CS 207 Systems Development for Computational Science

A time series class

Intro: This is your first project requirement

Today will be the first installment of your term-long time series project. As the class progresses, you'll gradually build this codebase into a flexible tool for manipulating and analyzing time series data. Because this is a little different than most classwork, we'd like you to keep a couple things in mind:

1. This is a team-based project, and we expect everyone to contribute. This sounds obvious, but sometimes it's easy to let one person do the lion's share of the work. We will be tracking each team's github repositories for commits, and discussions, as well evaluating

2. Subsequent project requirements will continue to build on each other, so it really behooves you to write solid code and be disciplined about staying on top of the assignments. If you decide to build a shoddy implementation of some feature one week, you'll likely be paying for that decision down the road.

Also, if you recall from the class syllabus, you'll be graded at the project milestones, which will evaluate how well your entire codebase behaves technically. The latter means that we'll be grading several weeks worth of your work at once.

Why do we do it this way? Well, because that's the way large software projects work. It doesn't really matter if components X, Y, and Z work if component W has a package-breaking bug; nobody will be able to use your software. This is why discipline and extensive testing matter: the larger and more complex your software gets, the easier it is for bugs to creep in, especially between interfaces.

Obviously, we're not draconian in this respect. We're not going to just give you 100% if it doesn't have bugs and 0% if it does. We'll be testing each individual component and looking at its behavior. Still, if you have problems with your fundamental data structures, it's easy for those problems to fan out into other parts of your software. So be vigilant!

And finally **dont wait** until the milestone is due. It will be a disaster!

Organizing your team

The first task is to choose your team. Hopefully you have already done this.

Teams must be 4-5 people. If you don't have a team right now, your first priority should be to find one. You can play around here without one, but you won't be able to submit it for credit unless you're a part of one.

Create a github organization

Create a github organization for your team. The steps are not that onerous, and github has decent documentation for it. You can get started here: Creating a new organization from scratch (https://help.github.com/articles/creating-a-new-organization-from-scratch/)

Only **one** person from the team needs to do this.

You can name your team/organization whatever you like, and you should select the "free" plan. Next, make sure all of your team members are invited to the organization. You should invite them as "members", not "outside collaborators". Also, you don't need to worry about "teams" for now: this is helpful for managing differing permissions across a large organization, but we won't need it.

Create a new project repository

As your organization's first action, we'd like you to create a new repo for your project. Please name this repo "cs207project".

Back to top ↑

The `TimeSeries` class

We'll be creating a basic interface and implementation of the time series class today. This will look somewhat similar to this week's homework, but we'll be building on and using this class for the rest of the term.

Basic functionality Courtney (Oct 2 commits)

✓ Create a folder called `timeseries` at the top level in your repository. Inside this folder create a file (name your choice) in which you implement the class below.

✓ Please write a class named `TimeSeries` that stores a single, ordered set of numerical data. You can store this data as a Python list.

Now implement the following basic functions:

- ✓ `__init__` , your constructor, which should take one *mandatory* argument which represents initial data to fill the time series instance with. This argument should be any object that can be treated like a sequence. The argument can be zero-length, but it can't be omitted.
- ✓ `__len__` , `__getitem__` , and `__setitem__` , which all do exactly what they sound like. Lu Shen

Back to top ↑

An abbreviating `__str__` Courtney (Oct 2 commit)

✓ We'd like you to implement formal (`__repr__`) and informal (`__str__`) string representations of your time series class. However, we'd like you to do it such that large time series don't overwhelm the user's terminal when printed. For instance, when the user calls `print( TimeSeries([1,2,3]) )` , it should print something that looks like a `TimeSeries` class with those three elements. But if a user called `print( TimeSeries(range(0,1000000)) )` , it should print something that is substantially *less* than a million elements long. For instance, it might show the first couple elements and then an ellipsis, or maybe the length and the first/last element. The specific behavior is up to you, but it should have this general flavor.

Back to top ↑

Docstrings Sarah (Oct 15 commit)

✓ Documentation is essential, both for you, your teammates, and anyone else who will be reading or using this code. We'd like you to add docstrings to some of the components you've built. Specifically, please document: the `TimeSeries` class, its constructor, and the `__str__` function. Try to include meaningful notes, instead of "this is a time series class" or "this returns a string". For the constructor, maybe describe its argument and what values it can take; for the string function, maybe describe how it abbreviates the output.

tests

Test your constructor and the other methods you have defined. (See the requirements for the constructor above: you dont need to formally test against the sequence protocol, but testing common sequences is key. Also test against things like ranges, which are "unrealized" sequences.)

Back to top ↑ Laura Wan volunteered to do this

Smoke Test

Vernacular for turning on an electrical device the first time and seeing whether a bug causes it to catch fire, a smoke test is an incomplete, quick test that makes sure nothing truly fundamental is wrong. Let's try one on your new class.

Laura Wan volunteered to do this

```
from yourfile import TimeSeries

# projecteuler.net/problem=1
# Note: this is decidely *not* the intended purpose of this class.

threes = TimeSeries(range(0,1000,3))
fives = TimeSeries(range(0,1000,5))

s = 0
for i in range(0,1000):
  if i in threes or i in fives:
    s += i

print("sum",s)
```