PySunlight – Documentation

PySunlight is a module created in Python that allows to compute the sunlight hours of an apartment.

API Documentation

Constructor

INPUT: Rise and Setting times (optional) in hh:mm format.

Init

INPUT: JSON describint the city, with this format:

[{ neighborhood: <name_string>, apartments_height: <number>, buildings: [{name: <name_string>, apartments_count: <number>, distance: <number>}]}]

The building list is assumed to be ordered from east to west. *Distance* is seen as the distance between the reference point and the building, so distance of element *n* < distance of element *n+1*. The reference point is the point at X meters eastern from the first element of the list, being X the *distance* value of that element.

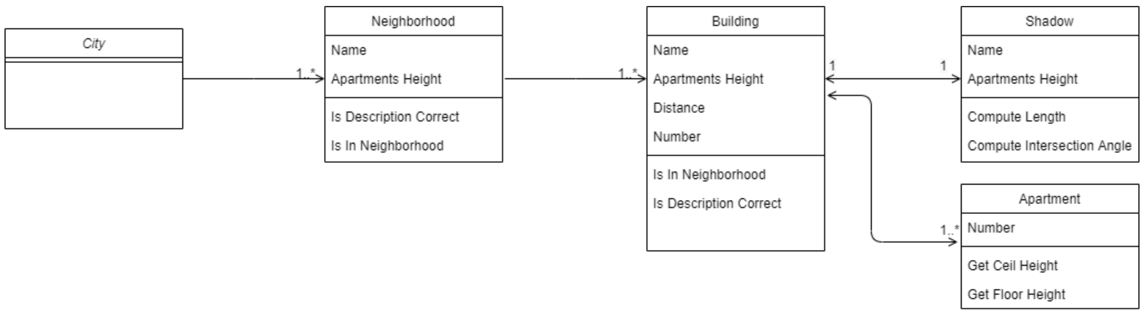
Get_sunlight_hours

INPUT: takes a neighbourhood name, building name, and apartment number
OUTPUT: It returns the sunlight hours as a string like "hh:mm:ss - hh:mm:ss" in 24hr format.

Domain Model

These are the objects used by the system: *City, Neighborhood, Building, Apartment* and *Shadow*.

The interactions are limited: city only interacts with Neighborhood, Neighborhood with Building, and so on.

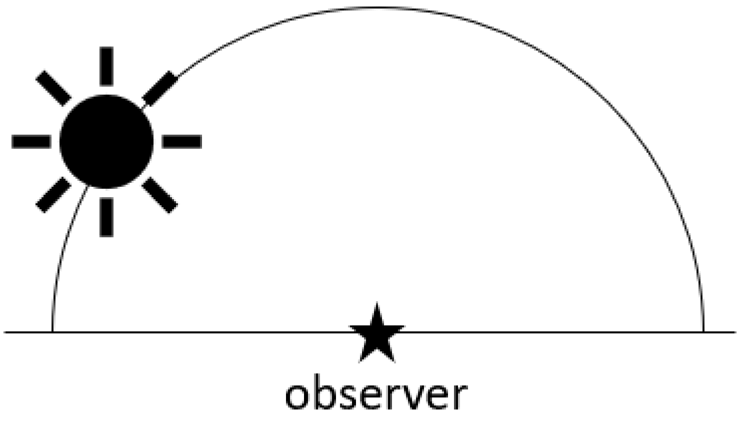


Then, there is a *SunlightCalculator* class that interacts with them and implements the API requested by the system.

Sunlight hours computation

Approximations

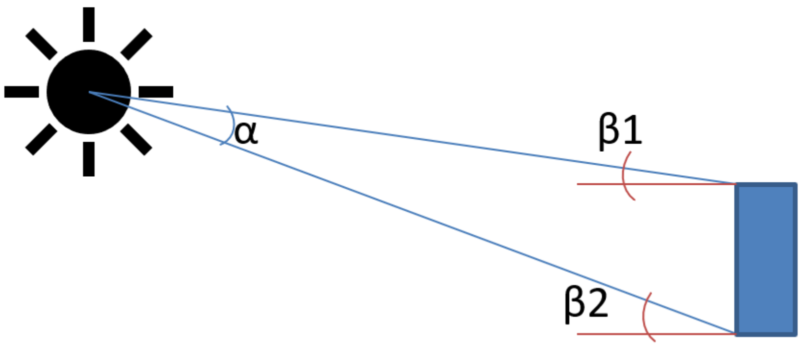
i. 2D model : the buildings of a neighborhood are aligned east to west and the Sun rises east to west.
ii. The Sun travels at a constant radial speed. This means that we can assume that the trajectory of the sun is circular:



iii. For simplicity of the model, it will be assumed that the sun is so far than the angle of incidence of the rays, at a given time, is the same at any point (where the rays are incident) of our system (a neighborhood). This means that, it is not taken into account:

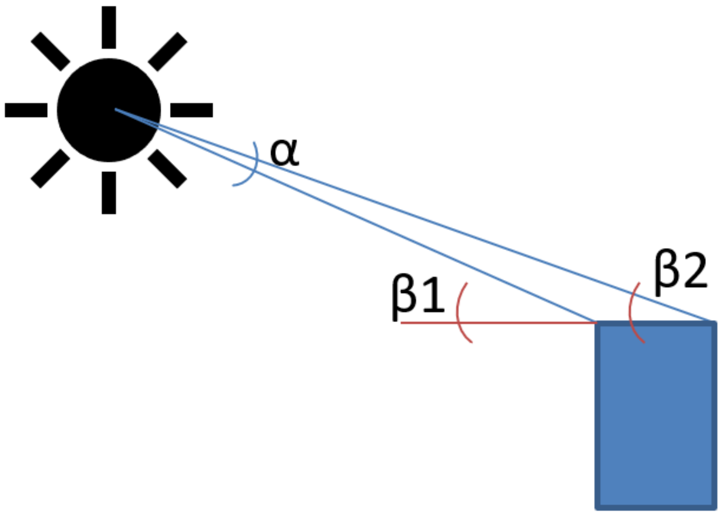
- The height of the buildings:

$$\boldsymbol{\alpha \to 0 => \beta 1 \approx \beta 2}$$



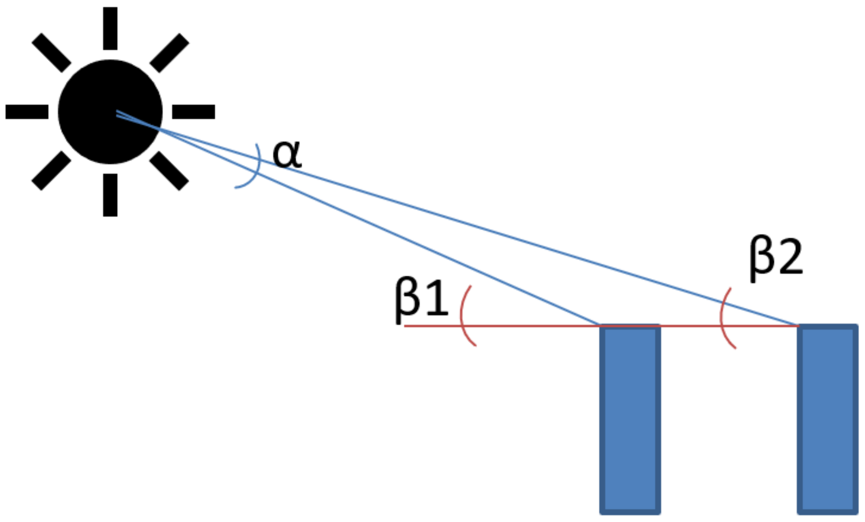
- The width of the buildings:

$$\boldsymbol{\alpha \to 0 => \beta 1 \approx \beta 2}$$



- The distance between buildings:

$$\boldsymbol{\alpha \to 0 => \beta 1 \approx \beta 2}$$

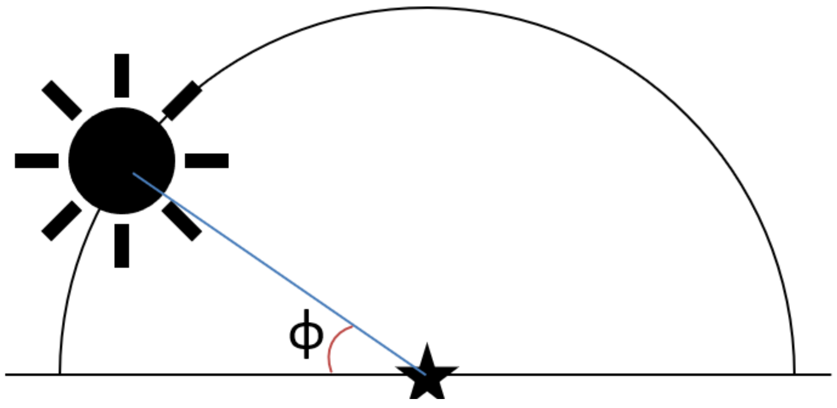


Methodology

The objective of the API is to compute the sunlight hours at of a given apartment specified by neighborhood, building and number of apartment. To do so, each neighborhood corresponds to a 2D (x,y) system (approximation 1) where only the first quadrant is used:

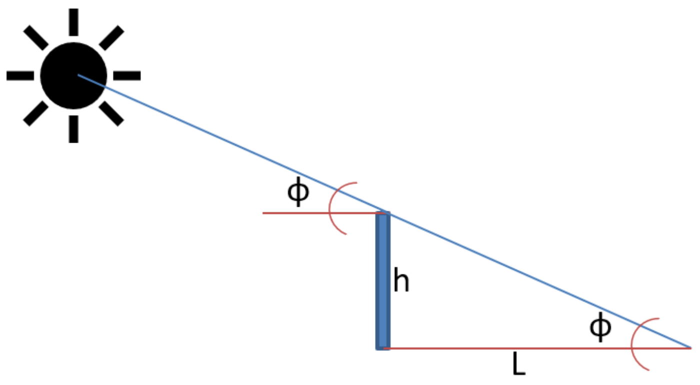
1. X-axis for the distance of the buildings. The *distance* value assigned to a building is directly its position in the x-axis. X-axis is defined from east to west.
2. Y-axis for the height of the buildings.

Approximations (ii) and (iii) allows us to model the position of the Sun by using one single variable, radians, and frees us from taking into account distances:



At any time, a building has a shadow assigned. Once again, due to approx. (ii) and (iii), the shape of the shadow is modeled using the previous incidence angle, which will be the same at any position of the system. Hence, we can compute, for example, the length of the shadow based on the height of the building and the angle:

$$L = h/tg(\Phi)$$



The API compute the sunlight hours of an apartment input by the user. When the Sun is rising, an apartment sunlight will be only affected by those buildings in the east (smaller *x* or *distance*). Moreover, note that when the sun is raising, the shadow of a building is getting shorter and shorter.

To obtain the start hour, the system takes the buildings in the east of the apartment and computes the slope or angle of the straight line that crosses the top of the eastern building and the floor of the apartment (the apartment will be fully covered by sunlight when the shadow is shorter than the ceil of the apartment). This angle is seen as the Φ value on which the apartment starts being fully covered by sunlight.Once again, this can be done because of approx. (iii).

If there is more than one eastern building, the system computes the angle for each one of them, the final result will be the maximum of those values, since there will be no more shadows covering the apartment.

The same procedure is followed when the Sun is setting but using the western buildings. Now, the final result will be the minimum of the values.

Note that, because of the assumptions and approximations, the shadow of a building will never influence to higher apartments of other buildings.

Implementation Details

The *SunlightCalculator* is the class in charge of creating the *City*, *Neighborhoods* and *Buildings* objects. However, the number of building is assigned by the neighborhood since it is the entity containing and ordering them. Finally, the *Apartments* are created by the *Building* itself, since they are part of it and the directly depends on it.

The *Neighborhood* class has two containers defined to store its *Buildings*.

i. buildings: list of objects, ordered from east to west as input by the user.
ii. buildings_numbers: dictionary that maps the name to the number (in the list) of the buildings.

This implementation does not imply a significant increase in memory usage but reduces the computational time of the functions to get the eastern and western buildings.

Error Handling

Several exceptions have been created to distinguish the errors. However, since it is supposed to be an API used in code, the exceptions are not caught but raised.

Testing

I have implemented a unit test for the methods of the classes *City, Neighborhood, Building, Apartment* and *Shadow*. However, the *SunlightCalculator*, which is the “controller”, needs a complete environment to be tested so its testing is not independent from other classes. The test creates the calculator, init() the calculator and tests the sunlight hours of part of the apartments.