Recitation Class

Week 1

Syllabus

Please note that these course schedules are subject to change based on actual circumstances and are for reference only.

Week1-Week3	C/C++ Programmiong，OOP	**Week10-Week11**	Graph Traversal, Topological Sorting, Minimum Spanning Tree
Week4-Week5	Arrays, Linked Lists, Stacks, Queues, Hash Tables, Asymptotic Notation	**Week12-Week 13**	Dijkstra, Bellman-Ford, A*, Floyd-Warshall
Week6-Week7	Bubble Sort, Insertion Sort, Merge Sort, Quick Sort	**Wekk14-Week16**	AI Tools
Week8 - Week9	BFS, DFS, Binary Trees, Heaps, Balanced Binary Trees		

Grading Policy

Homework	5%
Quiz	5%
Midterm	30%
Final	30%
Project	30%

Usage of AI tools

- The use of AI tools for searching information or obtaining assistance **directly** on homework is **strictly prohibited**.
- **All solutions must be derived from your own understanding and effort.**
- Submissions will be reviewed carefully, and any indication of reliance on AI-generated content will result in severe penalties.
- We employ a sophisticated AI-detection tool to scrutinize code for AI generation. We reserve the right to impose the most severe academic penalties if it is determined that your code was either fully or partially generated by an AI. (**Including ChatGPT, Copilot, Cursor, Gemini, Claude, etc.**)
- Additionally, following each recitation class that includes homework, your code will undergo a manual check.

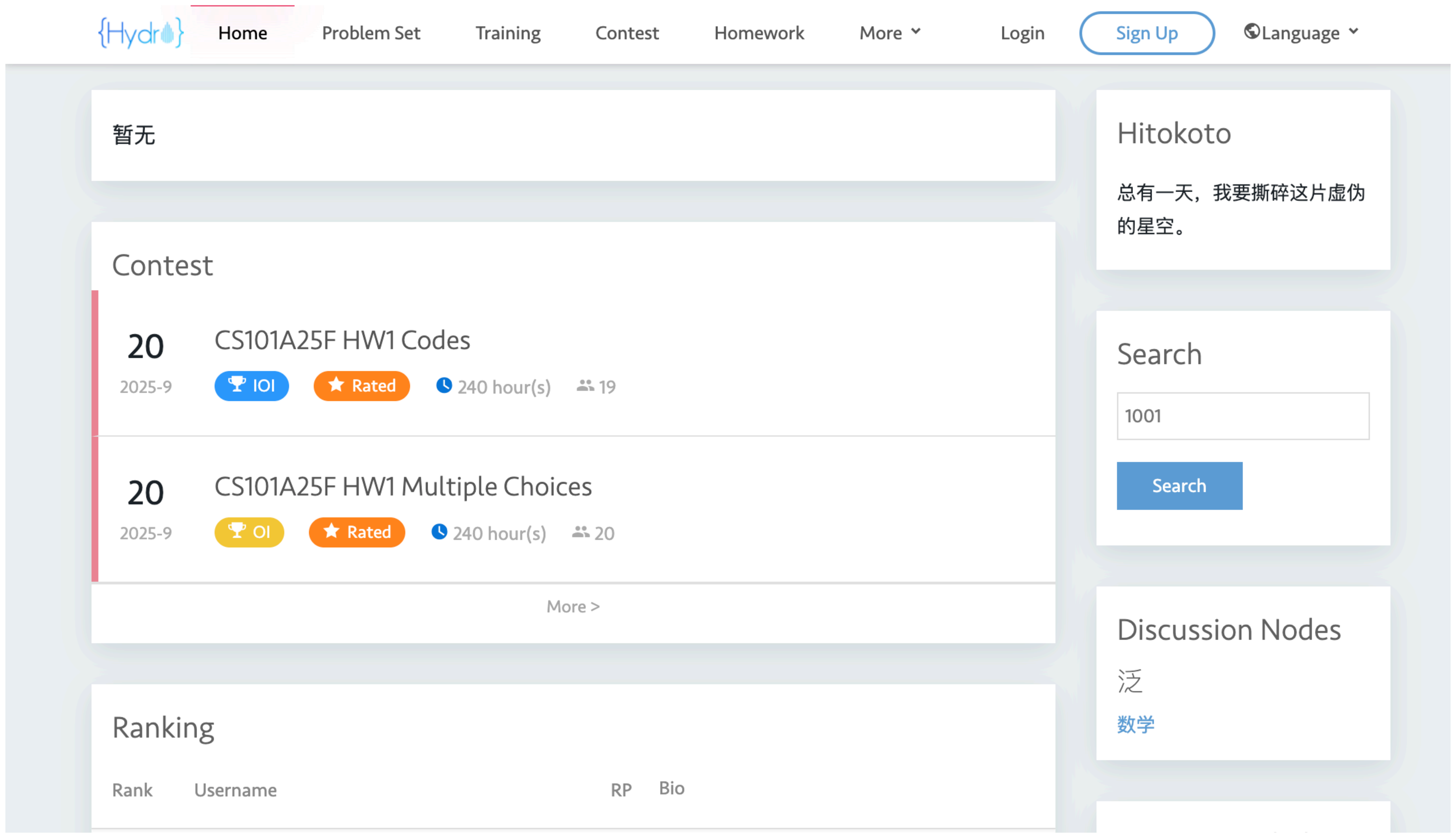
Academic Integrity

- **Your honest, individual effort is the foundation of true learning and growth in this course.**
- What is Prohibited:
  - **Plagiarism**: Submitting work that is not your own without proper citation.
  - **Unauthorized Collaboration**: Working with others on assignments meant to be individual.
  - **Use of Prohibited Tools**: Using generative AI tools unless explicitly permitted for a specific task.
  - **Cheating**: Any dishonest attempt to gain an advantage in exams or assignments.
- Why This Matters:
  - **Protects Your Learning**: True understanding comes from your own effort.
  - **Ensures Fairness**: Creates a level playing field for all students.
- **Any violation will be treated with the utmost seriousness, resulting in severe penalties ranging from a zero on the assignment to an automatic course failure.**
- We are here to support your success through honest hard work.

OJ

https://acm.shanghaitech.edu.cn/d/CS101A2025Fall/

- Register your account with your ShanghaiTech email "xxxxx@shanghaitech.edu.cn"
- Submit your code via your account.
- Your grade is based on the score of the latest submission by default.
- Usually, MCQ answers come out after the deadline. You get your coding scores right away, but extra credit submissions usually show o points at first.



暂无

Contest

20 2025-9 CS101A25F HW1 Codes IOI Rated 240 hour(s) 19

20 2025-9 CS101A25F HW1 Multiple Choices OI Rated 240 hour(s) 20

More >

Ranking

Rank	Username	RP	Bio

Hitokoto

总有一天，我要撕碎这片虚伪的星空。

Search

1001

Search

Discussion Nodes

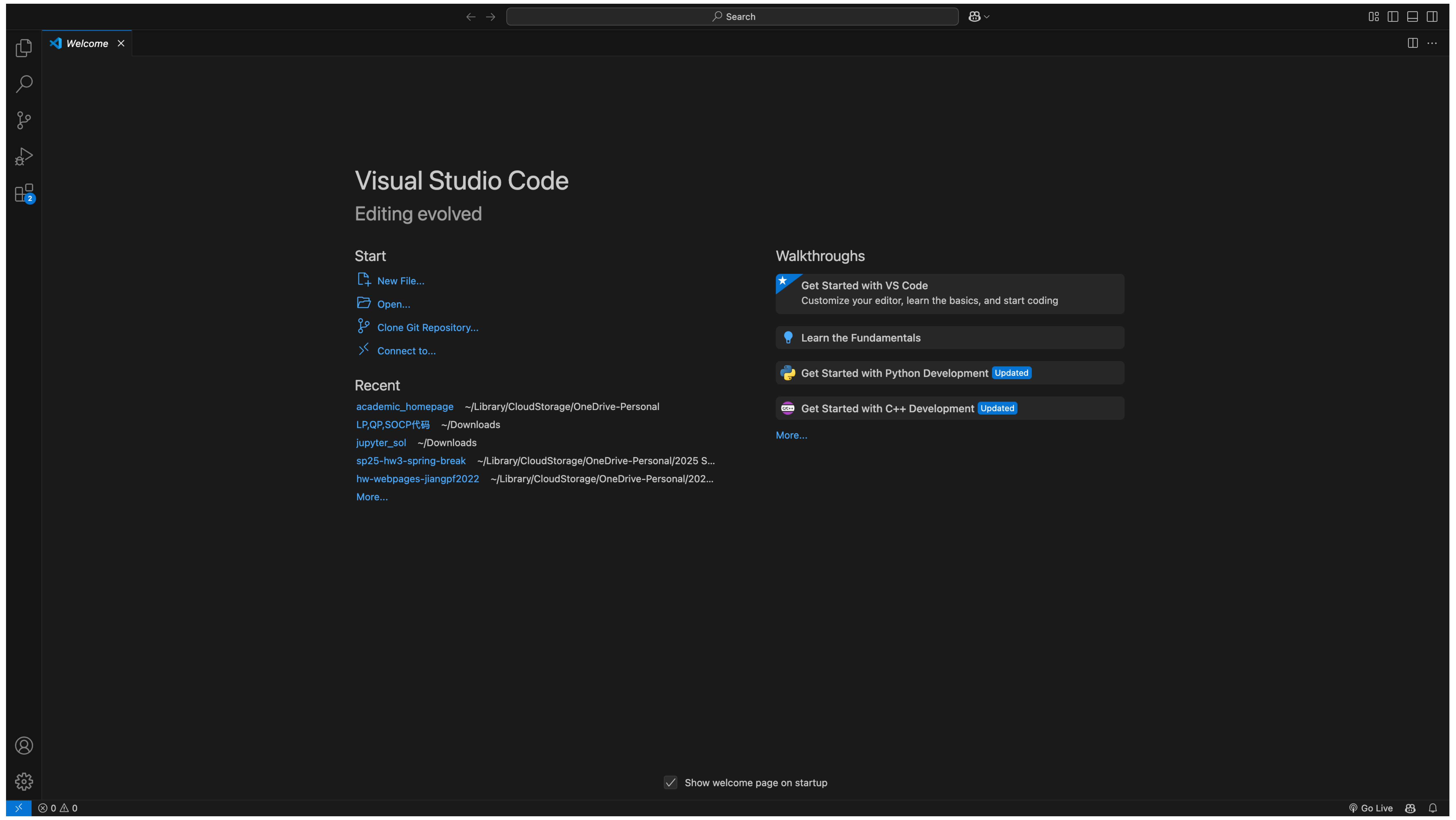
泛

数学

VSCode

https://code.visualstudio.com/

- **Lightweight yet Powerful**: Fast startup, low resource use, but incredibly capable through extensions.
- **Intelligent Code Completion**: Suggests code as you type, catching errors and speeding up development.
- **Integrated Debugger**: Step through your code, inspect variables, and find bugs efficiently.
- **Built-in Terminal & Git**: Compile, run, and manage your code version without switching windows.
- **Massive Extension Library**: Customize for any language (C++, Python, etc.) or workflow.



Visual Studio Code
Editing evolved

Start
New File...
Open...
Clone Git Repository...
Connect to...

Recent
academic_homepage ~/Library/CloudStorage/OneDrive-Personal
LP,QP,SOCP代码 ~/Downloads
jupyter_sol ~/Downloads
sp25-hw3-spring-break ~/Library/CloudStorage/OneDrive-Personal/2025 S...
hw-webpages-jiangpf2022 ~/Library/CloudStorage/OneDrive-Personal/202...
More...

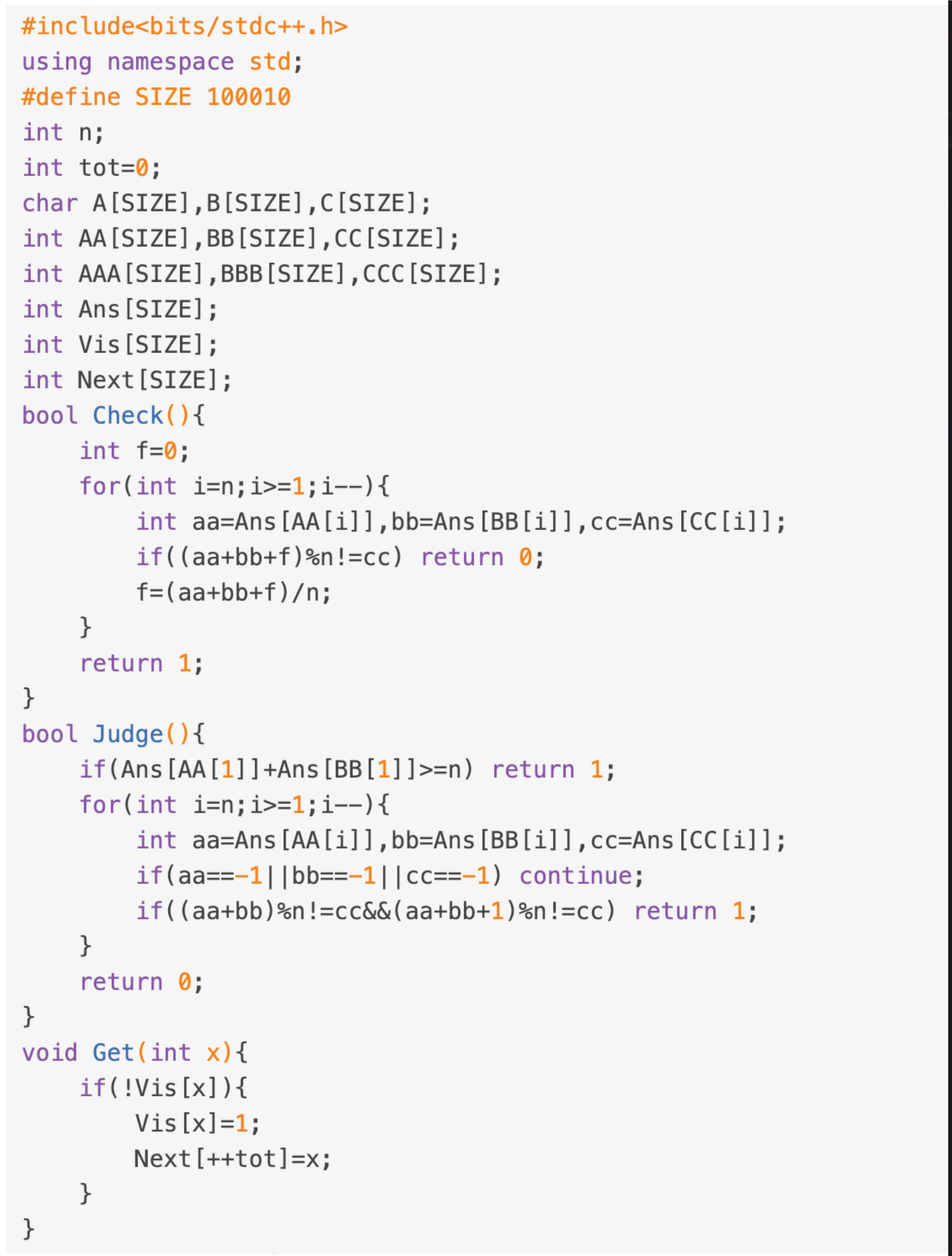
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Learn the Fundamentals
Get Started with Python Development Updated
Get Started with C++ Development Updated
More...

Show welcome page on startup

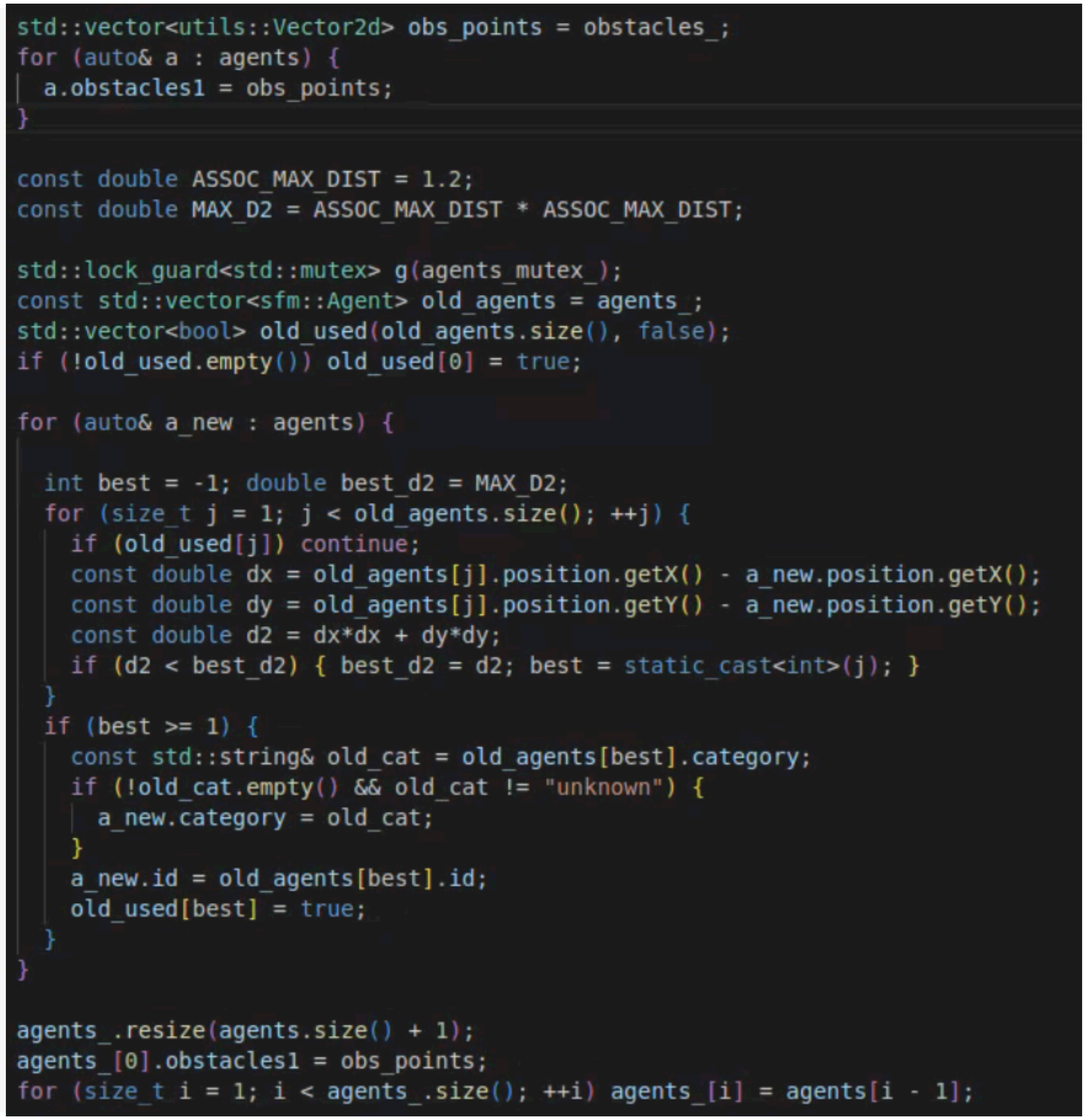
Code Style

Why important?

- Makes code easier to read for others.
- Easier to read for yourself.
- Also makes code aesthetically pleasing
- Not just for you but also for other guys! (and that guy may be yourself three months later)
- There is no rule for code style, but you should always code with your logic.



```cpp
#include<bits/stdc++.h>
using namespace std;
#define SIZE 100010
int n;
int tot=0;
char A[SIZE],B[SIZE],C[SIZE];
int AA[SIZE],BB[SIZE],CC[SIZE];
int AAA[SIZE],BBB[SIZE],CCC[SIZE];
int Ans[SIZE];
int Vis[SIZE];
int Next[SIZE];
bool Check(){
    int f=0;
    for(int i=n;i>=1;i--){
        int aa=Ans[AA[i]],bb=Ans[BB[i]],cc=Ans[CC[i]];
        if((aa+bb+f)%n!=cc) return 0;
        f=(aa+bb+f)/n;
    }
    return 1;
}
bool Judge(){
    if(Ans[AA[1]]+Ans[BB[1]]>=n) return 1;
    for(int i=n;i>=1;i--){
        int aa=Ans[AA[i]],bb=Ans[BB[i]],cc=Ans[CC[i]];
        if(aa==-1||bb==-1||cc==-1) continue;
        if((aa+bb)%n!=cc&&(aa+bb+1)%n!=cc) return 1;
    }
    return 0;
}
void Get(int x){
    if(!Vis[x]){
        Vis[x]=1;
        Next[++tot]=x;
    }
}
```



```cpp
std::vector<utils::Vector2d> obs_points = obstacles_;
for (auto& a : agents) {
  a.obstacles1 = obs_points;
}

const double ASSOC_MAX_DIST = 1.2;
const double MAX_D2 = ASSOC_MAX_DIST * ASSOC_MAX_DIST;

std::lock_guard<std::mutex> g(agents_mutex_);
const std::vector<sfm::Agent> old_agents = agents_;
std::vector<bool> old_used(old_agents.size(), false);
if (!old_used.empty()) old_used[0] = true;

for (auto& a_new : agents) {

  int best = -1; double best_d2 = MAX_D2;
  for (size_t j = 1; j < old_agents.size(); ++j) {
    if (old_used[j]) continue;
    const double dx = old_agents[j].position.getX() - a_new.position.getX();
    const double dy = old_agents[j].position.getY() - a_new.position.getY();
    const double d2 = dx*dx + dy*dy;
    if (d2 < best_d2) { best_d2 = d2; best = static_cast<int>(j); }
  }
  if (best >= 1) {
    const std::string& old_cat = old_agents[best].category;
    if (!old_cat.empty() && old_cat != "unknown") {
      a_new.category = old_cat;
    }
    a_new.id = old_agents[best].id;
    old_used[best] = true;
  }
}

agents_.resize(agents.size() + 1);
agents_[0].obstacles1 = obs_points;
for (size_t i = 1; i < agents_.size(); ++i) agents_[i] = agents[i - 1];
```

Code Style

Naming Rules

- **Snake Case**
  - find_location
  - train_acc
- **Little Camel Case**
  - evaluateAccuracyGpu
  - numEpochs
- **Big Camel Case**
  - InitWeight
  - CrossRntropyLoss
  - **Do not use this for variables.*

Code Style

Naming Rules

- **Good variables names:**
  - Reflect its value or function.
  - Eliminate ambiguity.
  - Fit the enviroment and its function.
- **Good names:**
  - read_from_csv, FibSeries, countAllMoves ...
- **AWFUL names:**
  - ooOOoo,l1L111ll, I_HATE_TA, xx, a,x,r,i,k,j ...

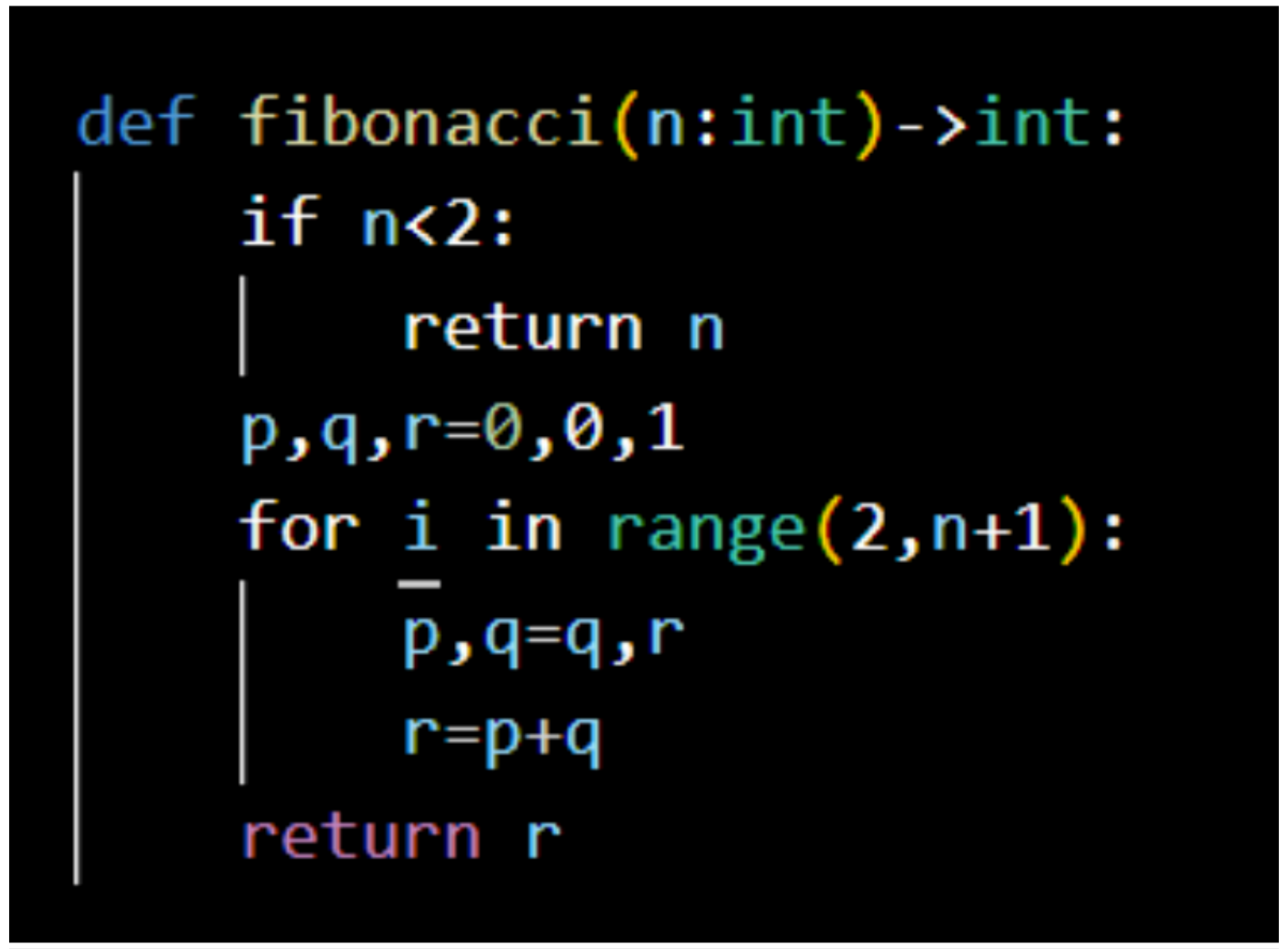
Use as less magic number as you can!

If you have to, remember to write a comment about the magic number you use.

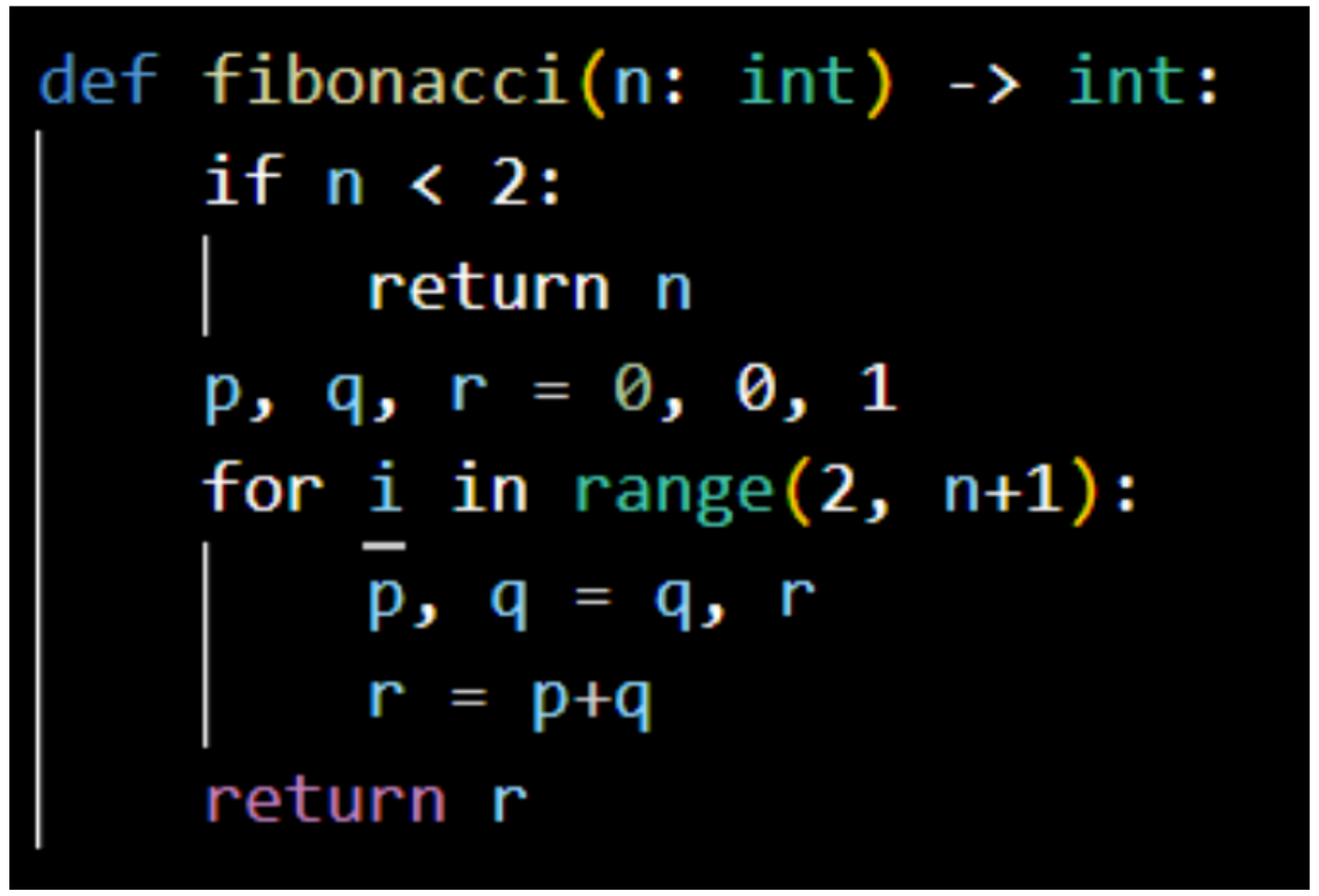
Code Style

Whitespaces

- It is impropiate to use **whitespaces** after a punctuation mark as you are writing a English article, like:
  - ”What+ does? spring== look, like. on@ Jupiter”
- You **SHOULD** use whitespaces before & after some operators like +, −, ==, >, and = .
  - For example: 1 + 1, ans += 1



```
def fibonacci(n:int)->int:
    if n<2:
        return n
    p,q,r=0,0,1
    for i in range(2,n+1):
        p,q=q,r
        r=p+q
    return r
```



```
def fibonacci(n: int) -> int:
    if n < 2:
        return n
    p, q, r = 0, 0, 1
    for i in range(2, n+1):
        p, q = q, r
        r = p+q
    return r
```

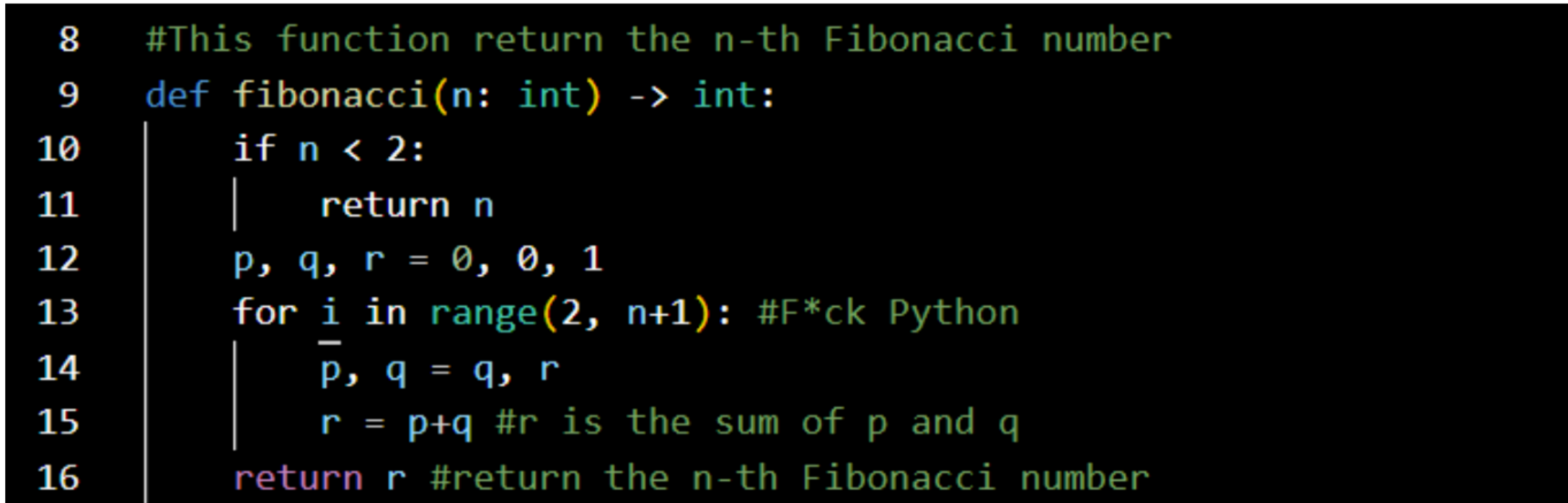
Code Style

Comments

- **Meaningful comments are**
  - Complicate calculus/control flow/binary magic/magic number
  - Regular expressions
  - No nonsense
  - Better in English
- **Awful comments are**
  - Transliteral of your code
  - Hard to read
  - Unrelated to the content

Good naming reduce the need of commenting!

Which one is meaningful comment?



```python
#This function return the n-th Fibonacci number
def fibonacci(n: int) -> int:
    if n < 2:
        return n
    p, q, r = 0, 0, 1
    for i in range(2, n+1): #F*ck Python
        p, q = q, r
        r = p+q #r is the sum of p and q
    return r #return the n-th Fibonacci number
```

Makefile

How Does Your Code Become an Executable Program?

- **The Manual Compilation Steps:**
  - Preprocessing: Handles header files, macros (g++ -E).
  - Compilation: Translates C++ into machine code (object files .o, g++ -c).
  - Linking: Combines multiple object files into one executable (g++).
- **A Simple Example:**
  - You have one file: hello.cpp
  - Command: g++ -o hello hello.cpp
  - Run it: ./hello

Makefile

How to Manage a Project with Multiple Files?

- **Method 1: One Big Command:**
  - g++ -o my_program main.cpp utils.cpp
  - Disadvantage: If you change just utils.cpp, both files are recompiled.
  - Slow!

Structure

```
project/
  ├── main.cpp
  ├── utils.cpp
  └── utils.h
```

Makefile

How to Manage a Project with Multiple Files?

- **Method 2: Separate Compilation:**
  - # 1. Compile each .cpp file into a .o object file
  - g++ -c main.cpp   # produces main.o
  - g++ -c utils.cpp   # produces utils.o
  - # 2. Link all .o files into the final program
  - g++ -o my_program main.o utils.o
- Big Advantage: If you only change utils.cpp, just re-run the last two steps.
- Fast!
- New Problem: Too many commands to remember and type. Tedious and error-prone!

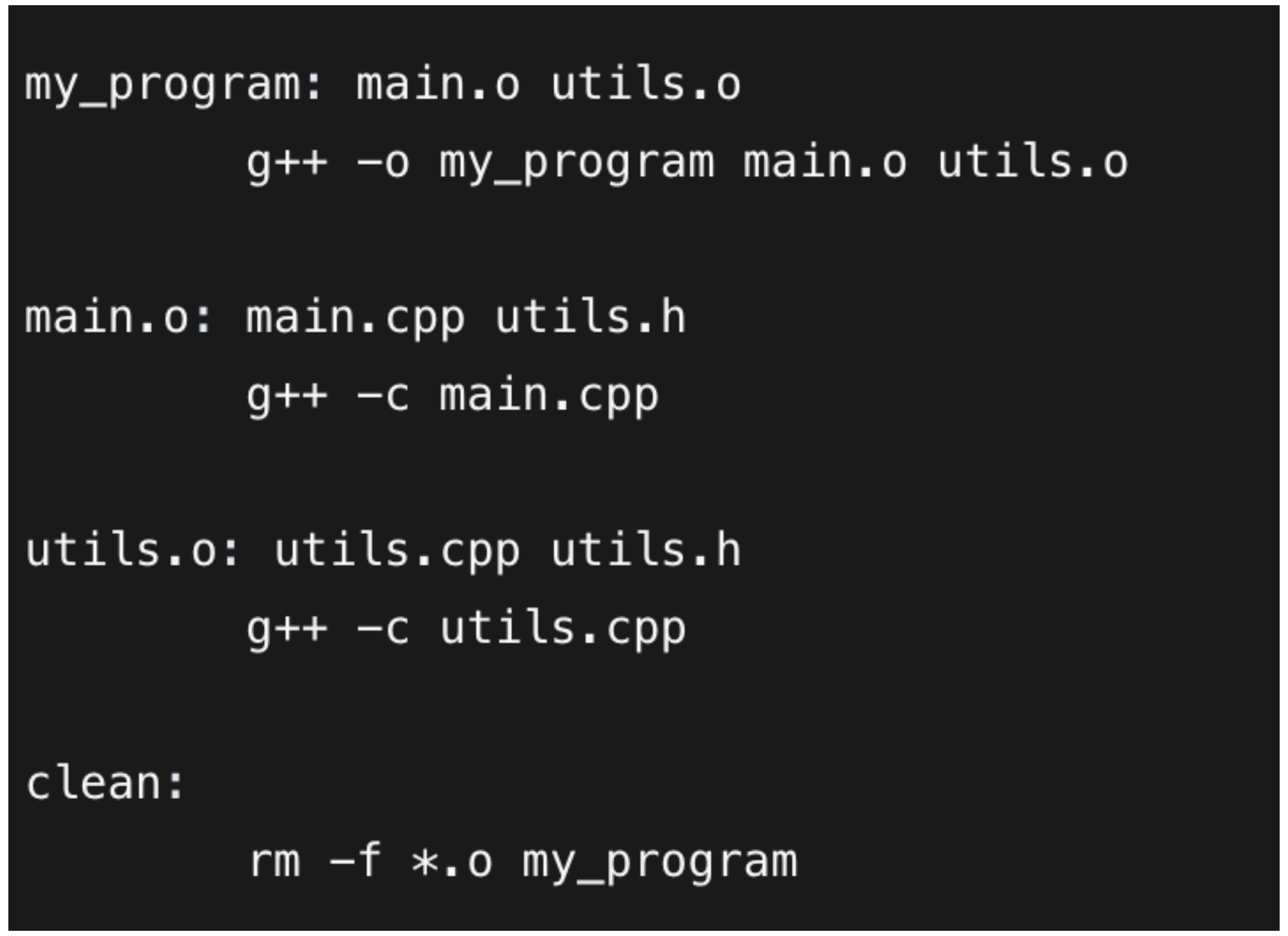
Structure

```
project/
  ├── main.cpp
  ├── utils.cpp
  └── utils.h
```

Makefile

Makefile: Your Automated Build Assistant

- Concept: A Makefile is a text file that defines:
  - Target: What you want to build (e.g., an executable or object file).
  - Dependencies: What files the target needs (e.g., .cpp or .h files).
  - Recipe/Command: How to build the target from its dependencies.
- Run ***make*** in your project directory. It automatically runs the necessary commands.
- Run ***make clean*** to remove all generated files.



```
my_program: main.o utils.o
	g++ -o my_program main.o utils.o

main.o: main.cpp utils.h
	g++ -c main.cpp

utils.o: utils.cpp utils.h
	g++ -c utils.cpp

clean:
	rm -f *.o my_program
```

Makefile

Why Use a Makefile?

- **The Three Key Benefits:**
  - **Automation**: One command, make, replaces long, complex compilation commands.
  - **Efficiency**: The make tool is smart. It checks file timestamps and only recompiles what has changed, saving a huge amount of time.
  - **Clarity**: The Makefile acts as clear documentation for how your project is built.
- **The New Workflow:**
  - You write code (.cpp, .h).
  - You write a Makefile (define the build rules).
  - You run make (compile automatically).
  - You run your program ./my_program.
- Start using a Makefile for your homework1 today!

```
my_program: main.o utils.o
        g++ -o my_program main.o utils.o

main.o: main.cpp utils.h
        g++ -c main.cpp

utils.o: utils.cpp utils.h
        g++ -c utils.cpp

clean:
        rm -f *.o my_program
```