Commenting Your Code

by Sophia

WHAT'S COVERED

In this lesson, we will learn about commenting our code. Specifically, this lesson covers:

1. Usefulness of Comments

Although we've previously placed comments within our algorithm, they may no longer be fully accurate, as our code could have changed during the process of coding and debugging. In addition, those comments may not be specific to our final program, so it's a good idea to revisit the commenting process. The reason they may not be specific is that we converted from the pseudocode which was defined line by line rather than explaining each section. Programming is rarely done entirely on our own. We want to ensure that others know what is being done in the code. Even if this code is only meant for us, we also want to make it easy to remember what the code is doing months or years later. Rather than trying to figure out what the code is doing, we can look at the comments and quickly remember. Let's go back to our demonstration program, modify the existing comments, and add additional comments.

2. Modifying Our Program Comments

For most of our program, we have very detailed comments. For example, in the `Die` and `Player` classes of those respective modules, we have many of the comments in place for all the methods for each, so that we're able to know exactly what they do.

Here is the `Die` class:

EXAMPLE

```
#Importing the randint module
from random import randint

#The class Die implements a six-sided die
```

```
class Die:

    #The __init__ method is used to create a new instance of die with a default value of 1
    def __init__(self):
        self.value = 1

    #The roll method is used to set the value to a random number between 1 and 6
    def roll(self):
        self.value = randint(1, 6)

    #The getValue method returns the top face value of the die
    def getValue(self):
        return self.value

    #The __str__ method returns the string representation of the value of the die
    def __str__(self):
        return str(self.getValue())
```

For the `Player` class, we nearly have the same level of detail. Two methods within the class could use more explanation:

EXAMPLE

```
    def isWinner(self):
        """Returns True if player has won."""
        return self.winner

    def isLoser(self):
        """Returns True if player has lost."""
        return self.loser
```

It's a good idea to explain further what these methods do, even if it's a short snippet:

```
    #Returns True if the player has won
    def isWinner(self):
        """Returns True if player has won."""
        return self.winner

    #Returns True if the player has lost
    def isLoser(self):
        """Returns True if player has lost."""
        return self.loser
```

Having a short snippet to explain the method is helpful. It's also a good idea to explain the detailed code. Remember that we went back to make some changes to our main program. However, we didn't go back to add the necessary comments to explain what the code is doing. We did have some comments in place but not for the logic itself:

```
#The main function as the point of entry
def main():
    #Play one game
```

```
    print("Running one sample game in full:")
    playOneGame()
    number = 0
    #Play multiple games based on the entry by the user
    while number <= 0:
      user_input = input("How many games would you want to have tested: ")

      try:
        number = int(user_input)
        if number <=0:
          print("Please enter in a positive number.")
          number = 0
        else:
          playMultipleGames(number)
      except ValueError:
        print("Please enter in a number for the number of games.")
```

We should include some detailed comments to explain the `while` loop and the `try and except` statement, including what they are doing. It does not have to be extremely detailed. However, having a general explanation will help:

```
#The main function as the point of entry
def main():
    #Play one game
    print("Running one sample game in full:")
    playOneGame()
    number = 0
    #Play multiple games based on the entry by the user

    #Loop while a valid number greater than 0 is entered
    while number <= 0:
      user_input = input("How many games would you want to have tested: ")

      #Check if a valid number is entered
      try:
        number = int(user_input)

        #Check if the number entered is > 0
        if number <=0:
          print("Please enter in a positive number.")
          number = 0
        else:
          playMultipleGames(number)
      except ValueError:
        print("Please enter in a number for the number of games.")
```

That should be clearer.

TRY IT

Directions: Try adding additional comments to all the modules in the demonstration program. When you are done, review the Example Python Journal Submission document to see what was included for comments and see how close you were. Did you add more or fewer comments?

3. Adding Our Fifth Journal Entry

THINK ABOUT IT

The commenting in your code should be something that has been done up to this point, but it's quite possible that you may not have included all of the key details when it comes to making your code easier to follow.

Bad Example of Journal Entry for Part 5

A bad entry for the journal and for everyday programming would be to not have any comments included. But even having some comments, if they are not easily understood or are very limited, can have negative results if someone else is trying to understand the program's logic. And yes, this can include the programmer of the program as well. Maybe it has been a while since they have been "in" the code. With enough well-placed and thorough comments, they should be able to get back "up to speed" rather quickly.

Here is a bad example of journal entry for Part 5.

```
#The main function as the point of entry
def main():
    print("Running one sample game in full:")
    number = 0
    while number <= 0:
      user_input = input("How many games would you want to have tested: ")
      try:
        number = int(user_input)

        #Check if the number entered is > 0
        if number <=0:
          print("Please enter in a positive number.")
          number = 0
        else:
          playMultipleGames(number)
      except ValueError:
        print("Please enter in a number for the number of games.")
```

A better entry for Part 5 would have comments with more details on what the logic is doing. Although we don't have to comment on every single line, we should at least comment on each section of code to describe what it is doing. Let's see what a good entry would look like:

Good Example of Journal Entry for Part 5

EXAMPLE

```
#The main function as the point of entry
def main():
```

```
    #Play one game
    print("Running one sample game in full:")
    playOneGame()
    number = 0
    #Play multiple games based on the entry by the user

    #Loop while a valid number greater than 0 is entered
    while number <= 0:
      user_input = input("How many games would you want to have tested: ")

      #Check if a valid number is entered
      try:
        number = int(user_input)

        #Check if the number entered is > 0
        if number <=0:
          print("Please enter in a positive number.")
          number = 0
        else:
          playMultipleGames(number)
      except ValueError:
        print("Please enter in a number for the number of games.")
```

Remember, this was just an example of the main.py file. We still needed to go back into our `die>` and `<player>` modules to make sure they had sufficient commenting as well.

If we preview the Example Python Journal Submission document, we will see **ALL** components of our program with good comments added as the entry to Part 5. With this example, each section of code is commented on to describe what it is doing. As programs have more lines of code, we will want to ensure that the comments also increase to clearly describe the additional functionality that comes with more code.

---

3. Guided Brainstorming

When it comes to comments, we can never add too much but we can add too little. In looking at the code now, are we still unsure about some aspects? If so, that's a great place to add comments. In addition, think about explaining the program to someone who may not read code. Or perhaps we may be coming back to the program later. If we're going from the pseudocode to translate it to Python, it'll be easier to include the comments. However, as we've seen between the start and finished code, there's a lot more functionality, code, and comments as we march towards the finished program. It's always easier to comment as we go rather than leave them all until the end.

Let's go back to our Drink Order program and see. Notice in this code, we currently don't have any comments. For anyone that may not know what the program is meant to do, it can be hard to read through the code.

```
drinkDetails=""
drink = input('What type of drink would you like to order?\nWater\nCoffee\nTea\nEnter your choice: ')
if drink == "Water":
    drinkDetails=drink
```

```
    temperature = input("Would you like your water? Hot or Cold: ")
    if temperature == "Hot":
        drinkDetails += ", " + temperature
    elif temperature == "Cold":
        drinkDetails += ", " + temperature
        ice = input("Would you like ice? Yes or No: ")
        if ice == "Yes":
            drinkDetails += ", Ice"
    else:
        drinkDetails += ", unknown temperature entered."
elif drink == "Coffee":
    drinkDetails=drink
    decaf = input("Would you like decaf? Yes or No: ")
    if decaf == "Yes":
        drinkDetails += ", Decaf"
    milkCream = input("Would you like Milk, Cream or None: ")
    if milkCream == "Milk":
        drinkDetails += ", Milk"
    elif milkCream == "Cream":
        drinkDetails += ", Cream"
    sugar = input("Would you like sugar? Yes or No: ")
    if sugar == "Yes":
        drinkDetails += ", Sugar"
elif drink == "Tea":
    drinkDetails=drink
    teaType = input("What type of tea would you like? Black or Green: ")
    if teaType == "Black":
        drinkDetails += ", " + teaType
    elif teaType == "Green":
        drinkDetails += ", " + teaType
else:
    print("Sorry, we did not have that drink available for you.")
print("Your drink selection: ",drinkDetails)
```

A proper way to comment on this code would look like the following, as we've commented on each section of code and for each set of conditional statements:

```
#Used to store the results from the drink
drinkDetails=""

#Ask user for the input of the drink
drink = input('What type of drink would you like to order?\nWater\nCoffee\nTea\nEnter your choice: ')

#Check if Water was entered
if drink == "Water":
    drinkDetails=drink
    temperature = input("Would you like your water? Hot or Cold: ")
```

```
    #Check if the temperature is hot or cold
    if temperature == "Hot":
        drinkDetails += ", " + temperature
    elif temperature == "Cold":
        drinkDetails += ", " + temperature

        #Check if ice should be added
        ice = input("Would you like ice? Yes or No: ")
        if ice == "Yes":
            drinkDetails += ", Ice"
    else:
        drinkDetails += ", unknown temperature entered."

#Check if Coffee was entered
elif drink == "Coffee":
    drinkDetails=drink
    #Check if decaf or not
    decaf = input("Would you like decaf? Yes or No: ")
    if decaf == "Yes":
        drinkDetails += ", Decaf"

    #Check if Milk, Cream or None was entered
    milkCream = input("Would you like Milk, Cream or None: ")
    if milkCream == "Milk":
        drinkDetails += ", Milk"
    elif milkCream == "Cream":
        drinkDetails += ", Cream"

    #Check if sugar should be added
    sugar = input("Would you like sugar? Yes or No: ")
    if sugar == "Yes":
        drinkDetails += ", Sugar"
#Check if Tea was entered
elif drink == "Tea":
    drinkDetails=drink
    teaType = input("What type of tea would you like? Black or Green: ")
    #Check if Black or Green tea was entered
    if teaType == "Black":
        drinkDetails += ", " + teaType
    elif teaType == "Green":
        drinkDetails += ", " + teaType
#If no valid drink was entered
else:
    print("Sorry, we did not have that drink available for you.")\
```

```
#Output the drink details
print("Your drink selection: ",drinkDetails)
```

A good habit is to include comments for the following at the very least:

- Classes
- Functions
- Methods
- Loops
- Conditional statements
- Try/except statements
- Complex code

TRY IT

Directions: Now it's time for you to go back and add comments if you haven't already done so. Remember that each segment should be commented on to explain what the code is meant to do for a non-programmer. A good habit is to comment on classes, functions, methods, loops, conditional statements, `try and except` statements or any complex code segments. Review the example of a good entry for Part 5 in the Example Python Journal Submission document and add your entry for Part 5 to your Python Journal. Remember, this should include all parts of your program if it contains multiple (reusable) components.

SUMMARY

In this lesson, we discussed again how **useful comments** are when added to our code. If someone is trying to figure out what a program does and the comments are either missing or very limited, this can make it very difficult to understand the program or the programmer's logic. This can also apply to the programmer, too, if it has been some time since they were in the program. We **modified and added comments** in the demonstration program to add more clarity on how the program is working. We then had an opportunity to see a good example for our **fifth journal entry**. Finally, in the **Guided Brainstorming** section, we saw some more good examples of commenting with the Drink Order program.

Best of luck in your learning!