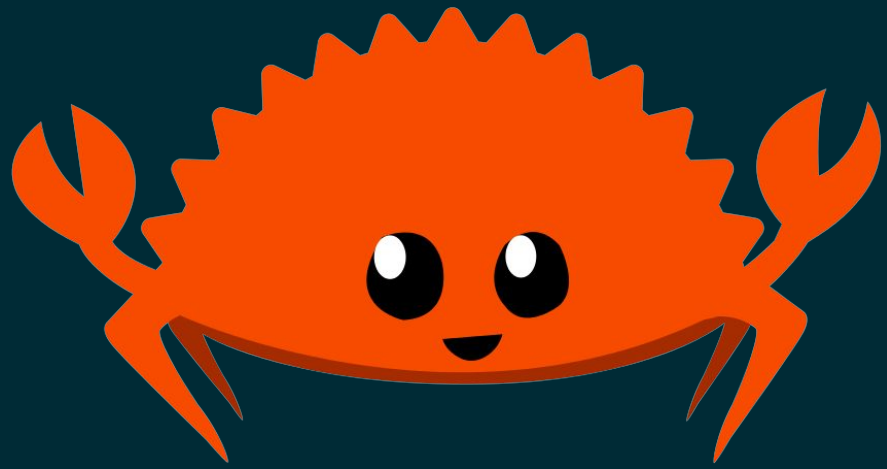
Portar una biblioteca de C a Rust:

El caso de librsvg

Federico Mena Quintero
federico@gnome.org



¿Qué usa librsvg?

GNOME, la plataforma y el escritorio:

gtk+ de forma indirecta via gdk-pixbuf
thumbnailer via gdk-pixbuf
eog
gstreamer-plugins-bad

¿Qué usa librsvg?

Aplicaciones:

gnome-games (gnome-chess, five-or-more, etc.)
gimp
gcompris
claws-mail
darktable

¿Qué usa librsvg?

Ambientes de escritorio:

mate-panel
Evas / Enlightenment
emacs-x11

¿Qué usa librsvg?

Cosas que no te esperabas

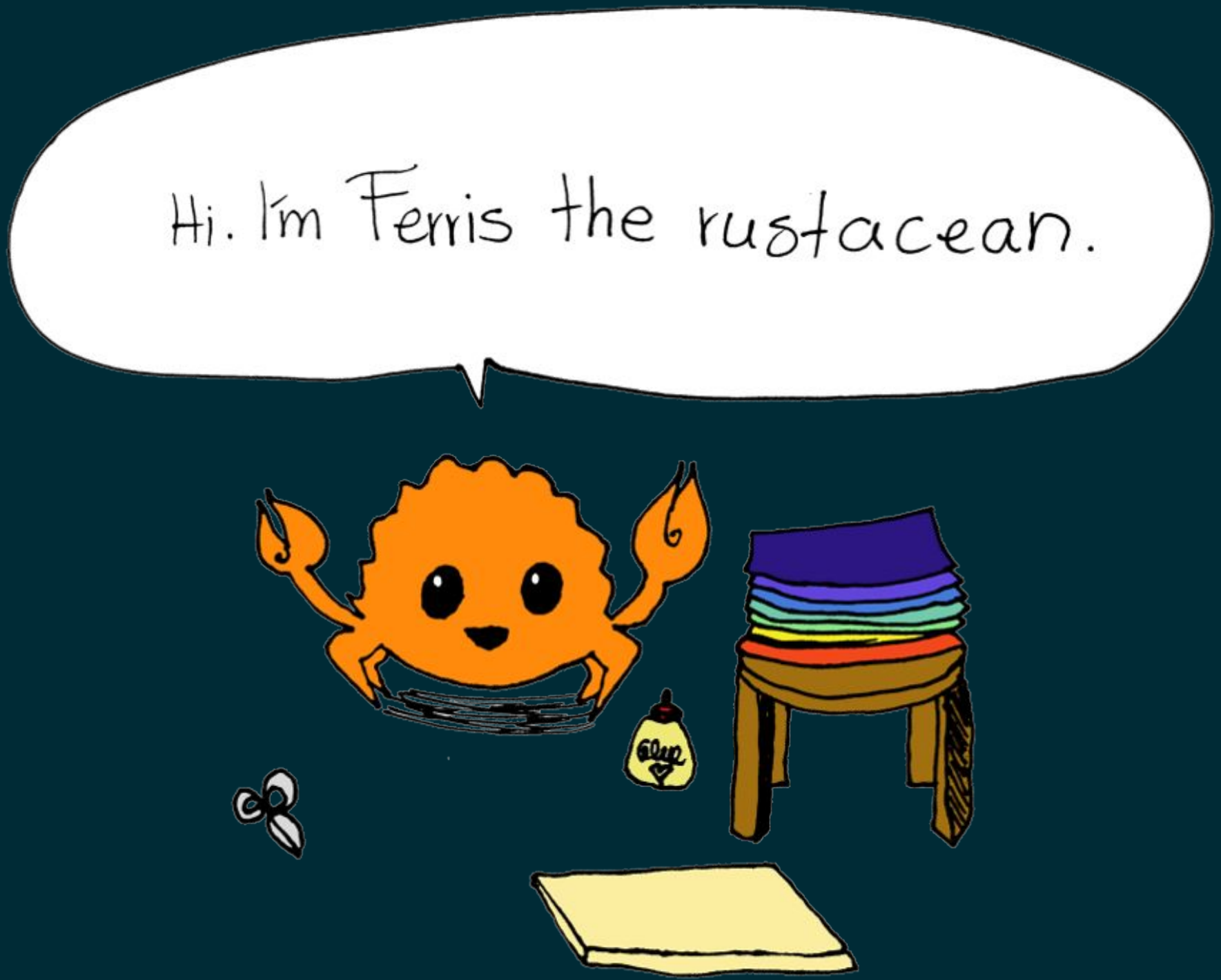
ImageMagick
Wikipedia ← *han sido fantásticos*

Una larga historia

- Primer commit, Eazel, 2001
- Experimento para usar un parser SAX en vez de leer todo el DOM en un solo paso.
- Renderizaba con libart
- Gill / librsvg → Sodipodi → Inkscape
- Librsvg se escribió mientras se escribía la especificación de SVG
- Se portó a Cairo eventualmente

Federico toma las riendas

- Librsvg no tenía maintainer en 2015
- Yo lo tomé en febrero de 2015
- Comienzo el port a Rust en octubre de 2016


Hi. I'm Ferris the rustacean.
Glup

La decisión de Rustificar

- Macros en glib para aritmética sin desborde de Allison Lortie
-
- **"Rust out your C"**
  Carol Nichols
  https://github.com/carols10cents/rust-out-your-c-talk
-
- **"Writing GStreamer elements in Rust"**
  Sebastian Dröge
  https://coaxion.net/blog/2016/05/writing-gstreamer-plugins-and-elements-in-rust/
-

Integrar los sistemas de compilación

- Librsvg usa autotools
- Rust usa cargo
- Compilar librsvg_internals.a
- En Cargo.toml:

```
[lib]
name = "rsvg_internals"
crate-type = ["staticlib"]
```

```
librsvg/
  configure.ac
  Makefile.am
  *.[ch]

  rust/
    Cargo.toml
    src/
      *.rs
    target/
```

Agradecimientos de Autotools

- Hubert Figuière
  - https://www.figuiere.net/hub/blog/?2016/10/07/862-rust-and-automake
- Luke Nukem
  - http://lukenukem.co.nz/gsoc/2017/05/17/gso_2.html
- Havoc Pennington
  - https://blog.ometer.com/2017/01/10/dear-package-managers-dependency-resolution-results-should-be-in-version-control/

Propagación de errores

- Librsvg “manejaba” errores devolviendo valores por defecto, o basura.

```
<rect x=”” y=”5” width=”hola”.../>
<path fill=”#wxyz”/>
```

- La especificación de SVG a veces dice qué hacer
  - No es consistente consigo misma
  - Las implementaciones difieren de la especificación
  - ¯_(ツ)_/¯


Glue
Glue

Expose stuff from Rust to C

- Parallel structs with #[repr(C)]
- Expose opaque pointers to structs, and functions to frob them
- Get the memory management right
- new() / destroy() - https://people.gnome.org/~federico/news-2016-11.html#14
- Reference counting - https://people.gnome.org/~federico/news-2017-02.html#17
-

Parsers

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Un parser malo para #rrggbb

```
guint32
rsvg_css_parse_color (const char *str, ...)
{
    gint val = 0;

    if (str[0] == '#') {
        int i;
        for (i = 1; str[i]; i++) {
            int hexval;
            if (str[i] >= '0' && str[i] <= '9')
                hexval = str[i] - '0';
            else if (str[i] >= 'A' && str[i] <= 'F')
                hexval = str[i] - 'A' + 10;
            else if (str[i] >= 'a' && str[i] <= 'f')
                hexval = str[i] - 'a' + 10;
            else
                break;
            val = (val << 4) + hexval;
        }
        /* handle #rgb case */
        if (i == 4) {
            val = ((val & 0xf00) << 8) | ((val & 0x0f0) << 4) |
                    (val & 0x00f);
            val |= val << 4;
        }

        val |= 0xff000000; /* opaque */
```

Parsear un dígito hex en Rust

```rust
fn from_hex(c: u8) -> Result<u8, ()> {
    match c {
        b'0' ... b'9' => Ok(c - b'0'),
        b'a' ... b'f' => Ok(c - b'a' + 10),
        b'A' ... b'F' => Ok(c - b'A' + 10),
        _ => Err(())
    }
}
```

Parsear un dígito hex en Rust

```
fn from_hex(c: u8) -> Result<u8, ()> {
    match c {
        b'0' ... b'9' => Ok(c - b'0'),
        b'a' ... b'f' => Ok(c - b'a' + 10),
        b'A' ... b'F' => Ok(c - b'A' + 10),
        _ => Err(())
    }
}
```

Parsear un color hex en Rust

```rust
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

Parsear un color hex en Rust

```
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

Parsear un color hex en Rust

```
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

Parsear un color hex en Rust

```
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

Parsear un color hex en Rust

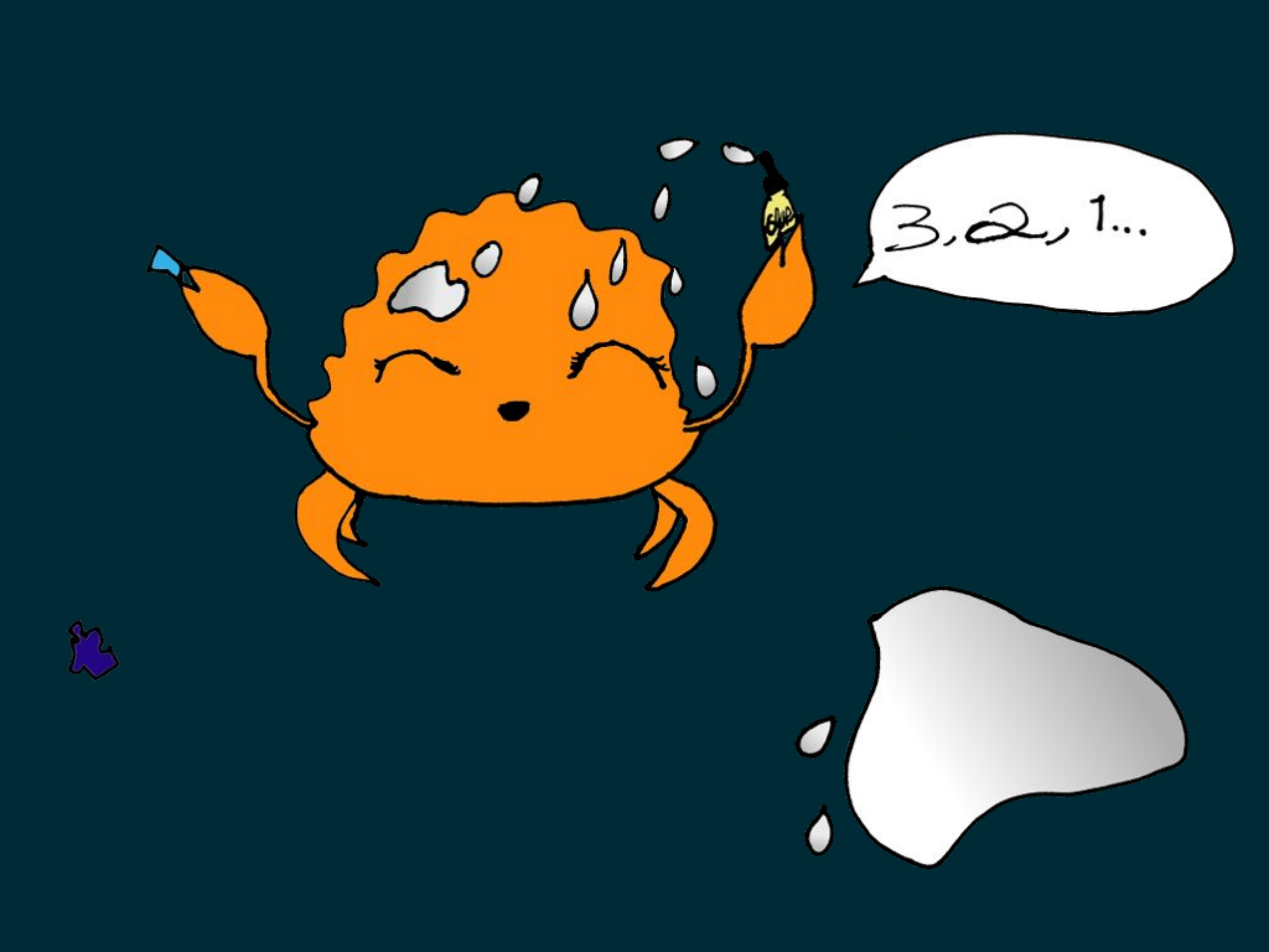
```
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

Parsear un color hex en Rust

```
impl Color {
    pub fn parse_hash(value: &[u8]) -> Result<Self, ()> {
        match value.len() {
            8 => Ok(rgba(
                from_hex(value[0])? * 16 + from_hex(value[1])?,
                from_hex(value[2])? * 16 + from_hex(value[3])?,
                from_hex(value[4])? * 16 + from_hex(value[5])?,
                from_hex(value[6])? * 16 + from_hex(value[7])?),
            ),

            6 => Ok(rgb(...)),

            4 => Ok(rgb(...)),

            3 => Ok(rgb(...)),

            _ => Err(())
        }
    }
}
```

No son programadores malos; es que C es un lenguaje hostil

- Le puse “git blame” para ver quién había modificado esos parsers.
- Tres o cuatro de los mejores hackers de GNOME que hemos tenido.
- Sé que no hubieran escrito *eso* desde cero.
- C hace difícil el manejo de errores, o cambiar las cosas después.


3, 2, 1...

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...


};
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...


};

cairo_fill_rule_t rsvg_state_get_fill_rule     (RsvgState *state);

gboolean          rsvg_state_get_has_fill_rule (RsvgState *state);
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...


};
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {

}

#[no_mangle]
pub extern fn state_rust_new()
    -> *mut StateRust;

#[no_mangle]
pub extern fn state_rust_free(
    state: *mut StateRust
);
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {

}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t fill_rule;
    gboolean has_fill_rule;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    cairo_fill_rule_t clip_rule;
    gboolean has_clip_rule;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    RsvgPaintServer *stroke;
    gboolean has_stroke_server;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    guint8 stroke_opacity;
    gboolean has_stroke_opacity;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    RsvgLength stroke_width;
    gboolean has_stroke_width;

    double miter_limit;
    gboolean has_miter_limit;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

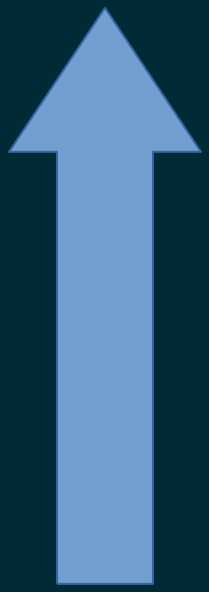
```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```



```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    cairo_matrix_t affine;

    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct RsvgState {
    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
cairo_matrix_t rsvg_state_get_affine(RsvgState *state)
{
    return state_rust_get_affine(state->state_rust);
}
```

```
struct RsvgState {
    ...

    StateRust *state_rust;
};
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
cairo_matrix_t rsvg_state_get_affine(RsvgState *state)
{
    return state_rust_get_affine(state->state_rust);
}
```

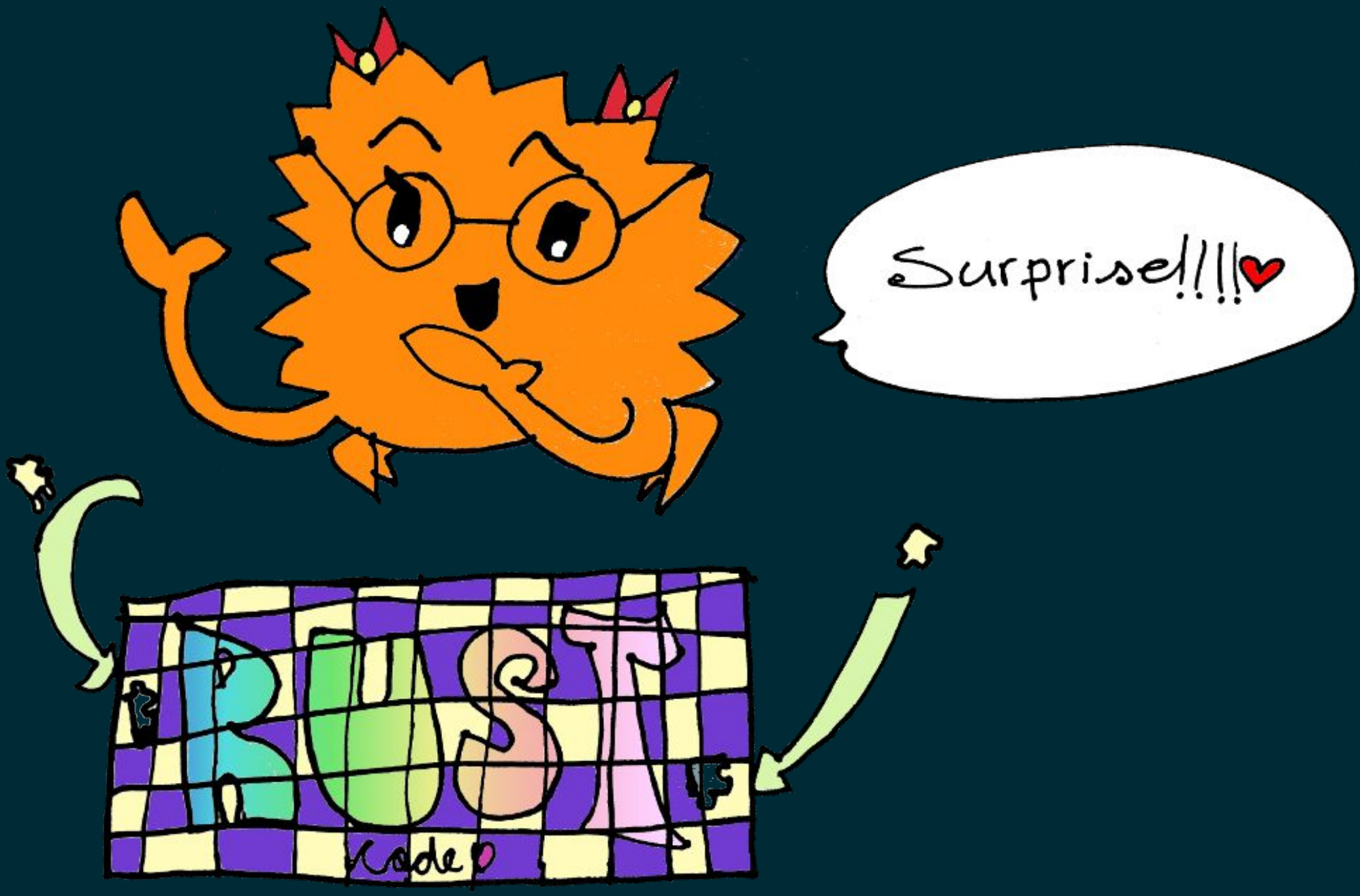
```
#[no_mangle]
pub extern fn state_rust_get_affine(s: *const StateRust) -> cairo::Matrix
{
    let state = unsafe { &*s };

    state.affine
}
```

```
struct RsvgState {

};
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```

```
struct StateRust {
    affine: cairo::Matrix;

    fill_rule: Option<FillRule>,
    clip_rule: Option<FillRule>,
    stroke: Option<PaintServer>,
    stroke_opacity: Option<u8>,
    stroke_width: Option<RsvgLength>,
    miter_limit: Option<f64>,
}
```


Surprise!!!!
RUST
code

Cuando Rust te aclara la mente

```
if (!cairo_matrix_invert (...))
    return;
```

- Las matrices degeneradas se ignoraban.
- En realidad sí eran bugs:
  - Memoria no inicializada
  - Floats inválidos con todos los bits en cero
  - SVGs inválidos – y sin validación de datos
- Cairo-rs hacía panic!() con matrices degeneradas
  - Le puse try_invert(); ahora todo se valida.

Tu salud mental

- Rust no es fácil de aprender
- Conceptos nuevos (pertenencia, paradigmas funcionales, mucha sintaxis, etc.)
- Etapa de aprendizaje: “pelearse con el borrow checker”
- C es un océano de referencias compartidas; ***Rust No Hace Eso™***

Tu salud mental

- Al principio no sabrás cómo portar cosas, o no sabrás los mejores patrones que se pueden usar.
- Está bien empezar con código feo; se puede refactorizar después. ¡Rust se refactoriza lindísimo!
- ¡Las pruebas son súper importantes!
- Aprende el manejo de errores: Result<Ok, Err>

- La comunidad de Rust es FANTÁSTICA.

Gracias

- Luciana Mena-Silva (mi hija) por los dibujos de Ferris
- Katrina Owen por la técnica de narrar código en vivo - http://www.kytrinyx.com/talks/
- Alberto Ruiz, Joaquín Rosales por el hackfest de GNOME+Rust
- irc.mozilla.org #rust-beginners
- #nom
- Sebastian Dröge por responder mis dudas
- Paolo Borelli por portar RsvgState

Blog posts

- https://people.gnome.org/~federico/blog/tag/librsvg.html
- https://people.gnome.org/~federico/blog/tag/rust.html