**HAUB SCHOOL OF BUSINESS**

**SAINT JOSEPH’S UNIVERSITY**

**DSS 615: Python Programming**

**Instructor: Michael Ghen**

**Assignment 6**

By:

Vinayak Suresh Tayshetye (10673718)

**Section 4.1**

In Exercises 1 through 24, give the output of the program.

**1. H**

**w**

**3. Enter the population growth as a percent: 2**

**The population will double in about 36.00 years.**

**5. Your income tax is $499.00**

**7. Why do clocks run clockwise?**

**Because they were invented in the northern**

**hemisphere where sundials go clockwise.**

**9. 168 hours in a week**

**76 trombones in the big parade**

**11. President Bush is a graduate of Yale.**

**President Obama is a graduate of Columbia.**

**13. 7**

**5**

**15. Fredrick**

**17. Total cost: $106.00**

**19. 5**

**21. When in the course of human events**

**23. Enter grade on midterm exam: 85**

**Enter grade on final exam: 94**

**Enter type of student (Pass/Fail) or (Letter Grade): Letter Grade**

**Semester grade: A**

**25.**

**def maxim(list1):**

**list1.sort()**

**return list1[len(list1) - 1]**

**list2 = [1076, 4454, 994, 687, 1, 0, 1.9]**

**print(maxim(list2))**

**27.**

**def main():**

**word = input("Enter a word: ")**

**if isQwertyWord(word) == False:**

**print("{:s} is not a Qwerty word.".format(word))**

**else:**

**print("{:s} is a Qwerty word.".format(word))**

**def isQwertyWord(word):**

**qwertyLetters = ["q", "w", "e", "r", "t", "y", "u", "i", "o", "p",**

**"Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"]**

**i = 0**

**for letter in word:**

**if letter in qwertyLetters:**

**i += 1**

**return i == len(word)**

**main()**

**29.**

**def main():**

**salaryOpt1 = salaryOption1()**

**salaryOpt2 = salaryOption2()**

**print("Option 1 pays ${0:,.2f}\nOption 2 pays ${1:,.2f}".format(salaryOpt1, salaryOpt2))**

**if salaryOpt1 == salaryOpt2:**

**print("No option is different.")**

**elif salaryOpt1 > salaryOpt2:**

**print("Option 1 is better.")**

**else:**

**print("Option 2 is better.")**

**def salaryOption1():**

**totalSalary = 100 \* 10**

**return totalSalary**

**def salaryOption2():**

**salary = 1**

**totalSalary = 0**

**for day in range(10):**

**totalSalary += salary**

**salary \*= 2**

**return totalSalary**

**main()**

**33.**

**def main():**

**global colors**

**colors = []**

**letter1 = requestLetter1()**

**colors = obtainColors(letter1)**

**displayColors()**

**def requestLetter1():**

**letter = input("Enter a letter: ")**

**return letter**

**def obtainColors(letter):**

**file = open("G:/SJU/10 - Python/Text File/Colors.txt")**

**list1 = [line.rstrip() for line in file]**

**for color in list1:**

**if (color[0] == letter):**

**colors.append(color)**

**return colors**

**def displayColors():**

**for color in colors:**

**print(" " + color)**

**main()**

**Section 4.2**

In Exercises 1 through 24, determine the output of the program.

**1. 24 blackbirds baked in a pie.**

**3. Cost: $250.00**

**Shipping cost: $15.00**

**Total cost: $265.00**

**5. Enter first grade: 88**

**Enter second grade: 99**

**Enter third grade: 92**

**[88, 92, 99]**

**7. ['Banana', 'apple', 'pear']**

**['apple', 'Banana', 'pear']**

**9. nudge nudge**

**nudge nudge nudge nudge**

**11. spam and eggs**

**spam and eggs**

**13. George Washington**

**John Adams**

**15. Amadeus**

**Joseph**

**Sebastian**

**Vaughan**

**17. ['M', 'S', 'a', 'l', 'o', 't']**

**['a', 'l', 'M', 'o', 'S', 't']**

**19. VB Ruby Python PHP Java C++ C**

**21. Python Java Ruby C++ PHP VB C**

**23. -3 -2 4 5 6**

In Exercises 25 through 30, determine the value of list2, wherelist1 = ["democratic", "sequoia", "equals", "brrr", "break", "two"].

**25. [10, 7, 6, 4, 5, 3]**

**27. ['BRRR', 'TWO']**

**29. ['M', 'S', 'a', 'l', 'o', 't']**

**['a', 'l', 'M', 'o', 'S', 't']**

In Exercises 31 and 32 use list comprehension to simplify the code.

**31.**

**names = ["George Boole", "Charles Babbage", "Grace Hopper"]**

**lastNames = [name.split()[-1] for name in names]**

**In Exercises 33 through 36, describe the output displayed by the lines of code, where the file States.txt contains the 50 U.S. states in the order they joined the union.**

**33. ['DELAWARE', 'PENNSYLVANIA', 'NEW JERSEY', 'GEORGIA', 'CONNECTICUT', 'MASSACHUSETTS', 'MARYLAND', 'SOUTH CAROLINA', 'NEW HAMPSHIRE', 'VIRGINIA', 'NEW YORK', 'NORTH CAROLINA', 'RHODE ISLAND', 'VERMONT', 'KENTUCKY', 'TENNESSEE', 'OHIO', 'LOUISIANA', 'INDIANA', 'MISSISSIPPI', 'ILLINOIS', 'ALABAMA', 'MAINE', 'MISSOURI', 'ARKANSAS', 'MICHIGAN', 'FLORIDA', 'TEXAS', 'IOWA', 'WISCONSIN', 'CALIFORNIA', 'MINNESOTA', 'OREGON', 'KANSAS', 'WEST VIRGINIA', 'NEVADA', 'NEBRASKA', 'COLORADO', 'NORTH DAKOTA', 'SOUTH DAKOTA', 'MONTANA', 'WASHINGTON', 'IDAHO', 'WYOMING', 'UTAH', 'OKLAHOMA', 'NEW MEXICO', 'ARIZONA', 'ALASKA', 'HAWAII']**

**35. ['OHIO', 'IOWA', 'UTAH', 'MAINE', 'TEXAS', 'IDAHO', 'OREGON', 'KANSAS', 'NEVADA', 'ALASKA', 'HAWAII', 'GEORGIA', 'VERMONT', 'INDIANA', 'ALABAMA', 'FLORIDA', 'MONTANA', 'WYOMING', 'ARIZONA', 'DELAWARE', 'MARYLAND', 'VIRGINIA', 'NEW YORK', 'KENTUCKY', 'ILLINOIS', 'MISSOURI', 'ARKANSAS', 'MICHIGAN', 'NEBRASKA', 'COLORADO', 'OKLAHOMA', 'TENNESSEE', 'LOUISIANA', 'WISCONSIN', 'MINNESOTA', 'NEW JERSEY', 'CALIFORNIA', 'WASHINGTON', 'NEW MEXICO', 'CONNECTICUT', 'MISSISSIPPI', 'PENNSYLVANIA', 'RHODE ISLAND', 'NORTH DAKOTA', 'SOUTH DAKOTA', 'MASSACHUSETTS', 'NEW HAMPSHIRE', 'WEST VIRGINIA', 'SOUTH CAROLINA', 'NORTH CAROLINA']**

**In Exercises 37 through 42, determine whether the calling statement is valid where the function definition has the header def bestFilm(year, film, star):.**

**37. Valid**

**39. Valid**

**41. Not Valid**

**In Exercises 43 through 46, determine whether the calling statement is valid where the function definition has the header def breakfast(toast, coffee, spam=0, eggs=0):.**

**43. Valid**

**45. Not Valid**

In Exercises 47 and 48, determine the output displayed by the statement.

**47. almost**

**49.**

**def main():**

**numOunces = eval(input("Enter the number of ounces: "))**

**print("Cost: ${:.2f}".format(cost(numOunces)))**

**def cost(numOunces):**

**cost = .05**

**ounces = ceil(numOunces)**

**for ounce in range(numOunces - 1):**

**cost += .10**

**return cost**

**def ceil(x):**

**if isinstance(x, float): #To round up.**

**x = int(x) + 1**

**else:**

**pass**

**return x**

**main()**

**51.**

**def main():**

**phrase1 = input("Enter the first word or phrase: ")**

**phrase2 = input("Enter the second word or phrase: ")**

**if areAnagrams(phrase1, phrase2):**

**print("Are anagrams.")**

**else:**

**print("Are not anagrams.")**

**def areAnagrams(phrase1, phrase2):**

**remove = [" ", ".", ","]**

**str1 = sorted([word.lower() for word in phrase1 if word not in remove])**

**str2 = sorted([word.lower() for word in phrase2 if word not in remove])**

**return str1 == str2**

**main()**

**53.**

**def main():**

**pres = [("Lydon", "Johnson"), ("John", "Kennedy"), ("Andrew", "Johnson")]**

**for tuple in NameSort(pres):**

**tuple = (tuple[1], tuple[0])**

**print(", ".join(tuple))**

**def NameSort(list1):**

**list2 = firstNameSort(list1)**

**list3 = lastNameSort(list2)**

**return list3**

**def firstNameSort(list1):**

**list1.sort(key=lambda x: x[0])**

**return list1**

**def lastNameSort(list1):**

**list1.sort(key=lambda x: x[-1])**

**return list1**

**main()**

**55.**

**NE = [("Maine", 30840, 1.329), ("Vermont", 9217, .626), ("New Hampshire", 8953,**

**1.321), ("Massachusetts", 7800, 6.646), ("Connecticut", 4842, 3.59), ("Rhode Island",**

**1044, 1.05)]**

**def main():**

**print("Sorted by land area in descending order:")**

**print(" ".join(descLandSort(NE)))**

**def descLandSort(list1):**

**list1.sort(key = lambda x: -x[1])**

**list1 = [word[0] for word in list1]**

**return list1**

**main()**

**57.**

**def main():**

**print("Sorted by population density in ascending order:")**

**print(" ".join(densityAscSort(NE)))**

**def densityAscSort(list1):**

**list1.sort(key = lambda x: x[-1] / x[1])**

**list1 = [word[0] for word in list1]**

**return list1**

**main()**

**59.**

**numbers = [865, 1169, 1208, 1243, 290]**

**def main():**

**print("Sorted by largest prime factor:")**

**print(primeAscSort(numbers))**

**def primeAscSort(list):**

**list.sort(key = largestPrimeFac)**

**return list**

**def largestPrimeFac(num):**

**f = 2**

**while num > 1:**

**if num % f == 0:**

**num = num / f**

**else:**

**f += 1**

**return f**

**main()**

**61.**

**def main():**

**print("Sorted by sum of odd digits:")**

**print(orderSort(numbers))**

**def orderSort(list1):**

**list1.sort(key = findOddDigit)**

**return list1**

**def findOddDigit(num):**

**sum = 0**

**for digit in str(num):**

**if (int(digit) % 2) != 0:**

**sum += - int(digit)**

**else:**

**pass**

**return sum**

**main()**

**63.**

**def main():**

**file = open("G:/SJU/10 - Python/Text File/USPres.txt", 'r')**

**list1 = nameSort([line.rstrip() for line in file])**

**file.close()**

**printFirstSix(list1)**

**def nameSort(list2):**

**list2.sort(key=firstNameLength)**

**return list2**

**def firstNameLength(name):**

**list2 = name.split(" ")**

**return (len(list2[0]))**

**def printFirstSix(list2):**

**for pres in range(6):**

**print (list2[pres])**

**main()**

**65.**

**def main():**

**file = open("G:/SJU/10 - Python/Assignments/Data Files/States.txt", 'r')**

**list1 = nameSort([line.rstrip() for line in file])**

**file.close()**

**printFirstSix(list1)**

**def listSort(list2):**

**list2.sort(key = numberOfVowels)**

**return list2**

**def numberOfVowels(name):**

**i = 0**

**vowels = ["a", "e", "i", "o", "u"]**

**for letter in name:**

**if letter in vowels:**

**i += -1**

**return (i)**

**def printFirstSix(list2):**

**for x in range(6):**

**print (list2[x])**

**main()**

**67.**

**def main():**

**input()**

**result = calculatedNewBalance(oldBal, charges, credit)**

**output(result[0], result[1])**

**def input():**

**global oldBal, charges, credit**

**oldBal = eval(input("Enter old balance: "))**

**charges = eval(input("Enter charges for month: "))**

**credit = eval(input("Enter credits: "))**

**def calculatedNewBalance(oldBal, charges, credit):**

**newBal = (oldBal \* 0.015) + charges + credit**

**if newBal <= 20:**

**minPay = newBal**

**else:**

**minPay = 20 + ((newBal - 20) \* 0.1)**

**return newBal, minPay**

**def output(x, y):**

**print("New balance: ${0:.2f}.\nMinimum payment: ${1:.2f}".format(x, y) )**

**main()**

**69.**

**def main():**

**enterInput()**

**result(weeklyPay(hoursWorked, hourlyPay))**

**def enterInput():**

**global hoursWorked, hourlyPay**

**hoursWorked = eval(input("Enter hours worked: "))**

**hourlyPay = eval(input("Enter hourly pay: "))**

**def weeklyPay(hoursWorked, hourlyPay):**

**if hoursWorked > 40:**

**weekPay = (40 \* hourlyPay) +((hoursWorked - 40) \* (hourlyPay \* 1.5))**

**else:**

**weekPay = (hoursWorked \* 10)**

**return weekPay**

**def result(x):**

**print("Week's pay: ${:.2f}".format(x))**

**main()**