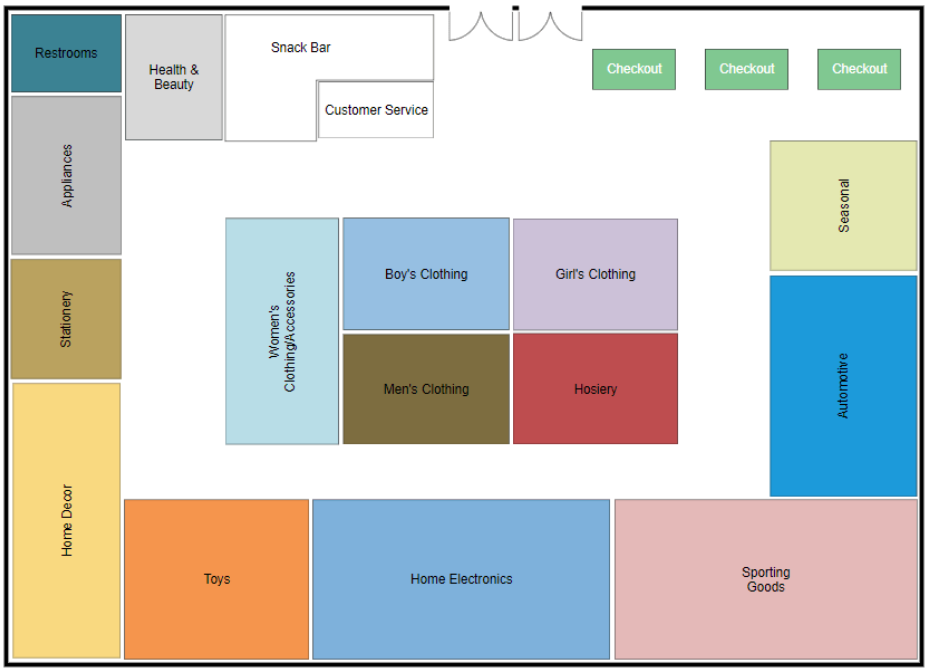
GeoJSON Annotation

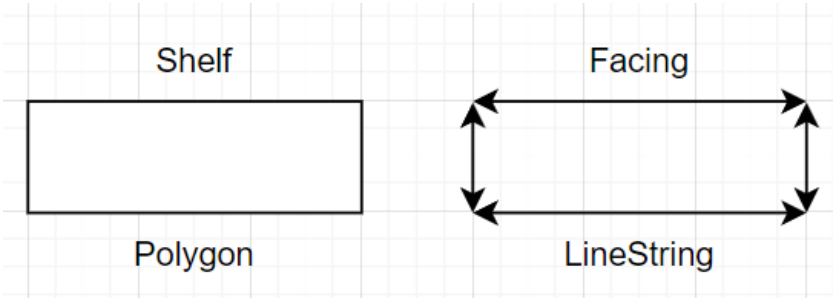
To annotate an indoor map captured by autonomous robots in retail stores, drawings of geometric shapes such as lines and polygons are produced to store information about the indoor structures. These geometric drawings guide robots to correctly identify where things are located and how shelves relate to each other. An example of such annotation can be abstractly visualized as the picture below.



GeoJSON is an open source format based on standard JSON for storing a variety of geometric drawings such as Point, Polygon and LineString. For more information and documentation on GeoJSON format, please refer to https://en.wikipedia.org/wiki/GeoJSON or other resources. In short, GeoJSON is a *FeatureCollection* of a list of GeoJSON *Features* as such

```
{
  "type": "FeatureCollection",
  "features": [array of GeoJSON Features]
}
```

For our shelf annotation purpose, 2 hierarchical geometric structures are needed to encode shelf structures: Shelf and Facing, as demonstrated in below diagram



Shelf - top level drawing encapsulating the boundary of entire shelf display as Polygons
Facing - bottom level drawing representing the four sides of parent Shelf as LineStrings

Example provided here is an annotation system based on the standard GeoJSON Feature format with additional annotation properties:

Shelf

```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [[0,10],[10,10],[10,0],[0,0],[0,10]]
    ]
  },
  "properties": {
    "id": "01" (str),
    "type": "shelf" (str),
    "parent": null,
    "label": "shelf_01" (str),
    "angle": 0.0 (float)
  }
}
```

Facing

```
{
  "type": "Feature",
  "geometry": {
    "type": "LineString",
    "coordinates": [
      [0, 10], [10, 10]
    ]
  },
  "properties": {
    "id": "02" (str),
    "type": "facing" (str),
    "parent": "01" (str),
    "label": "N"(str),
  }
}
```

Custom properties

`"id"` - unique id for each Feature object
`"type"` - Feature type, either “shelf” or “facing”
`"parent"` - id of the parent level drawing, Shelf has no parent (null) and Facing stores id of the parent Shelf
`"label"` - label of the Feature object, Shelf objects are called “shelf_{num}”
Facing can be N/E/S/W representing four sides of the parent shelf in compass directions
`"angle"` - angle of the shelf drawing of float values

Note:

In the provided sample **geojson.json**, there are drawings for 2 square shelves and each shelf has 4 facings. You may modify the **geojson.json** to add more drawing content following the rules defined above as you work through this exercise, but please include the final geojson.json in submission if it's edited. For visualization purposes, geojson.io may be a helpful resource.

You may use any common libraries to accomplish below tasks, but please **do not** use any libraries that directly implements representation or functionality of GeoJSON data structure.

Task 1

In Python3, design and implement class structures to represent above unique custom GeoJSON objects such as *`GeojsonFeatureCollection`*, *`GeojsonShelf`* and *`GeojsonFacing`*. You may introduce additional classes to logically utilize class inheritance (e.g. it's illogical for Shelf to inherit Facing). You may consider various implementations where appropriate such as ***`class`*, `dataclass`, `attrs`, `pydantic`,** etc. to construct proper class hierarchies and inheritance.

Task 2

In the *`GeojsonFeatureCollection`* class from task 1, implement below key functionality to interact with the GeoJSON file and its Feature objects.

- `get_feature(input_feature_id: str) -> Feature`
  Given the id field of Feature as input, retrieve the Feature object. If no Feature with such id exists, return None.

- `get_parent_feature(input_feature: Feature) -> Feature`
  Given a Feature object as input, retrieve its parent Feature object. If there's no parent, return None.

- `get_children_feature(input_feature: Feature) -> Feature`
  Given a Feature object as input, retrieve its children Feature object(s) as a list. If there are no children, return an empty list.

- `get_all_shelves()` and `get_all_facings()`
  Return all available Shelf or Facing objects as a list. If there are none, return an empty list.
- `get_facing_compound_label(facing: Facing) -> str`
  Return a formatted string label of its parent Shelf's label + given Facing's label as *`{shelf_label}_{facing_label}`* such as "*`shelf_01_N`*"

Bonus task

Write simple unit tests for the functionality implemented in above tasks to ensure proper code behaviors and edge cases. You may use any testing framework for this.

Submission

Please include a brief README and zip all relevant files as {firstname}_{lastname}_geojson for submission.