```
run;  print (10 / 2)   ---> / float sayi elde eder. Eger int elde etmek istiyorsan // kullanmalisin.
answer; 5.0

print(*"Mobilhanem",sep=".")
M.o.b.i.l.h.a.n.e.m



isim = input("İsminiz :")
yas = input("Yaşınız :")

print("Merhaba {} bey yaşınız {} hala çok gençsiniz".format(isim,yas))



run; print (10 + 5.0)
answer; 15.0          ---> int ile float beraber isleme konulabilir.


run; print (2 ** 4)
answer; 16           ---> ** isareti us almaya yarar.

run; print (2 ** 3 ** 2)
answer; 512              ---> oncelik us te oldugu icin once 3 ** 2 u yapar.

run; print (20 % 6)
answer; 2           ---> % isareti mod alir.

run; print ("deniz pinar")
     print ('deniz pinar')
answer; deniz pinar         ---> '  ile " arasinda bir fark yok.
       deniz pinar


run;
   print ('deniz\'in arabasi')

answer;
   deniz'in  arabasi                    ---> ' isaretini cumlenin icerisinde yazdirabilmek icin backslash
kullanmalisin.


run;
   print ('one \n two \n three')
answer;
   one
   two
   three                 ---> \n alt satira gecmesini saglar.


run;
   print ('one \t two \t three')
answer;
   one     two     three        ---> \t tab yapar bosluk birakip yazar.
```

```
run;
    print (""" deniz
            pinar
                geliyoooo""")
answer;
    deniz
 pinar
    geliyoooo                  ---> """ kullanilirsa icine nasil yazildiysa oyle cikti dondurur. newline demene
gerek kalmaz.


run;
    print ("deniz" + "pinar")
answer;
    deniz pinar              ---> + concat yapar.


run;
    print ("deniz" * 3)
answer;
    denizdenizdeniz      ---> * stringler icin kullanildiginda tekrarlama yapar.


run;
    deniz-pinar = "deniz pinar"
    print (deniz-pinar)
answer;
    error...                   ---> variable ismi olarak numbers,letters and underscores kullanilabilir sadece.
ve variable ismi sayiyla da baslayamaz.


run;
    x = input()
answer;
    no output            ---> kullanicidan bir input alir.kullanici input olarak sayi bile girse her turlu string
e donusur.ama int(input()) olarak yazarsan sayi olarak kullanabilirsin.


run;
    x += 3   demek aslinda x = x + 3 demektir.bu kullanima in place operators denir.


run;
    True or False buyuk harfle baslamali.


==    ---> esitse...
!=    ---> esit degilse...


run;
    print (7 < 5)
answer;
    False


run;
    print ("annie" > "andy")
```

answer;
    False   ---> a=a, n=n, n>d ise false

run;
    if (10<5):
        print("10 is greater than 5")
 print("yes")   ---> yes ifadesi if condition i dogru olsa da olmasa da yazilir. cunku if in icinden sayilmaz(indentation)
answer;
    yes

run;
    x=4
    if (x == 5):
        print ("yes")
    else:
        print ("no")

answer;
    no   ---->else kullanimi...

run;
    num = 3
    if (num == 1):
        print ('one')
    elif (num == 2):
        print ('two')
    elif (num == 3):
        print ('three')
    else:
        print ('something else')

answer;
    three   ---->else if yazacaksan elif yazarsin.

run;
    print (1==1 and 2!=3)
    print (1==1 or 2==3)
    print (not 1<5)
answer;
    True
    True
    False   ---> and, or, not boolean ifadeler bu sekilde kullanilir. == in boolean ifadelere islem onceligi vardir. parantez> carpma/bolme >toplama/cikarma.

run;
    words = ['deniz' , 'pinar' , '!']
    print (words[0])
answer;
    deniz   ---> diziler koseli parantezle ve ayrica normal parantezle de kullanilabilir. mesela words = ('deniz' , 'pinar') da olur.

```
run;
        m = [
                [1,2,3],
                [4,5,6]
            ]
        print ( m[1][2] )
answer;
        6                   ---> m nin yanindaki 1 satir sayisi, 2 sutun sayisini belirtir.


run;
        str = “hello world”
        print ( str[6] )
answer;
        w                       ---> stringin icerisinden bir karakter secmek mumkun


run;
        nums = [1, 2, 3]
        print(nums + [4, 5, 6])
        print(nums * 3)

answer;
        [1,2,3,4,5,6]
        [1,2,3,1,2,3,1,2,3]                 ---> iki diziyi birlestirip tek bir dizi elde etmek.


run;
        words = ["spam", "egg", "spam", "sausage"]
        print("spam" in words)
        nums = [1, 2, 3]
        print(not 4 in nums)
answer;
        True                        ---> bir dizi icerisinde bir eleman aramak.
        True


run;
        nums = [ 1,2,3]
        nums.append (4)

answer;
        [1,2,3,4]                   ---> bir listenin sonuna yeni bir eleman ekleme. append gibi
fonksiyonlari kullanirken () parantez kullanilmalidir. string olsaydi nums , append ve insert
komutunu kullanamazdin. string nedir? mesela nums=’dsds’ bir stringdir.


run;
        nums = [1, 3, 5, 2, 4]
        print(len(nums))

answer;
        5                   ---> bir dizinin kac elemani oldugunu bulmak


run;
        words = ['deniz' , 'pinar']
```

```
        words.insert(1,'coook')
        print (words)
answer;
        [‘deniz’ , ‘coook’ , ‘pinar’]                    ---> insert ile bir dizinin herhangi bir yerine bir eleman ekleyebiliriz. append ile farki index verebilmemizdir.


run;
        letters = ['p', 'q', 'r', 's', 'p', 'u', ‘q’]
        print(letters.index('r'))
        print(max(letters)
        print(min(letters)
        print(letters.count(‘q’)
        letters.remove(‘q’)
        print(letters)
        letters.reverse()
        print (letters)
answer;
        2                       ---> index, max, min, count(kac tane oldugunu sayar), reverse, remove.
        u
        p
        2
        [‘p’,’r’,’s’,’p’,’u’,’q’]
        [‘q’,’u’,’p’,’s’,’r’,’p’]


run;
        i = 0
        while i <5:
        print(i)
        i +=1

answer;
        0
        1
        2
        3
        4                               --->while loop.


run;
        i = 0
        while i <5:
          print(i)
          i +=1
          if i==3:
            break


answer;
        0
        1
        2                       ---> break.


run;

i = 1
```

```
while i<=5:
    print(i)
    i+=1
    if i==3:
      i+=1
      continue
```

answer;
1
2
4
5 ---> continue while loop un basina tekrar dondurur.

```
run;
        words = ['1','2','3']
        for word in words:
          print(word + "!")
```

answer;
1!
2!
3! ---> for loop ozellikle diziler icin kullanilir. for loop ile while loop arasindaki fark su aslinda ikisi de ifadeyi tekrar tekrar execute etmeye yarar. ama for loop iterasyon sayisi bilindiginde kullanilirken while bilinmedigi zamanda kullanilir.

```
run;
        numbers = list(range(10))
        print(numbers)
```

answer;
[0,1,2,3,4,5,6,7,8,9] ---> range(10) sayisi 0-10 arasindaki sayilari alir, range iki parametre de alabilir,hatta 3 parametre de alabilir, list ise bu sayilarii liste yapar. mesela list(range(3,8,2)) = [3,5,7]

DRY principle (dont repeat yourself)
WET principle (write everything twice)
Kod yazarken tekrardan kacinmali .Yukarida yazdigim DRY principle bunu anlatiyormus.

```
run;
def my_func():
  print("spam")
my_func()
```
---> define ile fonksiyon olusturduk ve cagirdik.

```
run;
        def max(x, y):
            if x >=y:
                return x
            else:
                return y

        print(max(4, 7))
```
answer;

7 ---> return ve def kullanimi. return den sonra yazilan kodlar execute edilmez.

comment yazmaya yarar.

“”” 3 tane cift tirnak multiline comment yazmaya yarar.

run;

```
def multiply(x, y):
 return x * y

 a = 4
 b = 7
 operation = multiply
 print(operation(a, b))
```

answer;

28 ---> birden fazla fonksiyonu ic ice yazabiliriz boylece. operation yerine deniz yazsaydim da olurdu.

run;

```
import random
for i in range(5):
 value = random.randint(1, 6)
print(value)
```

answer;

5
1
3
4
2 ---> cevap 1-6 arasindaki sayilari rastgele siralayan randint fonksiyonunu kullandik. ama bunun icin random modulunu import etmemiz gerekti. baska math.sqrt

run;

```
from math import pi
print(pi)
```

answer;

3.14... ---> pi sayisini math modulunden import ettik. from math import * yazsaydik math modulundeki tum fonksiyonlari kullanabiliriz.

run;

```
from math import sqrt as square_root
print(square_root(100))
```

answer;

10 ---> bir modul icerisindeki fonksiyonu baska bir isimle import edip kullanmak icin.

pip install library_name. ---> pip install komutlariyla istedigimiz modulu indirebiliriz.

exceptions ---> aslinda aldigimiz hatalardir. TypeError, ValueError, IndexError gibi...
ImportError: an import fails;
IndexError: a list is indexed with an out-of-range number;

NameError: an unknown variable is used;
SyntaxError: the code can't be parsed properly;
TypeError: a function is called on a value of an inappropriate type;
ValueError: a function is called on a value of the correct type, but with an inappropriate value.

run;
```
try:
    num1 = 7
    num2 = 0
    print (num1 / num2)
    print("Done calculation")
except ZeroDivisionError:
    print("An error occurred")
    print("due to zero division")
finally:
    print(‘no matter print this”)
```

answer;
An error occurred
due to zero division ---> try except ile kodun hangi hatadan dolayi error verdigini buluruz. finally ise error alinsa bile calisir her zaman...

run;
```
print(1)
raise ValueError
print(2)
```
--->raise ValueError(“you entered a number”) gibi aciklama da eklenebilir

answer;
ValueError ---> raise ise hic hata olmadiginda bile hata dondurmeye yarar. eger raise i except icin icine yazarsan hic bir arguman vermeden de yazabilirsin. sadece raise yazman yeterli.

run;
```
assert 2 + 2 == 4
print(2)
assert 1 + 1 == 3
print(3)
```
-->assert (4 < 0), "Colder than absolute zero!" gibi aciklama ok.

answer;

AssertionError ---> assertion ile programimizi test ederiz, eger testi gecemezse diger adimlari calistirmadan error verir.

run;
```
myfile = open("filename.txt")
# write mode
open("filename.txt", "w")

# read mode
open("filename.txt", "r")
```
---> parantez icine directory tam verilmeli. open default read yapar.

```
        open("filename.txt")

# binary write mode
        open("filename.txt", "wb")              ---> text formatta degilse binary mode da acilir
dosya.
        file.close()                            ---> dosya bir kere acildiginda kapatmayi unutmamalidir.

        file = open("filename.txt", "r")
        cont = file.read()
        print(cont)
        file.close()                --> read boyle de kullanilabilir. dosyayi bir kere read ettikten sonra
close demeden  bir daha read edersen bos gorunur.

        file = open("filename.txt", "r")
        print(file.read(16))                    ---> sayi vererek kac byte(karakter) print
edecegimizi belirtebiliriz.
        file.close()


        file = open("filename.txt", "r")
        print(file.readlines())                 ---> readlines herseyi tek satira toplar.
        file.close()



        file = open("filename.txt", "r")

        for line in file:                       ---> for loop ile teker teker satirlari okumak.
            print(line)

        file.close()



        file = open("newfile.txt", "w")         ---> eger bu dosya yoksa bile w ile olusturulur.
        file.write("This has been written to a file")
        file.close()                            ---> write ile parantez icindeki ifade dosyaya yazilir.
dosyayi w modunda acarsan icindekiler silinir. DIKKAT!


        run;
                msg = "Hello world!"
                file = open("newfile.txt", "w")
                amount_written = file.write(msg)
                print(amount_written)
                file.close()                    ---> amount.written ile kac byte(karakter) dosyaya
                                                        yazildigini goruruz.deniz de koysak olurdu
        answer;
                12


        try:
          f = open("filename.txt")
          print(f.read())
        finally:
          f.close()             ---> dosya ile islem yapilacaksa bu yontemin kullanilmasi cok
        yerindedir. boylece veri kaybetme riski sifira iner. finally ile ne olursa olsun dosyayi close
        etmis oluruz.
```

```
with open("filename.txt") as f:
  print(f.read())
```
---> with ile filename dosyasini f olarak acar ve filename dosyasini otomatik olarak kapatir. boylece security saglanmis olur.

Dictionaries data structure lara bir key degeri atar. Bu degerin degistirilemez olmasi gerekir.(string gibi mi?) Suana kadar degistirilebilir olarak gordugumuz sey; lists and dictionaries.
Aslinda listeler de belli int key lere atanan bir dictionary lerdir.

```
liste =[2,"Python",5.4,[5,3],("Java",5,'a')]
,liste[4][0] =
Java
,liste[0:5:2] =
[2, 5.4, ('Java', 5, 'a')]
```

```
run;
ages = {"Dave": 24, "Mary": 42, "John": 58}
print(ages["Dave"])
print(ages["Mary"])

answer;
24
42
```
--> isimler ile yaslari pair yaptik aslinda. Eger dictionary de olmayan bir isim aratirsan KeyError alirsin. (isim = key, yas = value) (key:value) Eger key liste olursa TypeError alirsin.

eğer tek elemanlı bir veri türü bir **Tuple** oluşturacaksanız elamanın sonuna bir virgül koymalısınız yoksa yorumlayıcı bu türü **String** olarak alacaktır. Şu şekilde kullanım yanlıştır;
tupple = ‘hello world’
dogrusu soyle olmalidir;
tupple = ‘hello world’, yani sonunda virgul olmalidir.

"one two three" ---> bu bir liste degildir. Tuple dir.

Tuple ile listenin tek farki tuple in immutable(degistirilemez olmasidir)

words = **(**"spam", "eggs", "sausages",**)** **bu da bir tuple dir.**

```
run;
squares = {1: 1, 2: 4, 3: "error", 4: 16,}
squares[8] = 64
squares[3] = 9
print(squares)

answer;
{1: 1, 2: 4, 3: 9, 4: 16, 8: 64}
```
---> dictionary ye yeni bir key:value eklemek veya degistirmek icin bu yapilabilir.

```
run;
pairs = {1: "apple",
   "orange": [2, 3, 4],
```

```
    True: False,
    None: "True",
}

print(pairs.get("orange"))
print(pairs.get(7))
print(pairs.get(12345, "not in dictionary"))

answer;
[2, 3, 4]
None
not in dictionary                    ---> get metodu dictionary de index dondurme icin guzel bir yontemdir. Son ifadede 12345 dictionary de olsaydi, not in dictionary ifadesi yazilmayacakti sonucta.

run;
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[2:6])
print(squares[3:8])
print(squares[0:1])

answer;
[4, 9, 16, 25]
[9, 16, 25, 36, 49]
[0]


run;
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[:7])
print(squares[7:])                   ---> eger birsey yazmazsan en bastan alir veya en sondan alir.

answer;
[0, 1, 4, 9, 16, 25, 36]
[49, 64, 81]


run;
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[::2])
print(squares[2:8:3])          ---> ucuncu ifade kacar kacar indexi artiracagini soyler.

answer;
[0, 4, 16, 36, 64]
[4, 25]


run;
sums = [1,2,3,4,5]
print (sums[::-1])

answer;
[5, 4, 3, 2, 1]          --> negatif deger koyarsan sondan basa dogru gider.


run;
cubes = [ i**3 for i in range(5) ]
```

```
print(cubes)

answer;
[0, 1, 8, 27, 64]


run;
cubes = [i**3 for i in range(5) if i**3 % 2 == 0]
print(cubes)

answer;
[0, 8, 64]

run;
nums = [4,5,6]
msg = 'numbers: {0} {1} {2}'.format(nums[0], nums[1], nums[2])

answer;
numbers: 4 5 6              --->nums adli liste ile msg adli tuple i bilerstirmek icin format.{}-->position
format listeyi tuple a cevirmeye yariyor.

run;
 a = "{x}, {y}".format(x=5, y=12)
print(a)

answer;
5, 12

run;
print( '*' .join (['spam', 'eggs' , 'ham' ]))
print('hello Me' . replace ('Me' , 'world'))
print('hello Me' . startswith('hello'))
print('hello Me' . endswith('Me'))
print('hello Me' . upper())
print('helLo Me' . lower())
print(max(1,2,3,4))
print(min(1,2,3,4))
print(abs(-4))
print(sum([-4,3,5]))

answer;
spam*eggs*ham
hello world
True
True
HELLO ME
hello me                              ---> bunlara built-in functions deniyor.
4
1
4

run;
nums = [55,44,33,22,11]
if all([i>5 for i in nums]):
    print("All larger than 5")

answer;
```

All larger than 5 ---> all ile tek tek nums dizisindeki tum elemanlari inceler ve hepsi icin sonuc dogruysa print eder.

```
run;
nums = [55,45,33,22,11]
if any([i%2 == 0 for i in nums]):
    print("At least one is even")
```

answer;
At least one is even ---> any ile dizideki elemanlardan bir tanesi bile dogru olsa print eder.

```
run;
nums = [55,45,33,23,11]
for v in enumerate(nums):
    print(v)
```

answer;
(0, 55)
(1, 45)
(2, 33)
(3, 23)
(4, 11) ---> enumerate = numaralandirmak demektir index atayabiliriz.

```
run;
def count_char(text, char):
  count = 0
  for c in text:
    if c == char:
      count += 1
  return count

print(count_char("deniddz" , 'd'))
```

answer;
3 ---> bir text in icinde bir karakterin kac kere yer aldigini boyle bulabiliriz.

```
run;
for char in "abcdefghijklmnopqrstuvwxyz":
  perc = 100 * count_char(text, char) / len(text)
  print("{0} - {1}%".format(char, round(perc, 2)))
```

answer;
a - 3.86%
b - 1.12%
c - 1.12%
d - 1.97%
e - 7.58%
f - 2.29%
g - 1.08%
h - 2.18%
i - 4.69%

j - 0.0%
k - 0.16%
l - 1.82%
m - 1.19%
n - 4.04%
o - 3.93%
p - 1.61%
q - 0.52%
r - 4.15%
s - 4.13%
t - 5.25%
u - 2.18%
v - 1.08%
w - 0.76%
x - 1.52%
y - 0.45%
z - 0.0% ---> dosyadaki her harfin kac kere kullanildiginin yuzdesini cikariyor.

run;
```
def apply_twice(func,arg):
    return func(func(arg))

def add_five(x):
    return x + 5

print(apply_twice(add_five, 4))
```

answer;
14 ---> higher order functions ya arguman olarak fonksiyon alirlar ya da sonuc olarak fonksiyon dondururler. functional programming pure functionlarin kullanilmasi gerektigini belirtir. Cunku pure fonksiyonlarin side effectleri yoktur ve bir deger sadece kendi argumanina baglidir. Pure fonksiyonlarda input ayni oldugu surece sonuc da ayni olur.

run;
```
some_list = []

def impure(arg):
  some_list.append(arg)
impure(1)
print(some_list)
```

answer;
[1] ---> impure fonskiyon orenegi, fonksiyon her cagrildiginda ayni inputu koymamiza ragmen farkli outputlar aliyoruz.

Pure functions are:
- easier to reason about and test.
- more efficient. Once the function has been evaluated for an input, the result can be stored and referred to the next time the function of that input is needed, reducing the number of times the function is called. This is called memoization.

run;
```
def my_func(f, arg):
```

```
  return f(arg)

print(my_func(lambda x: 2*x*x, 5))
```

answer;
50 ---> lambda ile basit isimsiz fonksiyon tanimlari yapabiliriz. ama def daha iyi cunku lambda ile sadece tek bir tanim yapabiliyoruz.

run;

```
def add_five(x):
    return x + 5

nums= [11,22,33,44,55]
result = list(map(add_five, nums))
print(result)
```

answer;
[16, 27, 38, 49, 60] ---> map (bir higer order function) ile iterative bir sekilde fonk uygulamasi yapabiliriz.

run;

```
nums =[11,22,33,44,55]
res = list(filter(lambda x: x%2==0, nums))
print(res)
```

answer;
[22, 44] ---> filter ile dizinin elemanlarini filtreleyebiliriz.

run;

```
def countdown():
    i=5
    while i >0:
        yield i
        i-=1
for i in countdown():
    print(i)
```

answer;
5
4
3
2
1 ---> The yield statement is used to define a generator, replacing the return of a function to provide a result to its caller without destroying local variables.Due to the fact that they yield one item at a time, generators don't have the memory restrictions of lists.
In fact, they can be infinite! Using generators results in improved performance, which is the result of the lazy (on demand) generation of values, which translates to lower memory usage. Furthermore, we do not need to wait until all the elements have been generated before we start to use them.

run;

```
deniz = [1,2,3,4]
deniz.pop(2)
print(deniz)
```

answer;
[1, 2, 4] -->Belli bir konumdaki elemanın değerini verir ve listeden siler. Argüman almazsa, son elemanı siler.

run;
```
deniz = [1,7,6,4]
deniz.sort()
print(deniz)
deniz.sort(reverse =True)
print(deniz)
```

answer;
[1, 4, 6, 7]
[7, 6, 4, 1]

sets = unordered data, no duplicate(curly braces ile yazilir) her cagirdiginda farkli siralanmis sonuc alirsin. set ile dictionary farkli seylermis. bos bir set ayarlamak icin su girilmeli; empty_set = set() eger {} bu sekilde yaparsan dictionary olur bos bi. dictionary bize key value hakki tanir. dictionaries is mutable.
tuple = sequential data(immutable)
list = [‘d’ , ‘df’] listenin index vererek herhangi bir elemanina erisilebilir
--> course = [‘dd’ , ‘dg’ , ‘dfdd’]
course_2 = [‘deb’ , ‘fdbfg’]
course.extend(course_2) -->iki listeyi birlestirip tek bir liste yaparken extend kullanilir.

string e cevrilen liste koseli parantez olmadan yazilir. join komutu listeyi string e ceviriyor. split komutu da stringi listeye ceviriyor.
run;
```
 course = ['ddd' , 'ddd']
course_deniz= ','.join(course)
print(course_deniz)
```

answer;
ddd,ddd ---> sonuc string oldu.

run;
```
liste= "python, c, java"
print(liste.split(","))
```

answer;
['python', ' c', ' java'] --> string listeye donustu.

```
for i in range(25,-1,-1):
        if occurrences[i] >= best_res:
        best_char = chr(ord('a') + i)
        best_res = occurrences[i]
```

```
kelime=input("Bir kelime girin :")
sozluk ={"Computer":"Bilgisayar",
```

```
            "Driver":"Sürücü",
            "Memory":"Hafıza",
            "Output":"Çıktı",
            "Software":"Yazılım",
            "Printer":"Yazıcı"}
print(sozluk.get(kelime,"Aradığınız kelime Sözlük içinde bulunmamaktadır"))



sayilar ={"1":"bir","2":"iki","3":"üç","4":"dört","5":"beş"}

print(sayilar.setdefault("8", 'sekiz'),)    // boylece yeni bir deger eklemis olduk disctionary ye


liste_1 ={"Ali":70,"Mehmet":50,"Kemal":60,"Mustafa":75}


liste_2 ={"Ali":80,"Mehmet":60,"Kemal":70,"Mustafa":85}

liste_1.update(liste_2)                         // boylece listeyi update etmis olduk


bos kume tanimlamak icin(kume ile dictionary farkli seyler);
kume = set()


Set(Küme) aynı Sözlükler gibi sıralı veri tipi değildir. Bu yüzden indexleme desteklemez.


metin ="Python"
print(metin.zfill(8))

00Python

https://www.mobilhanem.com/python-karakter-dizileri/?__cf_chl_jschl_tk__=7d51d4f8dc1cfc8b80f45d2d45883ff2ca423393-1612544108-0-AZZi0gLi9EOzcG8UFxpFQAWwauOKdGlM8qSbh1i0UIiDBOeow3XeKCePCFVgnyflY6GVFTI-YQPDYp4tkFhYHsXC8ZDl52NTQ2peI484dvkL4Oh7wKJMy0tMgXRlGB7rHD7YUI2x8K6nF3gBmfvL7KWfP_iJ970l-4CR363PgkYOFXEjSw31tA7sXn8ZmQ4lsCq6eIFdL2dPWHcvN9Vxy53szAmpBzyRy_fw-WtJM-Q3RdyFVxNV2uRXPe7QDlzvemirDTnxzAncmGh0024fAGMZoaiggV8hcJRslVBfoBTFVNDTcEATB7zHEODqwjXePpiHv-e4pyXXUj3ftcoKc_A


kaynak="abcçdefgğhıijklmnoöprsştuüvyz"
hedef ="çdefgğhıijklmnoöprsştuüvyzabc"
tablom=str.maketrans(kaynak,hedef)
print(hedef.translate(tablom))
sonuc;
fgğhıijklmnoöprsştuüvyzabcçde            ????


no = [23,56,87,47,12,36,45,47]
isim=["Ahmet","Mehmet","Ayşe","Zeynep","Elif","Kemal","Fatma","Can"]

zip1 =zip(no,isim)
x =list(zip1)
print(x[1][1])
sonuc;
```

Mehmet