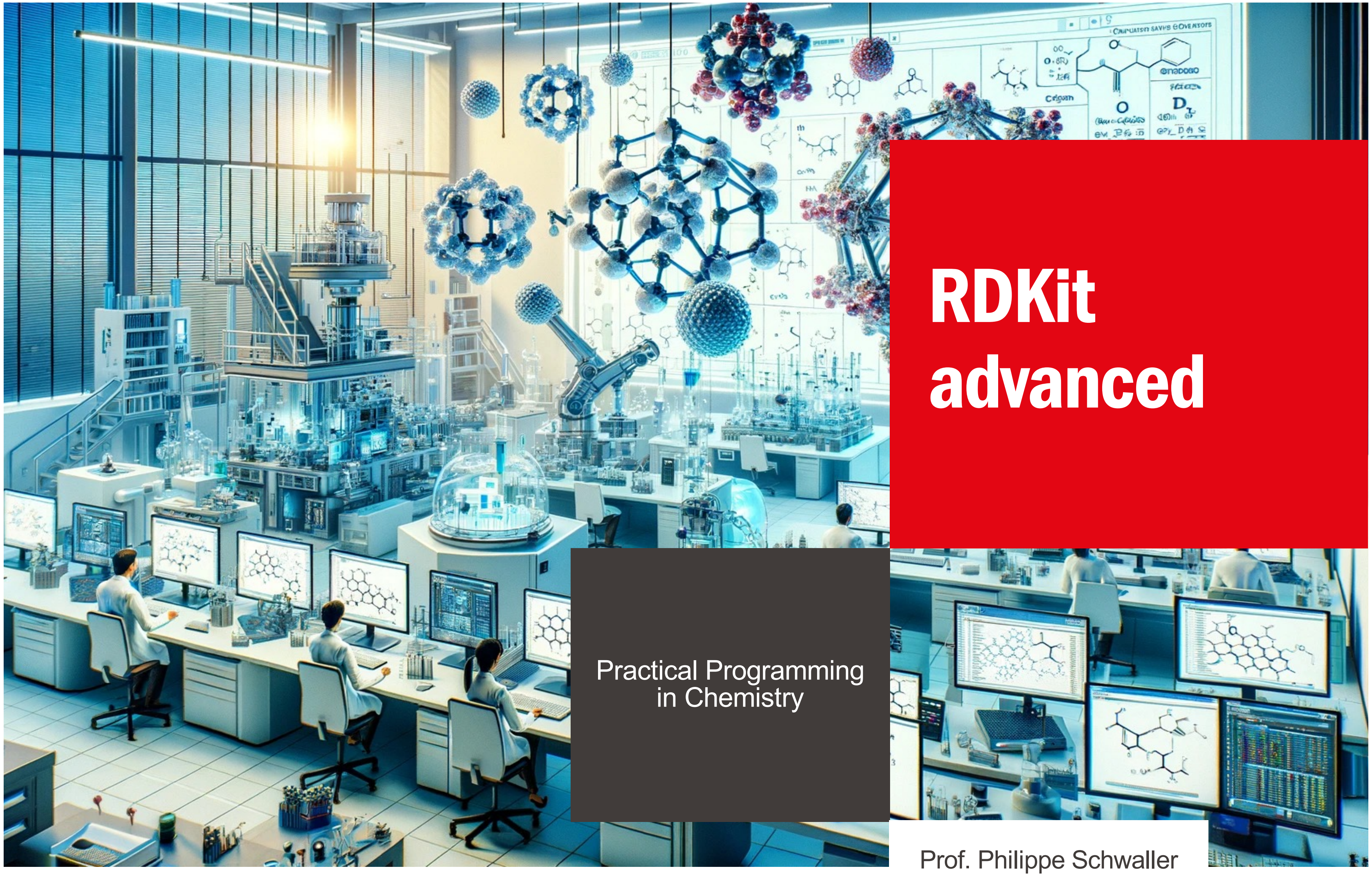
EPFL



RDKit advanced

Practical Programming in Chemistry

Prof. Philippe Schwaller

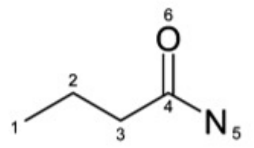
Why fingerprints?

- Molecular structures (graphs) by themselves are **hard to compare**
- If you assume that molecules with **similar substructures have similar properties** → you want to turn molecules into substructure vectors
- As soon as you have a vector (bit/int/float), it is easy to compute a similarity between two molecules

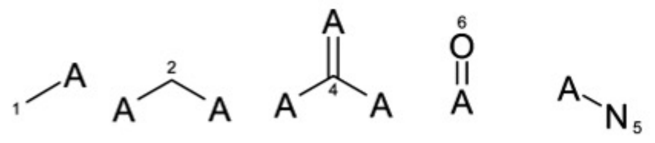
- Use cases:
  - Searching for similar molecules / clustering molecules into classes / …
  - Input to machine learning models

How the ECPF/Morgan fingerprint is generated? (so, you have an intuition, it's a one-liner in RDKit)

Simplified case: radius 1



1. Get radius 1 substructures

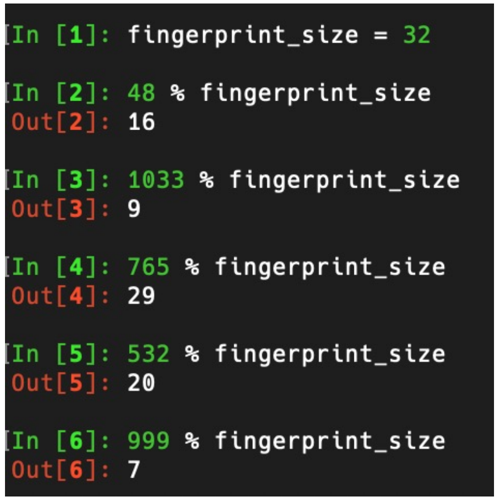


48	1033	765	532	999
16	9	29	20	7

2. Assign integer based on hash function (mathematical blender) on atomic properties.

3. Define fingerprint size: 32 and calculate the modulo (divide by 32 and take rest)

Note: all numbers between 0 and 31.



```
In [1]: fingerprint_size = 32

In [2]: 48 % fingerprint_size
Out[2]: 16

In [3]: 1033 % fingerprint_size
Out[3]: 9

In [4]: 765 % fingerprint_size
Out[4]: 29

In [5]: 532 % fingerprint_size
Out[5]: 20

In [6]: 999 % fingerprint_size
Out[6]: 7
```

4. Convert to size 32 bit vector by assigning a one to the numbers in the list.

[0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0] → molecular fingerprint

Based on them we can structurally compare molecules.

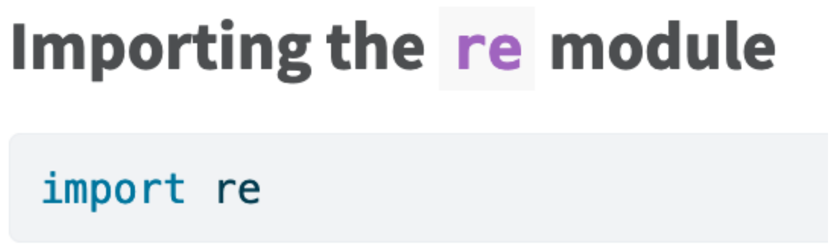
This lecture

- Regular expressions (regex)
- More RDKit functionality – yay!
- SMARTS and substructure search
- Chemical reactions

Regular expressions – regex

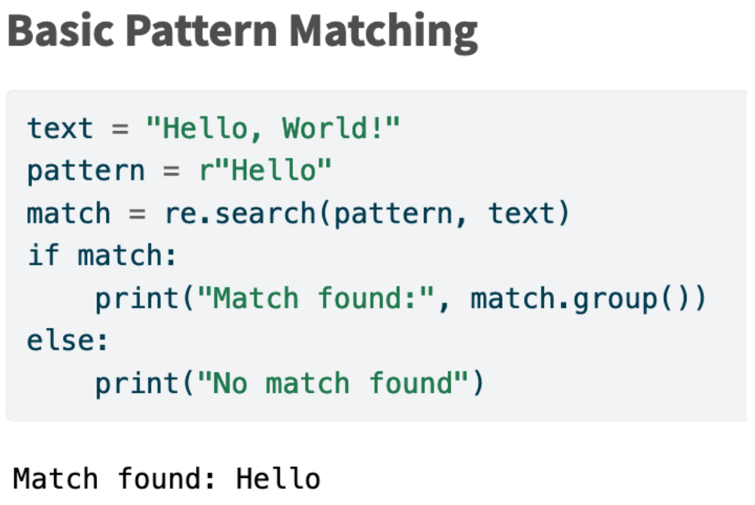
Regular expressions (regex) in Python

- powerful text patterns used for text matching, manipulation, and searching



Importing the re module

```python
import re
```



Basic Pattern Matching

```python
text = "Hello, World!"
pattern = r"Hello"
match = re.search(pattern, text)
if match:
    print("Match found:", match.group())
else:
    print("No match found")
```

```
Match found: Hello
```

- https://docs.python.org/3/howto/regex.html

Regex basics

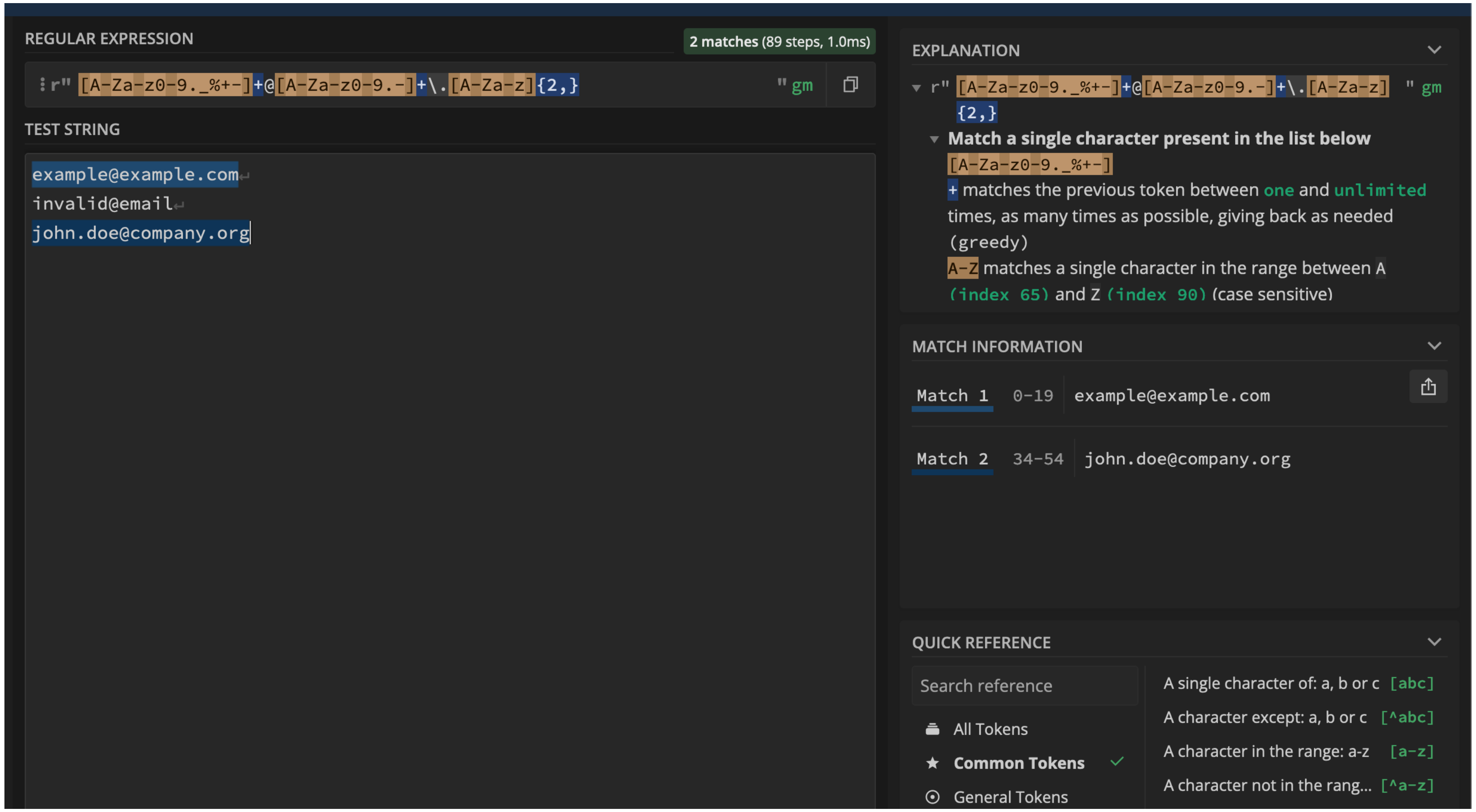
- `.` (dot): matches any character except a newline
- `\d` : matches a digit character
- `\w` : matches a word character (letter, digit, or underscore)
- `\s` : matches a whitespace character
- `^` : matches the start of a string
- `$` : matches the end of a string
- `[]` : matches any character inside the brackets
- `|` : matches either the expression before or after the `|`
- `*` : matches zero or more occurrences of the preceding pattern
- `+` : matches one or more occurrences of the preceding pattern
- `?` : matches zero or one occurrence of the preceding pattern

- `re.search` : searches for the first location where the pattern matches
- `re.match` : checks if the pattern matches at the beginning of the string
- `re.findall` : returns a list of all non-overlapping matches
- `re.split` : splits the string by the occurrences of the pattern
- `re.sub` : substitutes occurrences of the pattern with a replacement string

```python
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
email = "example@example.com"
if re.match(email_pattern, email):
    print("Valid email address")
else:
    print("Invalid email address")
```

```
Valid email address
```

Regex101 - https://regex101.com/



REGULAR EXPRESSION

2 matches (89 steps, 1.0ms)

```
r" [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,} " gm
```

TEST STRING

```
example@example.com
invalid@email
john.doe@company.org
```

EXPLANATION

r" [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,} " gm

Match a single character present in the list below [A-Za-z0-9._%+-]

+ matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)

A-Z matches a single character in the range between A (index 65) and Z (index 90) (case sensitive)

MATCH INFORMATION

```
Match 1   0-19   example@example.com
Match 2   34-54  john.doe@company.org
```

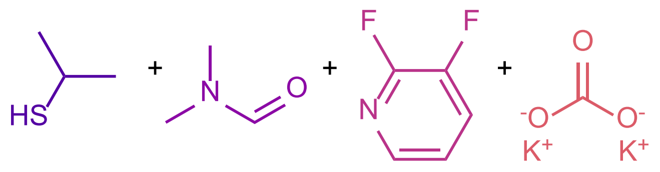
QUICK REFERENCE

A single character of: a, b or c	[abc]
A character except: a, b or c	[^abc]
A character in the range: a-z	[a-z]
A character not in the rang...	[^a-z]

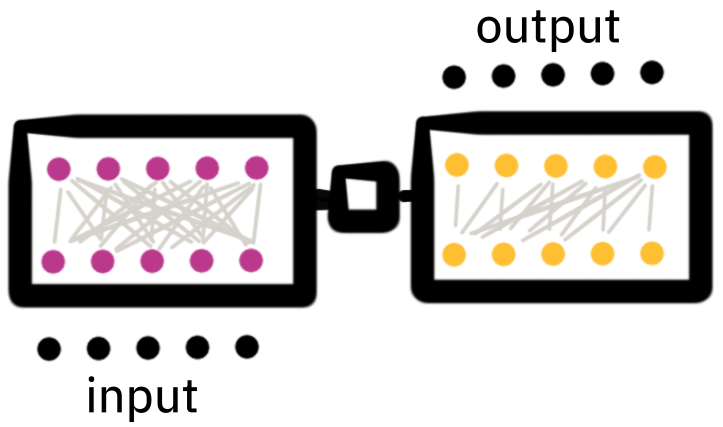
EPFL

Molecular Transformer – a language model in chemistry

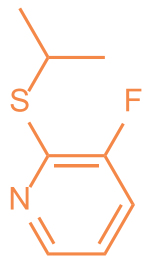
precursors



CC(C)S.CN(C)C=O.Fc1cccnc1F.O=C([O-])[O-].[K+].[K+]



products



CC(C)Sc1ncccc1F

- **No rules** integrated / no chemical knowledge
- Learning from examples (similar to translation models)
- **Accurate predictions** on unseen reactions (top on benchmarks, back then)
- Better than rule and graph-based approaches

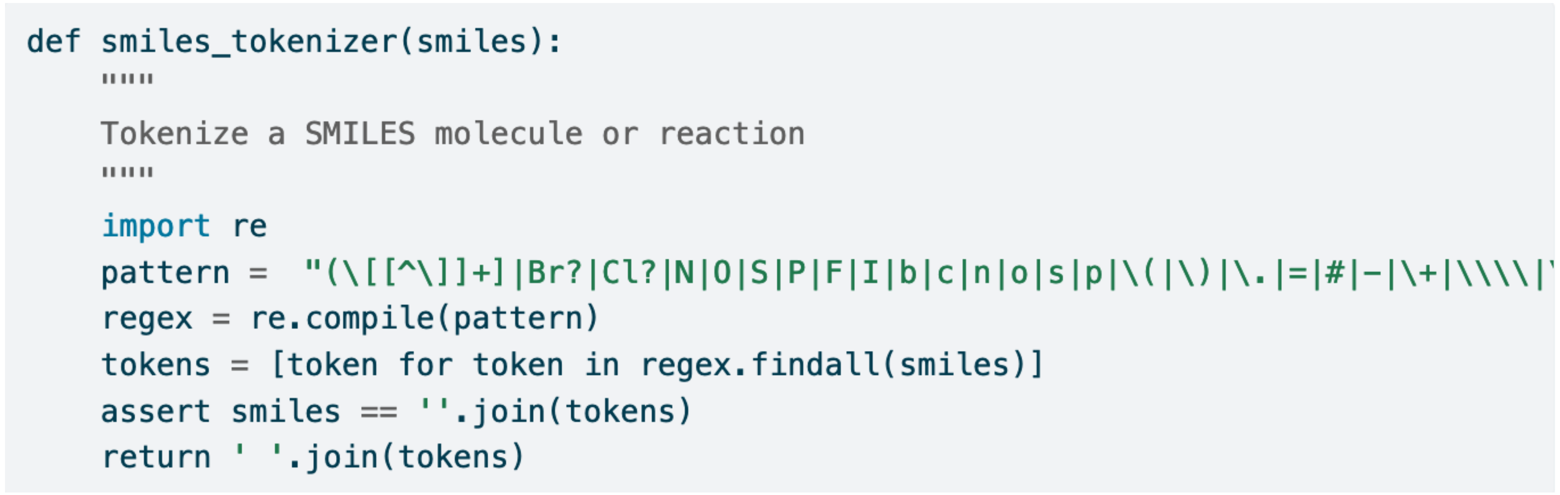
UNIVERSITY OF CAMBRIDGE

IBM **Research**

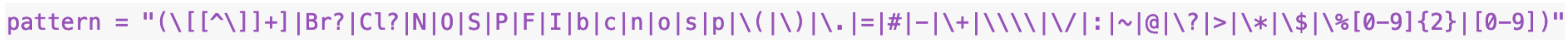
Schwaller et al., Molecular Transformer – A Model for Uncertainty-Calibrated Chemical Reaction Prediction. ACS Central Science, 2019

Example: SMILES tokenizer

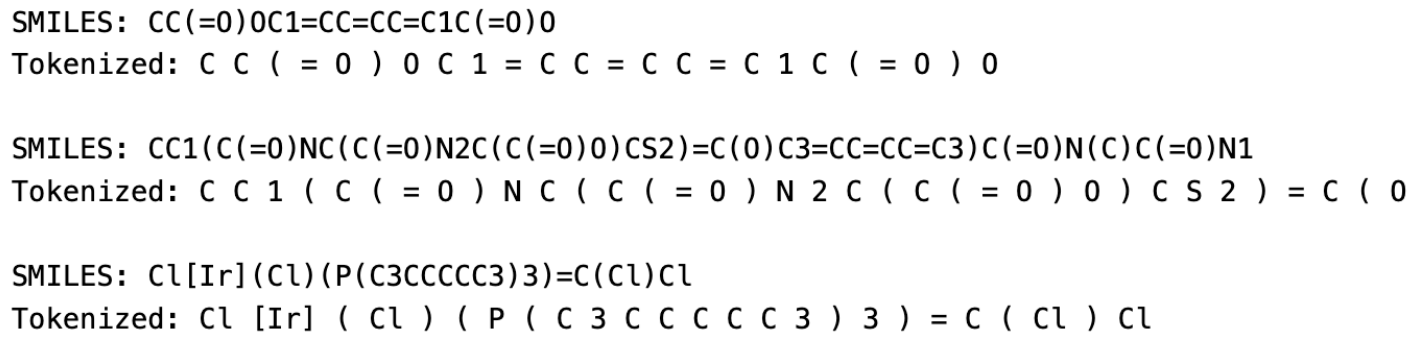
SMILES: CC(=O)OC1=CC=CC=C1C(=O)O $\rightarrow$ Tokenized: C C (= O) O C 1 = C C = C C = C 1 C (= O) O
"atom sequence"



```python
def smiles_tokenizer(smiles):
    """
    Tokenize a SMILES molecule or reaction
    """
    import re
    pattern =  "(\[[^\]]+]|Br?|Cl?|N|O|S|P|F|I|b|c|n|o|s|p|\(|\)|\.|=|#|-|\+|\\\\|
    regex = re.compile(pattern)
    tokens = [token for token in regex.findall(smiles)]
    assert smiles == ''.join(tokens)
    return ' '.join(tokens)
```



```
pattern = "(\[[^\]]+]|Br?|Cl?|N|O|S|P|F|I|b|c|n|o|s|p|\(|\)|\.|=|#|-|\+|\\\\|\/|:|~|@|\?|>|\*|\$|\%[0-9]{2}|[0-9])"
```



```
SMILES: CC(=O)OC1=CC=CC=C1C(=O)O
Tokenized: C C ( = O ) O C 1 = C C = C C = C 1 C ( = O ) O

SMILES: CC1(C(=O)NC(C(=O)N2C(C(=O)O)CS2)=C(O)C3=CC=CC=C3)C(=O)N(C)C(=O)N1
Tokenized: C C 1 ( C ( = O ) N C ( C ( = O ) N 2 C ( C ( = O ) O ) C S 2 ) = C ( O

SMILES: Cl[Ir](Cl)(P(C3CCCCC3)3)=C(Cl)Cl
Tokenized: Cl [Ir] ( Cl ) ( P ( C 3 C C C C C 3 ) 3 ) = C ( Cl ) Cl
```

Do regex work for SMILES search?

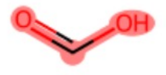
- NO.
- Let’s take carboxylic acids
  - Formic acid SMILES: O=CO
  - Acetic acid SMILES: CC(O)=O
  - Propionic acid SMILES: CCC(=O)O
- The carboxylic group is written 3 times differently.
- So, what can we do?

SMARTS – Regex for molecules

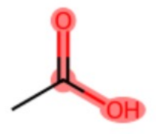
Regex for chemical structures are SMARTS

- **S**MILES **a**rbitrary **t**arget **s**pecification
- A Language for Describing Molecular Patterns

```
# Convert SMILES to RDKit molecules
mols = [Chem.MolFromSmiles(smi) for smi in smiles]

# Highlight carboxylic acid groups
pattern = Chem.MolFromSmarts('C(=O)O')

# Get the atoms that match the pattern for each molecule
matches = [mol.GetSubstructMatches(pattern) for mol in mols]
```



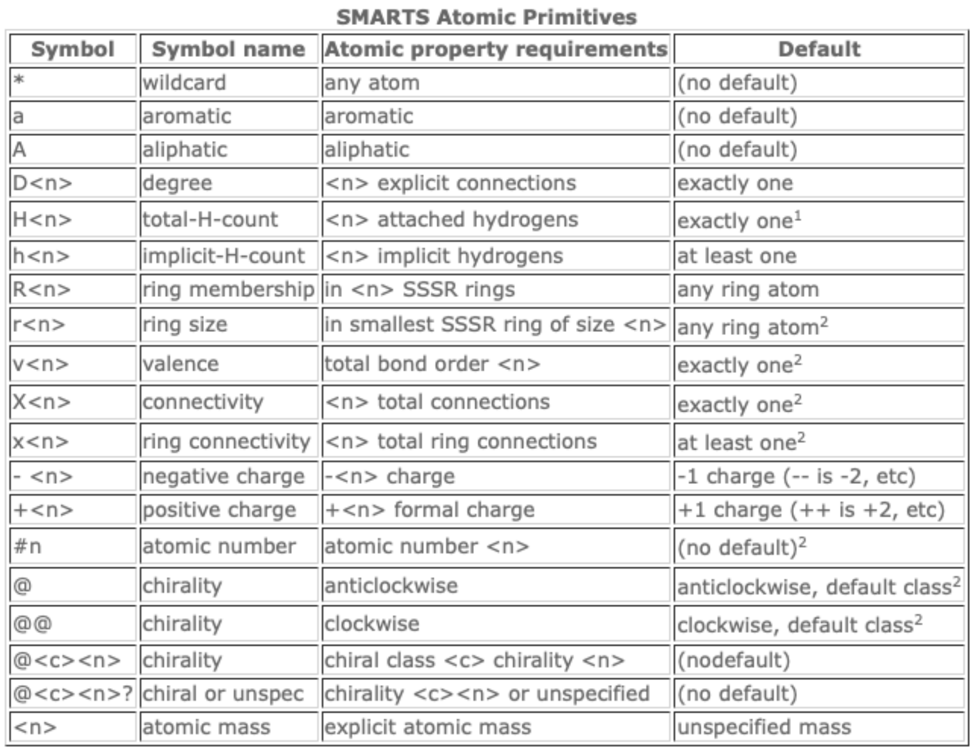
O=CO



CC(=O)O

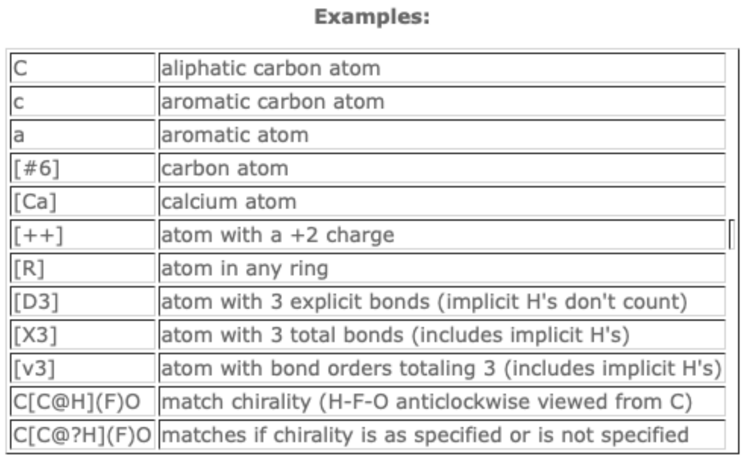
- Many more details on: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

SMARTS primitives and examples



SMARTS Atomic Primitives

Symbol	Symbol name	Atomic property requirements	Default
*	wildcard	any atom	(no default)
a	aromatic	aromatic	(no default)
A	aliphatic	aliphatic	(no default)
D<n>	degree	<n> explicit connections	exactly one
H<n>	total-H-count	<n> attached hydrogens	exactly one[1]
h<n>	implicit-H-count	<n> implicit hydrogens	at least one
R<n>	ring membership	in <n> SSSR rings	any ring atom
r<n>	ring size	in smallest SSSR ring of size <n>	any ring atom[2]
v<n>	valence	total bond order <n>	exactly one[2]
X<n>	connectivity	<n> total connections	exactly one[2]
x<n>	ring connectivity	<n> total ring connections	at least one[2]
- <n>	negative charge	-<n> charge	-1 charge (-- is -2, etc)
+<n>	positive charge	+<n> formal charge	+1 charge (++ is +2, etc)
#n	atomic number	atomic number <n>	(no default)[2]
@	chirality	anticlockwise	anticlockwise, default class[2]
@@	chirality	clockwise	clockwise, default class[2]
@<c><n>	chirality	chiral class <c> chirality <n>	(nodefault)
@<c><n>?	chiral or unspec	chirality <c><n> or unspecified	(no default)
<n>	atomic mass	explicit atomic mass	unspecified mass

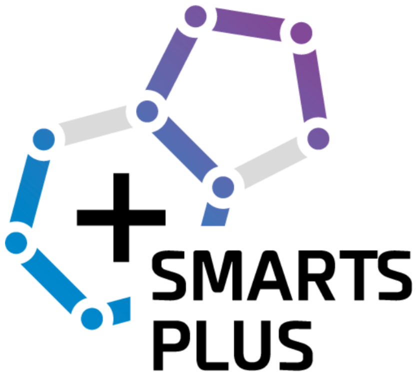


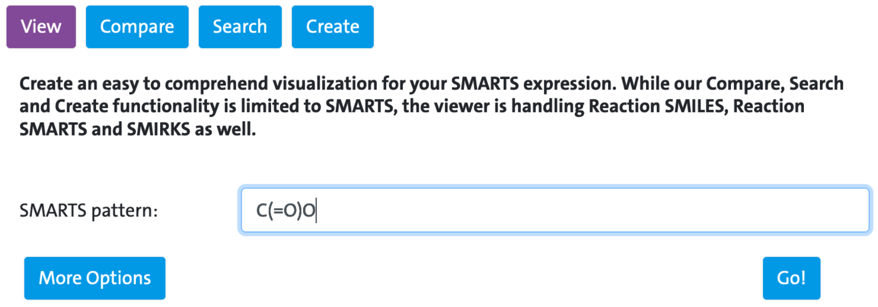
Examples:

C	aliphatic carbon atom
c	aromatic carbon atom
a	aromatic atom
[#6]	carbon atom
[Ca]	calcium atom
[++]	atom with a +2 charge
[R]	atom in any ring
[D3]	atom with 3 explicit bonds (implicit H's don't count)
[X3]	atom with 3 total bonds (includes implicit H's)
[v3]	atom with bond orders totaling 3 (includes implicit H's)
C[C@H](F)O	match chirality (H-F-O anticlockwise viewed from C)
C[C@?H](F)O	matches if chirality is as specified or is not specified

https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

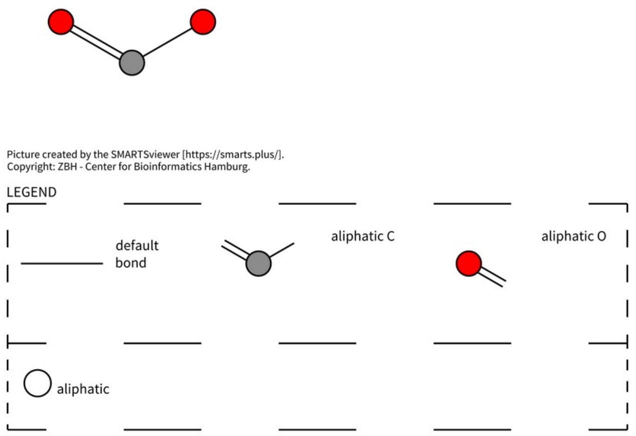
SMARTS.Plus

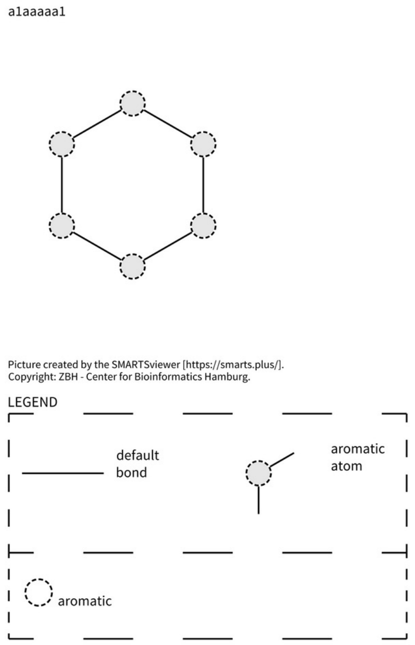




Create an easy to comprehend visualization for your SMARTS expression. While our Compare, Search and Create functionality is limited to SMARTS, the viewer is handling Reaction SMILES, Reaction SMARTS and SMIRKS as well.

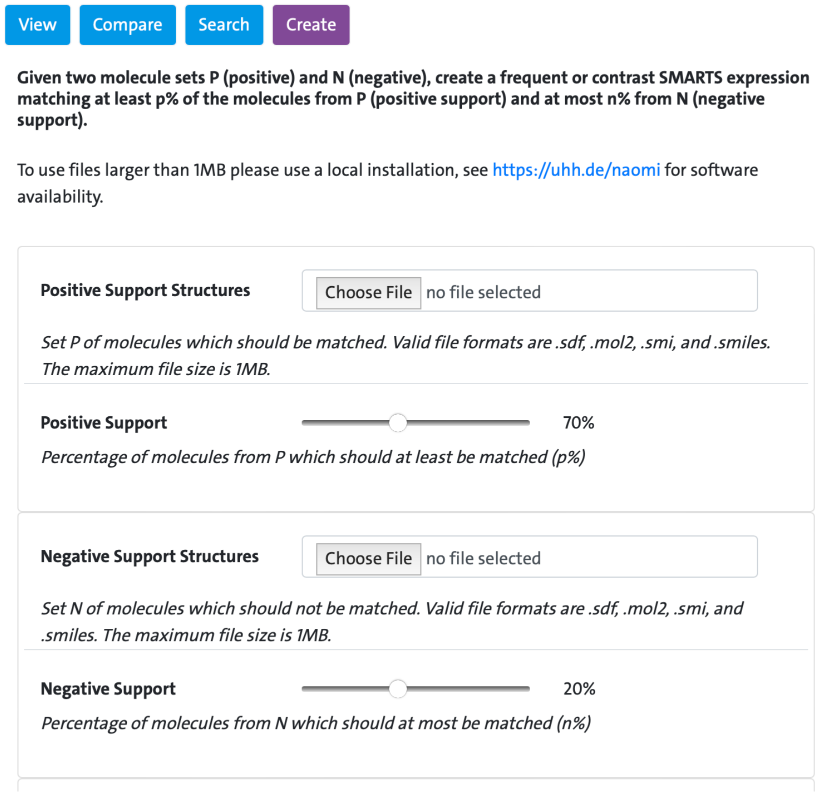
SMARTS pattern: C(=O)O


Picture created by the SMARTSviewer [https://smarts.plus/].
Copyright: ZBH - Center for Bioinformatics Hamburg.
LEGEND
default bond
aliphatic C
aliphatic O
aliphatic


a1aaaaa1
Picture created by the SMARTSviewer [https://smarts.plus/].
Copyright: ZBH - Center for Bioinformatics Hamburg.
LEGEND
default bond
aromatic atom
aromatic

- https://smarts.plus

Create SMARTS using SMARTS.plus



Given two molecule sets P (positive) and N (negative), create a frequent or contrast SMARTS expression matching at least p% of the molecules from P (positive support) and at most n% from N (negative support).

To use files larger than 1MB please use a local installation, see https://uhh.de/naomi for software availability.

Positive Support Structures Choose File no file selected

Set P of molecules which should be matched. Valid file formats are .sdf, .mol2, .smi, and .smiles. The maximum file size is 1MB.

Positive Support 70%

Percentage of molecules from P which should at least be matched (p%)

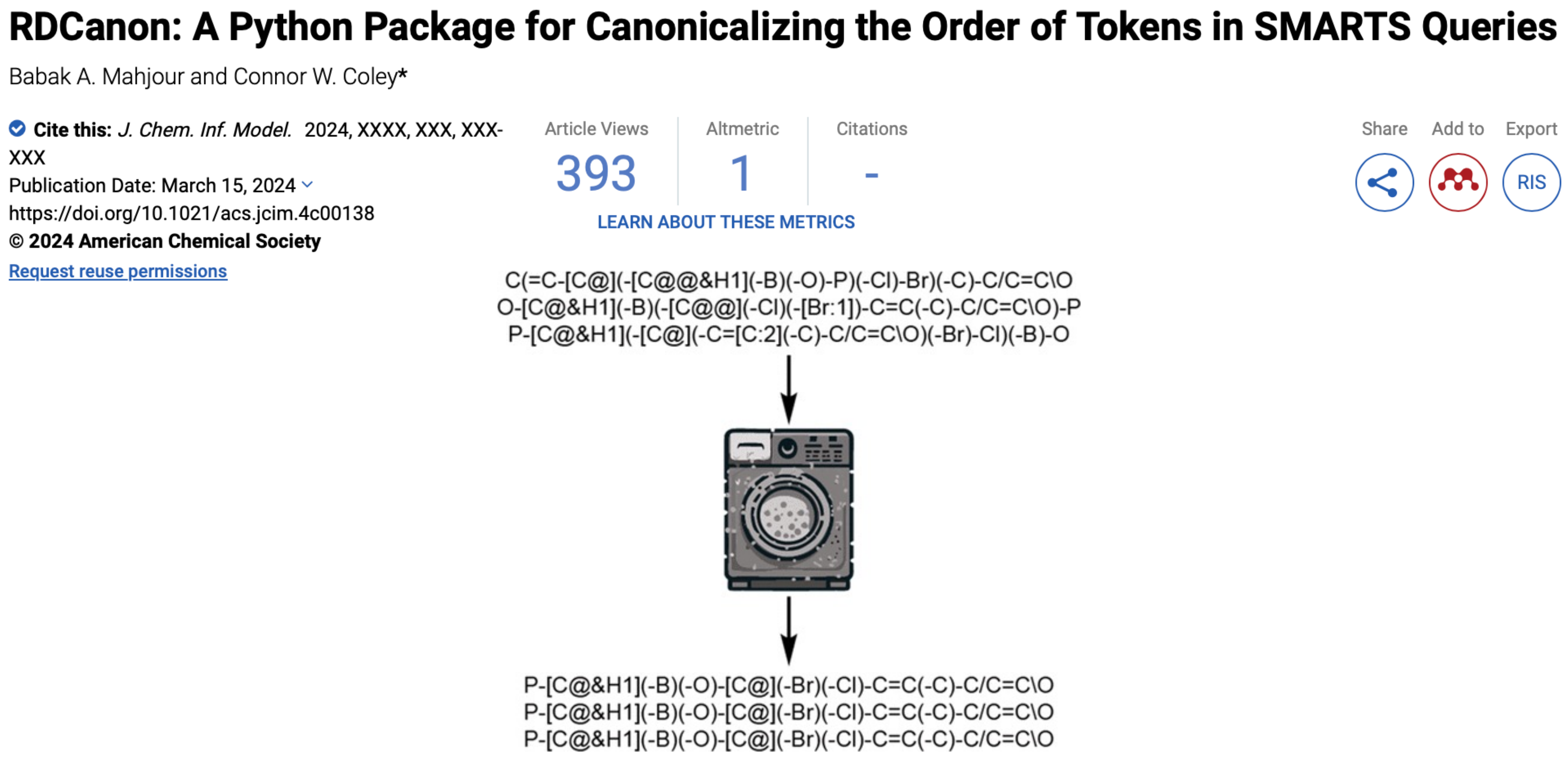
Negative Support Structures Choose File no file selected

Set N of molecules which should not be matched. Valid file formats are .sdf, .mol2, .smi, and .smiles. The maximum file size is 1MB.

Negative Support 20%

Percentage of molecules from N which should at most be matched (n%)

Canonicalisation SMARTS



RDCanon: A Python Package for Canonicalizing the Order of Tokens in SMARTS Queries

Babak A. Mahjour and Connor W. Coley*

Cite this: *J. Chem. Inf. Model.* 2024, XXXX, XXX, XXX-XXX
Publication Date: March 15, 2024
https://doi.org/10.1021/acs.jcim.4c00138
© 2024 American Chemical Society
Request reuse permissions

Article Views	Altmetric	Citations
393	1	-

LEARN ABOUT THESE METRICS

Share Add to Export

RIS

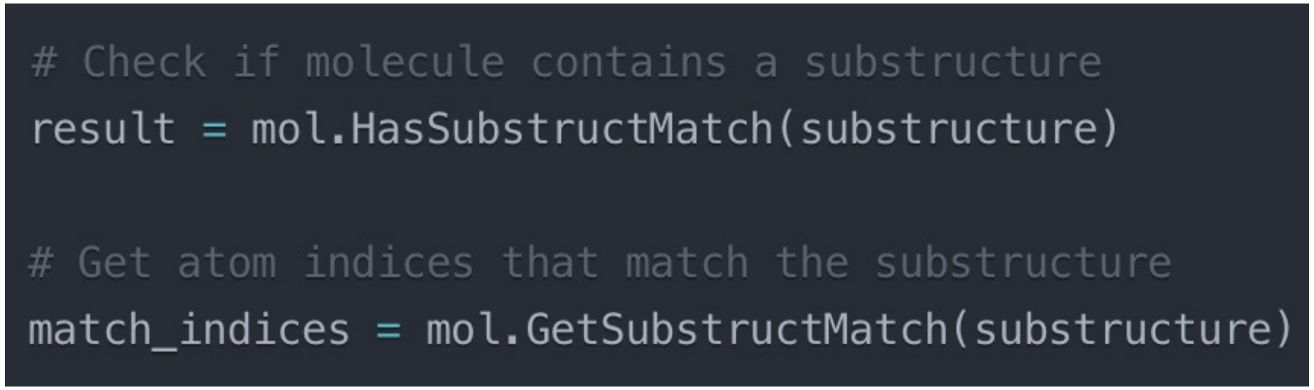
```
C(=C-[C@](-[C@@&H1](-B)(-O)-P)(-Cl)-Br)(-C)-C/C=C\O
O-[C@&H1](-B)(-[C@@](-Cl)(-[Br:1])-C=C(-C)-C/C=C\O)-P
P-[C@&H1](-[C@](-C=[C:2](-C)-C/C=C\O)(-Br)-Cl)(-B)-O
```

```
P-[C@&H1](-B)(-O)-[C@](-Br)(-Cl)-C=C(-C)-C/C=C\O
P-[C@&H1](-B)(-O)-[C@](-Br)(-Cl)-C=C(-C)-C/C=C\O
P-[C@&H1](-B)(-O)-[C@](-Br)(-Cl)-C=C(-C)-C/C=C\O
```

- https://github.com/coleygroup/rdcanon/

Substructure Matching

- Find specific patterns within molecules
- Essential for drug discovery and SAR studies
- RDKit provides multiple functions:
  - HasSubstructMatch - Boolean result
  - GetSubstructMatch - Returns atom indices that match



```python
# Check if molecule contains a substructure
result = mol.HasSubstructMatch(substructure)

# Get atom indices that match the substructure
match_indices = mol.GetSubstructMatch(substructure)
```

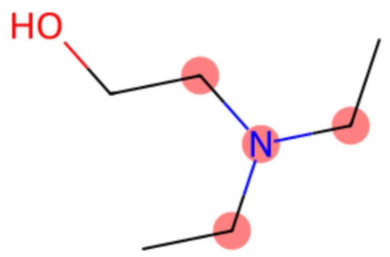
Substructure matches / search / atom highlight

```
for smiles in ['OCCS', 'CCS', 'OCC', 'CC']:
    mol = Chem.MolFromSmiles(smiles)
    pattern = Chem.MolFromSmarts('[O,S]')
    matches = mol.HasSubstructMatch(pattern)
    print(f"{smiles} ontains oxygen or sulfur: {matches}")
```

```
OCCS ontains oxygen or sulfur: True
CCS ontains oxygen or sulfur: True
OCC ontains oxygen or sulfur: True
CC ontains oxygen or sulfur: False
```

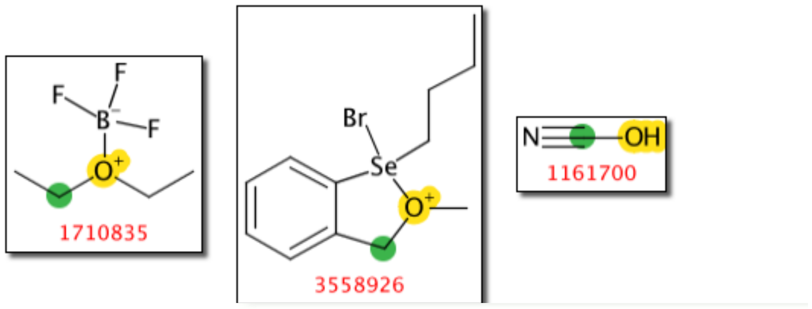
```
mol = Chem.MolFromSmiles('CCN(CC)CCO')
pattern = Chem.MolFromSmarts('[N](C)C')

# Get the atoms that match the pattern
matches = mol.GetSubstructMatches(pattern)

print(matches)
# Highlight the matching substructures
Draw.MolToImage(mol, highlightAtoms=[atom for match in matches for atom in match])
```

```
((2, 1, 3), (2, 1, 5), (2, 3, 5))
```

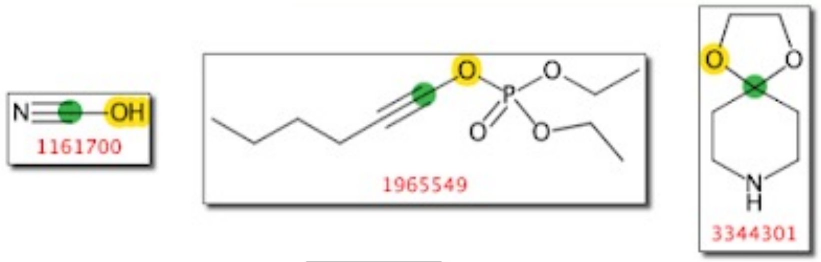


Being more specific

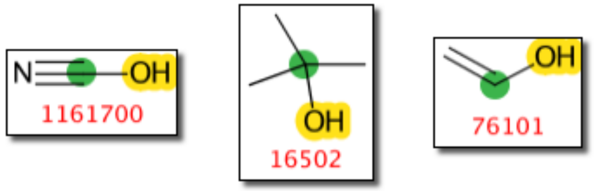
let's suppose we want to create a SMARTS pattern to match alcohols → “CO”



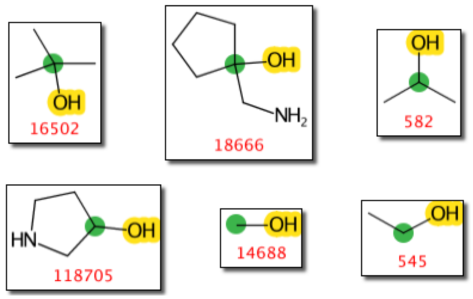
Ok, obviously we don't want a charged oxygen, so let's try "C[O+0]" and look at the first three hits:



And the oxygen should have a hydrogen, "C[Oh1+0]":



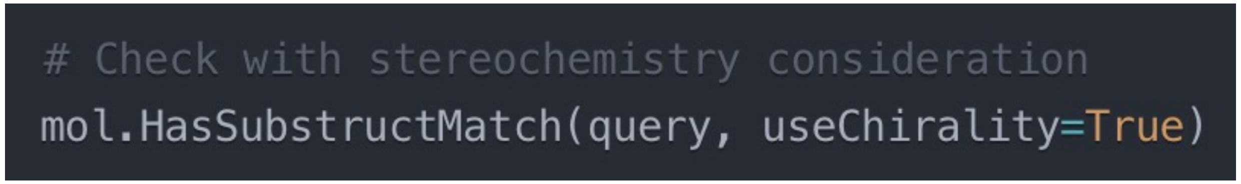
Now, about that carbon. I only had sp3-hybridised carbons in mind, so how about "[CX4][Oh1+0]"? Here is the full list of hits at this point:



- https://baoilleach.blogspot.com/2018/11/smarts-for-dummies.html

Stereochemistry in substructure match

- By default, RDKit ignores stereochemistry in substructure matching
- Important for cases like thalidomide (R vs S enantiomers)
- Use useChirality=True to enforce stereochemical matching

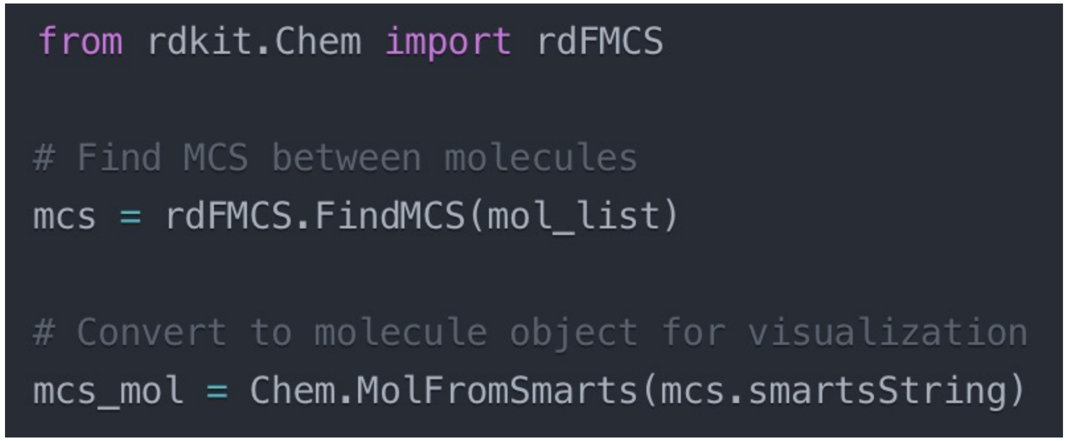


```
# Check with stereochemistry consideration
mol.HasSubstructMatch(query, useChirality=True)
```

Maximum common substructure (MCS)

Maximum Common Substructure (MCS)

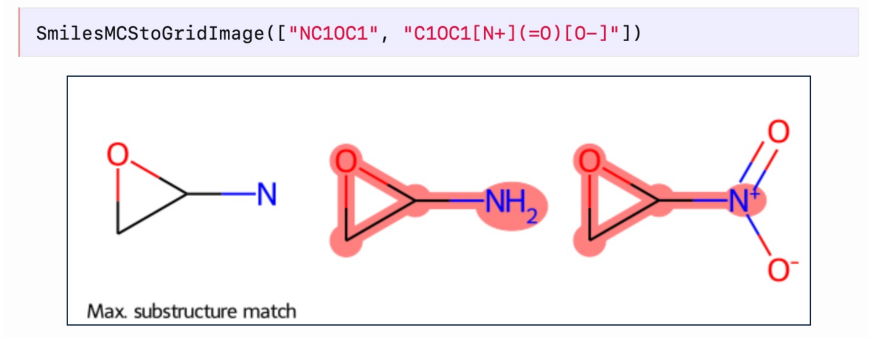
- Finds largest substructure shared by multiple molecules
- Applications:
  - Identifying pharmacophores
  - Clustering similar compounds
  - Structure-activity relationship studies



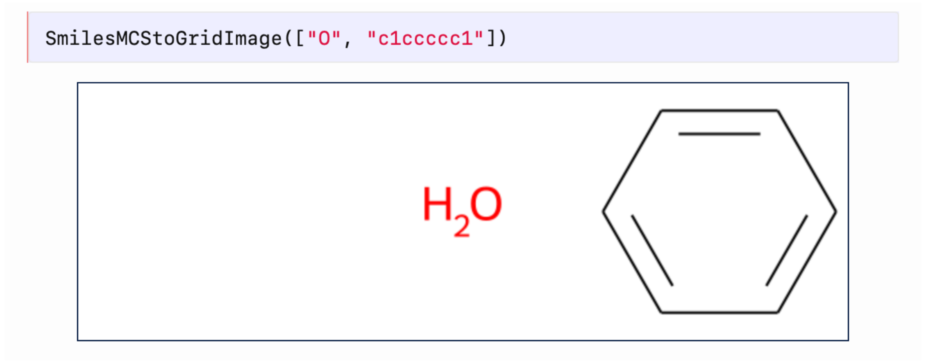
```python
from rdkit.Chem import rdFMCS

# Find MCS between molecules
mcs = rdFMCS.FindMCS(mol_list)

# Convert to molecule object for visualization
mcs_mol = Chem.MolFromSmarts(mcs.smartsString)
```

Find maximum common substructure in list of molecules

- **def SmilesMCStoGridImage**(smiles: list[str] **or** dict[str, str], align_substructure: bool **=** True, verbose: bool **=** False, **kwargs):
  """ Convert a list (or dictionary) of SMILES strings to an RDKit grid image of the maximum common substructure (MCS) match between them :returns: RDKit grid image”””



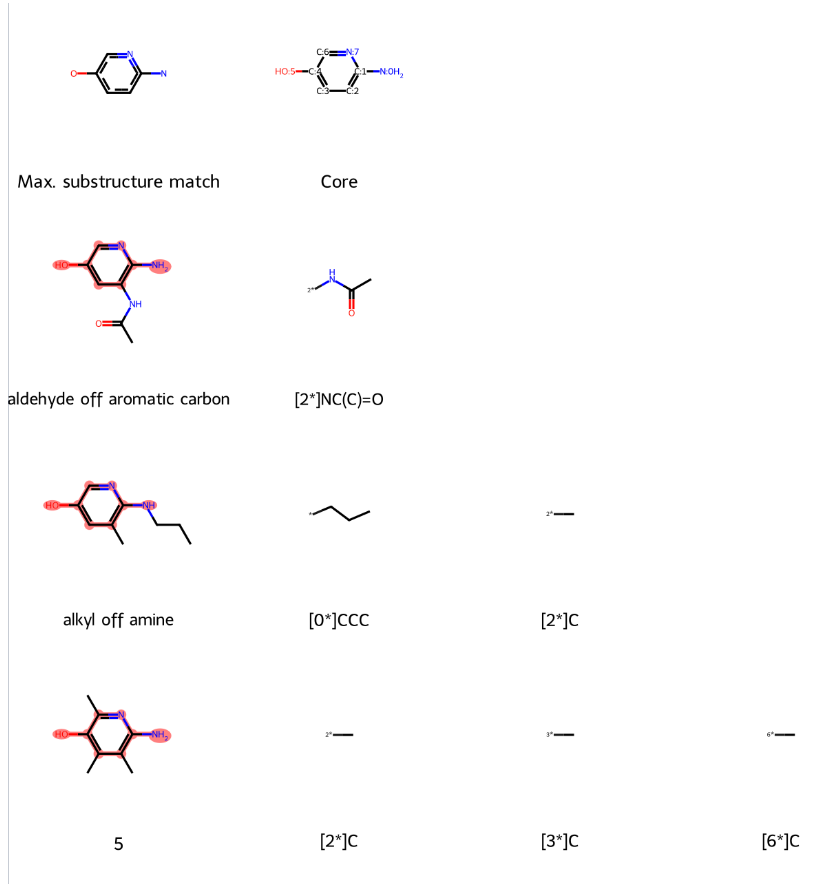
```
SmilesMCStoGridImage(["NC1OC1", "C1OC1[N+](=O)[O-]"])
```



```
SmilesMCStoGridImage(["O", "c1ccccc1"])
```

- https://bertiewooster.github.io/2022/10/09/RDKit-find-and-highlight-the-maximum-common-substructure-between-molecules.html

Extended version to find groups off common core

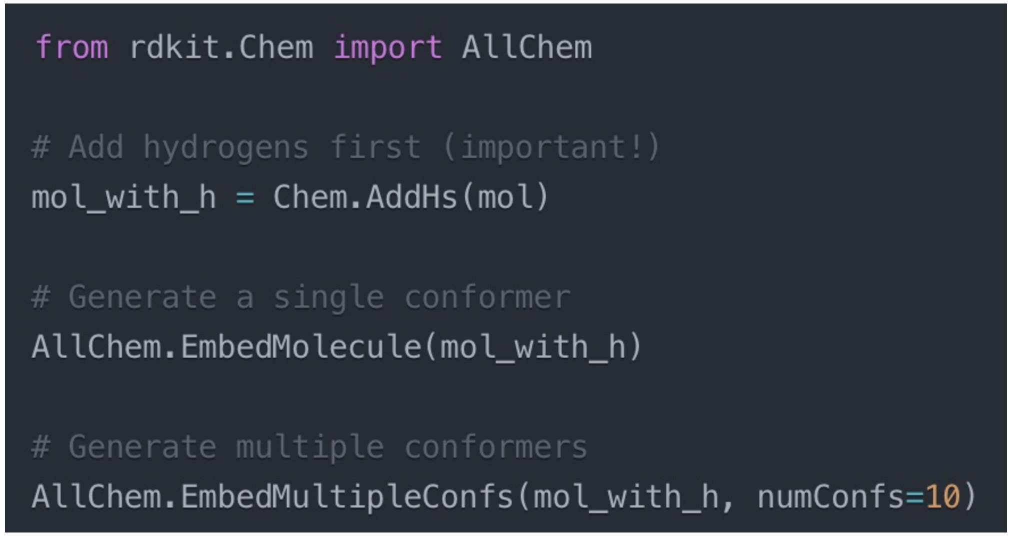


- https://bertiewooster.github.io/2022/12/25/RDKit-Find-Groups-Off-Common-Core.html

Additional RDKit functionality: e.g., going from 2D to 3D structures

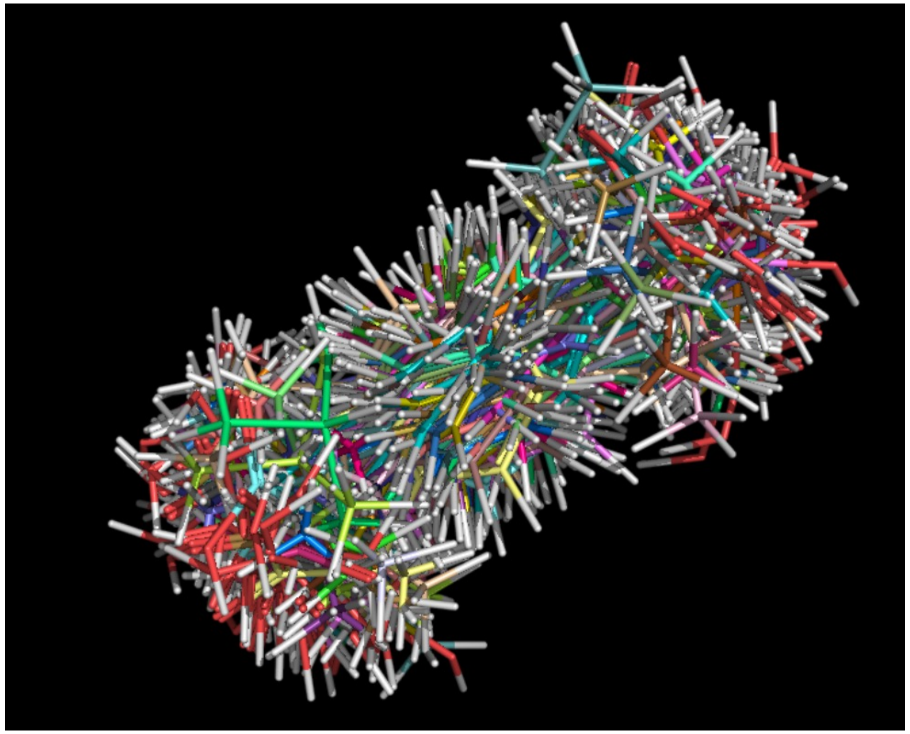
Conformer Generation

- Molecules exist in multiple 3D arrangements (conformations)
- Important for:
  - Understanding binding to targets
  - Predicting physicochemical properties
  - Docking studies



```python
from rdkit.Chem import AllChem

# Add hydrogens first (important!)
mol_with_h = Chem.AddHs(mol)

# Generate a single conformer
AllChem.EmbedMolecule(mol_with_h)

# Generate multiple conformers
AllChem.EmbedMultipleConfs(mol_with_h, numConfs=10)
```

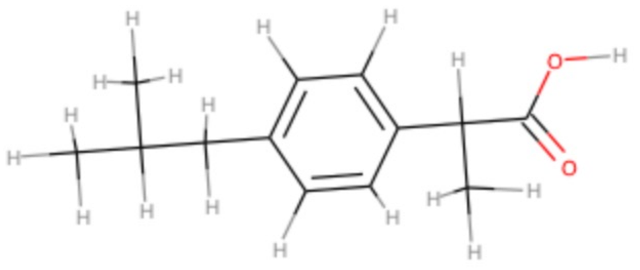
Conformer generation

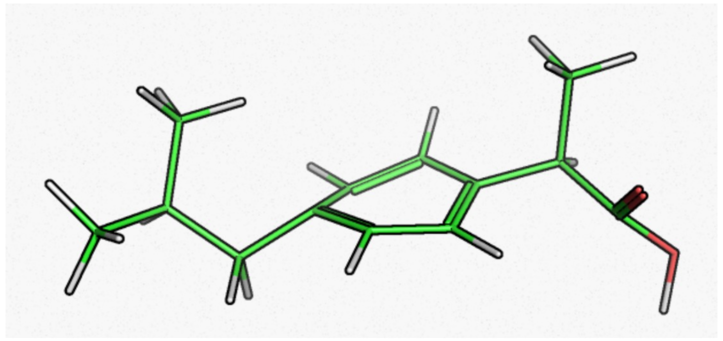


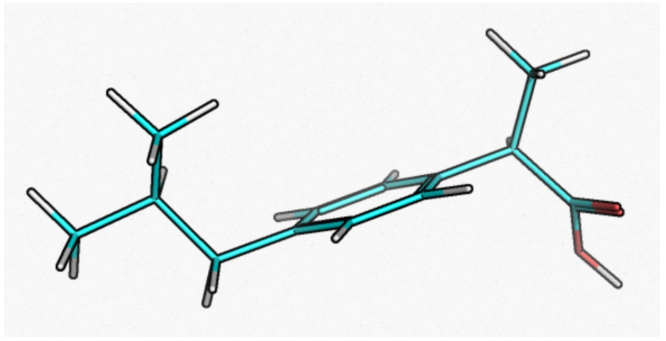
Conformers generation: https://asteeves.github.io/blog/2015/01/12/conformations-in-rdkit/
https://www.rdkit.org/docs/Cookbook.html#conformer-generation-with-etkdg
Force-field optimization: https://asteeves.github.io/blog/2015/01/12/optimizing-in-rdkit/

Conformer Energy Minimization and Analysis

- Minimize energy using force fields (e.g., UFF)
- Select lowest energy conformer for further analysis







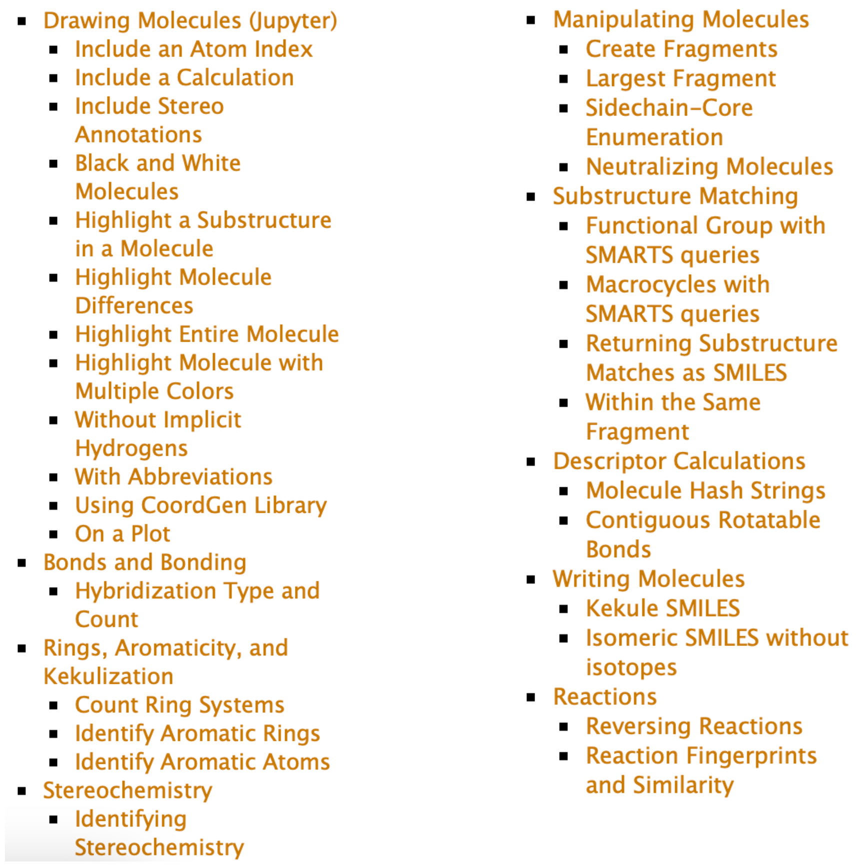
```
# Create the UFF force field object for this specific conformer
ff = AllChem.UFFGetMoleculeForceField(mol, confId=conf_id)
```

UFF stands for universal force field.

In the exercises, you will use py3dmol to visualise different conformers.

RDKit cookbook

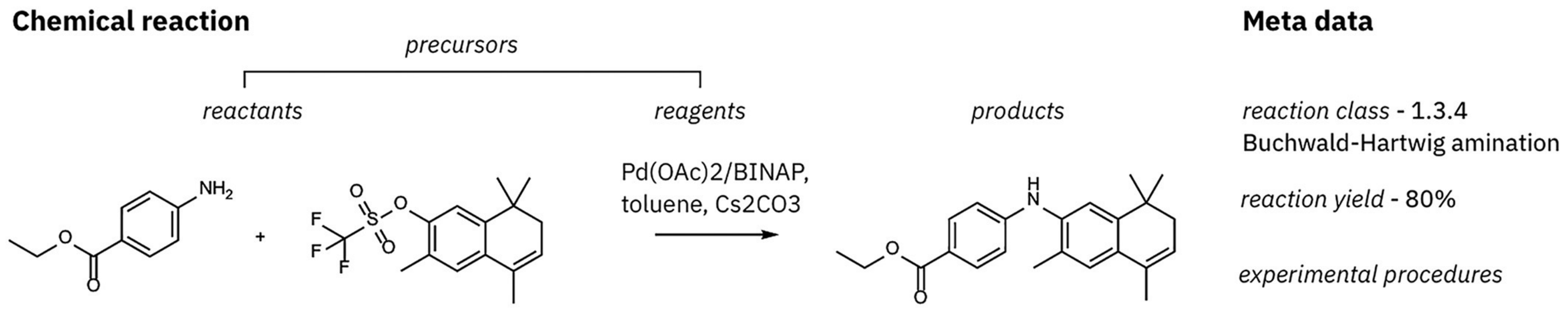
- https://www.rdkit.org/docs/Cookbook.html



- Drawing Molecules (Jupyter)
  - Include an Atom Index
  - Include a Calculation
  - Include Stereo Annotations
  - Black and White Molecules
  - Highlight a Substructure in a Molecule
  - Highlight Molecule Differences
  - Highlight Entire Molecule
  - Highlight Molecule with Multiple Colors
  - Without Implicit Hydrogens
  - With Abbreviations
  - Using CoordGen Library
  - On a Plot
- Bonds and Bonding
  - Hybridization Type and Count
- Rings, Aromaticity, and Kekulization
  - Count Ring Systems
  - Identify Aromatic Rings
  - Identify Aromatic Atoms
- Stereochemistry
  - Identifying Stereochemistry
- Manipulating Molecules
  - Create Fragments
  - Largest Fragment
  - Sidechain-Core Enumeration
  - Neutralizing Molecules
- Substructure Matching
  - Functional Group with SMARTS queries
  - Macrocycles with SMARTS queries
  - Returning Substructure Matches as SMILES
  - Within the Same Fragment
- Descriptor Calculations
  - Molecule Hash Strings
  - Contiguous Rotatable Bonds
- Writing Molecules
  - Kekule SMILES
  - Isomeric SMILES without isotopes
- Reactions
  - Reversing Reactions
  - Reaction Fingerprints and Similarity

Chemical reactions

Chemical reactions



How chemical reactions are typically reported in literature and patents…

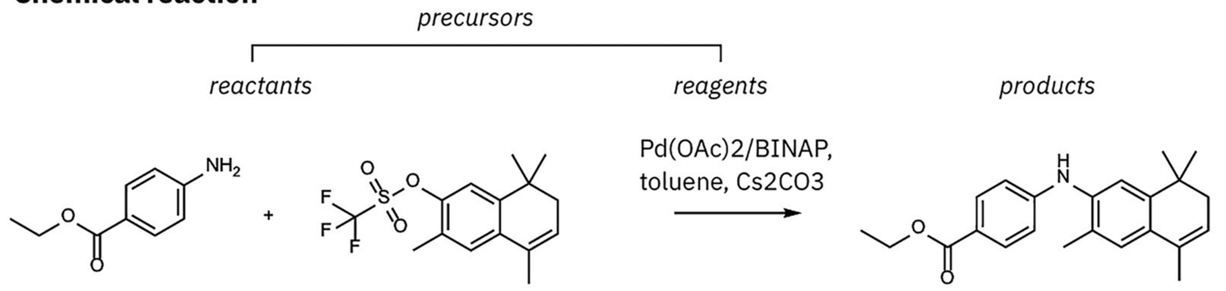
- No information on side products or byproducts…
- Incomplete metadata
- Often missing reagents (obvious for the human)

If you ever publish reactions, make sure the data is **machine-accessible and complete**.

- https://wires.onlinelibrary.wiley.com/doi/full/10.1002/wcms.1604

Chemical reaction SMILES

Chemical reaction



Meta data

reaction class - 1.3.4
Buchwald-Hartwig amination

reaction yield - 80%

experimental procedures

Reaction SMILES (text-based reaction representation, precursors>>products) Or reactants>reagents>products

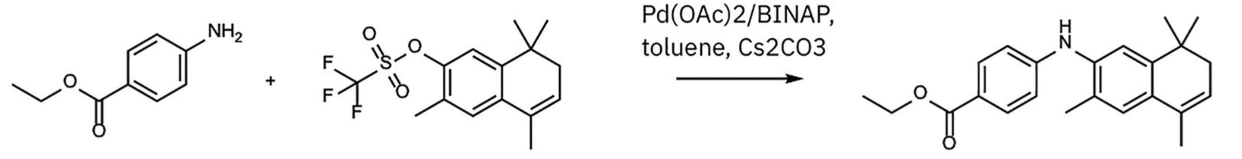
CC(=O)[O-].CC(=O)[O-].CC1=CCC(C)(C)c2cc(OS(=O)(=O)C(F)(F)F)c(C)cc21.CCOC(=O)c1ccc(N)cc1.Cc1ccccc1.O=C([O-])[O-].[Cs+].[Cs+].
[Pd+2].c1ccc(P(c2ccccc2)c2ccc3ccccc3c2-c2c(P(c3ccccc3)c3ccccc3)ccc3ccccc23)cc1>>CCOC(=O)c1ccc(Nc2cc3c(cc2C)C(C)=CCC3(C)C)cc1

Atom-mapping

Chemical reaction

precursors

reactants *reagents* *products*



Meta data

reaction class - 1.3.4
Buchwald-Hartwig amination

reaction yield - 80%

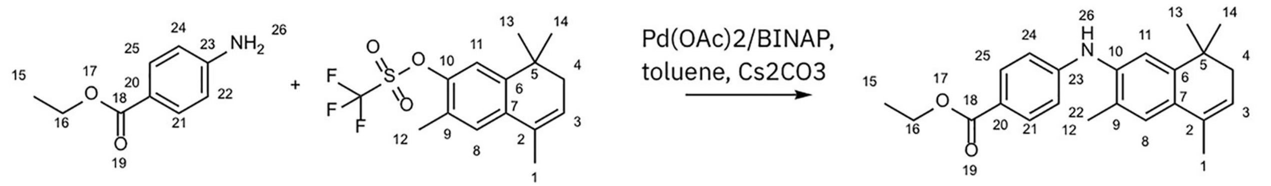
experimental procedures

Reaction SMILES (text-based reaction representation, precursors>>products)

CC(=O)[O-].CC(=O)[O-].CC1=CCC(C)(C)c2cc(OS(=O)(=O)C(F)(F)F)c(C)cc21.CCOC(=O)c1ccc(N)cc1.Cc1ccccc1.O=C([O-])[O-].[Cs+].[Cs+].
[Pd+2].c1ccc(P(c2ccccc2)c2ccc3ccccc3c2-c2c(P(c3ccccc3)c3ccccc3)ccc3ccccc23)cc1>>CCOC(=O)c1ccc(Nc2cc3c(cc2C)C(C)=CCC3(C)C)cc1

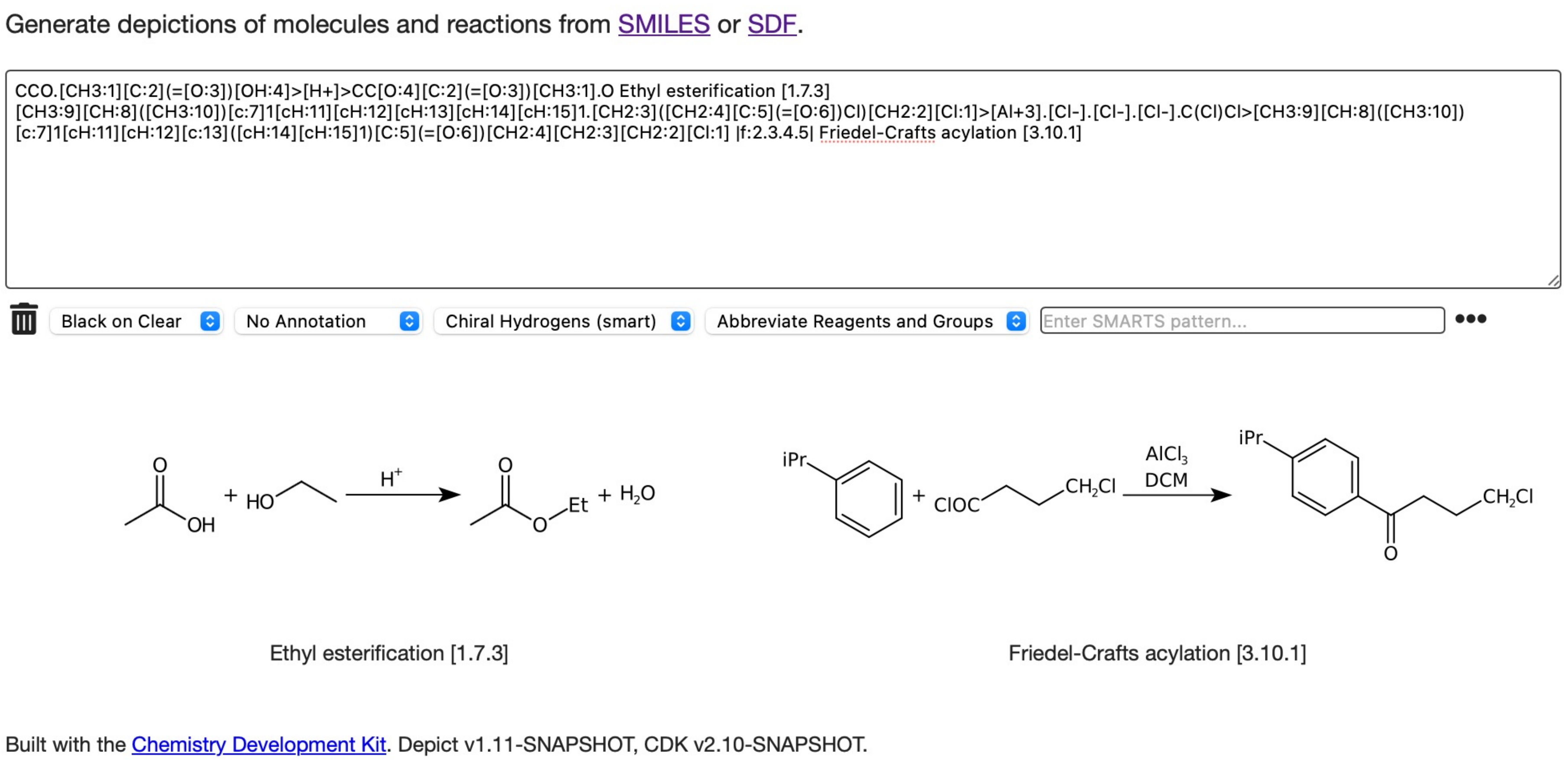
Atom-mapping (e.g. RXNMapper)

Atom-mapped reaction (required for reaction template, centre and bond change extraction)



CC(=O)[O-].CC(=O)[O-].Cc1ccccc1.O=C([O-])[O-].O=S(=O)(O[c:11]1[cH:12][c:13]2[c:14]([cH:15][c:16]1[CH3:17])[C:18]([CH3:19])=[CH:20]
[CH2:21][C:22]2([CH3:23])[CH3:24])C(F)(F)F.[CH3:1][CH2:2][O:3][C:4](=[O:5])[c:6]1[cH:7][cH:8][c:9]([NH2:10])[cH:25][cH:26]1.[Cs+].[Cs+].
[Pd+2].c1ccc(P(c2ccccc2)c2ccc3ccccc3c2-c2c(P(c3ccccc3)c3ccccc3)ccc3ccccc23)cc1>>[CH3:1][CH2:2][O:3][C:4](=[O:5])[c:6]1[cH:7][cH:8]
[c:9]([NH:10][c:11]2[cH:12][c:13]3[c:14]([cH:15][c:16]2[CH3:17])[C:18]([CH3:19])=[CH:20][CH2:21][C:22]3([CH3:23])[CH3:24])[cH:25][cH:26]1

CDK depict



Generate depictions of molecules and reactions from SMILES or SDF.

```
CCO.[CH3:1][C:2](=[O:3])[OH:4]>[H+]>CC[O:4][C:2](=[O:3])[CH3:1].O Ethyl esterification [1.7.3]
[CH3:9][CH:8]([CH3:10])[c:7]1[cH:11][cH:12][cH:13][cH:14][cH:15]1.[CH2:3]([CH2:4][C:5](=[O:6])Cl)[CH2:2][Cl:1]>[Al+3].[Cl-].[Cl-].[Cl-].C(Cl)Cl>[CH3:9][CH:8]([CH3:10])
[c:7]1[cH:11][cH:12][c:13]([cH:14][cH:15]1)[C:5](=[O:6])[CH2:4][CH2:3][CH2:2][Cl:1] |f:2.3.4.5| Friedel-Crafts acylation [3.10.1]
```

Black on Clear | No Annotation | Chiral Hydrogens (smart) | Abbreviate Reagents and Groups | Enter SMARTS pattern...

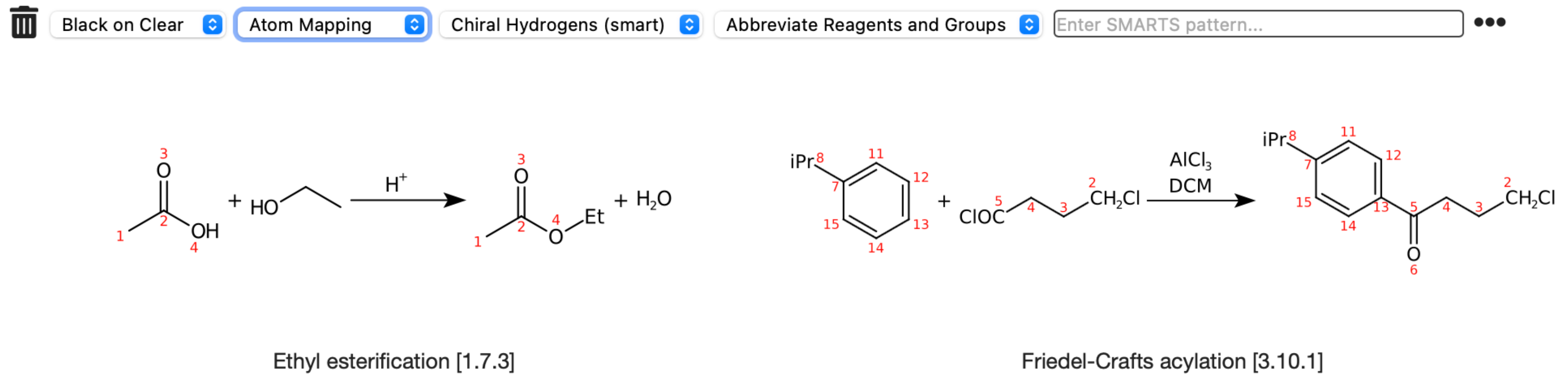
Ethyl esterification [1.7.3]

Friedel-Crafts acylation [3.10.1]

Built with the Chemistry Development Kit. Depict v1.11-SNAPSHOT, CDK v2.10-SNAPSHOT.

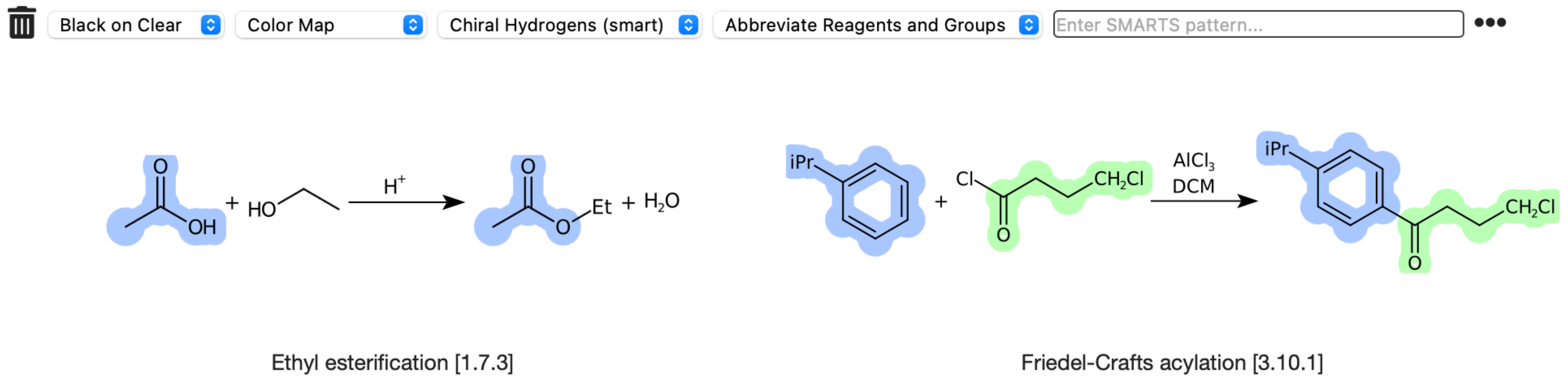
- https://www.simolecule.com/cdkdepict/depict.html

CDK depict options


Black on Clear | Atom Mapping | Chiral Hydrogens (smart) | Abbreviate Reagents and Groups | Enter SMARTS pattern...

Ethyl esterification [1.7.3]

Friedel-Crafts acylation [3.10.1]

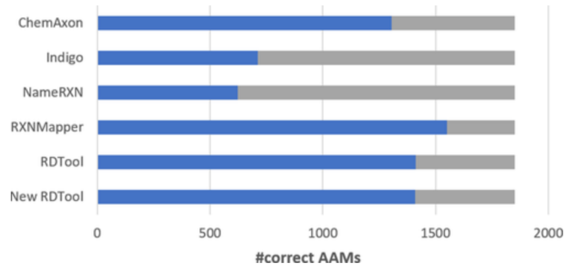

Black on Clear | Color Map | Chiral Hydrogens (smart) | Abbreviate Reagents and Groups | Enter SMARTS pattern...

Ethyl esterification [1.7.3]

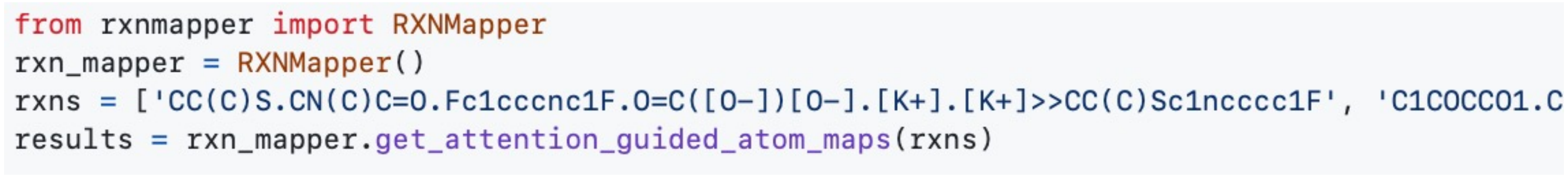
Friedel-Crafts acylation [3.10.1]

EPFL

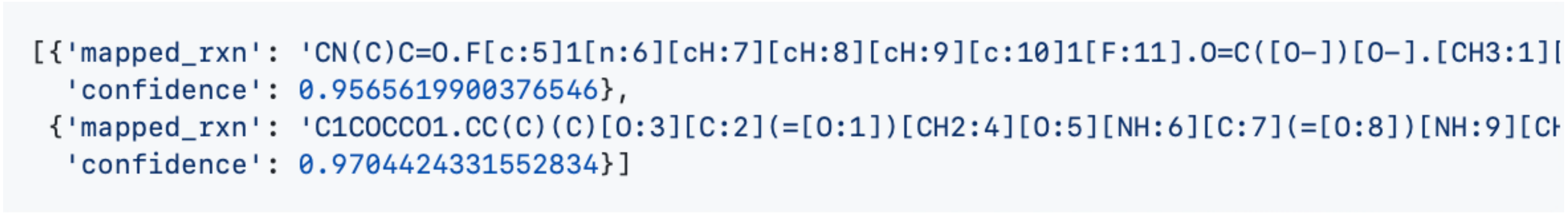
Open source atom-mapping tools



- RXNMapper (https://github.com/rxn4chemistry/rxnmapper)
  - pip install "rxmapper[rdkit]"



```
from rxnmapper import RXNMapper
rxn_mapper = RXNMapper()
rxns = ['CC(C)S.CN(C)C=O.Fc1cccnc1F.O=C([O-])[O-].[K+].[K+]>>CC(C)Sc1ncccc1F', 'C1COCCO1.C
results = rxn_mapper.get_attention_guided_atom_maps(rxns)
```

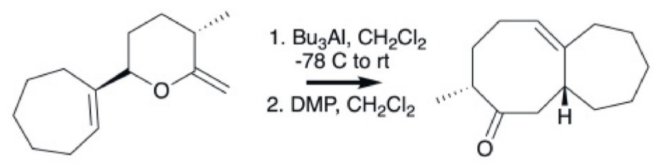


```
[{'mapped_rxn': 'CN(C)C=O.F[c:5]1[n:6][cH:7][cH:8][cH:9][c:10]1[F:11].O=C([O-])[O-].[CH3:1][
  'confidence': 0.9565619900376546},
 {'mapped_rxn': 'C1COCCO1.CC(C)(C)[O:3][C:2](=[O:1])[CH2:4][O:5][NH:6][C:7](=[O:8])[NH:9][CH
  'confidence': 0.9704424331552834}]
```

- Comparison of different approaches: https://onlinelibrary.wiley.com/doi/10.1002/minf.202100138

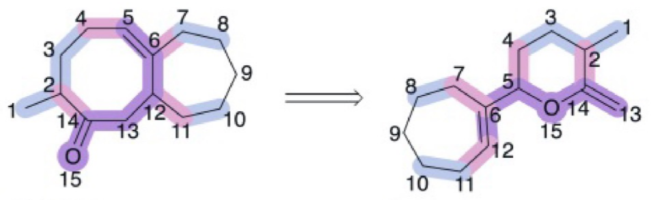
Reaction templates

a) Forward Reaction



Reactant **Product**

b) Retrosynthetic Reaction

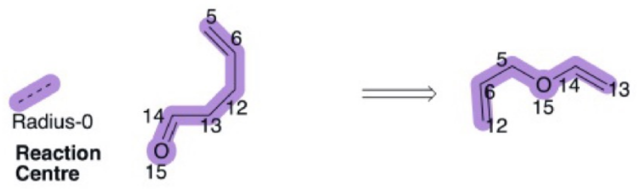


Product **Reactant**

Atom-mapped reaction SMILES

```
[CH3:1][CH:2]1[CH2:3][CH2:4][CH:5]([C:6]2=[CH:12][CH2:11][CH2:10][CH2:9]
[CH2:8][CH2:7]2)[O:15][C:14]1=[CH2:13]>>[CH3:1][CH:2]1[CH2:3][CH2:4]/
[CH:5]=[C:6]2/[CH2:7][CH2:8][CH2:9][CH2:10][CH2:11]CH:12]2[CH2:13]
[C:14]1=[O:15]
```

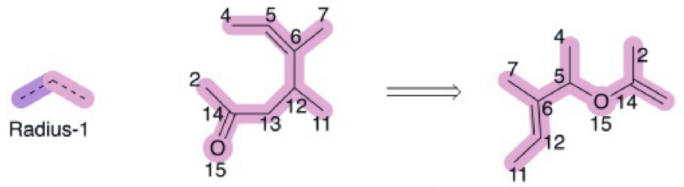
c)



Reaction SMARTS - Shell/Radius 0

```
([CH;D2;+0:4]=[C;H0;D3;+0:5]\\[CH;D3;+0:6]-[CH2;D2;+0:1]-
[C;H0;D3;+0:2]=[O;H0;D1;+0:3])>>([CH2;D1;+0:1]=[C;H0;D3;+0:2]-
[O;H0;D2;+0:3]-[CH;D3;+0:4]-[C;H0;D3;+0:5]=[CH;D2;+0:6])
```

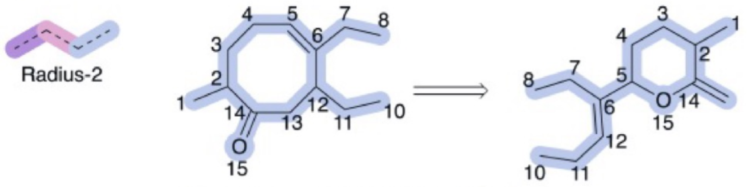
d)



Reaction SMARTS - Shell/Radius 1

```
([C:1]-[CH;D3;+0:2](-[CH2;D2;+0:10]-[C;H0;D3;+0:8](-
[C:9])=[O;H0;D1;+0:7])/[C;H0;D3;+0:3](-[C:4])=[CH;D2;+0:5]\\[C:6])>>([C:1]-
[CH;D2;+0:2]=[C;H0;D3;+0:3](-[C:4])-[CH;D3;+0:5](-[C:6])-[O;H0;D2;+0:7]-
[C;H0;D3;+0:8](-[C:9])=[CH2;D1;+0:10])
```

e)



Reaction SMARTS - Shell/Radius 2

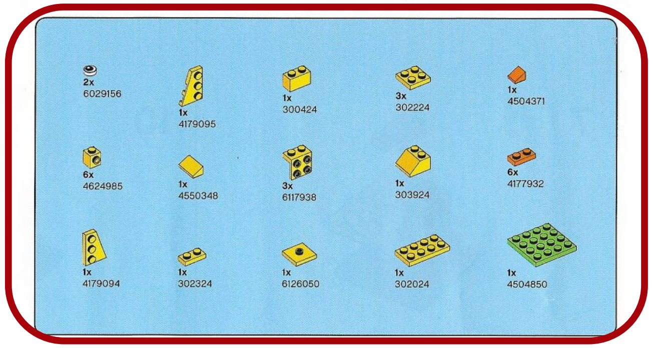
```
([C:1]-[C:2]-[CH;D3;+0:3]1-[CH2;D2;+0:13]-[C;H0;D3;+0:12]
(=[O;H0;D1;+0:14])-[C:10](-[C;D1;H3:11])-[C:9]-[C:8]/
[CH;D2;+0:7]=[C;H0;D3;+0:4]\\1-[C:5]-[C:6])>>([C:1]-[C:2]-
[CH;D2;+0:3]=[C;H0;D3;+0:4](-[C:5]-[C:6])-[CH;D3;+0:7]1-[C:8]-[C:9]-[C:10](-
[C;D1;H3:11])-[C;H0;D3;+0:12](=[CH2;D1;+0:13])-[O;H0;D2;+0:14]-1)
```

- https://www.chimia.ch/chimia/article/view/2022_294/5301

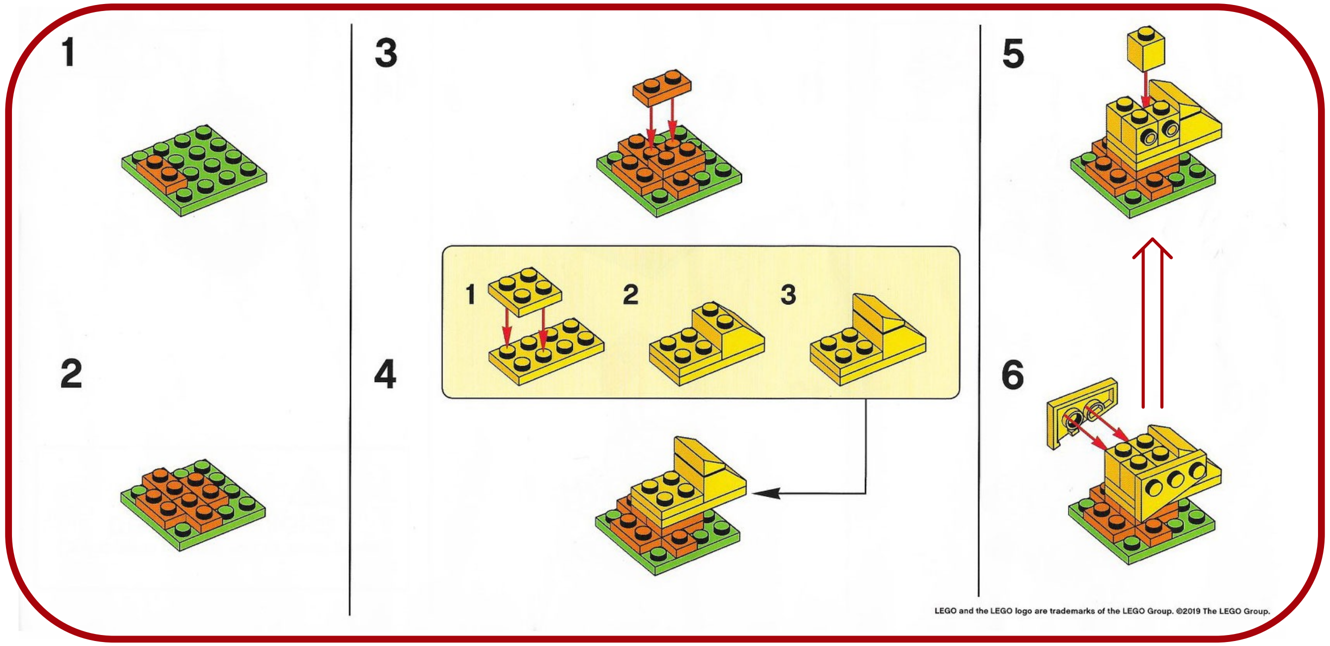
Retrosynthesis (Corey, Nobel prize, 1990)

Target molecule

Known (commercially available) building blocks



Steps to construct the target

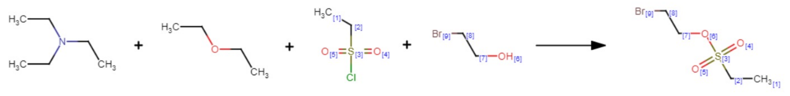

1
2
3
4
1
2
3
5
6
LEGO and the LEGO logo are trademarks of the LEGO Group. ©2019 The LEGO Group.

Lego analogy: Amol Thakkar

Do I have to write those templates myself?

```
pip install reaction-utils
```

- There are tools for automatic extractions:
  https://github.com/MolecularAI/reaction_utils (code)
  https://molecularai.github.io/reaction_utils/ (documentation)



```
CCN(CC)CC.CCOCC.Cl[S:3]([CH2:2][CH3:1])(=[O:4])=[O:5].[OH:6][CH2:7][(
```

First we create a `ChemicalReaction` object that is encapsulating the reaction and provides some simple curation routines.

```
from rxnutils.chem.reaction import ChemicalReaction

reaction = "CCN(CC)CC.CCOCC.Cl[S:3]([CH2:2][CH3:1])(=[O:4])=[O:5].[OH
rxn = ChemicalReaction(reaction)
```

if you inspect the `reactants_list` property, you will see that two of the reactants from the reaction SMILES have been moved to the list of agents because they are not mapped.

```
rxn.reactants_list
>> ['Cl[S:3]([CH2:2][CH3:1])(=[O:4])=[O:5]', '[OH:6][CH2:7][CH2:8][Br

rxn.agents_list
>> ['CCN(CC)CC', 'CCOCC']
```

```
rxn.generate_reaction_template(radius=1)

rxn.retro_template
>> <rxnutils.chem.template.ReactionTemplate at 0x7fe4e9488d90>

rxn.retro_template.smarts
>> '[C:2]-[S;H0;D4;+0:1](=[O;D1;H0:3])(=[O;D1;H0:4])-[O;H0;D2;+0:6]-
```

```
smiles="CCS(=O)(=O)OCCBr"
reactant_list = rxn.retro_template.apply(smiles)
reactant_list
>> (('CCS(=O)(=O)Cl', 'OCCBr'),)
```

Happy coding!

(don’t forget to form groups for the projects → 3-4 students per group)