10/11/2021 11:15 AM Chicago Time

Python Athletics (pyAthletics) Design

Richard Robbins

A tool for gathering, organizing and analyzing fitness activities.

pyAthletics allows athletes to keep track of their exercise activity and gear. An athlete can establish weekly, monthly, annual or cumulative goals. Athletes can generate summary activity reports that show, among other things, progress to goals. The system will generate gear maintenance alerts and provide a means for athletes to clear the alert and log maintenance activity.

pyAthletics will be able to read Garmin fitness activity files and create activity objects from those files. A command loop interface will be be provided for some elements of the system. Various configuration and system instantiation tasks will be handled by Python scripts. The command loop is meant to showcase how a running system operates.

An `Athlete` has a name and collections of `Activities`, `Goals` and `Gear`. The system can read a Garmin activity file and create Activity objects from that file for **new** Activities. An Athlete's information can be saved to a file and restored from a file. The system includes methods to generate various reports pertaining to an Athlete's Activities, Gear and Goals.

pyAthletics command line commands:

COMMAND	DESCRIPTION
create name script	creates an Athlete and runs an optional script to specify Gear, Gear defaults and Goals
run script	execute a Python script with pyAthletics method calls to access all pyAthletics capability inside command loop
read filename	reads a Garmin activity file, creates Activity objects and associates those objects with the Athlete
save filename	save the Athlete object to a file
load filename	load an Athlete object from a file
add activity	creates a new activity in response to a series of prompts from the system
print report_type	print report_type where report_type is one of activity, goal, maintenance_alert, maintenance_log
maintain name comment	log a maintenance event on the item of Gear named name
quit	quit

Each `Activity` has start and end timestamps and may include gear as well as other attributes reflected in a Garmin activity file or added manually by calling a **pyAthletics** method. The Activity subclasses consist of `Walk`, `Run`, `Tennis`, `Cycling` and `GymWorkout`. An Athlete can associate a set of default Gear to be added to Activity subclass instances. For example an Athlete can identify specific running shoes to be added by default to each run and a set of cycling shoes to be added by default to each bike ride and so forth. An Athlete can also specify that gear be allocated to Activities on a rotating basis, for example, alternating between two sets of running shoes evenly. Gear defaults are a convenient way to pre-populate activities with commonly used gear.

Athletes can adjust the gear used in any particular activity instance as need be. The system does not include a means for the end user to generate additional Activity subclasses. That capability could be a future enhancement.

Every item of `Gear` has an in service date and may include a range of attributes contemplated by the Gear base class or Gear subclasses. The Gear subclasses consist of `Shoes`, `TennisRacquet`, `Bicycle`,`BicycleWheels`, `BicycleTires`, and `StationaryTrainer`. Each Gear subclass provides `add_activity` and `remove_activity` methods to associate and disassociate an activity instance from a Gear instance. An item of gear cannot be used in an Activity before its in_service_date or after its retirement_date. A Gear subclass may provide a group of maintenance methods.

A `Goal` consists of an `Activity` subclass, a `metric` , a `target` and a `time_frame` which can be weekly, monthly, annual or cumulative. The goal is satisfied if a method that measures the metric during a time frame of all of the Athlete's Activities of the specified subclass meets or exceeds the target. Metrics include things like activity counts or cumulative distance. For example, 150 miles of cycling in a week or 6 GymWorkouts a month. The Athlete.activity summary method calls uses Goal methods to generate goal-related reports.

For purposes of this Project, the user interface will be limited to a command loop providing the ability to load and save the system state, use the automated update process when presented with new Garmin data files, add an activity, review and log maintenance events and run various activity, goal and gear reports. Athlete and Gear instantiation and default Gear use specification will be limited to a Python script that contains direct calls to the relevant methods and then saves the system state to a file. All of these activities could ultimately be replaced with a more refined graphical user interface, but for our purposes, demonstrating base capabilities will need to suffice.

I realize this is a very ambitious specification and that I am unlikely to be able to implement all of it in the time allotted to this exercise. I intend to take a minimal viable product approach in order to showcase all of the critical elements. Some more elaborate features may be omitted. Some subclasses may be omitted. pyAthletics has a rich collection of object classes and can be used in an interactive fashion. If I run into code size issues, I will scale back what I deploy. I assume that initialization scripts and external data will not count against the code size ceiling. The interaction between objects is rich and I believe this design satisfies the project requirements.

Classes

`Athlete`

An Athlete object consists of a name and lists of activities, gear and goals.
instance attributes: name, activities, gear, goals

Methods:

Athlete(name: string): create an Athlete object

add_activity(activity: Activity) add an activity to an Athlete's Activity collection

add_gear(gear: Gear) add an item of Gear to an Athlete's Gear collection

add_goal(goal: Goal) add a Goal to an Athlete's Goal collection

save(filename: string) save an Athlete to a file

load(filename: string) load an Athlete from a file load_garmin_data(filename: string) parse a Garmin activity file and add new Activity objects

activity_summary(type={run|bike|walk|tennis|workout|comprehensive}, start: date, end: date): generate and print a summmary of activity based on the type string argument for the specified time frame

goal_summary(type={week|month|year|comprehensive}, start: date, end: date): print a summary of goal attainment information for the specified time frame

maintenance_alerts(): print a list of suggested maintenance for an Athlete's gear

maintenance_log(): print a log of maintenance activity for an Athlete's gear

`Activity {base class}`

Every Activity instance will have start and end time stamps and may include the other attributes. An Activity subclass may have a default_gear attribute that is used to pre-populate subclass instances with particular gear objects. A means will be provided that allows an Athlete to specify that certain default gear objects be used as defaults in an even rotation.

subclass class attributes: default_gear
subclass instance attributes: start, end, venue, venue_type {indoor|outdoor}, maximum_heart_rate, average_heart_rate, calories, gear, comments

Methods:

Activity(start: datetime, end: datetime): create an Activity object, other class attributes noted above or in subclass specifications below may also be specified

add_gear(gear: Gear): associate a Gear instance with an Activity instance, at a minimum, by calling gear.add_activity(Activity)

remove_gear(gear: Gear): disassociate a Gear instance with an Activity instance, at a minimum by calling gear.remove_activity(Activity)

Subclasses:

`Walk` attributes: distance, maximum_speed, average_speed, type {track|road|treadmill}

`Run` attributes: distance, maximum_speed, average_speed, type {track|road|treadmill}

`Tennis` attributes: type = {cardio|drill|ball_machine|lesson|match|hitting_session}, partner

`Cycling` attributes: distance, maximum_speed, average_speed, maximum_power, average_power, type = {road|trail|commute|stationary}, wheels

`GymWorkout` attributes: type {personal_training|solo}, trainer

`Gear {base class}`

Each gear subclass must provide `add_activity` and `remove_activity` methods to associate and disassociate an activity instance from a gear instance. An item of gear cannot be used in an activity before its in_service_date or after its retirement_date. A gear subclass may provide a group of maintenance methods.

attributes: name, brand, model, size, in_service_date, retirement_date, maintenance_rules, maintenance_events, activities

Class Methods:

Gear(name, in_service_date: datetime): create a Gear object, other class attributes noted above or in subclass specifications below may also be specified

Subclass Methods:

add_activity(activity: Activity) called by Activity.add_gear to associate an Activity instance with the Gear instance

remove_activity(activity: Activity) called by Activity.remove_gear to disassociate an Activity instance with the Gear instance

maintenance_alerts(): returns a string describing any recommended maintenance or None if no maintenance is recommended

log_maintenance(date: date, description: string) records a maintenance event for the gear object with the effect of deeming all prior maintenance alerts satisfied

Subclasses:

`Shoes` attributes: size, color, type = {running|cycling|tennis|cross_training|recreation}

`TennisRacquet` attributes: grip, string, string_tension, string_gauge, string_pattern = {full_bed|hybrid_main|hybrid_cross}, stringer, stringing_date

`Bicycle` attributes: size, color, default_wheels

`BicycleWheels` attributes: size, tires

`BicycleTires` attributes: size, type {road|trainer}

`StationaryTrainer` attributes: type {direct_drive|through_axle}

`Goal`

A `Goal` consists of an `Activity` subclass, a `metric_calculator` , a `target` and a `time_frame` which can be weekly, monthly, annual or cumulative. The goal is satisfied if a method that measures the metric during a time frame of all of the Athlete's Activities of the specified subclass meets or exceeds the target. Metrics include things like activity counts or cumulative distance. For example, 150 miles of cycling in a week or 6 GymWorkouts a month. The Athlete.activity summary method calls uses Goal methods to generate goal-related reports.

instance attributes: activity_subclass, metric, target, time_frame-{week|month|year|cumulative}, target

Methods:

Goal(activity_subclass: class, metric: callable, target: number, time_frame=-{week|month|year|cumulative)) Create a Goal object

goal_attained(Athlete, start: datetime) returns True if the Athlete reached the Goal during the time period commencing with start

goal_measure(Athlete, start: datetime) returns the value of the goal metric over the time period commencing with start

goal_gap(Athlete, start: datetime) returns the surplus (deficit) of the goal_measure to the target over the time period commencing with start