python_basic

官方文档

python基础

```
是一个免费、开源、跨平台、动态、面向对象的编程语言。
Python是一种跨平台的计算机编程语言，只要该平台下有Python解释器，那么Python编写的程序就能在该平台运行，目前Python支持的平台主要有：Linux，Windows，Mac os，Android，iOS等。Python完全可以创建大型软件，例如豆瓣、知乎服务端。Python语言底层是基于C语言，开发效率高，但是运行效率低。
动态编程语言是指可在运行阶段时执行那些在编译阶段执行的操作的编程语言。比如，在 JavaScript 中，我们可以在程序运行时改变变量的类型，或者为一个对象增加一个新属性或者方法。
```

python程序的执行方式

交互式

```
在命令行输入指令，回车即可得到结果。
1. 打开终端
2. 进入交互式：python3，ipython3
3. 编写代码：print(“hello world”)
4. 离开交互式：exit(),exit
```

文件式

```
将指令编写到.py文件，可以重复运行程序。
1. 编写文件。
2. 打开终端
3. 进入程序所在目录：cd 目录
4. 执行程序： python3 文件名
```

执行过程

```
计算机只能识别机器码(1010)，不能识别源代码(python)。
1. 由源代码转变成机器码的过程分成两类：编译和解释。
2. 编译：在程序运行之前，通过编译器将源代码变成机器码，例如：C语言。
-- 优点：运行速度快
-- 缺点：开发效率低，不能跨平台。
3. 解释：在程序运行之时，通过解释器对程序逐行翻译，然后执行。例如Javascript
-- 优点：开发效率高，可以跨平台；
-- 缺点：运行速度慢。
4. python是解释型语言，但为了提高运行速度，使用了一种编译的方法。编译之后得到pyc文件，存储了字节码（特定于Python的表现形式，不是机器码）。导模块才会编译。
源代码 -- 编译 --> 字节码 -- 解释 --> 机器码
|———1次——|
```

解释器类型

```
1. CPython（C语言开发）
2. Jython （java开发）
3. IronPython （.net开发）
```

数据基本运算

注释

```
给人看的，通常是对代码的描述信息。
1. 单行注释：以#号开头。
2. 多行注释：三引号开头，三引号结尾。
```

函数

```
表示一个功能，函数定义者是提供功能的人，函数调用者是使用功能的人。
例如：
1. print(数据) 作用：将括号中的内容显示在控制台中
2. 变量 = input("需要显示的内容") 作用：将用户输入的内容赋值给变量
```

变量

```
"""
变量：在内存中存储数据
程序运行在哪里？ -- 内存
程序在处理什么？ -- 数据
定义：关联一个对象的标识符。
命名：必须是字母或下划线开头，后跟字母、数字、下划线。
    不能使用关键字(蓝色)，否则发生语法错误：SyntaxError: invalid syntax。
建议命名：字母小写，多个单词以下划线隔开。
    class_name
语法：变量名 = 对象
    变量名1 = 变量名2 = 对象
    变量名1, 变量名2, = 数据1, 数据2  优雅
价值：在内存中操作数据的技术
作用：在内存中存储数据。
语义：
     内存图
     变量名：真实内存地址的别名
     见名知意
     赋值号：将右边对象的地址复制给左边内存空间。


"""


# 创建单个变量
```

```
person_name01 = "吉多"
# 用多个数据，创建多个变量
person_name02, person_name03 = "范罗苏姆", "林纳斯"
# 用单个数据，创建多个变量
person_name04 = person_name05 = "托瓦兹"

del person_name04
del person_name01, person_name02

person_name06 = person_name05 + person_name05

# 根据内存图理解-变量交换
data1, data2 = 1, 2
temp = data1
data1 = data2
data2 = temp
```

del 语句

```
1. 语法：
del 变量名1, 变量名2
2. 作用：
用于删除变量，同时解除与对象的关联.如果可能则释放对象。
3. 自动化内存管理的引用计数：
每个对象记录被变量绑定(引用)的数量，当为0时被销毁。
```

优雅的赋值

```
"""
变量名1, 变量名2, = 数据1, 数据2
变量名1, 变量名2, ... = 可迭代对象
    自动解包可迭代对象
"""

```

```
# content of demo01.py
c, d = 1, 2
```

- 字节码分析

```
D:\doing\study\demo\python_project>python -m dis demo01.py
  1           0 LOAD_CONST               0 ((1, 2))
              2 UNPACK_SEQUENCE          2
              4 STORE_NAME               0 (c)
              6 STORE_NAME               1 (d)
              8 LOAD_CONST               1 (None)
             10 RETURN_VALUE
```

执行顺序

```
# 从上到下，从左到右，遇到赋值语句先执行右边的
D:\doing\study\demo\python_project>ipython
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.25.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a, b = b, a = 1, 2

In [2]: a
Out[2]: 2

In [3]: b
Out[3]: 1
```

核心数据类型

```
1. 在python中变量没有类型，但关联的对象有类型。
2. 通过type()函数可查看。
```

空值对象None

```
1. 表示不存在的特殊对象。
2. 作用：占位和解除与对象的关联。
```

整形int

```
1. 表示整数，包含正数、负数、0。
如：  -5, 100, 0
2. 字面值：
十进制：5
二进制：0b开头，后跟1或者0  binary
八进制：0o开头，后跟0~7  octal
十六进制：0x开头，后跟0~9,A~F,a~f   hex
3. 小整数对象池：CPython 中整数 -5 至 256,永远存在小整数对象池中,不会被释放并可重复使用。
```

浮点型float

```
1. 表示小数，包含正数、负数，0.0。
2. 字面值：
小数：1.0  2.5
科学计数法：e/E （正负号） 指数
•       1.23e-2 （等同于0.0123）
•       1.23456e5(等同于123456.0)
```

字符串str

```
是用来记录文本信息(文字信息)。
字面值：双引号
```

复数complex

```
由实部和虚部组成的数字。
虚部是以j或J结尾。
字面值：  1j   1+1j  1-1j
```

布尔bool

```
用来表示真和假的类型
True 表示真(条件满足或成立)，本质是1
False 表示假(条件不满足或不成立)，本质是0
```

数据类型转换

```python
# 转换为整形：int(数据)
# 转换为浮点型:float(数据)
# 转换为字符串:str(数据)
# 转换为布尔:bool(数据)
# 结果为False：bool(0)  bool(0.0)  bool(None)
# 混合类型自动升级：
1 + 2.14  返回的结果是 3.14
1 + 3.0  返回结果是：  4.0
# str -> int
data01 = int("3")
# int -> str
data02 = str(5)

# str -> float
data03 = float("1.2")
# float -> str
data04 = str(1.2)

# int -> float
data05 = float(250)
# float -> int
data06 = int(1.9)
print(data06)  # 1 向下取整(截断删除)

```

```
# 注意：字符串转换为其他类型时，
# 必须是目标类型的字符串表达形式
# print(int("10.5"))  # 报错
# print(float("abc"))# 报错
```

运算符

算术运算符

```
+    加法
-    减法
*    乘法
/    除法：结果为浮点数
//   地板除：除的结果去掉小数部分
%    求余
**   幂运算
优先级从高到低：
()
**
* / % //
+  -
```

增强运算符

```
y += x      等同于 y = y + x
y -= x      等同于 y = y - x
y *= x      等同于 y = y * x
y /= x      等同于 y = y / x
y //= x     等同于 y = y // x
y %= x      等同于 y = y % x
y **= x     等同于 y = y ** x
```

比较运算符

```
<       小于
<=      小于等于
>       大于
>=      大于等于
==      等于
!=      不等于
返回布尔类型的值
比较运算的数学表示方式：0 <= x <= 100
```

逻辑运算符

这些属于布尔运算，按优先级升序排列：

运算	结果：	备注
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)

注释：

1. 这是个短路运算符，因此只有在第一个参数为假值时才会对第二个参数求值。
2. 这是个短路运算符，因此只有在第一个参数为真值时才会对第二个参数求值。
3. `not` 的优先级比非布尔运算符低，因此 `not a == b` 会被解读为 `not (a == b)` 而 `a == not b` 会引发语法错误。

身份运算符

```
    语法：
          x is y
          x is not y
作用：
is 用于判断两个对象是否是同一个对象，是时返回True，否则返回False。is 的本质就是通过 i d 函数进行判断的
    is not 的作用与is相反
### 优先级
•    高到低：
算数运算符
比较运算符
快捷运算符
身份运算符
逻辑运算符
```

语句

行

```
1. 物理行：程序员编写代码的行。
2. 逻辑行：python解释器需要执行的指令。
3. 建议一个逻辑行在一个物理行上。
4. 如果一个物理行中使用多个逻辑行，需要使用分号；隔开。
5.  如果逻辑行过长，可以使用隐式换行或显式换行。
隐式换行：所有括号的内容换行，称为隐式换行
          括号包括：  ()  []  {} 三种
显式换行：通过折行符 \ （反斜杠）换行，必须放在一行的末尾，目的是告诉解释器，下一行也是本行的语句。
```

pass 语句

```
通常用来填充语法空白。
```

选择语句

```
"""
If  elif    else    语句
1.作用:
    让程序根据条件选择性的执行语句。
2.语法:
if 条件1:
    语句块1
elif 条件2:
    语句块2
else:
    语句块3
3.说明:
    elif 子句可以有0个或多个。
    else 子句可以有0个或1个，且只能放在if语句的最后。
4.if 语句的真值表达式
if 100:
    print("真值")
等同于
if bool(100):
    print("真值")
5.条件表达式
    语法: 变量 = 结果1 if 条件 else 结果2
    作用: 根据条件(True/False) 来决定返回结果1还是结果2。
"""

```

循环语句

```
"""
while语句 适合根据条件循环执行。
1.作用:
可以让一段代码满足条件，重复执行。
2.语法:
while 条件:
    满足条件执行的语句
else:
    不满足条件执行的语句
3.说明:
    else子句可以省略。
    在循环体内用break终止循环时,else子句不执行。
"""

```

for 语句

```
"""
```

```
适合执行预定次数。
1.作用：
    用来遍历可迭代对象的数据元素。
    可迭代对象是指能依次获取数据元素的对象，例如：容器类型。
2.语法：
for 变量列表 in 可迭代对象：
    语句块1
else：
    语句块2
3.说明：
    else子句可以省略。
    在循环体内用break终止循环时，else子句不执行。

range 函数
1.作用：
    用来创建一个生成一系列整数的可迭代对象(也叫整数序列生成器)。
2.语法：
    range(开始点，结束点，间隔)
3. 说明：
函数返回的可迭代对象可以用for取出其中的元素
返回的数字不包含结束点
开始点默认为0
间隔默认值为1
"""
```

跳转语句

```
# break 语句
1. 跳出循环体，后面的代码不再执行。
2. 可以让while语句的else部分不执行。
# continue 语句
跳过本次，继续下次循环。
```

容器类型

通用操作

```
# 数学运算符
1. +：用于拼接两个容器
2. +=：用原容器与右侧容器拼接，并重新绑定变量
3. *：重复生成容器元素
4. *=：用原容器生成重复元素， 并重新绑定变量
5. < <= > >= == !=：依次比较两个容器中元素，一但不同则返回比较结果。
# 成员运算符
1.语法：
    数据 in 序列
    数据 not in 序列
2. 作用：
    如果在指定的序列中找到值，返回bool类型。
# 索引index

1.作用：访问容器元素
```

```
2.语法：容器[整数]
3.说明：
正向索引从0开始，第二个索引为1，最后一个为len(s)-1。
反向索引从-1开始，-1代表最后一个，-2代表倒数第二个，以此类推，第一个是-len(s)。
# 切片slice
1.作用：
从容器中取出相应的元素重新组成一个容器。
2.语法：
容器[(开始索引):(结束索引)(:(步长))]
3.说明：
小括号()括起的部分代表可省略
结束索引不包含该位置元素
步长是切片每次获取完当前元素后移动的偏移量
# 内建函数
1. len(x)    返回序列的长度
2. max(x)    返回序列的最大值元素
     内置函数max对字典进行操作时，使用key做参数，字典的key类型要一致，且能比较，否则将报错。
3. min(x)    返回序列的最小值元素
4. sum(x)    返回序列中所有元素的和(元素必须是数值类型)
```

字符串 str

```
# 定义
由一系列字符组成的不可变序列容器，存储的是字符的编码值。
# 编码
1.字节byte：计算机最小存储单位，等于8 位bit.
2.字符：单个的数字，文字与符号。
3.字符集(码表)：存储字符与二进制序列的对应关系。
4.编码：将字符转换为对应的二进制序列的过程。
5.解码：将二进制序列转换为对应的字符的过程。
6.编码方式：
--ASCII编码：包含英文、数字等字符，每个字符1个字节。
--GBK编码：兼容ASCII编码，包含21003个中文；英文1个字节，汉字2个字节。
--Unicode字符集：国际统一编码，旧字符集每个字符2字节，新字符集4字节。
-- UTF-8编码：Unicode的存储与传输方式，英文1字节，中文3字节。
# 相关函数
1.ord(字符串):返回该字符串的Unicode码。
2.chr(整数):返回该整数对应的字符串。
# 字面值
# 单引和双引号的区别
1.单引号内的双引号不算结束符
2.双引号内的单引号不算结束符
# 三引号作用
1. 换行会自动转换为换行符\n
2. 三引号内可以包含单引号和双引号
3. 作为文档字符串
# 转义字符
1. 改变字符的原始含义。
\'  \"  \"""  \n  \\  \t  \0 空字符
2. 原始字符串：取消转义。
    a = r"C:\newfile\test.py"

```

```
相邻的两个或多个 字符串字面值 （引号标注的字符）会自动合并：
>>> 'Py' 'thon'
'Python'


```

`printf` 风格的字符串格式化

格式字符串语法

格式字符串字面值

列表 list

```
### 定义
由一系列变量组成的可变序列容器。
### 基础操作
1. 创建列表：
列表名 = []
列表名 = list(可迭代对象)
2. 添加元素：
列表名.append(元素)
列表.insert(索引，元素)
3. 定位元素：
索引、切片
4. 遍历列表：
    正向：
    for 变量名 in 列表名：
        变量名就是元素
        反向：
    for 索引名 in range(len(列表名)-1,-1,-1)：
        列表名[索引名]就是元素
5. 删除元素：
列表名.remove(元素)
    del 列表名[索引或切片]
### 深拷贝和浅拷贝
浅拷贝：复制过程中，只复制一层变量，不会复制深层变量绑定的对象的复制过程。
深拷贝：复制整个依懒的变量。

### 列表VS字符串
1. 列表和字符串都是序列，元素之间有先后顺序关系。
2. 字符串是不可变的序列，列表是可变的序列。
3. 字符串中每个元素只能存储字符，而列表可以存储任意类型。
4. 列表和字符串都是可迭代对象。
5. 函数：
将多个字符串拼接为一个。
result = "连接符".join(列表)
将一个字符串拆分为多个。
列表 = “a-b-c-d”.split(“分隔符”)
### 列表推导式
1. 定义：
```

使用简易方法，将可迭代对象转换为列表。
2. 语法：
变量 = [表达式 for 变量 in 可迭代对象]
变量 = [表达式 for 变量 in 可迭代对象 if 条件]
3. 说明：
如果if真值表达式的布尔值为False,则可迭代对象生成的数据将被丢弃。
列表推导式嵌套
1. 语法：
变量 = [表达式 for 变量1 in 可迭代对象1 for 变量2 in可迭代对象2]
2. 传统写法：
```
result = []
for r in  ["a", "b", "c"]:
  for c in ["A", "B", "C"]:
    result.append(r + c)
```
3. 推导式写法：
```
result = [r + c for r in list01 for c in list02]
```

元组tuple

定义
1. 由一系列变量组成的不可变序列容器。
2. 不可变是指一但创建，不可以再添加/删除/修改元素。
基础操作
1. 创建空元组：
元组名 = ()
元组名 = tuple()
2. 创建非空元组：
元组名 = (20,)
  元组名 = (1, 2, 3)
元组名 = 100,200,300
元组名 = tuple(可迭代对象)
3. 获取元素：
索引、切片
4. 遍历元组：
    正向：
    for 变量名 in 列表名：
        变量名就是元素
        反向：
    for 索引名 in range(len(列表名)-1,-1,-1):
        元祖名[索引名]就是元素
作用
1. 元组与列表都可以存储一系列变量，由于列表会预留内存空间，所以可以增加元素。
2. 元组会按需分配内存，所以如果变量数量固定，建议使用元组，因为占用空间更小。
3. 应用：
变量交换的本质就是创建元组：x, y = y, x
格式化字符串的本质就是创建元祖："姓名:%s, 年龄:%d" % ("tarena", 15)

字典dict

定义

1. 由一系列键值对组成的可变映射容器。

```
2. 映射：一对一的对应关系，且每条记录无序。
3. 键必须惟一且不可变(字符串/数字/元组)，值没有限制。
### 基础操作
1. 创建字典：
字典名 = {键1：值1，键2：值2}
字典名 = dict (可迭代对象)
2. 添加/修改元素：
语法：
  字典名[键] = 数据
说明：
  键不存在，创建记录。
  键存在，修改映射关系。
3. 获取元素：
变量 = 字典名[键]  # 没有键则错误
4. 遍历字典：
•   for 键名 in 字典名：
•       字典名[键名]
for 键名,值名 in 字典名.items()：
语句
5. 删除元素：
del 字典名[键]
### 字典推导式
1. 定义：
使用简易方法，将可迭代对象转换为字典。
2. 语法：
{键:值 for 变量 in 可迭代对象}
  {键:值 for 变量 in 可迭代对象 if 条件}
### 字典VS 列表
1. 都是可变容器。
2. 获取元素方式不同,列表用索引,字典用键。
3. 字典的插入,删除,修改的速度快于列表。
4. 列表的存储是有序的,字典的存储是无序的。
```

集合set

```
### ****定义\****
1. 由一系列不重复的不可变类型变量组成的可变映射容器。
2. 相当于只有键没有值的字典(键则是集合的数据)。
### 基础操作
1. 创建空集合：
集合名 = set()
集合名 = set(可迭代对象)
2. 创建具有默认值集合：
集合名 = {1, 2, 3}
集合名 = set(可迭代对象)
3. 添加元素：
集合名.add(元素)
4. 删除元素：
集合名.discard(元素)
### 运算
1. 交集&：返回共同元素。

s1 = {1, 2, 3}
```

```
s2 = {2, 3, 4}
s3 = s1 & s2  # {2, 3}
2. 并集：返回不重复元素
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = s1 | s2  # {1, 2, 3, 4}
3. 补集-：返回只属于其中之一的元素
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1 - s2  # {1} 属于s1但不属于s2
补集^：返回不同的的元素
    s1 = {1, 2, 3}
    s2 = {2, 3, 4}
    s3 = s1 ^ s2  # {1, 4}  等同于(s1-s2 | s2-s1)
4. 子集<：判断一个集合的所有元素是否完全在另一个集合中
5. 超集>：判断一个集合是否具有另一个集合的所有元素
    s1 = {1, 2, 3}
    s2 = {2, 3}
    s2 < s1  # True
    s1 > s2  # True
6. 相同或不同== !=：判断集合中的所有元素是否和另一个集合相同。
 s1 = {1, 2, 3}
    s2 = {3, 2, 1}
    s1 == s2  # True
    s1 != s2  # False
子集或相同，超集或相同 <=  >=
### 集合推导式
1. 定义：
使用简易方法，将可迭代对象转换为集合。
2. 语法：
{表达式 for 变量 in 可迭代对象}
   {表达式 for 变量 in 可迭代对象 if 条件}
```

固定集合 frozenset

```
### 定义
不可变的集合。
#### 作用
固定集合可以作为字典的键，还可以作为集合的值。
#### 基础操作
创建固定集合：frozenset(可迭代对象)
#### 运算
等同于set
```

函数 function

```
## 定义
1. 用于封装一个特定的功能，表示一个功能或者行为。
2. 函数是可以重复执行的语句块，可以重复调用。
功能、参数、返回值

## 作用
```

提高代码的可重用性和可维护性（代码层次结构更清晰）。
定义函数
1. 语法：
def 函数名(形式参数)：
    函数体
2. 说明：
def 关键字：全称是define，意为"定义"。
函数名：对函数体中语句的描述，规则与变量名相同。
形式参数：方法定义者要求调用者提供的信息。
函数体：完成该功能的语句。
3. 函数的第一行语句建议使用文档字符串描述函数的功能与参数。
调用函数
1. 语法：函数名(实际参数)
2. 说明：根据形参传递内容。
返回值
1. 定义：
方法定义者告诉调用者的结果。
2. 语法：
return 数据
3. 说明：
return后没有语句，相当于返回 None。
函数体没有return，相当于返回None。
可变 / 不可变类型在传参时的区别
1. 不可变类型参数有：
数值型(整数，浮点数，复数)
布尔值bool
None 空值
字符串str
元组tuple
固定集合frozenset
2. 可变类型参数有：
列表 list
字典 dict
集合 set
3. 传参说明：
不可变类型的数据传参时，函数内部不会改变原数据的值。
可变类型的数据传参时，函数内部可以改变原数据。

函数参数

实参传递方式argument
位置传参
定义：实参与形参的位置依次对应。
序列传参
定义：实参用*将序列拆解后与形参的位置依次对应。
关键字传参
定义：实参根据形参的名字进行对应。
字典关键字传参
1. 定义：实参用**将字典拆解后与形参的名字进行对应。
2. 作用：配合形参的缺省参数，可以使调用者随意传参。
形参定义方式parameter

缺省参数

```
1. 语法:
def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):
    函数体
2. 说明:
缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。
缺省参数可以有0个或多个，甚至全部都有缺省参数。
#### 位置形参
语法:
def 函数名(形参名1, 形参名2, ...):
函数体
##### 星号元组形参
1. 语法:
def 函数名(*元组形参名):
 函数体
2. 作用:
收集多余的位置传参。
3. 说明:
一般命名为'args'
形参列表中最多只能有一个
#### 命名关键字形参
1. 语法:
def 函数名(*, 命名关键字形参1, 命名关键字形参2, ...):
函数体
def 函数名(*args, 命名关键字形参1, 命名关键字形参2, ...):
        函数体
2. 作用:
强制实参使用关键字传参
##### 双星号字典形参
1. 语法:
def 函数名(**字典形参名):
函数体
2. 作用:
收集多余的关键字传参
3. 说明:
一般命名为'kwargs'
形参列表中最多只能有一个
#### 参数自左至右的顺序
位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参
```

作用域LEGB

```
1. 作用域：变量起作用的范围。
2. Local局部作用域：函数内部。
3. Enclosing  外部嵌套作用域 ： 函数嵌套。
4. Global全局作用域：模块(.py文件)内部。
5. Builtin内置模块作用域：builtins.py文件。
## 变量名的查找规则
1. 由内到外：L -> E -> G -> B
2. 在访问变量时，先查找本地变量，然后是包裹此函数外部的函数内部的变量，之后是全局变量，最后是内置变量。
## 局部变量

1. 定义在函数内部的变量(形参也是局部变量)
```

2. 只能在函数内部使用
3. 调用函数时才被创建，函数结束后自动销毁
全局变量
1. 定义在函数外部,模块内部的变量。
2. 在整个模块(py文件)范围内访问（但函数内不能将其直接赋值）。
global 语句
1. 作用:
在函数内部修改全局变量。
在函数内部定义全局变量(全局声明)。
2. 语法:
global 变量1, 变量2, …
3. 说明
在函数内直接为全局变量赋值，视为创建新的局部变量。
不能先声明局部的变量，再用global声明为全局变量。
nonlocal 语句
1. 作用:
在内层函数修改外层嵌套函数内的变量
2. 语法
nonlocal 变量名1,变量名2, ...
3. 说明
在被嵌套的内函数中进行使用