```
    # method that returns the sleep state of the wolf
    def show_sleep_state(self):
        if self.is_sleeping == False:
            return self.name + " is awake"
        else:
            return self.name + " is sleeping"

def main():
    # initialising a wolf object and printing the initial sleep
    # state which is awake
    silver_tooth = Wolf("Silver Tooth", 6)
    print(silver_tooth.show_sleep_state())

    # changing sleep state to sleeping using dot notation and then
printing that state
    silver_tooth.is_sleeping = True
    print(silver_tooth.show_sleep_state())

# running main method
main()
```

Instructions

First, read example.py, open it using VS Code.

Read through **example.py** carefully and then run the code and examine the output. When you feel confident that you understand the concept, you can move on to completing the compulsory task below.

Compulsory Task 1

In this task, we're going to be creating an **Email Simulator** using OOP. Follow the instruction and fill in the rest of logic to fulfil the below program requirements.

- Open the file called **email.py**.
- Create an **Email class** and initialise a constructor that takes in three arguments:
    - `email_address` - the email address of the sender
    - `subject_line` - the subject line of the email.
    - `email_content` - the contents of the email.
- The email class should contain the following class **variable** and default value:
    - `has_been_read` - initialised to False.
- The Email class should also contain the following class **method** to edit the values of the email objects:
    - `mark_as_read` which should change `has_been_read` to True.
- Initialise an empty `Inbox` **list** to store, and access, the email objects. **Note**: you can have a list of objects.
- Create the following **functions** to add functionality to your email simulator:
    - `populate_inbox()` - a function which creates an email object with the email address, subject line and contents, and stores it in the *Inbox* list.

      Note: At program start-up, this function should be used to populate your Inbox with three sample email objects for further use in your program. This function does not need to be included as a menu option for the user.
    - `list_emails()` - a function that loops through the Inbox and prints the email's *subject_line*, along with a corresponding number. For example, if there are three emails in the Inbox:

      0 Welcome to HyperionDev!
      1 Great work on the bootcamp!
      2 Your excellent marks!

This function can be used to list the messages when the user chooses to read, mark as spam, and delete an email.

Tip: Use the `enumerate()` function for this function.

- `read_email()` - a function that displays a selected email, together with the *email_address*, *subject_line,* and *email_contents,* and then sets its `has_been_read` instance variable to True.

  For this, allow the user to input an index i.e. `read_email(i)` prints the email stored at position `i` in the list. Following the example above, an index of 0 will print the email with the subject line "Welcome to HyperionDev!".

- Your task is to build out the Class, Methods, Lists, and functions to get everything working! Fill in the rest of the logic for what should happen when the user chooses to:

```
1. Read an email
2. View unread emails
3. Quit application
```

Note: menu option 2 does not require a function. Access the corresponding class variable to retrieve the *subject_line* only.

- Keep the readability of print outputs in mind and take initiative to show the user what is being viewed and what has been executed.

  For example: `print(f"\nEmail from {email.email_address} marked as read.\n")`

Completed the task(s)?

Ask an expert to review your work!

Review work