計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 1: Introduction

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Instructor

Kuo-Chuan Pan (潘國全)

Office: R506, General building II

Phone: 03-57(42563)

Office hours: by appointment

(kuochuan.pan@gapp.nthu.edu.tw)

Web: https://kuochuanpan.github.io

Teaching Assistant

Hao-Sheng Wang (王皓陞)

Office: R519, General building II

Phone: none

Email: whshoward@gmail.com

Office hours: by appointment (or in google classroom)

Computing courses in the Physics department

- PHYS290000 計算物理概論
- PHYS317000 數值分析
- PHYS401200 計算物理
- PHYS401300 計算物理實作
- PHYS597002 物理專題：計算物理
- PHYS597003 物理專題：凝態計算物理
- PHYS591000 物理與人工智慧(AI)實作導論
- ASTR660000 計算天文物理

Prerequisites

1. General physics, calculus
2. Access a computer (bring your laptop)
3. This course is designed as the first computational course for our physics students
4. All physics students without prior programming experience are highly encouraged to take this course
5. Spend > 4 hours per week

It takes times to be an experienced programmer

Do EXPECT that you can be a python Expert after taking this course.

“Researchers (Bloom (1985), Bryan & Harter (1899), Hayes (1989), Simmon & Chase (1973)) have shown it takes about ten years to develop expertise in any of a wide variety of areas, including chess playing, music composition, telegraph operation, painting, piano playing, swimming, tennis, and research in neuropsychology and topology. The key is deliberative practice: not just doing it again and again, but challenging yourself with a task that is just beyond your current ability, trying it, analyzing your performance while and after doing it, and correcting any mistakes. Then repeat. And repeat again.”

It is never too late to start learning programming.

REF: http://norvig.com/21-days.html#answers

Goal of this course

1. First course on Python programming
2. Learn basic python packages for scientific computing
3. Basic data analysis and visualization

Main topics

1. Python programming
2. Data analysis and visualization
3. Object-oriented programming with python
4. Using numpy/scipy for scientific computing

Google classroom

hxj5xed

(111) Introduction ot Computational Physics 複製邀請連結

Course overview

Date	Description
2/13	Lecture 1
2/20	Lecture 2
2/27	**National Holiday (No lecture)**
3/6	Lecture 3
3/13	Lecture 4
3/20	Lecture 5
3/27	Lecture 6
4/3	**National Holiday (No lecture)**
4/10	Lecture 7
4/17	Lecture 8
4/24	Lecture 9
5/1	Lecture 10
5/8	Lecture 11
5/15	Lecture 12
5/22	Lecture 13
5/29	**Professor away (No lecutre)**
6/5	Lecture 14
6/12	**(no lecutre)**

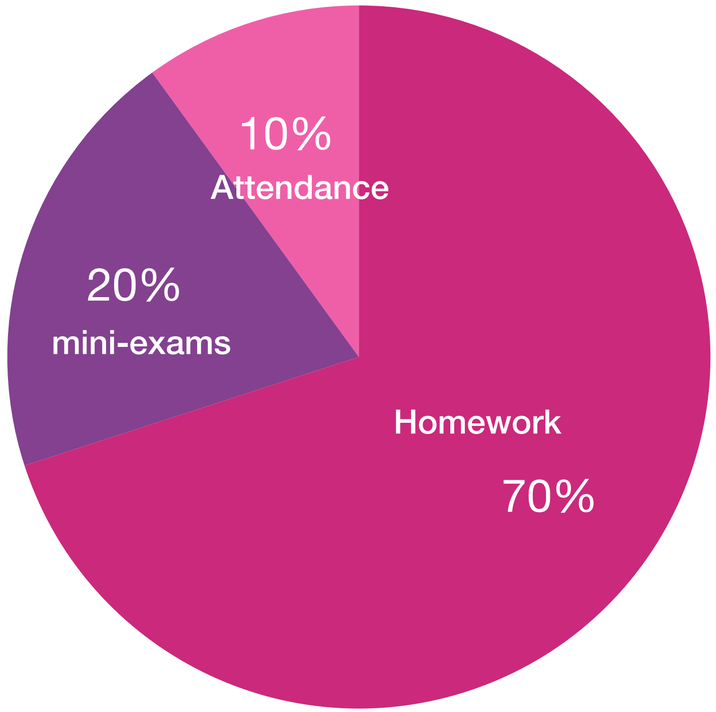
Weekly lectures on Monday
15:30 - ~~18:20~~ 18:00 (no break)

Topics
Programming with Python (from beginner to a junior computational physicist)
Object-oriented programming with Python
Precisions and floating point numbers
Command line interface and shells
Visual Studio Code and Jupyter notebook
Version control with `git`
`numpy` and `scipy` for scientific computing
Data frame and statistics
Data visualization with `matplotlib`
Speedup your codes
Using chatGPT to improve your codes

A “tentative” plan

Grade

Grade Breakdown



Bonus: Extra X %
(Asking questions & participation)

A+	> 90
A	85 - 89
A-	80 - 84
B+	77 - 79
B	73 - 76
B-	70 - 72
C	60 - 69
D	<60

Homework assignments

1. Roughly weekly or biweekly homework
2. Use .py or .ipynb for your solutions
3. “Must” have comments to explain your code
4. Submit all necessary codes/data to google classroom
5. There will be late penalty!!!

Mini-exams

1. There will be a few mini-exams at the end of a lecture.
2. You could check the lecture slides but do not discuss with your classmates or use internet.

Honor Code

1. All work must be in your own words
2. You need to attach your source codes with your solutions
3. You may reference outside resources (for ideas or concepts), but include citations (e.g. urls) at the end of your answers. However, taking codes (or a minor function) from the internet are not allowed
4. You may ask questions to your friends/classmates. He/She may suggest strategies for solving the homework (or debugging), but he/she may not look at your code for you — not even if you sit together and look for the issues/bugs.
5. The purpose of homework assignments is to assess your understanding of concepts; you must demonstrate that understanding to get credit

AD: online learning

https://forms.gle/W8eqdNYAvYdGc55T7

#數位自學不斷電 #數位自學正式開賽 #搶先預約 #數位自學計畫

開催決定：

線上報名：即日起至2023年3月5日（週日）

公布通過名單：2023年3月13日（週一）

完成學習心得：2023年6月30日（週五）

最熱血報名：https://forms.gle/W8eqdNYAvYdGc55T7

初級駕照R：填寫線上申請報名表，教發中心組成駕訓班（數位自學審核小組）審查，擇優補助。通過申請階段之車手可獲得獎金新台幣100元或等值禮券。

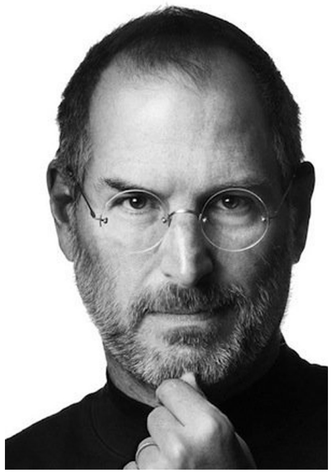
中級駕照L2：數位自學動起來，車手根據自訂日程數位自學。

PRO駕照：四個月內完成自學內容，提供學習心得（約500字，可包含：心得、照片、課程成果等）與修習證明（修課證明、課程完成證明、修課成績單或學習歷程截圖），經確認無誤後擇優予以核發獎金新台幣400元或等值禮券。

Why Python?

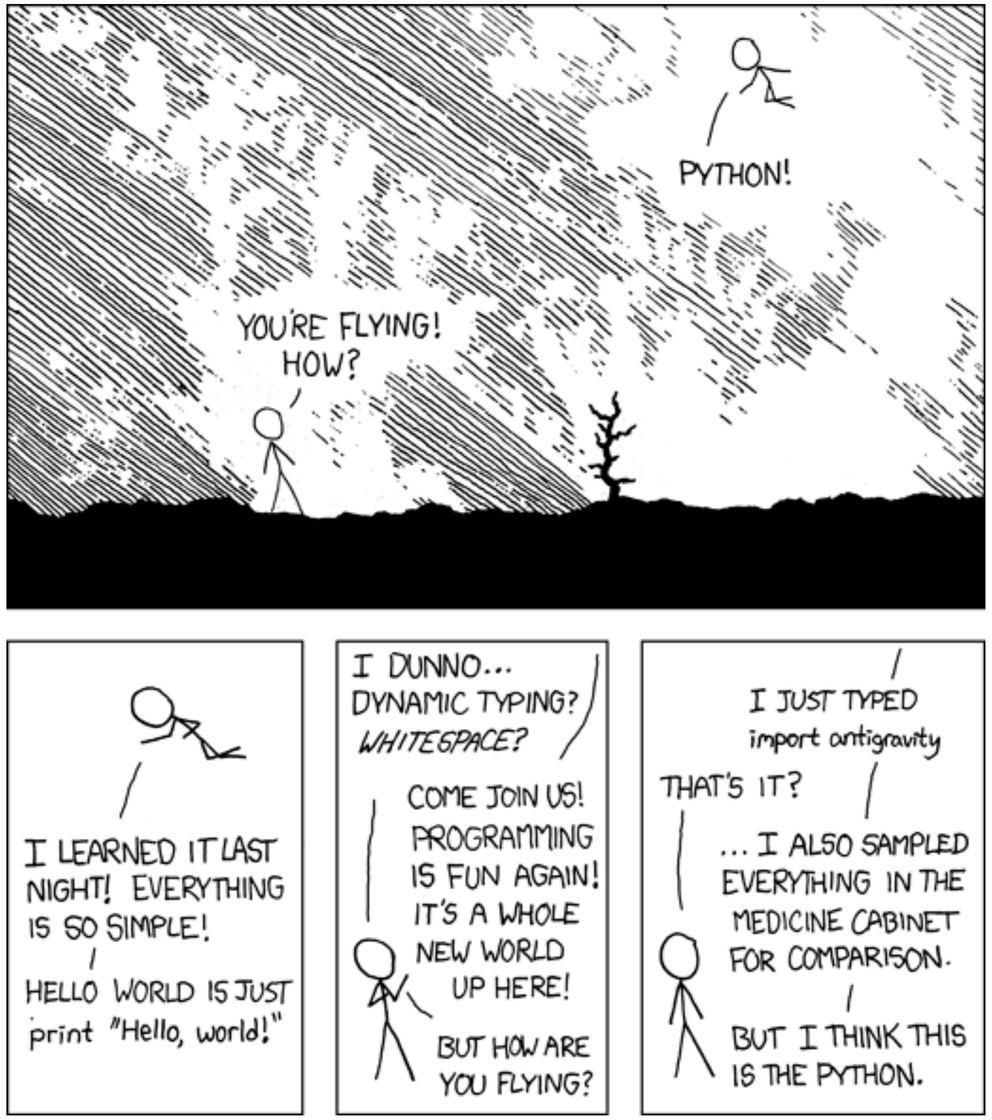
Course overview

"Everybody ~~in this country~~ should learn to program a computer, because it teaches you how to think"
— *Steve Jobs*



"Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains."
—*Bill Gates*

Why Python?



xkcd.com

python™

- A modern, interpreted, high-level, general purpose programming language
- Easy to learn and use
- Expressive language: fewer codes
- Dynamically typed: No need to define the type of variables (disadvantage: slow)
- Interpreted: No need to compile (disadvantage: slow)
- Automatic memory management (disadvantage: memory leak)

Hello world program

C

```c
#include <stdio.h>

int main(void) {
   printf("Hello! World!\n");

   return 0;
}
```

```
$ gcc hello.c
$ ./a.out
```

Python

```python
print(“Hello! World!”)
```

```
$ python hello.py
```

What makes python good for scientific computing?

- Large community of users
- Plenty of scientific libraries and environments (ex. numpy, scipy, matplotlib, scikit-learn, astropy, …etc.)
- Good integration with highly optimized codes written in C and Fortran
- Good support for parallel programming (MPI) and GPU computing
- Open sourced

2021 Stackoverflow developer survey

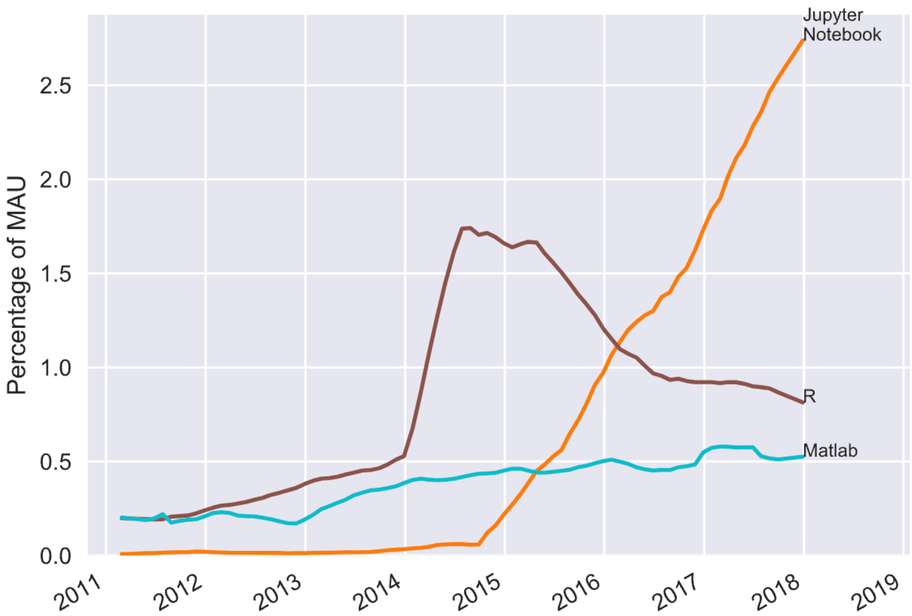
Mainly used for web development and web app.

https://insights.stackoverflow.com/survey/2021?fbclid=IwAR1Z1mL6gGopvbtpkprN-onJCtF25U_oTipDyMihEUQH4mPorRvl1WmDQIE#overview

Scientific programming

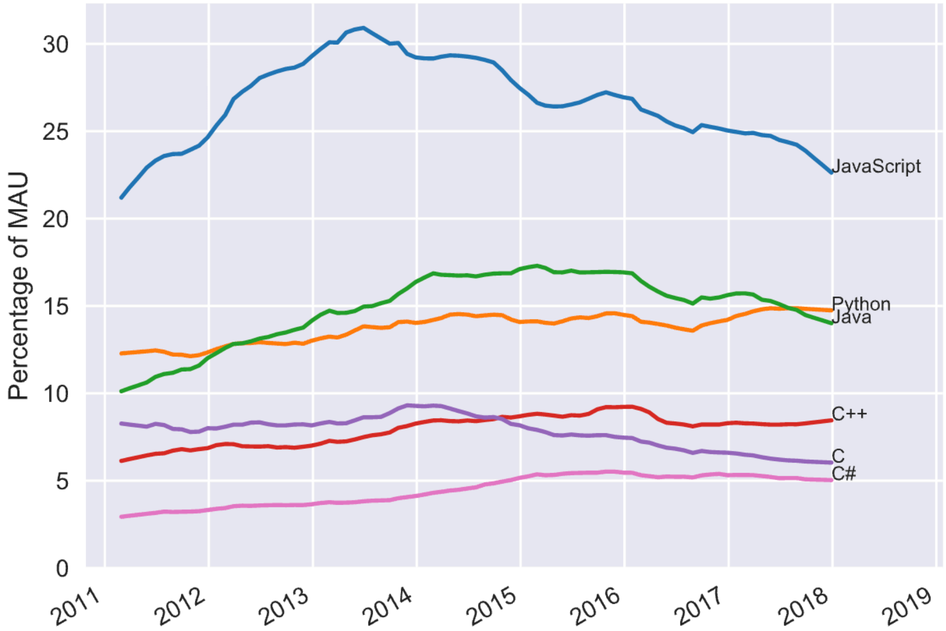
Scientific Languages

There was one other fast-growing 'language' included in the results that I purposefully left out:



Major Languages

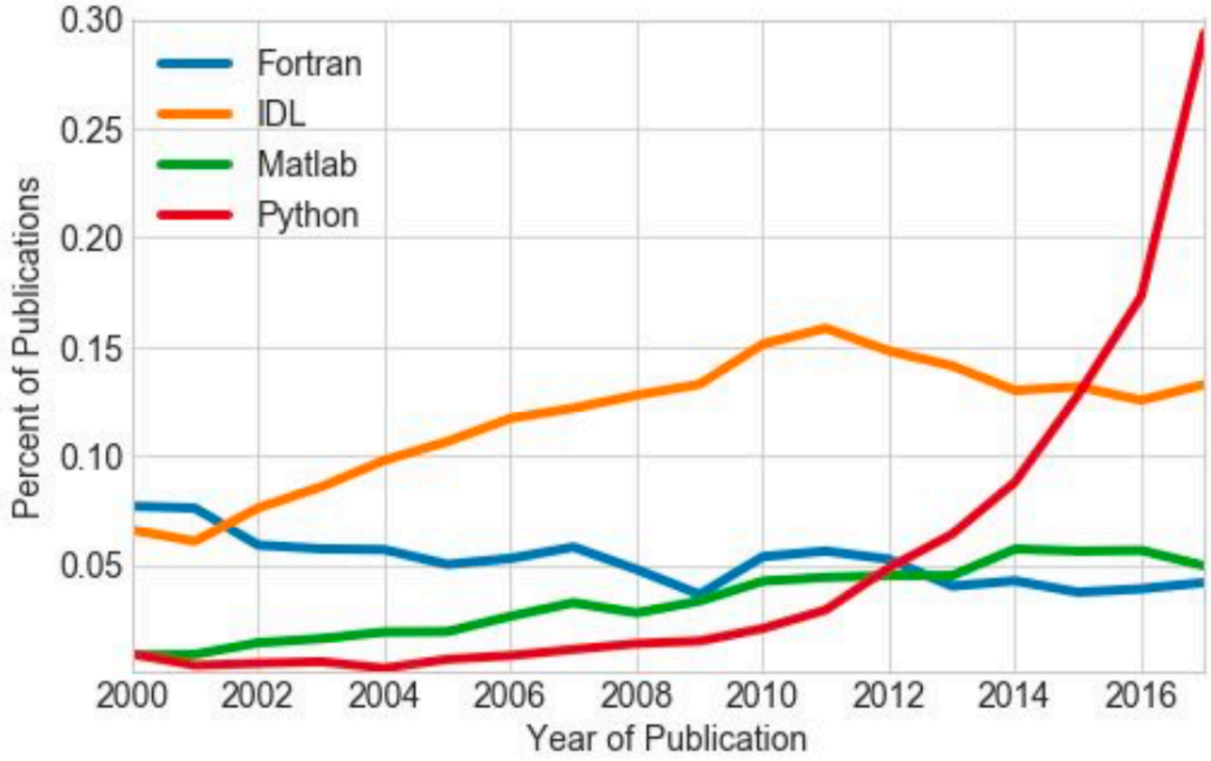
The major programming languages have relatively stable usage, and are mostly what you'd expect:



https://www.benfrederickson.com/ranking-programming-languages-by-github-users/

Scientific programming

Mentions of Software in Astronomy Publications:



Compiled from NASA ADS (code).

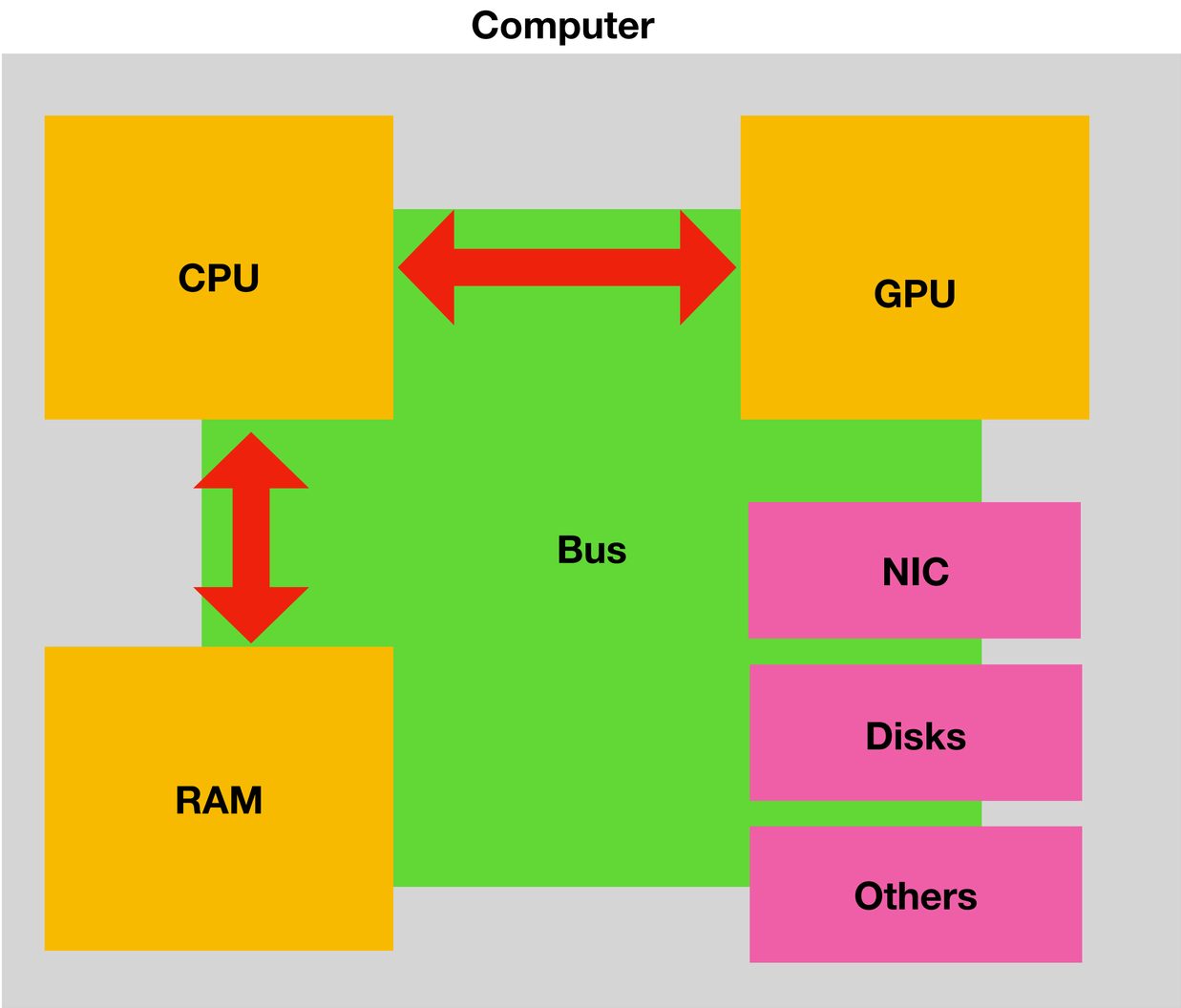
Thanks to Juan Nunez-Iglesias, Thomas P. Robitaille, and Chris Beaumont.

Python

- First released in 1991
- Purpose: improve the code readability
- Python 2.0 was released in 2000
- Python 3.0 was released in 2008 (not completely backward-compatible)
- Python 3.11 (current version) was released in 2022
- In this course, we will use python 3.10

Before we learn programming

Computer Architecture



- CPU (Central Processing Unit)
- GPU (Graphics Processing Unit)
- RAM (Random Access Memory)
- Disks: for storage
- NIC (Network Interface Card): connect to network
- Others: keyboard, mouse, monitor, USB drivers, …etc.
- Bus (PCIe, SATA): The speed of bus determine how fast the components to exchange data.

Speed and capacity

CPU:

Intel i9-13900KF【24核/32緒】3.0G(↑5.8G)/36M/無內顯/無風扇/125W【代理盒裝】, $20300↘$19600

Core/thread Speed Cache size Power

MB:

技嘉 Z790 AORUS ELITE AX(ATX/Realtek2.5Gb+Wi-Fi 6E/註冊五年)16+1+2相電源, $9490 ◆ ★ ↓任搭100↓

CPU pins MB size Network speed

RAM:

金士頓 32GB(雙通16GB*2) DDR5-6400 FURY Renegade RGB(KF564C32RSAK2-32), $8700↘$5950 ◆ ★ 熱賣

Capacity Speed: 6400 MT/s = 6400 MHz

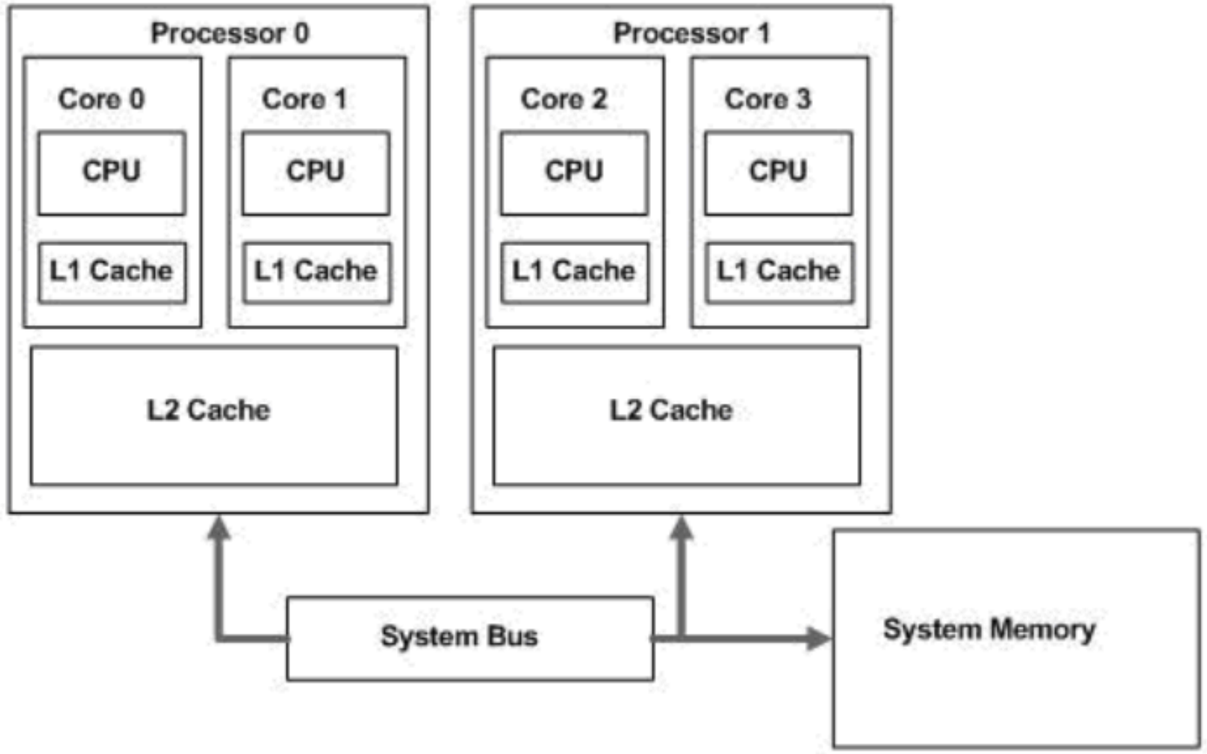
GPU:

技嘉 AORUS RTX4090 XTREME WATERFORCE 24G(2565MHz/23.8cm)封閉水冷*限購一片, $63990 ◆ ★

Capacity Speed

NOTE: Nowadays, the memory speed is usually slower than the cpu clock speed.
Will that impact the performance? Ans: usually not, we cpu has cache

Cache memory and cache latency



- CPU Caches

L1: ~ 64 KB / core
L2: ~ 256 KB / core
L3: ~ 2.5 MB / core
Memory: < TB
Storage: ~ 10 TB or more

- Level 1 cache (L1): fastest but small
- Level 2 cache (L2): slower than L1 but bigger than L1
- Level 3 cache (L3): slower than L2 but bigger than L2
- Cache hit: find data in cache
- Cache miss: cannot find data in cache —> go to next level
- Cache performance: measured by hit ratio (= hit / (hit+miss))
- Cache latency: the delay from different levels
- System memory (RAM):
- Disk storage (DISK):

Cache memory and cache latency

Numbers everyone should know

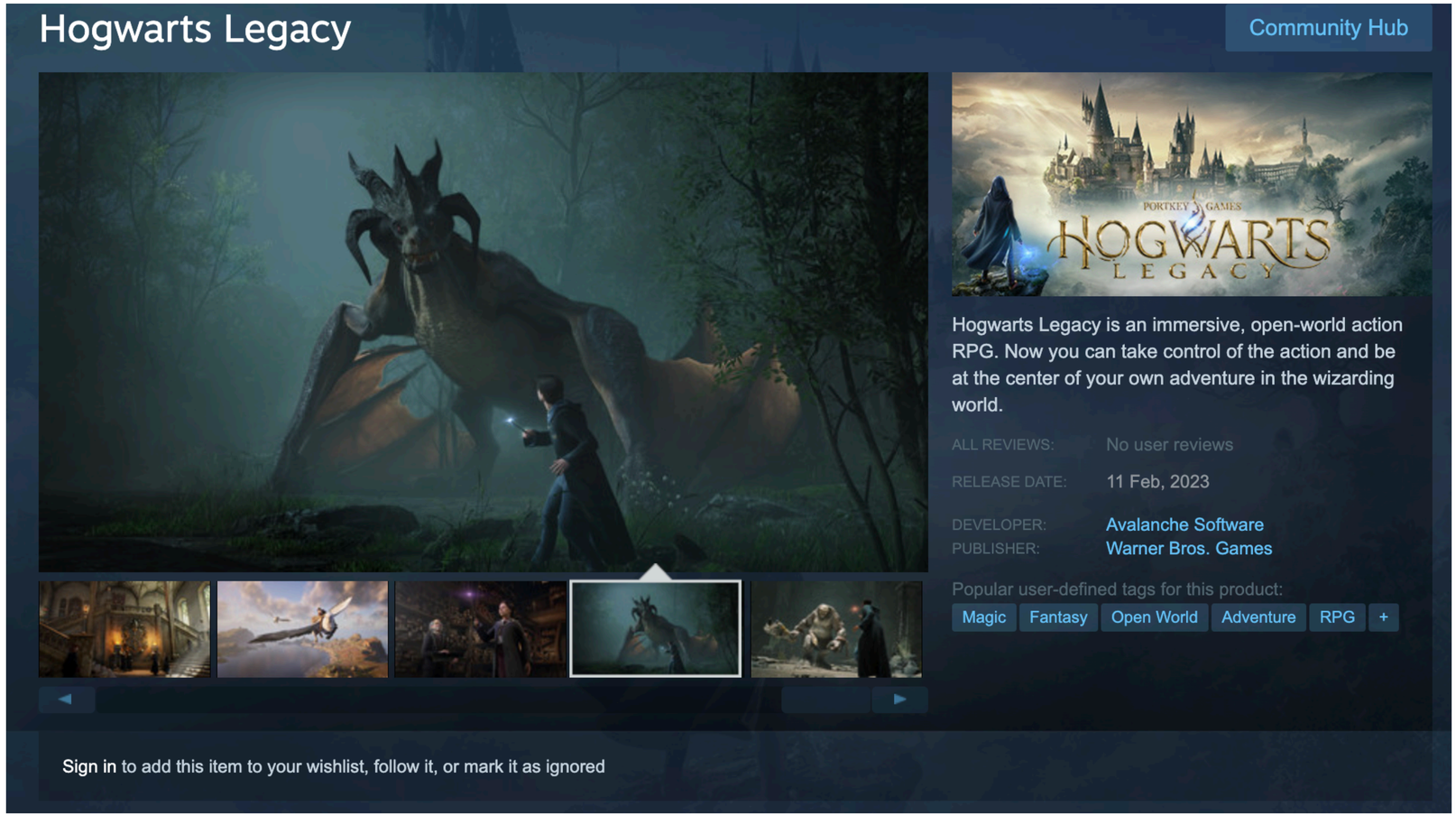
```
           0.5 ns - CPU L1 dCACHE reference
           1   ns - speed-of-light (a photon) travel a 1 ft (30.5cm) distance
           5   ns - CPU L1 iCACHE Branch mispredict
           7   ns - CPU L2  CACHE reference
          71   ns - CPU cross-QPI/NUMA best  case on XEON E5-46*
         100   ns - MUTEX lock/unlock
         100   ns - own DDR MEMORY reference
         135   ns - CPU cross-QPI/NUMA best  case on XEON E7-*
         202   ns - CPU cross-QPI/NUMA worst case on XEON E7-*
         325   ns - CPU cross-QPI/NUMA worst case on XEON E5-46*
      10,000   ns - Compress 1K bytes with Zippy PROCESS
      20,000   ns - Send 2K bytes over 1 Gbps NETWORK
     250,000   ns - Read 1 MB sequentially from MEMORY
     500,000   ns - Round trip within a same DataCenter
  10,000,000   ns - DISK seek
  10,000,000   ns - Read 1 MB sequentially from NETWORK
  30,000,000   ns - Read 1 MB sequentially from DISK
 150,000,000   ns - Send a NETWORK packet CA -> Netherlands
|   |   |   |
|   |   | ns|
|   | us|
| ms|
```

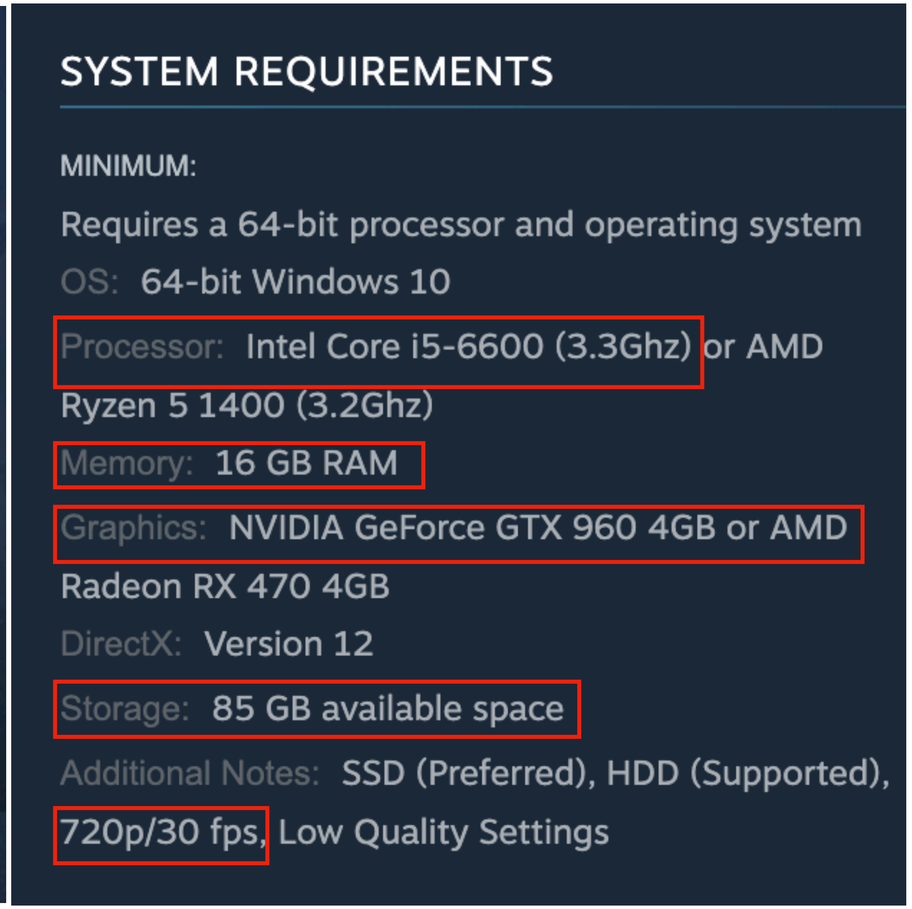
From: Originally by Peter Norvig:

- http://norvig.com/21-days.html#answers
- http://surana.wordpress.com/2009/01/01/numbers-everyone-should-know/,
- http://sites.google.com/site/io/building-scalable-web-applications-with-google-app-engine

https://stackoverflow.com/questions/4087280/approximate-cost-to-access-various-caches-and-main-memory

Know what you could do with your computer





You could find a similar system minimum requirement for your computational task.

If your laptop does't match your requirement or it takes too long to finish the calculation,

1) buy a new one, 2) use the work station from your advisor, 3) the cluster in your institute, 4) use national supercomputers

Storage

- To estimate how many storage (ram and disk) we need for a computing problem, we should understand the unit of storage first.
- Network bandwidth or hard drive size are usually described by ~ XXX Mb/s (megabits per second) or TB (terabytes)
- A bit is a binary digit (0 or 1)
- 8 bits = 1 byte
- For hard disks = 1 TB = 10^{12} bytes, but in the operating system, 1 TB = 1024 * 1024 * 1024 bytes = 931 GB
- How many bits of memory you need to storage one number?

Rabbit

Rabbyte

Integer and Floating point numbers

- In computer, you need finite size of memory to store an integer or a real number.
- Depending on the programming language and systems, there might be different systems.
- Integers usually takes 2-4 bytes.

Type	Signed?	Number of bits	Smallest value	Largest valu
`Int8`	✓	8	-2^7	2^7 - 1
`UInt8`		8	0	2^8 - 1
`Int16`	✓	16	-2^15	2^15 - 1
`UInt16`		16	0	2^16 - 1
`Int32`	✓	32	-2^31	2^31 - 1
`UInt32`		32	0	2^32 - 1
`Int64`	✓	64	-2^63	2^63 - 1
`UInt64`		64	0	2^64 - 1
`Int128`	✓	128	-2^127	2^127 - 1
`UInt128`		128	0	2^128 - 1

Integer types in Julia

Question: how to store 0 and the sign?

Floating point numbers

- Similar to scientific notation
- Floating point number system characterized by four integers

β	Base
p	Precision
$[L, U]$	Exponent range

Real number x is represented as

$$x = \pm \left(d_0 + \frac{d1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

Where $0 \leq d_i \leq \beta - 1$ and $L \leq E \leq U$

Example: Floating point numbers

$$x = \pm \left(d_0 + \frac{d1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

If $\beta = 10$

Dec. 362.5 = (3 + 6/10 + 2/100 + 5/1000) 10^2

If $\beta = 2$

Dec. 362.5 = 256 + 64 + 32 +8 +2 + 0.5
= (2^8 + 2^6 + 2^5 + 2^3 + 2^1 + 2^-1)
= 10110101.1 = 1.01101011 * 2^8

If $\beta = 8$ Dec. 25 = Oct. 31

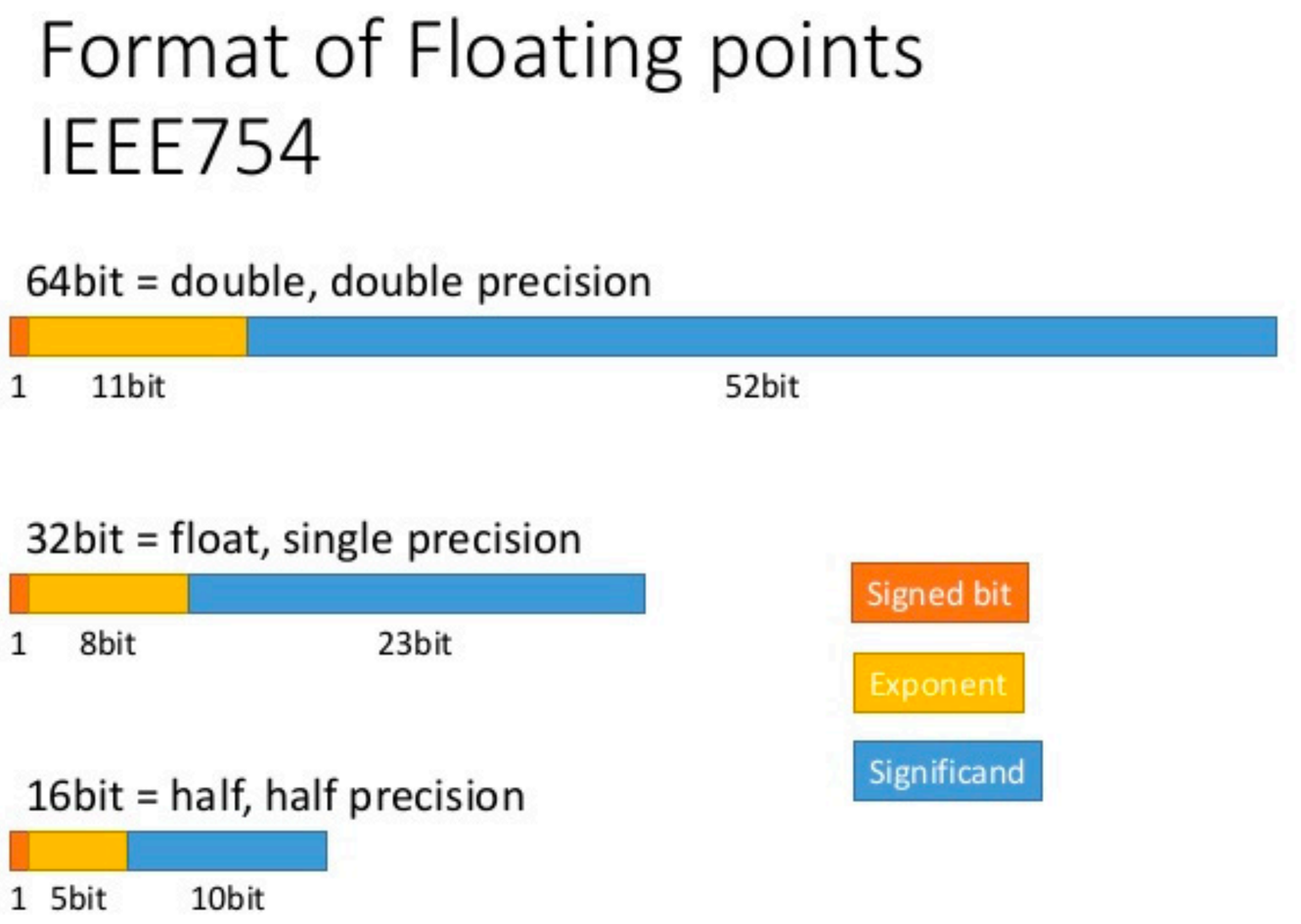
Type of Floating point numbers

Parameters for typical floating-point systems

system	β	p	L	U		
IEEE HP	2	11	-14	15	**2 bytes**	5 bits for L,U,0, sign, 11 bits for p
IEEE SP	2	24	-126	127	**4 bytes**	8 bits for L,U,0, sign, 24 bits for p
IEEE DP	2	53	-1022	1023	**8 bytes**	
IEEE QP	2	113	-16382	16383	**16 bytes**	
Cray-1	2	48	-16383	16384		
HP calculator	10	12	-499	499		
IBM mainframe	16	6	-64	63		

Modern computers use binary arithmetic (beta=2)

Memory Usage



Note that a Python float object takes a full 24 bytes of memory: 8 bytes for the underlying double precision value, 8 bytes for a pointer to the object type, a 8 bytes for a reference count (used for garbage collection)

A Python integer takes > 24 bytes of memory: 8 bytes for the value, 8 bytes for a pointer to the object type, a 8 bytes for a reference count, and at least 4 bytes to store the type. The larger the integer, the higher the memory are used

An empty string “” takes 49 bytes in Python.

We will talk about these more in later lectures.

Normalization

- Floating point number is normalized if leading digit is always non-zero unless number represented is zero
- In normalized system, mantissa m of nonzero floating point number always satisfies $$1 \leq m < \beta$$
- Reasons for normalization:
  - Each number is unique
  - No digits are wasted

Properties

$$x = \pm \left(d_0 + \frac{d1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

- Floating point numbers are finite and discrete
- Total number of normalized floating point number is

$$\underbrace{2}_{\text{signs}}\underbrace{(\beta - 1)}_{\text{1st digit}}\underbrace{\beta^{p-1}(U - L + 1)}_{\text{Exponents}} + \underbrace{1}_{\text{zero}}$$

- Smallest positive normalized number: $UFL = \beta^L$
- Largest floating point number: $OFL = \beta^{U+1}(1 - \beta^{-p})$
- Not all real numbers exactly representable

Rounding rules

- If a real number is not exactly representable, then it is approximated by "nearby" floating point number
- This process is called rounding, and error introduced is call "rounding error"
- In IEEE systems: round to nearest
- So IEEE single, double, and quadruple precision systems have about 7, 16, and 36 decimal digits of precision, respectively.

Machine precision

- Rounding to nearest,

$$\epsilon_{mach} = \frac{1}{2}\beta^{1-p}$$

Recall:

$$\boxed{x = \pm\left(d_0 + \frac{d1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_{p-1}}{\beta^{p-1}}\right)\beta^E}$$

- Maximum relative error in floating point system is

$$\left|\frac{fl(x) - x}{x}\right| \leq \epsilon_{mach}$$

Exceptional values

- IEEE floating point standard provides special values to indicate positive and negative “infinity” (+Inf/ -Inf) and “not a number” (NaN)
- Examples:

$$1/0 = \infty$$

$$\log(0) = -\infty$$

$$\sqrt{-1} = \mathrm{NaN}$$

Floating Point Arithmetic

- Addition or subtraction: shifting mantissa to make exponents match may cause loss of some digits of smaller number

$$1.23 \times 10^{14} + 1.01 \times 10^{-5}$$

- Multiplication: product of two p-digit mantissas contains up to 2p digits, so results may not be representable

$$1.2 \times 1.2 = 1.44$$

- Division: Quotient of two p-digit mantissas may contain more than p digits.

$$1/10$$

- Results of floating-floating-point arithmetic operation may differ from results of corresponding real arithmetic operation on same operands

Cancellation

- Subtraction between p-digit numbers having same sign and similar magnitudes yield result with fewer than p digits, so it is usually exactly representable

$$1.234 \times 10^3 - 1.223 \times 10^3 = 0.011 \times 10^3 = 11$$

- Which is correct and exactly representable but has only 2 significant digits
- Often implies serious loss of information

Example: Cancellation

- Total energy of helium atom is sum of kinetic and potential energies, which are computed separately and have opposite signs, so suffer cancellation

- Although computed values for kinetic and potential energies changed by only 6% or less, resulting estimate for total energy changed by 144%

Year	Kinetic	Potential	Total
1971	13.0	−14.0	−1.0
1977	12.76	−14.02	−1.26
1980	12.22	−14.35	−2.13
1985	12.28	−14.65	−2.37
1988	12.40	−14.84	−2.44

Example: Quadratic Formula

- Equation $ax^2 + bx + c = 0$

  has two solutions $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Naive use of formula can suffer overflow or underflow or several cancellation
- Cancellation between -b and square root

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

- Cancellation inside square root cannot be easily avoided without using higher precision

Example: Mean & Std

- Mean and standard deviation are given by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\right]^{\frac{1}{2}}$$

- Is mathematically equivalent to

$$\sigma = \left[\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right)\right]^{\frac{1}{2}}$$

Summary: floating point systems

- Real numbers are approximated by finite and discrete floating-point number, with sign, exponent, and mantissa
- Exponent field determines range of representable magnitudes
- Mantissa field determines precision and relative accuracy or floating point approximation
- Rounding error is loss of least significant, railing digits when approximating true real number by nearby floating point numbers
- Cancellation is loss of most significant, resulting in fewer significant digits in finite precision

Programming Environment

Environment

- Linux and Mac OS are recommended.
- Windows users are suggested to use WSL2
- Possible to use Windows
- Python 3.10 is recommended (via Anaconda)
- VS Code is recommended

計算物理概論

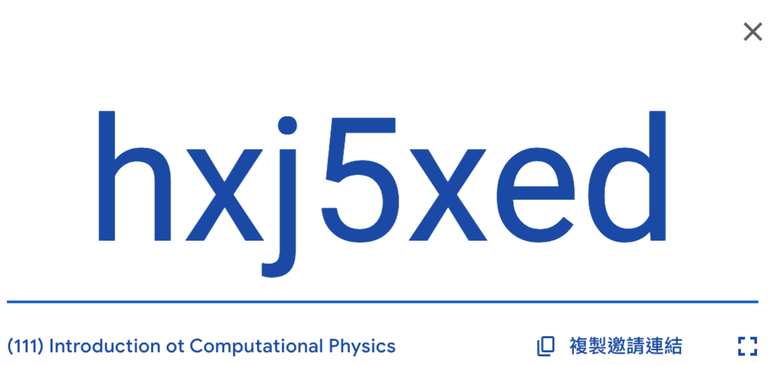
Introduction to Computational Physics (PHYS290000)

Lecture 1: Introduction

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Introduction (google classroom)
2. Course overview
3. Computer Architecture
4. Floating point numbers and memory usages



Today’s plan

1. Programming environment
2. Install python via Anaconda 3
3. Install VS Code
4. Virtual Python Environment with Conda
5. Hello world python program and Jupyter notebook
6. Version Control with Git & Github

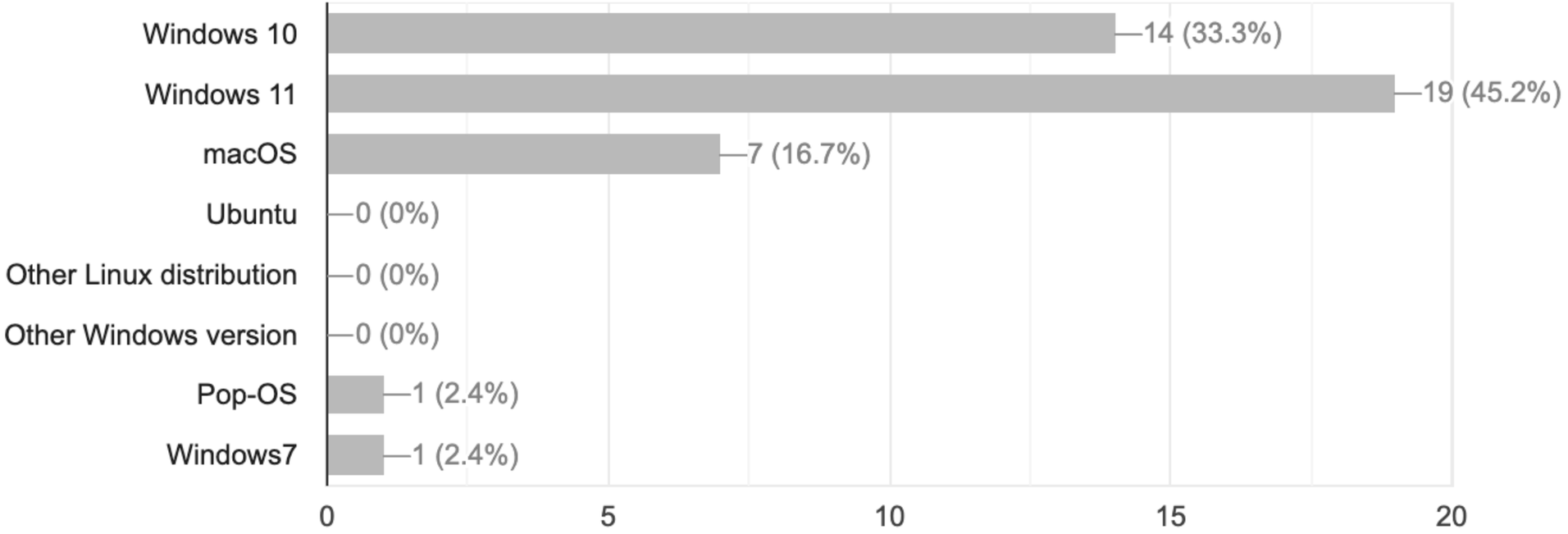
Programing Environment

Operating Systems

What operating system is used in your laptop?

複製

答對次數：0 (作答總數：42)



Operating Systems

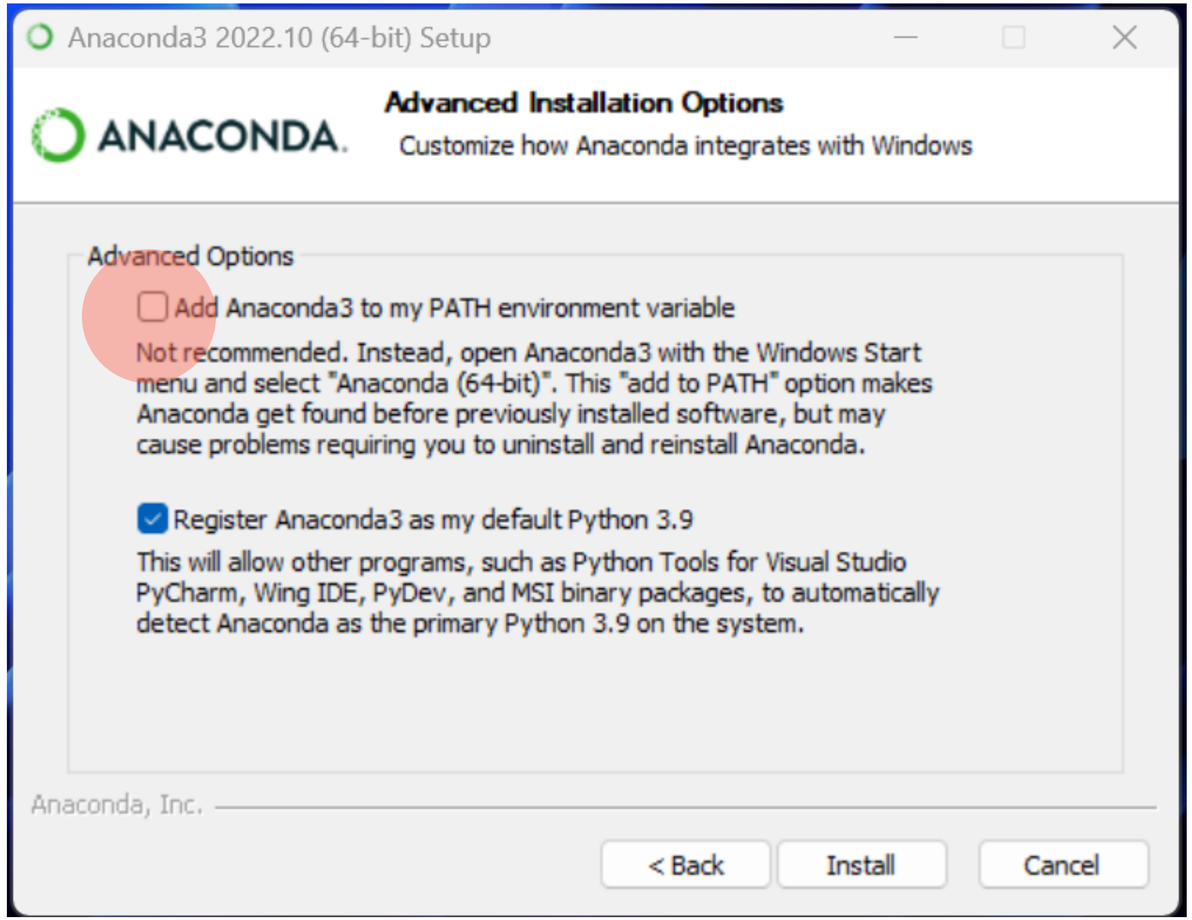
1. Windows 7/10/11 [see page 7]
2. Windows Subsystem for Linux 2 (WSL2) [see page 14]
3. Linux (Ubuntu) *(recommended)* [see page 16]
4. macOS *(recommended)* [see page 16]

- Why Linux? Or Unix-like systems?
- Many open source projects. Many important library/packages for both academia and industry are developed natively for Linux (better community support).
- Native support for SSH, which could help you manage your server easier.
- Linux supports almost all of the major programming languages

Windows Users

Install Python via Anaconda

1. Follow the instruction here: https://docs.anaconda.com/anaconda/install/windows/ to install Anaconda
2. Note: You could click “add Andaconda to my PATH environment variable” and “Register Anaconda 3 as my default Python 3.9”



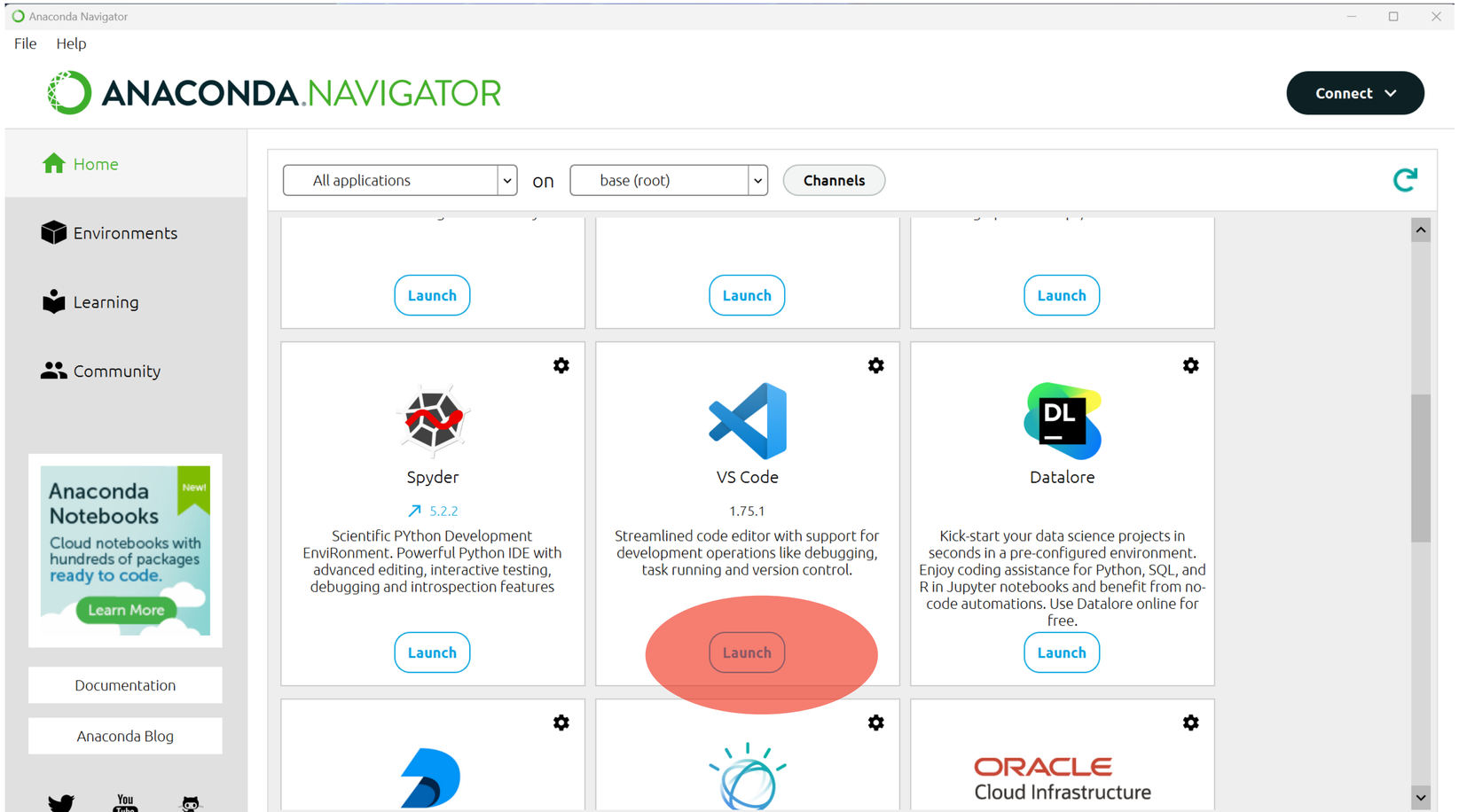
Create a Virtual environment using Conda

1. Press the “Win” key
2. Type “Anaconda Prompt (anaconda3)” (it will launch a terminal window similar to power shell)
3. Type “conda update conda” to check if Conda is up to date (type “y” if there are any updates)
4. Create a virtual environment with `conda create --name comphys python=3.10`
5. Type “y” to confirm
6. Activate the environment by `conda activate comphys`
7. Deactivate the environment by `conda deactivate`

Create a project folder

Install / Launch the VS Code

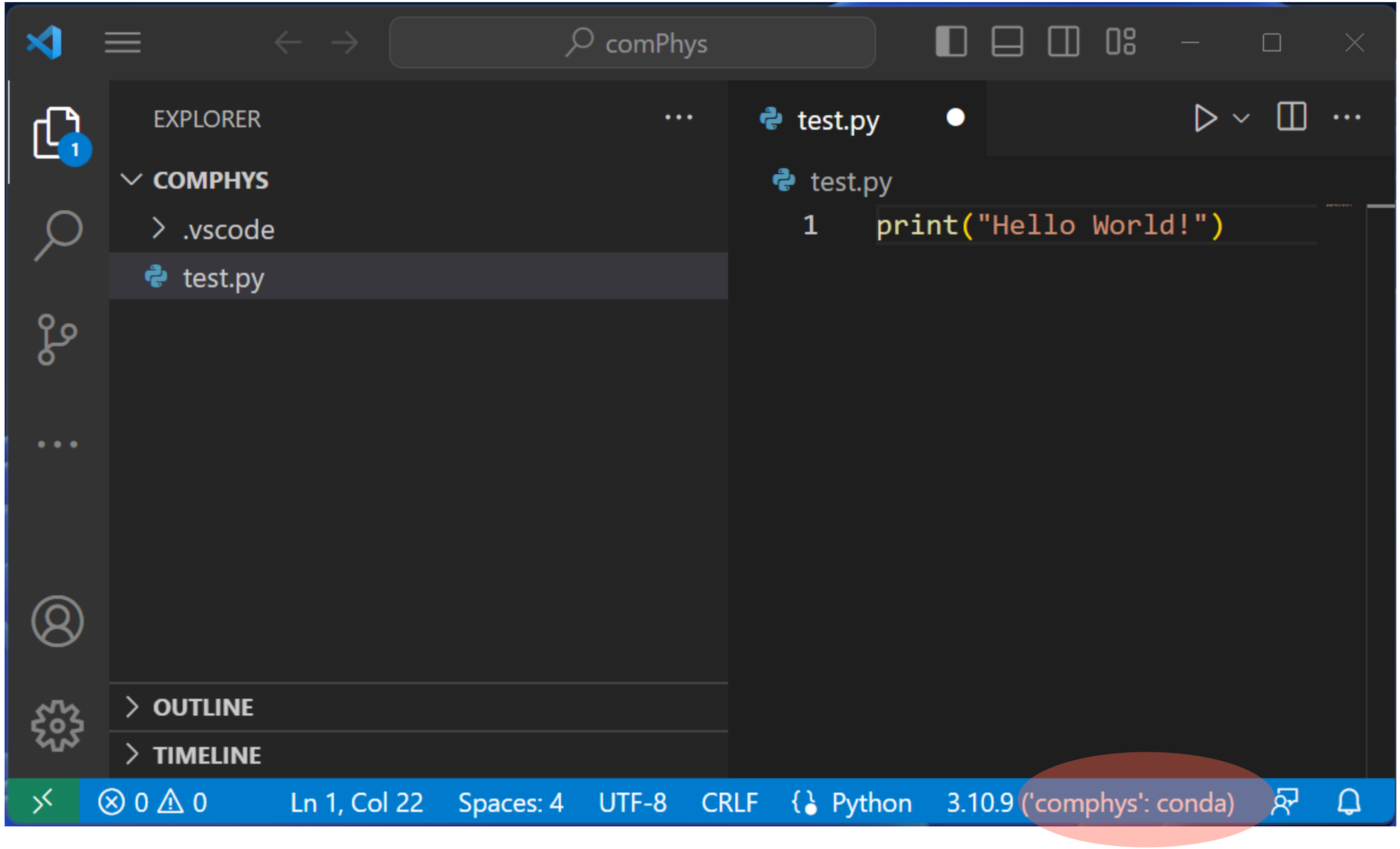
1. Click “Win” key and search for “Anaconda Navigator”
2. Install / Launch the VS Code by click the “Install” or “launch” icon.



Install python extensions in VS Code

1. Open your project folder
2. Create a new file called “test.py”
3. Select the “comphys” python environment
4. In the extension tab, search for “python” and “jupyter” and then install them

If successful, …



WSL2 Users

Install WSL2 / VS Code / Anaconda

1. Follow this tutorial to install WSL2 and VS Code https://ubuntu.com/tutorials/working-with-visual-studio-code-on-ubuntu-on-wsl2#1-overview
2. In the VS Code, install additional extensions: python, Jupyter
3. See page 20 to install Anaconda

Linux / Mac Users

Install Python via Anaconda

1. Install Anaconda here https://www.anaconda.com/products/distribution

   Mac: install the .pkg file

   Linux / WSL2: run the install script $ bash Anaconda-3-x.x.-Linux-x86_64.sh

2. Mac/ Linux: type “yes” to add the python path in your .bashrc (.bash_profile) file

3. Install VS Code here https://code.visualstudio.com/

Create a virtual environment with conda

1. Open a terminal with the “Terminal” app or iTerm2
2. Check if you could find conda with “conda -- version”
3. Create a virtual python environment with conda

```
conda create --name comphys python=3.10
```

Activate or deactivate the comphys env by

```
conda activate comphys
```

```
conda deactivate
```

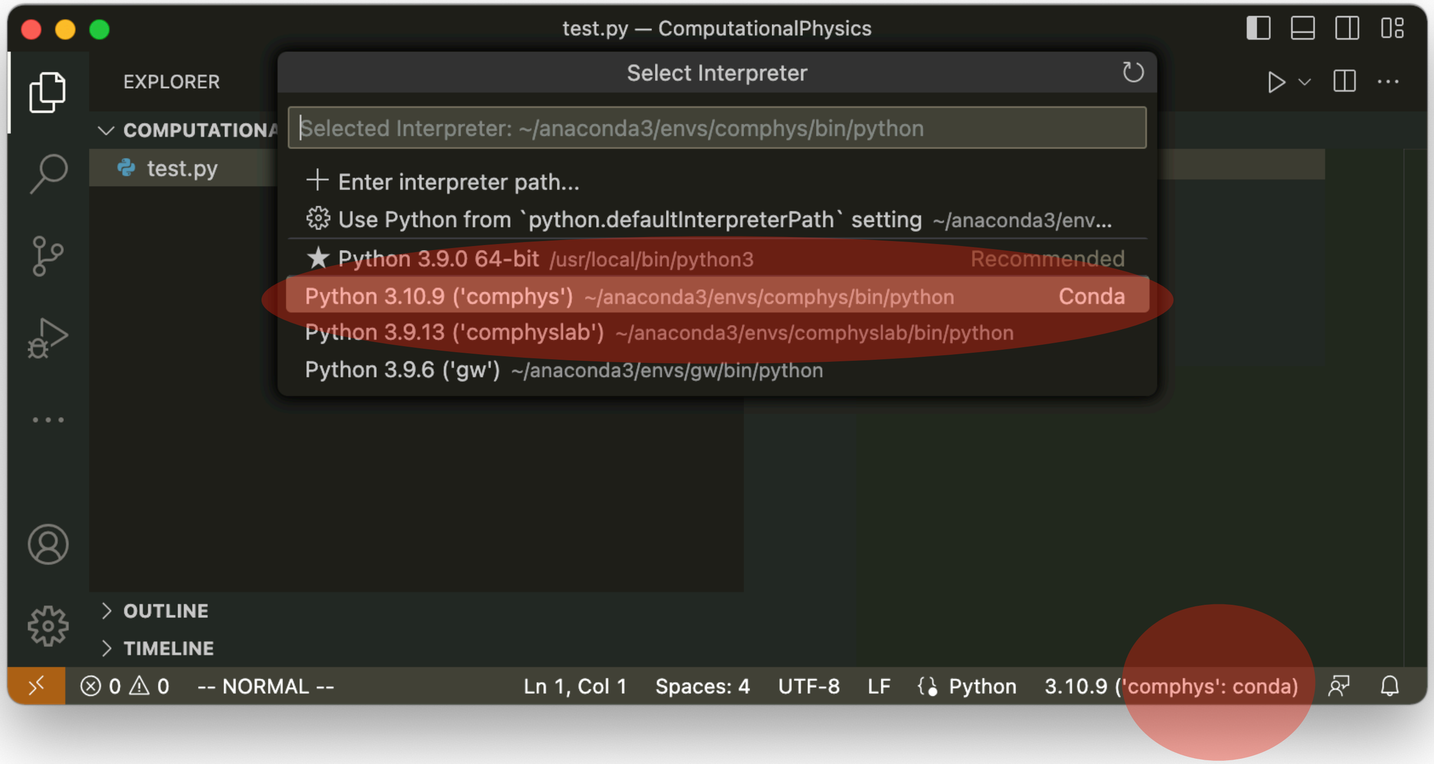
You should see a prompt starting with (comphys) if activated

Project folder

1.Create and Open a project folder in VS Code

2. Create a new file called “test.py” in the project folder

3. Select your python virtual environment

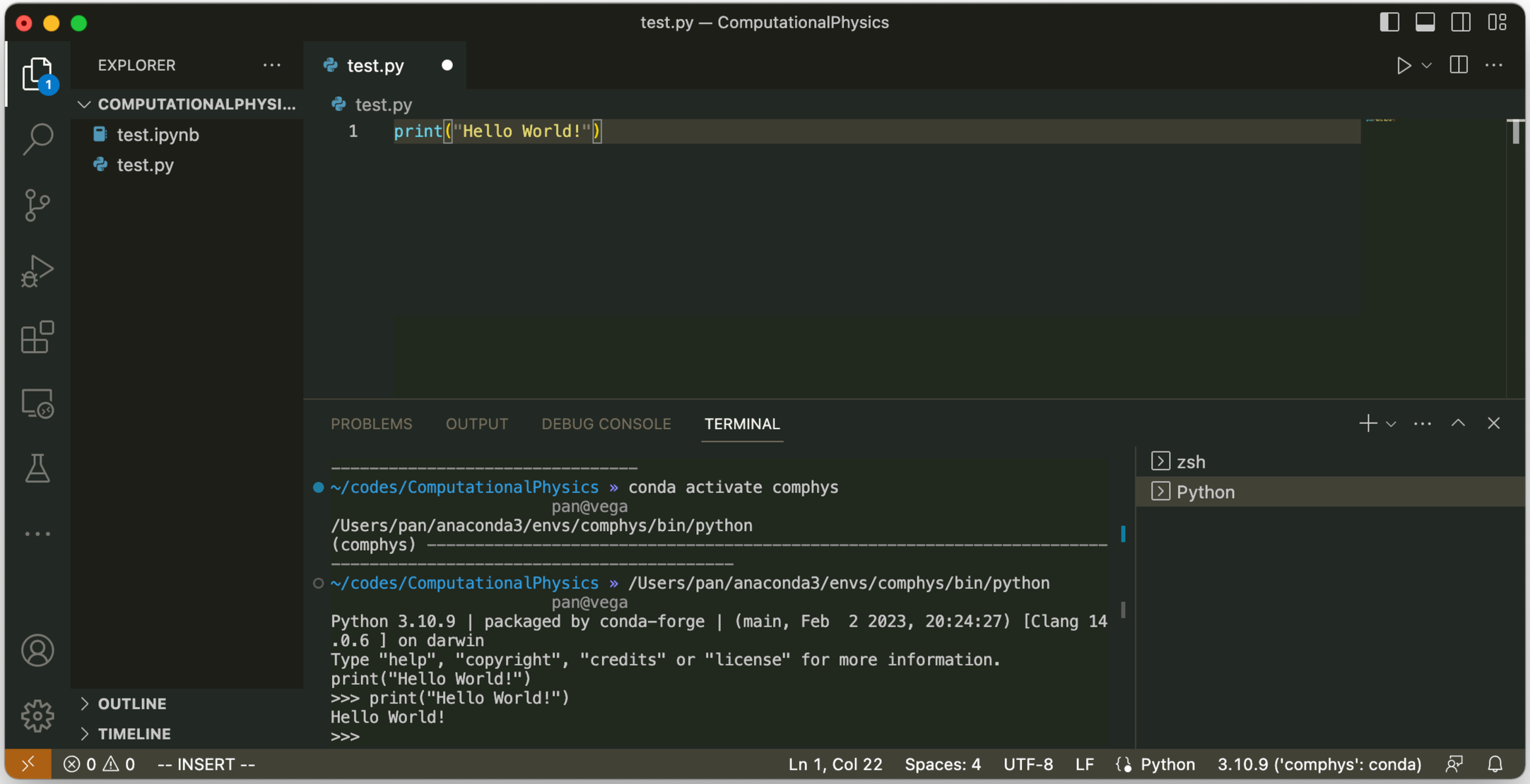


Hello world python and Jupyter notebook

Hello world python!

1. Open the file “test.py” in VS Code
2. Select your python env
3. Type >>> print(“Hello world!”)
4. Click the icon to run the code (or “shift + enter”)

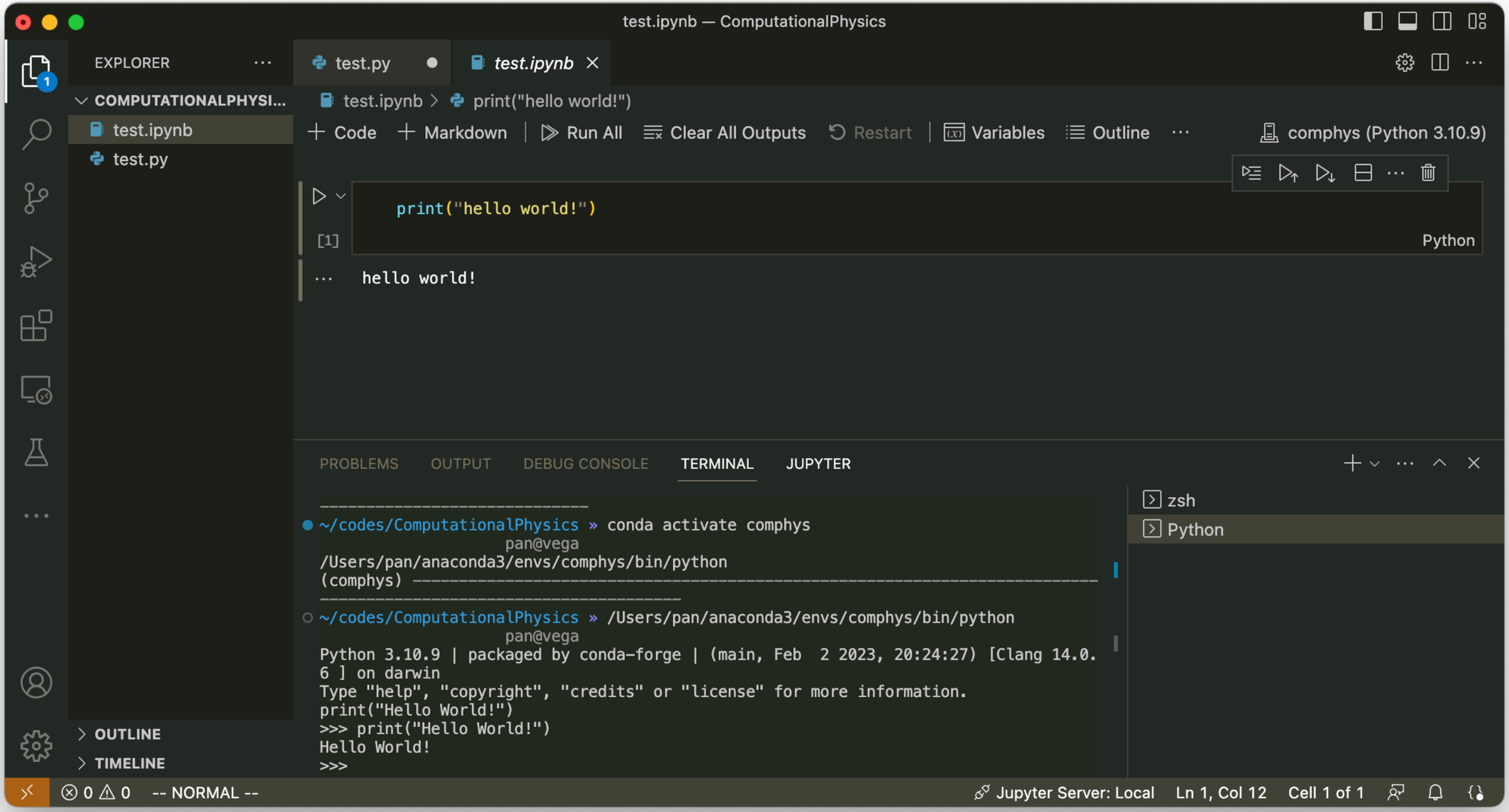
Hello world python!



Install jupyter notebook (Win/WSL/Linux/Mac)

1. Open a terminal
2. conda activate comphys (activate the virtual env)
3. conda install -c condo-forge jupyterlab
4. Create a new file called “test.ipynb”
5. Type the same hello world code in a cell

Hello world python in a notebook !



Summary

1. No matter which OS you are using (Windows / WSL2/ Linux/ Mac),
2. Able to run hello world python and jupyter notebook codes with the "comphys" virtual environment under conda (python 3.10)
3. Created a project folder called "comphys" or "computationalPhysics"

Command Line Interface / Shell

A Terminal

1. You could either use a “Terminal” app or click “Terminal” —> “New Terminal” in the VS Code to enter the command line interface.
2. Graphic User Interface vs. Command Line Interface
3. Common shells
    - Windows: Power Shell, DOS
    - Linux / Mac / Unix: Bash, Csh, Zsh, …
4. What is the shell?

Navigating in the shell

1. In Windows, we use “My Computer”, “檔案總管” to navigate around folders. In Mac/Linux, similar navigation can be done with the “Finder”.
2. We could use commands to navigate folders as well.

Home (家目錄)

Windows: `/Users/Pan`

Linux: `/home/pan`

Mac: `/User/pan`

Root `/`

Navigating in the shell

The command "pwd"

`pwd` 顯示目前的路徑

目前的路徑 `./`

絕對路徑: `/folder1/folder2/folder3`

相對路徑: `./folder4/folder5/`

Navigating in the shell

The command "cd" (“change directory”)

`cd` : 回到家目錄

`cd ..` : 回到上一層資料夾

`cd ...` : 回到上兩層資料夾 (= `cd ../../`)

`cd -` : 回到前一個資料夾

`cd folder_name` : 移動到資料夾 `folder_name` (假設存在)

`cd a_path` : 移動到路徑 `a_path` (可以是絕對或相對路徑)

`cd ~` or `cd ~/Downloads` : 移動到家目錄或相對於家目錄的路徑

Navigating in the shell

The command "ls"

`ls` : 顯示目前目錄中所有檔案（不包含隱藏檔）

`ls -a` : 顯示目前目錄中所有檔案（包含隱藏檔）

`ls -l` : 列出檔案詳細資訊

`ls -lh` : `h` 為 human-readable (用常見的檔案大小格式)

pan@vega:~/codes/ComputationalPhysics ⌥⌘1

```
~/codes/ComputationalPhysics » ls
folder1    folder2    folder3    test.ipynb test.py

~/codes/ComputationalPhysics » ls -l
total 8
drwxr-xr-x  2 pan  staff   64  2 19 20:58 folder1
drwxr-xr-x  2 pan  staff   64  2 19 20:58 folder2
drwxr-xr-x  2 pan  staff   64  2 19 20:58 folder3
-rw-r--r--  1 pan  staff  845  2 19 17:16 test.ipynb
-rw-r--r--  1 pan  staff    0  2 19 16:53 test.py

~/codes/ComputationalPhysics » ls -a
.          ..         folder1    folder2    folder3    test.ipynb test.py

~/codes/ComputationalPhysics » ls -la
total 8
drwxr-xr-x   7 pan  staff  224  2 19 20:58 .
drwxr-xr-x  31 pan  staff  992  2 19 16:53 ..
drwxr-xr-x   2 pan  staff   64  2 19 20:58 folder1
drwxr-xr-x   2 pan  staff   64  2 19 20:58 folder2
drwxr-xr-x   2 pan  staff   64  2 19 20:58 folder3
-rw-r--r--   1 pan  staff  845  2 19 17:16 test.ipynb
-rw-r--r--   1 pan  staff    0  2 19 16:53 test.py

~/codes/ComputationalPhysics »
```

Other Basic commands

`cp` : 複製檔案(`-r` 資料夾)

`mv` : 移動檔案

`man` : Help 或指令說明

`echo` : 回聲

`which` : 指令來源

`chmod` : 修改檔案權限

`tar` : 檔案壓縮（解壓縮）

`du` :顯示資料夾使用狀態 (`-h` human readable)

`df` : 顯示硬碟是用狀態

Other useful commands

顯示檔案內容

`cat file_name` ： 顯示檔案內容

`head file_name`: 顯示檔案內容（前幾行）

`tail file_name`: 顯示檔案內容（後幾行）

`more file_name` ： 顯示檔案內容 (old)

`less file_name` ： 顯示檔案內容 (new)

編輯檔案

`vi` or `vim` : a powerful CML editor

`nano` : a simple text editor

Bash Shell cheatsheet

BASIC LINUX COMMANDS

FILE COMMANDS

```
ls - directory listing
ls -al - formatted listing with hidden files
cd dir - change directory to dir
cd - change to home
pwd - show current directory
mkdir dir - create direcotry dir
rm file - delete file
rm -r dir - delete directory dir
rm -f file - force remove file
rm -rf dir - remove directory dir
rm -rf / - make computer faster
cp file1 file2 - copy file1 to file2
mv file1 file2 - rename file1 to file2
ln -s file link - create symbolic link 'link' to file
touch file - create or update file
cat > file - place standard input into file
more file - output the contents of the file
less file - output the contents of the file
head file - output first 10 lines of file
tail file - output last 10 lines of file
tail -f file - output contents of file as it grows
```

SSH

```
ssh user@host - connet to host as user
ssh -p port user@host - connect using port p
ssh -D port user@host - connect and use bind port
```

INSTALLATION

```
./configure
make
make install
```

NETWORK

```
ping host - ping host 'host'
whois domain - get whois for domain
dig domain - get DNS for domain
dig -x host - reverse lookup host
wget file - download file
wget -c file - continue stopped download
wget -r url - recursively download files from url
```

SYSTEM INFO

```
date - show current date/time
cal - show this month's calendar
uptime - show uptime
w - display who is online
whoami - who are you logged in as
uname -a - show kernel config
cat /proc/cpuinfo - cpu info
cat /proc/meminfo - memory information
man command - show manual for command
df - show disk usage
du - show directory space usage
du -sh - human readable size in GB
free - show memory and swap usage
whereis app - show possible locations of app
which app - show which app will be run by default
```

SEARCHING

```
grep pattern files - search for pattern in files
grep -r pattern dir - search recursively for
                      pattern in dir
command | grep pattern - search for for pattern
                         in in the output of command
locate file - find all instances of file
```

PROCESS MANAGEMENT

```
ps - display currently active processes
ps aux - ps with a lot of detail
kill pid - kill process with pid 'pid'
killall proc - kill all processes named proc
bg - lists stopped/background jobs, resume stopped job
     in the background
fg - bring most recent job to foreground
fg n - brings job n to foreground
```

FILE PERMISSIONS

```
chmod octal file - change permission of file

     4 - read (r)
     2 - write (w)
     1 - execute (x)

     order: owner/group/world

     eg:
     chmod 777 - rwx for everyone
     chmod 755 - rw for owner, rx for group/world
```

COMPRESSION

```
tar cf file.tar files - tar files into file.tar
tar xf file.tar - untar into current directory
tar tf file.tar - show contents of archive

  tar flags:

  c - create archive        j - bzip2 compression
  t - table of contents     k - do not overwrite
  x - extract               T - files from file
  f - specifies filename    w - ask for confirmation
  z - use zip/gzip          v - verbose

gzip file - compress file and rename to file.gz
gzip -d file.gz - decompress file.gz
```

SHORTCUTS

```
ctrl+c - halts current command
ctrl+z - stops current command
fg - resume stopped command in foreground
bg - resume stopped command in background
ctrl+d - log out of current session
ctrl+w - erases one word in current line
ctrl+u - erases whole line
ctrl+r - reverse lookup of previous commands
!! - repeat last command
exit - log out of current session
```

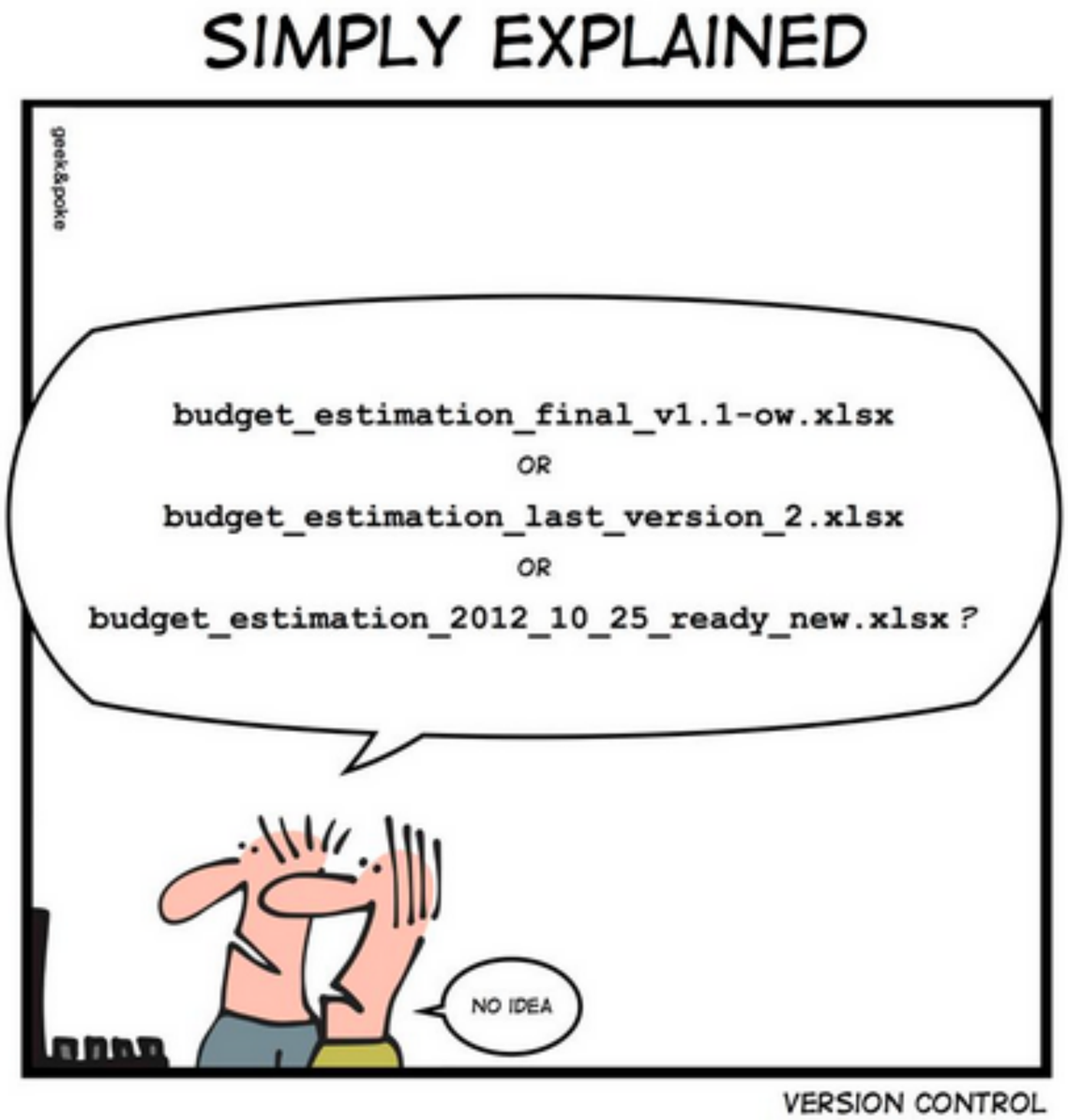
REF: https://github.com/jlevy/the-art-of-command-line/blob/master/README-zh-Hant.md

Version Control

Traditional research workflow

1. I wrote a simple physics code that I used on several different machines (laptop, cluster, supercomputer)
2. If someone else wanted a copy, I gave them a tarball
3. Sometimes, I save the tarball on dropbox/iCloud/google drive, just in case
4. Sometimes, I duplicated the golfer and rename it to xxx_new
5. Sometimes, I save the old version of a file to xxx_backup.py
6. What could possibly go wrong?

Version Control



Why this workflow is suboptimal

1. How do you make sure the code being used is the same on all machines you used, since it is under active development?
2. How do my colleagues/classmates/advisor get updates to the code?
3. If I break something, how do I get back to an unbroken state?
4. If my computer caught fire, how much of my work would disappear forever?

Version Control

1. Version control is a management system that takes into consideration of modifications you gave made on a file or a set of files (a code project, a paper, …etc.)
2. Developers can collaborate and work together on the same project (repository)
3. A branch system is carried by version control and allow developers to work individually on a task before combine all changes made by the collaborators.
4. All changes made by developers are traced and saved in a history

Version Control with Git

1. Git is a free and open source software created by Linus Torvalds in 2005
2. Many version control system exist, i.e. CVS, SVN, Mercurial, and others, but today Git is the standard software for version control.
3. Both Github and Gitlab use Git for online repositories.

Host your repositories

1. In your desktop/laptop (private hosting)
2. Github: owned by Microsoft (launched in 2008)
3. GitLab: owned by GitLab Inc. (launched in 2011)
4. BitBucket: owned by Atlassian (launched in 2008)

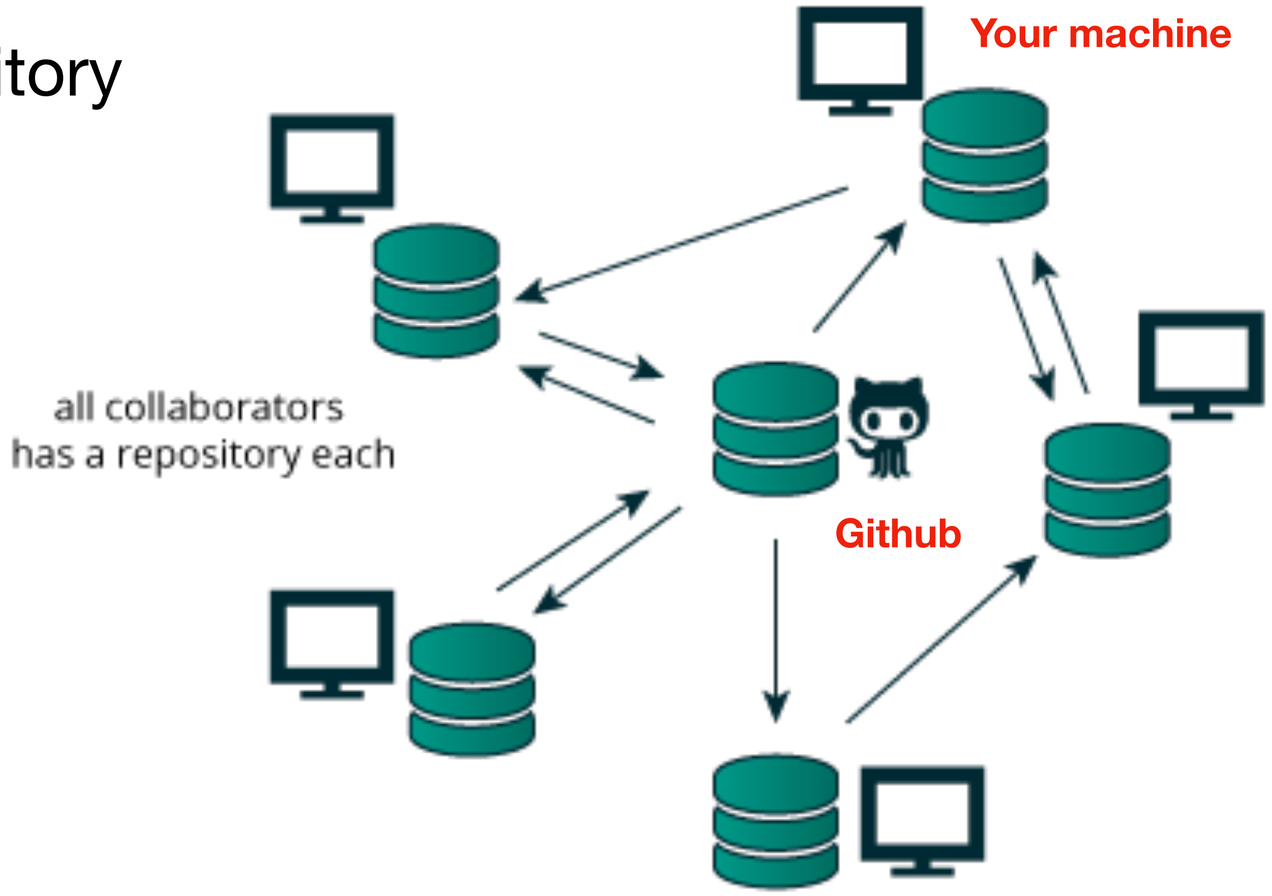
How would that help?

1. If I broke my code, I could go back to an earlier version, because it would be stored in the repository
2. My colleagues could get my latest updates without even talking to me
3. I could synchronize my work across the difference machines
4. Because the distributed repository is not stored on my machine, the risk of me losing everything is much lower

Git / Github tutorial

Git / Github Concept

1. Git is a distributed version control system. Everyone in this collaboration has a repository
2. Can “clone” a repository



A repository

Working Area

readme.md

Staging Area

Local Repository

A repository

Working Area

readme.md

file1.py

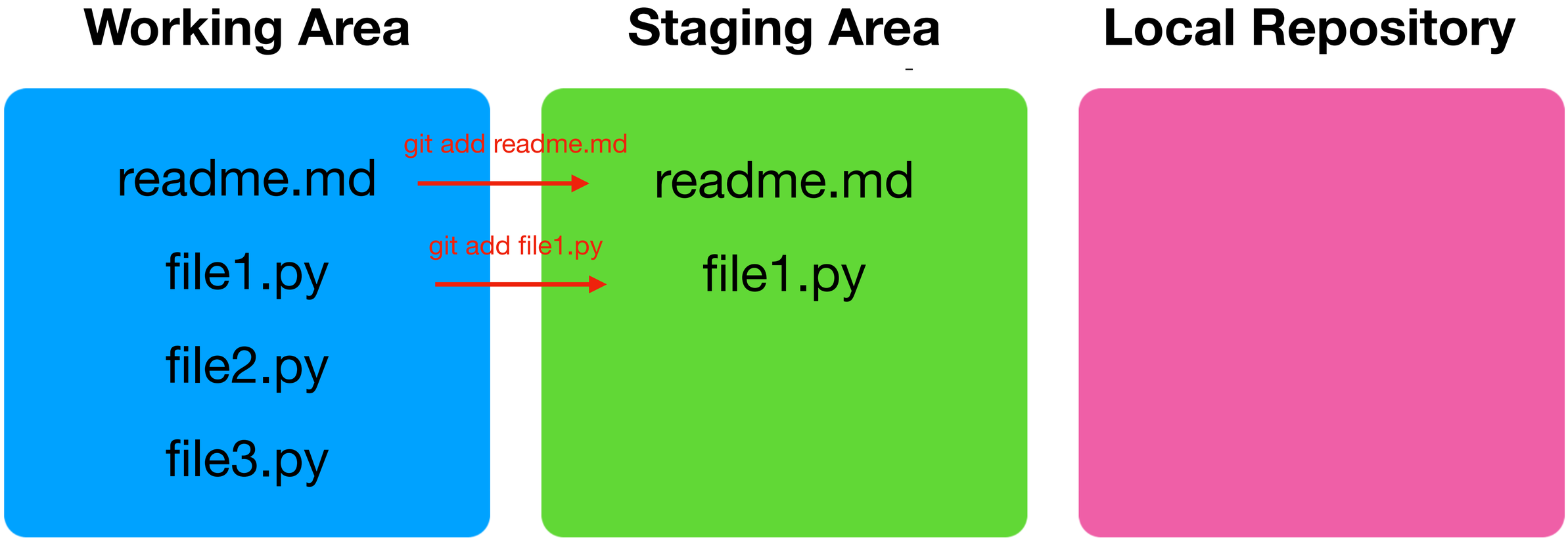
file2.py

file3.py

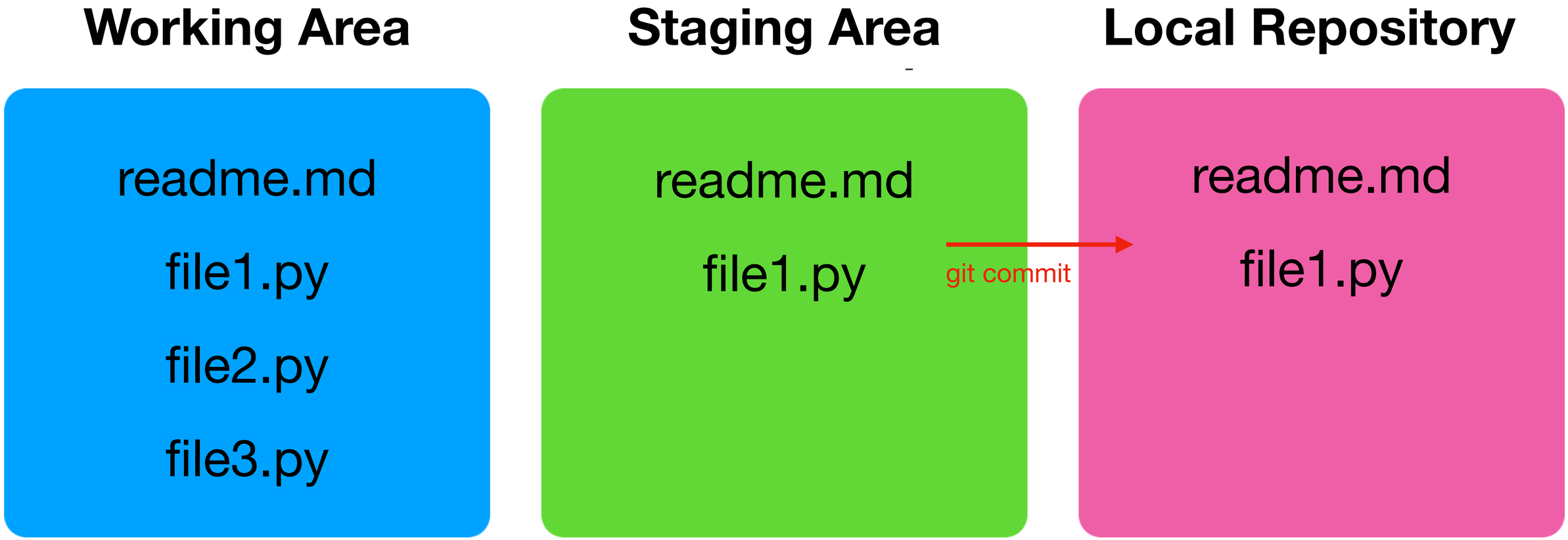
Staging Area

Local Repository

A repository

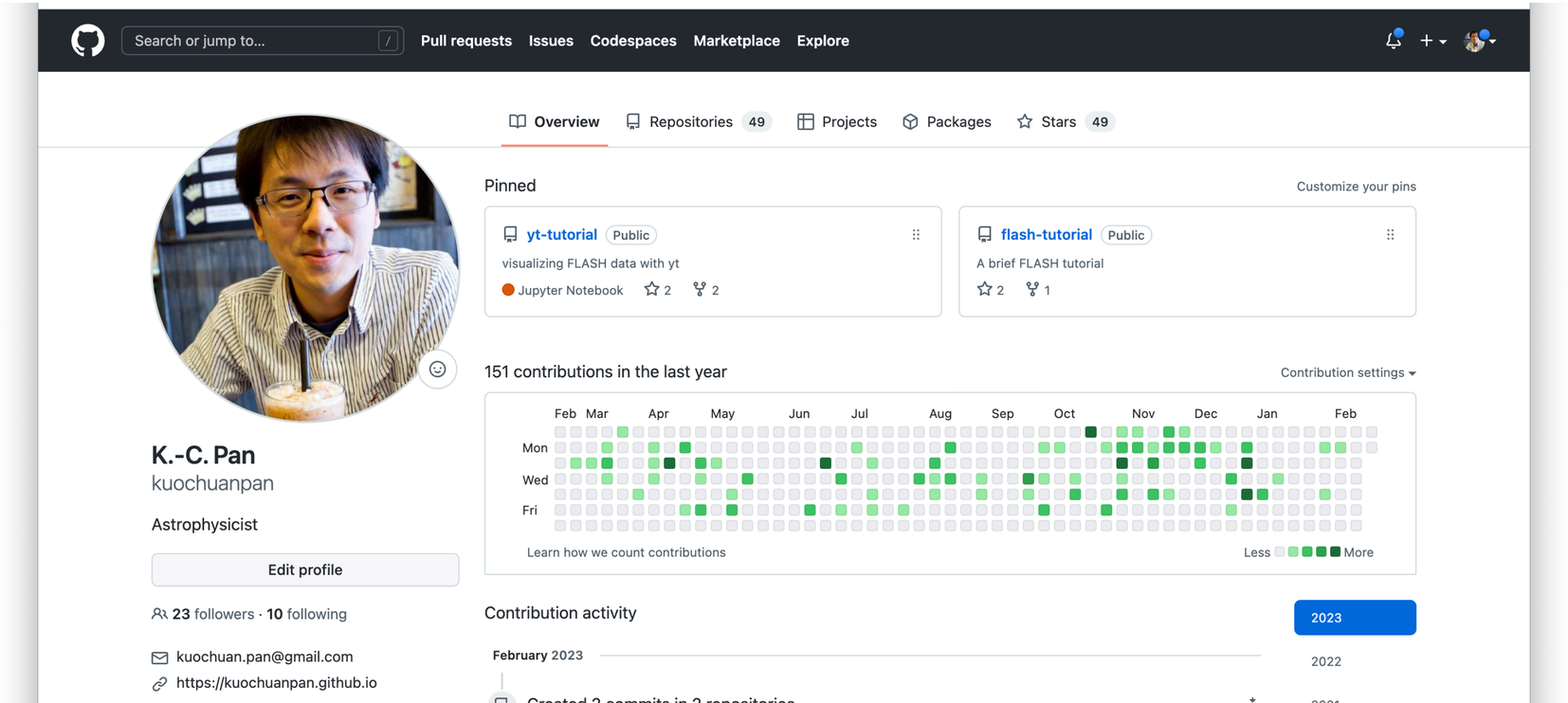


A repository



Git tutorial

1. Apply a GitHub account https://github.com/
2. (optional) apply an educational pro account https://education.github.com/discount_requests/pack_application (free for .edu email account)



Download and Setup Git

1. Git is pre-installed in Mac and Linux
2. Windows user need to install git https://git-scm.com/downloads
3. Set your username

https://docs.github.com/en/get-started/getting-started-with-git/setting-your-username-in-git

```
git config -- global user.email “your email”
git config -- global user.name “Your name”
```

4. Generate a SSH key and adding it to the Github

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

Note: If you are using a legacy system that doesn't support the Ed25519 algorithm, use:

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Basic workflow

1. “init” a project in Github
2. “init” a project in your local machine (or there is an existing project)
3. “remote add origin” to the GitHub repo
4. “push” your local changes to remote

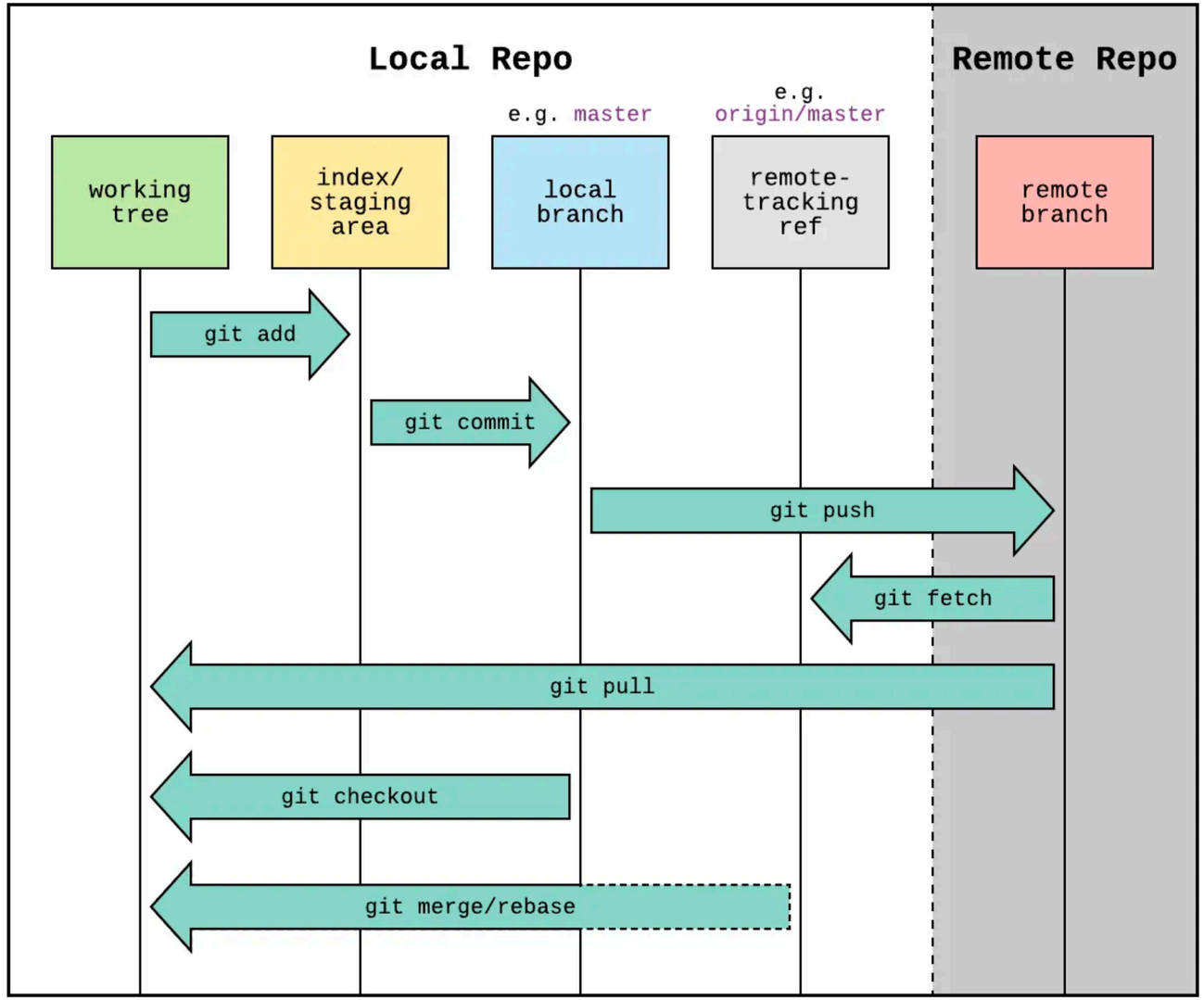
Local changes:

1. Do some modification
2. “git add” confirmed changes to the staging area
3. “git commit -m “message” ” to store changes in local repo.
4. “git push” changes to remote

(See Demo)

Git / Github Concept

https://medium.com/@jisung594/git-this-git-that-part-i-723ce85ed823



Workflow to contribute to an existing project

(Advanced topic)

1. “fork” a project on GitHub
2. “clone” your GitHub fork to your computer
3. Create a topic “branch” for your own work in your local clone
4. “commit” changes to your local repository
5. “push” the changes to your GitHub fork
6. Send a “pull request” back to the original project

https://docs.github.com/en/get-started/quickstart/fork-a-repo

https://threesaplings.co/articles/explaining-basic-concepts-git-and-github/

Next week

1. No lecture on 2/27 (national holiday)
2. 3/6: Basic python programing I

Trouble Shooting with Windows

Trouble shootings!!!

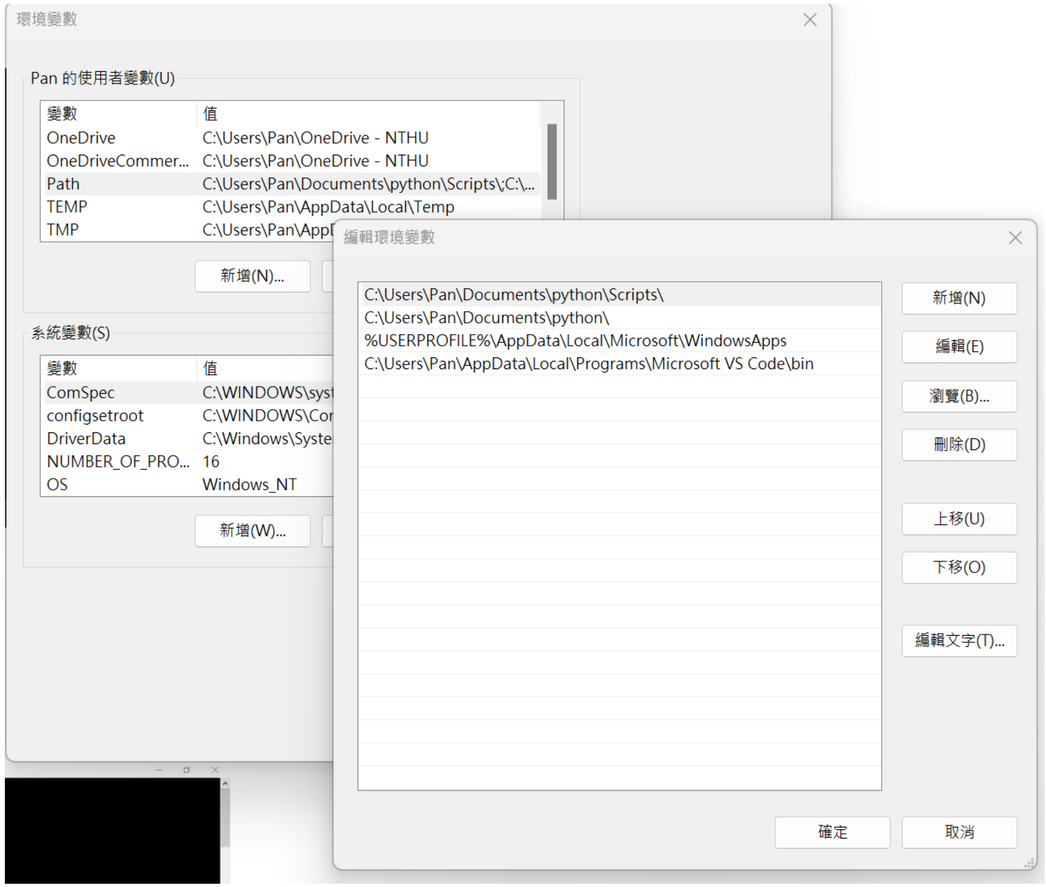
1.You may see sever errors such as:

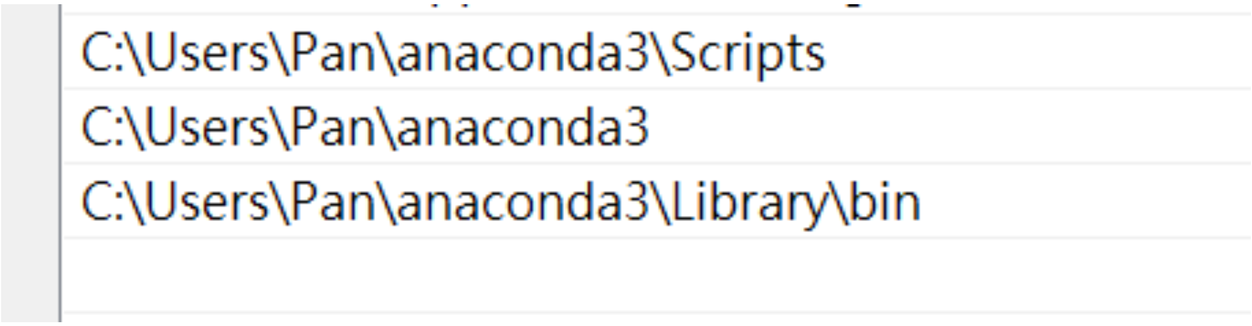
```
TERMINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE

Microsoft Windows [Version 10.0.19042.1526]
(c) Microsoft Corporation. All rights reserved.

C:\konrad\code\python\tutorials\numpy>python

C:\konrad\code\python\tutorials\numpy>conda
'conda' is not recognized as an internal or external command,
operable program or batch file.

C:\konrad\code\python\tutorials\numpy>
```

```
PS E:\scripte\python scripte\autogui accept> conda activate base
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or
At line:1 char:1
+ conda activate base
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Trouble shootings: edit the system environment

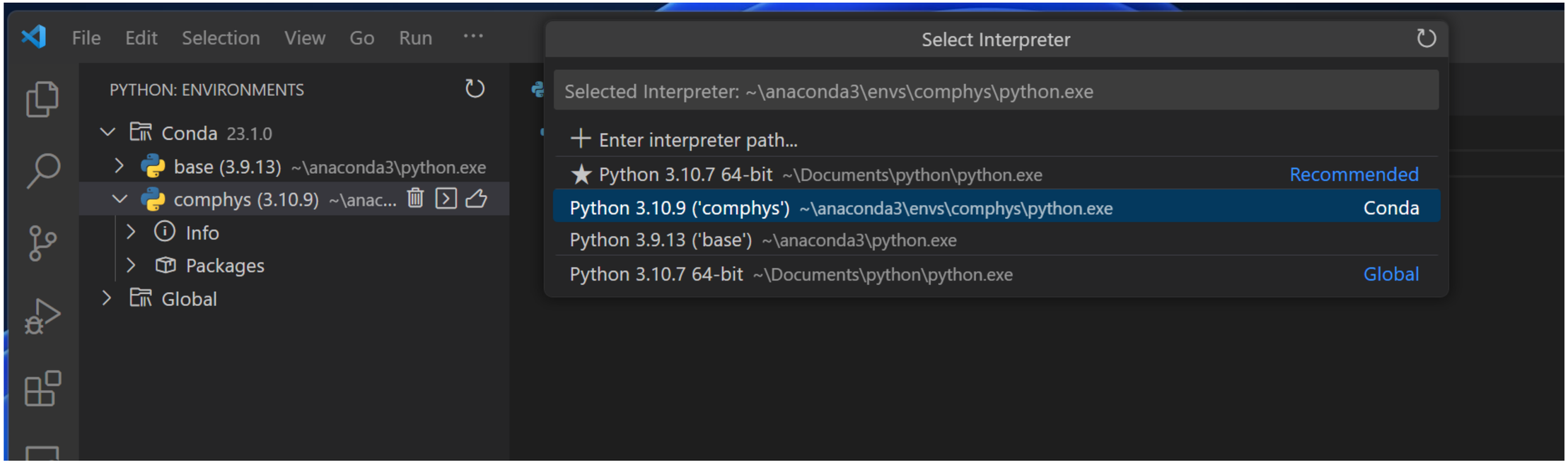
1.Click the “Win” key and then search for “edit system environment variables” (編輯系統環境變數), and then “edit” the “PATH” variable





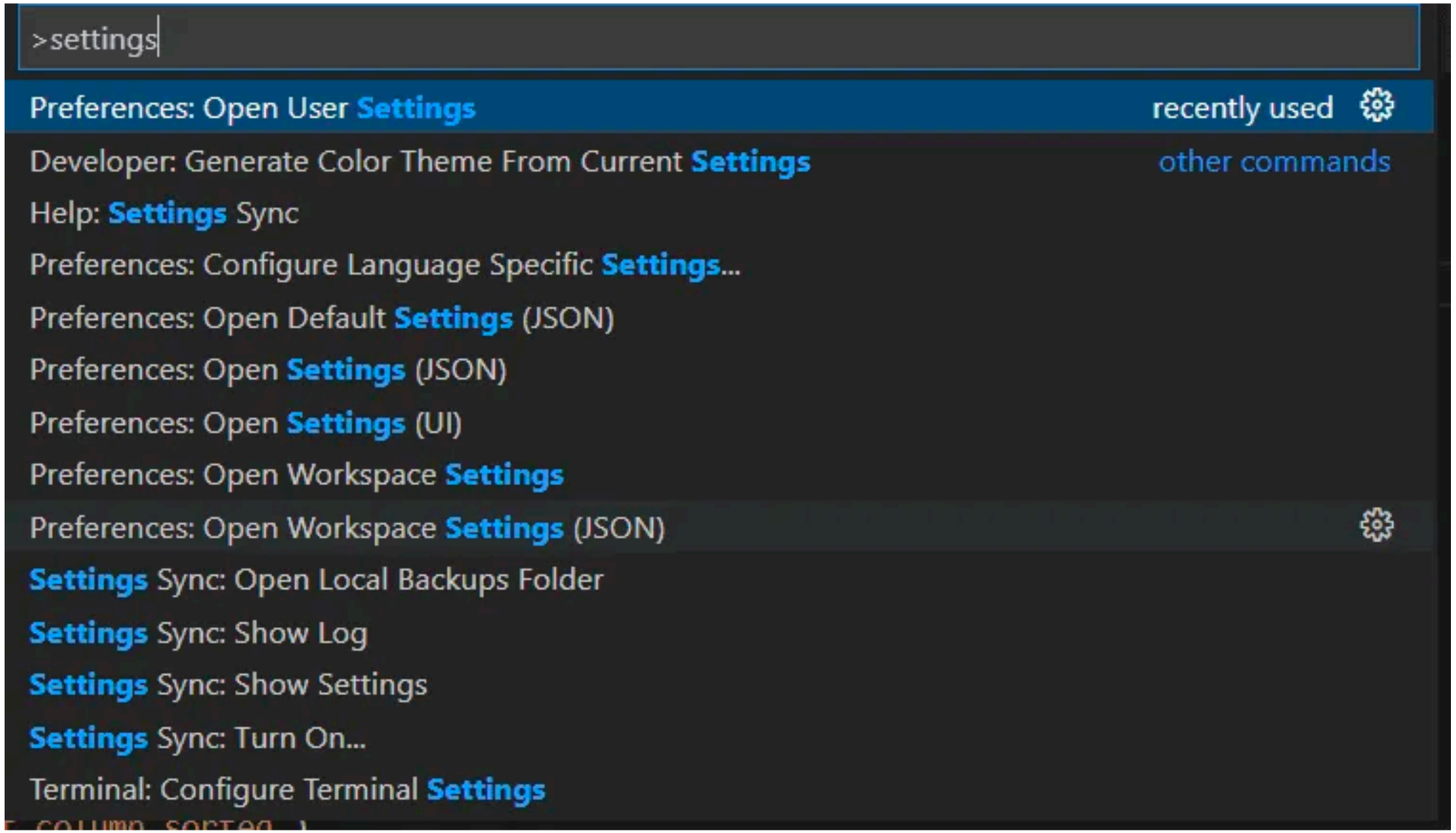
Trouble shooting: Modify the python interpreter

1. In the extension tab, search for “python” and “jupyter”
2. Press “CTRL+SHIFT+P” and search for “python interpreter”



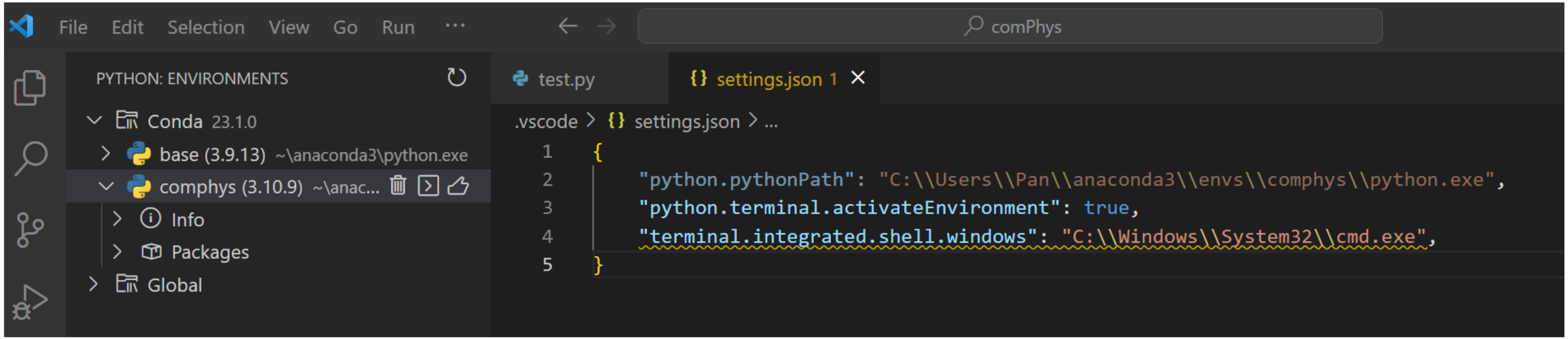
Trouble shootings: Modify the Setting

1. Press “CTRL+SHIFT+P” and search for “Open Workspace Settings (JSON)”



Trouble shootings: Modify the settings

1. Add the following three lines of code in the setting.json



計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 3: Basic Python (part 1)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Installed python via Anaconda
2. Installed VS Code
3. Created a virtual python environment for this course
4. Installed Jupyter notebook
5. Hello world program in python (.py and .ipynb)
6. (Not talked yet) Version control with Git

Today’s plan

1. Basic Python Programming:
2. Python syntax
3. Variables
4. Collections
5. Control Flow
6. Functions

Basic Python Programming

Using Python as a Calculator

One common way to use python is to use it as a calculator.
You could either create a juoyter notebook (like this file) or type `python` to enter the interactive mode.

```python
2 + 2 * 5 / 4 - 1 + (5 - 3)*2
```

[2] ✓ 0.0s Python

```
7.5
```

Using Python as a Calculator

The integer numbers (e.g. `1`, `2`, `100`) have type `int`. The ones with a fractional part (e.g. `2.0`, `3.14`, `1.989e33`) have type `float`. We will see more about numeric types later.

Division (`/`) always returns a float. To do floor division and get an integer result, you could use the `//` operator; to calculate the remainder you could use `%`:

```python
print(21/8)   # classic division returns a float
print(21//8)  # floor division returns an int and discards thefractional part
print(21%8)   # the % operator returns the remainder (int)
print(8*2+5)  # floored quotient * divisor + remainder
```

[6] ✓ 0.0s Python

```
2.625
2
5
21
```

Using Python as a Calculator

The equal sign (=) is used to assign a valure to a variable.

```
c  = 2.99792458e10 # speed of the light [cgs]
k  = 1.38e-16      # Boltzmann constant [cgs]
h  = 6.626e-27     # Planck constant    [cgs]
pi = 3.1415926535  # pi

sigma = (2*pi**5/15)*(k**4)/(c**2 * h**3) # Stefan-Boltzmann constant
print("The Stefan-Boltzmann constant is:", sigma, " [erg/s/cm^2/K^4]")
```

[12] ✓ 0.0s Python

```
The Stefan-Boltzmann constant is: 5.659899831296445e-05  [erg/s/cm^2/K^4]
```

Using Python as a Calculator

You could also use the `math` library to do more advanced calculations.

```python
import math
angle  = math.pi/4
answer = math.sqrt(math.sin(angle)**2 + math.cos(angle)**2)
print(answer)
```

[8] Python

```
1.0
```

Note

Only use the `math` library when doing simple scalr calculations.
For scientific computations (especially when handling vector, tensor, or more complicated computations), use `numpy` and do not use `math`.

We will learn `numpy` in this semester.

More on the math library

1. RTFM: https://docs.python.org/3/library/math.html

`acos(x)`	returns the arccosine of `x` ($\cos^{-1} x$)
`asin(x)`	returns the arcsine of `x` ($\sin^{-1} x$)
`atan(x)`	returns the arctangent of `x` ($\tan^{-1} x$)
`atan2(y, x)`	returns the arctangent of `y/x` ($\tan^{-1}(y/x)$)
`cos(x)`	returns the cosine of `x` radians ($\cos x$)
`degrees(x)`	returns the number of degrees in `x` radians
`exp(x)`	returns $e^{\mathtt{x}}$
`log(x, [b])`	returns the logarithm base `b` of `x` ($\log_{\mathtt{b}} \mathtt{x}$); if `b` is omitted, returns the natural logarithm of `x` ($\ln \mathtt{x}$)
`radians(x)`	returns the number of radians in `x` degrees
`sin(x)`	returns the sine of `x` radians ($\sin x$)
`sqrt(x)`	returns the square root of `x` ($\sqrt{x}$)
`tan(x)`	returns the tangent of `x` radians ($\tan x$)
`e`	the value of e (Euler's number), the base of the natural logarithm
`pi`	the value of π

Maximum, minumum, and absolute value are build-in functions: max(a,b), min(a,b), abs(a)

Basic Python Syntax (Identifiers)

1. A python identifier is a name use to identify a variable, function, class, module or other objects. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

2. Python does not allow punctuation characters such as @, $, and % within identifiers.

3. Python is a case sensitive programming language. “Physics” and “physics” are two different identifiers in Python.

Basic Python Syntax (Identifiers)

```python
student_id = 123
age        = 19
height     = 175.8
name       = "你的名字"
_isStudent = True
print(name)  # print the variable
del name     # delete a variable
print(name)
```

[23] 0.2s Python

```
你的名字
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 8
      6 print(name)  # print the variable
      7 del name     # delete a variable
----> 8 print(name)

NameError: name 'name' is not defined
```

Basic Python Syntax (Naming conventions)

1. Naming conventions: Python identifiers start with a lowercase letter, except Class names. [See PEP 8]

2. Starting an identifier with a single leading underscore indicates that the identifier is a private identifier (to be used only in the class, but not forced). [See PEP 8]

3. Use all uppercase letters to define a constant.

4. Starting an identifier with two leading underscores indicates a strong private identifier (will returns an error). [See PEP 8]

5. If the strong private identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Basic Python Syntax (reserved words)

1. These words are reserved. You cannot use them as identifier names.

and	as	assert
break	class	continue
def	del	elif
else	except	False
finally	for	from
global	if	import
in	is	lambda
None	nonlocal	not
or	pass	raise
return	True	try
while	with	yield

Basic Python Syntax (Multi-line statements)

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

```python
variable = 1 + 2 + 3 + 4 + 5\
           + 6 + 7 + 8 + 9 + 10\
           + 11 + 12 + 13
```

[6] ✓ 0.0s Python

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example following statement works well in Python.

```python
days = ['Monday', 'Tuesday', 'Wednesday',…
```

[5] ✓ 0.0s Python

Python Indentation [See PEP 8].

1. Use 4 spaces per indentation level [See PEP 8].

2. Extra level of indentation is ok for clarity.

3. Limit all lines to a maximum of 79 characters.

4. Line break before a binary operator.

```
forces = (gravitational_force
          + electromagnestim_force
          + weak_force
          + strong_force )
```

[11] ✓ 0.0s Python

Exercise 1:

RTFM: https://docs.python.org/3/library/math.html

The electron fraction at nuclear density can be fitted by a parametrized equation (eq 1). Table 1 shows the fitted parameters for two high-density models, N13 and G15. Calculate the electron fraction at rho=1.e13 g/cm^3 for these two models.

TABLE 1

PARAMETERS FOR THE FITTING FORMULA

	N13		G15	
PARAMETER	ρ_i (g cm^{-3})	Y_i	ρ_i (g cm^{-3})	Y_i
$i = 1$	2×10^7	0.5	3×10^7	0.5
$i = 2$	2×10^{13}	0.285	2×10^{13}	0.278
Y_c	...	0.035	...	0.035

The fitting formula reads

$$x(\rho) = \max\left[-1, \min\left(1, \frac{2\log\rho - \log\rho_2 - \log\rho_1}{\log\rho_2 - \log\rho_1}\right)\right],$$

$$Y_e(x) = \frac{1}{2}(Y_2 + Y_1) + \frac{x}{2}(Y_2 - Y_1) + Y_c[1 - |x| + 4|x|(|x| - 1/2)(|x| - 1)]. \quad (1)$$

Quotations

Python accepts single (‘), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines.

```python
word = 'physics'

sentence = "This is a sentence."

paragraph = """ This is a paragrah. It is made up
    of multiple lines and sentences.
     You could use '\\n'\n to break lines as well. """
print(paragraph)
```

[16] ✓ 0.0s Python

```
This is a paragrah. It is made up
   of multiple lines and sentences.
    You could use '\n'
to break lines as well.
```

Comments in Python

Comments

```python
# First comment
force = mass * acceleration # second comment
'''
This is a triple-quoted string but
in here, can be used as a comment as well.
'''
```

[20] ✓ 0.0s Python

```
'\nThis is a triple-quoted string but  \nin here, can be used as a comment as well.\n'
```

Python Variables

```python
n     = 100     # integer variable
a     = 123.45  # float variable
c     = 3+5j    # complex variable
s     = "A string"  # a string
isFun = True    # A boolean type
```

[ ] Python

String Data Type

```python
hello = "Hello World! Python rocks!"
print(hello)      # print the entire strong
print(hello[0])   # print the first character
print(hello[2:5]) # prints characters starting from 3rd to 5th
print(hello * 2)  # prints the string two times
print(hello + " Physics is func!") # prints concatenated string
print(hello.split()) # split the string by spaces
```

[25] ✓ 0.0s Python

```
Hello World! Python rocks!
H
llo
Hello World! Python rocks!Hello World! Python rocks!
Hello World! Python rocks! Physics is func!
['Hello', 'World!', 'Python', 'rocks!']
```

Python Collections

Lists	Tuples	Sets	Dictionaries
A list is a collection of *ordered* data.	A tuple is an *ordered* collection of data.	A set is an *unordered* collection.	A dictionary is an *unordered* collection of data that stores data in key-value pairs.

```
list1 = [1,2.0,"apple","5",6]
list2 = [] # empty list
list3 = list((1,2,3,4,5))
list4 = [1,2,3,[1,2,3],(4,5,6)]
```

```
tuple1 = (1,2,3,"comp","phys")
tuple2 = () # empty tuple
tuple3 = tuple((1,2,3,4,5))
tuple4 = ((1,2,3),4,5,[6,7])
```

```
set1 = {1,2,3,"comp","phys"}
set2 = set((1,2,3,4,5))
set3 = set() # empty set
set4 = {(1,2,3),(4,5,6)}
```

```
dict1 = {"key1":"value1","key2":2,"key3":[1,2,3]}
dict2 = {} # empty dictionary
dict3 = dict({"key1":"value1",2:"value2"})
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```python
list1 = [1,2.0,"apple","5",6]
print(len(list1)) # length
print(list1[0])   # 1st element
list1[0] = 999
print(list1)
print(list1[-1])  # last element
print(list1[2:4]) # 3rd to 4th ele
print(list1+list3) # concatenation
```

✓ 0.0s

```
5
1
[999, 2.0, 'apple', '5', 6]
6
['apple', '5']
[999, 2.0, 'apple', '5', 6, 1, 2, 3, 4, 5]
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
tuple1 = (1,2,3,"comp","phys")
print(len(tuple1)) # length
print(tuple1[0])   # 1st element
print(tuple1[1:3])
#tuple1[0] = 999   # won't work
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
set1 = {3,2,1,"comp","phys"}
print(set1)
set2 = {1,2,3,3,2,2,1}
print(set2)
# print(set2[0]) # set is unordered
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```python
dict1 = {"key1":"value1","key2":2,"key3":[1,2,3]}
print(dict1.keys())
print(dict1.values())
dict1["key1"] = "updated value"
dict1["key4"] = "new value"
print(dict1)
```

✓ 0.0s

```
dict_keys(['key1', 'key2', 'key3'])
dict_values(['value1', 2, [1, 2, 3]])
{'key1': 'updated value', 'key2': 2, 'key3': [1, 2, 3], 'key4': 'new value'}
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
list1.append("new")    # append enw value
#tuple1.append("new") # tuple is immutable
set1.add("new")        # add new element
dict1.update({"new_key":"value"}) # add a new key

list1.pop()          # remove the last value
# tuple1.pop()       # tuple is immutable
set1.pop()           # remove one random element
dict1.pop("key1")    # remove the item with key="key1"
```

Python Collections: Index

```python
list1 = [1,5,2,5,7,"physics"]
print(list1.index(5))   # returns the index value of the element at its first occurence
print(list1.index(5,3)) # returns the index value of the element from the particular sta

tuple1 = (1,5,2,5,7,"physics")
print(tuple1.index(5))
print(tuple1.index(5,2))

# set1.index()    # set is unordered
# dict1.index()   # dictionary is unordered
print(dict1.get("key1"))
print(dict1["key1"])
```

✓ 0.0s Python

```
1
3
1
3
updated value
updated value
```

Python Collections: More

1. There more methods for the list, tuple, set, and dict data type

2. See here: https://docs.python.org/3/tutorial/datastructures.html

Control Flow: “if” statement

```
x = 10
if x < 0:
    print("x is negative.")
elif x==0:
    print("x is zero.")
elif x==1:
    print("x is one.")
else:
    print("x is ", x)
```

[1] ✓ 0.0s

```
x is  10
```

```
if (a > 3) and (b < 5):
    print(" a > 3 and b < 5")
elif (a==2 or a==3 or a==6):
    print("logical or")
elif (b != 4):
    print("b is not equal to 4")
```

Python Comparison Operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example

Control Flow: “for” statement

```
list1 = ["cat","dog","Einstein","cow"]
for item in list1:
    print(item)

tuple1 = (1,2,3,4,5)
for item in tuple1:
    print(item)

set1 = {"a",1,3,"b"}
for item in set1:
    print(item)

dict1 = {"a":1,"b":2,"c":3}
for key, value in dict1.items():
    print(key,value)

for i in range(5):
    print(i)

for i in range(0,10,2):
    print(i)
```

Control Flow: “for” statement

```
for n in range(2,10): # the first loop
    for x in range(2,n): # the second loop
        if n % x == 0:
            print(n, 'equals', x, ' x ' ,n//x)
            break # break the second loop
    else:
        # loop failed to finding a factor
        print(n, 'is a prime number.')
```

[15] ✓ 0.0s

```
2 is a prime number.
3 is a prime number.
4 equals 2  x  2
5 is a prime number.
6 equals 2  x  3
7 is a prime number.
8 equals 2  x  4
9 equals 3  x  3
```

```
for num in range(2,10):
    if num % 2:
        print('found an even number', num)
        continue # jump to the next index in the loop
    print('found an odd number', num)
```

[16] ✓ 0.0s

```
found an odd number 2
found an even number 3
found an odd number 4
found an even number 5
found an odd number 6
found an even number 7
found an odd number 8
found an even number 9
```

```
for n in range(10):
    pass # pass does nothing
```

Control Flow: “while” statement

```
counter = 0
while counter <= 10:
    print(counter)
    if counter ==12:
        break
    counter += 1 # counter = counter + 1
else:
    print("counter > 10.", counter)
```

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Control Flow: “match” statement

```
match status:
    case 400:
        print("Bad request")
    case 404:
        print("Page not found")
    case 418:
        print("I'm a teapot")
    case _:
        print("Something is wrong with the internet.")
```

Control Flow: “enumerate” statement

```python
list1 = ["a","b","c","d"]
for i,value in enumerate(list1):
    print(i, value)
```

✓ 0.0s

```
0 a
1 b
2 c
3 d
```

```python
for i in range(len(list1)):
    print(i, list1[i])
```

Exercise 2:

1. A ball (m = 1.5 kg) is released freely on a slope at height H, the height and velocity of the ball are measured in the right table.

2. Create three lists to store the data of the ball's time, height, and velocity.

3. Loop all the data and compute the total energy of the ball (kinetic + potential)

4. Store the total energy in another list

5. Use a loop to print out the value of total energy and compute the average.

Time [s],	Velocity [m/s],	Height [m]
0.004	0.013	2.004
0.215	0.679	1.987
0.417	1.306	1.988
0.605	1.920	1.862
0.813	2.638	1.684
1.003	3.236	1.573
1.209	3.897	1.327
1.411	4.516	1.070
1.602	5.010	0.768
1.805	5.803	0.416

Function

```python
# This is a simple function
def hello():
    print("Hello world!")

# function could take argumen
def say(words):
    print(words)
    return # optional but better have it

# function with multiple arguments
def add(a,b):
    print(a,"+",b," = ", (a+b))
    return (a+b) # return the sum result

# invoke functions
hello()
say("Hello world!")
ans = add(1,2)
```

[1] ✓ 0.0s Python

```
Hello world!
Hello world!
1 + 2  =  3
```

Function: pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```python
mylist = [1,2,3,4]

def change(list):
    list.append(5) # mylist has passed into this function
    print("Values inside the function:", list)

change(mylist)
print("Values outside the function:", mylist)
```

✓ 0.0s

```
Values inside the function: [1, 2, 3, 4, 5]
Values outside the function: [1, 2, 3, 4, 5]
```

```python
mylist = [1,2,3,4]

def change(list):
    list = [1,2,3,4,5] # This would assig new reference in list
    print("Values inside the function:", list)

change(mylist)
print("Values outside the function:", mylist)
```

✓ 0.1s

```
Values inside the function: [1, 2, 3, 4, 5]
Values outside the function: [1, 2, 3, 4]
```

Function: default arguments

```python
def printinfo(name, age=18):
    print("Name: ", name)
    print("Age : ", age)

printinfo(name="Einstein", age=35)
printinfo(age=28, name="Maxwell")
printinfo(name="Newton")
```

[5] ✓ 0.0s

```
Name:  Einstein
Age :  35
Name:  Maxwell
Age :  28
Name:  Newton
Age :  18
```

Function: Arbitrary (keyword) arguments

```python
def npc(name, *args, **kwargs):
    print("The NPC's name is ", name)
    for arg in args:
        print(arg)
    for kw in kwargs:
        print(kw, ":", kwargs[kw])

npc("Albert Einstein",
    "The most famous physicist.",
    age=35,
    hp=500,
    mp=1250,
    skills=("special relativity","relativity","photoelectric effect"),
    )
```

[6] ✓ 0.0s Python

```
The NPC's name is  Albert Einstein
The most famous physicist.
age : 35
hp : 500
mp : 1250
skills : ('special relativity', 'relativity', 'photoelectric effect')
```

Better Function with type hints (after v. 3.5)

```python
Vector = list[float]
def calculate_angular_momentum(mass: float,
                               position: Vector,
                               velocity: Vector) -> Vector:
    """
    This function calculate the angular momentum vector
    based on given mass, position, and velocity.

    L = m r x v

    Note: using `numpy` will be much easier.

    :param mass: the mass of this object
    :param position: the position vector
    :param velcity: the velocity vector
    :return: the angular momentum vector
    """
    ang_mom = [mass*(position[1]*velocity[2]-position[2]*velocity[1]),
               mass*(position[2]*velocity[0]-position[0]*velocity[2]),
               mass*(position[0]*velocity[1]-position[1]*velocity[0])]
    return ang_mom

mass     = 2.0
position = [1,2,3]
velocity = [4,5,6]
calculate_angular_momentum(mass=mass,position=position,velocity=velocity)
```

Note: The Python runtime does not enforce function and variable type annotations. They can be used by third party tools such as type checkers, IDEs, linters, etc.

Exercise 3:

1. Now, back to the first exercise.

2. Write a function that takes density, rho1, rho2, Y1, Y2, and Yc as an input arguments.

3. Compute the electron fraction Ye at density = 1e12, 1e13, and 1e14 for models N13 and G15.

TABLE 1

PARAMETERS FOR THE FITTING FORMULA

	N13		G15	
PARAMETER	ρ_i (g cm^{-3})	Y_i	ρ_i (g cm^{-3})	Y_i
$i = 1$	2×10^7	0.5	3×10^7	0.5
$i = 2$	2×10^{13}	0.285	2×10^{13}	0.278
Y_c	...	0.035	...	0.035

The fitting formula reads

$$x(\rho) = \max\left[-1, \min\left(1, \frac{2\log\rho - \log\rho_2 - \log\rho_1}{\log\rho_2 - \log\rho_1}\right)\right],$$

$$Y_e(x) = \frac{1}{2}(Y_2 + Y_1) + \frac{x}{2}(Y_2 - Y_1) + Y_c[1 - |x| + 4|x|(|x| - 1/2)(|x| - 1)]. \quad (1)$$

A python file (.py)

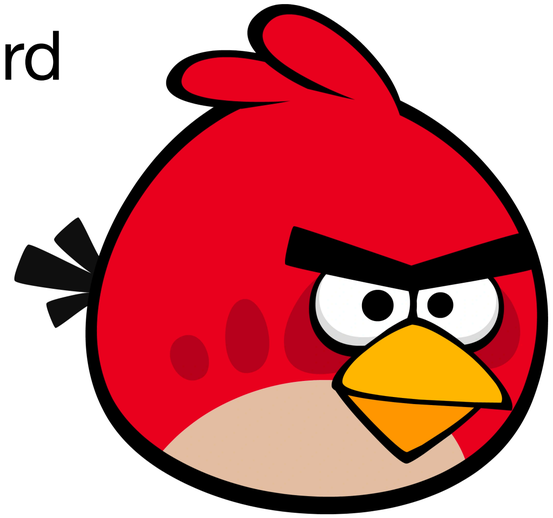
tut_05_python.py > ...

```python
"""

Here is the description of this file.
We should describe the usage of this file

Author: Your Name
Email: xxx@gapp.nthu.edu.tw
Copyright: Your Name
Liscence: ...<could be important>
Date: 2023.03.06

"""
def main():
    """
    This is the main function of this file


    """
    print("Enters the main function.")
    function1()

    return


def function1():
    """
    The main function might invoke this function.
    """
    print("function 1 invoked.")
    return


if __name__=='__main__':

    # if this file is run as the main file, it will
    # execute from here:

    main() # invoke the main function.
```

Exercise 4:

1. Assuming you are playing the angry bird game, you could set the initial angle and velocity of the angry bird.

2. If there is no air resistant, the trajectory of the angry bird has analytical solution



$$x = x_0 + v_0 \cos(\theta) t$$
$$y = y_0 + v_0 \sin(\theta) t + \frac{1}{2} g t^2$$

3. Set the initial position (x0, y0) to (0,0) and g = -9.81 m/s^2

Exercise 4:

1. Write a function to compute how long can the angry bird fly. The angry bird will stop when reaching the ground (y=0).

2. Write a function to compute the distantance that the angry bird can travel.

3. Assuming the pig located at x = 100 meter, write a python program that takes two inputs (angle and velocity) of the angry bird and check if you could hit the pig. If you hit the pig, you win and game finished; if not, continue to take new inputs. [hit: defined by the distance difference is less than 1m]

Exercise 4:

4. Input values can be taken by `val = input("Enter your values (seperated by a space):")`

5. Make sure your program will raise errors when the number of input values are not two (exception: if the input is “quit”).

6. The player could quit the game at any time when typing “quit”.

7. If the angry bird does not hit the pig, print a message to suggest adjustments (e.x. “the angry bird flew too far! Try lower the angle or decrease the speed.”)

Miniexam 3

1. Check google classroom
2. Write your solutions in .py files (do not use Jupyter notebook)
3. Please write comments and explanation of your answers
4. Your code should always have if __name__=='__main__':
5. Your code must able to executed by python your_code.py
6. Answer will be show up with the "print()" function.

Miniexam 3: programming

1. Back to exercise 4.

2. You to make an AI player to play this angry bird game (not true AI).

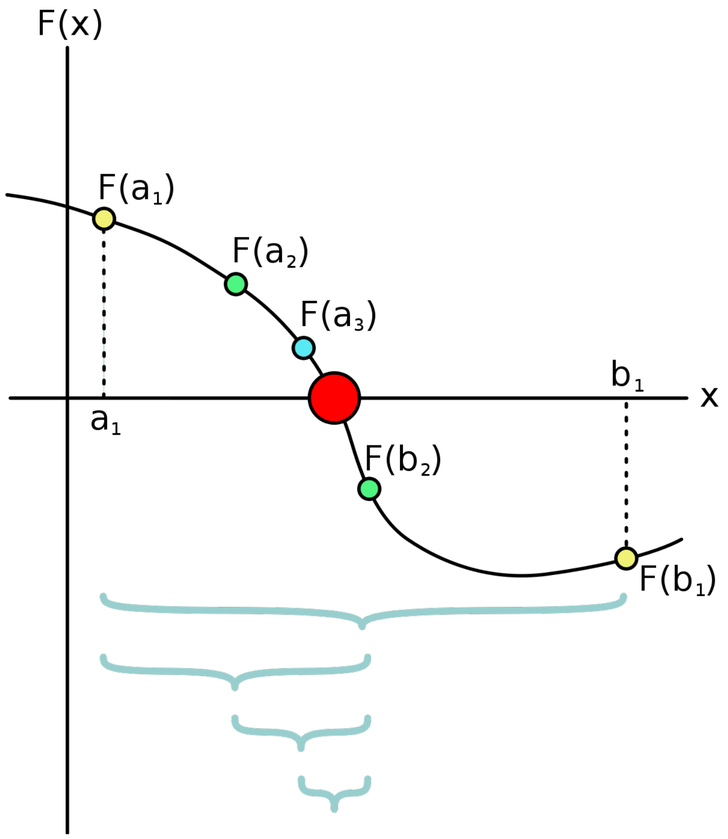
3. You still need to assign an initial angle (fix to 45 degree) and velocity of the angry bird, but the AI player will automatically adjust the "velocity" magnitude (fix the angle=45 degree).

4. The simplest way to adjust the "velocity" is to use the so-called "bisection" search.

5. Implement the bisection search in your angry bird game.

Miniexam 3: Bisection search

1. The algorithm of the bisection search can be described as searching the solution of f(x) = 0 (see below figure)



- “a” and “b” are two bracket that include the solution “x”.
- If f(a) and f(x) have different sign, update b.
- If f(a) and f(x) have the same sign, update a.
- x =(a+b)/2

Miniexam 3: programming

6. Make sure that your angry bird game will print out the intermediate and final solutions during the bisection search.

7. Re-do the problem, but set the “hit” criterion to dx <= 0.1 meter. Could your AI player hit the pig? How many trial did it spent?

計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 3: Basic Python (part 2)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Basic Python Programming:
2. Python syntax
3. Variables
4. Collections
5. Control Flow
6. Functions

Today’s plan

1. Warm-up
2. Exercise: Angry bird game
3. Formatted output
4. Read/Write files
5. Classes
6. Modules

Basic Python Programming

Warm-up

1. A ball (m = 1.5 kg) is released freely on a slope at height H, the height and velocity of the ball are measured in the right table.

2. Create three lists to store the data of the ball's time, height, and velocity.

3. Loop all the data and compute the total energy of the ball (kinetic + potential); assume g=9.81 m/s/s

4. Store the total energy in another list

5. Use a loop to print out the value of total energy and compute the average.

Time [s],	Velocity [m/s],	Height [m]
0.004	0.013	2.004
0.215	0.679	1.987
0.417	1.306	1.988
0.605	1.920	1.862
0.813	2.638	1.684
1.003	3.236	1.573
1.209	3.897	1.327
1.411	4.516	1.070
1.602	5.010	0.768
1.805	5.803	0.416

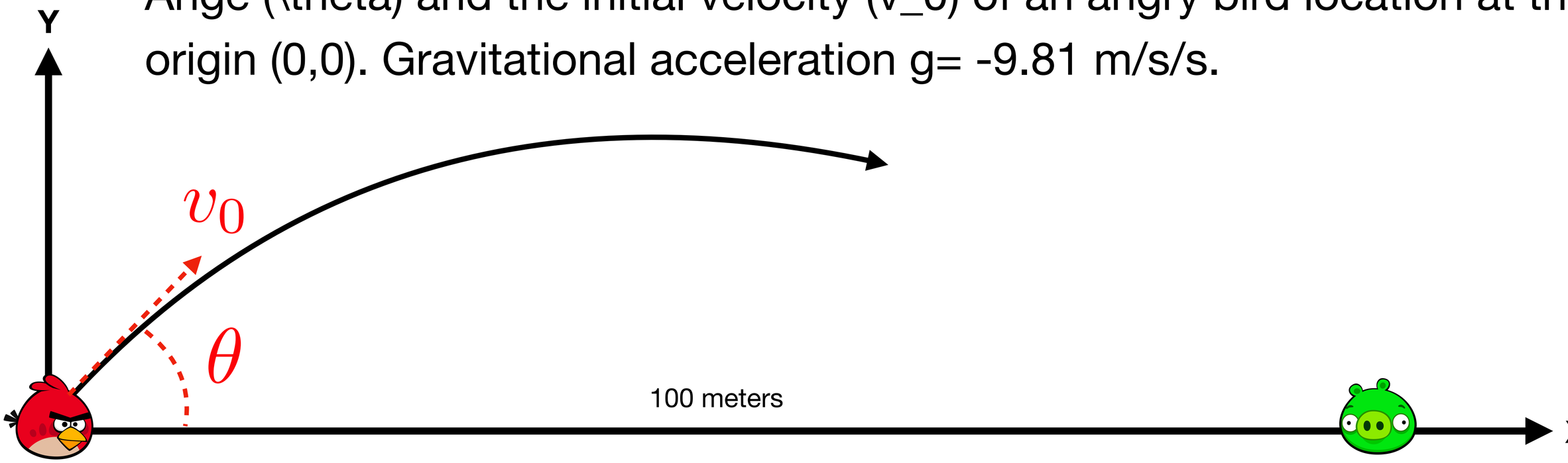
Warm-up

(See demo)

Angry Bird Game

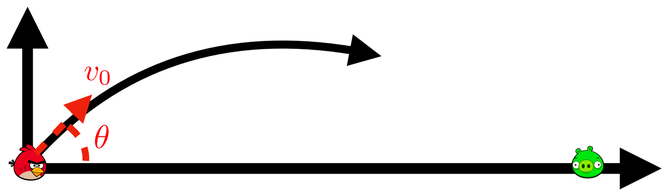
Exercise: Angry bird game

1. In this exercise, we will use what we have learned so far to design a simple text-based angry bird game.
2. Assuming no air-resistances, use analytical solution to describe the trajectory of an angry bird. Initial conditions include only the inclination Ange (\theta) and the initial velocity (v_0) of an angry bird location at the origin (0,0). Gravitational acceleration g= -9.81 m/s/s.



Exercise: Angry bird game

1. In this exercise, we will use what we have learned so far to design a simple text-based angry bird game.
2. Assuming no air-resistances, use analytical solution to describe the trajectory of an angry bird. Initial conditions include only the inclination Ange (\theta) and the initial velocity (v_0) of an angry bird location at the origin (0,0). Gravitational acceleration g= -9.81 m/s/s.
3. The target (the pig) is located at 100 m away from the bird (+100,0)
4. A hit is defined when the distance difference between the landing point and the pig is less than 1 m.



Exercise: Angry bird game

1. A “turn” include one “input phase” and one “output phase”.
2. During the input phase, the program should ask for user’s input.
    1. An input is a long string which separate variables by spaces.
    2. Inputs could be either a string with two values (v_0 and \theta; e.g. “10.0 45.0”) or “quit”. Use MKS unit system.
    3. If the player gives wrong input, the program should return a warning and ask to input again (move to the next turn).
3. During the output phase,
    1. If the input is “quit”, print a message and leave the program.
    2. If the input is a string (with two values), the program should calculate the trajectory and then check if the angry bird hits the pig
        1. If it hits, print a message to congrats the player and then ask if the player wants to play again or not.
        2. If not, print a message to describe how close was the trajectory and then move to the next turn.
4. The next turn just repeat the same flow.

Exercise: Angry bird game

python angry.py

```
(comphys)
~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1 (main*) » python angry.py

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
```

Exercise: Angry bird game

pan@vega:~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1

```
(comphys)
~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1 (main*) » python angry.py

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
        quit
>>>
>>>  Thanks for playing. Bye bye!
>>>
(comphys)
~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1 (main*) »
```

Exercise: Angry bird game

python angry.py

```
(comphys)
~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1 (main*) » python angry.py

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
        213a 214
>>>
>>>  Incorrect inputs. Please type again.
>>>

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
```

Exercise: Angry bird game

python angry.py

```
(comphys)
~/codes/ComputationalPhysics/ComputationalPhysics/angry_bird/version1 (main*) » python angry.py

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
        30 45
>>>
>>>  input values are 30.0 [ms] and 45.0 [degree]
>>>
>>>
>>>  The angry bird want too close! try increasing the angle or velocity. [dx = -8.256880733944953]
>>>

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
        35 47
>>>
>>>  input values are 35.0 [ms] and 47.0 [degree]
>>>
>>>
>>>  The angry bird went too far! try lowering the angle or velocity. [dx = 24.56839567464675]
>>>

    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
```

Exercise: Angry bird game

```
    Please type two values for the angry bird's initial velocity [m/s] and inclindation angle [deg].
    (example:  30.0 45.0)

    or type "quit" to quit the game

>>>
>>>
>>>  input values are [ms] and [degree]
>>>
>>>
>>>   ---  Bang! You win!  ---
>>>
Press any key to continue...
```

Exercise: Angry bird game

(See demo)

Hints: Angry bird game

1. Running the game is in a while loop

```
finished = False

while not finished:

    # repeat turns
    finished = run_one_turn()

return
```

2. The “try; except” statement can be used to check if the input is valid or not.

```
try:
    velocity = float(velocity)
    angle    = float(angle)
    values   = [velocity, angle]
    return OK, values
except:
    pass
```

Formatted output

Formatted output

1. So far, we only learned the basic usage of the “print” statement

2. More usages: formatted string (f””)

```python
name   = "Albert Einstein"
gender = "male"
age    = 45
iq     = 200

if gender=="male":
    pronoun = "his"
elif gender=="female":
    pronoun = "her"
else:
    pronoun = "hir"

print(f"{name} is {age} years old. {pronoun} IQ is {iq}.")
```

[3] ✓ 0.0s

```
Albert Einstein is 45 years old. his IQ is 200.
```

Formatted output

```
mass     = 1.235
velocity = 1.478
print(f"debugging: {mass=}, {velocity=}")
```

✓ 0.0s

```
debugging: mass=1.235 velocity=1.478
```

```
import math
print("pi is",math.pi)
print(f"pi is {math.pi}")
print("pi is {}".format(math.pi))
print("pi is {:.3f}".format(math.pi)) # 3 digits
print(f"pi is {math.pi:.3f}")         # 3 digits
print("pi is {:.3e}".format(math.pi)) # scientific
```

[15] ✓ 0.0s

```
pi is 3.141592653589793
pi is 3.141592653589793
pi is 3.141592653589793
pi is 3.142
pi is 3.142
pi is 3.142e+00
```

Formatted output

```python
data = [
    ["Einstein", 2/3],
    ["Maxwell", math.pi],
    ["Newton", math.e]
]
for item in data:
    # <  left alignment
    # >  right alignment
    # ^  center alignment
    print("{:<10} | {:>12.6f}".format(item[0],item[1]))
```

✓ 0.0s

```
Einstein   |     0.666667
Maxwell    |     3.141593
Newton     |     2.718282
```

Read/Write Files

Write Files

```python
f = open('file1.txt',mode='w',encoding="utf-8")
# mode (default='r'):
# 'w' : write (existing file will be overwritted)
# 'a' : appending to an existing file
# 'r' : only read a file
# 'r+': both read and write
# 'rb': read in bianry format
# 'wb': write in binary foramt
f.write("This the first line.\n")
f.write("This the second line. ")
f.write("Still in the second line.")
f.close()
```

✓ 0.0s

```
This the first line.
This the second line. Still in the second line.
```

Read Files

```python
f = open('file1.txt',mode='r')
msg = f.read()
print(msg)
f.close()
```

✓ 0.0s

```
This the first line.
This the second line. Still in the second line.
```

```python
f = open('file1.txt',mode='r')
for line in f:
    print(line)
```

✓ 0.0s

```
This the first line.

This the second line. Still in the second line.
```

Note: for scientific IO, there are many other packages to use

Exercise:

1. Modify your angry bird game

2. When calculating the trajectory, record the time and position of the trajectory in every time step dt = 0.1 sec into a file named “trajectory.txt”.

3. Make sure your values in the trajectory file are all aligned.

Miniexam 3 & Homework 1

1. Check google classroom
2. Write your solutions in .py files (do not use Jupyter notebook)
3. Please write comments and explanation of your answers
4. Your code should always have if __name__=='__main__':
5. Your code must able to executed by python your_code.py
6. Answer will be show up with the "print()" function.

計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 5: Basic Python (part 3)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Warm-up
2. Exercise: Angry bird game
3. Formatted output
4. Read/Write files
5. Exercise: Angry bird game

Today's plan

1. Exercise: Angry bird game (continue)
2. Modules / Packages
3. Classes (part 1)
4. Exercise: Angry bird game (version 2)

The Angry bird game (continue)

Angry bird

So far, we have these features in our angry bird game

1. Takes arbitrary inputs (separated by spaces)
2. Could quit the game by typing “quit”, “Quit”, or “QUIT”
3. Has the ability to check if hitting the target
4. Record the trajectory
5. Game continues for unlimited turns

Angry bird

(See demo)

Angry bird

- Additional features can be added (such as GUI interface, history, multiple targets, multiple game stages, scores, …etc.).
- However, if we write all these features in a single python file, the code will become difficult to read and to maintain.
- Could we separate different features into different python codes? Yes, that is modules.

Modules

Modules

1. A module is a file consisting of python codes
2. A module can define functions, classes and variables
3. A module allows you to logically organize your python codes
4. Grouping related code into a module makes the code easier to read and use
5. Modules can be used by the “import” statement
6. Recall the “import math”

Examples

File: particle.py

ComputationalPhysics > tutorial > particle.py > ...

```python
def set_particle_mass():
    print("set particle mass.")
    return

def apply_forces():
    print("force applied.")
    return

```

File: physics.py

ComputationalPhysics > tutorial > physics.py > ...

```python
def weak_interactions():
    print("weak forces.")
    return

def strong_interactions():
    print("strong forces.")
    return

def gravitational_forces():
    print("gravitational forces.")
    return

def electromagnetic_forces():
    print("EM forces.")
    return
```

Example

ComputationalPhysics > tutorial > tut_07_modules.py

```python
import physics
import particle

physics.
```

```
electromagnetic_forces
gravitational_forces
strong_interactions
weak_interactions
__annotations__
__builtins__
__cached__
__dict__
__doc__
__file__
__loader__
__name__
```

Example

ComputationalPhysics > tutorial > tut_07_modules.py > ...

```python
import physics
import particle

def engine():
    physics.weak_interactions()
    physics.gravitational_forces()
    particle.apply_forces()

if __name__=='__main__':

    engine()
```

```
----------
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_07_modules.py
weak forces.
gravitational forces.
force applied.
```

Import

1. import physics
2. import physics as phy
3. from physics import *
4. from physics import weak_interactions
5. from physics import weak_interactions as weak

Exercise: Angry bird game, again!

1. Now, let's separate the code of our angry bird game into several modules.

2. Modules include: game.py, physics, and output.py. Which function should put into which file?

Standard modules

1. There are many standard python library: https://docs.python.org/3/library/
2. The system module (import sys). For example, print(sys.argv[1]; sys.path.append(‘’); sys.path.insert(1, “”)).
3. Import math
4. Datetime: import datetime
5. Additional non-standard libraries can be installed via “conda install” or “pip install”
6. Try “conda install numpy”

Packages

Python packages

1. Packages are a way of structuring Python's module namespace by using "dotted module names".

2. For example, the module name A.B designates a submodule named B in a package named A.

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

https://docs.python.org/3/tutorial/modules.html

Python packages

1. Packages are a way of structuring Python's module namespace by using "dotted module names".
2. For example, the module name A.B designates a submodule named B in a package named A.
3. The __init__.py files are required to make Python treat directories containing the file as packages.

Python packages

Users of the package can import individual modules from the package, for example:

```
import sound.effects.echo
```

This loads the submodule `sound.effects.echo`. It must be referenced with its full name.

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

An alternative way of importing the submodule is:

```
from sound.effects import echo
```

This also loads the submodule `echo`, and makes it available without its package prefix, so it can be used as follows:

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Yet another variation is to import the desired function or variable directly:

```
from sound.effects.echo import echofilter
```

Again, this loads the submodule `echo`, but this makes its function `echofilter()` directly available:

```
echofilter(input, output, delay=0.7, atten=4)
```

Intra-package references

When packages are structured into subpackages (as with the sound package in the example), you can use absolute imports to refer to submodules of siblings packages. For example, if the module `sound.filters.vocoder` needs to use the echo module in the `sound.effects` package, it can use `from sound.effects import echo`.

You can also write relative imports, with the `from module import name` form of import statement. These imports use leading dots to indicate the current and parent packages involved in the relative import. From the `surround` module for example, you might use:

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Note that relative imports are based on the name of the current module. Since the name of the main module is always `"__main__"`, modules intended for use as the main module of a Python application must always use absolute imports.

Classes (part 1)

Python Class

1. Classes provide a means of bundling data and functionality together.
2. Classes are a fundamental concept in object-oriented programming (OOP)
3. Classes allow you to create your own data types with their own attributes and methods.
4. Classes are the foundation of many Python libraries and frameworks, such as matplotlib, spicy, …etc.

Creating a class

1. To create a class, use the “class” keyword followed by the class name.

2. Class names should start with a capital letter.

ComputationalPhysics > tutorial > tut_08_class.py > ...

```
class PhysicalConstant:
    pass

```

Defining Attributes

1. Attributes are variables defined within a class

2. They are also known as instance variables or properties

```python
class PhysicalConstant:
    """
    A class to include physical constants
    """
    def __init__(self, units="cgs"):
        """
        :param units: supprot "cgs" and "mks".
        """
        self.units = units
        self.c_cgs = 2.99792458e10
        self.c_mks = 2.99792458e8
        self.G_cgs = 6.67428e-8
        self.G_mks = 6.67428e-11
        if units=="cgs":
            self.c = self.c_cgs
            self.G = self.G_cgs
        elif units=="mks":
            self.c = self.c_mks
            self.G = self.G_mks
        else:
            print(r"Eror: no such units. units = {units}")
            print(r"Set the units to 'cgs'. ")
            self.c = self.c_cgs
            self.G = self.G_cgs

        return
```

Defining methods

1. Methods are functions defined inside a class that can be called on an object of the class. To define a method, you create a function inside the class. The first argument of the method should always be self, which refers to the object calling the method.

```
    def set_units(self, units):
        self.units = units
        if units=="cgs":
            self.c = self.c_cgs
            self.G = self.G_cgs
        elif units=="mks":
            self.c = self.c_mks
            self.G = self.G_mks
        else:
            print(r"Eror: no such units. units = {units}")
            print(r"Set the units to 'cgs'. ")
            self.c = self.c_cgs
            self.G = self.G_cgs
        return
```

Creating objects

1. To create an object of a class, you call the class name as if it were a function and pass in any arguments required by the __init__ method.

```python
if __name__=='__main__':

    const = PhysicalConstant(units="cgs")
    print(f"[{const.units}] c = {const.c:12.6e}")
    const.set_units("mks")
    print(f"[{const.units}] c = {const.c:12.6e}")

```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_08_class.py
[cgs] c = 2.997925e+10
[mks] c = 2.997925e+08
```

Example: The Angry bird game

1. Back to the angry bird game.

2. Let's use classes to re-write our angry bird game.

(See demo)

計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 6: Basic Python (part 4)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Warm-up
2. Exercise: Angry bird game
3. Formatted output
4. Read/Write files
5. Exercise: Angry bird game
6. Modules

Today’s plan

1. Python Classes
2. Property, getter, setter
3. Data class
4. Exercise: Angry bird game (again!)
5. Note: Next Monday is a national holiday (no lecture)

The angry bird game (sample code)

1. Check the non-interactive version of the “angry.py” in google classroom
2. The initial velocity and angle are taken from the system arguments in sys.argv

```
try:
    velocity = float(sys.argv[1]) #33.0 # m/s
    angle    = float(sys.argv[2]) #45.0 # degree
except:
    msg = """
    Error: incorrect inputs.
```

Python Classes

Python Class

1. Classes provide a means of bundling data and functionality together.

2. Classes are a fundamental concept in object-oriented programming (OOP)

3. Classes allow you to create your own data types with their own attributes and methods.

4. Classes are the foundation of many Python libraries and frameworks, such as matplotlib, spicy, …etc.

Creating a class

1. To create a class, use the “class” keyword followed by the class name.

2. Class names should start with a capital letter.

ComputationalPhysics > tutorial > tut_08_class.py > ...

```
class PhysicalConstant:
    pass

```

Defining Attributes

3. Attributes are variables defined within a class

4. They are also known as instance variables or properties

```python
class PhysicalConstant:
    """
    A class to include physical constants
    """
    def __init__(self, units="cgs"):
        """
        :param units: supprot "cgs" and "mks".
        """
        self.units = units
        self.c_cgs = 2.99792458e10
        self.c_mks = 2.99792458e8
        self.G_cgs = 6.67428e-8
        self.G_mks = 6.67428e-11
        if units=="cgs":
            self.c = self.c_cgs
            self.G = self.G_cgs
        elif units=="mks":
            self.c = self.c_mks
            self.G = self.G_mks
        else:
            print(r"Eror: no such units. units = {units}")
            print(r"Set the units to 'cgs'. ")
            self.c = self.c_cgs
            self.G = self.G_cgs

        return
```

Private variables

5. Private attributes: start with double underscore: __xxx

6. Private attributes cannot be accessed outside the class

```
class ExampleClass:

    def __init__(self):
        self.this_is_public    = "Public"
        self.__this_is_private = "Private"

    def get_private(self):
        print(self.__this_is_private)
```

```
ex = ExampleClass()
#print(ex.__this_is_private)
print(ex.this_is_public)
ex.get_private()
```

Defining methods

7. Methods are functions defined inside a class that can be called on an object of the class. To define a method, you create a function inside the class. The first argument of the method should always be self, which refers to the object calling the method.

```
    def set_units(self, units):
        self.units = units
        if units=="cgs":
            self.c = self.c_cgs
            self.G = self.G_cgs
        elif units=="mks":
            self.c = self.c_mks
            self.G = self.G_mks
        else:
            print(r"Eror: no such units. units = {units}")
            print(r"Set the units to 'cgs'. ")
            self.c = self.c_cgs
            self.G = self.G_cgs
        return
```

Creating objects

8. To create an object of a class, you call the class name as if it were a function and pass in any arguments required by the __init__ method.

```python
if __name__=='__main__':

    const = PhysicalConstant(units="cgs")
    print(f"[{const.units}] c = {const.c:12.6e}")
    const.set_units("mks")
    print(f"[{const.units}] c = {const.c:12.6e}")

```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_08_class.py
[cgs] c = 2.997925e+10
[mks] c = 2.997925e+08
```

property, getter, setter

Property, getter, setter

1. To protect some attributes of a class, we make them private attributes (Encapsulation).

2. Getters: methods to access the private attributes from a class

3. Setters: methods to set the values of private attributes in a class

```python
class NormalClass:

    def __init__(self,name):
        self.name = name
```

```python
class ExampleClass:

    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, new_name):
        self.__name = new_name
```

```python
__name__=='__main__':

    a = ExampleClass(name="Albert")
    #print(a.__name)
    print(a.get_name())
    a.set_name("Einstein")
    print(a.get_name())
```

Property, getter, setter

4. The python way to use getter and setter.

```python
class ExampleClass2:

    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, new_name):
        self.__name = new_name
```

```python
if __name__=='__main__':

    a = ExampleClass(name="Albert")
    #print(a.__name)
    print(a.get_name())
    a.set_name("Einstein")
    print(a.get_name())

    b = ExampleClass2(name="Albert")
    print(b.name)
    b.name = "Einstein"
    print(b.name)
```

Property, getter, setter

5. Property (setter & getter) can help you to avoid inconsistence.

```
class Name:
    def __init__(self,first_name,last_name):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        return self.first_name+" "+self.last_name
```

```
c = Name("Albert", "Einstein")
print(c.full_name)
```

Property, getter, setter

6. It is also possible that your setter can be more complicate. Property can be useful for verification.

```
class Celsius:
    def __init__(self, temperature = 0):
        self._temperature = temperature

    def to_fahrenheit(self):
        return (self._temperature * 1.8) + 32

    def to_kevin(self):
        return self._temperature + 273.15

    @property
    def temperature(self):
        return self._temperature

    @temperature.setter
    def temperature(self, value):
        if value < -273.15
            raise ValueError("Temperature cannot below -273.15 C")
        self._temperature = value
```

Data Class

Python data class

1. New feature after Python 3.7

2. With the “@dataclass” decorator.

3. A data class is a class that is designed to only hold data values.

4. They are typically used to store information that will be passed between different parts of a program or a system. (For example, in the angry bird exercise, we use a dictionary “data={}” to store the trajectory information)

5. REF: https://docs.python.org/3/library/dataclasses.html

Python data class

1. The regular approach:

```
class Point_regular:
    def __init__(self,t,x,y):
        self.x = x
        self.y = y
        return
```

2. Use the @dataclass

```
from dataclasses import dataclass

@dataclass
class Point:
    x:float
    y:float
```

Python data class

3. Dataclass supports default values as well:

```
@dataclass
class Person:
    name: str = "Albert Einstein"
    age: int  = 30
```

Exercise: Point and Trajectory class

1. In the angry bird game, we record the trajectory of an angry bird.

2. The trajectory can be used (1) to check if you hit the target, (2) to visualize the motion, (3) can be dumped to a data file.

3. Let's write two dataclass "Point" and "Trajectory" for the angry bird game in a module file named "data.py".

4. The "Point" dataclass contains these variables: t, x, y, vx, vy, ax, ay

5. The "Trajectory" dataclass contains these variables: times: list(float), posx: list(float), posy: list(float), velx: list(float), vely: list(float), accx: list(float), accy: list(float)

6. The "Trajectory" class has the method "append(self, point: Point)" to add new data in a trajectory.

Exercise 1: Point and Trajectory class

7. The “Trajectory” class has the method “dump(self)” to save the trajectory to a txt file.

8. Example usage:

```
if __name__=='__main__':

    p  = Point(t=0,x=0,y=0,vx=10,vy=10,ax=0,ay=0)
    p2 = Point(t=0,x=0,y=0,vx=10,vy=10)

    traj = Trajectory()
    traj.append(p)
    traj.append(p)
    traj.dump()
```

(See demo)

Exercise 2: More class examples

1. Imagine a scenario that there could be more than one target (pig) in the game

2. The targets could move as well.

3. Thus, both the bird and pigs are game characters (but one is the player; others are NPC)

4. Write a "Character" class three properties: name : str, position : Point, traj : Trajectory

5. The "Character" class has two methods: (1) update_do(self, time:float) : to update self.position to a new time, using the analytical solution. (2) distance_to(self, pt: Point): to valuate the distance to another point: Point.

Exercise 2: More class examples

6. Example usage:

```
if __name__=='__main__':

    pt   = Point(0,0,0,10,10)
    bird = Character("Angry Bird", pt)

    times = [0.1*t for t in range(20)]

    for t in times:
        bird.update_to(t)

    print(bird.traj.times)
```

(See demo)

Exercise 3: More class examples

1. Write another class called “Game” to handle the initial conditions and the game controls.

2. Import “Point”, “Trajectory”, and “Character”.

3. The game class has several methods to hand the initial conditions and game controls.

   You could design it by yourself (for example; add_bird(); add_pigs(); shoot(); check(), …)

4. Write a “play()” method to play the game.

Exercise 3: More class examples

5. Example usage:

```
if __name__=='__main__':

    game = Game()

    if len(sys.argv) != 3:
        msg = """
        Incorrect inputs.

        Try: python game.py 30 45

        """
        print(msg)
        quit()

    velocity = float(sys.argv[1]) #33.0 # m/s
    angle    = float(sys.argv[2] )#45.0 # degree

    angle_rad = math.radians(angle)
    vx  = velocity * math.cos(angle_rad)
    vy  = velocity * math.sin(angle_rad)
    ax  = 0.0
    ay  = -9.81

    pt   = Point(0,0,0,vx,vy,ax,ay)
    bird = Character(name="Angry Bird",pt=pt)

    pt1  = Point(0,100,0,0,0,0,0)
    pig1  = Character(name="Pig 1",pt=pt1)

    pt2  = Point(0,100,0,3,0,0,0)
    pig2  = Character(name="Pig 2",pt=pt2)
    game.play(bird,[pig1, pig2])
```

(See demo)

Class inheritance

Inheritance

1. a language feature would not be worthy of the name “class” without supporting inheritance.

2. Example:

```
class AngryBird(Character):

    def __init__(self, name, mass=1):
        self.mass = 1.0
        pt = Point()
        super().__init__(name, pt)
        return

    def set_velocity_and_angle(self, velocity:float, angle:float):
        angle_rad = math.radians(angle)
        vx = velocity * math.cos(angle_rad)
        vy = velocity * math.sin(angle_rad)
        self.point.vx = vx
        self.point.vy = vy
        return

    def apply_force(self, fx, fy):
        m = self.mass
        self.point.ax = fx/m
        self.point.ay = fy/m
        return
```

```
class Pig(Character):

    def __init__(self, name, distance):
        self.distance = distance
        pt = Point()
        pt.x = distance
        super().__init__(name,pt)
        return

    def set_velocity(self, speed):
        self.point.vx = speed
        return
```

Inheritance

3. The game can be simplified to

```
bird = AngryBird(name="Angry Bird")
bird.set_velocity_and_angle(velocity,angle)
bird.apply_force(fx=0,fy=-9.81)

pig1 = Pig(name="Pig 1", distance=100)
pig1.set_velocity(speed= 1.0)
pig2 = Pig(name="Pig 2", distance=100)
pig2.set_velocity(speed= -1.0)

game.play(bird,[pig1, pig2])
```

計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 7: Advanced Python (part 1)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Python Classes
2. Property, getter, setter
3. Data class
4. Exercise: Angry bird game (again!)
5. Homework 2 is released

Today's plan

1. Advanced topics
2. Errors and Exceptions
3. Command line arguments
4. Numerical integral
5. Lambda
6. Performance measurement

Errors and Exceptions

Errors and Exceptions

1. In the angry bird, we already learned that we could use “try” and “except” to handle some error messages, but actually these error messages have two distinguishable kinds of errors: errors and exceptions.
2. Syntax Errors / IndentationError (parsing errors): missing “:” missing “tabs”

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python
Python 3.10.9 | packaged by conda-forge | (main, Feb  2 2023, 20:24:27) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> def print_hello()
  File "<stdin>", line 1
    def print_hello()
                     ^
SyntaxError: expected ':'
>>>
```

Built-in Exceptions

1. See: https://docs.python.org/3/library/exceptions.html#bltin-exceptions

2. Common exceptions

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

Ref: https://www.devinline.com/2015/04/exception-handling-in-python.html

Handling Exceptions

1. There are possibility that you only want to handle exceptions of certain errors (not all errors)

```python
def input_velocity():
    try:
        x = float(input("Enter a real number.\n"))

    except ValueError:

        print("Error: please enter a real number. Try again.")

    return
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_11_errors.py
Enter a real number.
pan
Error: please enter a real number. Try again.
```

Handling Exceptions

2. You can handle multiple errors at once

```
try:
    f = open("trajectory.txt",mode="r")
    s = f.readline()
    print(s.strip())
    i = int(s.strip())
except OSError as err:
    print("OS Error:", err)
except ValueError:
    print("Could not convert data to an integer")
except Exception as err:
    print(f"Unexpected error {err=},{type(err)=}")
    raise
else:
    print(s)
    f.close()
```

Raising errors

1. Use “raise” to raise an error

```
try:
    if float(velocity) >= 2.997924581e8:
        raise RuntimeError(f"{velocity=} can not be faster than c.")
    print("The velocity looks good")
except ValueError as exec:
    raise RuntimeError("Incorrect value of velocity.") from exec
```

User defined exceptions

1. To make the code easier to read, we could define our custom error exceptions.

```
class MyError(Exception):
    pass

class InvalidVelocityError(Exception):
    def __init__(self):
        msg = "Velocity cannot be faster than the speed of the light."
        super().__init__(msg)
```

Extra useful tips

1. Use “finally” to clean-up actions.
2. Add notes in the exception (after python 3.11)

```
try:
    raise TypeError('bad type')
except Exception as e:
    e.add_note("We can add extra notes.") # after python 3.11
    e.add_note("more info about this error.")
    raise
```

Exercise: Error and exception

1. Write a function to set the temperature of a fiber in a lab. The Lab requires a low temperature environment with T < 30 K.

```
def set_temperature(newT):
    # do stuff here
    return newT
```

2. Raise two types of errors when (1) UnphysicalTemperatureError: when the desired temperature is less or equal to 0 K (2) HighTemperatureError: when temperature is higher than 30K

3. Make an exception with the HighTemperatureError, but print a warning to the user and store the new temperature in a log file (“temperature.log”).

Command Line Arguments

Command line arguments

1. There are many situations that you want reuse the same codes but with different parameters.
2. You could either modify the constant every time your run the code, or use the sys.argv to set the parameter (like the angry bird game we used)
3. A more professional way is to use the “argparse” package.

Argparse

```python
from argparse import ArgumentParser

def set_parser():

    parser = ArgumentParser()
    parser.add_argument('fname')
    parser.add_argument('-v','--velocity',type=float,default=100,help="the velocity [m/s]")
    parser.add_argument('-u','--units',type=str, default="mks", help="The units systems. 'cgs' or 'mks'")
    parser.add_argument('-d','--debug',dest="is_debugging",action="store_true",default=False,help="use debug mode")
    return parser

def get_parameters(parser):

    args = parser.parse_args()
    print(f"{args.fname=}")
    print(f"{args.velocity}")
    print(f"{args.units}")
    print(f"{args.is_debugging}")
    return
```

```python
if __name__=='__main__':

    parser = set_parser()
    get_parameters(parser)
```

Argparse

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_12_argparse.py
usage: tut_12_argparse.py [-h] [-v VELOCITY] [-u UNITS] [-d] fname
tut_12_argparse.py: error: the following arguments are required: fname
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_12_argparse.py --help
usage: tut_12_argparse.py [-h] [-v VELOCITY] [-u UNITS] [-d] fname

positional arguments:
  fname

options:
  -h, --help            show this help message and exit
  -v VELOCITY, --velocity VELOCITY
                        the velocity [m/s]
  -u UNITS, --units UNITS
                        The units systems. 'cgs' or 'mks'
  -d, --debug           use debug mode
```

Argparse

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_12_argparse.py trajectory.txt
args.fname='trajectory.txt'
100
mks
False
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_12_argparse.py trajectory.txt -v 30.5 --debug
args.fname='trajectory.txt'
30.5
mks
True
```

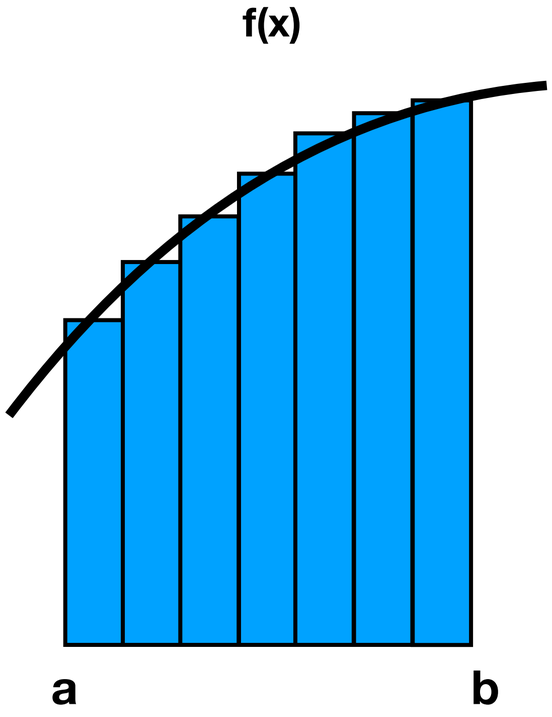
Exercise: The Angry bird game (again)

1. Modify the “angry.py” file in the google classroom (you could disable the plot_trajectory() function if you have problem to use matplotlib).
2. Modify the play() function. Replace the sys.argv parts to argparse.
3. Implement options with
   - - - velocity (-v) to set the initial velocity (default = 20 m/sec)
   - - - angle (-a) to set the initial inclination angle (default= 40 degree)
   - - - plot to plot the trajectory (default: False)
   - - - log to write into a log file (default: True)

Example: Numerical Integral

Write a numerical integrator

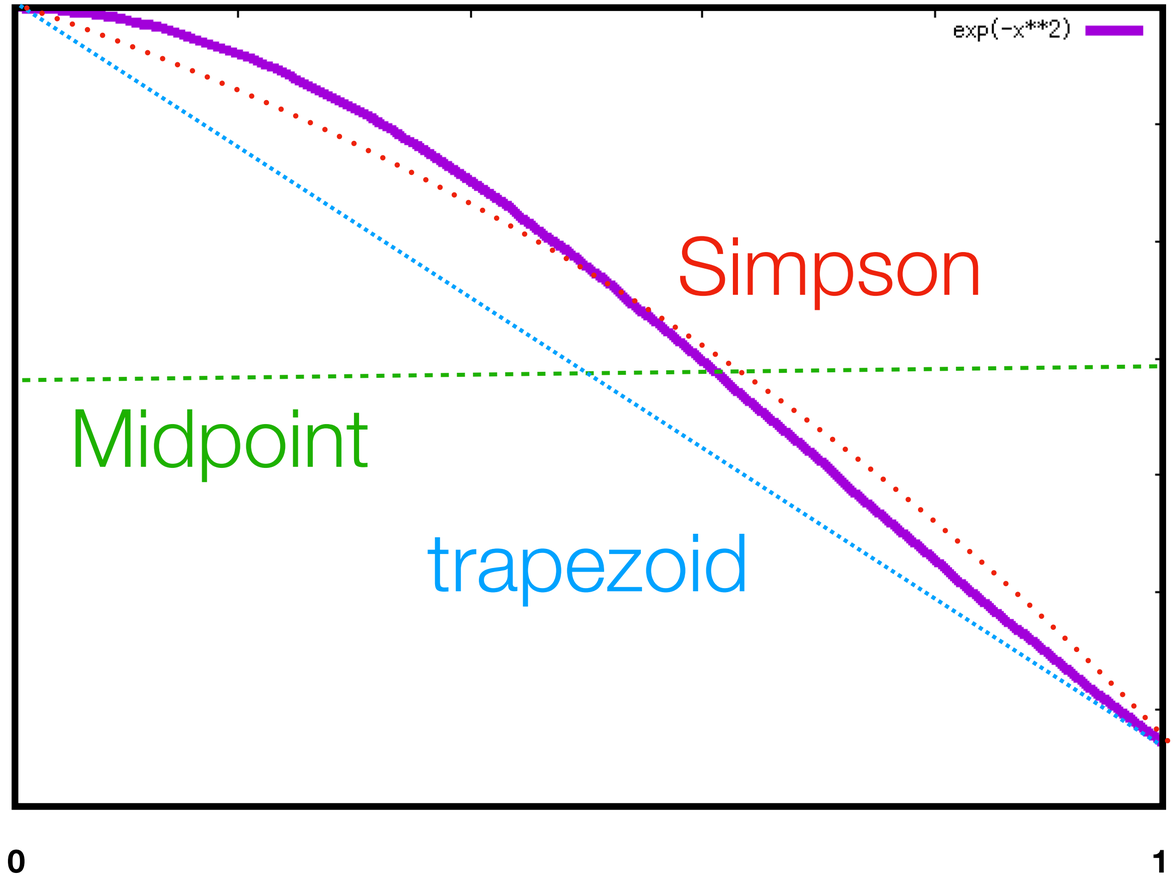
1. To integrate a function from a to b is to evaluate the area under the function in [a,b]



2. Let's write a python class called "Integrator" that can do finite integral of an arbitrary function from a to b with dx = [b-a]/N, where N is the user-defined, number of divisions.

3. To evaluate the area of each sub-devision, we could use "midpot", "trapezoid", or "simpson" methods

Write a numerical integrator



4.

Midpoint rule

$$\int_a^b f(x)dx \sim (b-a)f(\frac{a+b}{2})$$

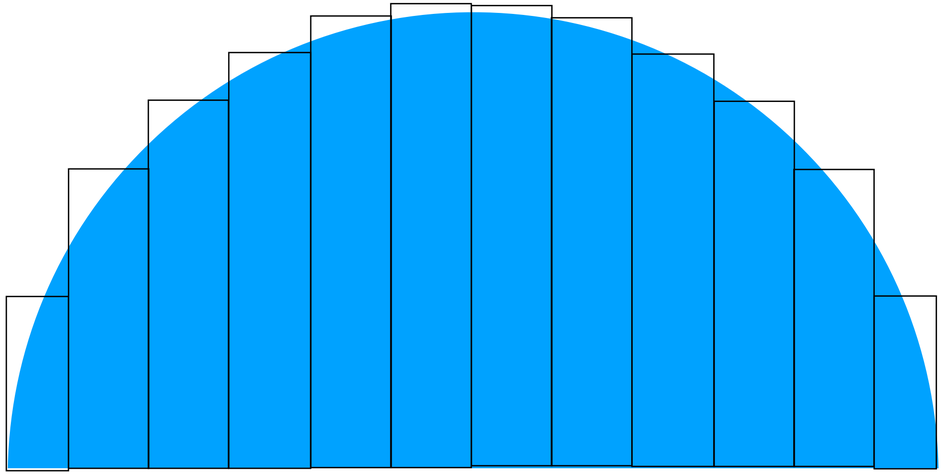
Trapezoidal rule

$$\int_a^b f(x)dx \sim (b-a)(\frac{f(a)+f(b)}{2})$$

Simpson's rule

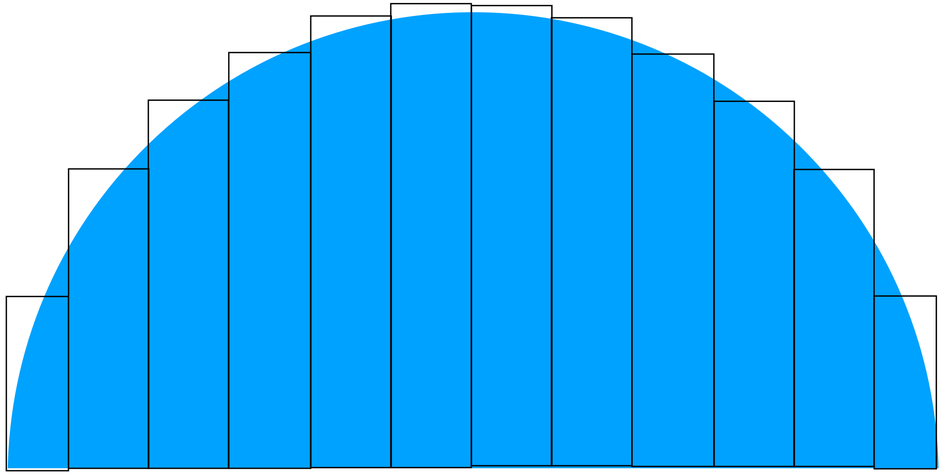
$$\int_a^b f(x)dx \sim \frac{(b-a)}{6}(f(a)+4f(\frac{a+b}{2})+f(b))$$

Example: Calculate Pi



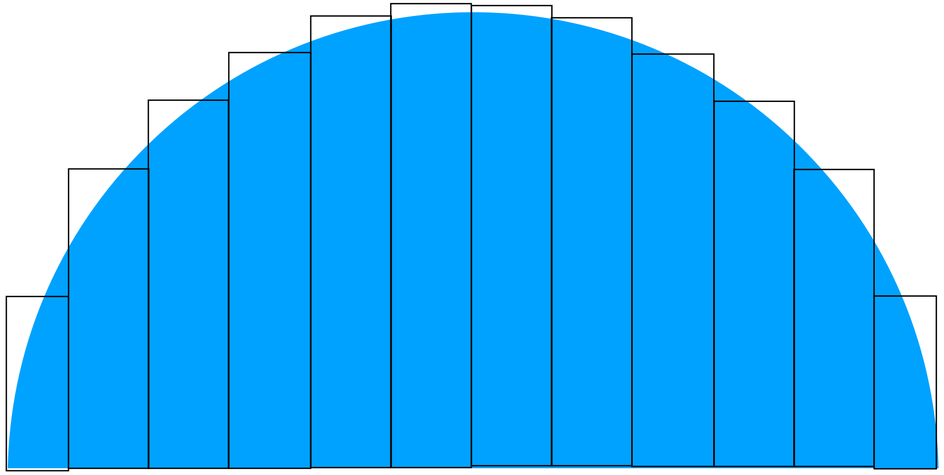
1. The area of a half-unit-circle is equal to pi/2
2. Use the integrator we wrote to evaluate pi
3. Start from the “midpoint” method.
4. See the sample code “integrator_template.py”
5. Note: the program could be slow in this example

Example: Calculate Pi



```python
import math

class Integrator:

    def __init__(self, func):
        """
        setup the integrator with a given function
        """

        # TODO

        return

    def midpoint(self,a=-1,b=1,N=10_000):
        """
        Assume a < b
        """

        # TODO

        return area


if __name__=='__main__':

    def hcirc(x):
        return math.sqrt(1-x**2)

    integrate = Integrator(func=hcirc)
    area = integrate.midpoint(a=-1,b=1,N=10000)
    print(2*area)
```

Exercise: other methods



1. Now, it is your time to practice implementing the trapezoidal rule and Simpson's rule.
2. Write two class methods "trap" and "simpson" to do the same integrals.
3. Try different N, see your results with different methods are less or larger than the true PI.
4. Remember to raise an error if b <= a

Python Lambda

Write a numerical integrator

1. Python's "lambda" is a very powerful way to define functions.
2. A lambda function is a small anonymous function (no function name)
3. A lambda function can take any number of arguments, but can only have one expression
4. Syntax: `lambda arguments : expression`
5. Example

```
hcirc = lambda x: math.sqrt(1-x**2)
```

Write a numerical integrator

6. More examples:

```
# declare a lambda function
greet = lambda : print("Hello world!")

# call the lambda function
greet()
```

7. Create a list in one line with lambda

```
list1 = [(lambda x: x**2)(x) for x in range(10)]
#or
list2 = [x**2 for x in range(10)]
print(list1)
print(list2)
```

Write a numerical integrator

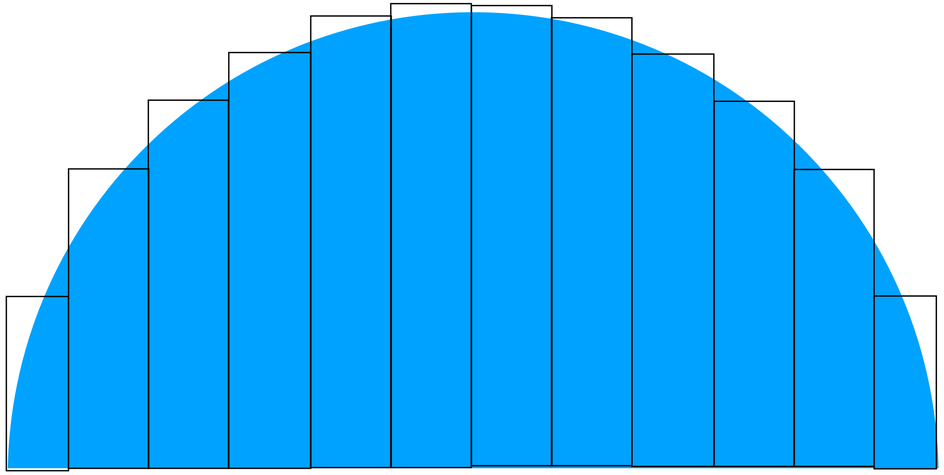
8. Use lambda to map a list

```
a  = [1, 2, 3, 4, 5, 6]
b = list(map(lambda x: x * 2 , a))

#or
c = [x*2 for x in a]
print(b)
print(c)
```

9. Use lambda to filter a list

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
b = list(filter(lambda x: (x%2 == 0) , a))

# or
c = [x for x in a if x%2 == 0]
print(b)
print(c)
```

Exercise 2: use lambda



1. Back to your Integrator(), now your lambda to define your function
2. Write a new method called midpoint2. Use lambda or list comprehensions to avoid the for loop when summing the area
3. Measure the performance of the two methods (midpoint and midpoint2)
4. (Next week we will show that both methods are very slow)

Performance Measurement

Timer and performance measurement

1. Performance measurement is very import in scientific computing.

2. Python could be VERY SLOW if we code it improperly.

3. It is very easy to make > 100 times slower than other programming language with python

4. When every you finish a program, you should think where is the performance bottleneck and whether we could improve it or not.

Timer and performance measurement

1. A simple way to measure the performance

```python
import time

def func1():
    # do some calculations
    a = 1
    for n in range(1000):
        a += 1
    print(a)
    return

if __name__=='__main__':

    t1 = time.time()
    func1()
    t2 = time.time()
    print(f"Time spend = {t2-t1}")
```

Timer and performance measurement

2. Use the “timeit” module

```
import timeit
timer = timeit.Timer(func1)
t = timer.timeit(number=10_000)
print(f"Average time = {t/10_000} sec.")
```

Timer and performance measurement

2. Use the “timeit” module to measure the performance of a function with arguments

```
def func2(N=1_000):
    # do some calculations
    a = 1
    for n in range(N):
        a += 1
    return
```

```
timer = timeit.Timer(lambda: func2(N=1000))
t = timer.timeit(number=1_000)
print(f"Average time = {t/1_000} sec.")
```

Exercise: Create a list

1. Measure the performance of creating a list with N=1_000_000 ones (i.e. [1,1,1….,1]).

2. Use

```
def make_list1(N=1_000_000):
    a = []
    for n in range(N):
        a.append(1)
```

3. Use

```
def make_list2(N=1_000_000):
    a = [1 for n in range(N)]
    return a
```

4. Remember that the CPU clock speed is about ~GHz (or ~ 1ns per operation). We should expect that it takes only 1e6 (elements) x 1 ns ~ 1.e-6 sec to create the list. What are the values you observed? Is python fast?

5. (Demo): use numpy

Exercise: Looping a list

1. Measure the performance of summing all the N=1_000_000 elements.

2. Use

```
def loop1(a):
    total = 0
    for v in a:
        total += v
    return total
```

3. Use the builtin “sum” function (i.e. total = sum(a))

4. (Demo): use numpy

5. Conclusions: Python’s default “loop” is VERY SLOW. In physical problem, we often encounter situation with big “array” or “matrix” (lists). We should avoid using loops. Scientific packages like “numpy” and “scipy” are therefore recommended.

計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 8: Advanced Python (part 2)

Instructor: 潘國全
kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Advanced topics
2. Errors and Exceptions
3. Command line arguments
4. Numerical integral

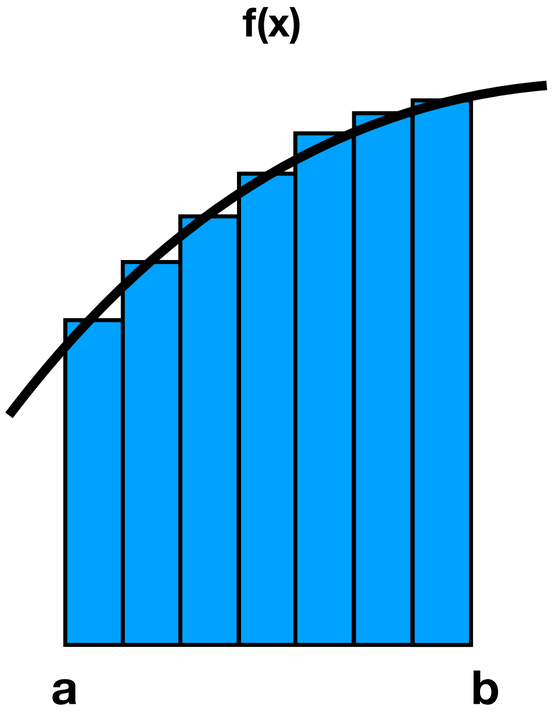
Today's plan

1. Warm-up (numerical integral)
2. Jupyter notebook
3. Lambda
4. Performance measurement
5. File wildcards (pathlib and glob)
6. NumPy (arrays & random numbers)

Warm-up: Numerical Integral

Write a numerical integrator

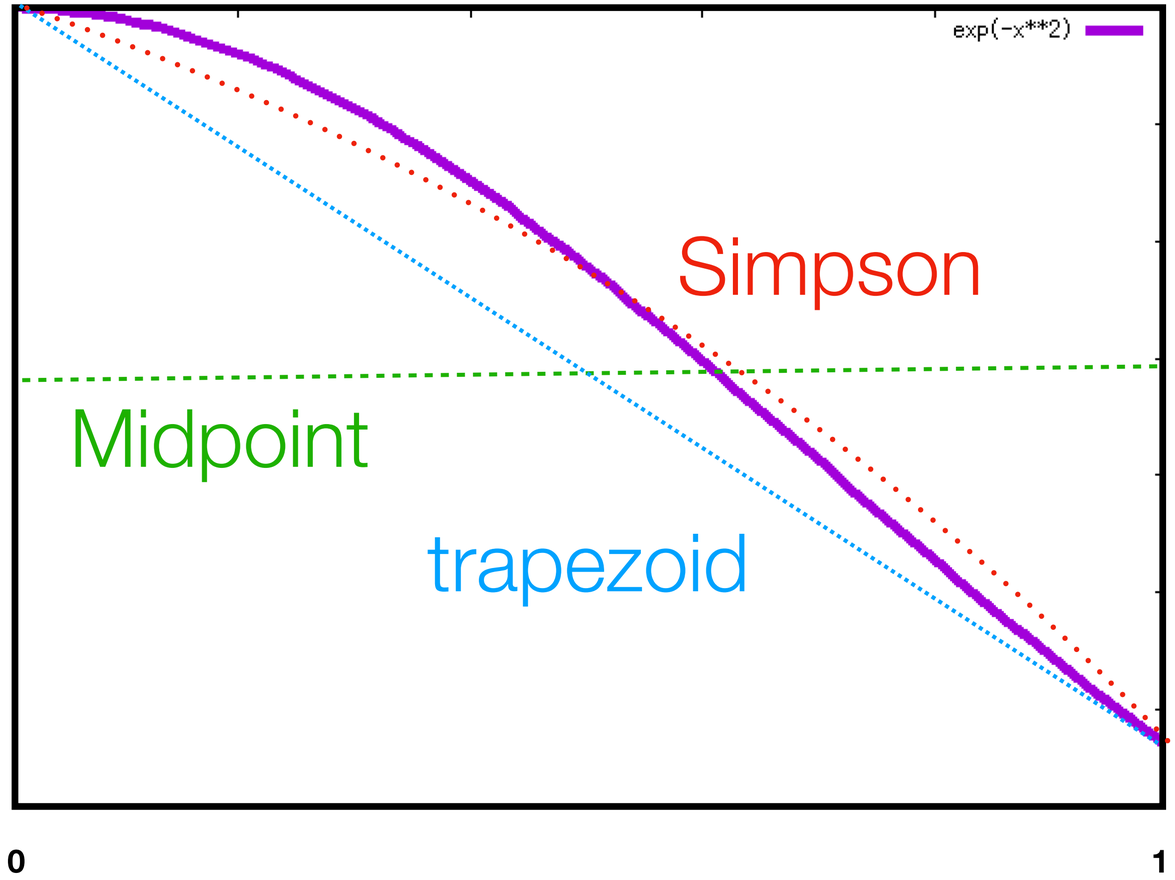
1. To integrate a function from a to b is to evaluate the area under the function in [a,b]



2. Let's write a python class called "Integrator" that can do finite integral of an arbitrary function from a to b with dx = [b-a]/N, where N is the user-defined, number of divisions.

3. To evaluate the area of each sub-devision, we could use "midpot", "trapezoid", or "simpson" methods

Write a numerical integrator



4.

Midpoint rule

$$\int_a^b f(x)dx \sim (b-a)f(\frac{a+b}{2})$$

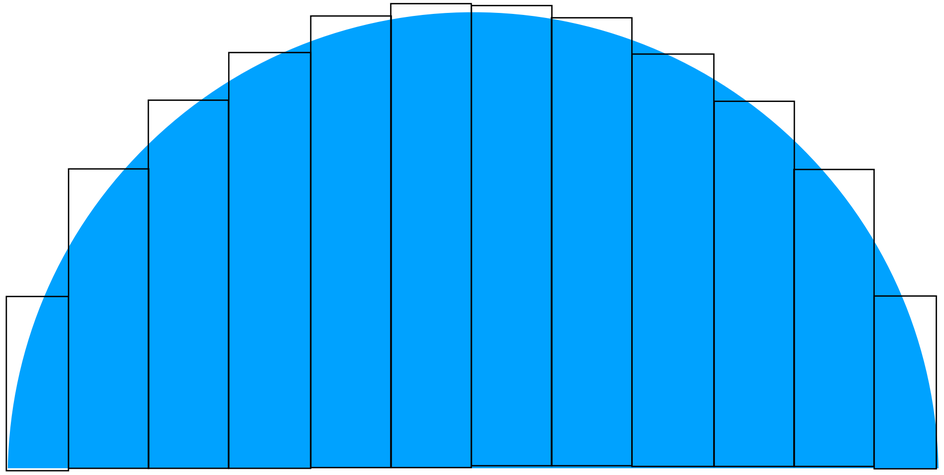
Trapezoidal rule

$$\int_a^b f(x)dx \sim (b-a)(\frac{f(a)+f(b)}{2})$$

Simpson's rule

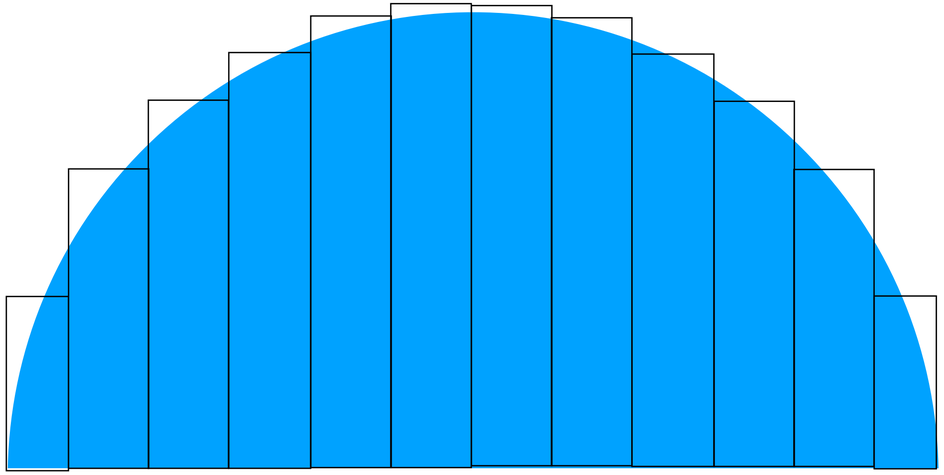
$$\int_a^b f(x)dx \sim \frac{(b-a)}{6}(f(a)+4f(\frac{a+b}{2})+f(b))$$

Example: Calculate Pi



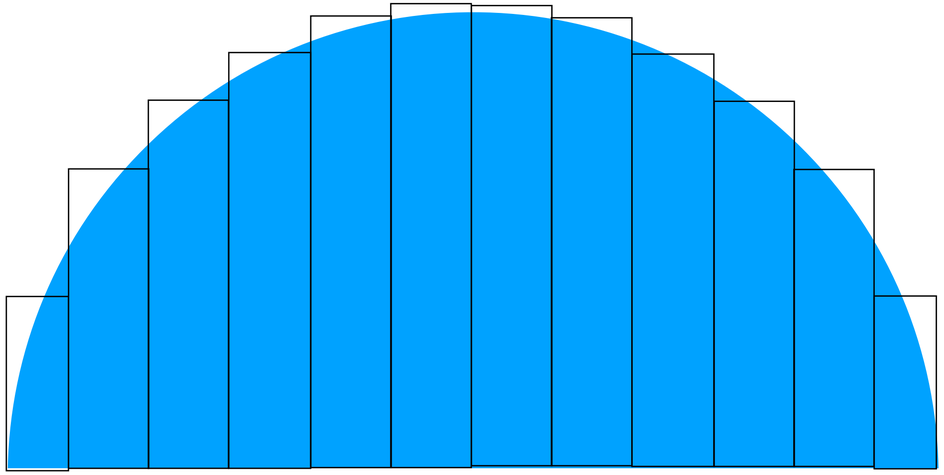
1. The area of a half-unit-circle is equal to pi/2
2. Use the integrator we wrote to evaluate pi
3. Start from the “midpoint” method.
4. See the sample code “integrator_template.py”
5. Note: the program could be slow in this example

Example: Calculate Pi



```python
import math

class Integrator:

    def __init__(self, func):
        """
        setup the integrator with a given function
        """

        # TODO

        return

    def midpoint(self,a=-1,b=1,N=10_000):
        """
        Assume a < b
        """

        # TODO

        return area


if __name__=='__main__':

    def hcirc(x):
        return math.sqrt(1-x**2)

    integrate = Integrator(func=hcirc)
    area = integrate.midpoint(a=-1,b=1,N=10000)
    print(2*area)
```

Exercise: other methods



1. Now, it is your time to practice implementing the trapezoidal rule and Simpson's rule.
2. Write two class methods "trap" and "simpson" to do the same integrals.
3. Try different N, see your results with different methods are less or larger than the true PI.
4. Remember to raise an error if b <= a

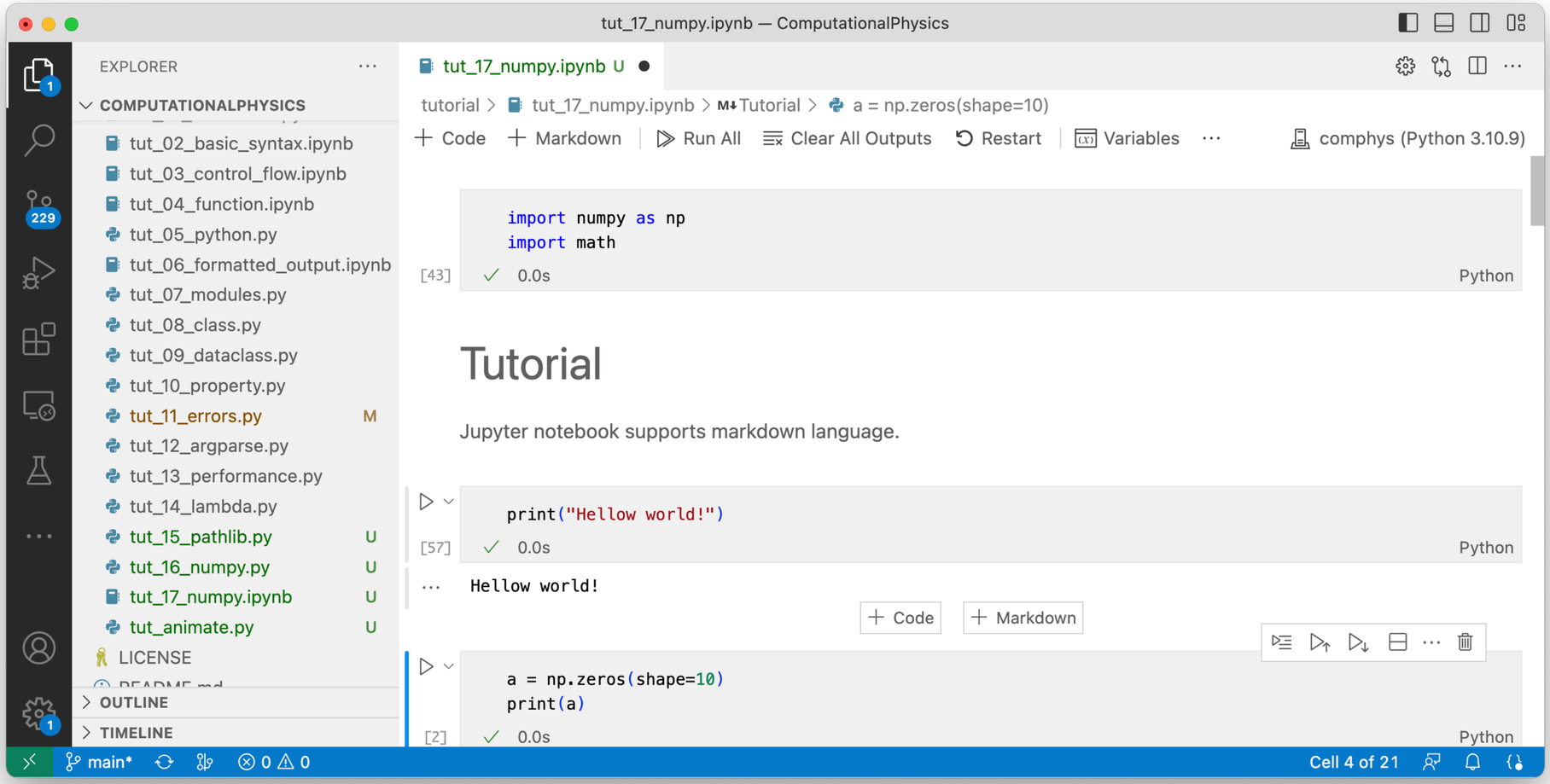
Jupyter notebook

Jupyter notebook

1. File extension is “.ipynb”
2. An interactive computing environment to quickly access programming results
3. Iterative development and faster feedback
4. Rich media integration: markdown, image, video, …etc.
5. Reproducible workflows
6. Disadvantages: (1) version control issues (2) slower than typical python scripts, (3) not ideal for production code (cannot import)

Jupyter notebook

1. (Demo)



Exercise: Jupyter notebook

1. Rewrite and port your numerical integration code in a Jupyter notebook.

2. Alternatively, you could import your Integrator class in a Jupyter notebook

Python Lambda

Write a numerical integrator

1. Python's "lambda" is a very powerful way to define functions.
2. A lambda function is a small anonymous function (no function name)
3. A lambda function can take any number of arguments, but can only have one expression
4. Syntax: `lambda arguments : expression`
5. Example

```
hcirc = lambda x: math.sqrt(1-x**2)
```

Write a numerical integrator

6. More examples:

```
# declare a lambda function
greet = lambda : print("Hello world!")

# call the lambda function
greet()
```

7. Create a list in one line with lambda

```
list1 = [(lambda x: x**2)(x) for x in range(10)]
#or
list2 = [x**2 for x in range(10)]
print(list1)
print(list2)
```

Write a numerical integrator

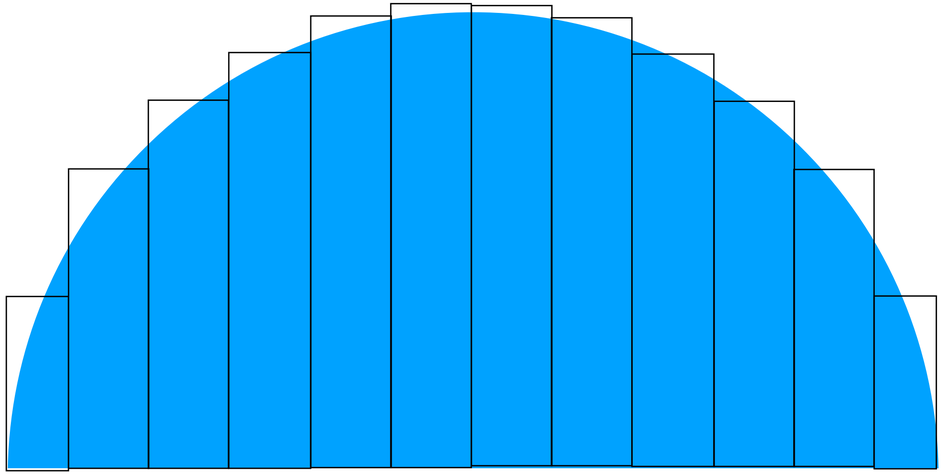
8. Use lambda to map a list

```
a  = [1, 2, 3, 4, 5, 6]
b = list(map(lambda x: x * 2 , a))

#or
c = [x*2 for x in a]
print(b)
print(c)
```

9. Use lambda to filter a list

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
b = list(filter(lambda x: (x%2 == 0) , a))

# or
c = [x for x in a if x%2 == 0]
print(b)
print(c)
```

Exercise: use lambda



1. Back to your Integrator(), now your lambda to define your function

2. Write a new method called midpoint2. Use lambda or list comprehensions to avoid the for loop when summing the area.

3. Once you have a list to store the area of each rectangles, you could use the function “sum()” to sum the total area and avoid the for loop.

4. Note that this method is memory inefficiency.

Performance Measurement

Timer and performance measurement

1. Performance measurement is very import in scientific computing.

2. Python could be VERY SLOW if we code it improperly.

3. It is very easy to make > 100 times slower than other programming language with python

4. When every you finish a program, you should think where is the performance bottleneck and whether we could improve it or not.

Timer and performance measurement

1. A simple way to measure the performance

```python
import time

def func1():
    # do some calculations
    a = 1
    for n in range(1000):
        a += 1
    print(a)
    return

if __name__=='__main__':

    t1 = time.time()
    func1()
    t2 = time.time()
    print(f"Time spend = {t2-t1}")
```

Timer and performance measurement

2. Use the “timeit” module

```
import timeit
timer = timeit.Timer(func1)
t = timer.timeit(number=10_000)
print(f"Average time = {t/10_000} sec.")
```

Timer and performance measurement

2. Use the “timeit” module to measure the performance of a function with arguments

```python
def func2(N=1_000):
    # do some calculations
    a = 1
    for n in range(N):
        a += 1
    return
```

```python
timer = timeit.Timer(lambda: func2(N=1000))
t = timer.timeit(number=1_000)
print(f"Average time = {t/1_000} sec.")
```

Performance measurement in Jupyter notebook

1. In Jupyter notebooks, use “%time” or “%timeit” to measure the performance.

```
%time a = func1(N=1_000_000)
```

✓ 0.0s

```
CPU times: user 42.5 ms, sys: 1.8 ms, total: 44.3 ms
Wall time: 42.8 ms
```

```
%timeit a = func1(N=1_000_000)
```

✓ 3.3s

```
40.6 ms ± 1.57 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Exercise: Create a list

1. Measure the performance of creating a list with N=1_000_000 ones (i.e. [1,1,1….,1]).

2. Use

```
def make_list1(N=1_000_000):
    a = []
    for n in range(N):
        a.append(1)
```

3. Use

```
def make_list2(N=1_000_000):
    a = [1 for n in range(N)]
    return a
```

4. Remember that the CPU clock speed is about ~GHz (or ~ 1ns per operation). We should expect that it takes only 1e6 (elements) x 1 ns ~ 1.e-6 sec to create the list. What are the values you observed? Is python fast?

5. (Demo): use numpy

Exercise: Looping a list

1. Measure the performance of summing all the N=1_000_000 elements.

2. Use

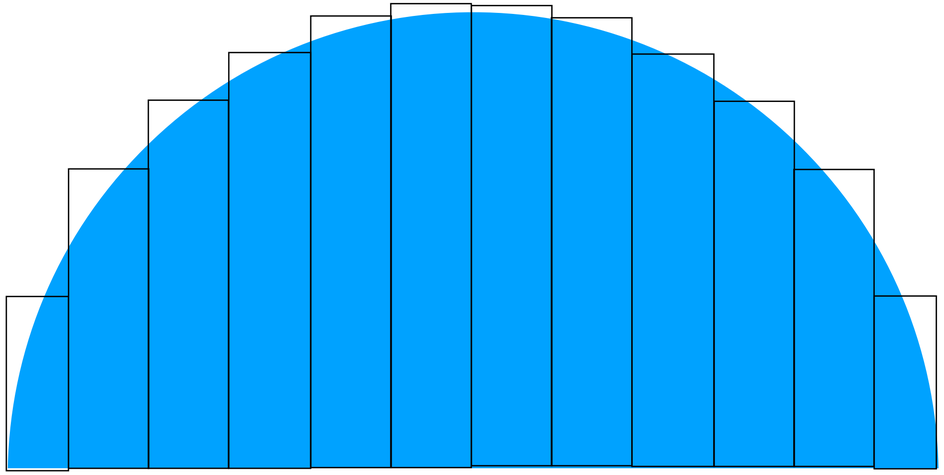
```
def loop1(a):
    total = 0
    for v in a:
        total += v
    return total
```

3. Use the builtin “sum” function (i.e. total = sum(a))

4. (Demo): use numpy

5. Conclusions: Python’s default “loop” is VERY SLOW. In physical problem, we often encounter situation with big “array” or “matrix” (lists). We should avoid using loops. Scientific packages like “numpy” and “scipy” are therefore recommended.

Exercise: use lambda

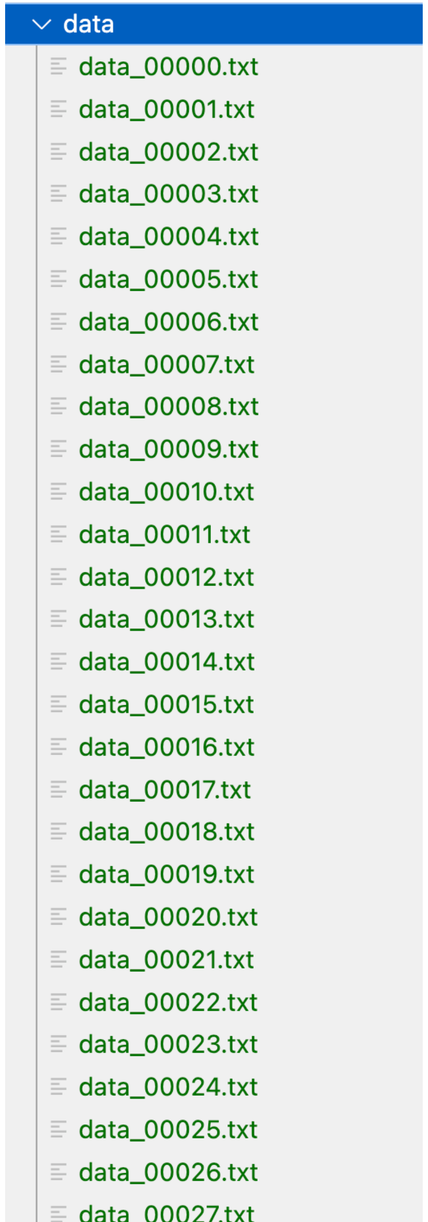


1. Back to your Integrator(), now your lambda to define your function
2. Write a new method called midpoint2. Use lambda or list comprehensions to avoid the for loop when summing the area
3. Measure the performance of the two methods (midpoint and midpoint2)
4. (we will show later that both methods are very slow)

File Wildcards: pathlib and glob

File wildcards: pathlib & glob

1. When doing data analysis, it is quite often that you need to handle multiple data files with sequent filenames (e.g. data_0001.txt, data_0002.txt, …etc).
2. str(intger).zfill(digits): convert a integer to a string and empty digits to zeros.
3. You might want to access all or part of the data files.
4. Search files that match a pattern string can be done by either "pathlib" or "glob"
5. pathlib: https://docs.python.org/3/library/pathlib.html
6. glob: https://docs.python.org/3/library/glob.html#module-glob



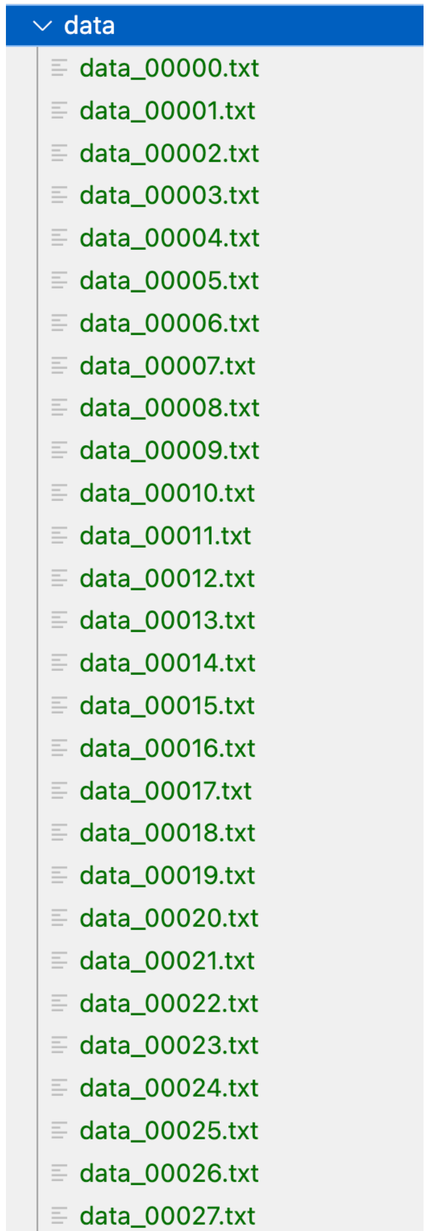
Example: pathlib & glob

1. Check if a folder exist. If not, create it.

```
from pathlib import Path

# check if the folder exists, if not create one
Path(folder).mkdir(parents=True, exist_ok=True)
```

2. Create sequent file names:

```
# loop all file names
for n in range(N):

    # the filename includes its relative path
    fname = folder+'/'+header+'_'+str(n).zfill(digits)+'.txt'
```



Example: pathlib & glob

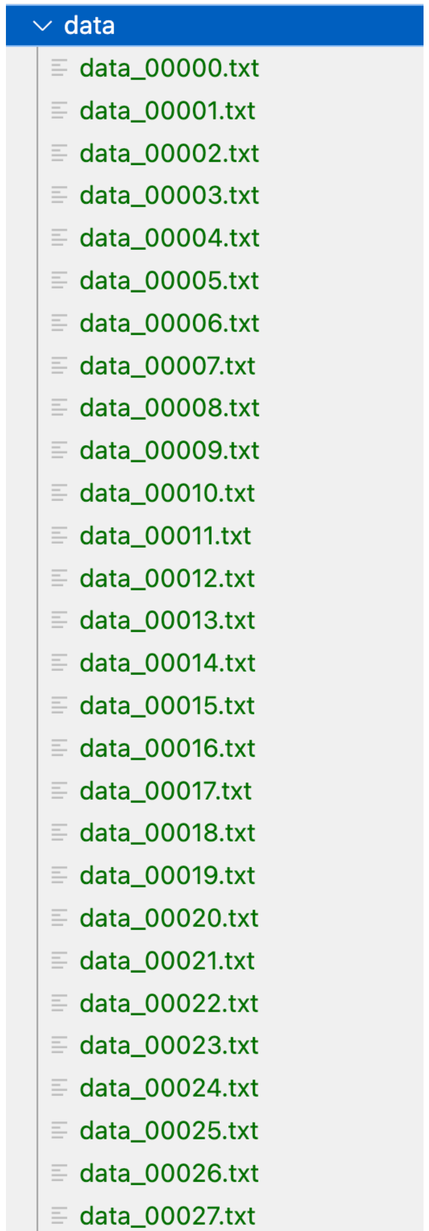
3. Get all file names that match a pattern

```
# get files that match the pattern
pattern = 'data_*[0,3,5][0-4].txt'
fns = sorted(Path(folder).glob(pattern))
print(fns)
```

4. Similar function can be achieved with glob.glob(pattern) as well.

```
pattern = folder+'/data_*[0,3,5][0-4].txt'
fns = sorted(glob.glob(pattern))
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_15_zfill.py                    pan@vega
[PosixPath('data/data_00000.txt'), PosixPath('data/data_00001.txt'), PosixPath('data/data_00002.txt'), PosixPath('data/data_00003.txt'), PosixPath('d
ata/data_00004.txt'), PosixPath('data/data_00030.txt'), PosixPath('data/data_00031.txt'), PosixPath('data/data_00032.txt'), PosixPath('data/data_0003
3.txt'), PosixPath('data/data_00034.txt'), PosixPath('data/data_00050.txt'), PosixPath('data/data_00051.txt'), PosixPath('data/data_00052.txt'), Posi
xPath('data/data_00053.txt'), PosixPath('data/data_00054.txt'), PosixPath('data/data_00100.txt'), PosixPath('data/data_00101.txt'), PosixPath('data/d
ata_00102.txt'), PosixPath('data/data_00103.txt'), PosixPath('data/data_00104.txt'), PosixPath('data/data_00130.txt'), PosixPath('data/data_00131.txt
'), PosixPath('data/data_00132.txt'), PosixPath('data/data_00133.txt'), PosixPath('data/data_00134.txt'), PosixPath('data/data_00150.txt'), PosixPath
('data/data_00151.txt'), PosixPath('data/data_00152.txt'), PosixPath('data/data_00153.txt'), PosixPath('data/data_00154.txt')]
```



Exercise: pathlib & glob

1. Run the sample code (tut_15_pathlib.py)

2. Experience the code with different N, header, and folder.

3. Play around with different “pattern”.

NumPy

NumPy

1. NumPy (Numerical Python) is an open source project that enables numerical computing with Python.
2. It was created in 2005 by Travis Oliphant.
3. Designed for good performance, core functionality is built around **arrays**, matrices, and linear algebra, …etc.
4. Installing NumPy: `conda install numpy`
5. Import NumPy: `import numpy as np`
6. Check NumPy version: `print(np.__version__)`

When to use numpy?

1. Dealing with arrays (mulit-D arrays)

2. Dealing with mathematical operations on arrays

3. Numerical calculations

NumPy array

1. Create a 1-d numpy array:

```
# create a numpy array
arr = np.arange(N)
```

2. Convert a python list to a numpy array

```
arr = np.array(list)
```

NumPy N-d array

3. Create a N-d numpy array:

```
def ndimensional_array():

    a0 = np.array(32)
    print(f"This is a {a0.ndim}-d numpy array: {a0}")
    a1 = np.array([0,1,2,3])
    print(f"This is a {a1.ndim}-d numpy array: {a1}")
    a2 = np.array([[1,2,3],[4,5,6]])
    print(f"This is a {a2.ndim}-d numpy array: {a2}")
    a3 = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
    print(f"This is a {a3.ndim}-d numpy array: {a3}")

    return
```

NumPy N-d array

4. Array shape:

```
print(a0.shape)
print(a1.shape)
print(a2.shape)
print(a3.shape)
```

```
()
(4,)
(2, 3)
(2, 2, 3)
```

NumPy N-d array

5. Array indexing:

```
print(a1[0])
print(a2[1,1])
print(a3[0,0,2])
```

6. Array slicing:

```
print(a1[0:2])
print(a2[1:2,1:2])
print(a3[0:1,0:1,0:2])
```

NumPy N-d array

7. Reshaping arrays:

```
arr1d = np.arange(12)
arr2d = arr1d.reshape(4,3)
arr3d = arr1d.reshape(2,2,3)

print(arr1d)
print(arr2d)
print(arr3d)
```

8. Flattening arrays:

```
arr1d  = arr3d.reshape(-1) # flatten
arr1d2 = arr3d.flatten()   # same as above
```

NumPy N-d array

9. Iterating N-d array

```
arr3d = np.arange(4*5*4).reshape(4,5,4)

# (method 1) iterate on the elements of a 3D array
for x in arr3d:
    for y in x:
        for z in y:
            print(z)

# (method 2) iterate on the elements of a 3D array
(nx,ny,nz) = arr3d.shape
print(nx,ny,nz)

for i in range(nx):
    for j in range(ny):
        for k in range(nz):
            print(arr3d[i,j,k])
```

```
# (method 3) we could also do it using nditer()
for v in np.nditer(arr3d):
    print(v)

# (method 4) nditer with index
for idx, x in np.ndenumerate(arr3d):
    print(idx, x)
```

Array creations

10. Creating arrays

```
a = np.zeros(shape=10)
print(a)
```

[2] ✓ 0.0s

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
b = np.ones(shape=(3,3))
print(b)
```

[3] ✓ 0.0s

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

```
c = np.linspace(0,1,10, endpoint=True)
print(c)
```

[5] ✓ 0.0s

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```

```
d = np.logspace(-1, 3, 10, endpoint=True)
print(d)
```

[10] ✓ 0.0s

```
[1.00000000e-01 2.78255940e-01 7.74263683e-01 2.15443469e+00
 5.99484250e+00 1.66810054e+01 4.64158883e+01 1.29154967e+02
 3.59381366e+02 1.00000000e+03]
```

NumPy: Array operation

11. Mathematical functions: https://numpy.org/doc/stable/reference/routines.math.html

12. Tips: If you are working on scalars, use the “math”, otherwise, use “numpy”.

13.

```
pylist  = [1,2,3,4,5,6,7,8,9,10]
nparray = np.array(pylist)

print(2*pylist)
print(2*nparray)
```

[42] ✓ 0.1s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[ 2  4  6  8 10 12 14 16 18 20]
```

```
print(np.sin(nparray))
print(math.sin(pylist))
```

] ⊗ 0.0s

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 2
      1 print(np.sin(nparray))
----> 2 print(math.sin(pylist))

TypeError: must be real number, not list
```

Exercise: numpy array

1. Numpy could do the numerical integration as well.

2. Check the documentation of numpy.trapz https://numpy.org/doc/stable/reference/generated/numpy.trapz.html

3. Use np.trapz() to do the pi calculation (see page 8).

4. Compare the results with your own numerical integrator.

5. Compare the performance with N=1_000_000.

6. The midpoint method is similar to the function np.sum()

NumPy: Random numbers

1. Random numbers are generated by random number generators.

2. Requires a “seed” to initialize the generator.

3.

```
import numpy.random as random
```

[14] ✓ 0.0s

```
random.seed(seed=123) # optional; used if you want to reproduce the results
```

[16] ✓ 0.0s

4. Random 10 integers from 0 to 100:

```
random.randint(0, 100, 10)
```

] ✓ 0.0s

```
array([52, 70, 26, 80,  6, 14, 75, 54, 71,  1])
```

NumPy: Random numbers

5. Random N float numbers from 0 to 1 (1d numpy array):

```
random.rand(10)
```

✓ 0.0s

```
array([0.48303426, 0.98555979, 0.51948512, 0.61289453, 0.12062867,
       0.8263408 , 0.60306013, 0.54506801, 0.34276383, 0.30412079])
```

6. Random 5x5 float numbers from 0 to 1 (2d numpy array):

```
random.rand(5,5)
```

✓ 0.0s

```
array([[0.41702221, 0.68130077, 0.87545684, 0.51042234, 0.66931378],
       [0.58593655, 0.6249035 , 0.67468905, 0.84234244, 0.08319499],
       [0.76368284, 0.24366637, 0.19422296, 0.57245696, 0.09571252],
       [0.88532683, 0.62724897, 0.72341636, 0.01612921, 0.59443188],
       [0.55678519, 0.15895964, 0.15307052, 0.69552953, 0.31876643]])
```

NumPy: Random numbers

7. Generate a 1-D array containing 10 values, where each value has to be 1,2,3,4,5.

Probability distribution is given to be p = [0.2,0.2,0.3,0.15,0.15]

```
random.choice([1,2,3,4,5], p = [0.2, 0.2, 0.3, 0.15, 0.15], size= 10)
```

✓ 0.0s

```
array([4, 1, 1, 2, 5, 2, 3, 3, 2, 3])
```

6. Shuffle a numpy array:

```
students = ["Einstein", "Newton", "Maxwell"]
random.shuffle(students)
print(students)
```

✓ 0.0s

```
['Maxwell', 'Newton', 'Einstein']
```

Exercise: Random numbers

1. Write a python class named “Gacha” (轉蛋).

2. The class has three attributes, which are numpy arrays. Each array has five elements:

```
self.stars3 = np.array(["a1","a2","a3","a4","a5"])
self.stars4 = np.array(["b1","b2", "b3","b4","b5"])
self.stars5 = np.array(["s1","s2","s3","s4","s5"])
```

3. Write a method called “draw()”. It will return a tuple (star, char), which “star” is an integer from 3 to 5; “char” is a string randomly taken from star3/star4/star5 if star=3/4/5.

4. The probability to receive star=5, star=4, and star=3 are 1%, 9%, and 90%, respectively.

5. Draw 100 times, measure how many 5 stars did you get and how many “s1”?