عنوان: **داستانی از Unicode**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۰/۱۱ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: Unicode, utf-8, ASCII

یکی از مباحثی که به نظرم هر دانشجوی رشته کامپیوتر، فناوری اطلاعات و علاقمند به این حوزه باید بداند بحث کاراکترهاست؛ جدا از اینکه همه ما در مورد وجود ascii یا UTF-8 و ... و توضیحات مختصر آن اطلاع داریم ولی عده‌ای از دوستان مثل من هنوز اطلاعات پایه‌ای‌تر و جامع‌تری در این باره نداریم؛ در این مقاله که برداشتی از وب سایت smashing magazine و W3 است به این مبحث می‌پردازیم.

کامپیوترها تنها با اعداد سر و کار دارند نه با حروف؛ پس این بسیار مهم هست که همه کامپیوترها بر روی یک سری اعداد مشخص به عنوان نماینده‌ای از حروف به توافق برسند. این توافق یکسان بین همه کامپیوترها بسیار مهم هست و باید طبق یک استاندارد مشترک استفاده شود تا در همه سیستم‌ها قابل استفاده و انتقال باشد؛ برای همین در سال 1960 اتحادیه استانداردهای آمریکا، یک سیستم رمزگذاری 7 بیتی را ایجاد کرد؛ به نام American Standard Code for Information Interchange یا کد استاندارد سازی شده آمریکایی برای تبادل اطلاعات یا همان ASCII. این هفت بیت به ما اجازه می‌داد تا 128 حرف را کدگذاری کنیم. این مقدار برای حروف کوچک و بزرگ انگلیسی و هم چنین حروف لاتین، همراه با کدگذاری ارقام و یک سری علائم نگارشی و کاراکترهایی از قبیل tab ، space و موارد مشابه و نهایتا کلیدهای کنترلی کافی بود. در سال 1968 این استاندارد توسط رییس جمهور وقت آمریکا لیندون جانسون به رسمیت شناخته شده و همه سیستم‌های کامپیوتری ملزم به رعایت و استفاده از این استاندارد شدند.

برای لیست کردن و دیدن این کدها و نمادهای حرفیشان می‌توان با یک زبان برنامه نویسی یا اسکریپتی آن‌ها را لیست کرد. کد زیر نمونه‌ای از کد نوشته شده در جاوااسکریپت است.

```
<html>

<body>
 <style type="text/css">p {float: left; padding: 0 15px; margin: 0;}</style>

<script type="text/javascript">
 for (var i=0; i<128; i++) document.writeln ((i%32?'':'<p>') + i + ': ' + String.fromCharCode (i) +
'<br>');

</script>
</body>
 </html>
```

در سال‌های بعدی، با قوی‌تر شدن پردازش‌گرها و 8 بیت شدن یک بایت به جای ذخیره 128 عدد توانستند 256 عدد را ذخیره کنند ولی استاندارد اسکی تا 128 کد ایجاد شده بود و مابقی را به عنوان ذخیره نگاه داشتند. در ابتدا کامپیوترهای IBM از آن‌ها برای ایجاد نمادهای اضافه‌تر و همچنین اشکال استفاده می‌کرد؛ مثلا کد 200 شکل ╚ بود که احتمالا برنامه نویسان زمان داس، این شکل را به خوبی به خاطر میاورند یا مثلا حروف یونانی را اضافه کردند که با کد 224 شکل آلفا α بود و بعدها به عنوان code page 437 نامگذاری شد. هر چند که هرگز مانند اسکی به یک استاندارد تبدیل نشد و بسیاری از کشورها از این فضای اضافی برای استانداردسازی حروف خودشان استفاده می‌کردند و در کشورها کدپیج‌های مختلفی ایجاد شد. برای مثال در روسیه کد پیج 885 از کد 224 برای نمایش Я بهره می‌برد و در کد پیج یونانی 737 برای نمایش حرف کوچک امگا ω استفاده می‌شد. این کار ادامه داشت تا زمانیکه مایکروسافت در سال 1980 کد پیج Windows-1251 الفبای سریلیک را ارئه کرد. این تلاش تا سال 1990 ادامه پیدا کرد و تا آن زمان 15 کدپیج مختلف استانداردسازی شده برای الفبایی چون سیریلیک، عربی، عبری و ... ایجاد شد که این استانداردها از ISO-8859-1 شروع و تا ISO-8859-16 ادامه داشت و موقعی که فرستنده پیامی را ارسال می‌کرد، گیرنده باید از کدپیج مورد نظر مطلع می‌بود تا بتواند پیام را صحیح بخواند.

بیایید با یک برنامه علائم را در این 15 استاندارد بررسی کنیم. تکه کدی که من در اینجا نوشتم یک لیست را که در آن اعداد یک تا 16 لیست شده است، نشان میدهد که با انتخاب هر کدام، کدها را از 0 تا 255 بر اساس هر استاندارد به ترتیب نمایش می‌دهد. این کار توسط تعیین استاندارد در تگ متا رخ میدهد.

در زمان بارگذاری، استانداردها با کد زیر به لیست اضافه می‌شوند.در مرحله بعد لیستی که postback را در آن فعال کرده‌ایم، کد زیر را اجرا می‌کند. در این کد ابتدا charset انتخاب شده ایجاد شده و سپس یکی یکی کدها را به کاراکتر تبدیل می‌کنیم و رشته

نهایی را درج می‌کنیم: (دانلود فایل‌های زیر)

```
private String ISO = "ISO-8859-";
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                for (int i = 1; i < 16; i++)
                {
                    ListItem item = new ListItem();
                    item.Text = ISO + i.ToString();
                    item.Value = i.ToString();
                    DropDownList1.Items.Add(item);
                }
                ShowCodes(1);
            }

        }


        protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (DropDownList1.SelectedItem != null)
            {
                int value = int.Parse(DropDownList1.SelectedValue);
                ShowCodes(value);
            }

        }

        private void ShowCodes(int value)
        {
            Response.Charset = ISO + value;
            string s = "";
            for (int i = 0; i < 256; i++)
            {
                char ch = (char)i;
                s += i + "-" + ch;
                s += "<br/>";//br tag
            }
            Label1.Text = s;
        }
```

تقریبا سال 1990 بود که بسیاری از اسناد به همین شیوه‌ها نوشته و ذخیره شد. ولی باز برای بسیاری از زبان‌ها، حتی داشتن یکی دو حرف بیشتر مشکلاتی را به همراه داشت. مثلا حروف بعضی زبان‌ها مثل چینی و ژاپنی که 256 عدد، پاسخگو نبود و با آمدن شبکه‌ای چون اینترنت و بحث بین المللی شدن و انتقال اطلاعات، این مشکل بزرگتر از آنچه بود، شد.

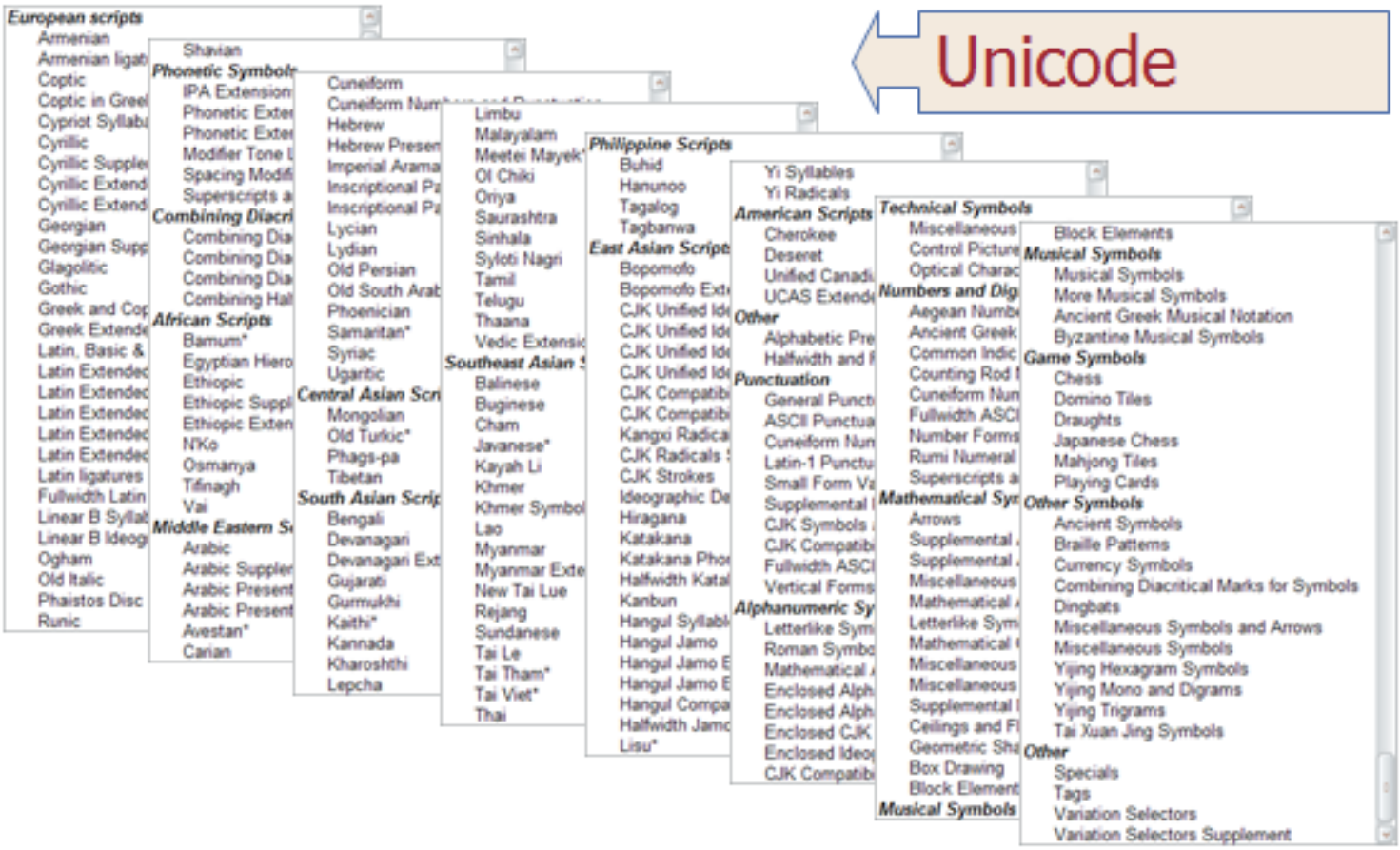
یونیکد نجات بخش

اواخر سال 1980 بود که پیشنهاد یک استاندارد جدید داده شد و در آن به هر حرف و یا نماد در هر زبانی یک عدد یکتا نسبت داده میشد و باید بیشتر از 256 عدد می‌بود که آن را یونیکد نامیدند. در حال حاضر یونیکد نسخه 601 شامل 110 هزار کد می شود. 128 تای آن همانند اسکی است. از 128 تا 255 مربوط به علائم و علامت‌هاست که بیشتر آن‌ها از استاندارد ISO-8859-1 وام گرفته شده‌اند. از 256 به بعد هم بسیاری از علائم تلفظی و ... وجود دارد و از کد 880 زبان یونایی آغاز شده و پس از آن زبان‌های سیریلیک، عبری، عربی و الی آخر ادامه می‌یابند. برای نشان دادن یک کد یونیکد به شکل هگزادسیمال U+0048 نوشته می‌شود و برای تبدیل آن به دسیمال 4*16+8=72 استفاده می‌شود. به هر کد یونیکد، کد پوینت code point گفته میشود.

در ویکی پدیای فارسی، یونیکد اینگونه توضیح داده شده است: *"نقش یونیکد در پردازش متن این است که به جای یک تصویر برای هر نویسه یک کد منحصر به فرد ارایه می‌کند. به عبارت دیگر، یونیکد یک نویسه را به صورت مجازی ارایه می‌کند و کار ساخت تصویر (شامل اندازه، شکل، قلم، یا سبک) نویسه را به عهده نرم‌افزار دیگری مانند مرورگر وب یا واژه‌پرداز می‌گذارد. "*

یونیکد از 8 بیت یا 16 بیت استفاده نمی‌کند و با توجه به اینکه دقیقا 110، 116 کد را حمایت می‌کند به 21 بیت نیاز دارد. هر چند که کامپیوترها امروزه از معمارهای 32 بیتی و 64 بیتی استفاده می‌کنند، این سوال پیش می‌آید که ما چرا نمی‌توانیم کاراکترها را بر اساس این 32 بیت و 64 بیت قرار بدهیم؟ پاسخ این سوال این‌است که چنین کاری امکان پذیر است و بسیاری از نرم افزارهای نوشته شده در زبان سی و سی ++ از wide character حمایت می‌کنند. این مورد یک کاراکتر 32 بیتی به نام wchar_t است که نوعی داده char توسعه یافته هشت بیتی است و بسیاری از مرورگرهای امروزی از آن بهره مند هستند و تا 4 بیلیون کاراکتر را

حمایت می‌کنند.
شکل زیر دسته بندی از انواع زبان‌های تحت حمایت خود را در نسخه 5.1 یونیکد نشان می‌دهد:



کد زیر در جاوااسکریپت کاراکترهای یونیکد را در مرز معینی که برایش مشخص کرده‌ایم نشان می‌دهد:

```
<html>

<body>
 <style type="text/css">p {float: left; padding: 0 15px; margin: 0;}</style>

<script type="text/javascript">
for (var i=0; i<2096; i++)
   document.writeln ((i%256?'':'<p>') + i + ': ' + String.fromCharCode (i) + '<br>');

</script>
</body>
 </html>
```

CSS & Unicode

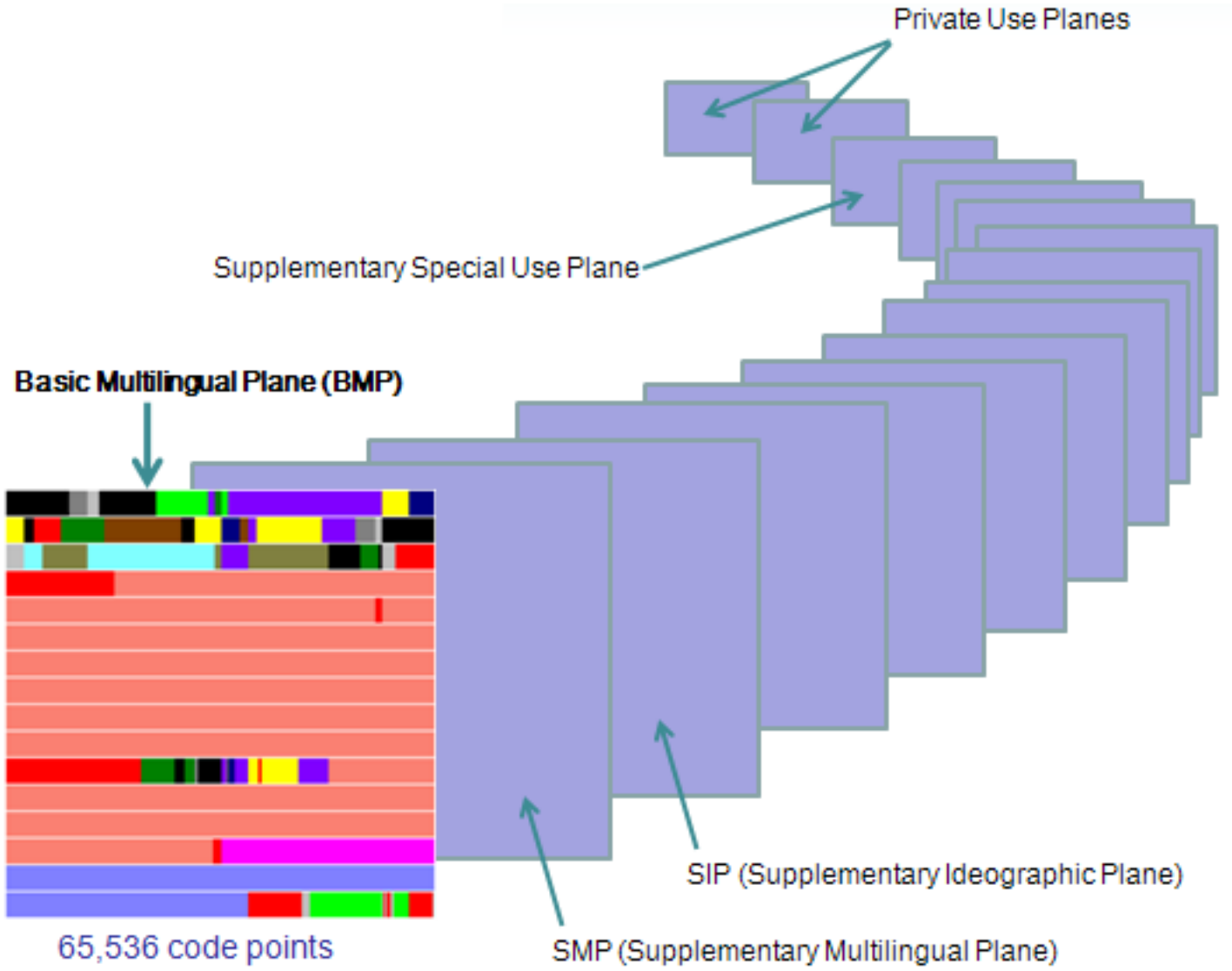
یکی از جذاب‌ترین خصوصیات در css، خصوصیت Unicode-range است. شما میتوانید برای هر کاراکتر یا حتی رنج خاصی از کاراکترها، فونت خاصی را اعمال کنید. به دو نمونه زیر دقت کنید:

```
/* cyrillic */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/mErvLBYg_cXG3rLvUsKT_fesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;
}
```

```
/* greek-ext */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'), url(http://fonts.gstatic.com/s/roboto/v14/-
2n2p-_Y08sg57CNWQfKNvesZW2xOQ-xsNqO47m55DA.woff2) format('woff2');
  unicode-range: U+1F00-1FFF;
}
/* greek */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/u0TOpm082MNkS5K0Q4rhqvesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0370-03FF;
}
/* vietnamese */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/NdF9MtnOpLzo-noMoG0miPesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0102-0103, U+1EA0-1EF1, U+20AB;
}
/* latin-ext */
@font-face {
  font-style: normal;
  src: local('Roboto Regular'), local('Roboto-Regular'),
url(http://fonts.gstatic.com/s/roboto/v14/Fcx7Wwv8OzT71A3E1XOAjvesZW2xOQ-xsNqO47m55DA.woff2)
format('woff2');
  unicode-range: U+0100-024F, U+1E00-1EFF, U+20A0-20AB, U+20AD-20CF, U+2C60-2C7F, U+A720-A7FF;
}
```

در صورتی که در Unicode-range، تنها یک کد مانند U+20AD نوشته شود، فونت مورد نظر فقط بر روی کاراکتری با همین کد اعمال می‌شود. ولی اگر بین دو کد از علامت - استفاده شود، فونت مورد نظر بر روی کاراکترهایی که بین این رنج هستند اعمال می‌شود U+0025-00FF و حتی می‌توان اینگونه نوشت U+4?? روی کاراکترهایی در رنج U+400 تا U+4FF اعمال می‌شوند. برای اطلاعات بیش‌تر به اینجا و اینجا مراجعه کنید.

به 65536 کد اول یونیکد Basic Multilingual Plan یا به اختصار BMP می‌گویند و شامل همه کاراکترهای رایجی است که مورد استفاده قرار می‌گیرند. همچنین یونیکد شامل یک فضای بسیار بزرگ خالی است که به شما اجازه توسعه دادن آن را تا میلیون‌ها کد می‌دهد. به کاراکترهایی که در این موقعیت قرار می‌گیرند supplementary characters یا کاراکترهای مکمل گویند. برای اطلاعات بیشتر می‌توانید به سایت رسمی یونیکد مراجعه کنید. در اینجا هم مباحث آموزشی خوبی برای یونیکد دارد، هر چند کامل‌تر آن در سایت رسمی برای نسخه‌های مختلف یونیکد وجود دارد.



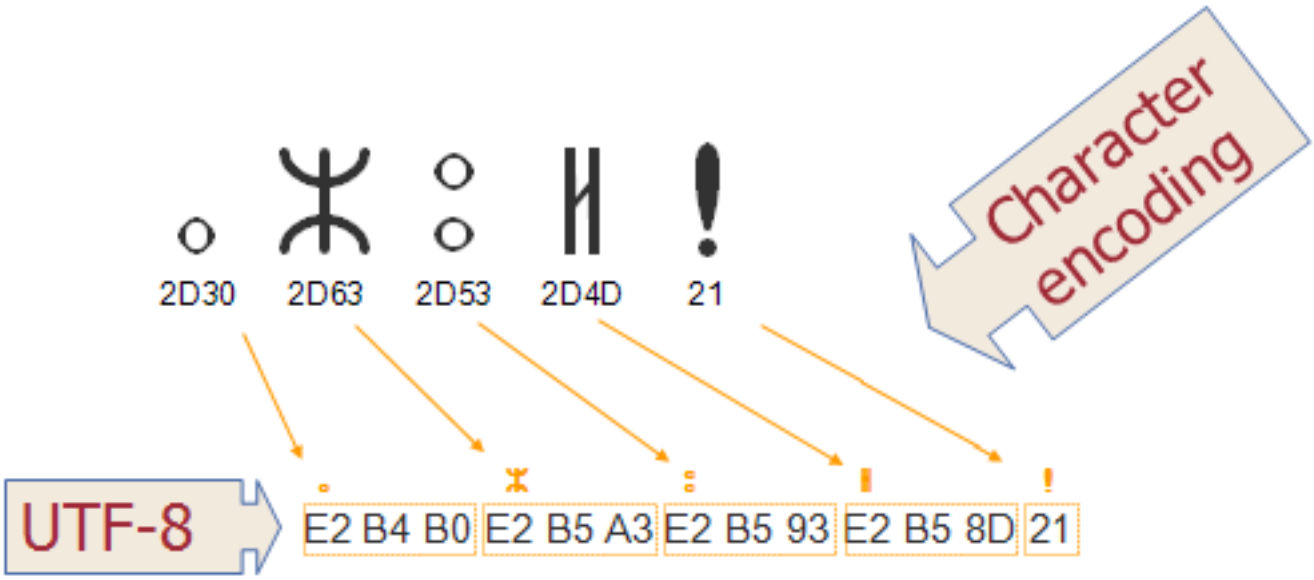
UTF-8 نجات بخش می‌شود

بسیاری از مشکلات ما حل شد. همه حروف را داریم و مرورگرها نیز همه حروف را میشناسند؛ ولی برای ما دو مشکل ایجاد کرده است:

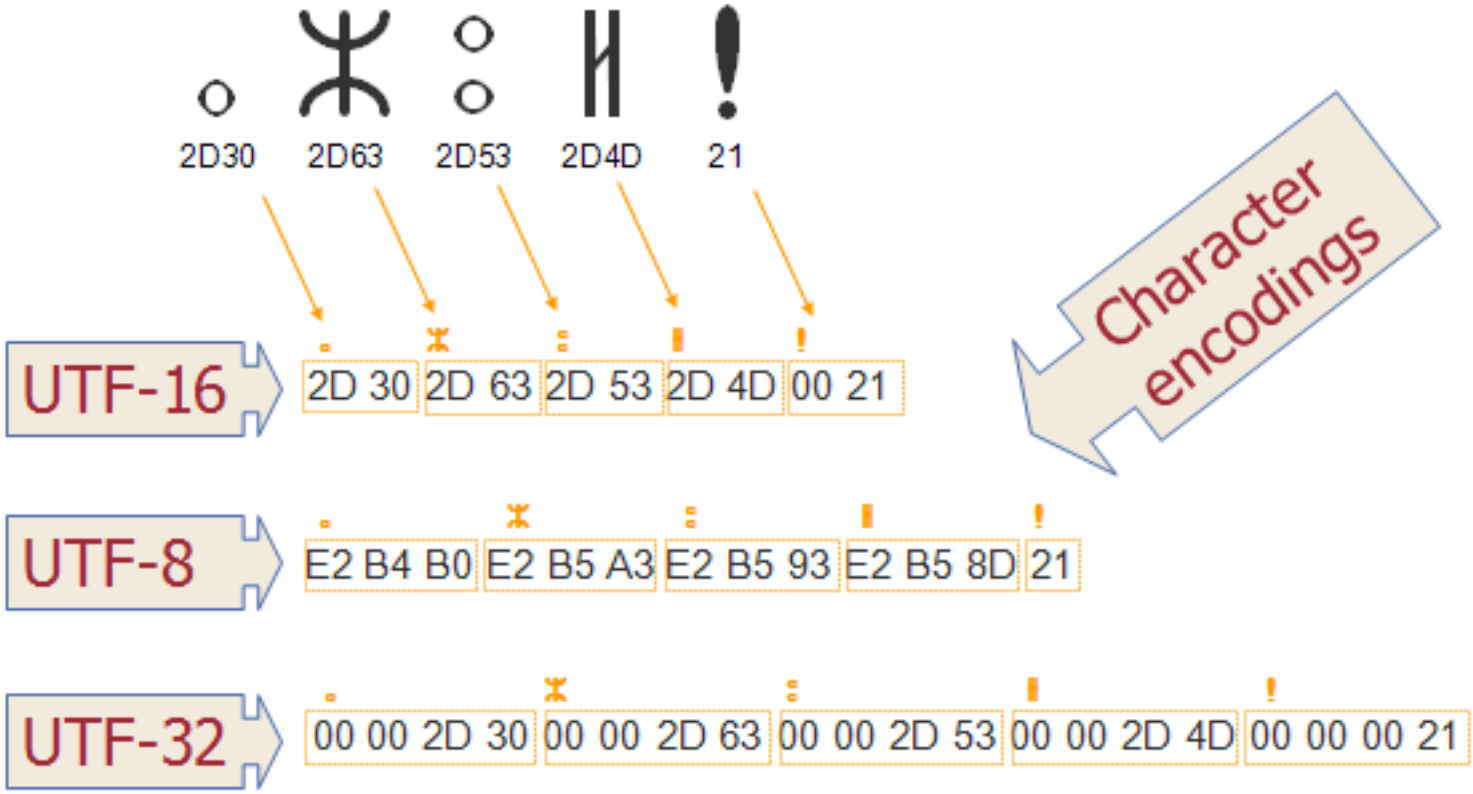
بسیاری از نرم افزارها و پروتکل‌ها هنوز 8 بیتی کار می‌کنند.

اگر یک متن انگلیسی ارسال کنید، 8 بیت هم کافی است ولی در این حالت 32 بیت جابجا می‌شود؛ یعنی 4 برابر و در ارسال و دریافت و پهنای باند برایمان مشکل ایجاد می‌کند.

برای حل این مشکل استانداردهای زیادی چون USC-2 یا UTF-16 ایجاد شدند ولی در سال‌های اخیر برنده رقابت، UTF-8 بود که مخفف عبارت Universal Character Set Transformation Format 8 bit می‌باشد. این کدگذاری بسیار هوشمندانه عمل می‌کند. موقعی که شما کاراکتری را وارد می‌کنید که کدش بین 0 تا 255 است، 8 بیت به آن اختصاص می‌دهد و اگر در محدوده‌ای است که بتوان دو بایت را به آن اختصاص داد، دوبایت و اگر بیشتر بود، سه بایت و اگر باز بیشتر بود 4 بایت به آن اختصاص میدهد. پس با توجه به محدوده کد، تعداد بایت‌ها مشخص می‌شوند. بنابراین یک متن نوشته شده انگلیسی که مثلا از کدهای بین 0تا 128 استفاده می‌کند و فرمت ذخیره آن UTF-8 باشد به ازای هر کارکتر یک بایت ذخیره می‌کند.



مقایسه‌ای بین نسخه‌های مختلف :



همانطور که می‌بینید UTF-8 برای کاراکترهای اسکی، از یک بایت و برای دیگر حروف از دوبایت و برای بقیه BMPها از سه بایت استفاده میکند و در صورتی که کاراکتری در ناحیه مکمل supplementary باشد، از چهار بایت استفاده خواهد کرد. UTF-16 از دو بایت برای نمایش کاراکترهای BMP و از 4 بایت برای نمایش کاراکترهای مکمل استفاده می‌کند و در UTF-32 از 4 بایت برای همه

کاراکترها یا کد پوینت‌ها استفاده می‌شود.