**Debugging & Error Handling**

**1. Type and fix the following broken codes. This program will average 3 test scores and print out each test score, the average test score, the corresponding letter grade, and a message stating whether the student is passing or not.**

****

#Q1 - Answer

exam\_one = int(input("input exam grade one: "))

exam\_two = int(input("input exam grade two: "))

exam\_three = int(input("input exam grade three: ")) #the name was fixed, and adding one parenthesis at the end

grades = [exam\_one, exam\_two, exam\_three] #adding camma

sum=0

avg=0 #this should be removed

for grade in grades: #fix variable names (grades)

sum = sum + grade

avg = sum / len(grades)#it should be in the for loop so needs indentation

if grade >=90:

letter\_grade = "A"

elif grade >= 80 and grade < 90: #need : at the end

letter\_grade = "B"

elif grade >= 80 and grade < 90:

letter\_grade = "A"

elif grade > 69 and grade < 80:

letter\_grade = "C"

elif grade <= 69 and grade > 65:

letter\_grade = "D"

else: #elif is incorrect, changed by else

letter\_grade = "F"

print("Exam: " + str(grade)) #for loop was removed as it was used in upper lines (was moved to upper lines)

print("Grade: " + letter\_grade)

print("Average:" + str(avg))

if letter\_garde=="F": #is is incorrect(using == instead)

print("Student is failing.")

else:

print("student is passing.")

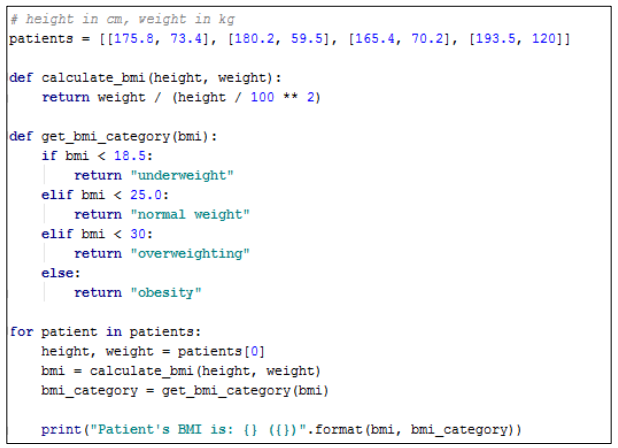
**2. Fix the following codes that include logical errors. BMI (Body Mass Index) = weight (kg) / height (m)2 Outputs should be:**

**Patient's BMI is: 23.74971040884447 (normal weight)**

**Patient's BMI is: 18.323456117940236 (underweight)**

**Patient's BMI is: 25.660558332809398 (overweight)**

**Patient's BMI is: 32.04935600825271 (obesity)**

****

#Q2 – Answer:

patients = [[175.8, 73.4], [180.2, 59.5], [165.4, 70.2], [193.5, 120]]

def calculate\_bmi(height, weight):

return weight / ((height/100)\*\*2) #modifying the structure of parenthesis to correct the formula

def get\_bmi\_category(bmi):

if bmi < 18.5:

return "underweight"

elif bmi < 25.0:

return "normal weight"

elif bmi < 30:

return "overweighthing"

else:

return "obesity"

for patient in patients:

height = patient[0] #height is the first elemant of patient list

weight = patient[1]#weight is the second elemant of patient list

bmi = calculate\_bmi(height,weight)

bmi\_category = get\_bmi\_category(bmi)

print("Patient's BMI is: {} {}".format(bmi, bmi\_category)) #paranthesis were modified

**3. Write a program that asks a user to input a numeric value. If the user inputs a value that cannot be converted to a numeric value, the program will catch the exception and print the exception message. Use str.format() to format and output the message.**

#Q3 #using the try except structure, and ValueError built-in function in case the user intered an invalid value. Also using while ture and break to break if the user entered the correct input

while True:

try:

numeric = int(input("please enter an integer: "))

print("Your number is {}".format(numeric))

break

except ValueError:

print("Please enter the correct value-an integer!")

**4. Write a function, divide(), that takes 2 numeric arguments, dividend and divisor. The function returns a quotient (dividend/divisor). Handle ZeroDivisionError. Execute the function to test if the exception is handled correctly.**

#Q4 #using a function with two arguments. using try and except for obliging the user to enter a viable divisor; ZeroDivisionError built-in function is used

def devide(dividend, divisor):

try:

print("The quotient is {}".format(dividend/divisor))

except ZeroDivisionError:

print("Divided by zero is not possible")

devide(12,0)

**5. Write a function that takes one numeric type positional argument (required). The function should keep asking a user to input two numbers, computes the sum of two numbers, and exits the function when the sum is the same as the input numeric argument. When a user input non-numeric values, then catch the exception, print the exception type and value, and terminate the program.**

**(Pseudo Codes: This is just one example!)**

**Function definition for sum\_two\_numbers (one numeric argument)**

**Loop**

**Try**

**Get the first input from a user**

**Get the second input from a user**

**Sum two values**

**Catch any errors**

**Handle errors**

**If the sum is equal to the input argument then exit the function**

**Call the function here**

**(Hint 1): use input() to get two string inputs from a user**

**(Hint 2): use float() to convert those strings to floating values when possible**

#Q5 – Answer:

#defining the function to take the sum value as a variable argument. using the try except with while and break to set the algorithm. If the user entered two numerical values, equality of their sum to the entered sum value will be chacked (here it is 15)

#in case a non-numerical value is entered an error message is displayed, then it will be shown and the code ends.

def sum\_two\_nums(numeric):

while True:

try:

first\_input=float(input("please enter the first integer: "))

second\_input=float(input("please enter the second integer: "))

sum= first\_input+second\_input

except Exception as e:

print(type(e))

print(e)

break

if sum==numeric:

break

sum\_two\_nums(15)

**6. Practice conda commands.**

**Execute the following conda commands on your command prompt (no point, no need to answer).**

**conda -h**

**conda -V**

**conda create -h**

**conda info**

**conda info --envs**

**(PCs in SAL) conda.exe is in C:\Apps\Anaconda3\Script\conda.exe**

**7. Create a conda virtual environment of your own with Python version 2.7. Also, create a conda virtual environment of your own with a Python version 3.x. Answer the following questions and write down your answers using comments in your python script.**

**a) Write down the command syntaxes you used to create those virtual environments.**

**b) Where are your virtual environments stored on your computer? Write down the directory paths.**

**e.g.) C:\Users\MyUserName\MyVirtualEnv\py39**

**c) Set up your PyCharm project with the virtual environment, Python 3.x, that you created above (no need to answer this question).**

#Q7 – Answer:

#I used these command syntaxes to create virtual environments

#conda update conda

#conda create -n Mahy1 python=2.7 anaconda

#conda activate Mahy1

#conda update conda

#conda create -n Mahy2 python=3.7 anaconda

#conda activate Mahy2

#Directory path to Pythone version 2.7: C:\Users\Mahdieh\Anaconda3\envs\Mahy1

#Directory path to Pythone version 3.7: C:\Users\Mahdieh\Anaconda3\envs\Mahy2