CS 114 Tutorial 10

Nov 21 2025

Goals for this Week:

- Have assignment 05 started (due next Friday)
- Be familiar with classes
- Know how to use special methods (init, repr, eq)
- Understand how to create your open class methods
- Know what an invariant is

Suppose we set up the Statistic class, and someone broke your code by doing this: sl.lst.append(10.0)
Now sl.sum, sl.product, sl.min, and sl.max are incorrect.
Write a method rebuild_invariants(self) → None that recomputes all four fields (sum, product, min, max) from scratch using only self.lst.

```
sl = StatsList()
sl.append(2.0)
sl.append(3.0)
# Oops — this line bypasses the invariants:
sl.lst.append(10.0)
print([f"Sum = {sl.sum}", f"Product = {sl.product}",
       f"Min = {sl.min}",  f"Max = {sl.max}"])
sl.rebuild_invariants()
print([f"Sum = {sl.sum}", f"Product = {sl.product}",
       f"Min = {sl.min}",  f"Max = {sl.max}"])
```

Go to vevox.com

Sign in using the session ID: 184-083-553

Special Methods

```
class Student:
    """Describe a student"""
    name: str
    number: int
    average: float

    def __init__(self, name:str, number:int, average:float) -> None:
        self.name = name
        self.number = number
        self.average = average

    def __repr__(self) -> str:
        return f"Student: {self.name} ID: {self.number}"

print(Student("Random Name", 12345678, 100.00))
```

What will the code print?

A. “Random Name 12345678” B. “Student: Random Name ID: 12345678”

C. error

Special Methods

```
def __eq__(self, other) -> bool:
    return ((self.name == other.name)
            and (self.number == other.number))

student1 = Student("1st Student", 12345678, 100.0)
student2 = Student("2nd Student", 12345678, 100.0)

student1 == student2
```

If the following special method is added to the Student class, what will this comparison return?

A. True B. False C. Error

Special Methods

What is the special method for the following operation on classes: x < = y

A. ___lt___
B. ___less_than___
C. ___less_than_or_equal_to___
D. ___le___
E. ___lte___

```python
class Person:
    name: str
    age: int

    def __init__(self, name: str, age: int) -> None:
        self.name = age
        self.age = name

    def birthday(self) -> None:
        self.age += 1
        print("Happy Birthday!")

    def __repr__(self) -> str:
        return f"My name is {self.name}"

person1 = Person("Bob", 10)
print(person1)
```

What does this code print?

a. Happy Birthday! b. My name is Bob c. Bob is 10 d. My name is 10

```python
class Person:
    name: str
    age: int

    def __init__(self, name: str, age: int) -> None:
        self.name = name
        self.age = age

    def birthday(self) -> None:
        self.age += 1
        print("Happy Birthday!")

    def __repr__(self) -> str:
        return f"My name is {self.age}"

person1 = Person("Bob", 10)
person1.birthday()
```

What does this code print?

a. Happy Birthday! b. My name is Bob c. Bob is 11 d. Nothing

Invariant Concept

What is an invariant?

A. Any variable that is stored in an object or used in a loop.
B. A rule or condition that should stay true every time we “check” it, as long as the program is working correctly.
C. A value that can never change in the whole program.
D. A line of code that only runs once at the start.

Create a class Card with two attributes, value and suit which are strings. The class should work as follows:

- Card("A", "Hearts") should create a Card object representing the ace of hearts and printing your card should say “A of Hearts”
- Two Card objects are equal if have the same value regardless of their suit
- Card.compare(card2) should compare your card to another (card2) and return True if your card has a larger value.
  Recall the order of values from greatest to least are: 'K', 'Q', 'J', '10', '9' ... '2', 'A'. You might have to be creative with how to do this!
- Two cards are considered a good pair if they are consecutive, both the same value or both the same suit, add a method called good_pair that prints a message about which condition your pair meets but returns None.

```
card1 = Card("A", "Hearts")
card2 = Card("5", "Spades")
card3 = Card("A", "Clubs")

print(card1)
print(card1 == card3)
print(card2.compare(card1))
```

```
A of Hearts
True
True
```