Міністерство освіти і науки України

НТУУ «Київський політехнічний інститут ім. Ігоря Сікорського»

Фізико-технічний інститут

# Програмування 4

# Лабораторна робота №9

«Базові поняття ООП у мові програмування Python»

**Виконав:**

Студент II курсу ФТІ групи ФЕ-81

Безуглий Ростислав Сергійович

2020

1. Завдання лабораторної роботи

Побудова об’єктної ієрархії геометричних фігур за своїм варіантом побудови геометричних фігур за вказівками викладача.

**TLocation**

**TPoint(40, рух, колір)**

**TEllipse(колір, розмір)**

**TKrug(рух, колір)**

**TPie(градусна міра)**

1. Аналіз умови задачі.

Базовий клас TLocation буде являти собою локацію в межах вікна і складатися с конструктору, а також головним чином з двох координат і методів для їх встановлення і отримання. Для контролю кількості створюваних об’єктів перевизначимо спеціальний метод \_\_new\_\_. На базі класу TLocation визначимо клас TPoint, сутність якого – точка, що має положення у вікні, атрибут колір, та методи для його встановлення та отримання, метод руху, що визначає переміщення об’єкту на певну кількість пікселів по двом координатам. Перевизначимо попередньо визначений метод \_\_new\_\_, зазначивши в ньому ліміт на кількість створених одночасно об’єктів класу. Атрибути і методи класу TPoint наслідує клас TEllipse, що характеризує еліпс та відрізняється наявністю атрибутів ширини та висоти та методів для їх встановлення та отримання. Як і попередній клас, він частково використовує в своєму конструкторі конструктор батьківського класу. Метод встановлення коліру перевизначений – тут може бути встановлено два кольори замість одного. Наступний за ієрархією клас TKrug, що містить перевизначені методи руху і коліру, представляє собою круг, задається радіусом, що визначає висоту й ширину фігури. Метод руху для об’єктів цього класу змінює їх положення одразу до початку координатної прямої за встановлених значень менше нуля, або в кінець, якщо встановлені значення більші або дорівнюють нулю. Колір знову може буті встановлений лише один. Останній клас TPie представляє собою сектор кругу, визначається початковим кутом та градусною мірою, містить методи їх встановлення та отримання.

Для зручної і швидкої оцінки роботи програми побудуємо простий графічний інтерфейс, що дозволятиме переглядати списки та адреса створених об’єктів класів, їхню кількість та ліміт одночасного існування, з’ясовувати та змінювати стан об’єктів, використовуючи їх методи. Результати команд будуть відображатися у текстовому вікні, у вікні консолі можливо переглядати системні повідомлення передбачені в самих класах. За рахунок такої будови зручно розділити програму на два модулі: основний із графічним інтерфейсом, та додатковий із визначеннями класів.

1. Код реалізації

|  |
| --- |
| Python\_lab9.py |
| import module1 as md  import PySimpleGUI as sg  answer = ''  while (True):  while(answer.count('\n')>=20):  answer = answer[answer.find('\n')+1:]  sg.ChangeLookAndFeel('GreenTan')  window = sg.Window('Object Menu', default\_element\_size=(40, 1), size = (600,550))  sg.Spin(('new','del','set location','get location'))  layout = [  [sg.Radio('TLocation',"group", size=(7,1), default = True), sg.Text(size=(3,1), text=str(len(md.TLocation.instances))), sg.Spin(values=(md.TLocation.instances), size=(47,1)), sg.Spin(size=(10,1),values=('new','delete','set location','get location'))],  [sg.Radio('TPoint',"group", size=(7,1)), sg.Text(size=(3,1), text=str(len(md.TPoint.instances))+'/'+str(md.TPoint.limit)), sg.Spin(values=(md.TPoint.instances), size=(47,1)), sg.Spin(size=(10,1),values=('new','delete','set location','get location','move','set colour','get colour'))],  [sg.Radio('TEllipse',"group", size=(7,1)), sg.Text(size=(3,1), text=str(len(md.TEllipse.instances))+'/'+str(md.TEllipse.limit)), sg.Spin(values=(md.TEllipse.instances), size=(47,1)), sg.Spin(size=(10,1),values=('new','delete','set location','get location','move','set colour','get colour','set size','get size'))],  [sg.Radio('TKrug',"group", size=(7,1)), sg.Text(size=(3,1), text=str(len(md.TKrug.instances))+'/'+str(md.TKrug.limit)), sg.Spin(values=(md.TKrug.instances), size=(47,1)), sg.Spin(size=(10,1),values=('new','delete','set location','get location','move','set colour','get colour','set size','get size','set radius','get radius'))],  [sg.Radio('TPie',"group", size=(7,1)), sg.Text(size=(3,1), text=str(len(md.TPie.instances))+'/'+str(md.TPie.limit)), sg.Spin(values=(md.TPie.instances), size=(47,1)), sg.Spin(size=(10,1),values=('new','delete','set location','get location','move','set colour','get colour','set size','get size','set radius','get radius','set degrees','get degrees'))],  [sg.Text(size=(72, 20),text=answer,background\_color='#d3dfda')],  [sg.Submit(size=(16,2)), sg.Column([[sg.InputText(size=(12,1),default\_text='move x, set x')], [sg.InputText(size=(12,1),default\_text='move y, set y')]]), sg.Column([[sg.InputText(size=(12,1),default\_text='set width')], [sg.InputText(size=(12,1),default\_text='set hight')]]), sg.Column([[sg.InputText(size=(12,1),default\_text='start degree')], [sg.InputText(size=(12,1),default\_text='extent degree')]]), sg.Column([[sg.InputText(size=(12,1),default\_text='set radius')], [sg.InputText(size=(12,1),default\_text='colour, colour')]])]  ]  event, values = window.Layout(layout).Read()  window.close()  if event in (None, 'Exit'):  break  if values[0]==1:  c = md.TLocation  cname = 'TLocation'  do = values[2]  obj = values[1]  elif values[3]==1:  c = md.TPoint  cname = 'TPoint'  do = values[5]  obj = values[4]  elif values[6]==1:  c = md.TEllipse  cname = 'TEllipse'  do = values[8]  obj = values[7]  elif values[9]==1:  c = md.TKrug  cname = 'TKrug'  do = values[11]  obj = values[10]  elif values[12]==1:  c = md.TPie  cname = 'TPie'  do = values[14]  obj = values[13]  else:  c = None  if not c == None:  if do == 'new':  if c == md.TLocation:  instance = c(values[15], values[16])  elif c == md.TPoint:  instance = c(values[15], values[16], values[22])  elif c == md.TKrug:  instance = c(values[15], values[16], values[22], values[21])  elif c == md.TPie:  instance = c(values[15], values[16], values[22], values[21], values[19], values[20])  elif c == md.TEllipse:  instance = c(values[15], values[16], values[22], values[17], values[18])  answer += '>>'+str(instance)[9:-1]+'\n'  if instance == None:  answer += 'Reached limit for objects of class {}.\n'.format(cname)  else:  answer += 'Created new object of class {}.\n'.format(cname)  else:  answer += '>>'+str(obj)[9:-1]+'\n'  if do == 'delete' and type(obj)==c:  c.instances[c.instances.index(obj)].\_\_del\_\_()  answer += 'Deleted selected object of class {}.\n'.format(cname)  elif do == 'set location' and type(obj)==c:  answer += 'Setting selected object of {} class on: x = {}, y = {}.\n'.format(cname, values[15], values[16])  obj.set\_location(values[15], values[16])  coordinates = obj.get\_location()  answer += 'Object is now located on: x = {}, y = {}.\n'.format(coordinates[0], coordinates[1])  elif do == 'get location' and type(obj)==c:  coordinates = obj.get\_location()  answer += 'Selected object of {} class is located on: x = {}, y = {}.\n'.format(cname, coordinates[0], coordinates[1])  elif do == 'set colour' and type(obj)==c:  if cname == 'TEllipse':  answer += 'Changing colours of selected object of {} class to: {}.\n'.format(cname, values[22])  temp = values[22].split(', ')  obj.set\_colour(temp)  coordinates = obj.get\_colour()  answer += 'Colours of the selected object are now: {}, {}.\n'.format(coordinates[0], coordinates[1])  else:  answer += 'Changing colour of selected object of {} class to: {}.\n'.format(cname, values[22])  obj.set\_colour(values[22])  answer += 'Colour of the selected object is now: {}.\n'.format(obj.get\_colour())  elif do == 'get colour' and type(obj)==c:  if cname == 'TEllipse':  coordinates = obj.get\_colour()  answer += 'Colours of the selected object of {} class are: {}, {}.\n'.format(cname, coordinates[0], coordinates[1])  else:  answer += 'Colour of the selected object of {} class is: {}.\n'.format(cname, obj.get\_colour())  elif do == 'move' and type(obj)==c:  answer += 'Moving selected object of {} class to: x = {}, y = {}.\n'.format(cname, values[15], values[16])  obj.move(values[15], values[16])  coordinates = obj.get\_location()  answer += 'Object is now located on: x = {}, y = {}.\n'.format(coordinates[0], coordinates[1])  elif do == 'set size' and type(obj)==c:  answer += 'Setting size of selected object of {} class to: width = {}, hight = {}.\n'.format(cname, values[17], values[18])  obj.set\_size(values[17], values[18])  coordinates = obj.get\_size()  answer += 'Size of selected object is now: width = {}, hight = {}.\n'.format(coordinates[0], coordinates[1])  elif do == 'get size' and type(obj)==c:  coordinates = obj.get\_size()  answer += 'Size of selected object of {} class is: width = {}, hight = {}.\n'.format(cname, coordinates[0], coordinates[1])  elif do == 'set radius' and type(obj)==c:  answer += 'Setting radius of selected object of {} class to: {}.\n'.format(cname, values[21])  obj.set\_radius(values[21])  answer += 'Radius of the selected object is now: {}\n'.format(obj.get\_radius())  elif do == 'get radius' and type(obj)==c:  answer += 'Radius of the selected object of {} class is: {}\n'.format(cname, obj.get\_radius())  elif do == 'set degrees' and type(obj)==c:  answer += 'Setting start degree and extent degree of selected object of {} class to: {} and {}.\n'.format(cname, values[19], values[20])  obj.set\_degrees(values[19], values[20])  coordinates = obj.get\_degrees()  answer += 'Start degree and extent degree of selected object are now: start degree = {}, extent = {}.\n'.format(coordinates[0], coordinates[1])  elif do == 'get degrees' and type(obj)==c:  coordinates = obj.get\_degrees()  answer += 'Start degree and exnent degree of selected object are: start degree = {}, extent = {}.\n'.format(coordinates[0], coordinates[1])  else:  if do != 'new' and type(obj)==c:  answer += 'Wrong syntax: there is no method \_{}\_ for class {}.\n'.format(do, cname)  elif do != 'new' and type(obj)!=c:  answer += 'Did you forget to create an object of {} class?\n'.format(cname)  print('Press Enter to exit...') |
| module1.py |
| windowHight = 500  windowWidth = 500  class LimitError(Exception):  pass  class TLocation(object):  instances = []  def \_\_new\_\_(cls, \*args, \*\*kwargs):  instance = object.\_\_new\_\_(cls)  cls.instances.append(instance)  return instance  def \_\_del\_\_(self):  self.instances.remove(self)  def \_\_init\_\_(self, x=1, y=1):  try:  x = int(x)  if x>0 and x<=windowWidth:  self.x = x  else:  raise ValueError  except ValueError:  print('unable to set horisontal coordinate {}; was set to default setting instead'.format(x))  self.x = 1  try:  y = int(y)  if y>0 and y<=windowHight:  self.y = y  else:  raise ValueError  except ValueError:  print('unable to set vertical coordinate {}; was set to default setting instead'.format(y))  self.y = 1  def set\_location(self, x, y):  try:  x = int(x)  if x>0 and x<=windowWidth:  self.x = x  else:  raise ValueError  except ValueError:  print('unable to set horisontal coordinate {}'.format(x))  try:  y = int(y)  if y>0 and y<=windowHight:  self.y = y  else:  raise ValueError  except ValueError:  print('unable to set vertical coordinate {}'.format(y))  def get\_location(self):  return [self.x, self.y]  class TPoint(TLocation):  instances = []  limit = 40  availableColours = ['none', 'black', 'white', 'green', 'red', 'blue']  def \_\_new\_\_(cls, \*args, \*\*kwargs):  try:  if len(cls.instances) >= cls.limit:  raise LimitError  instance = object.\_\_new\_\_(cls)  cls.instances.append(instance)  return instance  except LimitError:  print('unable to create; limit of objects created exceed for this type')  def \_\_init\_\_(self, x=1, y=1, colour='none'):  try:  TLocation.\_\_init\_\_(self, x, y)  colour = str(colour)  if self.availableColours.count(colour)!=0:  self.colour = colour  else:  raise ValueError  except ValueError:  print('unable to set colour {}'.format(colour))  self.colour = 'none'    def set\_colour(self, colour):  try:  colour = str(colour)  if self.availableColours.count(colour)!=0:  self.colour = colour  else:  raise ValueError  except ValueError:  print('unable to set colour {}'.format(colour))  def move(self, mx, my):  try:  mx = int(mx)  my = int(my)  if self.x+mx>0 and self.x+mx<=windowWidth and self.y+my>0 and self.y+my<=windowHight:  self.x += mx  self.y += my  else:  raise ValueError  except ValueError:  print('unable to move vertically by {} and horizontally by {}'.format(mx, my))  def get\_colour(self):  return self.colour  class TEllipse(TPoint):  instances = []  def \_\_init\_\_(self, x=1, y=1, colours=['none', 'none'], width=1, hight=1):  TPoint.\_\_init\_\_(self, x, y, colours[0])  try:  colourAddition = str(colours[1])  if self.availableColours.count(colourAddition)!=0:  self.colourAddition = colourAddition  else:  raise ValueError  except ValueError:  print('unable to set additional colour {}'.format(colourAddition))  self.colourAddition = 'none'  except IndexError:  print('no additional colour is given')  try:  width = int(width)  if width>0:  self.width = width  else:  raise ValueError  except ValueError:  print('unable to set width of {}; was set to default setting instead'.format(width))  self.width = 1  if self.width%2==0:  self.width -= 1  print('width was even, redused by 1 to be centred properly')  try:  hight = int(hight)  if hight>0:  self.hight = hight  else:  raise ValueError  except ValueError:  print('unable to set hight of {}; was set to default setting instead'.format(hight))  self.hight = 1  if self.hight%2==0:  self.hight -= 1  print('hight was even, redused by 1 to be centred properly')  def set\_size(self, width, hight):  try:  width = int(width)  if width>0:  self.width = width  else:  raise ValueError  except ValueError:  print('unable to set width of {}'.format(width))  if self.width%2==0:  self.width -= 1  print('width was even, redused by 1 to be centred properly')  try:  hight = int(hight)  if hight>0:  self.hight = hight  else:  raise ValueError  except ValueError:  print('unable to set hight of {}'.format(hight))  if self.hight%2==0:  self.hight -= 1  print('hight was even, redused by 1 to be centred properly')  def get\_size(self):  return [self.width, self.hight]  def set\_colour(self, colours):  try:  colour = str(colours[0])  if self.availableColours.count(colour)!=0:  self.colour = colour  else:  raise ValueError  except ValueError:  print('unable to set colour {}'.format(colour))  try:  colour = str(colours[1])  if self.availableColours.count(colour)!=0:  self.colourAddition = colour  else:  raise ValueError  except ValueError:  print('unable to set colour {}'.format(colour))  except IndexError:  print('no colour additional colour is given'.format(colour))  def get\_colour(self):  return [self.colour, self.colourAddition]  class TKrug(TEllipse):  instances = []  def \_\_init\_\_(self, x=1, y=1, colour='none', radius=1, width=1, hight=1):  TEllipse.\_\_init\_\_(self, x, y, [colour], width, hight)  try:  radius = int(radius)  if radius>0:  self.radius = radius  else:  raise ValueError  except ValueError:  print('unable to set radius of {}; was set to default setting instead'.format(radius))  self.radius = 1  def set\_radius(self, radius):  try:  radius = int(radius)  if radius>0:  self.radius = radius  TEllipse.set\_size(self, radius\*2, radius\*2)  else:  raise ValueError  except ValueError:  print('unable to set radius of {}'.format(radius))  def get\_radius(self):  return self.radius  def set\_colour(self, colour):  TPoint.set\_colour(self, colour)  def get\_colour(self):  return self.colour  def move(self, mx, my):  try:  mx = int(mx)  my = int(my)  if mx<0:  self.x = 0  elif mx>=0:  self.x = windowWidth  if my<0:  self.y = 0  elif my>=0:  self.y = windowHight  else:  raise ValueError  except ValueError:  print('unable to move vertically by {} and horizontally by {}'.format(mx, my))  def set\_size(self, width, hight):  TEllipse.set\_size(self, width, hight)  if self.width != self.radius\*2-1:  self.radius = int(self.width/2)+1  self.hight = self.width  elif self.hight != self.radius\*2-1:  self.radius = int(self.hight/2)+1  self.width = self.hight  class TPie(TKrug):  instances = []  def \_\_init\_\_(self, x=1, y=1, colour='none', radius=1, width=1, hight=1, startDegree=0, extentDegree=0):  TKrug.\_\_init\_\_(self, x, y, colour, radius)  try:  startDegree = int(startDegree)  if startDegree>=0 and startDegree<360:  self.startDegree = startDegree  else:  raise ValueError  except ValueError:  print('unable to set start degree of pie {}; was set to default setting instead'.format(startDegree))  try:  extentDegree = int(extentDegree)  if abs(extentDegree)<=360:  self.extentDegree = extentDegree  else:  raise ValueError  except ValueError:  print('unable to set extent degree of pie to {} from start degree; was set to default setting instead'.format(startDegree))  def set\_degrees(self, startDegree, extentDegree):  try:  startDegree = int(startDegree)  if startDegree>=0 and startDegree<360:  self.startDegree = startDegree  else:  raise ValueError  except ValueError:  print('unable to set start degree of pie {}'.format(startDegree))  try:  extentDegree = int(extentDegree)  if abs(extentDegree)<=360:  self.extentDegree = extentDegree  else:  raise ValueError  except ValueError:  print('unable to set extent degree of pie to {} from start degree'.format(startDegree))  def get\_degrees(self):  return [self.startDegree, self.extentDegree] |

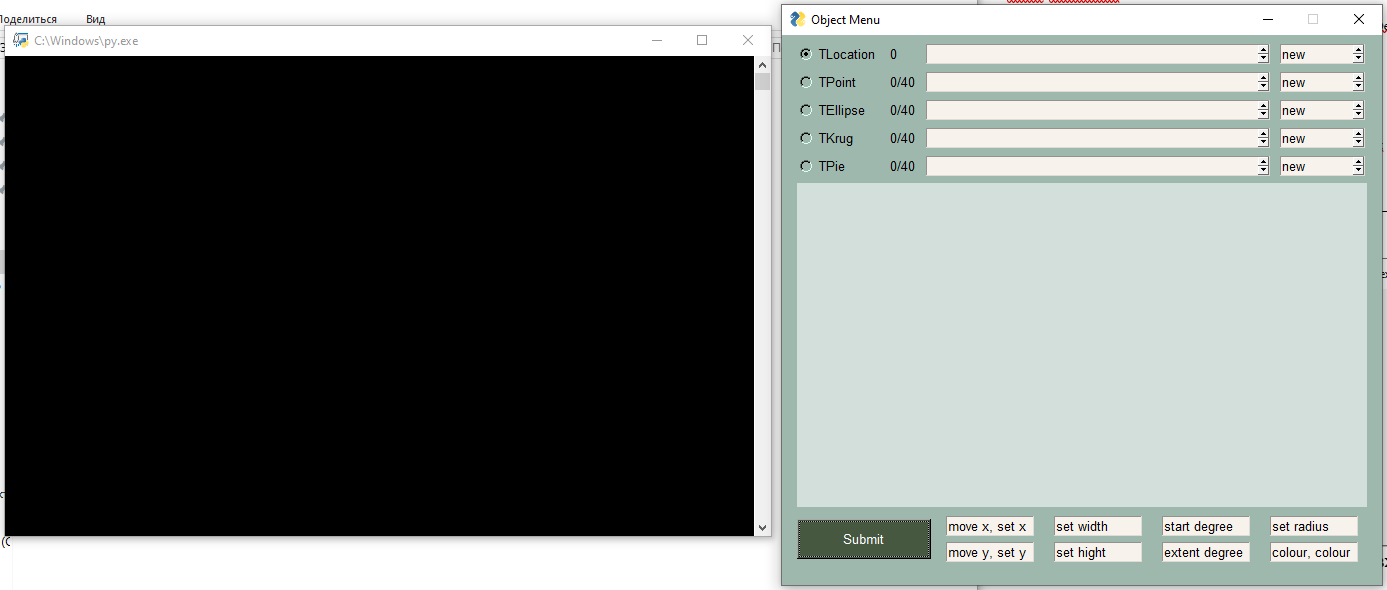
1. Виконання програми

Рис.1.1. Початковий вигляд.

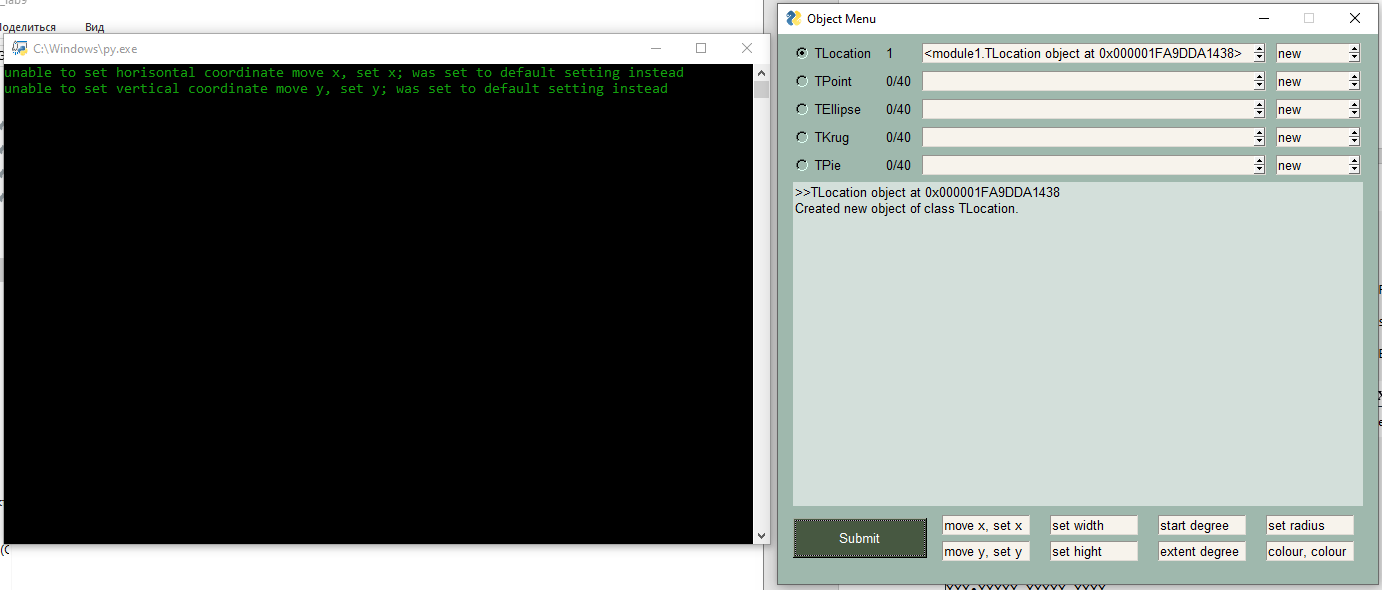
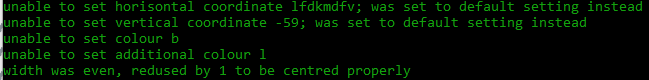
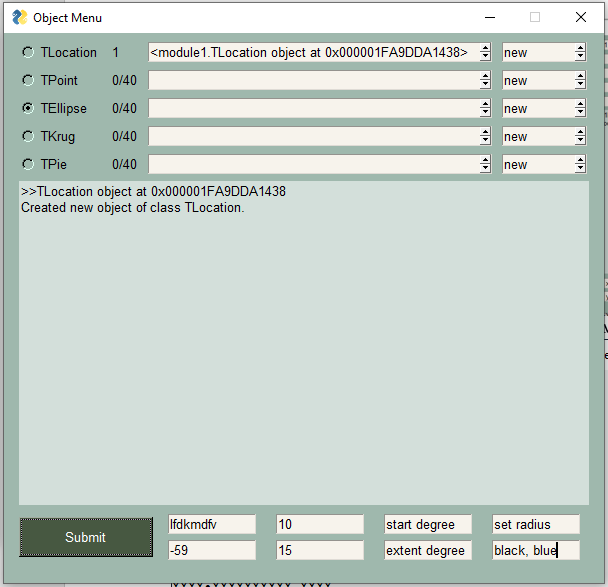
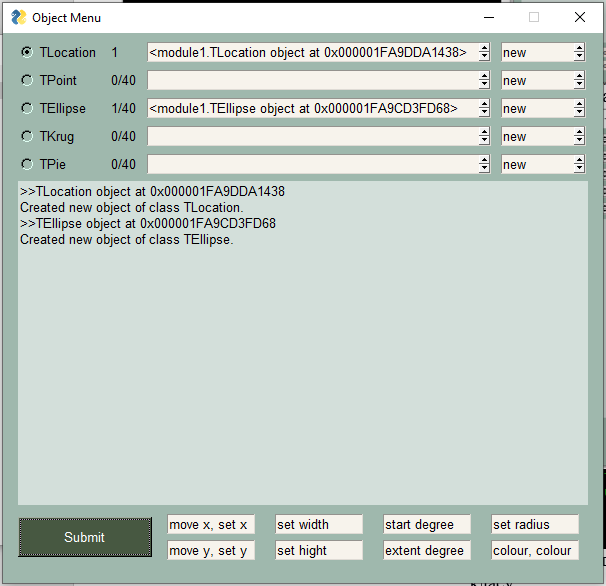
Рис.1.2. Створення об’єктів без параметрів.

Рис.1.3. Створення об’єкту с параметрами та адекватна реакція конструктору класу.

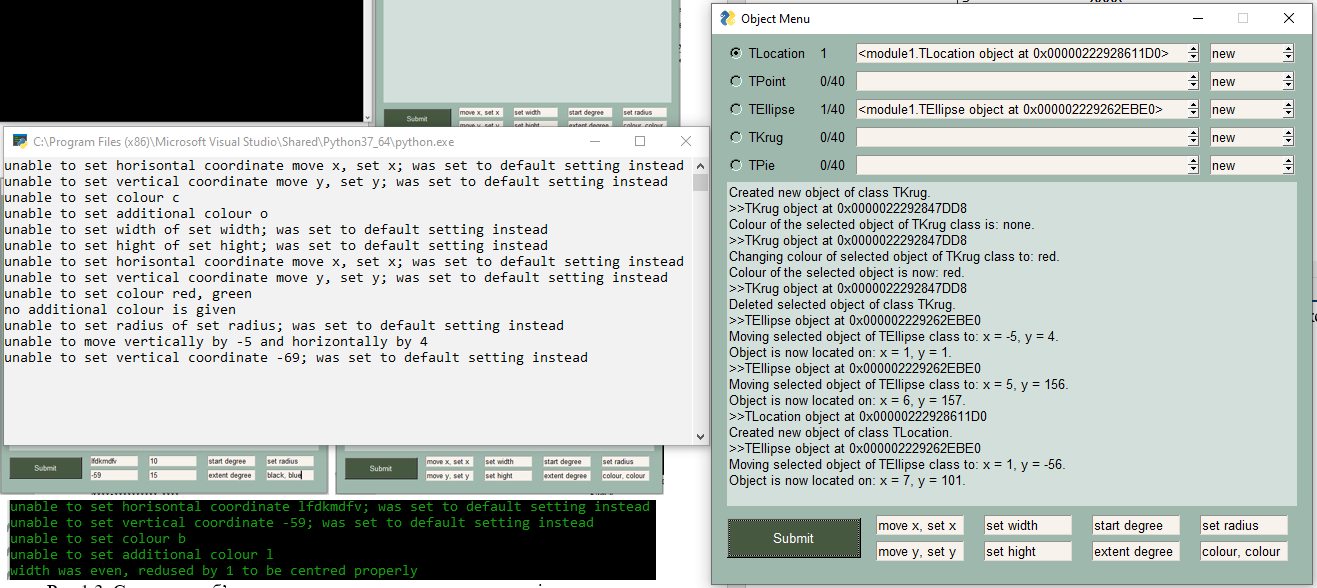


Рис.1.4. Результат роботи програми з різними методами для різних класів.