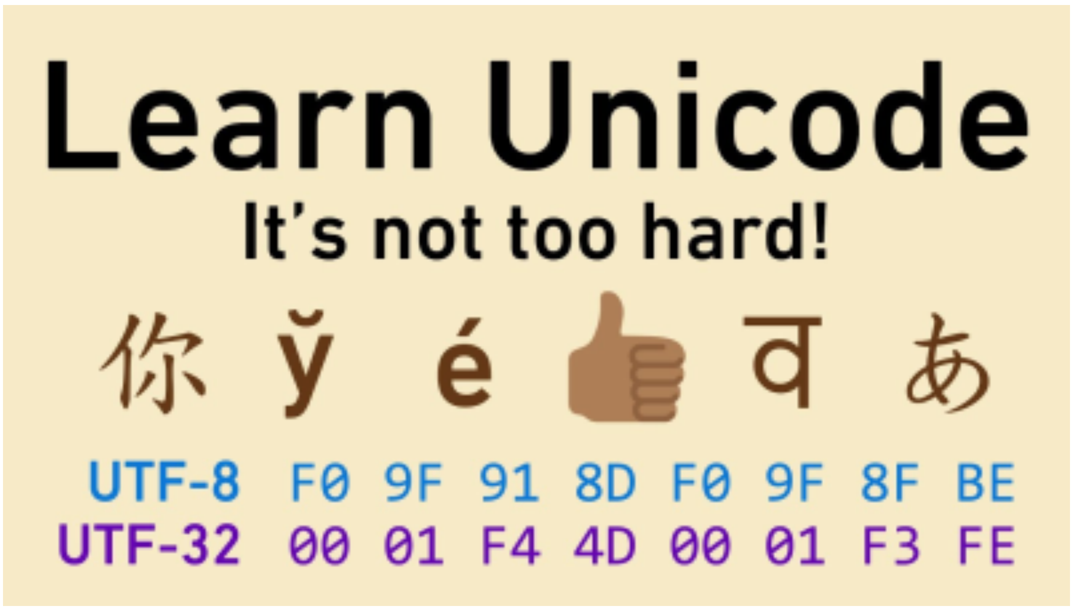
Unicode

We explained unicode in depth in the Assembly programming notes.

WHAT IS UNICODE?

Unicode is a character encoding standard that covers most of the world's writing systems. It uses 16 bits per character, which allows it to represent over a million different characters. This includes letters, numbers, symbols, and punctuation marks from many different languages.



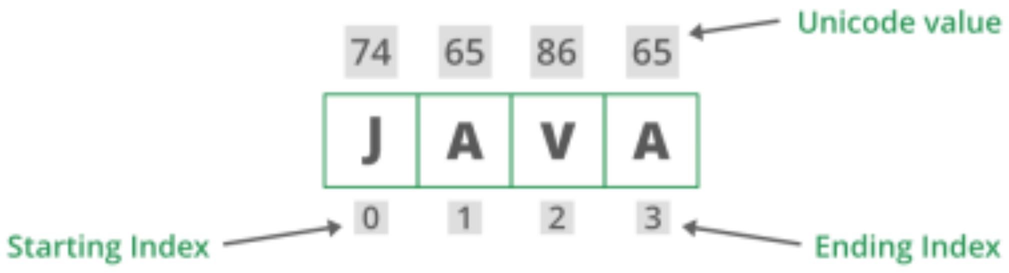
IMPORTANCE OF UNICODE?

Unicode is important because it allows computers to communicate with each other in any language. It also makes it possible to create software that can be used by people from all over the world.

HOW DOES UNICODE WORK?

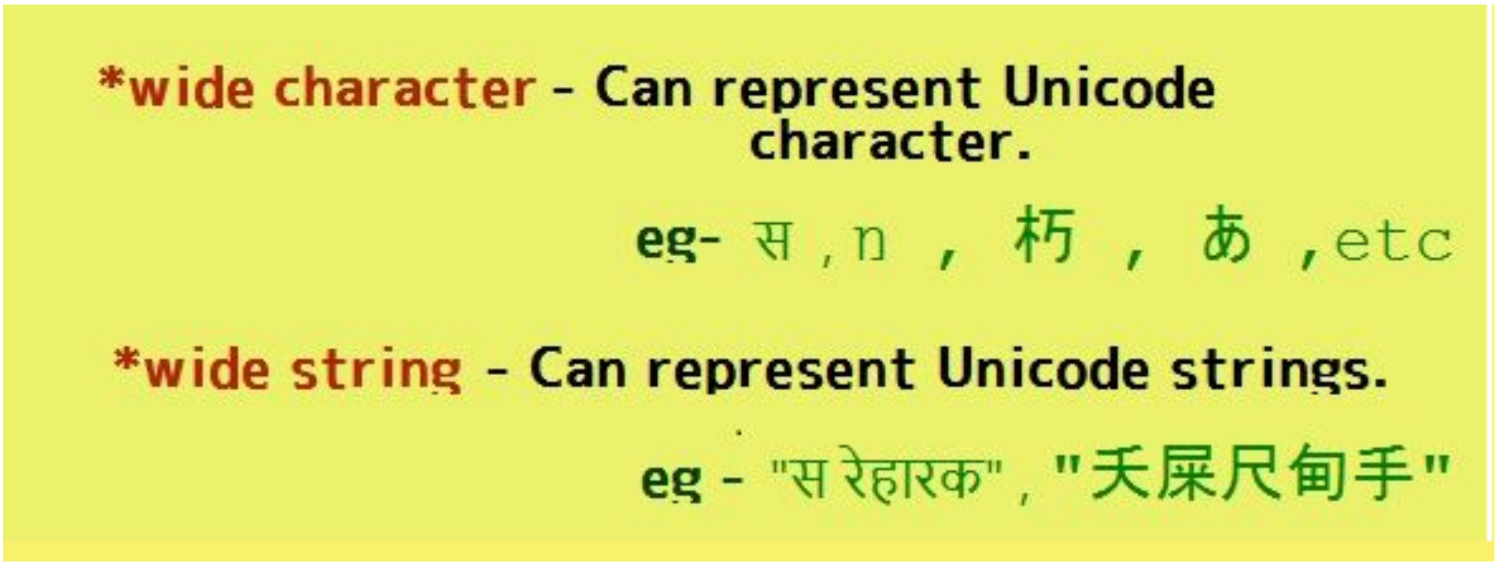
Unicode assigns a **unique code point** to each character. These code points are then used to encode and decode characters in computer data.

Code point is the code for a single character. Any encoding scheme will have a total number of bits used to represent a single character in that particular encoding scheme.



WIDE CHARACTERS IN C

Wide characters are a way of representing Unicode characters in the C programming language. They are 16 bits wide, just like Unicode characters. Wide characters are 16-bit characters that can be used to represent Unicode characters.


*wide character - Can represent Unicode character.

eg- स , n , 朽 , あ ,etc

*wide string - Can represent Unicode strings.

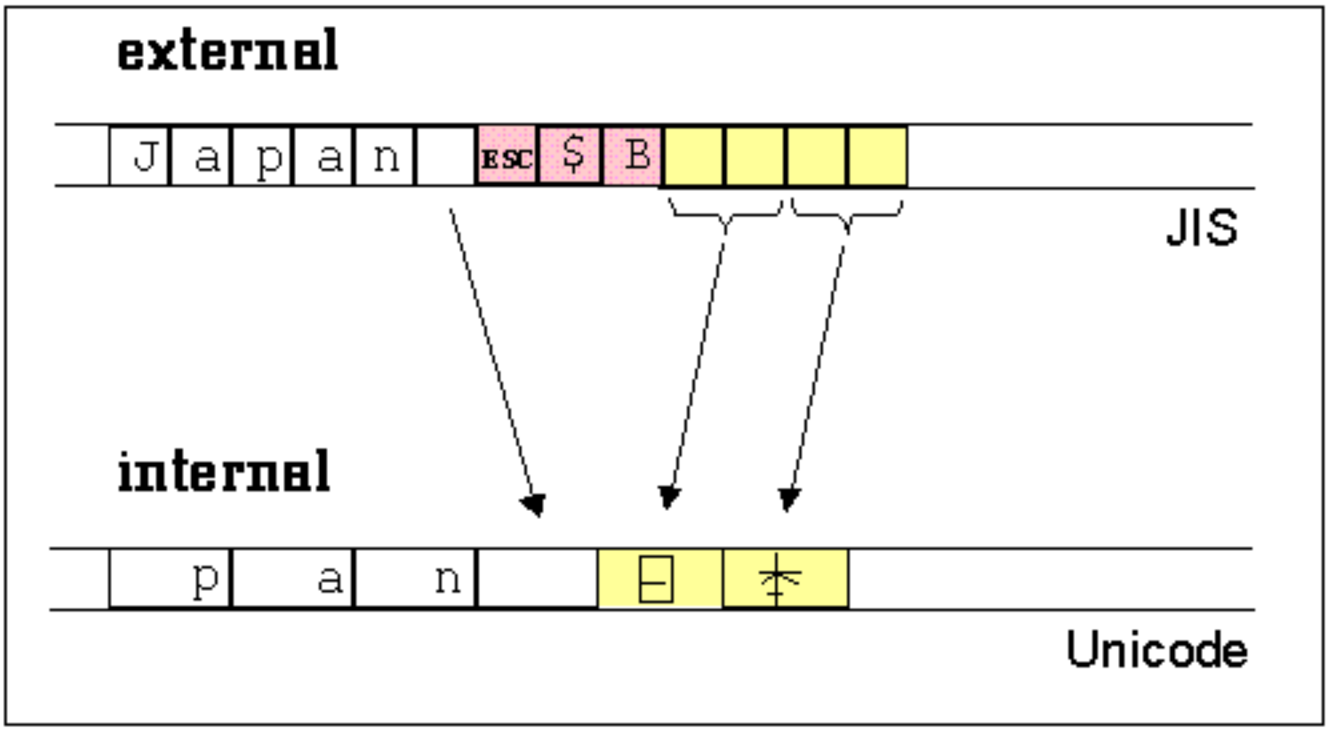
eg - "स रेहारक" , "夭屎尺甸手"

Wide characters are called wide characters because they are not standard ASCII characters and occupy a larger space on screen.

ASCII characters are **7 bits wide**, which means that they can **represent 128 different characters.** This is enough to represent the English alphabet, but not enough to represent all of the characters used in the world's languages.

Unicode characters are **16 bits wide**, which means that they can **represent over a million different characters.** This includes letters, numbers, symbols, and punctuation marks from many different languages.

When a wide character is displayed on the screen, it **occupies two times as much space as an ASCII character.** This is because each wide character takes **two bytes of memory**, while each ASCII character takes only **one byte of memory.**



Here is an example of a wide character string:

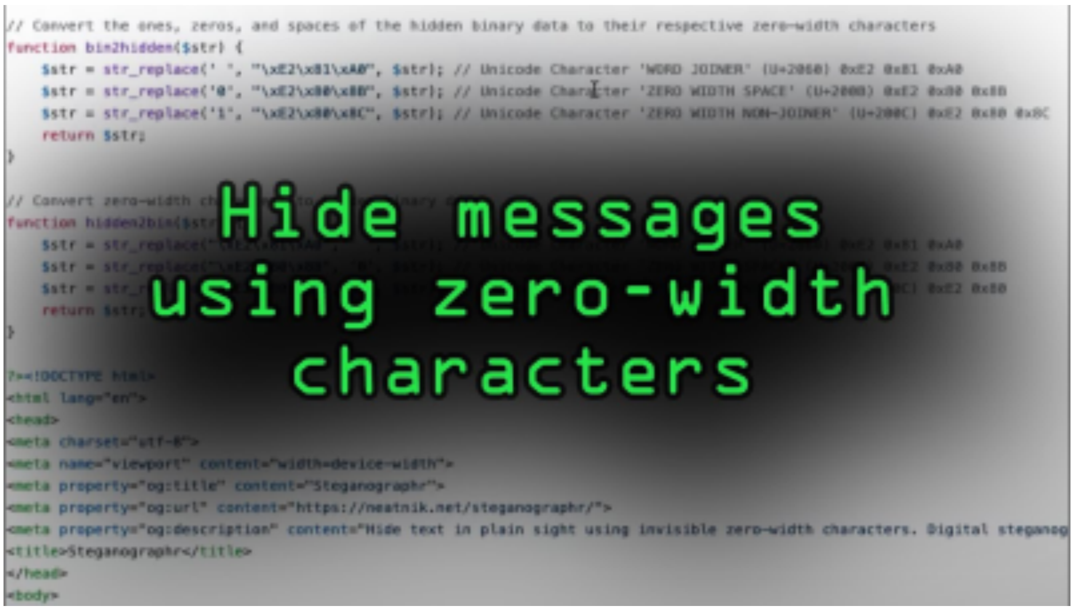
UNICODE IN WINDOWS

Windows NT supports Unicode from the ground up. This means that all of the Windows APIs are Unicode-enabled.

Making your programs Unicode-ready. There are a few things you can do to make your programs Unicode-ready:

- Use wide characters whenever possible.
- Use Unicode-enabled functions and libraries.

- Compile your program with the **/Unicode** compiler switch.



```
// Convert the ones, zeros, and spaces of the hidden binary data to their respective zero-width characters
function bin2hidden($str) {
    $str = str_replace(' ', "\xE2\x81\xA0", $str); // Unicode Character 'WORD JOINER' (U+2060) 0xE2 0x81 0xA0
    $str = str_replace('0', "\xE2\x80\x8B", $str); // Unicode Character 'ZERO WIDTH SPACE' (U+200B) 0xE2 0x80 0x8B
    $str = str_replace('1', "\xE2\x80\x8C", $str); // Unicode Character 'ZERO WIDTH NON-JOINER' (U+200C) 0xE2 0x80 0x8C
    return $str;
}

// Convert zero-width ch[illegible]
function hidden2bin($str[illegible]
    $str = str_replace("\xE2\x81\xA0", [illegible] 0xE2 0x81 0xA0
    $str = str_replace("\xE2\x80\x8B", '0', [illegible] 0xE2 0x80 0x8B
    $str = str_r[illegible] 0xE2 0x80
    return $str;
}

?><!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<meta property="og:title" content="Steganographr">
<meta property="og:url" content="https://neatnik.net/steganographr/">
<meta property="og:description" content="Hide text in plain sight using invisible zero-width characters. Digital steganog
<title>Steganographr</title>
</head>
<body>
```

Character sets: A character set is a collection of characters that are used to represent text. ASCII is a common character set that uses 7 bits per character. Unicode is a superset of ASCII that uses 16 bits per character.

Code points: A code point is a unique identifier for a character in a character set. Each Unicode character has a unique code point.

Encodings: An encoding is a way of representing characters in computer data. The most common Unicode encoding is **UTF-8.**

Normalization: Normalization is the process of converting Unicode characters to a consistent form. This is important for ensuring that characters are displayed correctly and that they can be compared accurately.

This means that all of the **Windows APIs are Unicode-enabled.** This makes it easy to write Unicode programs in Windows.

Unicode is a powerful character encoding standard that allows computers to communicate with each other in any language. It is important to make your programs Unicode-ready so that they can be used by people from all over the world.

```
wprintf(L"Hello, world!");
```

EXTENDING ASCII

A number of different methods have been used to extend ASCII to support other languages and writing systems.

One common approach is to use 8-bit bytes to represent characters.

This allows for up to 256 different characters to be represented, which is enough to support most languages that use the Latin alphabet, as well as some additional characters, such as accented characters and symbols.

Another approach is to use multiple bytes to represent a single character.

This allows for a much larger number of characters to be represented, which is necessary for supporting languages that use other writing systems, such as Chinese and Japanese.

Some examples of extended ASCII character sets include:

ISO 8859-1 (Latin Alphabet No. 1): This character set includes support for most Western European languages.

ISO 8859-2 (Latin Alphabet No. 2): This character set includes support for Central and Eastern European languages.

ISO 8859-5 (Cyrillic): This character set includes support for Russian and other Cyrillic languages.

ISO 8859-6 (Arabic): This character set includes support for Arabic.

ISO 8859-7 (Greek): This character set includes support for Greek.

Windows-1252: This character set is a superset of ISO 8859-1 and includes additional characters for Western Europe, Central and Eastern Europe, and Cyrillic languages.

UTF-8: This character set is a variable-width encoding that can be used to represent any character in the Unicode character set.

ASCII is a very important character encoding standard, but it has a number of limitations. Extended ASCII character sets have been developed to overcome these limitations and support other languages and writing systems. UTF-8 is the most widely used extended ASCII character set today.

CODE PAGES AND THEIR PROBLEMS

Code pages are a powerful tool for supporting different languages and writing systems, but they have a number of problems:

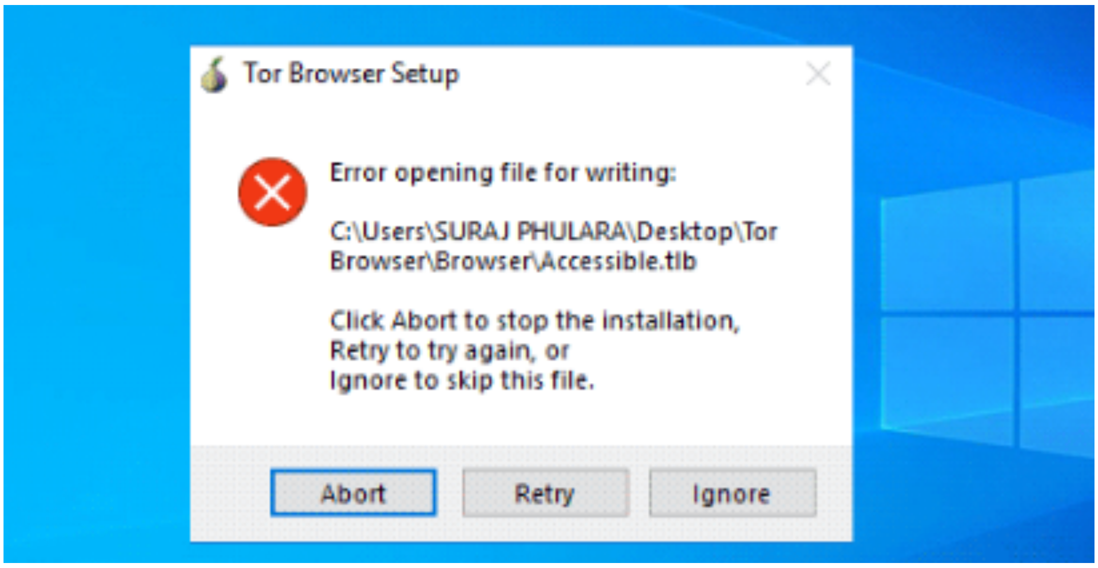
Complexity: Code pages can be complex and difficult to understand. There are many different code pages, and each one has its own unique mapping of character codes to characters. This can make it difficult to troubleshoot problems and ensure that data is displayed and exchanged correctly.

Incompatibility: Code pages are not always compatible with each other. If two systems use different code pages, data may be corrupted when it is exchanged between them. This can be a major problem for businesses and organizations that need to share data with other businesses and organizations that use different systems.

Security vulnerabilities: Code pages can introduce security vulnerabilities into systems. For example, attackers can exploit differences in code pages to inject malicious code into systems.

EXAMPLES OF PROBLEMS WHEN USING CODEPAGES:

If a user attempts to **open a file that was created using a different code page**, the characters in the file may be displayed incorrectly.



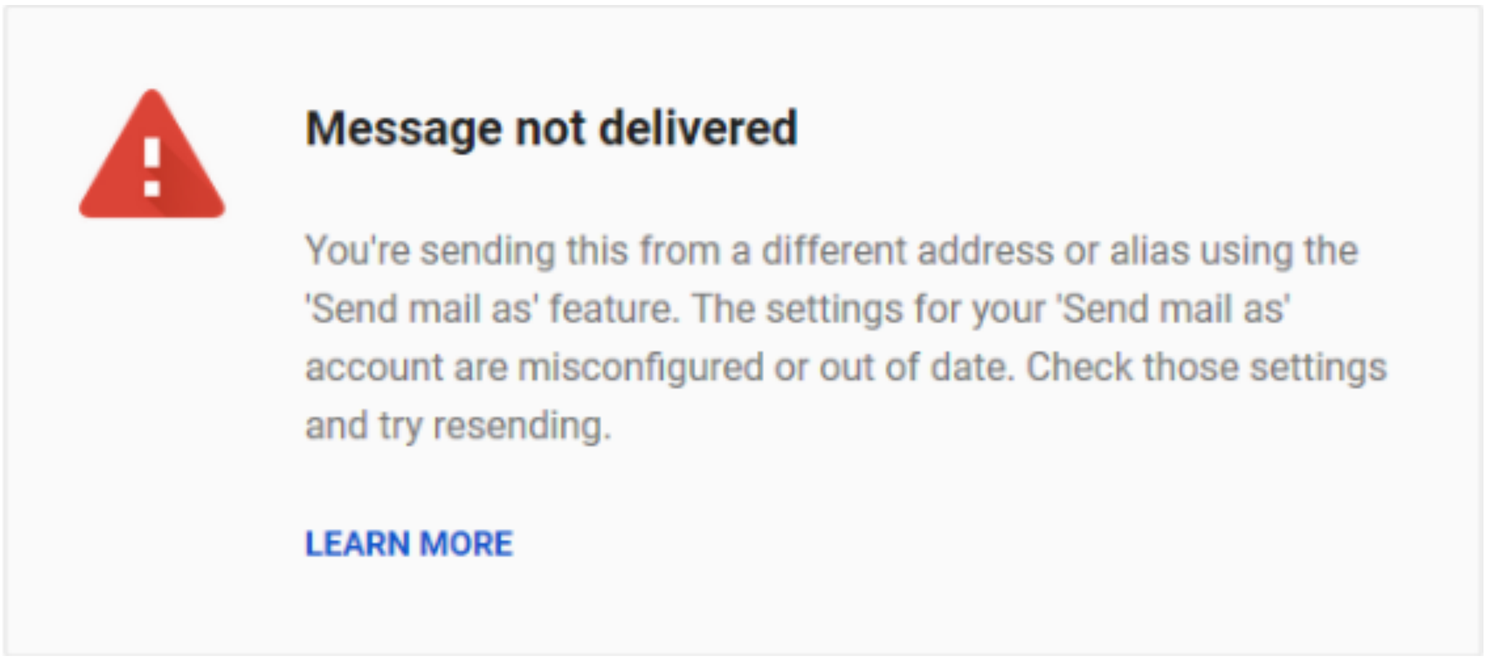
Tor Browser Setup

Error opening file for writing:

C:\Users\SURAJ PHULARA\Desktop\Tor Browser\Browser\Accessible.tlb

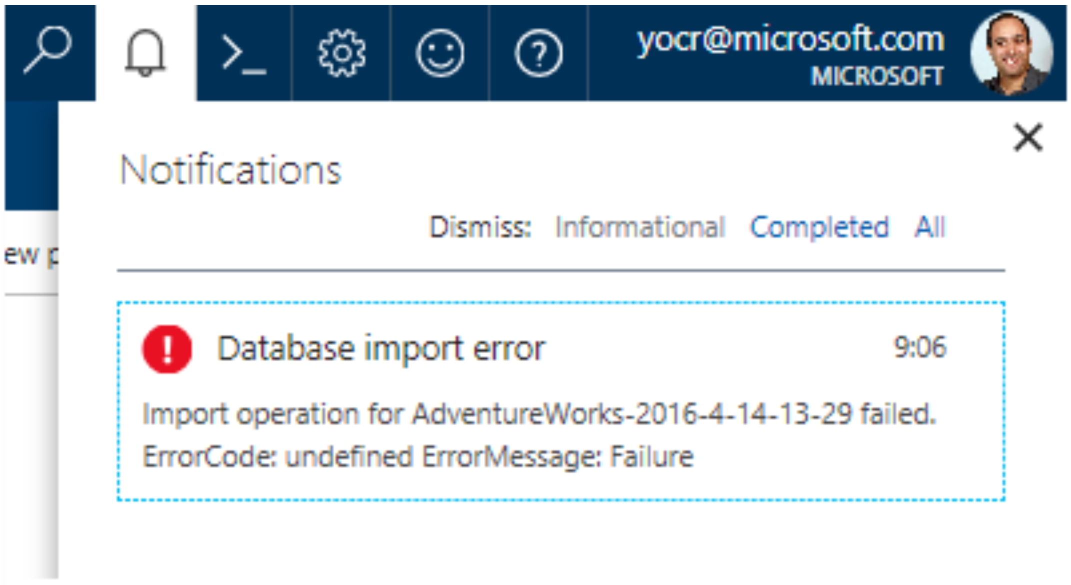
Click Abort to stop the installation,
Retry to try again, or
Ignore to skip this file.

If a user attempts to **send an email** to someone who is using a different code page, the email may be corrupted.


Message not delivered

You're sending this from a different address or alias using the 'Send mail as' feature. The settings for your 'Send mail as' account are misconfigured or out of date. Check those settings and try resending.

If a database is configured to use one code page, and data is imported into the database using a different code page, the data may be corrupted.

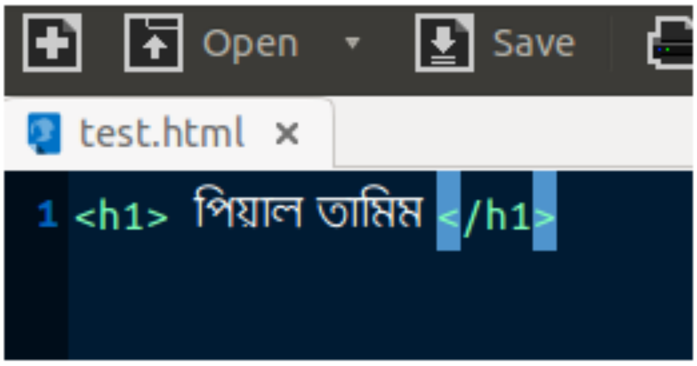


Database import error 9:06

Import operation for AdventureWorks-2016-4-14-13-29 failed. ErrorCode: undefined ErrorMessage: Failure

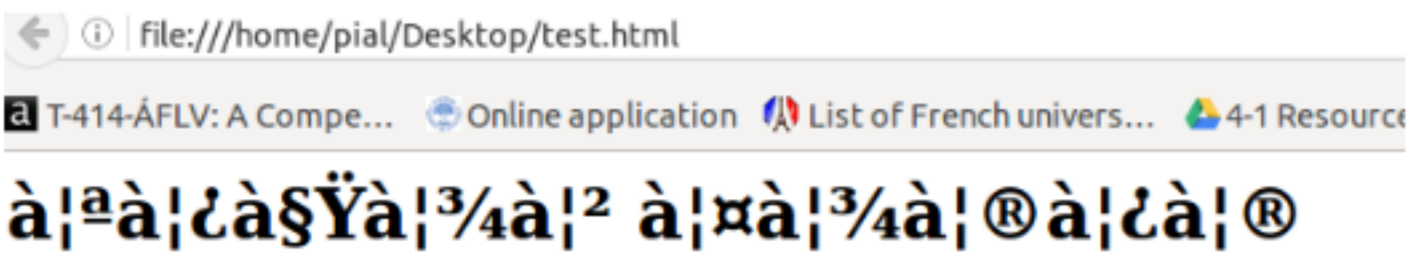
If a web page is encoded using one code page, and a user's browser is configured to use a different code page, the web page may be displayed incorrectly.

For clarification , here is the original text in gedit:



```
<h1> পিয়াল তামিম </h1>
```

And here is how is it viewed in browser:


file:///home/pial/Desktop/test.html
T-414-ÁFLV: A Compe...
Online application
List of French univers...
4-1 Resource

à¦ªà¦¿à§Ÿà¦¾à¦² à¦¤à¦¾à¦®à¦¿à¦®

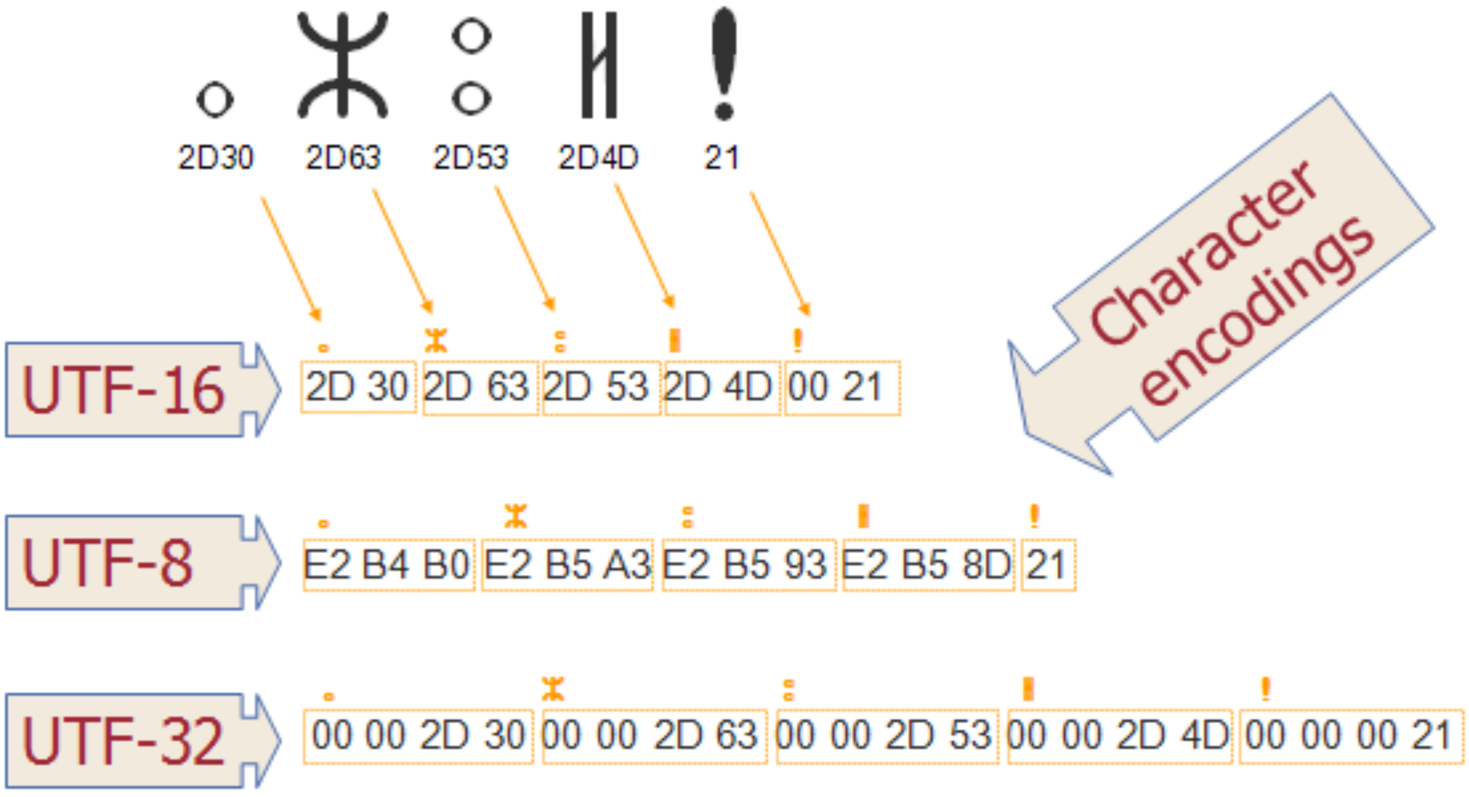
Declare the encoding of your HTML file. Make sure to save it in UTF-8. Here's a minimal example:

```
<meta charset="utf-8">
<h1>उद् उ ज्योतिः अमृतम् विश्वजन्यम्</h1>
```

AVOIDING CODE PAGES

The best way to avoid code page problems is to use a single, universal encoding standard, such as UTF-8.

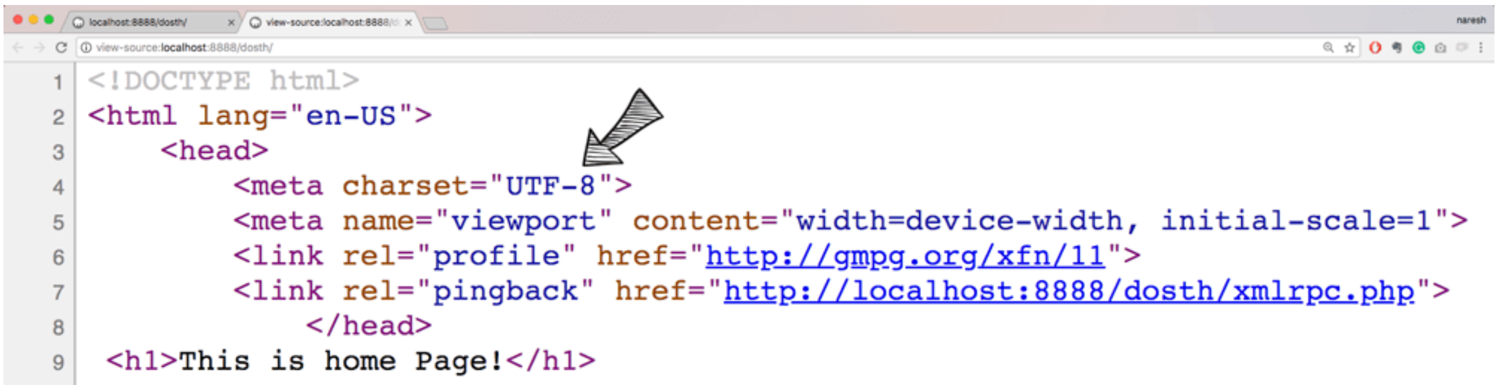
UTF-8 is a variable-width encoding that can represent any character in the Unicode character set. Unicode is a universal character encoding standard that includes characters from all major languages and writing systems.



ⴰ ⵣ ⵓ ⵍ !
2D30 2D63 2D53 2D4D 21
Character encodings
UTF-16
2D 30 2D 63 2D 53 2D 4D 00 21
UTF-8
E2 B4 B0 E2 B5 A3 E2 B5 93 E2 B5 8D 21
UTF-32
00 00 2D 30 00 00 2D 63 00 00 2D 53 00 00 2D 4D 00 00 00 21

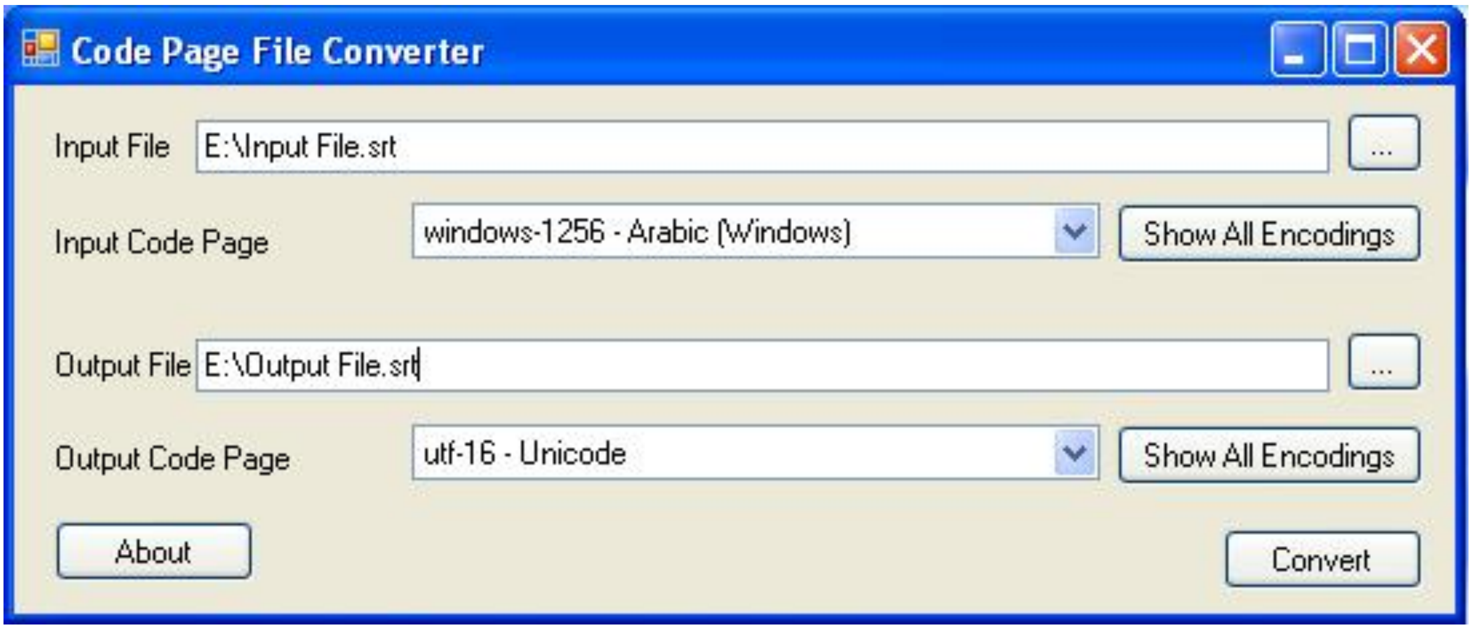
Use a consistent code page throughout your system.

Make sure that all users and systems that need to exchange data are using the **same code page.**



```
<!DOCTYPE html>
<html lang="en-US">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="profile" href="http://gmpg.org/xfn/11">
        <link rel="pingback" href="http://localhost:8888/dosth/xmlrpc.php">
            </head>
 <h1>This is home Page!</h1>
```

Use a tool to **convert between code pages** when necessary.



Code Page File Converter

Input File: E:\Input File.srt

Input Code Page: windows-1256 - Arabic (Windows)

Output File: E:\Output File.srt

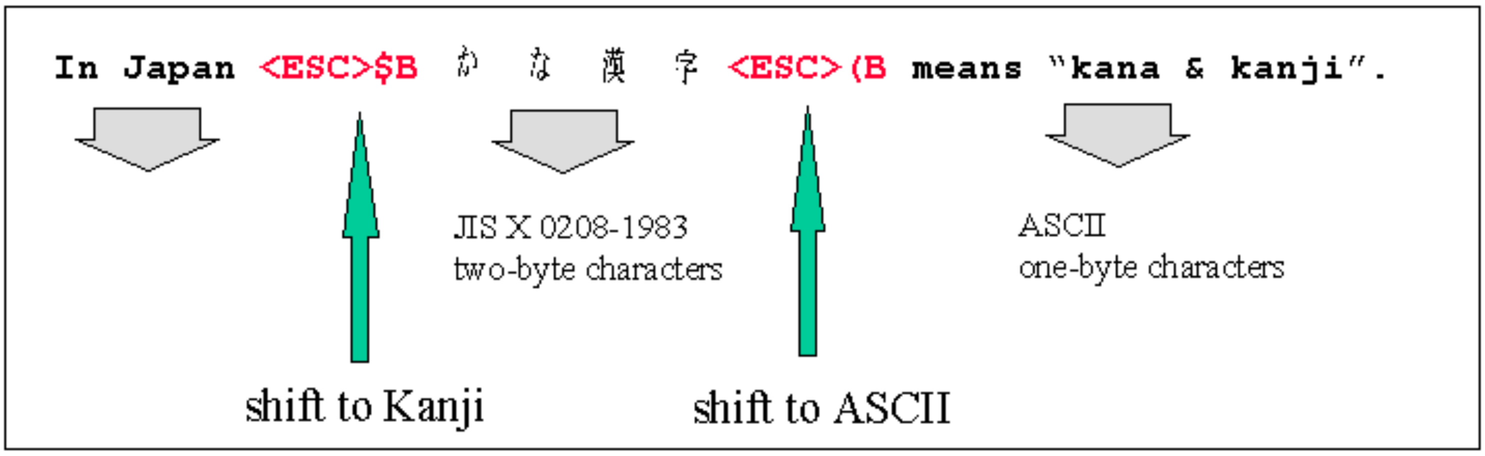
Output Code Page: utf-16 - Unicode

Be aware of the security vulnerabilities that can be introduced by code pages and take steps to mitigate them.

Code pages can be a useful tool for supporting different languages and writing systems, but they have a number of problems. The best way to avoid code page problems is to use a single, universal encoding standard, such as UTF-8.

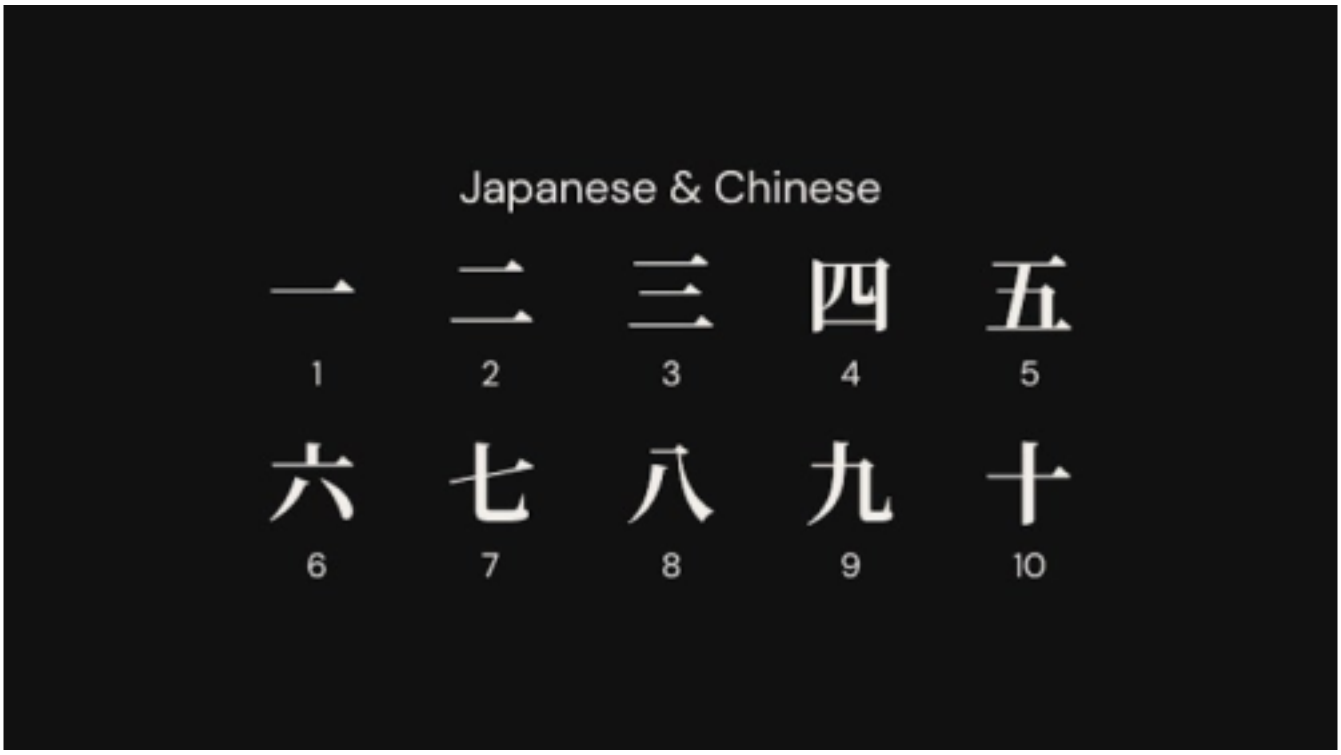
DOUBLE BYTE CHARACTER SET

A double-byte character set (DBCS) is a character encoding scheme that uses two bytes to represent each character. This is in contrast to single-byte character sets (SBCS), which use one byte to represent each character. DBCS is necessary for representing languages such as Chinese, Japanese, and Korean, which have a large number of characters.



To accommodate DBCS while maintaining some kind of compatibility with ASCII, the first 128 codes in a DBCS are the same as ASCII.

However, some of the codes in the higher 128 are always followed by a second byte. The two bytes together (called a lead byte and a trail byte) define a single character, usually a complex ideograph.



Windows supports four different double-byte character sets: code page 932 (Japanese), 936 (Simplified Chinese), 949 (Korean), and 950 (Traditional Chinese). DBCS is supported only in the versions of Windows that are manufactured for these countries.

PROBLEMS WITH DBCS

The main problem with DBCS is that it creates **odd programming problems.** For example, the number of characters in a character string cannot be determined by the byte size of the string.

```
        document.getElementById('bigImageDesc').innerHTML = descriptions[(page * 9) + ...
    }

function updatePhotoDescription() {
    if (descriptions.length > (page * 9) + (currentImage.substring(...
        document.getElementById('bigImageDesc').innerHTML = descriptions[(page ...
    }
}

function updateAllImages() {
    var i = 1;
    while (i < 10) {
        var elementId = 'foto' + i;
        var elementIdBig = 'bigImage' + i;
```

The string has to be parsed to determine its length, and each byte has to be examined to see if it's the lead byte of a 2-byte character.

If you have a pointer to a character somewhere in the middle of a DBCS string, what is the address of the previous character in the string? The customary solution is to parse the string starting at the beginning up to the pointer!

Another problem with DBCS is that it can lead to security vulnerabilities. For example, attackers can exploit differences in DBCS implementations to inject malicious code into systems.

AVOIDING DBCS PROBLEMS

The best way to avoid DBCS problems is to use a single, universal encoding standard, such as UTF-8. UTF-8 is a variable-width encoding that can represent any character in the Unicode character set. \

Unicode is a universal character encoding standard that includes characters from all major languages and writing systems.

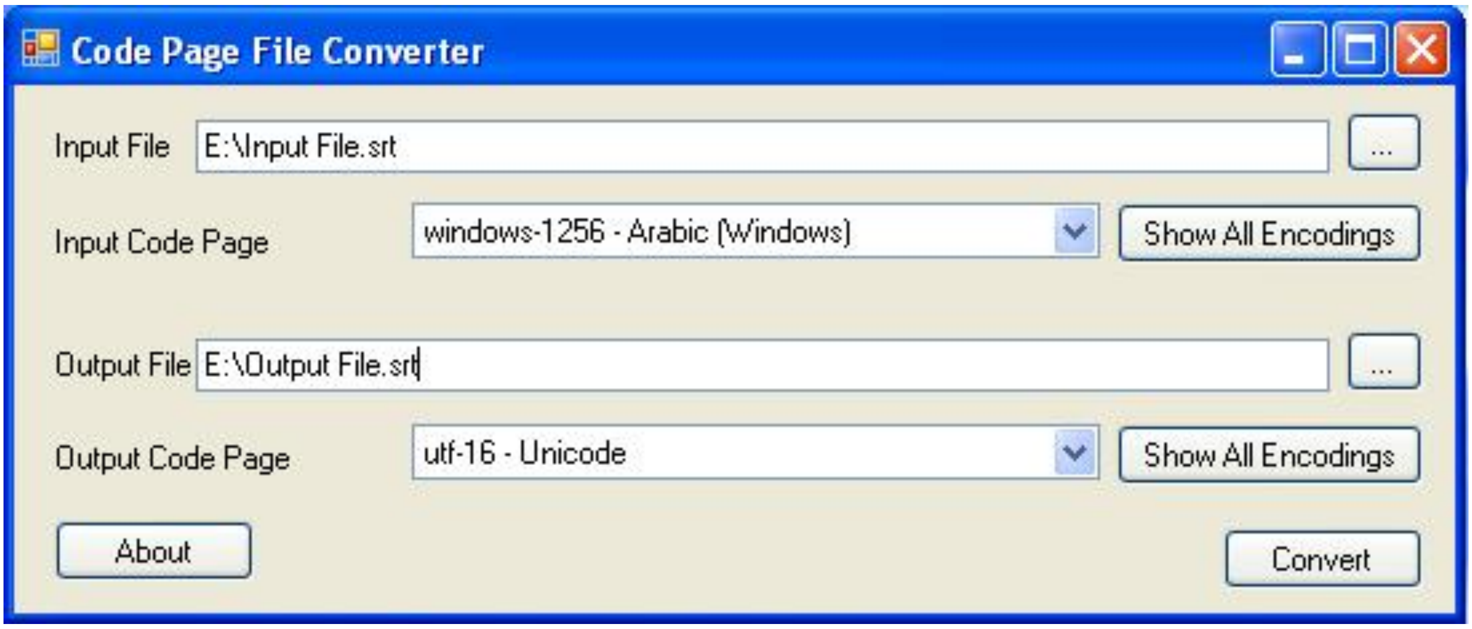
If you must use DBCS, there are a few things you can do to reduce the risk of problems:

Use a consistent code page throughout your system.

Consistent Coding

Make sure that all users and systems that need to exchange data are using the same code page.

Use a tool to convert between code pages when necessary.



Code Page File Converter

Input File: E:\Input File.srt

Input Code Page: windows-1256 - Arabic (Windows)

Output File: E:\Output File.srt

Output Code Page: utf-16 - Unicode

Be aware of the **security vulnerabilities** that can be introduced by DBCS and take steps to mitigate them.

DBCS is a complex and **error-prone encoding scheme.** It is best avoided if possible. If you must use DBCS, be sure to take steps to mitigate the risks associated with it.

UNICODE TO THE RESCUE OF REPRESENTING DATA

Unicode is a universal character encoding standard that includes characters from all major languages and writing systems.

It is a 16-bit system, which means that it can represent up to 65,536 characters. This is in contrast to ASCII, which is an 8-bit system and can only represent 256 characters.

Unicode is the real solution to the problem of representing the world's written languages in computers. It is a single, unambiguous character set that can be used by all programmers and software developers.

ADVANTAGES OF UNICODE

Unicode has a number of advantages over other character encoding schemes, such as ASCII and DBCS:

It is universal and can be used to represent all major languages and writing systems.

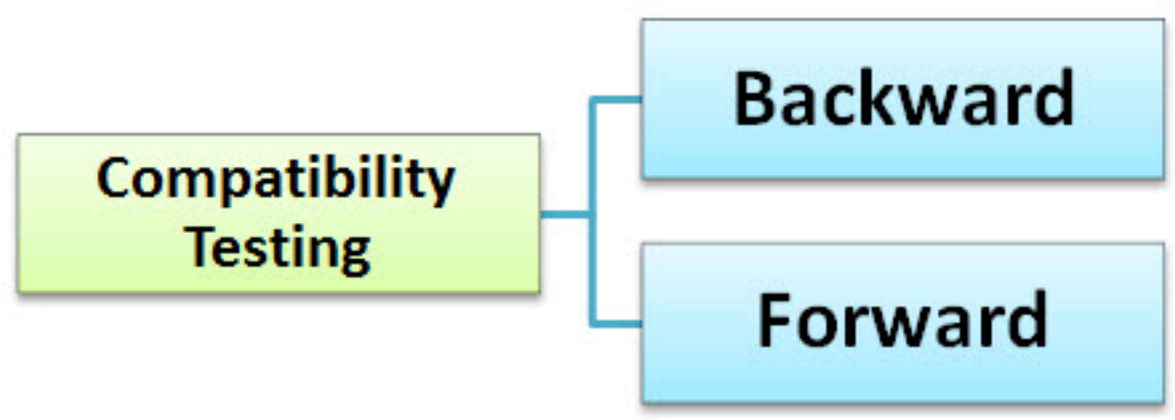
Unicode is supported by all major operating systems, programming languages, and software applications.

Unicode is used in the world wide web, and all major web browsers support it.

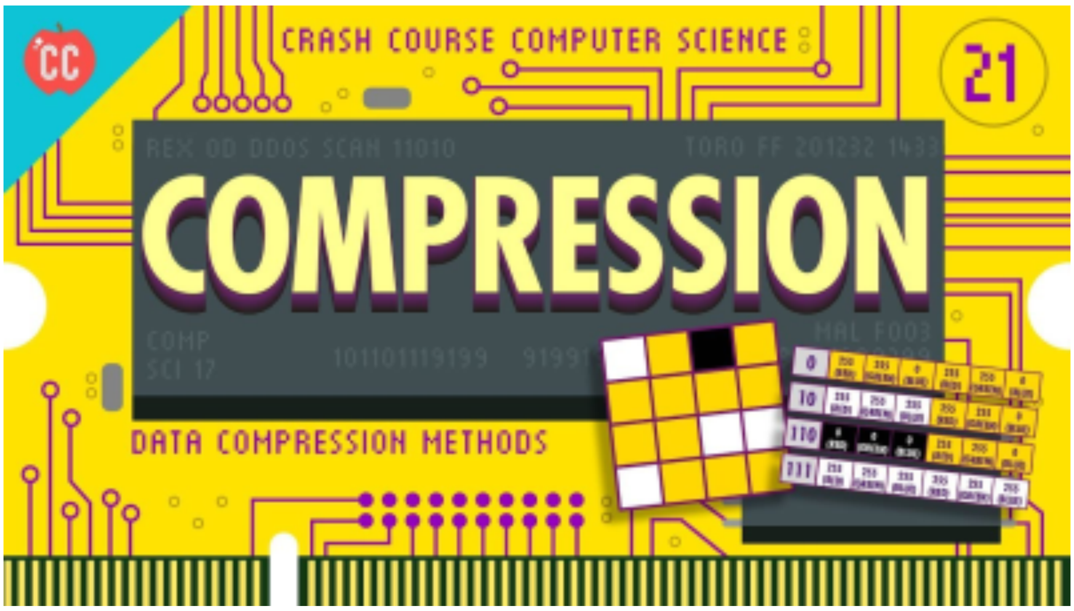
Unicode is used in many other areas, such as email, document processing, and electronic publishing.

It is unambiguous, which means that there is only one way to represent each character.

It is **forward-compatible,** which means that new characters can be added to the standard without breaking existing software.



It is efficient, as Unicode strings can be compressed to reduce their size.



DISADVANTAGES OF UNICODE

The main disadvantage of Unicode is that it **requires more memory** than other character encoding schemes. However, this is becoming less of an issue as computers become more powerful.

To use Unicode, you need to make sure that your programming language and software applications support it. Once you have confirmed that your programming language and software applications support Unicode, you can start using it to represent text in your code.

- Use Unicode strings whenever possible.
- Encode all text in Unicode before storing it in a database or file.
- Decode all text from Unicode before displaying it to the user.
- Use a Unicode font when displaying text.

Unicode is the future of character encoding. It is the best way to represent the world's written languages in computers. As programmers, we should all strive to use Unicode in our code.