Відокремлений підрозділ Національного університету
біоресурсів і природокористування України
„Бережанський агротехнічний інститут”

Кафедра інформаційних технологі
та вищої математики

ВІЗУАЛІЗАЦІЯ ДАНИХ ЗАСОБАМИ CANVAS API

(Методичні рекомендації для самостійної роботи здобувачів вищої освіти за спеціальністю 122 – Комп’ютерні науки з освітньої компоненти «Web-технології та Web-дизайн»)

Бережани 2021

Качурівський В.О., канд.пед.наук, доцент кафедри інформаційних технологій та вищої математики Бережанського агротехнічного інституту.

Рекомендовано до друку вченою радою відокремленого підрозділу Національного університету біоресурсів і природокористування України „Бережанський агротехнічний інститут" (Протокол № __ від __.____.20___ р.)

Висвітлено питання побудови діаграм із написання коду відповідного JavaScript сценарій, який буде здійснювати автоматичний підбір ширини полотна побудови, використовувати числові дані для побудови діаграми та здійснювати анімаційну побудову.

Розглянуто та схвалено на засіданні кафедри інформаційних технологій та вищої математики Протокол № _9_ від 28.05.2021р.

Рецензенти: Мартинюк С.В., канд.фіз.-мат.наук, доц.
Струбицька І.П., канд.техн.наук, доц..

Зміст

ВСТУП

Одним із способів візуалізації інформації, який набув широкого застосування, є використання різного виду діаграм: стовпчикова гістограма, кругова діаграма, двовимірний графік тощо. Популярним засобом створення діаграм є додаток Microsoft Office Excel. Створену та форматовану діаграму зберігають у файлі графічного типу для подальшого використання. Такі діаграми є інформативними та сприяють позитивному сприйняття інформації.

Поряд з простотою та перевагами цього способу, він має ряд недоліків, а саме:

- створені діаграми є статичними, тобто із зміною числової інформації графік незмінний;
- розміри файлу зображення є достатньо великим, тому з використанням декількох зображень, значно збільшується час на завантаження web-сторінки;
- відкритим залишається питання адаптації відображення інфографіки до різних видів мобільних пристроїв та інтернет-браузерів.

Вирішити дану проблему можна із використанням графічних можливостей HTML5, зокрема програмованої графіки Canvas API.

Для побудови діаграм необхідно створити відповідний JavaScript сценарій, який буде здійснювати автоматичний підбір ширини полотна побудови, використовувати числові дані для побудови діаграми та здійснювати анімаційну побудову.

Мета роботи і задачі дослідження. Метою роботи є дослідження та розробка JavaScript сценарію для реалізації способів анімації діаграм, які є структурним елементом HTML-сторінки. Для реалізації даних мети визначено такі задачі:

1. За необхідність визначити спосіб адаптації полотна побудови діаграми до ширини екрану браузера та мобільних пристроїв.
2. Розробити методику масштабування величини числових даних до висоти полотна побудови графіка.
3. Визначити та реалізувати способи програмної анімації складових елементів діаграми.

Об'єкт дослідження. Canvas API, як засіб для малювання графіки через JavaScript та HTML

Предмет дослідження. Анімаційна побудова стовпчикової, кругової та кільцевої діаграм.

РОЗДІЛ 1. ПОБУДОВА ЗОБРАЖЕНЬ

1.1. Підготовка полотна до побудови.

Розпочнемо з розгляду елемента <canvas> HTML.
<canvas id="graph " width="150" height="150"></canvas>

На перший погляд <canvas>виглядає як <img>елемент, з тією лише явною відмінністю, що він не має атрибутів. Дійсно, <canvas> елемент має лише два атрибути, width та height. Вони є необов'язковими, і їх також можна встановити за допомогою властивостей DOM . Якщо не вказано атрибути то полотно спочатку буде мати 300 пікселів у ширину та 150 пікселів у висоту. Елемент може бути розміром довільно за допомогою CSS , але під час відтворення зображення масштабується відповідно до розміру макета: якщо розмір CSS не відповідає співвідношенню початкового полотна, він буде виглядати спотвореним.width height.

id Атрибут не є специфічним для <canvas> елемента , це однин з глобальних HTML атрибутів, які можуть бути застосовані до будь-якого HTML - елементу (як class, наприклад). Правилами хорошого тону програмування є надати елементу, id оскільки це значно полегшує його ідентифікацію в сценарії.

<canvas>- елемент може бути стилізована так само, як будь-який елемент правилами (margin, border, background...). Однак ці правила не впливають на фактичний малюнок на полотні. Якщо до полотна не застосовуються правила стилізації, воно спочатку буде повністю прозорим.

Полотно спочатку порожнє. Щоб щось відобразити, сценарію спочатку потрібно отримати доступ до контексту візуалізації та намалювати його. <canvas> Елемент має метод , званий getContext(), який використовується для отримання контексту рендеринга і його функції малювання. getContext() приймає один параметр, тип контексту. Для 2D-графіки, наприклад, тієї, ви вказуєте, "2d" щоб отримати CanvasRenderingContext2D.

```
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
```

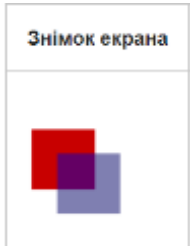
Перший рядок у сценарії отримує вузол у DOM, що представляє <canvas> елемент, викликаючи document.getElementById() метод. Отримавши вузол елемента, ви

можете отримати доступ до контексту малювання, використовуючи його getContext() метод.

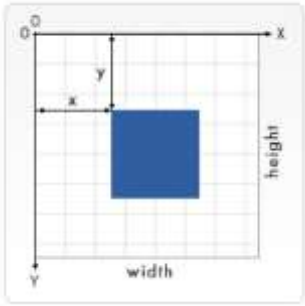
Для початку давайте розглянемо простий приклад, який малює два прямокутники, що перетинаються, один з яких має альфа-прозорість. На наступних прикладах ми детальніше дослідимо, як це працює.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
    function draw() {
      var canvas =
document.getElementById('canvas');
      if (canvas.getContext) {
        var ctx = canvas.getContext('2d');

        ctx.fillStyle = 'rgb(200, 0, 0)';
        ctx.fillRect(10, 10, 50, 50);

        ctx.fillStyle = 'rgba(0, 0, 200, 0.5)';
        ctx.fillRect(30, 30, 50, 50);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150"
height="150"></canvas>
 </body>
</html>
```

Зображення виглядає так



1.2. Побудова елементарних геометричних фігур



Перш ніж почати малювати, нам слід поговорити про сітку полотна або про координатний простір. Наш код HTML із попередньої сторінки мав елемент полотна шириною 150 пікселів та висотою 150 пікселів. Справа ви бачите це полотно із накладеною сіткою за замовчуванням. Зазвичай 1 одиниця в сітці відповідає 1 пікселю на полотні. Початок цієї сітки розміщений у верхньому лівому куті на координаті (0,0). Всі елементи розміщуються відносно цього походження. Отже, положення верхнього лівого кута синього квадрата стає x пікселів зліва та y пікселів зверху за координатою (x, y). Пізніше ми побачимо, як ми можемо перевести початок координат в інше положення, повернути сітку і навіть масштабувати її, але наразі ми дотримуватимемось типового.

Малювання прямокутників

На відміну від SVG , <canvas> підтримує лише дві примітивні фігури: прямокутники та контури (списки точок, з'єднаних лініями). Всі інші фігури повинні бути створені комбінуванням одного або декількох контурів. Ми маємо набір функцій для малювання контурів, які дозволяють складати дуже складні фігури.

Спочатку давайте розглянемо прямокутник. Є три функції, які малюють прямокутники на полотні:

`fillRect(x, y, width, height)`- малює заповнений прямокутник.
`strokeRect(x, y, width, height)`- малює прямокутний контур.
`clearRect(x, y, width, height)`- очищає вказану прямокутну область, роблячи її повністю прозорою .

Кожна з цих трьох функцій приймає однакові параметри, де x і y положення на полотні (відносно початку координат) верхнього лівого кута прямокутника. `Width` і `height` розмір прямокутника.

Приклад прямокутної форми

```
function draw() {
 var canvas= document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');

    ctx.fillRect(25, 25, 100, 100);
    ctx.clearRect(45, 45, 60, 60);
    ctx.strokeRect(50, 50, 50, 50);
  }
}
```

Малювання доріжок

А тепер давайте розглянемо шляхи. Шлях - це список точок, з'єднаних відрізками ліній, які можуть бути різної форми, вигнуті чи ні, різної ширини та різного кольору. Шлях можна закрити. Щоб зробити фігури за допомогою контурів, ми робимо кілька додаткових кроків:

- Спочатку ви створюєте шлях.
- Потім ви використовуєте команди малювання, щоб намалювати шлях.
- Щойно шлях буде створений, його можна обвести або заповнити, щоб відтворити його.

Функції, які використовуються для виконання цих кроків:

`beginPath()`

Створює новий шлях. Одного разу створені майбутні команди малювання спрямовуються у шлях та використовуються для побудови шляху.

Методи шляху

Методи встановлення різних шляхів для об'єктів.

`closePath()` - додає пряму лінію до шляху, що переходить до початку поточного під-шляху.

`stroke()` - малює фігуру, погладжуючи її контур.

`fill()` - малює суцільну форму, заповнивши область вмісту контуру.

Першим кроком для створення шляху є виклик `beginPath()`. Внутрішньо шляхи зберігаються як список під-шляхів (ліній, дуг тощо), які разом утворюють фігуру. Щоразу, коли цей метод викликається, список скидається, і ми можемо почати малювати нові фігури.

Переміщення пера

Однією дуже корисною функцією, яка насправді нічого не малює, але стає частиною списку шляхів, описаного вище, є `moveTo()` функція. Це як про підняття ручки або олівця з одного місця на аркуші паперу та розміщення його на наступному.

`moveTo(x, y)` – переміщує перо за координатами, заданими x та y.

Коли полотно ініціалізується або `beginPath()` викликається `moveTo()`, щоб розмістити початкову точку в іншому місці. Ми також могли б використовувати `moveTo()` для малювання несполучених шляхів.

Лінії

Для малювання прямих ліній використовуйте `lineTo()` метод.

`lineTo(x, y)` - проводить лінію від поточної позиції креслення до позиції, вказаної x та y.

Цей метод приймає два аргументи x і y, які є координатами кінцевої точки рядка. Початкова точка залежить від раніше намальованих шляхів, де кінцева точка попереднього шляху є початковою для наступних і т. д. Початкову точку також можна змінити за допомогою `moveTo()` методу.

У наведеному нижче прикладі намальовано два трикутники, один заповнений і один окреслений.

```
    function draw() {
      var canvas =
document.getElementById('canvas');
      if (canvas.getContext) {
        var ctx = canvas.getContext('2d');

        // Filled triangle
        ctx.beginPath();
        ctx.moveTo(25, 25);
        ctx.lineTo(105, 25);
        ctx.lineTo(25, 105);
        ctx.fill();

        // Stroked triangle
        ctx.beginPath();
        ctx.moveTo(125, 125);
```

```
        ctx.lineTo(125, 45);
        ctx.lineTo(45, 125);
        ctx.closePath();
        ctx.stroke();
      }
    }
```

Усе починається з виклику `beginPath()` для початку нового шляху фігури. Потім ми використовуємо `moveTo()` метод, щоб перемістити початкову точку у бажане положення. Під цим проведено дві лінії, які складають дві сторони трикутника.

Дуги.

Щоб намалювати дуги або кола, ми використовуємо методи `arc()` або `arcTo()`.

`arc(x, y, radius, startAngle, endAngle, counterclockwise)`

Малює дугу, яка відцентрована в положенні (x, y) з радіусом r, починаючи з `startAngle` і закінчуючи `endAngle`, рухаючись у заданому напрямку, вказаному проти годинникової стрілки (за замовчуванням за годинниковою стрілкою).

`arcTo(x1, y1, x2, y2, radius)`

Малює дугу із заданими контрольними точками та радіусом, з'єднану з попередньою точкою прямою лінією.

Давайте детальніше розглянемо `arc` метод, який приймає шість параметрів: x і y є координатами центру кола, на якому слід намалювати дугу. `Radius` зрозуміло. Параметри `startAngle` і `endAngle` визначають початкову і кінцеву точки дуги в радіанах вздовж кривої кола. Вони вимірюються від осі x. `counterclockwise` - параметр є логічне значення, яке, коли true, малює дуги проти годинникової стрілки; в іншому випадку дугу малюють за годинниковою стрілкою.

Примітка : Кути у `arc` функції вимірюються в радіанах, а не градусах. Щоб перетворити градуси в радіани ви можете використовувати такий вираз JavaScript:

`radians = (Math.PI/180)*degrees.`

РОЗДІЛ 2. ПОБУДОВА АДАПТИВНОЇ, СТАТИЧНОЇ ІНТЕРАКТИВНОЇ ДІАГРАМИ.

2.1. Адаптація полотна побудови до ширини вікна браузера.

Розробниками HTML5 додано новий елемент `<canvas>`, який призначений для створення графічного полотна та побудови зображень за допомогою сценаріїв у JavaScript [2].

Основними етапами створення графіки є:

1. Створення графічного полотна в HTML-документі за допомогою парного тегу.

```
<canvas id=”Your name”></canvas>
```

2. Написання JavaScript сценарію для побудови графічних зображень. Кожен сценарій, який забезпечує побудову певного виду діаграми, повинен мати свій унікальний ідентифікатор. Наприклад: gistogram().

3. Виконання JavaScript сценарію побудови зображень при активації документа.

```
<script>
window.onload = gistogram(); //виклик сценарію
побудови діаграми
</script>
```

Розглянемо детальніше, на прикладі побудови вертикальної стовпчикової діаграми, на основі одного числового ряду.

Canvas використовує полотно яке складається з множини графічних точок – пікселів. Розміри полотна в пікселях визначаються атрибутами `width` та `height` тегу `<canvas>`. За замовчуванням встановлюється ширина полотна – 300 px, а висота – 150 px. Задавши абсолютні розміри ми прив’язуємо полотно до визначеного відображення на екрані. При перегляді сторінки на екранах мобільних пристроїв з меншою кількістю пікселів, ніж ширина полотна, зображення виходить за межі екрану. Постає питання, яким чином адаптувати ширину полотна до повноцінного відображення.

Одним із способів адаптації ширини полотна є зміна атрибута `width` тегу canvas до ширини контейнера, в якому відображається

полотно програмним способом при завантаженні web-сторінки. Розробники сайтів застосовують різноманітні способи адаптації сайту до відображення на екранах різних пристроїв. Вказати універсальний ідентифікатор чи клас контейнера, за яким можна визначити реальну ширину в пікселях, яка відведена на відображення інформації, неможливо. Ми пропонуємо помістити тег `<canvas>` у контейнер `<div>` з унікальним ідентифікатором. Для прикладу: `id="canvas_for_drawing"`, який і слугуватиме відправною точкою для визначення ширини та адаптації полотна під його ширину програмним способом. Фрагмент HTML-документа буде наступним [9].

```
<div id="canvas_for_drawing">
<canvas id="Your_name"></canvas>
</div>
```

При виклику JavaScript сценарію передбачено можливість передачі вхідних параметрів до самого коду. Таким способом передаємо ідентифікатори контейнера та графічного полотна в код `gistogram ('canvas_for_drawing', 'Your_name')`. В ім'я сценарію побудови внесемо формальні змінні `div_id` та `canvas_id`, які будуть приймати відповідні значення. Активізація JavaScript сценарію побудови задається у такий спосіб.

```
window.onload = gistogram('canvas_for_drawing','
Your_name');
```

До сценарію побудови записуємо відповідний фрагмент:

```
//ідентифікація полотна
var canvas = document.getElementById(canvas_id);
var ctx = canvas.getContext("2d");
```

Визначення ширини контейнера за формалізованою змінною `div_id` та встановлення нових параметрів проводимо так:

```
var cont= document.getElementById (div_id);
// визначення ширини контейнера та зменшення його
на 5%
var W=cont.offsetWidth-cont.offsetWidth*0.05;
var H=W/2;
```

```
// встановлення нових параметрів полотна
canvas.width = W; canvas.height = H;
```

Отже, в одному документі можна застосувати декілька полотен та один JavaScript сценарій для побудови одного типу діаграм з різними числовими даними.

2.2. JavaScript сценарій для відображення графіки.

Побудову самої діаграми потрібно розділити на декілька частин:

Частина 1: Побудова сітки діаграми (горизонтальних та/або вертикальних допоміжних ліній).

Частина 2: Задання кольору та числових даних для побудови діаграми.

Частина 3: Масштабування значень даних. для побудови.

Частина 4: Побудова на полотні геометричних фігур.

Частина 5: Підписи осі та ряду даних [7].

При адаптації полотна побудови до розмірів контейнера були використано локальні змінні W та H – фактичні розміри canvas. Ці змінні будуть використовуватися при побудові сітки та масштабування побудови геометричних фігур.

Частина 1: Побудова сітки діаграми (горизонтальних та/або вертикальних допоміжних ліній).

Для побудови сітки використано наступний код.

```
// вертикальні лінії
ctx.beginPath();ctx.lineWidth=1;
for(var t=W/10; t<W; t+=W/10)
{ctx.moveTo(t,0);ctx.lineTo(t,H-
20);ctx.stroke();}
// горизонтальні лінії
ctx.beginPath();
for(var v=H/4; v<H; v+=(H-25)/4)
{ctx.moveTo(0,v);ctx.lineTo(W,v);ctx.stroke();}
```

Десять допоміжних вертикальних та чотири горизонтальні лінії по всій ширині та усій висоті полотна. Змінивши значення знаменника у виразі W/10 або H/4, змінимо кількість допоміжних ліній. Для підписів осі X зарезервовано 20 точок, тому H – 20.

Частина 2. Для побудови інтерактивної діаграми необхідно передати до JavaScript сценарію код кольору та числові дані, на

основі яких будуються прямокутники. Код кольору елементів діаграми передаємо безпосередньо під час виклику сценарію як один з вхідних параметрів.

Передачу числових даних можна реалізувати трьома способами:

- передати конкретні числові дані безпосередньо при виклику сценарію,
- провести зчитування значень з файлу даних певного формату,
- отримати дані з таблиці, яка розміщена у HTML-документі, інше [8].

Передачу кольору та числових даних безпосередньо у сценарій побудови діаграми під час його виклику проводять таким чином:

```
gistogram('canvas_for_drawing','
Your_name','#770000',20,30,50,40);
```

У цьому випадку передано колір #770000 елементів діаграми та чотири числові значення 20 30 50 та 40.

Сценарій зберігає передані дані в асоційованому масиві `arguments`. Кількість переданих даних визначається конструкцією `arguments.length`. Індексація елементів масиву починається з нуля. Кількість елементів масиву 7. Передані числові дані у даному випадку знаходяться у елементах масиву `arguments[3], arguments[4], arguments[5], arguments[6].`

Передані числові дані сформуємо у масив y=[] .

```
//формування числового масиву
var d=arguments.length;var y=[];
for(i=3;i<d;i++) y[i-3]=arguments[i];
```

Кількість числових даних `var n=d-3;`

Частина 3: Масштабування значень даних.

Для побудови прямокутників припустимо, що одиниця числа відповідає одному пікселю. Оскільки значення числових даних можуть бути більшими від фізичної висоти полотна в пікселях H є необхідність ввести коефіцієнт масштабування k. Обчислення цього коефіцієнта проведемо таким чином: висоту полотна H поділимо на найбільше передане числове значення. Для визначення

максимального скористаємося загальновідомим алгоритмом пошуку найбільшого елемента масиву.

```
max=y[0];
for(i=1;i<n;i++){if(y[i]>max) max=y[i];}
k=(H-20)/(max+10);//коефіцієнт масштабування
```

Частина 4: Графічна побудова геометричних фігур (елементів діаграми)

Ширину прямокутників визначаємо як ww=W/n/2, а крок між прямокутниками як W/n. Побудову прямокутників проведемо за допомогою циклу, використовуючи масив числових значень y.

```
var x=W/n/4;var ww=W/n/2;
var colorf= arguments[2];
for(i=0;i<n;i++)
{ctx.fillStyle = colorf;
ctx.fillRect (x,H-y[i]*k-20,ww,y[i]*k);
x+=W/n;}
```

***Частина5*:** Підписи осі та ряду даних.

Вісь OX будуємо наступними фрагментом

```
ctx.strokeStyle = "rgb(1,1,1)";ctx.beginPath();
ctx.moveTo(0,H-20);ctx.lineTo(W,H-
20);ctx.stroke();
```

Підписи осі X та ряду даних проведемо у цьому ж циклі за допомогою такої конструкції

```
ctx.fillStyle ='#333333';
ctx.fillText(y[i],x+ww/n,H-y[i]*k-22);
ctx.fillText(i+1,x+ww/n,H-10);
```

У результаті, ми отримаємо таке зображення. (Рис.1.)

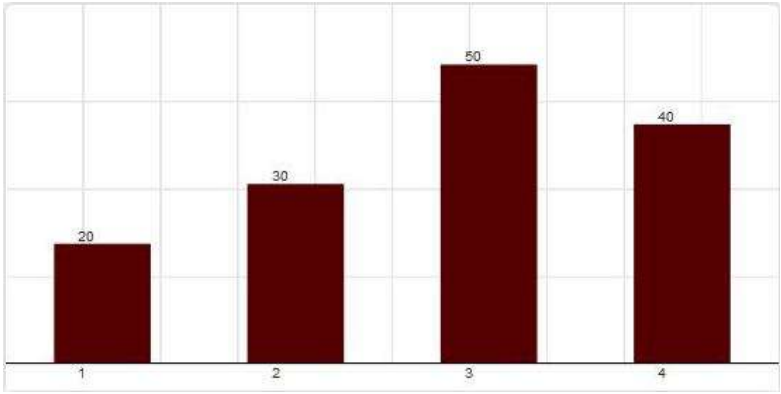


Рис.1. Адаптована стовпчикова діаграма

Результуючий JavaScript сценарій із використання зовнішнього файлу даних буде наступним [9].

```
/* Вертикальна діаграма*/
function
gistogram_d(div_id,canvas_id,colorf,path){
//ідентифікація полотна за id
var canvas = document.getElementById (canvas_id);
var ctx = canvas.getContext ("2d");
//вибір контейнера за id
var cont= document.getElementById (div_id);
var W=cont.offsetWidth-cont.offsetWidth*0.05;
var H=W/2;
//встановлення параметрів canvas
canvas.width = W; canvas.height = H;
ctx.strokeStyle = "rgb(231,231,231)";
//сітка
ctx.beginPath();ctx.lineWidth=1;
for(var t=W/10; t<W;t+=W/10)
{ctx.moveTo(t,0);ctx.lineTo(t,H-20);
ctx.stroke();}
ctx.beginPath();
for(var v=H/4; v<H; v+=(H-25)/4)
{ctx.moveTo(0,v);ctx.lineTo(W,v);ctx.stroke();}
//формування числового масиву
var i,max,k,d;var y=[];
var d=arguments.length;var y=[];
```

```
for(i=3;i<d;i++) y[i-3]=arguments[i];
//закінчення формування числового масиву
var n=d-3;max=y[0];
for(i=1;i<n;i++){if(y[i]>max) max=y[i];}
k=(H-20)/(max+10);
var x=(W/n)/4;var ww=W/n/2;
for(i=0;i<d;i++){
ctx.fillStyle = colorf;
ctx.fillRect (x,H-y[i]*k-20,ww,y[i]*k);
ctx.fillStyle ='#333333';
ctx.fillText(y[i],x+ww/n,H-y[i]*k-22);
ctx.fillText(i+1,x+ww/n,H-10);x+=W/n;}
ctx.strokeStyle = "rgb(1,1,1)";ctx.beginPath();
ctx.moveTo(0,H-20);ctx.lineTo(W,H-20);
ctx.stroke();}
```

Результат побудови анімаційної діаграми зображено на Рис.2.

Рис. 2. Зображення адаптованої стовпчикової гістограми

Кожному JavaScript сценарій для побудови певного виду діаграми необхідно присвоювати унікальний ідентифікатор, що дозволить організувати їх у зовнішній файл та підключати за необхідністю до конкретної web-сторінки для застосування.

2.3. Способи відбору даних до сценарії побудови графіки

Побудова діаграми засобами CANVAS API здійснюється на основі числових даних, сформованих у масиві, для прикладу `data`. Спосіб передачі числових даних безпосередньо при виклику JavaScript сценарію має певну незручність, а саме: при зміні числових даних необхідно проводити зміни в записаних параметрах виклику сценарію на нові числові дані в тексті Web-сторінки. Одним із способів вирішити дану проблему є реалізація відбору даних із зовнішнього файлу. Змінивши файл на новий, з актуальними даним, при активізації web-сторінки проводиться зчитування уже нових числових даних та здійснюється відповідна побудова діаграми.

Зосередимо свою увагу на формуванні числового масиву, використовуючи зовнішні джерела даних, який використовується для побудови діаграми.

Відбір числових даних із зовнішнього файлу.

Проаналізувавши різного типу файли збереження даних, зробили висновок, що найпростішим та зручним для формування числових даних є звичайний текстовий файл, створений в будь-якому текстовому редакторі з розширенням txt. Текстовий файл підпорядковується таким правилам запису: числові дані записуються у рядок, між даними ставиться розподільник «пробіл».

Наприклад, файл `data.txt` має такі числові дані: `20 30 50 40`

Заголовок JavaScript сценарію побудови діаграми буде наступним:

`function diagram_2(path)`, де `path` – формальна змінна, яка прийме значення визначеного шляху доступу до файлу даних при його виклику.

Виклик JavaScript сценарію буде таким: `diagram_2('.../data.txt');`

Фрагмент коду. Зчитування даних з файлу `data.txt` проведемо асинхронно замдопомогою технології AJAX.

```
var oAJAX= new XMLHttpRequest();
oAJAX.open("GET",path,false);
oAJAX.send();
```

Результатом роботи операторів буде тестовий рядок. Для розподілу даних скористаємося функцією `split()` із вказаним розподільником «пробіл», згідно до формату файлу даних.

```
var arr=oAJAX.responseText.split(' ');
```

Результатом роботи функції буде масив `arr` із текстовими значеннями.

Для перетворення символьних рядків у числові дані використаємо функції Number та наступну конструкцію.

```
var data=[];var j=0;
for(i=0;i<arr.length;i++)
if ( !isNaN(Number(arr[i].innerHTML)))
{data[i]=Number(arr[i]);j++;}
```

Кількість переданих числових значень визначається як `n=data.length;` Сформований числовий масив `data` використовується для побудови діаграми.

Відбір числових даних таблиці, вкладеної у HTML-сторінку.

Ще одним способом відбору даних для побудови діаграми є використання значень з таблиці, яка розміщена у HTML-документі. Таблиця формується парними тегами `<table> </table>`, `<tr> </tr>` та `<td> </td>`. Для реалізації вибірки числових даних таблиця повинна бути простою (відсутні об'єднання комірок). Також врахуємо такий факт, що перший стовпець або рядок таблиці містять текстові підписи.

Відбір даних з рядка таблиці

Зазвичай джерелом даних для побудови діаграми є певний рядок таблиці. Рядок даних таблиці формується парними тегами `<tr></tr>` та `<td></td>`. Таблиця може містити декілька рядків. Для однозначного визначення рядка таблиці з якого необхідно взяти дані для побудови діаграми, доцільно задати ідентифікатор рядка. Наприклад (фрагмент HTML-документа конструювання таблиці):

```
<table>
      <tr id="data1">
            <td>20</td>
            <td>30</td>
            <td>50</td>
            <td>40</td>
      </tr>
</table>
```

,де data1 – ідентифікатор рядка даних.

Заголовок JavaScript сценарію буде наступним:

function diagram_3(id_tr), де id_tr – формальна змінна-ідентифікатор рядка таблиці.

Виклик JavaScript сценарію буде таким: diagram_3('data1');

Фрагмент коду. Вибір числових даних здійснюється такою конструкцією.

```
var data=[];
var row=document.getElementById(id_tr);
cell = row.getElementsByTagName('td');
var n=cell.length;var j=0;
 for(i=0;i<n;i++)
if      (      !isNaN(Number(cell[i].innerHTML)))
{data[j]=Number(cell[i].innerHTML);j++;}
```

Примітка: isNaN()- перевірка чи є вміст комірки числом.

Числовий масив data сформований і може використовуватися для побудови.

Відбір числових даних із вказаного стовпця таблиці

Враховуючи тегову структуру побудови таблиці, однозначної ідентифікації стовпця, з якого необхідно взяти дані для побудови діаграми, провести неможливо. Для програмування відбору даних слід скористалися такою гіпотезою:

Із колекції вибраних комірок (за тегом <td>) необхідно вибирати кожну k-ту (k – кількість стовпців таблиці) починаючи з n-ої за індексом колекції.

Опишемо гіпотезу, детальніше:

1. Створити колекцію комірок таблиці;
2. Визначити кількість рядків та відповідно кількість стовпців;
3. Запустити процедуру відбору необхідних комірок даних.

Заголовок JavaScript сценарію буде наступним:

functiondiagram_4(id_table,n_coll),деid_table-

ідентифікатор таблиці, яка містить необхідні дані, n_coll – номер колонки, з якої вибирати числові дані.

Фрагмент коду. Вибір числових даних з стовпця здійснюється такою конструкцією.

```
var data_td=document.getElementById(id_table);
var cell=data_td.getElementsByTagName('td');
var n=cell.length;
var row = data_td.getElementsByTagName('tr');
var k=n/row.length;
var data=[];
var j=0;
for(i=n_coll-1;i<n;i+=k)
      if ( !isNaN(Number(cell[i].innerHTML)))
      {data[j]=Number(cell[i].innerHTML);j++;}
```

Числовий масив `data` сформовано.

РОЗДІЛ 3. АНІМАЦІЯ ПОБУДОВИ ДІАГРАМИ

3.1. Алгоритм анімації стовпчикової діаграми

У веб-дизайні графіки є одним з кращих інструментів для візуалізації даних. Застосування ефекту анімації до інфографіки затримує увагу користувача на ній та створює позитивний момент при перегляді текстової інформації. При побудові діаграм засобами CANVAS API доцільно розробити способи програмної анімації елементів діаграм.

Припустимо, що діаграма складається з чотирьох елементів. Визначимо такі сценарії анімації діаграм:

Перший спосіб – почергова побудова елементів діаграми. До кожного елемента діаграми застосуємо спосіб збільшення його розміру від 0 (нуль) до заданого числового значення. Спочатку будується перший елемент, за ним другий, третій та четвертий. Час побудови одного елемента діаграми залежить від фактичного числового значення, на основі якого будується елемент. Чим більше числове значення, тим довше будується елемент.

Другий спосіб – одночасна побудова елементів діаграми. Усі елементи будуються одночасно від нульового до відповідного числового значення масиву вхідних даних. Побудова закінчується тоді, коли вичерпано усі значення. Найдовше будується елемент із максимальним числовим значенням.

Розглянемо детальніше перший спосіб (почергова побудова елементів діаграми).

Алгоритм для створення анімації наступній:

1. Визначити, який складовий елемент діаграми буде програмно анімуватися.
2. Сформувати числовий масив показників, на основі яких будується діаграма. Про способи відбору числових значень до діаграми детальніше описано у праці [8]. Провести масштабування числових значень відповідно до висоти полотна побудови `canvas`. Масштаб визначається коефіцієнтом `k`, як співвідношення висоти полотна `H` до максимального числового значення.

3. Провести конструювання функції побудови статичного елемента діаграми. У параметрах функції необхідно передбачити передачу параметрів зміни розміру елемента.

4. Здійснити циклічне застосування методу `setTimeout()` для створення кадрування елемента діаграми та часову затримку у його відображенні. Це створить ефект анімації нашої інфографіки [6].

Даний алгоритм є універсальним та може бути застосований для довільного типу діаграми чи графіка.

3.2 Анімація стовпчикової діаграми

Розглянемо реалізацію даного алгоритму для побудови вертикальної гістограми. Елементами гістограми є прямокутники, які будуть створюватися командою `ctx.fillRect(x,y,w,h)` [3]. Необхідно провести програмування ***анімації одного стовпчика гістограми***, який буде основою для усієї інфографіки. Для прикладу візьмемо полотно canvas розміру `300px` на `150px`. Висоту полотна зафіксуємо у змінній `H=150`. Числове значення стовпчика гістограми записуємо у масив, наприклад: `var y=[300]`. Коефіцієнт масштабування `k` буде дорівнювати `k=H/300=0,5`. Формуємо новий масштабований масив значень `yy[0]=y[0]*k`.

Здійснюємо програмування функції користувача `plot` для побудови відображення прямокутника:

```
function plot(xx, hh) {
      ctx.fillStyle = '#67828E';
      ctx.fillRect(xx,H-hh,ww,hh);}
```

де – `xx` та `hh` формальні параметри початкової точки по осі X та висота прямокутника відповідно.

Здійснюємо побудову стовпчика зі зростання висоти `h` прямокутника від `1px` до масштабованого раніше значення `yy[0]`з кроком `1px`.

Для часової затримки у побудові наступного прямокутника, застосуємо метод `setTimeout()` [4]. Числове значення часової затримки для побудови наступного прямокутника формуємо у змінній `time`. Змінну часу побудови збільшуємо на `10мс` лічильником `time+=10`.

Фрагмент програмного коду для анімаційної побудови стовпчика гістограми є наступним:

```
var time=0;
for(h=1;h<=yy[0];h++)
   {time+=10;setTimeout(plot,time,x,h);}
```

Параметрами методу `setTimeout()` є: функція побудови прямокутника `plot`, змінна `time` та параметри для побудови `x` – координата по осі X та `h` – висота чергового прямокутника, якій передаються функції `plot`. Оскільки значення координати `x` є незмінною величиною, то прямокутники будуються таким чином, що кожен наступний перекриває попередній. Таким чином, створюється анімаційний ефект плавного зростання прямокутника.

Тепер розглянемо програмну реалізацію ***почергової побудови елементів гістограми***.

Припустимо, що наша гістограма повинна мати чотири стовпчики, які представляють такі числові дані: `200, 180, 100, 150.` Формуємо масив вхідних даних:

```
var y = [200, 180, 100, 150]; var n = y.length;
```

де `n` – кількість числових значень масиву.

Побудову елементів гістограми будемо здійснювати поступово від першого стовпчика до останнього за допомогою циклу. Для створення анімаційної побудови застосуємо змінну `time`, у якій будемо формувати час запізнення побудови наступного кадру анімації. Після побудови відповідного кадру збільшимо значення `time` на 5 мілісекунд. Дана змінна буде накопичувати числове значення запізнення для побудови другого, третього та четвертого стовпчиків.

Фрагмент коду почергової побудови стовпчиків гістограми з анімаційним ефектом зростання висоти кожного прямокутника буде наступнім.

```
var time = 0;
for(j=0; j<n;j++)
    for (h = 1; h<=yy[j]; h++) {
      time+=5;setTimeout(plot, time,  x, h, j);}
```

Розглянемо другий спосіб анімації діаграми: одночасна побудова елементів діаграми.

Для конкретизації JavaScript-сценарію розглянемо побудову гістограми на основі чотирьох числових значень.

Алгоритм побудови буде наступним:

1. Формування числового масиву показників гістограми. Припустимо `var y = [200, 180, 100, 150];`

2. Обчислення коефіцієнта масштабування. `k= H/max`, де `max` – найбільше числове значення масиву даних `max = Math.max(...);`

3. Формування масштабованого масиву для побудови діаграми

`var n=y.length; for(i=0; i<n; i++) yy[i]=y[i]*k;`

де `n=y.length` – кількість елементів масиву;

4. Зміна функції `plot()` для побудови чотирьох прямокутників. Побудову прямокутників визначимо у конструкції циклу.

```
function plot(xx, hh) {
for(j=0; j<n; j++)
 if(hh<=yy[j])
  {ctx.fillStyle='#67828E';
   ctx.fillRect(xx+j*70,H-hh,ww,hh);}}
```

Для зміщення побудови чергового прямокутника використаємо конструкцію `xx+j*70`, де `j*70` значення на зміщення. Число `70` – це крок зміщення по осі X.

5. Застосування методу `setTimeout()` для створення кадрування та анімації прямокутників:

```
for (h = 1; h<=max*k; h++ )
{time+=10;setTimeout(plot, time,  x, h);}
```

Щодо тривалості анімації. Оскільки висота побудови наступного прямокутник збільшується на одиницю, часова затримка збільшується на 10мс, то час анімації триватиме `max*k*10` секунд, що створює приємну та м'яку анімацію.

3.3. Анімація побудови секторної діаграми.

Програмна анімація секторної діаграми буде залежати від способу анімації:

1. Почергова побудова кожного повного сектора діаграми із затримкою визначеного часу.
2. Плавна побудова одного сектора діаграми з переходом до побудови кожного наступного сектора.

Побудова секторної діаграми здійснюється за допомогою команди побудови дуги `void ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise)[1];`

Для побудови секторів діаграми використаємо заливку `ctx.fill();` певним кольором `ctx.fillStyle = 'color';`

Для побудови кожного сектора діаграми необхідно мати початковий та кінцевий кути. Проведемо формування кутів секторів у масиві `angel`. Для зручності обчислення кутів, формуємо питому вагу кожного значення числового масиву, на основі яких необхідно будувати діаграму [1]. Для прикладу, необхідно побудувати секторну діаграму, яка буде мати шість секторів, які відповідають таким числовим даним; `14,9,7,20,28,34`. Знайдемо загальну суму.

```
var znach=[14,9,7,20,28,34];
var s=0; znach.forEach((item)=>{s+=item});
```

Для зручності обчислень – кути розраховуємо в діапазоні від 0 до 1. Перший кут рівний 0, кінцевий – 1. Кожен наступний кут побудови визначається, як попередній, до якого додано величину питомої ваги числового значення.

```
var n=znach.length; var angel=[];
angel[0]=0; angel[n+1]=1;
for(i=0;i<n;i++){
angel[i+1]=angel[i]+znach[i]/s;}
```

У результаті сформовано масив питомих значень кутів для побудови секторів діаграми.

Побудовн конкретного сектора здійснюємо командою `ctx.arc(x,y,r,angel[i]*Math.PI*2,angel[i+1]*Math.PI*2,false);` де `i` - це індекс сектора побудови. Побудова секторів діаграми здійснюється у циклі. Попередньо сформуємо кольорову заливку секторів у масиві `color_s`.

```
var color_s=['#FF7C00', '#BF7630',
      '#A65100','#FF9D40','#FFB773','#8F4600'];
```

Функція `plot_sector` () побудови одного сектора діаграми є такою:

```
function plot_sector(ii) {
ctx.beginPath();
```

```
ctx.fillStyle = color_s[ii];
ctx.moveTo(200,200); // центр дуги
ctx.arc(200,200,180,angel[ii]*Math.PI*2,angel[ii
+1]*Math.PI*2,false);
ctx.fill();}
```

де `ii` – це формальний параметр, який визначає номер сектора побудови.

Для анімації елементів діаграми застосуємо метод `setTimeout()`, до якого передаємо значення визначеного номера сектора та час затримки для побудови наступного. Побудова секторів проводиться циклічно від першого до шостого.

Програмна реалізація почергової побудови кожного повного сектора діаграми із затримкою визначеного часу буде такою.

```
var time=0;
  for (i= 0;  i<=n;  i++ ){
      time+=200;
      setTimeout(plot_sector, time, i);}
```

Більш привабливою буде анімація секторної діаграми з плавною побудовою одного сектора діаграми з переходом до побудови наступного сектора.

Для плавної побудови секторів діаграми будемо здійснювати побудову одиничного сектора величиною в 1 градус тобто $\pi/180$ радіан. Колір одиночного сектора буде визначатися поточним кольором до тих пір поки значення буде у кутових межах відповідного сектора, кути якого задані у масиві `angel`.

Умова для визначення кольору одиничного сектору є такою:

```
if(aa>=angel[i]*Math.PI*2&&aa<=angel[i+1]*Math.P
I*2)
c=i; ctx.fillStyle = color_s[c];
```

де `aa` – кут одиничного сектора.

Функція побудови сектора буде наступною:

```
function plot_sector(aa) {
ctx.beginPath ();
for(i=0;i<=n;i++)
if(aa>=angel[i]*Math.PI*2&&aa<=angel[i+1]*Math.P
I*2)
   c=i;
```

```
ctx.fillStyle = color_s[c];
ctx.moveTo(cen_x,cen_y);
ctx.arc(cen_x,cen_y,r,aa-Math.PI/180,aa,false);
ctx.fill();}
```

Побудова одиничного сектора здійснюється при зміні кута побудови від 0 до 360 градусів. Затримка побудови одиничного сектора 5 мілісекунд. Уся побудова триватиме близько двох секунд.

Код побудови кругової діаграми є таким:

```
var time=20;
  for (a= 0; a<=Math.PI*2;a+=Math.PI/360){
   time+=5;
   setTimeout(plot_sector,time,a);}
```

Результат роботи JavaScript сценарію побудови зображено на Рис. 3.

Анімація секторів діаграми

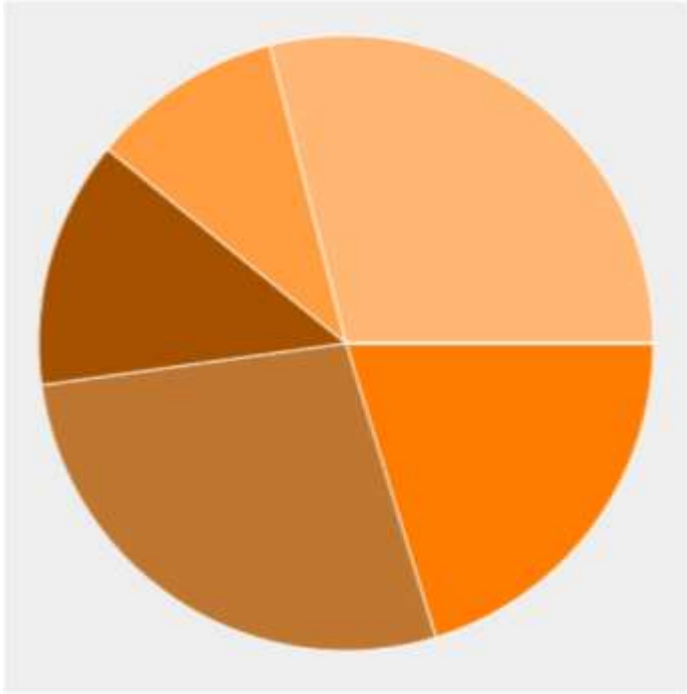


Рис. 3. Результат анімований вивід секторної діаграми.

Описані Javascript-сценарії є валідними та можуть бути застосовані у HTML документах з прив'язкою до певної події при формуванні аналітичних матеріалів.

3.4. Анімація кільцевої діаграми.

Складовими елементами кільцевої діаграми є дуги певної довжини. Для побудови дуги необхідно задати значення початкового та кінцевого кута. Оскільки кільцева діаграма відображає питому вагу певного числового показника від загальної сума показників, то необхідним є визначення суми та відносного значення показника від суми. Числові значення для побудови формуємо у масив даних та проводимо відповідні обчислення [5]. Способи відбору числових даних до масиву, для побудови діаграми із визначених джерел, детально описано у праці «Способи відбору даних до Javascript сценарію побудови діаграми» [8].

Розглянемо програмну реалізацію першого етапу на прикладі таких числових даних: 145, 125, 108, 144, 100. Числові дані формуємо у масив dat.

```
var dat=[145,125,108,144,100];
```

Знаходимо суму елементів масиву.

```
var s=0; dat.forEach((item)=>{s+=item});
```

Формуємо масив кутів ang для побудови секторів з врахуванням питомих значень кожного показника. Першим значенням кута є 0, а кінцеве значення 1. Кожен наступний кут – це попереднє значення плюс питоме значення з масиву dat.

```
var n=dat.length;
var ang=[];
ang[0]=0; ang[n+1]=1;
for(i=0;i<n;i++){
ang[i+1]=ang[i]+dat[i]/s;}
```

де n – кількість елементів масиву dat.

Розглянемо другий етап алгоритму – конструювання функції одиничної побудови складового елемента діаграми.

Для побудови застосуємо полотно `<canvas>` – `400px` на `400px`.

Елементом для побудови діаграми буде дуга в один градус. Кожна одинична дуга будується певним кольором. Для визначення кольору дуги використаємо конструкцію:

```
for(i=0;i<=n;i++)
if(aa>=ang[i]*Math.PI*2&&aa<ang[i+1]*Math.PI*2)
   c=i;
 ctx.fillStyle = color_s[c];
```

де `aa` – формальний параметр кута чергової побудови, а `color_s[c]` – масив кольорів дуг діаграми.

Наприклад:

```
var  color_s=['#FF7C00',  '#BF7630',  '#A65100',
'#FF9D40', '#FFB773'];
```

Кількість кодів кольорів відповідає кількості числових даних. При збільшення кількості чисел необхідно збільшити кількість кольорів у відповідному масиві.

Оскільки для побудови дуги необхідно оператору `ctx.arc()` передавати кут в радіанах, тому помножимо значення кута `ang[i]` на константу `Math.PI*2`.

Код функції побудови дуги в один градус буде таким:

```
function plot_arc(aa) {
ctx.beginPath ();
for(i=0;i<n;i++)
    if(aa>=ang[i]*Math.PI*2&&aa<=ang[i+1]*Math.PI
    *2) c=i;
ctx.strokeStyle = color_s[c];
ctx.arc(200,200,150,aa-Math.PI/180,aa,false);
ctx.lineWidth = 50;
ctx.stroke();}
```

Оператор `ctx.lineWidth = 50;` визначає товщину дуги побудови. Центр дуги становить `200px` на `200px`, а радіус побудови `150px`.

Розглянемо третій етап – анімаційна побудова усієї діаграми.

Для створення ефекту анімації застосуємо метод `setTimeout ()`. Даний методом виконує функцію або вказаний

фрагмент коду один раз, щойно спливе заданий час [4]. Для відтермінування побудови чергового елемента діаграми використаємо лічильник `time`. Даний лічильник буде накопичувати час запізнення при побудові наступного елемента. Початкове значення рівне лічильника встановимо 500, що дозволяє розпочати саму побудову трошки пізніше. Окрім того, метод `setTimeout ()`, дозволяє функції, яка викликається, передавити додаткові параметри. Таким параметром буде значення побудови чергової одиничної дуги діаграми.

Результат побудови кільцевої діаграми зображено на Рис. 4.

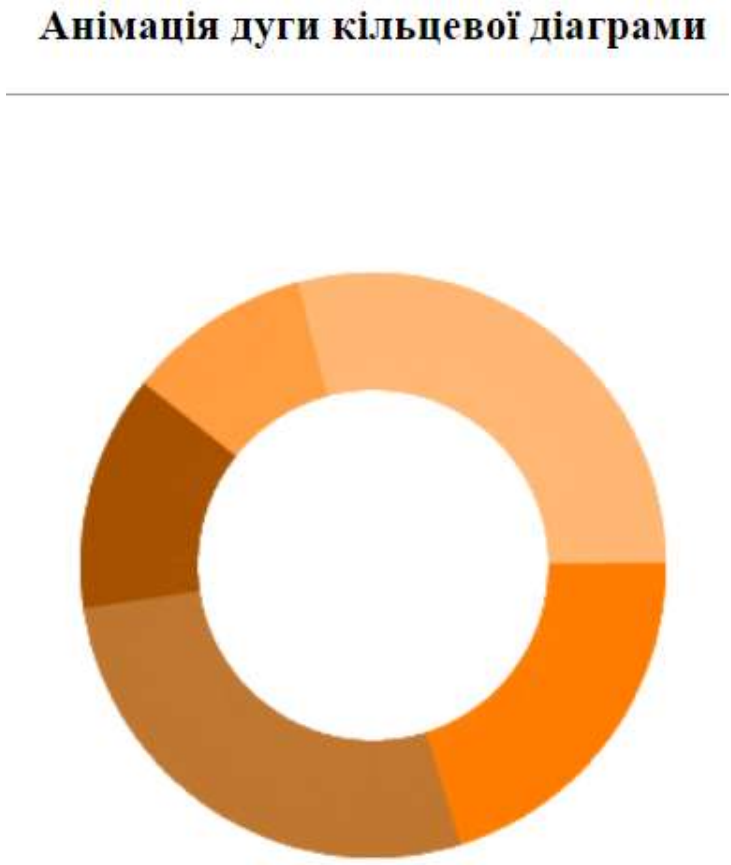


Рис. 4. Результат анімації кільцевої діаграми.

При зміні кута від 0 до 360 градусів викликаємо функцію побудови одиничної дуги та вказуємо час затримки побудови. Крок для зміни значень кута рівний одному градусу. Затримку побудови встановимо 5 мілісекунд. Уся побудова триватиме 360*5= 1800 мілісекунд [5].

Код побудови кільцевої діаграми є таким:

```
var time=500;
for ( a= 0;
      a<=2*Math.PI;
      a+=Math.PI/360 )
      { time+=5;
    setTimeout(plot_arc,time,a); }
```

Даний код можна використати для побудови анімації секторної діаграми. Для цього необхідно у функції побудови одиничної дуги plot_arc() встановити числове значення товщини лінії побудови у два рази більшими за значення радіуса дуги.

Наприклад:

```
ctx.arc(200,200,75,aa-Math.PI/180,aa,false);
ctx.lineWidth = 150;
```

З результатами роботи JavaScript сценаріїв побудови анімаційних побудов можна ознайомитися на онлайновій платформі Codepen за такими посиланнями.

- Анімація вертикальної гістограми https://codepen.io/volodimir-kachurwskij/pen/xxwzOoE
- Анімація секторної діаграми https://codepen.io/volodimir-kachurwskij/pen/oNjPaxo

САМОСТІЙНА РОБОТА

Завдання 1. На відрізку [-5;5] побудувати графік функції.
а) на площині окремо кожен з графік для завдань 1,16; 2,17 і т.д.;
б) на площині два графіки згідно завдань 1,16; 2,17 і т.д.;

[redacted]

Завдання №2. Побудувати стовпчикову діаграму для завдань 1,16; 2,17 і т.д

1) Залежність між зростом X (м) та масою дітей Y (кг) наведена в таблиці:

$Y = y_i$	0,620	0,580	0,640	0,650	0,670	0,680	0,695	0,699
$X = x_i$	0,531	0,524	0,541	0,550	0,559	0,620	0,632	0,672
$Y = y_i$	0,715	0,725	0,781	0,790	0,795	0,800	0,810	0,850
$X = x_i$	0,689	0,692	0,694	0,698	0,690	0,710	0,720	0,725

2) Зі старшого класу навмання вибраної середньої школи було відібрано групу учнів. Дані про їх середньорічні оцінки з математики X та решти дисциплін Y в балах наведено в таблиці:

$Y = y_i$	45	25	48	52	54	51	59	60	62	69
$X = x_i$	30	35	31	38	41	48	50	55	51	58
$Y = y_i$	72	78	76	80	82	85	81	90	93	95
$X = x_i$	60	59	65	73	78	71	79	80	81	82

3) Конденсатор було заряджено до повної напруги в певний момент часу t, після цього він починає розряджатися. Залежність напруги Y від часу розрядження X наведено в таблиці:

$Y = y_i$	100	85	70	65	60	55	50
$X = x_i$	0	1	2	3	4	5	6
$Y = y_i$	45	40	35	30	25	22	20
$X = x_i$	7	8	9	10	11	12	13

4) Залежність урожайності пшениці Y (ц/га) від глибини зволоження X (см) наведено в таблиці:

$Y = y_i$	10	12	14	16	18	20	22	24	26	28
$X = x_i$	0	5	8	10	12	14	16	18	20	22
$Y = y_i$	30	32	34	36	38	40	42	44	46	48
$X = x_i$	24	26	28	30	32	34	36	38	40	42

5) Показники товарообігу Y та суми витрат X, які досліджувалися в 20-ти магазинах, наведено в таблиці:

$Y = y_i$	480	510	530	540	555	564	570	575	580	585
$X = x_i$	30	25	31	32	38	41	40	46	49	54
$Y = y_i$	590	596	605	618	625	635	640	650	660	
$X = x_i$	58	60	64	75	78	82	83	85	90	

6) Результати вимірювання чутливості Y відеоканалу та звукового каналу X наведено в таблиці:

$Y = y_i$	240	200	190	180	170	160	150	140	130	120

$X = x_i$	170	180	200	230	240	250	280	300	310	320
$Y = y_i$	110	100	90	80	70	65	60	55	50	45
$X = x_i$	330	350	380	400	410	420	430	440	450	460

7) Залежність величини зносу різця Y від тривалості роботи X показано в таблиці:

$Y = y_i$	30,0	29,1	28,4	28,1	28,0	27,7	27,5	27,2	27,0
$X = x_i$	6	7	8	9	10	11	12	13	14
$Y = y_i$	26,8	26,5	26,3	26,1	25,7	25,3	24,3	24,1	24,0
$X = x_i$	15	16	17	18	19	20	21	22	23

8) Залежність кров'яного тиску Y людини (в умовних одиницях) від довжини руки X (см) наведена в таблиці:

$Y = y_i$	115	116	117	118	119	120	121	122	123
$X = x_i$	62,1	61,1	61,0	60,5	60,0	59,0	58,5	58,0	57,5
$Y = y_i$	124	125	126	127	128	129	130	135	150
$X = x_i$	56,5	56,0	55,5	55,0	54,5	54,0	53,5	53,0	52,5

9) Залежність пружності сталевих болтів Y від вмісту в них нікелю X наведена в таблиці:

$Y = y_i$	35,4	35,0	35,8	36,2	36,7	36,9	37,3	37,8	38,2
$X = x_i$	2,20	2,35	2,42	2,58	2,65	2,69	2,74	2,88	2,91
$Y = y_i$	39,1	40,5	42,4	43,8	45,6	46,9	48,5	49,4	50,0
$X = x_i$	2,95	2,99	3,00	3,11	3,21	3,29	3,34	3,44	3,50

10) Результати порівняння нового методу газового аналізу Y зі старим X наведено в таблиці:

$Y = y_i$	2,88	2,91	2,92	2,96	3,01	3,11	3,21	3,25
$X = x_i$	2,07	2,12	2,11	2,58	2,89	2,92	3,01	3,12
$Y = y_i$	3,32	3,36	3,42	3,46	3,58	3,88	4,12	
$X = x_i$	3,21	3,29	3,31	3,35	3,41	3,48	3,81	

11) Показники річної продуктивності праці в розрахунку на одного робітника Y і енергомісткості праці X на підприємствах однієї галузі наведено в таблиці:

$Y = y_i$	5,4	5,6	6,2	6,8	7,1	7,8	8,5	9,1	10,5
$X = x_i$	1,8	2,1	2,8	3,0	3,2	3,8	3,9	4,2	4,5
$Y = y_i$	11,0	11,6	12,1	12,7	13,2	13,9	14,1	14,6	14,9
$X = x_i$	5,2	5,8	5,9	6,2	6,9	7,2	7,5	8,5	8,8

12) Залежність денного споживання масла Y (г) певної особи від розміру її заробітної плати за місяць X (ум. од) наведено в таблиці:

$Y = y_i$	10,5	15,8	17,8	19,5	20,4	21,5	22,2	24,3	25,8
$X = x_i$	70	75	82	89	95	100	105	110	115
$Y = y_i$	28,1	30,1	35,2	36,4	37,0	38,5	39,5	40,5	41,0
$X = x_i$	125	130	135	140	145	150	155	160	165

13) Залежність маси монети Y від часу її обігу в роках X наведено в таблиці:

$Y = y_i$	9,35	9,21	9,18	9,50	9,10	9,08	9,05	9,01	9,00
$X = x_i$	4,0	5,0	5,5	6,0	6,8	7,5	8,5	10,8	12,0
$Y = y_i$	8,98	8,94	8,90	8,88	8,82	8,78	8,75	8,70	8,65
$X = x_i$	14,5	15,9	25,0	28,5	30,5	36,8	40,0	45,8	50,0

14) Залежність вмісту кремнію Y у чавуні від температури шлаку X наведено в таблиці:

$Y = y_i$	0,27	0,40	0,36	0,42	0,45	0,51	0,55	0,58	0,61
$X = x_i$	1330	1340	1350	1360	1370	1380	1390	1400	1410
$Y = y_i$	0,64	0,68	0,72	0,76	0,78	0,82	0,88	0,95	1,20
$X = x_i$	1420	1430	1440	1450	1460	1470	1480	1490	1500

15) Залежність міцності волокна бавовни в умовних одиницях Y від його товщини X (мкм) наведено в таблиці:

$Y = y_i$	6,02	6,12	6,22	6,28	6,30	6,35	6,39

$X = x_i$	0,41	0,48	0,56	0,66	0,72	0,79	0,85
$Y = y_i$	6,44	6,48	6,52	6,54	6,56	6,60	6,69
$X = x_i$	0,86	0,88	0,92	0,94	0,96	0,98	0,99

16) Залежність кількості проданих пар чоловічого взуття Y від його розміру X наведена в таблиці:

$Y = y_i$	10	25	68	136	152	162	170	180
$X = x_i$	44	43	42	41	40	39	38	37

17) Залежність граничного навантаження на болт Y від його твердості X наведено в таблиці:

$Y = y_i$	10,10	10,30	10,45	10,90	11,20	11,35	11,90	12,45
$X = x_i$	50,0	50,2	52,8	53,5	54,0	56,8	58,8	59,5
$Y = y_i$	12,96	13,44	13,60	13,95	14,50	14,98	15,48	15,96
$X = x_i$	64,8	65,4	68,4	69,2	70,5	74,5	76,8	78,5

18) Залежність урожайності пшениці Y (ц/га) від глибини оранки X (см) наведено в таблиці:

$Y = y_i$	7	8	9	10	11	12
$X = x_i$	8,1	8,3	8,2	9,1	10,3	10,8

19) Залежність вмісту свинцю Y в руді від вмісту X срібла наведено в таблиці:

$Y = y_i$	2,0	2,5	3,0	3,5	4,0	4,5	5,0	5,5	6,0
$X = x_i$	2,0	7,5	12,5	14,5	16,0	18,5	20,0	20,5	22,0
$Y = y_i$	7,0	7,5	8,0	8,5	9,0	10,5	12,5	14,5	15,0
$X = x_i$	26,0	28,5	30,0	32,5	34,0	36,5	38,0	40,5	42,0

20) Залежність числа гризунів Y, які загинули від наявності отрути в їжі при концентрації X, наведено в таблиці:

$Y = y_i$	32	36	38	42	46	49	55	59	62
$X = x_i$	3,0	3,5	4,0	4,5	5,0	6,0	6,5	7,0	7,5
$Y = y_i$	68	70	73	75	81	88	92	94	98
$X = x_i$	8,0	8,5	9,0	9,5	10,0	10,5	11,0	11,5	12,0

21) Виміри температури в грудні здійснені в двох містах, які умовно позначено А і В, наведено в таблиці:

$Y = y_i$	–10,2º	–11,5º	–12,4º	–12,8º	–13º	–13,5º	–14,2º	–14,6º
$X = x_i$	–20,2º	–20,5º	–21,4º	–21,8º	–22º	–22,5º	–22,8º	–22,8º

$Y = y_i$	–15,7º	–16,4º	–17,2º	–17,5º	–18,2º	–18,6º	–18,9º
$X = x_i$	–24,1º	–24,5º	–25,1º	–25,8º	–26º	–26,5º	–27º

22) Величину зносу різця Y (мм) від часу роботи X (год), наведено у таблиці:

$Y = y_i$	30	29,1	28,4	28,1	28	27,7	27,5	27,2	27
$X = x_i$	6	7	8	9	10	11	12	13	14
$Y = y_i$	26,8	26,5	26,3	26,1	25,7	25,3	24,3	24,1	24
$X = x_i$	15	16	17	18	19	20	21	22	23

23) Залежність врожайності Y (ц/га) пшениці від кількості внесених добрив X (кг/га), наведено у таблиці:

$Y = y_i$	10	12	14	16	18	20	22	24	26	28	30	32
$X = x_i$	10	30	40	50	60	70	80	90	100	110	120	130

24) Вплив температури Y (ºС) середовища на добовий хід хронометра X . наведено в таблиці:

$Y = y_i$	5	5,5	6	6,5	7	7,5	8	8,5	9	9,5	10
$X = x_i$	2,6	2,3	2,11	2,01	1,92	1,82	1,55	1,34	1,3	1,28	1,22

$Y = y_i$	10,5	11	11,5	12	12,5	13	14	18	24	30
$X = x_i$	1,18	1,12	1,1	0,98	0,92	0,9	0,89	0,88	0,8	0,79

25) Залежність кількості споживання масла на добу певною категорією пенсіонерів X (г) від розміру Y (грн) отриманої місячної пенсії, наведено в таблиці:

$Y = y_i$	290	380	490	540	620	700	790	980
$X = x_i$	15,99	19,75	23,1	26.44	29,79	33,13	36,89	44,54

26) Залежність місткості відсотків срібла у руді від місткості у відсотках свинцю, наведено у таблиці:

$Y = y_i$	2	6	10	14	18	22	26	30
$X = x_i$	2,5	7,5	12,5	17,5	22,5	27,5	32,5	37,5

27) Залежність Y (%) відсоткової місткості кремнію від температури X (ºС), наведена у таблиці:

$Y = y_i$	0,27	0,26	0,27	0,28	0,29	0,3	0,31	0,32	0,33
$X = x_i$	1330	1340	1350	1360	1370	1380	1390	1400	1410

28) Залежність міцності волокна бавовни Y (в умовних одиницях) від граничного навантаження X (г), наведено в таблиці:

$Y = y_i$	4100	4300	4500	4700	4900	5100	5200	5300	5500
$X = x_i$	3,75	4,25	4,75	5,25	5,75	6,25	6,75	7	7,25

29) Залежність врожайності цукрових буряків Y (ц/га) від кількості внесених у грунт поживних речовин X (кг/га), наведено у таблиці:

$Y = y_i$	369	380	370	395	420	412	436	420
$X = x_i$	83	92	112	132	144	154	162	189

30) Залежність між собівартістю X (тис.грн.) та кількістю виготовлених виробів Y (тис.шт) наведено в таблиці:

$Y = y_i$	2,2	3,5	3,7	3,8	4,5	5,7
$X = x_i$	1,5	1,4	1,2	1,1	0,9	0,8

ВИСНОВКИ

Підсумовуючи можна зробити такі висновки.

1. Основним методом автоматичного адаптування полотна побудови до ширини вікна браузера є використання властивості `offsetWidth` графічної блоку елемента об'єктної моделі DHTML. За допомогою методу доступу до елемента сторінки `getElementById` та властивості `Width` визначаємо нову ширину полотна побудови.

2. Для здійснення анімаційної побудови визначеного типу діаграми необхідно визначити одиничний елемент побудови. Застосовуючи розроблений алгоритм побудови, із застосування методу `setTimeout`, дозволить впровадити кадрування довільного графіка.

3. Наведені сценарії побудови є валідними, пройшли верифікацію на робочих сайтах закладів вищої освіти Відокремленого підрозділу Національного університету біоресурсів та природокористування України «Бережанський агротехнічний інститут», Відокремленого структурного підрозділу Національного університету біоресурсів та природокористування України «Бережанський фаховий коледж» а також медичного центру «Біомед» м.Бережани.

4. Додаткового дослідження потребують питання кросбраузерності розроблених сценарії побудови анімаційних діаграм та графіків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Canvas API. MDN web docs mozilla. URL: https://developer.mozilla.org/uk/docs/Web/API/Canvas_API (дата звернення: 1.02.2021)
2. HTML Canvas Graphics. w3schools.com. URL: https://www.w3schools.com/html/html5_canvas.asp (дата звернення: 1.02.2021)
3. Steve Fulton, Jeff Fulton. HTML5 Canvas: Native Interactivity and Animation for the Web. O'Reilly Media, 2011. 628p.
4. WindowTimers.setTimeout(). MDN web docs mozilla. https://developer.mozilla.org/uk/docs/Web/API/WindowTimers/setTimeout (дата звернення: 1.02.2021)
5. Вальський М.О. Анімаційна побудова дуги кругової діаграми засобами javascript. *Актуальні питання агропромислового комплексу*: зб. матеріалів учасн. V студентської науково–практичної конференції. Частина 1 «Електроенергетика, електротехніка та електромеханіка» / Бережани. ВП НУБІП України «Бережанський агротехнічний інститут, 2020. С. 66
6. Вальський М.О., Качурівський В.О. Анімація кільцевої діаграми за допомогою JavaScript сценарію засобами CANVAS API. Матеріали Міжнародної науково-практичної інтернет-конференції *«Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*: Зб. наук. праць. Переяслав, 2021. Вип. 67. С. 345-347.
7. Джош Мариначи. Практика: создание диаграмм. URL: https://webref.ru/dev/canvasdeepdive/chapter02 (дата звернення: 1.02.2021)
8. Качурівська Г., Качурівський В. Способи відбору даних до Javascript сценарію побудови діаграми. *«Інтернет-освіта-наука-2018»*: збірник праць одинадцята міжнар. наук-практ. конф. (Вінниця, 22-25 травня 2018 р.). Вінниця: ВНТУ, 2018. С.238-240.
9. Качурівський В.О. Побудова адаптивних та динамічних діаграм засобами CANVAS API. Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2019. С.132-137.

Методичне видання

Качурівський В.О.

ВІЗУАЛІЗАЦІЯ ДАНИХ ЗАСОБАМИ CANVAS API

(Методичні рекомендації для самостійної роботи здобувачів вищої освіти за спеціальністю 122 – Комп'ютерні науки з освітньої компоненти «Web-технології та Web-дизайн»)

Підписано до друку __.__.20__р. Вид № __
Формат 60x84 1/16 Папір офсетний.
Друк на різографі. Гарнітура Times/
Умовно-друк. арк. __ Облік-вид. арк. ____
Тираж 100 прим. Замовлення № ___.

Видавництво ВІКТ БАТІ
м. Бережани