athlib Documentation

Release 0.1

Andy Robinson and others

February 23, 2017

Contents:

CHAPTER 1

Introduction

CHAPTER 2

athlib package

Note that all functions are available in the top level athlib package, even thought they may be defined in submodules.

Utilities

General athlib utility functions

athlib.utils.**check_performance_for_discipline**(*discipline*, *textvalue*)
Fix up and return what they typed in, or raise ValueError

athlib.utils.**event_sort_key**(*event_name*)
Return a tuple which will sort into programme order

Track should be ordered by distance.

athlib.utils.**get_distance**(*discipline*)
Return approx distance in metres, for sanity checking :param discipline: :return:

athlib.utils.**normalize_gender**(*gender*)
Return M, F or raise a ValueError

athlib.utils.**parse_hms**(*t*)
Parse a time duration with 0, 1 or 2 colons and return seconds.

```
>>> from athlib.utils import parse_hms
>>> parse_hms('10')
10
>>> parse_hms('1:10')
70
>>> parse_hms('1:1:10')
3670
>>> parse_hms('1:1:10.1')
3670.1
>>> parse_hms(3670.1)
3670.1
```

athlib.utils.**sort_by_discipline**(*stuff*, *attr='discipline'*)
Sort dicts or objects into the normal athletics order

athlib.utils.**text_event_sort_key**(*event_name*)
Return a text version of the event_sort_key

Utilities for working with JSON and json-like structures - deeply nested Python dicts and lists.

This lets us iterate over child nodes and access elements with a dot-notation.

class athlib.jsondict.**JSONDict**
Allows dotted access

class athlib.jsondict.**JSONDictSafe**
Allows dotted access

Age Groups

athlib.uka.agegroups.**calc_age_group**(*birth_date*, *match_date*, *category*, *vets=True*, *under-age=False*)
Return UKA age group

IAAF scoring

This file contains definitions and utility functions for determining IAAF event scores.

athlib.iaaf_score.**performance**(*gender*, *event_code*, *score*)
Function to determine performance required to achieve IAAF score, given gender and event.

In the interface, we assume performance is <seconds> for track events, and <metres> for throws and jumps. Ihe the Wikipedia-sourced factors, jumps are <centimetres>. Therefore there is a factor of 100 applied at the end.

athlib.iaaf_score.**score**(*gender*, *event_code*, *value*)
Function to determine IAAF score, based on gender, event and performance.

In the interface, we assume performance is <seconds> for track events, and <metres> for throws and jumps. Ihe the Wikipedia-sourced factors, jumps are <centimetres>. Therefore there is a factor of 100 applied at the end.

athlib.iaaf_score.**scoring_key**(*gender*, *event_code*)
Utility function to get the <gender>-<event> scoring key.

athlib.iaaf_score.**unit_name**(*event_code*)
Utility function to get the unit name based on event type.

Masters utilities

class athlib.wma.agegrader.**AgeGrader**
We implement an object to cache the data used for lookups.

end users will appear to be calling a function.

calculate_age_grade(*gender*, *age*, *event*, *performance*, *verbose=False*)
Return the age grade score (0 to 100ish) for this result.

```
>>> from athlib.wma.agegrader import AgeGrader
>>> ag=AgeGrader()
>>> "%0.4f" % ag.calculate_age_grade('m',50,'5K', '16:23')
'0.9004'
>>> "%0.4f" %  ag.calculate_age_grade('f',50,'5K', '18:00')
'0.9179'
>>>
```

calculate_factor(*gender*, *age*, *event*, *distance=None*)
Work out 'slowdown factor' for a geezer of this age taking part in this event e.g.

```
>>> from athlib.wma.agegrader import AgeGrader
>>> ag=AgeGrader()
>>> ag.calculate_factor('M',68,'5k')
0.7592
>>> ag.calculate_factor('M',68,'200K')
0.7561
>>> ag.calculate_factor('M',68.5,'200K')
0.7522
>>> ag.calculate_factor('f',35,'5k')
0.9935
>>> ag.calculate_factor('f',35,'200K')
0.9926
>>> ag.calculate_factor('F',35.5,'200K')
0.99095
>>> ag.calculate_factor('M',65,'10000')
0.7691
>>> ag.calculate_factor('M',69,'10000')
0.7402
>>> ag.calculate_factor('F',35,'1500')
0.9822
>>> ag.calculate_factor('f',39,'1500')
0.9547
>>> ag.calculate_factor('f',35,'SH')
0.9791
>>> ag.calculate_factor('f',39,'SH')
0.9576
>>> ag.calculate_factor('m',35,'LH')
0.9647
>>> ag.calculate_factor('m',39,'LH')
0.9254
```

get_data()
Defer this until the first call, so we can bubble a function up to the top of the package

world_best(*gender*, *event*)
The relevant world-record performance on the date stats were compiled

Codes

codes.**JUMPS** = **('HJ', 'PV', 'LJ', 'TJ')**

codes.**THROWS** = **('DT', 'JT', 'HT', 'SP', 'WT')**

codes.**MULTI_EVENTS** = **('PEN', 'HEP', 'DEC', 'PENI', 'PENWT')**

codes.**FIELD_EVENTS** = **('HJ', 'PV', 'LJ', 'TJ', 'DT', 'JT', 'HT', 'SP', 'WT')**

codes.**FIELD_SORT_ORDER** = **['HJ', 'PV', 'LJ', 'TJ', 'SP', 'DT', 'HT', 'JT']**

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a

A

C

E

F

G

J

M

N

P

S

T

U

W