import sys  
import math  
from abc import ABC, abstractmethod  
  
  
def is\_digit(d: str):  
 if d[0] == "-":  
 d = d[1:]  
 l = d.split(".")  
 if len(l) != 2 and len(l) != 1:  
 return False  
 for i in l:  
 if not i.isdigit():  
 return False  
 return True  
  
  
class InputError(Exception):  
 def \_\_init\_\_(self, info):  
 super().\_\_init\_\_(self)  
 self.errorInfo = info  
  
 def \_\_str\_\_(self):  
 return self.errorInfo  
  
  
class AbstractFunction(ABC):  
 a = 0  
 b = 0  
 c = 0  
 d = 0  
 type = "0"  
  
 def \_\_init\_\_(self, a, b, c, d):  
 self.a = a  
 self.b = b  
 self.c = c  
 self.d = d  
  
 @abstractmethod  
 def getfirstInter(self, a: float, b: float):  
 pass  
  
 @abstractmethod  
 def getvalue(self, x: float):  
 pass  
  
  
class FirstFunction(AbstractFunction):  
 def \_\_init\_\_(self, a, b, c, d):  
 super(FirstFunction, self).\_\_init\_\_(a, b, c, d)  
 self.type = "1"  
  
 def getfirstInter(self, a: float, b: float):  
 return (0.25 \* self.a \* b \*\* 4 + 1/3 \* self.b \* b \*\* 3 + self.c/2 \* b \*\* 2 + self.d\*b) - (0.25 \* self.a \* a \*\* 4 + 1/3 \* self.b \* a \*\* 3 + self.c/2 \* a \*\* 2 + self.d\*a)  
  
 def getvalue(self, x: float):  
 return self.a \* x \*\* 3 + self.b \* x \*\* 2 + self.c \* x + self.d  
  
  
class SecondFunction(AbstractFunction):  
 def \_\_init\_\_(self, a, b, c, d):  
 super().\_\_init\_\_(a, b, c, d)  
 self.type = "2"  
  
 def getfirstInter(self, a: float, b:float):  
 return ( self.a \* math.exp(b) + 1/3 \* self.b \* b \*\* 3 + 0.5 \* self.c \* b \*\*2 + self.d \* b ) - ( self.a \* math.exp(a) + 1/3 \* self.b \* a \*\* 3 + 0.5 \* self.c \* a \*\*2 + self.d \* a )  
  
 def getvalue(self, x: float):  
 return self.a \* math.exp(x) + self.b \* x \*\* 2 + self.c \* x + self.d  
  
  
class ThirdFunction(AbstractFunction):  
 def \_\_int\_\_(self, a, b, c, d):  
 super().\_\_init\_\_(a, b, c, d)  
 self.type = "3"  
  
 def getfirstInter(self, a: float, b:float):  
 return ( self.a \* math.log(b) + 1/3 \* self.b \* b \*\* 3 + 0.5 \* self.c \* b \*\*2 + self.d \* b ) - ( self.a \* math.log(a) + 1/3 \* self.b \* a \*\* 3 + 0.5 \* self.c \* a \*\*2 + self.d \* a )  
  
 def getvalue(self, x: float):  
 return self.a / x + self.b \* x \*\* 2 + self.c \* x + self.d  
  
class ForthFunction(AbstractFunction):  
 def \_\_int\_\_(self, a, b, c, d):  
 super().\_\_init\_\_(a, b, c, d)  
 self.type = "4"  
  
 def getfirstInter(self, a: float, b: float):  
 return (2 \* self.a \* b\*\*0.5 + 1/3 \* self.b \* b \*\* 3 + 0.5 \* self.c \* b \*\*2 + self.d \* b ) - ( 2 \* self.a \* (a-3)\*\*0.5 + 1/3 \* self.b \* a \*\* 3 + 0.5 \* self.c \* a \*\* 2 + self.d \* a)  
  
 def getvalue(self, x: float):  
 return self.a / x\*\*0.5 + self.b \* x \*\* 2 + self.c \* x + self.d  
  
class AbstractSolver(ABC):  
 a: float  
 b: float  
 accuracy: float  
 myfunction: AbstractFunction  
  
 def \_\_init\_\_(self, myfunction: AbstractFunction, accuracy: float, a: float, b: float):  
 self.myfunction = myfunction  
 self.accuracy = accuracy  
 self.a = a  
 self.b = b  
  
 @abstractmethod  
 def solve(self):  
 pass  
  
 @abstractmethod  
 def get\_accuracy(self, a, b):  
 pass  
  
  
class SquareMethodSolver(AbstractSolver):  
 def \_\_init\_\_(self, myfunction, accuracy, a, b):  
 super().\_\_init\_\_(myfunction, accuracy, a, b)  
  
 def get\_accuracy(self, current, next):  
 return (next - current) / 3  
  
 def solve(self):  
 current\_accuracy = 100  
 next\_integral = 0  
 print("1) Левые")  
 print("2) Средние")  
 print("3) Правые")  
 choice = input("Choose which kind of method:\n")  
 if choice != "1" and choice != "2" and choice != "3":  
 raise InputError("Choose form 1, 2 and 3")  
 n\_string = input("Choose the start numbers of square(later will \*2 every time):\n")  
 if not is\_digit(n\_string):  
 raise InputError("Enter a number for step")  
 elif int(n\_string) <= 0:  
 raise InputError("Step should bigger 0")  
 # how many square are there  
 n = int(n\_string)  
  
 is\_first = True  
 while current\_accuracy > self.accuracy:  
 if choice == "1":  
 current\_integral = next\_integral  
 next\_integral = 0  
 k = 0  
 position = self.a  
 gap = (self.b - self.a) / n  
 while k < n:  
 next\_integral += self.myfunction.getvalue(position) \* gap  
 position = position + gap  
 k = k + 1  
 if not is\_first:  
 current\_accuracy = abs(self.get\_accuracy(current\_integral, next\_integral))  
 else:  
 is\_first = False  
 n = 2 \* n  
 elif choice == "2":  
 current\_integral = next\_integral  
 next\_integral = 0  
 gap = (self.b - self.a) / n  
 position = self.a + gap/2  
 k = 0  
 while k < n:  
 next\_integral += self.myfunction.getvalue(position) \* gap  
 position = position + gap  
 k = k + 1  
 if not is\_first:  
 current\_accuracy = abs(self.get\_accuracy(current\_integral, next\_integral))  
 else:  
 is\_first = False  
 n = 2 \* n  
 elif choice == "3":  
 current\_integral = next\_integral  
 gap = (self.b - self.a) / n  
 position = self.a + gap  
 k = 0  
 next\_integral = 0  
 while k < n:  
 next\_integral += self.myfunction.getvalue(position) \* gap  
 position = position + gap  
 k = k + 1  
 if not is\_first:  
 current\_accuracy = abs(self.get\_accuracy(current\_integral, next\_integral))  
 else:  
 is\_first = False  
 n = 2 \* n  
 print("Your range is dived to %d part, and the accuracy is %f"%(n/2, current\_accuracy))  
 print("The result integral is %.5f" % current\_integral)  
  
  
class TrapezoidMethodSolver(AbstractSolver):  
 def \_\_init\_\_(self, myfunction, accuracy, a, b):  
 super().\_\_init\_\_(myfunction, accuracy, a, b)  
  
 def get\_accuracy(self, current, next):  
 return (next - current) / 3  
  
 def solve(self):  
 current\_accuracy = 100  
 next\_integral = 0  
 current\_integral = 0  
 n\_string = input("Choose the start numbers of square(later will \*2 every time):\n")  
 if not is\_digit(n\_string):  
 raise InputError("Enter a number for step")  
 elif int(n\_string) <= 0:  
 raise InputError("Step should bigger 0")  
 # how many Trapezoid are there  
 n = int(n\_string)  
 is\_first = True  
 while current\_accuracy > self.accuracy:  
 current\_integral = next\_integral  
 next\_integral = 0  
 gap = (self.b - self.a) / n  
 position = self.a  
 k = 0  
 while k < n:  
 next\_integral += (self.myfunction.getvalue(position)+self.myfunction.getvalue(position+gap))/2 \* gap  
 position = position + gap  
 k = k + 1  
 if not is\_first:  
 current\_accuracy = abs(self.get\_accuracy(current\_integral, next\_integral))  
 else:  
 is\_first = False  
 n = 2 \* n  
 print("Your range is dived to %d part, and the accuracy is %f"%(n/2, current\_accuracy))  
 print("The result integral is %.5f"% current\_integral)  
  
  
class SimonMethodSolver(AbstractSolver):  
 def \_\_init\_\_(self, myfunction, accuracy, a, b):  
 super().\_\_init\_\_(myfunction, accuracy, a, b)  
  
 def get\_accuracy(self, current, next):  
 return (next - current) / 15  
  
 def solve(self):  
 current\_accuracy = 100  
 next\_integral = 0  
 current\_integral = 0  
 n\_string = input("Choose the start numbers of square(later will \*2 every time):\n")  
 if not is\_digit(n\_string):  
 raise InputError("Enter a number for step")  
 elif int(n\_string) <= 0:  
 raise InputError("Step should bigger 0")  
 # how many Trapezoid are there  
 n = int(n\_string)  
 is\_first = True  
 while current\_accuracy > self.accuracy:  
 current\_integral = next\_integral  
 next\_integral = 0  
 sum\_n = 0  
 sum\_2n = 0  
 k = 1  
 gap = (self.b - self.a)/n  
 while k < n:  
 if k % 2 != 0:  
 sum\_n = sum\_n + self.myfunction.getvalue(self.a + k \* gap)  
 else:  
 sum\_2n = sum\_2n + self.myfunction.getvalue(self.a + k \* gap)  
 k = k + 1  
 next\_integral = gap/3 \* (self.myfunction.getvalue(self.a) + self.myfunction.getvalue(self.b) + 4 \* sum\_n + 2 \* sum\_2n)  
 if not is\_first:  
 current\_accuracy = abs(self.get\_accuracy(current\_integral, next\_integral))  
 else:  
 is\_first = False  
 n = n \* 2  
 print("Your range is dived to %d part, and the accuracy is %f"%(n/2, current\_accuracy))  
 print("The result integral is %.5f"%current\_integral)  
  
while True:  
 try:  
 print("Please choose your equations:")  
 print("1) ax^3 + bx^2 + cx + d")  
 print("2) ae^x + bx^2 + cx + d")  
 print("3) a/x + bx^2 + cx + d")  
 print("4) a/x^0.5 + bx^2 + cx + d")  
 equation = input()  
 if equation != "1" and equation != "2" and equation != "3" and equation != "4":  
 raise InputError("Choose form 1, 2, 3 and 4")  
 string\_coeffi = input("Please input the coefficients of your equation(a, b, c and d). Split by space:\n")  
 for i in string\_coeffi.split(" "):  
 if not is\_digit(i):  
 raise InputError("The coefficients should be number")  
 list\_coeffi = string\_coeffi.split(" ")  
 if len(list\_coeffi) != 4:  
 raise InputError("Make sure that you input all 4 coefficient")  
  
 range\_string = input("Please input the range of calculation, split by spare:\n")  
 list\_range = range\_string.split(" ")  
 for i in list\_range:  
 if not is\_digit(i):  
 raise InputError("Input only number for range")  
 if float(list\_range[0]) >= float(list\_range[1]):  
 raise InputError("Right number should bigger than left")  
 elif len(list\_range) != 2:  
 raise InputError("Make sure only two number were input for range")  
  
 accuracy\_string = input("Please input the accuracy of calculation:\n")  
 if not is\_digit(accuracy\_string):  
 raise InputError("Accuracy should be a number")  
 elif float(accuracy\_string) <= 0:  
 raise InputError("Accuracy should be bigger than 0")  
  
 print("Please choose the method:")  
 print("1) Метод прямоугольников")  
 print("2) Метод трапеций")  
 print("3) Метод Симпсона")  
 method = input()  
 if method != "1" and method != "2" and method != "3":  
 raise InputError("Choose form 1, 2 and 3")  
  
 # create the function  
 my\_function: AbstractFunction  
 if equation == "1":  
 my\_function = FirstFunction(float(list\_coeffi[0]), float(list\_coeffi[1]), float(list\_coeffi[2]),  
 float(list\_coeffi[3]))  
 elif equation == "2":  
 my\_function = SecondFunction(float(list\_coeffi[0]), float(list\_coeffi[1]), float(list\_coeffi[2]),  
 float(list\_coeffi[3]))  
 elif equation == "3":  
 if float(list\_range[0]) <= 0 or float(list\_range[1]) >= 0:  
 raise InputError("Интеграл не существует")  
 my\_function = ThirdFunction(float(list\_coeffi[0]), float(list\_coeffi[1]), float(list\_coeffi[2]),  
 float(list\_coeffi[3]))  
 elif equation == "4":  
 if float(list\_range[0]) < 0 or float(list\_range[1]) <= 0:  
 raise InputError("На этой отрезке функция не определена")  
 if float(list\_range[0]) == 0:  
 list\_range[0] = "0.0001"  
 my\_function = ForthFunction(float(list\_coeffi[0]), float(list\_coeffi[1]), float(list\_coeffi[2]),  
 float(list\_coeffi[3]))  
  
 # create the solver  
 if method == "1":  
 solver = SquareMethodSolver(my\_function, float(accuracy\_string), float(list\_range[0]), float(list\_range[1]))  
 solver.solve()  
 elif method == "2":  
 solver = TrapezoidMethodSolver(my\_function, float(accuracy\_string), float(list\_range[0]), float(list\_range[1]))  
 solver.solve()  
 elif method == "3":  
 solver = SimonMethodSolver(my\_function, float(accuracy\_string), float(list\_range[0]), float(list\_range[1]))  
 solver.solve()  
 go\_on = input("Want to go on?\n")  
 if not go\_on.lower() == "yes":  
 sys.exit(1)  
 print("\n")  
 except InputError as e:  
 print(e)  
 go\_on = input("Want to go on?\n")  
 if not go\_on.lower() == "yes":  
 sys.exit(1)  
 print("\n")