Version 2, comments to karin.treyer@psi.ch, Jan 2025

Brightway version 2 Cheatsheet for Beginners (not 2.5, but basic code often compatible)

Numbering = consecutive actions
a., b., = different ways of how to do this

> Everything around BW

Get a minimum understanding of the following:

- *Anaconda/Miniconda*: The engine to run the BW.
- *Python*: The language in which BW is written.
- *Jupyter*: Used to create and share notebooks which contain your LCA study code
- *Github*: Sharing code openly, collaborating
- *Activity Browser*: Graphical User Interface of BW
- *BW*: docs.brightway.dev // learn.brightway.dev // live.brightway.dev // try.brightway.dev
- *I need help*: check/post on stackoverflow.com/questions/tagged/brightway OR brightway.groups.io (subgroup beginners)

> Installing, opening, upgrading bw – in prompt window

Open your (ana-/mini-)conda prompt window – this is always the start of your work in bw!
«Installing» = creating a conda environment and attaching the bw package to it:

```
conda create -n yourenv brightway2 jupyterlab
```

Starting = opening bw: we want to open a jupyter notebook in the environment you want to work with:
open anaconda prompt
`(conda env list` for knowing which envs you have)

```
conda activate yourenv
jupyter lab or jupyter notebook)
```

Updating:

```
conda activate yourenv
conda update brightway2
```

> Starting to use bw – in jupyter notebook

1. «open bw» = open your jupyter notebook (see above)
2. Import all necessary packages:

Minimum is this:

```
import brightway2 as bw //import bw2io
as bi import bw2data as bd //import
bw2calc as bc
```

Often, on top: #data science

```
import numpy as np //import pandas as pd
```

#plotting `import matplotlib.pyplot as plt`

```
import seaborn as sns
```

> Working on projects – in jupyter notebook

`bd.projects.` press tab → shows you all functions for project handling. Choose one of them, and add ? → shows you how this function works

`list(bd.projects)` (which projects do I have)
`bd.projects.dir` (where does my current project sit)
`bd.projects.report()` (general information)
`bd.projects.current` (checking in which project you are)

Activate your project/create one

`a. bd.projects.set_current('myproject')` (activates myproject, and creates it first if it doesn't exist yet)
`b. bd.projects.create_project('myproject')` (creates project, but you remain in your current project)

`b.` ***Rename/Copy/Delete a project***

You need to copy your project if you want to rename it, and delete the old one.

`bd.projects.copy_projects('newname', switch=True)` (copies the current project, and if you add the «switch=True» it switches to this new copy.)
`bd.projects.delete_project('myproject', delete_dir=True)` (deletes myproject, and also deletes it from your computer directory)

> Import databases – biosphere, methods, ecoinvent

`bd.databases` (which databases do I have in my project)

a. biosphere, ecoinvent

```
bi.import_ecoinvent_release(‘versionnumber',
'systemmodel', 'ecoinvent User Name', 'ecoinvent
password')
```

System model = cutoff / apos / consequential / EN15804

b1. biosphere

`bi.bw2setup()` (creates the 'biosphere3' db and imports the LCIA methods)

b2. Any other database with spold2 format, e.g. ecoinvent

```
fp = r'C:\ecoinvent_3.9.1_cutoff_ecoSpold02\datasets'
```

(filepath example)

```
ei39imp = bi.SingleOutputEcospold2Importer(fp,
'ev391cutoff') (give your own name to the db)
ei39imp.apply_strategies()
ei39imp.statistics()
if len(list(ei39imp.unlinked) == 0):
    ei39imp.write_database()
```

> Import databases – own data / premise

Your own LCI from an excel sheet

```
imp = bi.ExcelImporter(r'C:\yourpath)
imp.apply_strategies()
imp.match_database('ev391cutoff', fields=('name',
'unit', 'location'))
imp.statistics()
```

Which are the unlinked flows? →`list(imp.unlinked)` *or*
`imp.write_excel() #(only_unlinked=True)`
Normally, you MUST find the error leading to the unliked flow. Usually it's typos, wrong geographies/databases etc.
If you KNOW that you don't need it, you can do this:
`imp.drop_unlinked(i_am_reckless=True)`
In the end, write the database
`imp.write_database()`
→ It is good practice to do few quick checks to see if the db is functional: length, do LCA calc (see next page of cheatsheet for this)

SimaPro/openLCA datasets – not painless yet

SP:https://gist.github.com/cmutel/963905e16bedbeffb40d2df005d0e7ae
oLCA:https://github.com/cyrillefrancois/openlca2bw

Premise: follow instructions on premise.readthedocs.io/

> Manage databases

Checking which dbs you have: `list(bd.databases)`
Delete: `del bd.databases['dbname']`
Copy:

```
original = bd.Database('mydb')
copy = original.copy('mydb_newname')
```

Rename: `original.rename('mydb_renamed')`

> Activity Browser – installation & opening

Create projects, import databases, look at the databases, create own datasets, do LCIA calculations – and many helpful things more!
Recommendation: Set up your own LCI in a spreadsheet (reproducibility, e.g. linking to original data&calculations) and import it. If database imports fails in AB, it may be easier to do the import via a jupyter notebook, because you can identify unlinked exchanges. But once you have the database(s), it may bemore human friendly to look at it in AB and do quick LCIA calcs there.
When you create an env in the prompt window, set up a project&import databases in jupyter notebook, all these will directly be displayed in AB (and vice versa)!

Installation (in prompt window) = creating a conda env and attaching the ab package to it:

```
conda create -n ab -c conda-forge activity-
browser
```

Opening AB: `conda activate ab`
`activity-browser`

> Checking the content of the dbs

```
bio = bd.Database('biosphere3')
ei =bd.Database('ev391cutoff')
m =bd.methods
```

Misc, applicable for all above (except sometimes m, i.e. replace as you want with bio, ei or m):

`len(bio) // type(bio)`

`act=ei.random()` #picking a random activity/biosphere flow/method

`act.as_dict()` #Looking at an activity/biosphere flow

Diving deeper: looking at all technosphere/biosphere exchanges of an activity:

`a. list(act.technosphere())` or `list(act.biosphere())` or `list(act.production())`

```
b. for e in act.exchanges():
        if e ['type'] == 'technosphere':
        print(e['name'], e['amount'], e['type'],
        e.get('location')
```

Searching:

Knowing for which fields I can search (categories):

in *ecoinvent*: `act = ei.random()` then `list(act.keys())`

In *biosphere*: `set(list(f['categories'] for f in bio))`

a. Search function:

```
ei.search('electricity')
bio.search('carbon dioxide', filter =
{'categories': 'urban', 'name': 'fossil'})
```

b. ***List comprehension***, examples for each database:

Biosphere:
```
co2 = [ flow for flow in bio
        if 'Carbon dioxide' in flow['name']
        and ',  fossil' not in flow['name']
        and flow['categories'] ==('soil',) ]
```

Ecoinvent:
```
coalDE = [a for a in ei
         if 'electricity production' in a['name']
        and 'coal' in a['name']
         and a['location'] == 'DE'    ]
```

Methods: `ilcd = [m for m in bd.methods if 'ILCD 2.0' in str(m) and 'LT' not in str(m)]`

→ Once you have that list, you can choose a list element (=specific LCIA method) with e.g. [1]. Then, you can explore that element, e.g.

```
bd.Method(ilcd).metadata
bd.Method(ilcd).metadata['unit']
```

You can also use the full list for LCIA calculations.

> Calculating LCIA results for 1 activity&1 method

1. Choose an activity and give it a variable name (act)
2. Choose one method and give it a variable name (ipcc)
3. `lca = bc.LCA({act: 1}, ipcc[0])` #act:1 is the FU
4. `lca.lci()` # Builds matrices, solves the system, generates an LCI matrix.
5. `lca.lcia()` # Characterization, i.e. the multiplication of the elements of the LCI matrix with characterization factors from the chosen method
6. `lca.score` #Returns the LCIA score

> Calculating LCIA results for multiple activities (functional units)/methods

1. Create a list with all activities you want («acts»)
2. Create a list with all methods you want («methods»)
3. Create a list of functional units:

```
FU = [{x:1} for x in acts]
4. bd.calculation_setups['setupname'] =
{'inv':FU, 'ia': methods}
5. mLCA =  bc.MultiLCA('setupname')
6. mLCA.results
7. #Showing results in a dataframe (import pandas as pd):
mLCAdf = pd.DataFrame(index = methods,
        columns = [(x['name'], x['location']) for
y in FU for x in y], data=mLCA.results.T)
8. #export to excel, e.g. for creating figures
df.to_excel('excelname.xlsx')
```

> Plotting results – using pandas dataframes

1. Create a results df as shown above
2. More human friendly labels: `labels_methods = { "('ILCD 2.0 2018 midpoint', 'climate change', 'climate change biogenic')": 'CC\nbio', addmoreasyouneed, }`

```
labels_act =  { "'electricity production,
wind, >3MW turbine, onshore' (kilowatt hour,
DE, None)": "wind", addmoreasyouneed, }
df = df.rename(columns = labels_act, index =
labels_methods)
```

3. Plotting

```
df.plot.bar(xlabel='Impact category',
ylabel='Impact score', figsize=(14,8))
plt.xticks(rotation=0)
```

#Normalisation:

```
df = (df.T/df.abs().max(axis=1)).T
```

You can also use seaborn or matplotlib for plotting.

> Contribution analysis

→ This has become much more intuitive in bw2.5. But we are in bw2, thus:

```
1.  from polyviz.utils import calculate_supply_chain
2.  Use the «acts» and «methods» list from above
3.  df = pd.DataFrame()
4.  for act in acts:
        for m in methods:
            print(act, m)
            data = calculate_supply_chain(act, m,
2, 1e-4)[0]
#choose down to which level you want to go, and the cutoff
            data = [[act['name'], act['location'],
'-'.join(m)] + d for d in data]
           df = pd.concat([df,
pd.DataFrame(data)], axis=0, ignore_index=True)
df.columns = ['activity', 'location_activity',
'lcia_method', 'level', 'contribution share',
'absolute score', 'amount_in_dataset',
'activity_contributing', 'location_contributing',
'unit']
df.to_excel('contribution.xlsx')
```

> Project management: Reproducibility & collaboration

E.g. when you want to freeze the status of a project (e.g. after manuscript submission); or when you want to collaborate: Share env and project with all databases

Saving your ***env*** as .yaml file: `conda list -n envname -export C:\yourpath\envname_20240401.yml`

Recreating the env: `conda env create -f C:\yourpath\envname_20240401.yml`

Backup existing project:

`bi.backup.backup_project_directory(project ='yourname')` (saved in your C:\Users folder)

Restore project from backup:

`bi.backup.restore_project_directory(fp =yourpath, project_name = 'yourprojectname'`

> Small difference: Method/Database, methods/databases

Method/Database: If you want to pick elements and work with them, you need to work with the object. Mind the different brackets.

methods/databases: If you want to use a function, get list of names, you use this

```
eu = bw.Method(('ReCiPe 2016 v1.03, midpoint
(E)', 'eutrophication: marine', 'marine
eutrophication potential (MEP)')
bw.methods.random()
ei = bw.Database('ev391cutoff')
del bw.databases['ev391cutoff']
```