

第五章 函数





主要内容

- 函数的定义
- 内置函数
- 参数及返回值
- 变量作用域
- 模块化编程 (*)
- 递归





传统做面包过程：揉面，发酵，烘烤



面包机做面包



传统扫地



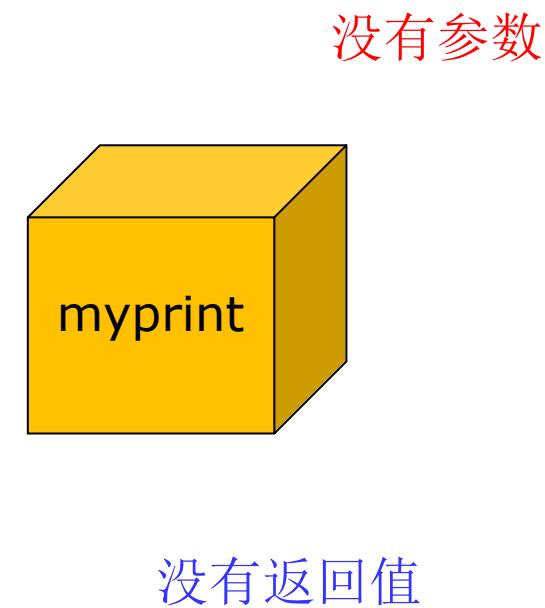
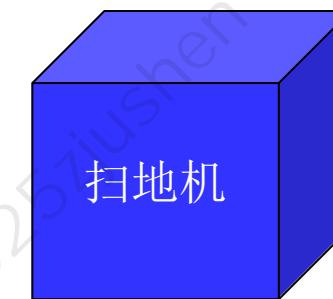
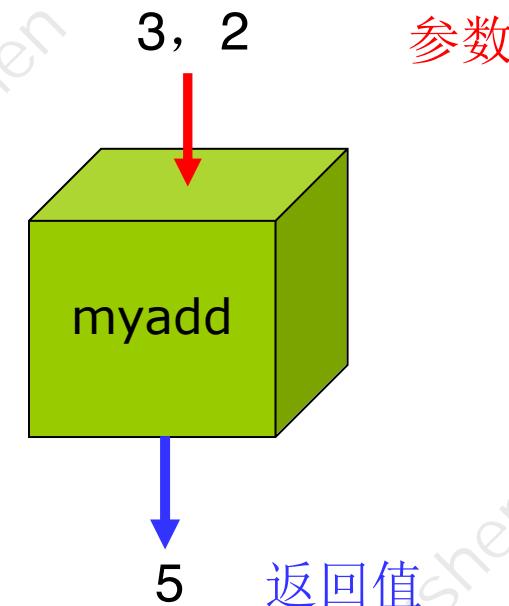
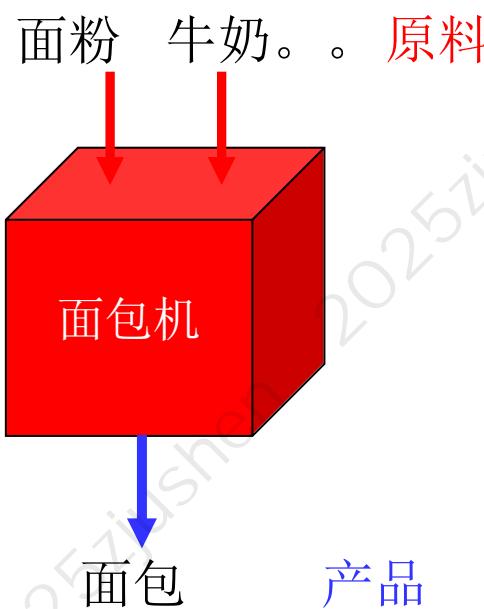
扫地机器人扫地





什么是函数

- 函数是具有名称的代码块，用于封装一组相关指令以实现特定功能。
- 在程序的不同位置通过该名称多次调用（重用）。
- 函数可接收输入参数，执行内部逻辑，并返回结果。





函数种类

内置函数	Python 的标准库包含许多内置函数。Python 的一些内置函数包括 print()、int()、len()、sum() 等。这些函数始终可用，因为它们会在启动 Python 解释器时立即加载到计算机内存中。
内置模块中的函数	标准库还捆绑了许多模块。每个模块都定义了一组函数。这些函数需要从各自的模块将它们导入到内存中后使用。
用户自定义函数	除了内置函数和内置模块中的函数外，自己定义创建的函数。





函数是一等公民

这是Python设计中的一个**核心**概念，如何理解？

函数与其他基本数据类型（如整数、字符串、列表等）同等的地位和操作自由度。

- 赋值给变量
- 作为参数传递
- 作为返回值
- 存储在数据结构中

这种特性是现代编程语言强大表现力的基石，使得开发者能够以更简洁、抽象的方式编写代码，同时为函数式编程、动态行为扩展等高级技术提供了基础。



内置函数





Python内置函数

■ Python解释器提供了很多内置函数 (<https://docs.python.org/zh-cn/3/library/functions.html>)

内置函数			
A	E	L	R
abs() aiter() all() anext() any() ascii()	enumerate() eval() exec()	len() list() locals()	range() repr() reversed() round()
B	F	M	S
bin() bool() breakpoint() bytearray() bytes()	filter() float() format() frozenset()	map() max() memoryview() min()	set() setattr() slice() sorted() staticmethod() str() sum()
C	G	N	O
callable() chr() classmethod() compile() complex()	getattr() globals()	next()	object() oct() open() ord()
D	H	P	T
delattr() dict() dir() divmod()	hasattr() hash() help() hex()	pow() print() property()	tuple() type()
I	I	V	Z
	id() input() int() isinstance() issubclass() iter()	vars()	zip()
		-	__import__()

涵盖了多个方面：

- 数学运算
- 类型转换
- 序列操作
- 输入输出
- 逻辑判断





enumerate()

enumerate(iterable, start=0)

- 用于在循环遍历可迭代对象（如列表、字符串、元组等）时，同时获取元素的索引和值。

```
for i,fruit in enumerate(["苹果","香蕉"],start=1):  
    print(f"第{i}个水果 : {fruit}")
```

第1个水果：苹果
第2个水果：香蕉

```
{i: char for i, char in enumerate("Python")}
```

```
{0: 'P', 1: 'y', 2: 't', 3: 'h', 4: 'o', 5: 'n'}
```

- 掌握 `enumerate()` 能显著提升代码的简洁性和可读性，是 Pythonic 编程的重要技巧之一





sorted()

sorted(iterable[,key[, reverse]])

- **iterable** – 可迭代对象，如字符串，列表，元组、字典、集合等。
- **key** -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是来自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。
- **reverse** -- 排序规则
reverse = True 降序， **reverse = False** 升序（默认）。
- 返回值：返回新列表，原可迭代对象不会被修改。





sorted使用举例

- 若： students = [('Mike',89, 15), ('Linda',80, 14), ('Sean', 85, 14)], 三个分量分别为姓名、成绩、年龄

```
sorted(students, key=lambda s : s[1], reverse=True))
```

- 若： a={1:4,4:2,3:8,0:9}

```
sorted(a.items(),key=lambda x : x[0], reverse=True)
```

- 若： ['apple', 'banana', 'orange', 'pear']

```
sorted(words,key=len)
```

```
sorted(words,key=lambda x:x.count('a'))
```





zip()

- 用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的迭代器。

若： a = [1,2,3] b = [4,5,6] c = [4,5,6,7,8]

■ list(zip(a,b)) [(1, 4), (2, 5), (3, 6)]

■ (list(zip(a,c)) # 元素个数与最短的列表一致 [(1, 4), (2, 5), (3, 6)]

用途：

转置矩阵（行列互换）：

```
matrix = [ [1, 2, 3],  
          [4, 5, 6],  
          [7, 8, 9]  
 ]  
transposed = list(zip(*matrix))  
[(1,4,7), (2,5,8), (3,6,9)]
```

重新构建字典：

```
d={'blue':500,'red':100,'white':300}  
d1=dict(zip(d.values(),d.keys()))  
{500: 'blue', 100: 'red', 300: 'white'}
```





eval()和exec()

1. eval()

- 功能：计算并返回表达式的值
- 输入：字符串形式的表达式
- 返回值：表达式的计算结果

```
result = eval("3 + 4 * 2") # 返回 11  
x = 5  
  
result = eval("x * 2")    # 返回 10  
a = eval("[1,2,3]")     #a的类型是list
```

2. exec()

- 功能：执行Python代码
- 输入：字符串形式的代码块（可以是多行）
- 返回值：总是返回 None

存在安全风险，需要谨慎使用

```
exec("x = 10 + 20") # 执行赋值
```





isinstance()

isinstance(object, classinfo)

- 用于检查对象类型，能够判断一个对象是否属于某个类或其子类的实例。它在类型验证、多态处理和代码健壮性中扮演重要角色。

```
d=eval(input())
s=0
for i in d:
    if isinstance(i,(int,float)):
        s=s+i
    elif isinstance(i,list):
        s=s+sum(i)
print(s)
```





all()和any()

- all()和any()用于检查可迭代对象中的元素是否满足特定条件。
- all()当可迭代对象中所有元素都为真（或可迭代对象为空）时返回True，否则返回False。（可以看作多个参数的与操作）
- any()当可迭代对象中任一元素为真时返回True，如果可迭代对象为空则返回False。（可以看作多个参数的或操作）

示例：

n = 47

all([n%k for k in range(2,n)]) #True 思考：当结果为True代表什么？





★ 高阶函数：map()

- 用于对可迭代对象中的每个元素应用一个指定的函数，并返回一个迭代器（map 对象）

```
list(map(int,input().split())) #输入1, 2, 3, 4, 5  
[1, 2, 3, 4, 5]
```

```
list(map(lambda x: x ** 2, [1, 2, 3, 4, 5]))  
[1, 4, 9, 16, 25]
```

```
list(map(lambda x, y: x + y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))  
[3, 7, 11, 15, 19]
```



自定义函数



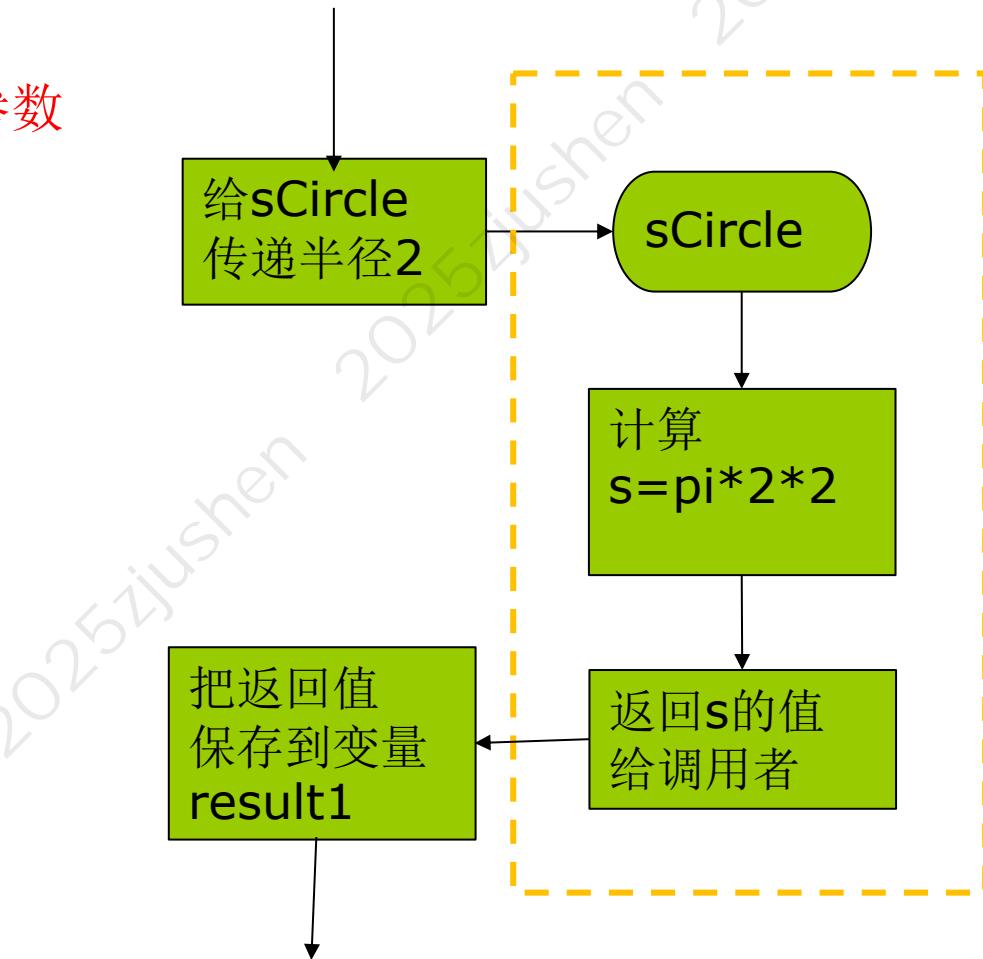
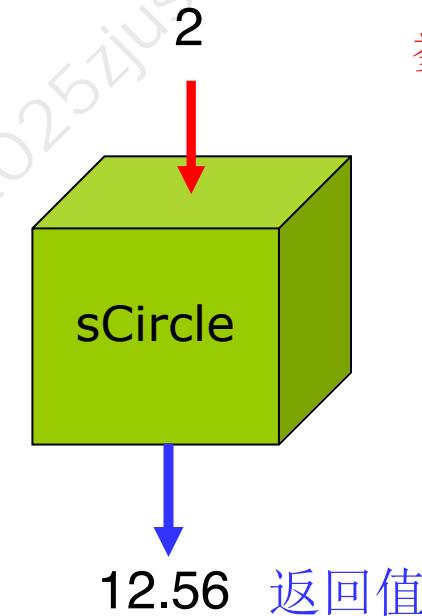


一个实例

编写一个函数计算给定半径的圆的面积(保留两位小数输出)。

步骤：

- 给函数取个名字
- 定义参数
- 实现面积计算
- 返回结果





函数的定义

关键字 函数名 形式参数

↓ ↓ /

函数头 [**def** **sCircle(r):** ← 冒号不能少
 [**s = math.pi * r * r**
]
函数体 [**return s**

退出函数
并返回结果





■ 语法形式如下：

```
def <函数名>(<参数>):  
    <函数体>  
    return <返回值>
```

■ 注意事项

- 一对圆括号是函数的重要部分，对即使该函数不需要接收任何参数，也必须保留一对空的圆括号
- 括号后面的冒号必不可少
- 函数体相对于**def**关键字必须进行缩进
- 如果没有返回值，**return**语句可以缺省，默认返回**None**
- Python允许嵌套定义函数

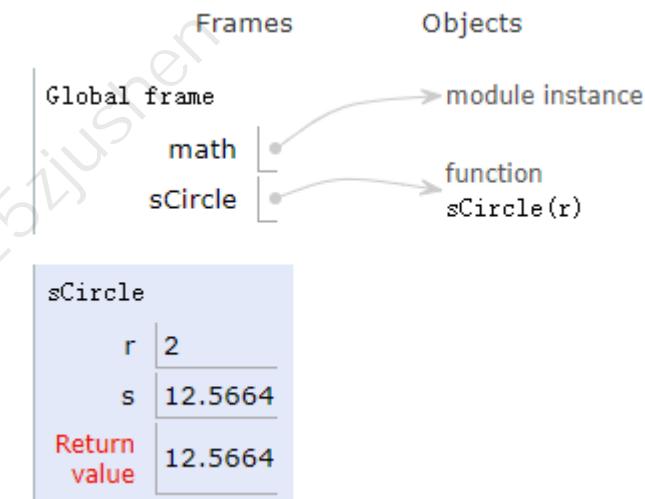
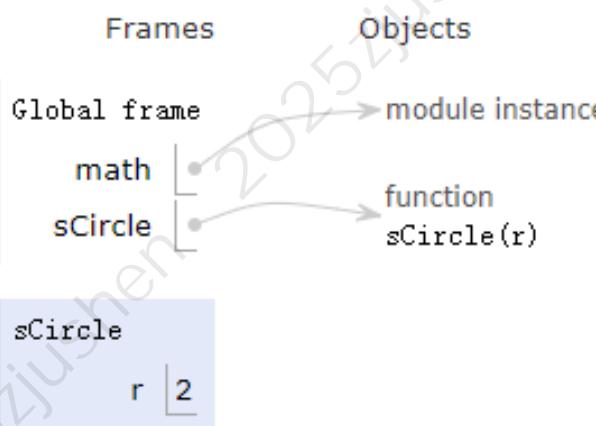




函数的调用

■ 函数通过函数名调用：

```
result1 = sCircle(2)           实际参数  
print(f"圆面积为:{result1:.2f}")
```





函数的术语

函数名	面包机	sCircle
功能定义		<pre>def sCircle(r): s = math.pi * r * r return s</pre>
形式参数	放原料的格子	r
实际参数	巧克力、面粉、酵母...	2
参数的个数	可以多种原料	可以0、1或者多个
调用	按下按钮	sCircle(2)
返回值	一个面包	s



参数和返回值





参数传递

没有参数的函数

```
1 def draw():
2     print('*****')
3     print('*      *')
4     print('*****')
5
6 draw()
```

有参数的函数

```
1 def draw(a,b): ← 形式参数
2     print('*'*a)
3     for _ in range(b-2):
4         print(f'*{"*"}{"*"(a-2)}{"*"}')
5     print('*'*a)
6
7 draw(10,5) ← 实际参数
```

对比两者，理解参数的意义

形参：parameter

实参：argument

传递机制：传递的是对象引用

- ① 若对象是可变类型（如列表、字典），在函数内修改对象内容会影响原始对象。
- ② 若对象是不可变类型（如整数、字符串、元组），无法直接修改对象内容，但重新赋值会创建新对象。





参数类型

位置参数（必需参数）

函数调用时，实参默认按位置顺序进行传递，并且要求个数和形参完全匹配。

关键字参数

为了避免位置参数带来的混乱，调用参数时可以指定对应参数的名字，它甚至可以采用与函数定义不同的顺序调用。

```
def demo1(name,age):
    print(f"My name is {name}.I am {age}.")
    return

demo1("Jack",20)
demo1(20,"Jack")
demo1(age=19,name="Tom")
```

My name is Jack.I am 20.
My name is 20.I am Jack.
My name is Tom.I am 19.





默认值参数

当调用方没有提供对应的参数值时，可以指定默认参数值。而如果提供了参数值，在调用时会代替默认值

```
def demo2(name,age=20):
    print(f"My name is {name}.I am {age}.")
return

demo2("Tom",19)
demo2("Jack")
```

```
My name is Tom.I am 19.
My name is Jack.I am 20.
```





不定长参数

当函数参数数目不确定时，星号将一组可变数量的位置参数变成参数值的容器

一个星号：作为元组接受数据

```
def demo3(a,*b):
    print(a)
    print(b)
    return

demo3(1,2,3,4)
```

```
1
(2, 3, 4)
```

两个星号：会接受一个字典类型

```
def demo4(a,**b):
    print(a)
    print(b)
    return

demo4(1,x=2,y=3,z=4)
```

```
1
{'x': 2, 'y': 3, 'z': 4}
```





返回值

函数的返回值可以看作是函数执行的结果，可以通过**return**语句实现返回

- 用于结束函数调用的执行，并将结果（**return** 关键字后面的表达式值）返回给调用方。可以返回一个值，也可以返回多个值。
- 一旦执行了**return** 语句，那么之后的语句将不会执行。
- 如果 **return** 语句没有任何表达式，则返回特殊值 **None**。
- 如果不出现**return**语句，那么同样返回特殊值 **None**。





习题讲解

6-1 使用函数求余弦函数的近似值 分数 10

查看题目详情 全屏浏览 切换布局

作者 陈春晖 单位 浙江大学

本题要求实现一个函数，用下列公式求 $\cos(x)$ 近似值，精确到最后一项的绝对值小于 eps （绝对值小于 eps 的项不要加）：

$$\cos(x) = \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

函数接口定义：`funcos(eps,x)`,其中用户传入的参数为`eps`和`x`；函数`funcos`应返回用给定公式计算出来，保留小数4位。

函数接口定义：

- 1 函数接口：
- 2 `funcos(eps,x)`,返回 $\cos(x)$ 的值。

裁判测试程序样例：

```
1 在这里给出函数被调用进行测试的例子。例如:  
2  
3  
4 /* 请在这里填写答案 */  
5  
6 eps,x=input().split()  
7 eps,x=float(eps),float(x)  
8 value=funcos(eps,x )  
9 print("cos({0}) = {1:.4f}".format(x,value))
```

输入样例：

0.0001 -3.1

函数题只需提交函数定义部分即可。

理解函数接口的意义

确定函数功能

确定函数参数

确定函数返回值



命名空间和作用域





命名空间

命名空间：Python程序中每个名字都会在于某一个空间，在这个空间中存在并被操作，互不影响。

- 局部命名空间（Local）
- 闭包命名空间（Enclosing）（不要求）
- 全局命名空间（Global）
- 内建命名空间（Built-in）

加载的顺序：

解释器启动—>内建命名空间—>加载模块—>全局命名空间—>函数调用—>
局部命名空间或闭包命名空间





作用域 (Scope)

作用域是变量在代码中的 可见性范围，由命名空间的层级关系决定。Python 遵循 **LEGB** 规则（查找顺序）：

- 1. L (Local)** : 当前函数或方法的局部作用域。
- 2. E (Enclosing)** : 外层嵌套函数的作用域（闭包场景，不要求）。
- 3. G (Global)** : 模块级别的全局作用域。
- 4. B (Built-in)** : 内建作用域。

若未找到，触发 `NameError`





一个程序中的变量包括两类：全局变量和局部变量。

■ 全局变量

定义：在函数或类外部定义的变量是全局变量，作用域为当前模块。

作用域：在模块内可直接读取，函数内修改需 **global** 声明；局部同名变量会屏蔽全局变量，但不影响全局变量的值。

生命周期：从模块加载到程序结束或模块卸载。

■ 局部变量

定义：在函数内部定义的变量（包括参数），只能在函数内部被访问。

作用域：默认限定在它们被定义的函数内部。

生命周期：从函数开始执行时开始，到函数执行结束结束。





示例

```
1 x = 10
2
3 def my_function():
4     x = 20 #局部变量x, 同名全局变量被屏蔽
5     print(x)
6
7 my_function() # 输出20
8 print(x) # 仍然输出10, 全局变量x的值未被修改
```

>>>

```
20
10
```

```
1 x = 10 #全局变量x
2
3 def my_function():
4     global x # 声明x是一个全局变量
5     x = 20
6     print(x)
7
8 my_function() # 输出20
9 print(x) # 输出20, 全局变量x的值已经被修改
```

>>>

```
20
20
```





示例

```
1 def modify_string(s):
2     # 尝试修改字符串 (实际上是创建了一个新的字符串)
3     s = s + " world"
4
5 my_string = "Hello"
6 modify_string(my_string)
7 print(my_string)
```

```
1 def modify_list(lst):
2     # 向传入的列表添加元素
3     lst.append(4)
4
5 my_list = [1, 2, 3]
6 modify_list(my_list)
7 print(my_list)
```

- 不可变类型：
函数内部不能修改原始对象，而是会创建新的对象（如果需要修改的话）。

- 可变类型：
函数内部对参数的修改会影响到原始对象。

注意这种差异，它决定了函数是否会改变调用者传入的数据。
对于可变类型，如果不希望函数修改原始数据，需要在函数内部创建数据的副本。





6-1.下面程序的运行结果是 5 4 5 8

```
def scope():
    n=4
    m=5
    print(m,n,end=' ')
n=5
t=8
scope()
print(n,t)
```

6-2.下面程序的运行结果是 1 6

```
l=[1]
def scope1():
    l.append(6)
    print(*l)
scope1()
```

6-3.下面程序的运行结果是 20 20

```
a=10
def func():
    global a
    a=20
    print(a,end=' ')
func()
print(a)
```



模块化编程





■ 模块化编程的意义：复用、协作、维护

- 核心思想

- 拆分代码为独立单元
 - 提升复用性与可维护性

- 应用场景

- 大型项目开发
 - 团队协作
 - 开源库设计





模块 (Module) 与包 (Package)

模块定义

- .py 文件即模块
- 包含变量、函数、类

示例：

```
math_tools.py x
1 PI = 3.14159
2 def circle_area(radius):
3     return PI * radius ** 2

import math_tools
print(math_tools.PI)
print(math_tools.circle_area(3))
```

包的定义

- 含 __init__.py 的文件夹
- 多级嵌套包结构

示例：

```
project/
└── tools/
    └── __init__.py # 标记这是一个包
        ├── math_tools.py
        └── file_tools.py
    └── main.py
```



导入方式

- 全量导入: `import math_tools` → 通过`math_tools.PI`调用
- 精准导入: `from math_tools import PI` → 直接使用`PI`
- 别名导入: `import math_tools as mt` → 简化长模块名
- 包内导入: `from tools.math_tools import circle_area` → 多级包结构





标准模块

模块	核心功能	典型应用场景
math	数学运算	科学计算、几何问题
os	与操作系统交互	文件/目录管理、路径操作
sys	与 Python 解释器交互	标准输入/输出、命令行参数、模块路径
datetime	时间处理	时间计算、格式化显示
json	数据序列化	API 数据交换、配置文件读写
random	随机函数	数据科学、机器学习、模拟测试、密码生成

直接调用标准库，避免重复造轮子，这也是模块化思维的延伸



随机函数





随机函数模块 (random)

- 伪随机性: random 模块生成的随机数基于算法和初始种子值, 不是真正的物理随机数 (适合大多数非安全场景)。
- 可重复性: 通过设置种子 (seed), 可以复现相同的随机序列 (常用于测试和调试)。
- 应用场景: 游戏开发、随机抽样、模拟实验、密码学 (仅限非安全场景) 等。





随机函数库

功能	函数	典型应用场景
设置种子	<code>random.seed()</code>	实验复现、“42”梗
生成随机浮点数	<code>random()</code> , <code>uniform()</code>	模拟连续值（如温度、位置）
生成随机整数	<code>randint()</code> , <code>randrange()</code>	骰子游戏、随机抽样
打乱或选择序列	<code>shuffle()</code> , <code>choice()</code>	洗牌、随机抽奖
概率分布(*)	<code>gauss()</code> , <code>expovariate()</code>	科学计算、统计模拟
安全随机数(*)	<code>secrets</code> 模块	密码、令牌、密钥生成





示例

```
>>> import random  
  
>>> random.random()  
0.11529299890219902  
  
>>> random.uniform(1,10)  
4.6467045646433975  
  
>>> random.randint(20,30)  
20  
  
>>> random.randrange(10, 30, 2)  
24
```

```
>>> random.choice([3,78,43,7])  
3  
>>> lst=['A',1,78,'b']  
>>> random.shuffle(lst)  
>>> lst  
[1, 'b', 78, 'A']  
  
>>> random.sample([1,4,5,89,7],3)  
[7, 5, 4]  
  
>>> random.sample("This is a sample",5)  
['s', 'h', ' ', 'a', 'a']
```





例题：扔硬币

模拟掷硬币实验，统计正面向上的概率是多少？

思路：用随机函数模拟投掷硬币的过程

掷10000次硬币，正面向上用1表示，反面向上用0表示。



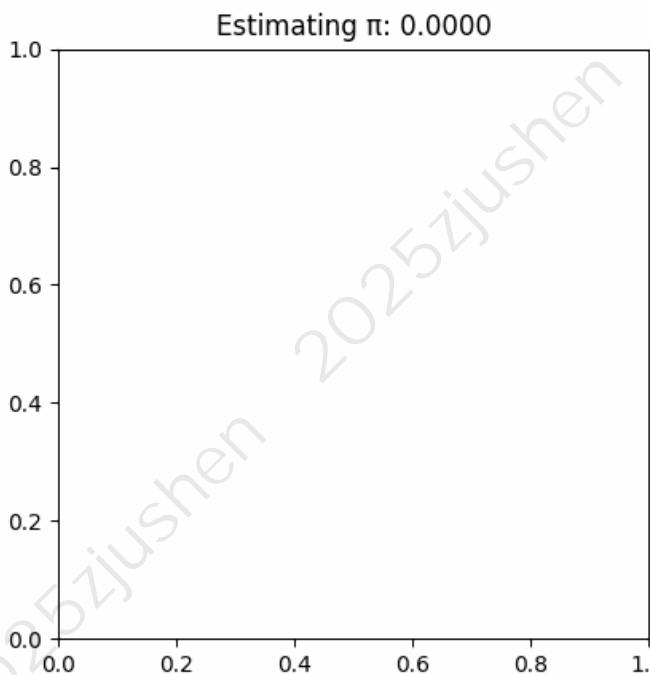
```
a.py - C:\Users\rshen\Desktop\a.py (3.8.1)
File Edit Format Run Options Window Help
import random
#产生10000个随机数，值为0或1
lst=[random.randint(0, 1) for i in range(10000)]
print("{:.1%}".format(sum(lst) / len(lst)))
Ln: 6 Col: 0
```





例题：蒙特卡罗法求圆周率

使用蒙特卡罗方法估算圆周率（ π ）是一种经典的随机模拟实验，通过几何概率和大数定律实现。主要通过在一个正方形内随机撒点，并判断这些点是否落在内切的圆内来估算 π 值



核心思想

1. 几何模型：在边长为2的正方形内画一个半径为1的圆（面积为 π ），圆心与正方形中心重合。
 - 正方形面积： $S_{\text{square}} = (2r)^2 = 4$ (当 $r = 1$ 时)。
 - 圆面积： $S_{\text{circle}} = \pi r^2 = \pi$ 。
2. 概率关系：随机向正方形内撒点，点落在圆内的概率为：

$$P = \frac{S_{\text{circle}}}{S_{\text{square}}} = \frac{\pi}{4}$$

⇒ 通过统计点的比例即可估算 π :

$$\pi \approx 4 \times \frac{\text{圆内点数}}{\text{总点数}}$$





具体步骤

■ 准备工作

- 构造几何形状：设定一个边长为1的正方形，其内部包含一个半径为1的 $1/4$ 圆
- 面积关系：根据几何知识， $1/4$ 圆的面积为 $S = \pi r^2 / 4 = \pi / 4$ ，而正方形的面积为 $r^2 = 1$ 。因此， $1/4$ 圆的面积与正方形面积之比为 $\pi / 4$ 。

■ 随机撒点

- 在正方形内随机生成 n 个点，这些点的坐标 (x, y) 均匀分布在区间 $[0, 1]$ 。
- 对每个点，计算其到原点的距离： $\text{distance} = x^2 + y^2$ 。

■ 点数统计

- 统计落在圆内的点数。若距离小于或等于1，则该点在圆内。设定一个计数器 `inside_circle` 来记录落在圆内的点数。

■ π 值估算

- 根据落在圆内的点数与总点数的比例来估算 π 值：
- 随着撒点数量 nn 的增加，估算结果会逐渐接近真实值。
- 理论上，撒点越多，计算出的 π 值越精确。





参考代码

```
1 import random
2 import math
3
4 inside_circle = 0
5 num_samples = 100000000 # 设置样本数量
6 for _ in range(num_samples):
7     # 在单位正方形内随机生成一个点 (x, y), 其中 x 和 y 的取值范围都是 [0, 1)
8     x = random.random()
9     y = random.random()
10
11     '''判断该点是否落在单位圆内 (半径为1的圆, 圆心在原点)
12     如果点 (x, y) 到原点的距离小于等于1, 则该点落在单位圆内'''
13     if x**2 + y**2 <= 1:
14         inside_circle += 1
15
16     '''根据落在单位圆内的点的比例估算圆周率π
17     单位正方形的面积为1, 单位圆的面积为π/4
18     因此, 落在单位圆内的点的比例应该近似等于π/4
19     '''
20 pi_estimate = (inside_circle / num_samples) * 4
21 print(f"使用{num_samples}个样本点估算的圆周率π值为: {pi_estimate}")
```





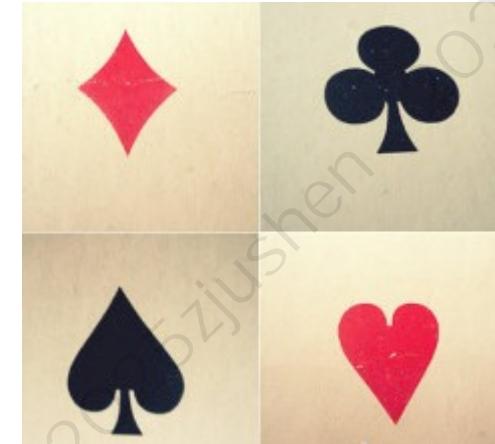
思考：随机抽取扑克牌

题目1. 模拟抽取一张扑克牌

题目2. 统计抽到红桃的概率

思路：用随机函数模拟抽牌过程

记录并统计





例题：生成随机密码

随机产生8位密码，密码由数字和字母组成。如：ItP6Q187

思路：生成密码字符池

随机选取

随机排列

```
import random
import string

digits=string.digits
ascii_letters=string.ascii_letters

num_of_numeric = random.randint(1, 7)
num_of_letter = 8 - num_of_numeric

numerics = random.sample(digits, num_of_numeric)
letters = random.sample(ascii_letters, num_of_letter)

all_chars = numerics + letters
random.shuffle(all_chars)
result = ''.join([i for i in all_chars])
print(result)
```



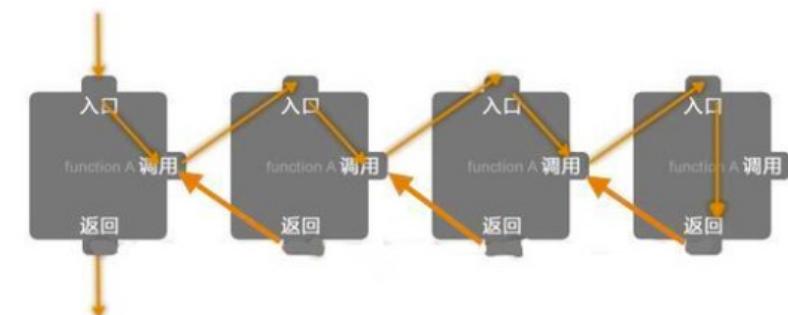
递归函数





递归的定义

- 函数调用自身的编程技巧称为递归
- 函数作为一种代码封装，可以被其他程序调用，当然，也可以被函数内部代码调用。递归在数学和计算机应用上非常强大，能够非常简洁的解决重要问题。
- 一般来说，递归需要**终止条件**和**递归条件**。当终止条件不满足时，递归前进；当终止条件满足时，递归返回。编写递归函数时，必须告诉它何时停止递归，直接返回结果，从而避免形成无限循环。



图片来自网络





经典示例：阶乘

- 数学上有个经典的递归例子叫阶乘，阶乘通常定义为：

$$n! = n(n-1)(n-2)\dots(1)$$

- 这个关系给出了另一种方式表达阶乘的方式：

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

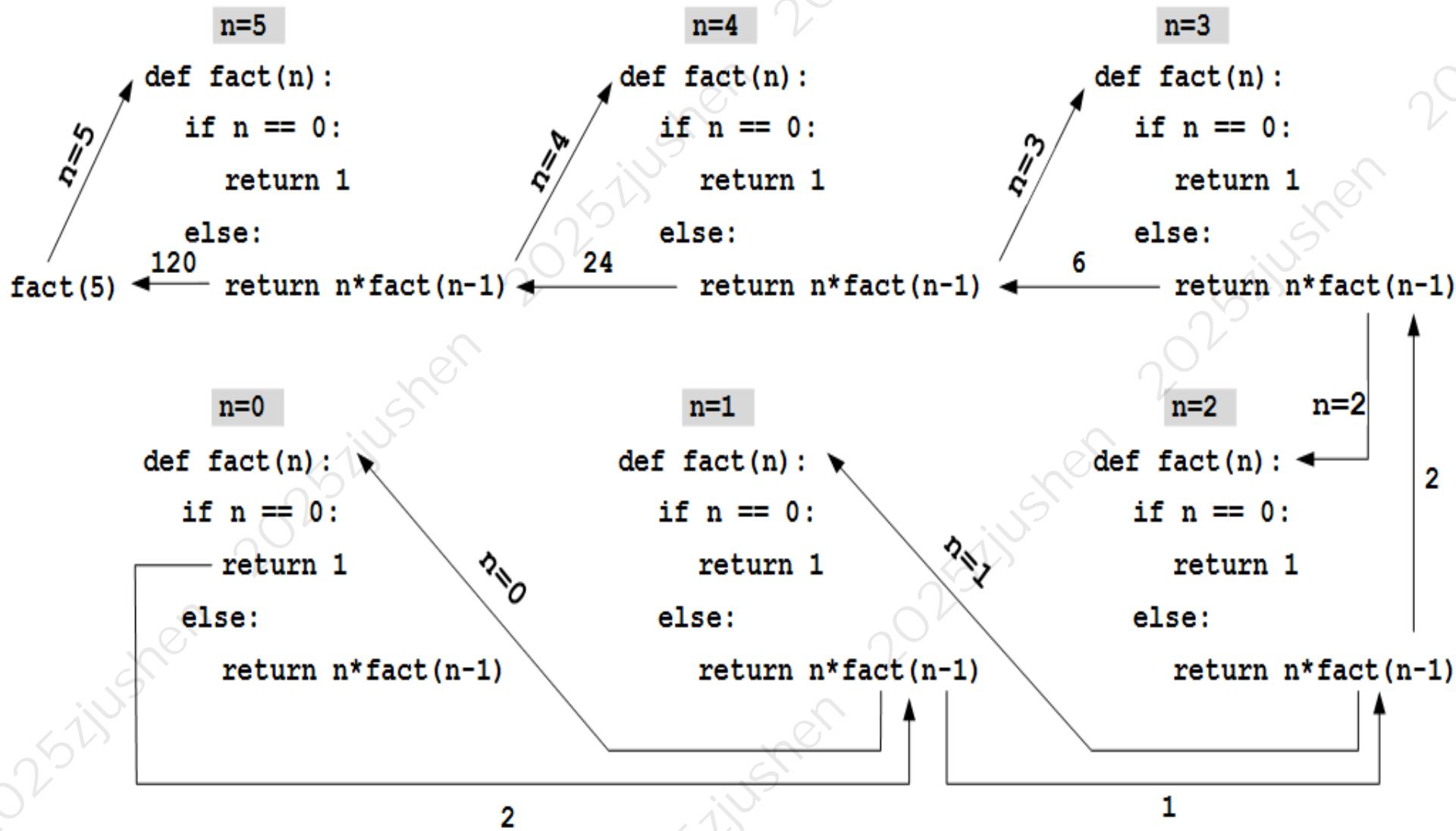
```
def factorial(number):
    product = 1
    for i in range(1, number+1):
        product = product * i
    return product
```

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```





递归的调用过程





设计递归的要点

要和编写循环区别，这是两种不同的思维方式，递归的核心在于把问题简化成更简单的问题。

- 找到递归式：把规模大的问题转化为规模小的相似的子问题来解决，往往是同一个方法。
- 找到递归出口：存在一个递归调用的终止条件 每次递归的调用必须越来越靠近终止条件，只有这样递归才会终止，否则是不能使用递归的！





求和

例：编写一个函数digitsum()求整数n的所有位之和

■ 方法一：

```
n = input()
result = sum([int(i) for i in n])
print(result)
```

■ 方法二（递归）：

```
def digitsum(n):
    if n==0:
        return 0
    else:
        return n%10+digitsum(n//10)

print(digitsum(123))
```





递归的时间开销

- 什么样的问题适合用递归：

例：求fib数列

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

```
def fib_loop(n):
    if n==1 or n==2:
        return 1
    else:
        i=2
        f1=f2=1
        while i < n:
            f1, f2 = f2, f1+f2
            i+=1
        return f2
```

子问题如果有重叠就不适合用递归，耗时耗资源





思考题

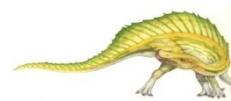
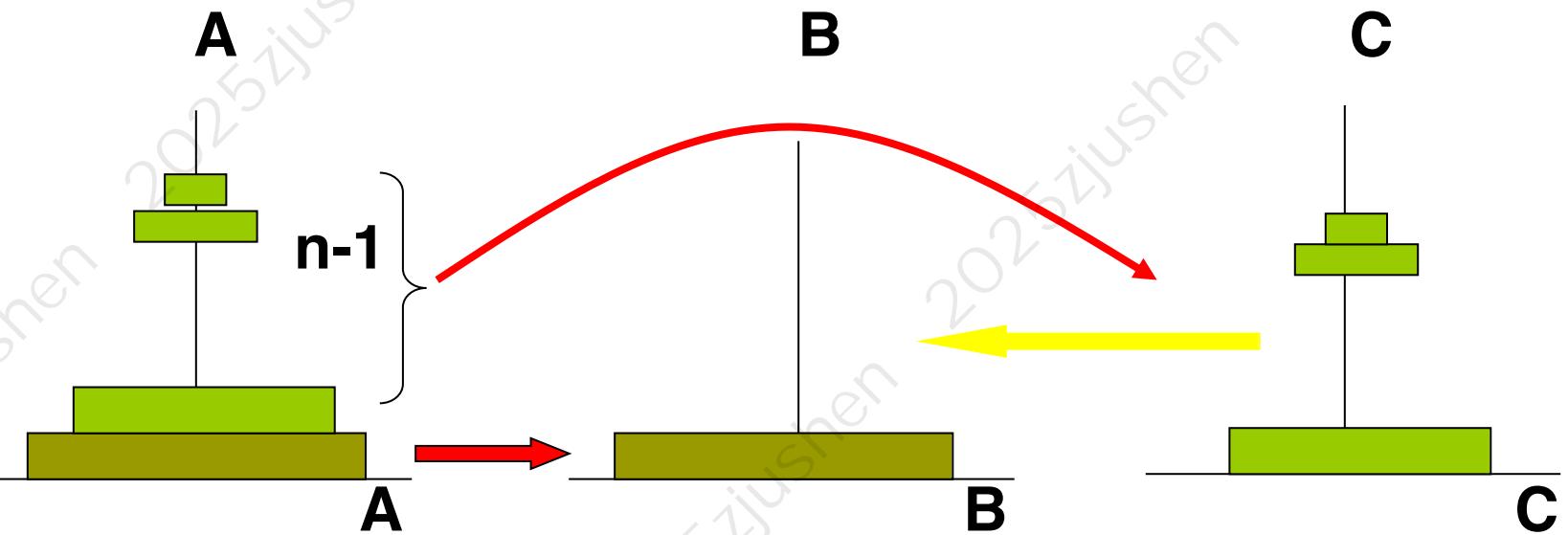
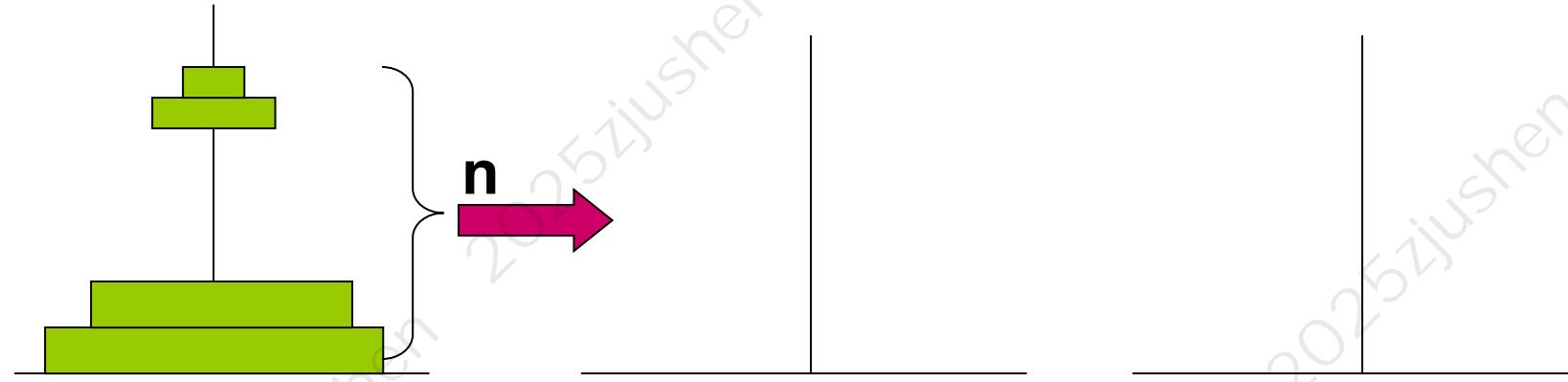
汉诺塔问题

■ 用递归的方法去解决





分析



END





Lambda匿名函数

`lambda [arg1 [,arg2,....argn]] : expression`

lambda是Python预留的关键字，arg和expression由用户自定义。通常用于编写简单的、只使用一次，而不需要正式定义的一个函数。

```
def func1(x):
    y = 2 * x + 5
    return y

print(func1(10))
```

```
func2 = lambda x : 2 * x + 5

print(func2(10))
```

```
def func1(a,b):
    y = a + b
    return y

print(func1(10,11))
```

```
func2 = lambda a,b : a + b

print(func2(10,11))
```

