МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет лінгвістики та соціальних комунікацій
Кафедра англійської філології і перекладу

НАВЧАЛЬНО-МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДЛЯ СТУДЕНТІВ 1 КУРСУ ЗАОЧНОЇ ФОРМИ НАВЧАННЯ

з дисципліни «САТ-технології у перекладі»

Галузь знань	03 Гуманітарні науки
Спеціальність:	035 Філологія
Спеціалізація:	035.041 Германські мови та літератури (переклад включно), перша - англійська
ОПП:	Германські мови та літератури (переклад включно), перша - англійська

Укладач:
ст. викладач Пилипчук М.Л.

Навчально-методичні рекомендації розглянуті та схвалені на засіданні кафедри англійської філології і перекладу
Протокол № 2 від 22.03.2023 р.
Завідувач кафедри___Буданова Л.Г.

Дана навчальна дисципліна є практичною основою сукупності знань та вмінь, що формують профіль фахівця в галузі автоматизованого перекладу.

Метою викладання дисципліни є формування компетентностей та набуття практичних навичок використання основних CAT-програм у галузі автоматизованого перекладу науково-технічних текстів.

Завданнями вивчення навчальної дисципліни є:

- ознайомити студентів із сучасними програмами автоматизованого перекладу (CAT-tools), їх перевагами та недоліками;
- удосконалити навички роботи з машинним перекладом (Machine Translation);
- сформувати вміння створювати та використовувати перекладацьку пам'ять (Translation Memory);
- сформувати вміння студентів виконувати власний перекладацький проєкт у програмах автоматизованого перекладу.

Зміст навчальної дисципліни

Навчальний матеріал дисципліни структурований за модульним принципом і складається з одного навчального **модуля №1 «CAT-технології у перекладі»,** який є логічно завершеною, самостійною, цілісною частиною навчального плану, засвоєння якої передбачає проведення модульної контрольної роботи та аналіз результатів її виконання.

Модульне структурування та інтегровані вимоги до модуля

Модуль 1. CAT-технології у перекладі.

Інтегровані вимоги до модуля № 1: у результаті засвоєння навчального матеріалу навчального модуля № 1 «CAT-технології у перекладі» здобувач повинен **знати** основні програми автоматизованого перекладу (Wordfast Anywhere, Omega T; Memo Q; SmartCat; MateCat); **вміти:** створювати та використовувати базу даних перекладацької пам'яті, редагувати машинний переклад, створювати та керувати перекладацьким проєктом у програмі автоматизованого перекладу.

Тема 1. Програми автоматизованого перекладу. Поняття про автоматизований переклад. Відмінність між автоматизованим та машинним перекладом. Основні програми автоматизованого перекладу. Бази перекладацької пам'яті. Редагування машинного перекладу у CAT-програмі.

Тема 2. Створення та управління перекладацьким проєктом. Створення перекладацького проєкту. Робота з файлами різних форматів (Word, PDF, PowerPoint).

Тема 3. Функціонал програми Wordfast Anywhere. Створення перекладацького проєкту в програмі Wordfast Anywhere. Редагування машинного перекладу у програмі Wordfast Anywhere. Робота з електронними словниками. Функції тегів у програмах автоматизованого перекладу.

Тема 4. Переклад у програмі Omega T. Функціонал програми Omega T. Виконання перекладацького проєкту у програмі Omega T. Порівняльний аналіз програм Wordfast Anywhere та Omega T.

Тема 5. Створення бази перекладацької пам'яті в програмі Omega T. Створення власного словника перекладацької пам'яті та його використання в інших перекладацьких проєктах.

Тема 6. Переклад у програмі Memo Q. Функціонал програми Memo Q. Порівняльний аналіз програм Memo Q та Omega T.

Тема 7. Хмарне середовище програми SmartCat. Функціонал програми SmartCat. Створення перекладацького проєкту та бази перекладацької пам'яті у програмі SmartCat.

Тема 8. Програма автоматизованого перекладу MateCat. Функціонал програми MateCat. Керування перекладацьким проєктом у програмі MateCat. Порівняльний аналіз програм SmartCat та MateCat.

Тема 9. Створення бази перекладацької пам'яті в програмі MateCat. Створення власного словника перекладацької пам'яті та його імплементація в інші перекладацькі проєкти.

Тема 10. Система управління перекладами (TMS). Поняття про систему управління перекладами (Translation Management System). Основні платформи TMS (Lokalise, Transifex, Memsource, Smartling, Phase).

Завдання на контрольну (домашню) роботу.

Контрольна (домашня) робота з дисципліни виконується у другому семестрі, відповідно до затверджених в установленому порядку методичних рекомендацій, з метою закріплення та поглиблення практичних знань та вмінь студента при вивченні дисципліни.

Завдання на контрольну роботу розміщені кафедрою англійської філології і перекладу в електронному репозитарії НАУ. Виконання контрольної (домашньої) роботи здійснюється студентом в індивідуальному порядку відповідно до методичних рекомендацій, розроблених провідним викладачем кафедри.

Час, потрібний для виконання контрольної роботи, складає 8 годин самостійної роботи. Шкала оцінювання КДР

Рейтингова оцінка в балах	Оцінка за національною шкалою
Виконання та захист контрольної (домашньої) роботи	
27-30	Відмінно
23-26	Добре
18-22	Задовільно
Менше 18	Незадовільно

(зразок оформлення роботи)

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет лінгвістики та соціальних комунікацій

Кафедра англійської філології і перекладу

ДОМАШНЯ КОНТРОЛЬНА РОБОТА

з дисципліни:

«CAT-технології у перекладі»

ВИКОНАВ:

студент ___курсу ____ групи

Спеціальність: 035 «Філологія»

ІВАНЕНКО ІВАН ІВАНОВИЧ

ПЕРЕВІРИЛА:

старший викладач кафедри

англійської філології і перекладу

ПИЛИПЧУК М.Л.

Київ – 2023

Завдання КДР

Project 1. SmartCat

1. Створіть проєкт у SmartCat
2. Створіть глосарій у проєкті (Linguistic tools) на 5-10 термінів
3. Перекладіть текст
4. Завантажте переклад у word та вставте у файл рисунки
5. Збережіть у pdf
6. Прикріпіть файл та скріншот роботи у програмі до завдання

Текст для перекладу

Developer console

Code is prone to errors. You will quite likely make errors… Oh, what am I talking about? You are *absolutely* going to make errors, at least if you're a human, not a robot.

But in the browser, users don't see errors by default. So, if something goes wrong in the script, we won't see what's broken and can't fix it.

To see errors and get a lot of other useful information about scripts, "developer tools" have been embedded in browsers.

Most developers lean towards Chrome or Firefox for development because those browsers have the best developer tools. Other browsers also provide developer tools, sometimes with special features, but are usually playing "catch-up" to Chrome or Firefox. So most developers have a "favorite" browser and switch to others if a problem is browser-specific.

Developer tools are potent; they have many features. To start, we'll learn how to open them, look at errors, and run JavaScript commands.

Google Chrome

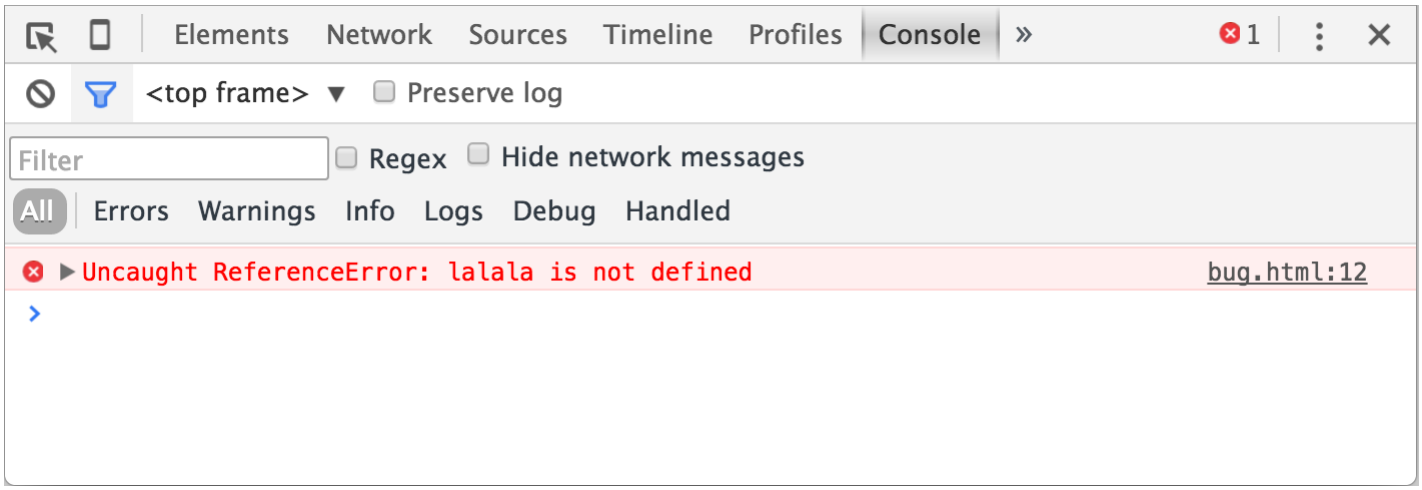
Open the page bug.html.

There's an error in the JavaScript code on it. It's hidden from a regular visitor's eyes, so let's open developer tools to see it.

Press `F12` or, if you're on Mac, then `Cmd+Opt+J`.

The developer tools will open on the Console tab by default.

It looks somewhat like this:



The exact look of developer tools depends on your version of Chrome. It changes from time to time but should be similar.

- Here we can see the red-colored error message. In this case, the script contains an unknown "lalala" command.
- On the right, there is a clickable link to the source bug.html:12 with the line number where the error has occurred.

Below the error message, there is a blue > symbol. It marks a "command line" where we can type JavaScript commands. Press `Enter` to run them.

Now we can see errors, and that's enough for a start. We'll come back to developer tools later and cover debugging more in-depth in the chapter Debugging in the browser.

Multi-line input

Usually, when we put a line of code into the console, and then press `Enter`, it executes.

To insert multiple lines, press `Shift+Enter`. This way one can enter long fragments of JavaScript code.

Firefox, Edge, and others

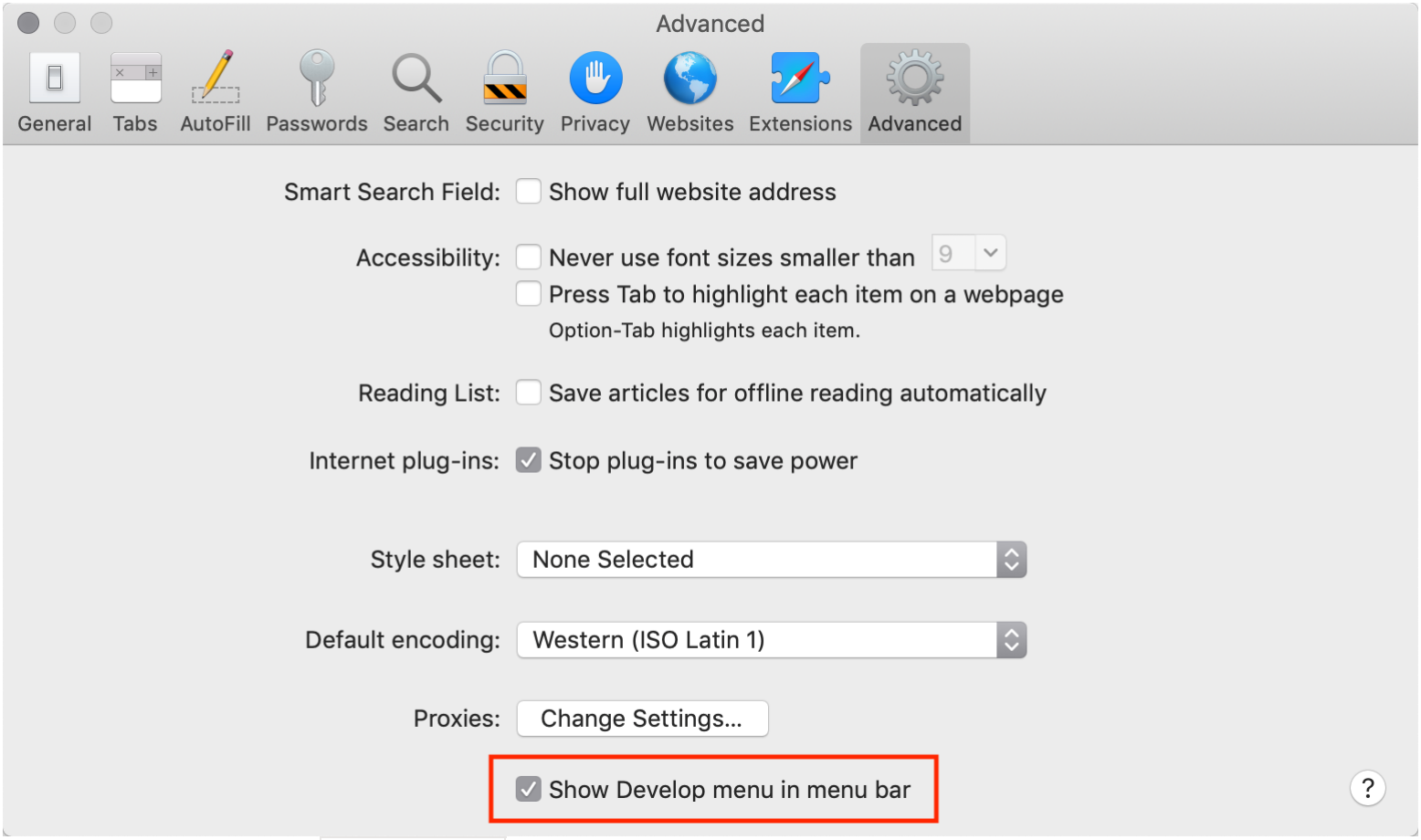
Most other browsers use `F12` to open developer tools.

The look & feel of them is quite similar. Once you know how to use one of these tools (you can start with Chrome), you can easily switch to another.

Safari

Safari (Mac browser, not supported by Windows/Linux) is a little bit special here. We need to enable the "Develop menu" first.

Open Preferences and go to the "Advanced" pane. There's a checkbox at the bottom:



Now `Cmd+Opt+C` can toggle the console. Also, note that the new top menu item named "Develop" has appeared. It has many commands and options.

Summary

- Developer tools allow us to see errors, run commands, examine variables, and much more.
- They can be opened with `F12` for most browsers on Windows. Chrome for Mac needs `Cmd+Opt+J`, Safari: `Cmd+Opt+C` (need to enable first).

Now we have the environment ready. In the next section, we'll get down to JavaScript.

Project 2. OmegaT

1. Встановіть програму OmegaT http://omegat.org/uk/
2. Створіть проєкт
3. Виконайте переклад у програмі
4. Прикріпіть файл з теки проєкту "target" до завдання
5. Прикріпіть скріншот роботи у програмі до завдання

Навчальне відео:
https://www.youtube.com/watch?v=Ttv4dgxiFuo&t=361s&ab_channel=LesiaIvashkevych

Текст для перекладу

Структура коду

Спочатку розглянемо “будівельні блоки” коду.

Інструкції

Інструкції – це синтаксичні конструкції та команди, які виконують дії.

Ми вже бачили інструкцію alert('Привіт, світ!'), яка показує повідомлення “Привіт, світ!”.

Можна писати стільки інструкцій, скільки завгодно. Інструкції можна розділяти крапкою з комою.

Наприклад, тут ми розділити “Привіт, світ” на два виклики alert:

alert('Привіт'); alert('Світ');

Зазвичай кожну інструкцію пишуть з нового рядка, щоби код було легше читати:

```
alert('Привіт');
alert('Світ');
```

Крапка з комою

Здебільшого крапку з комою можна пропустити, якщо є перенесення на новий рядок.

Такий код буде працювати:

alert('Привіт')

```
alert('Світ')
```

У цьому разі JavaScript інтерпретує перенесення рядка як “неявну” крапку з комою. Це називається автоматичне вставлення крапки з комою.

Переважно новий рядок означає крапку з комою. Але “переважно” не означає “завжди”!

У деяких випадках новий рядок не означає крапку з комою. Наприклад:

```
alert(3 +
1
+ 2);
```

Код виведе 6, тому що JavaScript не вставить тут крапку з комою. Інтуїтивно зрозуміло, що, якщо рядок закінчується плюсом "+", то це “незакінчений вираз”, тому крапка з комою не потрібна. І в цьому випадку все працює як задумано.

Але є ситуації, коли JavaScript “забуває” вставити крапку з комою там, де це дійсно потрібно.

Помилки, які виникають у таких ситуаціях, досить важко виявити й виправити.

Приклад такої помилки

Якщо хочете побачити конкретний приклад такої помилки, зверніть увагу на цей код:

```
alert("Привіт");

[1, 2].forEach(alert);
```

Поки що не задумуйтеся, що означають квадратні дужки [] і forEach. Ми вивчимо їх пізніше. Зараз просто запам’ятайте результат виконання коду: спочатку виведеться Привіт, далі 1, а потім 2.

А тепер видалимо крапку з комою після першого alert:

```
alert("Привіт")

[1, 2].forEach(alert);
```

Різниця з кодом вище лише в одному символі: крапка з комою, яку ми видалити в кінці першого рядка.

Якщо ми запустимо цей код, виведеться лише Привіт (а потім виникне помилка, яку можна побачити в консолі). Числа більше не виводяться.

Це тому що JavaScript не вставляє крапку з комою перед квадратними дужками [...]. Оскільки крапка з комою автоматично не вставиться, код у цьому прикладі виконається як одна інструкція.

Ось як рушій побачить її:

```
alert("Привіт")[1, 2].forEach(alert);
```

Виглядає дивно, чи не так? У цьому випадку таке об’єднання неправильне. Щоби код правильно працював, нам потрібно поставити крапку з комою після alert.

Це може статися в інших випадках.

Ми рекомендуємо ставити крапку з комою між інструкціями, навіть якщо вони розділені новими рядками. Це правило широко використовується в спільноті розробників. Варто повторити ще раз – здебільшого *можна* пропускати крапки з комою. Але безпечніше – особливо для новачка – використовувати їх.

Project 3. MemoQ

1. Створіть проєкт у MemoQ.
2. Перекладіть текст.
3. Прикріпіть текст перекладу та скріншот роботи у програмі.

Текст для перекладу

The modern mode, "use strict"

For a long time, JavaScript evolved without compatibility issues. New features were added to the language while old functionality didn't change.

That had the benefit of never breaking existing code. But the downside was that any mistake or an imperfect decision made by JavaScript's creators got stuck in the language forever.

This was the case until 2009 when ECMAScript 5 (ES5) appeared. It added new features to the language and modified some of the existing ones. To keep the old code working, most such modifications are off by default. You need to explicitly enable them with a special directive: "use strict".

"use strict"

The directive looks like a string: "use strict" or 'use strict'. When it is located at the top of a script, the whole script works the "modern" way.

For example:

```
"use strict";

// this code works the modern way
...
```

Quite soon we're going to learn functions (a way to group commands), so let's note in advance that "use strict" can be put at the beginning of a function. Doing that enables strict mode in that function only. But usually people use it for the whole script.

Ensure that "use strict" is at the top

Please make sure that "use strict" is at the top of your scripts, otherwise strict mode may not be enabled.

Strict mode isn't enabled here:

```
alert("some code");
// "use strict" below is ignored--it must be at the top

"use strict";

// strict mode is not activated
```

Only comments may appear above "use strict".

There's no way to cancel use strict

There is no directive like "no use strict" that reverts the engine to old behavior.

Once we enter strict mode, there's no going back.

Browser console

When you use a developer console to run code, please note that it doesn't use strict by default.

Sometimes, when use strict makes a difference, you'll get incorrect results.

So, how to actually use strict in the console?

First, you can try to press Shift+Enter to input multiple lines, and put use strict on top, like this:

```
'use strict'; <Shift+Enter for a newline>
//  ...your code
<Enter to run>
```

It works in most browsers, namely Firefox and Chrome.

If it doesn't, e.g. in an old browser, there's an ugly, but reliable way to ensure use strict. Put it inside this kind of wrapper:

```
(function() {
  'use strict';

  // ...your code here...
})()
```

Should we "use strict"?

The question may sound obvious, but it's not so.

One could recommend to start scripts with "use strict"… But you know what's cool?

Modern JavaScript supports "classes" and "modules" – advanced language structures (we'll surely get to them), that enable use strict automatically. So we don't need to add the "use strict" directive, if we use them.

So, for now "use strict"; is a welcome guest at the top of your scripts. Later, when your code is all in classes and modules, you may omit it.

As of now, we've got to know about use strict in general.

In the next chapters, as we learn language features, we'll see the differences between the strict and old modes. Luckily, there aren't many and they actually make our lives better.

All examples in this tutorial assume strict mode unless (very rarely) specified otherwise.

Project 4. Wordfast Anywhere.

1. зареєструйтеся у програмі WFA за логіном та паролем https://www.wordfast.com/myaccount
2. створіть проєкт
3. перекладіть файл
4. створіть глосарій (мінімум 10 слів)
5. завантажте файл перекладу
6. прикріпіть 4 файли:
-переклад у ворді
-скрін, щоб було видно створений глосарій
-2 скріни перевірки якості перекладу (review)
- transcheck
- spellcheck

Текст для перекладу

Global object

The global object provides variables and functions that are available anywhere. By default, those that are built into the language or the environment.

In a browser it is named window, for Node.js it is global, for other environments it may have another name.

Recently, globalThis was added to the language, as a standardized name for a global object, that should be supported across all environments. It's supported in all major browsers.

We'll use window here, assuming that our environment is a browser. If your script may run in other environments, it's better to use globalThis instead.

All properties of the global object can be accessed directly:

```
alert("Hello");
// is the same as
window.alert("Hello");
```

In a browser, global functions and variables declared with var (not let/const!) become the property of the global object:

```
var gVar = 5;

alert(window.gVar); // 5 (became a property of the global object)
```

Function declarations have the same effect (statements with function keyword in the main code flow, not function expressions).

Please don’t rely on that! This behavior exists for compatibility reasons. Modern scripts use JavaScript modules where such a thing doesn’t happen.

If we used let instead, such thing wouldn’t happen:

```
let gLet = 5;

alert(window.gLet); // undefined (doesn't become a property of the global object)
```

If a value is so important that you’d like to make it available globally, write it directly as a property:

```
// make current user information global, to let all scripts access it
window.currentUser = {
  name: "John"
};

// somewhere else in code
alert(currentUser.name);  // John

// or, if we have a local variable with the name "currentUser"
// get it from window explicitly (safe!)
alert(window.currentUser.name); // John
```

That said, using global variables is generally discouraged. There should be as few global variables as possible. The code design where a function gets “input” variables and produces certain “outcome” is clearer, less prone to errors and easier to test than if it uses outer or global variables.

Using for polyfills

We use the global object to test for support of modern language features.

For instance, test if a built-in Promise object exists (it doesn’t in really old browsers):

```
if (!window.Promise) {
  alert("Your browser is really old!");
}
```

If there’s none (say, we’re in an old browser), we can create “polyfills”: add functions that are not supported by the environment, but exist in the modern standard.

```
if (!window.Promise) {
  window.Promise = ... // custom implementation of the modern language feature
}
```

Summary

- The global object holds variables that should be available everywhere.

That includes JavaScript built-ins, such as Array and environment-specific values, such as window.innerHeight – the window height in the browser.

- The global object has a universal name globalThis.

…But more often is referred by "old-school" environment-specific names, such as window (browser) and global (Node.js).

- We should store values in the global object only if they're truly global for our project. And keep their number at minimum.
- In-browser, unless we're using modules, global functions and variables declared with var become a property of the global object.
- To make our code future-proof and easier to understand, we should access properties of the global object directly, as window.x.