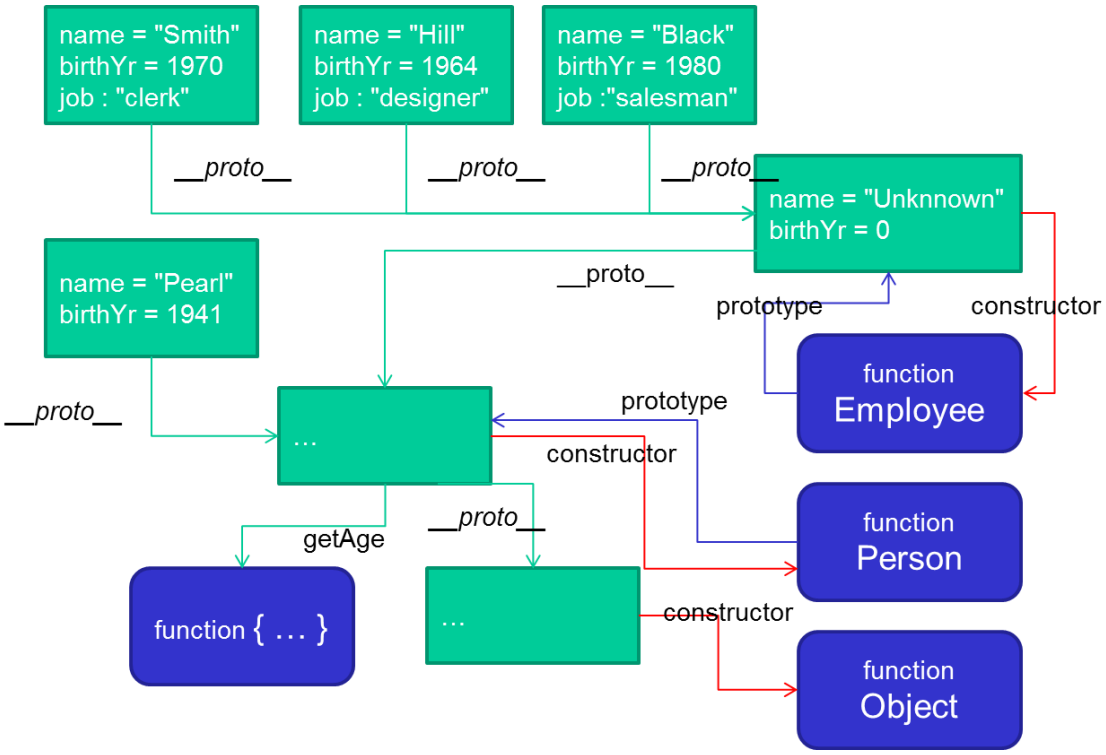
Celowi temu służy pojęcie prototypu. Każdy obiekt jest powiązany z innym obiektem, stanowiącym jego prototyp. Jeśli szukanej właściwości nie ma w danym obiekcie, będą przeszukane kolejne obiekty w łańcuchu jego prototypów. Zwykle nie ma potrzeby jawnej nawigacji w łańcuchu prototypów, choć dla istniejącego obiektu jego prototyp jest dostępny we właściwości o nazwie `__proto__`.

Z powyższą właściwością __proto__ nie należy mylić właściwości `prototype`. Ta druga może występować jako właściwość funkcji (przewidzianej do pełnienia roli konstruktora). Właściwość `prototype` określa mianowicie, który obiekt należy podłączyć jako prototyp do nowych obiektów tworzonych z pomocą tego konstruktora. Jak zobaczymy w przykładach, modyfikacja (np. uzupełnienie) zestawu właściwości obiektu wskazywanego właściwością `prototype` może nam zapewnić odpowiednią zmianę cech całego zestawu (także już istniejących) powiązanych z nim obiektów.

Przeciwległa do `prototype` jest właściwość `constructor` (tzn. łączy ona parę obiektów w przeciwnym kierunku).

Ponieważ funkcja pełniąca rolę konstruktora determinuje właściwości obiektu, to jej nazwa niesie informację o podobnym charakterze, jak nazwy klas w językach z klasami. Dlatego też znany nam z Javy operator `instanceof` działa tutaj następująco. Sprawdza, czy funkcja o podanej nazwie jest przypisana jako właściwość `constructor` dla któregokolwiek obiektu w łańcuchu prototypów testowanego obiektu.

Poniższy rysunek ilustruje opisane wyżej zależności. Zwróćmy uwagę na pewną sztuczność: obiekt-prototyp zawierający tylko dane właściwe obiektowi Person (tj. nazwisko i rok urodzenia) został tutaj przez nas manualnie przypisany jako posiadający konstruktor `Employee` (zastosowano: `Employee.prototype.constructor = Employee;`). Celem było tutaj zaimprowizowanie efektu analogicznego jak w hierarchii klas (która jest dla JS pojęciem nieco obcym) tak, by `instanceof` działało zgodnie z naszymi oczekiwaniami. W praktyce jednak rzadko będziemy potrzebować tego typu zgodności, toteż podobne nie do końca intuicyjne zabiegi nie będą dla nas konieczne.



Kontrola bieżąca	( )	*komentarz odpowiedzi*
Dla istniejącego obiektu JS możemy się dostać to jego obiektu-prototypu za pośrednictwem właściwości:		
protytype		Nie. Tę właściwość posiada funkcja pełniąca rolę konstruktora.
__proto__	x	Tak.
this		Nie.
__this__		Nie.
super		Nie.
parent		Nie.

Klasy i dziedziczenie

W ostatnich wersjach języka, wykorzystując (w większości zarezerwowane już od lat) słowa kluczowe, zbudowano rozwiązania składniowe dla prostszego tworzenia hierarchii klas na podobieństwo popularnych języków obiektowych z klasami. W istocie jest to jedynie tzw. „lukier syntaktyczny”, gdyż faktyczna implementacja opiera się na mechanizmie prototypów. Niezbędne słowa kluczowe to:

- `class`
- `extends`
- `constructor`
- `get`
- `set`
- `super`
- `static`

Zastosowanie ich jest w dużej mierze intuicyjne dla osób znających popularne języki obiektowe. Zwróćmy jedynie uwagę, że w przeciwieństwie do funkcji deklaracji klas nie obejmuje tzw. *hoisting* stąd musimy zadbać o odpowiednią kolejność kodu wykorzystującego te deklaracje.

Samodzielne zapoznanie się ze szczegółami tego rozwiązania składniowego jest przedmiotem jednej z części zadania do tego modułu.

Domknięcia (*closures*)

Nie będziemy tutaj kompleksowo omawiać pojęcia domknięć, warto je jednak bodaj zasygnalizować, bo jest jedną z ważniejszych cech JS, związaną z obsługą przez ten język typu `function`. Istotę domknięć można streścić następująco. Funkcja w trakcie jej interpretacji ma dostęp do swego zewnętrznego środowiska i może korzystać ze zdefiniowanych tam zmiennych. Środowisko to następnie może przestać istnieć, bo zakończyło się przetwarzanie odpowiedniego bloku kodu. Jednakże, jeśli tę funkcję przekazujemy gdzieś dalej jako parametr, to aby zapewnić jej pomyślne wykonanie po tym przekazaniu, jej oryginalne otoczenie jest zachowywane i staje się dostępne dla tej funkcji na okoliczność jej wykonania.

Nie rozwijając tu szerzej tego tematu, ograniczmy się do zilustrowania, jak ów mechanizm mógłby posłużyć do zapewnienia w JS hermetyzacji. Załóżmy, że chcemy „opakować" właściwość o nazwie `prywatne` w getter i setter. Poniższy przykład ilustruje i komentuje takie rozwiązanie.

```
interfejs = (function(){
    var prywatne = "wartosc chroniona";
    /* ta zmienna normalnie przestałaby istnieć wraz z zakończeniem wykonania tego bloku (zauważmy, ze funkcja jest tu natychmiastowo wykonywana za sprawa umieszczenia () poniżej) */

    var getW = function() {
        return prywatne;
    }
    var setW = function(nowa) {
        prywatne = nowa;
    }
    return { getWartosc : getW, setWartosc : setW };
    /* jednak, dzięki mechanizmowi domknięć, środowisko używane przez funkcje getW i setW - a wiec zmienna prywatne, jest zachowywane, aby umożliwić przyszłe wywołania tych funkcji po ich zwróceniu - return do zewnętrznego środowiska */
})();
alert(interfejs.getWartosc());
```

Zwróćmy uwagę na składnię instrukcji. Zmiennej interfejs przypisana jest nie sama zdefiniowana tu funkcja, ale (zob. puste nawiasy na zakończeniu instrukcji) – rezultat jej natychmiastowego wywołania.

Kontrola bieżąca	( )	*komentarz odpowiedzi*
Pojęcie domknięcia (closure) w języku JavaScript wiąże się z		
ograniczeniem środowiska działania skryptu do bieżącego okna przeglądarki		Odpowiedź błędna.

stosowaniem funkcji jako pełnoprawnego typu stosowanego do zmiennych, atrybutów oraz parametrów	x	Tak.
domyślnym traktowaniem atrybutów obiektu jako prywatnych		Odpowiedź błędna.
niemożnością dodawania nowych atrybutów po utworzeniu obiektu instrukcją `new`		Odpowiedź błędna.

Dalsze informacje

Nie sposób wyczerpać tematu języka JS w pojedynczym module kurs. Z uwagi na obecną ważność i powszechność tego języka, zachęcamy do poszerzenia swojej wiedzy o zagadnieniu. Pośród dostępnych nieodpłatnie źródeł można wskazać m.in.:

- Wprowadzenie do JS w języku polskim: http://ferrante.pl/category/frontend/vademecum/page/5/
- Omówienie DOM HTML na portalu W3Schools: http://www.w3schools.com/js/js_htmldom.asp
- Kompleksowy przewodnik po cechach języka (w języku angielskim). Doraźnie interesujące dla nas byłyby w kontekście tego bloku treści zawarte w sekcjach 1..9: https://javascript.info/js
- Bardziej systematyczne omówienie semantyki JS: JavaScript Succintly - http://www.syncfusion.com/resources/techportal/ebooks

Podsumowanie

Niniejszy moduł stanowił, siłą rzeczy, bardzo wybiórcze wprowadzenie do języka JavaScript. Zapoznanie się z jego składnią oraz najważniejszymi cechami jego modelu obiektowego i funkcjonalnością predefiniowanych typów, pozwoli nam w kolejnej części kursu przejść do omówienia sposobu wykorzystania tego języka w programowaniu funkcjonalności działającej po stronie przeglądarki oraz tworzeniu funkcjonalności serwerowej w środowisku Node.js.

Zadanie

Część 1:

Sporządź plik funkcje.js i zaimplementuj w nim następujące funkcje:

1. Wyliczanie n-tej (parametr funkcji) liczby z szeregu Fibonacciego
2. Sprawdzanie, czy podany jako argument łańcuch tekstu jest palindromem
3. Określanie nazwy typu dla przekazanego do funkcji argumentu
4. Konwertowanie wartości całkowitej na kolekcję monet o zadanych (jako tablica) dostępnych nominałach – np. wywołanie tej funkcji mogłoby wyglądać: amountToCoins(46, [25, 10, 5, 2, 1])

Część 2:

Sporządź plik klasy.js i zaimplementuj w nim składającą się z co najmniej dwóch klas hierarchię generalizacji-specjalizacji o tematyce zbieżnej z wybraną przez Ciebie dziedziną problemową projektu semestralnego. W powstałym kodzie uwzględnij:

- funkcję pełniącą rolę konstruktora
- adekwatne do semantyki poszczególnych algorytmów udostępnienie getterów i setterów.