Exercise: Appointment Class

Table of Contents

Background

For this exercise, you will write an `Appointment` class that tracks information about an appointment such as the start and end times and a name for the appointment. Your Appointment class will also have an `overlaps()` method that takes another `Appointment` object as an argument and determines whether the two appointments overlap.

Instructions

Write a script named `appointment.py` containing a class called `Appointment` with methods as described below:

`__init__()` method

Define an `__init__()` method with four parameters:

- `self`
- `name`, a string indicating the name of the appointment (for example, "Sales brunch").
- `start`, a tuple consisting of two integers representing the start time of the appointment. The first integer represents an hour in 24-hour time; the second integer represents a number of minutes. For example, the tuple (10, 30) would represent 10:30 am.
- `end`, a tuple similar to `start` representing the end time of the appointment.

The `__init__()` method should create attributes `name`, `start`, and `end`, and use them to store the information from the parameters.

`overlaps()` method

Define an `overlaps()` method with two parameters, `self` and `other`, where `other` is another `Appointment` object.

This method should return `True` if `self` and `other` overlap and `False` if they do not. Be sure to return a boolean value rather than a string.

Note that two appointments overlap if

- the start time of one appointment occurs between the start and end times of the other appointment (including if the start times are equal), OR
- the end time of one appointment occurs between the start and end times of the other appointment (including if the end times are equal)

For this assignment, if the start time of one appointment is equal to the end time of the other appointment, the two appointments are not considered to overlap.

Hints

1. Assuming you have two values `a` and `c` such that `a` is less than `c`, you can determine if a third value `b` is between `a` (inclusive) and `c` (exclusive) with an expression like this:

```
a <= b < c
```

2. When Python compares two sequences (such as tuples), it compares each item in the first sequence to the corresponding item in the second sequence until it finds a difference. For example, given the expression `(10, 4, 12) < (10, 5, 1)`, Python first compares `10` to `10`. Because they are the same, Python then compares `4` to `5`. Because `4` is smaller than 5, Python stops comparing and the expression evaluates to `True`: `(10, 4, 12)` is considered less than `(10, 5, 1)` in Python.
3. A boolean expression such as <code>a <= b < c</code> evaluates to <code>True</code> or <code>False</code>. You don’t need to put it inside a conditional statement to return a boolean value. In other words, instead of this:

```
if a <= b < c:
    return True
else:
    return False
```

you can do this:

```
return a <= b < c
```

Docstrings

Write a docstring for your class that documents the purpose of the class as well as the purpose and expected data type of each attribute.

For each of the methods you wrote, write a docstring that documents the purpose of the method as

well as the purpose and expected data type of each parameter other than `self` (you never need to document `self` in a docstring). If your document returns a value, document that. If your document has side effects, such as modifying attributes or writing to standard output, document those as well.

Be sure your docstrings are the first statement in the class or method they document. Improperly positioned docstrings are not recognized as docstrings by Python.

Using your class

You can use your class by importing your module into another script. Below is an example of such a script; it assumes the `Appointment` class is defined in a script called `appointment.py` which is located in the same directory as this script.

use_appointment.py

```
from appointment import Appointment


def main():
    """ Demonstrate use of the Appointment class. """
    appt1 = Appointment("Physics meeting", (9, 30), (10, 45))
    appt2 = Appointment("Brunch", (10, 30), (11, 00))
    appt3 = Appointment("English study session", (13, 00), (14, 00))
    if appt1.overlaps(appt2):
        print(f"{appt1.name} overlaps with {appt2.name}")
    else:
        print(f"{appt1.name} does not overlap with {appt2.name}")
    if appt1.overlaps(appt3):
        print(f"{appt1.name} overlaps with {appt3.name}")
    else:
        print(f"{appt1.name} does not overlap with {appt3.name}")
    assert appt1.overlaps(appt2)
    assert appt2.overlaps(appt1)
    assert not appt1.overlaps(appt3)
    assert not appt3.overlaps(appt1)
    assert not appt2.overlaps(appt3)
    assert not appt3.overlaps(appt2)


if __name__ == "__main__":
    main()
```