Brewday Documentation

Release 0.0.5

Chris Gilmer

September 26, 2016

BrewDay is a set of tools for homebrewers written in python.

CHAPTER 1

Tutorial

1.1 Features

- Work with recipes, grains, hops and yeast
- Develop recipes
- Understand bitterness and color

Back to Index

CHAPTER 2

API Docs

2.1 brew.grains

class brew.grains.**Grain**(*name*, *color=None*, *ppg=None*, *hwe=None*)
A representation of a type of grain.

format()

get_working_yield(*percent_brew_house_yield*)
Working Yield Working Yield is the product of the Hot Water Extract multiplied by the Brew House Yield. This product will provide the percent of extract collected from the malt.

WY = (HWE as-is)(BHY)

to_dict()

to_json()

class brew.grains.**GrainAddition**(*grain*, *weight=None*, *grain_type='cereal'*, *units='imperial'*)
A representation of the grain as added to a Recipe.

change_units()
Change units from one type to the other return new instance

format()

get_cereal_weight()
Get the weight of the addition in cereal weight

get_dry_weight()
Get the weight of the addition in Dry Malt Extract weight

get_lme_weight()
Get the weight of the addition in Liquid Malt Extract weight

get_weight_map()

set_units(*units*)

to_dict()

to_json()

classmethod validate(*grain_data*)

2.2 brew.hops

class brew.hops.**Hop**(*name*, *percent_alpha_acids=None*)

A representation of a type of Hop.

format()

to_dict()

to_json()

class brew.hops.**HopAddition**(*hop*, *weight=None*, *boil_time=None*, *hop_type='pellet'*, *utilization_cls=<class 'brew.utilities.hops.HopsUtilizationGlennTinseth'>*, *utilization_cls_kwargs=None*, *units='imperial'*)

A representation of the Hop as added to a Recipe.

change_units()

Change units from one type to the other return new instance

format()

get_alpha_acid_units()

Alpha Acid Units

Defined as ounces of hops * alpha acids

get_hops_weight(*sg*, *target_ibu*, *final_volume*, *percent_contribution*)

Weight of Hops IBUs or International Bittering Units measures a bitterness unit for hops. IBUs are the measurement in parts per million (ppm) of iso-alpha acids in the beer. For example, an IPA with 75 IBUs has 75 milligrams of isomerized alpha acids per liter. The equation used to calculate the weight of hops for the boil is as follows.

Ounces hops = (IBU Target)(galbeer)(IBU%) / (%a-acid)(%Utilization)(7489)

The IBU target equals the total bitterness for the beer. (e.g. an IPA may have an IBU target of 75 IBUs) The percent IBU is equal to the percent of IBUs from each hop addition. You may wish for your first hop addition to contribute 95% of the total IBUs. This would make your IBU% 95%. The %a-acid is the amount of alpha acid in the hops and can be found on the hop packaging. The % Utilization is a measurement of the percentage of alpha acid units that will isomerize in the boil. The following chart outlines the typical utilizations and hop boil times.

60 min = 30% utilization 30 min = 15% 5 min = 2.5%

The 7489 is a conversion factor and used to cancel the units in the equation, converting oz/gallon to mg/l. For the hops equation, the units for the % must be expressed in decimal form. (e.g. 10%= .10)

Source: - http://www.learntobrew.com/page/1mdhe/Shopping/Beer_Calculations.html # nopep8

get_ibus(*sg*, *final_volume*)

set_units(*units*)

to_dict()

to_json()

classmethod validate(*hop_data*)

2.3 brew.yeasts

class brew.yeasts.**Yeast**(*name*, *percent_attenuation=0.75*)
A representation of a type of Yeast as added to a Recipe.

format()

to_dict()

to_json()

classmethod validate(*yeast_data*)

2.4 brew.recipes

class brew.recipes.**Recipe**(*name*, *grain_additions=None*, *hop_additions=None*, *yeast=None*, *percent_brew_house_yield=0.7*, *start_volume=7.0*, *final_volume=5.0*, *units='imperial'*)
A representation of a Recipe that can be brewed to make beer.

change_units()
Change units from one type to the other return new instance

format()

get_boil_gravity()

get_boil_gravity_units()

get_brew_house_yield(*plato_actual*, *vol_actual*)
Brew House Yield (BHY) Brew house yield is a measurement that tells the efficiency of the brewing. The actual degrees Plato from the brew and the actual gallons collected out of the kettle are needed to calculate the BHY.

BHY = [(Pactual)(galactual)(BHYtarget)] / [(Ptarget)(galtarget)]

get_bu_to_gu()
Returns ratio of Bitterness Units to Original Gravity Units

get_degrees_plato()

get_extract_weight()
Weight of Extract The weight of extract is the amount of malt extract present in the wort.

Lbs extract = (density of water) * (gal of wort) * (SG) * (P/100)

The weight of one gallon of water in the above formula is 8.32 lbs/gal

To find the weight of a gallon of wort, multiply the specific gravity of the wort by the density of water.

Plato is a percentage of sugars by weight. So 10 Plato means solution is 10% sugars. In this equation we convert the degrees plato to a decimal number between 0.0 and 1.0 by dividing it by 100. This is multiplied by the weight of a gallon of wort.

get_final_gravity()

get_final_gravity_units()

get_grain_add_cereal_weight(*grain_add*)
When converting DME or LME to grain its important to remember that you can't get 100% efficiency from grains. Dividing by the brew house yield will increase the size of the grain accordingly.

get_grain_add_dry_weight(*grain_add*)
When converting Grain to DME its important to remember that you can't get 100% efficiency from grains. Multiplying by the brew house yield will decrease the size of the DME accordingly.

get_mash_water_volume(*liquor_to_grist_ratio*)
Mash Water Volume To calculate the mash water volume you will need to know your liquor to grist ratio. The term liquor refers to the mash water and grist refers to the milled malt. We need to calculate the appropriate amount of water to allow for enzyme action and starch conversion take place.

gallons H2O = (Lbs malt)(L:G)(1gallon H2O) / 8.32 pounds water

get_original_gravity()

get_original_gravity_units()

get_percent_ibus(*hop_add*)
Get the percentage the hops contributes to total ibus

get_percent_malt_bill(*grain_add*)
Percent malt bill is how much extract each grain addition adds to the recipe. To ensure different additions are measured equally each is converted to dry weight.

classmethod get_strike_temp(*mash_temp*, *malt_temp*, *liquor_to_grist_ratio*)
Strike Water Temp As you know when you are mashing, your strike water has to be warmer than the target mash temperature because the cool malt will cool the temperature of the water. To correctly calculate the temperature of the strike water, use the following formula.

Strike Temp = [((0.4)(T mash-T malt)) / L:G] + T mash

get_total_dry_weight()

get_total_grain_weight()

get_total_ibu()
Convenience method to get total IBU for the recipe

get_total_points()

get_total_wort_color()
Convenience method to get total wort color

get_total_wort_color_map()
Convenience method to get total wort color

get_wort_color(*grain_add*)

get_wort_color_mcu(*grain_add*)
Calculation of Wort and Beer Color

Color of Wort = S [(% extract)(L of malt)(P wort / 8P reference)]

Source: http://beersmith.com/blog/2008/04/29/beer-color-understanding-srm-lovibond-and-ebc/ http://brewwiki.com/index.php/Estimating_Color

grain_lookup = {}

hop_lookup = {}

set_units(*units*)

to_dict()

to_json()

classmethod validate(*recipe*)

2.5 brew.parsers

class brew.parsers.**DataLoader**(*data_dir*)
Base class for loading data from data files inside the data_dir.

DATA = {}

EXT = ''

classmethod format_name(*name*)
Reformat a given name to match the filename of a data file.

get_item(*dir_suffix*, *item_name*)

classmethod read_data(*filename*)

class brew.parsers.**JSONDataLoader**(*data_dir*)
Load data from JSON files inside the data_dir.

DATA = {}

EXT = 'json'

format_name(*name*)
Reformat a given name to match the filename of a data file.

get_item(*dir_suffix*, *item_name*)

classmethod read_data(*filename*)

parsers.**parse_cereals**(*cereal*, *loader*)
Parse grains data from a recipe

Grain must have the following top level attributes: - name (str) - weight (float) - data (dict) (optional)

Additionally grains may contain override data in the 'data' attribute with the following keys: - color (float) - ppg (int)

parsers.**parse_hops**(*hop*, *loader*)
Parse hops data from a recipe

Hops must have the following top level attributes: - name (str) - weight (float) - boil_time (float) - data (dict) (optional)

Additionally hops may contain override data in the 'data' attribute with the following keys: - percent_alpha_acids (float)

parsers.**parse_yeast**(*yeast*, *loader*)
Parse yeast data from a recipe

Yeast must have the following top level attributes: - name (str) - data (dict) (optional)

Additionally yeast may contain override data in the 'data' attribute with the following keys: - percent_attenuation (float)

parsers.**parse_recipe**(*recipe*, *loader*, *cereals_loader=None*, *hops_loader=None*, *yeast_loader=None*)
Parse a recipe from a python Dict

recipe: a python dict describing the recipe loader: a data loader class that loads data from data files

A recipe must have the following top level attributes: - name (str) - start_volume (float) - final_volume (float) - grains (list(dict)) - hops (list(dict)) - yeast (dict)

Additionally the recipe may contain override data in the 'data' attribute with the following keys: - percent_brew_house_yield (float) - units (str)

All other fields will be ignored and may be used for other metadata.

The dict objects in the grains, hops, and yeast values are required to have the key 'name' and the remaining attributes will be looked up in the data directory if they are not provided.

2.6 brew.validators

validators.**validate_grain_type**(*grain_type*)

validators.**validate_hop_type**(*hop_type*)

validators.**validate_percentage**(*percent*)

validators.**validate_units**(*units*)

validators.**validate_required_fields**(*data*, *required_fields*)
Validate fields which are required as part of the data.

data: a python dict required_fields: a list of tuples where the first element is a string with

a value that should be a key found in the data dict and where the second element is a python type or list/tuple of python types to check the field against.

validators.**validate_optional_fields**(*data*, *optional_fields*, *data_field='data'*)
Validate fields which are optional as part of the data.

data: a python dict optional_fields: a list of tuples where the first element is a string with

a value that should be a key found in the data dict and where the second element is a python type or list/tuple of python types to check the field against.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

C

D

E

F

G

H

J

P

R

S

T

V

Y