

PARSER PLIKÓW PDBML

Implementacja modułu odczytującego plik opisujący strukturę
molekularną w formacie PDBML/XML

PARSER PDBML

*Implementation of the module reading the file describing the
molecular structure in PDBML / XML format*

Sandra Sobierajska

Kierunek: Informatyka

Uniwersytet Jagielloński

Spis treści

1. Informacje ogólne	3
2. Format PDBML/XML	4
3. Opis projektu.....	5
4. Opis klas.....	6
a) Klasa Atom	6
b) Klasa Logger.....	6
c) Klasa Parser	6
d) Klasa Residue.....	7
e) Klasa Vector.....	8
f) Klasa Main	8
5. Wymagania	8
6. Obsługa błędów oraz obsługa wejścia	9
7. Testy	9
8. Wyniki	12
9. Podsumowanie	14
10. Bibliografia	14

1. Informacje ogólne

Przedstawiona praca ma na celu stworzenie możliwości wykorzystania plików w formacie PDBML/XML, w którym są zapisane dane strukturalne cząsteczek biologicznych. Na początku kilka słów wstępu.

XML – sposób zapisu danych, język, który przedstawia je w strukturalizowany sposób

Parser – służy do analizy składniowej danych wejściowych

W związku z tym, że dane struktur biologicznych są zapisane w różnych formatach, naukowcy mają większe pole manewru w przeprowadzaniu badań. Najbardziej znanym jest parser plików PDB (Protein Data Bank), który jest wykorzystywany najczęściej i jest najlepiej rozwinięty jak do tej pory. Biopython wykorzystuje zatem o białkach i kwasach nukleinowych bezpośrednio z bazy PDB, a użytkownik może korzystać z parsera w ustalony sposób.

Ponadto dokonuje się parsowania plików w formacie mmCIF, jednak powoli ten format jest wycofywany. Ma on jednak przewagę nad formatem PDB, ponieważ nie nakłada żadnych limitów na liczbę atomów, reszt i łańcuchów w pojedynczym pliku. Dodatkowo format ten łatwiej można parsować, ponieważ jest on napisany zgodnie z gramatyką bekontekstową, a wyniki są otrzymywane w postaci tabel lub słów-kluczy.

mmCIF (macromolecular Crystallographic Information File) – format plików, który powstał pod patronatem International Union of Crystallography, który jest używany do opisywania małych struktur cząsteczek i powiązanych z nimi eksperymentów dyfrakcyjnych
--

Do tej pory nie zajmowano się parsowaniem plików w formacie PDBML, bądź dokonywano tego tylko dla swoich potrzeb, jednak nie jest on ogólnodostępny dla zwykłych użytkowników. Dlatego w tej pracy przyjrzymy się zapisowi danych w tym formacie oraz będziemy korzystać z metod, które umożliwią nam uzyskanie oczekiwanych przez nas danych.

Parser ten został stworzony na podstawie analizatora składniowego, o którym była wcześniej mowa, a mianowicie z Parsera PDB, dostępnego w tutorialu biopythona.

2. Format PDBML/XML

Przyjrzyjmy się niektórym składowym formatu PDBML.

```
<PDBx:atom_site id="1">
  <PDBx:B_iso_or_equiv>99.85</PDBx:B_iso_or_equiv>
  <PDBx:Cartn_x>50.193</PDBx:Cartn_x>
  <PDBx:Cartn_y>51.190</PDBx:Cartn_y>
  <PDBx:Cartn_z>50.534</PDBx:Cartn_z>
  <PDBx:auth_asym_id>A</PDBx:auth_asym_id>
  <PDBx:auth_atom_id>OP3</PDBx:auth_atom_id>
  <PDBx:auth_comp_id>G</PDBx:auth_comp_id>
  <PDBx:auth_seq_id>1</PDBx:auth_seq_id>
  <PDBx:group_PDB>ATOM</PDBx:group_PDB> grupa - sekcja - section
  <PDBx:label_alt_id xsi:nil="true" /> alt loc
  <PDBx:label_asym_id>A</PDBx:label_asym_id>
  <PDBx:label_atom_id>OP3</PDBx:label_atom_id>
  <PDBx:label_comp_id>G</PDBx:label_comp_id>
  <PDBx:label_seq_id>1</PDBx:label_seq_id>
  <PDBx:occupancy>1.00</PDBx:occupancy> occupancy
  <PDBx:pdxb_PDB_model_num>1</PDBx:pdxb_PDB_model_num>
  <PDBx:type_symbol>O</PDBx:type_symbol>
</PDBx:atom_site>
```

<code>atom_site id</code>	- pozycja atomu
<code>B_iso_or_equiv</code>	- wskaźnik T_Factor, używany w krystalografii do identyfikacji wielkości i kierunków wibracji termicznych atomów w strukturach krystalicznych
<code>Cartn_x, Cartn_y, Cartn_z</code>	- zawierają dane o współrzędnych
<code>auth_asym_id</code>	- nazwa łańcucha
<code>auth_atom_id</code>	- nazwa atomu
<code>auth_comp_id</code>	- nazwa reszty (residue)
<code>auth_seq_id</code>	- numer reszty (residue)

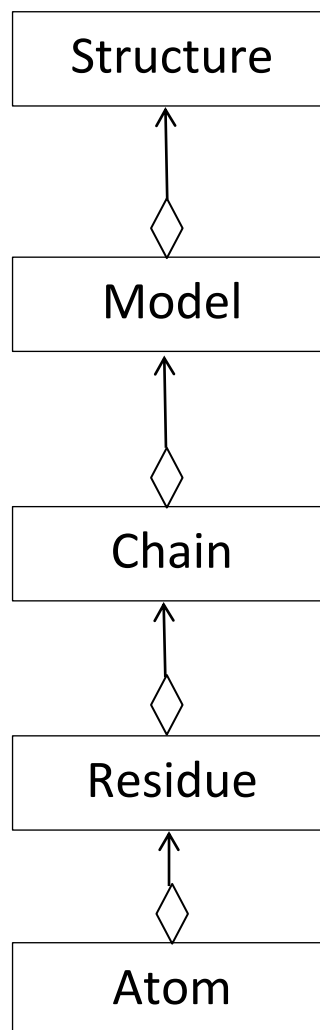
<code>group_PDB</code>	- sekcja w pliku PDB (np. atom)
<code>label_alt_id</code>	- składowa identyfikatora pozycji atomu
<code>occupancy</code>	- współczynnik obsadzenia
<code>pdxb_PDB_model_num</code>	- numer modelu w pdbx
<code>type_symbol</code>	- oznaczenie typu

Przedstawiony fragment jest niezbędny do zrozumienia programu, ponieważ większość metod będzie korzystała właśnie z tych danych. Część danych jest taka sama, jednakże inaczej opisana. Jeśli spojrzymy na rekord o nazwie np. *auth_comp_id* i *label_comp_id* to informacje w tym przypadku są takie same, w niektórych inne. Jest to związane z tym, że *label* są oznaczeniami wprowadzonymi przez autora i zazwyczaj pokrywają się one z ogólnoprzyjętymi.

3. Opis projektu

Architektura struktury obiektu:

- struktura składa się z modelu
- model składa się z łańcucha
- łańcuch składa się z reszt
- reszty składają się z atomów



Projekt składa się z jednej głównej klasy **Main** oraz pięciu dodatkowych klas: **Atom**, **Parser**, **Residue**, **Vector**, **Logger**. Aby móc korzystać w pełni z projektu należy mieć możliwość zaimportowania tych klas. Ponadto każda klasa dodatkowa tworzy nam nasz oczekiwany obiekt jakim jest struktura cząsteczki.

4. Opis klas

a) Klasa Atom

Klasa Atom zawiera wiele metod umożliwiających uzyskanie informacji z naszego pliku w formacie XML.

- **Get_name()** nazwa atomu
- **Get_id()** numer atomu
- **Get_coord()** współrzędne atomu
- **Get_vector()** współrzędne atomu jako obiekt wektora
- **Get_bfactor()** izotropowy parametr przemieszczeń atomowych
- **Get_occupancy()** współczynnik obsadzenia
- **Get_altloc()** składnik identyfikatora miejsca
- **Get_fullname()** nazwa atomu (np. 'CA')
- **Get_comp_id()** nazwa reszty

Na końcu klasy znajduje się funkcja umożliwiająca wypisywanie wszystkich informacji, które możemy uzyskać za pomocą wyżej wymienionych metod.

b) Klasa Logger

Klasa, która została stworzona po to, aby użytkownik mógł uzyskane wyniki zapisać w pliku tekstowym na swoim dysku.

c) Klasa Parser

Klasa, która dokonuje parsowania pliku w formacie PDBML/XML. Uzyskuje ona informacje na temat naszej struktury, na temat atomów, reszt oraz nazwy struktury. Parser ten dokonuje również obliczeń statystycznych, czyli długości struktury, pod względem liczby atomów oraz liczby reszt.

Klasa Parser działa dla kilku różnych wersji pdbx. Jest to obowiązkowe, ponieważ różne cząsteczki mają dane zapisane w odmiennych wersjach, przez co program byłby niepoprawny dla niektórych plików, i nie dawał by żadnych wyników. Gdyby użytkownik chciał użyć pliku, którego wersja nie jest brana pod uwagę (np. będzie korzystał z projektu w przyszłości, kiedy będą już nowsze wersje) wystarczy, że doda odpowiednią linijkę z odpowiednią wersją do już istniejących:

```
self.remove_namespace(self.results, u'http://pdml.pdb.org/schema/pdbx-v40.xsd')  
  
self.remove_namespace(self.results, u'http://pdml.pdb.org/schema/pdbx-v42.xsd')  
  
self.remove_namespace(self.results, u'http://pdml.pdb.org/schema/pdbx-v50.xsd')
```

Zabieg, który ma na celu uzyskanie informacji na temat atomu, bierze pod uwagę tylko *atom_siteCategory*. Natomiast jeśli chodzi o reszty, napotkano tutaj niewielki problem, ponieważ w różnych cząsteczkach są one zapisywane w inny sposób, w szczególności jeśli porównamy plik zawierający dane o RNA z plikiem zawierającym dane białka. Aby zapobiec nieuzyskaniu danych wprowadzono pomocniczą funkcję *residues_attributes*, która bierze pod uwagę wszystkie możliwe przypisy dotyczące reszt w plikach.

W związku z tym, że w plikach w formacie XML nie mamy do czynienia z nagłówkiem, parser odczytuje jedynie nazwę struktury z *datablockName*.

Na końcu znajduje się funkcja wypisująca dane, które są wyświetlane:

```
def print_file_info(self):  
    print("Structure: " + self.get_name())  
    print("Atoms count: " + self.atoms_length())  
    print("Residues count: " + self.residues_length())
```

d) Klasa Residue

Klasa ta, została stworzona na podobieństwo klasy Atom. Dokonuje ona analizy składniowej cząsteczki pod względem zawartych w nich reszt.

Możemy tam znaleźć metody:

- **Get_resname()** wypisuje nazwę reszty
- **Get_details()** wyświetla pełną nazwę reszty, jeśli znajduje się ona w pliku
- **Get_segid()** wypisuje numer reszty
- **Has_id()** umożliwia sprawdzenie czy reszta zawiera konkretny atom

Jak w poprzednich klasach, na końcu wyświetlenie wyników uzyskanych za pomocą tych metod.

e) Klasa Vector

Plik źródłowy do tej klasy został zaczerpnięty z ogólnodostępnego pliku <https://gist.github.com/mcleonard/5351452>. Zgodnie z licencją, która została również dołączona w kodzie programu, można korzystać z tego pliku dla swoich celów. W tym projekcie nie skupiono się na samodzielnym stworzeniu tej klasy, ponieważ nie różniłaby się ona od użytego kodu oraz służy ona jedynie do uzyskania jednej wartości, co nie jest sztandarowym celem tego projektu.

f) Klasa Main

Główna klasa, którą uruchamiamy, kiedy chcemy zobaczyć działanie całego projektu.

Dzięki niej użytkownik może używać napisany projekt. Wystarczy, że poda on ścieżkę do pliku w formacie XML, który zawiera informacje o strukturze cząsteczki.

Warto również zwrócić uwagę, iż są wypisane tylko przykładowe rekordy, czyli np. 5 pierwszych atomów i reszt. Oczywiście użytkownik może zmienić tę liczbę na dowolną przez siebie wybraną. Taki sposób wyświetlania ma na celu ułatwienie sprawdzenia przejrzystości i poprawności danych.

5. Wymagania

Projekt jest kompatybilny zarówno z pythonem 2.7, jak i pythonem 3.* (*dotyczy różnych wersji, cyfry 1-6), co umożliwia dopasowanie programu do ustawień użytkownika.

Jednak, aby móc korzystać z projektu w obydwóch wersjach, użytkownik jest zobligowany do zainstalowania dodatkowej biblioteki. Może to zrobić wpisując w konsoli **pip install -r requirements.txt** bądź jeśli korzysta z dostępnego środowiska, jakim jest np.

Pycharm oraz wersji python 3.* może skorzystać z dostępnej usługi **Code compatibility inspection** i wybrać wersję 2.7.

6. Obsługa błędów oraz obsługa wejścia

Użytkownik, który chce skorzystać z projektu już na samym początku zostanie poproszony o wprowadzenia ścieżki folderu, w którym znajduje się plik w formacie XML.

Jeśli chodzi o możliwości znalezienia plików w tym formacie opisujących struktury cząsteczek można je pobrać w bardzo łatwy sposób z Protein Data Bank (PDB). Wystarczy wpisać interesującą nas cząsteczkę i pobrać dane w formacie PDBML/XML (gz). Gz oznacza, że nasze dane będą w formie spakowanej, tak więc należy je rozpakować w dowolnym folderze i zapamiętać ścieżkę do tego pliku, aby móc ich użyć w programie.

Gdyby zdarzyło się tak, iż użytkownik wprowadził złą ścieżkę na ekranie zostanie wyświetlona informacja: „**Path to the file is incorrect or the file does not exist.**”, co oznacza, że w danym folderze nie znajduje się plik wejściowy o danej nazwie.

Jeśli natomiast zostanie wprowadzona poprawna ścieżka oraz nazwa pliku, natomiast nie będzie ten plik w formacie XML, użytkownik uzyska informację: "**Invalid XML format file**".

7. Testy

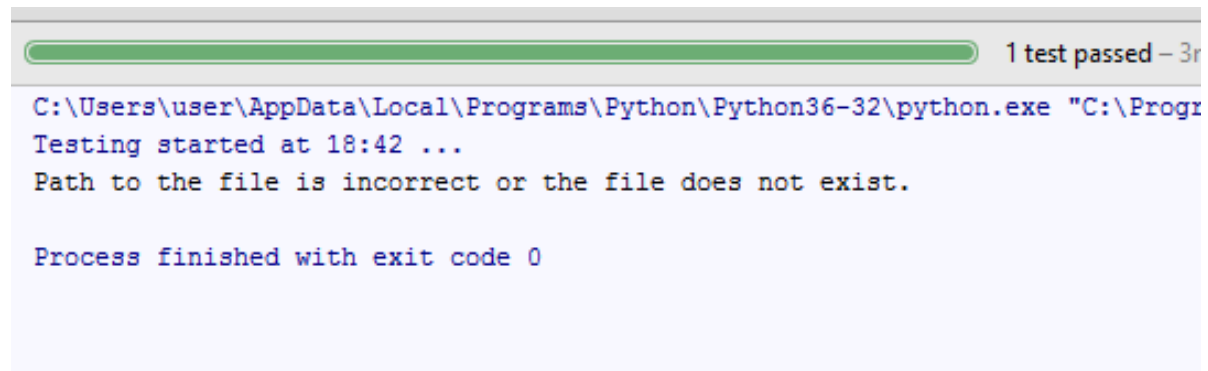
Zostało stworzonych pięć testów, które mają na celu weryfikację poprawności działania programu. Sprawdzają one funkcjonalność na podstawie utworzonych przykładowych plików w formacie XML o nazwach *ok* i *corrupted*, odpowiednio plik poprawnie stworzony oraz plik błędny.

Test 1: Sprawdza czy dany plik istnieje.

```
def test_file_does_not_exist(self):
    no_file = PDBMLParser('xml/nofile.xml')

    with self.assertRaises(SystemExit):
        no_file.load()
```

W związku z tym, iż dany plik nie istnieje otrzymamy informację:



```
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python.exe "C:\Program
Testing started at 18:42 ...
Path to the file is incorrect or the file does not exist.

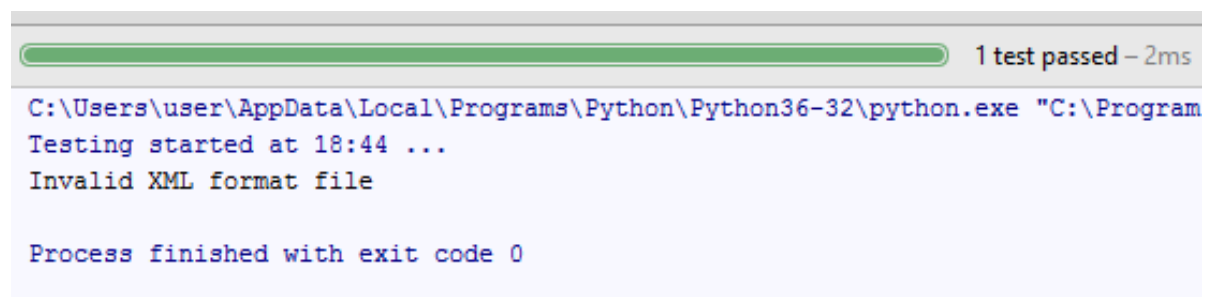
Process finished with exit code 0
```

Test 2: Sprawdza czy format pliku jest poprawny.

```
def test_file_is_corrupted(self):
    corrupted = PDBMLParser('test/corrupted.xml')

    with self.assertRaises(SystemExit):
        corrupted.load()
```

Plik *corrupted.xml* zawiera niezgodny zapis z formatem XML, dlatego otrzymujemy taką odpowiedź:



```
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python.exe "C:\Program
Testing started at 18:44 ...
Invalid XML format file

Process finished with exit code 0
```

Test 3: Sprawdza poprawność liczenia długości atomów i reszt.

```
def test_atoms_and_residues_count(self):
    self.assertEqual(len(self.parser.residues), 1)
    self.assertEqual(len(self.parser.atoms), 1)
```

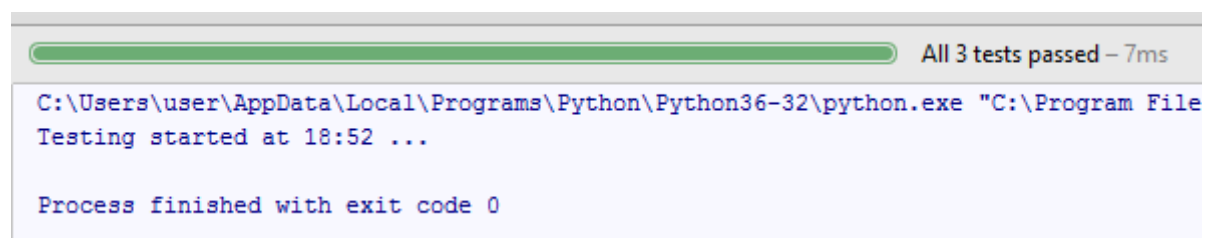
Test 4 : Sprawdza poprawność uzyskiwanych danych z parsowania reszt, sprawdza informacje o nazwie reszty oraz tego czy zawiera dany atom.

```
def test_pass_residue(self):  
    self.assertEqual(self.parser.residues[0].get_resname(), '2MG')  
    self.assertEqual(self.parser.residues[0].has_atom('OP3'), False)
```

Test 5 : Sprawdza poprawność uzyskiwanych danych pod względem danych o atomie, nazwie atomu oraz parametrze przemieszczeń atomowych.

```
def test_pass_atom(self):  
    self.assertEqual(self.parser.atoms[0].get_name(), 'OP3')  
    self.assertEqual(self.parser.atoms[0].get_bfactor(), 99.85)
```

Z funkcji 3-5, ponieważ plik zawiera poprawne dane, uzyskujemy informację o zakończeniu pomyślnie testów:



The screenshot shows a terminal window with a green progress bar at the top indicating 'All 3 tests passed - 7ms'. Below the bar, the command prompt shows the execution of a Python script: 'C:\Users\user\AppData\Local\Programs\Python\Python36-32\python.exe "C:\Program File'. The output of the script is 'Testing started at 18:52 ...'. At the bottom, it states 'Process finished with exit code 0'.

8. Wyniki

Poniżej zostały przedstawione przykładowe wyniki:

- dla cząsteczki RNA

```
main
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python.exe
Path to the file (e.g. xml/1ehz.xml): xml/1evv.xml

Protein: 1EVV
Atoms count: 1896
Residues count: 58

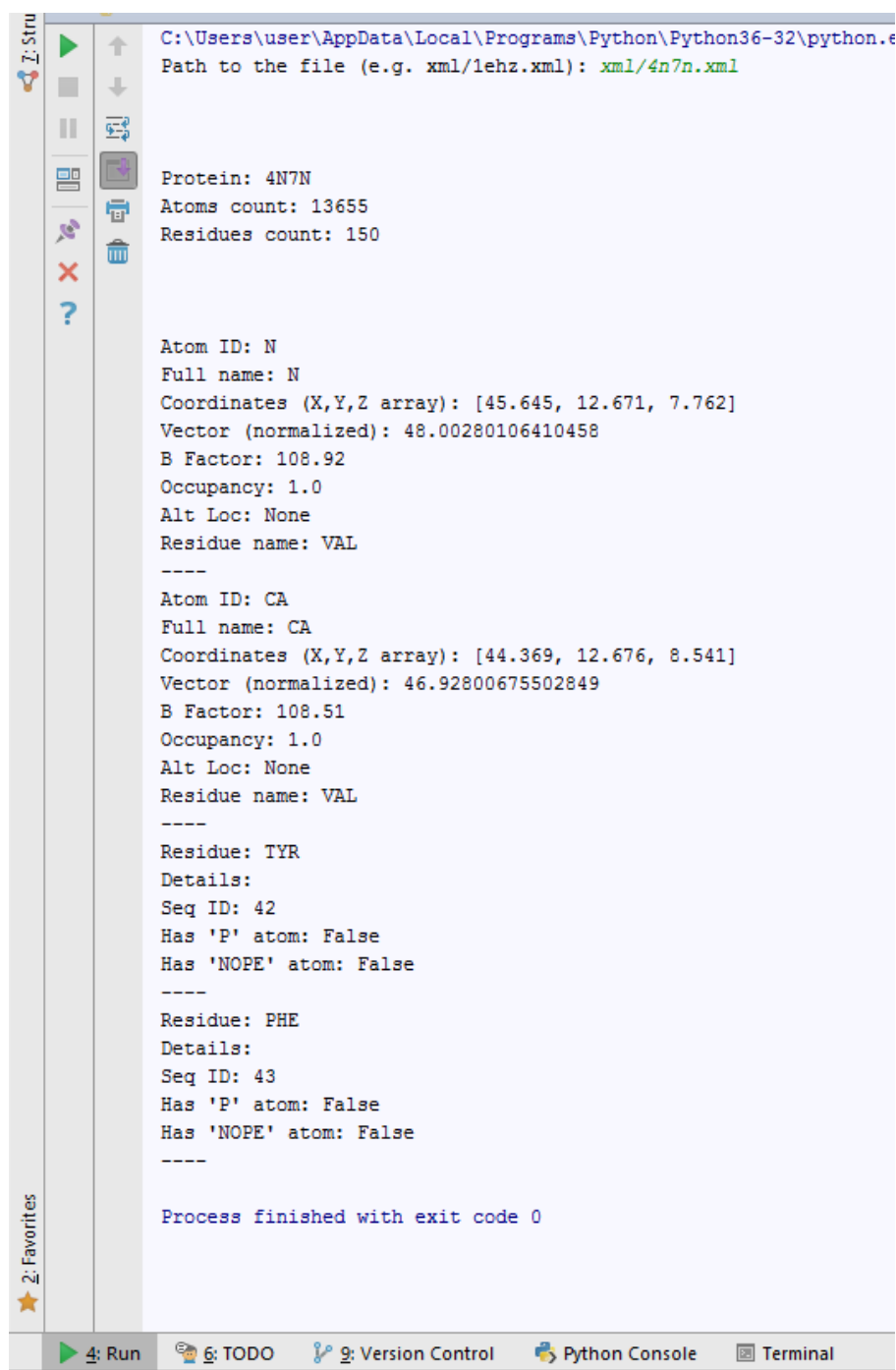
Atom ID: OP3
Full name: OP3
Coordinates (X,Y,Z array): [21.435, 6.0, 49.476]
Vector (normalized): 54.25250041242339
B Factor: 120.6
Occupancy: 1.0
Alt Loc: None
Residue name: G
----
Atom ID: P
Full name: P
Coordinates (X,Y,Z array): [22.4, 4.865, 49.778]
Vector (normalized): 54.8021670100736
B Factor: 119.67
Occupancy: 1.0
Alt Loc: None
Residue name: G
----
Residue: 2MG
Details: 2N-METHYLGUANOSINE-5'-MONOPHOSPHATE
Seq ID: 10
Has 'P' atom: True
Has 'NOPE' atom: False
----
Residue: H2U
Details: 5,6-DIHYDROURIDINE-5'-MONOPHOSPHATE
Seq ID: 16
Has 'P' atom: True
Has 'NOPE' atom: False
----

Process finished with exit code 0
```

6: TODO 9: Version Control Python Console Terminal

ised: 3 passed (today 18:52)

- dla cząsteczki białka



The screenshot shows a Python console window with a toolbar on the left containing icons for running, undo, redo, and other standard editing functions. The main area displays the output of a script that has loaded protein data for the PDB entry 4N7N. The output includes general statistics (Protein: 4N7N, Atoms count: 13655, Residues count: 150) and detailed information for three specific atoms: N, CA, and TYR. Each atom's details include its ID, full name, 3D coordinates, a normalized vector, B-factor, occupancy, and alternative location. The TYR and PHE residues are also shown with their sequence IDs and whether they have 'P' or 'NOPE' atoms. The console ends with a message indicating the process finished successfully.

```
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python.exe
Path to the file (e.g. xml/1ehz.xml): xml/4n7n.xml

Protein: 4N7N
Atoms count: 13655
Residues count: 150

Atom ID: N
Full name: N
Coordinates (X,Y,Z array): [45.645, 12.671, 7.762]
Vector (normalized): 48.00280106410458
B Factor: 108.92
Occupancy: 1.0
Alt Loc: None
Residue name: VAL
----
Atom ID: CA
Full name: CA
Coordinates (X,Y,Z array): [44.369, 12.676, 8.541]
Vector (normalized): 46.92800675502849
B Factor: 108.51
Occupancy: 1.0
Alt Loc: None
Residue name: VAL
----
Residue: TYR
Details:
Seq ID: 42
Has 'P' atom: False
Has 'NOPE' atom: False
----
Residue: PHE
Details:
Seq ID: 43
Has 'P' atom: False
Has 'NOPE' atom: False
----

Process finished with exit code 0
```

2: Favorites

4: Run 6: TODO 9: Version Control Python Console Terminal

9. Podsumowanie

Projekt został stworzony z myślą o naukowcach, którzy zajmują się analizą cząsteczek biologicznych i korzystają z danych zapisanych w formacie PDBML/XML. Przygotowany parser ma na celu umożliwić pracę na plikach właśnie w tym formacie.

Program nie jest skomplikowany dzięki czemu osoby, które nie znają się zbyt dobrze na informatyce będą mogły oczywiście skorzystać z funkcji oraz metod jakie oferuje ten program. Ponadto kod można modyfikować i dodawać nowe funkcjonalności. W tym projekcie natomiast zostały utworzone podstawowe klasy, które są najczęściej używane.

10. Bibliografia

- *Biopython Tutorial and Cookbook*, Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński; <http://biopython.org/DIST/docs/tutorial/Tutorial.html>
- *PDB to PDBx/mmCIF Data Item Correspondences*, dokumentacja mmCIF, http://mmcif.wwpdb.org/docs/pdb_to_pdbx_correspondences.html
- *Baza struktur PDB* – Protein Data Bank - <http://www.rcsb.org/pdb/home/home.do>