How to print European train timetables on a (very long) receipt

A tale of "surely everything is consistent about *this*, right?"

> Romane, she/her

⚠️ Disclaimer

- this will mostly be about GTFS, but there are other formats out there for specific uses
- the way i'm using the data is by no means efficient (was a first transit and Go project, not best practices)
- this talk will not be about real time data

⚡ Initial Problem

- Often spend time near railways or in trains
- *wow, there's traffic! Passenger traffic at that as well!*
- Trains are fun!™
    - **Can I see more? 👈 👉**

⚡ Initial Problem

- **Why not use a transit app?**
  - Whichever: `bahnhof.de`, `bahn.expert`, `sncf-connect.fr`, ...
  - Pros: Often lots of infos about the trains and/or realtime info
  - Cons: Not great if not precisely at a station of if traffic doesn't stop
- Great for info about a trip, not great for knowing *what* trips there are

🇪🇺 EU transit data regulations

- Let's get the data ourselves, shall we?
- Introducing: *Commission Delegated Regulation (EU) 2017/1926 of 31 May 2017 supplementing Directive 2010/40/EU of the European Parliament and of the Council with regard to the provision of EU-wide multimodal travel information services*
- *huh? the what?* 🤔

📖 TLDR: EU transit data regulations

- (lots of stuff not relevant for us)
- Transit authorities in each country MUST publish in one of the following formats to the country's national access point (Art. 4):
    - ✅ Network Timetable Echange (NeTEx) CEN/TS 16614
    - ❌ technical documents defined in Regulation (EU) No 454/2011 and subsequent versions
    - ❌ technical documents elaborated by IATA
    - ❌ any machine-readable format fully compatible and interoperable with those standards and technical specifications;

NeTEx

- **"Network Timetable Exchange"**
- European standardization effort to represent *all* aspects of public transit (network, timetable, fares, ...)
- very long spec, inspired by other EU specifications
- "we need to make sure interoperable formats are used" ***proceeds to put the specification behind a massive paywall***

🤔 "but this sounds really cursed!"

- that's because *it is*
- huge XML file with only an open validation file
- most publishing countries just use a different subset of it ("profiles")
- not quite ready enough to sift through this quite yet
- **isn't there anything simpler?**

GTFS vs NeTEx: Why? Huh? Who? What?

- Turns out: GTFS might be a tiny bit better for our purposes and *can* be generated from *some* NeTEx profiles

Format	GTFS	NeTEx
Created in	2006	2014
Standardized by	Google	EU
Usage	Worldwide	Europe
Timetables?	✅	✅
Open spec?	✅	❌

Getting GTFS data

- Legally available through the National Access Points in the EU
- Aggregators like the Mobility Database have a lot of feeds too
- https://gtfs.de provides compressed/optimized GTFS from the nationwide german DELFI data feeds (collection of local transit data)
- a lot of operators also publish their feeds as GTFS
- let's open a feed then!

CSVs? in *my* transit data? (always has been)

- yes, as `.txt` files too!
- a few ***Required*** files (aite sure)
- a few ***Optional*** files (hmm, okay)
- a few ***Conditionally Required*** files (edge cases???)
- a few ***Conditionally Forbidden*** files (i'm gonna commit crimes)

⚙ How it works

- Basically a standard DB schema, except it's written as a CSV
- TLDR (*veeery* loose approximation here):
  - `routes.txt`, `trips.txt` = what kinds of services are there?
  - `calendar.txt`, `calendar_dates.txt` = when do services run?
  - `stops.txt`, `stop_times.txt` = where/when do services stop?
- philosophy behind the spec: ***as easy to write as possible***

Minimal working feed contents (required data)

- `agency.txt`: operator/provider info (name, URL, timezone)
- `routes.txt`: transit line details (ID, name, route type)
- `trips.txt`: simple trip (ID, route ID, service ID)
- `calendar.txt` *or* `calendar_dates.txt`: defining what dates the service IDs run on
- `stop_times.txt`: trip calls (trip ID, sequence number, stop ID, arrival time)
- `stops.txt`: stops (ID, name, latitude/longitude)

But wait, there's more!

- the spec provides a lot of fields that can be filled
- multiple transit agencies, `feed_info`, `transfer`s
- `stop_time` departure times
- `route` names, descriptions, and colors
- `trip` short names, headsigns
- detailed station layouts (`stops`, `pathways`, `levels`)
- wheelchair and bike accessibility
- transit `shape`s, fares, continuous pickup, translations, `frequencies`, ...
- **not all fields are relevant to us**

Actually using the data

- Pre-process trips to get a geographic bounding box
- Get all trips *possibly* going by that point
- Refine using stop times to get trips that actually go through/near the wanted points
- **easy, right?**

Actually using the data

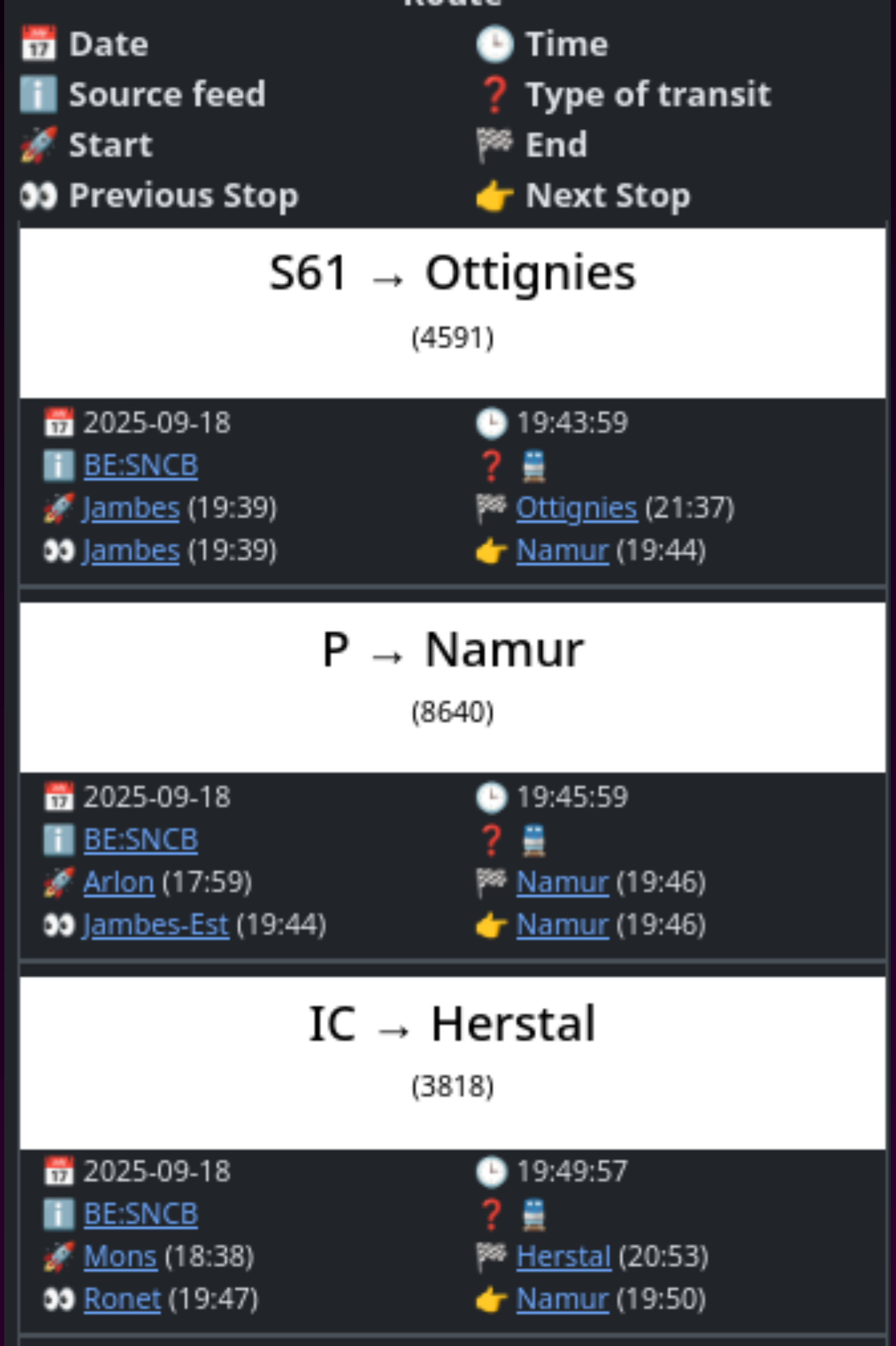
- wait, how *do* we filter trips?
- no perfect metric out there, results will vary
- chosen method: checking angles
- Include if: angle difference between `obsPoint.bearingTo(prevStop)` and `obsPoint.bearingTo(nextStop)` >= threshold
- Also include a constant radius around stop points
- works well enough for ***most*** cases

Making a sights API

- Fetch and parse feeds into a DB
- Basic main endpoint: (lat, lon) $\rightarrow$ list of sights
- Sight: source feed, passing time, trip details (short name, stop times)

It works!

(we can see *something*)



📅 Date	🕒 Time
ℹ️ Source feed	❓ Type of transit
🚀 Start	🏁 End
👀 Previous Stop	👉 Next Stop

S61 → Ottignies
(4591)

📅 2025-09-18	🕒 19:43:59
ℹ️ BE:SNCB	❓ 🚆
🚀 Jambes (19:39)	🏁 Ottignies (21:37)
👀 Jambes (19:39)	👉 Namur (19:44)

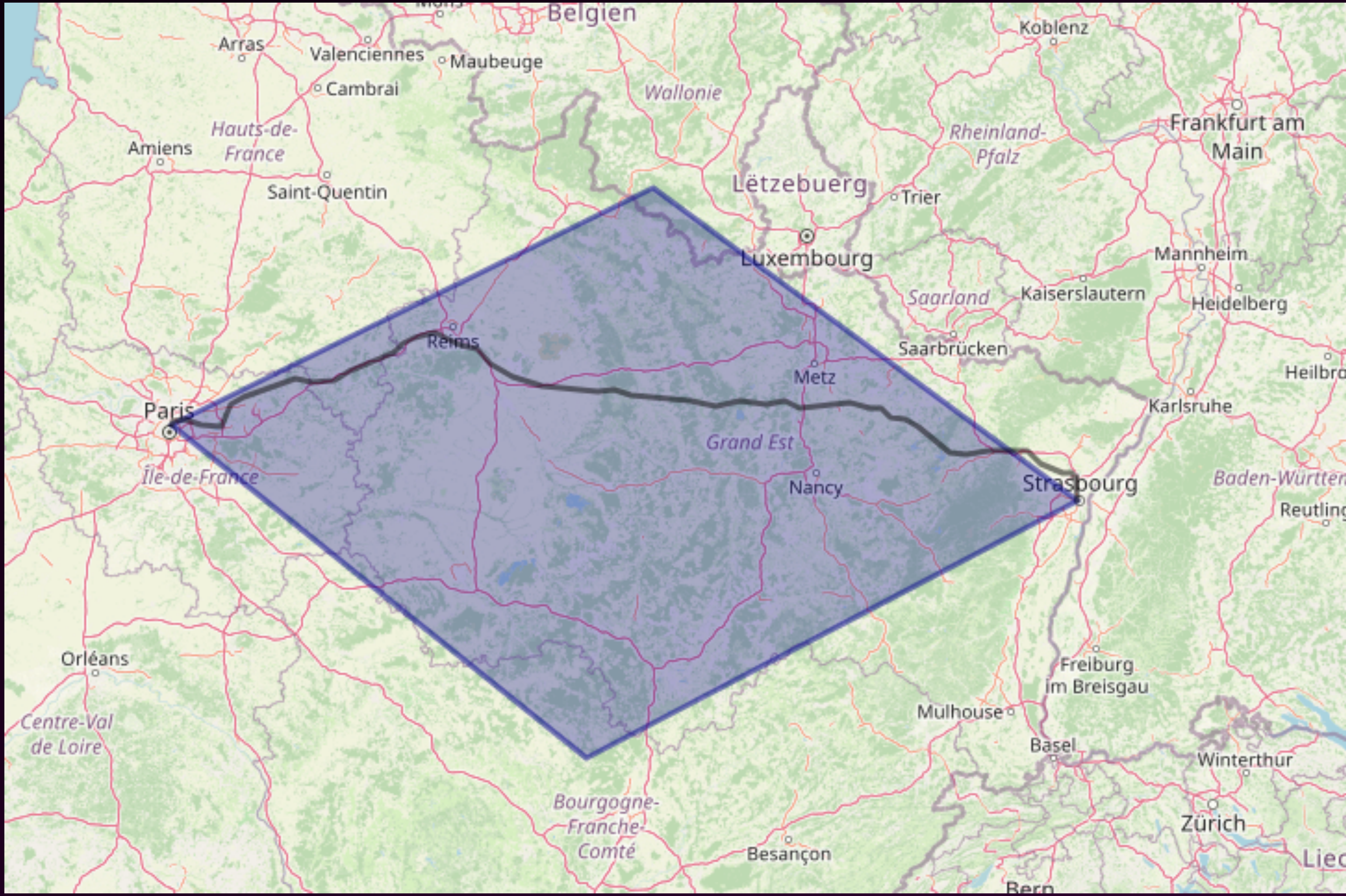
P → Namur
(8640)

📅 2025-09-18	🕒 19:45:59
ℹ️ BE:SNCB	❓ 🚆
🚀 Arlon (17:59)	🏁 Namur (19:46)
👀 Jambes-Est (19:44)	👉 Namur (19:46)

IC → Herstal
(3818)

📅 2025-09-18	🕒 19:49:57
ℹ️ BE:SNCB	❓ 🚆
🚀 Mons (18:38)	🏁 Herstal (20:53)
👀 Ronet (19:47)	👉 Namur (19:50)

uh oh...


Belgien
Arras
Valenciennes
Maubeuge
Cambrai
Wallonie
Koblenz
Frankfurt am Main
Rheinland-Pfalz
Hauts-de-France
Amiens
Saint-Quentin
Lëtzebuerg
Trier
Luxembourg
Mannheim
Saarland
Kaiserslautern
Heidelberg
Reims
Saarbrücken
Metz
Heilbro
Paris
Karlsruhe
Grand Est
Île-de-France
Nancy
Strasbourg
Baden-Württem
Reutling
Orléans
Freiburg im Breisgau
Mulhouse
Centre-Val de Loire
Basel
Winterthur
Bourgogne-Franche-Comté
Besançon
Zürich
Bern

Actually using the data

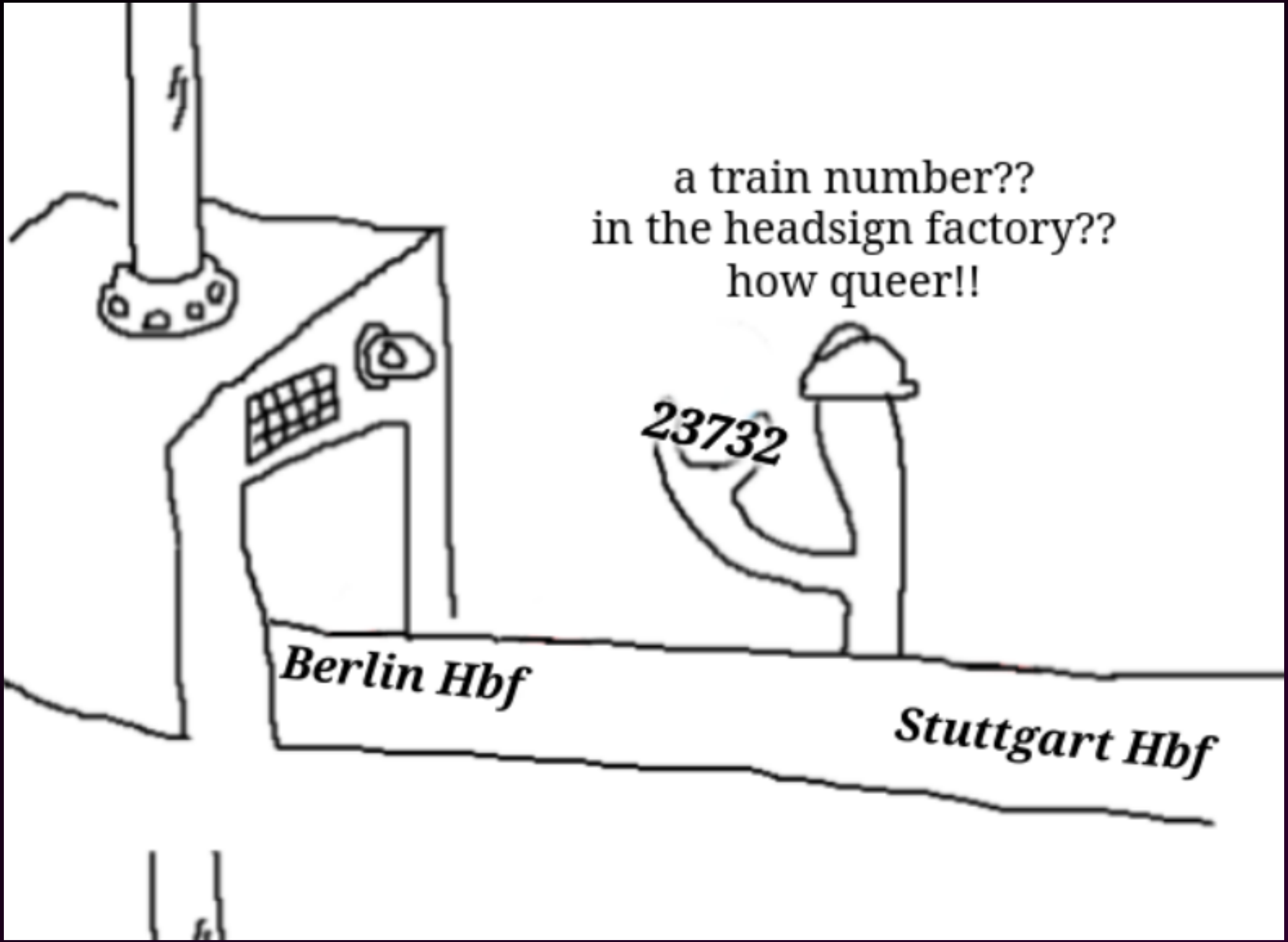
- Problem: this ***will*** contain errors!!
- Main issue: false positives with trips doing massive hops
- Long distance trains often run on the same physical lines as trains will less distance between hops
- Tried to approximate shape using the "smaller" trips
- took me ***a lot*** of compute time and memory (wasn't being very efficient though)

Actually using the data

- This issue is not impossible to fix
- Can precompute transit legs when generating the DB
- Can use detailed `shapes.txt` data (if supplied by transit agency)
- Possible solution: use data source (OSM) to route trips (don't really want to integrate)
- Utils like `pfaedle` can generate `shapes.txt` data using OSM data

Actually using the data

- Need to rework mapper at some point to have better routing and produce more accurate results
- In the meantime, outliers can be pretty easily excluded through a quick lookup
- So we can at least display unified information when we have a match, right?
- ...right?


a train number??
in the headsign factory??
how queer!!
23732
Berlin Hbf
Stuttgart Hbf

Quirks of specific GTFS datasets

- there are ***a lot*** of ways feeds can be harder to process than they need to be
- turns out, many feed providers just:
    - ignore the guidelines
    - don't put all the data they have
    - put the right data *in the wrong field*
    - put outright false data (passenger misinformation systems)

Quirks of specific GTFS datasets

- "missing datapoints": gtfs.de
  - Basic free feed is pretty complete overall, has quite a bit of preprocessing
  - ❌ no train numbers
  - ✅ Can still get the uncompressed full feed for all of Germany and then filter rail routes from there with train numbers

Quirks of specific GTFS datasets

- "you wouldn't get it": opentransportdata.swiss
  - Pretty complete feed with quite a bit of data
  - ❌ nonstandard extensions for `route_type`s
  - ❌ BOMs at the start of files
  - ✅ can convert them back to the standard values
  - ✅ ... can fix parser

Quirks of specific GTFS datasets

- "huh?": Renfe
    - Technically a valid feed
    - ❌ Padded CSV (350 char lines)
    - ✅ ... rewrite the whole parser

Quirks of specific GTFS datasets

- "professional transit misinformation": SNCF
    - truly one of the feeds of all time
    - ❌ every service is a `service_exception`, nothing regular
    - ❌ train numbers in the `trip_headsign` field
    - ❌ plainly wrong route types (buses get shown as trains)
    - ✅ can reformat fields to work well enough
    - ✅ actual route type can be inferred from ***Stop IDs*** (cursed)

Printer protocols aka How Epson Runs The World

- How to print all of this on a reciept?
- Depends on the actual reciept printer
- Most printers (including off-brand ones) support ESC/POS or variations of it
- "Escape / Point of Sale"
- Text-ish based protocol invented by Epson
- *implementations vary*

Printer protocols aka How Epson Runs The World

- Can send text according to a currently selected codepage
- Can also send commands to change codepage
- …or print a whole image (ESC *)
- …or cut the reciept (ESC i/m)
- …or beep (Morse code!)

Rendering entries

- Data in a sight
  - passing time, source feed, trip object, matching stop times/stops
- Text-only is hard to fine-tune
- Fix: render text as an image first, then print as an image
- Thankfully, library has some pretty straightforward image printing

Comically Long Roll™ printing session

- Printer has several interfaces, we use TCP on port 9100
- For each sight, display feed + timestamp + start/previous/next/last stop
- **let's test it!**

Inspiration/Thanks to

- https://signal.eu.org/rail (similar but more advanced project, is integrated with some realtime data and has OSM routing)
- https://github.com/ad-freiburg/pfaedle
- https://github.com/justinmichaelvieira/escpos for letting me just print the damn receipt
- All feed providers for their data

Stuff made

- https://trainmap.cozytren.ch (sights are here)
- https://github.com/rom-vtn/trainmap-db (sights logic)
- https://github.com/rom-vtn/sncf-unfucker (converting feeds)
- https://github.com/rom-vtn/sights-printing (print script, will add presentation there)

That's about it!

Any questions?

Fedi: @trainsignalenjoyer@tech.lgbt