Artificial Intelligence and Augmented Intelligence for Automated Investigations for Scientific Discovery

High-throughput generation of structural isomers for fast development of molecular datasets to train machine learning algorithms

Project Report
[Project Dates: 24/06/2021 - 09/09/2021]
Swansea University

Project Student: Anna Catton, Swansea University
Supervised by: Dr Francisco Martin-Martinez, Swansea University

Report Date: 17/09/2021

AI3SD-Intern-Series:Report-10_Catton

High-throughput generation of structural isomers for fast development of molecular datasets to train machine learning algorithms

AI3SD- Intern-Series:Report-10_Catton
Report Date: 17/09/2021
DOI: 10.5258/SOTON/AI3SD0141
Published by University of Southampton

Network: Artificial Intelligence and Augmented Intelligence for Automated Investigations for Scientific Discovery

This Network+ is EPSRC Funded under Grant No: EP/S000356/1
Principal Investigator: *Professor Jeremy Frey*
Co-Investigator: *Professor Mahesan Niranjan*
Network+ Coordinator: *Dr Samantha Kanza*

Table of Contents

1. Project Details

Title	High-throughput generation of structural isomers for fast development of molecular datasets to train machine learning algorithms
Project Reference	AI3SD-FundingCall3_011
Supervisor Institution	Swansea University
Project Dates	24/06/2021 - 09/09/2021
Website	https://www.martinmartinezlab.com/
Keywords	Computational Chemistry, Big data, DFT, Isomers

2. Project Team

2.1 Project Student

Name and title **Miss Anna Catton**	
Employer name/University Department Name **Swansea University, Chemistry Department**	
Work Email **1902838@swansea.ac.uk**	**Website Link** **https://www.martinmartinezlab.com/**

2.2 Project Supervisor

Name and title **Dr Francisco Martin-Martinez**	
Employer name/University Department Name **Swansea University, Chemistry Department**	
Work Email **f.j.martin-martinez@swansea.ac.uk**	**Website Link** **https://www.martinmartinezlab.com/**

2.3 Researchers & Collaborators

The work built on some previous work by Dan York, a Master student at Swansea University who provided the xyz files for the fast and isothermal hydrothermal liquefaction (HTL) molecules that have been used in the case study. Additionally, we also establish an international collaboration with Isaac Vidal Daza, researcher at the University of Granada (Spain), who advised on python programming and discussed the code.

3. Lay Summary

The search for new molecules to fit any potential application, or the identification of chemical species in complex fluid materials, are problems that can be largely solved by a computational optimization of a molecular design space before any experimental developments. Nowadays, such optimized molecular design spaces constitute structural datasets for the training of machine learning (ML) algorithms that can further accelerate molecular discovery. In this project, we have used the python programming language to automate the generation of structural isomers for any given molecule, as well as to perform high throughput DFT calculations in Supercomputing Wales (SCW) computational facilities. The python code developed here converts initial cartesian coordinates of the atoms in any molecule (.xyz files) into machine-readable SMILES-format molecular structures that are easy to process by a computer. Then, the code generates all possible structural isomers and stereo isomers and stores them into input files for the ORCA computational package to perform DFT calculations that can then provide the lowest energy molecular structure and the associated electron density of each molecule. The DFT results will constitute a series of datasets for training ML algorithms in the future. The method will also provide the most stable molecular structures among possible isomers, which forms an optimized atomistic model for the complex fluid of interest. This model can be used for further computational studies, like molecular dynamics simulations.

4. Aims and Objectives

The aim of this project was to develop a high-throughput computational method that generates fit-for-purpose structural isomers of any given molecule and use a python-based code to automate the computational optimization of all molecular geometries, as well as the calculation of their associated electronic structure and molecular energies using density functional theory (DFT) methods. The objectives were:

- Convert .xyz files in SMILES molecular format
- Generate the SMILES of all possible structural isomers and stereoisomers for a molecule
- Store the SMILES for their utilization in DFT calculations
- Develop a code to work with several molecules automatically and simultaneously
- Integrate all pieces of code into a one single and easy-to-use program
- Test the code against a model for biocrude oils from fast and isothermal HTL

The initial motivation was the development of better models for complex fluids. The challenges associated to the characterization of molecular species in materials like biocrude oils or asphalt, makes it very difficult for the development of atomistic models, since the detailed chemical composition is largely unknown. Only some average molecules and/or molecular classes are usually described, which was used to set the starting point for this project. Thus, starting from some suggested molecules that have been experimentally characterized in different types of biocrude oils (i.e., fast, and isothermal HTL-derived oils), we have developed a code that generates all possible isomers and identifies those that are the most stable using DFT calculations. The new stable isomers can constitute a refined atomistic model, based on the initial characterization, but improved by this computational procedure. As a case study, and to test the python code, the method was applied to the development of molecular models of biocrude oils derived from the HTL of biomass, a case of particular interest among complex fluids, given their potential as sustainable alternatives to fossil fuels, or asphalt bitumen, while providing an avenue for waste utilization.

5. Methodology

As mentioned before, python has been the selected programming language for the project, given its versatility and also its intrinsic focus in data science and scientific applications. RDKit Python Cheminformatics Tools [1] and the python Atomic Simulation Environment (ASE) [2] was used to develop the code for these high-throughput generation of structural isomers. The RDKit package was very useful with working alongside the SMILES and with the generation of the isomers, as it contained a variety of chemistry specific commands which posed most useful. The ASE tool was used to conduct the ORCA calculations within. Through using the ASE environment, ORCA was easily called to hold the calculations, whilst the conditions for the basis set and functional could also be easily inputted. This meant that the

ASE tool made the entire end programme more user friendly, whilst also increasing the overall efficiency of the calculations.

The basis of the code came from a github repository, which was further expanded and integrated with other pieces of code to build the intended program. This xyz-to-SMILES code came from a section of code designed to convert .xyz files into molecular images [3], and it was adapted also to integrate well with other sections of code I compiled. The finding of this available code enabled the start of an integrated program that was then worked through and adapted to reach the end goal of the project. When trying to produce a section of code to help with the stereoisomer generation, a section of RDKit posed most useful with the commands that could be used throughout this section of python code. [4]

The python code was initially developed with Jupyter Lab, used within Anaconda. This was chosen so the code could be easily developed and tested in small sections, thus reducing the amount of time needed to debug and searching for errors to fix. It also allowed easy sharing and co-development if needed. Once the code was able to run smoothly, it was exported to .py files and these were run using Visual Studio Code, which was found very good for development and synchronization with github. After all this process, the code was in the right format to be submitted into the supercomputer at Supercomputing Wales (SCW).

The DFT calculations where set-up also with python code, using the ASE and the calculators available within this python package. DFT is a quantum-chemistry-based method that provides the energy and electronic structure of any given molecule. It requires the selection of a so-called functional, which is the specific method used to calculate the energy, and a basis set, which is a set of functions used to approximate the molecular orbitals calculated during the process. We selected B3LYP as functional because it is already a standard in the field, and the double zeta 6-31G as basis set, which is usually combined with B3LYP for standard DFT calculations. The functional and basis set can be easily change in the calculator part of the ASE code, so we just used B3LYP/6-31G as the initial test.[5]

6. Results

The intended code was fully developed and two types of biocrude oil were tested, those coming from fast HTL, which is at a higher temperature, and those coming from isothermal HTL, which occurs at a lower, more constant temperature.[6] Building on the available xyz files for both fast and isothermal, we developed the code to generate the isomers for the molecules constituting both of these biocrude oils. The xyz files were generated from the available experimental analysis of the biocrude oil.

First, an initial python script was generated to allow a 2D visualisation of the molecules. This meant to develop a code that took the xyz files as input, converted them to SMILES-format and then output a visual representation of all the molecules. In a second step, the SMILES that were previously generated became the input for two separate python scripts that produced the isomers: one to produce the stereoisomers of all the molecules, and the other to produce all of the structural isomers. The two set of SMILES were then stored in python lists and combined into one that contained all the isomers. The two lists were also checked for duplicates, to make sure that the two scripts used for isomers generation were not generating some redundant geometries. This would increase the efficiency of the DFT calculations as well, by reducing the number of molecules having geometries calculated that were redundant. The search for these duplicates confirmed that there were not redundant SMILES (apart from the original starting geometries) and therefore the list with the total number of isomers contained indeed all possible isomers, without any duplicates. The rdkit package in python was utilised continuously throughout the project, as it allowed the chemical functions to be easily imported and performed; especially when working with the SMILES. The list of SMILES was then input into another part of the code to allow the density functional theory (DFT) calculations to be performed. ASE and its calculator for ORCA was utilised.

There were many challenges throughout this project, as it implied a first main exposure to both coding and python. Learning how to use the different programmes was challenging at times, but very rewarding when these challenges were overcome. The basis of building the codes was made simpler through the use of Jupyter Lab, which enabled to run small sections of the code to help minimise time spent solving errors. When combining all the scripts into one to make for easier use, there were challenges encountered, this meant different varieties of approaches were trialled before the end script was developed. To effectively generate this code, and for making a more organized program, a separate python script containing all of the functions for all the scripts was created, meaning this script could be imported at the start of the main code, keeping it tidy and more user friendly. This meant that you could easily change the inputs and see where changes needed to be made in the script calculating the isomers and then inputting them into the supercomputer, without worrying about function definitions.

As we ran out of time towards the end of the project, we encountered unexpected external challenges associated with the set-up of the SCW supercomputing environment. Thus, calculating the stability of all the molecules by running DFT calculations was not possible. However, the python code was ready, and everything set up for computing. Therefore, the isomers being generated was the end point of the project, although it is planned on following this research through, and inputting the code into the SCW computer as soon as the required python environment is available. The isomers generated for the fast HTL molecules can be seen below in figure 1. This shows the variety of isomers generated, as well as the removal of the duplicates that occur. Duplicates occur through the fact that both stereoisomers and structural isomers were generated, and the input molecules therefore appeared in both lists produced, as mentioned before. Once the duplicates had been highlighted, it showed there were no redundant isomers produced, emphasising that the list we had obtained contained all of the isomers produced.

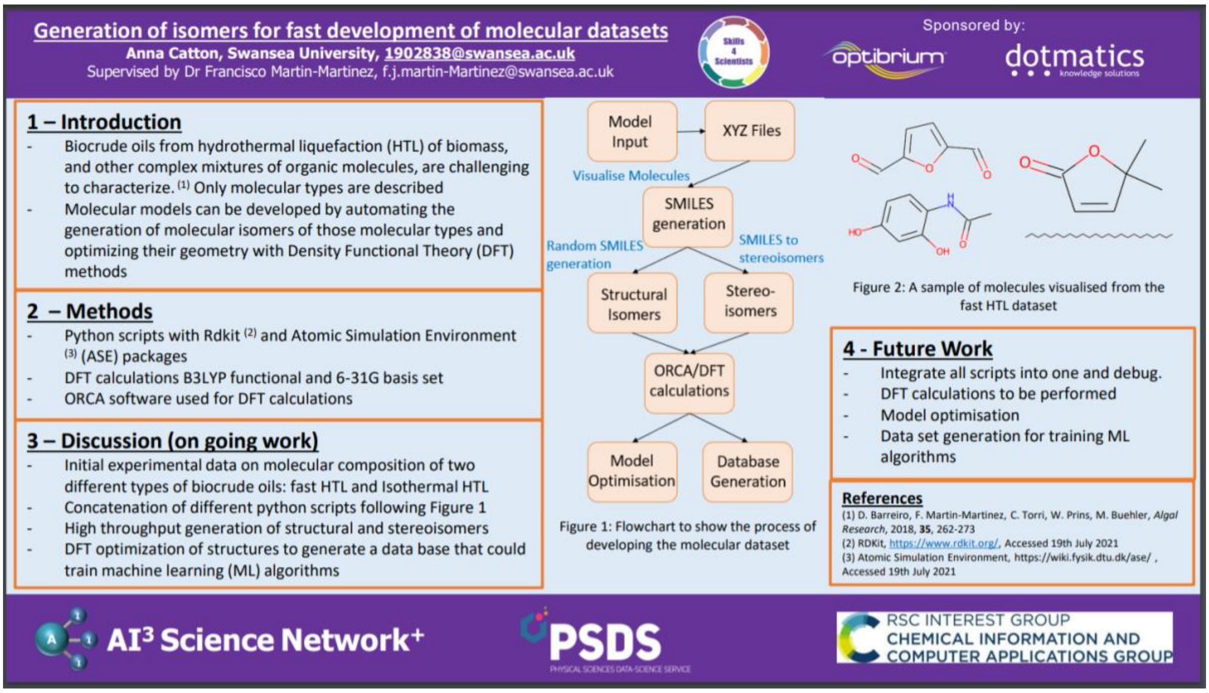
Within the end section of the report, under 'outputs, data and software links' screenshots can be seen of all the isomers generated from each molecule within the fast hydrothermal liquefaction data set that we used. These isomers will then be inputted into the supercomputer, to generate the outcome data value. There are large numbers of isomers generated, showing how long the calculations may take to perform. The isomer generation showed the large variety of different isomers of each molecule that could be present within the biocrude oil model, therefore, through looking at all of the isomers and performing the calculations a significantly optimised model can then be produced to improve on the future work of the project.

The isomers could all be generated without the need for the supercomputer, however, the calculations required too much computational power to be able to run from an individual computer. Once the calculations have been performed, the research will be updated to show the most stable forms of the original molecules.

7. Conclusions & Future Work

From the process of the project, a python program to easily generate all possible isomers for any given molecule has been developed. The code can import the xyz coordinates of all the molecules in a molecular set, transform them into SMILES, a machine readable format, and generate all possible isomers, thus expanding the initial molecular set. The code wasn't tested against the case of biocrude oils generated from fast and isothermal HTL, but it applies to any molecular data set, which makes it very powerful and transferable. Because of the application of the code to the bio crude cases study, a comprehensive list of all the biocrude isomers was generated as previously stated. Finally, the code was cleaned and rearranged in a well-structured way, with commented along the code, so anyone in the future can use it as well as expanding upon it. It is also in a github repository, opened to the community. This code will be taken forward to be used in several projects headed by Dr Francisco Martin-Martinez at his research group. In particular, the code will be used to generate better models for biocrude oils coming from different biomass sources, but also to identify potential alternative molecules to initial candidates selected in solar energy, and energy storage.

8. Outputs, Data & Software Links



Generation of isomers for fast development of molecular datasets

Anna Catton, Swansea University, 1902838@swansea.ac.uk

Supervised by Dr Francisco Martin-Martinez, f.j.martin-Martinez@swansea.ac.uk

Skills 4 Scientists

Sponsored by:

optibrium
dotmatics knowledge solutions

1 – Introduction

- Biocrude oils from hydrothermal liquefaction (HTL) of biomass, and other complex mixtures of organic molecules, are challenging to characterize. (1) Only molecular types are described
- Molecular models can be developed by automating the generation of molecular isomers of those molecular types and optimizing their geometry with Density Functional Theory (DFT) methods

2 – Methods

- Python scripts with Rdkit (2) and Atomic Simulation Environment (3) (ASE) packages
- DFT calculations B3LYP functional and 6-31G basis set
- ORCA software used for DFT calculations

3 – Discussion (on going work)

- Initial experimental data on molecular composition of two different types of biocrude oils: fast HTL and Isothermal HTL
- Concatenation of different python scripts following Figure 1
- High throughput generation of structural and stereoisomers
- DFT optimization of structures to generate a data base that could train machine learning (ML) algorithms

Model Input
XYZ Files
Visualise Molecules
SMILES generation
Random SMILES generation
SMILES to stereoisomers
Structural Isomers
Stereo-isomers
ORCA/DFT calculations
Model Optimisation
Database Generation

Figure 1: Flowchart to show the process of developing the molecular dataset

Figure 2: A sample of molecules visualised from the fast HTL dataset

4 - Future Work

- Integrate all scripts into one and debug.
- DFT calculations to be performed
- Model optimisation
- Data set generation for training ML algorithms

References

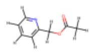
(1) D. Barreiro, F. Martin-Martinez, C. Torri, W. Prins, M. Buehler, *Algal Research*, 2018, **35**, 262-273
(2) RDKit, https://www.rdkit.org/, Accessed 19th July 2021
(3) Atomic Simulation Environment, https://wiki.fysik.dtu.dk/ase/ , Accessed 19th July 2021

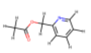
AI³ Science Network⁺
PSDS PHYSICAL SCIENCES DATA-SCIENCE SERVICE
RSC INTEREST GROUP CHEMICAL INFORMATION AND COMPUTER APPLICATIONS GROUP

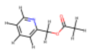
Poster Presentation

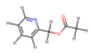
Github repository: https://github.com/fjmartinmartinez/anna_catton_ai3n

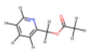
Screenshots from the output of the isomer generation:

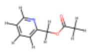

C(C)(OCc1ccccn1)=O

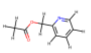

c1c(COC(=O)C)nccc1

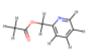

C(c1ccccn1)OC(C)=O

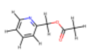

c1ccnc(COC(=O)C)c1

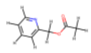

C(OCc1ccccn1)(C)=O

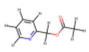

C(=O)(C)OCc1ccccn1

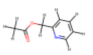

c1c(COC(C)=O)nccc1

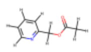

c1ccc(COC(C)=O)nc1

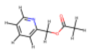

c1cccnc1COC(C)=O

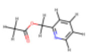

O=C(C)OCc1ccccn1

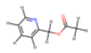

c1(COC(C)=O)ncccc1

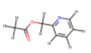

c1nc(COC(=O)C)ccc1

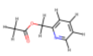

c1(COC(=O)C)ncccc1

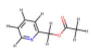

C(OC(=O)C)c1ccccn1

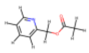

c1cnc(COC(=O)C)cc1

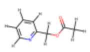

C(OC(C)=O)c1ccccn1

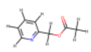

c1cc(COC(C)=O)ncc1

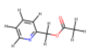

c1nc(COC(C)=O)ccc1

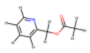

n1ccccc1COC(C)=O

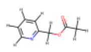

C(C)(=O)OCc1ccccn1

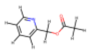

C(OC(C)=O)c1ncccc1

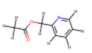

C(OCc1ncccc1)(=O)C

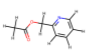

C(=O)(OCc1ncccc1)C

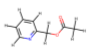

O(Cc1ccccn1)C(=O)C

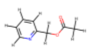

C(OC(=O)C)c1ncccc1

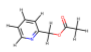

c1ccnc(COC(C)=O)c1

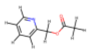

c1ccc(COC(=O)C)nc1

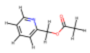

c1cc(COC(=O)C)ncc1

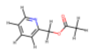

C(c1ncccc1)OC(=O)C

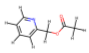

C(C)(=O)OCc1ncccc1

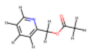

O=C(OCc1ncccc1)C

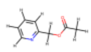

O=C(OCc1ccccn1)C

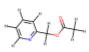

C(=O)(OCc1ccccn1)C

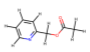

C(OCc1ccccn1)(=O)C

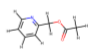

c1(COC(C)=O)ccccn1

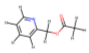

O(C(=O)C)Cc1ccccn1

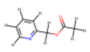

c1cccc(COC(=O)C)n1

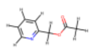

c1cccc(COC(C)=O)n1

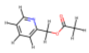

CC(=O)OCc1ncccc1

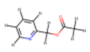

c1cccnc1COC(=O)C

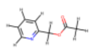

O(C(C)=O)Cc1ccccn1

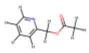

C(OCc1ncccc1)(C)=O

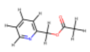

O(C(C)=O)Cc1ncccc1

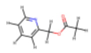

CC(=O)OCc1ccccn1

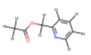

C(c1ncccc1)OC(C)=O

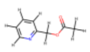

C(=O)(C)OCc1ncccc1

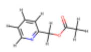

c1(COC(=O)C)ccccn1

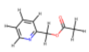

n1ccccc1COC(=O)C

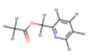

C(c1ccccn1)OC(=O)C

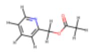

c1cnc(COC(C)=O)cc1

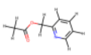

O(Cc1ncccc1)C(C)=O

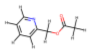

C(C)(OCc1ncccc1)=O

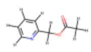

O=C(C)OCc1ncccc1

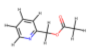

n1c(COC(C)=O)cccc1

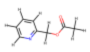

O(Cc1ccccn1)C(C)=O

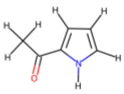

n1c(COC(=O)C)cccc1


CC(OCc1ccccn1)=O

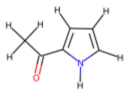

CC(OCc1ncccc1)=O


O(Cc1ncccc1)C(=O)C

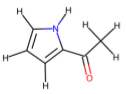

O(C(=O)C)Cc1ncccc1



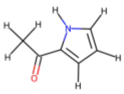
C(c1[nH]ccc1)(=O)C



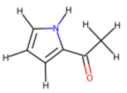
CC(c1[nH]ccc1)=O



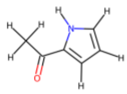
c1cc(C(=O)C)[nH]c1



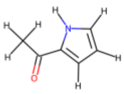
c1cc[nH]c1C(C)=O



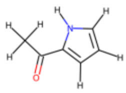
c1cc(C(C)=O)[nH]c1



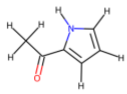
c1cc[nH]c1C(=O)C



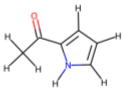
C(c1ccc[nH]1)(C)=O



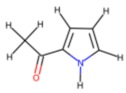
O=C(c1ccc[nH]1)C



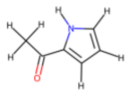
CC(=O)c1ccc[nH]1



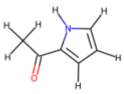
[nH]1cccc1C(C)=O



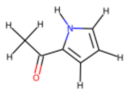
C(=O)(C)c1[nH]ccc1



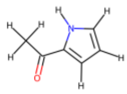
C(=O)(c1ccc[nH]1)C



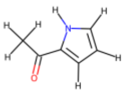
c1c[nH]c(C(=O)C)c1



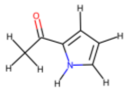
O=C(C)c1ccc[nH]1



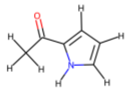
C(C)(=O)c1ccc[nH]1



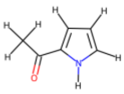
C(c1ccc[nH]1)(=O)C



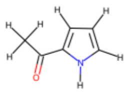
c1ccc(C(C)=O)[nH]1



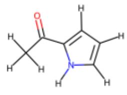
[nH]1cccc1C(=O)C



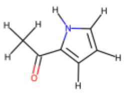
O=C(c1[nH]ccc1)C



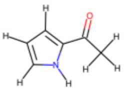
C(=O)(c1[nH]ccc1)C



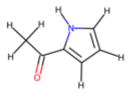
c1ccc(C(=O)C)[nH]1



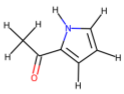
c1(C(=O)C)ccc[nH]1



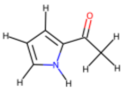
c1[nH]c(C(=O)C)cc1



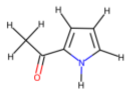
C(C)(c1ccc[nH]1)=O



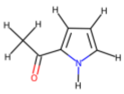
C(=O)(C)c1ccc[nH]1



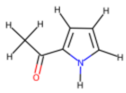
c1[nH]c(C(C)=O)cc1



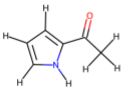
c1(C(C)=O)[nH]ccc1



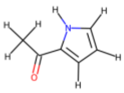
O=C(C)c1[nH]ccc1



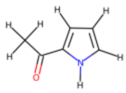
CC(=O)c1[nH]ccc1



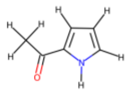
[nH]1c(C(=O)C)ccc1



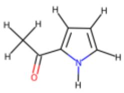
c1(C(C)=O)ccc[nH]1



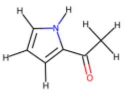
c1(C(=O)C)[nH]ccc1



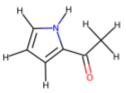
C(C)(c1[nH]ccc1)=O



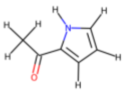
C(c1[nH]ccc1)(C)=O



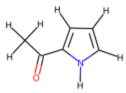
c1c(C(C)=O)[nH]cc1



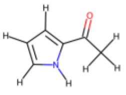
c1c(C(=O)C)[nH]cc1



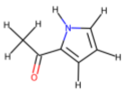
c1c[nH]c(C(C)=O)c1



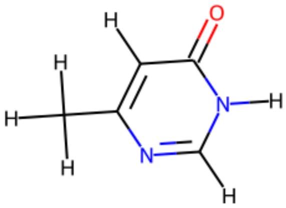
C(C)(=O)c1[nH]ccc1



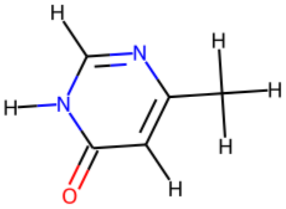
[nH]1c(C(C)=O)ccc1



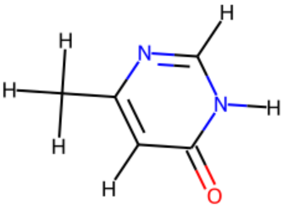
CC(c1ccc[nH]1)=O



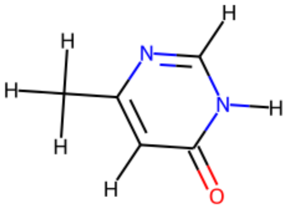
c1nc(C)cc(=O)[nH]1



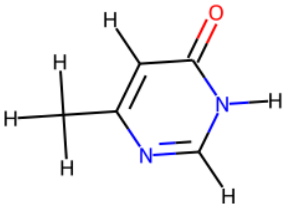
c1(=O)[nH]cnc(C)c1



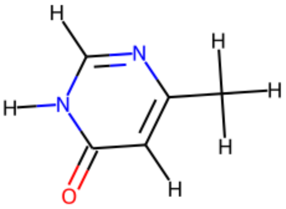
O=c1cc(C)nc[nH]1



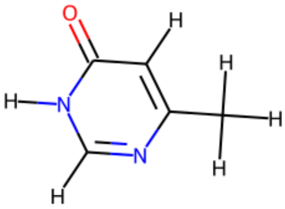
c1(=O)cc(C)nc[nH]1



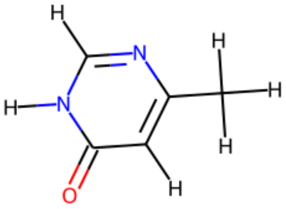
c1(C)cc(=O)[nH]cn1



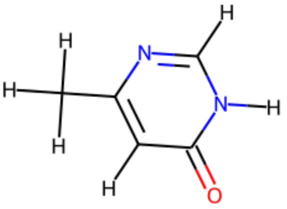
c1c(=O)[nH]cnc1C



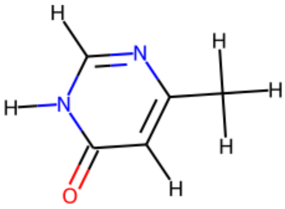
Cc1nc[nH]c(=O)c1



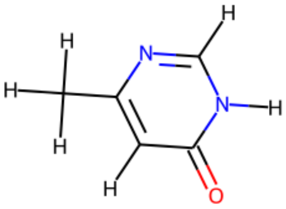
O=c1[nH]cnc(C)c1



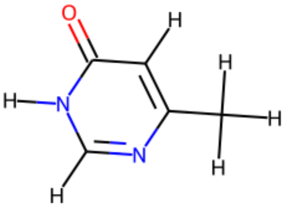
c1c(C)nc[nH]c1=O



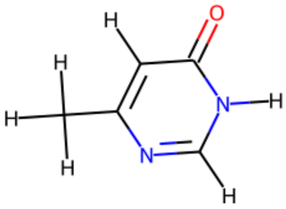
[nH]1cnc(C)cc1=O



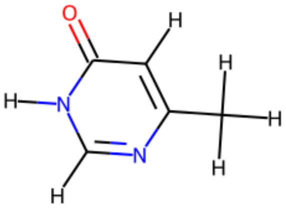
[nH]1c(=O)cc(C)nc1



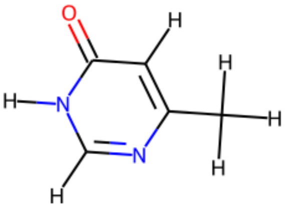
n1c[nH]c(=O)cc1C



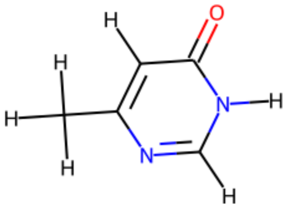
n1c(C)cc(=O)[nH]c1



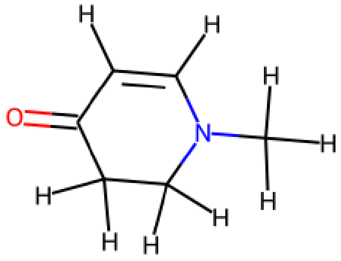
c1[nH]c(=O)cc(C)n1



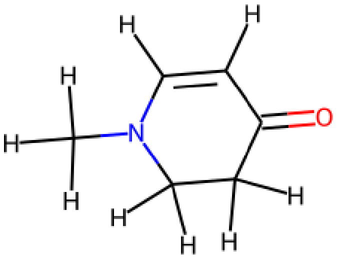
c1(C)nc[nH]c(=O)c1



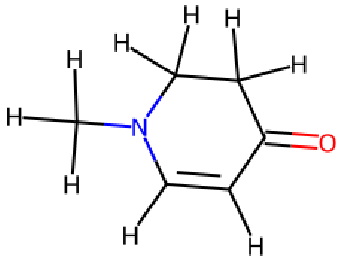
Cc1cc(=O)[nH]cn1



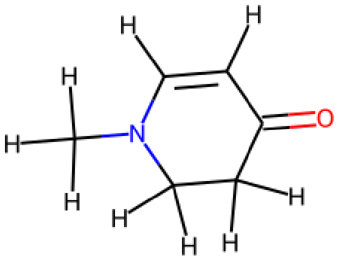
C1CC(=O)C=CN1C



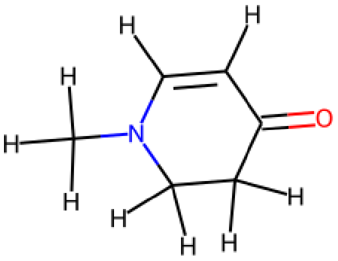
CN1C=CC(=O)CC1



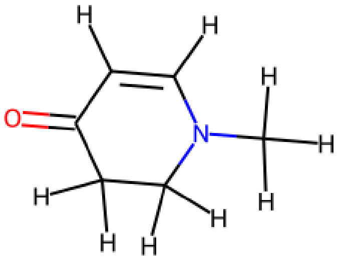
O=C1C=CN(C)CC1



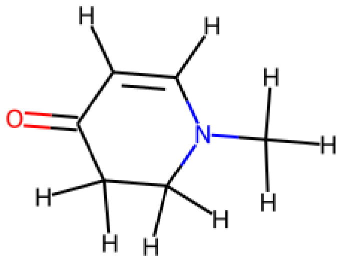
N1(C)C=CC(=O)CC1



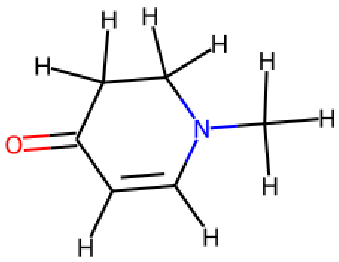
C1N(C)C=CC(=O)C1



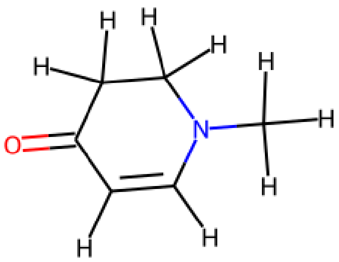
N1(C)CCC(=O)C=C1



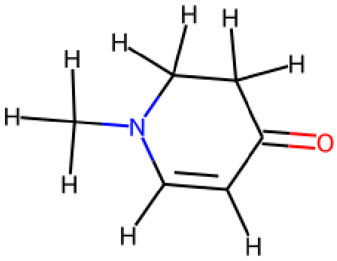
C1C(=O)C=CN(C)C1



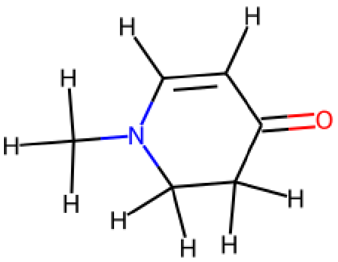
O=C1CCN(C)C=C1



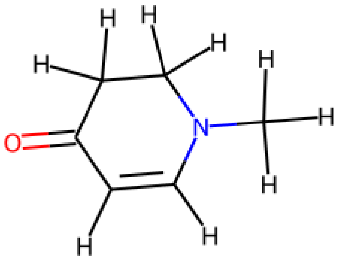
C1(=O)CCN(C)C=C1



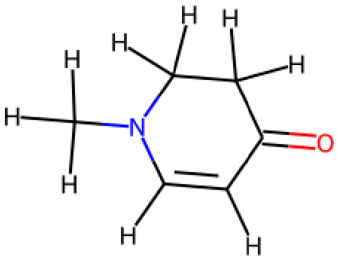
C1(=O)C=CN(C)CC1



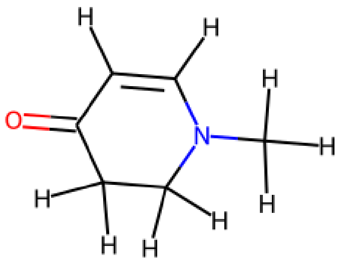
C1CN(C)C=CC1=O



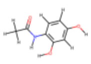
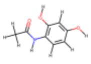
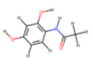
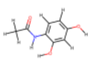
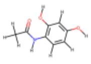
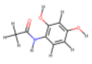
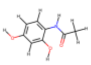
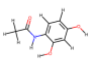
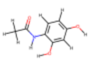
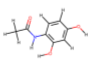
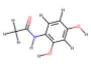
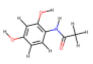
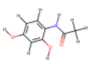
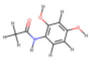
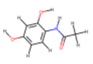
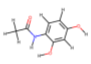
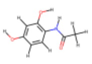
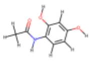
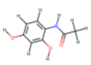
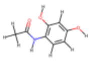
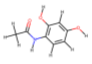
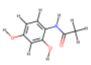
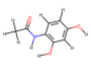
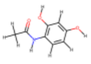
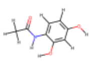
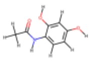
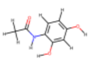
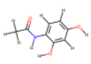
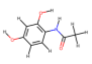
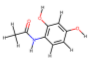
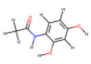
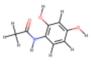
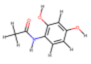
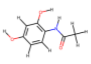
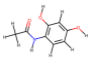
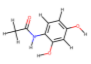
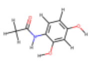
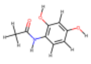
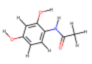
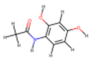
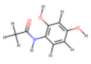
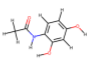
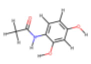
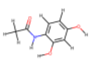
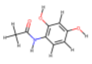
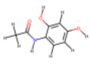
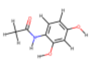
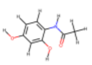
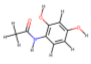
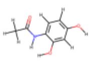
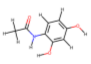
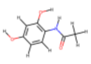
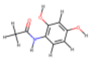
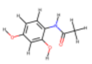
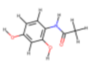
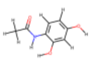
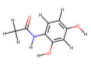
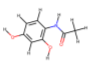
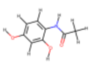
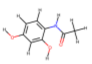
C1=CC(=O)CCN1C

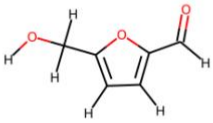


C1=CN(C)CCC1=O

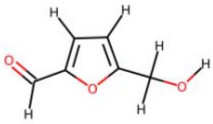


CN1CCC(=O)C=C1

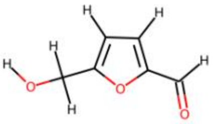

Oc1c(NC(=O)C)ccc(O)c1

c1(NC(C)=O)c(O)cc(O)cc1

Oc1cc(O)c(NC(=O)C)cc1

C(Nc1ccc(O)cc1O)(C)=O

c1cc(NC(=O)C)c(O)cc1O

O=C(C)Nc1c(O)cc(O)cc1

Oc1ccc(NC(=O)C)c(O)c1

c1c(O)c(NC(=O)C)ccc1O

O=C(Nc1ccc(O)cc1O)C

c1(O)c(NC(C)=O)ccc(O)c1

c1(O)c(NC(=O)C)ccc(O)c1

Oc1cc(O)c(NC(C)=O)cc1

Oc1cc(O)ccc1NC(=O)C

N(c1c(O)cc(O)cc1)C(C)=O

c1c(O)cc(O)c(NC(C)=O)c1

N(C(C)=O)c1ccc(O)cc1O

c1cc(O)cc(O)c1NC(=O)C

N(C(C)=O)c1c(O)cc(O)cc1

c1(O)ccc(NC(C)=O)c(O)c1

C(Nc1c(O)cc(O)cc1)(=O)C

C(=O)(Nc1c(O)cc(O)cc1)C

c1(O)cc(O)ccc1NC(=O)C

Oc1c(NC(C)=O)ccc(O)c1

O=C(Nc1c(O)cc(O)cc1)C

C(=O)(C)Nc1ccc(O)cc1O

c1c(NC(C)=O)c(O)cc(O)c1

CC(=O)Nc1ccc(O)cc1O

N(c1ccc(O)cc1O)C(=O)C

c1(O)cc(O)c(NC(C)=O)cc1

C(=O)(C)Nc1c(O)cc(O)cc1

N(c1ccc(O)cc1O)C(C)=O

C(C)(Nc1c(O)cc(O)cc1)=O

N(c1c(O)cc(O)cc1)C(=O)C

c1cc(O)cc(O)c1NC(C)=O

c1(NC(=O)C)c(O)cc(O)cc1

O=C(C)Nc1ccc(O)cc1O

CC(Nc1ccc(O)cc1O)=O

N(C(=O)C)c1c(O)cc(O)cc1

c1c(O)cc(O)c(NC(=O)C)c1

C(C)(=O)Nc1c(O)cc(O)cc1

C(Nc1c(O)cc(O)cc1)(C)=O

c1c(O)c(NC(C)=O)ccc1O

C(Nc1ccc(O)cc1O)(=O)C

c1(NC(C)=O)ccc(O)cc1O

c1cc(NC(C)=O)c(O)cc1O

c1c(NC(=O)C)c(O)cc(O)c1

N(C(=O)C)c1ccc(O)cc1O

Oc1cc(O)ccc1NC(C)=O

CC(Nc1c(O)cc(O)cc1)=O

C(C)(=O)Nc1ccc(O)cc1O

C(=O)(Nc1ccc(O)cc1O)C

c1(O)cc(O)c(NC(=O)C)cc1

CC(=O)Nc1c(O)cc(O)cc1

c1(O)cc(O)ccc1NC(C)=O

c1c(O)ccc(NC(C)=O)c1O

C(C)(Nc1ccc(O)cc1O)=O

c1(NC(=O)C)ccc(O)cc1O

c1c(O)ccc(NC(=O)C)c1O

Oc1ccc(NC(C)=O)c(O)c1

c1(O)ccc(NC(=O)C)c(O)c1



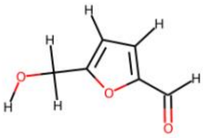
c1cc(C=O)oc1CO



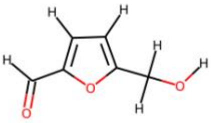
c1(C=O)oc(CO)cc1



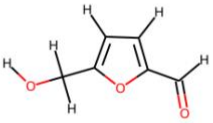
C(c1ccc(CO)o1)=O



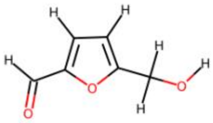
C(O)c1oc(C=O)cc1



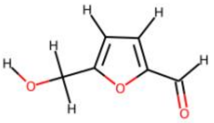
C(c1ccc(C=O)o1)O



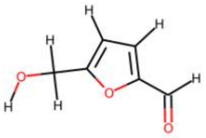
c1(C=O)ccc(CO)o1



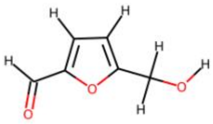
c1(CO)ccc(C=O)o1



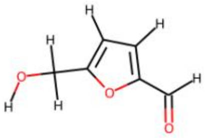
o1c(C=O)ccc1CO



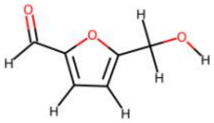
C(c1oc(C=O)cc1)O



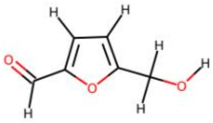
o1c(CO)ccc1C=O



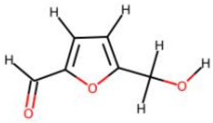
OCc1oc(C=O)cc1



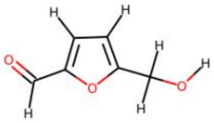
c1c(CO)oc(C=O)c1



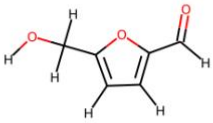
C(=O)c1oc(CO)cc1



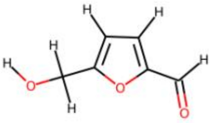
OCc1ccc(C=O)o1



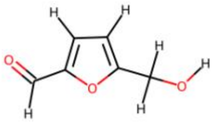
C(c1oc(CO)cc1)=O



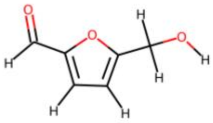
c1c(C=O)oc(CO)c1



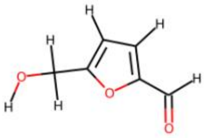
C(=O)c1ccc(CO)o1



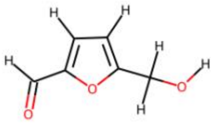
O=Cc1oc(CO)cc1



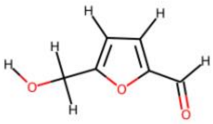
c1cc(CO)oc1C=O



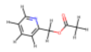
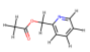
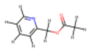
c1(CO)oc(C=O)cc1



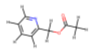
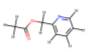
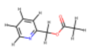
C(O)c1ccc(C=O)o1



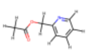
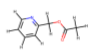
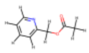
O=Cc1ccc(CO)o1

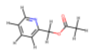
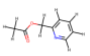
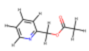
C(C)(OCc1ccccn1)=O c1c(COC(=O)C)nccc1 C(c1ccccn1)OC(C)=O

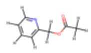
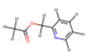
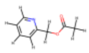
C(OCc1ccccn1)(C)=O c1c(COC(C)=O)nccc1 c1(COC(C)=O)ncccc1

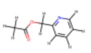
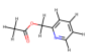
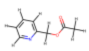
c1ccc(COC(C)=O)nc1 c1cccnc1COC(C)=O O=C(C)OCc1ccccn1

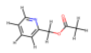
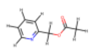
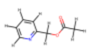
C(=O)(C)OCc1ccccn1 c1nc(COC(=O)C)ccc1 c1(COC(=O)C)ncccc1

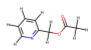
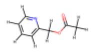
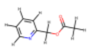
C(OC(=O)C)c1ccccn1 c1cnc(COC(=O)C)cc1 C(OC(C)=O)c1ccccn1

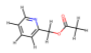
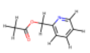
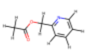
c1cc(COC(C)=O)ncc1 c1nc(COC(C)=O)ccc1 n1ccccc1COC(C)=O

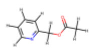
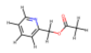
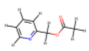
C(C)(=O)OCc1ccccn1 C(OC(C)=O)c1ncccc1 C(OCc1ncccc1)(=O)C

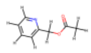
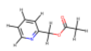
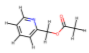
C(=O)(OCc1ncccc1)C O(Cc1ccccn1)C(=O)C C(OC(=O)C)c1ncccc1

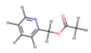
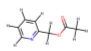
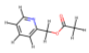
c1ccnc(COC(C)=O)c1 c1ccc(COC(=O)C)nc1 c1cc(COC(=O)C)ncc1

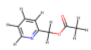
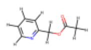
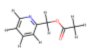
C(c1ncccc1)OC(=O)C O=C(OCc1ccccn1)C O=C(OCc1ncccc1)C

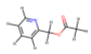
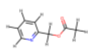
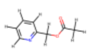
C(=O)(OCc1ccccn1)C C(C)(=O)OCc1ncccc1 C(OCc1ccccn1)(=O)C

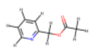
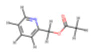
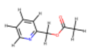
c1(COC(C)=O)ccccn1 c1cccc(COC(=O)C)n1 O(C(=O)C)Cc1ccccn1

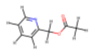
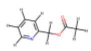
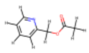
c1cccc(COC(C)=O)n1 CC(=O)OCc1ncccc1 c1cccnc1COC(=O)C

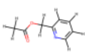
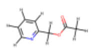
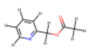
O(C(C)=O)Cc1ccccn1 C(OCc1ncccc1)(C)=O O(C(C)=O)Cc1ncccc1

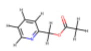
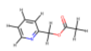
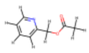
C(c1ncccc1)OC(C)=O CC(=O)OCc1ccccn1 C(=O)(C)OCc1ncccc1

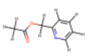
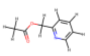
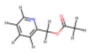
c1(COC(=O)C)ccccn1 n1ccccc1COC(=O)C C(c1ccccn1)OC(=O)C

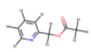
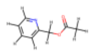
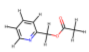
c1cnc(COC(C)=O)cc1 O(Cc1ncccc1)C(=O)C O(Cc1ncccc1)C(C)=O

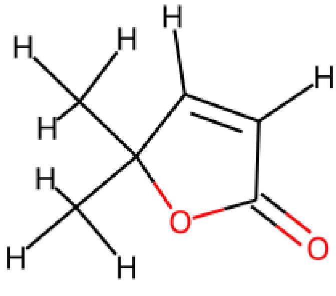
CC(OCc1ncccc1)=O O=C(C)OCc1ncccc1 O(Cc1ccccn1)C(C)=O

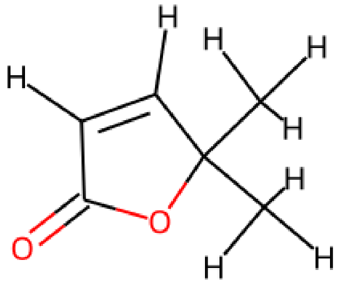
n1c(COC(C)=O)cccc1 n1c(COC(=O)C)cccc1 CC(OCc1ccccn1)=O

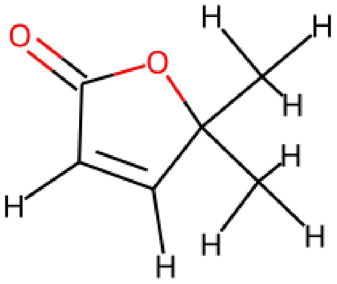
C(C)(OCc1ncccc1)=O c1ccnc(COC(=O)C)c1 O(C(=O)C)Cc1ncccc1



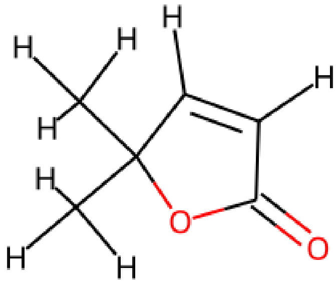
O=C1C=CC(C)(C)O1



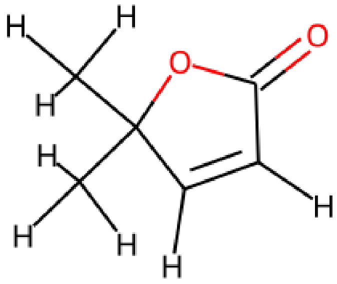
O1C(C)(C)C=CC1=O



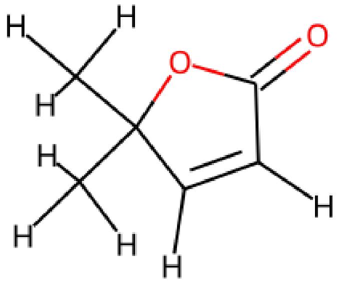
C1=CC(C)(C)OC1=O



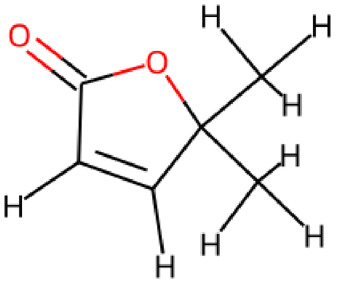
O1C(=O)C=CC1(C)C



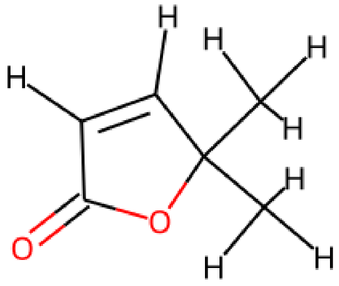
C1(C)(C)C=CC(=O)O1



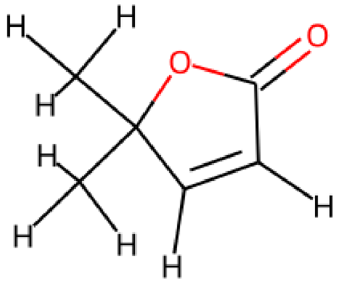
C1=CC(=O)OC1(C)C



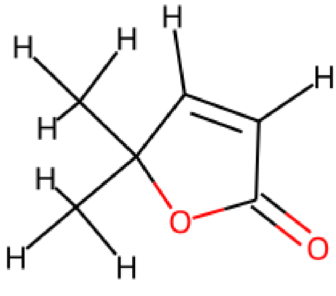
C1(C)(C)OC(=O)C=C1



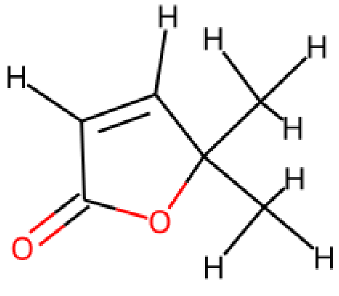
C1(=O)OC(C)(C)C=C1



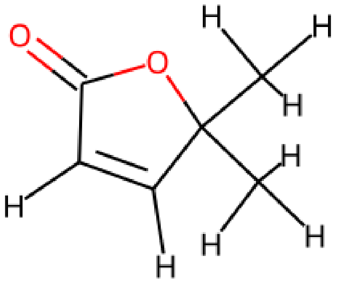
CC1(C)C=CC(=O)O1



C1(=O)C=CC(C)(C)O1



O=C1OC(C)(C)C=C1



CC1(C)OC(=O)C=C1

C(CC)CCCCCCCCCCCCCCCCCCCCC

C(CCCCCCCCCCCCCCCCC)CCCCCC

C(CCCCCCC)CCCCCCCCCCCCCCCC

C(CCCCCCCCCCCCCCCCCCCC)CCC

C(CCCCCCCCCC)CCCCCCCCCCCCC

C(CCCCCCCCCCCCCCCCCCCCCC)C

C(CCCCCCCCCCCCCCCCCCC)CCCC

C(CCCCCCCCCCCCCCCCCC)CCCCC

C(CCCCCCCCCCCC)CCCCCCCCCCC

C(CCCCCCCCCCCCCCCC)CCCCCCC

C(CCCCCCCCCCCCCC)CCCCCCCCC

C(CCCCCCCCCCC)CCCCCCCCCCCC

C(CCCCCCCCC)CCCCCCCCCCCCCC

C(CCCCCCCCCCCCCCCCCCCCC)CC

C(CCCCCCCCCCCCC)CCCCCCCCCC

C(C)CCCCCCCCCCCCCCCCCCCCCC

C(CCCCCC)CCCCCCCCCCCCCCCCC

C(CCCCCCCCCCCCCCC)CCCCCCCC

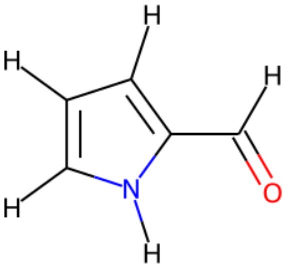
C(CCC)CCCCCCCCCCCCCCCCCCCC

C(CCCCC)CCCCCCCCCCCCCCCCCC

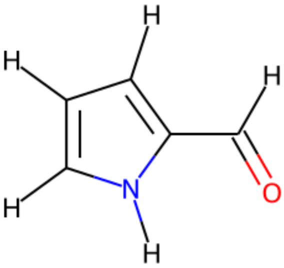
C(CCCC)CCCCCCCCCCCCCCCCCCC

CCCCCCCCCCCCCCCCCCCCCCCC

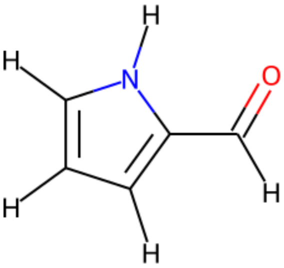
C(CCCCCCCC)CCCCCCCCCCCCCCC



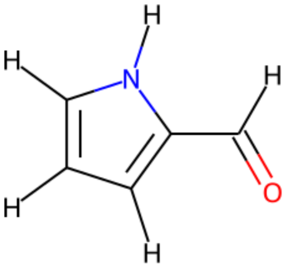
C(=O)c1ccc[nH]1



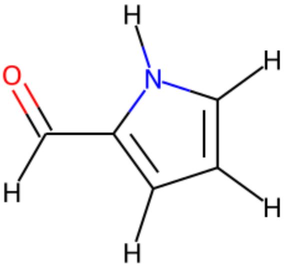
c1[nH]c(C=O)cc1



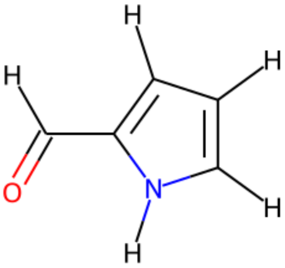
c1cc(C=O)[nH]c1



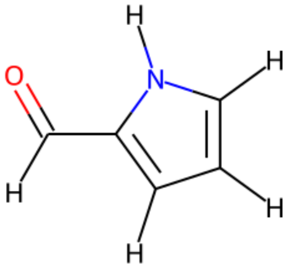
C(c1[nH]ccc1)=O



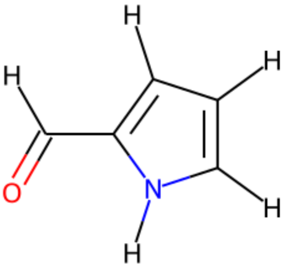
c1c[nH]c(C=O)c1



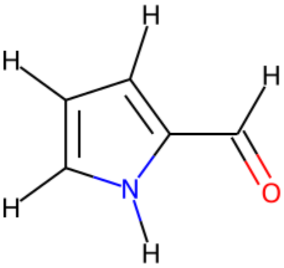
c1ccc(C=O)[nH]1



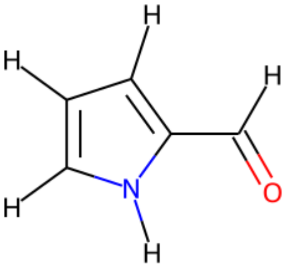
c1cc[nH]c1C=O



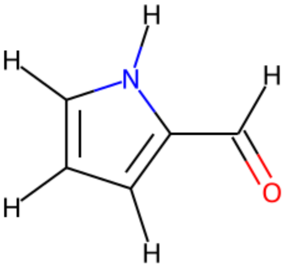
[nH]1cccc1C=O



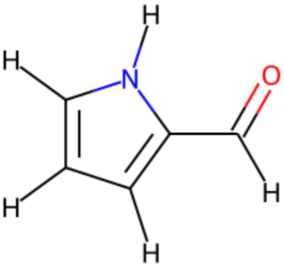
c1(C=O)ccc[nH]1



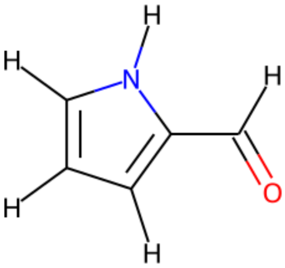
[nH]1c(C=O)ccc1



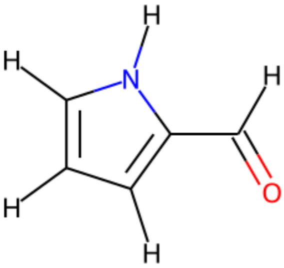
O=Cc1[nH]ccc1



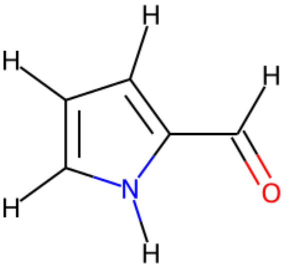
c1c(C=O)[nH]cc1



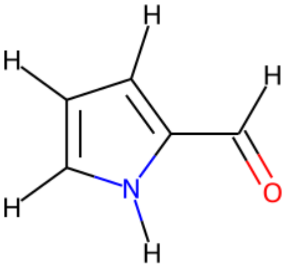
c1(C=O)[nH]ccc1



C(=O)c1[nH]ccc1



O=Cc1ccc[nH]1



C(c1ccc[nH]1)=O

9. References

[1] RDKit, https://www.rdkit.org/, Accessed 19th July 2021

[2] Atomic Simulation Environment, https://wiki.fysik.dtu.dk/ase/ , Accessed 19th July 2021

[3] Github repository, XYZ2mol, https://github.com/jensengroup/xyz2mol, Accessed 9th July 2021

[4] RDKit, Stereoisomers, https://www.rdkit.org/docs/source/rdkit.Chem.EnumerateStereoisomers.html, Accessed 21st July

[5] N. Treitel, R. Shenhar, I. Aprahamian, T. Sheradsky,M. Rabinnovitz, Phys. Chem. Chem. Phys., 2004, 6, 1113-1121

[6] M. Wadrzyk, R. Janus, M. P. Vos, D. W. F. Brilman, *J. Anal. Appl. Pyrolysis,* 2018, **134,** 415-426