PyDynamicList

PyDynamicList is a feature-rich, robust custom list class for Python built as a comprehensive OOP project. It goes beyond a standard Python list by providing strict type enforcement, advanced statistical analysis, operator overloading, and a full suite of custom exceptions for safe and predictable data manipulation.

This project was built from the ground up to demonstrate mastery of core Object-Oriented Programming concepts, including encapsulation, inheritance, polymorphism, and abstraction.

Core Features

- **Strict Type Enforcement**: Restrict list elements to specific types (e.g., int, str, or a tuple like (int, float)).
- **Full Dunder Method Support**: Behaves like a real list. Supports indexing, slicing ([1:4]), iteration (for item in...), len(), and more.
- **Operator Overloading**:
  - + / -: Perform element-wise math (list1 + list2).
  - * / rmul: Repeat the list's sequence (list * 3).
  - ==, !=, <, >: Compare lists based on their content.
- **Advanced Statistical Suite**: Built-in methods for common data analysis:
  - .mean(), .median(), .mode()
  - .min(), .max(), .sum(), .product()
  - .variance(), .std() (Standard Deviation)
  - .percentile(p), .data_range()
  - .describe(): A one-stop method (like pandas) for a full statistical summary.
- **Functional Methods**: Includes .map() and .filter() methods that work with lambda functions and return new DynamicList instances.
- **Robust Error Handling**: A full hierarchy of custom exceptions (InvalidElementTypeError, EmptyListError, etc.) to prevent common errors and make debugging easy.
- **Interoperability**: Easily convert to and from standard Python lists using .to_list() and @classmethod .from_list().

How to Use

1. Initialization and Type Checking

You can create a list that accepts the default (int, float) or specify your own types.

```
# Create a list that only accepts integers
int_list = DynamicList(allowed_type=int)

# This works
int_list.append(10)
```

```
int_list.append(20)
print(int_list)
# Output: [10, 20]

# This fails with a clear error
try:
    int_list.append("hello")
except InvalidElementTypeError as e:
    print(e)
# Output: This list only accepts types <class 'int'>, but got str.
```

2. Core List Operations (Indexing, Slicing)

It behaves just like a standard list.

```
data = DynamicList.from_list([10, 20, 30, 40, 50], int)

# Get item by index
print(data[2])
# Output: 30

# Set item by index
data[0] = 99
print(data)
# Output: [99, 20, 30, 40, 50]

# Get a slice (returns a new DynamicList)
data_slice = data[1:4]
print(data_slice)
# Output: [20, 30, 40]

# Delete an item
del data[1]
print(data)
# Output: [99, 30, 40, 50]
```

3. Operator Overloading

Perform element-wise math or sequence operations.

```
list_a = DynamicList.from_list([1, 2, 3], int)
list_b = DynamicList.from_list([10, 20, 30], int)
```

```
# Element-wise addition
list_c = list_a + list_b
print(list_c)
# Output: [11, 22, 33]

# Sequence repetition
list_d = list_a * 3
print(list_d)
# Output: [1, 2, 3, 1, 2, 3, 1, 2, 3]

# Concatenation
list_e = list_a.concat(list_b)
print(list_e)
# Output: [1, 2, 3, 10, 20, 30]
```

4. Functional Methods (map & filter)

Use lambda functions to transform your list.

```
data = DynamicList.from_list([1, 2, 3, 4, 5], int)

# Map: Apply a function to all elements
squared = data.map(lambda x: x * x)
print(squared)
# Output: [1, 4, 9, 16, 25]

# Filter: Keep only elements that pass a test
evens = data.filter(lambda x: x % 2 == 0)
print(evens)
# Output: [2, 4]
```

5. Statistical Analysis

The real power of PyDynamicList is its built-in statistical suite.

```
stats_list = DynamicList.from_list([10, 2, 38, 23, 38, 23, 21, 47], int)

print(f"Mean:   {stats_list.mean()}")
print(f"Median: {stats_list.median()}")
print(f"StdDev: {stats_list.std()}")
print(f"75th %: {stats_list.percentile(75)}")
```

```
# Output:
# Mean:   25.25
# Median: 23.0
# StdDev: 15.1185...
# 75th %: 38.0

# Get a full summary with .describe()
stats_list.describe()
# Output:
#
# Summary Statistics ((<class 'int'>, <class 'float'>))
# ---------------------------------
#   count : 8
#    mean : 25.25
#     std : 15.119
#     var : 228.5
#     min : 2
#     25% : 18.25
#     50% : 23.0
#     75% : 38.0
#     max : 47
#   range : 45
```

Custom Exceptions

This class provides a clear set of custom exceptions to make error handling precise.

Exception	**When It's Raised**
InvalidElementTypeError	When adding an item that does not match the list's allowed_type.
EmptyListError	When calling a statistical method (like .mean()) on an empty list.
WrongDataTypeError	1. When comparing (==) two lists with different allowed_types. 2. When calling a statistical method on a list that allows non-numeric types (e.g., str).

LengthNotEqualError	When trying element-wise math (+, -) on lists of different lengths.
IndexOutOfRangeError	When pop(), del, or insert() use an index that doesn't exist.
UnknownValueError	1. When .index() can't find the specified value. 2. When .percentile() gets a value not between 0-100.