

最新版参见：
<https://perso.limsi.fr/pointal/python:memento>

容器类型

容器类型

	<code>list</code>	<code>[1,5,9]</code>	<code>["x",11,8.9]</code>	<code>["mot"]</code>	<code>[]</code>
	<code>tuple</code>	<code>(1,5,9)</code>	<code>11,"y",7.4</code>	<code>("mot",)</code>	<code>()</code>
值不能修改 (不可变的, <i>immutable</i>)			用逗号分隔的表达式 → <code>tuple</code>		
	<code>str bytes</code>	(有序的字符 / 字节序列)			<code>""</code> <code>b""</code>
■ 关键字容器 , 未预设顺序, 用键来快速访问, 每个键是唯一的					
字典	<code>dict</code>	<code>{"key": "value"}</code>	<code>dict(a=3,b=4,k="v")</code>		<code>{}</code>
(键 / 值关联)		<code>{1: "one", 3: "three", 2: "two", 3.14: "n"}</code>			
集合	<code>set</code>	<code>{"key1", "key2"}</code>	<code>{1, 9, 3, 0}</code>	<code>set</code>	<code>()</code>
键 = 可哈希化的值 (基础数据类型, 不可变对象...)		<code>frozenset</code>	不可变的集, 冻结集		

类型转换

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) False 对应空 x, 空容器 x, None 或 False x; True 对应其他的 x
str(x) → "..." x 所对应的字符串表示 (参见背面的格式化内容)
chr(64) → '@' ord('@') → 64
repr(x) → "..." x 对应的直接表达的 (literal) 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'}
```

分隔字符串 **str** 与 **str** 序列 → 组装的 **str**

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

str 用空格分割 → 由 **str** 构成的 **list**

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

str 以 **str** 字符串来分割 → 由 **str** 构成的 **list**

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

同一种类型的序列 → 另一种类型的 **list** (通过列表推导式)

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

类型转换

变量赋值

赋值 \Leftrightarrow 将值与一个名字进行绑定

- 1) 计算表达式右侧的值
- 2) 按顺序对左侧的名字赋值

x=1.2+8*sin(y)
a=b=c=0 赋给同一个值
y,z,r=9.2,-7.6,0 多个赋值
a,b=b,a 值交换
a,*b=seq
***a,b=seq** } 将序列拆分成条目和列表

x+=3 自增 \Leftrightarrow **x=x+3**
x-=2 自减 \Leftrightarrow **x=x-2**
x=None 未定义 常量
del x 删除名字 **x**

序列容器的索引

适用于列表、元组、字符串、字节数组……

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	

项目计数

len(lst) → 5

索引从0开始
(此处对应0到4)

通过 **lst[start slice:end slice:step]** 的方式访问子序列

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]	序列的影子拷贝	

缺失的切片索引 → 从开始到结束。

对可易的序列 (**list**)，用 **del lst[3:5]** 进行删除，用赋值 **lst[1:4]=[15, 25]** 进行修改

序列容器的索引

通过 **lst[index]** 来访问单个元素 (**items**)

lst[0] → 10 ⇒ 第一个 **lst[1] → 20**

lst[-1] → 50 ⇒ 最后一个 **lst[-2] → 40**

对可易的序列 (**list**)，
用 **del lst[3]** 来删除，
用赋值 **lst[4]=25** 来修改

程序块

布尔逻辑

比较算符: `<` `>` `<=` `>=` `==` `!=`
 (返回布尔类型)
`a` **and** `b` 逻辑与 全部计算
`a` **or** `b` 逻辑或 其一或另一或全部

⚠ 陷阱: **and** 及 **or** 的返回值为 **a** 或为 **b** (简化计算)。

⇒ 要确保 **a** 及 **b** 为布尔类型

not a 逻辑非

True
False } 真与假常量

导入模块 / 名字

module **truc** ⇨ file **truc.py**

from monmod import nom1, nom2 as fct
 → 直接访问名字, 重命名用 **as**

import monmod → 访问通过 **monmod.nom1...**

模块及包的检索使用 **python path** (参见 **sys.path**)

仅当 **if** 的条件为真才执行程序块

if logical condition:
 → statements block

可紧跟着几个 **elif, elif...** 语句, 及唯一的 **else** 语句。只有第一个为真的条件块才会执行。

with a var **x**:

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

导入模块 / 名字

条件判断语句

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

数学

📌 浮点数……近似值	角度用弧度
Operators: + - * / // % **	<code>from math import sin, pi...</code>
Priority (...) $\times \div$ \uparrow \uparrow a^b	<code>sin(pi/4)→0.707...</code>
integer \div \div remainder	<code>cos(2*pi/3)→-0.4999...</code>
@ → matrix \times python3.5+ numpy	<code>sqrt(81)→9.0</code> $\sqrt{}$
<code>(1+5.3)*2→12.6</code>	<code>log(e**2)→2.0</code>
<code>abs(-3.2)→3.2</code>	<code>ceil(12.5)→13</code>
<code>round(3.57,1)→3.6</code>	<code>floor(12.5)→12</code>
<code>pow(4,3)→64.0</code>	模块 <code>math, statistics, random,</code>
📌 <i>usual order of operations</i>	<code>decimal, fractions, numpy, etc. (cf. doc)</code>

错误与异常

触发一个错误：
`raise ExcClass(...)`

错误处理过程：
`try:`
 → 正常处理的程序块
`except Exception as e:`
 → 处理错误的程序块

错误与异常

```
graph TD
    Start(( )) --> Normal[正常  
raise X()  
处理]
    Normal --> Try[处理错误的过程]
    Try --> Finally[finally 任何情况下的最终处理块]
    Try --> Raise[raise]
    Raise --> Try
```

`finally` 任何情况下的最终处理块

条件为真时执行的程序块

while 逻辑条件：
→ 程序块

s = 0
i = 1

循环体之前的初始化
至少有一个条件为真（此处为 i）

while i <= 100:
s = s + i**2
i = i + 1
print("sum:", s)

改变条件变量！

小心无限循环体！

条件循环语句

循环控制
break 中间退出
continue 下一次迭代
else 正常循环退出后的程序块

目标：
$$S = \sum_{i=1}^{100} i^2$$

枚举循环语句

for var in 序列：
→ 程序块

查看序列的值
s = "Some text"
cnt = 0
for c in s:
if c == "e":
cnt = cnt + 1
print("found", cnt, "e")

在 dict/set 上的循环 ⇒ 对序列的键 (keys) 进行循环
使用切片 (slices) 对的序列子集进行循环

查看序列的索引
修改索引处的条目
访问索引处 (前 / 后) 的条目

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)

目的：限制值大于 15，记录缺失的值

容器的常见操作

len(c) → 项目计数
min(c) max(c) sum(c)
sorted(c) → list 排序后的拷贝
val in c → 布尔型，从属操作符 in (不属于用 not in)
enumerate(c) → 迭代器，返回 (index, value)
zip(c1, c2...) → 迭代器，返回相同索引位置包含有 c1 项目的元组
all(c) → True 如果 c 中所有项目都为真，否则返回 False
any(c) → True 如果 c 中至少有一个项目为真，否则返回 False
特别针对有序序列容器 (列表，元组，字符串，字节数组...)
reversed(c) → 取反迭代器
c*5 → 复制
c+c2 → 连接
c.index(val) → 返回值得位置
c.count(val) → 事件统计
import copy
copy.copy(c) → 容器的影子拷贝
copy.deepcopy(c) → 容器的深拷贝

注：对于字典和集，这些操作针对的是键。

列表的操作

lst.append(val) 在尾部添加项目
lst.extend(seq) 在尾部添加项目序列
lst.insert(idx, val) 在索引处插入项目
lst.remove(val) 删除值为 val 的第一个项目
lst.pop([idx]) → value 删除并返回索引 idx (默认为结尾) 处的项目
lst.sort() lst.reverse() 列表在原地进行排序 / 反向

字典的操作

d[key]=值 d.clear()
d[key]→值 del d[key]
d.update(d2) { 更新 / 添加
关联数据
d.keys() → 用键 / 值 / 关联数组
d.values() 方式枚举访问
d.items() }
d.pop(key[,default]) → value
d.popitem() → (key,value)
d.get(key[,default]) → value
d.setdefault(key[,default]) → value

集的操作

运算符：
| → 合集 (竖线)
& → 交集
- ^ → 差集 / 不重复差集
< <= > >= → 包含关系
运算符也存在对应的方法。
s.update(s2) s.copy()
s.add(key) s.remove(key)
s.discard(key) s.clear()
s.pop()

字符串操作

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...() 字符类别测试 (如 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)

文件

向磁盘存储数据，再读回来
f = open("file.txt", "w", encoding="utf8")
用于操作的文件变量名 磁盘 (+ 路径...) 上的文件名 文件打开模式 文件文本的字符编码：
"r" 读
"w" 写
"a" 添加
... '+' 'x' 'b' 't' latin1, ...
参见模块 os, os.path 以及 pathlib

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

遇到文件结尾返回空字符串
f.read([n]) → 读后续字符
如果 n 未指定，直接读到文件结尾！
f.readlines([n]) → 后续文本行的列表
f.readline() → 后续文本行

读取
默认认为文本模式 t (读 / 写 str)，也可使用二进制模式 b (读 / 写 bytes)。格式之间的转换需要指定类型！
f.close() 用后请勿忘记关闭文件！
f.flush() 写入缓冲区 f.truncate([size]) 重新调整文件大小
对文件的读 / 写过顺序进行，可通过如下函数调整：
f.tell() → 位置 f.seek(position[, origin])
最常见操作：用 with 程序块打开文件 (会自动关闭文件)，然后对文件的行执行读循环。
with open(...) as f:
for line in f:
处理 line

函数定义

函数名 (标识符)
参数名字
def fct(x, y, z):
""" 函数文档 """
程序块，返回值计算，等
return res ← 调用的返回值，如果没有计算结果要返回，则：return None
此程序块中的参数及所有变量在程序被调用期间只存在于此程序块 (可视其为“黑盒子”)
进阶用法：def fct(x, y, z, *args, a=3, b=5, **kwargs):
*args 变量表示扩展位置参数 (→ tuple)，含默认值，
**kwargs 变量表示关键字参数 (→ dict)

函数调用

r = fct(3, i+2, 2*i)
存储 / 使用 每个变量
返回值 对应一个参数
函数名后面加上圆括号表示对函数的调用。
进阶使用：
*sequence
**dict

格式化

格式化指令 待格式化的值 格式化
"modele{ } { } { }".format(x, y, r) → str
"{selection:formatting!conversion}"
Selection:
2
nom
0.nom
4[key]
0[2]
格式化：
fill char alignment sign mini width . precision-maxwidth type
< > ^ = + - space 0 前置不足用 0 填充
整数：b binary, c char, d decimal (default), o octal, x 或 X hexa...
浮点数：e 或 E 指数, f 或 F 固定小数点, g 或 G 适当调整 (默认)
字符串：s... % 百分号
惯例：s (可读文本) 或 r (原样输出)

好习惯：绝不修改循环变量