Lesson 4: Mad Libs –

In this lesson, you will create a mad-libs program to generate a funny story. You will ask the user for nouns, verbs, and adjectives then use those words to fill in the blanks for a story. By making this program, you will learn about lists and for loops.

Part 1 – Getting Nouns, Verbs, and Adjectives.

Create a new file like before and save it as **YourName LastInitial Lesson 4 – Mad Libs.py** in your folder.

Let's think about how our program is going to flow.

1. We need to tell the user what it is they'll be doing.
2. We need to get a specific number of nouns from the user.
3. We need to get a specific number of verbs from the user.
4. We need to get a specific number of adjectives from the user.
5. We need to plug in the words into a story and print it out

Step 1 can be a simple print statement, but steps 2-5 will all require several lines of code. We should probably make these functions to organize our code into neat and organized steps.

Let's start by creating a function to get a specific number of nouns. Inside this function, we will create an empty list that we will use later to store our nouns.

```python
def getNouns(numOfNouns):
    nounList = [] #[] creates a list with no values aka an empty list
```

The numOfNouns parameter represents how many nouns we need for our story. We need a for loop so that we keep asking for nouns until we have all the ones we need.

```python
def getNouns(numOfNouns):
    nounList = [] #[] creates a list with no values aka an empty list
    for i in range(numOfNouns):
        noun = input("Give me a noun: ")
```

Right now, our function will ask for the correct number of nouns, but it is not adding it to the list. So, we need to use the append function every time we get a new noun to add it to the list. Make sure you are calling append *inside* the for loop. After the for loop ends, the list will be complete and we can return it.

```
1  def getNouns(numOfNouns):
2      nounList = [] #[] creates a list with no values aka an empty list
3      for i in range(numOfNouns):
4          noun = input("Give me a noun: ")
5          nounList.append(noun)
6      return nounList
```

Remember when a function returns a value, it is given you some sort of result that you can then use later. For example, the input function returns a string and the random.randint() function returns a random integer. The function we just made will return a list of nouns that we can use to fill in the blanks of the story.

To test our function, write these two lines of code. The program will ask you for 3 nouns and then print out all 3 nouns. DELETE THESE TWO LINES AFTER YOU FINISH TESTING.

```
1  def getNouns(numOfNouns):
2      nounList = [] #[] creates a list with no values aka an empty list
3      for i in range(numOfNouns):
4          noun = input("Give me a noun: ")
5          nounList.append(noun)
6      return nounList
7
8  nouns = getNouns(3)
9  print(nouns)
```

Next, let's create a function called getVerbs. It is actually going to be almost identical to the getNouns function, so you can actually just copy and paste lines 1-6 then make the highlighted changes.

```
1  def getNouns(numOfNouns):
2      nounList = [] #[] creates a list with no values aka an empty list
3      for i in range(numOfNouns):
4          noun = input("Give me a noun: ")
5          nounList.append(noun)
6      return nounList
7
```

```
def getVerbs(numOfVerbs):
    verbList = []
    for i in range(numOfVerbs):
        verb = input("Give me a verb: ")
        verbList.append(verb)
    return verbList
```

Lastly, let’s create a variable to get all of our adjectives. Use the other two functions to help you fill in the blanks.

```
def getVerbs(numOfVerbs):
    verbList = []
    for i in range(numOfVerbs):
        verb = input("Give me a verb: ")
        verbList.append(verb)
    return verbList

def getAdjectives(numOfAdjectives):
    adjectiveList = __
    for i in range(___):
        adjective = input("Give me an adjective: ")
        ________________
    return adjectiveList
```

Part 2 – Filling in the Story.

Now, we just need one last function to print out the story for us. Let’s write the function declaration for printStory() which will have three parameters: a list of nouns, a list of verbs, and a list of adjectives.

```
def getAdjectives(numOfAdjectives):
    ...

def printStory(nounList, verbList, adjectiveList):
```

To actually print out the story, we need to access the individual words inside of the lists. Remember, a list is like a container with numbered compartments. The number of the compartment is referred to as an index. To access an element from a list all you use the following format: listName[index]. In this function, we will print each line of the story one line at a time.

```
22 def printStory(nounList, verbList, adjectiveList):
23     print("There once was a", adjectiveList[0], nounList[0], "named Leo.")
```

With the line above, we are printing out the first word from the adjective list and the first word from the noun list. Remember that in programming, we always start counting at 0. On line 24, we will print out the second word from nounList. We used the + for the concatenation instead because we did not want a space between the noun and period. On line 25, we will print out the second word from adjectiveList and the third word from nounList.

```
22 def printStory(nounList, verbList, adjectiveList):
23     print("There once was a", adjectiveList[0], nounList[0], "named Leo.")
24     print("He needed to save the world with his " + nounList[1] + ".")
25     print("Leo's mortal enemy the", adjectiveList[0], nounList[2], "needed
                                                              to be stopped.")
```

All that's left is to print out the two words from verbList. On line 26, print out the first word from verbList. On line 27, print out the second word.

```
22 def printStory(nounList, verbList, adjectiveList):
23     print("There once was a", adjectiveList[0], nounList[0], "named Leo.")
24     print("He needed to save the world with his " + nounList[1] + ".")
25     print("Leo's mortal enemy the", adjectiveList[0], nounList[2], "needed
                                                              to be stopped.")
26     print("The enemy tried to", verbList[_], "Leo but it was not very
                                                                  effective.")
27     print("Then Leo decided to", verbList[_] + ": the ultimate move only a
                                                            few dare to do.")
28     print("With that move, the world was saved!")
```

Part 3 – Main Program

Now that we have written our functions, we can write out the actual program. Just like we said above, we're first going to tell the user what they'll be doing. Then we will get a list of nouns using our getNouns() function. Then get a list of verbs and adjectives using our getVerbs() and getAdjectives() functions. Lastly, we'll use all 3 lists to fill in and print out the story.

```
22 def printStory(nounList, verbList, adjectiveList):
```

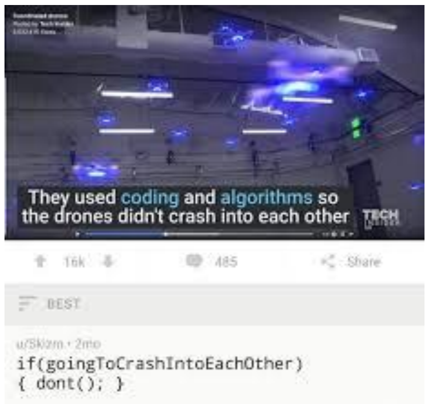
```
...

print("I can create a wacky Mad-Libs story for you")
print("I just need some nouns, verbs, and adjectives from you.")
nouns = getNouns(3) #gets 3 nouns
verbs = getVerbs(2) #gets 2 verbs
adjectives = getAdjectives(_) #gets 2 adjectives
printStory(nouns, verbs, adjectives)
```

Test your program. Make sure that everything is printing out correctly in the end. It should look something like this:

```
I can create a wacky Mad-Libs story for you
I just need some nouns, verbs, and adjectives from you.
Give me a noun: puppy
Give me a noun: paws
Give me a noun: monkey
Give me a verb: tickle
Give me a verb: dab
Give me an adjective: tiny
Give me an adjective: smelly
There once was a tiny puppy named Leo.
He needed to save the world with his paws.
Leo's mortal enemy the smelly monkey needed to be stopped.
The enemy tried to tickle Leo but it was not very effective.
Then Leo decided to dab: the ultimate move only a few dare to do.
With that move, the world was saved!
```

Just like our last program, the program at the end reads very nicely because of our functions. Writing code like this, allows somebody to understand the steps you are taking to complete the program, without necessarily knowing all the details. If they want to look more carefully at how you're doing each step, they can look at the functions. If your final program has parts where it's long and unclear, that is a good indication of that you should probably create a function.

The picture below is a joke about writing a program to avoid having two drones crash into each other, but it does show just how easy it can be to read code by making functions.



They used coding and algorithms so the drones didn't crash into each other

```
if(goingToCrashIntoEachOther)
{ dont(); }
```

Part 4 – Changing the Story

10-15 min.

If you feel confident about functions and want a challenge, you can move on to the BONUS part of this lesson. Otherwise you can just continue with part 4.

Now that you've finished the program, you can try changing the story or just ask for more words so you can make the story longer. Share your stories with your friends as you finish up.

If you want a slightly harder task, try to ask the user for different things like adverbs, foods, names, etc. To do that, just create functions like the ones we used to get the nouns, verbs, and adjectives. You'll also need to add the new type as a parameter for the printStory() function.

BONUS LESSON - REFACTORING:

Sometimes, it is best to just form the basic idea of a program. Then you can just code the program and make sure it works. That being said, you should always have some idea as to what you will be doing before starting to code; even if it's not perfect. That is what we did in this lesson and what we do in each overview.

Once you've made a working program, you can take a close look at your program and see if there aren't any improvements you can make. The process of improving code that has already been written is known as refactoring.

Let's take a look at our code. One easy way to spot code that needs refactoring is to look for any repetitive code. In our program, three functions are nearly identical. Look at these functions and try to see if you can tell what is changing. Do not count variable/function names. These names can be anything and do not affect what the function actually does. Hint: There is only 1 thing that is functionally different between all 3 of them.

```python
def getNouns(numOfNouns):
    nounList = [] #[] creates a list with no values aka an empty list
    for i in range(numOfNouns):
        noun = input("Give me a noun: ")
        nounList.append(noun)
    return nounList

def getVerbs(numOfVerbs):
    verbList = []
    for i in range(numOfVerbs):
        verb = input("Give me a verb: ")
        verbList.append(verb)
    return verbList

def getAdjectives(numOfAdjectives):
    adjectiveList = []
    for i in range(numOfAdjectives):
        adjective = input("Give me an adjective: ")
        adjectiveList.append(adjective)
    return adjectiveList
```

Did you see it? The only thing that is different is the prompt inside the input function telling the user what type of word we want them to type. Instead of having to make different functions for each type of words, why don't we just make one function that will take the prompt as a parameter. Delete the getVerbs and getAdjectives functions, so you only have the getNouns function and the printStory functions.

```
def getNouns(numOfNouns):
    nounList = [] #[] creates a list with no values aka an empty list
    for i in range(numOfNouns):
        noun = input("Give me a noun: ")
        nounList.append(noun)
    return nounList
*DELETE getVerbs AND getADJECTIVES
def printStory(nounList, verbList, adjectiveList):
...
```

We'll change the getNouns() function to the getWords() function. The getWords() function will have two parameters: the prompt telling the user what type of words they should put in and the number of words.

```
def getWords(prompt, numOfWords):
    nounList = [] #[] creates a list with no values aka an empty list
    for i in range(numOfNouns):
        noun = input("Give me a noun: ")
        nounList.append(noun)
    return nounList
```

Now that our function has changed, we need to make a few changes inside of it. First, since it will be used to ask for different types of words, it doesn't make sense to name our variable after nouns. Let's change those names to reflect that they can have any word. Second, we need to use our prompt parameter inside the input function to print out the correct message. The changes are highlighted below.

```
def getWords(prompt, numOfWords):
    wordList = [] #[] creates a list with no values aka an empty list
    for i in range(numOfWords):
        word = input(prompt)
        wordList.append(word)
    return wordList
```

At the end of our program, instead of calling 3 different functions, we will call one function 3 times. Each time, we will use a different prompt so we get different types of words.

```
def getWords(prompt, numOfWords):
...

def printStory(nounList, verbList, adjectiveList):
…

print("I can create a wacky Mad-Libs story for you")
print("I just need some nouns, verbs, and adjectives from you.")
nouns = getWords("Give me a noun: ", 3) #gets 3 nouns
verbs = getWords("Give me a verb: ", 2) #gets 2 verbs
adjectives = getWords("Give me an adjective: ", 2) #gets 2 adjectives
printStory(nouns, verbs, adjectives)
```

By making this change, our program went from 35 lines down to only 20! Also, although we changed the function, our program in the end did not lose any readability. Our variable names and arguments still make it clear what type of words we are getting. We didn’t even have to change our printStory function at all!

Creating the perfect program on the first attempt is impossible. Once they finish writing their code, programmers always realize they could have made their programs better by writing their code differently. Why do you think your games and apps are always getting updated?

Recognizing the ways to improve your code is something that comes with experience, so don’t stress too much about making your programs absolutely perfect. Just focus on making something that works, then make whatever improvements you can.