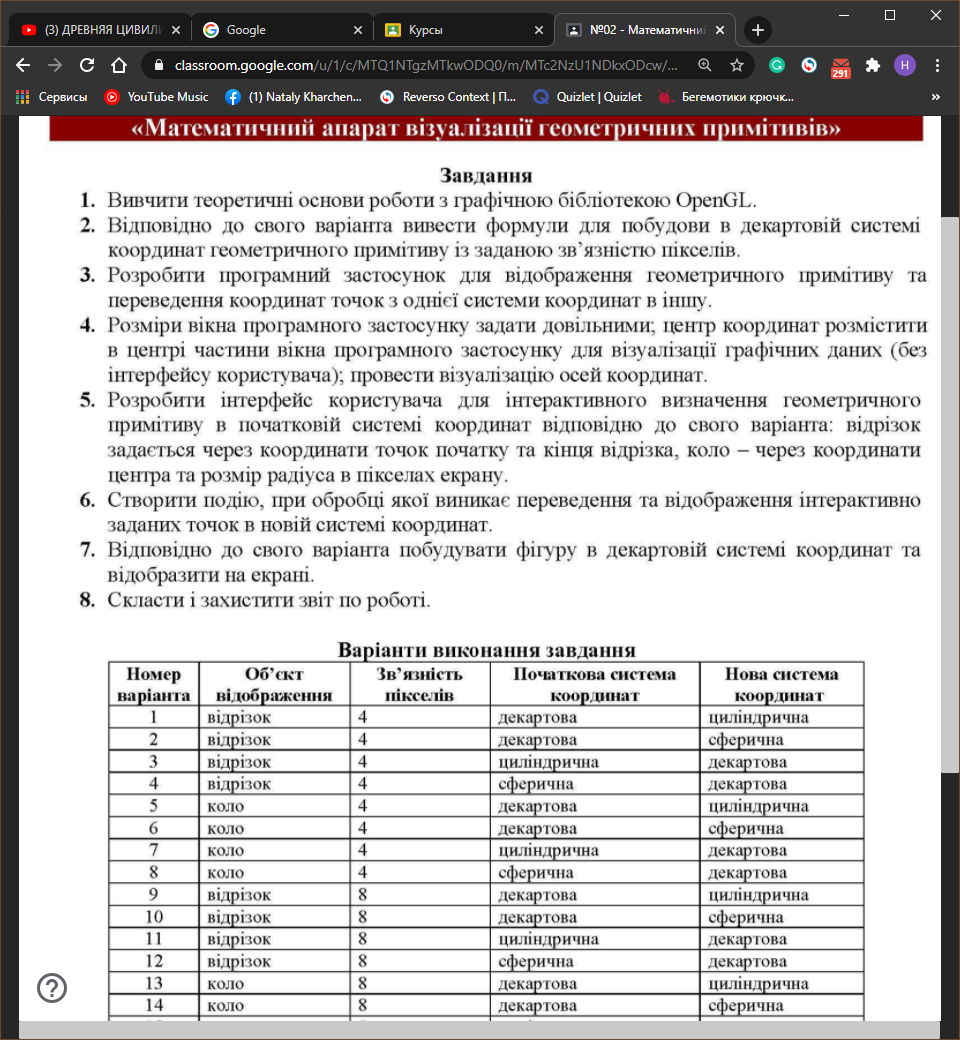
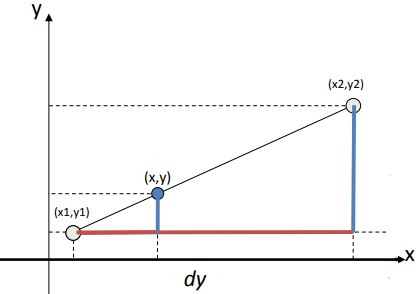
**Завдання**



Отже, для побудови відрізку в декартовій системі координах із зв’язністю пікселів 4 будемо використовувати алоритм Брезенхейма.



– рівняння прямої

Принцип роботи алгоритма Брезенхейма:

Обираеться відрізок та його початкова координата *x*. У циклі до поч. координати додаеться по одиниці у сторону кінця відрізка. На кожному кроці розраховується помилка – відстань між реальною координатою *y* у тому місці та ближньої комірки сітки. Якщо помилка не перевищує половини висоти комірки, то вона заповнюється.

Отже нам необхідно спочатку обчислюється кутовий коефіцієнт *(y1 - у0) / (x1 - x0)*. Значення помилки в початковій точці відрізка *(0,0)* приймається рівним нулю і перша осередок заповнюється. На наступному кроці до помилки додається кутовий коефіцієнт і аналізується її значення, якщо помилка менше *0.5*, то заповнюється осередок *(x0 + 1, у0)*, якщо більше, то заповнюється осередок *(x0 + 1, у0 + 1)* і з значення помилки віднімається одиниця . На зображенні нижче жовтим кольором показана лінія до растеризації, зеленим і червоним - відстань до найближчих осередків. Кутовий коефіцієнт дорівнює одній шостій, на першому кроці помилка стає рівною кутовому коефіцієнту, що менше *0.5*, а значить ордината залишається колишньою. До середини лінії помилка перетинає кордон, з неї віднімається одиниця, а новий піксель піднімається вище. І так до кінця відрізка.

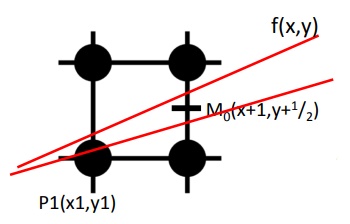
Алгоритм Брезенхейма (метод центральної точки)

| 1 | Необхідно підставити точку *M* у функцію *f*:   * якщо *f(M)<0* обираємо точку *NE* * якщо *f(M)>=0* обитаємо точку *E* |
| --- | --- |

Зміна значення *f(M)* при переході до нових точок (*NE* або *E*) виражається:

– формула знаходження верхньої точки

– формула знаходження нижньої точки



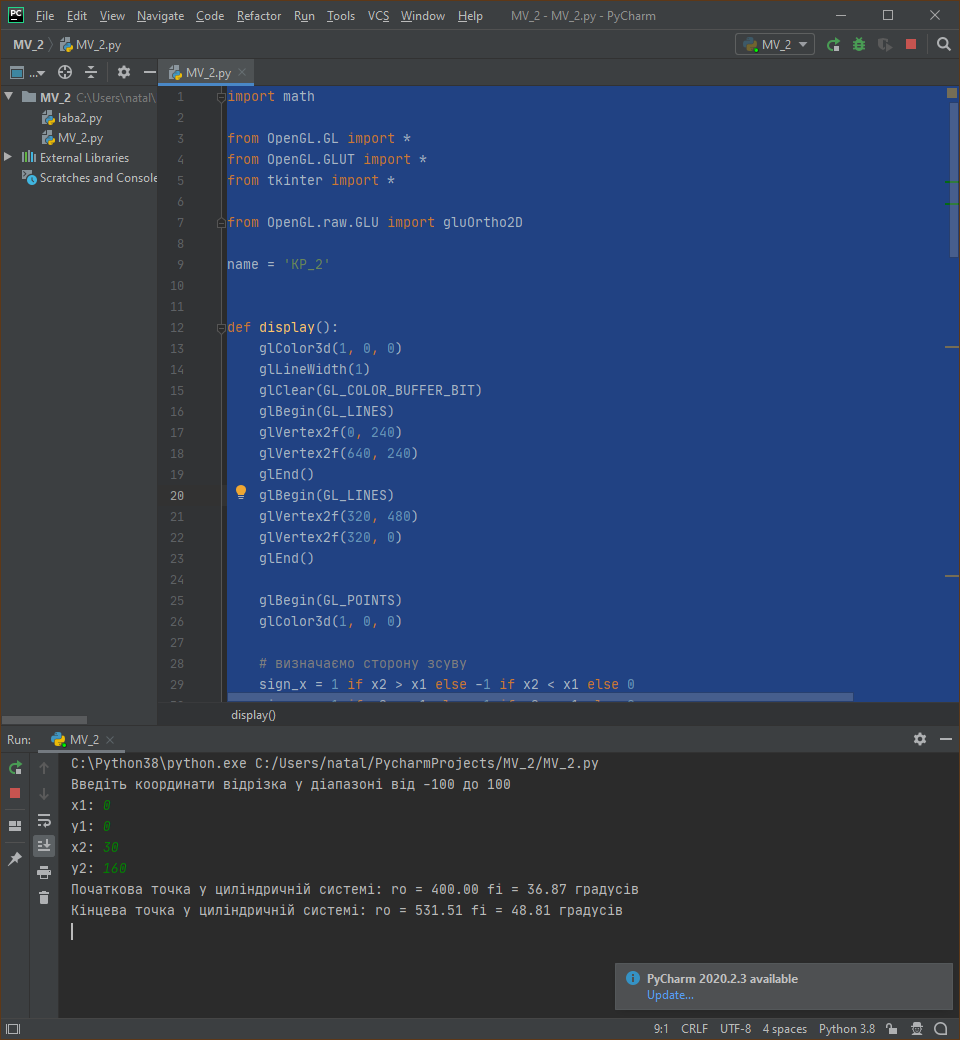
Формула для знаходження початкового значеня точки *(x1,y1)*:

Отже, для розрахунку необхідної точки нам потрібно визначити центральну точку і відповідно до її значення обрати: якщо вона >= 0 верхню, а якщо < 0 нижню.

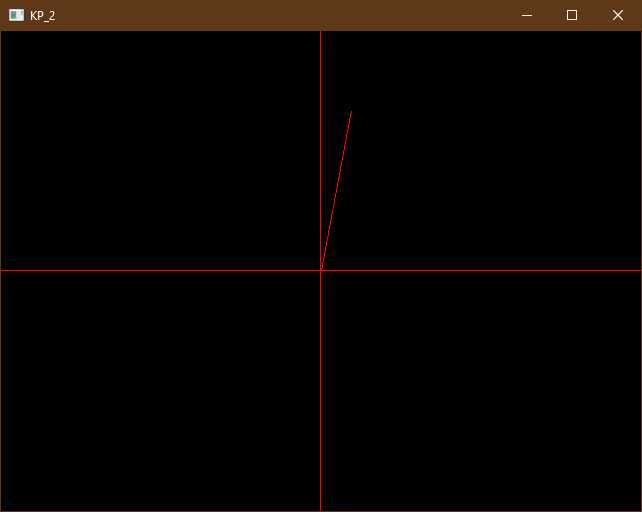
**Лістинг программи**

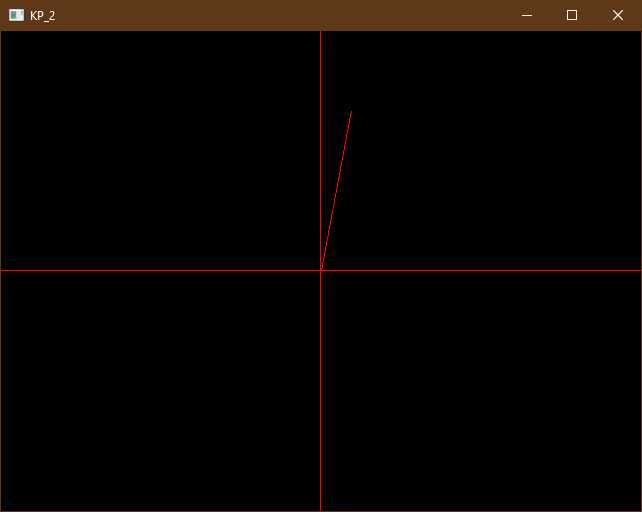
import math  
  
from OpenGL.GL import \*  
from OpenGL.GLUT import \*  
from tkinter import \*  
  
from OpenGL.raw.GLU import gluOrtho2D  
  
name = 'KP\_2'  
  
  
def display():  
 glColor3d(1, 0, 0)  
 glLineWidth(1)  
 glClear(GL\_COLOR\_BUFFER\_BIT)  
 glBegin(GL\_LINES)  
 glVertex2f(0, 240)  
 glVertex2f(640, 240)  
 glEnd()  
 glBegin(GL\_LINES)  
 glVertex2f(320, 480)  
 glVertex2f(320, 0)  
 glEnd()  
  
 glBegin(GL\_POINTS)  
 glColor3d(1, 0, 0)  
  
 # визначаємо сторону зсуву  
 sign\_x = 1 if x2 > x1 else -1 if x2 < x1 else 0  
 sign\_y = 1 if y2 > y1 else -1 if y2 < y1 else 0  
  
 dx = math.fabs(x2 - x1)  
 dy = math.fabs(y2 - y1)  
  
 # визначаємо нахил відрізку  
 # ex відповідає за протяжність по координаті x  
 # ey відповідає за протяжність по координаті y  
 if dx > dy:  
 pdx, pdy = sign\_x, 0  
 ex, ey = dy, dx  
 else:  
 pdx, pdy = 0, sign\_y  
 ex, ey = dx, dy  
  
 x, y = x1, y1  
  
 err, t = ey / 2, 0  
  
 while t < ey:  
 glVertex2i(x, y)  
 err -= ex  
 if err < 0:  
 err += ey  
 x += sign\_x  
 y += sign\_y  
 else:  
 x += pdx  
 y += pdy  
 t += 1  
 glEnd()  
 glFlush()  
  
 glutSwapBuffers()  
 glutPostRedisplay()  
 return  
  
  
def cartesian\_to\_cylindrical(x, y):  
 ro = math.sqrt(x\*\*2 + y\*\*2)  
 fi = math.degrees(math.atan(y / x))  
  
 return ro, fi  
  
  
def main():  
 glutInit(sys.argv)  
  
 glutInitWindowSize(640, 480)  
 glutInitWindowPosition(50, 50)  
 glutCreateWindow(name)  
 glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGB)  
  
 glClearColor(0, 0, 0, 0)  
 glColor3f(0, 0, 0)  
 glPointSize(1)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluOrtho2D(0, 640, 0, 480)  
  
 global x1, x2, y1, y2  
 print("Введіть координати відрізка у діапазоні від -100 до 100")  
 x1 = int(input('x1: ')) + 320  
 y1 = int(input('y1: ')) + 240  
 x2 = int(input('x2: ')) + 320  
 y2 = int(input('y2: ')) + 240  
 glutDisplayFunc(display)  
 ro1, fi1 = cartesian\_to\_cylindrical(x1, y1)  
 ro2, fi2 = cartesian\_to\_cylindrical(x2, y2)  
 print("Початкова точка у циліндричній системі: ro = %.2f" % ro1, "fi = %.2f" % fi1, "градусів")  
 print("Кінцева точка у циліндричній системі: ro = %.2f" % ro2, "fi = %.2f" % fi2, "градусів")  
 glutMainLoop()  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Консоль



Результат





**Контрольні запитання**

1. Чим пояснюється необхідність отримання растового подання геометричних об’єктів при розробці графічних програмних застосунків?

Растерізація використовується для подання геомертичної фігури у вигляді сукупності окремих точок(пікселів), що необхідно для найбільш просте представлення цієї фігури.

1. В чому полягають недоліки найпростіших алгоритмів растового подання лінії?

* такі алгоритми проводять розрахунки над числами з плаваючою точкої, що призводить до менш точних результатів;
* після кожного кроку растерізації виконується округлення значень, что значно знижує швидкість роботи;
* через округлення знічень з’являється похибка у розрахунках.

1. В чому полягає основна ідея алгоритму Брезенхейма для отримання растового полання лінії?

Основна ідея полягає у тому, що якщо кутовий коєфіцієнт прямої менше *0,5* то наступною точкою ми обираємо верхню(тобто точку над опорною), а якщо менше то точку яка знаходиться зліва від опорної.

1. В чому полягає основна ідея алгоритму Брезенхейма для отримання растового полання кола?

По-перше ми можемо розрахувати тільки ¼ кола, а усе інше побудувати за допомогою відображення.

По-друге за розрахованим параметром , де *r* – це радіус кола, ми визначаємо де розмістити наступну точку. Якщо він менше або рівний *0* то ми додамо до координати *x* одиницю, якщо менше *0* то дадамо до координати *x* одиницю та выднымено выд координати *y* одиницю.