###### **一、项目简介**

本项目运用强大的遗传算法来攻克经典且极具挑战性的旅行商问题（Travelling Salesman Problem，简称 TSP）。TSP 的核心任务是：给定一组城市以及城市之间相互连接的距离信息，需找出一条能遍历所有城市仅一次，并最终回到起始城市的最短闭合路径。这一问题在物流配送、电路布线、旅行规划等诸多现实领域有着广泛的应用场景，而本项目所实现的遗传算法方案，旨在高效且精准地为其提供近似最优解。

###### **二、项目亮点**

算法先进性：遗传算法模拟自然界生物进化的过程，通过选择、交叉、变异等关键操作，让潜在的优质解在多代繁衍中逐渐脱颖而出，避免了传统穷举法面临的 “组合爆炸” 困境，能在可接受的时间成本内逼近最优路径，尤其适用于大规模城市数据集。

代码模块化与可读性：项目采用面向对象编程范式，将旅行商问题的核心数据与操作封装在 TSP 类中，遗传算法的各个流程步骤分别由 GeneticAlgorithm 类中的独立方法实现，如初始化种群、选择、交叉、变异以及整体算法运行逻辑，极大地提升了代码的可维护性与可读性，方便后续的优化拓展。

可视化辅助理解：借助 matplotlib 库，项目具备直观的可视化功能。不仅能够绘制出随机生成的城市坐标分布地图，让用户对问题规模有直观感受，还能动态展示每一代种群中最优路径的演化过程，以及以折线图呈现随着迭代推进，最优路径适应度（即路径长度倒数）的提升趋势，助力用户深度洞察遗传算法的内在优化机制。

###### **三、核心功能**

问题建模与数据生成：在 TSP 类的初始化阶段，依据给定的城市数量 num\_cities，通过 numpy 的随机数生成功能，快速创建出各个城市的二维坐标，精准模拟出 TSP 问题的基础数据环境，为后续算法求解搭建好舞台。

遗传算法精细实现：

初始化种群：initialize\_population 方法利用随机洗牌策略，批量生成多样化的初始路径排列，确保种群具备丰富的基因多样性，为进化提供充足的素材。

选择操作：基于轮盘赌选择原理的 selection 方法，巧妙地将路径适应度转化为选择概率，使得适应度高的优质路径更大概率被选中作为下一代繁衍的 “亲本”，引导种群朝着更优方向迭代进化。

交叉操作：以顺序交叉（OX）为核心的 crossover 方法，在继承父代优良基因片段的同时，通过精心设计的交叉点位选取与基因重组逻辑，孕育出具备新颖基因组合的子代路径，有效拓展了解空间。

变异操作：mutation 方法以一定的变异概率随机扰动路径中的城市顺序，犹如自然界中的基因突变，防止算法过早陷入局部最优的 “进化死胡同”，持续为种群注入创新活力。

结果输出与深度分析：运行完指定代数的遗传算法后，不仅精准输出当前找到的最优旅行路线及其对应的实际长度，更通过记录每一代最优适应度值，绘制出详细的进化曲线，为用户评估算法性能、调整参数策略提供关键依据。

###### **四、使用指南**

在项目主运行脚本中，可以灵活调整以下关键参数：

num\_cities：定义待求解 TSP 问题中的城市总量，默认设置为 [X]，适度增大该值可模拟更复杂的现实场景，但会相应提升计算复杂度与时间成本。

population\_size：遗传算法运行的种群规模，默认为 [X]，较大的种群虽蕴含更多优质解可能性，但也意味着更高的内存占用与运算时长；反之，过小的种群可能导致过早收敛至次优解。

mutation\_rate：控制变异操作发生频率的参数，取值区间为 0 到 1，默认值 [X]。过高的变异率易破坏已积累的优良基因组合，而过低则难以打破局部最优的桎梏，需依据问题特性审慎调整。

generations：指定遗传算法的迭代次数，初始设定为 [X]，增加迭代轮数通常有助于提升解的质量，但同时会延长运行时间，建议通过多次试验寻找最佳平衡点。

启动运行：在命令行终端切换至项目根目录，执行 python main.py（假设主文件名如此），终端将输出最优路径详情，同时自动弹出可视化窗口展示算法演进历程。

五、项目架构剖析

tsp.py：作为旅行商问题的核心定义模块，封装了城市数据生成、路径适应度精确计算等关键逻辑，对外提供简洁易用的接口，是整个项目数据基石。

genetic\_algorithm.py：全方位承载遗传算法的实现细节，从种群的初始混沌构建，到每一代的选择、交叉、变异精细操作，再到多代循环迭代的整体掌控，是算法驱动的核心引擎。

main.py（或类似名称）：统筹各方资源，完成参数初始化、对象实例化，精准启动遗传算法求解流程，并妥善处理最终结果输出与可视化呈现，是用户交互的直接窗口。