Assignment 1

Software Technology 1 - 4483

Laika Marshall - u3224614

Table of Contents

Question 1

Analysis

This question was not at all difficult, and really just relies on some basic arithmetic questions and taking inputs from a user. It takes the inputs of name, hours worked, pay rate, tax rate and medicare levy, runs some basic calculations, and spits some numbers back out.

Algorithm Design

For this question I decided to use functions rather than putting it all into one main function, partially because it makes me feel cool and smart and partially because Julio said it was good practice to do in Intro to IT. Calculations are as follows:
Tax Deduction = tax rate * pay
Medicare levy = levy rate * pay
Tax amount = tax rate * 100
Medicare deduction amount = medicare levy * 100
Deductions = tax deductions + Medicare levy
Net pay = pay - deductions

Code

```
def main():
	employeeName = getName()
	hoursWorked = getHours()
	hourlyRate = getHourlyRate()
	atoRate = getATORate()
	medicareLevy = getMedicareLevi()
	grossPay = calculateGrossPay(hoursWorked, hourlyRate)
	print("\n=====================================================")
	print("Employee Name: {}".format(employeeName))
	print("Hours Worked: {}".format(hoursWorked))
	print("Pay Rate: ${}".format(hourlyRate))
	print("Gross Pay: ${}".format(grossPay))
	netPay = getDeductions(grossPay, atoRate, medicareLevy)
	print("Net pay: {}".format(netPay))
	print("=====================================================\n")


def getName():

	name = input("Enter employee's Name: ")
	return name

def getHours():
	hours = float(input("Enter number of hours worked in a week: "))
	return hours
```

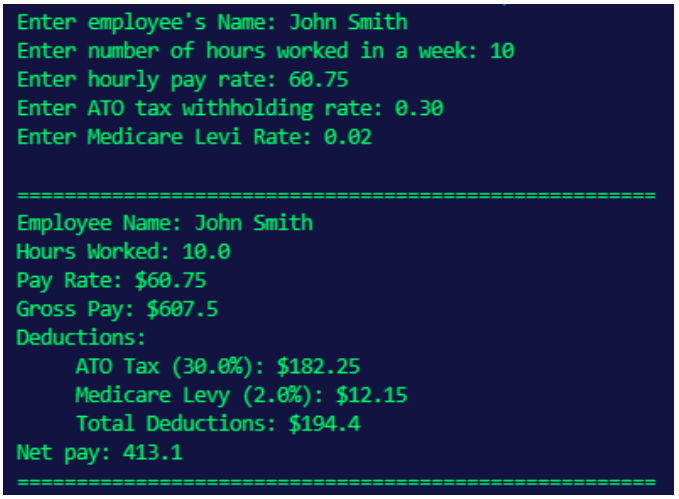
```
def getHourlyRate():
      rate = float(input("Enter hourly pay rate: "))
      return rate

def getATORate():
      ATO = float(input("Enter ATO tax withholding rate: "))
      return ATO

def getMedicareLevi():
      levi = float(input("Enter Medicare Levi Rate: "))
      return levi

def calculateGrossPay(hours, rate):
      pay = hours * rate
      return pay

def getDeductions(pay, tax, levy):
      taxDeduction = tax * pay
      medicare = levy * pay
      taxAmt = tax * 100
      levyAmt = levy * 100
      deductions = taxDeduction + medicare
      netPay = pay - deductions
      print("Deductions: ")
      print("     ATO Tax ({}%): ${}".format(taxAmt, taxDeduction))
      print("     Medicare Levy ({}%): ${}".format(levyAmt, medicare))
      print("     Total Deductions: ${}".format(deductions))
      return(netPay)
main()
```

Test Plan

Really just run different variables through the program

Testing Screencaps

Control:



Test case 1:

```
Enter employee's Name: Amber Katt
Enter number of hours worked in a week: 17
Enter hourly pay rate: 42
Enter ATO tax withholding rate: 0
Enter Medicare Levi Rate: 0

==================================================
Employee Name: Amber Katt
Hours Worked: 17.0
Pay Rate: $42.0
Gross Pay: $714.0
Deductions:
     ATO Tax (0.0%): $0.0
     Medicare Levy (0.0%): $0.0
     Total Deductions: $0.0
Net pay: 714.0
==================================================
```

Test case 2

```
Enter employee's Name: the cat
Enter number of hours worked in a week: 0
Enter hourly pay rate: 0
Enter ATO tax withholding rate: 0
Enter Medicare Levi Rate: 0

==================================================
Employee Name: the cat
Hours Worked: 0.0
Pay Rate: $0.0
Gross Pay: $0.0
Deductions:
     ATO Tax (0.0%): $0.0
     Medicare Levy (0.0%): $0.0
     Total Deductions: $0.0
Net pay: 0.0
==================================================
```

Question 2

Analysis

This question is largely based on conditionals and branching statements, and takes simple yes/no inputs and spits out restaurants relevant to what you answered. Basically, the goal is to sort through a list of restaurants and display the ones that fit a specific set of conditions.

Algorithm design

My first attempt took inspiration for how the unix file system handles permissions, which is the addition of certain odd integers to a counter, with each possible final result corresponding with a combination of options. This quickly proved to be impractical.
My second attempt was utilising just plain conditional statements, which quickly got convoluted and confusing. It did, however, prove to be a step in the right direction.
Eventually I ended up using nested conditionals, which I found worked on the principle of eliminating options rather than adding options. Since some options would be eliminated by different conditions, valueError exceptions where added to handle removing what has already been removed

Code

(excluding unused code blocks)

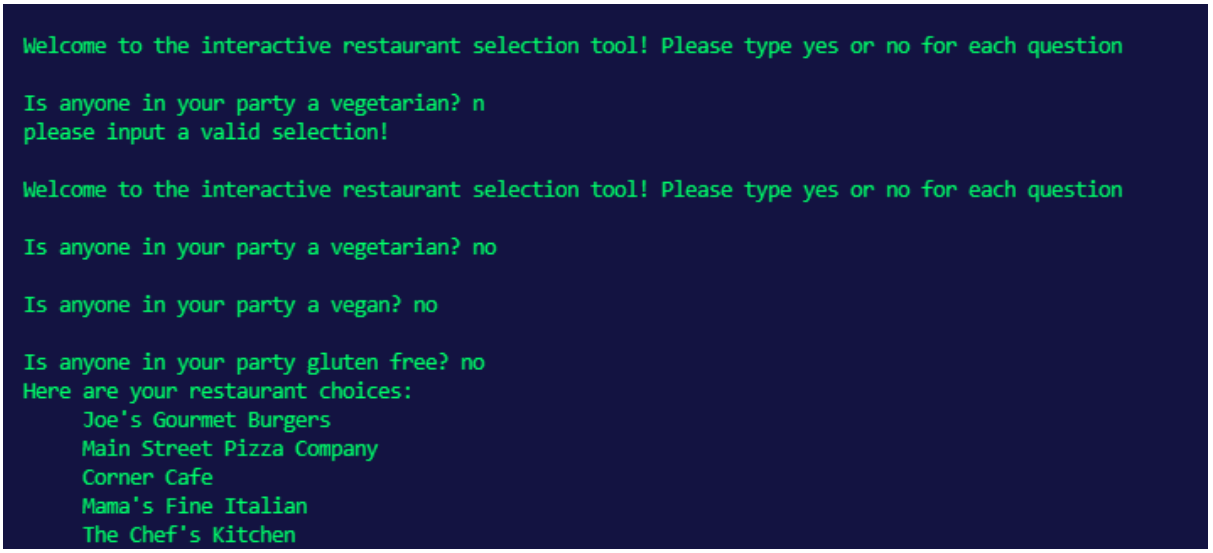
```
def main():
    while True:
        print("\nWelcome to the interactive restaurant selection tool!
Please type yes or no for each question")
        veggie = input("\nIs anyone in your party a vegetarian? ")
        if veggie == "no":
            #partyCounter = 0
            vegetarian = False
            break
        elif veggie == "yes":
            #partyCounter = 1
            vegetarian = True
            break
        else:
            print("please input a valid selection!")
    while True:
        vegan = input("\nIs anyone in your party a vegan? ")
        if vegan == "no":
            vegan = False
            break
        elif vegan == "yes":
            vegan = True
            break
```

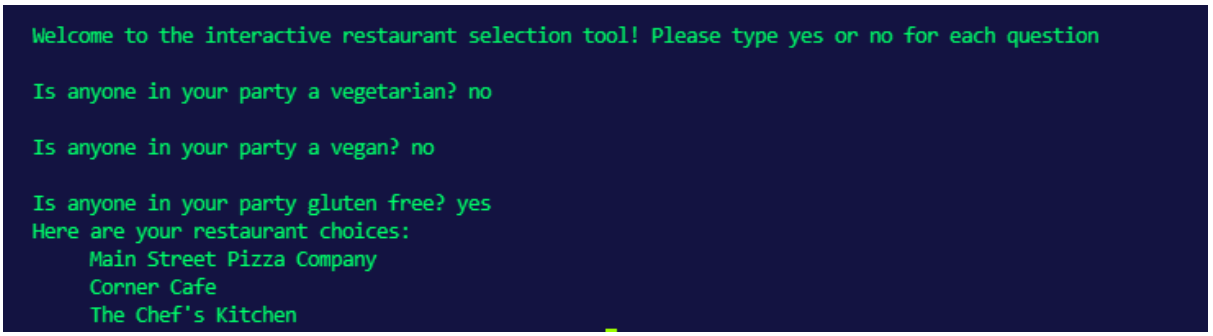
```
        else:
            print("please input a valid selection! (yes/no)")
    while True:
        gf = input("\nIs anyone in your party gluten free? ")
        if gf == "no":
            glutenFree = False
            break
        elif gf == "yes":
            glutenFree = True
            break
        else:
            print("please input a valid selection! (yes/no)")

    outputSelection(vegetarian, vegan, glutenFree)
def outputSelection(vegetarian, vegan, glutenFree):
    print("Here are your restaurant choices:")
####### my third attempt, where i realised that rather than adding onto
what we say, we should remove from a list. The try/except calls are so that
we don't get valueErrors from trying to remove something that's already
been removed
    restaurants = ["Joe's Gourmet Burgers", "Main Street Pizza Company",
"Corner Cafe", "Mama's Fine Italian", "The Chef's Kitchen"]
    if vegetarian == True:
        restaurants.remove("Joe's Gourmet Burgers")
    if vegan == True:
        try:
            restaurants.remove("Joe's Gourmet Burgers")
        except ValueError:
            pass #do nothing!
        restaurants.remove("Main Street Pizza Company")
        restaurants.remove("Mama's Fine Italian")
    if glutenFree == True:
        try:
            restaurants.remove("Joe's Gourmet Burgers")
        except ValueError:
            pass #do nothing!
        try:
            restaurants.remove("Mama's Fine Italian")
        except ValueError:
            pass #do nothing!
    for i in restaurants:
        print("     {}".format(i))
main()
```

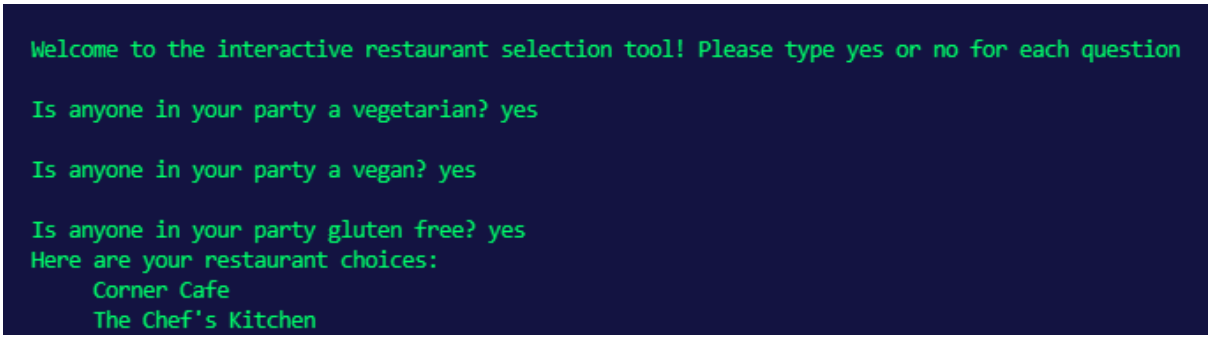
Testing Plan

Well, it's not like there's a huge amount of things I can really put here other than every combination of answer, which would kinda just be like y/y/y or y/n/y.. Etc. I did implement input checking for this one as it would cause the program to crash if it had a non expected input but I didn't go super wild with it since I've lost marks for going above and beyond before

Testing Screencaps







Obviously I won't run every single possible iteration as that's like 2^3 screenshots and it's tedious to go through all that

Question 3

Analysis

The aim here is to create a program that follows arithmetical operations a specific number of times, taking a starting number, percentage increase and number of iterations and spits out information according to each iteration.

Algorithm Design

My thought process follows: “this question obviously calls for the use of repetition structures. The choice of using a while loop or for loop, at least at this level of simplicity, is a completely arbitrary choice that just slightly changes the metaphorical wording of the program. From there, it’s just a simple point of using the arithmetics”
I did have a slight issue with the mathematics of this, as originally I thought to get the increase of the organisms you needed to follow a formula of
$increase = (organisms / multiplication) * 100$
$organisms = organisms + increase$
Which, in fairness, was just a mistake of me being bad at maths. Lol. But eventually I figured it out!
I couldn’t figure out what rounding rules where implied in the assignment sheet so I guessed that we round up to the 7th decimal place
I also put stuff in main() because I’m lazy!

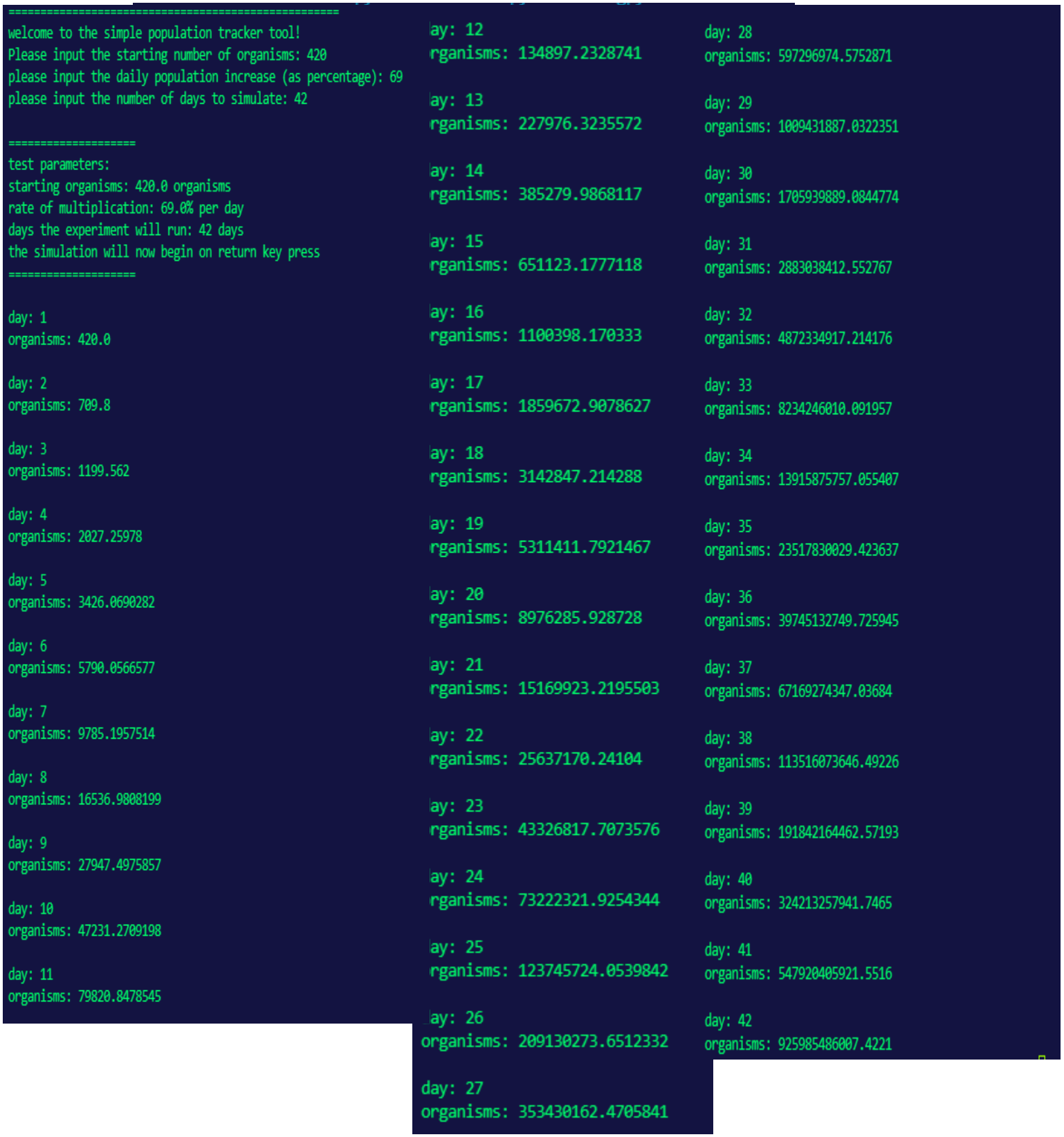
Code

```
def main():
    print("=====================================================")
    print("welcome to the simple population tracker tool!")
    startOrganisms = float(input("Please input the starting number of
organisms: "))
    popIncrease = float(input("please input the daily population increase
percentage (as decimal): "))
    simLength = int(input("please input the number of days to simulate: "))
    print("\n====================")
    print("test parameters: ")
    print("starting organisms: {} organisms".format(startOrganisms))
    print("rate of multiplication: {}% per day".format(popIncrease))
    print("days the experiment will run: {} days".format(simLength))
    print("the simulation will now begin on return key press")
    input("====================")
    calculate(startOrganisms, popIncrease, simLength)


def calculate(organisms, multiplication, days):
    for i in range(days):
        if i > 0:
            organisms = organisms * (1 + (multiplication / 100))
        # print("increase: {}".format(increase))                here for
testing purposes!
        print("\nday: {}".format(i+1))
        print("organisms: {}".format(round(organisms, 7)))
        i + 1


main()
```

Testing plan

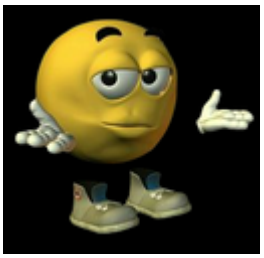
Uhhhh again just gonna throw in some random data. I'll also prove that input validation isn't implemented but y'know it wasn't in the assignment question and I've proven I can do it so I didn't feel like spending the extra time. Hopefully it's fine since I've already demonstrated it once?

Testing Screencaps



```
==================================================
welcome to the simple population tracker tool!
Please input the starting number of organisms: no
Traceback (most recent call last):
  File "c:\Users\danie\Desktop\software technology a1\populationTracker.py", line 38, in <module
    main()
  File "c:\Users\danie\Desktop\software technology a1\populationTracker.py", line 15, in main
    startOrganisms = float(input("Please input the starting number of organisms: "))
ValueError: could not convert string to float: 'no'
```

Obviously, non numerical input breaks it. I could implement input validation but uhhhhhh:



Question 4

Analysis

Okay, this is a pretty cool question. It involves taking a password (and requirements) from a user and checking if the password fits said requirements. In the assignment sheet it takes a password and states “valid” or “invalid”. I went a bit further for the sake of being pedantic, and implemented the ability to set the rules. But that was really to I could get more practice with class structure

Algorithm Design

This program was designed using classes and objects. I defined a password class, and incorporated all the checks as methods. The class is constructed with the requirements (which are passed to the methods) and the password itself
From there it’s simply a matter of getting the inputs from the user, using them to create a password object and using that object's methods. It was quite insightful as to getting a better understanding of how classes and objects work in OOP.

Code

```
from string import punctuation
#define the class for passwords
class passwordChecker:
      def __init__(self, charMin, symbolsMin, specialChars, passwrdToChk):
      self.charMin=int(charMin)
      self.symbolsMin=int(symbolsMin)
      self.specialChars=bool(specialChars)
      self.passwrdToChk=passwrdToChk
#checks the length of the password
      def charCheck(self):
      if len(self.passwrdToChk) < self.charMin:
            return False
      elif len(self.passwrdToChk) > self.charMin:
            return True
#checks if numbers (0-9) and punctuation (!,@,#,$ etc..) are present, and
how many are present
      def symbolsCheck(self):
      punctuationCounter = 0
      numCounter = 0
      nums = ("1","2","3","4","5","6","7","8","9","0")
      symbols = set(punctuation)
      for i in self.passwrdToChk:
            if i in symbols:
            punctuationCounter += 1
            if i in nums:
            numCounter += 1
```

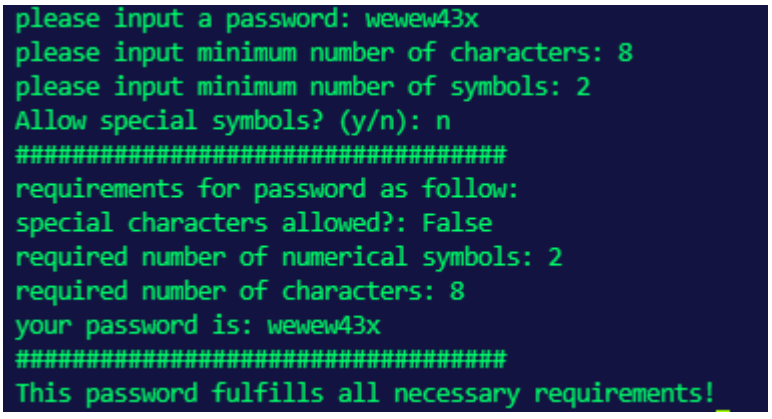
```
      symbolsTotal = numCounter + punctuationCounter
      if symbolsTotal >= self.symbolsMin:
            return True
      else:
            return False
#checks if special characters are allowed or not, and if they aren't,
checks if the password is alpha-numeric only (doesn't include punctuation!)
      def specialCharsCheck(self):
      if self.specialChars == False:
            return self.passwrdToChk.isalnum()
      else:
            return True
#reminds you of the requirements for a password
      def requirements(self):
      print("requirements for password as follow: ")
      print("special characters allowed?: {}".format(self.specialChars))
      print("required number of numerical symbols:
{}".format(self.symbolsMin))
      print("required number of characters: {}".format(self.charMin))
      print("your password is: {} \n".format(self.passwrdToChk))
# runs the checks and balances
      def assignmentPasswordCheck(self):
      if self.charCheck() == True and self.symbolsCheck() == True and
self.specialCharsCheck() == True:
            print("This password fulfills all necessary requirements!")
      elif self.charCheck() == False:
            print("This password does not have enough characters!")
      elif self.symbolsCheck() == False:
            print("This password does not contain the required amount of
symbols!")
      elif self.specialCharsCheck() == False:
            print("This password contains forbidden special characters!")
      else:
            print("something went very wrong here")

#Requirements are assigned here!

def getInputs():
      global passwordInput
      global charMinInput
      global symbolsMinInput
      passwordInput = input("please input a password: ")
      charMinInput = int(input("please input minimum number of characters:
"))
      symbolsMinInput = int(input("please input minimum number of symbols:
"))

      while True:
      specialCharsInput = input("Allow special symbols? (y/n): ")
      global specialCharsChoice
```

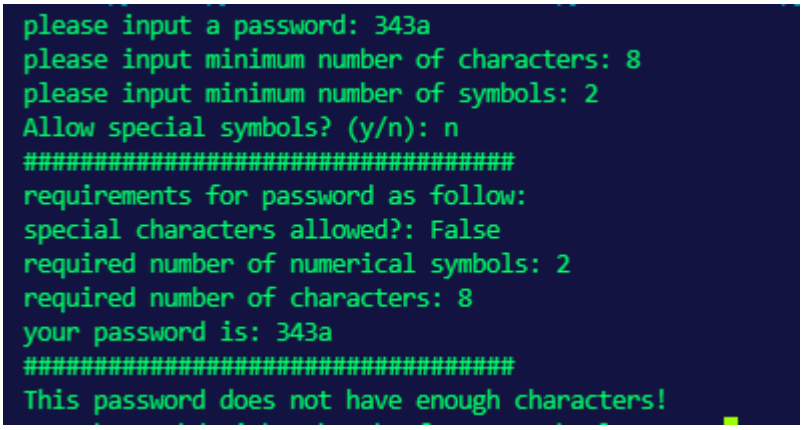
```
        if specialCharsInput == "n":
                specialCharsChoice = False
                break
        elif specialCharsInput == "y":
                specialCharsChoice = True
        else:
                print("Please input a valid selection!")


def main():
        global passwordInput
        global charMinInput
        global symbolsMinInput
        global specialCharsChoice
        getInputs()
        assignReqs = passwordChecker(charMinInput, symbolsMinInput,
specialCharsChoice, passwordInput)
        assignReqs.requirements()
        assignReqs.assignmentPasswordCheck()

main()
```

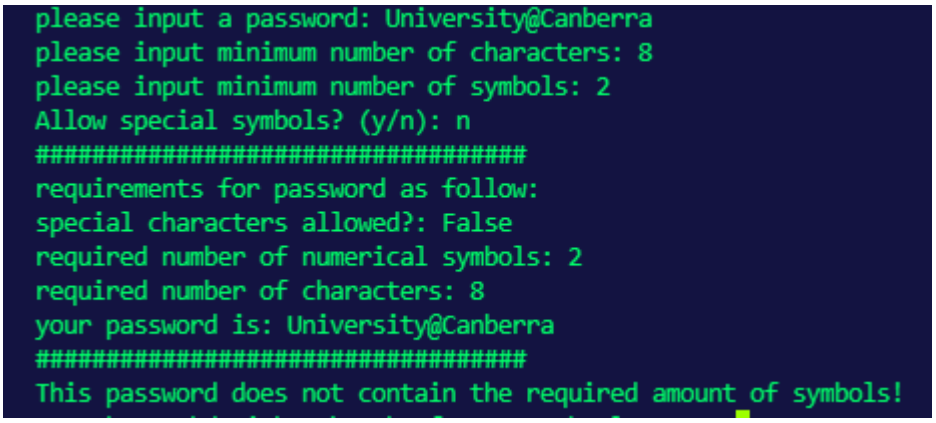
Test Plan

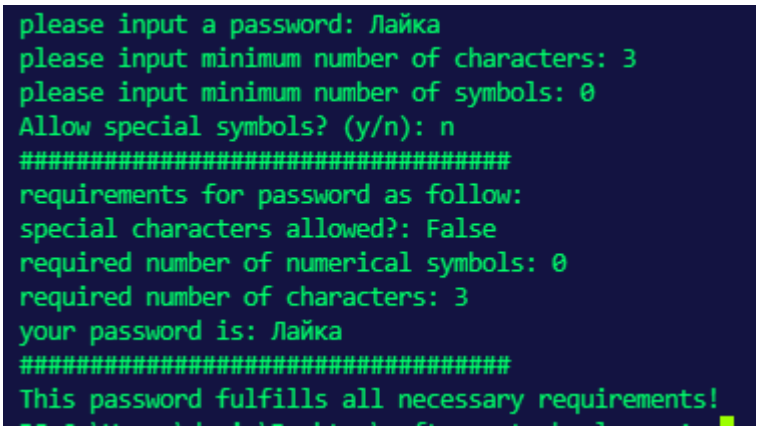
I kinda feel like a broken record at this point, but that said;
I kinda just uh put in some different inputs and see if they work?

Testing Screencaps









A note on this; it seems that checking if alphanum checks against ascii or something, in which I assume non-english characters are standard. Just good to know!

Question 5

Analysis

The goal of this question is to create a simple program that utilises the *Turtle* library for python, and to create a stop sign (red hexagon + the word “STOP” written in the middle in white). This doesn’t take any inputs, but it does output the said stop sign.

Algorithm Design

For this question I broke it up into two functions: the stop sign itself, and the lettering. For the stop sign, I utilised a for loop which iterates 6 times in total, each time for each line of the hexagon.
Then in the lettering function, it moves the pen to the correct position and renders the text. Notably, in order to generate white text one must change the colour of the pen, which is different to how shapes are coloured.
I then created a third function, main(), which serves to just contain the code for good practice. I also utilised the Screen class of the turtle library which allowed me to make the QoL (Quality of Life) change that stops the program from exiting until the window is clicked. For good measure.

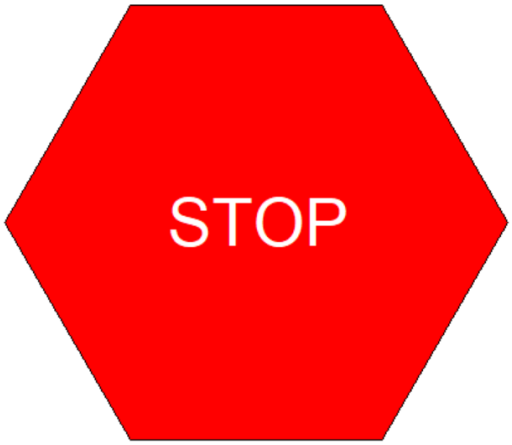
Code

```
from turtle import Turtle, Screen
screen = Screen()
turtle = Turtle()

def drawHexagon():
      turtle.fillcolor("red")
      turtle.begin_fill()
      for i in range(6):
      turtle.forward(200)
      turtle.left(60)
      turtle.end_fill()
      turtle.hideturtle()
      print(turtle.fillcolor())

def writeStop():
      turtle.penup()
      turtle.right(240)
      turtle.forward(115)
      turtle.right(100)
      turtle.forward(35)
      turtle.pendown()
      turtle.pencolor("white")
      turtle.write("STOP", font=("Comic Sans", 70, "normal"))


def main():
      drawHexagon()
      writeStop()
      screen.exitonclick()

main()
```

Testing plan

Well, there isn't really anything to test here? I mean, I did just mess around with the variables for pen movement, but that was just a part of building it. Due to the lack of inputs, there isn't really much to test, as the program would be running the exact same way every time. I guess I'll put screenshots of getting the text to the right size? I don't know if that counts as testing

First iteration




```
ef writeStop():
    turtle.penup()
    turtle.right(240)
    turtle.forward(125)
    turtle.right(100)
    turtle.forward(100)
    turtle.pendown()
    turtle.pencolor("white")
    turtle.write("STOP", font=("Comic Sans", 40, "normal"))
```

Increasing the size - first go

```
def drawHexagon():
    turtle.fillcolor("red")
    turtle.begin_fill()
    for i in range(6):
        turtle.forward(200)
        turtle.left(60)
    turtle.end_fill()
    turtle.hideturtle()
    print(turtle.fillcolor())


def writeStop():
    turtle.penup()
    turtle.right(240)
    turtle.forward(125)
    turtle.right(100)
    turtle.forward(100)
    turtle.pendown()
    turtle.pencolor("white")
    turtle.write("STOP", font=("Comi
```

BLEMS OUTPUT TERMINAL COMMENTS DEB

Change up the parameters a bit (guessing)

```
    turtle.hideturtle()
    print(turtle.fillcolor())


def writeStop():
    turtle.penup()
    turtle.right(240)
    turtle.forward(105)
    turtle.right(100)
    turtle.forward(70)
    turtle.pendown()
    turtle.pencolor("white")
    turtle.write("STOP", font=("Comi
```

Lets move it to the left a bit

```
            turtle.forward(200)
            turtle.left(60)
    turtle.end_fill()
    turtle.hideturtle()
    print(turtle.fillcolor())


def writeStop():
    turtle.penup()
    turtle.right(240)
    turtle.forward(110)
    turtle.right(100)
    turtle.forward(45)
    turtle.pendown()
    turtle.pencolor("white")
    turtle.write("STOP", font=("Comi
```

This looks good!

```
    turtle.end_fill()
    turtle.hideturtle()
    print(turtle.fillcolor())


def writeStop():
    turtle.penup()
    turtle.right(240)
    turtle.forward(115)
    turtle.right(100)
    turtle.forward(35)
    turtle.pendown()
    turtle.pencolor("white")
    turtle.write("STOP", font=("Comi
```