Brewday Documentation

Release 0.0.5

Chris Gilmer

September 28, 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.

Parameters

- **name** (*str*) – The name of the grain
- **color** (*float*) – The color of the grain in SRM
- **ppg** (*float*) – The potential points per gallon
- **hwe** (*float*) – The hot water extract value

Raises **Exception** – If both ppg and hwe are provided

format()

get_working_yield(*percent_brew_house_yield*)

Get Working Yield

Parameters **percent_brew_house_yield** (*float*) – The Percent Brew House Yield

Returns The working yield

Return type float

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.

Parameters

- **grain** (Grain) – The Grain object
- **weight** (*float*) – The weight of the grain addition
- **grain_type** (*str*) – The type of the grain being used
- **units** (*str*) – The units

change_units()

Change units of the class from one type to the other

Returns Grain Addition in new unit type

Return type *GrainAddition*

format()

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

Returns Cereal weight

Return type float

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

Returns Dry weight

Return type float

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

Returns LME weight

Return type float

get_weight_map()
Get map of grain weights by type

Returns Grain weights

Return type dict

set_units(*units*)
Set the units and unit types

Parameters **units** (*str*) – The 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.

Parameters

- **name** (*str*) – The name of the hop
- **percent_alpha_acids** (*float*) – The percent alpha acids in the 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.

Parameters

- **hop** (Hop) – The Hop object
- **weight** (*float*) – The weight of the hop addition
- **boil_time** (*float*) – The amount of time the hop is boiled
- **hop_type** (*float*) – The type of the hop being used
- **utilization_cls** (HopsUtilization) – The utilization class used for calculation
- **utilization_cls_kwargs** (*dict*) – The kwargs to initialize the utilization_cls object
- **units** (*str*) – The units

change_units()
Change units of the class from one type to the other

Returns Hop Addition in new unit type

Return type *HopAddition*

format()

get_alpha_acid_units()
Get Alpha Acid Units

Returns alpha acid units

Return type float

get_hops_weight(*sg*, *target_ibu*, *final_volume*, *percent_contribution*)
Get the Weight of Hops

Parameters

- **sg** (*float*) – The specific gravity of the wort
- **target_ibu** (*float*) – The target IBU
- **final_volume** (*float*) – The final volume of the wort
- **percent_contribution** (*float*) – The percent contribution of the hops to the total bitterness

Returns The weight of hops

Return type float

get_ibus(*sg*, *final_volume*)
Get the IBUs

Parameters

- **sg** (*float*) – The specific gravity of the wort
- **final_volume** (*float*) – The final volume of the wort

Returns The IBUs of the wort

Return type float

set_units(*units*)
Set the units and unit types

Parameters **units** (*str*) – The 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.

Percent Attenuation - The percentage the yeast is expected to attenuate the sugar in the beer to create alcohol.

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.

Parameters

- **name** (*str*) – The name of the recipe
- **grain_additions** (*list of GrainAddition objects*) – A list of Grain Additions
- **hop_additions** (*list of HopAddition objects*) – A list of Hop Additions
- **percent_brew_house_yield** (*float*) – The brew house yield
- **start_volume** (*float*) – The starting volume of the wort
- **final_volume** (*float*) – The final volume of the wort
- **units** (*str*) – The units

Raises

- **Exception** – If the units of any GrainAddition is not the same as the units of the Recipe
- **Exception** – If the units of any HopAddition is not the same as the units of the Recipe

change_units()
Change units of the class from one type to the other

Returns Recipe in new unit type

Return type *Recipe*

format()

get_boil_gravity()
Get the boil specific gravity

Returns The boil specific gravity

Return type float

get_boil_gravity_units()
Get the boil gravity units

Returns The boil gravity units

Return type float

get_brew_house_yield(*plato_actual*, *vol_actual*)
Get the Brew House Yield

Parameters

- **plato_actual** (*float*) – The actual degrees Plato
- **vol_actual** (*float*) – The actual volume collected from the kettle

Returns Brew House Yield

Rtyle float

get_bu_to_gu()
Get BU to GU Ratio

Returns Ratio of Bitterness Units to Original Gravity Units

Return type float

get_degrees_plato()
Get the degrees plato

Returns The degrees plato of the wort

Return type float

get_extract_weight()
Get the weight of the extract

Returns The weight of extract

Return type float

get_final_gravity()
Get the final specific gravity

Returns The final specific gravity

Return type float

get_final_gravity_units()
Get the final gravity units

Returns The final gravity units

Return type float

get_grain_add_cereal_weight(*grain_add*)
Get Grain Addition as Cereal

Parameters **grain_add** (GrainAddition) – The Grain Addition

Returns The weight of the grain as Cereal

Return type float

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*)
Get Grain Addition as DME

Parameters **grain_add** (GrainAddition) – The Grain Addition

Returns The weight of the grain as DME

Return type float

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*)
Get the Mash Water Volume

Parameters **liquor_to_grist_ratio** (*float*) – The Liquor to Grist Ratio

Returns The mash water volume

Return type float

get_original_gravity()
Get the original specific gravity

Returns The original specific gravity

Return type float

get_original_gravity_units()
Get the original gravity units

Returns The original gravity units

Return type float

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

Parameters **hop_add** (HopAddition) – The Hop Addition

Returns The percent the hops contributes to total ibus

Return type float

get_percent_malt_bill(*grain_add*)
Get Percent Malt Bill

Parameters **grain_add** (GrainAddition) – The Grain Addition

Returns The percent extract the addition adds to the bill

Return type float

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*)
Get Strike Water Temperature

Parameters

- **mash_temp** (*float*) – Mash Temperature
- **malt_temp** (*float*) – Malt Temperature
- **liquor_to_grist_ratio** (*float*) – The Liquor to Grist Ratio

Returns The strike water temperature

Return type float

get_total_dry_weight()
Get total DME weight

Returns The total weight of the DME

Return type float

get_total_grain_weight()
Get total Cereal weight

Returns The total weight of the Cereal

Return type float

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

Returns The total IBU for the Recipe

Return type float

get_total_points()
Get the total points of the recipe

Returns PPG or HWE depending on the units of the Recipe

Return type float

get_total_wort_color()
Get the Total Color of the Wort in SRM

Returns The total color of the wort in SRM

Return type float

get_total_wort_color_map()
Get a map of wort color by method

Returns A map of wort color in SRM and EBC by method (Morey, Daniels, and Mosher)

Return type dict

get_wort_color(*grain_add*)
Get the Wort Color in SRM

Parameters **grain_add** (GrainAddition) – The Grain Addition to calculate

Returns The SRM of the Grain Addition

Return type float

get_wort_color_mcu(*grain_add*)
Get the Wort Color in Malt Color Units

Parameters **grain_add** (GrainAddition) – The Grain Addition to calculate

Returns The MCU of the Grain Addition

Return type float

grain_lookup = {}

hop_lookup = {}

set_units (*units*)
Set the units and unit types

Parameters **units** (*str*) – The 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

Utilities API Docs

3.1 brew.utilities.abv

abv.**apparent_attenuation**(*original_extract*, *apparent_extract*)

abv.**real_attenuation**(*original_extract*, *real_extract*)

abv.**real_attenuation_from_apparent_extract**(*original_extract*, *apparent_extract*)

abv.**alcohol_by_volume_standard**(*og*, *fg*)
Alcohol by Volume Standard Calculation

Most brewing sites use this basic formula:

ABV = (og - fg) * 131.25

This equation was created before the computer age. It is easy to do by hand, and over time became the accepted formula for home brewers!

Variations on this equation which report within tenths of each other come from The Joy of Homebrewing Method by Charlie Papazian, Bee Lee's Method, Beer Advocate Method. Some variations use 131 instead of 131.25. The resulting difference is pretty minor.

Source:

- http://www.brewersfriend.com/2011/06/16/alcohol-by-volume-calculator-updated/
- http://www.brewmorebeer.com/calculate-percent-alcohol-in-beer/

ABV $= \frac{46.07\text{g/mol C2H6O}}{44.0095\text{g/mol CO2}} \times \frac{1.0}{0.7936} \times 100 \times (og - fg)$

abv.**alcohol_by_volume_alternative**(*og*, *fg*)
Alcohol by Volume Alternative Calculation

Alternate Formula:

A more complex equation which attempts to provide greater accuracy at higher gravities is:

ABV =(76.08 * (og - fg) / (1.775 - og)) * (fg / 0.794)

The alternate equation reports a higher ABV for higher gravity beers. This equation is just a different take on it. Scientists rarely agree when it comes to equations. There will probably be another equation for ABV down the road.

The complex formula, and variations on it come from Ritchie Products Ltd, (Zymurgy, Summer 1995, vol. 18, no. 2) -Michael L. Hall's article Brew by the Numbers: Add Up What's in Your Beer, and Designing Great Beers by Daniels.

Source: - http://www.brewersfriend.com/2011/06/16/alcohol-by-volume-calculator-updated/

abv.**alcohol_by_weight**(*abv*)
Alcohol by Weight from ABV

3.2 brew.utilities.color

color.**srm_to_ebc**(*srm*)
Convert SRM to EBC Color

color.**ebc_to_srm**(*ebc*)
Convert EBC to SRM Color

color.**calculate_mcu**(*grain_weight*, *beer_color*, *final_volume*, *units='imperial'*)
Calculate MCU from Grain

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

http://beersmith.com/blog/2008/04/29/beer-color-understanding-srm-lovibond-and-ebc/

color.**calculate_srm_mosher**(*mcu*)
Mosher Equation for SRM

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

color.**calculate_srm_daniels**(*mcu*)
Daniels Equation for SRM

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

color.**calculate_srm_daniels_power**(*mcu*)
Daniels Power Equation for SRM based on work by Druey

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

color.**calculate_srm_noonan_power**(*mcu*)
Noonan Power Equation for SRM based on work by Druey

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

color.**calculate_srm_morey_hybrid**(*mcu*)
A hybrid approach used by Morey for SRM.

Assumptions:

1. SRM is approximately equal to MCU for values from 0 to 10.
2. Homebrew is generally darker than commercial beer.
3. Base on the previous qualitative postulate, I assumed that Ray Daniels' predicted relationship exists for beers with color greater than 10.
4. Since Mosher's equation predicts darker color than Daniels' model for values of MCU greater than 37, I assumed that Mosher's approximation governed beer color for all values more than 37 MCUs.
5. Difference in color for beers greater than 40 SRM are essentially impossible to detect visually; therefore, I limited the analysis to SRM of 50 and less.

http://babblehomebrewers.com/attachments/article/61/beercolor.pdf

color.**calculate_srm_morey**(*mcu*)
Morey Equation for SRM

http://www.morebeer.com/brewingtechniques/beerslaw/morey.html

grain_weight - in lbs or kg beer_color - in deg Lovibond final_volume - in gal or liters

http://beersmith.com/blog/2008/04/29/beer-color-understanding-srm-lovibond-and-ebc/

color.**calculate_srm**(*mcu*)
General srm calculation uses the Morey Power Equation

color.**lovibond_to_srm**(*lovibond*)
Convert deg Lovibond to SRM https://en.wikipedia.org/wiki/Standard_Reference_Method

color.**srm_to_lovibond**(*srm*)
Convert SRM to deg Lovibond https://en.wikipedia.org/wiki/Standard_Reference_Method

color.**srm_to_a430**(*srm*, *dilution=1.0*)
Get attenuation at A430 from SRM and dilution https://en.wikipedia.org/wiki/Standard_Reference_Method

color.**ebc_to_a430**(*ebc*, *dilution=1.0*)
Get attenuation at A430 from EBC and dilution https://en.wikipedia.org/wiki/Standard_Reference_Method

3.3 brew.utilities.hops

class brew.utilities.hops.**HopsUtilization**(*hop_addition*, *units='imperial'*)
http://www.boondocks-brewing.com/hops

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

classmethod format_utilization_table()
Percent Alpha Acid Utilization - Boil Time vs Wort Original Gravity

Source: http://www.realbeer.com/hops/research.html

get_ibus(*sg*, *final_volume*)

classmethod get_percent_utilization(*sg*, *boil_time*)

classmethod get_utilization_table(*gravity_list*, *boil_time_list*, *sig=3*)

set_units(*units*)

class brew.utilities.hops.**HopsUtilizationJackieRager**(*hop_addition*, *units='imperial'*)
Jackie Rager

Best for extract and partial mash brewing.

Source: http://www.rooftopbrew.net/ibu.php

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

format_utilization_table()
Percent Alpha Acid Utilization - Boil Time vs Wort Original Gravity

Source: http://www.realbeer.com/hops/research.html

classmethod get_c_gravity(*sg*)
Cgravity is a constant to adjust the boil size when dealing with specific gravity greater than 1.050 in the calculation of IBUs.

get_ibus(*sg*, *final_volume*)

classmethod get_percent_utilization(*sg*, *boil_time*)

get_utilization_table(*gravity_list*, *boil_time_list*, *sig=3*)

set_units(*units*)

class brew.utilities.hops.**HopsUtilizationGlennTinseth**(*hop_addition*, *units='imperial'*)
Glenn Tinseth

Best for all grain brewing.

Source: http://www.realbeer.com/hops/research.html Source: http://www.rooftopbrew.net/ibu.php

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

format_utilization_table()
Percent Alpha Acid Utilization - Boil Time vs Wort Original Gravity

Source: http://www.realbeer.com/hops/research.html

classmethod get_bigness_factor(*sg*)

classmethod get_boil_time_factor(*boil_time*)

get_ibus(*sg*, *final_volume*)

classmethod get_percent_utilization(*sg*, *boil_time*)
The Bigness factor accounts for reduced utilization due to higher wort gravities. Use an average gravity value for the entire boil to account for changes in the wort volume.

Bigness factor = 1.65 * 0.000125^(wort gravity - 1)

The Boil Time factor accounts for the change in utilization due to boil time:

Boil Time factor = (1 - e^(-0.04 * time in mins)) / 4.15

Source: http://www.realbeer.com/hops/research.html

get_utilization_table(*gravity_list*, *boil_time_list*, *sig=3*)

set_units(*units*)

3.4 brew.utilities.malt

malt.**dry_to_liquid_malt_weight**(*malt*)
DME to LME Weight

Source: http://www.weekendbrewer.com/brewingformulas.htm

malt.**liquid_to_dry_malt_weight**(*malt*)
LME to DME Weight

Source: http://www.weekendbrewer.com/brewingformulas.htm

malt.**grain_to_liquid_malt_weight**(*grain*)
Grain to LME Weight

Source: http://www.weekendbrewer.com/brewingformulas.htm

malt.**liquid_malt_to_grain_weight**(*malt*)
LME to Grain Weight

malt.**dry_malt_to_grain_weight**(*malt*)
DME to Grain Weight

malt.**grain_to_dry_malt_weight**(*malt*)
Grain to DME Weight

malt.**specialty_grain_to_liquid_malt_weight**(*grain*)
Specialty Grain to LME Weight

Source: http://www.weekendbrewer.com/brewingformulas.htm

malt.**liquid_malt_to_specialty_grain_weight**(*malt*)
LME to Specialty Grain Weight

malt.**fine_grind_to_coarse_grind**(*fine_grind*, *fc_diff=0.017*)
Fine Grind to Coarse Grind Percentage

fine_grind A percentage from the malt bill

fc_diff The F/C difference percentage from the malt bill

malt.**coarse_grind_to_fine_grind**(*coarse_grind*, *fc_diff=0.017*)
Coarse Grind to Fine Grind Percentage

coarse_grind A percentage from the malt bill

fc_diff The F/C difference percentage from the malt bill

malt.**dry_basis_to_as_is_basis**(*dry_basis*, *moisture_content=0.04*)
Dry Basis to As-Is Basis Percentage

dry_basis A percentage from the malt bill in decimal form

moisture_content Apercentage of moisture content in finished malt in decimal form

malt.**as_is_basis_to_dry_basis**(*as_is*, *moisture_content=0.04*)
As-Is Basis to Dry Basis Percentage

asi_is A percentage from the malt bill in decimal form

moisture_content Apercentage of moisture content in finished malt in decimal form

malt.**sg_from_dry_basis**(*dbcg*, *moisture_content=0.04*, *moisture_correction=0.0*, *brew_house_efficiency=0.9*)
Specific Gravity from Dry Basis Percentage

dbcg Dry Basis Coarse Grain in decimal form

moisture_content Apercentage of moisture content in finished malt in decimal form

moisture_correction A percentage correction in decimal form

brew_house_efficiency The efficiency in decimal form

Returns: Specific Gravity available from Malt

malt.**plato_from_dry_basis**(*dbcg*, *moisture_content=0.04*, *moisture_correction=0.0*, *brew_house_efficiency=0.9*)
Degrees Plato from Dry Basis Percentage

dbcg Dry Basis Coars Grain in decimal form

moisture_content A percentage of moisture content in finished malt in decimal form

moisture_correction A percentage correction in decimal form

brew_house_efficiency The efficiency in decimal form

Returns: Degrees Plato available from Malt

malt.**basis_to_hwe**(*basis_percentage*)
Basis Percentage to Hot Water Extract

basis_percentage decimal form

Return Hot Water Extract as Ldeg/kg, dry basis

Ldeg/kg means how many litres of wort with a specific gravity of 1.001 you could produce from a kilogram of the fermentable

For example, if you had a kilogram of sucrose, you could make up 386 litres of wort with a specific gravity of 1.001.

`malt.`**`hwe_to_basis`**(*hwe*)
Hot Water Extract to Basis Percentage

`malt.`**`ppg_to_hwe`**(*ppg*)
Points Per Gallon to Hot Water Extract

`malt.`**`hwe_to_ppg`**(*hwe*)
Hot Water Extract to Points Per Gallon

3.5 brew.utilities.sugar

`sugar.`**`sg_to_gu`**(*sg*)
Specific Gravity to Gravity Units

`sugar.`**`gu_to_sg`**(*gu*)
Gravity Units to Specific Gravity

`sugar.`**`plato_to_sg`**(*deg_plato*)
Degrees Plato to Specific Gravity

Specific Gravity (S.G.) S.G. is the density of a liquid or solid compared to that of water. The simple formula for S.G. is:

S.G. = 1 + 0.004 x Plato

The more precise calculation of S.G. is:

S.G. = [(Plato) / (258.6 - (Plato/258.2 x 227.1))] + 1

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

`sugar.`**`sg_to_plato`**(*sg*)
Specific Gravity to Degrees Plato

Degrees Plato is the weight of the extract in a 100gram solution at 64 degrees Fahrenheit.

Plato = [(S.G. - 1) x 1000] / 4

The more precise calculation of Plato is:

Plato = -616.868 + 1111.14 * sg - 630.272 * sg ** 2 + 135.997 * sg ** 3

Source: http://www.brewersfriend.com/2012/10/31/on-the-relationship-between-plato-and-specific-gravity/

`sugar.`**`brix_to_sg`**(*brix*)
Degrees Brix to Specific Gravity

Source: http://www.brewersfriend.com/brix-converter/

`sugar.`**`sg_to_brix`**(*sg*)
Specific Gravity to Degrees Brix

Source: http://en.wikipedia.org/wiki/Brix http://www.brewersfriend.com/brix-converter/

sugar.**brix_to_plato**(*brix*)
Degrees Brix to Degrees Plato

The difference between the degBx and degP as calculated from the respective polynomials is:

degP - degBx = (((-2.81615*sg + 8.79724)*sg - 9.1626)*sg + 3.18213)

The difference is generally less than +/-0.0005 degBx or degP with the exception being for weak solutions.

https://en.wikipedia.org/wiki/Brix

sugar.**plato_to_brix**(*plato*)
Degrees Plato to Degrees Brix

sugar.**apparent_extract_to_real_extract**(*original_extract*, *apparent_extract*)
Apparent Extract to Real Extract in degrees Plato

Formula from Balling De Clerck, Jean, A Textbook Of Brewing, Chapman & Hall Ltd., 1958

sugar.**hydrometer_adjustment**(*sg*, *temp*, *units='imperial'*)
Adjust the Hydrometer if the temperature deviates from 59degF.

http://hbd.org/brewery/library/HydromCorr0992.html

The correction formula is from Lyons (1992), who used the following formula to fit data from the Handbook of Chemistry and Physics (CRC):

Correction(@59F) = 1.313454 - 0.132674*T + 2.057793e-3*T2 - 2.627634e-6*T**3** where T is in degrees F.

Sources: http://www.topdownbrew.com/SGCorrection.html http://hbd.org/brewery/library/HydromCorr0992.html http://www.brewersfriend.com/hydrometer-temp/ http://www.primetab.com/formulas.html

sugar.**refractometer_adjustment**(*og*, *fg*)
Adjust the Refractometer for the presence of alcohol.

NOTE: This calculation assumes using Brix or Plato, so the input will be converted from SG to Plato and then converted back.

Returns: Final Gravity

Sources: http://seanterrill.com/2011/04/07/refractometer-fg-results/

3.6 brew.utilities.temperature

temperature.**fahrenheit_to_celsius**(*temp*)
Convert degrees Fahrenheit to degrees Celsius

temperature.**celsius_to_fahrenheit**(*temp*)
Convert degrees Celsius to degrees Fahrenheit

3.7 brew.utilities.yeast

class brew.utilities.yeast.**YeastModel**(*method*, *units='imperial'*)

METHOD_TO_GROWTH_ADJ = {'shaking': 0.0, 'stir plate': 0.0, 'no agitation': 0.0}

get_growth_rate(*inoculation_rate*)

get_inoculation_rate(*growth_rate*)

get_resulting_pitch_rate(*starter_cell_count*, *original_gravity=1.036*, *final_volume=5.0*)

get_starter_volume(*available_cells*, *starter_volume=0.5283443537159779*, *original_gravity=1.036*)
Calculate the number of cells given a stater volume and gravity

get_viability(*days_since_manufacture*)
Yeast viability drops 21% each month or 0.7% per day from the date of manufacture. Assume linear change.

get_yeast_pitch_rate(*original_gravity=1.05*, *final_volume=5.0*, *target_pitch_rate=1.42*, *yeast_type='liquid'*, *cells_per_pack=100*, *num_packs=1*, *days_since_manufacture=30*)
Determine yeast pitch rate

original_gravity - specific gravity of original beer final_volume - volume of the batch post fermentation target_pitch_rate - million cells / (ml * degP) yeast_type - liquid, dry cells_per_pack - Billions of cells num_packs - how many in units days_since_manufacture - the older the yeast the less viable units - imperial, metric

Yeast Viability: lose 20% viability / month or 0.66% / day

Imperial: B / Gal / GU Metric: M / ml / Plato

Sources: - http://beersmith.com/blog/2011/01/10/yeast-starters-for-home-brewing-beer-part-2/

set_units(*units*)

class brew.utilities.yeast.**KaiserYeastModel**(*method='stir plate'*, *units='imperial'*)
Kaiser Yeast Model

Only works for Stir Plage Growth

Sources:

- http://braukaiser.com/blog/blog/2012/11/03/estimating-yeast-growth/

METHOD_TO_GROWTH_ADJ = {'stir plate': 0.0}

get_growth_rate(*initial_cells*)
initial_cells - Billion / gram extract (B/g)

get_inoculation_rate(*growth_rate*)

get_resulting_pitch_rate(*starter_cell_count*, *original_gravity=1.036*, *final_volume=5.0*)

get_starter_volume(*available_cells*, *starter_volume=0.5283443537159779*, *original_gravity=1.036*)
Calculate the number of cells given a stater volume and gravity

get_viability(*days_since_manufacture*)
Yeast viability drops 21% each month or 0.7% per day from the date of manufacture. Assume linear change.

get_yeast_pitch_rate(*original_gravity=1.05*, *final_volume=5.0*, *target_pitch_rate=1.42*, *yeast_type='liquid'*, *cells_per_pack=100*, *num_packs=1*, *days_since_manufacture=30*)
Determine yeast pitch rate

original_gravity - specific gravity of original beer final_volume - volume of the batch post fermentation target_pitch_rate - million cells / (ml * degP) yeast_type - liquid, dry cells_per_pack - Billions of cells num_packs - how many in units days_since_manufacture - the older the yeast the less viable units - imperial, metric

Yeast Viability: lose 20% viability / month or 0.66% / day

Imperial: B / Gal / GU Metric: M / ml / Plato

Sources: - http://beersmith.com/blog/2011/01/10/yeast-starters-for-home-brewing-beer-part-2/

set_units(*units*)

class brew.utilities.yeast.**WhiteYeastModel**(*method='no agitation'*, *units='imperial'*)

Sources:

- http://www.brewersfriend.com/yeast-pitch-rate-and-starter-calculator/
- White, Chris, and Jamil Zainasheff. Yeast: The Practical Guide to Beer Fermentation. Boulder, CO: Brewers Publications, 2010. 139-44. Print.

INOCULATION_CONST = [-0.999499, 12.547938, -0.459486]

METHOD_TO_GROWTH_ADJ = {'shaking': 0.5, 'stir plate': 1.0, 'no agitation': 0.0}

get_growth_rate(*inoculation_rate*)

initial_cells - Billion / gram extract (B/g)

G = (12.54793776 * x^-0.4594858324) - 0.9994994906

get_inoculation_rate(*growth_rate*)

get_resulting_pitch_rate(*starter_cell_count*, *original_gravity=1.036*, *final_volume=5.0*)

get_starter_volume(*available_cells*, *starter_volume=0.5283443537159779*, *original_gravity=1.036*)

Calculate the number of cells given a stater volume and gravity

get_viability(*days_since_manufacture*)

Yeast viability drops 21% each month or 0.7% per day from the date of manufacture. Assume linear change.

get_yeast_pitch_rate(*original_gravity=1.05*, *final_volume=5.0*, *target_pitch_rate=1.42*, *yeast_type='liquid'*, *cells_per_pack=100*, *num_packs=1*, *days_since_manufacture=30*)

Determine yeast pitch rate

original_gravity - specific gravity of original beer final_volume - volume of the batch post fermentation target_pitch_rate - million cells / (ml * degP) yeast_type - liquid, dry cells_per_pack - Billions of cells num_packs - how many in units days_since_manufacture - the older the yeast the less viable units - imperial, metric

Yeast Viability: lose 20% viability / month or 0.66% / day

Imperial: B / Gal / GU Metric: M / ml / Plato

Sources: - http://beersmith.com/blog/2011/01/10/yeast-starters-for-home-brewing-beer-part-2/

set_units(*units*)

CHAPTER 4

Appendix

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

A

B

C

D

E

F

G

V

W

Y