Python

Physics 129AL

Zihang Wang
10/10/2023

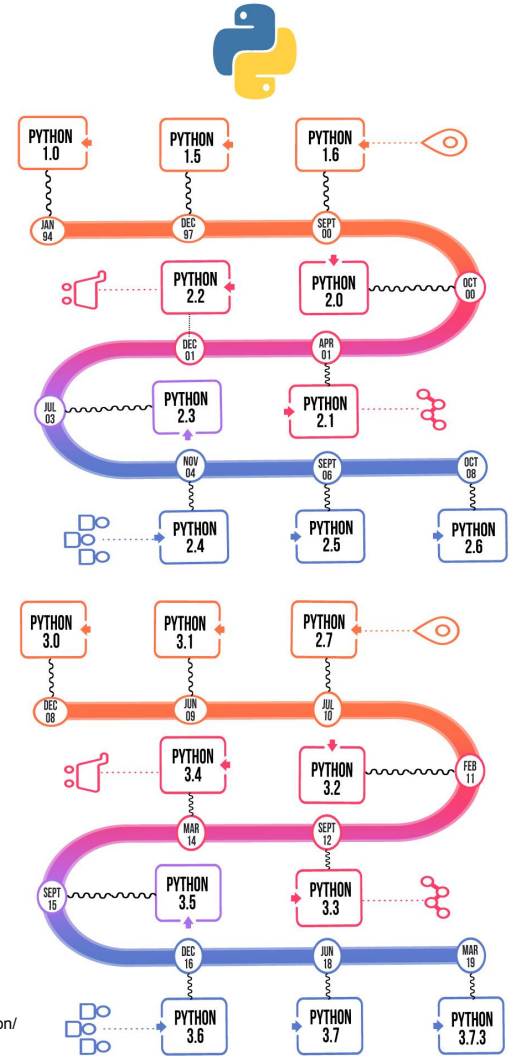
History of Python

Guido van Rossum

Logo: 1990s–2006

Logo: 1990s–2006

https://gvanrossum.github.io/

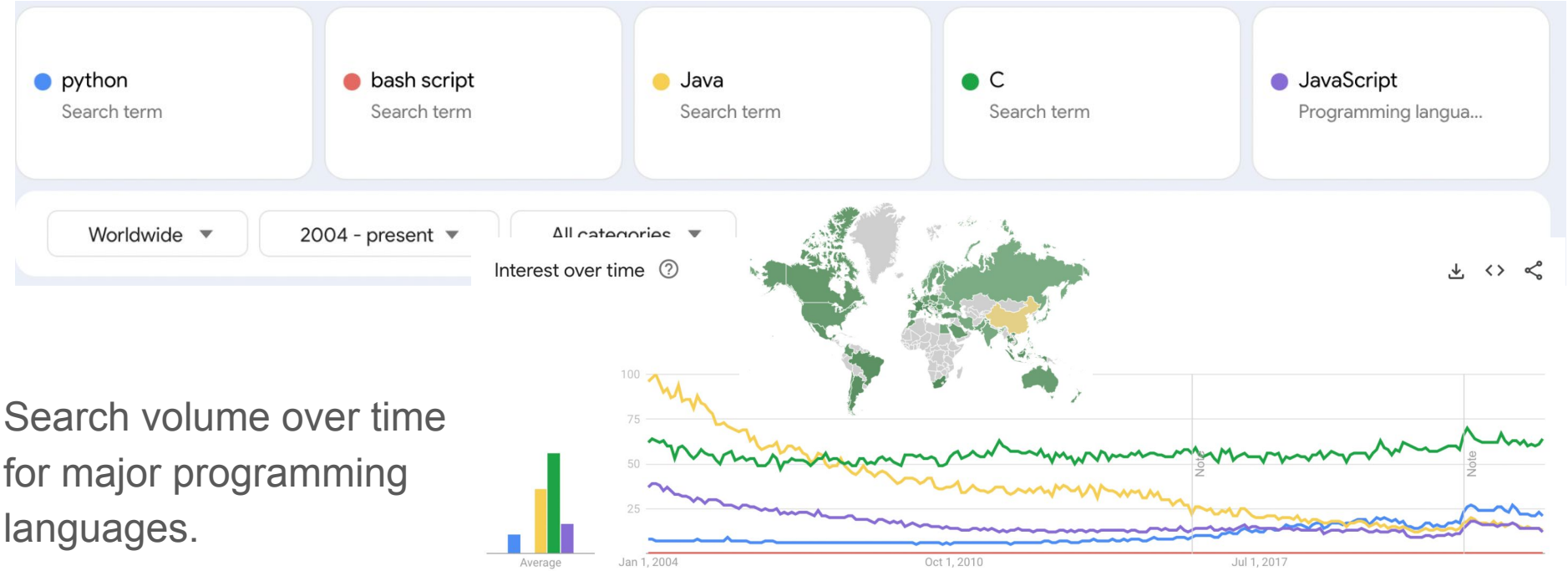


- An easy and intuitive language
- **Open source**, so anyone can contribute to its development Code
- English Suitability for everyday tasks

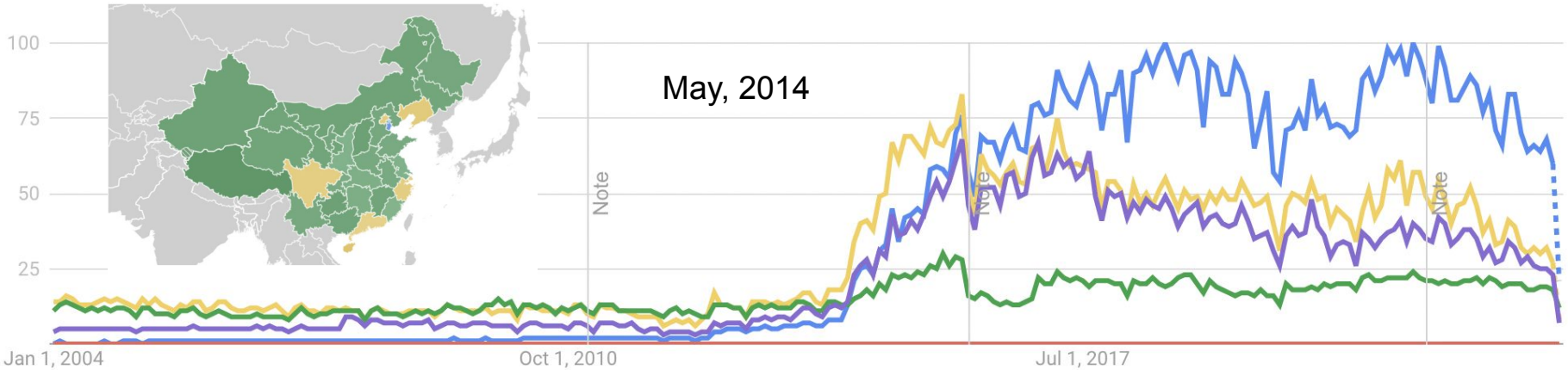
https://www.geeksforgeeks.org/history-of-python/

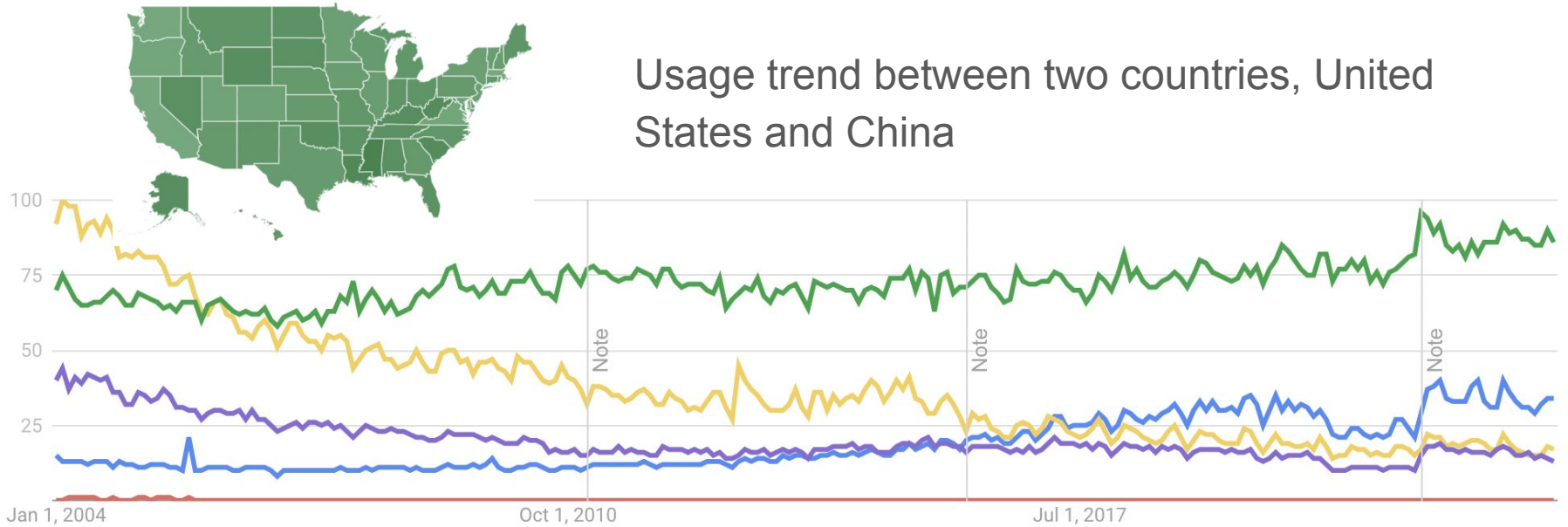
Trends on the Internet

A very useful tool for checking internet search trend. https://trends.google.com/trends



Search volume over time for major programming languages.


May, 2014
100
75
50
25
Note
Note
Note
Jan 1, 2004
Oct 1, 2010
Jul 1, 2017


Usage trend between two countries, United States and China
100
75
50
25
Note
Note
Note
Jan 1, 2004
Oct 1, 2010
Jul 1, 2017

The Nature of Python

It is a scripting language that supports object-oriented programming and structured programming.

Object-oriented programming typically use **inheritance** for code reuse and extensibility.

Structured programming support for block structures.

Data Type	Description
int	Integer type
float	Floating-point type
bool	Boolean type
str	Text type (String)
list	Mutable sequence
tuple	Immutable sequence
set	Mutable set
frozenset	Immutable set
dict	Dictionary (Key-Value pairs)
None	Null value
bytes	Immutable sequence of bytes
bytearray	Mutable sequence of bytes
type	Type object
Custom Classes	User-defined data types

Table 1: Common Data Types in Python

Python Environment

- Python Interpreter: responsible for executing Python code.

  Retired 2020 ⟹ Python 2.x or Python 3.x

- Package Manager: manage external libraries and packages.

  pip (Python's package manager) and conda (a package manager commonly used with Anaconda)

- Virtual Environment: prevent conflicts between packages by keeping each project in its own environment.

  venv or conda

- Other Environments: such as Jupyter and docker.

Python Interpreter

- The default python Interpreter lives in the binary directory, **/usr/bin/python**

To run python, **/usr/bin/python your_script.py**, and the alias is **python your_script.py**

- If you want to choose different interpreter, you can enter the path,

For example, **/path/to/anaconda3/bin/python your_script.py,** or if you are in an active environment, you can directly use **python your_script.py.**

Python Environment

- A virtual environment in Python is a self-contained, isolated workspace where you can develop and run Python projects with their own sets of dependencies.

venv module: a python built-in module that creates virtual environments.

```
python3 -m venv name_of_env

source name_of_env/bin/activate
```


```
conda create --name myenv python=3.8

conda activate myenv
```

Python Package Manager

- Python package managers are tools and systems that facilitate the installation, management, and distribution of Python libraries and packages.

pip (Python's package manager):

For example: pip install numpy

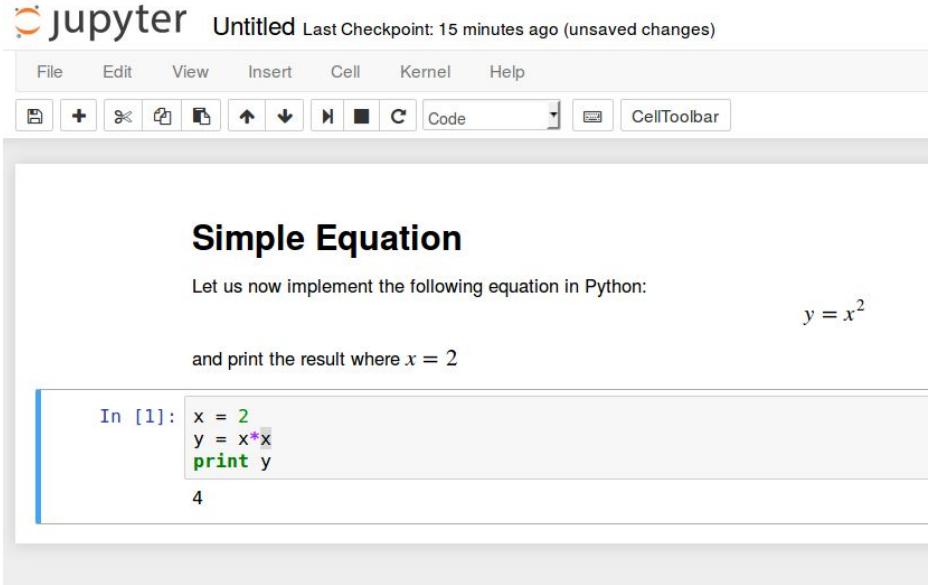
conda (used with Anaconda):

For example: conda install numpy

Popular packages includes, numpy, pandas, matplotlib, scipy, sympy, scikit-learn, pytorch, tensorflow…

Jupyter Notebook

- Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.



Fernando Pérez

Class and objects

In Python, a class is a blueprint or a template for creating objects (instances). Classes define the structure and behavior of objects, encapsulating data (attributes) and methods (functions) that operate on that data.

Term	Description
Class	A blueprint or template for creating objects.
Object (Instance)	A specific instance of a class that represents a concrete realization of the class's attributes and methods.
Attributes (Properties)	Variables that store data associated with a class, representing the characteristics or state of objects.
Methods	Functions defined within a class that define the behavior or actions that objects of the class can perform.
Constructor	A special method (__init__) called when an object is created from the class. It initializes the object's attributes.
Self	A reference to the current instance of the class, used within class methods to access and modify instance attributes.
Inheritance	The ability to create new classes based on existing classes. The new class inherits attributes and methods from the parent class, allowing code reuse and extension.

Table 2: Python Class Overview

Example: Particle type

Let's define a class that generates an instance of particle.

```
class Particle:
    def __init__(self, name, charge, mass):
        self.name = name  # Particle name (e.g., "Electron")
        self.charge = charge  # Electric charge (in Coulombs)
        self.mass = mass  # Mass (in kilograms)

    def describe(self):
        return f"{self.name}: Charge = {self.charge} C, Mass = {self.mass} kg"
```

Example: Particle type

We can the define electrons and protons (particle inheritance)

```
class Electron(Particle):
    def __init__(self):
        # Call the parent class constructor to initialize attributes
        super().__init__("Electron", -1.602e-19, 9.109e-31)  # Electron charge and mass

    def describe(self):
        return super().describe() + ", Charge -e"

class Proton(Particle):
    def __init__(self):
        # Call the parent class constructor to initialize attributes
        super().__init__("Proton", 1.602e-19, 1.673e-27)  # Proton charge and mass

    def describe(self):
        return super().describe() + ", Charge e"
```

Example: Particle type

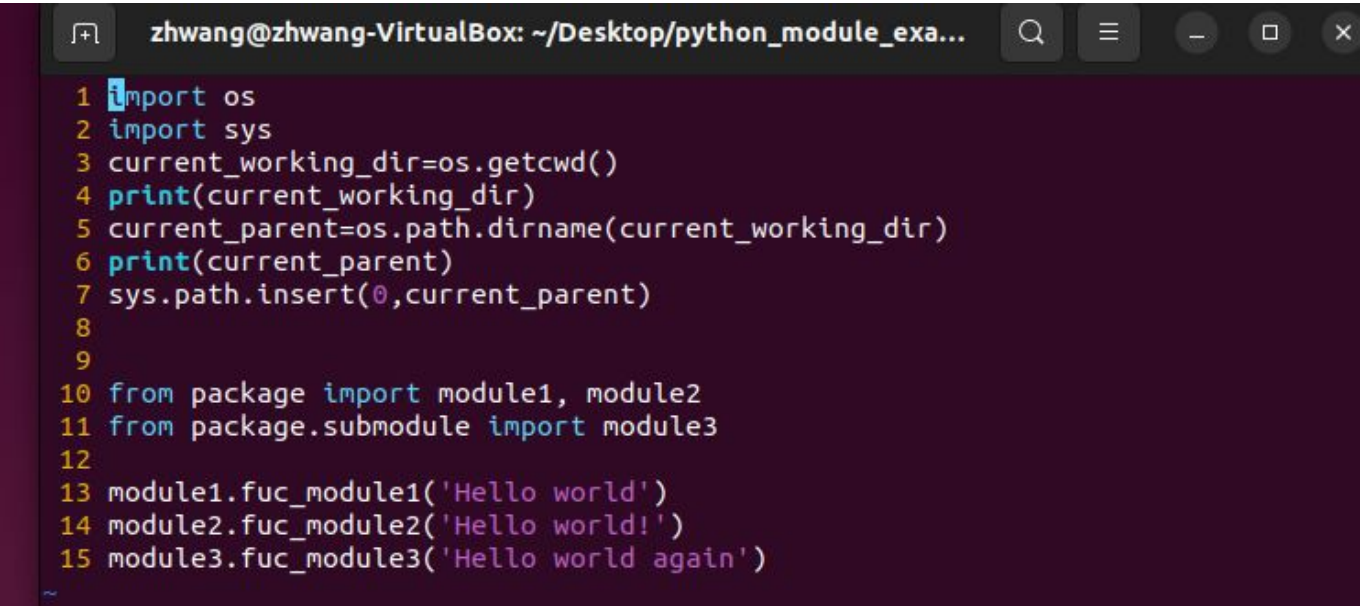
We then can create objects (instance of a class)

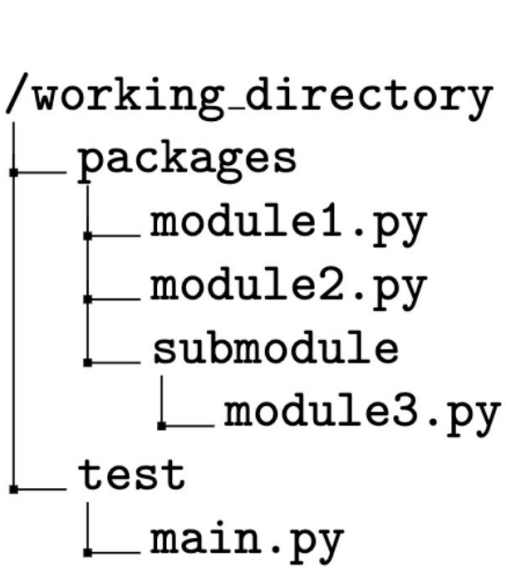
```
# Create instances of Electron and Proton
electron = Electron()
proton = Proton()

# Display information about the particles
print("Electron:")
print(electron.describe())

print("\nProton:")
print(proton.describe())
```

```
Electron:
Electron: Charge = -1.602e-19 C, Mass = 9.109e-31 kg, Charge -e

Proton:
Proton: Charge = 1.602e-19 C, Mass = 1.673e-27 kg, Charge e
```

Packaging and imports

It is convenient to have packages for large-scale projects.





Script Execution

When import from packages, keeping executable lines in the main.

```
def my_function():
    print("This function is defined in the script")

if __name__ == "__main__":
    print("This code runs only when the script is executed directly.")
    my_function()

~
```

```
from python_example import *
my_function()
~
~
```