LECTURE 9 · HANDOUT

Classes I: Objects and Encapsulation

Every slide from the lecture, with what was said over it. The slides are rendered natively — the text is selectable, searchable, and prints cleanly.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Introduction to Computation (CS1602) · Lecture 9

Objects and Encapsulation 对象与封装

Instructor: Tao Huang

Part 4: Abstraction and Organization · Fall 2026

1

Slide 1 · Objects and Encapsulation 对象与封装

> 上一讲通过规格、接口、边界和副作用检查程序。本讲继续研究数据与操作怎样组织，先观察一个对象怎样初始化和调用方法，再分析共享数据和有效状态。
>
> 类是组织方式，不会自动保证实现正确。讲义保留完整程序，课堂按主线理解调用与数据关系；三角形、向量等较长例子放在参考区。

Part IV · Abstraction and Organization

> 课程进入抽象与组织部分。L09 学习类、实例、初始化、方法绑定和状态规则；L10 再讨论继承、运算协议与模块。
>
> 先说明本讲边界，不提前使用后续语法。已经学过的函数仍是有效工具，今天增加一种组织多个记录和相关操作的方法。

Data, Interfaces and Verification 数据与接口

- **Contract 约定**: inputs and promised results
- **State 状态**: changes retained after a call
- **Evidence 证据**: examples, boundaries and invariants

Slide 3 · Data, Interfaces and Verification 数据与接口

> 回顾 L08 最后一页的数据与操作。一个接口不仅有返回值，也可能修改原对象、打印消息或拒绝某些输入。
>
> 本讲继续分别观察这些结果。使用类以后，原有的规格、边界和副作用检查仍然适用，不能因为程序看起来更有结构就跳过验证。

Two Position Records 两条位置记录

Record 记录	Coordinates 坐标	Tasks 操作
A	(0, 0)	Show the position; move A
B	(3, 4)	Measure the distance from A to B

先区分状态、操作与查询，不预先指定实现。

Slide 4 · Two Position Records 两条位置记录

> 先不看代码。两个点各有坐标，移动 A 后要保留新的位置，B 的坐标通常不应改变；距离查询需要读取两个点。
>
> 请学生说出哪些数据需要长期保存、哪些操作修改它们、哪些操作只查询。现在只描述任务，不提前要求理解 class 或 self。

Quiz 1 · Record Updates

扫码作答 · 1 题

1 普通函数：状态修改与返回值

Submit this round once. 按已学规则分析。

https://taohuang.info/cs1602/zh/quiz/9/

Slide 5 · Quiz 1 · Record Updates

> 开场题只使用已经教过的字典、函数调用、赋值和 print。预测原字典的 x 和函数返回值，不需要类的知识。
>
> 本题提交一次，先不揭晓，第 26 页用函数与方法的对应行为回顾原分布，不重新提交。

Lecture Structure

1	**Objects 对象**	Initialization, attributes and aliases 初始化与引用
2	**Methods 方法**	Receivers, locals and results 接收者与返回值
3	**Rules 规则**	Valid states and public interfaces 有效状态与接口
4	**Sharing 共享**	Class attributes and method kinds 类属性与方法类别

6

Slide 6 · Lecture Structure

> 顺序从创建记录、调用操作推进到验证状态，再比较哪些数据或操作属于类级共享。前一段得到的规则支持后一段的问题。
>
> 不重复独立 AI 政策页，实验中保留实际使用声明。下面的例子先从已有字典与函数开始。

I · Objects and Initialization

> 函数已经可以接受字典记录，并隔离局部参数。引入类的目的不是否定这些写法，而是为一类实例组织共同接口和各自状态。
>
> 先看类、属性和初始化，再区分别名与新实例。没有初始化的属性与非法数值是两个不同问题。

Records and Operations 记录与操作

Coordinates and related operations

Two dictionary records

```
a = {"x": 0, "y": 0}
b = {"x": 3, "y": 4}
```

- **Show** a position
- **Move** one record
- **Measure** a distance

> 类不能代替公式与输入检查。

Slide 8 · Records and Operations 记录与操作

> 字典版本把两个坐标放在同一个记录中，distance 接收两份记录，实际计算得到 5.0。讲义中提供完整独立程序。
>
> 类可以使相关操作的接口更集中，但不能因此声称字典版本不正确、方法调用必然更自然或类更快。后面始终保留这种选择依据。

Existing Types and Methods 既有类型与方法

code

```
print(type(42), type("abc"))
text = "hello world"
print(text.count("o"))
items = [3, 1, 2]
print(items.reverse())
print(items)
```

output

```
<class 'int'> <class 'str'>
2
None
[2, 1, 3]
```


Slide 9 · Existing Types and Methods 既有类型与方法

> 学生已经使用整数、字符串和列表对象。type 返回运行时类型，字符串 count 是通过对象访问的方法。
>
> 列表 reverse 返回 None，但列表实际变成 [2, 1, 3]。这仍是返回值与副作用的区分，不把 None 理解为没有执行。

Classes and Instances 类与实例

code

```
class Point:
    pass

p = Point()
print(type(p))
print(type(p) is Point)
```

output

```
<class '__main__.Point'>
True
```


Slide 10 · Classes and Instances 类与实例

> class 语句定义一种对象，Point() 创建一个实例。pass 是空操作，用在这里需要语句块但尚无操作的位置。
>
> type(p) 是 Point，身份比较为 True。类本身也可以被名字引用；默认对象地址或模块显示与执行环境有关，不把它们当成固定预期值。

Instance Attributes 实例属性

code

```python
class Point:
    pass

x = 100
p = Point()
p.x = 3
p.y = 4
print(p.x, p.y, x)
```

output

```
3 4 100
```


Slide 11 · Instance Attributes 实例属性

> 本讲的普通类允许通过 p.x = 3 增加实例属性。p.x 与外部变量 x 是不同绑定位置，真实输出为 3、4、100。
>
> 先明确点号选择了哪个对象，再谈属性名字。这里没有声称所有 Python 类都支持任意动态属性；特殊限制留到后续机制。

Missing Attributes 缺失属性

Predict both accesses before advancing. 先预测两次访问。

```
class Point:
    pass

p = Point()
p.x = 3
q = Point()
print(p.x)
print(q.x)
```


Slide 12 · Missing Attributes 缺失属性

> 先预测哪一次访问成功，哪一次报错。p 和 q 都由同一个类创建，但只有 p 被添加了 x。
>
> 不要在这页显示输出。下一页先用实际结果说明 q 缺属性，再解释为什么希望初始化建立共同字段。

Initialization with __init__ 初始化

code

```
class Point:
    def __init__(self, x: float = 0, y: float = 0) -> None:
        self.x = x
        self.y = y

p = Point(3, 4)
print(p.x, p.y)
```

output

```
3 4
```


Slide 13 · Initialization with __init__ 初始化

> 先回顾上一页：p.x 打印 3，q.x 触发 AttributeError。相同类的实例不自动复制别人的字段。
>
> 现在 __init__ 接收新实例，惯用参数名 self。参数 x 是当前调用的局部名字，self.x = x 则给新实例建立属性；y 同理。先解释接收者，再使用代码，不把注解当作验证。

Default Coordinates 默认参数

code

```python
class Point:
    def __init__(self, x: float = 0, y: float = 0) -> None:
        self.x = x
        self.y = y

for p in [Point(), Point(3), Point(3, 4)]:
    print(p.x, p.y)
```

output

```
0 0
3 0
3 4
```


Slide 14 · Default Coordinates 默认参数

> 三个调用依次使用默认或显式参数，真实坐标为 0 0、3 0、3 4。规则与普通函数默认参数相同。
>
> 零坐标在本任务中合法，不意味着每个类都应该用零作默认值。参考区会核对旧三角形零边默认状态。

Class Body and Initialization 类体与初始化

How often does each message appear? 各打印几次?

```
class Trace:
    print("class body")
    def __init__(self) -> None:
        print("init")

a = Trace()
b = Trace()
```


Slide 15 · Class Body and Initialization 类体与初始化

> 先问 class body 和 init 各打印几次。class 语句运行类体，其中 def 建立函数，不立即执行函数体。
>
> 这页只呈现源码和问题，实际顺序写在下一页的静态内容中，不留下依赖 JavaScript 才出现的关键答案。

Class Execution and Initialization Results

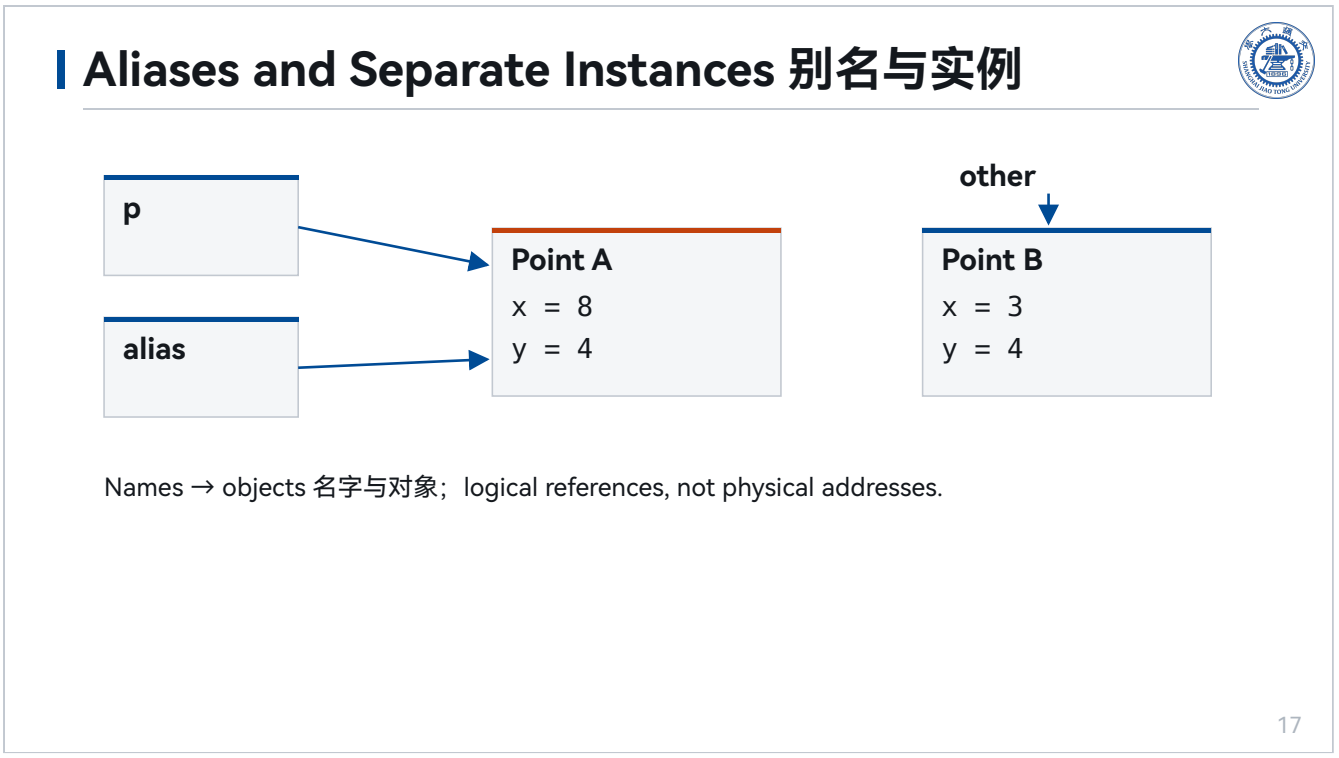
Actual output 实跑输出

```
class body
init
init
```

- **Class body 类体**: once in this program
- **Initialization 初始化**: once per construction
- **__init__ returns None**; the class call returns the instance

Slide 16 · Class Execution and Initialization Results

> 实际顺序是 class body、init、init。类定义被执行一次，创建两个实例分别初始化。若再次执行 class 语句，类体也会再次运行，不能概括为永远只运行一次。
>
> init 自身返回 None，Point(...) 的结果是实例。错误的非 None 返回与创建钩子在参考区解释，不在主线实现元类。



Slide 17 · Aliases and Separate Instances 别名与实例

> alias = p 绑定同一个对象，other = Point(3, 4) 则新建对象。通过 alias 将 x 改为 8，p 同样观察到 8，other 仍为 3。
>
> 坐标相同不等于一个对象。图的箭头是逻辑引用，不是物理地址；is 的真实结果分别为 True 和 False，沿用 L05 的模型。

Initialization Contract 初始化约定

- **Required attributes 必需属性**
- **Accepted values 输入范围**
- **Defaults 默认值**
- **Failure behavior 失败行为**

属性存在不等于数值合法。

18

Slide 18 · Initialization Contract 初始化约定

> 初始化要说明哪些属性必须建立、哪些初值有效、默认参数是什么，以及失败时发生什么。属性存在和数据合法是两个检查。
>
> Point 的例子用于理解初始化，没有自动验证任意传入对象的数值类型。后半场账户和温度会把状态规则写成显式检查。

II · Instance Methods

> 已经知道一个实例怎样获得坐标。现在多个点可以共用同一个 show 或 move 实现，但每次调用要明确操作哪一个对象。
>
> 首先学习普通实例方法，再比较局部参数、属性与结果。类方法和静态方法暂不混入这些例子。

Method Definitions 方法定义

code

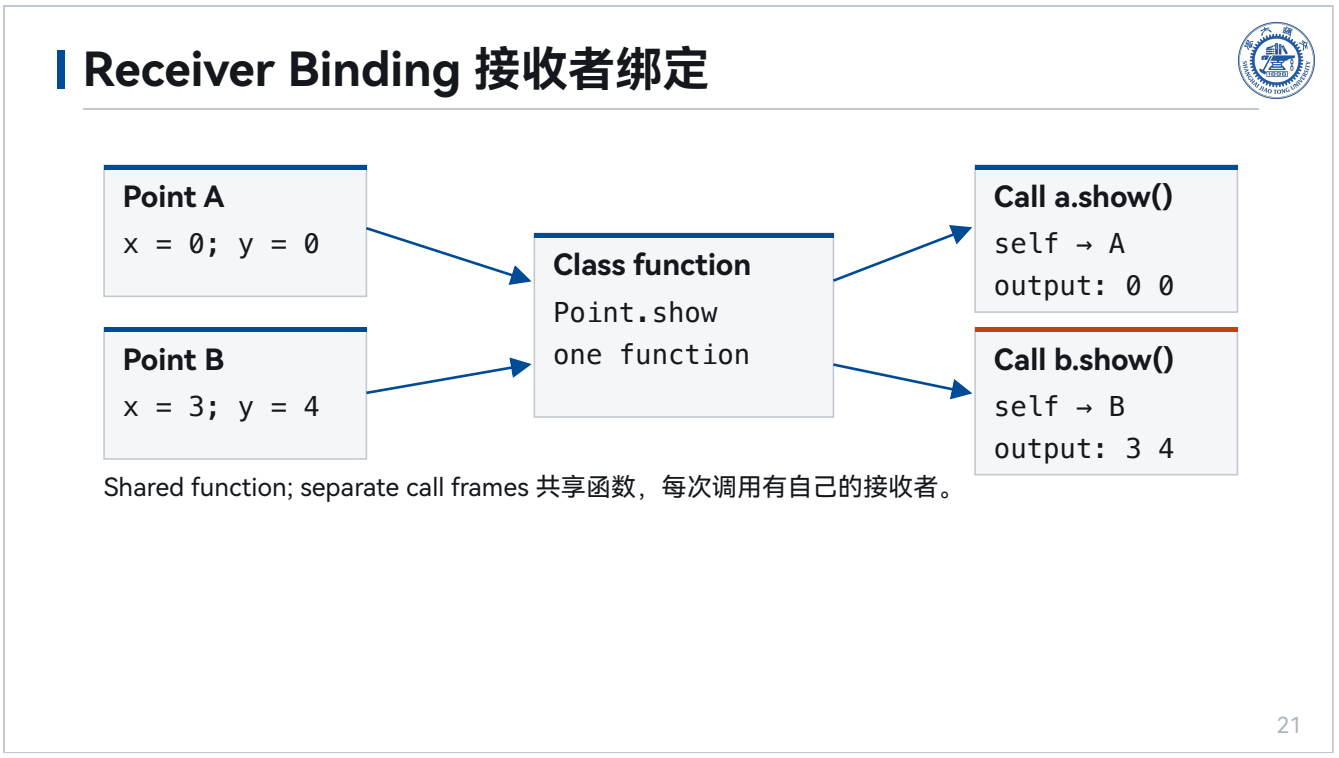
```python
class Point:
    def __init__(self, x: float = 0, y: float = 0) -> None:
        self.x = x
        self.y = y

    def show(self) -> None:
        print(self.x, self.y)

p = Point(3, 4)
p.show()
```

output

```
3 4
```


Slide 20 · Method Definitions 方法定义

> 普通实例方法的第一个参数接收绑定对象。定义写 self，p.show() 选择 p 作为接收者并打印 3 4。
>
> self 是惯用名字，不是关键字。参数规则仍是函数规则；不能把普通实例方法的绑定规则泛化到所有点号调用。



Slide 21 · Receiver Binding 接收者绑定

> 对这个普通 Python 方法，a.show() 可以显式写为 Point.show(a)。函数实现相同，但每次调用有自己的接收者与局部状态。
>
> 图中两次调用分别选择 A 和 B，不把两个实例画成各自复制一份方法代码。完整元数据证明放参考区，在这里不先用 __func__。

A Missing Receiver Parameter

What does p.show() supply? 调用传入什么?

```
class Point:
    def show():
        print("hi")

p = Point()
p.show()
```


Slide 22 · A Missing Receiver Parameter

> 函数定义不接收参数，实例调用仍会自动传一个接收者。请先预测发生的是正常打印还是参数错误。
>
> 本页不写答案，下一页结合局部参数解释。问题在参数数量，不在函数参数必须拼成 self。

Local Parameters and Attributes 局部与属性

code

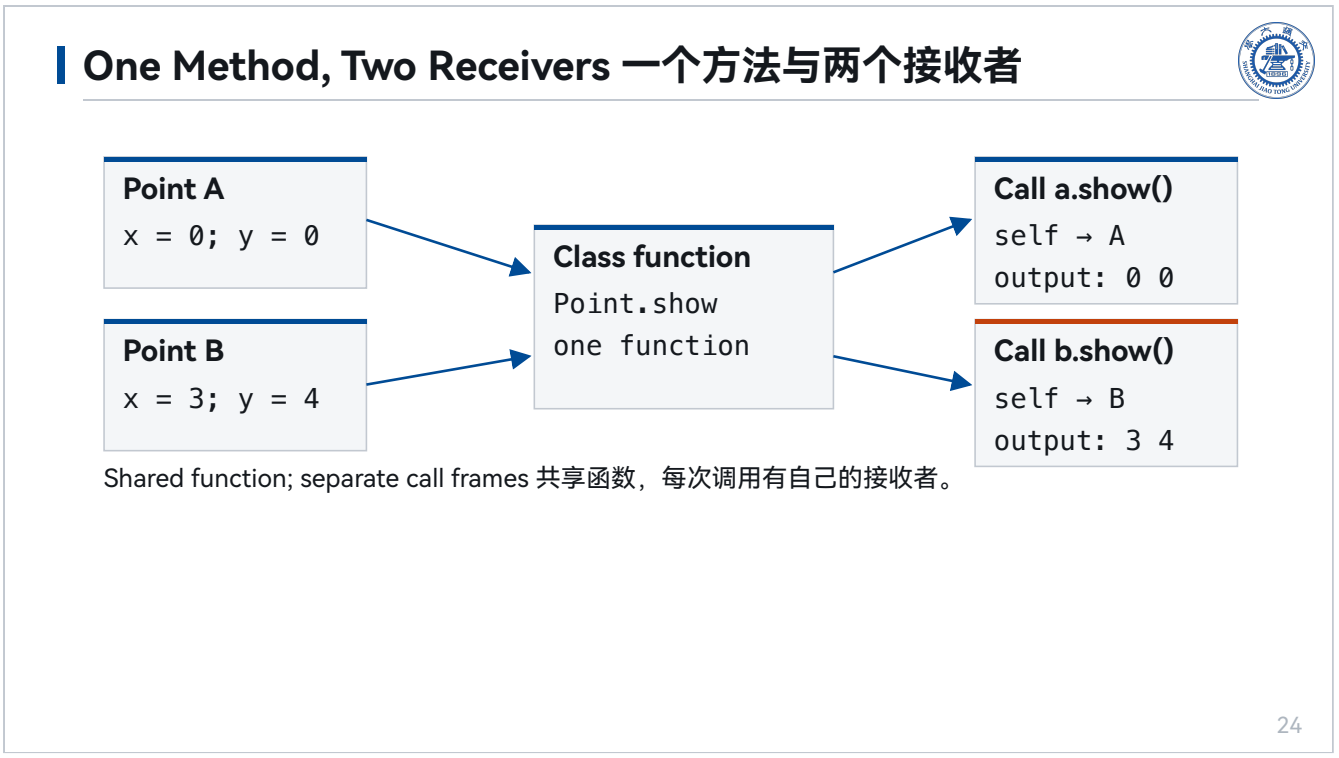
```python
class Point:
    def __init__(self, x: int) -> None:
        self.x = x
    def shift(self, x: int) -> None:
        self.x += x

x = 100
p = Point(3)
p.shift(2)
print(p.x, x)
```

output

```
5 100
```


Slide 23 · Local Parameters and Attributes 局部与属性

> 上一页真实 TypeError 表示零参数函数收到了一个实例。改用其他名字的第一参数在语法上可行，省略接收参数才造成不匹配。
>
> 这里 shift 的局部 x 是 2，self.x 原来是 3，外部 x 仍为 100，得到 5 100。局部名字和对象属性的绑定位置不同。



Slide 24 · One Method, Two Receivers 一个方法与两个接收者

> 先让学生分别说出 a.show 和 b.show 中 self 指向哪里。一个底层函数通过不同接收者读取不同坐标，真实输出为 0 0、3 4。
>
> 每次调用的局部状态独立，不表示实例可达的所有对象都不共享。第四节会区分独立绑定和共享列表。

Point Movement 点的平移

code

```python
class Point:
    def __init__(self, x: float = 0, y: float = 0) -> None:
        self.x = x
        self.y = y

    def move(self, dx: float, dy: float) -> None:
        self.x += dx
        self.y += dy

a = Point(0, 0)
b = Point(3, 4)
a.move(1, 1)
print(a.x, a.y)
print(b.x, b.y)
```

output

```
1 1
3 4
```


Slide 25 · Point Movement 点的平移

> move 将增量加入接收者的两个坐标。a 从 0 0 变为 1 1，b 仍为 3 4。源码包含完整最小定义，独立运行不会借用前页状态。
>
> 修改对象与创建新点是不同接口。这里明确原地操作，下一页再核对返回值，不根据方法名猜行为。

State Changes and Return Values 状态与返回值

code

```
def move(record, dx, dy):
    record["x"] += dx
    record["y"] += dy

p = {"x": 3, "y": 4}
result = move(p, 1, 0)
print(p["x"], result)
```

output

```
4 None
```


Slide 26 · State Changes and Return Values 状态与返回值

> 回收开场题：move 修改原字典的 x 为 4，没有显式 return，因此结果是 None，实际为 4 None。查看第 1 题原分布，不再次提交。
>
> Point.move 使用同样的状态/结果区别。None 不代表调用没有修改数据，数值结果也不证明方法只读；两项都要观察。

Calls Within a Method 方法内调用

code

```
1  def message() -> str:
2      return "global function"
3
4  class Record:
5      def __init__(self, text: str) -> None:
6          self.text = text
7      def message(self) -> str:
8          return self.text
9      def report(self) -> str:
10         return self.message()
11
12 r = Record("instance method")
13 print(message())
14 print(r.report())
```

output

```
global function
instance method
```


Slide 27 · Calls Within a Method 方法内调用

> 普通 message() 选择外部函数，self.message() 选择 Record 的方法，真实输出分别为 global function 和 instance method。
>
> 类不会自动把每个裸名字变成 self 的属性。这里先使用明确访问路径，完整 LEGB 规则仍在 L14，不先引入闭包。

Forward Type References 前向类型引用

- `def distance_to(self, other: "Point") -> float:`
- **The class name is bound after its body finishes.**
- **The string annotation defers the reference.**

> 先解释引号的作用，再使用引用当前类的类型提示。

Slide 28 · Forward Type References 前向类型引用

> 下一方法需要另一个 Point。在 Python 3.13 执行类体时，类名尚未完成绑定，直接在注解中使用 Point 可能提前求值而报 NameError。
>
> 字符串前向引用是本讲主线用法。future 的可选版本信息放参考，不把推迟注解理解为运行时类型校验。

Distance Between Objects 对象间距离

code

```python
class Point:
    def __init__(self, x: float = 0, y: float = 0) -> None:
        self.x = x
        self.y = y

    def distance_to(self, other: "Point") -> float:
        dx = self.x - other.x
        dy = self.y - other.y
        return (dx * dx + dy * dy) ** 0.5

a = Point(0, 0)
b = Point(3, 4)
print(a.distance_to(b))
```

output

```
5.0
```


Slide 29 · Distance Between Objects 对象间距离

> distance_to 从 self 与 other 分别读取坐标，计算差值和距离。本例得到 5.0，不移动两个点。
>
> 接口约定查询不修改状态，还应实际核对坐标。几何例子使用尺度适中的有限数值，不声称任意大数都具有数值稳定性。

Quiz 2 · Initialization and Receivers

扫码作答 · 2 题

1 局部参数与实例属性　2 显式选择方法接收者

Submit this round once. 按已学规则分析。

https://taohuang.info/cs1602/zh/quiz/9/

Slide 30 · Quiz 2 · Initialization and Receivers

> 两题分别检查是否建立实例字段，以及显式普通方法调用选择哪个实例。所有语法已经解释，不提前考类方法或共享类列表。
>
> 每题提交一次，下一页按原代码与实际结果解释。不要凭参数名字或最近创建的对象猜接收者。

Quiz Review · Attribute Creation

code

```
class Label:
    def __init__(self, text: str) -> None:
        text = text.strip()

a = Label(" hi ")
print(a.text)
```

Actual exception · final line 实际异常

```
AttributeError:
'Label' object has no attribute 'text'
```


Slide 31 · Quiz Review · Attribute Creation

> text.strip 的结果被绑定给局部 text，没有建立 self.text；访问 a.text 得到实际 AttributeError。注解和参数名字不创建字段。
>
> 这里核对的是属性是否存在，后面的第 4 题则检查属性初值是否合法，两项不同。查看原题分布，不新发一题。

Quiz Review · Receiver Selection

code

```python
class Counter:
    def __init__(self, start: int = 0) -> None:
        self.value = start
    def add(self, delta: int) -> None:
        self.value += delta

a = Counter(1)
b = Counter(10)
Counter.add(b, 2)
print(a.value, b.value)
```

output

```
1 12
```


Slide 32 · Quiz Review · Receiver Selection

> Counter.add(b, 2) 显式把 b 作为 self。b.value 从 10 更新到 12，a.value 保持 1，实际打印 1 12。
>
> 普通实例调用可写 b.add(2)，这里的类访问用于把隐含参数说清楚。没有根据变量名或创建顺序选择对象。

The Point Interface Point 接口

Operation 操作	State effects 状态	Result / output
Point(x, y)	Initialize new coordinates	Class call → instance; init → None
show()	Coordinates unchanged	Printed coordinates; None
move(dx, dy)	Change this instance	None
distance_to(other)	Both points unchanged	Distance

Slide 33 · The Point Interface Point 接口

> 把已经学过的 Point 接口集中核对。类调用返回实例，init 本身返回 None；显示、修改与查询是不同操作。
>
> 表格描述本实现的约定，不是 Python 自动保证。实际检查状态、输出和返回值，并在讲义中看完整独立程序。

Instance-Method Reasoning 实例方法分析

1	**Receiver 接收者**	Identify the object supplied as self.
2	**Bindings 绑定**	Separate local arguments from attributes.
3	**Calls 调用**	Follow ordinary call and return order.
4	**Observations 观察**	Check state, returned value and output separately.

同一个方法的两次调用，先确定各自的接收者和原状态。

34

Slide 34 · Instance-Method Reasoning 实例方法分析

> 上半场形成四步分析：找接收者、分清绑定位置、按调用顺序执行、分别观察状态和结果。递归与普通函数的调用规则仍适用。
>
> 后半场再问：即使调用过程理解正确，对象状态是否符合约定？这需要显式规则及验证，并不是使用类就已经完成。

HALFWAY

Break

> 在这里进入两个 45 分钟的分界。上半场已经解释对象、初始化、self 和状态/返回值。
>
> 后半场从账户的非法操作进入有效状态与封装，再比较类级共享、实例字段和类方法。参考区不计入必讲时间。

III · Encapsulation and State Rules

> 方法把相关操作组织起来，但错误操作仍可能破坏状态。先写账户任务的单位、输入和规则，再审查候选。
>
> 以非负余额为例，初始状态与每个公开更新都要检查。类不是权限边界，直接写属性仍可能绕过约定接口。

The Account Contract 账户约定

Item 项目	Accepted behavior 约定
State 状态	Nonnegative balance; integer cents
deposit(amount)	Positive integer; returns None
withdraw(amount)	Positive integer; enough funds; returns bool
Rejected calls 拒绝	Balance and history stay unchanged
statement()	Return a string; do not change state

先确定单位、输入与失败行为，再审查更新语句。

Slide 37 · The Account Contract 账户约定

先规定金额单位为整数分，初始余额非负，存取金额为正整数。存款返回 None，取款通过 bool 报告成功，拒绝不改变余额和历史。

这是教学模型，不具有审计、货币或并发的生产保证。属性仍可直接赋值，下面的规则仅保证正确使用约定方法的行为。

A Withdrawal Candidate 取款候选

Predict withdraw(-20). 候选，不是正确示范。

```python
class CandidateAccount:
    def __init__(self, balance: int) -> None:
        self.balance = balance
    def withdraw(self, amount: int) -> bool:
        if amount > self.balance:
            return False
        self.balance -= amount
        return True

a = CandidateAccount(100)
print(a.withdraw(-20), a.balance)
```


Slide 38 · A Withdrawal Candidate 取款候选

> 这页明确是有错误的候选。余额为 100，取款 amount 为 -20，先预测原分支会不会拒绝，以及减去负数的结果。
>
> 不能因为放在类里就按常识认为它会报错。下一页展示真实结果并修正条件，不在预测页写答案。

Withdrawal Validation 取款检查

code

```python
class BankAccount:
    def __init__(self, balance: int) -> None:
        if balance < 0:
            raise ValueError("negative opening balance")
        self.balance = balance
    def withdraw(self, amount: int) -> bool:
        if amount <= 0 or amount > self.balance:
            return False
        self.balance -= amount
        return True

a = BankAccount(100)
print(a.withdraw(-20), a.balance)
```

output

```
False 100
```


Slide 39 · Withdrawal Validation 取款检查

> 旧候选的真实结果为 True、120，原因是只检查余额不足。新版本在更新前拒绝非正金额或余额不足，负取款实际为 False、100。
>
> 完整教学账户还检查整数类型，并同时观察历史是否变化。本页是最小独立整数例子，完整程序与非法初值检查在讲义。

Rules at Every Entry Point 入口与规则

Entry 入口	Validate 检查	On rejection 拒绝后
Constructor 构造	Nonnegative opening balance	ValueError; no valid instance returned
Deposit 存款	Positive integer amount	Balance/history unchanged
Withdraw 取款	Valid amount and enough funds	False; balance/history unchanged

Slide 40 · Rules at Every Entry Point 入口与规则

> 验证规则覆盖构造、存款与取款各入口。一个 setter 有检查，不会让构造自动执行相同规则。
>
> 拒绝还需要检查没有先修改余额、没有先加入历史。按接受和拒绝路径分别记录原状态、操作、结果与新状态。

Account State Transitions 账户状态变化

Actual output 实跑输出

```
True 120
False 120
False 120
0 []
[('deposit', 50), ('withdraw', 30)]
```

- **100 → +50 → −30 → 120**
- **Rejected 1000 and −20** leave state unchanged
- **Separate account 独立账户**: zero balance, empty history
- **History** records accepted actions only

Slide 41 · Account State Transitions 账户状态变化

> 真实程序存入 50 后取出 30，余额为 120；过额和负取款都返回 False，余额和历史保留。历史只含两次接受的操作。
>
> 第二账户保持余额 0 和空历史。其列表来自独立初始化，不共享类级列表。账单实现用普通循环，不提前使用生成器表达式。

The Temperature Contract 温度约定

Check 检查	Rule 规则
Domain 输入域	Finite real values; celsius ≥ −273.15
Initialization 初值	Invalid starting value → ValueError
set_celsius(value)	False and unchanged on rejection; True on acceptance
math.isfinite(value)	True for finite values, False for NaN/infinity

> 非法初值也要拒绝，不能只检查后来的 setter。

Slide 42 · The Temperature Contract 温度约定

> 先解释 math.isfinite 是有限性检查，再在温度代码中使用。值不能低于绝对零度，构造和更新都需检查。
>
> 内部 _celsius 在此首次说明为非公开命名约定，不强制保护。课件只给最小摄氏/华氏接口，实验拓展三种查询，方法名以实验页为准。

Temperature Updates 温度更新

Excerpt · complete definition in notes

```python
def set_celsius(
    self,
    value: float,
) -> bool:
    if (
        not math.isfinite(value)
        or value < -273.15
    ):
        return False
    self._celsius = value
    return True
```

Complete runnable example 完整程序

- **Starting state**: 25 Celsius
- **First call**: set_celsius(-300)
- **Second call**: set_celsius(100)
- **Predict each result and retained value.**

Slide 43 · Temperature Updates 温度更新

> 先按已写明的条件预测两次更新：一个低于边界，一个有效。分别观察布尔返回值与保留的数值。
>
> 本页明确展示 setter 片段，构造和查询定义在完整讲义。真实答案放下一页，不在预测内容中显示，也不给不完整片段加运行按钮。

Public and Nonpublic Interfaces 公开与非公开接口

Previous prediction · actual result

```
False 25
True 212.0
```

- **Public interface 公开接口**: supported operations and rules
- **_name 非公开约定**: implementation detail
- **Direct assignment remains possible**; no enforced privacy

下划线不提供权限控制。

Slide 44 · Public and Nonpublic Interfaces 公开与非公开接口

> 先回顾上一页实际为 False 25、True 212.0，拒绝保留旧值，接受后正确更新并转换。构造负温度也应拒绝。
>
> 再区分公开接口与内部字段。单下划线说明不稳定实现，语言仍允许访问或写入；封装不是安全沙箱。

Name Mangling 名称改写

code

```
class Secret:
    def __init__(self) -> None:
        self.__hidden = 42

s = Secret()
print(s._Secret__hidden)
print(s.__hidden)
```

Actual exception · final line 实际异常

```
AttributeError:
'Secret' object has no attribute '__hidden'
```


Slide 45 · Name Mangling 名称改写

> 双前导下划线在类体发生名称改写，_Secret__hidden 可以读取 42，原拼写触发 AttributeError。这些结果来自实际运行。
>
> 这里不使用继承代码。避免子类无意冲突的理由在 L10 继续解释，不能称它为绝对私有访问保护。

Underscore Naming 下划线命名

Form 形式	Meaning 含义	Access control?
_x	Nonpublic convention	No
__x	Name mangling in a class body	No
__init__	A documented special-method name	Not the __x mangling rule

Slide 46 · Underscore Naming 下划线命名

> 三种形式不能混为一谈。_x 是非公开约定，__x 在类体中改写名字，__init__ 是文档规定的特殊方法。
>
> 任意自造的双端下划线名字不自动具有解释器行为。读写是否被允许与接口是否建议使用也需要分开。

Quiz 3 · Initial-State Rules

扫码作答 · 1 题

1 构造入口与有效状态

Submit this round once. 按已学规则分析。

https://taohuang.info/cs1602/zh/quiz/9/

47

Slide 47 · Quiz 3 · Initial-State Rules

> 本题 setter 有负数检查，但构造直接存入参数。按实际执行判断，不能认为构造自动调用 setter。
>
> 这次检查数值是否有效，与第 2 题是否创建属性不同。单下划线已经解释，直接读取内部字段仅作为代码审查手段。

IV · Shared Data and Method Kinds

> 实例有自己的绑定位置，但两个位置仍可能指向同一个列表。先分析普通类属性查找与赋值，再区分修改对象和建立新绑定。
>
> 最后补齐往年的类方法与 cls，不把缺失内容删掉，也不提前展开通用装饰器。

Instance and Class Attributes 实例与类属性

code

```python
class Point:
    dimensions = 2
    def __init__(self, x: float, y: float) -> None:
        self.x = x
        self.y = y

a = Point(0, 0)
b = Point(3, 4)
print(a.x, b.x)
print(a.dimensions, b.dimensions, Point.dimensions)
```

output

```
0 3
2 2 2
```


Slide 49 · Instance and Class Attributes 实例与类属性

> 两个 Point 分别保存坐标，dimensions 是类体建立的属性。本例各维度访问都为 2，但 a.x 与 b.x 不同。
>
> 不采用国籍或未经核查动物分类的类比。这里的几何例子与主线一致，Dog 原字段更新例子仍在参考区保留。

Ordinary Attribute Lookup 普通属性查找

Predict a.label, b.label and Record.label. 先确定绑定位置。

```
class Record:
    label = "class"

a = Record()
b = Record()
a.label = "only A"
print(a.label, b.label, Record.label)
```


Slide 50 · Ordinary Attribute Lookup 普通属性查找

> 在本讲没有描述符与继承参与的普通字段例子中，实例缺少同名绑定时可以访问类属性。先预测对 a 的赋值会影响哪些访问。
>
> 下一页显示实际结果并区分两种赋值位置，不把简化表当作 Python 的完整查找算法。

Attribute Binding and Mutation 绑定与修改

Previous prediction · actual result

```
only A class class
```

Operation 操作	Effect 影响
a.label = ...	Establish an ordinary instance binding
Record.label = ...	Change the class binding
a.items.append(...)	Mutate the referenced list

区分重新绑定与修改对象。

Slide 51 · Attribute Binding and Mutation 绑定与修改

> 上一页真实为 only A、class、class，给 a 的绑定没有改类或 b 的访问。普通实例赋值和类赋值的目标不同。
>
> append 则修改已经找到的列表，不自动新建实例字段。下面先按这个区别预测共享篮子的结果。

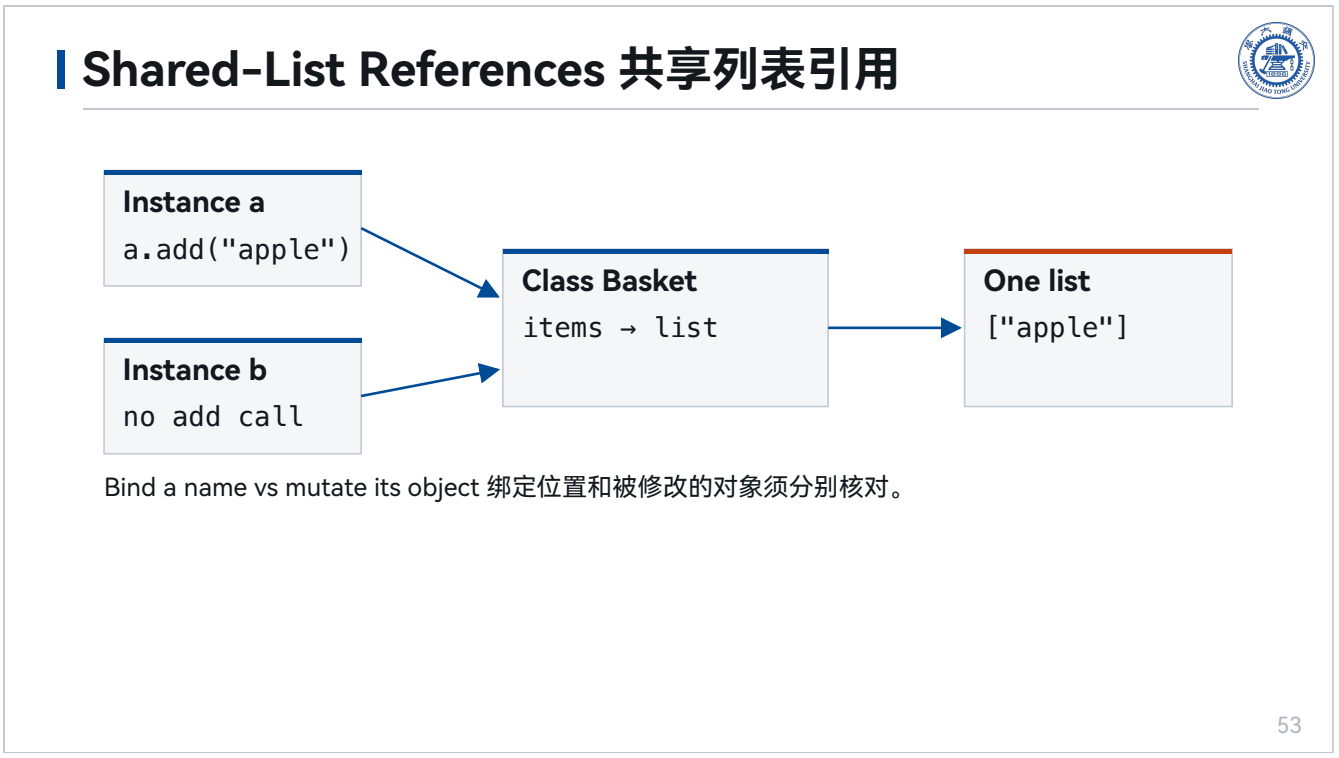
A Shared Basket List 共享列表

Which list does add() modify? 修改的是哪份列表?

```python
class Basket:
    items: list[str] = []
    def add(self, item: str) -> None:
        self.items.append(item)

a = Basket()
b = Basket()
a.add("apple")
print(b.items)
print(a.items is b.items)
```


Slide 52 · A Shared Basket List 共享列表

> 类体只有一次列表构造，add 中 self 选择当前篮子，但 self.items 可能通过类属性找到同一个列表。先预测 b.items 和身份比较。
>
> 不要提前显示结果。下一页用真实关系图说明，不把新实例画成隐式复制了集合。



Slide 53 · Shared-List References 共享列表引用

> 实际 b.items 含 apple，a.items is b.items 为 True。两个实例经同一个类属性找到一份列表，append 修改该对象。
>
> 共享本身并非语法错误；本任务要求独立篮子，才需要修正。若接口有意登记共有状态，类级集合也可能合理。

Per-Instance Collections 实例集合

code

```python
class Basket:
    def __init__(self) -> None:
        self.items: list[str] = []
    def add(self, item: str) -> None:
        self.items.append(item)

a = Basket()
b = Basket()
a.add("apple")
print(a.items, b.items)
print(a.items is b.items)
```

output

```
['apple'] []
False
```


Slide 54 · Per-Instance Collections 实例集合

> 把 [] 放进 init，每次初始化执行新列表构造。添加 a 后 b 仍为空，身份比较为 False。
>
> 独立性来自实际构造新对象，不来自 self 名字。空列表值可以相等而身份不同，继续沿用 L05 的判断方式。

Shared Input References 输入引用

Does initialization copy an input list? 初始化会自动复制吗?

```python
class Bucket:
    def __init__(self, items: list[str]) -> None:
        self.items = items

shared = []
a = Bucket(shared)
b = Bucket(shared)
a.items.append("apple")
print(b.items)
print(a.items is b.items)
```


Slide 55 · Shared Input References 输入引用

> self.items = items 直接绑定传来的对象。两个构造都传 shared，先预测 b 的内容及两字段是否同一个对象。
>
> 这是不同于类级列表的共享来源。下一页先回顾真实结果，再介绍类级计数，不把所有共享都归因于类变量。

An Instance-Creation Counter 创建计数

code

```python
class Student:
    count = 0
    def __init__(self, name: str) -> None:
        self.name = name
        Student.count += 1


Student("A")
Student("B")
print(Student.count)
```

output

```
2
```


Slide 56 · An Instance-Creation Counter 创建计数

> 先回顾上一页：初始化没有复制 shared，b 也看到 apple，身份为 True。需要隔离外层时可以按接口复制，并注意浅拷贝的范围。
>
> 这里通过 Student.count 更新类绑定，两个初始化得到 2。计数是创建事件，不是存活对象数；失败初始化应先校验再决定是否计数。

Class Methods and cls 类方法

- **@classmethod** marks a method receiving the class
- **cls** is the conventional first parameter
- **No instance required** for access through the class

> 确定自动参数传的是实例还是类。

Slide 57 · Class Methods and cls 类方法

> 先明确内置 @classmethod 标记的具体行为：自动参数是类，惯用名 cls，从类访问无需先创建实例。
>
> 下页才使用这段语法。本讲不展开高阶函数或一般装饰器原理，那部分仍在 L14；普通实例方法的 self 规则不能套到这里。

Calling a Class Method 类方法调用

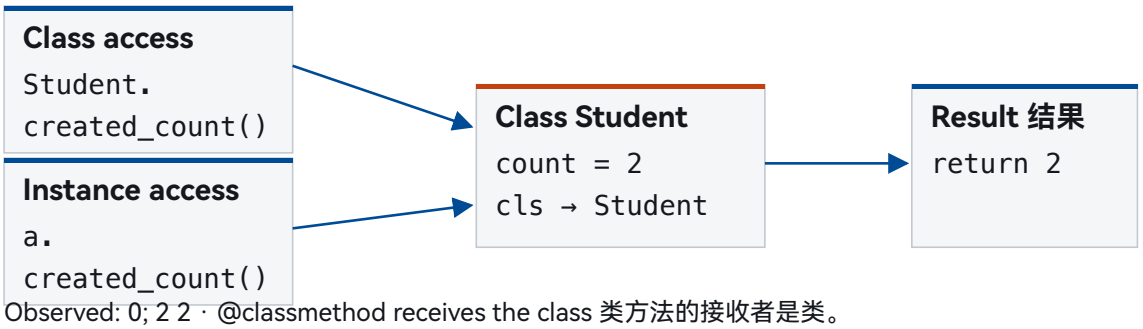
Which receiver and results? 接收者与结果分别是什么?

```python
class Student:
    count = 0
    def __init__(self, name: str) -> None:
        self.name = name
        Student.count += 1

    @classmethod
    def created_count(cls) -> int:
        return cls.count

print(Student.created_count())
a = Student("A")
b = Student("B")
print(Student.created_count(), a.created_count())
```


Slide 58 · Calling a Class Method 类方法调用

> 先预测未创建和创建后的计数，再说明从 a 访问类方法时 cls 是谁。源码有完整独立定义，计数从这次类体的 0 开始。
>
> 答案在下一页静态表中。不是从实例访问就一定传实例，class method 仍绑定类。

Method Categories 方法类别



Observed: 0; 2 2 · @classmethod receives the class 类方法的接收者是类。

Kind 类别	Implicit receiver 自动参数	Typical purpose
Instance method	Instance (self)	Per-instance operations
@classmethod	Class (cls)	Class-level operations
@staticmethod	None	Related operation; no receiver state

59

Slide 59 · Method Categories 方法类别

> 前页实际为 0、2 2，两种 created_count 访问都将 Student 传给 cls。先显示真实结果再比较方法类别。
>
> 这里首次说明静态方法不自动传接收者，完整程序在参考区。小测只考已经展示的类方法和共享，不考未展示的静态实现。

Quiz 4 · Sharing and Class Methods

扫码作答 · 2 题

1 列表修改与实例重新绑定　2 类方法的自动接收者

Submit this round once. 按已学规则分析。

https://taohuang.info/cs1602/zh/quiz/9/

Slide 60 · Quiz 4 · Sharing and Class Methods

> 第 5 题结合类列表与实例遮蔽，第 6 题只检查 cls 绑定。分别按已经教过的规则执行，不能只按关键词做选择。
>
> 两题各提交一次，下面保留静态结果解释与原题分布。不重复开场题，也不加入静态方法的新实现。

Quiz Review · Sharing and Binding

Q5 · actual list lengths

```
0 1 1
```

Q6 · actual class count

```
2
```

- **Q5**: mutate the class list, then bind a new instance list
- **Q6**: both accesses supply Sensor as cls

> 创建实例不自动调用任意方法；给实例新绑定也不替换类上的列表。

Slide 61 · Quiz Review · Sharing and Binding

> 第 5 题先给原类列表加入 apple，再给 a 新列表，长度为 0、1、1；b 与 Basket 仍访问原列表。
>
> 第 6 题两次类方法访问都给 Sensor.count 增加 1，结果为 2，构造 Sensor 并没有自动执行 bump。查看原分布，不新增提交。

Design Choices 类的设计选择

Consider 考虑因素	Reasoning 判断依据
State lifetime 状态持续	Must data be retained across calls?
Related operations 相关操作	Does one interface make them clearer?
Rules and failures 规则与失败	Which updates and invariants need a contract?
Alternatives 其他组织	A function, dictionary or record may suffice

按状态与接口选择组织方式。

Slide 62 · Design Choices 类的设计选择

> 选择类要根据状态持续、操作关系、有效规则和接口边界判断。普通函数、字典和记录也可能更直接。
>
> 一个实例、一个方法或只读数据都不能机械排除类。旧材料中的规模与协作动机保留，未经来源核对的行数与城市类比不使用。

Lab 9 · Design and Verification

Part 实验段	Tasks 任务	Raw points
A	Rectangle / Stack / WordCounter / Temperature	16 auto + 8 staff = 24
B	Gradebook refactoring and regression	8 auto + 4 staff = 12
C	Class/function choices; receiver debugging	0-point learning record

HW3 due: **17 Nov, 23:59 GMT+8**. 接口和入口以实验页为准。

Slide 63 · Lab 9 · Design and Verification

保留原六项任务，分为四个基础类、成绩簿重构、设计判断与调试记录。五个代码入口与三张文字卡已经在本地准备，不能把本地完成称为线上提交。

矩形原地 scale 返回 None，空 pop/peek 抛 IndexError，空成绩簿 top_student 返回 None。代码与文字记录分别评分，C 不增分。截止从统一校历读取；需要真实 AI 使用或未使用声明。

Inheritance, Protocols and Modules

- **A class organizes one kind of state and behavior.**
- **How can related classes reuse implementation?**
- **How can classes and functions be organized into files?**

参考区供课后查阅。

Slide 64 · Inheritance, Protocols and Modules

> 回到本讲的问题：已经能为一类状态说明初始化、方法接收者、返回与副作用、公开规则以及共享层次。还需要通过测试验证接口是否实现。
>
> 下一讲研究继承、运算协议、多态和模块。这些旧合并课件内容不是被删除，本讲没有提前使用它们。课堂主线到这里结束。

Reference 参考

> 参考区保留完整实现、更多验证、往年的三角形、Dog、快乐数和向量等内容，不计入本讲必讲时间。
>
> 片段页面明确标记并链接完整讲义。输出、异常和数值由实际运行获得，后续 L10 内容仍按范围承接。

Built-in Constructors 内置构造器

code

```
classes = [int, float, complex, str,
           list, tuple, dict, set]
for cls in classes:
    value = cls()
    print(cls.__name__, repr(value))
```

output

```
int 0
float 0.0
complex 0j
str ''
list []
tuple ()
dict {}
set set()
```


Slide 66 · Built-in Constructors 内置构造器

> int、float、complex、str 与几个容器的空构造有各自默认值。repr 让空字符串可见，__name__ 给出这里的类名。
>
> 输出来自当前 3.13，不固定任何 id 或地址。名字重新绑定到另一类对象，不表示原来的对象类型改变。

Class Execution and Object Identity

Actual output 实跑输出

```
True
False True
```

- **Class statements execute when reached.**
- **Instance creation differs from alias assignment.**
- **Identity 身份** is separate from coordinate values.

Slide 67 · Class Execution and Object Identity

> 补充旧材料中类只是等到实例化才使用的误述。执行类体时已经建立类、字段和函数，实例创建又是后续调用。
>
> 同坐标两个新 Point 不是同一个对象，别名则引用同一对象。用 True/False 身份关系核对，不复制旧内存地址。

Point · Complete Program

Excerpt · complete definition in notes

```python
def distance_to(
    self,
    other: "Point",
) -> float:
    dx = self.x - other.x
    dy = self.y - other.y
    return (dx * dx + dy * dy) ** 0.5
```

Actual complete-program output 实跑输出

```
5.0
1 1
3.605551275463989
```

- **Constructor defaults** and required x/y
- **show / move / distance_to**
- **Check state and result separately**

Complete runnable example 完整程序

Slide 68 · Point · Complete Program

> 完整 Point 同时提供默认构造、显示、原地平移和距离查询。片子展示距离方法片段，讲义包含完整定义和三次观察。
>
> 实际为初始距离 5.0，移动后 1 1，再查询约 3.60555。片段不加运行按钮，不要求先运行其他页面。

Three Sources of Shared Collections

Source 来源	What is shared? 共享内容	Check 核对
Class list	One class-level object	Class access and instance shadows
Mutable default	One default object created at definition	Calls omitting that argument
Repeated input	The same caller-owned object	Aliased constructor arguments

构造表达式与传入引用须分别核对。

Slide 69 · Three Sources of Shared Collections

> 三种情况都可能出现共享，但机制不同：类级列表、一次定义的可变默认对象、重复传入的同一对象。不能全部归因于类变量。
>
> 需要隔离时明确是否新建或复制，浅拷贝只隔离外层。各完整例子在讲义独立运行，避免前一段留下的状态干扰。

Triangle · Interface and Validation

Interface 接口	Contract 约定
Defaults	Edges (1, 1, 1), not (0, 0, 0)
is_valid()	Finite positive edges; strict inequalities
set_edges(a, b, c)	False on invalid update; retain old edges
area / print / print_area	Check current data; use self.area()

Slide 70 · Triangle · Interface and Validation

> 完整保留往年三角形的字段、有效性检查、更新、面积与两种打印方法。零边虽然可存进字段，不表示合法三角形，因此正式默认值改为 1。
>
> 更新拒绝要保留原三边，area 也检查当前状态，因为直接写公开字段可以绕过 setter。示例只承诺尺度适中的有限数值。

Triangle · Heron's Formula

Excerpt · complete definition in notes

```
def area(self) -> float:
    if not self.is_valid():
        raise ValueError("invalid current edges")
    s = (self.a + self.b + self.c) / 2
    product = s * (s - self.a)
    product *= s - self.b
    product *= s - self.c
    return product ** 0.5
```

Actual complete-program output 实跑输出

```
Area: 6.000
```

- **s = (a+b+c)/2**
- **Reject invalid current edges**
- **3,4,5 → area 6**

Complete runnable example 完整程序

Slide 71 · Triangle · Heron's Formula

> Heron 公式用半周长 s 计算面积。area 先检查当前字段，不能对无效三角形返回一个虚构面积。
>
> 3、4、5 的面积实际为 6；print_area 通过 self.area 使用当前实例的方法。片段和完整讲义入口分开，长类不硬塞进一页。

Triangle · Independent Instances

Actual output 实跑输出

```
6.0
3.897114317029974
False 6.0
```

- **Separate edge data 独立边数据**
- **Rejected updates** keep all old values
- **Same method implementation** selects its receiver

Slide 72 · Triangle · Independent Instances

> 两个对象使用同一个 area 实现，但分别读取各自三边。实际面积为 6.0 和 3.897114317029974，无效更新后 a 的面积仍为 6。
>
> 保留旧例的多个实例、字段访问和方法内调用知识，不沿用源程序旧行号或地址。

Dog · Records and Updates

Excerpt · complete definition in notes

```
def print_dog(self) -> None:
    print(
        self.name, self.color,
        self.age, self.weight,
    )
```

Actual complete-program output 实跑输出

```
Fox Red 3 4
Tiger Red 4 2.0
```

- **name / color / age / weight**
- **External functions** can also operate on objects
- **Setter names do not imply validation**

Complete runnable example 完整程序

Slide 73 · Dog · Records and Updates

> 保留往年 Dog 的四字段、外部 train 函数、直接更新与 setter 接口。没有必要使用出处不明的动物或表情图片。
>
> 本例只说明记录组织，不宣称年龄体重已经符合业务规则。setter 这个名字不自动提供检查，需看实现和约定。

Happy Numbers · Steps and Cycles

Actual output 实跑输出

```
19 82 True
2 4 False
1234567890987654321 570 False
```

- **Positive integer input 正整数**
- **Digit-square sum** defines the next value
- **Stop at 1 or a repeated value**
- **Ordered trace**, not set display order

Slide 74 · Happy Numbers · Steps and Cycles

> 首步实际为 82、4、570，三个判定依次 True、False、False。19 经过 82、68、100、1，2 则进入包含 4 的循环。
>
> 集合用于检测出现过的值，不把集合显示顺序当执行轨迹。这个判定也可用普通函数，不是算法必须采用类。

Happy Numbers · Query Side Effects

Version 版本	Actual result 实际结果	Retained input?
Legacy candidate	`True 100`	No: number is changed
Local working state	`True 19`	Yes: number stays 19

- **current** holds this query's working value
- **seen** holds processed values
- **A query name does not guarantee no mutation**

Slide 75 · Happy Numbers · Query Side Effects

> 旧候选对 19 返回 True 时，number 已变成 100；修正版用局部 current 保留字段 19。这两行由两份实际完整程序运行得到。
>
> 只比较 True 会漏掉副作用。保留原数和相关操作是否值得类接口，需要根据任务判断，不声称类是快乐数算法必需。

Vector · Storage and Length

Excerpt · complete definition in notes

```
def length(self) -> float:
    total = 0.0
    for value in self.values:
        total += value * value
    return math.sqrt(total)
```

Actual complete-program output 实跑输出

```
Length: 5.000
```

- **Scalar list**, copied at construction
- **length / print_length**
- **Zero vector** has length 0

Complete runnable example 完整程序

Slide 76 · Vector · Storage and Length

> n 维数据存放在标量列表中，构造复制外层。length 用普通循环累计平方和，print_length 只是显示查询结果。
>
> [3,4] 长度实际为 5，空向量和零向量长度为 0。浅拷贝适合本例标量元素，不保证任意嵌套可变对象都独立。

Vector · Dot Product and Scaling

Excerpt · complete definition in notes

```
def dot(self, other: "Vector") -> float:
    if len(self.values) != len(other.values):
        raise ValueError("different dimensions")
    total = 0.0
    for i in range(len(self.values)):
        total += self.values[i] * other.values[i]
    return total
```

Actual complete-program output 实跑输出

```
11.0
None
10.0 [3.0, 4.0]
```

- **Same dimension** required for dot
- **scale mutates** and returns None
- **Caller's outer list** is retained

Complete runnable example 完整程序

Slide 77 · Vector · Dot Product and Scaling

> 点积要求两个向量维度一致，逐个坐标乘积累加。[3,4] 与 [1,2] 的实际点积为 11。
>
> scale 原地修改分量并返回 None，乘 2 后长度为 10，传入原列表保持。完整定义包含全部方法，片段不单独提供运行按钮。

Vector · Direction Angles 方向角

Excerpt · complete definition in notes

```python
def angle(self) -> list[float]:
    length = self.length()
    if length == 0:
        raise ValueError("zero vector")
    angles = []
    for value in self.values:
        direction = value / length
        cosine = max(-1.0, min(1.0, direction))
        angles.append(math.acos(cosine))
    return angles
```

Actual complete-program output 实跑输出

```
Angle: 0.9273 rad
Angle: 0.6435 rad
```

- **Angles to coordinate axes**, in radians
- **acos domain**: −1 to 1
- **Zero vector** has no direction

Complete runnable example 完整程序

Slide 78 · Vector · Direction Angles 方向角

> 明确方向角是向量与各坐标轴的夹角列表，单位弧度。先解释反余弦的定义域，再用分量除以长度得到方向余弦。
>
> [3,4] 两个实际角约为 0.9273 和 0.6435；零向量抛 ValueError。钳位用于微小舍入越界，不代表极端数值下的一般稳定性保证。

Namespaces and Function Objects 名称与函数对象

Excerpt · complete definition in notes

```
def area(self) -> float:
    return area(self.a, self.b, self.c)

f = print
f("function object")
```

Actual complete-program output 实跑输出

```
6.0 6.0
function object
```

- **External area** and **instance area** have explicit paths
- **Functions are objects** that names can reference
- **Ordinary functions already isolate locals**

Complete runnable example 完整程序

Slide 79 · Namespaces and Function Objects 名称与函数对象

> 外部 area 与实例 area 通过明确访问路径区分，实例方法取自身字段调用外部函数，两个面积都实际为 6.0。
>
> f = print 绑定函数对象，后续 f 调用正常打印。类不是局部参数隔离的唯一机制，不使用未经核查的超大软件数字强调必要性。

Bound-Method Metadata 绑定方法元数据

Excerpt · complete definition in notes

```
print(a.show.__self__ is a)
print(a.show.__func__ is b.show.__func__)
f = a.show
a.x = 8
f()
```

Actual complete-program output 实跑输出

```
True
True
8 0
```

- **__self__**: bound receiver
- **__func__**: underlying Python function
- **Saved method** observes later instance state

Complete runnable example 完整程序

Slide 80 · Bound-Method Metadata 绑定方法元数据

> 先解释 __self__ 与 __func__ 的定义再访问。两次身份比较实际 True、True，保存 a.show 后改变 a.x，f 仍选择同一个实例并打印 8 0。
>
> 这组属性针对本例 Python 定义的普通方法，不推广到所有内置方法。完整程序包含 Point 定义，不需要先运行之前页面。

Static and Unbound Calls 静态与显式调用

code

```python
class Test:
    @staticmethod
    def add(u: int, v: int) -> int:
        return u + v

print(Test.add(1, 1))
print(Test().add(3, 4))
```

output

```
2
7
```


Slide 81 · Static and Unbound Calls 静态与显式调用

> 在第 59 页已说明静态方法不自动传接收者。这里展示完整标记和程序，两种访问实际为 2、7。
>
> 没有标记的 add(u,v) 可以显式由 Test.add(1,1) 调用，但实例访问额外传对象导致参数过多。工具函数也可放类外，不把位置当作必须使用类的理由。

Annotations and Initialization Failures

code

```
class Wrong:
    def __init__(self):
        return 1

Wrong()
```

Actual exception · final line 实际异常

```
TypeError:
__init__() should return None, not 'int'
```


Slide 82 · Annotations and Initialization Failures

> init 返回 1 在 3.13 产生实际 TypeError，init 必须返回 None。误拼初始化方法不会自动运行它，缺接收者则是参数数量问题。
>
> 通常 __new__ 创建、__init__ 初始化；不在这里实现创建钩子。字符串前向注解是主线，future 从 3.7 可选且 3.13 仍需显式启用，不把它当类型校验。

Object-Oriented Organization and Alternatives

- **Simula (1960s)** and **Smalltalk (1970s)**: early influential mechanisms
- **Python supports objects and procedural organization.**
- **C++, Java and Python** are familiar related languages
- **Functions, dictionaries and records** remain valid alternatives

按状态、接口与生命周期选择，不用范式名称代替设计依据。

Slide 83 · Object-Oriented Organization and Alternatives

保留往年 OO 的组织动机和简短历史。公开来源支持 Simula 与 Smalltalk 的影响，不足以声称 OO 在所有领域取代其他范式。

函数、字典、元组与结构化记录仍可满足任务。NamedTuple 只介绍名称和适用性，不提前使用新 API；单实例、单方法或只读数据不是硬判断规则。

Independent Verification 独立核验

Dimension 检查维度	Evidence 证据
Construction 构造	Defaults, attributes, valid and invalid inputs
Sharing 共享	Two instances, aliases and shared arguments
Effects 副作用	Results, output and retained state
Failure 拒绝	All old fields unchanged, not a partial update

Slide 84 · Independent Verification 独立核验

> 围绕一个独立程序核对默认值、初始范围、多个对象和返回值，再分别验证接受与拒绝路径。只看正常样例无法证明独立性或失败原状态。
>
> 用本讲普通例子说明验证方法，不公开五道实验完整参考实现。可信教师测试也不能声称已经完成生产安全沙箱验证。

Lab 9 · Interfaces and Edge Cases

Class 类	Fixed contract 固定约定
Rectangle	Finite positive dimensions; in-place scale → None
Stack	Empty pop and peek → IndexError
WordCounter	Exact keys; count descending, word ascending
Temperature	Invalid start → ValueError; rejected update → False
Gradebook	Empty top → None; missing average → KeyError; fixed ties

> 不要把另一页示例的方法名或返回行为直接当作本题接口。

Slide 85 · Lab 9 · Interfaces and Edge Cases

> 实验、骨架和测试采用同一固定接口。五项特别容易混淆的边界在这里集中列出，详细范围与分数以两语实验页为准。
>
> A/B 文字卡有设计和回归的人工分，C 是零分学习记录。声明要对应真实操作，不虚构已经运行或线上提交。

Scope of L09 and L10 本讲与下讲范围

Lecture 范围	Topics 知识
L09 · this lecture	Objects, initialization, method binding, interfaces, sharing, method kinds
L10 · next lecture	Inheritance, protocols, polymorphism, multiple inheritance, modules and execution

Slide 86 · Scope of L09 and L10 本讲与下讲范围

> 旧合并课件 87 页中，前面的类与实例核心在本讲保留；p33 之后的继承、运算协议、鸭子类型与模块明确由 L10 承接。
>
> 没有把复杂内容删除，也没有在开场或测验中使用尚未解释的语法。参考区供查阅，课堂主线与时间安排仍为 64 页。