Biopython Comprehensive Cheat Sheet

Umar-1623

Sequence Manipulation

```python
from Bio.Seq import Seq
from Bio.SeqUtils import seq3, seq1

# Creating a Seq object
my_seq = Seq("ATCG")

# Transcribing and translating
transcribed_seq = my_seq.transcribe()
translated_seq = my_seq.translate()

# Getting three-letter and one-letter codes
three_letter_code = seq3(translated_seq)
one_letter_code = seq1(translated_seq)
```

Listing 1: Working with Seq Objects

```python
# Reverse complement
reverse_complement = my_seq.reverse_complement()

# GC content
gc_content = SeqUtils.GC(my_seq)

# Codon usage
codon_usage = my_seq.count_codons()
```

Listing 2: Sequence Manipulation

Reading and Writing Sequence Files

```python
from Bio import SeqIO

# Reading a FASTA file
record = SeqIO.read("sequence.fasta", "fasta")

# Iterating over records in a file
for record in SeqIO.parse("input.fasta", "fasta"):
    print(record.id, len(record))
```

Listing 3: Reading Sequence Data

```python
# Writing a GenBank file
SeqIO.write(record, "output.gb", "genbank")
```

Listing 4: Writing Sequence Files

Multiple Sequence Alignment

```python
from Bio.Align.Applications import ClustalOmegaCommandline
from Bio import AlignIO

# Perform multiple sequence alignment
clustalomega_cline = ClustalOmegaCommandline("clustalo", infile="input.fasta",
    outfile="output.aln", verbose=True)
clustalomega_cline()
alignment = AlignIO.read("output.aln", "clustal")
```

Listing 5: Multiple Sequence Alignment

Phylogenetics

```python
from Bio import Phylo

# Parse a Newick format tree
tree = Phylo.read("tree.nwk", "newick")

# Draw the tree
Phylo.draw(tree)
```

Listing 6: Phylogenetic Tree Construction

```python
from Bio.Phylo.TreeConstruction import DistanceCalculator
from Bio.Phylo.TreeConstruction import DistanceTreeConstructor

# Calculate distances
calculator = DistanceCalculator('identity')
dm = calculator.get_distance(alignment)

# Build a tree
constructor = DistanceTreeConstructor(calculator)
tree = constructor.build_tree(alignment)
```

Listing 7: Calculating Distances

BLAST Searches

```python
from Bio.Blast import NCBIWWW, NCBIXML

# Performing a BLAST search
result_handle = NCBIWWW.qblast("blastn", "nt", my_sequence)

# Parsing BLAST results
blast_record = NCBIXML.read(result_handle)
```

Listing 8: Performing a BLAST Search

Accessing Online Databases

```python
from Bio import Entrez

# Setting up Entrez
Entrez.email = "your@email.com"

# Fetching a record from GenBank
handle = Entrez.efetch(db="nucleotide", id="NM_001301717", rettype="gb",
    retmode="text")
record = SeqIO.read(handle, "genbank")
```

Listing 9: Accessing Online Databases

Biological Databases

```
from Bio import ExPASy
from Bio import SwissProt

# Accessing protein information from UniProt
handle = ExPASy.get_sprot_raw('Q5SLP9')
record = SwissProt.read(handle)
print(record.description)
```

Listing 10: Fetching Protein Information from UniProt

Pharmacophore Searching

```
from Bio.PDB import ChemicalFeatures
from Bio.PDB import PDBParser
from Bio.PDB import Selection

# Parse a PDB file
parser = PDBParser()
structure = parser.get_structure("protein", "protein.pdb")

# Define pharmacophore features
feat_def = ChemicalFeatures.defaultFeatureDefinitions

# Search for pharmacophores
cf = ChemicalFeatures.GetFeaturesFromDistanceEvaluator(feat_def, max_dist=4.5)
pharmacophores = cf.get_features_for_structure(structure)

# Select atoms within a pharmacophore
selected_atoms = Selection.unfold_entities(structure, 'A')
for feat in pharmacophores:
    atoms_in_feat = Selection.unfold_entities(feat, 'A')
    selected_atoms = [a for a in selected_atoms if a not in atoms_in_feat]
```

Listing 11: Pharmacophore Searching

Working with Population Genetics

```
from Bio.PopGen import HardyWeinberg

# Calculate Hardy-Weinberg equilibrium
observed = [60, 20, 20]
expected = HardyWeinberg.expected_freq(observed)

# Calculate chi-square statistic
chi_square = HardyWeinberg.chisquare(observed, expected)
```

Listing 12: Population Genetics

BioSQL Database Integration

```
from BioSQL import BioSeqDatabase
from Bio import SeqIO

# Connect to a BioSQL database
server = BioSeqDatabase.open_database(driver="MySQLdb", user="root", passwd="
    password", host="localhost", db="bioseqdb")
```

```

# Create a new namespace
namespace = server.new_namespace("example")

# Load sequences into the database
for record in SeqIO.parse("sequences.fasta", "fasta"):
    namespace.load(SeqIO.to_dict(record))
```

Listing 13: BioSQL Database Integration

Phylogenetic Tree Visualization

```
from Bio import Phylo
import matplotlib.pyplot as plt

# Parse a Newick format tree
tree = Phylo.read("tree.nwk", "newick")

# Visualize the tree
Phylo.draw(tree)
plt.show()
```

Listing 14: Phylogenetic Tree Visualization