起步

编码

> python 2默认ASCII
>
> python3默认Unicode（UTF-8）

看默认编码

```
import sys
print(sys.getdefaultencoding())
```

`# -*- coding: utf-8 -*-` 在python脚本前加coding

输入

```
user_name = input("请输入你的姓名")#python3
user_name = raw_input("请输入你的姓名") #python2
```

输出

```
print(message)#python3
print message #python2
```

注释

```
#这是一个注释

'''
这是多行注释
'''
```

条件语句

```
#例一：找小姐姐
sex = input("请输入性别")
if sex=='女':
    print("来呀！！")
elif sex=='男':
    print("再见!!")
else:
    print("滚")

#例二：登入失败
```

```
selx = input("输入用户名")
password = input("输入密码")
if selx == 'selx' and password == "654321":
    print("登入成功")
else:
    print("登入失败")
```

循环语句

```
count = 1
while True:
    print(count)
    count = count+1
    if count>80:
        break
"""
else:
    while不同时进行语言
"""
```

字符串格式化

```
a ='我'
a = '%s是2'%(a)
print(a)
"""
化简
a = '我'
b = '%s是2'
c = b%a
"""
```

> 要用‘%’时使用 `%%`

四则运算

符号	名称	实例
%	取余	5%2=1
//	整除	5//2=2
**	幂	5**2=25
/	除	5/2=2.5
+	加	5+2=7
-	减	5-2=3

字符转换

string --》int

```
number = int("55")
```

int --> string

```
string = str(233)
```

string --> bool

```
boo = bool(v1)
```

int --> bool

```
boo = bool(4)
```

变量

字符串数组

切片

```
a = 'ABCDEFGHIJK'
print(a[0:3])  # print(a[:3]) 从开头开始取0可以默认不写
print(a[2:5])
print(a[:]) #默认到最后
print(a[:-1]) # -1 是列表中最后一个元素的索引，但是要满足顾头不顾腚的原则，所以取不到K元素
print(a[:5:2]) #加步长
print(a[-1:-5:-2]) #反向加步长
```

方法	作用
uppar	全部大写
append(x)	追加字符x

元组

字典

不可变（可哈希）的数据类型：int，str，bool，tuple。

可变（不可哈希）的数据类型：list，dict，set。

字典是Python语言中的映射类型，他是以{}括起来，里面的内容是以键值对的形式储存的：

Key: 不可变（可哈希）的数据类型.并且键是唯一的，不重复的。

Value:任意数据(int，str，bool，tuple，list，dict，set)，包括后面要学的实例对象等。

在Python3.5版本（包括此版本）之前，字典是无序的。

在Python3.6版本之后，字典会按照初建字典时的顺序排列(即第一次插入数据的顺序排序)。

- **创建方式**

```
# 创建字典的几种方式：
```

```python
# 方式1:
dic = dict((('one', 1),('two', 2),('three', 3)))
# dic = dict([('one', 1),('two', 2),('three', 3)])
print(dic)  # {'one': 1, 'two': 2, 'three': 3}


# 方式2:
dic = dict(one=1,two=2,three=3)
print(dic)  # {'one': 1, 'two': 2, 'three': 3}


# 方式3:
dic = dict({'one': 1, 'two': 2, 'three': 3})
print(dic)  # {'one': 1, 'two': 2, 'three': 3}

# 方式5: 后面会讲到先了解
dic = dict(zip(['one', 'two', 'three'],[1, 2, 3]))
print(dic)

# 方式6: 字典推导式 后面会讲到
# dic = { k: v for k,v in [('one', 1),('two', 2),('three', 3)]}
# print(dic)

# 方式7:利用fromkey后面会讲到。
# dic = dict.fromkeys('abcd','太白')
# print(dic)  # {'a': '太白', 'b': '太白', 'c': '太白', 'd': '太白'}
```

```python
#创建方式
#方式一
# dic = dict((('one',1),('two',2),('three',3)))
# print(dic)

#方式二
dic = dict(one=1,two=2,three=3 )
print(dic)

#方式三
dic = dict({'one':1,'two':2,'three':3})
print(dic)


# 添加
dic['hh'] = 23
dic.setdefault('sdf')
```

```
print(dic)


# 查
ll = dic.get('hh')
ll = dic.get('hh1','没有此件')
print(ll)

for key in dic:
    print(key)

#values
print(dic.values())
print(list(dic.values()))
for values in dic:
    print(values)

#items()
print(dic.items())#dict_items([('one', 1), ('two', 2), ('three', 3),
('hh', 23), ('sdf', None)])
for i in dic.items():
    print(i)#('three', 3)
```

字符串

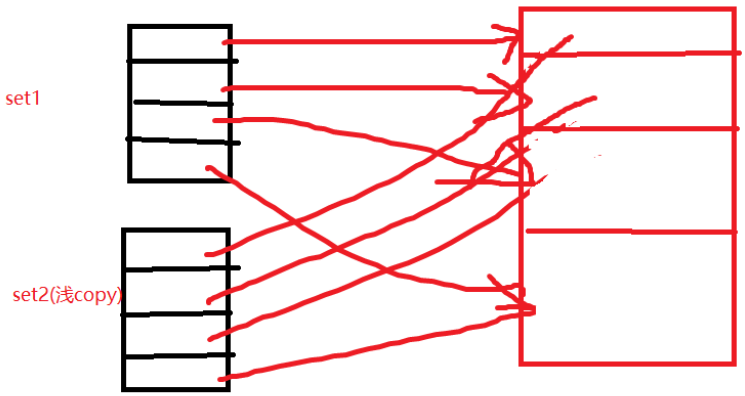
地址

```
#is 变量的id
l1 = [1,2,3]
l2 = [1,2,3]
print(id(l1)) #l1的地址
print(id(l2)) #l2的地址
print(l1 is l2)#False
```

集合

```python
#集合的创建
# set1 = set({1,3,"5dfsg","dsaf"})
# print(set1)


# #pop随即删除
# set1.pop()
# print(set1)

#变相改值
# set1.remove(1)
# set1.add('男神')
# print(set1)

set1 = {1,2,3,4,5}
set2 = {4,5,6,7,8}


#交集
print(set1 & set2)
#并集
print(set1 | set2)
#差集
print(set1 - set2)
#反交集
print(set1 ^ set2)


#子集
set1 = {1,2,3}
set2 = {1,2,3,4,5,67}
print(set1 > set2)

#列表的去重
l1 = [1,'太白',1,2,2,'太白',2,2,6,6,6,3,'太白']
set1 = set(l1)
l1 = list(set1)
print(l1)
```

浅拷贝和深拷贝



> 1、浅拷贝：对基本数据类型进行值传递，对引用数据类型进行引用传递般的拷贝，此为浅拷贝。

```
#浅copy
l1 = [1,2,3,4,5,[22,33]]
l2 = l1.copy()
l1.append(666)
print(l1,id(l1))
print(l2,id(l2))

l1 = [1,2,3,4,5,[22,33]]
l2 = l1.copy()
l1[-1].append(666)
l1.remove(2)
print(l1,id(l1))
print(l2,id(l2))
```

> 2、深拷贝：对基本数据类型进行值传递，对引用数据类型，创建一个新的对象，并复制其内容，此为深拷贝

```
#深copy
import copy
l1 = [1,2,3,4,5,[22,33]]
l1 = [1,2,3,4,5,[22,33]]
l2 = copy.deepcopy(l1)
l1[-1].append(666)
l1.remove(2)
print(l1,id(l1))
print(l2,id(l2))
```

编码

```
s1 = "中国"
b = s1.encode('utf-8')#uncode->utf-8
print(b,type(b)) #b'\xe4\xb8\xad\xe5\x9b\xbd'

b1 = b'\xe4\xb8\xad\xe5\x9b\xbd'
s2 = b1.decode('utf-8')
print(s2)
```

- gbk-->utf-8

```
b1 = b'\xd6\xd0\xb9\xfa'
s = b1.decode('gbk')
# print(s)
b2 = s.encode('utf-8')
print(b2)
```

文件

格式化输入输出

- 一般格式:

```
#%s format
name = '太白'
age = 18
msg = '我叫%s,今年%s'%(name,age)
msg1 = '我叫{},今年{}'.format(name,age)
```

- 新特性

```
name = '太白'
age = 18
msg = f'我叫{name}，今年{age}' #F/f ''
print(msg)

#可以用表达式
dic = {'name':'alex','age':73}
msg1 = f'我叫{dic["name"]}，今年{dic["age"]}'
print(msg1)

#结合运算
count = 7
print(f'最终结果：{count**2}')

#结合函数写
def _sum(a,b):
    return a+b
msg = f'最终结果：{_sum(3,4)}'
print(msg)
```

函数

- 结构

```
def 函数名():
    函数体
```

- 万能函数

```
#*函数定义时，*代表聚合他将所有的位置参数聚成一个元组，赋值给args
def eat(*args):
    print('我请你吃：%s,%s,%s,%s,%s'%args)

eat('蒸羔羊','蒸熊掌','正路一','烧化鸡','烧子牙')


#**函数定义时，**代表聚合他将所有的位置参数聚成一个键值对，赋值给kwargs
def func(**kwargs):
    print(kwargs)

func(name="sdlkjaf",age=10)
```

- **kwargs的位置？仅限关键字参数

  > 默认参数，*函数，关键字参数，标记参数，**函数

  ```
  def func(a,b,*args,sex='男',c,**kwargs,):
      print(a,b)
      print(sex)
      print(c)
      print(args)

  func(1,2,3,4,5,6,7,sex='女',c="sdf",name="的撒佛")
  ```

- *在函数的调用中代表打散

  ```
  def func(*args):
      print(args)

  func(*[1,2,3],*[22,33])
  ```

- 名称空间
  - 分三种：
    - 内置名称空间（）
    - 全局空间（当前py文件）
    - 局部空间（函数内部）

- 加载顺序
- 内置名称空间--->全局名称空间--->局部名称空间

- 内置函数

```
#glbals() locals()
a = 1
b = 2
def func():
    name="alex"
    age = 73
    print(globals())#返回的是字典：字典里的键值对：全局作用域的所有内容
    print(locals())#返回的是字典：字典里的键值对：局部作用域的所有内容
print(globals())#返回的是字典：字典里的键值对：全局作用域的所有内容
print(locals())#返回的是字典：字典里的键值对：局部作用域的所有内容
func()
```

- 陷阱

```
#只针对于默认参数是可变的数据类型
def func(name,alist=[] ):
    alist.append(name)
    return alist

print(func("alex1"))#['alex1']
print(func("alex3"))#['alex1', 'alex3']
#无论调用多少次，默认参数都是同一个
```

例2:

```
def func(name,alist=[] ):
    alist.append(name)
    return alist
print(func(10)) #[10]
print(func(20,[])) #[20]
print(func(100)) #[10, 100]
```

- 函数的命名

1. 函数指向的是函数的内存地址
   函数名+()就可以执行函数
2. 函数名就是变量

迭代器

- 可迭代的对象

  可以更新的值

  含有 `__iter__` 方法的值

- 判断一个对象是否是可迭代对象

```
s1 = 'fdsjakha'
# print(dir(s1))
print('__iter__' in dir(s1))
```

- 优点
  1. 存储的数据直接能显示，比较直观
  2. 拥有的方法比较多
- 缺点
  1. 占用内存
  2. 不能通过for循环

生成器

> return :函数中只存在一个return结束函数，并且给函数的执行者返回值
>
> yield：只要函数中有yield那么他就是生成器函数而不是函数了
>
> 生成器函数中可以有多个yield，yield不会结束生成器函数，一个yield对应一个next

```
def func():
    print(111)
    print(222)
    yield 3#第一次执行结束
    a = 1
    b = 2
    c=a+b
    print(c)
    yield 4#第二次执行
ret = func()
# print(ret)
print(next(ret))
print(next(ret))
```

- 应用

  假如店家生产了500-个包子，我只要200个包子，不需要都显示

  即：要多少给多少

```
def gen_func():
    l1 = []
    for i in range(1,5000):
        yield (f'{i}号包子')
ret = gen_func()
for i in range(1,200):
    print(next(ret))
```

-

列表推导式

```
# 循环模式
# [变量(加工后的变量) for 变量 in iterable]
print([i**2 for i in range(1,11)])

print([i for i in range(1,101,2)])
print([f"python第{i}期" for i in range(1,10)])

#筛选模式
#30以内能被3整除的数
l1 = [i for i in range(1,31) if i%3==0]
print(l1)

#查找长度小于3的
l1 = ['barry','ab','alex','wusir','xo']
print([i for i in l1 if len(i)>=3])
```

内置函数

黄色一带而过:

all() any() bytes() callable() chr() complex() divmod() eval() exec() format() frozenset() globals() hash() help() id() input() int() iter() locals() next() oct() ord() pow() repr() round()

红色重点讲解:

abs() enumerate() filter() map() max() min() open() range() print() len() list() dict() str() float() reversed() set() sorted() sum() tuple() type() zip() dir()

蓝色未来会讲:

classmethod() delattr() getattr() hasattr() issubclass() isinstance() object() property() setattr() staticmethod() super()

- `eval` 剥去字符串的外衣服运算里面的代码

```
s1 = '1+3'
print(eval(s1))

s = '{"name":"alex"}'
print(s,type(s1))
# print(dict(s))
print(eval(s),type(eval(s)))
#需要网络传输的str input输入时，sql注入绝对不用eval
```

- exec 与eval几乎一样
  `exec` :执行字符串类型的代码。

```
msg = """
for i in range(10):
    print(i)
"""
# print(msg)
# exec(msg)
```

- `hash`：获取一个对象（可哈希对象：int，str，Bool，tuple）的哈希值。

```
print(hash("1"))
```

- `callable` 判断是否可调用

```
s1 = "fdsaf"
def func():
    pass
print(callable(s1)) #false
print(callable(func)) #True
```

- `int` 函数用于将一个字符串或数字转换为整型。

```
print(int())  # 0
print(int('12'))  # 12
print(int(3.6))  # 3
print(int('0100',base=2))  # 将2进制的 0100 转化成十进制。结果为 4
```

- `bin`：将十进制转换成二进制并返回。

  `oct`：将十进制转化成八进制字符串并返回。

  `hex`：将十进制转化成十六进制字符串并返回。

- `divmod`：计算除数与被除数的结果，返回一个包含商和余数的元组(a // b, a % b)。

  `round`：保留浮点数的小数位数，默认保留整数。

  `pow`：求xy**次幂。（三个参数为**xy的结果对z取余）

- `ord` :输入字符找该字符编码的位置

  `chr` :输入位置数字找出其对应的字符

- repr：返回一个对象的Sting形式

```
s1 = '村龙'
#print(s1)
print(repr(s1) ) #'村龙'
msg = '我叫%r'%s1 #我叫'村龙'
print(msg)
```

- list : 创建列表

  dict : 创建字典

- reversed 返回是一个反转的迭代器

```
l1 = [i for i in range(10)]
print(l1)
obj = reversed(l1)
print(obj)
print(list(obj))
```

- sum：求和

```
l1 = [i for i in range(10)]
print(sum(l1)) #45
print(sum(l1,100)) #145
```

- zip：拉链方法

```
l1 = [1,2,3,4]
tul = ('太白','b哥','德刚')
s1 = 'abcd'
obj = zip(l1,tul,s1)
print(obj)
for i in obj:
    print(i) #(1, '太白', 'a')
```

- min：求最小值

```
l1 = [33,2,1,54,7,-1,-9]
l2 = []

ret = min(l1,key=abs)  # 按照绝对值的大小，返回此序列最小值
print(ret)
# 加key是可以加函数名，min自动会获取传入函数中的参数的每个元素，然后通过你设定的返回值比较大小，返回最小的传入的那个参数。
print(min(l1,key=lambda x:abs(x)))  # 可以设置很多参数比较大小
dic = {'a':3,'b':2,'c':1}
print(min(dic,key=lambda x:dic[x]))

# x为dic的key，lambda的返回值（即dic的值进行比较）返回最小的值对应的键
```

闭包

- 概念：
  1. 闭包只能存在嵌套函数中
  2. 内层函数对外层函数非全局变量的引用，就会形成闭包
     被引用的非全局变量也称作自由变量，这个自由变量会与内层函数产生一个绑定关系
     自由变量不会在内存中消失
     闭包的作用：保证数据的安全

```
def make_averager():
    series = []
    def averager(new_value):
        series.append(new_value)
        total = sum(series)
        return total/len(series)
    return averager
avg = make_averager()
# 函数名.__code__.co_freevars 查看函数的自由变量
print(avg.__code__.co_freevars)  # ('series',)
```

当然还有一些参数，仅供了解：

```
# 函数名.__code__.co_freevars 查看函数的自由变量
print(avg.__code__.co_freevars)  # ('series',)
# 函数名.__code__.co_varnames 查看函数的局部变量
print(avg.__code__.co_varnames)  # ('new_value', 'total')
# 函数名.__closure__ 获取具体的自由变量对象，也就是cell对象。
# (<cell at 0x0000020070CB7618: int object at 0x000000005CA08090>,)
# cell_contents 自由变量具体的值
print(avg.__closure__[0].cell_contents)  # []
```

装饰器

- 原则
  装饰器：装饰，装修
  开放封闭原则：
  开放：对代码的拓展开放，更新地图，加新枪等等
  封闭：对代码的修改是封闭的。闪躲用q。就是一个功能，一个函数。

  装饰器完全遵循开放封闭原则
  装饰器：在不改变原函数的代码以及调用方式的前提下，为其增加新功能

- 语法糖

```
def wrapper(f):
    def inner(*args,**kwargs):
        '''添加额外的功能：执行之前的操作'''
        ret = f(*args,**kwargs)
        '''添加额外的功能：执行之后的操作'''
        return ret
    return inner

@wrapper
def dairy(name,age):
    time.sleep(0.5)
    print(f"欢迎{age}岁{name}登入日记")
dairy('梨树qi',18)

@wrapper
def index():
    time.sleep(3)
    print("欢迎登入博客园")
    return 66
index()
```

模块

1. 运行方式：
   - 脚本方式

- 模块方式

`__name__` 的使用方式

1. 在脚本运行时
2. 在

导入方法