Zaawansowane Techniki WWW

(HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 6

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

Przypomnienie z poprzedniego wykładu

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

RWD, Mobile-First, Mediaqueries

Wprowadzenie do języka JavaScript

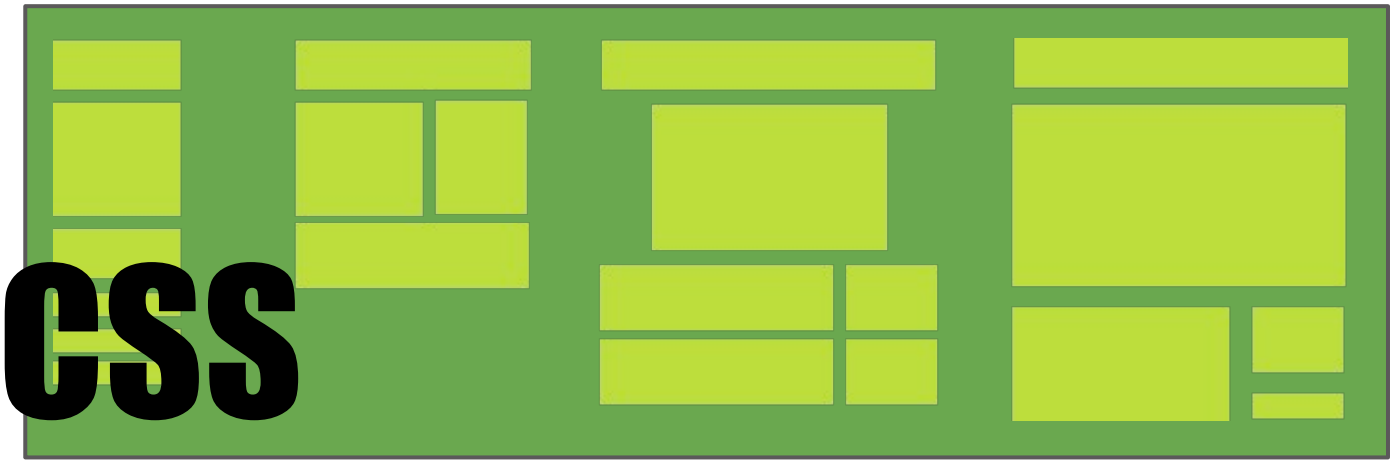
DOM - Document Object Model

Obiekty w JavaScript

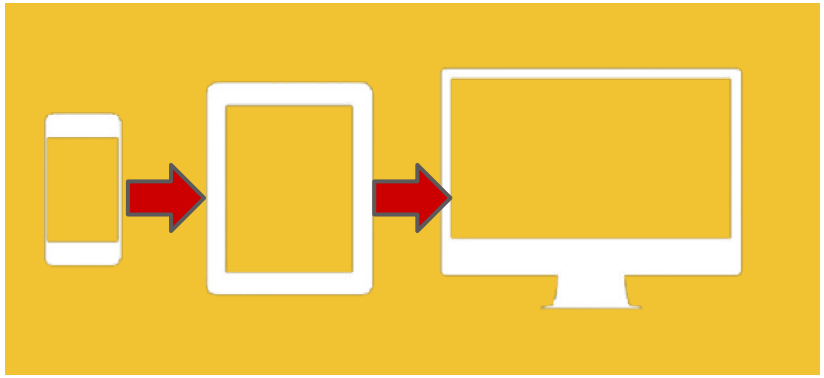
Jak stosować Mediaqueries?

ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wygądu wiele “layautów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.

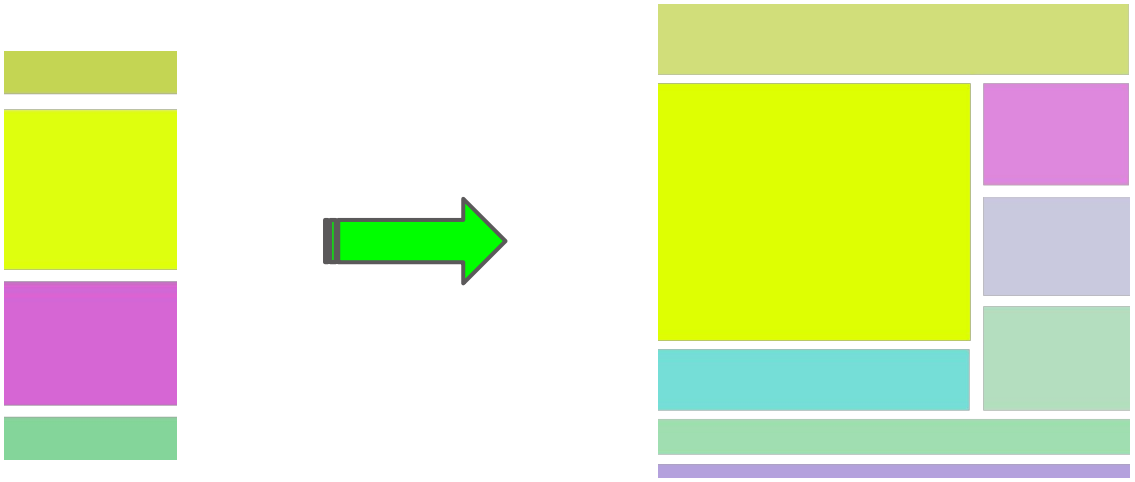


Które CSSy tworzyć najpierw ?



Filozofia Mobile-First

Zgodnie z filozofią MobileFirst tworzenie strony powinno być rozpoczęte od najmniejszych ekranów czyli dla urządzeń mobilnych.



Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (embedded-script),
- umieścić w osobnym pliku (linked-script).

HTML5

Embedded-Script

```
<body>
 <script>
   alert("Hello World!");
 </script>
</body>
```

Linked-Script

```
<head>
  <script src="skrypt.js">
  </script>
</head>
```

Plik .html

```
alert("Hello World!");
```

Plik .js

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste		Obiekty
Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są niezmienne.	VS.	Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie modyfikować.

W języku JavaScript:

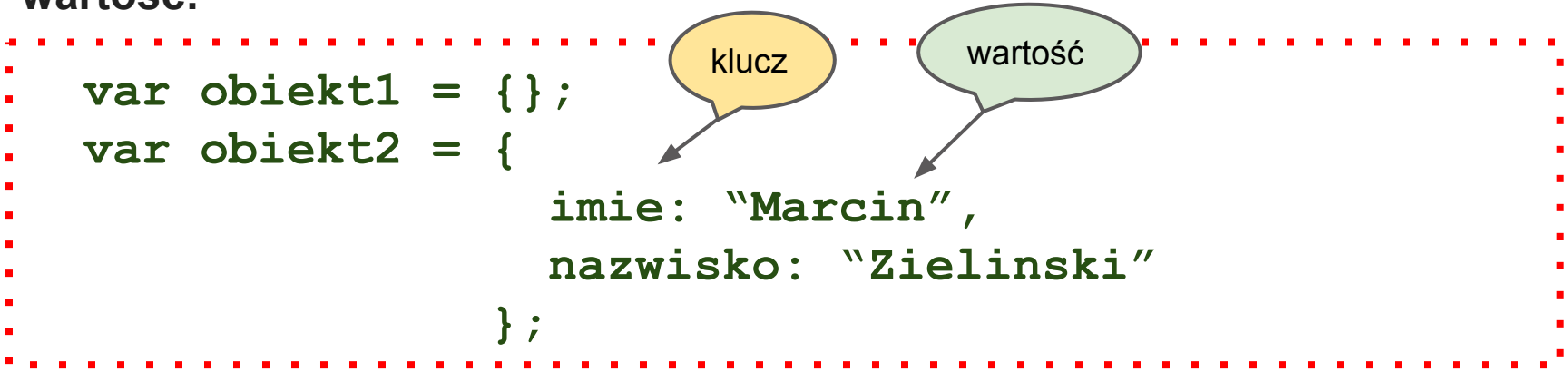
- tablice są obiektami,
- funkcje są obiektami,
- wyrażenia regularne są obiektami,
- obiekty są obiektami.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Tworzenie obiektów

Jedną z podstawowych umiejętności w JavaScript jest tworzenie własnych obiektów reprezentujących logicznie powiązane kolekcje danych.

Obiekt a w zasadzie "literał obiektowy" jest zapisywany za pomocą nawiasów klamrowych który zawiera zero lub więcej par klucz (nazwa)-wartość:



Uwaga:
Nazwa własności może być dowolnym słowem, jednak kiedy jest to np. słowo zastrzeżone dla składni języka JavaScript powinno być pisane w cudzysłowie.

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {
            carrier: "LOT",
            number: 3913,
            equipment = "388",
            origin: "WAW",
            destination: "KRK",
            schedule:{
                    departure_time: "22:45",
                    arrival_time: "23:35"
                    },
            showFlightNumber: function(){
                    alert(this.number);
            }
        };

    // wywołanie funkcji będącej częścią obiektu
    flight.showFlightNumber();
```

Stworzyliśmy wewnątrz obiektu funkcję nienazwaną!

Odwołanie do obiektu z wnętrza samego siebie !

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JavaScript i aplikacje WWW

Jak już wspomnielismy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JavaScript i aplikacje WWW

Jak już wspomnielismy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.

Ale co to w zasadzie znaczy ??

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JavaScript i aplikacje WWW

Jak już wspomnielismy język JavaScript powstał niejako z potrzeby podniesienia atrakcyjności stron www oraz nadania im odpowiedniej dynamiki oraz nowych funkcjonalności. Jedną z jego naważniejszych cech jest to że jest całkowicie obiektowy i operuje na zdarzeniach.

Ale co to w zasadzie znaczy ??

Obiektami są wszystkie elementy strony jak i sama strona w całości. Jest to konsekwencją zastosowanie modelu DOM. Te obiekty można modyfikować za pomoca metod.

Zdarzenie to określona czynność, która została wykonana przez użytkownika strony WWW podczas odwiedzania danej strony. Przykładowe zdarzenia to m.in. przesunięcie wskaźnika myszki nad obrazkiem, zaznaczenie zawartości obiektu czy wysłanie zawartości formularza.

W języku JavaScript zdarzenia są obsługiwane za pomocą specjalnych poleceń, zwanych funkcjami obsługi zdarzeń. Oznacza to, że określona czynność, wykonana przez użytkownika strony WWW, wywoła przypisaną do tej czynności (do tego zdarzenia) funkcję obsługi zdarzenia.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

Zdarzeniem jest np. naciśnięcia klawisza myszy. Funkcja obsługi zdarzenia odpowiedzialna dla takiego przypadku to “onClick()”, która spowoduje wykonanie określonej czynności jaką jej przypisano w sytuacji zajścia danego zdarzenia:

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Obiektem jest cały dokument hipertekstowy html. Obiektem który go reprezentuje jest “document”. Obiekt ten posiada własności oraz kilka wbudowanych metod które pozwalają na modyfikację zawartości strony lub uzyskanie informacji. Przykładowo gdy chcemy dodać

```
// właściwość obiektu document
document.lastModified;

// metoda dla obiektu document
document.write("Modyfikujemy zawartość dokumentu");
```

Zdarzeniem jest np. naciśnięcia klawisza myszy. Funkcja obsługi zdarzenia odpowiedzialna dla takiego przypadku to “onClick()”, która spowoduje wykonanie określonej czynności jaką jej przypisano w sytuacji zajścia danego zdarzenia:

```
document.getElementById(“button”).onclick = function() {
        alert(“Nacisnąłeś przycisk”);
    }
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Pozostałe typy zdarzeń które mogą zostać obsłużone:

onabort	Zaniechano ładowania strony
onblur	Obiekt przestał być aktywny (wybrany)
onchange	Obiekt zmienił swój stan
onclick	Kliknięto obiekt
onerror	Błąd w skrypcie
onfocus	Obiekt został uaktywniony (wybrany)
onload	Obiekt (strona) został załadowany (zakonczono jego ładowanie)
onmouseover	Obiekt został wskazany myszką

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Pozostałe typy zdarzeń które mogą zostać obsłużone:

onmouseout	Obiekt przestał być wskazywany myszką
onselect	Zawartość obiektu została zaznaczona
onsubmit	Zawartość formularza została przesłana do serwera
onunload	Zmieniono wyświetlaną stronę

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Przykład:

```
<script>
var name="NN"
function hello()
{
name=prompt("Podaj swoje imię:",name);
alert("Witaj "+name+" na naszej stronie!");
}
function goodbye()
{
alert("Do widzenia "+name+"!");
}
</script>

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

<body onload="hello();" onunload="goodbye();">
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Przykład:

```
<script>
var name="NN"
function hello()
{
name=prompt("Podaj swoje imię:",name);
alert("Witaj "+name+" na naszej stronie!");
}
function goodbye()
{
alert("Do widzenia "+name+"!");
}
</script>

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

<body onload="hello();" onunload="goodbye();">
```

Procedura obsługi *onload* gwarantuje, iż cała zawartość strony zostanie wyświetlona przed wykonaniem tej procedury. Podobnie, procedura obsługi *onunload* pozwala na wykonanie skryptu przed załadowaniem nowej strony.

Obiekty i zdarzenia na stronie

Przykład:

```
function listaURL(a,b,c,d,e)
{
this[0]=a;
this[1]=b;
this[2]=c;
this[3]=d;
this[4]=e;
}

function wyborStrony(lista)
{
var dzis = new Date();
var numer = dzis.getSeconds() % 5;
window.open(lista[numer],"oknoLosowe")
}

listaAdresow = new listaURL("http://www.onet.pl",
"http://www.wp.pl.pl",
"http://www.interia..pl",
"http://www.uj.edu.pl",
"http://www.gazeta.pl");

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

<body onload="wyborStrony(listaAdresow);">
```

Obiekty i zdarzenia na stronie

Przykład:

```
function listaURL(a,b,c,d,e)
{
this[0]=a;
this[1]=b;
this[2]=c;
this[3]=d;
this[4]=e;
}

function wyborStrony(lista)
{
var dzis = new Date();
var numer = dzis.getSeconds() % 5;
window.open(lista[numer],"oknoLosowe")
}

listaAdresow = new listaURL("http://www.onet.pl",
"http://www.wp.pl.pl",
"http://www.interia..pl",
"http://www.uj.edu.pl",
"http://www.gazeta.pl");

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

<body onload="wyborStrony(listaAdresow);">
```

W kodzie JavaScriptu wszystkie odwołania do procedur obsługi zdarzeń należy zapisywać za pomocą małych liter, np. *onclik*. Wewnątrz zaś etykiet języka HTML można zamiennie używać małych i dużych liter, pisząc np. *onClick.*

- ***ale nie ma to żadnego znaczenia !!!***

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

Emulowanie zdarzeń może się okazać przydatne, gdy np. trzeba wysłać formularz bez konieczności proszenia użytkownika o kliknięcie na przycisku Submit lub gdy trzeba np. zmienić miejsce wprowadzania informacji w zależności od czynności podejmowanych przez użytkownika.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Emulowanie zdarzeń:

blur()	Metoda JavaScriptu.	Usuwa miejsce wprowadzania z aktywnego pola.
click()	Metoda JavaScriptu.	Emuluje kliknięcie myszą na obiekcie.
focus()	Metoda JavaScriptu.	Emuluje uaktywnienie danego obiektu.
reset()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Reset przez użytkownika.
submit()	Metoda JavaScriptu.	Emuluje kliknięcie na przycisku Submit przez użytkownika.
select()	Metoda JavaScriptu.	Powoduje wybranie określonego pola formularza.

Emulowanie zdarzeń może się okazać przydatne, gdy np. trzeba wysłać formularz bez konieczności proszenia użytkownika o kliknięcie na przycisku Submit lub gdy trzeba np. zmienić miejsce wprowadzania informacji w zależności od czynności podejmowanych przez użytkownika.

Przykład:
W przypadku formularza który chcemy zwalidować przed wysłaniem, możemy wykonać odpowiedni skrypt sprawdzający po naciśnięciu przycisku wyślij, a dopiero gdy wynik sprawdzenia jest pozytywny skrypt za pomocą zdarzenia ***submit()*** przesyła dane na serwer.

Obiekty i zdarzenia na stronie

Przykład emulowania zdarzenia “click”:

```
<form>
  <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
alert('click event occured')">
</form>

<script>
function myFunction() {
    document.getElementById("myCheck").click();
}
</script>
```

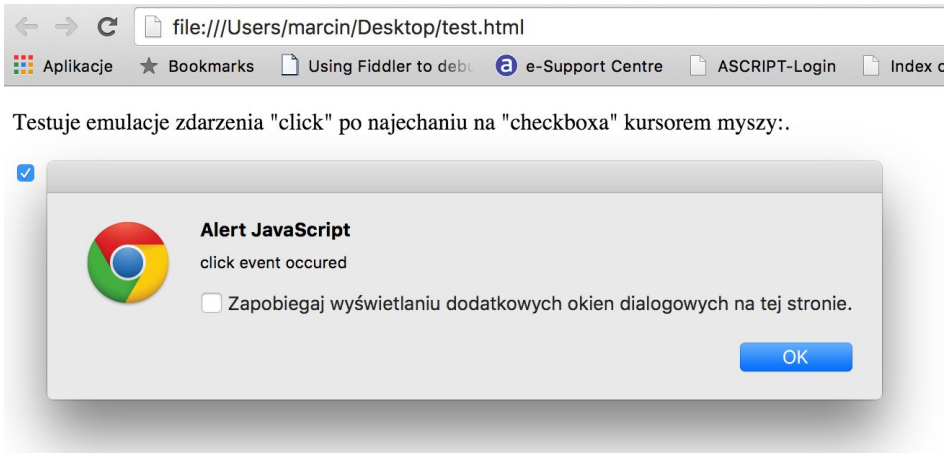
UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Obiekty i zdarzenia na stronie

Przykład emulowania zdarzenia “click”:

```
<form>
  <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
alert('click event occured')">
</form>

<script>
function myFunction() {
    document.getElementById("myCheck").click();
}
</script>
```



UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JSON

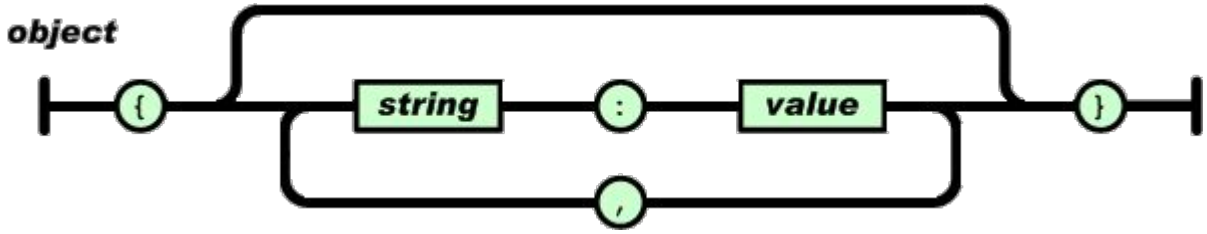
JSON (JavaScript Object Notation) ***[Notacja Obiektowa JavaScriptu]*** **- jest to “lekki” format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.**

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JSON

JSON (JavaScript Object Notation) ***[Notacja Obiektowa JavaScriptu]*** **- jest to "lekki" format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.**

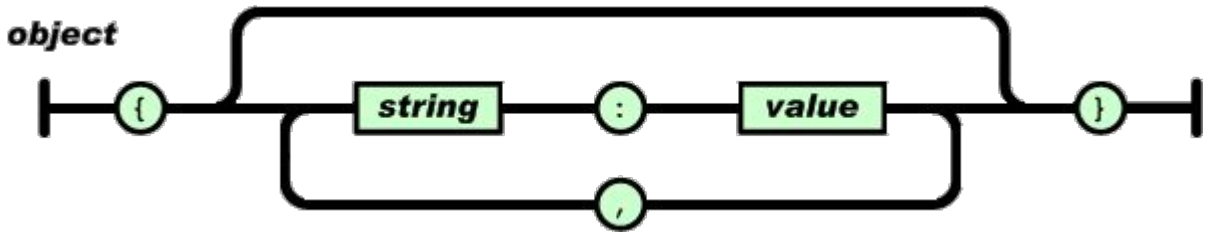
http://www.json.org/



JSON

JSON (JavaScript Object Notation) ***[Notacja Obiektowa JavaScriptu]*** **- jest to "lekki" format do przenoszenia danych oparty o literały obiektowe JavaScriptu. Jest podzbiorem JS, ale kompletnie niezależnym i może być używany do wymiany danych w zasadzie w każdym współczesnym języku programowania.**

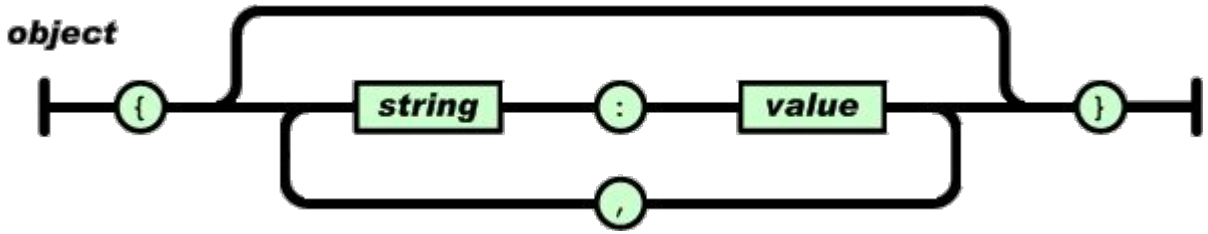
http://www.json.org/




```
{
    "firstname": "Jan",
    "lastname": "Kowalski",
    "age": 20
}
```

Obiekt w formacie JSON jest nieuporządkowanym zbiorem klucz-wartość, gdzie klucz może być dowolnym łańcuchem, natomiast wartość jednym z dowolonych typów (integer, string etc) włączając w to tablice i inne obiekty.

JSON



JSON posiada sześć rodzajów wartości (value):

1. **obiekty,**
2. **tablice,**
3. **łańcuchy,**
4. **liczby,**
5. **wartości logiczne (TRUE FALSE),**
6. **wartość NULL.**

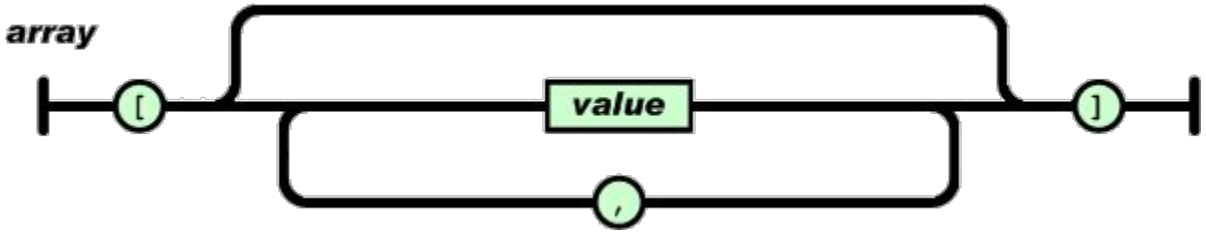
Obiekty JSON można zagnieżdżać w dowolnym stopniu jednak z reguły dla zachowania przejrzystości stosowana jest zasada “im bardziej płaska struktura tym lepiej”. Jednak nie w każdym przypadku i dla wszystkich zastosowań jest to możliwe do utrzymania.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

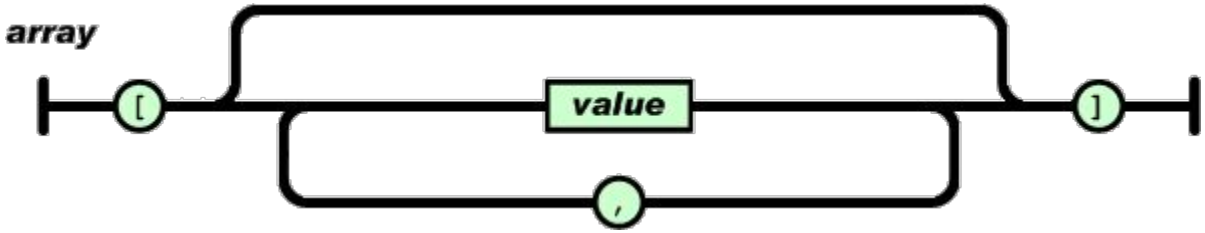
http://www.json.org/



JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

http://www.json.org/




```
[ {
    "firstname": "Jan",
    "lastname": "Kowalski",
    "age": 20
},
{
    "firstname": "Anna",
    "lastname": "Nowak",
    "age": 25
} ]
```

Uwaga:
Podobnie jak w JavaScript niedopuszczalne jest rozpoczynanie liczb całkowitych od "zera" np.:

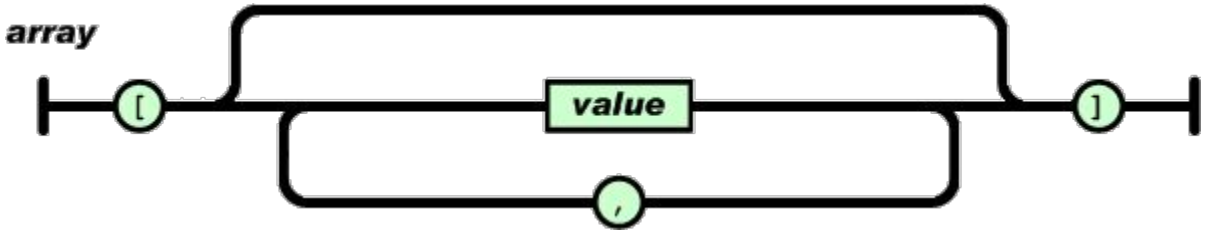
```
{ "liczba": 023 }
```

W niektórych przypadkach zapis taki może zostać zinterpretowany jako liczba w formacie ósemkowym.

JSON

W formacie JSON możemy także posługiwać się tablicami, które tworzą uporządkowane ciągi wartości o dowolnych typach dozwolonych przez JSONa (w tym tablice i obiekty).

http://www.json.org/




```
[ {
    "firstname": "Jan",
    "lastname": "Kowalski",
    "age": 20
},
{
    "firstname": "Anna",
    "lastname": "Nowak",
    "age": 25
} ]
```

Uwaga:

Format tekstowy (podobnie zresztą jak XML-a) zapewnia czytelność nie tylko dla parserów, ale również dla ludzi.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

JSON

Przykład tablicy obiektów:

```
{ "samochod": [
    {
        "Marka": "VW",
        "Model": "Golf",
        "Rocznik": 1999
    },
    {
        "Marka": "BMW",
        "Model": "S6",
        "Rocznik": 2007
    },
    {
        "Marka": "Audi",
        "Model": "A4",
        "Rocznik": 2009
    }
]}
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

XML

XML (eXtensible Markup Language) ***[Rozszerzalny Język Znaczników]*** **- jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekomendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaleta (zreszta podobną do JSONa) jest to że jest jest zapisywany w postaci zwykłych plików ASCII.**

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

XML

XML (eXtensible Markup Language) ***[Rozszerzalny Język Znaczników]*** **- jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekomendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaleta (zreszta podobną do JSONa) jest to że jest jest zapisywany w postaci zwykłych plików ASCII.**

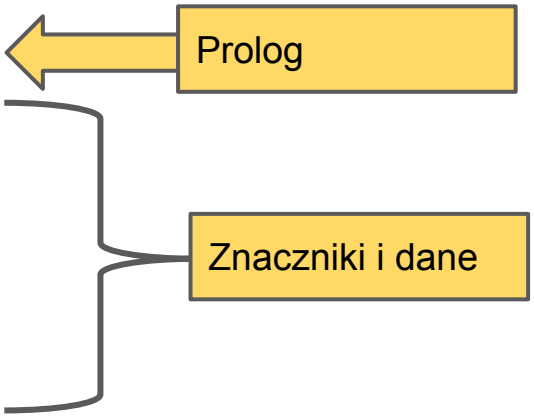
```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

XML

XML (eXtensible Markup Language) ***[Rozszerzalny Język Znaczników]*** **- jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekomendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaleta (zreszta podobną do JSONa) jest to że jest jest zapisywany w postaci zwykłych plików ASCII.**

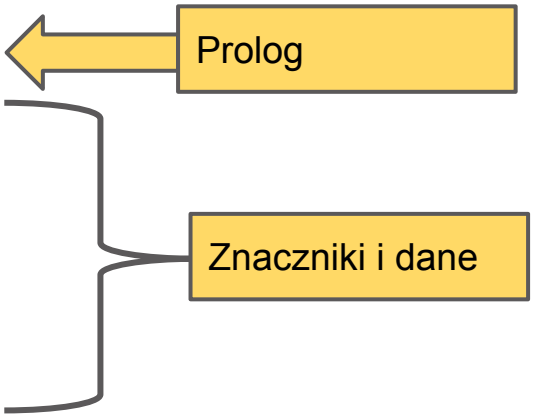
```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```



XML

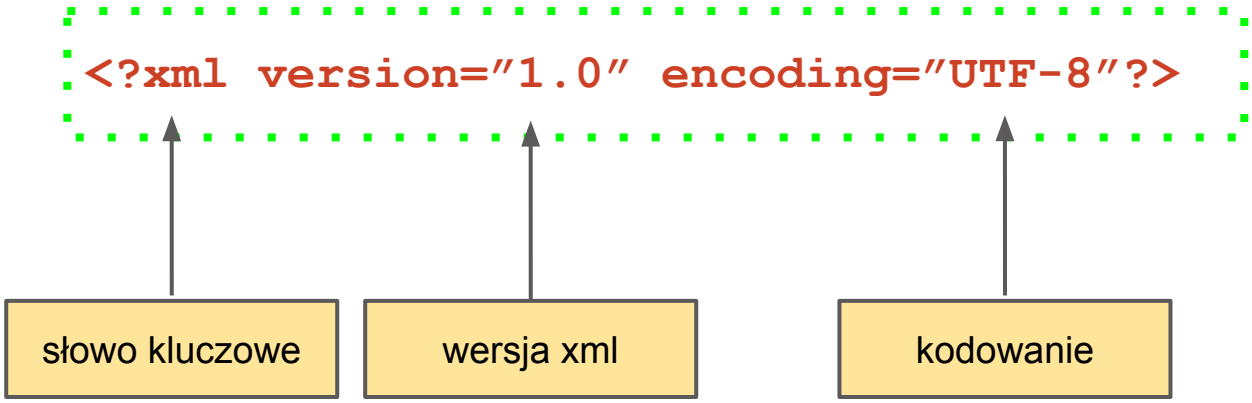
XML (eXtensible Markup Language) ***[Rozszerzalny Język Znaczników]*** **- jest to format do przenoszenia danych oparty o znaczniki. W zasadzie jest przeznaczony do opisu struktury danych oraz powiązań między nimi. Jest rekomendowany przez W3C jako standard wymiany danych. Jego niezwykłą zaleta (zreszta podobną do JSONa) jest to że jest jest zapisywany w postaci zwykłych plików ASCII.**

```
<?xml version="1.0" encoding="UTF-8"?>
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

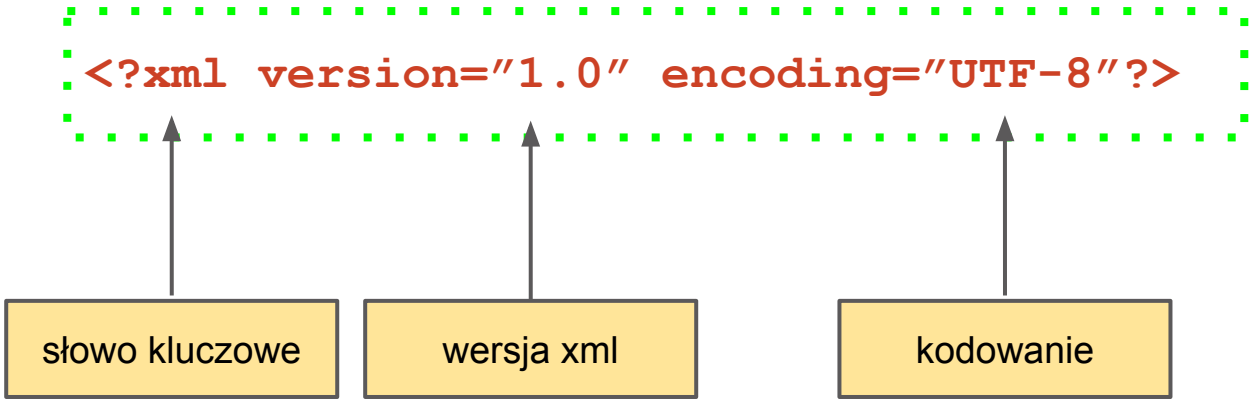


Prolog - powinien być ale nie zawsze jest konieczny!!!

XML


<?xml version="1.0" encoding="UTF-8"?>
słowo kluczowe
wersja xml
kodowanie

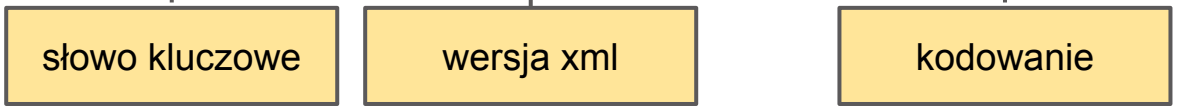
XML



W prologu moga znaleźć się też inne elementy np. informacja o stylach lub DTD.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
```



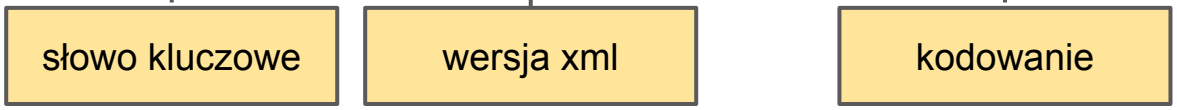
W prologu moga znaleźć się też inne elementy np. informacja o stylach lub DTD.

```
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

Nazwy kolejnych elementów mogą być dowolnym siągiem znaków rozpoczynającym się do litery lub podkreślenia.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
```



W prologu moga znaleźć się też inne elementy np. informacja o stylach lub DTD.

```
<osoba>
    <imie>Jan</imie>
    <nazwisko>Kowalski</nazwisko>
    <pesel>83010198741</pesel>
</osoba>
```

Nazwy kolejnych elementów mogą być dowolnym siągiem znaków rozpoczynającym się do litery lub podkreślenia.

Znacznik z tekstem (lub innymi znacznikami wewnątrz) jest bardzo często nazywany węzłem (node). Natomiast pierwszy element "wezłem głownym" (root-node).

XML

W plikach XML możemy również podobnie jak w HTML stosować kaskadowe arkusze stylu, które pozwalają odpowiednio sformatować wygląd przenoszonych danych. Dołączenie CSS odbywa się przez dyrektywę “xml-stylesheet”.

```
KOMIS {
        display: block;
        background-colr: #AAAAA;
        color: #FFFFFF;
    }
```

```
<?xml version=”1.0” encoding=”utf-8”?>
<?xml-stylesheet type=”text/css” href=”styl.css”?>
<KOMIS>
    <Samochod>
        <marka>BMW</marka>
    </Samochod>
</KOMIS>
```

XML

W plikach XML możemy również podobnie jak w HTML stosować kaskadowe arkusze stylu, które pozwalają odpowiednio sformatować wygląd przenoszonych danych. Dołączenie CSS odbywa się przez dyrektywę “xml-stylesheet”.

```
KOMIS {
        display: block;
        background-colr: #AAAAA;
        color: #FFFFFF;
    }
```

```
<?xml version=”1.0” encoding=”utf-8”?>
<?xml-stylesheet type=”text/css” href=”styl.css”?>
<KOMIS>
    <Samochod>
        <marka>BMW</marka>
    </Samochod>
</KOMIS>
```

Sposób przygotowanie CSS jest dokładnie taki sam jak dla HTML.

Porównanie JSON vs. XML

```
{ "samochod": [
    {
        "Marka": "VW",
        "Model": "Golf",
        "Rocznik": 1999
    },
    {
        "Marka": "BMW",
        "Model": "S6",
        "Rocznik": 2007
    },
    {
        "Marka": "Audi",
        "Model": "A4",
        "Rocznik": 2009
    }
]}
```

```
<?xml version="1.0" encoding="utf-8"?>
<KOMIS>
  <Samochod>
    <Marka>VW</Marka>
    <Model>Golf</Model>
    <Rok>1999</Rok>
  </Samochod>
  <Samochod>
    <Marka>BMW</Marka>
    <Model>S6</Model>
    <Rok>2007</Rok>
  </Samochod>
  <Samochod>
    <Marka>Audi</Marka>
    <Model>A3</Model>
    <Rok>2009</Rok>
  </Samochod>
</KOMIS>
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Porównanie JSON vs. XML

Obecnie wiele usług sieciowych wykorzystuje do przekazywania danych pliki XML oraz JSON. Standard XML jest tzw. “ścisły formatem”, który pozwala na łatwą, szybką i wydajną interpretację, niezależnie od platformy sprzętowej czy systemowej. Natomiast JSON cechuje się “lekkością” oraz bardzo przerzystą strukturą. Dla obu formatów w większości języków programowania (kompilowalnych oraz interpretowalnych) istnieją parsery pozwalające na łatwe odczytywanie, zapisywanie i modyfikowanie danych.

Który format wybrać ?

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Porównanie JSON vs. XML

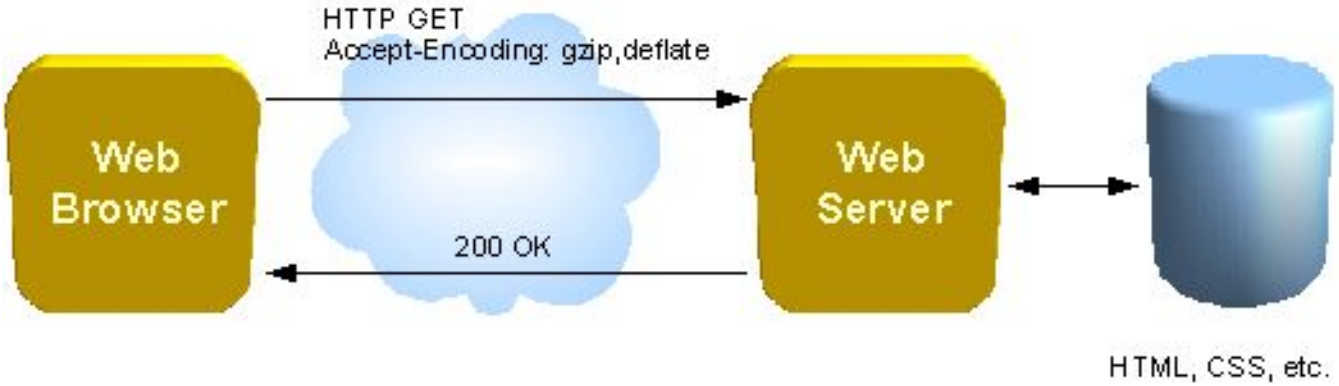
Obecnie wiele usług sieciowych wykorzystuje do przekazywania danych pliki XML oraz JSON. Standard XML jest tzw. “ścisły formatem”, który pozwala na łatwą, szybką i wydajną interpretację, niezależnie od platformy sprzętowej czy systemowej. Natomiast JSON cechuje się “lekkością” oraz bardzo przerzystą strukturą. Dla obu formatów w większości języków programowania (kompilowalnych oraz interpretowalnych) istnieją parsery pozwalające na łatwe odczytywanie, zapisywanie i modyfikowanie danych.

Który format wybrać ?

- **każdy z nich ma wady i zalety, dlatego wóbór zależy od tego do czego dany format ma służyć oraz od preferencji programisty.**
- **w przypadku JavaScriptu, lepiej wybrać JSONa, ponieważ łatwiej jest operować na obiektach JSONowych niż paroswać i XML.**

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

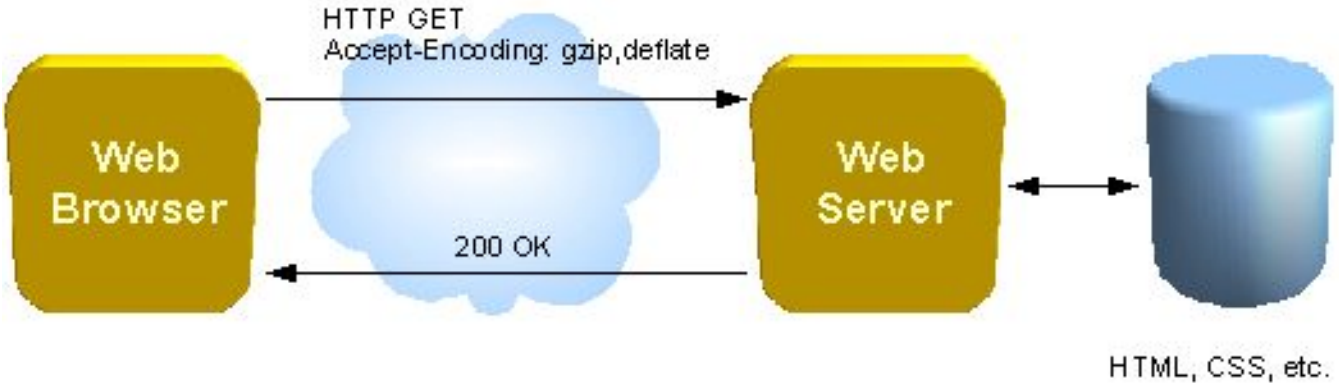
Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź.
W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

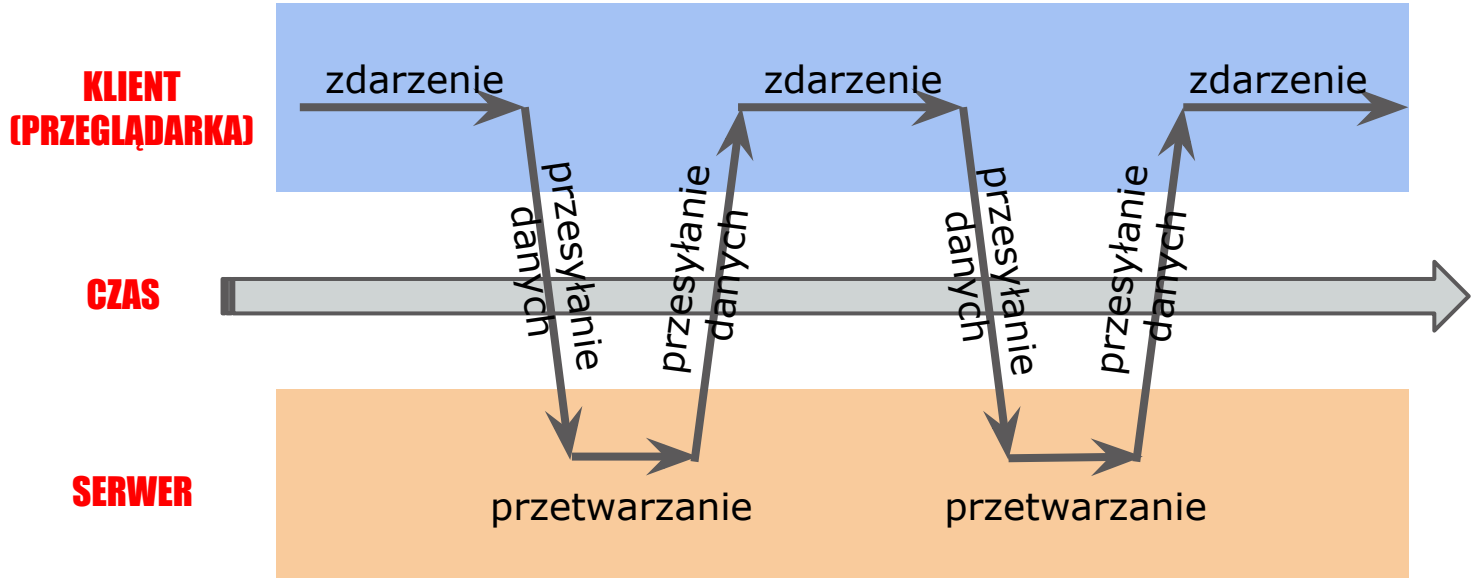
Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź.
W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrana) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

Synchroniczne żądanie http



Jest to model synchronicznej komunikacji HTTP, gdzie klient wysyła żądanie, serwer je odbiera następnie przetwarza i na końcu generuje odpowiedź.
W sytuacji takiej klient musi czekać z kolejnym żądaniem do momentu kiedy nie dostanie odpowiedzi od serwera.

W modelu synchronicznym mamy bardzo mały poziom aktywności oraz interaktywności strony, strona musi być przeładowana (pobrana) po każdym żądaniu klienta, jeśli strony są złożone to proces ten jest długi.

AJAX (“asynchroniczność”)

W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.

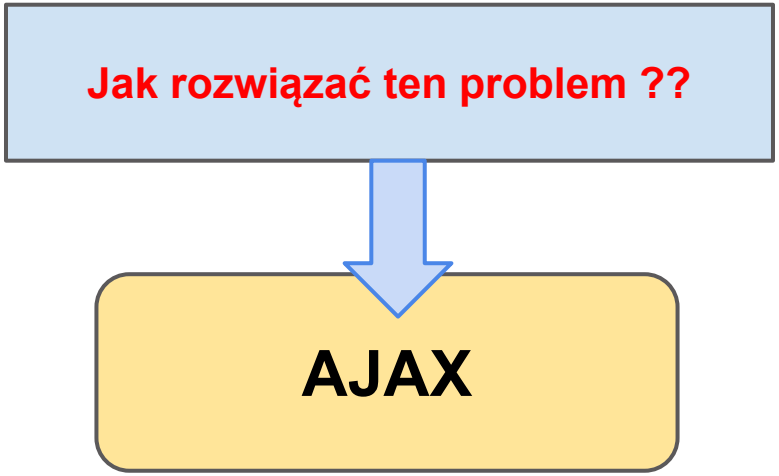
AJAX (“asynchroniczność”)

W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.

Jak rozwiązać ten problem ??

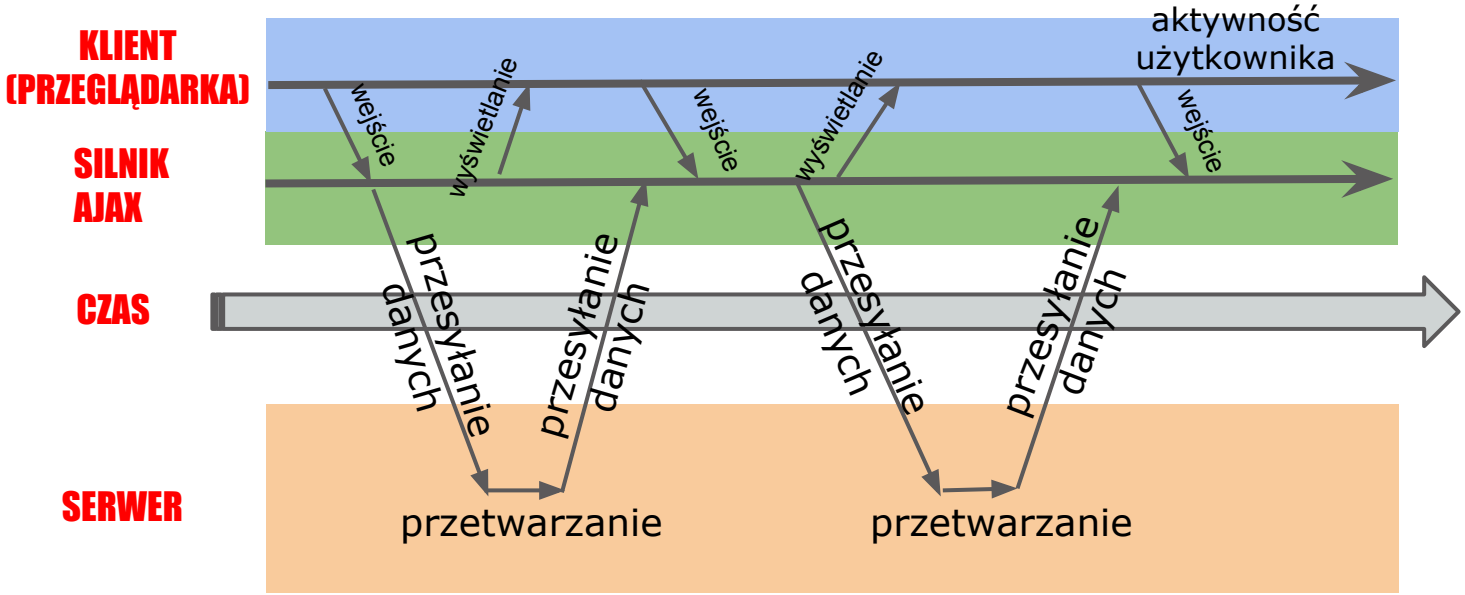
AJAX (“asynchroniczność”)

W klasycznych stronach (takich jak omawialiśmy do tej chwili) wszystkie żadania jakie były przekazywane przez protokół http (lub https) były synchroniczne. Oznaczało to, że strona po wykonaniu “działania” przez użytkownika musiał zostać przeładowana. Było to wygodne jednak nosło ze sobą konieczności “oczekiwania” na zakończenie poprzedniej operacji aby można było wysłać kolejne żądanie oraz niepotrzebnie generowała ruch sieciowy przez ciągłe przesyłanie tych samych plików.



(asynchroniczne przesyłanie danych przez protokół HTTP (HTTPS)

Asynchroniczne żądanie http



Jest to model asynchronicznej komunikacji HTTP, gdzie klient (przeglądarka) nie czeka na przyjście odpowiedzi na żądanie serwera, a wykonuje dalsze żądania.

W takim modelu nie ma konieczności przeładowania strony przy każdej operacji klineta, wystarczy, że zostaną doczytane brakujące dane, a dzięki odpowiednim narzędziom zmodyfikowana zostanie zawartość strony.

AJAX (asynchroniczność)

AJAX (Asynchronous JavaScript and XML) [Asynchroniczny JavaScript i XML] - model komunikacji sieciowej w której komunikacja pomiędzy klientem a serwerem odbywa się bez przeładowania dokumentu. Techniki służące do obsługi tej usługi są następujące:

1. **XMLHttpRequest - klasa która umożliwiająca asynchroniczne przesyłanie danych pomiędzy klientem a serwerem.**
2. **JavaScript - język skryptowy pośredniczący w komunikacji.**
3. **XML - język znaczników który opisuje dane (w ogólności może to być dowolny format np. JSON).**

Głównym zadaniem modelu AJAX jest otwarcie połączenia z pomiędzy klientem a serwerem.

AJAX (asynchroniczność)

Aby rozpocząć korzystanie z AJAX należy utworzyć obiekt klasy XMLHttpRequest():

```
var request = new XMLHttpRequest();
```

Od tego miejsca mamy dostęp do zestawu metod i własności przynależących do klasy XMLHttpRequest, które pozwalają na obsługę połączenia.

Zestaw metod:

- **open("metoda", "url", isAsynchronic, "user", "password")** - metoda otwiera połączenie do serwera. Przyjmuje połączenie parametry: *"metoda"* - definijemy GET czy POST, *"url"* - adres pliku na zdalnym serwerze, *"isAsynchronic"* - paramter true/false określa czy żądanie ma być asynchroniczne.
- **send("content")** - przesyłą żądanie do serwera.
- **abort()** - przerywa żądanie do serwera.
- **setRequestHeader()** - wysyła nagłówek do serwera.
- **getResponseHeader("nazwa_nagłówka")** - pobiera nagłówek z serwera
- **getAllResponseHeaders()** - pobiera wszystkie wysłane nagłówki http jako string.

AJAX (asynchroniczność)

Aby rozpocząć korzystanie z AJAX należy utworzyć obiekt klasy XMLHttpRequest():

```
var request = new XMLHttpRequest();
```

Od tego miejsca mamy dostęp do zestawu metod i własności przynależących do klasy XMLHttpRequest, które pozwalają na obsługę połączenia.

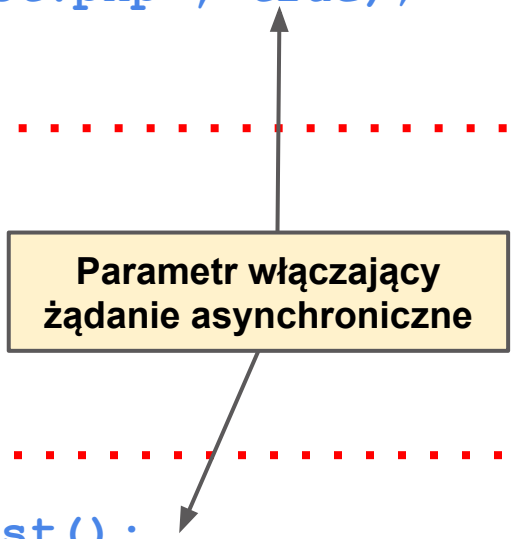
Zestaw własności:

- **onreadystatechange** - zdarzenie odpalane w chwili zmiany stanu danego połączenia
- **readyState** - zawiera aktualny status połączenia
  responseText - zawiera zwrócone dane w formie tekstu
- **responseXML** - zawiera zwrócone dane w formie drzewa XML (jeżeli zwrócone dane są prawidłowym dokumentem XML)
- **status** - zwraca status połączenia np 404 - gdy strona nie istnieje, itp)
- **statusText** - zwraca status połączenia w formie tekstowej - np 404 zwróci Not Found

AJAX (asynchroniczność)

Przykład utworzenia żadania asynchronicznego metodą GET:

```
var request = new XMLHttpRequest();
request.open("GET", "/test/test.php", true);
request.send();
```




```
var request = new XMLHttpRequest();
request.open("POST", "/test/test.php", true);
request.setRequestHeader("Content-Type","text/xml");
request.send("<osoba>Jan Kowalski</osoba>");
```

AJAX (asynchroniczność)

Możemy również sprawdzać stan połączenia z serwerem za pomoca własności readyState:

- **0: połączenie nie nawiązane,**
- **1: połączenie nawiązane,**
- **2: żądanie odebrane,**
- **3: przetwarzanie,**
- **4: dane zwrócone i gotowe do użycia**

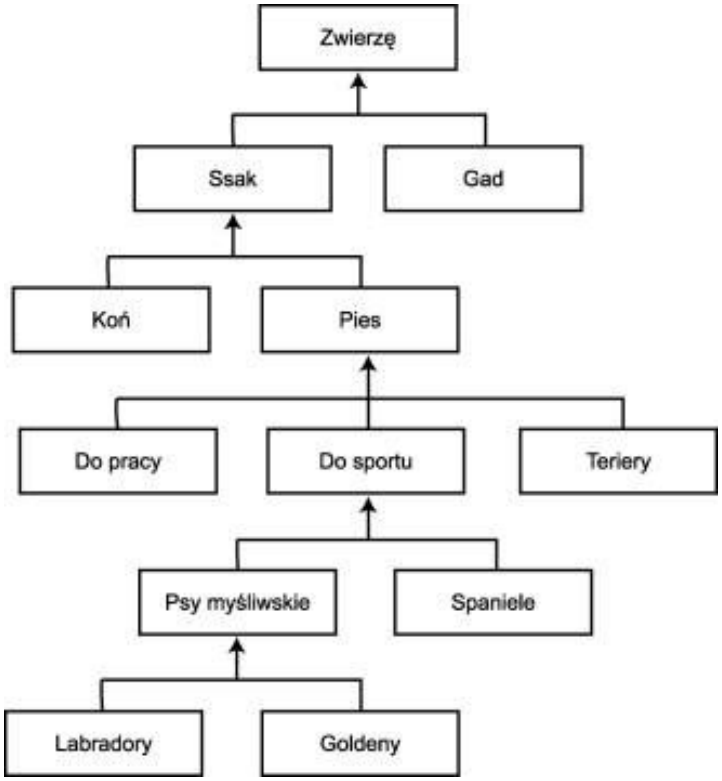
```
var request = new XMLHttpRequest();

request.open("GET", "/test/test.php", true);
request.onreadystatechange = function(){
    if ( request.readyState == 4 ) {
        request = null;
    }
};
request.send();
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie

JavaScript jak już wielokrotnie wspominano jest językiem bezklasowym i nie przenosi typowego z języków kompilowalnych sposobu dziedziczenie.

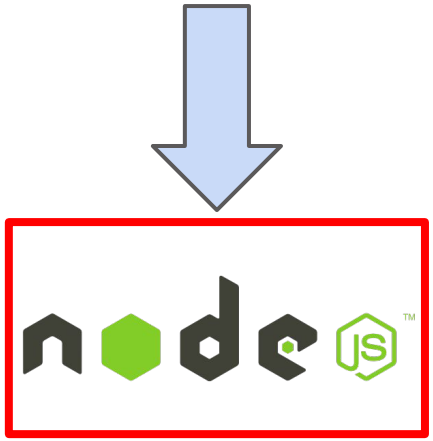


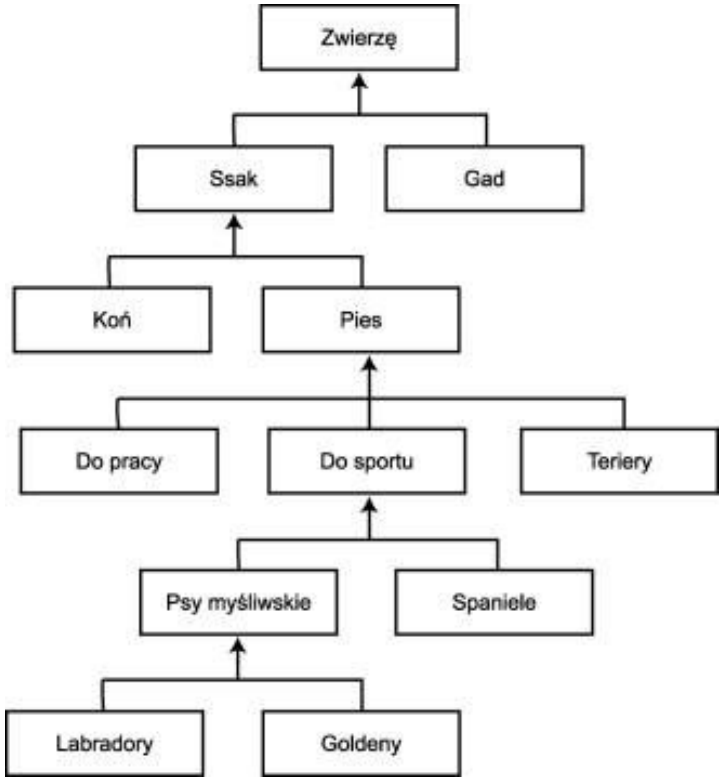
UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie

JavaScript jak już wielokrotnie wspominano jest językiem bezklasowym i nie przenosi typowego z języków kompilowalnych sposobu dziedziczenie.

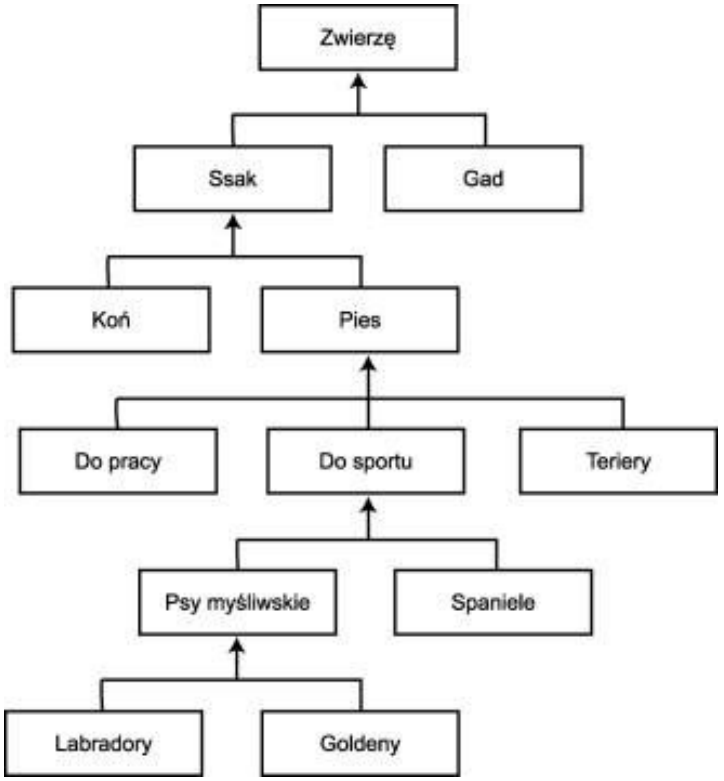
Sposób dziedziczenia w JS różni się od klasycznego podejścia a tym samym i w ...





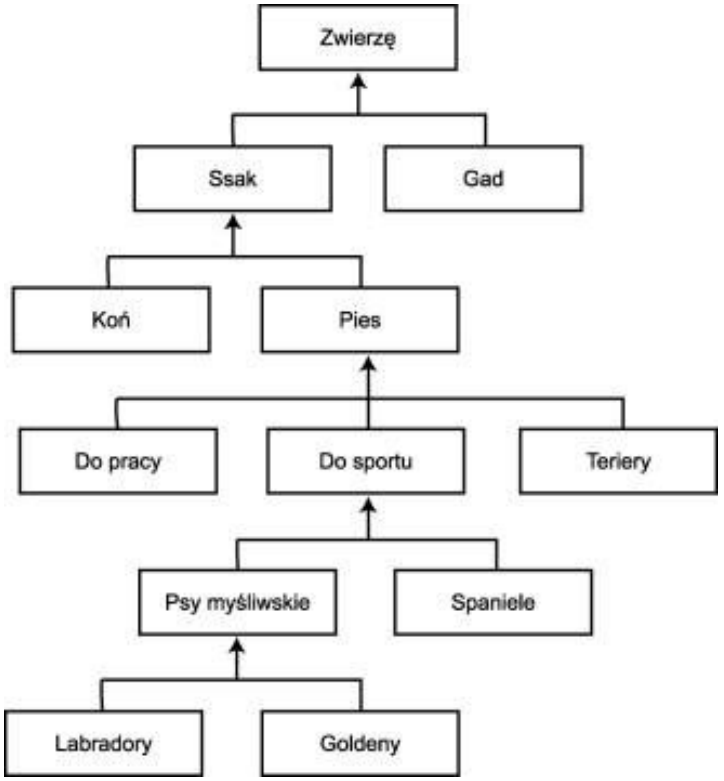
Dziedziczenie

Dziedziczenie w językach klasycznych ma przede wszystkim za zadanie umożliwiać wielokrotne wykorzystanie tego samego kodu: np.jeśli dwie klasy są podobne, należy określić tylko dzielące je różnice. Po drugie dziedziczenie ma w sobie specyfikację typów co pozwala na zmniejszenie konieczności rzutowania



Dziedziczenie

Dziedziczenie w językach klasycznych ma przede wszystkim za zadanie umożliwiać wielokrotne wykorzystanie tego samego kodu: np.jeśli dwie klasy są podobne, należy określić tylko dzielące je różnice. Po drugie dziedziczenie ma w sobie specyfikację typów co pozwala na zmniejszenie konieczności rzutowania



JS nie posiada typów!

Co wtedy z dziedziczeniem ??

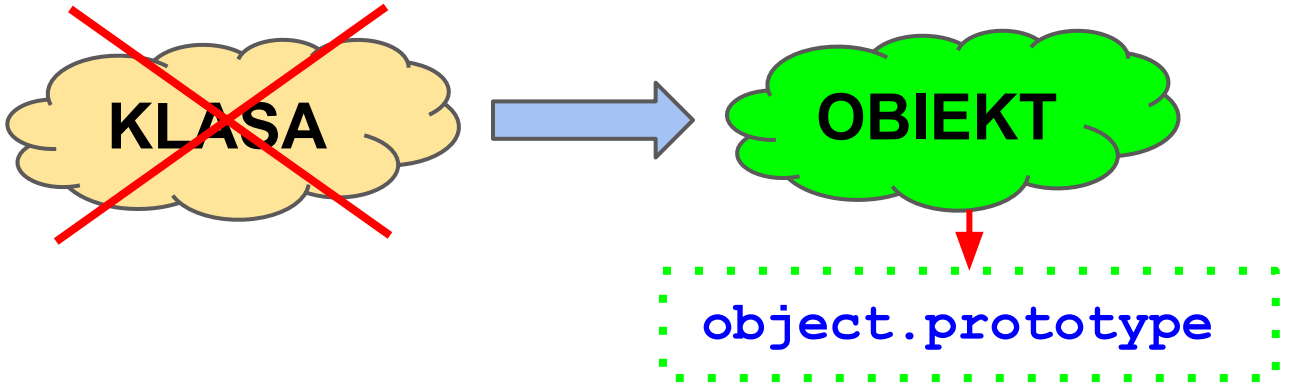
UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

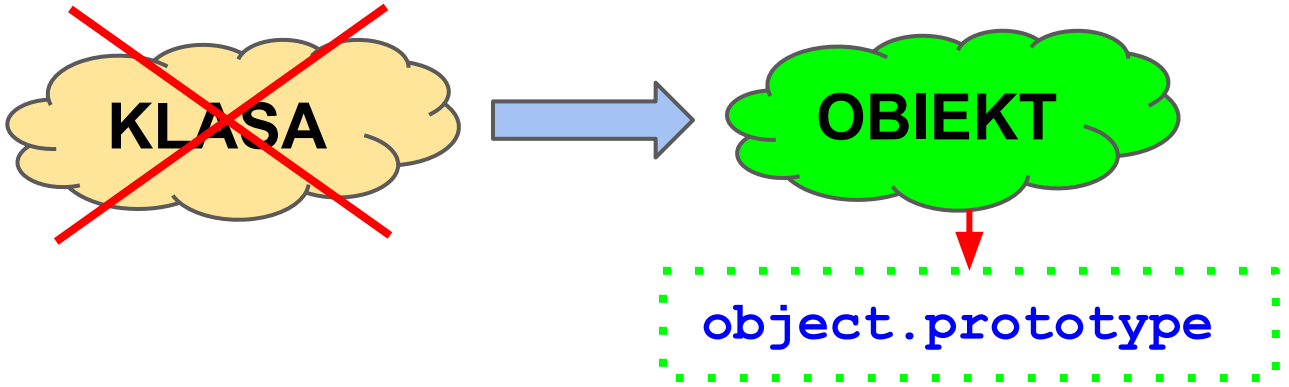
Dziedziczenie prototypowe w JS

Dziedziczenie prototypowe w JS



W języku JavaScript (a tym samym we wszystkich środowiskach programistycznych opartych o ten język) dziedziczenie odbywa się w sposób prototypowy. Każdy obiekt w JavaScript można traktować jako prototyp który w dowolnym momencie można rozszerzać, a jego własności przekazywać (cedować) na inne dziedziczące obiekty.

Dziedziczenie prototypowe w JS



W języku JavaScript (a tym samym we wszystkich środowiskach programistycznych opartych o ten język) dziedziczenie odbywa się w sposób prototypowy. Każdy obiekt w JavaScript można traktować jako prototyp który w dowolnym momencie można rozszerzać, a jego własności przekazywać (cedować) na inne dziedziczące obiekty.

Obiekty dziedziczą po obiektach!

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

Obiekty dziedziczą po obiektach!

`object.prototype`

Właściwość prototype posiadają obiekty które są tworzone przez funkcję konstruktora. Jednocześnie sam prototyp jest obiektem. Często dziedziczenie w JS określa się mianem dziedziczenia opartego o instancje.

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);
```

```
> undefined
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć własności prototype (który też jest obiektem):

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

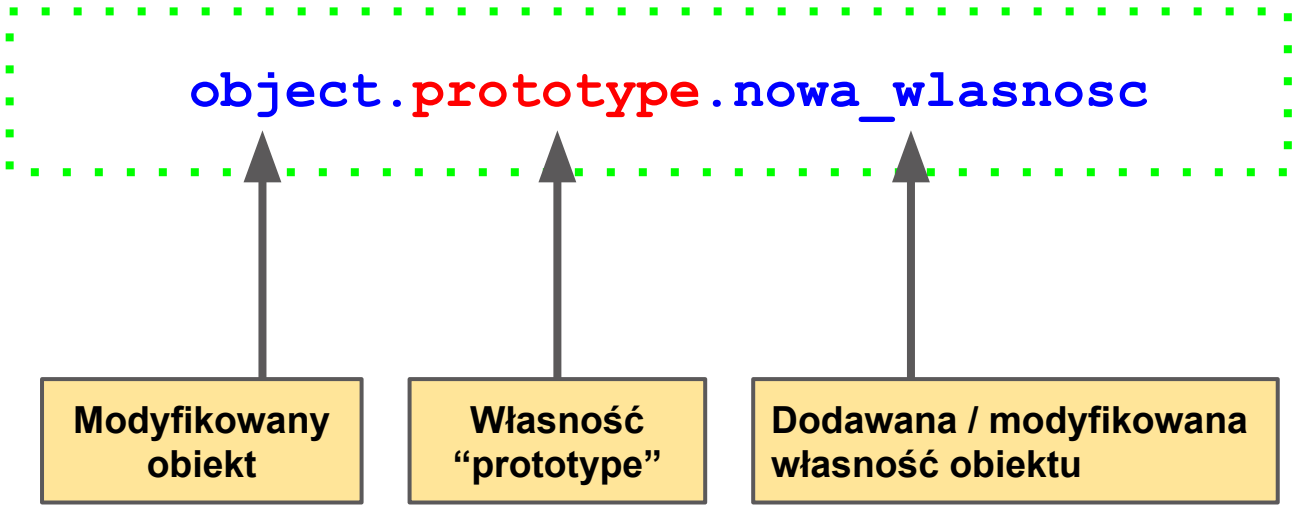
Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć własności prototype (który też jest obiektem):

```
object.prototype.nowa_wlasnosc
```

Dziedziczenie prototypowe w JS

Do zmiany lub rozszerzenia utworzonego obiektu możemy użyć własności prototype (który też jest obiektem):



UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);
```

```
> undefined
  czesc
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);

var p = new Obiekt();
console.log(p.czesc);
```

```
> undefined
  czesc
  czesc
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);

var p = new Obiekt();
console.log(p.czesc);

Obiekt.prototype.PowedzCzesc = function(){
    console.log(this.czesc);
}
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);

var p = new Obiekt();
console.log(p.czesc);

Obiekt.prototype.PowedzCzesc = function(){
    console.log(this.czesc);
}
o.PowedzCzesc();
p.PowedzCzesc();
```

```
> undefined
  czesc
  czesc
  czesc
  czesc
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

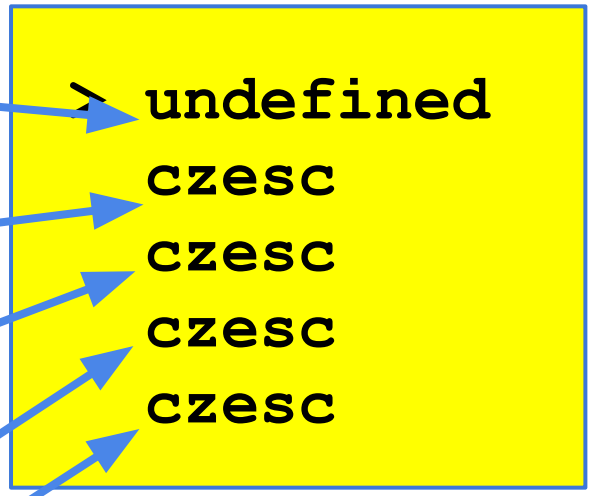
Dziedziczenie prototypowe w JS

```
var Obiekt = function() {};
var o = new Obiekt();
console.log(o.czesc);

Obiekt.prototype.czesc = 'czesc';
console.log(o.czesc);

var p = new Obiekt();
console.log(p.czesc);

Obiekt.prototype.PowedzCzesc = function(){
    console.log(this.czesc);
}
o.PowedzCzesc();
p.PowedzCzesc();
```



Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){
        this.x = x;
        this.y = y;
        this.dodaj = function(){
           return this.x + this.y;
        };
};
var f = new Wspolrzedne(4,7);
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){
        this.x = x;
        this.y = y;
        this.dodaj = function(){
           return this.x + this.y;
        };
};
var f = new Wspolrzedne(4,7);

Wspolrzedne.prototype.pokazX = function(){
        console.log(this.x);
};
Wspolrzedne.prototype.pokazY = function(){
        console.log(this.y);
};
Wspolrzedne.prototype.pomnoz = function(){
        return this.x * this.y;
};
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
var Wspolrzedne = function(x,y){
        this.x = x;
        this.y = y;
        this.dodaj = function(){
           return this.x + this.y;
        };
};
var f = new Wspolrzedne(4,7);

Wspolrzedne.prototype.pokazX = function(){
        console.log(this.x);
};
Wspolrzedne.prototype.pokazY = function(){
        console.log(this.y);
};
Wspolrzedne.prototype.pomnoz = function(){
        return this.x * this.y;
};
Wspolrzedne.prototype.pokazWynikDodaj = function(){
        console.log(this.dodaj())};
Wspolrzedne.prototype.pokazWynikPomnoz = function(){
        console.log(this.pomnoz())};
```

```
f.pokazX();
f.pokazY();
f.pokazWynikDodaj();
f.pokazWynikPomnoz();
```

```
> 4
  7
  11
  28
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
function Mother() {
  this.methodA = function() {
    console.log('Mother::methodA()');
  }
  this.methodB = function() {
    console.log('Mother::methodB()');
  }
}

function Child() {
    this.methodC = function () {
    console.log('Child::methodC()');
  }
}
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
function Mother() {
  this.methodA = function() {
    console.log('Mother::methodA()');
  }
  this.methodB = function() {
    console.log('Mother::methodB()');
  }
}

function Child() {
    this.methodC = function () {
    console.log('Child::methodC()');
  }
}

Child.prototype = new Mother();

m = new Mother();
c = new Child();
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

```
function Mother() {
  this.methodA = function() {
    console.log('Mother::methodA()');
  }
  this.methodB = function() {
    console.log('Mother::methodB()');
  }
}

function Child() {
    this.methodC = function () {
    console.log('Child::methodC()');
  }
}

Child.prototype = new Mother();

m = new Mother();
c = new Child();
```

```
m.methodA();
m.methodB();
c.methodA();
c.methodB();
c.methodC();
```

```
> Mother::methodA()
  Mother::methodB()
  Mother::methodA()
  Mother::methodB()
  Child::methodC()
```

UNIWERSYTET JAGIELLOŃSKI W KRAKOWIE

Dziedziczenie prototypowe w JS

Możemy też bez żadnych problemów “przeciążyć” metodę B zdefiniowaną w funkcji “Child”, oddziedziczoną po funkcji “Mother”:

```
Child.prototype.methodB = function () {
    console.log('Child::methodB()');
  }


m.methodA();
m.methodB();
c.methodA();
c.methodB();
c.methodC();
```

Dziedziczenie prototypowe w JS

Możemy też bez żadnych problemów “przeciążyć” metodę B zdefiniowaną w funkcji “Child”, oddziedziczoną po funkcji “Mother”:

```
Child.prototype.methodB = function () {
    console.log('Child::methodB()');
  }


m.methodA();
m.methodB();
c.methodA();
c.methodB();
c.methodC();
```

```
> Mother::methodA()
  Mother::methodB()
  Mother::methodA()
  Child::methodB()
  Child::methodC()
```

UNIWERSYTET
JAGIELLOŃSKI
W KRAKOWIE

KONIEC WYKŁADU 6