Fundusze Europejskie dla Wielkopolski
Dofinansowane przez Unię Europejską
SAMORZĄD WOJEWÓDZTWA WIELKOPOLSKIEGO

Projektowanie stron internetowych

Materiały dydaktyczne dla uczestników kursu

Miłosz Olejniczak
Paweł Sobczak

Konin 2025

Tytuł

Projektowanie stron internetowych
Materiały dydaktyczne dla uczestników kursu

Autorzy

Miłosz Olejniczak

Paweł Sobczak

Projekt pn.
„Rozwój studiów o profilu praktycznym i form kształcenia ustawicznego dostosowanych do potrzeb Wielkopolski Wschodniej”,
realizowany przez Akademię Nauk Stosowanych w Koninie,
jest współfinansowany przez Unię Europejską
ze środków Funduszu na rzecz Sprawiedliwej Transformacji
w ramach programu Fundusze Europejskie dla Wielkopolski 2021-2027

Wydawca

Akademia Nauk Stosowanych w Koninie
ul. Przyjaźni 1, 62-510 Konin

Spis treści

Frontend stron WWW (*Miłosz Olejniczak*)

Wprowadzenie do technologii internetowych

Współczesne strony internetowe „pracują" dzięki zestawowi protokołów i standardów, pozwalających na szybkie i skuteczne komunikowanie się użytkownika (klienta) z serwerem. Jednym z kluczowych elementów tej układanki jest **HTTP** (*Hypertext Transfer Protocol*), który reguluje sposób, w jaki przeglądarka żąda informacji od serwera. Kiedy zaś w grę wchodzi ochrona danych, w szczególności wrażliwych (jak numer karty kredytowej czy dane logowania), sięgamy po szyfrowaną wersję tego protokołu – **HTTPS**.

Jeżeli wyobrazimy sobie sam proces, wygląda on mniej więcej tak: gdy w przeglądarce wpiszesz adres strony (URL), twoja przeglądarka wysyła zapytanie do serwera. Serwer odnajduje zasób (np. plik HTML) i odsyła go wraz z potrzebnymi arkuszami stylów, grafikami czy skryptami. Twoja przeglądarka składa te wszystkie części w jedną całość, a ty oglądasz gotową stronę.

Znaczenie i rola języka HTML

HTML (*HyperText Markup Language*) stanowi bazę zdecydowanej większości stron internetowych. Można go porównać do konstrukcji nośnej budynku – zapewnia nie tylko fundament, ale przede wszystkim określa, jaką formę przyjmują poszczególne elementy (np. nagłówek, lista, paragraf). Aktualnie obowiązujący standard, **HTML5**, usprawnia wiele aspektów związanych z integracją elementów multimedialnych (audio, wideo) i ułatwia tworzenie stron bardziej zrozumiałych dla przeglądarek i urządzeń wspomagających.

Warto pamiętać, że choć HTML organizuje treść i nadaje jej znaczenie semantyczne, to nie odpowiada bezpośrednio za wygląd – tym zajmuje się **CSS** (*Cascading Style Sheets*). Dopiero wraz z CSS uzyskujemy pełen obraz, jak strona będzie się prezentowała. Niemniej sam HTML jest nieodzowny, ponieważ bez niego nie mielibyśmy żadnego „szkieletu" do ozdobienia.

Struktura dokumentu HTML5

Każda strona HTML5 zaczyna się od:

```
<!DOCTYPE html>
```

To niewielkie polecenie w nagłówku dokumentu mówi przeglądarce, że mamy do czynienia z nowoczesnym formatem HTML. Zaraz po nim z reguły widzimy:

```
<html lang="pl">
<head>
```

```
  <meta charset="UTF-8">
  <title>Tytuł dokumentu</title>
</head>
<body>
  <!-- Treść strony -->
</body>
</html>
```

- **<html>** – główny kontener całego dokumentu. Atrybut lang="pl" informuje przeglądarkę (oraz różne narzędzia typu czytniki ekranowe) o języku zawartości.
- **<head>** – przestrzeń na informacje meta, takie jak kodowanie znaków (<meta charset="UTF-8">), tytuł (wyświetlany w karcie przeglądarki) oraz linki do zewnętrznych arkuszy stylów i skryptów.
- **<body>** – tu pojawia się właściwa treść strony, czyli teksty, obrazy, formularze i inne elementy, z którymi użytkownik ma bezpośredni kontakt.

Niezwykle ważnym aspektem jest deklaracja kodowania **UTF-8**, która zapewnia prawidłowe wyświetlanie polskich znaków (ą, ć, ę, ł, ń, ó, ś, ź, ż). Dzięki temu nie musimy obawiać się, że odwiedzający naszą stronę użytkownicy zobaczą nieczytelne symbole zamiast poprawnie wyświetlonych polskich liter.

Edytory do pisania kodu

Choć większość systemów operacyjnych oferuje proste narzędzia do edycji plików tekstowych (jak Notatnik w Windows), w praktyce warto sięgnąć po bardziej zaawansowane edytory, które ułatwiają pracę dzięki szeregowi użytecznych funkcji. Oto kilka popularnych propozycji:

- **Visual Studio Code (Microsoft)** – wieloplatformowe, darmowe środowisko programistyczne z bogatym zestawem wtyczek.
- **Sublime Text (Sublime HQ)** – wyróżnia się szybkością działania i dużymi możliwościami personalizacji.
- **Atom (GitHub)** – przyjazny, otwartoźródłowy edytor, który można szeroko konfigurować.
- **Phoenix Code** – nowoczesny edytor kodu, który łączy intuicyjność, wydajność i zaawansowane funkcje wspierające programistów.
- **Brackets** – lekki, otwartoźródłowy edytor kodu stworzony z myślą o projektantach i programistach webowych, oferujący funkcje takie jak podgląd na żywo i edycję w kontekście.
- **Notepad++** – lekki i intuicyjny, szczególnie polecany osobom rozpoczynającym przygodę z tworzeniem stron.

Wybór jest uzależniony od indywidualnych preferencji, jednak warto, aby dany edytor umożliwiał choćby podstawowe: **podświetlanie składni**, **automatyczne podpowiedzi**, a także **wygodną organizację plików**.

Ćwiczenia

Na koniec tej części zachęcamy do wykonania krótkiego ćwiczenia w celu utrwalenia zdobytej wiedzy:

1. **Utwórz główny folder projektu** (np. projekt).
2. **Dodaj w nim foldery**: images (na pliki graficzne), css (na style) i js (na skrypty).
3. **Otwórz ulubiony edytor kodu** i w głównym katalogu stwórz plik index.html.
4. **Wklej szkielet HTML5**, np.:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Moja Pierwsza Strona</title>
</head>
<body>
  <h1>Wprowadzenie do HTML5</h1>
  <p>Przykładowy akapit z polskimi znakami: ą, ć, ę, ł, ń, ó, ś, ź, ż.</p>
</body>
</html>
```

5. **Zapisz plik** i otwórz go w wybranej przeglądarce (Google Chrome, Mozilla Firefox itp.). Upewnij się, że tekst wyświetla się prawidłowo – zwłaszcza polskie znaki.

Dzięki temu prostemu zadaniu w praktyce sprawdzisz, czy poprawnie zbudowałeś strukturę pliku HTML i czy kodowanie znaków działa tak, jak powinno. Jeżeli wszystko jest w porządku, możesz śmiało przejść do dalszych zadań i poszerzać możliwości swojej przyszłej strony WWW.

Podstawowe znaczniki HTML

Tym razem skupimy się na wybranych, najbardziej typowych znacznikach, dzięki którym można zorganizować i uporządkować treści na stronie. Świadome posługiwanie się nagłówkami, akapitami i innymi elementami HTML pozwala bowiem nie tylko na zapewnienie czytelności kodu, lecz także na podniesienie dostępności oraz walorów estetycznych serwisu.

Znaczniki nagłówków (*heading*)

Czym są nagłówki?

Nagłówki (headingi) to elementy stosowane w celu wydzielenia i zasygnalizowania ważniejszych części tekstu w dokumencie HTML. Dostępne są w sześciu poziomach:

```
<h1>Najważniejszy Nagłówek</h1>
<h2>Podrozdział / Nagłówek Drugiego Poziomu</h2>
<h3>Kolejne Podpoziomy...</h3>
<h4></h4>
<h5></h5>
<h6></h6>
```

- **<h1>** jest zwykle traktowany jako tytuł główny strony lub sekcji.
- **<h2>** i kolejne poziomy służą do hierarchizacji treści: warto wyobrazić je sobie jako nagłówki rozdziałów i podrozdziałów pracy pisanej w edytorze tekstu.

Dlaczego to jest istotne?

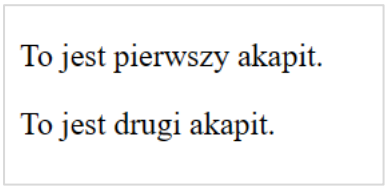
Nagłówki znacząco wpływają na sposób interpretacji treści przez przeglądarki, wyszukiwarki (SEO) oraz czytniki ekranu dla osób z niepełnosprawnościami wzrokowymi. Właściwe wykorzystanie hierarchii sprawia, że cała strona jest łatwiejsza w nawigacji zarówno dla użytkowników, jak i dla robotów indeksujących.

Akapity (paragrafy)

Podstawowy element treści

Znacznik **<p>** („paragraph”) służy do definiowania bloków tekstu, zwanych potocznie akapitami. Każdy akapit oddzielany jest wizualnie od następnego standardowym odstępem. Przykładowo:

```
<p>To jest pierwszy akapit.</p>
<p>To jest drugi akapit.</p>
```



Dzięki temu kod HTML zyskuje większą przejrzystość, co przekłada się na wygodę czytania tekstu na stronie. W artykułach i wpisach blogowych akapity pełnią kluczową rolę w budowaniu zrozumiałych i przyjaznych w odbiorze treści.

Łamanie linii (nowa linia) i linia horyzontalna

Nowa linia –

Znacznik
 (skrót od *break*) umożliwia wstawienie ręcznego łamania linii w danym miejscu. Stosuje się go w sytuacjach, gdy nie chcemy lub nie możemy rozpoczynać kolejnego akapitu, a zależy nam na przejściu do następnej linii – na przykład w wypunktowaniach tworzonych bez list HTML lub w wierszach poezji.

```
<p>To jest tekst, w którym chcę wymusić<br>złamanie linii.</p>
```

Linia horyzontalna – <hr>

Znacznik <hr> (ang. *horizontal rule*) wstawia poziomą linię oddzielającą sekcje treści:

```
<hr>
```

Przykładowo może posłużyć do wizualnego wydzielenia cytatu, dygresji bądź podsumowania w obrębie artykułu. Choć w wielu nowoczesnych projektach graficzne „linie" tworzy się głównie przy użyciu CSS, hr wciąż znajduje zastosowanie jako element semantyczny.

Ćwiczenia

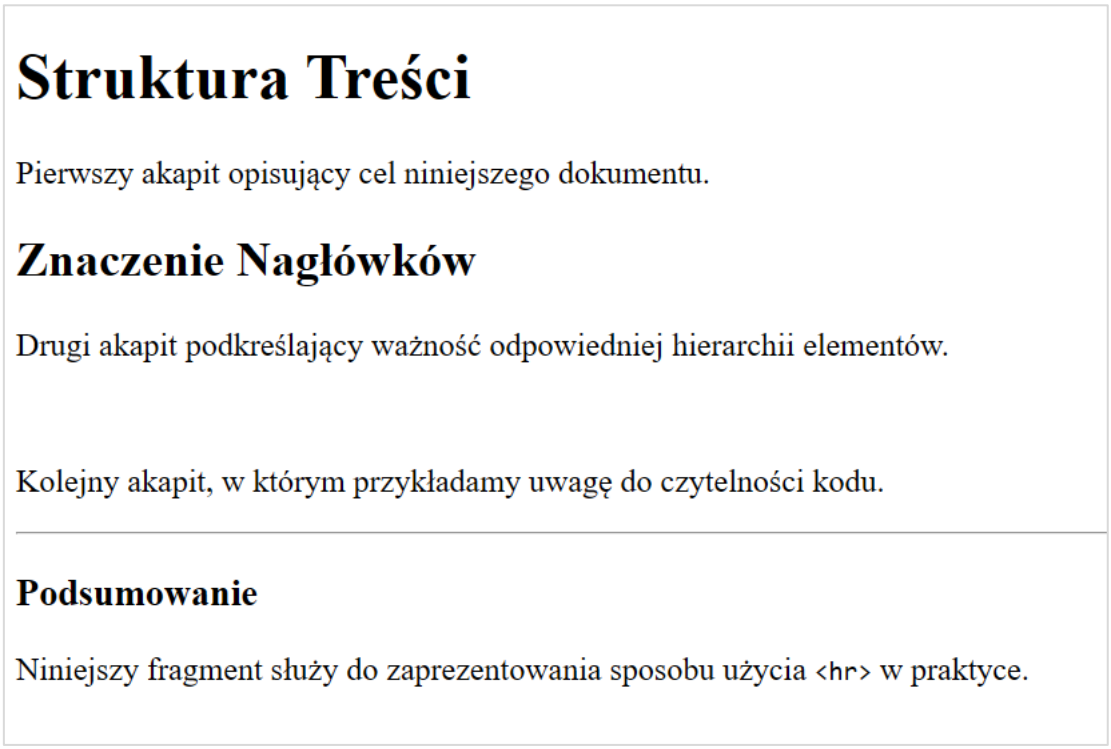
Aby przećwiczyć omawiane zagadnienia, warto stworzyć na lokalnym komputerze nowy plik HTML (np. podstawowe-znaczniki.html) i umieścić w nim następujące elementy:

1. **Sekwencję nagłówków** (od <h1> do <h3> lub <h4>), które będą imitować strukturę krótkiego artykułu bądź spisu treści.
2. **Kilka akapitów** z przykładowymi treściami, w których sprawdzimy poprawne wyświetlanie polskich znaków i odstępów między blokami tekstu.
3. **Celowe łamania linii (
)** w miejscach, gdzie chcemy od siebie oddzielić krótki tekst.
4. **Poziomą linię (<hr>)** sygnalizującą koniec jednego rozdziału i początek kolejnego.

Proponowany kod może wyglądać następująco:

```
  <!DOCTYPE html>
  <html lang="pl">
  <head>
    <meta charset="UTF-8">
    <title>Nagłówki i Akapity</title>
  </head>
  <body>
    <h1>Struktura Treści</h1>
    <p>Pierwszy akapit opisujący cel niniejszego dokumentu.</p>

    <h2>Znaczenie Nagłówków</h2>
    <p>Drugi akapit podkreślający ważność odpowiedniej hierarchii
elementów.</p>
    <br>
    <p>Kolejny akapit, w którym przykładamy uwagę do czytelności kodu.</p>

    <hr>

    <h3>Podsumowanie</h3>
    <p>Niniejszy fragment służy do zaprezentowania sposobu użycia
    <code>&lt;hr&gt;</code> w praktyce.</p>
  </body>
  </html>
```

Po otwarciu pliku w przeglądarce internetowej możemy na bieżąco obserwować efekty i modyfikować zawartość, aby jeszcze lepiej zrozumieć działanie każdego ze znaczników.



Formatowanie tekstu, listy, linki

W tym rozdziale zapoznamy się z kolejnym zestawem kluczowych elementów języka HTML, które pozwalają nie tylko na wzbogacenie treści o wyróżnienia i odnośniki, ale także na stworzenie przejrzystych struktur w postaci list. Podczas gdy poprzednie zajęcia dotyczyły podstawowych znaczników, takich jak nagłówki oraz akapity, w niniejszej części kursu nauczymy się nadawać tekstowi odpowiednią „intonację" oraz organizować go w bardziej złożone formy.

Formatowanie tekstu w HTML

Semantyczne znaczniki wyróżniające treść

- **** i **** – znaczniki te, choć kojarzone odpowiednio z pogrubieniem (strong) oraz pochyleniem (em), mają w istocie głębsze znaczenie semantyczne.
    - **** wskazuje, iż dana fraza ma szczególne znaczenie lub wyjątkową wagę (np. kluczowe pojęcie w akapicie).
    - **** sugeruje natomiast, że dany fragment jest intonacyjnie lub logicznie wyróżniony (np. akcent w zdaniu, cytat lub słowo kluczowe).

W przeciwieństwie do czysto wizualnych znaczników <b> i <i> (o których mowa poniżej), strong i em pozwalają wyszukiwarkom oraz czytnikom ekranu lepiej zrozumieć, co w tekście jest naprawdę istotne.

Znaczniki prezentacyjne

- **<b>** (*bold*) i **<i>** (*italic*) – te znaczniki pełnią funkcję wyłącznie wizualnego pogrubienia bądź pochylenia tekstu. Nie dodają jednak żadnej informacji semantycznej.
- **<u>** (*underline*) – odpowiedzialny za podkreślenie tekstu. Zalecany ostrożnie, ponieważ w wielu projektach podkreślenie wykorzystuje się do oznaczania linków i może to wprowadzać niepotrzebne zamieszanie.
- **<mark>** – służy do wyróżnienia fragmentu tekstu w sposób przypominający zaznaczenie zakreślaczem (np. ważny cytat czy definicja terminu).

W kontekście projektów serwisów internetowych istotne jest, aby stosować semantyczne znaczniki (,) zamiast czysto prezentacyjnych (<b>, <i>) tam, gdzie komunikujemy faktyczne znaczenie lub podkreślamy rangę danej informacji.

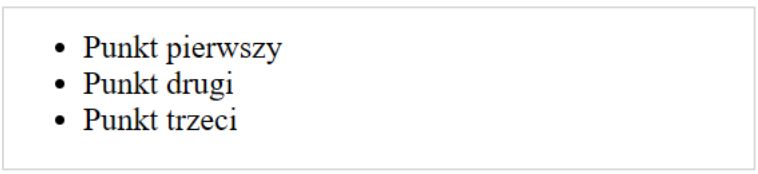
Listy

Listy w języku HTML to wygodny sposób na uszeregowanie lub wyliczenie pewnych elementów. Usprawniają organizację treści oraz zwiększają czytelność dokumentu.

Listy nieuporządkowane (ang. *unordered lists*)

Definiowane za pomocą znacznika **<ul>** i poszczególnych elementów **<li>** (*list item*). Przed każdym elementem listy pojawia się symbol wypunktowania (domyślnie kropka, choć można to zmienić w CSS).

```
<ul>
  <li>Punkt pierwszy</li>
  <li>Punkt drugi</li>
  <li>Punkt trzeci</li>
</ul>
```



Taka postać list przydaje się w sytuacjach, gdy nie zależy nam na kolejności elementów, np. przy tworzeniu menu nawigacyjnego czy list zadań.

Listy uporządkowane (ang. *ordered lists*)

Tworzone za pomocą znacznika **<ol>**, przy czym każdy element nadal definiujemy poprzez **<li>**. Przyjmuje się, że elementy w <ol> są liczone domyślnie od 1 w górę:

```
<ol>
  <li>Element pierwszy</li>
  <li>Element drugi</li>
  <li>Element trzeci</li>
</ol>
```

1. Element pierwszy
2. Element drugi
3. Element trzeci

Listy uporządkowane sprawdzają się, gdy należy zachować kolejność pozycji, np. przy instrukcjach „krok-po-kroku” czy prezentowaniu hierarchii procedur w projekcie inżynierskim.

Listy zagnieżdżone

Wewnętrznie można zagnieżdżać kolejne listy <ul> lub <ol> w obrębie elementów <li>, co przydaje się przy budowaniu wielopoziomowych struktur:

```
<ul>
  <li>Punkt główny 1
    <ul>
      <li>Podpunkt 1.1</li>
      <li>Podpunkt 1.2</li>
    </ul>
  </li>
  <li>Punkt główny 2</li>
</ul>
```

- Punkt główny 1
  - Podpunkt 1.1
  - Podpunkt 1.2
- Punkt główny 2

Ważne jest, by zachować przejrzystość kodu i czytelne wcięcia, ułatwiające nawigację po treści.

Linki (kotwice)

Znacznik <a> i atrybut href

Linki (hiperłącza) to esencja hipertekstowości w World Wide Web. Tworzymy je za pomocą znacznika **<a>** i podstawowego atrybutu **href**:

```
<a href="https://www.przyklad.pl">Kliknij tutaj, aby przejść na inną stronę</a>
```

Kliknij tutaj, aby przejść na inną stronę

Po kliknięciu w powyższy odnośnik przeglądarka przekieruje użytkownika na wskazany adres (URL). Linki mogą również prowadzić do innej podstrony w obrębie tego samego serwisu (np. onas.html) bądź odwoływać się do zasobów takich jak dokumenty PDF, obrazy czy sekcje w ramach jednej strony (tzw. linki kotwicowe).

Atrybut target i inne

- **target="_blank"** – otwiera link w nowej karcie lub nowym oknie przeglądarki.
- **title="Opis linku"** – dostarcza dodatkowego opisu linku, widocznego po najechaniu kursorem myszy. Atrybut ten bywa pomocny w pozycjonowaniu i w narzędziach ułatwiających dostępność.

Ćwiczenia

1. **Formatowanie wybranych fragmentów tekstu**
   - Utwórz stronę formatowanie.html i dodaj kilka akapitów, w których strategicznie zastosujesz znaczniki , , <u> oraz <mark>.
   - Obserwuj, jak przeglądarka interpretuje te znaczniki i czytelnie wyróżnia fragmenty treści.
2. **Tworzenie list**
   - W tym samym pliku lub w nowym, np. listy.html, utwórz przykładową listę zakupów w postaci listy nieuporządkowanej (<ul>).
   - Dodaj listę uporządkowaną (<ol>) pokazującą np. kolejność kroków montażu urządzenia bądź procedurę w projekcie inżynierskim.
   - Spróbuj też stworzyć listę zagnieżdżoną, aby zobaczyć, jak elementy podlisty są reprezentowane przez przeglądarkę.

3. **Linki**
    - Wypróbuj tworzenie odnośników, zarówno wewnętrznych np. href="index.html") jak i zewnętrznych (href="https://...").
    - Dodaj atrybut target="_blank" do jednego z linków i przetestuj zachowanie w przeglądarce.

Grafika na stronie

Poniższy materiał poświęcony jest zagadnieniu osadzania grafiki w dokumencie HTML. Umiejętne korzystanie z obrazów pozwala nie tylko na uatrakcyjnienie warstwy wizualnej witryny, lecz także na wzbogacenie przekazu informacyjnego. Poniższy rozdział prezentuje zarówno podstawowe aspekty techniczne, takie jak formaty plików czy atrybuty znacznika <img>, jak i kwestie związane z semantyką, dostępnością oraz optymalizacją.

Wstawianie obrazów w HTML

Najpopularniejszym i najprostszym sposobem umieszczenia grafiki w dokumencie HTML pozostaje znacznik **<img>**. Jest **elementem pustym**, co oznacza, że nie posiada pary zamykającej (np. </img>). Jego użycie sprowadza się do zadeklarowania atrybutów odpowiadających za lokalizację pliku oraz opis dodatkowy.

```
  <img src="images/przyklad.jpg" alt="Opis grafiki" title="Tytuł obrazka"
width="300" height="200">
```

- **src** (*source*) – wskazuje ścieżkę do pliku graficznego (może być ścieżka względna, np. images/przyklad.jpg, lub absolutna, np. https://www.przyklad.pl/baner.jpg).
- **alt** (*alternative text*) – kluczowy atrybut z perspektywy dostępności (*accessibility*) i pozycjonowania (SEO). Tekst wpisany do alt jest odczytywany przez czytniki ekranu (np. wykorzystywane przez osoby niewidome) i wyświetlany w sytuacji, gdy grafika nie może zostać załadowana.
- **title** – wyświetla dodatkowy opis lub podpowiedź (tzw. *tooltip*) po najechaniu kursorem na grafikę.
- **width** i **height** – określają wymiary obrazka w pikselach. W praktyce często rezygnuje się z bezpośredniego definiowania szerokości i wysokości w HTML na rzecz elastycznych stylów w CSS.

Warto pamiętać o semantyce i funkcji atrybutu alt. Na przykład, jeżeli grafika jest czysto dekoracyjna i nie wnosi treści informacyjnej, można pozostawić pusty alt="", aby czytniki ekranu mogły ją pominąć.

Formaty plików graficznych

W kontekście projektowania stron internetowych wyróżnia się kilka kluczowych formatów plików graficznych, z których każdy sprawdza się w nieco innych zastosowaniach.

1. **JPEG (JPG)**
    - Najczęściej używany do fotografii oraz obrazów o bogatej kolorystyce.
    - Umożliwia stratną kompresję, co przekłada się na mniejszy rozmiar pliku, ale może skutkować utratą jakości.
    - Dobrze sprawdza się w sytuacjach, gdy priorytetem jest redukcja rozmiaru pliku.
2. **PNG (*Portable Network Graphics*)**
    - Obsługuje kompresję bezstratną i kanał przezroczystości (*alpha channel*).
    - Zalecany przy logotypach, ikonach, wykresach i wszelkich grafikach wymagających ostrości krawędzi.
    - Lepszy wybór w przypadku obrazów z tekstem lub silnymi kontrastami.
3. **GIF (*Graphics Interchange Format*)**
    - Formatuje obrazy w ograniczonej palecie (do 256 kolorów).
    - Zyskał popularność przede wszystkim z uwagi na możliwość tworzenia krótkich animacji (tzw. animowane gify).
    - Mniej przydatny do wyświetlania fotografii, ale nadal stosowany w prostych animowanych banerach czy ikonach.

Obecnie można również spotkać **format WebP** (wspierany przez nowsze przeglądarki), który oferuje wysoką kompresję przy zachowaniu dobrej jakości, a także format **SVG** (ang. *Scalable Vector Graphics*), idealny do prezentacji elementów wektorowych, takich jak logotypy lub ikony skalowane w nieskończoność. Wybór optymalnego formatu wpływa znacząco na wydajność strony i doświadczenia użytkowników – grafiki o zbyt dużej wadze mogą wydłużać czas ładowania witryny.

Podpisy i semantyka obrazów

<figure> i <figcaption>

Wprowadzony wraz z HTML5 zestaw znaczników **<figure>** i **<figcaption>** pozwala na semantyczne grupowanie obrazu (i/lub innej zawartości multimedialnej) wraz z jego podpisem. Przykład:

```
  <figure>
    <img src="images/diagram.png" alt="Schemat działania aplikacji">
    <figcaption>Schemat przedstawiający główny mechanizm działania
aplikacji.</figcaption>
  </figure>
```

- **<figure>** – blok, w którym umieszczamy grafikę (lub np. tabelę, wykres, kod) stanowiącą samodzielny element treści.
- **<figcaption>** – podpis opisujący dany element (np. tytuł, źródło).

Takie rozwiązanie jest rekomendowane w sytuacjach, gdy obraz pełni istotną rolę w zrozumieniu przekazu lub wymaga dodatkowego komentarza. Zabieg ten wspiera dobre praktyki semantyki i dostępności.

Optymalizacja obrazów

W projektach, szczególnie tych rozbudowanych, znaczną wagę przywiązuje się do optymalizacji w celu zapewnienia szybkiego ładowania stron. Poniżej zaprezentowano kilka kluczowych zasad:

1. **Kompresja** – w przypadku JPEG można dostosować stopień kompresji (tzw. *quality*). Z kolei PNG z reguły zapewnia bezstratną kompresję, natomiast warto korzystać z narzędzi redukujących nieużywane informacje (np. pngquant).
2. **Odpowiedni rozmiar** – czasem wystarczającym rozwiązaniem jest zmniejszenie wymiarów pliku w edytorze graficznym (np. GIMP, Photoshop), aby dostosować obraz do faktycznych potrzeb projektu.
3. **Lazy loading** – strategia polegająca na ładowaniu obrazów dopiero w momencie, gdy pojawiają się w widocznym obszarze ekranu (tzw. *viewport*). Znacząco skraca to czas pierwszego renderowania strony.
4. **Formy wektorowe** – w przypadku ikon lub logo dobrze rozważyć zastosowanie formatu SVG, który jest skalowalny i zwykle ma mniejszy rozmiar niż odpowiednik PNG przy porównywalnej jakości.

Dbałość o te elementy stanowi ważny krok w kierunku budowania profesjonalnych, wydajnych witryn internetowych.

Ćwiczenia

1. **Osadź grafikę w HTML**
    - Utwórz nowy plik, np. grafika.html.
    - Wstaw do niego co najmniej dwa obrazy, jeden w formacie JPEG, drugi w PNG.
    - Każdemu przypisz atrybut alt (nadaj mu sensowny opis), tytuł title oraz określ wymiary w pikselach lub pozostaw ich naturalny rozmiar do stylowania w CSS.
2. **Zastosuj <figure> i <figcaption>**
    - Zaprojektuj sekcję, w której umieścisz obraz wraz z podpisem pod nim.
    - W ten sposób przećwiczysz semantyczne sposoby reprezentacji danych w HTML5.
3. **Porównaj rozmiar i jakość plików**
    - Przygotuj dwa obrazy w różnych formatach (np. JPEG i PNG), a następnie sprawdź, jak różnią się ich rozmiary.
    - Zastanów się, który format jest bardziej odpowiedni w konkretnych sytuacjach (np. fotografia produktu vs. schematyczny diagram).

Tabele w HTML

Ten rozdział poświęcony jest zagadnieniu prezentacji danych w formie tabel. Jest to kolejny krok w stronę kompleksowego zrozumienia struktury dokumentu HTML, który doskonale uzupełnia dotychczas nabyte umiejętności tworzenia i formatowania treści czy osadzania grafiki. Choć w wielu współczesnych projektach coraz częściej wykorzystuje się zaawansowane narzędzia i biblioteki do prezentowania danych, znajomość podstaw tabel HTML pozostaje nieoceniona. W niniejszym rozdziale przyjrzymy się podstawowym znacznikom pozwalającym na budowanie tabel, omówimy ich znaczenie semantyczne, a także zaprezentujemy dobre praktyki związane z czytelnością i dostępnością.

Podstawowe znaczniki tabel

Struktura elementarna: <table>, <tr>, <td>, <th>

- **<table>** – główny kontener, w którym umieszczamy całą tabelę.
- **<tr>** (*table row*) – reprezentuje pojedynczy wiersz tabeli.
- **<td>** (*table data*) – odpowiada za pojedynczą komórkę (cechuje się zawartością tekstową lub innymi elementami HTML).
- **<th>** (*table header*) – działa analogicznie do <td>, jednak jest przeznaczony na nagłówki kolumn bądź wierszy, co semantycznie wyróżnia je w tabeli.

Przykład minimalnej tabeli:

```
<table>
  <tr>
    <th>Nagłówek 1</th>
    <th>Nagłówek 2</th>
  </tr>
  <tr>
    <td>Wartość 1</td>
    <td>Wartość 2</td>
  </tr>
</table>
```

Nagłówek 1	Nagłówek 2
Wartość 1	Wartość 2

Dzięki th przeglądarka oraz czytniki ekranu mogą lepiej zinterpretować, że dana komórka pełni funkcję nagłówka, co jest istotne z punktu widzenia dostępności.

Dodatkowe sekcje: <thead>, <tbody>, <tfoot>

- **<thead>** (*table head*) – część tabeli przeznaczona na wiersze nagłówkowe.
- **<tbody>** (*table body*) – zawiera główną treść tabeli (wiersze z danymi).
- **<tfoot>** (*table foot*) – sekcja zamykająca, w której można umieścić np. podsumowania, sumy czy statystyki.

Tego typu podział ułatwia obsługę i stylowanie tabel w języku CSS, a także pozwala czytnikom ekranu (ang. *screen readers*) wyraźniej zidentyfikować strukturę danych. Przykład:

```
<table>
  <thead>
    <tr>
      <th>Produkt</th>
      <th>Cena</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Książka</td>
      <td>35 PLN</td>
    </tr>
    <tr>
```

```
      <td>Długopis</td>
      <td>2 PLN</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <th>Razem</th>
      <td>37 PLN</td>
    </tr>
  </tfoot>
</table>
```

Produkt	Cena
Książka	35 PLN
Długopis	2 PLN
Razem	37 PLN

Choć sekcje <thead> i <tfoot> nie są obligatoryjne w najprostszych przypadkach, to w rozbudowanych projektach zdecydowanie ułatwiają porządkowanie struktury.

Łączenie komórek: colspan i rowspan

W pewnych sytuacjach pojawia się potrzeba połączenia kilku komórek w poziomie (np. w wierszu nagłówkowym obejmującym wiele kolumn) lub w pionie. Do tego celu służą atrybuty **colspan** (łączenie w poziomie) oraz **rowspan** (łączenie w pionie):

```
<table>
  <tr>
    <th colspan="2">Nagłówek łączony</th>
  </tr>
  <tr>
    <td>Komórka 1</td>
    <td>Komórka 2</td>
  </tr>
</table>
```

Nagłówek łączony	
Komórka 1	Komórka 2

Przy wykorzystaniu colspan="2" nagłówek zostanie rozciągnięty na dwie kolumny, a analogiczne działanie w pionie (scalanie komórek w obrębie kolumny) uzyskuje się przez rowspan. Podobna funkcjonalność bywa pomocna, gdy chcemy sformatować np. tabelę podsumowującą wyniki eksperymentu w pracy inżynierskiej.

Właściwości wizualne tabel

W dawniejszych czasach HTML-a powszechnie stosowano atrybuty takie jak border="1", cellpadding, cellspacing czy bgcolor. Obecnie zaleca się jednak kontrolowanie wyglądu tabel poprzez **CSS**, co pozwala na oddzielenie warstwy semantycznej (treść i struktura) od prezentacyjnej (style):

```
table {
  border-collapse: collapse; /* łączy krawędzie sąsiadujących komórek */
  width: 100%;
}

th, td {
  border: 1px solid #ccc;
  padding: 8px;
  text-align: left;
}

thead {
  background-color: #f2f2f2;
}
```

Nagłówek łączony	
Komórka 1	Komórka 2

Takie podejście zapewnia większą elastyczność, łatwiejszą konserwację kodu i zgodność z dobrymi praktykami tworzenia stron internetowych.

Dobre praktyki i semantyka

1. **Czytelne nagłówki** – zawsze warto używać <th> zamiast <td> w wierszach i kolumnach nagłówkowych, co znacząco ułatwia odbiór danych.
2. **Zwięzła treść** – opisanie komórek w sposób zwięzły, a jednocześnie trafny, zwiększa czytelność, zarówno w przypadku wyświetlania przez przeglądarki graficzne, jak i narzędzia wspomagające.

3. **Minimalizm wizualny** – unikajmy przeładowania tabel zbyt wieloma kolorami i ramkami, by nie zaciemnić przedstawianych danych.

Ćwiczenia

1. **Prosta tabela z nagłówkiem**
   - Stwórz nowy plik HTML (np. tabela.html) i zbuduj tabelę prezentującą kilka przykładowych danych (np. lista przedmiotów z ceną).
   - Zastosuj semantyczne elementy <thead>, <tbody>, a w <th> użyj scope="col".
2. **Łączenie komórek**
   - W tej samej tabeli połącz dwa nagłówki w poziomie (atrybut colspan) i przećwicz łączenie komórek w pionie (rowspan).
   - Dodaj do jednej z komórek wartości objaśnienia, np. „Dane łączone w wierszu 2 i 3”.
3. **Stylizacja w CSS**
   - Utwórz plik styles.css i zdefiniuj podstawowe reguły wpływające na wygląd tabeli (szerokość, ramki, odstępy).
   - Zaimportuj plik CSS do dokumentu HTML za pomocą <link rel="stylesheet" href="styles.css">.

Wykonanie tych ćwiczeń pozwoli w praktyce zrozumieć mechanizmy konstrukcji tabeli w HTML, a także przećwiczyć korzystanie z atrybutów semantycznych i możliwości stylizowania.

Elementy blokowe i liniowe, klasy i identyfikatory

Tym razem skoncentrujemy się na dwóch istotnych obszarach związanych z językiem HTML. Po pierwsze, omówimy koncepcję **elementów blokowych** oraz **liniowych**, co pozwali zrozumieć ich wpływ na układ strony i pozycjonowanie poszczególnych fragmentów treści. Po drugie, znaczenie **klas** oraz **identyfikatorów (ID)**, które umożliwiają elastyczne i wydajne zarządzanie stylami w języku CSS, a także pomagają w selektywnym odwoływaniu się do konkretnych elementów w dokumentach HTML.

Elementy blokowe a elementy liniowe

Podstawowe różnice

- **Elementy blokowe**

  - Zajmują całą dostępną szerokość kontenera, w którym się znajdują (np. szerokość strony lub elementu nadrzędnego).
  - Zawsze rozpoczynają się od nowej linii i powodują przeniesienie kolejnych elementów niżej (co wizualnie formuje tzw. blok).
  - Mogą zawierać inne elementy blokowe i liniowe.

Przykładami elementów blokowych są: , <p>, <h1>–<h6>, <ul>, <table>.

- **Elementy liniowe**

  - Zajmują tylko tyle miejsca, ile wynika z ich treści (np. długość tekstu czy wielkość ikony).
  - Nie powodują automatycznego przejścia do nowej linii.
  - Mogą zawierać tylko elementy liniowe (z kilkoma wyjątkami, jak np. <a> mogące obejmować także niektóre elementy blokowe w HTML5, jednak w praktyce stosuje się to z rozwagą).

Przykładowymi elementami liniowymi są: , <a>, , , <img>.

Znaczenie w układzie strony

Zrozumienie, że np. ** ** (typowo element blokowy) zajmuje całą dostępną szerokość i „spycha" kolejny element poniżej, jest kluczowe w budowaniu layoutów stron. Elementy liniowe, takie jak ****, można natomiast umieszczać wewnątrz akapitów, aby wyróżnić wybrane fragmenty tekstu, nie przerywając ciągłości linii. Dzięki temu łatwiej tworzyć złożone projekty, w których chcemy dzielić treść na sekcje (bloki) oraz modyfikować konkretny tekst czy grafikę wewnątrz tych sekcji (*inline*).

Klasy (*class*) i identyfikatory (id)

Klasy (class)

- **Czym jest klasa w HTML?**

  Klasa to atrybut, którego można użyć wielokrotnie w różnych miejscach w obrębie jednego dokumentu HTML.

  - **class="nazwa-klasy"**

- **Zastosowanie**:
  - Nadawanie jednego stylu wspólnego dla wielu elementów (np. .czerwony-tekst, .duzy-naglowek).
  - Grupowanie tematyczne (np. .oferta, .product-box) w celu selektywnego stylowania i jednoczesnego zachowania porządku.
- **Przykład**:

```
  <p class="czerwony-tekst">Ten akapit będzie wyróżniony kolorem
czerwonym.</p>
  <span class="czerwony-tekst">Ten fragment wciąż należy do tej samej
klasy.</span>
```

W arkuszu stylów (CSS) wystarczy użyć selektora .czerwony-tekst, aby zdefiniować reguły dla wszystkich elementów, które posiadają taką klasę.

Identyfikatory (id)

- **Charakterystyka identyfikatora**
  - id musi być unikatowe w obrębie całego dokumentu – nie powinno się powtarzać.
  - Nadajemy je elementom, które wymagają specyficznego, unikalnego stylu lub są kluczowe dla mechaniki strony (np. #header, #footer, #formularz-kontaktowy).

- **Przykład**:

```
<div id="nawigacja">
  <!-- Element odpowiadający za menu nawigacyjne strony -->
</div>
```

W arkuszu stylów użyjemy selektora **#nawigacja** w celu stylizowania konkretnego elementu, np.:

```
#nawigacja {
    background-color: #333;
    color: #fff;
  }
```

Różnice w praktyce

Zarówno class jak i id pełnią podobną funkcję – służą do odwoływania się do konkretnego elementu lub ich grupy. Kluczowa różnica polega na tym, że klasę można stosować wiele razy, zaś identyfikator powinien być unikatowy. To sprawia, że class jest bardziej elastyczna w projektach wielokrotnie używających tych samych reguł CSS, a id jest idealne, gdy potrzebujemy jednego specyficznego stylu bądź chcemy łatwo nawigować do wybranego fragmentu strony (np. w linku kotwicowym href="#formularz-kontaktowy").

Wprowadzenie do CSS: selektory, Box Model, podstawy stylowania

Ta część materiału stanowi przełomowy punkt w kursie, gdyż wprowadzamy kolejny kluczowy element w procesie projektowania stron – **kaskadowe arkusze stylów (CSS)**. O ile HTML odpowiada za strukturę i semantykę treści, o tyle CSS nadaje stronom finalny wygląd, pozwalając kontrolować kolorystykę, typografię, rozmieszczenie elementów oraz wszelkie inne aspekty wizualne. Niniejszy dział przedstawia najważniejsze informacje związane z rozpoczęciem pracy z CSS, a jednocześnie zawiera wskazówki i przykłady ułatwiające zrozumienie tej technologii w przystępny sposób.

Rola i zastosowanie CSS w projektowaniu stron WWW

Czym jest CSS?

CSS (*Cascading Style Sheets*) to język służący do definiowania sposobu prezentacji treści na stronach internetowych. Dzięki CSS możliwe jest modyfikowanie koloru i rozmiaru tekstu, tła, obramowań, a także rozmieszczanie elementów w układzie kolumnowym czy tworzenie bardziej złożonych kompozycji graficznych.

Dlaczego „kaskadowe”?

Przymiotnik „kaskadowe” wynika z hierarchicznego (kaskadowego) sposobu interpretacji arkuszy stylów. Oznacza to, że styl zadeklarowany w jednym miejscu może zostać nadpisany (przezwyciężony) przez bardziej szczegółową lub wyżej priorytetyzowaną definicję w innym miejscu. Kolejność i specyficzność reguł odgrywają tu ważną rolę, stąd mowa o kaskadzie, w której poszczególne zasady mogą „spływać” przez różne poziomy deklaracji.

Metody osadzania CSS w dokumencie HTML

1. **Osadzanie w linii (ang. *inline styles*)**
    - Stosowanie atrybutu style="..." w obrębie konkretnego znacznika HTML, np.

```
<p style="color: red;">Przykładowy tekst w kolorze czerwonym</p>
```

    - Metoda szybka i prosta w użyciu, jednak utrudniająca utrzymanie spójności stylów w większych projektach.
2. **Style osadzone (ang. *embedded styles*)**
    - Umieszczanie reguł CSS w sekcji <head> dokumentu, wewnątrz znacznika <style>.
    - Użyteczne w przypadku niewielkich stron lub gdy chcemy przetestować konkretne reguły bezpośrednio w dokumencie HTML.

```
<head>
  <style>
    p { color: blue; }
  </style>
</head>
```

3. **Zewnętrzny arkusz stylów (ang. *external stylesheet*)**
    - Najbardziej rekomendowana forma w większości przypadków – stylizację umieszcza się w osobnym pliku .css, a w dokumencie HTML dodaje się link:

```
<link rel="stylesheet" href="styles.css">
```

- Pozwala na łatwiejszą konserwację kodu, ponowne wykorzystanie reguł w wielu plikach oraz utrzymanie logicznego podziału między strukturą (HTML) a wyglądem (CSS).

W wszelkich zadaniach o większej skali najczęściej stosuje się zewnętrzne arkusze stylów, ponieważ promują modularność i pozwalają na zachowanie wysokiej czytelności kodu.

Podstawowe selektory w CSS

1. **Selektor tagu (elementu)**
   - Odnosi się bezpośrednio do nazwy znacznika w HTML, np.

```
p {
    font-size: 16px;
    line-height: 1.5;
  }
```

   - Wszystkie paragrafy (<p>) w dokumencie zostaną sformatowane zgodnie z powyższymi zasadami.
2. **Selektor klasy**
   - Używamy kropki . przed nazwą klasy, np.:

```
.czerwony-tekst {
    color: red;
  }
```

   - Dotyczy wszystkich elementów w HTML, które posiadają atrybut class="czerwony-tekst" (patrz poprzednie spotkania).
3. **Selektor identyfikatora**
   - Poprzedzony znakiem #, np.:

```
#nawigacja {
    background-color: #333;
    color: #fff;
  }
```

   - Odnosi się wyłącznie do elementu o atrybucie id="nawigacja". Pamiętajmy, że id musi być unikatowe w dokumencie.

4. **Selektory zagnieżdżone (*descendant selectors*)**
    - Pozwalają na bardziej precyzyjne targetowanie elementów. Przykładowo:

```
div p {
    color: #555;
  }
```

    - Reguła ta mówi: „Nadaj kolor #555 wszystkim akapitom (<p>) znajdującym się wewnątrz elementu ”.

Prócz nich istnieje wiele bardziej zaawansowanych selektorów (np. selektory potomków, rodzeństwa, pseudoklasy), jednak na początkowym etapie nauki warto skupić się na fundamentach, by móc efektywnie tworzyć stylizowane strony.

Box Model – fundament CSS

Box Model (model pudełkowy) definiuje, jak przeglądarki obliczają rozmiary i przestrzeń wokół elementów HTML. Zrozumienie go jest kluczowe przy układaniu layoutu:

- ***content*** – obszar treści (np. tekst, obraz).
- ***padding*** – wewnętrzne wypełnienie między treścią a krawędzią elementu.
- ***border*** – obramowanie.
- ***margin*** – zewnętrzne odstępy wokół elementu, oddzielające go od sąsiednich elementów.

W praktyce, jeśli ustawimy width: 200px; dla danego elementu, to rzeczywista całkowita szerokość pudełka może być większa, gdy doliczymy do niej padding, border i margin. Przykładowa deklaracja:

```
div {
    width: 200px;
    padding: 20px;
    border: 2px solid #000;
    margin: 10px;
  }
```

Wygląd i zachowanie tego „pudełka” w obrębie strony będą zależały właśnie od ustawionych wartości i sposobu interpretacji modelu pudełkowego (Domyślnie: box-sizing: content-box; alternatywnie: box-sizing: border-box).

Ćwiczenia

1. **Stworzenie zewnętrznego pliku CSS**
    - Załóż plik style.css i podłącz go do nowo utworzonej strony HTML (np. index.html) poprzez <link rel="stylesheet" href="style.css">.
    - W pliku CSS zdefiniuj proste reguły, np. body { font-family: Arial, sans-serif; }.
2. **Wykorzystanie różnych selektorów**
    - Napisz selektor tagu (np. h1 { color: blue; }),
    - selektor klasy (.podkreslony { text-decoration: underline; }),
    - selektor identyfikatora (#stopka { text-align: center; }).
    - Następnie zaimplementuj te klasy i identyfikatory w kodzie HTML.
3. **Testowanie Box Model**
    - Utwórz z tekstem i nadaj mu wyraźne padding i border.
    - Obserwuj, jak zmiana wartości margin wpływa na odstępy od sąsiednich elementów.
    - Rozważ zastosowanie box-sizing: border-box; i sprawdź, jak modyfikuje to wymiary elementu.

Wykonanie tych ćwiczeń pozwoli na praktyczne zrozumienie, dlaczego CSS nazywamy kaskadowym oraz jak ważne jest rozróżnianie podstawowych selektorów i stosowanie Box Model przy projektowaniu layoutów.

Pseudoklasy w CSS

Czym są pseudoklasy?

Pseudoklasy (ang. *pseudo-classes*) to rozszerzenie koncepcji selektorów w CSS, umożliwiające stylowanie elementów w zależności od ich specyficznych stanów, pozycji w drzewie dokumentu lub interakcji użytkownika. Dzięki nim możemy zmieniać wygląd elementu np. w momencie najechania kursorem, zaznaczenia, kliknięcia lub kiedy pełni on określoną rolę w strukturze strony.

- Składnia pseudoklasy polega na dodaniu dwukropka (:) po nazwie selektora, a następnie nazwy pseudoklasy, np.:

```
a:hover {
    color: red;
  }
```

- Pseudoklasy przydają się szczególnie w projektach, w których chcemy dynamicznie reagować na działania użytkownika (np. najechanie myszką, fokus w polu formularza) lub wyróżniać pierwszy/ostatni element listy bądź poszczególne fragmenty formularza.

Najpopularniejsze pseudoklasy

1. **:hover**
    - Stylowany element, gdy kursor myszy znajduje się nad nim.
    - Najczęściej używany dla linków (np. zmiana koloru, podkreślenie), ale może dotyczyć też innych elementów (div, button itp.).
    - Przykład:

```
a:hover {
    text-decoration: underline;
    color: #f00;
  }
```

2. **:focus**
    - Stan aktywnego fokusu, np. gdy użytkownik kliknie w pole formularza lub przejdzie do niego klawiszem TAB.
    - Pozwala na podkreślenie aktualnie aktywnego elementu, co jest ważne z punktu widzenia dostępności.
    - Przykład:

```
input:focus {
    outline: 2px solid #00f;
  }
```

3. **:active**
    - Zastosowanie w chwili „aktywności” elementu, np. gdy użytkownik klika link czy przycisk i jeszcze nie zwolnił przycisku myszy.
    - Dobrze sprawdza się np. w tworzeniu wrażeń interaktywnych (przycisk może się „wciskać”).

- Przykład:

```
button:active {
    background-color: #ccc;
    transform: scale(0.98);
  }
```

4. **:visited**
   - Dotyczy linków, które zostały już odwiedzone przez użytkownika.
   - Pomaga w odróżnieniu wcześniej otwieranych stron.
   - Przykład:

```
a:visited {
    color: #800080; /* fioletowy */
  }
```

5. **:first-child, :last-child, :nth-child()**
   - Pseudoklasy umożliwiające stylowanie elementu w zależności od jego pozycji wśród rodzeństwa w drzewie DOM.
   - :first-child – odnosi się do pierwszego dziecka w obrębie rodzica.
   - :last-child – odnosi się do ostatniego dziecka.
   - :nth-child() – pozwala na wskazanie konkretnej pozycji bądź wzorca, np. co drugi element listy.
   - Przykłady:

```
li:first-child {
    font-weight: bold;
  }

  li:nth-child(2) {
    color: green;
  }

  li:nth-child(odd) {
    background-color: #fafafa;
  }
```

6. **:not()**
    - Pozwala wykluczyć selektor z określonego wzorca, np.

```
p:not(.wyrozniony) {
    color: #555;
  }
```

    - Stylowanie wszystkich paragrafów z wyjątkiem tych oznaczonych klasą .wyrozniony.

Zasady priorytetów i łączenie pseudoklas

Pseudoklasy można **łączyć** z innymi selektorami (tagu, klasy, identyfikatora), tworząc jeszcze bardziej precyzyjne reguły. Na przykład:

```
button.duzy-przycisk:hover {
    background-color: #ff0;
  }
```

Z uwagi na kaskadowość i specyficzność stylów, może się zdarzyć, że reguły z pseudoklasą zostaną nadpisane przez inne, mocniejsze selektory (np. z użyciem **!important** lub selektorów ID). Dlatego warto dbać o czytelność i unikać niepotrzebnie skomplikowanych „łańcuchów" pseudoklas.

Dlaczego pseudoklasy są ważne?

- **Reagowanie na interakcje**: :hover, :focus, :active wprowadzają elementy dynamiki i lepszą dostępność.
- **Precyzyjne stylowanie**: :first-child, :nth-child() itp. oszczędzają konieczność dodawania zbędnych klas w HTML.
- **Czytelność i oszczędność kodu**: możemy mniej polegać na dodatkowych znacznikach i atrybutach, a bardziej na selektorach i pseudoklasach.

Dzięki pseudoklasom możemy stworzyć stronę, która nie tylko wygląda statycznie ładnie, lecz także w sposób przyjazny reaguje na działania użytkownika. To istotny krok na drodze do profesjonalnych i dopracowanych layoutów, szczególnie w połączeniu z innymi technikami (flexbox, grid, responsywność).

Layout strony, semantyczne sekcje

Ten rozdział stanowi rozszerzenie dotychczas zdobytej wiedzy dotyczącej HTML i CSS o bardziej zaawansowane zagadnienia związane z tworzeniem układu (layoutu) strony. W niniejszym rozdziale skupimy się na semantycznych znacznikach wprowadzonych w HTML5, takich jak <main>, <header>, <nav>, <aside>, <footer>, <article> i <section>. Omówimy również rolę klasycznego kontenera , niezbędnego w wielu konstrukcjach układu. Celem jest pokazanie, jak w sposób świadomy i uporządkowany tworzyć rozbudowane struktury, które będą zarówno czytelne dla deweloperów, jak i przyjazne użytkownikom oraz wyszukiwarkom.

Semantyka i struktura HTML5

Dlaczego semantyka jest ważna?

Semantyczne znaczniki HTML5 dają przeglądarkom i narzędziom (np. czytnikom ekranu) wyraźne wskazówki, jak interpretować poszczególne fragmenty treści. Ułatwiają również utrzymanie czytelności kodu na większych projektach, gdyż zamiast serii z różnymi klasami otrzymujemy czytelne wyróżnienie roli poszczególnych sekcji strony.

Kiedy stosować semantyczne elementy?

Zaleca się, aby w miarę możliwości używać znaczników takich jak <header> czy <nav> zamiast zwykłego , o ile dany element faktycznie pełni określoną funkcję – np. jest nagłówkiem całej strony, listą linków nawigacyjnych czy sekcją poboczną.

Omówienie poszczególnych sekcji

1. **<header>**
    - Przeznaczony na główkę strony lub nagłówek sekcji.
    - Często zawiera logo, tytuł, menu główne bądź inne elementy identyfikacyjne.
2. **<nav>**
    - Przeznaczony dla obszaru nawigacji: menu linków, spis treści, itp.
    - Ułatwia narzędziom wspomagającym (np. screen readerom) szybkie wykrycie obszaru nawigacji w witrynie.
3. **<main>**
    - Rdzeń dokumentu: w tym znaczniku umieszczamy główną treść, unikalną dla danej strony.
    - Zaleca się, by w danej witrynie stosować jeden <main>.
4. **<aside>**
    - Zawiera treści poboczne, uzupełniające główny wątek, np. pasek boczny z reklamami, ciekawostkami, dodatkowymi informacjami.

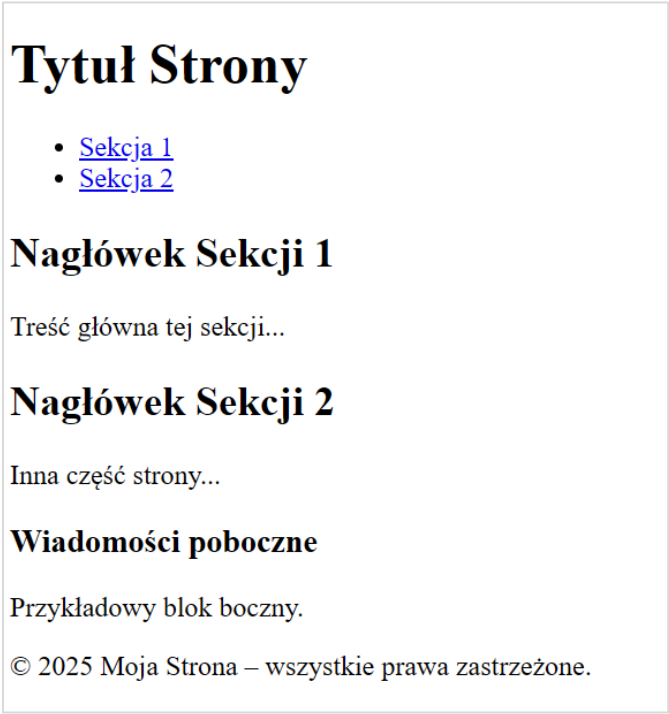
5. **<footer>**
    - Umieszczamy w nim informacje „stopki” strony: prawa autorskie, linki do kontaktu, odnośniki do regulaminu, itp.
    - Można stosować również w obrębie pojedynczych sekcji, np. <article>, jeżeli zachodzi potrzeba dodania stopki konkretnego artykułu.
6. **<article>**
    - Definiuje samodzielny, niezależny fragment treści, np. wpis blogowy, artykuł, komentarz.
    - Może zawierać własne nagłówki, stopki czy sekcje.
7. **<section>**
    - Służy do pogrupowania powiązanych tematycznie elementów w wyodrębnionej sekcji strony.
    - Zalecane przy dzieleniu dłuższego contentu na logiczne części (np. rozdziały, bloki tematyczne).
8. ** **
    - Uniwersalny kontener blokowy (*block-level*), używany do grupowania elementów w celu zastosowania wspólnych reguł CSS lub dynamicznych operacji w JavaScripcie.
    - Mimo że sam w sobie nie ma znaczenia semantycznego, jest nadal nieodzowny w wielu scenariuszach, szczególnie gdy potrzebujemy złożyć złożony layout.

Tworzenie layoutu strony

Prosta struktura

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Moja strona</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <header>
    <h1>Tytuł Strony</h1>
    <!-- Logo, ewentualnie menu -->
  </header>
```

```
  <nav>
    <ul>
      <li><a href="#sekcja1">Sekcja 1</a></li>
      <li><a href="#sekcja2">Sekcja 2</a></li>
    </ul>
  </nav>

  <main>
    <section id="sekcja1">
      <h2>Nagłówek Sekcji 1</h2>
      <p>Treść główna tej sekcji...</p>
    </section>

    <section id="sekcja2">
      <h2>Nagłówek Sekcji 2</h2>
      <p>Inna część strony...</p>
    </section>

    <aside>
      <h3>Wiadomości poboczne</h3>
      <p>Przykładowy blok boczny.</p>
    </aside>
  </main>

  <footer>
    <p>&copy; 2025 Moja Strona – wszystkie prawa zastrzeżone.</p>
  </footer>

</body>
</html>
```



Powyższy układ obrazuje standardowe zastosowanie semantycznych znaczników HTML5, gdzie <main> trzyma zasadniczą treść, <nav> zapewnia menu nawigacyjne, a <aside> pojawia się jako obszar dodatkowy – często z prawej lub lewej strony.

Rola CSS w rozmieszczeniu elementów

Aby nadać stronie atrakcyjny wygląd i układ (np. kolumnowy), sięgamy po narzędzia CSS, takie jak **display** (m.in. flex, grid), float (dawniej) czy pozycjonowanie. Dzięki nim można zdefiniować, czy <aside> ma być po prawej, <nav> poziomy czy pionowy, i jak zachowuje się layout przy zmianie rozmiaru okna przeglądarki (responsive design).

Ćwiczenia praktyczne

1. **Tworzenie semantycznego szablonu strony**
   - Utwórz plik index.html zawierający strukturę z <header>, <nav>, <main>, <section>, <aside> i <footer>.
   - Wypełnij je przykładową treścią, np. nagłówkami i krótkimi akapitami, tak aby zachować logikę i czytelność.
2. **Podstawowy layout CSS**
   - W pliku style.css ustaw elementy <header>, <nav>, <main>, <aside> i <footer> tak, aby tworzyły sensowny układ (np. nagłówek i stopka na całą szerokość, obok <main> sekcja <aside> z wąską szerokością).

- W tym celu możesz skorzystać z display: flex; i ustawić np. .container { display: flex; } dla głównego kontenera.

3. **Responsywność**
   - Jeżeli czas pozwoli, dodaj proste reguły **media queries** (@media) w CSS, aby kolumny zmieniały się w jeden wąski układ przy mniejszych szerokościach okna.
   - Sprawdź, jak zachowują się poszczególne elementy przy zwężaniu ekranu przeglądarki.

Formularze i komentarze

Ten rozdział koncentruje się na dwóch pozornie niezależnych zagadnieniach, które jednak są niezwykle istotne w procesie tworzenia stron WWW. Po pierwsze, zapoznamy się z **formularzami** – jednym z głównych sposobów interakcji użytkownika z serwisem, umożliwiającym wysyłanie danych (np. w celu logowania, rejestracji, wysyłki wiadomości). Po drugie, omówimy **komentarze** w HTML i CSS, dzięki którym możliwe jest lepsze dokumentowanie kodu, co nabiera szczególnego znaczenia w większych projektach internetowych.

Formularze w HTML

Formularze stanowią podstawę wielu aplikacji internetowych, gdyż pozwalają użytkownikom przekazać dane na serwer, np. podczas wysyłania maila za pomocą formularza kontaktowego, rejestrowania konta czy składania zamówienia w sklepie online.

Podstawowe elementy formularza

1. **Znacznik <form>**
   - Służy do zdefiniowania obszaru formularza, w którym umieścimy pola, przyciski oraz inne elementy.
   - Główne atrybuty to:
     - action – określa adres (URL) strony lub skryptu obsługującego przesyłane dane,
     - method – definiuje metodę przesyłu (np. GET lub POST),
     - enctype – sposób kodowania danych (np. multipart/form-data dla przesyłania plików).

Przykład:

```
<form action="wyslij.php" method="POST">
  <!-- Pola formularza -->
</form>
```

2. **Pola <input>**
    - Podstawowy element do wprowadzania danych, z wieloma wariantami atrybutu type:
        - type="text" – zwykłe pole tekstowe,
        - type="password" – ukrywa wpisywane znaki,
        - type="email" – w HTML5 może wspomagać walidację adresu email,
        - type="checkbox" – pole wyboru (można zaznaczyć wiele opcji),
        - type="radio" – przyciski radiowe (zwykle wybór jednej opcji z wielu),
        - type="file" – umożliwia wgranie pliku z dysku,
        - type="submit" – przycisk zatwierdzający formularz.
3. **Etykiety (ang. *labels*)**
    - Znacznik <label> powiązany z konkretnym polem input (za pomocą for="id-pola").
    - Ułatwia dostępność, np. umożliwiając klikanie w tekst etykiety, co automatycznie ustawia fokus w powiązanym polu:

```
<label for="email">Adres email:</label>
<input type="email" id="email" name="email">
```

4. **<select> i <option>**
    - Lista rozwijana, pozwalająca wybrać jedną (lub kilka – przy multiple) z dostępnych opcji:

```
<select name="kraj">
  <option value="pl">Polska</option>
  <option value="de">Niemcy</option>
  <option value="uk">Wielka Brytania</option>
</select>
```

5. **<textarea>**
    - Pole tekstowe wielolinijkowe, przydatne np. w formularzu kontaktowym, gdy chcemy wprowadzić dłuższą wiadomość.

6. **Przycisk wysyłający**
    - type="submit" – standardowy przycisk wysyłający dane formularza do adresu określonego w action.
    - Można też używać button type="submit" i wewnątrz np. ikon, aby bardziej customizować wygląd.

Walidacja formularzy w HTML5

HTML5 wprowadził szereg atrybutów, które pozwalają wstępnie zweryfikować dane jeszcze przed wysłaniem na serwer, m.in.:

- **required** – użytkownik nie może pominąć pola,
- **pattern="[0-9]{3}"** – dopasowanie do wyrażenia regularnego (np. wymuszenie wpisania dokładnie 3 cyfr),
- **min, max, maxlength** – ograniczenia co do wartości liczbowej, zakresu czy długości tekstu.

Walidacja po stronie klienta (przeglądarki) jest wygodna i oszczędza użytkownikowi zbędnych odświeżeń strony, ale nie zwalnia z konieczności weryfikowania danych na serwerze.

Komentarze w HTML i CSS

Komentarze stanowią nieodzowną część kodu źródłowego, zwłaszcza w zespołowych projektach, gdzie uczestnicy muszą rozumieć zamierzenia i kontekst poszczególnych fragmentów.

Komentarze w HTML

- Składnia:

```
<!-- To jest komentarz w HTML. Nie będzie widoczny w przeglądarce. -->
```

- Zastosowanie:
    - Wyłączanie fragmentów kodu z renderowania (testy, debugowanie).
    - Dokumentowanie sekcji, np. wyjaśnianie, co oznaczają danego rodzaju bloki (<!-- stopka strony -->).

Komentarze w CSS

- Składnia:

```
/* To jest komentarz w CSS. Nie będzie interpretowany przez przeglądarkę. */
```

- Zastosowanie:
    - Komentowanie reguł, opis np. zastosowania poszczególnych selektorów,
    - Czasowe wyłączanie reguł (np. zablokowanie stylu, gdy testujemy inny wariant).

Dzięki komentarzom w HTML i CSS możemy znacznie szybciej odświeżyć sobie założenia projektu po przerwie bądź skutecznie podzielić się wiedzą z innymi osobami uczestniczącymi w pracach nad danym serwisem.

Ćwiczenia

1. **Tworzenie prostego formularza kontaktowego**
    - Utwórz plik formularz.html z podstawową strukturą <form>.
    - Dodaj pola: Imię (text), Email (email), Wiadomość (textarea) oraz przycisk „Wyślij”.
    - Nad etykietami i w pola wprowadź atrybut required dla tych, które są obowiązkowe.
    - W sekcji <head> osadź proste reguły CSS (lub w pliku zewnętrznym), aby formularz był czytelnie ułożony.
2. **Komentarze w HTML i CSS**
    - Zamieść komentarz w dokumencie HTML, opisujący cel formularza (np. <!-- Formularz kontaktowy do przesyłania uwag użytkowników -->).
    - W pliku CSS dopisz komentarze opisujące reguły np. kolorów i rozmiarów czcionki.

Zaawansowane możliwości CSS (czcionki, tekst)

W rozdziale skupiamy się na bardziej zaawansowanych aspektach języka **CSS**, koncentrując się na stylistyce tekstu i typografii oraz na kluczowych wskazówkach dotyczących dalszego rozwoju w tym obszarze.

Czcionki w CSS (font-family, font-size, font-weight, font-style)

Typografia i jej znaczenie

Współczesne witryny internetowe coraz częściej przywiązują ogromną wagę do estetyki wyświetlanego tekstu. Właściwy dobór kroju pisma, rozmiaru czy grubości pozwala nie tylko na zwiększenie czytelności, lecz także na wzmocnienie identyfikacji wizualnej projektu.

1. **font-family**
    - Definiuje rodzinę czcionek, jaką chcemy zastosować w danym elemencie, np.:

```
body {
    font-family: 'Open Sans', Arial, sans-serif;
  }
```

    - Zwykle dodajemy tzw. fallbacki (np. Arial, sans-serif), dzięki czemu jeśli przeglądarka nie znajdzie fontu „Open Sans”, użyje kolejnego dostępnego kroju.

2. **font-size**
    - Odpowiada za rozmiar czcionki. Najczęściej stosuje się wartości w pikselach (px), em, rem lub %:

```
p {
    font-size: 16px;
  }
```

    - W kontekście responsywności i dostępności, popularne są jednostki relatywne (em, rem), które skalują się w zależności od ustawień bazowych przeglądarki.

3. **font-weight**
    - Steruje grubością pisma, np. *normal*, *bold*, bądź wartością liczbową (np. 300, 400, 700):

```
h1 {
    font-weight: 700;
  }
```

4. **font-style**
    - Definiuje styl pisma (np. *normal*, *italic*, *oblique*). W zależności od użytego kroju pisma może niekiedy brakować danego stylu.

Czcionki z Google Fonts

Niezwykle popularnym źródłem bezpłatnych krojów pisma jest Google Fonts. Integracja polega najczęściej na wklejeniu linku w sekcji <head>:

```
  <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;700&display=
swap">
```

a następnie zdefiniowaniu w CSS:

```
  body {
      font-family: 'Open Sans', sans-serif;
    }
```

Ta metoda umożliwia uatrakcyjnienie wyglądu witryny bez konieczności instalowania czcionek po stronie użytkownika.

Zaawansowane formatowanie tekstu w CSS

1. **color** – określa barwę tekstu, np.:

```
p {
    color: #333;
  }
```

2. **text-align** – wyrównanie tekstu (*left*, *center*, *right*, *justify*):

```
.centered-text {
    text-align: center;
  }
```

3. **text-decoration** – dekoracja tekstu (np. *underline*, *line-through*):

```
a {
    text-decoration: none;
  }
```

4. **text-transform** – zmiana wielkości liter (*uppercase*, *lowercase*, *capitalize*):

```
h2 {
    text-transform: uppercase;
  }
```

5. **letter-spacing** i **line-height** – odstępy między znakami i wysokość linii:

```
p {
    letter-spacing: 0.5px;
    line-height: 1.6;
  }
```

6. **text-shadow** – dodawanie cienia do tekstu:

```
h1 {
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
  }
```

Odpowiednie użycie tych właściwości potrafi znacząco poprawić czytelność i atrakcyjność projektowanej strony.

Podstawy JavaScript i praktyczne zastosowania

Język **JavaScript** (w skrócie JS) jest trzecim, nieodłącznym filarem współczesnych aplikacji webowych, pozwalając na tworzenie dynamicznych i interaktywnych witryn.

Wprowadzenie do JavaScript

Charakterystyka i zastosowania

JavaScript to język programowania działający po stronie klienta, interpretowany przez przeglądarki internetowe. Dzięki niemu można:

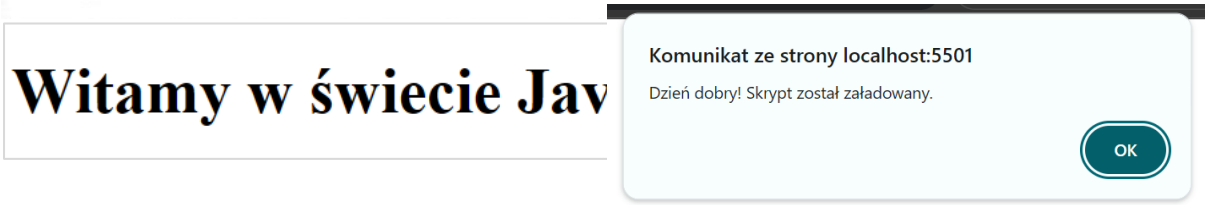
- Reagować na interakcje użytkownika w czasie rzeczywistym (np. kliknięcia, wprowadzanie tekstu w formularzach).
- Modyfikować zawartość i wygląd strony bez konieczności jej ponownego przeładowania (ang. *DOM manipulation*).
- Tworzyć gry, wizualizacje danych, rozbudowane aplikacje typu SPA (*Single-Page Application*).

JavaScript jest również podstawą wielu nowoczesnych frameworków (np. *React*, *Vue*, *Angular*), dzięki czemu stał się kluczowym narzędziem w świecie front-end developmentu.

Włączenie JavaScript do strony

Najprostszym sposobem jest dodanie znacznika <script> w pliku HTML:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Moja Strona z JavaScript</title>
</head>
<body>
  <h1>Witamy w świecie JavaScript!</h1>

  <script>
    alert("Dzień dobry! Skrypt został załadowany.");
  </script>
</body>
</html>
```



Jednak w większych projektach zaleca się umieszczenie kodu w osobnym pliku .js, np. script.js, a następnie zaimportowanie go przez:

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

Podstawy składni i struktury języka

Zmienne i stałe

- **var** – starsza forma deklaracji zmiennej, obecnie rzadziej zalecana (ma zakres funkcyjny i może powodować konflikty nazw).
- **let** – zmienna o zakresie blokowym; preferowana w nowoczesnych projektach.
- **const** – stała, której wartości nie można później zmienić.

Przykłady:

```
let imie = "Jan";
const WERSJA = 1.0;
```

Typy danych

JavaScript charakteryzuje się dynamicznym typowaniem:

- **String** (łańcuch znaków): "Tekst"
- **Number** (liczba): 123, 3.14
- **Boolean**: true lub false
- **Null** i **undefined**: oznaczają brak wartości
- **Object**, **Array**, **Function**: bardziej złożone struktury

```
let wiek = 25;         // Number
let nazwisko = "Kowalski"; // String
let czyAktywny = true; // Boolean
```

Operatory i instrukcje warunkowe

- **Operatory arytmetyczne**: +, -, *, /, % (reszta z dzielenia)
- **Operatory porównania**: === (równość ścisła), !==, <, >, <=, >=
- **Instrukcje warunkowe**: if, else, switch

```
if (wiek >= 18) {
    console.log("Pełnoletni");
  } else {
    console.log("Niepełnoletni");
  }
```

Pętle

- **for** – iteracja po określonej liczbie powtórzeń,
- **while** / **do...while** – iteracja dopóki warunek jest spełniony.

```
for (let i = 0; i < 5; i++) {
    console.log("Iteracja nr: " + i);
  }
```

Funkcje

- Definicja standardowa:

```
function obliczSume(a, b) {
    return a + b;
  }
```

- Funkcja strzałkowa (ang. *arrow function*):

```
const obliczRoznice = (a, b) => {
    return a - b;
  }
  // lub
  const obliczRozniceKrotko = (a, b) => a - b;
```

Funkcje pozwalają na modularyzację kodu i ponowne wykorzystanie logiki.

Manipulacja DOM i zdarzenia

DOM (*Document Object Model*) to struktura dokumentu HTML „reprezentowana" w przeglądarce jako drzewa obiektów. Za pomocą JavaScript możemy dynamicznie tworzyć, usuwać lub modyfikować elementy, reagując na akcje użytkownika.

Wybieranie elementów

- **document.getElementById("nazwa")**
- **document.querySelector(".klasa")** (lub #id, element)

```
const przycisk = document.getElementById("przycisk-klik");
const tytul = document.querySelector("h1");
```

Reagowanie na zdarzenia

- **addEventListener** – umożliwia przypisanie funkcji do konkretnego zdarzenia (kliknięcia, najechania, zmiany rozmiaru okna, itp.).

```
przycisk.addEventListener("click", () => {
    tytul.textContent = "Tekst zmieniony po kliknięciu!";
  });
```

Praktyczne przykłady

1. **Rozwijane menu** – Po kliknięciu przycisku menu, klasa .aktywny dodaje się do <nav>, co powoduje wyświetlenie / ukrycie elementów nawigacji (Skrzypek, 2020).
2. **Walidacja formularza** – Przed wysłaniem danych na serwer możemy sprawdzić poprawność wypełnienia pól (np. email, hasło) i wyświetlić komunikaty.
3. **Prosty slider obrazów** – Zmiana źródła <img> co kilka sekund lub po kliknięciu strzałek.

Ćwiczenia praktyczne

1. **Interaktywny przycisk**
    - W HTML stwórz przycisk o id przycisk-toggle.
    - W JavaScript podłączony do tej samej strony zapisz kod, który po kliknięciu zmienia kolor tła dokumentu (np. toggluje klasę .dark-mode w <body>).

2. **Prosta walidacja formularza**
    - Wykorzystaj istniejący formularz kontaktowy (z poprzednich zajęć).
    - Dodaj skrypt, który po wciśnięciu przycisku „Wyślij” sprawdzi, czy pola nie są puste; jeśli są, wyświetli komunikat ostrzegawczy w elemencie obok pola.
3. **DOM – dodawanie elementów**
    - Stwórz pustą listę <ul id="lista"> w HTML.
    - Utwórz przycisk „Dodaj element”, który po kliknięciu dopisuje nowy <li> z kolejnym numerem do listy.

Niniejszy zakres materiału stanowi dopiero początek przygody z JavaScript. W dalszej kolejności warto przyjrzeć się:

- **Asynchroniczności** (AJAX, fetch API, Promises, async/await) – pozwala na pobieranie i wysyłanie danych bez przeładowywania strony.
- **Modułom** (import/export) – ułatwiają podział kodu na mniejsze pliki i ich ponowne wykorzystanie.
- **Frameworkom i bibliotekom** (React, Vue, Angular, jQuery) – przyspieszają pracę przy większych projektach, zapewniając wiele gotowych rozwiązań.
- **ESNext** – stałe aktualizacje standardu ECMAScript (który definiuje JavaScript) wnoszą kolejne możliwości do języka.

Poznanie podstaw JavaScript – obok umiejętności pisania semantycznego kodu HTML oraz stylów w CSS – czyni z uczestników kursu wszechstronnych twórców front-endu, zdolnych do kreowania nowoczesnych i funkcjonalnych stron internetowych.

Backend stron WWW (*Paweł Sobczak*)

Wprowadzenie do PHP

Język PHP (*Hypertext Preprocessor*) to popularny, otwarty język skryptowy, który jest używany szczególnie do tworzenia aplikacji webowych. PHP działa po stronie serwera i jest często wykorzystywany do generowania dynamicznych stron internetowych oraz zarządzania danymi na stronach. Pełnię swoich możliwości osiąga dzięki integracji z bazami danych, takimi jak MySQL czy PostgreSQL. Znajomość języka PHP stanowi jedną z fundamentalnych kompetencji, które powinien posiadać każdy twórca serwisów internetowych. Jest to nieodzowna umiejętność zarówno przy tworzeniu zaawansowanych, profesjonalnych portali, jak i przy projektowaniu prostszych stron domowych. Dzięki swojej wszechstronności PHP umożliwia realizację szerokiego zakresu funkcjonalności, od dynamicznego generowania treści, po integrację z bazami danych i zaawansowanymi systemami zarządzania treścią. Era statycznego HTML-a należy już do przeszłości – obecnie standardem w branży jest dynamiczne generowanie treści, które zapewnia elastyczność, interaktywność i możliwość dostosowywania stron do indywidualnych potrzeb użytkowników. Co więcej, PHP jako język szeroko wspierany przez społeczność programistów i regularnie aktualizowany, pozostaje jednym z filarów nowoczesnego programowania internetowego, idealnie wpisując się w wymagania współczesnych aplikacji webowych.

Możliwości języka PHP są niezwykle szerokie i wszechstronne. Od podstawowych zadań, takich jak przetwarzanie danych z formularzy w witrynach internetowych, po bardziej zaawansowane operacje, takie jak przetwarzanie tekstu czy budowa złożonych aplikacji współpracujących z dużymi bazami danych. Język PHP wspiera wiele protokołów sieciowych, w tym NNTP, SMTP, POP3 i IMAP, a także umożliwia komunikację sieciową za pomocą gniazd (ang. *sockets*). Dodatkowo, dzięki jego funkcjonalnościom, można dynamicznie generować obrazy, dokumenty PDF, czy dane w formacie XML.

Jak działa język PHP?

PHP działa na zasadzie **skryptu po stronie serwera**. Oznacza to, że kod PHP jest wykonywany na serwerze, a wynik tej operacji (zazwyczaj HTML lub inny format danych) jest wysyłany do przeglądarki użytkownika. Cały proces wygląda następująco:

1. **Przeglądarka wysyła żądanie do serwera** – Gdy użytkownik wchodzi na stronę internetową, przeglądarka wysyła żądanie do serwera, na którym znajduje się strona (np. kliknięcie w link lub wpisanie adresu URL).
2. **Serwer odbiera żądanie i przetwarza skrypt PHP** – Na serwerze zainstalowane jest oprogramowanie, które interpretuje kod PHP (np. serwer Apache z wtyczką PHP). Serwer przetwarza skrypt PHP zawarty w pliku (np. index.php), wykonując zapisane w nim instrukcje.
3. **PHP przetwarza dane** – Skrypt PHP może wykonać różnorodne operacje, takie jak:
    - Przetwarzanie danych z formularzy (np. zapisanie danych do bazy danych).
    - Pobieranie i wyświetlanie danych z bazy danych (np. wyświetlenie listy produktów z bazy).
    - Generowanie dynamicznego HTML-a na podstawie warunków, zmiennych i danych wejściowych.
4. **Wynik zwrócony do przeglądarki** – Po przetworzeniu skrypt PHP generuje odpowiedź, którą najczęściej jest zwykły kod HTML, CSS lub JavaScript. Ta odpowiedź jest przesyłana do przeglądarki użytkownika.
5. **Przeglądarka wyświetla stronę** – Przeglądarka odbiera dane i wyświetla zawartość strony, którą użytkownik widzi w swoim oknie przeglądarki.

Przykład prostego procesu:

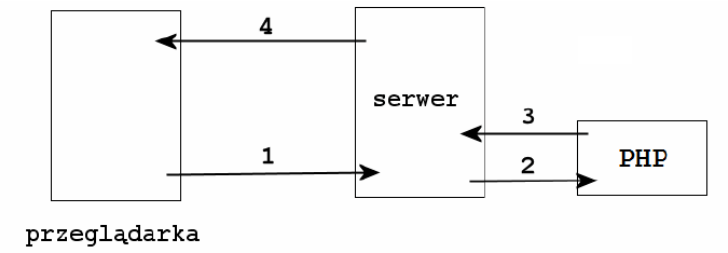
1. Użytkownik wchodzi na stronę www.przyklad.com.
2. Serwer odbiera żądanie i znajduje plik index.php.
3. Skrypt PHP w pliku index.php łączy się z bazą danych, pobiera listę produktów, a następnie generuje HTML z danymi produktów.

4. PHP wysyła ten HTML do przeglądarki użytkownika.
5. Przeglądarka wyświetla listę produktów.

Kluczowe elementy:

- **Interpretacja**: PHP jest językiem interpretowanym, co oznacza, że kod jest wykonywany linia po linii w czasie rzeczywistym, a nie kompilowany przed uruchomieniem.
- **Dynamiczność**: PHP umożliwia generowanie dynamicznych treści na podstawie zmiennych, danych wejściowych (np. z formularzy) oraz danych z baz danych.
- **Bezpieczeństwo**: Ponieważ kod PHP jest wykonywany po stronie serwera, użytkownicy nie mają dostępu do samego kodu źródłowego, co zapewnia pewien poziom bezpieczeństwa.

Zatem zadaniem skryptowego języka PHP jest takie przetworzenie danych, aby została wygenerowana strona zawierająca kod zrozumiały dla przeglądarki WWW (np. kod HTML), co w sposób ideowy przedstawia rys. 2.1.



Rys. 2.1 Ideowa interpretacja działania języka PHP

Źródło: Opracowanie własne.

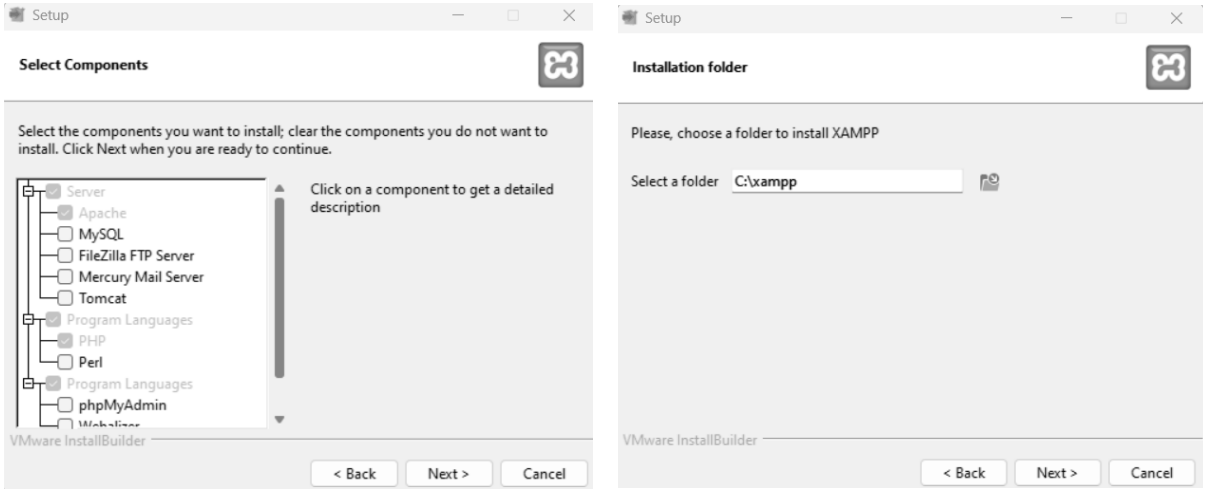
Instalacja oprogramowania

XAMPP to pakiet oprogramowania, który pozwala zamienić komputer domowy w serwer internetowy, który będzie potrafił interpretować i wykonywać skrypty PHP. Składniki pakietu to: serwer WWW Apache, baza danych MySQL oraz język skryptowy PHP (rys. 2.2).

Rys. 2.2 Pakiet oprogramowania XAMPP

Źródło: Opracowanie własne.

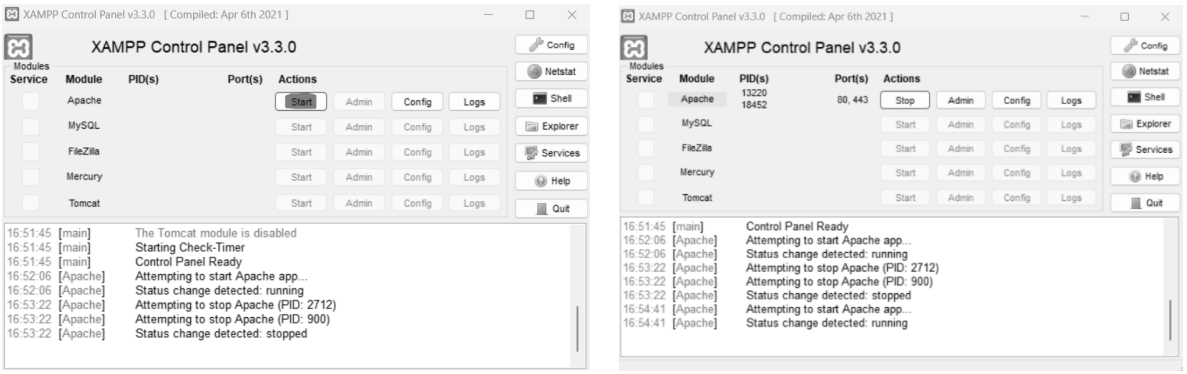
Pakiet oprogramowanie można pobrać ze strony internetowej pod adresem: https://www.apachefriends.org/pl/index.html. Jest to szczególnie wygodny pakiet dla celów testowych, my jednak zainstalujemy tylko Server Apache i język programowania PHP, bazę internetową zainstalujemy sobie za pomocą innego oprogramowania, ale o tym w dalszej części skryptu. Podczas instalacji XAMP odznaczamy wszystko co możliwe i zostawiamy PHP i Apache (rys. 2.3). Katalog instancji wybieramy **c:\xampp**, a język to angielski.



Rys. 2.3 Instalacja pakietu oprogramowania XAMPP

Źródło: Opracowanie własne.

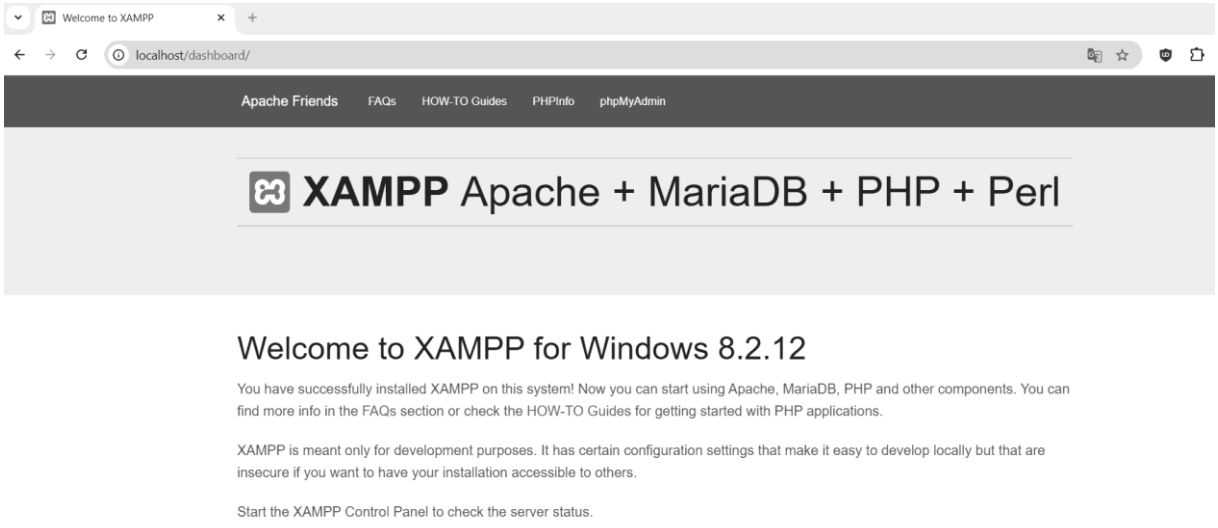
Po zainstalowaniu pakietu możemy uruchomić Apache, jak zostało to pokazane na rys. 2.4.



Rys. 2.4. Uruchomienie serwera Apache

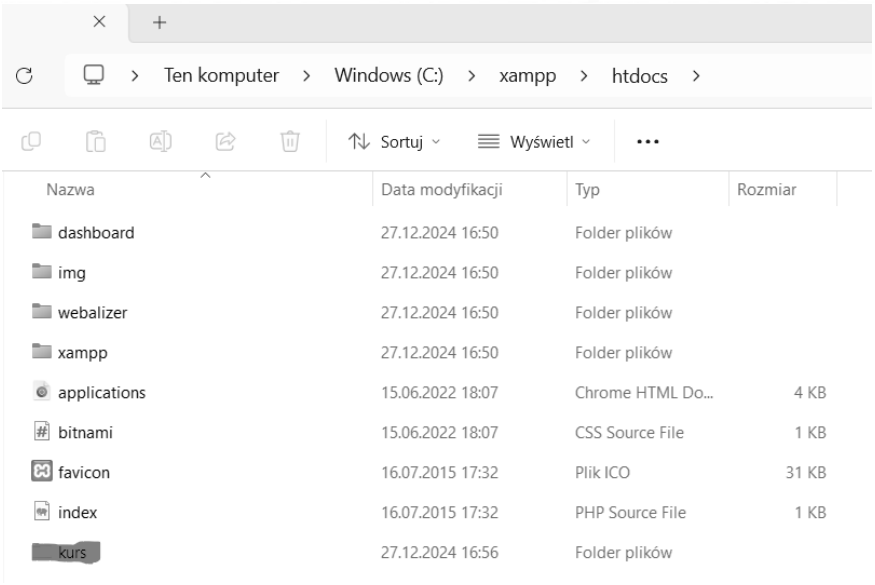
Źródło: Opracowanie własne.

Działanie serwera można sprawdzić, wpisując w pasku adresowym przeglądarki **http://localhost/**, wówczas powinna pojawić się strona startowa widoczna na rys. 2.5. Strona ta jest umieszona w katalogu **c:\xampp\htdocs**. W tym właśnie katalogu utworzymy katalog **kurs** i w nim będziemy trzymać wszystkie pliki (skrypty) PHP, jak zostało to pokazane na rys. 2.6.



Rys. 2.5. Strona startowa serwera Apache

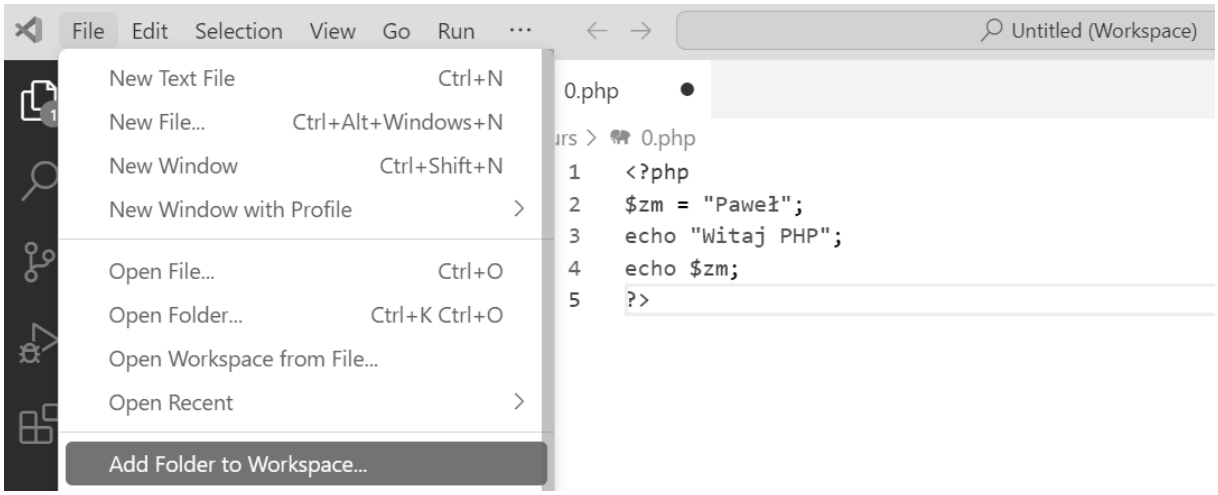
Źródło: Opracowanie własne.



Rys. 2.6. Utworzenie katalogu kurs w katalogu c:\xampp\htdocs

Źródło: Opracowanie własne.

Od teraz wszystkie skrypty będziemy uruchamiać, wpisując w pasku adresowym przeglądarki **http://localhost/kurs/nazwa_skrpytu.php**. W celu napisania pierwszego skryptu PHP można wykorzystać dowolny prosty edytor tekstu, taki jak **Notatnik** czy **Notepad++**. Można również wykorzystać bardziej zaawansowane narzędzie, takie jak **Visual Studio Code**, które można pobrać ze strony: https://code.visualstudio.com/. Po zainstalowaniu edytora warto dodać nasz katalog **kurs** do **Workspace** (obszaru roboczego), jak zostało to pokazane na rys. 2.7.

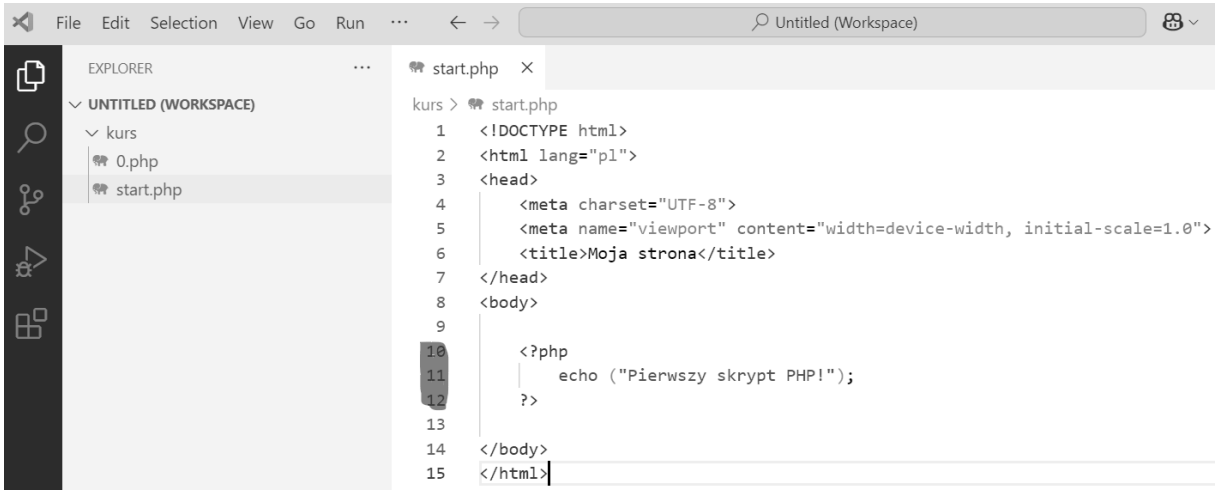


Rys. 2.7. Dodanie katalogu kurs do obszaru roboczego

Źródło: Opracowanie własne.

Pierwszy skrypt PHP

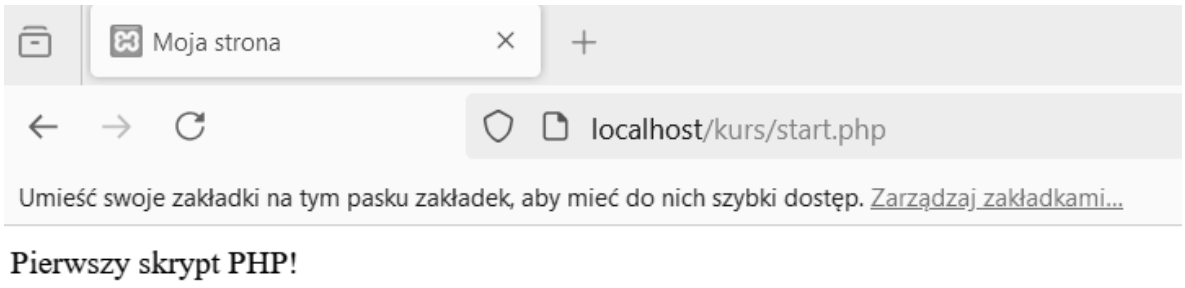
Poznawanie nowego języka programowania najlepiej rozpocząć od przykładu, który spowoduje wyświetlenie dowolnego napisu/tekstu. Każdy skrypt będziemy zapisywać z rozszerzeniem **.php** (nazwa_pliku.php). Skrypt w języku PHP umieszcza się w znacznikach **<?php ?>**, można go łączyć ze znacznikami HTML. Czyli np. w znaczniku <body> możemy umieścić skrypt PHP, ale również w skrypcie PHP możemy używać znaczników HTML.



Rys. 2.8. Pierwszy skrypt PHP wyświetlający napis/tekst

Źródło: Opracowanie własne.

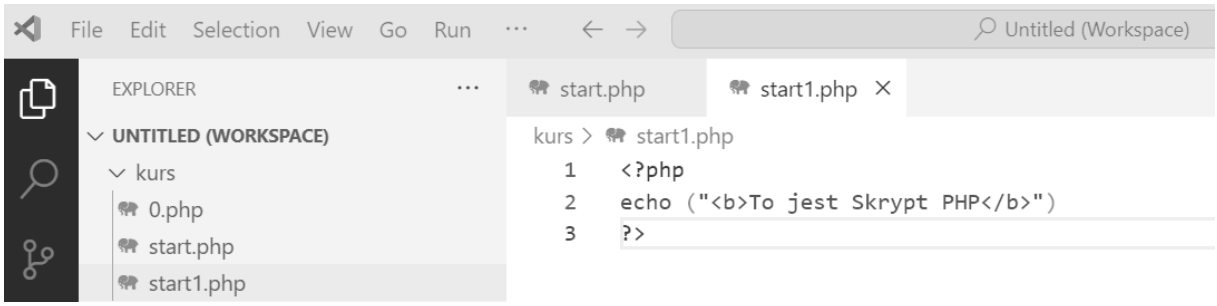
Teraz możemy uruchomić nasz skrypt, który zapisaliśmy pod nazwą **start.php,** wpisując w pasku adresowym przeglądarki **http://localhost/kurs/start.php**. Efekt wykonania skryptu prezentuje rys. 2.9, czyli po prostu wyświetlenie napisu „Pierwszy skrypt PHP!”.



Rys. 2.9. Efekt uruchomienia pierwszego skryptu wyświetlającego napis/tekst

Źródło: Opracowanie własne.

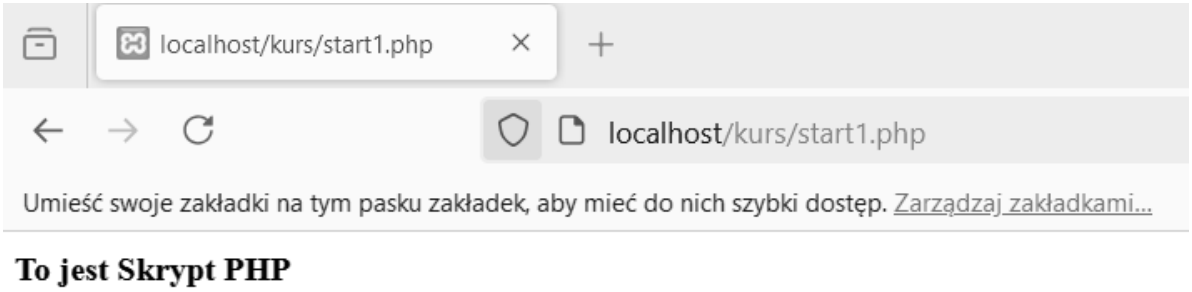
Warto zauważyć, że instrukcja **echo** wyświetla tekst, a na końcu linii znajduje się znak **;**. Prawie po każdej linii w języku PHP stawia się znak średnika. Wszystko to, co znajduje się między znacznikami **<?php ?>**, stanowi kod PHP i jest przetwarzane przez aparat wykonawczy PHP.



Rys. 2.10. Skrypt PHP ze znacznikiem HTML

Źródło: Opracowanie własne.

Tak jak już zostało wspomniane, w skryptach PHP można używać znaczników HTML (rys. 2.10). Efekt wykonania skryptu przedstawia rys. 2.11, czyli wyświetlenie pogrubionego tekstu „To jest skrypt PHP".



Rys. 2.11. Skrypt PHP ze znacznikami HTML

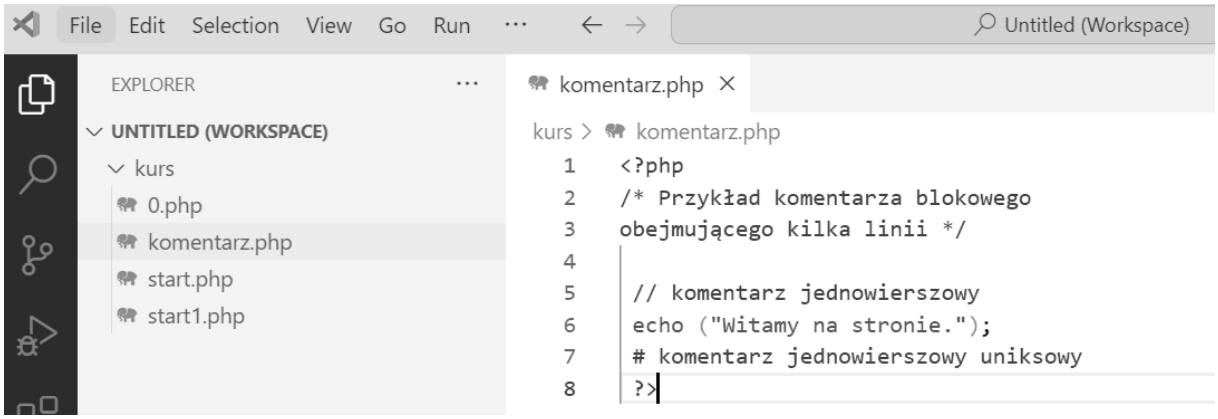
Źródło: Opracowanie własne.

Komentarze w skryptach PHP

Pisząc kod skryptu w PHP, można stosować komentarze, które są ignorowane w trakcie przetwarzania skryptu, pozwalają za to umieścić w kodzie uwagi, które mogą być dla nas przydatne podczas jego późniejszej analizy. Do wyboru mamy trzy rodzaje komentarzy:

- komentarz blokowy, obejmujący więcej linii /* */
- komentarz jednowierszowy //
- komentarz jednowierszowy uniksowy #.

Przykład wykorzystania komentarzy prezentuje rys. 2.12. Po uruchomieniu skryptu w przeglądarce wyświetli się jedynie tekst „Witamy na stronie”, pozostały tekst to różne komentarze.



Rys. 2.12. Przykład wykorzystania komentarza w PHP

Źródło: Opracowanie własne.

Zmienne w PHP

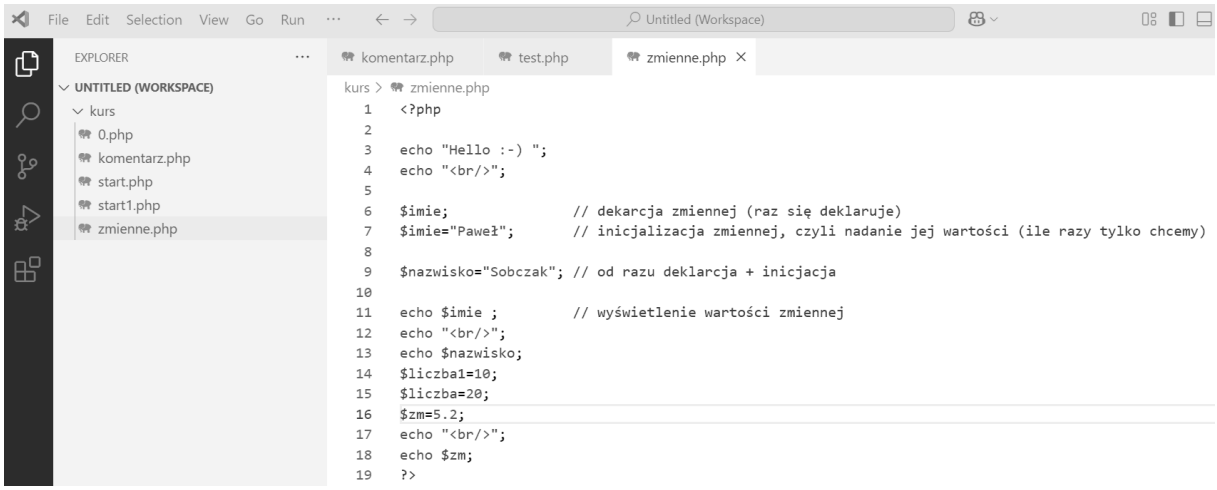
Zmienne to takie wirtualne „pudełka”, które **pozwalają na przechowywanie danych**. **Każda zmienna posiada swoją nazwę,** dzięki której można się do niej odwoływać w kodzie, oraz typ określający, jakiego rodzaju dane może przechowywać. **Nazwa zmiennej nadawana jest przez programistę**. Zasadniczo może być ona dowolna, choć musi spełniać następujące kryteria:

- musi zaczynać się od litery lub znaku podkreślenia,
- może zawierać jedynie litery, cyfry i znaki podkreślenia,
- przed nazwą zmiennej należy postawić znak **$**,
- nazwa zmiennej powinna zaczynać się od małej litery,
- w PHP rozróżniane są wielkie i małe litery, a zatem przykładowo **$Liczba** i **$liczba** to **dwie różne zmienne**!

Deklaracja i inicjalizacja zmiennej w PHP

Podczas deklaracji zmiennej nie trzeba podawać typu danych, jakie będzie przechowywać. Aby zadeklarować zmienną, należy podać jej nazwę i przypisać jej wartość (inicjalizacja).

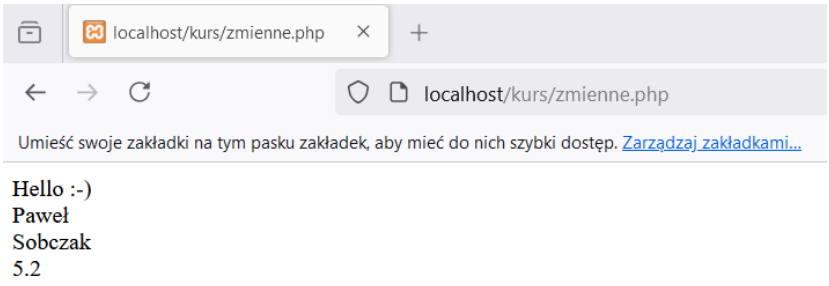
$nazwaZmiennej = wartośćZmiennej



Rys. 2.13, Przykład deklaracji zmiennych w PHP

Źródło: Opracowanie własne.

Przykład deklaracji zmiennych prezentuje rys. 2.13. Zmienna **$imie** i **$nazwisko** przechowują tekst, czyli typ napisowy, z kolei **$liczba** i **$liczba1** przechowuje liczby całkowite, a **$zm** przechowuje liczbę zmiennoprzecinkową. W PHP, podobnie jak np. w języku Python, bardzo wygodne jest to, że nie trzeba podawać typu deklarowanej zmiennej. Efekt uruchomienia skryptu z deklaracją zmiennych pokazany został na rys. 2.14.



Rys. 2.14. Uruchomienie skryptu z deklaracją zmiennych

Źródło: Opracowanie własne.

Rodzaje zmiennych, typy danych

- W klasycznych językach programowania każda zmienna ma swój typ.
- Typ zmiennej określa wartości, jakie mogą być jej przypisywane.

- W języku PHP, zmienna może przechowywać dowolne wartości.
- Nie oznacza to jednak, że nie ma ona typu, ale to, że jej typ może się dynamicznie zmieniać w trakcie działania kodu.
- W PHP typy danych możemy podzielić na trzy ogólne rodzaje: (**typy skalarne, typy złożone, typy specjalne**).

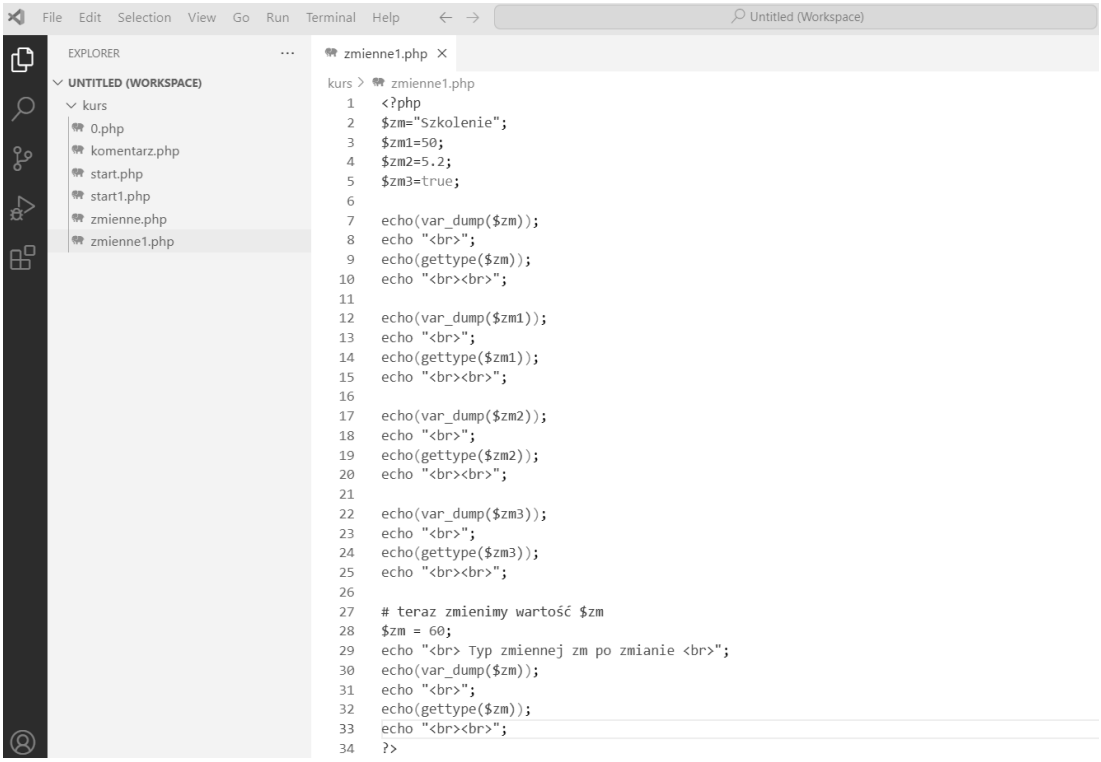
Typy skalarne to inaczej **typy proste**, dzielimy je na następujące rodzaje:

- typ **boolean**, // **true, false**
- typ **integer**, // **52, 100**
- typ **float, double**, // **3.2, 10.9** (separatorem jest **.**)
- typ **string** // **”Szkolenie”**.

Sprawdzenie typu zmiennej

Sprawdzenie typu zmiennej możesz wykonać na dwa sposoby:

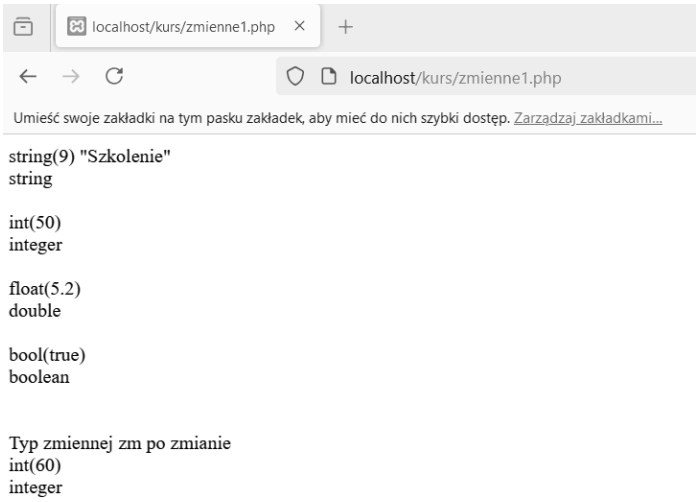
- **var_dump()**
- **gettype()**



Rys. 2.15. Przykład skryptu sprawdzającego typ zmiennej

Źródło: Opracowanie własne.

Rys. 2.15 przedstawia przykład skryptu sprawdzającego typ zmiennej na dwa sposoby, za pomocą funkcji **var_dump()** i **gettype()**, z kolei rys. 2.16 przedstawia efekt uruchomienia skryptu z rys. 2.15.



Rys. 2.16. Efekt uruchomienia skryptu sprawdzającego typ zmiennej

Źródło: Opracowanie własne.

Opis poszczególnych typów zmiennych

- **Typ boolean –** jest to **typ logiczny**, który może przyjmować tylko dwie wartości – **true** (prawda) lub **false** (fałsz). Ten typ wykorzystywany jest przy konstruowaniu wyrażeń logicznych oraz sprawdzaniu warunków.

  True, true, False, false

- **Typ integer** – jest to **typ całkowitoliczbowy**, dzięki któremu można reprezentować zarówno dodatnie, jak i ujemne liczby całkowite.

  123456, 28, -29, 0

- **Typ float, double** – jest to **typ** reprezentuje dodatnie i ujemne liczby **rzeczywiste**.

  123.54, -32.5, 0.5

- **Typ string** – jest to **typ łańcuchowy**, który służy do zapamiętywania sekwencji znaków. Łańcuch znaków można utworzyć za pomocą apostrofów i cudzysłowów. Jest jednak pewna różnica z użyciem tych znaków, wspomnimy o niej podczas zajęć.

  ‘Szkolenie, ”Szkolenie”

Typy złożone dzielą się na dwa rodzaje. Są to:

- **typ array (tablicowy),**

- **typ object (obiektowy)**.

Typy specjalne

- **Typ null** – jest **typem specjalnym** informującym o tym, że dana zmienna nie przechowuje żadnej wartości. Wielkość liter nie ma znaczenia, prawidłowe są zatem zapisy: **null**, **NULL**,

$zmienna = null.

Operacja na zmiennych w PHP

Przypisanie wartości do zmiennej

Utworzenie zmiennej polega na umieszczeniu w kodzie skryptu jej nazwy poprzedzonej znakiem **$** i przypisaniu jej wartości.

$nazwa_zmiennej = wartość_zmiennej;

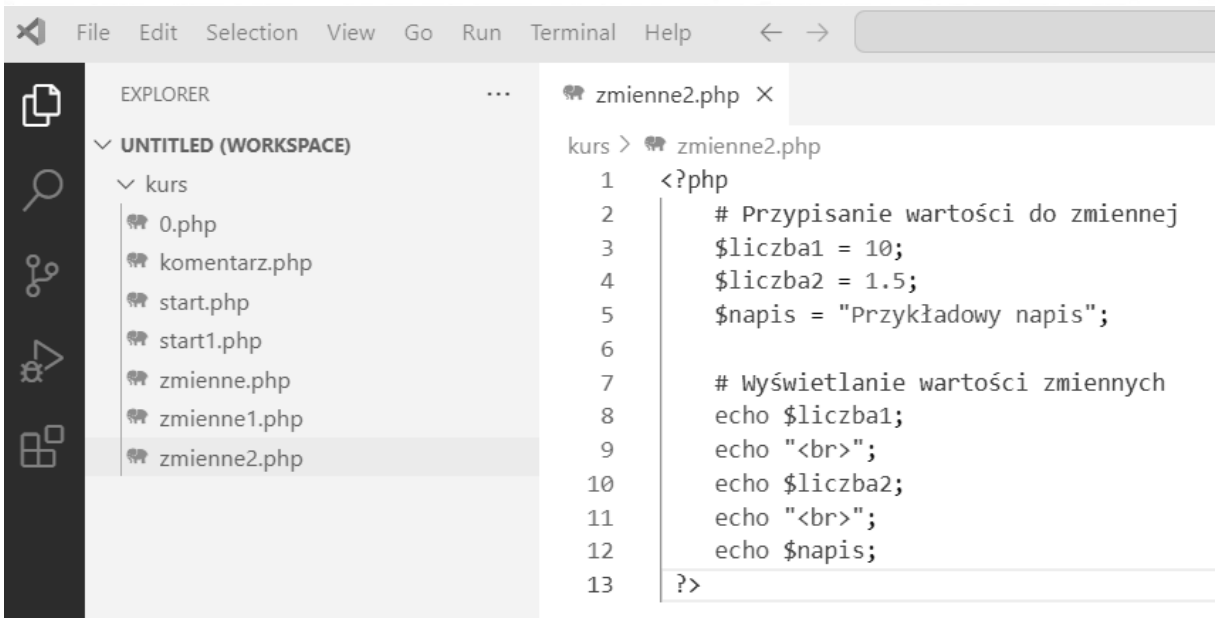
Powyższy zapis oznacza: utwórz zmienną o nazwie **$nazwa_zmiennej** oraz przypisz jej wartość **wartość_zmiennej**.

Wyświetlanie wartości zmiennych

Do wyświetlenia wartości zmiennej, a dokładniej do wysłania tej wartości w postaci ciągu znaków do okna przeglądarki, posługujemy się zazwyczaj instrukcją echo.

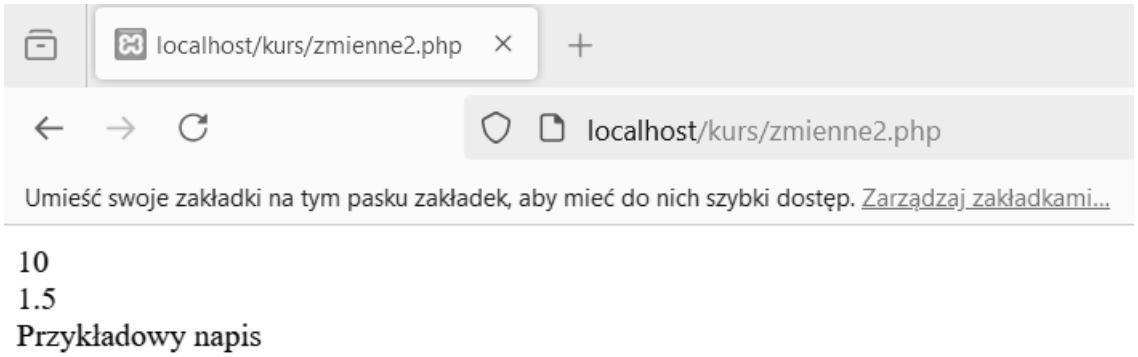
echo $nazwa_zmiennej;

Przykład skryptu wyświetlającego wartości zmiennych pokazany jest na rys. 2.17, a efekt wykonania tego skryptu przedstawia rys. 2.18.



Rys. 2.17. Przykład skryptu wyświetlającego wartości zmiennych

Źródło: Opracowanie własne.



Rys. 2.18. Efekt uruchomienia skryptu wyświetlającego wartości zmiennych

Źródło: Opracowanie własne.

Modyfikowanie wartości zmiennych

Modyfikacji wartości zmiennej wykonujemy tak samo jak byśmy przypisywali wartość do zmiennej, z tą różnicą, że musimy podać taką samą nazwę istniejącej już zmiennej, aby ją zmodyfikować.

$nazwaStarejZmiennej = nowaWartośćStarejZmiennej;

Przykład skryptu modyfikującego wartość zmiennej pokazany jest na rys. 2.19, a efekt wykonania tego skryptu przedstawia rys. 2.20.

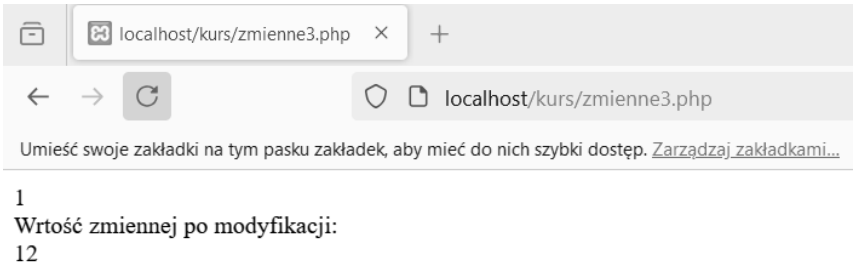
zmienne3.php

kurs > zmienne3.php

```php
<?php

$a = 1;
echo($a);
$a = 12;   # Modyfikowanie wartości zmiennej
echo "<br> Wrtość zmiennej po modyfikacji: <br>";
echo($a);

?>
```

Rys. 2.19. Przykład skryptu modyfikującego wartość zmiennej

Źródło: Opracowanie własne.



Rys. 2.20. Efekt uruchomienia skryptu modyfikującego wartość zmiennej

Źródło: Opracowanie własne.

Przypisywanie do zmiennej wartości innej zmiennej

Polega na przypisaniu do istniejącej zmiennej wartości innej zmiennej podając jej nazwę

$zmienna1 = 5;

$zmienna2 = $zmienna1;

$zmienna1 ma wartość 5, do zmiennej $zmienna2 zostaje przypisana wartość zmiennej $zmienna1, która ma wartość 1. Od tego momentu również $zmienna2 ma wartość 1.

Nadpisywanie wartości innej zmiennej

Czynność ta, polega na przypisaniu do istniejącej zmiennej wartości innej zmiennej podając jej nazwę, przykład poniżej. W poniższym przykładzie $zmienna2 ma wartość równą 100, jednak po nadpisaniu zmienną $zmienna1 ma wartość 5 (czyli taką samą jak $zmienna1).

$zmienna1 = 5;

$zmienna2 = 100;

$zmienna2 = $zmienna1;

Przykład operacji na zmiennych

Utwórz 5 zmiennych (zm_1, zm_2…), które będą przechowywać poniższe wartości.

- Ciekawe
- Programowanie
- Jest
- Wciągające
- I

Następnie w kodzie **przypisz zmienne do zmiennych**, tak by odczytując kolejne wartości zmiennych otrzymać tekst "Programowanie Jest Ciekawe I Wciągające". Do rozwiązania zadania wykorzystaj dodatkową zmienną. Przykładowe rozwiązania powyższego zadania zostało pokazane na rys. 2.21. W przykładzie tym zostało pokazane, jak połączyć w instrukcji echo tekst z wyświetleniem wartości zmiennej (konkatenacja).

Operatory

W języku programowania PHP występują operatory, które służą do wykonywania różnorodnych operacji, można podzielić na kilka grup:

- arytmetyczne,
- logiczne,
- przypisania,
- relacyjne (porównywania),
- inkrementacji/dekrementacji,
- trójargumentowy.

zmienna4.php

kurs > zmienna4.php

```php
<?php
    $zm1 = "Ciekawe";
    $zm2 = "Programowanie";
    $zm3 = "Jest";
    $zm4 = "Wciągające";
    $zm5 = "I";
    $pomoc = "";

    $pomoc=$zm1;
    $zm1=$zm2;
    $zm2=$zm3;
    $zm3=$pomoc;
    $pomoc=$zm4;
    $zm4=$zm5;
    $zm5=$pomoc;

    echo $zm1." ".$zm2." ".$zm3." ".$zm4." ".$zm5;
    // Programowanie Jest Ciekawe I Wciągające
    // kropka (.) pozwala połaczyć tekst wyświetlany
    // instrukcją echo z wartością zmiennej
?>
```

Rys. 2.21. Efekt uruchomienia skryptu nadpisującego wartości zmiennych

Źródło: Opracowanie własne.

Operatory arytmetyczne (, *, /, +, -, %)**

```
# $wynik = 2 ** 3;     (potęgowanie)
# $wynik = 2 * 3;      (mnożenie)
# $wynik = 5 / 3;      (dzielenie)
# $wynik = 7 + 5;      (dodawanie)
# $wynik = 8 - 3;      (odejmowanie)
# $wynik = 7 % 3;      (reszta z dzielenia)
```

Operatory logiczne

Operatory logiczne pozwalają na **wykonywanie operacji logicznych**. Można je wykonywać na argumentach, które posiadają wartość logiczną prawda lub fałsz. W języku PHP wartości te są oznaczane jako **true** i **false**. Iloczyn logiczny **and** lub **&&**, suma logiczna **or** lub ||, negacja**!**

```
# $a and $b     lub     # $a && $b
# $a or $b      lub     # $a || $
# !$ab
```

Operatory przypisania

Operatory są dwuargumentowe i **powodują przypisanie wartości argumentu znajdującego się z prawej strony operatora temu znajdującemu się z lewej**. Można stosować zapis skrócony (np. x += y), jednak początkowym programistą nie zalecamy tej formy zapisu.

```
# x = y
# x = x + y
# x = x – y
# x = x * y
# x = x / y
# x = x % y
# x = x . y
```

Operatory inkrementacji i dekrementacji

Operatory inkrementacji, czyli zwiększania (++), oraz dekrementacji, czyli zmniejszania (--).

- Operator ++ zwiększa po prostu wartość zmiennej o jeden, a **--** zmniejsza o jeden.
- Oba mogą występować w formie przedrostkowej lub przyrostkowej.

++$liczba, $liczba++

- Jak różnica? Obie postacie powodują zwiększenie wartości o jeden, ale w przypadku formy przedrostkowej (**++$liczba**) odbywa się to **przed wykorzystaniem zmiennej**, a w przypadku przyrostkowej (**$liczba++**) - **dopiero po jej wykorzystaniu**.

Operatory relacyjne

Operatory relacyjne służą do porównywania argumentów (stąd też nazywane są również operatorami porównywania). Wynikiem ich działania jest wartość logiczna **true** lub **false**, czyli prawda lub fałsz.

- == Wynikiem jest true, jeśli argumenty są sobie równe. **$a == $b**
- === Wynikiem jest true, jeśli argumenty są sobie równe i są tego samego typu. **$a === $b**

- <> Wynikiem jest true, jeśli argumenty są różne. **$a <> $b**
- **!=** Wynikiem jest true, jeśli argumenty są różne. **$a != $b**
- **!==** Wynikiem jest true, jeśli argumenty są różne lub są różnych typów. **$a !== $b**
- **>** **$a > $b**
- **<** **$a < $b**
- **>=** **$a >= $b**
- **<=** **$a <= $b**

Operator trójargumentowy

Wyrażenie warunkowe to pewien specyficzny operator – operator trójargumentowy.

{warunek} ? {wartość pierwsza} : {wartość druga}

Przykład

```
$a = 10;
$b = 15;
$wynik = $a < $b ? $b : $a;
echo ($wynik);
```

1. Najpierw obliczane jest wyrażenie po lewej stronie znaku zapytania (**$a < $b**). W tym przypadku 10 < 15, co jest prawdą (**true**).
2. **Jeśli wyrażenie jest prawdziwe, zwracana jest wartość znajdująca się po znaku zapytania, a przed dwukropkiem ($b). Jeśli wyrażenie jest fałszywe, zwracana jest wartość znajdująca się po dwukropku ($a).**
3. W naszym przypadku, ponieważ **$a < $b** jest **prawdą**, operator trójargumentowy zwraca wartość **$b**, czyli **15**.
4. Ta wartość (15) jest następnie przypisywana do zmiennej $wynik.
5. echo ($wynik); wyświetla wartość zmiennej $wynik, czyli 15.

Przykład podsumowujący dotychczasowy materiał

Napisz program, w którym zadeklarujesz dwie zmienne typu int o nazwach **x** oraz **y**. Przypisz do nich losowe liczby, a następnie za pomocą operatorów logicznych i matematycznych wyświetl wynik następujących zdań logicznych:

- Czy x jest większe od y?
- Czy x pomnożone przez 2 jest większe od y?

- Czy y jest mniejsze od sumy x+3 i jednocześnie większe od x pomniejszonego o 2?
- Czy iloczyn liczb x i y jest parzysty? (Wykorzystaj do sprawdzenia operację modulo).

Do utworzenia losowej liczby wykorzystaj funkcję **rand(0,100)** – która zwraca liczbę losowa z przedziału od 0 do 100. Rys. 2.22 przedstawia przykładowe rozwiązanie przykładu podsumowującego dotychczasowy materiał.

zm_podsum.php

kurs > zm_podsum.php

```php
<?php
echo "<br/><br/>";
$x = rand(0,100);
$y = rand(0,100);
echo "<br/>x = ".$x;
echo "<br/>y = ".$y;

$wynik = ($x > $y) ? "TAK": "NIE";
echo "<br/> czy x > y";
echo $wynik;

$wynik = (($x*2) > $y) ? "TAK": "NIE";
echo "<br/> czy x*2 > y ";
echo $wynik;

$wynik = ((($x+3) > $y) && ($y>$x-2)) ? "TAK": "NIE";
echo "<br/>Przykład 3";
echo $wynik;

$wynik = ((($x*$y)%2==0)) ? "TAK": "NIE";
echo "<br/>Przykład 4";
echo $wynik;
?>
```

Rys. 2.22. Efekt uruchomienia skryptu podsumowującego

Źródło: Opracowanie własne.

Zadanie

Napisz program, który wykonuje sumę cen produktów dla konkretnego zamówienia:

- Chleb (5,99 zł / 1 szt.)
- Mleko (3,50 zł / 1l)
- Cukierki (32,99 / 1kg)
- Zamówienie: 3 szt. chleba + 4 l mleka + 0,5 kg cukierków

Zadanie

Zaprojektuj odpowiedni algorytm i oblicz poniższą wartość z wykorzystaniem zmiennych:

- Samochód na 100 km spala 8,5 l paliwa.

Ile spali paliwa po przejechaniu 782 km?

Instrukcje sterujące

Instrukcja if

Podczas pisania programów (skryptów) bardzo często zachodzi konieczność wykonywania różnych operacji w zależności od prawdziwości jakiegoś warunku. Aby to zrealizować należy skorzystać z instrukcji sterującej **if**. Instrukcja ta występuje w kilku wersjach, najprostsza z nich ma postać:

```
if (warunek){
        instrukcja1;
        instrukcja2;
        …
}
```

Jeżeli warunek jest prawdziwy, zostaną wykonane instrukcje w {}, w przeciwnym razie zostanie wykonana kolejna instrukcja, która występuje po instrukcji **if**. Gdy w {} mamy tylko jedną instrukcję, klamry można pominąć, jednak dla początkujących programistów zaleca się pisanie tych klamer. Przykład skryptu, który zawiera instrukcje **if** przedstawia rys. 2.23. Skrypt sprawdza, czy wartość zmiennej jest mniejsza od zera, a następnie w drugim **if** sprawdzamy, czy zmienna jest większa lub równa zeru.

if.php

kurs > if.php

```php
<?php
$liczba = 5;
if($liczba < 0){
   echo "Wartość zmiennej jest mniejsza od 0.";
}

if($liczba >= 0){
    echo "Wartość zmiennej jest większa lub równa zero.";
}

?>
```

Rys. 2.23. Przykład instrukcji sterującej if

Źródło: Opracowanie własne.

Instrukcja if else

Instrukcja ta działa podobnie jak **if**, z tą różnicą, że blok **else** jest wykonywany w sytuacji, gdy warunek jest fałszywy (nieprawdziwy).

```
if (warunek){
        // Instrukcje do wykonania, gdy warunek jest prawdziwy
}
else{
        // Instrukcje do wykonania, gdy warunek jest fałszywy
}
```

Przykład realizacji instrukcji **if else** prezentuje rys. 2.24. Gdy **$zm1** ma większą wartość od **$zm2**, to wówczas wykona się instrukcja po **if**, w przeciwnym razie instrukcja po **else**. W każdym bloku jest po jednej instrukcji, więc można pominąć **{}**. Skrypt nie zadziała prawidłowo, gdyby zmienne miały taką samą wartość.

if1.php

kurs > if1.php

```php
<?php
$zm1 = 5;
$zm2 = 2;

if($zm1 > $zm2){
    echo "zm1 jest większa od zm2";
}
else{
    echo "zm1 jest mniejsza od zm2";
}
?>
```

Rys. 2.24. Przykład instrukcji sterującej if else

Źródło: Opracowanie własne.

Instrukcja if else if

Kolejna wersja instrukcji **if** pozwala na badanie wielu warunków. Po **if** może wystąpić wiele dodatkowych bloków **else if**.

```
f (warunek1){
        instrukcje1;
}
 else if (warunek2){
        instrukcje2;
}
else{
        instrukcje;
}
```

W języku PHP instrukcję **if…else if** można również zapisać tak, by ciąg **elseif** stanowił jedno słowo. Efekt wykonania instrukcji jest taki sam, a zapis bardziej przejrzysty.

```
if (warunek1){
        instrukcje1;
} elseif (warunek2) {
        instrukcje2;
}
```

Przykład realizacji instrukcji **if elseif** przedstawia rys. 2.25. W skrypcie tym wyeliminowano problem nieprawidłowego działania skryptu, gdy zmienne są równe z rys. 2.24.

if3.php

kurs > if3.php

```php
<?php
$zm1 = 5;
$zm2 = 5;

if($zm1 > $zm2){
    echo "zm1 jest większa od zm2";
}
elseif($zm1 < $zm2){
    echo "zm1 jest mniejsza od zm2";
}
else{
    echo "Zmienne są równe!";
}
?>
```

Rys. 2.25. Przykład instrukcji sterującej if elseif

Źródło: Opracowanie własne.

Zarówno w bloku **if,** jak i w **else** mogą wystąpić dowolne instrukcje PHP. Oznacza to, że można tam umieścić kolejne instrukcje **if**, a więc że mogą być **zagnieżdżane**. Wyrażenia warunkowe, stosowane w instrukcji **if**, mogą składać z wielu członów połączonych operatorami logicznymi. Warto poszczególne człony rozdzielone operatorami logicznymi umieszczać w nawiasach, co daje większą przejrzystość kodu.

Zadanie

Napisz program, który sprawdza, czy wpisana liczba jest liczbą parzystą, czy nieparzystą. Wykorzystaj operator reszty z dzielenia (%).

Zadanie

Napisz program, który oblicza wartość współczynnika BMI wg wzoru (waga / wzrost**2). Wzrost podawany jest w metrach. Jeżeli wynik jest w przedziale (18.5 – 24.9), to wypisuje „prawidłowa waga”, jeżeli poniżej to „niedowaga”, jeżeli powyżej to „nadwaga”.

Zadanie

Zadeklaruj trzy zmienne i sprawdź, czy z odcinków o takich długościach da się zbudować trójkąt prostokątny. Załóżmy, że dwie pierwsze zmienne stanowią długości boków przyprostokątnych, natomiast trzecia zmienna stanowi długość przeciwprostokątnej.

Instrukcje wyboru

Instrukcja switch

Instrukcja **switch** pozwala w wygodny sposób sprawdzić ciąg warunków i wykonać różne instrukcje w zależności od wyników porównywania. Instrukcja idealnie nadaje się do pogrupowania operacji w zależności o wartości zmiennej, która steruje wykonaniem operacji w instrukcji **switch**.

```
switch(wartość){
        case wartość1 :
                instrukcje1;
         break;
        case wartość2 :
                instrukcje2;
         break;
 default :
        instrukcje;
}
```

Przykład wykorzystania instrukcji **switch** pokazany jest na rys. 2.26. Poniższy rysunek (rys. 2.26) przedstawia jedynie fragment skryptu, który realizuje funkcje kalkulatora prostego. W zależności od wartości zmiennej sterującej **$z** instrukcja **switch** wykona dodawanie, odejmowanie, mnożenie lub dzielenie. Gdy osoba wykonująca skrypt wprowadzi nieprawidłowy operator (+, -, *, /), wykona się opcja domyślna, czyli wyświetli się napis o błędnym operatorze.

```
54  switch($z){
55      case "+":
56          echo $l1." + ".$l2." = ".$l1+$l2;
57      break;
58
59      case "-":
60          echo $l1." - ".$l2." = ".$l1-$l2;
61      break;
62
63      case "*":
64          echo $l1." * ".$l2." = ".$l1*$l2;
65      break;
66
67      case "/":
68          echo $l1." / ".$l2." = ".$l1/$l2;
69      break;
70
71      default:
72          echo "Błednie wybrany operator!";
73
74  }
```

Rys. 2.26. Przykład fragmentu skryptu wykorzystującego instrukcję switch

Źródło: Opracowanie własne.

Zadanie

Napisz prosty kalkulator, który pozwala użytkownikowi kolejno na:

- wprowadzenie pierwszej liczby,
- wprowadzenie jednego z podstawowych działań matematycznych (+, -, /, *),
- wprowadzenie drugiej liczby.

Instrukcje iteracyjne (pętle)

Pętlami nazywamy konstrukcje języka, które pozwalają na wielokrotne wykonywanie powtarzających się instrukcji (np. 100 razy wyświetlamy napis ”Witaj Świecie!”). W języku programowania PHP występują cztery rodzaje instrukcji iteracyjnych: (**for, while, do…while, foreach**). Każda z tych pętli ma inne przeznaczenie.

Pętla for

Ten rodzaj pętli wykorzystuje się, gdy wiemy, ile razy ma wykonać się dana instrukcja. Pętla typu for ma ogólną postać:

for(wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące){
instrukcje do wykonania
}

- **wyrażenie początkowe** jest stosowane do zainicjowania zmiennej używanej jako licznik liczby wykonań pętli,
- **wyrażenie warunkowe** określa warunek, jaki musi zostać spełniony, aby wykonać kolejny obieg pętli,
- **wyrażenie modyfikujące** jest zwykle używane do modyfikacji zmiennej będącej licznikiem (inkrementacji, bądź dekrementacji).

Przykład użycia pętli **for** przedstawiony został na rys. 2.27. Skrypt wyświetli na ekranie 10 razy napisy „Ala ma kota". Na zajęciach zostanie szczegółowo wyjaśniona kolejność wykonywanych operacji.

for.php

kurs > for.php

```
1  <?php
2
3  for($i = 0; $i < 10; $i++){
4     echo "Ala ma kota <br>";
5  }
6
7  ?>
```

Rys. 2.27. Przykład pętli for

Źródło: Opracowanie własne.

Wewnątrz każdej pętli można umieścić dowolne instrukcje, a więc i kolejną pętlę. Pętle mogą być zagnieżdżane. Najczęściej zagnieżdżane są właśnie pętle for.

Zadanie

Napisz program, który spośród liczb 1-100 wyświetli tylko te podzielne przez 4.

Zadanie

Zaimplementuj mechanizm obliczania potęgi bez użycia operatora **

- Użytkownik podaje podstawę i wykładnik potęgi.
- Wykorzystując mechanizm pętli zaimplementuj odpowiedni kod.

- Pamiętaj, wszystko, co jest podniesione do potęgi 0, jest równe 1.
- Wykładnikiem potęgi są liczby >= 0.

Pętla while

Pętla typu **while** służy, podobnie jak **for**, do wykonywania powtarzających się czynności. Pętlę **for** wykorzystuje się najczęściej, gdy liczba powtarzanych operacji jest znana (np. jest zapisana w pewnej zmiennej), natomiast pętlę **while**, gdy liczby powtórzeń z góry nie znamy, a zakończenie pętli jest uzależnione od spełnienia pewnego warunku

```
while (warunek){
        instrukcje;
}
```

Podobnie jak w przypadku instrukcji **if** w pętlach możemy pominąć klamry {}, gdy występuje tylko jedna instrukcja, jednak nie jest to zalecane dla początkujących programistów. Pętla będzie wykonywała instrukcje tak długo, jak warunek będzie prawdziwy. **W pętli while najpierw sprawdzany jest warunek, a dopiero potem wykonywana jest instrukcja, gdy warunek jest prawdziwy. Gdy warunek będzie od początku fałszywy, pętla nie wykona się ani razu!**

w.php ×

kurs > w.php

```php
<?php
$i = 0;
while($i < 10){
   echo "Pętla while, i = $i";
   echo "<br>";
   $i++;
}
?>
```

Rys. 2.27. Przykład pętli while

Źródło: Opracowanie własne.

Pętla do while

Kolejny rodzaj pętli występującej w języku PHP to **do...while**. Ogólna postać tej pętli została przedstawiona poniżej:

```
do{
        instrukcje;
}
while(warunek);
```

W pętli do while najpierw wykonywana jest instrukcja, a dopiero potem sprawdzany jest warunek. Nawet gdy warunek będzie od początku fałszywy, pętla wykona się przynajmniej jeden raz! Przykład wykorzystania pętli **do while** przedstawia rys. 2.28.

dw.php

kurs > dw.php

```php
<?php
$i = 0;

do{
   echo "Pętla do...while, i = $i";
   echo "<br>";
}
while($i++ < 9);

?>
```

Rys. 2.28. Przykład pętli do while

Źródło: Opracowanie własne.

Zadanie

Napisz pętlę, która będzie losowała liczby z zakresu 1-10.

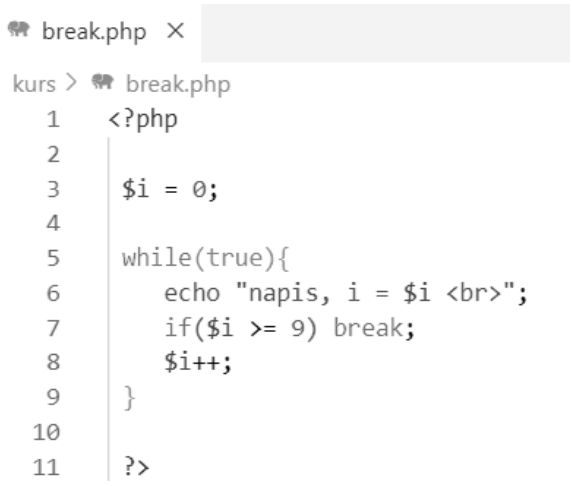
- Każda wylosowana liczba jest wypisywana.
- Pętla ma skończyć swoje działanie, jeżeli wylosuje cyfrę 7.
- Ostatecznie pętla powinna wypisać po ilu próbach wylosowała cyfrę 7.

Instrukcje break i continue

Podczas korzystania z pętli bardzo często korzysta się z instrukcji **break** i **continue**. Instrukcje **break** i **continue** pozwalają na modyfikacje zachowań pętli i można je stosować

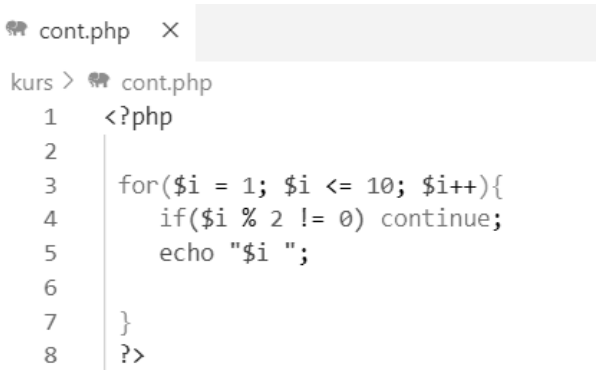
z każdym ich rodzajem niezależnie od tego, czy będzie to **for**, **while**, **do…while**, czy **foreach**. Instrukcja **break** była już przedstawiona w instrukcji **switch**.

- Instrukcja **break** powoduje **przerwanie działania całej pętli.**
- Instrukcja **continue** powoduje **przerwanie działania jednej iteracji pętli.**



Rys. 2.29. Przykład użycia instrukcji break

Źródło: Opracowanie własne.



Rys. 2.30. Przykład użycia instrukcji continue

Źródło: Opracowanie własne.

Przykład użycia instrukcji **break** pokazany jest na rys. 2.29, z kolei instrukcji **continue** na rys. 2.30. W powyższych przykładach instrukcja **break** służy do przerwania nieskończonej pętli, natomiast instrukcja **continue** służy do pominięcia liczb nieparzystych podczas wyświetlania wartości zmiennej.

Zadanie

Zaprojektuj pętlę, która wypisze liczby z zakresu od -50 do 50, przy założeniach:

- Liczby ujemne są wypisane tylko nieparzyste.
- Liczby dodatnie są wypisane tylko parzyste.
- Zero jest wypisane.

Zadanie

Zaprojektuj program, w którym użytkownik podaje dowolną liczbę z zakresu 0-20.

- Zaprojektuj program, który będzie losował w zakresie liczb 0-20 tak długo, aż wylosuje wprowadzoną wartość przez użytkownika.
- Każda wylosowana liczba zostanie wypisana.

Tablice w PHP

Wprowadzenie do tablic

Tablica jest to zbiór danych tego samego typu. **Na uwagę zasługuje fakt, że w PHP wartości poszczególnych komórkach tablicy nie muszą być tego samego typu** (w odróżnieniu od wielu innych języków). Najprostszym typem tablicowym są stringi.

$imie = ”Paweł”;

0	1	2	3	4
P	a	w	e	ł

echo $imie[1]; **// a**
echo $imie[4]; **// ł**
echo strlen($imie); **// 5 – służy do określania długości ciągu znaków**

Tablice są to struktury danych pozwalające na przechowywanie uporządkowanego zbioru elementów. **W celu utworzenia tablicy używamy słowa kluczowego array**.

$tablica = array(wartość1, wartość2, ..., wartość_n);

Przykład deklaracji tablicy jest pokazany na rys. 2.31. Aby uzyskać dostęp do wartości zapisanej w danej komórce, należy podać jej indeks w nawiasie kwadratowym występującym za nazwą tablicy. **Należy pamiętać, że indeksowanie tablicy zaczyna się od 0, co oznacza,**

że indeksem pierwszej komórki jest 0. Jeśli tablica ma duży rozmiar, to do odczytu jej zawartości lepiej użyć pętli. Może być to zarówno zwykła pętla **for**, jak i pętla typu **foreach**, przykład skryptu można zobaczyć na rys. 2.32.

tab.php

kurs > tab.php

```php
<?php

$kolory = array("czerwony", "zielony", "niebieski");
echo "kolory[0] = ", $kolory[0], "<br>";
echo "kolory[1] = ", $kolory[1], "<br>";
echo "kolory[2] = ", $kolory[2], "<br>";

?>
```

Rys. 2.31. Przykład deklaracji tablicy

Źródło: Opracowanie własne.

tab1.php

kurs > tab1.php

```php
<?php

$kolory = array("czerwony", "zielony", "niebieski", "czarny", "biały");

echo "Pętla for: <br>";
for($i = 0; $i < 5; $i++){
   echo $kolory[$i], "<br>";
}

echo "<br><br>";
echo "Pętla foreach: <br>";
foreach($kolory as $kolor){
   echo $kolor, "<br>";
}

?>
```

Rys. 2.32. Przykład wykorzystania pętli do odczytu elementów tablicy

Źródło: Opracowanie własne.

Nie ma konieczności, aby wypełniać tablice danymi już podczas ich tworzenia. Mogą być one także przypisywane w dalszej części skryptu. Można jedynie zadeklarować pustą tablicę, a następnie przypisać wartości poszczególnym elementom tablicy. **Podczas**

zapisywania w tablicy nowego elementu nie trzeba podawać jego indeksu, element zostanie dopisany wówczas na końcu tablicy.

```
$kolory = array();
$kolory[0] = "czerwony";
$kolory[1] = "zielony";
$kolory[2] = "niebieski";
$kolory[] =”czarny”;
```

Zadanie

Przy pomocy pętli wygeneruj tablicę 10 losowych liczb 0-100, a następnie wypisz te liczby oraz oblicz ich średnią. Poszukaj najmniejszego i największego elementu tablicy.

Pętla foreach

Po wprowadzaniu tablic można przejść do opisu pętli **foreach**, która pozwala na dostęp do kolejnych elementów tablicy. Elementami tablic mogą być wartości, napisy, obiekty itp. Może występować w dwóch postaciach:

- pierwsza postać,

```
foreach($tablica as $wartość){
        instrukcje;
}
```

- druga postać

```
foreach($tablica as $klucz => $wartość){
        instrukcje;
}
```

Jest to wygodny sposób na **przetworzenie każdego elementu tablicy (kolekcji) bez konieczności ręcznego zarządzania indeksami**. Będziemy się w pętli odwoływać do elementów tablicy o nazwie **$tablica**. W każdej iteracji pętli do tej zmiennej **$wartość**

przypisywana jest wartość aktualnego elementu tablicy. Wewnątrz bloku {} mamy dostęp do zmiennej **$wartość**, która zawiera aktualny element tablicy. Przykład wykorzystania pętli **forech** w pierwszej postaci pokazany jest na rys. 2.33.

Pierwsza postać pętli **foreach** zwraca jedynie w pętli wartości poszczególnych elementów tablicy. Gdy oprócz wartości elementów tablicy potrzebujemy ich klucze, to wówczas należy użyć pętli w drugiej postaci. Wewnątrz bloku {} podczas każdej iteracji pętli mamy dostęp do zmiennych **$klucz i $wartość**. Przykład wykorzystania pętli **forech** w drugiej postaci pokazany jest na rys. 2.34.

forech.php ×

kurs > forech.php

```php
<?php

$owoce = array("jabłko", "banan", "gruszka", "truskawka");

foreach ($owoce as $owoc) {
    echo "Mam owoc: " . $owoc . "<br>";
}

?>
```

Rys. 2.33. Przykład wykorzystania pętli forech do odczytu elementów tablicy (pierwsza postać)

Źródło: Opracowanie własne.

Pętli **foreach** jest szczególnie przydatna, gdy pracujemy z **tablicami asocjacyjnymi**, czyli takimi, w których elementy są identyfikowane za pomocą kluczy (tekstowych lub numerycznych), a nie tylko indeksów liczbowych.

foreach.php ×

kurs > foreach.php

```php
<?php

$osoba = array(
    "imie" => "Jan",
    "nazwisko" => "Kowalski",
    "wiek" => 30,
    "miasto" => "Warszawa"
);

foreach ($osoba as $klucz => $wartosc) {
    echo $klucz . ": " . $wartosc . "<br>";
}

?>
```

Rys. 2.34. Przykład wykorzystania pętli forech do odczytu elementów tablicy (druga postać)

Źródło: Opracowanie własne.

Tablice asocjacyjne

Oprócz tablic indeksowanych numerycznie w języku PHP istnieją również tablice asocjacyjne. Ten rodzaj tablicy został wykorzystany w drugim przykładzie pętli f**oreach** (rys. 2.34). **W tablicach tego typu każdemu indeksowi można nadać unikalną nazwę**, czyli zamiast indeksów 0, 1, 2 itd. mogą występować indeksy: imie, nazwisko, wiek itp. Zamiast terminu **„indeks”** stosuje się określenie **„klucz”**. Kluczami mogą być ciągi znaków bądź wartości całkowite. Tablicę tego typu tworzy się, podobnie jak w przypadku tablic klasycznych indeksowanych numerycznie, za pomocą słowa kluczowego **array**, konstrukcja ta ma jednak nieco inną postać.

```
$nazwa_tablicy = array(
        klucz1 => wartość1,
        klucz2 => wartość2,
        klucz3 => wartość3,
);
```

Drugim ze sposobów tworzenia tablicy asocjacyjnej jest użycie składni z nawiasami kwadratowymi, podobnie jak miało to miejsce w przypadku tablic indeksowanych numerycznie.

$nazwa_tablicy["nazwa_klucza"] = wartość_klucza;

Przykład

$kolory["kolor1"] = "czerwony";
$kolory["kolor2"] = "zielony";
$kolory["kolor3"] = "niebieski";

Do odczytu tablic asocjacyjnych można użyć pętli, podobnie jak dla zwykłych tablic. Nie może być to jednak zwykła pętla **for**, gdyż nie może ona „twierdzić", jakie są wartości kluczy. W związku z tym tablice asocjacyjne najczęściej są obsługiwane przez pętle typu **foreach**. Taka pętla potrafi pobrać kolejne wartości kluczy. Modyfikacji zawartości tablic asocjacyjnych dokonuje się w sposób analogiczny jak w przypadku tablic klasycznych. Oczywiście zamiast indeksów numerycznych trzeba zastosować wartości kluczy. Ogólna postać:

$tablica["klucz"] = wartość;

Przykład

$kolory["kolor1"] = "zielony";

Zadanie

Napisz skrypt, który zamieni ciąg cyfr na formę tekstową. Znaki niebędące cyframi mają być ignorowane, konwertujemy cyfry, nie liczby, a zatem:

- 911 to "dziewięć jeden jeden"
- 1100 to "jeden jeden zero zero".

Wykorzystaj tablice asocjacyjne.

Zadanie

Wykorzystując poznane typy sekwencyjne zaprojektuj kod dla poniższej funkcjonalności: Użytkownik podaje dwie cyfry z zakresu (1-9) słownie: np. "jeden", "cztery", a program oblicza sumę liczb.

Operacje na tablicach

Funkcja **count($tablica)** zwraca **liczbę elementów** zwykłej tablicy i tablicy asocjacyjnej. Często w pętlach musimy znać liczbę elementów tablicy podczas iteracji na jej elementach.

Do **dodawanie elementów do tablicy** można użyć funkcji **array_push**. Funkcja służy do dodawania jednego lub więcej elementów na końcu tablicy:

array_push($nazwa_tablicy, "wartość");

Funkcja przyjmuje dwa argumenty: **$nazwa_tablicy**, do której chcemy dodać element oraz **wartość**.

Usunięcie ostatniego elementu tablicy można dokonać za pomocą funkcji **array_pop**.

array_pop($nazwa_tablicy);

Funkcja przyjmuje jako argument **nazwę tablicy,** z której zostanie usunięty ostatni element.

Funkcja **unset()** w PHP służy do **usuwania zmiennych, w tym również elementów tablicy** identyfikowanych przez ich indeks (klucz). Jest to podstawowy sposób na usuwanie elementów z tablic w PHP. Zapis

unset($tablica[$indeks]);

usuwa element tablicy o podanym indeksie **$indeks**.

Do **usuwania lub modyfikacji wybranego indeksu** z tablicy służy funkcja **array_splice**. Funkcja **array_splice()** przyjmuje co najmniej dwa argumenty:

- **nazwa_tablicy**, z której chcemy usunąć elementy,
- **indeks_początkowy**, od którego rozpoczynamy usuwanie,
- **ile_elementów** (opcjonalny), liczba elementów do usunięcia; jeśli pominiemy ten argument, zostaną usunięte wszystkie elementy od **indeks_początkowy** do końca tablicy,
- **nazwa_tablicy_zastępującej** (opcjonalny), tablica z elementami, które mają zastąpić usunięte elementy; jeśli ten argument nie zostanie podany, elementy zostaną tylko usunięte.

Do **szukania w tablicy wartości** służy funkcja **array_search**, która zwraca **klucz** tablicy pod którym znajduje się szukana **wartość**.

Funkcje w PHP

Funkcje są to wydzielone bloki kodu przeznaczone do wykonywania konkretnych zadań. Dzięki nim unikniemy wielokrotnego powtarzania w skrypcie tych samych instrukcji, a kod stanie się krótszy i bardziej czytelny. W celu utworzenia funkcji należy użyć słowa kluczowego **function**.

```
function nazwa_funkcji()
{
        // instrukcje wnętrza funkcji
}
```

W nazewnictwie funkcji obowiązują podobne zasady jak przy zmiennych. Aby wykonać instrukcje znajdujące się wewnątrz funkcji (pomiędzy znakami {}), należy ją wywołać. Wywołanie polega na umieszczeniu w kodzie skryptu nazwy funkcji wraz z występującym po niej nawiasem okrągłym. Przykład użycia funkcji został pokazany na rys. 2.35. Funkcjom można przekazywać argumenty, czyli wartości, które mogą wpływać na ich zachowanie lub też być przez nie przetwarzane. Listę parametrów należy umieścić w nawiasie okrągłym za nazwą funkcji, oddzielając je od siebie przecinkami.

```
function nazwa_funkcji(parametr1, parametr2, ...)
{
        // instrukcje z wnętrza funkcji

}
```

fun.php

kurs > fun.php

```php
<?php

// deklaracja funkcji
function func()
{
   echo "Instrukcja echo z funkcji. <br>";
}

// wywołanie funkcji
func();

?>
```

Rys. 2.35. Przykład wykorzystania funkcji w skrypcie

Źródło: Opracowanie własne.

Począwszy od wersji 7 PHP, dozwolone jest deklarowanie typów parametrów przekazywanych do funkcji (typ: int, float, string itp.).

```
function nazwa_funkcji(typ1, parametr1, typ2, parametr2, ...)
{
      // instrukcje z wnętrza funkcji

}
```

Funkcje oprócz obierania argumentów (parametrów), mogą również zwracać wartości przez funkcje. Czynność ta jest wykonywana za pomocą instrukcji **return**. Również można deklarować typ zwracanej wartości.

```
function nazwa_funkcji(parametry)
{
        // Instrukcje z wnętrza funkcji
         return wartość;
}
```

Przykład funkcji, która pobiera parametry, a zarazem zwraca wartość jest funkcja z rys. 2.36.

fun1.php

kurs > fun1.php

```php
<?php

$a = 20;
$b = 10;

function dodaj($wart1, $wart2)
{
   $wynik = $wart1 + $wart2;
   return $wynik;
}

$wartosc = dodaj ($a, $b);
echo "Wynikiem dodawania $a + $b jest $wartosc.";

?>
```

Rys. 2.36. Przykład funkcji z parametrami

Źródło: Opracowanie własne.

Zadanie

Utwórz 10-cio elementową tablicę losowych liczb 0-100, a następnie napisz funkcję, która jako parametr przyjmuje ww. tablicę i zwróci:

- średnią elementów tablicy.

Elementy programowania obiektowego

Obiekty mogą przechowywać pewne dane (atrybuty) i wykonywać pewne zadania (funkcje). Nie można jednak tworzyć obiektów bez ich wcześniejszego zaprojektowania za pomocą klasy (która określa, jak będą zbudowane obiekty tworzone na jej podstawie). Innymi słowy, **klasa to zdefiniowany przez programistę nowy typ danych**. Klasa jest definiowana za pomocą słowa kluczowego **class**. Ogólna postać takiej konstrukcji jest następująca:

```
class NazwaKlasy{
        //definicja klasy
}
```

Klasa może zawierać:

- Nieograniczoną liczbę zmiennych (nazywanych polami lub właściwościami).
- Nieograniczoną liczbę funkcji (nazywanych metodami) – elementy te nazywa się składowymi klasy.

Przykład klasy:

```
class Samochod{
        public $marka;
        public $model;
        public $rokProdukcji;
}
```

Gdy mamy zdefiniowaną klasę (czyli nowy typ danych), możemy na jej podstawie tworzyć obiekty, czyli konkretne egzemplarze tej klasy, używając słowa kluczowego **new**.

$obiekt = new NazwaKlasy(); **np. $obiekt = new Samochod();**

Odwołania do składowych obiektu wykonuje się za pomocą operatora operatora **->**.

- Dla pól: **$obiekt->nazwa_pola;**
- Dla metod **$obiekt->nazwa_metody(argumenty_metody);**

Programowanie obiektowe to bardzo złożony i obszerne zagadnienie, dlatego więcej informacji na ten temat podanych zostanie podczas kurs. Przykład programowania obiektowego pokazany został na rys. 2.38.

Zadanie

Napisz program składający się z klasy Auto, która zawiera pola (marka, model, cena, kolor). Utwórz 3 obiekty klasy Auto i wypisz ich cechy na ekran.

Zadanie

Napisz program, który będzie dodawał pracowników w postaci obiektów do tablicy w poniższym zakresie danych: (imie, nazwisko, data_urodzenia, staz) a następnie wyświetli elementy listy obiektów wprowadzonych do listy.

klasa.php

kurs > klasa.php

```php
<?php
class Samochod {
    public $marka;
    public $model;
    public $kolor;
    public $rokProdukcji;

    // Metoda wyświetlająca dane samochodu
    public function wyswietlDane() {
        echo "Marka: " . $this->marka . "<br>";
        echo "Model: " . $this->model . "<br>";
        echo "Kolor: " . $this->kolor . "<br>";
        echo "Rok produkcji: " . $this->rokProdukcji . "<br>";
    }

    //Konstruktor - metoda specjalna wywoływana przy tworzeniu obiektu
    public function __construct($marka, $model, $kolor, $rokProdukcji) {
        $this->marka = $marka;
        $this->model = $model;
        $this->kolor = $kolor;
        $this->rokProdukcji = $rokProdukcji;
    }
}

// Tworzenie obiektów klasy Samochod
$mojSamochod = new Samochod("Toyota", "Corolla", "Czerwony", 2020);
$samochodSasiada = new Samochod("Ford", "Mustang", "Niebieski", 2022);
// Wywoływanie metody wyswietlDane() dla każdego obiektu
echo "<h3>Mój samochód:</h3>";
$mojSamochod->wyswietlDane();
echo "<h3>Samochód sąsiada:</h3>";
$samochodSasiada->wyswietlDane();
// Modyfikacja właściwości obiektu
$mojSamochod->kolor = "Czarny";
echo "<h3>Mój samochód po zmianie koloru:</h3>";
$mojSamochod->wyswietlDane();
?>
```

Rys. 2.38. Przykład skryptu z programowania obiektowego

Źródło: Opracowanie własne.

Bazy danych MySQL

Wprowadzenie do baz

Baza danych to kolekcja wzajemnie powiązanych danych przechowywana w pamięciach dyskowych i udostępniania jej użytkownikom na określonych zasadach. Bazy danych umożliwiają szybkie wyszukiwanie informacji według określonego kryterium, nawet z bardzo dużego zbioru. Każdy użytkownik komputera może stworzyć własna bazę danych, pod warunkiem, że zna podstawowe metody jej tworzenia. Baza danych może zawierać jedną lub więcej tabel, w których przechowywane będą połączone informacje. Język **SQL** (*Structured Query Language*) jest najbardziej znanym i rozpowszechnionym strukturalnym językiem zapytań, który służy do tworzenia, modyfikacji oraz zarządzania bazami danych. Język ten nie służy do tworzenia programów, lecz wysyłaniu **zapytań** (*query*) do bazy i odebraniu / przetworzeniu zwrotnego wyniku. **Podczas szkolenia zdecydujemy jakie oprogramowanie zainstalujemy, by mieć możliwość tworzenia relacyjnych baza danych.**

Podstawowe typy danych w bazie danych

Podobnie jak w PHP są typy danych, tak również podczas tworzenia baz danych, a w nich tabel musimy zaprojektować kolumny o właściwym typie.

Liczbowe:

- Całkowite: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT (różne zakresy).
- Zmiennoprzecinkowe: FLOAT, DOUBLE, DECIMAL (różna precyzja).

Tekstowe (łańcuchowe):

- CHAR(n): Stała długość n znaków.
- VARCHAR(n): Zmienna długość do n znaków.
- TEXT, MEDIUMTEXT, LONGTEXT: Długie teksty.
- ENUM, SET: Typy wyliczeniowe.

Data i czas:

- DATE: Data (RRRR-MM-DD).
- DATETIME: Data i czas.
- TIMESTAMP: Znacznik czasu.
- TIME: Czas.
- YEAR: Rok.

Podczas projektowania baz danych i kolumn w tabelach pojawią się takie własności pól:

- **PRIMARY KEY** (klucz główny) – powoduje, że dane w kolumnie nie mogą się powtarzać, służy do identyfikacji rekordu.
- **FOREIGN KEY** (klucz obcy) – odwołanie do klucza głównego z innej tabeli.
- **UNIQUE** (niepowtarzalny) – dane nie mogą przyjmować tych samych wartości.
- **AUTO_INCREMENT** – automatyczne zwiększanie wartości.
- **NOT NULL** – konieczność wypełnia pola.
- **DEFAULT wartość** – przypisanie wartości domyślnej.

Podstawowe operacje na bazie danych

- Dodawanie rekordów do tabeli (**INSERT**)

insert into table(column1,column2, …) values (value1,value2, ...);

- Aktualizowanie rekordów (**UPDATE**)

update table_name set column1=expression1,column2=expression2, ... [where]

- Usuwanie rekordów – wierszy z tabeli (**DELETE**)

delete from table_name where …

- Pobierania danych z bazy danych, tabel (**SELECT**)

SELECT kolumna1, kolumna2, ... FROM nazwa_tabeli;

- Pobierania warunkowe danych z bazy danych, tabel (**SELECT**)

SELECT kolumna1, kolumna2, ... FROM nazwa_tabeli WHERE warunek;

Do konstruowania zapytań można wykorzystywać operatory relacyjne, logiczne itp.

To tylko podstawowe operacje na bazie danych, które możemy przeprowadzić na tabelach bazy danych, w rzeczywistości jest ich znacznie więcej i pozwalają na pobieranie danych w określonym porządku, agregowanie, porcjowanie danych, pobieranie z różnych tabel jednocześnie, łączenie tabel itp., które w miarę możliwości wprowadzane na szkoleniu.

Bibliografia

Nixon, R. (2018). *PHP i MySQL. Tworzenie stron WWW. Vademecum profesjonalisty*. Gliwice: Helion.

Welling, L., & Thomson, L. (2017). *PHP i MySQL. Wprowadzenie*. Gliwice: Helion.

Ullman, L. (2020). *PHP i MySQL dla każdego*. Warszawa: Wydawnictwo Naukowe PWN.

Gilmore, J. W. (2021). *PHP i MySQL. Od podstaw do zaawansowanych technik*. Gliwice: Helion.

Zandstra, M. (2019). *PHP. Obiekty, wzorce, narzędzia*. Gliwice: Helion.

Duckett, J. (2022). *PHP i MySQL. Aplikacje internetowe po stronie serwera*. Kraków: Znak.

Beighley, L., & Morrison, M. (2015). *Head First PHP & MySQL*. Warszawa: Wydawnictwo Naukowe PWN.

Zalewski, M. (2019). *Tao of the Web: Architektura nowoczesnych aplikacji internetowych*. Gliwice: Helion.

Duckett, J. (2014). *HTML i CSS. Zaprojektuj i zbuduj witrynę WWW*. Gliwice: Helion.

MacCaw, A. (2012). *JavaScript. Aplikacje WWW.* Gliwice: Helion.

Lis, M. (2017). *PHP i MySQL. Dla każdego.* Gliwice: Helion.