

3python基本的数据类型

2019年2月28日 9:28

代码是现实世界事物在计算机世界的映射

写代码是将现实世界的事物用计算机语言描述出来

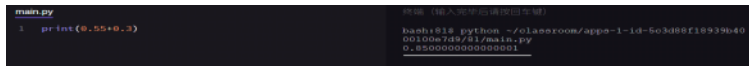
1. python的基本数据类型

a. Number

i. Int: 整数

ii. float: 浮点数

- 1) 两个整数相除默认为浮点型
- 2) 整形: $2//1$ (这里做的是整除运算)
- 3) 浮点数的加减:



```
main.py 1 pr int(0.55+0.3)
bash:1: python -f/olassroom/app-1-1d-5c3d8f19939b40: 0.8500000000000001
```

因为浮点数计算时, 会先转成二进制相加, 再转回来, 所以会多一个小尾巴

iii. 进制

- 1) 二进制: 0b
- 2) 八进制: 0o
- 3) 十六进制: 0f
- 4) 十进制不需要特殊符号
- 5) 其它进制→2: bin()
- 6) 其它进制→10: int()
- 7) 其它进制→16: hex()
- 8) 其它进制→8: oct()
- 9) 其它进制→bool: bool()

iv. bool布尔类型

- 1) 真: True
- 2) 假: False
- 3) bool(a)为假, 当a为0, 为'', 为[], 为{}, 为None

v. complex复数

- 1) 表示: 数 + j

b. 组:


i. 序列: str字符串

i. 字符串里面还有引号

- 1) 'Let's go'
- 2) 'Let\'s go'

ii. 多行字符串:

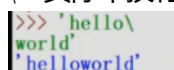
- 1) 三引号



```
>>> """
hello world
hello world
"""
```

- 2) \n: 实际也换行了

- 3) \: 实际不换行

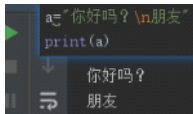


```
>>> 'hello\
world'
'helloworld'
```

iii. 转义字符

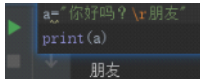
- 1) 作用:
 - a) 表示无法看见的字符
 - b) 与语法有冲突的字符
- 2) \单引号
- 3) \t横向制表符
- 4) \n换行: 光标到下一行行首

```
a="你好吗?\n朋友"
print(a)
```



- 5) \r回车: 光标在本行行首

```
a="你好吗?\r朋友"
print(a)
```



- 6) 作业:
 - a) 题目:

```
hello \n world
```
 - b) 答案:

```
>>> print('hello \n world')
```
- 7) 不输出为转义字符: r或R
 - a)

```
>>> print(r'c:\northwind\northwest')
```

iv. 字符串的运算

- 1) 拼接: +
- 2) 重复: *数字
- 3) 获取单个字符:
 - a) [某字符的位置]从0开始
 - b) [负数]: 从未位倒数的字符, -1是最后一位

```
>>> "hello world"[6]
'w'
>>> "hello world"[-5]
'w'
```

- 4) 一组字符:
 - a) [开始字符: 想截取的字符终点的下一位]

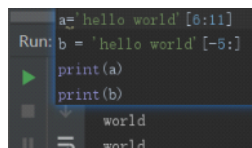
```
>>> "hello world"[0:4]
'hell'
>>> "hello world"[0:5]
'hello'
>>> "hello world"[0:-1]
'hello worl'
```

- c) 对位置的理解

i) 可以理解为光标的位置, 以'hello world'为例, 0在h的坐标, 如果是4在l和o中间, 所以截取的是'hell', 同理 - 1可以从d的右边移到了l和d中间, 所以选取的是'hello worl'

- 5) 作业: 用两种不同方式截取'hello world'中的'world'

```
a="hello world"[6:11]
Run: b="hello world"[-5:]
print(a)
print(b)
```



- b) 什么都不用输入, 表示截取到末尾

ii. 序列: 列表list

- i. []
- ii. 特点: 其元素不一定是一种类型

```
>>> type(["hello", "world", 1, 9])
<class 'list'>
>>> ["hello", "world", 1, 9, True, False]
['hello', 'world', 1, 9, True, False]
>>> type([[1, 2], [3, 4], [True, False]])
<class 'list'>
```

iii. 基本操作：（与字符串操作类似）

- 1) 访问其中元素：[位置]
- 2) 注意，选中一个是显示元素，选中多个是列表类型

```
>>> ["新月打击", "苍白之瀑", "月之降临", "月神冲刺"][3]
'月神冲刺'
>>> ["新月打击", "苍白之瀑", "月之降临", "月神冲刺"][0:2]
['新月打击', '苍白之瀑']
```

- 3) 加法与数乘与字符串同

iii. 序列：元组tuple

- i. ()
- ii. 特点：其元素不一定是同种数据类型
- iii. 操作：与列表相同
- iv. 注意：一个元素的元组，括号会被误认为是要做运算
- v. 定义只有一个元素的元组：

1)

```
>>> type((1,))
<class 'tuple'>
```

- 2) 定义空元素的元组：

```
>>> type(())
<class 'tuple'>
```

iv. 序列的共同点：

- i. 每个元素都会分配有一个序号
- ii. 切片：
 - 1) 选取其中的某一段（同上）
 - 2) 以某种规律重复选取

```
>>> "hello world"[0:8:2]
'hlow'
```

iii. 元素是否在某序列中：in /not in

```
>>> 10 in [1, 2, 3, 4, 5, 6]
False
>>> 3 not in [1, 2, 3, 4, 5, 6]
False
```

iv. 判断元素个数：len

```
>>> len([1, 2, 3, 4, 5, 6])
6
>>> len("hello world")
11
```

v. 最大最小：max, min

```
>>> max('hello world')
'w'
```

- 1) 字符编码：ord

字符串每个对应一个ascii码

```
>>> ord('w')
119
```

v. 集合：{}

i. 特点：

- 1) 集合无序
- 2) 不重复

ii. 常见操作：

- 1) 元素个数：len
- 2) 是否包含某元素：in
- 3) 从一个集合剔除另一个集合的元素：-（差集）

```
>>> {1, 2, 3, 4, 5, 6} - {3, 4}
{1, 2, 5, 6}
```

- 4) 两个集合的交集：&

```
>>> {1, 2, 3, 4, 5, 6} & {3, 4}
{3, 4}
```

- 5) 两个集合的并集：|

```
>>> {1, 2, 3, 4, 5, 6} | {3, 4, 7}
{1, 2, 3, 4, 5, 6, 7}
```

iii. 定义空集: set ()

注意: {}的数据类型是dict

```
>>> type(set())  
<class 'set'>
```

vi. 字典: dict

i. 字典是集合类型, 可以有很多key和value

ii. 定义:

```
{key1:value1, key2:value2...}
```

iii. 通过key 访问value

```
>>> {'Q': '新月打击', 'W': '苍白之瀑', 'E': '月之降临', 'R': '月神冲刺'}['Q']  
'新月打击'
```

iv. 不能有相同的key

v. value:

可以取任意一种数据类型

vi. key:

必须是不可变的数据类型: int、str、元组 (列表可变所以不可以是key)

vii. 空的字典: {}

5变量与运算符

2019年3月1日 10:45

1. 变量

- a. 赋值符号：=
- b. 命名
 - i. 可读性要强
 - ii. 只能使用字母、数字、下划线
 - iii. 首字母不能是数字
 - iv. 保留关键字不能用在变量名
 - v. 区分大小写
 - vi. 不需要定义变量类型→python是动态语言
- c. 分类：值类型与引用类型
 - i. 值类型变量
 - 1) 不可变
 - 2) 当改变变量值时，生成一个新的指针
 - ii. 引用类型
 - 1) 可变
 - 2) 当改变变量值时，直接在原变量上改变

iii.

```
>>> a = 1
>>> b = a
>>> a = 3
>>> print(b)
1
>>> a = [1, 2, 3, 4, 5]
>>> b = a
>>> a[0] = '1'
>>> print(a)
['1', 2, 3, 4, 5]
>>> print(b)
['1', 2, 3, 4, 5]
>>>
```

- iv. 不可变/可变：对应地址上的值能不能修改
？值是不是唯一的，等以后深入了解python内存地址

2. 运算符

- a. 算术运算符
 - i. + - * /
 - ii. //整除
 - iii. %取余
 - iv. **相当于^指数运算
- b. 赋值运算符
 - i. 为了给变量赋值，先做运算再赋值
 - ii. =
 - iii. += -= *= /=
 - iv. //=
 - v. %=

补充知识点

1. 对象三大特征

- a. 值
- b. 身份
- c. 特征
 - i. isinstance(变量名, 类型)
 - ii. isinstance(变量名, (各种类型))
 - iii. 不要用type

vi. **=

c. 关系运算符

i. 结果是布尔值

ii. ==

iii. !=

iv. >

1) 字符串也可以比较大小，比较的是ASCII码

```
>>> 'a' > 'b'
False
>>> ord('a')
97
>>> ord('b')
98
>>> 'abc' < 'abd'
True
>>> ord('abc')
Traceback (most recent call last):
  File "<pysHELL#6>", line 1, in <module>
    ord('abc')
TypeError: ord() expected a character, but string of length 3 found
```

2) 列表可以比较大小

3) 元组也可以比较大小

v. >=

d. 逻辑运算符

e. 成员运算符

i. 种类

1) in

2) not in

ii. 返回：布尔类型

iii. 适用范围

1) 列表，元组，字符串

2) 字典：判断的是key值而不是value值

f. 身份运算符

i. 种类：

1) is

2) is not

ii. 返回：布尔类型

iii. is和==的区别

1) ==表示值是否相等

2) is表示内存地址是否相等

3) 作业：

```

>>> a = {1, 2, 3}
>>> b = {2, 1, 3}
>>> a == b 比较 a is b 比较
SyntaxError: invalid syntax
>>> c = (1, 2, 3)
>>> d = (2, 1, 3)
>>> a == b 比较 c == d 比较 c is d
True
>>> a is b
False
>>> id(a)
59656592
>>> id(b)
59656472
>>> c == d
False
>>> c is d
False
>>>

```

g. 位运算符

- i. 特定：非二进制数转化为二进制数再进行每位运算
- ii. &按位与：——对应每位进行and
- iii. |按位或：——对应每位or
- iv. ^按位异或
- v. <<左移动
- vi. >>右移动

6 表达式

2019年3月2日 11:06

1. 表达式:

a. 定义: 表达式是运算符和操作数所构成的序列

b. 优先级:

i. 序号越小优先级越高

序号	运算符	描述
1	**	指数(次幂)运算
2	~, +, -	补码, 一元加减(最后两个的方法名称是 +@ 和 -@)
3	*, /, %, //	乘法, 除法, 模数和地板除
4	+, -	
5	>>, <<	向右和向左位移
6	&	按位与
7	^,	按位异或和常规的 "OR"
8	<=, <, >, >=	比较运算符
9	<>, ==, !=	等于运算符 @5880511
10	=, %=, /=, //=, -=, +=, *=, **=	赋值运算符
11	is, is not	身份运算符
12	in, not in	成员运算符
13	not, or, and	逻辑运算符

ii.

iii. 同级从左到右

iv. 通常是左结合, 但含赋值符号要先右结合

v. 作业:用括号标注出运算顺序

1)

```
>>> not a or b + 2 == c
False
```

2)

```
>>> ((not a) or ((b + 2) == c))
False
```

vi. 总的顺序: 算术运算符>比较运算符>逻辑运算符

6 流程控制语句

2019年3月2日 11:28

1. 条件语句

a. 定义:

if 判断语句:

操作语句

else:

操作

if 判断语句:

pass

elif 判断语句:

pass

else:

pass

2. 循环

a. while 循环

while 判断语句:

pass

else:

pass

无限死循环: 用ctrl+c退出

场景: 递归

b. for 循环

场景: 遍历、循环 序列或者集合、字典

for target_list in expression_list:

pass

else:

pass

强行跳出循环, break: 只跳出当前的循环

跳出当次循环, continue

执行n次循环

```
for x in range(0,10):  
    pass
```

注：range (start,stop,step)

start包含 stop不包含 step间隔

```
for x in range(0,10,2):  
    pass
```

作业：打印a中的间隔元素

```
a = [1,2,3,4,5,6,7,8]  
  
for i in range(0, len(a), 2):  
    print(a[i], end=' | ')
```

编程规范

2019年3月2日 14:14

1.

- a. 不强制要求;
- b. 不需要{}把一段代码包裹起来, 用缩进 (4个空格)
- c. 对于形式上的常量: 所有字母大写
- d. 单行注释#
- e. 多行注释""" """
- f. 每个模块前应该有一段说明用""" """给出
- g. 变量应该丢到函数、类里面封装起来, 不推荐在模块里面直接放变量
- h. 冒号前不需要空格
- i. 程序最后应该有一个空行
- j. 运算符左右两边各加个空格
- k. 命名规则:
 - a. 变量都小写, 单词组合用_
- l. pass: 空语句/站位语句, 使其不会报错
- m. 一个代码块同级别的代码同时执行

n.

```
#代码块
if condition:
    code1
    code11
    code22
    code333
    code444
    code5555
    code6666
code2
code3
else:
    code1
    code2
    code3
```

- o. 不要多重嵌套, 用函数减少嵌套
- p. 解决代码换行:
 - \
 - 加括号()

7 包、模块、函数、变量作用域

2019年3月2日 17:11

1. 包和模块

a. 层级划分

i. 包

1) 模块(.py文件)

a) 类

i) 函数、变量是类的一个特性，不属于类本身

b. 命名空间：区分相同模块

i. 包.模块

c. 包的下面可以包含子包，子包可以和类同级

d. 包必要条件

i. `_init_.py`

ii. `_init_.py`的名字=包的名字

e. 导入模块：

i. `import 模块的命名空间 as 简化名`

ii. `from 模块 import 变量(*:所有变量, 不推荐)`

iii. `from 模块 import 变量,变量,变量...`

iv. 导入模块时就会执行模块代码

? v. 模块的内置属性

f. `_init_.py`

i. 只要导入包的某部分，自动首先执行

ii. 应用：

1) 做包和模块的初始化

2) 用于后面需要重复批量导入的库

iii. `_all_ =`:用于规定模块需要导入的变量

g. 包和模块是不会重复导入，需要避免循环导入

8函数

2019年3月3日 9:16

1. 常用函数

1. print
2. round(变量, n)四舍五入保留小数点后n位

2. 函数

1. 功能性
2. 隐藏细节
3. 避免编写重复的代码
4. 组织代码

3. 查看函数

1. 在命令行输入 help(函数)

4. 定义函数

```
def funcname(parameter_list):  
    pass
```

1. 参数列表可以没有,按顺序一一对应
2. return value 没有的话默认为None
3. return 后的语句不会被执行

5. 序列解包

1. a,b,c = 1,2,3 一一对应
2. d = 1,2,3 type(d)是tuple
3. a,b,c = d

6. 参数

1. 必须参数

- a. 函数列表里定义的参数是必须要调用的, 不然报错
- b. 形式参数
- c. 实际参数

2. 关键字参数

- a. 任意制定参数的顺序

- b. `c = add(y=3, x=2)`
 - c. 提高阅读代码的理解能力
- 3. 多参数：应该封装成一个类
- 4. 默认参数：
 - a. `def print_student_files(name, gender='M', age=18, college='人民路小学')`:
 - b. 如果需要改参数，按必须参数传
 - c. 如果不需要改参数，只传不需要参数的
 - d. n个容易错的坑
 - i. `def function(name, gender='M', teacher)` 错：默认参数要放在必须参数之后
 - ii. 多个默认参数只想修改其中几个，要么按顺序依次改，要么使用关键字参数改有缺省的参数
- 5. 函数下的局部变量不会覆盖同名的全局变量

9面向对象

2019年3月5日 11:56

1. 类

a. 概述

- a) 关键字: `class`
- b) 类的命名规则:
 - i. 首字母大写, 单词连接每个首字母大写而非_
- c) 定义:
- d) 类的内容
- e) 类和对象的区别:
- f) 作用:
- g) 易错点:

b. 变量

- a) 类变量和实例变量:

c. 方法

- a) 使用类: 实例化 (方法)
- b) 构造函数: `def __init__(self,形参):`
- c) 构造函数和实例方法的区别
- d) 类方法:
- e) 静态方法:

d. 成员可见性:

e. 三大特性

- i. 继承性
- ii. 封装性
- iii. 多态性

10 正则表达式

2019年3月6日 10:50

1. 正则表达式

a. 定义:

- i. 特殊的字符序列用于检测一个字符串是否与我们设定的字符串相匹配
- ii. 快速检索文本, 实现对文本的操作

b. 使用正则表达式

```
import re
```

c. 组成:

- 1) 普通字符: 常量
- 2) 元字符:

d. `re.findall('正则表达式', 字符串)`

i. 定义:

```
r = re.findall('正则表达式', 字符串, 模式参数)
```

ii. 模式参数有多个

iii. 模式参数用|连接

e. `re.sub('匹配字符串', 更正后内容, 原字符串, count, 模式)`

f. `re.match`

g. `re.search`

h. 针对`match`和`search`从返回对象中提取信息

i. 模式

1) 或:[]

a) `s`中寻找中间字符为`c`或`f`的单词

```
r = re.findall('a[cf]c', s)
```

b) `s`中寻找中间字符不是`c`也不是`f`的单词

```
r = re.findall('a[^cf]c', s)
```

c) `s`中寻找中间字符是`c-f`的单词

```
r = re.findall('a[c-f]c', s)
```


2) 概括字符集

- a) `\d`: 数字, 匹配单一字符
- b) `\D`: 非数字, 匹配单一字符
- c) `\w`: 数字或字母或`_`, 匹配单一字符
- d) `\W`: 非单词字符, 匹配单一字符
- e) `\s`: 空白字符, 匹配单一字符
- f) `\S`: 非空白字符, 匹配单一字符
- g) `.`: 匹配除换行符外所有字符

3) 数量词, 匹配多个字符

- a) `s`中寻找被数字间隔开的多组单词
`r = re.findall('[a-z]{3,6}')`
- b) 贪婪与非贪婪
 - i) 一般Python默认是贪婪
 - ii) 非贪婪的表示方法: `{ }?` : 取可以满足的下限
- c) `*`
对星号前的一个字符可以匹配0次1次或无限多次
- d) `+`
对加号前的一个字符可以匹配1次或无限多次
- e) `?`
对问号前的一个字符可以匹配0次1次, 多的次数会被去掉
常用来做去重
- f) 完全匹配: `^$`
 - i) `^`: 从字符串的开始匹配
 - ii) `^$`: 匹配到字符串的结尾

4) 组: `()`里面为且关系

- a) `'Python'`重复3次
`r = re.findall('(Python){3}')`

b) ps:一个正则表达式可以有多个组同时存在

11JSON

2019年3月6日 19:28

1. JSON

a. 定义

- i. JavaScript对象标记
- ii. 轻量级的数据交换格式
- iii. 字符串是JSON的表现形式
- iv. 符合JSON格式的字符串叫JSON字符串
- v.

```
{ "name" : "qiyue" }
```

b. JSON比XML的优势

- i. 易阅读
- ii. 易解系
- iii. 网络传输效率高
- iv. 适合做跨语言交换数据（XML也可以）

c. JSON和Python的转化

i. 反序列化（由json字符串到某种数据结构

```
import json
```

```
json_str = '{"name":"qiyue", "age":18}' : 对象
```

```
json_str = '[{"name":"qiyue","age":18},
```

```
 {"name":"qiyue","age":18}]' : 数组
```

```
json.loads(json_str)
```

注意json里面要求是双引号

对象读入之后变量的类型是dict

数组读入之后变量类型是list

ii. 对应

json	python
object	dict
array	list
string	str
number	int
number	float
true	True
false	False
null	None

iii. 序列化(Python数据结构到json字符串)

1) 方式:

```
json_str = json.dumps(对象)
```

iv. JSON可以看做是和JAVASCRIPT同级的语言，是一种中介的数据类型，有自己的数据类型（尽管和JAVASCRIPT数据类型有些相似）

12 Python高级语法和用法

2019年3月6日 20:57

1. 枚举

a. 作用：刻画类型

b. 方式

i. from enum import Enum

```
class VIP(Enum):
```

```
    YELLOW = 1
```

```
    GREEN = 2
```

```
    BLACK = 3
```

```
print(VIP.YELLOW) #VIP.YELLOW
```

```
print(VIP[YELLOW]) #VIP.YELLOW
```

```
print(VIP.YELLOW.value) # 1
```

```
print(VIP.YELLOW.name) # YELLOW
```

ii. 类要全部大写，重在标签而非它的值

iii. 用字典或者普通类存储类型的缺点：

1) 可变

2) 没有防止相同值重复

iv. 用Enum类的优点：

1) 更改类的赋值会报错

2) 重复标签会报错

? c. 枚举值枚举类型和枚举名称的比较

```

print(Color.GREEN.value) # 枚举值 取值 |
print(type(Color.GREEN.value))

print(Color.GREEN.name) # 枚举名称
print(type(Color.GREEN.name)) # 结果是字符串类型

print(Color.GREEN) # 枚举类型
print(type(Color.GREEN)) # 结果是枚举下的一个类型

i. print(Color['GREEN']) # 也能打印出枚举类型
   print(type(Color['GREEN']))

2
<class 'int'>
GREEN
<class 'str'>
Color.GREEN
<enum 'Color'>
Color.GREEN
<enum 'Color'>

```

d. 枚举类型的比较

i. 可以做的

- 1) 枚举类型和枚举类型的等值比较(==)
- 2) 枚举类型和枚举类型的身份比较(is)

ii. 不能做的

- 1) 枚举类型和值的等值比较
- 2) 枚举类型和枚举类型的大小比较
- 3) 不同枚举类型的比较

e. 注意事项

- i. 枚举类型标签名不能相同，数值可以相同

1)

```
3 class VIP(Enum):
4     YELLOW = 1
5     GREEN = 1
6     BLACK = 3
7     RED = 4
8
9 class Common():
10     YELLOW = 1
11
12
13 print(VIP.GREEN)
14 # VIP.YELLOW = 6
```

问题 输出 调试控制台 终端

```
PS D:\python\eleven> python c2.py
VIP.YELLOW
PS D:\python\eleven>
```

2) 其实是别名 GREEN实际上是YELLOW_ALIAS

ii. 遍历

1) for u in 枚举类: 别名会被省略

2) 为了显示别名:

```
for v in VIP.__members__:
    print(v)
```

f. 枚举转换

i. 在数据库存储类型用数字

ii. 在代码里用枚举类来存储类型

iii. 把数字转换为枚举类型

```
class Common():
    YELLOW = 1
print(VIP(1)) # VIP.YELLOW
```

g. 拓展

i. IntEnum

1) 强制每个枚举类型的值都为int类型, Enum都可以

ii. 使每个枚举类型的值都是唯一的

1)

```
2  from enum import IntEnum,unique
3
4  @unique
5  class VIP(IntEnum):
6      YELLOW = 1
7      GREEN = 1
8      BLACK = 3
9      RED = 4

PS D:\python\eleven> python c5.py
Traceback (most recent call last):
  File "c5.py", line 5, in <module>
    class VIP(IntEnum):
      File "D:\Server\Python3.6\lib\enum.py", line 834, in unique
        (enumeration, alias_details))
ValueError: duplicate values found in <enum 'VIP': GREEN -> YELLOW
PS D:\python\eleven>
```

2) 不能对其实例化

13 高级语法与用法

2019年3月7日 20:49

1. 函数式编程

a. 函数

i. 其它语言与python中函数的区别

- 1) 其它语言：函数不能实例化
- 2) Python:一切皆对象，可以实例化

ii. 函数

- 1) 可以赋值给变量
- 2) 可以当做另外一个函数的参数，传到另一个函数

b. 闭包

i. 现象：

ii. 定义：

- 1) 闭包 = 函数+环境变量

2) 形式：

```
a) def curve_pre():  
    a = 25  
    def curve(x):  
        return a*x^2  
    return curve
```

3) 闭包的环境变量查看：

```
a) f = curve_pre()  
    f.closure # 环境变量
```

iii. 闭包的存在条件：

iv. 闭包的使用：

- a) 旅行者 就算路径长度x=0 走一步 + 1走n步，再走k步，输出结果

```
origin = 0  
  
def factory(pos):  
    def go(step):  
        nonlocal pos
```

b)

```
origin = 0

def factory(pos):
    def go(step):
        nonlocal pos
        new_pos = pos + step
        pos = new_pos
        return new_pos
    return go

tourist = factory(origin)
print(tourist(2))
print(tourist(3))
```

c) 闭包可以记忆上次的执行环境

d) 闭包实现函数外部调用函数内部的变量

v. 关于闭包概念的理解：

<https://www.cnblogs.com/JohnABC/p/4076855.html>

vi. 闭包问题：内存泄漏

c. 匿名函数

i. 形式：

a) lambda 参数: 表达式

ii. 三元表达式

a) 形式：

i) 条件为真的结果 if 条件判断 else 条件为假的结果

b) 比较xy大小返回较大的一个

i) result = x if x > y else y

iii. map类

a) 形式

i) map(function, 参数)

b) 本质：通过函数将参数列表映射到结果列表

c) map + lambda

i) 将列表里的每个数平方后返回原列表

ii) result = map(lambda x: x*x, list_x)

iii) 注map可以传入多个列表参数

result = map(lambda x,y: x+y, list_x,list_y)

iv) map计算结果的个数取决于多个list中个数最

少的一个 (即可以满射的部分)

iv. reduce (函数)

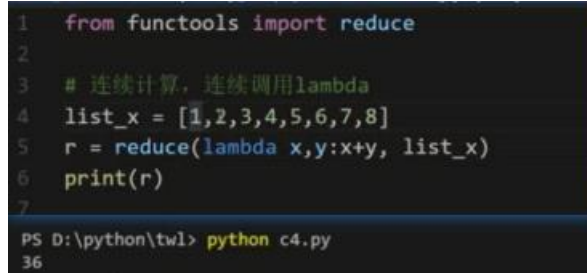
a) 形式

i) from functools import reduce

reduce(lambda x,y: x+y, list_x, 参数1)

b) 原理:

i) 连续调用lambda



```
1 from functools import reduce
2
3 # 连续计算, 连续调用lambda
4 list_x = [1,2,3,4,5,6,7,8]
5 r = reduce(lambda x,y:x+y, list_x)
6 print(r)
7
PS D:\python\twl> python c4.py
36
```

ii)

(((1+2)+3)+4)+5

iii)

iv) lambda 是做连续运算, 不是连续相加

v) 参数1, 初始值, 没有默认为0

v. filter:

a) 形式

i) r = filter(lambda x: True if x==1 else False, list_x)

ii) 返回的类型必须代表真假

iii) 所以上可以写成:

r = filter(lambda x:x, list_x)

2. 装饰器

a. 形式:

i. 不带参数

```
def decorator(func):
    def wrapper():
        pass(新增业务逻辑)
    return wrapper
```

@decorator

```
def f1():
    print('AAA')
```

f1()

ii. 带参数:

```
def decorator(func):
```

```
    def wrapper(*args, **kw) # 支持可变参数, 支持
```

```
    多关键字
```

```
        pass
```

```
        func(*args, **kw)
```

```
    return wrapper
```

b. 作用:

i. 保持原有调用模式

c. 一个函数可以加多个装饰器