Normal Form
Grapheme

Jonathan Worthington
Hi!

I'm still Jonathan.
Unicode
A unified scheme to encode and manipulate all the writing systems of the world (present, historical, math, emoji…)}
How hard could that be?
Well, let's look at some languages.
Just give numbers to each character in the alphabet. Easy!

Aardvarks don’t make a tasty meal.
Мне нравится пиво!

Just another alphabet. More numbers. Easy!
Chinese?

你好吗？很好，谢谢。

A character set. Thousands and thousands more numbers.
Slovak

Slovenky sú veľmi pekné.

Doesn't seem too scary. I mean, it's Latin-based. But...
Slovenky sú veľmi pekné.

...what about the diacritics? Do we give numbers to them?
Slovak

Slovenky sú veľmi pekné.

Or number the combinations of letters with diacritics?
Slovak

Slovenky sú veľmi pekné.

Spoiler: it's Unicode, so of course we...do both!
Korean

안녕하세요

A syllabary, with each syllable block made of 2-3 "atoms".
And yes, again, there are two ways to represent it.
So how do we even compare strings?!
Unicode defines normalization forms. Comparison should be done on normalized strings.
Unicode is fairly complex, because human language itself is complex.
So how well do our programming languages cope with Unicode?
U+1F639

CAT FACE WITH TEARS OF JOY
U+0044 U+0323 U+0307

LATIN CAPITAL LETTER D

COMBINING DOT BELOW

COMBINING DOT ABOVE
C#/.Net
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
}
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
}

using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);
}

😹
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);
}


using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);
    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
}

ישי
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);

    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
}

​
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);

    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
    fh.WriteLine(dWithDots.Length);
}

RH2.
using (var fh = new StreamWriter(
    File.OpenWrite("output"))))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);

    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
    fh.WriteLine(dWithDots.Length);
}
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);
    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
    fh.WriteLine(dWithDots.Length);
    dWithDots = dWithDots.Normalize();
    fh.WriteLine(dWithDots.Length);
}
using (var fh = new StreamWriter(File.OpenWrite("output")))
{
    var catFace = "\U0001F639";
    fh.WriteLine(catFace);
    fh.WriteLine(catFace.Length);
    var dWithDots = "D\u0323\u0307";
    fh.WriteLine(dWithDots);
    fh.WriteLine(dWithDots.Length);
    dWithDots = dWithDots.Normalize();
    fh.WriteLine(dWithDots.Length);
}

-cat-
Java
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());
String dWithDots = "D\u0323\u0307";
writer.println(dWithDots);
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());
String dWithDots = "D\u0323\u0307";
writer.println(dWithDots);
PrintWriter writer = new PrintWriter("output-java", "UTF-8");  
String catFace = new String(Character.toChars(0x1F639));  
writer.println(catFace);  
writer.println(catFace.length());  
String dWithDots = "D\u0323\u0307";  
writer.println(dWithDots);  
writer.println(dWithDots.length());
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());

String dWithDots = "D\u0323\u0307";
writer.println(dWithDots);
writer.println(dWithDots.length());
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());

String dWithDots = "D\u0323\u0307";
writer.println(dWithDots);
writer.println(dWithDots.length());
dWithDots = Normalizer.normalize(dWithDots, Normalizer.Form.NFC);
writer.println(dWithDots.length());
PrintWriter writer = new PrintWriter("output-java", "UTF-8");
String catFace = new String(Character.toChars(0x1F639));
writer.println(catFace);
writer.println(catFace.length());
String dWithDots = "D\u0323\u0307";
writer.println(dWithDots);
writer.println(dWithDots.length());
dWithDots = Normalizer.normalize(dWithDots, Normalizer.Form.NFC);
writer.println(dWithDots.length());
Python
(3, with gooder Unicode)
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\\U0001F639"
f.write(catFace + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")
    dWithDots = "D\u0323\u0307"
    f.write(dWithDots + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
catFace = "\U0001F639"
f.write(catFace + "\n")
f.write(str(len(catFace)) + "\n")

dWithDots = "D\u0323\u0307"
f.write(dWithDots + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")

dWithDots = "D\u0323\u0307"
    f.write(dWithDots + "\n")
    f.write(str(len(dWithDots)) + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")

dWithDots = "D\\u0323\\u0307"
    f.write(dWithDots + "\n")
    f.write(str(len(dWithDots)) + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")

dWithDots = "D\u0323\u0307"
    f.write(dWithDots + "\n")
    f.write(str(len(dWithDots)) + "\n")
    dWithDots = unicodedata.normalize('NFC', dWithDots)
    f.write(str(len(dWithDots)) + "\n")
with codecs.open("output-python", "w", "utf-8") as f:
    catFace = "\U0001F639"
    f.write(catFace + "\n")
    f.write(str(len(catFace)) + "\n")

dWithDots = "D\u0323\u0307"
    f.write(dWithDots + "\n")
    f.write(str(len(dWithDots)) + "\n")
    dWithDots = unicodedata.normalize('NFC', dWithDots)
    f.write(str(len(dWithDots)) + "\n")
given open("output-perl6", :w) {
    my $catFace = "\x1F639";
    say($catFace);
    .close;
}
given open("output-perl6", :w) {
    my $catFace = "\x1F639";
    .say($catFace);
    .close;
}
given open("output-perl6", :w) {
  my $catFace = "\x1F639";
  .say($catFace);
  .say($catFace.chars);
  .close;
}
given open("output-perl16", :w) {
    my $catFace = "\x1F639";
    .say($catFace);
    .say($catFace.chars);
    .close;
}
given open("output-perl6", :w) { 
  my $catFace = "\x1F639";
  .say($catFace);
  .say($catFace.chars);
  my $dWithDots = "D\x0323\x0307";
  .say($dWithDots);
  .close;
}
given open("output-perl16", :w) {
    my $catFace = "\x1F639";
    say($catFace);
    say($catFace.chars);
    my $dWithDots = "D\x0323\x0307";
    say($dWithDots);
    close;
}
given open("output-perl6", :w) {  
  my $catFace = "\x1F639";
  .say($catFace);
  .say($catFace.chars);

  my $dWithDots = "D\x0323\x0307";
  .say($dWithDots);
  .say($dWithDots.chars);

  .close;
}
given open("output-perl6", :w) {
  my $catFace = "\\x1F639";
  say($catFace);
  say($catFace.chars);

  my $dWithDots = "D\\x0323\\x0307";
  say($dWithDots);
  say($dWithDots.chars);

  close;
}
U+270C

VICTORY HAND
So, how does Perl 6 get the right answers, and why do others get the wrong answers?
The 3 levels of Unicode

- **Graphemes**: Things a human would consider a character.
- **Codepoints**: Things the Unicode spec gives a number to.
- **Bytes, C# chars, etc.**: How things look on disk or in memory.
C# and Java store strings as **UTF-16**, using arrays of 16-bit integers.
They work down here!

- **Graphemes**
  - Things a human would consider a character

- **Codepoints**
  - Things the Unicode spec gives a number to

- **Bytes, C# chars, etc.**
  - How things look on disk or in memory
So when you ask for the length of a string...
...they tell you the number of 16-bit things needed to represent the string in UTF-16.
It's...enterprise.
Some languages have codepoint-level strings

- **Graphemes**: Things a human would consider a character
- **Codepoints**: Things the Unicode spec gives a number to
- **Bytes, C# chars, etc.**: How things look on disk or in memory
The string length is the number of codepoints.
Which is close, but...
Also, it's common to store the strings in memory as UTF-8, meaning string indexing, etc. is $O(n)$, not $O(1)$. 
Perl 6 strings work at the grapheme level

- Graphemes: Things a human would consider a character
- Codepoints: Things the Unicode spec gives a number to
- Bytes, C# chars, etc.: How things look on disk or in memory
LATIN CAPITAL LETTER D
COMBINING DOT BELOW
COMBINING DOT ABOVE

= 3 codepoints
= 2 normalized (NFC) codepoints
= 1 grapheme
And...we get to have $O(1)$ string indexing.
How do we do it?
We first turn all input strings into NFC.

Then, if we still have multi-codepoint graphemes, we create synthetic codepoints.
Data rocks!

We represent synthetics using negative integers.
But only internally. You never get to see them.

And on output, we turn everything back into the usual NFC again.
Awesome!
Questions?