```
"""This is the py_athletics goal module."""

from activity.activity import Activity
import datetime
import calendar


class Goal:
    """py_athletics Goal base class."""

    GOAL_METRICS = ("count", "distance", "duration")
    GOAL_TIMEFRAMES = ("month", "year", "cumulative")

    def __init__(self, activity_type, metric, target):
        """Create a Goal.

        A Goal consists of an activity class, a metric, and a target.
        The metric attribute is a string and the target attribute is an
        integer.

        The metric attribute must be one of "count", "distance" or "duration".
        The target attribute must be positive.

        There are three Goal subclasses that have different timeframes.
        They are CumulativeGoal, AnnualGoal and MonthGoal.
        """

        if not issubclass(activity_type, Activity):
            raise ValueError("invalid activity_Class")

        if not isinstance(metric, str):
            raise TypeError("metric must be a string")

        if metric not in Goal.GOAL_METRICS:
            raise ValueError("invalid metric")

        if metric == "distance" and activity_type.__name__ in ["Tennis", "Workout"]:
            raise ValueError("invalid metric for activity_class")

        target = target
        if not isinstance(target, int):
            raise TypeError("target must be an integer")

        if target <= 0:
            raise ValueError("target must be greater than zero")

        self.activity_type = activity_type
        self.metric = metric
        self.target = target

    def __repr__(self):

        # Goal subclasses replace TIMEFRAME with relevant text.

        f_1 = f"[Goal: {self.activity_type.__name__} metric: {self.metric} "
        f_2 = f"timeframe: TIMEFRAME target: {self.target:,}]"

        return f_1 + f_2

    def __str__(self):

        # Goal subclasses replace TIMEFRAME with relevant text.

        if self.metric == "count":
            stub = "times"
        elif self.metric == "distance":
```

```
            stub = "miles"
        else:
            stub = "hours"

        string = f"[Goal: {self.activity_type.__name__} {self.target:,} {stub} "

        return string + "TIMEFRAME]"

    def report(self, tally: dict) -> None:
        print("Not implemented for this goal subclass yet")
        return


class CumulativeGoal(Goal):
    """CumulativeGoal is a goal measured with respect to all relevant
    Activities without reference to any timeframe."""

    def __repr__(self):
        string = super().__repr__()
        return string.replace("TIMEFRAME", "cumulative")

    def __str__(self):
        string = super().__str__()
        return string.replace("TIMEFRAME", "on a cumulative basis")

    def report(self, athlete) -> None:
        tally = Activity.tally(athlete, self.activity_type.__name__)

        target = self.target
        current = tally[self.metric]

        preamble = str(self)
        current_str = f" Current: {current:,} "
        if current >= target:
            delta = f"Achieved with surplus: {current - target:,}"
        else:
            delta = f"Deficit: {target - current:,}"

        summary = preamble[1:-1] + current_str + delta
        print(summary)

        return


class YearGoal(Goal):
    """YearGoal is a goal measured with respect to a calendar year."""

    def __repr__(self):
        string = super().__repr__()
        return string.replace("TIMEFRAME", "year")

    def __str__(self):
        string = super().__str__()
        return string.replace("TIMEFRAME", "each year")

    def report(self, athlete) -> None:
        now = datetime.datetime.now()
        date = now.date()
        year = date.strftime("%Y")
        start = f"{year}-01-01"
        end = f"{year}-12-31"

        tally = Activity.tally(athlete, self.activity_type.__name__, start, end)
        target = self.target
        current = tally[self.metric]
```

```
        preamble = str(self)
        current_str = f", year to date: {current:,} "
        if current >= target:
            delta = f"goal achieved with surplus: {current - target:,}"
        else:
            delta = f"deficit: {target - current:,}"

        result = preamble[1:-1] + current_str + delta
        print(result)

        return


class MonthGoal(Goal):
    """MonthGoal is a goal measured with respect to a calendar month."""

    def __repr__(self):
        string = super().__repr__()
        return string.replace("TIMEFRAME", "month")

    def __str__(self):
        string = super().__str__()
        return string.replace("TIMEFRAME", "each month")

    def report(self, athlete) -> None:
        now = datetime.datetime.now()
        date = now.date()
        year = date.strftime("%Y")
        month = date.strftime("%m")
        start = f"{year}-{month}-01"
        last_day = calendar.monthrange(date.year, date.month)[1]
        end = f"{year}-{month}-{last_day}"

        tally = Activity.tally(athlete, self.activity_type.__name__, start, end)
        target = self.target
        current = tally[self.metric]

        preamble = str(self)
        current_str = f", month to date: {current:,} "
        if current >= target:
            delta = f"goal achieved with surplus: {current - target:,}"
        else:
            delta = f"deficit: {target - current:,}"

        result = preamble[1:-1] + current_str + delta
        print(result)

        # For monthly goal reports we show historical information
        # The Athlete.earlist_activity method shows how far back to go.

        earliest = athlete.earliest_activity(self.activity_type.__name__)

        # If there is no earlier period to summarize, we are done.
        if earliest is None:
            return

        earliest_month = earliest.month
        earliest_year = earliest.year

        # We start at the earliest month we have and push strings on
        # a result string stack.

        result_string_stack = []

        if earliest_year < now.year or earliest_month < now.month:
```

```
        month_index = earliest_month
        year_index = earliest_year

        while year_index < now.year or month_index < now.month:
            start = f"{year_index}-{month_index}-01"
            last_day = calendar.monthrange(year_index, month_index)[1]
            end = f"{year_index}-{month_index}-{last_day}"

            tally = Activity.tally(athlete, self.activity_type.__name__, start, end)
            target = self.target
            prior = tally[self.metric]

            prior_str = f"{year_index}-{month_index:02}: {prior:,} "
            if prior >= target:
                delta = f"goal achieved with surplus: {prior - target:,}"
            else:
                delta = f"deficit: {target - prior:,}"

            result = "      " + prior_str + delta

            result_string_stack.append(result)

            month_index += 1
            if month_index == 13:
                year_index += 1
                month_index = 1

    while result_string_stack:
        print(result_string_stack.pop())
    print()
    return
```