

Pierwsza strona i narzędzia

Czas: 2 × 45 min

praca indywidualna

Cel: Przygotowanie stanowiska pracy (edytor, serwer lokalny, przeglądarka), utworzenie pierwszego dokumentu HTML5 oraz poznanie narzędzi deweloperskich i walidatora W3C.

Po zajęciach potrafisz:

- zakłada uporządkowany folder projektu i otwiera go w VS Code
- tworzy poprawny szkielet dokumentu HTML5 z deklaracją języka i kodowania
- uruchamia stronę przez serwer lokalny (Live Server lub Apache z XAMPP)
- odczytuje w F12 kody odpowiedzi HTTP (200, 404) i nagłówki odpowiedzi
- poprawia kod na podstawie komunikatów walidatora W3C

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- folder `r1_01_start` z Pliki_do_zadan (pliki `do_walidacji.html` i `logo.svg`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **XAMPP** — serwer Apache; pliki stron w `C:\xampp\htdocs` apachefriends.org/download.html
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.01, czym jest żądanie i odpowiedź HTTP oraz co oznaczają kody 200 i 404.

Bezpieczeństwo

- Instaluj rozszerzenia VS Code tylko z oficjalnego panelu Rozszerzeń i sprawdzaj autora — fałszywe rozszerzenia o podobnych nazwach się zdarzają.
- Nie wyłączaj zapory systemu Windows dla Apache; serwer lokalny ma działać tylko na Twoim komputerze.

Przebieg zadania

1 Folder projektu i edytor

Wszystkie zadania z przedmiotu zapisuj w jednym folderze `nazwisko_imie` (bez polskich znaków i spacji). Każde zadanie ma swój podfolder.

```
nazwisko_imie/  
└─ zad01/  
   │ index.html  
   │ o-mnie.html  
   │ do_walidacji.html  
   │ css/  
   └─ img/  
      └─ logo.svg
```

1. Utwórz strukturę folderów jak na schemacie (folder `css` na razie zostaje pusty). Do `zad01` skopiuj `do_walidacji.html`, a do `zad01/img` plik `logo.svg` z folderu `r1_01_start`.
2. Uruchom VS Code, wybierz Plik → Otwórz folder... i wskaż `zad01`. Zainstaluj rozszerzenie Live Server (Ctrl+Shift+X, wyszukaj „Live Server”, autor Ritwick Dey).
3. Utwórz plik `index.html`, wpisz `!` i naciśnij Tab (skrót Emmet). Obejrzyj wygenerowany szkielet: zmień `lang="en"` na `lang="pl"` i wpisz własny tytuł strony w `title`.

2 Treść pierwszej strony

1. W `body` dodaj: komentarz z imieniem, nazwiskiem i klasą, nagłówek `h1`, obraz `img/logo.svg` z atrybutami `alt`, `width` i `height` (120), dwa akapity o sobie (jedno słowo wyróżnij znacznikiem `strong`), nagłówek `h2` i listę `ul` z trzema punktami.
2. Utwórz drugą stronę `o-mnie.html` (też z pełnym szkieletem) z nagłówkiem i akapitem. Połącz obie strony odnośnikami `a` w obie strony.
3. Na końcu `index.html` dodaj celowo obraz `img/brak.png` (pliku nie twórz) — posłuży do testu błędu 404.

3 Uruchomienie przez serwer lokalny

Strona otwarta podwójnym kliknięciem ma adres `file:///...` — przeglądarka czyta plik z dysku bez serwera. Na zajęciach używamy serwera HTTP.

1. Kliknij „Go Live” na pasku stanu VS Code. Zapisz adres, pod którym otworzyła się strona (adres IP i port). Zmień tekst w `index.html`, zapisz (Ctrl+S) i obserwuj przeglądarkę.
2. Skopiuj folder `nazwisko_imie` do `C:\xampp\htdocs`, w XAMPP Control Panel uruchom moduł Apache i otwórz `http://localhost/nazwisko_imie/zad01/`. Zanotuj, czym różnią się adresy z Live Server i z XAMPP.

4 Narzędzia deweloperskie (F12)

1. Zakładka Elements (Elementy / Inspektor): kliknij prawym przyciskiem nagłówek strony → Zbadaj. Zmień w drzewie DOM tekst nagłówka, a potem odśwież stronę (F5). Co się stało ze zmianą?
2. Zakładka Network (Sieć): zaznacz Disable cache (Wyłącz pamięć podręczną) i odśwież stronę. Zapisz w tabeli nazwę, status, typ i rozmiar każdego pobranego pliku. Który plik ma status 404?
3. Kliknij wiersz z dokumentem HTML i otwórz Headers (Nagłówki). Przepisz metodę żądania, kod stanu oraz wartości nagłówków odpowiedzi `Content-Type` i (jeśli jest) `Server`. Porównaj je dla strony z Live Server i z XAMPP.
4. Zakładka Console (Konsola): przepisz komunikat o błędzie, który pojawił się z powodu brakującego obrazka.

5 Walidator W3C

Walidator sprawdza zgodność kodu ze standardem HTML. Błędy (Error) trzeba poprawić, ostrzeżenia (Warning) warto przeanalizować.

1. Otwórz `do_walidacji.html` w przeglądarce — wygląda prawie poprawnie. Wklej jego kod na stronie validator.w3.org/nu (Check by: text input) i sprawdź. Zapisz liczbę błędów i ostrzeżeń.
2. Poprawiaj kolejne błędy w VS Code i sprawdzaj ponownie, aż zobaczysz komunikat o braku błędów i ostrzeżeń. Dla każdego błędu zanotuj komunikat i swoją poprawkę.
3. Sprawdź walidatorem także `index.html` i `o-mnie.html` (Check by: file upload). Oba pliki mają przejść bez błędów — zrób zrzut ekranu wyniku dla `index.html`.

W sprawozdaniu umieść

- ☐ zrzut struktury folderu `zad01` w VS Code (panel Eksplorator)
- ☐ adresy strony z Live Server i z XAMPP
- ☐ tabela plików z zakładki Network (nazwa, status, typ, rozmiar) i nagłówki odpowiedzi dla `index.html`
- ☐ lista błędów walidatora dla `do_walidacji.html` z poprawkami
- ☐ zrzut wyniku walidacji `index.html` bez błędów
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego zmiana wprowadzona w zakładce Elements znika po odświeżeniu strony?
2. Co oznaczają kody 200 i 404? Kto je wysyła — przeglądarka czy serwer?
3. Do czego służą `<meta charset="utf-8">` i atrybut `lang="pl"`?
4. Dlaczego strona z błędami walidacji często i tak wyświetla się poprawnie? Dlaczego mimo to trzeba je poprawiać?

Kryteria oceny

Kryterium	Punkty
Folder projektu o wymaganej strukturze, strona uruchomiona przez Live Server i XAMPP	3
<code>index.html</code> i <code>o-mnie.html</code> : pełny szkielet HTML5, <code>lang="pl"</code> , wymagane elementy, wzajemne odnośniki	4
Tabela z zakładki Network ze statusami 200 i 404 oraz nagłówki odpowiedzi	4
<code>do_walidacji.html</code> poprawiony — 0 błędów, opisane wszystkie poprawki	4
Walidacja <code>index.html</code> i <code>o-mnie.html</code> bez błędów (zrzut)	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Szkielet semantyczny strony

Czas: 2 × 45 min

praca indywidualna

Cel: Budowa strony ze znacznikami semantycznymi HTML5 i poprawną hierarchią nagłówków oraz porównanie jej z wersją zbudowaną z samych elementów div.

Po zajęciach potrafisz:

- dobiera znaczniki header, nav, main, section, article, aside i footer do roli fragmentu strony
- buduje poprawną hierarchię nagłówków h1–h6 bez przeskakiwania poziomów
- oznacza daty znacznikiem time z atrybutem datetime
- porównuje drzewo dostępności strony semantycznej i strony na div w narzędziach F12
- sprawdza stronę walidatorem W3C i poprawia błędy

Sprzęt i materiały

- komputer z Windows 11
- plik `r1_02_orion_div.html` z folderu Pliki_do_zadan
- folder `nazwisko_imie` z zadania 1

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przygotowanie przed zajęciami

- Przejrzyj z wykładu 1.02 tabelę znaczników semantycznych i zasady hierarchii nagłówków.

Bezpieczeństwo

- Na stronie nie umieszczaj danych osobowych (adresu, telefonu, zdjęć innych osób) — wystarczy imię i klasa.

Przebieg zadania

1 Analiza strony na div

Strona koła astronomicznego „Orion” wygląda w przeglądarce poprawnie, ale jej kod nic nie mówi o znaczeniu poszczególnych części.

1. W folderze `nazwisko_imie` utwórz `zad02` i skopiuj do niego `r1_02_orion_div.html`. Otwórz plik przez Live Server.
2. Na wydruku lub w notatkach przypisz każdemu elementowi `div` znacznik semantyczny, który powinien go zastąpić (np. `div class="naglowek" → header`). Wyniki wpisz do tabeli w sprawozdaniu.
3. Sprawdź plik walidatorem W3C. Zapisz błędy dotyczące nagłówków i wypisz obecną kolejność poziomów nagłówków (h1, h1, h4, ...).
4. Otwórz F12 → Elements → panel Accessibility (Ułatwienia dostępu) i włącz „Enable full-page accessibility tree” (w Firefoksie: zakładka Accessibility). Zrób zrzut drzewa — zwróć uwagę, czy pojawiają się role banner, navigation, main, complementary, contentinfo.

2 Wersja semantyczna

Utwórz nowy plik `index.html` i przenieś do niego treść, zmieniając strukturę. Treść ma zostać ta sama — zmieniasz tylko znaczniki.

```
body
├─ header    (h1 + podtytuł)
├─ nav       (lista ul z 4 odnośnikami)
├─ main
│   ├─ section#o-kole      (h2)
│   ├─ section#spotkania  (h2)
│   └─ section#obserwacje (h2)
│       └─ article × 4     (h3 + time + p)
├─ aside     (h2 Ciekawostka)
└─ footer#kontakt
```

1. Zbuduj szkielet według schematu. Odnośniki menu umieść w liście `ul` wewnątrz `nav` (bez znaków `|`).
2. Ustal hierarchię: jeden `h1` w nagłówku strony, tytuły sekcji jako `h2`, tytuły wpisów jako `h3`. Podtytuł strony to zwykły akapit, nie nagłówek.
3. Każdy wpis obserwacji zamień na `article`, a datę zapisz jako `<time datetime="2026-03-12">12 marca 2026</time>`.
4. Dopisz czwarty `article` o swoim hobby (tytuł h3, data dzisiejsza w `time`, co najmniej dwa zdania). W stopce dodaj akapit z imieniem, nazwiskiem i klasą (`© 2026 ...`).
5. Sprawdź, czy odnośniki z menu nadal przewijają stronę do sekcji (atrybuty `id`).

3 Walidacja i porównanie

1. Sprawdź `index.html` walidatorem W3C i popraw wszystkie błędy. Zaznacz opcję „outline” i zapisz zarys nagłówków (Heading-level outline).
2. Ponownie otwórz drzewo dostępności w F12, tym razem dla `index.html`. Zrób zrzut i wypisz role, które pojawiły się w porównaniu z wersją na div.
3. Porównaj obie wersje w przeglądarce bez stylów — czy wyglądają inaczej? Zapisz wniosek: co daje semantyka, skoro wygląd prawie się nie zmienia?
4. W VS Code użyj Edit → Replace in Files (Ctrl+Shift+H), aby policzyć wystąpienia `<div` w obu plikach (szukaj bez zamiany). Wpisz liczby do sprawozdania.

W sprawozdaniu umieść

- ☐ tabela: element div z pliku startowego → znacznik semantyczny i uzasadnienie
- ☐ błędy walidatora dla wersji div i wynik walidacji `index.html` (0 błędów)
- ☐ zarys nagłówków (outline) przed i po zmianach
- ☐ zrzuty drzewa dostępności obu wersji i lista nowych ról
- ☐ plik `index.html` w folderze `zad02`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się `section` od `article`? Podaj przykład treści dla każdego z nich.
2. Dlaczego na stronie powinien być jeden element `main` i jeden nagłówek `h1`?
3. Komu i do czego służą znaczniki semantyczne (podaj co najmniej trzy przykłady: czytnik ekranu, wyszukiwarka, ...)?
4. Kiedy nadal warto użyć zwykłego `div`?

Kryteria oceny

Kryterium	Punkty
Tabela przypisania div → znacznik semantyczny z uzasadnieniem	3
Szkielet: header, nav z listą, main, 3 section, aside, footer zgodnie ze schematem	5
Hierarchia h1–h3 bez przeskoków, artykuły z <code>time datetime</code>	3
Własny artykuł o hobby i stopka z autorem; działające kotwice menu	2
Walidacja bez błędów, outline i porównanie drzew dostępności (zrzuty, wnioski)	4
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Tekst, listy i odnośniki — mini serwis

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie serwisu z trzech podstron połączonych menu, z poprawnie dobranymi znacznikami tekstu, listami zagnieżdżonymi oraz różnymi rodzajami odnośników.

Po zajęciach potrafisz:

- dobiera znaczniki tekstu (strong, em, mark, q, blockquote, br) do znaczenia treści
- tworzy listy uporządkowane, nieuporządkowane, zagnieżdżone i listę definicji
- stosuje ścieżki względne do plików w podkatalogach i katalogu nadrzędnym (../)
- tworzy kotwice wewnątrz długiej strony oraz odnośniki mailto i tel
- otwiera odnośniki zewnętrzne w nowej karcie z atrybutem rel

Sprzęt i materiały

- komputer z Windows 11
- pliki `r1_03_tresci.txt` i `r1_03_rower.svg` z folderu Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Bezpieczeństwo

- Odnośniki zewnętrzne otwierane w nowej karcie (`target="_blank"`) opatruj atrybutem `rel="noopener noreferrer"` — nowa strona nie dostaje wtedy dostępu do karty, z której ją otwarto.
- W prawdziwym serwisie nie publikuj prywatnych adresów e-mail i numerów telefonów uczniów.

Przebieg zadania

1 Struktura serwisu

Serwis ma trzy podstrony. Plik `trasy.html` leży w podkatalogu — zwróć uwagę na ścieżki.

```
zad03/  
├── index.html  
├── kontakt.html  
├── img/  
│   └── rower.svg  
└── trasy/  
    └── trasy.html
```

1. Utwórz foldery i puste pliki według schematu. Plik `r1_03_rower.svg` zapisz jako `img/rower.svg`.
2. W każdym pliku wpisz szkielet HTML5 (`lang="pl"` , własny `title`) z elementami `header` , `nav` , `main` i `footer` . W stopce podaj imię, nazwisko i klasę, rozdzielone znacznikiem `br` .
3. W `nav` każdej strony umieść listę `ul` z trzema odnośnikami: Strona główna, Trasy, Kontakt.
Uwaga: w `trasy/trasy.html` ścieżki muszą zaczynać się od `../` (np. `../index.html`), a w plikach głównych — nie.
4. W nagłówku `index.html` i `trasy/trasy.html` wstaw rysunek roweru (`width="200"` `height="120"` , sensowny `alt`). Jaka będzie ścieżka do obrazu w każdym z tych plików?

2 Strona główna — tekst i listy

Treść skopiuj z pliku `r1_03_tresci.txt` .

1. Wyróżnij `strong` słowa „każdy uczeń szkoły”, hasło klubu wstaw w znacznik `q` (cudzysłowy doda przeglądarka — nie wpisuj ich), słowo „Najważniejsze” oznacz `mark` , a słowo „zawsze” — `em` .
2. Plan sezonu zapisz jako listę `ul` zagnieżdżoną na dwóch poziomach (pora roku → wydarzenia). Pamiętaj: lista wewnętrzna leży wewnątrz `li` , przed jego znacznikiem zamykającym.
3. Słowniczek zapisz listą definicji `dl` / `dt` / `dd` .
4. Słowa patrona klubu umieść w `blockquote` , a autora cytatu w osobnym akapicie pod cytatem.

3 Długa strona z kotwicami

Kotwica to odnośnik do elementu o danym `id` na tej samej (`#id`) lub innej stronie (`plik.html#id`).

1. W `trasy/trasy.html` opisz pięć tras — każda w osobnej `section` z `id` (`trasa1` ... `trasa5`) i nagłówkiem `h2` . Długość trasy wyróżnij `strong` .
2. Na początku `main` utwórz spis treści jako listę `ol` z odnośnikami `#trasa1` ... oraz do sekcji z ekipunkiem.
3. Nadaj `header` identyfikator `gora` i pod każdą sekcją dodaj odnośnik „Do góry”. Zmniejsz okno przeglądarki i sprawdź działanie wszystkich kotwic.
4. Listę rzeczy do zabrania zapisz jako `ol` . Dodaj odnośnik do OpenStreetMap otwierany w nowej karcie z `rel="noopener noreferrer"` .
5. Na stronie głównej dodaj zdanie z odnośnikiem prowadzącym bezpośrednio do opisu trasy nad jezioro (plik i kotwica w jednym adresie).

4 Kontakt i sprawdzenie

1. W `kontakt.html` dodaj odnośnik `mailto:` z tematem wiadomości „Zapis do klubu” (spację w adresie zapisz jako `%20`) oraz odnośnik `tel:` (numer bez spacji, z `+48`). Odnośnik do PTTK otwórz w nowej karcie.
2. Przejdź cały serwis wyłącznie przez menu i kotwice. W F12 → Network sprawdź, czy żaden plik nie ma statusu 404.
3. Sprawdź wszystkie trzy pliki walidatorem W3C i popraw błędy.

W sprawozdaniu umieść

- ☐ zrzut struktury folderu `zad03` w VS Code
- ☐ tabela odnośników: skąd → dokąd, wpisana ścieżka, rodzaj (względny, kotwica, zewnętrzny, mailto, tel)
- ☐ zrzut strony głównej z listą zagnieżdżoną i słowniczkiem
- ☐ wynik walidacji trzech plików (0 błędów)
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się ścieżka względna `img/rower.svg` od `../img/rower.svg` i od `/img/rower.svg`?
2. Kiedy użyjesz `strong`, a kiedy `mark`? Dlaczego nie warto używać `b` tylko po to, by pogrubić tekst?
3. Do czego służy `rel="noopener noreferrer"` przy odnośniku z `target="_blank"`?
4. Co się stanie po kliknięciu odnośnika `mailto:` na komputerze bez skonfigurowanego programu pocztowego?

Kryteria oceny

Kryterium	Punkty
Struktura folderów i działające menu na 3 stronach (poprawne <code>../</code> w podkatalogu), obraz widoczny na obu stronach	4
Znaczniki tekstu: strong, em, mark, q, blockquote, br użyte zgodnie z treścią	3
Lista zagnieżdżona (2 poziomy), lista ol i lista definicji dl	3
Spis treści z 6 kotwicami, odnośniki „Do góry”, odnośnik plik + kotwica ze strony głównej	3
mailto z tematem, tel, dwa odnośniki zewnętrzne z target i rel	2
Walidacja bez błędów, brak 404 w Network, tabela odnośników	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Obrazy i galeria

Czas: 2 × 45 min

praca indywidualna

Cel: Przygotowanie grafiki do publikacji w sieci (skalowanie, kompresja, format WebP) i zbudowanie galerii miniatur z poprawnie opisanymi obrazami.

Po zajęciach potrafisz:

- zmienia rozmiar obrazu i eksportuje go do JPEG i WebP z wybraną jakością
- porównuje rozmiary plików w różnych formatach i rozdzielczościach
- wstawia obraz z atrybutami alt, width, height i loading
- grupuje obraz z podpisem w figure i figcaption
- tworzy galerię miniatur prowadzących do pełnych zdjęć

Sprzęt i materiały

- komputer z Windows 11
- folder `r1_04_zdjecia` z Pliki_do_zadan (6 plików JPEG i 1 PNG) lub własne zdjęcia (bez osób i danych prywatnych)

Oprogramowanie

- **GIMP 3** — skalowanie i eksport do WebP gimp.org/downloads
- **Squoosh (alternatywa w przeglądarce)** — porównanie jakości przed i po kompresji suwakiem squoosh.app
- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Validator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przygotowanie przed zajęciami

- Możesz przynieść 3–6 własnych zdjęć krajobrazów lub przedmiotów (bez twarzy innych osób). Zdjęcia z internetu tylko z wolnych licencji z podaniem autora.

Bezpieczeństwo

- Zdjęcia z telefonu zawierają metadane EXIF (m.in. położenie GPS). Przed publikacją usuń je: w GIMP przy eksporcie odznacz zapis metadanych Exif/XMP; Squoosh nie zapisuje metadanych.

Przebieg zadania

1 Rozmiar i format pliku

Zdjęcie 3200 × 2400 px jest kilka razy większe, niż potrzeba na stronie. Przygotujesz dwie wersje każdego zdjęcia: dużą 1600 × 1200 px i miniaturę 400 × 300 px.

```
zad04/  
├─ index.html  
└─ img/  
    ├─ duze/    (1600 × 1200, WebP)  
    └─ min/     (400 × 300, WebP)
```

1. Utwórz strukturę folderów. W Eksploratorze Windows odczytaj rozmiar plików `jezioro_rano.jpg` i `jezioro_rano.png` (Właściwości) — wpisz do tabeli.
2. Otwórz `jezioro_rano.jpg` w GIMP. Wybierz Obraz → Skaluj obraz..., ustaw szerokość 1600 px (wysokość przeliczy się sama przy zamkniętym łańcuchu) i zatwierdź.
3. Wyeksportuj obraz (Plik → Eksportuj jako..., Shift+Ctrl+E) trzy razy: `test_80.jpg` (jakość 80), `test_80.webp` (jakość 80) i `test.png`. Zapisz rozmiary w tabeli.
4. Powiększ fragment z odbiciami na wodzie w każdym pliku. Czy widzisz różnicę jakości? Sprawdź też eksport WebP z jakością 40.

2 Przygotowanie zdjęć do galerii

1. Dla każdego z 6 zdjęć przygotuj wersję 1600 × 1200 px (WebP, jakość 80) w `img/duze` i miniaturę 400 × 300 px (WebP, jakość ok. 75) w `img/min`. Zachowaj te same nazwy plików bez polskich znaków.
2. Zamiast GIMP możesz użyć Squoosh: wczytaj zdjęcie, w panelu Resize ustaw szerokość, w Compress wybierz WebP, przesun suwak podziału i porównaj obie wersje.
3. Zsumuj rozmiar folderu `img` i porównaj z rozmiarem sześciu oryginałów.

3 Galeria w HTML

Każde zdjęcie to `figure` z podpisem. Miniatura jest odnośnikiem do dużego zdjęcia.

```
<figure>  
  <a href="img/duze/...webp">  
      
  </a>  
  <figcaption>...</figcaption>  
</figure>
```

1. Utwórz `index.html` ze szkieletem (header z h1, main, footer). W `main` umieść sekcję „Zdjęcia” z sześcioma elementami `figure` według wzoru.
2. Uzupełnij `alt` opisem tego, co widać na zdjęciu (jedno zdanie), a nie słowami „zdjęcie 1”. Podpis `figcaption` to krótki tytuł.
3. Podaj rzeczywiste wymiary miniatury w `width` i `height`. Dodaj `loading="lazy"`.
4. W drugiej sekcji opisz wyniki porównania rozmiarów plików (2–3 zdania). W stopce podaj źródło zdjęć i autora strony.
5. Sprawdź działanie galerii: każda miniatura otwiera właściwe duże zdjęcie. Sprawdź stronę walidatorem.

4 Pomiar w narzędziach F12

1. Otwórz F12 → Network, zaznacz Disable cache i filtr Img, odśwież stronę. Zapisz łączny rozmiar przesłanych obrazów (pasek na dole).
2. Tymczasowo zamień w jednej `figure` miniaturę na oryginał `jezioro_rano.jpg` (z folderu startowego) i porównaj rozmiar przesłanych danych. Przywróć miniaturę.
3. Usuń na chwilę `width` i `height` z jednego obrazu, w Network ustaw ograniczenie przepustowości (No throttling → Slow 4G) i odśwież. Opisz, co dzieje się z układem treści podczas wczytywania. Przywróć atrybuty.

W sprawozdaniu umieść

- ☐ tabela rozmiarów plików (format, wymiary, jakość, rozmiar)
- ☐ zrzut galerii w przeglądarce i zrzut zakładki Network z filtrem Img
- ☐ wnioski z porównania formatów i z testu bez width/height
- ☐ folder `zad04` z `index.html` i obrazami
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Do jakich obrazów wybierzesz JPEG, PNG, WebP i SVG? Podaj po jednym przykładzie.
2. Po co podaje się `width` i `height` obrazu, skoro przeglądarka i tak zna jego wymiary po pobraniu?
3. Jak powinien wyglądać tekst `alt` dla zdjęcia w galerii, a jak dla obrazu czysto dekoracyjnego?
4. Dlaczego nie wystarczy wstawić oryginału 3200 px i zmniejszyć go atrybutem `width="400"`?

Kryteria oceny

Kryterium	Punkty
Tabela porównania formatów (co najmniej 5 wierszy) z poprawnymi rozmiarami i wnioskiem	4
6 dużych zdjęć 1600 × 1200 i 6 miniatur 400 × 300 w WebP w osobnych folderach	4
Galeria: 6 × figure z figcaption, miniatura jako odnośnik do dużego zdjęcia	4
Atrybuty alt (opisowe), width, height, loading; walidacja bez błędów	3
Pomiary w Network i opis testu bez width/height	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Tabele HTML

Czas: 2 × 45 min

praca indywidualna

Cel: Budowanie tabel danych z łączeniem komórek oraz elementami poprawiającymi strukturę i dostępność (caption, thead, tbody, tfoot, scope).

Po zajęciach potrafisz:

- łączy komórki w poziomie (colspan) i w pionie (rowspan)
- dzieli tabelę na thead, tbody i tfoot oraz dodaje tytuł caption
- oznacza nagłówki kolumn i wierszy atrybutem scope
- odtwarza tabelę z obrazka, planując układ komórek na kartce
- sprawdza poprawność tabeli walidatorem W3C

Sprzęt i materiały

- komputer z Windows 11, kartka i ołówek do szkicu siatki
- pliki `r1_05_dane.txt`, `r1_05_wzor.png` i `r1_05_tabele.css` z folderu Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przebieg zadania

1 Przygotowanie

Arkusze `tabele.css` tylko rysuje krawędzie komórek i koloruje nagłówki — style nie są przedmiotem tego zadania.

```
<head>
  ...
  <link rel="stylesheet" href="tabele.css">
</head>
```

1. Utwórz folder `zad05` i skopiuj do niego `r1_05_tabele.css` pod nazwą `tabele.css`.
2. Utwórz trzy pliki: `plan.html`, `wyniki.html`, `cennik.html`, każdy z pełnym szkieletem HTML5 i dołączonym arkuszem.

2 Plan lekcji — colspan i rowspan

Zasada: komórka z `rowspan="3"` zajmuje miejsce w trzech wierszach, więc w dwóch kolejnych wierszach pomijasz po jednej komórce `td`.

1. Narysuj na kartce siatkę 6 × 7 (kolumna godzin + 5 dni, wiersz nagłówek + 6 godzin) i zaznacz komórki połączone według `r1_05_dane.txt`.
2. W `plan.html` zbuduj tabelę z tytułem `caption`. Pierwszy wiersz (dni tygodnia) umieść w `thead`, a godziny jako `th scope="row"` na początku każdego wiersza `tbody`.
3. Połącz komórki: Witryny (2 godziny), Pracownia urządzeń (3 godziny), Systemy operacyjne (2 godziny), przerwę obiadową (5 dni) i zajęcia z doradcą (środa + czwartek).
4. Policz w każdym wierszu `tr` komórki z uwzględnieniem `rowspan` z wierszy wyżej — zawsze musi wyjść 6. Sprawdź tabelę walidatorem — ostrzeżenie „A table row was 7 columns wide and exceeded the column count established by the first row” oznacza złe liczenie.

3 Tabela danych — thead, tbody, tfoot

1. W `wyniki.html` zbuduj tabelę wyników turnieju: `caption` z zasadami punktacji, `thead` z nagłówkami (`scope="col"`), `tbody` z pięcioma zawodnikami (imię i nazwisko jako `th scope="row"`).
2. Dodaj kolumnę Suma — oblicz ją dla każdego zawodnika.
3. Dodaj `tfoot` z wierszem „Średnia”: komórka z napisem łączy dwie pierwsze kolumny, a w pozostałych wpisz średnią punktów w rundzie i średnią sumę (zaokrąglone do 0,1).
4. Sprawdź w F12 → Elements, czy przeglądarka nie dodała do tabeli elementów, których nie ma w Twoim kodzie (np. dodatkowego `tbody`).

4 Tabela z obrazka (styl egzaminu INF.03)

W arkuszach egzaminacyjnych często trzeba odtworzyć tabelę z rysunku. Najpierw zaplanuj, potem pisz.

1. Obejrzyj `r1_05_wzor.png`. Ustal liczbę kolumn i wierszy siatki oraz wypisz wszystkie komórki połączone (gdzie, `colspan` czy `rowspan`, ile).
2. W `cennik.html` odtwórz tabelę: dwuwierszowy nagłówek w `thead`, kategorie (Narty, Snowboard, Inne) jako `th`, kaucja łączona dla całej kategorii, cena kasku wspólna dla dorosłych i dzieci.
3. Dla nagłówka „Cena za dzień” użyj `scope="colgroup"`, a dla kategorii obejmujących kilka wierszy `scope="rowgroup"`.
4. Porównaj wynik z obrazkiem (np. ustawiając okna obok siebie) i sprawdź walidatorem wszystkie trzy pliki. Zrób zrzuty ekranu tabel.

W sprawozdaniu umieść

- ☐ szkic siatki planu lekcji z zaznaczonymi połączeniami (zdjęcie kartki lub rysunek)
- ☐ zrzuty trzech tabel
- ☐ lista połączonych komórek w tabeli z obrazka
- ☐ wynik walidacji trzech plików (0 błędów)
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego tabel nie używa się do budowania układu strony?
2. Co zyskuje użytkownik czytnika ekranu dzięki `caption` i atrybutowi `scope`?
3. W jakiej kolejności mogą wystąpić `thead`, `tbody` i `tfoot` w kodzie i gdzie przeglądarka wyświetli stopkę tabeli?
4. Jak sprawdzisz, że w wierszu jest właściwa liczba komórek, gdy wyżej występuje `rowspan`?

Kryteria oceny

Kryterium	Punkty
Plan lekcji: 5 połączeń (3 × rowspan, 2 × colspan) zgodnych z opisem, caption, thead	5
Wyniki: thead/tbody/tfoot, th scope col/row, poprawne sumy i średnie	4
Cennik zgodny z obrazkiem: 2-wierszowy nagłówek, wszystkie łączenia, scope colgroup/rowgroup	5
Walidacja trzech plików bez błędów, szkic siatki i zrzuty	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Formularze HTML

Czas: 2 × 45 min

praca indywidualna

Cel: Budowa formularza zgłoszeniowego z różnymi typami pól, poprawnymi etykietami i walidacją natywną oraz obserwacja danych wysyłanych metodą GET.

Po zajęciach potrafisz:

- dobiera typ pola input do rodzaju danych (text, email, tel, date, number, range, radio, checkbox)
- łączy każde pole z etykietą label (for + id) i grupuje pola w fieldset z legend
- stosuje atrybuty walidacji required, pattern, min, max, maxlength
- odczytuje wysłane pary nazwa=wartość w pasku adresu i w F12 → Network
- wyjaśnia, dlaczego walidacja w przeglądarce nie zastępuje sprawdzania danych na serwerze

Sprzęt i materiały

- komputer z Windows 11
- folder `r1_06_start` z Pliki_do_zadan (plik `potwierdzenie.html`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.05 różnicę między metodami GET i POST.

Bezpieczeństwo

- Metodą GET dane trafiają do adresu URL, historii przeglądarki i logów serwera — nigdy nie wysyłaj tak haseł ani danych wrażliwych.
- Do testów wpisuj dane fikcyjne (adresy z domeny `example.com`).

Przebieg zadania

1 Szkielet formularza

Formularz zapisów na Szkolny Bieg na Orientację wyśle dane do strony `potwierdzenie.html`, która pokaże, co dotarło.

```
<form action="potwierdzenie.html" method="get">
  <fieldset>
    <legend>Dane uczestnika</legend>
    <p>
      <label for="imie">Imię *</label>
      <input type="text" id="imie" name="imie" required>
    </p>
    ...
  </fieldset>
</form>
```

1. Utwórz folder `zad06`, skopiuj do niego `potwierdzenie.html` (nie zmieniaj go) i utwórz `index.html` z nagłówkiem strony, `main` i stopką.
2. Każde pole umieść w akapicie, z etykietą `label` połączoną przez `for` i `id`. Atrybut `name` jest obowiązkowy — bez niego pole nie zostanie wysłane.

2 Pola formularza

Pola oznaczone * mają atrybut `required`.

1. Fieldset „Dane uczestnika”: imię * i nazwisko * (text, `maxlength` 30 i 40), e-mail * (email), telefon opiekuna (tel, dokładnie 9 cyfr — `pattern="[0-9]{9}"` i odpowiedź w `title`), data urodzenia * (date, od 2006-01-01 do 2013-12-31).
2. Klasa * — lista `select` z pierwszą opcją „— wybierz —” o pustej wartości oraz dwiema grupami `optgroup`: Technik informatyk (1T, 2T, 3T) i Technik programista (1P, 2P, 3P).
3. Fieldset „Udział w zawodach”: trasa * jako trzy przyciski `radio` o wspólnej nazwie (3 km, 5 km, 8 km), kod drużyny (3 wielkie litery, myślnik, 2 cyfry, np. ABC-12), liczba osób towarzyszących (number 0–3, domyślnie 0), ocena kondycji (range 1–10).
4. Trzy pola `checkbox` o wspólnej nazwie `dodatki` (posiłek, koszulka, transport) oraz `textarea` „Uwagi” (4 wiersze, maks. 300 znaków).
5. Poza fieldsetami: obowiązkowa zgoda na regulamin (checkbox) oraz przyciski `button` typu submit i reset.

3 Wysłanie i obserwacja danych

1. Uruchom stronę przez Live Server. Kliknij etykietę „Imię” — kursor powinien przejść do pola. Sprawdź to dla kilku pól (to test poprawności `for / id`).
2. Wypełnij formularz poprawnymi danymi (zaznacz 2 dodatki) i wyślij. Przepisz z paska adresu część po znaku `?`. Jak zapisano znak `@` i spacje?
3. Wróć, wyślij ponownie z otwartym F12 → Network. Kliknij żądanie `potwierdzenie.html` → Payload (ładunek) i zrób zrzut parametrów.
4. Odpowiedz na podstawie tabeli w `potwierdzenie.html`: czy wysłano niezaznaczone checkboxy? Ile razy pojawia się `dodatki`? Co wysłał pusty telefon? Czy wysłano przyciski?

4 Walidacja natywna i jej ograniczenia

1. Przetestuj i zapisz komunikaty przeglądarki dla: pustego imienia, e-maila `ola@`, telefonu z 8 cyframi, kodu `abc-12`, daty spoza zakresu, niezaznaczonej trasy.
2. W F12 → Elements usuń atrybut `required` z pola nazwisko (prawy przycisk → Edit attribute lub Delete) i wyślij formularz z pustym nazwiskiem. Co się stało? Zapisz wniosek o zaufaniu do danych z przeglądarki.
3. Dodaj tymczasowo do znacznika `form` atrybut `novalidate` i sprawdź, czy formularz z błędami zostanie wysłany. Usuń atrybut.
4. Sprawdź `index.html` walidatorem W3C i popraw błędy.

W sprawozdaniu umieść

- ☐ zrzut wypełnionego formularza i zrzut tabeli z `potwierdzenie.html`
- ☐ część adresu po `?` oraz zrzut zakładki Payload w F12
- ☐ tabela testów walidacji natywnej (pole, wpisana wartość, komunikat)
- ☐ wnioski z testów usunięcia `required` i `novalidate`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się atrybut `id` od `name` w polu formularza? Który z nich trafia do serwera?
2. Dlaczego przyciski radio jednej grupy muszą mieć ten sam `name`, a różne `value`?
3. Kiedy wybierzesz metodę GET, a kiedy POST? Podaj po jednym przykładzie.
4. Dlaczego dane z formularza trzeba sprawdzić ponownie na serwerze, mimo atrybutów `required` i `pattern`?

Kryteria oceny

Kryterium	Punkty
Wszystkie wymagane typy pól (text, email, tel, date, select z optgroup, radio, number, range, checkbox, textarea, button)	5
Etykiety label for/id dla każdego pola, 2 × fieldset z legend, poprawne atrybuty name/value	3
Atrybuty walidacji: required, pattern (2), min/max, maxlength zgodnie z treścią	3
Obserwacja wysłanych danych: adres, Payload, odpowiedzi o checkboxach i pustych polach	3
Tabela testów walidacji i wnioski z testów required/novalidate; walidator bez błędów	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Multimedia i dostępność

Czas: 2 × 45 min

praca indywidualna

Cel: Osadzanie wideo z napisami, dźwięku i mapy oraz przeprowadzenie audytu dostępności strony i poprawienie wykrytych problemów.

Po zajęciach potrafisz:

- osadza wideo i audio z kilkoma źródłami i treścią zastępczą
- przygotowuje plik napisów WebVTT i dołącza go elementem track
- osadza mapę w iframe z atrybutem title i loading="lazy"
- przeprowadza audyt dostępności (Lighthouse, WAVE lub axe DevTools) i interpretuje wyniki
- poprawia typowe błędy: brak alt, lang, etykiet, niski kontrast, kolejność nagłówków, nazwy przycisków

Sprzęt i materiały

- komputer z Windows 11 i słuchawkami
- folder `r1_07_start` z Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Chrome lub Edge — Lighthouse** — F12 → zakładka Lighthouse, kategoria Accessibility
- **WAVE (rozszerzenie) lub axe DevTools (wersja bezpłatna)** — axe DevTools: <https://www.deque.com/axe/devtools/> — wystarczy jedno z narzędzi wave.webaim.org/extension
- **Validator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.06 zasady WCAG: tekst alternatywny, kontrast co najmniej 4,5:1 dla zwykłego tekstu, etykiety pól.

Bezpieczeństwo

- Osadzaj w iframe tylko zaufane serwisy (np. mapy, filmy z oficjalnych kanałów) — osadzona strona może śledzić użytkownika i wykonywać własne skrypty.
- Napisy i plik .vtt wczytują się tylko przez serwer (Live Server) — z adresu `file:///` przeglądarka ich nie pobierze.

Przebieg zadania

1 Audyt strony startowej

Strona Eko-tygodnia wygląda poprawnie, ale ma wiele problemów dostępności. Najpierw je zmierz.

1. Skopiuj zawartość `r1_07_start` do folderu `zad07` i otwórz `index.html` przez Live Server.
2. Uruchom F12 → Lighthouse, zaznacz tylko kategorię Accessibility, tryb Navigation, urządzenie Desktop i kliknij Analizę page load. Zapisz wynik punktowy i listę problemów.
3. Uruchom drugie narzędzie (WAVE lub axe DevTools → Scan). Wpisz do tabeli wszystkie problemy: opis, element, sposób poprawy.
4. Sprawdź stronę walidatorem W3C — które błędy pokrywają się z wynikami audytu?
5. Spróbuj obsłużyć stronę samą klawiaturą (Tab, Shift+Tab, Enter, spacja). Czy wiesz, gdzie jest fokus i do czego służy przycisk formularza?

2 Wideo z napisami i dźwięk

Film nie ma lektora — napisy z komentarzem przydadzą się osobom niesłyszącym i oglądającym bez dźwięku.

WEBVTT

1

00:00:00.000 --> 00:00:05.000

Witamy! Pokażemy, jak segregujemy
odpady w naszej szkole.

2

00:00:05.000 --> ...

1. Na podstawie `komentarz_do_napisow.txt` utwórz plik `media/segregacja.vtt` (kodowanie UTF-8, 5 napisów, każdy w najwyżej dwóch wierszach).
2. Zamień element `video` tak, aby miał dwa elementy `source` (najpierw WebM, potem MP4 z atrybutem `type`), element `track` (`kind="subtitles"`, `srclang="pl"`, `label="Polski"`, `default`) oraz treść zastępczą z odnośnikiem do pobrania filmu.
3. Dodaj sekcję z elementem `audio` (plik `media/dzwonek.mp3`, `controls`, treść zastępcza). Nie używaj `autoplay`.
4. Odtwórz film i sprawdź, czy napisy zmieniają się zgodnie z planszami. W menu odtwarzacza wyłącz i włącz napisy.

3 Mapa w iframe

1. Na openstreetmap.org znajdź swoją szkołę lub miejscowość, wybierz Udostępnij (ikona po prawej) → HTML, zaznacz „Dołącz znacznik” i skopiuj kod.
2. Wklej `iframe` zamiast odnośnika „kliknij tutaj”. Dodaj atrybuty `title` (opis mapy) i `loading="lazy"`. Pod mapą zostaw odnośnik z opisową treścią do pełnej mapy.

4 Poprawki dostępności i ponowny audyt

1. Popraw stronę: `lang="pl"`, struktura `header / main / footer`, nagłówki `h2` zamiast `h4`, tekst alternatywny plakatu, kontrast tekstu (np. `#555555` zamiast `#b5b5b5` — sprawdź współczynnik w F12 → Elements, klikając próbkę koloru), etykiety `label` dla pól formularza.
2. Przycisk wysyłania ma nazwę tylko z obrazka bez `alt`. Dodaj widoczny tekst „Wyślij zgłoszenie”, a ikonę nadaj `alt=""` (obraz dekoracyjny).
3. Powtórz audyt Lighthouse i drugim narzędziem. Celem jest 100 pkt i brak błędów. Zapisz wynik po zmianach.
4. Sprawdź `index.html` walidatorem i powtórz test obsługi klawiaturą.

W sprawozdaniu umieść

- ☐ tabela problemów dostępności (problem, element, poprawka) i wyniki Lighthouse przed i po
- ☐ plik `segregacja.vtt` i zrzut filmu z widocznym napisem
- ☐ zrzut strony z mapą i odtwarzaczem audio
- ☐ wynik walidacji `index.html` (0 błędów)
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różnią się napisy `subtitles` od `captions`? Które lepiej służą osobom niesłyszącym?
2. Dlaczego podaje się kilka elementów `source` z atrybutem `type`?
3. Dlaczego ikona w przycisku z tekstem ma pusty `alt`, a plakat — opisowy?
4. Czy wynik 100 w Lighthouse oznacza, że strona jest w pełni dostępna? Uzasadnij.
5. Jakie obowiązki wprowadza Europejski akt o dostępności (od 28.06.2025) i kogo dotyczy?

Kryteria oceny

Kryterium	Punkty
Audyt przed zmianami: wynik Lighthouse i tabela co najmniej 6 problemów z poprawkami	4
Plik .vtt zgodny z komentarzem (5 napisów, poprawny format czasu) i działające napisy	4
video z 2 × source, track i treścią zastępczą; audio bez autoplay	3
Mapa w iframe z title i loading="lazy" oraz opisowy odnośnik	2
Poprawki dostępności: lang, landmarki, nagłówki, alt, kontrast, etykiety, nazwa przycisku; ponowny audyt bez błędów	4
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Zadanie kontrolne: struktura HTML

Czas: 2 × 45 min

praca samodzielna — sprawdzian praktyczny (bez pomocy innych osób; wolno korzystać z dokumentacji MDN)

Cel: Samodzielne zbudowanie w ograniczonym czasie kompletnej strony HTML (bez CSS) na podstawie opisu i zrzutu ekranu, z poprawną semantyką i bez błędów walidacji.

Po zajęciach potrafisz:

- planuje strukturę strony na podstawie opisu i zrzutu
- stosuje znaczniki semantyczne, listy, tabelę, obraz z podpisem i formularz
- dba o dostępność: alt, label, caption, scope, lang
- sprawdza i poprawia kod walidatorem W3C w wyznaczonym czasie

Sprzęt i materiały

- komputer z Windows 11
- pliki `r1_08_teksty.txt`, `r1_08_kajak.png` i `r1_08_wzor.png` z folderu Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu
- **MDN Web Docs** — jedyne dozwolone źródło pomocy developer.mozilla.org/pl

Przygotowanie przed zajęciami

- Powtórz zadania 2–7: znaczniki semantyczne, listy, tabele (colspan, caption, scope), formularze (label, typy pól), obraz z figure.

Bezpieczeństwo

- Pracujesz samodzielnie — nie korzystaj z chatów, narzędzi AI ani cudzych plików. Pliki oddajesz w folderze `nazwisko_imie/zad08`.

Przebieg zadania

1 Przygotowanie (5 min)

Masz 75 minut na stronę i 10 minut na walidację i oddanie pracy. Teksty kopiuj z pliku — nie przepisyj ich ręcznie.

1. Utwórz folder `zad08`, skopiuj do niego `r1_08_kajak.png` jako `kajak.png`. Utwórz `index.html`.
2. Obejrzyj `r1_08_wzor.png` — tak ma wyglądać strona bez stylów. Wygląd domyślny (czcionki, odstępy) może się nieznacznie różnić w zależności od przeglądarki.

2 Wymagania dotyczące strony

Kolejność bloków jak na zrzucie. Strona nie ma żadnych stylów CSS ani atrybutu `style`.

1. Dokument: `lang="pl"`, kodowanie UTF-8, viewport, tytuł karty „Przystań kajakowa Wiosło”.
2. `header` z nagłówkiem pierwszego stopnia i podtytułem w akapicie. `nav` z listą czterech odnośników do kotwic `#oferta`, `#cennik`, `#rezerwacja`, `#kontakt`.
3. `main` z trzema sekcjami (`id` jak w menu), każda z nagłówkiem drugiego stopnia.
4. Sekcja Oferta: akapit (wyróżnij „kamizelkę asekuracyjną” znacznikiem ważności), obraz w `figure` z podpisem, opisem alternatywnym z pliku tekstowego oraz wymiarami 480 × 240, nagłówek trzeciego stopnia „Trasy” i lista uporządkowana.
5. Sekcja Cennik: tabela z tytułem, nagłówkami kolumn (`thead`) i nagłówkami wierszy (nazwa trasy) z atrybutem `scope`; ostatni wiersz to jedna komórka na całą szerokość. Pod tabelą nagłówek „W cenie” i lista nieuporządkowana.
6. Sekcja Rezerwacja: formularz wysyłany metodą POST; pola zgodnie z plikiem tekstowym — odpowiednie typy (text, email, date, select, number 1–10), wymagane pola z `required`, każde pole z etykietą, przycisk „Zarezerwuj”.
7. `aside` z nagłówkiem drugiego stopnia „Ciekawostka”. `footer` z `id="kontakt"`: adres, numer telefonu jako odnośnik `tel:` i drugi akapit z Twoim imieniem i nazwiskiem.

3 Sprawdzenie i oddanie pracy (10 min)

1. Sprawdź działanie odnośników menu i kliknij każdą etykietę formularza (fokus w polu).
2. Sprawdź `index.html` walidatorem W3C (file upload) i popraw wszystkie błędy. Zrób zrzut wyniku walidacji.
3. Zrób zrzut całej strony (np. F12 → Ctrl+Shift+P → „Capture full size screenshot”) i umieść oba zrzuty w folderze `zad08`.

W sprawozdaniu umieść

- ☐ folder `zad08` z plikami `index.html` i `kajak.png`
- ☐ zrzut całej strony i zrzut wyniku walidacji
- ☐ krótka samoocena: które wymagania spełniłeś, a których nie zdążyłeś wykonać

Pytania kontrolne

1. Dlaczego tabela cennika ma nagłówki wierszy (th z `scope="row"`), a nie tylko nagłówki kolumn?
2. Który element na stronie najtrudniej było odtworzyć ze zrzutu i dlaczego?
3. Jak sprawdzisz bez walidatora, czy etykiety formularza są poprawnie połączone z polami?

Kryteria oceny

Kryterium	Punkty
Szkielet: lang, meta charset i viewport, title; header z h1 i podtytułem; nav z listą 4 kotwic działających	3
main z 3 sekcjami o właściwych id i h2; aside i footer#kontakt z tel: i autorem	3
Oferta: strong, figure + figcaption, img z alt, width i height, h3 i lista ol	3
Cennik: caption, thead, th scope col/row, wiersz z colspan="3", lista ul	4
Formularz: method post, 5 pól właściwych typów, label for/id, required, min/max, przycisk	4
Walidacja W3C bez błędów (−1 pkt za każdy błąd), brak CSS, zrzuty w folderze	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

CSS — selektory i kaskada

Czas: 2 × 45 min

praca indywidualna

Cel: Dołączenie zewnętrznego arkusza CSS i stosowanie różnych rodzajów selektorów oraz zrozumienie kaskady i specyficzności na podstawie przewidywań sprawdzanych w narzędziach F12.

Po zajęciach potrafisz:

- dołącza zewnętrzny arkusz stylów elementem link
- stosuje selektory elementu, klasy, id, potomka, dziecka, sąsiada, atrybutu i pseudoklasy
- oblicza specyficzność selektora i przewiduje, która reguła wygra
- wyjaśnia działanie kolejności reguł, stylu inline i !important
- odczytuje w F12 → Styles zastosowane i przekreślone deklaracje

Sprzęt i materiały

- komputer z Windows 11
- folder `r1_09_start` z Pliki_do_zadan (możesz też użyć własnej strony z zadania 2, ale wtedy dopisz brakujące elementy: tabelę, klasy `uwaga`, `wazne`, `wyrozniione`, odnośnik zewnętrzny)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator CSS W3C** — sprawdzanie pliku style.css jigsaw.w3.org/css-validator

Przygotowanie przed zajęciami

- Powtórz z wykładu 1.07 sposób liczenia specyficzności (id, klasa/atribut/pseudoklasa, element).

Przebieg zadania

1 Przewidź kolor — specyficzność

Specyficzność zapisujemy jako trójkę (id, klasy, elementy), np. `div .info` = (0,1,1). Przy remisie wygrywa reguła zapisana później. Styl inline i `!important` mają pierwszeństwo.

1. Otwórz `specyficznosc.html` w VS Code (nie w przeglądarce!). Dla każdego akapitu P1-P8 wypisz pasujące reguły z ich specyficznością i przewidź kolor. Wpisz do tabeli w sprawozdaniu.
2. Dopiero teraz otwórz plik w przeglądarce, sprawdź kolory i dla każdego akapitu zajrzyj do F12 → Elements → Styles: która deklaracja wygrała, które są przekreślone? Zaznacz w tabeli trafione przewidywania.
3. Dla P6 i P7 wyjaśnij, co zdecydowało o kolorze. Dla P8 wyjaśnij, dlaczego nie działa reguła z `:nth-child(2)`.

2 Zewnętrzny arkusz stylów

Każdą regułę poprzedź komentarzem z numerem, np. `/* S5. Selektor potomka */`.

```
zad09/  
├─ index.html  
├─ specyficzosc.html  
└─ css/  
    └─ style.css
```

1. Skopiuj pliki startowe do `zad09`, utwórz `css/style.css` i dołącz go w `head` strony `index.html` (zastąp komentarz). Wpisz w stopce swoje dane.
2. S1. `body`: krój bezszeryfowy z listą zapasowych krojów, kolor tekstu `#222222`, tło `#f4f6fb`.
3. S2. Element o `id="naglowek"`: tło `#14213d`, biały tekst, `padding: 16px`.
4. S3. Jedną regułą (grupowanie) nadaj `h2` i `h3` kolor `#1d4e89`.
5. S4. Tylko akapit będący bezpośrednim dzieckiem `header`: kursywa. S5. Odnośniki wewnątrz `nav`: pogrubione, bez podkreślenia. S6. Po najechaniu kursorem: kolor `#e76f51` i podkreślenie.
6. S7. Klasa `uwaga`: kolor `#9b2226`, pogrubienie. S8. Tylko `section` z klasą `wazne`: lewa ramka 6px w kolorze `#fca311` i `padding-left: 10px`.
7. S9. Co drugi wiersz `tbody` (parzyste) — tło `#dde7f5`; dodaj też krawędzie komórek tabeli. S10. Odnośniki, których `href` zaczyna się od `https` — kolor `#2a9d8f`.
8. S11. Akapit leżący bezpośrednio po `h3` (data wpisu): rozmiar 0.9em, kolor `#555555`. S12. Pierwszy odnośnik menu (`:first-child`): wielkie litery (`text-transform`). S13. `article` z klasą `wyznione`: tło `#fff3c4`, `padding: 8px`.

3 Sprawdzenie w F12 i walidacja

1. Zbadaj (F12) podtytuł strony i datę dowolnego wpisu — zapisz, które reguły z Twojego arkusza na nie działają i jakie właściwości są dziedziczone z `body` (sekcja Inherited from).
2. W zakładce Styles wyłącz chwilowo (checkbox) regułę S9 i włącz ją z powrotem. Zmień w F12 kolor z S7 i sprawdź, czy zmiana zapisła się w pliku.
3. Sprawdź `style.css` walidatorem CSS W3C, a `index.html` walidatorem HTML. Popraw błędy i zrób zrzut strony.

W sprawozdaniu umieść

- ☐ tabela przewidywań P1–P8 (reguły, specyficzność, przewidziany i rzeczywisty kolor)
- ☐ wyjaśnienie wyników P6, P7 i P8
- ☐ plik `css/style.css` z regułami S1–S13 i zrzut strony
- ☐ zrzut F12 → Styles z przekreśloną deklaracją i informacją o dziedziczeniu
- ☐ wyniki walidacji HTML i CSS

Pytania kontrolne

1. Oblicz specyficzność selektorów: `#menu a`, `.ramka p:nth-child(2)`, `ul li a:hover`, `a[href^="https"]`. Który jest najsilniejszy?
2. Dlaczego nie należy nadużywać `!important` i stylów inline?
3. Czym różni się selektor `header p` od `header > p`?
4. Które właściwości dziedziczą się z rodzica (podaj dwie), a które nie (podaj dwie)?

Kryteria oceny

Kryterium	Punkty
Tabela przewidywań P1–P8 z poprawnie obliczoną specyficznością	4
Wyjaśnienie P6 (!important), P7 (inline) i P8 (nth-child liczy wszystkie dzieci rodzica, a p8 nie jest w .ramka)	2
Arkusz zewnętrzny dołączony, reguły S1–S7 poprawne	4
Reguły S8–S13: klasa z elementem, nth-child, atrybut, sąsiad, first-child	4
Praca z F12 (przekreślenia, dziedziczenie, wyłączanie reguł) i walidacja HTML/CSS bez błędów	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Model pudełkowy i typografia — karty informacyjne

Czas: 2 × 45 min

praca indywidualna

Cel: Stylowanie kart informacyjnych z wykorzystaniem modelu pudełkowego, jednostek rem i em, kolorów hsl, cieni i zaokrągleń oraz lokalnie dołączonego kroju pisma.

Po zajęciach potrafisz:

- oblicza rzeczywistą szerokość elementu dla box-sizing content-box i border-box
- mierzy marginesy, ramki i dopełnienie w panelu Computed (F12)
- dołącza krój z plików WOFF2 regułą @font-face
- stosuje jednostki rem/em oraz kolory w zapisie hsl i hsla
- dodaje zaokrąglenia, cienie i odstępy między literami

Sprzęt i materiały

- komputer z Windows 11
- folder `r1_10_start` z Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator CSS W3C** jigsaw.w3.org/css-validator

Przygotowanie przed zajęciami

- Powtórz z wykładu 1.08: części pudełka (content, padding, border, margin), box-sizing, jednostki rem i em, zapis hsl.

Bezpieczeństwo

- Kroje z Google Fonts dołączaj lokalnie (pobrane pliki), a nie z serwerów Google — dołączenie zdalne przekazuje adres IP odwiedzającego firmie trzeciej (problem z RODO).
- Sprawdzaj licencję kroju przed użyciem; krój Inter jest na licencji SIL OFL 1.1 — zostaw plik `0FL.txt` w folderze `fonts`.

Przebieg zadania

1 Obliczenia przed stylowaniem

Karta będzie miała: `width: 280px`, `padding: 20px`, `border: 3px`, `margin: 12px`.

1. Oblicz na kartce całkowitą szerokość, jaką karta zajmie na stronie (z marginesami i bez), dla `box-sizing: content-box` oraz dla `box-sizing: border-box`. Oblicz też szerokość obszaru treści w obu przypadkach. Wyniki wpisz do tabeli.

2 Krój pisma i typografia

Utwórz `css/karty.css` (plik jest już dołączony w `index.html`). Reguły numeruj komentarzami K1–K5.

```
@font-face {
  font-family: "Inter";
  src: url("../fonts/Inter-Regular.woff2") format("woff2");
  font-weight: 400;
  font-display: swap;
}
```

1. K1. Zdefiniuj dwie reguły `@font-face` dla rodziny „Inter”: Regular (`font-weight: 400`) i Bold (700). Zwróć uwagę na ścieżkę `../` — arkusz leży w folderze `css`.
2. K2. `body`: krój Inter z zapasowymi `Arial, sans-serif`, `font-size: 1rem`, `line-height: 1.5`, tekst `hsl(220, 30%, 20%)`, tło `hsl(210, 40%, 96%)`, `margin: 0`, `padding: 1.5rem`. `h1`: `2.5rem`, `letter-spacing: -0.02em`. Akapit `.lead`: `1.25rem`, kolor `hsl(220, 15%, 40%)`.
3. Sprawdź w F12 → Network (filtr Font), że pobrano oba pliki WOFF2 ze statusem 200, a w Elements → Computed → Rendered Fonts (dół panelu), że tekst jest rysowany krojem Inter.

3 Karta — model pudełkowy

1. K3. `.karta`: `box-sizing: content-box`, wymiary z kroku 1, ramka 3px ciągła w kolorze `hsl(210, 30%, 80%)`, białe tło, `display: inline-block`, `vertical-align: top`, `border-radius: 12px`, cień `0 4px 12px` w kolorze `hsla(220, 40%, 20%, 0.15)`.
2. Nagłówek karty: `1.25rem`, marginesy `0 0 0.5rem`. Akapit `.dane`: `0.875rem`, szary kolor, górna ramka 1px przerywana, `padding-top: 0.5rem`.
3. K4. Etykieta `.poziom`: `display: inline-block`, dopełnienie `0.125rem 0.75rem`, `border-radius: 999px`, `0.75rem`, pogrubienie, wielkie litery, `letter-spacing: 0.08em`, biały tekst. Kolory tła: `.latwy` `hsl(140, 55%, 32%)`, `.sredni` `hsl(35, 90%, 32%)`, `.trudny` `hsl(0, 65%, 42%)`.
4. K5. `.wyzniona`: `box-sizing: border-box`, `width: 280px`, ramka 3px `hsl(35, 90%, 50%)`, tło `hsl(45, 100%, 95%)`.

4 Pomiary w F12

Panel Computed w zakładce Elements pokazuje schemat pudełka z wartościami margin, border, padding i content.

1. Zbadaj pierwszą kartę i kartę wyróżnioną. Odczytaj ze schematu wymiary content, padding, border, margin i porównaj z obliczeniami z kroku 1. Zrób zrzuty schematów.
2. Najedź w drzewie DOM na kartę — przeglądarka podświetli obszary pudełka kolorami. Opisz, który kolor oznacza margines, a który dopełnienie.
3. Zmień w F12 kolor `.sredni` na `hsl(35, 90%, 50%)` i sprawdź kontrast w próbniku kolorów (Contrast ratio). Czy biały tekst nadal spełnia wymóg 4,5:1? Przywróć wartość.
4. Sprawdź `karty.css` walidatorem CSS i zrób zrzut całej strony.

W sprawozdaniu umieść

- ☐ tabela obliczeń i pomiarów dla content-box i border-box
- ☐ zrzuty schematu pudełka (Computed) dla zwykłej i wyróżnionej karty
- ☐ zrzut zakładki Network z pobranymi plikami WOFF2
- ☐ wynik testu kontrastu etykiety i zrzut strony
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego w wielu projektach ustawia się `box-sizing: border-box` dla wszystkich elementów?

2. Czym różni się jednostka `rem` od `em`? Ile pikseli ma `0.875rem` przy domyślnym rozmiarze czcionki?
3. Co oznaczają trzy liczby w `hsl(35, 90%, 32%)`? Jak w tym zapisie uzyskać jaśniejszy odcień tego samego koloru?
4. Po co w `font-family` podaje się kroje zapasowe i co robi `font-display: swap`?

Kryteria oceny

Kryterium	Punkty
Poprawne obliczenia szerokości dla obu wartości box-sizing	3
K1-K2: @font-face dla 2 grubości z poprawną ścieżką, typografia w rem, kolory hsl	4
K3: karta z wymaganymi wymiarami, ramką, zaokrągleniem i cieniem hsla	4
K4-K5: etykiety poziomemu i karta wyróżniona z border-box	3
Pomiary w F12 zgodne z obliczeniami, test kontrastu, walidacja CSS bez błędów	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Układ strony float — styl INF.03

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie klasycznego układu egzaminacyjnego (baner, trzy panele obok siebie, stopka) z użyciem float i szerokości w procentach według podanych wymagań kolorów i wymiarów.

Po zajęciach potrafisz:

- buduje układ kolumnowy z elementów pływających (float) o szerokościach w %
- kończy opływanie stopką z clear: both
- wyjaśnia wpływ padding na szerokość panelu i stosuje box-sizing: border-box
- formatuje obraz, listę, odnośniki i tabelę zgodnie z wymaganiami w stylu arkusza egzaminacyjnego
- sprawdza zgodność strony z wymaganiami i walidatorem

Sprzęt i materiały

- komputer z Windows 11
- folder `r1_11_start` z Pliki_do_zadan

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (autor: Ritwick Dey) — instalacja z panelu Rozszerzenia (Ctrl+Shift+X) code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie — klawisz F12
- **Walidator W3C (Nu Html Checker)** — sprawdzanie przez wklejenie kodu (text input) lub wgranie pliku (file upload) validator.w3.org/nu
- **Walidator CSS W3C** jigsaw.w3.org/css-validator

Przygotowanie przed zajęciami

- Powtórz z wykładu 1.09 działanie float i clear oraz typowy układ: baner, panele, stopka.

Przebieg zadania

1 Struktura HTML

Utwórz folder `zad11`, skopiuj do niego grafiki i utwórz `index.html` oraz `styl.css`. Teksty kopiuj z `teksty.txt`. Wygląd docelowy pokazuje `wzor.png`.

```
header#baner    – obraz pszczoła.png + h1
nav#lewy        – h2 + lista odnośników
main#srodkowy   – h2#miody, akapit, tabela, h2#pszczoły, akapit
aside#prawy     – h2, obraz miod.png, akapit
footer#kontakt  – dwa akapity
```

1. Dokument: `lang="pl"`, UTF-8, tytuł karty „Pasieka Miodowy Zakątek”, arkusz `styl.css` dołączony w `head`.
2. Zbuduj bloki według schematu (identyfikatory dokładnie jak w schemacie). Obraz w banerze: `alt` „Pszczoła — logo pasieki”. Obraz w panelu prawym: `alt` z pliku tekstowego.
3. Tabela cennika: wiersz nagłówek w `thead` z komórkami `th`, cztery rodzaje miodu w `tbody`. Adres e-mail w stopce jako odnośnik `mailto:`, w drugim akapicie Twoje imię i nazwisko.

2 Styl — wymagania jak w arkuszu

Wymiary i kolory muszą być dokładnie takie, jak podano. Selektory dobierz samodzielnie.

1. Cała strona: krój Verdana (z zapasowymi), tło `#FFFBEA`, kolor tekstu `#3B2A00`, marginesy `body` 0.
2. Baner: tło `#F2B705`, wysokość 120px, tekst wyśrodkowany, krój Georgia (zapasowy szeryfowy). Obraz pływa do lewej z marginesem 10px. Nagłówek h1: rozmiar 36px, bez marginesów, wysokość linii 120px.
3. Panele lewy, środkowy, prawy: pływające do lewej, wysokość 600px, dopełnienie 10px, szerokości 25%, 50%, 25%; tła odpowiednio `#FFF4CC`, `#FFFFFF`, `#FFF4CC`. Panel prawy: tekst wyśrodkowany.
4. Najpierw napisz panele bez `box-sizing`. Zmniejszaj i powiększaj okno przeglądarki — co się dzieje z panelem prawym? Zrób zrzut, a potem dodaj panelom `box-sizing: border-box` i sprawdź ponownie.
5. Lista w panelu lewym: punktory kwadrat. Odnośniki: kolor `#8A5A00`, bez podkreślenia; po najechaniu kursorem tło `#F2B705` i kolor `#3B2A00`.
6. Obraz w panelu prawym: szerokość 90%, wysokość automatyczna, zaokrąglenie narożników 10px, cień 4px 4px 8px w kolorze gray.
7. Tabela: szerokość 80%, wyśrodkowana (`margin: 0 auto`), krawędzie komórek 1px ciągłe `#C79100` bez podwójnych linii, dopełnienie komórek 6px, tekst wyśrodkowany; komórki nagłówek z tłem `#F2B705`; wiersz `tbody` po najechaniu kursorem z tłem `#FFF4CC`.
8. Stopka: opływanie wyłączane z obu stron, tło `#3B2A00`, kolor tekstu biały, dopełnienie 15px, tekst wyrównany do prawej; odnośnik w stopce w kolorze `#F2B705`.

3 Sprawdzenie

1. Usuń na chwilę z CSS deklarację `clear: both` stopki. Opisz, co się stało, i przywróć ją.
2. W F12 → Elements zmierz szerokości paneli przy oknie zmaksymalizowanym (Computed lub podpowiedź po najechaniu na element). Czy wynoszą 25%, 50% i 25% szerokości okna?
3. Sprawdź walidatorem `index.html` i `styl.css`, porównaj stronę z `wzor.png` i zrób zrzut całego okna.

W sprawozdaniu umieść

- ☐ zrzut strony z panelami bez `box-sizing` (problem) i po poprawce
- ☐ opis efektu usunięcia `clear: both`

- ☐ zmierzone szerokości paneli i szerokość okna
- ☐ pliki `index.html` i `styl.css`, wyniki walidacji HTML i CSS
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego bez `box-sizing: border-box` panele o szerokościach 25% + 50% + 25% z dopełnieniem 10px nie mieszczą się w jednym rzędzie? Jak inaczej rozwiązać ten problem?
2. Do czego służy `clear: both` i co by się stało ze stopką bez tej deklaracji?
3. Dlaczego wysokość linii równa wysokości banera centruje jednowierszowy nagłówek w pionie? Kiedy ta metoda zawodzi?
4. Jak ten sam układ zbudujesz przy użyciu flexboxa (zajęcia z wykładu 1.10)? Podaj kluczowe deklaracje.

Kryteria oceny

Kryterium	Punkty
Struktura HTML: 5 bloków z wymaganymi id, tabela z thead/tbody, obrazy z alt, mailto	4
Baner i body zgodne z wymaganiami (kolory, wysokość, kroje, obraz pływający)	3
Panele: float, szerokości w %, wysokość, tła, padding z border-box — trzy panele w jednym rzędzie	4
Lista, odnośniki z :hover, obraz z zaokrągleniem i cieniem	3
Tabela i stopka zgodne z wymaganiami (clear, kolory, wyrównanie)	3
Opis testów (box-sizing, clear), walidacja HTML i CSS bez błędów	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Pozycjonowanie elementów

Czas: 2 × 45 min

praca indywidualna

Cel: Stosowanie właściwości position (relative, absolute, fixed, sticky), z-index i overflow do budowy typowych elementów strony: przyklejonego menu, przycisku powrotu na górę i etykiety na karcie produktu.

Po zajęciach potrafisz:

- przykleja menu do górnej krawędzi okna właściwością position: sticky
- umieszcza element absolutnie względem rodzica z position: relative
- ustala kolejność nakładających się warstw właściwością z-index i przewiduje wynik
- ogranicza wysokość bloku i włącza przewijanie właściwością overflow
- sprawdza w narzędziach F12 (Elements → Computed), skąd element bierze pozycję

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_12_start` z Pliki_do_zadan (index.html i style.css)

Oprogramowanie

- **Visual Studio Code** — z rozszerzeniem Live Server (podgląd odświeżany po zapisie)
code.visualstudio.com
- **Live Server (rozszerzenie VS Code)** — instalacja: Extensions (Ctrl+Shift+X) → wyszukaj „Live Server”
marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer
- **Validator CSS W3C** — zakładka „By file upload”
jigsaw.w3.org/css-validator

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.09 różnicę między position: relative, absolute, fixed i sticky.
- Skopiuj folder `r1_12_start` do swojego folderu `nazwisko_imie` i zmień jego nazwę na `zad12`.

Bezpieczeństwo

- Pracuj na kopii plików w swoim folderze — oryginał w Pliki_do_zadan zostaw bez zmian.
- Nie zmieniaj gotowych reguł na początku arkusza; dopisuj swoje na końcu, w oznaczonych sekcjach.

Przebieg zadania

1 Rozpoznanie strony

Każdą zmianę zapisuj (Ctrl+S) i sprawdzaj w przeglądarce. Wszystkie reguły dopisuj pod komentarzem `Zadanie 12: Twoje reguły` w pliku `style.css`.

1. Otwórz folder `zad12` w VS Code (File → Open Folder) i uruchom `index.html` przez Live Server (prawy przycisk → Open with Live Server).
2. Przewiń stronę. Zanotuj, co dzieje się z menu i z czerwonym przyciskiem „↑” (gdzie jest teraz?) oraz gdzie wyświetla się etykieta „Nowość” i trzy kwadraty warstw.

2 Menu przyklejone (sticky)

Element sticky zachowuje się jak zwykły, dopóki nie dojdzie do progu (np. `top: 0`) — wtedy „przykleja się” do okna.

1. P1. Ustaw dla `.menu` pozycję `sticky` i próg `top: 0`. Przewiń stronę i sprawdź, że menu zostaje u góry, a baner odjeżdża.
2. P2. Kliknij w menu „Warstwy”. Zauważ, że nagłówek sekcji chowa się pod menu. Popraw to regułą `scroll-margin-top` dla elementów `section` (wartość nieco większa niż wysokość menu — odczytaj ją w F12).
3. P3. Nadaj menu `z-index: 100`, aby po zakończeniu kolejnych kroków żaden element strony nie przykrywał menu podczas przewijania.

3 Etykieta „Nowość” (absolute w relative)

Element absolute jest pozycjonowany względem najbliższego przodka, który ma position inne niż static. Jeśli takiego nie ma — względem całego dokumentu.

1. P4. Ustaw `.nowosc` pozycję `absolute` z `top: -10px` i `right: -10px`, ale **bez** zmiany kart. Zrób zrzut — gdzie pojawiły się etykiety i dlaczego?
2. P5. Dodaj `.karta` pozycję `relative`. Etykiety powinny przesunąć się do prawego górnego narożnika swoich kart, lekko poza krawędź.
3. P6. Obróć etykietę o 8 stopni (`transform: rotate(8deg)`). W F12 zaznacz etykietę i w zakładce Computed odczytaj wartości `position`, `top` i `right`.

4 Warstwy i z-index

Trzy kwadraty w `.scena` mają się nakładać schodkowo. Kwadrat 1: `top 20px, left 20px`; kwadrat 2: `top 50px, left 100px`; kwadrat 3: `top 80px, left 180px`.

1. P7. Ustaw `.scena` pozycję `relative`, a `.warstwa` — `absolute`, i rozmieść kwadraty według współrzędnych podanych wyżej. Który kwadrat jest na wierzchu bez `z-index`? Zapisz regułę, która o tym decyduje.
2. P8. Zanim zapiszesz plik, przewidź wynik: kwadrat 1 dostaje `z-index: 1`, kwadrat 2 — `z-index: 3`, kwadrat 3 — `z-index: 2`. Zapisz przewidywaną kolejność od góry, potem sprawdź w przeglądarce.
3. P9. Dodaj regułę `.warstwa:hover`, która wyciąga najechany kwadrat na wierzch (`z-index: 10`) i dodaje mu ciemny obrys (`outline`).

5 Overflow i przycisk „do góry”

1. P10. Ogranicz wysokość bloku `.regulamin` do 180px i włącz pionowe przewijanie (`overflow-y: auto`). Sprawdź także wartości `hidden` i `visible` — opisz różnicę w sprawozdaniu, a w pliku zostaw `auto`.
2. P11. Ustaw przycisk `.do-gory` na stałe w prawym dolnym rogu okna: `position: fixed`, 24px od prawej i od dołu. Kliknij go po przewinięciu strony — powinien przenieść Cię na górę.
3. P12. Nadaj przyciskowi `z-index` większy niż menu i sprawdź w trybie urządzeń (Ctrl+Shift+M w F12, np. szerokość 375px), że przycisk nie zasłania treści w sposób uniemożliwiający czytanie.
4. P13. Sprawdź `style.css` walidatorem CSS W3C (oczekiwany wynik: brak błędów). Zrób zrzuty: strona przewinięta do sekcji Kontakt (widać menu u góry i przycisk w rogu), karty z etykietami, warstwy po P8.

W sprawozdaniu umieść

- ☐ tabela P1–P13 z regułami CSS i opisem wyniku
- ☐ zrzuty: etykiety po P4 i po P5, warstwy po P8, przewinięta strona po P11
- ☐ przewidywana i rzeczywista kolejność warstw w P8
- ☐ porównanie `overflow: visible, hidden, auto`
- ☐ folder `zad12` z plikami `index.html` i `style.css`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się `position: fixed` od `position: sticky`? Podaj po jednym zastosowaniu.
2. Względem czego pozycjonowany jest element `absolute`, gdy żaden przodek nie ma ustawionego `position`?
3. Dlaczego `z-index` nie działa dla elementu z `position: static`?
4. Co się stanie z treścią bloku o ograniczonej wysokości przy `overflow: hidden` i dlaczego może to być problem dla użytkownika?

Kryteria oceny

Kryterium	Punkty
Menu sticky, scroll-margin-top i z-index menu (P1–P3)	4
Etykieta „Nowość” w narożniku każdej karty, wyjaśnienie wyniku P4 (P4–P6)	4
Warstwy rozmieszczone wg współrzędnych, poprawne przewidywanie P8, efekt :hover (P7–P9)	4
Overflow regulaminu i przycisk fixed w prawym dolnym rogu (P10–P12)	4
Walidacja CSS bez błędów, zrzuty i sprawozdanie	2
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Flexbox w praktyce

Czas: 2 × 45 min

praca indywidualna

Cel: Budowanie typowych fragmentów układu strony Flexboxem: poziomego paska nawigacji, wyśrodkowanego bloku, sekcji kart z zawijaniem i stopki przyklejonej do dołu okna.

Po zajęciach potrafisz:

- tworzy kontener flex i ustawia elementy na osi głównej (justify-content) i poprzecznej (align-items)
- buduje poziome menu z odstępami gap i rozsuwa elementy nagłówka
- wyśrodkowuje blok w pionie i poziomie
- układa karty z zawijaniem (flex-wrap) i steruje ich szerokością skrótem flex
- przykleja stopkę do dołu okna przy krótkiej treści strony

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_13_start` z Pliki_do_zadan (index.html i style.css)

Oprogramowanie

- **Visual Studio Code z rozszerzeniem Live Server** — Live Server: Extensions (Ctrl+Shift+X) → „Live Server” code.visualstudio.com
- **Flexbox Froggy (opcjonalnie)** — gra w przeglądarce, wersja polska flexboxfroggy.com/#pl
- **Narzędzia F12 przeglądarki** — plakietka „flex” przy elemencie w zakładce Elements włącza nakładkę Flexbox developer.chrome.com/docs/devtools/css/flexbox

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.10 pojęcia: kontener, elementy flex, oś główna i poprzeczna.
- Skopiuj folder `r1_13_start` do folderu `nazwisko_imie` jako `zad13`.

Bezpieczeństwo

- Pracuj na kopii plików; nie zmieniaj pliku index.html — całe zadanie wykonujesz w CSS.

Przebieg zadania

1 Rozgrzewka (opcjonalnie, maks. 10 min)

Jeśli skończysz zadanie wcześniej, wróć do gry i przejdź dalsze poziomy.

1. Otwórz Flexbox Froggy i przejdź poziomy 1–10. Zanotuj numer ostatniego ukończonego poziomu.
2. Uruchom `zad13/index.html` przez Live Server. Zrób zrzut strony przed zmianami — porównasz go z wynikiem końcowym.

2 Pasek nawigacji

Reguły dopisuj w `style.css` pod komentarzem `Zadanie 13`. Po każdej zmianie zaznacz kontener w F12 i kliknij plakietkę `flex`, aby zobaczyć nakładkę.

- F1. Zamień `.naglowek` w kontener flex. Logo, menu i przycisk mają stać w jednym rzędzie i być wyśrodkowane w pionie (`align-items`).
- F2. Rozsuń elementy nagłówka: logo przy lewej krawędzi, przycisk „Zapisz się” przy prawej (`justify-content: space-between`).
- F3. Ustaw odnośniki listy `.menu` w poziomie z odstępem 8px (`display: flex` i `gap`), bez użycia `float` i `inline-block`.
- F4. Przesuń menu do prawej, tuż obok przycisku: nadaj elementowi `nav` `margin-left: auto`, a nagłówkowi `gap: 16px`. Wyjaśnij w sprawozdaniu, dlaczego automatyczny margines „wypycha” element.

3 Centrowanie

- F5. Wyśrodkuj biały blok `.hero-tekst` w sekcji `.hero` (wysokość 260px) jednocześnie w poziomie i w pionie — tylko właściwościami kontenera flex.
- F6. Zmień na chwilę `flex-direction` sekcji hero na `column` i sprawdź, czy blok nadal jest wyśrodkowany. Zapisz w sprawozdaniu, która właściwość odpowiada teraz za pion, a która za poziom. Przywróć `row`.

4 Karty z zawijaniem

Skrót `flex: 1 1 260px` oznacza: element może rosnąć (grow 1), może się kurczyć (shrink 1), a jego szerokość bazowa to 260px.

- F7. Ułóż karty w kontenerze `.karty` w rzędzie z odstępem 16px. Sprawdź, co się dzieje przy szerokości okna 600px (tryb urządzeń Ctrl+Shift+M). Następnie włącz zawijanie (`flex-wrap: wrap`).
- F8. Nadaj kartom `flex: 1 1 260px`. Zmieniaj szerokość okna od 1280px do 375px i zapisz, przy jakich szerokościach w rzędzie są 4, 3, 2 i 1 karta.
- F9. Wyrównaj dolną linię kart: każda `.karta` ma być kontenerem flex w kolumnie, a akapit `.info` ma się trzymać dołu karty (`margin-top: auto`). Wszystkie linie z liczbą graczy w jednym rzędzie mają być na tej samej wysokości.
- F10. Bez zmiany HTML przenieś ostatnią kartę („Labirynt Smoka”) na początek właściwością `order` i wyróżnij ją pomarańczowym obramowaniem 2px (`#f59e0b`).

5 Stopka przyklejona do dołu

Przy krótkiej treści stopka „wisi” w połowie okna. Aby to zobaczyć, ustaw w trybie urządzeń wysokość okna 1400px (np. 1280 × 1400).

- F11. Zamień `body` w kontener flex w kolumnie o minimalnej wysokości `100vh` i pozwól `main` zająć całe wolne miejsce (`flex: 1`). Stopka ma dotykać dolnej krawędzi okna.
- F12. Ułóż dwa akapity stopki w jednym rzędzie: tekst o klubie po lewej, opiekun po prawej; na wąskim ekranie akapity mają się zawinąć jeden pod drugi.
- F13. Zrób zrzuty wyniku przy 1280 × 1400 i 375 × 800. Sprawdź `style.css` walidatorem CSS W3C (<https://jigsaw.w3.org/css-validator/>).

W sprawozdaniu umieść

- ☐ tabela F1–F13 z regułami CSS i opisem wyniku
- ☐ zrzuty: strona przed zmianami, po zmianach przy 1280 × 1400 i przy 375 × 800, nakładka Flexbox z F12 na kontenerze `.karty`
- ☐ tabela szerokości okna i liczby kart w rzędzie (F8)

- ☐ wyjaśnienie działania margin-left: auto (F4) i zamiany osi (F6)
- ☐ folder `zad13` z plikami
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Która właściwość wyrównuje elementy na osi głównej, a która na poprzecznej? Co się zmienia po ustawieniu flex-direction: column?
2. Co oznaczają trzy wartości w skrócie flex: 1 1 260px?
3. Dlaczego właściwość order nie powinna zmieniać kolejności istotnej dla treści (pomyśl o czytniku ekranu i klawiszu Tab)?
4. Jak działa stopka przyklejona do dołu: która reguła powoduje, że main „rozpycha się” na wolne miejsce?

Kryteria oceny

Kryterium	Punkty
Nagłówek: logo po lewej, menu (4 odnośniki w rzędzie) i przycisk po prawej, wyśrodkowane w pionie (F1–F4)	4
Blok hero wyśrodkowany w pionie i poziomie, opis zamiany osi (F5–F6)	3
Karty zawijają się, flex 1 1 260px, tabela szerokości, linia .info wyrównana, Labirynt Smoka pierwszy z obramowaniem (F7–F10)	5
Stopka przy dolnej krawędzi okna 1280 × 1400, dwa akapity w rzędzie (F11–F12)	3
Zrzuty, walidacja CSS bez błędów, sprawozdanie	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

CSS Grid w praktyce

Czas: 2 × 45 min

praca indywidualna

Cel: Budowanie układu strony siatką CSS Grid z nazwanymi obszarami oraz galerii, która sama dopasowuje liczbę kolumn do szerokości okna; porównanie z rozwiązaniem we Flexboxie.

Po zajęciach potrafisz:

- definiuje siatkę kolumn i wierszy z jednostką `fr`, `repeat` i `minmax`
- nazywa obszary (`grid-area`) i rozmieszcza je właściwością `grid-template-areas`
- buduje galerię `auto-fit/minmax` i rozróżnia `auto-fit` od `auto-fill`
- rozciąga element na kilka komórek (`span`)
- dobiera Grid albo Flexbox do problemu i uzasadnia wybór

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_14_start` z Pliki_do_zadan (`index.html`, `style.css`, folder `img`)

Oprogramowanie

- **Visual Studio Code z rozszerzeniem Live Server** — podgląd odświeżany po zapisie code.visualstudio.com
- **Grid Garden (opcjonalnie)** — gra w przeglądarce, wersja polska cssgridgarden.com/#pl
- **Narzędzia F12 — nakładka Grid** — plakietka „grid” w Elements; w zakładce Layout włącz „Show area names” developer.chrome.com/docs/devtools/css/grid

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 1.11: `fr`, `repeat()`, `minmax()`, `grid-template-areas`.
- Skopiuj folder `r1_14_start` do folderu `nazwisko_imie` jako `zad14`. **Zachowaj wynik** — w zadaniu 15 przerobisz tę stronę na responsywną.

Bezpieczeństwo

- Pracuj na kopii plików; nie zmieniaj pliku `index.html` (wyjątek: plik `galeria-flex.html` w G9).
- Zmiany wprowadzane w F12 (np. usuwanie elementów) działają tylko w podglądzie — nie zapisują się w pliku.

Przebieg zadania

1 Rozgrzewka i rozpoznanie (maks. 10 min)

1. Opcjonalnie przejdź poziomy 1–8 gry Grid Garden; zanotuj ostatni ukończony poziom.
2. Uruchom `zad14/index.html` przez Live Server i zrób zrzut strony przed zmianami. Zwróć uwagę, że wszystkie bloki stoją jeden pod drugim, a ilustracje mają pełną szerokość.

2 Układ strony z obszarami

Docelowy układ (szkic, kontener `.strona`, maks. 1200px):

nagłówek		
menu		
tresc (1fr)	boczny 280px	
stopka		

1. G1. Przypisz nazwy obszarów: `.nagłówek` → `nagłówek`, `.menu` → `menu`, `.tresc` → `tresc`, `.boczny` → `boczny`, `.stopka` → `stopka` (właściwość `grid-area`).
2. G2. Zamień `.strona` w siatkę o kolumnach `1fr 280px` i rozmieść obszary według szkicu właściwością `grid-template-areas` (każdy wiersz w osobnym cudzysłowie).
3. G3. Ustaw wiersze `auto auto 1fr auto` i minimalną wysokość `.strona` równą `100vh` — stopka ma być na dole okna. Ułóż odnośniki menu w poziomie (Flexbox z zadania 13).
4. G4. W F12 zaznacz `.strona`, włącz nakładkę Grid z nazwami obszarów i zrób zrzut. Odczytaj szerokość kolumny `tresc` przy oknie 1280px.

3 Galeria auto-fit / minmax

`repeat(auto-fit, minmax(180px, 1fr))` = tyle kolumn, ile się zmieści, każda co najmniej 180px, a wolne miejsce dzielone równo.

1. G5. Zbuduj galerię `.galeria` jako siatkę z kolumnami `repeat(auto-fit, minmax(180px, 1fr))` i odstępem 12px. Zmieniaj szerokość okna i zapisz liczbę kolumn przy 1280px, 1000px i 800px.
2. G6. Porównaj auto-fit i auto-fill: w F12 usuń z podglądu 6 ostatnich elementów `figure` (prawy przycisk → Delete element), a potem zmieniaj w zakładce Styles `auto-fit` na `auto-fill`. Opisz różnicę. Odśwież stronę, aby przywrócić figury.
3. G7. Wyróżnij pierwszą ilustrację: ma zajmować 2 kolumny i 2 wiersze (`grid-column: span 2; grid-row: span 2`).
4. G8. Sprawdź, czy przy 1280px galeria nie ma pustych dziur. Jeśli są, wypróbuj `grid-auto-flow: dense` i opisz, co się zmieniło (w pliku zostaw wersję, która lepiej wygląda).

4 Porównanie z Flexboxem

Ta sama galeria zbudowana Flexboxem — w osobnych plikach, aby nie psuć wersji Grid.

1. G9. Skopiuj `index.html` jako `galeria-flex.html` i zmień w nim tylko odnośnik do arkusza na `style-flex.css`. W nowym pliku `style-flex.css` w pierwszej linii zaimportuj główny arkusz: `@import url("style.css");`, a poniżej nadpisz galerię: kontener flex z zawijaniem i odstępem 12px, figury `flex: 1 1 180px`, pierwsza figura z podstawą 372px.
2. G10. Porównaj obie wersje przy szerokościach 1280px i 1000px (zrzuty obok siebie). Zwróć uwagę na ostatni rząd i na wyróżnioną ilustrację. Zapisz w sprawozdaniu, kiedy lepszy jest Grid, a kiedy Flexbox.
3. G11. Sprawdź oba arkusze walidatorem CSS W3C. Zrób zrzuty końcowe strony przy 1280px.

W sprawozdaniu umieść

- ☐ tabela G1–G11 z regułami CSS i opisem wyniku
- ☐ zrzut nakładki Grid z nazwami obszarów (G4)
- ☐ liczba kolumn galerii przy 1280px, 1000px i 800px (G5)

- ☐ opis różnicy auto-fit i auto-fill (G6)
- ☐ rzuty galerii Grid i Flexbox obok siebie z wnioskami (G10)
- ☐ folder `zad14` z plikami `index.html`, `style.css`, `galeria-flex.html`, `style-flex.css`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Co oznacza jednostka `fr` i czym różni się kolumna `1fr` od kolumny `100%`?
2. Kiedy `auto-fit` i `auto-fill` dają ten sam wynik, a kiedy różny?
3. Dlaczego Grid nazywamy układem dwuwymiarowym, a Flexbox jednowymiarowym? Podaj przykład zadania dla każdego.
4. Jak zmienić układ z obszarami, aby panel boczny był po lewej stronie? Ile linii CSS trzeba zmienić?

Kryteria oceny

Kryterium	Punkty
Obszary nazwane i rozmieszczone wg szkicu, kolumny <code>1fr</code> 280px, stopka na dole okna, menu w poziomie (G1–G3)	5
Zrzut nakładki Grid i odczyt szerokości (G4)	1
Galeria <code>auto-fit</code> / <code>minmax</code> z odstępem 12px, liczby kolumn, opis <code>auto-fit</code> vs <code>auto-fill</code> (G5–G6)	4
Pierwsza ilustracja na 2×2 komórki, analiza dziur (G7–G8)	2
Wersja Flexbox w osobnych plikach i porównanie z wnioskami (G9–G10)	4
Walidacja CSS bez błędów, kompletne sprawozdanie	1
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik `.docx` o tym samym numerze).

Strona responsywna (mobile first)

Czas: 2 × 45 min

praca indywidualna

Cel: Przebudowa gotowej strony na responsywną w podejściu mobile first: układ 1 → 2 → 3 kolumny w punktach przerwania, menu zwijane bez JavaScriptu, obrazy responsywne (srcset, sizes, picture) i test w trybie urządzeń.

Po zajęciach potrafisz:

- pisze style bazowe dla telefonu i rozszerza je media queries z min-width
- dobiera punkty przerwania do treści i sprawdza je w trybie urządzeń F12
- buduje menu zwijane bez JavaScriptu (checkbox i label)
- wstawia obrazy responsywne atrybutami srcset i sizes oraz element picture
- stosuje płynną typografię funkcją clamp()

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome lub Edge (tryb urządzeń, zakładka Network)
- wynik zadania 14 (`zad14`) albo folder `r1_15_start` z Pliki_do_zadan
- opcjonalnie własny telefon w tej samej sieci (podgląd przez Live Server)

Oprogramowanie

- **Visual Studio Code z rozszerzeniem Live Server** — Live Server udostępnia stronę pod adresem IP komputera — można ją otworzyć na telefonie, jeśli pozwala na to sieć szkolna
code.visualstudio.com
- **Tryb urządzeń w F12** — Ctrl+Shift+M; pasek „Responsive” pozwala wpisać szerokość
developer.chrome.com/docs/devtools/device-mode
- **Validator HTML W3C** — zakładka „Validate by File Upload”
validator.w3.org

Przygotowanie przed zajęciami

- Skopiuj folder `zad14` jako `zad15` (albo `r1_15_start` jako `zad15`) i dograj do jego folderu `img` pliki WebP z `r1_15_start/img`.
- Przypomnij sobie z wykładu 1.12: meta viewport, media queries z min-width, srcset i sizes.

Bezpieczeństwo

- Pracuj na kopii — wersja z zadania 14 ma zostać bez zmian (porównasz je w sprawozdaniu).
- Jeśli otwierasz stronę na telefonie, rób to tylko w sieci szkolnej i wyłącz Live Server po zajęciach.

Przebieg zadania

1 Analiza wersji z zadania 14

1. Otwórz `zad15/index.html` w trybie urządzeń (Ctrl+Shift+M) przy szerokości 375px. Zrób zrzut i zapisz co najmniej 3 problemy (np. panel boczny, menu, rozmiar tekstu).
2. Sprawdź, czy w `head` jest `<meta name="viewport" content="width=device-width, initial-scale=1.0">`. Usuń go na chwilę, odśwież i opisz różnicę, potem przywróć.

2 Style bazowe dla telefonu

Mobile first: najpierw style bez media queries (telefon), potem dopisujesz tylko zmiany dla większych ekranów. Punkty przerwania: **600px** (tablet) i **900px** (komputer).

1. R1. Przebuduj `style.css`: obecny układ siatki z zadania 14 przenieś do `@media (min-width: 900px)`, a w stylach bazowych ustaw jedną kolumnę z obszarami w kolejności: nagłówek, menu, treść, boczny, stopka. Zmniejsz poziome odstępy (padding) do 16px. Dodaj regułę `img { max-width: 100%; height: auto; }`.
2. R2. W `nav.menu` przed listą `ul` dodaj pole `<input type="checkbox" id="przelacz" class="przelacz">` i etykietę `<label for="przelacz" class="hamburger">☰ Menu</label>`. Na telefonie lista jest ukryta (`display: none`), a po zaznaczeniu pola wyświetla się pionowo — selektor `.przelacz:checked ~ ul`.
3. R3. Ukryj samo pole wizualnie (np. `position: absolute; opacity: 0`), ale nie przez `display: none`, aby dało się je obsłużyć klawiaturą. Dodaj widoczny obrys etykiety, gdy pole ma fokus (`.przelacz:focus-visible + .hamburger`). Sprawdź obsługę klawiszami Tab i Spacja.
4. R4. Galeria na telefonie: jedna kolumna, odstęp 12px.

3 Punkty przerwania

1. R5. `@media (min-width: 600px)`: galeria w 2 kolumnach (pierwsza ilustracja na 2 kolumny), menu zawsze widoczne w poziomie, etykieta `☰` ukryta.
2. R6. `@media (min-width: 900px)`: układ z panelem bocznym 280px po prawej (jak w zadaniu 14), galeria w 3 kolumnach, pierwsza ilustracja 2 × 2.
3. R7. Rozmiar nagłówka `h1` ustaw płynnie: `font-size: clamp(1.5rem, 1rem + 2.5vw, 2.5rem)`. Zapisz, jaki rozmiar w px pokazuje zakładka Computed przy 375px i przy 1280px.

4 Obrazy responsywne

Pliki: `eksperymentN-400.webp` (400 × 300) i `eksperymentN-800.webp` (800 × 600), `baner-waski.webp` (800 × 600) i `baner-szeroki.webp` (1600 × 400).

```

```

1. R8. Zamień ilustracje SVG w galerii na pliki WebP z atrybutami `srcset` i `sizes` według wzoru. Zachowaj teksty `alt`. Dla pierwszej (wyróżnionej) ilustracji dobierz własną wartość `sizes`, bo jest szersza.
2. R9. Na początku `main` dodaj baner w elemencie `picture`: na ekranach od 600px szeroki kadr (`source media=...`), na telefonie wąski (`img`). Wpisz atrybuty `alt`, `width` i `height`.
3. R10. W zakładce Network (filtr Img, zaznaczone „Disable cache”) sprawdź, które pliki pobiera przeglądarka przy 375px i 1280px (oraz przy przełączniku DPR 2 w trybie urządzeń). Zapisz wyniki w tabeli.

5 Testy i dokumentacja

1. R11. Przetestuj stronę przy 320px, 375px, 600px, 768px, 900px i 1280px: brak poziomego paska przewijania, tekst czytelny bez powiększania, menu działa. Zrób zrzuty przy 375px (menu rozwinięte), 768px i 1280px.
2. R12. Sprawdź `index.html` walidatorem HTML i `style.css` walidatorem CSS W3C. Jeśli możesz, otwórz stronę na telefonie i zrób zrzut ekranu.

W sprawozdaniu umieść

- ☐ lista problemów wersji z zadania 14 na telefonie i opis skutku usunięcia meta viewport
- ☐ tabela R1–R12 z fragmentami kodu
- ☐ zrzuty przy 375px (menu zwinięte i rozwinięte), 768px i 1280px
- ☐ tabela pobranych obrazów z zakładki Network
- ☐ rozmiar h1 z clamp() przy 375px i 1280px
- ☐ folder `zad15`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Na czym polega podejście mobile first i dlaczego używa się w nim media queries z min-width, a nie max-width?
2. Co mówi przeglądarce atrybut sizes i dlaczego bez niego srcset z deskryptorami w działa gorzej?
3. Czym różni się element picture z source media od samego srcset? Kiedy użyjesz picture?
4. Dlaczego pole checkbox w menu nie powinno być ukryte przez display: none?
5. Po co obrazom atrybuty width i height, skoro CSS i tak zmienia ich rozmiar?

Kryteria oceny

Kryterium	Punkty
Style bazowe: jedna kolumna, kolejność obszarów, brak poziomego przewijania przy 320px (R1, R4)	3
Menu zwijane bez JS, dostępne z klawiatury, ukryte ☰ od 600px (R2–R3, R5)	4
Punkty przerwania 600px i 900px: galeria 1 → 2 → 3 kolumny, panel boczny od 900px, clamp() (R5–R7)	4
srcset/sizes w galerii, picture z dwoma kadrami, tabela z Network (R8–R10)	4
Testy przy 6 szerokościach, zrzuty, walidacja HTML i CSS (R11–R12)	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Efekty, animacje i tryb ciemny

Czas: 2 × 45 min

praca indywidualna

Cel: Ożywianie strony efektami CSS (stany `:hover`, `:focus-visible`, `:active`, przejścia, transformacje, animacje `@keyframes`, pseudoelementy) z poszanowaniem preferencji użytkownika: trybu ciemnego i ograniczenia ruchu.

Po zajęciach potrafisz:

- porządkuje kolory strony w zmiennych CSS (custom properties)
- dodaje stany przycisków z płynnym przejściem `transition` i wyraźnym fokusem klawiatury
- buduje kartę obracaną transformacjami 3D
- tworzy animację `@keyframes` (loader) i wstawia treść pseudoelementem `::before`
- dostosowuje stronę do `prefers-color-scheme` i `prefers-reduced-motion` i testuje to w F12

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome lub Edge
- folder `r1_16_start` z Pliki_do_zadan (index.html i style.css)

Oprogramowanie

- **Visual Studio Code z rozszerzeniem Live Server** — podgląd odświeżany po zapisie code.visualstudio.com
- **F12 → panel Rendering** — Ctrl+Shift+P → „Show Rendering”; sekcje „Emulate CSS media feature `prefers-color-scheme`” i „`prefers-reduced-motion`” developer.chrome.com/docs/devtools/rendering/emulate-css
- **F12 → panel Animations (opcjonalnie)** — spowalnianie animacji do 25% i 10% developer.chrome.com/docs/devtools/css/animations

Przygotowanie przed zajęciami

- Skopiuj folder `r1_16_start` do folderu `nazwisko_imie` jako `zad16`.
- Przypomnij sobie z wykładu 1.13: `transition`, `transform`, `@keyframes`, zmienne CSS.

Bezpieczeństwo

- Animacje migające szybciej niż 3 razy na sekundę mogą wywołać atak u osób z padaczką świetłoczułą — nie twórz błysków ani szybkiego migania kolorów.
- Pracuj na kopii plików w swoim folderze.

Przebieg zadania

1 Zmienne CSS

Zmienne deklarujesz w `:root`, a używasz przez `var(--nazwa)`. Dzięki nim tryb ciemny będzie wymagał zmiany tylko kilku wartości.

```
:root {
  --tlo: #fdf8f3;
  --tekst: #3b2414;
  --akcent: #6f4e37;
  /* ... dopisz pozostałe kolory ... */
  --czas: 0.25s;
}
body { background-color: var(--tlo); }
```

1. E1. Wypisz wszystkie kolory użyte w `style.css` (jest ich 6). Zadeklaruj je jako zmienne w `:root` (nazwy po polsku, np. `--tlo`, `--tekst`, `--akcent`, `--akcent-tekst`, `--jasny`, `--ramka`) i zamień **każde** wystąpienie koloru w arkuszu na `var(...)`. Dodaj zmienną `--czas: 0.25s` dla przejść. Sprawdź, czy strona wygląda tak samo jak przed zmianą.

2 Przyciski: stany i przejścia

1. E2. Po najechaniu (`:hover`) przycisk ma zmienić tło na kolor akcentu, tekst na jasny, unieść się o 3px (`transform: translateY(-3px)`) i dostać cień. Zmiany mają trwać `var(--czas)` — wymień w `transition` konkretne właściwości zamiast `all`.
2. E3. Po wciśnięciu (`:active`) przycisk lekko się zmniejsza (`scale(0.97)`) i traci cień.
3. E4. Dla `:focus-visible` dodaj przerywany obrys 3px w kolorze akcentu z odsunięciem 4px. Sprawdź, przechodząc klawiszem Tab, że obrys pojawia się tylko przy obsłudze klawiaturą, a nie po kliknięciu myszą.

3 Karta obracana

Budowa: `.karta` (scena z perspektywą) → `.karta-wnetrze` (to się obraca) → `.przod` i `.tyl` (nałożone na siebie).

1. E5. Ustaw `.karta` perspektywę 800px. `.karta-wnetrze` ma mieć `position: relative`, `transform-style: preserve-3d` i przejście `transform 0.6s`.
2. E6. Nałóż `.przod` i `.tyl` na siebie (`position: absolute; inset: 0`), ukryj ich tylne strony (`backface-visibility: hidden`) i obróć `.tyl` o 180° wokół osi Y.
3. E7. Po najechaniu na kartę lub fokusie z klawiatury (`.karta:focus-visible`) obróć `.karta-wnetrze` o `rotateY(180deg)`. Sprawdź oba sposoby i zrób zrzut obróconej karty.

4 Animacje i pseudoelement

1. E8. Napisz animację `@keyframes obrot` (od 0 do 360°) i przypisz ją do `.loader: 1s`, liniowo, w nieskończoność. W panelu Animations spowolnij ją do 10% i obejrzyj.
2. E9. Dodaj tekstowi obok loadera łagodne pulsowanie przezroczystości (od 1 do 0,4, 1,5s, `alternate`).
3. E10. Pseudoelementem `::before` dodaj na początku bloku `.promo` znak 🍷 i spację (`content`) — bez zmiany HTML. Sprawdź w F12, że pseudoelement widać w drzewie Elements jako `::before`.

5 Preferencje użytkownika

Do testów użyj panelu Rendering w F12 — nie musisz zmieniać ustawień systemu Windows.

1. E11. W `@media (prefers-color-scheme: dark)` nadpisz **tylko zmienne** w `:root` ciemną paletą (tło ok. #1c1410, tekst jasny, akcent jaśniejszy, np. #d9a77a). Włącz emulację `prefers-color-scheme: dark` i sprawdź każdą sekcję — czy cały tekst jest czytelny (np. stopka)? Popraw kolory, jeśli trzeba.
2. E12. Kontrast: w F12 zaznacz tekst przycisku „Zamów na wynos” w obu trybach i odczytaj współczynnik kontrastu z próbnika koloru w zakładce Styles (wymagane co najmniej 4,5:1).
3. E13. W `@media (prefers-reduced-motion: reduce)` wyłącz obrót loadera i kart (karta ma od razu pokazywać tył bez obracania) oraz skróć przejścia. Sprawdź emulację `prefers-reduced-motion: reduce`.
4. E14. Sprawdź arkusz walidatorem CSS W3C. Zrób zrzuty: tryb jasny z przyciskiem w stanie hover, tryb ciemny z obróconą kartą, loader.

W sprawozdaniu umieść

- ☐ tabela zmiennych CSS z wartościami w trybie jasnym i ciemnym
- ☐ tabela E1–E14 z fragmentami kodu
- ☐ zrzuty: tryb jasny z hover, tryb ciemny z obróconą kartą, panel Animations z loaderem
- ☐ współczynniki kontrastu przycisku w obu trybach
- ☐ folder `zad16`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego w transition lepiej wymienić konkretne właściwości niż użyć all?
2. Czym różni się `:focus` od `:focus-visible` i dlaczego nie wolno usuwać obrysu fokusu bez zamiennika?
3. Jaką rolę pełnią `perspective` i `backface-visibility` w karcie obracanej?
4. Dlaczego tryb ciemny łatwiej zrobić na zmiennych CSS? Ile reguł trzeba by zmienić bez nich?
5. Komu pomaga ustawienie `prefers-reduced-motion` i co powinna wtedy zrobić strona?

Kryteria oceny

Kryterium	Punkty
Wszystkie kolory zamienione na zmienne w <code>:root</code> , strona wygląda bez zmian (E1)	3
Przyciski: hover z przejściem, active, widoczny focus-visible (E2–E4)	4
Karta obraca się po najechaniu i z klawiatury, tył czytelny, przód ukryty (E5–E7)	4
Loader obraca się w pętli, pulsujący tekst, ikona przez <code>::before</code> (E8–E10)	3
Tryb ciemny tylko przez zmienne, czytelny w każdej sekcji, kontrast $\geq 4,5:1$, reduced motion (E11–E13)	4
Zrzuty, walidacja i odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Grafika do strony w GIMP

Czas: 2 × 45 min

praca indywidualna

Cel: Przygotowanie grafik na stronę WWW w programie GIMP według precyzyjnych poleceń (wymiary, kadr, tryb koloru, przezroczystość, format) oraz świadomy wybór formatu i licencji grafiki.

Po zajęciach potrafisz:

- skaluje obraz do zadanych wymiarów z zachowaniem proporcji
- kadruje obraz do danego fragmentu i proporcji
- zamienia obraz na odcienie szarości i usuwa tło logo, zachowując przezroczystość (PNG)
- eksportuje grafikę do JPG, WebP, PNG i ICO oraz porównuje rozmiary plików
- odczytuje warunki licencji Creative Commons i zapisuje poprawne oznaczenie autorstwa

Sprzęt i materiały

- komputer z Windows 11, myszą i przeglądarką
- folder `r1_17_grafika` z Pliki_do_zadan: krajobraz.jpg, logo_biale_tlo.png, podglad.html

Oprogramowanie

- **GIMP 3** — nazwy menu podano po polsku i (w nawiasie) po angielsku gimp.org/downloads
- **Squoosh (opcjonalnie, do porównania)** — kompresja w przeglądarce, plik nie jest wysyłany na serwer squoosh.app
- **Openverse** — wyszukiwarka grafik na wolnych licencjach (polecenie G12) openverse.org

Przygotowanie przed zajęciami

- Skopiuj folder `r1_17_grafika` do folderu `nazwisko_imie` jako `zad17`. Wszystkie wyeksportowane pliki zapisuj w tym folderze, dokładnie pod podanymi nazwami — wtedy `podglad.html` pokaże je automatycznie.

Bezpieczeństwo

- Nie nadpisuj plików źródłowych — zawsze używaj Plik → Wyeksportuj jako... (File → Export As...) z nową nazwą.
- Do własnych stron używaj tylko grafik własnych lub na licencjach, które na to pozwalają, i zawsze podawaj wymagane oznaczenie autorstwa.

Przebieg zadania

1 Zdjęcie: skalowanie, kadr, szarość

Po każdym eksporcie zapisz rozmiar pliku (Eksplorator plików → Właściwości) w tabeli sprawozdania.

1. G0. Otwórz `krajobraz.jpg`. Odczytaj w Obraz → Właściwości obrazu (Image → Image Properties) wymiary w pikselach, rozdzielczość i rozmiar pliku.
2. G1. Przeskaluj obraz do szerokości **1200px** z zachowaniem proporcji: Obraz → Skaluj obraz... (Image → Scale Image...), łańcuch połączony. Zanotuj wysokość, jaką wyliczył GIMP. Wyeksportuj jako `krajobraz_1200.jpg` z jakością **80**.
3. G2. Cofnij skalowanie (Ctrl+Z) i wyeksportuj oryginał z jakością 40 jako `test_q40.jpg`. Powiększ oba pliki do 200% i porównaj artefakty kompresji (niebo, brzeg słońca).
4. G3. Wykadruj kwadratowy fragment ze słońcem: narzędzie Zaznaczanie prostokątne (R), w opcjach narzędzia wpisz położenie **1700, 300** i rozmiar **1000 × 1000**, następnie Obraz → Przytnij do zaznaczenia (Image → Crop to Selection). Przeskaluj do **600 × 600** i wyeksportuj jako `kadr_600.jpg` (jakość 80).
5. G4. Otwórz `krajobraz_1200.jpg`, zamień go na odcienie szarości: Kolory → Desaturacja → Desaturacja... (Colors → Desaturate → Desaturate...) i wyeksportuj jako `krajobraz_szary.jpg` (jakość 80).
6. G5. Wyeksportuj kolorowy obraz 1200 × 800 jako `krajobraz_1200.webp` (jakość 80, bez kompresji bezstratnej). Porównaj rozmiar z `krajobraz_1200.jpg`.

2 Logo z przezroczystym tłem

Logo ma białe tło na zewnątrz, ale także białe koło w środku, które ma **zostać białe**.

1. G6. Otwórz `logo_biale_tlo.png`. Dodaj kanał alfa: Warstwa → Przezroczystość → Dodaj kanał alfa (Layer → Transparency → Add Alpha Channel).
2. G7. Narzędziem Zaznaczanie rozmyte (różdżka, klawisz U) kliknij białe tło **poza** zielonym pierścieniem i usuń je klawiszem Delete. Sprawdź, że w środku zostało białe koło, a tło jest w szachownicę. Wyeksportuj jako `logo_przezroczyste.png`.
3. G8. Dla porównania na kopii logo użyj Kolory → Kolor na alfę... (Colors → Color to Alpha...) z kolorem białym. Co stało się ze środkiem logo? Zrób zrzut i nie zapisuj tej wersji pod nazwą z G7.
4. G9. Z przezroczystego logo przygotuj ikony: przeskaluj do **180 × 180** i wyeksportuj jako `apple-touch-icon.png`, następnie przeskaluj do **32 × 32** i wyeksportuj jako `favicon.ico`.
5. G10. Wyeksportuj logo 1000 × 1000 także jako JPG (`logo_test.jpg`). Co stało się z przezroczystością? Wyjaśnij w sprawozdaniu.

3 Sprawdzenie na stronie i licencje

1. G11. Otwórz `podglad.html` w przeglądarce. Wszystkie 6 obrazów i ikona na karcie przeglądarki mają się wyświetlić; logo w nagłówku ma leżeć na zielonym pasku bez białego kwadratu. Zrób zrzut.
2. G12. W Openverse znajdź jedną grafikę związaną z Twoim hobby na licencji CC BY lub CC BY-SA. Zapisz: tytuł, autora, źródło (adres) i licencję z odnośnikiem (zasada TASL). Wyjaśnij, czego ta licencja wymaga, a na co pozwala, i czym różni się od CC BY-NC-ND.

W sprawozdaniu umieść

- ☐ tabela plików: nazwa, wymiary, format, jakość, rozmiar w KB
- ☐ zrzuty: porównanie jakości 80 i 40 (G2), wynik Koloru na alfę (G8), podgląd strony (G11)
- ☐ wyjaśnienie różnicy między różdżką a Kolorem na alfę i utraty przezroczystości w JPG
- ☐ oznaczenie autorstwa grafiki z Openverse (G12)

- ☐ folder **zad17** z wyeksportowanymi plikami
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego przy skalowaniu zdjęcia trzeba zachować proporcje i dlaczego nie powiększa się małych zdjęć?
2. Który format wybierzesz dla zdjęcia, który dla logo z przezroczystością, a który dla ikony karty przeglądarki? Uzasadnij.
3. Czym różni się kompresja stratna od bezstratnej? Podaj po jednym formacie każdego rodzaju.
4. Co oznaczają człony BY, SA, NC i ND w licencjach Creative Commons?

Kryteria oceny

Kryterium	Punkty
krajobraz_1200.jpg 1200 × 800 i porównanie jakości (G1–G2)	3
kadr_600.jpg — kwadrat 600 × 600 ze słońcem (G3)	2
krajobraz_szary.jpg i krajobraz_1200.webp, porównanie rozmiarów (G4–G5)	3
logo_przezroczyste.png: tło przezroczyste, środek biały; wyjaśnienie G8 i G10 (G6–G8, G10)	4
apple-touch-icon.png 180 × 180 i favicon.ico 32 × 32, poprawny podgląd strony (G9, G11)	3
Poprawne oznaczenie autorstwa i opis licencji (G12)	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt: strona wizytówka — część 1 (makieta, HTML, grafika)

Czas: 2 × 45 min

zespoły 1-2 osobowe

Cel: Zaplanowanie i zbudowanie struktury HTML wszystkich podstron strony wizytówki fikcyjnej firmy: od karty projektu i makiety, przez semantyczny HTML z menu łączącym podstrony, po przygotowane grafiki.

Po zajęciach potrafisz:

- określa cele strony, grupę docelową i mapę strony
- rysuje makietę (wireframe) strony w wersji na komputer i telefon
- buduje spójną strukturę semantyczną kilku podstron ze wspólnym nagłówkiem, menu i stopką
- przygotowuje grafiki o właściwych wymiarach i formatach oraz ikonę strony
- dzieli pracę w zespole i dokumentuje postęp

Sprzęt i materiały

- komputer z Windows 11, przeglądarką, VS Code i GIMP
- folder `r1_18_szablon` z Pliki_do_zadan
- kartka A4 i ołówki (makieta na papierze) lub konto w Penpot/Figma

Oprogramowanie

- **Penpot (makieta, opcjonalnie)** — darmowy, open source, działa w przeglądarce; alternatywnie Figma (<https://www.figma.com/>) lub papier penpot.app
- **Visual Studio Code z Live Server** — HTML podstron code.visualstudio.com
- **GIMP 3** — przygotowanie grafik jak w zadaniu 17 gimp.org/downloads
- **Validator HTML W3C** — „Validate by File Upload” dla każdej podstrony validator.w3.org

Przygotowanie przed zajęciami

- Wybierz w zespole firmę z listy w kroku 1 (albo zaproponuj własną fikcyjną) i zbierz pomysły na treść.
- Grafiki: własne zdjęcia lub rysunki albo pliki na licencji CC0, CC BY lub CC BY-SA (np. z Openverse) — zapisz źródło i licencję każdej grafiki.

Bezpieczeństwo

- Firma i dane kontaktowe mają być fikcyjne: adres z kodem 00-000, telefon 500 000 000, e-mail w domenie example.com. Nie podawaj danych prawdziwych osób ani firm.
- Nie umieszczaj zdjęć osób bez ich zgody (wizerunek jest chroniony prawem). Nie używaj logo ani nazw istniejących marek.
- Pracujcie w folderze `projekt_nazwisko1_nazwisko2`; róbcie kopię folderu na koniec zajęć (pendrive lub dysk sieciowy).

Przebieg zadania

1 Karta projektu (15 min)

Lista firm do wyboru (wszystkie fikcyjne): warsztat rowerowy „Szprycha”, salon groomerski „Futrzak”, szkoła jazdy „Zielone Światło”, studio fotograficzne „Kadr”, kwiaciarnia „Bratek”, gospodarstwo agroturystyczne „Pod Lipą”, pracownia ceramiki „Glina”, wypożyczalnia kajaków „Nurt”.

1. P1. Skopiuj `r1_18_szablon` jako `projekt_nazwisko1_nazwisko2`. Wypełnij plik `PROJEKT.txt`: zespół, firma, grupa docelowa, 3 cele strony, paleta 3–5 kolorów (hex), kroje pisma, podział pracy.
2. P2. Zaplanuj mapę strony: co najmniej 4 podstrony — `index.html`, `oferta.html`, `o-nas.html`, `kontakt.html` — i dla każdej wypisz w `PROJEKT.txt` jej zawartość (sekcje).

2 Makieta (20 min)

Makieta to szkic bloków (prostokąty z podpisami), bez kolorów i zdjęć. Krzyżyk w prostokącie oznacza obraz.

1. P3. Narysuj makietę strony głównej w dwóch wersjach: komputer (ok. 1280px) i telefon (ok. 375px). Zaznacz nagłówek z logo, menu (na telefonie zwijane), baner, sekcje treści, stopkę.
2. P4. Narysuj makietę jednej podstrony (oferta lub kontakt) w wersji na komputer. Sfotografuj lub wyeksportuj makiety do pliku `makieta.png` (lub `.svg` / `.pdf`) w folderze projektu.

3 Struktura HTML wszystkich podstron (35 min)

Najpierw dokończ wspólny szkielet w `index.html` (nagłówek, menu, stopka), dopiero potem kopiuj go do pozostałych podstron — zmiany we wspólnych częściach trzeba by potem nanosić 4 razy.

1. P5. Uzupełnij szablon: nazwa firmy w `title`, `description`, nagłówku i stopce. Każda podstrona ma własny `title` i `description` (do 155 znaków).
2. P6. Menu w `nav` łączy wszystkie podstrony ścieżkami względnymi. W każdej podstronie aktualny link ma atrybut `aria-current="page"`. Dodaj na początku `body` odnośnik „Przejdź do treści” prowadzący do `main` (`id="tresc"`).
3. P7. Strona główna: baner z obrazem i nagłówkiem `h1`, przycisk-odnośnik do oferty, sekcja z co najmniej 3 atutami firmy (lista lub karty `article` / `li`).
4. P8. Oferta: co najmniej 3 produkty/usługi w elementach `figure` z `figcaption` i ceną oraz tabela (`caption`, `thead`, `th scope`) — np. cennik albo godziny.
5. P9. O nas: historia firmy w `article`, cytat w `blockquote`, lista osób zalet. Kontakt: dane w `address`, godziny otwarcia w liście `dl`, formularz (min. 4 pola z `label`, walidacja natywna `required` / `pattern`).
6. P10. Hierarchia nagłówków: jeden `h1` na podstronę, bez przeskoków poziomów. Sprawdź każdą podstronę walidatorem W3C (cel: 0 błędów).

4 Grafiki (15 min) i podsumowanie (5 min)

Grafiki zapisuj w folderze `img`, nazwy małymi literami, bez polskich znaków i spacji.

1. P11. Przygotuj w GIMP: logo z przezroczystym tłem (PNG), `favicon.ico` 32 × 32 i `apple-touch-icon.png` 180 × 180, baner o szerokości 1600px (WebP lub JPG, do 200 KB) i co najmniej 3 grafiki produktów o jednakowych wymiarach (np. 800 × 600, do 100 KB każda).
2. P12. Wstaw grafiki do HTML z sensownym `alt`, `width` i `height` (dla logo-odnośnika `alt` opisuje cel linku albo jest pusty, jeśli obok jest nazwa firmy). Dopisz źródła i licencje grafik do `PROJEKT.txt`.
3. P13. Zapisz w sprawozdaniu, co zostało do zrobienia w zadaniu 19 (CSS), i zrób kopię folderu projektu.

W sprawozdaniu umieść

- ☐ wypełniony `PROJEKT.txt` (cele, mapa strony, paleta, podział pracy, źródła grafik)

- ☐ makiety: strona główna (komputer i telefon) oraz jedna podstrona
- ☐ zrzuty 4 podstron bez CSS i wynik walidacji W3C każdej z nich
- ☐ tabela grafik: plik, wymiary, format, rozmiar, źródło i licencja
- ☐ kto wykonał które podstrony i grafiki
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Po co robić makietę przed pisaniem kodu? Co zyskuje zespół?
2. Dlaczego każda podstrona powinna mieć inny title i description?
3. Do czego służą odnośnik „Przejdź do treści” i atrybut aria-current?
4. Kiedy tekst alternatywny obrazu powinien być pusty (alt=""), a kiedy opisowy?

Kryteria oceny

Kryterium	Punkty
Karta projektu: cele, grupa docelowa, mapa strony, paleta, podział pracy (P1–P2)	3
Makiety: strona główna w 2 wersjach i jedna podstrona (P3–P4)	3
4 podstrony ze wspólnym nagłówkiem, menu z aria-current, odnośnikiem „Przejdź do treści” i stopką; różne title/description (P5–P6)	3
Wymagana zawartość podstron: baner i atuty, figure + tabela, article + blockquote, address + dl + formularz (P7–P9)	5
Jeden h1, poprawna hierarchia, 0 błędów walidatora na każdej podstronie (P10)	2
Grafiki o zadanych wymiarach i rozmiarach, alt/width/height, licencje (P11–P12)	2
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt: strona wizytówka — część 2 (CSS, RWD, audyt, prezentacja)

Czas: 2 × 45 min

zespoły 1-2 osobowe (te same co w zadaniu 18)

Cel: Dokończenie strony wizytówki z zadania 18: wspólny arkusz CSS ze zmiennymi, układ responsywny (mobile first), efekty interakcji, kontrola jakości (walidacja W3C, Lighthouse) i krótka prezentacja projektu.

Po zajęciach potrafisz:

- tworzy jeden zewnętrzny arkusz CSS dla wszystkich podstron, z paletą w zmiennych
- buduje układ responsywny z menu zwijanym i siatką kart
- dodaje efekty `:hover/:`focus-visible z przejściami, szanując `prefers-reduced-motion`
- przeprowadza audyt Lighthouse i walidację W3C oraz poprawia znalezione błędy
- prezentuje projekt i uzasadnia decyzje projektowe

Sprzęt i materiały

- komputer z Windows 11, Chrome lub Edge, VS Code
- folder projektu z zadania 18
(`projekt_nazwisko1_nazwisko2`)
- projektor (prezentacje na koniec zajęć)

Oprogramowanie

- **Visual Studio Code z Live Server** — Lighthouse uruchamiaj na stronie z Live Server ([http://127.0.0.1:5500/...](http://127.0.0.1:5500/)), nie z `file://code.visualstudio.com`
- **Lighthouse (wbudowany w Chrome/Edge)** — F12 → zakładka Lighthouse; tryb Navigation, urządzenie Mobile developer.chrome.com/docs/lighthouse/overview
- **Walidatory W3C** — CSS: <https://jigsaw.w3.org/css-validator/validator.w3.org>

Przygotowanie przed zajęciami

- Sprawdź, czy masz kompletny folder projektu z zadania 18 (4 podstrony, grafiki, PROJEKT.txt, makiety). Brakujące elementy HTML uzupełnij w pierwszych 10 minutach.
- Przypomnij sobie zadania 13–16 (Flexbox, Grid, RWD, efekty) — możesz korzystać z własnych rozwiązań.

Bezpieczeństwo

- Przed większymi zmianami zrób kopię folderu (np. `projekt_..._kopia`). Zmiany w podstronach nanoś we wszystkich plikach jednako.
- Dane w projekcie pozostają fikcyjne (adres 00-000, telefon 500 000 000, domena `example.com`).

Przebieg zadania

1 Arkusz i układ (35 min)

Wszystkie style w jednym pliku `css/style.css` dołączonym do każdej podstrony. Pisz mobile first: najpierw telefon, potem `@media (min-width: ...)`.

1. S1. Zapisz paletę i kroje z `PROJEKT.txt` jako zmienne w `:root` (kolory, promień zaokrąglenia, cień, czas przejścia). W regułach używaj tylko zmiennych — żadnych kolorów wpisanych na sztywno poza `:root` i ewentualnym trybem ciemnym.
2. S2. Style bazowe: `box-sizing: border-box`, krój tekstu i nagłówków, `line-height` ok. 1,5–1,6, `img { max-width: 100%; height: auto; }`, rozmiar `h1` przez `clamp()`, treść `main` z maksymalną szerokością ok. 1100px i wyśrodkowana.
3. S3. Nagłówek z logo i nazwą (Flexbox). Menu: na telefonie zwijane bez JS (checkbox + label jak w zadaniu 15), od 600px w poziomie; aktywna podstrona wyróżniona przez selektor `[aria-current="page"]`; menu przyklejone (`position: sticky`).
4. S4. Odnośnik „Przejdź do treści” ukryty poza ekranem i widoczny dopiero po fokusie klawiaturą.
5. S5. Karty atutów i produktów w siatce Grid: 1 kolumna → 2 (od 600px) → 3 (od 900px). Na stronie kontaktu od 900px dane i formularz obok siebie.

2 Szczegóły i efekty (20 min)

1. S6. Tabela: obramowania zwinięte (`border-collapse`), wyróżniony wiersz nagłówka, co drugi wiersz z tłem (`:nth-child(even)`). Formularz: pola na pełną szerokość kolumny, widoczny fokus, przycisk w stylu strony.
2. S7. Przyciski i karty: `:hover` i `:focus-visible` z płynnym `transition` (uniesienie, cień). Dodaj `@media (prefers-reduced-motion: reduce)` wyłączające ruch.
3. S8. Opcjonalnie (bez dodatkowych punktów, jeśli zostanie czas): tryb ciemny przez `prefers-color-scheme` — tylko nadpisanie zmiennych.

3 Kontrola jakości (20 min)

Każdy błąd znaleziony w audycie zapisz w tabeli sprawozdania razem ze sposobem poprawy.

1. S9. Przetestuj każdą podstronę w trybie urządzeń przy 375px, 768px i 1280px: brak poziomego przewijania, menu działa, grafiki się mieszczą. Przejdź stroną samą klawiaturą (Tab, Enter, Spacja).
2. S10. Sprawdź wszystkie podstrony walidatorem HTML i arkusz walidatorem CSS — cel: 0 błędów.
3. S11. Uruchom Lighthouse (Mobile; kategorie Accessibility, Best Practices, SEO) dla strony głównej i kontaktu. Wymaganie: **dostępność ≥ 90**. Popraw zgłoszone problemy (najczęściej kontrast, brak alt, etykiety, kolejność nagłówków) i uruchom audyt ponownie — zapisz wyniki przed i po poprawkach.

4 Prezentacja (15 min na całą klasę)

Każdy zespół ma maksymalnie 3 minuty.

1. S12. Przedstaw projekt na projektorze: firma i grupa docelowa, makieta vs wynik, strona na telefonie (tryb urządzeń) i komputerze, wynik Lighthouse, jedna rzecz, z której jesteście dumni, i jedna, którą byście poprawili.
2. S13. Oceń jeden inny zespół na karcie koleżeńskej (czytelność, spójność, działanie na telefonie — po 1–3 pkt i jedno zdanie komentarza).

W sprawozdaniu umieść

- ☐ zrzuty strony głównej i jednej podstrony przy 375px (menu rozwinięte), 768px i 1280px
- ☐ zestawienie makieta ↔ wynik (co się zmieniło i dlaczego)
- ☐ tabela audytu: problem, podstrona, poprawka; wyniki Lighthouse przed i po

- ☐ wyniki walidacji HTML i CSS
- ☐ kto wykonał które części arkusza
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego wszystkie podstrony korzystają z jednego zewnętrznego arkusza CSS? Co by się stało przy stylach wpisanych w każdą podstronę?
2. Jak selektor [aria-current="page"] łączy dostępność z wyglądem menu?
3. Lighthouse pokazał 100 punktów dostępności. Czy to znaczy, że strona jest w pełni dostępna? Czego automat nie sprawdzi?
4. Jaki punkt przerwania wybraliście dla zmiany układu kart i dlaczego właśnie taki?

Kryteria oceny

Kryterium	Punkty
Jeden zewnętrzny arkusz, paleta w zmiennych, typografia i clamp() (S1–S2)	3
Nagłówek, menu zwijane i przyklejone z wyróżnioną podstroną, skip link (S3–S4)	4
Siatka kart 1 → 2 → 3, układ kontaktu, tabela i formularz (S5–S6)	4
Efekty hover/focus-visible z przejściami i prefers-reduced-motion (S7)	2
Testy przy 3 szerokościach i z klawiatury, 0 błędów W3C, Lighthouse dostępność ≥ 90 z tabelą poprawek (S9–S11)	4
Prezentacja i ocena koleżeńska (S12–S13)	2
Odpowiedzi na pytania kontrolne	1
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Próba INF.03: witryna bez skryptów

Czas: 2 × 45 min

praca indywidualna, samodzielna (jak na egzaminie)

Cel: Samodzielne wykonanie w 90 minut witryny według szczegółowego opisu: przygotowanie grafiki, struktura HTML z tabelą, listą i obrazami oraz arkusz CSS z zadanymi kolorami, krojami i wymiarami — w warunkach zbliżonych do egzaminu.

Po zajęciach potrafisz:

- przygotowuje grafikę według poleceń (kadrowanie, skalowanie, format)
- buduje strukturę witryny z bloków o zadanych nazwach klas
- tworzy tabelę i listę numerowaną z podanych danych
- formatuje witrynę według specyfikacji: kolory, kroje, szerokości, wysokości, obramowania, pseudoklasy
- kontroluje czas i ocenia własną pracę według kryteriów

Sprzęt i materiały

- komputer z Windows 11, przeglądarką, VS Code (lub Notepad++) i GIMP
- folder `r1_20_pliki` z Pliki_do_zadan: `niebo_oryginal.png`, `logo.png`, `tresci.txt`

Oprogramowanie

- **Visual Studio Code** lub **Notepad++** — na egzaminie zwykle bez rozszerzeń generujących kod — w tej próbie nie używaj Emmeta do całych bloków ani asystentów AI
code.visualstudio.com
- **GIMP 3** — część graficzna
gimp.org/downloads
- **Walidator HTML W3C** — tylko na końcu, do samooceny
validator.w3.org

Bezpieczeństwo

- Pracujesz samodzielnie — bez internetu (poza walidatorem na końcu), notatek i pomocy kolegów, jak na egzaminie.
- Wyniki zapisz w folderze `nazwisko_imie_zad20` na pulpicie (lub wskazanym przez nauczyciela); na koniec spakuj go do ZIP.

Przebieg zadania

1 Grafika

Do przygotowania grafiki użyj programu graficznego (np. GIMP).

1. G1. Obraz `niebo_oryginal.png` ma białą ramkę o szerokości 50px z każdej strony. Wykadruj go tak, aby usunąć ramkę w całości (wynik: 1500 × 1000 px).
2. G2. Przeskaluj wykadrowany obraz do szerokości 450px z zachowaniem proporcji. Zapisz go w formacie JPG jako `niebo.jpg` w folderze z witryną.
3. G3. Skopiuj do folderu z witryną plik `logo.png` (bez zmian).

2 Witryna — struktura

Utwórz plik `orion.html`. Teksty skopiuj z pliku `tresci.txt`. Układ bloków:

```
+-----+
|                baner (header)                |
+-----+
|  lewy  |  srodkowy (main)  |  prawy  |
|  25%   |        50%       |   25%   |
+-----+
|                stopka (footer)                |
+-----+
```

- W1. Ustawienia witryny: język polski, kodowanie UTF-8, tytuł na karcie przeglądarki „Klub Astronomiczny Orion”, arkusz `styl.css` dołączony w nagłówku dokumentu.
- W2. Baner: element `header` z klasą `baner`; w nim obraz `logo.png` z tekstem alternatywnym „Logo klubu — gwiazdozbiór Oriona”, nagłówek pierwszego stopnia z nazwą klubu i akapit z informacją o spotkaniach.
- W3. Panel lewy (`section`, klasa `lewy`): nagłówek drugiego stopnia „Najbliższe obserwacje” i lista numerowana z 4 elementami.
- W4. Panel środkowy (`main`, klasa `srodkowy`): nagłówek drugiego stopnia „Kalendarz zjawisk”, tabela z 3 kolumnami i 5 wierszami (pierwszy wiersz to komórki nagłówkowe Data, Zjawisko, Godzina) oraz akapit pod tabelą.
- W5. Panel prawy (`section`, klasa `prawy`): nagłówek drugiego stopnia „Dołącz do nas”, obraz `niebo.jpg` z tekstem alternatywnym „Nocne niebo nad obserwatorium”, akapit i odnośnik „Napisz do opiekuna” otwierający pocztę na adres `orion@example.com`.
- W6. Stopka (`footer`, klasa `stopka`): akapit „Stronę wykonał: ” i Twój numer z dziennika oraz klasa.

3 Witryna — styl CSS

Plik `styl.css` zawiera formatowanie (pozostałe właściwości domyślne):

- S1. Cała strona (body): krój Tahoma, gdy niedostępny — Verdana, a w ostateczności dowolny bezszeryfowy; marginesy 0.
- S2. Baner: kolor tła `#0B1E3F`, kolor czcionki biały, wysokość 120px, wypełnienie 10px. Logo w banerze przyległe do lewej strony (float) z marginesem prawym 20px; nagłówek h1 w banerze: rozmiar czcionki 200%.
- S3. Panele lewy, środkowy i prawy: ułożone obok siebie (przyległe do lewej), wysokość 520px, wypełnienie 10px, a wypełnienie wliczone w wymiary (`box-sizing`). Szerokości: lewy i prawy 25%, środkowy 50%. Kolor tła paneli lewego i prawego `#DCE6F5`.
- S4. Stopka: czyszczenie opływania z obu stron, kolor tła `#0B1E3F`, kolor czcionki biały, tekst wyśrodkowany, wypełnienie 15px.
- S5. Nagłówki drugiego stopnia: kolor `#0B1E3F`, dolne obramowanie 3px ciągłe w kolorze `#F2B134`, dolne wypełnienie 5px. Elementy listy: margines dolny 8px.
- S6. Tabela: szerokość 100%, obramowania zwinięte. Komórki tabeli: obramowanie 1px ciągłe `#0B1E3F`, wypełnienie 5px. Komórki nagłówkowe: kolor tła `#F2B134`. Wiersz po najechaniu kursorem: kolor tła `#DCE6F5`.
- S7. Obraz w panelu prawym: szerokość 100%, zaokrąglone narożniki 20px. Odnośniki: kolor `#0B1E3F`, pogrubione, bez podkreślenia; po najechaniu — podkreślone.
- S8. Na koniec sprawdź witrynę walidatorem HTML i porównaj wynik z opisem. Spakuj folder do ZIP i wypełnij tabelę samooceny.

W sprawozdaniu umieść

- ☐ folder `nazwisko_imie_zad20` (orion.html, styl.css, niebo.jpg, logo.png) spakowany do ZIP
- ☐ zrzut ekranu witryny w oknie 1280px
- ☐ tabela samooceny: każde kryterium z punktacją własną i uzasadnieniem
- ☐ czas ukończenia części: grafika, struktura, CSS
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego w stopce trzeba użyć clear: both? Co się stanie bez tej właściwości?
2. Co zmienia box-sizing: border-box w panelach o szerokości 25% + 50% + 25% z wypełnieniem 10px?
3. Dlaczego w deklaracji kroju podaje się kilka nazw zakończonych rodziną sans-serif?
4. Które polecenia sprawiły Ci największą trudność i ile czasu na nie straciłeś? Co powtórzysz przed egzaminem?

Kryteria oceny

Kryterium	Punkty
Grafika: niebo.jpg bez ramki, 450 × 300 px, format JPG; logo w folderze (G1–G3)	3
Ustawienia dokumentu: lang, UTF-8, title, dołączony styl.css (W1)	1
Baner i stopka z wymaganą zawartością i klasami (W2, W6)	2
Panel lewy z listą numerowaną; panel środkowy z tabelą 3 × 5 z komórkami th i akapitem (W3–W4)	3
Panel prawy: obraz z alt, akapit, odnośnik mailto (W5)	2
Formatowanie body, banera i stopki zgodne z opisem (S1–S2, S4)	3
Trzy panele obok siebie z zadanymi szerokościami, wysokością i tłem (S3)	2
Nagłówki, lista, tabela z :hover, obraz i odnośniki zgodne z opisem (S5–S7)	3
Samoocena i odpowiedzi na pytania kontrolne	1
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

JavaScript — pierwsze skrypty

Czas: 2 × 45 min

praca indywidualna

Cel: Pierwsze programy w JavaScriptcie: konsola przeglądarki, zmienne i typy, operatory (w tym `==` i `===`), instrukcje warunkowe i pętle, wyświetlanie wyniku na stronie oraz szukanie błędów z punktem przerwania.

Po zajęciach potrafisz:

- dołącza skrypt z pliku i wypisuje wyniki w konsoli F12
- deklaruje zmienne `let/const` i rozpoznaje typy operatorem `typeof`
- przewiduje wynik operatorów `+`, `-`, `%`, `==` i `===` dla liczb i tekstów
- stosuje `if/else if` oraz pętle `for` i `while` (także zagnieżdżone)
- znajduje błąd w skrypcie, zatrzymując go punktem przerwania i obserwując wartości zmiennych

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome lub Edge
- folder `r1_21_start` z Pliki_do_zadan (`index.html`, `style.css`, `skrypt.js`, `bledy.js`)

Oprogramowanie

- **Visual Studio Code z Live Server** — po zapisie strona odświeża się sama code.visualstudio.com
- **Konsola i debugger przeglądarki** — F12 → Console oraz Sources (punkty przerwania) developer.chrome.com/docs/devtools/javascript

Przygotowanie przed zajęciami

- Skopiuj folder `r1_21_start` do folderu `nazwisko_imie` jako `zad21`.
- Przypomnij sobie z wykładu 1.15: `let` i `const`, typy, `==` i `===`, `if`, `for`, `while`.

Bezpieczeństwo

- Nie wklejaj do konsoli przeglądarki kodu z nieznanych źródeł (np. „wklej to, a dostaniesz darmowe punkty”) — może przejąć Twoje konto na tej stronie. W tym zadaniu wpisuj tylko własny kod.
- W pętli `while` zawsze sprawdź, czy warunek kiedyś stanie się fałszywy — nieskończona pętla zawiesza kartę (zamknij ją wtedy przez Menedżera zadań przeglądarki: Shift+Esc).

Przebieg zadania

1 Konsola i zmienne

Uruchom `index.html` przez Live Server i otwórz konsolę (F12 → Console). Każdą część kodu w `skrypt.js` poprzedź komentarzem z numerem polecenia.

- J1. Sprawdź, że w konsoli widać „Skrypt działa!”. W `head` znajdź znacznik `script` z atrybutem `defer` — usuń na chwilę `defer` z `bledy.js`, odśwież i zapisz komunikat błędu z konsoli. Przywróć `defer`.
- J2. Zadeklaruj stałą `imie` (tekst), zmienną `wiek` (liczba), stałą `czyUczen` (wartość logiczna) i zmienną bez wartości. Wypisz każdą z jej typem: `console.log(wiek, typeof wiek)`.
- J3. Zwiększ `wiek` o 1 i wypisz zdanie „Za rok Ola będzie mieć 16 lat.” szablonem tekstowym (odwrotne apostrofy i `${...}`). Spróbuj zmienić wartość stałej `imie` — zapisz komunikat błędu i usuń tę linię.
- J4. Zanim uruchomisz, zapisz w sprawozdaniu przewidywany wynik, potem sprawdź w konsoli: `"5" + 3`, `"5" - 3`, `"5" * "2"`, `5 == "5"`, `5 === "5"`, `7 % 3`, `0.1 + 0.2`.

2 Kalkulator BMI w konsoli

BMI = masa [kg] / wzrost² [m]. Przedziały: poniżej 18,5 — niedowaga; od 18,5 do mniej niż 25 — waga prawidłowa; od 25 do mniej niż 30 — nadwaga; od 30 — otyłość.

- J5. Zapisz w stałych masę 62 kg i wzrost 1,70 m (w kodzie kropka: `1.70`). Oblicz BMI.
- J6. Instrukcją `if ... else if ... else` dobierz opis według przedziałów. Wypisz wynik z jednym miejscem po przecinku: `bmi.toFixed(1)`.
- J7. Przetestuj skrypt dla 3 innych zestawów danych tak, aby otrzymać trzy różne opisy. Zapisz dane i wyniki w tabeli sprawozdania. Do której kategorii trafi BMI równe dokładnie 25?

3 Pętle

- J8. Pętlą `for` wypisz w jednym wierszu konsoli liczby parzyste od 2 do 20 (krok 2, bez instrukcji `if`).
- J9. Pętlą `while` policz, ile razy trzeba podwoić liczbę 1, aby przekroczyła 1000. Wypisz liczbę kroków i końcową wartość.
- J10. Dwoma zagnieżdżonymi pętlami `for` wypisz w konsoli tabliczkę mnożenia 10 × 10 — każdy wiersz w osobnej linii, liczby wyrównane do 4 znaków (`String(liczba).padStart(4)`).
- J11. Zbuduj tę samą tabliczkę jako tabelę HTML w zmiennej tekstowej (wiersz nagłówek 1–10, pierwsza kolumna jako `th`) i wstaw ją do `div#tabliczka` przez `innerHTML`. Komórki na przekątnej (`w === k`) mają klasę `kwadrat`.

4 Debugowanie skryptu z błędami

Skrypt `bledy.js` ma policzyć średnią 7 ocen (6 z tablicy i jedna dodatkowa), ale strona pokazuje „Średnia: NaN”. Poprawna średnia to 4,14.

- J12. W F12 → Sources otwórz `bledy.js` i kliknij numer linii z `suma = suma + oceny[i];`, aby ustawić punkt przerwania. Odśwież stronę. Przyciskiem „Step over” (F10) przejdź kilka obrotów pętli i zapisz w tabeli wartości `i`, `oceny[i]` i `suma`. W którym obrocie pojawia się `undefined` i `NaN`?
- J13. Popraw warunki pętli. Odśwież — jaki wynik pokazuje teraz strona? Ponownie zatrzymaj skrypt (punkt przerwania na linii z `dodatkowa`) i sprawdź w panelu Scope typ zmiennej `suma` po dodaniu oceny dodatkowej. Popraw drugi błąd (`Number(...)`).
- J14. Strona ma pokazywać „Średnia: 4.14”. Zrób zrzuty: konsola z wynikami J2–J10, strona z tabliczką i średnią, debugger zatrzymany w pętli.

W sprawozdaniu umieść

- ☐ tabela przewidywań i wyników operatorów (J4)
- ☐ tabela testów BMI (J7)
- ☐ tabela wartości zmiennych z debugera (J12) i opis obu błędów z poprawkami
- ☐ rzuty: konsola, strona z tabliczką i średnią, debugger z punktem przerwania
- ☐ pliki skrypt.js i błedy.js
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się let od const? Kiedy użyjesz każdego z nich?
2. Dlaczego "5" + 3 daje "53", a "5" - 3 daje 2? Którego porównania (== czy ===) warto używać i dlaczego?
3. Co robi atrybut defer i co się dzieje bez niego, gdy skrypt w head szuka elementu na stronie?
4. Jak punkt przerwania pomaga znaleźć błąd szybciej niż dopisywanie console.log?

Kryteria oceny

Kryterium	Punkty
Zmienne, typy, szablon tekstowy, błąd zmiany stałej, komunikat bez defer (J1–J3)	3
Tabela przewidywań operatorów z wyjaśnieniem różnic (J4)	2
Kalkulator BMI z poprawnymi przedziałami i 4 testami (J5–J7)	4
Pętle: parzyste, podwajanie, tabliczka w konsoli (J8–J10)	3
Tabliczka jako tabela na stronie z wyróżnioną przekątną (J11)	2
Debugowanie: tabela wartości, oba błędy znalezione i poprawione, wynik 4.14 (J12–J14)	4
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

JavaScript — warunki i pętle na stronie

Czas: 2 × 45 min

praca indywidualna

Cel: Pisanie funkcji, które odczytują liczbę z pola formularza, sprawdzają poprawność danych, podejmują decyzje (if, switch), liczą w pętli i wyświetlają wynik w akapicie strony.

Po zajęciach potrafisz:

- odczytuje wartość pola input i zamienia ją na liczbę
- sprawdza poprawność danych i wyświetla komunikat o błędzie
- stosuje if/else if, operator % i instrukcję switch
- oblicza sumę w pętli for i sprawdza wynik wzorem
- wypisuje wynik w akapicie przez `textContent` i testuje funkcję zestawem danych

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome lub Edge
- folder `r1_22_start` z Pliki_do_zadan (index.html, style.css, skrypt.js)

Oprogramowanie

- **Visual Studio Code z Live Server** — konsola F12 otwarta przez cały czas — tam zobaczysz błędy skryptu code.visualstudio.com
- **MDN — dokumentacja JavaScript (po polsku/angielsku)** — `Number()`, `Number.isInteger()`, `toFixed()`, `switch` developer.mozilla.org/pl/docs/Web/JavaScript

Przygotowanie przed zajęciami

- Skopiuj folder `r1_22_start` do folderu `nazwisko_imie` jako `zad22`.
- Przypomnij sobie z zadania 21: if/else, pętla for, operator %.

Bezpieczeństwo

- Wynik wyświetlaj przez `textContent`, nie `innerHTML` — tekst wpisany przez użytkownika nie może stać się kodem HTML strony.
- Nie ufaj danym z formularza: atrybuty min/max nie blokują wpisania innej wartości, więc zawsze sprawdzaj dane w skrypcie.

Przebieg zadania

1 Przelicznik temperatury

Funkcje są już podpięte do przycisków atrybutem `onclick` w HTML — uzupełniasz tylko ich treść w `skrypt.js`. Wzory: $^{\circ}\text{F} = ^{\circ}\text{C} \cdot 9/5 + 32$; $\text{K} = ^{\circ}\text{C} + 273,15$.

```
function przelicz() {
  const pole = document.getElementById("celsjusz").value; // tekst!
  const c = Number(pole);
  // ... obliczenia ...
  document.getElementById("wynikTemp").textContent = "...";
}
```

1. K1. Odczytaj temperaturę w $^{\circ}\text{C}$, przelicz na $^{\circ}\text{F}$ (1 miejsce po przecinku) i K (2 miejsca). Wyświetl w akapicie `wynikTemp` np. „25 $^{\circ}\text{C}$ = 77.0 $^{\circ}\text{F}$ = 298.15 K”.
2. K2. Gdy pole jest puste, wyświetl „Wpisz temperaturę.” i zakończ funkcję (`return`). Gdy temperatura jest niższa niż $-273,15^{\circ}\text{C}$ — komunikat o zerze bezwzględnym.
3. K3. Komunikaty o błędach mają być czerwone: nadaj akapitowi klasę `bład` (`element.className = "bład"`), a przy poprawnym wyniku usuń ją. Napisz funkcję pomocniczą `pokaz(id, tekst, czyBład)` i używaj jej we wszystkich kolejnych blokach.

2 Parzystość

1. K4. Sprawdź operatorem `%`, czy liczba jest parzysta, i wyświetl np. „14 jest liczbą parzystą”.
2. K5. Liczby ujemne też mają działać (sprawdź -3 i -4). Jaki wynik daje w konsoli `-3 % 2`? Dlaczego warunek `x % 2 === 1` byłby błędny?
3. K6. Dla pola pustego lub liczby niecałkowitej (np. 2.5) wyświetl „Podaj liczbę całkowitą.” — użyj `Number.isInteger()`.

3 Ocena z punktów

Sprawdzian ma 40 punktów. Progi procentowe: od 96% — 6, od 85% — 5, od 70% — 4, od 55% — 3, od 40% — 2, poniżej 40% — 1.

1. K7. Sprawdź, czy liczba punktów jest liczbą całkowitą z zakresu 0–40; w przeciwnym razie komunikat o błędzie.
2. K8. Oblicz procent i instrukcjami `if ... else if` wyznacz ocenę (liczbę 1–6). Kolejność warunków ma znaczenie — zacznij od najwyższego progu.
3. K9. Instrukcją `switch` zamień ocenę na słowo (celujący, bardzo dobry, dobry, dostateczny, dopuszczający, niedostateczny). Wyświetl np. „34 pkt (85.0%) — ocena 5 (bardzo dobry)”. Przetestuj wartości progowe z tabeli sprawozdania.

4 Suma ciągu i testy

1. K10. Pętlą `for` oblicz sumę $1 + 2 + \dots + n$ dla n podanego w polu (n całkowite, co najmniej 1; w przeciwnym razie komunikat).
2. K11. Dla $n \leq 10$ wyświetl cały zapis, np. „1 + 2 + 3 + 4 + 5 = 15”; dla większych tylko wynik. Dopisz sprawdzenie wzorem $n(n+1)/2$ — oba wyniki muszą być równe.
3. K12. Wypełnij tabelę testów w sprawozdaniu (wszystkie cztery bloki, w tym dane błędne). Zrób zrzut strony z wynikami wszystkich bloków i sprawdź, że konsola nie pokazuje błędów.

W sprawozdaniu umieść

- ☐ tabela testów: blok, dane wejściowe, oczekiwany wynik, wynik na stronie
- ☐ odpowiedź do K5 (reszta z dzielenia liczb ujemnych)
- ☐ zrzut strony z wynikami czterech bloków i pustą konsolą błędów

- ☐ plik skrypt.js z komentarzami K1–K12
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego wartość z pola input trzeba zamienić funkcją Number()? Co zwróci Number("") i Number("abc")?
2. Dlaczego w K8 warunki sprawdza się od najwyższego progu? Co się stanie przy odwrotnej kolejności?
3. Kiedy wygodniej użyć switch, a kiedy if/else if?
4. Dlaczego walidacja w skrypcie jest potrzebna, skoro pole ma atrybuty min i max?

Kryteria oceny

Kryterium	Punkty
Przelicznik: poprawne °F i K z zaokrągleniem, obsługa pustego pola i zera bezwzględnego (K1–K2)	3
Funkcja pokaz() i czerwone komunikaty błędów we wszystkich blokach (K3)	2
Parzystość z liczbami ujemnymi i walidacją, odpowiedź do K5 (K4–K6)	3
Ocena: walidacja zakresu, poprawne progi (test wartości 15, 16, 22, 28, 34, 38, 40), switch ze słowną nazwą (K7–K9)	4
Suma ciągu w pętli z zapisem dla $n \leq 10$ i sprawdzeniem wzorem (K10–K11)	3
Tabela testów, zrzut, brak błędów w konsoli (K12)	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

JavaScript — funkcje i tablice

Czas: 2 × 45 min

praca indywidualna

Cel: Pisanie własnych funkcji z parametrami i wartością zwracaną oraz przetwarzanie tablic liczb: średnia, minimum i maksimum, sortowanie liczbowe, filter i map.

Po zajęciach potrafisz:

- deklaruje funkcje klasyczne i strzałkowe z parametrami, wartością domyślną i instrukcją return
- wywołuje jedną funkcję wewnątrz drugiej i rozumie zasięg zmiennych lokalnych
- oblicza średnią, minimum i maksimum elementów tablicy
- sortuje liczby poprawnie funkcją porównującą i wyjaśnia, dlaczego `sort()` bez niej daje zły wynik
- stosuje metody `push`, `pop`, `forEach`, `filter` i `map`

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_23_start` z Pliki_do_zadan (`index.html`, `styl.css`, `skrypt.js`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (podgląd odświeżany po zapisie) code.visualstudio.com
- **Przeglądarka z narzędziami F12** — zakładka Console — komunikaty błędów i testy wyrażeń

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 1.16 deklarację funkcji, funkcję strzałkową i metody tablic.
- Skopiuj folder `r1_23_start` do swojego folderu `nazwisko_imie` i zmień jego nazwę na `zad23`.

Bezpieczeństwo

- Pracuj na kopii plików startowych w swoim folderze; nie zmieniaj plików w Pliki_do_zadan.

Przebieg zadania

1 Przygotowanie

Skrypt dołączono w sekcji head z atrybutem `defer`, więc wykona się po wczytaniu całej strony. Funkcja `dodajDoListy(idListy, tekst)` jest gotowa — używaj jej do wyświetlania każdego wyniku.

```
dodajDoListy("wyniki-funkcje", "F1. Pole prostokąta 4 × 7.5 = " + poleProstokata(4, 7.5));
```

1. Otwórz folder `zad23` w VS Code (Plik → Otwórz folder) i uruchom `index.html` przez Live Server (Go Live na pasku stanu).
2. Wpisz w `index.html` i w komentarzu na początku `skrypt.js` swoje imię i nazwisko. Otwórz konsolę F12 i sprawdź, że na stronie widać napis „Skrypt działa!”. Potem usuń ten przykład.
3. Każdy wynik wyświetlaj z numerem polecenia (F1, B3, C2...), tak jak w przykładzie powyżej. Po każdym poleceniu zapisz plik i sprawdź konsolę — czerwony komunikat zatrzymuje cały dalszy skrypt.

2 Część A. Funkcje liczące (lista wyniki-funkcje)

1. F1. Napisz funkcję `poleProstokata(a, b)` zwracającą pole prostokąta. Wyświetl pole dla boków 4 i 7.5.
2. F2. Napisz funkcję `poleKoła(r)` korzystającą z `Math.PI`. Wyświetl pole koła o promieniu 3 zaokrąglone do 2 miejsc metodą `toFixed(2)`.
3. F3. Zapisz jako funkcję strzałkową `cenaPoRabacie(cena, procent)`. Wyświetl cenę towaru za 249.99 zł po rabacie 15% (2 miejsca po przecinku).
4. F4. Napisz funkcję `brutto(netto, stawka = 23)` z parametrem domyślnym. Wyświetl cenę brutto dla 100 zł netto przy wywołaniu z jednym argumentem oraz ze stawką 8%.
5. F5. Napisz funkcję `kwotaVat(netto, stawka = 23)`, która **wywołuje** `brutto()` i zwraca różnicę brutto – netto. Wyświetl VAT od 350 zł netto.
6. F6. W funkcji `test()` zadeklaruj `const wewnatrz = "..."` i zwróć tę stałą. Wyświetl wynik `test()`, a następnie poza funkcją dopisz `console.log(wewnatrz);`. Zanotuj komunikat z konsoli, sprawdź, czy wyświetliły się dalsze wyniki, i zamień tę linię na komentarz.

3 Część B. Tablica ocen (lista wyniki-oceny)

Pracuj na tablicy `oceny` z pliku startowego. Nie wpisuj wyników ręcznie — mają być liczone przez skrypt, tak by po zmianie tablicy zmieniły się same.

1. B1. Wyświetl wszystkie oceny oddzielone przecinkami (`join`) i ich liczbę (`length`).
2. B2. Napisz funkcję `srednia(tab)`, która pętlą `for...of` sumuje elementy i zwraca średnią. Wyświetl średnią ocen z 2 miejscami po przecinku.
3. B3. Wyświetl najniższą i najwyższą ocenę (`Math.min`, `Math.max` z operatorem `...` albo pętlą).
4. B4. Wyświetl oceny posortowane rosnąco, a pod spodem tablicę oryginalną. Oryginał ma zostać bez zmian — sortuj kopię `[...oceny]`.
5. B5. Metodą `filter` utwórz tablicę ocen co najmniej dobrych (≥ 4). Wyświetl ją i liczbę jej elementów.
6. B6. Napisz funkcję `slovnio(o)` z instrukcją `switch` (6 $\rightarrow$ „celujący” ... 1 $\rightarrow$ „niedostateczny”) i metodą `map` wyświetl wszystkie oceny słownie.
7. B7. Dodaj na koniec tablicy ocenę 6 (`push`) i wyświetl nową liczbę ocen i średnią, korzystając z funkcji z B2. Następnie usuń ostatni element (`pop`) i wyświetl zwróconą wartość.

4 Część C. Punkty z testu (lista wyniki-punkty)

Tablica `punkty` zawiera wyniki testu, w którym można było zdobyć 120 punktów.

1. C1. Wyświetl wynik `[...punkty].sort()` bez argumentów, potem posortowane rosnąco funkcją porównującą `(a, b) => a - b`, a na końcu malejąco. Porównaj pierwszy wiersz z drugim.
2. C2. Metodą `map` utwórz tablicę `procenty` (`punkty / 120 * 100`, zaokrąglone `Math.round`) i wyświetl ją ze znakiem %.
3. C3. Metodą `filter` policz, ilu uczniów zaliczyło test (co najmniej 50%). Wyświetl np. „Zaliczyło: 5 z 9”.
4. C4. Metodą `forEach` z dwoma parametrami (`p, i`) wyświetl każdy wynik w osobnym wierszu: „Uczeń 1: 87 pkt (73%)”.
5. Zapisz pliki, zrób zrzut całej strony oraz zrzut konsoli z komunikatem z F6. Sprawdź, czy w konsoli nie ma żadnych błędów.

W sprawozdaniu umieść

- ☐ tabela F1–F6, B1–B7 i C1–C4 z fragmentem kodu i wynikiem wyświetlonym na stronie
- ☐ zrzut całej strony z wynikami i zrzut konsoli z komunikatem z F6
- ☐ wyjaśnienie różnicy między `sort()` a `sort((a, b) => a - b)` na przykładzie z C1

☐ folder `zad23` z plikami `index.html`, `styl.css`, `skrypt.js`

☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się funkcja, która wyświetla wynik, od funkcji, która go zwraca (`return`)? Która jest bardziej uniwersalna i dlaczego?
2. Co się stanie, gdy wywołasz `brutto(100)` bez drugiego argumentu, a co, gdyby funkcja nie miała wartości domyślnej?
3. Dlaczego `[87, 9, 100].sort()` zwraca `[100, 87, 9]`? Jak działa funkcja porównująca `(a, b) => a - b`?
4. Czym różnią się metody `map` i `filter` pod względem długości tablicy wynikowej?

Kryteria oceny

Kryterium	Punkty
F1-F5 — poprawne funkcje z parametrami i return, toFixed, funkcja strzałkowa, wartość domyślna	5
F6 — test zasięgu z zanotowanym komunikatem błędu	1
B1-B7 — wyniki liczone przez skrypt, oryginał tablicy niezmieniony po sortowaniu	6
C1-C4 — sortowanie liczbowe, map, filter, forEach z indeksem	4
Kod czytelny (komentarze z numerami poleceń, wcięcia), brak błędów w konsoli, rzuty	2
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

JavaScript — obiekty i tablice obiektów

Czas: 2 × 45 min

praca indywidualna

Cel: Przetwarzanie tablicy obiektów: wyświetlanie danych w tabeli HTML, obliczanie statystyk, sortowanie po różnych polach, filtrowanie na żywo oraz użycie obiektów wbudowanych `Math` i `Date`.

Po zajęciach potrafisz:

- tworzy obiekt literalny z właściwościami i metodą oraz odczytuje i dodaje właściwości
- generuje wiersze tabeli HTML z tablicy obiektów pętlą
- sortuje tablicę obiektów po polu liczbowym i tekstowym (`localeCompare`)
- filtruje dane według tekstu wpisanego przez użytkownika bez rozróżniania wielkości liter
- losuje element tablicy (`Math.random`) i oblicza liczbę dni między datami (`Date`)

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_24_start` z Pliki_do_zadan (`index.html` , `styl.css` , `dane.js` , `skrypt.js`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server code.visualstudio.com
- **Przeglądarka z narzędziami F12** — Console (`console.table`), Elements — podgląd wygenerowanych wierszy

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 1.16 obiekty literalne, tablice obiektów, `sort` z funkcją porównującą, `Math` i `Date`.
- Skopiuj folder `r1_24_start` do swojego folderu `nazwisko_imie` jako `zad24`.

Bezpieczeństwo

- Teksty wstawiaj do strony przez `textContent`, a nie `innerHTML` — to dobry nawyk, który w zadaniu 25 uzasadnimy dokładniej.
- Nie zmieniaj danych w `dane.js` — wyniki porównujesz z rozwiązaniem nauczyciela.

Przebieg zadania

1 Obiekt i dane w konsoli

Plik `dane.js` wczytuje się przed `skrypt.js` i tworzy tablicę `produkty`. Każdy element ma pola `id`, `nazwa`, `kategoria`, `cena`, `ilosc`. Przyciski i pole wyszukiwania są już podłączone do pustych funkcji na końcu `skrypt.js`.

1. Uruchom stronę przez Live Server i otwórz konsolę F12. Wpisz w swoim komentarzu imię i nazwisko.
2. O1. Utwórz obiekt `sklep` z właściwościami `nazwa`, `miasto`, `godziny` i metodą `opis()` zwracającą jeden tekst z tymi danymi (`this`). Po utworzeniu dodaj właściwość `telefon`. Wyświetl opis i telefon w akapicie `#info-sklep`.
3. O2. Wyświetl tablicę w konsoli poleceniem `console.table(produkty)`, a w `console.log` nazwę pierwszego produktu i liczbę produktów. Zrób zrzut konsoli.

2 Tabela i statystyki

Komórkę tabeli tworzysz tak (powtórz w pętli dla wszystkich pól):

```
const td = document.createElement("td");
td.textContent = p.nazwa;
tr.append(td);
```

1. O3. Uzupełnij funkcję `renderujTabele(lista)`: wyczyść `#tabela-produkty` (`tbody.textContent = ""`), a następnie dla każdego produktu dodaj wiersz z nazwą, kategorią, ceną z 2 miejscami i „zł” (komórka z klasą `cena`) oraz ilością („4 szt.” albo „brak”). Wiersz produktu z ilością 0 ma dostać klasę `brak`. Na końcu wpisz do `#znaleziono` tekst „Wyświetlono: X z 14”.
2. O4. Napisz funkcję `statystyki()`, która dopisuje do listy `#statystyki`: liczbę produktów, najtańszy i najdroższy produkt (nazwa i cena), średnią cenę, wartość magazynu (suma `cena × ilość`) i nazwy produktów niedostępnych (`filter` + `map` + `join`). Wywołaj ją po załadowaniu.

3 Sortowanie i wyszukiwanie

Zadeklaruj zmienną `let widoczne = produkty`; przechowującą aktualnie wyświetloną listę. Sortuj zawsze kopię (`[...widoczne]`), aby tablica `produkty` zachowała pierwotną kolejność.

1. O5. `sortujPoCenie()` — sortuje widoczne produkty rosnąco po cenie i ponownie rysuje tabelę.
2. O6. `sortujPoNazwie()` — sortuje alfabetycznie metodą `a.nazwa.localeCompare(b.nazwa, "pl")`. Sprawdź, gdzie trafił „łańcuch 8-rzędowy”, a potem sprawdź (np. w konsoli), gdzie trafiłby przy porównaniu operatorem `<`.
3. O7. `szukaj()` — pokazuje tylko produkty, których nazwa zawiera wpisany tekst, bez względu na wielkość liter (`toLowerCase`, `includes`). Sprawdź: „KASK” → 1 wynik, „ro” → 3 wyniki. Puste pole ma pokazać wszystkie produkty. Po wyszukaniu kliknij „Sortuj po cenie” — sortuje się tylko wynik wyszukiwania.

4 Math i Date

Pamiętaj: w `new Date(rok, miesiąc, dzień)` miesiące liczy się od 0 (styczeń = 0). Różnica dwóch dat to liczba milisekund.

1. O8. `losujProduktDnia()` — losuje (`Math.floor(Math.random() * n)`) jeden produkt **dostępny** (ilość `> 0`) i wyświetla w `#produkt-dnia` nazwę, cenę po rabacie 10% i cenę pierwotną. Kliknij kilka razy i sprawdź, czy nigdy nie wypada produkt z ilością 0.
2. O9. Napisz funkcję `odliczanie()`, która w `#odliczanie` wyświetla dzisiejszą datę (`toLocaleDateString("pl-PL")`) i liczbę dni do rajdu 26 czerwca 2027 r. (`Math.ceil`). Gdy data minęła, ma się pojawić komunikat „Rajd już się odbył”. Sprawdź tę gałąź, zmieniając tymczasowo rok na 2025.
3. Sprawdź, czy konsola nie pokazuje błędów. Zrób zrzuty: strona po załadowaniu, wynik wyszukiwania „ro” posortowany po cenie, tabela posortowana po nazwie.

W sprawozdaniu umieść

- ☐ tabela O1–O9 z fragmentami kodu i wynikami
- ☐ zrzuty: konsola z `console.table`, strona po załadowaniu, wyszukiwanie „ro” po sortowaniu, sortowanie po nazwie
- ☐ porównanie sortowania z `localeCompare` i operatorem `<` (pozycja „łańcucha”)
- ☐ folder `zad24` z wszystkimi plikami
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Jak odczytać cenę trzeciego produktu z tablicy `produkty`? Podaj dwa sposoby zapisu dostępu do właściwości.
2. Dlaczego sortujemy kopię `[...widoczne]`, a nie tablicę `produkty`? Co by się stało z wyszukiwarką, gdyby oryginał był sortowany?
3. Co zwraca `Math.random()` i dlaczego do wylosowania indeksu potrzebne są `Math.floor` i mnożenie przez długość tablicy?
4. Dlaczego `new Date(2027, 5, 26)` oznacza czerwiec, a nie maj?

Kryteria oceny

Kryterium	Punkty
O1–O2 — obiekt z metodą i dodaną właściwością, dane w konsoli	2
O3 — tabela 14 wierszy generowana skryptem, ceny z „zł”, wiersze „brak” wyróżnione klasą	4
O4 — 6 poprawnych statystyk liczonych z danych	3
O5–O7 — sortowanie po cenie i nazwie (polskie znaki), wyszukiwarka bez względu na wielkość liter	5
O8–O9 — losowanie tylko dostępnych produktów, poprawna liczba dni i obsługa daty minionej	3
Zrzuty, brak błędów w konsoli, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

DOM — modyfikowanie strony

Czas: 2 × 45 min

praca indywidualna

Cel: Zmienianie treści, klas, atrybutów i stylów elementów strony w reakcji na kliknięcia oraz bezpieczne wstawianie tekstu wpisanego przez użytkownika.

Po zajęciach potrafisz:

- wybiera elementy metodami `querySelector`, `querySelectorAll` i `getElementById`
- zmienia tekst (`textContent`), atrybuty (`src`, `alt`, `data-*`) i style elementów
- przełącza klasy CSS metodami `classList.toggle`, `add`, `remove`, `contains`
- tworzy nowe elementy (`createElement`, `append`) i aktualizuje licznik na żywo
- wyjaśnia, dlaczego dane od użytkownika wstawia się przez `textContent`, a nie `innerHTML`

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_25_start` z Pliki_do_zadan (`index.html`, `styl.css`, pusty `skrypt.js`, folder `img`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server
code.visualstudio.com
- **Przeglądarka z narzędziami F12** — Elements — obserwuj zmiany klas i atrybutów na żywo; Application → Local Storage

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 1.17 metody wyboru elementów, `classList` i różnicę między `textContent` a `innerHTML`.
- Skopiuj folder `r1_25_start` jako `zad25` do swojego folderu `nazwisko_imie`.

Bezpieczeństwo

- Tekst wpisany przez użytkownika wstawiaj wyłącznie przez `textContent`. W D9 porównujesz obie metody tylko na zwykłym znaczniku formatowania `<b>` — nie testuj innych znaczników.
- Nie zmieniaj pliku CSS — wszystkie efekty uzyskujesz, dodając i usuwając gotowe klasy.

Przebieg zadania

1 Rozpoznanie strony

Cały kod piszesz w `skrypt.js` (dołączonym z `defer`). Obejrzyj w `styl.css` klasy, które będziesz przełączać: `ukryty`, `ciemny` (na body), `otwarte` (na `.pytanie`), `aktywna` (na miniaturze), `mało` (na liczniku).

```
const btn = document.querySelector("#btn-godziny");
btn.addEventListener("click", () => {
  // co ma się stać po kliknięciu
});
```

1. Uruchom stronę przez Live Server, otwórz F12 → Elements. Wpisz w komentarzu `skrypt.js` swoje imię i nazwisko.
2. W konsoli wypróbuj: `document.querySelector("h1").textContent = "Test"` oraz `document.body.classList.add("ciemny")`. Odśwież stronę — zmiany z konsoli znikają, a te zapisane w skrypcie zostaną.

2 Tekst, klasy i style

1. D1. Przycisk „Pokaż godziny otwarcia” pokazuje i ukrywa akapit `#godziny` (klasa `ukryty`). Tekst przycisku zmienia się na „Ukryj godziny otwarcia” i z powrotem.
2. D2. Przyciski A+ i A- zmieniają `document.body.style.fontSize` o 2 px, w zakresie od 12 do 24 px (zaczynij od 16). Sprawdź w Elements, że pojawia się atrybut `style` na body.
3. D3. Przycisk „Tryb ciemny” przełącza klasę `ciemny` na body, a jego napis zmienia się na „Tryb jasny”. Dodatkowo zapamiętaj wybór w `localStorage` (klucz `tryb`) i odtwórz go po odświeżeniu strony; sprawdź wpis w F12 → Application → Local Storage.

3 FAQ i galeria — wiele elementów naraz

`querySelectorAll` zwraca listę elementów — do każdego dodaj obsługę zdarzenia w pętli `forEach`.

1. D4. Kliknięcie pytania w FAQ przełącza klasę `otwarte` na jego rodzicu (`.pytanie`, właściwość `parentElement`) i ustawia atrybut `aria-expanded` na `true` lub `false`. Pozostałe pytania mają się nie zmieniać.
2. D5. Kliknięcie miniatury w galerii podmienia `src` dużego zdjęcia na wartość z atrybutu `data-duze` (`element.dataset.duze`), przepisuje `alt` i ustawia podpis `#podpis`. Klasa `aktywna` ma być tylko na klikniętej miniaturze.

4 Księga gości — licznik, podgląd i nowe elementy

1. D6. Licznik pod polem wpisu pokazuje na żywo (zdarzenie `input`) „Pozostało: N znaków” (limit 200). Gdy zostanie mniej niż 20 znaków, licznik dostaje klasę `mało`.
2. D7. Przycisk „Podgląd” wyświetla w `#podglad` tekst „Imię: treść wpisu” przez `textContent`. Gdy imię jest puste, użyj słowa „Gość”.
3. D8. Przycisk „Dodaj wpis” tworzy nowy element `li` (imię pogrubione w elemencie `strong`, potem treść) i wstawia go na początek listy `#lista-wpisow` (`prepend`). Zaktualizuj liczbę wpisów w `#liczba-wpisow`, wyczyść pole i licznik. Pusty wpis ma nie zostać dodany — wyświetl komunikat w `#podglad`.
4. D9. Eksperyment: wpisz w polu tekst `Piękne miejsce, <b>połączam</b>!` i kliknij Podgląd. Następnie tymczasowo zamień w D7 `textContent` na `innerHTML`, powtórz próbę i porównaj wynik oraz kod w Elements. Przywróć `textContent`.
5. Sprawdź konsolę (brak błędów) i zrób zrzuty: strona w trybie ciemnym z otwartym jednym pytaniem i inną miniaturą, księga gości z 2 wpisami, wyniki D9 dla obu metod.

W sprawozdaniu umieść

- ☐ tabela D1–D9 z fragmentami kodu i opisem działania
- ☐ zrzuty: tryb ciemny z otwartym pytaniem i zmienionym zdjęciem, księga gości z 2 wpisami, D9 dla `textContent` i `innerHTML`, Local Storage z kluczem `tryb`
- ☐ wniosek z eksperymentu D9
- ☐ folder `zad25` z plikami strony
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się `querySelector` od `querySelectorAll`? Co zwraca każda z nich, gdy nie znajdzie elementu?
2. Dlaczego lepiej przełączać klasę CSS (`classList.toggle`) niż ustawiać wiele właściwości `element.style` w skrypcie?
3. Co pokazał eksperyment D9? Dlaczego tekst od użytkownika wstawiamy przez `textContent`?
4. Do czego służą atrybuty `data-*` i jak odczytać je w JavaScript?

Kryteria oceny

Kryterium	Punkty
D1–D2 — pokazywanie godzin ze zmianą napisu, rozmiar czcionki z ograniczeniem 12–24 px	3
D3 — tryb ciemny przełączany klasą, zapamiętany po odświeżeniu	3
D4–D5 — FAQ niezależne dla każdego pytania z <code>aria-expanded</code> , galeria podmienia <code>src</code> , <code>alt</code> , podpis i klasę aktywna	5
D6–D8 — licznik z klasą mało, podgląd przez <code>textContent</code> , dodawanie wpisów z licznikiem i blokadą pustego wpisu	5
D9 i zrzuty — poprawny wniosek, kod bez błędów w konsoli	2
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik `.docx` o tym samym numerze).

Zdarzenia i formularze w JavaScript

Czas: 2 × 45 min

praca indywidualna

Cel: Obsługa zdarzeń `click`, `input`, `change` i `submit`: przeliczanie zamówienia na żywo oraz walidacja formularza w JavaScript z czytelnymi komunikatami pod polami.

Po zajęciach potrafisz:

- podłącza obsługę zdarzeń metodą `addEventListener` do pojedynczych pól i grup elementów
- dobiera zdarzenie do kontrolki (`input` dla pól liczbowych i tekstowych, `change` dla checkboxów, radio i list)
- odczytuje wartości pól, stan `checked` i atrybuty `data-*` oraz zamienia tekst na liczbę
- zatrzymuje domyślne wystąpienie formularza (`preventDefault`) i waliduje dane, wyświetlając komunikaty pod polami
- wyjaśnia, dlaczego walidacja w przeglądarce nie zastępuje sprawdzania danych na serwerze

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_26_start` z Pliki_do_zadan (`index.html`, `styl.css`, pusty `skrypt.js`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server
code.visualstudio.com
- **Przeglądarka z narzędziami F12** — Elements → panel Event Listeners; Console

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 1.17 `addEventListener`, obiekt zdarzenia i `event.preventDefault()`; z zadania 6 — atrybuty walidacji HTML.
- Skopiuj folder `r1_26_start` jako `zad26` do swojego folderu `nazwisko_imie`.

Bezpieczeństwo

- Do testów używaj zmyślonych danych (np. adres w domenie `example.com`, telefon 600 100 200) — nie wpisuj prawdziwych danych swoich ani kolegów.
- Pamiętaj: walidacja w JavaScript ułatwia życie użytkownikowi, ale nie chroni serwera — dane i tak trzeba sprawdzić w PHP (rok 2).

Przebieg zadania

1 Podsumowanie liczone na żywo

Każde pole ilości ma klasę `ilosc` oraz atrybuty `data-nazwa` i `data-cena`; checkboxy dodatków — klasę `dodatek`. Wartości pól są tekstami — przed liczeniem zamień je funkcją `Number()`.

```
const ilosci = document.querySelectorAll(".ilosc");
function przelicz() {
  let suma = 0;
  ilosci.forEach(pole => {
    const ile = Number(pole.value);
    // ...
  });
}
ilosci.forEach(p => p.addEventListener("input", przelicz));
```

1. Uruchom stronę przez Live Server, wpisz w komentarzu `skrypt.js` swoje imię i nazwisko.
2. Z1. Napisz funkcję `przelicz()`, która czyści listę `#lista-pozycji` i dla każdej pozycji z ilością większą od 0 dopisuje wiersz „Pierogi ruskie × 2 = 48.00 zł”, a wartości sumuje. Podłącz ją do zdarzenia `input` wszystkich pól ilości.
3. Z2. Dolicz zaznaczone dodatki (zdarzenie `change`, właściwość `checked`) i dopisz je do listy. Gdy nic nie wybrano, lista ma pokazywać „Koszyk jest pusty”.
4. Z3. Obsłuż wybór odbioru (radio `name="dostawa"`, selektor `input[name=dostawa]:checked`): dostawa kosztuje 9 zł, a od 100 zł zamówienia jest gratis — wtedy w `#koszt-dostawy` wyświetl słowo „gratis”.
5. Z4. Wyświetl łączną kwotę w `#suma` z 2 miejscami po przecinku. Sprawdź: 2 × ruskie, 1 × jagody, śmietana, kompot → 81.00 zł; z dostawą 90.00 zł; po dodaniu 1 × z mięsem → 108.00 zł i dostawa „gratis”.

2 Walidacja przy wysłaniu

Pod każdym polem jest element `span.blad` o id `blad-<id pola>`; klasa `niepoprawne` obramowuje pole na czerwono.

1. Z5. Kliknij „Zamawiam” bez wypełniania danych i zobacz, co się dzieje z adresem strony. Dodaj do znacznika `form` atrybut `novalidate` (walidację przejmie skrypt), a w obsłudze zdarzenia `submit` wywołaj `event.preventDefault()`.
2. Z6. Napisz funkcję `pokazBlad(id, tekst)`, która wpisuje komunikat do `#blad-<id>` i dodaje polu klasę `niepoprawne` (pusty tekst — usuwa ją). Zwraca `true`, gdy pole jest poprawne.
3. Z7. W funkcji `waliduj()` sprawdź: imię i nazwisko co najmniej 3 znaki (po `trim`), telefon dokładnie 9 cyfr (spacje usuń, wzorzec `/^\d{9}$/` i metoda `test`), e-mail ze znakiem `@` i kropką w domenie, wybraną godzinę, zaznaczoną zgodę. Wyświetl **wszystkie** błędy naraz, a nie tylko pierwszy.
4. Z8. Gdy formularz jest niepoprawny lub nie zamówiono żadnych pierogów, wyświetl w `#komunikat` odpowiednią informację i ustaw kursor (`focus()`) w pierwszym błędnym polu. Gdy wszystko jest dobrze — komunikat „Dziękujemy, Anna Nowak! Zamówienie na kwotę 108.00 zł będzie gotowe o 13:00.” (klasa `ok`) i zablokuj przycisk (`disabled`).
5. Z9. Dodaj do pól imienia, telefonu i e-maila obsługę zdarzenia `input`, która od razu usuwa komunikat błędu, gdy użytkownik zaczyna poprawiać pole.

3 Testy i dokumentacja

1. Z10. Przetestuj formularz według tabeli w sprawozdaniu: puste pola, imię „Al”, telefon „12 345 678”, e-mail bez kropki w domenie, brak godziny, brak zgody, zamówienie bez pierogów, zamówienie poprawne.
2. Z11. W F12 → Elements zaznacz przycisk „Zamawiam” lub formularz i obejrzyj panel Event Listeners. Zrób zrzut.
3. Zrób zrzuty: podsumowanie z kwotą 108.00 zł i dostawą gratis, formularz z wszystkimi pięcioma komunikatami błędów, komunikat o przyjęciu zamówienia.

W sprawozdaniu umieść

- ☐ tabela Z1–Z11 z fragmentami kodu i wynikami
- ☐ tabela testów z Z10 (dane wejściowe, oczekiwany i otrzymany wynik)
- ☐ zrzuty: podsumowanie 108.00 zł, pięć komunikatów błędów, zamówienie przyjęte, panel Event Listeners
- ☐ folder `zad26` z plikami strony
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różnią się zdarzenia `input` i `change`? Które wybrać dla pola liczbowego, a które dla checkboxa?
2. Co robi `event.preventDefault()` w obsłudze `submit` i co by się stało bez niego?
3. Dlaczego `pole.value` trzeba zamienić funkcją `Number()` przed dodawaniem? Co da `"2" + 1`?
4. Dlaczego dane z formularza trzeba ponownie sprawdzić na serwerze, mimo walidacji w JavaScript?

Kryteria oceny

Kryterium	Punkty
Z1–Z2 — pozycje i dodatki w podsumowaniu, liczby z data-cena, „Koszyk jest pusty”	4
Z3–Z4 — koszt dostawy z progiem 100 zł, poprawne kwoty kontrolne 81 / 90 / 108 zł	3
Z5–Z7 — <code>preventDefault</code> , funkcja <code>pokazBlad</code> , 5 reguł walidacji, wszystkie błędy naraz	5
Z8–Z9 — komunikaty końcowe, focus na pierwszym błędzie, blokada przycisku, czyszczenie błędu przy poprawianiu	3
Z10–Z11 — tabela testów i zrzuty	3
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Mini-aplikacja JS: quiz lub lista zakupów

Czas: 2 × 45 min

praca indywidualna (wariant do wyboru)

Cel: Zbudowanie małej aplikacji jednostronicowej, w której stan przechowywany jest w tablicy obiektów, a widok odtwarza jedna funkcja renderująca wywoływana po każdej zmianie danych.

Po zajęciach potrafisz:

- projektuje dane aplikacji jako tablicę obiektów i zmienne stanu
- pisze funkcję renderującą, która buduje widok z danych przez `createElement` i `textContent`
- obsługuje zdarzenia kliknięcia i formularza, zmieniając najpierw dane, a potem widok
- stosuje `filter`, `find` i `forEach` do operacji na danych aplikacji
- testuje aplikację według przygotowanych scenariuszy i dokumentuje wyniki

Sprzęt i materiały

- komputer z Windows 11 i przeglądarką Chrome, Edge lub Firefox
- folder `r1_27_start` z Pliki_do_zadan (podfoldery `quiz` i `lista` z gotowym HTML i CSS)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server
code.visualstudio.com
- **Przeglądarka z narzędziami F12** — Console, Elements, Application → Local Storage (wariant B)

Przygotowanie przed zajęciami

- Powtórz zadania 24–26: tablice obiektów, `createElement`, `addEventListener`, `preventDefault`.
- Wybierz wariant: A — quiz z punktacją, B — lista zakupów. Skopiuj odpowiedni podfolder jako `zad27` do folderu `nazwisko_imie`.

Bezpieczeństwo

- Teksty z danych i od użytkownika wstawiaj przez `textContent` — w quizie odpowiedzi zawierają znaczniki (np. `<img>`), które mają się wyświetlić jako tekst.
- W `localStorage` zapisuj tylko dane aplikacji; nie przechowuj tam haseł ani danych osobowych.

Przebieg zadania

1 Zasada działania (oba warianty)

Zdarzenie → zmiana danych w tablicy → wywołanie funkcji renderującej. Nie zmieniamy ręcznie pojedynczych elementów listy — po każdej zmianie odtworz widok z danych.

```
let dane = [ /* obiekty */ ];
function render() {
  kontener.textContent = "";          // 1. wyczyść widok
  dane.forEach(d => { /* 2. utwórz element z danych */ });
}
przycisk.addEventListener("click", () => {
  /* 3. zmień dane */
  render();                          // 4. odśwież widok
});
```

1. Otwórz wybrany podfolder w VS Code i uruchom Live Server. Obejrzyj HTML i CSS — wypisz w sprawozdaniu identyfikatory i klasy, których użyjesz (np. `#odpowiedzi`, `.dobra`, `.zla` albo `#lista`, `.kupione`, `.aktywny`).
2. Wpisz w komentarzu `skrypt.js` imię, nazwisko i wybrany wariant.

2 Wariant A — quiz „Jak działa WWW?”

Pytanie to obiekt `{ tresc, odpowiedzi: [4 teksty], poprawna: indeks }`.

1. Q1. Utwórz tablicę `pytania` z co najmniej 6 pytaniami o HTML, CSS, JavaScript i sieci (własne, z jedną poprawną odpowiedzią). Co najmniej jedna odpowiedź ma zawierać znacznik HTML, np. `<img>`.
2. Q2. Zadeklaruj zmienne stanu: bieżący numer pytania, liczba punktów, historia odpowiedzi.
3. Q3. Napisz `pokazPytanie()`: tekst „Pytanie 2 z 6”, szerokość paska `#pasek-wypelnienie` w %, treść pytania i 4 przyciski `.odp` utworzone w pętli.
4. Q4. Po kliknięciu odpowiedzi: zablokuj wszystkie przyciski (jedna próba), zaznacz poprawną klasą `dobra`, a błędnie wybraną klasą `zla`, wypisz komunikat w `#informacja`, dolicz punkt i odblokuj „Dalej”.
5. Q5–Q6. „Dalej” przechodzi do kolejnego pytania; po ostatnim pokaż `#ekran-wynik` (atrybut `hidden`) z wynikiem „4 / 6 (67%)”, oceną słowną (100% / co najmniej 50% / poniżej 50%) i listą pytań z oznaczeniem ✓/✗.
6. Q7–Q8. „Zagraj ponownie” zeruje stan i zaczyna od nowa. Pytania mają być za każdym razem w innej kolejności — napisz funkcję `potasuj(tab)` (algorytm Fishera–Yatesa z prezentacji lub własny), która zwraca przetasowaną kopię.

3 Wariant B — lista zakupów

Produkt to obiekt `{ id, nazwa, ilosc, kupione }`. Jako `id` możesz użyć `Date.now()`.

1. L1. Utwórz tablicę `produkty` z 3 przykładowymi pozycjami (jedna kupiona) i zmienną `filtr`.
2. L2. Napisz `render()`: dla każdego produktu element `li` z checkboxem, tekstem „Chleb × 1” i przyciskiem × (`.usun`); kupione mają klasę `kupione`. Pod listą licznik „Do kupienia: 2 z 3”.
3. L3. Formularz dodaje produkt (`submit` + `preventDefault`). Walidacja: nazwa niepusta, ilość całkowita 1–99 — komunikat w `#blad`. Po dodaniu wyczyść pole i ustaw w nim kursor.
4. L4–L5. Odhaczanie i usuwanie obsłuż **jednym** zdarzeniem `click` na `#lista` (delegowanie: `e.target.closest("li")`, `e.target.matches(...)`) — zmień dane i wywołaj `render()`.
5. L6–L7. Przycisk „Usuń kupione” (`filter`). Dodanie produktu o tej samej nazwie (bez względu na wielkość liter) zwiększa ilość zamiast tworzyć duplikat (`find`).
6. L8. Filtry Wszystkie / Do kupienia / Kupione (klasa `aktywny` na wybranym) oraz zapis listy w `localStorage` (`JSON.stringify` / `JSON.parse`) — po odświeżeniu lista ma zostać.

4 Testy i dokumentacja

1. T1. Wykonaj co najmniej 6 testów swojego wariantu i zapisz je w tabeli. Quiz: odpowiedź dobra, zła, próba drugiego kliknięcia, wynik końcowy, ponowna gra (inna kolejność), odpowiedź ze znacznikiem `<img>` wyświetlona jako tekst. Lista: pusta nazwa, ilość 150, dodanie „chleb” przy istniejącym „Chleb”, odhaczenie, filtr, odświeżenie strony.
2. T2. Sprawdź, czy konsola nie pokazuje błędów, zrób 3 zrzuty działającej aplikacji i — jeśli pracujesz z Git — zatwierdź gotową wersję (commit).

W sprawozdaniu umieść

- ☐ wybrany wariant i lista użytych identyfikatorów i klas
- ☐ tabela wykonanych poleceń (Q1–Q8 albo L1–L8) z fragmentami kodu
- ☐ tabela testów T1 (co najmniej 6 testów)
- ☐ 3 zrzuty ekranu działającej aplikacji
- ☐ folder `zad27` z plikami aplikacji
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego po każdej zmianie danych odtwarzamy cały widok funkcją `render()`, zamiast poprawiać pojedyncze elementy na stronie?
2. Quiz: dlaczego tasujemy kopię tablicy `pytania`, a nie oryginał? Lista: na czym polega delegowanie zdarzeń i jaką ma przewagę nad dodawaniem obsługi do każdego przycisku ×?
3. Dlaczego do `localStorage` zapisujemy `JSON.stringify(dane)`, a nie samą tablicę? Co zwróci `getItem`, gdy klucza nie ma?
4. Gdzie w Twojej aplikacji `textContent` chroni przed błędnym wyświetleniem danych?

Kryteria oceny

Kryterium	Punkty
Dane w tablicy obiektów (min. 6 pytań albo 3 produkty) i zmienne stanu	2
Funkcja renderująca budująca widok z danych (createElement, textContent), licznik/postęp	4
Podstawowa obsługa zdarzeń: Q3–Q5 albo L3–L5 działa bez błędów	4
Funkcje rozszerzone: Q6–Q8 albo L6–L8	4
Tabela testów T1 i zrzuty, konsola bez błędów	3
Czytelny kod z komentarzami i odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Git i GitHub — wersjonowanie i GitHub Pages

Czas: 2 × 45 min

praca indywidualna

Cel: Prowadzenie historii zmian projektu w Git (commit, gałąź, scalenie z rozwiązaniem konfliktu), wysłanie repozytorium na GitHub i opublikowanie strony przez GitHub Pages.

Po zajęciach potrafisz:

- konfiguruje Git i zakłada repozytorium z plikiem `.gitignore` wykluczającym pliki prywatne i robocze
- zatwierdza zmiany z opisowymi komunikatami i czyta historię (`log`, `diff`, `status`)
- tworzy gałąź, scala ją z główną i samodzielnie rozwiązuje konflikt
- łączy repozytorium lokalne z GitHubem (`remote`, `push`, `pull`)
- publikuje stronę statyczną przez GitHub Pages i sprawdza działanie ścieżek względnych

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- projekt strony wizytówki z zadań 18–19 albo folder `r1_28_projekt` z Pliki_do_zadan
- konto na GitHub (założone w domu lub na lekcji; wymagany wiek co najmniej 13 lat)

Oprogramowanie

- **Git for Windows** — Git Bash lub terminal w VS Code; zawiera Git Credential Manager (logowanie przez przeglądarkę) git-scm.com/downloads
- **GitHub** — repozytorium publiczne i GitHub Pages (bezpłatnie dla repozytoriów publicznych) github.com
- **Visual Studio Code** — panel Source Control pokazuje te same zmiany co `git status` code.visualstudio.com

Przygotowanie przed zajęciami

- Załóż konto na GitHub i włącz uwierzytelnianie dwuskładnikowe (GitHub wymaga go od większości kont).
- Sprawdź w GitHub → Settings → Emails swój adres prywatny typu `...@users.noreply.github.com` — użyjesz go w konfiguracji Git.

Bezpieczeństwo

- Repozytorium publiczne widzi każdy. Nie wysyłaj haseł, notatek prywatnych ani danych osobowych — dlatego plik `notatki_prywatne.txt` musi trafić do `.gitignore` przed pierwszym commitem.
- Na komputerze szkolnym po zajęciach wyloguj się z GitHub w przeglądarce i usuń zapisane poświadczenia (Menedżer poświadczeń Windows → Poświadczenia systemu Windows → wpisz `git:https://github.com`).
- Nie używaj prawdziwego adresu e-mail w `user.email`, jeśli nie chcesz go publikować w historii commitów — użyj adresu noreply.

Przebieg zadania

1 Repozytorium lokalne

Pracuj w terminalu (Git Bash lub Terminal w VS Code) w folderze projektu `zad28`. Po każdym poleceniu czytaj komunikat Git.

```
git config --global user.name "Imię Nazwisko"
git config --global user.email "login@users.noreply.github.com"
git config --global init.defaultBranch main
git init
git status
```

1. G1. Skopiuj swój projekt z zadań 18–19 (lub `r1_28_projekt`) do folderu `nazwisko_imie/zad28`. Skonfiguruj Git jak wyżej i wyświetl ustawienia `git config --list --global`.
2. G2. Załóż repozytorium (`git init`) i wyświetl `git status`. Zapisz, które pliki są „Untracked”.
3. G3. Utwórz plik `.gitignore` wykluczający: `notatki_prywatne.txt`, wszystkie pliki `*.bak`, folder `.vscode/` oraz pliki systemowe `Thumbs.db` i `desktop.ini`. Sprawdź `git status --short` — wykluczonych plików ma już nie być na liście.
4. G4. Dodaj pliki (`git add .`) i zatwierdź: `git commit -m "Pierwsza wersja strony wizytówki"`. Wyświetl `git log --oneline`.
5. G5. Zmień jeden tekst na stronie głównej, obejrzyj zmianę poleceniem `git diff` (co oznaczają linie z `-` i `+`?) i zatwierdź ją z opisowym komunikatem.

2 Gałąź, scalenie i konflikt

Konflikt powstaje, gdy w dwóch gałęziach zmieniono tę samą linię. Git zostawia w pliku oba warianty między znacznikami `<<<<<<`, `=====` i `>>>>>>` — Ty decydujesz o wersji końcowej.

1. G6. Utwórz gałąź i przejdź na nią: `git switch -c kontakt`. Dodaj podstronę `kontakt.html` (adres fikcyjny, e-mail w domenie `example.com`), link „Kontakt” w menu strony głównej oraz link w stopce. Zatwierdź. Przełącz się na `main` i sprawdź, czy plik `kontakt.html` jest widoczny w folderze.
2. G7. Na gałęzi `main` zmień tę samą linię stopki strony głównej (np. dopisz miasto) i zatwierdź.
3. G8. Scal gałąź: `git merge kontakt`. Odczytaj komunikat o konflikcie, otwórz plik w VS Code, połącz obie zmiany w jedną stopkę, usuń znaczniki konfliktu, a potem `git add` i `git commit`. Usuń scaloną gałąź (`git branch -d kontakt`).
4. G9. Wyświetl historię jako graf: `git log --oneline --graph --all` i zrób zrzut. Poleceniem `git ls-files` sprawdź, które pliki są śledzone.

3 GitHub i GitHub Pages

Na GitHub utwórz **puste** repozytorium publiczne (New repository; bez README, .gitignore i licencji), np. `wizytowka`.

```
git remote add origin https://github.com/LOGIN/wizytowka.git
git remote -v
git push -u origin main
```

1. G10. Połącz repozytorium lokalne z GitHub i wyślij gałąź `main`. Przy pierwszym `push` zaloguj się w oknie przeglądarki. Odśwież stronę repozytorium i sprawdź, czy nie ma tam plików z `.gitignore`.
2. G11. Włącz publikację: Settings → Pages → Build and deployment → Source: Deploy from a branch → Branch: `main`, folder `/ (root)` → Save. Po 1–2 minutach otwórz adres `https://LOGIN.github.io/wizytowka/`. Sprawdź, czy wczytują się style, grafiki i wszystkie podstrony; jeśli nie — popraw ścieżki (wielkość liter, ukośniki `/`, ścieżki względne), zatwierdź i wyślij ponownie.
3. G12. Na GitHub dodaj plik `README.md` (Add file → Create new file) z nazwą projektu, autorem i adresem strony. Pobierz zmianę poleceniem `git pull` i sprawdź `git log --oneline`.
4. Zrób zrzuty: repozytorium na GitHub z listą commitów, ustawienia Pages z adresem strony, opublikowana strona. Na koniec usuń poświadczenia z komputera szkolnego (patrz Bezpieczeństwo).

W sprawozdaniu umieść

- ☐ tabela G1–G12 z poleceniami i najważniejszymi komunikatami
- ☐ zawartość pliku `.gitignore`
- ☐ fragment pliku z konfliktem (przed rozwiązaniem) i wersja końcowa
- ☐ zrzuty: `git log --oneline --graph --all`, repozytorium na GitHub, ustawienia Pages, opublikowana strona
- ☐ adres repozytorium i adres strony na GitHub Pages
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się obszar roboczy, poczekalnia (staging) i repozytorium? Które polecenia przenoszą zmiany między nimi?
2. Dlaczego `.gitignore` trzeba utworzyć przed pierwszym commitem? Co się stanie, gdy plik z hasłem trafi już do historii?
3. Kiedy powstaje konflikt przy `git merge` i jak go rozwiązać?
4. Czym różni się `git push` od `git pull`? Czym jest `origin`?

Kryteria oceny

Kryterium	Punkty
G1–G4 — konfiguracja, repozytorium, poprawny .gitignore (4 reguły), pierwszy commit bez plików prywatnych	4
G5 — git diff i commit z opisowym komunikatem	1
G6–G9 — gałąź, konflikt rozwiązany bez pozostawionych znaczników, graf historii z rozgałęzieniem	5
G10–G12 — repozytorium na GitHub, działająca strona na GitHub Pages (style, grafiki, podstrony), README pobrane przez pull	5
Sprawozdanie: zrzuty, adresy, fragment konfliktu	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Publikacja na hostingu i podstawy SEO

Czas: 2 × 45 min

praca indywidualna

Cel: Opublikowanie strony na serwerze przez SFTP/FTP, sprawdzenie HTTPS oraz przygotowanie podstawowych elementów SEO i podglądu w mediach społecznościowych, zweryfikowanych audytem Lighthouse.

Po zajęciach potrafisz:

- konfiguruje połączenie SFTP/FTP w Menedżerze witryn FileZilla i wysyła pliki do właściwego katalogu serwera
- sprawdza certyfikat HTTPS, przekierowanie z HTTP i kody odpowiedzi w zakładce Network
- pisze unikatowe `title` i `meta description` oraz znaczniki Open Graph dla każdej podstrony
- tworzy pliki `robots.txt` i `sitemap.xml` i wyjaśnia ich rolę
- przeprowadza audyt Lighthouse SEO i test strony na telefonie, a następnie poprawia wykryte problemy

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu; smartfon (własny, opcjonalnie)
- projekt strony z zadania 28 albo folder `r1_28_projekt` z Pliki_do_zadan
- dane dostępu do konta na serwerze szkolnym lub hostingu (od nauczyciela) albo GitHub Pages z zadania 28

Oprogramowanie

- **FileZilla Client** — pobieraj tylko z oficjalnej strony; wybierz instalator bez dodatkowych ofert (Show additional download options) filezilla-project.org/download.php?type=client
- **Google Chrome lub Microsoft Edge** — F12 → Lighthouse (kategoria SEO), Network, tryb urządzeń
- **Visual Studio Code** — edycja meta znaczników, `robots.txt` i `sitemap.xml` code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 1.18: hosting, domena, DNS, SFTP, HTTPS i podstawy SEO.
- Skopiuj projekt jako `nazwisko_imie/zad29`. Zapisz dane dostępu podane przez nauczyciela tylko na kartce lub w menedżerze haseł — nie w plikach projektu.

Bezpieczeństwo

- Używaj SFTP (port 22) lub FTP z szyfrowaniem TLS, jeśli serwer na to pozwala. Zwykły FTP przesyła hasło jawnym tekstem.
- Nie zapisuj hasła w FileZilla na komputerze szkolnym (typ logowania „Pytaj o hasło”), nie wysyłaj na serwer plików prywatnych (`notatki_prywatne.txt`, `.git/`, `.vscode/`).
- Wysyłaj pliki tylko do swojego katalogu na serwerze; nie usuwaj i nie nadpisuj plików innych osób.

Przebieg zadania

1 SEO na stronach (przed wysłaniem)

Adres strony (domenę) poda nauczyciel; w przykładach jest `https://twoja-domena.pl/`. Title najlepiej do ok. 60 znaków, description do ok. 155 — dłuższe wyszukiwarka zwykle przycina.

```
<title>Warsztaty ceramiczne – Glina i Ogień, Miasteczko</title>
<meta name="description" content="...">
<link rel="canonical" href="https://twoja-domena.pl/">
<meta property="og:title" content="...">
<meta property="og:description" content="...">
<meta property="og:image" content="https://twoja-domena.pl/img/og.png">
<meta property="og:url" content="https://twoja-domena.pl/">
<meta property="og:type" content="website">
```

1. S1. Każdej podstronie nadaj unikatowy `title` (treść + nazwa firmy) i `meta description` zachęcający do kliknięcia. Zapisz w tabeli długości tekstów.
2. S2. Sprawdź strukturę: jeden `h1` na stronie, logiczna kolejność nagłówków, opisowe `alt` dla grafik, atrybut `lang="pl"`, opisowe teksty linków (nie „kliknij tutaj”).
3. S3. Przygotuj grafikę podglądu 1200 × 630 px (GIMP: Obraz → Wymiary płótna lub nowy obraz, eksport PNG/JPG poniżej 300 KB) i dodaj znaczniki Open Graph na stronie głównej oraz `link rel="canonical"`.
4. S4. Utwórz w katalogu głównym projektu plik `robots.txt`: dostęp dla wszystkich robotów, zablokowany katalog `/szkice/`, wiersz `Sitemap:` z pełnym adresem mapy.
5. S5. Utwórz `sitemap.xml` (format z [sitemaps.org](http://www.sitemaps.org), przestrzeń nazw `http://www.sitemaps.org/schemas/sitemap/0.9`) z pełnymi adresami `https://` wszystkich podstron i datą `lastmod`. Otwórz plik w przeglądarce — błąd składni XML zobaczysz od razu.

2 Wysłanie plików przez FileZilla

1. P1. W FileZilla otwórz Plik → Menedżer witryn → Nowa witryna. Ustaw protokół SFTP (albo FTP z „Wymagaj jawnego FTP przez TLS”), serwer i port od nauczyciela, typ logowania „Pytaj o hasło”, użytkownika. Połącz się i zrób zrzut okna (bez hasła).
2. P2. Odczytaj w panelu zdalnym, który katalog jest publiczny (np. `public_html`, `www`, `htdocs` — według instrukcji nauczyciela). Przeciągnij do niego pliki strony z panelu lokalnego. Nie wysyłaj `.git`, `.vscode` ani plików prywatnych.
3. P3. Obejrzyj okno kolejki i komunikatów: ile plików wysłano, czy były błędy? Zmień jeden tekst lokalnie, wyślij plik ponownie i potwierdź nadpisanie.
4. Jeśli serwer nie jest dostępny, opublikuj wersję z SEO na GitHub Pages (`git commit` i `git push` z zadania 28) i wykonaj dalsze kroki na adresie `github.io`.

3 Kontrola opublikowanej strony

1. K1. Otwórz stronę przez `https://`. Kliknij ikonę obok adresu i sprawdź certyfikat: dla kogo wystawiony, przez kogo, do kiedy ważny. Wpisz adres z `http://` i sprawdź w F12 → Network, czy następuje przekierowanie (kod 301/302/308) na HTTPS.
2. K2. W zakładce Network odśwież stronę i zanotuj kody odpowiedzi dla HTML, CSS, grafik oraz `robots.txt` i `sitemap.xml`. Wpisz adres nieistniejącej podstrony i zanotuj kod.
3. K3. Uruchom F12 → Lighthouse, tryb Navigation, urządzenie Mobile, kategorie SEO i Accessibility. Zapisz wyniki, popraw co najmniej jeden zgłoszony problem, wyślij plik ponownie i powtórz audyt.
4. K4. Otwórz stronę na telefonie (lub w trybie urządzeń F12, profil telefonu) i sprawdź czytelność tekstu, menu i tabel bez przewijania w poziomie.
5. Zrób zrzuty: FileZilla po wysłaniu, certyfikat, Network z kodami, raport Lighthouse przed i po poprawce, widok na telefonie.

W sprawozdaniu umieść

- ☐ tabela S1–S5, P1–P3, K1–K4 z opisem i wynikami
- ☐ tabela title/description dla każdej podstrony z liczbą znaków
- ☐ zawartość `robots.txt` i `sitemap.xml`
- ☐ zrzuty: FileZilla, certyfikat, Network, Lighthouse przed i po, widok mobilny
- ☐ adres opublikowanej strony
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się SFTP od FTP? Dlaczego zwykły FTP nie powinien być używany w publicznej sieci?
2. Do czego służą `robots.txt` i `sitemap.xml`? Czy wpis `Disallow` chroni katalog przed dostępem osób postronnych?
3. Do czego służą znaczniki Open Graph i gdzie widać ich efekt?
4. Dlaczego `robots.txt` musi leżeć w katalogu głównym domeny? Co z tego wynika dla strony na `login.github.io/repozytorium/`?

Kryteria oceny

Kryterium	Punkty
S1–S2 — unikatowe title i description na każdej podstronie (rozsądna długość), poprawna struktura nagłówków i alt	4
S3–S5 — Open Graph z grafiką 1200 × 630, canonical, poprawny robots.txt i sitemap.xml z pełnymi adresami	5
P1–P3 — bezpieczne połączenie, pliki w katalogu publicznym bez plików prywatnych, ponowne wysłanie	4
K1–K4 — certyfikat i przekierowanie, kody odpowiedzi, Lighthouse przed i po poprawce, test mobilny	4
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Próba INF.03: witryna ze skryptem JS

Czas: 2 × 45 min

praca samodzielna — warunki egzaminacyjne (bez internetu i pomocy kolegów)

Cel: Samodzielne wykonanie w 90 minut witryny według opisu w stylu egzaminu INF.03: przygotowanie grafiki, struktura HTML, formatowanie CSS i skrypt obliczający koszt na podstawie formularza.

Po zajęciach potrafisz:

- przygotowuje grafikę według parametrów (kadrowanie do proporcji, skalowanie, eksport JPG)
- buduje stronę z bloków o zadanych szerokościach, kolorach i treściach
- formatuje elementy dokładnie według opisu (kolory, krój, marginesy, efekt najechania)
- pisze skrypt odczytujący pola formularza i wyświetlający wynik obliczeń w akapicie
- ocenia własną pracę według kryteriów egzaminacyjnych

Sprzęt i materiały

- komputer z Windows 11, przeglądarka, edytor kodu bez podpowiedzi AI
- folder `r1_30_start` z Pliki_do_zadan: `kajak_surowy.png`, `rzeka_baner.jpg`, `teksty.txt`

Oprogramowanie

- **Visual Studio Code lub Notepad++** — wyłącz rozszerzenia z podpowiedziami AI; Notepad++: <https://notepad-plus-plus.org/code.visualstudio.com>
- **GIMP** — kadrowanie, skalowanie, eksport JPG gimp.org/downloads

Przygotowanie przed zajęciami

- Powtórz zadania 20 i 22: układ bloków, formularz, odczyt pól w JavaScript.
- Utwórz folder `nazwisko_imie/zad30` — wszystkie pliki wynikowe zapisz bezpośrednio w nim.

Bezpieczeństwo

- Pracuj samodzielnie i bez dostępu do internetu — tak jak na egzaminie. Co 15 minut zapisuj pliki.

Przebieg zadania

1 Operacje na grafice

1. Plik `kajak_surowy.png` wykadruj do proporcji 4:3 (narzędzie Kadrowanie z opcją Stałe: proporcje 4:3) tak, aby kajak z wiosłarzem był w całości widoczny i znajdował się w środku kadru.
2. Przeskaluj obraz z zachowaniem proporcji do wymiarów 400 × 300 px i wyeksportuj jako `kajak.jpg` (jakość 80). Plik `rzeka_baner.jpg` skopiuj bez zmian.

2 Witryna — struktura

Teksty skopiuj z pliku `teksty.txt`. Strona ma tytuł (karta przeglądarki) „Wypożyczalnia kajaków”, język polski i kodowanie UTF-8. Arkusz stylów `styl.css` i skrypt `skrypt.js` są w osobnych plikach.

1. Bloki: baner, pod nim panel lewy i panel prawy obok siebie, na dole stopka. Stosuj znaczniki sekcji semantycznych.
2. Baner: nagłówek pierwszego stopnia z tekstu.
3. Panel lewy: nagłówek drugiego stopnia, tabela 3 × 3 z wierszem nagłówkowym (4 wiersze razem), akapit, obraz `kajak.jpg` z tekstem alternatywnym „kajak na rzece”.
4. Panel prawy: nagłówek drugiego stopnia; pole liczbowe (id `liczba`, min 1, max 10, wartość 1); lista rozwijana (id `trasa`) z trzema trasami, których wartościami są ceny 35, 55, 80; pole wyboru (id `transport`); dwa przyciski radiowe w jednej grupie (dzień powszedni — zaznaczony, weekend); przycisk „Oblicz”; pusty akapit (id `wynik`). Każde pole ma etykietę `label` powiązaną atrybutem `for`.
5. Stopka: akapit „Stronę wykonał: ” i Twoje imię i nazwisko.

3 Witryna — formatowanie CSS

1. Cała strona: krój Verdana, tło #F2F7F9, brak marginesu body.
2. Baner: wysokość 110 px, tło — obraz `rzeka_baner.jpg` dopasowany do bloku (`cover`) i kolor zapasowy #1E5F74, biały tekst wyśrodkowany w poziomie i pionie, cień tekstu.
3. Panel lewy: szerokość 35%, tło #D7EAF0, wypełnienie 20 px. Panel prawy: szerokość 65%, tło białe, wypełnienie 20 px, wysokość minimalna 560 px.
4. Nagłówki drugiego stopnia w kolorze #1E5F74. Tabela na całą szerokość panelu, obramowanie 1 px ciągłe #1E5F74, scalone, wiersz nagłówkowy na tle #1E5F74 białą czcionką; po najechaniu na wiersz komórki zmieniają tło na #F5B414.
5. Obraz: zaokrąglenie narożników 10 px. Przycisk: tło #F5B414, bez obramowania, wypełnienie 10 px 30 px, pogrubiony; po najechaniu tło #1E5F74 i biały tekst. Akapit `wynik`: 22 px, pogrubiony, kolor #1E5F74.
6. Stopka: tło #1E5F74, biały tekst wyrównany do prawej.

4 Skrypt

Skrypt uruchamia się po kliknięciu przycisku „Oblicz”.

1. Odczytaj liczbę kajaków i cenę trasy. Jeśli liczba jest spoza zakresu 1–10, w akapicie `wynik` wypisz „Podaj liczbę kajaków od 1 do 10” i zakończ obliczenia.
2. Cena za kajak = cena trasy, powiększona o 20 zł, gdy zaznaczono transport. Koszt = cena za kajak × liczba kajaków; w weekend koszt rośnie o 20%.
3. Wypisz w akapicie `wynik`: „Koszt wynajmu: X zł”, gdzie X ma 2 miejsca po przecinku. Kontrola: 3 kajaki, trasa B, transport, weekend → 270.00 zł.
4. Samoocena: po upływie czasu sprawdź swoją pracę według tabeli w sprawozdaniu, zrób zrzut strony z wynikiem dla danych kontrolnych i oddaj folder `zad30`.

W sprawozdaniu umieść

- ☐ folder `zad30`: `index.html`, `styl.css`, `skrypt.js`, `kajak.jpg`, `rzeka_baner.jpg`
- ☐ zrzut strony z wynikiem dla danych kontrolnych
- ☐ tabela samooceny z przyznanymi sobie punktami i komentarzem
- ☐ wnioski: co zajęło najwięcej czasu i co powtórzysz przed egzaminem

Pytania kontrolne

1. Dlaczego wartości listy rozwijanej (`value`) zawierają same ceny, a nie nazwy tras?

2. Jak zachować proporcje obrazu podczas skalowania w GIMP i dlaczego kadrujemy przed skalowaniem?
3. Co zwróci `document.getElementById("liczba").value` i dlaczego potrzebna jest zamiana na liczbę?

Kryteria oceny

Kryterium	Punkty
Grafika: kadr 4:3 z całym kajakiem, 400 × 300 px, plik JPG o właściwej nazwie	2
Struktura HTML: metadane i tytuł, 4 bloki semantyczne, tabela 4 wiersze, obraz z alt, linki do CSS i JS	4
Formularz: 5 kontrolek z właściwymi id, wartościami i etykietami <code>for</code> , grupa radio, akapit wyniku	3
CSS: baner z tłem i wyśrodkowaniem, panele 35/65%, tabela z obramowaniem i hover, przycisk z hover, stopka	5
Skrypt: walidacja zakresu, transport, weekend +20%, wynik z 2 miejscami (wynik kontrolny 270.00 zł)	4
Rzetelna samoocena i wnioski	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny cz. 1 — planowanie i makieta

Czas: 2 × 45 min

praca w zespołach 2–3 osobowych

Cel: Zaplanowanie witryny zespołowej: określenie odbiorców i wymagań, mapa strony, makiety mobile i desktop, repozytorium Git z podziałem zadań.

Po zajęciach potrafisz:

- formułuje sprawdzalne wymagania funkcjonalne i нефункционаłne witryny
- projektuje mapę strony i nawigację między podstronami
- rysuje makiety (wireframe) widoku mobilnego i desktopowego
- dobiera paletę kolorów z kontrolą kontrastu i kroje pisma
- zakłada wspólne repozytorium i dzieli zadania w zespole

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu; kartki A4 i ołówki (makiety odręczne są dopuszczalne)
- plik `r1_31_karta_projektu.txt` z Pliki_do_zadan

Oprogramowanie

- **Penpot lub Figma (plan darmowy)** — makiety; Figma: <https://www.figma.com/> — wystarczą prostokąty i teksty penpot.app
- **draw.io (diagrams.net)** — mapa strony jako diagram app.diagrams.net
- **Git i GitHub** — wspólne repozytorium (zadanie 28) git-scm.com/downloads
- **WebAIM Contrast Checker** — sprawdzenie kontrastu kolorów webaim.org/resources/contrastchecker

Przygotowanie przed zajęciami

- Dobierzcie się w zespoły i wybierzcie temat z listy: serwis koła zainteresowań lub klubu szkolnego, katalog filmów/gier/książek, strona wydarzenia szkolnego (dzień otwarty, turniej, koncert), przewodnik po okolicy, schronisko lub zbiórka charytatywna (fikcyjne). Inny temat — po akceptacji nauczyciela.
- Przejrzyjcie 2–3 istniejące strony o podobnym temacie i zanotujcie, co jest w nich wygodne, a co nie.

Bezpieczeństwo

- Nie publikujcie danych osobowych ani zdjęć osób bez ich zgody. Jeśli strona dotyczy prawdziwej szkoły lub klubu, uzgodnijcie treści z nauczycielem.
- Używajcie tylko własnych grafik lub grafik z licencją pozwalającą na użycie (zapisujcie autora i licencję w karcie projektu).

Przebieg zadania

1 Temat, odbiorcy i wymagania

Wymaganie ma być sprawdzalne: „katalog zawiera min. 8 filmów z plakatem, tytułem i gatunkiem” zamiast „ładny katalog”.

1. P1. Skopiujcie `r1_31_karta_projektu.txt` jako `PROJEKT.txt`. Wpiszcie skład zespołu z rolami, temat, 2–3 grupy odbiorców i cel strony (punkty 1–2 karty).
2. P2. Zapiszcie co najmniej 6 wymagań funkcjonalnych (W1...) i 4 niefunkcjonalne (N1...), w tym: responsywność od 360 px, walidacja W3C bez błędów, Lighthouse dostępność ≥ 90 .
3. P3. Zaplanujcie co najmniej 2 elementy JavaScript, które dodacie w części 3 (np. filtr, walidacja formularza, galeria, tryb ciemny, quiz).

2 Mapa strony i makiety

Makieta pokazuje układ bloków (prostokąty z opisem, obrazki jako prostokąt z krzyżykiem), a nie gotową grafikę.

1. P4. Narysujcie mapę strony: co najmniej 4 podstrony, menu globalne i najważniejsze linki między podstronami (draw.io lub lista z wcięciami).
2. P5. Wykonajcie makietę strony głównej w dwóch wersjach: mobile (360 px, menu zwinięte) i desktop (≥ 900 px). Opiszcie w makiecie nagłówki, liczbę kolumn i kolejność bloków.
3. P6. Wykonajcie makietę (wystarczy mobile) jeszcze jednej podstrony — z tabelą, formularzem lub katalogiem.
4. P7. Dobierzcie paletę (3–5 kolorów HEX) i 1–2 kroje pisma. Sprawdźcie kontrast tekstu z tłem — co najmniej 4.5:1 — i wpiszcie wyniki do karty.

3 Repozytorium i podział pracy

1. P8. Jedna osoba zakłada repozytorium na GitHub i dodaje pozostałych (Settings → Collaborators). Każdy klonuje repozytorium (`git clone`).
2. P9. Dodajcie do repozytorium `PROJEKT.txt`, makiety (PNG/SVG/PDF lub zdjęcia rysunków) w folderze `projekt/` i pusty szkielet folderów (`css/`, `js/`, `img/`). Każda osoba wykonuje co najmniej jeden commit.
3. P10. Wpiszcie podział zadań na części 2 i 3 (kto, co, do kiedy) i 2 ryzyka z planem działania. Dla chętnych: zadania jako Issues na GitHubie.
4. Pokażcie nauczycielowi kartę i makiety — zatwierdzenie jest warunkiem rozpoczęcia części 2.

W sprawozdaniu umieść

- ☐ wypełniona karta `PROJEKT.txt` (wszystkie 10 punktów)
- ☐ mapa strony
- ☐ makiety: strona główna mobile i desktop, druga podstrona mobile
- ☐ adres repozytorium i zrzut listy commitów (co najmniej 1 commit każdej osoby)
- ☐ odpowiedzi na pytania kontrolne (każdy członek zespołu osobno)

Pytania kontrolne

1. Czym różni się wymaganie funkcjonalne od niefunkcjonalnego? Podaj po jednym przykładzie z waszego projektu.
2. Dlaczego makietę projektuje się najpierw dla telefonu (mobile first)?
3. Po co sprawdzać kontrast kolorów i jaka jest minimalna wartość dla zwykłego tekstu według WCAG (poziom AA)?
4. Jak zespół może uniknąć konfliktów przy wspólnej pracy w jednym repozytorium?

Kryteria oceny

Kryterium	Punkty
Temat, odbiorcy, cel i min. 6 sprawdzalnych wymagań funkcjonalnych + 4 нефункционалне	5
Mapa strony (min. 4 podstrony, menu, powiązania)	2
Makiety: główna mobile i desktop, druga podstrona; opis kolumn i bloków	5
Paleta i kroje z kontrastem $\geq 4.5:1$, źródła grafik	2
Repozytorium z kartą i makietami, commity wszystkich członków, podział zadań i ryzyka	3
Indywidualne odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny cz. 2 — HTML i CSS

Czas: 2 × 45 min

praca w zespołach 2–3 osobowych

Cel: Zbudowanie wszystkich podstron projektu zespołowego w semantycznym HTML i responsywnym CSS zgodnie z makietami, z regularnymi commitami każdego członka zespołu.

Po zajęciach potrafisz:

- tworzy spójny zestaw podstron z wspólnym nagłówkiem, menu i stopką
- stosuje znaczniki semantyczne, poprawną hierarchię nagłówków, tabele i formularze z etykietami
- pisze arkusz CSS w podejściu mobile first z punktami przerwania, Flexboxem i Gridem
- przygotowuje grafiki o właściwych wymiarach i rozmiarach plików
- pracuje w zespole z Gitem: osobne gałęzie, scalanie, opisowe commity

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- repozytorium projektu z zadania 31 (karta `PROJEKT.txt`, makiety)

Oprogramowanie

- **Visual Studio Code** — Live Server, panel Source Control
code.visualstudio.com
- **Git i GitHub** — gałęzie i scalanie (zadanie 28)
git-scm.com/downloads
- **GIMP** — skalowanie i eksport grafik (WebP/JPG/PNG)
gimp.org/downloads
- **Validator W3C** — HTML; CSS:
<https://jigsaw.w3.org/css-validator/validator.w3.org>

Przygotowanie przed zajęciami

- Każdy członek zespołu wykonuje `git pull`, aby mieć aktualną wersję repozytorium.
- Przejrzyjcie makiety i podział zadań z karty projektu; ustalcie wspólne nazwy klas (np. `.wrap`, `.karta`, `.przycisk`).

Bezpieczeństwo

- Każdy pracuje na swojej gałęzi i edytuje przede wszystkim swoje pliki — tak unikniecie nadpisania pracy kolegi.
- Grafiki tylko własne lub z licencją wpisaną do karty projektu; bez zdjęć osób bez ich zgody.

Przebieg zadania

1 Wspólny szkielet (cały zespół, 20 min)

Najpierw jedna osoba przygotowuje wspólny szablon strony, wysyła go, a pozostali go pobierają — dopiero potem praca równoległa.

1. H1. Utwórzcie szablon: `<!DOCTYPE html>`, `lang="pl"`, `meta charset` i `viewport`, unikatowy `title` i `meta description`, link do `css/styl.css`, nagłówek z logo i menu (lista odnośników), `main`, stopka. Zatwierdźcie i wyślijcie (`git push`).
2. H2. Każdy pobiera szablon (`git pull`) i tworzy swoją gałąź, np. `git switch -c strona-filmy`.

2 Podstrony (praca równoległa)

Treści mają być prawdziwe dla waszego tematu — bez „lorem ipsum”.

1. H3. Każda podstrona z mapy strony ma treść zgodną z makietą: jeden `h1`, logiczne `h2 / h3`, sekcje semantyczne (`section`, `article`, `aside`), aktywna pozycja menu oznaczona klasą i `aria-current="page"`.
2. H4. W projekcie musi wystąpić co najmniej: tabela z `caption`, `thead` i `th scope`; formularz z `fieldset`, `legend`, etykietami `label for` i walidacją natywną (`required`, `type="email"`, `minlength`); lista; co najmniej 4 grafiki z `alt`, `width` i `height`.
3. H5. Przygotujcie grafiki w GIMP: wymiary dopasowane do miejsca na stronie (np. plakaty 300 × 450 px), każdy plik poniżej 150 KB, nazwy małymi literami bez polskich znaków i spacji. Dodajcie favicon.
4. H6. Każdy członek zespołu wykonuje co najmniej 3 commity z opisowymi komunikatami (np. „Tabela harmonogramu seansów”, nie „zmiany”).

3 CSS — mobile first

Jeden wspólny arkusz `css/styl.css`. Style bazowe piszcie dla telefonu, a szersze ekrany obsłużcie regułami `@media (min-width: ...)`.

1. C1. Zdefiniujcie kolory i odstępy jako zmienne CSS w `:root` (paleta z karty projektu) i używajcie ich w całym arkuszu.
2. C2. Układ: menu zwijane bez JavaScript na telefonie (checkbox + etykieta lub `details`), poziome od szerokości desktopowej; sekcje kart w Gridzie lub Flexboksie — 1 kolumna na telefonie, 2 na tablecie, 3–4 na desktopie, zgodnie z makietą.
3. C3. Tabela na wąskim ekranie przewija się wewnątrz kontenera (`overflow-x: auto`), a strona nie przewija się w poziomie. Obrazy nie wychodzą poza kontener (`max-width: 100%`).
4. C4. Dostępność i efekty: widoczny `:focus-visible`, `:hover` na linkach i przyciskach z `transition`, kontrast zgodny z kartą; reguła `prefers-reduced-motion` wyłączająca animacje.

4 Scalanie i kontrola

1. K1. Scalcie gałęzie do `main` (rozwiązując ewentualne konflikty) i wyślijcie na GitHub. Wyświetlcie `git log --oneline --graph --all` i zróbcie zrzut.
2. K2. Sprawdźcie każdą podstronę walidatorem W3C HTML oraz arkusz walidatorem CSS; poprawcie błędy (ostrzeżenia zapiszcie).
3. K3. W trybie urządzeń F12 sprawdźcie szerokości 360, 768 i 1280 px — zrzuty strony głównej w trzech szerokościach porównajcie z makietami. Zapiszcie różnice i ich powód.

W sprawozdaniu umieść

- ☐ lista podstron z autorem każdej (kto co wykonał)
- ☐ wyniki walidacji HTML i CSS (zrzuty lub liczba błędów przed i po poprawkach)
- ☐ zrzuty strony głównej przy 360, 768 i 1280 px obok makiet

- ☐ zrzut grafu historii Git z commitami wszystkich członków
- ☐ adres repozytorium
- ☐ odpowiedzi na pytania kontrolne (każdy osobno)

Pytania kontrolne

1. Na czym polega podejście mobile first w CSS i jak zapisujecie reguły dla większych ekranów?
2. Do czego służą zmienne CSS i jaką korzyść dały w waszym projekcie?
3. Jak zrobiliście menu zwijane bez JavaScript i jak działa ono z klawiatury?
4. Dlaczego obrazy mają atrybuty `width` i `height`, mimo że ich rozmiar ustala CSS?

Kryteria oceny

Kryterium	Punkty
H1-H3 — wszystkie podstrony z mapy, wspólny szablon, semantyka, aktywne menu	4
H4-H5 — tabela, formularz z walidacją natywną, lista, min. 4 grafiki z alt i wymiarami, pliki < 150 KB	4
C1-C4 — zmienne CSS, mobile first (1/2/3-4 kolumny), menu bez JS, brak przewijania w poziomie, focus i reduced motion	5
K2-K3 — walidacja HTML i CSS bez błędów, porównanie z makietami	2
H6, K1 — min. 3 opisowe commity każdej osoby, scalone gałęzie, graf historii	3
Indywidualne odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny cz. 3 — JavaScript, testy i publikacja

Czas: 2 × 45 min

praca w zespołach 2–3 osobowych

Cel: Dodanie do projektu zespołowego interaktywnych elementów JavaScript, przetestowanie witryny (scenariusze, W3C, Lighthouse), opublikowanie jej i zaprezentowanie efektów.

Po zajęciach potrafisz:

- implementuje zaplanowane funkcje JavaScript w osobnym pliku, uruchamiane tylko na podstronach, które ich potrzebują
- przygotowuje i wykonuje scenariusze testów oraz dokumentuje wyniki w protokole
- poprawia błędy wskazane przez walidator W3C i audyt Lighthouse
- publikuje witrynę (GitHub Pages lub hosting) i sprawdza działanie wersji online
- prezentuje projekt w 5 minut, uzasadniając decyzje projektowe

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu; smartfon do testu (opcjonalnie)
- repozytorium projektu po zadaniu 32

Oprogramowanie

- **Visual Studio Code** — Live Server
code.visualstudio.com
- **Przeglądarka Chrome lub Edge** — F12: Console, Lighthouse, tryb urządzeń
- **Walidator W3C** — HTML; CSS:
<https://jigsaw.w3.org/css-validator/validator.w3.org>
- **GitHub Pages** — lub hosting szkolny przez SFTP (zadania 28–29)
pages.github.com

Przygotowanie przed zajęciami

- `git pull` u każdego członka zespołu; sprawdźcie w karcie projektu, które elementy JS zaplanowaliście i kto je wykonuje.
- Powtórzcie zadania 25–27: DOM, zdarzenia, walidacja formularza, localStorage.

Bezpieczeństwo

- Dane wpisane przez użytkownika wstawiajcie przez `textContent`; nie używajcie `innerHTML` z danymi z formularza.
- Formularz na stronie statycznej nigdzie nie wysyła danych — nie zbierajcie prawdziwych danych osobowych, a w testach używajcie danych zmyślonych (`example.edu.pl`).
- Przed publikacją usuńcie z repozytorium pliki prywatne i dane dostępne (zadanie 28).

Przebieg zadania

1 JavaScript

Cały kod w pliku `js/main.js` dołączonym z `defer`. Kod dla jednej podstrony uruchamiajcie warunkowo, np. `const katalog = document.querySelector("#katalog"); if (katalog) { ... } —` —
inaczej na pozostałych stronach pojawi się błąd.

1. J1. Zaimplementujcie co najmniej 2 elementy JS z karty projektu, w tym co najmniej jeden reagujący na dane wpisane przez użytkownika (filtr, wyszukiwarka, walidacja formularza, kalkulator, quiz).
2. J2. Jeśli projekt ma formularz: własna walidacja przy `submit` z `preventDefault`, komunikaty pod polami (`textContent`), ustawienie focusu w pierwszym błędnym polu, potwierdzenie po poprawnym wypełnieniu.
3. J3. Strona ma pozostać używalna bez JavaScript: treść (np. karty katalogu) jest w HTML, a skrypt ją tylko filtruje lub uzupełnia. Sprawdźcie to, wyłączając JS (F12 → Ctrl+Shift+P → „Disable JavaScript”).
4. J4. Każdy członek zespołu zatwierdza swoją część kodu osobnym commitem z opisowym komunikatem.

2 Testy i poprawki

Test zapisujcie w tabeli: numer, funkcja, dane lub czynność, oczekiwany wynik, wynik (OK / błąd i poprawka).

1. T1. Przygotujcie i wykonajcie co najmniej 8 scenariuszy testów (w tym: dane poprawne, puste pole, wartość graniczna, brak wyników, wielkość liter, odświeżenie strony, wąski ekran 360 px, nawigacja klawiaturą). Sprawdźcie konsolę na każdej podstronie.
2. T2. Sprawdźcie wszystkie podstrony walidatorem W3C HTML i arkusz walidatorem CSS — cel: 0 błędów.
3. T3. Uruchomcie Lighthouse (Mobile) na stronie głównej i jeszcze jednej podstronie. Cel z karty: dostępność i SEO ≥ 90 . Poprawcie co najmniej dwa zgłoszone problemy i zapiszcie wyniki przed i po.

3 Publikacja i prezentacja

1. U1. Opublikujcie witrynę przez GitHub Pages (Settings → Pages) lub na serwerze szkolnym. Sprawdźcie wersję online: wszystkie podstrony, grafiki, skrypty (wielkość liter w nazwach plików!), test na telefonie.
2. U2. Uzupełnijcie `PROJEKT.txt` lub `README.md`: adres strony, autorzy i ich wkład, lista wymagań z karty z oznaczeniem spełnione / niespełnione.
3. U3. Przygotujcie prezentację na 5 minut: cel i odbiorcy, pokaz na żywo (wersja online), jedna ciekawa decyzja techniczna każdej osoby, wynik Lighthouse, czego się nauczyliście. Każdy członek zespołu mówi.
4. U4. Po prezentacjach oceńcie jeden inny projekt według kryteriów z tego zadania (ocena koleżeńska: 2 mocne strony, 1 propozycja zmiany).

W sprawozdaniu umieść

- ☐ opis elementów JS z fragmentami kodu i autorem
- ☐ protokół testów (min. 8 scenariuszy)
- ☐ wyniki W3C oraz Lighthouse przed i po poprawkach
- ☐ adres opublikowanej strony i repozytorium, zestawienie wymagań spełnionych / niespełnionych
- ☐ ocena koleżeńska innego zespołu
- ☐ odpowiedzi na pytania kontrolne (każdy osobno)

Pytania kontrolne

1. Dlaczego kod JavaScript dla jednej podstrony uruchamiamy warunkowo (`if (element)`)? Jaki błąd pojawi się bez tego?
2. Co oznacza, że strona działa bez JavaScript (progressive enhancement), i jak to sprawdziliście?
3. Który problem wskazany przez Lighthouse poprawiliście i dlaczego był ważny dla użytkowników?
4. Które wymaganie z karty projektu nie zostało spełnione i co należałoby zrobić inaczej?

Kryteria oceny

Kryterium	Punkty
J1–J2 — min. 2 działające elementy JS, w tym obsługa danych użytkownika; walidacja z komunikatami i focusem	6
J3–J4 — strona używalna bez JS, warunkowe uruchamianie, brak błędów w konsoli, commity każdej osoby	2
T1 — protokół min. 8 scenariuszy z wynikami i poprawkami	3
T2–T3 — W3C bez błędów, Lighthouse dostępność i SEO ≥ 90 lub opisane poprawki	3
U1–U2 — działająca wersja online, uzupełniona dokumentacja projektu	3
U3–U4 — prezentacja 5 min z udziałem wszystkich, ocena koleżeńska	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Zadanie zaliczeniowe roku 1

Czas: 2 × 45 min

praca samodzielna — warunki egzaminacyjne

Cel: Sprawdzenie umiejętności z całego roku: samodzielne wykonanie w 90 minut witryny z opisu — grafika, semantyczny HTML, CSS Grid/Flexbox z responsywnością, tabela, formularz i skrypt obsługujący zdarzenia.

Po zajęciach potrafisz:

- przygotowuje grafiki w GIMP według parametrów (skalowanie z zachowaniem proporcji, przezroczyste tło, eksport)
- buduje stronę z sekcji semantycznych z menu prowadzącym do kotwic
- projektuje układ CSS Grid z obszarami i przebudową dla szerokich ekranów
- formatuje tabelę i formularz według opisu
- pisze skrypt reagujący na zmiany pól formularza i wyświetlający wynik na żywo

Sprzęt i materiały

- komputer z Windows 11, przeglądarka, edytor kodu bez podpowiedzi AI, bez dostępu do internetu
- folder `r1_34_start` z Pliki_do_zadan:
`sciana_surowa.png`, `logo_biale_tlo.png`,
`teksty.txt`

Oprogramowanie

- **Visual Studio Code** lub **Notepad++** — bez rozszerzeń z podpowiedziami AI; Notepad++: <https://notepad-plus-plus.org/code.visualstudio.com>
- **GIMP** — Skaluj obraz, Kolor na alfę, Eksportuj jako gimp.org/downloads

Przygotowanie przed zajęciami

- Powtórz zadania 17 (GIMP), 14–15 (Grid, RWD), 26 i 30 (formularz i skrypt).
- Utwórz folder `nazwisko_imie/zad34` — wszystkie pliki wynikowe zapisz bezpośrednio w nim.

Bezpieczeństwo

- Pracuj samodzielnie, bez internetu i bez pomocy innych osób. Zapisuj pliki co kilka minut.

Przebieg zadania

1 Grafika

1. Plik `sciana_surowa.png` przeskaluj z zachowaniem proporcji do szerokości 600 px i wyeksportuj jako `sciana.jpg` (jakość 80).
2. W pliku `logo_biale_tlo.png` usuń białe tło tak, aby było przezroczyste: Warstwa → Przezroczystość → Dodaj kanał alfa, potem narzędzie Zaznaczanie według koloru (niski próg, np. 15) kliknij białe tło i naciśnij Delete. Jasnoszary trójkąt w logo ma zostać nieprzezroczysty. Przeskaluj do 120 × 120 px i wyeksportuj jako `logo.png`.

2 Struktura HTML

Teksty skopiuj z `teksty.txt`. Strona: język polski, UTF-8, `viewport`, tytuł karty „Ścianka wspinaczkowa Chwyć”, arkusz `styl.css`, skrypt `skrypt.js` dołączony z `defer`.

1. Bloki: `header` (logo z tekstem alternatywnym „logo ścianki Chwyć”, nagłówek `h1`, akapit z podtytułem), `nav` (lista 4 odnośników do sekcji o id `cennik`, `kalkulator`, `zasady`, `kontakt`), `main` z czterema sekcjami: O nas, Cennik, Kalkulator, Zasady; `footer` o id `kontakt`.
2. Sekcja O nas: `h2`, akapit, obraz `sciana.jpg` z opisowym tekstem alternatywnym i atrybutami `width`, `height`.
3. Sekcja Cennik: `h2` i tabela z podpisem (`caption`), wierszem nagłówkowym w `thead` (`th scope="col"`) i 5 wierszami danych w `tbody`.
4. Sekcja Kalkulator: formularz o id `formularz`: lista `bilet` (wartości 35 i 28), pole liczbowe `wejścia` (1–20, wartość 1), pola wyboru `buty` i `uprzaz` — każde z etykietą; akapit z informacją o rabacie; akapit `cena` z tekstem „Do zapłaty: 35.00 zł”.
5. Sekcja Zasady: `h2` i lista numerowana. Stopka: akapit z adresem, odnośnikiem `mailto:` i godzinami oraz akapit „Stronę wykonał(a): ” i Twoje imię i nazwisko.

3 CSS

1. Krój Tahoma, tło strony `#F4F1EC`. Układ całej strony w CSS Grid z nazwanymi obszarami (`grid-template-areas`): nagłówek, menu, treść, stopka.
2. Nagłówek: tło `#1E78DC`, biały tekst, logo 80 × 80 px obok tekstu (Flexbox, elementy wyśrodkowane w pionie). Menu: tło `#FAC81E`, odnośniki w poziomie wyśrodkowane, pogrubione, bez podkreślenia; po najechnięciu tło `#DC283C` i biały tekst.
3. Sekcje: białe tło, zaokrąglenie 12 px, cień, nagłówki `h2` w kolorze `#1E78DC` z dolną linią 3 px `#FAC81E`. Obraz dopasowany do szerokości sekcji.
4. Tabela: na całą szerokość, scalone obramowania, nagłówki na tle `#1E78DC` białą czcionką, co drugi wiersz danych na tle `#EAF2FC`, ceny wyrównane do prawej. Akapit `cena`: 24 px, pogrubiony, kolor `#DC283C`.
5. Stopka: tło `#222222`, biały, wyśrodkowany tekst. Responsywność: do 799 px sekcje jedna pod drugą; od 800 px sekcje w dwóch kolumnach, a sekcja Zasady na całą szerokość. Przy szerokości 360 px strona nie może się przewijać w poziomie.

4 Skrypt

Cena przelicza się na żywo — przy każdej zmianie dowolnego pola formularza (zdarzenia `input` i `change`), bez przycisku.

1. Cena za wejście = cena biletu + 10 zł, gdy zaznaczono buty, + 5 zł, gdy zaznaczono uprzęż. Suma = cena za wejście × liczba wejść; od 10 wejść suma jest obniżona o 15%.
2. Gdy liczba wejść jest spoza zakresu 1–20, w akapicie `cena` wyświetl „Liczba wejść musi być od 1 do 20”. Wynik wyświetlaj w formacie „Do zapłaty: 323.00 zł” (ulgowy, 10 wejść, buty).
3. Po zakończeniu sprawdź stronę przy szerokości 360 i 1280 px oraz konsolę. Oddaj folder `zad34`: `index.html`, `styl.css`, `skrypt.js`, `sciana.jpg`, `logo.png`.

W sprawozdaniu umieść

- ☐ folder `zad34` z pięcioma plikami wynikowymi
- ☐ zrzuty strony przy 360 i 1280 px z wynikiem kontrolnym 323.00 zł
- ☐ tabela testów skryptu (4 przypadki)
- ☐ krótka samoocena: co sprawiło trudność

Pytania kontrolne

1. Dlaczego skrypt dołączony w `head` z atrybutem `defer` może od razu wyszukać elementy formularza?
2. Czym różni się zapis `column 1fr` od `minmax(0, 1fr)` i kiedy ma to znaczenie na wąskim ekranie?
3. Jak w GIMP uzyskać przezroczyste tło i dlaczego logo trzeba zapisać jako PNG, a nie JPG?

Kryteria oceny

Kryterium	Punkty
Grafika: sciana.jpg 600 × 400 px, logo.png 120 × 120 px z przezroczystym tłem	2
HTML: metadane, 4 bloki semantyczne, menu z kotwicami, obrazy z alt i wymiarami, tabela z caption/thead/tbody, lista numerowana	4
Formularz: 4 kontrolki z właściwymi id, wartościami i etykietami, akapit <code>cena</code>	2
CSS: Grid z obszarami, nagłówek Flexbox, menu z hover, sekcje i tabela według opisu	5
RWD: 1 kolumna do 799 px, 2 kolumny od 800 px, Zasady na całą szerokość, brak poziomego przewijania przy 360 px	2
Skrypt: przeliczanie na żywo (input/change), dodatki, rabat od 10 wejść, komunikat zakresu, format 2 miejsc	5
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).