Machine Learning Development Environment

Xiang HAO

Department of Computing

xiang.hao@connect.poly.hk

Conda, PyCharm IDE, and Python

Conda and Pip

Package, dependency and environment management for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN.

- Conda is an open-source **package management system** and **environment management system** that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. Conda easily creates, saves, loads, and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language.
- Why not use Pip (package installer for Python)?
  - Just a package installer, not an environment manager.
  - Neglecting non-Python dependencies (e.g. HDF5, MKL, etc.).

```
# Use Conda
conda create --name ml biopython
conda install scipy

# Use Pip
pip install biopython
```

Anaconda, Miniconda, and Mamba

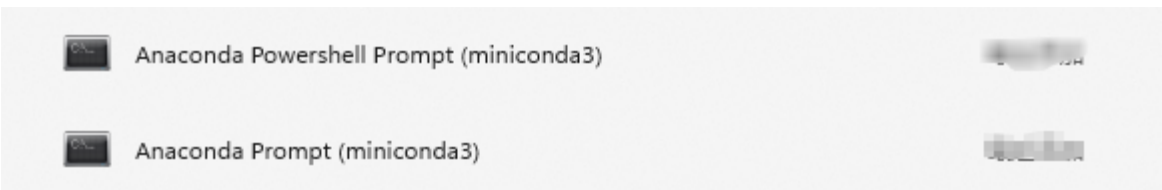
- **Anaconda** is a set of about a hundred packages including conda, numpy, scipy, ipython notebook, and so on. Use it if you:
  - Are new to conda or Python.
  - Like the convenience of having Python and over 150 scientific packages automatically installed at once.
  - Have the time and disk space (a few minutes and 3 GB), and/or.
  - Don't want to install each of the packages
- **Miniconda** is a much smaller distribution, it includes only conda, Python, the packages they depend on, and a small number of other useful packages, including `pip`, `zlib`, and a few others. Use it if you:
  - Do not mind installing each of the packages.
  - Do not have time or disk space to install over 150 packages at once.
  - Just want fast access to Python and the ability to easily install just the packages you want using conda or pip.
- **Mamba** is a re-implementation of the conda package manager in C++. (**Not** the Mamba model)
  - Parallel downloading of repository data.

Install Miniconda

Please download and install the latest Miniconda from https://docs.anaconda.com/free/miniconda/. Please **keep the default settings** during the installation process.

For MacOS: https://www.youtube.com/watch?v=K5IrMNwJf7I.

After installation, for Windows users, please use the **Anaconda Prompt** instead of the default Command Prompt. It adds the conda command to your `PATH` environment variable.



Check the installation by running `conda -V` or `conda info`.

```
(base) PS C:\Users\xhao> conda -V
conda xx.xx.xx

(base) PS C:\Users\xhao> conda info
...
```

Managing Environments with Conda

```
conda --version # or `conda -V`

conda update conda
```

Create a new environment with Python 3.10 and install some packages:

```
conda create --name ml python=3.10
# Proceed ([y]/n)? y

conda activate ml

python --version

# Python 3.10.14
```

List all environments that you have created:

```
conda env list

conda environments:
base                     C:\Users\xhao\miniconda3
ml                    *  C:\Users\xhao\miniconda3\envs\ml
```

Managing Packages with Conda

Manage packages in an active environment.

```
python --version

# Install a new package
conda install numpy

# List installed packages
conda list

# Delete a package
conda remove numpy

# Deactivate the environment to switch to another one
conda deactivate
```

Notes:

- Some dependencies should be installed automatically, e.g. MKL for numpy (accelerated linear algebra computation)
- Conda cheat sheet: https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html

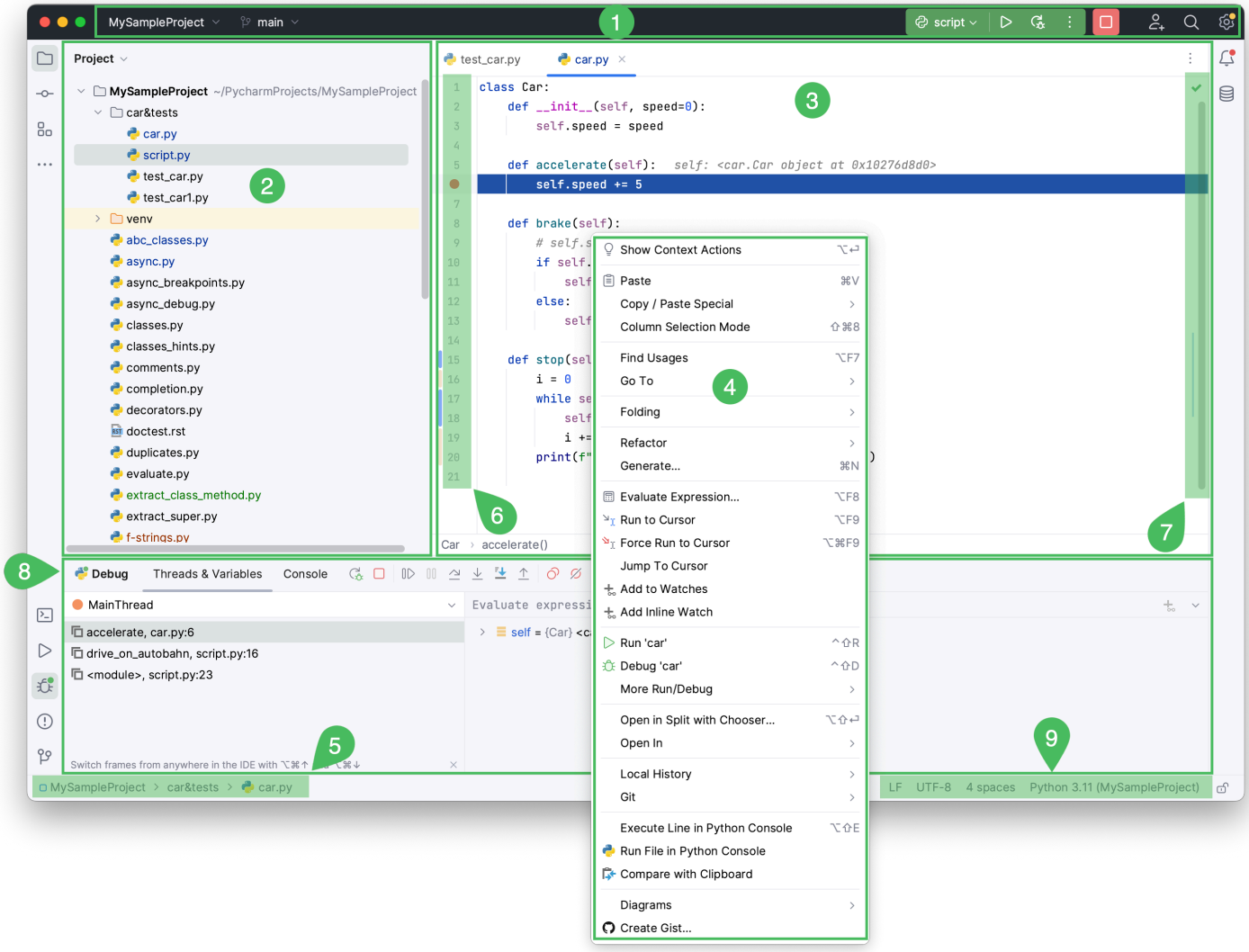
PyCharm IDE

IDE (Integrated Development Environment) for Python, developed by JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda.

- Why not use a text editor (e.g. VS Code)?
  - No code analysis (`python-language-server`)
  - No debugger (`python debugger`)
  - No formatter, refactoring and linter (`pylance` , `pylint` , `ruff` , and `black`)
  - No scientific tools integration (`jupyter`)
- PyCharm is **easy for beginners**
  - IDE for Python development and Web development
  - Community edition is free
  - Professional edition is free for students by Github Student license (web, remote, database, scientific, and mobile development)
  - Coding assistance, code inspection, refactoring, debugging, and testing

Install PyCharm

- Download the installer from https://www.jetbrains.com/pycharm/download/
- Run the installer
- Try free for 30 days
- Create a new project
- Set the project name and location
- Select the project interpreter
- Create a new file and start coding
- Run the code `print("Hello, World!")`


MySampleProject
main
script
1
Project
MySampleProject ~/PycharmProjects/MySampleProject
car&tests
car.py
script.py
test_car.py
test_car1.py
2
venv
abc_classes.py
async.py
async_breakpoints.py
async_debug.py
classes.py
classes_hints.py
comments.py
completion.py
decorators.py
doctest.rst
duplicates.py
evaluate.py
extract_class_method.py
extract_super.py
f-strings.py
test_car.py
car.py
3
class Car:
def __init__(self, speed=0):
self.speed = speed
def accelerate(self): self: <car.Car object at 0x10276d8d0>
self.speed += 5
def brake(self):
def stop(sel
i = 0
while se
print(f"
6
Car › accelerate()
7
8
Debug
Threads & Variables
Console
MainThread
accelerate, car.py:6
drive_on_autobahn, script.py:16
<module>, script.py:23
Evaluate expressi
self = {Car} <c
5
Switch frames from anywhere in the IDE with ⌥⌘↑ ⌥⌘↓
9
MySampleProject › car&tests › car.py
LF UTF-8 4 spaces Python 3.11 (MySampleProject)
Show Context Actions ⌥↵
Paste ⌘V
Copy / Paste Special
Column Selection Mode ⇧⌘8
Find Usages ⌥F7
Go To
4
Folding
Refactor
Generate... ⌘N
Evaluate Expression... ⌥F8
Run to Cursor ⌥F9
Force Run to Cursor ⌥⌘F9
Jump To Cursor
Add to Watches
Add Inline Watch
Run 'car' ^⇧R
Debug 'car' ^⇧D
More Run/Debug
Open in Split with Chooser... ⌥⇧↵
Open In
Local History
Git
Execute Line in Python Console ⌥⇧E
Run File in Python Console
Compare with Clipboard
Diagrams
Create Gist...

Python

Python is a programming language that lets you work more quickly and integrate your systems more effectively.

Why Python?

- Standard language for data science and machine learning. Python is the most popular language for data science and machine learning.
- Easy to learn. Python offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems.
- **Life is short, you need Python.**

Python is dynamically, **implicitly typed** (i.e. you don't have to declare variables), **case sensitive** (i.e. var and VAR are different variables) and **object-oriented** (i.e. everything is an object).

```
# show the usage of the object 'int'
help(int)
#help(1)
# show the properties and methods of the object 'int'
dir(int)
```

Basic Syntax

Single-line comments start with a `#` and are ignored by the interpreter. Multi-line comments are enclosed by triple quotes. Values are assigned to variables using the `=` operator. Python is a dynamically typed language, so you don't have to declare the type of a variable when you create one. Equality is checked using the `==` operator. You can increment/decrement values using the `+=` and `-=` operators respectively by the right-hand amount. You can also use multiple variables on one line.

```
# assign a value to variable
x = 1
# x = x + 2
x += 2
x -= 1
print(x)
# concatenate two string by '+'
y = 'Hello '
y1 = 'world'
print(y + y1)
# Assign two values to two variables respectively
z1, z2 = 1, 'str'
print(z1, z2)
'''
Block comments single/double quotes
'''
```

Data Structures

Python has four basic built-in data structures: `list`, `tuple`, `set`, and `dictionary`. The indexing of the data structures starts from 0. Negative indexing starts from the end of the data structure. `-1` refers to the last element, `-2` refers to the second last element, and so on.

```
# list is a bit like array we have learned in C or C++
list_sample = [1,2]
print(list_sample)
list_sample1 = ['hello', 'world']
print(list_sample1)
# tuple is a bit like list,but its element can't be changed **immutable**
tuple_sample = (1, 2, 3)
print(tuple_sample)
tuple_sample1 = ('hello', 'world')
print(tuple_sample1)
# try to change the element of tuple
tuple_sample1[0] = 'hey'
print(tuple_sample1)
# dictionnary = {key:value}
dict_sample = {'key1': 'value1', 2:3, 'tuple': (1,2)}
# get value by key
print(dict_sample['key1'])
print(dict_sample['tuple'][0])
```

Strings

Strings are sequences of characters, using the syntax of either single quotes or double quotes. You can have a single quote or double quote in a string if you use the other one to enclose the string. i.e. `print('I don't know')` or `print("I don't know")`.

```
a = "Hello, World!"
print(a)

# Traditional string
tom = 'Tom'
print("Name: %s, Age: %d" % (tom, 20))

# f-string after Python 3.6
name = 'Tom'
print(f'Name: {name}, Age: {20}')

# print a string with multiple lines
multi_string = '''this is a
multiline
string'''
print(multi_string)

x = 3.14159265
print(f'pi = {x:.2f}')
```

Flow Control Statements

Flow control statements are used to control the flow of execution in a program. Python has three types of flow control statements: `if`, `for`, and `while`.

```
# generate a list which has elements from zero to nine
range_list = range(10)

for number in range_list:
    # Check if number is one of
    # the numbers in the tuple.
    if number in (3, 4, 7, 9):
        # "Break" terminates a for without
        # executing the "else" clause.
        break
    else:
        # "Continue" starts the next iteration
        # of the loop. It's rather useless here,
        # as it's the last statement of the loop.
        continue

while True:
    # This will always be executed.
    pass  # An empty block
```

Functions

A function is a block of code that only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

```
# define a function
def my_function():
    print("Hello from a function")

# define a function with parameters
def my_function_with_param(name):
    print("Hello, " + name)

# define a function with default parameter
def my_function_with_default_param(name = "Tom"):
    print("Hello, " + name)

# call the function
my_function()
my_function_with_param("Alice")
my_function_with_default_param()
```

- Positional arguments: `def my_function(name, age):`
- Keyword arguments: `def my_function(name="Tom", age=20):`

Classes

Classes are declared using the `class` keyword. The class's constructor is the `__init__` method. The `self` parameter is a reference to the current instance of the class and is used to access variables that belong to the class. Python does not have access specifiers like C++ or Java, so all variables are public. For private variables and methods, you can use the underscore prefix `_` or `__`. Inheritance is declared by putting the parent class name in parentheses after the class name.

```
class MyClass:
    common = 10

def __init__(self):
    self.my_variable = 3

def my_function(self, arg1, arg2):
    return self.my_variable

# This is the class initializer
my_class = MyClass()
my_class2 = MyClass()
```

File I/O

Python has a built-in `open()` function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file. The `open()` function takes two parameters: the file name and the mode. The mode can be `r` for reading, `w` for writing, and `a` for appending. The `open()` function is usually used with two arguments: `open(filename, mode)`.

```
with open("./example.txt",'w') as f:
    # write 'hello world!' to files first line
    f.writelines('hello world!')
    # write 'Bye bye!' to files second line
    f.writelines('Bye bye!')

# Reading a file, must exists first
# 'r' => reading only mode
with open('./example.txt','r') as f:
    print(f.readlines()) # read all content at one time
```

File I/O (cont.)

The `os` module in Python provides a way of using operating system-dependent functionality. The `os` and `os.path` modules include many functions to interact with the file system. `shutil` module in Python provides many functions of high-level operations on files and collections of files. This module helps in automating process of copying and removal of files and directories.

```
import os
import shutil

os.mkdir(dir_path)
# Create folders recursively
os.makedirs("/home/hao/hello")
# Create a empty file
os.mknod("text.txt")
# Delete a file
os.remove(file_path)
# Delete a folder
os.rmdir(dir_path)

shutil.mtree(src)
# Copy file
shutil.copyfile(src, dst)
# Move file
shutil.move(src, dst)
```

Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

```
import numpy as np

b = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

# get the shape of array
print(b.shape)

# get the dimension of array
print(b.ndim)

# get the size of array
print(b.size)

# get the type of array
print(b.dtype)
```

Check the official documentation for more details: https://numpy.org/doc/stable/user/quickstart.html

Numpy (cont.)

```
import numpy as np

# create a 3x3 2D array with all zeros
a = np.zeros((3, 3))

# Indexing
a[0, 0] = 1

# Slicing
b = a[:2, :2]

# create a 3x3 2D array with all constants
d = np.full((3, 3), 7)

# create a 3x3 array filled with random values
e = np.random.random((3, 3))

# create a 3x3 identity matrix
f = np.eye(3)
```

Check the official documentation for more details: https://numpy.org/doc/stable/user/quickstart.html

Numpy (cont.)

```
import numpy as np

# Sum of all the elements
print(np.sum(a))

# Transpose of a matrix
print(a.T)

# Matrix multiplication
print(np.dot(a, b))

# Matrix inverse
print(np.linalg.inv(a))

# Matrix determinant
print(np.linalg.det(a))
```

Check the official documentation for more details: https://numpy.org/doc/stable/user/quickstart.html

Resources

- Basics
  - Getting started with conda
  - Conda Cheat Sheet
  - Python Cheat Sheet
  - PyCharm Reference Card
  - Numpy Cheat Sheet
- Advanced
  - Fluent Python: https://www.oreilly.com/library/view/fluent-python/9781491946237/
  - Refactoring: Improving the Design of Existing Code: https://www.oreilly.com/library/view/refactoring-improving-the/9780134757681/
  - Clean Code: A Handbook of Agile Software Craftsmanship: https://www.oreilly.com/library/view/clean-code-a/9780136083238/

Numpy Exercises

1. One Very Simple NumPy Tutorial:

   https://nbviewer.jupyter.org/github/imkalyan/Numpy/blob/master/NumPy.ipynb
2. Numpy performance tricks: https://nbviewer.org/gist/rossant/4645217
3. 100 NumPy exercises with solutions: https://github.com/rougier/numpy-100

Finally

You are welcomed to walk-in to discuss your problems or questions during the Office Hours (Friday 4-6pm).

- Email: xiang.hao@connect.polyu.hk
- Office: PQ 605, Table T19
- Office Hours: Friday 4 - 6pm