CS50's Introduction to Programming with Python

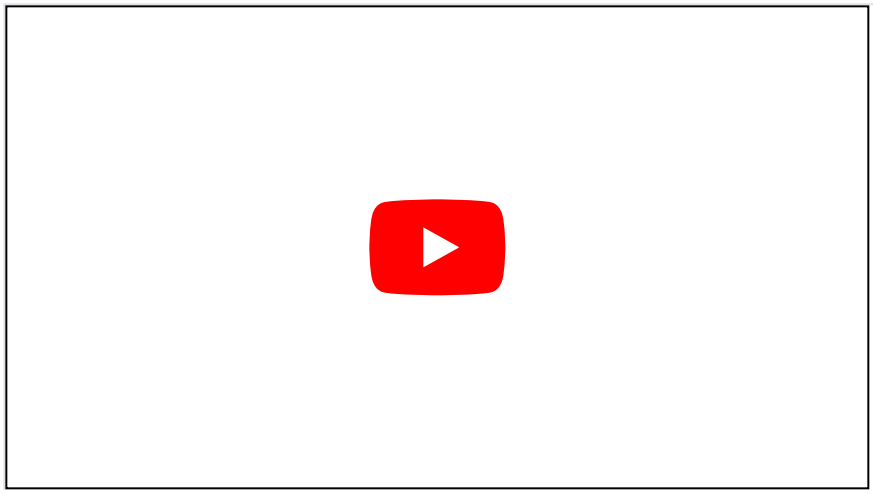
OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan)
(https://www.tiktok.com/@davidjmalan) (https://twitter.com/davidjmalan)

Seasons of Love

> Five hundred twenty-five thousand, six hundred minutes
> Five hundred twenty-five thousand moments so dear
> Five hundred twenty-five thousand, six hundred minutes
> How do you measure, measure a year?
>
> — "Seasons of Love (https://en.wikipedia.org/wiki/Seasons_of_Love)," *Rent* (https://en.wikipedia.org/wiki/Rent_(musical))



Assuming there are 365 days in a year, there are $365 \times 24 \times 60 = 525,600$ minutes in that same year (because there are 24 hours in a day and 60 minutes in an hour). But how many minutes are there in two or more years? Well, it depends on how many of those are leap years (https://en.wikipedia.org/wiki/Leap_year) with 366 days, per the Gregorian calendar (https://en.wikipedia.org/wiki/Gregorian_calendar), as some of them could have $1 \times 24 \times 60 = 1,440$ additional minutes. In fact, how many minutes has it been since you were born? Well, that, too, depends on how many leap years there have been since! There is an algorithm (https://en.wikipedia.org/wiki/Leap_year#Algorithm) for such, but let's not reinvent that wheel. Let's use a library instead. Fortunately, Python comes with a `datetime` module that has a class called `date` that can help, per docs.python.org/3/library/datetime.html#date-objects (https://docs.python.org/3/library/datetime.html#date-objects).

In a file called `seasons.py`, implement a program that prompts the user for their date of birth in `YYYY-MM-DD` format and then ~~sings~~ prints how old they are in minutes, rounded to the nearest integer, using English words instead of numerals, just like the song from *Rent*, without any `and` between words. Since a user might not know the time at which they were born, assume, for simplicity, that the user was born at midnight (i.e., 00:00:00) on that date. And assume that the current time is also midnight. In other words, even if the user runs the program at noon, assume that it's actually midnight, on the same date. Use `datetime.date.today` to get today's date, per docs.python.org/3/library/datetime.html#datetime.date.today (https://docs.python.org/3/library/datetime.html#datetime.date.today).

Structure your program per the below, not only with a `main` function but also with one or more other functions as well:

```
from datetime import date


def main():
    ...


...


if __name__ == "__main__":
```

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

You're welcome to import other (built-in) libraries. Exit via `sys.exit` if the user does not input a date in `YYYY-MM-DD` format. Ensure that your program will not raise any exceptions.

Either before or after you implement `seasons.py`, additionally implement, in a file called `test_seasons.py`, **one or more** functions that test your implementation of any functions besides `main` in `seasons.py` thoroughly, each of whose names should begin with `test_` so that you can execute your tests with:

```
pytest test_seasons.py
```

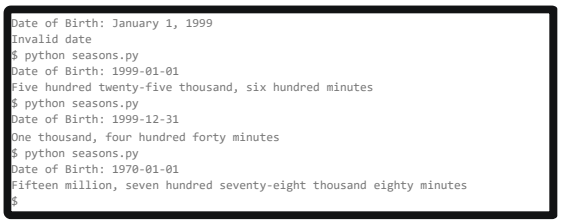
▼ **Hints**

- Note that the `date` class comes with quite a few methods and "supported operations," per docs.python.org/3/library/datetime.html#date-objects (https://docs.python.org/3/library/datetime.html#date-objects). In particular, the class implements `__sub__`, per docs.python.org/3/library/operator.html#operator.__sub__ (https://docs.python.org/3/library/operator.html#operator.__sub__), overloading `-` in such a way that subtracting one `date` object from another returns a `timedelta` object, which itself comes with several (read-only) "instance attributes," per docs.python.org/3/library/datetime.html#timedelta-objects (https://docs.python.org/3/library/datetime.html#timedelta-objects).
- Note that the `inflect` module comes with quite a few methods, per pypi.org/project/inflect (https://pypi.org/project/inflect/). You can install it with:

  ```
  pip install inflect
  ```

Demo

Assume that this demo was recorded on January 1, 2000.



Recorded with asciinema

Before You Begin

Log into code.cs50.io (https://code.cs50.io/), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
mkdir seasons
```

to make a folder called `seasons` in your codespace.

Then execute

```
cd seasons
```

to change directories into that folder. You should now see your terminal prompt as `seasons/ $`. You can now execute

```
code seasons.py
```

to make a file called `seasons.py` where you'll write your program. Be sure to also execute

```
code test_seasons.py
```

to create a file called `test_seasons.py` where you'll write tests for your program.

How to Test

How to Test `seasons.py`

Here's how to test `seasons.py` manually:

- Run your program with `python seasons.py`. Ensure your program prompts you for a birthdate. Type a date one year ago from today, in the specified format, then press Enter. Your program should ~~sing~~ print `Five hundred twenty-five thousand, six hundred minutes`

- program should say print `Five hundred twenty-five thousand, six hundred minutes`.
- Run your program with `python seasons.py`. Type a date two years ago from today, in the specified format, then press Enter. Your program should print `One million, fifty-one thousand, two hundred minutes`.
- Run your program with `python seasons.py`. Type a date of your choice, but this time use an invalid format. Press Enter and your program should exit using `sys.exit` without raising an Exception.

How to Test `test_seasons.py`

To test your tests, run `pytest test_seasons.py`. Try to use correct and incorrect versions of `seasons.py` to determine how well your tests spot errors:

- Ensure you have a correct version of `seasons.py`. Run your tests by executing `pytest test_seasons.py`. `pytest` should show that all of your tests have passed.
- Modify one of the functions you've implemented in `seasons.py` and imported into `test_seasons.py`. One of your functions might, for example, fail to raise a `ValueError` when it should. Run your tests by executing `pytest test_seasons.py`. `pytest` should show that at least one of your tests has failed.
- Continue to modify the behavior of `seasons.py`, creating (predictably) incorrect versions of your implementation. Run your tests by executing `pytest test_seasons.py`. Do the tests you expect to fail, fail?

You can execute the below to check your code using `check50`, a program that CS50 will use to test your code when you submit. But be sure to test it yourself as well!

```
check50 cs50/problems/2022/python/seasons
```

Green smilies mean your program has passed a test! Red frownies will indicate your program output something unexpected. Visit the URL that `check50` outputs to see the input `check50` handed to your program, what output it expected, and what output your program actually gave.

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2022/python/seasons
```