Python Language Basics

This notebook introduces some Python language basics. It is designed for a code-along during the workshop. Some code blocks are deliberately incomplete or have bugs included.

To execute code: Place your cursor in the cell, then hit SHIFT + ENTER

When writing computer code, there are *rules* and there are *conventions* (adapted from David Dempsey's Python for Geoscientists course)

- If you break a ***rule***, the code will not work.
- If you break a ***convention***, someone, somewhere puts a mark against your name in a book. At the end of times, there will be a final accounting.

Some Python rules:

- ***Syntax*** - we have very precise expectations about how you write computer code. If you type an opening a bracket, you must close it again later. Some lines must be terminated by a colon, - if you omit this, the code will not work. ***Learning syntax for a new code is very pedantic and a total pain.*** But you have to do it anyway, and at least Python returns readable error messages to help you understand your missteps...
- ***Indentation*** - Python reads indentation (not all languages do) and your code will not work if it is not correctly indented.

Some Python conventions:

- ***Commenting*** - it is helpful for the poor soul who has to read your poorly written Python (sometimes that is you, weeks or months later) if you have included little 'sign-posts' in the code, articulating what you are doing. These are called comments. They begin with a # symbol, after which you can write whatever you like and it will not be executed as a command.
- ***Sensible variable names*** - a word or words relating to the thing the variable represents but not too long. For example, if a variable contains the mean temperature, then Tmean is a sensible variable name, where as the_mean_temperature_of_the_profile or a1 are not sensible variable names.
- ***Layout*** - Although the computer reads Python line-by-line, using a vertical layout improves the human readability and your ability to comment code. Headings and the liberal use of whitespace improves readbility.
- ***Structure*** - Python files and projects have a standard structure that improves redabality. The structure of files vary, but all files should start with an explanation of what they do and all packages/modules should be imported at the top. For the advanced folks, more on Python project structure can be found here.

Units are a common source of error. Approches to managing this include:

- Include units in the variable names (my preferred) `Tmean_degC = 300`
- Include in-line comments on units and conversions used (good, but can be lost information) `Tmean = 300 # mean temp in degC`
- Consider tools like Pint to manage units, if it is a large problem (good, but assumes everyone knows how to use pint)

Python Packages

In [ ]:
```
# Import packages

''' Good practice:
Import all libraries/packages used at the top of the document'''

import matplotlib.pyplot as plt # foundation module for plotting
import numpy as np # for array based programming and advanced math
import math # for basic mathematical methods
import pandas as pd # for spreadsheet-like methods + more
import matplotlib.pyplot as plt # foundation module for plotting
import seaborn as sns # for advanced plotting methods
import matplotlib.dates as mdates # for formatting datetime in plot axis

''' Good practice:
Packages have standard short names.
Using the standard name improves readability.
Look at the docs if you are unsure what the standard name is.
For example, https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html) ''';
```

In [ ]:
```
import antigravity

# installing a package can launch code
# be careful what you install!
```

Python Language Basics

In [ ]:
```
print('hello world')
```

```
hello world
```

In [ ]:
```
# variable
# the string 'hello world' has been assigned to the variable x

x = 'hello world'

print(x)
print(len(x))
```

```
hello world
11
```

In [ ]:
```
# Types: string, integer, float (more on slide)
# Python is a dynamic typed language, checking type (variable state)

print(type(x))
```

```
<class 'str'>
```

Data structures and the many forms of list

https://docs.python.org/3/tutorial/datastructures.html#

In [ ]:
```
# Lists and array (numpy)

my_list = [1,2,3,4]
print(my_list)
print(type(my_list))

my_array = np.array([1,2,3,4])
print(my_array)
print(type(my_array))
```

```
[1, 2, 3, 4]
<class 'list'>
[1 2 3 4]
<class 'numpy.ndarray'>
```

In [ ]:
```
array_of_10s = np.array([10,10,10,10])
array_of_5s = np.array([5,5,5,5])

new_array = array_of_5s + array_of_10s

print(new_array)

# array-based programming
```

```
[15 15 15 15]
```

In [ ]:
```
list_of_10s = [10,10,10,10]
list_of_5s = [5,5,5,5]

new_list = list_of_5s + list_of_10s

print(new_list)
```

```
[5, 5, 5, 5, 10, 10, 10, 10]
```

In [ ]:
```
# Basics: Complex lists and arrays

complex_list = ['harry', 1, 9.5, [2,3,4]]
print(complex_list)

complex_array = np.array(['harry', 1, 9.5,[2,3,4]],dtype=object)
print(complex_array, complex_array.shape)

nD_array = np.array([[4,5,6],[1,2,3]])
```

```
print(nD_array, nD_array.shape)

# shape tells us the number of elements in each dimension
```

```
['harry', 1, 9.5, [2, 3, 4]]
['harry' 1 9.5 list([2, 3, 4])] (4,)
[[4 5 6]
 [1 2 3]] (2, 3)
```

In [ ]:
```
# tuple - a fixed collection of objects or variables

my_tuple = (3, 5, 7)

my_object = [9.1,9.2,9.3]

complex_tuple = (3, 5, 7, ['d','wow',3.2], 'fish', my_object)

print(type(complex_tuple))

# Lists vs tuples
# Tuples are more memory efficient than lists
# Generally, people expect lists to contain one type while it is more acceptable to
```

```
<class 'tuple'>
```

In [ ]:
```
# indexing (lists, tuples and arrays)

print(complex_tuple[1])
print(complex_tuple[3][1])

print(complex_tuple[5][-1]) # get 9.3 out of complex_tuple
print(complex_tuple[5][2]) # get 9.3 out of complex_tuple

# note how python starts counting at zero, so the first element = 0
```

```
5
wow
9.3
9.3
```

In [ ]:
```
# dictionary - a value is mapped to a key
colors = {
    'blue': '#36648B', # nice low-saturation blue
    'brown': '#8B5D36', # complementary color to the blue
}

print(colors)

print(type(colors))

print(colors['blue'])

colors['blue'] = 'another blue'
print(colors['blue'])
```

```
{'blue': '#36648B', 'brown': '#8B5D36'}
<class 'dict'>
#36648B
another blue
```

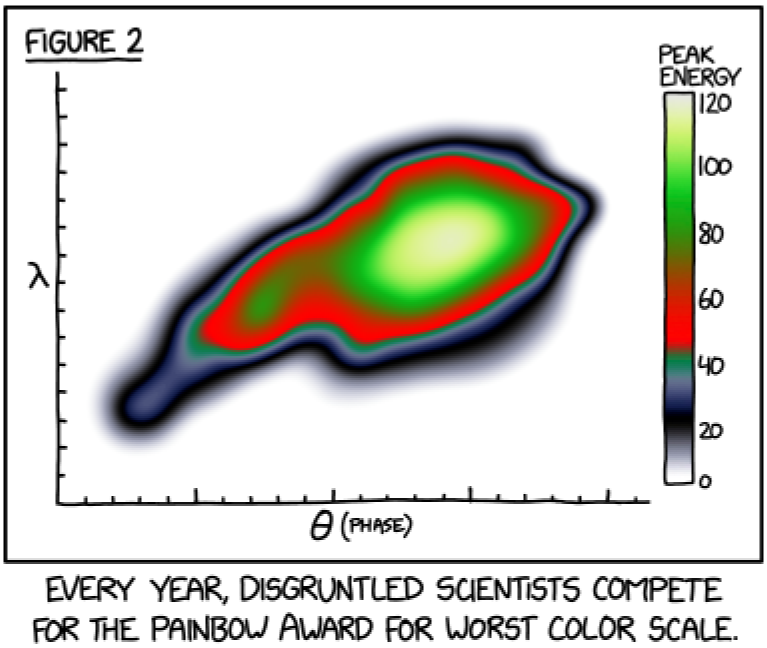
Colors in Python:

- Hex colors https://www.color-hex.com/
- Named colors https://matplotlib.org/stable/gallery/color/named_colors.html
- Colormaps https://matplotlib.org/stable/tutorials/colors/colormaps.html

Beware the impact of color on your analysis!
https://hess.copernicus.org/articles/25/4549/2021/

In [ ]:
```
from IPython import display
display.Image("https://imgs.xkcd.com/comics/painbow_award.png")
```

Out[ ]:



Python as Calculator

In [ ]:
```
radius_m = 3
area_m2 = math.pi * radius_m**2

print(area_m2)
```

```
28.274333882308138
```

Estimating well injectivity using field operational data:

$$II = \frac{Q}{P_H + WHP - P_F - P_{FZ}}$$

where Q Flow rate (t/hr), P_H is hydrostatic pressure inside the well (bara), WHP is wellhead pressure (bara), P_F is the pressure of frictional losses (bara), and P_{FZ} is the reservoir pressure at the feedzone or pivot point (bara).

(Siega et al. (2014) Quantifying the effect of temperature on well injectivity, New Zealand Geothermal Workshop)

Python uses Latex for formatting mathematical expressions https://www.overleaf.com/learn/latex/Mathematical_expressions

In [ ]:
```
# Variable names:
# Use common abbreviations or write it out
# Do not just use equation terms (Q, Ph, WHP) unless they are universal and defined

flow_rate_tph = 400
hydrostatic_pressure_bara = 115
wellhead_pressure_bara = 1
friction_loss_bara = 9.7
feedzone_pressure_bara = 84

ii_tphrpbar = flow_rate_tph / (hydrostatic_pressure_bara + wellhead_pressure_bara -

print(ii_tphrpbar)
```

```
17.937219730941706
```

In [ ]:
```
# Using for loops to calculate value for multiple values - iterate over one list

flow_rates_tph = [400, 300, 200, 100]

varied_rate_ii_tphrpbar = []

for rate in flow_rates_tph:
    ii = rate / (hydrostatic_pressure_bara + wellhead_pressure_bara - friction_loss
    varied_rate_ii_tphrpbar.append(ii)

print(varied_rate_ii_tphrpbar)
```

```
[17.937219730941706, 13.45291479820628, 8.968609865470853, 4.4843049327354265]
```

Algorithm building: Logical operators and flow control

Tests that return True or False

Boolean operators

- And
- Or
- Not

Comparison operators

- == Equal to
- != Not equal to
- < Less than
- > Greater Than
- <= Less than or Equal to
- >= Greater than or Equal to

"if" statements

- if
- elif
- else

'for' loop

- pass in a list or array that is iterated over

Refer to this link for other options and various examples.

In [ ]:
```
# Basics: "for loop" and whitespace
for n in [1,2,3,4,5]:
    print(n + 10)
```

```
11
12
13
14
15
```

In [ ]:
```
# Using for loops to calculate value for multiple values - iterate over many lists

flow_rates_tph = [400, 300, 200, 100]
wellhead_pressures_bara = [10, 8, 5, 1]

varied_rate_and_whp_ii_tphrpbar = []

for rate, whp in zip(flow_rates_tph, wellhead_pressures_bara):
    ii = rate / (hydrostatic_pressure_bara + whp - friction_loss_bara - feedzone_pr
    varied_rate_and_whp_ii_tphrpbar.append(ii)

print(varied_rate_and_whp_ii_tphrpbar)
```

```
[12.779552715654953, 10.238907849829353, 7.604562737642587, 4.4843049327354265]
```

In [ ]:
```
# Using if statements to dictate behavior

flow_rates_tph = [400, 300, 200, 100]
wellhead_pressures_bara = [10, 8, 5, 1]

for rate, whp in zip(flow_rates_tph, wellhead_pressures_bara):
    ii = rate / (hydrostatic_pressure_bara + whp - friction_loss_bara - feedzone_pr
    if ii > 10:
```

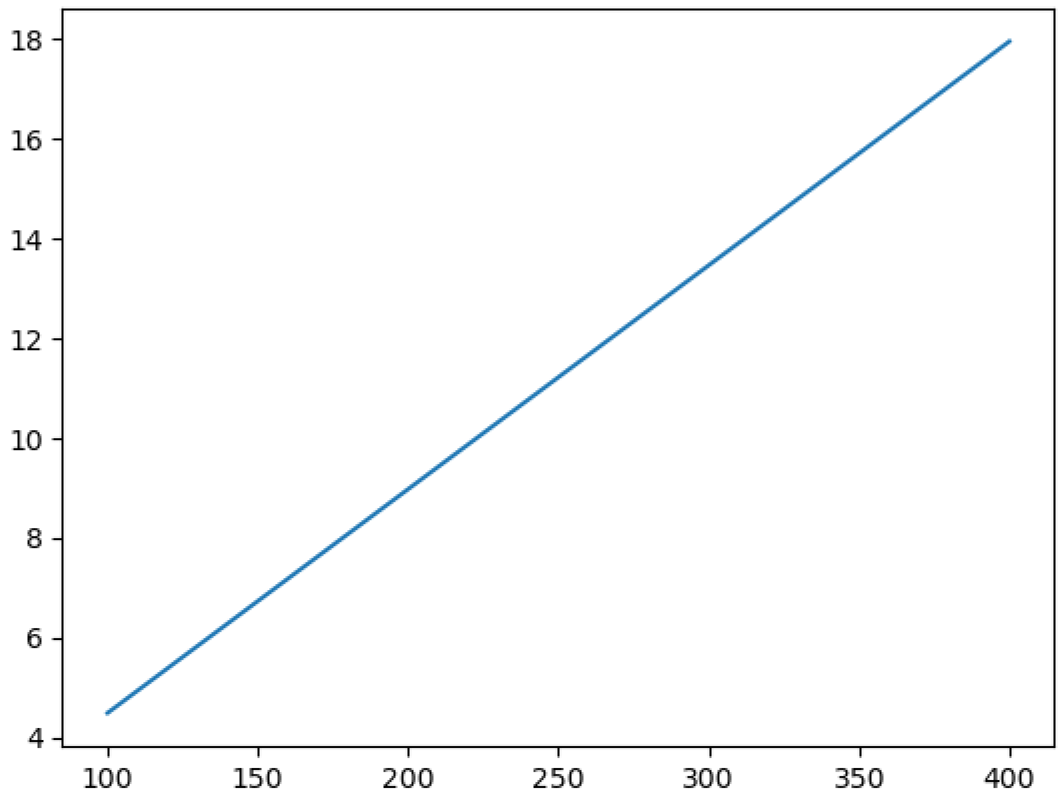
```
        print('Great well!')
    else:
        print(':-(')
```

```
Great well!
Great well!
:-(
:-(
```

Plotting

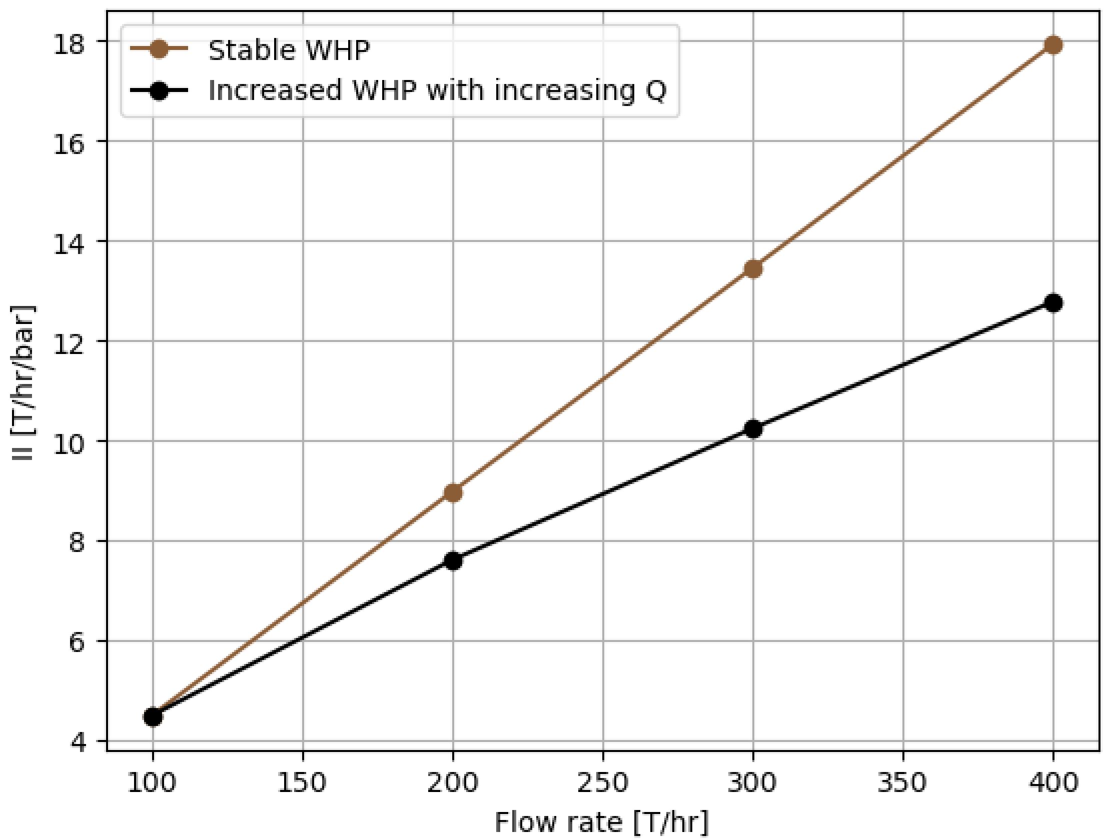
In [ ]:
```
# Simple

plt.plot(flow_rates_tph, varied_rate_ii_tphrpbar)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x1d7ea4d5760>]
```



In [ ]:
```
# More flexibility: the axs method
# when googling, use "matplotlib axs" to get the correct documentation

fig, ax = plt.subplots(1,1)

# the method plt.subplots() returns two things
# one we have called figure (f or fig) and this is analogous to a page in your book
# the other is the axis (one or many ax) which is analogous to the plot you draw on

ax.plot(
    flow_rates_tph,
    varied_rate_ii_tphrpbar,
    color = colors['brown'],
```

```
    marker = 'o',
    label = 'Stable WHP'
    )

ax.plot(
    flow_rates_tph,
    varied_rate_and_whp_ii_tphrpbar,
    color = 'k',
    marker = 'o',
    label = 'Increased WHP with increasing Q'
    )

ax.set_ylabel('II [T/hr/bar]')
ax.set_xlabel('Flow rate [T/hr]')
ax.legend()
ax.grid()

# https://github.com/ICWallis/tutorial-publication-ready-figures
```



Functions

In [ ]:
```
# Functions = re-useable tools

def circle_area(radius):
    '''Calculate the area of a circle from the radius'''

    area = math.pi * radius**2
```

```
    return area

a = circle_area(5)
print(a)
```

78.53981633974483

In [ ]:

```
#
# Function
#

def ii_from_operational_conditions(Q,Ph,WHP,Pf,Pfz):
    '''Calculate injectivity using field operational data

    For usage and case study, refer to Siega et al. (2014)
    Quantifying the effect of temperature on well injectivity,
    New Zealand Geothermal Workshop

        Args:
            Q (float): flow rate - t/hr
            Ph (float): hydrostatic pressure inside the well - bara
            WHP (float): wellhead pressure - bara
            Pf (float): pressure due to friction - bara
            Pfz (float): reservoir pressure at the feedzone or pivot point - bara

        Returns:
            injectivity - T/hr/bar
    '''

    ii = Q / (Ph + WHP - Pf - Pfz)

    return ii

#
# Usage
#

# Input parameters for well WJ-13
Q_tph = 400 # average for June
Ph_bara = 115 # calculated using depth to feedzone and average injection temperatur
WHP_bara = 1 # average for June
Pf_bara = 9.7 # calculated by finding the root of the Colebrook equation
Pfz_bara = 84 # from the reservoir pressure correlation

WJ13_June_ii_tphrpbar = ii_from_operational_conditions(Q_tph, Ph_bara, WHP_bara, Pf

print(round(WJ13_June_ii_tphrpbar,2))


# Functions can be placed in a file that is imported at the start
# This reduces the code you are looking at
# and enables you to focus on the science, rather than the code
```

17.94

In [ ]:

```
df = pd.read_excel(r'1_Introduction__Case_study_data.xlsx', sheet_name='Sheet1')
```

```
df
```

Out[ ]:

	Well	Date	Q_tph	Ph_bara	WHP_bara	Pf_bara	Pfz_bara
0	W-1	2022-01-01	400	115	1.0	9.70	84
1	W-1	2022-02-01	400	115	1.5	9.70	84
2	W-1	2022-03-01	400	115	2.0	9.70	84
3	W-1	2022-04-01	400	115	2.8	9.70	84
4	W-2	2022-01-01	200	100	8.0	9.68	70
5	W-2	2022-02-01	200	100	5.0	9.68	70
6	W-2	2022-03-01	200	100	2.0	9.68	70
7	W-2	2022-04-01	200	100	1.0	9.68	70

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Well      8 non-null      object
 1   Date      8 non-null      datetime64[ns]
 2   Q_tph     8 non-null      int64
 3   Ph_bara   8 non-null      int64
 4   WHP_bara  8 non-null      float64
 5   Pf_bara   8 non-null      float64
 6   Pfz_bara  8 non-null      int64
dtypes: datetime64[ns](1), float64(2), int64(3), object(1)
memory usage: 576.0+ bytes
```

In [ ]:

```
df['ii_tphrpbar'] = ii_from_operational_conditions(df.Q_tph, df.Ph_bara, df.WHP_bar
df
```

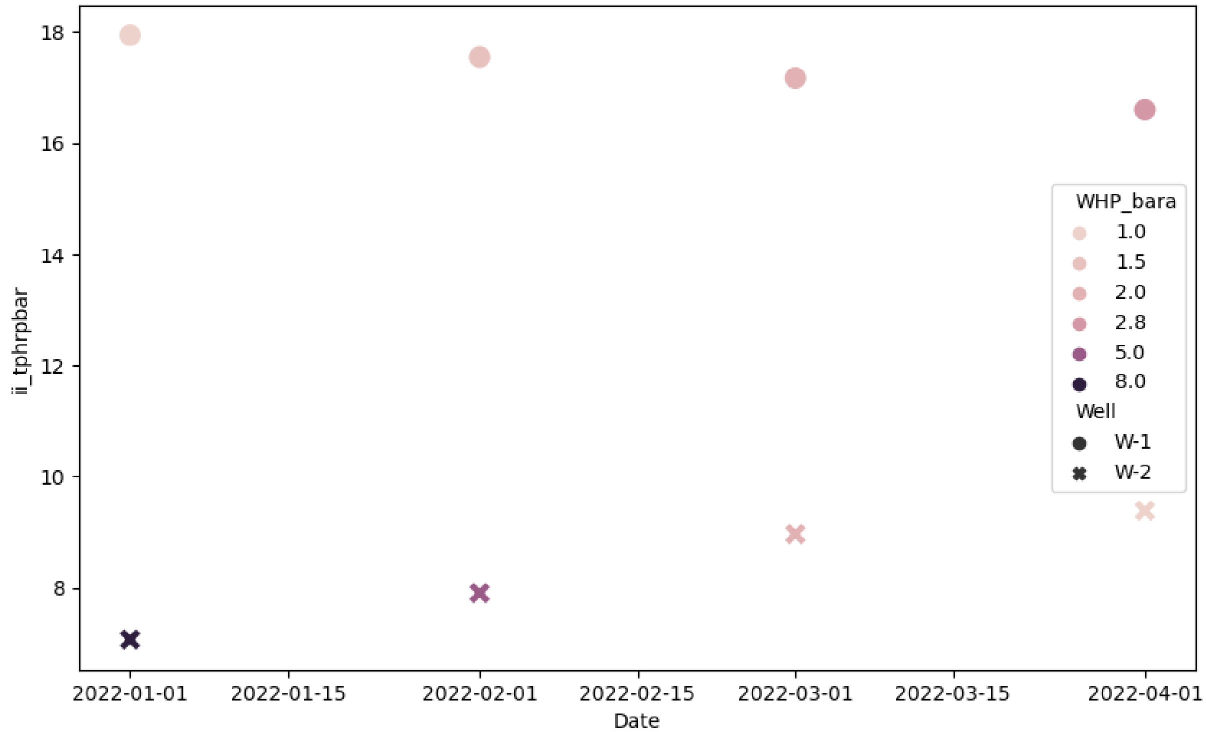
Out[ ]:

	Well	Date	Q_tph	Ph_bara	WHP_bara	Pf_bara	Pfz_bara	ii_tphrpbar
0	W-1	2022-01-01	400	115	1.0	9.70	84	17.937220
1	W-1	2022-02-01	400	115	1.5	9.70	84	17.543860
2	W-1	2022-03-01	400	115	2.0	9.70	84	17.167382
3	W-1	2022-04-01	400	115	2.8	9.70	84	16.597510
4	W-2	2022-01-01	200	100	8.0	9.68	70	7.062147
5	W-2	2022-02-01	200	100	5.0	9.68	70	7.898894
6	W-2	2022-03-01	200	100	2.0	9.68	70	8.960573
7	W-2	2022-04-01	200	100	1.0	9.68	70	9.380863

In [ ]:
```
# Advanced plot for data analysis

fig, ax = plt.subplots(1,1,figsize=(10,6))

# https://seaborn.pydata.org/generated/seaborn.scatterplot.html

sns.scatterplot(
    x='Date', # Pandas column name for x data
    y='ii_tphrpbar', # Pandas column name for y data
    data=df, # Pandas dataframe name
    ax=ax, # name of the axis that the seaborn plot goes in
    s=140, # marker size
    hue='WHP_bara', # marker colour
    style='Well', # marker style
)

#plt.savefig('Case_study_plot1.png',facecolor='w')
```

```
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

Out[ ]:
```
<Axes: xlabel='Date', ylabel='ii_tphrpbar'>
```

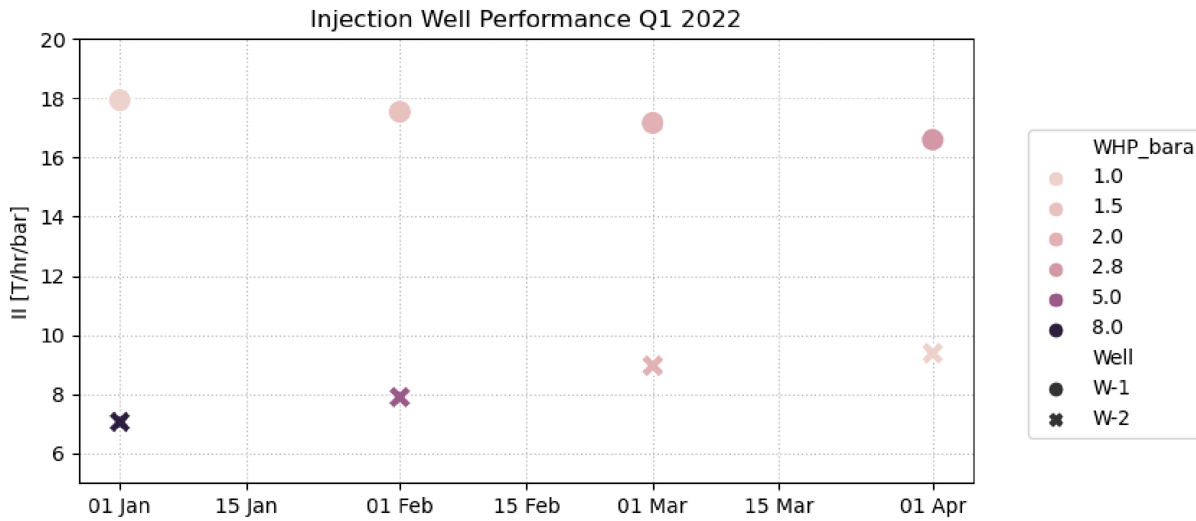


In [ ]:
```
# Building on the above code to make a nice version for a report/publication

# Make empty plot
fig, ax = plt.subplots(1,1,figsize=(8,4))

# Add data to plot
sns.scatterplot(
    x='Date', # Pandas column name for x data
    y='ii_tphrpbar', # Pandas column name for y data
    data=df, # Pandas dataframe name
    ax=ax, # name of the axis that the seaborn plot goes in
    s=140, # marker size
    style='Well', # marker style
    hue='WHP_bara', # marker colour
    #palette='cividis',
    #size='Q_tph',
    #sizes=(140, 200)
    zorder=10 # places markers in front of grid
)

# Seaborn color pallets https://seaborn.pydata.org/tutorial/color_palettes.html

# Format axis
ax.set_ylim(5,20)

ax.xaxis.set_major_formatter(mdates.DateFormatter("%d %b"))

ax.set_xlabel('') # Replaces auto date label with nothing

ax.set_ylabel('II [T/hr/bar]')

# Place legend outside the plot area
ax.legend(
```

```
    loc='center left',
    bbox_to_anchor=(1.05, 0.45),
    ncol=1,
)

# Add title
plt.title('Injection Well Performance Q1 2022')

# Add grid
ax.grid(linestyle=':', alpha=0.8)

# Export figure
'''
plt.savefig(
    'Case_study_plot2.png',
    dpi=400,
    facecolor='w',
    bbox_inches='tight'
    )'''
```

```
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
c:\Users\Irene\miniconda3\envs\workshop_env\lib\site-packages\seaborn\_oldcore.py:14
98: FutureWarning: is_categorical_dtype is deprecated and will be removed in a futur
e version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

Out[ ]:
```
"\nplt.savefig(\n    'Case_study_plot2.png',\n    dpi=400,\n    facecolor='w',\n
bbox_inches='tight'\n    )"
```



Sub-select data using comparison operator

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Well', 'Date', 'Q_tph', 'Ph_bara', 'WHP_bara', 'Pf_bara', 'Pfz_bara',
       'ii_tphrpbar'],
      dtype='object')
```

In [ ]:
```
df
```

Out[ ]:

	Well	Date	Q_tph	Ph_bara	WHP_bara	Pf_bara	Pfz_bara	ii_tphrpbar
0	W-1	2022-01-01	400	115	1.0	9.70	84	17.937220
1	W-1	2022-02-01	400	115	1.5	9.70	84	17.543860
2	W-1	2022-03-01	400	115	2.0	9.70	84	17.167382
3	W-1	2022-04-01	400	115	2.8	9.70	84	16.597510
4	W-2	2022-01-01	200	100	8.0	9.68	70	7.062147
5	W-2	2022-02-01	200	100	5.0	9.68	70	7.898894
6	W-2	2022-03-01	200	100	2.0	9.68	70	8.960573
7	W-2	2022-04-01	200	100	1.0	9.68	70	9.380863

In [ ]:
```
# using reassignment - take care of notebook order!
df = df[df.Well == 'W-1']
```

In [ ]:
```
df
```

Out[ ]:

	Well	Date	Q_tph	Ph_bara	WHP_bara	Pf_bara	Pfz_bara	ii_tphrpbar
0	W-1	2022-01-01	400	115	1.0	9.7	84	17.937220
1	W-1	2022-02-01	400	115	1.5	9.7	84	17.543860
2	W-1	2022-03-01	400	115	2.0	9.7	84	17.167382
3	W-1	2022-04-01	400	115	2.8	9.7	84	16.597510

Where to from here?

The best way forward is to learn by doing!

I most commonly use python to...

Document methods and analysis in Jupyter notebooks:

- Methods are displayed in sequence
- Can combine notes, equations, views on the data and plots in an easily read format
- Can be exported to html so non-coding folks can read them

Automate tasks:

- Especially useful for repetitive data processing tasks
- Automate the Boring Stuff is a great book for getting started with Python and automation

Handle large datasets, do complex calculations, make interactive graphics, and more!

Learn more Python skills using on-line resources:

- A list of useful resources for geoscientists learning python has been put together by Michael Harty
- Practical Python great course if you can survive stock tickers. Self driven and comprehensive.
- David Dempsey's Python for Geoscientists specific to geosciences. Designed to be classroom taught, but chock-full of useful examples.
- Plotting with Python for basic to advanced plotting with matplotlib. Designed to be classroom taught, but see solutions and recipes folders for code examples.
- Software carpentry a wide range of learning resources
- EdX course

Find your community and join Software Underground to connect with people who do computers + subsurface

- Active slack community where people help each other (note that they are currently moving to a new platform because of $$)
- Annual virtual conference with lots of free training opportunities
- SWUG YouTube channel, with loads of subsurface-relevant tutorials

Don't be discouraged. It takes time to learn any language, including Python. But even if it takes years to become fluent, just a few Python basics can open a whole new world up for you. Have a read of this post by Peter Norvig for some sage advice on how to learn programming.