



## 中山大学计算机学院

### 人工智能

### 本科生实验报告

(2022 学年春季学期)

课程名称: Artificial Intelligence

教学班级	信息与计算科学班	专业 (方向)	信息与计算科学
学号	21311359	姓名	何凯迪

## 一、实验题目

1-1 最短路径算法

1-2 罗马尼亚旅行问题

## 二、实验内容

### 1. 算法原理

假定源点为  $u$ , 顶点集合  $V$  被划分为两部分: 集合  $S$  和  $V-S$ 。初始时  $S$  中仅含有源点  $u$ , 其中  $S$  中的顶点到源点的最短路径已经确定。

集合  $S$  和  $V-S$  中所包含的顶点到源点的最短路径的长度待定, 并且称从源点出发只经过  $S$  中的点到达  $V-S$  中的点的路径为特殊路径, 并用  $dist[]$  记录当前每个顶点对应的最短特殊路径长度。

### 2. 伪代码

Dijkstra 算法:

输入: 图  $graph$ , 起始节点  $start\_node$ , 目标节点  $target\_node$

1. 初始化距离字典  $distances$ , 将起始节点的距离设为 0, 其他节点的距离设为无穷大
2. 初始化已访问列表  $visited$ , 将所有节点的已访问状态设为  $False$
3. 初始化前驱节点字典  $previous$ , 将所有节点的前驱节点设为  $None$
4. 循环直到所有节点都被访问:
  - a. 找到距离最小且未访问过的节点  $min\_node$ , 将其设为当前节点
  - b. 将当前节点设置为已访问
  - c. 遍历当前节点的所有邻居节点  $neighbor$ :
    - 计算从起始节点到邻居节点的新距离  $new\_dist = distances[current] + \text{边的权重}$
    - 如果新距离小于邻居节点的当前距离  $distances[neighbor]$ :
      - \* 更新邻居节点的距离  $distances[neighbor]$  为新距离  $new\_dist$
      - \* 更新邻居节点的前驱节点  $previous[neighbor]$  为当前节点
5. 如果目标节点的距离  $distances[target\_node]$  为无穷大, 则表示没有路径连接起始节点和目标节点
6. 否则, 通过遍历前驱节点字典  $previous$ , 构建最短路径的节点序列:
  - a. 初始化一个空列表  $path$
  - b. 将目标节点  $target\_node$  添加到  $path$
  - c. 从目标节点  $target\_node$  开始, 通过  $previous$  字典逐步向前找到前驱节点, 并将其添加到  $path$  中
  - d. 将  $path$  列表进行反转, 得到最短路径的节点序列
7. 返回最短路径的节点序列  $path$



### 3. 关键代码展示（带注释）

#### 实验 1.1

```
def dijkstra(graph, start_node, target_node):
    distances = {node: float('inf') for node in graph.keys()}
    distances[start_node] = 0
    visited = {node: False for node in graph.keys()}
    previous = {node: None for node in graph.keys()}

    while True:
        min_dist = float('inf')
        min_node = None
        for node in graph:
            if not visited[node] and distances[node] < min_dist:
                min_dist = distances[node]
                min_node = node
        if min_node is None:
            break
        visited[min_node] = True
        for neighbor, weight in graph[min_node].items():
            new_dist = distances[min_node] + weight
            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                previous[neighbor] = min_node
        if distances[target_node] == float('inf'):
            return []
    path = []
    node = target_node
    while node is not None:
        path.append(node)
        node = previous[node]
    path.reverse()
    return path

# 计算最短路径
for start_node, target_node in queries:
    shortest_path = dijkstra(graph, start_node, target_node)
    if shortest_path:
        shortest_path_length = sum(graph[node1][node2] for node1, node2 in
zip(shortest_path[:-1], shortest_path[1:]))
        print(f"最短路径长度: {shortest_path_length}")
        print(f"最短路径节点序列: {' -> '.join(shortest_path)}")
    else:
        print("无最短路径")
```



## 实验 1.2

funchead.py:

```
# funchead.py

def read_graph_from_file(file_name):
    graph = {}
    with open(file_name, 'r') as file:
        # 读取第一行获取城市数和道路数
        city_count, road_count = map(int, file.readline().split())
        # 读取道路信息并构建图
        for _ in range(road_count):
            start_city, end_city, distance = file.readline().split()
            start_city = start_city.lower() # 将城市名称转换为小写
            end_city = end_city.lower()
            distance = int(distance)
            # 将起始城市和目标城市添加到图中
            if start_city not in graph:
                graph[start_city] = {}
            if end_city not in graph:
                graph[end_city] = {}
            # 将道路信息添加到图中
            graph[start_city][end_city] = distance
            graph[end_city][start_city] = distance
    return graph

def find_city_node(graph, city_name):
    for city in graph.keys():
        if city.startswith(city_name.lower()): # 匹配城市名称（不区分大小写）
            return city
    return None

def calculate_shortest_path(graph, start_node, end_node):
    distances = {city: float('inf') for city in graph.keys()} # 将每个城市的距离初始化为无穷大
    previous_nodes = {city: None for city in graph.keys()} # 将每个城市的前一个节点初始化为 None
    distances[start_node] = 0
    unvisited_cities = set(graph.keys())
    while unvisited_cities:
        current_city = min(unvisited_cities, key=lambda city: distances[city]) # 选择距离最短的城市
        if distances[current_city] == float('inf'):
            break
        unvisited_cities.remove(current_city)
```



```
for neighbor_city, distance in graph[current_city].items():
    new_distance = distances[current_city] + distance
    if new_distance < distances[neighbor_city]:
        distances[neighbor_city] = new_distance
        previous_nodes[neighbor_city] = current_city
if distances[end_node] == float('inf'):
    return [], 0
# 构建最短路径
shortest_path = []
current_city = end_node
while current_city is not None:
    shortest_path.append(current_city)
    current_city = previous_nodes[current_city]
shortest_path.reverse()
return shortest_path, distances[end_node]
def print_shortest_path(shortest_path, total_distance):
    path_str = ' -> '.join(city.capitalize() for city in shortest_path)
    print("最短路径: ", path_str)
    print("总路程: ", total_distance)
def append_to_log_file(shortest_path, total_distance, log_file="log.txt"):
    path_str = ' -> '.join(city.capitalize() for city in shortest_path)
    with open(log_file, 'a') as file:
        file.write("最短路径: " + path_str + "\n")
        file.write("总路程: " + str(total_distance) + "\n")
```

classhead.py:

```
class Node:
    def __init__(self, city_name):
        self.city_name = city_name
        self.neighbors = {}
    def add_neighbor(self, neighbor, distance):
        self.neighbors[neighbor] = distance
class Graph:
    def __init__(self):
        self.nodes = {}
    def add_node(self, city_name):
        node = Node(city_name)
        self.nodes[city_name] = node
    def add_edge(self, start_city, end_city, distance):
        self.nodes[start_city].add_neighbor(end_city, distance)
        self.nodes[end_city].add_neighbor(start_city, distance)
    def get_node(self, city_name):
        return self.nodes.get(city_name)
```

主函数:

```
import funchead
import classhead

# 读取城市和道路数据
graph = funchead.read_graph_from_file("Romania.txt")
# 用户输入
start_city = input("请输入出发城市的名称: ")
end_city = input("请输入到达城市的名称: ")
# 找到出发城市和到达城市的节点
start_node = funchead.find_city_node(graph, start_city)
end_node = funchead.find_city_node(graph, end_city)
# 计算最短路径
shortest_path, total_distance = funchead.calculate_shortest_path(graph, start_node,
end_node)
# 输出最短路径和总路程给用户，并追加写入日志文件
funchead.print_shortest_path(shortest_path, total_distance)
funchead.append_to_log_file(shortest_path, total_distance)
```

### 三、 实验结果及分析

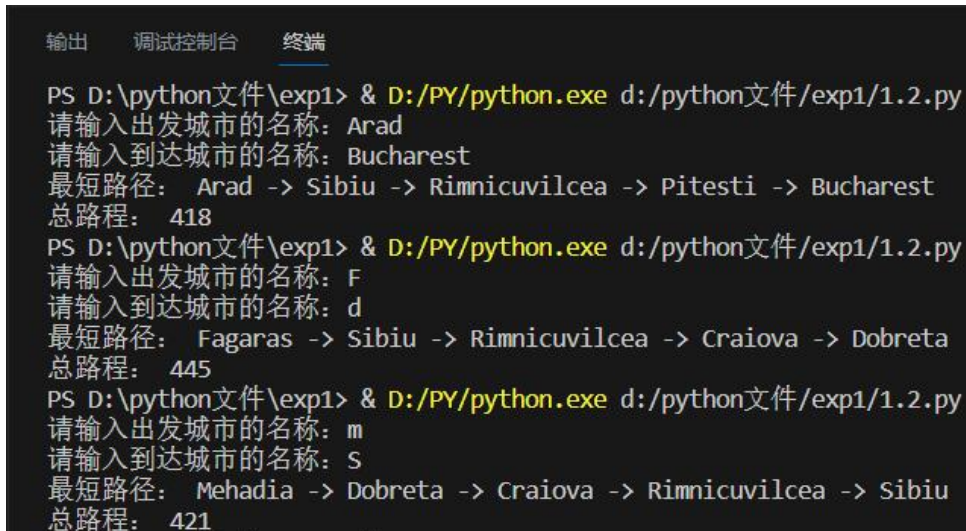
#### 1. 实验结果展示示例（可图可表可文字，尽量可视化）

##### 1.1 最短路径算法



```
输出 调试控制台 终端
PS D:\python文件\exp1> & D:/PY/python.exe d:/python文件/exp1/1.1.py
最短路径长度: 7
最短路径节点序列: a -> b -> e -> d -> z
PS D:\python文件\exp1>
```

##### 1.2 罗马尼亚旅行问题



```
输出 调试控制台 终端
PS D:\python文件\exp1> & D:/PY/python.exe d:/python文件/exp1/1.2.py
请输入出发城市的名称: Arad
请输入到达城市的名称: Bucharest
最短路径: Arad -> Sibiu -> Rimnicuvillea -> Pitesti -> Bucharest
总路程: 418
PS D:\python文件\exp1> & D:/PY/python.exe d:/python文件/exp1/1.2.py
请输入出发城市的名称: F
请输入到达城市的名称: d
最短路径: Fagaras -> Sibiu -> Rimnicuvillea -> Craiova -> Dobreta
总路程: 445
PS D:\python文件\exp1> & D:/PY/python.exe d:/python文件/exp1/1.2.py
请输入出发城市的名称: m
请输入到达城市的名称: S
最短路径: Mehadia -> Dobreta -> Craiova -> Rimnicuvillea -> Sibiu
总路程: 421
```



```
log.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
最短路径: Arad -> Sibiu -> Rimnicuvillea -> Pitesti -> Bucharest
总路程: 418
最短路径: Fagaras -> Sibiu -> Rimnicuvillea -> Craiova -> Dobreta
总路程: 445
最短路径: Mehadia -> Dobreta -> Craiova -> Rimnicuvillea -> Sibiu
总路程: 421
```

## 四、 思考题

1. 在你的电脑上配置你的 Python 开发环境，并运行 Hello World 程序

```
设置 hello.py x
hello.py
1 print('Hello World')

问题 输出 调试控制台 终端
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
Hello World
PS D:\python文件>
```



## 2. 编写一个程序，该程序打印用户输入的内容

```
hello.py x
hello.py > ...
1 content=input()
2 print(content)
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
sysu
sysu
PS D:\python文件> |
```

## 3. 编写 4 个表达式，它们分别使用加法、减法、乘法和除法运算，但结果都是数字 8。

```
1 print(2+6)
2 print(10-2)
3 print(2*4)
4 print(int(16/2))
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
8
8
8
8
PS D:\python文件>
```





4. 找一句你喜欢的名人名言，将这个名人的姓名和他的名言打印出来。其中，名人的姓名存储在变量 `famous_person` 中，消息存储在变量 `message` 中。

```
1 famous_person="Socrates"
2 message="Other men live to eat, while I eat to live"
3 print(famous_person+" once said,"+' '+message+' ')
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
Socrates once said,"Other men live to eat, while I eat to live"
PS D:\python文件>
```

5. 先用 `lstrip()`、`rstrip()` 和 `strip()` 分别处理 `s` 中存储的机构名，观察输出结果；之后，用变量保存 `strip()` 处理后的名称，并分别以小写、大写和首字母大写的方式显示。

```
1 s = "\n\tSchool of Computer Science and Engineering \n"
2 print(s.strip())
3 print(s.rstrip())
4 print(s.lstrip())
5
6 s1=s.strip()
7 print(s1.lower())
8 print(s1.upper())
9 print(s1.title())
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
School of Computer Science and Engineering
School of Computer Science and Engineering
School of Computer Science and Engineering
school of computer science and engineering
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
School Of Computer Science And Engineering
PS D:\python文件>
```





6.假设在游戏中刚射杀了一个外星人，请创建一个名为 alien\_color 的变量，并将其设置为 'green'、'yellow' 或 'red'。

- a) 如果外星人是绿色的，打印一条消息，指出玩家获得了 5 个点。
- b) 如果外星人是黄色的，打印一条消息，指出玩家获得了 10 个点。
- c) 如果外星人是红色的，打印一条消息，指出玩家获得了 15 个点。

```
1 import random
2 alien_color=random.choice(['green','yellow','red'])
3 if alien_color=='green':
4     print('score 5 point')
5 elif alien_color=='yellow':
6     print('score 10 point')
7 else:
8     print('score 15 point')
9
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 10 point
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 5 point
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 15 point
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 10 point
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 5 point
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
score 5 point
PS D:\python文件>
```

7.编程求出从 1 到 100 所有偶数的和

```
1 s=0
2 for i in range(2,101,2):
3     s+=i
4 print(s)
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
2550
PS D:\python文件>
```



8.编写一个猜数字游戏的程序:

- a) 用 random 模块中的 randint() 函数生成一个在 1 到 100 之间的随机整数作为答案;
- b) 程序循环读取用户输入的猜测数字,并向用户提示猜测偏大/偏小,直到猜中;
- c) 用户猜中后,输出一共猜测了多少次。

```
1 import random
2 a=random.randint(1,100)
3 b=int(input())
4 count=1
5 while (b!=a):
6     count=count+1
7     if b>a :
8         print("偏大")
9     elif b<a :
10        print("偏小")
11        b=int(input())
12
13 print("count is:"+str(count))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
偏大
40
偏大
30
偏大
20
偏大
10
偏小
15
偏小
16
count is:7
```



9.编程求出 1 到 100 的和（你能用一行代码完成吗？）

```
1 print(sum(range(1,101)))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py  
5050  
PS D:\python文件>
```

10.实现矩阵乘法的程序（使用二维列表存储矩阵）

```
1 import numpy  
2 A=[[1],[2]]  
3 B=[[1,2]]  
4 C=numpy.dot(A,B)  
5 print(C)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py  
[[1 2]  
 [2 4]]  
PS D:\python文件>
```



11. 创建一个名为 `cities` 的字典，其中将三个城市名用作键；对于每座城市，都创建一个字典，并在其中包含该城市所属的国家、人口约数以及一个有关该城市的事实。在表示每座城市的字典中，应包含 `country`、`population` 和 `fact` 等键。将每座城市的名字以及有关它们的信息都打印出来

```
1 cities={
2     "guangzhou":{
3         "country":"China",
4         "population":18810600,
5         "fact":"good"
6     },
7     "NewYork":{
8         "country":"US",
9         "population":8398000,
10        "fact":"well"
11    },
12    "Tokyo":{
13        "country":"Japan",
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py

guangzhou:  
country:China  
population:18810600  
fact:good

NewYork:  
country:US  
population:8398000  
fact:well

Tokyo:  
country:Japan



**12.魔术师，了不起的魔术师与不变的魔术师：**

a) 创建一个包含魔术师名字列表，并将其传递给 `show_magicians()` 函数，这个函数打印列表中每个魔术师的名字。

```
1 def show_magicians(magicians):  
2     for magician in magicians:  
3         print(magician)  
4  
5 magicians=["aha","bhb","chc"]  
6  
7 show_magicians(magicians)
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py  
aha  
bhb  
chc  
PS D:\python文件>
```



b) 编写一个名为 `make_great()` 的函数，对魔术师列表进行修改，在每个魔术师的名字中都加入字样 “the Great”。调用 `show_magicians()`，确认魔术师列表确实变了。

```
1 def show_magicians(magicians):
2     for magician in magicians:
3         print(magician)
4
5 def make_great(magicians):
6     for i in range(0, len(magicians)):
7         magician = magicians.pop(i)
8         magician = "the great " + magician
9         magicians.insert(i, magician)
10
11 magicians = ["aha", "bhb", "chc"]
12
13 show_magicians(magicians)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
aha
bhb
chc
the great aha
the great bhb
the great chc
PS D:\python文件>
```





c) 在调用 `make_great()` 时，向它传递魔术师列表的副本。由于不想修改原始列表，请返回修改后的列表，并将其存储到另一个变量中。分别对这两列表调用 `show_magicians()`，确认一个列表包含的是原来的魔术师名字，而另一个列表包含的是添加了字样 “the Great” 的魔术师名字

```
1 def show_magicians(magicians):
2     for magician in magicians:
3         print(magician)
4
5 def make_great(magicians):
6     for i in range(0, len(magicians)):
7         magician = magicians.pop(i)
8         magician = "the great " + magician
9         magicians.insert(i, magician)
10    return(magicians)
11
12 magicians = ["aha", "bhb", "chc"]
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
aha
bhb
chc
the great aha
the great bhb
the great chc
PS D:\python文件>
```





### 13. 餐馆、就餐人数与冰淇淋小店：

a)

创建一个名为 Restaurant 的类，其方法 `__init__()` 设置两个属性：`restaurant_name` 和 `cuisine_type`。创建一个名为 `describe_restaurant()` 的方法和一个名为 `open_restaurant()` 的方法，其中前者打印前述两项信息，而后者打印一条消息，指出餐馆正在营业。

```
1 class Restaurant:
2     def __init__(self,name,cuisine_type):
3         self.name=name.title()
4         self.cuisine_type=cuisine_type
5
6     def describe_restaurant(self):
7         msg=f'{self.name}卖{self.cuisine_type}'
8         print(f'{msg}')
9
10    def open_restaurant(self):
11        msg=f'{self.name}在营业'
12        print(f'{msg}')
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
Restaurant1卖food1
Restaurant1在营业
PS D:\python文件>
```



b)

添加一个名为 `number_served` 的属性，并将其默认值设置为 0。根据这个类创建一个名为 `restaurant` 的实例；打印有多少人在这家餐馆就餐过，然后修改这个值并再次打印它。添加一个名为 `set_number_served()` 的方法，它让你能够设置就餐人数；调用这个方法并向它传递一个值，然后再次打印这个值。添加一个名为 `increment_number_served()` 的方法，它让你能够将就餐人数递增；调用这个方法并向它传递一个这样的值：你认为这家餐馆每天可能接待的就餐人数。

```
14
15     def set_number_served(self,number_served):
16         self.number_served=number_served
17
18     def increment_number_served(self,additional_served):
19         self.number_served+=additional_served
20
21 restaurant = Restaurant('restaurant1', 'food1')
22
23 restaurant.describe_restaurant()
24 restaurant.open_restaurant()
25
26
27 print(f'\n服务数量:{restaurant.number_served}')
28 restaurant.number_served = 100
29 print(f'\n服务数量:{restaurant.number_served}')
30
31 restaurant.set_number_served(200)
32 print(f'\n服务数量:{restaurant.number_served}')
33
34 restaurant.increment_number_served(300)
35 print(f'\n服务数量:{restaurant.number_served}')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
Restaurant1卖food1
Restaurant1在营业
```

服务数量:0

服务数量:100

服务数量:200

服务数量:500

```
PS D:\python文件>
```



c)

冰淇淋小店是一种特殊的餐馆。编写一个名为 IceCreamStand 的类，让它继承 Restaurant 类。添加一个名为 flavors 的属性，用于存储一个由各种口味的冰淇淋组成的列表。编写一个显示这些冰淇淋的方法。创建一个 IceCreamStand 实例，并调用这个方法。

```
20
21
22 class IceCreamStand(Restaurant):
23
24
25     def __init__(self, name, cuisine_type='冰激凌'):
26
27         super().__init__(name, cuisine_type)
28         self.flavors = []
29
30     def show_flavors(self):
31
32         print('\n有以下口味的冰激凌:')
33         for flavor in self.flavors:
34             print(f'{flavor.title()}')
35
36
37 big_one = IceCreamStand('A')
38 big_one.flavors = ['A1', 'A2', 'A3']
39
40 big_one.describe_restaurant()
41 big_one.show_flavors()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
A卖冰激凌
```

```
有以下口味的冰激凌:
```

```
A1
```

```
A2
```

```
A3
```

```
PS D:\python文件>
```



14. 访问 Project Gutenberg (<https://gutenberg.org/>)，并找一些你想分析的图书。下载这些作品的文本文件或将浏览器中的原始文本复制到文本文件中。编写一个程序读入这些文本文件，分别对每个文本文件进行初步的统计分析：

a) 统计文本中包含多少个单词（提示：使用字符串方法 `split()` 与函数 `len()`）

```
1 with open("text.txt") as file_object:
2     content = file_object.read()
3     words = content.split()
4     num_words = len(words)
5     print("The file has " + str(num_words) + " words.")
6
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
The file has 197 words.
PS D:\python文件>
```

b) 统计单词“the”出现了多少次（提示：使用字符串方法 `count()` 与 `lower()`）

```
7 the_num = 0
8 with open("text.txt") as file_object:
9
10     for line in file_object:
11         the_num += line.lower().count('the')
12     print(the_num)
13
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
5
PS D:\python文件>
```



15. 如果用列表作为字典的键，会发生什么现象？用元组呢？

python 中列表不能作为字典的键，因为 python 中字典原理是哈希表，因此的键的对象必须是可哈希的对象。由于列表是可变类型，不是可哈希的，所以不能作为键。如下报错：

```
1  s={}
2  a=[1,2,3,4,5]
3  s[a]=6
4  print(s[a])
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
Traceback (most recent call last):
  File "d:/python文件/hello.py", line 3, in <module>
    s[a]=6
TypeError: unhashable type: 'list'
PS D:\python文件>
```

而 python 中可以使用元组作为字典中的键值，因为元组是不可变的，是可哈希的。





```
1 s={}
2 a=(1,2,3,4,5)
3 s[a]=6
4 print(s[a])
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
6
PS D:\python文件>
```



16. 在本课件第2章和第4章提到的数据类型中，哪些是可变数据类型，哪些是不可变数据类型？试结合代码分析。

- 可变/不可变数据类型：变量值发生改变时，变量的内存地址不变/改变。
- 提示：① 你可能会用到 `id()` 函数。② Python 的赋值运算符 (`=`) 是引用传递。

```
1 a=1
2 print(id(a))
3 b=1
4 print(id(b))
5 c=2
6 print(id(c))
7 a=3
8 print(id(a))
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
140711744902928
140711744902928
140711744902960
140711744902992
PS D:\python文件>
```





```
1 a=2.3
2 print(id(a))
3 b=2.3
4 print(id(b))
5 c=3.2
6 print(id(c))
7 a=4.1
8 print(id(a))
9
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
1298105332752
1298105332752
1298105333104
1298104908048
PS D:\python文件>
```



```
1 a="abcd"
2 print(id(a))
3 b="abcd"
4 print(id(b))
5 c="abcde"
6 print(id(c))
7 a="abcdef"
8 print(id(a))
9 |
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
2894834843952
2894834843952
2894834844016
2894835196528
PS D:\python文件>
```



```
1 a=[1,2,3]
2 print(id(a))
3 b=[1,2,3]
4 print(id(b))
5 c=[1,2,3,4]
6 print(id(c))
7 a=[1,2,3,4]
8 print(id(a))
9 a[0]=100
10 print(id(a))
11 |
```

问题 输出 调试控制台 终端

```
PS D:\python文件> & D:/PY/python.exe d:/python文件/hello.py
1635963519872
1635967995200
1635967995264
1635967995136
1635967995136
PS D:\python文件>
```

用同样的方法可依次得出：int、float、bool、str 和元组是不可变数据类型；列表和字典是可变数据类型。

## 五、 参考资料

[https://edu.csdn.net/skill/python?ops\\_request\\_misc=&request\\_id=&biz\\_id=&utm\\_medium=distribute.pc\\_search\\_result.none-task-skillTree-2~all~top\\_card~default-1-python-null-null.142^v73^insert\\_down1,201^v4^add\\_ask,239^v2^insert\\_chatgpt&utm\\_term=python&spm=1018.2226.3001.4187&category=1](https://edu.csdn.net/skill/python?ops_request_misc=&request_id=&biz_id=&utm_medium=distribute.pc_search_result.none-task-skillTree-2~all~top_card~default-1-python-null-null.142^v73^insert_down1,201^v4^add_ask,239^v2^insert_chatgpt&utm_term=python&spm=1018.2226.3001.4187&category=1)