

ŚCIAĞAWKA EGZAMINACYJNA

HTML5, CSS, formularze i PHP

Przypomnienie najważniejszych elementów przed egzaminem INF.03

Zawartość ściągawki:

- Wszystkie znaczniki semantyczne HTML5 z polskim wyjaśnieniem
- CSS: kiedy używać id, kiedy class, i jak się do nich odwołać
- Pełna budowa formularza HTML ze wszystkimi elementami
- Różnica między metodami GET i POST
- PHP: pętle, zliczanie elementów, różnica między echo ' a echo "

1. Znaczniki semantyczne HTML5

NOWE POJĘCIE: Znacznik semantyczny

To znacznik, którego SAMA NAZWA mówi, jakiego rodzaju treść jest w środku (w odróżnieniu od `<div>`, który nic nie mówi o zawartości). Dzięki temu przeglądarki, czytniki ekranu dla niewidomych i wyszukiwarki „rozumieją” strukturę strony, a nie tylko widzą bezkształtne pudełka.

Znacznik	Opis (ang.)	Wyjaśnienie po polsku
<code><article></code>	Defines independent, self-contained content	Samodzielna, kompletna treść — ma sens sama w sobie, np. wpis na blogu, wiadomość, karta produktu/aktora
<code><aside></code>	Defines content aside from the page content	Treść POBOCZNA względem głównej — np. panel z reklamą, dodatkowe informacje, „na marginesie”
<code><details></code>	Defines additional details that the user can view or hide	Rozwijany/zwijany blok treści (kliknięcie pokazuje/ukrywa zawartość) — jak „FAQ” z możliwością rozwinięcia
<code><figcaption></code>	Defines a caption for a <code><figure></code> element	Podpis pod obrazkiem/ilustracją — używany TYLKO wewnątrz <code><figure></code>
<code><figure></code>	Specifies self-contained content, like illustrations, diagrams, photos, code listings	Samodzielny blok „ilustracyjny” — obraz, wykres, fragment kodu — zwykle razem z <code><figcaption></code>
<code><footer></code>	Defines a footer for a document or section	Stopka dokumentu lub sekcji — autor, kontakt, prawa autorskie, linki pomocnicze
<code><header></code>	Specifies a header for a document or section	Nagłówek dokumentu lub sekcji — logo, tytuł, wprowadzenie
<code><main></code>	Specifies the main content of a document	Główna, UNIKATOWA treść strony — tylko JEDEN <code><main></code> na całej stronie
<code><mark></code>	Defines marked/highlighted text	Tekst zaznaczony/wyróżniony — domyślnie żółte tło, jak zakreślacz
<code><nav></code>	Defines navigation links	Blok z odnośnikami nawigacyjnymi — menu strony
<code><section></code>	Defines a section in a document	Wydzielona, tematyczna sekcja dokumentu — może być ich wiele na stronie
<code><summary></code>	Defines a visible heading for a <code><details></code> element	Widoczny nagłówek/etykieta elementu <code><details></code> — to, co widać, zanim rozwiniesz szczegóły
<code><time></code>	Defines a date/time	Data lub godzina, zapisana w formacie zrozumiałym dla komputera (atrybut <code>datetime</code>)

Dodatkowe znaczniki semantyczne (poziom tekstu)

Te znaczniki nie dzielą strony na bloki, tylko nadają ZNACZENIE fragmentom tekstu wewnątrz akapitu:

Znacznik	Domyślny wygląd	Znaczenie
<code></code>	pogrubiony	tekst WAŻNY
<code></code>	pochylony (kursywa)	tekst z NACISKIEM/uwypuklony
<code><small></code>	mniejsza czcionka	drobny druk, zastrzeżenia prawne
<code><cite></code>	kursywa	tytuł dzieła (książki, filmu)
<code><abbr></code>	linia kropkowana pod spodem	skrót, z pełnym rozwinięciem w atrybucie title

Jak czytać treść zadania i wybrać właściwy znacznik

Zadanie prawie nigdy nie mówi wprost „użyj `<em>`” — opisuje słowami: „tekst uwypuklony, formatowany domyślnie jako pochyłony” (= `<em>`) albo „tekst ważny, formatowany domyślnie jako pogrubiony” (= `<strong>`). Dopasuj opis DOMYŚLNEGO WYGLĄDU z tabeli powyżej do zdania w treści zadania.

2. CSS — kiedy id, kiedy class, i jak się odwołać

W HTML	W CSS	Kiedy używać
id="cos"	#cos { }	gdy element jest JEDEN, unikatowy na całej stronie (np. konkretny nagłówek, konkretna sekcja)
class="cos"	.cos { }	gdy ten sam styl ma dotyczyć WIELU elementów naraz (np. każda karta produktu, każdy wiersz listy)

Jak to zapamiętać na stałe

- **#** (krzyżyk) → **id** — bo id jest jak numer PESEL: dotyczy dokładnie JEDNEJ osoby/elementu
- **.** (kropka) → **class** — bo klasa jest jak grupa uczniów: wielu należy do tej samej klasy naraz

Przykład

```
<header id="naglowek1">...</header>
<article class="aktor-item">...</article>
<article class="aktor-item">...</article>
#naglowek1 {
    background-color: navy;
}

.aktor-item {
    width: 15em;
}
```

Ważny wyjątek na egzaminie: „tylko selektor znacznika”

Gdy treść zadania mówi „styl CSS obrazu i odnośnika należy zdefiniować WYŁĄCZNIE przy pomocy selektora tego znacznika” — dla TYCH konkretnych elementów nie wolno używać ani id (#), ani class (.) — tylko sam znacznik: `img { }` albo `a { }`. To zastrzeżenie dotyczy wyłącznie elementów wymienionych wprost w treści zadania.

3. Budowa formularza HTML — wszystkie elementy

```
<form method="get" action="strona.php">

  <label for="imie">Imię:</label>
  <input type="text" id="imie" name="imie" required>

  <label for="haslo">Hasło:</label>
  <input type="password" id="haslo" name="haslo">

  <label for="wiek">Wiek:</label>
  <input type="number" id="wiek" name="wiek" min="1" max="120">

  <label for="data">Data:</label>
  <input type="date" id="data" name="data">

  <input type="radio" id="plecK" name="plec" value="k" checked>
  <label for="plecK">Kobieta</label>
  <input type="radio" id="plecM" name="plec" value="m">
  <label for="plecM">Mężczyzna</label>

  <input type="checkbox" id="zgoda" name="zgoda" value="tak">
  <label for="zgoda">Akceptuję regulamin</label>

  <label for="miasto">Miasto:</label>
  <select id="miasto" name="miasto">
    <option value="WAW">Warszawa</option>
    <option value="KRK">Kraków</option>
  </select>

  <label for="opis">Opis:</label>
  <textarea id="opis" name="opis" rows="4" cols="30"></textarea>

  <button type="submit">Wyślij</button>
  <button type="reset">Wyczyść</button>
</form>
```

Element / atrybut	Rola
<code><form></code>	opakowuje cały formularz; method i action decydują, JAK i DOKĄD wysłać dane
<code><label for="..."></code>	podpis pola, powiązany z jego id — kliknięcie w etykietę zaznacza/aktywuje pole
<code><input type="text"></code>	zwykłe pole tekstowe jednej linii
<code><input type="password"></code>	pole tekstowe z ukrytymi znakami (kropki zamiast liter)
<code><input type="number"></code>	pole akceptujące tylko liczby (opcjonalnie min/max)
<code><input type="date"></code>	wybór daty z wbudowanego kalendarza przeglądarki
<code><input type="radio"></code>	wybór JEDNEJ opcji z kilku — pola z tym samym name są ze sobą powiązane
<code><input type="checkbox"></code>	pole zaznacz/odznacz — niezależne, można wybrać kilka naraz
<code><select>/<option></code>	lista rozwijana; value opcji = wartość wysyłana, tekst = to, co widzi użytkownik

Element / atrybut	Rola
<code><textarea></code>	wieloliniowe pole tekstowe (rows/cols = wysokość/szerokość)
<code><button type="submit"></code>	wysyła formularz
<code>name</code>	KLUCZOWY atrybut — to pod tą nazwą wartość trafia do PHP (\$_GET/\$_POST)
<code>required</code>	pole musi być wypełnione, zanim formularz się wyśle (walidacja w przeglądarce)
<code>checked</code>	pole radio/checkbox domyślnie zaznaczone

name w HTML musi być identyczne jak klucz w PHP

Żeby PHP odczytał wartość przez `$_GET["imie"]`, w HTML pole MUSI mieć dokładnie `name="imie"` — ta sama pisownia, ta sama wielkość liter. To najczęstsza przyczyna „nie działających” formularzy.

4. Metody przesyłania formularza: GET i POST

Cecha	GET	POST
Dane widoczne w adresie URL	✗ tak	☑ nie
Dane mogą trafić do historii przeglądarki	✗ tak	☑ nie (nie w samym adresie)
Nadaje się do haseł	✗ nie	☑ tak
Nadaje się do danych osobowych	⚠ niewskazane	☑ tak
Duża ilość danych	✗ ograniczona (długość URL)	☑ praktycznie nieograniczona
Tablica PHP	\$_GET	\$_POST
Można dodać do zakładek?	tak — adres z parametrami można zapisać	nie — dane znikają po zamknięciu strony
Typowe zastosowanie	wyszukiwanie, filtrowanie, wybór z listy	logowanie, rejestracja, zapis danych

Który jest naprawdę „bezpieczniejszy”? — POST, nie GET

Dla formularzy z danymi użytkownika (hasła, dane osobowe, cokolwiek wrażliwego) zdecydowanie lepszym wyborem jest POST: dane nie pojawiają się w adresie URL, nie zapisują się w historii przeglądarki ani w logach serwera, i nie da się ich przypadkiem "wysłać dalej" komuś, kopiując link ze strony. **WAŻNE ZASTRZEŻENIE:** sam POST nie "szyfruje" ani nie zabezpiecza danych — to jedynie kwestia tego, GDZIE dane są umieszczone w żądaniu, a nie CZY są chronione. Prawdziwe bezpieczeństwo (szyfrowanie przesyłu) zapewnia dopiero protokół HTTPS, niezależnie od tego, czy formularz używa GET czy POST.

```
<form method="get" action="wynik.php">
<form method="post" action="wynik.php">
```

Odczyt w PHP musi używać tej samej tablicy, co metoda formularza:

```
$imie = $_GET["imie"]; // gdy method="get"
$imie = $_POST["imie"]; // gdy method="post"
```

5. PHP — najważniejsze elementy na egzamin INF.03

5.1. Pętle

Na egzaminie niemal zawsze używa się pętli while do przechodzenia przez wynik zapytania SQL:

```
while ($wiersz = mysqli_fetch_array($wynik)) {
    echo "<li>" . $wiersz["nazwa"] . "</li>";
}
```

Jak to działa: każdy obieg pętli pobiera KOLEJNY wiersz z **\$wynik** i przypisuje go do **\$wiersz**. Gdy wierszy zabraknie, funkcja zwraca **FALSE**, warunek pętli staje się fałszywy i pętla sama się zatrzymuje — bez liczenia, ile wierszy było.

Pętla for przydaje się, gdy z góry znasz liczbę powtórzeń (np. wypisanie 20 takich samych obrazków):

```
for ($i = 1; $i <= 20; $i++) {
    echo "<img src='obraz.png'>";
}
```

5.2. Zliczanie elementów

Sytuacja	Jak policzyć
liczba wierszy zwróconych zapytaniem SQL	mysqli_num_rows(\$wynik)
liczba elementów tablicy PHP	count(\$tablica)
suma/średnia/min/max wartości w bazie	licz to w SQL: SUM(), AVG(), MIN(), MAX() — nie w PHP

```
$wynik = mysqli_query($polaczenie, "SELECT * FROM produkty");
$liczbaProduktow = mysqli_num_rows($wynik);

if ($liczbaProduktow == 0) {
    echo "Brak produktów.";
} else {
    echo "Znaleziono " . $liczbaProduktow . " produktów.";
}
```

Pułapka: = a ==

\$liczba = 0 to PRZYPISANIE (wstawia 0 do zmiennej). \$liczba == 0 to PORÓWNANIE (pyta, czy zmienna jest równa 0). W warunku if zawsze chcesz ==, nigdy pojedynczego = — pomyłka o jeden znak całkowicie zmienia działanie kodu.

5.3. Różnica między echo ' a echo "

To jedna z tych rzeczy, która wygląda błaho, ale ma realne znaczenie w konkretnych sytuacjach.

Zapis	Co robi ze zmiennymi w środku
echo "..."	PODSTAWIA wartość zmiennej wprost w tekście; rozumie \n, \t
echo '...'	bierze tekst DOSŁOWNIE, nic nie podstawia

```
$imie = "Jan";

echo "Cześć, $imie!"; // Cześć, Jan!
echo 'Cześć, $imie!'; // Cześć, $imie! (dosłownie, bez podstawienia)
```

Gdy sklejasz zmienną kropką `.` (konkatenacja), obie wersje działają IDENTYCZNIE — bo wtedy PHP i tak „wychodzi” z cudzysłowu, żeby doczepić zmienną:

```
echo "Cześć, " . $imie . "!"; // Cześć, Jan!
echo 'Cześć, ' . $imie . '!'; // Cześć, Jan!
```

Gdzie to naprawdę ma znaczenie — generowanie HTML z atrybutami

HTML też używa cudzysłowów na atrybutach (np. `src="..."`), więc trzeba uważać, żeby nie „zderzyć się” z cudzysłowem PHP:

```
// wersja 1: cały tekst PHP w apostrofach - cudzyslow HTML w srodku NIE koliduje
echo '<img src=' . $plik . ' ' alt="zdjęcie">';

// wersja 2: cały tekst PHP w cudzyslowie - trzeba uciec wewnetrzny cudzyslow znakiem \
echo "<img src=\"\" . $plik . \"\" alt=\"zdjęcie\">";
```

Rekomendacja na egzamin

Trzymaj się JEDNEJ zasady przez całe zadanie: cały tekst echo w apostrofach `'...'`, a atrybuty HTML w środku w cudzysłowie `"..."`. Unikniesz znaków ucieczki i pomyłek pod presją czasu — to dokładnie styl używany we wszystkich przykładach w tym materiale.

5.4. Klamry `{ }` przy wstawianiu elementów tablicy w cudzysłowie

Gdy jednak zdecydujesz się pisać duży blok HTML w jednym echo z cudzysłowem `"..."` (wygodne przy wielu liniach HTML naraz), pojawia się dodatkowa zasada dotycząca wstawiania ELEMENTÓW TABLICY (jak `$row["nazwa"]`) — w odróżnieniu od prostych zmiennych typu `$imie`.

Prosta zmienna działa bez problemu:

```
echo "Cześć, $imie!"; // działa poprawnie
```

Element tablicy z apostrofami w kluczu — BEZ klamer NIE działa poprawnie:

```
echo "<h3>$row['nazwa']</h3>"; // ŹLE - apostrofy w środku psują składnię
```

PHP wewnątrz cudzysłowu rozumie proste podstawienie `$row[nazwa]` (BEZ apostrofów wokół klucza), ale gdy dopiszesz apostrofy (`'nazwa'`), PHP się "gubi" — apostrofy przestają być częścią nazwy klucza, a stają się zwykłym, wypisanym tekstem. Wynik wygląda jak zepsuty ciąg znaków zamiast prawdziwej wartości.

Rozwiązanie: opakuj cały zapis w klamry `{ }`:

```
echo "<h3>{$row['nazwa']}</h3>"; // DOBRZE
```

Klamry mówią PHP: „wszystko pomiędzy `{ }` potraktuj jako jedno, kompletne wyrażenie PHP — oblicz je i wstaw wynik w to miejsce”. Dzięki temu apostrofy wewnątrz `'nazwa'` nie kolidują z niczym.

Zapis	Działa poprawnie?
<code>"\$imie"</code>	☒ tak (prosta zmienna)
<code>"\$row[nazwa]"</code> (bez apostrofów w kluczu)	☒ tak, ale niezalecane (PHP zgłasza ostrzeżenie)
<code>"\$row['nazwa']"</code> (z apostrofami, BEZ klamer)	☒ nie działa poprawnie
<code>"{\$row['nazwa']}"</code> (z klamrami)	☒ tak, zawsze bezpiecznie

Przykład: cały blok HTML za jednym razem

Ten styl (cudzysłów na zewnątrz + klamry) najbardziej opłaca się przy dużych, wieloliniowych fragmentach HTML z wieloma zmiennymi naraz — sklejanie tylu fragmentów kropką byłoby nieczytelne:

```
echo "<div class='produkt'>
  <div class='obraz'>
    <img src='img/{\$row['plik']}' alt='produkt'>
  </div>
```

```
<h3>{$row['nazwa']}</h3>
<h4>styl {$row['styl']}</h4>
<p>CENA: {$row['cena']} zł</p>
</div>";
```

Kiedy który styl wybrać

Krótką linijką, 1–2 zmienne → apostrofy na zewnątrz + kropka ('...' . \$zmienna . '...'). Duży, wieloliniowy blok HTML z wieloma zmiennymi → cudzysłów na zewnątrz + {\$row['klucz']} w środku. Obie metody są równie poprawne na egzaminie — druga bywa czytelniejsza przy generowaniu całego, rozbudowanego fragmentu HTML za jednym razem.