```
import cmd
import readline
import os
import shlex
from collections import defaultdict
from athlete.athlete import Athlete


class PythonAthleticsShell(cmd.Cmd):
    """py_athletics shell class provides the py_athletics command line interpreter.

    This is a subclass of Python's cmd class and inherits tab-key command line
    completion and bash-like history-list editing.  A line beginning with the
    character '!' is executed as an OS shell command.  Help handling is also
    inherited.
"""

    intro = "Welcome to the python_athletics.  Type help or ? for help.\n"
    prompt = "py_athletics: "
    file = None

    athlete = Athlete()

    def do_load(self, arg):
        """Restore py_athletics session from a file.

        The default filename is py_athletics.pickle, a different name can be
        specified with the filename argument.

        Optional Parameters
        -------------------
        filename: string

        Examples
        --------
        load
        load ../test/py_athletics_session.pickle
        """
        try:
            if not arg:
                arg = "py_athletics.pickle"
            PythonAthleticsShell.athlete = Athlete.load(arg)
        except (ValueError, TypeError, FileNotFoundError) as message:
            print(f"load command failed: {message}")

    def do_save(self, arg):
        """Save py_athletics session to a file.

        The default filename is py_athletics.pickle, a different name can be
        specified with the filename argument.

        Optional Parameters
        -------------------
        filename: string

        Examples
        --------
        save
        save ../test/py_athletics_session.pickle
        """
        try:
            if not arg:
                arg = "py_athletics.pickle"
            PythonAthleticsShell.athlete.save(arg)
        except Exception as message:
            print(f"save command failed: {message}")
```

```
    def do_read(self, arg):
        """Read a Garmin activity file and create Activity objects.

        Garmin fitness data is stored at http://connect.garmin.com.
        Subscribers can download comprehensive activity data into a CSV file.
        This method reads a Garmin activity file, creates py_athletics
        Activity objects and adds them to the Athlete's activity collection.

        The  default filename is Activities.csv, a different name can be
        specified with the filename argument.

        Optional Parameters
        -------------------
        filename: string

        Examples
        --------
        read
        read ../test/garmin_data/2021-01.csv
        """

        try:
            if not arg:
                arg = "Activities.csv"
            PythonAthleticsShell.athlete.read_garmin_activity_file(arg)
        except Exception as message:
            print(f"read command failed: {message}")

    def do_add_goal(self, arg):
        """Add a Goal.

        Adds a goal and replaces existing Goals for the same exercise,
        timeframe and metric.  The distance metric is only valid for Cycle,
        Run and Walk activities.

        Keyword Parameters
        ------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}
        metric: string = {count|distance|duration}
        timeframe: string = {month|year|cumulative}
        target: a positive integer

        Examples
        --------
        add_goal exercise=Cycle metric=distance timeframe=year target=1500
        add_goal exercise=Tennis metric=count timeframe=month target=8
        """
        try:
            Athlete.add_goal(PythonAthleticsShell.athlete, **parse(arg))
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_delete_goal(self, arg):
        """Delete a Goal.

        Keyword Parameters
        ------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}
        metric: string = {count|distance|duration}
        timeframe: string = {month|year|cumulative}

        Examples
        --------
        delete_goal exercise=Cycle metric=distance timeframe=year
        delete_goal exercise=Tennis metric=count timeframe=month
```

```
        """
        try:
            Athlete.delete_goal(PythonAthleticsShell.athlete, **parse(arg))
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_show_goals(self, arg):
        """Display a list of Goals.

        If exercise is specified, the listing is limited to that exercise.

        Optional Parameters
        -------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}

        Examples
        --------
        show_goals
        show_goals Cycle
        """
        try:
            if not arg:
                arg = None
            Athlete.show_goals(PythonAthleticsShell.athlete, arg)
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_summarize_goals(self, arg):
        """Display a summary of Goals.

        If exercise is specified, the listing is limited to that exercise.

        A goal summary shows the goal and how the athlete is tracking relative
        to the goal for the appropriate timeframe.  In addition, for monthly
        goals, summary information for prior months is also shown.

        Optional Parameters
        -------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}

        Examples
        --------
        summarize_goals
        summarize_goals Cycle
        """
        try:
            if not arg:
                arg = None
            Athlete.summarize_goals(PythonAthleticsShell.athlete, arg)
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_show_activities(self, arg):
        """Display a list of Activities.

        If exercise is specified the listing is limited to that exercise.
        A timeframe for the listing can be established with one or both of the
        start and end keywords.

        Optional Parameters
        -------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}
        start: string in the form YYYY-MM-DD
        end: string in the form YYYY-MM-DD

        Examples
```

```
        --------
        show_activities
        show_activities exercise=Tennis
        show_activities exercise=Tennis start=2021-05-01 end=2021-06-30
        """
        try:
            Athlete.show_activities(PythonAthleticsShell.athlete, **parse(arg))
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_summarize_activities(self, arg):
        """Display a summary of Activities.

        If exercise is specified the listing is limited to that exercise.
        A timeframe for the listing can be established with one or both of the
        start and end keywords.

        Optional Parameters
        -------------------
        exercise: string = {Cycle|Run|Tennis|Walk|Workout}
        start: string in the form YYYY-MM-DD
        end: string in the form YYYY-MM-DD

        Examples
        --------
        summarize_activities
        summarize_activities exercise=Tennis
        summarize_activities exercise=Tennis start=2021-05-01 end=2021-06-30
        """

        try:
            Athlete.summarize_activities(PythonAthleticsShell.athlete, **parse(arg))
        except (ValueError, TypeError) as message:
            print(f"py_athletics command failed: {message}")

    def do_exit(self, arg):
        """Exit py_athletics."""
        print("Thank you for using py_athletics.")
        return True

    def do_run_script(self, arg):
        """ Run py_athletics commands from a script.

        The default filename is py_athletics.cmd, a different name can be
        specified with the filename argument.

        Optional Parameters
        -------------------
        filename: string

        Examples
        --------
        run_script
        run_script ../test/goals.cmd
        """

        try:
            if arg:
                filename = arg
            else:
                filename = "py_athletics.cmd"

            with open(filename) as file:
                self.cmdqueue.extend(file.read().splitlines())
        except (ValueError, TypeError, FileNotFoundError) as message:
            print(f"run_script command failed: {message}")
```

```
    # ----- shell  -----
    def do_shell(self, arg: str):
        """Run an OS shell command.

        Example
        -------
        ! ls ../test/
        """

        try:
            print(arg)
            os.system(arg)
        except Exception as message:
            print(f"OS shell command failed: {message}")


def parse(arg: str) -> dict:
    "py_athletics shell command parser."

    # Valid arguments are of the form keyword=value.
    # The parser splits arg into tokens and then parses
    # each resulting token into a keyword value pair to
    # appear in an argument dictionary that is returned.
    # The parser also casts integer strings to integers.

    try:
        token_list = shlex.split(arg)
        arg_dict = defaultdict(lambda: None)
        if token_list:
            arg_dict = dict(token.split("=") for token in token_list)
    except ValueError:
        print("could not parse command arguments")

    for key, value in arg_dict.items():
        try:
            arg_dict[key] = int(value)
        except ValueError:
            pass
    return arg_dict
```