|  |  |
| --- | --- |
| **LAB 10: Lists (and Dictionaries)** | Name: Lex Baker  Date: 10/25/22 |

**1. Lists -** Almost everything you can do with a tuple, you can also do with a list. The indexing and slicing talked about in the last worksheet works in exactly the same way. In addition, lists are mutable, so you can add, remove, and change elements. Note that lists are defined with square brackets instead of parentheses, but otherwise the syntax is identical to that of tuples.

1. **Adding elements to a list**. Suppose you keep a list of the grades your students receive for each exam.

>>> torisGrades= [80, 95, 88, 73, 92];

If you just had another test so want to add one element to a list you can use the append method:

>>> torisGrades;

>>> torisGrades.append(89);

>>> torisGrades;

You can also use concatenation to add elements to a list:

>>> torisGrades;

>>> torisGrades = torisGrades + [79, 73];

>>> torisGrades;

**b.** Open up a new file in the editor. Add torisGrades to this file and then create an empty list representing another student's grades. Suppose you accidentally lost the data you had saved about one student. Instead of asking to see all the old exams, you decide to randomly assign him grades for each exam. Use a for loop to add a random number between 50 and 100 to this new list until the list is as long as torisGrades

**2.** **Changing elements of a list**

Oops!! You accidentally entered the wrong grade for the second test. Tori actually scored a 93. Go back to the command line in PyScripter where you had already entered Tori's grades (or enter them again if you closed thisRun the following statement to fix the torisGrades list.

>>> torisGrades;

>>> torisGrades[1] = 93;

>>> torisGrades;

You can also delete selected elements. Pretend that for some reason the 5th exam had a couple of ambiguous problems so needed to be deleted from the list. Try the following to see how this works.

>>> torisGrades;

>>> del torisGrades[4];

>>> torisGrades;

Lists are said to be **mutable** because it is possible to change or remove/delete their elements, unlike tuples and strings which are **immutable**.

**3.** Here are some built-in functions and operations that are useful with lists. Run them, write the results, and explain what they do.

>>> len(torisGrades) **# Returns the length of the list**

>>> max(torisGrades) **# Returns the highest value in the list**

>>> min(torisGrades) **# Returns the lowest value in the list**

>>> sum(torisGrades) **# Returns the sum of all values in the list**

>>> 88 in torisGrades **# Returns True if 88 is in the list, False if not**

>>> 70 in torisGrades **# Returns True if 70 is in the list, False if not**

**Output:**

**7**

**93**

**73**

**575**

**True**

**False**

Now open up a file in the editor and write a program that computes the median grade Tori received. Hint – the sort() method or sorted() function is very helpful. What is the difference between the sort() method and the sorted() function?

**sorted(*list*) returns an ordered version of the list without modifying the original, while *list*.sort() modifies and orders the original list directly.**

**median\_grade = sorted(torisGrades)[len(torisGrades) // 2]**

**4.** **Deck of cards.**

**a.** Write a program that creates a list which represents a deck of cards. You start by initializing the following lists:

suits = ['S', 'C', 'H', 'D'];

ranks = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];

deck = [];

Use nested for-loops to fill the deck list so that in the end it contains 52 elements that correspond to the cards in a standard deck. A print call to deck should give:

['2S', '3S', '4S', '5S', '6S', '7S', '8S', '9S', '10S', 'JS', 'QS', 'KS', 'AS', '2C', '3C', '4C', '5C', '6C', '7C', '8C', '9C', '10C', 'JC', 'QC', 'KC', 'AC', '2H', '3H', '4H', '5H', '6H', '7H', '8H', '9H', '10H', 'JH', 'QH', 'KH', 'AH', '2D', '3D', '4D', '5D', '6D', '7D', '8D', '9D', '10D', 'JD', 'QD', 'KD', 'AD']

**b.** Use loops to find a way to "shuffle" the deck. The end result should be a randomly shuffled version of the deck.

**c.** Write a program that will calculate "experimental" probabilities for the situations below. Have the user input how many times they would like to "experimentally" test this (at least 1000) and print out the "experimental probability"

i. Drawing an ace or a club when a single card is drawn.

**Answered in code file, function named: ace\_first()**

ii. Getting a pair when two cards are drawn (make sure you don't draw the same card twice)

**Answered in code file, function named: pair\_first()**

**Extensions** *(If you have extra time, feel free to play around with dictionaries and list extensions in the next two problems. If you feel shaky on any of the concepts you've already learned and think your time would be better spent reviewing those concepts, please ask for more practice problems instead.)*

1. A dictionary is another way to store information. This time, you store information in pairs like a dictionary (a word + it's definition), as the name implies.
   1. Here is an example of CS1 test scores (example scores given by Tom). Type the following line into the interpreter:

>>> scores = {"Ben": 4.1, "Angie": 3.7, "Liz": 2.5, "Heather": 3.3}

This creates a dictionary called scores that has 4 *items*, pairs with a *key* and a *value*. Now let's see how dictionaries work. Type the following commands into the interpreter, figure out what each one does, and write that below next to the line of code.

>>> scores.keys(); # Returns all keys in the dictionary

>>> scores.values(); # Returns all values in the dictionary

>>> scores["Ben"]; # Returns value for key ‘Ben’

>>> scores.get("Angie"); # Returns value for key ‘Angie’

>>> scores.get("Angie", "Key does not exist in dictionary."); # Returns value for key ‘Angie’, and returns "Key does not exist in dictionary." if the key doesn’t exist

>>> scores.get("Mike", "Key does not exist in dictionary."); # Returns value for key ‘Mike’, and returns "Key does not exist in dictionary." if the key doesn’t exist

>>> "Heather" in geek ; # Returns True if key ‘Heather’ is in geek, and False if key is not

# Also throws error because geek doesn’t exist

>>> del scores["Liz"]; # Deletes the key and value for key ‘Liz’

>>> scores["Gen"] = 3.9; # Sets the value for key ‘Gen’ to 3.9

>>> scores.items(); # Returns all key-value pairs for dictionary

To get the values out of a dictionary, you "index" with the keys instead of with a position number.

1. Play around with the commands listed above until you can answer the questions below:
   * 1. Can you check whether a given value is in a dictionary?

**Using << key in dictionary>>**

* + 1. Can you have the same key appear twice in a dictionary?

**No**

* + 1. Can you have the same value appear twice in a dictionary?

**Yes**

* + 1. Can you have an immutable object as a key?

**Yes, in fact, the key must be an immutable object.**

* + 1. Can you have an immutable object as a value?

**Yes, but the value could also be an mutable object.**

**6.** List Comprehensions: Python has another built-in way to very efficiently form new lists called list comprehensions. These are unique to python.

**a.** Type the following in the interpreter and see what happens.

>>> squares = [x\*x for x in range(10)]

>>> squares

>>> evenSquares = [x\*x for x in range(10) if x%2==0]

>>> evenSquares

As you may have noticed, the following for loop:

for item in list

if condition

new\_list.append(expression)

is condensed into a single line

new\_list = [expression for item in list if conditional]

**b.** Using list comprehensions, create a list that multiplies each number in a given list by 7 if the number is divisible by three and ignores all numbers not divisible by three.

**mult\_by\_7 = [x\*7 for x in given\_list if x%3==0]**

**c.** Using list comprehensions, create a new list that contains the first letter of each word in the list words.

>>> words= ['python','onomatopaeia','pernicious','orangutang','ephemeral']

**first\_letters = [x[0] for x in words]**