INTRODUCTION TO UNICODE AND IMPORTANCE OF LOCALIZATION IN FINTECH APPLICATIONS

Harsha Wijayawardhana B.Sc. (Miami), CITP (UK),FBCS(UK)
COO/CTO Theekshana R & D
Member, LLWG

WHAT IS UNICODE

- Unicode Standard provides a unique number for every character in all languages in the world today (no matter what platform, device, application or language).
- Unicode also provides a platform to encode scripts which are not in use such as Javanese, etc.
- Previously, many character encoding existed and Unicode brought a single standard throughout.
- Support of Unicode forms the foundation for the representation of languages and symbols in all major operating systems, search engines, browsers, laptops, and smart phones—plus the Internet and World Wide Web (URLs, HTML, XML, CSS, JSON, etc.).

UNICODE CONSORTIUM

- Many large Multi National Businesses are members of Unicode Consortium such as Google, Microsoft, Apple, etc.
- ICTA had an associate membership and LLWG will explore the possibility of retaining the membership

WHAT UNICODE CONSORTIUM DOES?

- "The assignment of characters is only a small fraction of what the Unicode Standard and its associated specifications provide. They give programmers extensive descriptions and a vast amount of data about how characters function: how to form words and break lines; how to sort text in different languages; how to format numbers, dates, times, and other elements appropriate to different languages; how to display languages whose written form flows from right to left, such as Arabic and Hebrew, or whose written form splits, combines, and reorders, such as languages of South Asia; and how to deal with security concerns regarding the many "look-alike" characters from alphabets around the world. Without the properties, algorithms, and other specifications in the Unicode Standard and its associated specifications, interoperability between different implementations would be impossible."

(Reference Unicode Consortium)

UNIVERSAL CODED CHARACTER SET

- The **Universal Coded Character Set** (**UCS, Unicode**) is a standard set of characters defined by the International Standard **ISO/IEC 10646**, *Information technology — Universal Coded Character Set (UCS)* (plus amendments to that standard), which is the basis of many character encodings, improving as characters from previously unrepresented writing systems are added.
- The UCS has over 1.1 million possible code points available for use/allocation, but only the first 65,536, which is the Basic Multilingual Plane (BMP), had entered into common use before 2000.

OTHER STANDARDS

- National or Regional Standards
  - ISO-Latin-1/9: targets Western Europe
  - JIS: targets Japan
- Platform Standards
  - Microsoft code pages
  - Apple: MacRoman, etc.
  - Adobe: PDFDocEncoding, etc.

UNICODE

- ❑ enables world-wide interchange of data
- ❑ contains all the major living scripts
- ❑ simple enough to be implemented everywhere
- ❑ supports legacy data and implementation
- ❑ allows a single implementation of a product
- ❑ supports multilingual users and organizations
- ❑ conforms to international standards
- ❑ can serve as the foundation for other standards

FOUR LAYERS

- Abstract Character Set
  - smallest components of written language
- Coded character set
  - adds name and code point
- Character Encoding Forms
  - representation in computer
- Character Encoding Schemes
  - byte serialization

ABSTRACT CHARACTER SET

- character:

the smallest component of written language that has semantic value

- Wide Variation Across Scripts

alphabetic, syllabary, abugidas, abjad, logographic

- abstract character:

a unit of information used for the organization, control, or

representation of textual data.

CODED CHARACTER SET

- give a name and a code point to each abstract character
- name: LATIN CAPITAL LETTER A

  code point: pure number, no computer connection

  legal values: U+0000 - U+10FFFF

  space for over a million characters
- characters specific to a script mostly grouped

17 PLANES

- 17 planes of 64k code points each

plane 0: Basic Multilingual Plane (BMP, 1.0) frequent characters

- plane 1: Supplementary Multilingual Plane (SMP, 3.1) infrequent, non-ideographic characters
- plane 2: Supplementary Ideographic Plane (SIP, 3.1) infrequent, ideographic characters
- plane 14: Supplementary Special-purpose Plane (SSP, 3.1)
- planes 15 and 16: Private use planes (2.0)

PRIVATE USE AREA

- for your own characters; will never be assigned; must agree on the meaning of those code points.
- Unicode does not provide a mechanism to do so
- very delicate to use; avoid it if possible
- distribution:

  U+E000 - U+F8FF: 6,400 in the BMP

  U+F0000 - U+FFFFF: 64k in plane 15

  U+100000 - U+10FFFF: 64k in plane 16

SURROGATE CODE POINTS AND SCALAR VALUES

- Unicode was originally defined as a “16 bit character set”
- in 1996 (Unicode 2.0), realized that this was not enough
- code points set aside: surrogates code points

    U+D800 - U+DBFF: 1,024 high surrogates

    U+DC00 - U+DFFF: 1,024 low surrogates
- remaining code points: scalar values

    U+0000 - U+D7FF

    U+E000 - U+10FFFF
- surrogates code points must never appear in data

SCALAR MEAN IN UNICODE

A Unicode Scalar is a code point which is not serialised as a pair of UTF-16 code units. A code point is the number resulting from encoding a character in the Unicode standard. For instance, the code point of the letter A is 0x41 (or 65 in decimal).

WHAT IS SURROGATE PAIR MEAN?

A surrogate pair is two 16-bit code units used in UTF-16 (16-bit - two-byte) that represents a character above the maximum value stored in 16bit. (ie 0xFFFF HEXA or 65535 decimal)

Why ? Because the Unicode set has way more character than 65535 (16bit), therefore to represent a Code Point (Characters) above 0xFFFF (such 0x10000 to 0x10FFFF,), a pairs of code units known as surrogates is used.

CHARACTER ENCODING FORMS: UTFS

- the representation of scalar values in computers
- each scalar value represented by a sequence of code units
- three forms, defined by:

size of the underlying code unit (8, 16, 32 bits)

method to convert a scalar value to code units

UTF 8

UTF-8 is a variable-width character encoding used for electronic communication. Defined by the Unicode Standard, the name is derived from *Unicode* (or *Universal Coded Character Set*) *Transformation Format – 8-bit*.

UTF-8 is capable of encoding all 1,112,064 valid character code points in Unicode using one to four one-byte (8-bit) code units. Code points with lower numerical values, which tend to occur more frequently, are encoded using fewer bytes. It was designed for backward compatibility with ASCII: the first 128 characters of Unicode,

UTF 16

- **UTF-16** (16-bit Unicode Transformation Format) is a character encoding capable of encoding all 1,112,064 valid character code points of Unicode (in fact this number of code points is dictated by the design of UTF-16). The encoding is variable-length, as code points are encoded with one or two 16-bit *code units*. UTF-16 arose from an earlier obsolete fixed-width 16-bit encoding, now known as UCS-2 (for 2-byte Universal Character Set), once it became clear that more than 2^{16} (65,536) code points were needed.
- Two groups worked on this in parallel, ISO/IEC JTC 1/SC 2 and the Unicode Consortium, the latter representing mostly manufacturers of computing equipment. The two groups attempted to synchronize their character assignments so that the developing encodings would be mutually compatible. The early 2-byte encoding was originally called "Unicode", but is now called "UCS-2".

UTF 32

- **UTF-32** (32-bit Unicode Transformation Format) is a fixed-length encoding used to encode Unicode code points that uses exactly 32 bits (four bytes) per code point (but a number of leading bits must be zero as there are far fewer than 2^{32} Unicode code points, needing actually only 21 bits). UTF-32 is a fixed-length encoding, in contrast to all other Unicode transformation formats, which are variable-length encodings. Each 32-bit value in UTF-32 represents one Unicode code point and is exactly equal to that code point's numerical value.
- The original ISO 10646 standard defines a 32-bit *encoding form* called **UCS-4**, in which each code point in the Universal Character Set (UCS) is represented by a 31-bit value from 0 to 0x7FFFFFFF (the sign bit was unused and zero). In November 2003, Unicode was restricted by RFC 3629 to match the constraints of the UTF-16 encoding: explicitly prohibiting code points greater than U+10FFFF (and also the high and low surrogates U+D800 through U+DFFF). This limited subset defines UTF-32.Although the ISO standard had (as of 1998 in Unicode 2.1) "reserved for private use" 0xE00000 to 0xFFFFFF, and 0x60000000 to 0x7FFFFFFF these areas were removed in later versions. Because the Principles and Procedures document of ISO/IEC JTC 1/SC 2Working Group 2 states that all future assignments of code points will be constrained to the Unicode range, UTF-32 will be able to represent all UCS code points and UTF-32 and UCS-4 are identical.

BIG ENDIAN AND LITTLE ENDIAN

The adjectives *big-endian* and *little-endian* refer to which bytes are most significant in multi-byte data types and describe the order in which a sequence of bytes is stored in a computer's memory.

In a big-endian system, the most significant value in the sequence is stored at the lowest storage address (i.e., first). In a little-endian system, the least significant value in the sequence is stored first. For example, consider the number 1025 (2 to the tenth power plus one) stored in a 4-byte integer:

CHARACTER LATIN A

abstract character:

the letter of the Latin script

coded character:

name: LATIN CAPITAL LETTER A

code point: U+0041

encoding forms:

UTF-8: 41

UTF-16: 0041

UTF-32: 00000041

CHARACTER HIRAGANA MA

abstract character:

the letter of the Hiragana script

coded character:

name: HIRAGANA LETTER MA

code point: U+307E

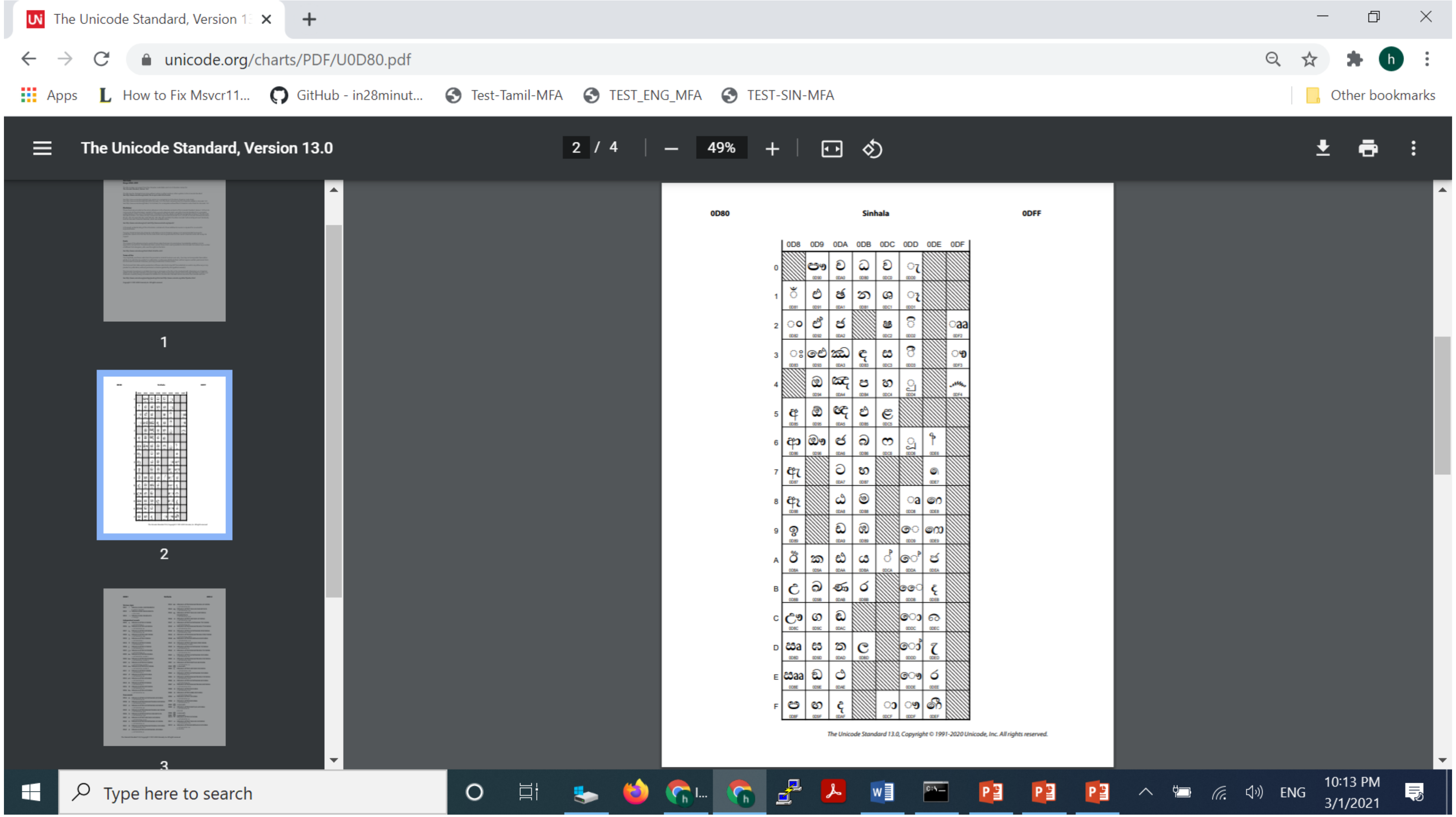
Encoding forms:

UTF-8: E3 81 BE

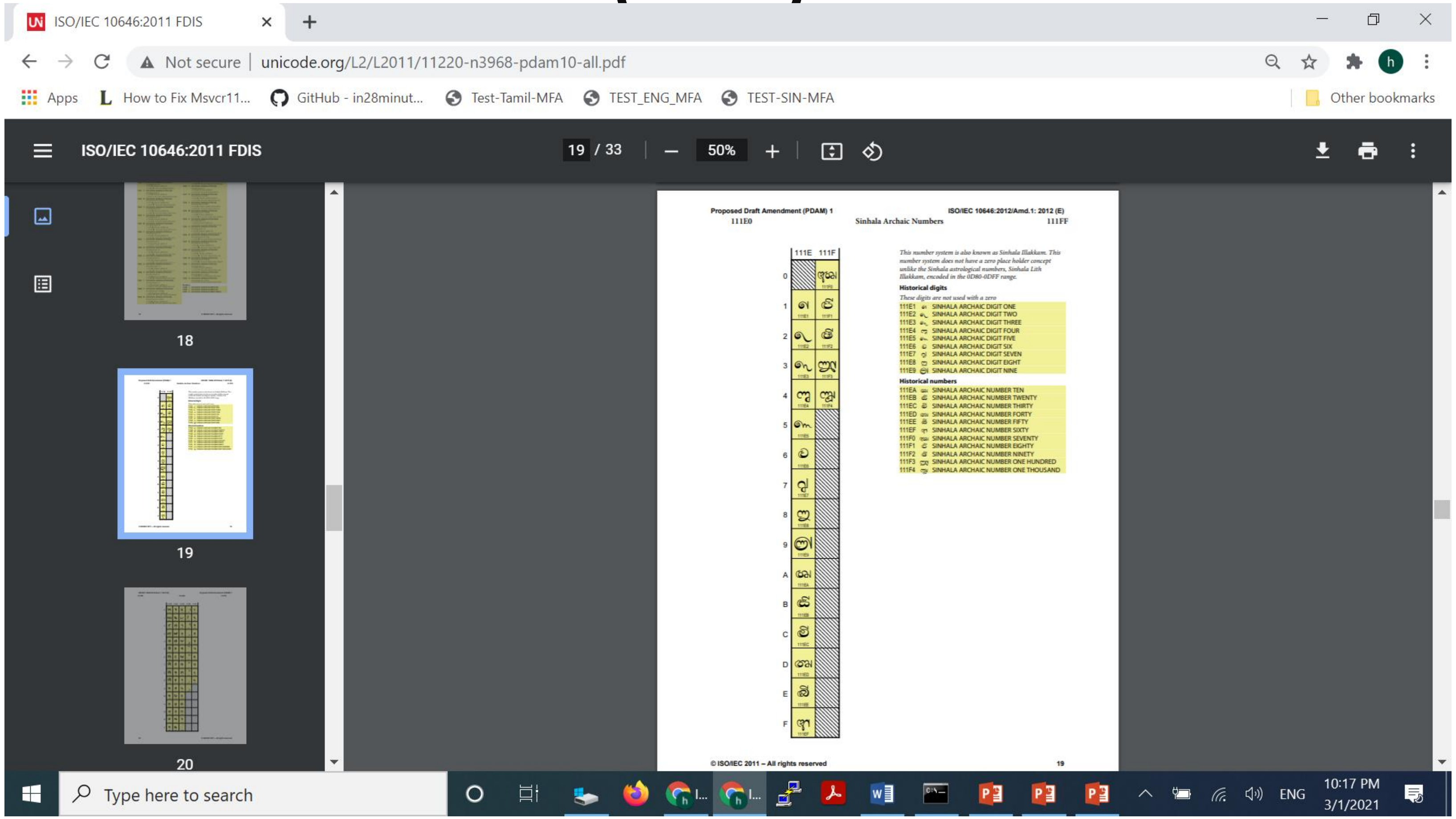
UTF-16: 307E

UTF-32: 0000307E

UNICODE VERSION 13.1



SINHALA ARCHAIC NUMBERS IN SUPPLEMENTARY MULTILINGUAL PLANE (SMP)



PRINCIPLES OF THE ABSTRACT CHARACTER SET

characters, not glyphs

plain text only

unification, within each script, across languages

well-defined semantics for characters

dynamic composition of marked forms

equivalence for pre-composed forms

characters are stored in logical order

round-tripping with some other standards

SINHALA AND TAMIL SUPPORT IN PHP APPLICATIONS, MYSQL AND MSSQL SERVER

mb_convert_encoding — Convert character encoding

n MySQL 5.5.3, this was addressed with the addition of support for the utf8mb4 character set which uses a maximum of four bytes per character and thereby supports the full UTF-8 character set. So if you're using MySQL 5.5.3 or later, use utf8mb4 instead of UTF-8 as your database/table/row character set.

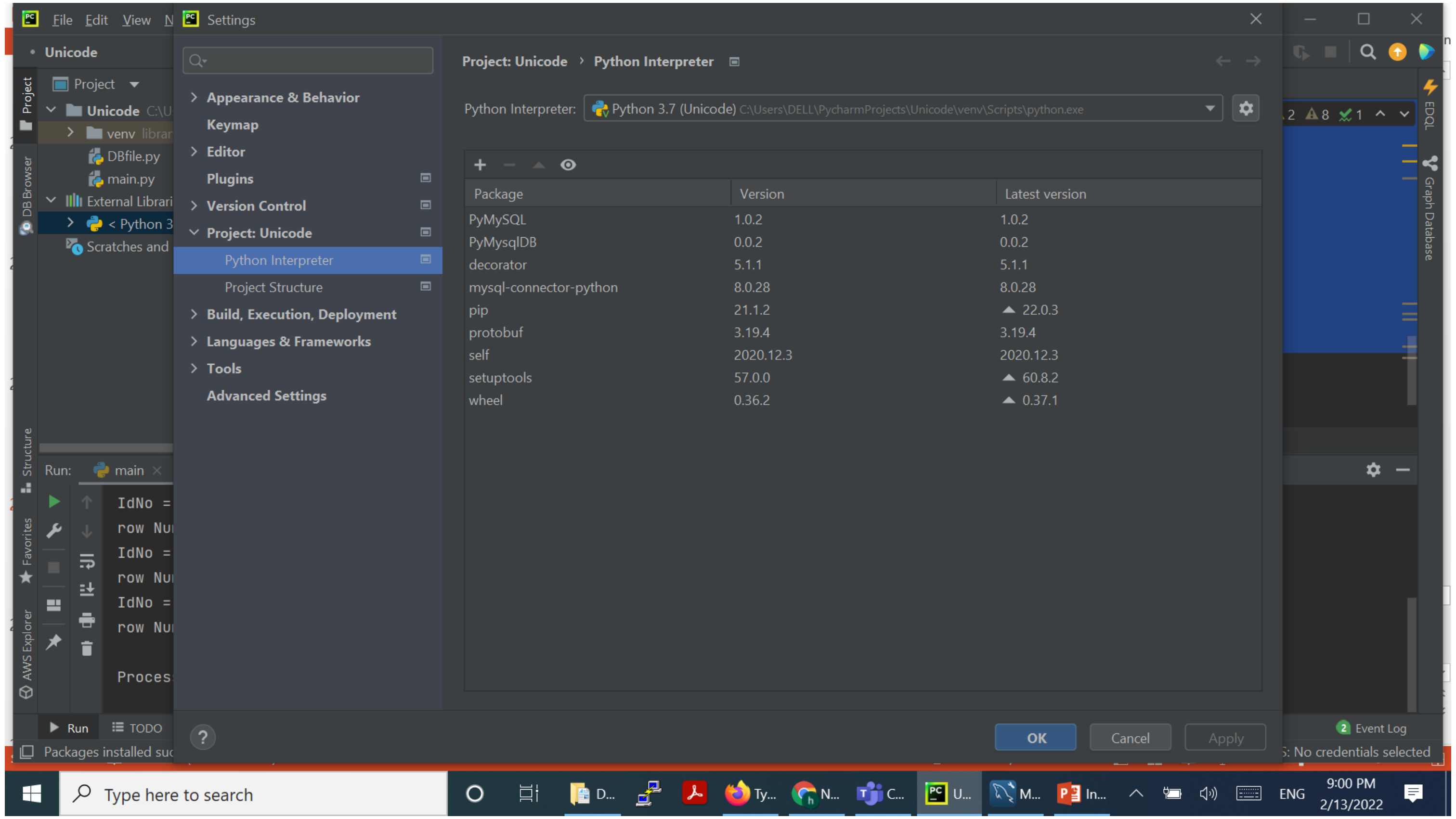
MSSQL Server handles UTF 8, 16 and 32 without an issue.

SQL Server 2012 (11.x) introduced a new family of supplementary character (_SC) collations that can be used with the data types nchar, nvarchar, and sql_variant to represent the full Unicode character range (000000 to 10FFFF).

SOME OF PACKAGES REQUIRE INSTALLING

- PyCharm Community Edition 2021.2
- Navigate File->Settings->Project and click on the name of the project and check on the packages
- PyMysql

PACKAGES CONT.

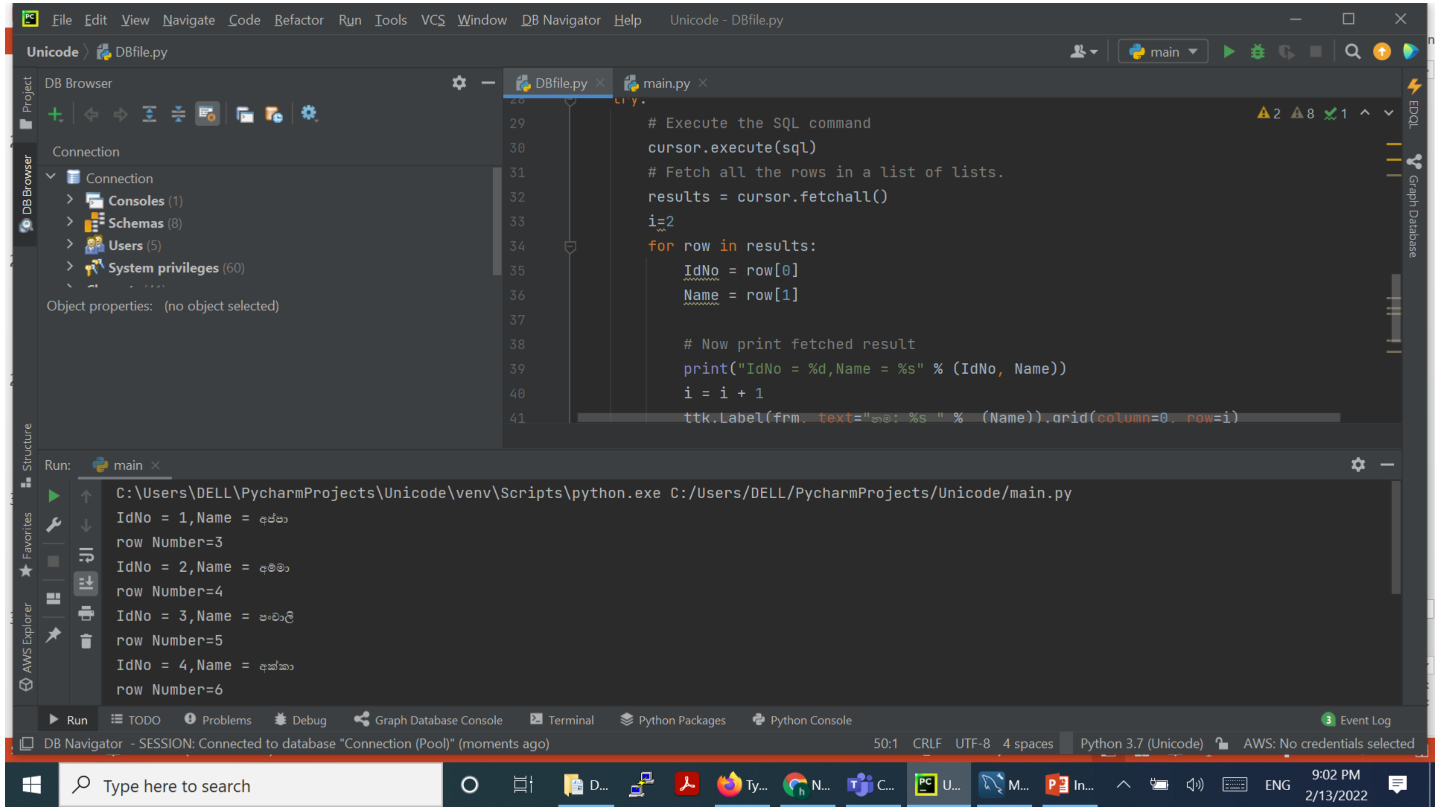


DATABASE CONNECTION USING PYTHON

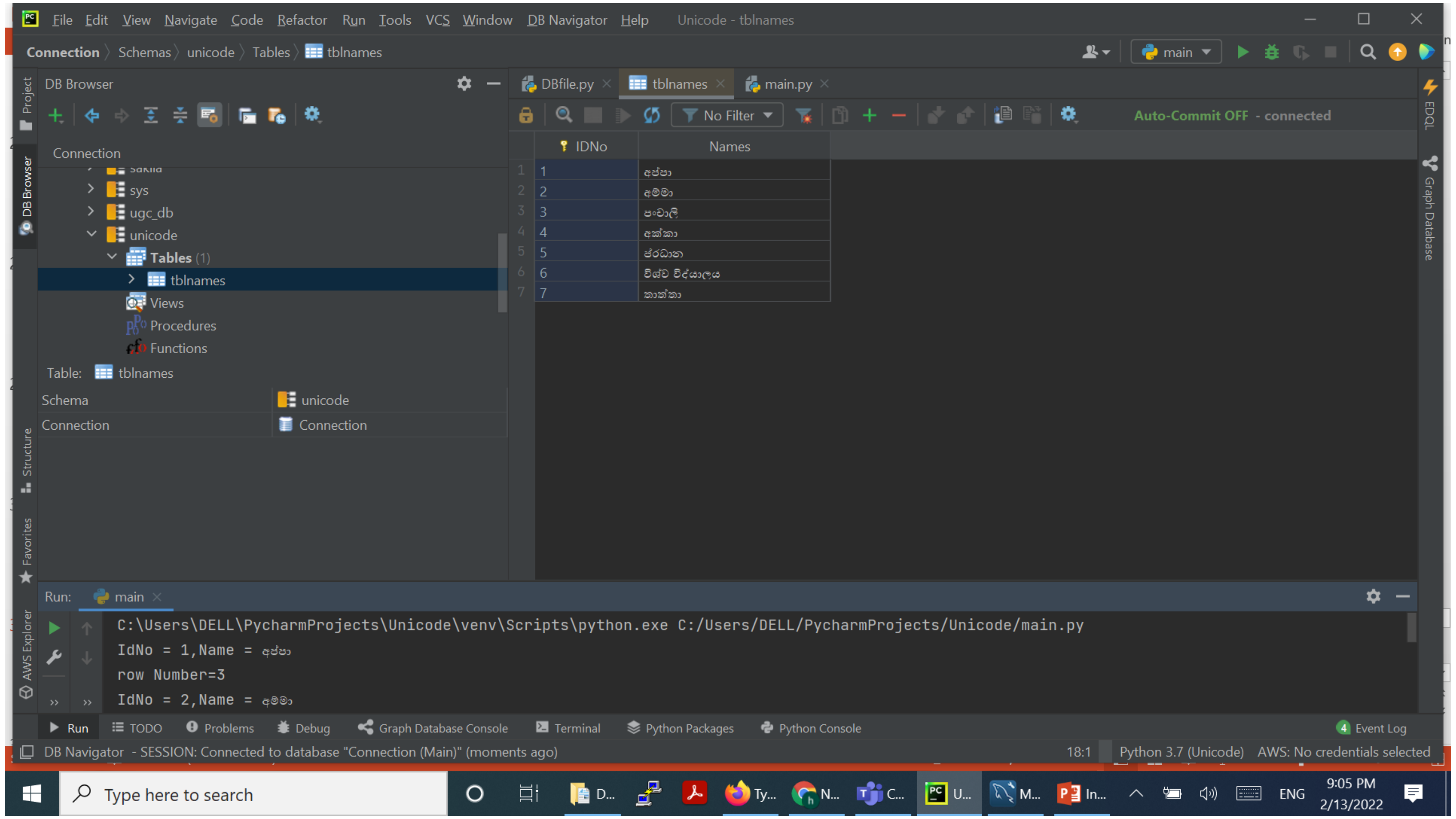
```
# open database connection

db = pymysql.connect(host="Localhost", user="root", passwd="har245", database="unicode")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "select * from tblnames"

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Fetch all the rows in a list of lists.
    results = cursor.fetchall()
    i=2
    for row in results:
        IdNo = row[0]
        Name = row[1]

        # Now print fetched result
        print("IdNo = %d,Name = %s" % (IdNo, Name))
        i = i + 1
        ttk.Label(frm, text="නම: %s " %  (Name)).grid(column=0, row=i)
        print("row Number=%d" % i)
except:
    print("Error: unable to fetch data")

    # disconnect from server
db.close()
ttk.Button(frm, text="Quit", command=root.destroy).grid(column=5, row=0)
root.mainloop()
```

PLUGINS SETUP

- DB Browser



DB BROWSER CONT.



FONTS

- True Type Fonts-True Type Fonts is an outline fonts standard developed by Apple and Microsoft.
- The font was developed in 1980s as a competitor for Adobe Type 1 fonts
- It has become the most common format for fonts on the classic Mac OS, macOS, and Microsoft Windows operating systems.
- TrueType fonts are not made up of individual pixels, but based on the principle of vector graphics
- the TrueType format uses quadratic Bézier curves and can give very detailed instructions to the renderer for hinting.

TRUE TYPE FONTS CONT…

- Compact Font Format (CFF), which is used in the compression processes for the Type2 fonts.
- True Type Font Technology consists of two components: True Type Font file and outline format
- and the TrueType rasterizer, a piece of software built into System 7.x on the Apple Macintosh range of computers, and also into Microsoft's Windows family of operating systems.

TRUE TYPE RASTERIZER

- The TrueType font technology consists of two parts: the description of the fonts themselves (the TrueType font files), and the program which reads the font description and generates the bitmaps (the TrueType Rasterizer).
- The TrueType Rasterizer is a computer program which is typically incorporated as part of an operating system or printer control software.

  The job of the TrueType Rasterizer is to generate character bitmaps for screens and printers (otherwise known as raster devices). It accomplishes this by performing the following tasks:

  Reading the outline description of the character (lines and splines) from the TrueType font file.

  Scaling the outline description of the character to the requested size and device resolution.

  Adjusting the outline description to the pixel grid (based on hinting information).

  Filling the adjusted outline with pixels (scan conversion).

OPEN TYPE FONTS

- The OpenType font format is a widely-supported format for font data with a rich set of capabilities for digital typography.
- It was developed as an extension of the original TrueType format.
- Glyph outline data can use the CFF or CFF version 2 formats, as well as the TrueType glyph format.
- Multicolor glyph presentation is supported using embedded color bitmaps or SVG documents, or using layered compositions of colored, outline-format glyphs defined within the font.
- All Unicode characters can be supported, including supplementary-plane characters, as well as Unicode variation sequences.

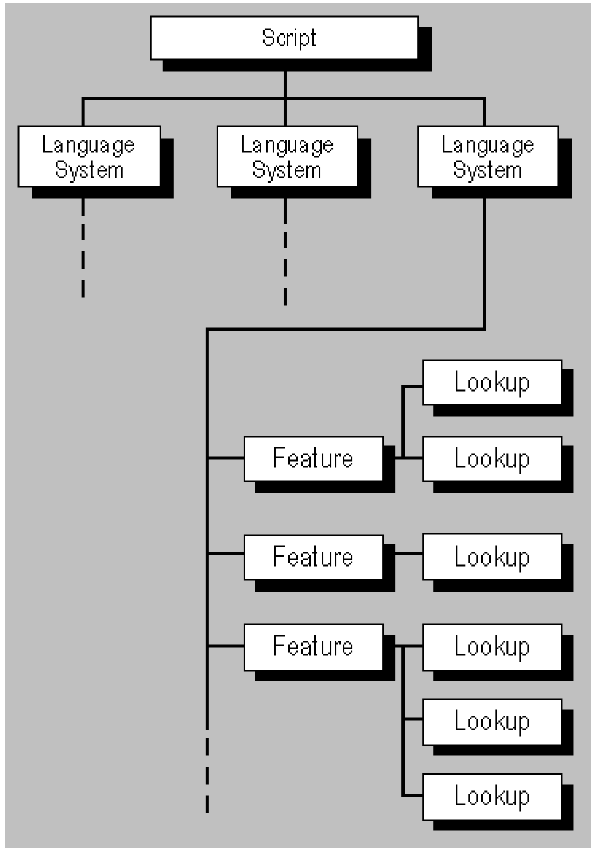
OPEN TYPE FONT CONT…

- OpenType Layout tables provide the advanced typographic capabilities needed for high-quality typography as well as for international text using the wide variety of scripts supported in The Unicode Standard
- The *mathematical typesetting table* allows a font to include data required for layout of complex, math formulas.
- OpenType collection files enable multiple fonts that share common data to be housed within a single file, allowing for de-duplication of data. This is especially useful, for example, for sets of CJK (Chinese, Japanese, Korean) fonts of the same design that share most glyphs in common but that vary with locale-specific glyphs for certain characters.

OPEN TYPE FONT CONT…

- A rich mapping between characters and glyphs, allowing for ligatures, positional forms, alternates, and other substitutions.
- Ability to perform two-dimensional positioning and glyph attachment.
- Explicit script and language information, so a text-processing application can adjust its behavior accordingly.
- OpenType Layout tables provide advanced typographic capabilities for high-quality international typography:
- An open format that allows font developers to define their own typographical features.

OPEN TYPE FONT CONT…



OpenType Layout makes use of five tables: GSUB, GPOS, BASE, JSTF, and GDEF.

GSUB AND GPOS

GSUB:Contains information about glyph substitutions to handle single glyph substitution, one-to-many substitution (ligature decomposition), aesthetic alternatives, multiple glyph substitution (ligatures), and contextual glyph substitution.

GPOS:Contains information about X and Y positioning of glyphs to handle single glyph adjustment, adjustment of paired glyphs, cursive attachment, mark attachment, and contextual glyph positioning.

BASE,JSTF AND GDEF

BASE: Contains information about baseline offsets on a script-by-script basis.

JSTF: Contains justification information, including whitespace and Kashida adjustments.

GDEF: Contains information about all individual glyphs in the font: type (simple glyph, ligature, or combining mark), attachment points (if any), and ligature caret (if a ligature glyph).

DATABASE CONNECTION CONT.

Thank you!

Any Questions?