

第四课 在程序中调用大模型

在这个章节我们会学习一个很有趣的内容，我们会尝试在程序中调用大语言模型。通过几节课的学习，同学们已经体验了页面版的ChatGPT，了解到这其实是一个能够进行对话的智能模型。实际上，大模型公司会提供一种API，允许外部的开发者，在自己的程序中调用语言模型的能力。我们会在这节课利用语言模型，来做一个聊天的机器人。

重构“人工智障”

在第二课的循环语句学习中，我们学习过一个人工智障的代码。这段代码在资源包的"pygame_codes/2_idiot_AI.py"，或者同学们可以重新看一下循环章节里面的prompt复现这段代码。然后同学们复制这段代码，形成以下的prompt

```
1 {复制pygame_codes/2_idiot_AI.py的代码}
2
3 我希望重构这段程序，将字符串处理的部分分离成一个函数question2response，保持程序的整体功能不变
```

ChatGPT很快完成了这个任务

```
1 def question2response(user_input):
2     user_input = user_input.strip()
3     if user_input.endswith("吗?") or user_input.endswith("吗? "):
4         return user_input[:-2] + "!"
5     elif user_input.endswith("?") or user_input.endswith("? "):
6         return user_input[:-1] + "!"
7     else:
8         return user_input
9
10 while True:
11     user_input = input("请输入一句话: ")
12     response = question2response(user_input)
13     print("输出:", response)
```

我们看到程序被分成了一个函数和一个while循环。while循环的部分是非常简洁的，只有4行代码。而在前面，程序额外定义了一个question2response的函数。这个函数实际上会接收一个字符串形式的变量user_input，进行处理后仍然返回一个字符串。即输入是问题，输出是回答。

当然，同学们或许能认识到，在这里这个函数是个过于粗糙的“简易”人工智能实现，如果把这个函数换成一个更聪明的函数，或许就能真正地实现有一定对话能力的人工智能。这就是我们在这个章节里面要进行学习的内容。

函数的基本概念

遇到基础知识的内容，同学们可以向ChatGPT进行对应的提问

- 1 给我举一些python中函数实现的例子，并向我介绍关于函数的基础概念

ChatGPT的输入如下：

在Python中，函数是可以重复使用的代码块，用来执行特定任务。函数可以接受输入（称为参数或参数），可以返回输出（返回值）。使用函数的主要好处是代码的重用性和结构化，使得代码更易于维护 and 理解。

Python函数的基础概念

1. **定义函数**：使用 `def` 关键字来定义一个函数。函数名通常遵循小写字母和下划线的命名规则（例如，`calculate_sum`）。
2. **函数参数**：函数可以接收输入参数，这些参数可以在函数内部使用。参数可以有默认值，也可以是可选的。
3. **返回值**：函数可以使用 `return` 语句返回一个值。如果没有指定返回值，函数默认返回 `None`。
4. **调用函数**：定义函数之后，你可以通过其名称调用函数并传递必要的参数。
5. **作用域**：函数内部定义的变量具有局部作用域，只在函数内部可见。全局变量在函数内部也可访问，但若需要在函数内修改全局变量的值，需要使用 `global` 关键字。

函数的例子

1. 简单的无参数函数

```
1 def greet():
2     print("Hello, World!")
3
4 greet() # 调用函数
```

输出：

```
1 Hello, World!
```

2. 带参数的函数

```
1 def greet(name):  
2     print(f"Hello, {name}!")  
3  
4 greet("Alice") # 调用函数并传递参数
```

输出：

```
1 Hello, Alice!
```

3. 有返回值的函数

```
1 def add(a, b):  
2     return a + b  
3  
4 result = add(5, 3) # 调用函数并接收返回值  
5 print(result)
```

输出：

```
1 8
```

4. 带有默认参数的函数

```
1 def greet(name="Guest"):  
2     print(f"Hello, {name}!")  
3  
4 greet() # 不传递参数调用函数  
5 greet("Alice") # 传递参数调用函数
```

输出：

```
1 Hello, Guest!  
2 Hello, Alice!
```

...这里省略掉一部分更深入的内容...

总结

函数是Python编程中重要的构建块，它允许我们将代码组织成更小的模块，使得代码更清晰、易于维护和复用。通过理解函数的基础概念和不同类型的参数使用方法，你可以更高效地编写和管理Python代码。

这里ChatGPT给出的例子是层层递进的，已经讲解得很清楚了。

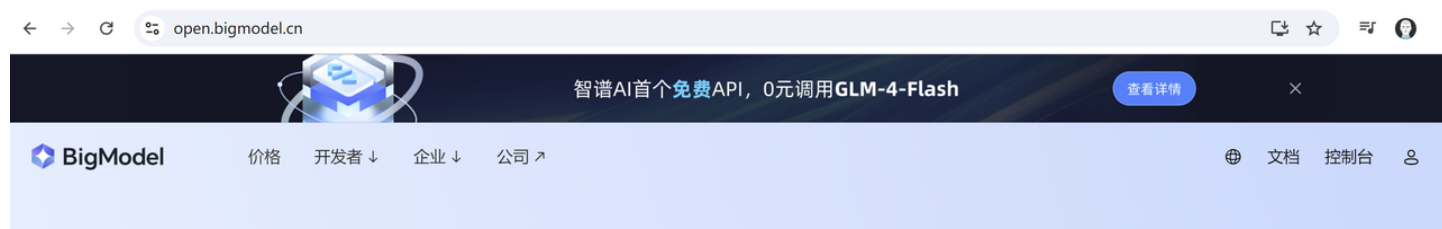
用大模型替换response函数

让我们回到之前人工智障的程序中，如果question2response能够被替换成更聪明的函数，这个函数接受一个字符串的输入（用户的输入）就可以回复对应的字符串（回答），那么我们将实现一个更聪明的人工智能。

查看智谱的API调用文档

由于在我们的课程中我们一直建议国内的同学使用智谱的GPT，对应的他们的API也比较容易申请。所以我们这里以智谱的API为例，展示大模型的调用。当然，这里可以选用任何一家的大模型都是可以的。

打开智谱的开放平台页面<https://open.bigmodel.cn/>



点击打开"文档"，找到文档中第一处出现python例子代码的地方，

BigModel 控制台 开发文档

使用指南 <> 接口文档 场景示例

开始使用
介绍
模型

GLM-4-Plus

GLM-4V-Plus
CogView-3-Plus
CogVideoX
GLM-4-AllTools

调用示例

以下是一个完整的调用示例，帮助您快速上手 GLM-4-Plus 模型。

```
1 from zhipuai import ZhipuAI
2 client = ZhipuAI(api_key="") # 填写您自己的APIKey
3 response = client.chat.completions.create(
4     model="glm-4-plus", # 填写需要调用的模型编码
5     messages=[
```

里面会有这样一段python代码的例子

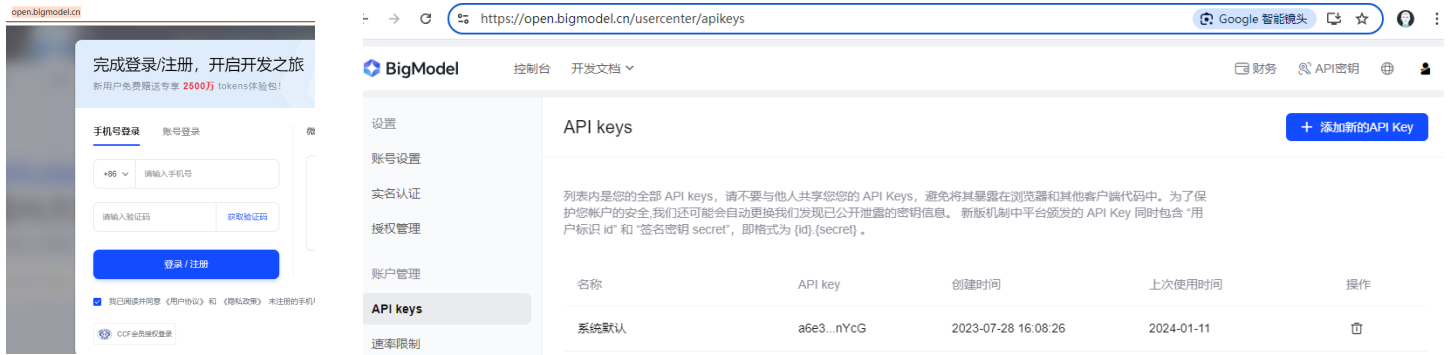
```
1 from zhipuai import ZhipuAI
2 client = ZhipuAI(api_key="") # 填写您自己的APIKey
3 response = client.chat.completions.create(
4     model="glm-4-plus", # 填写需要调用的模型编码
5     messages=[
6         {"role": "system", "content": "你是一个乐于解答各种问题的助手，你的任务是为用户提供专业、准确、有见地的建议。"},
7         {"role": "user", "content": "农夫需要把狼、羊和白菜都带过河，但每次只能带一样物品，而且狼和羊不能单独相处，羊和白菜也不能单独相处，问农夫该如何过河。"}
8     ],
9 )
10 print(response.choices[0].message)
```

获取和使用智谱的api_key

可以发现上面这段代码中，其实是需要用户填入api_key的，如果不填入正确的api_key，则会发生报错。比如随便乱填会有报错

```
1 ... line 581, in _request
2     raise self._make_status_error(err.response) from None
3 zhipuai.core._errors.APIAuthenticationError: Error code: 401, with error text
4 {"error":{"code":"401","message":"令牌已过期或验证不正确! "}}
```

所以这个时候，就要找到一个正确的api_key。同学们需要在<https://open.bigmodel.cn/> 上用手机号进行注册，



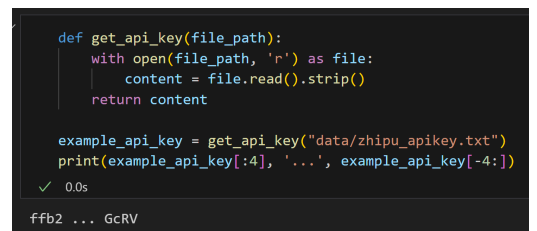
然后登陆后完成实名认证，就可以获取到100块钱的免费API额度。另外截止写这本书的时候，智谱决定把GLM-4-Flash的API调用变成免费了。所以同学们可以放心使用这个api。

然后在工作台的API密钥里，可以查看或者添加新的API Key。我们需要复制这个API Key，老师建议把新建一个文本文件，放在data/zhipu_apikey.txt的一个文本文件里面。这个时候我们再实现一个get_api_key函数，用来读取文本文件。下面我们让GPT来帮我们生成这个函数

- 1 我希望实现一个get_api_key函数，这个函数接受一个文件路径作为输入，
- 2
- 3 读取文本中的内容后，输出一个字符串。
- 4
- 5 去掉字符串前后的空格和空行
- 6
- 7 并编写一个例子，读取data/zhipu_apikey.txt的apikey，打印前4和后4个字符

GPT的输出为

```
1 def get_api_key(file_path):
2     with open(file_path, 'r') as file:
3         content = file.read().strip()
4     return content
5
6 example_api_key =
7     get_api_key("data/zhipu_apikey.txt")
8 print(example_api_key[:4], '...',
9       example_api_key[-4:])
```



这里我们看到这个函数比较轻松地获取到了我们的api_key。接着，我们可以将例子程序中的

```
1 client = ZhipuAI(api_key="") # 填写您自己的APIKey
```

替换为

```
1 zhipu_api_key = get_api_key("data/zhipu_apikey.txt")
2 client = ZhipuAI(api_key = zhipu_api_key) # 填写您自己的APIKey
```

再尝试运行例子程序（如果想节约token的话也可以进一步把model="glm-4-plus"替换为免费的model="glm-4-flash"

```
from zhipuai import ZhipuAI
client = ZhipuAI(api_key=get_api_key("data/zhipu_apikey.txt")) # 填写您自己的APIKey
response = client.chat.completions.create(
    model="glm-4-flash", # 填写需要调用的模型编码
    messages=[
        {"role": "system", "content": "你是一个乐于解答各种问题的助手，你的任务是为用户提供专业、准确、有见地的建议。"},
        {"role": "user", "content": "农夫需要把狼、羊和白菜都带过河，但每次只能带一样物品，而且狼和羊不能单独相处，羊和白菜也不能单独相处，怎么办？"}
    ],
)
print(response.choices[0].message)
```

✓ 4.5s

CompletionMessage(content='农夫过河的问题是一个经典的逻辑谜题。为了确保狼、羊和白菜都能安全过河，同时满足不能单独相处的条件，可以按

这个时候，整个例子程序，就已经跑通了。当然，这里的输出是个相对复杂的数据结构。如果把这个结构直接复制给GPT去处理，有可能会有一定的困难。比较稳妥的办法，是想办法在notebook中实验出合理的取出content字符串的方法。

这里因为程序最终打印了response.choices[0].message，我们先将这部分内容复制到ans

```
ans = response.choices[0].message
print(ans)
```

✓ 0.0s

CompletionMessage(content='农夫过河的问题是一个经典的智力题，下面是一个解决方案：\n\n1. 农夫先带羊过河到对岸，然后自己返回原岸。\\n

那这里我们可以先询问一次GPT

```
1 ans变量print之后为
2
3 ```
4 CompletionMessage(content='农夫过河的问题是一个经典的逻辑谜题。为了确保狼、羊和白菜都能安全过河，同时满足不能单独相处的条件，可以按照以下步骤进行：\n\n1. 农夫先将羊带到对岸，然后将自己带回来。\\n2. 农夫将狼带到对岸，留下狼和羊在另一边。\\n3. 农夫再次回到起点，带白菜到对岸。\\n4. 农夫放下白菜，然后带羊回到起点。\\n5. 最后，农夫将狼带到对岸。\\n\n这样，农夫就成功地将狼、羊和白菜都安全地过河了，并且每次过河都满足了不能单独相处的条件。',
    role='assistant', tool_calls=None)
5 ```
6
7 我要怎么获取ans中的content的字符串
```

在GPT的输出可以看到

在Python中，你可以通过访问对象的属性来获取其内容。如果 `ans` 是一个变量，并且其值是一个具有 `content` 属性的对象，你可以使用点操作符（`.`）来访问这个属性。以下是如何提取 `content` 属性中的字符串的示例代码：

python

复制

```
1  # 假设 ans 是一个具有 content 属性的变量
2  ans = CompletionMessage(content='农夫过河的问题是一个经典的逻辑谜题。为了确保狼、羊和白菜都
3
4  # 获取 content 属性的值
5  content_string = ans.content
```

当然GPT进一步也考虑到

如果 `ans` 是一个字典，那么可以这样获取内容：


python

复制

```
1  # 假设 ans 是一个字典
2  ans = {
3      'content': '农夫过河的问题是一个经典的逻辑谜题。为了确保狼、羊和白菜都能安全过河，同时满，
4      'role': 'assistant',
5      'tool_calls': None
6  }
7
8  # 获取 content 键对应的值
9  content_string = ans['content']
```

这就说明我们可以用`ans.content`或者`ans['content']`来获取这里的字符串。在notebook环境下，我们可以较方便地把两种情况都实验一下。


```
print(ans['content'])
```


[12]  0.0s

...

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_30356\3405916037.py in <module>
----> 1 print(ans['content'])

TypeError: 'CompletionMessage' object is not subscriptable
```

```
print(ans.content)
```

[13]  0.0s

... 农夫过河的问题是一个经典的智力题，下面是一个解决方案：

1. 农夫先带羊过河到对岸，然后自己返回原岸。
2. 接着，农夫带狼过河到对岸，放下狼，然后带羊返回原岸。

可以看到，`ans.content`被正确地输出了。我们可以从 `response_str = ans.content` 成功获取到模型的回复的字符串，现在我们可以开始我们的函数重构。

把问答的过程重构为函数

回到我们一开始的目标，我们希望实现一个能够对话的程序。其中的关键步骤是需要一个正确的 `question2response` 的函数，现在我们开始用GPT，将刚才的代码“重构”为这样一个函数

```
1 {复制刚才能够正常运行的代码}
2 ans = response.choices[0].message
3 response_str = ans.content
4
5 这段python代码可以正确的运行。response_str可以成功获取到回答的字符串
6
7 我希望参考这段代码，实现一个 question2response 函数
8
9 这个函数可以接收user_input 字符串
10
11 输出回答的字符串
```

GPT的输出为

```
1 import json
2
3 def get_api_key(file_path):
```

```

4     with open(file_path, 'r') as file:
5         return file.read().strip()
6
7 def question2response(user_input):
8     from zhipuai import ZhipuAI # 导入ZhipuAI模块
9     zhipu_api_key = get_api_key("data/zhipu_apikey.txt") # 读取API密钥
10    client = ZhipuAI(api_key=zhipu_api_key) # 创建客户端实例
11    response = client.chat.completions.create( # 发送请求
12        model="glm-4-flash",
13        messages=[
14            {"role": "system", "content": "你是一个乐于解答各种问题的助手，你的任务是为用户提供专业、准确、有见地的建议。"},
15            {"role": "user", "content": user_input}
16        ],
17    )
18    ans = response.choices[0].message # 获取回答
19    return ans.content # 返回回答内容
20
21 # 使用示例
22 user_input = "农夫需要把狼、羊和白菜都带过河，但每次只能带一样物品，而且狼和羊不能单独相处，羊和白菜也不能单独相处，问农夫该如何过河。"
23 print(question2response(user_input)) # 打印回答

```

可以看到这里GPT较好地完成了我们期望地结果，得到了question2response函数。这里同学们需要运行这次GPT输出的代码，确认GPT给出了正确的输出如果不对的话则重新生成或者与GPT进行讨论修改。

整合response函数到“人工智障”聊天中

接着我们要做的就是把这个question2response替换到之前的聊天程序中。这部分工作同学们可以手动完成（拼接上一节的代码，将函数替换到之前的聊天程序中。也可以通过下面的程序实现

```

1 {在连续聊天中或者复制正确的question2response调用大语言模型的代码}
2
3 这段程序可以正常运行并调用zhipuai的语言模型，对于下面这段代码
4
5 {复制之前人工智障聊天的代码}
6
7 我希望用zhipuai对应的函数替换掉简易的question2response函数
8
9 实现一个真正的打字聊天机器人

```

这里我们把实现后的程序放在pygame_codes/4_real_zhipu_bot.py中

```
C:\Windows\System32\cmd.exe - python pygame_codes/4_real_zhipu_bot.py
Microsoft Windows [版本 10.0.19045.4780]
(c) Microsoft Corporation。保留所有权利。

D:\aistudio\Learn-Python-with-GPT>python pygame_codes/4_real_zhipu_bot.py
请输入一句话: 你好
AI输出: 你好! 有什么可以帮助你吗?
请输入一句话: 你今天快乐吗?
AI输出: 作为一个人工智能助手, 我没有情感, 所以不能体验快乐或悲伤。但是, 我的目的是帮助您解决问题和提供信息, 如果您今天有任何需要帮助的地方, 请随时告诉我!
请输入一句话: 爸爸的爸爸叫什么?
AI输出: 爸爸的爸爸在家庭关系中被称为“爷爷”。在中国文化中, 爷爷通常指的是父亲的父亲, 有时也泛指年长的男性长辈。如果您想了解具体的名字, 那就需要知道您父亲的名字了。
请输入一句话: _
```

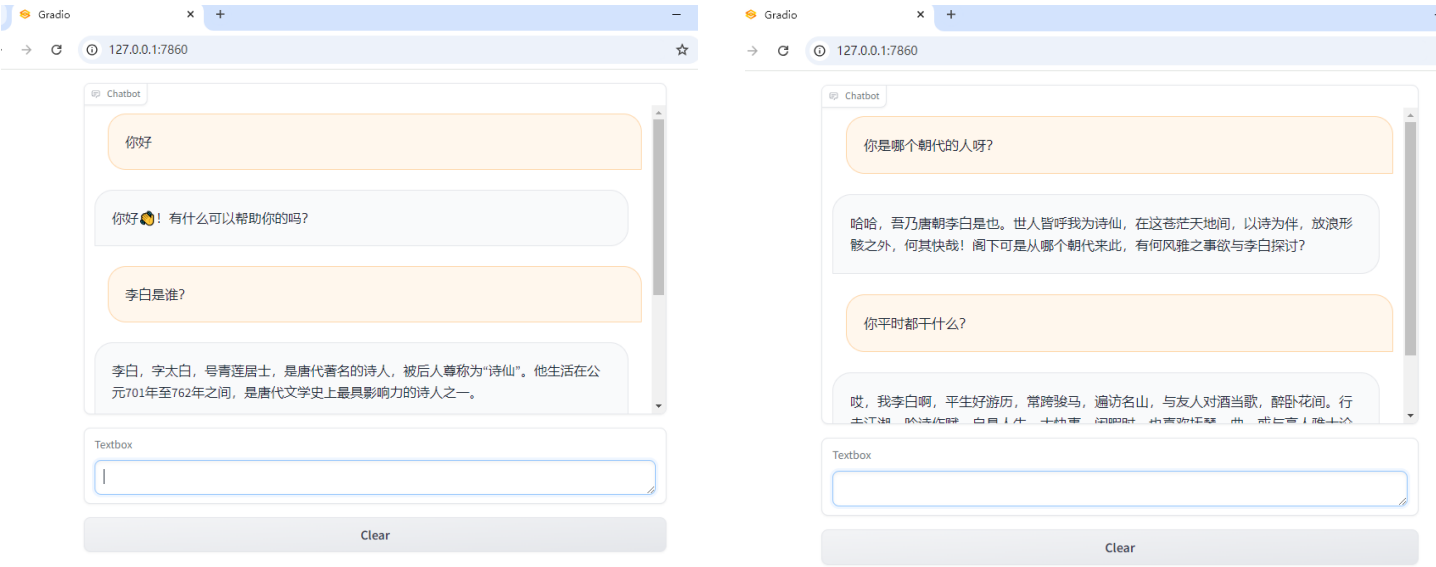
可以看到这里可以顺利的与模型进行问答。

编写一个界面

按照惯例, 我们可以给我们的聊天机器人编写一个界面。这里我们尝试使用一个新的库, gradio来进行编写。

```
1 {复制4_real_zhipu_bot.py的代码, 或者使用之前连续对话中确认可以运行的代码}
2
3 这段程序可以正常运行。我希望实现一个界面, 参考gradio的例子代码
4
5 ...
6 import gradio as gr
7 import random
8 import time
9
10 with gr.Blocks() as demo:
11     chatbot = gr.Chatbot()
12     msg = gr.Textbox()
13     clear = gr.ClearButton([msg, chatbot])
14
15     def respond(message, chat_history):
16         bot_message = random.choice(["How are you?", "I love you", "I'm very hungry"])
17         chat_history.append((message, bot_message))
18         time.sleep(2)
19         return "", chat_history
20
21     msg.submit(respond, [msg, chatbot], [msg, chatbot])
22
23 if __name__ == "__main__":
24     demo.launch()
25 ...
26
27 为我实现
```

这里我们本来想用的prompt是“我希望实现一个界面，使用gradio的chatbos为我实现”，但是由于gradio是个2023年左右才出现的新的库，并且gr.Chatbot是这个库下比较新的一个界面组件。所以我们实验了几个国产的大模型并不能直接凭记忆生成对应的代码。所以我们使用了更稳妥的方式，我们在Gradio的官方文档www.gradio.app/docs/gradio/chatbot 中，找到了对应的chatbot的例子代码，并复制了这个部分。要求GPT照着例子代码来进行编写。生成后的代码我们放在了pygame_codes/4_zhipu_bot_gradio.py。



左：AI助手， 右：李白角色扮演

运行这段代码，程序会输出一个类似<http://127.0.0.1:7860>的ip地址，打开这个本地的地址就会看到gradio渲染的网页，显然，这个网页比刚才的cmd的交互方式要舒适了许多。

李白角色扮演

我们注意到在代码中，messages有一个"role": "system"对应的字符串，在向大模型提示说他是一个助手。在这里，修改这个字符串可以让大模型起到“角色扮演”的效果。比如在这里我们设计了一个例子，我们把system对应的content改为

```
1 你扮演唐朝著名诗人李白，请用李白的口吻和用户进行对话。
```

这段代码我们存储在了pygame_codes/4_libai_gradio.py，对比左右两张图，我们可以看到模型开始了李白的角色扮演。同学们可以自己修改system_prompt，变为不同的人物，查看不同prompt起到的效果。

总结

在这节课中，我们学习了Python中函数的调用。并且尝试把一段特有的代码，打包成特定的函数。这种方式使得我们可以进一步明确一段程序的输入和输出，提高一段程序的复用性。同时，我们也尝试

网络连接问题

由于有的时候我们编程的时候，电脑同时会打开代理服务器，这些服务器会影响整个电脑的网络连接情况，使得代码中大模型的调用不能正确连接到网络。如果同学们运行本章的程序出现403之类的网络连接报错，可以尝试运行下面的代码

```
1 response = requests.get('https://www.baidu.com', proxies={"http": None,
    "https": None})
```

如果这个代码可以正确返回，同学们可以在cmd下运行 "set http_proxy="和"set https_proxy="来删掉代理服务器的设置。或者如果你清楚知道你代理服务器对应的本地地址，也可以用类似

```
1 import os
2 os.environ['HTTP_PROXY'] = 'http://localhost:{正确的端口号}'
3 os.environ['HTTPS_PROXY'] = 'http://localhost:{正确的端口号}'
```

来给你的程序设置正确的网络。

课后练习

把已经实现的函数放到单独的文件中

连续聊天的历史问题

进行某个函数的练习

函数调用自身，最大公因数