程序设计（Python）

函　数 2

主讲：数据与目标工程学院 胡瑞娟 副教授

知识回顾

【案例背景】在军事战争中，不同的作战单位需要配备不同数量的弹药。现代智能化作战背景下，假设有一种新型AI弹药分配系统，可以根据作战单位的类型和任务需求，快速计算所需的弹药数量。在分配弹药时，需要考虑不同弹药类型的排列组合方式对装载方案的影响。

任务：

（1）计算n种弹药的分配方案数：一个作战单位有n种不同类型的弹药，计算这些弹药的所有可能分配顺序。

求n!

（2）从n种弹药中选取m种进行排列的方案数：在自动装弹机中，弹药的顺序可能会影响装填速度，考虑不同弹药的装填顺序。

P(n, m) = n! / (n-m)!

（3）从n种弹药中选取m种进行组合的方案数：计算从弹药库中选择哪些弹药类型，不考虑顺序。

C(n, m) = n!/(m!*(n-m)!)

知识回顾

- 什么是函数?

  函数就是执行特定任和以完成特定功能的一段代码。

```
def 函数名（[输入参数]）：
    函数体
    [return xxx]
```

- 为什么需要函数?
  - 复用代码
  - 隐藏实现细节
  - 提高可维护性
  - 提高可读性
  - 便于调试

案例解决

问题：

（1）计算n种弹药的分配方案数。

（2）从n种弹药中选取m种进行排列的方案数。

（3）从n种弹药中选取m种进行组合的方案数。

```python
# 定义阶乘函数
def fact(n):
    """计算n的阶乘"""
    s = 1
    for i in range(1, n + 1):
        s *= i
    return s
```

阶乘的数学定义：

$n! = n \times (n-1)!$　$(n > 0)$

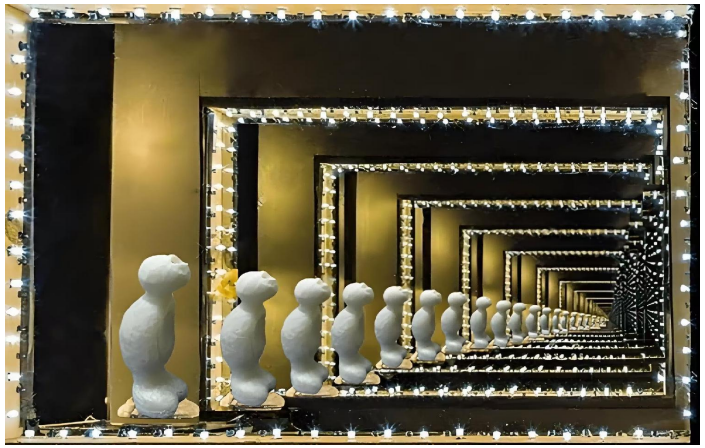
$0! = 1$

递归

```python
# 定义排列函数
def perm(n, m):
    """计算从n个元素中取m个元素的排列数"""
    if m > n:
        return 0
    per_nm= fact(n)//fact(n-m)
    return per_nm
```

```python
# 定义组合函数
def comb(n, m):
    """计算从n个元素中取m个元素的组合数"""
    if m > n:
        return 0
    comb_nm = fact(n)//(fact(m)*fact(n-m))
    return comb_nm
```

你有没有曾站在两面镜子中间？会看到什么？



“无限镜隧道”

德罗斯特效应(一张图片的某个部分与整张图片相同，从而产生无限循环的效果)

德罗斯特效应

主要内容

01 函数递归

02 模块

01

函数递归

1. 函数递归

- 函数定义中调用函数自身的方式形成递归。
- 递归函数：如果在一个函数的函数体内调用了该函数本身，这个函数就称为递归函数。

【案例问题】计算n种弹药的分配方案数（用递归解决）。

阶乘一般定义：n！=n(n-1)(n-2).....(1)

另一种表达方式：

$$n!=\begin{cases}1 & (n=0)\\ n(n-1)! & (n>0)\end{cases}$$

【案例问题】计算n种弹药的分配方案数（用递归解决）。

求第n项的阶乘

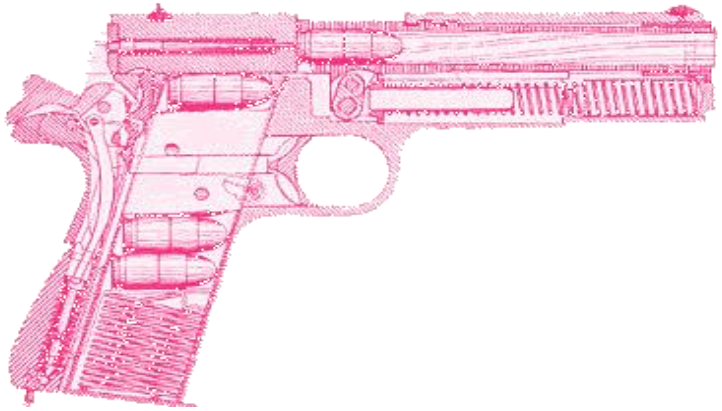
```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
num = int(input("输入一个整数: "))
print(fact(num))
```

函数的调用机制
执行过程？？（画图）

递归的调用过程

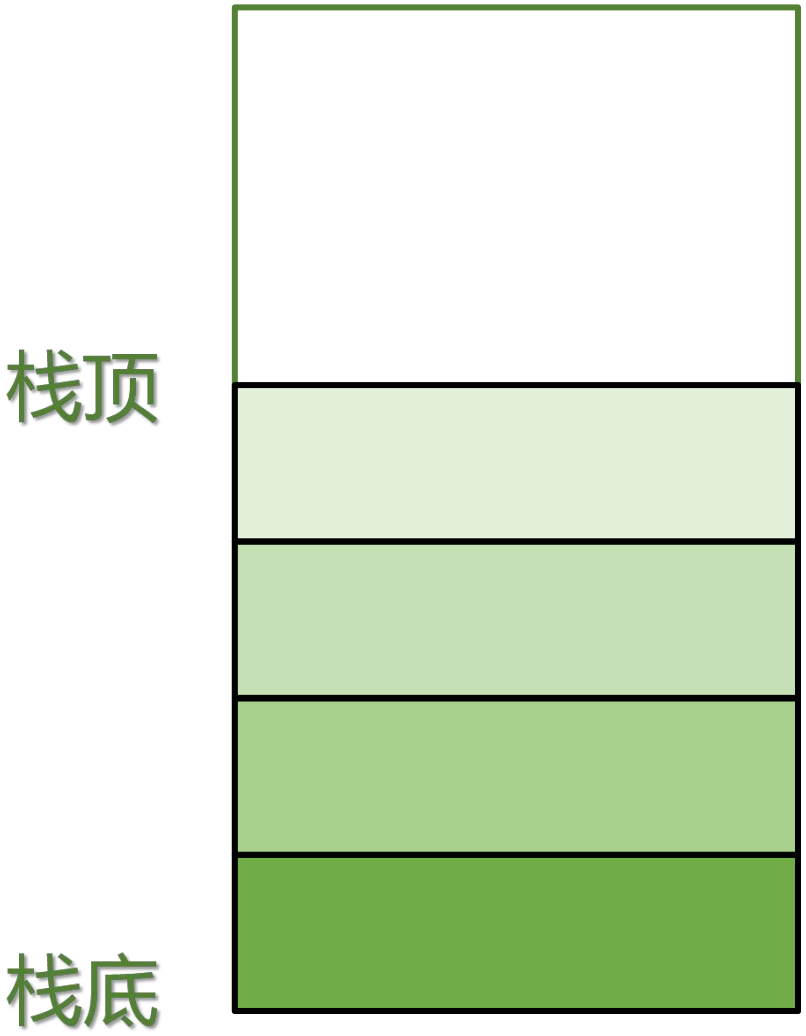
- ✓ 每递归调用一次函数，都会在**栈内存**分配一个栈帧
- ✓ 每执行完一次函数，都会释放相应的空间

栈的基本概念



- 栈的操作：入栈出栈
- 栈的性质：后入先出

函数最本质的调用机制，就是在内存栈区后入先出的执行过程



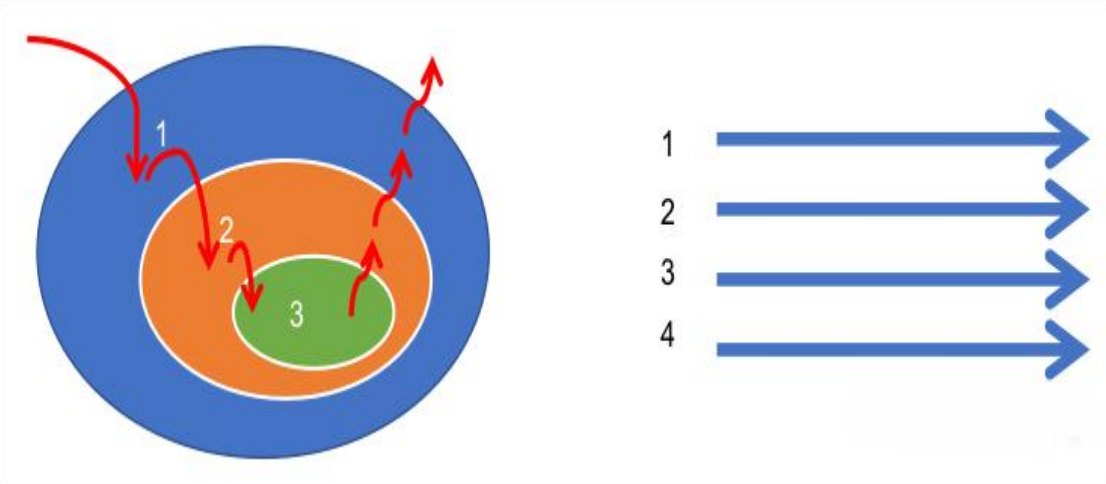
1. 函数递归

执行过程

```
f(6)
=> 6 * f(5)
=> 6 * (5 * f(4))
=> 6 * (5 * (4 * f(3)))
=> 6 * (5 * (4 * (3 * f(2))))
=> 6 * (5 * (4 * (3 * (2 * f(1)))))
=> 6 * (5 * (4 * (3 * (2 * 1))))
=> 6 * (5 * (4 * (3 * 2)))
=> 6 * (5 * (4 * 6))
=> 6 * (5 * 24)
=> 6 * 120
=> 720
```

递

归



递归的重复是层层渐近，每层之间互相影响，而循环则是互不干扰。

1. 函数递归

递归的条件：

（1）存在一个或多个**基例（n==0)**，不需要再次递归，它是确定的表达式。

（2）所有**递归链**要以一个或多个**基例**结尾。

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
num = int(input("输入一个整数: "))
print(fact(num))
```

以下哪个是递归函数的必备要素?

A 局部变量

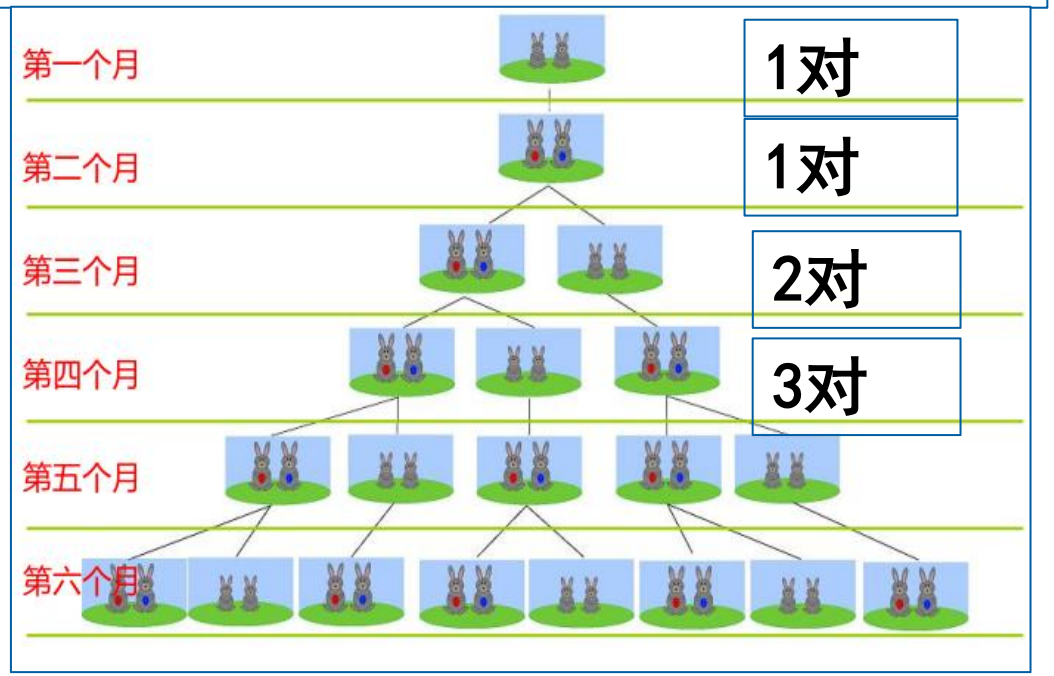
B 一定要有基例（终止条件）

C 循环结构

D 全局变量

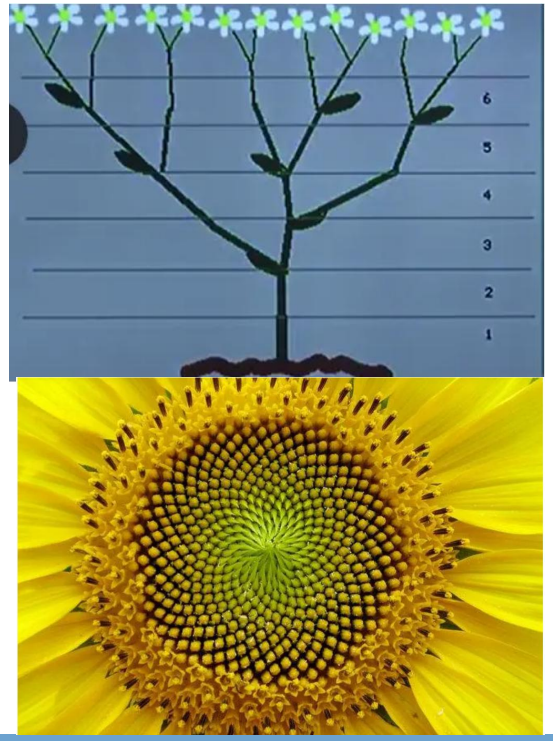
1. 函数递归

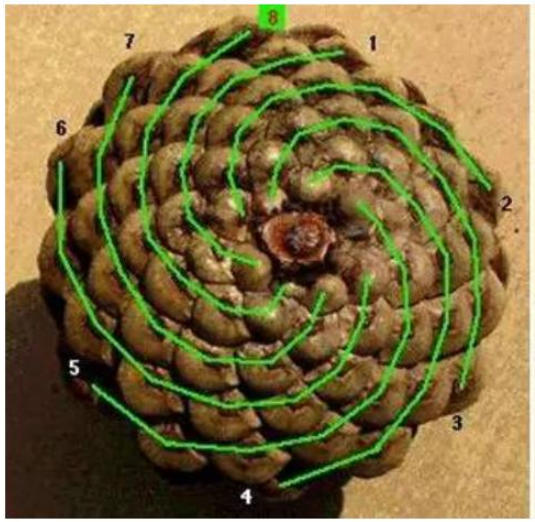
斐波那契数列（Fibonacci Sequence）最早是由意大利数学家莱昂纳多·斐波那契（Leonardo Fibonacci）在1202年的著作《计算之书》（Liber Abaci）中提出的，而它的起源正是基于一个兔子繁殖问题。

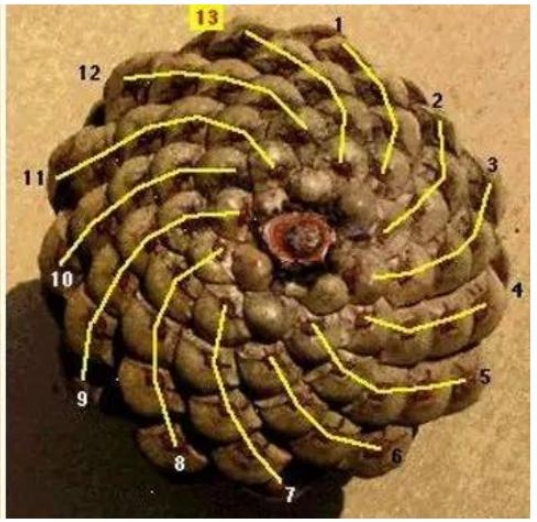


1. 函数递归

【植物与斐波那契数】大自然中大多数花的花瓣的个数为斐波那契数，如2,3,5,8,13等等。植物学家还发现树木的分叉也与斐波那契数列有关;植物的生长周期中分叉的个数基本符合斐波那契数列：向日葵花盘中顺时针旋转的螺线条数和逆时针旋转的螺线条数恰好是两个相邻的斐波那契数.譬如34和55，或者89和144等。人们还发现菠萝和松果的花和种子也是类似的排列，上面的螺旋线有的是8条，有的是13条。34、21、8、13，这些都是斐波那契数。不过，这样的排列是如何形成的，科学家们还没有找到答案。







1. 函数递归

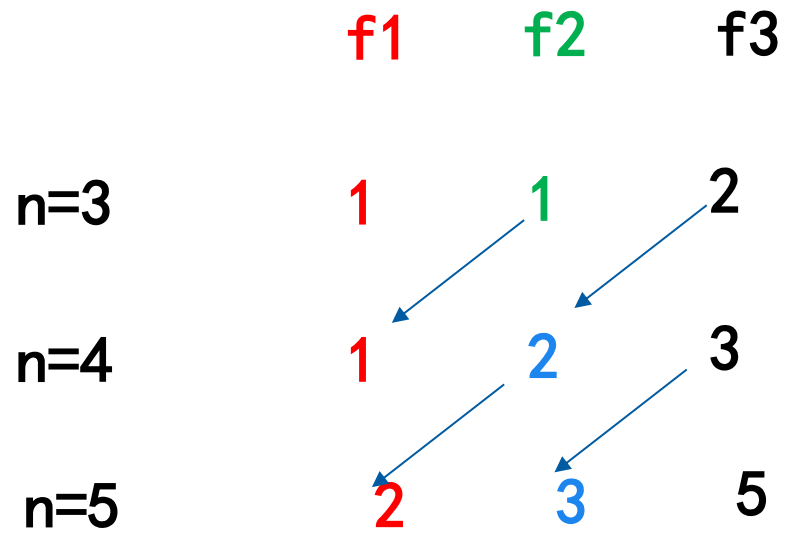
- 斐波那契数列:1，1，2，3，5，8.....

$$n!=\begin{cases}1 & n=1\\ 1 & n=2\\ f(n-1)+f(n-2) & n>2\end{cases}$$

迭代法是一种通过不断重复步骤来逼近解的方法。

- **n为1或者2时**，返回值为1
- **n为其他值时**，**迭代**计算
  - 初始化: **f1,f2=1,1**
  - 计算的次数：3——n
  - 计算的过程：
    ```
    f3=f1+f2
    f1=f2
    f2=f3
    ```

□ 非递归思路：



1. 函数递归

- 斐波那契数列:1，1，2，3，5，8.....

$$n!=\begin{cases}1 & n=1\\ 1 & n=2\\ f(n-1)+f(n-2) & n>2\end{cases}$$

- **非递归函数（递推）**

```
def fs1(n):
    if n==1 or n==2:
        return 1
    f1,f2=1,1
    for i in range(3,n+1):
        f3=f1+f2
        f1=f2
        f2=f3
    return f3
```

- **你能用递归方法求解吗?**

1. 函数递归

- 斐波那契数列:1，1，2，3，5，8.....

✓ 递归函数

定义递归函数

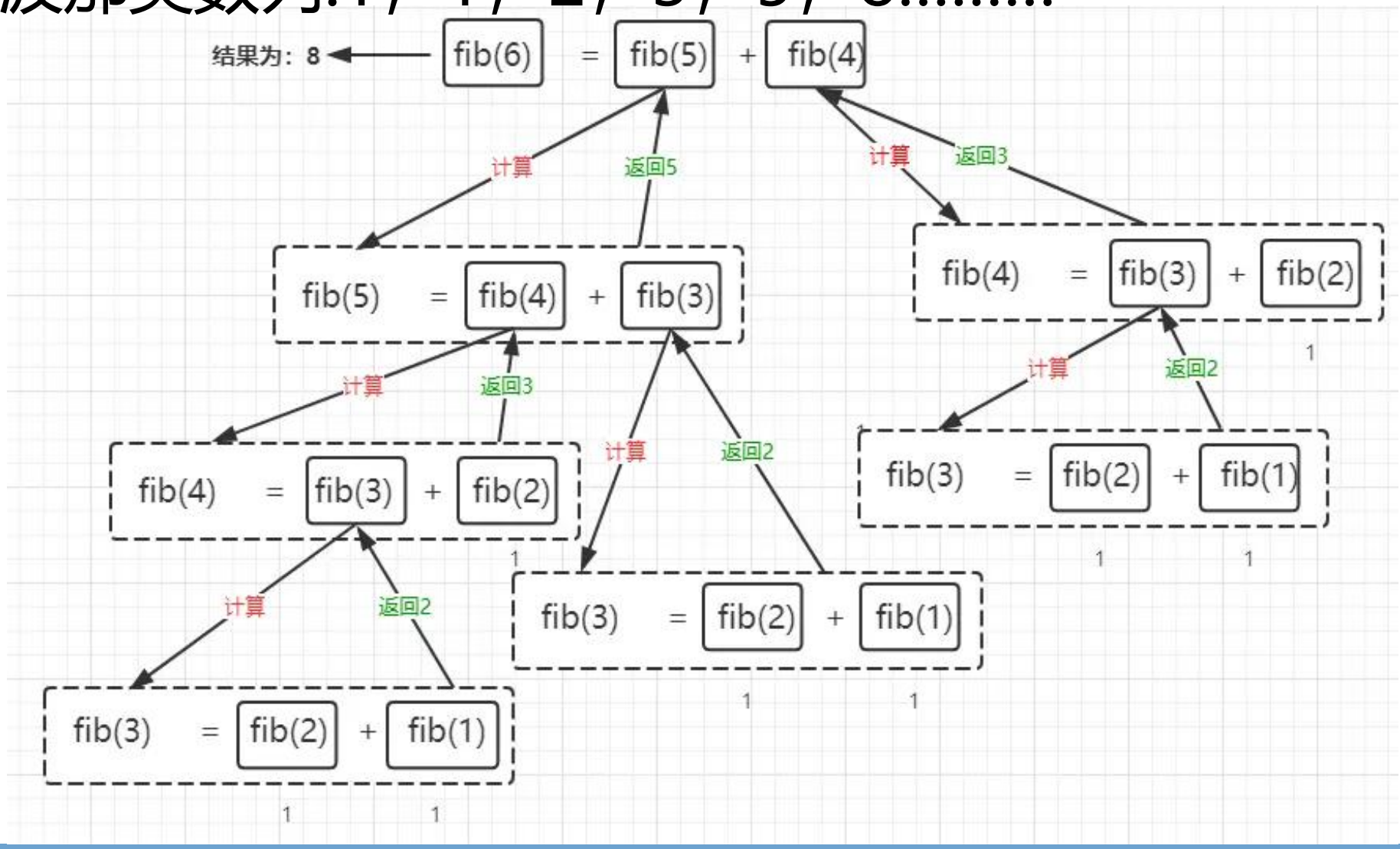
```
def fs(n):
    if n==1 or n==2:
        return 1
    else:
        return fs(n-1)+fs(n-2)
```

调用递归函数

```
n=eval(input("请输入一个正整数："))
print(fs(n))
```

1. 函数递归

- 斐波那契数列:1，1，2，3，5，8.....



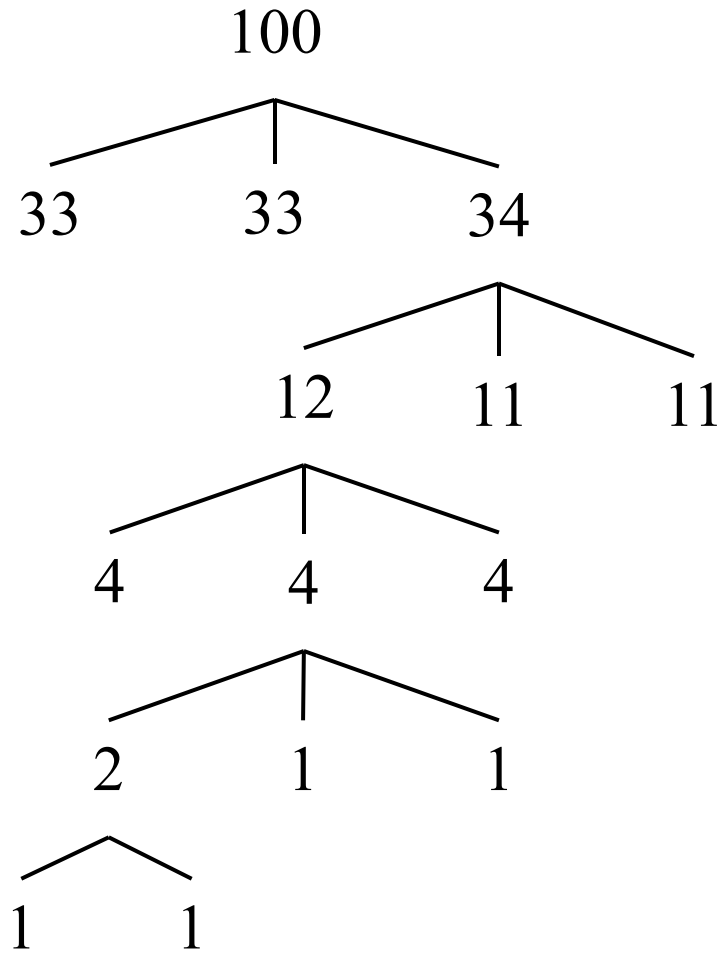
1. 函数递归

- 递归是一种思维方式，即通过将问题分解为更小的子问题来解决。
- 复杂问题的核心解决逻辑：

  “分解问题 → 递归解决 → 组合结果”

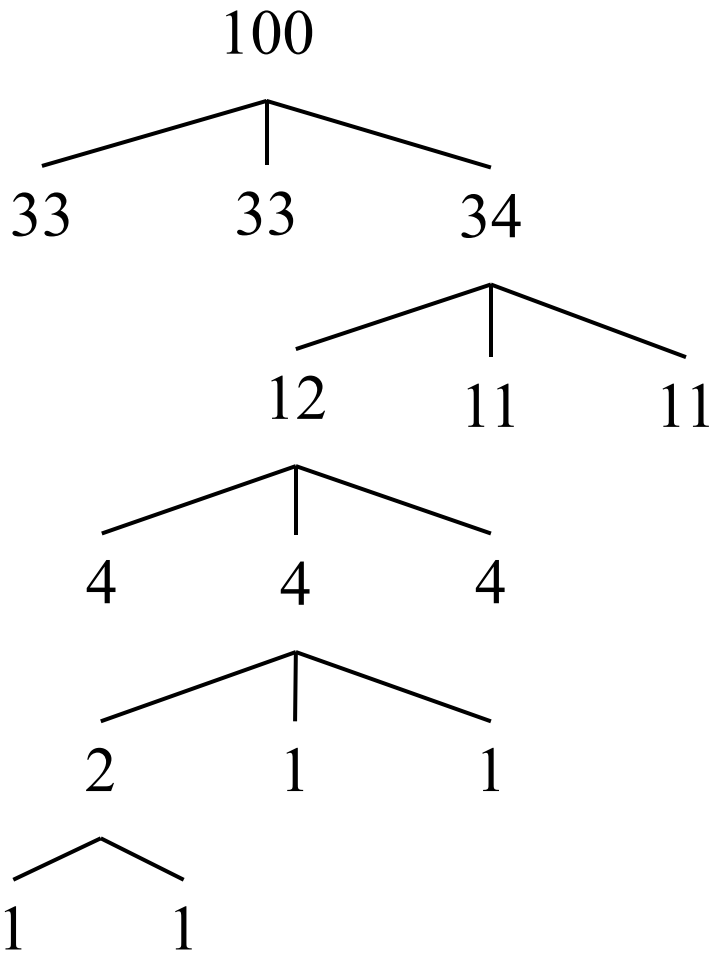
  无论是编程、数学，还是人生决策，这种思维模式都至关重要（但：嵌套层数深，函数调用开销大）
- 递归关键
  - 确定何时停止递归和何时继续递归的条件。
  - 如何递归

【举一反三】找假币问题，用函数递归解决。




```
def f(n):
    if n == 2:
        return 1
    elif n == 3:
        return 1
    elif n % 3 == 0:
        return 1 + f(n//3)
    else:
        return 1 + f(n//3+1)
n = int(input("请输入硬币的数量: "))
print(f"最少称重次数是: {f(n)}")
```

【举一反三】找假币问题



$$f(100) = 1 + f(34) = 5$$

$$f(34) = 1 + f(12) = 4$$

$$f(12) = 1 + f(4) = 3$$

$$f(4) = 1 + f(2) = 2$$

$$f(2) = 1$$

用f(n)表示当硬币数为n时需要次数

【举一反三】找假币问题

两个妙处

递归思维

计算思维中，最重要的是一种**自顶向下、先全局后局部**的逆向思维。

自顶向下逐步分解

$f(100) = 1 + f(34) = 5$

$f(34) = 1 + f(12) = 4$

$f(12) = 1 + f(4) = 3$

$f(4) = 1 + f(2) = 2$

$f(2) = 1$

自底向上层层回退

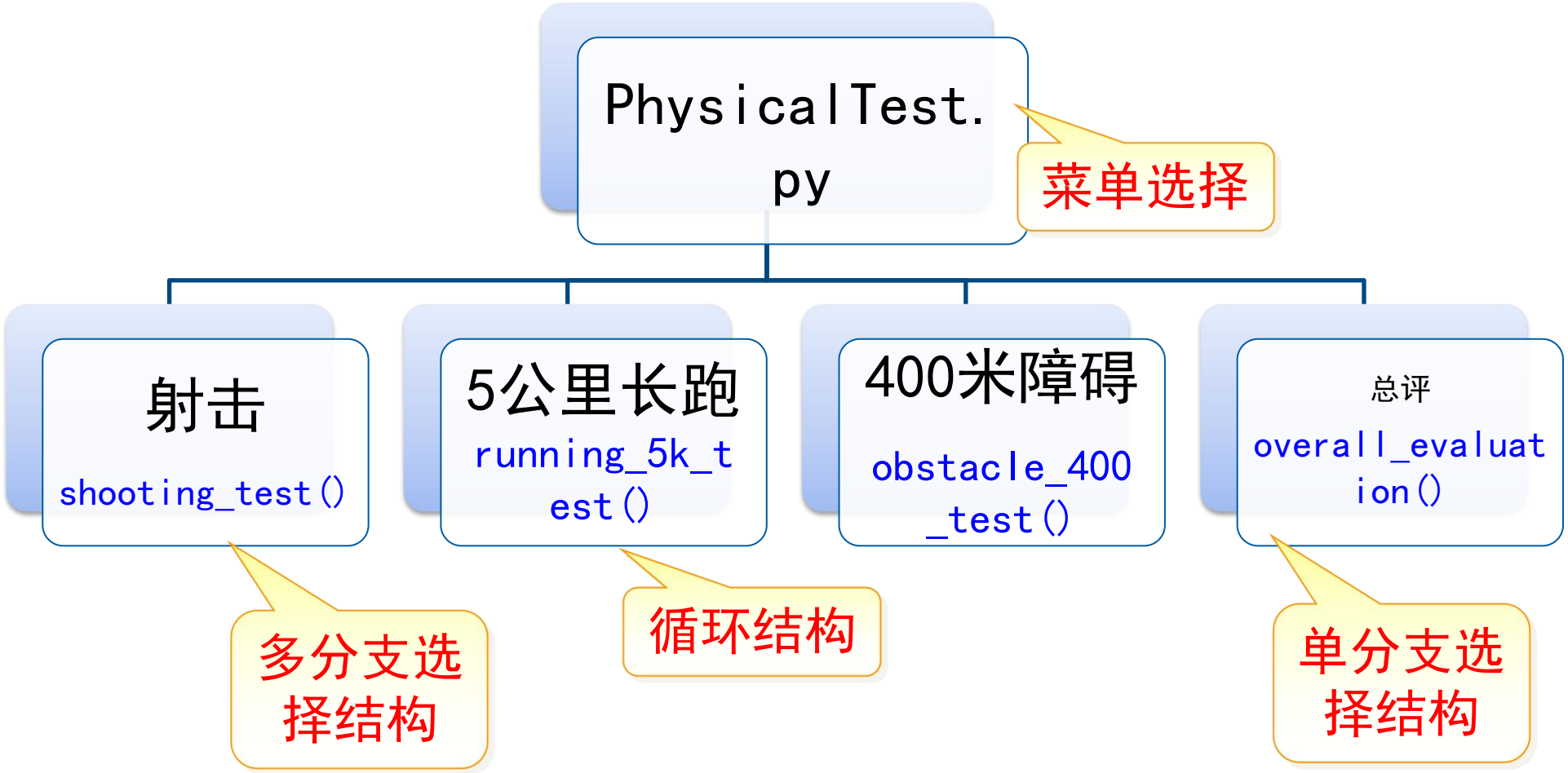
用f(n)表示当硬币数为n时需要次数

- 学生参加体能测试，测试项目有射击、5公里长跑、400米障碍，编程实现以下功能：
  - 菜单选择：1. 射击；2. 长跑5公里；3. 400米障碍；4. 总评；5. 退出
  - 射击（按成绩输出等级，如射中9环为优秀）
  - 5公里（记录每1公里或每圈用时、总时间）
  - 400米障碍
  - 总评

```
==============================
     体能测试管理系统
==============================
1. 射击测试
2. 5公里长跑测试
3. 400米障碍测试
4. 查看总评
5. 退出
------------------------------
请选择 (1-5) : 1

=== 射击测试 ===
说明：满环10环，9环以上优秀
请输入射击成绩 (0-10环) : 9.5
你的成绩：9.5环
等级：优秀
```

- 自顶向下的设计（模块化编程思想）



03

模块

模块就是把**函数、变量等结合**起来，形成一个**Python文件**，文件名字也就是模块的名字。

函数

模块化编程

模块化编程的好处：

- ✓ **简化编程**，把重点放在整个项目的逻辑上；
- ✓ **可维护性好**，即使出了问题也便于排查；
- ✓ **复用性好**，直接使用编写好的模块去实现功能，当需要重复实现时，再次调用即可，不必再重新编写了；
- ✓ **范围性好**，每个模块都有单独的命名空间，避免发生一些例如变量命名上的冲突。

Python模块分类

1.内置模块（库）（标准库）

- Python**安装时默认自带的模块**（库），例如：random库、time库、turtle库

2.第三方模块（库）

- 需要下载安装，然后才能使用，例如：jieba库，pygame库

3. 自定义模块

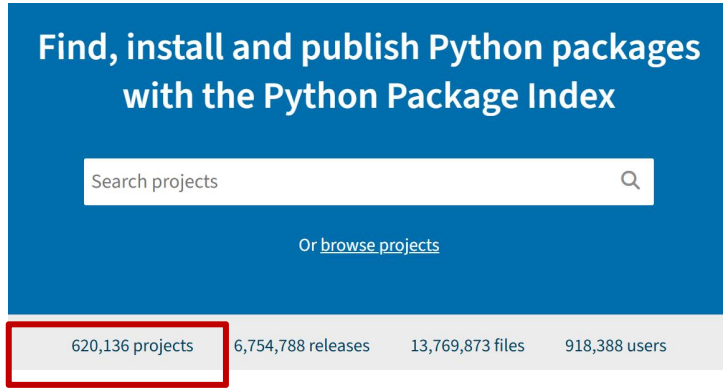
- 用户自定义，然后使用

计算生态“胶水语言”

- 以开源项目为代表的大量第三方库

  Python语言提供 >60多万个第三方库

  - 数据分析（NumPy、Pandas）
  - 机器学习（Scikit-learn、TensorFlow）
  - Web开发（Django、Flask）

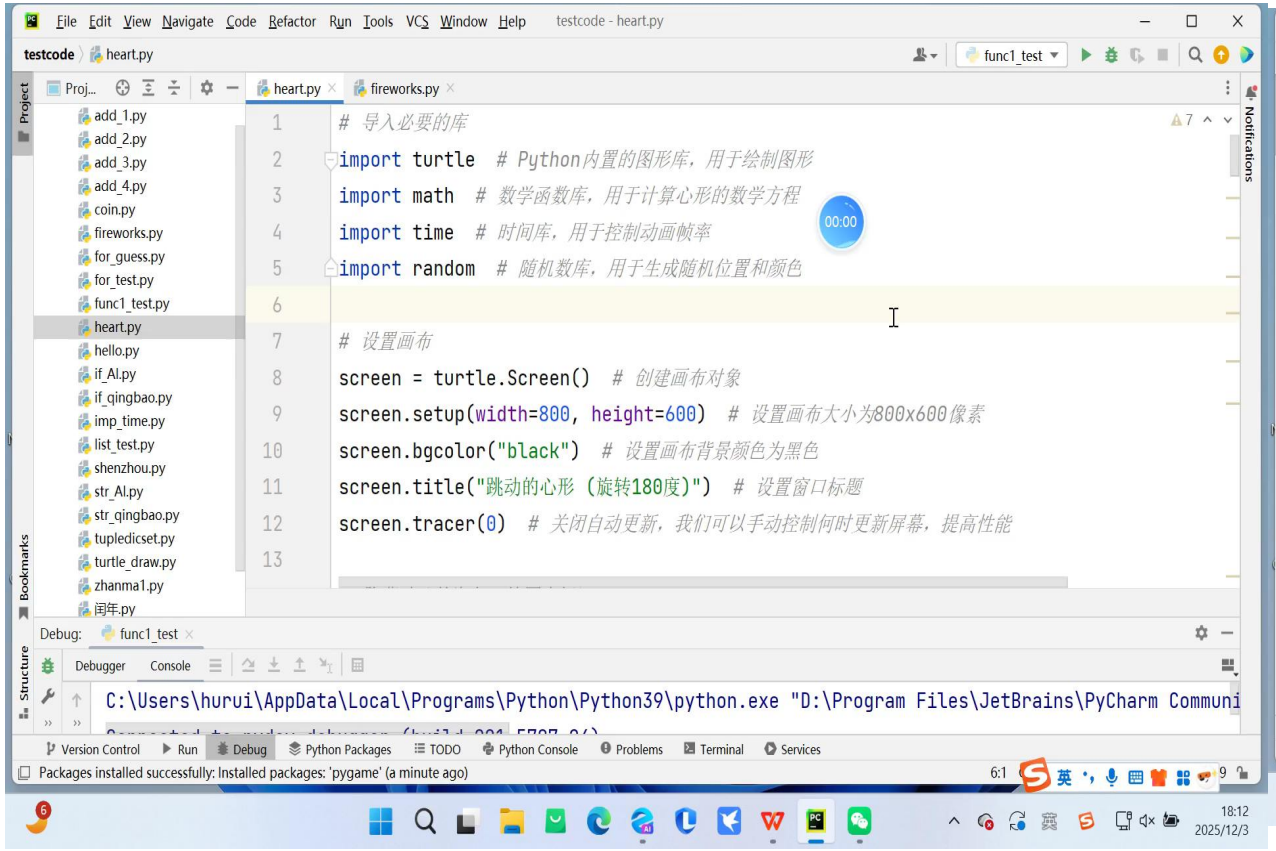


- 库的建设经过野蛮生长和自然选择

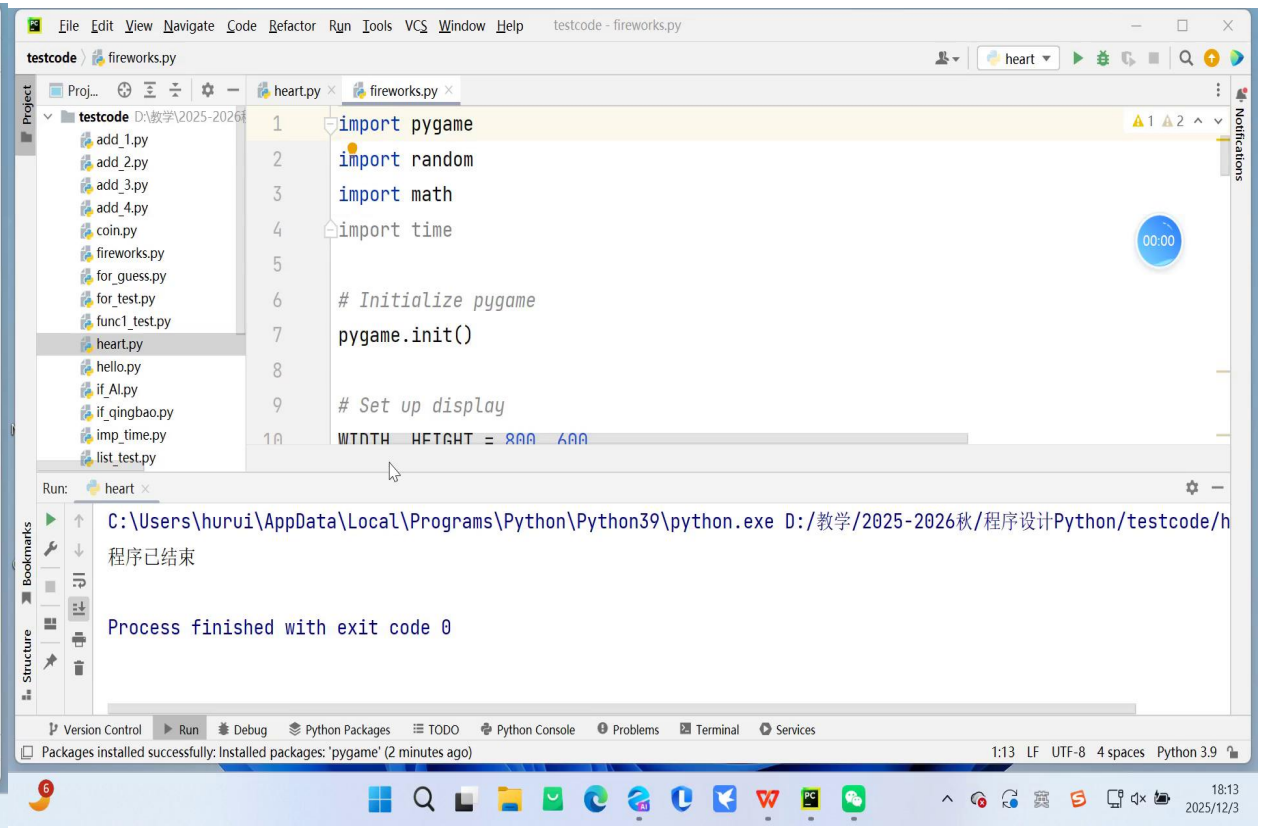
  同一个功能，Python语言2个以上第三方库

- 你不用从零造轮子，只需用Python胶水把现成的优秀库粘合起来，就能快速搭建复杂系统。这种生态让Python成为最易用、最高效的编程语言之一。

□跳动的心


testcode - heart.py
testcode
heart.py
func1_test
fireworks.py
# 导入必要的库
import turtle # Python内置的图形库，用于绘制图形
import math # 数学函数库，用于计算心形的数学方程
import time # 时间库，用于控制动画帧率
import random # 随机数库，用于生成随机位置和颜色
# 设置画布
screen = turtle.Screen() # 创建画布对象
screen.setup(width=800, height=600) # 设置画布大小为800x600像素
screen.bgcolor("black") # 设置画布背景颜色为黑色
screen.title("跳动的心形（旋转180度）") # 设置窗口标题
screen.tracer(0) # 关闭自动更新，我们可以手动控制何时更新屏幕，提高性能
Debug: func1_test
C:\Users\hurui\AppData\Local\Programs\Python\Python39\python.exe "D:\Program Files\JetBrains\PyCharm Communi
Packages installed successfully: Installed packages: 'pygame' (a minute ago)

□浪漫的烟花


testcode - fireworks.py
testcode
fireworks.py
heart
import pygame
import random
import math
import time
# Initialize pygame
pygame.init()
# Set up display
C:\Users\hurui\AppData\Local\Programs\Python\Python39\python.exe D:/教学/2025-2026秋/程序设计Python/testcode/h
程序已结束
Process finished with exit code 0
Packages installed successfully: Installed packages: 'pygame' (2 minutes ago)

Python模块如何用?

步骤：先导入，再使用

3种使用方法：

1、导入方法：import 模块名

使用方法：模块名.函数名()，可以使用模块中的所有函数

2、导入方法：from 模块名 import 函数名，函数名.....,函数名

使用方法：函数名()，可以使用模块中导入的函数

3、导入方法： from 模块名 import * 其中*是通配符，表示所有函数

使用方法：函数名()，可以使用模块中所有的函数

（1）Python标准库使用示例

```
import random#第一种方法导入math模块
r=random.randint(1,10)# 使用方法：模块名.函数名（）
print(r)

from random import * 第二种方法导入math模块
r=randint(1,10)# 使用方法：函数名（）
print(r)
```

（2）Python第三方库使用示例

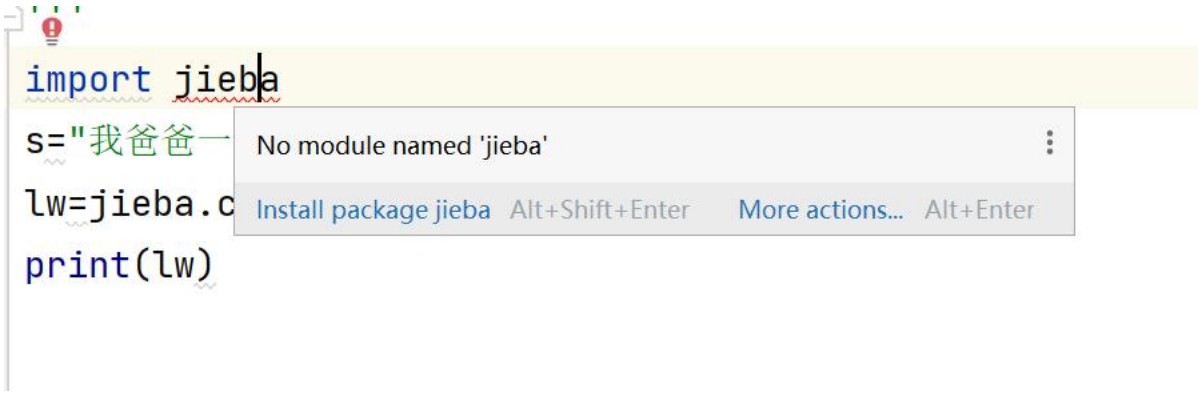
中文分词库，先安装

```
import jieba
s="我爸爸一把把把把住了"
lw=jieba.cut(s)
print("/".join(lw))
```

我/爸爸/一把/把/把/把住/了

```
s="我也想过过过儿过过的生活"
```

我/也/想/过/过/过儿/过过/的/生活



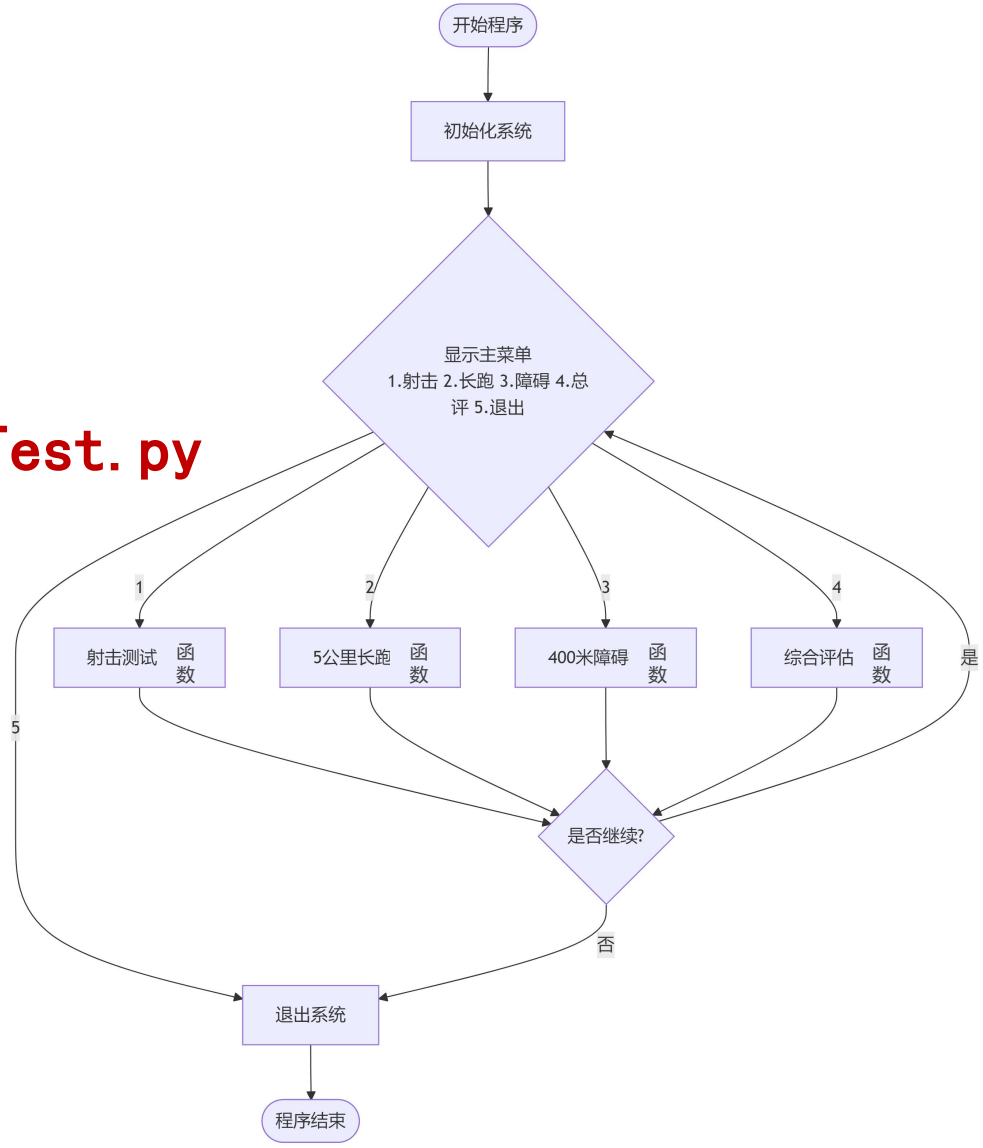
其他安装方法，自行查阅资料

（3）Python自定义模块使用示例

```
==============================
     体能测试管理系统
==============================
1. 射击测试
2. 5公里长跑测试
3. 400米障碍测试
4. 查看总评
5. 退出
------------------------------
请选择 (1-5) : 1

=== 射击测试 ===
说明：满环10环，9环以上优秀
请输入射击成绩 (0-10环) : 9.5
你的成绩：9.5环
等级：优秀
```

模块文件PhysicalTest.py



（3）Python自定义模块使用示例

模块文件PhysicalTest.py

```
def shooting_test():
    """射击测试：输入环数，输出等级"""
    global shooting_score
    ......
def running_5k_test():
    """5公里长跑：记录每圈用时，计算总时间"""
    global running_times
    ......
def obstacle_400_test():
    """400米障碍：记录完成时间"""
    global obstacle_time
    ......
```

模块文件：多个函数的文件

主文件：调用模块中函数

PhysicalMain.py

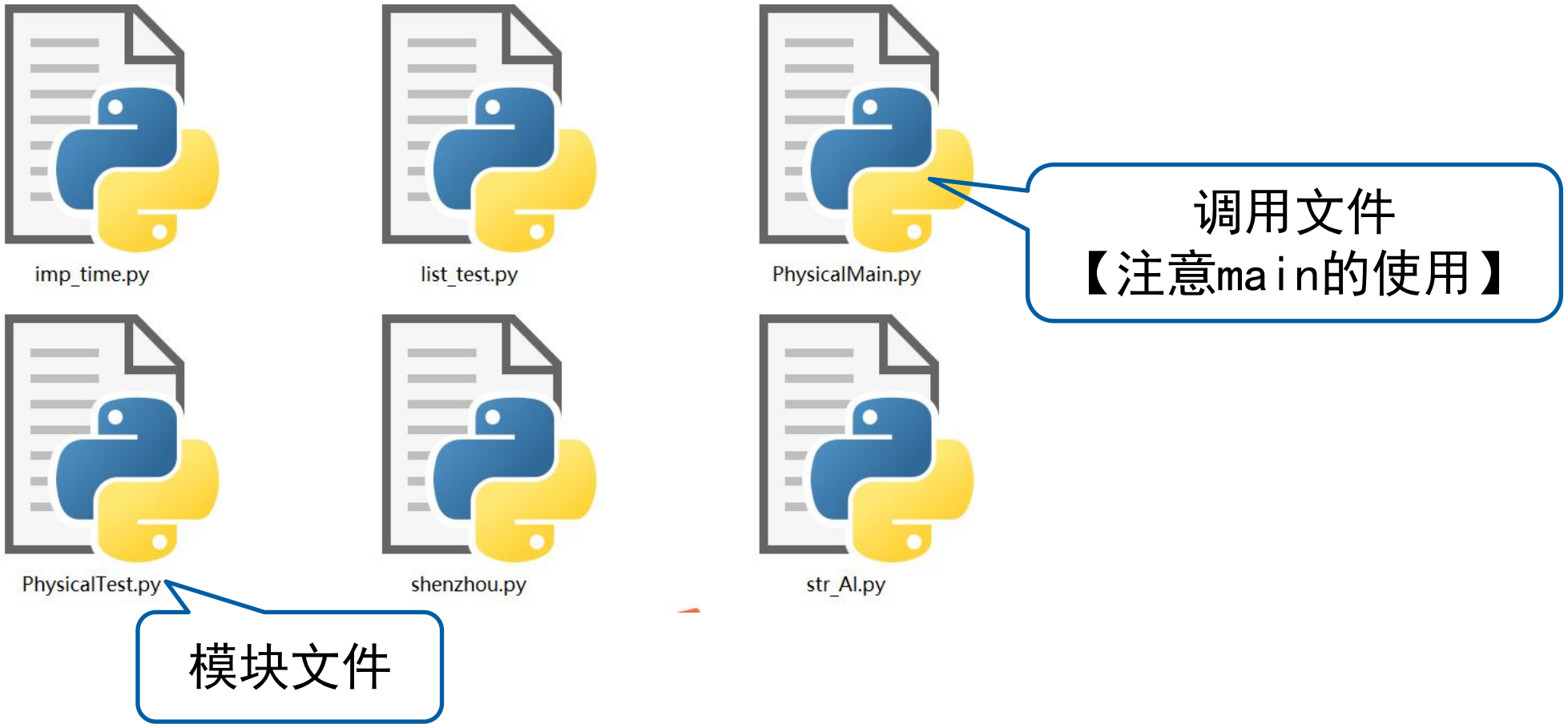
```
import PhysicalTest
PhysicalTest.shooting_test()
```

```
from PhysicalTest import *
shooting_test()
```

自定义模块建议和主程序文件放在同目录

先导入，再使用

强调：先有模块文件，才能调用（即：先写模块文件，再写调用文件）



PhysicalMain.py

```
def main():
    """主程序"""
    print("欢迎使用体能测试管理系统")

    while True:
        show_menu()
        choice = input("请选择（1-5）：")
        if choice == '1':
            PhysicalTest.shooting_test()  # 使用test.前缀
        elif choice == '2':
            PhysicalTest.running_5k_test()
```

在Python中，main()函数并不是语言内置的关键字或语法结构，而是一种约定俗成的编程习惯。通常，在脚本的末尾定义一个main()函数，并通过以下代码块来调用它：

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

main()函数的作用：
1. 程序入口点：明确地指定程序的入口点，使得代码更加清晰。
2. 模块化编程：Python强调代码的模块化和复用性。将程序的主逻辑封装在main()函数中，可以将业务逻辑与其他辅助功能分离，提高代码的可读性和可维护性。

这段代码的意思是：如果当前模块是作为主程序运行的（而不是被其他模块导入），则执行main()函数。

Python模块使用步骤总结

步骤一：导入标准库

步骤二：使用标准库

步骤一：安装第三方库

步骤二：导入第三方库

步骤三：使用第三方库

步骤一：定义模块（同目录）

步骤二：导入自定义模块

步骤三：使用自定义模块

模块化设计

分而治之

- 通过**函数**将程序划分为**模块及模块间**的表达
- 具体包括：**主程序、子程序和子程序间关系**
- 分而治之：一种分层抽象、体系化的设计思想

模块化设计

高内聚 低耦合

- **紧耦合（高内聚）：两个部分之间交流很多，无法独立存在**
- **松耦合（低耦合）：两个部分之间交流较少，可以独立存在**
- **模块内部高内聚、模块之间低耦合**

总结

➢ 函数递归

- 递归基：必须有终止条件（如 n==0）
- 递归步：问题规模的逐渐缩小

➢ 模块

- 高内聚：每个模块专注一个功能
- 低耦合：模块间依赖最小化
- 可重用：像标准零件随处可用

感悟

关于递归：

“递归不是无限循环，而是向着目标的有限累积——人生亦如是，每天进步一点点，终会抵达梦想的彼岸。”

关于模块化：

“没有人是全能的超人，但每个人都可以成为优秀的‘模块’——在自己的领域做到极致，然后通过清晰的‘接口’与世界协作。”

单选题 1分

设置

下面哪个函数调用会因类型不一致而导致错误?

```
def add(a, b):
    return a + b
```

A **add(5, 3.2)**

B **add("Hello", "World")**

C **add([1,2], [3,4])**

D **add(5, "3")**

提交

单选题 1分

设置

```
a=[1,2]
b=a
b.append(3)
print(a)
```

```
a=1
b=a
b=3
print(a)
```

这两个程序的输出结果是?

A [1,2] 1

B [1,2,3] 1

C [1， 2] 3

D [1， 2， 3] 3

提交

单选题 1分

设置

右边程序的输出结果是?

A
0
0

B
20
0

C
20
20

D
0
20

```
total = 0
def  mul(a1, a2):
    total = a1 * a2
    print(total)
mul(5, 4)
print(total)
```

提交

单选题 1分

设置

程序的运行结果是（ ）。

```
n=1
def func(a,b):
    c=a*b
    return c
s=func("knock~",2)
print(c)
```

A knockknock

B 程序报错

提交

下课并不代表思考的终止
期待我们下次的思想碰撞