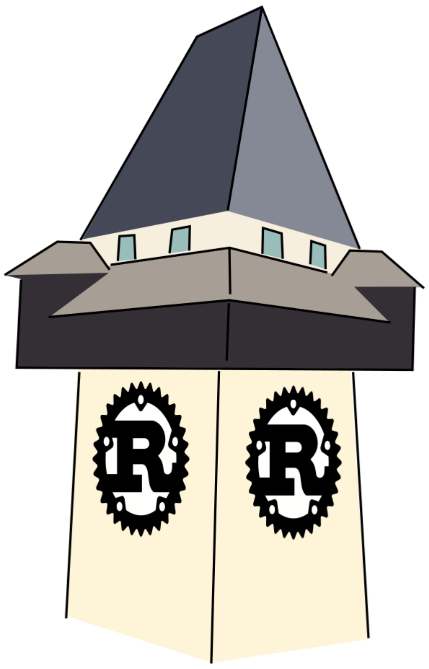


Unicode & Jeopardy

18th of December 2019

Rust Graz, lab10

PROLOGUE

PROLOGUE

CLARIFICATION 1: DEBUG/RELEASE BUILD, STACK SIZE, CARGO

```rust
fn count_calls(n: u64) -> u64 {
    println!("{:p}", &n);
    if n < 1 {
        0
    } else {
        1 + count_calls(n - 1)
    }
}

fn main() {
    println!("{}", count_calls(174470))
}
```

CLARIFICATION 1: DEBUG/RELEASE

```
% cargo run
…
0x7ffc9324f6b0
0x7ffc9324f610
0x7ffc9324f570

thread 'main' has overflowed its stack
fatal runtime error: stack overflow
[1]    11645 abort      cargo run
```

⇒ result of last time: 160 bytes per stackframe

CLARIFICATION 1: DEBUG/RELEASE

```
% cargo run --release
…
0x7ffe628fe5a8
0x7ffe628fe548
0x7ffe628fe4e8

thread 'main' has overflowed its stack
fatal runtime error: stack overflow
[1]    11803 abort      cargo run --release
```

⇒ 96 bytes per stackframe

CLARIFICATION 1: DEBUG/RELEASE

"The Manifest Format" via Cargo book

```
# The development profile, used for `cargo build`.
[profile.dev]
# controls the `--opt-level` the compiler builds with.
# 0-1 is good for debugging. 2 is well-optimized. Max is 3.
# 's' attempts to reduce size, 'z' reduces size even more.
opt-level = 0

# (u32 or bool) Include debug information (debug symbols).
# Equivalent to `-C debuginfo=2` compiler flag.
debug = true

# Link Time Optimization usually reduces size of binaries
# and static libraries. Increases compilation time.
# If true, passes `-C lto` flag to the compiler, and if a
# string is specified like 'thin' then `-C lto=thin` will
```

CLARIFICATION 1: DEBUG/RELEASE

debug build (160 bytes, 2.4MB):

```
% ls -l ./target/debug/buildtest
-rwxrwxr-x 2 user user 2514680 Dec 17 22:15 ./target/debug/bui
```

release build (96 bytes, 2.4MB):

```
% ls -l ./target/release/buildtest
-rwxrwxr-x 2 user user 2497912 Dec 17 22:22 ./target/release/b
```

CLARIFICATION 1: DEBUG/RELEASE

Old `Cargo.toml`:

```
[package]
name = "buildtest"
version = "0.1.0"
authors = ["meisterluk <admin@lukas-prokop.at>"]
edition = "2018"

# See more keys and their definitions at https://doc.rust-lang

[dependencies]
```

CLARIFICATION 1: DEBUG/RELEASE

New `Cargo.toml`:

```
[package]
name = "buildtest"
version = "0.1.0"
authors = ["meisterluk <admin@lukas-prokop.at>"]
edition = "2018"

# See more keys and their definitions at https://doc.rust-lang

[profile.release]
opt-level = 3
debug = true
lto = true

[dependencies]
```

CLARIFICATION 1: DEBUG/RELEASE

custom release build (always opt-level=3)

- `debug=true, lto=false`:
  96 bytes stackframe & 2507520 bytes executable
- `debug=true, lto=true`: 96 & 974440 bytes
- `debug=false, lto=true`: 96 & 965904 bytes

I don't know how to go below 96 bytes stack frames.

CLARIFICATION 2: `PartialEq` AND `Eq`

`PartialEq`

symmetric
: `a == b` implies `b == a`; and

transitive
: `a == b` and `b == c` implies `a == c`.

`Eq`

additionally reflexive
: `a == a`;

CLARIFICATION 2: `PartialEq` AND `Eq`

```rust
use std::f64;
fn main() {
    println!("{}", f64::NAN == f64::NAN); // false
}
```

⇒ If you implement `PartialEq` then `#[derive(Eq)]` as well unless you can't

Related traits: `Hash`, `PartialOrd`, `Ord`

DIALOGUE: UNICODE

DIALOGUE: UNICODE

UNICODE

Unicode maps numbers to *code points*.

Unicode 12.1 maps numbers to 137,994 unicode code points.

How can we encode 137,994 Unicode code points to bytes? ⇒ Unicode Transformation Format (UTF).

れい　わ

令和 ⇒ ㋿

- 2019/04/30: Emperor Akihito abdicated. 2019/05/01: Emperor Naruhito ascended the throne. 1 character added in 12.1
- Previously へいせい 平成 ⇒ ㍻: U+5E73 CJK UNIFIED IDEOGRAPH-5E73 and U+6210 CJK UNIFIED IDEOGRAPH-6210 merged into U+337B SQUARE ERA NAME HEISEI

UNICODE

I came up with some UTF. I will introduce 5 versions of a “Complementary Properties Encoding” (CPE). Let's discuss its properties.

CPE5

2 bytes = 16 bits. Fixed-width encoding. What are potential problems?

$$\boxed{\text{xxxx xxxx}}\ \boxed{\text{xxxx xxxx}}\quad 2^{16}$$

CPE4

xxxx xxx1				2^7
xxxx xxx0	xxxx xxx1			$2^{2\times7}$
xxxx xxx0	xxxx xxx1	xxxx xxx1		$2^{3\times7}$
xxxx xxx0	xxxx xxx1	xxxx xxx1	xxxx xxx1	$2^{4\times7}$

CPE3

1xxx xxxx				2^7
0xxx xxxx	1xxx xxxx			$2^{2\times7}$
0xxx xxxx	0xxx xxxx	1xxx xxxx		$2^{3\times7}$
0xxx xxxx	0xxx xxxx	0xxx xxxx	1xxx xxxx	$2^{4\times7}$

CPE2

00xx xxxx				2^6
01xx xxxx	xxxx xxxx			2^{6+8}
10xx xxxx	xxxx xxxx	xxxx xxxx		$2^{6+8\times2}$
11xx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	$2^{6+8\times3}$

CPE1

1xxx xxxx				2^7
001x xxxx	01xx xxxx			2^{5+6}
0001 xxxx	01xx xxxx	01xx xxxx		$2^{4+6\times2}$
0000 1xxx	01xx xxxx	01xx xxxx	01xx xxxx	$2^{3+6\times3}$

UTF-8

0xxx xxxx				2^{7}
110x xxxx	10xx xxxx			2^{5+6}
1110 xxxx	10xx xxxx	10xx xxxx		$2^{4+6\times2}$
1111 0xxx	10xx xxxx	10xx xxxx	10xx xxxx	$2^{3+6\times3}$

UTF-8 PROPERTIES

Backward/ASCII compatibility

Setting one special bit of single byte, we have 7 remaining bit with same assignment like ASCII

Extended ASCII detection/fallback

UTF-8 multibyte strings are rarely linguistically legit Extended ASCII strings.

þ ⇒ Ã¾, ø ⇒ Ã¸, ß ⇒ ÃŸ

Prefix freedom

There is no whole code word in the system that is a prefix of any other code word in the system

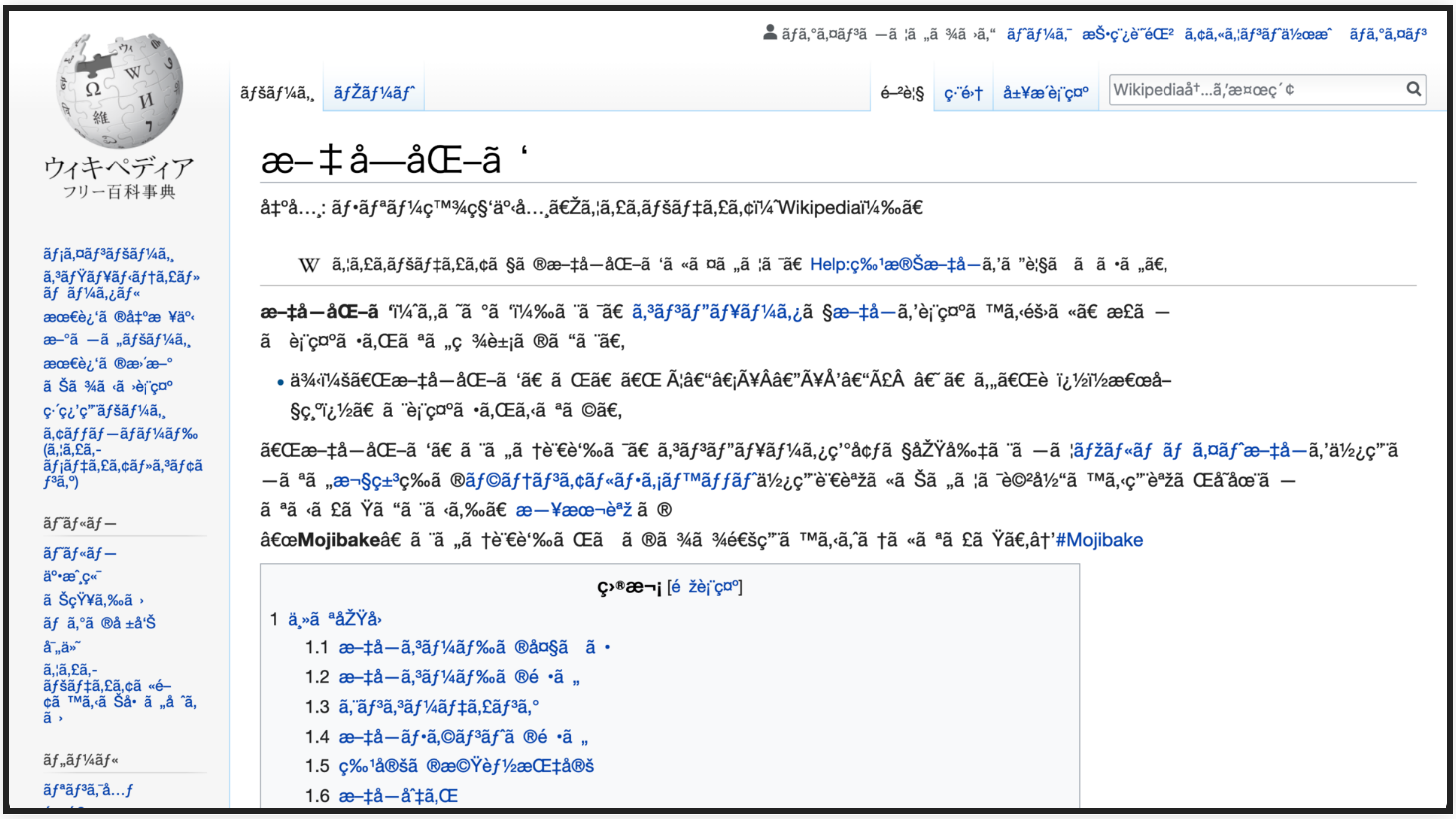
UTF-8 PROPERTIES

Self-synchronization
: If we jump to some byte, we can easily determine the start of the next character

Sorting order
: Lexicographical order of bytes equal unicode codepoint order

via Wikipedia

UTF-8 FALLBACK EXAMPLE



ãƒã,°ã,¤ãƒ³ã —ã ¦ã „ã ¾ã ›ã,“ ãƒˆãƒ¼ã,¯ æŠ•ç¨¿è¨˜éŒ² ã,¢ã,«ã,¦ãƒ³ãƒˆä½œæˆ ãƒã,°ã,¤ãƒ³

ãƒšãƒ¼ã,¸ ãƒŽãƒ¼ãƒˆ é–²è¦§ ç·¨é›† å±¥æ´è¡¨ç¤º Wikipediaå†…ã,'æ¤œç´¢

ウィキペディア
フリー百科事典

ãƒ¡ã,¤ãƒ³ãƒšãƒ¼ã,¸
ã,³ãƒŸãƒ¥ãƒ‹ãƒ†ã,£ãƒ» ãƒ ãƒ¼ã,¿ãƒ«
æœ€è¿'ã ®å‡ºæ ¥äº‹
æ–°ã —ã „ãƒšãƒ¼ã,¸
æœ€è¿'ã ®æ›´æ–°
ã Šã ¾ã ‹ã ›è¡¨ç¤º
ç·´ç¿'ç"¨ãƒšãƒ¼ã,¸
ã,¢ãƒƒãƒ—ãƒãƒ¼ãƒ‰ (ã,¦ã,£ã,ãƒ¡ãƒ‡ã,£ã,¢ãƒ»ã,³ãƒ¢ãƒ³ã,º)

ãƒ˜ãƒ«ãƒ—

ãƒ˜ãƒ«ãƒ—
äº•æˆ¸ç«¯
ã ŠçŸ¥ã,‰ã ›
ãƒ ã,°ã ®å ±å'Š
å¯„ä»˜
ã,¦ã,£ã,ãƒšãƒ‡ã,£ã,¢ã «é–¢ã ™ã,‹ã Šå• ã „å ˆã,ã ›

ãƒ„ãƒ¼ãƒ«

ãƒªãƒ³ã,¯å…ƒ

æ–‡å—åŒ–ã '

å‡ºå…¸: ãƒ•ãƒªãƒ¼ç™¾ç§'äº‹å…¸ã€Žã,¦ã,£ã,ãƒšãƒ‡ã,£ã,¢ï¼ˆWikipediaï¼‰ã€

W ã,¦ã,£ã,ãƒšãƒ‡ã,£ã,¢ã §ã ®æ–‡å—åŒ–ã 'ã «ã ¤ã „ã ¦ã ¯ã€ Help:ç‰¹æ®Šæ–‡å—ã,'ã "è¦§ã ã ã •ã „ã€‚

æ–‡å—åŒ–ã 'ï¼ˆã‚‚ã ˜ã °ã 'ï¼‰ã ¨ã ¯ã€ ã,³ãƒ³ãƒ"ãƒ¥ãƒ¼ã,¿ã §æ–‡å—ã,'è¡¨ç¤ºã ™ã,‹éš›ã «ã€ æ£ã —ã è¡¨ç¤ºã •ã,Œã ªã „ç ¾è±¡ã ®ã "ã ¨ã€‚

- ä¾‹ï¼šã€Œæ–‡å—åŒ–ã 'ã€ ã ŒãŒã€ ã€Œ Ã¦â€"â€¡Ã¥Ââ€"Ã¥Å'â€"Ã£Â â€˜ã€ ã,,ã€Œè ï¿½ï¿½æ€œå–§ç¸ºï¿½ã€ ã ¨è¡¨ç¤ºã •ã,Œã,‹ã ªã ©ã€‚

ã€Œæ–‡å—åŒ–ã 'ã€ ã ¨ã „ã †è¨€è'‰ã ¯ã€ ã,³ãƒ³ãƒ"ãƒ¥ãƒ¼ã,¿ç'°å¢ƒã §åŽŸå‰‡ã ¨ã —ã ¦ãƒžãƒ«ãƒ ãƒ ã,¤ãƒˆæ–‡å—ã,'ä½¿ç"¨ã —ã ªã „æ¬§ç±³ç‰ã ®ãƒ©ãƒ†ãƒ³ã,¢ãƒ«ãƒ•ã,¡ãƒ™ãƒƒãƒˆä½¿ç"¨è€…ã «ã Šã „ã ¦ã ¯è©²å½“ã ™ã,‹ç"¨èªžã ŒåˆåœØã —ã ªã ‹ã £ã Ÿã "ã ¨ã ‹ã,‰ã€ æ—¥æœ¬èªž ã ®

â€œ**Mojibake**â€ ã ¨ã „ã †è¨€è'‰ã Œã ã ®ã ¾ã ¾é€šç"¨ã ™ã,‹ã,ˆã †ã «ã ªã £ã Ÿã€‚â†'#Mojibake

ç›®æ¬¡ [é žè¡¨ç¤º]

UTF-8 encoded Japanese Wikipedia rendered in cp1252

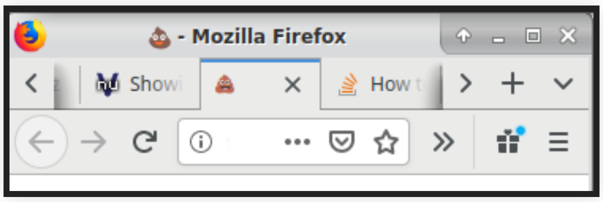
UTF-8 FALLBACK EXAMPLE

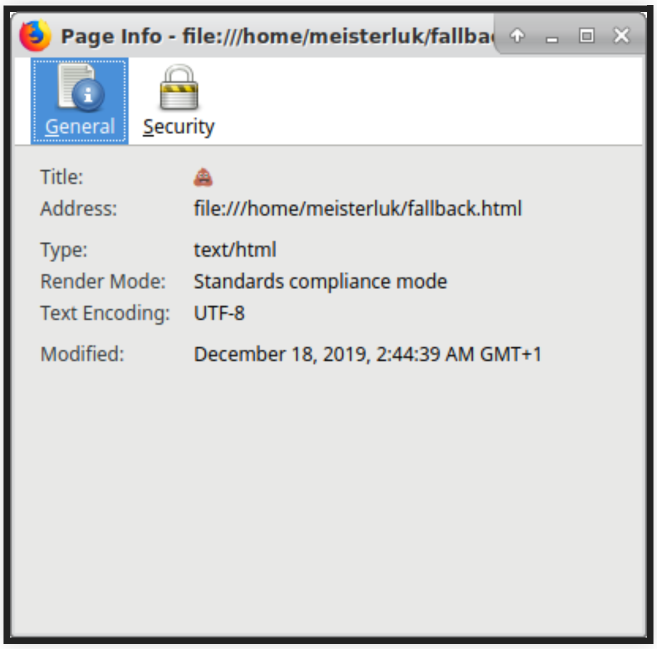
```rust
use std::fs::File;
use std::io::prelude::*;

fn main() -> std::io::Result<()> {
    let mut fd = File::create("pile_of_poo.html")?;
    fd.write(b"<!DOCTYPE html>\n<head><title>\
\xf0\x9f\x92\xa9</title>\n")?;
    Ok(())
}
```

UTF-8 FALLBACK EXAMPLE

```
use std::fs::File;
use std::io::prelude::*;

fn main() -> std::io::Result<()> {
    let mut fd = File::create("mojibake.html")?;
    fd.write(b"<!DOCTYPE html>\n<head><title>\xda \
\xf0\x9f\x92\xa9</title>\n")?;
    Ok(())
}
```

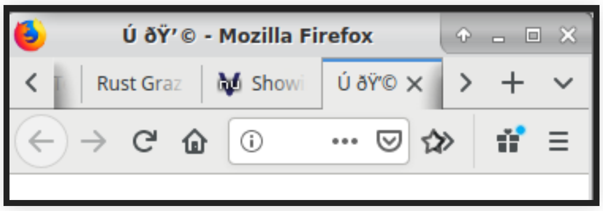
UTF-8 FALLBACK EXAMPLE

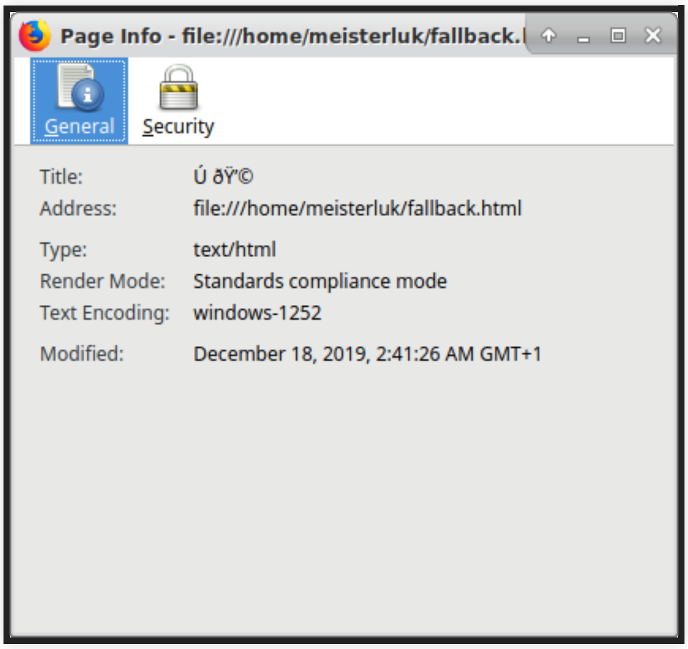




Title:	💩
Address:	file:///home/meisterluk/fallback.html
Type:	text/html
Render Mode:	Standards compliance mode
Text Encoding:	UTF-8
Modified:	December 18, 2019, 2:44:39 AM GMT+1

UTF-8 FALLBACK EXAMPLE





Title: Ú ðŸ'©
Address: file:///home/meisterluk/fallback.html
Type: text/html
Render Mode: Standards compliance mode
Text Encoding: windows-1252
Modified: December 18, 2019, 2:41:26 AM GMT+1

CPE

version	ASCII compat	fallback	prefix-free	self-sync	sort
5	✗	?	✓	✗	✓
4	✓	?	✗	✗	✗
3	✓	?	✓	✗	✗
2	✗	?	✓	✗	✓
1	✓	?	✓	✓	✗

UNICODE / UTF-8 TERMINOLOGY

Mojibake

Character rendered in wrong encoding

Han unification

Korean and Japanese writing systems are based on Chinese characters ⇒ huge overlap ⇒ merge different writing systems

Overlong encoding

Remove leading zeros in your binary string. Then cram those bits into 1-4 UTF-8 bytes; as *few* as needed! If you take more bytes, you have overlong encoding; which is disallowed.

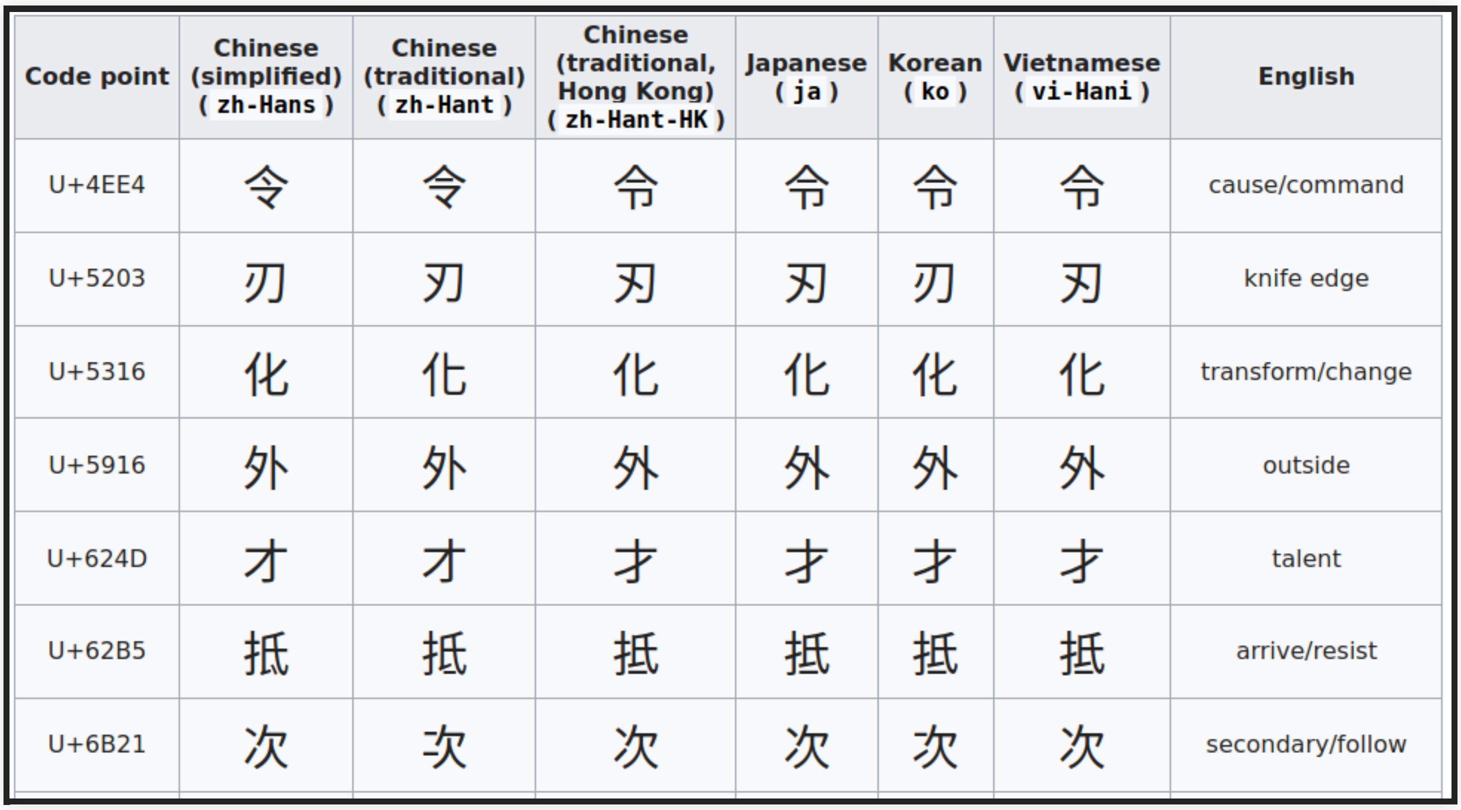
UTF-8

0xxx xxxx				2^{7}
110x xxxx	10xx xxxx			2^{5+6}
1110 xxxx	10xx xxxx	10xx xxxx		$2^{4+6\times2}$
1111 0xxx	10xx xxxx	10xx xxxx	10xx xxxx	$2^{3+6\times3}$

UNICODE: HAN UNIFICATION

One possible rationale is the desire to limit the size of the full Unicode character set, where CJK characters as represented by discrete ideograms may approach or exceed 100,000 characters. Version 1 of Unicode was designed to fit into 16 bits and only 20,940 characters (32%) out of the possible 65,536 were reserved for these CJK Unified Ideographs.

UNICODE: HAN UNIFICATION



Code point	Chinese (simplified) (zh-Hans)	Chinese (traditional) (zh-Hant)	Chinese (traditional, Hong Kong) (zh-Hant-HK)	Japanese (ja)	Korean (ko)	Vietnamese (vi-Hani)	English
U+4EE4	令	令	令	令	令	令	cause/command
U+5203	刃	刃	刃	刃	刃	刃	knife edge
U+5316	化	化	化	化	化	化	transform/change
U+5916	外	外	外	外	外	外	outside
U+624D	才	才	才	才	才	才	talent
U+62B5	抵	抵	抵	抵	抵	抵	arrive/resist
U+6B21	次	次	次	次	次	次	secondary/follow

UNICODE: HAN UNIFICATION

TRON Code is a multi-byte character encoding used in the TRON project. It is similar to Unicode but does not use Unicode's Han unification process: each character from each CJK character set is encoded separately, including archaic and historical equivalents of modern characters

via Wikipedia: TRON encoding

UNICODE / UTF-8 TERMINOLOGY

Surrogates

In UTF16: Extension to encode 2.5 bytes in a 2 bytes fixed-width encoding.
In UTF8: Invalid bit patterns for compatibility with UTF16.

Basic Multilingual Plane

Plane of common use characters (65,536 code points)

UNICODE: SURROGATES

UTF16: 2 bytes encoding

2.5 bytes char

$\boxed{\text{   aaaa}}\ \boxed{\text{aaaa aabb}}\ \boxed{\text{bbbb bbbb}}$

$\Rightarrow$ high surrogate $\boxed{\text{1101 10aa}}\ \boxed{\text{aaaa aaaa}}$

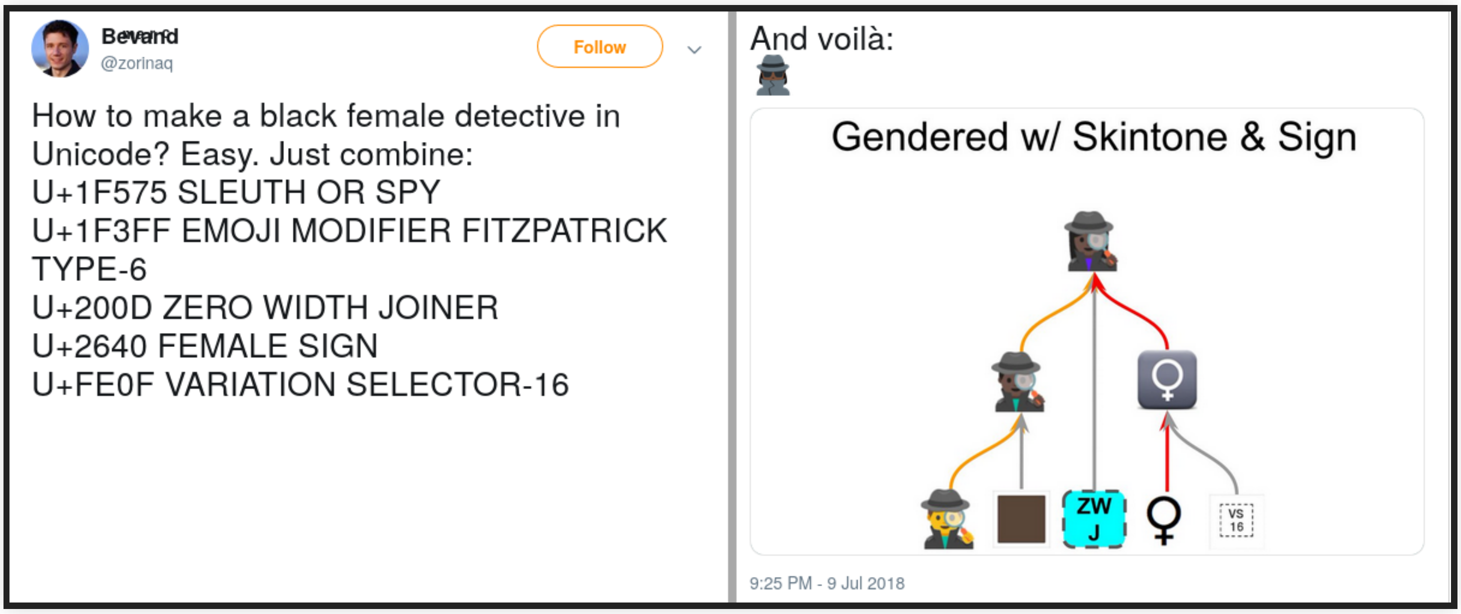
low surrogate $\boxed{\text{1101 11bb}}\ \boxed{\text{bbbb bbbb}}$

$\Rightarrow$ UTF16 encoding

$\boxed{\text{1101 10aa}}\ \boxed{\text{aaaa aaaa}}\ \boxed{\text{1101 11bb}}\ \boxed{\text{bbbb bbbb}}$

compare with Wikipedia

UNICODE



Bevand
@zorinaq

Follow

How to make a black female detective in Unicode? Easy. Just combine:
U+1F575 SLEUTH OR SPY
U+1F3FF EMOJI MODIFIER FITZPATRICK TYPE-6
U+200D ZERO WIDTH JOINER
U+2640 FEMALE SIGN
U+FE0F VARIATION SELECTOR-16

And voilà:
🕵🏿‍♀️

9:25 PM - 9 Jul 2018

via twitter

UNICODE

1. shapecatcher.com
2. joelonsoftware.com: The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets

UTF-8 IN RUST

```rust
const TEXT: &str = "Héllö Wørl̶d";

fn main() {
    println!("{}", TEXT);
}
```

String

```rust
fn main() {
    let s = String::from("Hello Graz");
    println!("{}", s);
}
```

String

```rust
fn main() {
    // let s = String::with_capacity(11);
    let s = String::from("Hello Graz");
    s += "!";  // also:  s.push_str("!");
    println!("{}", s);
}
```

Does it compile?

`String`

```rust
fn main() {
    // let s = String::with_capacity(11);
    let s = String::from("Hello Graz");
    s += "!";  // also:  s.push_str("!");
    println!("{}", s);
}
```

Does it compile? No, it's immutable.

String

```rust
fn main() {
    let mut s = String::from("Hello Graz");
    s += "!";
    println!("{}", s);
}
```

Does it compile?

`String`

```rust
fn main() {
    let mut s = String::from("Hello Graz");
    s += "!";
    println!("{}", s);
}
```

Does it compile? Yes.

`String`

- data must be valid UTF-8 string
- owns its data (dropping `String` means deallocate data)
- ⇒ “owned string”
- does not implement Copy, thus move semantics apply
- consists of {&data, length, capacity}

String

```
// \xD8 is a surrogate code point
fn main() {
    let s = String::from("Hello\xD8Graz!");
    println!("{}", s);
}
```

String

```
error: this form of character escape may only
       be used with characters in the range [\x00-\x7f]
 --> src/main.rs:3:32
  |
2 |     let s = String::from("Hello\xD8Graz!");
  |                                ^^^^
```

`String`

```rust
fn sub(arg: String) {
    println!("{}", arg);
}

fn main() {
    let s = String::from("Hello Graz");
    sub(s);
    println!("{}", s);
}
```

Does it compile?

String

```
error[E0382]: borrow of moved value: `s`
 --> src/main.rs:8:20
  |
6 |     let s = String::from("Hello Graz");
  |         - move occurs because `s` has type
  |           `std::string::String`, which does
  |           not implement the `Copy` trait
7 |     sub(s);
  |         - value moved here
8 |     println!("{}", s);
  |                    ^ value borrowed here after move
```

`&str`

- data must be valid UTF-8 string
- stored in .data/.rodata, does not deallocate
- ⇒ “borrowed string”, lives as long as the program
- consists of {&data, length}

`&str`: SYNTAX

```rust
fn main() {
    let s = "H\x65\u{6C}lo \
             Graz";
    println!("{}", s);
}
```

&str: SYNTAX

```rust
fn main() {
    println!("\u{1F4A9}");
}
```

```
💩
```

&str

```rust
fn main() {
    let s: &str = "Hello Graz";
    s += "!";
    println!("{}", s);
}
```

&str

```
error[E0368]: binary assignment operation
    `+=` cannot be applied to type `&str`
 --> src/main.rs:3:5
  |
3 |     s += "!";
  |     -^^^^^^^^
  |     |
  |     cannot use `+=` on type `&str`
```

&str

```rust
fn sub(arg: &str) {
    println!("{}", arg);
}

fn main() {
    let s: &str = "Hello Graz!";
    sub(s);
    println!("{}", s);
}
```

Does it compile?

&str

```
fn sub(arg: &str) {
    println!("{}", arg);
}

fn main() {
    let s: &str = "Hello Graz!";
    sub(s);
    println!("{}", s);
}
```

Does it compile? Yes.

&str

```rust
fn main() {
    let s = "Hello Graz!";
    let b = "Hello Graz!";
    println!("{:p} {:p}", s, b);
    // 0x557ab7f09cc0 0x557ab7f09cc0
}
```

MEMORY SIZE

```rust
use std::mem;

fn main() {
    println!("{}", mem::size_of::<A>());
}
```

where A is `u8` (1), `u32` (4), `f64` (8), `&u8` (8), `String` (24), `&str` (16), `Vec<u8>` (24) or `&[char]` (16).

OTHER TYPES

- `Vec<u8>` can contain an arbitrary non-UTF-8 string
- `char` is always 4 bytes and thus can contain any UTF-8 code point
- `[u8]` is a slice of `u8`. Cumbersome to handle
- `OsString`, ffi::CString … if you need compatibility strings.

STRING OPERATIONS

```rust
fn main() {
    println!("{}", "ß".to_uppercase());
}
```

Output:

Compare with Unicode casemap F.A.Q.

STRING OPERATIONS

```
fn main() {
    println!("{}", "ß".to_uppercase());
}
```

Output: SS

Compare with Unicode casemap F.A.Q.

STRING'S LIFETIME

```rust
fn main() {
    let s = {
        let alt: &str = "Graz";
        alt
    };
    println!("{}", s);
}
```

Does it compile?

STRING'S LIFETIME

```rust
fn main() {
    let s = {
        let alt: &str = "Graz";
        alt
    };
    println!("{}", s);
}
```

Does it compile? Yes.

STRING'S LIFETIME

```rust
fn main() {
    let s = {
        let alt: String = "Graz".to_string();
        alt
    };
    println!("{}", s);
}
```

Does it compile?

STRING'S LIFETIME

```rust
fn main() {
    let s = {
        let alt: String = "Graz".to_string();
        alt
    };
    println!("{}", s);
}
```

Does it compile? Yes.

STRING'S LIFETIME

```rust
fn main() {
    let s = {
        let alt: String = "Graz".to_string();
        alt.as_str()
    };
    println!("{}", s);
}
```

Does it compile?

STRING'S LIFETIME

```
error[E0597]: `alt` does not live long enough
 --> src/main.rs:4:9
  |
2 |     let s = {
  |         - borrow later stored here
3 |         let alt: String = "Graz".to_string();
4 |         alt.as_str()
  |         ^^^ borrowed value does not live long enough
5 |     };
  |     - `alt` dropped here while still borrowed
```

`Deref` TRAIT MAGIC

```rust
fn takes_str(s: &str) {}

let s = String::from("Hello");

takes_str(&s);
```

via std::string::String

INDEXING

```
fn main() {
    let mut s = "合気道";
    println!("{}", s[1]);
}
```

INDEXING

```
error[E0277]: the type `str` cannot be indexed by `{integer}`
 --> src/main.rs:3:20
  |
3 |     println!("{}", s[1]);
  |                    ^^^^
  | string indices are ranges of `usize`
```

INDEXING

```rust
fn main() {
    let mut s = "合気道".chars();
    println!("{}", s.nth(1).unwrap()); // 気
}
```

INDEXING

```rust
fn main() {
    for s in "देवनागरी".chars() {
        println!("{}", s);
    }
}
```

```
द
◌े
व
न
◌ा
ग
र
◌ी
```

UNICODE

Interesting read: Stackoverflow “Why does modern Perl avoid UTF-8 by default?”

EPILOGUE

QUIZ

ASCII is a __-bit encoding

Maximum number of bytes of a UTF-8 code point?

Which string types does rust define?

`std::mem::size_of::<char>()` **gives?**

How to iterate over characters of a string?

QUIZ

ASCII is a __-bit encoding

ASCII is a 7-bit encoding

Maximum number of bytes of a UTF-8 code point?

Which string types does rust define?

`std::mem::size_of::<char>()` gives?

How to iterate over characters of a string?

QUIZ

ASCII is a __-bit encoding

ASCII is a 7-bit encoding

Maximum number of bytes of a UTF-8 code point?

4

Which string types does rust define?

`std::mem::size_of::<char>()` gives?

How to iterate over characters of a string?

QUIZ

ASCII is a __-bit encoding

ASCII is a 7-bit encoding

Maximum number of bytes of a UTF-8 code point?

4

Which string types does rust define?

`&str`, `String`

`std::mem::size_of::<char>()` gives?

How to iterate over characters of a string?

QUIZ

ASCII is a __-bit encoding

ASCII is a 7-bit encoding

Maximum number of bytes of a UTF-8 code point?

4

Which string types does rust define?

`&str`, `String`

`std::mem::size_of::<char>()` gives?

4

How to iterate over characters of a string?

QUIZ

ASCII is a __-bit encoding

ASCII is a 7-bit encoding

Maximum number of bytes of a UTF-8 code point?

4

Which string types does rust define?

`&str`, `String`

`std::mem::size_of::<char>()` gives?

4

How to iterate over characters of a string?

```
let mut s = "Hello world".chars();
```

NEXT SESSION

Wed, 2019/01/29 19:00

Topic: traits

THANKS!