

integer, float, boolean, string, bytes

```
int 783 0 -192 0b010 0o642 0xF3
float 9.23 0.0 零 -1.7e-6
bool True False
str "One\nTwo"
bytes b"toto\xfe\x775"
```

二进制 八进制 十六进制
多行字符串:
换行符
输出
输出 tab
十六进制 八进制
immutables

基本类型

有序序列, 通过下标快速索引, 值可以重复

```
list [1, 5, 9] ["x", 11, 8.9] ["mot"]
tuple (1, 5, 9) 11, "y", 7.4 ("mot",)
str bytes
```

值不可变
用逗号分割的几个对象 → tuple (也是有序的序列, 由字符/字节组成)

容器类型

键容器, 没有顺序, 通过键快速索引, 键值必须唯一

```
字典 dict {"key": "value"} dict(a=3, b=4, k="v")
集合 set {"key1", "key2"} {1, 9, 3, 0} set
frozenset 不可变集合 空对象
```

keys=hashable values (base types, immutables...)

标识符命名规范

a~z A~Z _ 后跟 0~9 a~z A~Z _

- 语法允许使用变音符号但是实际代码不要
- 禁用Python关键词
- 区分大小写

```
@ a toto x7 y_max BigOne
@ 8y and for
```

变量赋值

=

- 赋值: 将一个值绑定到一个变量
- 计算右侧表达式的值
- 将值按顺序赋给左侧变量

```
x=1.2+8+sin(y)
a=b=c=0 赋予同样的值
y, z, r=9.2, -7.6, 0 同时给多个变量赋值
a, b=b, a 交换值
a, *b=seq 将序列首个元素赋给a, 其余赋给b
*a, b=seq 将序列末个元素赋给b, 其余赋给a
x+=3 自增 ⇔ x=x+3
x-=2 自减 ⇔ x=x-2
x=None x等于未定义的常量值
del x 删除名称x
```

类型转换

int("15") → 15
int("3f", 16) → 63
int(15.56) → 15
float("-11.24e8") → -1124000000.0
round(15.56, 1) → 15.6
bool(x) x为null、空容器, None或False时为False, 其他为True
str(x) → "..." 转成字符串
chr(64) → '@' ord('@') → 64
repr(x) → "..." 转换成可解析的字符串
bytes([72, 9, 64]) → b'H\t@'
list("abc") → ['a', 'b', 'c']
dict([(3, "three"), (1, "one")]) → {'1': 'one', '3': 'three'}
set(["one", "two"]) → {'one', 'two'}
单个str + str列表 → 组合str
':'.join(['toto', '12', 'pswd']) → 'toto:12:pswd'
str 用空白字符分割 → str序列
"words with spaces".split() → ['words', 'with', 'spaces']
str 用指定的字符分割 → str序列
"1,4,8,2".split(",") → ['1', '4', '8', '2']
一个序列变成另一个序列
[int(x) for x in ('1', '29', '-3')] → [1, 29, -3]

序列索引

支持 lists, tuples, strings, bytes...

负索引	-5	-4	-3	-2	-1
正索引	0	1	2	3	4

```
lst=[10, 20, 30, 40, 50]
```

正切片: 0 1 2 3 4 5
负切片: -5 -4 -3 -2 -1

元素个数: len(lst) → 5
索引从 0 开始 (0 to 4)

通过 lst[index] 访问各个元素
lst[0] → 10 (第一个)
lst[-1] → 50 (最后一个)
lst[1] → 20
lst[-2] → 40

删除元素: del lst[3]
修改元素: lst[4]=25

通过 lst[切片起点: 终点: 步长] 获取子序列
lst[:-1] → [10, 20, 30, 40]
lst[1:-1] → [20, 30, 40]
lst[::2] → [10, 30, 50]
lst[::-1] → [50, 40, 30, 20, 10]
lst[1:-2] → [20, 30, 10]
lst[::] → [10, 20, 30, 40, 50]

不指定起止点会从头取到尾。
对于可变序列 (list), 使用 del lst[3:5] 删除切片; 使用 lst[1:4]=[15, 25] 修改切片

布尔运算

比较: < > <= >= == !=
(boolean results) ≤ ≥ = ≠

a and b 逻辑与 两个都为真
a or b 逻辑或 一个或两个都为真

陷阱: and 和 or 返回的是 a 的值或 b 的值。
⇒ 使用时确保指定的 a 和 b 为布尔类型。

not a 逻辑否
True } 逻辑真假 常量
False }

语句块

父语句:
语句块1...
父语句:
语句块2...
另一个语句块

将编辑器设置为每个Tab等于4个空格。

模块/名称导入

```
module truc ⇔ file truc.py
from monmod import nom1, nom2 as fct
import monmod
```

→ 直接导入的名字调用, 用 as 起别名
→ 通过 monmod.nom1 调用
在 python path (cf sys.path) 中搜索包和模块

条件语句

只有 if 条件为真时才执行

if 条件表达式:
语句块

if 后面可以跟多个 elif, 只有最后一个可以是 else, 也可以没有 else。只有第一个为真的语句块被执行。

```
if age <= 18:
    state = "Kid"
elif age > 65:
    state = "Retired"
else:
    state = "Active"
```

数学

浮点数 近似值
操作符: + - * / // % **
优先级 (...)
@ → matrix × python3.5+numpy
(1+5.3)*2 → 12.6
abs(-3.2) → 3.2
round(3.57, 1) → 3.6
pow(4, 3) → 64.0
usual order of operations

数学

三角函数的角度单位是弧度

```
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
```

modules math, statistics, random, decimal, fractions, numpy, etc. (cf. doc)

异常处理

主动抛出一个异常:
raise ExcClass(...)

异常处理:
try:
正常处理语句
except Exception as e:
异常处理语句

finally 无论是否有异常都会执行的语句

只要条件为真就重复执行

while 逻辑表达式:

语句块

s = 0

i = 1

在循环之前初始化

以变量i的值作为条件

while i <= 100:

s = s + i**2

i = i + 1

print("sum:", s)

修改循环条件变量i的值!

beware of infinite loops!

yes

no

条件循环语句

循环控制

break 立即退出循环

continue 转入下一次循环

否则迭代完毕正常退出

左侧代码实现的算法:

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

迭代循环语句

for var in sequence:

语句块

遍历序列的值

s = "Some text"

cnt = 0

在循环之前初始化

循环变量, 由for语句幅值

for c in s:

if c == "e":

cnt = cnt + 1

print("found", cnt, "e")

对dict/set循环 对键组成的列表循环

使用slices对子序列进行循环

遍历序列的索引

访问/修改索引处的元素

访问索引附近的其他元素

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: 将列表中大于15的值设定为15, 并记录被修改的值

同时遍历序列的索引与值

for idx, val in enumerate(lst):

打印显示

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

可打印对象: 常量, 变量, 表达式

print 参数选项:

sep = " " 不同项的分割符, 默认为空格

end = "\n" 打印结尾字符, 默认为换行符

file = sys.stdout 指定打印到文件还是命令行输出 (默认)

输入

s = input("输入提示字符:")

input 返回的是字符串, 需要手动转换为所要的类型

(cf. boxed Conversions on the other side).

基本容器操作

len(c) 元素数量

min(c) max(c) sum(c)

sorted(c) list 返回排序后的copy

val in c 返回布尔值, 从属关系运算符 in (或not in)

enumerate(c) 按照(index或value)迭代c的元素

zip(c1, c2...) 迭代c中index相同的子元素组成的元组

all(c) c的所有元素都为True时返回True, 否则返回False

any(c) c至少一个元素为True时返回True, 否则返回False

有序序列的特殊操作 (lists, tuples, strings, bytes...)

reversed(c) 反序

c*5 返回重复5遍的序列

c+c2 序列连接

c.index(val) 返回在序列中的位置

c.count(val) 返回在序列中出现的次数

import copy

copy.copy(c) 返回c的浅拷贝

copy.deepcopy(c) 返回c的深拷贝

注: 对于字典或集合, 这些操作使用键作为参数

操作列表

这些操作会修改原列表

lst.append(val) 向尾部追加单个值

lst.extend(seq) 向尾部追加一个序列的所有值

lst.insert(idx, val) 向idx处插入值

lst.remove(val) 删除发现的第一个val

lst.pop([idx]) 删除并返回idx处的值, 不指定idx则为最后一个值

lst.sort() lst.reverse() 将lst排序/反序

操作字典

d[key]=value 赋值

d[key] value 读值

d.clear() del d[key]

d.update(d2) 更新键对应的值或增加新键值对

d.keys() d.values() d.items() 获取keys/values/元素的可迭代集合

d.pop(key[, default]) value

d.popitem() (key, value)

d.get(key[, default]) value

d.setdefault(key[, default]) value

操作集合

操作符 (有对应的函数):

| 并集 union()

& 交集 intersection()

- 差集 difference() (前一个集合减去交集)

^ 对称差集 symmetric_difference() (前一个集合减去交集, 加上后一个集合减去交集)

< <= > >= 包含关系

s.update(s2) s.copy()

s.add(key) s.remove(key)

s.discard(key) s.clear()

s.pop()

函数定义

函数名 (标识符)

命名的输入参数

def fct(x, y, z):

"""函数说明, 描述文档"""

语句块 # 比如返回res的计算过程等

return res 返回计算结果值.

函数的输入参数与

语句块中的所有变量的作用域仅限于本语句块, 其生命周期仅限于函数被调用期间

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

x, y, z 是 (位置) 参数, 传入的参数值根据位置对应

*args 表示任意数量的 (位置) 参数 (-tuple)

a, b 是有默认值的参数 **kwargs 表示任意数量的有名参数 (-dict)

调用函数

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

返回赋值给r

给每个函数输入参数

传一个值或表达式

通过函数名称和用括号包裹的参数值来调用函数

高级用法:

通过*参数传入任意个参数(序列)

通过**参数传入任意个有名参数(字典)

操作字符串

s.startswith(prefix[, start[, end]])

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]])

s.is...() 判断字符类型 (ex. s.isalpha())

s.upper() s.lower() s.title() s.swapcase()

s.casefold() s.capitalize() s.center([width, fill])

s.ljust([width, fill]) s.rjust([width, fill]) s.zfill([width])

s.encode(encoding) s.split([sep]) s.join(seq)

格式化

格式指令

需要格式化的值

格式化

"modele{} {} {}".format(x, y, r) str

"{ 数据选择器: 格式指示符! 转换指示! }"

格式指示符组成:

填充字符 对齐方向 符号 最小宽度.精度~最大宽度 数据类型

"<"左对齐 ">"右对齐 "&"居中

"^"居中 "="用填充字符填充数字长度不足部分

"+"正数前加+号, 负数前加-号

"-"负数前加-号, 正数不加符号

" "正数加空格, 负数加-号

整型: b 二进制, c 字符, d 十进制 (默认), o 八进制, x 或 X 十六进制, 浮点数: e 或 E 科学法, f 或 F 固定位数, g 或 G 自适应 (默认), 字符串: s ...% 百分比

转换提示: s (适合可读) r (适合解析)

'f': {} s: {} '.format(1, 'two')->'f:1 s:two' '{:}{a[2]}'.format(1, 2, a=(5, 4, 3))

's': {} f: {} '.format(1, 'two')->'s:two f:1' '+1 2 3'

'a': {} s: {} '.format(1, 'two')->'a:1 s:two' '{:2.3f}'.format(45.72793) -> '+45.728'

'a:3, 1st:1, a again:3'

'<:10s'.format('toto')-> 'toto' '{x!z}'.format(x='I'm') -> 'I\'m'

'{:0:10s}'.format('toto')-> 'toto' '{x!s}'.format(x='I'm') -> 'I'm'

'{:1:<10s}'.format(8, 'toto')-> 'toto*****' '{:,}'.format(12345) -> '12,345'

文件

从磁盘文件读写数据

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

文件变量(句柄) 用来访问函数

文件路径及名称

打开模式

打开文件的编码格式:

cf. modules os, os.path and pathlib

写数据

f.write("coucou")

f.writelines(list of lines)

读数据

f.read([n])

f.readlines([n])

f.readline()

f.close()

f.flush()

f.truncate([size])

f.tell()

f.seek(position[, origin])

慎用方法: 使用with语句打开文件(用完后会自动关闭, 无需手动调用Close()), 通过循环读取文件的所有行:

with open(...) as f:

for line in f:

处理字符串line