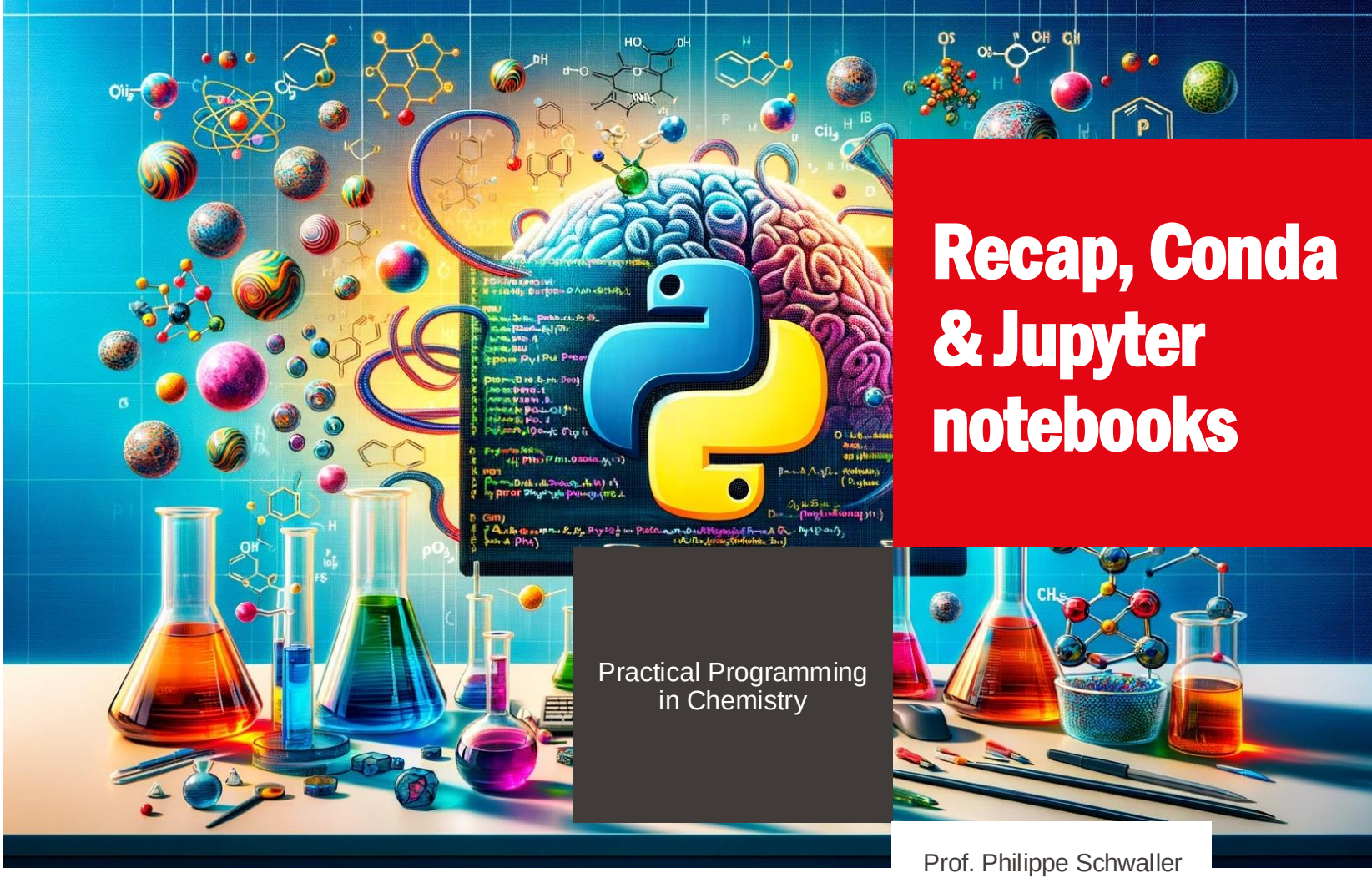


Recap, Conda & Jupyter notebooks

Practical Programming in Chemistry

Prof. Philippe Schwaller

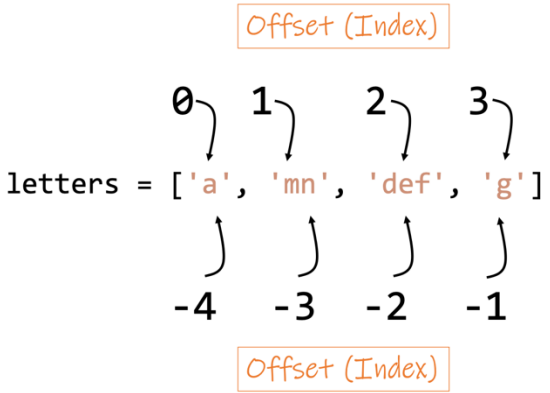
Python recap (continuation of last week) – lists, tuples, sets, dicts, and functions

Basic Python recap – lists

Lists are ordered collections of arbitrary objects: numbers, strings, even other lists!

```
# Create an empty list
my_empty_list = []
```

Square brackets mark the beginning and the end of a list



Indexing works the same as for strings

```
# Lists of compound names and their molecular weights
compound_names = ['Water (H2O)', 'Carbon Dioxide (CO2)', 'Sodium Chloride (NaCl)', 'Glucose (C6H12O6)']
molecular_weights = [18.01528, 44.0095, 58.443, 180.156]
```

Basic Python recap – lists


```
# Lists of compound names and their molecular weights
compound_names = ['Water (H2O)', 'Carbon Dioxide (CO2)', 'Sodium Chloride (NaCl)', 'Glucose (C6H12O6)']
molecular_weights = [18.01528, 44.0095, 58.443, 180.156]
```

```
In [3]: compound_names[1]
Out[3]: 'Carbon Dioxide (CO2)'
```

Access second object of list

```
In [4]: compound_names[-1]
Out[4]: 'Glucose (C6H12O6)'
```

Access last object

```
In [5]: len(compound_names)
Out[5]: 4
```

Count the number of objects in the list

Basic Python recap – lists

```
# Lists of compound names and their molecular weights
compound_names = ['Water (H2O)', 'Carbon Dioxide (CO2)', 'Sodium Chloride (NaCl)', 'Glucose (C6H12O6)']
molecular_weights = [18.01528, 44.0095, 58.443, 180.156]
```

```
In [6]: compound_names + molecular_weights
Out[6]:
['Water (H2O)',
 'Carbon Dioxide (CO2)',
 'Sodium Chloride (NaCl)',
 'Glucose (C6H12O6)',
 18.01528,
 44.0095,
 58.443,
 180.156]
```

Concatenation with **+** operator

```
In [7]: compound_names *2
Out[7]:
['Water (H2O)',
 'Carbon Dioxide (CO2)',
 'Sodium Chloride (NaCl)',
 'Glucose (C6H12O6)',
 'Water (H2O)',
 'Carbon Dioxide (CO2)',
 'Sodium Chloride (NaCl)',
 'Glucose (C6H12O6)']
```

Repetition with ***** operator

Basic Python recap – lists (Membership, is an object in a list)

```
# Lists of compound names and their molecular weights
compound_names = ['Water (H2O)', 'Carbon Dioxide (CO2)', 'Sodium Chloride (NaCl)', 'Glucose (C6H12O6)']
molecular_weights = [18.01528, 44.0095, 58.443, 180.156]
```

```
In [8]: 'Carbon Dioxide (CO2)' in compound_names
Out[8]: True
```

```
In [9]: 'Water' in compound_names
Out[9]: False
```

```
chemical_elements = ['Hydrogen', 'Carbon', 'Oxygen', 'Nitrogen']
element_to_check = 'Oxygen'

# Check if the element exists in the list
if element_to_check in chemical_elements:
    print(f"{element_to_check} is in the list of chemical elements.")
else:
    print(f"{element_to_check} is not in the list of chemical elements.")
```

Basic Python recap – lists (growing)

```
a = [0, 1.1, 2.2]

# Appending
a.append(3.3) # => [0, 1.1, 2.2, 3.3]

# Extending (argument must be an iterable object)
a.extend([4.4, 5.5]) # => [0, 1.1, 2.2, 3.3, 4.4, 5.5]

# Inserting (1st arg. is the el. index before which to insert)
a.insert(-1, 0) # => [0, 1.1, 2.2, 3.3, 4.4, 0, 5.5]
```

Single object to add at the end of the list **a**

Add a **list** at the end of the list **a**

Basic Python recap – lists (searching)


```
a = [0, 1.1, 2.2]

# Searching for an element
a.index(1.1) # => 1
a.index(3.3) # => ValueError: 3.3 is not in list

# Count the number of occurences
a = a * 2 # => [0, 1.1, 2.2, 0, 1.1, 2.2]
a.count(0) # => 2
a.count(3.3) # => 0
```

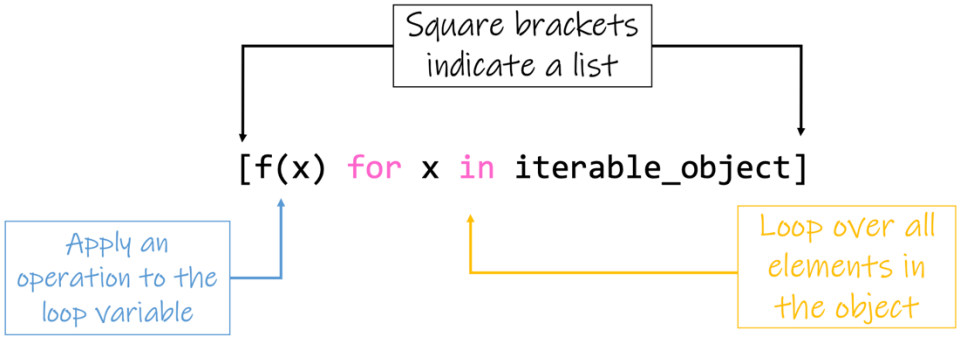
Basic Python recap – lists (sorting, reversing)

```
a = [0, 99, 3, 11, -5]

# Sorting the list
a.sort() # => [-5, 0, 3, 11, 99], increasing order by default
a.sort(reverse = True) # => [99, 11, 3, 0, -5]


# Reverse element order
a = [0, 99, 3, 11, -5]
a.reverse() # => [-5, 11, 3, 99, 0]
```

Basic Python recap – list comprehensions




```
In [14]: celsius = [0, 10, 20, 30]
    ...: fahrenheit = [(9/5) * temp + 32 for temp in celsius]
    ...: print(fahrenheit)  # Output: [32.0, 50.0, 68.0, 86.0]
    ...:
[32.0, 50.0, 68.0, 86.0]
```

Conversion from celsius to fahrenheit.

```
In [15]: ph_values = [3.5, 7.0, 8.3, 6.5, 4.8]
    ...: acidic_solutions = [ph for ph in ph_values if ph < 7]
    ...: print(acidic_solutions)  # Output: [3.5, 6.5, 4.8]
    ...:
[3.5, 6.5, 4.8]
```

Filtering for acidic solutions (ph < 7).

Basic Python recap – tuples

- Very similar to lists, but (”H2O, “CH4”) instead of [”H2O, “CH4”]
- Main difference, they cannot be changed (immutable).

```
In [17]: compound_list = ['H2O', 'CH4']
    ...: compound_tuple = ('H2O', 'CH4')
    ...:
    ...: compound_list.append('CO2') # ['H2O', 'CH4', 'CO2']
    ...: compound_tuple.append('CO2')
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[17], line 5
      2 compound_tuple = ('H2O', 'CH4')
      4 compound_list.append('CO2') # ['H2O', 'CH4', 'CO2']
----> 5 compound_tuple.append('CO2')

AttributeError: 'tuple' object has no attribute 'append'
```

Basic Python recap – sets

Sets are unordered collections of unique elements

```
In [22]: # List of compound names with some duplicates
    ...: compound_names = ["Water", "Sodium Chloride", "Carbon Dioxide", "Glucose", "Wate
    ...: r", "Sodium Chloride"]
    ...:
    ...: # Convert the list to a set to filter out duplicates
    ...: unique_compound_names = set(compound_names)
    ...:
    ...: print(f"The unique compound names are: {unique_compound_names}")
    ...:
The unique compound names are: {'Carbon Dioxide', 'Water', 'Glucose', 'Sodium Chloride'}
```

```
In [21]: # Define the atoms in ethanol
    ...: ethanol_atoms = "CCHHHHOH"
    ...:
    ...: # Convert the string of atoms into a set to find unique atoms
    ...: unique_atoms = set(ethanol_atoms)
    ...:
    ...: print(f"The unique atoms in ethanol are: {unique_atoms}")
The unique atoms in ethanol are: {'H', 'O', 'C'}
```

- Let's imagine you have a list 1M molecules, and now you would like to know how many unique ones there are.

Basic Python recap - sets

```
In [24]: # Set of elements commonly found in organic compounds
    ...: organic_elements = {"C", "H", "O", "N", "S", "P"}
    ...:
    ...: # Set of elements commonly found in inorganic compounds
    ...: inorganic_elements = {"O", "N", "Na", "Cl", "K", "Ca"}
```

```
In [25]: # Union: Elements that are found in either organic or inorganic compounds (or both)
    ...: union_elements = organic_elements.union(inorganic_elements)
    ...: print(f"Union of Elements (Either Organic or Inorganic): {union_elements}")
Union of Elements (Either Organic or Inorganic): {'H', 'K', 'N', 'S', 'Na', 'Cl', 'O', 'Ca', 'P', 'C'}
```

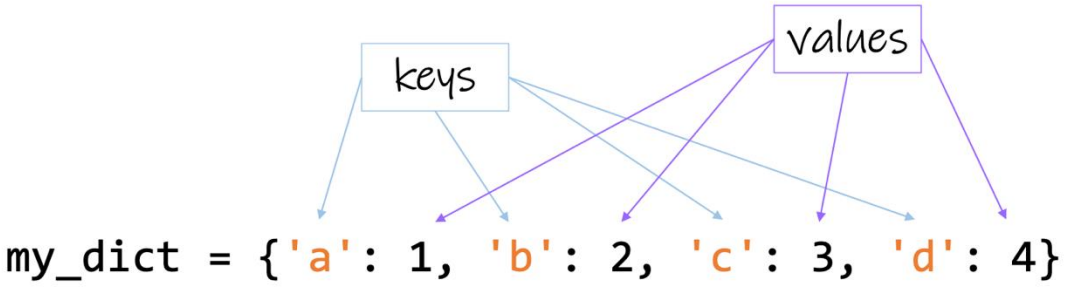
```
In [26]: # Intersection: Elements that are found in both organic and inorganic compounds
    ...: intersection_elements = organic_elements.intersection(inorganic_elements)
    ...: print(f"Intersection of Elements (Both Organic and Inorganic): {intersection_elements}")
Intersection of Elements (Both Organic and Inorganic): {'O', 'N'}
```

```
In [27]: # Difference: Elements that are unique to organic compounds (not found in inorganic compounds)
    ...: difference_elements = organic_elements.difference(inorganic_elements)
    ...: print(f"Elements Unique to Organic Compounds: {difference_elements}")
Elements Unique to Organic Compounds: {'H', 'P', 'C', 'S'}
```

```
In [28]: # Symmetric Difference: Elements that are in either organic or inorganic compounds, but not in both
    ...: symmetric_difference_elements = organic_elements.symmetric_difference(inorganic_elements)
    ...: print(f"Elements Unique to Each Type (Not Shared): {symmetric_difference_elements}")
Elements Unique to Each Type (Not Shared): {'K', 'S', 'Cl', 'Ca', 'C', 'H', 'Na', 'P'}
```

Basic Python recap - dictionaries

- Key, value pairs



keys	values
'a'	1
'b'	2
'c'	3
'd'	4

Basic Python recap - dictionaries

```
# Creating an empty dictionary
my_dict = {}
# Initializing with key-value pairs
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Get the value associated to a key
my_dict['a'] # 1
my_dict['e'] # Raises KeyError

# Change the value or add a new key-value pair
my_dict['e'] = 5 # {'a':1,'b':2,'c':3,'d':4,'e':5}
```

Basic Python recap – functions

```
def name (arg1, arg2, …, argN):
    statements
    return value
```

Similar to if/else statements, indentation defines block

This function will come back, in one of the exercises (tomorrow).

```python
def reaction_yield(theoretical_yield, actual_yield):
    """
    Calculate the percent yield of a reaction.
    theoretical_yield: Theoretical yield in grams
    actual_yield: Actual yield obtained from the reaction in grams
    Returns the percent yield as a percentage.
    """
    percent_yield = (actual_yield / theoretical_yield) * 100
    return percent_yield

# Example usage:
print(reaction_yield(10.0, 8.5))  # Output for a reaction with 10 g theoretical
yield and 8.5 g actual yield
```

=> 85

More exercises and resources

- https://github.com/sib-swiss/first-steps-with-python-training (useful more in-depth exercise notebooks)
- https://realpython.com/search?kind=course&level=basics&order=newest (different topics in more details)
- https://www.kaggle.com/learn/intro-to-programming (super basic intro)
- https://www.kaggle.com/learn/python (intro to Python, different topics)

- If you want to have more than what is shown in today's exercises.

Python packages – how to access additional functions without coding them yourself

What is a package?

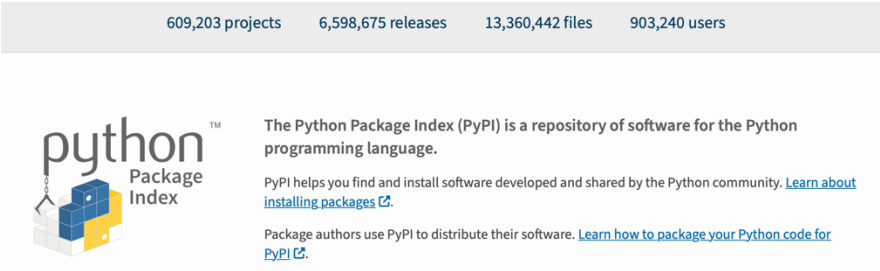
- A package is a **collection of pre-written code** that **adds specific functionality** to your programming environment.
- Think of it like an add-on or plugin that gives you new tools to work with.

```
import numpy as np
average = np.mean([list_of_numbers])
```

```
import pandas as pd
data = pd.read_csv("myfile.csv")
```

How to install packages in Python?

- Through package managers, the two most common are pypi and conda.



pip install numpy

to install the numpy package

ANACONDA®

conda install numpy

to install the numpy package

Environments

What is an environment?

- An environment is like a **separate, isolated workspace** for your programming projects.

Project A
– uses Python 3.7

Project B
– uses Python 3.10

Solution
→ Create an environment for project A and one for B

Conda
-- a powerful package and environment manager

How to create an environment in conda?

- In your command line interface, run the following:

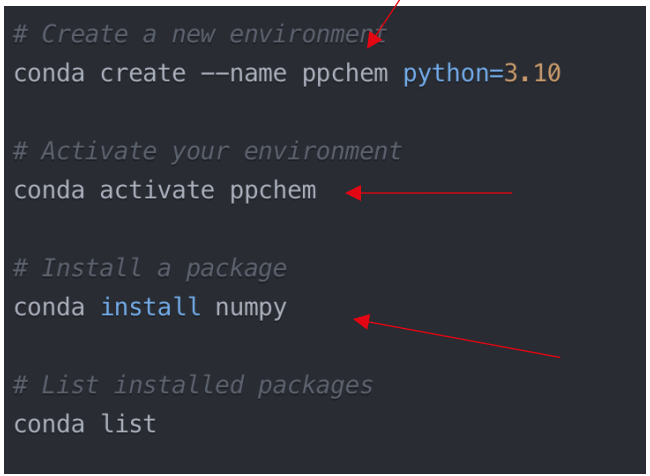
```
# Create a new environment
conda create --name myenv python=3.10

# Activate your environment
conda activate myenv

# Install a package
conda install numpy

# List installed packages
conda list

# Deactivate current environment
conda deactivate
```



Demo on the command line interface

Which environment am I in at the moment?


```
~ ❯                                  protavision system 23:34:07
~ ❯                                  protavision system 23:34:45
~ ❯                                  protavision system 23:34:45
```

```bash
# Check current environment (shows *)
conda env list

# Or use this command
conda info --envs

# See full environment details
conda info
```

```
practical                /Users/pschwllr/miniforge3/envs/practical
practical_programming     /Users/pschwllr/miniforge3/envs/practical
_programming
protavision           *  /Users/pschwllr/miniforge3/envs/protavisio
n
publi                    /Users/pschwllr/miniforge3/envs/publi
py-depict                /Users/pschwllr/miniforge3/envs/py-depict
rdenv                    /Users/pschwllr/miniforge3/envs/rdenv
```

Good practices

- Always activate an environment before installing packages

```
# ❌ Bad Practice:
pip install numpy  # Installing without activating environment

# ✅ Good Practice:
conda activate ppchem
conda install numpy  # Now installing in correct environment
```

Or pip install numpy.

Listing all requirements in a conda environment

- Document your installation steps, e.g. in the README.md

```
# ✅ Keep a record of your environment:
conda list > requirements.txt
# or
conda env export > environment.yml

# ✅ Document your installation steps:
# Installation Steps:
# 1. conda create --name ppchem python=3.10
# 2. conda activate ppchem
# 3. conda install numpy pandas matplotlib
```

Python file formats
- .py and .ipynb

.py - Python Script File

- This is the standard Python source code file
- Plain text file containing Python code
- Can be run directly from the command line or imported as a module

Lecture03 > example.py > ...

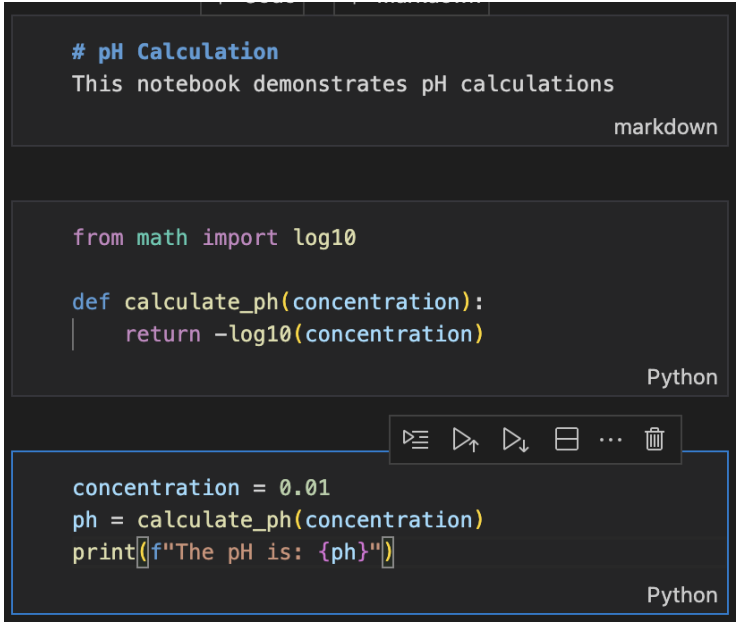
```python
from math import log10

def calculate_ph(concentration):
    return -log10(concentration)

if __name__ == "__main__":
    print(calculate_ph(0.01))
```

.ipynb - Jupyter Notebook File

- Interactive document that combines:
  - Live code
  - Rich text (markdown)
  - Visualizations
  - Equations
- Great for:
  - Data analysis
  - Teaching
  - Documenting research
- Step-by-step explanations



Key differences between .py and .ipynb files

- **.py files** are *linear scripts that run from top to bottom*
- **.ipynb files** are *interactive*, can run cells in *any order*
- **.py files** are better for *production code and software development*
- **.ipynb files** are better for *exploration, analysis, and presentation*

Lecture03 > example.py > ...

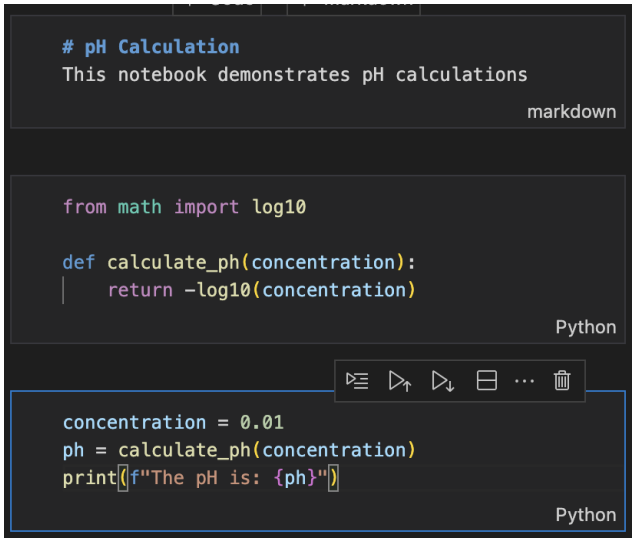
```python
from math import log10

def calculate_ph(concentration):
    return -log10(concentration)

if __name__ == "__main__":
    print(calculate_ph(0.01))
```

To run: *python example.py*

example.ipynb



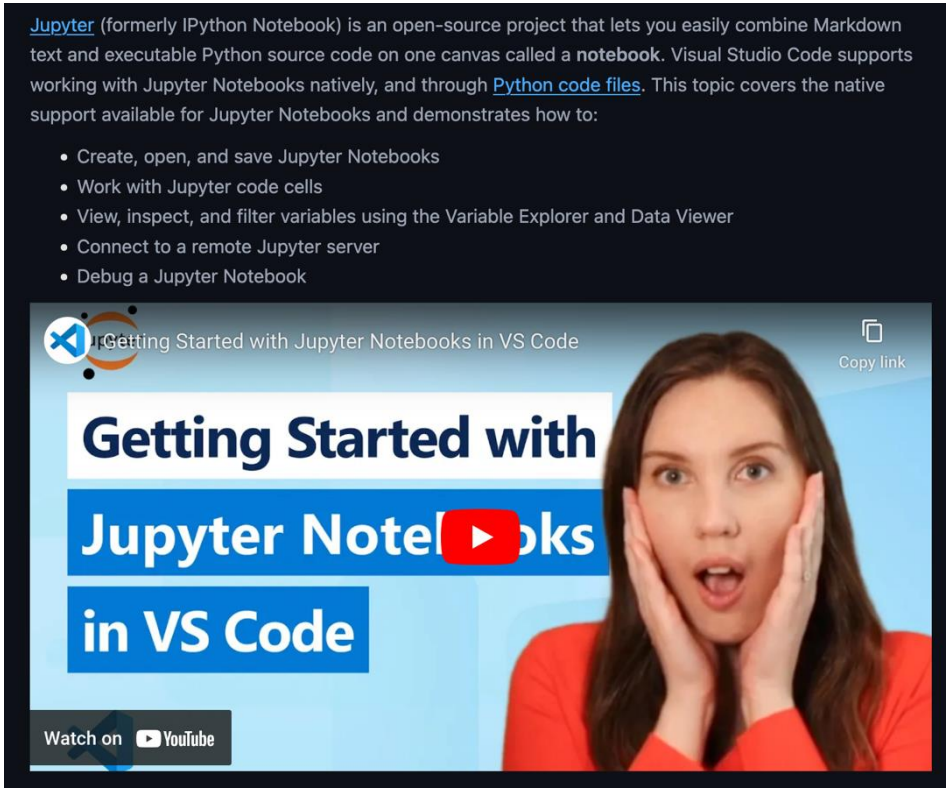
So, when to use which format?

- **.py files** for reusable functions and scripts
- **.ipynb notebooks** for learning, experimenting, and documenting their work

- At the beginning, we will use Jupyter notebooks (.ipynb).

Jupyter notebooks in VS Code

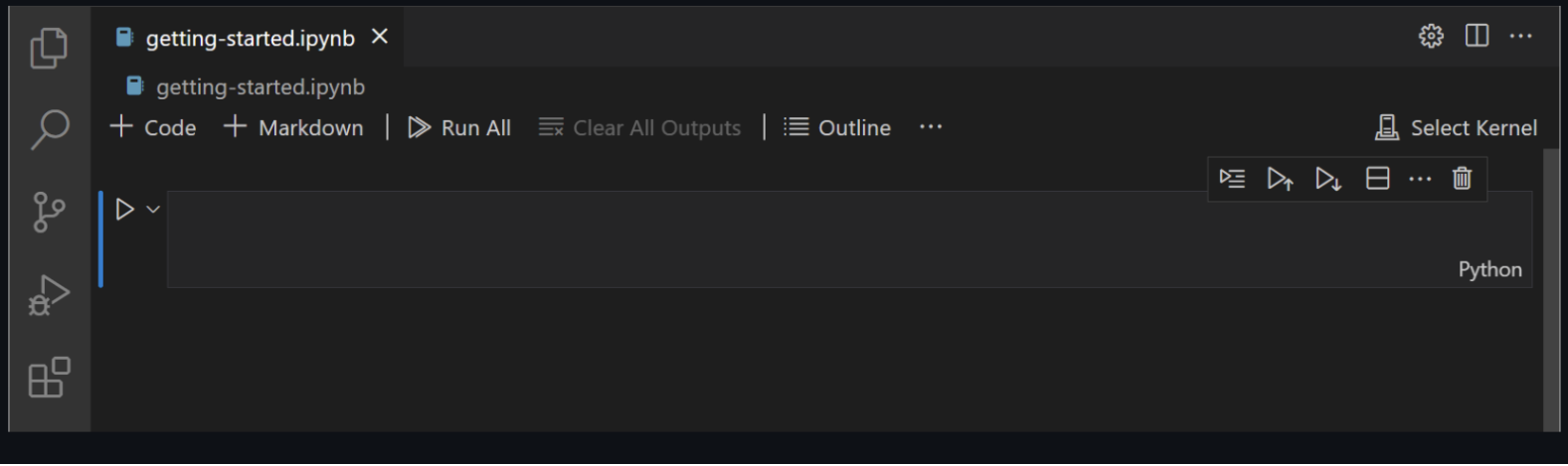
- https://code.visualstudio.com/docs/datascience/jupyter-notebooks



It's a very useful ~6 minute video to watch and follow during the exercise session.

Create or open a Jupyter Notebook

You can create a Jupyter Notebook by running the **Create: New Jupyter Notebook** command from the Command Palette (`⇧⌘P`) or by creating a new `.ipynb` file in your workspace.

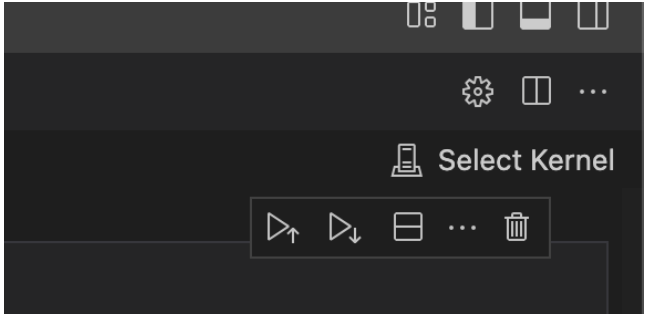


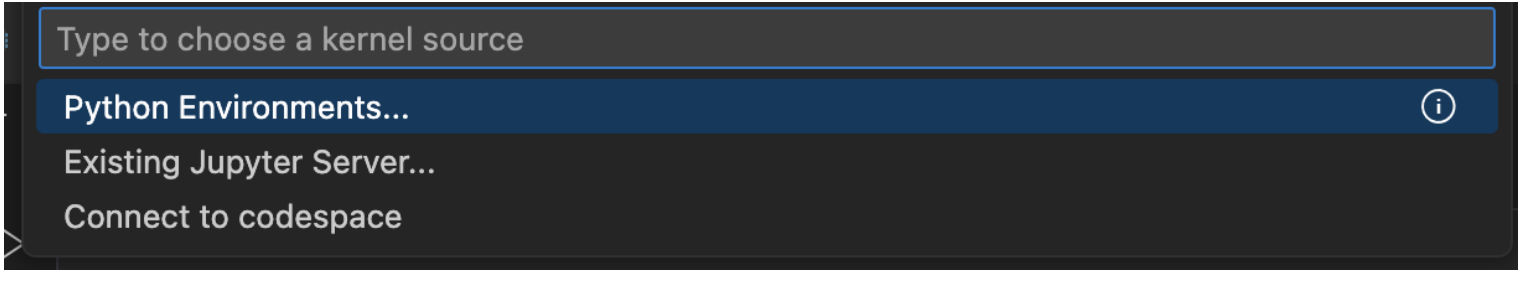
win	shft + ctrl + p
mac	shft + cmd + p

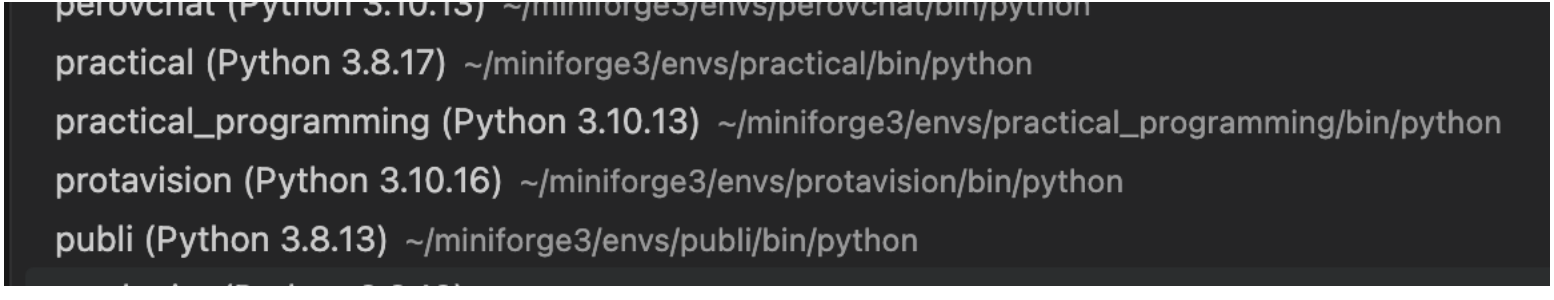
To open **Command Palette**

Select Kernel (select conda environment)

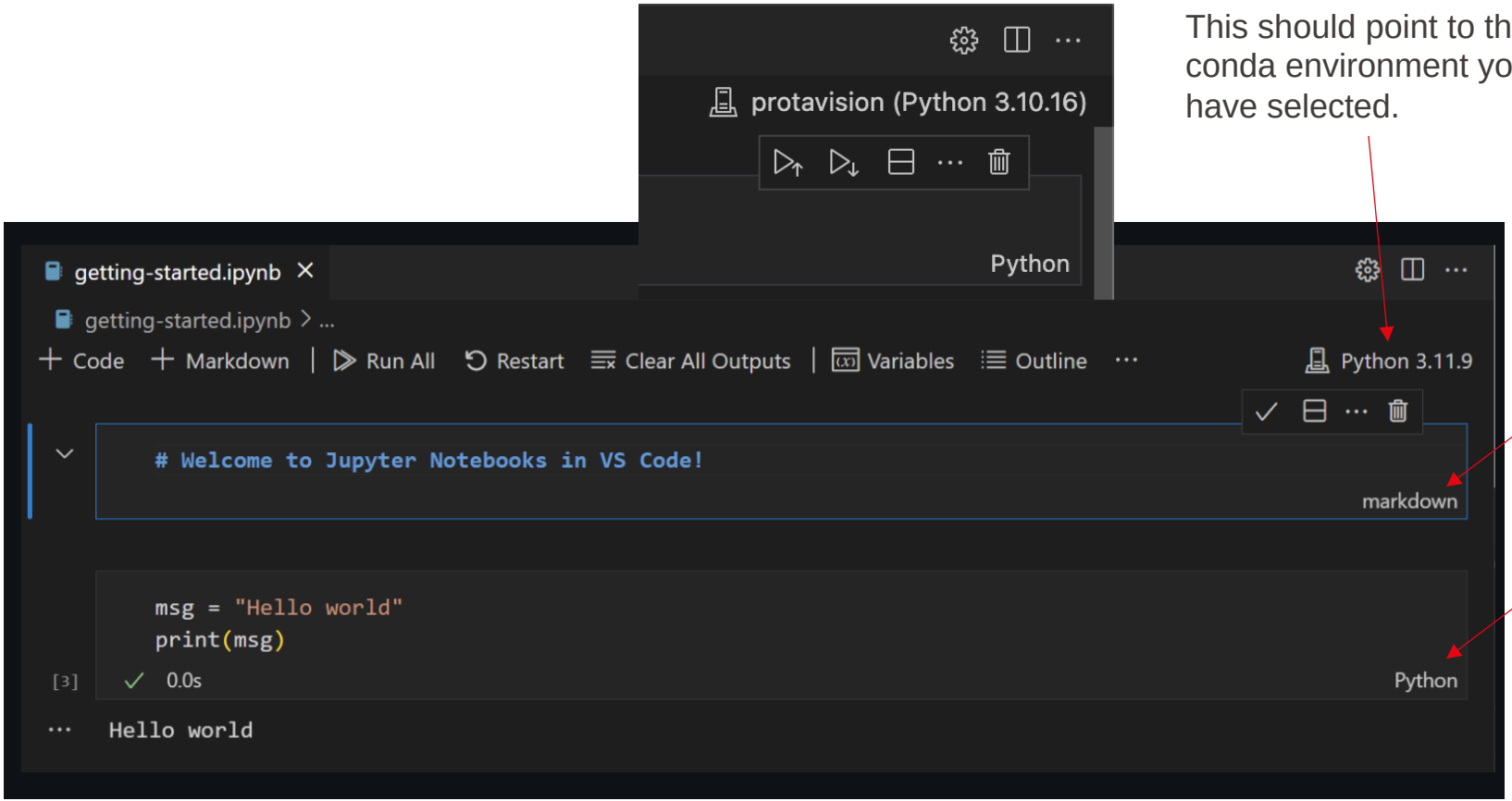
Top right







Markdown vs Python cell types



This should point to the conda environment you have selected.

Exercises today:

- Creating your first conda environment
- Using jupyter notebooks in VSCode
- Basic Python recap (data types, lists, loops, and paths)