Giovani Pereira Follow
Jun 7, 2019 · 3 min read · Listen




Photo by JuniperPhoton on Unsplash

# Dealing with Asian characters in swift

> *Based on the speech of Tomoki Yamaguchi, from LINE @ altconf 2019*

Localizing an app in an Asian language, if you are not particularly familiar with it, can be more difficult than you expect. Not only because it's a totally different language, and the characters, *letters* are different from what you are used to, but interacting, typing, and reading are different too.

With that in mind, Tomoki Yamaguchi made a lightning speech on the Altconf 2019 to give a few advice if you're looking into making an app available in an Asian country.

### Remember there are multiple languages

If you are looking to the Asian market, you will step into a bunch of different idioms;

Some of them are left-to-right writings, and even vertical. To this, *auto-layout* already has a lot of tools to automatically change the position of your views, based on *leading* and *trailing* constraints.

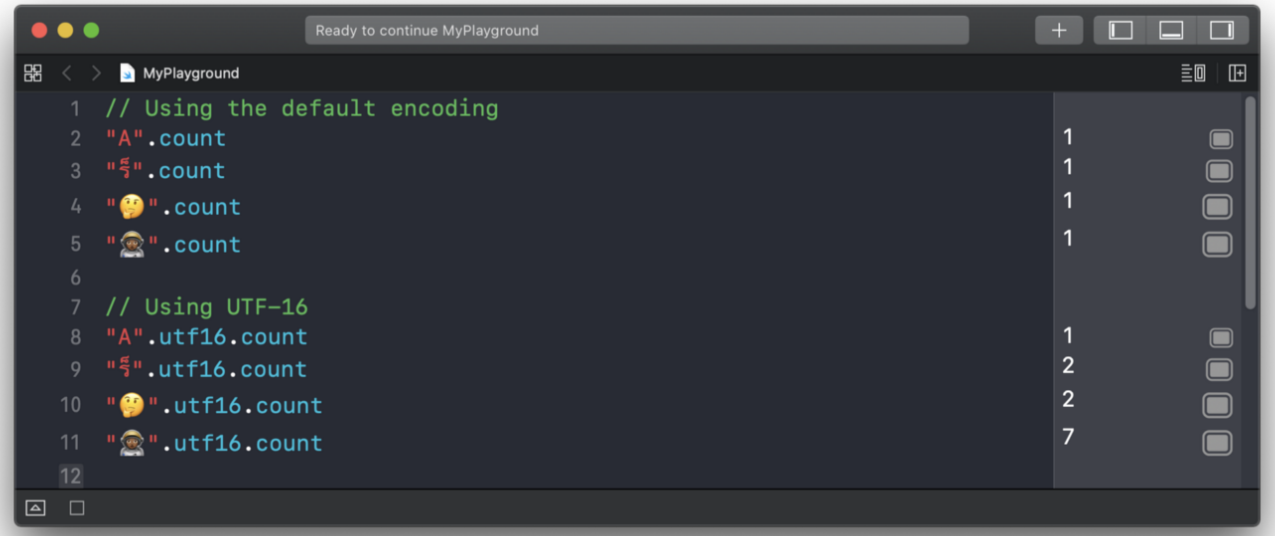
One particularity of the Thai language is that the font exceeds the frame boxes of the UILabels, so setting `clipToBounds` to `true` will hide pieces of the labels.


### Font size

Different of the characters we are used to, which can have different widths, and usually a character is *half-width*, most of the Asian characters are always *full-width*, which means they occupy more space horizontally (similar to an emoji size when compared to a single letter).

### Character counting

Let's suppose you want to get the number of characters someone typed, to do any kind of validation. Characters can be counted differently depending on your counting strategy.




Check the difference in here, the character count is 1 in UTF-8, but the same character can be more in a different encoding

For example, a single Korean character 있 can be more than one character on a different encoding (UTF-8, for example). The same will happen when you count string sizes with emojis.

I had a problem with this when creating an app (the *Fieldbook*) because we needed to split strings based on the cursor position, and when counting with emojis it simply didn't work. I had to change the count for the number of characters in UTF-16 to get the behavior I wanted.

Probably the result of calling `.count` is likely to be what you are looking for, but it's always nice to keep in mind that maybe... it doesn't. Because of that, you cannot block an input by the number of characters, just validate it after the input.

Keep in mind that complex characters are built as the composition of multiple characters. In emojis, if you add skin tones or any other variations, is like you are adding another character on top of that.

**Replacing characters**


iOS Japanese Keyboard, while typing the characters combine into new ones

Keep that in mind the next time you use these methods, to ensure you are replacing the correct part of the string, not a sub-character.

## Resume

So, we can resume the lessons here in 4 points:



- Be careful about counting strategy
- Don’t block input while editing

And if I may add one more:

- Have a proficient person check your app before you make it available on a foreign language — that should always help a lot 😉

And have fun exploring new markets around the world!

*Originally published at https://giovaninppc.github.io on June 8, 2019.*