Step assignment 1:
Monster High fandom

Introduction

In this assignment, you'll work and extended version of the code we used in class to generate random fan tweets from the account @MonsterHighlights about the current episode of the soap opera, *Monster High*.

This code works by first creating a plausible plot point that might occur in an episode, and then print it in the form of a tweet. Plot points will be represented as tuples, so if you haven't done the reading on tuples, do that now.

For example, we might represent the plot point that Jayden confesses their love to Tiana with the tuple: `[confess_love jayden tiana]`. That might then be printed as "`It was so great when Jayden confessed their love to Tiana!`" However, this only makes sense if we know that Jayden has a crush on Tiana, otherwise, why would they confess? And it doesn't make sense if the two are already dating. So you need to make sure that your plot point generator only generates `confess_love` plot points when there's an attraction that isn't being acted on.

Note also that we've provided you with a version of `Mention` that understands things like pronoun generation and capitalization of character names. You don't need to do anything with it, just realize that it's there. If you don't know what `Mention` is, read part 6 of the tutorial on generating text in context.

Getting started

To begin with, drag the Monster Highlights folder into the Documents/Step folder on your machine (the Step folder inside your Documents folder). Then, open the Monster Highlights folder you just put inside the Step folder in Visual Studio Code.

Finally, start Step, and type "project Monster Highlights".

Part One (optional): Making a student body

This includes a bare bones version of `Students.step` and `Romances.step`. There are only a few characters defined in them. So feel free to add to it or to replace it with the code you and your troupe added to those files in class. However, you don't need to do this if you don't want to.

Note that the `Mention` code we gave you will use they as the pronoun for all characters by default. If you prefer, you may optionally specify preferred pronouns for some or all of your characters. To do that, just add statements to your `Students.step` file of the form:

```
PreferredPronoun student  pronoun.
```

Where student is the *student* you're specifying a pronoun for, and *pronoun* is either `he` or `she`. You can specify `they`, but since that's the default anyway, there's no particular point in doing so. Anyone you don't specify a pronoun for will be referred to using `they`.

Part two: Generating plot points

Here's how tweet generation works. The task called `Tweet` generates a random plot point and prints it as a tweet. If you press Alt-T/Option-T, Step will print a bunch of random tweets.

Tweet works by calling `PlotPoint`, which generates plot points, and `PrintPlotPoint`, which prints them. These are the tasks you will fill in, and you'll fill them in at the end of `Twitter.step`.

`PlotPoint` is a predicate: `[PlotPoint ?event]` means that `?event` is a plausible thing to happen in a Monster High episode given the characters and relationships specified in `Students.step` and `Romances.step`. We will represent plot points as tuples:

- `[hot_character ?who ?monsterType]`
  This isn't really something that happens in the episode; it just means the fan is expressing their enthusiasm for how hot `?who` is and that they like `?monsterType` characters (vampires, humans, etc.). You are free to generate this for any character, but the monster type should be correct for the character.
- `[confess_love ?a ?b]`
  means `?a` confessed their love for `?b`. This plot point should only be generated when `?a` has a crush on `?b` but they aren't already dating.
- `[smoldering_look ?a ?b ?club]`
  means `?a` gave `?b` a smoldering look when they were together at `?club`. This should only be generated when the characters have crushes on one another and they both belong to the club.
- `[got_together ?a ?b]`
  means `?a` and `?b` started dating. This should only be generated when they have crushes on one another and they aren't already dating.
- `[breakup ?a ?b]`
  means `?a` and `?b` broke up. This only makes sense if they're actually dating.
- `[rejected ?a ?b]`
  means `?a` rejected `?b` when `?b` confessed to `?a`. This only makes sense if `?b` has a crush on `?a` but not vice-versa.
- `[conflict ?attacker ?defender ?reason]`
  means `?attacker` started a fight (verbal or physical) with `?defender` because of ?reason. Reason is another tuple, describing why the fight broke out:
  - `[conflict ?cheatee ?cheater [cheating ?cheater ?cheetee ?other]]`
    `[conflict ?cheatee ?other [cheating ?cheater ?cheetee ?other]]`
    These mean `?cheater` and `?cheetee` are dating but `?cheater` is also dating `?other` and `?cheatee` starts the fight. So only generate this event when cheating is happening in the story world.
  - `[conflict ?attacker ?defender [triangle ?attacker ?defender ?loveInterest]]`

Means there's a love triangle and both attacker and defender are interested in the love interest. Only generate this event when the triangle exists in the story world.

- `[cheating ?a ?b]`
  means `?a` is cheating on `?b`. Only generate this when cheating is happening in the story world.
- `[star_crossed_lovers ?a ?atype ?b ?btype]`
  means `?a` and `?b` are dating, but their monster types (`?atype` and `?btype`) are supposed to be in conflict with one another. Only generate this when they are dating and their types are incompatible. Incompatible types are specified using the `RivalMonsterTypes` predicate, count in `Students.step`.

So for this part, you need to add methods for `PlotPoint` to the end of `Twitter.step`. The methods are going to look like:

`PlotPoint [`*eventType other-information …*`]:` *condition*

Where:

- *eventType* is the kind of event (e.g. `cheating`, `star_crossed_lovers`, etc.)
- *other-information* is the other information that appears in that kind of event: definitely the characters participating in it, but for some of them, there are things like the monster types or clubs the students belong to
- *condition* is the information to determine whether this plot point makes sense given the story world

Here's an example. For the breakup plot point, it only makes sense for it to happen if the characters are already dating. So the rule for that would look like:

```
PlotPoint [breakup ?a ?b]: [Dating ?a ?b]
```

This says that `[breakup ?a ?b]` is a valid plot point provided `[Dating ?a ?b]`. In other words, breaking up is a valid plot point when the characters are currently dating.

Add methods at the end of `Twitter.step` for each of the types of plot points and make sure they only generate in plausible circumstances.

Testing your plot-point generator

You can test your generator in a few different ways:

- Running: `PlotPoint ?`
  That is, typing `PlotPoint ?` in the green box, and hitting return, will generate a completely random plot point and print it in tuple form.
- Running: `PlotPoint [`*type args …*`]`
  Will force it to try to generate a plot point of the specified type. For example:
  - `PlotPoint [cheating ?a ?b]`
    Will find an `?a` that is cheating on `?b`
  - `PlotPoint [cheating cameron hailey]`
    Will test whether `[cheating cameron hailey]` is a valid plot point. If it doesn't print anything, it's valid. If it prints Call Failed, then that's not a valid plot point. This is a

valid plot point in the version of the code we hand out because Cameron is actually cheating on Hailey. But it might not be true in your story world.

- `PlotPoint [cheating tiana jada]`
  Tests if that's a valid plot point in your story world. It isn't in the version of the code we hand out because they aren't even dating in that story world. So in that code, this would print "Call Failed".

Part three: generating text

Now write methods for `PrintPlotPoint`. These should take the form of:

> `PrintPlotPoint [`*eventType other-information …*`]:` *text-to-print …*

Where, *eventType* and *other-information* are as above, and *text-to-print* is what to print to describe this kind of plot point. The text to print will need to refer to the variables in other-information to fill in blanks. For example, we can write something like this:

```
PrintPlotPoint [breakup ?a ?b]: I'm so bummed that ?a and ?b broke up.
?a seemed so great!
```

Note that these are supposed to be tweets printed by a fan of the series, so they should not only say what happened (e.g. `?a` and `?b` broke up), but also what the fan thinks about it ("I'm so bummed", "`?a` seemed so great").

Write at least one `PrintPlotPoint` method for each event type, so that the system will be able to print any plot points it generates. You are welcome, but not required, to write multiple methods if you want to have more variation in how the system prints a given plot point. You can test PrintPlotPoint by typing a call into the green box. For example, typing:

```
PrintPlotPoint [breakup jayden cameron]
```

Will run whatever method you specified for printing a breakup. If you used the method above, it would print:

```
I'm so bummed that Jayden and Cameron broke up.  Jayden seemed so
great!
```

When you have print methods for all your plot points, you can generate random tweets by typing Alt-T/Option-T. However, if you don't have print methods for all the plot points, it might generate a plot point it can't print in which case it will say that the call to `PrintPlotPoint` failed.

Pronoun generation

As discussed above, if a tweet uses a character's name twice, it will generate pronouns to refer to them when appropriate. By default, it uses they as everyone's pronoun, but you can specify pronouns for specific character using `PreferredPronoun` (see above). That's purely up to you, though.

What you should pay attention to is whether it's generating pronouns in the right case. If your code to print says something like:

```
I love ?who.  ?who is just so great.
```

This will generate something like:

```
I like Jayden.  They are so great.
```

Because the second use of ?who is the subject of a sentence, this is fine. We want it to say they rather than them. But if our code says:

```
I love ?who.  I could just hug ?who.
```

Then we get something a little odd:

```
I like Jayden.  I could just hug they.
```

Here we want to use them rather than they because the second use of ?who is the object of the verb rather than the subject. This is easy to fix. Just change ?x to ?x/Obj:

```
I love ?who.  I could just hug ?who/Obj.
```

This will produce the correct text:

```
I like Jayden.  I could just hug them.
```

When you see your code is generating pronouns in the wrong case, just change it as follows:

- ?x
  Prints they/she/he
- ?x/Obj
  Prints them/her/him
- ?x/Possessive
  Prints their/her/his

Verb conjugation

Hopefully this won't affect you for this assignment. I didn't have to use it for my reference solution, but you may want to generate more ambitious text than I did. As discussed in the reading, you sometimes want to generate different forms of verbs, depending on how the subject is in third person plural or not. We've provided you with the [Is] task from the reading, which prints "is" or "are", depending on what's appropriate, as well as [Has], which prints "has" or "have" as appropriate. For example:

```
?a [Is] going out with ?b
```

Will generate "bill is going out with jenny" or "he is going out with jenny", but "they are going out with jenny" when ?who is realized with the pronoun they.

For regular verbs, you can add [s] at the end of the verb and it will add an -s or an -es, as appropriate. For example:

```
TestCongation ?who: I like ?who.  ?who read[s] books.
```

Will generate:

```
I like Jayden.  They read books.
```

If Jayden's preferred pronoun is they, but:

```
I like Jayden.  He reads books.
```

If Jayden's preferred pronoun is he.

The magic `[s]` task only works for regular verbs. If you want to handle a different verb, then grab the code for Is from `Mention.step` and just change it for your verb. Or just post to Discord and we'll walk you through it.

Part four: Invent some more plot points

Now write `PlotPoint` and `PrintPlotPoint` methods for two more kinds of plot points. They can be anything of your choosing, although they need to depend on the story world. Feel free to add new information to the story world if you like.

Some ideas:

- Two characters have a heart-to-heart discussion while at their club. This would obviously only make sense when they're in the same club.
- Two characters become friends after a fight. For that, there'd need to be a reason for them to have the fight. And they probably shouldn't be dating, since people who are dating should probably already be friends.
- A werewolf character does something on the full moon
- A vampire character bites a human character

Part five: showing off

At this point, you should be able to hit Alt-T/Option-T over and over again and generate lots of tweets. Do this until you find one or more that you like and post them to the #MonsterHighlights channel on Discord.

Turning it in

Save all your files, reload (control-R) to make sure everything really does load. Assuming everything seems okay, make a zip file of your Monster Highlights directory and upload it to canvas.

Grading

This assignment will be machine graded by running `PlotPoint` to verify that particular plot points that should exist can be generated by your code and that others that shouldn't aren't generated by your code. Since you're able to change your story world by adding and removing characters and romances, we'll remove your characters and romance information and replace it with our own, just for testing purposes.

Since we didn't specify specific text to generate, we will not check that your code generates specific text. We will just test that for each of the plot points specified in part 2, it is able to generate some text without failing or producing an error.