1. Definition and code implementation(Python)：

Q: 2022/9/19 10:00 to UTC to BDS time then to GPS time.

① Local time to UTC: When the UTC is the world coordination, the local time here is the Beijing time, and the UTC time is equivalent to the Beijing time minus 8h.

Code:

```
def local_time2UTC(self):
    UTC_hour = (self.hour-8)
    if UTC_hour >= 0:
        self.hour = UTC_hour
    else:
        self.day = self.day - 1
        if self.day < 0:
            self.month -= 1
            if self.month < 0:
                self.year -= 1
        self.hour = UTC_hour + 24

    if self.hour < 10:
        self.hour = str(0)+str(self.hour)
    if self.min < 10:
        self.min = str(0)+str(self.min)
    if self.sec < 10:
        self.sec = str(0)+str(self.sec)
    print("UTC : %d年%d月%d日%s时%s分%s秒" % (self.year, self.month, self.day, self.hour, self.min, self.sec))
```

Idea: First judge the time zone, for Beijing time only need the last time hours to be minus 8, and then judge the time carry, if the number of days is less than one day, the number of days minus one, and then judge the number of months, years, and so on.

② UTC to BDS time: The BDS time starting calendar is coordinated world time (UTC) at 0:0:00 seconds, in the form of week and week seconds count, with the international system of unit (SI) seconds as the basic unit, without second adjustment.

Code:

```
def UTC2BDS(self):
    leap_list, un_leap_list = self.leap(2006)
    e = len(leap_list) + len(un_leap_list) - 1
    day = e * 365 + 1 * len(leap_list)
    monthdays = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]
    if (self.year in leap_list) | (self.month < 3):
        day += (monthdays[self.month-1] + self.day)
    if self.year in un_leap_list:
        day += ((monthdays[self.month-1] - 1) + self.day)
    time = day*24*60*60 + int(self.hour) * 60 * 60 + int(self.sec)
    week = int(time/604800)
    sec = time - week * 604800+4
    if sec >= 604800:
        sec = sec - 604800
        week = week + 1
    print("BDS--周：", week, " 周内秒：", sec)
```

Idea: First, judge the number of leap years from 2006 to 2022, then calculate the number of days, accumulate the days before the required calculation year, and then calculate the time of the current year. When calculating the time of the year, establish a cumulative list of the days of a month. If the year to be calculated is a leap year, add one to the days after February to calculate the cumulative number of seconds. Finally, turn the cumulative number of seconds to week and week seconds (because there is 4 seconds to leap seconds, so there is need to add 4 seconds).

③ UTC to GPS time: The GPS time has a constant difference of 18s from the International atom, and is consistent with the UTC in the GPS standard calendar at 00:00 on January 6,1980.

Codes:

```
def BDS2GPS(self):
    leap_list, un_leap_list = self.leap(1980)
    e = len(leap_list) + len(un_leap_list) - 1
    day = e * 365 + 1 * len(leap_list)
    monthdays = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]
    if (self.year in leap_list) | (self.month < 3):
        day += (monthdays[self.month - 1] + self.day)
    if self.year in un_leap_list:
        day += ((monthdays[self.month - 1] - 1) + self.day)
    time = (day-5) * 24 * 60 * 60 + int(self.hour) * 60 * 60 + int(self.sec)
    week = int(time / 604800)
    sec = time - week * 604800 + 18
    if sec >= 604800:
        sec = sec - 604800
        week = week + 1
    print("GPS--周：", week, " 周内秒：", sec)
```

Idea: consistent with the previous, but the base year is 1980, January 6, finally because of the jump seconds, so need to add 18 seconds of time difference, finally judge whether the number of seconds to reach a week, reach a week, the number of weeks to carry.

④ Judge whether it is a leap year：

Codes:

```python
'''leap year or not'''
def leap(self, start_year):
    leap_list = []
    un_leap_list = []
    for year in range(start_year, self.year+1):
        if year % 4 == 0:
            if year % 100 == 0:
                if year % 400 == 0:
                    leap_list.append(year)
                else:
                    un_leap_list.append(year)
            else:
                leap_list.append(year)
        else:
            un_leap_list.append(year)
    return leap_list, un_leap_list
```

Idea: The key to judging whether it is a leap year is to judge whether the year is a multiple of 4 or a multiple of 400, and finally keep the year in two lists.

2. Results：