Introduction to computing B

网站：
https://calvinxiaocao.github.io/#/teaching/fa24/B/IntroToComputingB

Introduction to computing B

2024-2025 Fall Final Mini-lecture

十院联合期末学业辅导活动

Something about ME!

曹彧 · Calvin Cao

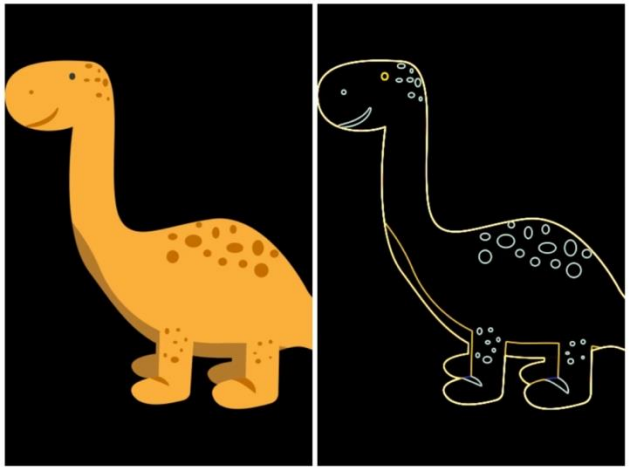
信息科学技术学院 2023级 本科生

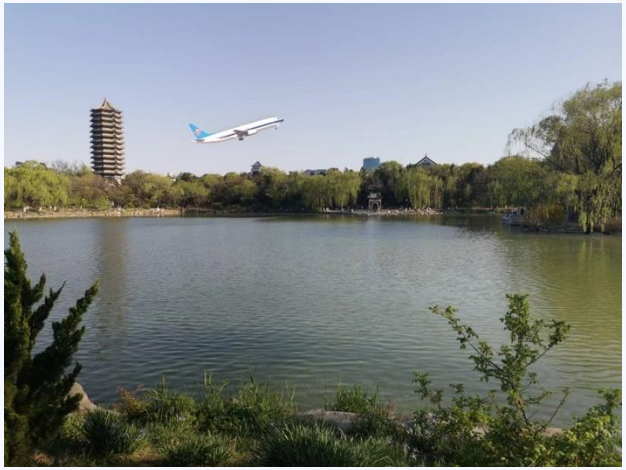
目前学习AI方向，对计算机视觉、图形学领域感兴趣
（高中无竞赛和编程基础）

My projects … (Python based)

Image blending (CV)

My projects … (C based)





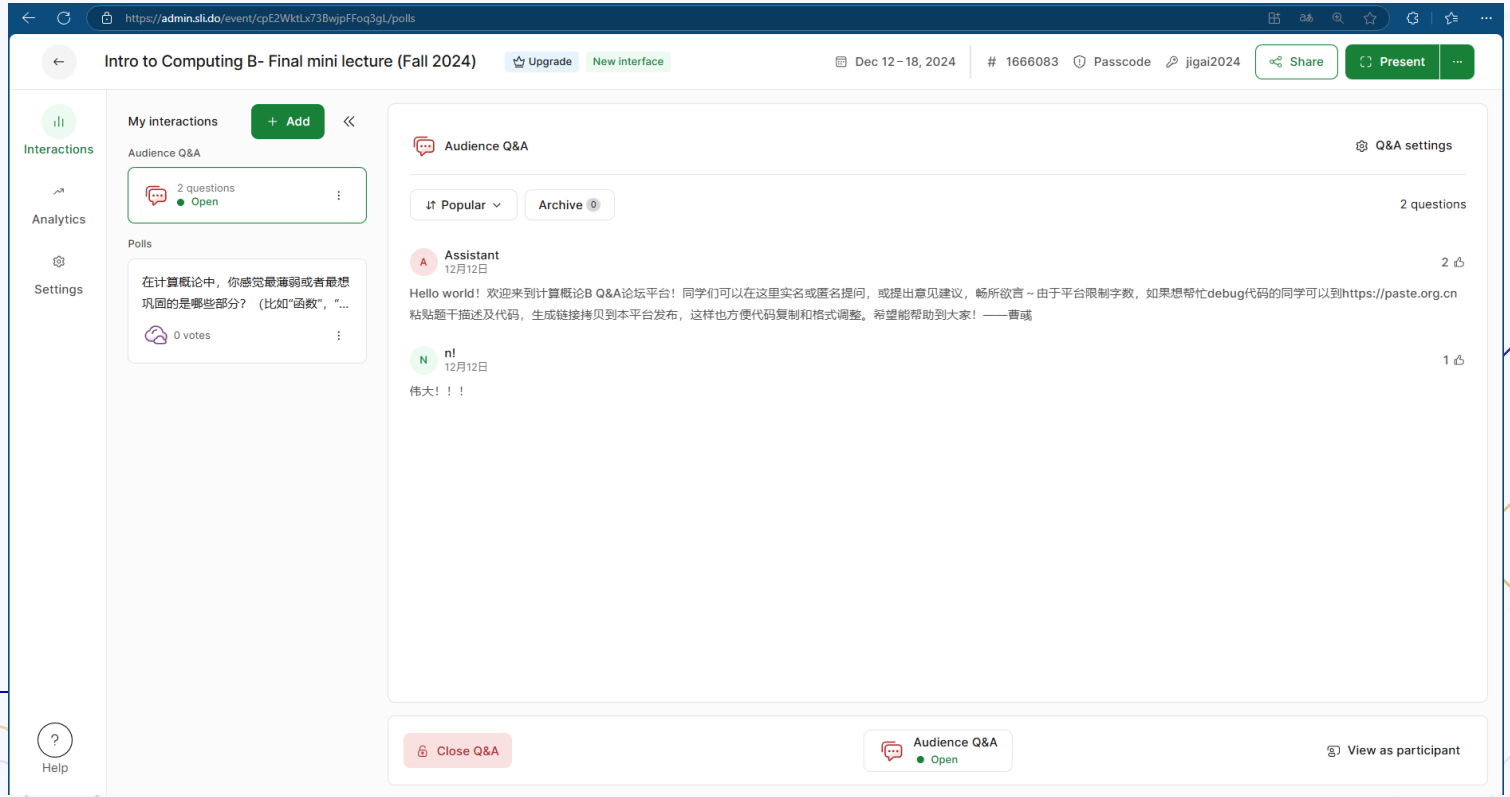
Computer Graphics (VCL)

Something about The Activity!

Web

Something about The Activity!

Q&A



Something about The Activity!

NOTE: 由于资源限制，本次辅导为C语言和Python合上。

但是Python和c的底层思想逻辑（如递归、枚举、二分、动态规划等）是相通的，同学们无需担心。

（统计：Python和c的同学占比是多少？）

Table of contents 目录

Before all… Some suggestions!

1. Open your laptops, and **Enjoy C & Python!**
2. 计概是门**实践出真知**的课程，学会和编程语言“互动”
3. 尽量少用AI等辅助工具，而是养成独立思考的习惯
4. 要有一定量的**题目练习**，可以掐时间进行“模考”
5. **大作业**可能会有一定挑战性，因此**Start early!**
6. 遇到不会的问题及时向老师、TA、其他同学求助

“A language isn't something you learn so much as something you join.”

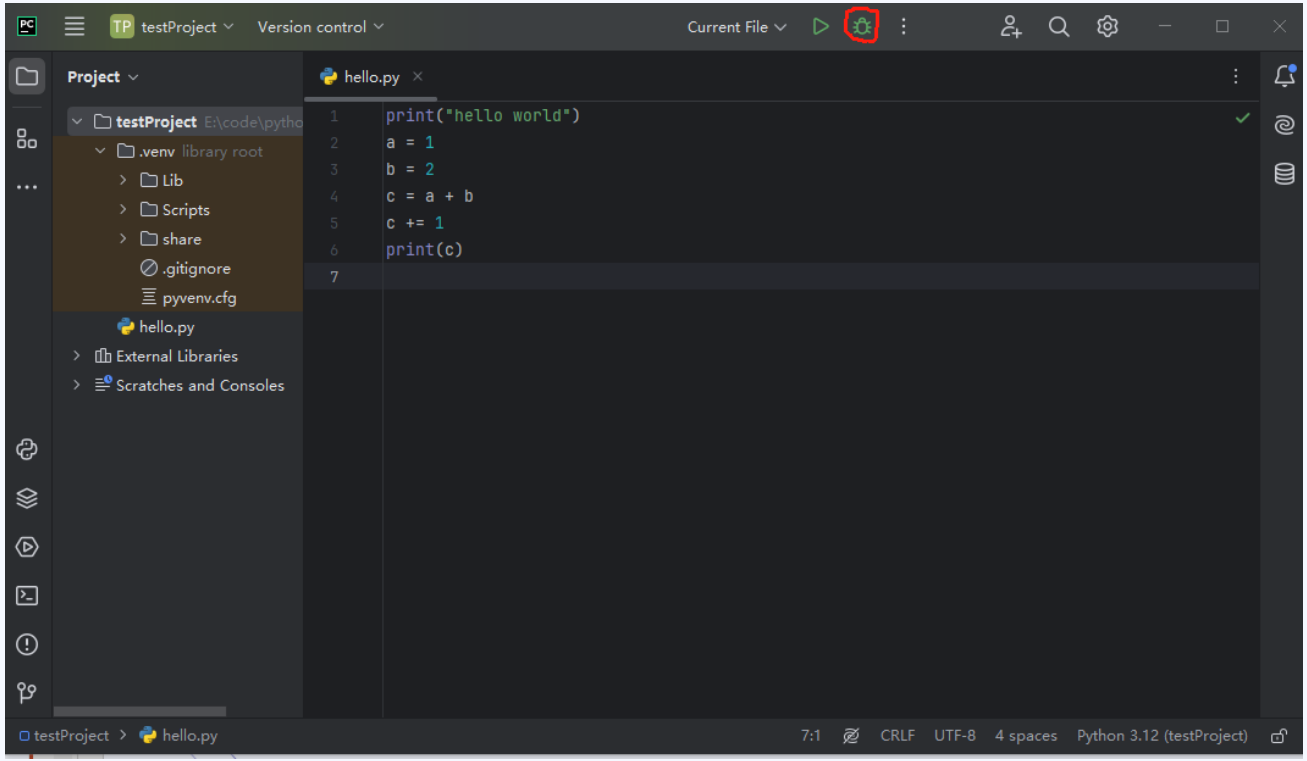
—**CS61A Textbook**

01

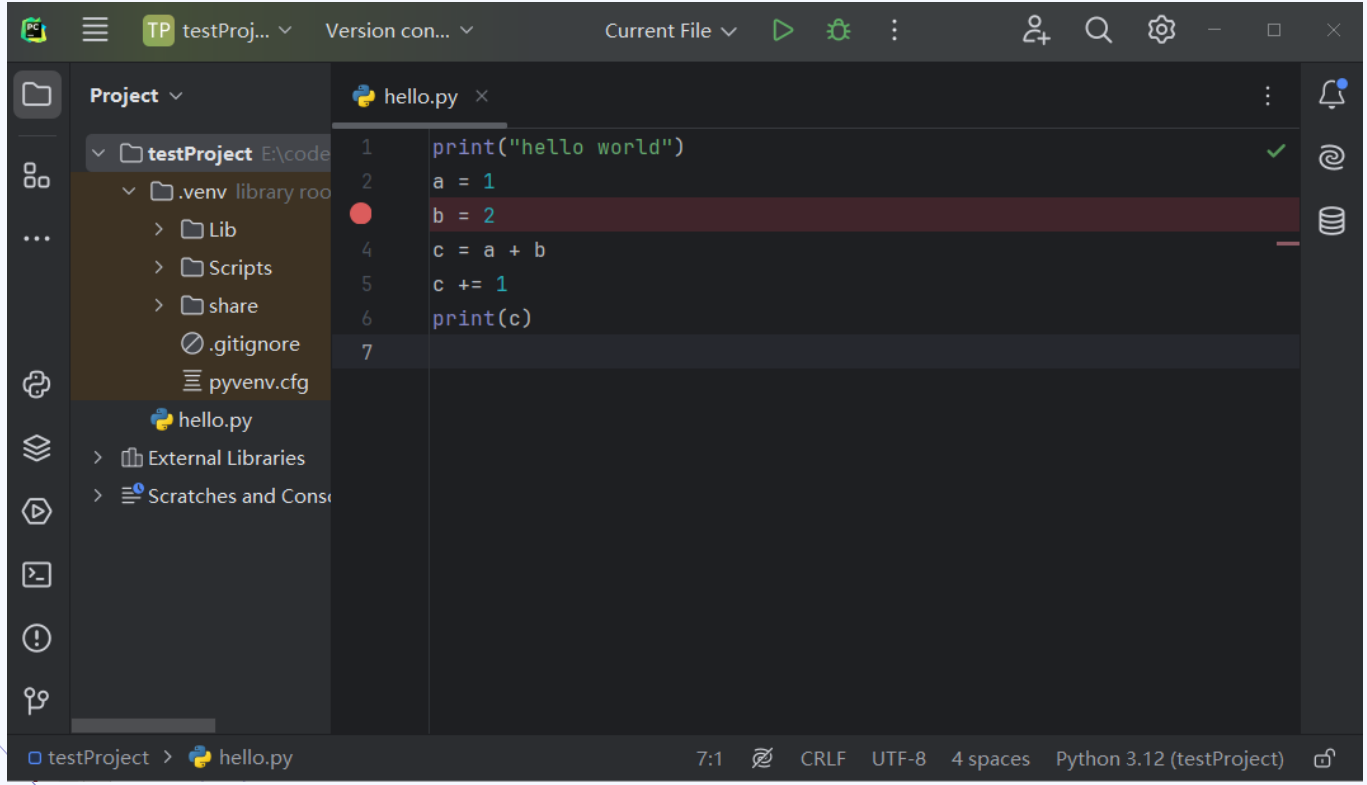
Leading Part

C与Python的断点调试法

Debugging Mode (DEMO!)

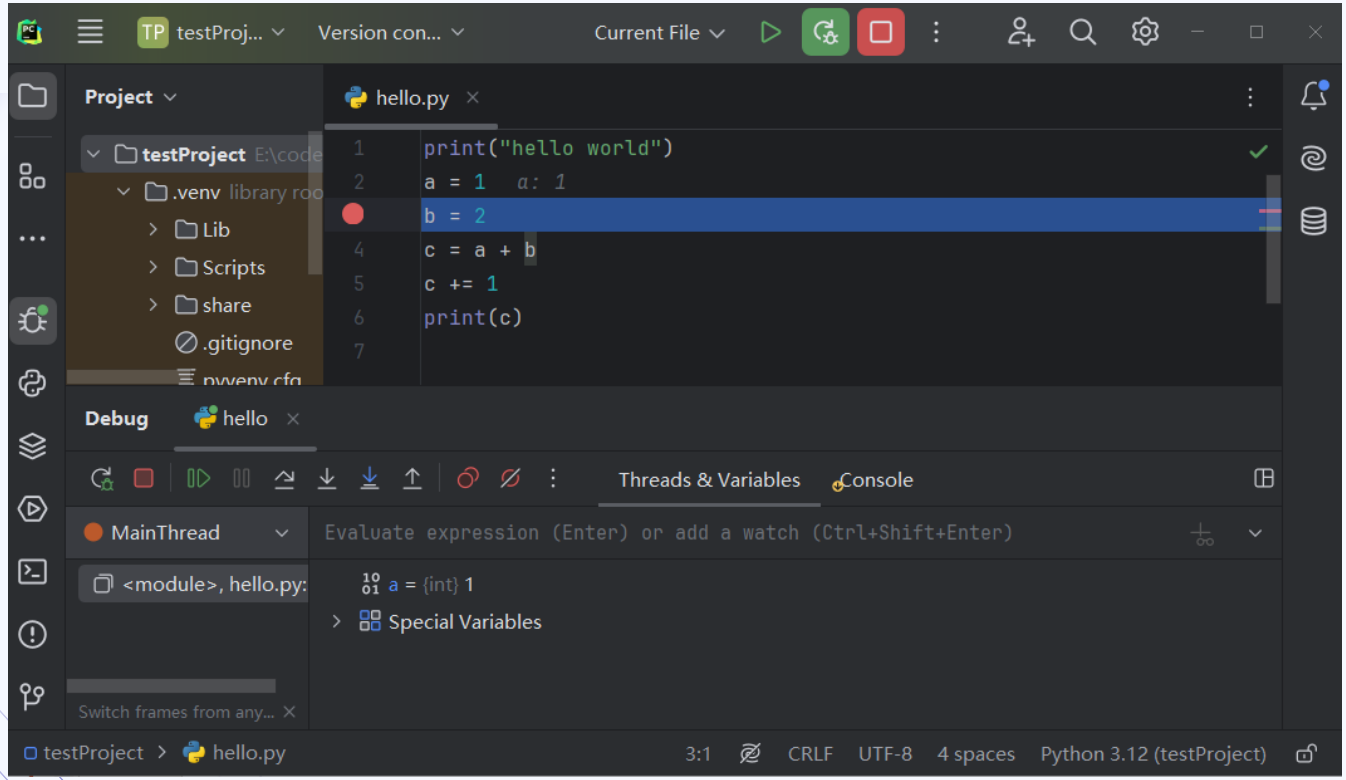


Pycharm Debugging Mode



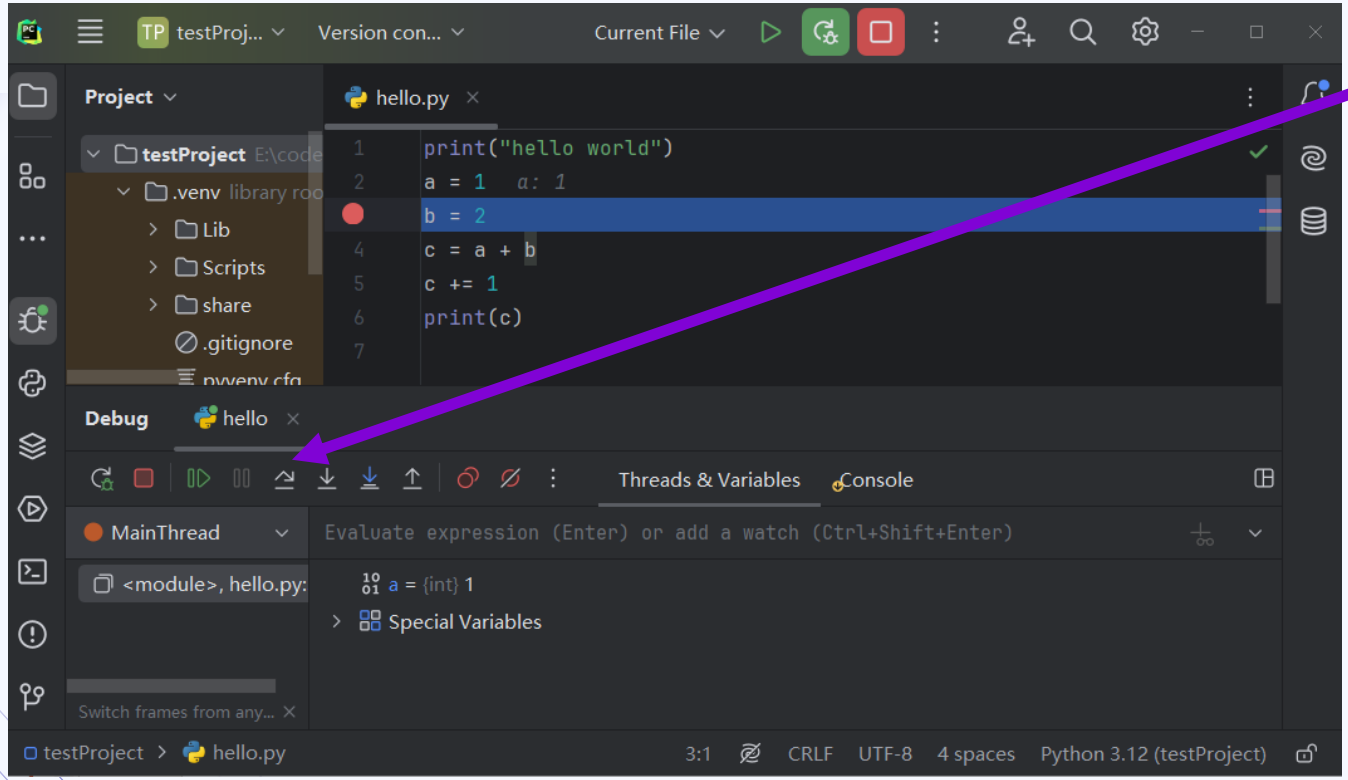
断点
Breakpoint

Pycharm Debugging Mode



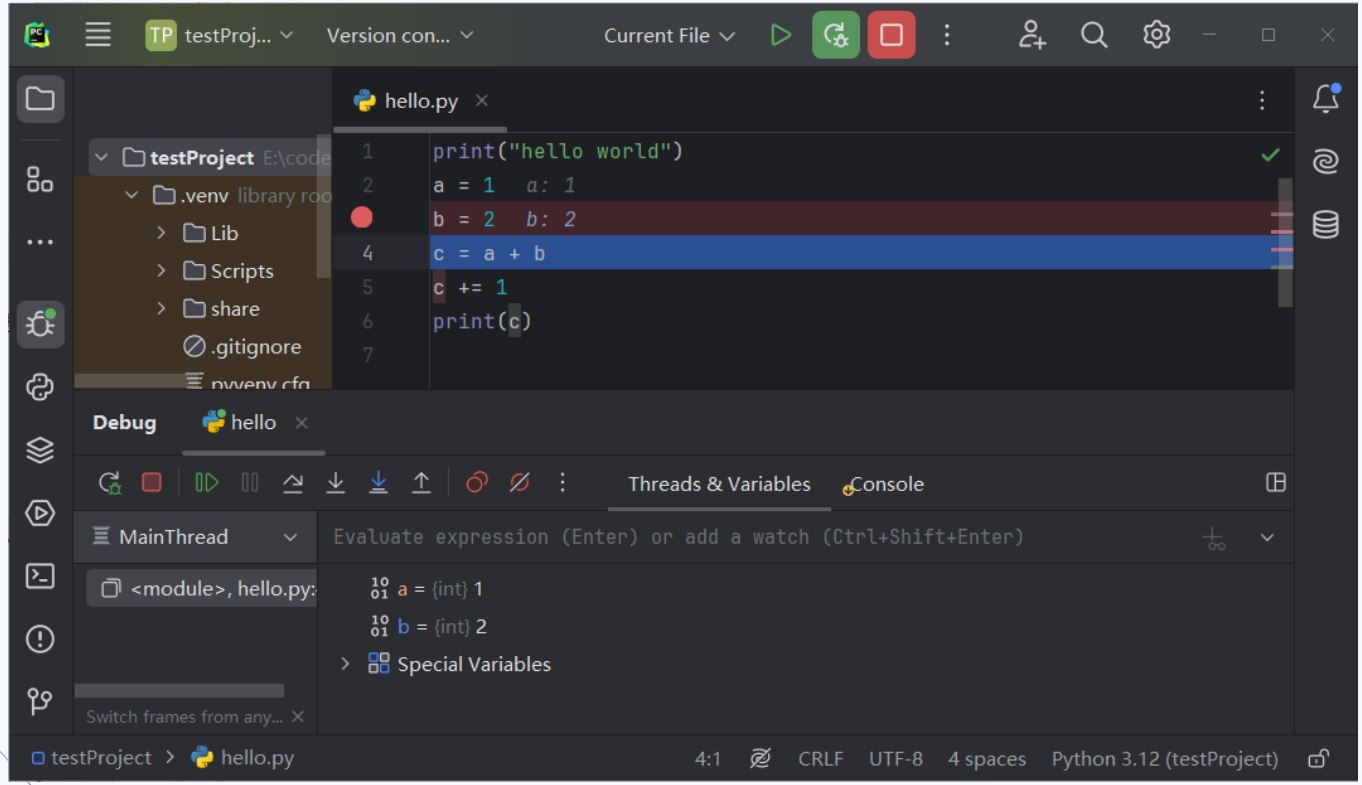
程序开始debug模式后，会自动跳转到第一个断点处暂停

Pycharm Debugging Mode

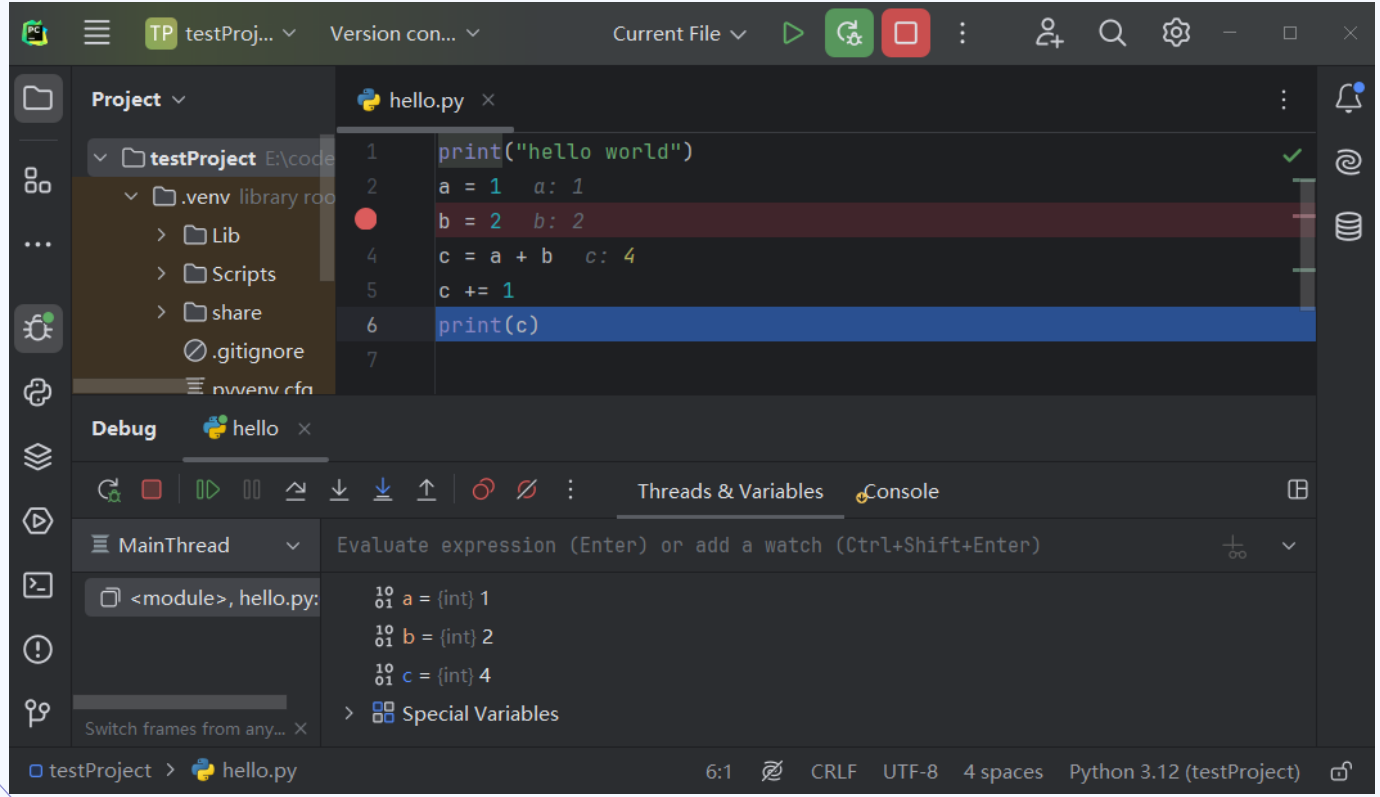


点这个按钮，会跳转到下一步

Pycharm Debugging Mode



Pycharm Debugging Mode

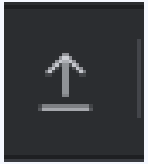


Debugging Mode

到下一个断点（或结束）

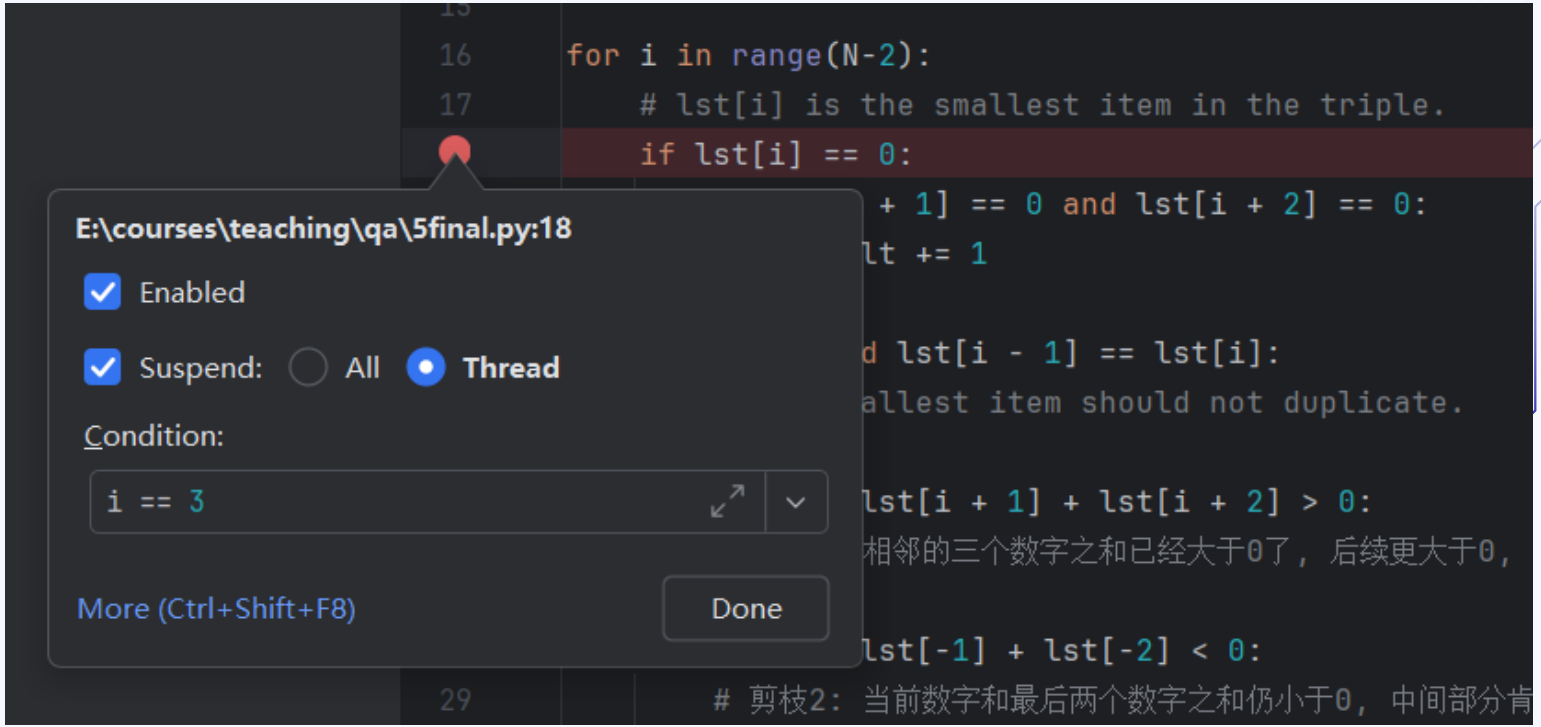
向下执行一步（遇到函数则进入）

向下执行一步

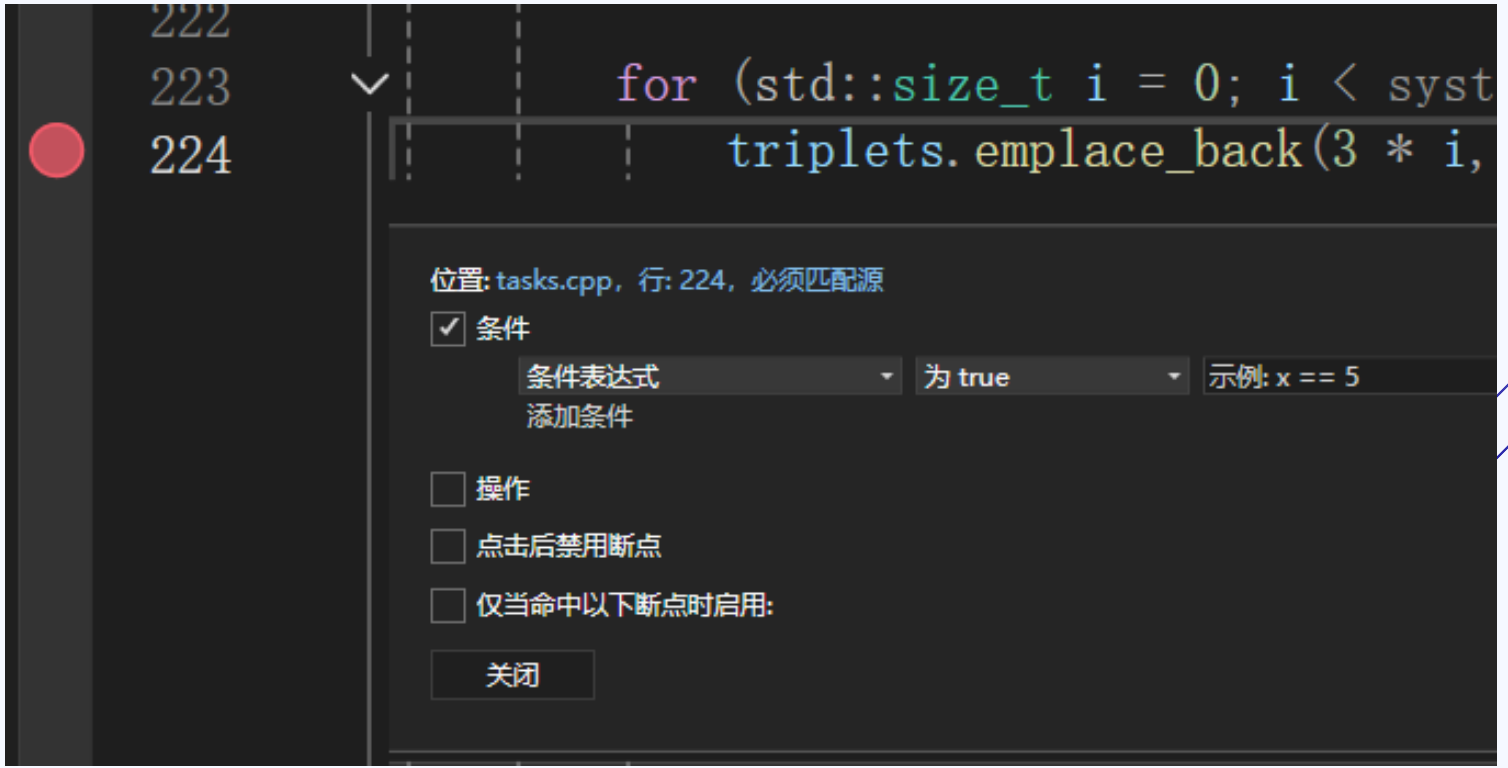


跳出当前函数

Conditional Breakpoint

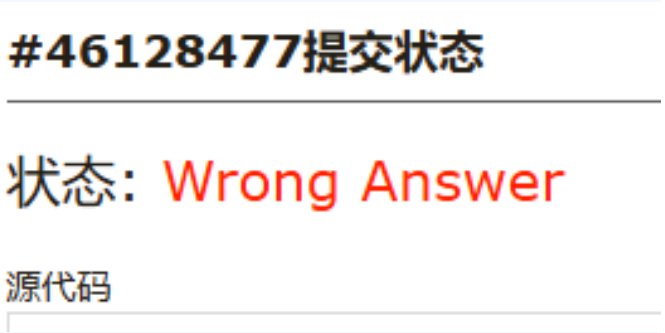


Conditional Breakpoint (in VS)



Several Common Error… Annoying

状态	
Accepted	Wrong Answer
Wrong Answer	Time Out
Wrong Answer	Time Out
Time Out	Time Out
Wrong Answer	Time Out
Empty Output	Compile Error
Wrong Answer	Time Out
Empty Output	Time Out
Empty Output	Wrong Answer
Empty Output	



你的提交记录

#	结果	时间
18	Accepted	2024-09-20
17	Runtime Error	2024-09-20
16	Runtime Error	2024-09-20
15	Wrong Answer	2024-09-20
14	Runtime Error	2024-09-20
13	Compile Error	2024-09-20
12	Runtime Error	2024-09-20
11	Runtime Error	2024-09-20
10	Runtime Error	2024-09-20
9	Runtime Error	2024-09-20
8	Runtime Error	2024-09-20
7	Runtime Error	2024-09-20
6	Runtime Error	2024-09-20
5	Runtime Error	2024-09-20
4	Wrong Answer	2024-09-20
3	Runtime Error	2024-09-20
2	Runtime Error	2024-09-20
1	Runtime Error	2024-09-20

No more hint??

学会**自己编测试数据！！**

再次审题，看看题目中的边界条件有没有考虑到

如果运用了某些优化的算法，**剩余时间充裕**，

可以将**暴力解法**的结果和**优化算法**的结果进行比对

Again：**编程网格的比较逻辑不会忽略输出的行末空格！！**

02

Review Session

从数据建立抽象、从函数建立抽象

The Key Word in Computer Science

抽象

In English: Abstraction(a noun, not an adjective!)

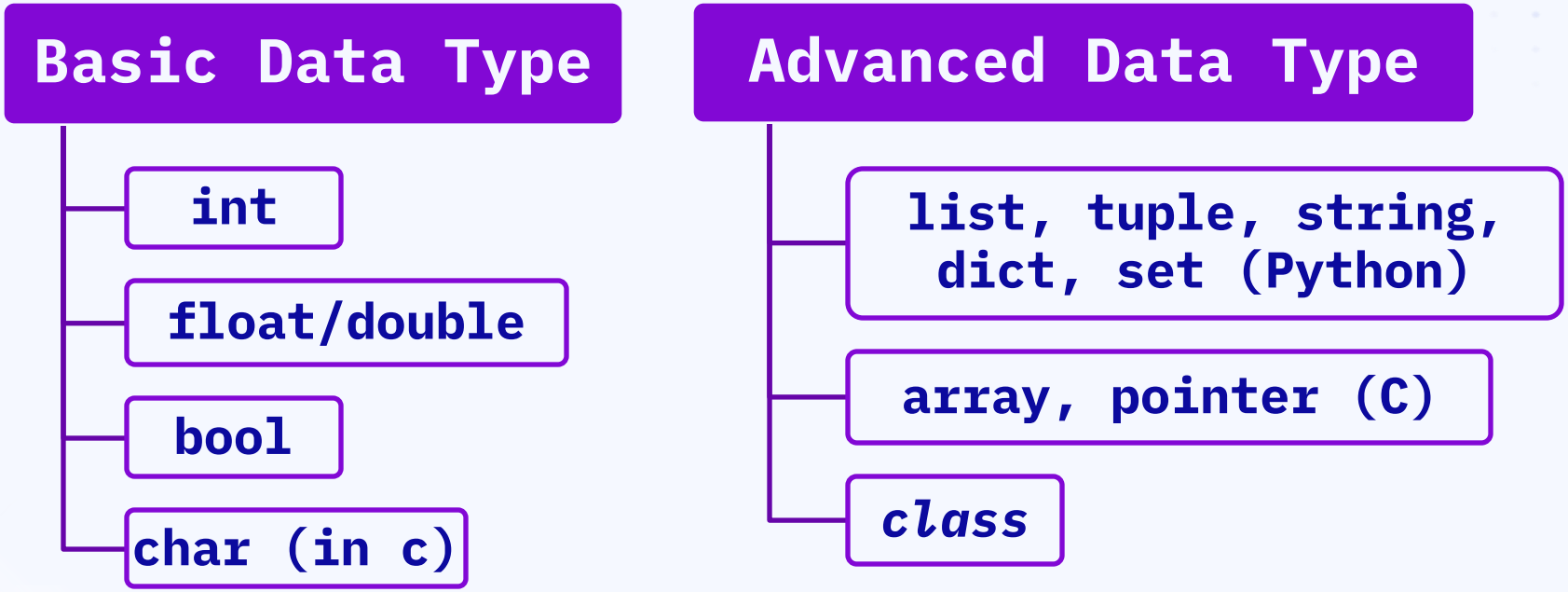
Without Abstraction?

The Code Would Be Like This!

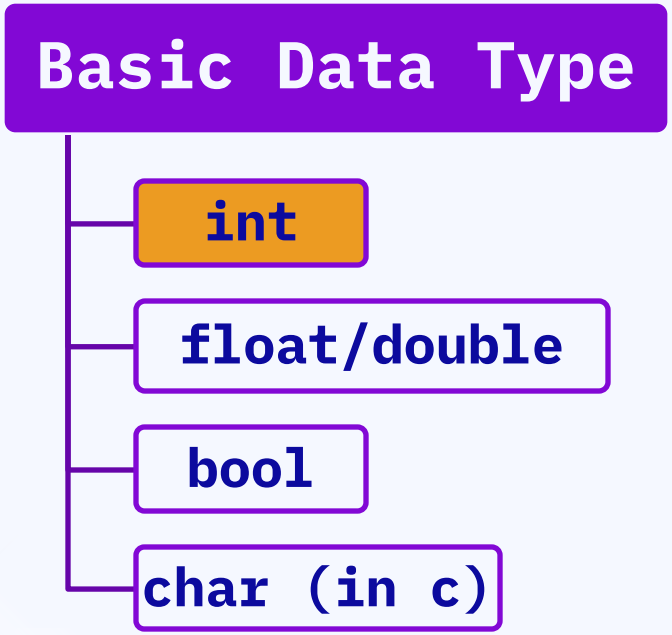
```
466  0000000000002678 <abracadabra>:
467      2678:   f3 0f 1e fa             endbr64
468      267c:   48 81 ec 98 00 00 00    sub    $0x98,%rsp
469      2683:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
470      268a:   00 00
471      268c:   48 89 84 24 88 00 00    mov    %rax,0x88(%rsp)
472      2693:   00
473      2694:   31 c0                   xor    %eax,%eax
474      2696:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
475      269b:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
476      26a0:   4c 8d 44 24 10          lea    0x10(%rsp),%r8
477      26a5:   48 8d 35 ef 2a 00 00    lea    0x2aef(%rip),%rsi        # 519b <_IO_stdin_used+0x19b>
478      26ac:   48 8d 3d d5 70 00 00    lea    0x70d5(%rip),%rdi        # 9788 <input_strings+0x168>
479      26b3:   e8 88 fc ff ff          call   2340 <__isoc99_sscanf@plt>
480      26b8:   83 f8 03                cmp    $0x3,%eax
481      26bb:   74 20                   je     26dd <abracadabra+0x65>
482      26bd:   b8 00 00 00 00          mov    $0x0,%eax
483      26c2:   48 8b 94 24 88 00 00    mov    0x88(%rsp),%rdx
484      26c9:   00
485      26ca:   64 48 2b 14 25 28 00    sub    %fs:0x28,%rdx
486      26d1:   00 00
487      26d3:   75 2b                   jne    2700 <abracadabra+0x88>
488      26d5:   48 81 c4 98 00 00 00    add    $0x98,%rsp
489      26dc:   c3                      ret
```

Data are all Bits… We need to “abstract” them

2.1 Abstraction with data



2.1 Abstraction with data

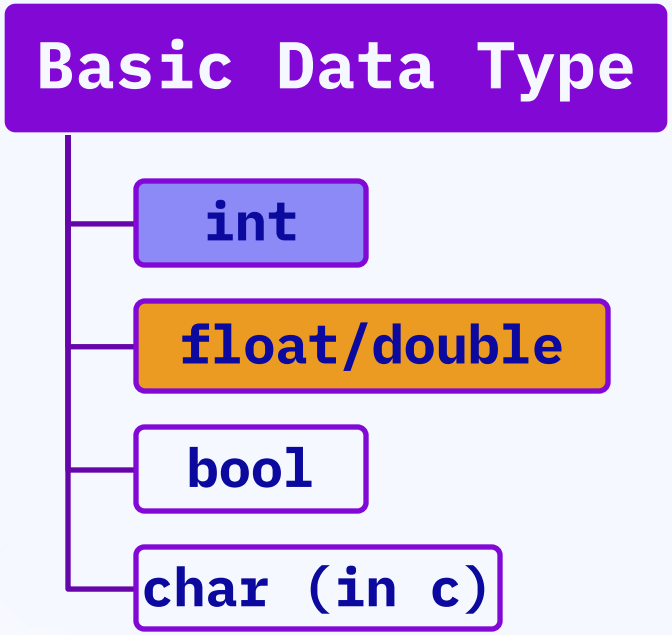


在python中的int不用考虑长度

在c语言中int为32位，有上下限

（这也就是为什么有些题目提示要使用long）

2.1 Abstraction with data



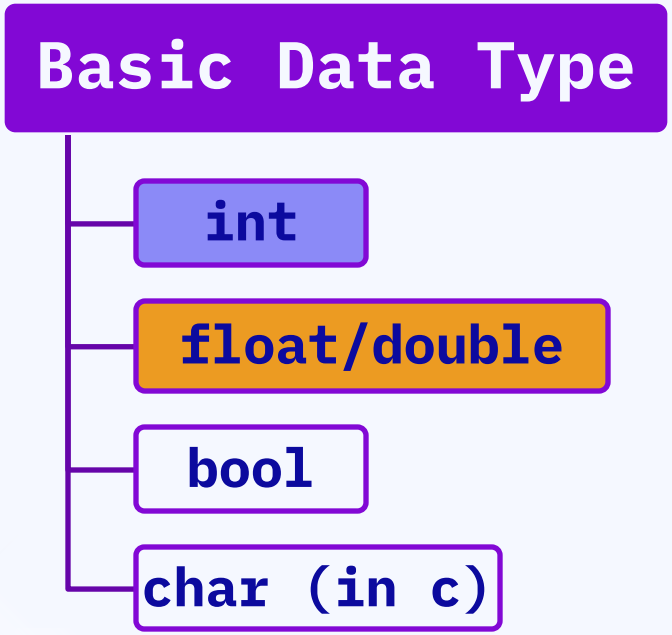
NOTE: C语言中“/”代表除，

“//”代表注释。

Python中“/”代表除，

“//”代表整除！

2.1 Abstraction with data



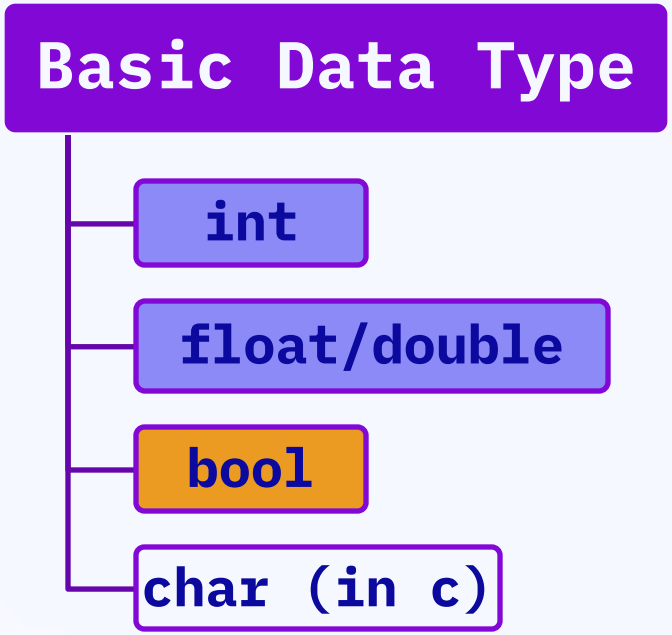
Test: 3 / 2 * 2 = ?

(The result may be surprising!)

2.1 Abstraction with data

```
123     Returns:
124         None
125     """
126     ## 请于此填写你的代码
        epoch_num = X.shape[0] / batch   epoch_num: 600.0
        for i in range(epoch_num):
129         training_x = X[i * batch: (i + 1) * batch]
130         training_y = y[i * batch: (i + 1) * batch]
131         x_loss = SoftmaxLoss()( *args: forward(training_x, weights), t
132         x_loss.backward()
133         for w in weights:
```

2.1 Abstraction with data



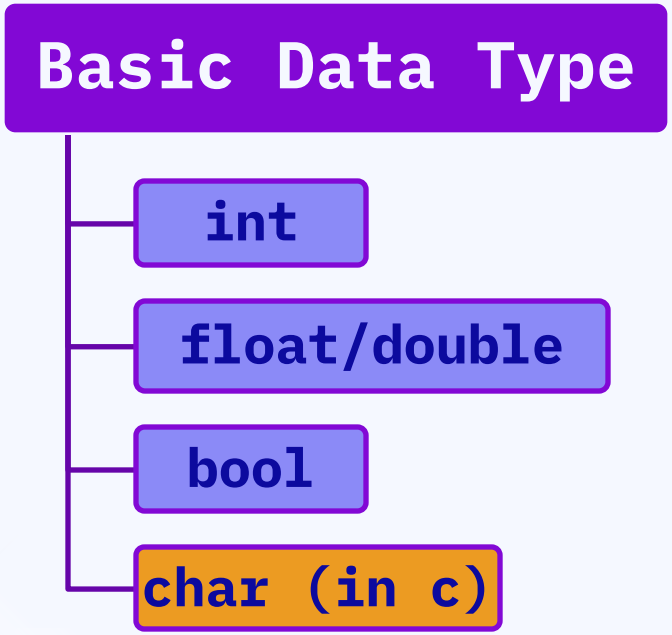
and和or的“短路运算”

expr1 ||(or) expr2

expr1 &&(and) expr2

(用途：检查空指针)

2.1 Abstraction with data



Only in C!

C中的字符串：char数组

char可以转int（对应ASCII码）

可以做加减法

2.1 Abstraction with data

Basic Data Type

- int
- float/double
- bool
- char (in c)

```
>>> for i in "123":
...         print(type(i))
...
<class 'str'>
<class 'str'>
<class 'str'>
>>>
```

ch 码）

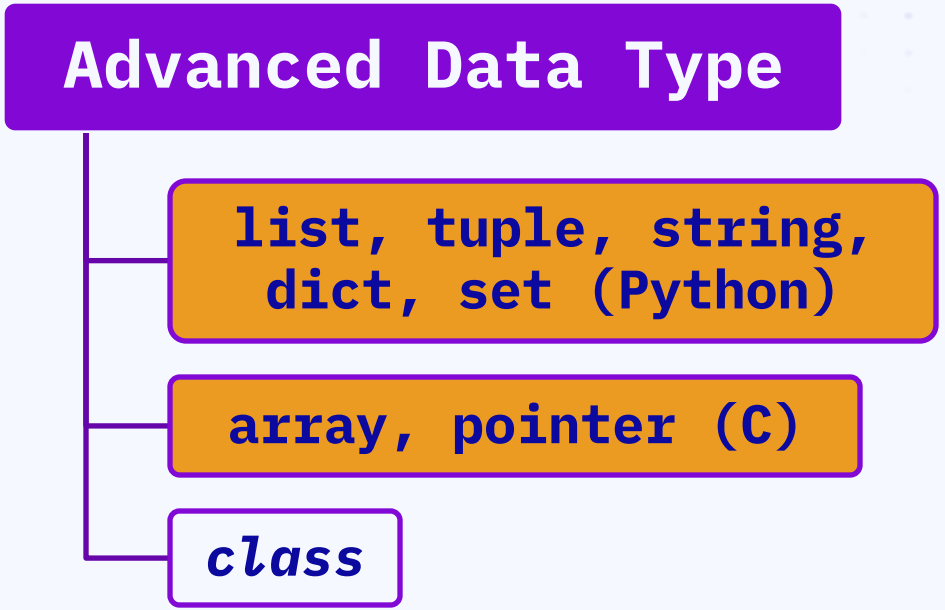
2.1 Abstraction with data

Python中的“序列”：

(list, tuple, string, range…)

C中最基本的“序列”：

数组



2.1 Abstraction with data

指针**pointer:**

指向一个变量的内存地址

Advanced Data Type

- **list, tuple, string, dict, set (Python)**
- **array, pointer (C)**
- ***class***

2.1 Abstraction with data

指

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

C++ (C++20 + GNU extensions)
known limitations

```
1  #include <stdio.h>
2  int a = 3;
3
4  int main() {
5      int* k = &a;
6      printf("K: 0x%x\n", k);
7      *k = 4;
8      printf("a: %d\n", a);
9      return 0;
10 }
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 6

Print output (drag lower right corner to resize)

Stack Heap

Global variables

a: 0x601040, int, 3

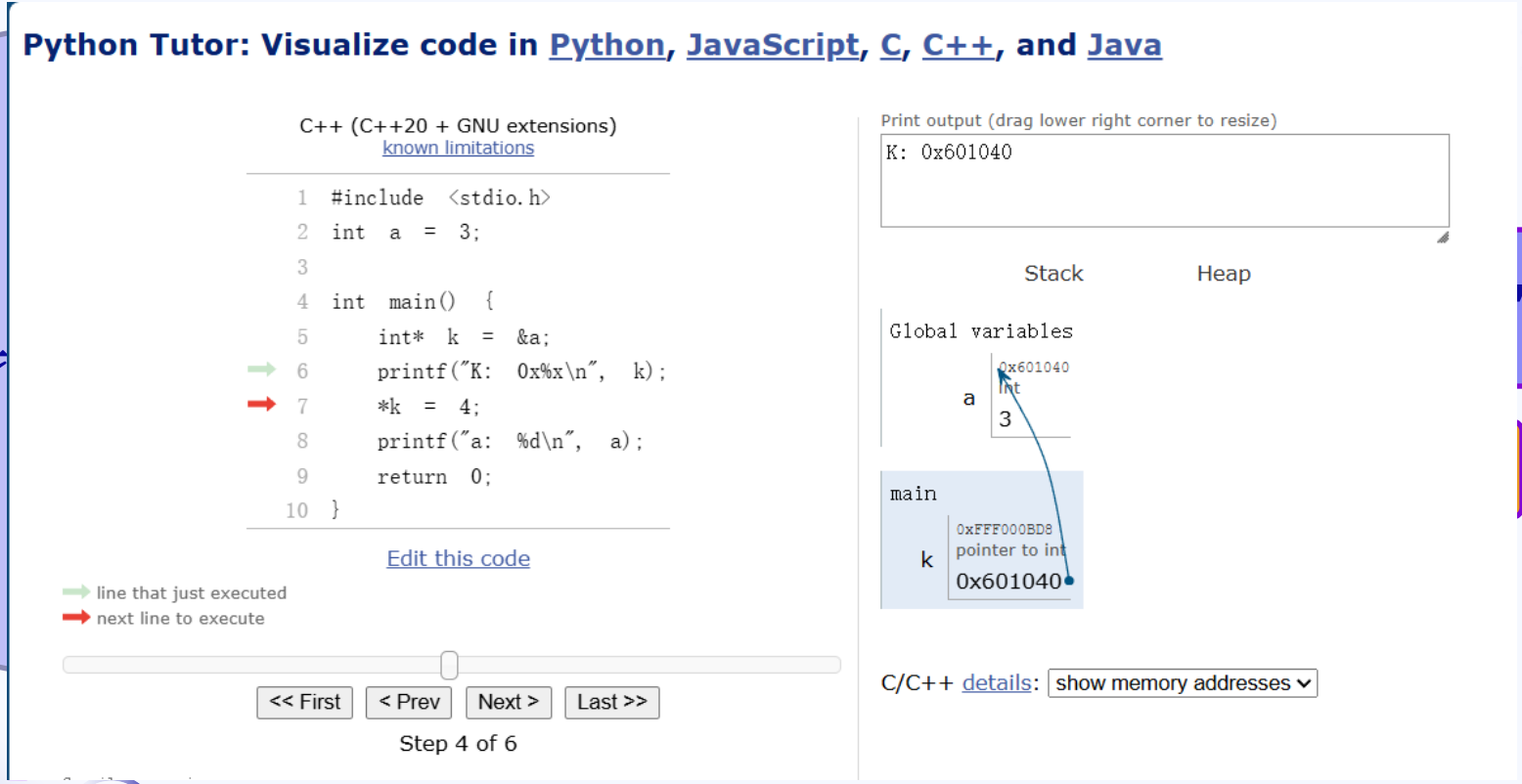
main

k: 0xFFF000BD8, pointer to int, ?

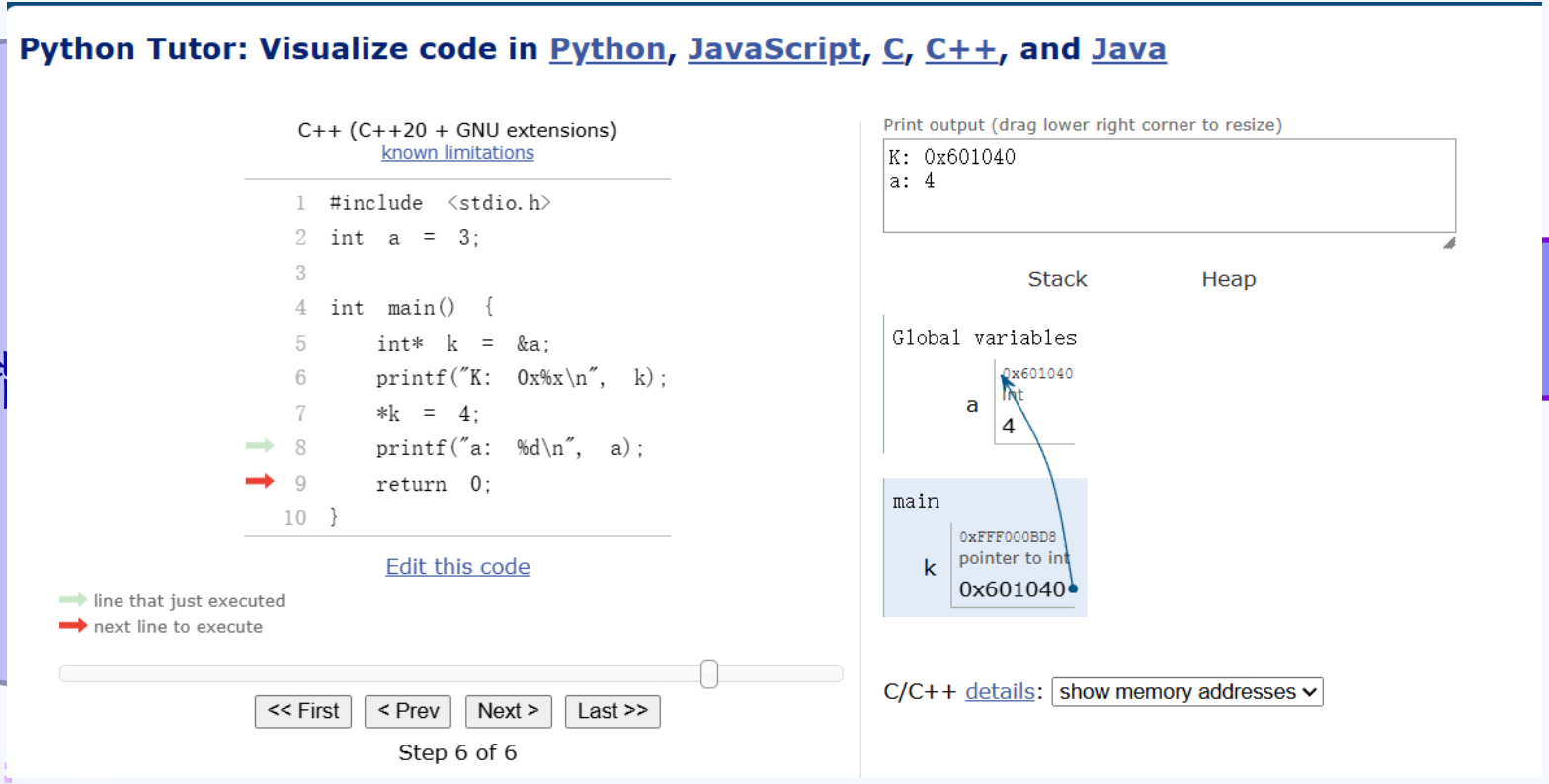
Note: ? refers to an uninitialized value

C/C++ details: show memory addresses

2.1 Abstraction with data



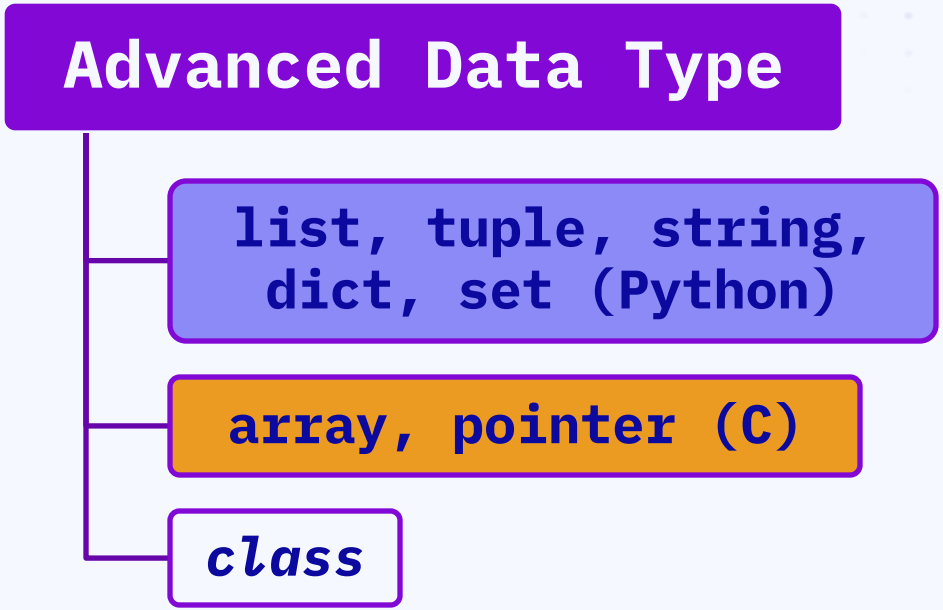
2.1 Abstraction with data



2.1 Abstraction with data

指针pointer:

Python中，“指针”是被隐藏起来的！



2.1 Abstraction with data

2.1 Abstraction with data

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
known limitations

```
1  a = [1, 2, 3]
2  b = a
3  b[1] = "Strange value."
4  print(a[1])
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 4

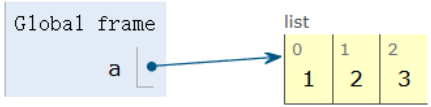
NEW: follow our YouTube, TikTok, and Instagram for free tutorials

Get AI Help

Move and hide objects

Print output (drag lower right corner to resize)

Frames Objects



2.1 Abstraction with data

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
known limitations

```
1  a = [1, 2, 3]
2  b = a
3  b[1] = "Strange value."
4  print(a[1])
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 3 of 4

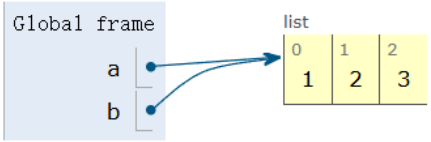
NEW: follow our YouTube, TikTok, and Instagram for free tutorials

Get AI Help

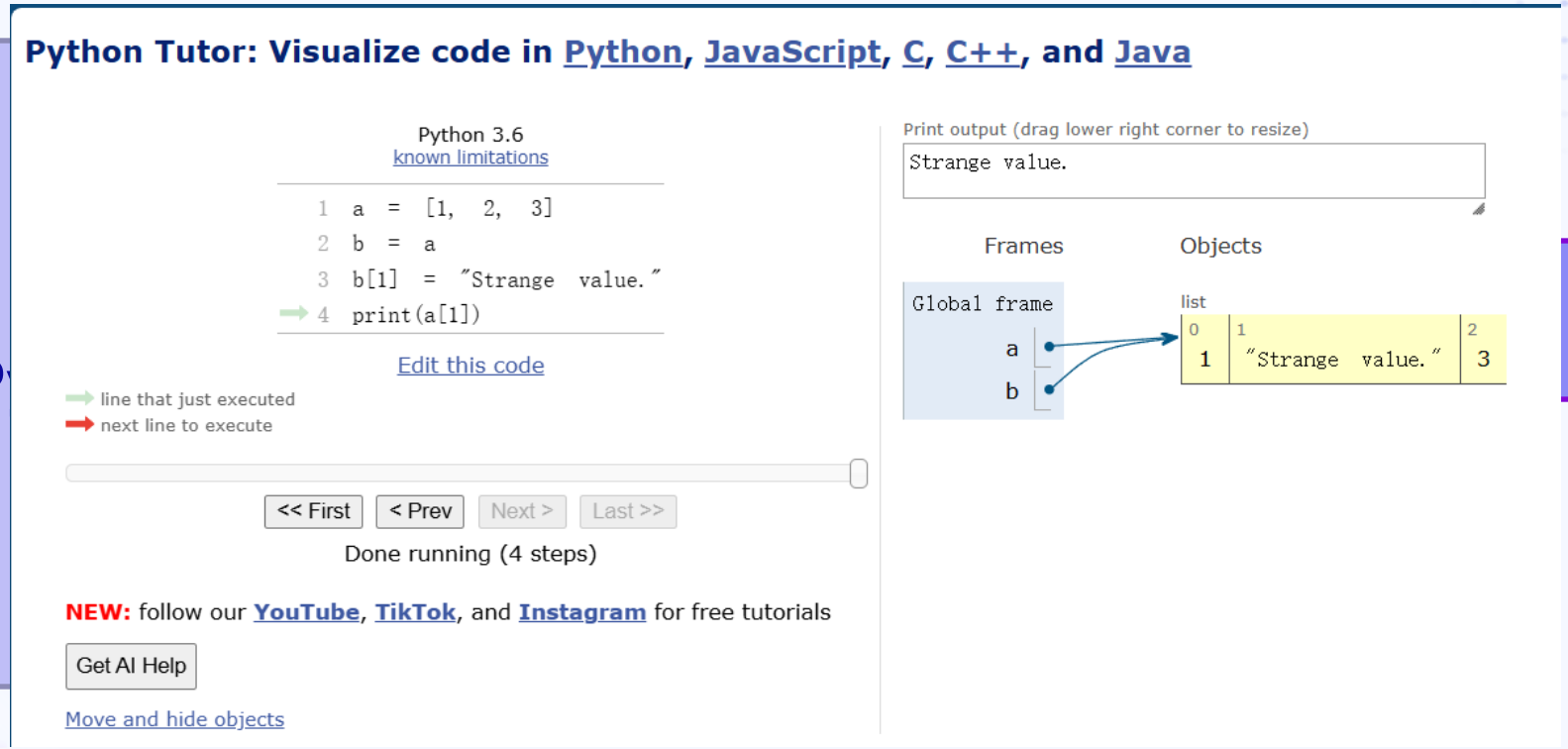
Move and hide objects

Print output (drag lower right corner to resize)

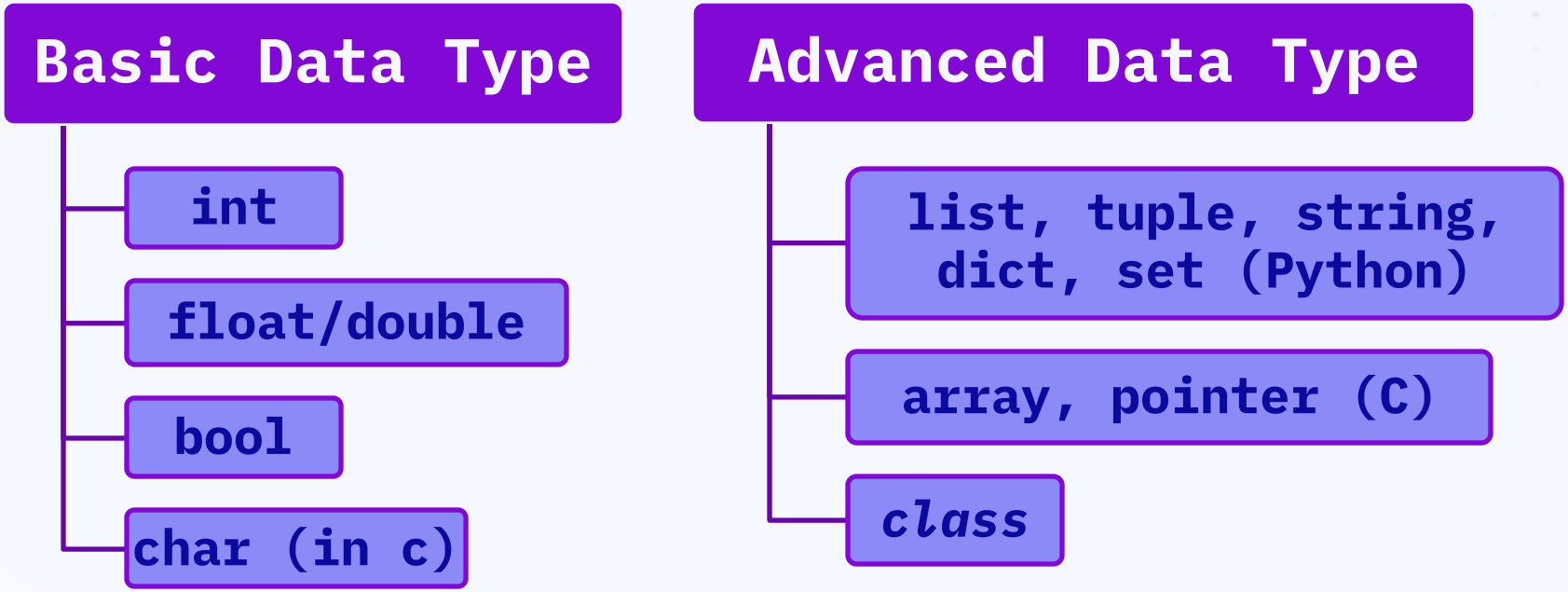
Frames Objects



2.1 Abstraction with data

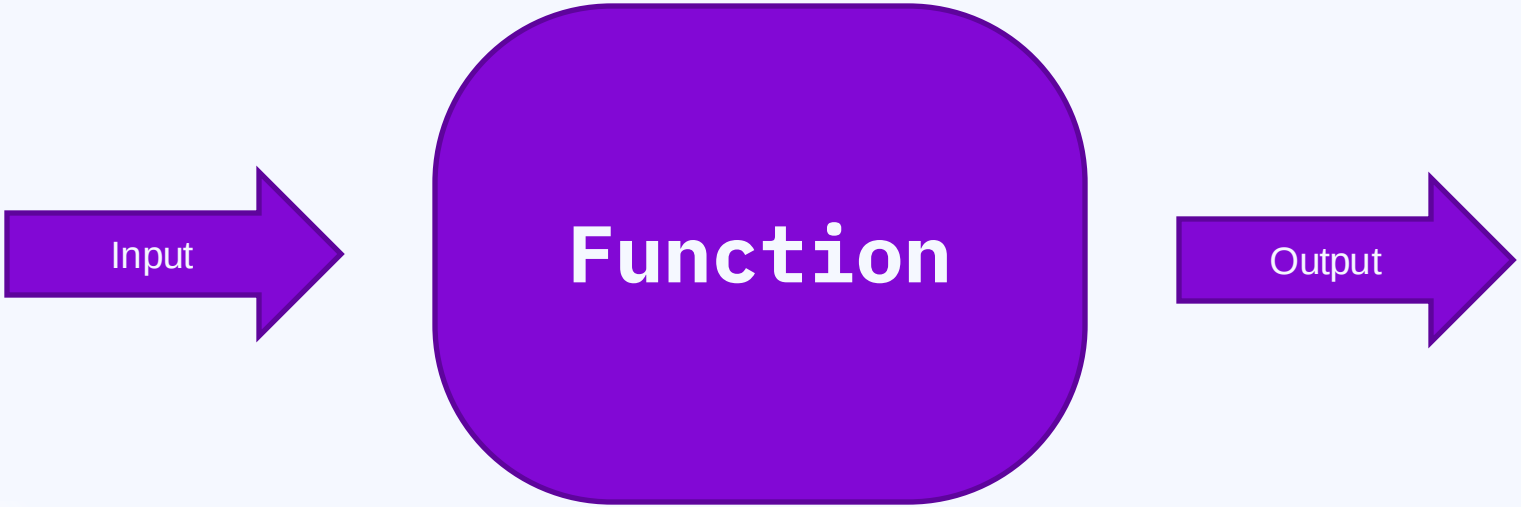


2.1 Abstraction with data



2.2 Abstraction with function

函数：“实现特定功能的代码块”



2.2 Abstraction with function

函数：“实现特定功能的代码块”

例：max(a, b)

min(t)

print(...) (python) / printf(...) (c)

fibonacci(i) // 自定义，返回第n项fibonacci数列的值

2.2 Abstraction with function

函数：“实现特定功能的代码块”

Why abstraction?

当调用一个函数时，不用关心函数内部的实现机制，只需要认为“它是对的”

（比如……你可能永远也不会关心print函数的底层是什么样）

2.2 Abstraction with function

函数：“实现特定功能的代码块”

Why abstraction?

当定义一个函数时，把它和外界其他函数的代码分隔开。

想象成独立的代码区块，接收输入，返回输出。

2.2 Abstraction with function

递归："定义函数时，自己调用自己"

A really important idea

in computer science!

1. **Base case**
2. **Recursive Leap of Faith**

2.2 Abstraction with function

递归– 例1: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

Base case:

fibonacci(0) = 0

fibonacci(1) = 1

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例1: 斐波那契数列(again.)

Recursive leap of faith:

怎么计算fibonacci(n)?

 # 假设：你已经会计算n-1

以下的情况了！！

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例1: 斐波那契数列(again.)

Recursive leap of faith:

怎么计算fibonacci(n)?

f(n) = f(n-1) + f(n-2)

Always assume f(n-1) and f(n-2) is correct!!

2.2 Abstraction with function

递归– 例1: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

```
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

2.2 Abstraction with function

递归– 例1: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

```
int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例2: 整数划分问题

将正整数n 表示成一系列正整数之和，n=n1+n2+…+nk, 其中n1>=n2>=…>=nk>=1，k>=1。正整数n 的这种表示称为正整数n 的划分。正整数n 的不同的划分个数称为正整数n 的划分数。

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

5

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

4 1

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

3 2

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

3 1 1

1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

2 2 1

1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

2 1 1

1

1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

7种情况。怎么求解情况的种数？

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例2: 整数划分问题(again.)

7种情况。怎么求解情况的种数？

尝试: def partition(n) …

if (n == 1)? 1种情况。

if partition(n – 1) is known…

2.2 Abstraction with function

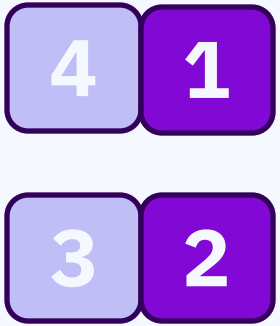
递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

5

2.2 Abstraction with function

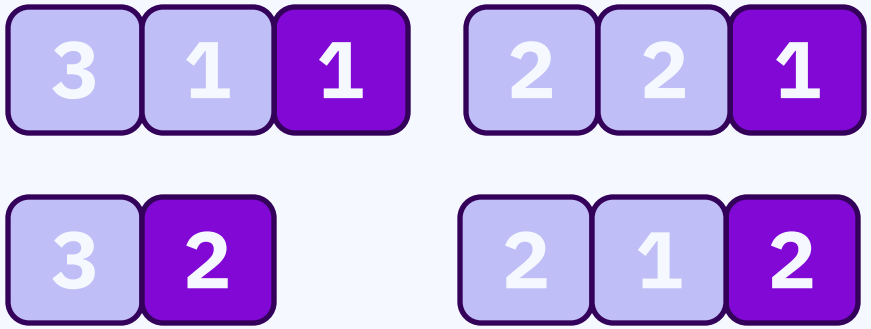
递归– 例2: 整数划分问题(again.)



1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)



Wait... 重复了！！

1. Base case
2. Recursive Leap of Faith

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例2: 整数划分问题(again.)

7种情况。怎么求解情况的种数？

尝试: def partition(n) …

if (n == 1)? 1种情况。

if partition(n – 1) is known…

不太够。还需要更强的限制。

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

5 = 3 + 2，已经划分出了一个2，

不希望接下来划分3的时候划分出比2小的元素。

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. **Base case**
2. **Recursive Leap of Faith**

尝试: def partition(n, min_constraint) ...?

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

尝试: def partition(n, min_constraint) ...?

`5` → `(5, 1)`

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

递归– 例2: 整数划分问题(again.)

怎么求？

partition(n, min_constraint)

`5` → `(5, 1)`

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

1. Base case

partition(n, min_constraint)

`(1, 1)` → `1`

`(n, n)` → `1`

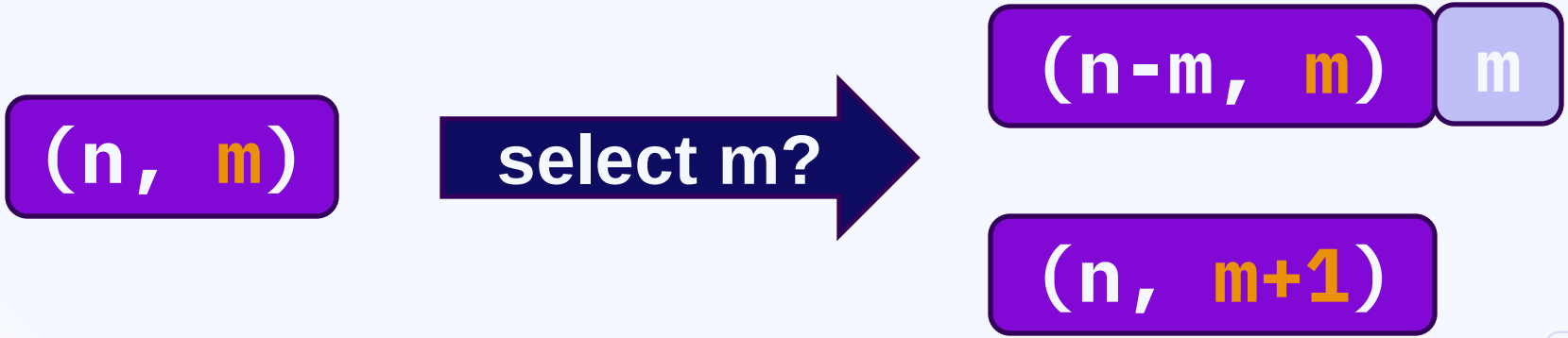
`(n, m(>n))` → `0`

2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

1. Base case
2. Recursive Leap of Faith

2. Recursive Leap of Faith partition(n, min_constraint)

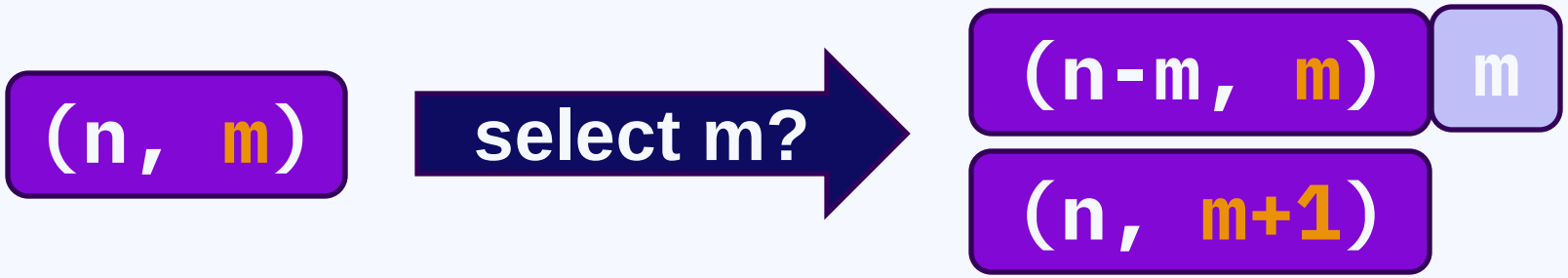


2.2 Abstraction with function

递归– 例2: 整数划分问题(again.)

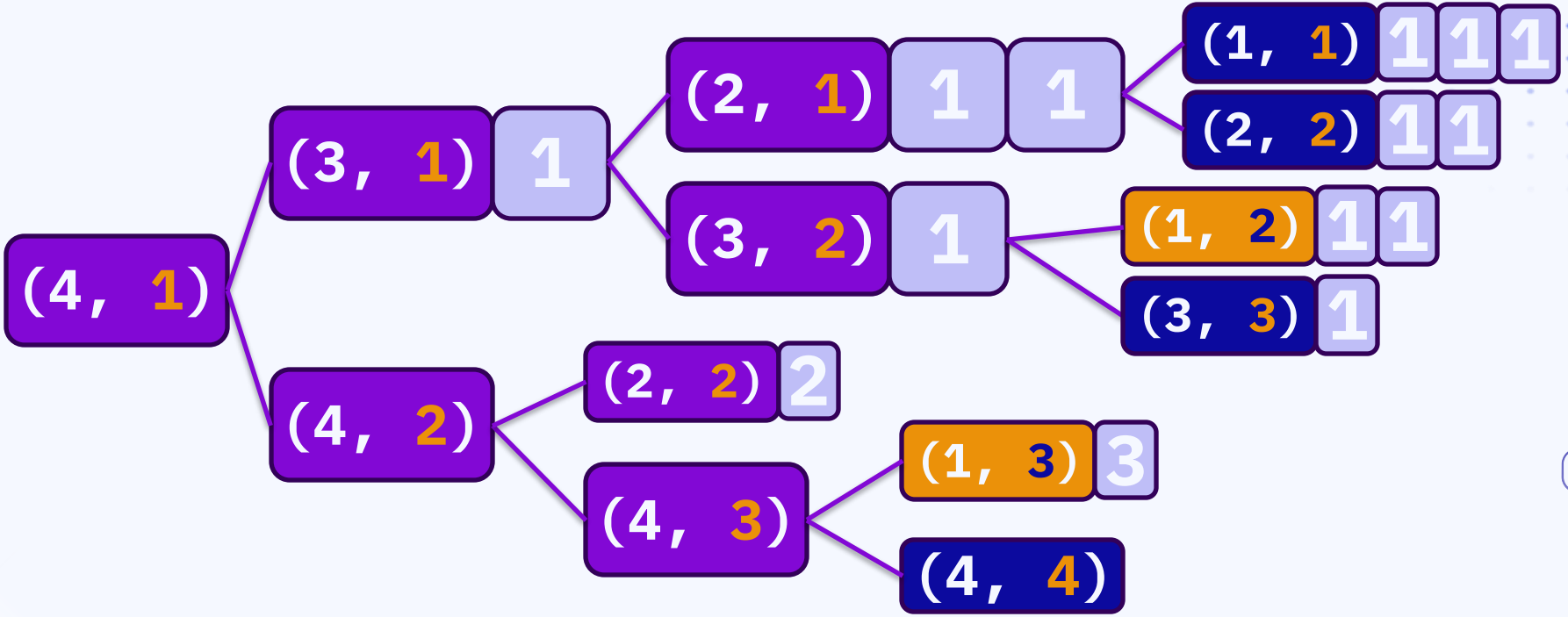
1. Base case
2. Recursive Leap of Faith

2. Recursive Leap of Faith partition(n, min_constraint)



partition(n, m) = partition(n-m, m) + partition(n, m+1) !!!!!

2.2 Abstraction with function



partition(n, m) = partition(n-m, m) + partition(n, m+1) !!!!!

2.2 Abstraction with function

1. Base case
2. Recursive Leap of Faith

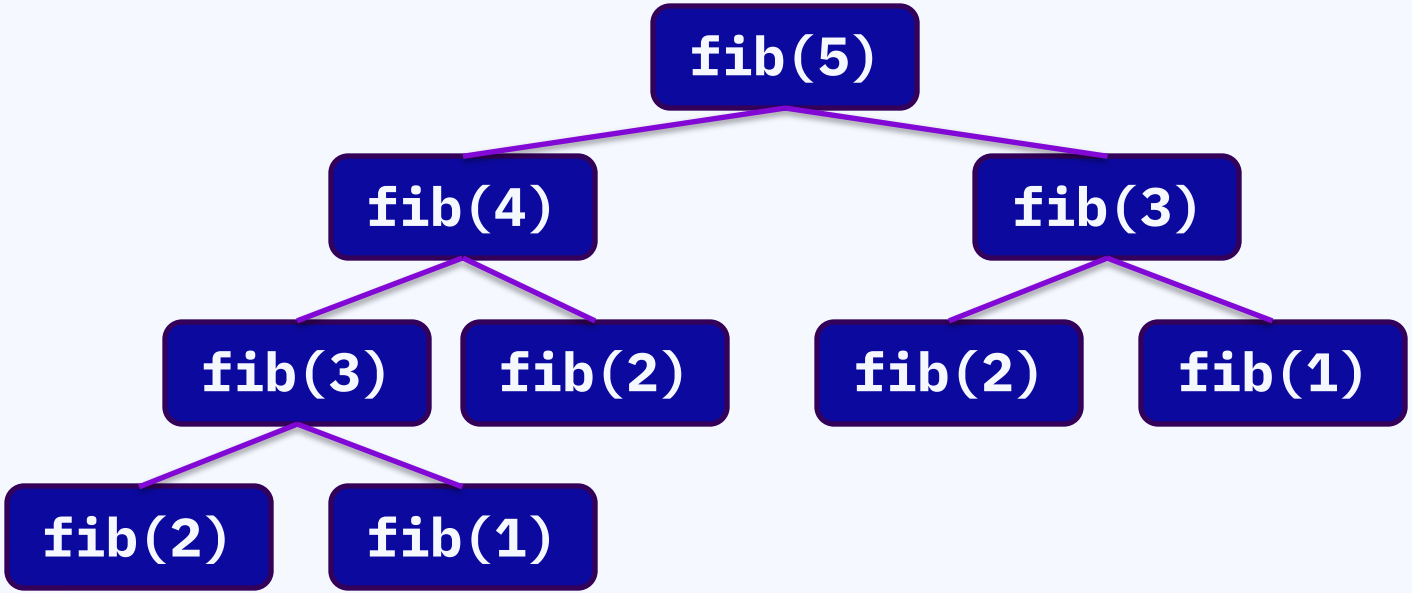
递归– 例2: 整数划分问题(again.)

```
int partition(int n, int m) {
        if (n == m) return 1;
        if (m > n) return 0;
        return partition(n-m, m) + partition(n, m+1);
}
```

求n的划分数，就是求partition(n, 1)

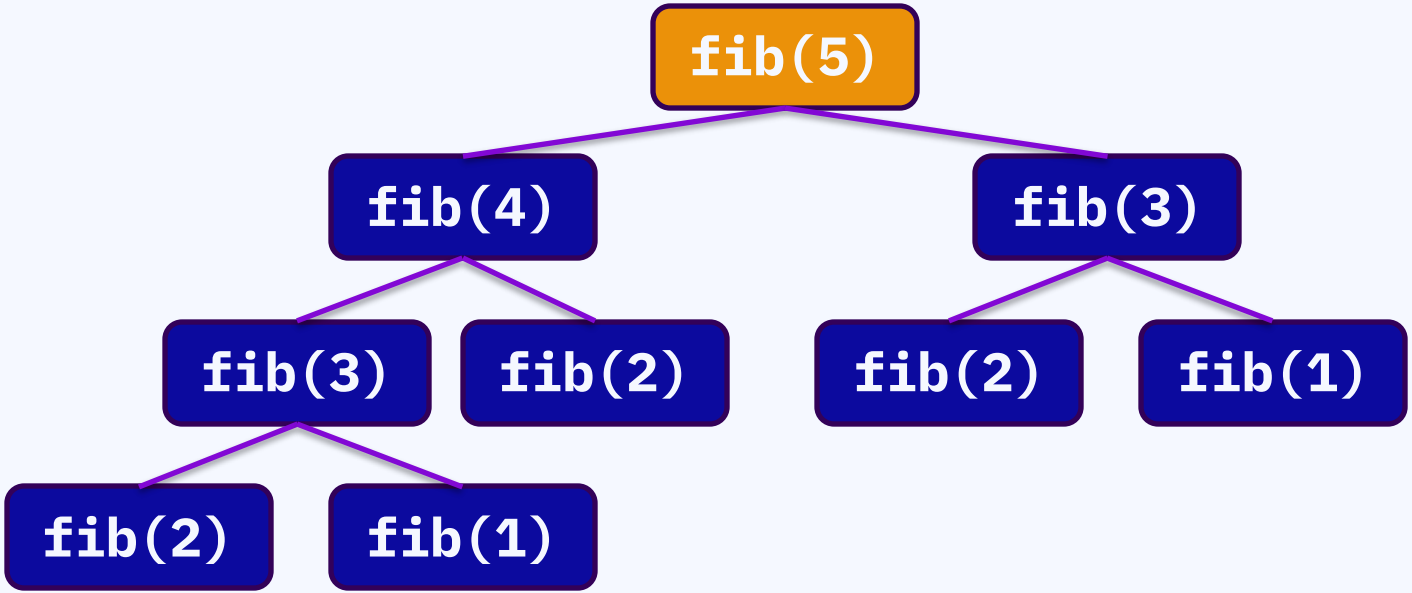
2.2 Abstraction with function

递归– 问题：重复计算！！



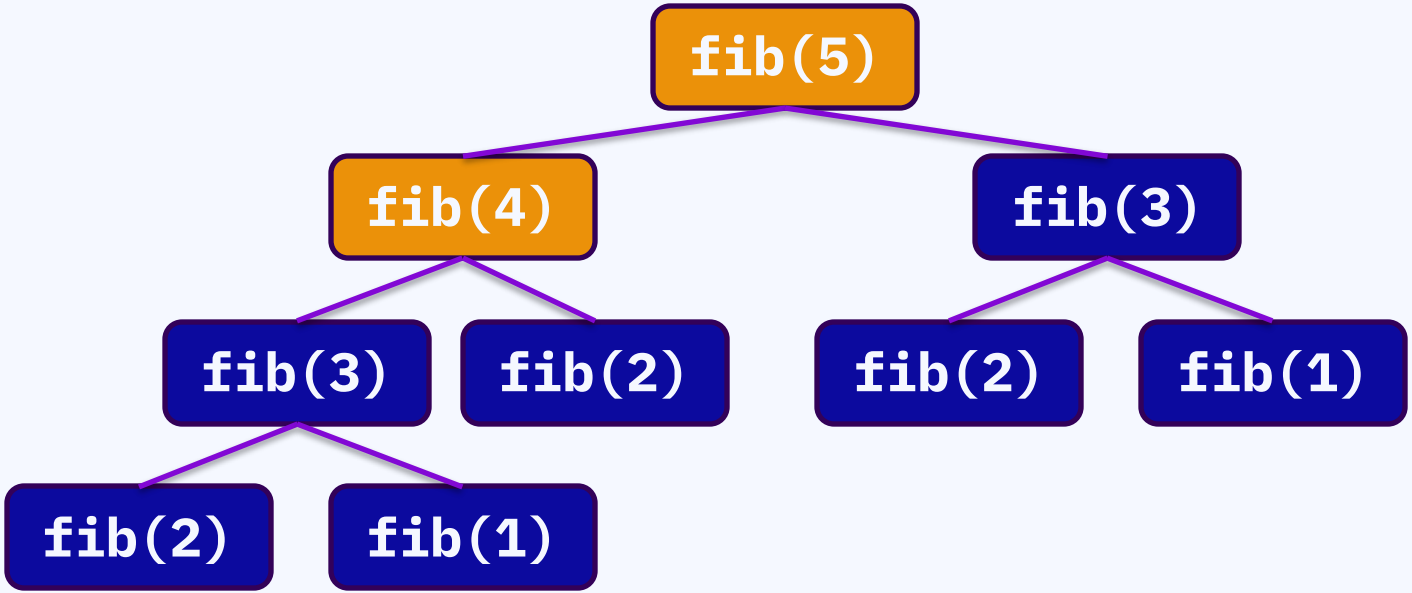
2.2 Abstraction with function

递归– 问题：重复计算！！



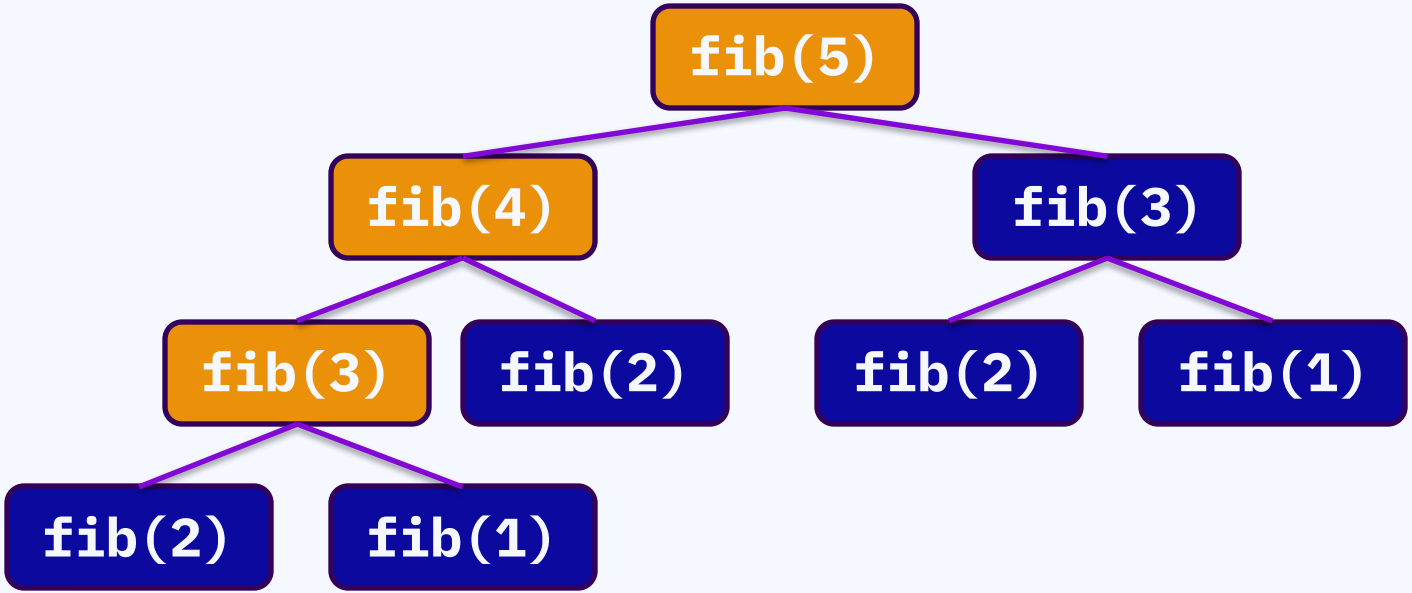
2.2 Abstraction with function

递归– 问题：重复计算！！



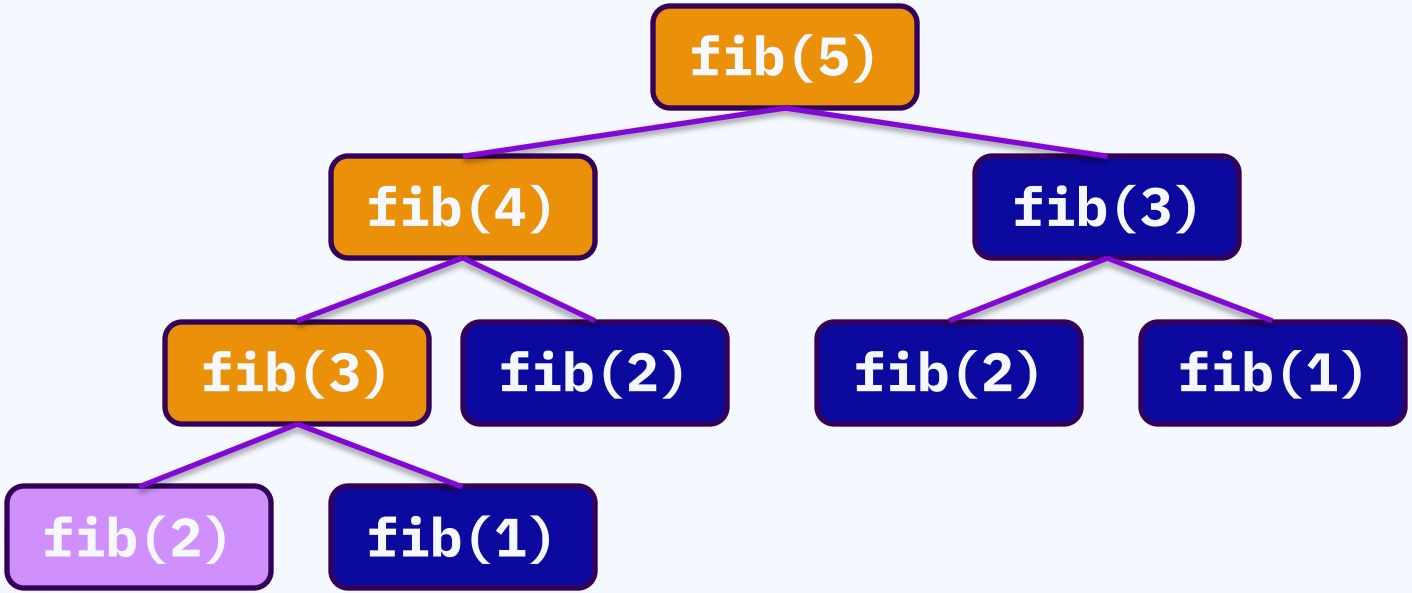
2.2 Abstraction with function

递归– 问题：重复计算！！



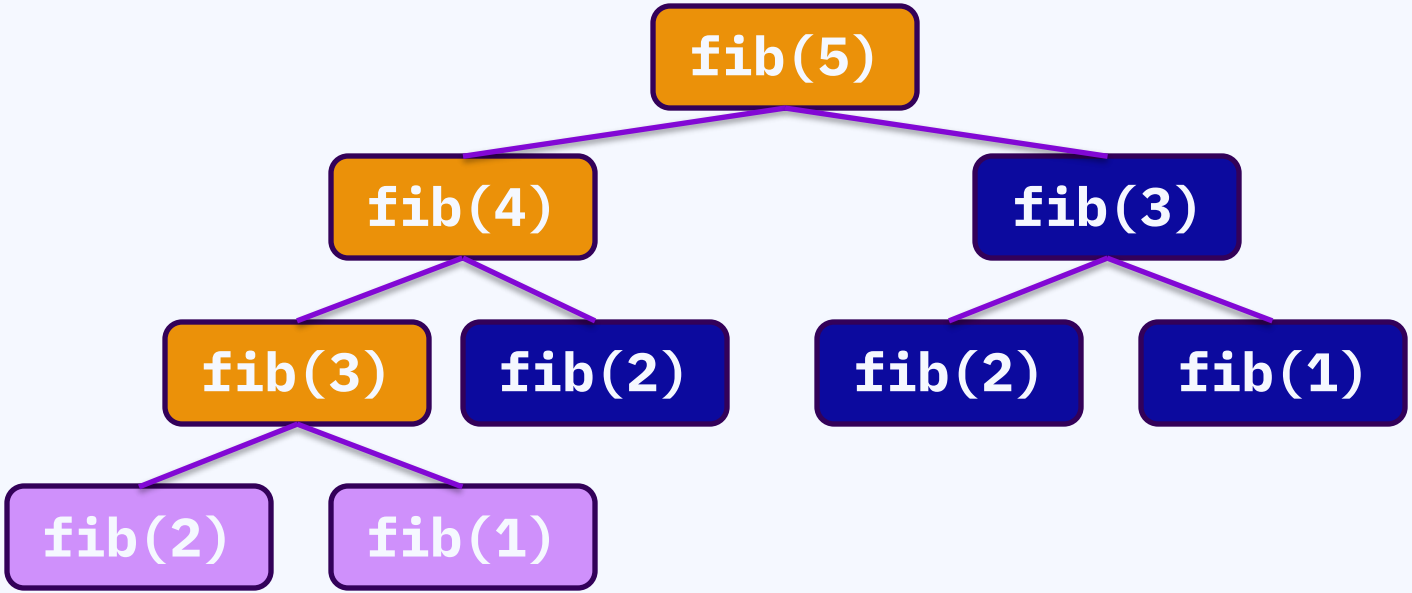
2.2 Abstraction with function

递归– 问题：重复计算！！



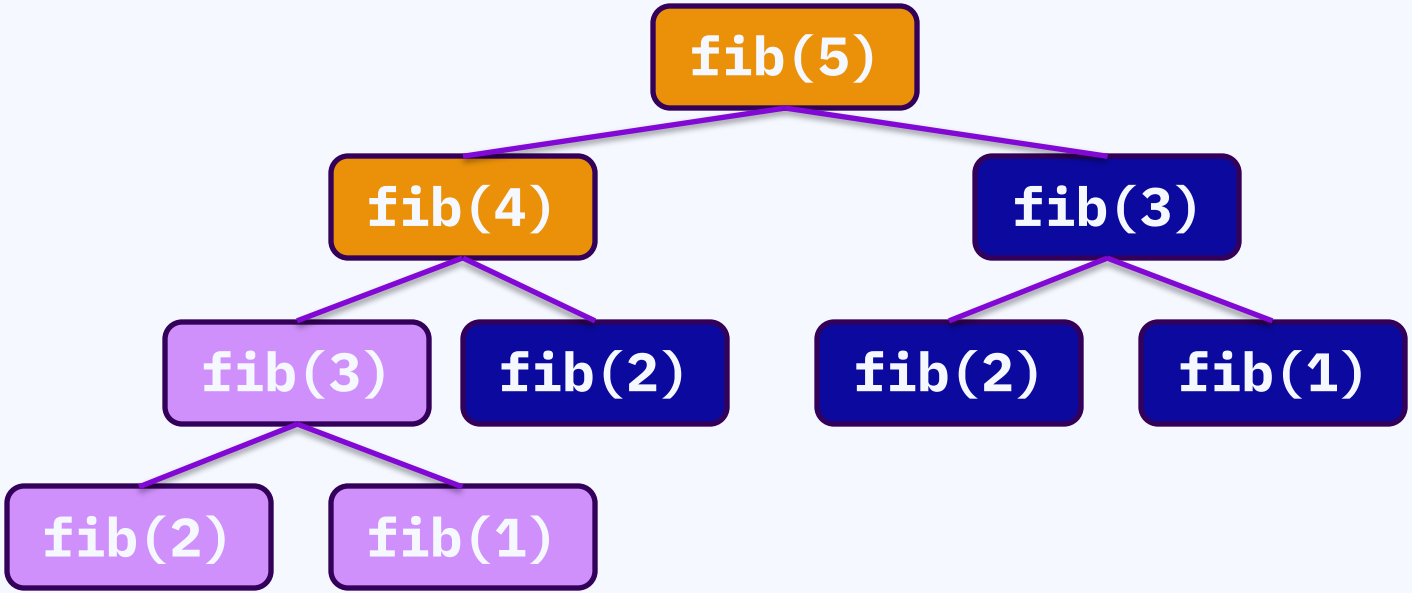
2.2 Abstraction with function

递归– 问题：重复计算！！



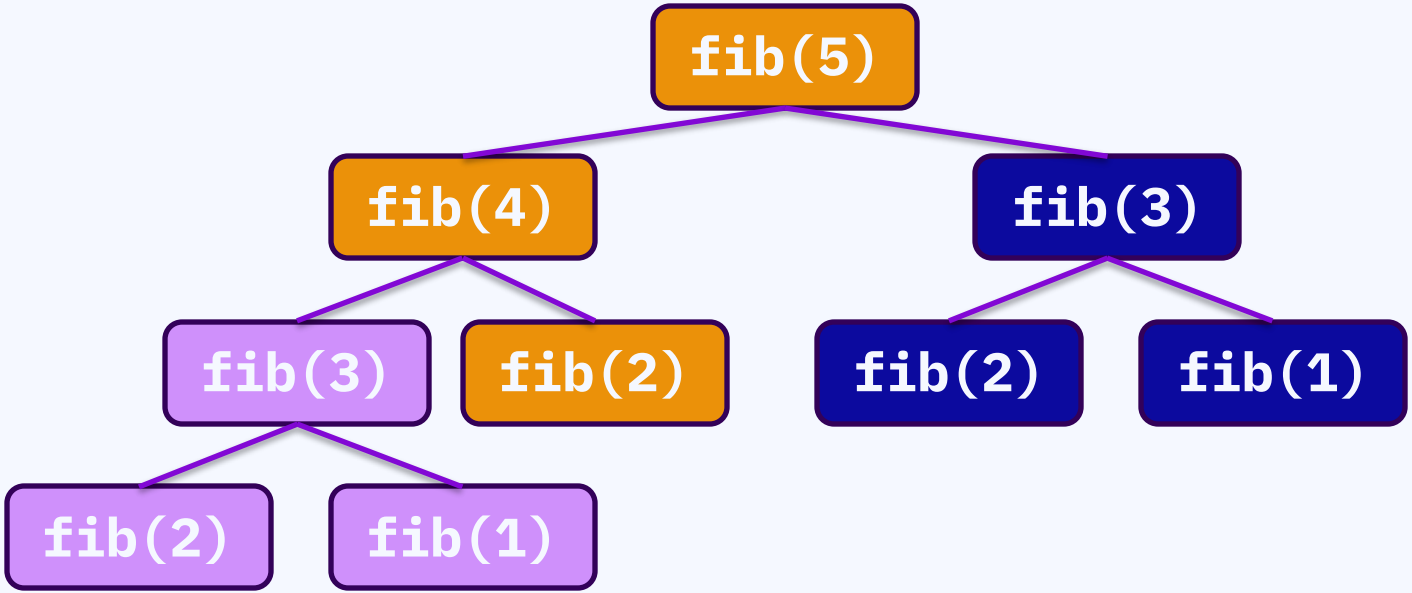
2.2 Abstraction with function

递归– 问题：重复计算！！



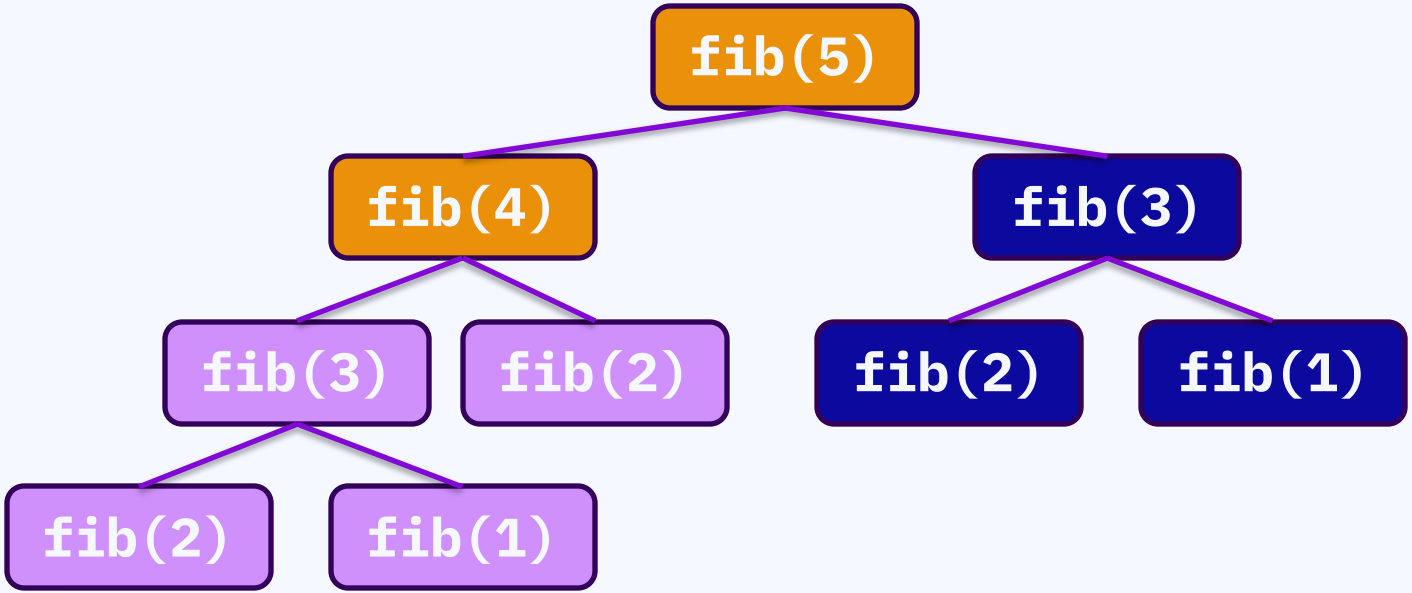
2.2 Abstraction with function

递归– 问题：重复计算！！



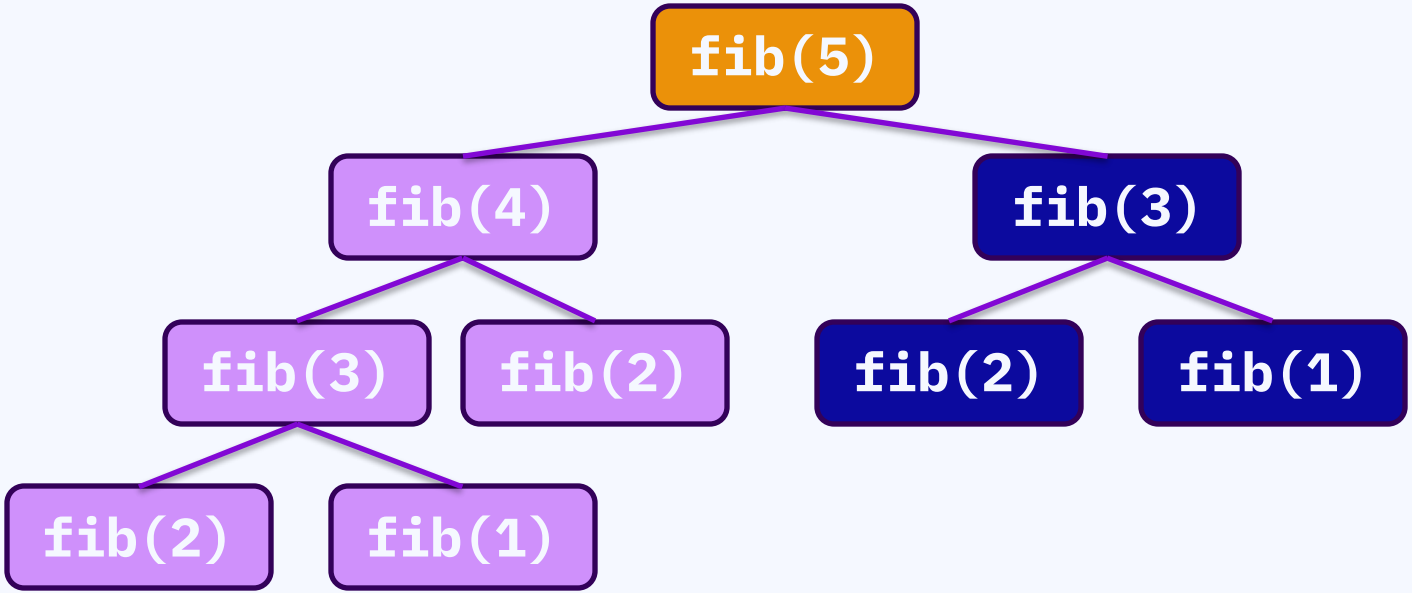
2.2 Abstraction with function

递归– 问题：重复计算！！



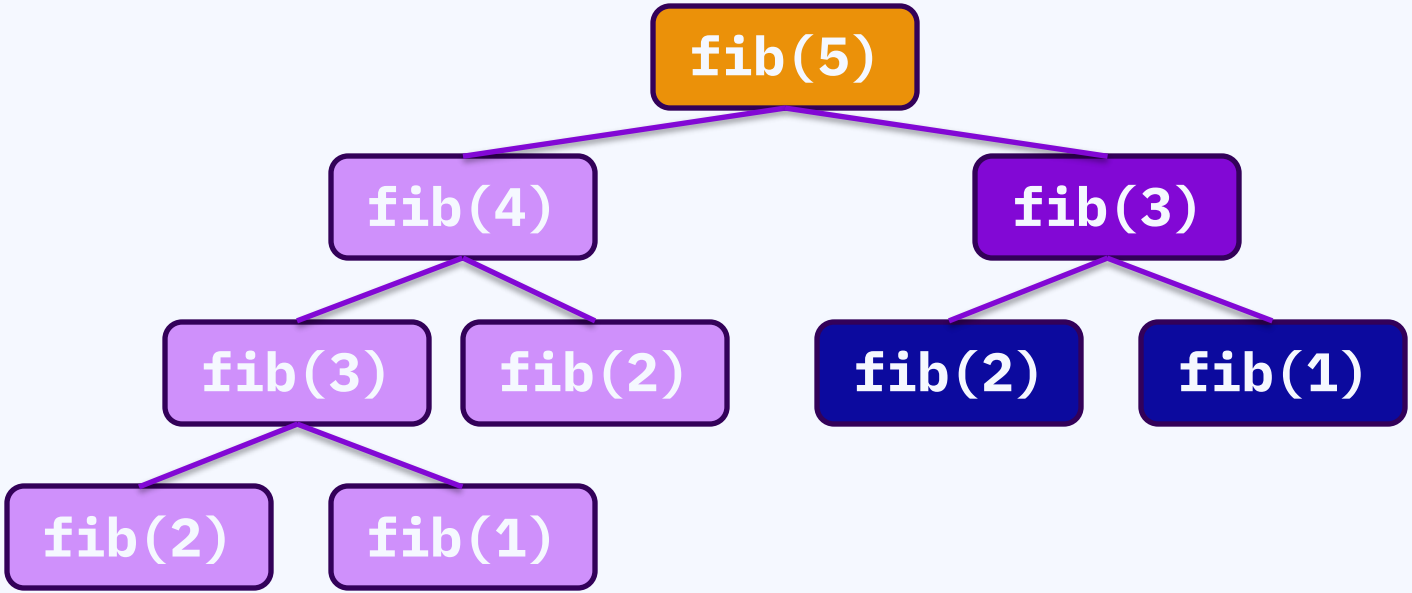
2.2 Abstraction with function

递归– 问题：重复计算！！



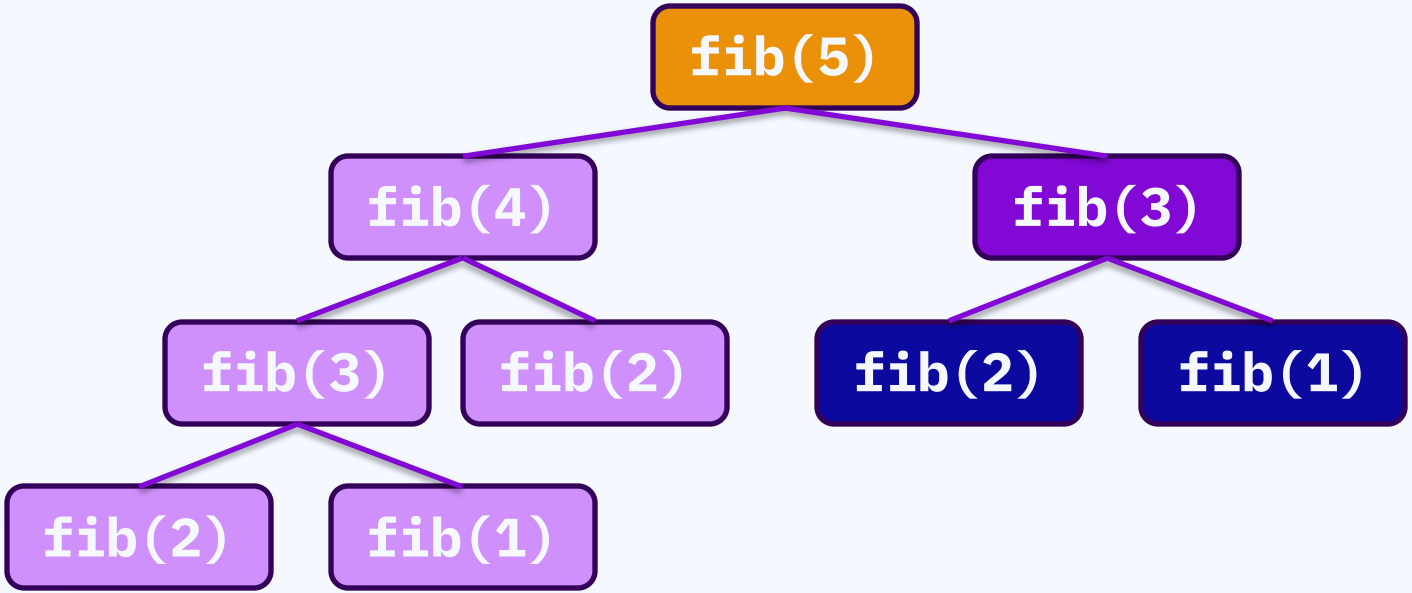
2.2 Abstraction with function

递归– 问题：重复计算！！



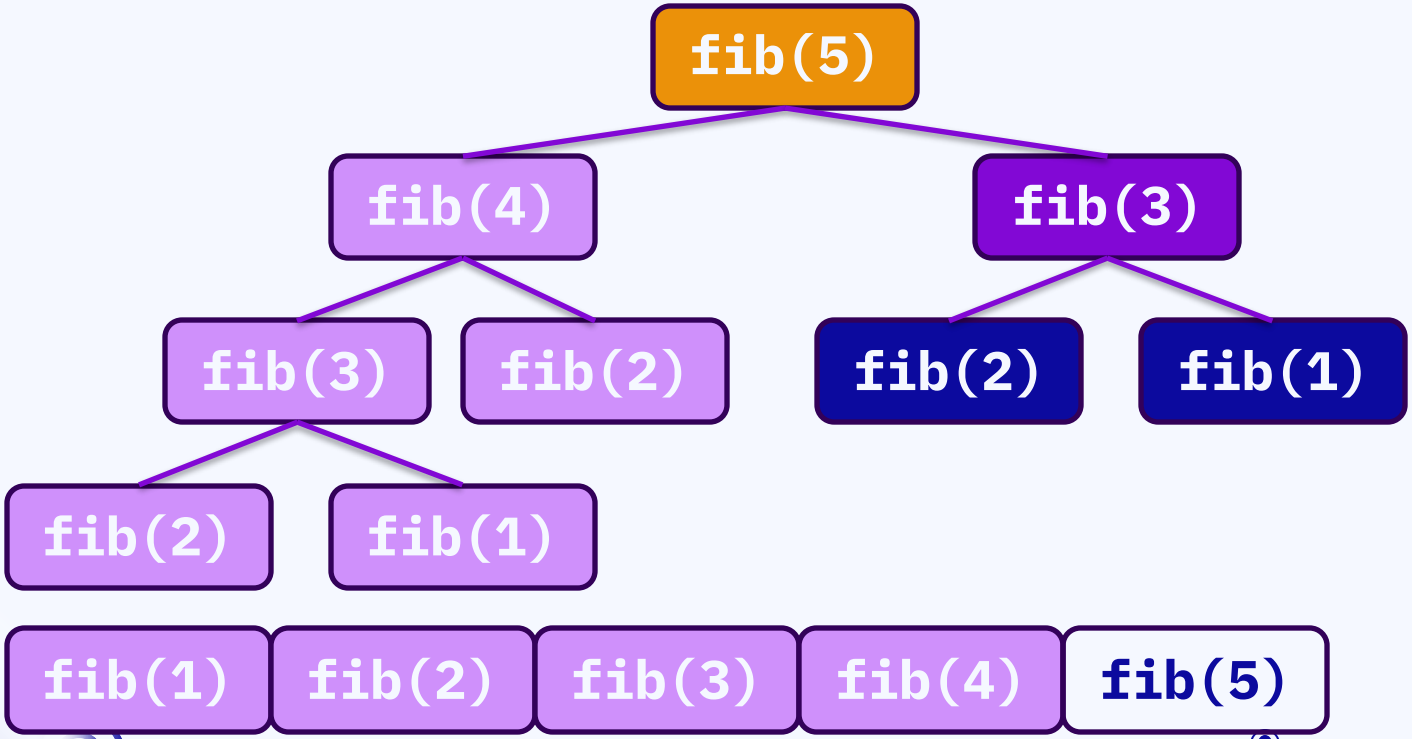
2.2 Abstraction with function

递归– 问题：fib(3)已经算过了，可还要计算一遍！！



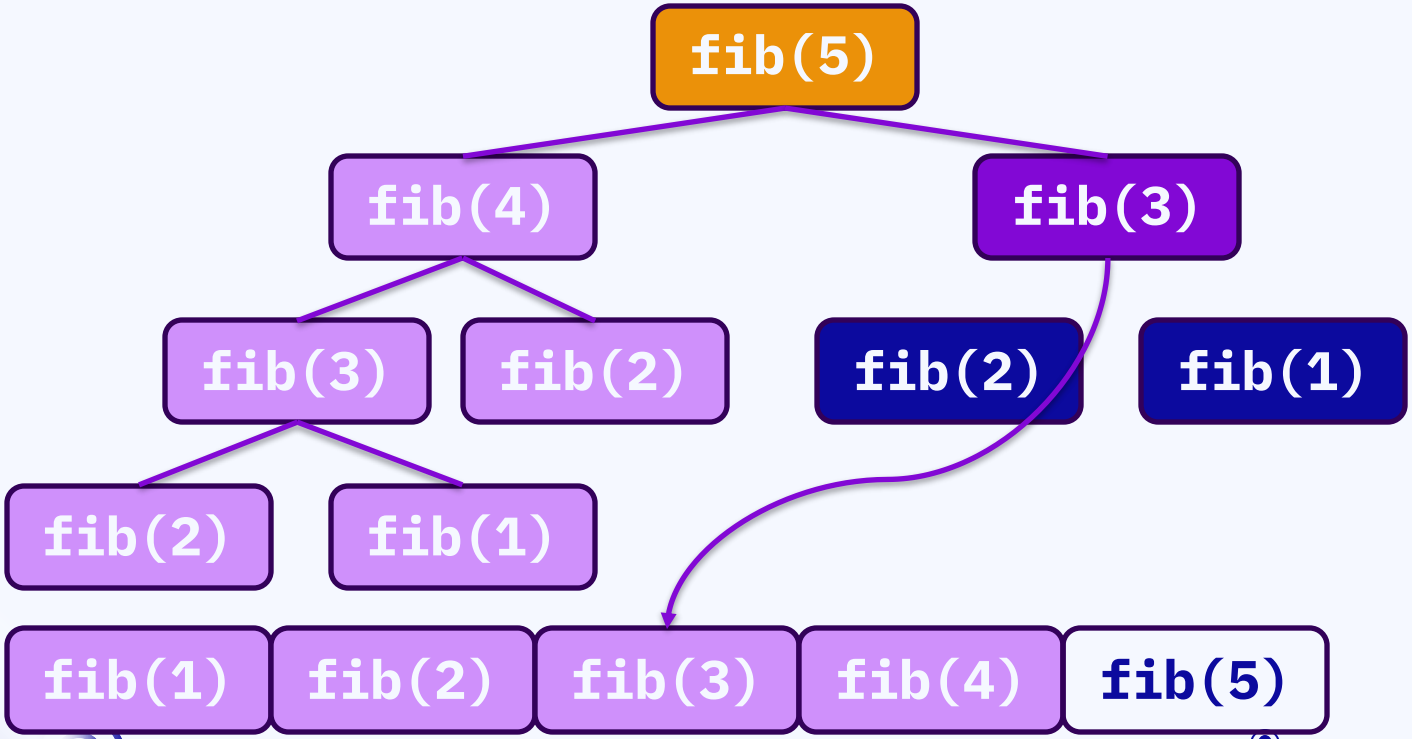
2.2 Abstraction with function

递归– 方法1:存储运算过的结果



2.2 Abstraction with function

递归– 方法1:存储运算过的结果



2.2 Abstraction with function

递归– 方法1:存储运算过的结果（python version):

```
results = [-1] * 3000
def m(arg):
        if results[arg] != -1  # it was computed before
                return results[arg];
        result = ...  # compute...
        results[arg] = result
        return result
```

2.2 Abstraction with function

递归– 方法1:存储运算过的结果（c version):

```
results[M][N] = {};
int m(arg1, arg2) {
        if (results[arg1][arg2] != 0)  // it was computed before.
                return results[arg1][arg2];
        int result = … // computing the result
        results[arg1][arg2] = result;
        return result;
}
```

2.2 Abstraction with function

递归– 方法1:存储运算过的结果

好处：非常简便！！

首先实现基本的递归函数，然后加上“记忆”，判断之前有没有算过

如果没算过：计算，并且存到记忆中

如果算过：直接输出记忆值

“递归函数有几个参数，就开几维数组/列表”

在初级阶段完全可以作为动态规划的平替方式

Disadvantages: 调用函数的开销比较大，内存消耗较大

2.2 Abstraction with function

递归– 方法2: 动态规划

能不能……干脆用一个循环去“取代”函数？

```
for (int i = 2; i < N; ++i) {
        fibo[i] = fibo[i – 2] + fibo[i – 1];   // 状态转移方程
}
```

转“递归”为“递推”

状态转移方程：理解成调用递归函数中的那个公式

2.2 Abstraction with function

递归– 方法2: 动态规划

能不能……干脆用一个循环去“取代”函数？

```
for (int i = 2; i < N; ++i) {
        fibo[i] = fibo[i – 2] + fibo[i – 1];   // 状态转移方程
}
```

转“递归”为“递推”

状态转移方程：理解成调用递归函数中的那个公式

注意循环的顺序！！

03

Exercises

实战演练

1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**

例：输入231，输出33

（231 = 33 + 66 + 132，他们的最大公因数是33）

How to solve it?

1. 数学密码-求最大公因数

补充：求最大公因数最快的方法：辗转相除

```
def gcd(a, b):
    m, n = max(a, b), min(a, b)
    if m % n == 0:
        return n
    return gcd(n, m % n)
```

1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**

想法1：先拆出这三个正整数，确保他们的和是N，

然后计算他们的最大公因数。

可是……情况太多了！！！会Time Out。

1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**

想法1：先拆出这三个正整数，确保他们的和是N，

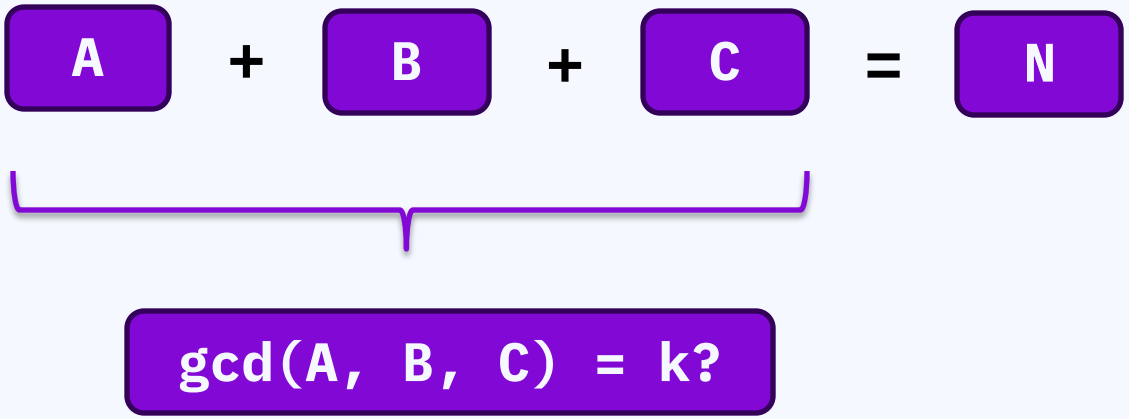
然后计算他们的最大公因数。

可是……情况太多了！！！会Time Out。

这种类型的题目，一定有数学上的Trick!

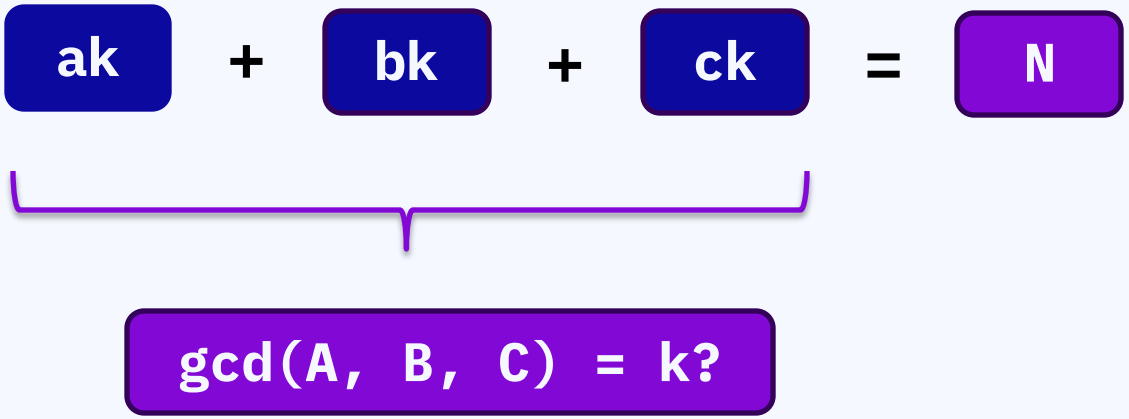
1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**



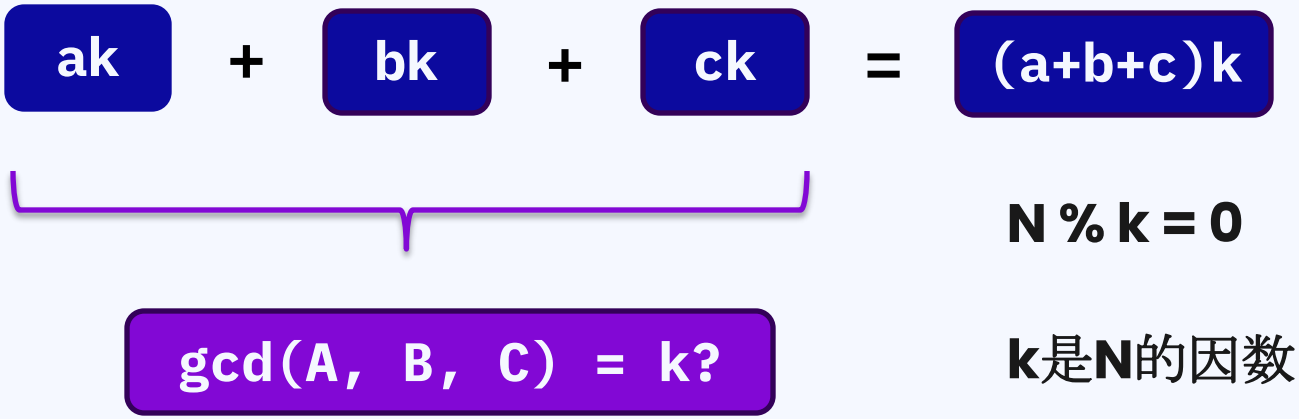
1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**



1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**



1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**

(a+b+c)k

N / k = a + b + c

A, B, C互不相等，则**a, b, c**也互不相等

a + b + c >= 1 + 2 + 3 = 6

1. 数学密码

给定三个互不相同的正整数的和**N**，快速求出**这三个正整数的最大公因数。(N < 10^9)**

(a+b+c)k

因此这道题变成了：

求**N**的最大的因数**m**，且满足**N % m >= 6**

!!!

1. 数学密码- Conclusion

逆向思维很重要！！

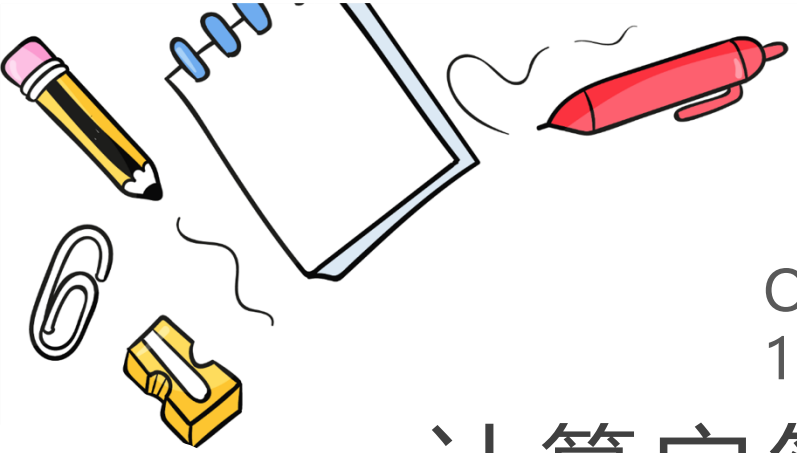
分析题干里面的数值范围，如果范围很大（一般超过10^5左右），简单进行两次循环一定是会超时的，一定会有更巧妙的算法。

如：整个过程有没有可以避免的重复性计算？

有没有什么还没发现的数学规律？...

Openjudge 自由演武场
11-H

计算字符串的距离




计算字符串距离

对于两个不同的字符串，我们有一套操作方法来把他们变得相同：

·修改一个字符（如把“a”替换为“b”）

·删除一个字符（如把“traveling”变为“travelng”）

比如对于“abcdefg”和“abcdef”两个字符串来说，我们认为可以通过增加/减少一个“g”的方式来达到目的。无论增加还是减少“g”，我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。

给定任意两个字符串，写出一个算法来计算出他们的距离。

计算字符串距离

输入和输出

输入

第一行有一个整数n。表示测试数据的组数，

接下来共n行，每行两个字符串，用空格隔开。表示要计算距离的两个字符串

字符串长度不超过1000。

输出

针对每一组测试数据输出一个整数，值为两个字符串的距离。

计算字符串距离

例子分析

答：一个通宵。

a: 赶上明天

b: 赶不上思修[illegible]的DDL，摆烂了

问：a和b的距离是多少？

计算字符串距离

例子分析

AB	AB	答案：0
ABC	BC	答案：1
APC	CPC	答案：1

计算字符串距离

可能部分同学对这道题已经很熟悉了，部分同学还是第一次见……

但它是字符串操作里的经典！

如果从未见过这类题，第一次能想出题解确实很难。

计算字符串距离

同类题好像都会了……

思考过程呢?

遇到新题

不会

AC!

从0到1的思考过程真的很重要!

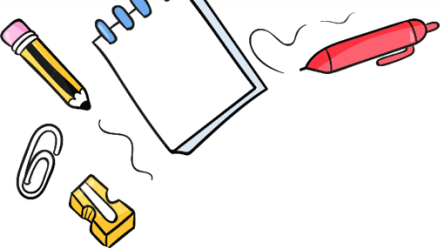
计算字符串距离

第一次看到

UC Berkeley CS 61A Project 2

CATS

CS 61 A
Autocorrected
Typing
Software



计算字符串距离

CS61A Autocorrected Typing Software

Leaderboard

WPM: 17.0 | Accuracy: 90.9 | Time: 47.4

Look at the following words:

When dribbling past an opponent, the dribbler should dribble with the hand farthest from the opponent, making it more difficult for the defensive player to get to the ball.

And type them below:

When dribbling past an **opponent,** the **fibber** should dribble **with** the

Enable Auto-Correct

Restart

Specify topics of interest

Cat, Cats, Kittens, ... Submit

List topics separated by commas.

计算字符串距离

Programmers dream of
Abstraction, recursion,
and
Typing really fast.

Problem 7 (3 pts) 计算字符串距离

Implement `minimum_mewtations`, which is a diff function that returns the minimum number of edit operations needed to transform the `typed` word into the `source` word.

There are three kinds of edit operations, with some examples:

1. Add a letter to `typed`.
    - Adding `"k"` to `"itten"` gives us `"kitten"`.
2. Remove a letter from `typed`.
    - Removing `"s"` from `"scat"` givs us `"cat"`.
3. Substitute a letter in `typed` for another.
    - Substituting `"z"` with `"j"` in `"zaguar"` gives us `"jaguar"`.

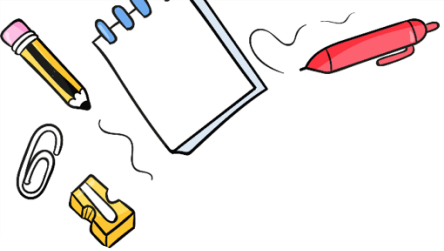
Each edit operation contributes 1 to the difference between two words.

```
>>> big_limit = 10
>>> minimum_mewtations("cats", "scat", big_limit)       # cats -> scats -> scat
2
>>> minimum_mewtations("purng", "purring", big_limit)   # purng -> purrng -> purring
2
>>> minimum_mewtations("ckiteus", "kittens", big_limit) # ckiteus -> kiteus -> kitteus ->
3
```

思路详解-举例

A B
C

A B
C



举例

上北大　　上北大

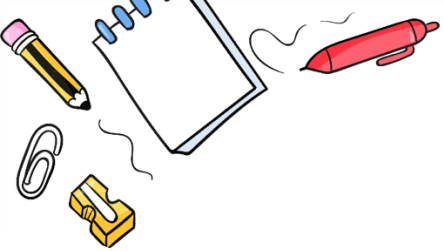
Distance：0

是不是直接比较两个字符串之间的差异就行了？
好像不太行……
字符串之间的差异可能比较大

举例

D A B
C

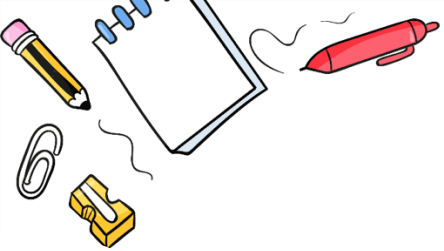
A B
C



举例

上北大　　上北大

Distance：1



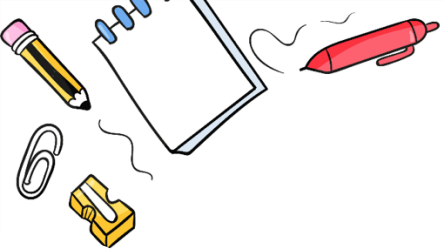
举例

上北大　　要上北大

Distance：1

Observation 1:两侧的字符串可以互换，结果不变

Observation 2:添加一个字符，相当于在对面删除一个字符



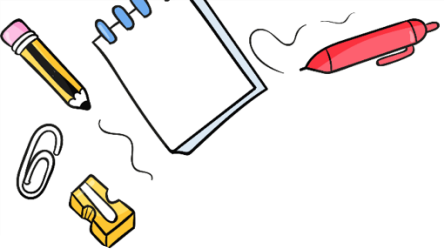
举例

北大　　清华

Distance：2

Observation 3:前端相同的字符可以删去，不影响结果

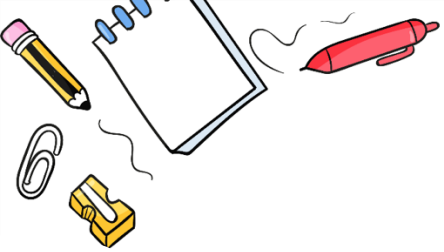
Intuition: 从两个字符串的前端开始操作



举例

Question:如果前端的字符不同呢?

上清华	上清华	1	Distance 0 1
上清华	上清华	2	Distance 0 1
上清华	上清华	3	Distance 0 1



举例

Question:如果前端的字符不同呢?

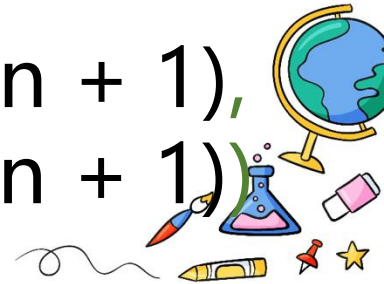
1. 两边同时删掉一个字符
2. 右边字符串删掉一个字符
3. 左边字符串删掉一个字符

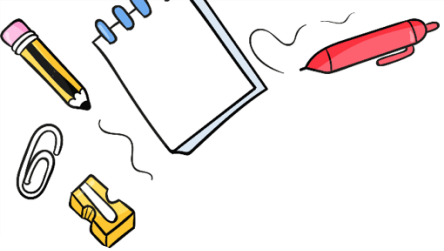
取后面字符串距离的min加1

Key

Define distance(m, n) as the distance between StringA[m:] to StringB[n:]

```
if StringA[m] == StringB[n] then
   distance(m, n) = distance(m + 1, n + 1)
else
   distance(m, n) = min(distance(m, n + 1), distance(m + 1, n), distance(m + 1, n + 1)) + 1
```



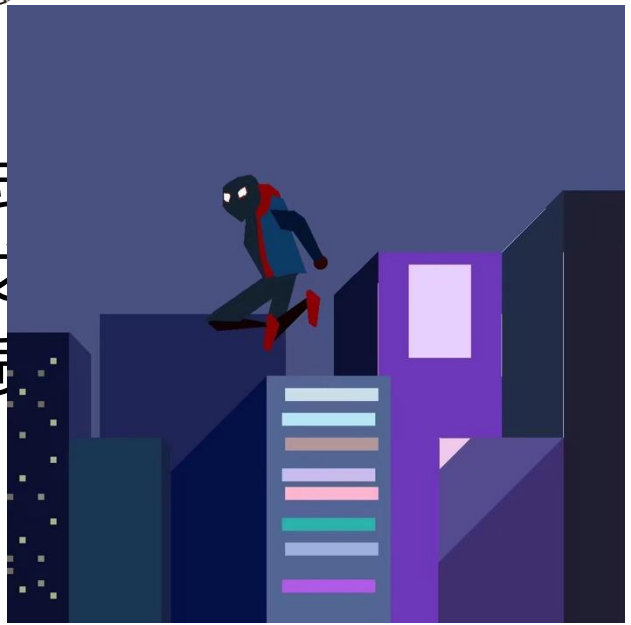


Base Case

distance (strlen(a), strlen(b)) = 0
distance (strlen(a), n) = strlen(b) - n
distance (m, strlen(b)) = strlen(a) - m

求a和b的距离，就相当于求distance (0, 0)

怎样一步步想出来的？



1.举例：遇到没思路的题，先举例子
2.分类：分情况讨论，抓住不同情况
3.将问题分解：递归、动态规划的本
解成子问题

The Recursive Leap of Faith!

2022 Final No.8 01串翻转

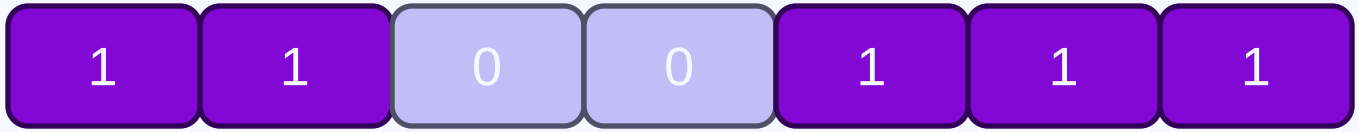
你有两个仅由 0 跟 1 组成的相同长度的字符串 a, b 。
现在你可以对字符串 a 进行如下操作：
选择其中一个 1 不动，并让其余所有的数字翻转 (1 -> 0, 0 -> 1)。

例如：假如 a = 0011，我们固定第一个 1，那么操作之后的 a 为 1110 （只有第一个 1 不改变，别的所有数字都翻转）。
再假如 a = 1101011，我们固定最后一个 1，那么操作之后的 a 为 0010101 （只有最后一个 1 不改变，别的所有数字都翻转）。

现在请问你可以把字符串 a 变成字符串 b 吗？如果可以，请问最少需要多少次操作能把 a 变成 b 呢？

2022 Final No.8 01串翻转

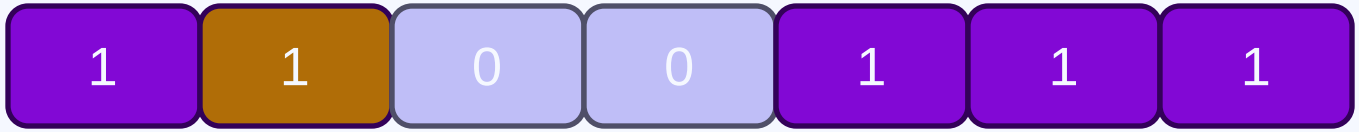
String 1



String 2

2022 Final No.8 01串翻转

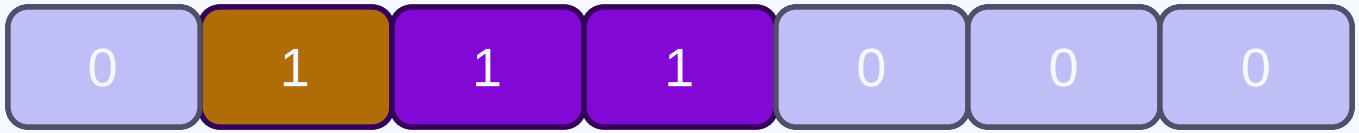
String 1



String 2

2022 Final No.8 01串翻转

String 1



String 2

2022 Final No.8 01串翻转

String 1

String 2

2022 Final No.8 01串翻转

String 1

String 2

2022 Final No.8 01串翻转

String 1

String 2

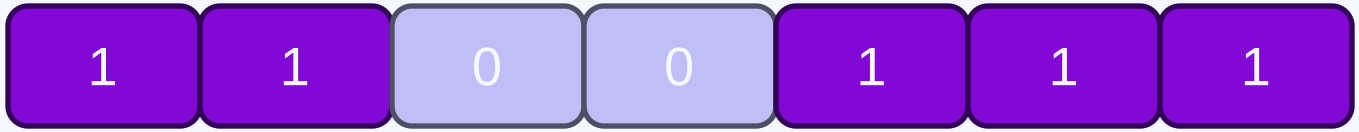
2022 Final No.8 01串翻转

String 1

String 2

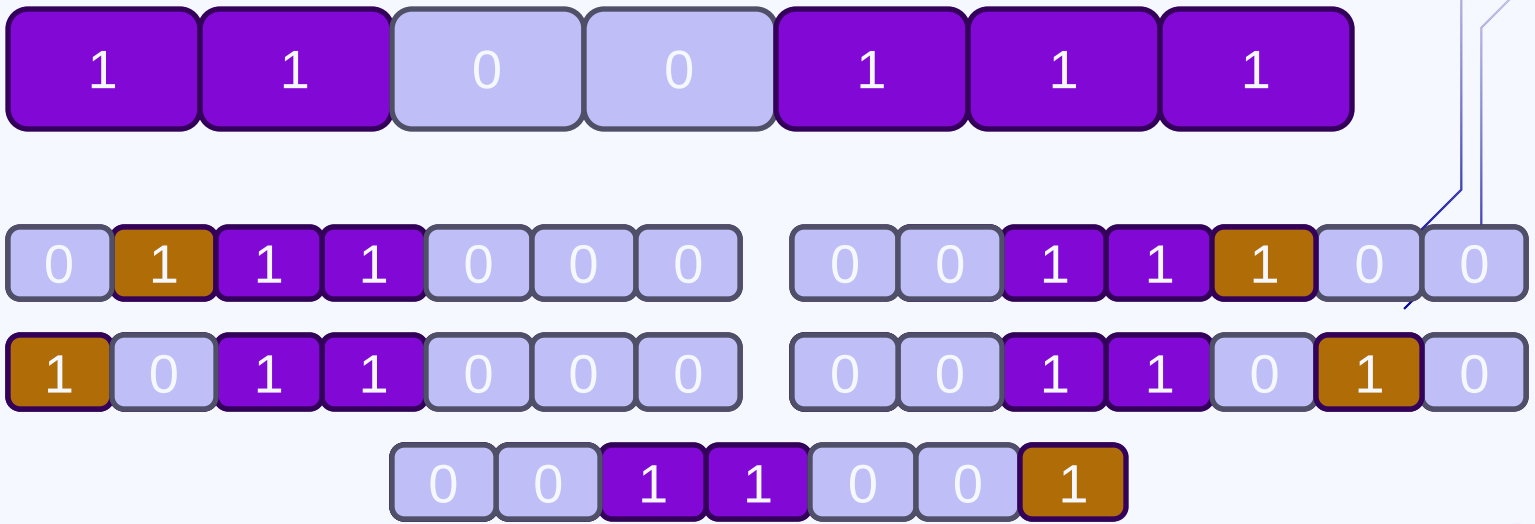
2022 Final No.8 01串翻转

How to solve it? No idea…



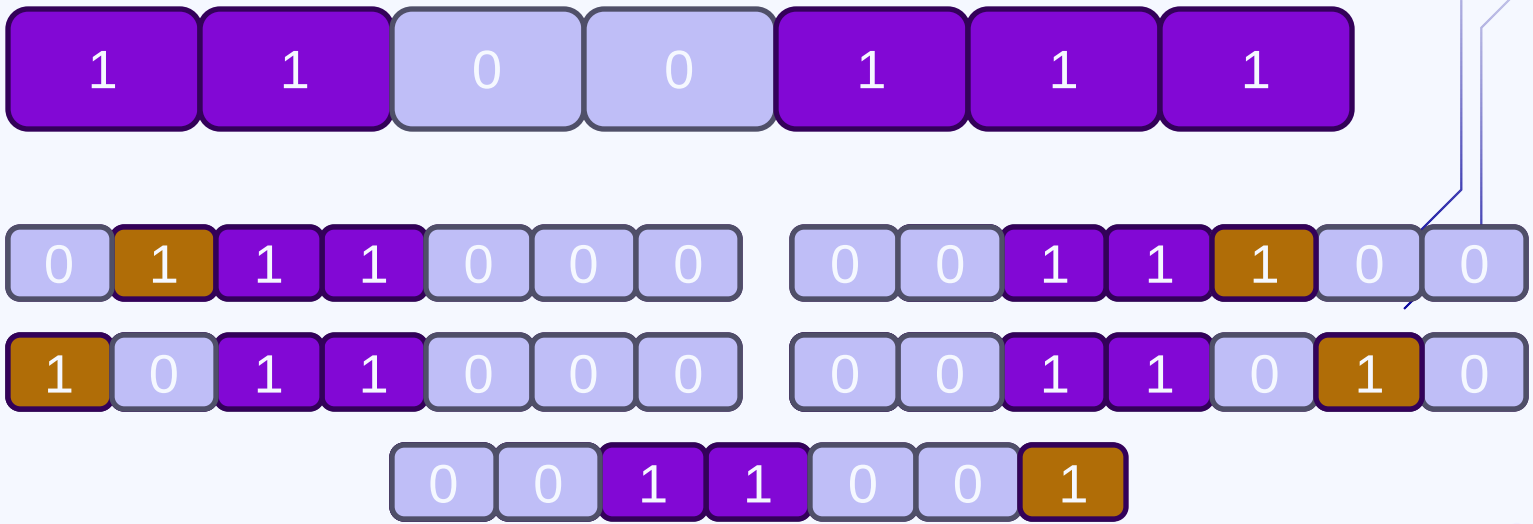
2022 Final No.8 01串翻转

How to solve it? No idea…



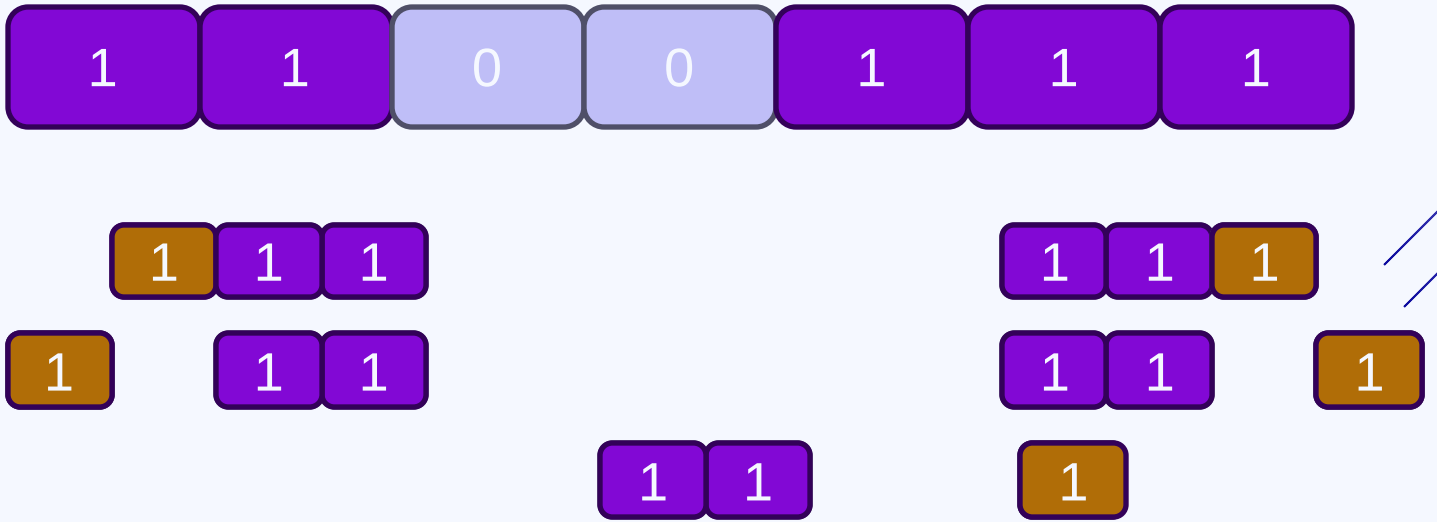
2022 Final No.8 01串翻转

So many cases…
But isn't there something similar?



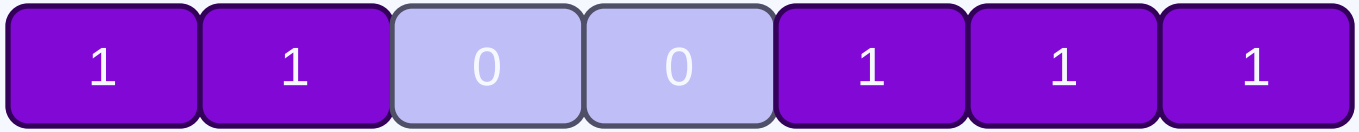
2022 Final No.8 01串翻转

So many cases…
But isn't there something similar?



2022 Final No.8 01串翻转

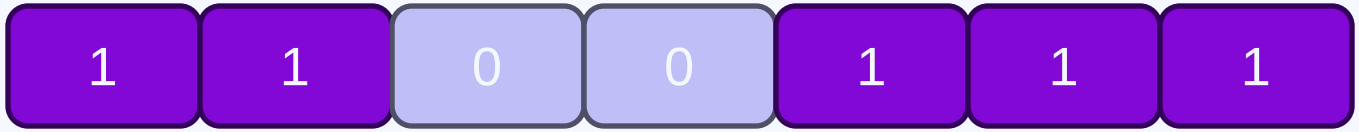
So many cases…
But isn't there something similar?



The number of 1s after flipping is always 3!!

2022 Final No.8 01串翻转

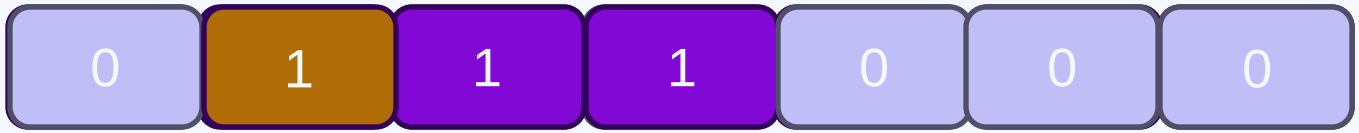
So many cases…
But isn’t there something similar?



Initial: L items, k 1s, L – k 0s

2022 Final No.8 01串翻转

So many cases…
But isn't there something similar?



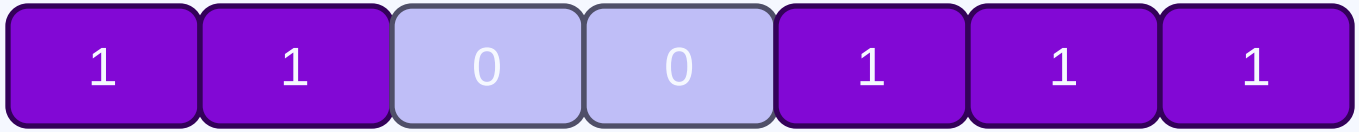
Initial: L items, k 1s, L – k 0s

After flipping: (L – k + 1) 1s

What about… flipping again?

2022 Final No.8 01串翻转

So many cases…
But isn't there something similar?



Initial: L items, k 1s, L – k 0s

After flipping: (L – k + 1) 1s

What about… flipping again?

L – (L – k + 1) + 1 = … k !!!!!

2022 Final No.8 01串翻转 Intuition 1

因此，假设原先的字符串中有k个1，

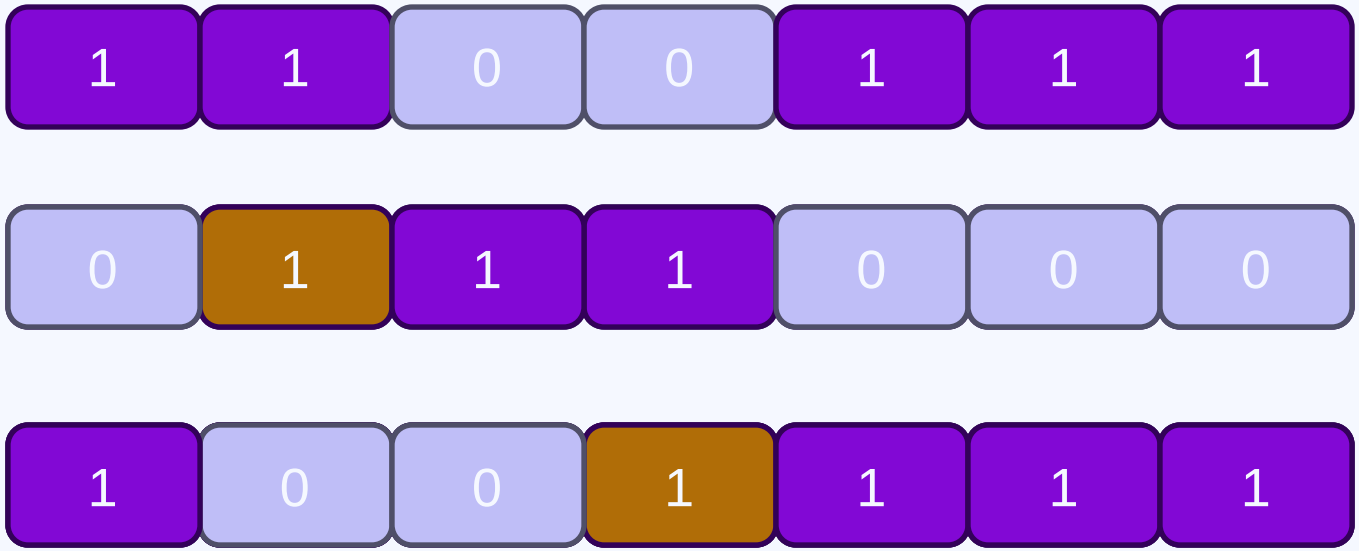
改变后只能产生有k个1或（n-k+1）个1.

K对应的翻转次数为偶数（case 1），

n-k+1对应的翻转次数为奇数（case 2）。

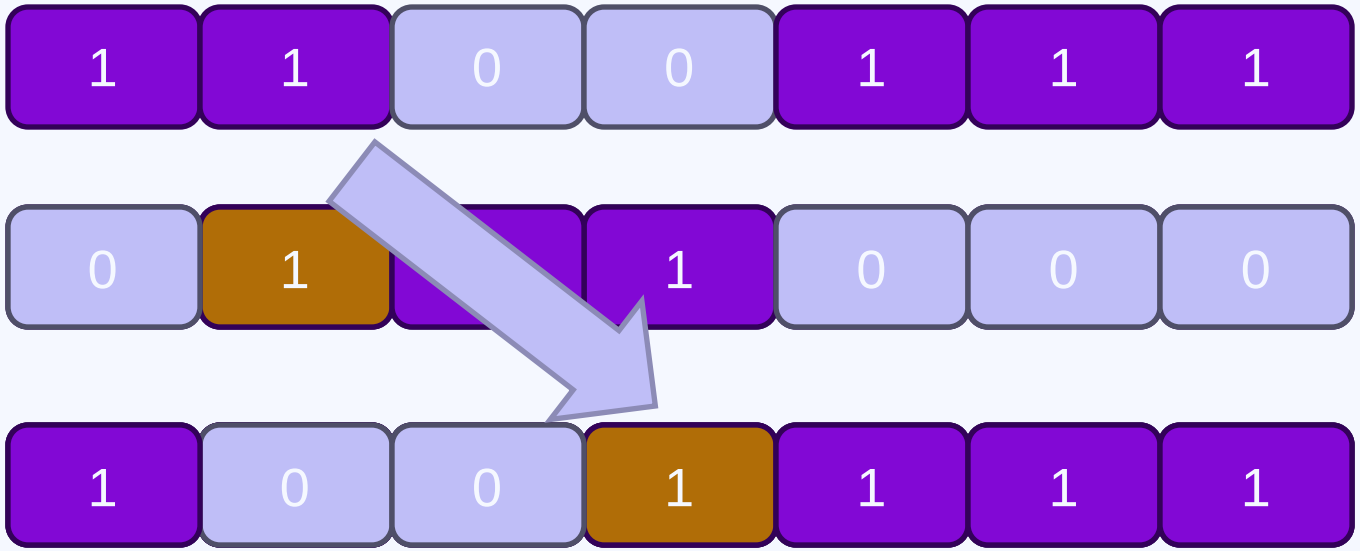
2022 Final No.8 01串翻转

翻转两次会有什么效果呢？



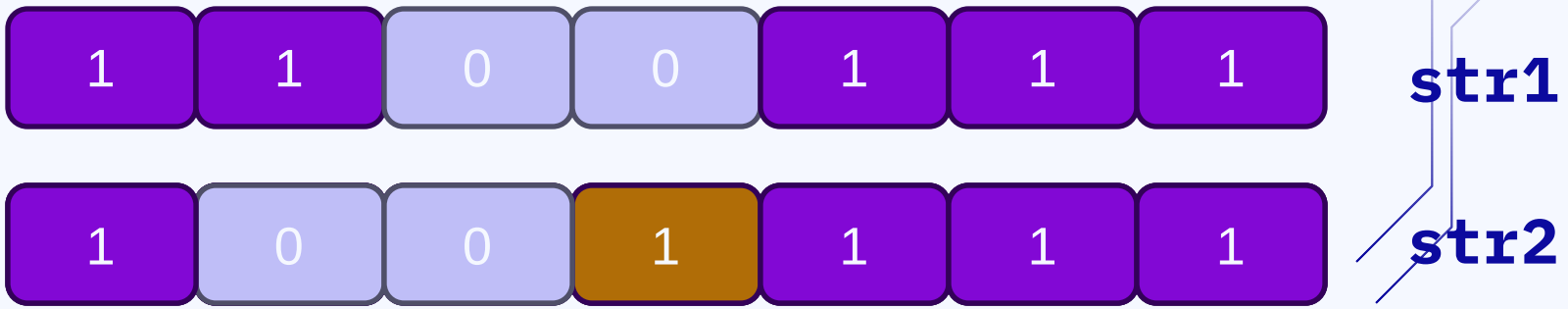
2022 Final No.8 01串翻转 Intuition 2

翻转两次会有什么效果呢？1被移动了！！



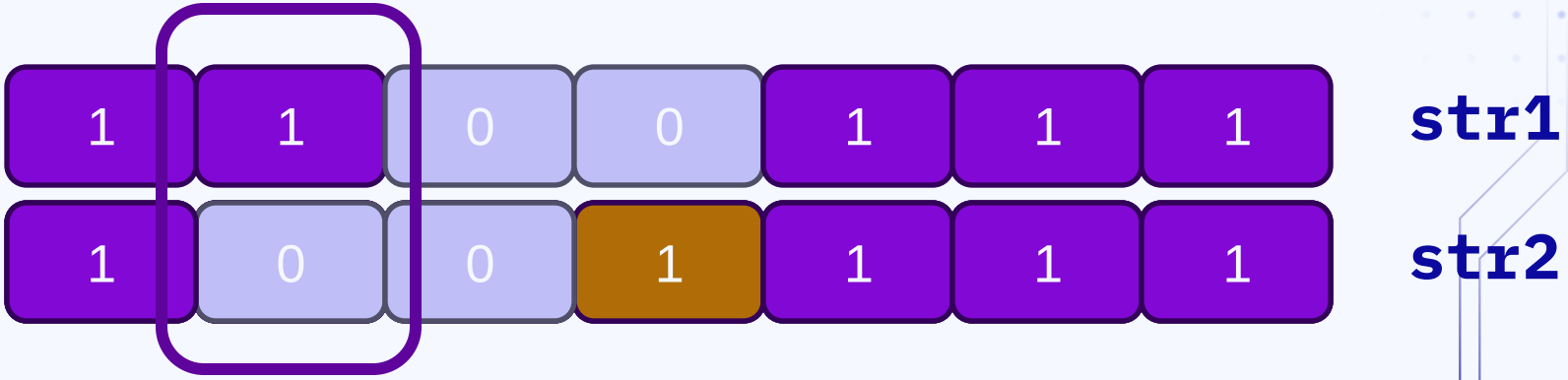
2022 Final No.8 01串翻转 Intuition 2

翻转两次会有什么效果呢？1被移动了！！



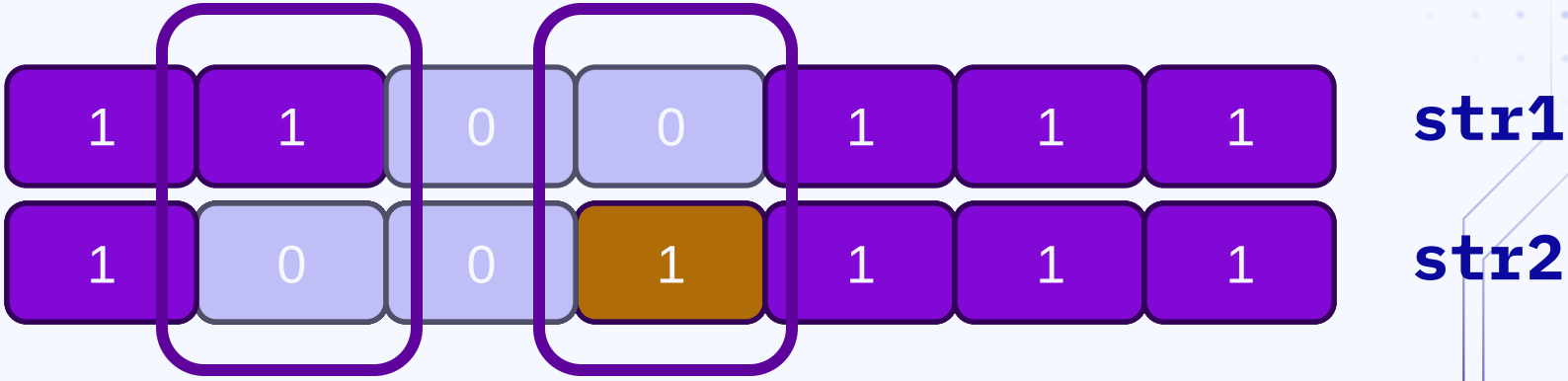
移动两次能任意改变1的位置。最优？向着目标的方向移动！

2022 Final No.8 01串翻转 Intuition 2



有几个1的位置本该是0，就该移动2*几下

2022 Final No.8 01串翻转 Intuition 2



有几个1的位置本该是0，就该移动2*几下

等价于：str1和str2有几个位置元素有差别，

就要移动几下！

2022 Final No.8 01串翻转 Intuition 2

因此，假设原先的字符串中有k个1，

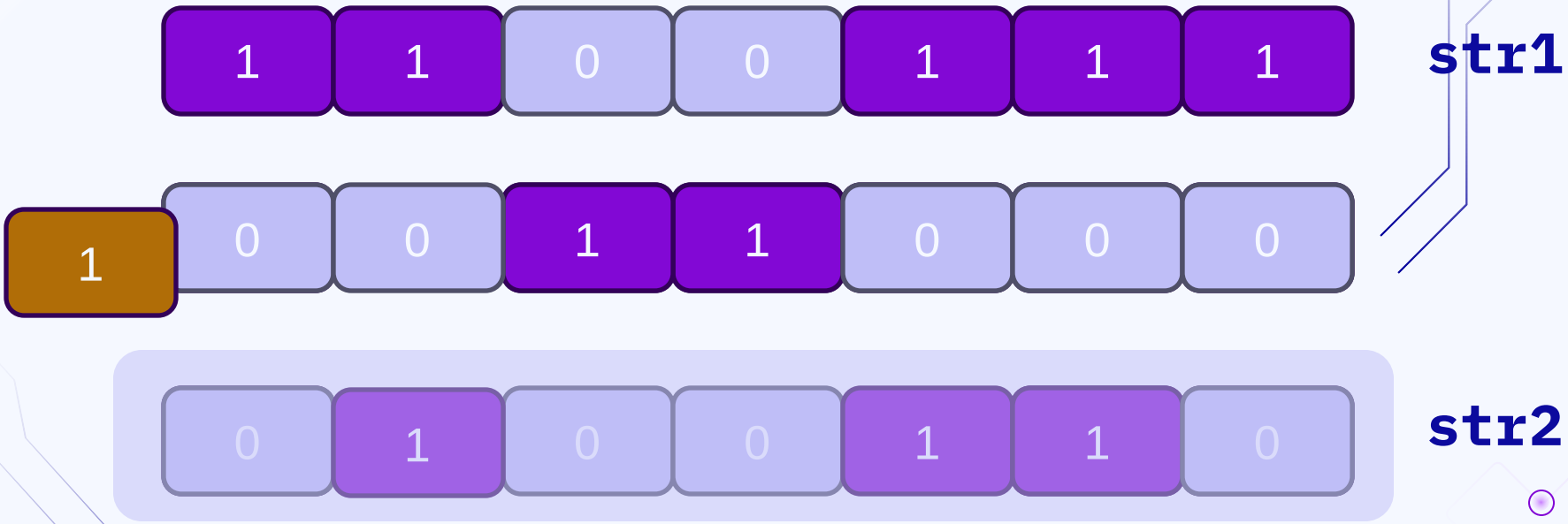
改变后只能产生有k个1或（n-k+1）个1.

K对应的翻转次数为偶数（case 1），(dif(str1, str2))

n-k+1对应的翻转次数为奇数（case 2）。

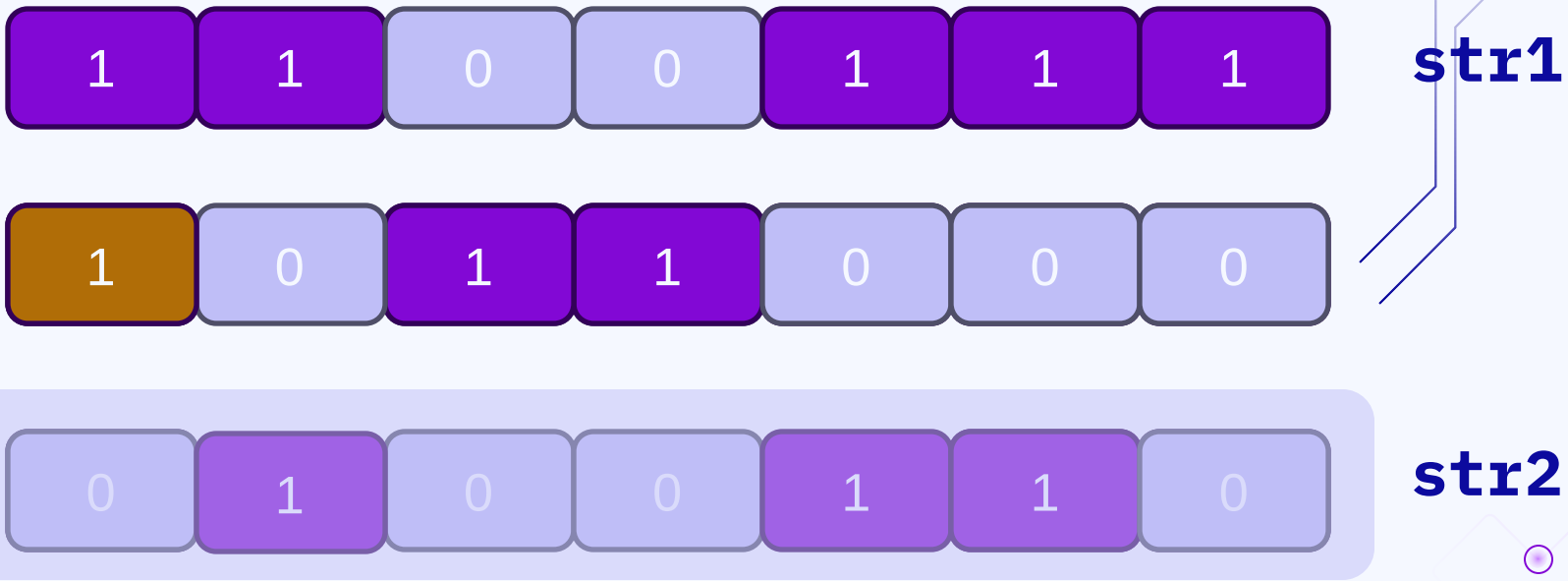
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？



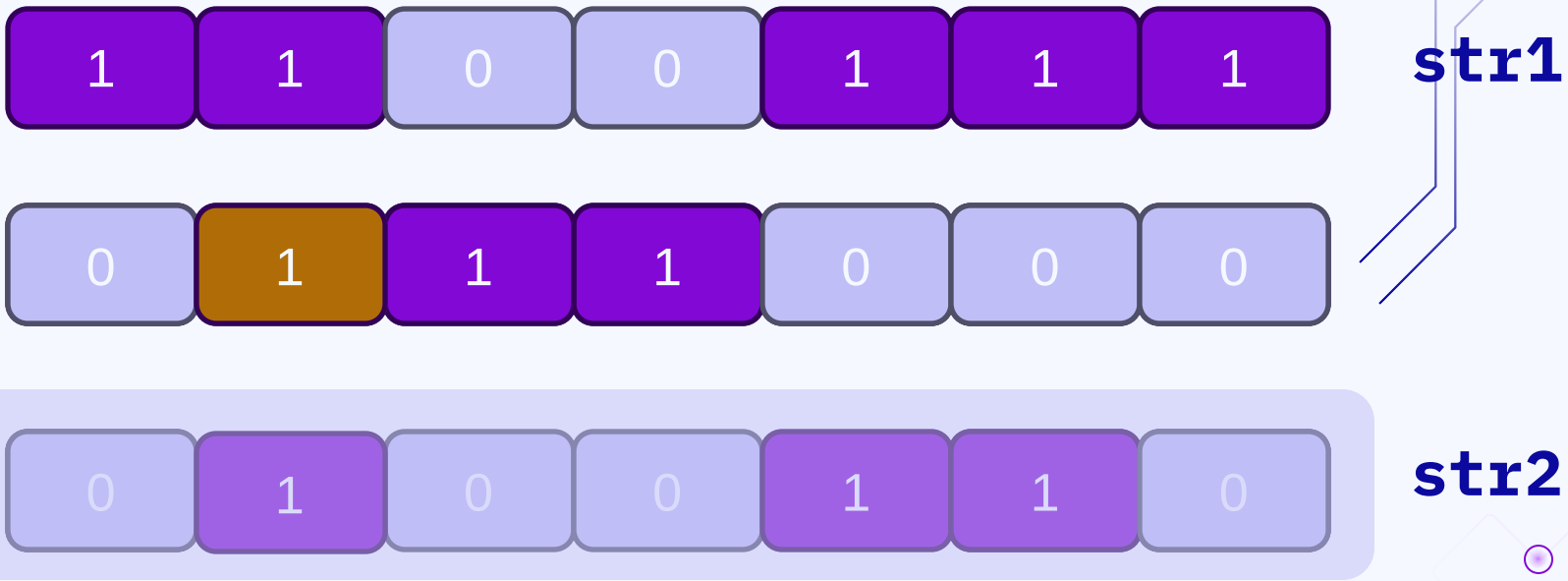
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？



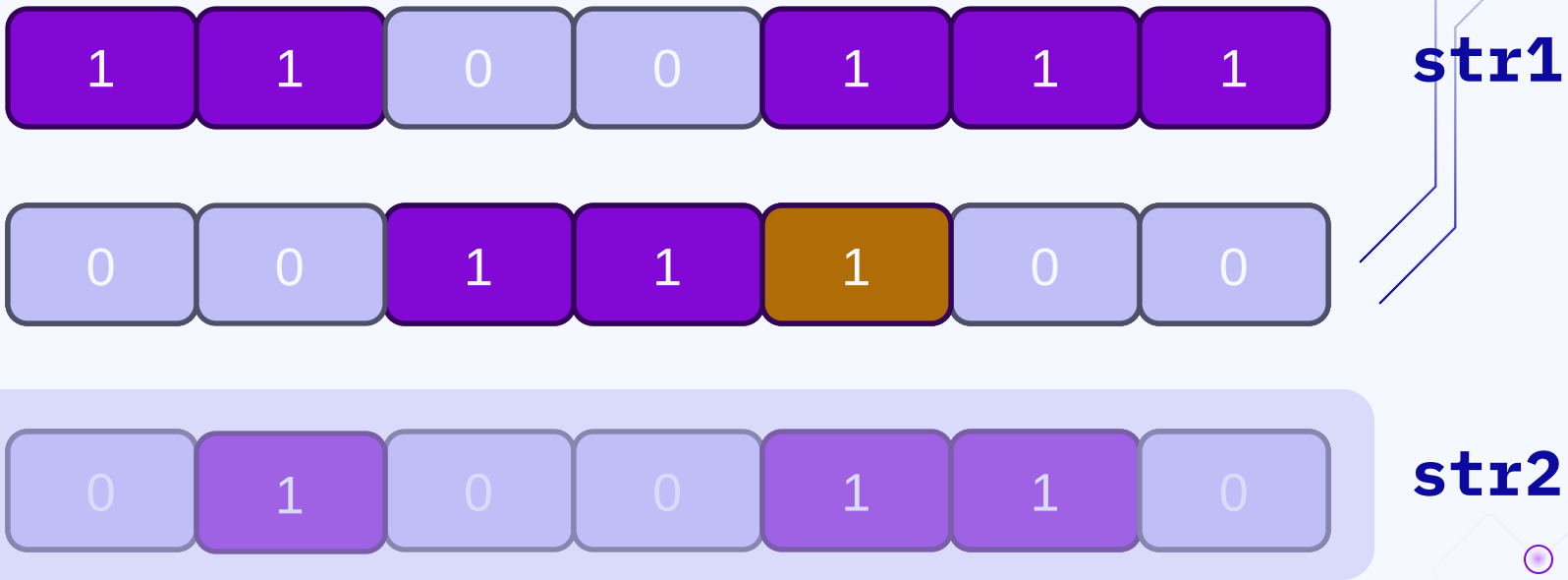
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？



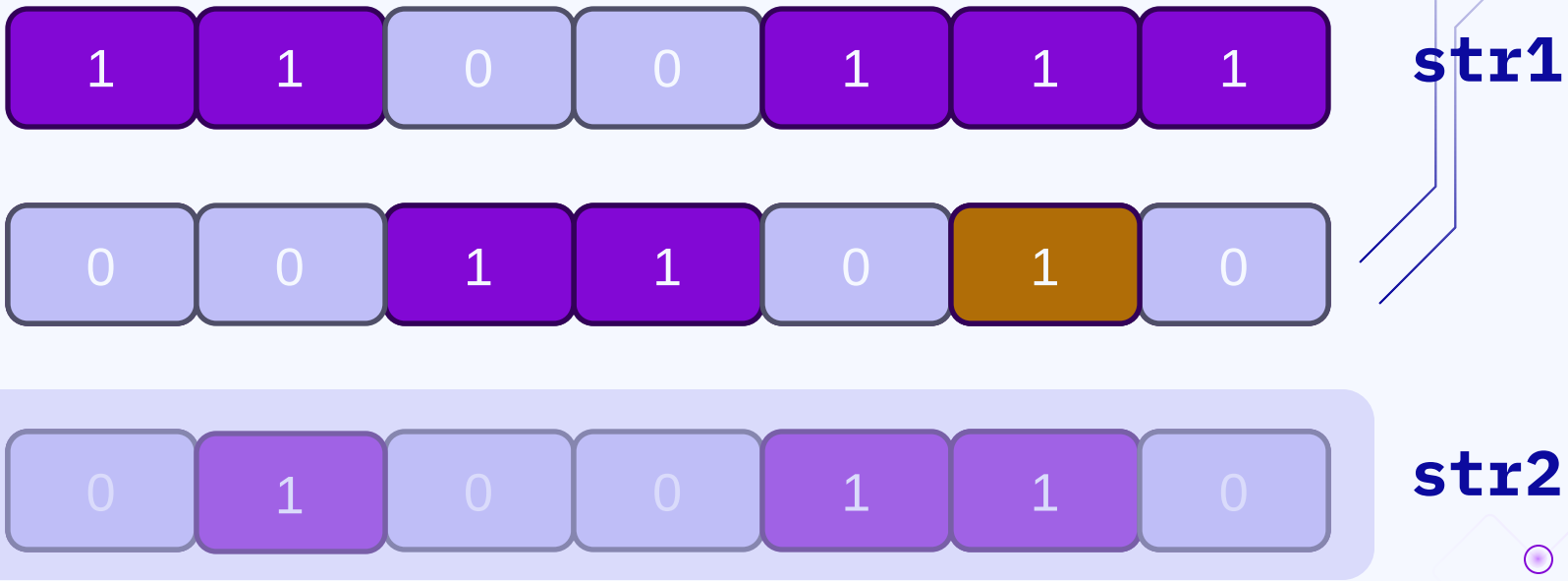
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？



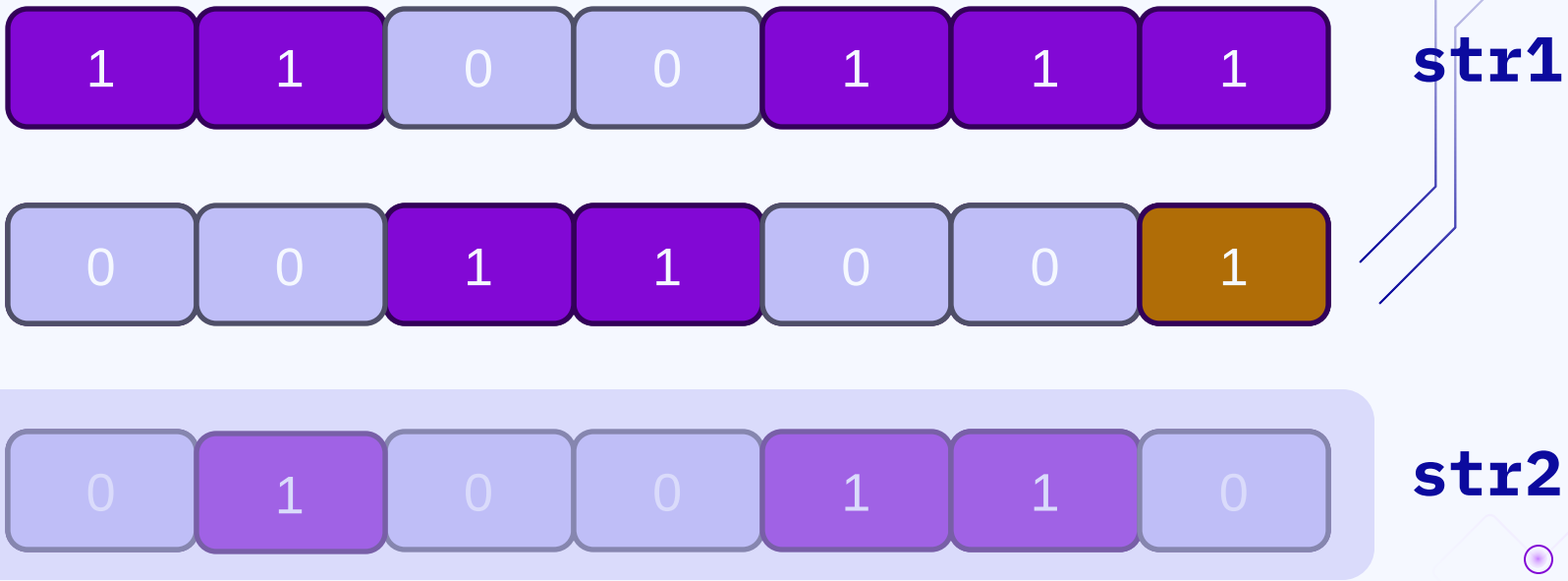
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？



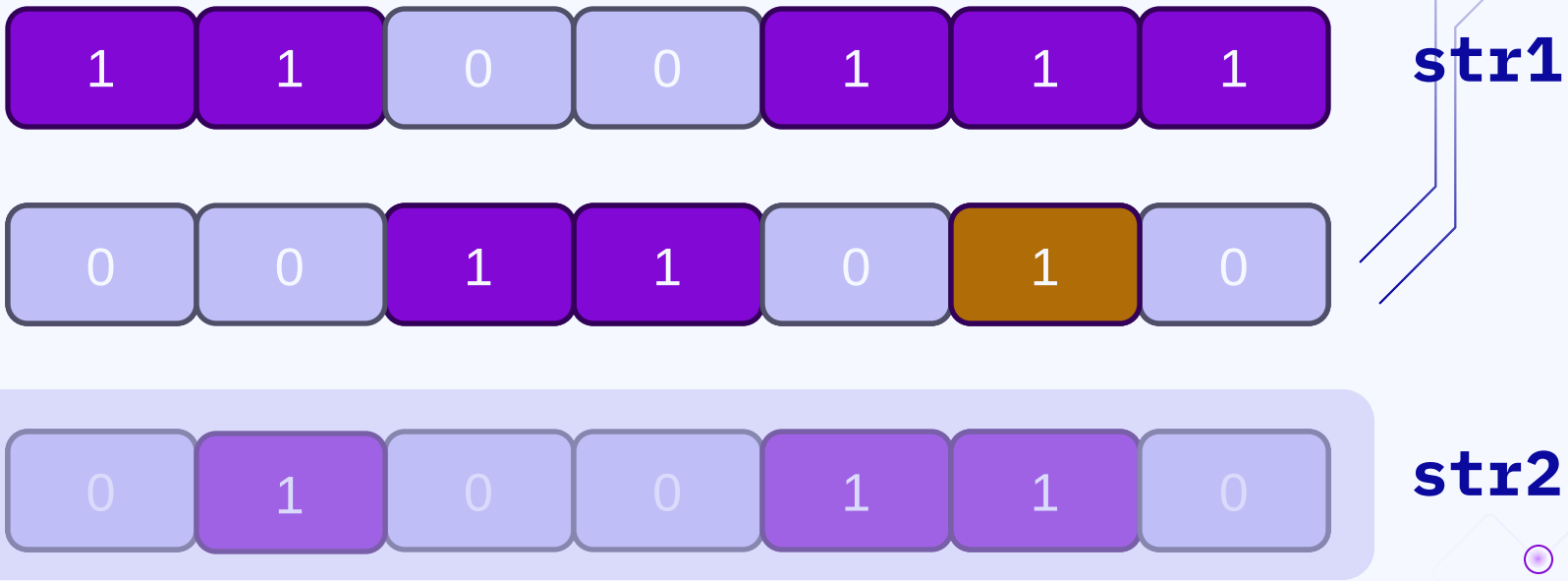
2022 Final No.8 01串翻转

翻转一次会有什么效果呢？

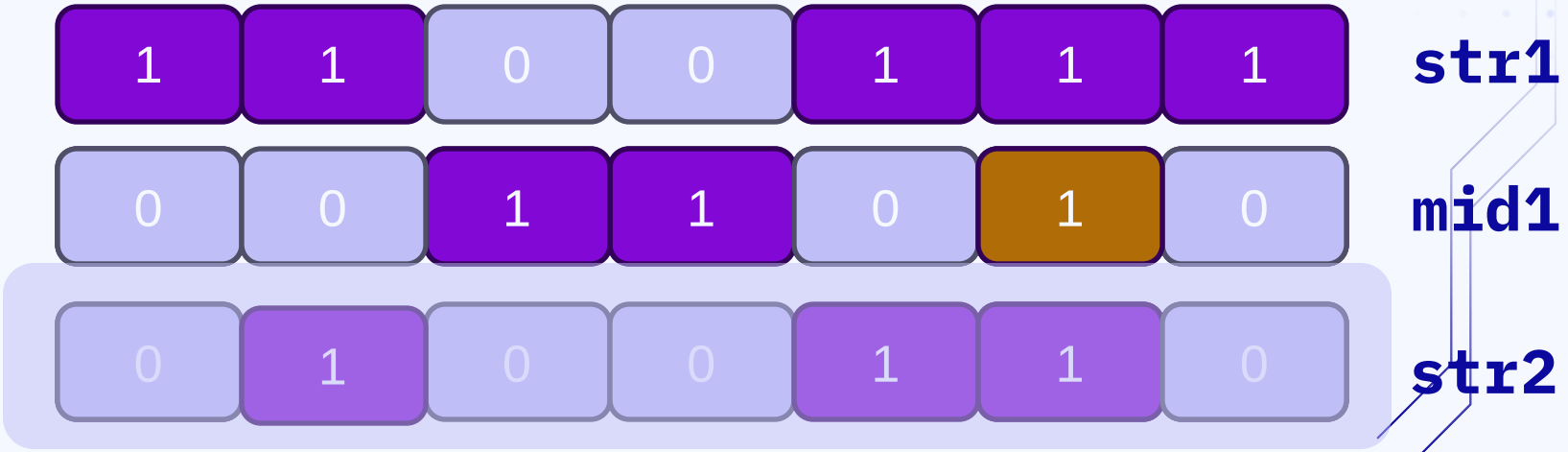


2022 Final No.8 01串翻转

全部翻转，并且改变一个0变成1。

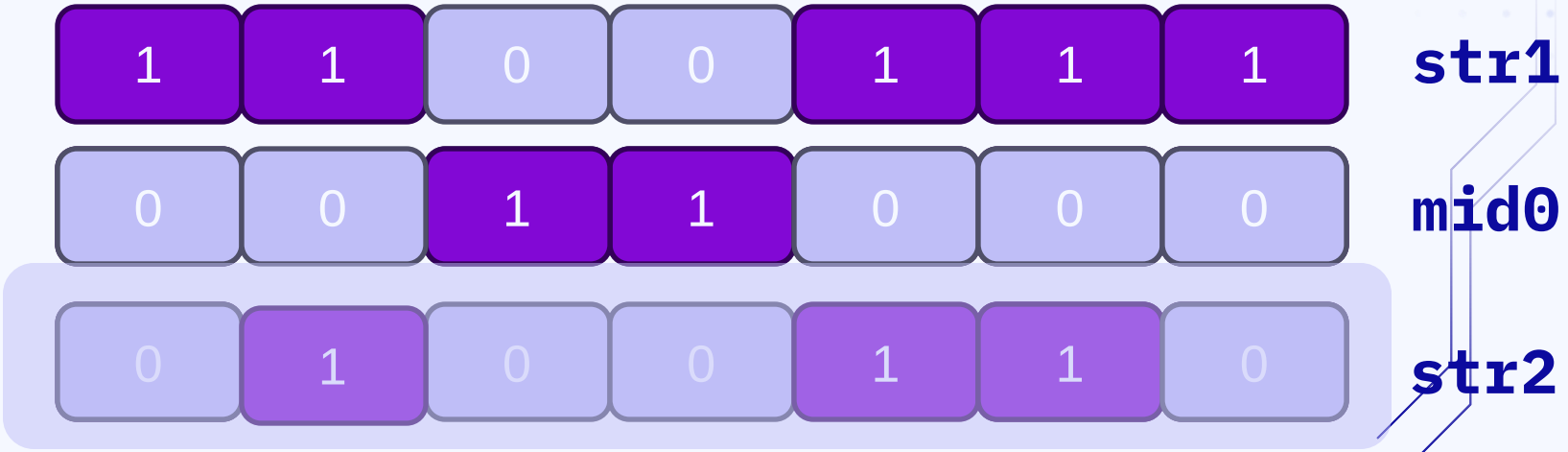


2022 Final No.8 01串翻转



从mid1到str2的最少步骤：dif(mid1, str2)(同case1)

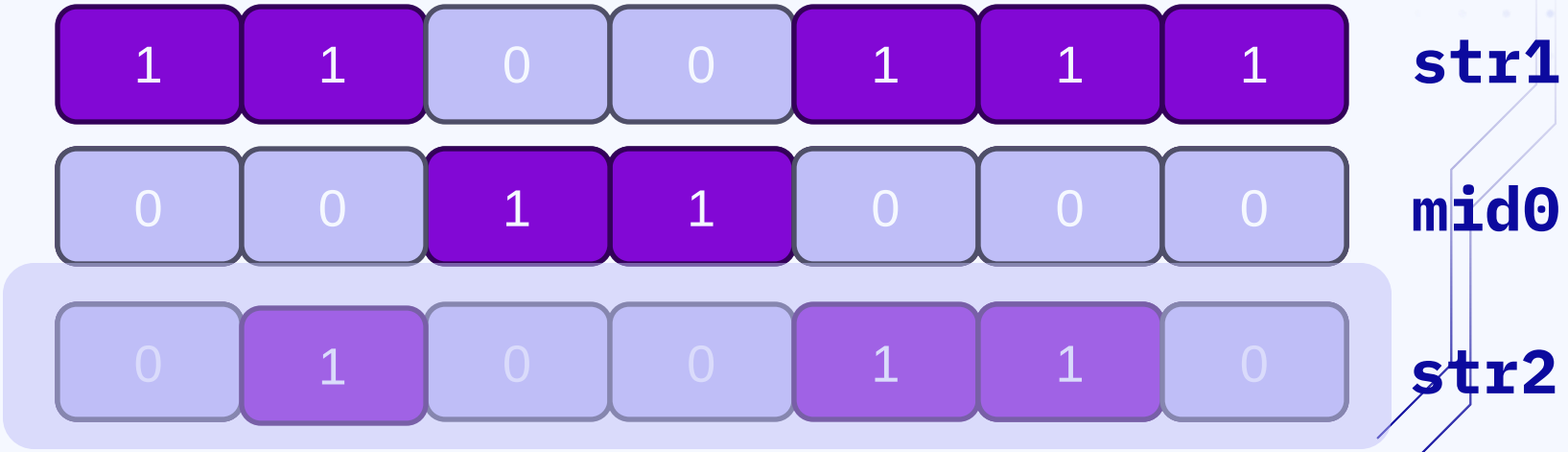
2022 Final No.8 01串翻转



从mid1到str2的最少步骤：dif(mid1, str2)(同case1)

dif(mid1, str2) = dif(mid0, str2) - 1

2022 Final No.8 01串翻转

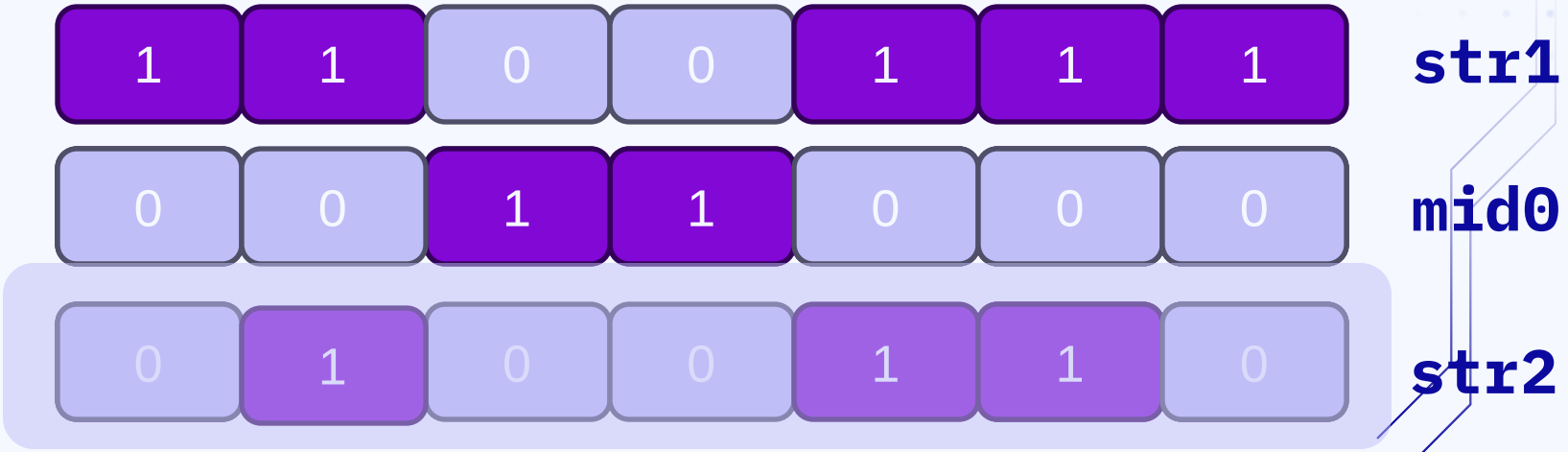


从mid1到str2的最少步骤：dif(mid1, str2)(同case1)

dif(mid1, str2) = dif(mid0, str2) - 1

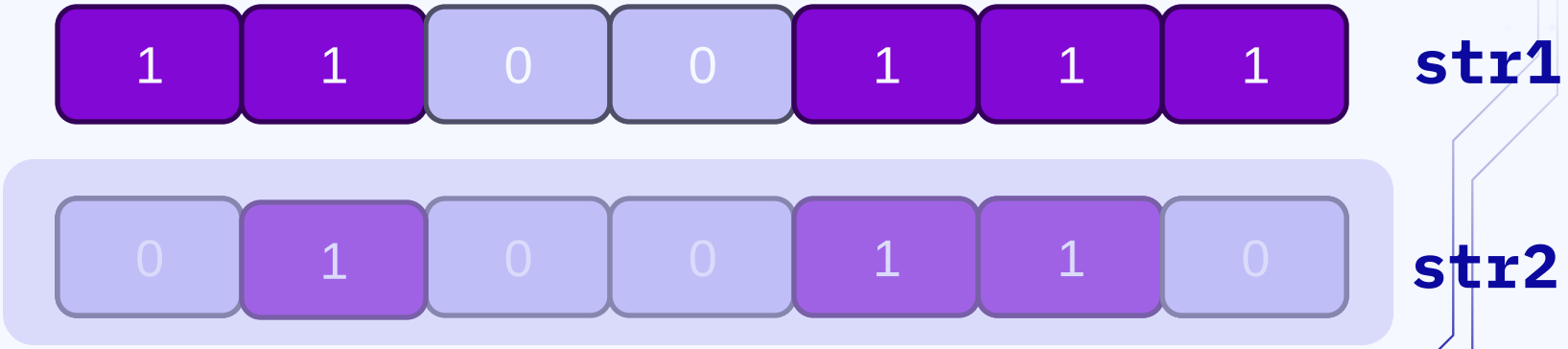
dif(mid0, str2) = same(str1, str2)

2022 Final No.8 01串翻转



从str1到str2的最少步骤：same(str1, str2) !!!

2022 Final No.8 01串翻转



从str1到str2的最少步骤：same(str1, str2) !!!

2022 Final No.8 01串翻转 Intuition 3

因此，假设原先的字符串中有k个1，

改变后只能产生有k个1或（n-k+1）个1.

K对应的翻转次数为偶数（case 1），(dif(str1, str2))

n-k+1对应的翻转次数为奇数（case 2），

(same(str1, str2))

2022 Final No.8 01串翻转 Intuition 3

因此，假设原先的字符串中有k个1，

改变后只能产生有k个1或（n-k+1）个1.

```
if ones1 == ones2:
    # it can be changed by even number of times.
    total = dif(str1, str2)
    print(total)

elif len1 - ones1 + 1 == ones2:
    # it can be changed by odd number of times.
    total = same(str1, str2)
    print(total)
```

2022 Final No.8 01串翻转 Intuition 4

因此，假设原先的字符串中有k个1，

改变后只能产生有k个1或（n-k+1）个1.

（Subtle Point）可如果 k == n - k + 1 呢？

Answer：取两种结果的最小！！

2022 Final No.8 01串翻转 Final Code

```
if ones1 == ones2 and len1 - ones1 + 1 == ones2:
    total = min(dif(str1, str2), same(str1, str2))

elif ones1 == ones2:
    # it can be changed by even number of times.
    total = dif(str1, str2)

elif len1 - ones1 + 1 == ones2:
    # it can be changed by odd number of times.
    total = same(str1, str2)
    print(total)

else:
    total = -1

# C语言完全类似，将str1和str2改成char数组
```

04

Conclusion

一点结语

Conclusion-Some suggestions!

1. 题目顺序并不一定能反应题目难度，**不要在一道题上卡太长时间**！可以看榜，通过准确率来判断难易。
2. 如果考试发草纸的话要充分利用，多列举几种情况、多画画草图。
3. 放松心态，某些大佬们的做题速度可能是很快的（计概A也有部分同学开考40分钟AK所有题目离场……）不用要被他们带跑节奏，按照自己的做题节奏来。

Conclusion From the final lecture of cs61a

You are coming to the end of your FIRST course about computer science! But... It is just a starting point.

Conclusion From the final lecture of cs61a

You are coming to the end of your FIRST course about computer science! But... It is just a starting point.

1. **It is worth building stuff that matters to the world.**
2. **Think of the technology in a critical way ...Let AI be your helper!**
3. **NEVER COMPARE.**

Thanks!

Good luck on your final exam!!

calvincao@stu.pku.edu.cn
+86 186 8665 9012
calvinxiaocao.github.io