

Git的使用：首先安装git，进入文件夹是在所要建立git仓库的文件夹，然后打开Git Bash Here。先配置用户名，`git config --global user.name "your name"`，再配置邮箱，`git config --global user.email "youremail@github.com"`；再`git add`加上要添加的文件名，`git commit -m "说明"`，`git remote add origin https://github.com/.../.git`，建立远程仓库的连接，再推送到远程分支，即`git push -u origin master`（这是第一次提交改变）。

git配置命令：`git config --list`列出当前配置，拉取远程分支为`git pull origin master`，其中`git add`为放到缓存区，`git commit -m " "`则是提交到本地仓库，`git push`为推送到远程。`git branch`创建分支，`git branch -d testing`删除分支，`git checkout`为切换分支，`git merge`为融合分支，`git stash`为做一个暂存区，`git tag`为打表标记，`git branch -v`输出创建时间，`git branch -merged`查看已融合分支，`git branch --no-merged`为查看哪些分支没有融合，`git status`并查看状态，`git log`查看日志，`git diff --cached`为比较暂存区和本地仓库不同，`git log ↑-2`显示最近的两次更新。

使用VS code和VS是都可以运行C#的，其中VS code要添加插件C#，C# extension等，然后装.NET Core SDK，命令行输入dotnet 九检测，使用`dotnet new console -o 项目路径`，然而我下载.net上面提示检测不到，连终端都打不开了，初步估计是VS code当时下载得有问题。F5是启动调试，五行提示了找不到该目录下的dotnet文件，该目录指工作目录，就越来越迷了。甚至在everything里面查找了powershell.exe路径，修改`settings.json`，结果设置文件里面显示的完全不一样了。

使用VS运行C#，File→new→project→C#，windows，控制台，输入项目名，自动有个“Hello world”代码还是挺直接的。

正则表达式之贪婪匹配和非贪婪匹配：.？意味着贪婪匹配，.？意味着非贪婪匹配。比如：`import re`

解决冲突合并，^{git pull} git pull --rebase origin master

s='hello1234567 world'

res=re.match('he.*(\d+).*\1\1\1\s', s)

print(res.group(1))

只配到'7'，因为.头尽量匹配 多的字符，如果加?则匹配到'1234567'

numpy的练习题讲解：arr[arr % 2 == 1] 抽选出奇数，y=np.where(x>3) 直接输出。print(np.where(arr % 2 == 1, x=-1, y=arr))，那么a=np.arange(10), reshape((2,-1)), b=np.repeat(1,10), reshape(2,-1), h=np.concatenate([a,b], axis=0), t=np.concatenate([a,b], axis=1), print(h), print(t)

a=np.array([1,2,3]), b=np.repeat(a,3), c=np.tile(a,3), print(b), print(c), b为array([1,1,1,2,2,2,3,3,3])

repeat 3倍九维都是拉平了复制，c为[1,2,3,1,2,3,1,2,3]

array([[2,3,2,2,1]]): 编码：arr=np.random.randint(1,4, size=6), [arr[:,None]=np.unique(arr)].view(int8), print(arr)., [2, 表示一维 arr[:,None] 可将原本一维的转为二维 利用广播机制]。

Pandas 可用于分析时间序列，开源 anaconda自带 pandas 主要数据结构 series 和 DataFrame, Series - 维

import pandas as pd, sd={} # 是一个字典，只是 pd.Series(sd) 可转化为字典，和字符串，即等价，一对

键对。DataFrame 是一表格型数据结构。a.to_dict() 转换为字典。frame2.iloc[1] 拿出第一行，

frame2.at['one', 'year'], obj2=obj.reindex(['a','b','c','d','e']) 重新
排序。obj3=pd.Series(['blue','purple','yellow']), index=[0,2,4], obj3.reindex
(range(6), method='ffill') 直接填充两端。obj2.drop('a') 删除一行。axis=1 (删除)
列。切片，赋值过滤：支持文本索引，data[2] 切片切行，不可切成其他的。

作业讲解：append 是加一个数，extend 是加一个列表。frame=pd.DataFrame(np.random.rand(4,3), columns=['b','c','d'], index=['Utah','Ohio','Texas','...']).f=lambda x:

$x_{\max} - x_{\min}$, frame.apply(f), apply 方法用的较多的。
pandas过滤筛选：CSV文件用notepad 打开是用';'分隔，euro12.info()为信息描述，要抽某
列就用euro12.goals^{列名}，euro12 = pd.read_csv(' .csv', sep=';'), discipline.sort_values
(['Red Cards', 'Yellow Cards'], ascending=False), euro12[euro12.Team.str.startswith('G')]
euro12.Team.isin([, ,])

DataFrame 按列删除 del baby_names[列名], names=baby_names.groupby('name').sum()
value_counts() → 计数 names.Count.idxmax() 并为出现频率最大, names[names.Count
== names.Count.median()] 出现为中位数。

nlp (自然语言处理) 属于AI一个方向, natural language processing, 比如文本朗读 / 语音合成, 语音识别, 教自动分词, 词性标注, 句法分析^{知识图谱}, 自然语言生成^{机器自动生成文本}, 文本分类^{机器学习}, 语义理解^{文本生成模型}, 问答系统^{对话系统}, 推荐系统^{推荐系统}, 信息检索^{百度贴吧}, 信息抽取^{Information extraction}, 语义图谱^{text-profiling}, 文字纠错^{百度贴吧}, 问答系统^{机器翻译}, 自动摘要^{百度百科}, 金句^{金句识别}, 金句^{自动摘要}, 文字蕴涵^{百度百科}。

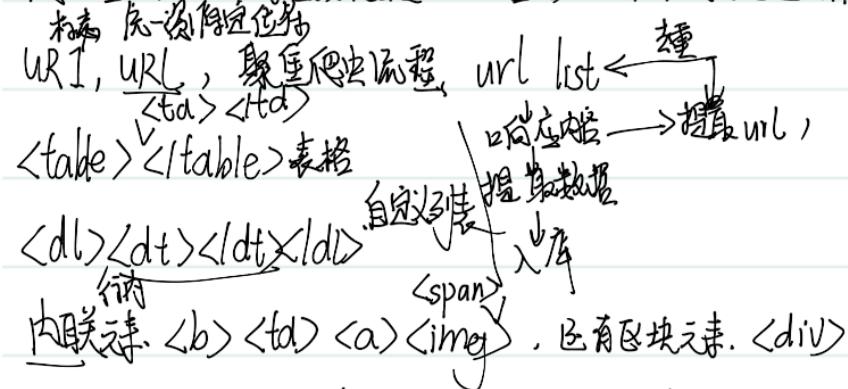
HanLP 工具, 研究算法精确度高于工具。

import pyhanlp 从汉乐网从 jupyter 安装，再找 jdk。← HanLP 环境搭建
HanLP 环境搭建(二) CP37 安装 python 3.7, 命令行 python 得到版本。新技术要看论文。github上有 hanlp 项目 提供校分词, 词性标注, 命名实体识别, 语义推断, 自动摘要, 查哈工nlp, 这个平台也使用, 部分开源
Stanford nlp 也支持中文, 开源(不太佳), jieba 是比较简单使用, hanlp import * 最好不要用, hanlp import JClass,
HanLP, print(HanLP.segment(' ')), PerceptronLexicalAnalyzer=JClass('com.hanlp.model.perception.
PerceptronLexicalAnal')).

爬虫：只要浏览器能做，原则上爬虫都能做。<html>是根元素。
 换行, font-family:
arial 字体, font-size: 20px^{字体大小}.

text-align: center 居中, (3).55

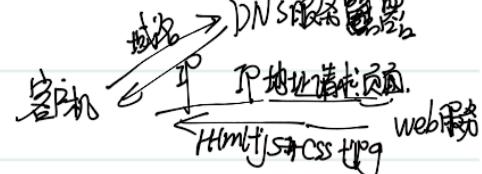
爬虫基本概念: 可以将 HTML 转为数据, 打开一个网页通过 HTTP 协议, 向 B 端发 HTML 文档, 进行文档遍历.



<form>
First name: <input type="text" name="firstname">
Last name: <input type="text" name="lastname">
</form>

Xpath 可用来提取数据 red 定义 url, 由行起始, robots.txt 是根据文件内容 → 允许爬的范围 可遵守
协议 eg https://www.taobao.com/robots.txt. 如果爬超出范围的 为非法的。

Headers 包含 status code, request method, etc.



请求
4 部分
1. 请求方法
2. 请求网址
3. 请求头 (request headers)
4. 请求体 (Request body)

{ GET 会体现在 url,
POST 包含在表单里, }

cookie: 为辨别用户会话进行跟踪, 有值且用户名为数据 主要功能维护前访问会话。

200 成功 404 未找到. Content-type, Server, Set-cookie, etc., 响应头, 500 服务器内部错误

一般不用 urllib;
coding = utf-8

import urllib.request

response = urllib.request.urlopen('https://www.python.org')

print(response.read().decode('utf-8'))

除 get
还有其他方法, post, put, delete 等

主要讲 requests 库
coding = utf-8
import requests
r = requests.get('https://www.baidu.com/')

print(type(r))

print(r.status_code) # 200 为请求成功

print(type(r.text)) # (!DOCTYPE html)

print(r.text)

print(r.cookies)

读取的如果是二进制，可写入二进制文件：

```
with open('image.png', 'wb') as f:  
    f.write(r.content).
```

爬虫基础 (requests): User-Agent (UA) 是一个特殊字符串，请求多了最好加上 UA.

```
for key, value in r.cookies.items():  
    print(key + '=' + value)
```

通过键值对
根据头部规则解码类型
未指定编码方式

requests.get 返回的并非全是 HTML，有 json, text, print (type(response.json())), proxy 是代理。

如果想爬取比较多，^{一般} 加代理。

Xpath 是在 XML 文档中查找信息的语言，可用来在 XML 文档中对文本和属性进行遍历。node.name 子节点，node.parent 父节点

节点关系	示例	说明
选取节点	从多行， <code>from lxml import etree text = "" html = etree.HTML(text) p = html.xpath('//head/title/text()')</code>	// 为从根目录 // 为指定某节点， ^{选择}
谓语	<code>print(p)</code>	提取节点文字 以列表显示，选当前节点
选取未知节点		// 选当前节点的后代
Xpath 运算符		改为层级 <code>li = p[0].xpath('.//li/text()')</code>

```
print(int(p[0].split(':')[1].replace('px', '')))
```

一般用 XML 的 etree 库。

`p = html.xpath('//p[last()]/text()')`
如果是一或以行的 1, 2, 3 分别代表第 1, 2, -
(last() - 1) 倒数第二个，从此类推

[position() < 3] 前两个

p[@style]/text()

// title[@lang='eng'] 选取所有 title 语义 lang='eng'。
price > 35

// div[contains(@class, 'f1')] 包含 "f1" 的 div 元素

`p[0].xpath('.//p/text()')`

@ 选取属性。p/@style 也是

比如：a = html.xpath('//a/@href')

作业练习：在jupyter notebook中 In[1] 行运行，

```
import pandas as pd
```

```
from matplotlib import pyplot as plt
```

```
import matplotlib as mpl
```

```
%matplotlib inline
```

```
data=pd.read_excel('..\\xlsx\\
```

```
data
```

```
plt.savefig('..\\png') 保存图
```

```
ax.spines['right'].set_color('none')
```

右坐标轴取消掉

做数据分折，机器学习需要。

Xticks 可替换坐标轴，不替换可能出现小数点

```
year_data=data.iloc[:,0:2].groupby('年份').sum()
```

```
plt.plot(year_data.index,year_data['乘客数量'],color='r', linewidth=2,  
marker='o', markersize=8, label='年度乘客量变化')
```

```
plt.xticks([1949.0, 1950.0, 1951.0, 1952.0, 1953.0], [1949, 1950, 1951, 1952, 1953])
```

```
plt.ylim(1200, 1800)
```

```
plt.xlim(1948, 1954)
```

```
plt.xlabel('年份') plt.ylabel('乘客数量') plt.title('年度变化')
```

```
mpl.rcParams['font.sans-serif']=['SimHei'] - mpl.rcParams['axes.unicode  
ax=plt.gca(), ax.spines['right'].set_color('none')] = False  
ax.spines['top'].set_color('none'), plt.legend(loc='best')
```

fig=plt.figure

新建图

ax=fig.add_subplot() 新建子图

plt.gca().add_patch(plt.

Rectangle((0.1, 0.1), 0.5, 0.3))

plt.show() 获得当前图

① plt.gcf() ② plt.gca()

get current figure

获取当前图。

而 plt.plot() 实际上通过 plt.gca() 得到当前

axes 对象 ax，然后再调用 ax.plot()

获得真正绘图。

`plt.savefig('图例.png'), plt.show()`)

beautiful soup 使用:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(text, 'xml')
print(soup.th.string) → 文本
soup.th 都是获取第一个元素
print(soup.prettify()) 美化获得的格式
```

print(soup.table.attrs['width']) → 通过层级
#从 table 开始到结束.

print(soup.title.name) → 标签名字

print(soup.th.attrs) → 当标签里面的
返回字典曲

print(soup.table.attrs['width']) → 通过层级
#从 table 开始到结束.

print(soup.head.title) → 用 来获得嵌套。

print(soup.tr.children) 和 contents

是一样的结果 不过返回的是迭代器

print([i for i in soup.tr.parents]) → 父节点

print([i.name for i in soup.tr.parents]) [, 'document'] → 每个文件都有的

soup.tr.parent 不是迭代器, print(soup.tr.parent) 找父节点(一个).

兄弟节点: print(soup.tr.next_sibling), print(soup.tr.previous_sibling) → 相邻节点

方法选择器 find_all: print(soup.find_all('tr')) 找到所有的 tr 节点, 通过类

print(soup.find_all(attrs={'align': 'left'})), find() 是返回一个.

CSS 选择器(通常使用), 而 xpath 用的是最多的,

select('ul li') 基本为嵌套, soup.select(), 选取元素后(在谷歌)右键 copy

xpath, 创建半 Pinard 工具栏的博客挺好的. 猫眼加一个请找头就可以了.

右键检查 → 右键 copy → 右键 copy path. → 在下方有与复制的, → 浏览器中的粘贴生选择, 把向键头. 可以快速浏览。

在谷歌中, (检查元素). 下标从 1 开始。 才匹配住意元数据。

args 是列表, print({}\n{})\n, format(args))

enumerate()同时遍历索引和值 for index, 值 in enumerate(page.html), zip()返回元组。
strip('')删除空格。print 打印中间结果，帮助理解，获取多页(博客第几页)。
cnblogs.com/pinard/default.html?page=1, 获得第一页。如果 title 为空就停止。
url = 'https://www.cnblogs.com/pinard/default.html?page=' + page
page=1 就会变为 https://www.cnblogs.com/pinard/default.html?page=1
+27M url.format(page) 包括传入了 page 参数。

安装mysql: 下载那个 400 多兆的.msi 文件, 安装 navicat 安装包(破解版), 64位能用 32 位的, 而且安装自己显示什么选择, 需要装 Microsoft Visual C++ 才可用 mysql, mysql -u root -p 验证是否配置好环境变量。
show databases; 可看数据库, navicat 是工具, 破解包一般安装在该软件安装目录下。
use mysql; Database changed. 表明对表, tables 相应文件。
加分号才执行, 可看作文件夹
select * from user limit 2.
假如有少量数据

navicat 使用: 主机localhost, 端口 3306, 用户名为 root, 密码输入。
右击新建数据库, utf8mb4 排序规则(建议选的)。create database 数据库名; (语句创建)。
右键设计表, 体现名, 类型, 大度, 小数点, 不是 null, 空值。
右击查询 → 新建查询 → create table test2 (

a VARCHAR(20)

VAR CHAR(30)

)

insert into test2(a,b,c) VALUES ('123', '333', '444') 插入数据

并更新数据 update test2 set c='444' where a='123'

并查询数据 select d from test2 where c='77'

mysql 基础语法: 数据一般存放在数据库中。

delete from test2 where c='77' 删除数据 select * from test2 where c like '%>' 代表大于的

inner join $\equiv \text{join}$ left join 右边没有的左边表中数据, right join 类似.

select * from test1 以哪边为主键, 以哪边为外键, 哪边就保留.

select * from test2 并左关联.

select t₁* , t₂* from test1 t₁ left join test2 t₂ on t₁.cc = t₂.c ^{关联条件} ^{on+条件} 通过字段判断关联条件
右键设计表索引, 可加速(尤其在数据多的时候).

接下来如何把取到数据库中:

coding=utf-8

import pymysql

db = pymysql.connect(host='127.0.0.1', #localhost 数据库服务器地址
user='root', ^{用户名} password='123456', port=3306, db='python-test')
定义游标

cursor=db.cursor()

cursor.execute('select * from test1')

data=cursor.fetchone() ^{可获取多个} # 获取一条数据. cursor.fetchall() 返回元组

print(data)

try:

sql="insert into test1(aa,bb,cc) values('abc','123','456')"

cursor.execute(sql)

db.commit()

except Exception as e:

db.rollback

print('数据库插入错误', e)

、表示任意字符 正则表达式默认贪婪匹配量 re? 减配叫反向，非贪婪匹配。 3次
、*? 表示非贪婪模式，组合：匹配一个字符串，出现n次，eg. pattern=r'[^d]{a-zA-Z}{3}'
|w 代表单词字母 pattern=r'|w*|w*'，result=re.findall(pattern, string, re.I)
string='---', pattern=r'[.,。()a-zA-Z0-9]'
result=re.sub(pattern, ' ', string), print(result). 忽略 可能输出大写

若 pattern=r'[-]', 那么 result=re.split(pattern, string) 注意 split 使用方法
表示字符串，可以用';', '' (三个单引号) \B 代表 ^{数字}字母和非字母数字的边界 []
[u4E00-u9FA5] 中文字符串 { 1. 单词与界线 ^{其他} 2. 单词中间 }

pattern=r'(?=|df{3}|(1d{4})|(?=1d{4}))' with open(file_name, 'w', encoding='utf-8') as f:
{ if not os.path.exists(dir): f.write(string)
os.mkdir(dir)

reduce(lambda x,y: int(x)+int(y), data-list), print(sum) 高阶函数
如何找请求头？右键检查该元素，进中network，设name 1 的某项，右键去headers，
为请求头，有User-Agent等。 requests 从取的

使用网页F12 键得到代码与右键打开的网页原代码是一样的：

有些url是经过js处理再生成，(比如url加密解密，id的运算) 这样就造成了上述问题

解决方法：①了解其生成方式，自行设计算法，②借助selenium或者splinter 模拟浏览器
访问 → 可以处理js程序。

{ 新推荐 splinter phantomjs
- selenium + chrome

淘宝爬虫需要分析cookie，自行构造。爬虫神器，但速度慢。

{ response = requests.get(url=url) 显示中文且使用 notepad+
response.encoding='utf-8' 不可直接写里面

1. 安装 selenium: pip install selenium
2. 安装 driver, 看文档就行了(官方), 兼32位
装自己 chrome 对应版本的 webdriver。

初步尝试: 想爬取 QQ 音乐歌曲, 结果发现检查与腾讯不同, 在于它处理了 CSRF, 那么使用 selenium + chrome 来模拟登录, 事实上点击了歌曲, 会转到一个 html 页面, 应该是包含着歌曲信息的一串代码传递过去, ①要么跳过, 解析网页, 整个过程都模拟②对信息直接穿引出歌曲。(无论如何, 都是仅用 requests 网页就可以的)

nlp 新生儿, CV 成熟

transform, bert2, gpt3

?<= 是以什么为前缀, ?= 以什么为后缀, ?<! 3. 以什么为前缀, ?!= 3. 以什么为后缀, 几者都是子捕获的分组

pattern = r'(?<=\d{3})(\d{4})(?=\d{4})'

文件操作一般 os 模块: import os

result = re.sub(pattern, '***', string)

if not os.path.exists(dir):

def outer_func():

os.mkdir(dir)

loc_list = []

嵌套函数 里面的调用外面的函数.

def inner_func(name):

inner func

loc_list.append(len(loc_list) + 1)

特点: 1. 是一个嵌套, 外层函数的返回值是内层函数对象.
outer_func

print('%s loc_list=%s' % (name, loc_list)) print(inner_func.__closure__) 是闭包输出, 返回 none

return inner_func

闭包是一个函数包括一个函数, 如果只是一个函数, 每次调用

clo_func_0 = outer_func()

局部变量都会重新初始化。为了每次调用, 旧的变量

clo_func_0('clo_func_0')

比如列表这个可变对象, 用外面的函数返回里面的函数
相当于内部函数别名(指向同一对象)

调用一次外部函数, 可初始化外部函数内的变量, 而每次调

用内部函数别名执行, 由于只调用了一次外部函数, 所以可变
变量的生命期

不会被重新初始化, 但内部函数的变量会初始化, 所以沿用了

相当与外部函数调用一次，内部调用多次。虽使用内部函数功能，但内部函数的变量也可以调用，并多次使用。

对于可变参数 `(lst) = loc_list`，是地址赋值，闭包主要用于装饰器。

闭包陷阱：

```
def my_func(*args):
```

`f's = []`

`for i in range(3):` 生命周期被延长

`def func():` 报错调用，`id()` 为取地址
return 代码并不执行未起作用。

`f's.append(func)` 如果加上 `if` 结果会全部变为64

`return f's` 并返回时 `i=2`

地址不一样
`f1, f2, f3 = my_func()`

`print(f1())` #4

`print(f2())` #4

`print(f3())` #4

魔术方法：会自己在合适时刻调用。

生成器：可迭代，调用才会计算。

遍历生成器 `{for i in generator}`

`next(generator)`

迭代器：不可以直接实现了 `iter()` 和 `next()` 两个方法，就可以以迭代器调用
生成器的下一个数据

字符串、列表、元组、字典、文件 为数据结构

`print(dir(lst))` 方法属性，魔术方法
没有 `next()`

装饰器：(只是闭包的方便化而已)

语法糖：糖衣语法；计算机语言中添加的某种语法，对语言功能无影响，但更方便使用。

可以使用多个装饰器，添加多个功能。

反射机制：通过字符串形式调用模块

通过字符串模式寻找指定函数。
getattribute()
hasattribute()
setattr()
delattribute()

getattr(instance, 'name', 'not find')

魔术方法是对象中的方法

`-<-->`

`-<-->`

class A:

def __init__(self, a, b):

self.a = a

self.b = b

def print_data(self):

print self.a, self.b

object1 = A(3, 4)

object1.print_data() 普通方法, 对象直接调用

def __eq__(self, other):

if self.a + self.b == other.a + other.b:

return True

else:

return False

需要判断两个实例是否相同, 通过比较参数和判断-

self.a 代表次方,

def __str__(self):

return str(self.a + self.b) 在 print(object1) 时调用

object1.__str__()

print([list1 > list2])

print(list1 + list2)) 实例方法调用比较从

list1 包含的支撑多个比较。

__set__(self, instance, value):

当描述属性的值改变时行为。

C.X 可直接触发 __get__

C.X = 1 __set__

__new__ 是真正被调用第一次方法。

__init__(self, ...) 初始化方法。

是 __new__ 返回的

int('3') 自动转换是有魔力

方法的。

def __call__(self, m, n):

return self.print_mn.

print(foo(2, 3))

X.__call__(a, b) 调用

eg. def __getitem__(self, key):

return self.data[key]

print(object1[1])

{ __delitem__

__getitem__

__setitem__

反转 ← b = reversed(a)

print(44 in object1) 全自动

调用 __getitem__ ,一遍而。

如果有 __contains__ , 全调用它。

```
def __init__(self):  
    return self
```

jupyter notebook 中，其中运行，list 中会出错数字。
与遍历顺序有关，和位置无关。

yield 作用：带有 yield，为生成器，调用函数，但不会执行。np.asarray([1, 2, 3, 4]).reshape(12, 2)
而是个 iterable 迭代对象。

不可变对象，要在函数中修改需要 global 声明。

b.shape → (2, 2)

q = np.linspace(0, 9, 10) → array([0, 1, ..., 9])
间隔 10 个间隔
等差数列。

q.astype('int32') 需要加引号。

{ os.path.isfile() 是否为文件

logspace 等比，np.logspace

os.path.isdir() 是否为目录

w = np.random.random(12, 3)

os.path.exists() 是否存在

np.random.seed(4) # 同一个随机数种子，每次运行程序都一样。

os.path.split() 返回路径目录和文件名

b = np.asarray([1, 2, ..., 12]).reshape(12, 3) # 行计算。

os.system() 执行 shell 命令，os.system('cmd') 启动 cmd

X = np.arange(5, dtype=int)

anaconda prompt 相当于 cmd，有了 numpy。

随机数种子是一致的，用于深度学习。

numpy：1维数组对象 < ndarray，要注意存放同类型。

arange(1, 5) # 其中上取不到。

a = np.array([[1, 2, 3], [4, 5, 6]])

print(a[1, ...]) # 第二行元素。

a.ndim → 2, a.shape → (2, 3), a.dtype → int32.
dtype('int32')

a = np.array([[1, 2, 3], [3, 4, 5], [4, 5, 6]])

a.size → 6,

a[1::2, ::2] → array([[3, 5]])
对第 1 维 第 2 维 → ndim 为 2 极端
只写 1 数字，列表索引，只改
索引 变维度

a = np.array([[1, 2, 3], [4, 5, 6]]) → b.ndim = 3

标量的 ndim 为 0，会降一个维度

b.shape → (1, 2, 3) # 一层层去掉括号

一个方括号代表一个维度。

a3 = a2.astype(float64)

print(a3[1]) → 默认靠左，如果 a 是两维的。

X = np.empty([3, 2], dtype=np.int)

x = np.array([[1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5]])

y = np.zeros([5], dtype=np.int), y.shape → (5,) y.ndim → 1

高维索引：x[[2, 1, 1], [0, 1, 0]], y → [3, 4, 5]
array

z = np.zeros([5], dtype=np.int), z.shape → (5,)

布尔索引：b = x[[true, true, false], [~]]
花括号：用 0, 1 替换 true, false
也是 true

`print(x[4:7])` 按行

若想要引用整数索引，bool序引则传入bool型的。

`from operator import mul, add`

`from functools import partial`

`triple = partial(mul, 3)`

`print(triple(7))` 传参数。

类方法需要 `@classmethod`, 因类名, 方法名(↓)
`A.class_foo('a')`

`@staticmethod`

`def static_foo(x):` 为了修改类属性

`print("executing static_foo(%s)" % x)`

类方法可以被类直接调用, 不依赖于任何对象。

静态方法是类调用, 和对象、类均没关系。

`pattern=r'ab??'` 非贪婪, 加0次
0次 1次

`print('\n\na') → \n a` 不会再双转义

python操作mysql数据库: 清华镜像

`pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pymysql`

`conda install pymysql` 也可。 2:17