

**ANL252**

**Python for Data Analytics**

# **Tutor-Marked Assignment**

**July 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Tea Ich Ngy** | **Y2210678** |
| Click to enter NAME. | Click to enter PI NO. |
| Click to enter NAME. | Click to enter PI NO. |
| Click to enter NAME. | Click to enter PI NO. |

# **Tutorial Group: ­­­­­­­­­­­­ T 03**

**Instructor’s Name: Mr. Munish Kumar**

**Submission Date: 15/09/2023**

**Question 1 (a):**

According to an article by Codequiry, it defines code plagiarism as “copying or reproducing source code without written permission from the original creator.” (*Combatting Code Plagiarism in Computer Science | Codequiry*, n.d.). Code plagiarism happens because the source codes are easily available on the internet like Github and Stack Overflow, making it very easy to take them without correctly asking permission from original creator(s). Not only that, Generative AI software such as ChatGPT, and Bard (by Google) are free to use and can easily provide students with coding solutions without students doing it themselves.

Turnitin, a plagiarism solution platform had suggested some solutions as follows.

1. Making clear the rules on academic integrity, including defining collaboration versus collusion (Lee, 2023). This is because many coding projects are collaboration in nature and thus by setting out clear rules, students can understand how to avoid code plagiarism.
2. Emphasize policies on using outside code. For example, having a restricted source list and ask students to cite these sources in their work (Lee, 2023).
3. Encourage students to brainstorm and come up with original and unique code (Lee, 2023). Instructors can model students coding assignments that encourage students to come up with their own code.

(Words: 202)

**Question 1 (b):**

Number guessing game python code:

|  |  |
| --- | --- |
| 𝟷  𝟸  𝟹  𝟺  𝟻  𝟼  𝟽  𝟾  𝟿  𝟷0  𝟷𝟷  𝟷𝟸  𝟷𝟹  𝟷𝟺  𝟷𝟻  𝟷𝟼  𝟷𝟽  𝟷𝟾  𝟷𝟿  𝟸0  𝟸𝟷  𝟸𝟸  𝟸𝟹  𝟸𝟺  𝟸𝟻  𝟸𝟼  𝟸𝟽  𝟸𝟾 | 𝚒𝚖𝚙𝚘𝚛𝚝 𝚛𝚊𝚗𝚍𝚘𝚖  𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎 = 𝚒𝚗𝚙𝚞𝚝("𝚃𝚢𝚙𝚎 𝚊 𝚗𝚞𝚖𝚋𝚎𝚛: ")  𝚒𝚏 𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎.𝚒𝚜𝚍𝚒𝚐𝚒𝚝():  𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎 = 𝚒𝚗𝚝(𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎)  𝚒𝚏 𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎 <= 0:  𝚙𝚛𝚒𝚗𝚝("𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚢𝚙𝚎 𝚊 𝚗𝚞𝚖𝚋𝚎𝚛 𝚕𝚊𝚛𝚐𝚎𝚛 𝚝𝚑𝚊𝚗 0 𝚗𝚎𝚡𝚝 𝚝𝚒𝚖𝚎. ")  𝚚𝚞𝚒𝚝()  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚢𝚙𝚎 𝚊 𝚗𝚞𝚖𝚋𝚎𝚛 𝚗𝚎𝚡𝚝 𝚝𝚒𝚖𝚎. ")  𝚚𝚞𝚒𝚝()  𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛 = 𝚛𝚊𝚗𝚍𝚘𝚖.𝚛𝚊𝚗𝚍𝚒𝚗𝚝(0, 𝚝𝚘𝚙\_𝚘𝚏\_𝚛𝚊𝚗𝚐𝚎)  𝚐𝚞𝚎𝚜𝚜𝚎𝚜 = 0  𝚠𝚑𝚒𝚕𝚎 𝚃𝚛𝚞𝚎:  𝚐𝚞𝚎𝚜𝚜𝚎𝚜 += 𝟷  𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 = 𝚒𝚗𝚙𝚞𝚝("𝙼𝚊𝚔𝚎 𝚊 𝚐𝚞𝚎𝚜𝚜: ")  𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜.𝚒𝚜𝚍𝚒𝚐𝚒𝚝():  𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 = 𝚒𝚗𝚝(𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜)  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚢𝚙𝚎 𝚊 𝚗𝚞𝚖𝚋𝚎𝚛 𝚗𝚎𝚡𝚝 𝚝𝚒𝚖𝚎. ")  𝚌𝚘𝚗𝚝𝚒𝚗𝚞𝚎  𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 == 𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚐𝚘𝚝 𝚒𝚝 𝚛𝚒𝚐𝚑𝚝! ")  𝚋𝚛𝚎𝚊𝚔  𝚎𝚕𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 > 𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚠𝚎𝚛𝚎 𝚊𝚋𝚘𝚟𝚎 𝚝𝚑𝚎 𝚗𝚞𝚖𝚋𝚎𝚛!")  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚠𝚎𝚛𝚎 𝚋𝚎𝚕𝚘𝚠 𝚝𝚑𝚎 𝚗𝚞𝚖𝚋𝚎𝚛!")  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚐𝚘𝚝 𝚒𝚝 𝚒𝚗", 𝚐𝚞𝚎𝚜𝚜𝚎𝚜, "𝚐𝚞𝚎𝚜𝚜𝚎𝚜. ") |

The code first imports the random module in order to generate a random number. Line 2 to 10 of the code instruct Python for player to input a number. Line 3, 4 and 8 – 10 ensure that the player input a numeric number and tell the player if they do not input as instructed. Line 5 – 7 ensure that the player will not input any number less than 0, as the randomly generated figure is a positive figure. Line 11 is to create a variable for a random number and line 12 is used to keep the number of guesses. Line 13 to 27 execute the main code for guessing the number using while loop. Line 14 will add 1 to line 12 variable every time a player makes a guess. Line 16 to 20 ensure that the player input a numeric figure. Line 21 to 23 will execute when the player guessed the random number. Line 24 to 26 is a loop to show the player if their current guess is above or below the random number. Lastly, line 28 will print the statement on how many guesses the player had after they have guessed the random number.

(Words: 199)

**Question 1 (c):**

Rewritten number guessing code:

|  |  |
| --- | --- |
| 𝟷  𝟸  𝟹  𝟺  𝟻  𝟼  𝟽  𝟾  𝟿  𝟷0  𝟷𝟷  𝟷𝟸  𝟷𝟹  𝟷𝟺  𝟷𝟻  𝟷𝟼  𝟷𝟽  𝟷𝟾  𝟷𝟿  𝟸0  𝟸𝟷 | 𝚒𝚖𝚙𝚘𝚛𝚝 𝚛𝚊𝚗𝚍𝚘𝚖  𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛 = 𝚛𝚊𝚗𝚍𝚘𝚖.𝚛𝚊𝚗𝚍𝚒𝚗𝚝(0, 𝟸0)  𝚐𝚞𝚎𝚜𝚜𝚎𝚜 = 0  𝚠𝚑𝚒𝚕𝚎 𝚃𝚛𝚞𝚎:  𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 = 𝚒𝚗𝚙𝚞𝚝("𝙼𝚊𝚔𝚎 𝚊 𝚐𝚞𝚎𝚜𝚜 𝚋𝚎𝚝𝚠𝚎𝚎𝚗 0 𝚝𝚘 𝟸0: ")  𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜.𝚒𝚜𝚍𝚒𝚐𝚒𝚝():  𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 = 𝚒𝚗𝚝(𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜)  𝚒𝚏 0 <= 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 <= 𝟸0:  𝚐𝚞𝚎𝚜𝚜𝚎𝚜 += 𝟷  𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 == 𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚐𝚘𝚝 𝚒𝚝 𝚛𝚒𝚐𝚑𝚝! ")  𝚋𝚛𝚎𝚊𝚔  𝚎𝚕𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚐𝚞𝚎𝚜𝚜 > 𝚛𝚊𝚗𝚍𝚘𝚖\_𝚗𝚞𝚖𝚋𝚎𝚛:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚠𝚎𝚛𝚎 𝚊𝚋𝚘𝚟𝚎 𝚝𝚑𝚎 𝚗𝚞𝚖𝚋𝚎𝚛! ")  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚠𝚎𝚛𝚎 𝚋𝚎𝚕𝚘𝚠 𝚝𝚑𝚎 𝚗𝚞𝚖𝚋𝚎𝚛 !")  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝙸𝚗𝚟𝚊𝚕𝚒𝚍. 𝙸𝚗𝚙𝚞𝚝 𝚒𝚜 𝚗𝚘𝚝 𝚋𝚎𝚝𝚠𝚎𝚎𝚗 0 - 𝟸0. 𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚛𝚢 𝚊𝚐𝚊𝚒𝚗. ")  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝("𝙸𝚗𝚟𝚊𝚕𝚒𝚍. 𝙸𝚗𝚙𝚞𝚝 𝚒𝚜 𝚗𝚘𝚝 𝚗𝚞𝚖𝚎𝚛𝚒𝚌. 𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚛𝚢 𝚊𝚐𝚊𝚒𝚗. ")  𝚙𝚛𝚒𝚗𝚝("𝚈𝚘𝚞 𝚐𝚘𝚝 𝚒𝚝 𝚒𝚗", 𝚐𝚞𝚎𝚜𝚜𝚎𝚜,"𝚐𝚞𝚎𝚜𝚜. ") |

The code is rewritten to be more condensed and shortened the code from 28 lines to 22 lines of code. The following are the changes:

1. Instead of asking player to input a number and randomly generate another number, the rewritten code in line 2 generate a random number from 0 to 20 right away. This is different from the original code as this range can be amended but line 5, line 8, line 18 must be amended too to reflect the change.
2. The rewritten code also incorporates another potential error by player if they key in a letter instead of a digit. Lines 6 – 7 uses “**.isdigit():**” in the “**if**” condition to ensure player input a digit, “**else**” it will print in invalid statement in lines 19 – 20.
3. The rewritten code also ensure that the player keyed in the number within the given range. In line 5, the code instructs the player to input a number between 0 to 20. The lines 8 – 9 uses **0 <= user\_guess <= 20** “**if**” condition to ensure that the player’s input is within the given range, “**else**” it will print an invalid statement in lines 17 – 18.
4. Like the original code, the rewritten code uses variable “guesses” to track and add the number of guesses to display to the player once they guessed the randomly generated number from the given range. In line 9, the code is used to add 1 to variable “**guesses**” after every wrong guess. It is put right after lines 10 – 12, the lines of code for the “**if**” condition when the player guessed correctly, to ensure that the variable guesses is being added correctly every time before the player finally guessed correctly and break the while loop in line 12.

(Words: 296)

**Question 2:**

Improved code:

|  |  |
| --- | --- |
| 𝟷  𝟸  𝟹  𝟺  𝟻  𝟼  𝟽  𝟾  𝟿  𝟷0  𝟷𝟷  𝟷𝟸  𝟷𝟹  𝟷𝟺  𝟷𝟻  𝟷𝟼  𝟷𝟽  𝟷𝟾  𝟷𝟿  𝟸0  𝟸𝟷 | 𝚙𝚛𝚘𝚍𝚞𝚌𝚝𝚜 = ['𝚕𝚊𝚙𝚝𝚘𝚙', '𝚖𝚘𝚞𝚜𝚎', '𝚠𝚎𝚋𝚌𝚊𝚖𝚎', '𝚔𝚎𝚢𝚋𝚘𝚊𝚛𝚍', '𝚜𝚙𝚎𝚊𝚔𝚎𝚛']  𝚞𝚙𝚍𝚊𝚝𝚎𝚍\_𝚒𝚝𝚎𝚖𝚜 = []  𝚠𝚑𝚒𝚕𝚎 𝚃𝚛𝚞𝚎:  𝚙𝚛𝚒𝚗𝚝(𝚏'𝚆𝚎 𝚑𝚊𝚟𝚎 𝚊 𝚕𝚒𝚜𝚝 𝚘𝚏 𝚙𝚛𝚘𝚍𝚞𝚌𝚝𝚜 𝚑𝚎𝚛𝚎: {𝚙𝚛𝚘𝚍𝚞𝚌𝚝𝚜}.')  𝚋𝚞𝚢𝚒𝚗𝚐\_𝚒𝚗𝚙𝚞𝚝 = 𝚜𝚝𝚛(𝚒𝚗𝚙𝚞𝚝("𝙷𝚎𝚕𝚕𝚘! 𝚆𝚑𝚊𝚝 𝚍𝚘 𝚢𝚘𝚞 𝚠𝚊𝚗𝚝 𝚝𝚘 𝚋𝚞𝚢? "))  𝚒𝚏 𝚋𝚞𝚢𝚒𝚗𝚐\_𝚒𝚗𝚙𝚞𝚝 𝚗𝚘𝚝 𝚒𝚗 𝚙𝚛𝚘𝚍𝚞𝚌𝚝𝚜:  𝚙𝚛𝚒𝚗𝚝("𝚆𝚛𝚘𝚗𝚐 𝚙𝚛𝚘𝚍𝚞𝚌𝚝! 𝙿𝚕𝚎𝚊𝚜𝚎 𝚝𝚛𝚢 𝚊𝚐𝚊𝚒𝚗.")  𝚙𝚛𝚒𝚗𝚝("--------------------------------")  𝚌𝚘𝚗𝚝𝚒𝚗𝚞𝚎  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚌𝚎\_𝚘𝚏\_𝚒𝚝𝚎𝚖 = 𝚒𝚗𝚙𝚞𝚝("𝙷𝚘𝚠 𝚖𝚞𝚌𝚑 𝚒𝚜 𝚒𝚝 (𝚒𝚗 𝚂𝙶𝙳)? ")  𝚙𝚛𝚒𝚗𝚝("--------------------------------")  𝚎𝚗𝚝𝚎𝚛𝚎𝚍\_𝚒𝚗𝚙𝚞𝚝 = [𝚋𝚞𝚢𝚒𝚗𝚐\_𝚒𝚗𝚙𝚞𝚝, 𝚙𝚛𝚒𝚌𝚎\_𝚘𝚏\_𝚒𝚝𝚎𝚖]  𝚞𝚙𝚍𝚊𝚝𝚎𝚍\_𝚒𝚝𝚎𝚖𝚜.𝚊𝚙𝚙𝚎𝚗𝚍(𝚎𝚗𝚝𝚎𝚛𝚎𝚍\_𝚒𝚗𝚙𝚞𝚝)  𝚞𝚜𝚎𝚛\_𝚒𝚗𝚙𝚞𝚝 = 𝚒𝚗𝚙𝚞𝚝("𝚆𝚘𝚞𝚕𝚍 𝚢𝚘𝚞 𝚕𝚒𝚔𝚎 𝚝𝚘 𝚌𝚘𝚗𝚝𝚒𝚗𝚞𝚎? (𝚢𝚎𝚜/𝚗𝚘) ").𝚕𝚘𝚠𝚎𝚛()  𝚙𝚛𝚒𝚗𝚝("--------------------------------")  𝚒𝚏 𝚞𝚜𝚎𝚛\_𝚒𝚗𝚙𝚞𝚝 == "𝚢𝚎𝚜":  𝚌𝚘𝚗𝚝𝚒𝚗𝚞𝚎  𝚎𝚕𝚜𝚎:  𝚙𝚛𝚒𝚗𝚝(𝚏'𝚃𝚑𝚒𝚜 𝚒𝚜 𝚘𝚞𝚛 𝚞𝚙𝚍𝚊𝚝𝚎𝚍 𝚜𝚑𝚘𝚙𝚙𝚒𝚗𝚐 𝚕𝚒𝚜𝚝: {𝚞𝚙𝚍𝚊𝚝𝚎𝚍\_𝚒𝚝𝚎𝚖𝚜}')  𝚋𝚛𝚎𝚊𝚔 |

The following are improvements:

1. To enhance readability for the user, print lines using dashes (“------“) were added so that it is easier for the user to read. These print statements are used in line 8 after the invalid statement telling the user they have keyed in the wrong product, line 12 after asking how much is the item and line 16 after the line asking an input statement asking if the user want to continue.
2. Next is the overall quality, which is to use “**while true**” loop. This eliminate the use to using variable “**query**” that was used in the original code in line 2 and 6, and also gives the coder more freedom to add more conditions if necessary, instead of limited to the while loop being only “**yes**”.
3. Lastly, for reliability, in line 15, “**.lower()**” were used for the user input to automatically convert the inputs to lower cases. This ensure that the line 17 will always execute without error as the code will only execute when the user input is exactly “**yes**” in lower case.

(Words: 182)

**References:**

*Combatting code plagiarism in computer science | Codequiry*. (n.d.). <https://codequiry.com/code-plagiarism>

Lee, C. (2023, September 12). What is Programming Plagiarism? Why Is It on the Rise? *Turnitin*. <https://www.turnitin.com/blog/what-is-programming-plagiarism-why-is-it-on-the-rise>

*Stack Overflow - where developers learn, share, & build careers*. (n.d.). Stack Overflow. <https://stackoverflow.com/>

*GitHub: Let’s build from here*. (n.d.). GitHub. <https://github.com/>

Tech With Tim. (2021, June 10). *5 Mini Python Projects - For Beginners* [Video]. YouTube. <https://www.youtube.com/watch?v=DLn3jOsNRVE>

Wu, K. Y., & Zhu, S. (2023). ANL252 Python for data analytics. Singapore University of Social Sciences.

Shaw, Z. A. (2017). *Learn Python 3 the hard way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code*. Addison-Wesley Professional.