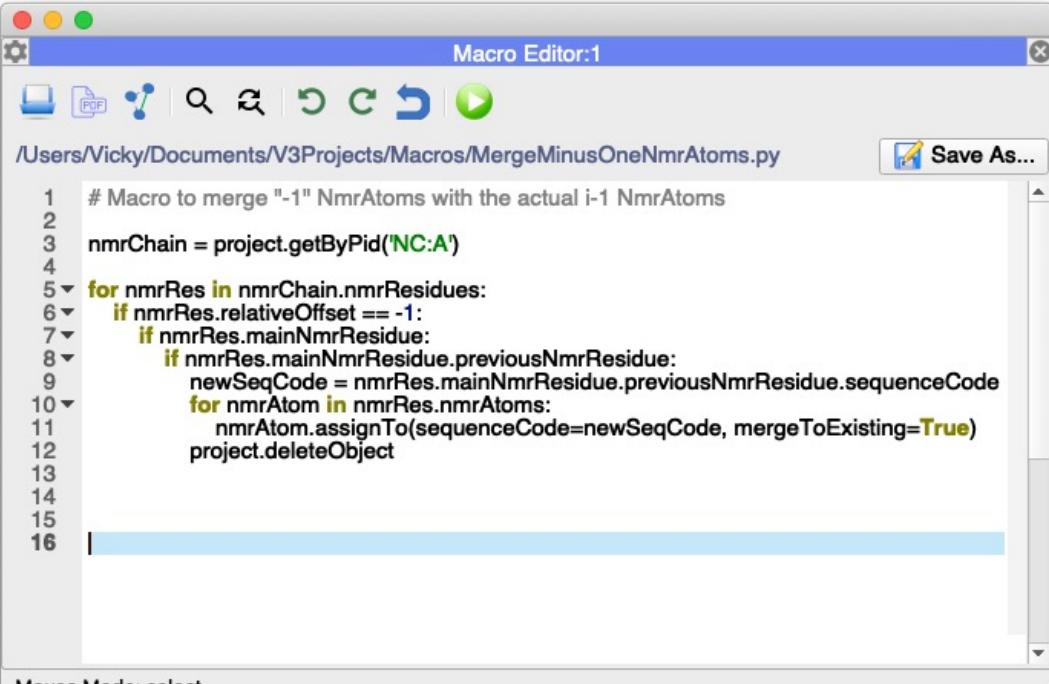


Macro Writing Tutorial



The screenshot shows the CCPN Macro Editor window titled "Macro Editor:1". The file path is "/Users/Vicky/Documents/V3Projects/Macros/MergeMinusOneNmrAtoms.py". The code in the editor is:

```
1 # Macro to merge "-1" NmrAtoms with the actual i-1 NmrAtoms
2
3 nmrChain = project.getByPid('NC:A')
4
5 for nmrRes in nmrChain.nmrResidues:
6     if nmrRes.relativeOffset == -1:
7         if nmrRes.mainNmrResidue:
8             if nmrRes.mainNmrResidue.previousNmrResidue:
9                 newSeqCode = nmrRes.mainNmrResidue.previousNmrResidue.sequenceCode
10                for nmrAtom in nmrRes.nmrAtoms:
11                    nmrAtom.assignTo(sequenceCode=newSeqCode, mergeToExisting=True)
12                project.deleteObject
13
14
15
16
```

At the bottom left of the editor, it says "Mouse Mode: select".

Introduction

This tutorial is designed to introduce you to macro writing in CcpNmr Analysis 3.1. It begins by taking you through some basic concepts in object oriented programming and the CcpNmr Analysis data structure. This is followed by a series of worked examples and problems which gradually increase in complexity. It is assumed that you have some basic familiarity with the CcpNmr Analysis programme (e.g. from having completed our Beginners Tutorial). It will also help if you have some basic knowledge of Python programming (e.g. indentations, conditionals and loops).

You can use your own NMR data or that located in the **/CcpnTutorialDataMacroWritingApril22** directory provided by CCPN.

Please note that the images shown are only representative and you may encounter minor differences in your setup.

Contents:

1. CcpNmr Analysis data structure
2. Using the Python Console
3. Using the Macro Editor
4. Finding methods and properties
5. Loops and conditionals
6. Adding an Undo Block
7. Subroutines / Functions
8. Worked Example
9. Advanced Macro Writing

Start CcpNmr Analysis V3

Apple users by double-clicking the *CcpNmrAnalysis* icon



Linux users by using the terminal command:
bin/assign

Windows users by double-clicking on the *assign.bat* file

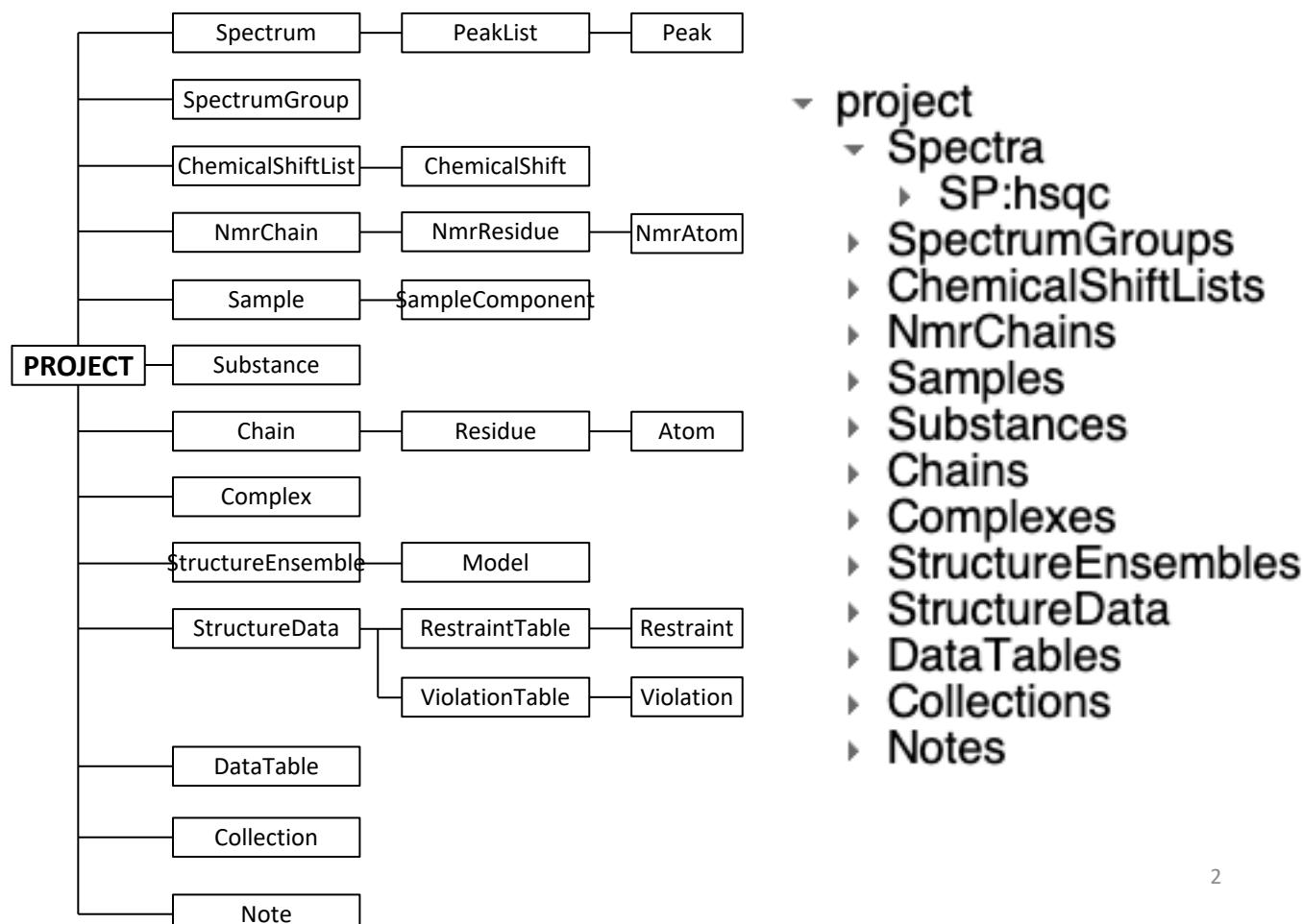
CcpNmr Analysis data structure

Classes, objects, methods and properties in CcpNmr Analysis V3 (Explanation only)

The main types of data contained within Analysis are shown below (left). Some of these are nested within each other, e.g. a peak is always contained within a peak list and a peak list is always associated with a spectrum. The data structure of the programme is reflected in the way the data is shown in the sidebar of Analysis (right).

Each data type forms a so-called class within the programme. Actual pieces of data within these classes are referred to as objects. Each class has a variety of properties (think of them as variables) and methods (essentially subroutines or functions) associated with it, i.e. different types of data (e.g. spectra, peak lists, peaks, chemical shifts, NmrAtoms etc.) automatically have certain properties and methods associated with them. For example, spectrum.name (a property) would give you the name of a spectrum and spectrum.rename(value='HSQC') (a method) would change your spectrum's name to HSQC.

Thus, if you want to write a macro, there are a large number of routines that you can draw on to access and manipulate all the data in your project. The tutorial will show you how to find and use these routines to change or display your data as you would like.



Project IDs (Pids)

(Explanation only)

Each object within Analysis has its own unique Project ID (or PID/Pid).

These Pids contain a short code to signify the data type (e.g. SP for spectrum – see next page for a full list) followed by the name of the data item, e.g. SP:hsqc. Nested items also contain the names of their parents/grandparents, separated by a full stop. For example, the PID of a peak list is given as PL:spectrumName.peakListName or of an NmrAtom as NA:A.23.THR.CA.

These Pids are very useful during macro writing for getting hold of particular items of data. You will see in Section 2C that get('Pid') will give you the object with that Pid for you to do something with (e.g. execute a method on, assign to a variable etc.).

You can easily find an object's Pid, as they are all shown in the sidebar and you can also view them in tables. You can also copy them to the clipboard via the right-hand mouse menu.

A full list of PIDs with examples is given on the next page.

- project
 - Spectra
 - ▶ SP:hsqc
 - ▶ SP:hnca**
 - PeakLists
 - <New PeakList>
 - PL:hnca.1**
 - PL:hnca.1
 - right-click
 - ▶ integralList
 - ▶ SpectrumGroup
 - ▶ ChemicalShiftList
 - ▶ NmrChains
 - ▶ Samples
 - ▶ Substances
 - ▶ Chains
 - ▶ Complexes
 - ▶ StructureEnsembles
 - ▶ StructureData
 - ▶ DataTables
 - ▶ Collections
 - ▶ Notes

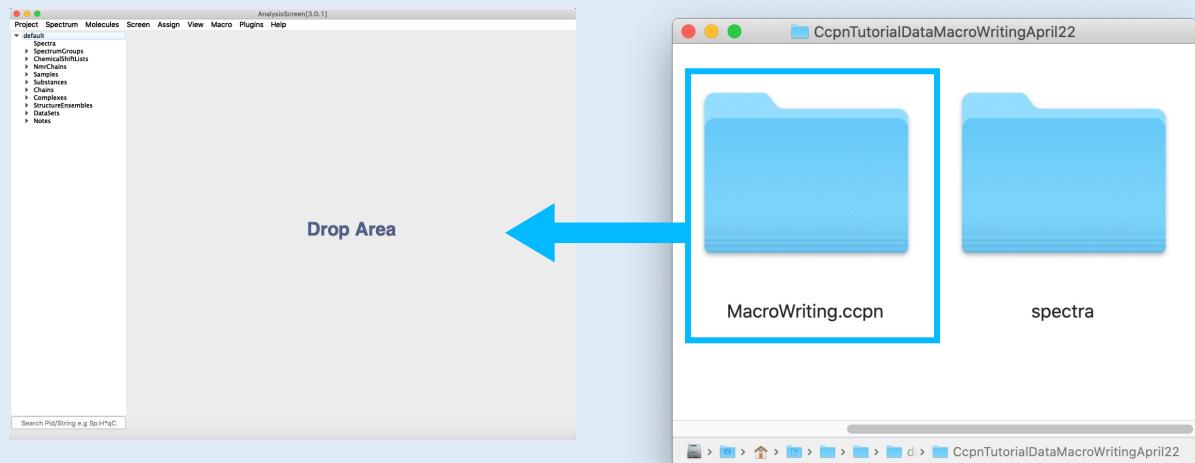
The screenshot shows a software interface for CCP-NMR Analysis. On the left, there is a sidebar with a tree view of the project structure. A context menu is open over a row in a table on the right. The menu options include "Open as a Module", "Copy Pid to clipboard", "Add to Collection", "Remove from Collection", and "Edit Properties". The table itself displays a list of peaks with columns for Pid, Assign F1, Assign F2, Assign F3, and Pos F1. The "Pid" column is highlighted with a blue border. The "Assign F1" column contains labels like "@-.@1..H", "@-.@1..H", and "@-.@3..H". The "Assign F2" and "Assign F3" columns show chemical shifts, and the "Pos F1" column shows values like 8.595, 9.669, and 8.188. A "Column Settings..." button is visible at the top of the table, and a "Display Columns" section at the bottom allows for selecting or deselecting columns like "#", "Pid", and "Spectrum".

#	Pid	Assign F1	Assign F2	Assign F3	Pos F1
1	PK:hnca.1.1	@-.@1..H			8.595
2	PK:hnca.1.2	@-.@1..H			
3	PK:hnca.1.3	@-.@3..H			
		@-.@3..H			
11	PK:hnca.1.11	@-.@6..H			9.669
12	PK:hnca.1.12	@-.@8..H			8.188
13	PK:hnca.1.13	@-.@8..H			
14	PK:hnca.1.14	@-.@9..H			
15	PK:hnca.1.15	@-.@9..H	@-.@9-1..CA	@-.@9..N	8.660
16	PK:hnca.1.16	@-.@10..H	@-.@10..CA	@-.@10..N	9.189
17	PK:hnca.1.17	@-.@10..H	@-.@10-1..CA	@-.@10..N	9.189
18	PK:hnca.1.18	@-.@11..H	@-.@11..CA	@-.@11..N	8.034
19	PK:hnca.1.19	@-.@12..H	@-.@12..CA	@-.@12..N	7.464

Project IDs in Analysis

PID	Data Object	Nesting	Example
Short Code			
PR	Project	Project	PR:GP41fab8066Complex
SP	Spectrum	Spectrum	SP:hsqc
PL	Peak List	Spectrum / Peak List	PL:hsqc.1
PK	Peak	Spectrum / Peak List / Peak	PK:hsqc.1.1
ML	Multiplet List	Spectrum / Multiplet List	ML:hsqc.1
MT	Multiplet	Spectrum / Multiplet List / Multiplet	MT:hsqc.1.1
IL	Integral List	Spectrum / Integral List	IL:hsqc.1
IT	Integral	Spectrum / Integral List / Integral	IT:hsqc.1.1
SG	Spectrum Group	Spectrum Group	SG:T1Data
CL	Chemical Shift List	Chemical Shift List	CL:default
CS	Chemical Shift	Chemical Shift List / Chemical Shift	CS:default.A.23.THR.H
NC	NmrChain	NmrChain	NC:A
NR	NmrResidue	NmrChain / NmrResidue	NR:A.23.THR
NA	NmrAtom	NmrChain / NmrResidue / NmrAtom	NA:A.23.THR.CA
SA	Sample	Sample	SA:complex
SC	Sample Component	Sample / Sample Component	SC:complex.GP41
SU	Substance	Substance	SU:GP41
MC	(Molecule) Chain	Chain	MC:A
MR	(Molecule) Residue	Chain / Residue	MR:A.23.THR
MA	(Molecule) Atom	Chain / Residue / Atom	MA:A.23.THR.H
MX	(Molecule) Complex	Complex	MX:GP41-fab8066
SE	Structure Ensemble	Structure Ensemble	SE:4kht
SD	Structure Data	Structure Data	DS:run1
RT	Restraint Table	Structure Data / Restraint Table	RT:run1.TalosRestraints
RE	Restraint	Data Set / Restraint List / Restraint	RE:run1.TalosRestraints.1
DT	Data Table	Data Table	DT:RamachandranData
CO	Collection	Collection	CO:run1
NO	Note	Note	NO:note1
GS	GUI Strip	GUI Strip	GS:2D_HN.1

Using the Python Console



```
Python Console
application.showNmrResidueTable(position='bottom', relativeTo=None, nmrChain=None, selectFirstItem=True)

Jupyter QtConsole 5.1.1
Python 3.8.10 (default, May 19 2021, 11:01:55)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

2A Open Spectra/Project

- Drag the **CcpnTutorialDataMacroWritingApril22/MacroWriting ccpn** project into the Drop Area.

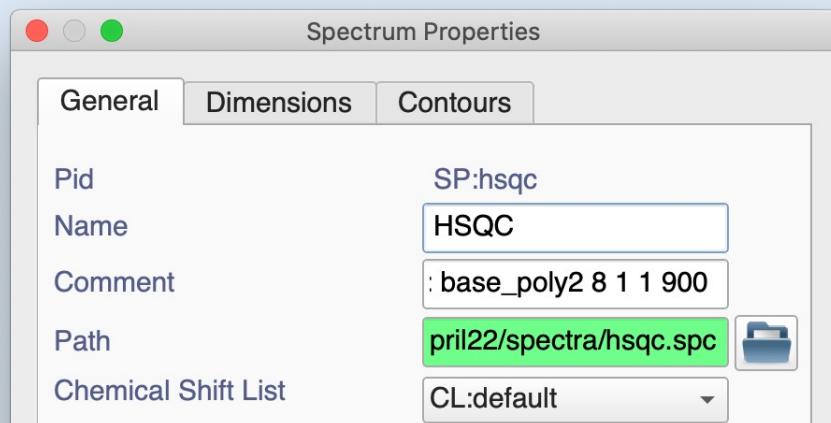
2B Open Python Console

- Go to Main Menu → View → Python Console
- OR
- Use the shortcut **Space-Space**

The top half of the console will echo commands that you execute using the graphical user interface (GUI).

The lower half is where you can start typing commands.

Using the Python Console



Python Console

```
get('SP:hsqc').rename(value='HSQC')
get('SP:HSQC').rename(value='HSQC_Sec5')

Jupyter QtConsole 5.1.1
Python 3.8.10 (default, May 19 2021, 11:01:55)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [2]: get('SP:HSQC').rename(value='HSQC_Sec5')
Out[2]: True

In [3]:
```

2C Rename Spectrum in the GUI

- Double-click on the **hsqc** spectrum in the sidebar to bring up the **Spectrum Properties** dialog box.
- Change the name of the spectrum to **HSQC** and click **OK**.

In the top half of the Python Console you should see an echo of the command that renamed your spectrum:

```
get('SP:hsqc').rename(value='HSQC')
```

2D Rename Spectrum in the Python Console

- Now have a go at renaming your spectrum in the lower part of the python console by typing

```
get('SP:HSQC').rename(value='HSQC_Sec5')
```

The screenshot shows a Python Console window with the title bar "Python Console". The console area contains several lines of Python code and their corresponding output. The code includes operations like renaming spectra, picking peaks, and setting spectrum properties. It also shows the Jupyter QtConsole version, Python version, and IPython version. The output includes command history ("In [2]:" and "Out[2]:") and a blank line ("In [3]:").

```

get('SP:hsqc').rename(value='HSQC')
get('SP:HSQC').rename(value='HSQC_Sec5')
mainWindow.newSpectrumDisplay('SP:HSQC_Sec5', axisCodes=('H', 'N'), stripDirection='Y', position='top',
relativeTo='M0:Python Console')
get('GS:2D_HN.1').pickPeaks(regions=[(9.186, 8.475), (113.139, 117.558)])
get('SP:HSQC_Sec5').positiveContourColour = '#8A2BE2'
mainWindow.newSpectrumDisplay('SP:SH3_2C_PDS50', axisCodes=('C', 'C_2'), stripDirection='Y',
position='left', relativeTo='GD:2D_HN')
get('SP:SH3_2C_PDS50').spinningRate = 8000.0
In [2]: get('SP:HSQC').rename(value='HSQC_Sec5')
Out[2]: True
In [3]:

```

2E Try other commands

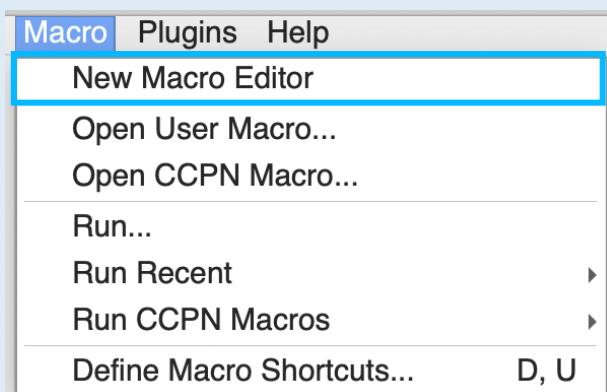
Try finding some other commands in the GUI and then executing them in the Python Console e.g.

- Peak pick a region in the HSQC_Sec5 spectrum with **Shift+Ctrl** (**Shift+Cmd** on a Mac) and **left-drag**.
- Change the spectrum contour colours (e.g. with shortcut **CO**).
- Double-click on the **sh3_2c_pdsd** in the sidebar to bring up the Spectrum Properties and in the **General** tab, set the **Spinning Rate** to **8000**. (You may afterwards like to **right-click** on the spectrum and choose **Customise / Show MAS Sidebands** to see the effect of choosing different spinning rates.)
- Move a peak by selecting it and dragging it with the middle mouse button.
- Go to **Main Menu** → **Assign** → **Set up NmrResidues** and do this for the **HSQC_Sec5** spectrum
- Open the **Peak Assigner** with **AP** and Assign/Deassign some peaks.

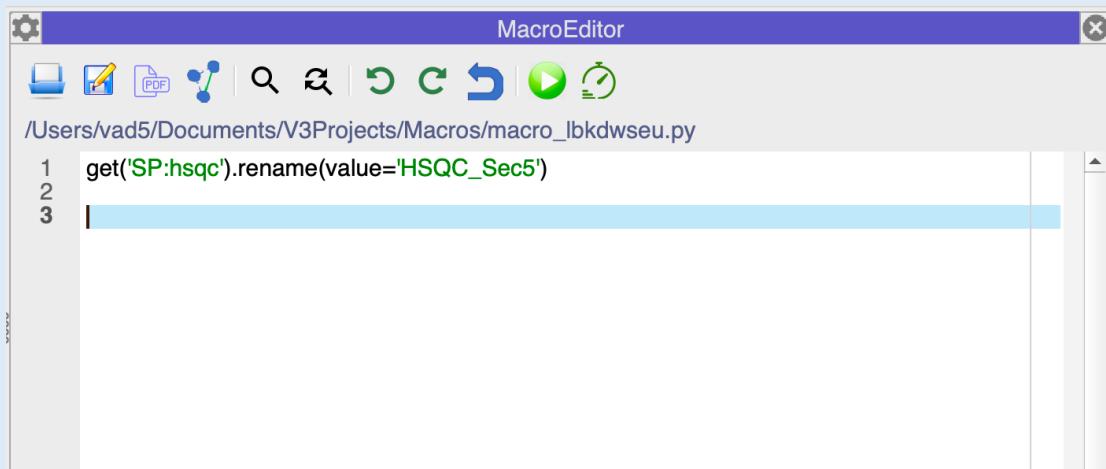
Notice that in some cases, a single operation on the GUI actually executes several commands on the Console.

Opening a new module is also echoed.

Using the Macro Editor



NM



3A Open Macro editor

- Go to Main Menu → Macro → New Macro Editor
- OR
- use shortcut NM.

3B Write and Run Macro

- Type some of your commands from Section 2 into the Macro Editor, e.g.
`get('SP:HSQC').rename(value='Sec5_HSQC')`

- Click on to execute the macro

- Try executing several commands in one go, e.g.

```

get('SP:hncacb').rename(value='HNCACB')
get('SP:cbcacnh').rename(value='CBCACONH')

```

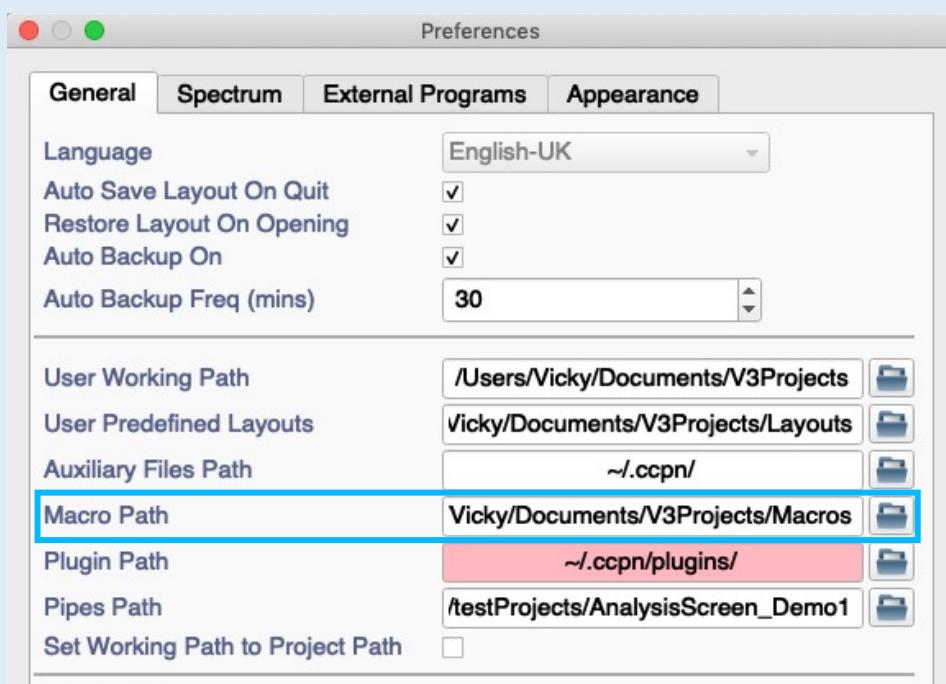
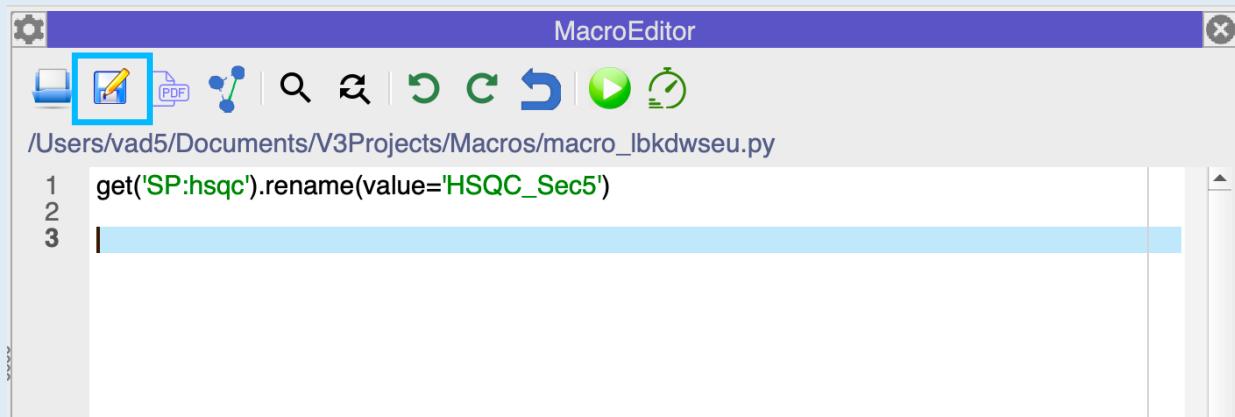
OR

```

get('SP:HSQC_Sec5').rename(value='HSQC')
get('SP:HSQC').positiveContourColour = '#FFC800'
get('GS:2D_HN.1').pickPeaks(regions=[(11.0, 5.0), (100.0, 135.0)])

```

Using the Macro Editor



3c Save Macro

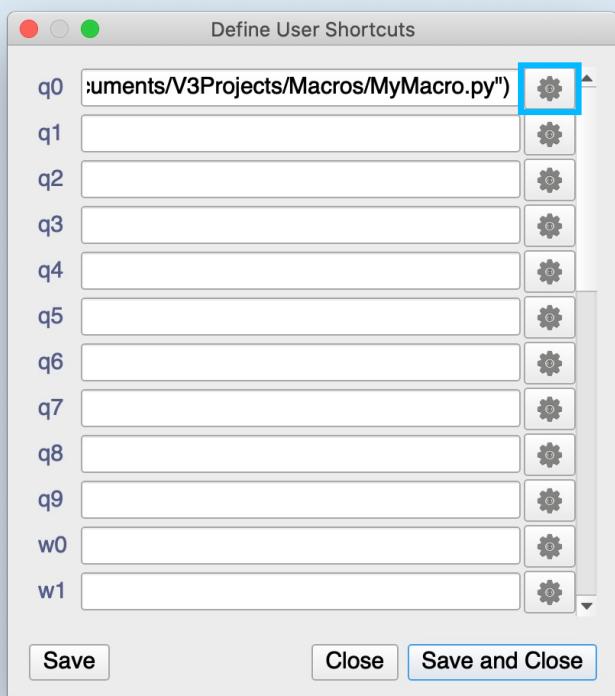
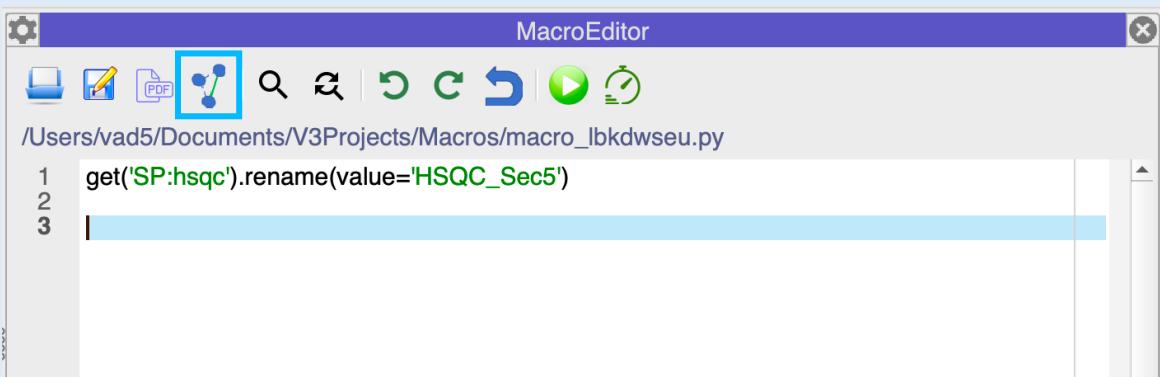
- Save one of your macros by clicking on the Save icon.



Note that you can set a default Macro directory in which to save all your macros in your User Preferences. To change this directory:

- Go to **Main Menu → Project → Preferences...**
- Change **Macro Path** to whichever directory you want to use instead.

Using the Macro Editor

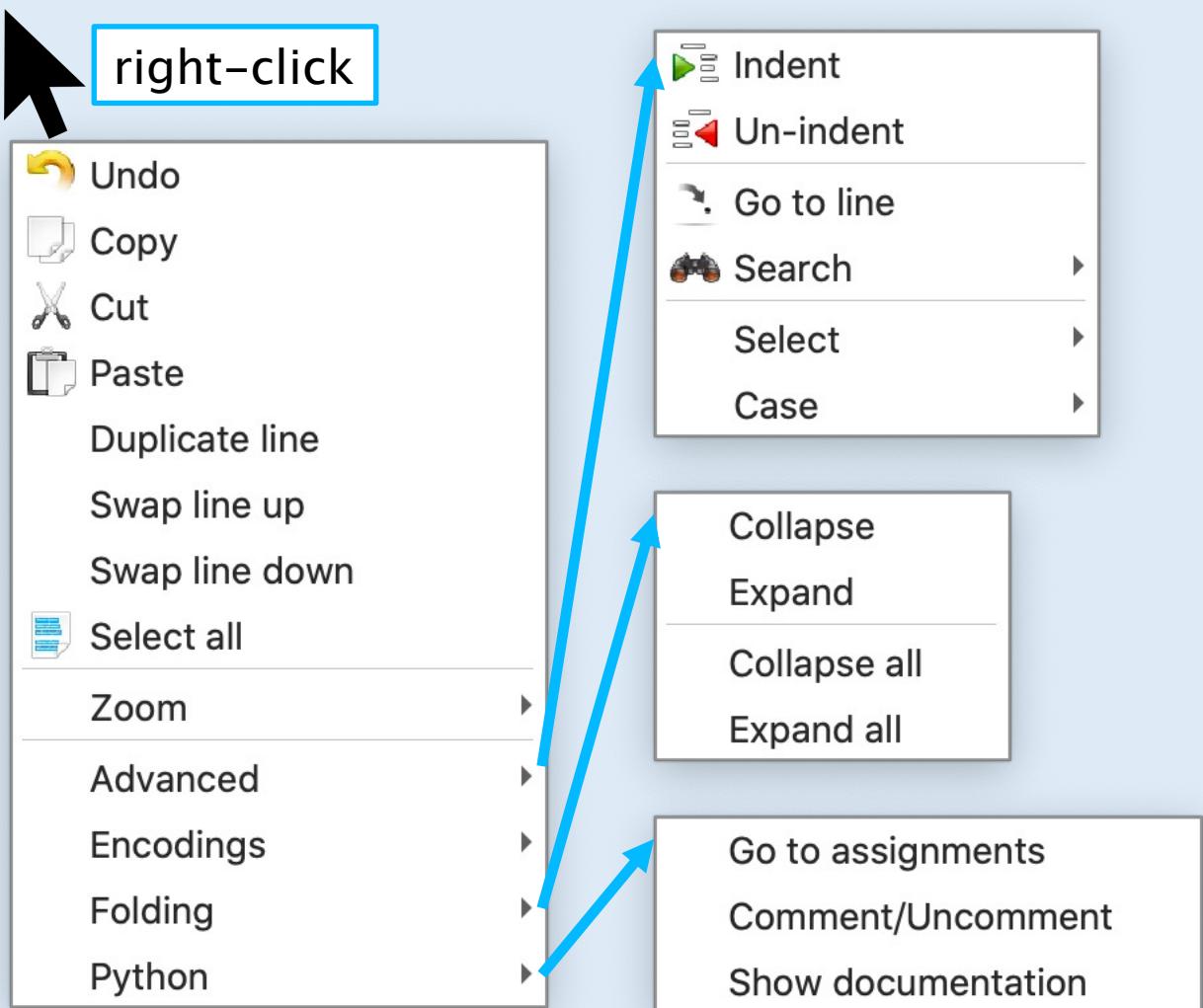


3D Link macro to keyboard shortcut

- Click on in the button in the toolbar menu
- OR
- Go to **Main Menu → Macro → Define Macro Shortcuts...**
- OR
- Use the shortcut **DU**.

Select a shortcut code from the list and associate your macro with it by **clicking on the gear button** and selecting your macro in the file browser.

Using the Macro Editor



3E More Macro Editor Features

The **right-hand mouse menu** in the Macro Editor gives you access to a number of useful features, including being able to Comment/Uncomment, Indent/Un-indent or Collapse/Expand sections easily.

Advanced users may also find it useful to run more complicated macros using a profiler to help you identify areas for improvement. Rather than using the play button, run the macro with the stop watch icon:

Finding Methods and Properties

```

Python Console:1
mainWindow._deleteSpectrumDisplay(display='GD:HN')

In [4]: spectrum = get('SP:hsqc')

In [5]: spectrum
Out[5]: <Spectrum:hsqc>

In [6]: spectrum. followed by Tab

```

4A Finding Methods and Properties via the Console

You can get a list of methods and properties belonging to an object in the Python Console.

- Following the example above, type

```
spectrum = get('SP:GB1_hsqc')
```

to assign the **SP:GB1_hsqc** object to the **spectrum** variable. (This is where **right-click / Copy Pid to Clipboard** can come in handy!)

- Typing

```
spectrum
```

should echo the object (see above) – a useful way to check for typos.

- Typing

```
spectrum.
```

followed by **Tab** will give you a list of the methods and properties available to a **spectrum** object.

- Type **Esc** to remove the list again.

You can also type

```
spectrum.p
```

followed by **Tab** in order to see all methods and properties beginning with **p** etc.

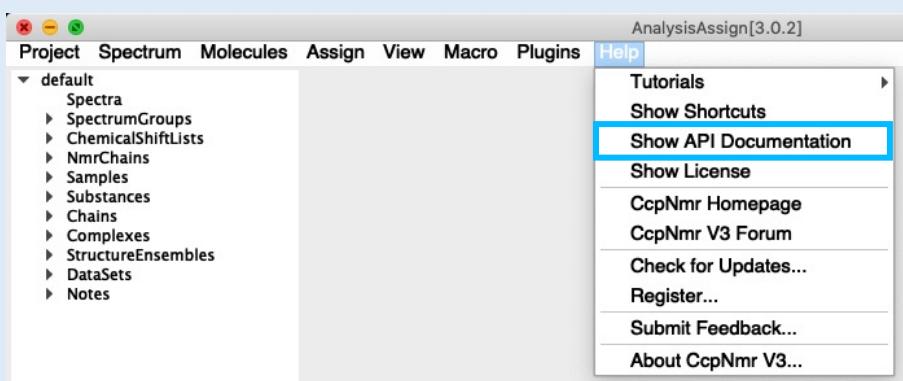
```

spectrum.positiveContourColour = '#FFBC00'
mainWindow._deleteSpectrumDisplay(display='GD:HN')

acquisitionAxisCode      getPlaneData      phases1
aliasingLimits           getPositionValue   pid
aliasingRange             getProjection     pointCounts
allPlanes                 getPseudoDimension pointOfSets
assignmentTolerances     getRegionData    positions
automaticIntegration     getSlice          positiveContourBase
axisCodes                 getSliceData     positiveContourColour
axisUnits                 getSpectrumHit   positiveContourCount
CCPNMR_NAMESPACE         getSpectrumReference positiveContourFactor
chemicalShiftlist         hasNotifier      preferredAxisOrdering
className                 hasParameter     printInfoString
comment                  headerSize       project
complexStoredBy          id                pseudoDimensions
copyParameters           includeNegativeContours referencePoints
defaultAssignmentTolerances includePositiveContours referenceSubstance
delete                   indices          referenceValues
deleteAllNotifier        integrallists   rename
deleteNotifier           integrals        resetSerial
dimensionCount           intensities     sample
dimensions               isComplex        scale
dimensionTypes           isDeleted        searchAxisCodePermutations
displayFoldedContours   isotopeCodes    searchNotifiers
doubleCrosshairOffsets   isValidPath    setBlankingAllNotifiers
estimateNoise            longPid         setByAxisCodes
experiment               lorentzianBroadenings setByDimensions
experimentName           magnetisationTransfers setGuiNotifier
experimentType           mainSpectrumReferences setNotifier
extendAliasingRangeFlag MAXIM           setParameter
extractPlaneToFile      measurementTypes shortClassName
extractProjectionToFile multipletLists  showDoubleCrosshair
extractSliceToFile       multiplets      sineWindowShifts
filePath                 name              sliceColour
foldingModes             negativeContourBase spectralWidths
gaussianBroadenings     negativeContourColour spectralWidthsHz
getAsDict                negativeContourCount spectrometerFrequencies
getByAxisCodes           negativeContourFactor spectrumGroups
getRvDimensions          negativeNoiseLevel spectrumHits

```

Finding Methods and Properties



Python documentation »

next | modules | index

Table of Contents

- Welcome to CCPN's documentation!
- CCPN Implementation layer (for experts only)
- Indices and tables

Next topic

CCPN

This Page

Show Source

Quick search

Go

Welcome to CCPN's documentation!

- CCPN
 - ccpn package
 - Subpackages
 - ccpn.AnalysisAssign package
 - ccpn.AnalysisMetabolomics package
 - ccpn.AnalysisScreen package
 - ccpn.AnalysisStructure package
 - **ccpn.core package**
 - ccpn.framework package
 - ccpn.pipes package
 - ccpn.plugins package
 - ccpn.ui package
 - ccpn.util package

4B Finding Methods and Properties via the API Documentation

- Go to Main Menu → Help → Show API Documentation to bring up the API Documentation.

(Go to Main Menu → Project → Preferences and select Use Native Web Browser to view the documentation in your usual web browser, or deselect to to view it in a module within the program.)

- Click on **ccpn.core package**.

The left-hand menu gives you a list of all the data types/classes. To view methods and properties associated with Peaks, for example, click on **ccpn.core.Peak module**.

- **ccpn.core.NmrResidue module**
- **ccpn.core.Note module**
- **ccpn.core.PMIListABC module**
- **ccpn.core.Peak module**
- **ccpn.core.PeakCluster module**
- **ccpn.core.PeakList module**

Methods are then shown in bold with the option of viewing the source code by clicking on [source]:

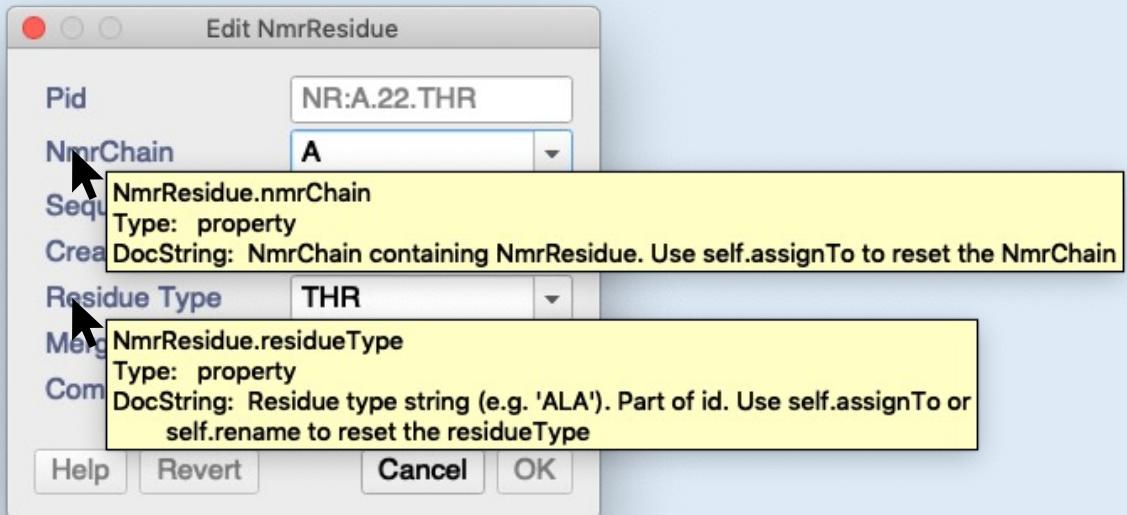
`addAssignment(value: Sequence[str / NmrAtom])`
Add a peak assignment - a list of one NmrAtom or Pid for each dimension

[source]

Properties are marked as properties:

`property aliasing`

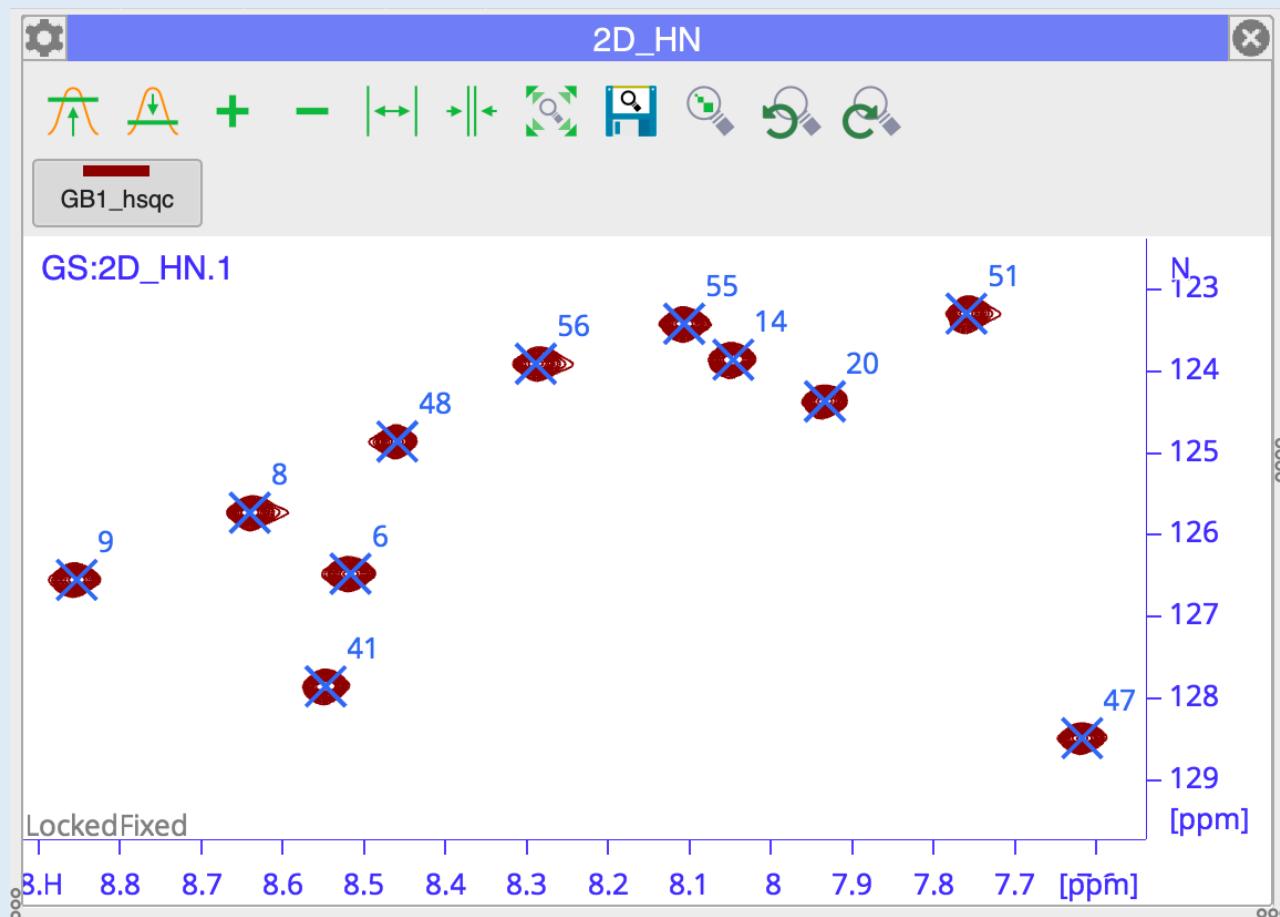
- `Tuple[float=None, ...], mutable` - Aliasing for the peak in each dimension. Defined as integer number of spectralWidths added or subtracted along each dimension



4C Finding Properties in pop-ups

- Double-click on an object in the sidebar to bring up a pop-up that will allow you to edit the object.
- Move the mouse over the descriptions on the left. Those which are properties of the object will bring up a tooltip after a few seconds which shows you the property name and a brief description.

The pop-ups do not contain an exhaustive list of all object properties, but it can be a quick way to find a property and some related methods.



4D Exercises

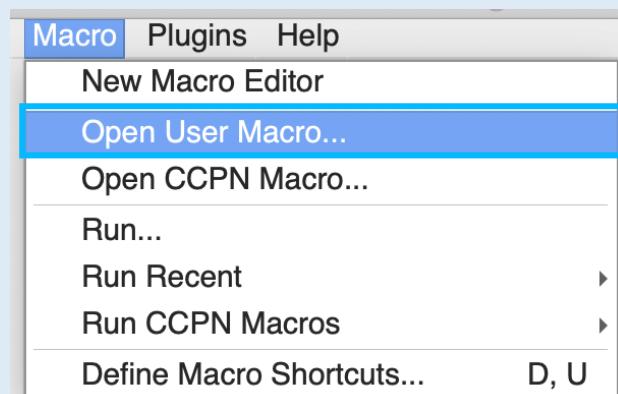
1. Write a short macro to change the symbol and peak label colours of to a blue colour (e.g. #3A6BEC).
2. Get a peak's position in ppm and in points. (hint: use `print()` to show your result)
3. Find the spectra associated with the 'default' chemical shift list.
4. Renumber the NC:GB1 NmrChain, increasing the sequence number by 2.
5. Count the number of peaks in the PL:GB1_hsqc.2 peak list. (hint: `len()` will measure the length of a list or tuple).

Possible answers are in the `CcpnTutorialDataMacroWritingApril22/macros` folder:

4D.1.changePeakListColours.py
 4D.2.printPeakPositions.py
 4D.3.printCSLSpectra.py
 4D.4.renumberNmrChain.py
 4D.5.countPeaks.py

Loops and conditionals

Often you will want to manipulate not just one item of data, but several. This section will show you how you can easily loop through items of data.



A screenshot of the CCPN Macro Editor showing a Python script named '5A.PeakListSetFigureOfMerit.py'. The code is as follows:

```

1 # Macro to set the peak merit value to a specific value for all the peaks in a peak list
2
3 pl = get('PL:HSQC.1')
4
5 -> for peak in pl.peaks:
6     peak.figureOfMerit = 0.0
7
8

```

The code sets the peak merit value to 0.0 for all peaks in a specified peak list.

5A Looping through Peaks in a Peak List

- Go to Main Menu → Macro → Open User Macro... and open the **5A.peakListSetFigureOfMerit.py** macro from the **CcpnTutorialDataMacroWritingApril22/macros** directory.

This macro will loop through all the peaks in a peak list and set the peak merit value to 0.0. This might be useful if you have a peak list whose peaks you don't want to contribute to the chemical shift list.

Comments:

Line 3 sets the peak list that will be used via its **PID**. Change this if you want to use a different peak list.

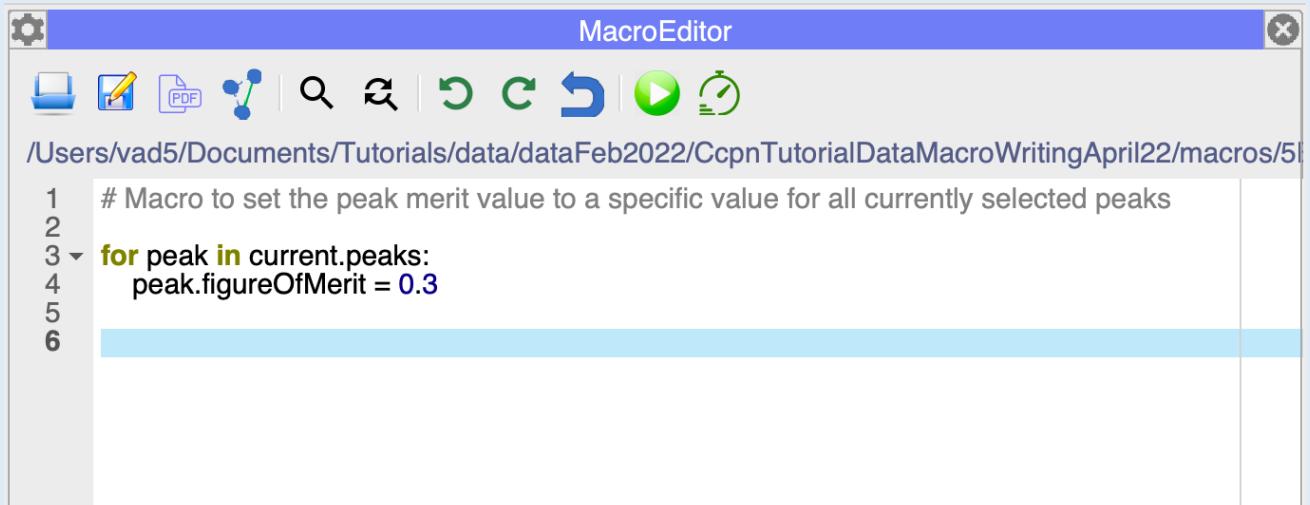
Line 5 loops through the peaks in the peak lists using **pl.peaks**. The **PeakList** property **peaks** contains all the **peak** objects in that peak list.

All data types / classes that have child classes, have such a property, e.g. all **NmrAtoms** in an **NmrResidue** are given by the **nmrAtoms** property, all **PeakLists** in a **Spectrum** are given by the **Spectrum** property **peakLists** etc.

Line 6 sets the new figure of merit to 0.

Loops and conditionals

In order to loop through currently selected items, you can use **current**.



The screenshot shows the MacroEditor software window. The title bar says "MacroEditor". Below the title bar is a toolbar with various icons: a file folder, a pencil, a PDF icon, a zoom-in/out icon, a search icon, a magnifying glass, a green circular arrow, a blue circular arrow, a play button, and a power button. The main area of the window displays a Python script. The path shown in the top bar is "/Users/vad5/Documents/Tutorials/data/dataFeb2022/CcpnTutorialDataMacroWritingApril22/macros/5B.currentPeaksSetPeakMerit.py". The script content is as follows:

```

1 # Macro to set the peak merit value to a specific value for all currently selected peaks
2
3 - for peak in current.peaks:
4     peak.figureOfMerit = 0.3
5
6

```

5B Using ‘Current’ to loop through selected peaks

- Go to Main Menu → Macro → Open... and open the **5B.currentPeaksSetPeakMerit.py** macro from the **CcpnTutorialDataMacroWritingApril22/macros** directory.

This macro is similar to the one in **Section 5A**, except that rather than looping through all the peaks in one peak list, it loops through all the currently selected peaks.

You can use **current** to loop through many currently selected objects such as peaks, peakLists, nmrAtoms/Residues/Chains, chemicalShifts, chemicalShiftLists, restraints, samples, substances, spectrumGroups etc. (see next page for a full list).

You also have access to some GUI (graphical user interface) elements, including cursorPosition, guiTable and strip.

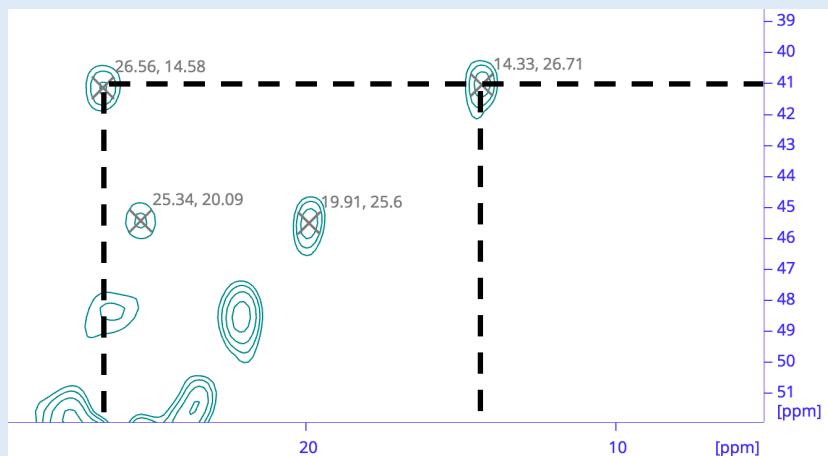
Spectrum is not included, because you can't select a spectrum. Instead, use **current.strip.spectra** as a way to access a selected spectrum or set of spectra.

For most items both the plural and singular exist, i.e. **peaks** and **peak**.

Current.peaks will give you all the currently selected peaks; **current.peak** will only give you the most recently selected peak, even if other peaks have been selected as well.

Objects available in 'Current'

axisCode
chain / chains
chemicalShift / chemicalShifts
chemicalShiftList / chemicalShiftLists
collection / collections
cursorPosition
dataTable / dataTables
guiTable
integral / integrals
macroFile / macroFiles
multiplet / multiplets
nmrAtom / nmrAtoms
nmrChain / nmrChains
nmrResidue / nmrResidues
pcaComponent / pcaComponents (AnalysisScreen and AnalysisMetabolomics)
peak / peaks
pid
position / positions
project
residue / residues
restraint / restraints
sample / samples
spectrumGroup / spectrumGroups
spectrumHit (AnalysisScreen)
strip
substance / substances
violationTable
violationTables

 $x = \sim 26.6$ $x = \sim 14.4$

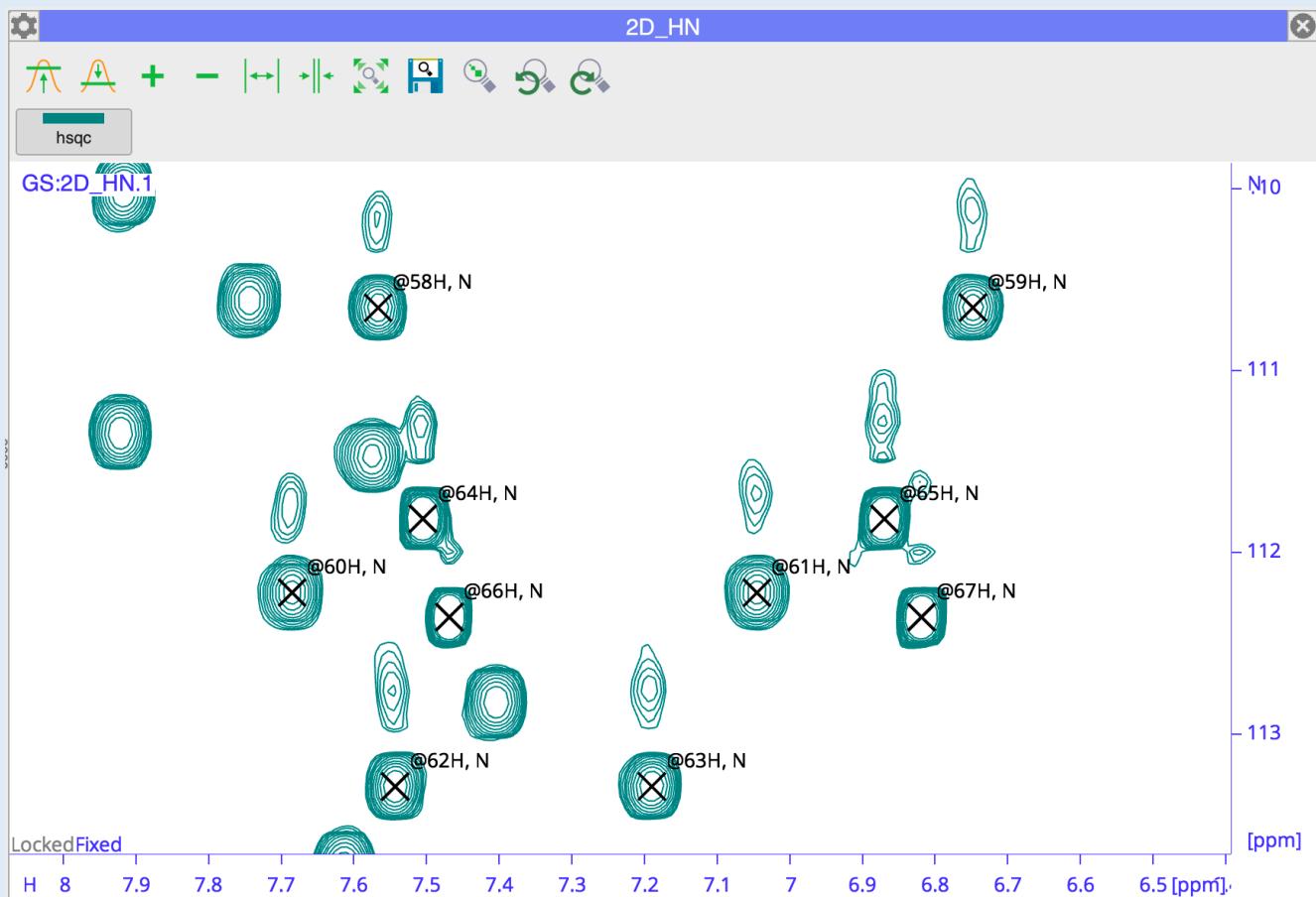
$$y = 14.4 + 26.6 \\ = 41.0$$

5c Excercises

1. Add 'new_' to start of each spectrum name (hint: use f-strings or join() to do the actual addition of the prefix)
2. Add a comment (such as 'noise?') to all currently selected peaks.
3. Deconvolute the chemical shifts in a DQ spectrum: the **SP:yada_postc7** spectrum is a double quantum (DQ) spectrum in which each peak position reflects two chemical shifts (CS1 and CS2) at position $x = CS1$ and $y = CS1 + CS2$ (see above). Pick some peaks in this spectrum (note they come in horizontal pairs!) and then write a macro to deconvolve them and place them in the peak Annotation. At the end you can go to the Spectrum Settings and select **Annotation** to display the chemical shifts.

Suggested answers are in the **macros** directory.

5 Finding Methods and Properties



5D Fetch

1. **SP:hsqc** (or whatever you have renamed it to!) contains a set of peaks which have been set up with **NmrAtoms** using the **Assign/Set up NmrAtoms** feature. This assumes that all atom names will be **H** and **N**. But of course some peaks will belong to Asn/Gln side chains and therefore be **ND2** and **HD21/22** or **NE2** and **HE21/22** atoms.

Write a macro which allows the user to select a pair of Asn/Gln side-chain peaks and then automatically changes the **NmrAtom** names to **ND2/NE2** and **HD/E21** or **HD/E22** and the residue type to **ASN/GLN**.

Our example solution macro includes a method called **fetchNmrAtom**. You will find that many objects have a **fetch** method associated with them. **Fetch** methods will always look to see if the specified object already exists and if it does, it will **get** it. If it doesn't already exist, it will **create** it. It is a very useful method to use when you want to use an object and can't be sure whether it exists already or not. It means you will neither accidentally try and **get** an object that doesn't yet exist or try and **create** an object that does already exist.

```

if x == 1.0:
    do something
elif 1.0 < x <= 10.0:
    do something different
else:
    do something different again

if x != 'this_string':

if peak.lineWidth is None:

if len(project.spectra) >= 1:

if project.spectra:

```

5_E Introducing conditionals

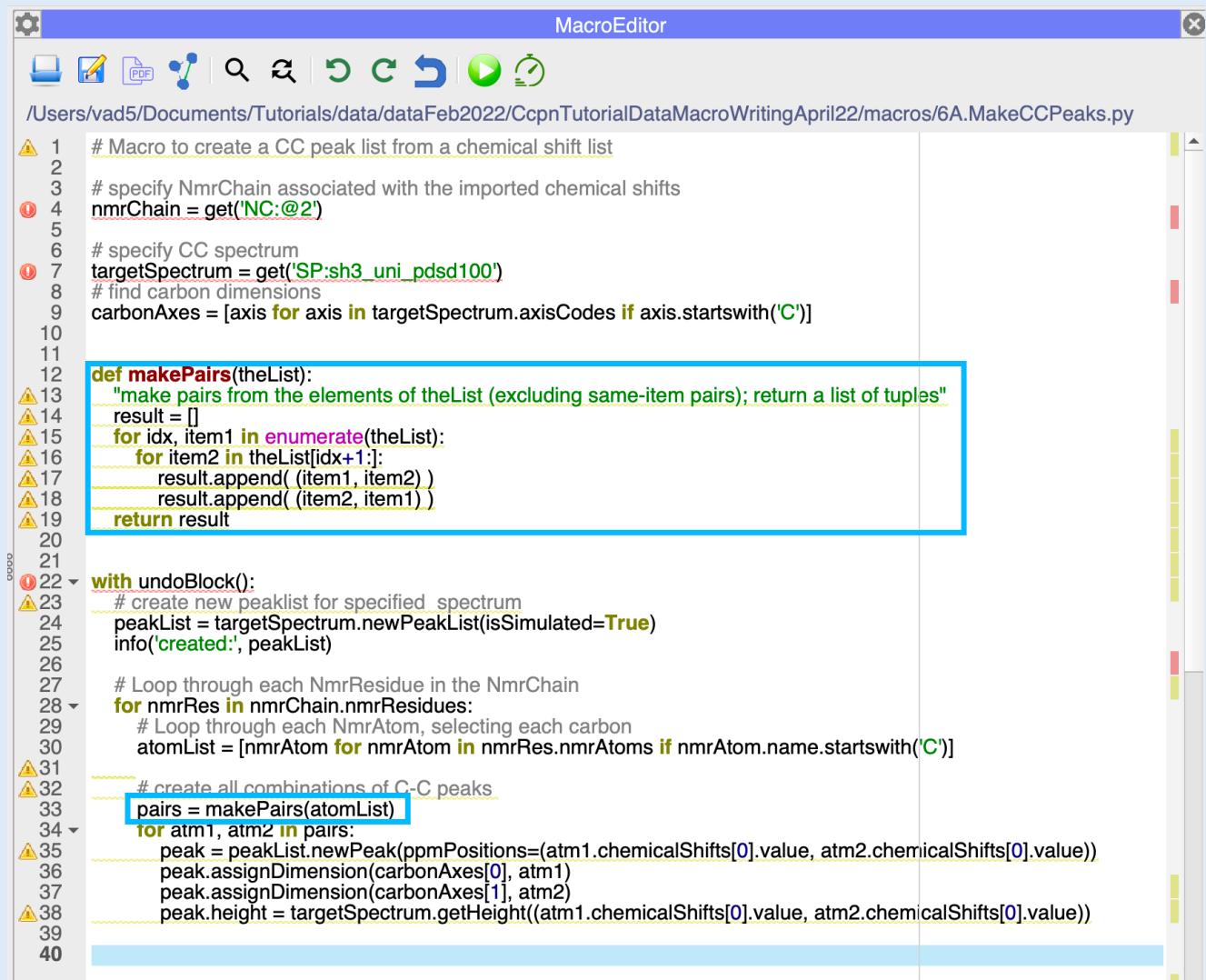
Above is a reminder of some of the conditionals that you can use in python and you might find useful.

Note that **None** means something is undefined. You will sometimes see the word **None** in tables, e.g. the `lineWidth` of a peak could be **None**. Remember that **None** is not the same as **0.0**, although some routines will treat them the same (`if not peak.lineWidth:` will find line widths that are `None` *and* `0.0`). Try to be as precise as you can in your coding!

5_F Excercises

Write macros using loops and conditionals to do the following

1. Change the '`Hn`' `NmrAtom` names in the `NC:GB1 NmrChain` to '`H`'.
2. Delete all the `NmrAtoms` in the `NC:GB1 NmrChain` that do not have any chemical shifts associated with them.
3. Delete all unassigned peaks in the `PL:GB1_hsqc.2` peak list.
4. Delete all diagonal peaks in the `PL:sh3_2c_pdsd50.1` peak list. (You will need to decide what kind of tolerance to work with.)
5. Move currently selected peaks in the `PL:sh3_2c_pdsd50.1` peak list to its sister peak list `PL:sh3_2c_pdsd50.2`.
6. Merge the `-1` `NmrAtoms` in the `NC:A nmrChain` into the `NmrAtom` of the previous residue.



```

MacroEditor

/Users/vad5/Documents/Tutorials/data/dataFeb2022/CcpnTutorialDataMacroWritingApril22/macros/6A.MakeCCPeaks.py

1 # Macro to create a CC peak list from a chemical shift list
2
3 # specify NmrChain associated with the imported chemical shifts
4 nmrChain = get('NC:@2')
5
6 # specify CC spectrum
7 targetSpectrum = get('SP:sh3_uni_pdsd100')
8 # find carbon dimensions
9 carbonAxes = [axis for axis in targetSpectrum.axisCodes if axis.startswith('C')]
10
11
12 def makePairs(theList):
13     "make pairs from the elements of theList (excluding same-item pairs); return a list of tuples"
14     result = []
15     for idx, item1 in enumerate(theList):
16         for item2 in theList[idx+1:]:
17             result.append( (item1, item2) )
18             result.append( (item2, item1) )
19
20     return result
21
22 with undoBlock():
23     # create new peaklist for specified spectrum
24     peakList = targetSpectrum.newPeakList(isSimulated=True)
25     info['created:', peakList]
26
27     # Loop through each NmrResidue in the NmrChain
28     for nmrRes in nmrChain.nmrResidues:
29         # Loop through each NmrAtom, selecting each carbon
30         atomList = [nmrAtom for nmrAtom in nmrRes.nmrAtoms if nmrAtom.name.startswith('C')]
31
32         # create all combinations of C-C peaks
33         pairs = makePairs(atomList)
34         for atm1, atm2 in pairs:
35             peak = peakList.newPeak(ppmPositions=(atm1.chemicalShifts[0].value, atm2.chemicalShifts[0].value))
36             peak.assignDimension(carbonAxes[0], atm1)
37             peak.assignDimension(carbonAxes[1], atm2)
38             peak.height = targetSpectrum.getHeight((atm1.chemicalShifts[0].value, atm2.chemicalShifts[0].value))
39
40

```

6A Subroutine/Function Basics

- Go to Main Menu → Macro → Open... and open the **6A.MakeCCPeaks.py** macro in the **macros** directory.

This macro will create intra-residue peaks for a ^{13}C - ^{13}C correlation spectrum and contains a subroutine called **makePairs** (boxed in image above).

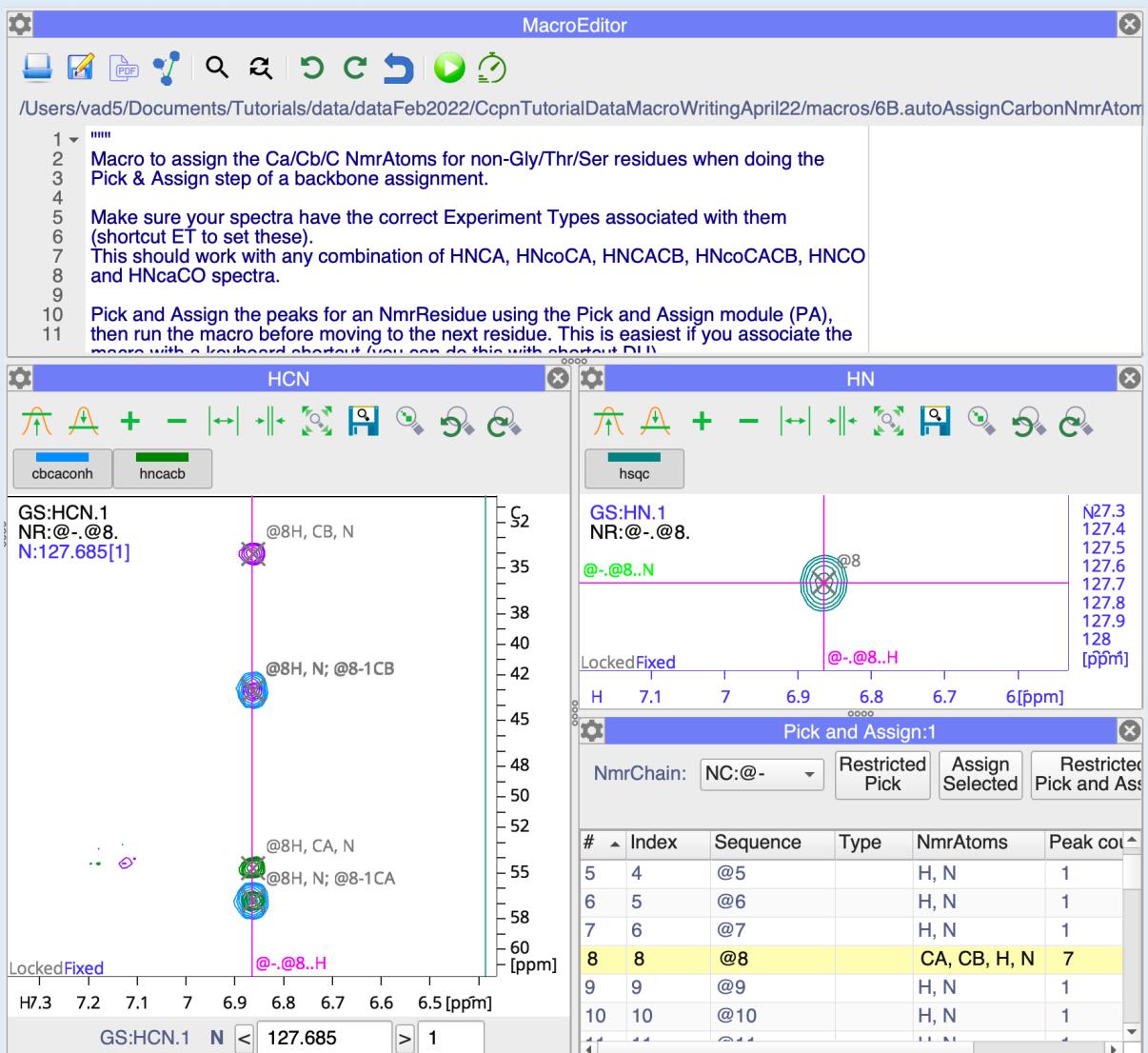
The **def** command (short for **define**) marks the beginning of the function and the remainder is indented.

makePairs is the function name which is used lower down to call it.

theList is (in this case) the only input parameter used by the function and when the function is called, an appropriate parameter must be specified. Use **()** for no input parameters and **(theList, anotherList, theTuple)** for several. **return** marks the end of the function with **result** (in this case a list) being passed back to the main program.

Using subroutines or functions is good programming practice and you should try to use them whenever you can, especially to avoid code duplication.

Subroutines / Functions



6B Subroutine/Function example

This is another example of a macro which uses a function to avoid code duplication.

- Open the **CcpnTutorialDataMacroWritingApril22/Sec5Assignment ccpn** project.
- Go to **Main Menu → Macro → Open User Macro...** and open the **6B.autoAssignCarbonNmrAtoms** macro in the macro directory.
- Go to the **Pick and Assign** settings to set your Target display as **GD:HCN**.
- **Double-click** on an NmrResidue in the **Pick and Assign** table (e.g. @8). If the residue pattern does not include Gly/Thr/Ser, then the macro will automatically assign the Carbon NmrAtoms for you. Run the macro in the editor or assign it to a keyboard shortcut.

Worked Example

```

1  """
2 This Macro illustrates how to work with spectral data and spectral parameters.
3 It creates a series of nD spectra from a pseudo-nD spectrum, collated in a new
4 spectrumGroup.
5
6 """
7
8 # The pseudo3D spectrum
9 spectrum = get('SP:Bruker-pseudo3D-1')
10
11 def splitPseudo3DSpectrumIntoPlanes(spectrum):
12     # we first establish what the "Time" dimension is
13     # getByAxisCodes returns a list, so we take first element
14     timeDimension = spectrum.getByAxisCodes('dimensions', ['Time'])[0]
15
16     # Find the number of points along the Time dimension
17     # dim's and positions are 1-based; python lists/tuples are zero-based
18     timeDimensionSize = spectrum.pointCounts[timeDimension-1]
19
20     # get the frequency dimensions; ie. all dimensions that are not Time dimension(s)
21     # Python's list comprehensions are a powerful way of doing this
22     freqDims = [dim for dim in spectrum.dimensions if dim != timeDimension]
23     # and the corresponding axis codes using build-in functionality of the Spectrum class
24     freqAxisCodes = spectrum.getByDimensions('axisCodes', freqDims)
25
26     # Anything could define the series values of a spectrumGroup
27     # Assume that the series values are defined by the spectralWidth/dwellTime
28     # This is likely incorrect, but can be corrected later if need be.
29     seriesValue = 0.0
30     seriesIncrement = 1.0 / spectrum.spectralWidths[timeDimension-1]
31     seriesUnits = 'ms'
32
33     # For now: we need this to silence some of the output
34     with notificationEchoBlocking():
35         # For now: we want all of this to be one "undo" operation
36         with undoBlock():
37
38             # Create a new spectrumGroup to collate the newly created spectra
39             spectrumGroup = project.newSpectrumGroup(name=spectrum.name, seriesUnits=seriesUnits)
40
41             position = [1]*spectrum.dimensionCount # dims and positions are 1-based
42
43             # loop over the points in the time dimension
44             for timePoint in range(timeDimensionSize):
45
46                 # set the position of the plane we are now doing
47                 # dims and positions are 1-based; Python is zero-based
48                 position[timeDimension-1] = timePoint + 1
49
50                 info(f'==> extracting {freqAxisCodes} plane at {position}')
51
52                 sp = spectrum.extractPlaneToFile(axisCodes = freqAxisCodes, position = position)
53                 spectrumGroup.addSpectrum(sp, seriesValue)
54
55                 seriesValue += seriesIncrement
56
57
58
59     # Check if we have a valid spectrum for the macro to function
60     if not 'Time' in spectrum.dimensionTypes:
61         warning('This macro only works for pseudo-nD spectra; no time axis found.')
62     else:
63         splitPseudo3DSpectrumIntoPlanes(spectrum)
64
65

```

most of macro is contained in this function

find a dimension from its AxisCode

silence terminal messages

wrap a set of operations into a single Undo operation

output an information message to the terminal

write out the NMR data

output a warning to the terminal

The actual macro consists of just these few lines at the bottom which include a test to see if the input data was appropriate or not

7A Pseudo 3D to SpectrumGroup

The **7A.pseudo3dToSpectrumGroup.py** macro will take a Pseudo 3D spectrum and split it up into individual planes which are written out as **.ndf5** spectra and placed into a new SpectrumGroup. This macro illustrates some useful methods including how to extract raw data from a spectrum and write it into a new file.

- Try it out using the **Bruker-pseudo3D.zip** spectrum in the **spectra** directory.

External modules included in the V3.1 environment

Advanced users who are familiar with these packages, can include these in their macros if they wish. We've highlighted some of the most popular ones.

alabaster 0.7.12	jbig 2.1
appnope 0.1.2	jdcal 1.4.1
asn1crypto 1.4.0	jedi 0.17.2
asttokens 2.0.5	jinja2 3.0.2
babel 2.9.1	joblib 1.1.0
backcall 0.2.0	jpeg 9d
backports 1.1	jupyter_client 7.1.0
backports.shutil_get_terminal_size 1.0.0	jupyter_core 4.9.1
biopython 1.78	kiwisolver 1.3.1
blas 1.0	krb5 1.19.2
bottleneck 1.3.2	lcms2 2.12
brotli 1.0.9	libclang 11.1.0
brotlipy 0.7.0	libcurl 7.80.0
bzip2 1.0.8	libcxx 12.0.0
c-ares 1.18.1	libcxxabi 4.0.1
ca-certificates 2021.10.26	libedit 3.1.20210910
cairo 1.16.0	libev 4.33
certifi 2021.10.8	libffi 3.3
cffi 1.15.0	libgfortran 3.0.1
chardet 4.0.0	libglib 2.68.2
charset-normalizer 2.0.4	libiconv 1.16
colorama 0.4.4	libllvm11 11.1.0
cryptography 36.0.0	libnnghttp2 1.46.0
curl 7.80.0	libopenblas 0.3.13
cycler 0.11.0	libpng 1.6.37
cython 0.29.25	libpq 13.3
dbus 1.13.18	libsodium 1.0.18
debugpy 1.5.1	libssh2 1.9.0
decorator 5.1.0	libtiff 4.2.0
docutils 0.17.1	libwebp 1.2.0
entrypoints 0.3	libwebp-base 1.2.0
et_xmlfile 1.1.0	libxml2 2.9.12
executing 0.8.2	libxslt 1.1.33
expat 2.4.1	llvm-openmp 12.0.0
fontconfig 2.13.1	llvmlite 0.37.0
fonttools 4.25.0	lz4-c 1.9.3
freetype 2.11.0	markupsafe 2.0.1
fribidi 1.0.10	matplotlib 3.5.0
get_terminal_size 1.0.0	matplotlib-base 3.5.0
gettext 0.21.0	matplotlib-inline 0.1.2
giflib 5.2.1	memory_profiler 0.58.0
git 2.34.1	mkl 2022.0.0
glib 2.68.2	munkres 1.1.4
graphite2 1.3.14	mysql-common 8.0.25
h5py 3.5.0	mysql-libs 8.0.25
harfbuzz 2.8.1	ncurses 6.3
hdf5 1.10.6	nest-asyncio 1.5.1
icecream 2.1.1	nomkl 3.0
icu 68.1	nose 1.3.7
idna 3.3	nspr 4.32
imageio 2.9.0	nss 3.69
imagesize 1.3.0	numba 0.54.1
intel-openmp 2022.0.0	numexpr 2.8.1
ipykernel 6.4.1	numpy 1.20.3
ipython 7.29.0	numpy-base 1.20.3
ipython_genutils 0.2.0	

External modules included in the V3.1 environment

continued

olefile 0.46	scipy 1.7.3
openbabel 3.1.1	seaborn 0.11.2
openblas 0.3.13	setuptools 58.0.4
openblas-devel 0.3.13	simplegeneric 0.8.1
openpyxl 3.0.9	sip 4.19.25
openssl 1.1.1m	six 1.16.0
packaging 21.3	snowballstemmer 2.2.0
pandas 1.3.5	sphinx 4.2.0
pango 1.45.3	sphinx_rtd_theme 0.4.3
parso 0.7.0	sphinxcontrib-applehelp 1.0.2
path 16.2.0	sphinxcontrib-devhelp 1.0.2
path.py 12.5.0	sphinxcontrib-htmlhelp 2.0.0
pcre 8.45	sphinxcontrib-jsmath 1.0.1
pcre2 10.37	sphinxcontrib-qthelp 1.0.3
perl 5.26.2	sphinxcontrib-serializinghtml 1.1.5
pexpect 4.8.0	sqlite 3.37.0
pickleshare 0.7.5	tabulate 0.8.9
pillow 8.4.0	tbb 2021.5.0
pip 21.2.4	threadpoolctl 2.2.0
pixman 0.40.0	tk 8.6.11
prompt-toolkit 3.0.20	tornado 6.1
prompt_toolkit 3.0.20	tqdm 4.62.3
psutil 5.8.0	traitlets 5.1.1
ptyprocess 0.7.0	urllib3 1.26.7
pycparser 2.21	wcwidth 0.2.5
pygments 2.10.0	wheel 0.37.1
pyopenssl 21.0.0	xlrd 2.0.1
pyparsing 3.0.4	xlsxwriter 3.0.2
pyqt 5.12.3	xz 5.2.5
pyqt-impl 5.12.3	yaml 0.2.5
pyqt5-sip 4.19.18	zeromq 4.3.4
pyqtchart 5.12	zlib 1.2.11
pyqtgraph 0.11.0	zstd 1.4.9
pyqtwebengine 5.12.1	asteval 0.9.26
pysocks 1.7.1	docx 0.2.4
python 3.8.10	future 0.18.2
python-dateutil 2.8.2	hjson 3.0.2
python-pptx 0.6.21	lmfit 1.0.3
python_abi 3.8	lxml 4.7.1
pytz 2021.3	nmrglue 0.8
pyyaml 6.0	pycodestyle 2.8.0
pyzmq 22.3.0	pyflakes 2.4.0
qt 5.12.9	pynmrstar 3.3.0
qtconsole 5.1.1	pyopengl 3.1.5
qtpy 1.10.0	pyopengl-accelerate 3.1.5
readline 8.1.2	pyqode-core 2.12.1
reportlab 3.5.67	pyqode-python 2.12.1
requests 2.27.1	pyqode-qt 2.10.0
scikit-learn 1.0.2	uncertainties 3.1.6
	pyqode-qt

Advanced Python users can go on to include GUI elements, including messages, selection boxes or plots. Please ask the CCPN team for more information or examples of this.

Contact Us

Website:

www ccpn.ac.uk

Suggestions and comments:

support@ccpn.ac.uk

Issues and bug reports:

<https://forum.ccpn.ac.uk/>

Cite Us

Skinner, S. P. et al. CcpNmr AnalysisAssign: a flexible platform for integrated NMR analysis. *J. Biomol. NMR* 66, (2016)