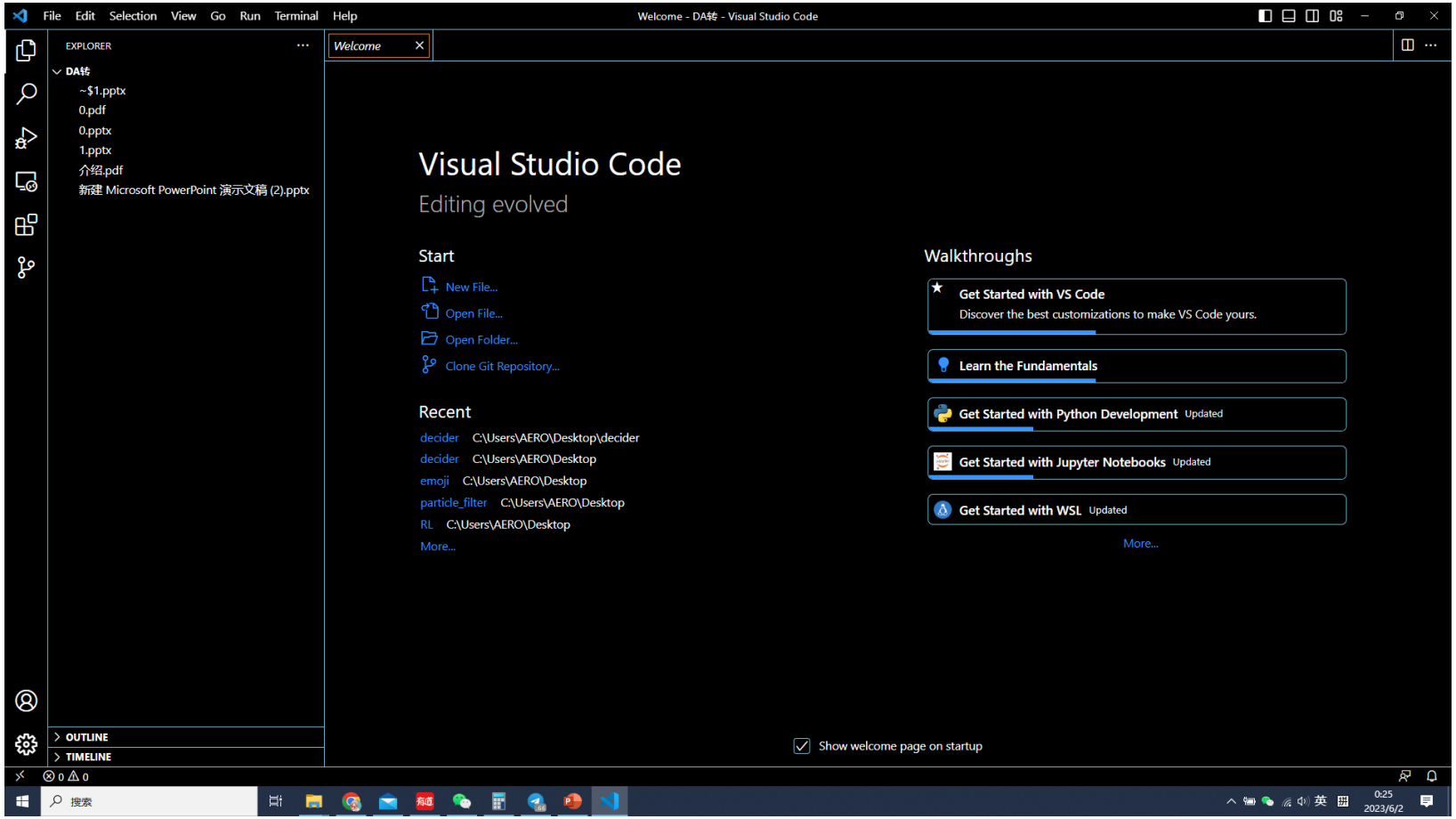
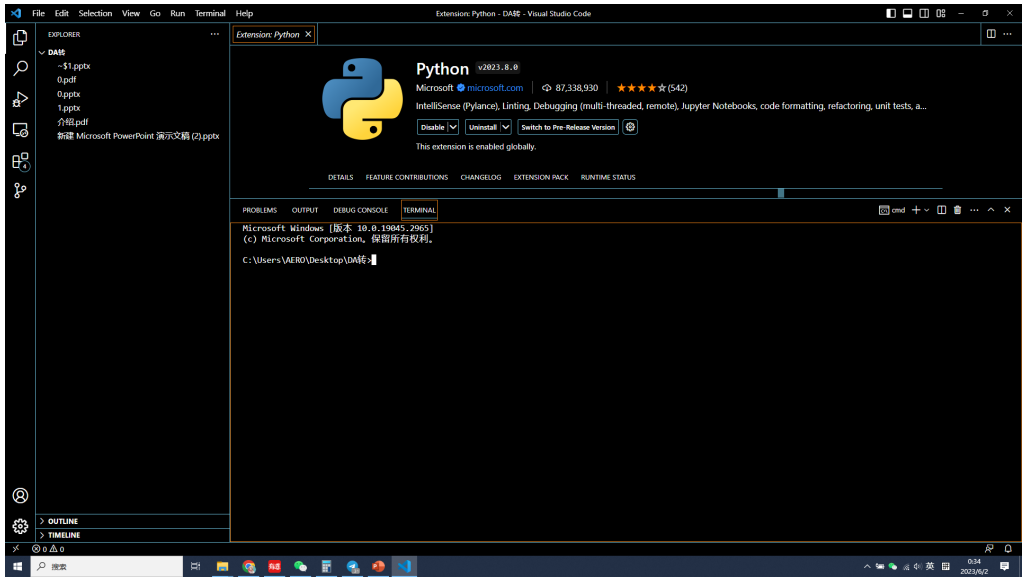
Vscode 简单py 运行环境



Vscode 简单py 运行环境: 安装扩展

Vscode 简单py 运行环境: 简单启动终端

- Terminal->New terminal (or use shortcut: ctrl+shift+`)
- 修改默认终端 powershell 为 Command Prompt （个人喜好）
- View->Command Palette(Ctrl+Shift+P)
- 输入Select Default Profile选择更改
- 此处终端的权限是你vscode的权限
要获得管理员权限的shell需要打开
- 管理员权限的vscode



Vscode 简单py 运行环境: 查看py是否在环境变量中可用

- 终端中输入python --version
- pip --version
- 基础的包管理器

pip是Python的包管理器，用于安装、升级和卸载py包

Pip list 查看已安装包

pip uninstall卸载已安装包

此处出于简便考虑暂不讨论其他如anaconda之类的包管理器，或者虚拟环境等等

Vscode 简单py 运行环境: 运行hello world

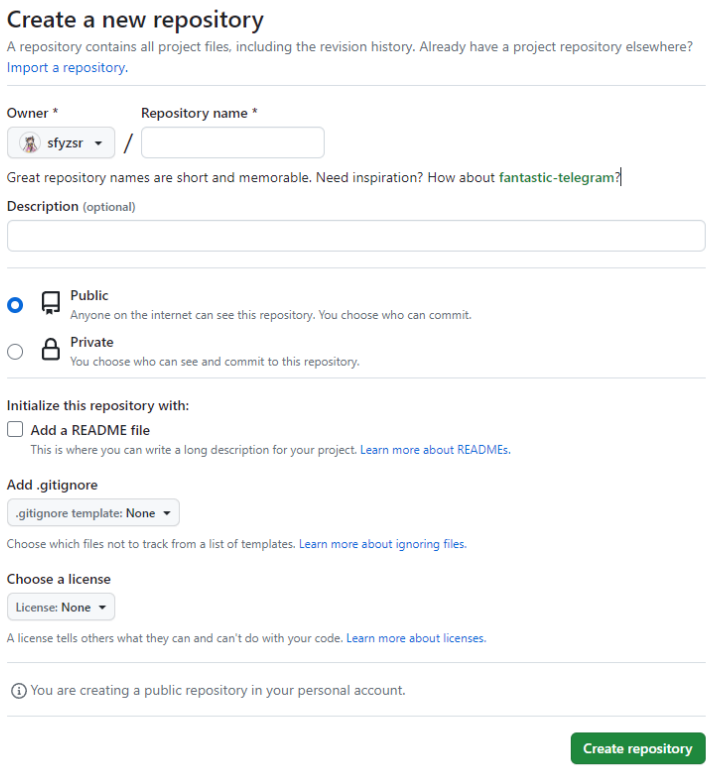
- 左侧新建文件 hello.py
- Print(“hello world”)
- 打开终端
- 输入“py hello.py”

Vscode 简易github 入门

- GitHub是一个基于云端的代码托管平台
- 首先 下载git 添加到环境变量
- Git是一种分布式版本控制系统，用于跟踪和管理项目的代码变更
- 根据网站下载https://git-scm.com/downloads
- 终端中输入git –version判断安装成功

Vscode 简易github 入门

- 这里 我们只讨论简单功能 创建仓库 上传代码 更新仓库 拉取代码



sfyzsr/learn

github.com/sfyzsr/learn

编程相关 深度学习 动画漫画 网路电台 游戏攻略 想法 ICE sfyzsr Google 翻译 登录 Courses Dashboard My Timetable Cyber Soul Radio gmail 设置 Google COMP390 - View... COMP390 Honou... 其他书签

Search or jump to... / Pull requests Issues Codespaces Marketplace Explore

sfyzsr / learn Public

Pin Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Invite collaborators
Find people using their GitHub username or email address.

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/sfyzsr/learn.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# learn" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/sfyzsr/learn.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/sfyzsr/learn.git
git branch -M main
git push -u origin main
```

...or import code from another repository

搜索 英 0:59 2023/6/2

Vscode 简易github 入门

```
C:\Users\AERO\Desktop\DA转>git init
Initialized empty Git repository in C:/Users/AERO/Desktop/DA转/.git/

C:\Users\AERO\Desktop\DA转>git add -A

C:\Users\AERO\Desktop\DA转>git commit -m "update"
[master (root-commit) 84c0da3] update
 6 files changed, 1 insertion(+)
 create mode 100644 0.pdf
 create mode 100644 0.pptx
 create mode 100644 1.pptx
 create mode 100644 1.py
 create mode 100644 "\344\273\213\347\273\215.pdf"
 create mode 100644 "\346\226\260\345\273\272 Microsoft PowerPoint \346\274\224\347\244\272\346\226\207\347\250\277 (2).pptx"

C:\Users\AERO\Desktop\DA转>git branch -M main

C:\Users\AERO\Desktop\DA转>git remote add origin https://github.com/sfyzsr/learn.git

C:\Users\AERO\Desktop\DA转>git push -u origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 1.58 MiB | 1.62 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/sfyzsr/learn.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

C:\Users\AERO\Desktop\DA转>
```

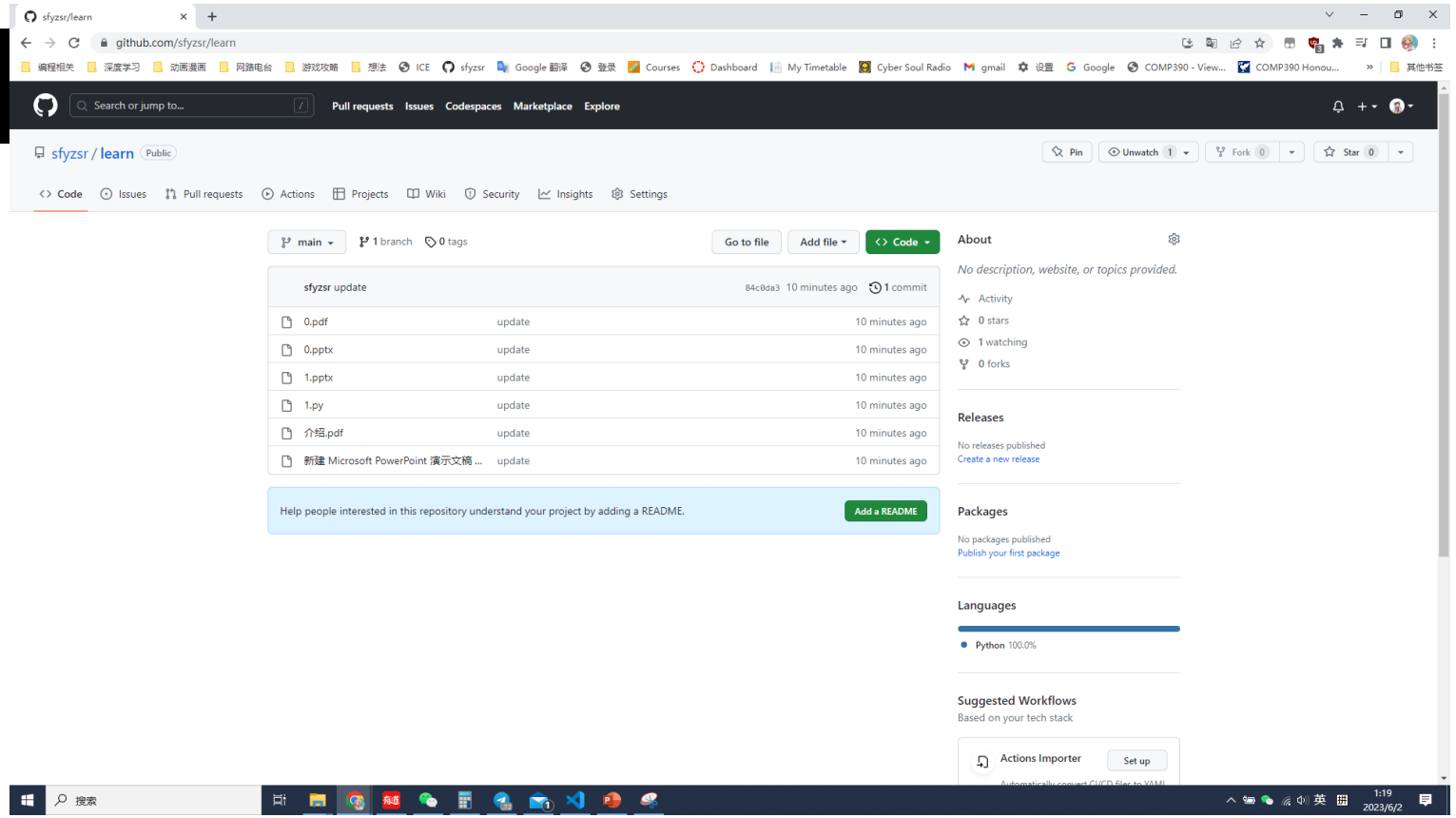
Vscode 简易github 入门

- Git init 初始本地git仓库
- Git add -A 添加所有变动到Staging area（暂存区）
- git commit -m “update” 创建一个提交 将暂存区变动提交到仓库
- git branch -M main 重命名当前分支为main
- git remote add origin https://github.com/sfyzsr/learn.git 将本地与线上链接 此处的origin仅仅是个默认名字类似于main
- git push -u origin main 推送本地仓库代码到到远程仓库中的指定分支

Vscode 简易github 入门

- Git pull 拉取更新到本地

```
C:\Users\AERO\Desktop\DA转>git pull
Already up to date.
```



python

- Basic Data Types
- For loop
- While loop

Python function

定义函数

在Python中可以使用`def`关键字来定义函数，和变量一样每个函数也有一个响亮的名字，而且命名规则跟变量的命名规则是一致的。在函数名后面的圆括号中可以放置传递给函数的参数，这一点和数学上的函数非常相似，程序中函数的参数就相当于是数学上说的函数的自变量，而函数执行完成后我们可以通过`return`关键字来返回一个值，这相当于数学上说的函数的因变量。

在了解了如何定义函数后，我们可以对上面的代码进行重构，所谓重构就是在不影响代码执行结果的前提下对代码的结构进行调整，重构之后的代码如下所示。

```
"""
输入M和N计算C(M,N)

Version: 0.1
Author: 骆昊
"""
def fac(num):
    """求阶乘"""
    result = 1
    for n in range(1, num + 1):
        result *= n
    return result


m = int(input('m = '))
n = int(input('n = '))
# 当需要计算阶乘的时候不用再写循环求阶乘而是直接调用已经定义好的函数
print(fac(m) // fac(n) // fac(m - n))
```

> **说明：** Python的`math`模块中其实已经有一个名为`factorial`函数实现了阶乘运算，事实上求阶乘并不用自己定义函数。下面的例子中，我们讲的函数在Python标准库已经实现过了，我们这里是为了讲解函数的定义和使用才把它们又实现了一遍，**实际开发中并不建议做这种低级的重复劳动。**

函数的参数

函数是绝大多数编程语言中都支持的一个代码的"构建块"，但是Python中的函数与其他语言中的函数还是有很多不太相同的地方，其中一个显著的区别就是Python对函数参数的处理。在Python中，函数的参数可以有默认值，也支持使用可变参数，所以Python并不需要像其他语言一样支持函数的重载，因为我们在定义一个函数的时候可以让它有多种不同的使用方式，下面是两个小例子。

```
from random import randint


def roll_dice(n=2):
    """摇色子"""
    total = 0
    for _ in range(n):
        total += randint(1, 6)
    return total


def add(a=0, b=0, c=0):
    """三个数相加"""
    return a + b + c


# 如果没有指定参数那么使用默认值摇两颗色子
print(roll_dice())
# 摇三颗色子
print(roll_dice(3))
print(add())
print(add(1))
print(add(1, 2))
print(add(1, 2, 3))
# 传递参数时可以不按照设定的顺序进行传递
print(add(c=50, a=100, b=200))
```

用模块管理函数

对于任何一种编程语言来说，给变量、函数这样的标识符起名字都是一个让人头疼的问题，因为我们会遇到命名冲突这种尴尬的情况。最简单的场景就是在同一个.py文件中定义了两个同名函数，由于Python没有函数重载的概念，那么后面的定义会覆盖之前的定义，也就意味着两个函数同名函数实际上只有一个是存在的。

```
def foo():
    print('hello, world!')


def foo():
    print('goodbye, world!')


# 下面的代码会输出什么呢?
foo()
```

当然上面的这种情况我们很容易就能避免，但是如果项目是由多人协作进行团队开发的时候，团队中可能有多个程序员都定义了名为`foo`的函数，那么怎么解决这种命名冲突呢？答案其实很简单，Python中每个文件就代表了一个模块（module），我们在不同的模块中可以有同名的函数，在使用函数的时候我们通过`import`关键字导入指定的模块就可以区分到底要使用的是哪个模块中的`foo`函数，代码如下所示。

`module1.py`

```
def foo():
    print('hello, world!')
```

`module2.py`

```
def foo():
    print('goodbye, world!')
```

`test.py`

```
from module1 import foo

# 输出hello, world!
foo()
```

Python为字符串类型提供了非常丰富的运算符，我们可以使用`+`运算符来实现字符串的拼接，可以使用`*`运算符来重复一个字符串的内容，可以使用`in`和`not in`来判断一个字符串是否包含另外一个字符串（成员运算），我们也可以用`[]`和`[:]`运算符从字符串取出某个字符或某些字符（切片运算），代码如下所示。

```
s1 = 'hello ' * 3
print(s1) # hello hello hello
s2 = 'world'
s1 += s2
print(s1) # hello hello hello world
print('ll' in s1) # True
print('good' in s1) # False
str2 = 'abc123456'
# 从字符串中取出指定位置的字符(下标运算)
print(str2[2]) # c
# 字符串切片(从指定的开始索引到指定的结束索引)
print(str2[2:5]) # c12
print(str2[2:]) # c123456
print(str2[2::2]) # c246
print(str2[::2]) # ac246
print(str2[::-1]) # 654321cba
print(str2[-3:-1]) # 45
```

在Python中，我们还可以通过一系列的方法来完成对字符串的处理，代码如下所示。

```
str1 = 'hello, world!'
# 通过内置函数len计算字符串的长度
print(len(str1)) # 13
# 获得字符串首字母大写的拷贝
print(str1.capitalize()) # Hello, world!
# 获得字符串每个单词首字母大写的拷贝
print(str1.title()) # Hello, World!
# 获得字符串变大写后的拷贝
print(str1.upper()) # HELLO, WORLD!
# 从字符串中查找子串所在位置
print(str1.find('or')) # 8
print(str1.find('shit')) # -1
# 与find类似但找不到子串时会引发异常
# print(str1.index('or'))
# print(str1.index('shit'))
# 检查字符串是否以指定的字符串开头
```

生成式和生成器

我们还可以使用列表的生成式语法来创建列表，代码如下所示。

```
f = [x for x in range(1, 10)]
print(f)
f = [x + y for x in 'ABCDE' for y in '1234567']
print(f)
# 用列表的生成表达式语法创建列表容器
# 用这种语法创建列表之后元素已经准备就绪所以需要耗费较多的内存空间
f = [x ** 2 for x in range(1, 1000)]
print(sys.getsizeof(f))  # 查看对象占用内存的字节数
print(f)
# 请注意下面的代码创建的不是一个列表而是一个生成器对象
# 通过生成器可以获取到数据但它不占用额外的空间存储数据
# 每次需要数据的时候就通过内部的运算得到数据(需要花费额外的时间)
f = (x ** 2 for x in range(1, 1000))
print(sys.getsizeof(f))  # 相比生成式生成器不占用存储数据的空间
print(f)
for val in f:
    print(val)
```

定义类

在Python中可以使用`class`关键字定义类，然后在类中通过之前学习过的函数来定义方法，这样就可以将对象的动态特征描述出来，代码如下所示。

```
class Student(object):

    # __init__是一个特殊方法用于在创建对象时进行初始化操作
    # 通过这个方法我们可以为学生对象绑定name和age两个属性
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def study(self, course_name):
        print('%s正在学习%s.' % (self.name, course_name))

    # PEP 8要求标识符的名字用全小写多个单词用下划线连接
    # 但是部分程序员和公司更倾向于使用驼峰命名法(驼峰标识)
    def watch_movie(self):
        if self.age < 18:
            print('%s只能观看《熊出没》.' % self.name)
        else:
            print('%s正在观看岛国爱情大电影.' % self.name)
```

> **说明：** 写在类中的函数，我们通常称之为（对象的）方法，这些方法就是对象可以接收的消息。

创建和使用对象

当我们定义好一个类之后，可以通过下面的方式来创建对象并给对象发消息。

```
def main():
    # 创建学生对象并指定姓名和年龄
    stu1 = Student('骆昊', 38)
    # 给对象发study消息
    stu1.study('Python程序设计')
    # 给对象发watch_av消息
    stu1.watch_movie()
    stu2 = Student('王大锤', 15)
    stu2.study('思想品德')
    stu2.watch_movie()


if __name__ == '__main__':
    main()
```

访问可见性问题

对于上面的代码，有C++、Java、C#等编程经验的程序员可能会问，我们给`Student`对象绑定的`name`和`age`属性到底具有怎样的访问权限（也称为可见性）。因为在很多面向对象编程语言中，我们通常会将对象的属性设置为私有的（private）或受保护的（protected），简单的说就是不允许外界访问，而对象的方法通常都是公开的（public），因为公开的方法就是对象能够接受的消息。在Python中，属性和方法的访问权限只有两种，也就是公开的和私有的，如果希望属性是私有的，在给属性命名时可以用两个下划线作为开头，下面的代码可以验证这一点。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')


def main():
    test = Test('hello')
    # AttributeError: 'Test' object has no attribute '__bar'
    test.__bar()
    # AttributeError: 'Test' object has no attribute '__foo'
    print(test.__foo)


if __name__ == "__main__":
    main()
```

但是，Python并没有从语法上严格保证私有属性或方法的私密性，它只是给私有的属性和方法换了一个名字来妨碍对它们的访问，事实上如果你知道更换名字的规则仍然可以访问到它们，下面的代码就可以验证这一点。之所以这样设定，可以用这样一句名言加以解释，就是"**We are all consenting adults here**"。因为绝大多数程序员都认为开放比封闭要好，而且程序员要自己为自己的行为负责。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo
```

python

- 目前讲到8 没有讲到的部分看看就行 用的不多（或者现阶段用不到，用到再说）

Relational Terminology

Database: Set of named relations

Relation (Table):

Schema: description ("metadata") `Student(sid: int, name: text, dept: text)`

Instance: collection of data satisfying the schema

Tuple (record, row)

sid	name	dept
12344	Jones	CS
12355	Smith	Physics
12366	Gold	CS

Attribute (field, column)

Relational Tables

Schema is fixed

Unique attribute names, attribute types are **atomic**

`Student(sid: int, name: text, dept: text)`

Instances can change often

In SQL, an instance is a **multiset** (bag) of tuples

name	dept	age
Jones	CS	18
Smith	Physics	21
Jones	CS	18

Example Database

Student(sid, name, dept, age)

sid	name	dept	age
12344	Jones	CS	18
12355	Smith	Physics	23
12366	Gold	CS	21

Enrolled(sid, cid, grade)

sid	cid	grade
12344	INF-10080	65
12355	INF-11199	72
12355	INF-11122	61
12366	INF-10080	80
12344	INF-11199	53

Course(cid, name, year)

cid	name	year
INF-11199	Advanced Database Systems	2020
INF-10080	Introduction to Databases	2020
INF-11122	Foundations of Databases	2019
INF-11007	Data Mining and Exploration	2019

BASIC SINGLE-TABLE QUERIES

```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
```

```
SELECT *
  FROM Student
 WHERE age = 18
```

Get all 18-year-old students

Simplest version is straightforward

- Produce all tuples in the table that match the predicate
- Output the expressions in the SELECT list
- Expression can be a column reference, or an arithmetic expression over column refs
- DISTINCT removes duplicate rows before output

```
SELECT DISTINCT cid
  FROM Enrolled
 WHERE grade > 95
```

Get IDs of courses with grades > 95

ORDER BY

ORDER BY <column*> [ASC|DESC]

Sort the output tuples by the values in one or more of their columns

```
SELECT sid, grade FROM Enrolled
 WHERE cid = 'INF-11199'
 ORDER BY grade
```

sid	grade
12344	53
12399	72
12355	72
12311	76

Ascending order by default, but can be overridden

Can mix and match, lexicographically

```
SELECT sid, grade FROM Enrolled
 WHERE cid = 'INF-11199'
 ORDER BY grade DESC, sid ASC
```

sid	grade
12311	76
12355	72
12399	72
12344	53

LIMIT

```
SELECT sid, grade FROM Enrolled
 WHERE cid = 'INF-11199'
 ORDER BY grade
```

sid	grade
12344	53
12399	72
12355	72
12311	76

LIMIT <count> [offset]

Limit the # of tuples returned in the output

```
SELECT sid, grade FROM Enrolled
 WHERE cid = 'INF-11199'
 ORDER BY grade LIMIT 3
```

sid	grade
12344	53
12399	72
12355	72

Typically used with **ORDER BY**

Otherwise the output is **non-deterministic**, depends on the algo for query processing

Can set an offset to skip first records

```
SELECT sid, grade FROM Enrolled
 WHERE cid = 'INF-11199'
 ORDER BY grade LIMIT 3 OFFSET 1
```

sid	grade
12399	72
12355	72
12311	76

AGGREGATES

Functions that return a summary (aggregate) of some arithmetic expression from a bag of tuples

Get the average age of CS students

```
SELECT AVG(age) AS avg_age
  FROM Student WHERE dept = 'CS'
```

avg_age
20.5

Get the average age and # of CS students

```
SELECT AVG(age) AS avg_age,
       COUNT(sid) AS cnt
  FROM Student WHERE dept = 'CS'
```

avg_age	cnt
20.5	153

Aggregate functions can only be used in the **SELECT** list

Other aggregates: SUM, COUNT, MIN, MAX

Group By

Get the average age per department

```
SELECT dept, AVG(age) AS avg_age
  FROM Student
 GROUP BY dept
```

dept	avg_age
CS	20.5
Physics	21.1
Maths	19.8

Partition table into groups with the same GROUP BY column values

- Can group by a list of columns

Produce an aggregate result per group

- Cardinality of output = # of distinct group values

Can put grouping columns in the **SELECT** output list

Group By

Non-aggregated values in SELECT output clause must appear in GROUP BY clause

```
SELECT dept, name, AVG(age)
  FROM Student
 GROUP BY dept
```

```
SELECT dept, name, AVG(age)
  FROM Student
 GROUP BY dept, name
```

Filter Groups

Get the average age per department

```
SELECT dept, AVG(age) AS avg_age
  FROM Student
 GROUP BY dept
```

dept	avg_age
CS	20.5
Physics	21.1
Maths	19.8

Get departments with average student age above 21

```
SELECT dept, AVG(age) AS avg_age
  FROM Student
 WHERE avg_age > 21
 GROUP BY dept
```

dept	avg_age
Physics	21.1

HAVING

Get departments with average student age above 21

```
SELECT dept, AVG(age) AS avg_age
  FROM Student
 GROUP BY dept
HAVING AVG(age) > 21
```

HAVING filters results **after** grouping and aggregation

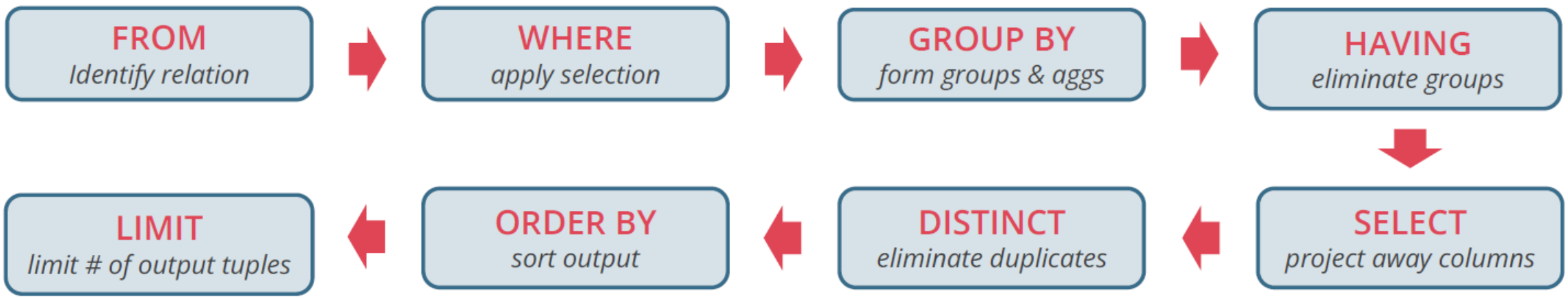
- Hence can contain anything that could go in the SELECT list
- I.e., GROUP BY columns or aggregates (e.g., `COUNT(*) > 5`)

HAVING can only be used in aggregate queries

- It's an optional clause

Conceptual SQL Evaluation

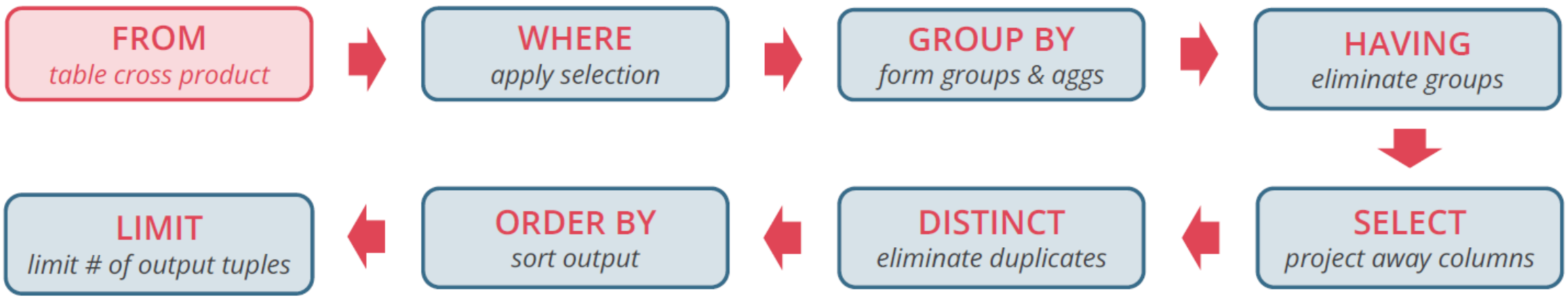
```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list> [HAVING <predicate>]]
[ORDER BY <column list>] [LIMIT <count>]
```



Does not imply the query will actually be evaluated this way!

Multiple-Table Queries

```
SELECT [DISTINCT] <column expression list>
  FROM <table1 [AS t1], ..., tableN [AS tn]>
[WHERE <predicate>]
[GROUP BY <column list> [HAVING <predicate>]]
[ORDER BY <column list>] [LIMIT <count>]
```



This evaluation strategy is almost always inefficient!

简单练习

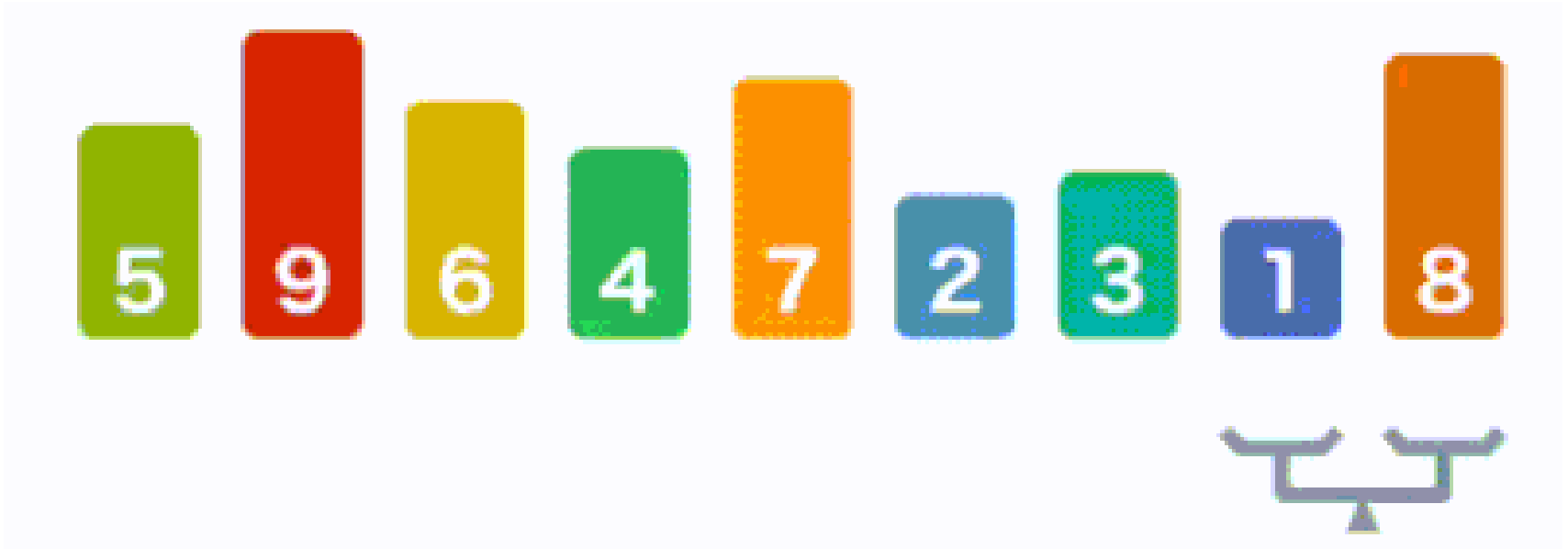
- Bubble Sort
- 输入[64, 34, 25, 12, 22, 11, 90]打乱数组

- 输出[11,12,22,25,34,64,90]
- 排序好的数组

- 可能的练习地址:
  https://practice.geeksforgeeks.org/problems/bubble-sort/1?utm_source=geeksforgeeks&utm_medium=article_practice_tab&utm_campaign=article_practice_tab

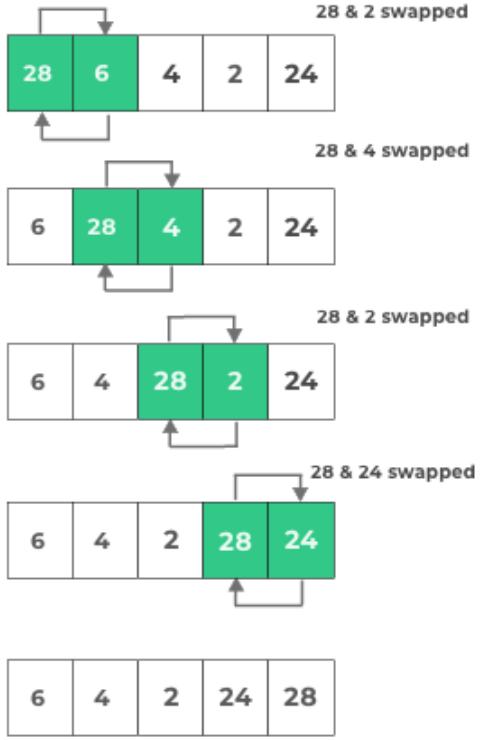
简单练习

- Bubble Sort

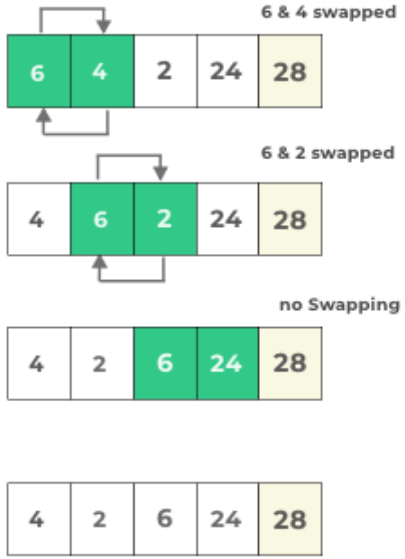


Bubble Sort in C++

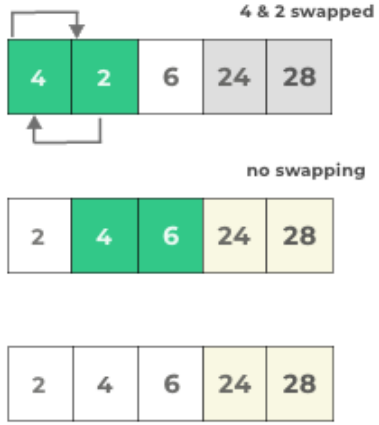
Pass 1



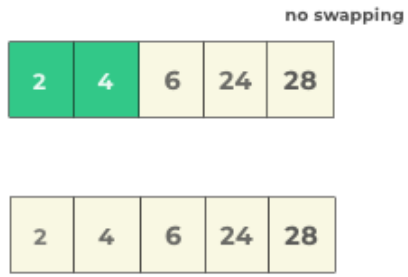
Pass 2



Pass 3



Pass 4



Final Result

PrepInsta

Algorithm 1: Bubble sort

Data: Input array *A[]*
Result: Sorted *A[]*

```
int i, j, k;
N = length(A);
for j = 1 to N do
    for i = 0 to N-1 do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
        end
    end
end
```

SQL

- SELECT 10 + 32 FROM Student

The query returns a table with one tuple containing 42.

The query returns a table with tuples 42, 42, 42, ... , repeated as many times as there are tuples in Student.

This is not a valid SQL query.

The query returns a table with one tuple containing the string '10 + 32'.

SQL

- Consider the following schema **Enrolled(sid, cid, grade).**
- Find the IDs of the students with top-5 grades, ordered by student ID (lowest to highest). The reported student IDs may contain duplicates (e.g., if one student has more than one top-5 grades).

```
SELECT sid
FROM Enrolled
ORDER BY grade DESC
LIMIT 5

SELECT sid
FROM Enrolled
ORDER BY sid
LIMIT 5

SELECT sid
FROM Enrolled
ORDER BY grade DESC, sid ASC
LIMIT 5

None of the above
```

SQL

Response Feedback: ORDER BY grade DESC is not correct because it does not order the output by sid.

ORDER BY sid is not correct because it does not order the students from highest to lowest grades first.

ORDER BY grade DESC, sid ASC is not correct because it orders by grades and then by sid but only for tied grades.

This query finds the 5 students with highest grades first and then sorts their sids in ascending order.

```
SELECT t.*
FROM (
   SELECT sid
   FROM Enrolled
   ORDER BY grade DESC
   LIMIT 5
) AS t
ORDER BY t.sid
```