Python Lab Test 1

# Program 1: Caesar cipher

## Code

*# Main functions...*

**def** offsetChar(c, key):

x = ord(c) + key

if c >= "A" and c <= "Z": max, min = ord("Z"), ord("A")

elif c >= "a" and c <= "z": max, min = ord("z"), ord("a")

if x > max: return chr(min + x - max - 1)

elif x < min: return chr(max - (min - x) + 1)

return chr(x)

**def** encrypt(original, key):

try: key = int(key)

except: return "Invalid key!"

encrypted = ""

for c in original: encrypted = encrypted + offsetChar(c, key)

return "Encrypted message:\n" + encrypted

**def** decrypt(encrypted, key):

try: key = int(key)

except: return "Invalid key!"

original = ""

for c in encrypted: original = original + offsetChar(c, -key)

return "Original message:\n" + original

*#------------------------*

*# Other functions...*

**def** sectionTitle(title): print("------------\n" + title.upper() + "\n\_\_\_\_\_\_")

**def** e():

sectionTitle("encrypter")

x, key = input("Input string to enrypt: "), input("Input key: ")

print(encrypt(x, key))

**def** d():

sectionTitle("decrypter")

x, key = input("Input string to decrypt: "), input("Input key: ")

print(decrypt(x, key))

**def** q():

print("Goodbye!\n")

exit()

**def** h():

print("Enter e for encryption.")

print("Enter d for decryption.")

print("Enter x for exit.")

**def** switch(option):

try: return {"e":e, "d":d, "x":q, "?":h}[option]

except: print("Invalid key!")

*#------------------------*

*# Main program*

print("\nCAESAR ENCRYPTION AND DECRYPTION")

print("(Enter ? for help)")

while True: switch(input(">> "))()

## Output



## Inferences

**ord** and **chr** are functions used to deal with characters with respect to their ASCII values, which allows us to deal with characters like integers.

The **offsetChar** function is used to account for character additions with the key that spill over the limits (lower limit being "a" or "A" and upper limit being "z" or "Z"). This function helps "wrap" the cipher-related additions and subtractions within the alphabet.

# Program 2: DOB checker and voter eligibility checker

## Code

from datetime import date

**def** checkDate(d):

x = d.split("-")

try:

dd, mm, yyyy = int(x[0]), int(x[1]), int(x[2])

dob = date(yyyy, mm, dd)

return dob

except: return False

**def** votingEligibility(dob):

dob = checkDate(dob)

if not dob: return -1 *# If the DOB is not in correct format.*

cur = date.today() *# Current date*

*# The following code ensures that the person's 18th birthday is today or has passed...*

yearDifference = cur.year - dob.year

monthDifference = cur.month - dob.month

dayDifference = cur.day - dob.day

if yearDifference > 18: return 1 *# Person is 19 or above.*

elif yearDifference == 18 and monthDifference > 0: return 1 *# Today's month is after birthday month.*

elif yearDifference == 18 and monthDifference == 0 and dayDifference >= 0: return 1 *# Today's day is birthday or after birthday (in the birthday month).*

return 0

print("\nDate format: dd-mm-yyyy")

dob = input("Enter DOB: ")

check = votingEligibility(dob)

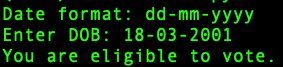
if check == -1: raise Exception("Wrong date format!")

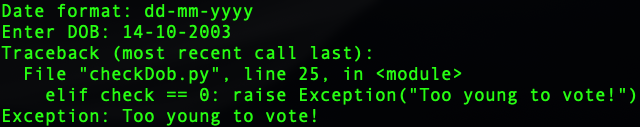
elif check == 0: raise Exception("Too young to vote!")

else: print("You are eligible to vote.")

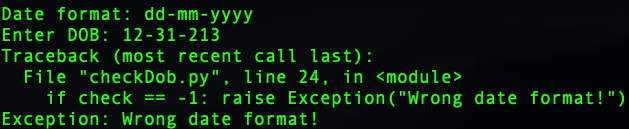
## Outputs

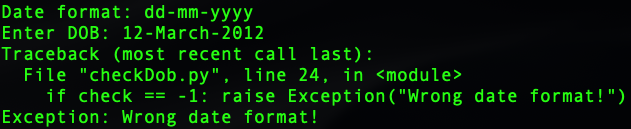
### Proper date given...





### Wrong date given...





## Inferences

I converted the string to a date object so that it would be

1. Easier to check for proper date format.
2. Easier or more efficient to perform other date-related operations and calculations.

However, for this program, such a conversion is not strictly necessary.