

Python/R for Data Science

Lecture notes for 2022 Fall at Arkansas Tech University

Xinli Xiao

2022-08-20T21:43:39-05:00

Table of contents

Preface	3
I Part I: Python	4
1 Preliminaries	5
1.1 Why Python?	5
1.2 Hello world!	5
1.3 Projects	7
2 Python Basics	14
2.1 Built-in Types: numeric types and <code>str</code>	14
2.2 Fundamentals	16
2.3 Flows and Logic	20
2.4 <code>list</code>	20
2.5 <code>dict</code>	21
2.6 Exercises	22
2.7 Projects	26
References	28
Appendices	28
A Setup	29
A.1 VS Code + Anaconda	29
A.2 Google Colab	35
B PATH	37

Preface

This is the lecture notes for STAT 2304 Programming languages for Data Science 2022 Fall at ATU. If you have any comments/suggetions/concers about the notes please contact me at my email xxiao@atu.edu.

Part I

Part I: Python

1 Preliminaries

1.1 Why Python?

1.1.1 Python is easy to learn and use

1.1.2 Python is easy to read

1.1.3 Python Community is mature and supportive

1.2 Hello world!

1.2.1 Setup the Python environment

In this section we are going to setup the Python developing environment.

1.2.1.1 VS Code + Anaconda

Click [Appendix A.1](#) to see the detailed steps for VS Code and Anaconda. You may also check out [the official document](#). It contains more features but less details.

We will talk about the relation between Python and Anaconda and more about packages sometime later.

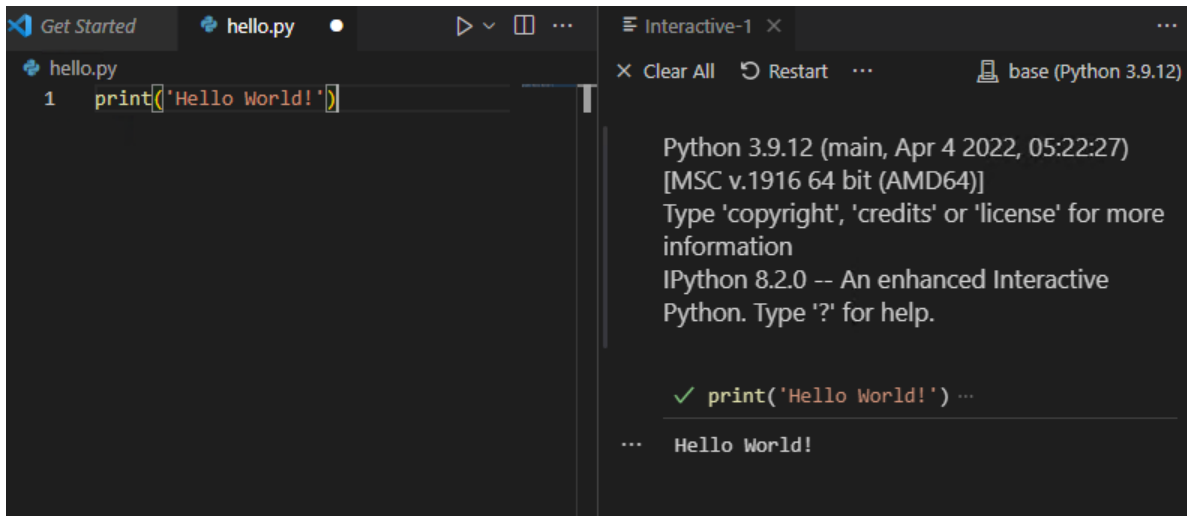
1.2.1.2 Google Colab

Click [Appendix A.2](#) for more details.

1.2.2 Hello World!

Take VS Code as an example. In the editor window, type in the code, and run the file in the interactive window.

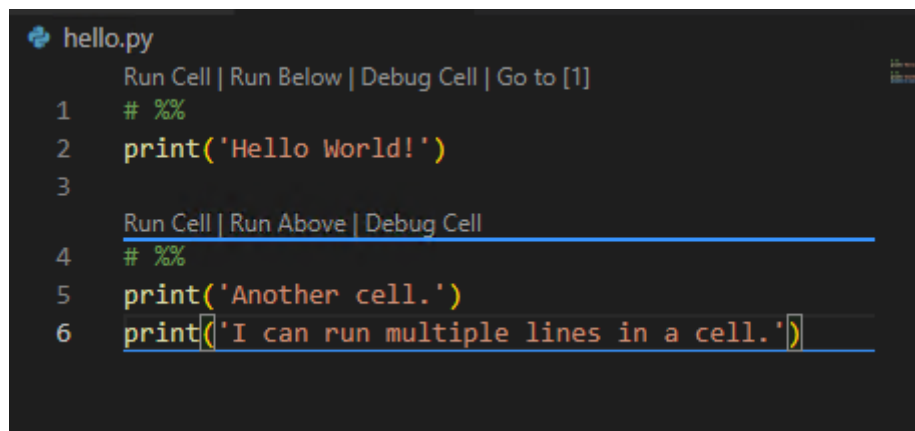
```
print('Hello World!')
```



If you see a small green check mark in the interactive window and also the output Hello World!, you are good to go!

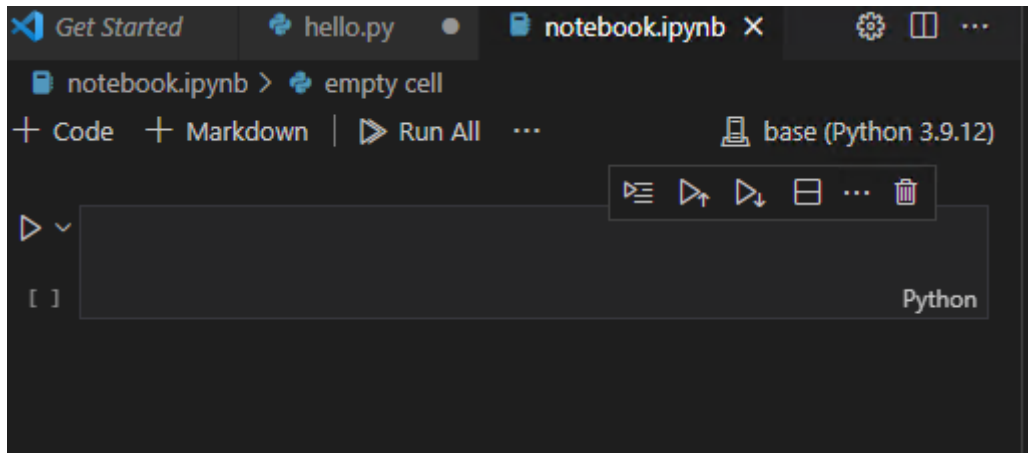
1.2.3 Python code cells and Notebooks

In VS Code you can run codes cell by cell. Each cell is separated by the symbol `# %%`. Each cell may contain multiple lines. You may click the small button on top of the cell or use keybindings.



This feature actually mimic the notebook. We may start a real Python Notebook file by directly creating a file with extension `.ipynb`.

The layout is straightforward.



1.2.4 Linters

1.2.5 IPython and Jupyter

1.3 Projects

Exercise 1.1 (Hello world!). Please set up a Python developing environment, including for .py file and for notebook, that will be used across the semester. Then print **Hello World!**.

Exercise 1.2 (Define a function and play with **time**). Please play with the following codes in a Jupyter notebook. We haven't talked about any of them right now. Try to guess what they do and write your guess in markdown cells.

```
import time

def multistr(x, n=2):
    return x * n

t0 = time.time()
x = 'Python'
print(multistr(x, n=10))
t1 = time.time()
print("Time used: ", t1-t0)
```

Exercise 1.3 (Fancy Basketball plot). Here is an example of the data analysis. We pull data from a dataset, filter the data according to our needs and plot it to visualize the data. This is just a show case. You are encouraged to play the code, make tweaks and see what would happen. You don't have to turn in anything.

The data we choose is Stephen Curry's shots data in 2021-2022 regular season. First we need to load the data. The data is obtained from `nba.com` using `nba_api`.

```
from nba_api.stats.static import players
from nba_api.stats.endpoints import shotchartdetail
player_dict = players.get_players()
```

The shots data we need is in `shotchartdetail`. However to use it we need to know the id of Stephen Curry using the dataset `player_dict`.

```
for player in player_dict:
    if player['full_name'] == 'Stephen Curry':
        print(player['id'])
```

201939

So the id of Stephen Curry is 201939. Let's pull out his shots data in 2021-2022 season.

```
results = shotchartdetail.ShotChartDetail(
    team_id = 0,
    player_id = 201939,
    context_measure_simple = 'FGA',
    season_nullable = '2021-22',
    season_type_all_star = 'Regular Season')
df = results.get_data_frames()[0]
df.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWarning
    return method()
```


	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID
0	Shot Chart Detail	0022100002	26	201939	Stephen Curry	161061274
1	Shot Chart Detail	0022100002	34	201939	Stephen Curry	161061274
2	Shot Chart Detail	0022100002	37	201939	Stephen Curry	161061274
3	Shot Chart Detail	0022100002	75	201939	Stephen Curry	161061274
4	Shot Chart Detail	0022100002	130	201939	Stephen Curry	161061274

`df` is the results we get in terms of a `DataFrame`, and we show the first 5 records as an example.

These are all attempts. We are interested in all made. By looking at all the columns, we find a column called `SHOT_MADE_FLAG` which shows what we want. Therefore we will use it to filter the records.

```
df_made = df[df['SHOT_MADE_FLAG']==1]
df_made.head()
```

C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWarning
return method()

	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID
2	Shot Chart Detail	0022100002	37	201939	Stephen Curry	161061274
6	Shot Chart Detail	0022100002	176	201939	Stephen Curry	161061274
9	Shot Chart Detail	0022100002	352	201939	Stephen Curry	161061274
16	Shot Chart Detail	0022100002	510	201939	Stephen Curry	161061274
18	Shot Chart Detail	0022100002	642	201939	Stephen Curry	161061274

We also notice that there are two columns `LOC_X` and `LOC_Y` shows the coordinates of the attempts. We will use it to draw the heatmap. The full code for drawing out the court `draw_court` is folded below. It is from [Bradley Fay GitHub](#).

i Note

Note that, although `draw_court` is long, it is not hard to understand. It just draws a court piece by piece.

```
from matplotlib.patches import Circle, Rectangle, Arc
import matplotlib.pyplot as plt

def draw_court(ax=None, color='gray', lw=1, outer_lines=False):
    """
    Returns an axes with a basketball court drawn onto to it.
```

This function draws a court based on the x and y-axis values that the NBA stats API provides for the shot chart data. For example, the NBA stat API represents the center of the hoop at the (0,0) coordinate. Twenty-two feet from the left of the center of the hoop in is represented by the (-220,0) coordinates. So one foot equals +/-10 units on the x and y-axis.

```

"""
if ax is None:
    ax = plt.gca()

# Create the various parts of an NBA basketball court

# Create the basketball hoop
hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

# Create backboard
backboard = Rectangle((-30, -7.5), 60, -1, linewidth=lw, color=color)

# The paint
# Create the outer box Of the paint, width=16ft, height=19ft
outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,
                      fill=False)
# Create the inner box of the paint, width=12ft, height=19ft
inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,
                      fill=False)

# Create free throw top arc
top_free_throw = Arc((0, 142.5), 120, 120, theta1=0, theta2=180,
                     linewidth=lw, color=color, fill=False)
# Create free throw bottom arc
bottom_free_throw = Arc((0, 142.5), 120, 120, theta1=180, theta2=0,
                        linewidth=lw, color=color, linestyle='dashed')
# Restricted Zone, it is an arc with 4ft radius from center of the hoop
restricted = Arc((0, 0), 80, 80, theta1=0, theta2=180, linewidth=lw,
                 color=color)

# Three point line
# Create the right side 3pt lines, it's 14ft long before it arcs
corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth=lw,
                           color=color)
# Create the right side 3pt lines, it's 14ft long before it arcs
corner_three_b = Rectangle((220, -47.5), 0, 140, linewidth=lw, color=color)

```

```

# 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
three_arc = Arc((0, 0), 475, 475, theta1=22, theta2=158, linewidth=lw,
                color=color)

# Center Court
center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,
                       linewidth=lw, color=color)
center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,
                       linewidth=lw, color=color)

# List of the court elements to be plotted onto the axes
court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                  bottom_free_throw, restricted, corner_three_a,
                  corner_three_b, three_arc, center_outer_arc,
                  center_inner_arc]

if outer_lines:
    # Draw the half court line, baseline and side out bound lines
    outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                             color=color, fill=False)
    court_elements.append(outer_lines)

# Add the court elements onto the axes
for element in court_elements:
    ax.add_patch(element)

return ax

# Create figure and axes
fig = plt.figure(figsize=(6, 6))
ax = fig.add_axes([0, 0, 1, 1])

# Plot hexbin of shots
ax.hexbin(df['LOC_X'], df['LOC_Y'], gridsize=(30, 30), extent=(-300, 300, 0, 940), bins='1
ax = draw_court(ax, 'black')

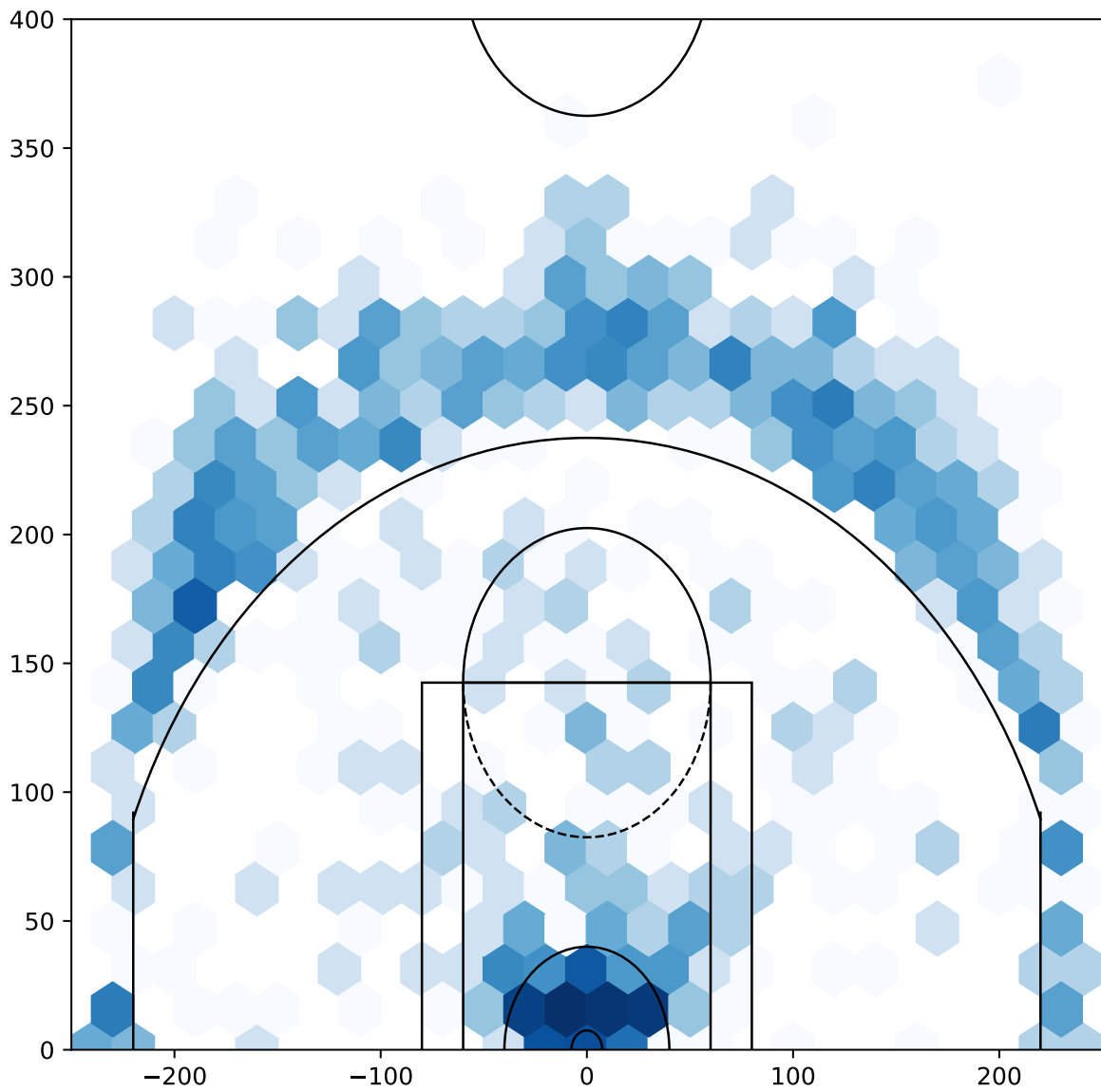
# Annotate player name and season
ax.text(0, 1.05, 'Stephen Curry\n2021-22 Regular Season', transform=ax.transAxes, ha='left

# Set axis limits
_ = ax.set_xlim(-250, 250)

```

```
_ = ax.set_ylim(0, 400)
```

Stephen Curry
2021-22 Regular Season



Click for Hint.

```
Solution  
from nba_api.stats.static import players  
from nba_api.stats.endpoints import shotchartdetail  
player_dict = players.get_players()
```

These lines import some packages and get player information and save them into `player_dict`.

```
for player in player_dict:  
    if player['full_name'] == 'Stephen Curry':  
        print(player['id'])
```

Go through all records in `player_dict`. If the name of a player is `Stephen Curry`, get his `id`. Then we will know the `id` of `Stephen Curry`.

To be omitted.

2 Python Basics

2.1 Built-in Types: numeric types and str

This section is based on [1].

There are several built-in data structures in Python. Here is an (incomplete) list:

- None
- Boolean – True, False
- Numeric Types — int, float, complex
- Text Sequence Type — str
- Sequence Types — list
- Map type - dict

We will cover numeric types and strings in this section. The rests are either simple that are self-explained, or not simple that will be discussed later.

2.1.1 Numeric types and math expressions

Numeric types are represented by numbers. If there are no confusions, Python will automatically detect the type.

```
x = 1 # x is an int.  
y = 2.0 # y is a float.
```

Python can do math just like other programming languages. The basic math operations are listed as follows.

- +, -, *, /, >, <, >=, <= works as normal.
- ** is the power operation.
- % is the mod operation.
- != is not equal

2.1.2 str

Scalars are represented by numbers and strings are represented by quotes. Example:

```
x = 1      # x is a scalar.
y = 's'    # y is a string with one letter.
z = '0'     # z looks like a number, but it is a string.
w = "Hello" # w is a string with double quotes.
```

Here are some facts.

1. For strings, you can use either single quotes ' or double quotes ".
2. \ is used to denote escaped words. You may find the list [Here](#).
3. There are several types of scalars, like `int`, `float`, etc.. Usually Python will automatically determine the type of the data, but sometimes you may still want to declare them manually.
4. You can use `int()`, `str()`, etc. to change types.

Although `str` is a built-in type, there are tons of tricks with `str`, and there are tons of packages related to strings. Generally speaking, to play with strings, we are interested in two types of questions.

- Put information together to form a string.
- Extract information from a string. We briefly talk about these two tasks.

Note

There is a very subtle relation between the variable / constant and the name of the variable / constant. We will talk about these later.

Example 2.1. Here is an example of playing with strings. Please play with these codes and try to understand what they do.

```
import re

def clean_strings(strings):
    result = []
    for value in strings:
        value = value.strip()
        value = re.sub('[!#?]', '', value)
        value = value.title()
```

```

        result.append(value)
    return result

states = [' Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
          'south carolina##', 'West virginia?']
print(clean_strings(states))

```

```
['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Florida', 'South Carolina', 'West Virginia']
```

2.2 Fundamentals

This section is mainly based on [2].

2.2.1 Indentation

One key feature about Python is that its structures (blocks) is determined by **Indentation**.

Let’s compare with other languages. Let’s take C as an example.

```

/*This is a C function.*/
int f(int x){return x;}

```

The block is defined by {} and lines are separated by ;. **space** and **newline** are not important when C runs the code. It is recommended to write codes in a “beautiful, stylish” format for readability, as follows. However it is not mandatory.

```

/*This is a C function.*/
int f(int x) {
    return x;
}

```

In Python, blocks starts from : and then are determined by indents. Therefore you won’t see a lot of {} in Python, and the “beautiful, stylish” format is mandatory.

```

# This is a Python function.
def f(x):
    return x

```

The default value for indentation is 4 spaces, which can be changed by users. We will just use the default value in this course.

Note

It is usually recommended that one line of code should not be very long. If you do have one, and it cannot be shortened, you may break it into multiline codes directly in Python. However, since indentation is super important in Python, when break one line code into multilines, please make sure that everything is aligned perfectly. Please see the following example.

```
results = shotchartdetail.ShotChartDetail(  
    team_id = 0,  
    player_id = 201939,  
    context_measure_simple = 'FGA',  
    season_nullable = '2021-22',  
    season_type_all_star = 'Regular Season')
```

2.2.2 Binary operators and comparisons

Most binary operators behaves as you expected. Here I just want to mention `==` and `is`.

- `==` is testing whehter these two objects have the same value.
- `is` is testing whether these two objects are exactly the same.

Note

You may use `id(x)` to check the id of the object `x`. Two objects are identical if they have the same id.

2.2.3 import

In Python a module is simply a file with the `.py` extension containing Python code. Assume that we have a Python file `example.py` stored in the folder `asests/codes/`. The file is as follows.

```
# from asests/codes/example.py  
display(Markdown(text))  
def f(x):  
    print(x)  
  
A = 'You get me!'
```

You may get access to this function and this string in the following way.

```
from assests.codes import example

example.f(example.A)
```

You get me!

2.2.4 Comments

Any text preceded by the hash mark (pound sign) `#` is ignored by the Python interpreter. In many IDEs you may use hotkeys to directly toggle multilines as comments. For example, in VS Code the default setting for toggling comments is `ctrl+/.`

2.2.5 Dynamic references, strong types

In some programming languages, you have to declare the variable's name and what type of data it will hold. If a variable is declared to be a number, it can never hold a different type of value, like a string. This is called *static typing* because the type of the variable can never change.

Python is a *dynamically typed* language, which means you do not have to declare a variable or what kind of data the variable will hold. You can change the value and type of data at any time. This could be either great or terrible news.

On the other side, “dynamic typed” doesn't mean that types are not important in Python. You still have to make sure that the types of all variables meet the requirements of the operations used.

```
a = 1
b = 2
b = '2'
c = a + b
```

`TypeError: unsupported operand type(s) for +: 'int' and 'str'`

In this example, `b` was first assigned by a number, and then it was reassigned by a `str`. This is totally fine since Python is dynamically types. However later when adding `a` and `b`, the type error occurs since you cannot add a number and a `str`.

Note

You may always use `type(x)` to detect the type of the object `x`.

2.2.6 Everything is an object

Every number, string, data structure, function, class, module, and so on exists in the Python interpreter in its own “box”, which is referred to as a *Python object*.

Each object has an associated type (e.g., string or function) and internal data. In practice this makes the language very flexible, as even functions can be treated like any other object.

Each object might have attributes and/or methods attached.

2.2.7 Mutable and immutable objects

An object whose internal state can be changed is *mutable*. On the other hand, *immutable* doesn't allow any change in the object once it has been created.

Some objects of built-in type that are mutable are:

- Lists
- Dictionaries
- Sets

Some objects of built-in type that are immutable are:

- Numbers (Integer, Rational, Float, Decimal, Complex & Booleans)
- Strings
- Tuples

Example 2.2 (Tuples are not really “immutable”). You can treat a tuple as a container, which contains some objects. The relations between the container and its contents are immutable, but the objects it holds might be mutable. Please check the following example.

```
container = ([1], [2])
print('This is `container`: ', container)
print('This is the id of `container`: ', id(container))
print('This is the id of the first list of `container`: ', id(container[0]))

container[0].append(2)
```

```
print('This is the new `container`: ', container)
print('This is the id of the new `container`: ', id(container))
print('This is the id of the first list (which is updated) of the new `container`: ', id(c
```

```
This is `container`: ([1], [2])
This is the id of `container`: 2106938066368
This is the id of the first list of `container`: 2106938021696
This is the new `container`: ([1, 2], [2])
This is the id of the new `container`: 2106938066368
This is the id of the first list (which is updated) of the new `container`: 2106938021696
```

You can see that the tuple `container` and its first object stay the same, although we add one element to the first object.

2.3 Flows and Logic

2.3.1 for loop

- `range(10)`
- `list`

2.3.2 if conditional control

2.4 list

Note

In Python, a list is an ordered sequence of object types and a string is an ordered sequence of characters.

- Access to the data
- Slicing
- Methods
 - `append` and `+`
 - `extend`
 - `pop`
 - `remove`

- `in`
- `for`
- `list()`
- `sorted`
- `str.split`
- `str.join`

2.4.1 List Comprehension

List Comprehension is a convenient way to create lists based on the values of an existing list. It cannot provide any real improvement to the performance of the codes, but it can make the codes shorter and easier to read.

The format of list Comprehension is

```
newlist = [expression for item in iterable if condition == True]
```

2.5 dict

- Access to the data
- Methods
 - directly add items
 - `update`
 - `get`
 - `keys`
 - `values`
 - `items`
- `dict()`
- dictionary comprehension

2.6 Exercises

Most problems are based on [3], [1] and [4].

Exercise 2.1 (Indentation). Please tell the differences between the following codes. If you don't understand **for** don't worry about it. Just focus on the indentation and try to understand how the codes work.

```
for i in range(5):  
    print('Hello world!')  
print('Hello world!')
```

```
for i in range(5):  
    print('Hello world!')  
    print('Hello world!')
```

```
for i in range(5):  
print('Hello world!')  
print('Hello world!')
```

```
for i in range(5):  
    pass  
print('Hello world!')  
print('Hello world!')
```

Exercise 2.2 (Play with built-in data types). Please first guess the results of all expressions below, and then run them to check your answers.

```
print(True and True)  
print(True or True)  
print(False and True)  
print((1+1>2) or (1-1<1))
```

Exercise 2.3 (**=** vs **is**). Please explain what happens below.

```

a = 1
b = 1.0
print(type(a))
print(type(b))

print(a == b)
print(a is b)

```

```

<class 'int'>
<class 'float'>
True
False

```

Exercise 2.4 (Play with strings). Please excute the code below line by line and explain what happens in text cells.

```

# 1
answer = 10
wronganswer = 11
text1 = "The answer to this question is {}. If you got {}, you are wrong.".format(answer,
print(text1)

# 2
var = True
text2 = "This is {}".format(var)
print(text2)

# 3
word1 = 'Good '
word2 = 'buy. '
text3 = (word1 + word2) * 3
print(text3)

# 4
sentence = "This is\ngood enough\nfor a exercise to\nhave so many parts. " \
           "We would also want to try this symbol: '. ' \
           "Do you know how to type \" in double quotes?"
print(sentence)

```

The answer to this question is 10. If you got 11, you are wrong.

This is True.

Good buy. Good buy. Good buy.

This is

good enough

for a exercise to

have so many parts. We would also want to try this symbol: '. Do you know how to type " in d

Exercise 2.5 (split and join). Please excute the code below line by line and explain what happens in text cells.

```
sentence = 'This is an example of a sentence that I expect you to split.'

wordlist = sentence.split(' ')

newsentence = '\n'.join(wordlist)
print(newsentence)
```

Exercise 2.6 (List reference). 1. Given the list **a**, make a new reference **b** to **a**. Update the first entry in **b** to be 0. What happened to the first entry in **a**? Explain your answer in a text block.

2. Given the list **a**, make a new copy **b** of the list **a** using the function **list**. Update the first entry in **b** to be 0. What happened to the first entry in **a**? Explain your answer in a text block.

Exercise 2.7 (List comprehension). Given a list of numbers, use list comprehension to remove all odd numbers from the list:

```
numbers = [3,5,45,97,32,22,10,19,39,43]
```

Exercise 2.8 (More list comprehension). Use list comprehension to find all of the numbers from 1-1000 that are divisible by 7.

Exercise 2.9 (More list comprehension). Count the number of spaces in a string.

Exercise 2.10 (More list comprehension). Use list comprehension to get the index and the value as a tuple for items in the list `['hi', 4, 8.99, 'apple', ('t,b', 'n')]`. Result would look like `[(index, value), (index, value), ...]`.

Exercise 2.11 (More list comprehension). Use list comprehension to find the common numbers in two lists (without using a tuple or set) `list_a = [1, 2, 3, 4]`, `list_b = [2, 3, 4, 5]`.

Exercise 2.12 (Probability). Compute the probability that two people out of 23 share the same birthday. The math formula for this is

$$1 - \frac{365!/(365-23)!}{365^{23}} = 1 - \frac{365}{365} \cdot \frac{365-1}{365} \cdot \frac{365-2}{365} \cdot \dots \cdot \frac{365-22}{365}.$$

1. To directly use the formula we have to use a high performance math package, e.g. `math`. Please use `math.factorial` to compute the above formula.
2. Please use the right hand side of the above formula to compute the probability using the following steps.
 - a. Please use the list comprehension to create a list `[$\frac{365}{365}, \frac{365-1}{365}, \frac{365-2}{365}, \dots, \frac{365-22}{365}$]`.
 - b. Use `numpy.prod` to compute the product of elements of the above list.
 - c. Compute the probability by finishing the formula.
3. Please use `time` to test which method mentioned above is faster.

2.7 Projects

Most projects are based on [2], [5].

Exercise 2.13 (Determine the indefinite article). Please finish the following tasks. 1. Please construct a list `aeiou` that contains all vowels. 2. Given a word `word`, we would like to find the indefinite article `article` before `word`. (Hint: the article should be `an` if the first character of `word` is a vowel, and `a` if not.)

Click for Hint.

Solution. Consider `in`, `.lower()` and `if` structure.

Exercise 2.14 (Datetime and files names). We would like to write a program to quickly generate `N` files. Every time we run the code, `N` files will be generated. We hope to store all files generated and organize them in a neat way. To achieve this, one way is to create a subfolder for each run and store all files generated during that run in the particular subfolder. Since we would like to make it fast, the real point of this task is to find a way to automatically generate the file names for the files generated and the folder names for the subfolders generated. You don't need to worry about the contents of the files and empty files are totally fine for this problem.

Click for Hint.

Solution. One way to automatically generate file names and folder names is to use the date and the time when the code is run. Please check `datetime` package for getting and formatting date/time, and `os` packages for playing with files and folders.

Exercise 2.15 (Color the Gnomonic data). We can use ASCII color codes in the string to change the color of strings, as an example `\033[91m` for red and `\033[94m` for blue. See the following example.

```
print('\033[91m'+ 'red' + '\033[92m'+ 'green' + '\033[94m'+ 'blue' + '\033[93m'+ 'yellow')
```

Consider an (incomplete) Gnomonic data given below which is represented by a long sequence of A, C, T and G. Please color it using ASCII color codes.

```
Gnomicdata = 'TCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAAATCTGTGTGGCTGTCACTCGG'\
              'CTGCATGCTTAGTGCACTCACGCAGTATAATTAATAACTAATTACTGTCGTTGACAGGAC'\
              'ACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTCGTCCGTGTTGCAGCCGATC'\
              'ATCAGCACATCTAGGTTTTGTCCGGGTGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCC'\
              'TGGTTTCAACGAGAAAAACACACGTCCAACCTCAGTTTGCCTGTTTTACAGGTTGCGGACGT'\
              'GCTCGTACGTGGCTTTGGAGACTCCGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCT'\
              'TAAAGATGGCACTTGTGGCTTAGTAGAAGTTGAAAAAGGCGTTTTGCCTCAACTTGAACA'\
              'GCCCTATGTGTTTCATCAAACGTTTCGGATGCTCGAACTGCACCTCATGGTCATGTTATGGT'\
              'TGAGCTGGTAGCAGAACTCGAAGGCATTTCAGTACGGTCGTAGTGGTGAGACACTTGGTGT'\
              'CCTTGTCCCTCATGTGGGCGAAATACCAGTGGCTTACCGCAAGGTTCTTCTTCGTAAGAA'\
              'CGGTAATAAAGGAGCTGGTGGCCATAGTTACGGCGCCGATCTAAAGTCATTTGACTTAGG'\
              'CGACGAGCTTGGCACTGATCCTTATGAAGATTTTCAAGAAAACGTGGAACACTAAACATAG'
```

Click for Hint.

Solution (Hint). You may use `if` to do the conversion. Or you may use `dict` to do the conversion.

References

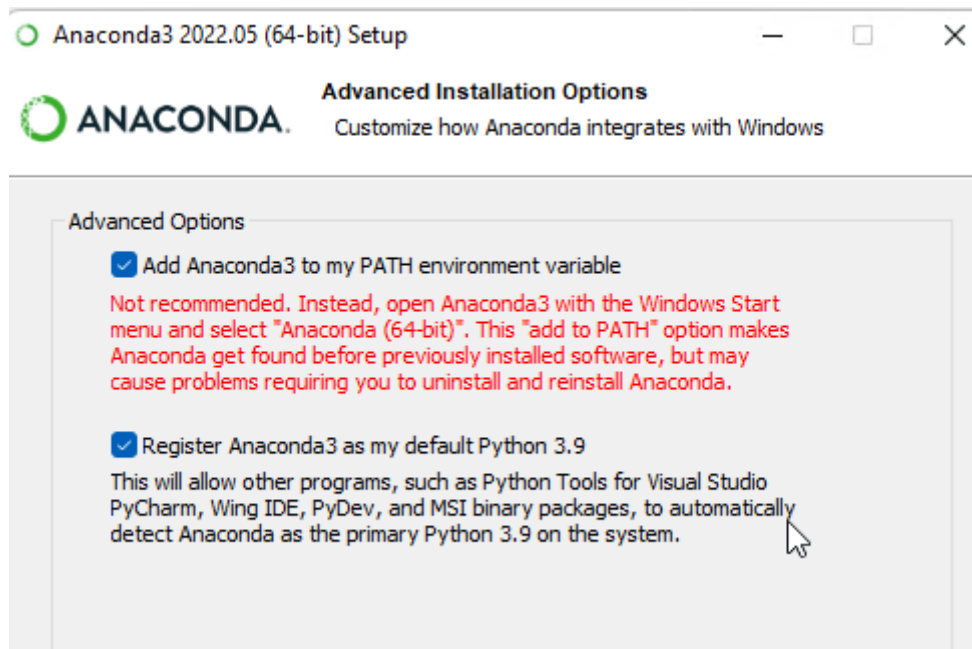
- [1] YOUENS-CLARK, K. (2020). *Tiny python projects*. Manning Publications.
- [2] MCKINNEY, W. (2017). *Python for data analysis: Data wrangling with pandas, NumPy, and IPython*. O'Reilly Media.
- [3] SHAW, Z. A. (2017). *Learn python 3 the hard way*. Addison Wesley.
- [4] SWEIGART, A. (2020). *Automate the boring stuff with python, 2nd edition practical programming for total beginners: Practical programming for total beginners*. No Starch Press.
- [5] KLOSTERMAN, S. (2021). *Data science projects with python: A case study approach to gaining valuable insights from real data with machine learning*. Packt Publishing, Limited.

A Setup

A.1 VS Code + Anaconda

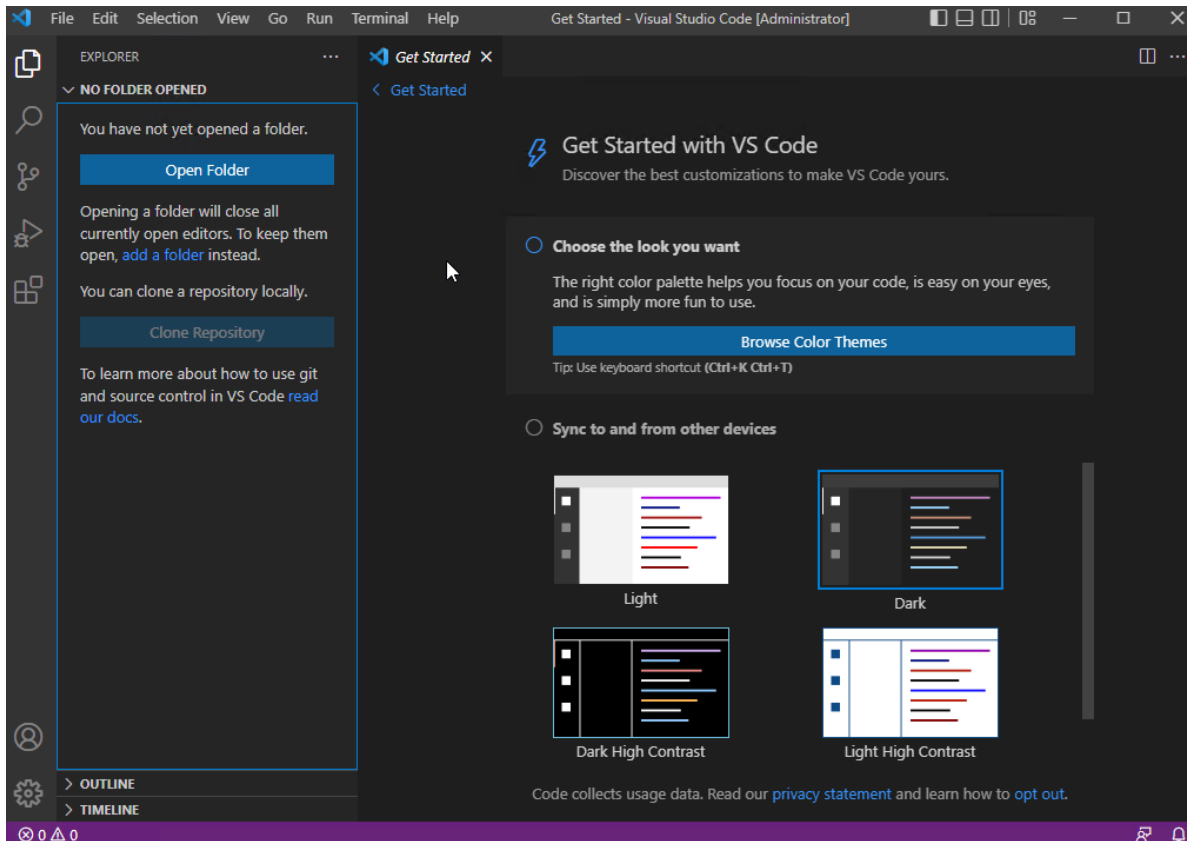
Note that all the following steps are tested in Windows 10/11. If you use other operation systems please contact me.

1. Go to [Anaconda download page](#). Download and install Anaconda.
2. Go to [VS Code download page](#). Download and install VS Code. Actually Anaconda contains one copy of VS Code. Here I just assume that some of you install VS Code before Anaconda.
3. When installing VS Code, you may accept all default settings. When installing Anaconda, please pay attention to the PATH setting.



The first box is unchecked by default. This setting is related to the ability to easily run Python code in Terminals. I recommend you to check it. If you don't check it during this step, you may add it to the system environment variable PATH manually later.

4. The UI of VS Code looks as follows.



Please look at the fifth tab from the left sidebar. It is the Extension tab.

Please search for `python` and install the first Python extension from Microsoft. It will actually install five extensions. These are all we need for now.

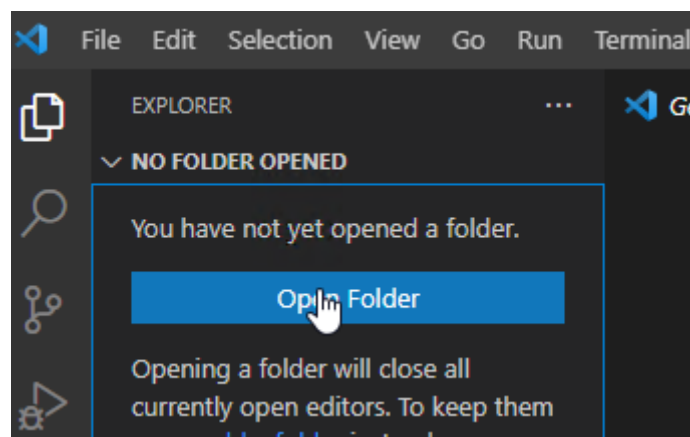
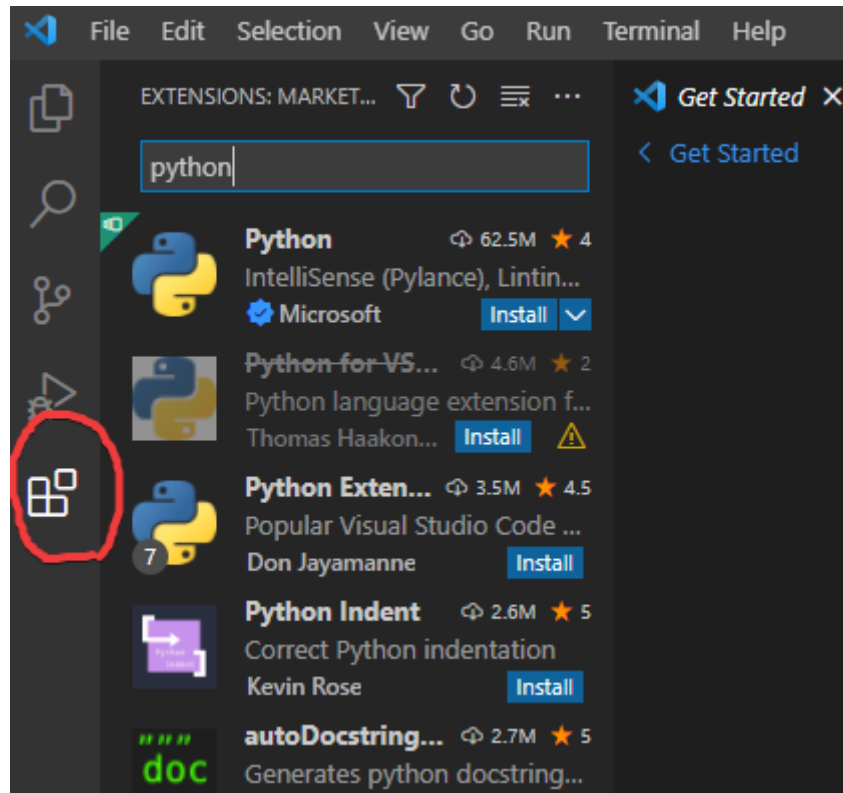
5. After all are installed, go to the first Explorer tab on the left side bar, and **Open Folder**. This is the working directory for your project.

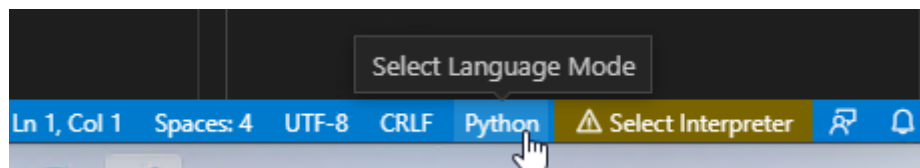
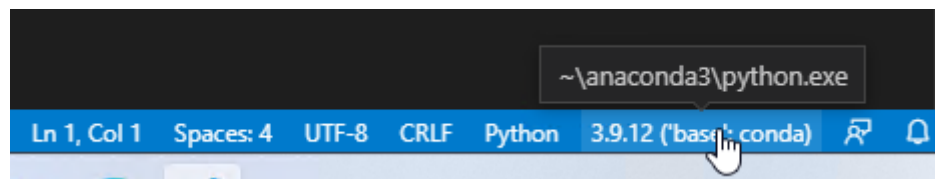
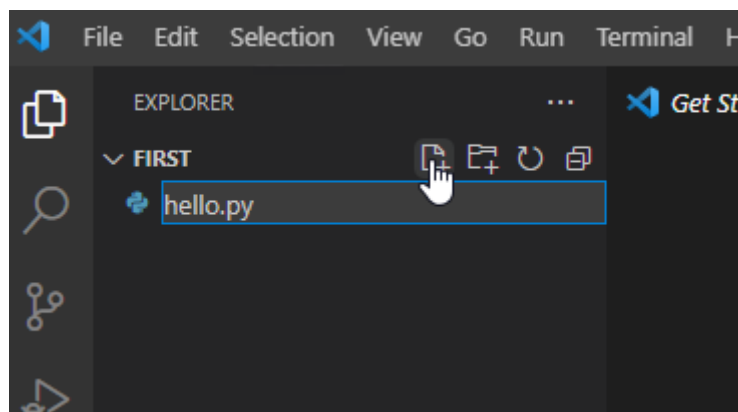
Choose one folder and start a new `.py` file.

6. If everything is setup correctly, you may see the Python version and environment name at the right lower corner. In our case the environment name is **base**. We will need it in the future.

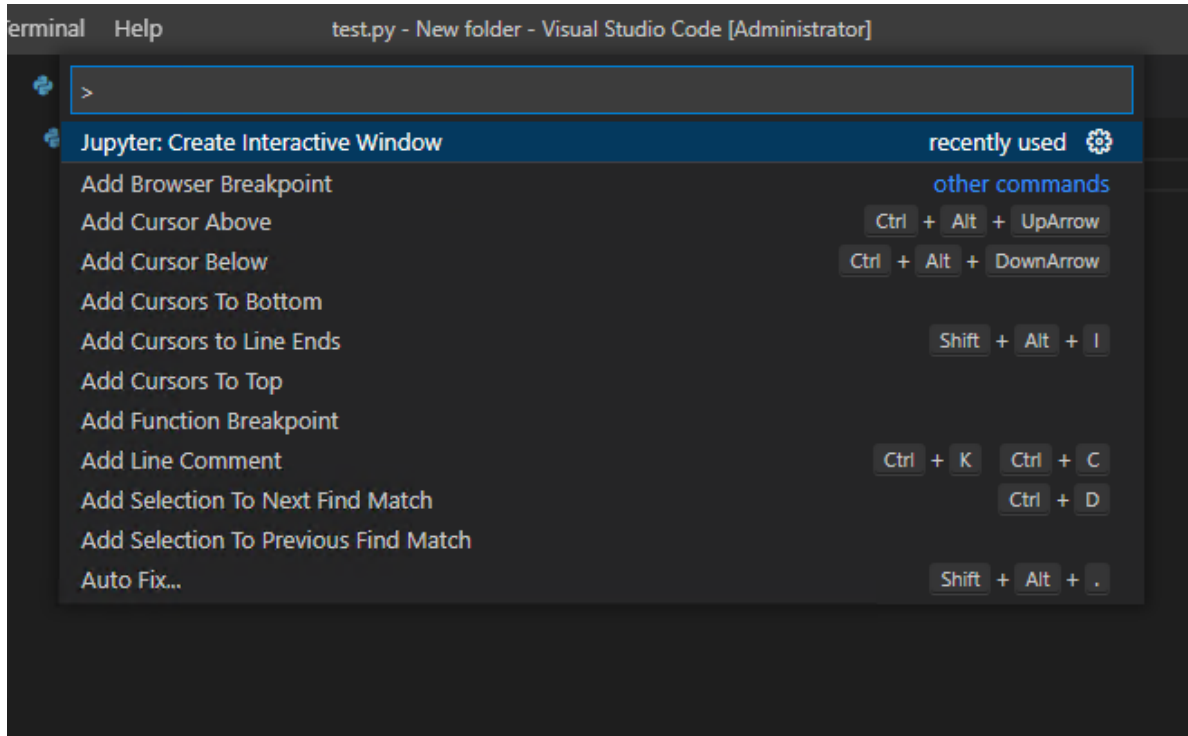
Note that we are not looking at the **Python for Language Mode**. If you see **Select Interpreter** there, it means that VS Code doesn't find your Python interpreter. Please restart VS Code or select it manually, or check whether Anaconda is installed correctly.

To check whether everything is setup correctly, please run the following tests.





1. Use `ctrl+shift+p` to open the Command Palette, type “Jupyter: Create Interactive Window” and press `enter` to open the Jupyter interactive window.



If the interactive window starts and you see the loading information of your kernel as follows, especially you see the environment name on the right upper corner, then you get everything correctly. However we will still do more tests.

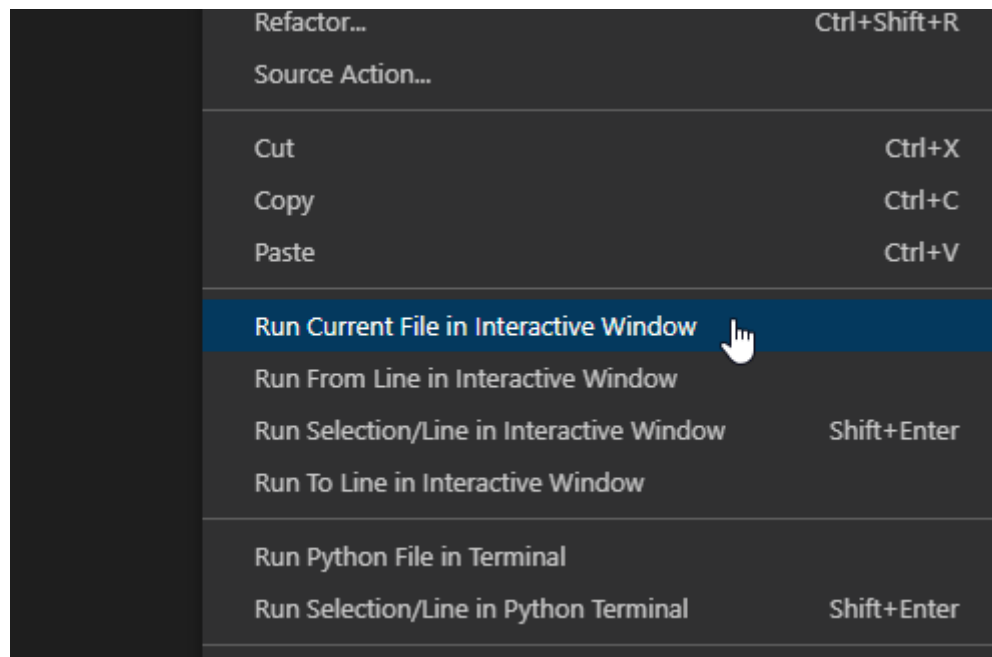
2. In the window type `import numpy as np` to test whether you are able to import packages. If you don't see any error messages then it means good.
3. In the editor window, type `import numpy as np` and right click the body to choose `Run Current File in Interactive Window`, and see whether it runs in interactive window.
4. Open the terminal. Please use `Command Prompt` instead of `Powershell`. Activate the conda environment by type the command `conda activate base` in the example above. Please change the name to match your own environment. If `conda` cannot be recognized, please register Python and Anaconda to the system environment path. Please see the next Appendix for details.

Interactive-1 X

X Clear All Restart ... base (Python 3.9.12)

```
Python 3.9.12 (main, Apr 4 2022,
05:22:27) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for
more information
IPython 8.2.0 -- An enhanced Interactive
Python. Type '?' for help.
```

```
[1] import numpy as np
✓ 0.2s
```



A.2 Google Colab

Google Colab is a product from Google Research, that allows anybody to write and execute arbitrary Python code through the browser, and is especially well suited to machine learning, data analysis and education.

Here is the link to [Google Colab](#). To use it you should have a Google account. Otherwise it is very simple to start, since a lot of packages for our course are already installed.

A.2.1 Install packages

If you would like to install more packages, you can type `%pip install + package name` in a code cell and execute it.

The drawback here is that Google Colab can only stay for 24 hours. After that, all additionally installed packages will be earsed. However you can put `%pip install + package name` at the beginning of your notebook and these packages will be installed every time you run the notebook.

A.2.2 Upload files

You may directly upload files to the working directory of Google Colab. This has to be done in the browser. When working with these files, you may just use relative paths.

The drawback here is that Google Colab can only stay for 24 hours. After that, although your `.ipynb` files will be stores, all other files will be earsed.

A.2.3 Mount Google Drive

One way to let the uploaded files stay in cloud is to upload them to Google Drive, and then load your Google Drive contents from Google Colab.

Goole Drive is a cloud storage service provided by Google. When you register a Google account you will be automatically assigned a Google Drive account. You may get access to it from [this link](#).

Here are the steps to mount Google Drive:

1. Upload your files to your Google Drive.
2. Run the following codes in Colab code cells before you are loading the uploaded files:

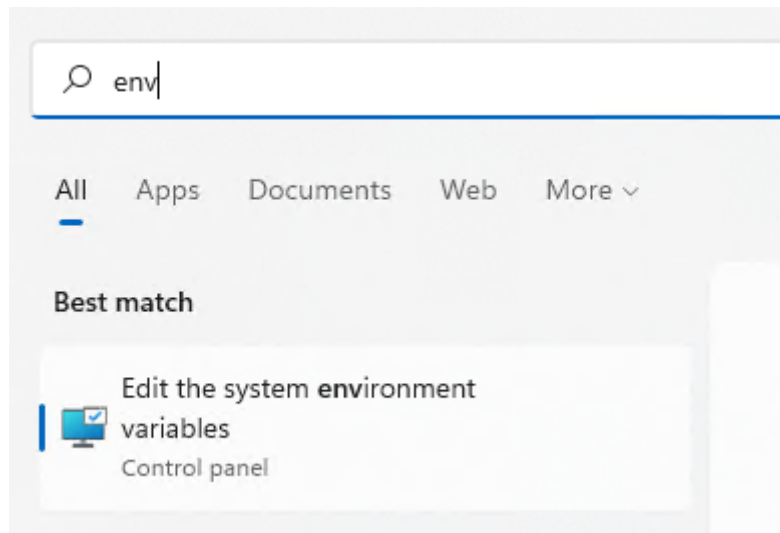
```
from google.colab import drive
drive.mount('/content/gdrive')
```

3. A window pop up asking you about the permission. Authorize and the drive is mounted.
4. To work in directories, the most popular commands are
 - `%ls`: list all files and folders in the working directory.
 - `%cd + folder name`: Get into a specific folder.
 - `%cd ..`: Get into the parent folder. Then use these commands to find the files your just uploaded.
5. Finally you may directly get access to those files just like they are in the working directory.

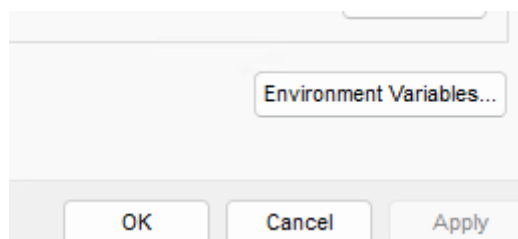
B PATH

Here are the steps to edit the system environment variables in Windows 10/11.

1. First in the start menu search for `Edit the system environment variables`.



2. Then click the `Environment Variables...` button at the right lower corner.



3. Find the `Path` variable in either the upper window or the lower window. Use which one depends on whether you want to register the variable for the user or for the machine. In this example I add for the user.
4. Finally double click the variable and add the following path to it. You need to make changes according to your installation. I recommend you to locate your Anaconda installation first to get the path.

User variables for WDAGUtilityAccount

Variable	Value
Path	C:\Users\WDAGUtilityAccount\anaconda3;C:\Users\WDAGUtilityAc...
TEMP	C:\Users\WDAGUtilityAccount\AppData\Local\Temp
TMP	C:\Users\WDAGUtilityAccount\AppData\Local\Temp

New... Edit... Delete

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

New... Edit... Delete

C:\Users\WDAGUtilityAccount\anaconda3
C:\Users\WDAGUtilityAccount\anaconda3\Library\mingw-w64\bin
C:\Users\WDAGUtilityAccount\anaconda3\Library\usr\bin
C:\Users\WDAGUtilityAccount\anaconda3\Library\bin
C:\Users\WDAGUtilityAccount\anaconda3\Scripts