A. Importing modules [10 min]

Review the following code and its output:

SAMPLE CODE

```
1   from datetime import date
2   from datetime import time
3
4   today = date(2020, 1, 24)
5   morningNewsTime = time(9)
6   bedTime = time(22, 30)
7
8   print('Today is', today)
9   print('Morning news time is', morningNewsTime)
10  print('Bedtime is', bedTime)
```

SAMPLE OUTPUT

```
Today is 2020-01-24
Morning news time is 09:00:00
Bedtime is 22:30:00
```

1. On line 4, what do the following values represent?
    a. 2020 -
    b. 1 -
    c. 24 -
2. How many `date` instances are created in this code?
3. How many `time` instances are created in this code?
4. Which line of code imports the module that allows us to use the `date` class?
5. Predict what will happen if you remove the line of code identified in question 4?
6. Copy and paste the sample code into a Python script and run this script. The output you get should match the sample output. Now *comment out* the line you identified in question 4 (by adding a # at the start of the line) and run the script. Did your prediction in question 5 match with what happened?

The `datetime` module is one of the *included batteries* that comes with Python. The `date` and `time` classes are defined in this module. To use the `date` and `time` classes we must import them from the module.
[PAUSE FOR A CLASS DISCUSSION]

B. Exploring Constructors [10 min]

1. Our earlier sample code created two time instances (on lines 5 and 6). What is the difference between how the two time instances are created?

2. If we wanted to create a time instance that represents 10:35:20 (ten thirty-five and 20 seconds), how do you think we could do that? (Add another time instance that represents 10:35:20 to the script you saved earlier, name this instance `myTime` and print the value of this instance).

3. To create instances of classes in Python, we use constructors. A constructor for a class may take several forms, and thus there are several ways to create instances of the same class. We've seen three different forms of the constructor of the `time` class. Describe these three constructors in terms of their parameters: the number of parameters and what each parameter represents.
    a.

    b.

    c.

4. Let's attempt to create instances of the `date` class using various values as arguments. Try the following code (one at a time) in your Python shell, and note which statements worked and which did not. (Make sure you first import the `date` class from the `datetime` module).
    a. someDate = date(1, 24, 2020)
    b. someDate = date(2020, 1, 31)
    c. someDate = date(2020, 2, 31)
    d. someDate = date(2020, 2, 29)
    e. someDate = date(2019, 2, 29)

    Explain why some of these statements did not work.

[PAUSE FOR A CLASS DISCUSSION]

C. Instance Attributes and Methods [10 min]

In Chapter 3 you read about the Object-Oriented Paradigm. As a recap, an object is represented internally by some pieces of data, we call that data instance ***attributes***. These attributes represent the state of the class instance. For example, instances of the `time` class have several attributes: `hour`, `minute`, `second`, and `microsecond`. We can access the attributes for a given instance using the following syntax:

```
identifier.attribute
```

For example, to access the `hour` attribute of the `morningNewsTime` instance of the `time` class, you can use the following line of code:

```
morningNewsTime.hour
```

You can print out the value of `morningNewsTime.hour` by using the following line of code:

```
print(morningNewsTime.hour)
```

1. Add code to your script that prints the `hour`, `minute`, and `second` attributes of `morningNewsTime` on separate lines.

2. Add code to your script that prints the `hour`, `minute`, and `second` attributes of `bedTime` on separate lines.

3. Why did you need to use the identifier name in front of the `hour`, `minute`, and `second` attributes (as in `morningNewsTime.hour` and `bedTime.hour`) when accessing those values?

Recall from your reading, that a class can also include some *behaviors* or ***methods*** that are appropriate for the abstraction that the class represents. The `date` class has a few interesting methods listed below:

- `toordinal()` - Returns the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1.
- `weekday()` - Returns the day of the week as an integer, where Monday is 0 and Sunday is 6.
- `isoweekday()` - Returns the day of the week as an integer, where Monday is 1 and Sunday is 7.

In Python, we access the methods of a given object using the following syntax:

```
identifier.method(arguments)
```

For example, to access the `toordinal()` method of `today`, you would use `today.toordinal()` (Note that in this example, there are no arguments because `toordinal()` method does not have any parameters.)

4. What is the ordinal date of July 18, 1978? (Write Python code to determine the answer.)

5. What is the day of the week of July 18, 1978? (Write Python code to determine the answer.)

[PAUSE FOR A CLASS DISCUSSION]