Introduction to computing C

网站：
https://calvinxiaocao.github.io/#/teaching/fa24/C/IntroToComputingC

Introduction to computing C

2024-2025 Fall Final Mini-lecture

国关计概期末辅导活动

Something about ME!

曹彧 · Calvin Cao

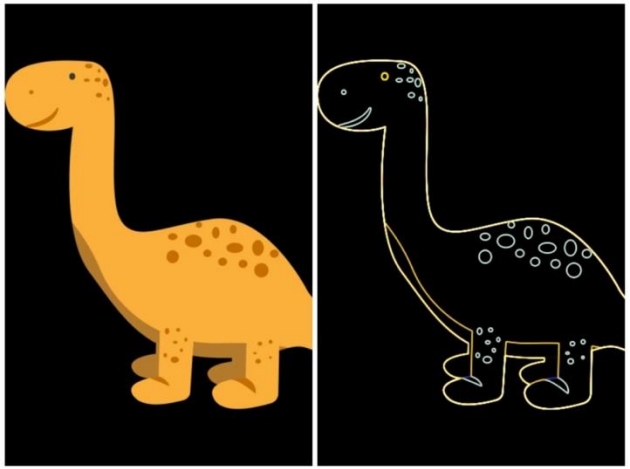
信息科学技术学院 2023级 本科生

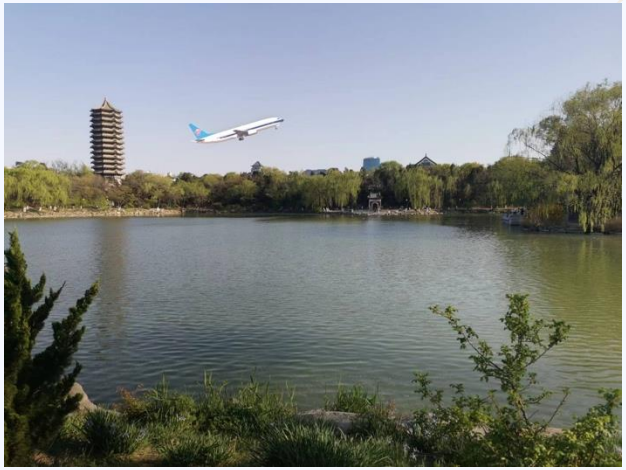
目前学习AI方向，对计算机视觉、图形学领域感兴趣
（高中无竞赛和编程基础）

My projects … (Python based)

Image blending (CV)

My projects … (C based)





Computer Graphics (VCL)

Something about The Activity!

Web

Table of contents 目录

Before all… Some suggestions!

1. Open your laptops, and **Enjoy 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. **大作业要Start early!** 不要压到DDL
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

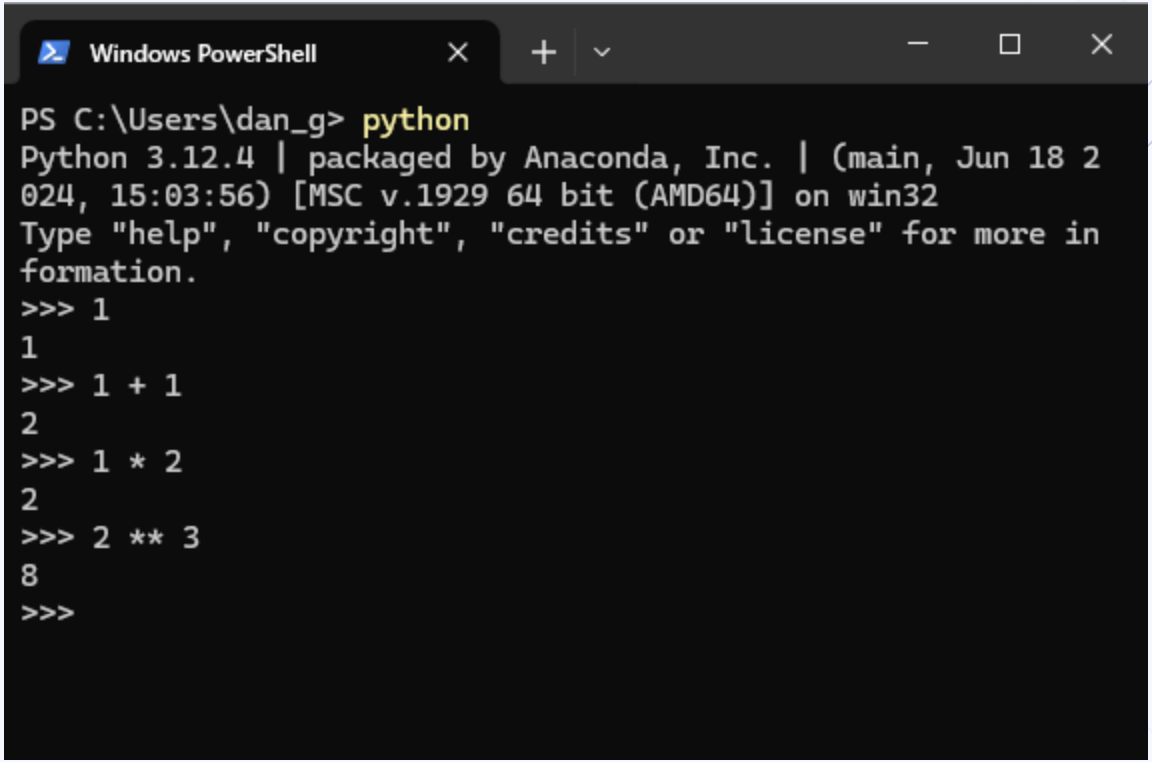
Leading Part

C与Python的断点调试法

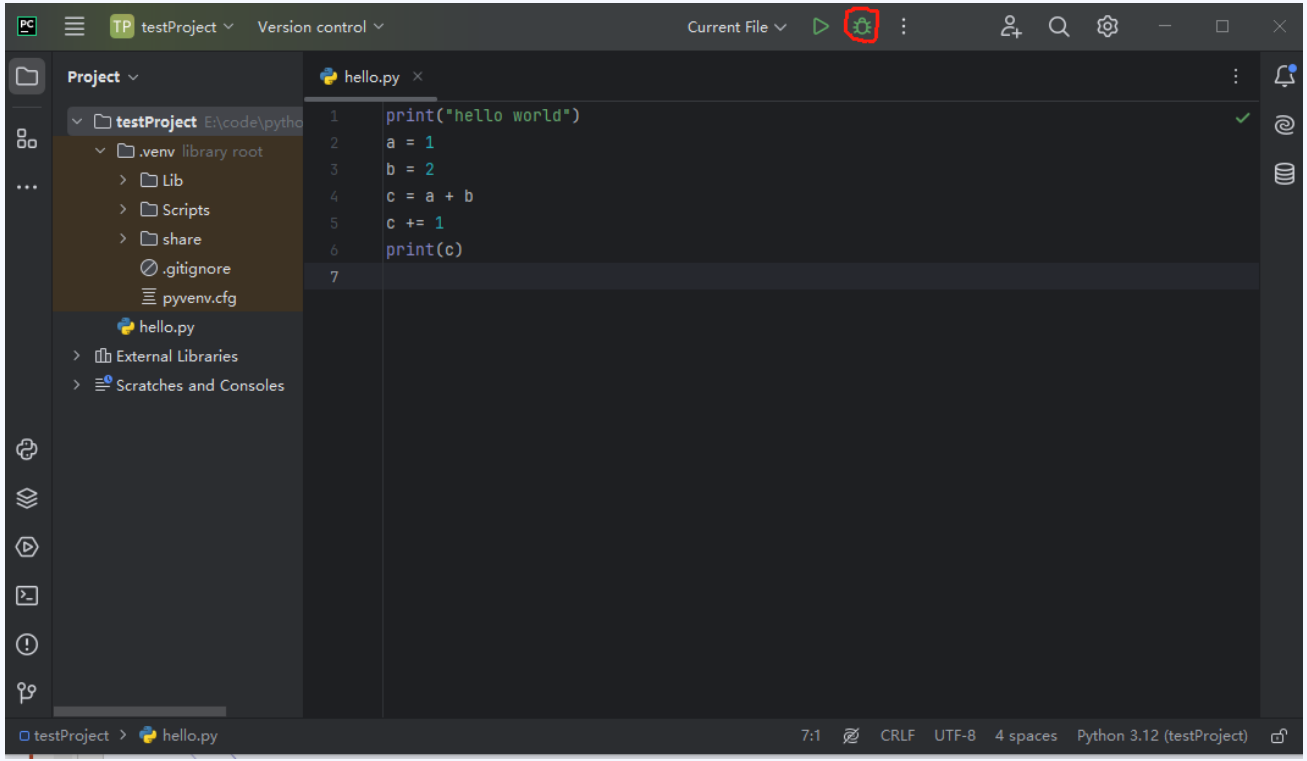
Python interactive mode（命令行）

(Demo here)

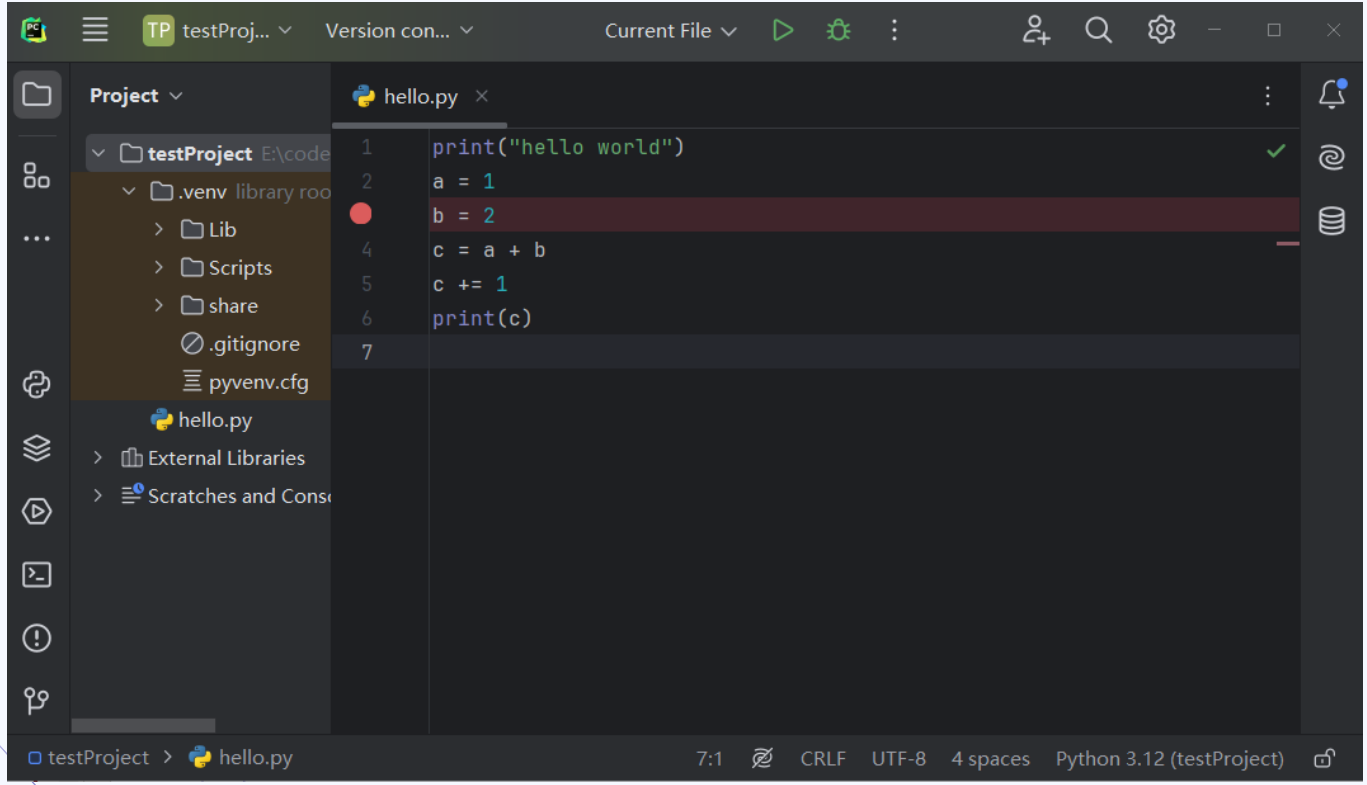
> help(func)



Debugging Mode (DEMO!)

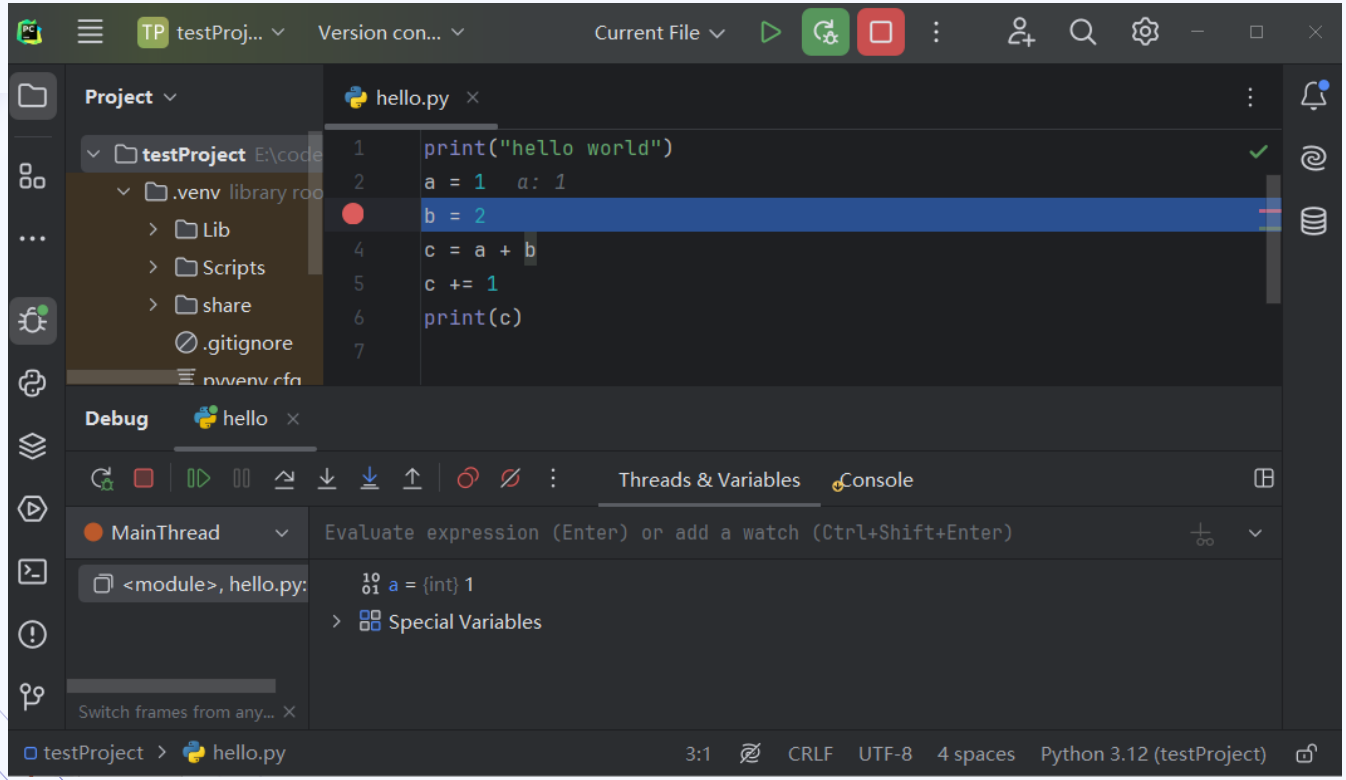


Pycharm Debugging Mode



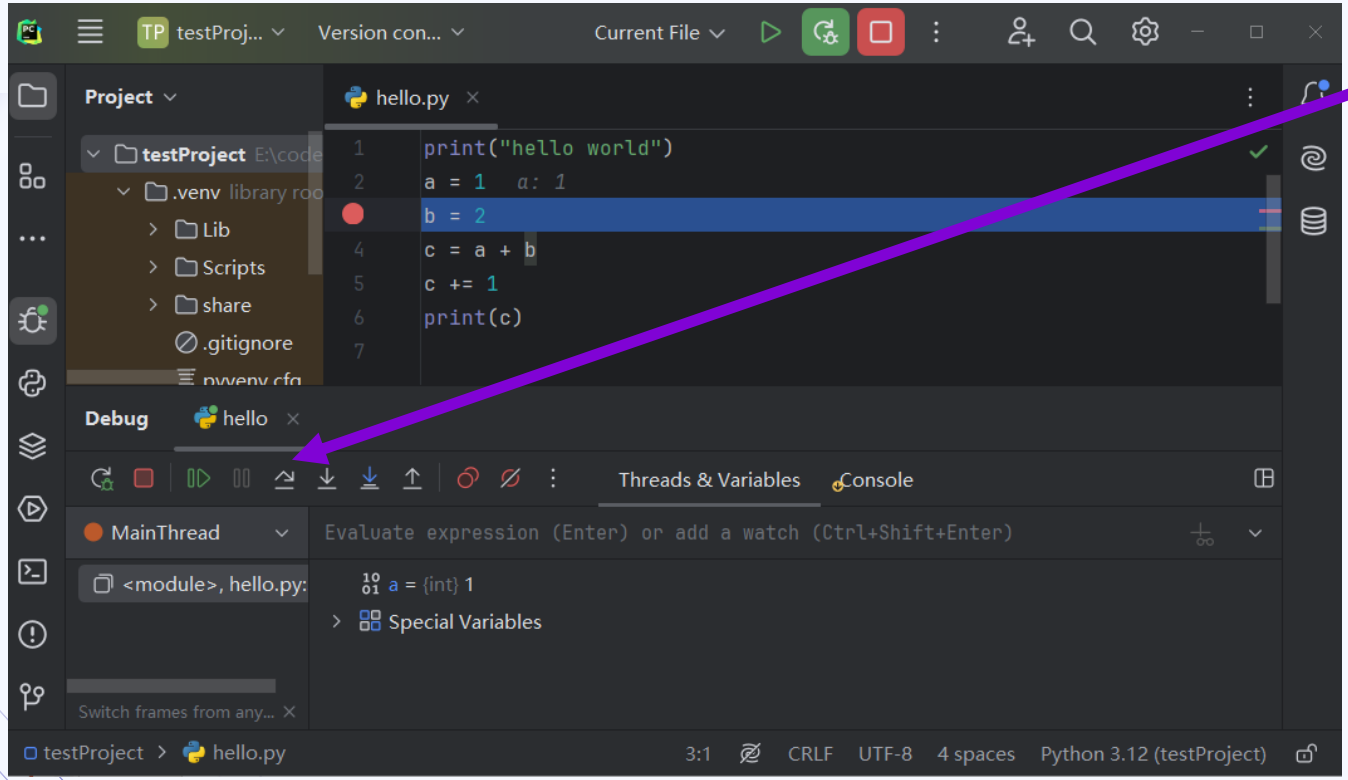
断点
Breakpoint

Pycharm Debugging Mode



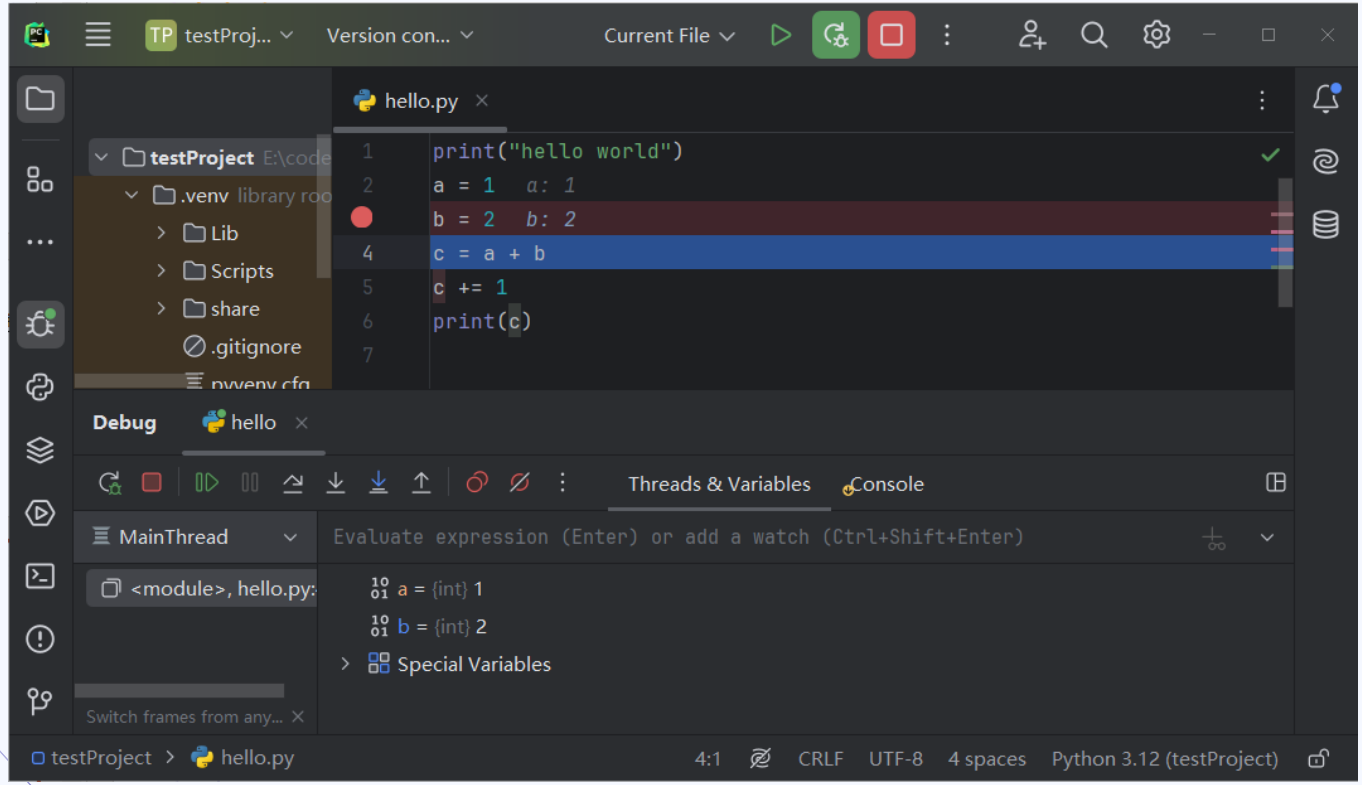
程序开始debug模式后，会自动跳转到第一个断点处暂停

Pycharm Debugging Mode

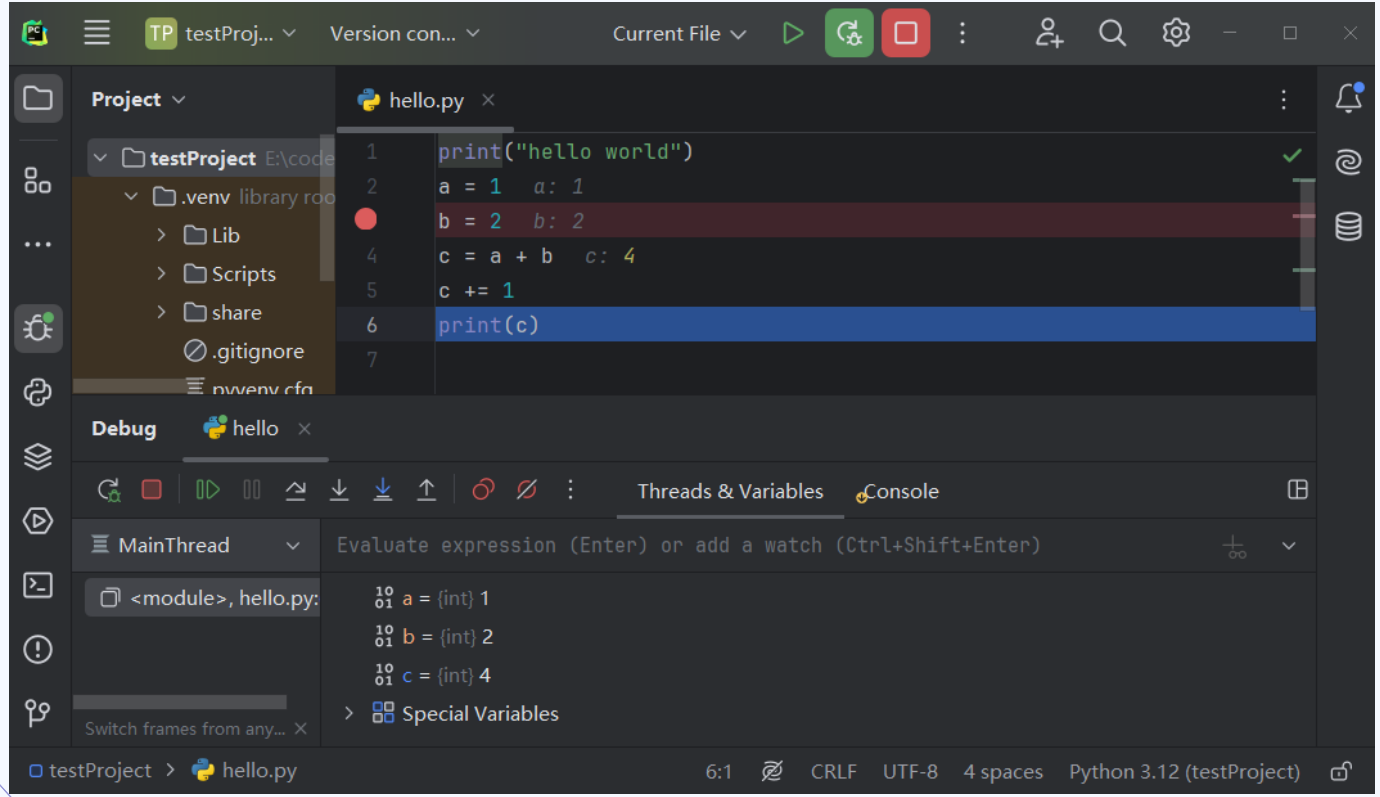


点这个按钮，会跳转到下一步

Pycharm Debugging Mode



Pycharm Debugging Mode

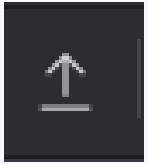


Debugging Mode

到下一个断点（或结束）

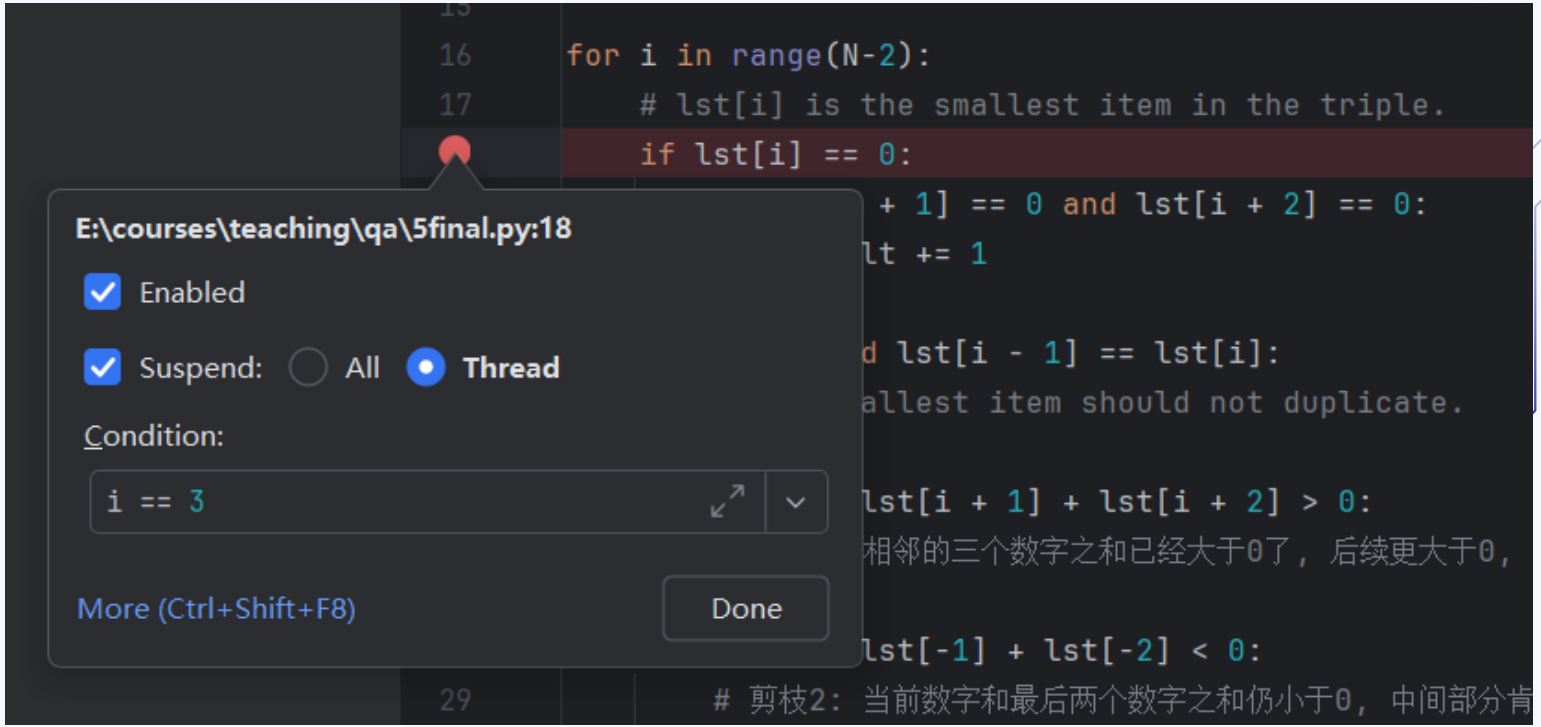
向下执行一步（遇到函数则进入）

向下执行一步



跳出当前函数

Conditional Breakpoint



Several Common Error… Annoying

8	Time Limit Exceeded	2024-12-17
7	Wrong Answer	2024-12-17
6	Memory Limit Exceeded	2024-12-16
5	Memory Limit Exceeded	2024-12-16
4	Wrong Answer	2024-12-16
3	Wrong Answer	2024-12-16
2	Compile Error	2024-12-16
1	Memory Limit Exceeded	2024-12-16

#	结果	时间
16	Accepted	2024-12-17
15	Time Limit Exceeded	2024-12-17
14	Wrong Answer	2024-12-17
13	Time Limit Exceeded	2024-12-17
12	Time Limit Exceeded	2024-12-17
11	Time Limit Exceeded	2024-12-17
10	Time Limit Exceeded	2024-12-17
9	Runtime Error	2024-12-17

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!)

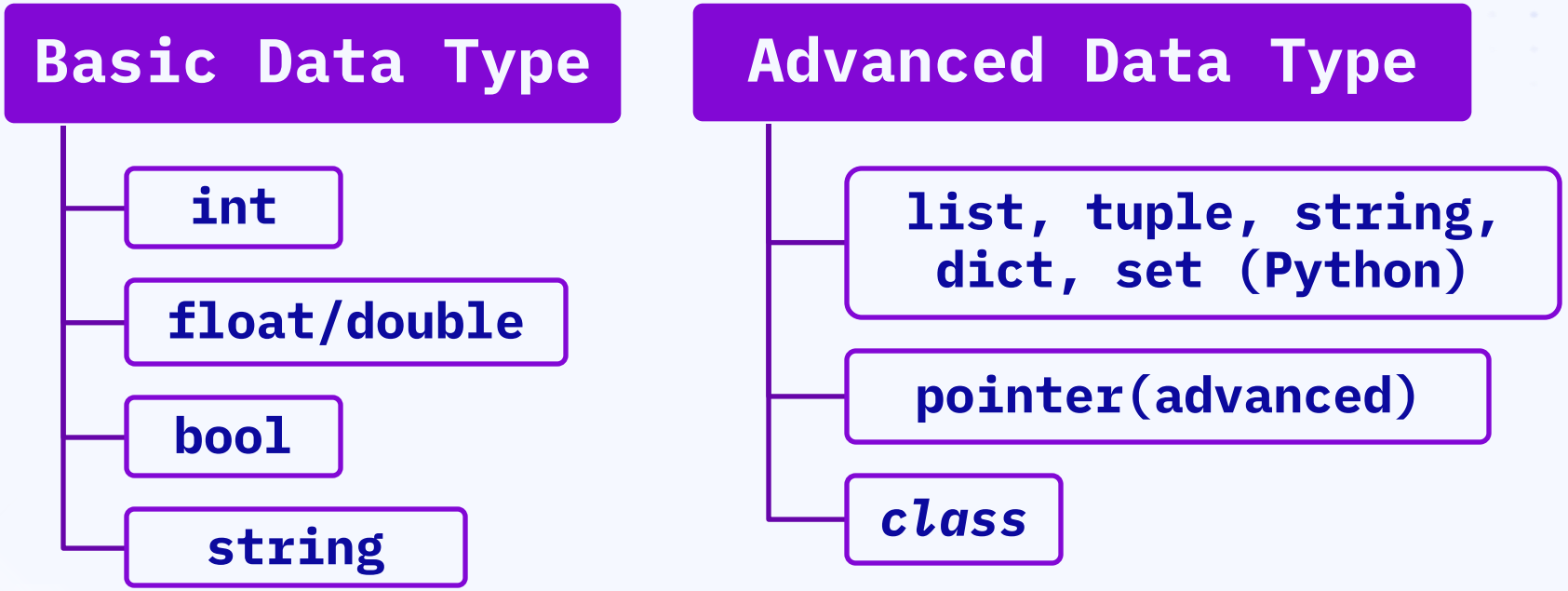
Without Abstraction?

The Code Would Be Like This!

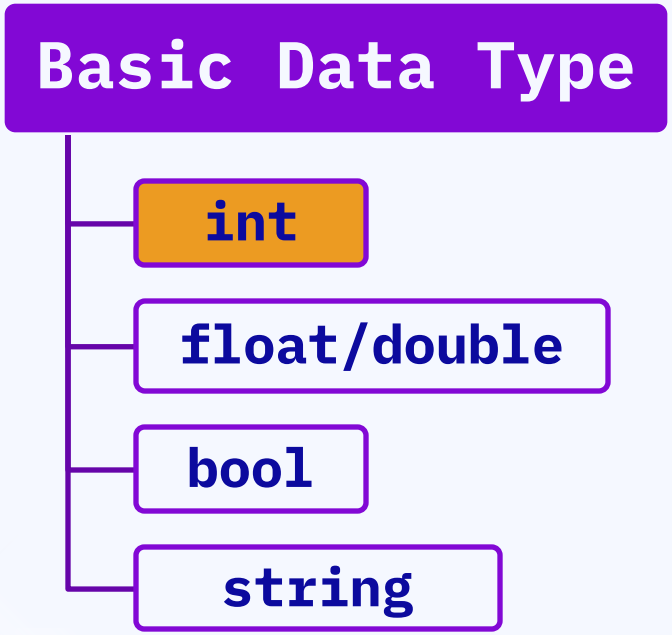
```
466  0000000000002678 <abracadabra>:
467      2678:   f3 0f 1e fa             endbr64
468      267c:   48 81 ec 98 00 00 00    sub    $0x98,%rsp
469      2683:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
470      268a:   00 00
471      268c:   48 89 84 24 88 00 00    mov    %rax,0x88(%rsp)
472      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



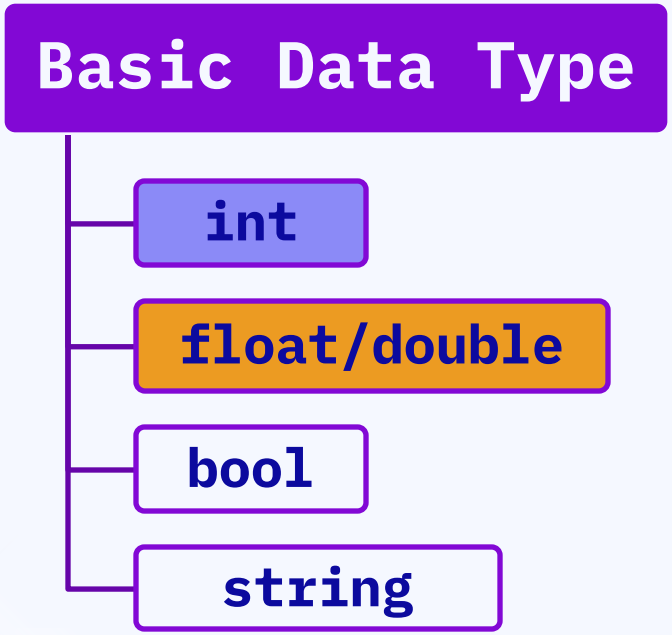
2.1 Abstraction with data



在python中的int不用考虑长度

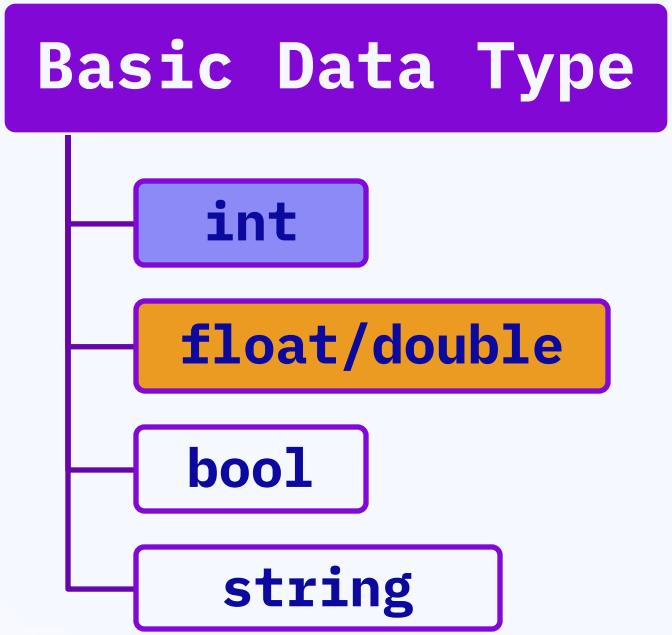
最基本的数据类型

2.1 Abstraction with data



NOTE: Python中“/”代表除，
“//”代表整除！

2.1 Abstraction with data



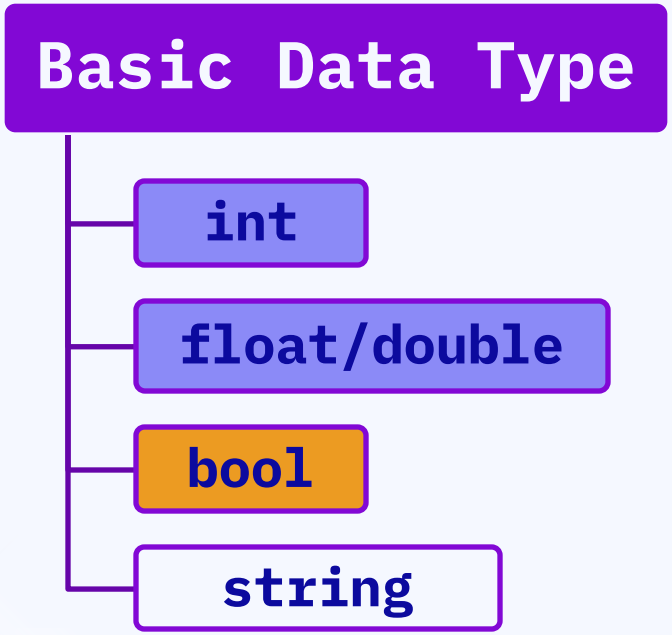
Test: 3 / 2 * 2 = ?

(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



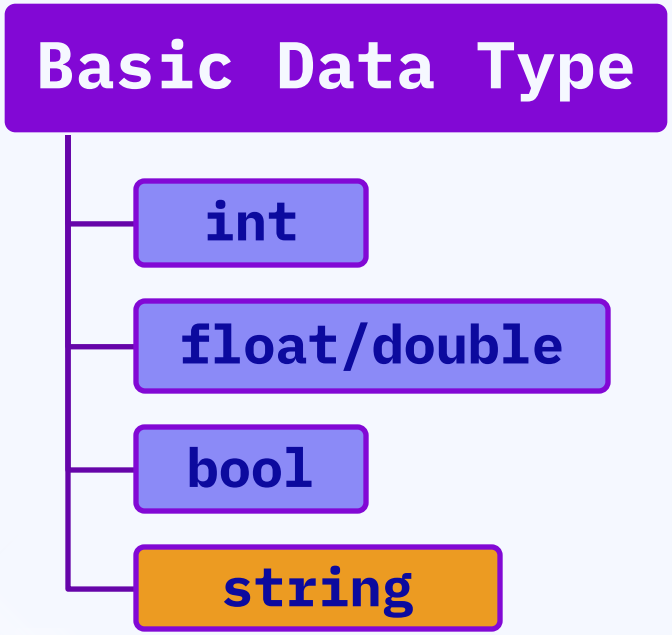
and和or的“短路运算”

expr1 ||(or) expr2

expr1 &&(and) expr2

(用途：检查空指针)

2.1 Abstraction with data



string（比较特殊的类型）

支持求长度、求子串等操作

支持遍历

“正则表达式！”

2.1 Abstraction with data

Basic Data Type

- int
- float/double
- bool
- string

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

2.1 Abstraction with data

Python中的“序列”：

(list, tuple, string, range…)

Advanced Data Type

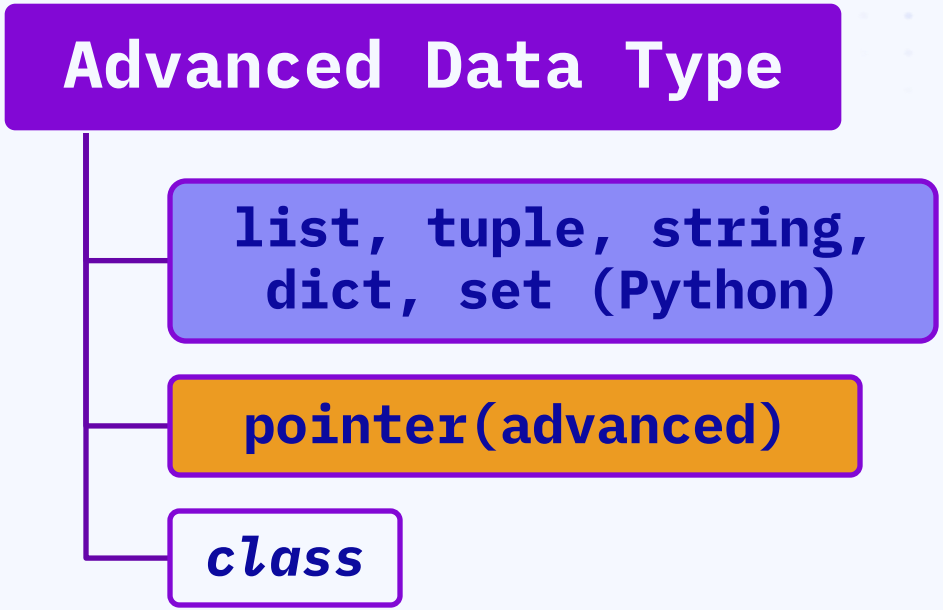
- list, tuple, string, dict, set (Python)
- pointer(advanced)
- *class*

2.1 Abstraction with data

指针pointer:

Python中，“指针”是被隐藏起来的！

Environment Diagram



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

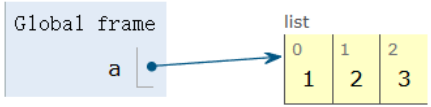
NEW: follow our YouTube, TikTok, and Instagram for free tutorials

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

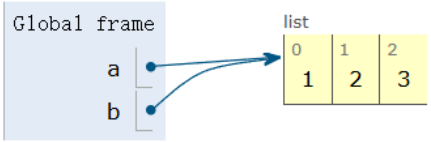
NEW: follow our YouTube, TikTok, and Instagram for free tutorials

Get AI Help

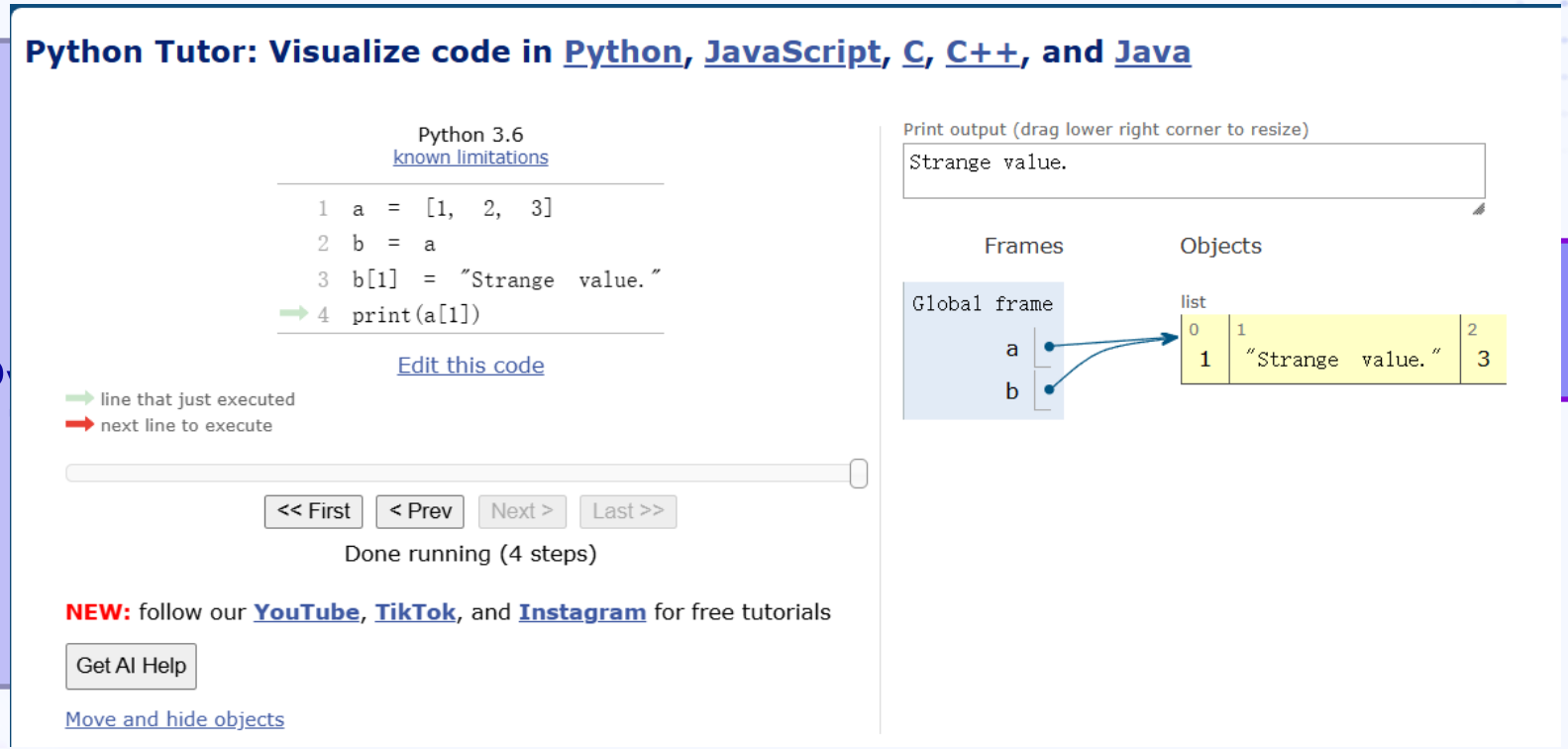
Move and hide objects

Print output (drag lower right corner to resize)

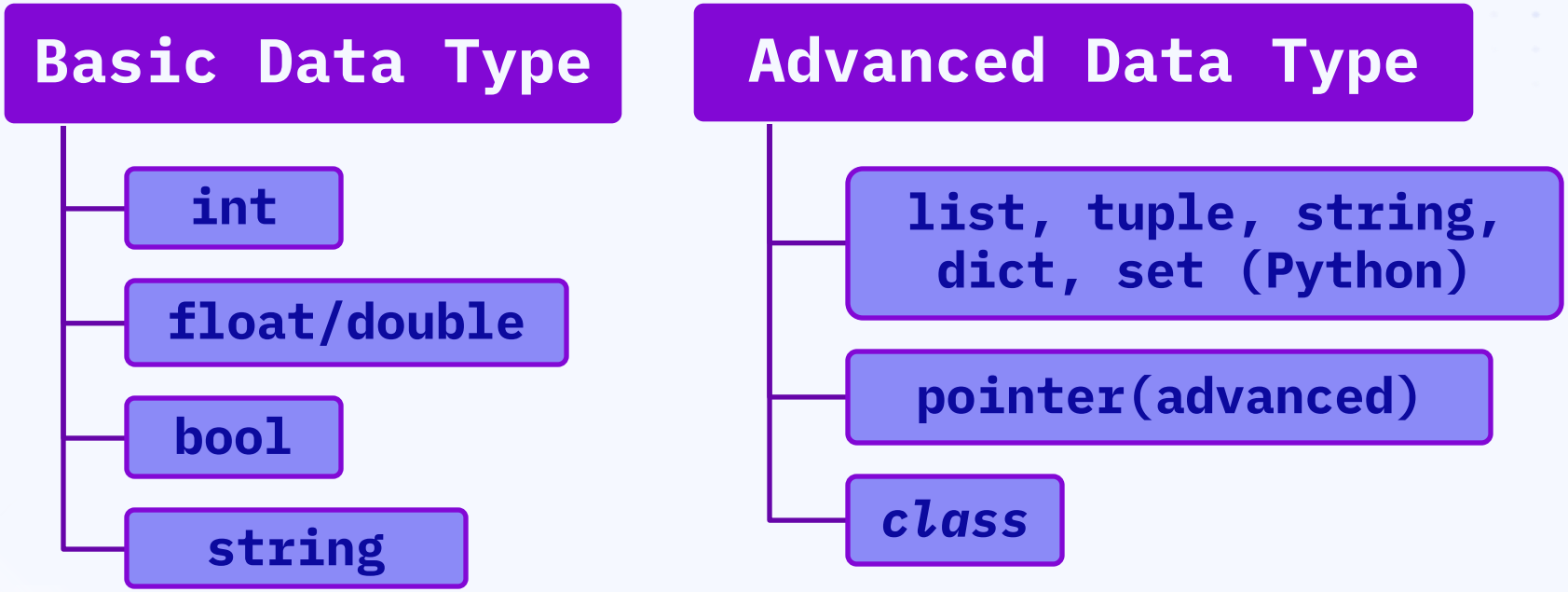
Frames Objects



2.1 Abstraction with data

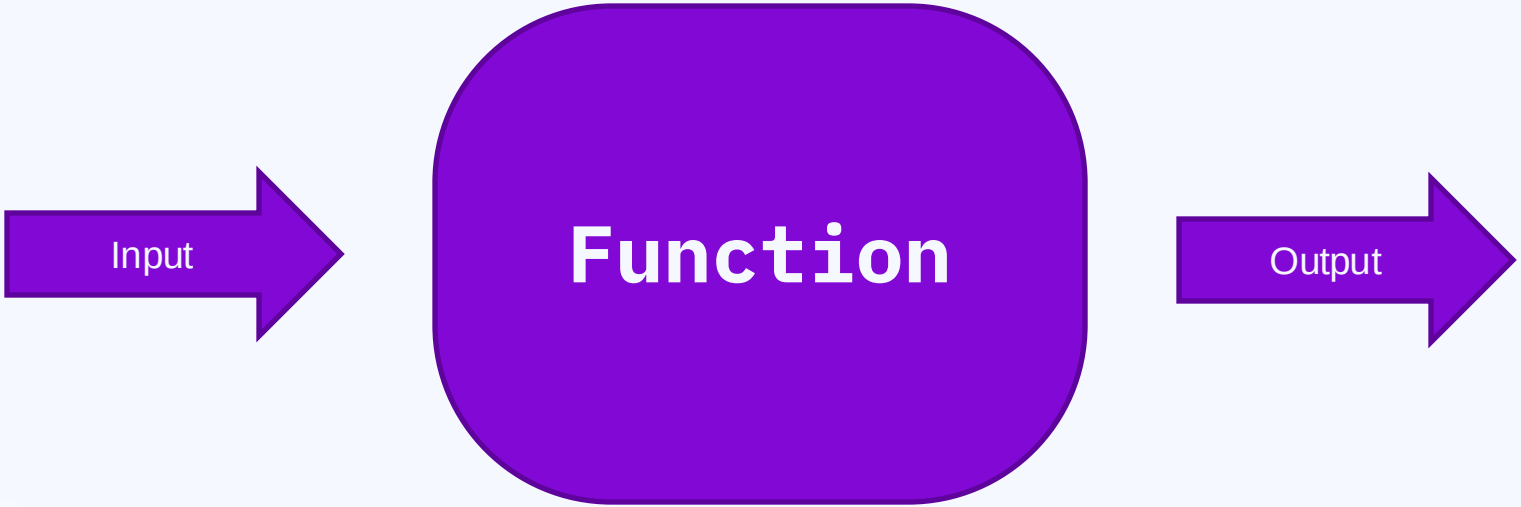


2.1 Abstraction with data



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)

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

函数：“实现特定功能的代码块”

函数的定义?

使用**def <function-name>: …**

在定义之前，想好函数的“输入”是什么，“输出”又是什么

2.2 Abstraction with function

函数：“实现特定功能的代码块”

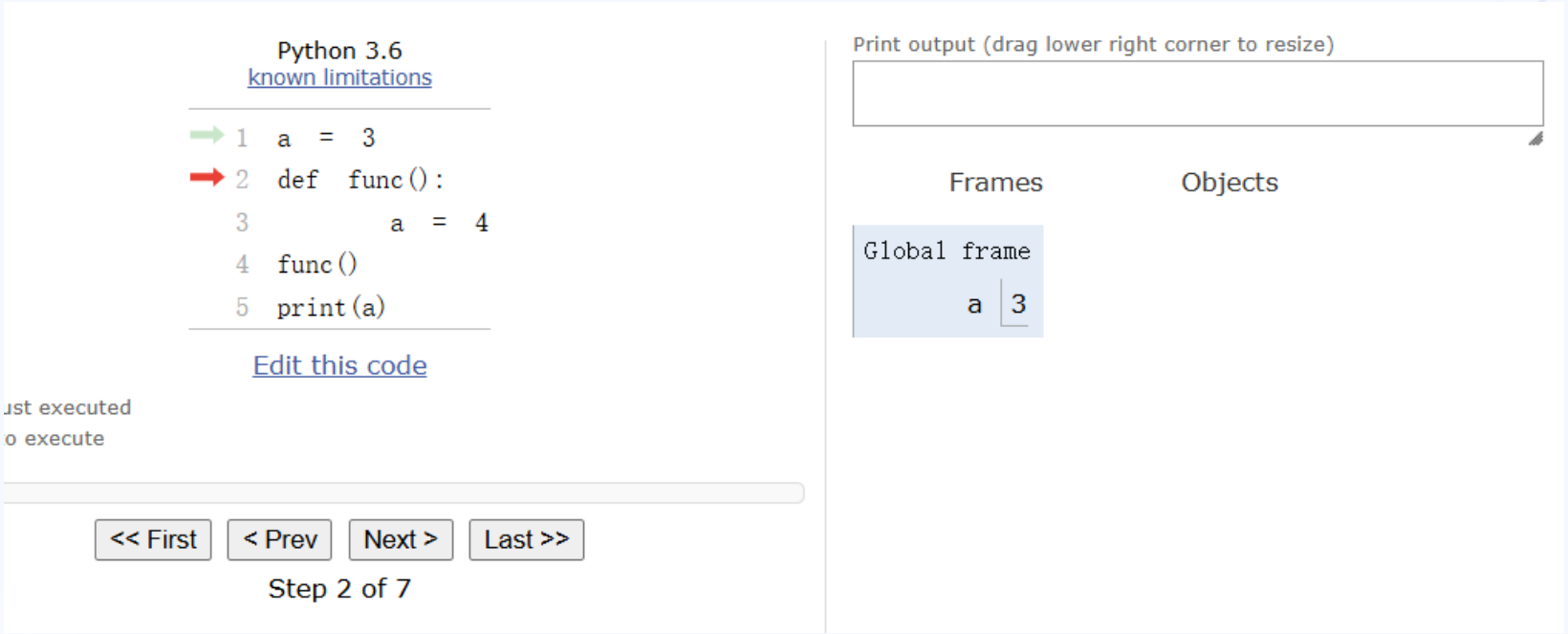
函数的命名域

右侧这段代码执行后，

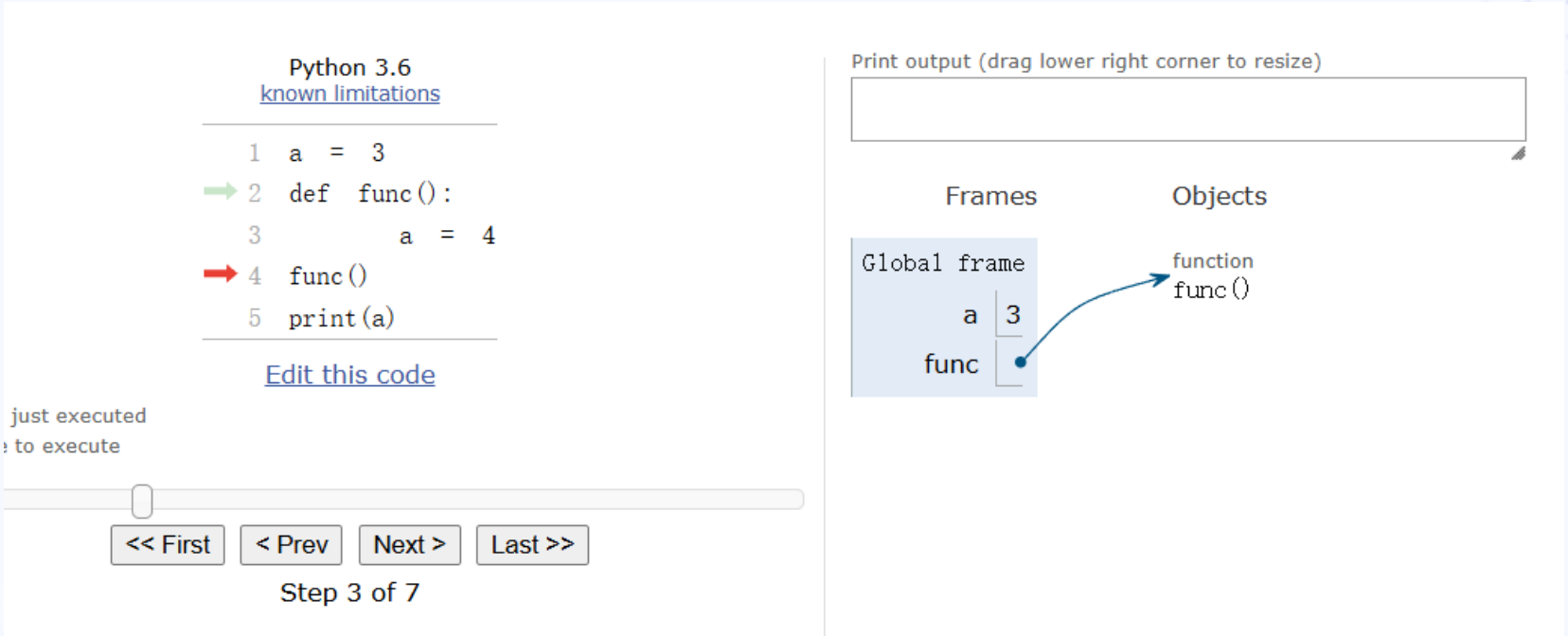
会输出什么？

```
1  a = 3
2  def func():
3      a = 4
4  func()
5  print(a)
6
```

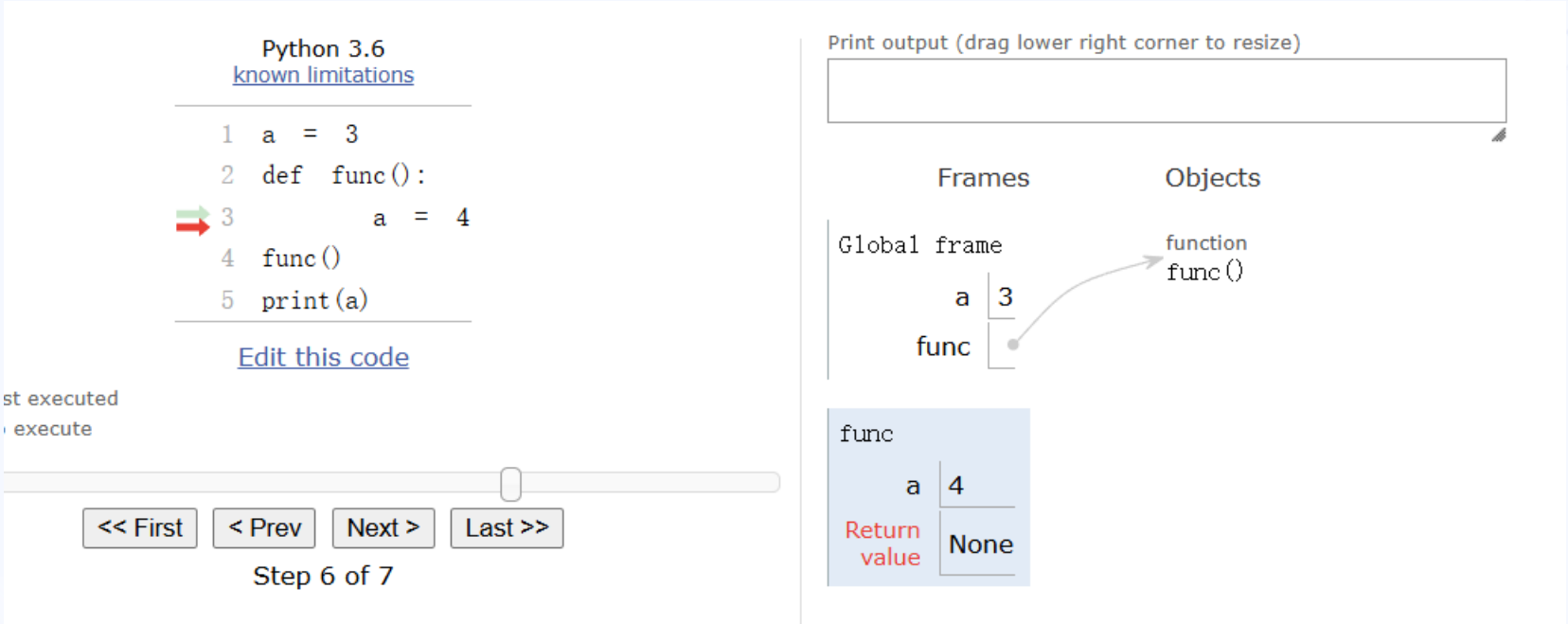
2.2 Abstraction with function



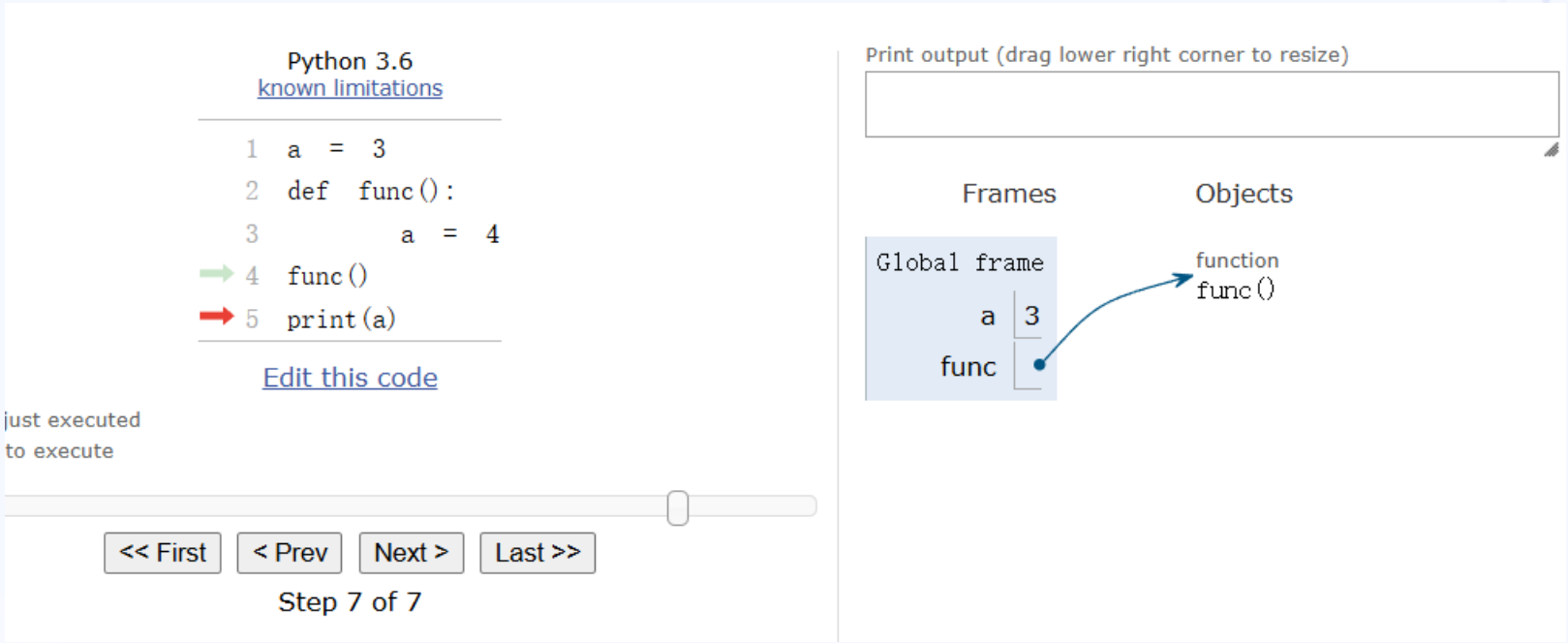
2.2 Abstraction with function



2.2 Abstraction with function



2.2 Abstraction with function



2.2 Abstraction with function

2.2 Abstraction with function

函数：“实现特定功能的代码块”

函数的命名域

(somehow tricky…)

右侧这段代码执行后，

会输出什么？

```
1  a = [1, 2, 3]
2  def func():
3      a[1] = 4
4  func()
5  print(a[1])
```

2.2 Abstraction with function

函数：“实现特定功能的代码块”

函数的命名域

(somehow tricky…)

右侧这段代码执行后，

会输出什么？

```
1  a = [1, 2, 3]
2  def func():
3      a = [4, 5, 6]
4      a[1] = 4
5  func()
6  print(a[1])
7
```

2.2 Abstraction with function

函数：“实现特定功能的代码块”

函数的命名域

(somehow tricky…)

右侧这段代码执行后，

会输出什么？

```
1  a = [1, 2, 3]
2  def func():
3      a = [4, 5, 6]
4      a[1] = 4
5  func()
6  print(a[1])
7
```

2.2 Abstraction with function

函数：“实现特定功能的代码块”

Conclusion:

1. 变量的命名很重要！！小心隐形的**bug**
2. 如果遇到笔试题出这类问题，记得分析变量的“作用域”
3. 除非有十足的把握，否则不要在函数内尝试改变函数外的变量

2.2 Abstraction with function

18. 以下程序的输出是_____。

```
x = 20
def f(a, b):
    x = b[0]
    b[0] = a
    a = x
c,d = 100,[1,2]
f(c, d)
print(x, c, d[0])
```

A) 1 100 100　　B) 20 100 100　　C) 20 100 1　　D) 1 100 1

2.2 Abstraction with function

18. 以下程序的输出是_____。

```
x = 20
def f(a, b):
    x = b[0]
    b[0] = a
    a = x
c,d = 100,[1,2]
f(c, d)
print(x, c, d[0])
```

A) 1 100 100　　B) 20 100 100　　C) 20 100 1　　D) 1 100 1

20 100 100

2.2 Abstraction with function

20. 以下程序的输出是_____。

```
def append_a(lst):
  t = lst
  t.append('a')
  return t
a = ['x','y','z']
b = append_a(a)
print(a, b)
```

A) ['x','y','z'] ['x','y','z']　　B) ['x','y','z','a'] ['x','y','z','a']

C) ['x','y','z'] ['x','y','z','a']　　D) ['x','y','z','a'] ['x','y','z']

2.2 Abstraction with function

20. 以下程序的输出是_____。

```
def append_a(lst):
  t = lst
  t.append('a')
  return t
a = ['x','y','z']
b = append_a(a)
print(a, b)
```

B

A) ['x','y','z'] ['x','y','z']　　B) ['x','y','z','a'] ['x','y','z','a']

C) ['x','y','z'] ['x','y','z','a']　　D) ['x','y','z','a'] ['x','y','z']

2.2 Abstraction with function

19. 以下程序的输出是_____。

```
def myfunc(x, y):
    x = x + 100
    y = y + 100
    z = x + y
    return z
x = 1
y = 2
z = x + y
z = myfunc(x, y)
print(x, y, z, sep=",")
```

域”

数外的

2.2 Abstraction with function

19. 以下程序的输出是_____。

```
def myfunc(x, y):
    x = x + 100
    y = y + 100
    z = x + y
    return z
x = 1
y = 2
z = x + y
z = myfunc(x, y)
print(x, y, z, sep=",")
```

1,2,203(中间没有空格！）

域”

数外的

2.2 Abstraction with function

20. 以下程序的输出是_____

```
out_list = []
x = 1
def f(x):
    x = x+1
    out_list.append(x)
def g(y):
    global x
    x = x+1
    out_list.append(x)
    out_list.append(y)
f(x)
g(x)
f(x)
print(out_list)
```

函

C

1.

2

3

“作用域”

变函数外的

2.2 Abstraction with function

20. 以下程序的输出是_____

```
out_list = []
x = 1
def f(x):
    x = x+1
    out_list.append(x)
def g(y):
    global x
    x = x+1
    out_list.append(x)
    out_list.append(y)
f(x)
g(x)
f(x)
print(out_list)
```

[2, 2, 1, 3]

函

C

1.

2

3

“作用域”

变函数外的

2.2 Abstraction with function

函数：“实现特定功能的代码块”

Conclusion:

1. 变量的命名很重要！！小心隐形的**bug**
2. 如果遇到笔试题出这类问题，记得分析变量的“作用域”
3. 除非有十足的把握，否则不要在函数内尝试改变函数外的变量

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: 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

递归– 例: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

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

递归– 例: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

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

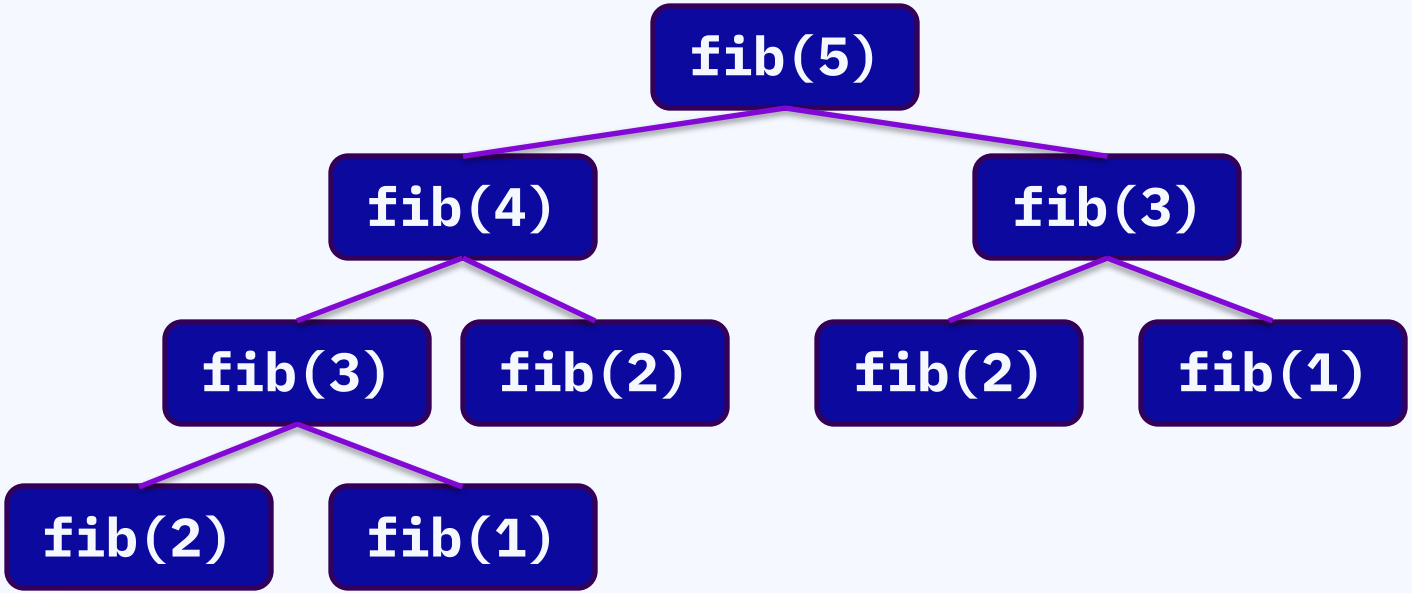
递归– 例1: 斐波那契数列(again.)

1. Base case
2. Recursive Leap of Faith

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

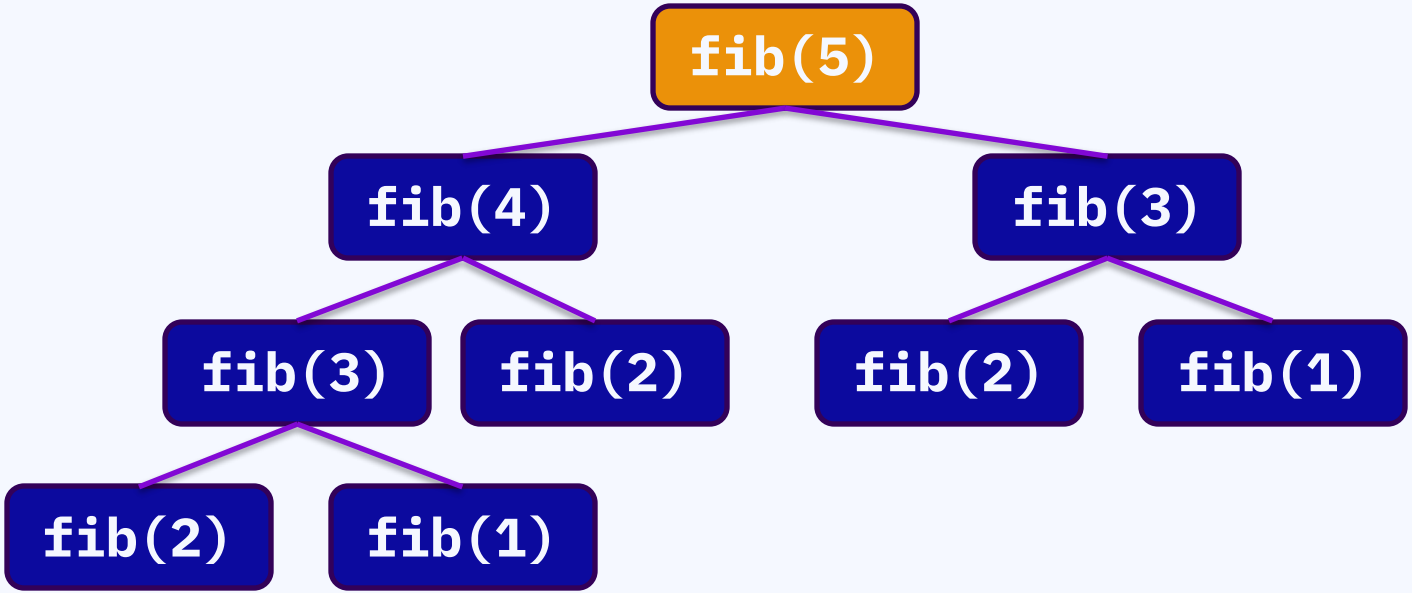
2.2 Abstraction with function

递归– 问题：重复计算！！



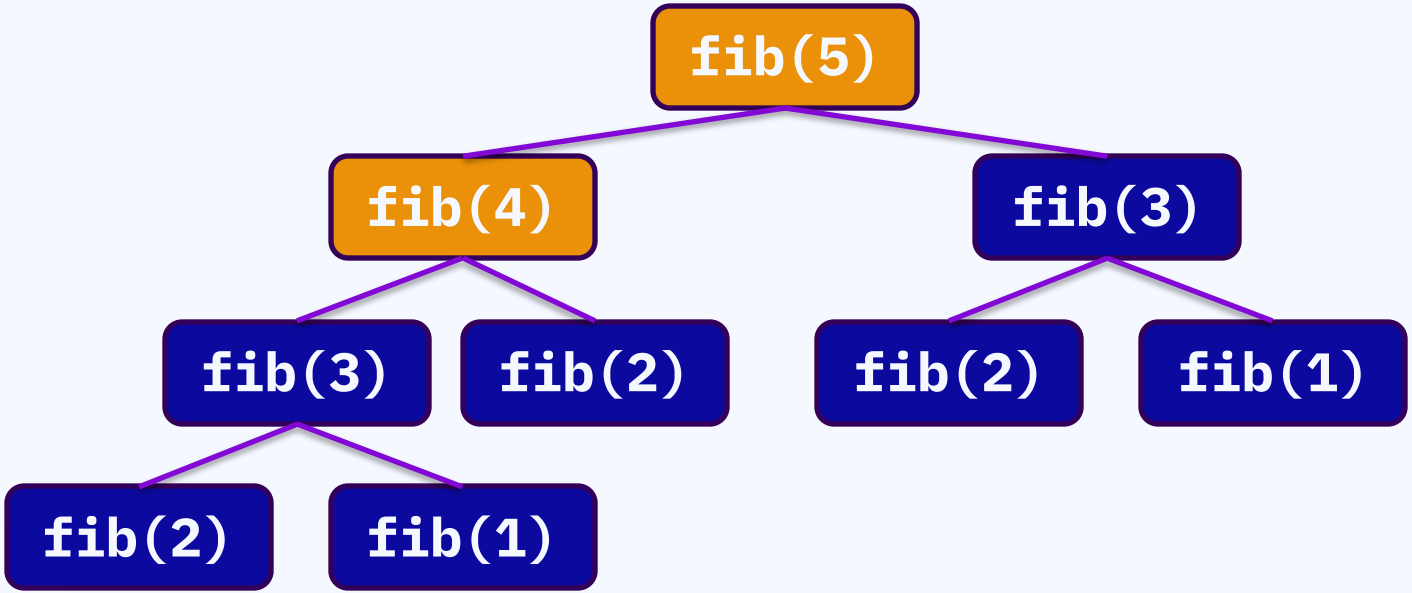
2.2 Abstraction with function

递归– 问题：重复计算！！



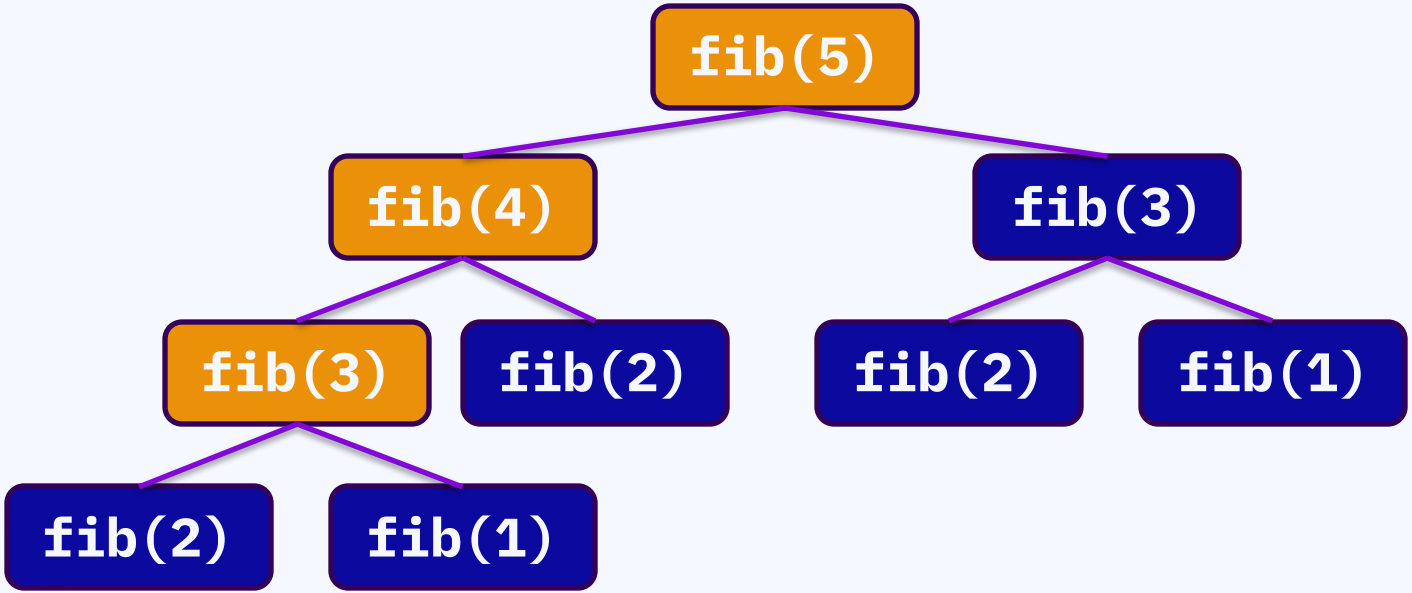
2.2 Abstraction with function

递归– 问题：重复计算！！



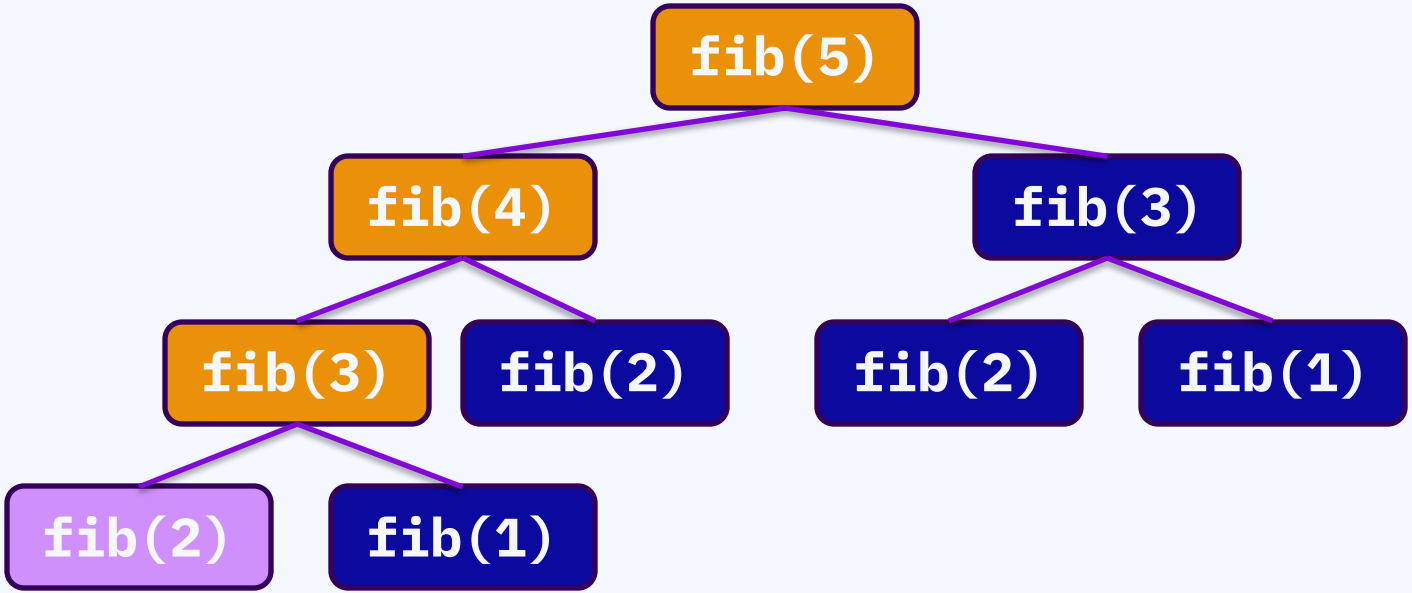
2.2 Abstraction with function

递归– 问题：重复计算！！



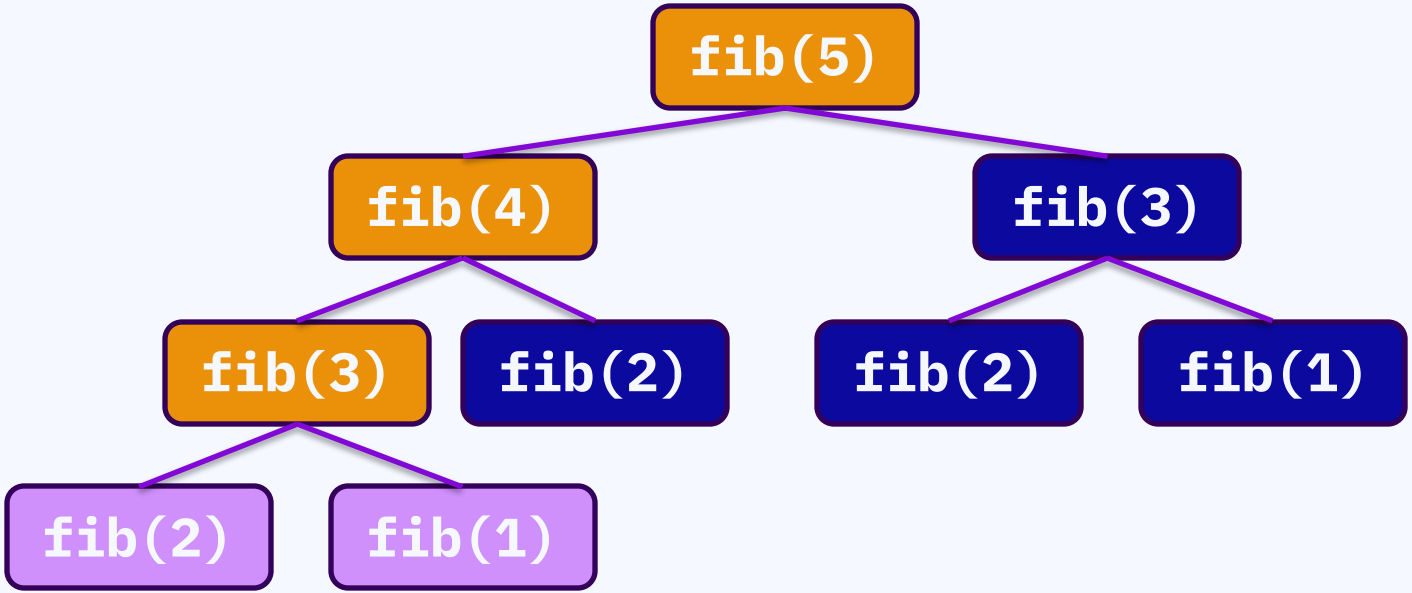
2.2 Abstraction with function

递归– 问题：重复计算！！



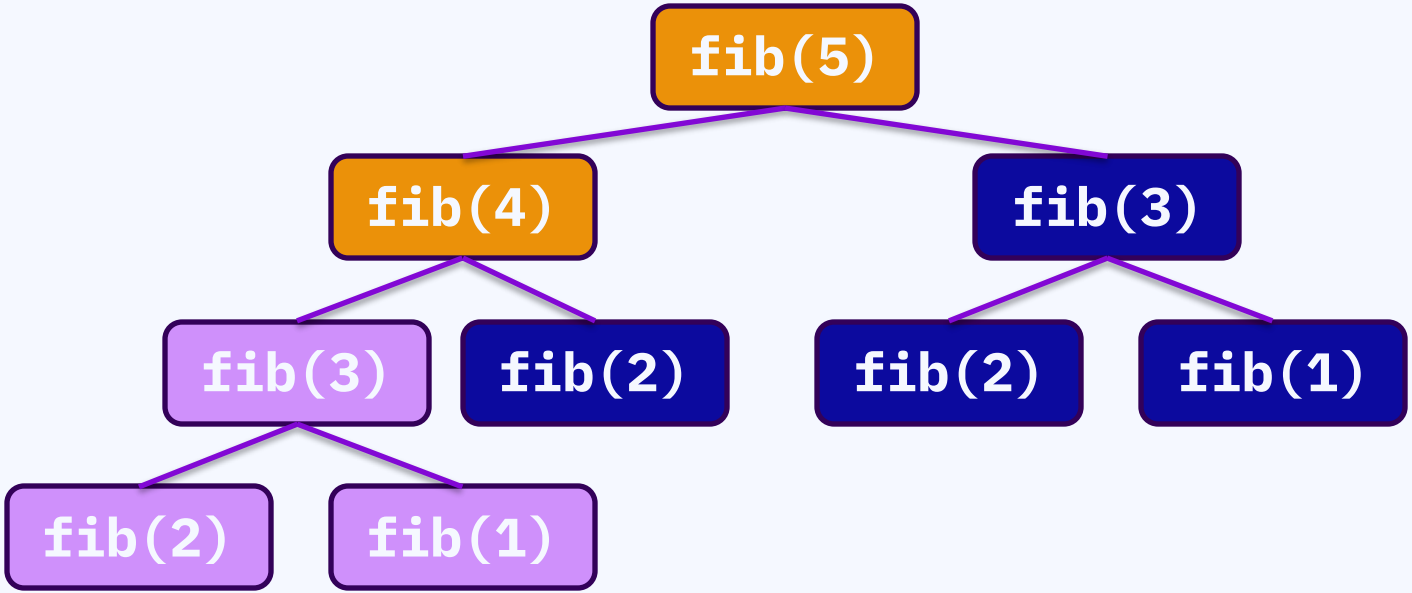
2.2 Abstraction with function

递归– 问题：重复计算！！



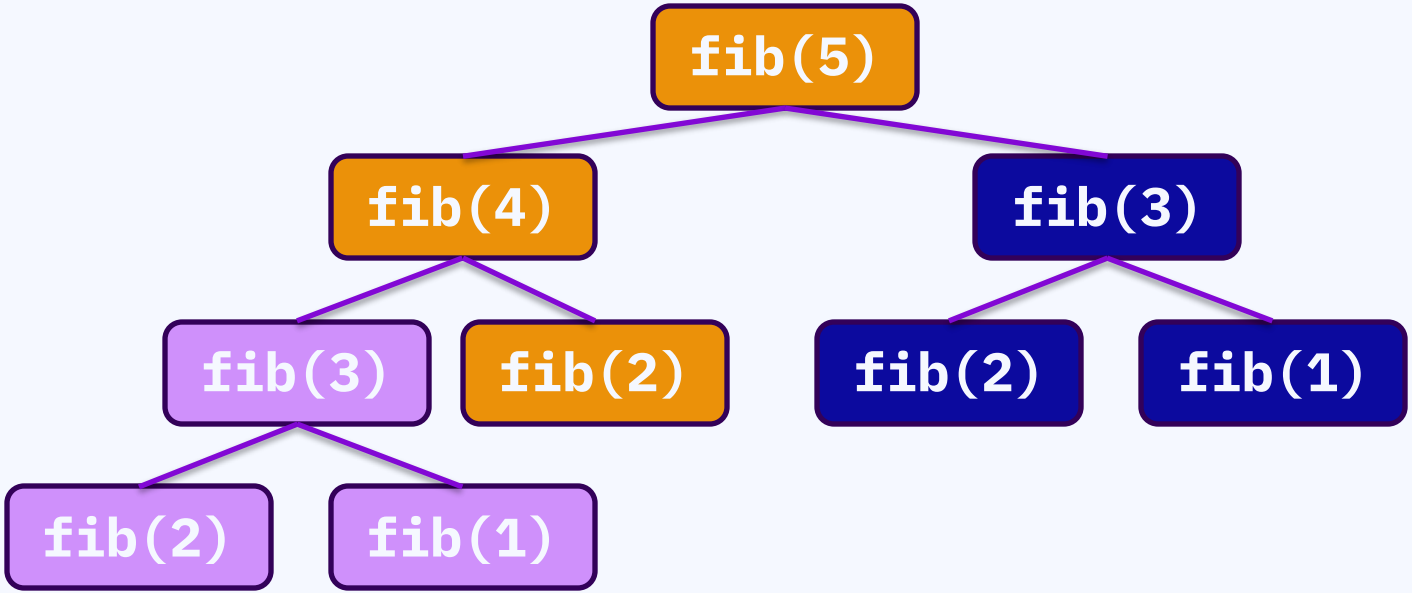
2.2 Abstraction with function

递归– 问题：重复计算！！



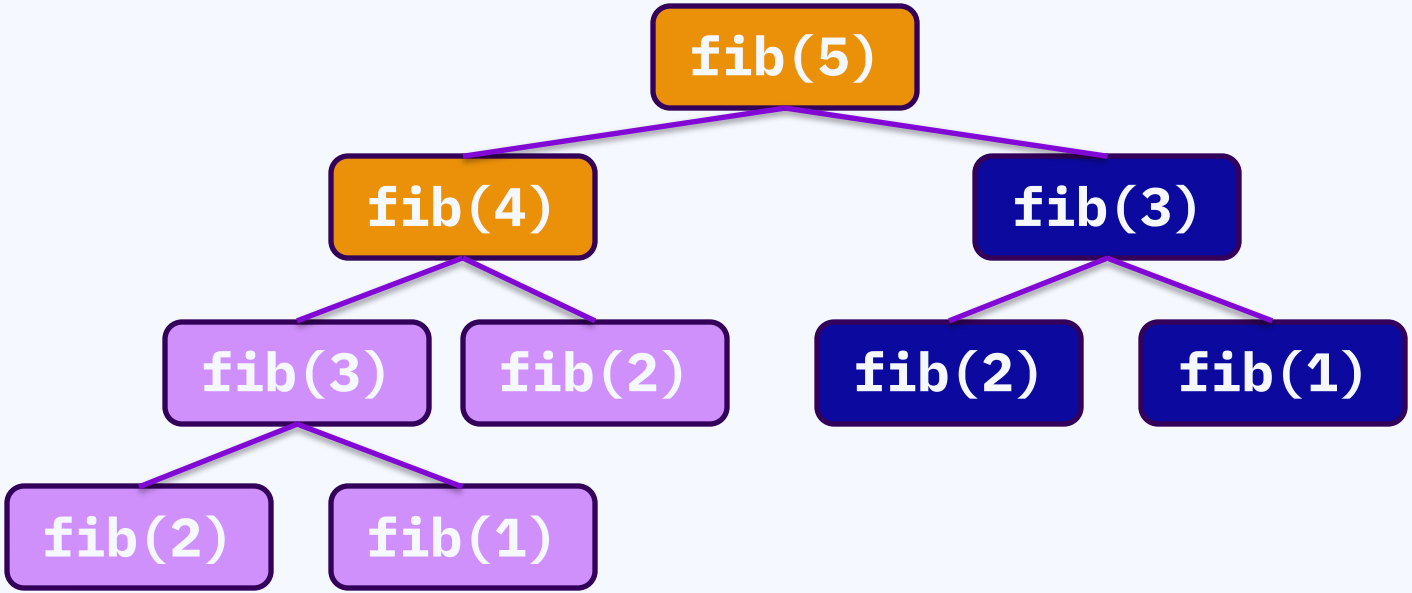
2.2 Abstraction with function

递归– 问题：重复计算！！



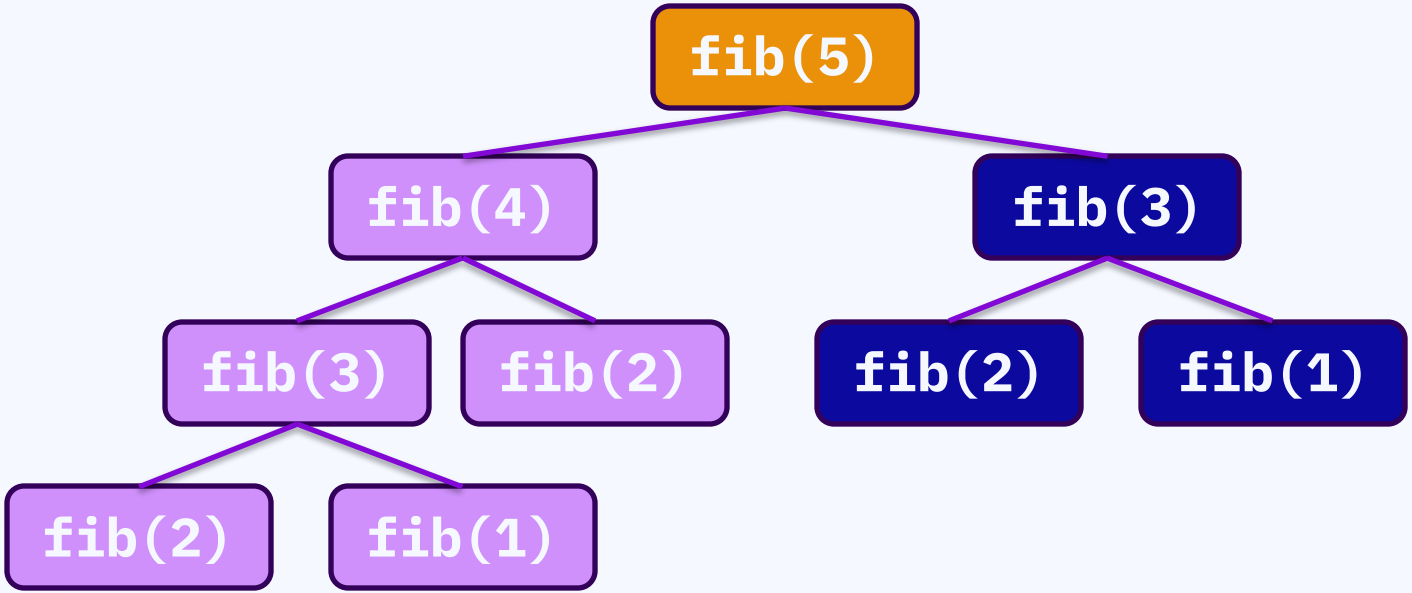
2.2 Abstraction with function

递归– 问题：重复计算！！



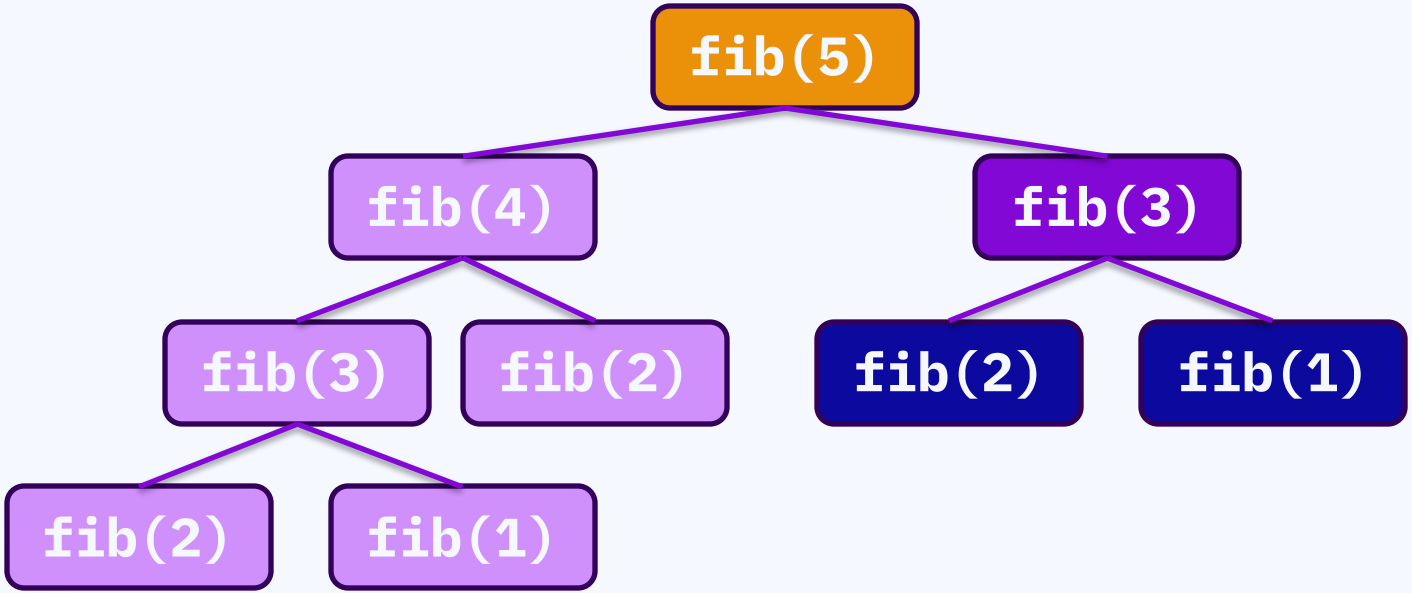
2.2 Abstraction with function

递归– 问题：重复计算！！



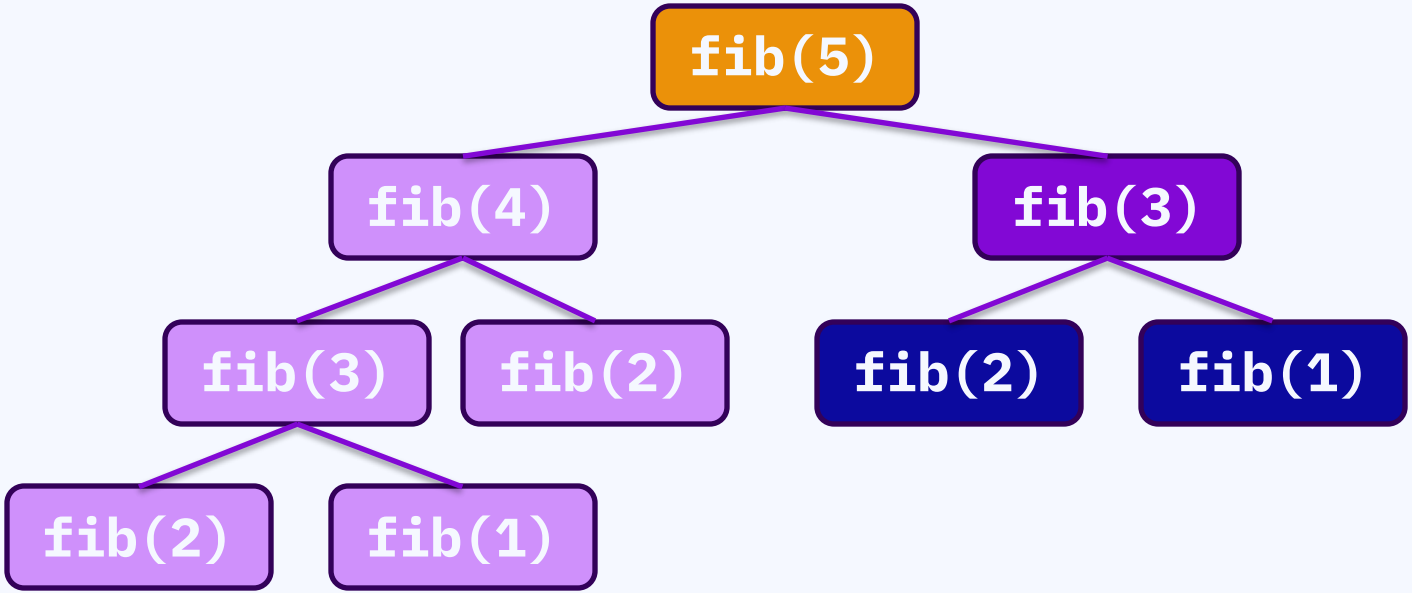
2.2 Abstraction with function

递归– 问题：重复计算！！



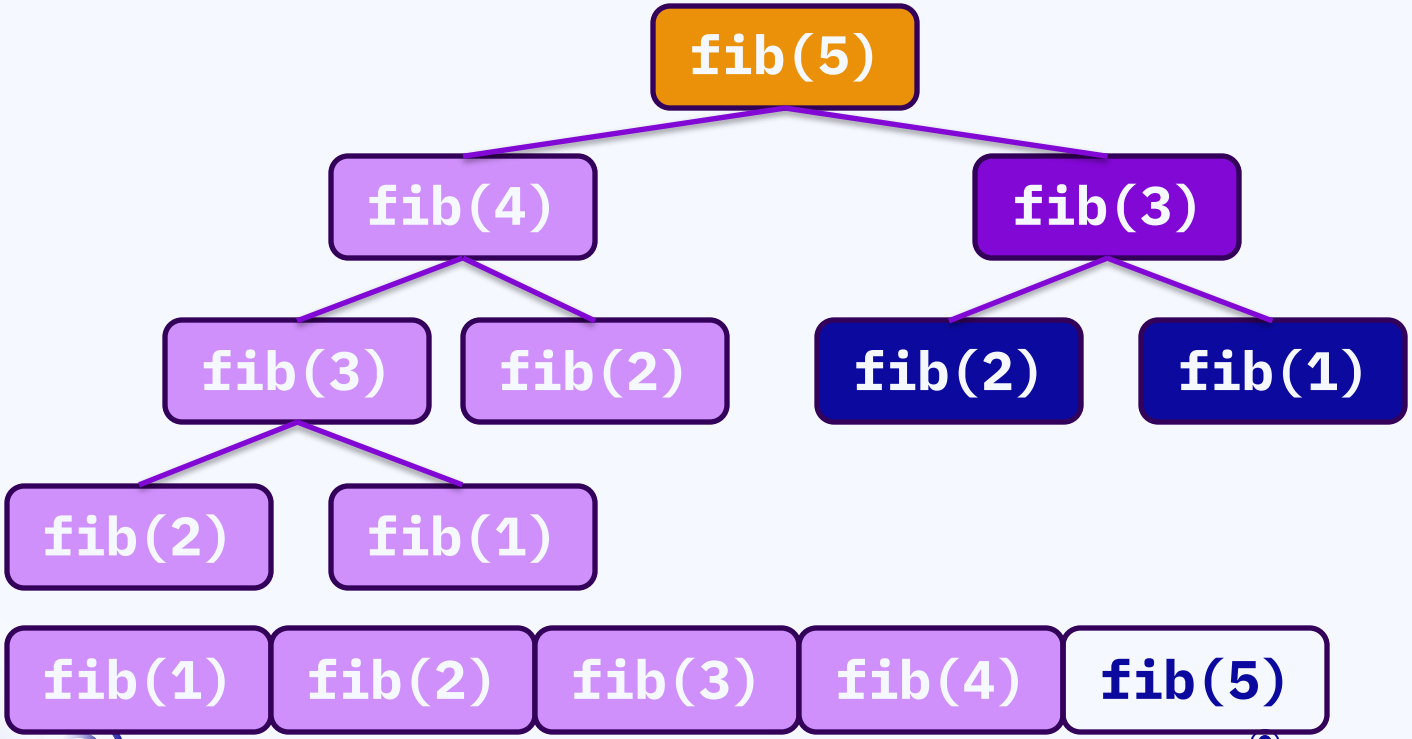
2.2 Abstraction with function

递归– 问题：fib(3)已经算过了，可还要计算一遍！！



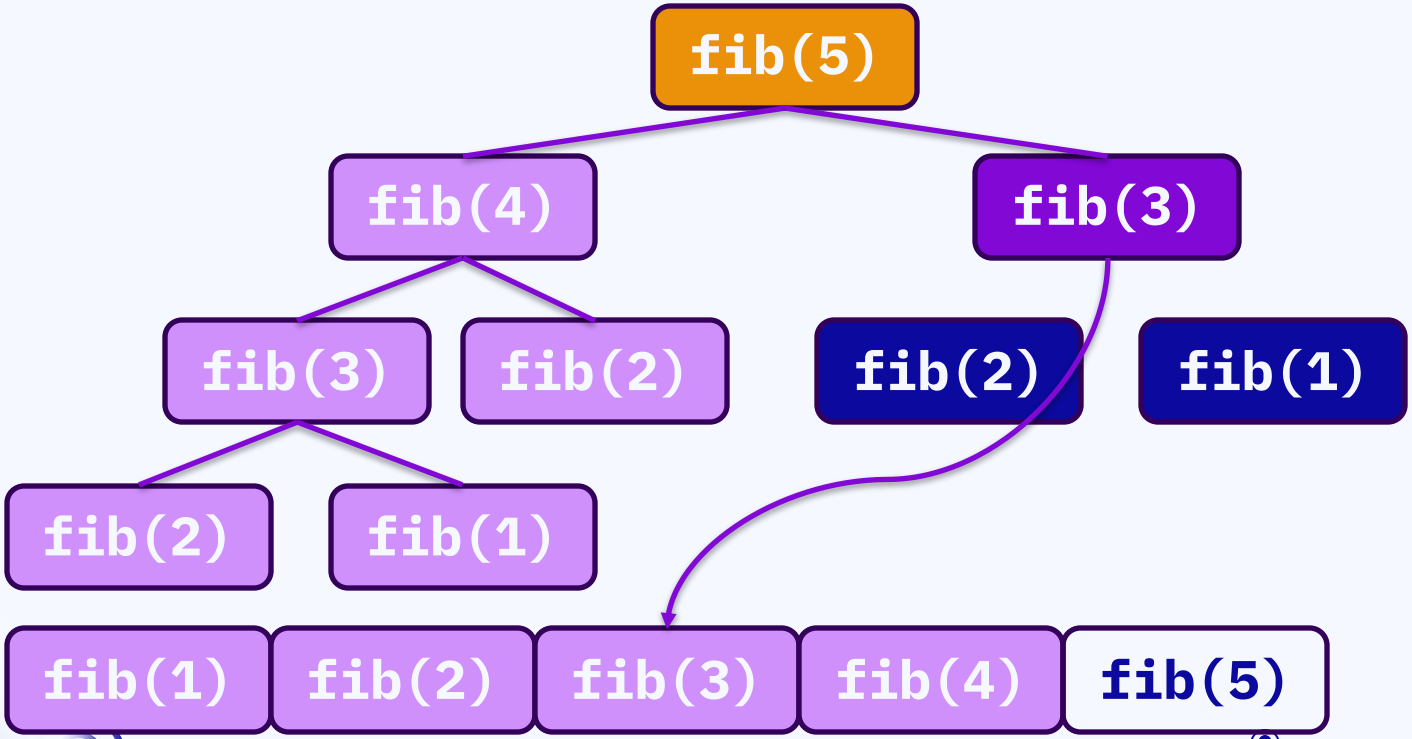
2.2 Abstraction with function

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

矩阵乘法-如何理解多维列表？

计算两个矩阵的乘法。n*m阶的矩阵A乘以m*k阶的矩阵B得到的矩阵C 是n*k阶的，且C[i][j] = A[i][0]*B[0][j] + A[i][1]*B[1][j] + ... +A[i][m-1]*B[m-1][j](C[i][j]表示C矩阵中第i行第j列元素)。

输入

第一行为n, m, k，表示A矩阵是n行m列，B矩阵是m行k列，n, m, k均小于100
然后先后输入A和B两个矩阵，A矩阵n行m列，B矩阵m行k列，矩阵中每个元素的绝对值不会大于1000。

输出

输出矩阵C，一共n行，每行k个整数，整数之间以一个空格分开。

样例输入

```
3 2 3
1 1
1 1
1 1
1 1 1
1 1 1
```

样例输出

```
2 2 2
2 2 2
2 2 2
```

矩阵乘法-如何理解多维列表？

Python 3.6
known limitations

```
1  a = [[1, 2], [3, 4], [5, 6]]
2  b = a[1]
3  c = a[1][0]
```

Edit this code

line that just executed
next line to execute

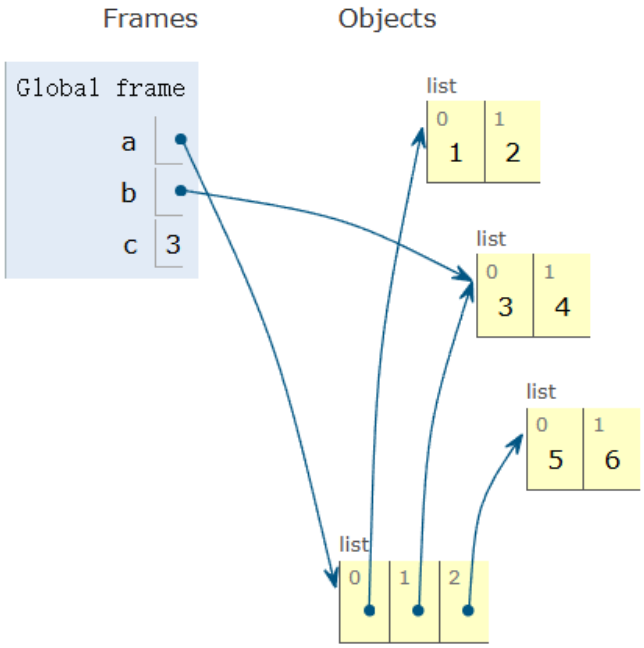
<< First | < Prev | Next > | Last >>

Done running (3 steps)

follow our YouTube, TikTok, Instagram for weekly tutorials

Get AI Help

Move and hide objects



报数（2022 Final No.4）

北京大学2022年军训开始啦！！！！！！！
大家都知道，军训过程中，报数是非常重要的一项内容。
所以这一次，我们聪明的教官突发奇想，设计出了一个报数小游戏，在锻炼同学的同时，又能增添军训的乐趣，可谓是一石二鸟，一箭双雕，赢麻了。
下面是游戏规则：
1、教官从同学中挑出n个同学，将他们编号为1,2,...,n，并让他们围成一圈（1->2->...->n->1）
2、教官给出一个数字m。
3、教官让编号为1的同学开始从1报数。
4、报数过程中，一旦有同学数到m，那么这个同学出圈。
5、有同学出圈后，下一个同学又从1开始报数。
6、同学们不断重复上述过程。问最后留下的同学的编号是什么。
输入样例：5 3　　输出样例：4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

N = 7, M = 4

报数（2022 Final No.4）

本质：把刚才做的整个过程，用代码复现一下！！

报数（2022 Final No.4）

本质：把刚才做的整个过程，用代码复现一下！！

Question 1：数据怎么存储？（列表？元组？字符串？集合？）

报数（2022 Final No.4）

本质：把刚才做的整个过程，用代码复现一下！！

Question 1：数据怎么存储？（列表？元组？字符串？集合？）

Question 2：怎么模拟刚才的过程？循环，淘汰n-1个人

报数（2022 Final No.4）

本质：把刚才做的整个过程，用代码复现一下！！

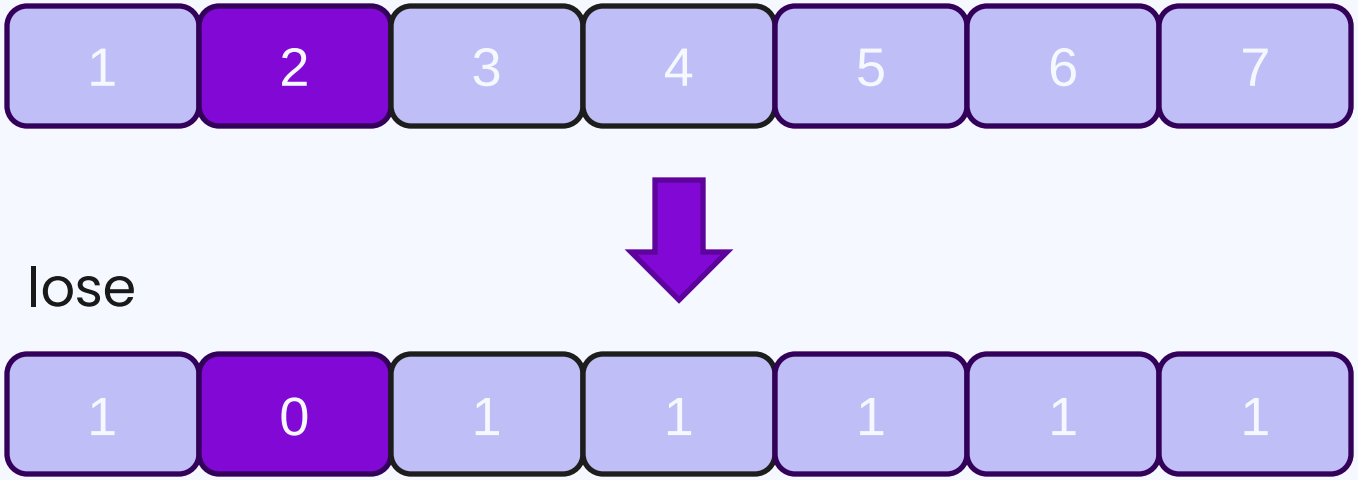
Question 1：数据怎么存储？（列表？元组？字符串？集合？）

Question 2：怎么模拟刚才的过程？循环，淘汰n-1个人

Question 3：怎么模拟“跳过”？需要记录被淘汰的人

报数（2022 Final No.4）

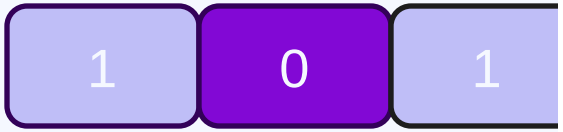
本质：把刚才做的整个过程，用代码复现一下！！



报数（2022 Final No.4）

本质：把刚才做的整

lose




```
n, m = tuple(map(int, input().split()))
lose = [0 for i in range(n)]
current = -1
for i in range(n - 1):
    count = 0
    while count < m:
        current = (current + 1) % n
        if not lose[current]:
            count += 1
    lose[current] = 1
for i in range(n):
    if not lose[i]:
        print(i + 1)
```

小鹿的工作休息时间（2022 Final No.5）

作为侦探，小鹿一刻不停地利用传送点在提瓦特大陆的各处奔波。每天他都会收到许多委托信件，他需要与委托人们见面以倾听事件的情况。每天他都会收到许多委托信件，他需要与委托人们见面以倾听事件的情况。委托人们的空闲时间各不相同并且有长有短，小鹿会排满自己的工作时间接受他们的委托并与其约见，在尽可能多见委托人的基础上，会和每个委托人尽可能充分交谈。

由于小鹿不喜欢工作与休息状态的切换，因此他希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

小鹿的工作休息时间（2022 Final No.5）

作为侦探，小鹿一刻不停地利用传送点在提瓦特大陆的各处奔波。每天他都会收到许多委托信件，他需要与委托人们见面以倾听事件的情况。每天他都会收到许多委托信件，他需要与委托人们见面以倾听事件的情况。委托人们的空闲时间各不相同并且有长有短，小鹿会排满自己的工作时间接受他们的委托并与其约见，在尽可能多见委托人的基础上，会和每个委托人尽可能充分交谈。

由于小鹿不喜欢工作与休息状态的切换，因此他**希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。**他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：第一行为小鹿今天的起床时间和入睡时间l,r。
第二行为委托数目n。
第三行起共n行，每行为某个委托人的空闲时间li,ri。

输出：第一行为小鹿今天的工作时间段，即开始工作的时间和结束工作的时间，用空格隔开。
第二行（及可能有的第三行）为小鹿今天的休息时间段，按时间顺序输出。

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15

5

1 5

9 10

5 6

6 9

8 10

输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15

5

1 5

9 10

5 6

6 9

8 10

输出：两行，1 10 \n 10 15

Wake up time! (1-15)

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10

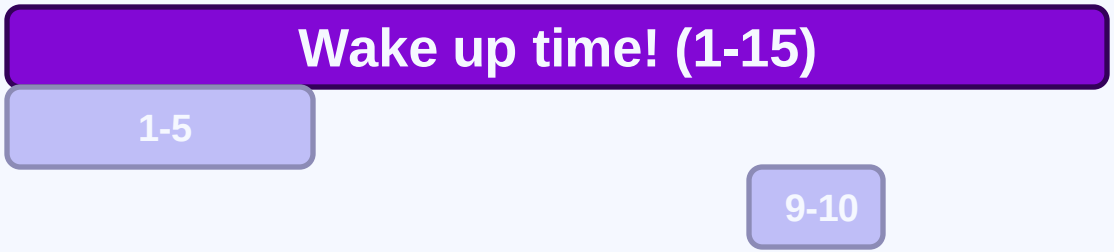
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



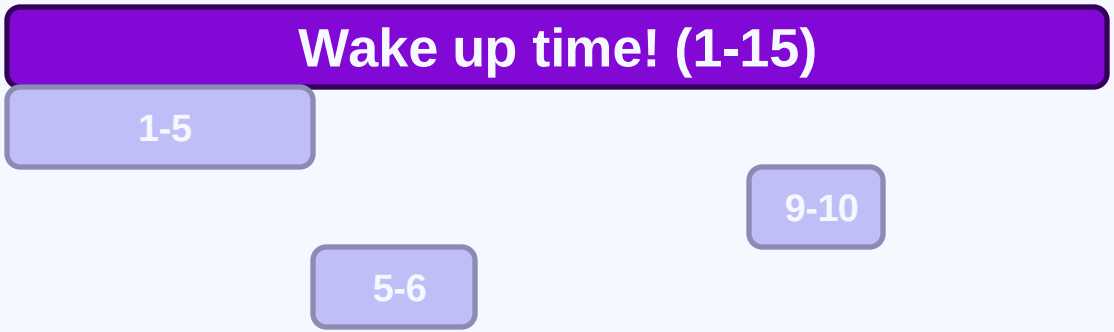
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



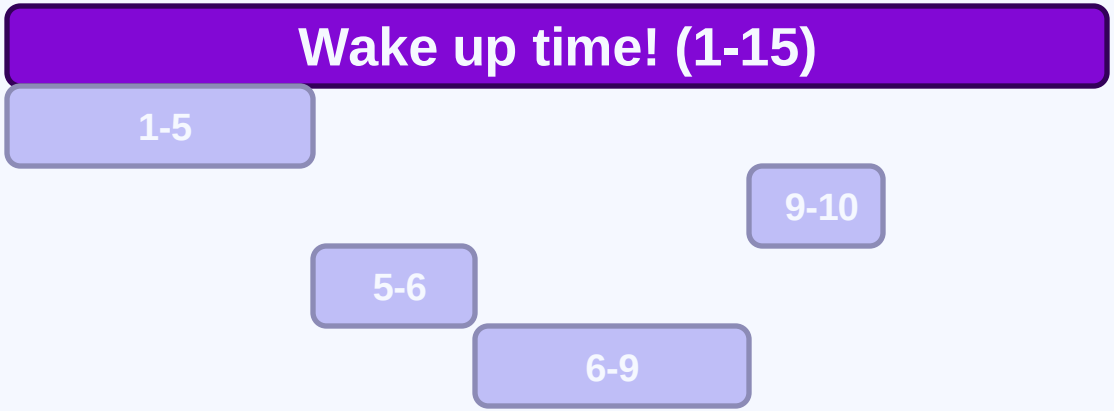
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



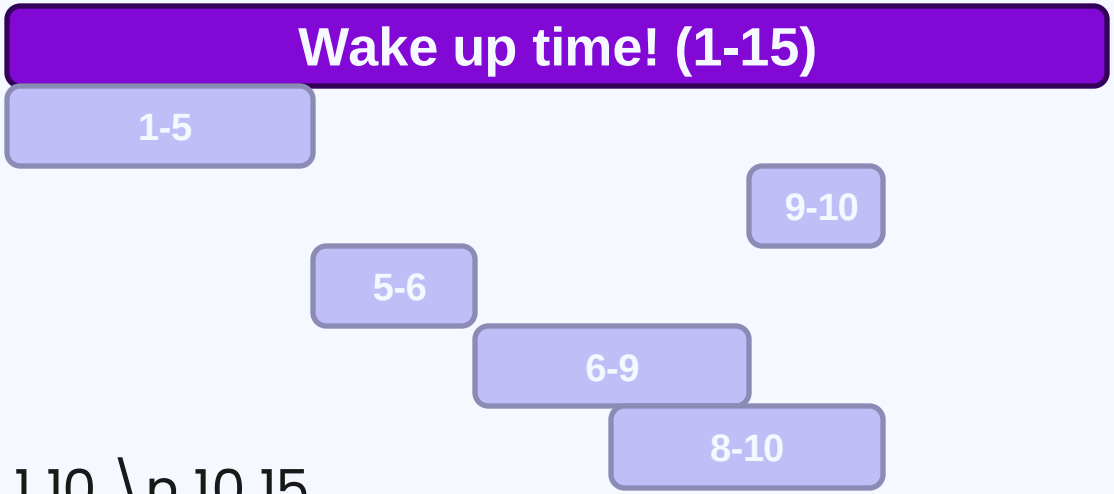
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



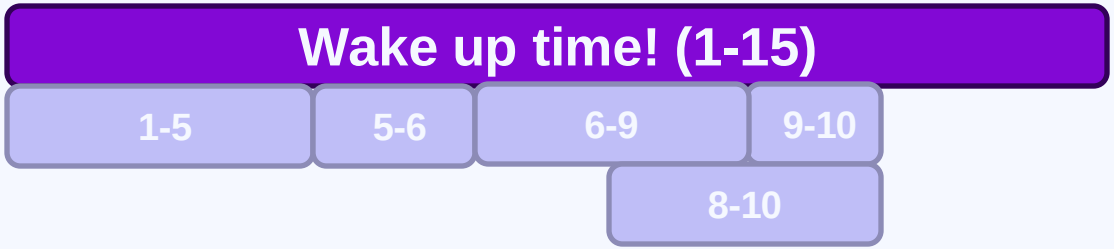
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



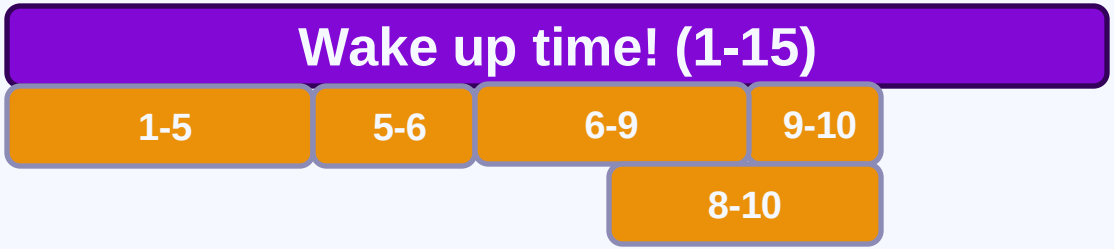
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
5
1 5
9 10
5 6
6 9
8 10



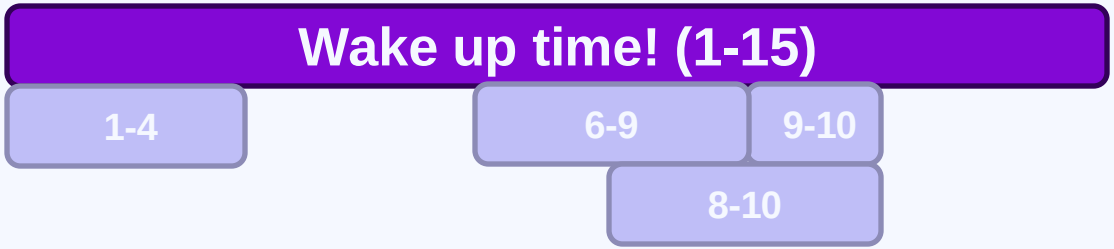
输出：两行，1 10 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

1 15
4
1 4
9 10
6 9
8 10



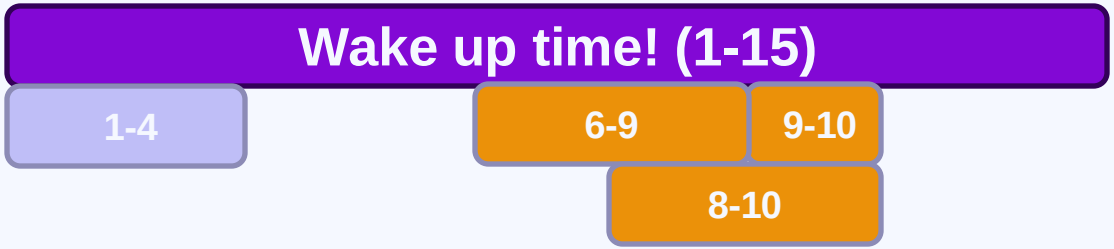
输出：三行，6 10 \n 1 6 \n 10 15

小鹿的工作休息时间（2022 Final No.5）

希望连续工作尽可能长的一整段时间，然后就开始休息。如果有多段可选的工作时间长度相同，就选最早的一段。他想计算在这个前提下，他的工作时间段和休息时间段分别如何分布。

输入：

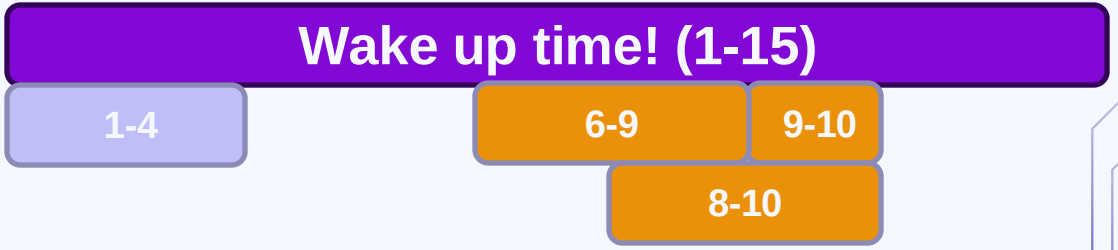
1 15
4
1 4
9 10
6 9
8 10



输出：三行，6 10 \n 1 6 \n 10 15

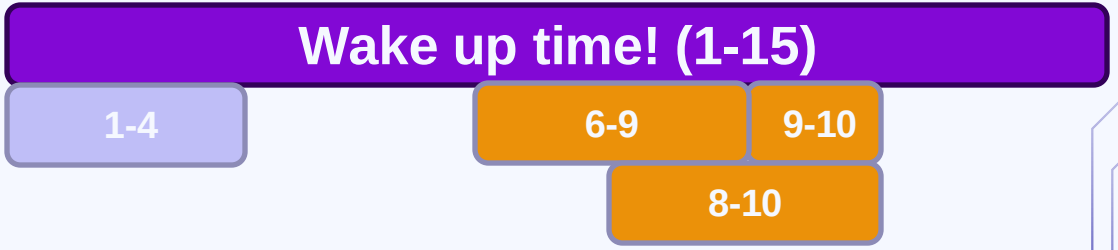
小鹿的工作休息时间（2022 Final No.5）

怎么分析？？



小鹿的工作休息时间（2022 Final No.5）

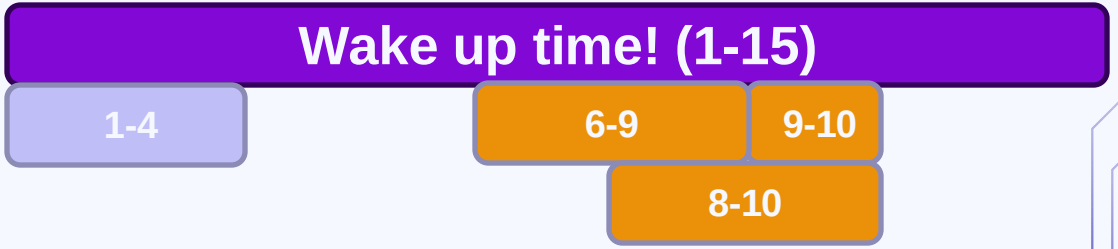
怎么分析？？



（如果不考虑编程，换成是你手动解决这个问题，你会怎么办？）

小鹿的工作休息时间（2022 Final No.5）

怎么分析？？



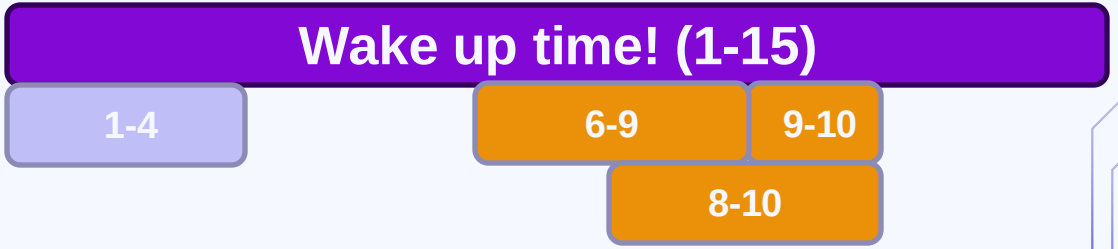
Answer：把有相同或者相邻时间段的都合并起来，
看看哪个时间段最长。

表示时间段？元组是一个不错的选择！（当然列表也可以）

(a, b): a period of time, start from a, end with b.

小鹿的工作休息时间（2022 Final No.5）

怎么分析？？



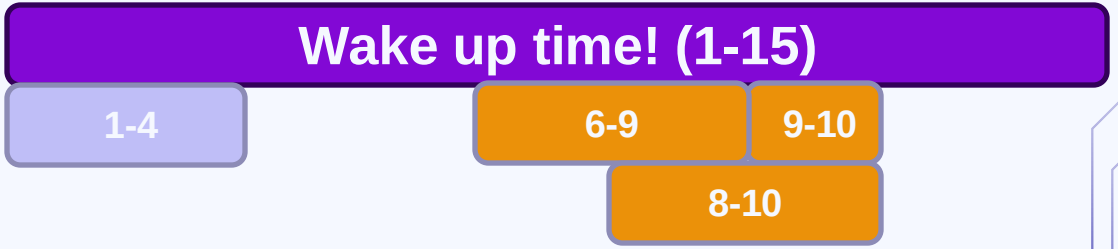
Answer：把有相同或者相邻时间段的都合并起来，看看哪个时间段最长。

表示所有的时间段？一个装着很多个元组的列表。

（为什么这里不用元组了？）

小鹿的工作休息时间（2022 Final No.5）

怎么分析？？



Answer：把有相同或者相邻时间段的都合并起来，看看哪个时间段最长。

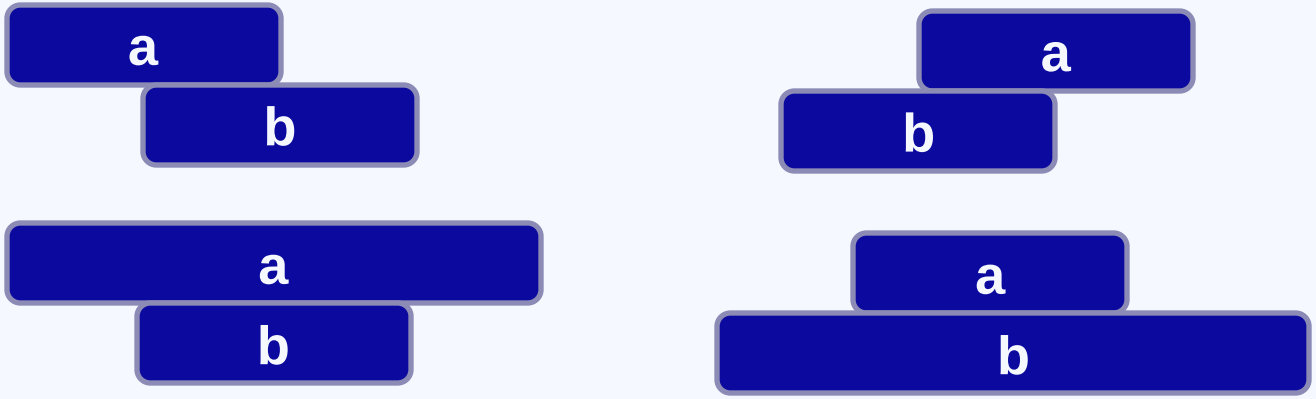
判断两个时间段是否可以合并？

```
def can_join(a, b)
```

小鹿的工作休息时间（2022 Final No.5）

判断两个时间段是否可以合并？

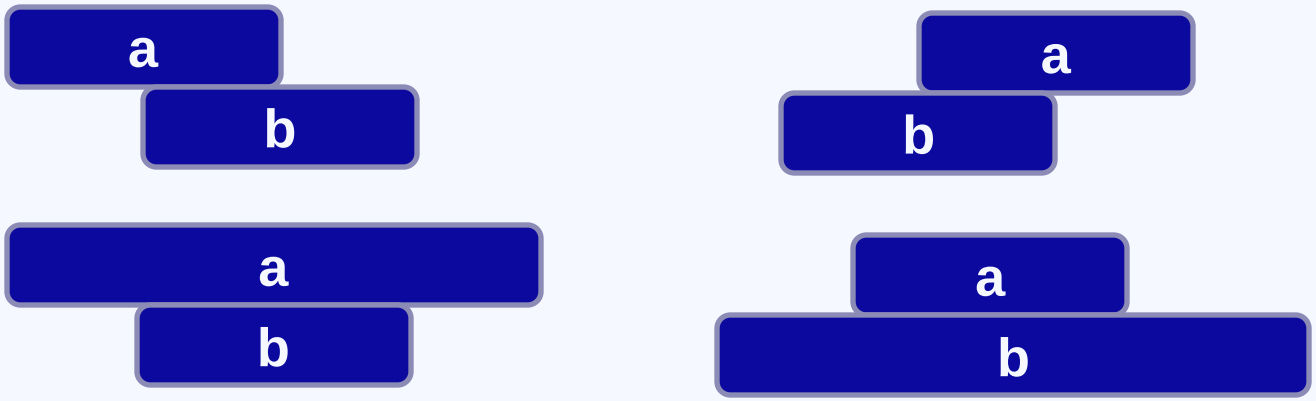
def can_join(a, b)



小鹿的工作休息时间 (2022 Final No.5)

合并两个时间段

def join(a, b)



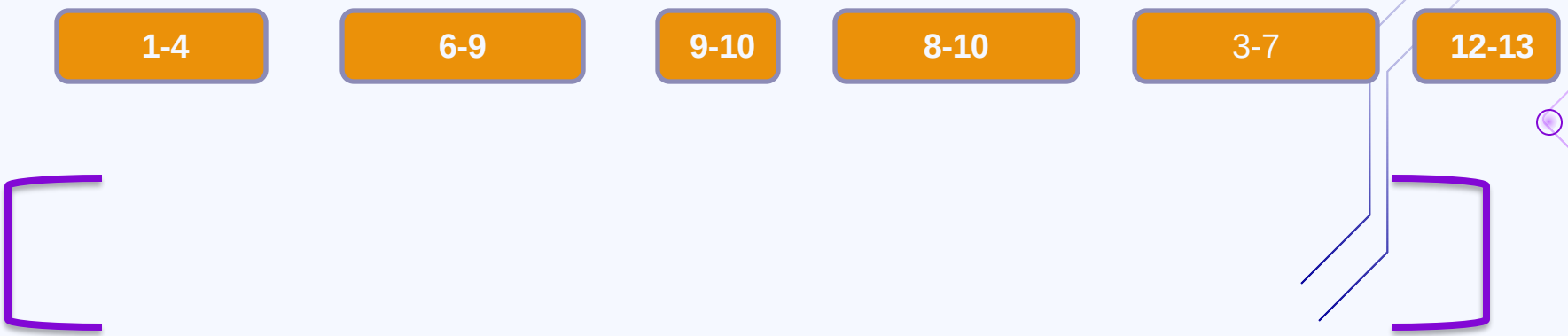
小鹿的工作休息时间 (2022 Final No.5)

合并两个时间段

def join(a, b)

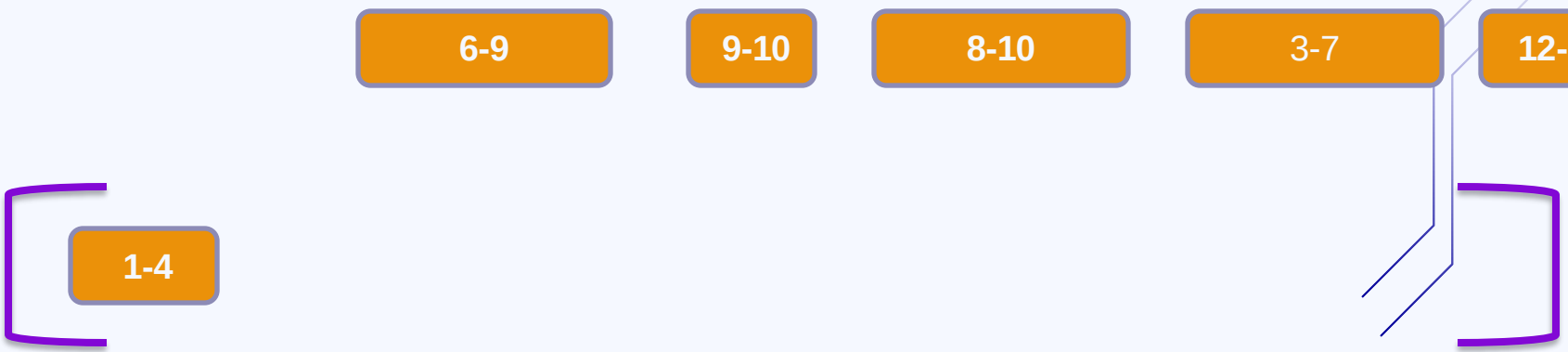
小鹿的工作休息时间（2022 Final No.5）

输入：



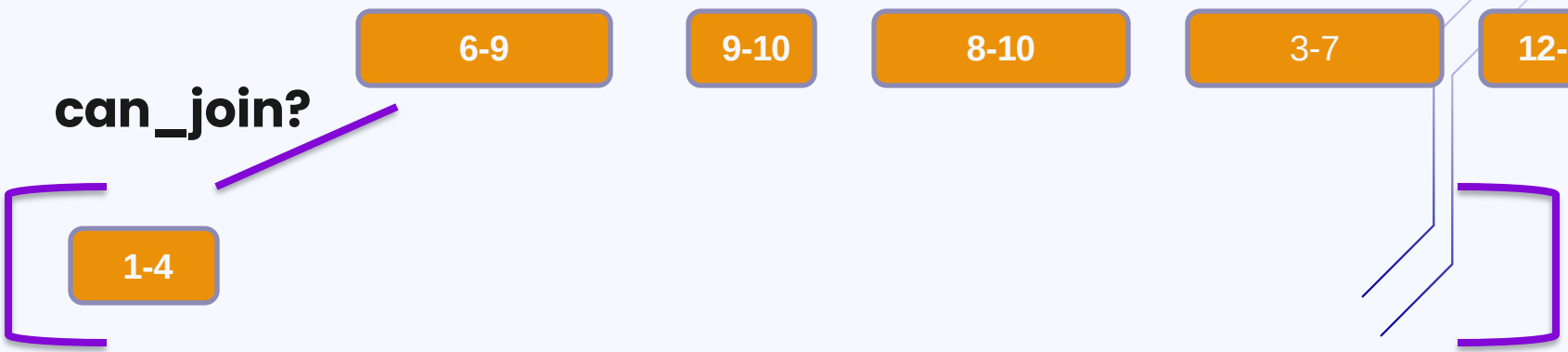
小鹿的工作休息时间（2022 Final No.5）

输入：



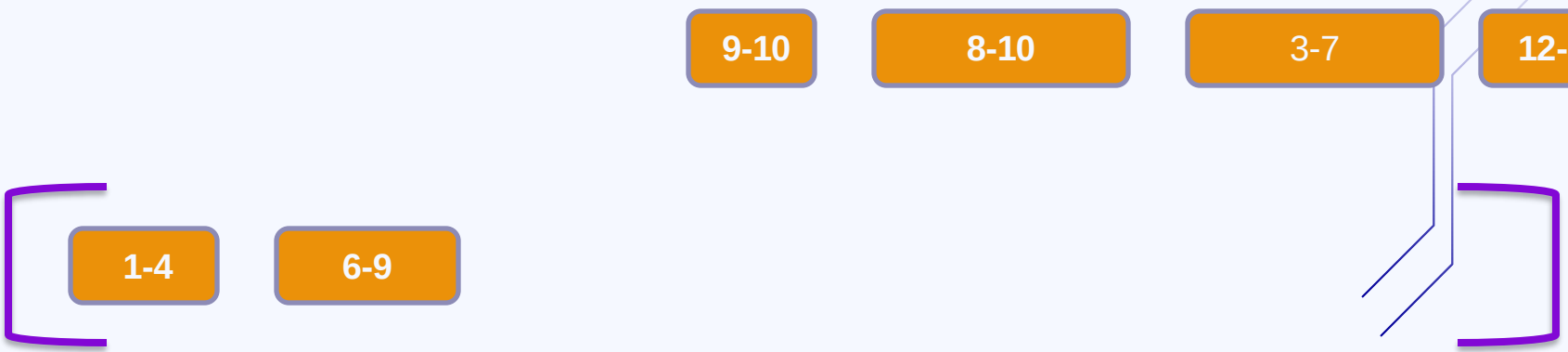
小鹿的工作休息时间（2022 Final No.5）

输入：



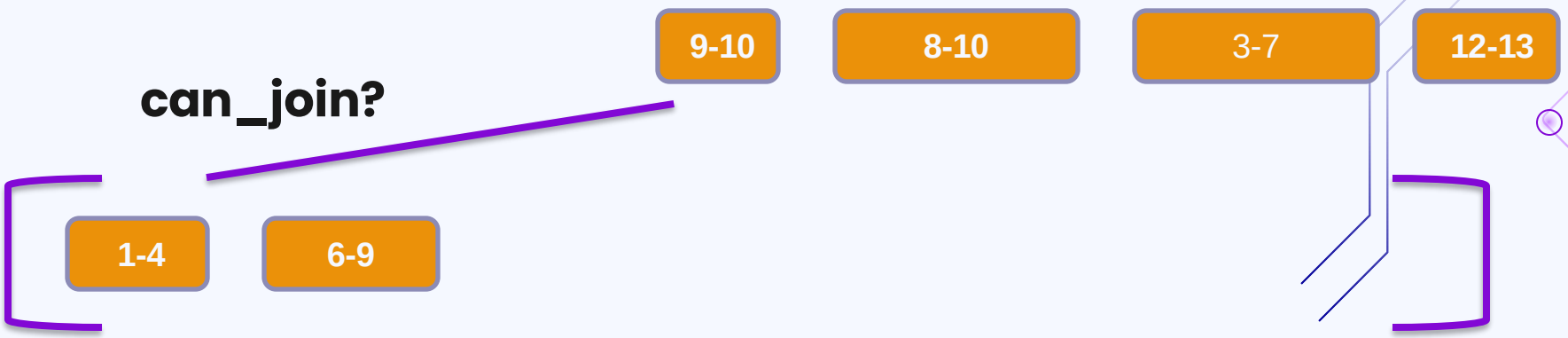
小鹿的工作休息时间（2022 Final No.5）

输入：



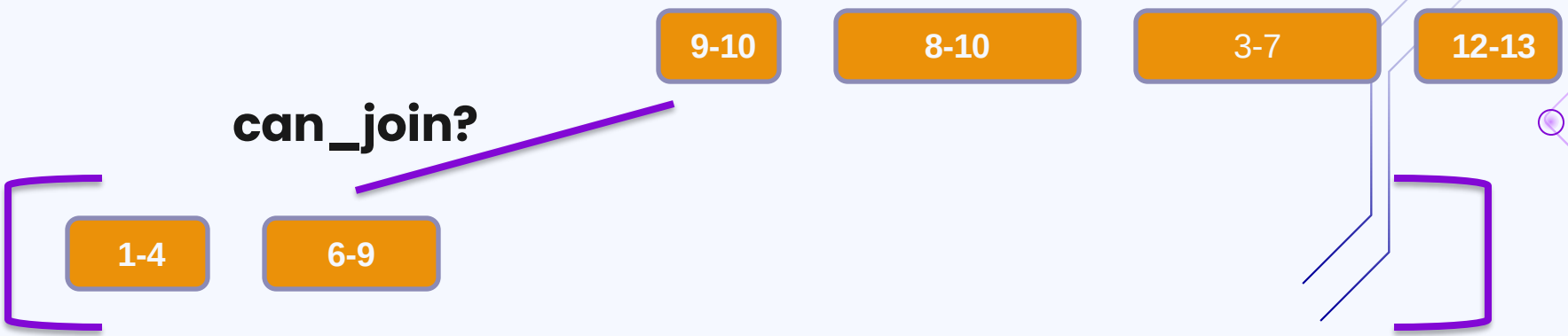
小鹿的工作休息时间（2022 Final No.5）

输入：



小鹿的工作休息时间（2022 Final No.5）

输入：



小鹿的工作休息时间（2022 Final No.5）

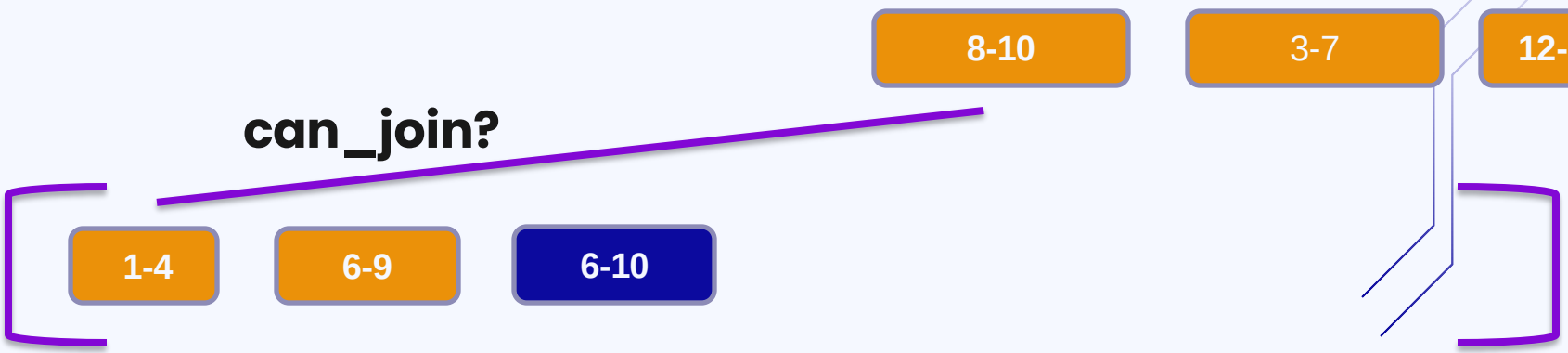
输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

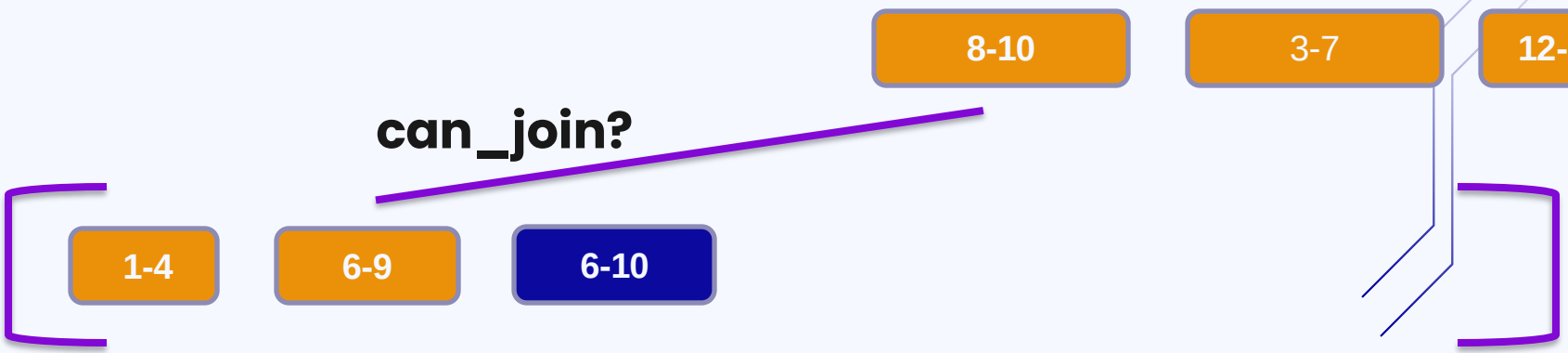
小鹿的工作休息时间（2022 Final No.5）

输入：



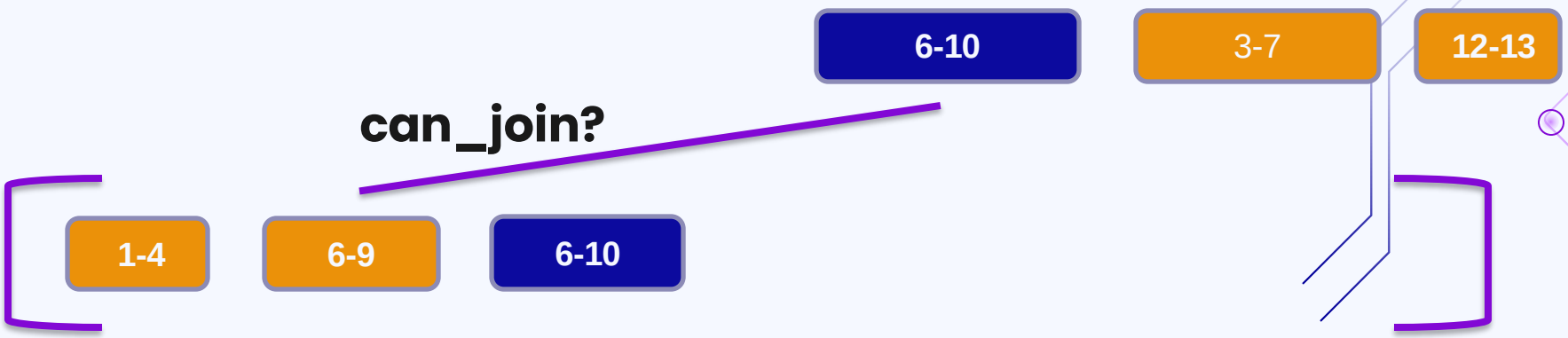
小鹿的工作休息时间（2022 Final No.5）

输入：



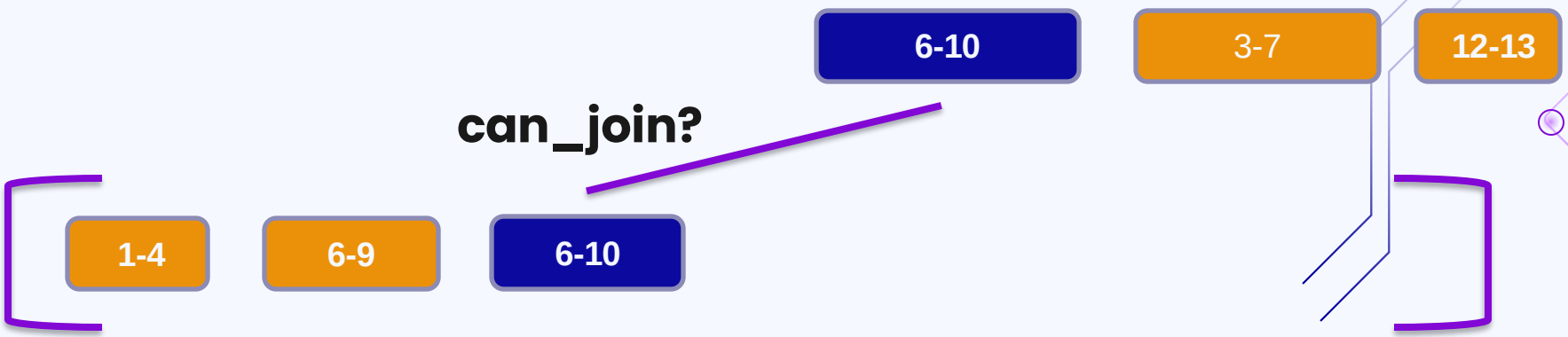
小鹿的工作休息时间 (2022 Final No.5)

输入：



小鹿的工作休息时间（2022 Final No.5）

输入：

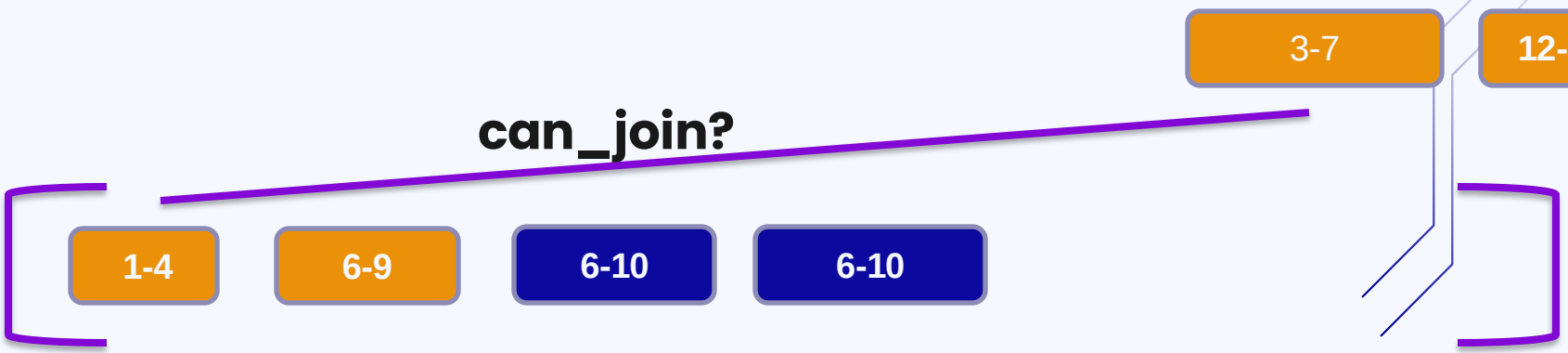


小鹿的工作休息时间（2022 Final No.5）

输入：

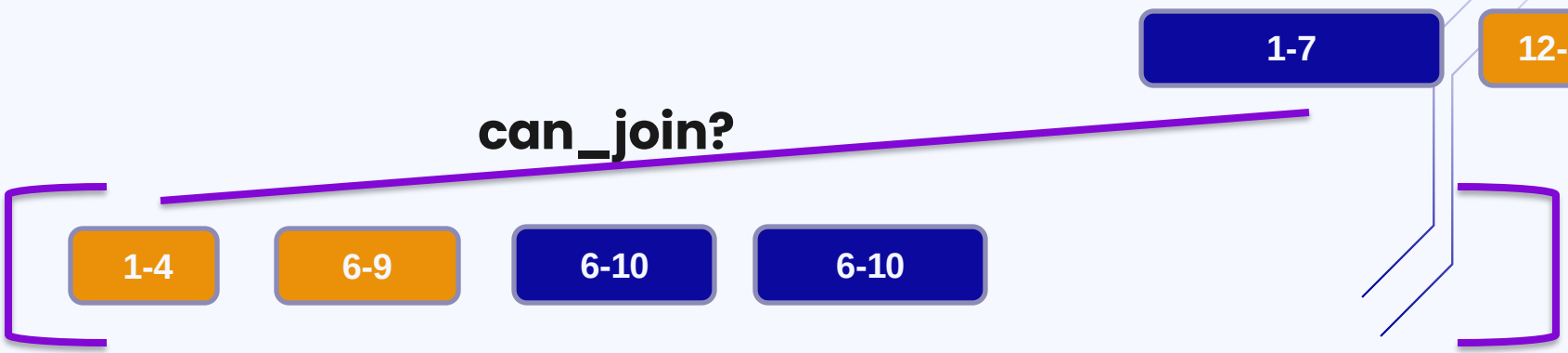
小鹿的工作休息时间（2022 Final No.5）

输入：



小鹿的工作休息时间（2022 Final No.5）

输入：



小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

输入：

小鹿的工作休息时间（2022 Final No.5）

得到的最终合并过后的列表，按照**时间的长度排序，取长度最大的**

小鹿的工作休息时间（2022 Final No.5）

得到的最终合并过后的列表，按照**时间的长度排序，取长度最大的**

小鹿的工作休息时间（2022 Final No.5）

得到的最终合并过后的列表，按照**时间的长度排序，取长度最大的**

多个长度最大的呢？看题干：“如果有多段可选的工作时间长度相同，就选最早的一段。”

小鹿的工作休息时间（2022 Final No.5）

```
def length(a):
    return a[1] - a[0]


def can_join(a, b):
    if (a[0] <= b[0] <= a[1] <= b[1] or b[0] <= a[0] <= b[1] <= a[1]
            or a[0] <= b[0] <= b[1] <= a[1] or b[0] <= a[0] <= a[1] <=
        return True
    return False


def join(a, b):
    new_min = min(a[0], b[0])
    new_max = max(a[1], b[1])
    return (new_min, new_max)
```

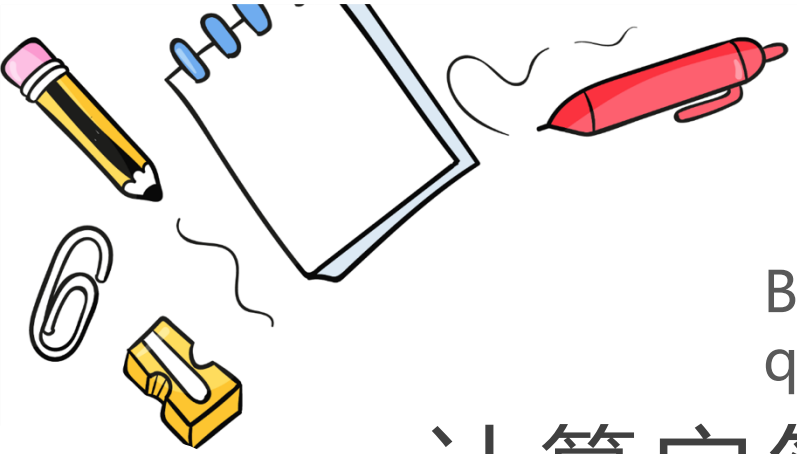
小鹿的工作休息时间（2022 Final No.5）

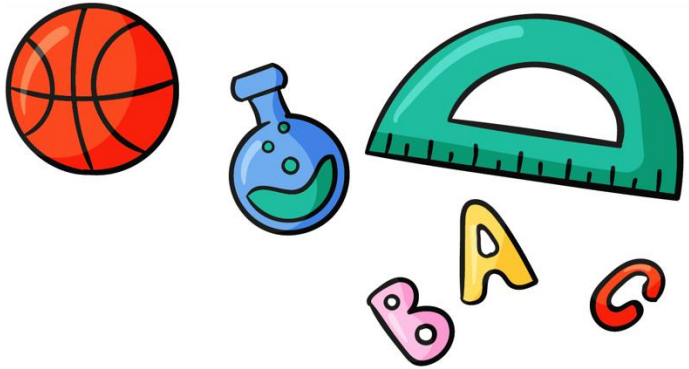
```
all_times = []
wake, sleep = tuple(map(int, input().split()))
N = int(input())
for i in range(N):
    current_time = tuple(map(int, input().split()))
    for j in range(len(all_times)):
        if can_join(current_time, all_times[j]):
            current_time = join(current_time, all_times[j])
    all_times.append(current_time)

all_times.sort(key=length, reverse=True)
longest_times = [t for t in all_times if length(t) == length(all_times[
longest_times.sort(key=lambda x: x[0])
final_choice = longest_times[0]

print(final_choice[0], final_choice[1])
if final_choice[0] > wake:
    print(wake, final_choice[0])
if final_choice[1] < sleep:
    print(final_choice[1], sleep)
```

得 …最大的

12-13

多 …长度相同，





Below is a bonus question!

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!

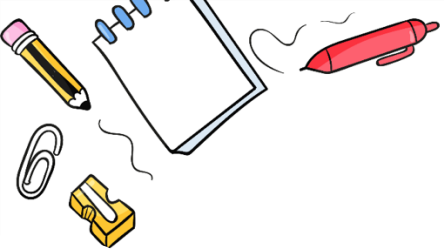
计算字符串距离

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"`.

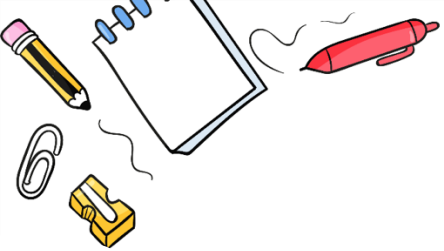
Each edit operation contributes 1 to the difference between two words.

```
>>> 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

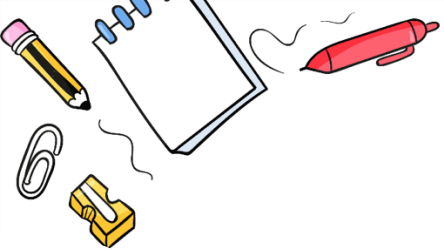


举例

上北大　　上北大

Distance：0

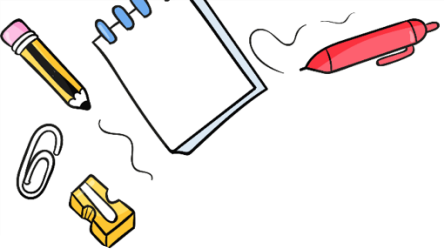
是不是直接比较两个字符串之间的差异就行了？
好像不太行……
字符串之间的差异可能比较大



举例

D A B C

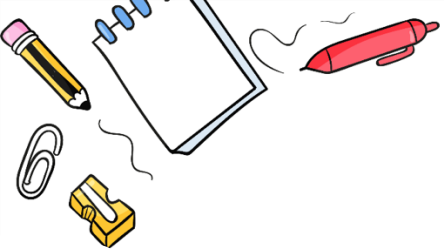
A B C



举例

上北大　　上北大

Distance：1



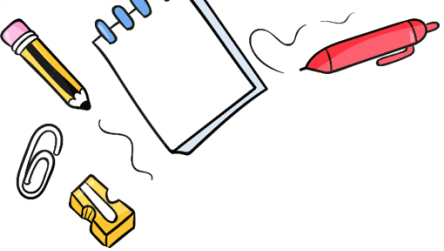
举例

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



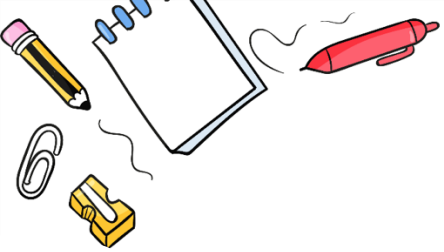
举例

北大　　清华

Distance：2

Observation 3:前端相同的字符可以删去，不影响结果

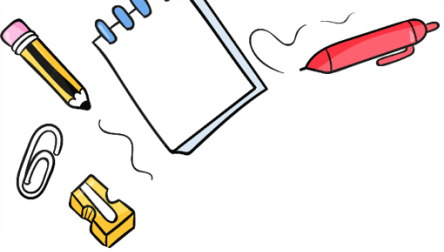
Intuition: 从两个字符串的前端开始操作



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?

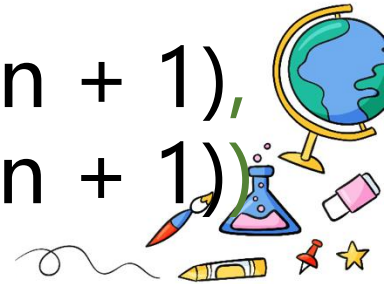
1. 两边同时删掉一个字符
2. 右边字符串删掉一个字符
3. 左边字符串删掉一个字符

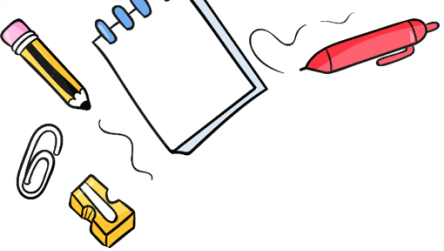
取后面字符串距离的min加1

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
```



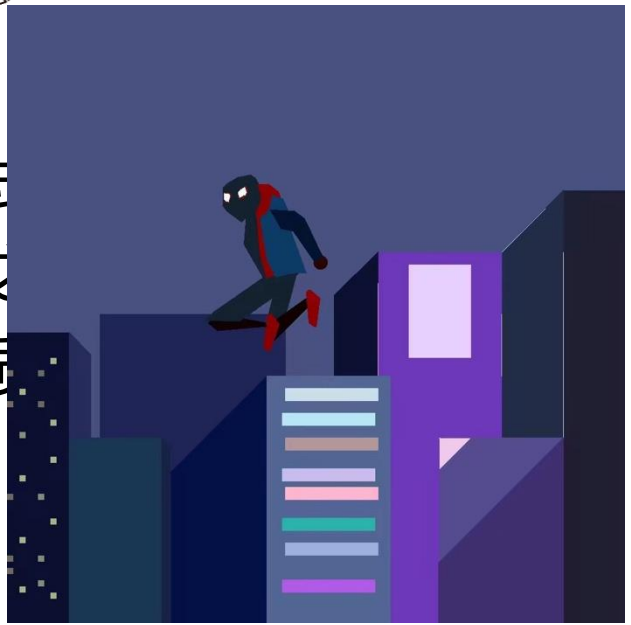


Base Case

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!

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



CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution