

CS229

Python & Numpy

Angelica Sun

How does python relate to other languages?

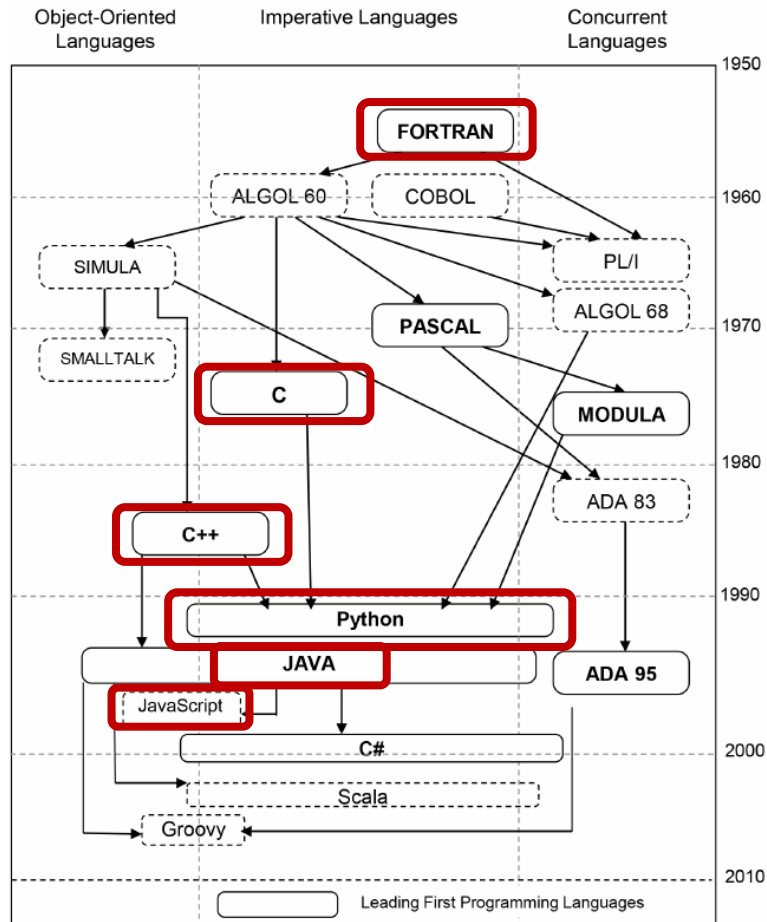
Python 2.0 released in 2000

(Python 2.7 “end-of-life” in 2020)

Python 3.0 released in 2008

(Python 3.6+ for CS 229)

- Interpreted, like MATLAB
- Object-oriented
- Dynamically-typed



Before you start

Always use **conda** for environment management

Create a new environment

```
conda create -n cs229 python=3.9
```

Create an environment (from configuration)

```
conda env create -f environment.yml
```

Activate an environment after creation

```
conda activate cs229
```

List existing environments

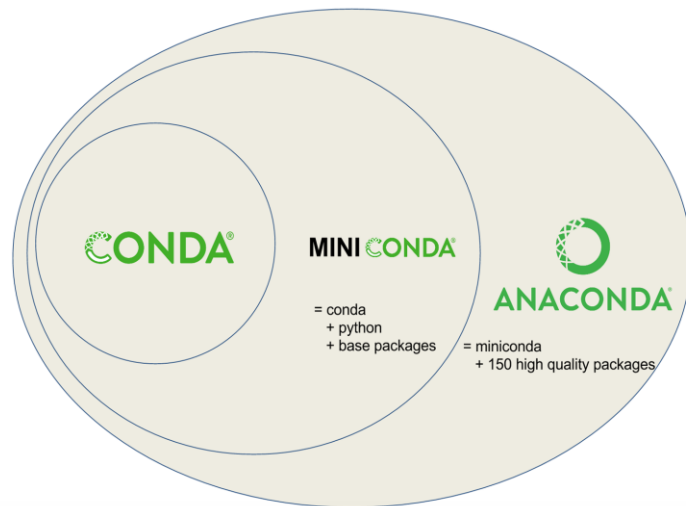
```
conda env list
```

Install a package in current environment

```
conda install PACKAGENAME (or pip)
```

More commands:

https://conda.io/projects/conda/en/latest/_downloads/843d9e0198f2a193a3484886fa28163c/conda-cheatsheet.pdf



Notepad is not your friend ...

Get a text editor/IDE

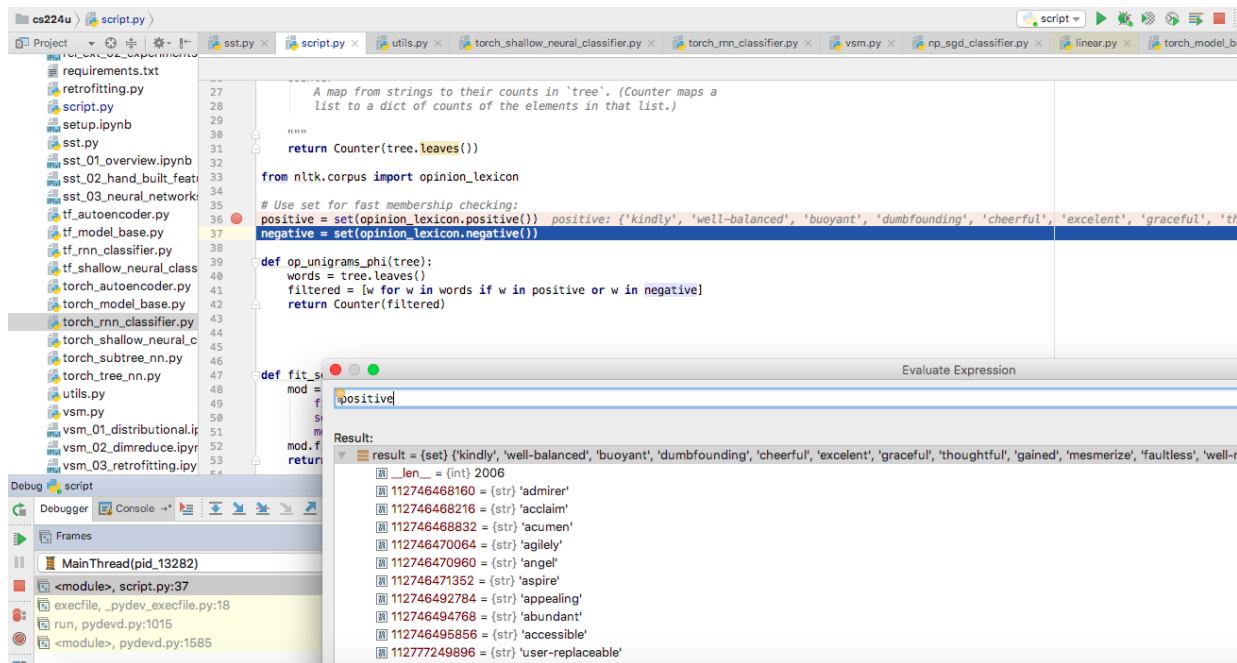
- PyCharm (IDE)
- Visual Studio Code (IDE??)
- Sublime Text (IDE??)
- Notepad ++/gedit
- Vim (for Linux)



To make you more prepared

PyCharm

- Great debugger
- Proper project management



FYI, professional version free for students: <https://www.jetbrains.com/student/>

To make you more prepared

Using PyCharm with conda

The image displays three PyCharm windows illustrating the setup of a Python environment using conda.

Settings - Python Interpreter

Project: featuremap.py > Python Interpreter For current project

Python Interpreter: Python 3.8 C:\Users\angel\anaconda3\python.exe

Package	Version	Latest version
_anaconda_depends	2020.07	
_ipyw_jlab_nb_ext_conf	0.1.0	
alabaster	0.7.12	
alembic	1.5.5	
anaconda	custom	
anaconda-client	1.7.2	
anaconda-navigator	1.9.12	
anaconda-project	0.8.4	
aniso8601	8.1.1	
appdirs	1.4.4	
argh	0.26.2	
asn1crypto	1.3.0	
astroid	2.4.2	
astropy	4.0.1.post1	
atomicwrites	1.4.0	
attrs	19.3.0	
audioread	2.1.9	
autopep8	1.5.3	
babel	2.8.0	
backcall	0.2.0	
backports	1.0	
backports.functools_lru_cache	1.6.1	
backports.shutil_get_terminal_size	1.0.0	
backports.tempfile	1.0	
backports weakref	1.0 post1	

Python Interpreters

- Python 3.8 C:\Users\angel\anaconda3\python.exe
- Python 3.8 (pythonProject) C:\Users\angel\anaconda3\envs\pythonProject\python.exe
- Python 2.7 (AA274A_HW2) C:\Users\angel\anaconda3\envs\AA274A_HW2\python.exe
- Python 2.7 (python2.7) C:\Users\angel\anaconda3\envs\python2.7\python.exe

Add Python Interpreter

☒ Virtualenv Environment

☐ Conda Environment

☐ System Interpreter

☐ Pipenv Environment

☒ New environment

Location: E:\stanford\cs229\cs229-content\Exercises\src\featuremap

Base interpreter: C:\Users\angel\AppData\Local\Microsoft\Wind

☐ Inherit global site-packages

☐ Make available to all projects

☐ Existing environment

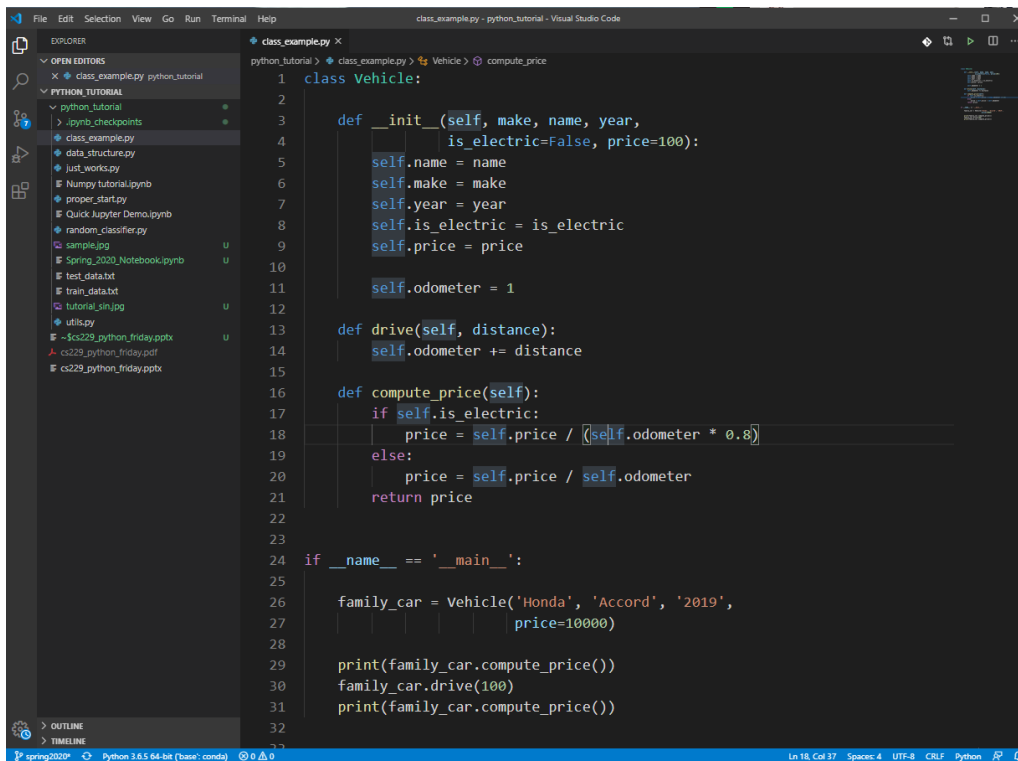
Interpreter: <No interpreter>

☐ Make available to all projects

To make you more prepared

Visual Studio Code

- Light weight
- Wide variety of plugins to enable support for all languages
- Better UI



The screenshot displays the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows a file tree for a project named 'python_tutorial', including files like 'class_example.py', 'data_structure.py', and 'utils.py'. The main editor window is open to 'class_example.py', showing a Python class named 'Vehicle'. The class has an '__init__' method that initializes attributes like 'name', 'make', 'year', 'is_electric', and 'price', and an 'odometer' attribute set to 1. It also includes a 'drive' method that increments the odometer and a 'compute_price' method that calculates the price based on whether the car is electric. At the bottom, a main block creates a 'family_car' instance and prints its computed price and after a drive.

```
1 class Vehicle:
2
3     def __init__(self, make, name, year,
4                 is_electric=False, price=100):
5         self.name = name
6         self.make = make
7         self.year = year
8         self.is_electric = is_electric
9         self.price = price
10
11         self.odometer = 1
12
13     def drive(self, distance):
14         self.odometer += distance
15
16     def compute_price(self):
17         if self.is_electric:
18             price = self.price / (self.odometer * 0.8)
19         else:
20             price = self.price / self.odometer
21         return price
22
23
24 if __name__ == '__main__':
25
26     family_car = Vehicle('Honda', 'Accord', '2019',
27                          price=10000)
28
29     print(family_car.compute_price())
30     family_car.drive(100)
31     print(family_car.compute_price())
32
```

Python


```
> featuremap.py
  main.py
  utils.py
> External Libraries
  Scratches and Consoles

1  import numpy                                # import package
2  from matplotlib import pyplot as plt        # import from package and rename
3  from utils import *                        # import from local file
4
5  var = 10                                    # variables don't need type declaration
6  anothervar = "Hello"                      # codes written at indentation level 0 always get executed
7  print("I always get executed!")            # even when imported
8
9  def print_hi(name):                        # define a function
10     print(f'Hi, {name}')
11
12  class Foo:                                # define a class
13     def __init__(self, x):                  # constructor
14         self.x = x                        # use first formal parameter (named "self" by convention) for self reference
15
16     def printX(self):                      # functions are instance by default
17         print(self.x)
18
19     @classmethod                            # use decorator to declare class methods
20     def printHello(self):
21         print("hello")
22
23  class Bar(Foo):                            # inherit a class
24     pass
25
26  if __name__ == '__main__':                # main function (doesn't execute on import)
27     print_hi('CS229')                      # call function
28     functionInUtils()                      # call imported function
29     obj = Bar(3)                           # create object
30     obj.PrintX()                          # call function using object
```

Live Demo in Jupyter Notebook

- Links:
- Notebook:
- <http://cs229.stanford.edu/notes2021spring/notes2021spring/cs229-python-review-code.ipynb>
- PDF Version:
- <http://cs229.stanford.edu/notes2021spring/notes2021spring/python-review-code.pdf>

Your friend for debugging

Python Command	Description
<code>array.shape</code>	Get shape of numpy array
<code>array.dtype</code>	Check data type of array (for precision, for weird behavior)
<code>type(stuff)</code>	Get type of a variable
<code>import pdb; pdb.set_trace()</code>	Set a breakpoint (https://docs.python.org/3/library/pdb.html)
<code>print(f'My name is {name}')</code>	Easy way to construct a message

Good luck on your
HW/Project!

Questions?