DSS5201 Data Visualization
Week 1

Yuting Huang

NUS DSDS

2024-08-12

Introduction to Python

What is coding?

Computing programming, or **coding**, is the process of issuing a series of commands that a computer can understand and execute in order to perform a task.

- Imagine a computer as a particularly mischievous genie.
- We need to write out the instruction (code) very explicitly so the computer will do it in the way we anticipate.

Why should we learn coding?

Today, almost all work with some quantitative aspect involves coding.

- An **essential skill** across a wide range of domains.
  - Widely used across industry, academia, and the public sector.
- Many of the tools and packages that have been developed will help you **do better work more productively**.

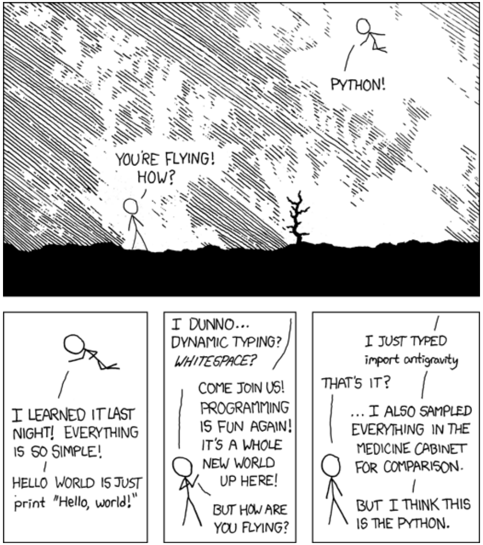
Learning basic programming concepts in *any* programming language is useful for almost any other language.

Preliminaries

This course uses Python, which is ranked as the first or second most popular programming language in the world.

- It's a general purpose language – can perform a wide range of tasks.
- Also, one of the easiest to learn.

Preliminaries


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Source: xkcd.

A few concepts

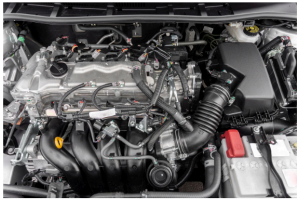
Now, let's introduce the key elements of a computational environment for coding.

- A computer with an **operating system**.
- An **installation of Python**, so the computer can interpret and execute Python code.
- An **integrated development environment (IDE)**, a place to write and run code.
- **Packages**, which extends the functionality of Python.

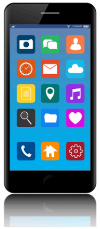
Let's talk through each of these in more details.

Python, IDE, and Python packages

Python: Engine **VSCode: Dashboard**



Python: A new phone **Python packages: Apps you can download**



Python is both a programming language that human can read, and a language that computers can read, interpret, and then carry out instructions.

- For a computer to read and execute Python code, it needs a Python interpreter.
- We can download the latest version from Python.org.
- Then install Python with default settings.

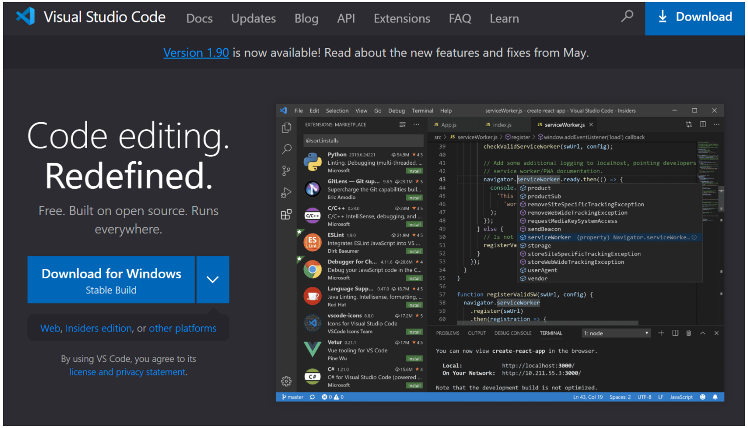
VSCode

An **integrated development environment (IDE)** is a software that provides tools to make coding easier.

We will use Microsoft's **VSCode**.

- A way to run code interactively.
- Automatic code checking for basic errors.
- Coloring your brackets in pairs so you can keep track of the logical order of execution of your code.
- A quick way to access help documentations about common packages.

- We can install the stable build of VSCode from the official website.



Version 1.90 is now available! Read about the new features and fixes from May.

Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its license and privacy statement.

Python 2019.6.24221	54.9M	4.5	Linting, Debugging (multi-threaded, ... Microsoft Install
GitLens — Git sup... 9.8.5	23.1M	5	Supercharge the Git capabilities buil... Eric Amodio Install
C/C++ 0.24.0	23M	3.5	C/C++ IntelliSense, debugging, and ... Microsoft Install
ESLint 1.9.0	21.9M	4.5	Integrates ESLint JavaScript into VS ... Dirk Baeumer Install
Debugger for Ch... 4.11.6	20.6M	4	Debug your JavaScript code in the C... Microsoft Install
Language Supp... 0.47.0	18.7M	4.5	Java Linting, Intellisense, formatting, ... Red Hat Install
vscode-icons 8.8.0	17.2M	5	Icons for Visual Studio Code VSCode Icons Team Install
Vetur 0.21.1	17M	4.5	Vue tooling for VS Code Pine Wu Install
C# 1.21.0	15.6M	4	C# for Visual Studio Code (powered ... Microsoft Install

```
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, pointing developers
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          console.
            'This
              'wor
          );
        });
      } else {
        // Is not
        registerVa
      }
    });
  }
}

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
```

```
product
productSub
removeSiteSpecificTrackingException
removeWebWideTrackingException
requestMediaKeySystemAccess
sendBeacon
serviceWorker  (property) Navigator.serviceWorke…
storage
storeSiteSpecificTrackingException
storeWebWideTrackingException
userAgent
vendor
```

```
You can now view create-react-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.211.55.3:3000/

Note that the development build is not optimized.
```

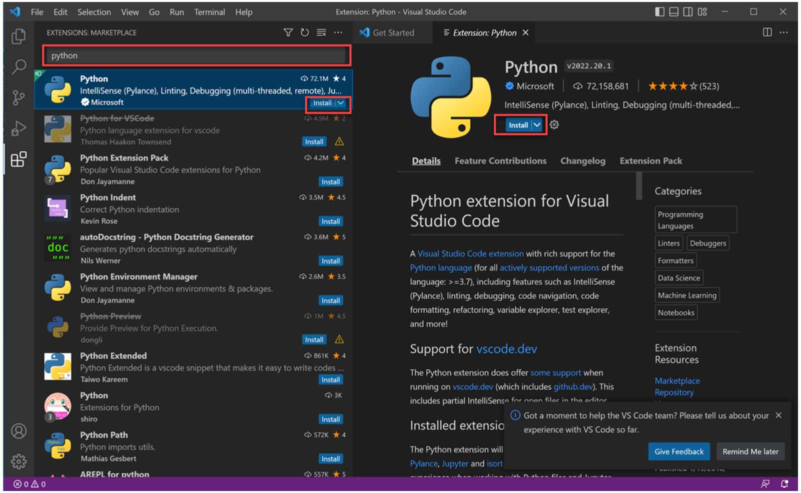
Essential VSCode Python extensions

The following extensions will provide us helpful functionalities in VSCode.

- Click on the activity bar or use a keyboard shortcut (*Cmd* + *Shift* + *X*) to open the extension panel.
- Type the following keywords to search for these extensions:
  - Python
  - Python Indent
  - Jupyter
  - Jupyter Notebook Renderers
  - Data Wrangler

Essential VSCode Python extensions

Install the following extensions: Python, Python Indent, Jupyter, Jupyter Notebook Renderers, Data Wrangler.



Python 72.1M ★ 4
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Ju...
Microsoft Install

~~Python for VSCode~~ 4.9M ★ 2
Python language extension for vscode
Thomas Haakon Townsend Install

Python Extension Pack 4.2M ★ 4
Popular Visual Studio Code extensions for Python
Don Jayamanne Install

Python Indent 3.5M ★ 4.5
Correct Python indentation
Kevin Rose Install

autoDocstring - Python Docstring Generator 3.6M ★ 5
Generates python docstrings automatically
Nils Werner Install

Python Environment Manager 2.6M ★ 3.5
View and manage Python environments & packages.
Don Jayamanne Install

~~Python Preview~~ 1M ★ 4.5
Provide Preview for Python Execution.
dongli Install

Python Extended 861K ★ 4
Python Extended is a vscode snippet that makes it easy to write codes ...
Taiwo Kareem Install

Python 3K
Extensions for Python
shiro Install

Python Path 572K ★ 4
Python imports utils.
Mathias Gesbert Install

AREPL for python 557K ★ 5

Python v2022.20.1
Microsoft | 72,158,681 | ★★★★☆ (523)
IntelliSense (Pylance), Linting, Debugging (multi-threaded,...
Install

Python extension for Visual Studio Code

A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.7), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!

Support for vscode.dev

The Python extension does offer some support when running on vscode.dev (which includes github.dev). This includes partial IntelliSense for open files in the editor.

Installed extensio

The Python extension will
Pylance, Jupyter and isort

Categories
Programming Languages
Linters
Debuggers
Formatters
Data Science
Machine Learning
Notebooks

Extension Resources
Marketplace
Repository

Got a moment to help the VS Code team? Please tell us about your experience with VS Code so far.

⊗ 0 ⚠ 0

Packages

A Python **package** is a collection of function, data, and documentation that extends the capabilities of Python.

```python
import numpy as np
```

You will see statements like the above at the start of many Python code scripts.

- Instructions to use an installed package and give it a shortened name in the rest of the script.
- We will see more on how to install and use packages soon.

Recap

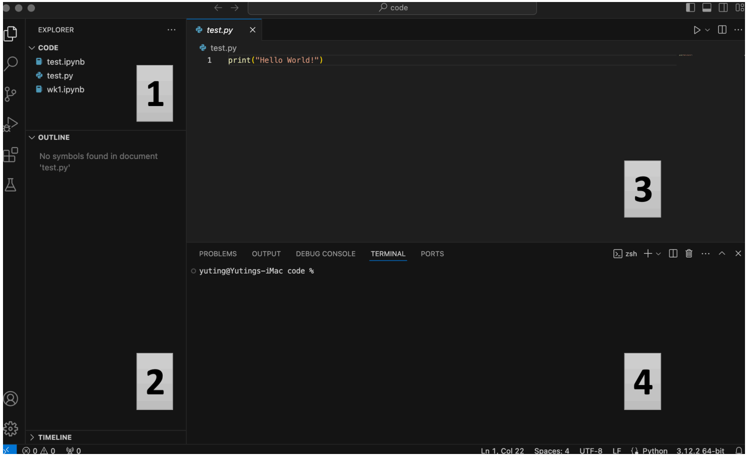
1. Install Python
    - Go to Python.org and download the latest version for your operating system.
2. Install Visual Studio Code (VSCode)
    - Visual Studio Code is a free and open-source IDE from Microsoft that is available on all major operating systems.
    - Download and install VS Code.
    - After that, we also need to install several essential VScode Python extensions.

Typical workflow

The **typical workflow** for analysis with code might be something like this:

- Open up your IDE.
- Write some code in a script in your IDE.
- If necessary for the analysis that you are doing, install some packages.
- Use the IDE to execute the code by Python and the add-on packages, and to display the results.
- Ensure the script can be run from top to bottom to reproduce your analysis.

VSCode interface



```
print("Hello World!")
```

```
yuting@Yutings-iMac code %
```

VSCode interface

1. The Explorer, showing the content of the folder that's currently open.
2. The Outline of key parts of the file that is open in Panel 3.
3. A VS Code Editor, showing a Python script (with an extension of `.py`).
   - Shortly, we can select code and press `Shift + Enter` to execute the code.
   - The result will appear in Panel 4.
4. The Terminal, a place where you can type in commands and your computer will execute.

Running Python code

Running Python code

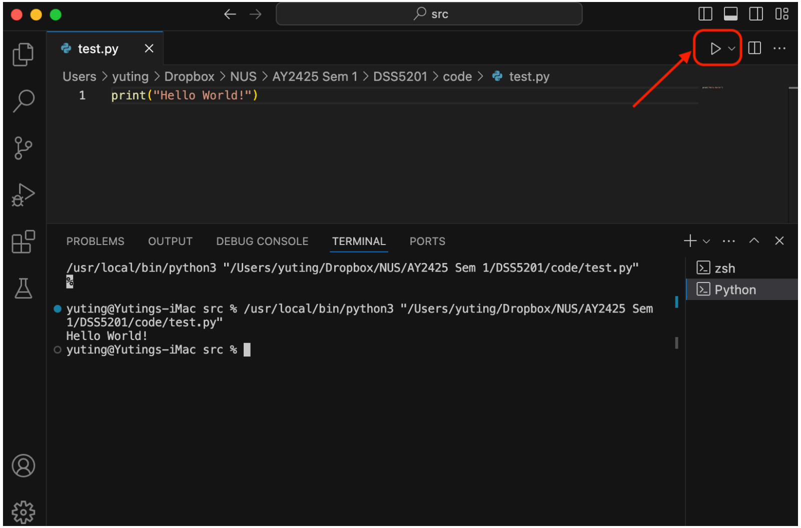
Now we will create and run our first code.

- Go to *File* > *New Text File* or use the keyboard shortcut (*Cmd* + *N*) to create a new file.
- Write a simple print expression to display “Hello World!”.

```
print("Hello World!")
```

- Save the file using *Cmd* + *S*. Select the file directory and type the file name.
  - Make sure to add `.py` at the end of the file name.
- To run the Python file, click on the *Run* button on the top right.
  - The output will display in the terminal.

Running Python code


src
test.py
Users > yuting > Dropbox > NUS > AY2425 Sem 1 > DSS5201 > code > test.py
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh
Python

```python
print("Hello World!")
```

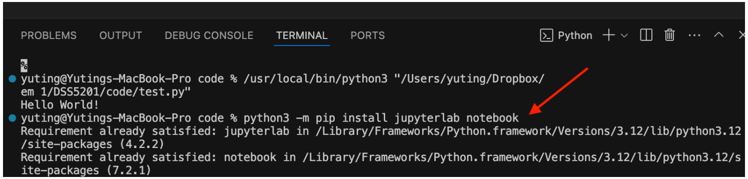
```
/usr/local/bin/python3 "/Users/yuting/Dropbox/NUS/AY2425 Sem 1/DSS5201/code/test.py"
%

yuting@Yutings-iMac src % /usr/local/bin/python3 "/Users/yuting/Dropbox/NUS/AY2425 Sem
1/DSS5201/code/test.py"
Hello World!
yuting@Yutings-iMac src %
```

Jupyter Notebook in VSCode

VSCode allows users to run code in Jupyter Notebook.

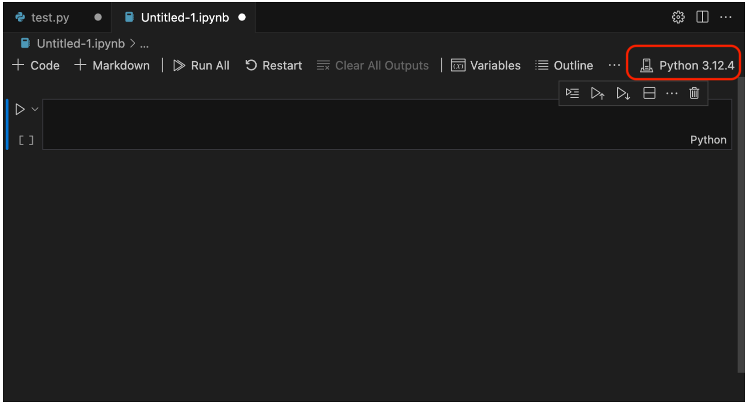
- That is, we can run the cell and visualize the result within VSCode.
- To use it, we need to install a few packages in the **Terminal**.

```
# For MacOS
python3 -m pip install jupyterlab notebook
# For Windows
py -m pip install jupyterlab notebook
```



```
%
yuting@Yutings-MacBook-Pro code % /usr/local/bin/python3 "/Users/yuting/Dropbox/
em 1/DSS5201/code/test.py"
Hello World!
yuting@Yutings-MacBook-Pro code % python3 -m pip install jupyterlab notebook
Requirement already satisfied: jupyterlab in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12
/site-packages (4.2.2)
Requirement already satisfied: notebook in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/s
ite-packages (7.2.1)
```

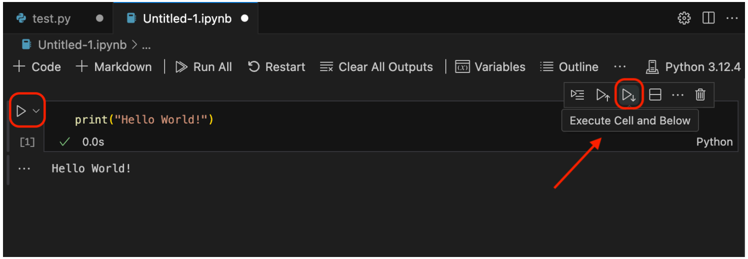
- Create a new file with `.ipynb` extension.
- Click on the *kernel picker* on the top right and select the Python version.



Jupyter Notebook in VSCode

Write a print expression to display “Hello World!” and press the run button.

- Run current cell.
- Run current cell and below.
- Run above cells.



```
print("Hello World!")
```

```
0.0s
```

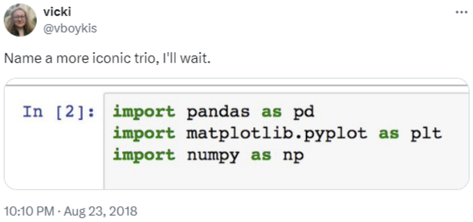
```
Hello World!
```

Pacakges

Packages (also known as **libraries**) are key to extending the functionality of Python.

- Written by core maintainers of the Python language, as well as enthusiasts, firms, researchers, academics, etc.
    - Because everyone can write packages, they vary widely in quality and usefulness.
- Today we will install a three key packages for data science workflow.
    - `numpy`, `pandas`, and `matplotlib`.

Packages


vicki
@vboykis

Name a more iconic trio, I'll wait.

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
```

10:10 PM · Aug 23, 2018

These three packages cover numerical, data, and plotting functionality.

- We need to install them (just once, like installing any other applications) in the Terminal.
- Then import them into our script.

Installing packages

On MacOS, here's how to install the three packages in the Terminal:

```
python3 -m pip install numpy pandas matplotlib
```

Here's how to install them on Windows:

```
py -m pip install numpy pandas matplotlib
```

Importing packages

After installation, we are able to load the packages in the notebook:

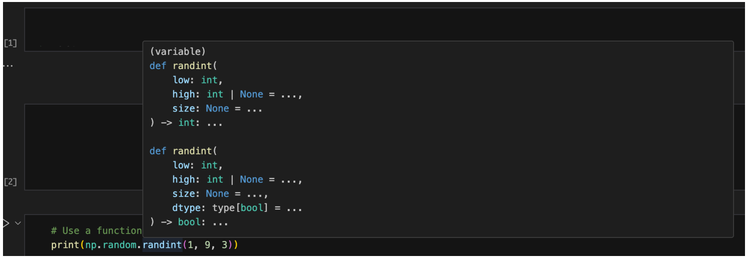
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

The code below uses `numpy` function to print three random integers between 1 and 9:

```
print(np.random.randint(1, 9, 3))
```

```
[1 2 4]
```

If you are ever unsure about a function on VSCode, hover your mouse over the function name to read its documentation.

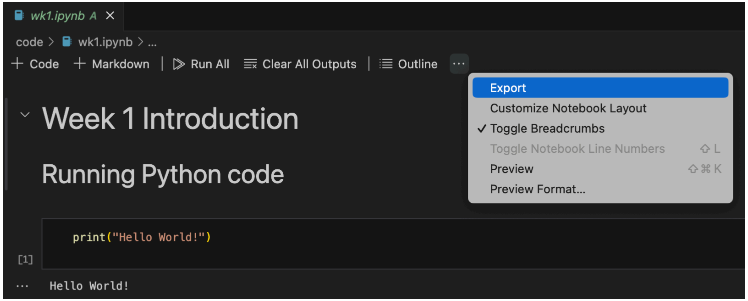


```
[1]
...
[2]

(variable)
def randint(
    low: int,
    high: int | None = ...,
    size: None = ...
) -> int: ...

def randint(
    low: int,
    high: int | None = ...,
    size: None = ...,
    dtype: type[bool] = ...
) -> bool: ...

# Use a function
print(np.random.randint(1, 9, 3))
```

Exporting a Jupyter Notebook

We can export a Jupyter Notebook as a Python script (`.py`) or an HTML file.

- Select `...` > `Export` on the main tool bar.
- Then select from a drop-down list for file format options.
- If encounter issues while exporting, go to system Terminal and install `nbconvert`.



Week 1 Introduction

Running Python code

```
print("Hello World!")
```

```
Hello World!
```

General guidelines

Lectures for this course will be highly **interactive** and requires **lots of learning-by-doing**.

- We recommend that you create a notebook to following along on your own laptop.
- As the lecture progresses, type the code from the lecture notes in your Code Editor.
- Run the code to see the outputs.

Setting up Python with VSCode

1. Install and configure VSCode for Python development, by Microsoft.

   https://learn.microsoft.com/en-us/training/modules/python-install-vscode/1-introduction

2. Setting Up VSCode For Python: A Complete Guide, by DataCamp.

   https://www.datacamp.com/tutorial/setting-up-vscode-python

3. Working with Jupyter Notebook in VS Code, by VS Code.

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

4. Python interactive window, by VS Code.

   https://code.visualstudio.com/docs/python/jupyter-support-py

Python basics

- Basic Python data types
- Lists and tuples
- Dictionaries
- Conditionals and repeated executions

Python objects

When working in Python, it is useful to remember two principles:

- Everything in Python is an object.
- Every object has a type (class).

Basically, Python uses an object-oriented style of programming.

Understanding the type of an object helps determine what methods and attributes are available for that object, and how it will behave under different conditions.

Naming objects

When working in Python, it is necessary to create new objects, either to store data or the output of functions.

- Remember to avoid the reserved and predefined words/letters.
- Examples include:

```
# Some reserved words in Python
False  True   else     import  from
None   break  except   in      is
def    pass   continue return
```

- Also, `Python` is case sensitive: `avg` and `Avg` refer to different objects.

Naming objects: Practical tips

snake_case

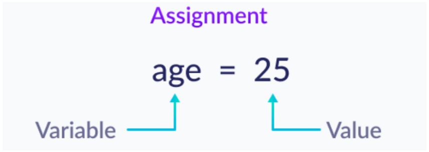
when all the words in a compound word are lower case but delimited by an underscore

1. Use descriptive names.
   - Choose names that describe the purpose of the variable or function.
   - Example: Instead of `x`, use `temperature` or age.
2. Use underscores for readability.
   - Use **underscores** to separate words in names for better readability.
   - **Explicit** is better than implicit.
   - Example: Use `avg_temperature` rather than `avgtemperature`.

Assignment operator

It can be helpful to think of a variable as a box that stores information:

- A single number, a vector, a string, etc.
- We use the **assignment operator** (=) to assign a value to a variable.



Common data types

In Python, variables can be of different types. Throughout your work, you will encounter several **common data types**:

- **Numeric (float)**: numbers that contain a decimal.
- **Integers (int)**: whole numbers.
- **Logical (bool)**: data that take on the value of either `TRUE` or `FALSE`.
- **String (str)**: data that are used to represent string values.
    - A special type of character string is called a **category** in `pandas`.

We can check the type of any object using the `type()` function.

```python
# Assign a floating-point number to variable 'a'
a = 2.7
type(a)
```

```
<class 'float'>
```

```python
# Assign a string to variable 'b'
b = "hello"
type(b)
```

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

```python
# Assign a boolean value to variable 'c'
c = True
# Check if 'c' is an instance of 'bool'
isinstance(c, bool)
```

```
True
```

Coercion functions

- Use the `type()` function to find out the type of an object.
- Use the following functions to conduct logical test and coerce objects.

Type	Checking	Coercing
numeric	`isinstance(var, (int, float))`	`float(var)` or `int(var)`
logical	`isinstance(var, bool)`	`bool(var)`
character	`isinstance(var, str)`	`str(var)`
categorical	`isinstance(var, pd.Categorical)`	`pd.Categorical(var)`

Arithmetic operations

Arithmetic operations are performed element by element.

- All of the basic operators in mathematics are available to use.

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
//	integer division
%	modulo

Examples

Let's apply these operators to **numeric** types:

```
# addition
1 + 2 + 3
```

```
6
```

```
# multiplication
2 * 3.1415926
```

```
6.2831852
```

```
# exponentiation
3 ** 2
```

```
9
```

Examples

Division, integer division, and modulo:

```python
# division
100 / 2
```

```
50.0
```

```python
# integer division (always rounds down)
100 // 3
```

```
33
```

```python
# modulo (remainder after division)
100 % 3
```

```
1
```

NoneType

NoneType is a unique data type in Python.

- Represents an object with no value.

```
# Create a NoneType object
x = None
print(x)
```

```
None
```

```
type(x)
```

```
<class 'NoneType'>
```

Strings

We can write strings as characters enclosed with either single quotes or double quotes.

- If the string contains a quotation or apostrophe, we can use a combination of single and double quotes to define it.

```python
# A string with apostrophe
str1 = "It is a rainy day."
print(str1)
```

```
It is a rainy day.
```

```python
# A string with quotation
str2 = 'Rosa: "Hello."'
print(str2)
```

```
Rosa: "Hello."
```

String methods

There are various useful **string manipulation methods** in Python.

```
# Convert into lower case
str2 = str1.lower()
print(str2)
```

```
it is a rainy day.
```

```
# Split strings into words
str3 = str1.split()
print(str3)
```

```
['It', 'is', 'a', 'rainy', 'day.']
```

Boolean

The boolean (`bool`) has two values: `True` and `False`.

- If we compare objects using **comparison operators**, we will get a boolean result.

Operator	**Description**
`x == y`	is `x` equal to `y`?
`x != y`	is `x` not equal to `y`?
`x > y`	is `x` greater than `y`?
`x >= y`	is `x` greater or equal to `y`?
`x < y`	is `x` smaller than `y`?
`x <= y`	is `x` smaller or equal to `y`?
`x is y`	is `x` the same object as `y`?

Boolean

We also have so-called **boolean operators**, which also returns either `True` or `False`.

Operator	Description
`x and y`	are `x` and `y` both True?
`x or y`	Is at least one of `x` and `y` True?
`not x`	is `x` False?

```
True and True
```

```
True
```

```
True or False
```

```
True
```

List and tuples

Lists and **tuples** allows us to store multiple elements in a single object.

- List are defined with square brackets `[]`.

```
# Create a list of length 3, containing values 1, 2, 3
list1 = [1, 2, 3]
print(list1)
```

```
[1, 2, 3]
```

- Now your environment contains the object `list1`. Go and click on the **Variables** tab. You should see it there.

List

A list can contain values of mixed types.

```python
# Create a list with mixed data types
list2 = [1, "hello", True, 3.14]
print(list2)
```

```
[1, 'hello', True, 3.14]
```

```python
# Length of the list
len(list2)
```

```
4
```

Tuples

Tuples look similar to lists, but have a key difference.

- They are defined with parentheses `()`.

```python
# Create a tuple with mixed data types
tuple1 = (1, "hello", True, 3.14)
print(tuple1)
```

```
(1, 'hello', True, 3.14)
```

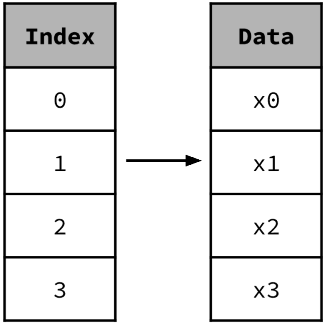
```python
type(tuple1)
```

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

Indexing and slicing

We can access values inside a list or tuple using square-bracket syntax.

- Python uses **zero-based indexing**: The first element of the list is in position 0.



Indexing and slicing

- Python uses **zero-based indexing**: The first element of the list is in position 0.

P	**y**	**t**	**h**	**o**	**n**
0	**1**	**2**	**3**	**4**	**5**

1	**hello**	**True**	**3.14**
0	**1**	**2**	**3**

```python
# The first element (index starts from 0)
list2[0]
```

```
1
```

```python
# The last element (with index -1)
list2[-1]
```

```
3.14
```

```python
# Modify the second element
list2[1] = 10
list2
```

```
[1, 10, True, 3.14]
```

Slicing

Now `list2` becomes:

1	**10**	**True**	**3.14**
0	1	2	3

We can use the colon (`:`) to access a sub-sequence.

- This is called **slicing**.
- Note that the ending index is **not inclusive**.

```
# Slice the first three elements
list2[0:3]
```

```
[1, 10, True]
```

List methods

Other list methods to perform operations with the data.

- Defined on the object itself and accessed using a period (`.`).

```python
# Add element to the end of the list
list2.append("one more entry")
list2
```

```
[1, 10, True, 3.14, 'one more entry']
```

```python
# Remove the second element
del list2[1]
list2
```

```
[1, True, 3.14, 'one more entry']
```

Dictionaries

A **dictionary** is a mapping between **key and value** pairs.

- The key is a unique identifier; the value can be of any data type.
- Defined with curly brackets.

```
# Define a dictionary
dict1 = {
    "A": 2024,
    "B": ["Stats", "Data Science"],
    "C": [True, False, True]
}
```

Dictionaries

```python
# Check the type of the dictionary
type(dict1)
```

```
<class 'dict'>
```

```python
# Display the dictionary
dict1
```

```
{'A': 2024, 'B': ['Stats', 'Data Science'], 'C': [True, False, True]}
```

Accessing components in dictionaries

```python
# Access the keys of the dictionary
dict1.keys()
```

```
dict_keys(['A', 'B', 'C'])
```

```python
# Access the values associated with the key "B"
dict1["B"]
```

```
['Stats', 'Data Science']
```

Accessing components in dictionaries

```python
# Modify an existing value
dict1["A"] = 5201
dict1
```

```
{'A': 5201, 'B': ['Stats', 'Data Science'], 'C': [True, False, True]}
```

```python
# Delet a key-value pair
del dict1["C"]
dict1
```

```
{'A': 5201, 'B': ['Stats', 'Data Science']}
```

More on brackets

We've introduced several types of brackets. They can play different roles depending on the context.

The most common uses are:

- `[]` is used to denote a list, or to signify accessing a position with an index.
    - `vec[0]` gets the first entry of a variable named `vec`.
- `{}` is used to denote a set or a dictionary.
    - `{"first_name": "a", "last_name": "b"}`.
- `()` is used to denote arguments of a function.
    - `type(x)` where `x` is the input passed the the function `type()`.

Check objects in the workspace

In Python, we can list the objects currently defined in the environment.

- We can inspect them in the **Variables** tab.

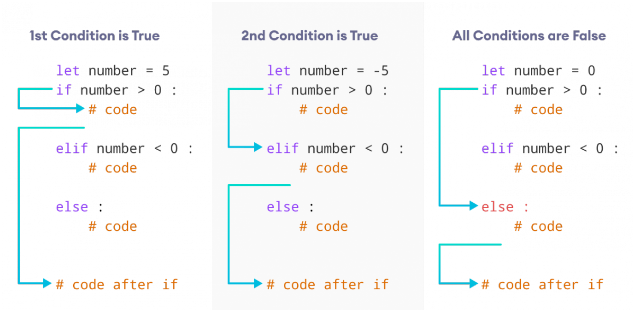
As we work, we may find that you accumulate a number of objects within the workspace.

- We can remove them with the `del` statement.

```
# Remove objects in environment
del a, b, c, list1, list2, str1, str2, str3, dict1
```

Conditional executions and **loops** are essential for controlling the flow of a program.

- The `if` statement is used to execute a block of code only if a specified condition is true.
- Optionally, we can use an `else` block to execute code if the condition is false.
- Additionally, we can use `elif` (short for else if) to check multiple conditions sequentially.



```
let number = 5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code

# code after if
```

```
let number = -5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code

# code after if
```

```
let number = 0
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code

# code after if
```

Example

Here is a simple example:

```
w = 10

if w > 0:
    w1 = 2
else:
    w1 = -2

print(w1)
```

```
2
```

Certain part of the code examples are **indented by 4 empty spaces**.

- Code that is part of a function, a conditional clause, or a loop is indented.
- **Not** a code-style choice, but a way to tell Python that some code is to be executed as part of, say, a loop, and not to be executed after the loop is finished.
- Consistent indentation is crucial.

Example without else

- If we don't need the `else` block, we can omit it:

```python
w = 10

if w > 0:
    w1 = 2

print(w1)
```

```
2
```

Example with elif

We can use `elif` to check multiple conditions.

```python
w = 0

if w > 0:
    w1 = 2
elif w == 0:
    w1 = 0
else:
    w1 = -2

print(w1)
```

```
0
```

Example: A dice game

Suppose that we wish to simulate a game of dice between two players, A and B.

- They each have a fair six-sided die. The player with the higher roll is the winner. If they both get the same number, the result is a draw.

```python
# Simulate a single roll of a six-sided die
roll = np.random.randint(1, 7)
print(roll)
```

```
4
```

- If they wish to roll two dice, then

```python
# Simulate two rolls of a six-sided die
roll = np.random.randint(1, 7, size = 2)
print(roll)
```

```
[5 3]
```

Example: A dice game

Now let's simulate the dice game – roll the dice for both players and determine the winner.

Step-by-step implementation:

1. Set the random seed to ensure reproducibility.
2. Simulate the dice rolls for both players.
3. Compare the results and determine the winner using conditional statements.

Example: A dice game

```python
# Set seed for reproducibility
np.random.seed(5201)

# Simulate rolling a die for players A and B
A = np.random.randint(1, 7)
B = np.random.randint(1, 7)

# Compare the result and determine the winner
if A > B:
    print("A is the winner.")
elif A == B:
    print("It is a draw.")
else:
    print("B is the winner.")
```

```
B is the winner.
```

Repeated executions

Now suppose we want to simulate the game for 1000 times.

We can use a **`for` loop**. For each iteration,

1. Roll the dice for both players.
2. Determine the winner.
3. Store the result in a list named `results`.

```python
np.random.seed(5201)       # For reproducibility
results = [0] * 1000       # Initiate a list to store results

for i in range(1000):
    A = np.random.randint(1, 7)
    B = np.random.randint(1, 7)

    if A > B:
        results[i] = "A"
    elif A == B:
        results[i] = "Draw"
    else:
        results[i] = "B"

# Print the first 5 results
print(results[:5])
```

```
['B', 'A', 'A', 'B', 'B']
```

Repeated executions

To summarize the results:

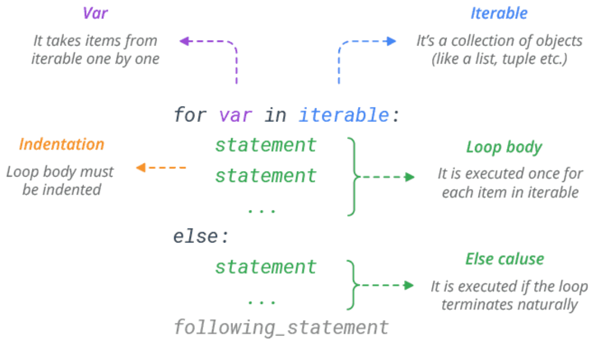
```
# Import a function from a built-in module
from collections import Counter

# Summarize the results
results_counter = Counter(results)
print(results_counter)
```

```
Counter({'A': 429, 'B': 413, 'Draw': 158})
```

For loop

What we just saw is a `for` loop.

- We specify what has to be done and how many times.


Var
It takes items from iterable one by one
Iterable
It's a collection of objects (like a list, tuple etc.)
Indentation
Loop body must be indented
Loop body
It is executed once for each item in iterable
Else caluse
It is executed if the loop terminates naturally

```
for var in iterable:
    statement
    statement
    ...
else:
    statement
    ...
following_statement
```

While loop

What if we want to stop the game as soon as A wins the first time?

- In this case, we use a `while` loop with a `break` statement, to break out of the loop.
- It will continue to execute until a specific condition is met – in this case, until player A wins.

Example: First win for player A

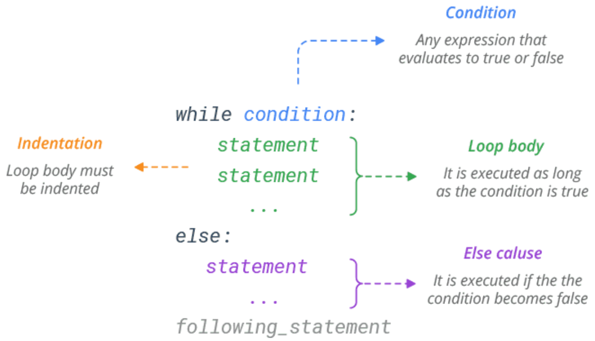
```python
np.random.seed(5201) # For reproducibility

# Initialize variables
iterations = 0

while True:
    iterations += 1
    A = np.random.randint(1, 7)
    B = np.random.randint(1, 7)

    if A > B:
        break

print("First win for A occurs at game", iterations)
```

```
First win for A occurs at game 2
```

While loop

What we just saw in the outer loop is a `while` loop.

- It is used when we want to perform a task indefinitely, **until** a certain condition is met.



```
while condition:
    statement
    statement
    ...
else:
    statement
    ...
following_statement
```

Condition
Any expression that evaluates to true or false
Indentation
Loop body must be indented
Loop body
It is executed as long as the condition is true
Else caluse
It is executed if the the condition becomes false

Summary: Python basics

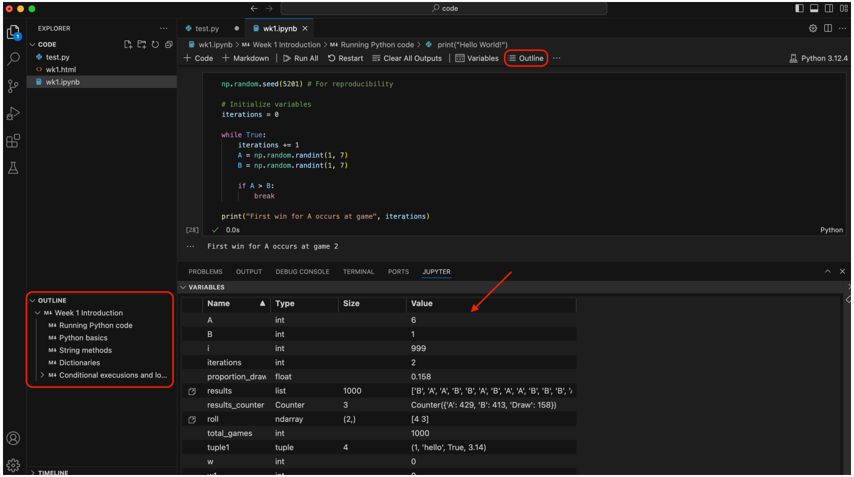
Basics in Python programming

- Basic Python data types
- List and tuples
- Dictionaries
- Conditionals and repeated executions

Assignment:

- Pre-course reflections on Canvas by **Friday of Week 2 11:59pm**.

Summary: VScode interface



OUTLINE
- Week 1 Introduction
  - Running Python code
  - Python basics
  - String methods
  - Dictionaries
  - Conditional executions and lo...

```
np.random.seed(5201) # For reproducibility

# Initialize variables
iterations = 0

while True:
    iterations += 1
    A = np.random.randint(1, 7)
    B = np.random.randint(1, 7)

    if A > B:
        break

print("First win for A occurs at game", iterations)
```

[28] ✓ 0.0s Python

```
First win for A occurs at game 2
```

Name	Type	Size	Value
A	int		6
B	int		1
i	int		999
iterations	int		2
proportion_draw	float		0.158
results	list	1000	['B', 'A', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', '
results_counter	Counter	3	Counter({'A': 429, 'B': 413, 'Draw': 158})
roll	ndarray	(2,)	[4 3]
total_games	int		1000
tuple1	tuple	4	(1, 'hello', True, 3.14)
w	int		0
w1	int		0