Python Lab Assignment 1

Table of Contents

[Program 1 1](#_Toc78982706)

[Question 1](#_Toc78982707)

[Code 1](#_Toc78982708)

[Outputs 3](#_Toc78982709)

[Normal output 3](#_Toc78982710)

[Maximum error output 3](#_Toc78982711)

[Inferences 3](#_Toc78982712)

[Program 2 4](#_Toc78982713)

[Question 4](#_Toc78982714)

[Code 4](#_Toc78982715)

[Outputs 5](#_Toc78982716)

[Normal output 5](#_Toc78982717)

[Maximum error output 5](#_Toc78982718)

[Inferences 5](#_Toc78982719)

[Program 3 5](#_Toc78982720)

[Question 5](#_Toc78982721)

[Code 5](#_Toc78982722)

[Outputs 8](#_Toc78982723)

[Normal output 8](#_Toc78982724)

[Error outputs 8](#_Toc78982725)

[Inferences 9](#_Toc78982726)

# Program 1

## Question

Write a program that asks the following information:

* Year
* Program
* Enrolment number

Using these, it generates the user's Register No. For example

**Year** = 2018

**Program** = BSC

Enrolment number = 3265

Output = 18BSC3265

## Code

from datetime import date

*# These functions allow for extra spaces in the input.*

**def** inputNumericString(promptMessage, errorMessage):

while True:

n = input(promptMessage)

if n.isnumeric():

return n

else:

print(errorMessage)

**def** inputYear(promptMessage, errorMessage):

currentYear = date.today().year

while True:

year = inputNumericString(promptMessage, errorMessage)

if int(year) >= currentYear - 1000 and int(year) <= currentYear + 1000:

return year

print(errorMessage)

**def** processProgramName(program):

processedProgramName = str()

for i in program:

if (i >= "a" and i <= "z") or (i >= "A" and i <= "Z"):

processedProgramName = processedProgramName + i.upper()

return processedProgramName

**def** inputProgramName(promptMessage, errorMessage):

while True:

programName = input(promptMessage)

programName = processProgramName(programName)

if len(programName) >= 2:

return programName

print(errorMessage)

print("\nENTER YOUR DETAILS\n")

year = inputYear("Year of admission: ", "Invalid year entry!")

programName = inputProgramName("Selected program: ", "Invalid program name!")

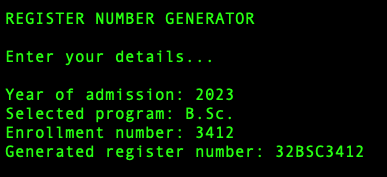
enrolmentNumber = inputNumericString("Enrollment number: ", "Invalid enrollment number!")

registerNumber = year[-1] + year[-2] + programName + enrolmentNumber

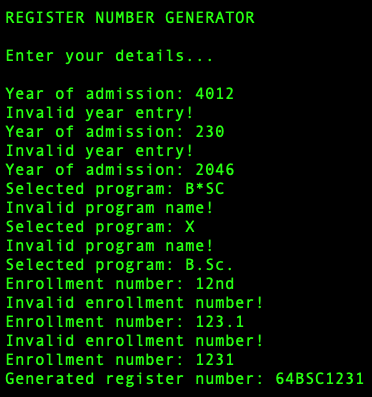
print("Generated register number: " + registerNumber + "\n")

## Outputs

### Normal output



### Maximum error output



## Inferences

The existence of the "len" function often removes the need for flag variables when dealing with lists that update with a loop's progression.

Definition of functions for different tasks makes the code more modular and easier to understand, since different functionalities of the code are neatly organised and labelled by the functions.

Strings in Python have an advantage over strings in both Java and C, since they combine the best of both languages. You can easily assign and operate on strings, as in Java, since they are processed as individual objects. You can also refer to and change individual characters in a string through indices, as in C, since strings are also processed as arrays of characters.

# Program 2

## Question

Write a Python program to store Pan number of each employee, and check whether the pan is valid or not. The PAN is a 10-char long alpha-numeric unique identifier, with the following conditions...

* The first five characters are letters
* The 5th character denotes the holder type
* The next four are are numerals
* The last character is a letter

## Code

print()

while True:

pan = input("Enter your PAN for format validation: ")

if len(pan) == 10:

break

print("PAN number must be 10 characters long!")

*# Note that the upper bound is not included in a slice*

part1 = pan[0:4].upper()

part2 = pan[4].upper()

part3 = pan[5:9]

part4 = pan[9].upper()

*# ERROR CHECKING*

errorList = []

holderTypes = ("A", "B", "C", "F", "G", "H", "L", "J", "P", "T")

for i in part1:

if i < "A" or i > "Z":

errorList.append("1st 4 characters must be letters.")

break

if part2 not in holderTypes:

errorList.append("Unrecognised holder type denoted by 5th character.")

if not part3.isnumeric():

errorList.append("Characters 6 to 9 must be numeric.")

if part4 < "A" or part4 > "Z":

errorList.append("Last character must be a letter.")

print()

pan = part1 + part2 + part3 + part4

if len(errorList) == 0:

print("Your PAN, " + pan + ", is in valid format.")

else:

print("Your PAN has the following issues:")

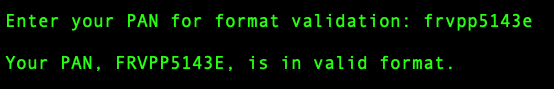
for i in errorList:

print("- " + i)

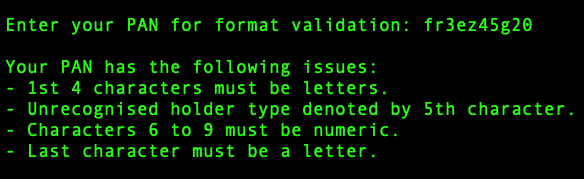
print()

## Outputs

### Normal output



### Maximum error output



## Inferences

While it is not necessary to assign different parts of the PAN number to different string variables, it makes the logic of the program much clearer. Also, converting string that is supposed to be alphabetic to upper case makes it easier to check whether it is alphabetical, and (for the 5th character) whether the letter is present in the valid account holder types.

# Program 3

## Question

Create a list and store the ID numbers, names, mobile numbers of multiple employees.

The correct name format is first name space and then last name (ex. Gautam Menon). Check the validity of the name and mobile number. A valid mobile number has 10 digits, and the first digit non-zero. Print 'Invalid input' when conditions are not satisfied. For invalid inputs, break the process of getting input. Display the results (stored values) in a list.

**Note:** Separate lists for ID, name, mobile number.

## Code

**def** nameInput(prompt):

name = input(prompt) + "\*"

nameParts = [] *# Stores each part of the name separated by spaces*

i = 0 *# Iterating variable*

checkpoint = 0 *# Marks the lower bound to slice the original name string*

while name[i] != "\*":

while name[i].isspace(): i = i + 1

checkpoint = i

if name[i].isalpha():

while name[i].isalpha(): i = i + 1

else:

break

nameParts.append(name[checkpoint:i])

if name[i] == "\*" and len(nameParts) == 2:

return nameParts[0] + " " + nameParts[1]

return "\*"

*# Note:*

*# Since the len function is available, I used it here.*

*# Alternatively, I could count the number of iterations*

*(to keep count of the number of name parts).*

**def** mobileNumberInput(prompt):

mobileNumber = input(prompt) + "\*"

i = 0

digitCount = 0

finalMobileNumber = str()

while digitCount < 10 and mobileNumber[i] != "\*":

while mobileNumber[i].isspace(): i = i + 1

if mobileNumber[i].isnumeric():

while mobileNumber[i].isnumeric() and digitCount < 10:

i = i + 1

digitCount = digitCount + 1

finalMobileNumber = finalMobileNumber + mobileNumber[i]

else:

digitCount = 11

if digitCount == 10 and finalMobileNumber[0] != "0":

return finalMobileNumber

return "\*"

i = 0

*# STORAGE LISTS*

ids = []

names = []

mobileNumbers = []

*# INFORMATION*

print("\nEMPLOYEE LIST CREATION PROGRAM")

print("Enter names and mobile numbers of employees.")

print("Name must have only a distinct first and last name, with no special characters.")

print("Mobile number must be 10 digits, and first digit must not be 0.")

print("Enter in a wrong input format to end input process.")

*# INPUT LOOP*

while i < 100:

i = i + 1

print("\nENTER EMPLOYEE", i, "DETAILS")

name = nameInput("Name\t: ")

if name == "\*":

break

mobileNumber = mobileNumberInput("Mobile\t: ")

if mobileNumber == "\*":

break

ids.append(i)

names.append(name)

mobileNumbers.append(mobileNumber)

print("Invalid input!") *# The loop only ends with a wrong input :)*

*# DISPLAY OF DATA*

print("------------------------------")

if len(ids) > 0:

print("EMPLOYEE LIST DISPLAY")

for i in ids:

print("\ID:", i)

print("Name\t: " + names[i - 1])

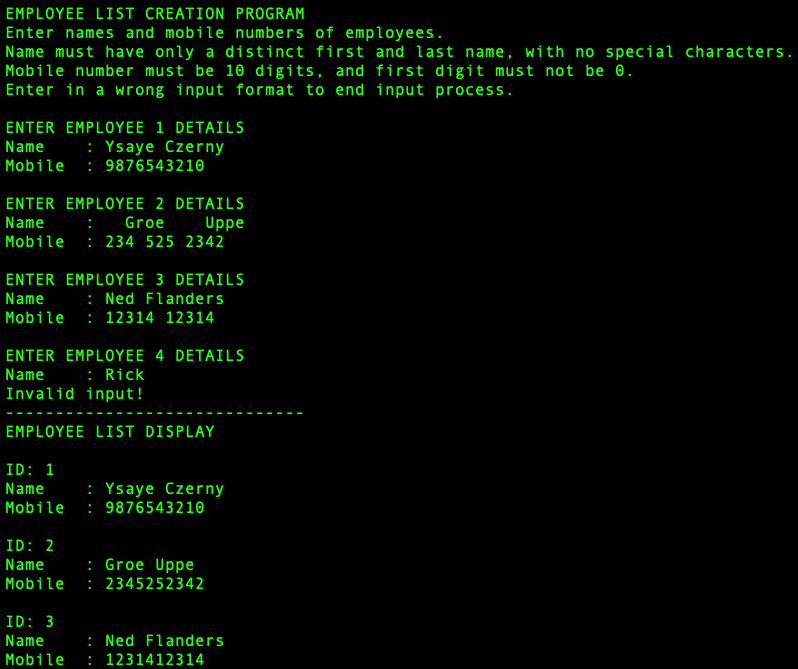
print("Mobile\t: " + names[i - 1])

else:

print("Nothing to display!\n")

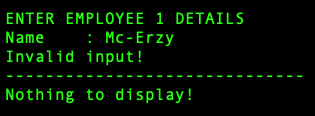
## Outputs

### Normal output

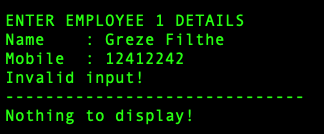


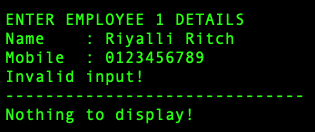
### Error outputs

#### Name entry error



#### Mobile number entry error





## Inferences

Usability is increased if you accommodate extra space or tab inputs, which is why there are loops to go over extra spaces or tabs in every function that processes the inputted data. Also, in this question, since the input process was supposed to end when the data is inputted in a wrong format, I specified that, so that the user can deliberately enter a wrong input format to end the input process whenever the user wants.