版本 v1.0.0

Python 3.7 总结

最新版本:Python 相关
https://liushilive.github.io/

基本类型

Integer,Floating,Boolean,string

```
int 783  0 -192  0b010  0o642  0xF3
                 二进制  八进制  十六进制
float 9.23 0.0  -1.7e-6
                 ×10^-6
bool True  False
str "One\nTwo"   多行文本：
        换行      """X\tY\tZ
                  1\t2\t3"""
    'it\'s'          制表符
      转义
                  不可变对象
```

容器类型

- **序列**：下标访问，值可重复

```
list [1,5,9]      ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)     [11,"y",7.4]   ("mot",)  ()
值不可修改(不可变对象)  用逗号分隔每个元素 →tuple
str（字符串为有序的序列）                  ""
                                           ''
```

- **关键字容器**，无序（字典在 3.6 版本后为插入顺序），关键字访问

```
字典  dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
（键值对）  {1:"one",3:"three",2:"two",3.14:"π"}
集合  set {"key1","key2"}  {1,9,3,0}               set()
keys=(基础类型,不可变对象...)                       空
```

标识符

变量、函数、模块、类等
任何对象的名称

字母、数字、下划线（_）、Unicode字符

- 首字母只能为字母、Unicode、下划线
- 不可为系统关键字
- 严格区分大小写
  - ☺ `a toto x7 y_max BigOne`
  - ☹ ~~`8y and for`~~

变量 & 赋值

`=`

赋值 ⇔ **绑定** *值到变量名*

1) 右侧为表达式
2) 左侧为变量名

```
x=1.2+8+sin(y)
a=b=c=0  连续赋值
y,z,r=9.2,-7.6,0  多重赋值，自动解包
a,b=b,a  交换两个值
a,*b=seq
*a,b=seq   解包赋值
x+=3 自增 ⇔ x=x+3
x-=2 自减 ⇔ x=x-2
x=None  空值《常量》
del x   删除 x
```

转换

type(expression)

```
int("15") → 15
int(15.56) → 15
float("-11.24e8") → -1124000000.0
round(15.56,1)→ 15.6     四舍五入，保留一位小数点
bool(x)   x 为 空、 None 、空容器、0 返回False；其他返回True
str(x) → "..."
chr(64) → '@'    ord('@') → 64    code ↔ char
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {3:'three',1:'one'}
set(["one","two"]) → {'one','two'}
```

将列表拼接为字符串

```
':'.join(['toto','12','pswd']) → 'toto:12:pswd'
```

将字符串按空格拆分为列表

```
"words with spaces".split() → ['words','with','spaces']
```

将字符串按指定分隔符拆分为列表

```
"1,4,8,2".split(",") → ['1','4','8','2']
```

列表解析

```
[int(x) for x in ('1','29','-3')] → [1, 29, -3]
```

序列索引

列表、元组、字符串 ...

负索引下标	-5	-4	-3	-2	-1	
正索引下标	0	1	2	3	4	
lst=[10,	20,	30,	40,	50]		
正向下标	0	1	2	3	4	5
负向下标	-5	-4	-3	-2	-1	

元素个数 `len(lst)→5`

索引下标从 0 开始（0 到 4）

通过 `lst[index]` 访问单个元素

```
lst[0] → 10 ⇒第一个      lst[1]→20
lst[-1] → 50 ⇒倒数第一个  lst[-2]→40
```

在可变对象序列（list）中

使用 `del lst[3]` 删除下标为 3 的元素

使用 `lst[4]=25` 修改下标为 4 的元素

范围引用（切片） `lst[下限:上限:步长]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  序列浅拷贝
```

忽略上/下限切片 → 从开始到结束，默认步长为 1，正步长从下限往右，负步长从下限往左

对于可变对象序列（`list`），使用 `del lst[3:5]` 删除多个元素，使用 `lst[1:4]=[15,25]` 修改多个元素。

布尔逻辑

比较运算:`< > <= >= == = !=`

`a and b` 逻辑与：有假为假

`a or b` 逻辑或：有真为真

注意：`and` 与 `or` 返回 `a` 或 `b`
⇒确保 `a` 与 `b` 为**布尔值**

`not a` 逻辑非：取反

`True` `False` 布尔值：**注意大小写**

语句块

```
申明:
代码块 1...
    ⋮
声明:
    代码块 2...
    ⋮
块后的下一个语句 1
```

注意缩进！

建议以 4 个空格作为缩进

模块

module `truc` ⇔ 文件 `truc.py`

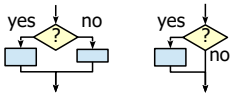
```
from monmod import nom1,nom2 as fct
        → 直接访问名称，别名使用 as
import monmod → 通过 monmod.nom1 访问
```

在python path路径中搜索模块和包（cf. `sys.path`）

条件语句

只执行条件为 `True` 的语句块

```
if 逻辑条件:
    语句块
```



可以搭配几个 *elif*，*elif*...
以及最后一个 *else*
只会执行第一个条件为 `True` 的语句块

```
if age <= 18:
  state="女孩"
elif age > 65:
  state="老妇人"
else:
  state="美女"
```

有一个变量 x:

```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

数学

运算优先级

```
**
* / // %
+ -
== > >= < <= !=
not
and
or
```

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

弧度角

```
from math import sin,pi...
sin(pi/4)→0.707...
cos(2*pi/3)→-0.4999...
sqrt(81)→9.0
log(e**2)→2.0
ceil(12.5)→13
floor(12.5,12)→12
modules math, statistics, random, decimal,
fractions, numpy, etc, etc. (cf. doc)
```

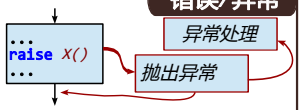
错误/异常

抛出异常：

```
raise ExcClass(...)
```

异常处理：

```
try:
    可能发生异常代码块
except ExcClass as e:
    异常处理代码块
```



`finally`
不管在什么情况下都会执行

条件循环

在条件为 True 的情况下执行指令块

```
while 逻辑条件:
    语句块
```

当心死循环！！！

```
s = 0          } 在循环之前初始化
i = 1          } 具有至少一个变量值的条件（i）
while i <= 100:
    s = s + i**2
    i = i + 1      # 改变条件 i 的值
print("sum:", s)
```

算法：

$$s=\sum_{i=1}^{i=100} i^2$$

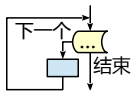
循环控制

break	结束循环
continue	跳过本轮循环
else	循环正常结束后执行

迭代循环

为容器或迭代器的每个元素执行语句块

```
for var in sequence:
    语句块
```



检查序列的值

```
s = "Some text"   } 在循环前初始化
cpt = 0
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found", cnt, "'e'")
```

循环变量，由 for 语句管理与赋值

算法：计算字符串中e的数目

检查序列索引
- 在索引处修改项
- 访问项目在索引(前/后)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:", lst, "-lost:", lost)
```

算法：将大于15的值替换为15，并记住被替换的值

同时检查序列的索引和值：

```
for idx,val in enumerate(lst):
```

好习惯：不要修改循环变量

打印

```
print("v=",3,"cm :",x,",",y+4)
```

控制台打印内容：文本值、变量、表达式

print 选项：

- sep=" " 项目分隔符，默认空格
- end="\n" 打印结束，默认换行
- file=sys.stdout 打印到文件，默认标准输出

输入

```
s = input("请输入一个值:")
```

从控制台接受输入，直到 Enter，默认返回 str

整数序列

```
range([start,] end [,step])
```

start 默认 0，end 不包含在序列中，step 有符号，默认 1

```
range(5)→ 0 1 2 3 4        range(2,12,3)→ 2 5 8 11
range(3,8)→ 3 4 5 6 7      range(20,5,-5)→ 20 15 10
```

range提供了一个根据需要构造的不可变整型序列

容器对象通用操作

```
len(c)→ 元素个数
min(c)  max(c)  sum(c)
sorted(c)→ list 已排序副本
val in c →  布尔值，成员关系
enumerate(c)→  迭代对象(index, value)
zip(c1,c2…)→  在具有相同索引的 ci 元素的元组上迭代
all(c)→ True   如果 c 内所有元素为真，否则 False
any(c)→ True   如果 c 内至少有一个元素为真，否则 False
c.clear()      清空容器对象内部所有元素
```

注意：对于字典和集合，这些操作使用 keys。

有序容器对象（lists, tuples, strings...）

```
reversed(c)→ 翻转      c*5→ 重复      c+c2→ 连接
c.index(val)→ 下标     c.count(val)→ 计数
```

```
import copy
copy.copy(c)→ 容器对象浅副本
copy.deepcopy(c)→ 容器对象深副本
```

列表对象基本操作

修改原始列表

```
lst.append(val)          添加元素到最末尾
lst.extend(seq)          最末尾追加一个序列
lst.insert(idx,val)      在指定位置插入值
lst.remove(val)          移除第一个元素 val
lst.pop([idx])→value     弹出最后一个元素
lst.sort()  lst.reverse()  排序/颠倒列表
```

函数定义

```
def fct(x,y,z):
    """文档注释"""
    # 语句块
    return res
```

形参；返回值：如果没有结果返回，默认：return None

此块的参数和所有变量仅存在于块中，并在函数内部调用（请理解为“黑盒子”）

高级用法：

```
def fct(x, y, z, *args, a=3, b=5, **kwargs):
```

位置传参，*args 包裹位置传参（→tuple），默认值参数，**kwargs 包裹关键字传参（→dict）

函数调用

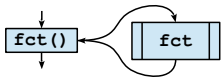
```
r = fct(3,i+2,2*i)
```

存储返回值　实参

通过函数名与括号调用函数

高级用法：

```
*sequence
**dict
```



字典对象基本操作

```
d[key]=valeur      del d[key]
d[key]→ valeur
d.update(d2)  更新/添加键值对
d.keys()
d.values()     →返回所有 键/值/键值对
d.items()
d.pop(key[,default])→ value
d.popitem()→ (key,value)
d.get(key[,default])→ value
d.setdefault(key[,default])→value
```

集合对象基本操作

运算符：

```
| → 并集
& → 交集
- ^ → 差集/对称差集
< <= > >= → 包含关系
```

```
s.update(s2)     s.copy()
s.add(key)   s.remove(key)
S.discard(key)   s.pop()
```

字符串对象基本操作

```
s.upper()        s.lower()
s.casefold()     s.capitalize()   s.title()
s.encode(encoding)  s.swapcase()  s.center([width,fill])
s.startswith(prefix[,start[,end]])
s.ljust([width,rempl])  s.rjust([width,rempl])
s.zfill([width])  s.split([sep])  s.join(sep)
s.endswith(suffix[,start[,end]])  s.strip([chars])
s.count(sub[,start[,end]])
s.partition(sep)→ (before,sep,after)
s.index(sub[,start[,end]])  s.find(sub[,start[,end]])
```

字符串格式化

```
name = "liu"
age = 28
person = {'name': 'liu', 'age': 28}

"%s. You are %s." % (name, age) # 官方不建议的方式
"{}. You are {}.".format(name, age) # str.format
"{1}. You are {0}.".format(age, name)
"{name}. You are {age}.".format(name=name, age=age)
"{name}. You are {age}.".format(**person)
f"{name}. You are {age}."  # f-Strings

  → "liu. You are 28."
```

如果你使用 Python 3.6+ 建议使用 f-Strings 方式格式化字符串
如果不是请使用 str.format
f-Strings 速度真的很快很快

文件

```
f = open("fic.txt","w",encoding="utf8")
```

变量 文件对象；文件路径；打开模式：'r' 读，'w' 写，'a' 追加，…'+' 'x' 'b' 't'；字符编码格式：utf8 ascii

写

```
f.write("coucou")
f.writelines(list of lines)
```

读

如果文件结束，读取空字符串

```
f.read([n]) → 下一个字符
    如果 n 没有指定，读到最后！
f.readlines([n])→ list of next lines
f.readline()    → next line
```

默认打开模式为 r，w 会覆盖原有内容

```
f.close()
```

不要忘记关闭文件

```
f.flush() 写缓存     f.truncate([size])  按顺序调整文件中的读/写进度，可修改为:
f.tell()→ 位置       f.seek(position[,origin])
```

强烈推荐使用上下文管理器进行文件读写

```
with open(…) as f:
    for line in f:
        # 处理行
```