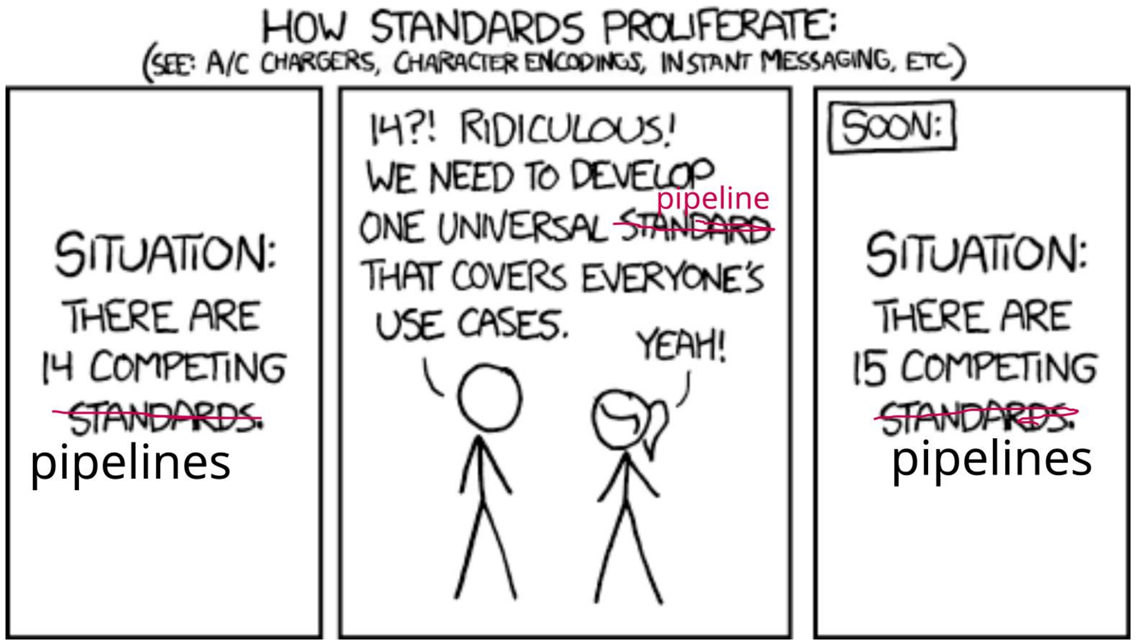
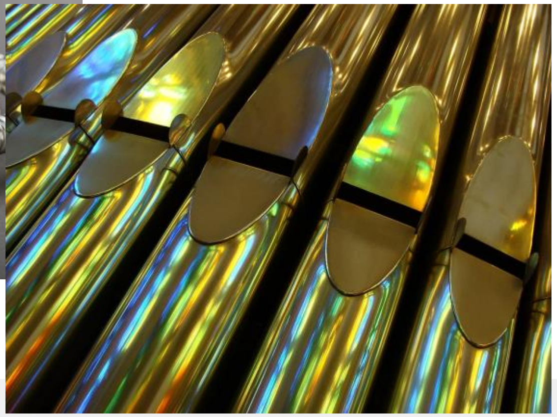
pdChemChain – Interactive Links for Chemistry Data Processing

Pipelining Tools


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING ~~STANDARDS.~~ pipelines
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL ~~STANDARD~~ pipeline THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING ~~STANDARDS.~~ pipelines

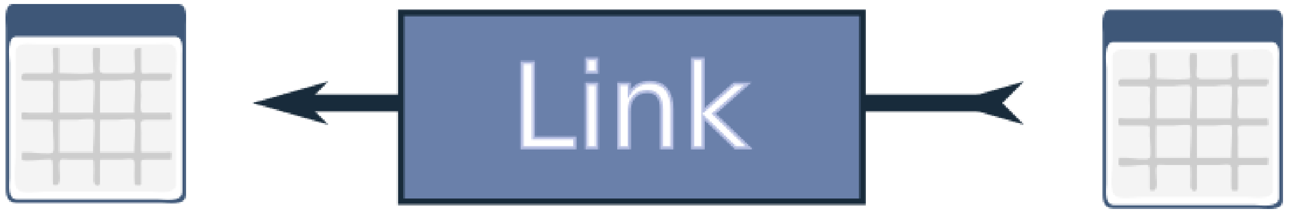
- 100’s of toolkits listed at https://github.com/pditommaso/awesome-pipeline
- But, I didn’t find one that suited me exactly right :-P



Images from Pixabay

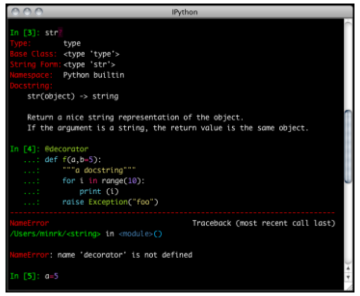
pdChemChain API 1: Pandas In - Pandas Out

- Something easy to set up in interactive notebooks
  - -> Short Code-Test cycles
  - -> Exploratory data analysis
  - -> Stepwise Application and Test
  - -> Interactive inspection of output



```
link = SmilesToMol()
df_out = link(df)
```

jupyter

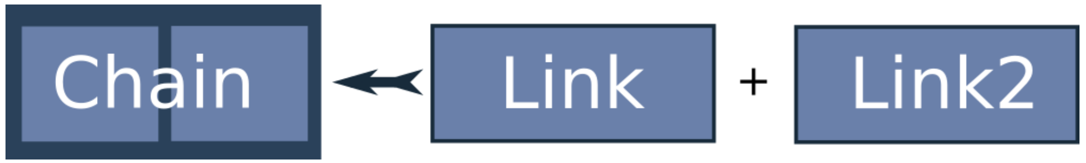


```
In [3]: str?
Type:        type
Base Class:  <type 'type'>
String Form: <type 'str'>
Namespace:   Python builtin
Docstring:
    str(object) -> string

    Return a nice string representation of the object.
    If the argument is a string, the return value is the same object.

In [4]: @decorator
   ...: def f(a,b=5):
   ...:     """a docstring"""
   ...:     for i in range(10):
   ...:         print (i)
   ...:     raise Exception("foo")
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/Users/minrk/<string> in <module>()

NameError: name 'decorator' is not defined

In [5]: a=5
```

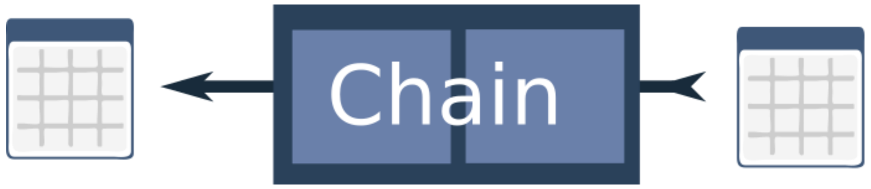
pdChemChain API 2: A Chain is Also a Link

- Summing Links gives a Chain
- Chains are also Links

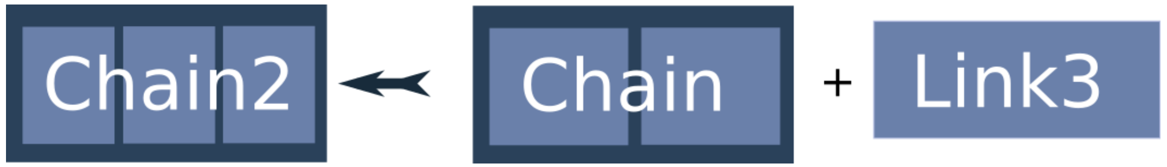
```
chain = link + link2
```



```
df_out = chain(df_in)
```



```
chain2 = chain + link3
```



```
chain2 = link + link2 + link3
chain2 = sum([link, link2, link3])
chain2 = Chain(links=[link, link2,
link3])
```

But … Python Notebooks are the Fast Food of Software Development! API 3: Auto-Documenting and Auto-Configurable

- Auto-Documenting and Auto-Configurable

```
from pdchemchain import Link
from pdchemchain.links import NullLink

link = NullLink(name='Demo link1')
parm_dict = link.get_params()

link_clone = Link.from_params(parm_dict)
link_clone

>>> NullLink(name='Demo link1')
```

- Save/Edit/Load from json/yaml and reuse from command-line for deployment

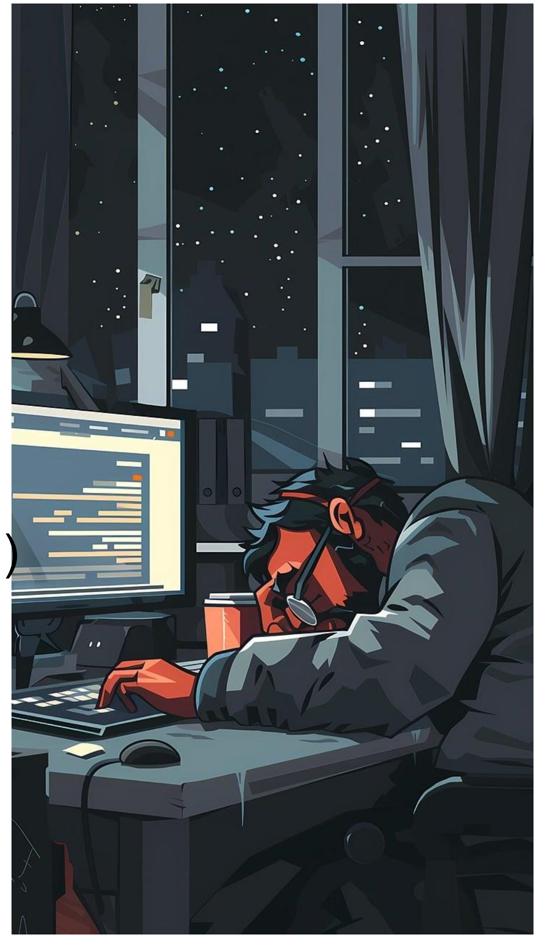
```
$ pdchemchain run --in_file mydata.csv --out_file dataout.csv mypipeline.yml
```

Images from Pixabay

Efficient Subclassing for Customization

- Low overhead to create new links

```
@dataclass
class HeavyAtomCount(RowLink):
        in_column: InColumnName = "ROMol"
        out_column: str = "HeavyAtomCount"

        def _row_apply(self, row: pd.Series) -> pd.Series:
                mol = row[self.in_column]
                row[self.out_column] = Descriptors.HeavyAtomCount(mol)
                return row
```

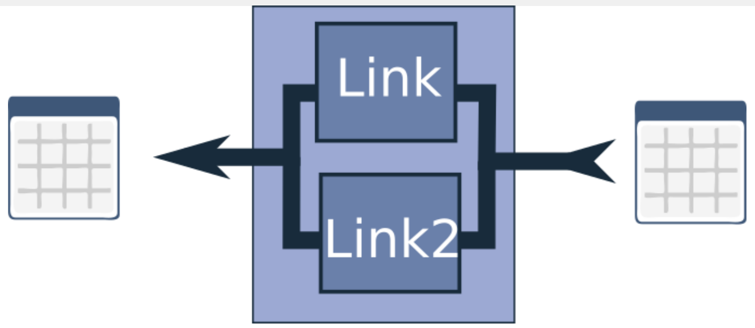


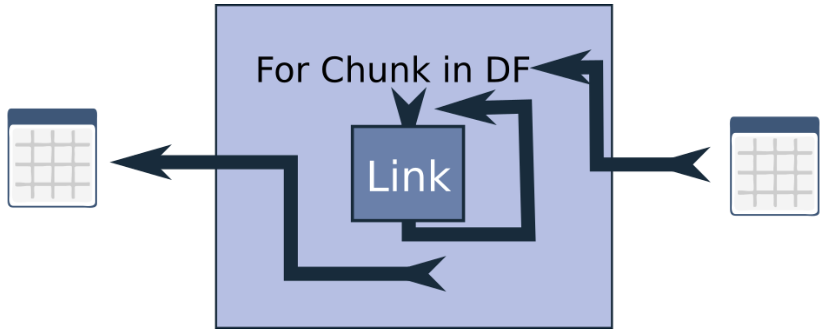
Images from Pixabay

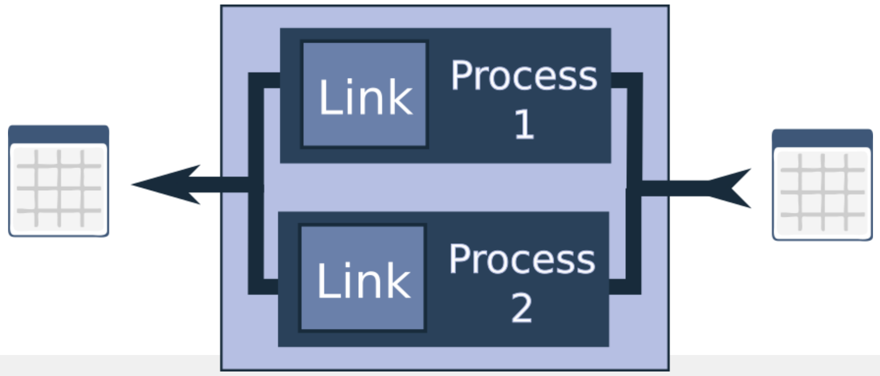
Other Features

- Error handling of rows
- Growing link library
- Advanced routing and high performance via compound links
- Auto-populated "toolbox" for overview

- Open Source on GitHub
  - https://github.com/EBjerrum/pdchemchain
  - contributions welcome
- num_pipeline_tools += 1
- **Thank you for your Attention!**




For Chunk in DF
Link


Link
Process 1
Link
Process 2