

Python基础教程

陈加忠

计算科学与技术学院

QQ群: 904987289

Email: jzchen@hust.edu.cn



学习中存在的问题

- 少数同学**不愿**从提供的.py源文件中找到类似的例子, 并模仿相关语句语法
 - ✓ 新建.py时, 在文件资源管理器→选项里打开了隐藏文件后缀名, 造成没有真正改掉.txt后缀名, 试图**直接运行.py.txt文件**
- 少数同学**不愿**跟上老师的节奏, 而是自己在课堂上安装开发环境
- 不反对同学们用VSC等其它IDE, 能配好解释器即可
 - ✓ 实际上, 这两者没有高低之分, 比如百度用VSC, 腾讯用PyCharm
 - ✓ 后续的《计算思维》课程还是要求用PyCharm
 - ✓ 没有把编写的.py文件按要求放在code_Uxxx文件夹, 造成调用失败
 - ✓ 部分同学用cmd+python调用解释器, 有点甚至还不会配置**解释器**
- 对于if语句, 少数同学忘记加冒号, 或者把冒号加在if后面, 如
 - ✓ if (a>b) 或者 if: (a>b)

学习中存在的问题

- 对于低一级的程序块, 少数同学不能规范地做缩进
 - ✓ 还不能熟练使用tab键做缩进, 或者使用shift+tab做反缩进; 还不能用快捷键ctrl+\做注释或者反注释
- 对于input语句
 - ✓ input后面忘记加小括号
 - ✓ 或者对input()得到的输入值直接做算术运算, 没有做类型转换, 不知道如何做类型转换
 - ✓ 在做类型转换时, 用(int)a, 而不是用int(a); 或者int(input"请输入成绩:")
 - ✓ 程序风格问题, 比如: int(a) 写成int (a), c = a + b写成了c=a+b
- 遇到程序报错不知如何下手解决
 - ✓ 根据提示找到报错的行, 借助print打印中间结果以辅助分析
 - ✓ 设计的算法有逻辑错误
- 对于核心的语句, 没有写注释的习惯, 大家可以观察老师写的代码, 语句后面会有必要的注释

Python基础知识

- Python开发环境
- Python基础语法
- Python数据类型
- 表达式
- 函数结构
- 流程控制

Python函数

- 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();可以自己创建函数,这被叫做用户自定义函数
- 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()**
 - ✓ 任何传入的参数和自变量必须放在圆括号中
 - ✓ 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] 结束函数,选择性地返回一个值给调用方
 - ✓ 不带表达式的**return**相当于返回 **None**

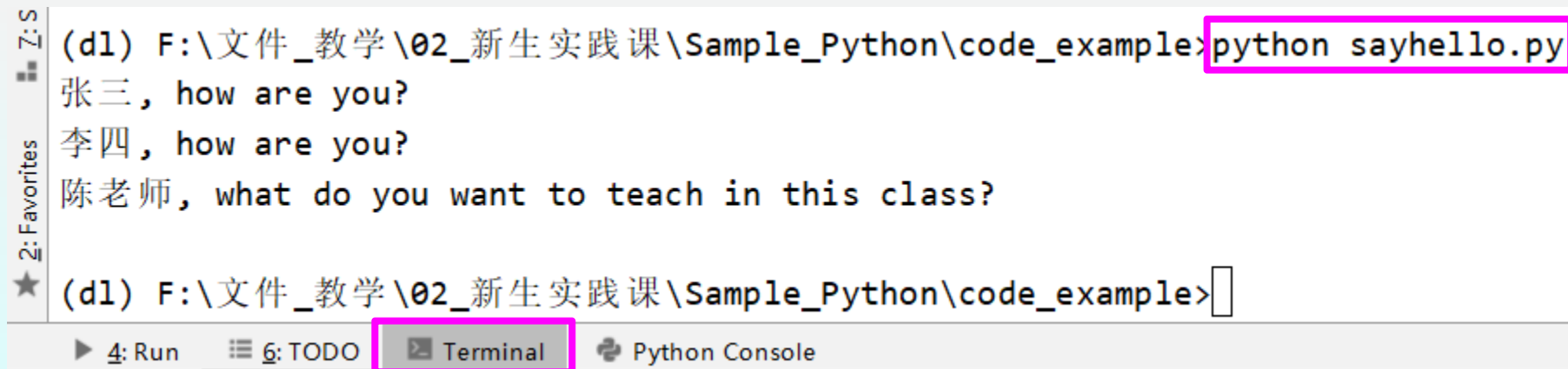
Python函数定义

- **sayhello.py**
- 默认情况下, 参数值和参数名称是按函数声明中定义的顺序匹配起来的
- 以下为一个简单的Python函数, 它将一个字符串作为传入参数, 再打印到标准显示设备上

```
2  def printsayhello(name):
3      if name == "陈老师":
4          name = name + ", what do you want to teach in this class?"
5      else:
6          name = name + ", how are you?"
7      print(name)
8      return name
9
10 printsayhello("张三")
11 printsayhello("李四")
12 printsayhello("陈老师")
```

Python函数调用

- 定义一个函数只给了函数一个名称, 指定了函数里包含的参数, 和代码块结构
- 这个函数的基本结构完成以后, 你可以通过另一个函数调用执行, 也可以直接从Python提示符执行
- 如下实例调用前页定义的printsayhello()函数:



The screenshot shows a terminal window with a light blue background. On the left side, there is a vertical sidebar with icons for 'Favorites', '2: Favorites', and a star icon. The terminal text is as follows:

```
(d1) F:\文件_教学\02_新生实践课\Sample_Python\code_example>python sayhello.py
张三, how are you?
李四, how are you?
陈老师, what do you want to teach in this class?

(d1) F:\文件_教学\02_新生实践课\Sample_Python\code_example>
```

At the bottom of the terminal, there is a horizontal bar with four tabs: '4: Run', '6: TODO', 'Terminal', and 'Python Console'. The 'Terminal' tab is currently selected and highlighted with a pink border.

函数参数类型

- 必选参数

- ✓ 定义加法函数plus, 参数a, b就是必选参数
- ✓ 调用函数plus时, 必须给参数a, b传递值
- ✓ 调用时候, 函数后面不能加冒号

```
def plus(a,b):  
    c = a + b  
    return c  
  
d = plus(1,2)  
print(d)
```


函数参数类型

- 默认参数

- ✓ 定义加法函数PLUS, a是必选参数, b是默认值为2的参数
- ✓ 调用函数PLUS时, 则必须给a传递值, 若不给b传递值, 则b默认为2
- ✓ 如果默认a, 怎么给b传递值?

```
def PLUS(a,b=2):  
    c = a + b  
    return c  
  
d = PLUS(1)  
print(d)
```

```
def PLUS(a=1,b=2):  
    c = a + b  
    return c  
  
d = PLUS(b=1)  
print(d)
```

函数参数类型

- **para_tuple.py**
- 可变参数, 参数汇集成元组

```
1  # para_tuple.py
2  # *numbers表示输入参数的个数可以是任意指
3  def myplus(*numbers):
4      add = 0
5      for i in numbers:
6          add += i
7      return add
8
9  # 调用3次plus函数, 每次参数个数都不相同
10
11  d1 = myplus(1,2,3)
12  d2 = myplus(1,2,3,4)
13  d3 = myplus(1,3,5,7,9)
14
15  # 向函数中可以传递0个参数
16  d4 = myplus()
17  print("d1=",d1,"d2=",d2,"d3=",d3,"d4=",d4)
```

函数参数类型

- **para_dict.py**
- 当我们声明一个诸如 ****number** 的双星号参数时, 从此处开始直至结束的所有关键字参数都将被收集并汇集成一个名为 **number** 的字典

```
2  #把赋值的等号两边变成key和value
3  def myplus(**number):
4      return number
5
6  d1 = myplus() # 向函数中可以传递0个参数
7  d2 = myplus(x=1)
8  d3 = myplus(x1=1, y1=2)
9
10 print("d1=", d1, "d2=", d2, "d3=", d3)
11
```

```
↑ C:\Users\jzchen\.conda\envs\dl\python.exe F:
↓ d1= {} d2= {'x': 1} d3= {'x1': 1, 'y1': 2}
⇅ Process finished with exit code 0
```

函数返回值return

- **return_a_function.py**
- 用return返回一个值
- 用return返回多个值(即一个元组)
 - ✓ 例如: return 1, 'abc', '1234'
- 用return返回函数

```
1  # return_a_function.py
2  # 定义求和函数, 返回的并不是求和结果, 而是计算求和的函数
3  def lazy_plus(*args):
4      def plus():
5          s = 0
6          for n in args:
7              s = s + n
8          return s
9      return plus
10
11 # 调用函数f时, 得到真正求和的结果
12 f = lazy_plus(1, 2, 3, 4, 5)
13 print(f())
```

嵌套调用

- **nest.py**
- 允许在函数内部创建另一个函数, 这种函数叫内嵌函数或者内部函数
- 内嵌函数的作用域在其**内部**, 如果内嵌函数的作用域超出了这个范围就不起作用, 如下所示代码:

```
def function_1():
```

```
    print('正在调用function_1()...')
```

```
    def function_2():
```

```
        print('正在调用function_2()...')
```

```
    function_2()
```

```
function_1()
```

```
function_2() #报错
```

使用闭包

- **closure.py**
- 闭包是函数式编程的一个重要的语法结构, 它是能够读取外部函数内的变量的函数, 外层函数的变量持久地保存在内存中
- 从表现形式上定义为, 如果在一个内部函数里对一个外部作用域(但不是在 全局作用域)的变量进行引用, 那么内部函数就认为是闭包, 如下所示代码:

```
def Fun_sub(a):
```

```
    def Fun_sub2(b):
```

```
        return a-b #内部函数里对一个外部作用域的变量进行引用
```

```
    return Fun_sub2
```

```
i = float(input('请输入减数: '))
```

```
j = float(input('请输入被减数: '))
```

```
print(Fun_sub(i)(j))
```

```
Fun_sub2(23) # NameError: name 'Fun_sub2' is not defined
```

函数的作用域

- 在Python中, 正常的函数和变量名是公开的(public), 都是可以被直接引用的, 比如: `abs()`、`max()` 等
- 类似`__xxx__`这种格式的变量是特殊变量, 允许被直接引用, 但是会被用作特殊用途, 比如`__author__`、`__name__`就是属于特殊变量
- 类似`_xxx`(一条下划线)和`__xxx` (两条下划线)这种格式的函数和变量就是非公开的(private) 不应该被直接引用
 - ✓ `_xxx`的函数和变量是protected, 我们直接从外部访问不会产生异常
 - ✓ `__xxx`的函数和变量是private, 我们直接从外部访问会报异常, 我们要注意前缀符号的区别

函数的作用域

- **private_example.py**
- **private**函数和变量是"不应该"被直接引用,而不是"不能"被直接引用,这是因为在Python中并没有一种方法可以真正完全限制访问**private**函数或变量,但是我们为了养成良好的编程习惯,是不应该引用**private**函数或变量的
- **private**函数的作用是隐藏函数的内部逻辑,让函数有更好的封装性,我们一般都限定函数的使用范围,把外部需要使用的函数定义为**public**函数
- 把只在内部使用,而外部不需要引用的函数定义成**private**函数

```
1      # private.py
2      def _private_1(name):
3          return 'Hello, %s' % name
4
5      def _private_2(name):
6          return 'Hi, %s' % name
7
8      def greeting(name):
9          if len(name) > 3:
10             return _private_1(name)
11          else:
12             return _private_2(name)
13      name = "陈加忠"
14      print(greeting(name))
```


函数的import *

- **all_example.py**
- **all_example_test.py**
- 某个名为all_example的模块, 如果定义了 `__all__`, 那么
 - ✓ 在执行 `from all_example import *` 的时候, all_example模块中不在 `__all__` `["xxx", "yyy"]` 中的不会被导入
 - ✓ `from all_example import *`, 表示从 `from all_example import` 导入所有允许导入的函数、变量等对象
- 如果导入了单下划线开头的 `_zzz`
 - ✓ 如 `__all__` `[xxx, yyy, _zzz]`, 则也能执行 `_zzz` 函数
 - ✓ 没有 `__all__` 语句, 则不能执行 `_zzz` 函数

类的定义与使用

- 类是创建对象的代码段, 描述了对对象的特征、属性、要实现的功能以及采用的方法等
- **属性**: 描述了对对象的静态特征, 属性就是变量, 静态的特征
- **方法**: 描述了对对象的动态动作, 方法就是一个一个的函数
- **对象**: 对象是类的一个实例, 就是模拟真实事件, 把数据和代码都集合到一起, 即属性、方法的集合
- **实例**: 就是类的实体
- 类的定义以关键字**class**开始, 类名必须以**大写字母(有歧义, 貌似小写也可以)**开头, 类名后面紧跟小冒号“:”

类的定义与使用

- **class_bear.py**
- 类的构造方法是: `__init__(self)`
- 实例化一个对象的时候, 这个方法就会在对象被创建的时候自动调用
- 实例化对象的时候是可以传入参数的, 这些参数会自动传入, 如下所示代码:

`class Bear:` # 开始类的定义

```
other_name = '老虎'
def __init__(self, name = other_name): # 类的构造
    self.name = name # name: 类的属性
def kill(self): # 类的方法
    print("%s, 是受保护动物, 不能杀..." % self.name)
```

`animal = '狗熊'`

`bear = Bear()` # 将类Bear实例化为对象bear

`bear.name = animal` # 对象的属性

`bear.kill()` # 对象的方法

正则表达式

- 正则表达式(Regular Expression), 此处的Regular即规则、规律的意思, Regular Expression即描述某种规则的表达式, 因此它又可称为正规表示式、正规表示法、正规表达式、规则表达式、常规表示法等, 在代码中常常被简写为regex、regexp或RE
- 正则表达式使用某些单个字符串, 来描述或匹配某个句法规则的字符串, 在很多文本编辑器里, 正则表达式通常被用来检索或替换那些符合某个模式的文本

字符	说明
.	匹配任意1个字符 (除了\n)
[]	匹配[]中列举的字符\d
\d	匹配数字, 即0-9
\D	匹配非数字, 即不是数字
\s	匹配空白, 即空格或Tab键
\S	匹配非空白
\w	匹配单词字符, 即a-z. A-Z. 0-9. _
\W	匹配非单词字符

正则表达式

- **regular_expression.py**
- 写出一个正则表达式来匹配是否是手机号

```
1  # regular_expression.py
2  # 写出一个正则表达式来匹配是否是手机号
3  import re
4  phone_rule = re.compile('1\d{10}') # 创建模式对象
5  print(phone_rule)
6  phone_num = input('请输入一个手机号') # 通过规则去匹配字符串
7  if len(phone_num) != 11:
8      print('手机号应该是11位数字')
9  elif len(phone_num) == 11:
10     sample_result = phone_rule.search(phone_num)
11     if sample_result != None:
12         print('这是一个手机号')
13         print(sample_result.group())
14     else:
15         print('这不是一个手机号')
```

Python基础知识






- Python开发环境
- Python基础语法
- Python数据类型
- 表达式
- 函数结构
- 流程控制

程序流程结构

- 流程控制是指在程序运行时, 对指令运行顺序的控制
- 通常, 程序流程结构分为三种: 顺序结构、分支结构和循环结构
 - ✓ 顺序结构是程序中最常见的流程结构, 按照程序中语句的先后顺序, 自上而下依次执行
 - ✓ 分支结构则根据if条件的真假(True或者False)来决定要执行的代码
 - ✓ 循环结构则是重复执行相同的代码, 直到整个循环完成或者使用break强制跳出循环, 其本质也是分支结构
- Python语言中, 我们一般使用if语句实现分支结构, 用for和while语句实现循环结构

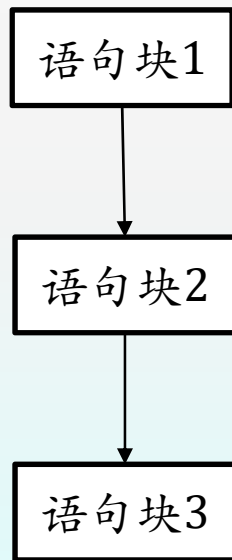
流程图

- 流程图, 是使用图形来表示流程控制的一种方法, 是一种传统的算法表示方法, 用特定的图形符号和文字对流程和算法加以说明, 叫做算法的图, 也称为流程图
- 流程图有它自己的规范, 按照这样的规范所画出的流程图, 便于技术人员之间的交流, 也是软件项目开发所必备的基本组成部分, 因此画流程图也应是开发者的基本功

符号	说明
	圆角矩形用来表示“开始”与“结束”
	矩形用来表示要执行的动作或算法
	菱形用来表示问题判断
	平行四边形用来表示输入输出
	箭头用来代表 workflow 方向

顺序结构

- 程序最基本的结构就是顺序结构, 顺序结构就是程序按照语句顺序, 从上到下依次执行各条语句



1	a=2
2	b=3
3	print(a+b)

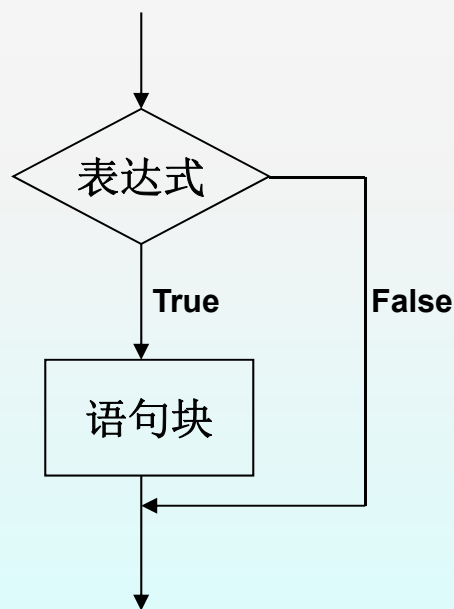
Run:	顺序结构 x
▶	↑ D:\Python\Python36\python.exe F:/py
■	↓ 5
	↺ Process finished with exit code 0
🖨	📄

条件流程控制

- Python条件语句由if发起的, 是通过一条或多条语句的执行结果(True或者False)来决定执行的代码块
- “判断条件”成立时(非零), 则执行后面的语句, 执行内容可以多行, 以**缩进**来区分表示同一范围, 每个条件后面要使用冒号“:”
- **else** 为可选语句, 当需要在条件不成立时执行内容则可以执行相关语句
- 在Python中**没有**switch...case语句

单向分支选择结构

- 是最简单的一种形式, **不包含** `elif` 和 `else`, 当表达式值为 `True` 时, 执行语句块, 否则该语句块不执行, 继续执行后面的代码



```
1  #判断变量a的值是否为True, 是则执行a=0, 否则直接输出a的值。
2  a = 1
3  if a:    # 等价于a>0或a!=0
4      a = 0
5  print(a) # 如果if条件的值为False则输出结果为1
```

if a

Run: demo03_02_01 x

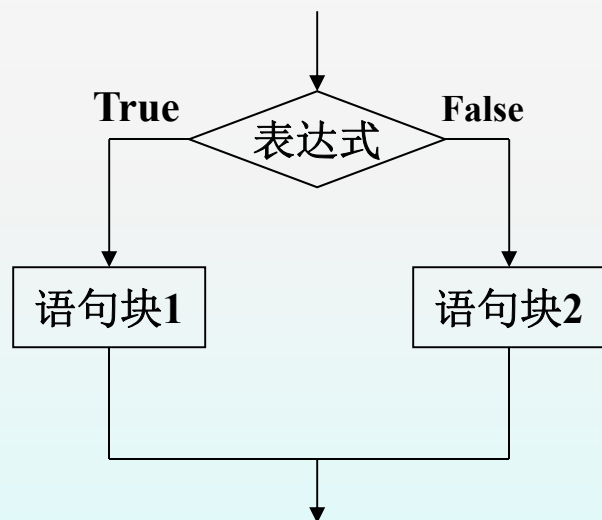
D:\Python\Python36\python.exe F:/python/new/demo03_02_01.py

0

Process finished with exit code 0

双分支选择结构

- 由if和else两部分组成, 当表达式的值为True时, 执行语句块1否则执行语句块2



```
1  #判断a的值是否为True, 是则执行语句块1, 否则执行else部分, 最后再输出a的值。
2  a = 5
3  if a < 0:
4      a = 10
5      print(a+1)    # 如果if的条件为True, 就输出11
6  else:
7      a = 15
8      print(a+2)    # 如果if的条件为False, 就输出17
9  print(a)
```

if a < 0

Run: demo03_02_02 x

D:\Python\Python36\python.exe F:/python/new/demo03_02_02.py

17

15

Process finished with exit code 0

多分支选择结构

- 多分支选择结构由if、**一个或多个elif**和一个else子块组成，else子块可省略
- 一个if语句可以包含多个elif语句，但结尾最多只能有一个else

```
1  # 根据你身上带的钱，来决定你今天中午能吃什么。
2  money = float(input("请输入你带的钱："))
3  if (money >= 1) and (money <= 5):    # 判断money是否在1到5之间
4      print("你可以吃包子")
5  elif (money > 5) and (money <= 10): # 判断money是否在6到10之间
6      print("你可以吃面条")
7  elif money < 0:                      # 如果money小于0，就说明你没有钱，否则就说明你的钱大于10元
8      print("你的钱不够")
9  else:
10     print("你可以吃大餐")

elif (money > 5) and (money <= ...

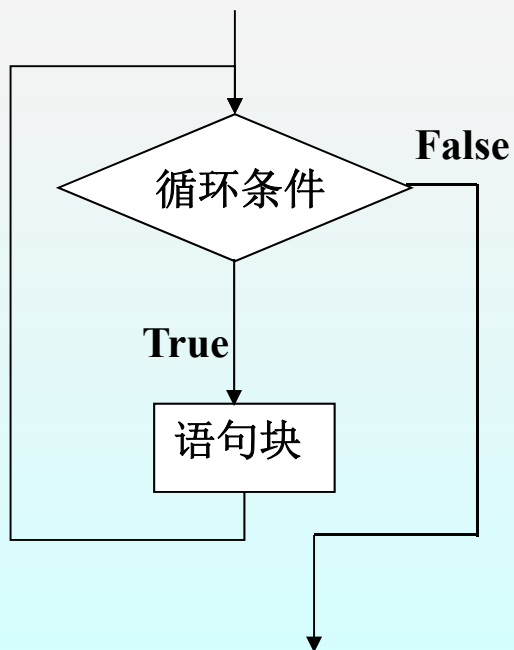
Run: demo03_02_03 x
D:\Python\Python36\python.exe F:/python/new/demo03_02_03.py
请输入你带的钱：50
你可以吃大餐
Process finished with exit code 0
```

三元操作符

- 三元操作符的基本格式为：
 - ✓ `result = x if x < y else y`
 - ✓ 其中, `x < y` 为判断语句, 若 `x < y` 为真则 `result = x`, 否则 `result = y`
 - ✓ `x = x if x is not None else y`
- 三元操作符的最大特点是不需要像 `if-else` 语句那样写多行代码, 三元操作符只需一行代码
- 但不会改变控制相关的本质!

循环流程控制

- 循环结构是指在程序中需要反复执行某个功能而设置的一种程序结构
- Python提供for和while两种循环语句
 - ✓ for语句, 用来遍历序列对象内的元素, 通常用在已知的循环次数
 - ✓ while语句, 提供了编写通用循环的方法



循环控制语句

- 循环控制语句可以更改语句执行的顺序
- **Python**支持以下循环控制语句：

控制语句	描述
<u>break 语句</u>	在语句块执行过程中终止循环，并且跳出整个循环。
<u>continue 语句</u>	在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。
<u>pass 语句</u>	pass是空语句，是为了保持程序结构的完整性。

Python循环语句实例

- range(101)函数是生成0到100的整数, 而range(1, 101)是生成1到100的整数
- 求1~100的累加和

```
1 Sum = 0
2 for s in range(1, 101): # 循环从1到100, 当101时就退出循环
3     Sum += s           # 求1到100的累加和
4 print(Sum)
```

Run: demo03_03_01 x

D:\Python\Python36\python.exe F:/python/new/demo03_03_01.py

5050

Process finished with exit code 0

Python循环语句实例

- **deleteEven.py**
- for和else的组合
- 删除列表对象中所有偶数

```
1
2 x = list(range(20))
3 for i in x:
4     x.remove(i)
5 else:
6     print("even deleted")
7 print(x)
```

Run: deleteEven ×

```
C:\ProgramData\Anaconda3\python.exe F:/教学/新生实践课/Sample_Python/deleteEven.py
even deleted
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

Process finished with exit code 0
```

Python循环语句实例

- **book_java.py**
- 输出列表中的第三个书名除外的元素

```
list = ['python', 'matlab', 'java', 'c', 'c++']
count = 0
for book in list:
    count += 1
    if count == 3:
        continue
    print("当前书籍名字为:", book)
    if book == 'matlab':
        print(book, "可以节约算法验证的时间, 但只会", book, "容易被鄙视")
```

```
for book in list > if book == 'matlab'
```

book_java ×

C:\ProgramData\Anaconda3\python.exe F:/教学/新生实践课/Sample_Python/book_java.py

当前书籍名字为: python

当前书籍名字为: matlab

matlab 可以节约算法验证的时间, 但只会 matlab 容易被鄙视

当前书籍名字为: c

当前书籍名字为: c++

Process finished with exit code 0

Python循环语句实例

- **while_count.py**
- 计数到5则结束

```
count = 0
while(count <= 10):
    print("The counter is:", count)
    count += 1
    if count > 5:
        break
```

```
while_count x
C:\ProgramData\Anaconda3\python.exe F:/教学/新生实践课/Sample_Python/while_count.py
The counter is: 0
The counter is: 1
The counter is: 2
The counter is: 3
The counter is: 4
The counter is: 5

Process finished with exit code 0
```

Python基本规范与语句

- 语句对齐规则

- ✓ 同一级别的语句需要用tab键对齐
- ✓ 可以看出, if语句与for语句, 和def一样, 都以冒号结尾

```
def feature_point(img, method=1): # 给形参指定默认值后
    img_shape = img.shape
    rows = img_shape[0]
    cols = img_shape[1]
    chan = img_shape[2]
    print("输入图像的长高及颜色通道数:", rows, cols, chan)

# 第1种每行每列相加的方法, 得到投影
    if method == 1:
        rows_line = np.zeros((rows))
        cols_line = np.zeros((cols))
        for i in range(rows):
            for j in range(cols):
                rows_line[i] = rows_line[i] + img[i, j, 0] # 每行元素累加
```

Python基本规范与语句

- 注释

- ✓ 暂时不用的语句, 或者对语句及程序功能的解释, 需要用注释符号界定
- ✓ python语言注释符号是#
- ✓ 注释的快捷键是Ctrl+/, 即可以打开又可以关掉注释
- ✓ 可以用Ctrl+z取消对语句的修改, 用Ctrl+y恢复对语句的修改

Python基本规范与语句

- **main_example.py**
- `if __name__ == '__main__':`
 - ✓ 一个python文件(.py)通常有两种使用方法
 - ◆ 第一是作为脚本直接执行
 - ◆ 第二是 **import** 到其他的 python 脚本中被调用(模块重用)执行
 - ✓ 因此 `if __name__ == '__main__':` 的作用就是控制这两种情况执行代码的过程, 在 `if __name__ == '__main__':` 下的代码只有在第一种情况下(即文件作为脚本直接执行)才会被执行
 - ✓ 而 **import** 到其他脚本中是不会被执行的

Python基本规范与语句

- numpy
 - ✓ **import** numpy as np
 - ✓ numpy (Numerical Python)是 Python 语言的一个扩展程序库, 支持大量的维度数组与矩阵运算, 此外也针对数组运算提供大量的数学函数库
 - ✓ 如果在脚本头通过**import** numpy as np导入则可以通过诸如**array = np.zeros((100))**创建新数组, 该数组有100个为0值的元素
 - ✓ MATLAB: **array = zeros(100,1)** # 开一个100行的列数组

Python基本规范与语句

- mean_example.py
- numpy求均值
 - ✓ 若x是一维数组(矢量或向量), 则均值为一变量
 - ◆ `mean_x = np.mean(x)`
 - ✓ 若x是二维数组, 可以试一下对二维矩阵求均值, 即对每行、每列或整个矩阵求均值
 - ◆ `mean_row=np. mean(x, axis=1)`
 - ◆ `mean_column=np. mean(x, axis=0)`
 - ◆ `mean_total=np. mean(x[:]))` # 求整个矩阵的极值

Python基本规范与语句

- numpy求极值
 - ✓ 若 x 是一维数组, 则均值为一个标量
 - ◆ `max_x = np.max(x)`
 - ◆ `min_x = np.min(x)`
 - ✓ 若 x 是二维数组, 可以试一下对二维矩阵求极值, 即对每行、每列或整个矩阵求极值
 - ◆ `max_r=np. max(x, axis=1)`
 - ◆ `min_c=np. min(x_array, axis=0)`
 - ◆ `min_total=np. min(x_array[:])` # 求整个矩阵的极值

Python基本规范与语句

- **sort_example.py**
- 排序算法: 先用numpy把列表转成数组

```
1  # sort_example.py
2  # 输入3位同学的成绩, 然后大到小排列
3  import numpy as np
4  a = float(input('请输入第一位同学的成绩:'))
5  b = float(input('请输入第二位同学的成绩:'))
6  c = float(input('请输入第三位同学的成绩:'))
7  score = [a, b, c]
8  score = np.array(score)
9  for k in range(len(score)):
10     for i in range(len(score) - 1):
11         if score[i] < score[i+1]:
12             temp = score[i]
13             score[i] = score[i+1]
14             score[i+1] = temp
15     print(score)
```

Python矩阵运算

- 数组的向量及矩阵表示
- 矩阵

$$\checkmark a = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}, d = \begin{bmatrix} 5 & 6 \\ 5 & 6 \end{bmatrix}$$

- 向量

$$\checkmark b = [5 \ 6]$$

$$\checkmark c = [2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]$$

$$\checkmark e = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Python矩阵运算

- 列表初始化
- 列表嵌套→二维列表
 - ✓ `a=[[2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9]]`
 - ✓ `b= [5, 6]`
 - ✓ `c= [2, 3, 4, 5, 6, 7, 8]`
 - ✓ `d= [[5, 6], [5, 6]]`
 - ✓ `e= [1, 2]`

Python矩阵运算


- 列表转数组
- **import numpy as np**
 - ✓ `a = np.array(a)` # list转array
 - ✓ `b = np.array(b)`
 - ✓ `c = np.array(c)`
 - ✓ `d = np.array(d)`
 - ✓ `e = np.array(e)`

Python矩阵运算

- 有两种比较常用的矩阵运算
 - ✓ 矩阵乘法: `np.dot()`
 - ✓ 矩阵中各元素逐元素相乘: `*`

Python基本规范与语句

- 数组维度摘取

- ✓ 假设是一个三维数组
- ✓ 彩色图像就是一个三维数组
- ✓ `img = img[:, :, -1]` # 第3个维度的最后(右)一个通道 (索引), 变成一个二维数组 (矩阵)
- ✓ `img = img[:, :, 0]` # 即取第1个颜色(红色)通道, 变成一个二维数组 (矩阵)
- ✓ `img = img[:, :, 1]` # 即取第2个颜色(绿色)通道, 变成一个二维数组 (矩阵)
- ✓ 请问: 怎么取蓝色的颜色通道? `img = img[:, :, -1]`

Python矩阵运算

- **product_matrix.py**

- $b \times a = [5 \ 6] \times \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$

- 行向量与矩阵做矩阵乘法运算

- ✓ `a = [[2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9]]`

- ✓ `b = [5, 6]`

- ✓ `a = np.array(a) # list转array`

- ✓ `b = np.array(b)`

- ✓ `output = np.dot(b[None, :], a) # b[None, :] # 增加维度`

- ✓ `output = [[28 39 50 61 72 83 94]]`

- ✓ `output = output1[-1, :] # 取最右通道以降低维度, -1→0行不行?`

- ✓ `output = [28 39 50 61 72 83 94]`

Python矩阵运算

- 行向量乘以列向量
 - ✓ `output = np.dot(b, e)`
 - ✓ $b = [5 \ 6], e = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
 - ✓ $b \times e = [5 \ 6] \times \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 17$

Python矩阵运算

- 列向量乘以行向量

- ✓ $b = [5 \ 6], e = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

- ✓ 把**b**变为列向量, 把**e**变成行向量

- ✓ `output = np.dot(b[:, None], e[None, :])`

- ✓ 不需要消除维度

- ✓ $e \times b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \times [5 \ 6] = \begin{bmatrix} 5 & 6 \\ 10 & 12 \end{bmatrix}$

Python矩阵运算

- 矩阵与向量逐元素相乘
- 为了下列实现逐元素乘法:
 - ✓ $a \otimes c = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \otimes [2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]$
- 步骤为:
 - ✓ `a=[[2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9]]` # 不要忘记逗号!
 - ✓ `c= [2, 3, 4, 5, 6, 7, 8]`
 - ✓ `a = np.array(a)` # list转array
 - ✓ `c = np.array(c)`
 - ✓ `output = a * c`
 - ✓ `output = [[4 9 16 25 36 49 64]
 [6 12 20 30 42 56 72]]`

OpenCV与图像

- **import**

- ✓ **import**跟C/C++中的**#include**作用有点类似,都是为了调用定义在其他文件中的变量、函数或者类
- ✓ 但实现的区别很大
- ✓ 举例: **import cv2 as cv**
- ✓ **import cv2 as cv**, 从**opencv**导入**cv2**, 并命名为**cv**, 就可以调用**opencv**的函数**imread**了
- ✓ **cv.imread**是**numpy**的子类, 因此 **cv.imread("./img/xxxx.png")**读进来的是数组
- ✓ **./**表示当前运行**py**文件的所在路径, **../**表示上级路径, 请问上上级路径?

OpenCV与图像

- 图像文件的读取

- ✓ **Imread**用于图像的读入

- ✓ 计算机视觉和深度学习方面的研究与应用都需要对图像做处理, 因此读入图像数据很重要

- ✓ `img = cv.imread("./img/xxxx.png")` # 可以用"`\`"吗?

- ✓ 括号中"`./`"表示**img**文件夹和脚本文件在同一个级别的路径中, 而名字为**xxxx.png**的图像在**img**文件夹中

- ✓ **切记:** "`./img/xxxx.png`"是相对路径

- ✓ 一定要养成用相对路径的习惯, 而不要用绝对路径如:
F:/Sample_Python/code_example/img/

OpenCV与图像

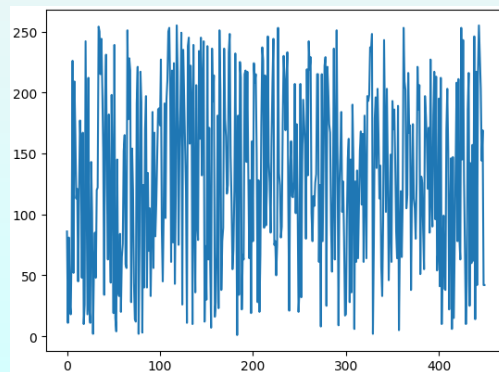
- **img_show.py**

- 图像操作

- ✓ `m, n, c = img.shape` # 获得图像的长宽与颜色通道数
- ✓ `img_r = img[:, :, 0]` # 取图像的红色通道
- ✓ `img_s = cv.resize(img_r, (x, y))` # 得到长宽为x, y的图像img_s. `resize`并不会改变img_r的长宽
- ✓ `cv.imshow('img_s', img_s)` # 显示图像
- ✓ `cv.waitKey(0)` # 设置显示图像的时间, 大于0的值表示显示的秒数

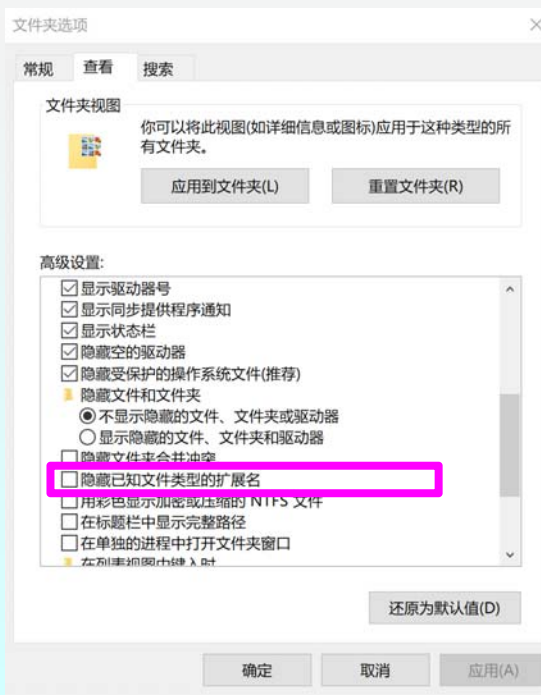
Matplotlib与曲线

- **import matplotlib.pyplot as plt**
- 图像操作
 - ✓ `Img=cv.resize(img, (x, y))` # 把图像长宽调整为x, y
 - ✓ `img = img[:, :, 0]` # 取图像红色通道
 - ✓ `Img_vector = sum(img, 2)` # 把图像每列元素相加, 得到一个向量
 - ✓ `plt.plot(img_vector)` # 画曲线
 - ✓ `plt.show()` # 展示曲线



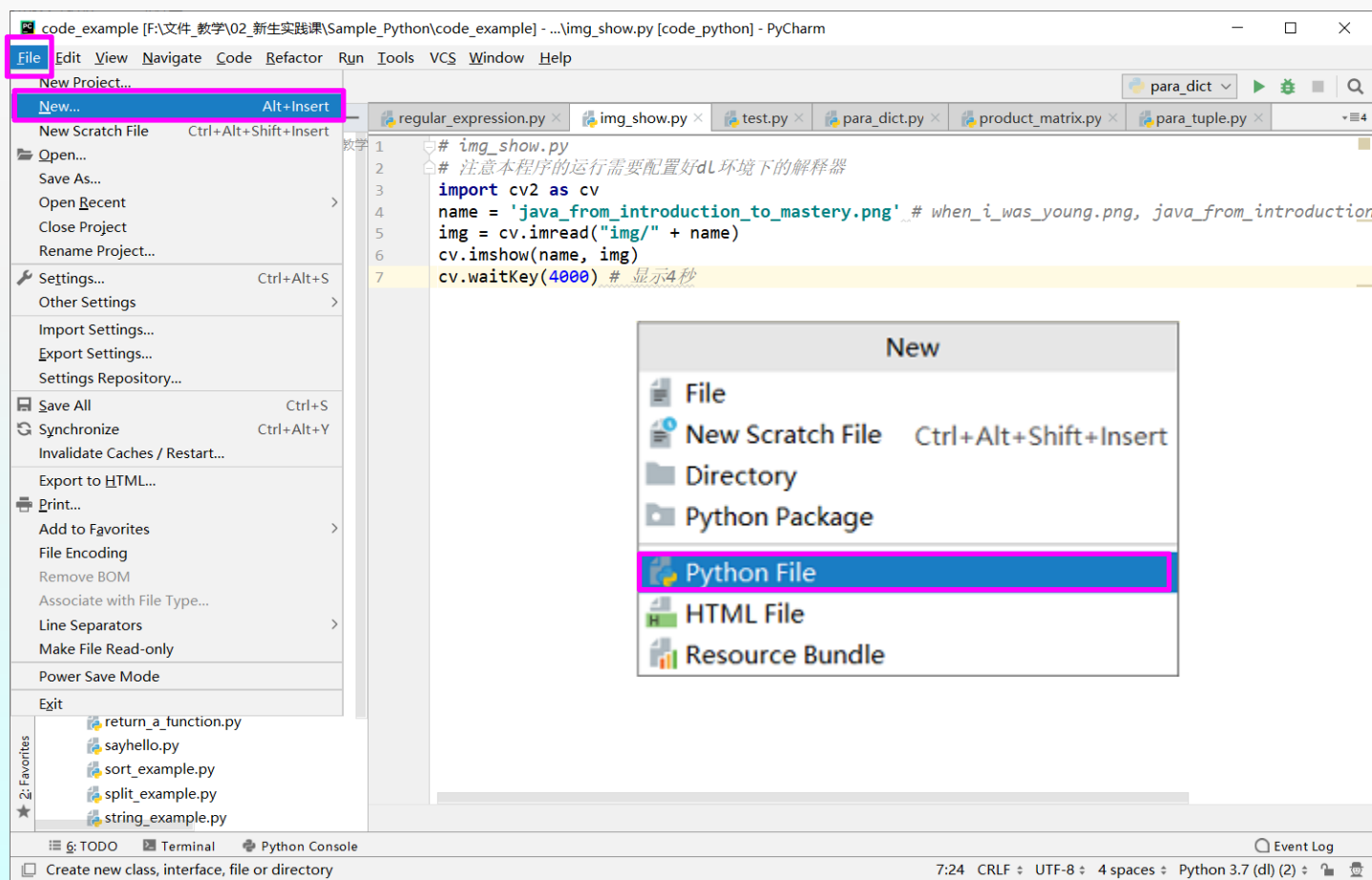
如何新建一个.py文件

- 方法1:可以先在工作路径下创建exe_07.txt文件,然后把后缀改为.py,方法:
 - ✓ 在资源管理器的工作路径所在文件夹的空白处,点鼠标右键→点新建→点文本文档→改文件名与后缀



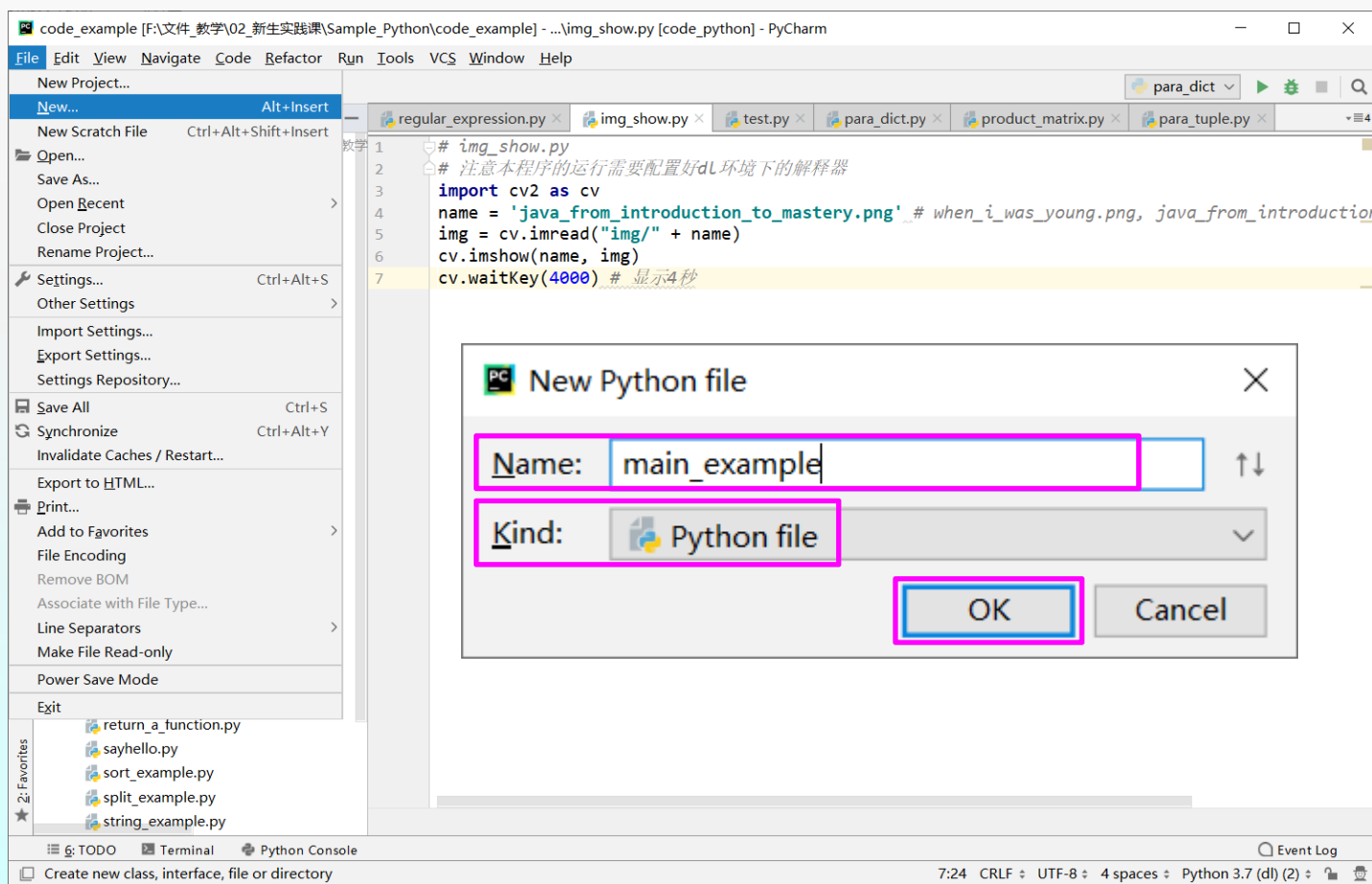
如何新建一个.py文件

- 方法2: 菜单File→New→Python File



如何新建一个.py文件

- 在对话框中输入文件名、选文件类型点ok



如何新建一个.py文件

- 可以发现在工作路径下有了刚创建的文件

