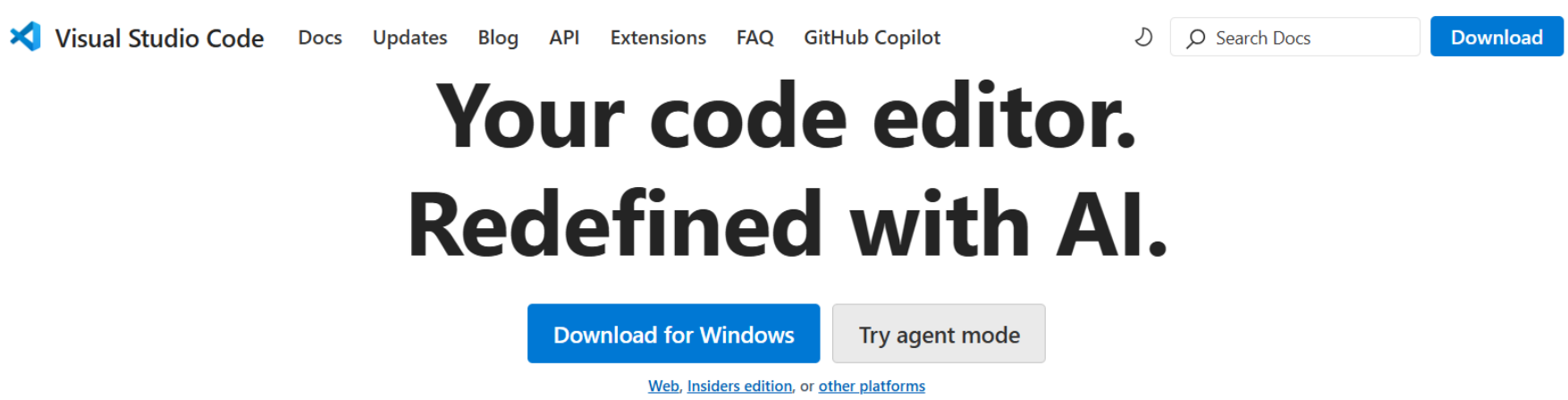
Python简单入门

> 此文档于2025.5.30，端午节前夕的夜晚开始编写。作为AI时代下对计算机技术的一次探索留念。另一个期待是初学者能够通过阅读这份文档对AI与Python发生兴趣。如果疑惑请直接问AI或者联系作者！AI时代的学习，请把与AI交互放在第一位。作者微信yes_smile_peace。
>
> 文档参考：
>
> 公众号Python卡皮巴拉
>
> 廖雪峰Python教程https://liaoxuefeng.com/books/python/introduction/index.html
>
> 白月黑羽Python教程
>
> https://www.byhy.net/py/lang/basic/01/

安装环境

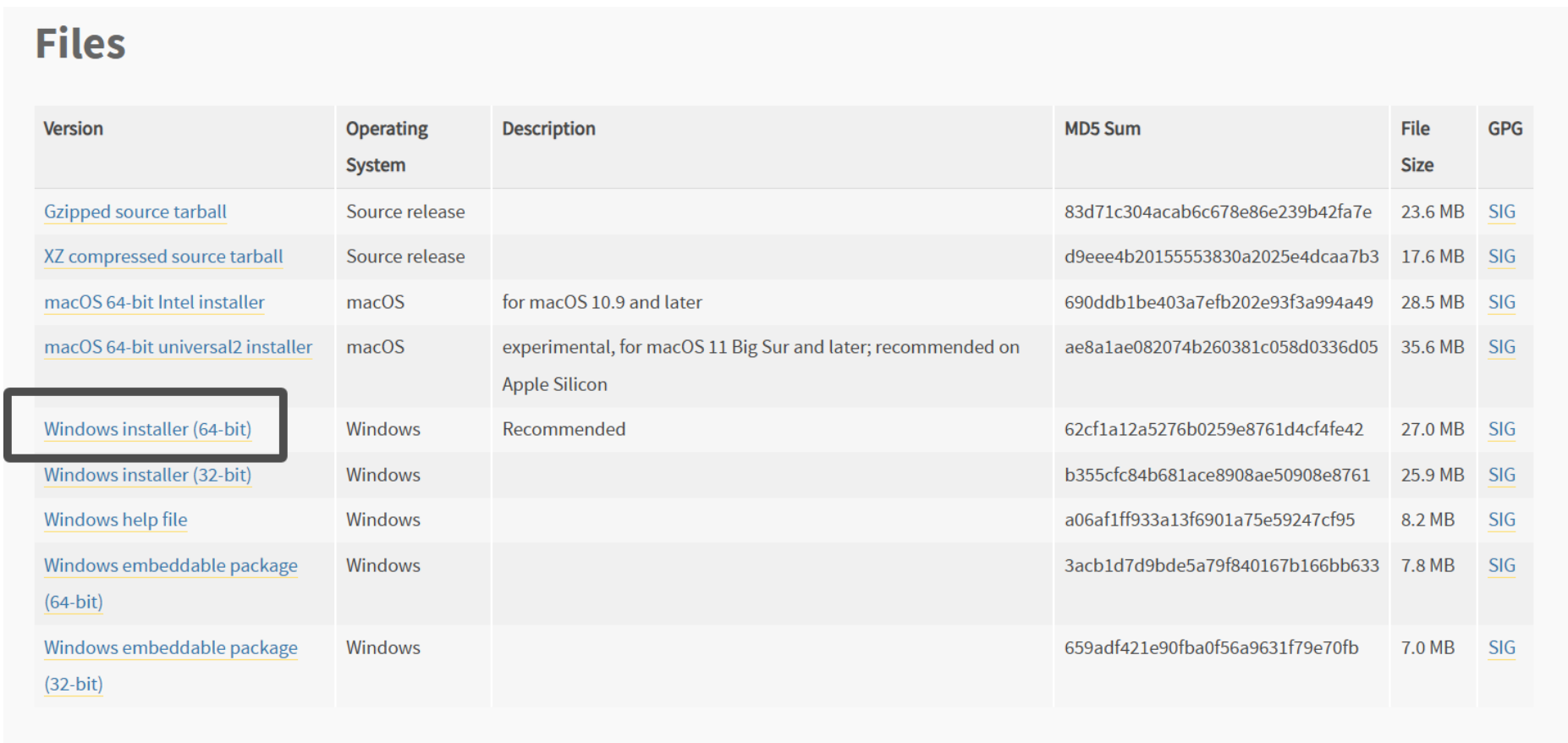
代码编辑器，点击download for windows下载代码编辑器

https://code.visualstudio.com/



Python3.8.10版本，点击Windows installer (64-bit)

https://www.python.org/downloads/release/python-3810/



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		83d71c304acab6c678e86e239b42fa7e	23.6 MB	SIG
XZ compressed source tarball	Source release		d9eee4b20155553830a2025e4dcaa7b3	17.6 MB	SIG
macOS 64-bit Intel installer	macOS	for macOS 10.9 and later	690ddb1be403a7efb202e93f3a994a49	28.5 MB	SIG
macOS 64-bit universal2 installer	macOS	experimental, for macOS 11 Big Sur and later; recommended on Apple Silicon	ae8a1ae082074b260381c058d0336d05	35.6 MB	SIG
Windows installer (64-bit)	Windows	Recommended	62cf1a12a5276b0259e8761d4cf4fe42	27.0 MB	SIG
Windows installer (32-bit)	Windows		b355cfc84b681ace8908ae50908e8761	25.9 MB	SIG
Windows help file	Windows		a06af1ff933a13f6901a75e59247cf95	8.2 MB	SIG
Windows embeddable package (64-bit)	Windows		3acb1d7d9bde5a79f840167b166bb633	7.8 MB	SIG
Windows embeddable package (32-bit)	Windows		659adf421e90fba0f56a9631f79e70fb	7.0 MB	SIG

打开文件并保存成后缀名是.py的文件，通过vscode打开。

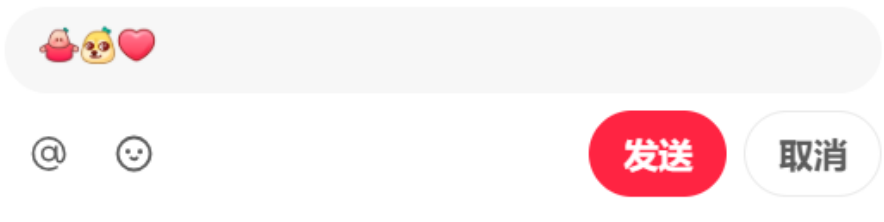
一、语言和函数

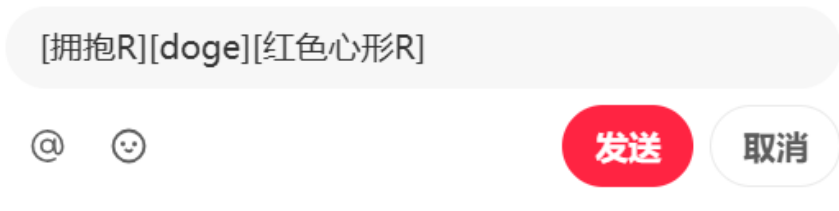
写在前面

这一章主要学习什么叫做语言，接着我们将直接开始创建函数，然后再用它来创建那些通常被认为是函数的预备知识的东西。通过这样前后倒置，我们会发现有些“预备知识”并不是说只有具备了那些所谓的预备知识之后才能创造出来，而是没有函数就无法彻底理解那些“预备知识”。

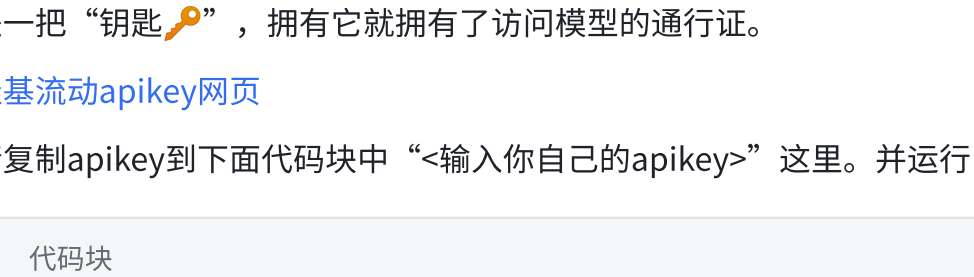
语言

什么是语言？**语言就是主体交互的时候共同遵守的约定。**尝试在小红书输入以下评论：





发现当点击发送时，会发现两者显示的效果一致，这是为什么？原因是第一个“拥抱狗头爱心”是“人”写给“人”看的，第二个则是“人”写给“机器”看的，它遵守了人与机器的约定。



我们所做的一切都能用自然语言去描述！请记住这一点。

只不过在这里，我们会学习到如何让机器听懂人说话。比如上述的例子，当人输入[doge]时，机器就会把[doge]翻译成狗头。这和直接输入狗头没什么区别。

函数

吞进去，吐出来

> 机器能做各种事情。面包机吞进面包的原材料，吐出面包。烤箱吞进各种东西，然后以更热的温度吐出来。无论如何，数学家决定用“函数”这个奇怪的词描述吞数字然后吐出其他数字的机器。更好的名字也许是……什么都行。我们就称它们为“机器，”等我们习惯了这个概念，我们也偶尔会称它们为“函数”。但只是偶尔。

现在来举一个最简单的例子说明函数是什么。也许你看过那些告诉你开始学习代码时，需要首先`print("hello world")`的教学。但在这里，这是之后才需要做的事。如果你感到疑惑，只需要重新回顾函数的定义：**函数就是吞进去东西然后再吐出来东西的玩意。**

插入例子：requests模块

请访问硅基流动官网并获取一个apikey。

什么是apikey？轨迹流动官网提供了许多AI模型的“接口”，但是没有“钥匙🔑”（权限）则无法“访问”这些“接口”。apikey是一把“钥匙🔑”，拥有它就拥有了访问模型的通行证。

硅基流动apikey网页

请复制apikey到下面代码块中“<输入你自己的apikey>”这里。并运行以下代码块。

代码块

```
1   def AI_talk():
```

```
    question = input("AI: 请提问:")
    url = "https://api.siliconflow.cn/v1/chat/completions"
    payload = {
        "model": "Qwen/Qwen3-8B",
        "messages": [
            {
                "role": "user",
                "content": f"{question}"
            }
        ],
        "stream": False,
        "max_tokens": 512,
        "enable_thinking": False,
        "thinking_budget": 4096,
        "min_p": 0.05,
        "stop": None,
        "temperature": 0.7,
        "top_p": 0.7,
        "top_k": 50,
        "frequency_penalty": 0.5,
        "n": 1,
        "response_format": {"type": "text"},
        "tools": [
            {
                "type": "function",
                "function": {
                    "description": "<string>",
                    "name": "<string>",
                    "parameters": {},
                    "strict": False
                }
            }
        ]
    }
    headers = {
        "Authorization": "Bearer <输入你自己的apikey>",
        "Content-Type": "application/json"
    }

    response = requests.request("POST", url, json=payload, headers=headers).text
    print(response)
if __name__ == "__main__":
    AI_talk()
```

程序会向用户呈现：AI: 请提问。此时用户输入一个问题，程序会输出：

代码块

```
AI: 请提问:你是谁？你从哪里来？你到哪里去？
{"id":"0197242a29961b2b61b07bb28bc4330d","object":"chat.completion","created":1748658432,"model":"Qwen/Qwen3-8B","choices":[{"index":0,"message":{"role":"assistant","content":"我是一个大型语言模型，名为Qwen，由通义实验室开发。我从大量的文本数据中学习，能够理解和生成自然语言，帮助用户解答问题、创作文字、编程等。我的目标是为用户提供有用的信息和帮助，促进人与机器之间的有效沟通。至于我“到哪里去”，这取决于用户的需求和使用场景，我致力于在各种任务中提供支持和协助。"},"finish_reason":"stop"}],"usage":{"prompt_tokens":137,"completion_tokens":85,"total_tokens":222},"system_fingerprint":""}
```

恭喜你，已完成你的第一个代码！代码的内容看不懂？没关系，你只需要知道它是一个吞进去用户输入，并且吐出来用户输出的一个函数就够了。还记得吗？当你有困惑时，回顾函数的定义。

函数就是吞进去东西然后再吐出来东西的玩意。

你现在只需要关心它吞进去什么，吐出来什么。在这个函数里：

它吞进去的是：

代码块

```
1   你是谁？你从哪里来？你到哪里去？
```

它吐出来的是：

代码块

```
1   {"id":"0197242a29961b2b61b07bb28bc4330d","object":"chat.completion","created":1748658432,"model":"Qwen/Qwen3-8B","choices":[{"index":0,"message":{"role":"assistant","content":"我是一个大型语言模型，名为Qwen，由通义实验室开发。我从大量的文本数据中学习，能够理解和生成自然语言，帮助用户解答问题、创作文字、编程等。我的目标是为用户提供有用的信息和帮助，促进人与机器之间的有效沟通。至于我“到哪里去”，这取决于用户的需求和使用场景，我致力于在各种任务中提供支持和协助。"},"finish_reason":"stop"}],"usage":{"prompt_tokens":137,"completion_tokens":85,"total_tokens":222},"system_fingerprint":""}
```

数据类型

当我们理解了——函数的本质——一个吞吐东西的玩意。现在可以继续来研究研究吞进去什么，吐出来什么了。

函数吞进去的东西有种种类型，吐出来的东西也有种种类型，在Python中，可以称这些类型叫做**数据类型**。不同类型吞吐东西的类有：

代码块

```python
def data_type():
    print(type(None))
    print(type("字符串"))
    print(type(123))
    print(type(1.23))
    print(type(True))
    print(type(False))
    print(type(["我爱你", "我超级爱你", 520]))
    print(type(("我爱你", "我超级爱你", 520)))
    print(type({"我说": "我爱你", "I say": "I love you"}))
    print(type({1, 2, 2}))
```

这个函数吐出来是：

代码块

```
<class 'NoneType'>
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'bool'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
<class 'set'>
```

定义一个不吞东西的函数

定义一个函数除了要有吞进去和吐出来的东西，这个函数还需要有一个名字。在Python中，给函数起名字的操作是：

代码块

```
def sayhello():
    pass
```

通过return给这个函定义一个吐出来的东西——向用户问好。并用一个变量接收函数吐出来的东西。

代码块

```
def sayhello():
    return "hello!user!"
hi = sayhello()
print(hi)
```

函数吐出来：

代码块

```
hello!user!
```

现在你已经学会了如何定义一个不吞东西的函数，现在我们说说函数如何吞东西。

函数会吞下什么？（参数）

让我们先定义一个好玩的函数，这个函数会调用api并随机返回一段情话。

代码块

```
import requests
def loveword():
    result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
    return result
```

位置参数

现在我们要添加一个功能，即向用户说出情话。

代码块

```
import requests
def loveword(user):
    result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
    return f"机器对{user}说: {result}"
```

这时调用函数，则需要传入参数（用户名字），比如

代码块

```
love = loveword("阿喵")
print(love)
```

函数吐出：

```
机器对阿喵说：给你讲个故事，这故事很长，所以我长话短说，我想你了。
```

默认参数

现在我们要添加一个功能，向两个用户说出情话。其中第二个用户的名字是固定的。叫做旺财。注意，必选参数在前，默认参数在后，否则Python的解释器会报错。

代码块

```python
import requests
def loveword(user, dog = "旺财"):
    result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
    return f"机器对{user}和{dog}说: {result}"
```

打印一下得到：

代码块

```
机器对阿喵和旺财说：想吃甜的，但是我这里没有甜的。我就只好想想你了。
```

可变参数

现在，除了阿喵和旺财，还有许许多多的名字，也就是不知道有多少位置参数？当出现多个未知量的参数的时候怎么做呢？即引入arg可变参数，当位置参数写入完成后，函数会将剩下的参数打包成一个元组传入。

代码块

```python
import requests

def loveword(user, dog = "旺财", *arg):

    print(f"arg的数据类型是: {type(arg)}")

    result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
    print(f"机器对{user}和{dog}说: {result}")

    for i in arg:
        result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
        print(f"机器对{i}说: {result}")
```

调用这个函数，现在可以不受限制的传入参数了。

代码块

```python
loveword("阿喵",
         "来福", "阿福", "球球", "咪咪", "阿咪", "元宝",
         "招财", "进宝", "崽崽", "奶糖", "布丁", "奶酪",
         "糯米", "小米", "大米", "花生", "核桃", "小白",
         "小黑", "小花", "小黄")
```

它的输出为：（注意看，此时的dog = "旺财" 由于是第二个位置参数，已经替换成了"来福"）

代码块

```
arg的数据类型是: <class 'tuple'>
机器对阿喵和来福说：我爱你，如鲸向海，鸟投林，不可避免，退无可退。  ——《黄碧云经典语录》

机器对阿福说：报告一下，我变心了，今天的我比昨天更喜欢你了
```

```
机器对球球说：世间万物皆苦，你明目张胆的偏爱就是救赎。

机器对咪咪说：打印一张你的照片放在口袋，今天我就是身藏巨款的人，你最珍贵

机器对阿咪说：你携风而来，如花绽放；于雪色之间，揽月而上。至此，世间所谓风花雪月，皆与你有关。

机器对元宝说：你知道为什么我长这么高么？ 为什么？ 因为你贴过来的时候就可以听见我的心跳了。

机器对招财说：遇见你之后，我就愈发贪懒了，每日啊，只想着早早的把银河打烊，带着晚归的星星跟月亮全都栖进你眼睛里”

机器对进宝说：第一次看到宇宙,是和你四目相对的时候。

机器对崽崽说：你其有点像天上的月亮，也像那闪烁的星星，可惜我不是诗人，否则当写一万首诗来形容你的美丽

机器对奶糖说：其实我有时候真的分不清对你是什么感情

机器对布丁说：我还是喜欢你，像小时候吃辣条，不看日期。

机器对奶酪说：我可以称呼你为您吗？ 这样我就可以把你放在心上了

机器对糯米说：我还是很喜欢你 长安城内 万丈灯起 执花静候 只等一人来取

机器对小米说：你会弹吉他吗？为什么拨动了我的心弦

机器对大米说：即使我张牙舞爪，也希望在你眼里我是最好的宝贝

机器对花生说：这是什么绝世美少女，万年一遇的颜啊，今天也是仔细体味姐姐盛世美颜的一天。姐姐的眼眸就像珍宝匣里最美的琥珀，是俘获我心的最美的那颗，这样明媚的脸庞是什么仙子颜啊！是真实存在的珍宝吗？我看姐姐是从拉斐尔画里走出来的自带光环的绝世贵族吧！所以上帝才会更偏心一些！

机器对核桃说：风很温柔 水很清澈 花很鲜艳 你很特别.

机器对小白说：今夜江河之源，只亮我的酥油灯，只照我的心上人

机器对小黑说：你的眼睛怎么业务能力这么差？连我喜欢你都看不出来？更差的可能是我的嘴巴，说个喜欢都一直支支吾吾。

机器对小花说：如果你是五彩的糖，那我就当保护你小小的糖纸

机器对小黄说：其实我有时候真的分不清对你是什么感情
```

关键字参数

可变参数用于传递没有参数名的参数，关键字参数用于传递有参数名的参数。

比如，你需要在吐出情话的同时，同时附上一些附加信息：年级一年级, 爱好喝酒

那么就可以通过关键词参数的形式，将有参数名的参数打包成字典传入。

代码块

```python
import requests

def loveword(user, dog = "旺财", **kwargs):

    print(f"kwarg的数据类型是: {type(kwargs)}")
    result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
    print(f"机器对{user}和{dog}说: {result}, 阿喵附加信息:{kwargs}")
```

此时调用函数并传入字典类型：

```
代码块
1   loveword("阿喵", 年级 = "一年级", 爱好 = "喝酒")
```

函数吐出：

```
代码块
1   kwarg的数据类型是: <class 'dict'>
2   机器对阿喵和旺财说: 我听说十三月有你 便穷尽这一生去寻你 漫天柳絮倾覆了我情义，阿喵附加信息:{'年级': '一年级', '爱
    好': '喝酒'}
```

对于可变参数和关键字参数，更多信息可以关注Python卡皮巴拉公众号这篇文章的介绍。

python中的万能参数*args和**kwargs怎么用？抓住这三个应用场景准没错

类

类与实例

string是一个类，"hello world"是这个类的一个实例。猫咪是一个类，阿喵是这个类的一个实例。

类的写作语法如下：

```
代码块
1   class Cat:
2       pass
```

给Cat类创建一个实例：名字叫Amiao

```
代码块
1   Amiao = Cat()
```

实例属性和实例方法

实例属性

由于类可以起到模板的作用，因此，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填写进去。

通过定义一个特殊的__init__方法，在创建实例的时候，就把属性绑上去：注意到__init__方法的第一个参数永远是self，表示创建的实例本身，因此，在__init__方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。

和普通的函数相比，在类中定义的函数只有一点不同，就是第一个参数永远是实例变量self，并且，调用时，不用传递该参数。除此之外，类的方法和普通函数没有什么区别，所以，你仍然可以用默认参数、可变参数、关键字参数和命名关键字参数。

```
代码块
1   class Cat:
2       def __init__(self):
3           self.品种 = "猫"
4           self.颜色 = "white"
5           self.体型 = "小"
```

实例方法

既然类本身就拥有属性，要访问这些属性，就没有必要从外面的函数去访问，可以直接在类的内部定义访问属性的函数，这些访问类的内部属性的函数，我们称之为类的方法。

要定义一个方法，除了第一个参数是self外，其他和普通函数一样。

比如：

代码块

```
class Cat:
    def __init__(self):
        self.品种 = "猫"
        self.颜色 = "白"
        self.体型 = "小"
    def miaomiaojiao(self):
        print(f"{self.颜色}色的{self.体型}体型的{self.品种}正在喵喵叫!")

Amiao = Cat()
Amiao.miaomiaojiao()
```

这个类的miamiaojiao方法吐出：

代码块

```
白色的小体型的猫正在喵喵叫!
```

属性的访问

前面说过：类本身就拥有属性，类的方法可以直接访问这些属性。那么问题来了：同一个类中，一个方法如何访问另一个方法的变量？答案是：将变量提升为这个实例的属性。这样方法就能够通过访问实例属性的方式访问变量。

习题

请根据上述知识设计一个程序，让ai接收一段随机情话并返回结果。

答案（请做完习题后再看！）

代码块

```
import requests

class AiLove:
    def __init__(self):
        pass

    def loveword(self):
        self.result = requests.get(url="https://api.pearktrue.cn/api/jdyl/qinghua.php").text
        print(f"请求情话成功:{self.result}\n")

    def AI_talk(self):
        self.loveword()
        url = "https://api.siliconflow.cn/v1/chat/completions"
        payload = {
            "model": "Qwen/Qwen3-8B",
            "messages": [
                {
                    "role": "user",
                    "content": f"{self.result}"
                }
            ],
            "stream": False,
            "max_tokens": 512,
            "enable_thinking": False,
            "thinking_budget": 4096,
```

```
            "min_p": 0.05,
            "stop": None,
            "temperature": 0.7,
            "top_p": 0.7,
            "top_k": 50,
            "frequency_penalty": 0.5,
            "n": 1,
            "response_format": {"type": "text"},
            "tools": [
                {
                    "type": "function",
                    "function": {
                        "description": "<string>",
                        "name": "<string>",
                        "parameters": {},
                        "strict": False
                    }
                }
            ]
        }
        headers = {
            "Authorization": "Bearer <输入你自己的apikey>",
            "Content-Type": "application/json"
        }
        print(f"AI收到情话: {self.result}\n正在分析...\n")
        AIresult = requests.request("POST", url, json=payload, headers=headers).text
        return f"AI返回结果是: \n{AIresult}"

if __name__ == "__main__":
    love = AiLove()
    while True:
        result = love.AI_talk()
        print(result)
```