调取主函数

```
直接输入函数名调用函数
```

字符串方法

capitalize

让字符串的第一个首字母进行大写

upper

让字符串全部大写

lower

让字符串全部小写

replace

```
string1.replace("wrold","world")
//把字符串中的wrold改为world
```

find

```
string1.find("hello")
//输出n，n为找到hello的第n位
```

boolean

只有两个值：True和False

isupper

```
string1.isupper()
//返回值为布尔类型
```

split

```
string1.split(',',最多分割次数)//不加分割次数默认为无限次最多分割
```

endswith

```
string.endswith(suffix[, start[, end]])
```

参数:

- `suffix`：必需，表示要检查的子字符串或元组。如果是元组，函数会检查字符串是否以元组中的任意元素结尾。
- `start`：可选，指定检查的起始位置。默认为 0，即从字符串的开始位置检查。
- `end`：可选，指定检查的结束位置。默认为字符串的结束位置。

strip

```
string1.strip()
```

去掉两边的空格

访问字符串某个区间的元素

```
string1[a:b]
```

文档

利用官方文档查看方法

https://docs.python.org/

列表

~~其实是一个容器~~

定义列表

```
list1 = [1,2,3,4,5]
```

访问列表

```
list1[0]
```

添加列表

```
list1.append(6)//在末尾添加
```

删除列表

```
list1.pop(要删除的序号)
list1.remove(序号)
```

插入列表

```
list1.insert(元素，序号)
```

排序列表

```
list1.sort()
```

索引列表

~~与字符串的find类似~~

```
list1.index(元素)
```

反转列表

```
list1.reverse()
```

修改列表

```
list[1]=9
```

元组

~~不能修改的列表~~

声明元组

```
tuple1=(1,2,3)
```

列表和元组的转换

```
list(tuple1)
tuple(list1)
```

字典

~~通过一个键查找一个值~~

声明字典

```
dict1 = {"name":"person1","height":170,"weight":100}
```

键值对

```
"name":"person1"
```

"name"为键，“person1”为值，值可以为各种类型

获取字典的键

```
dict1.keys()
```

获取字典的值

```
dict1.values()
```

修改字典的键值对

```
dict1["name"]="Alice"
```

添加字典的键值对

```
dict1["gender"]="male"
```

删除字典的键值对

```
dict1.pop("name")
```

集合

~~没有顺序，不能重复，不能进行索引等操作~~

声明集合

```
set1 = {1,2,3,5,2}
```

打印出来之后，只有{1，2，3，5}

不能重复

添加元素

```
set1.add(5)
```

删除元素

```
set1.discard(元素)
```

集合交集

```
set1.intersection(set2)
```

集合余集

```
set2.difference(set1)
```

会保留set2中不在set1中含有的部分

判断子集

```
set2.issubset(set1)
```

布尔类型

如果set2是set1的自己，返回True

强制类型转换

```
str(list1)
```

print函数

字符串拼接

```
print（“list1” + str（list1））
```

必须保证拼接+的左右都是字符串类型

f-string

`print(f"错误:{e}")`

加上f之后，再花括号中直接加变量就可以输出

值类型变量和引用类型变量

两种变量储存方式

```
a = 1
b = a
b = 3
```

结果：a=1，b=3

```
list1 = [1,2,3]
list2 = list1
list1[1] = 4
```

结果为：list1[1,4,3]

list2[1,4,3]

```
list1 = [1,2,3]
list2 = list1
list2 = [2,3,4]
```

结果为：list1[1,4,3]

list2[2,3,4]

如何解释?

list这种引用类型变量，储存列表时储存的是[1,2,3]的地址

`list2 = list1`会把地址进行传递

当修改list2的某一个值时，list1和list2都指向一个地址，都被修改

`list2 = [2,3,4]`会创建一个新地址，弃用原来的地址

类型举例

值类型：数值，布尔

引用类型：其他类型

条件控制

条件表达式的值

```
expensive = (prize > 800)
```

如果为真，expensive的值为布尔值True

不需要括号括起来循环条件

迭代

遍历

```
for 变量名 in 被遍历的对象：
```

```
list(range(开始序号，最后的元素要小于的值，隔几个遍历一次)
```

创造序列

`for i in range(10)`

结果：i一直从取0一直取到9

`for i in range(a,b)`

模块

使用系统自带模块

import 模块名

`import math`

自定义模块

在同一根目录下的两个python文件，其中一个文件可以通过import调用另一个文件的模块

`import 另一个文件名`

调用：`另一个文件名.函数`

若只想调用另一个文件的某个函数，可以写

`from 另一个文件名 import 函数名`

调用： `函数`

模块重命名

`import 另一个文件名 as 你想重命名的文件名`

调用全部函数

`form 另一个文件名 import *`

后续使用时不用加文件名

主代码的判断

在主代码中，加入以下语句：

```
if __name__ == "__main___":
    pass
else
    pass
```

安装外部模块

在cmd中输入`pip install 模块名`即可

输入函数input

基本用法

```
user_input = input("请输入一些内容: ")
print("你输入的是:", user_input)
```

在上面的示例中，`input()` 函数会**显示提示信息"请输入一些内容:"**，然后等待用户输入。当用户按下回车键时，输入的数据被读取并存储在变量 `user_input` 中，接着 `print()` 函数会输出用户输入的内容。

返回类型

`input()` 函数返回的数据类型**始终是字符串**。如果你需要将输入转换成整数、浮点数或其他类型，你需要使用**类型转换函数**，比如 `int()` 或 `float()`。

```
age = int(input("请输入你的年龄: "))
print("你输入的年龄是:", age)
```

处理数据异常结构

`try-except` 是 Python 中用于异常处理的结构，用来捕获和处理程序运行时可能出现的错误。通过 `try-except` 块，你可以让程序在遇到错误时不会崩溃，而是继续执行后续代码或采取相应的错误处理措施。

基本语法

```
try:
    # 可能会发生异常的代码块
    num = int(input("请输入一个数字: "))
    result = 100 / num
    print("结果是:", result)
except ValueError:
    # 当输入无法转换为整数时触发
    print("请输入有效的数字。")
except ZeroDivisionError:
    # 当除以零时触发
    print("除数不能为零。")
except Exception as e:
    # 捕获所有其他异常，并输出异常信息
    print("发生错误:", e)
finally:
    # 无论是否发生异常，最终都会执行
    print("程序执行结束。")
```

捕获异常并输出

`process_operrations()`

pip换源

更换pip源到国内镜像

pip国内的一些镜像

阿里云 http://mirrors.aliyun.com/pypi/simple/
中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/
豆瓣(douban) http://pypi.douban.com/simple/
清华大学 https://pypi.tuna.tsinghua.edu.cn/simple/
中国科学技术大学 http://pypi.mirrors.ustc.edu.cn/simple/

修改源方法：

临时使用：

可以在使用pip的时候在后面加上-i参数，指定pip源
`pip install scrapy -i https://pypi.tuna.tsinghua.edu.cn/simple`

永久修改：

linux:

修改 ~/.pip/pip.conf (没有就创建一个)， 内容如下：

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
```

windows:

直接在user目录中创建一个pip目录，如：C:\Users\xx\pip，新建文件pip.ini，内容如下

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
```

Anaconda

创建和管理 Conda 环境

步骤 1：打开 VS Code 的终端

1. 打开 Visual Studio Code。
2. 使用快捷键 `Ctrl +`（反引号）或在菜单中选择 **“终端” -> “新终端”** 来打开内置终端。

步骤 2：创建 Conda 环境

在 VS Code 的终端中输入以下命令来创建名为 `myenv` 的 Conda 环境，并指定 Python 版本为 3.8：

```
conda create -n myenv python=3.8
```

然后按下 **Enter**。Conda 会提示确认创建环境，输入 `y` 并按下 **Enter** 进行确认。Conda 将开始下载并安装所需的 Python 版本及其依赖项。

步骤 3：运行 `conda init`

1. 打开 **VS Code 的终端** 或 **命令提示符 (cmd)** / **PowerShell**。
2. 输入以下命令并按下 Enter

   ```
   conda init
   ```

步骤 4：激活环境

环境创建完成后，输入以下命令激活 `myenv` 环境：

```
conda activate myenv
```

此时，终端的提示符应该显示 `(myenv)`，表明你已经成功进入 `myenv` 环境。

步骤 5：验证 Python 版本

在激活的环境中，输入以下命令以验证 Python 版本：

```
python --version
```

这会输出当前使用的 Python 版本，应该显示 `Python 3.8.x`，确认环境中配置的 Python 版本正确。

在 VS Code 中使用新环境

1. 使用 `conda env list` 查看所有环境
2. 打开或创建一个 Python 文件。
3. 点击 VS Code 右下角的 Python 版本号，或使用 `Ctrl + Shift + P` 并输入 `Python: Select Interpreter`，选择 `myenv` 环境中的 Python 解释器。
4. 确认环境激活后，就可以在 `myenv` 环境中运行代码和开发。

这样，你就成功在 VS Code 中创建并激活了一个新的 Conda 环境，`myenv`，并指定 Python 版本为 3.8。

删除环境

步骤 1：列出所有 Conda 环境

首先，你可以使用以下命令查看当前所有 Conda 环境：

```
conda env list
```

或：

```
conda info --envs
```

这将显示所有 Conda 环境的列表，当前激活的环境会有 `*` 标记。

步骤 2：停用当前环境

如果你正在使用想要删除的环境，请首先停用它。使用以下命令停用当前环境：

```
conda deactivate
```

这样可以确保你不会在删除环境时误操作。

步骤 3：删除指定的环境

删除环境的命令如下：

```
conda env remove -n myenv
```

其中，`myenv` 是你想要删除的环境名称。

步骤 4：验证环境是否删除

删除环境后，你可以再次运行 `conda env list` 来确认该环境已经被删除。它不再出现在环境列表中。

示例：

1. 假设你想删除名为

   ```
   myenv
   ```

   的环境：

   ```
   conda deactivate      # 如果正在使用该环境，先停用
   conda env remove -n myenv  # 删除环境
   conda env list  # 验证环境是否已删除
   ```

安装、更新和删除包

使用 `conda install` 命令安装包。

使用 `conda update` 更新包。

使用 `conda list` 查看已安装的包。

使用 `conda remove` 卸载包。

Python的库

panda相关

引入整个pandas文件并重命名

```
import pandas as pd
```

调用函数：打开文件 .read_csv()

```
# 从 CSV 文件加载数据
df = pd.read_csv('data.csv')  # 假设文件名为 data.csv
```

将文件的所有列拆开储存在df变量中

方法：显示所有列的统计信息 .describe()

```
# 计算每个列的基本统计信息
print(df.describe())  # 显示所有列的统计信息（如均值、最大值、最小值等）
```

调用函数与方法的使用区别：

调用函数：库名.函数名

方法：对象名.方法

方式：访问文件中的某一列 变量名.['列名']

```
df['Age']
```

假如csv文件中Age这一列是[23,45,67,45],那么访问的时候df['Age']就代表[23,45,67,45]

方法：找最大值 .max()

```
max_age_row = df[df['Age'] == df['Age'].max()]
```

对 `df[df['Age'] == df['Age'].max()]` 的解释

`df[df['Age'] == df['Age'].max()]` 这一行代码的作用是从 DataFrame `df` 中选出年龄（`Age`）最大的那一行数据。下面是这行代码的详细解释：

1. `df['Age']`：选择 DataFrame `df` 中名为 `Age` 的列。它返回一个包含所有年龄数据的 Series。
2. `df['Age'].max()`：调用 `.max()` 方法，返回 `Age` 列中的最大值，即最大年龄。
3. `df['Age'] == df['Age'].max()`：这部分是一个条件表达式，它会返回一个布尔 Series。每个元素表示 `df['Age']` 列中相应位置的值是否等于 `df['Age'].max()`（即是否等于最大年龄）。如果某个元素是最大年龄，它会是 `True`，否则是 `False`。
4. `df[df['Age'] == df['Age'].max()]` ***其返回值为布尔值为true的那一行***
    - 这部分返回一个布尔 Series，它的长度与 `df` 中的行数相同。
    - 对于每一行，它的值是 `True` 或 `False`，具体取决于该行的 `Age` 是否等于最大年龄。

例如，如果 `df['Age']` 是 `[23, 45, 67, 45]`，并且最大年龄是 `67`，则 `df['Age'] == df['Age'].max()` 的结果将是：

```
[False, False, True, False]
```

`df[...布尔 Series...]`：

- 当你传递一个布尔 Series 作为 DataFrame 的索引时，pandas 会返回一个新的 DataFrame，其中对应 `True` 的行会被选中，而 `False` 的行会被排除。
- 具体来说，它会检查布尔 Series 中每个位置的值：
  - 如果是 `True`，则保留该行；
  - 如果是 `False`，则跳过该行。

方法：找最大值对应的索引 .idxmax()

```
daily_sales = df.groupby('Date')['TotalSales'].sum()
```

返回值为一个字符串

方法：排序 .sort_values

```
filtered_df.sort_values(by='Score', ascending=False)
```

`by='Score'` 表示根据 `Score` 列排序。

`ascending=False` 表示按降序排序（从高到低）

方法： 检验每一列中缺失的值 .isnull()

```
missing_values = df.isnull().sum()
```

返回值为布尔类型

方法：求和 .sum()

对布尔值为Ture的个数求和

```
missing_values = df.isnull().sum()
```

返回值为整数

对数值求和

```
daily_sales = sales_df.groupby('Date')['TotalSales'].sum()
```

返回值为整数

方法：计算某个列中的平均数 .mean()

```
df['Price'].mean()
```

返回值为数值

方法：填充一列中的缺失值 .fillna()

```
df['Price'].fillna(df['Price'].mean(), inplace=True)
```

`inplace=True` 表示修改原始 DataFrame

方式：对两个列的元素进行计算并创建新列

```
sales_df['TotalSales'] = sales_df['Quantity'] * sales_df['Price']
```

创建新列的结果与参与计算的列的元素一一对应

方法：分组+汇总 .groupby()

分组

```
df.groupby('Date')
df.groupby(['Date', 'Product'])
```

将表格的某一列或几列分成一个组

列中相同名称的行将被看作同一行

汇总

```
df.groupby('Date')['TotalSales'].sum()
```

将分成的组进行汇总（这行代码执行的是汇总中加和的操作）

第一个括号`('Date')`会被当作索引，['TotalSales'].sum()返回的值会作为这些索引的值

汇总操作：

除了 `.sum()`，`groupby` 还支持其他常见的聚合操作，例如：

- `.mean()`：计算每个组的均值
- `.max()`：计算每个组的最大值
- `.min()`：计算每个组的最小值
- `.count()`：计算每个组的记录数
- `.std()`：计算每个组的标准差
- `.agg()`：可以进行多个聚合操作

函数：两个列表文件的合并 .merge()

```
pd.merge(employees, salaries, on="EmployeeID")
```

- `employees` 和 `salaries`：是要合并的两个数据集，分别包含员工的基本信息和工资信息。
- `on="EmployeeID"`：指定了用来合并的列，这里是 `EmployeeID` 列。也就是说，`employees` 和 `salaries` 两个 DataFrame 会基于 `EmployeeID` 这列进行匹配。每个 `EmployeeID` 会在两个数据集中找到对应的行并合并在一起。
- 先加载数据集再合并

Open_cv

函数：加载图像 cv2.imread（）

```
image = cv2.imread('image.jpg')
```

函数：显示图像并设置窗口标题 .imshow（）

```
cv2.imshow('My Image', image)
```

(窗口标题，图像储存的变量)

函数：等待用户按下任意键关闭窗口 .waitKey()

```
cv2.waitKey(0)
```

函数：关闭所有 OpenCV 窗口 .destroyAllWindows()

```
cv2.destroyAllWindows()
```

函数：读取原始图像 cv2.imread()

```
image = cv2.imread('路径')
```

函数：将图像转换为灰度图 cv2.cvtColor()

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

(变量, 颜色cv2.COLOR_BGR2GRAY)

函数：对灰度图进行高斯模糊处理 cv2.GaussianBlur()

```
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
```

(5,5)表示5x5 内核大小

0: 这个参数是高斯核的标准差 `sigmaX`。标准差决定了高斯分布的宽度，也影响模糊的强度。

- 如果设置为 `0`，OpenCV 会自动根据内核的大小（这里是 `5x5`）来计算合适的标准差。通常，OpenCV 会选择一个适当的默认值来保证模糊效果符合预期。
- 如果你想自己控制标准差，可以将其设置为其他正数值。例如，`sigmaX = 1.5`，会使模糊效果更明显。

函数：获取图像的高和宽 image.shape[:2]

```
height, width = image.shape[:2]
```

返回两个整数，分别是图像长和宽的坐标（像素）

方式：获取图像长宽的中心坐标

```
center_x, center_y = width // 2, height // 2
```

方式：裁剪

```
cropped_image = image[center_y - 50:center_y + 50, center_x - 50:center_x + 50]
```

函数：缩放 cv2.resize()

```
resized_image = cv2.resize(image, (width // 2, height // 2))
```

函数：绘制图形

绘制矩形 cv2.rectangle()

```
# 绘制一个红色的矩形，大小为 200x100，线宽为 3 像素
top_left = (100, 150)  # 矩形左上角的坐标
bottom_right = (300, 250)  # 矩形右下角的坐标
cv2.rectangle(image, top_left, bottom_right, (0, 0, 255), 3)  # 红色 (BGR) = (0, 0, 255)
```

绘制圆形 cv2.circle()

```
center = (200, 200)  # 圆心位置（图像中心）
radius = 50  # 半径
cv2.circle(image, center, radius, (255, 0, 0), -1)  # 蓝色 (BGR) = (255, 0, 0)，填充圆形
```

补充：使用numpy创建一个空白图像

```
# 创建一张空白图像，背景为白色 (255, 255, 255)
image = np.ones((400, 400, 3), dtype=np.uint8) * 255

image = np.zeros((400, 400, 3), dtype=np.uint8)  # 创建一个全黑的图像
image[:] = [0, 0, 255]  # 将所有像素的颜色设置为红色 (BGR 顺序)
```

这行代码的目的是创建一个 400x400 像素的 **白色** 彩色图像。它使用了 **NumPy** 来创建一个图像矩阵，并将其初始化为白色。下面是逐部分解释：

1. `np.ones((400, 400, 3), dtype=np.uint8)`

`np.ones()`：

- `np.ones()` 是 NumPy 中的一个函数，用来创建一个 **指定形状**（size）的数组，并将每个元素的值都设置为 **1**。
- `np.ones(shape, dtype)` 生成一个指定形状和数据类型的数组，其中所有的元素值都是 `1`。

`(400, 400, 3)`：

- ```
  (400, 400, 3)
  ```

  是一个元组，表示图像的尺寸和颜色通道数：
  - `400`：图像的 **高度**（400 像素）。
  - `400`：图像的 **宽度**（400 像素）。
  - `3`：图像的 **颜色通道数。**对于彩色图像，通常是 RGB 格式，因此有 3 个通道：红色、绿色和蓝色（即 R、G、B）。

因此，`np.ones((400, 400, 3), dtype=np.uint8)` 会生成一个 **400x400 像素** 的图像，每个像素都有 **3 个颜色通道**（R、G、B）。初始时，所有通道的值为 `1`。由于数据类型是 `np.uint8`，每个像素值的范围是从 0 到 255（无符号 8 位整数）。

这时，`image` 的形状是 `(400, 400, 3)`，所有像素值是 `1`，即每个像素的 R、G、B 分量都为 `1`，因此图像中每个像素的颜色是非常接近黑色的。

2. `dtype=np.uint8`

- `dtype=np.uint8` 指定数据类型为 **无符号 8 位整数**（`uint8`），表示每个像素值的范围是 0 到 255。也就是说，每个颜色通道的值会限制在这个范围内，避免负值或者过大的数值。
- 在图像处理中，通常使用 `uint8` 类型来表示像素的颜色值。

3. `* 255`

- `* 255` 是对整个数组的每个元素进行乘法操作，**将每个像素的值从 `1` 调整为 `255`**。
- 由于原始图像的所有像素值都是 `1`，乘以 `255` 后，所有像素的颜色分量会变成 `255`，即每个像素的 R、G、B 分量都会变为 `255`。

结果：

- 每个像素的 R、G、B 分量都会是 `255`，表示白色，因为在 RGB 模型中，`[255, 255, 255]` 代表白色。
- 因此，`image` 变量最终将是一个 **400x400 像素** 的图像，所有像素的颜色是 **白色**。

4. 结果：

- `image` 是一个 3D 数组，形状是 `(400, 400, 3)`，表示一个 400x400 像素的图像，其中每个像素有 3 个颜色通道（R、G、B）。
- 图像中的每个像素的值是 `[255, 255, 255]`，因此它显示为 **白色图像**。

###
  ```

函数：保存图像为新的文件 cv2.imwrite()

```
cv2.imwrite(r'C:\Users\Administrator\Desktop\example_python\Open_cv\gray_image.jpg',
gray_image)
```

(输出路径，变量)