```
import numpy as np


class WeatherSimulation:
    #initialize current state to sunny
    def __init__(self, transition_probabilities, holding_times, cur_state='sunny'):

        #initialize transition probabilties,holding times of each weather state,
current weather state
        self.transition_probabilities = transition_probabilities
        self.holding_times = holding_times
        self.cur_state = cur_state
        self.remaining_time = 0

        #check if total probabiltites sum up to 1
        for key, val in transition_probabilities.items():
            total = sum(transition_probabilities[key].values())
            if total !=1:
                #raise runtime error
                raise RuntimeError(
                    'The sum of the transition probabilities is not 1!')

    #list all weather states
    def get_states(self):
        return list(self.transition_probabilities.keys())

    #return current weather state
    def current_state(self):
        return self.cur_state

    #transition to new weather state
    def next_state(self):
        if self.current_state_remaining_hours() == 0:
            # If the remaining hours in '0' time to move to the next stage using
the transition
            # probabilities and update the remaing time to holdin time for the
current state - 1
            # as the state holds the current state for atleast an hour
            current_proba = self.transition_probabilities[self.cur_state]
            new_state = np.random.choice(
                list(current_proba.keys()), p=list(current_proba.values())
            )
            self.set_state(new_state)
            holdin_time = self.holding_times[self.current_state()]
            self.remaining_time = self.holding_times[self.current_state()] - 1
        else:
            # If the remaing hours is not '0' then it doesnt move to the next stage
and decreases the
            # remaining time
            self.set_state(self.current_state())
            self.remaining_time = self.remaining_time - 1

    #change the current state to the new_state
    def set_state(self, new_state):
        self.cur_state = new_state

    #return the remaining hours in the current weather state
    def current_state_remaining_hours(self):
        return self.remaining_time
```

```
#generator function to yield current weather state
def iterable(self):
    while True:
        self.next_state()
        yield self.cur_state

#run the simulation for the amount of hours
def simulate(self, hours):
    self.percentages = {i: 0 for i in self.get_states()}
    for _ in range(hours):
        new_state = self.next_state()
        if(self.cur_state != new_state):
            self.percentages[self.cur_state] += 1
    per = list(self.percentages.values())
    #return probabilty percentages
    res = [i*100/sum(per) for i in per]
    return res
```