

Skrót do egzaminu EE09 / E14 / INF.03 posiada odnośniki do różnych dodatkowych materiałów. Zaleca się je sprawdzić, aby móc głębiej zaznajomić się z tematem. Fizycznie nie jesteśmy w stanie tutaj umieścić wszystkiego



1. TWORZENIE WITRYN INTERNETOWYCH:

Podstawowe pojęcia związane z witrynami internetowymi:

- strona internetowa - dokument stworzony w HTML lub XHTML, zapisany w pliku i umieszczony na serwerze, odczytywany przy pomocy przeglądarki internetowej
- witryna internetowa - zbiór stron internetowych powiązanych tematycznie i umieszczonych na jednym serwerze
- portal internetowy - serwis informacyjny zawierający informacje na różne tematy
- serwer internetowy - komputer podłączony do internetu, który świadczy różne usługi w internecie oraz w odniesieniu do oprogramowania uruchamianego na tym komputerze
- statyczna strona internetowa - strona, której zawartość i wygląd nie zmieniają się przy każdym odwołaniu się do niego
- dynamiczna strona internetowa - strona zmieniająca zawartość i wygląd w zależności od interakcji z użytkownikiem

HTML:

HTML5 - standard HTML posiadający nowe, lepsze możliwości niż jego poprzednie wersje:

<!DOCTYPE html> - deklaracja typu dokumentu

<html> - wszystko co jest pomiędzy <html>, a </html> będzie interpretowane jako strona internetowa

<head> - tutaj zamieszczamy informacje, które nie wyświetlają się bezpośrednio na stronie internetowej, takie jak np. tytuł, autorzy strony, kodowanie znaków, słowa kluczowe itp.

</head> - znacznik zamykający sekcję head

<body> - ciało dokumentu, wszystko to co ma zawierać strona internetowa (kod)

</body> - znacznik zamykający sekcję body

</html> - koniec HTMLa

Kodowanie polskich znaków (umieszczamy pomiędzy <head> , a </head>):

```
<meta charset="UTF-8">
```

Tytuł strony (także w head):

```
<title>Tytuł strony</title>
```

Słowa kluczowe (także w head):

```
<meta name="Keywords" content="słowa, kluczowe">
```

Zmiana koloru tła (umieszczamy atrybut w znaczniku body):

```
<body bgcolor="red">
```

Formatowanie tekstu:

CZCIONKA: Tekst w danej czcionce, który będzie się wyświetlał

POGRUBIENIE: Pogrubienie

WZMOCNIENIE: Pogrubienie z wyróżnieniem

KURSYWA: <i>Kursywa</i>

EMFAZA: Pochylenie z wyróżnieniem

PODKREŚLENIE: <u>Podkreślenie</u>

ZAZNACZENIE: <mark>Zaznaczenie</mark>

ZMIANA KOLORU CZCIONKI W CAŁYM DOKUMENCIE: <body text="green">

INDEKS GÓRNY: ^{Indeks górnny}

INDEKS DOLNY: _{Indeks dolny}

Pozioma linia: <hr/>

Znaki specjalne w HTML:

symbol	numer	nazwa	opis
 	 	 	twarda spacja
'	'	-	pojedyńczy cudzysłów
"	"	"	podwójny cudzysłów
&	&	&	znak ampersand
#	#	-	symbol numeru (hash)
\$	$	-	symbol dolara
£	£	£	symbol funta
€	€	€	symbol euro
™	™	™	znak towarowy
©	©	©	znak praw autorskich
®	®	®	zastrzeżony znak towarowy
<	<	<	znak mniejszości
«	«	&lquo;	podwójny znak mniejszości
>	>	≷	znak większości
»	»	»	podwójny znak większości
°	°	°	znak stopni
%	%	-	symbol procenta
@	@	-	małpa

Akapity:

ZWYKŁY AKAPIT: <p>Akapit</p>

WYSRODKOWANY: <p align="center">Wyśrodkowany akapit</p>

Wymuszenie wyświetlenie tekstu w nowej linii:

Listy:

LISTA NUMEROWANA:

Atrybut `type` pozwala na listowanie np. wg dużych (A)/małych (a) liter, cyfr rzymskich (I)/arabskich (1), natomiast atrybut `start`, mówi, od której liczby/litery ma zacząć się numeracja:

np.

```
<ol type="1">
<li>Pierwszy element</li>
<li> Drugi element</li>
</ol>
```

LISTA PUNKTOWANA:

Atrybut `type` pozwala wybór rodzaju punktowania np. `disc`, `square`, `circle`.

np.

```
<ul type="circle">
<li>Pierwszy element</li>
<li> Drugi element</li>
</ul>
```

Listy zagnieżdżone to takie listy, które są zamieszczone wewnątrz innej.

np.

```
<ul>
<li>Pierwszy</li>
<li>Drugi

<ul>
<li>Element 2.1</li>
<li>Element 2.2</li>
</ul>
</li>

<li>Trzeci</li>
</ul>
```

Tabele:

Znacznik `<table>` rozpoczyna tabelę, `<tr>` definiuje każdy jej wiersz, `<td>` każdą komórkę w wierszu.

np.

```
<table>
<tr><td>komórka 1-1</td><td>komórka 1-2</td></tr>
<tr><td>komórka 2-1</td><td>komórka 2-2</td></tr>
</table>
```

Odstępy między komórkami tworzącymi tabelę uzyskamy przy pomocy atrybutu (dla `table`) `cellspacing="n"`.

Łączenie komórek w wierszu: w znaczniku `<td>` definiujemy atrybut `colspan="n"` (n to ilość sąsiednich komórek do połączenia).

Łączenie komórek w kolumnie: w znaczniku `<td>` definiujemy atrybut `rowspan="n"` (n to ilość sąsiednich komórek do połączenia).

Wyrównywanie zawartości komórek: atrybut **align** (w poziomie), **valign** (w pionie) w znaczniku **<td>**

Tło komórki: w znaczniku **<td>** definiujemy atrybut **bgcolor**.

Blok służą do wydzielania większego fragmentu tekstu. Robimy to za pomocą znacznika **<div>**

np.

```
<div>  
Blok tekstu  
</div>
```

Za pomocą atrybutu align tekst zostanie odpowiednio wyrównany np. **<div align="center">**

Odsyłacze (hiperłącza):

Do innej strony internetowej: **Link do onet**

Do innych dokumentów: **Link do pliku**

Do pliku z grafiką: **Link do fotki**

Do innego miejsca w obrębie tej samej strony: **Okres godowy, a trójpolówka**

Grafika:

- format GIF - format 8-bitowy, pozwalający na zapisanie max 256 kolorów, niewielki rozmiar plików, umożliwia tworzenie grafik przezroczystych, istnieje możliwość stosowania przeplotu (obraz wczytywany fragmentami), można zastosować mechanizm redukcji palety kolorów
- format JPG - najpierw tworzony jest obraz czarno-biały, informacje o kolorach są zapisywane osobno, kolory niewidoczne dla oka zostają usunięte podczas analizy obrazu, można regulować stopień kompresji, 24-bitowa paleta kolorów
- format PNG - 48-bitowa paleta kolorów, 16-bitowa paleta odcieni szarości, efektywny i bezstratny mechanizm kompresji

Umieszczanie grafiki w kodzie HTML:

```

```

Jeśli obrazek znajduje się w tej samej lokalizacji co strona, wystarczy podać nazwę pliku. Jeśli jednak znajduje się w innej lokalizacji - wtedy należy podać pełny adres URL.

Atrybuty: *height* - wysokość, *width* - szerokość, *border* - grubość obramowania , *alt* - tekst zastępczy, na wypadek gdyby obrazek się nie wczytał.

Używając atrybutu *align*, możemy wyświetlić obraz otoczony tekstem:

```

```

Wszystko czego dziś chcę.

Za pomocą *align* możemy także wykorzystywać podczas wyrównywania obrazka względem tekstu (wartości *align* w takim wypadku to: *texttop*, *top*, *middle*, *bottom*).

Formularze:

Dzięki formularzom umieszczonym na stronie internetowej jesteśmy w stanie zbierać od użytkowników pewne informacje.

Atrybuty znacznika `<form>`:

- `action` - ustala adres poczty elektronicznej, skryptu PHP itd.
- `method` - określa metodę przekazywania skryptu pod wskazany adres (GET lub POST)

Pole `<input>` służy do wprowadzania danych tekstowych do formularza. Atrybuty:

- `name` - określa nazwę danego pola oraz nazwę zmiennej, do której zostanie podstawiona wprowadzona wartość, wartości tego atrybutu muszą być różne
- `value` - domyślna wartość pola
- `size` - określa liczbę znaków jaką można wpisać do pola
- `type` - określa typ pola
- `minlength / maxlength` - określa minimalną/maksymalną liczbę znaków, jaką można wpisać
- `min / max` - najwyższa / najniższa wartość (liczbową) jaką można wpisać
- `required` - służy do sprawdzenia, czy pole zostało wypełnione
- `pattern` - sprawdza, czy wartość została zdeklarowana tak, jak jest podane w wartości tego atrybutu (np. `pattern="[0-9]{2}-[a-z]{2}-[A-Z]{3}"`)

Typy pól:

- `text` - pole tekstowe
- `checkbox` - pole zaznaczenia opcji (można zaznaczyć lub odznać opcję)
- `radio` - pole wyboru (z grupy wybieramy tylko jedną)
- `password` - pole wprowadzania hasła (wprowadzane znaki nie są widoczne)
- `file` - możliwość dołączenia do formularza pliku, który zostanie przesłany na serwer
- `submit` - przycisk, uruchamiający wykonanie akcji zdefiniowanej w atrybucie "action"
- `reset` - przycisk, który usuwa wszystkie dane z formularza
- `button` - przycisk dowolnego przeznaczenia

```
<form action="mailto: aiobyplucek@mail.com method="post">
<input type="text" value="pole1" name="pierwsze" placeholder="Imię"maxlength="14">
</form>
```

Pole `<select>` służy do wyświetlenia w formularzu listy wartości i zachęca do wybrania jednej lub kilku z nich. Szczegółowe wartości tej listy są definiowane za pomocą `<option>`.

Atrybuty znacznika `<select>`:

- `size="n"` - określa ile pozycji widocznych jest w polu select
- `multiple` - można zaznaczyć więcej niż jedną pozycję

Atrybuty znacznika `<option>`:

- `selected` - opcja będzie domyślnie zaznaczona
- `value="wartość"` - określa wartość przypisaną do pozycji listy

```
<select name="lista">
<option selected>Mężczyzna</option>
<option>Kobieta</option>
</select>
```

Pole `<textarea>` służy do wpisania większej ilości tekstu. Przykłady jego atrybutów:

- `name="x"` - nazwa przypisana do pola
- `rows="n"` - liczba wierszy w polu tekstowym
- `cols="n"` - liczba kolumn (znaków) w polu tekstowym
- `wrap` - w trakcie wprowadzania tekstu linie są dostosowane do marginesów (jego wartości to: off, physical, virtual)

```
<textarea name="opinia" rows="4" cols="20" wrap="virtual">Popisz sobie</textarea>
```

Multimedia:

Wtyczka to program, który doinstalowany jest do przeglądarki po to, aby móc zwiększyć jej możliwości. Zwykle są to wtyczki do plików dźwiękowych i do filmów.

Formaty plików multimedialnych:

- .wav (format plików dźwiękowych)
- .midi (format plików dźwiękowych)
- .avi (format plików wideo)
- .mp3 (format plików dźwiękowych)
- .mpg / .mpeg (format plików wideo)
- .wmv (format plików wideo)
- .3gp (format plików wideo)
- .flac (format plików dźwiękowych)

W HTML5 do osadzenia plików multimedialnych na stronie internetowej wykorzystuje się znaczniki `<audio>` oraz `<video>` (Nie są wówczas potrzebne żadne wtyczki). Możemy użyć dodatkowych atrybutów takich jak:

- `poster` (miniaturka filmu)
- `autoplay` (automatyczne odtwarzanie)
- `controls` (wyświetlenie panelu sterowania wideo)
- `loop` (zapętlenie się odtwarzania wideo)
- `autobuffer` (automatyczne buforowanie wideo)
- `preload` (wczytywanie wideo wraz z załadowaniem strony)

```
<video width="400" controls>
<source src="mov_bbb.mp4" type="video/mp4">
<source src="mov_bbb.ogg" type="video/ogg">
Your browser does not support HTML5 video.
</video>
```

Znacznik `<canvas>` służy do tworzenia grafiki rastrowej na stronach internetowych. Definiuje on prostokątny obszar, w którym tworzony jest rysunek.

Zdarzenia:

Zdarzenia na stronie internetowej, występują, gdy użytkownik wykona jakąś czynność na danym elemencie strony, a ten element zareaguje robiąc kolejną czynność (na przykład uruchamiając jakąś funkcję).

Przykłady zdarzeń wewnętrznych:

- **onclick** (gdy wystąpiło kliknięcie myszą na dany element strony)
- **ondblclick** (gdy wystąpiło podwójne kliknięcie myszą na dany element strony)
- **onmousedown** (gdy przycisk myszy został wcisnięty na danym elemencie strony)
- **onmouseup** (gdy przycisk myszy został zwolniony z danego elementu strony)
- **onmouseover** (gdy kurSOR myszy został umieszczony na danym elemencie)
- **onmousemove** (gdy kurSOR myszy został przesunięty wewnątrz elementu)
- **onmouseout** (gdy kurSOR myszy został przesunięty poza element)
- **onkeypress** (gdy klawisz klawiatury został naciśnięty i zwolniony)
- **onkeydown** (gdy klawisz klawiatury został naciśnięty)
- **onkeyup** (gdy klawisz klawiatury został zwolniony)
- **onfocus** (gdy został wybrany element formularza)
- **onblur** (gdy kurSOR opuszcza element formularza)
- **onselect** (gdy w polu tekstowym został zaznaczony tekst)
- **onchange** (gdy element formularza zmienia wartość)
- **onsubmit** (gdy nastąpiło wysłanie formularza)
- **onreset** (gdy nastąpiło wyczyszczenie formularza)
- **onload** (gdy dokument został wczytany)
- **onunload** (gdy użytkownik opuszcza stronę)

Kaskadowe arkusze stylów (CSS):

Kaskadowe arkusze stylów definiują sposób formatowania elementów dokumentu HTML.

Podstawowa składnia języka CSS: selektor { właściwość: wartość; }

W dokumencie HTML istnieje kilka sposobów na dołączanie stylów CSS:

- styl lokalny
- wewnętrzny arkusz stylów
- zewnętrzny arkusz stylów

Znacznik **** służy do grupowania elementów strony w celu nadania im określonego stylu.

Znacznik **<div>** umożliwia podzielenie strony internetowej na bloki, które można formatować za pomocą stylów.

Styl lokalny umieszcza w tej samej linii, w której znajduje się element formatowany.

```
<body>
<p style="color:brown">Tekst będzie brązowy tylko w tym akapicie.</p>
</body>
```

Wewnętrzny arkusz stylów umieszcza się w dokumencie HTML umieszczając go między znacznikami **<style>** i **</style>** w części **<head>**.

```
<head>
<style>
body {
```

```
background-color: bisque;  
}  
</style>  
</head>
```

Zewnętrzny arkusz stylów polega na utworzeniu osobnego pliku z rozszerzeniem .css. W nim umieszcza się cały kod CSS.

W HTMLu podajemy tylko odnośnik:

```
<!DOCTYPE html>  
<html>  
<head>  
<link rel="stylesheet" type="text/css" href="arkusz.css">  
</head>  
<body>  
</body>  
</html>
```

Z uwagi na fakt, iż w dokumencie mogą być odwołania do kilku zewnętrznych arkuszy stylów lub na stronie może być użyty zewnętrzny i wewnętrzny arkusz stylów oraz style lokalne, które mogą powodować konflikt związany ze sposobem formatowania tego samego elementu w różnych arkuszach stylów, ustalono ważność stylów w html:

1. Styl lokalny
2. Rozciąganie stylu
3. Wydzielone bloki
4. Wewnętrzny arkusz stylów
5. Zewnętrzny arkusz stylów
6. Import stylów do zewnętrznego arkusza
7. Atrybuty definiowane w dokumencie HTML

Zasady kaskadowości można zmieniać poprzez polecenie *!important*. Spowoduje to, że wartość zdefiniowana dla elementu będzie miała pierwszeństwo bez względu na wszystko.

```
<p style="color:brown" !important>Tekst będzie brązowy tylko w tym akapicie.</p>
```

Selektorem może być dowolny element języka HTML, dla którego chcemy definiować parametry formatowania. W zależności od tego, w jaki sposób odwołujemy się w definicji reguły do formatowanych elementów, wyróżniamy następujące rodzaje selektorów:

- selektory elementów
- selektory atrybutów
- selektory specjalne
- selektory pseudoklas
- selektory pseudoelementów

Selektory elementów:

Selektor typu służy do definiowania formatowania znaczników występujących na stronie.

```
p {  
color: bisque;  
}
```

Selektor uniwersalny to selektor pasujący do wszystkich znaczników.

```
p,h1,table {  
color: bisque;  
}
```

Selektor potomka służy do formatowania elementów zawartych wewnątrz innych znaczników (wewnątrz znacznika znajduje się inny znacznik).

```
<h2>Witamy bardzo serdecznie w hotelu <i>Arkadia</i></h2>
```

Selektor dziecka służy do definiowania elementów, które znajdują się o jeden rząd niżej w hierarchii drzewa dokumenty. Symbol > oznacza bezpośredni związek między elementami. Znacznik będący dzieckiem musi wystąpić bezpośrednio wewnątrz znacznika nadzawanego.

```
<style>  
p>u {  
font-family: monospace;  
color: wheat;  
}  
</style>
```

Selektor braci używamy, gdy mamy elementy znajdujące się w tym samym rzędzie hierarchii. Umożliwia on nadanie drugiemu z sąsiadujących elementów zdefiniowanych atrybutów formatowania.

```
<style>  
u + i {font-size:10cm}  
</style>
```

Selektory atrybutów:

Prosty selektor atrybutu wykorzystujemy dla elementów o nadającym określonym atrybutem, którego wartość nie ma znaczenia.

```
p [title] {  
border: 2px solid #00ff00;  
width: 100px  
}
```

Selektor atrybutu o określonej wartości określa formatowanie, gdy atrybut ma określoną wartość.

```
p [align="center"] {  
font-family: 'Times New Roman', Times, serif  
}
```

Selektor atrybutu zawierającego określony wyraz określa formatowanie, gdy w wartości atrybutu wystąpi podany wyraz.

```
p [title~= "Witajcie"] {  
font-family: 'Times New Roman', Times, serif  
}
```

Selektory specjalne:

Selektor klasy używamy, gdy chcemy zastosować określony styl formatowania do jednej grupy elementów.

```
<head>
```

```
<style>
p.tekst1 {
font-family: 'Times New Roman', Times, serif
}
</style>
</head>
<body>


Cześć wszystkim!</p>


# Zebraliśmy się dzisiaj,....</h1> </body>


```

Selektor identyfikatora stosuje się do nadawania określonym atrybutów formatowania elementowi, który ma przypisany jednoznaczny identyfikator (id).

```
<style>
#tekst1 {
color: red;
}
</style>
</head>
<body>


Cześć wszystkim!</p>
</body>


```

Pseudoklasy:

Pseudoklasy służą do dynamicznej zmiany stylu elementu, na przykład gdy chcemy, aby coś zmieniło swój wygląd po najechaniu na dany element myszą.

Linki mają właściwości, które określają działanie użytkownika:

- `:active` (link odwiedzany, strona jest aktualnie wczytana)
- `:link` (link nieaktywny, nie został przez użytkownika odwiedzony)
- `:visited` (link odwiedzony, strona była otwierana)
- `:hover` (link gotowy do kliknięcia, kursor myszy ustawiony nad linkiem)

```
<style>
a:link{
color:red;
}
a:visited {
color: blue;
}
</style>
</head>
<body>
Kliknij!
</body>
```

Selektory pseudoelementów:

Selektor `:first-line` nadaje określone formatowanie wszystkim pierwszym liniom znacznika, do którego odnosi się selektor.

```
<style>
p :first-line {
color: blueviolet;
```

```
}
```

```
</style>
```

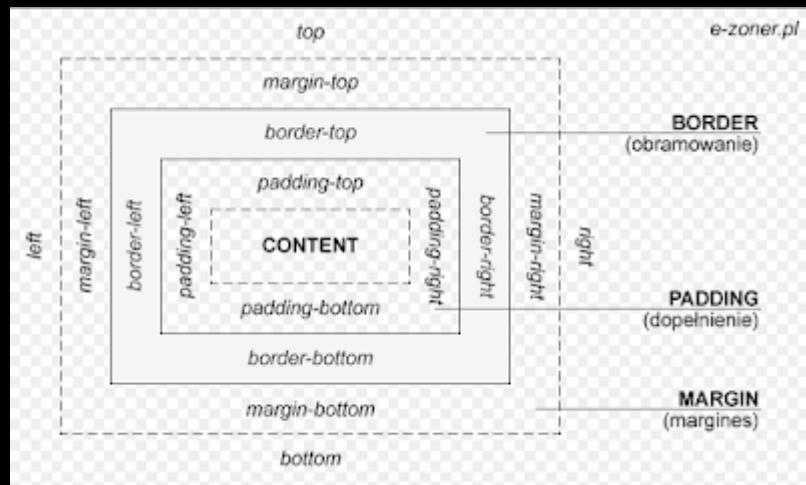
Selektor **:first-letter** nadaje odrębne formatowanie pierwszej literze np. akapitu.

```
<style>
p :first-letter {
color: blueviolet;
}
</style>
```

Właściwości elementów:

- czcionki (np. **font-family**, **font-size**, **font-weight**, **font-style**)
- tekst (np. **line-through**, **overline**, **underline**, **blink**)
- ścio i kolor (np. **repeat**, **no-repeat**, **scroll**, **fixed**, **local**, **background-position: (bottom, left, right, top, center)**, **background-image**)

Model blokowy CSS:



Obramowanie to ramka narysowana wokół elementu. Wykorzystuje się go do dekoracji elementu lub do oddzielenia go od innych elementów. Do ustawienia szerokości obramowania służy atrybut **border-width**. Jeśli podamy **jedną wartość** tego atrybutu to wówczas ta wartość będzie identyczna dla wszystkich krawędzi; **dwie wartości** - taka sama dla krawędzi poziomych i taka sama dla pionowych; **trzy wartości** - pierwsza dla krawędzi górnej, druga do dwóch krawędzi pionowych i trzecia dla krawędzi dolnej; **cztery wartości** - dla krawędzi kolejno: góra, prawa, dolna, lewa. Do definiowania koloru obramowania służy atrybut **border-color**, a do jego stylu **border-style**.

Margines zewnętrzny określa przestrzeń wokół definiowanego elementu. Przestrzeń ta oddziela element od innych elementów strony. Do każdego marginesu można zdefiniować jego odległość (**margin-left**, **margin-right**, **margin-top**, **margin-bottom**).

Margines wewnętrzny to przestrzeń między zawartością elementu, a obramowaniem (np. **padding-top** itd.).

W modelu blokowym można ustalać dokładne **rozmiary różnych elementów** takich jak akapit, tabela, blok itp. Aby zdefiniować szerokość elementu użyjemy atrybutu **width**; do określenia szerokości minimalnej (rozmiar nigdy nie będzie mniejszy od danej wartości) **min-width**; do określenia szerokości maksymalnej - **max-width**; do określenia wysokości - **height**; do określenia wysokości minimalnej - **min-height**; do wysokości maksymalnej - **max-height**.

Styl listy (punktowanej/numerowanej):

Zwykle w przeglądarkach w listach punktowanych wyświetla się okrągły punktor. Za pomocą atrybutu `list-style-type` można zmienić ten punktor. Dostępymi typami punktorów są np. `disc`, `circle`, `square`, `decimal`, `lower-roman`. Punktorem może być także obrazek zapisany w pliku (`list-style-image`). Pozycję punktora względem tekstu można określić poprzez atrybut `list-style-position`.

Tabele:

W celu określenia położenia podpisu dodanego do tabeli używa się atrybutu `caption-side`. Jako wartość ustawiamy: `top`, `bottom`, `left` albo `right`. Do definiowania krawędzi tabeli używamy atrybutu `border`. W celu rozplanowania tabeli na stronie używamy atrybutu `table-layout` (wartości: `auto`, `fixed`). Odstępy między komórkami definiujemy przy pomocy `border-spacing`. Jeśli nie chcemy mieć podwójnego obramowania, wówczas używamy: `border-collapse: collapse`.

Jeśli w CSSie zostały zdefiniowane elementy o tym samym stylu, to można te elementy połączyć w grupy i nadać im styl poprzez wspólną deklarację. Elementy oddzielone są wówczas przecinkami

```
p,h1,u {border-width: 20px }
```

Właściwość `overflow` określa co ma się stać z treścią, która nie mieści się w obszarze wyznaczonym przez zewnętrzną krawędź prawa lub dolnego wewnętrznego marginesu interesującego nas elementu HTML (`visible`, `hidden`, `scroll`, `auto`).

Pozycjonowanie pozwala zdefiniować położenie elementów na stronie internetowej. Elementy można rozmieścić zarówno względem brzegów strony jak i względem innych elementów. Do pozycjonowania elementów służy atrybut `position`. Wyróżniamy:

- **Pozycjonowanie statyczne** - przywraca normalne pozycjonowanie elementu. Używa się gdy, wcześniej podaliśmy np. w arkuszach stylów deklarację pozycjonowania tego typu elementów.
- **Pozycjonowanie względne** - pozwala przesunąć wybrany element w inne miejsce względem położenia pierwotnego. Parametry określają sposób tego przesunięcia.
- **Pozycjonowanie bezwzględne** - pozwala przesunąć wybrany element w inne miejsce względem brzegów strony, bloku albo ramki (gdy został umieszczony wewnętrz). Parametry określają sposób tego przesunięcia.
- **Pozycjonowanie stałe** - działa podobnie do pozycjonowania absolutnego, z tym, że pozycjonowanie stałe ustala elementu zawsze względem krawędzi okna przeglądarki (scrollując element nie rusza się).

```
<h1 style="position: static;">Statyczne</h1>
<p style="position: relative; left: 1cm;">Względne</p>
<h2 style="position: absolute; top 40px;">Bezwzględne</h2>
<div style="position: fixed; left 5cm; color: red;">Pozycjonowanie stałe</div>
```

Gdy nakładamy kilka elementów na siebie, możemy określić, w jaki sposób będą one nakładane. Robimy to za pomocą atrybutu `position`. Nakładającym się elementom nadajemy numer (`z-index`) i stosujemy regułę, że elementy z wyższym numerem są nakładane na te z niższym numerem.

```
<div style="color:silver; position:absolute; top 30px; z-index:1;">Coś</div>
```

Atrybut `vertical-align` pozwala na zdefiniowanie wyrównania elementu w stosunku do innych elementów strony. Do ustawiania elementu względem elementów, które z nim sąsiadują w poziomie, służy atrybut `float`. Właściwość `clear` określa, czy dany element pojawi się obok płynących elementów, które go poprzedzają, czy też ma zostać przesunięty poniżej ich.

Wszystkie definiowane elementy są wyświetlane na stronie w sposób domyślny. Jeśli chcemy zrezygnować z takiego

pokazywania elementu możemy zdefiniować atrybut *display*, a jego parametr będzie określał w jaki sposób dany element strony ma zostać zaprezentowany np.:

- *block* - element wyświetlany w bloku z odstępami od góry i od dołu;
- *inline* - element wyświetlany w linii z pozostałymi elementami;
- *list-item* - element wyświetlany jako pozycja listy listowanej
- *none* - element nie będzie wyświetlany

```
<head>
<style>
.blok {
display: inline-block
}
</style>
</head>
<body>
<h1 class="blok">Witryny internetowe</h1>
<h2 class="blok">Programowanie aplikacji internetowych</h2>
</body>
```

Grafika na stronach internetowych

Grafika wektorowa (obiektowa) to grafika, w której obraz jest tworzony z obiektów podstawowych takich jak figury geometryczne. Każdy z obiektów opisywany jest za pomocą parametrów (np. współrzędne itp.). Obrazy tej grafiki są skalowalne, czyli można zmieniać ich wielkość bez utraty jakości. Formaty plików to np. SVG, CDR. Programami do tworzenia tego typu grafiki są: CorelDRAW, Adobe Illustrator, AutoCAD.

Grafika rastrowa (bitmapowa) to grafika, w której obraz jest tworzony z pikseli. Ten rodzaj grafiki jest najczęściej stosowany do cyfrowego zapisu zdjęć lub obrazów ze skanera. Najczęściej spotykane formaty plików to: BMP, JPG, PNG, GIF, TIFF. Popularnymi programami do tworzenia tego typu grafiki są Paint, GIMP oraz Photoshop.

Parametry obrazów:

- rozdzielcość geometryczna - określa liczbę pikseli na jednostkę długości obrazu
- rozdzielcość radiometryczna - określa jasność piksela
- rozmiar obrazu - fizyczna wielkość obrazu (podana jest szerokość i wysokość)
- głębia koloru - określa liczbę barw możliwych do uzyskania w obrazie
- histogram obrazu - wykres przedstawiający liczbę pikseli o określonej jasności występujących w obrazie

Modele barw:

- **Model RGB** - w tym modelu dowolny kolor można uzyskać składając go z trzech podstawowych kolorów - czerwonego, zielonego i niebieskiego. W wyniku zmieszania tych kolorów, w zależności od natężenia każdej z barw tworzy się określony kolor. Nazywamy to modelem addytywnym. Każda z barw składowych jest zapisywana za pomocą 8 bitów i przyjmuje wartości z zakresu 0-255.
- **Model CMYK** (albo: CMY) - model stosowany przy różnego rodzaju wydrukach. Bazuje on na syntezie subtraktywnej - przez odejmowanie barw z bieli.
- **Model HSV** (albo: HSB) - powstał na podstawie, w jaki barwy są postrzegane przez człowieka. Człowiek zwykle korzysta ze standardowej palety kolorów. Dobierając barwę z palety odpowiedni kolor farby, a dodając kolor biały lub

czarny, przyciemnia lub rozmywa kolor podstawowy. H - określa kolor, S - określa nasycenie koloru, V - jasność koloru.

Formaty plików graficznych:

- **Format BMP** - podstawowy format grafiki rastrowej, każdy piksel opisany jest przez kolor w formacie RGB, umożliwia kompresję danych, bezstratny zapis
- **Format JPEG** - opracowany na potrzeby internetu, duża kompresja, wielokrotny zapis między tym formatem, a innymi prowadzi do zniekształceń ostatecznego obrazu, kompresja stratna jest o regulowanej jakości (24 bpp)
- **Format GIF** - do zapisu prostych obrazów o niewielkiej liczbie kolorów (max 256). wykorzystywany przy projektowaniu nagłówków, przycisków, ikon i prostej grafiki, pozwala na zapisywanie i odtwarzanie prostych animacji, możliwość przezroczystości jedno-bitowej, nie nadaje się do zdjęć
- **Format PNG** - do zapisu prostych plików graficznych, nie obsługuje animacji, bezstratna kompresja, obsługa 8-bitowej przezroczystości (kanał alfa)
- **Format PSD** - format programu Adobe Photoshop, zapisuje wszystkie elementy formatowania tego programu, służy do edytowania wielu warstw, do ich tworzenia lub modyfikowania.
- **Format RAW** - format zapisu obrazów zarejestrowanych na matrycach aparatów fotograficznych tzw. negatyw cyfrowych
- **Format TIFF** - pozwala na zapisywanie obrazów stworzonych w skali szarości oraz w wielu trybach koloru, przechowuje ścieżki i kanał alfa, profile koloru i komentarze tekstowe, kompresja bezstratna do dowolnie dużej głębi (32- lub 48-bitowej) w przestrzeni CMYK i RGB.

Animacja na stronie internetowej

Metody tworzenia animacji:

- animacja poklatkowa - kolejne klatki animacji tworzone są po kolei, każda klatka zawiera pełny obraz
- animacja z zastosowaniem klatek kluczowych - tworzone są wybrane klatki; pozostałe są interpolowane przez komputer na podstawie ramek kluczowych
- skrypty - opisują dynamiczną zmianę właściwości obiektów takich jak położenie, wygląd.

Dźwięk i wideo

Formaty plików audio:

- **WAVE** - dźwięk nie jest poddawany kompresji, duży rozmiar plików, łatwy do edycji
- **MP3** - format kompresji stratnej wykorzystujący niedoskonałości ludzkiego ucha (usunięte są częstotliwości, których człowiek nie słyszy), niewielki rozmiar plików
- **WMA** - format firmy Microsoft, konkurencja dla MP3
- **MIDI** - nie jest zapisywany sam dźwięk, ale informacje o tym jaki dźwięk ma zostać odtworzony, jaka ma być jego częstotliwość oraz jaki ma być instrument odtwarzania
- **AIFF** - dla komputerów Mac, pliki nie są kompresowane, dobra jakość dźwięku, duże rozmiary plików
- **OGG** - alternatywa przy MP3, na licencji GNU, wysoka jakość dźwięku przy niewielkiej objętości, kompresja stratna, zalecany w HTML5
- **FLAC** - kompresja bezstratna, bez utraty jakości, duże pliki

Kompresja stratna to usunięcie z zapisu dźwięków niesłyszalnych lub słabo słyszalnych w celu zmniejszenia rozmiaru pliku. Im większy stopień kompresji tym gorsza jakość. **Kompresja bezstratna** to metoda zmniejszenia objętości pliku z możliwością odtworzenia dźwięku w wersji identycznej z pierwotną. Polega ona na zmianie sposobu zapisu danych.

Do edycji dźwięku służy m.in. program Audacity.

Formaty plików wideo:

- **MPEG** - stratna kompresja międzyklatkowa, stosowana m.in. w DVD
- **MP4** - na strony internetowe
- **DivX, MKV** - stratna kompresja, kontenery zawierające dźwięk i wideo z różnymi formatami
- **MOV**
- **OGG** - na strony internetowe
- **AVI** - opracowany przez Microsoft, do zapisu dźwięku i sekwencji wideo, dane można zapisywać w postaci skompresowanej lub nieskompresowanej

Do edycji filmu służy m.in. program Adobe Premiere.

2. TWORZENIE BAZ DANYCH I ADMINISTROWANIE BAZAMI DANYCH:

Zalety korzystania z komputerowych baz danych:

- szybkie wyszukiwanie informacji
- łatwe wykonywanie obliczeń
- możliwość przechowywania dużej ilości danych na małym obszarze
- szybkie porządkowanie danych

Baza danych to uporządkowany zbiór danych na jakiś temat zorganizowany w sposób ułatwiający do nich dostęp. **System zarządzania bazą danych (SZBD)** to program zarządzający danymi w bazie i umożliwiający ich przetwarzanie. **System bazy danych** to baza danych i SZBD.

Modele baz danych:

- model hierarchiczny (dane przechowywane są w postaci drzewa, informacja zawarta jest w dokumentach)
- model sieciowy (połączenia pomiędzy dokumentami tworzą sieć, informacja jest zawarta w dokumentach oraz w przebiegu połączeń sieci)
- model obiektowy (łączy cechy programów komputerowych tworzonych w językach programowania obiektowego z cechami aplikacji bazodanowych, obiekt w bazie reprezentuje obiekt w świecie rzeczywistym)
- model relacyjny (informacja zapisywana w tabeli w formie wierszy, tabele tworzą ze sobą powiązania - czyli tzw. relacje, najczęściej wykorzystywany)
- model postrelacyjny (bazy danych poszerzone np. o elementy obiektowości)

Model relacyjny wg Coddza:

a) klucz (zbiór atrybutów mających określoną wartość)

- właściwość klucza pozwala jednoznacznie identyfikować krotki
- schemat może posiadać kilka kluczy (podstawowe, obce)
- każdy nadzbiór klucza jest kluczem

b) integralność danych

- integralność encji (każdy schemat relacji posiada klucz główny i żaden element klucza głównego nie może posiadać wartości pustej (NULL))

- integralność referencyjna (każda wartość klucza obcego jest równa wartości klucza właściwego określonej krotki w relacji nadzędnej lub wynosi NULL)
- więzy ogólne (dodatkowe warunki dotyczące poprawności danych określone przez użytkowników lub administratorów baz danych)

c) algebra relacji (zawiera zbiór jawnych operacji pozwalających na tworzenie wymaganych relacji z relacji dostępnych w bazie danych)

Relacyjny model baz danych:

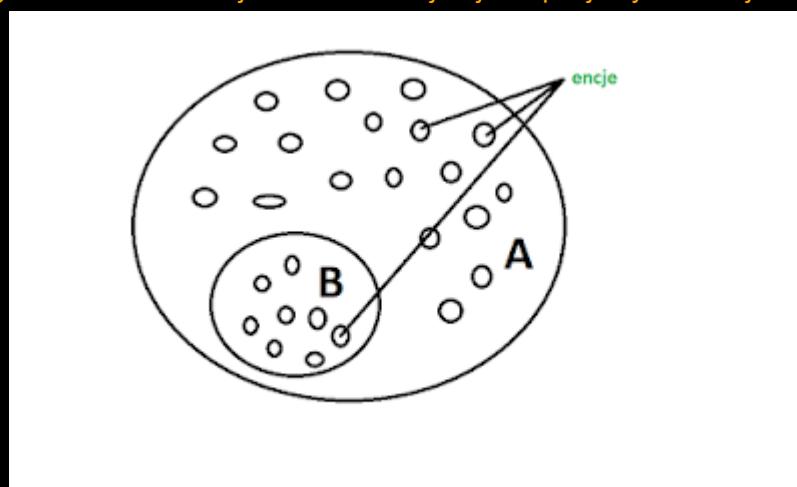
- W tabeli przechowuje się dane
- Podając wartość klucza podstawowego, nazwy tabeli i nazwy kolumny powinno się mieć dostęp do dowolnych danych
- Musi być obsługiwana wartość NULL
- Występuje integralność danych

Klucz podstawowy (primary key) to zestaw atrybutów relacji, który jednoznacznie identyfikuje każdy rekord tej relacji. Nie może on zawierać powtarzających się danych oraz nie może być pusty. Kluczem tym może być także kombinacja pól (np. imię i nazwisko). Często kluczem podstawowym jest **klucz sztuczny**, który jest polem zawierającym unikatowy numer identyfikacyjny nadany w sposób sztuczny każdemu obiekowi, który znajduje się w tabeli (ID). **Klucz obcy** jest kombinacją jednego lub wielu atrybutów tabeli, które wyrażają się w dwóch lub większej liczbie relacji. Wykorzystywany jest do tworzenia relacji pomiędzy dwoma tabelami, gdzie w jednej tabeli ten zbiór atrybutów jest kluczem obcym, a w drugiej kluczem głównym.

Relacja to zdefiniowanie logicznego połączenia między tabelami bazy danych. Wyróżniamy 4 typy relacji:

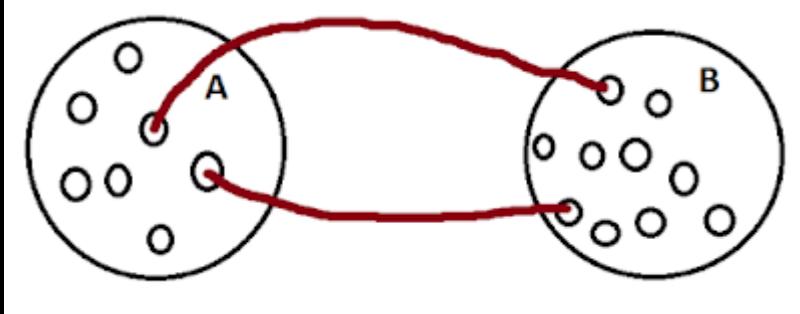
- **Związek "jest"**

Jeśli zbiór encji A stanowi uogólnienie zbioru encji B to zbiór encji B jest specjalnym rodzajem zbioru encji A.



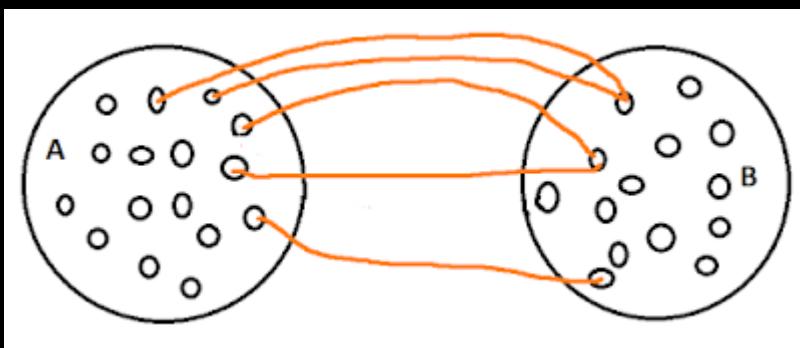
- **Związek jednojednoznaczny ("jeden do jednego")**

Każda encja ze zbioru encji A jest skojarzona z co najwyżej jedną encją ze zbioru B i na odwrót.



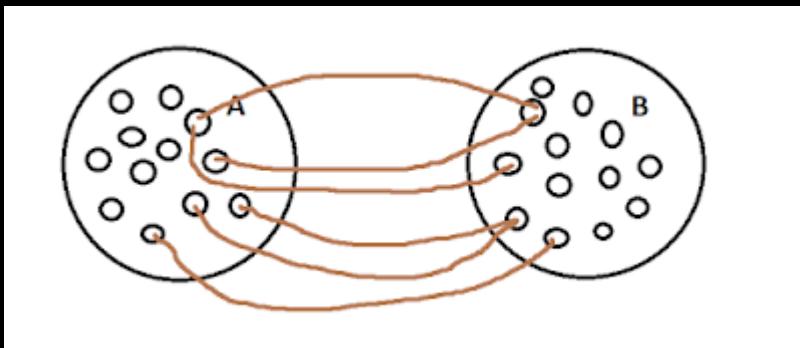
- **Związek jednoznaczny ("wiele do jednego")**

Każda encja ze zbioru encji A jest skojarzona z co najwyżej jedną encją ze zbioru encji B, natomiast jednej encji ze zbioru encji B może odpowiadać wiele encji ze zbioru encji A.



- **Związek wieloznaczny ("wiele do wielu")**

Każdej encji ze zbioru encji A może odpowiadać wiele encji ze zbioru encji B i na odwrót.



Encja to każdy obiekt (przedmiot, stan, zjawisko itp.), który jesteśmy w stanie odróżnić od innych obiektów. Cechy encji nazywamy **atrybutami** (np. jeśli encją jest człowiek, to jego atrybutami jest numer pesel, imię, nazwisko).

Przykład dotyczący planowania projektowania bazy danych

Reguły projektowania tabel:

- Do opisu każdego zbioru podobnych encji stosuje się oddzielną tabelę. Jednej encji odpowiada jeden wiersz. Atrybutowi odpowiada kolumna. Dla każdego atrybutu określa się typ informacji.

- Do opisu każdego dwustronnego związku między encjami można użyć oddzielnej tabeli. Kolumny tabeli są tworzone z kluczy encji należących do związku.
- Zapis związku jeden do jednego lub wiele do jednego może być umieszczony w dodatkowych kolumnach tabel pozostających w związku. W przypadku związku jeden do jednego kolumna ta może znaleźć się w dowolnej tabeli, w przypadku związku wiele do jednego musi znaleźć się w tabeli ze strony "wiele". Dołączona kolumna zawiera klucz encji, z którą zachodzi związek.
- Związek wiele do wielu opisuje się w oddzielnej tabeli, której kolumny tworzone są z kluczy encji należących do związku.
- Jeśli klucze w tabeli opisującej związek składają się z wielu atrybutów lub są za długie, należy zastąpić je kluczami sztucznymi.

Normalizację tabel stosuje się po to, aby sprawdzić, czy zaprojektowane tabele mają prawidłową strukturę. Normalizacja ma prowadzić do:

- zlikwidowania problemu powtarzania danych
- zoptymalizowania objętości bazy
- zoptymalizowania efektywności obsługi bazy
- najmniejszego prawdopodobieństwa wystąpienia błędu podczas wprowadzania danych.

Pierwsza postać normalna:

Tabela jest w pierwszej postaci normalnej, gdy pojedyncze pole tabeli zawiera informację elementarną (nie występuje lista wartości! - czyli jedno pole -> jedna wartość).

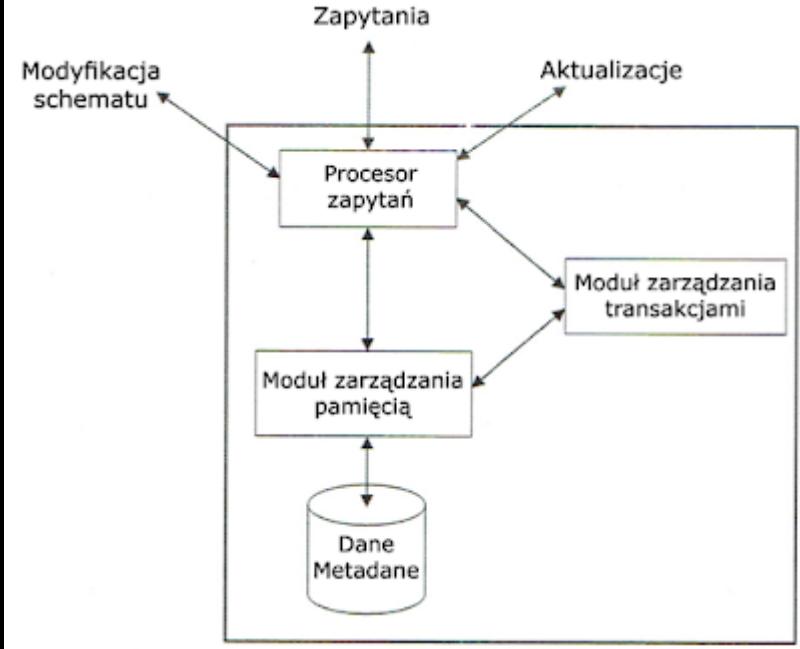
Druga postać normalna:

Tabela jest w drugiej postaci normalnej, gdy jest w pierwszej postaci normalnej oraz, gdy każde z pól niewchodzących w skład klucza podstawowego zależy od całego klucza, a nie od jego części.

Trzecia postać normalna:

Tabela jest w trzeciej postaci normalnej, gdy jest w pierwszej i drugiej postaci normalnej oraz każde z pól niewchodzących w skład klucza podstawowego niesie informację bezpośrednio o kluczu i nie odnosi się do żadnego innego pola.

Systemy zarządzania bazą danych służą do manipulowania i aktualizowania danych, które są zgromadzone w systemach komputerowych. Pozwalają one prowadzić operacje na dużych i bardzo dużych zbiorach danych oraz na zarządzanie złożonymi strukturami. Interakcja aplikacji z bazą danych odbywa się zwykle przy pomocy języka SQL.



W powyższym schemacie systemu zarządzania bazą danych:

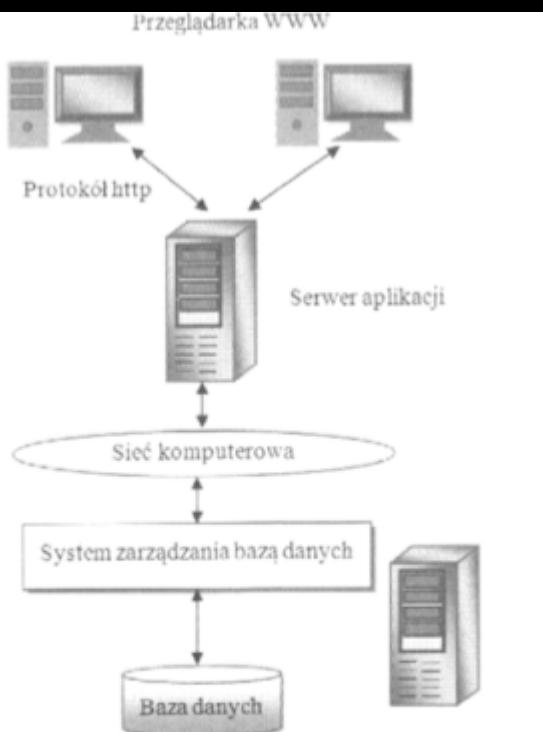
- moduł zarządzania pamięcią przechowuje informacje o miejscu zapisania plików bazy danych na dysku oraz obsługuje pamięć operacyjną
- procesor zapytań przekształca zapytanie lub operację języka zapytań w ciąg poleceń żądających określonych danych
- moduł zarządzania transakcjami kontroluje poprawność i kompletność wykonania wszystkich transakcji

Sposoby komunikacji z bazą danych:

- architektura klient-serwer (aplikacje zainstalowane na stacjach użytkowników komunikują się z bazą danych, wykorzystując sieciowe oprogramowanie dedykowane do komunikacji z systemem zarządzania bazą danych)
- architektura trójwarstwowa (pomiędzy użytkownikami, a serwerem bazy danych znajduje się serwer aplikacji udostępniający umieszczone na nim aplikacje)



Rysunek 3.2. Architektura komunikacyjna klient-serwer



Rysunek 3.3. Architektura komunikacyjna 3-warstwowa

Cechy dobrze zrobionej bazy danych:

- trwałość danych (dane zapisane są w sposób nieulotny)
- integralność danych (poprawność danych - dane muszą być rzeczywiste, spełniać ograniczenia nałożone przez użytkowników, odporne na błędy i awarie oraz na błędy użytkowników)
- bezpieczeństwo danych (dostęp do bazy mają tylko użytkownicy identyfikowani loginem i hasłem)
- współdzielenie danych (równoczesna praca wielu użytkowników)
- abstrakcja danych (struktura bazy powinna w poprawny sposób odzwierciedlać obiekty świata rzeczywistego oraz powiązania pomiędzy tymi obiektami)
- niezależność danych (oddzielenie danych od aplikacji, które używają tych danych)
- integracja danych (zapewnienie współpracy danych przechowywanych w różnych miejscach i obsługiwanych przez różne systemy)

Język SQL realizuje trzy podstawowe typy zadań:

- Instrukcje DDL (definiowanie danych oraz tworzenie, modyfikowanie i usuwanie obiektów bazy danych)
- Instrukcje DML (manipulowanie danymi, odczyt i modyfikacja danych)
- Instrukcje DCL (kontrola dostępu do danych, modyfikacja uprawnień użytkowników)

Typy danych w języku SQL:

Dane tekstowe	char, varchar, nchar, ntext, nvarchar
Liczbowe	int, smallint, bigint, tinyint, float, real, decimal, numeric
Data i czas	datetime, smalldatetime
Binarne	binary, varbinary
Waluta	money, smallmoney
Specjalne	text, image, xml, bit

Podstawowe instrukcje języka SQL:

- **SELECT nazwa_kolumny1, nazwa_kolumny2 FROM nazwa_tabeli;** - wyświetla wszystkie dane z wybranych kolumn znajdujących się w danej tabeli (podstawowa instrukcja)
- **SELECT DISTINCT nazwa_kolumny1, nazwa_kolumny2 FROM nazwa_tabeli;** - wyświetla dane z wybranych kolumn znajdujących się w danej tabeli, wyniki nie powtarzają się
- **SELECT nazwa_kolumny1, nazwa_kolumny2 FROM nazwa_tabeli WHERE warunek;** - wyświetla dane z wybranych kolumn znajdujących się w danej tabeli spełniających dany warunek
- Operatory **AND** (oraz), **OR** (lub), **NOT** (nie są) używamy w zapytaniach, w których chcemy podać kilka warunków
- **ORDER BY** - sortowanie w sposób alfabetyczny według danej kolumny (**ASC** - od a do z, **DESC** od z do a)
- **INSERT INTO nazwa_tabeli VALUES (wartość_kolumny1, wartość_kolumny2);** - instrukcja dodająca dane do danej tabeli
- **IS NULL / IS NOT NULL** - sprawdzanie, które rekordy mają wartość NULL / nie mają wartości NULL
- **UPDATE nazwa_tabeli SET nazwa_kolumny1 = nowa_wartość_kolumny1 WHERE warunek;** - modyfikowanie danych będących już w tabeli
- **DELETE FROM nazwa_tabeli WHERE warunek;** - usuwanie rekordów z tabeli, które spełniają podany warunek
- **SELECT TOP/LIMIT/ROWNUM** - wybiera pierwszych n rekordów z tabeli (liczbę n podajemy w instrukcji)
- **MIN()** oraz **MAX()** - funkcja zwracająca najmniejszą / największą wartość z wybranych kolumn
- **COUNT()** - funkcja zwracająca liczbę rekordów, które spełniają ewentualne kryteria podane w warunku
- **AVG()** - funkcja zwracająca średnią arytmetyczną z rekordów, które spełniają ewentualne kryteria podane w warunku
- **SUM()** - funkcja zwracająca sumę wartości rekordów, które spełniają ewentualne kryteria podane w warunku

- **LIKE** - funkcja szukająca rekordów, które spełniają specyficzny warunek (taki jak "szukaj wyrazy zaczynające się na daną literę np. `SELECT nazwa_kolumny1, nazwa_kolumny2 FROM nazwa_tabeli WHERE nazwa_kolumny1 LIKE 'a%';`)
- **IN()** - operator podawany w warunku (skrócenie zwielokrotnionego operatora OR), w nawiasie podaje się wartości rekordów, które mają zostać wyszukane
- **BETWEEN wartość1_kolumny AND wartość2_kolumny;** - operator podawany w warunku, pokazuje wartości, które mogą być w podanym zakresie, włącznie z wartościami granicznymi (np. pomiędzy 2000, a 2004, pokaże także wyniki z 2000 i 2004)
- **SELECT nazwa_kolumny1 AS nazwa_aliasu FROM nazwa_tabeli;** - wybiera wszystkie rekordy z kolumny1 nazywając kolumnę wynikową nazwa_aliasu
- **SELECT nazwa_tabeli1.nazwa_kolumny1, nazwa_tabeli2.nazwa_kolumny2 FROM nazwa_tabeli1 INNER JOIN nazwa_tabeli2 ON nazwa_tabeli1.nazwa_klucza_głównego_w_tabeli1 = nazwa_tabeli2.nazwa_klucza_głównego_w_tabeli2;** (gdzie klucz główny w tabeli1 jest kluczem obcym w tabeli2)
- łączenie danych z kilku tabel (**INNER JOIN / LEFT JOIN / RIGHT JOIN / FULL JOIN / SELF JOIN**)
- **SELECT zapytanie1 UNION SELECT zapytanie2;** - pokazuje wszystkie elementy, które są wynikiem pierwszego zapytania oraz drugiego zapytania
- **GROUP BY** - grupowanie wierszy według określonej kolumny
- **HAVING** - filtrowanie całych grup rekordów
- **CASE** - tworzenie warunku "co ma się stać gdy..."
- Komentarze w języku SQL (-- oraz /* ... */)
- Daty w języku SQL
- **REPAIR TABLE nazwa_tabeli** - naprawa uszkodzonej bazy danych (QUICK - naprawa tylko pliku indeksu)

Instrukcje SQL dotyczące tabel baz danych:

- **CREATE DATABASE nazwa_bazy;** - tworzenie bazy o określonej nazwie
- **DROP DATABASE nazwa_bazy;** - usuwanie określonej bazy danych
- **BACKUP DATABASE nazwa_bazy TO DISK 'ścieżka_do_miejsca_na_dysku';** - tworzenie kopii zapasowej bazy wskazując ścieżkę, gdzie ma się ona zapisać
- **CREATE TABLE (...);** - tworzenie tabeli, w nawiasie podajemy nazwę kolumny oraz typ danych, możemy także określić klucz podstawowy (**PRIMARY KEY**), klucz obcy (**FOREIGN KEY**), wartość **NOT NULL**, wartość unikatowa **UNIQUE**, indeksowanie - **INDEX**, wartość domyślną - **DEFAULT**, sprawdzanie określonego warunku - **CHECK**, autonumerowanie - **AUTO INCREMENT**)
- **DROP TABLE nazwa_tabeli;** - usuwanie tabeli z bazy danych
- **ALTER TABLE nazwa_tabeli operacja;** - operacje na tabeli takie jak dodawanie kolumny (**ADD nazwa_kolumny**), usuwanie kolumny (**DROP nazwa_kolumny**), modyfikacja kolumny (**ALTER lub MODIFY nazwa_kolumny**)

Polecenia DCL (tworzenie użytkownika, dodawanie uprawnień itd.):

Tworzenie użytkownika: **CREATE USER nazwa IDENTIFIED BY haslo;**

Tworzenie użytkownika z określonymi uprawnieniami:

GRANT prawa ON tabela TO nazwa_użytkownika;

Prawa, czyli prawa dostępu do wykonania określonych poleceń przez użytkownika np. **SELECT, INSERT, UPDATE, DELETE, REFERENCE, CREATE, DROP**. Jeśli natomiast ma mieć wszystkie te prawa, wówczas: **ALL PRIVILEGES**.

np. **GRAND SELECT, INSERT ON pracownicy TO szef;**

Odbieranie wcześniej nadanych uprawnień wygląda w ten sposób:

REVOKE prawa ON tabela *nazwa_ użytkownika*;

Inne polecenia:

- **DROP USER** *nazwa_ użytkownika*; - usuwa użytkownika
- **ALTER USER** *nazwa_ użytkownika IDENTIFIED BY nowe_ hasło*; - zmiana hasła użytkownika

Typy tabel w MySQL:

- MyISAM (domyślny typ, szybkie odczytywanie danych, nie obsługują transakcji)
- MEMORY (tabele przechowywane są w pamięci operacyjnej, tracone są bezpowrotnie po wyłączeniu serwera, traktuje się je jako tabele tymczasowe)
- FEDERATED (definiują dane znajdujące się na innym serwerze)
- BLACKHOLE (dane zapisywane na tabelach nie są przechowywane, zapisywane wiersze są automatycznie usuwane)
- CSV (przechowywanie danych w plikach tekstowych typu CSV, używane do importowania i eksportowania danych pomiędzy serwerem MySQL, a innymi serwerami)
- ARCHIVE (przechowywanie dużej ilości danych, przechowywane dane są bez indeksów oraz są kompresowane przed umieszczeniem w tabeli, zmiana lub usunięcie danych jest niedozwolona)
- InnoDB (tabele obsługujące transakcje, MySQL automatycznie blokuje odczytywanie i modyfikowane wiersze tych tabel)

3. TWORZENIE APLIKACJI INTERNETOWYCH:

Podstawowe pojęcia:

- **Program** - zbiór poleceń zapisanych w jakimś języku programowania zgodnie ze składnią obowiązującą w tym języku.
- **Programowanie** - proces tworzenia i testowania programu.
- Język programowania - to zestaw zasad, które określają kiedy ciąg symboli tworzy dany program oraz jakie obliczenia opisuje.
- **Kod źródłowy** - ciąg instrukcji i deklaracji zapisany w języku programowania opisujący operacje, jakie powinien wykonać komputer.
- **Translator** - program służący do tłumaczenia programu zapisanego w języku programowania z postaci źródłowej do postaci wynikowej.
- **Kompilator** - program służący do tłumaczenia kodu napisanego w języku źródłowym na odpowiadający mu kod w języku wynikowym.
- **Interpreter** - program analizujący kod źródłowy instrukcję po instrukcji i każdy przeanalizowany fragment kodu wykonuje na bieżąco.
- **Aplikacja** - program użytkowy wykonujący konkretne zadania i oferujący interfejs użytkownika.
- **Aplikacja internetowa** - program komputerowy, który pracuje na serwerze i komunikuje się z użytkownikiem poprzez sieć komputerową z wykorzystaniem przeglądarki internetowej.
- **Skrypt** - program napisany w języku skryptowym, który jest wykonywany wewnętrz aplikacji
- **Język skryptowy** - język programowania służący do wykonywania wyspecjalizowanych czynności.

Algorytm to zestaw ściśle określonych czynności, prowadzących do wykonywania pewnego zadania. Określa on sposób rozwiązania pewnych problemów.

Sposoby przedstawienia algorytmów (wraz z przykładami):

- opis słowny (w zapisie słownym są zapisane operacje, które należy wykonywać)
- lista kroków (w krokach zapisane są każde kolejne wykonywane czynności)
- pseudokod (opis słowny przypominający zapis kroków algorytmu, może zawierać instrukcje z języka programowania)
- drzewo algorytmu (reprezentacja graficzna algorytmu; korzeń stanowi początek algorytmu, gałęzie są reprezentacjami wykonywanych operacji, liście to wyniki końcowe)
- schemat blokowy (operacje, które należy wykonać, są przedstawione w postaci graficznej z użyciem symboli)

Tabela 1.1. Symbole wykorzystywane do tworzenia schematów blokowych

Symbol	Opis
	Początek algorytmu, start programu. Od tego miejsca rozpoczyna się wykonywanie operacji.
	Koniec algorytmu, zakończenie programu. W tym miejscu następuje zakończenie wykonywania operacji.
	Połączenie między blokami. Wskazuje kolejność wykonywania operacji.
	Wykonanie operacji, blok obliczeniowy. Wewnątrz tego symbolu znajdują się operacje do wykonania.
	Wprowadzanie danych. Wewnątrz tego symbolu określamy dane wejściowe, które muszą zostać wczytane.

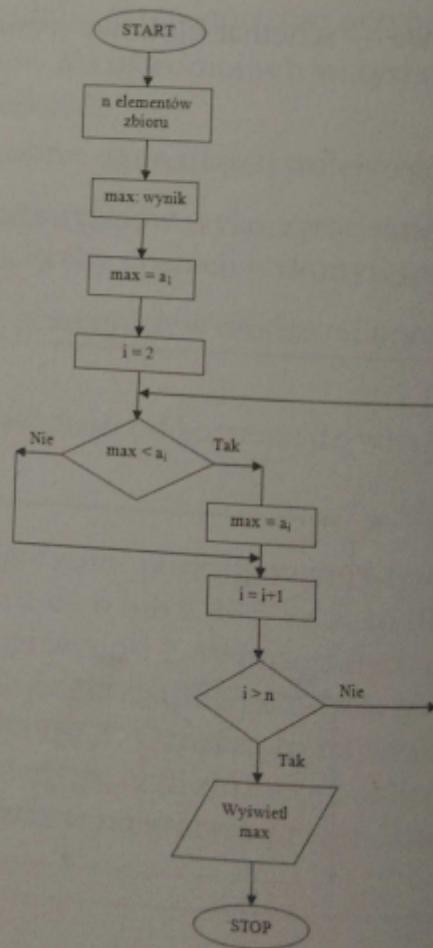
Symbol	Opis
	Wyprowadzanie danych. Wewnątrz tego symbolu określamy dane wyjściowe, które powinny zostać wyprowadzone jako wynik.
	Warunek logiczny, blok decyzyjny. Umożliwia tworzenie rozgałęzień w algorytmie. Jeżeli warunek jest spełniony, to następuje przejście do gałęzi oznaczonej „Tak”, w przeciwnym razie następuje przejście do gałęzi oznaczonej „Nie”.
	Proces wstępnie zdefiniowany. Symbol ten oznacza dołączenie podprogramu.
	Łącznik. Odwołanie na stronie. Służy do oznaczenia miejsc łączenia schematu, np. gdyby linie łączące na schemacie musiały się krzyżować.
	Łącznik międzystronicowy. Służy do oznaczenia miejsc łączenia schematu, gdy nie mieści się on na jednej stronie.

Specyfikacja algorytmu powinna zawierać:

- podanie danych wejściowych
- określenie wyniku, który algorytm powinien wygenerować
- określenie warunków, jakie powinny spełniać dane wejściowe i wyniki
- podanie zmiennych pomocniczych niezbędnych do realizacji algorytmu

Przykład 1.11

Znajdowanie największego elementu w zbiorze nieuporządkowanym — schemat blokowy (rysunek 1.8).



Rysunek 1.8.

Schemat blokowy znajdowania największego elementu w zbiorze nieuporządkowanym

24

Paradygmat programowania definiuje sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego. Paradygmaty programowania opisują m.in. programowanie:

- strukturalne (następuje dzielenie kodu na bloki; instrukcje są pobierane ze stosu i są wykorzystywane z wykorzystaniem pętli i instrukcji warunkowych)
- obiektowe (programy definiowane są za pomocą obiektów; program napisany zgodnie z zasadami programowania obiektowego składa się ze zbioru obiektów, które komunikują się między sobą, aby wykonać określone zadania)
- proceduralne (program dzielony jest na procedury, czyli fragmenty wykonujące ścisłe określone operacje; nie ma zmiennych globalnych)
- uogólnione (kod programu powstaje bez wcześniejszej znajomości typów danych, na których będzie pracował)

W programowaniu obiektowym program to zbiór powiązanych obiektów, które na siebie oddziałują. Obiekt jest elementem, który jest opisywany przez właściwości i funkcje. Klasa jest specjalną strukturą zawierającą grupę powiązanych ze sobą obiektów, która definiuje metody (funkcjonalność) oraz atrybuty wybranych obiektów. Obiekt jest elementem danej klasy. Programowanie obiektowe opiera się na tworzeniu programów, które przedstawiają świat rzeczywisty i relacje w nim zachodzące za pomocą obiektów.

Narzędzia stosowane podczas pracy z aplikacją internetową można podzielić na dwie grupy:

- Narzędzia po stronie klienta (pozwalały klientowi wykonywać część zadań związanych z obsługą aplikacji np. JavaScript, Java, AJAX)
- Narzędzia po stronie serwera (pozwalały wykonywać na serwerze zadania służące do obsługi żądań klienta (np. PHP, Perl, Ruby on Rails))

Do tworzenia aplikacji internetowych często wykorzystuje się platformy programistyczne definiujące strukturę aplikacji, określają mechanizm ich działania oraz dostarczając bibliotek umożliwiających wykonywanie określonych zadań. **Framework** posiada zdefiniowaną podstawową strukturę aplikacji, czyli elementy, które pozostają niezmienne we wszystkich utworzonych przy jego pomocy aplikacjach.

JavaScript:

Kod źródłowy napisany w języku JavaScript może być umieszczony wewnątrz dokumentu HTML między znacznikami `<script>` i `</script>` zarówno w sekcji `<head>`, jak i `<body>`. Może być także umieszczony w **oddzielnym dokumencie**.

Wyświetlanie danych w języku JavaScript:

- `innerHTML` (ustawia lub pobiera zbiór zawartych w danym elemencie znaczników razem z ich treścią)
- `document.write` (umieszcza na stronie WWW dany tekst)
- `window.alert` (wyświetla dany tekst w wyskakującym okienku z przyciskiem "OK")
- `console.log` (wyświetla dany tekst w konsoli przeglądarki)

Zmienne w JavaScript definiuje się za pomocą słowa kluczowego `var`. Można także używać nowszych: `let` (zmienna) oraz `const` (stała).

Działania arytmetyczne, sklejanie ciągów znaków są opisane bliżej [tutaj](#).

Komentarze są ignorowane w trakcie przetwarzania kodu. Stosuje się je głównie do opisywania co dany blok kodu robi (dla ułatwienia przy ewentualnej późniejszej edycji kodu).

```
<script>
// to jest komentarz w języku JavaScript
/* to też może
być komentarz w JS, ale rozbito
na kilka linijek */
</script>
```

Operatory w języku JavaScript:

Tabela 3.1. Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	$a + b$
-	odejmowanie	$a - b$
*	mnożenie	$a * b$
/	dzielenie	a / b
%	modulo (reszta z dzielenia)	$a \% b$
++	inkrementacja	$x++, ++x$
--	dekrementacja	$x--, --x$

Inkrementacja powoduje zwiększenie danej liczby o jeden. Dekrementacja zmniejsza daną liczbę o jeden.

Tabela 3.2. Operatory porównania

Operator	Działanie	Przykład
==	Wynik true, gdy argumenty są równe.	$a == b$
!=	Wynik true, gdy argumenty są różne.	$a != b$
====	Wynik true, gdy argumenty są tego samego typu i są równe.	$a === b$
!==	Wynik true, gdy argumenty są różne lub są różnych typów.	$a != b$
>	Wynik true, gdy argument pierwszy jest większy od drugiego.	$a > b$
<	Wynik true, gdy argument pierwszy jest mniejszy od drugiego.	$a < b$
>=	Wynik true, gdy argument pierwszy jest większy od drugiego lub jest mu równy.	$a >= b$
<=	Wynik true, gdy argument pierwszy jest mniejszy od drugiego lub jest mu równy.	$a <= b$

Tabela 3.3. Operatory bitowe

Operator	Działanie	Przykład
&	iloczyn bitowy (AND)	a & b
	suma bitowa (OR)	a b
~	negacja bitowa (NOT)	~a
^	bitowa różnica symetryczna	a ^ b
>>	przesunięcie bitowe w prawo	
<<	przesunięcie bitowe w lewo	a >> n
>>>	przesunięcie bitowe w prawo z wypełnieniem zerami	a >>> n

Tabela 3.4. Operatory logiczne

Operator	Działanie	Przykład
&&	iloczyn logiczny (AND)	a && b
	suma logiczna (OR)	a b
!	negacja logiczna (NOT)	! a

Tabela 3.5. Niektóre operatory przypisania

Operator	Przykład	Znaczenie
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Okna dialogowe w JavaScript:

- **alert** (wyświetla komunikat i przycisk "OK")
- **confirm** (wyświetla komunikat oraz przyciski "OK" i "Anuluj")
- **prompt** (wyświetla komunikat i pole do wprowadzenia danych)

parseInt przetwarza argument w postaci łańcucha znaków i zwraca liczbę całkowitą typu integer, o zadanej podstawie. **parseFloat** zmienia wartość stringa na typ float (zmiennoprzecinkowy).

Instrukcje warunkowe pozwalają na wykonywanie różnych instrukcji w zależności od tego czy zdefiniowane wyrażenie logiczne jest prawdziwe, czy fałszywe.

```
x = 40
if (x=10){
  console.log('x jest równe 10')
} else {
  console.log ('x nie jest równe 10')
}
```

Switch sprawdza, czy pewna wartość jest równa innym. Używa się w nim zwykle **break** lub **continue**.

```
var expr = 'Mandarynki';
switch (expr) {           // bierzemy pod lupa to co jest w zmiennej
case 'Pomarańcze':    // jeśli w zmiennej byłyby pomarańcze wtedy ten kod się wykona
  console.log('Pomarańcze są po 3 zł za kilogram.');
```

```

break;           // break przerwuje aktualne przejście bloku kodu
case 'Jabłka':          // jeśli jabłka lub mandarynki
case 'Mandarynki':       // to wykona się ten kod
console.log('Jabłek i mandarynek nie sprzedajemy.');
break;           // break przerwuje wykonywanie dalszej części kodu
default:          // jeśli cokolwiek innego będzie w zmiennej wtedy ten kod
console.log('Nasz sklep sprzedaje tylko pomarańcze. Wypad!');
}

```

Pętla umożliwia cykliczne wykonywanie ciągu instrukcji przez określoną liczbę razy, do momentu zajścia pewnych warunków. W JavaScript posiadamy pętle:

- while
- do-while
- for

```

// przykład z pętlą While:
var n = 0;
var x = 0;
while (n<3){ // dopóki n jest mniejsze od trzech
n++      // zwiększaj n o jeden
x += n    // x = x + n
}
alert(x)      // wyświetl x w okienku typu alert

// przykład z pętlą For:
for (var i=0; i<5; i++){ //pętla wykona się 5 razy
console.log(i);        // w konsoli pokażą się kolejno liczby od 0 do 4
}

```

Funkcja w języku JavaScript to jedno lub więcej poleceń zgrupowanych w całość. Instrukcja **return** powoduje przerwanie wykonywania poleceń funkcji i powrót do miejsca, z którego funkcja została wywołana.

```

var a = 4;
var b = 7;
function suma(a,b){
var c = a + b;
return c;
}

```

JavaScript jest językiem obiektowym. Oferuje on wbudowane obiekty oraz umożliwia definiowanie własnych obiektów. **Obiekt** to specjalny rodzaj danych posiadający metody i właściwości. Właściwości to wartości związane z obiektem, a metoda to działania, które mogą być wykonane na obiekcie.

nazwa_zmiennej.właściwość -> korzystanie z właściwości obiektu

nazwa_zmiennej.nazwa_metody -> korzystanie z metod

Przykładowe obiekty:

1. string (obiekt reprezentujący fragmenty tekstu)

```

var str = "Hello World!";

```

```
var n = str.length; // obliczy ile znaków włącznie ze spacją posiada dany string
```

Obiekt string stanowi każdy ciąg znaków ujęty w znakach cudzysłowu lub apostrofu.
Właściwości obiektu string:

Właściwości	Opis
length	Zwraca wartość liczbową charakteryzującą liczbę znaków w łańcuchu

Metody obiektu string:

Metody	Opis
big()	Zwiększa rozmiar czcionki; odpowiednik znacznika <big>
blink()	Tekst migający; odpowiednik znacznika <blink>
bold()	Tekst pogrubiony; odpowiednik znacznika
fixed()	Odpowiednik znacznika <tt>
italics()	Tekst pochylony; odpowiednik znacznika <i>
small()	Zmniejsza rozmiar czcionki; odpowiednik znacznika <small>
sub()	Odpowiednik znacznika <sub>
strike()	Tekst przekreślony; odpowiednik znacznika <strike>
sup()	Odpowiednik znacznika <sup>
fontColor(kolor)	Ustawia kolor czcionki na wartość kolor
fontSize(rozmiar)	Ustawia rozmiar czcionki na wartość rozmiar

2. **date** (służy do

przechowywania informacji o dacie i godzinie oraz wykonywaniu na nich określonych operacji)

Obiekt date pozwala na wykonanie operacji z wykorzystaniem daty i czasu. Pozwala na uzyskanie aktualnej wartości daty i czasu, na korzystanie z ich składowych oraz niezależną zmianę każdej z nich. Praca z obiektem **date** uzależniona jest od użycia konstruktora. Może to być konstruktor bezparametryowy:

var data_czas=new Date();

lub konstruktor mający od jednego do siedmiu parametrów (rok, miesiąc, dzień, godzina, minuty, sekundy, milisekundy)

Metody	Opis
getDay()	Zwraca dzień tygodnia
getDate()	Zwraca dzień miesiąca
getHours()	Zwraca wartość reprezentującą godzinę
getMinutes()	Zwraca wartość reprezentującą minuty
getMonth()	Zwraca wartość reprezentującą numer miesiąca
getSeconds()	Zwraca wartość reprezentującą sekundy
getTime()	Zwraca wartość numeryczną określającą czas; w milisekundach
getYear()	Zwraca rok
setDate()	Ustawia dzień miesiąca
setHour()	Ustawia godzinę
setMinutes()	Ustawia minutę
setMonth()	Ustawia miesiąc

3. **array** (do zapamiętywania tablicy, czyli zbioru wartości pod jedną zmienną)

```
var tablica1 = [] // stworzenie pustej tablicy  
var tablica2 = [1,3,4,10] // stworzenie tablicy z zawartością  
tablica2.length // zwróci ile elementów znajduje się w danej tablicy
```

Przydatne metody do stosowania w arrayach:

- `.push()` dodaje jeden lub kilka elementów na koniec tablicy
- `.pop()` usuwa ostatni element tablicy i zwraca tę wartość
- `.shift()` usuwa element z początku i wraca jego wartość
- `.unshift()` wstawia dany element na początek tabeli
- `.sort()` - sortuje danych w tablicy
- `.reverse()` - odwraca kolejność w tablicy
- `.slice(A,B)` - zwraca wycinek tablicy z przedziału od A do B (ale bez B)

4. `boolean` - używa się do konwersji dowolnych treści na wartości logiczne (true, false)

5. `math` - umożliwia wykonywanie podstawowych operacji matematycznych (logarytmy - `math.log(x)`, potęgowanie - `math.pow(x,y)`, pierwiastki - `math.sqrt(x)` itp.)

W programowaniu słowo `this` wskazuje na obiekt będący kontekstem wykonania.

Aby działać na elementach strony, musimy je wcześniej pobrać. Do odwoływania się do jakiegoś elementu skorzystamy z jednej z kilku metod:

- `getElementById(id)` - pobiera jeden element o danym id
- `getElementsByName(nazwa_tagu)` - pobiera elementy o danym znaczniku
- `getElementsByClassName(nazwa_klasy)` - pobiera elementy o danej klasie
- `querySelector(css_selector)` - pobiera pierwszy element pasujący do selektora css
- `querySelectorAll(css_selector)` - pobiera elementy pasujące do selektora css

jQuery to biblioteka JavaScript zawierająca różnorodne funkcje, przy których można realizować animacje, obsługę zdarzeń, tworzenie efektów wizualnych na stronie internetowych. Zwykle jQuery na egzaminie się nie pojawia, więc jeśli ktoś jest bardziej zainteresowany tematem to odsyłam tutaj.

PHP:

Skryty w PHP są umieszczane w kodzie HTML i wykonywane po stronie serwera. Klient otrzymuje wynik wykonania skryptu. Oprogramowanie PHP umożliwia przetwarzanie danych z formularzy, dynamiczne generowanie zawartości stron internetowych, wysyłanie i odbieranie cookies, pozwala na dynamiczne tworzenie obrazów.

Instrukcja `echo` pozwala na wyświetlanie danych. Wyświetlany ciąg znaków zawsze musi być ujęty w cudzysłowy lub apostrofy. Podobnie działa instrukcja `print`.

```
<?php  
echo "Witaj w świecie!"; // wyświetli "Witaj w świecie!"  
print "Witaj w świecie!"; // wyświetli "Witaj w świecie!"  
  
$a = "Świecie!"; // tworzymy zmienną  
echo "Witaj w $a"; // wyświetli "Witaj w Świecie!"  
print "Witaj w $a"; // wyświetli "Witaj w Świecie!"
```

```
echo 'Witaj w $a'; // wyświetli "Witaj w $a"
print 'Witaj w $a'; // wyświetli "Witaj w $a"

$b = 1;
$c = 4;

echo $b+$c; // wyświetli wynik działania (tak samo z print)
echo "$b+$c"; // wyświetli "4+1" (tak samo z print)
echo '$b+$c' // wyświetli "$b+$c"
?>
```

```
<?php
echo "Linia1";
echo "Linia2"; // wyświetli się dokładnie: "Linia1Linia2"

echo "Linia1\n";
echo "Linia2\n"; // wyświetli się dokładnie "Linia1 Linia2"

echo "Linia1</br>";
echo "Linia2</br>"; // wyświetli się "Linia 1" w pierwszej linijce,
// "Linia2" w drugiej linijce
?>
```

Aby stworzyć **zmienną** w języku PHP musimy nazwę zmiennej zawsze rozpoczęć od znaku dolara (\$). Nie może się ona zaczynać cyfrą lub znakiem podkreślenia.

Funkcje na stringach:

- **strlen** - wyświetla długość stringa
- **str_word_count** - wyświetla ilość wyrazów
- **strrev** - wyświetla string od końca do początku
- **str_replace** - podmiana fragmentów tekstu na inny tekst
- **trim / ltrim / rtrim** - usuwa białe znaki z lewej i / lub prawej strony
- **explode** - pozwala rozbić ciąg znaków i zwrócić go do tablicy
- **implode** - funkcja odwrotna do explode
- **strtolower / strtoupper** - zmiana całego stringu na małe/wielkie litery
- **substr** - zwraca część stringa (podajemy od którego, do którego znaki mają zostać wycięte)

Tabela 7.2. Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	\$a + \$b
-	odejmowanie	\$a - \$b
*	mnożenie	\$a * \$b
/	dzielenie	\$a / \$b
%	dzielenie modulo (reszta z dzielenia)	\$a % \$b

Tabela 7.3. Operatory porównania

Operator	Działanie	Przykład
==	Wynik true, gdy argumenty są równe.	\$a == \$b
!=	Wynik true, gdy argumenty są różne.	\$a != \$b
====	Wynik true, gdy argumenty są tego samego typu i są równe.	\$a === \$b
!==	Wynik true, gdy argumenty są różne lub są różnych typów.	%a != \$b
>	Wynik true, gdy argument pierwszy jest większy od drugiego.	\$a > \$b
<	Wynik true, gdy argument pierwszy jest mniejszy od drugiego.	\$a < \$b
>=	Wynik true, gdy argument pierwszy jest większy od drugiego lub mu równy.	\$a >= \$b
<=	Wynik true, gdy argument pierwszy jest mniejszy od drugiego lub mu równy.	\$a <= \$b
<>	Wynik true, gdy argumenty są różne.	\$a <> \$b

Tabela 7.4. Operatory bitowe

Operator	Działanie	Przykład
&	iloczyn bitowy (AND)	\$a & \$b
	suma bitowa (OR)	\$a \$b
~	negacja bitowa (NOT)	~\$a
^	bitowa różnica symetryczna	\$a ^ \$b
>>	przesunięcie bitowe w prawo	\$a >> n
<<	przesunięcie bitowe w lewo	\$a << n

Tabela 7.5. Operatory logiczne

Operator	Działanie	Przykład
and	iloczyn logiczny	\$a and \$b
&&	iloczyn logiczny	\$a && \$b
or	suma logiczna	\$a or \$b
	suma logiczna	\$a \$b
!	negacja logiczna (NOT)	! \$a
xor	różnica symetryczna	\$a xor \$b

Tabela 7.6. Niektóre operatory przypisania

Operator	Przykład	Znaczenie
=	\$x = 10	\$x = 10
+=	\$x += 5	\$x = \$x + 5
-=	\$x -= 5	\$x = \$x - 5
*=	\$x *= 5	\$x = \$x * 5
/=	\$x /= 5	\$x = \$x / 5
%=	\$x %= 5	\$x = \$x % 5
.=	\$x .= "ab"	\$x = \$x . "ab"

Tabela 7.7. Operator konkatenacji

Operator	Działanie	Przykład
.	łączenie łańcuchów znakowych	"łańcuch ." tekstowy"

Przykład 7.26

```
<?php
    $osoba["nazwisko"] = "Kowalski";
    $osoba["imie"] = "Jan";
    $osoba["wiek"] = 27;
    echo $osoba["nazwisko"] . " " . $osoba["imie"] . " ma " . $osoba["wiek"]
        . " lat.";
?>
```

Operatory inkrementacji i dekrementacji (np. \$x++ oraz \$x-- działają identycznie jak w JavaScript).

Aby stworzyć stałą w PHP należy użyć funkcji *define(nazwa, wartość, czy nazwa nie powinna uwzględniać wielkość liter - domyślnie FALSE)*.

```
<?php
define("stala", "Moja pierwsza stała!", true);
echo stala;
?>
```

Instrukcje warunkowe w PHP również zaczynają się od *if (warunek){ kod, gdy warunek jest prawdziwy; } else { kod, gdy warunek jest fałszywy; }*

```
$x=10;  
if ($x==10){  
echo "x jest równe 10";  
} else {  
echo "x jest równe $x";  
}
```

Switch sprawdza, czy pewna wartość jest równa innym.

```
$expr = "Mandarynki";  
switch ($expr) { // bierzemy pod lupę to co jest w zmiennej  
case "Pomarańcze": // jeśli w zmiennej byłyby pomarańcze wtedy ten kod się wykonaj  
echo 'Pomarańcze są po 3 zł za kilogram.';  
break; // break przerwuje aktualne przejście bloku kodu  
case "Jabłka": // jeśli jabłka lub mandarynki  
case "Mandarynki": // to wykona się ten kod  
echo "Jabłek i mandarynek nie sprzedajemy.";  
break; // break przerwuje wykonywanie dalszej części kodu  
default: // jeśli cokolwiek innego będzie w zmiennej wtedy ten kod  
echo "Nasz sklep sprzedaje tylko pomarańcze. Wypad!";  
}
```

```
// Przykład z użyciem operatora warunkowego (warunek ? wartość1 : wartość2)  
$x = 11;  
$wynik = ($x<0) ? "ujemna" : "dodatnia";  
echo "Wartość zmiennej x jest $wynik";
```

```
// Ten sam przykład z użyciem konstrukcji if...else:  
$y = 11;  
if ($y<0) {  
$wynik = "ujemna";  
} else {  
$wynik = "dodatnia";  
}  
echo "Wartość zmiennej y jest $wynik";
```

W PHP posiadamy pętle:

- *while*
- *do ... while*
- *for*
- *foreach* (używa się z tablicami)

```
// Pętla While wypisująca liczby od 1 do 10:  
$a=1;  
while ($a<10){  
echo "$a <br>"; // dodajemy znacznik <br> dla przejrzystości
```

```

$a++;
}

// Pętla do...while wypisująca liczby od 1 do 10:
$b=1;
do {
echo "$b </br>"; // dodajemy znacznik <br> dla przejrzystości
} while ($b++ <10);

// Pętla for wypisująca liczby od 1 do 10:
for ($c=1; $c=="10"; $c++){
echo "$c </br>"; // dodajemy znacznik <br> dla przejrzystości
}

// Pętla foreach:
$d=array('pierwszy','drugi','trzeci'); //tworzymy tablicę z trzema elementami
foreach ($d as $element){ // podajemy tzw. zmienną-sterującą
echo "$element </br>"; // dodajemy znacznik <br> dla przejrzystości
}

```

Tablice w PHP tworzy się za pomocą instrukcji **array()** przypisując tą tablicę do jakiejś zmiennej. **Count** obliczy ile elementów znajduje się w tablicy.

```

$a=array("t1","t2","t3"); // tworzymy tablicę z wartościami podanymi w cudzysłowie
echo $a[0]; // wyświetli indeks zerowy tablicy
$a[3]="t4"; // doda do tablicy pod indeks trzeci wartość "t4"
echo count($a); //count oblicza ile wartości znajduje się w tablicy

```

Funkcje tablicowe:

- **sort / rsort** - sortowanie tablicy rosnąco lub malejąco
- **asort** - sortowanie wartości przy zachowaniu kluczy
- **ksort** - sortowanie tablicy po wartości kluczy
- **array_rand** - wybieranie losowych kluczy istniejących elementów w tablicy
- **array_push** - wstawianie elementu na koniec tablicy
- **array_pop** - zwraca wartość ostatniego elementu i usuwa go z tablicy
- **array_shift** - usuwanie elementu z początkowej pozycji w tablicy
- **array_unshift** - wstawianie elementu na początkową pozycję w tablicy
- **array_sum** - sumowanie wartości elementów w tablicy
- **array_product** - iloczyn elementów w tablicy
- **in_array** - sprawdzanie, czy wartość istnieje w tablicy (zwraca True / False)
- **array_key_exists** - sprawdzanie, czy podany klucz lub indeks istnieje w tablicy
- **array_values** - zwraca wszystkie wartości z tablicy
- **array_merge** - scal tablice
- **range** - tworzy tablicę zawierającą przedział elementów
- **shuffle** - tasowanie tablicy

Funkcje w PHP definiujemy za pomocą słowa **function**. W nawiasach okrągłych podajemy argumenty funkcji, a w nawiasach klamrowych zapisujemy treść funkcji. Jeśli chcemy, aby funkcja zwracała wartość, która będzie wykorzystywana w innych działaniach używamy **return**.

```
function message() { //kod funkcji
echo "Super-kodzik";
}
message(); //wywołanie funkcji
```

Przekazywanie danych z formularza:

- metoda GET - używa się jej, wtedy gdy jest niewiele parametrów. W tej metodzie parametry przekazywane są za pomocą adresu URL. Adres skryptu PHP od przesyłanych parametrów oddzielony jest znakiem zapytania (parametr=wartość).
- metoda POST - do przekazywania parametrów wykorzystuje nagłówek strony. Metoda ta umożliwia przekazywanie większej ilości parametrów, a parametry nie są widoczne w pasku przeglądarki.

Przesyłane wartości parametrów zapisywane są w odpowiednich tablicach asocjacyjnych. Dane przesyłane metodą GET są zapisywane w tablicy `$_GET`, natomiast dane przesyłane metodą POST w tablicy `$_POST`. Tablice `$_GET` i `$_POST` są superglobalne.

Tworzymy dwa pliki jeden z rozszerzeniem HTML, drugi z PHP:

Pierwszy z HTML:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form action="php1.php" method="post"> <!-- action -> nazwa pliku PHP;
w method podajemy GET albo POST-->
<input type="text" name="imie"><br>
<input type="text" name="nazwisko"><br>
<input type="submit">
</form>
</body>
</html>
```

Drugi PHP:

```
<?php
$a=$_POST["imie"]; // gdybyśmy przesyłali metodą GET wtedy $_GET
$b=$_POST["nazwisko"];
// w nawiasach kwadratowych podajemy wartość atrybutu name z formularza

echo "Twoje imię to $a";
```

```
echo "Twoje nazwisko to $b";
```

```
?>
```

Operacje na plikach:

- **include** - służy do połączenia kodu znajdującego się w kilku oddzielnych plikach
- **file_exists** - ustala, czy plik lub katalog istnieje
- **is_file** - sprawdza, czy podany argument tej funkcji jest plikiem
- **filesize** - określa rozmiar pliku
- **touch** - tworzy pusty plik o podanej nazwie
- **unlink** - usuwa istniejący plik
- **fopen** - otwiera istniejący plik (podaje się w jakim trybie ma zostać otwarty: r - tylko do odczytu, w - tylko do zapisu, a - tryb dopisywania do pliku)
- **fclose** - zamykanie pliku tekstowego
- **fwrite(a,b)** - zapisywanie pliku (a -> plik zwrócony za pomocą funkcji fopen, b-> ciąg znaków, które mają zostać zapisane)
- **fgets** - odczytywanie pojedynczych wierszy (po otwarciu plików)
- **fread** - odczytywanie bloków danych
- **readfile** - odczyt danych z pliku
- **file_get_contents** - pozwala pobierać dane z zewnętrz przez URL

```
<?php  
$plik = fopen("plik.txt", "r") or die("Nie udało się wczytać pliku");  
echo fgets($plik);  
fclose($plik);  
  
/* Otwiera plik.txt tylko do odczytu, jeśli się nie uda pokaże tekst  
"Nie udało się wczytać pliku", Pokazuje zawartość pliku, a potem go zamyka  
?>
```

Pliki **cookies** to niewielkie pliki tekstowe wysyłane przez serwer lub skrypt do przeglądarki i umieszczane przez nią na dysku użytkownika. Pliki te są częścią specyfikacji protokołu HTTP i są wysyłane do przeglądarki w postaci nagłówka o nazwie **Set-Cookie**. Służą głównie do identyfikacji użytkownika. Plik cookie zawiera m.in. nazwę serwera, datę wygaśnięcia pliku oraz informacje na temat ścieżki i domeny. Maksymalny ich rozmiar to 4 kB.

Zastosowania plików cookies:

- przechowywanie nazwy użytkownika i hasła na komputerze
- przechowywanie zawartości koszyka w sklepach internetowych
- czas ostatniej wizyty na stronie

Sesje w języku PHP pozwalają na identyfikację użytkownika i śledzenie jego aktywności na stronie internetowej. Sesja to czas, w którym użytkownik przegląda stronę internetową. W momencie pierwszego wejścia na stronę tworzona zostaje ID

przechowywane na komputerze użytkownika. Zmienne związane z użytkownikiem przechowywane są na serwerze w tablicy `$_SESSION`.

Połączenie PHP z bazą danych:

Podejścia połączenia z bazą danych:

- proceduralne (strukturalne)
- obiektowe
- PDO

Różnice pomiędzy podejściem strukturalnym, a obiektowym można znaleźć [tutaj!](#)

Kolejność czynności:

1. Utworzenie obiektu połączenia (`new mysqli` lub `mysqli_connect`)
2. Sprawdzenie poprawności połączenia (`connect_error` lub `mysqli_connect_error()`)
3. Wykonanie poleceń SQL (`query` lub `multiquery`, gdy zapytań jest więcej niż jedno)
4. Zamknięcie obiektu połączenia (`close()` lub `mysqli_close()`)

```
<?php
$servername = "localhost"; // nazwa serwera
$username = "root";      // nazwa użytkownika
$password = "qwerty";    // hasło tego użytkownika
$database = "ksiegarnia" // nazwa bazy danych

// Tworzenie połączenia - podejście proceduralne:
$conn = mysqli_connect($servername, $username, $password, $database);

// Tworzenie połączenia - podejście obiektowe:
$conn = new mysqli($servername, $username, $password, $database);

// Tworzenie połączenia - podejście PDO:
$conn = new PDO("mysql:host=$servername;dbname=$database", $username, $password);

// Sprawdzenie połączenia - podejście proceduralne:
if (!$conn) {
die("Brak połączenia " . mysqli_connect_error());
echo "Połączenie z bazą zostało nawiązane";
}

// Sprawdzenie połączenia - podejście obiektowe:
if ($conn->connect_error) {
die("Brak połączenia: " . $conn->connect_error);
}

// Sprawdzenie połączenia - podejście PDO:
if ($conn->connect_error) {
die("Brak połączenia: " . $conn->connect_error);
}
```

```
// POLECENIA WYKONYWANIE NA BAZIE DANYCH:  
// Tworzenie bazy danych (jeśli nie mamy jej stworzonej) - podejście proceduralne:  
$sql = "CREATE DATABASE kawiarenka";  
if (mysqli_query($conn, $sql)) {  
echo "Baza danych została pomyślnie utworzona";  
} else {  
echo "Błąd" . mysqli_error($conn);  
}  
  
// Tworzenie bazy danych (jeśli nie mamy jej stworzonej) - podejście obiektywne:  
$sql = "CREATE DATABASE kawiarenka";  
if ($conn->query($sql) === TRUE) {  
echo "Baza danych została pomyślnie utworzona";  
} else {  
echo "Błąd" . $conn->error;  
}  
  
// Tworzenie bazy danych (jeśli nie mamy jej stworzonej) - podejście PDO:  
$sql = "CREATE DATABASE kawiarenka";  
if ($conn->exec($sql) === TRUE) {  
echo "Baza danych została pomyślnie utworzona";  
} else {  
echo "Błąd" . $conn->error;  
}  
  
// Polecenia typu INSERT - podejście proceduralne:  
$sql = "INSERT INTO Goscie (firstname, lastname, email)  
VALUES ('Janusz', 'Tracz', 'plebania@poczta.com')";  
if (mysqli_query($conn, $sql)) {  
echo "New record created successfully";  
} else {  
echo "Błąd: " . $sql . "<br>" . mysqli_error($conn);  
}  
  
// Polecenia typu INSERT - podejście obiektywne:  
$sql = "INSERT INTO Goscie (firstname, lastname, email)  
VALUES ('Janusz', 'Tracz', 'plebania@poczta.com')";  
if ($conn->query($sql) === TRUE) {  
echo "Rekord został dodany";  
} else {  
echo "Błąd: " . $sql . "<br>" . $conn->error;  
}  
  
// Polecenia typu INSERT - podejście PDO:  
$sql = "INSERT INTO Goscie (firstname, lastname, email)  
VALUES ('Janusz', 'Tracz', 'plebania@poczta.com')";  
if ($conn->exec($sql) === TRUE) {  
echo "Rekord został dodany";  
} else {  
echo "Błąd: " . $sql . "<br>" . $conn->error;  
}
```

```
// Polecenia typu SELECT - podejście proceduralne (while):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
while($row = mysqli_fetch_assoc($result)) {
echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}

// Polecenia typu SELECT - podejście proceduralne (foreach):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
foreach(mysqli_fetch_all($result, MYSQLI_ASSOC) as $row) {
echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}

// Polecenia typu SELECT - podejście obiektywne (while):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
while($row = $result->fetch_assoc()) {
echo "id: " . $row["id"]. " - Imie i nazwisko: " . $row["firstname"] .
" " . $row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}

// Polecenia typu SELECT - podejście obiektywne (foreach):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
foreach($result as $row) {
echo "id: " . $row["id"]. " - Imie i nazwisko: " . $row["firstname"] .
" " . $row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}

// Polecenia typu SELECT - podejście PDO (while):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = $conn->query($sql);
if ($result->rowCount() > 0) {
while($row = $result->fetch()) {
echo "id: " . $row["id"]. " - Imie i nazwisko: " . $row["firstname"] .
" " . $row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}
```

```
// Polecenia typu SELECT - podejście PDO (foreach):
$sql = "SELECT id, firstname, lastname FROM Goscie";
$result = $conn->query($sql);
$result->setFetchMode(PDO::FETCH_ASSOC);
foreach($result as $row) {
echo "id: " . $row["id"]. " - Imię i nazwisko: " . $row["firstname"] .
" " . $row["lastname"]. "<br>";
} else {
echo "Brak wyników";
}

// Zamknięcie połączenia z bazą danych:
mysqli_close($conn); // podejście proceduralne
$conn->close(); // podejście obiektywne
$conn = null; // podejście PDO

?>
```