

Inheritance, Protocols and Modules · Prime Counting Opens

Retain shapes, accounts, Money, Deck and module organization with fixed interfaces. Verify state and imports, and start solo assignment2's five prime-counting methods. Big integers remain a later topic.

Due: Homework 4 · Tue 8 Dec, 23:59

Goals and entries

Specify inherited state, operands, failures and effects before implementation. Verify a real file project rather than treating a single code block as module organization.

[HW4](#) covers weeks10–12. [Solo2](#) covers10/11 and uses its configured deadline after week11. Neither gets a new “this Sunday” date here.

Part	Code	Written work	Raw points
A	Four entries,4 automatic each	Design/verification,8 staff	24
B	Module project,8 automatic	Import evidence,4 staff	12
C	None	Environment/debug record,0	0

Staff points are not automatically awarded by code tests. C is required practice without extra weight; raw scores do not change course weights.

Week10 uses Level1

Establish contracts and checks yourself before reviewing assistance. Code notes and A/B cards declare use/nonuse, tool/model (unknown if necessary), adopted/revised scope and actual results. Do not invent evidence or relax week7/solo1’s independent implementation requirements.

Entries are standalone, annotated interfaces. Self needs no separate annotation. Do not use input or rely on a preceding slide. Unspecified object types need no automatic conversion; numerical tests use moderate magnitudes.

Part A · Classes and protocols

10-1 Shape family

```
class Shape:
    def area(self) -> float: ...
    def name(self) -> str: ...
class Circle(Shape):
    def __init__(self, r: float) -> None: ...
class Rectangle(Shape):
    def __init__(self, w: float, h: float) -> None: ...
class Triangle(Shape):
    def __init__(self, a: float, b: float, c: float) -> None: ...
def total_area(shapes: list[Shape]) -> float: ...
```

Shape can be constructed; its area raises `NotImplementedError`. Name reports the actual class name. Dimensions are finite and positive; triangles satisfy strict inequalities. Invalid construction raises `ValueError`. Use `math.pi` and Heron's formula. Empty `total_area` returns 0; neither the list nor objects are changed.

`Circle(1)` has area π ; `Rectangle(2,3)` and `Triangle(3,4,5)` have area 6. Reject `Triangle(1,1,3)`. Explain the degeneracy rejection, rather than selecting a different None/error policy. Code:

w10-shapes, 4 automatic.

10-2 Account inheritance

Submit all three definitions, not subclasses assuming L09 was run first.

```

class BankAccount:
    def __init__(self, owner: str, balance: int = 0) -> None: ...
    def deposit(self, amount: int) -> None: ...
    def withdraw(self, amount: int) -> bool: ...
    def statement(self) -> str: ...
class SavingsAccount(BankAccount):
    def __init__(self, owner: str, balance: int = 0,
                  interest_rate: float = 0.0) -> None: ...
    def add_interest(self) -> None: ...
class CheckingAccount(BankAccount):
    def __init__(self, owner: str, balance: int = 0,
                  overdraft_limit: int = 0) -> None: ...

```

Money uses integer cents. Public owner/balance/history follow L09. A nonnegative integer starting balance is required, otherwise `ValueError`. Nonpositive/noninteger deposit and withdrawal amounts are rejected: deposit returns `None`, withdrawal `False`, with unchanged state. Accepted history entries are `("deposit", amount)` or `("withdraw", amount)` using ordinary ASCII string quotes in code. Statement starts `owner: balance cents`, followed by `action: amount cents`, newline-joined without a trailing newline.

Both subclass initializers use `super`. Savings takes a finite rate $0 \leq \text{rate} \leq 1$; invalid rates raise `ValueError`. Quantize `Decimal(str(rate)) * balance` to integer cents using `ROUND_HALF_UP`. Positive interest adds `("interest", cents)`, zero adds no history, and `add_interest` returns `None`. This is an educational rounding contract, not a financial production system.

Checking starts nonnegative but supports later balances down to `-overdraft_limit`. The limit is a nonnegative integer. Overridden withdrawal rejects a value below that bound without changing history/balance. The base's nonnegative-balance guarantee is not a universal family contract. Discuss whether directly calling base withdrawal meets this policy.

Code: `w10-accounts`, 4 automatic.

10-3 Money

```
class Money:
    def __init__(self, yuan: float) -> None: ...
    def __repr__(self) -> str: ...
    def __add__(self, other: "Money") -> "Money": ...
    def __sub__(self, other: "Money") -> "Money": ...
    def __eq__(self, other: object) -> bool: ...
    def __lt__(self, other: "Money") -> bool: ...
```

Accept finite int/float yuan with $\text{magnitude} \leq 10^9$; otherwise `ValueError`. Store integer cents. Convert `str(value)` to `Decimal`, multiply by 100 and quantize using `ROUND_HALF_UP`: $1.005 \rightarrow 101$ cents, $-1.005 \rightarrow -101$. This operates on the supplied value's decimal display, not an imagined original decimal intent. Read the decimal interface first; `int(float*100)` truncation is not this contract.

`Repr` has exactly two decimals, including `Money(12.50)`, `Money(-1.01)`, `Money(0.00)`. Addition/subtraction return new objects and preserve inputs; comparisons use cents. Unsupported operand methods return `NotImplemented`. Equality to an unrelated object is `False`; unsupported arithmetic/ordering generally raises `TypeError`. Results may exceed the constructor-input bound without passing through float storage.

Check `Money(0.1)+Money(0.2)==Money(0.3)`, sorting, negatives, ties and input preservation. Code: `w10-money`, 4 automatic.

10-4 Deck

```
class Deck:
    def __init__(self) -> None: ...
    def __len__(self) -> int: ...
    def __getitem__(self, i: int) -> str: ...
    def __contains__(self, card: str) -> bool: ...
    def shuffle(self) -> None: ...
    def deal(self, n: int) -> list[str]: ...
```

Use ASCII rank+suit. Suits are C/D/H/S; ranks are A/2/.../10/J/Q/K. Initial ordering groups by suit then rank: AC at0, KC at12, AD at13, KS last. There are52 unique cards, no jokers, and independent decks.

Length reports remaining cards. Integer positive/negative indexing works; invalid indices raise `IndexError`. Membership returns `bool`. Sequence-style iteration must stop at an invalid index, not return `None` forever.

Deal takes `n` cards from the **front**, removes them and returns a new list. `n` is a nonnegative integer no larger than the remainder; invalid values raise `ValueError` without mutation. Zero deals nothing. Shuffle mutates and returns `None` while retaining length and card multiplicities. No particular shuffled sequence or mandatory change on every call is required; do not reseed internally to fake randomness.

Code: `w10-deck`,4 automatic.

A design record

`w10-a-notes` : four staff responses worth2 each: shape rejection/polymorphism; account and overdraft contracts/super; Money rounding/sentinel/preservation; Deck representation, edges and isolation. A separate zero-point declaration records actual assistance.

Part B · Real module organization

10-5 Separate modules and imports

Use five files, fixing the old “three files” count:

```
shapes.py    # shapes
accounts.py # accounts
money.py    # money
cards.py    # deck
main.py     # entry
```

Each business module starts with a docstring and has a main guard. Direct execution prints one recognizable self-test line: `SELFTEST shapes/accounts/money/cards`. Importing it prints **nothing**. Main imports all four, demonstrates at least one object from each, exits

successfully and prints a result without SELFTEST. Main also has a docstring/guard; importing it does not run the demo.

The existing single-source submission entry takes `project_files() -> dict[str, str]`, returning exactly these five filenames and their complete sources. The evaluator writes only these names into its own temporary sandbox and runs/imports there; it does not accept arbitrary absolute paths or add a zip-upload feature. Project_files must not run the project, read local files or install dependencies.

Keep the real five-file project locally. The adapter packages its sources rather than replacing module work. Business interfaces must support normal Part A examples; no input, network or GUI dependency. Use only the standard library and these files.

`w10-module-project`:8 automatic. `w10-b-notes`:2 staff for responsibilities/separating Money,2 for actual direct-run/import evidence; declaration0. A guard's textual presence is not evidence that imports behave correctly.

Part C · Environment and debugging

`w10-c-tools`,0 points. Record interpreter, matching pip, module source and environment. Observe a main guard or name shadow in an independent temporary folder, then remove your example. Do not leave random.py in a real project or uninstall another person's dependencies.

Identify self, the current class and MRO rather than defining super as always the direct parent. State limitations instead of fabricating outputs/screenshots. C adds no weight.

4. Solo assignment2: counting primes five ways

[Solo2](#) opens in week10 and covers10/11, worth5%; three solo assignments total15%. Its configured calendar determines the deadline. It is not the old7.5% big-integer project.

Entry	Function	Required method
w10-primes-brute	count_primes_brute(n: int) -> int	Try factors 2 through candidate-1
w10-primes-sqrt	count_primes_sqrt(n: int) -> int	Stop at $d^2 > \text{candidate}$
w10-primes-6k	count_primes_6k(n: int) -> int	Handle 2/3, then $6k \pm 1$ factors
w10-primes-sieve	count_primes_sieve(n: int) -> int	Eratosthenes marking
w10-primes-mr	count_primes_mr(n: int) -> int	Bounded-domain Miller-Rabin

Count primes $0 \leq p < n$ for $0 \leq n \leq 100000$. Values ≤ 2 return 0; negative n raises `ValueError`. Larger inputs are not required. Examples: $n=0 \rightarrow 0$, $n=3 \rightarrow 1$, $n=10 \rightarrow 4$. Return an integer without printing.

MR handles values below 2, 2/3 and other even values before bases 2, 3 on remaining odds. The known deterministic bound below 1,373,653 covers candidates below 100000. This is not a determinism claim for arbitrary integers. L11 and assignment material develop decomposition/modular exponentiation and the other algorithms; establish interfaces and evidence now rather than requiring untaught implementations immediately.

Each method entry carries 16 automatic raw points. The staff report carries 20; 100 raw points normalize to this assignment's 5%. Tests primarily check results/edges; staff inspect sources and records for genuinely different methods. Do not wrap one algorithm five times. Each entry includes its helpers; `pow(a,d,m)` is allowed for MR.

`w10-primes-report`: method/source evidence 6, boundary/cross-checking 6, fair reproducible timing 6, conclusions/limits 2, declaration 0. Verify correctness before timing. Slow methods may use smaller n rather than another algorithm. Record actual sizes, repetitions, environment, preparation/query accounting and counts; make at least three measurements, without claiming a single ratio is universal.

5. Later reference: big integers

BigInt starts in week12, not as a current7.5% deliverable. Keep decimal strings, sign/grouped-digit representation, str/repr/eq/lt/add/sub/mul and the prohibition on converting complete operands to int as the implementation. Int remains a test oracle. Check zero, leading zeros, negatives, unequal lengths, negative differences and carry/borrow/multiplication. A one-page report covers representation, sign logic and boundary/random checks.

The old20/20/20/10/15/5/10 breakdown is historical, not the current rubric. Future requirements retain the calendar's three5% assignments; knowledge is not removed.

Before submission

- Separate HW4 code/project/cards and match annotations, errors and results.
- Check independent instances, complete state after rejection, rounding/unsupported operands and finite Deck iteration.
- Verify module documentation, silent imports, direct self-tests and the real main demonstration.
- Match solo2's counting interval, five genuine implementations, and correctness-before-timing evidence.
- Declare actual assistance/nonuse and unfinished steps; a locally saved project is not an online submission.