Multi-Agent Coding Assistant

Enter a software requirement below to see how the multi-agent system analyzes, implements, reviews, documents, tests, and prepares deployment artifacts.

Software Requirement

Describe the software you want to build

Build a Python program that manages a simple in-memory todo list.
The system should allow adding a task, listing all tasks, and removing a task by index.
Each task should be a non-empty string.
The system should handle invalid inputs gracefully.

Run Multi-Agent System

Workflow completed successfully.

Refined Requirement

System purpose:

- Manage a simple in-memory todo list that allows adding tasks, listing all tasks, and removing a task by index.

Supported operations (each implemented as a Python method on a TodoList object or as standalone functions that operate on an in-memory list):

- add_task(task: str) → int
- list_tasks() → list[str]
- remove_task_by_index(index: int) → str

Operation details:

- add_task(task: str) → int
  - Inputs:
    - task: type str; constraint: after trimming leading and trailing whitespace, length must be ≥ 1.
  - Output:
    - Returns the integer index (zero-based) at which the task was appended to the internal list.
    - The returned index is an int in the range [0, current_length_before_add].
  - Error conditions and handling:
    - If task is not of type str, raise TypeError with a clear message like "task must be a string".
    - If task trimmed of whitespace is an empty string, raise ValueError with a clear message like "task must be a non-empty string".
    - Do not modify the list if an error is raised.
- list_tasks() → list[str]
  - Inputs:
    - No inputs.
  - Output:
    - Returns a list of strings representing tasks in insertion order. Each element is exactly the stored task string (no implicit trimming on output beyond what was stored).
    - Return a shallow copy of the internal list so callers cannot mutate internal state by modifying the returned list.
  - Error conditions and handling:
    - This operation does not raise errors for normal use. If an unexpected internal error occurs, propagate the exception (implementation should avoid side effects).
- remove_task_by_index(index: int) → str
  - Inputs:
    - index: type int; constraint: zero-based index that must satisfy 0 ≤ index < current_number_of_tasks.
  - Output:
    - Returns the removed task string.

        - Internally removes exactly one task at the specified index and shifts any subsequent tasks left (standard list behavior).
    - Error conditions and handling:
        - If index is not of type int, raise TypeError with a clear message like "index must be an integer".
        - If index is out of range (index < 0 or index ≥ current_number_of_tasks), raise IndexError with a clear message like "index out of range".
        - Do not modify the list if an error is raised.

Additional implementation constraints and details:

- The system must keep all data in memory only; do not use files, databases, or external services.
- Implement the operations as methods of a TodoList class (constructor TodoList()) or as standalone functions that operate on a single in-memory list object; each listed operation must map directly to one concrete Python function or method.
- Maintain insertion order for tasks.
- Tasks may be duplicates; no uniqueness enforcement.
- list_tasks must return a copy of the internal storage to prevent external modification of internal state.
- All error messages must be explicit strings as described above so calling code can detect and present them.

Assumptions:

- Indexing is zero-based.
- Trimming of whitespace is applied only for input validation on add_task (i.e., a task of " hi " is accepted and stored exactly as " hi ", but a task consisting only of whitespace is rejected).
- The environment is single-process and single-threaded; thread-safety is not required.

Non-goals (explicitly excluded):

- No persistence to disk or external storage.
- No editing/updating tasks in place (only add, list, remove by index).
- No task metadata (timestamps, priorities, ids beyond list index).
- No command-line, network, or GUI interface; only in-memory API-level operations.
- No automatic reindexing beyond standard list behavior after removals.

Examples of required function signatures (for implementer reference):

- class TodoList:
    - def **init**(self) → None

- def add_task(self, task: str) → int
- def list_tasks(self) → list[str]
- def remove_task_by_index(self, index: int) → str

Validation and error behavior must follow the rules above exactly so calling code receives consistent exceptions and messages.

Code Review Result

Code approved after 0 review attempt(s).

Generated Python Code

```python
#!/usr/bin/env python3
"""
In-memory TodoList manager.

Provides a TodoList class with methods:
- add_task(task: str) -> int
- list_tasks() -> list[str]
- remove_task_by_index(index: int) -> str
"""

from typing import List


class TodoList:
    """
    Simple in-memory todo list managing tasks in insertion order.

    Methods:
    - add_task(task: str) -> int
    - list_tasks() -> list[str]
    - remove_task_by_index(index: int) -> str
    """

    def __init__(self) -> None:
        """Initialize an empty TodoList."""
        self._tasks: List[str] = []
```

```
    def add_task(self, task: str) -> int:
        """
        Add a task to the end of the todo list.

        Args:
            task: The task string to add. Must be of type str. For validation,
                  task.strip() must have length >= 1. The original string
                  (without trimming) is stored.

        Returns:
            The zero-based index at which the task was appended.

        Raises:
            TypeError: If task is not a string (message: "task must be a string").
            ValueError: If task.strip() is empty (message: "task must be a non-empt
        """
        if not isinstance(task, str):
            raise TypeError("task must be a string")

        if task.strip() == "":
            raise ValueError("task must be a non-empty string")

        index = len(self._tasks)
        self._tasks.append(task)
        return index

    def list_tasks(self) -> List[str]:
        """
        Return a shallow copy of the list of tasks in insertion order.

        Returns:
            A list of task strings. Modifying the returned list does not affect
            the internal storage.
        """
        return self._tasks.copy()

    def remove_task_by_index(self, index: int) -> str:
        """
        Remove and return the task at the specified zero-based index.

        Args:
            index: Zero-based integer index of the task to remove.
```

```
        Returns:
            The removed task string.

        Raises:
            TypeError: If index is not an int (message: "index must be an integer")
            IndexError: If index is out of range (message: "index out of range").
        """
        if not isinstance(index, int):
            raise TypeError("index must be an integer")

        if index < 0 or index >= len(self._tasks):
            raise IndexError("index out of range")

        return self._tasks.pop(index)


if __name__ == "__main__":
    # Minimal guard to allow safe execution when run as a script.
    pass
```

Documentation

TodoList — In-memory todo list manager

Overview

TodoList is a simple, in-memory manager for a list of tasks. It supports adding tasks, listing tasks in insertion order, and removing a task by zero-based index. All data is kept only in memory; no persistence, threading, or metadata features are provided.

Architecture / Design

- Core type: `TodoList` class.
- Internal storage: `self._tasks: List[str]` — a Python list that preserves insertion order.

- Data flow:
  - `add_task(task: str)` validates the input and appends the exact provided string to `self._tasks`.
  - `list_tasks()` returns a shallow copy of `self._tasks` so callers cannot mutate internal state.
  - `remove_task_by_index(index: int)` validates the index, removes one element from `self._tasks` at that index, and returns the removed string.

Responsibility summary:

- Validation of input types and constraints is performed by each method and raises explicit exceptions with fixed messages on error.
- Methods avoid modifying internal storage if validation fails.

Public API

class TodoList

Constructor

- `TodoList() -> TodoList`
  - Create an empty todo list instance.

Methods

- `add_task(self, task: str) -> int`
  - Purpose: Append a task to the list.
  - Inputs:
    - `task`: must be a `str`. Validation uses `task.strip()` to ensure there is at least one non-whitespace character.
    - Note: the original `task` string (including surrounding whitespace) is stored; trimming is only used for validation.
  - Output:
    - Returns the zero-based index (int) where the task was appended. The return value equals the length of the list before the append.
  - Errors:
    - Raises `TypeError("task must be a string")` if `task` is not a `str`.
    - Raises `ValueError("task must be a non-empty string")` if `task.strip()` is empty.

- Side-effects:
  - On success, appends `task` to internal list.
  - On error, internal list is not modified.

- `list_tasks(self) -> list[str]`
  - Purpose: Retrieve tasks in insertion order.
  - Inputs: none.
  - Output:
    - Returns a shallow copy of the internal list of task strings (`list[str]`). Each element is exactly the stored string.
  - Errors:
    - No application-level errors; unexpected internal exceptions are propagated.
  - Side-effects:
    - None — returns a copy so caller modifications do not affect internal state.
- `remove_task_by_index(self, index: int) -> str`
  - Purpose: Remove and return the task at `index` .
  - Inputs:
    - `index` : must be an `int` and satisfy `0 <= index < current_number_of_tasks` .
  - Output:
    - Returns the removed task string.
  - Errors:
    - Raises `TypeError("index must be an integer")` if `index` is not an `int` .
    - Raises `IndexError("index out of range")` if `index` is out of bounds.
  - Side-effects:
    - On success, removes exactly one element from the internal list (standard list pop behavior).
    - On error, internal list is not modified.

Usage example

```
from your_module import TodoList  # import path where the class is defined

todo = TodoList()

# Add tasks
i1 = todo.add_task("Buy milk")          # returns 0
```

```
i2 = todo.add_task("  Call Alice  ")   # returns 1 (stored with surrounding spaces)

# List tasks (returns a copy)
tasks = todo.list_tasks()              # ["Buy milk", "  Call Alice  "]
tasks.append("mutated")                # does not change todo internal list

# Remove a task by index
removed = todo.remove_task_by_index(0) # removes and returns "Buy milk"
# Now todo.list_tasks() == ["  Call Alice  "]
```

Example error cases (behaviour shown, do not run interactively above):

- `add_task(123)` raises `TypeError("task must be a string")`.
- `add_task("   ")` raises `ValueError("task must be a non-empty string")`.
- `remove_task_by_index("0")` raises `TypeError("index must be an integer")`.
- `remove_task_by_index(10)` raises `IndexError("index out of range")` (if list has fewer elements).

Assumptions and Limitations

- Indexing is zero-based.
- Whitespace trimming is only used for validation in `add_task`; the stored task preserves original whitespace.
- Duplicate tasks are allowed; no uniqueness enforcement.
- Single-process, single-threaded environment; thread-safety is not provided.
- No persistence: data is lost when the `TodoList` instance is discarded.
- Non-goals: no editing/updating of tasks beyond remove/add, no metadata (timestamps/priorities/IDs), no CLI/network/GUI interfaces.

Generated Test Cases

```
import pytest
from app import TodoList


def test_add_task_valid_returns_index_and_appends():
    todo = TodoList()
    assert todo.list_tasks() == []
```

```
    idx0 = todo.add_task("first task")
    assert isinstance(idx0, int)
    assert idx0 == 0
    assert todo.list_tasks() == ["first task"]
    idx1 = todo.add_task("second task")
    assert idx1 == 1
    assert todo.list_tasks() == ["first task", "second task"]


def test_add_task_type_error_does_not_modify_list():
    todo = TodoList()
    todo.add_task("keep me")
    with pytest.raises(TypeError) as exc:
        todo.add_task(123)  # not a string
    assert str(exc.value) == "task must be a string"
    # list must remain unchanged
    assert todo.list_tasks() == ["keep me"]


def test_add_task_empty_after_trim_raises_value_error_and_does_not_modify():
    todo = TodoList()
    todo.add_task("stay")
    with pytest.raises(ValueError) as exc:
        todo.add_task("    ")  # only whitespace -> invalid
    assert str(exc.value) == "task must be a non-empty string"
    assert todo.list_tasks() == ["stay"]


def test_add_task_whitespace_surrounded_is_stored_exactly():
    todo = TodoList()
    original = "  surrounded by spaces  "
    idx = todo.add_task(original)
    assert idx == 0
    # stored exactly as provided (no trimming on storage)
    stored = todo.list_tasks()
    assert stored == [original]
    assert stored[0] == original


def test_list_tasks_returns_shallow_copy_prevents_external_mutation():
    todo = TodoList()
    todo.add_task("a")
    todo.add_task("b")
```

```
    returned = todo.list_tasks()
    assert returned == ["a", "b"]
    # mutate returned list
    returned.append("c")
    returned.pop(0)
    # internal state must remain unchanged
    assert todo.list_tasks() == ["a", "b"]


def test_remove_task_by_index_valid_removes_and_returns_task_and_shifts():
    todo = TodoList()
    todo.add_task("t0")
    todo.add_task("t1")
    todo.add_task("t2")
    removed = todo.remove_task_by_index(1)
    assert removed == "t1"
    # tasks after removal should shift left
    assert todo.list_tasks() == ["t0", "t2"]
    # removing first element now
    removed0 = todo.remove_task_by_index(0)
    assert removed0 == "t0"
    assert todo.list_tasks() == ["t2"]


def test_remove_task_by_index_type_error_and_no_modification():
    todo = TodoList()
    todo.add_task("only")
    with pytest.raises(TypeError) as exc:
        todo.remove_task_by_index("0")  # wrong type
    assert str(exc.value) == "index must be an integer"
    assert todo.list_tasks() == ["only"]


def test_remove_task_by_index_out_of_range_error_and_no_modification():
    todo = TodoList()
    todo.add_task("single")
    with pytest.raises(IndexError) as exc:
        todo.remove_task_by_index(5)  # out of range
    assert str(exc.value) == "index out of range"
    # negative out of range
    with pytest.raises(IndexError) as exc2:
        todo.remove_task_by_index(-1)
    assert str(exc2.value) == "index out of range"
```

```
    assert todo.list_tasks() == ["single"]


def test_integration_add_list_remove_flow():
    todo = TodoList()
    # add three tasks
    i0 = todo.add_task("task0")
    i1 = todo.add_task("task1")
    i2 = todo.add_task("task2")
    assert (i0, i1, i2) == (0, 1, 2)
    assert todo.list_tasks() == ["task0", "task1", "task2"]
    # remove middle
    removed = todo.remove_task_by_index(1)
    assert removed == "task1"
    # now list should have task0 and task2
    assert todo.list_tasks() == ["task0", "task2"]
    # add another and ensure index reflects current length before add
    i3 = todo.add_task("task3")
    assert i3 == 2  # appended at end
    assert todo.list_tasks() == ["task0", "task2", "task3"]


def test_duplicates_allowed_and_indexing_behaviour():
    todo = TodoList()
    idx_a = todo.add_task("dup")
    idx_b = todo.add_task("dup")
    assert idx_a == 0
    assert idx_b == 1
    assert todo.list_tasks() == ["dup", "dup"]
    # remove first duplicate, second should shift to index 0
    removed = todo.remove_task_by_index(0)
    assert removed == "dup"
    assert todo.list_tasks() == ["dup"]
```

Deployment Configuration

requirements.txt

run.sh

```bash
#!/usr/bin/env bash
set -euo pipefail

# Install dependencies if requirements.txt exists and is non-empty
if [ -s requirements.txt ]; then
  python3 -m pip install --user -r requirements.txt
fi

# Locate the single Python application file and run it.
# Prefer common names if multiple .py files exist.
shopt -s nullglob
py_files=( *.py )
if [ ${#py_files[@]} -eq 0 ]; then
  echo "No Python (.py) file found in the current directory." >&2
  exit 1
fi

if [ ${#py_files[@]} -gt 1 ]; then
  for preferred in main.py app.py todo.py; do
    if [ -f "$preferred" ]; then
      exec python3 "$preferred"
    fi
  done
  # Fallback to the first .py file found
  exec python3 "${py_files[0]}"
else
  exec python3 "${py_files[0]}"
fi
```