Day - 7

**HangMan & "in"**

Review: or & While Loops, IF /ELSE, Lists, Strings, Range, Modules

**7.1 Algorithm for HANGMAN**

|  |  |
| --- | --- |
| 1. Select a random word 2. Create blank space equal to word length 3. **Loop**: Ask for letter.    1. **If** yes: ask for whole word.       1. If correct: win.       2. Else: show letter in blank space    2. **Else**: step i to hang       1. If step i completed: game over | #Step 1  word\_list = ["ardvark", "baboon", "camel"]1  #T0D0-1 - Randomly choose a word from the word\_list and assign it to a variable called chosen\_word.  #T0D0-2 - Ask the user to guess a letter and assign their answer to a variable called guess. Make guess lowercase.  #T0D0-3 - Check if the letter the user guessed (guess) is one of the leters in the chosen word. |

**import** random

#*Step 1*

word\_list = ["ardvark", "baboon", "camel"]

#*T0D0-1 - Randomly choose a word from the word\_list and assign it to a variable called chosen\_word.*

randWord = random**.choice**(word\_list)

**print**(randWord)

#*T0D0-2 - Ask the user to guess a letter and assign their answer to a variable called guess. Make guess lowercase.*

askLetr = **input**("\n\t\tGuess the letter : ")**.lower**()

#*T0D0-3 - Check if the letter the user guessed (guess) is one of the leters in the chosen word.*

**for** ch **in** randWord :

**if** ch **==** askLetr:

**print**("Yes")

**else**:

**print**("No")

#*python hangman.py*

|  |
| --- |
| #Step 2  import random  word\_list = ["ardvark", "baboon", "camel"]  randWord = random.choice(word\_list)  #Testing code  print(f'Pssst, the solution is { randWord }.')  #T0D0-1: - Create an empty List called display. For each letter in the ***randWord***, add a "***\_***" to 'display. So if the ***randWord*** was "apple", display should be ["\_","\_","\_","\_","\_"] with 5 "\_" representing each letter to guess.  askLetr  = input("Guess a letter: ").lower()  #T0D0-2: - Loop through each position in the ***randWord***; If the letter at that position matches 'guess' then reveal that letter in the display at that position.  e.g. If the user guessed "p" and the chosen word was "apple", then display should be ["\_","p","p","\_","\_"]  #T0D0-3: - Print 'display' and you should see the guessed letter in the correct position and every other letter replace with "\_". |

**import** random

#*Step 1*

word\_list = ["ardvark", "baboon", "camel"]

randWord = random**.choice**(word\_list)

disPlay = []

**for** ch **in** randWord :

    disPlay**.append**("\_")

**print**(disPlay)

#*Testing code*

**print**(f'Pssst, the solution is { randWord }.')

askLetr = **input**("\n\t\tGuess a letter : ")**.lower**()

**for** i **in** **range**(0, len(randWord)) :

**if** randWord[i] **==** askLetr:

        disPlay[i] = randWord[i]

**else**:

        disPlay[i] = disPlay[i]

**print**(disPlay)

|  |
| --- |
| #Step 3  #T0D0-3.1: - Use a while loop to let the user guess again. The loop should only stop once the user has guessed all the letters in the ***randWord*** and 'display' has no more blanks. Then you can tell the user they've won. |

|  |  |
| --- | --- |
| My Version | Angela's Version |
| **while** disPlay**.count**("\_") **>** 0 :      askLetr = **input**("\n\t\tGuess a letter : ")**.lower**()  **for** i **in** **range**(0, len(randWord)) :  **if** randWord[i] **==** askLetr:              disPlay[i] = randWord[i]  **else**:              disPlay[i] = disPlay[i]  **print**(disPlay) | end\_game = **False**  **while** **not** end\_game :      askLetr = **input**("\n\t\tGuess a letter : ")**.lower**()  **for** i **in** **range**(0, len(randWord)) :  **if** randWord[i] **==** askLetr:              disPlay[i] = randWord[i]  **else**:              disPlay[i] = disPlay[i]  **print**(disPlay)  **if** **not** ("\_" **in** disPlay):          end\_game = **True** |

|  |
| --- |
| #Step 4  #T0D0-1: - Create a variable called 'lives' to keep track of the number of lives left. Set 'lives' to equal 6.  #T0D0-2: - If guess is not a letter in the ***randWord***, then reduce 'lives' by 1. If lives goes down to 0 then the game should stop and it should print "You lose."  #Join all the elements in the list and turn it into a String. print(f"{' '.join(display)}")  #T0D0-3: - print the *ASCII* *art* from *'stages'* that corresponds to the current number of ***'lives'*** the user has remaining. |

end\_game = **False**

lives = 6

**while** **not** end\_game :

     askLetr = **input**("\n\t\tGuess a letter : ")**.lower**()

**for** i **in** **range**(0, len(randWord)) :

**if** randWord[i] **==** askLetr:

             disPlay[i] = randWord[i]

**else**:

            disPlay[i] = disPlay[i]

**if** **not** (askLetr **in** randWord):

        lives = lives - 1

**print**(stages[6 - lives])

**print**(disPlay)

**if** lives **==** 0:

        end\_game = **True**

**print**("\n\t\t\tYou lose !!")

**elif** **not** ("\_" **in** disPlay):

        end\_game = **True**

**print**("\n\t\t\tYou Win !!")

**print**(f"\n\t\t\t{''**.join**(disPlay)}")

|  |
| --- |
| #Step 5 : Modularize and Enhance UI  #T0D0-1: - Update the word list to use the '***word\_list***' from ***hangman\_words.py*** #Delete this line: word\_list = ["ardvark", "baboon", "camel"]  #T0D0-2: - Import the logo from ***hangman\_art.py*** and print it at the start of the game.  #T0D0-3: - Import the stages from ***hangman\_art.py*** and make this error go away. print(stages [lives]) |

**7.2 "in" keyword:**

Definition and Usage: The in keyword has two purposes:

1. The ***in*** keyword is used to ***check*** if a value is present in a ***sequence*** (list, range, string etc.).
2. The ***in*** keyword is also used to ***iterate*** through a sequence in a ***for*** ***loop***:

|  |  |
| --- | --- |
| Example 1: Check if "banana" is present in the list:  fruits = ["apple", "banana", "cherry"]  **if** "banana" **in** fruits:  **print**("yes") | Example 2: Loop through a list and print the items:  fruits = ["apple", "banana", "cherry"]  **for** x **in** fruits:  **print**(x) |

**7.3 join()**

Join all the elements in the ***list*** and turn it into a ***String***.

**print**(f"{' '.**join**(list)}")

**def** **join**(iterable: Iterable[str])

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.**join**(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

fruits = ["apple", "banana", "cherry"]

**print**(f"{''**.join**(fruits)}")  -> "applebananacherry"

**print**(f"{'\_'**.join**(fruits)}")  -> "apple\_banana\_cherry"

Standard Solution

***python main.py***

**import** random

**import** hangman\_words

**import** hangman\_art

**print**(hangman\_art**.**logo)

randWord = random**.choice**(hangman\_words**.**word\_list)

disPlay = []

**for** ch **in** randWord :

    disPlay**.append**("\_")

**print**(disPlay)

#*Testing code*

**print**(f'Pssst, the solution is { randWord }.')

end\_game = **False**

lives = 6

**while** **not** end\_game :

    askChr = **input**("\n\t\tGuess a letter : ")**.lower**()

**for** i **in** **range**(0, len(randWord)) :

**if** randWord[i] **==** askChr:

            disPlay[i] = randWord[i]

**else**:

            disPlay[i] = disPlay[i]

**if** **not** (askChr **in** randWord):

        lives = lives - 1

**print**(hangman\_art**.**stages[lives])

**print**(disPlay)

**if** lives **==** 0:

        end\_game = **True**

**print**("\n\t\t\tYou lose !!")

**elif** **not** ("\_" **in** disPlay):

        end\_game = **True**

**print**("\n\t\t\tYou Win !!")

**print**(f"\n\t\t\t{''**.join**(disPlay)}")

#*python main.py*

***hangman\_art.py***

stages = ['''

  +---+

  |   |

  O   |

 /|\  |

 / \  |

      |

=========

''', '''

  +---+

  |   |

  O   |

 /|\  |

 /    |

      |

=========

''', '''

  +---+

  |   |

  O   |

 /|\  |

      |

      |

=========

''', '''

  +---+

  |   |

  O   |

 /|   |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

  |   |

      |

      |

=========

''', '''

  +---+

  |   |

  O   |

      |

      |

      |

=========

''', '''

  +---+

  |   |

      |

      |

      |

      |

=========

''']

logo = '''

 \_

| |

| |\_\_   \_\_ \_ \_ \_\_   \_\_ \_ \_ \_\_ \_\_\_   \_\_ \_ \_ \_\_

| '\_ \ / \_` | '\_ \ / \_` | '\_ ` \_ \ / \_` | '\_ \

| | | | (\_| | | | | (\_| | | | | | | (\_| | | | |

|\_| |\_|\\_\_,\_|\_| |\_|\\_\_, |\_| |\_| |\_|\\_\_,\_|\_| |\_|

                    \_\_/ |

                   |\_\_\_/    '''

213 words hangman\_words.py

|  |
| --- |
| word\_list = ["abruptly", "absurd", "abyss", "affix", "askew", "avenue", "awkward", "axiom", "azure", "bagpipes", "bandwagon", "banjo",  "bayou", "beekeeper", "bikini", "blitz", "blizzard", "boggle", "bookworm", "boxcar", "boxful", "buckaroo", "buffalo", "buffoon", "buxom",  "buzzard", "buzzing", "buzzwords", "caliph", "cobweb", "cockiness", "croquet", "crypt", "curacao", "cycle", "daiquiri", "dirndl",  "disavow", "dizzying", "duplex", "dwarves", "embezzle", "equip", "espionage", "euouae", "exodus", "faking", "fishhook", "fixable", "fjord", "flapjack", "flopping", "fluffiness", "flyby", "foxglove", "frazzled", "frizzled", "fuchsia", "funny", "gabby", "galaxy", "galvanize", "gazebo", "giaour", "gizmo", "glowworm", "glyph", "gnarly", "gnostic", "gossip", "grogginess", "haiku", "haphazard", "hyphen", "iatrogenic", "icebox", "injury", "ivory", "ivy", "jackpot", "jaundice", "jawbreaker", "jaywalk", "jazziest", "jazzy", "jelly", "jigsaw", "jinx", "jiujitsu", "jockey", "jogging", "joking", "jovial", "joyful", "juicy", "jukebox", "jumbo", "kayak", "kazoo", "keyhole", "khaki", "kilobyte", "kiosk", "kitsch", "kiwifruit", "klutz", "knapsack", "larynx", "lengths", "lucky", "luxury", "lymph", "marquis", "matrix", "megahertz", "microwave", "mnemonic", "mystify", "naphtha", "nightclub", "nowadays", "numbskull", "nymph", "onyx", "ovary", "oxidize", "oxygen", "pajama", "peekaboo", "phlegm", "pixel", "pizazz", "pneumonia", "polka", "pshaw", "psyche", "puppy", "puzzling", "quartz",  "queue", "quips", "quixotic", "quiz", "quizzes", "quorum", "razzmatazz", "rhubarb", "rhythm", "rickshaw", "schnapps", "scratch", "shiv",  "snazzy", "sphinx", "spritz", "squawk", "staff", "strength", "strengths", "stretch", "stronghold", "stymied", "subway", "swivel",  "syndrome", "thriftless", "thumbscrew", "topaz", "transcript", "transgress", "transplant", "triphthong", "twelfth", "twelfths", "unknown",  "unworthy", "unzip", "uptown", "vaporize", "vixen", "vodka", "voodoo", "vortex", "voyeurism", "walkway", "waltz", "wave", "wavy", "waxy", "wellspring", "wheezy", "whiskey", "whizzing", "whomever", "wimpy", "witchcraft", "wizard", "woozy", "wristwatch", "wyvern", "xylophone", "yachtsman", "yippee", "yoked", "youthful", "yummy", "zephyr", "zigzag", "zigzagging", "zilch", "zipper", "zodiac", "zombie"] |

Row Solution

#*============  step 1  ============*

#*T0D0-1.1 - Randomly choose a word from the word\_list and assign it to a variable called chosen\_word.*

#*T0D0-1.2 - Ask the user to guess a letter and assign their answer to a variable called guess. Make guess lowercase.*

#*T0D0-1.3 - Check if the letter the user guessed (guess) is one of the leters in the chosen word.*

#*============  step 2  ============*

#*T0D0-2.1: - Create an empty List called display. For each letter in the randWord, add a "\_" to 'display. So if the randWord was "apple", display should be ["\_","\_","\_","\_","\_"] with 5 "\_" representing each letter to guess.*

#*T0D0-2.2: - Loop through each position in the randWord; If the letter at that position matches 'guess' then reveal that letter in the display at that position.*

            #*e.g. If the user guessed "p" and the chosen word was "apple", then display should be  ["\_","p","p","\_","\_"]*

#*T0D0-2.3: - Print 'display' and you should see the guessed letter in the correct position and every other letter replace with "\_".*

#*============  step 3  ============*

#*T0D0-3.1: - Use a while loop to let the user guess again. The loop should only stop once the user has guessed all the letters in the randWord and 'display' has no more blanks.    Then you can tell the user they've won.*

#*============  step 4  ============*

#*T0D0-4.1: - Create a variable called 'lives' to keep track of the number of lives left. Set 'lives' to equal 6.*

#*T0D0-4.2: - If guess is not a letter in the randWord, then reduce 'lives' by 1. If lives goes down to 0 then the game should stop and it should print "You lose."*

#*Join all the elements in the list and turn it into a String.  print(f"{' '.join(display)}")*

#*T0D0-4.3: - print the ASCII art from 'stages' that corresponds to the current number of 'lives' the user has remaining.*

#*============  step 5  ============*

#*T0D0-5.1: - Update the word list to use the 'word\_list' from hangman\_words.py #Delete this line: word\_list = ["ardvark", "baboon", "camel"]*

#*T0D0-5.2: - Import the logo from hangman\_art.py and print it at the start of the game.*

#*T0D0-5.3: - Import the stages from hangman\_art.py and make this error go away. print(stages [lives])*

**import** random

stages = ['''

  +---+

  |   |

      |

      |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

      |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

  |   |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

 /|   |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

 /|\  |

      |

      |

=========''', '''

  +---+

  |   |

  O   |

 /|\  |

 /    |

      |

=========''', '''

  +---+

  |   |

  O   |

 /|\  |

 / \  |

      |

=========''']

#*Word bank of animals*

word\_list = ('ant ardvark baboon badger bat bear beaver camel cat clam cobra cougar '

         'coyote crow deer dog donkey duck eagle ferret fox frog goat '

         'goose hawk lion lizard llama mole monkey moose mouse mule newt '

         'otter owl panda parrot pigeon python rabbit ram rat raven '

         'rhino salmon seal shark sheep skunk sloth snake spider '

         'stork swan tiger toad trout turkey turtle weasel whale wolf '

         'wombat zebra ')**.split**()

randWord = random**.choice**(word\_list)

disPlay = []

**for** ch **in** randWord :

    disPlay**.append**("\_")

**print**(disPlay)

#*Testing code*

**print**(f'Pssst, the solution is { randWord }.')

end\_game = **False**

lives = 6

**while** **not** end\_game :

    askLetr = **input**("\n\t\tGuess a letter : ")**.lower**()

**for** i **in** **range**(0, len(randWord)) :

**if** randWord[i] **==** askLetr:

            disPlay[i] = randWord[i]

**else**:

            disPlay[i] = disPlay[i]

**if** **not** (askLetr **in** randWord):

        lives = lives - 1

**print**(stages[6 - lives])

**print**(disPlay)

**if** lives **==** 0:

        end\_game = **True**

**print**("\n\t\t\tYou lose !!")

**elif** **not** ("\_" **in** disPlay):

        end\_game = **True**

**print**("\n\t\t\tYou Win !!")

**print**(f"\n\t\t\t{''**.join**(disPlay)}")

#*python hangman.py*

word\_list = ["abruptly", "absurd", "abyss", "affix", "askew", "avenue", "awkward", "axiom", "azure", "bagpipes", "bandwagon", "banjo", "bayou", "beekeeper", "bikini", "blitz", "blizzard", "boggle", "bookworm", "boxcar", "boxful", "buckaroo", "buffalo", "buffoon", "buxom", "buzzard", "buzzing", "buzzwords", "caliph", "cobweb", "cockiness", "croquet", "crypt", "curacao", "cycle", "daiquiri", "dirndl", "disavow", "dizzying", "duplex", "dwarves", "embezzle", "equip", "espionage", "euouae", "exodus", "faking", "fishhook", "fixable", "fjord", "flapjack", "flopping", "fluffiness", "flyby", "foxglove", "frazzled", "frizzled", "fuchsia", "funny", "gabby", "galaxy", "galvanize", "gazebo", "giaour", "gizmo", "glowworm", "glyph", "gnarly", "gnostic", "gossip", "grogginess", "haiku", "haphazard", "hyphen", "iatrogenic", "icebox", "injury", "ivory", "ivy", "jackpot", "jaundice", "jawbreaker", "jaywalk", "jazziest", "jazzy", "jelly", "jigsaw", "jinx", "jiujitsu", "jockey", "jogging", "joking", "jovial", "joyful", "juicy", "jukebox", "jumbo", "kayak", "kazoo", "keyhole", "khaki", "kilobyte", "kiosk", "kitsch", "kiwifruit", "klutz", "knapsack", "larynx", "lengths", "lucky", "luxury", "lymph", "marquis", "matrix", "megahertz", "microwave", "mnemonic", "mystify", "naphtha", "nightclub", "nowadays", "numbskull", "nymph", "onyx", "ovary", "oxidize", "oxygen", "pajama", "peekaboo", "phlegm", "pixel", "pizazz", "pneumonia", "polka", "pshaw", "psyche", "puppy", "puzzling", "quartz", "queue", "quips", "quixotic", "quiz", "quizzes", "quorum", "razzmatazz", "rhubarb", "rhythm", "rickshaw", "schnapps", "scratch", "shiv", "snazzy", "sphinx", "spritz", "squawk", "staff", "strength", "strengths", "stretch", "stronghold", "stymied", "subway", "swivel", "syndrome", "thriftless", "thumbscrew", "topaz", "transcript", "transgress", "transplant", "triphthong", "twelfth", "twelfths", "unknown", "unworthy", "unzip", "uptown", "vaporize", "vixen", "vodka", "voodoo", "vortex", "voyeurism", "walkway", "waltz", "wave", "wavy", "waxy", "wellspring", "wheezy", "whiskey", "whizzing", "whomever", "wimpy", "witchcraft", "wizard", "woozy", "wristwatch", "wyvern", "xylophone", "yachtsman", "yippee", "yoked", "youthful", "yummy", "zephyr", "zigzag", "zigzagging", "zilch", "zipper", "zodiac", "zombie"]

Used:::

f = **open**("word.txt", "a")

f**.write**('\", \"'**.join**(word\_list))

f**.close**()

Appendix = Clear Screen

|  |  |
| --- | --- |
| import random  import hangman\_words  import hangman\_art  import os  ​  . .... .  . .... .  . .... .  ​      os.system('cls')      print(hangman\_art.stages[lives])      print(disPlay)    . .... .  . .... .  . .... . | #*========== clear screen ============*  **import** os  **import** time  #*for mac*  #*os.system('ls')*  #*for windows*  os**.system**("cls")  time**.sleep**(2)  #*Ubuntu version 10.10*  os**.system**('clear') |