

# Data Processing, Analysis and Visualization in Python

Basic Cheminformatics –  
RDKit

1

## Outline

- Introduction
- Representing molecules
  - SMILES
- Fingerprints
  - Morgan
- Similarity
  - Tanimoto

2

RDKit



Open-Source Cheminformatics  
and Machine Learning

- **Functionality**
  - 2D and 3D molecular operations
  - Descriptor generation for machine learning
  - Molecular database cartridge for PostgreSQL
  - Cheminformatics nodes for KNIME (distributed from the KNIME community site: <https://www.knime.com/rdkit>)
  - Supports Mac/Windows/Linux
- **Web presence:**
  - Homepage: <http://www.rdkit.org> Documentation, links
  - Github (<https://github.com/rdkit>) Downloads, bug tracker, git repository
  - Sourceforge (<http://sourceforge.net/projects/rdkit>) Mailing lists
  - Blog (<https://rdkit.blogspot.com>) Tips, tricks, random stuff
  - Tutorials (<https://github.com/rdkit/rdkit-tutorials>) Jupyter-based tutorials for using the RDKit
  - KNIME integration (<https://github.com/rdkit/knime-rdkit>)
- **Social media:**
  - Twitter: @RDKit\_org

3

RDKit



Open-Source Cheminformatics  
and Machine Learning

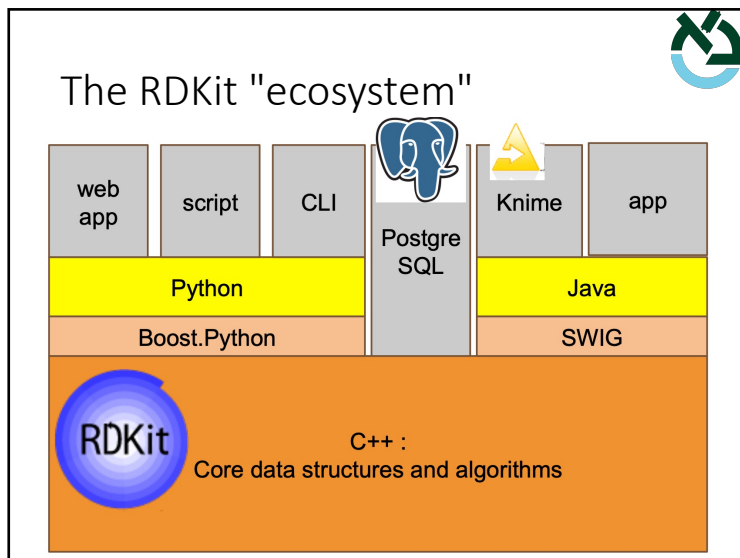
- **History**
  - 2000-2006: Developed and used at Rational Discovery (RD) for building predictive models for ADME, Tox, biological activity
  - June 2006: Open-source (BSD license) release of software, Rational Discovery shuts down
  - to present: Open-source development continues, use within Novartis, contributions from Novartis back to open-source version



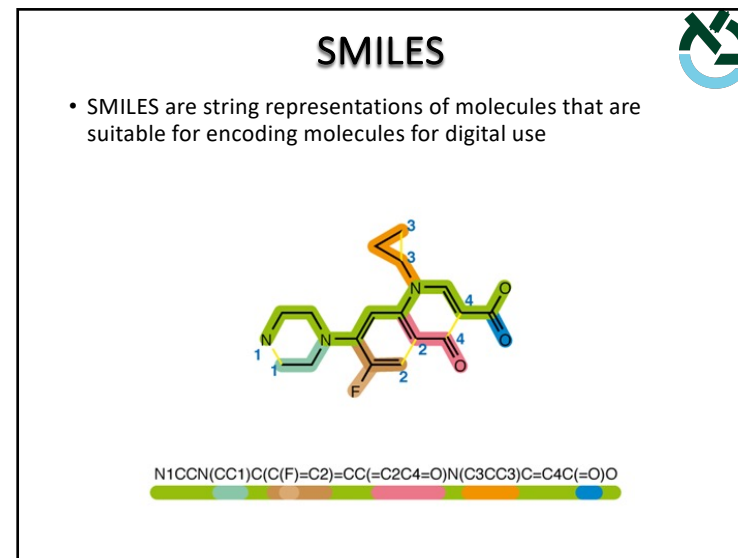
### Open Source Science

The Novartis Institutes for BioMedical Research (NIBR) is pioneering new informatics tools for drug discovery. We believe in the power of open-sourced, global collaboration for the greater good. Join us to help patients worldwide.

4



5



6

### SMILES

- What is SMILES (Simplified Molecular Input Line Entry System) ?
- SMILES is a chemical notation that allows a user to represent a chemical structure in a way that can be used by the computer.
- SMILES was developed through funding from the U.S. Environmental Protection Agency, Mid-Continent Ecology Division-Duluth, (MED-Duluth) Duluth, MN
- SMILES has five+ basic syntax rules which must be observed. If basic rules of chemistry are not followed in SMILES entry, the system will warn the user and ask that the structure be edited or reentered.

7

### SMILES

**SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules**

DAVID WEININGER  
Medicinal Chemistry Project, Pomona College, Claremont, California 91711  
Received June 17, 1987

SMILES (Simplified Molecular Input Line Entry System) is a chemical notation system designed for modern chemical information processing. Based on principles of molecular graph theory, SMILES allows rigorous structure specification by use of a very small and natural grammar. The SMILES notation system is also well suited for high-speed machine processing. The resulting ease of usage by the chemist and machine compatibility allow many highly efficient chemical computer applications to be designed including generation of a unique notation, constant-speed (zeroth order) database retrieval, flexible substructure searching, and property prediction models.

*J. Chem. Inf. Comput. Sci.* 1988, 28, 31-36.

8

## SMILES Rule #1



### Atoms and Bonds

- SMILES supports all elements in the periodic table.
- An atom is represented using its respective atomic symbol.
  - Upper case letters refer to non-aromatic atoms
  - Lower case letters refer to aromatic atoms
  - If the atomic symbol has more than one letter the second letter must be lower case.
- Bonds are denoted as shown below:
  - - Single bond
  - = Double bond
  - # Triple bond
  - \* Aromatic bond
  - . Disconnected structures

9

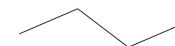
## SMILES with RDKit



- Installing RDKit (in jupyter notebook)

```
>>> %capture
>>> !pip install rdkit-pypi

>>> from rdkit import Chem
>>> mol = Chem.MolFromSmiles('CCCC')
>>> mol
```



```
>>> smiles = Chem.MolToSmiles(mol)
>>> smiles
'CCCC'
```

rdkit\_basics.ipynb

10

## SMILES Rule #1



### Atoms and Bonds

- Single bonds are the default and therefore need not be entered.
  - For example, 'CC' would mean that there is a non-aromatic carbon attached to another non-aromatic carbon by a single bond (ethane). 'cc' is two carbons connected via an aromatic bond.
  - A blank terminates the SMILES string.

11

## SMILES Rule #2



### Simple Chains

- By combining atomic symbols and bond symbols simple chain structures can be represented.
- The structures that are entered using SMILES are hydrogen-suppressed
- The SMILES software understands the number of possible connections that an atom can have.
  - If enough bonds are not identified by the user through SMILES notation, the system will automatically assume that the other connections are satisfied by hydrogen bonds.
- Some examples:
 

• CC	CH <sub>3</sub> CH <sub>3</sub>	Ethane
• C=C	CH <sub>2</sub> CH <sub>2</sub>	Ethene
• CBr	CH <sub>3</sub> Br	Bromomethane
• C#N	C≡N	Hydrocyanic acid
• Na.Cl	NaCl	Sodium chloride

Show these with RDKit!



12

## SMILES Rule #2



### Simple Chains

- The user can explicitly identify the hydrogen bonds, but if one hydrogen bond is identified in the string, the SMILES interpreter will assume that the user has identified all hydrogens for that molecule.
- [H]C([H])=C([H])[H] Ethene
- Because SMILES allows entry of all elements in the periodic table, and also utilizes hydrogen suppression, elements with two letters that could be misinterpreted by the computer.
  - For example, 'Sc' could be interpreted as a sulfur atom connected to an aromatic carbon by a single bond, or it could be the symbol for scandium.
  - The SMILES interpreter gives priority to the interpretation of a single bond connecting a sulfur atom and an aromatic carbon. To identify scandium the user should enter [Sc].

13

## SMILES Rule #3



### Branches

- A branch from a chain is specified by placing the SMILES symbol(s) for the branch between parentheses. The string in parentheses is placed directly after the symbol for the atom to which it is connected. If it is connected by a double or triple bond, the bond symbol immediately follows the left parenthesis.
- Some examples:
 

<chem>CC(O)C</chem>	2-Propanol
<chem>CC(=O)C</chem>	2-Propanone
<chem>CC(CC)C</chem>	2-Methylbutane
<chem>CC(C)CC(=O)</chem>	2-Methylbutanal
<chem>CCC(C)(C)C</chem>	2-Dimethylbutane

Show these with RDKit!



14

## SMILES Rule #4



### Rings

- SMILES allows a user to identify ring structures by using numbers to identify the opening and closing ring atom.
- For example, in C1CCCCC1, the first carbon has a number '1' which connects by a single bond with the last carbon which also has a number '1' (cyclohexane).
- Chemicals that have multiple rings may be identified by using different numbers for each ring.
- If a double, single, or aromatic bond is used for the ring closure, the bond symbol is placed before the ring closure number.

15

## SMILES Rule #4



### Rings

- Some examples:
 

<chem>C=1CCCCC1</chem>	Cyclohexene
<chem>C*1*C*C*C*C1</chem>	'*' is any atom
<chem>c1ccccc1</chem>	Benzene
<chem>C1OC1CC</chem>	Ethyloxirane
<chem>c1cc2ccccc2cc1</chem>	Naphthalene
<chem>c1c(N(=O)=O)cccc1</chem>	Nitrobenzene

Show these with RDKit!



16

## SMILES Rule #5

### Charged Atoms

- The format for identifying a charged atom consists of the atom and charge in square brackets.
- For example:
  - CCC(=O)[O-] Ionized form of propanoic acid
  - CCC(=O)[O-]
  - C1cccc[n+](C1)CC(=O)O 1-Carboxymethyl pyridinium

Show these with RDKit!



17

## SMILES Rule #6

### Stereochemistry

- The absolute stereochemistry at chiral atoms is indicated using the "@" symbol
- Alanine:
  - N[C@H](C)C(=O)O
  - N[C@@H](C)C(=O)O
- E/Z-isomers (cis/trans) are indicated using "/", "\"

Show these with RDKit!



- Butene:

Show these with RDKit!

- C/C=C/C
- C/C=C\C

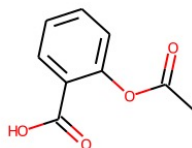


18

## Canonical SMILES

### SMILES are not unique!

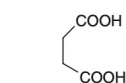
- Many possible SMILES strings can be written for aspirin, two of which are:
  - OC(=O)c1ccccc1OC(=O)C
  - c1cccc(OC(=O)C)c1C(=O)O



- Solution: Use a canonical representation (Morgan)

19

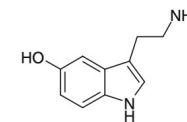
## Some SMILES Examples



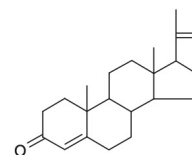
succinic acid:  
OC(=O)CCC(=O)O



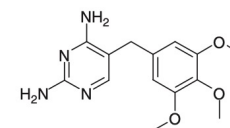
cubane:  
C1(C2C3C14)C5C2C3C45



serotonin:  
NCCc1c[nH]c2ccc(O)cc12



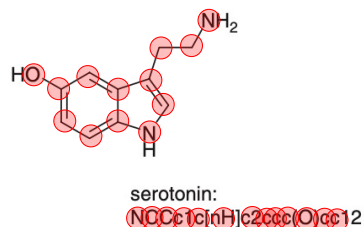
progesterone:  
CC(=O)C1CCC2C3CCC4=CC(=O)CCC4(C)C3CCC12C



trimethoprim: COc1cc(Cc2cnc(N)nc2N)cc(OC)c1OC

20


## Some SMILES Examples



21

## Basic Descriptors in RDKit

```
>>> from rdkit import Chem
>>> mol = Chem.MolFromSmiles('CCCC')
>>> mol
```



```
>>> from rdkit.Chem import Descriptors
>>> mw = Descriptors.MolWt(mol)
>>> mw
58.123999999999995
>>> logp = Descriptors.MolLogP(mol)
>>> logp # P = [organic]/[aqueous]
1.8064
```


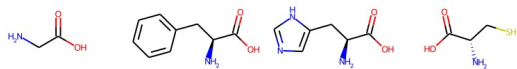
rdkit\_basics.ipynb

22

## Basic Descriptors in RDKit

```
>>> from rdkit.Chem import Draw
>>> smiles_list = ['C(C(=O)O)N',
                  'C1=CC=C(C=C1)C[C@@H](C(=O)O)N',
                  'C1=CC(=NC=C1)C[C@@H](C(=O)O)N', 'C([C@@H](C(=O)O)N)S']
>>> mol_list = []
>>> for smiles in smiles_list:
>>>     mol = Chem.MolFromSmiles(smiles)
>>>     mol_list.append(mol)
>>> img = Draw.MolsToGridImage(mol_list, molsPerRow=4)
>>> img
```

Try other aa!

rdkit\_basics.ipynb

23

## Find patterns in RDKit

```
>>> pattern = Chem.MolFromSmiles('S') # Has sulfur
>>> for mol in mol_list:
>>>     print(mol.HasSubstructMatch(pattern))
>>> pattern = Chem.MolFromSmiles('C(=O)O') # Has CO2 group
>>> for mol in mol_list:
>>>     print(mol.HasSubstructMatch(pattern))
>>> pattern = Chem.MolFromSmiles('CC(N)C') # Has ???
>>> for mol in mol_list:
>>>     print(mol.HasSubstructMatch(pattern))
```

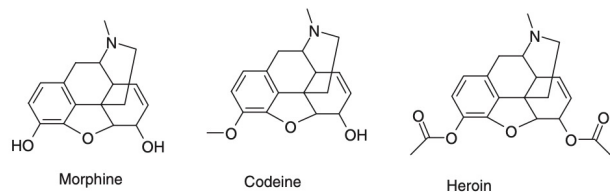


rdkit\_basics.ipynb

24

## Similarity Methods

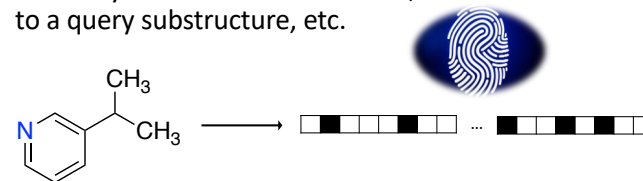
- Many molecules are similar. But how can we quantify this?
- Compounds active at opioid receptors:



25

## Fingerprints

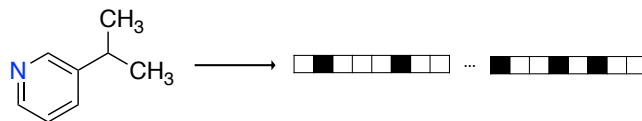
- Molecular fingerprints are a way of encoding the structure of a molecule. The most common type of fingerprint is a series of binary digits (bits) that represent the presence or absence of particular substructures in the molecule.
- Comparing fingerprints allows you to determine the similarity between two molecules, to find matches to a query substructure, etc.



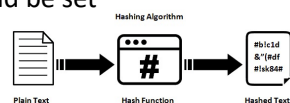
26

## Fingerprints

- Idea: Apply a function to a molecule to generate a bit vector



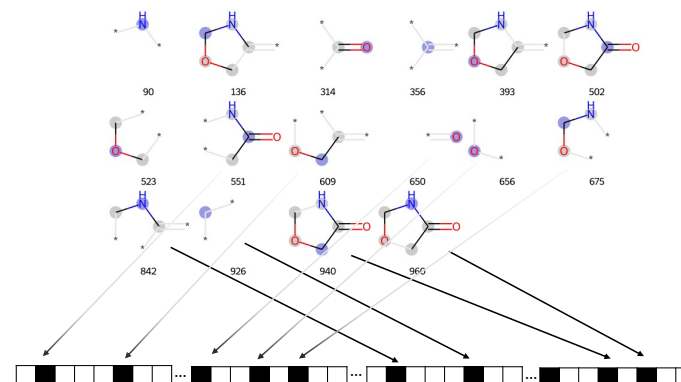
- Typical kernels extract features of the molecule, hash them, and use the hash to determine bits that should be set



- Typical fingerprint sizes: 1K-4K bits

27

## Fingerprints



28

## Fingerprints

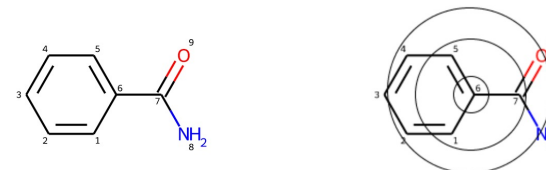
- Extended Connectivity FingerPrinting (ECFP)
  - ECFPs are a type of molecular fingerprint explicitly designed to capture molecular features relevant to molecular activity.
  - ECFPs are among the most popular similarity search tools in drug discovery and they are effectively used in a wide variety of applications.

Rogers, D.; Hahn, M. J. *Chem. Inf. Model.* **2010**, *50*, 742–754

29

## ECFP Example

- We want to create an ECFP fingerprint for benzamide
  - As an example, we'll look at C6



Rogers, D.; Hahn, M. J. *Chem. Inf. Model.* **2010**, *50*, 742–754

30

## ECFP Example

- Step 1 - Initial Stage:** First, each atom in the molecule is assigned a unique integer identifier. This identifier is generated by hashing a combination of properties like atomic numbers, atomic mass, etc.



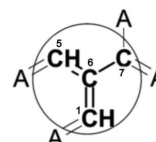
Iteration 0

- At the beginning, the initial atom identifier only represents information about the atom itself and its attached bonds.

31

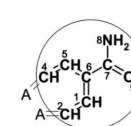
## ECFP Example

- Step 2 - Update Stage:** Next, each atom collects its identifier and the identifiers of its immediately neighboring atoms, into an array. A hash function is applied to reduce this array back into a new, single-integer identifier. This step is done to capture the neighborhood of the atom



Iteration 1

After one iteration, the identifier now contains information about atom 6's immediate neighbors.



Iteration 2

After two iterations, the identifier now contains information about atom 6's neighbors' neighbors.

32



## ECFP Example



- **Step 3 - De-duplication Stage:** The third stage removes the duplicate features from our generated feature list.
- **Step 4 - Forming the bit vector:** Once all the identifiers are calculated for a specified number of iterations, the final step is to reduce these identifiers into a bit vector.
- This process (1-4) is repeated for all the atoms in the molecule
- One by one, each atom is chosen as a center and then the same process is repeated.

33

## ECFP Comments



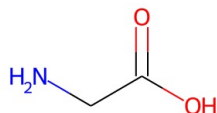
- Hydrogen atoms and bonds to hydrogen atoms are ignored
- Number of iterations depends on the desired use of the fingerprint. Typically, **two iterations** are sufficient for fingerprints that will be used for **similarity or clustering**, while **activity learning** methods often need greater structure detail, which is available after **three or even four or more iterations**.

34

## Fingerprints in RDKit



```
>>> smiles_list = ['C(C(=O)O)N',
                   'C1=CC=C(C=C1)C[C@@H](C(=O)O)N',
                   'C1=C(NC=N1)C[C@@H](C(=O)O)N', 'C([C@@H](C(=O)O)N)S']
>>> mol_list = []
>>> for smiles in smiles_list:
    mol = Chem.MolFromSmiles(smiles)
    mol_list.append(mol)
>>> glycine = mol_list[0]
>>> glycine
```



rdkit\_fingerprints.ipynb

35

## Fingerprints in RDKit



```
>>> fp =
    AllChem.GetMorganFingerprintAsBitVect(glycine, 2, nBits=1024)
>>> fp_arr = np.zeros((1,))
>>> print(fp_arr)
[0. 0. 0. ... 0. 0. 0.]
>>> DataStructs.ConvertToNumpyArray(fp, fp_arr)
>>> np.nonzero(fp_arr)
(array([ 27,  80, 147, 389, 650, 713, 807, 893, 966, 981]),)
```

rdkit\_fingerprints.ipynb

36

## Fingerprints in RDKit



```
>>> bi = {}
>>> fp =
    AllChem.GetMorganFingerprintAsBitVect(glycine, 2,
                                           nBits=1024, bitInfo=bi)

>>> fp_arr = np.zeros((1,))
>>> DataStructs.ConvertToNumpyArray(fp, fp_arr)
>>> np.nonzero(fp_arr)
>>> list(fp.GetOnBits())
[27, 80, 147, 389, 650, 713, 807, 893, 966, 981]
```

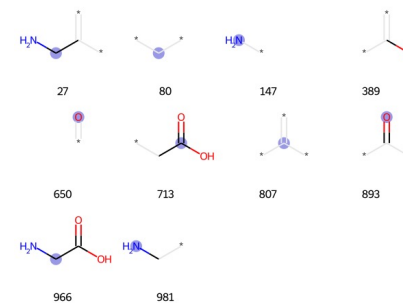
rdkit\_fingerprints.ipynb

37

## Fingerprints in RDKit



```
>>> prints = [(glycine, x, bi) for x in fp.GetOnBits()]
>>> legends = [str(x) for x in fp.GetOnBits()]
>>> Draw.DrawMorganBits(prints, molPerRow=4, legends=legends)
```



rdkit\_fingerprints.ipynb

38

## Fingerprints in RDKit



```
>>> cysteine = mol_list[3]
>>> img = Draw.MolsToGridImage([glycine, cysteine], molPerRow=4)
>>> img

>>> fp2 = AllChem.GetMorganFingerprintAsBitVect(cysteine, 2,
                                                nBits=1024, bitInfo=bi)

>>> print('Cys', list(fp2.GetOnBits()))
Cys [1, 48, 80, 147, 229, 321, 389, 403, 435, 650, 786, 807, 820, 825, 893, 902]
>>> print('Gly', list(fp.GetOnBits()))
Gly [27, 80, 147, 389, 650, 713, 807, 893, 966, 981]
```

rdkit\_fingerprints.ipynb



39

## 2D-Based Similarity



### Tanimoto similarity

- Similarity between two molecules is represented by 2D binary fingerprints
- The Tanimoto coefficient gives a measure of the number of fragments in common between the two molecules:

$$S_{AB} = \frac{c}{a + b - c}$$

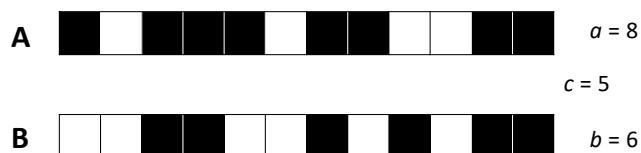
$S_{AB} \in [0, 1]$

Bits common to A and B (points to  $c$ )  
 Bits set to "1" in molecule A (points to  $a$ )  
 Bits set to "1" in molecule B (points to  $b$ )  
 100% similarity (points to 1)  
 0% similarity (points to 0)

40

## 2D-Based Similarity

Tanimoto similarity example:  $S_{AB} = \frac{c}{a + b - c}$



$$S_{AB} = \frac{5}{8 + 6 - 5} = 0.56$$

41

## 2D-Based Similarity

Additional similarity scores:

- Tanimoto coefficient

$$S_{AB} = \frac{c}{a + b - c} \quad S_{AB} \in [0,1]$$

- Dice coefficient:

$$S_{AB} = \frac{2c}{a + b} \quad S_{AB} \in [0,1]$$

- Tversky coefficient:

$$S_{AB} = \frac{c}{\alpha(a - c) + \beta(b - c) + c} \quad S_{AB} \in [0,1]$$

42

## 2D-Based Similarity

Additional similarity scores:

- Euclidean distance:

$$S_{AB} = \sqrt{a + b - 2c} \quad S_{AB} \in [0, N]$$

- Manhattan Distance

$$S_{AB} = a + b - 2c \quad S_{AB} \in [0, N]$$

$S_{AB} \in [0, N]$

Vector length

$S_{AB} \in [0, N]$

43

## 2D-Based Similarity

```
>>> print('Cys', list(fp2.GetOnBits()))
Cys [1, 48, 80, 147, 229, 321, 389, 403, 435, 650, 786, 807, 820, 825, 893, 902]
>>> print('Gly', list(fp.GetOnBits()))
Gly [27, 80, 147, 389, 650, 713, 807, 893, 966, 981]
>>> common = set(fp.GetOnBits()) & set(fp2.GetOnBits())
>>> print(common)
{389, 807, 650, 80, 147, 893}
>>> combined = set(fp.GetOnBits()) | set(fp2.GetOnBits())
>>> print(combined)
{1, 321, 389, 966, 902, 713, 650, 80, 786, 147, 403, 981, 27, 229, 807, 48, 435, 820, 825, 893}
>>> print('Similarity:', len(common)/len(combined))
Similarity: 0.3
>>> print('Similarity:', DataStructs.TanimotoSimilarity(fp, fp2))
Similarity: 0.3
```

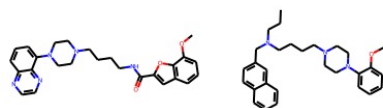
rdkit\_fingerprints.ipynb

44

## Similarity Maps



```
>>> mol = Chem.MolFromSmiles(
    'COc1cccc2cc(C(=O)NCCCN3CCN(c4cccc5nccnc54)CC3)oc21')
>>> refmol = Chem.MolFromSmiles(
    'CCCN(CCCCN1CCN(c2cccc2OC)CC1)Cc1ccc2ccccc2c1')
>>> mol_list = [mol, refmol]
>>> img = Draw.MolsToGridImage(mol_list, molsPerRow=2)
>>> img
```



rdkit\_fingerprints\_advanced.ipynb

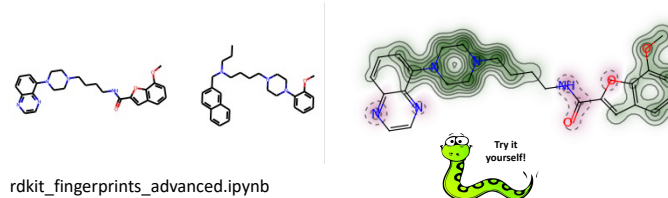
45

## Similarity Maps



```
>>> from rdkit.Chem.Draw import SimilarityMaps

>> # Use either of these
>>> fp = SimilarityMaps.GetAPFingerprint(mol, fpType='normal')
>>> fp = SimilarityMaps.GetTTFingerprint(mol, fpType='normal')
>>> fp = SimilarityMaps.GetMorganFingerprint(mol, fpType='bv')
>>> fig, maxweight =
    SimilarityMaps.GetSimilarityMapForFingerprint(refmol, mol,
    SimilarityMaps.GetMorganFingerprint)
```



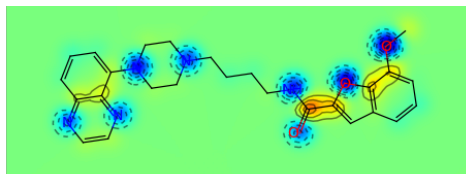
rdkit\_fingerprints\_advanced.ipynb

46

## Similarity Maps



```
>>> AllChem.ComputeGasteigerCharges(mol)
>>> contribs =
    [mol.GetAtomWithIdx(i).GetDoubleProp('_GasteigerCharge')
     for i in range(mol.GetNumAtoms())]
>>> fig = SimilarityMaps.GetSimilarityMapFromWeights(mol,
    contribs, colorMap='jet', contourLines=10)
```



rdkit\_fingerprints\_advanced.ipynb

47