# Seminar 4: Introduction to biopython

Biophysics

Course 2022-2023

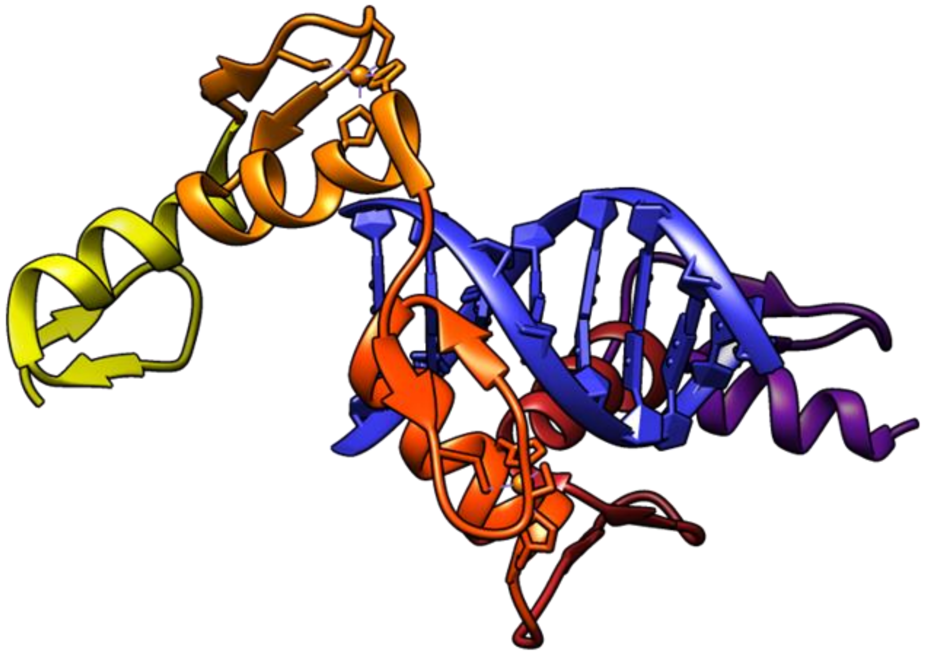
Alberto Meseguer
Irene Acero

## Biopython

**Biopython is a python package with tools for molecular biology analysis**

# Biopython: the PDB module

**We will focus on using the PDB module, which focuses on the manipulation of 3D structures of biomolecules**



# Object oriented programming

**Object oriented programming is based on setting very complex variables named objects**

As programming got more complex, the need for more complex ways to store data appeared. Objects allow you to:

**Store diverse and complex data**

↓

**Attributes**

**Define functions within the object**

↓

**Methods**

## Object oriented programming

Take as an example an object to describe a knight in a game of chess

**Attributes:**

- Position (X-axis)
- Position (Y-axis)
- Color
- Alive

**Methods:**

- Move()

Each knight has the same attributes but with different values assigned, as well as the same methods

## The structure object

**The most important object from the biopython.PDB module is the structure object. It represents the 3D structure of a molecule.**

The structure object contains objects within itself, somehow similarly as a matrioshka contains smaller versions of herself inside

## The structure object

This is how a structure object is organized:

Structure

Model

Chain

Residue

Atom

## The structure object

**The structure object is the biggest in the scale.**
**It contains one or more models within itself.**

To create a structure object, you have to parse a PDB file:

```
>>> from Bio.PDB import *
>>> parser = PDBParser()
>>> structure = parser.get_structure("2dkt",
"2dkt.pdb")
```

## The structure object

**The structure object is the biggest in the scale.**
**It contains one or more models within itself.**

You can access the models by indexing the structure object, or iterate them by using the get_models() function

```
>>> model = structure[0]
```

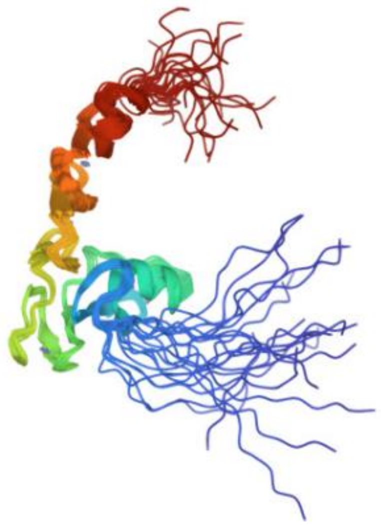
```
>>> for model in structure.get_models()
```

## The model object

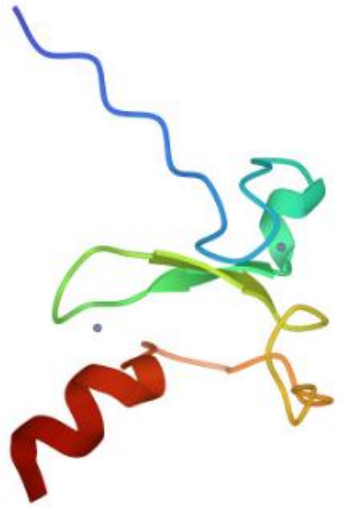
**The model object is the second in the scale.**
**It contains one or more chains within itself.**

NMR experiments can produce several models for the same structure, while X-ray only produces one model

**NMR**



**X-ray**



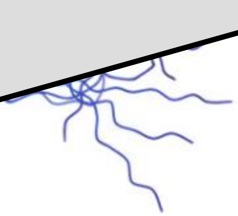
## The model object

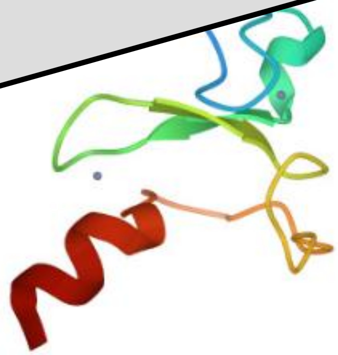
**The model object is the second in the scale.**
**It contains one or more chains within itself.**

NMR experiments can produce several models for th
same structure, while X-ray only produc

**NMR**

**The model object will only make sense for NMR structures, for X-ray structures there will be only one model**





# The model object

**The model object is the second in the scale.**
**It contains one or more chains within itself.**

You can access the chains by indexing the model object with the chain id, or iterate them by using the get_chains() function

```
>>> chain = model[“A”]
```

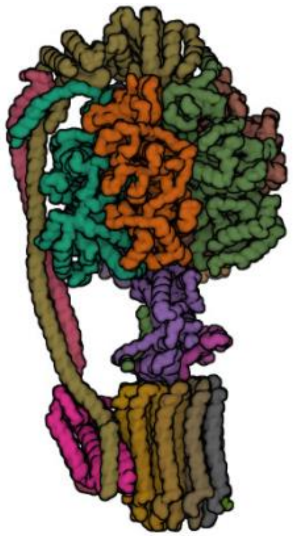
```
>>> for chain in model.get_chains()
```

## The chain object

**The chain object is the third in the scale.**
**It contains one or more residues within itself.**

Structures usually contain several chains, which can correspond with:

**Protein subunits for complexes**

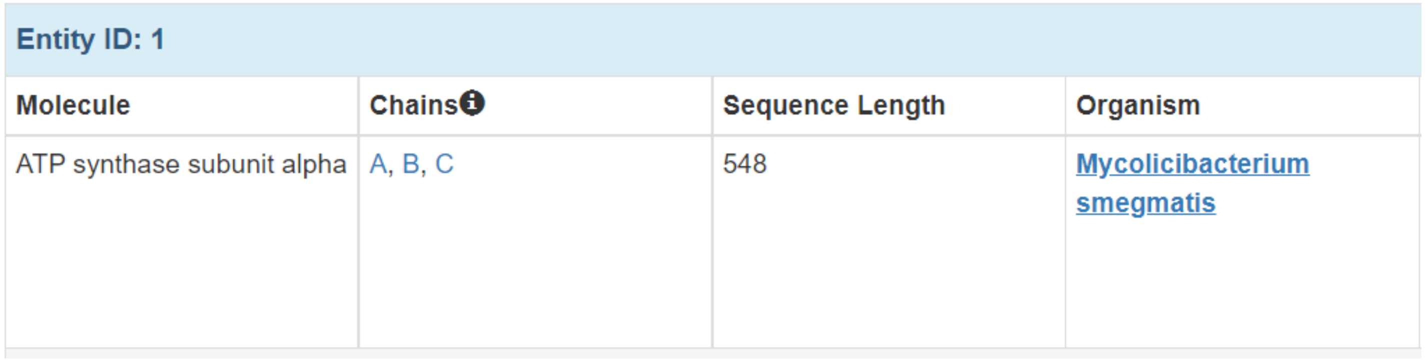


**DNA strands in a DNA molecule**

# The chain object

**The chain object is the third in the scale.**
**It contains one or more residues within itself.**

Chains are identified with an ID which is a letter. You can check what protein subunit corresponds with each chain ID in the PDB webpage:



Entity ID: 1

| Molecule | Chains | Sequence Length | Organism |
|---|---|---|---|
| ATP synthase subunit alpha | A, B, C | 548 | Mycolicibacterium smegmatis |

## The chain object

**The chain object is the third in the scale.**
**It contains one or more residues within itself.**

You can access the residues by indexing the chain object with the residue ID, or iterate them by using the get_residues() function

```
>>> residue = chain[94]
```

```
>>> for residues in chain.get_residues()
```

## The chain object

**The chain object is the third in the scale.**
**It contains one or more residues within itself.**

You can access the residues by indexing the chain object with the residue ID, or iterate them by using the get_residues() function

```
>>> residue = chain[94]
```

When doing this indexing, the index must be a residue number present in the structure. Remember that most structures are not complete, therefore they will miss some residues.

```
>>> for residues in chain.get_residues()
```

## The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

The residue object represents one monomer in the different macromolecules we can manipulate.

**Amino acids for protein structures**

**Nucleotides for DNA or RNA structures**

## The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

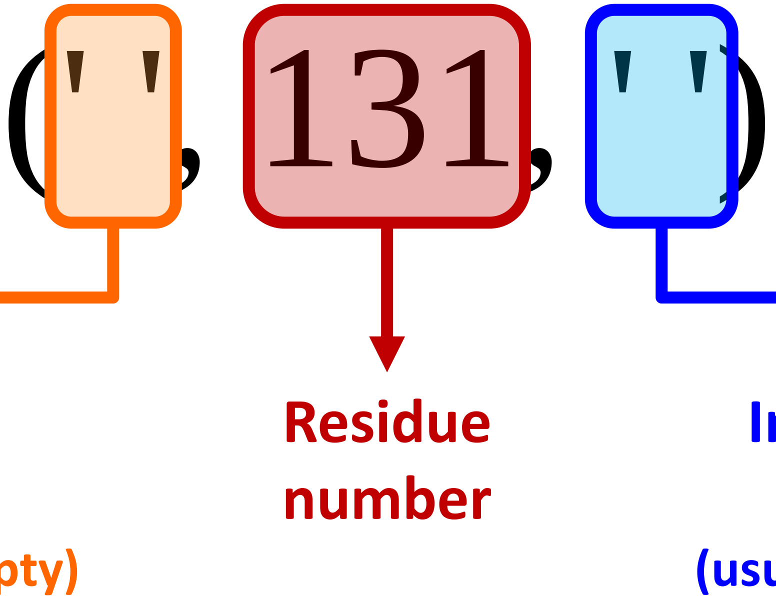
The residue object has a complicated ID, which is a tupple of 3 elements:

(' ', 131, ' ')

## The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

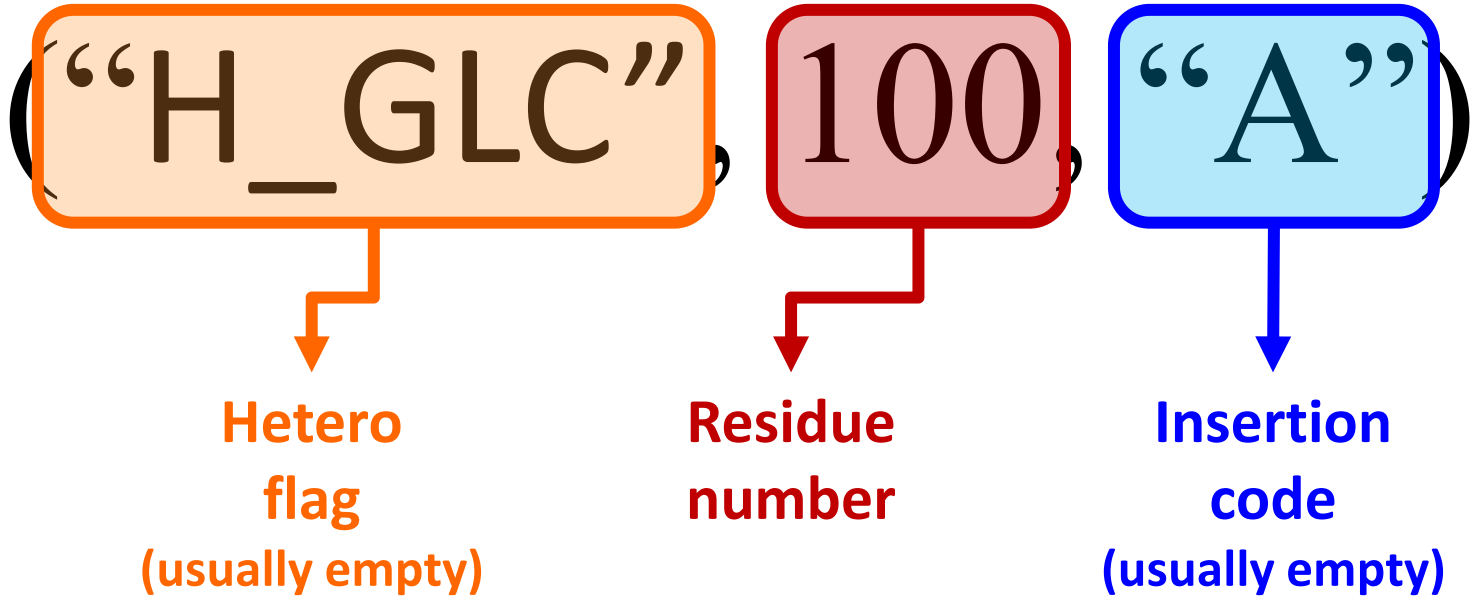
The residue object has a complicated ID, which is a tupple of 3 elements:



# The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

The residue object has a complicated ID, which is a tupple of 3 elements:



## The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

You can access the atoms by indexing the residue object with the atom identifier, or iterate them by using the get_atoms() function

```
>>> atom_CA = residue[“CA”]
```

```
>>> for atom in residue.get_atoms()
```

## The residue object

**The residue object is the fourth in the scale.**
**It contains one or more atoms within itself.**

You can access the atoms by indexing the residue object with the atom identifier, or iterate them by using the get_atoms() function

```
>>> atom_CA = residue[“CA”]
```

With this command I select the alpha carbon.

```
>>> atom_N = residue[“N”]
```

With this command I select the nitrogen from the amino group.

```
>>> for atom in residue.get_atoms()
```

# The atom object

**The atom object is the last in the scale.**
**There are no more objects within the atom object.**

The atom object is very useful to deal with molecule geometry, since it provides the coordinates for the position of each atom and allows the user to:

Get coordinates for an atom:

```
>>> CA.get_coord()
array([ -3.285, -10.587,   0.405], dtype=float32)
```

Calculate distances (in Ångstroms):

```
>>> distance = atomA - atomB
```

## The atom object

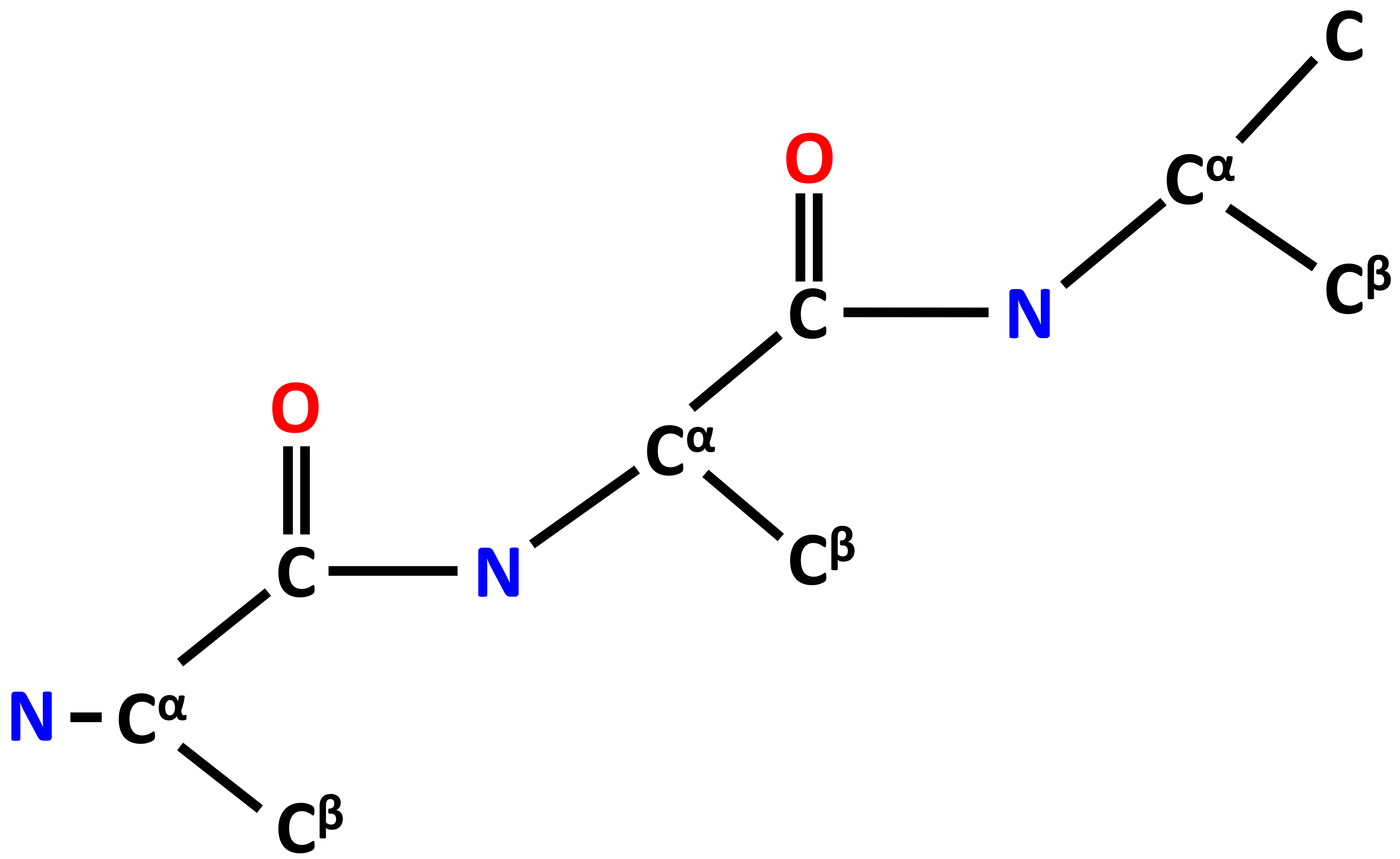
**The atom object is the last in the scale.**
**There are no more objects within the atom object.**

The atom object is very useful to deal with molecule geometry, since it provides the coordinates for the position of each atom and allows the user to:

Calculate the dihedrals between 4 atoms:

```
>>> v1 = chain[res_id][atom_name1].get_vector()
>>> v2 = chain[res_id+1][atom_name2].get_vector()
>>> v3 = chain[res_id+1][atom_name3].get_vector()
>>> v4 = chain[res_id+1][atom_name1].get_vector()
>>> phi_dihedral = calc_dihedral(v1, v2, v3, v4)
```

## Understanding protein geometry



## Calculate distances using biopython.PDB

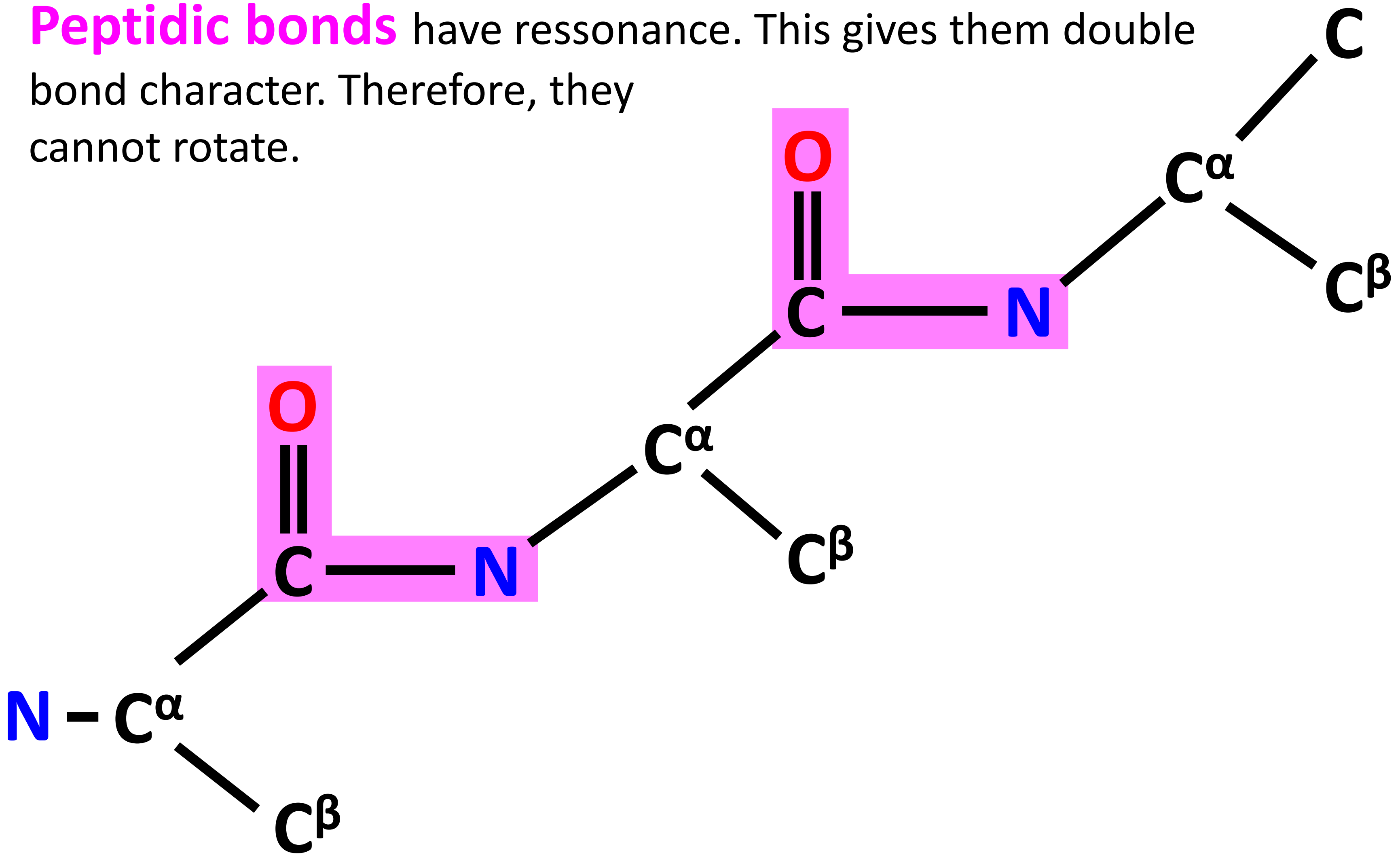
**Analyze the PDB file that we are visualizing (2dkt)**

Calculate distances between:

- Main chain nitrogen of Lysine 132 and carbonil oxigen of histidine 133.
- Alpha carbon of Lysine 132 and beta carbon of histidine 133.
- Carbonil oxygen of lysine 132 and carbonil oxygen of histidine 133.

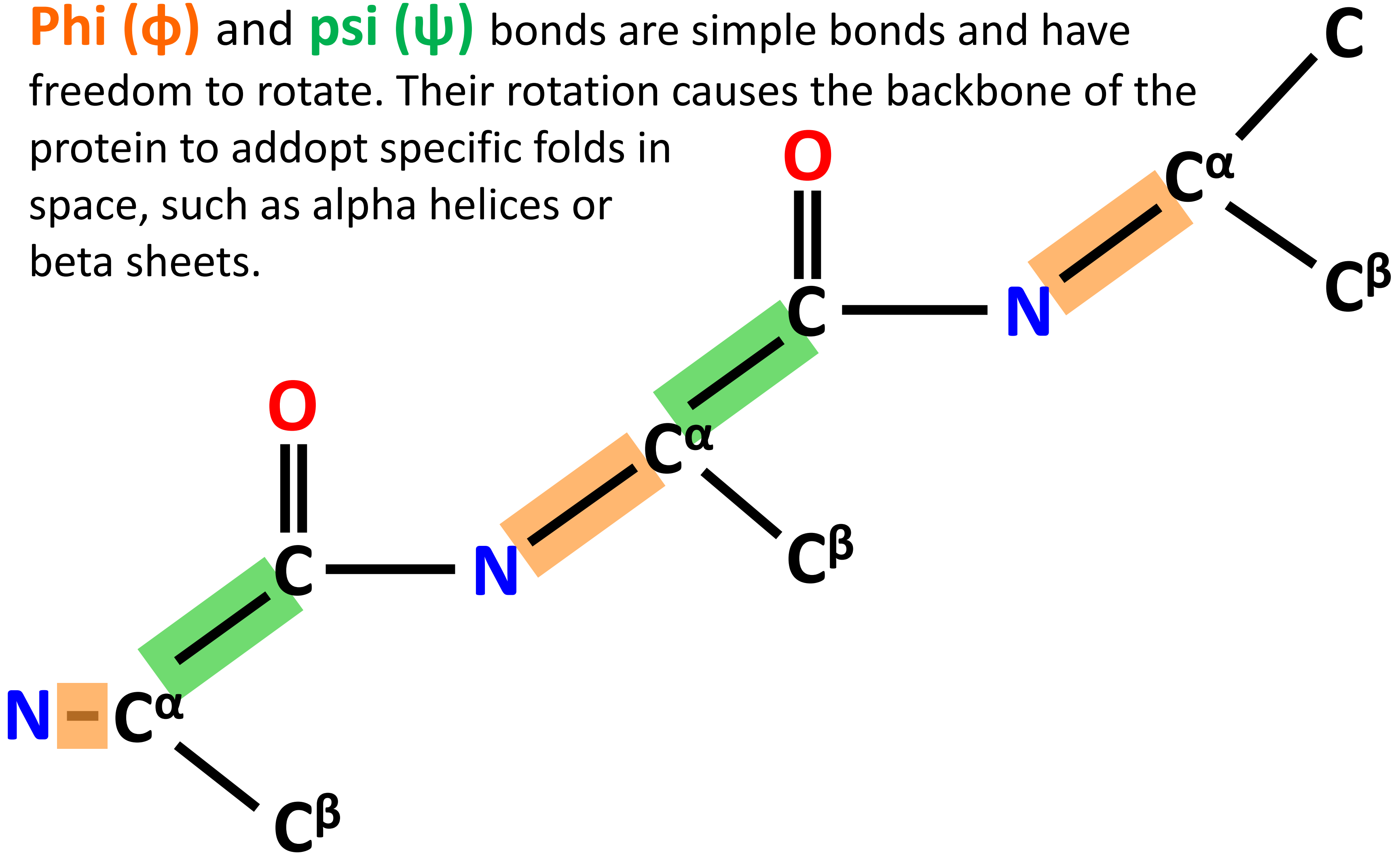
# Understanding protein geometry

**Peptidic bonds** have ressonance. This gives them double bond character. Therefore, they cannot rotate.



# Understanding protein geometry

**Phi (ϕ)** and **psi (ψ)** bonds are simple bonds and have freedom to rotate. Their rotation causes the backbone of the protein to addopt specific folds in space, such as alpha helices or beta sheets.

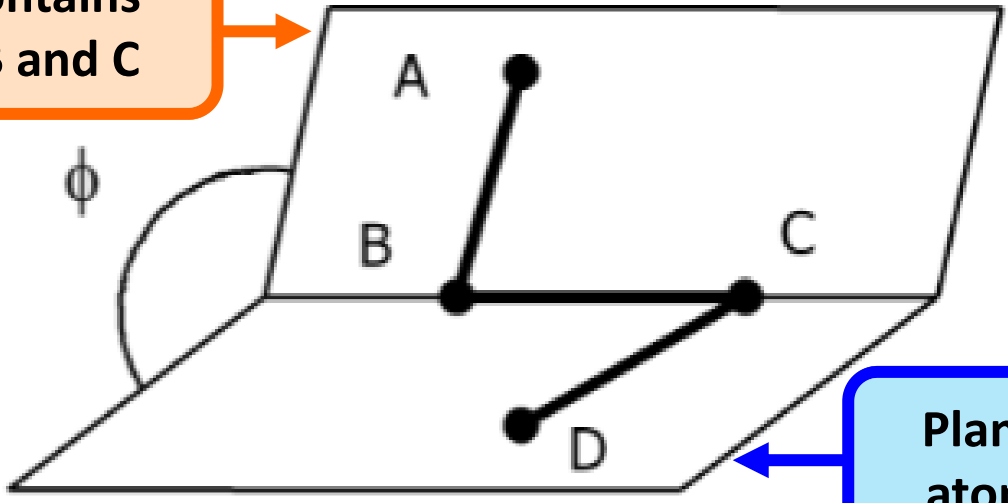


## Understanding protein geometry

**Remember biochemistry, dihedrals are the angle created by two planes**

To calculate the dihedral of a bond you select the atoms involved in the bond (B and C) and one atom before and another atom after (A and D)

**Plane 1: contains atoms A, B and C**



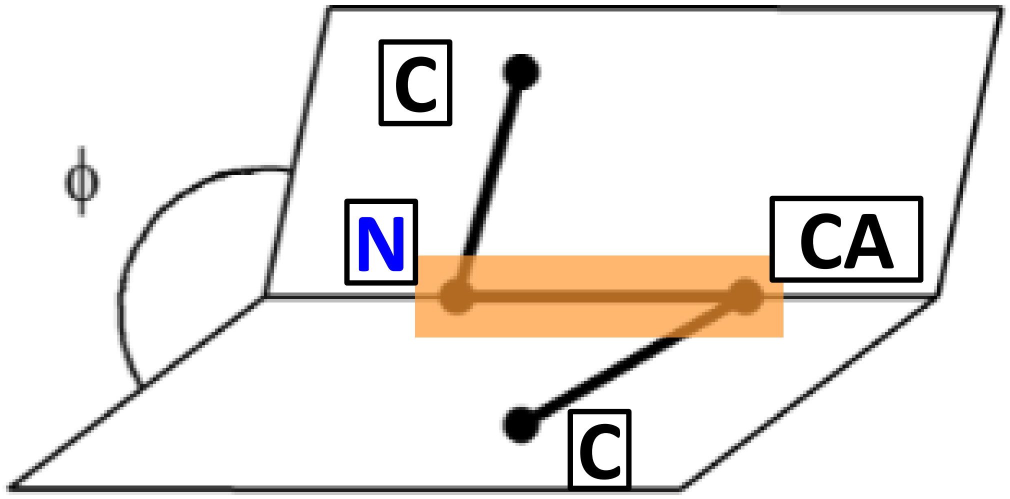
**Plane 2: Contains atoms B, C and D**

Dihedral Angle ϕ

# Understanding protein geometry

**Remember biochemistry, dihedrals are the angle created by two planes**

For the **Phi (ϕ)** angle, the involved atoms are the carbonil carbon and the nitrogen of one amino acid; and the alpha carbon and carbonil carbon of the next amino acid

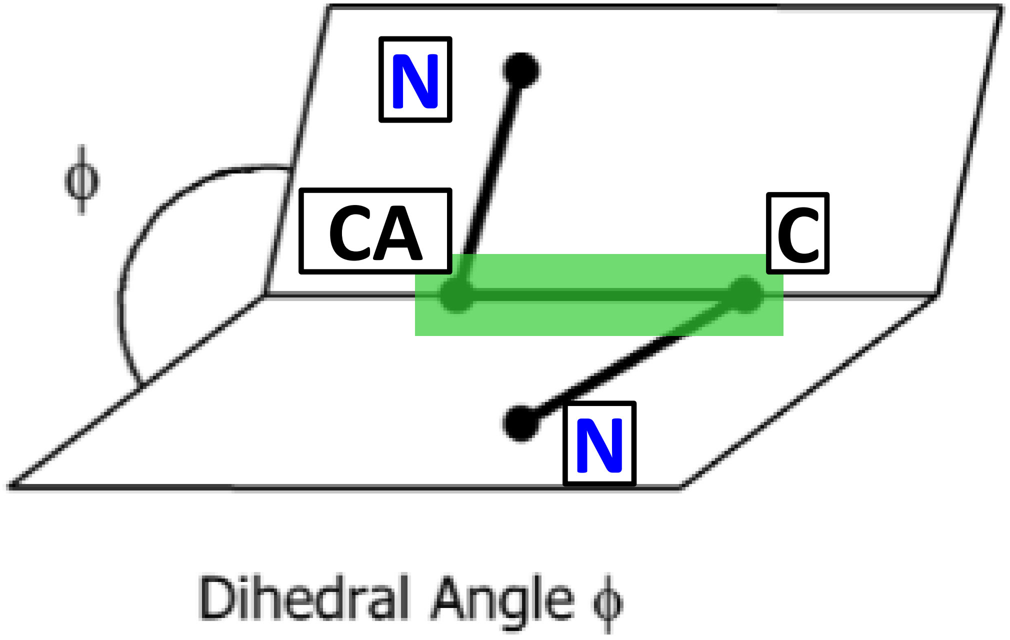


Dihedral Angle ϕ

# Understanding protein geometry

**Remember biochemistry, dihedrals are the angle created by two planes**

For the **Psi (ψ)** angle, the involved atoms are the the nitrogen of one amino acid; and the alpha carbon, carbonil carbon and nitrogen of the next amino acid



## Obtain dihedrals using biopython.PDB

**Analyze the PDB file that we are visualizing (2dkt)**

Use biopython to calculate the phi (ϕ) and psi (ψ) angles for the following residues:

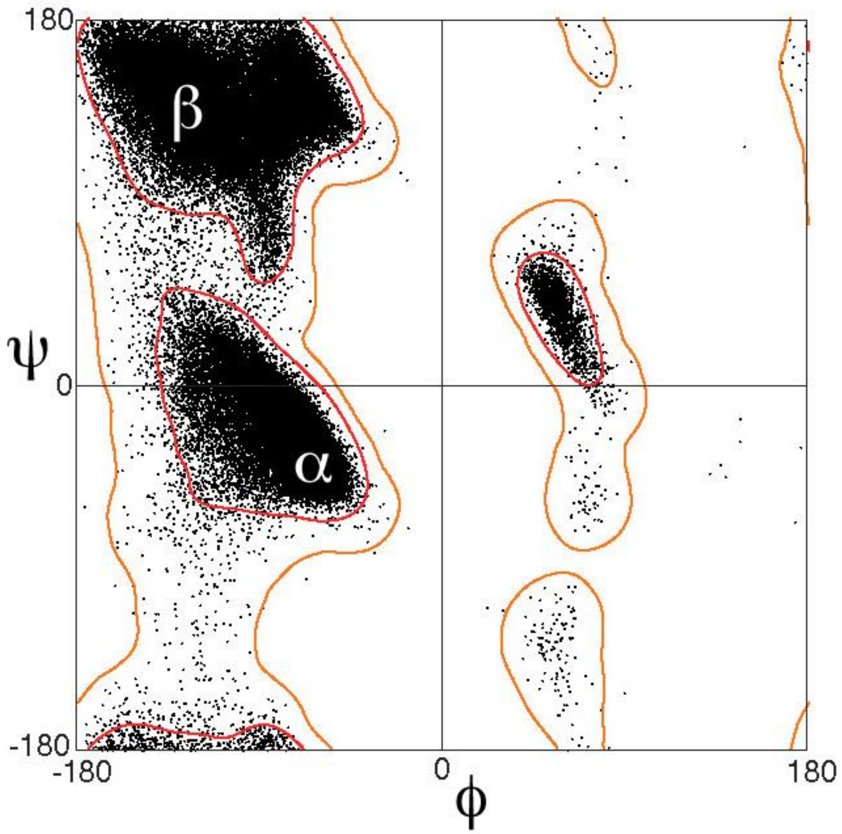
- Lysine 132
- Histidine 133
- Any amino acid you want...

Take this code as an example:

```
>>> v1 = chain[res_id][atom_name1].get_vector()
>>> v2 = chain[res_id+1][atom_name2].get_vector()
>>> v3 = chain[res_id+1][atom_name3].get_vector()
>>> v4 = chain[res_id+1][atom_name1].get_vector()
>>> phi_dihedral = calc_dihedral(v1, v2, v3, v4)
```

## The Ramachandran plot

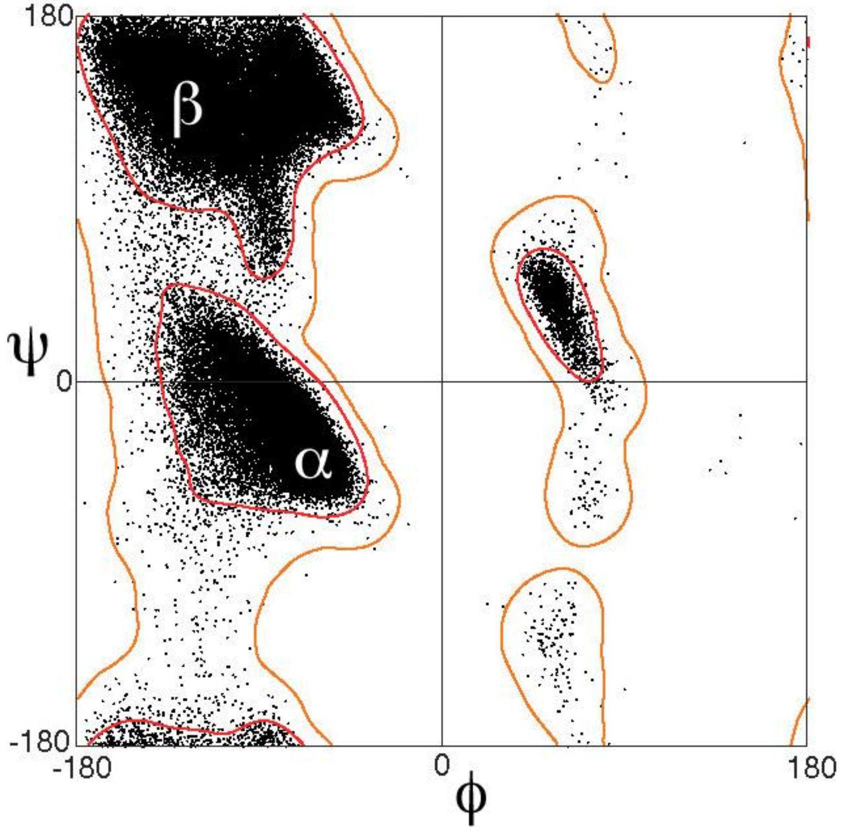
**The Ramachandran plot is a representation of the phi (ϕ) and psi (ψ) dihedrals in a protein.**



## The Ramachandran plot

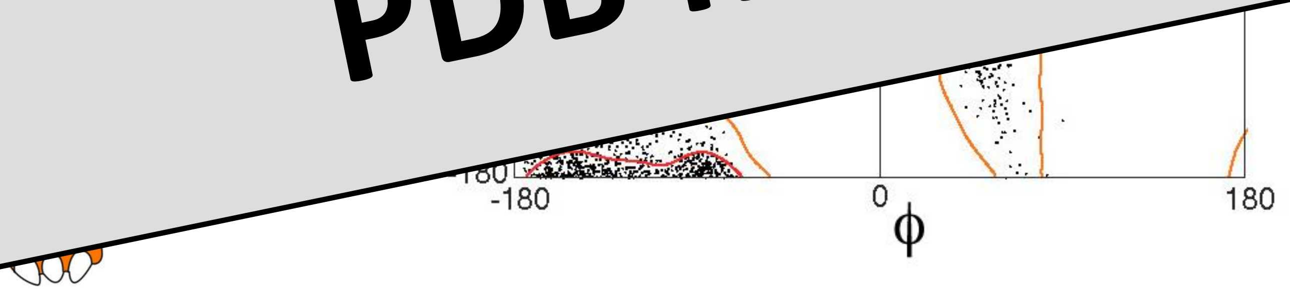
**The Ramachandran plot is a representation of the phi (ϕ) and psi (ψ) dihedrals in a protein.**

Do not confound Ramachandran with charmander (it happened to a friend!)



# The Ramachandran plot

The Ramachandran pl



**Write a script to make ramachandran plots taking as input a PDB file**

## A challenge for the weekend

**Using your knowledge on biochemistry and biopython, make a script that counts how many hydrogen bonds you can find in a protein structure**

The hydrogen bonds, I mean...

## Some links to explore biopython.PDB

Biopython.PDB FAQ:
https://biopython.org/wiki/The_Biopython_Structural_Bioinformatics_FAQ

Biopython.PDB documentation:
https://biopython.org/docs/1.75/api/Bio.PDB.html

Biopython.PDB cookbook:
http://biopython.org/DIST/docs/tutorial/Tutorial.html#sec182