
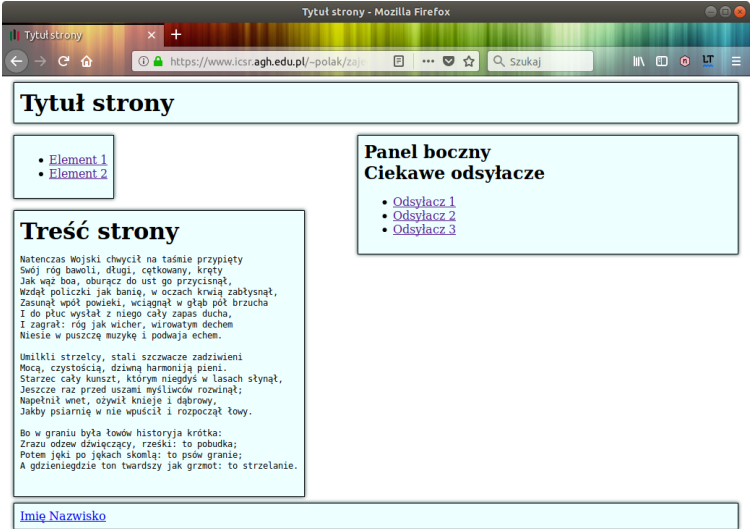


Standard DOM 4

1. Dostęp do stylów

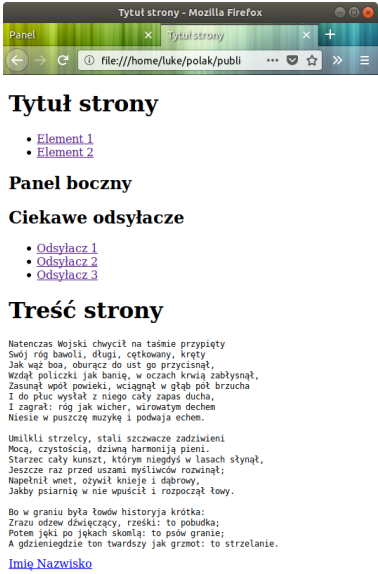
 **1 pkt** za wykonanie wszystkich zadań w tej sekcji

1. Zmodyfikuj pierwszy przykładowy dokument HTML, który był zamieszczony w konspekcie pierwszych zajęć — rozszerz go o formularz zawierający dwa przyciski: "Ustaw" oraz "Skasuj"
2. Stwórz JavaScriptową wersję arkusza 'sheet.css' utworzonego na pierwszych ćwiczeniach — po naciśnięciu przycisku "Ustaw" skrypt, **korzystając z własności DOM 'style' oraz 'class'**, styluje elementy dokumentu HTML; w rezultacie powinniśmy więc otrzymać




stronę o następującym wyglądzie:

3. W przypadku naciśnięcia przycisku "Skasuj", skrypt przywraca oryginalny (początkowy) wygląd strony, tzn. usuwa wszystkie style CSS związane z elementami (usuwa stylowanie elementów) — w tym przypadku strona powinna wyglądać jak na poniższym zrzucie




ekranowym:

2. Harmonogramowanie wywołań funkcji

 **1 pkt** za wykonanie wszystkich zadań z tej sekcji

1. Przeczytaj opis metod: [window.setInterval\(\)](#), [window.setTimeout\(\)](#), [window.requestAnimationFrame\(\)](#) oraz [wątków roboczych](#)
2. Wypróbuj [działanie aplikacji](#) przedstawionej w artykule [Intensive JavaScript](#) — zidentyfikuj, opisane tam, "wąskie gardła" aplikacji
3. Utwórz formularz zawierający:
 - o Przycisk *Start*
 - o Przycisk *Stop*
 - o Pole liczbowe *delay*
4. Napisz skrypt zawierający:
 - o Dwie globalne tablice: *SetIntervalTime* oraz *SetTimeoutTime*
 - o Dwie funkcje obciążające: `doTimeConsumingCallculationsWithSetInterval()` oraz `doTimeConsumingCallculationsWithSetTimeout()` — każda z nich ma wykonywać poniższy algorytm:
 1. [Dodaj](#) do prawego końca tablicy, odpowiednio: *SetIntervalTime* / *SetTimeoutTime*, wynik wykonania metody [performance.now\(\)](#).
 2. Jeżeli długość tablicy jest większa niż *N* (zmienna globalna w skrypcie), to [usuń](#) pierwszy element tablicy
 3. Wykonaj kod zawierający intensywne obliczenia, np. wywołaj funkcję [calculatePrimes\(1000, 1000000000\)](#) — jej implementacja znajduje się w powyższym artykule
 - o Funkcję wizualizacyjną `drawChart()`, która:
 1. Bazując na danych zawartych w obydwu tablicach, oblicza średnią z różnic `SetXxxTime[i+1]-SetXxxTime[i]` — średni czas cyklicznego wywołania funkcji obciążających
 2. Wizualizuje obydwie średnie na jednym wykresie — użyj biblioteki stworzonej w ramach zadania dla geek-ów (patrz ćwiczenie 2) lub [CanvasJS](#) (ewentualnie [dowolnej innej](#) służącej do tworzenia wykresów)

Dla chętnych / dociekliwych : Dokonaj wizualizacji (wykres słupkowy, liniowy lub warstwowy) *N-1* (ostatnich) czasów cyklicznego wywołania funkcji obciążających (`czasCyklicznegoWywołaniai = SetXxxTime[i+1]-`

- Czy czasy cyklicznego wywołania, w przypadku obydwu funkcji obciążających, są (w miarę) jednakowe, czy też może któraś z nich jest znacznie rzadziej wywoływana niż druga?
- Czy kolejne wartości $czasCyklicznegoWywołania_i$, dla danej (pojedynczej) funkcji obciążającej, są w miarę zbliżone (jednakowe), czy nie?

- Funkcję `start()`, która (po naciśnięciu przycisku *Start*):

1. Uruchamia cykliczne wywoływanie funkcji obciążającej `doTimeConsumingCallculationsWithSetInterval()` — należy wywołać metodę `window.setInterval(doTimeConsumingCallculationsWithSetInterval, M)`

Zakładamy, że wartość *M* (czas w milisekundach) to wartość wyspecyfikowana w polu *delay*

2. Uruchamia cykliczne działanie funkcji obciążającej `doTimeConsumingCallculationsWithSetTimeout()` — wywołaj metodę `window.setTimeout(doTimeConsumingCallculationsWithSetTimeout, M)` z ciała funkcji `start()` oraz umieść jej wywołanie w ostatniej linii funkcji obciążającej `doTimeConsumingCallculationsWithSetTimeout()` — zapętlj wywołanie funkcji obciążającej

```
function start() {  
  ...  
  window.setTimeout(doTimeConsumingCallculationsWithSetTimeout, M);  
  ...  
}  
  
function doTimeConsumingCallculationsWithSetTimeout() {  
  ...  
  window.setTimeout(doTimeConsumingCallculationsWithSetTimeout, M); // ⇔ window.setTimeout(arguments.callee, M);  
}
```

Copy

3. Uruchamia cykliczne działanie funkcji wizualizacyjnej `drawChart()` — użyj metody `window.requestAnimationFrame(drawChart)` w funkcji `start()`, a następnie wstaw wywołanie tej metody w ostatniej linii funkcji wizualizacyjnej

```
function start() {  
  ...  
  window.requestAnimationFrame(drawChart);  
  ...  
}  
  
function drawChart() {  
  ...  
  window.requestAnimationFrame(drawChart); // ⇔ window.requestAnimationFrame(arguments.callee);  
}
```

Copy

- Funkcję `stop()`, która (po naciśnięciu przycisku *Stop*) zatrzymuje proces cyklicznego działania funkcji: wizualizacyjnej oraz obciążających

5. Zaobserwuj:

- Czy obydwie średnie mają zbliżoną, czy znacząco różniącą się wartość?
- Czy w przypadku użycia `setInterval(funkcjaObciążająca, opóźnienie)` bądź `setTimeout(funkcjaObciążająca, opóźnienie)`, czas pomiędzy kolejnymi wykonywaniami kodu *funkcjaObciążająca* jest zawsze taki, jak określono w parametrze *opóźnienie* (w polu *delay* formularza)? Spróbuj wywnioskować, czy kod *funkcjaObciążająca* jest wykonywany w osobnym, czy w głównym wątku — swoje obserwacje porównaj z informacjami zawartymi w [artykule](#)

3. Użycie DOM

1. 🏠 **1 pkt** za wykonanie dwóch poniższych podpunktów:

- I. Stwórz dokument HTML składający z: formularza oraz wielu (np. 10) elementów `span` zawierających cyfrę 0, tj. `0`.

W oparciu o standard [HTML DOM](#) napisz skrypt o następujących założeniach:

- Skrypt cyklicznie, co sekundę, dekrementuje wartość zmiennej aż do osiągnięcia wartości 0
- Wypisuje jej, aktualną, wartość w obrębie każdego z elementów *span*
- Osiągnięcie wartości 0 ma nie wstrzymywać cyklicznego wykonywania kodu — ma się wykonywać pusty / jałowy kod
- Wartość początkową dekrementowanej zmiennej określa użytkownik za pomocą pola tekstowego (formularza) opatrzonego identyfikatorem 'licznik', a [wartością domyślną](#) wyświetlaną w tym polu ma być 10. Gdy wartość zmiennej osiągnie 0, to zawartość pola tekstowego też ma być 0
- Gdy użytkownik wpisze wartość większą od 0, to ponownie ma wykonywać się cykliczna dekrementacja

- Dostęp do treści elementów *span* oraz pola tekstowego ma się odbywać przy użyciu metody [document.getElementById\(\)](#), [document.querySelector\(\)](#), [document.getElementsByTagName\(\)](#) lub [document.querySelectorAll\(\)](#).
- Modyfikowanie treści dokumentu HTML może się odbywać, **tylko i wyłącznie**, w oparciu o API DOM 4, czyli przykładowo **nie można** używać [innerHTML](#), będącego częścią DOM 0

II. Utwórz alternatywną wersję skryptu — zamiast elementów *span* mamy komponenty webowe ([wprowadzenie](#), [opis 1](#), [opis 2](#)), a każdy z nich (niezależnie) odczytuje, a następnie wyświetla, aktualną wartość (dekrementowanej) zmiennej. W tym przypadku komponenty, ale tylko one, mogą korzystać z własności *innerHTML*

Wskazówki:

- Komponenty są niezależne, a więc w naszym przypadku, każdy z nich, niezależnie od pozostałych, modyfikuje swój licznik / zmienną i **każdy z nich używa odrębnej funkcji do harmonogramowania wywołania kodu**
- Artykuły: opisujący [wady oraz zalety technologii "Web Components"](#) oraz porównujący [komponenty webowe z React.JS](#)
- [Przykład "Hello World"](#)

Wprowadzenie do "Web Components"

The State of Web Components - Ana...



Watch on YouTube

Krótki tutorial

How to create a Web Component usi...



Watch on YouTube

2. (2 pkt.) Napisz skrypt, który:

- Modyfikuje treść / zawartość dokumentu HTML używając standardu DOM 4
- Obsługuje zdarzenia za pomocą odpowiedniego [obserwatora zdarzeń](#)
- Implementuje funkcjonalność określoną na początku ćwiczeń

- Strona zawiera:
 - pusty [wykaz UL](#) poprzedzony napisem "Spis treści"
 - [nagłówki](#) poziomu *N*
 - inne elementy
- Kliknięcie nagłówka ma spowodować:
 - Utworzenie [etykiety](#) do klikniętego nagłówka
 - Dodanie, do wykazu UL (spisu treści), nowego elementu 'li' zawierającego odsyłacz do etykiety nagłówka — kliknięcie elementu 'li' (spisu treści) ma spowodować, że przeglądarka przenosi użytkownika do danego nagłówka
- Treścią odsyłacza (w spisie treści) ma być treść nagłówka
- Kolejność elementów w spisie treści ma odpowiadać kolejności pojawiania się nagłówków w treści dokumentu
- Ponowne kliknięcie danego nagłówka ma powodować jego usunięcie ze spisu treści
- Skrypt ma współpracować z dowolnym kodem HTML, tzn. nie powinien wymagać modyfikowania treści (HTML) znaczników otwierających `<h1>`, `<h2>`, ... `<h6>`
- Przykład:
 - Dla kodu HTML

```
...
<strong>Spis treści</strong>
<ul></ul>

<h1>Wstęp</h1>
treść akapitu
<h2>Motywacja badań</h2>
treść akapitu
<h1>Przegląd istniejących rozwiązań</h1>
treść akapitu
<h1>Wnioski</h1>
treść akapitu
...
```

- Po kliknięciu nagłówka "Motywacja badań" spis treści ma wyglądać następująco:

Spis treści

... ..

- Po kliknięciu nagłówka "Wstęp" spis treści ma wyglądać następująco:

Spis treści

- [Wstęp](#)
- [Motywacja badań](#)

3.  (2 pkt.) W oparciu o [tutorial](#) napisz grę przeglądarkową:

1. Rozgrywka odbywa się w obrębie płótna "canvas" lub w obrębie całej strony WWW
2. Sterowanie odbywa się myszką lub przy pomocy klawiatury
3. Opis gry będzie podany po ćwiczeniach

Outline of the previous exercise

Podstawy JavaScript

4. Wyprowadzanie / wprowadzanie danych

 **1 pkt** za wykonanie wszystkich zadań z tej sekcji

1. Utwórz dokument HTML zawierający następujący kod:

```
<html>
  <body>
    <div>Treść dokumentu HTML przed skryptem</div>
    <script>
      console.log('Tekst 1');
      window.alert('Tekst 2');
      document.write('Tekst 3');
    </script>
    <div>Treść dokumentu HTML po skrypcie</div>
  </body>
</html>
```

Copy

2. Uruchom przeglądarkę WWW, a następnie [otwórz jej konsolę](#)
3. Załaduj powyższy dokument w bieżącej zakładce przeglądarki WWW
4. Spróbuj zlokalizować miejsce pojawiania się tekstów: Tekst 1, Tekst 2 oraz Tekst 3
5. Utwórz alternatywną wersję dokumentu HTML, w której to skrypt będzie wywoływany po zakończeniu renderowania dokumentu HTML, a nie w trakcie (renderowania) — utwórz, a następnie załaduj, następujący dokument HTML:

```
<html>
  <body onLoad="
    console.log('Tekst 1');
    window.alert('Tekst 2');
    document.write('Tekst 3');
  ">
    <div>Treść dokumentu HTML</div>
    <div>Treść dokumentu HTML</div>
    <div>Treść dokumentu HTML</div>
  </body>
</html>
```

Copy

6. Jak myślisz, dlaczego w tej wersji, treść dokumentu HTML nie jest widoczna w przeglądarce — usuń linię, która jest tego przyczyną
7. Dopisz w ciele dokumentu:

```
<script>
  window.prompt("Tekst1", "Tekst2");
</script>
```

Copy

8. Zbadaj, jakie znaczenie mają poszczególne argumenty metody `window.prompt()` i czy są one obowiązkowe
9. Zbadaj, co jest zwracane (wartość, [typ danych](#)) w przypadku:
 - o Wprowadzeniu wartości badanej liczba i naciśnięciu klawisza 'Enter' lub przycisku 'OK'

- Wprowadzeniu wartości będącej napisem i naciśnięciu klawisza Enter lub przycisku "OK"
- Niewprowadzeniu wartości i naciśnięciu powyższego klawisza / przycisku

- Wprowadzeniu wartości i naciśnięciu przycisku 'Anuluj'

10. Dopisz w obrębie elementu "body"

```
<form>
  <input name="pole_tekstowe" type="text">
  <input name="pole_liczbowe" type="number">
  <input type="button" value="Wypisz">
</form>
```

Copy

11. Korzystając z odpowiedniego [obserwatora zdarzeń](#) postaci 'on*NazwaZdarzenia*', kolekcji [DOM 0](#), takich jak: [document.forms\[\]](#) oraz [document.forms\[\].elements\[\]](#), spowoduj, aby po kliknięciu przycisku "Wypisz" wyświetliła się zawartość wprowadzona w polu tekstowym oraz liczbowym formularza
12. Zbadaj, co jest zwracane (wartość, typ danych) w przypadku:
 - Wprowadzeniu wartości będącej liczbą i naciśnięciu powyższego przycisku
 - Wprowadzeniu wartości będącej napisem i naciśnięciu w/w przycisku
 - Niewprowadzeniu wartości i naciśnięciu przycisku "Wypisz"
13. Przenieś całą zawartość (kod JS) elementu "script" do osobnego pliku "script.js" (utwórz zewnętrzny skrypt JS), a następnie załaduj ten skrypt z poziomu dokumentu HTML

5. Testy

1. Utwórz dokument HTML o nazwie 'test.html' i poniższej zawartości

```
<!-- Źródło / Source: https://mochajs.org/#running-mocha-in-the-browser -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>
      Mocha tests
    </title>
    <meta name="viewport"
      content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
      href="https://unpkg.com/mocha/mocha.css">
  </head>
  <body>
    <div id="mocha">
    </div>
    <script src="https://unpkg.com/chai/chai.js"></script>
    <script src="https://unpkg.com/mocha/mocha.js"></script>
    <script class="mocha-init">

      mocha.setup('bdd');
      mocha.checkLeaks();
    </script>
    <script src="script.js"></script>
    <script class="mocha-exec">

      mocha.run();
    </script>
  </body>
</html>
```

Copy

2. Zastąp zawartość pliku 'script.js' następującą:


```
var expect = chai.expect;

function sum(x,y) {
    return x+y;
}

describe('The sum() function', function() {
    it('Returns 4 for 2+2', function() {
        expect(sum(2,2)).to.equal(4);
    });
    it('Returns 0 for -2+2', function() {
        expect(sum(-2,2)).to.equal(0);
    });
});
```

Copy

3. Otwórz dokument 'test.html' w przeglądarce WWW

4. Przeczytaj [opis](#) tworzenia testów w oparciu o [Mocha](#) oraz [Chai](#)

5. 🏠 **1 pkt** za wykonanie poniższych zadań:

1. Włącz [tryb ścisły](#).

2. Napisz skrypt, który za pomocą metody `window.prompt()` wczytuje dane (napis), przekazuje je do poniższych funkcji, a następnie wypisuje na bieżąco (po każdym wczytaniu danych):

`\twynik_działania_funkcji_cyfry\twynik_działania_funkcji_litery\twynik_działania_funkcji_suma`

Przykład działania:

```
111
  3  0 111 <Suma=111<Suma=111
11aa
  2  2 122 <Suma=111+11<Suma=111+11
b3345a
 15  2 122 <Suma=111+11+0<Suma=111+11+0
```

Uwagi i informacje:

- Funkcja `cyfry(napis)`, dla wczytanego napisu, oblicza, a następnie zwraca **sumę** zawartych w nim cyfr
- Funkcja `litery(napis)`, oblicza, a następnie zwraca **ilość** zawartych w nim liter
- Funkcja `suma(napis)`, oblicza, na bieżąco, **sumę wszystkich** wczytanych liczb, o ile napis wygląda jak liczba, tzn. rozpoczyna się od ciągu cyfr (patrz linia 3 przykładu) lub zawiera same cyfry (patrz linia 1 przykładu)
- Wczytywanie danych ma się odbywać do momentu naciśnięcia przycisku 'Anuluj'

3. Zaimplementuj test, w oparciu o Mocha, weryfikujący poprawność działania powyższych funkcji, dla napisów zawierających:

- Same cyfry
- Same litery
- Litery, a po nich cyfry
- Cyfry, a po nich litery
- Pusty napis

6. 🖨️ **(2 pkt.)** Dokument HTML zawiera formularz. Napisz skrypt **(1.5 pkt.)** oraz towarzyszący mu test Mocha **(0.5 pkt.)**. Założenia dla skryptu:

- Po kliknięciu [przycisku formularza](#) skrypt wczytuje dane zawarte w obszarze tekstowym ([textarea](#)), korzystając z obiektów/kolekcji 'document.forms' oraz 'elements'
- Używa kolekcji [indeksowanych](#) lub kluczowanych ([Map / Set](#), [SessionStorage / localStorage](#)) przechowujących dane typów [prostych](#) lub [obiektowych](#)
- Realizuje funkcjonalność podaną na początku ćwiczeń

6. 🚩 (2 pkt.) Dokument HTML zawiera formularz. Napisz skrypt (1.5 pkt.) oraz towarzyszący mu test Mocha (0.5 pkt.). Założenia dla skryptu:
- o Po kliknięciu przycisku formularza skrypt wczytuje dane zawarte w obszarze tekstowym (textarea), korzystając z obiektów/kolekcji 'document.forms' oraz 'elements'
 - o Używa kolekcji indeksowanych lub kluczowanych (Map / Set, sessionStorage / localStorage) przechowujących dane typów prostych lub obiektowych
 - o Realizuje funkcjonalność podaną na początku ćwiczeń
 - o Wypisuje wynik

Proszę napisać skrypt do tworzenia systemu rejestracji samochodów

- Strona WWW zawiera formularz, którego zawartością są:
 - Obszar tekstowy
 - Przycisk zwykły "Dodaj"
 - Przycisk (zwykły) "Wypisz"
 - Pole wyboru (Checkbox) "Session Storage"
- Aplikacja ma umożliwiać:
 - Dodanie nowego samochodu wraz z polami (nazwa właściciela, numer rejestracyjny, itp.)
 - Wyszukiwanie samochodów i wypisanie informacji o nim.
- Dane wejściowe specyfikujemy w obszarze tekstowym — format:

Wstawianie danych

```
add: pole1=wartość1, pole2=wartość, ...
add: pole1=wartość1, pole2=wartość, ...
...
```

Wyszukiwanie

```
find: pole1=wartość1, pole2=wartość, ...
...
```

- Jeżeli checkbox "Session Storage" **nie jest** zaznaczony, to podstawową kolekcją kluczowaną jest Map / Set
- Jeżeli checkbox "Web Storage" **jest** zaznaczony to podstawową kolekcją kluczowaną jest sessionStorage / localStorage
- Bez względu na stan checkbox-a, podstawową kolekcją indeksowaną jest Array

6. Tworzenie dynamicznych grafik

1. Utwórz dokument HTML o nazwie 'rysunek.html' i poniższej zawartości

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8">
    <title>
      Page title
    </title>
    <script>
      var canvas = document.getElementById('canvas');
      var ctx = canvas.getContext('2d');
      ctx.fillText("Hello World", 10, 50); //Wykreślenie podanego tekstu na płótnie / Drawing given text on canvas
    </script>
  </head>
  <body>
    <main>
      <h1>
        Płótno
      </h1>

      <h1>
        Canvas
      </h1>

      <canvas id="canvas"
        width="200"
        height="300"
        style="border:1px solid #000000;">
        Wygląda na to, że twoja przeglądarka nie obsługuje elementu "canvas" / It looks like your browser does not
        support the "canvas" element
      </canvas>
    </main>
  </body>
</html>
```

Copy

2. Dlaczego na powierzchni [płótna](#) nie pojawił się napis "Hello World"? — zobacz jaki komunikat wyświetla się w konsoli przeglądarki WWW (**Ctrl+Shift+I**); spróbuj wprowadzić odpowiednie modyfikacje, tak aby powyższe instrukcje zadziałały
3. 🏠 (1 pkt) Narysuj trzy figury geometryczne: koło, trójkąt oraz prostokąt
4. 🚩 (2 pkt.) Stwórz bibliotekę do wizualizacji danych — szczegóły zostaną podane po zakończeniu zajęć

1. **Bez użycia dodatkowych bibliotek** stwórz zestaw funkcji do wizualizacji danych w postaci, wybranych przez Ciebie, **dwóch rodzajów wykresów** (np. słupkowego, tortowego, ...) — stwórz bibliotekę do tworzenia wykresów
2. Zadbaj o to, aby wykres, zawsze, mieścił się w obszarze płótna (ang. canvas), bez względu na wielkość prezentowanej wartości liczbowej, jak i ekranu użytkownika
3. Staraj się zaimplementować swój skrypt tak, aby dało się go, w przyszłości, używać w innych swoich projektach, np.

4. Utwórz przykładowe wykresy (statyczne) pokazujące możliwości Twojej biblioteki
5. Pokaż działanie biblioteki dla przypadku danych modyfikowanych "on-line" — na przykład, wykres obrazujący aktualną ilość elementów w poszczególnych kolekcjach (indeksowanych oraz kluczowanych), które zostały utworzone w ramach zadania ćwiczeniowego
6. Przeczytaj opis poświęcony opisowi [modułów JavaScript](#)
7. Wprowadź podział na moduły, np.: moduł główny, moduł tworzenia wykresu słupkowego, moduł tworzenia wykresu warstwowego, itp. i zastosuj opisany mechanizm w stworzonej bibliotece