人工智能实验

2022年春季

课程信息

- 课程网站
  - “超算习堂”教学平台：https://easyhpc.net/course/143
  - 加入课程邀请码：0277
  - 请在超算习堂的账号信息中，完善好真实姓名和学号

- 课程Q群：827840726

课程签到

实验课程要求

实验课程内容:

- 由助教讲解实验内容
- 验收前一次的实验内容（包括公式推导、代码解释、现场运行代码产生结果等）
- 会进行考勤

实验课程要求:

- 实验需要一定的数学基础以及编程基础（公式的推导以及代码的实现）
- 编程语言使用Python/C++/Java
  - 不能使用现有算法高级库（除非助教特别说明），否则扣分。
- 禁止抄袭（代码和实验报告都禁止抄袭，若被发现后果严重）

实验课程安排（暂定）

周次	课上	课下	重要时间节点
1	Python程序设计基础 I	实验1-1	
2	Python程序设计基础 II	实验1-2	
3	归结推理	实验2-1	实验1提交
4	知识图谱	实验2-2	
5	Prolog简介	实验2-3	
6	盲目搜索	实验3-1	实验2提交
7	启发式搜索	实验3-2	
8	博弈树搜索	实验4-1	实验3提交
9	高级搜索算法	实验4-2	
10	/		

6~7个实验板块，一般每个板块收一次实验代码，写一份实验报告

实验课程安排（暂定）

周次	课上	课下	重要时间节点
11	贝叶斯网络	实验5-1	实验4提交
12	机器学习 I	实验5-2	
13	机器学习 II	实验5-3	
14	机器学习 III	实验6-1	实验5提交
15	机器学习 IV	实验6-2	
16	智能规划	实验7	实验6提交
17	复习		实验7提交
18	操作考试		
19	期末验收		操作报告提交

实验成绩评定方法（暂定）：
实验成绩（100%） = 平时成绩（80%） + 操作考试（20%）
平时成绩（100%） = 考勤与课堂表现（15%） + 平时实验（85%）

实验报告要求

- 实验报告可使用Word/Markdown/Latex等撰写，以pdf格式提交，可参考课程网站（超算习堂）中的模板与实验报告编写建议，应包含如下内容：
  - (1) 算法原理：用自己的话解释一下自己对算法/模型的理解（不可复制PPT和网上文档内容）
  - (2) 伪代码：伪代码或者流程图（注意简洁规范清晰，包含关键步骤）
  - (3) 关键代码展示：可截图或贴文本并对每个模块进行解释，包括代码+注释
  - (4) 创新点&优化：如果有的话，分点列出自己的创新点（加分项）
  - (5) 实验结果展示：基础算法的结果&(4)中对应分点优化后的算法结果+**分析**
  - (6) 思考题：PPT上写的思考题（如有）一般需要在报告最后写出解答
  - (7) 参考资料：参考的文献、博客、网上资源等需规范引用，否则涉嫌抄袭

实验提交

- 提交到课程网站（超算习堂）中对应的课程作业，并注意网站上公布的截止日期
- 提交格式：提交一个命名为”学号_姓名拼音.zip”的压缩包，压缩文件下包含三部分：code文件夹、result文件夹和实验报告pdf文件
  - 实验报告是pdf格式，命名为：**学号_姓名拼音.pdf**
  - **code**文件夹：存放实验代码，一般有多个代码文件的话需要有readme
  - **result**文件夹：存放上述提到的结果文件（不是每次实验都需要交result，如果没有要求提交结果，则不需要result文件夹）
  - “学号_姓名拼音”样例：20******_wangxiaoming
- 如果需要更新提交的版本，则在后面加_v1，_v2。如第一版是”学号_姓名拼音.zip”，第二版是”学号_姓名拼音_v1.zip”，依此类推

目录

函数

- “带名字的代码块”
- 要执行函数定义的特定任务，可调用该函数。
- 需要在程序中多次执行同一项任务时，你无需反复编写完成该任务的代码，而只需调用执行该任务的函数。
- 通过使用函数，程序的编写、阅读、测试和修复都将更容易。

定义函数

5-1.py

- 最简单的函数结构

```
def greet_user():
    print("Hello!")

greet_user()
```

Hello!

- **函数定义**以关键字def开头
- 函数名、括号
- 定义以冒号结尾
- 紧跟的所有缩进行构成**函数体**
- **函数调用**

- 向函数传递参数

```
def greet_user(username):
    print("Hello, " + username.title() + "!")

greet_user('zachary')
```

Hello, Zachary!

- 变量username是一个**形式参数**
- 值'zachary'是一个**实际参数**

```
def greet_user(username:str) ->None:
    print("Hello, " + username.title() + "!")

greet_user('zachary')
```

传递实参

5-1.py

```
def describe_pet(pet_name, animal_type='dog'):
    """show descriptive information of a pet"""
    print("\nI have a " + animal_type + ".")
    print("The " + animal_type + "'s name is " + pet_name.title() + ".")
```

这里的注释称作函数的**文档字符串**，描述了函数的功能

- 位置实参：基于实参的顺序，将实参关联到函数定义中的形参

```
describe_pet('harry', 'cat')
```

```
I have a cat.
My cat's name is Harry.
```

- 默认值：具有默认值的形参需排列在参数列表的后面

```
describe_pet('willie')
```

```
I have a dog.
My dog's name is Willie.
```

- 关键字实参：无需考虑实参顺序

```
describe_pet(animal_type='dog', pet_name='willie’)
describe_pet('willie', animal_type='dog')
```

返回值

5-1.py

- 函数可以处理一组数据，并返回一个或一组值

```
def get_formatted_name(first_name, last_name, middle_name=''):
    if middle_name:
        full_name = first_name + ' ' + middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title()
```

通过默认值让实参变成可选的
Python将非空字符串解读为True
注意：这里是两个单引号

- 用一个变量存储返回的值，或直接使用返回的值

```
musician = get_formatted_name('jimi', 'hendrix')
print(musician)
print(get_formatted_name('john', 'hooker', 'lee'))
```

```
Jimi Hendrix
John Lee Hooker
```

返回值：返回多个值

5-1.py

- 函数可以处理一组数据，并返回一个或一组值

```
def get_formatted_name(first_name, last_name):
    return first_name.title(), last_name.title()
```

- 用多个变量存储返回的值

```
first_name, last_name = get_formatted_name('jimi', 'hendrix')
print(first_name, last_name)
```

```
Jimi Hendrix
```

- 返回的是其实是元组

```
name = get_formatted_name('jimi', 'hendrix')
print(type(name))
```

```
('Jimi', 'Hendrix')
```

返回值：返回字典

5-1.py

- 函数可以返回任何类型的值，包括列表和字典等复杂的数据结构

```
def build_person(first_name, last_name, age=''):
    person = {'first': first_name, 'last_name': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

```
{'first': 'jimi', 'last_name': 'hendrix', 'age': 27}
```

传递实参

- 对某些数据类型来说，在函数内部对传入变量所做的修改，会导致函数外的值同时发生修改，产生副作用。
  - 在目前学过的类型中，列表和字典符合这种情况

- 对于数值、字符串等，可通过返回值将函数内的值传至函数外。

传递列表

5-1.py

- 将列表传递给函数

```
def greet_users(names):
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

传递列表：在函数中修改列表

5-1.py

- 在函数中对传入列表所做的任何修改都是永久性的

```
def modify_list(source, index, target):
    source[index] = target
source = [1, 2, 3]
index = 2
target = 5
modify_list(source, index, target)
print("The modified source is:", source)
index = 1
target = [1,3]
modify_list(source, index, target)
print("The modified source is:", source)
```

```
The modified source is: [1, 2, 5]
The modified source is: [1, [1, 3], 5]
```

传递列表：防止函数修改列表

- 有时候，需要防止函数修改列表。
- 向函数传递列表副本，可保留函数外原始列表的内容：

```
modify_list(source[:], index, target)
```

  - function_name(list_name[:])
  - 利用切片创建副本
- 除非有充分的理由需要传递副本，否则还是应该将原始列表传递给函数。
  - 避免花时间和内存创建副本，从而提高效率
  - 在处理大型列表时尤其如此

传递任意数量的实参

5-1.py

- 形参前加*号，可传递任意数量的实参

```
def make_pizza(*toppings):
    print("\nMaking a pizza ...")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'peppers', 'cheese')
```

```
Making a pizza ...
Toppings:
- pepperoni
('pepperoni',)

Making a pizza ...
Toppings:
- mushrooms
- peppers
- cheese
('mushrooms', 'peppers', 'cheese')
```

- 结合使用位置实参和任意数量实参

```
def make_pizza(size, *toppings):
```

Python先匹配位置实参和关键字实参，
再将余下的实参收集到最后一个形参中

传递任意数量的关键字实参

5-1.py

- 形参前加**号，可传递任意数量的关键字实参

user_info是一个字典

```
def build_profile(first, last, **user_info):
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile

user_profile = build_profile('albert','einstein', location='princeton', field='physics')
print(user_profile)
```

```
{'first_name': 'albert', 'last_name': 'einstein', 'location': 'princeton', 'field': 'physics'}
```

类

- 面向对象编程
- 将现实世界中的事物和情景编写成类，并定义通用行为
- 实例化：基于**类**创建实例（对象）

定义类

5-2.py

- 创建Dog类

```
class Dog():
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def sit(self):
        print(self.name.title() + " is now sitting.")

    def roll_over(self):
        print(self.name.title() + " rolled over!")
```

- 方法__init__()
  - 构造函数，创建新对象时自动调用
  - 开头、末尾各有两个下划线
  - 类中的成员函数成为**方法**
- self
  - 要写在所有方法参数列表的第一位
  - 指代这个对象自身
  - 以self为前缀的成员变量可供类中所有方法使用，称为**属性**
  - 通过self.变量名，可访问、创建与修改属性

类的实例化：对象

5-2.py

- 使用Dog类

```
class Dog():
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def sit(self):
        print(self.name.title() + " is now sitting.")

    def roll_over(self):
        print(self.name.title() + " rolled over!")
```

- 创建对象

```
my_dog = Dog('willie', 6)
```

- 访问属性 Python默认是公有属性

```
print(my_dog.name.title())
print(my_dog.age)
```

```
Willie
6
```

- 调用方法 不需要传实参给self

```
my_dog.sit()
my_dog.roll_over()
```

```
Willie is now sitting.
Willie rolled over!
```

属性的修改

```
class Car():
    def __init__(self,make,model,year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = str(self.year) + " " + self.make + " " + self.model
        return long_name.title()

    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def updata_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles
```

```
my_new_car = Car("audi", "a4", 2016)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
2016 Audi A4
This car has 0 miles on it.
```

5-2.py

- 直接修改属性的值　破坏了封装

```
my_new_car.odometer_reading = 500
my_new_car.read_odometer()
```

```
This car has 500 miles on it.
```

- 通过方法修改属性的值

```
my_new_car.updata_odometer(23500)
my_new_car.read_odometer()
```

```
This car has 23500 miles on it.
```

- 通过方法递增属性的值

```
my_new_car.increment_odometer(100)
my_new_car.read_odometer()
```

```
This car has 23600 miles on it.
```

继承

5-2.py

- **子类**是**父类**的特殊版本
- 子类继承父类的所有属性和方法

```
class ElectricCar(Car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)

my_tesla = ElectricCar('tesla', 'model s', 2016)
print(my_tesla.get_descriptive_name())
```

class 子类名(父类名)

2016 Tesla Model S

- super()是一个特殊函数，在子类中通过super()指向父类

- 可以给子类定义自己的属性和方法
- 子类可以重写父类的方法

```
class ElectricCar(Car):
    …
    def get_descriptive_name(self):
        return " Electric" + super().get_descriptive_name()

my_tesla = ElectricCar('tesla', 'model s', 2016)
print(my_tesla.get_descriptive_name())
```

Electric 2016 Tesla Model S

- 实例可作为属性（类的成员）

函数与类：小结

- 函数
  - 定义函数：语法
  - 返回值：允许多个返回值，允许任意类型
  - 传递实参：关键字实参，默认值，副作用，传递任意数量的实参
- 类
  - 定义类：构造函数、self
  - 类实例化为对象：创建对象、访问属性、调用方法、修改属性
  - 继承

练习：函数与类

9. 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的列表，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。
   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。
   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。

练习：函数与类

10. 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。

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。

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。

目录

读取文件

pi_digits.txt

```
1    3.1415926535
2      8979323846
3      2643383279
```

6-1.py

- 读取整个文件：read()

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

```
3.1415926535
  8979323846
  2643383279
```

- 函数open的参数为**文件路径**
  - 相对路径与绝对路径都可以
- 关键字with在不再需要访问文件后将文件关闭
  - 此时无需调用close()

- 逐行读取：readlines()

```
with open('pi_digits.txt') as file_object:
    for line in file_object.readlines():
        print(line)
```

```
3.1415926535

  8979323846

  2643383279

```

- 得到一个列表
- 空白行：
- 文件中每行末尾有一个换行符，而print语句也会加上一个换行符。可以使用rstrip()去掉。

写入文件

open()的模式参数（第二个）：
'r'：读取模式（默认）
'w'：写入模式
'a'：追加模式
…

只能将字符串写入文件。
如需换行，记得写换行符。

6-1.py

- 写入空文件

```
filename = 'programming.txt'

with open(filename, ‘w') as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I love creating new games.\n")
```

```
1    I love programming.
2    I love creating new games.
3
```

- 追加到文件

```
filename = 'programming.txt'

with open(filename, 'a') as file_object:
    file_object.write("I also love finding meaning in large datasets.\n")
    file_object.write("I love creating apps that can run in a browser.\n")
```

```
1    I love programming.
2    I love creating new games.
3    I also love finding meaning in large datasets.
4    I love creating apps that can run in a browser.
5
```

使用模块json存储数据

- 支持将简单的Python数据结构转储到文件中，并在程序再次运行时加载该文件中的数据。
  - json.dump()：保存
  - json.load()：读取
- 序列化（Serialization）
  - 将对象的状态信息转换为可以存储或传输的形式的过程。
  - 反序列化
- 除了json模块，还有pickle等序列化模块，支持将Python对象存储在文件中，以供未来读取。

6-1.py

```
import json

data1 = {
    'date' : 224,
    'room' : 'D402',
    'time' : '14:20-16:00'
}

with open('data.json', 'w') as f:
    json.dump(data1, f)

with open('data.json', 'r') as f:
    data2 = json.load(f)

print(data2)
```

```
{'date': 224, 'room': 'D402', 'time': '14:20-16:00'}
```

https://www.runoob.com/python3/python3-json.html

异常

- Python使用被称为**异常**的特殊对象来管理程序执行期间发生的错误。每当Python运行发生错误时，它都会创建一个异常对象。
- 如果你未对异常进行处理，程序块将在错误处停止，并显示一个traceback，其中包含有关异常的报告，指出发生了哪种异常。

```
>>> 5/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- 使用try-except代码块处理异常或显示你编写的友好的错误信息，此时，即使出现异常，程序也将继续运行。

使用try-except代码块

6-2.py

- 处理FileNotFoundError异常

```
filename = 'alice.txt'

try:
    with open(filename) as f_obj:
        contents = f_obj.read()
except FileNotFoundError:
    print("Sorry, the file " + filename + " does not exist.")
```

```
Sorry, the file alice.txt does not exist.
```

- 如果try代码块中的代码正常运行，将跳过except代码块；如果代码出错，Python查找并运行对应类型的except代码块中的代码。

使用try-except-else代码块

6-2.py

- 处理ZeroDivisionError异常

```
first_number = input("\nFirst number: ")
second_number = input("Second number: ")
try:
    answer = int(first_number) / int(second_number)
except ZeroDivisionError:
    pass
else:
    print(answer)
print("Finished!")
```

```
First number: 5
Second number: 0
Finished!
```

```
First number: 5
Second number: 2
2.5
Finished!
```

- 仅在try代码块成功执行时才运行的代码，应放在else代码块中。
- pass语句：在代码块中使用，指示Python什么都不做。

自定义异常类

6-2.py

- 只要继承了Exception，就可以定义自己的异常类

```
class FileEmpytException(Exception):
    def __init__(self, file_name):
        self.file_name = file_name

    def __str__(self):
        return "file: " + self.file_name + " is empty!"
```

```
try:
    with open("data.json") as f:
        data = f.read()
    if len(data) == 0:
        raise FileEmpytException("data.json")
    else:
        print(data)

except FileEmpytException as e:
    print(e)
```

文件与异常：小结

- 文件处理
  - 读取文件：读取整个文件、逐行读取
  - 写入文件：写入空文件、追加到文件
  - 使用模块json存储数据

- 异常处理
  - try-except
  - try-except-else
  - pass语句
  - 自定义异常类

练习：文件与异常

11. 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()）
    b) 统计单词“the”出现了多少次（提示：使用字符串方法count()与lower()）

12. 请你建立一个字典，并以一个字典中不存在的键访问该字典。
    a) 观察报错信息，识别程序发生了哪种异常。
    b) 改写代码，使程序可以处理这种异常。

目录

将函数与类存储在模块中

- 将函数与类存储在被称为**模块**的独立文件中，与主程序分离。
- 模块是扩展名为.py的文件，包含要导入到程序中的代码。
- 模块的**导入**

模块名，即py模块文件的文件名

  - import 模块名
    - 调用方式：模块名.函数名或类名
  - from 模块名 import 函数名或类名
    - 调用方式：函数名或类名
    - 可以同时导入多个函数或类，中间用逗号分隔
  - import 模块名 as 模块别名
    - 调用方式：模块别名.函数名或类名
  - from 模块名 import 函数名或类名 as 函数或类的别名
    - 调用方式：函数或类的别名
  - from 模块名 import *
    - 调用方式：函数名或类名
    - 如遇相同名称容易造成覆盖，不推荐

Python中的“main函数”

- swap.py

```
def swap(a, b):
    return b, a

if __name__ == '__main__':
    a = 224
    b = 'Good day!'
    print(a, b)
    a, b = swap(a, b)
    print(a, b)
    a, b = b, a
    print(a, b)
```

```
224 Good day!
Good day! 224
224 Good day!
```

name和main的前与后，均有两个下划线。

- 导入一个模块时，该模块文件的无缩进代码将自动执行。
  - 例如，若当前“main”下的代码没有缩进在if中，则import swap时，这些代码全部都会执行一次。
- 因此，在编写自己的模块时，模块测试代码等要记得缩进于 if __name__ == '__main__':

if __name__ == '__main__' 如何正确理解？ https://www.zhihu.com/question/49136398

Python标准库

- Python标准库是一组Python自带的模块，例如：
  - math
  - random
  - copy
  - csv
  - heapq
  - time
  - os
  - multiprocessing
  - collections
  - unittest
  - …

- 了解Python标准库：Python Module of the Week
  - https://pymotw.com/

外部模块：安装

- 使用pip安装Python包
  - pip是一个负责为你下载并安装Python包的程序，大部分较新的Python都自带pip

- 安装命令：
  - pip install 包的名称

- 类似的，如果计算机同时安装了Python2和Python3，则需使用：
  - pip3 install 包的名称

一些常用的包

- 交互实时编程：jupyter notebook

- 数据分析、计算与可视化：numpy、scipy、pandas、matplotlib
- 机器学习：scikit-learn
- 深度学习：pytorch、tensorflow、keras
- 文本挖掘：genism

- 推荐学习：超算习堂-在线实训

模块与库：小结

- 模块导入
- 编写自己的模块
- Python标准库
- 外部模块

练习：模块与库

- 在超算习堂中，完成“NumPy入门”在线实训

目录

总结

- 至此，你已经对Python 3的基本语法有了足够的了解。
  - 学无止境，还有一些高级语法和丰富的第三方包等待你探索…
  - 在实践中，你还可能会遇到很多无法预料的报错和坑…

- Python官网：Python is a programming language that lets you work quickly and integrate systems more effectively.
  - “胶水语言”，完成不同模块间的简单处理
  - 例如：将A库的输出经过一定形式的转换送进B库做处理
    - 用numpy处理得到数据矩阵，送进sklearn中进行机器学习模型训练与预测

为什么只有python成了胶水语言？ https://www.zhihu.com/question/402762138

实验任务

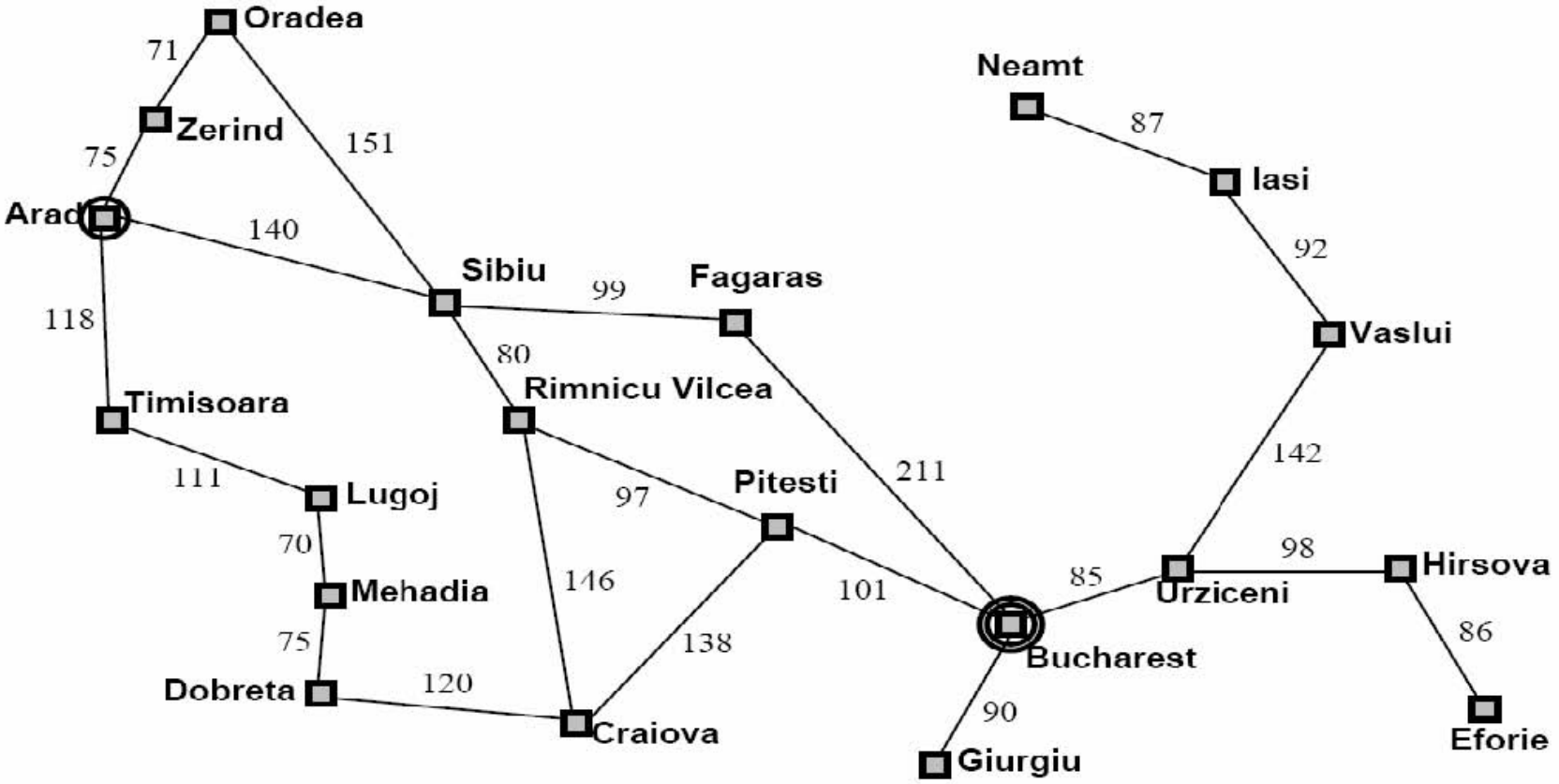
思考题：

- 1. 如果用列表作为字典的键，会发生什么现象？用元组呢？

- 2. 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。

实验1：罗马尼亚旅行问题

- 你是一名正在罗马尼亚旅行的游客，现在你急需从城市1赶到城市2搭乘航班，因此，你需要找到从城市1到城市2的最短路径。



罗马尼亚的地图信息文件：
Romania.txt

文件第1行：
城市数 道路数

文件第2行到第24行，每行是：
城市1名称 城市2名称 路程

实验1：罗马尼亚旅行问题

- 请基于上周你编写的最短路径程序，扩展实现一个搜索罗马尼亚城市间最短路径的导航程序，要求：
  - 出发城市和到达城市由用户在查询时输入
  - 对于城市名称，用户可以输入全称，也可以只输入首字母，且均不区分大小写
  - 向用户输出最短路径时，要输出途经的城市，以及路径总路程
  - 输出内容在直接反馈给用户的同时，还需追加写入一个文本文件中，作为记录日志
  - 为提升代码灵活性，你应在代码中合理引入函数和类（各定义至少一个）
  - 此外，将你定义的一些函数和类，存储在独立的模块文件中

实验1：报告要求

- 实验题目：最短路径搜索
- 报告中包含最短路径算法的算法原理与伪代码
- 分析罗马尼亚旅行问题的实验结果，至少展示以下城市间的路径：
  - Arad → Bucharest
  - Fagaras → Dobreta
  - Mehadia → Sibiu

- 记得完成思考题

实验报告要求

- 实验报告可使用Word/Markdown/Latex等撰写，以pdf格式提交，可参考课程网站（超算习堂）中的模板与实验报告编写建议，应包含如下内容：
  - (1) 算法原理：用自己的话解释一下自己对算法/模型的理解（不可复制PPT和网上文档内容）
  - (2) 伪代码：伪代码或者流程图（注意简洁规范清晰，包含关键步骤）
  - (3) 关键代码展示：可截图或贴文本并对每个模块进行解释，包括代码+注释
  - (4) 创新点&优化：如果有的话，分点列出自己的创新点（加分项）
  - (5) 实验结果展示：基础算法的结果&(4)中对应分点优化后的算法结果+**分析**
  - (6) 思考题：PPT上写的思考题（如有）一般需要在报告最后写出解答
  - (7) 参考资料：参考的文献、博客、网上资源等需规范引用，否则涉嫌抄袭

实验1：提交

- **作业名称：实验1**
- **截止时间：3月6日 23:00**

- 提交一个压缩包[CS/SM]_20******_wangxiaoming.zip，内含：
  - [CS/SM]_20******_wangxiaoming.pdf
  - /code
- 计科同学为CS，保密管理同学为SM

- 提交最终版代码（本周要求的代码）即可，有多个代码文件需要写readme

Thanks!

附录

Python代码风格规范

- 自行了解： PEP 8
- https://www.python.org/dev/peps/pep-0008/

Python之禅

- 在Python命令行中输入import this并回车，发现Python的隐藏彩蛋

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```