Charotar University of Science and Technology

Chadubhai S Patel Institute of Technology

U & P U. Patel Department of Computer Engineering

Academic Year: December-April, 2019-2020	**Semester: 6th**
Subject Coordinator: Divyesh Patel	
Subject Teacher Name:Divyesh Patel	
Subject: CE376: PROGRAMMING IN PYTHON	**Hrs./Week: 3**

Practical No	Practical	Extra	Hours	Concept Covered
1	Create a program that asks the user to enter their name and their age. Print out a message addressed to them that tells them the year that they will turn 100 years old.	1. Add on to the previous program by asking the user for another number and printing out that many copies of the previous message. (Hint: order of operations exists in Python) 2. Print out that many copies of the previous message on separate lines. (Hint: the string "\n is the same as pressing the ENTER button)	2	Getting user input Manipulating strings (a few ways)
	Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user. Hint: how does an even / odd number react differently when divided by 2?	1. If the number is a multiple of 4, print out a different message. 2. Ask the user for two numbers: one number to check (call it num) and one number to divide by (check). If check divides evenly into num, tell that to the user. If not, print a different appropriate message.		Modular arithmetic (the modulus operator) Conditionals (if statements) Checking equality
	Take a list, say for example this one: a = [0,0,1,1,1,2,3,5,6,7,13,15,24,43,57,65,87] and write a program that prints out all the elements of the list that are less than 5.	1. Instead of printing the elements one by one, make a new list that has all the elements less than 5 from this list in it and print out this new list. 2. Write this in one line of Python. 3. Ask the user for a number and return a list that contains only elements from the original list a that are smaller than that number		Lists More conditionals (if statements)
2	Create a program that asks the user for a number and then prints out a list of all the divisors of that number. (If you don't know what a divisor is, it is a number that divides evenly into another number. For example, 13 is a divisor of 26 because 26 / 13 has no remainder.)		2	
	Take two lists, say for example these two: a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.	1. Randomly generate two lists to test this 2. Write this in one line of Python (don't worry if you can't figure this out at this point - we'll get to it soon)		
	Ask the user for a string and print out whether this string is a palindrome or not. (A **palindrome** is a string that reads the same forwards and backwards.)			List indexing Strings are lists
	Let's say I give you a list saved in a variable: a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. Write one line of Python that takes this list a and makes a new list that has only the even elements of this list in it.			List comprehensions

3	Make a two-player Rock-Paper-Scissors game. (Hint: Ask for player plays (using input), compare them, print out a message of congratulations to the winner, and ask if the players want to start a new game) Remember the rules: Rock beats scissors Scissors beats paper Paper beats rock		2	While loops Infinite loops Break statements
	Generate a random number between 1 and 9 (including 1 and 9). Ask the user to guess the number, then tell them whether they guessed too low, too high, or exactly right. (Hint: remember to use the user input lessons from the very first practical)	1.Keep the game going until the user types “exit” 2. Keep track of how many guesses the user has taken, and when the game ends, print this out.		Modules Random numbers User input
4	This week’s exercise is going to be revisiting an old exercise (see Practical 2), except require the solution in a different way. Take two lists, say for example these two: a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes. Write this in one line of Python using at least one list comprehension	The original formulation of this exercise said to write the solution using one line of Python, but a few readers pointed out that this was impossible to do without using sets that I had not yet discussed on the blog, so you can either choose to use the original directive and read about the set command in Python 3.3, or try to implement this on your own and use at least one list comprehension in the solution. Extra: Randomly generate two lists to test this	2	List comprehensions Random numbers, continued
	Ask the user for a number and determine whether the number is prime or not. (For those who have forgotten, a prime number is a number that has no divisors.). You can (and should!) use your answer to Practical 2 to help you. Take this opportunity to practice using functions, described below.			Functions Reusable functions Default arguments
	Write a program that takes a list of numbers (for example, a = [5, 10, 15, 20, 25]) and makes a new list of only the first and last elements of the given list. For practice, write this code inside a function.			Lists and properties of lists List comprehensions (maybe) Functions
5	Write a program that asks the user how many Fibonnaci numbers to generate and then generates them. Take this opportunity to think about how you can use functions. Make sure to ask the user to enter the number of numbers in the sequence to generate.(Hint: The Fibonnaci seqence is a sequence of numbers where the next number in the sequence is the sum of the previous two numbers in the sequence. The sequence looks like this: 1, 1, 2, 3, 5, 8, 13, …)		2	Practice functions!
	Write a program (function!) that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates.	Write two different functions to do this - one using a loop and constructing a list, and another using sets. Go back and do Practical 2 using sets, and write the solution for that		Sets

	Write a program (using functions!) that asks the user for a long string containing multiple words. Print back to the user the same string, except with the words in backwards order. For example, say I type the string: My name is Michele Then I would see the string: Michele is name My shown back to me.			More string things
6	Write a password generator in Python. Be creative with how you generate passwords - strong passwords have a mix of lowercase letters, uppercase letters, numbers, and symbols. The passwords should be random, generating a new password every time the user asks for a new password. Include your run-time code in a main method.	Ask the user how strong they want their password to be. For weak passwords, pick a word or two from a list.	2	Random Module in Python
	Use the BeautifulSoup and requests Python packages to print out a list of all the article titles on the New York Times homepage.			Libraries requests BeautifulSoup

	Create a program that will play the “cows and bulls” game with the user. The game works like this:
	
	Randomly generate a 4-digit number. Ask the user to guess a 4-digit number. For every digit that the user guessed correctly in the correct place, they have a “cow”. For every digit the user guessed correctly in the wrong place is a “bull.” Every time the user makes a guess, tell them how many “cows” and “bulls” they have. Once the user guesses the correct number, the game is over. Keep track of the number of guesses the user makes throughout teh game and tell the user at the end.
	
	Say the number generated by the computer is 1038. An example interaction could look like this:
	
	Welcome to the Cows and Bulls Game!
	Enter a number:
	>>> 1234
	2 cows, 0 bulls
	>>> 1256
	1 cow, 1 bull
	...
	Until the user guesses the number.
				Randomness (we’ve covered this a few times before. Mainly in a previous exercise.)
				Functions (covered in a previous exercise also)
				Main method

	Using the requests and BeautifulSoup Python libraries, print to the screen the full text of the article on this website: any news website. The article will be too long, so it is split up between 4 pages. Your task is to print out the text to the screen so that you can read the full article without having to click any buttons. This will just print the full text of the article to the screen. It will not make it easy to read, so next exercise we will learn how to write this text to a .txt file.			Open the web page in Chrome. Right-click the page and click “Inspect Element” Use the magnifying glass on the bottom-left of the page to click on elements of the page and look at their properties.

7	Write a function that takes an ordered list of numbers (a list where the elements are in order from smallest to largest) and another number. The function decides whether or not the given number is inside the list and returns (then prints) an appropriate boolean.	Use Binary Search	2	Booleans - True and False Equality testing Binary search
	Take the code from the How To Decode A Website exercise , and instead of printing the results to a screen, write the results to a txt file. In your code, just make up a name for the file you are saving to.	Ask the user to specify the name of the output file that will be saved.		Writing to a file Gotchas and warnings
	Given a .txt file that has a list of a bunch of names, count how many of each name there are in the file, and print out the results to the screen.	Instead of using the .txt file from above (or instead of, if you want the challenge), take this .txt file, and count how many of each “category” of each image there are. This text file is actually a list of files corresponding to the SUN database scene recognition database, and lists the file directory hierarchy for the images. Once you take a look at the first line or two of the file, it will be clear which part represents the scene category. To do this, you’re going to have to remember a bit about string parsing in Python 3. I talked a little bit about it in this post.		Reading a file Dictionaries
8	Django Framework		2	Real Time Project
Following practicals need to be implement at home for self practice				
	Given two .txt files that have lists of numbers in them, find the numbers that are overlapping. One .txt file has a list of all prime numbers under 1000, and the other .txt file has a list of happy numbers up to 1000. (If you forgot, prime numbers are numbers that can’t be divided by any other number. And yes, happy numbers are a real thing in mathematics - you can look it up on Wikipedia. The explanation is easier with an example, which I will describe below.)			Reading a file, in Practical 7 Number types and converting to integers from strings, in Practical 1 Lists, in Practical 1 and Practical 2
	This exercise is Part 1 of 4 of the Tic Tac Toe exercise series. The other practicals are: 1 ,2 & 3. Time for some fake graphics! Let’s say we want to draw game boards that look like this: --- --- --- | | | | --- --- --- | | | | --- --- --- | | | | --- --- --- This one is 3x3 (like in tic tac toe). Obviously, they come in many other sizes (8x8 for chess, 19x19 for Go, and many more). Ask the user what size game board they want to draw, and draw it for them to the screen using Python’s print statement. Remember that in Python 3, printing to the screen is accomplished by print("Thing to show on screen")	this requires some use of functions, as were discussed		Funtions, ZetCode,

	we've written a program that "knows" a number and asks a user to guess it. This time, we're going to do exactly the opposite. You, the user, will have in your head a number between 0 and 100. The program will guess a number, and you, the user, will say whether it is too high, too low, or your number. At the end of this exchange, your program should print out how many guesses it took to get your number. As the writer of this program, you will have to choose how your program will strategically guess. A naive strategy can be to simply start the guessing at 1, and keep going (2, 3, 4, etc.) until you hit the number. But that's not an optimal guessing strategy. An alternate strategy might be to guess 50 (right in the middle of the range), and then increase / decrease by 1 as needed. After you've written the program, try to find the optimal strategy! (We'll talk about what is the optimal one next week with the solution.)			
9	Today, we will simply focus on checking whether someone has WON a game of Tic Tac Toe, not worrying about how the moves were made. If a game of Tic Tac Toe is represented as a list of lists, like so: game = [[1, 2, 0], [2, 1, 0], [2, 1, 1]] where a 0 means an empty square, a 1 means that player 1 put their token in that space, and a 2 means that player 2 put their token in that space. Your task this week: given a 3 by 3 list of lists that represents a Tic Tac Toe game board, tell me whether anyone has won, and tell me which player won, if any. A Tic Tac Toe win is 3 in a row - either in a row, a column, or a diagonal. Don't worry about the case where TWO people have won - assume that in every board there will only be one winner. Here are some more examples to work with: winner_is_2 = [[2, 2, 0], [2, 1, 0], [2, 1, 1]] winner_is_1 = [[1, 2, 0], [2, 1, 0], [2, 1, 1]] winner_is_also_1 = [[0, 1, 0], [2, 1, 0], [2, 1, 1]] no_winner = [[1, 2, 0],			
10				

	For this exercise, we will keep track of when our friend’s birthdays are, and be able to find that information based on their name. Create a dictionary (in your file) of names and birthdays. When you run your program it should ask the user to enter a name, and return the birthday of that person back to them. The interaction should look something like this:
	
	>>> Welcome to the birthday dictionary. We know the birthdays of:
	Albert Einstein
	Benjamin Franklin
	Ada Lovelace
	>>> Who's birthday do you want to look up?
	Benjamin Franklin
	>>> Benjamin Franklin's birthday is 01/17/1706.			

	In the previous exercise we created a dictionary of famous scientists’ birthdays. In this exercise, modify your program from Part 1 to load the birthday dictionary from a JSON file on disk, rather than having the dictionary defined in the program. Bonus: Ask the user for another scientist’s name and birthday to add to the dictionary, and update the JSON file you have on disk with the scientist’s name. If you run the program multiple times and keep adding new names, your JSON file should keep getting bigger and bigger.			

	In the previous exercise we saved information about famous scientists’ names and birthdays to disk. In this exercise, load that JSON file from disk, extract the months of all the birthdays, and count how many scientists have a birthday in each month.
	
	Your program should output something like:
	
	{
	"May": 3,
	"November": 2,
	"December": 1
	}			

	In the previous exercise we counted how many birthdays there are in each month in our dictionary of birthdays. In this exercise, use the bokeh Python library to plot a histogram of which months the scientists have birthdays in! Because it would take a long time for you to input the months of various scientists, you can use my scientist birthday JSON file. Just parse out the months (if you don’t know how, I suggest looking at the previous exercise or its solution) and draw your histogram. If you are using a purely web-based interface for coding, this exercise won’t work for you, since it requires installing the bokeh Python package. Now might be a good time to install Python on your own computer.			
11				

Practicals related to Networking, Database & GUI will cover in theory sessions itself.