```
#Zadanie 1.
import math as mt
def pierwiastki(a, b, c):
  delta = b*b - 4*a*c
  if(delta < 0):
    return ()
  sqrtdelta = mt.sqrt(delta)
  if(delta == 0):
    return ((-b)/(2*a))
  return ((-b-sqrtdelta)/(2*a), (-b+sqrtdelta)/(2*a))
print("Wynik funkcji dla a=2, b=2, c=2", pierwiastki(2,2,-10))
```

```
Wynik funkcji dla a=2, b=2, c=2 (-2.79128784747792, 1.79128784747792)
```

Dla większości wyników nadpisanie zmiennej delta wartością jej pierwiastka działało bez zarzutu. Jednak dla jednego przypadku a=2 b=2 i c=-10 funkcja zwracała błędne wartości. Kiedy do przechowywania wartości średniej wykorzystałem całkowicie nową zmienną sqrtdelta zauważyłem, że błąd przestał się pojawiać. Najwyraźniej może to być spowodowane tym, iż typ który był automatycznie przypisany dla wartości średniej był niewystarczającym do utrzymania odpowiedniej dokładności dla wyniku pierwiastkowania tej delty.

```
#zadanie 2.
#Gra w odgadywanie liczby
#features:
#wybór zakresu liczby
#wybór maksymalnej ilości prób
import random
def setup():
  while(True):
    minn = int(input("Podaj minimalną wartość liczby do zgadnięcia "))
    maxx = int(input("Podaj maksymalną wartość liczby do zgadnięcia "))
    chances = int(input("Podaj maksymalną ilość prób "))
    if(max > min and chances >= 1):
      number = random.randint(minn,maxx)
      return (number,minn, maxx, chances)
def showWinInfo(tryNumber):
  print("Gratulacje!!!")
  print("Udało Ci się zgadnąć poprawną liczbę!!!")
  print("Przy", tryNumber, "podejściu!")
def showLoseInfo(tryNumber, chances, guess, number):
  print("Niestety podana przez Ciebie liczba jest niepoprawna :(")
  print("Nie poddawaj się!")
  print("Pozostało Ci jeszcze",chances-tryNumber, "prób.")
  labelBegin = "Podpowiedź: Twoja liczba jest"
  labelEnd = "od wylosowanej."
  if(guess > number):
    print(labelBegin,"większa",labelEnd)
  else:
    print(labelBegin,"Mniejsza",labelEnd)
def game():
  number,minn, maxx, chances = setup()
  for tryNumber in range(1,chances+1,1):
    while(True):
      guess = int(input("Podaj liczbę "))
      if(guess >= minn and guess <= maxx):
        break
      print("Zgadujesz liczby z zakresu <",minn,",",maxx,">. Podaj więc liczbę z tego zakresu!")
    if(guess == number):
      showWinInfo(tryNumber)
      break
    else:
      if(tryNumber < chances):
        showLoseInfo(tryNumber, chances, guess, number)
      else:
        print("GAME OVER")

#odpalenie gry
game()
```

```
Podaj minimalną wartość liczby do zgadnięcia 1
Podaj maksymalną wartość liczby do zgadnięcia 3
Podaj maksymalną ilość prób 6
Podaj liczbę 0
Zgadujesz liczby z zakresu < 1 , 3 >. Podaj więc liczbę z tego zakresu!
Podaj liczbę 1
Niestety podana przez Ciebie liczba jest niepoprawna :(
Nie poddawaj się!
Pozostało Ci jeszcze 5 prób.
Podpowiedź: Twoja liczba jest Mniejsza od wylosowanej.
Podaj liczbę 2
```

```
Niestety podana przez Ciebie liczba jest niepoprawna :(
Nie poddawaj się!
Pozostało Ci jeszcze 4 prób.
Podpowiedź: Twoja liczba jest Mniejsza od wylosowanej.
Podaj liczbę 3
Gratulacje!!!
Udało Ci się zgadnąć poprawną liczbę!!!
Przy 3 podejściu!
```

```
#zadanie 3.
#a
def stringLength(elm):
  return len(elm)
#b
def checkIfGoodString(elm):
  for x in elm:
    if(x != "A" and x!= "T" and x!= "G" and x!= "C"):
      return False
  return True
#c
def createCompString(elm):
  result = ""
  for x in elm:
    match x:
      case "A":
        result += "T"
      case "T":
        result += "A"
      case "G":
        result += "C"
      case "C":
        result += "G"
  return result
#d
def createReverseString(elm):
  result = ""
  size = len(elm)
  i=size-1
  while(i>=0):
    result += elm[i]
    i-=1
  return result
#e
def countBases(elm):
  Acount = 0
  Tcount = 0
  Gcount = 0
  Ccount = 0
  for x in elm:
    match x:
      case "A":
        Acount+=1
      case "T":
        Tcount+=1
      case "G":
        Gcount+=1
      case "C":
        Ccount+=1
  return (Acount,Tcount,Gcount,Ccount)
#f
def countGG(elm):
  count =0;
  i=1
  while(i<len(elm)):
    if(elm[i-1] == "G" and elm[i] == "G"):
      count+=1
      i+=2
    else:
      i+=1
  return count
#g
def countATA(elm):
  count =0;
  i=2
  while(i<len(elm)):
    if(elm[i-2] == "A" and elm[i-1] == "T" and elm[i]=="A"):
      count+=1
      i+=3
    else:
      i+=1
  return count
```

```
def WorkWithATGCString(elm):
  #a
  print("Długość ciągu:",stringLength(elm))
  #b
  if(checkIfGoodString(elm)):
    print("Podany ciąg jest poprawny.")
  else:
    print("Podany ciąg jest niepoprawny")
  #c
  elmComp = createCompString(elm)
  print("Oryginalny ciąg:")
  print(elm)
  print("Komplementarny ciąg:")
  print(elmComp)
  #d
  elmRev = createReverseString(elm)
  print("Oryginał:", elm)
  print("Odwrócony:",elmRev)
  #e
  Acount,Tcount,Gcount,Ccount = countBases(elm)
  print("Liczba zasad: A:",Acount,"T:",Tcount,"G:",Gcount,"C:",Ccount)
  #f
  print("Ilość wystąpień GG:",countGG(elm))
  #g
  print("Ilość wystąpień ATA:",countATA(elm))

ATGCstring = input("Podaj ciąg ATGC:   ")
WorkWithATGCString(ATGCstring)
```

```
Podaj ciąg ATGC:   ATAGCATAGG
Długość ciągu: 10
Podany cią jest poprawny.
Oryginalny ciąg:
ATAGCATAGG
Komplementarny ciąg:
TATCGTATCC
Oryginał: ATAGCATAGG
Odwrócony: GGATACGATA
Liczba zasad: A: 4 T: 2 G: 3 C: 1
Ilość wystąpień GG: 1
Ilość wystąpień ATA: 2
```

```
Start coding or generate with AI.
```

Zadania dla biblioteki numpy:

```
#zadanie 1
import numpy as np
#losowa macierz 10x10:
mat = np.random.randint(np.random.randint(-1000,1),np.random.randint(1,1002),size= (10,10))
print(mat)
minn = mat[0,0]
maxx = mat[0,0]
for i in range(0,10,1):
  for k in range(0,10,1):
    if(mat[i,k] < minn):
      minn = mat[i,k]
    elif(mat[i,k] > maxx):
      maxx = mat[i,k]
print("Min: ",minn,", Max:",maxx)
```

```
[[ 392   21 -625   55   20   24 -117   92  227   82]
 [-140  347 -595  292 -559 -393 -572 -591  162  -80]
 [ -18  393  292 -624    6  112 -159  126  -98 -424]
 [-486  181 -266   28 -254 -532  377 -613  354  208]
 [ -76  175   52 -158  112   38  401  257 -151  180]
 [-272   98   84 -355   48 -298 -347 -585   81  231]
 [-114  142  363  182 -353  -93 -267 -140  133  366]
 [-219 -120  100  225  160 -663  236 -605 -542 -428]
 [ 138 -517  361 -228 -680  -85  176  124 -510 -402]
 [ 308  -38 -463 -549  -45 -416 -187 -578 -113 -468]]
Min:  -680 , Max: 401
```

Generuję losową macierz za pomocą funkcji randint() z biblioteki numpy z namespace random. Zauważyłem, że kiedy zakres pozostawał ustawiony stale na <-1000,1000> (randint(-1000,1001)) to wyniki były bardzo powtarzalne minimu większość czasu było w okolicach -900 i max również w okolicach 900. Zastosowałem więc losowe zakresy z tych przedziałów wię teraz maksymalny zakres jaki może się wylosować to <-1000,1000> a minimalny to cała macierz wypełniona zerami. Wyniki teraz nie są aż tak powtarzalne.

```
#zadanie 2
import numpy as np

def maxOfRowsAndCols(mat):
  sizeRows = len(mat)
  sizeCols = len(mat[0])
  maxFromRows = np.zeros((sizeRows), dtype= np.int32)
  maxFromCols = np.zeros((sizeCols), dtype= np.int32)
  for i in range(0,sizeRows,1):
    maxFromRows[i] = mat[i,0]
  for i in range(0,sizeCols,1):
    maxFromCols[i] = mat[0,i]
  for i in range(0,sizeRows,1):
    for j in range(0,sizeCols,1):
      if(mat[i,j] > maxFromRows[i]):
        maxFromRows[i] = mat[i,j]
      if(mat[i,j] > maxFromCols[j]):
        maxFromCols[j] = mat[i,j]
  return (maxFromRows,maxFromCols)
def showMaxInfo(maxFromRows, maxFromCols):
  print("----MAX-INFO-----------------")
  i = 0
  for x in maxFromRows:
    print("Max dla wiersza o indeksie",i,":",x)
    i+=1
  print("------------------------------")
  i = 0
  for x in maxFromCols:
    print("Max dla kolumny o indeksie",i,":",x)
    i+=1
  print("------------------------------")
mat = np.random.randint(np.random.randint(-1000,1),np.random.randint(1,1002),size= (10,10))
print(mat)
maxFromRows, maxFromCols= maxOfRowsAndCols(mat)
showMaxInfo(maxFromRows,maxFromCols)
```

```
[[ 137 -136 -172   50  -69 -142  108 -327 -200 -230]
 [-218  -16  -90  -75 -172    3  -50   28  116 -283]
 [-369  130   71 -340  -92  105  -68  112 -151 -136]
 [-241  -20  -14 -298 -275  -34   18   41 -374   26]
 [ 158   95  126 -206  142  153   79 -356  -10 -297]
 [-144  -66  180  155 -108 -260  -74 -310   41  173]
 [  77  -95  -79 -247 -146  142 -274  -79   44 -321]
 [ 113   81   19 -210   93   -6  123 -331 -262 -120]
 [-203 -377 -133 -254 -164  -83 -135 -178   83 -116]
 [ 131 -143    2 -329  147 -348   47   58   27 -295]]
----MAX-INFO-----------------
Max dla wiersza o indeksie 0 : 137
Max dla wiersza o indeksie 1 : 116
Max dla wiersza o indeksie 2 : 130
Max dla wiersza o indeksie 3 : 41
Max dla wiersza o indeksie 4 : 158
Max dla wiersza o indeksie 5 : 180
Max dla wiersza o indeksie 6 : 142
Max dla wiersza o indeksie 7 : 123
Max dla wiersza o indeksie 8 : 83
Max dla wiersza o indeksie 9 : 147
------------------------------
Max dla kolumny o indeksie 0 : 158
Max dla kolumny o indeksie 1 : 130
Max dla kolumny o indeksie 2 : 180
Max dla kolumny o indeksie 3 : 155
Max dla kolumny o indeksie 4 : 147
Max dla kolumny o indeksie 5 : 153
Max dla kolumny o indeksie 6 : 123
Max dla kolumny o indeksie 7 : 112
Max dla kolumny o indeksie 8 : 116
Max dla kolumny o indeksie 9 : 173
------------------------------
```

```
#zadanie 3.
import numpy as np
def change0to1(mat):
  result = mat
  for i in range(0,len(mat),1):
    for j in range(0,len(mat[0]),1):
      match mat[i][j]:
        case 0: result[i][j]=1
        case 1: result[i][j]=0
  return result
mat = np.zeros((5,5))
print("Macierz zer\n",mat)
for i in range(0,len(mat),1):
```

```
  for j in range(0,len(mat[0]),1):
    if(i == 0 or j==0 or i==4 or j==4):
      mat[i,j] = 1
print("Macierz z jedynkami po bokach:\n",mat)
print("Macierz z odwróconymi wartościami:\n",change0to1(mat))
```

```
Macierz zer
 [[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
Macierz z jedynkami po bokach:
 [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
Macierz z odwróconymi wartościami:
 [[0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0.]]
```

```
#zadanie 4.
import numpy as np
#a
def matrixAdd(mat1,mat2):
  if(len(mat1) == len(mat2) and len(mat1[0]) == len(mat2[0])):
    result = np.zeros((len(mat1),len(mat1[0])))
    for i in range(0,len(mat1),1):
      for j in range(0,len(mat1[0]),1):
        result[i,j] = mat1[i,j] + mat2[i,j]
    return result
  else:
    print("Błędne rozmiary macierzy.")
    result = np.zeros(0)
    return result
#b
def matrixSub(mat1,mat2):
  if(len(mat1) == len(mat2) and len(mat1[0]) == len(mat2[0])):
    result = np.zeros((len(mat1),len(mat1[0])))
    for i in range(0,len(mat1),1):
      for j in range(0,len(mat1[0]),1):
        result[i,j] = mat1[i,j] - mat2[i,j]
    return result
  else:
    print("Błędne rozmiary macierzy.")
    result = np.zeros(0)
    return result
#c
def matrixMul(mat1,mat2):
  if(len(mat1[0]) == len(mat2)):
    result = np.zeros((len(mat1),len(mat2[0])))
    for i in range(0,len(mat1),1):
      for j in range(0,len(mat2[0]),1):
        for k in range(0,len(mat1[0]),1):
          result[i][j] = mat1[i,k] * mat2[j,k]
    return result
  else:
    print("Błędne rozmiary macierzy.")
    result = np.zeros(0);
    return result
#d
def matrixmul2div10(mat):
  result = np.zeros((len(mat), len(mat[0])))
  for i in range(0,len(mat),1):
    for j in range(0,len(mat[0]),1):
      result[i,j] = (mat[i,j]*2)/10
  return result
#e
def matrixShowCentralValue(mat):
  return mat[int(len(mat)/2),int(len(mat[0])/2)]
def matrixToVector(mat):
  rc = len(mat)
  lc = len(mat[0])
  result = np.zeros(rc*lc)
  i = 0
  for j in range(0,rc,1):
    for k in range(0,lc,1):
      result[i] = mat[j,k]
      i+=1
```

```
  return result;
mat1 = np.random.randint(np.random.randint(-1000,1),np.random.randint(1,1002),size= (5,5))
mat2 = np.random.randint(np.random.randint(-1000,1),np.random.randint(1,1002),size= (5,5))
print("mat1:\n",mat1)
print("mat2:\n",mat2)
#a
mat3 = matrixAdd(mat1,mat2)
print("a) suma:\n",mat3)
#b
mat4 = matrixSub(mat1,mat2)
print("b) różnica:\n",mat4)
#c
mat5 = matrixMul(mat1,mat2)
print("c) mnożenie:\n",mat5)
#d
mat6 = matrixmul2div10(mat1)
print("d) mat1*2/10:\n",mat6)
#e
print("e) środkowe wartości:\nmat1: ",matrixShowCentralValue(mat1),"\tmat2: ",matrixShowCentralValue(mat2))
#f
vec = matrixToVector(mat1)
print("f) vektor z macierzy mat1:\n",vec)
```

```
mat1:
 [[ 462   88  -77  431 -337]
 [-338  216   14  -50 -497]
 [ 298  -74 -369  172 -322]
 [-148 -421  480  -21  407]
 [ -55 -229 -384   90  -30]]
mat2:
 [[-249  357  426 -123 -137]
 [-197 -125  -47  343  508]
 [ 649  527  121 -301  619]
 [  46  290  214 -212  226]
 [ 656  412  375  537  685]]
a) suma:
 [[ 213.  445.  349.  308. -474.]
 [-535.   91.  -33.  293.   11.]
 [ 947.  453. -248. -129.  297.]
 [-102. -131.  694. -233.  633.]
 [ 601.  183.   -9.  627.  655.]]
b) różnica:
 [[  711.  -269.  -503.   554.  -200.]
 [ -141.   341.    61.  -393. -1005.]
 [ -351.  -601.  -490.   473.  -941.]
 [ -194.  -711.   266.   191.   181.]
 [ -711.  -641.  -759.  -447.  -715.]]
c) mnożenie:
 [[  46169. -171196. -208603.  -76162. -230845.]
 [  68089. -252476. -307643. -112322. -340445.]
 [  44114. -163576. -199318.  -72772. -220570.]
 [ -55759.  206756.  251933.   91982.  278795.]
 [   4110.  -15240.  -18570.   -6780.  -20550.]]
d) mat1*2/10:
 [[ 92.4  17.6 -15.4  86.2 -67.4]
 [-67.6  43.2   2.8 -10.  -99.4]
 [ 59.6 -14.8 -73.8  34.4 -64.4]
 [-29.6 -84.2  96.   -4.2  81.4]
 [-11.  -45.8 -76.8  18.   -6. ]]
e) środkowe wartości:
mat1:  -369     mat2:  121
f) vektor z macierzy mat1:
 [ 462.   88.  -77.  431. -337. -338.  216.   14.  -50. -497.  298.  -74.
 -369.  172. -322. -148. -421.  480.  -21.  407.  -55. -229. -384.   90.
  -30.]
```