Kasia Rozanska

Nov. 17, 2020

Foundations of Programming: Python

Assignment05

Using Lists and Dictionaries

# Introduction

In this demonstration we will be looking at the differences between Lists and Dictionaries and creating a menu for a user to manipulate items in a ToDo list. At the end we will briefly discuss GitHub and post our files there.

## Script Templates

Before we dive into our demo, I will quickly step us through creating a script template. We have already covered the importance of adding a header and comments to your scripts, so to begin, open your most recent script that already has your perfected header. Copy the header and select **Settings/Preferences** (Ctrl+Alt+S), select **Editor | File and Code Templates** (See Figure 1). Here you can paste your header and add a name. Since I will be creating most scripts, I have named my template “KR Template” and already logged myself in the ChangeLog. Here you will also add steps that you outline when creating your script, like “Data”, and “Processing.” Templates are useful because they force you to be more consistent, which will also help your scripts look more professional.

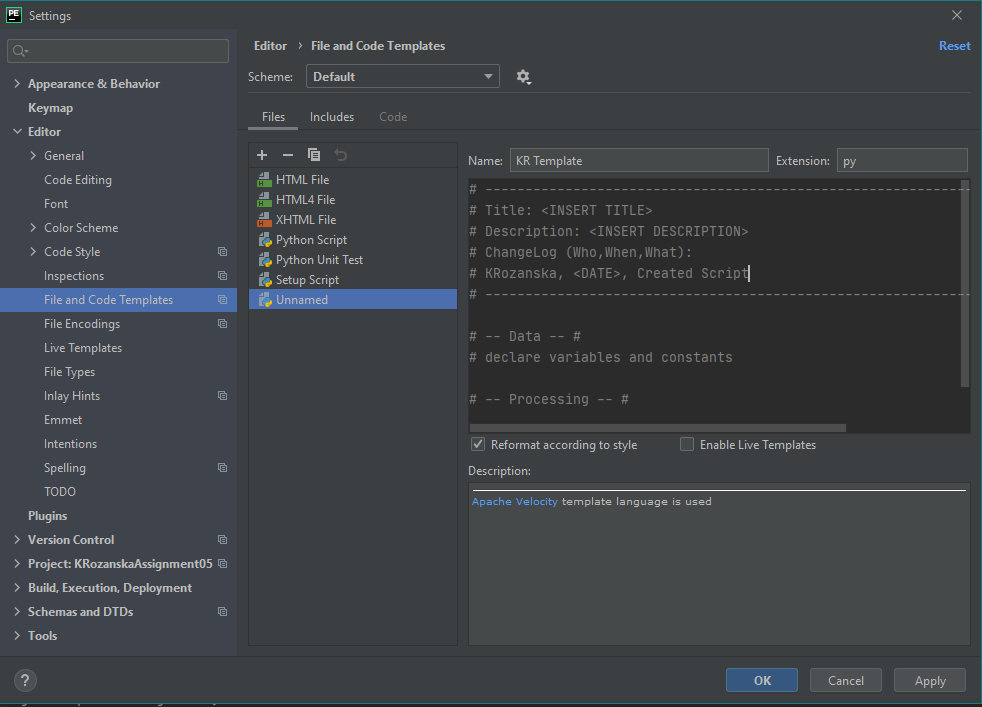


Figure 1: Create A New Script Template

You can now use your template by starting a new file through the Project Explorer, as shown in Figure 2.

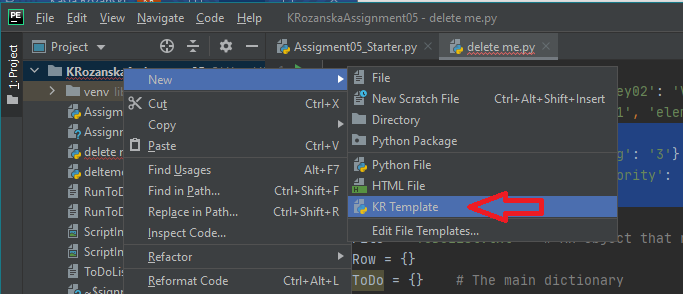


Figure 2: Use Script Template

## Lists vs Dictionaries

A List is a sequence, an ordered set of elements, and therefore uses numbers, Indexes, to access these elements. A Dictionary is the only mapping type of its kind, an unordered set of elements, and therefore uses characters, Keys, to access these elements. A List can be thought of as a row of data, whereas a Dictionary can be thought of as a self-describing row of data. Understanding these differences will show you that Dictionaries can be very handy because the Key is much more descriptive than just a number.

To code these you will use brackets for a List and curly brackets for a Dictionary. This is true for creating Lists and Dictionaries, and when reading and writing data to and from a file. In a List the elements are separated by a comma, and in a Dictionary you create a key-value pair, that is also separated by a comma from other pairs.

Dictionary = {'Key01': 'Value01', 'Key02': 'Value02'}  
 List = ['element01', 'element01', 'element01']

Lists and Dictionaries can also be turned into each other. Below is an example of how the same data would look like from a Dictionary that is transferred into a List.

dicToDo = {'Feed dog': '2', 'Walk dog': '3'} # dictionary {Task, Priority}  
lstToDo = [{'Task': 'Feed dog', 'Priority': '2'}, {'Task': 'Walk dog', 'Priority': '3'}]

## Data

In today’s program we will be presenting a menu to the user, asking for their input, and altering our ToDo List based on their selection. To begin our script, we will declare our variables. Although this is not necessary in Python, it is used in most other languages and will make it easier for us when we begin to work with functions.

You will notice that we have two Dictionaries, one will control the input of a new key-value pair, and the other will be the complete Dictionary that will be useful in determining if given tasks already exist.

# -- Data -- #  
# declare variables and constants  
objFile = "ToDoList.txt" # An object that represents a file  
strFile = "" # A string that represents the characters in a file  
strData = "" # A row of text data from the file  
dicToDo = {} # The main dictionary  
dicRow = {} # A row of data separated into elements of a dictionary lstTable = [] # A list that acts as a 'table' of rows  
strMenu = "" # A menu of user options  
strChoice = "" # A Capture the user option selection

## Processing

To begin, we will load any data from ToDoList.txt into a Python List of Dictionary rows. I wanted to also present the user with any data that may already exist, so that they are aware of what the program is loading into the List, “lstTable”. To achieve this I’ve added, “filesize = os.path.getsize(objFile)”, so that the program could determine whether ToDoList.txt has any data and present the user with either the message that there are no items, or a list of what is already entered.

# -- Processing -- #  
# Step 1 - When the program starts, load any data you have in a text file  
# called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)  
import os  
filesize = os.path.getsize(objFile)  
  
if filesize == 0:  
 print("You have no items on your ToDo List.")

To process the data in the file you want to read it line by line and code how elements are split up. The most common way that programmers split up elements is by using a comma, but you will want to confirm this with any new files you work with. Our string “strFile” represents the characters in the file and “strData” will break them up by row, using the **split()** method, which uses the comma to determine the different elements. Then the elements will be added to the Dictionaries and the List for further use.

else:  
 strFile = open(objFile, "r")  
 print("Current ToDo List:")  
 for row in strFile:  
 strData = row.split(",")  
 dicRow = {"Task": strData[0].strip(), "Priority": strData[1].strip()}  
 dicToDo[strData[0]] = strData[1].strip()  
 lstTable.append(dicRow)  
 print(dicRow["Task"] + ' - ' + dicRow["Priority"])  
 strFile.close()

## Input/Output – Option 1

Before we start manipulating the data in the file we will display a “Menu of Options” to the user. Here we use code we’ve used multiple times, the **print()** function and assigning the user’s input to string, “strChoice.”

# -- Input/Output -- #  
# Step 2 - Display a menu of choices to the user  
while True:  
 print("""  
 Menu of Options  
 1) Show current data  
 2) Add a new item.  
 3) Remove an existing item.  
 4) Save Data to File  
 5) Exit Program  
 """)  
 strChoice = str(input("Which option would you like to perform? [1 to 5] - "))  
 print() # adding a new line for looks

For the first option, “Show current data,” I have reused the same “filesize” logic and added “and not lstTable.” During the second option we will be writing data to our Dictionary and List, but not the file. This “and not lstTable” will ensure that our current list will be displayed to the user after they input new tasks.

# Step 3 - Show the current items in the table  
if (strChoice.strip() == '1'):  
 filesize = os.path.getsize(objFile)  
 if filesize == 0 and not lstTable:  
 print("You have no items on your ToDo List.")  
 else:  
 print("ToDo List:")  
 for row in lstTable:  
 print(row["Task"] + " - " + row["Priority"])  
 continue

## Input/Output – Option 2

During the second option I have used code that displays how to add data to both a List and a Dictionary. If the task the user is entering does not exist in the Dictionary (if strMenu not in dicToDo:) then the line “dicToDo[strMenu] = intPriority” will add both the task and priority. Otherwise, it will display, “That task already exists.” To add the new task to our List we create dicRow and append it to “lstTable,” and complete this option by displaying the input back to the user.

# Step 4 - Add a new item to the list/Table  
elif (strChoice.strip() == '2'):  
 strMenu = str(input("What task would you like to add?: ")).strip()  
 if strMenu not in dicToDo:  
 intPriority = input("What priority would you like to assign it?: ")  
 dicToDo[strMenu] = intPriority  
 dicRow = {"Task": strMenu, "Priority": intPriority}  
 lstTable.append(dicRow)  
 print("\nYou have added task \"" + strMenu + "\" with priority \"" + intPriority + "\".")  
 else:  
 print("\nThat task already exists.")  
 continue

## Input/Output – Option 3

During the third option I have used code that displays how to remove data from both a List and a Dictionary. You will notice that with a List we need to go row by row to find whether the task already exists. This is an example of that main difference between a Dictionary and a List. A Dictionary uses characters to access the Keys and therefore the code can be much simpler: if the Key exists in the Dictionary, delete it. In the List we need to access each row, see whether or not the task exists, and then be able to remove it.

# Step 5 - Remove a new item from the list/Table  
elif (strChoice.strip() == '3'):  
 strMenu = str(input("Which task would you like to remove?: ")).strip()  
 if strMenu in dicToDo:  
 del dicToDo[strMenu]  
 for row in lstTable:  
 if row["Task"] == strMenu:  
 lstTable.remove(row)  
 print("Your task \"", strMenu, "\" has been removed.")  
 else:  
 print("\nError. \"" + strMenu + "\" doesn't exist on the list.")  
 continue

## Input/Output – Options 4 and 5

For the fourth and fifth options I have used code that we are already familiar with. We will use our List to write the data to a file or exit the program.

# Step 6 - Save tasks to the ToDoToDoList.txt file  
elif (strChoice.strip() == '4'):  
 strFile = open(objFile, "w")  
 for row in lstTable:  
 strFile.write(row["Task"] + ',' + row["Priority"] + '\n')  
 strFile.close()  
 print("You list has been saved!")  
 continue  
# Step 7 - Exit program  
elif (strChoice.strip() == '5'):  
 print("Goodbye.")  
 break # and Exit the program

## GitHub

Git is a source control software which allows users to share and work on code, and GitHub is the Internet platform. Many organizations still store their code files on their servers in a location that can be accessed by anyone that needs it, and this ensures that if a personal computer crashes the code is not lost. However, GitHub (among many source control software) creates a platform that is extremely user friendly and already has many useful applications built into it. Among many features, GitHub stores your code in a way that allows for many users to work on it at once, merge changes, roll back updates, and track progress. On a more detailed level it allows users to assign tasks to each other, label them, and track whether they have been completed or not.

This service is used because it takes all the guess work and set-up out of the equation for an organization. Instead of spending time on creating protocols that describe where code is stored, managing spreadsheets that track progress, and endless email chains (that constantly get lost and deleted), programmers can spend their time programming. In my personal experience it helped the communication between upper management and the database team because GitHub is extremely easy to navigate.

## Upload to GitHub

A great introduction to GitHub can be found at <https://lab.github.com/githubtraining/introduction-to-github>. Use this to create a profile and set up a new repository called “IntroToProg-Python.”

Figure 3 shows how your repository menu appears. To add a new file, simply click on “Add file” and select either “Create new file” or “Upload files.” For this demonstration I will upload the Word document I have created.

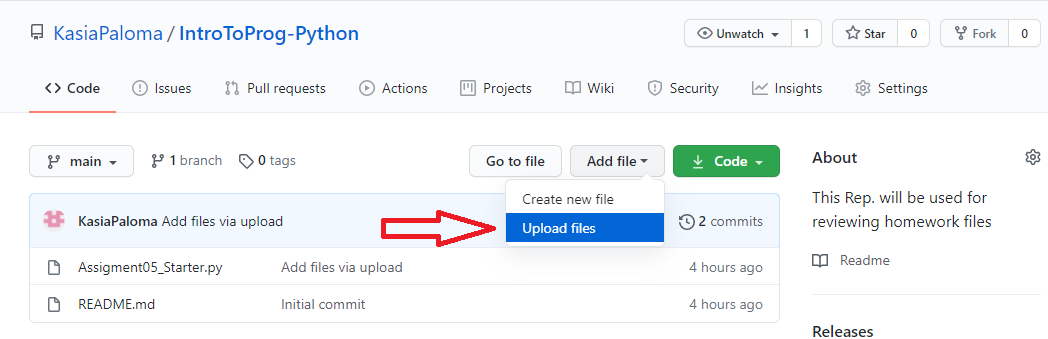


Figure 3: Add File to Repository

Now you can drop your file right into GitHub and save them by clicking “Commit changes” (Figure 4).

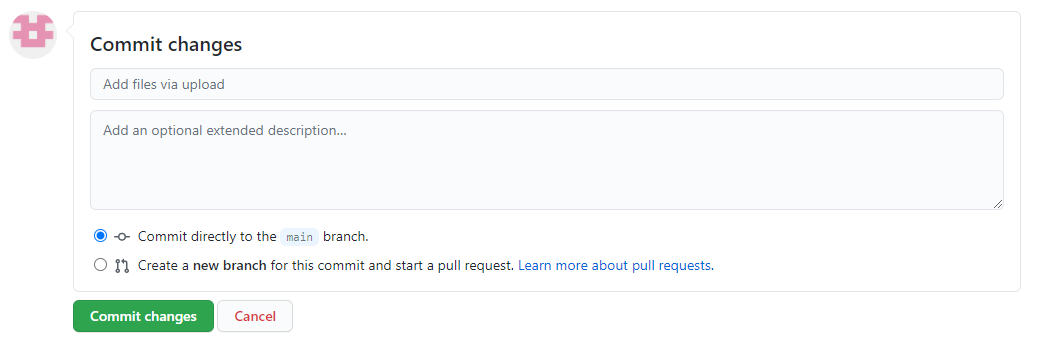


Figure 4

# Summary

You now know the differences between Lists and Dictionaries and have created a program that manipulates a ToDo list. You’ve seen how Lists use an Index and Dictionaries use a Key to find and change data. You’ve created a template for your scripts and created a GitHub repository so that you can share your code.