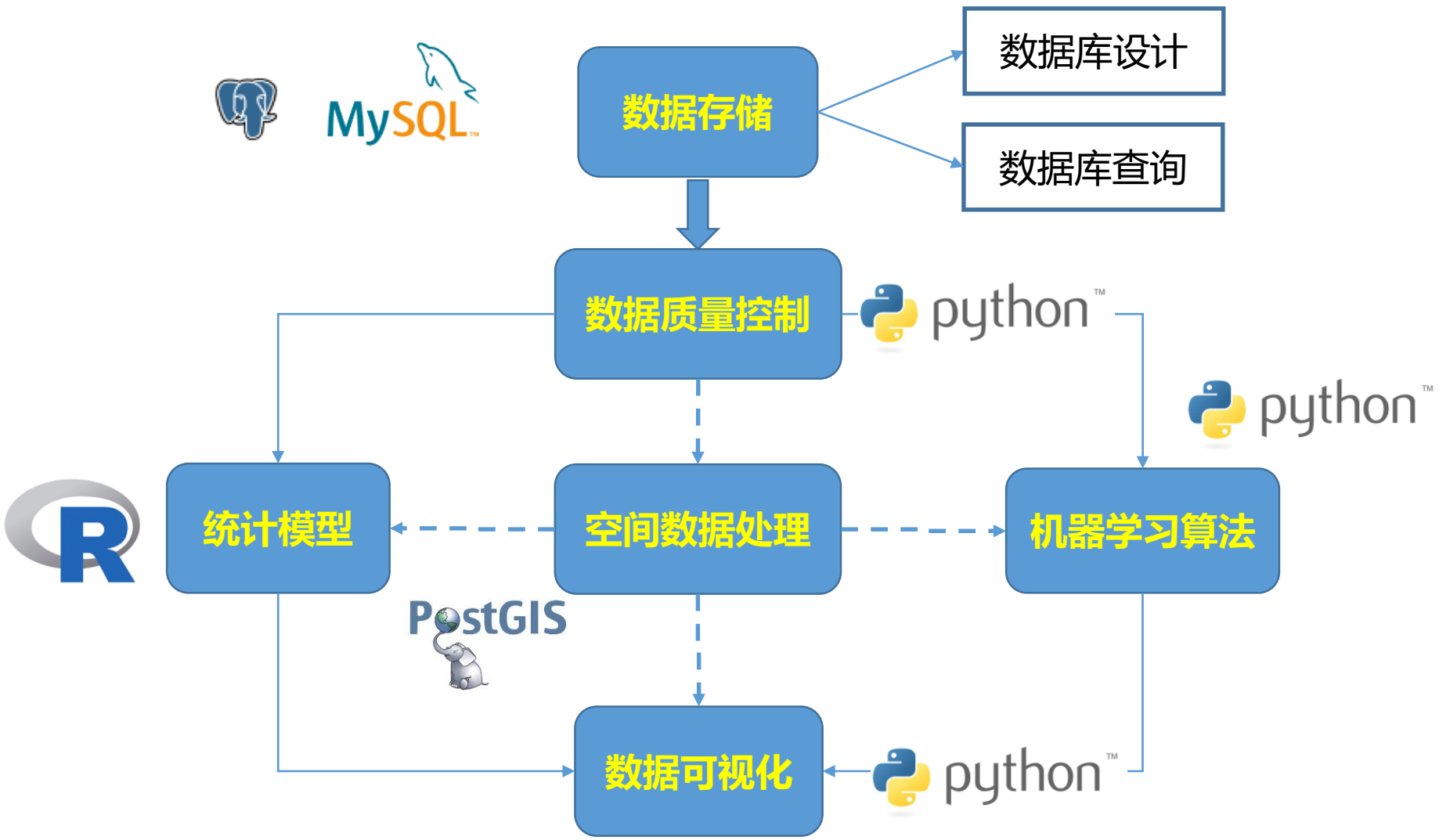
《交通大数据技术》

马晓磊、崔志勇
交通科学与工程学院
2024年4月

Python语言基础

交通大数据处理流程


数据存储
数据库设计
数据库查询
数据质量控制
python
python
统计模型
空间数据处理
机器学习算法
PostGIS
数据可视化
python

为什么要使用Python?

1. **Python 语法简单**：对代码格式不严格，类似伪代码
2. **Python 是开源的**：代码和解释器是开源的
3. **Python 是免费的**：开源!=免费（商用付费）
4. **Python 是高级语言**：能够自动管理内存
5. **Python 是解释型语言**：跨平台
6. **Python 是面向对象的编程语言**：类或对象组织代码
7. **Python 功能强大（模块众多）**：模块免费提供
8. **Python 可扩展性强**：与不同语言衔接好

Python缺点

1. **运行速度慢**
   - 一边运行一边“翻译”源代码
   - 远远慢于 C/C++，还慢于 Java
2. **代码加密困难**
   - 直接运行源代码，因此对源码加密是比较困难的

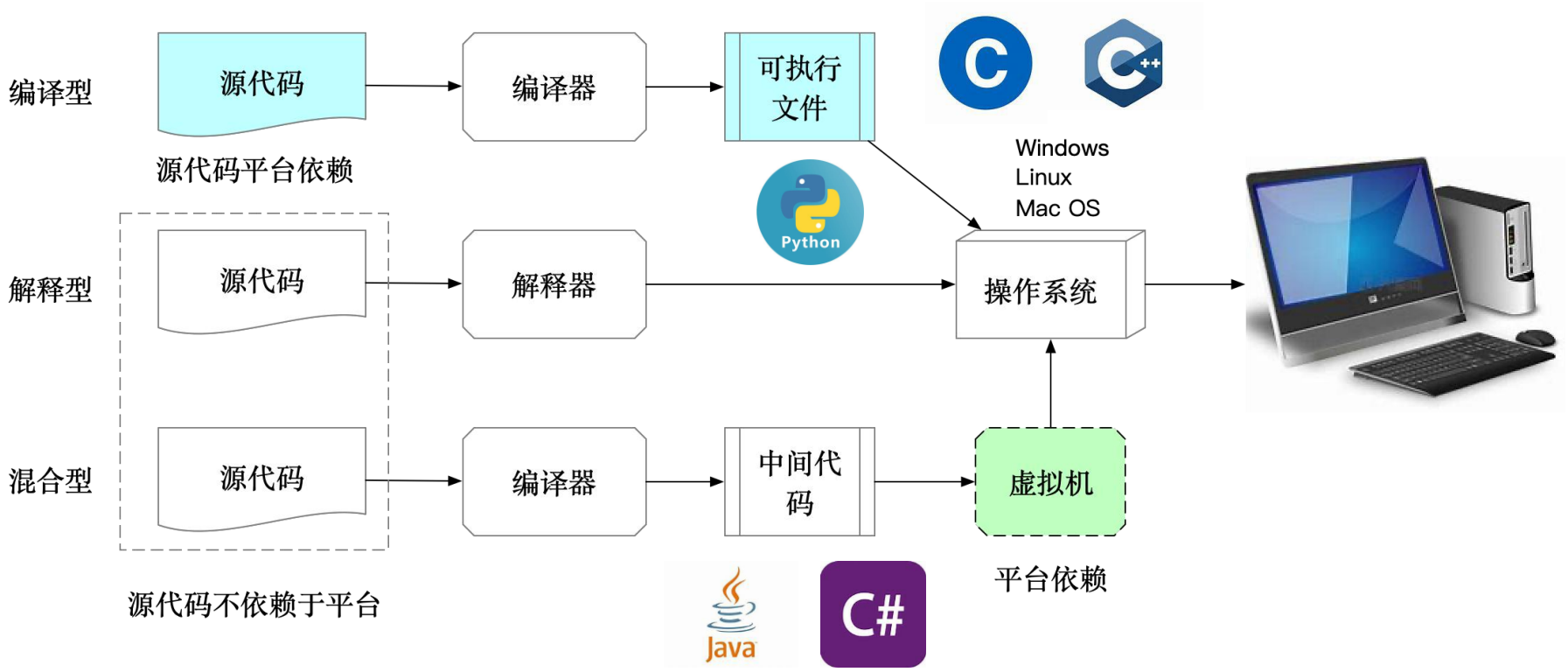

```
void main(void)
{
char* p = “Hello World1”;
char a[] = “Hello World2”;
p[2] = ‘A’;
a[2] = ‘A’;
char* p1 = “Hello World1”;
}
```

这种写法相当于

```
const char ro[] = {“Hello World1”};
char *p = a;
```

```
C_S   SEGMENT
      ASSUME   CS: C_S, DS: C_S
S_T:
      MOV   AX, C_S
      MOV   DS, AX
      LEA   DX, P_S
      MOV   AH, 9
      INT   21H
      MOV   AH, 4CH
      INT   21H
P_S   DB    'Hello World!', 36
C_S   ENDS
      END   S_T
```

计算机高级语言的类型


编译型
源代码
编译器
可执行文件
源代码平台依赖
Windows
Linux
Mac OS
解释型
源代码
解释器
操作系统
混合型
源代码
编译器
中间代码
虚拟机
平台依赖
源代码不依赖于平台
Python
Java
C#

一个简单的例子

问题描述

1. 200个学生成绩单的Excel文件，包含学生姓名
2. 要求将姓名转成对应的拼音

姓名	拼音	平时成绩	笔试成绩	总评成绩
杨		100	93	94
黄		100	85	87
刘		100	93	94
宋		100	93	94
苗		100	85	87
马		100	86	87
陈		100	80	82
杨		100	93	94
姜		100	85	87
刘		100	84	86
王		100	85	87
崔		100	78	80
孙		100	78	80
崔		100	78	80
王		100	78	80
刘		100	80	82
王		100	83	85
向		100	83	85
程		100	80	82
段		100	85	87
田		100	84	86
胡		100	78	80
田		100	80	82
王		100	78	80

百度搜索

二、pypinyin

另一种方法是使用pypinyin，安装同样可以使用pip

```
# 安装
pip install pypinyin -i http://pypi.douban.com/simple --trusted-host pypi.doub
```

直接导入就行?

```
import pypinyin
```

再来看看中文名转拼音的实现办法

```
result1 = pypinyin.pinyin('叶庭云', style=pypinyin.NORMAL)
result1
```

结果如下:

```
[['ye'], ['ting'], ['yun']]
```

启用多音节来实现声调

```
result2 = pypinyin.pinyin('叶庭云', heteronym=True)
result2
```

结果如下:

```
[['yè', 'xié'], ['tíng'], ['yún']]
```

因为返回的是一个嵌套的list，所以需要简单调整一下

程序编写

In [2]:
```
D=pd.read_excel('./Data_Sets/成绩单.xlsx')
```

In [5]:
```
D.iloc[:,[0,2,3,4]]
```

	序号	姓名	分组	成绩
0	1	杨哲晰	1	93
1	2	黄宸	3	85
2	3	刘岳	1	93
3	4	宋鑫锋	1	93
4	5	苗时雨	3	85
...	...	...	...	...
177	178	费腾	36	80
178	179	武冠韫	39	82
179	180	赵铭浩	38	86
180	181	孙文宁	36	80
181	182	韩肖悦	37	83

182 rows × 4 columns

In [48]:
```
D['Pinyin']=""
```

In [120]:
```
def toPinyin(str):
    tmp=pp.pinyin(str,style=pp.NORMAL)
    str_py=tmp[0][0][0].upper()+tmp[0][0][1:]
    str_py+=" "
    str_py+=tmp[1][0][0].upper()+tmp[1][0][1:]
    if(len(str)>2):
        str_py+=tmp[2][0]
    return str_py
```

In [121]:
```
for i in range(len(D)):
    D.iloc[i,-1]=toPinyin(D.iloc[i,2])
```

In [12]:
```
for i in range(len(D)):
    D.iloc[i,-1]=toPinyin(D.iloc[i,2])
```

In [13]:
```
D.iloc[:,[0,2,3,4,5]]
```

	序号	姓名	分组	成绩	Pinyin
0	1	杨哲晰	1	93	Yang Zhexi
1	2	黄宸	3	85	Huang Chen
2	3	刘岳	1	93	Liu Yue
3	4	宋鑫锋	1	93	Song Xinfeng
4	5	苗时雨	3	85	Miao Shiyu
...	...	...	...	...	...
177	178	费腾	36	80	Fei Teng
178	179	武冠韫	39	82	Wu Guanyun
179	180	赵铭浩	38	86	Zhao Minghao
180	181	孙文宁	36	80	Sun Wenning
181	182	韩肖悦	37	83	Han Xiaoyue

182 rows × 5 columns

In [38]:
```
D.to_excel('./Data_sets/MBA成绩单withPinyin.xlsx
```

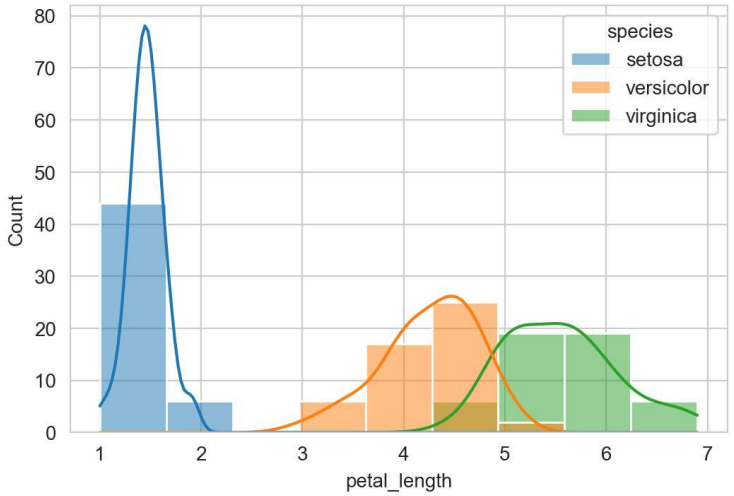
Python的应用场景——数据分析框架概览

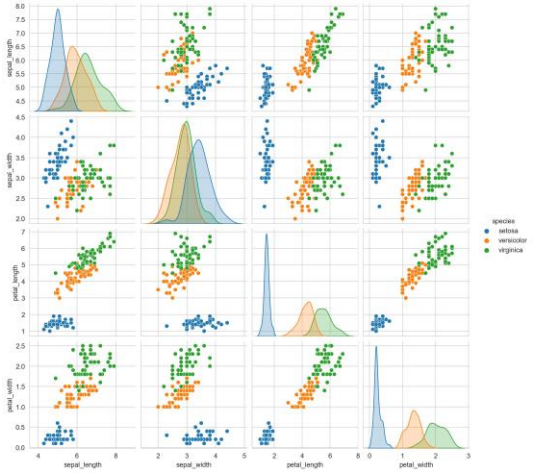
数据访问

- 数据库（MySQL/SQL Server/Oracle）
- 格式化文件（csv/xlsx/jason）

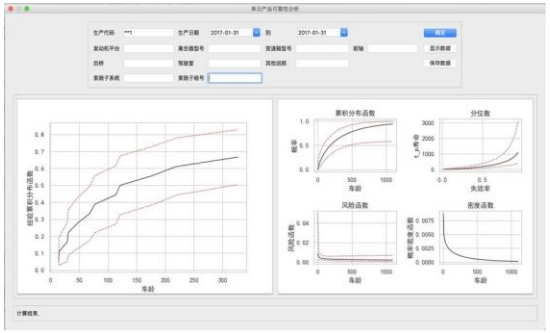
数据分析

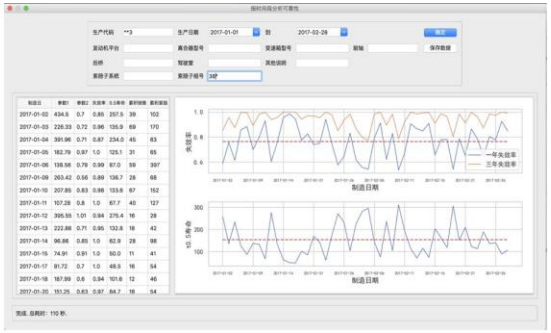
- 数据可视化（Matplotlib/seaborn）
- 数据操纵（Pandas）
- 数据建模
- 统计分析（scipy.stats/statsmodels）
- 机器学习（sklearn）
- 最优化（scipy.optimization/cvxpy）
- 符号计算（sympy）
- 深度学习（Tensorflow/Pytorch）

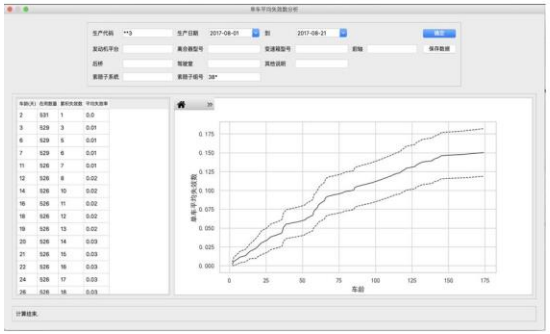


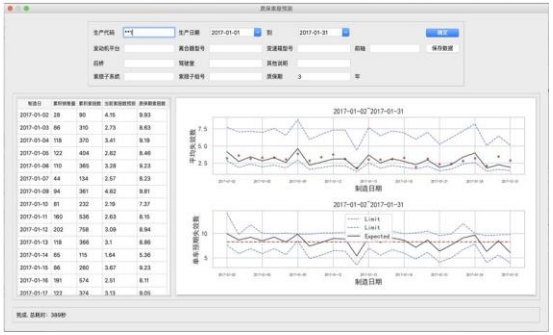


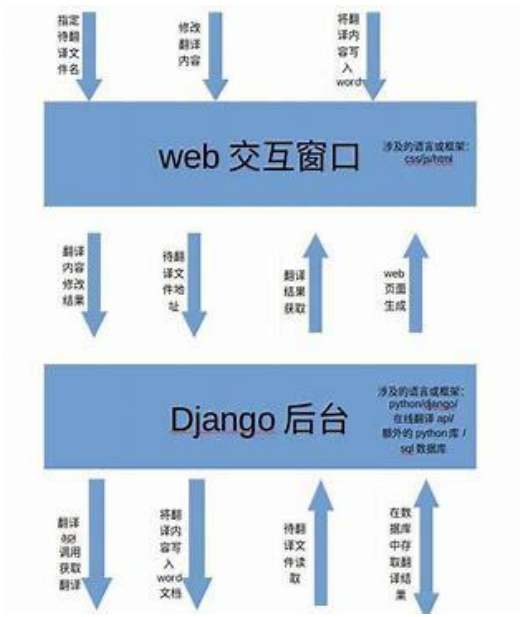
Python的应用场景——GUI+Web程序开发









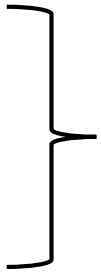


Python的版本

- Python 2.X 已停止维护
- 目前的发行版本是Python 3.x
- Python 3.x 对Python 2.x 不完全向下兼容代码（阅读代码时要注意），最明显特征
  - Python 2.x
    - print ‘Hello World.’
  - Python 3.x
    - Print(‘Hello World.’)

Python的版本

- numpy——矩阵运算
- pandas——数据操纵
- matplotlib & seaborn——数据可视化
- scipy——科学计算
- sympy——符号计算
- sklearn——机器学习算法
- nltk——自然语言处理



三剑客

使用Python需要注意的问题

Python的严谨性不足

- 除保留字外不做其他检查
- 不做变量类型检查

同样的功能可以有多种实现方法，容易混淆

开源包缺乏严谨的规划和设计

- 同样的函数在不同的包中重复出现
  - 或名称不同、或参数定义不同、或完全相同
- 部分函数的参数定义不符合大众习惯
  - 复杂的统计学函数

如何学好Python

- 立即开始使用——从简单的程序代码开始学习
- 不断练习，从已有代码的运行开始
- 善于利用网络资源和搜索引擎
- 不要试图搞清楚所有技术细节（绝大多数情况下这是不可能的），适用即可

Python标识符

Python注释

为方便对代码的理解，在代码中需要增加大量的注释

Python提供了两种注释方法

- 单行注释
  - #
- 多行注释
  - 三个单引号括起来的语句块
    - '''
    - a=3
    - b=4
    - '''

Python的行与缩进——最具特点的语法

```
If experession:
    print('True')
    print('123')
elif:
    print('False')
    print('456')
```

语句块

语句块

```
a=3
b=5
if a>b:
    print('True')
    print('123')
elif b==5:
    print('False')
else:
    print('456')
```

```
a=3
b=5
if a>b:
    print('True')
    print('123')
else:
    if b==5:
        print('False')
    else:
        print('456')
```

Python标识符

标识符（包括变量、函数）

以字母或下划线_开头的字母、数字或下划线组合。注意：大小写敏感

例如：

- a12bc.
- __abc
- __fx

注意：自定义变量不能与保留字重复(不包含内置函数)

- for=5（错误）
- str=5（语法正确，但可能导致不可预知的结果）

保留字-Python提供了保留字列表

In [12]:

```
import keyword
kw=keyword.kwlist
print(kw)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Python内置函数

注意：所有这些内置函数名不能当做 普通变量使用，尽管符合Python语法！

Python 内置函数

内置函数				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reverse()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	exec 内置表达式

Python内置函数

```
In [4]:
a=123
b=str(a)
a,b
```

```
(123, '123')
```

```
In [5]:
str=5
c=str(a)
c
```

内置函数不是保留字

```
TypeError                 Traceback (most recent call last)
<ipython-input-5-c6c08f1c4ab5> in <module>
      1 str=5
----> 2 c=str(a)
      3 c

TypeError: 'int' object is not callable
```

```
In [7]:
for=5
```

保留字检查

```
  File "<ipython-input-7-4414c26ff868>", line 1
    for=5
       ^
SyntaxError: invalid syntax
```

```
In [5]:
b='123'
b
```

```
'123'
```

```
In [7]:
a=int(b)
a
```

```
123
```

```
In [8]:
int=5
c=int(b)
```

内置变量类型不是保留字

```
TypeError                 Traceback (most recent call last)
<ipython-input-8-ce8f80fe7820> in <module>
      1 int=5
----> 2 c=int(b)

TypeError: 'int' object is not callable
```

- str作为内部函数，int作为变量名，被当做普通变量赋值， 在之后的函数调用中出错！
- for作为语法保留字，作为普通变量使用，提示错误！

Python基础——如何获得帮助

- Python基础函数以及包都提供了标准格式的帮助信息
- 可以通过两种形式获得帮助
  - help(函数名)——详细帮助信息
  - print(函数名.__doc__)
- 更详细的信息可以通过搜索引擎获得

```
help(str)

Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
```

```
print(str.__doc__)

str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
```

Python基础变量——数字类型

- 整数(int)
- 布尔型(bool)
- 浮点型(float)
- 复数(complex) X
  - counter=100 #整形变量
  - miles=1000.0 #浮点型变量
  - sig=True #bool变量（True, False)
  - print(counter)
  - print(miles)
  - print(name)

Python变量

- 不需要事先定义
- 不需要指定数据类型，直接赋值即可

Numpy数据类型——对Python数据类型进行了扩展

int8	字节（-128 to 127）
int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）
uint8	无符号整数（0 to 255）
uint16	无符号整数（0 to 65535）
uint32	无符号整数（0 to 4294967295）
uint64	无符号整数（0 to 18446744073709551615）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位

Python基础变量——字符串类型

- s='abc' 或 s=''abc''
- s='中国'
- 字符串
  - 字符串可以用''+''连接
  - 字符串不能改变
  - 在Python中，字符就是长度为1的字符串
  - 字符串的截取方法同列表

字符串操作

- len(s)
- s.strip() #去掉字符串s两端的空白符及格式符如’\n’、’\t’等
- s.strip(s1) #去掉s字符串两端包含在s1中的字符

```
s='\tabcd\n'
print(s)
print('----')
print(s.strip())
print('----’ )
s='abcdabcba'
print(s.strip('ab'))
```

多行语句

- 如果语句过长，会影响程序的可阅读性
- 可通过换行缩短语句将语句变短（相当于连接符）
- 同一行显示多条语句

```
total=3+\
      5+\
      8
print(total)
```

```
a=3;b=5;c='123'
```

```
a=b=c=1
```

Python基础数据结构

多行语句

- + #数、字符串、list
- 'abc'+'def' [1,2,3]+[4,5,6]
- -
- * #数、list,
- / #除法
- +=,-=,*=,/=
- % #取余数
- // #整除
- ** #乘方

5//2=2（2.5向负无穷方向取整为2），同时-5//2=-3（-2.5向负无穷方向取整为-3）

- ==
- !=
- >
- <
- >=
- <=
- and 或 &
- or |
- not
- in
- is #判断引用是否为同一个对象

示例

- A=[1,2,3]
- B=A
- A==B? True or False
- A is B? True or False
- 可以用id测试
- A=[1,2,3]
- B=[1,2,3]
- A==B?
- A is B?

```
>>> A==B
True
>>> A is B
True
```

"=="的正式运算是相等，而"is"的运算是标识。用"=="是比较两个对象的值。"a ==b"应解释为"a的值是否等于b的值"。在上述所有示例中，a的值始终等于b的值(即使对于空列的示例也是如此)，因此"a == b"” 始终为真。

```
>>> A==B
True
>>> A is B
False
```

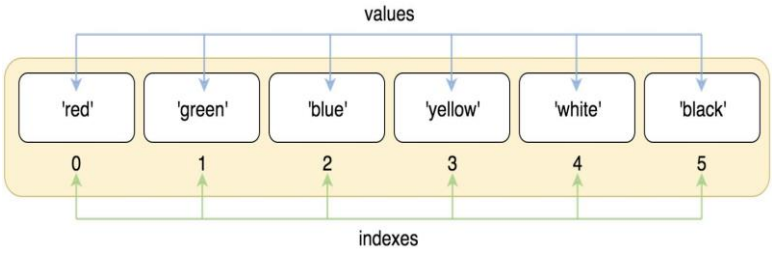
在解释标识的概念之前，我需要先介绍一下id函数。对象的标识可以通过id函数来获得。一个对象的标识始终是唯一且恒定的，你可以将其视为该对象的地址。如果两个对象的标识相同，则它们的值也一定相同。

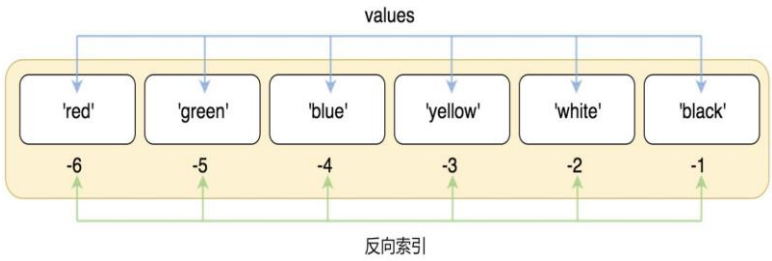
列表

- 列表是Python最基础的数据结构
- 列表索引从0开始
- 主要操作：
  - 加、乘、切片、索引
- 确定序列长度
- 确定最大和最小元素
- 列表中不需要所有的元素具有相同类型
- 用方括号表示列表，用逗号分割元素

列表索引

- list1=['red','green','blue','yellow','white','black']
- len(list1)#长度为6
- 正向索引：从0到5
- 反向索引：从-1到-6
- 索引和截取
- list1[0]
- list1[-1]
- list1[1:4] #截取第1，2，3个元素
- list1[:4] #截取第0，1，2，3个元素
- list1[1:] #从第一个元素开始的所有元素
- list1[:-2] #从第一个元素至倒数第二个元素
- list1[::-1] #从最后一个元素倒排





列表操作

- **列表拼接**
  - list1 = ['a','b','c’]
  - list 1 + =[1,2,3] 或者list1 =list1 +[1,2,3]

  思考：list1*4的结果？

- **列表嵌套**
  - a=[1,2,3]
  - b=['a','b','c']
  - list2=[a,b] #[[1,2,3],['a','b','c']]

列表其他函数和方法

- len(list)/max(list)/min(list)/list(seq)
- list对象方法
  - list.append(obj) #将对象obj加入到列表最后
  - list.count(obj) #计算某一个对象在列表中出现的次数
  - list.extend(seq) #将序列seq追加到列表末尾 #append与extend有什么区别？
  - list.index(obj) #元素obj在列表中的第一个索引值
  - list.insert(index,obj) #在位置index插入元素obj
  - list.pop() #将最后一个元素移除并返回
  - list.remove(obj) #移除obj元素的第一个匹配项
  - list.reverse() #将列表中的元素反向
  - list.sort(key=None,reverse=False) #列表元素排序，元素必须是可比较的
  - list.clear() #清空
  - list.copy() #复制列表

列表示例——列表构建和连接

```
A0=['A','B','C','D','E','F','G']
A1=list('ABCDEFG')
B1=['H','I','J']

print('A1+B1',A1+B1)
A1.append(B1)
print('A1.append(B1)',A1)
A1=A0
A1.extend(B1)
print('A1.extend(B1)',A1)
```

```
A1+B1 ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
A1.append(B1) ['A', 'B', 'C', 'D', 'E', 'F', 'G', ['H', 'I', 'J']]
A1.extend(B1) ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

列表示例——列表切片和索引

```
A=list('ABCDEFGHIJKLMN')
A
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']
```

```
print('A[1:4]',A[1:4])
print('A[:4]',A[:4])
print('A[:-2]',A[:-2])
print('A[::-1]',A[::-1])
```

```
A[1:4] ['B', 'C', 'D']
A[:4] ['A', 'B', 'C', 'D']
A[:-2] ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
A[::-1] ['N', 'M', 'L', 'K', 'J', 'I', 'H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

列表示例

列表示例——排序

```
A=list('ABCDEFG')
A
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

```
A.sort()
print('A.sort()',A)
A=A[::-1]
print('A[::-1]',A)
```

```
A.sort() ['A', 'B', 'C', 'D', 'E', 'F', 'G']
A[::-1] ['G', 'F', 'E', 'D', 'C', 'B', 'A']
```

```
B=['C','D','E']
A.append(B)
A.sort()
```

```
---------------------------------------------------
TypeError                 Traceback (most recent call last)
<ipython-input-36-0332df42f87d> in <module>
      1 B=['C','D','E']
      2 A.append(B)
----> 3 A.sort()

TypeError: '<' not supported between instances of 'list' and 'str'
```

```
print(A)
```

```
['G', 'F', 'E', 'D', 'C', 'B', 'A', ['C', 'D', 'E']]
```

其他操作

```
A=list('ABCACDEBCBE')
```

```
print("A.count('A')",A.count('A'))
B=A.pop()
print(B)
print(A)
```

```
A.count('A') 2
E
['A', 'B', 'C', 'A', 'C', 'D', 'E', 'B', 'C', 'B']
```

```
A.remove('B')
A
```

```
['A', 'C', 'A', 'C', 'D', 'E', 'B', 'C', 'B']
```

```
A.clear()
A
```

```
[]
```

元组（tuple）

元组是用小括号括起来的元素，用逗号分割:

- t=('a', 'b', 'c')

元组与列表类似，但元组不能修改元组的索引方法与列表相同

元组基本操作:

- len(tuple)
- (1,2,3)+(4,5,6)
- ('hi')*4.
- 3 in (1,2,3)
- for x in(1,2,3):

  print(x)

集合（set）——不可重复元素

用{}或set()函数创建，但空集只能用set()创建

集合中的元素不能重复

例：

- s={' apple',' pear',' orange'} 或s=set('apple', 'pear','orange’)
- 'orange' in s
- 'c' in s
- s.remove('apple') # 移除某一元素
- s.discard('apple') # 与remove功能相同， 但元素不存在不报错len(s)
- s.clear()
- s1.issubset(s2) # s1是否是s2的子集

```
s1=set('abracadabra')
s2=set('alacazam')
print(s1,s2)
```

```
{'b', 'c', 'r', 'd', 'a'} {'l', 'm', 'c', 'z', 'a'}
```

```
s1-s2
```

差集

```
{'b', 'd', 'r'}
```

```
s1|s2
```

并集

```
{'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
```

```
s1 & s2
```

交集

```
{'a', 'c'}
```

```
s1 ^ s2
```

补集

```
{'b', 'd', 'l', 'm', 'r', 'z'}
```

元组、列表和集合类型转换

```
A=[1,2,3,4]
A_set=set(A)
A_tuple=tuple(A)
B=(1,2,3,4)
B_list=list(B)
B_set=set(B)
C=set(1,2,3,4)
C_list=list(C)
C_tuple=tuple(C)
```

字典（dict）——键值对

- d={ key 1 : value 1 , key 2 : value 2 , key 3 : value 3 }
- 键必须是唯一的， 但值可以重复
- 值可以取任意数据类型， 键一般为数字或字符串
- 例:

d={' name':' wang',' age': 20 , 'salary': 5000 }

print(d['age’])

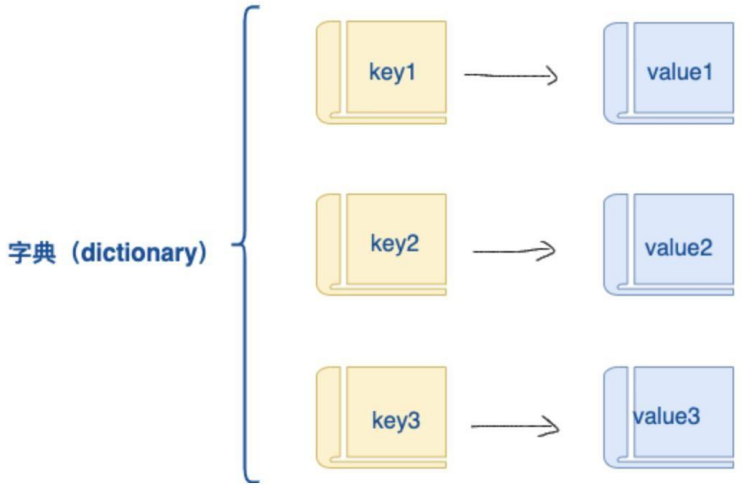
d['age'] = 25

del d[' name’]

d. clear() # 清空

del d # 删除 {}

创建一个空字典



字典基本操作

- items()
- keys()
- pop(key)
- len(dict)

```
d={'name':'wang','age':30,'salary':8000.0}
```

```
list(d.items())
```

```
[('name', 'wang'), ('age', 30), ('salary', 8000.0)]
```

```
list(d.keys())
```

```
['name', 'age', 'salary']
```

```
list(d.values())
```

```
['wang', 30, 8000.0]
```

```
print(d.pop('name'))
d
```

```
wang

{'age': 30, 'salary': 8000.0}
```

字典的访问和修改

- 设字典数据为

  D={'H':3,'B':2,'W':8,'L':12}

  - 如果访问D['H']返回3
  - 如果访问D['K']则报错
  - 如果不知道某个key是否存在于字 典中，则需要先进行判断

```
D={'H':3,'B':2,'W':8,'L':12}
D['H']
```

```
3
```

```
D['K']
```

```
---------------------------------------------------
KeyError                  Traceback (most recent call last)
<ipython-input-109-f1b7647e3a65> in <module>
----> 1 D['K']

KeyError: 'K'
```

```
if 'K' in D.keys():
    print(D['K'])
else:
    print("Key value %s 不存在"%('K'))
```

```
Key value K 不存在
```

字典的访问和修改

- 设字典数据为

  D={'H':3,'B':2,'W':8,'L':12}

  - 执行D['H']=5
  - 执行D['K']=5呢?
- 在修改时，如果键存在，则修改原值为新值；否则在字典中加入新的键值对

```
D
```

```
{'H': 3, 'B': 2, 'W': 8, 'L': 12}
```

```
D['H']=5
D
```

```
{'H': 5, 'B': 2, 'W': 8, 'L': 12}
```

```
D['K']=8
D
```

```
{'H': 5, 'B': 2, 'W': 8, 'L': 12, 'K': 8}
```

Python控制语句

条件语句（分支）

```
if condition1:
    语句块
elif condition2:
    语句块
else:
    语句块
```

注意条件语句后的冒号

```
if condition1:
    语句块
else:
    语句块
```

条件语句嵌套

```
if 表达式1:
    语句
    if 表达式2:
    语句
    elif 表达式3:
    语句
    else:
        语句
elif 表达4:
    语句
else:
    语句
```

过于复杂的嵌套会影响程序的可读性，不建议使用

多个逻辑表达式通过操作符连接

操作符	描述
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于，比较两个值是否相等
!=	不等于

循环 for形式

```
for <variable> in
    <sequence>:
<statements>  else:
<statements>
```

实例

```
languages=[' C',' C++','
    Perl',' Python',' PHP']
    for x in languages:
print( x)
```

```
for <variable> in <sequence>:
    <statements>  i f
    <condition>:
        break
else:
    <statements>
```

循环 for+range()

```
for i in range(n):
    print(i)
    <statement>
```

```
range( n)
输出0 ~( n- 1 ) 的序列
```

```
range( a, b)
输出a~( b- 1 ) 的序列
```

循环 while形式

```
while <expr>:
<statements>  else:
<additional_statements>
```

```
a= 1
while  a< 10 :  print( a)
a+= 2
```

break

符合条件的话退出循环

continue

符合条件则进行下一次循环条件判断

（忽略continue后面的语句）

分别用for和while循环求1~100的和?

```
n=100
sum=0 counter=1
while counter<=100:
    sum+=counter  counter+=1
print(sum)
```

```
n= 100
sum= 0
for i in range( 1 , 101 ):
    sum+=i

print(sum)
```

捕捉并处理运行时异常


try:
执行代码
except:
发生异常时执行的代码
try:
执行代码
except:
发生异常时执行的代码
else:
没有异常时执行的代码

异常处理实例

无异常

```
a=10
try:
    b=a/5
except:
    print('运行时异常! ')
else:
    print(b)
```

```
2.0
```

存在异常

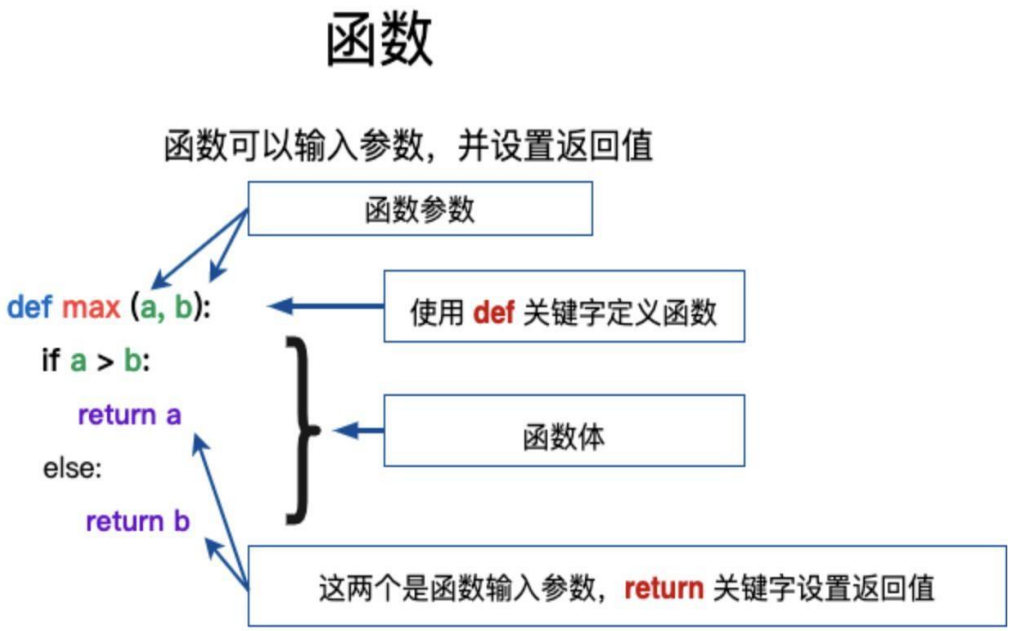
```
a=10
try:
    b=a/0
except:
    print('运行时异常! ')
else:
    print(b)
```

```
运行时异常!
```

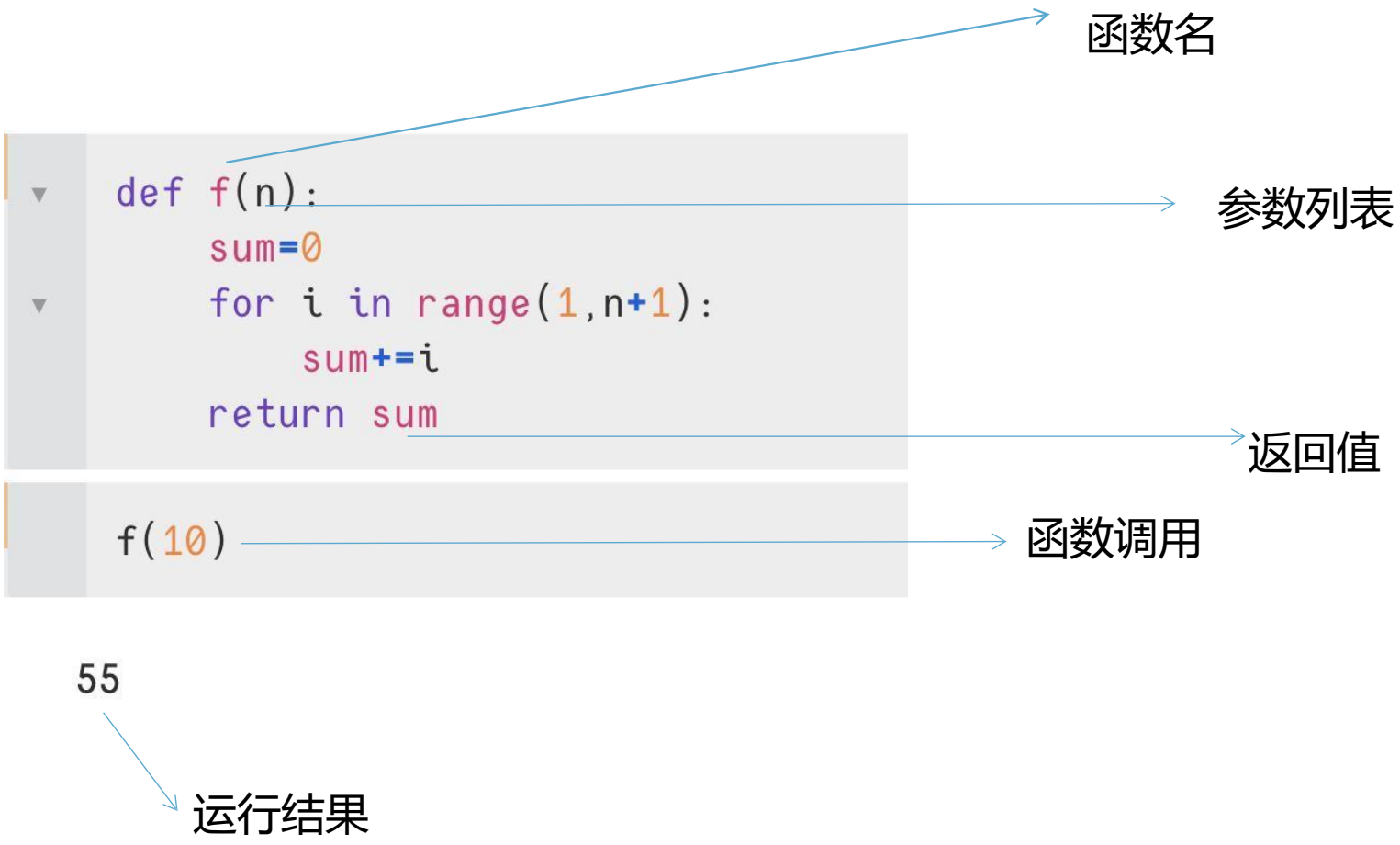
Python函数的定义与使用

函数（Function）

- 函数是组织好的，可重复使用，用来实现单一或相关联功能的代码段
- 可以通过自定义函数定义可重复使用的代码段，称为自定义函数
- Python函数不需要指定返回值的类型和个数

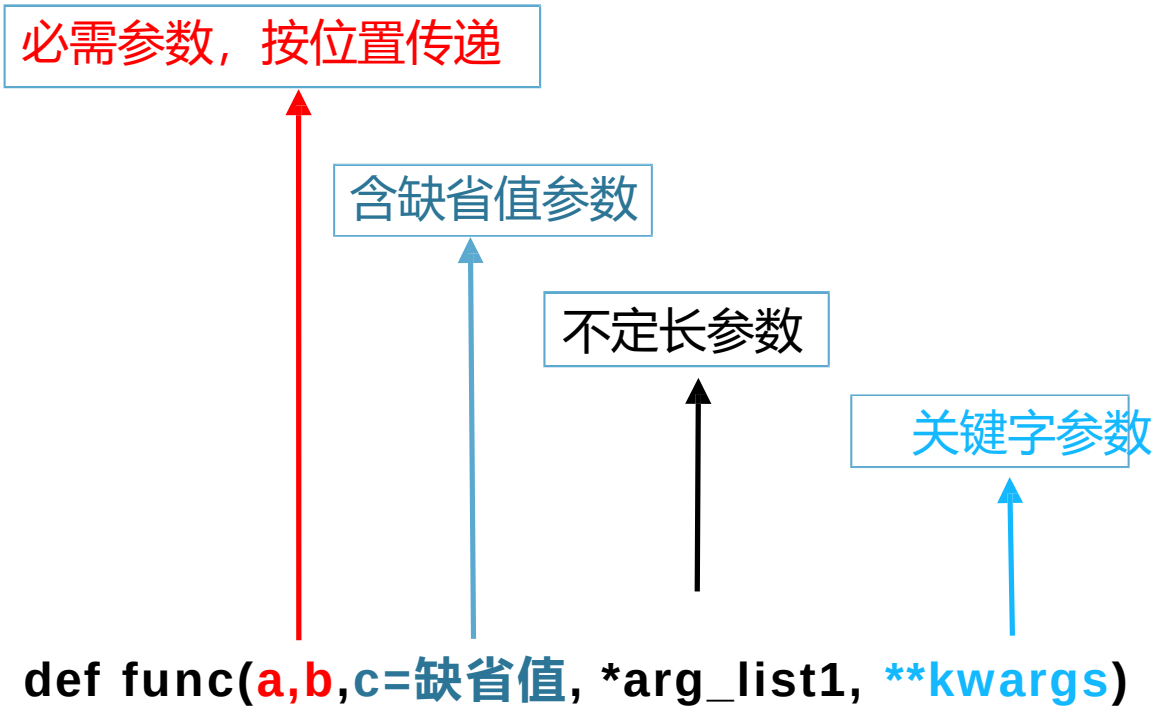


函数实例


函数名
def f(n):
sum=0
for i in range(1,n+1):
sum+=i
return sum
参数列表
返回值
f(10)
函数调用
55
运行结果

参数类型实例

- 必需参数
  - 按位置传递
- 默认参数
  - 可以给定值或使用缺省值
- 不定长参数
  - 参数个数不限，通过列表传递
- 关键字参数
  - 通过关键字传递



参数类型实例

```
def salary(name,age=20,*month_salary,**kwargs):
    print('姓名: ',name)
    print('年龄: ',age)
    if(len(month_salary)>0):
        print('月工资: ',month_salary)
        sum=0
        for i in range(len(month_salary)):
            sum+=float(month_salary[i])
    if(len(kwargs)>0):
        if(kwargs['method']=='total'):
            print('总工资: ',sum)
        if(kwargs['method']=='average'):
            print('平均工资: ',round(sum/len(sum),2))
```

参数类型实例

```
salary('wang')
```

```
姓名:  wang
年龄:  20
```

```
salary('wang',30)
```

```
姓名:  wang
年龄:  30
```

```
salary('wang',30,5800.0,6200.0,6800.0)
```

```
姓名:  wang
年龄:  30
月工资:  (5800.0, 6200.0, 6800.0)
```

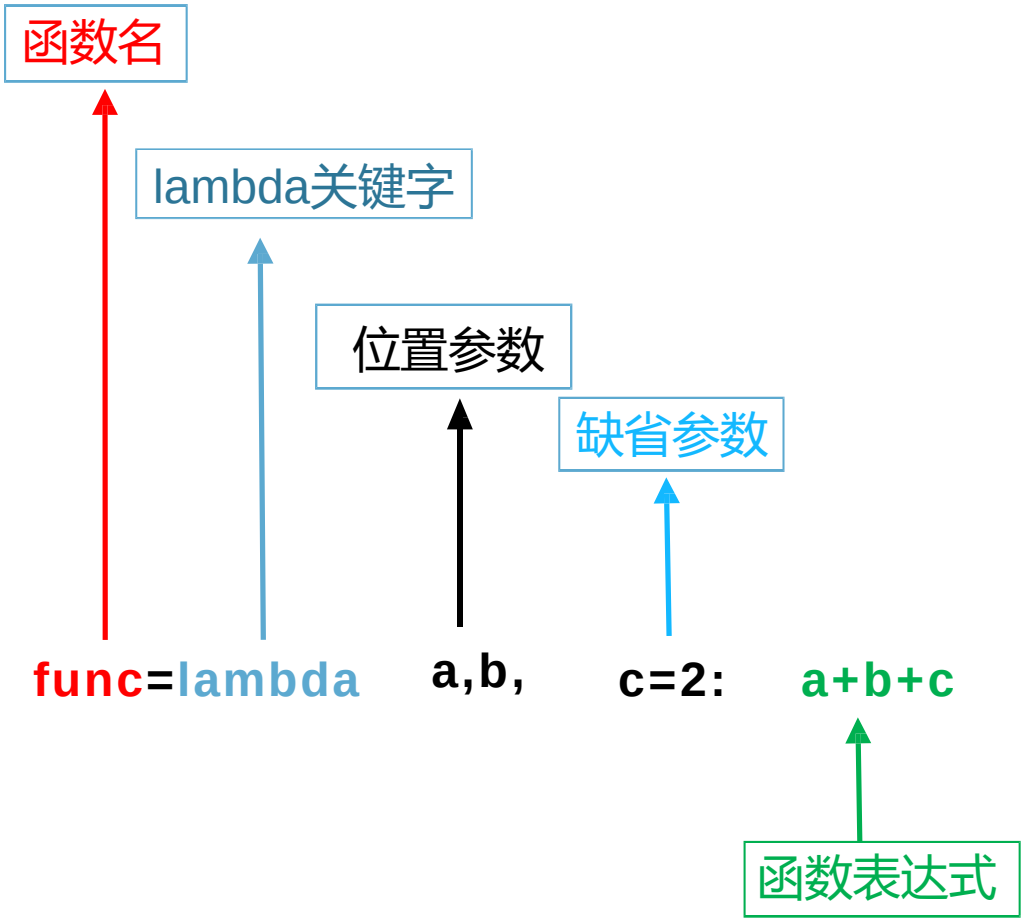
```
salary('wang',30,5800.0,6200.0,6800.0,method='average')
```

```
姓名:  wang
年龄:  30
月工资:  (5800.0, 6200.0, 6800.0)
平均工资:  6266.67
```

```
salary('wang',30,5800.0,6200.0,6800.0,method='total')
```

```
姓名:  wang
年龄:  30
月工资:  (5800.0, 6200.0, 6800.0)
总工资:  18800.0
```

lambda:定义简单函数的方法




```
func=lambda a,b,c=2:a+b+c
```

```
func(3,2)
```

```
7
```

```
func(3,2,8)
```

```
13
```

Python Type Hint(类型标注/类型提示)

- Python3中 箭头-> 是函数**注释**的一部分，表示函数返回值的类型。

```
def useful_function(x) -> int:
    # Useful code, using x, here
    return x
```

- Python是一门动态类型语言，就是它每一个变量是什么类型，是在它runtime的时候决定的。在写大型项目的时候，很多变量因为不知道是什么类型一眼看不明白其含义
- 那么从python3.5开始，就逐渐引入了type hint (type annotation)，就是让你在写python的时候，可以可选地标注变量类型
- 注意Python 运行时不强制执行函数和变量类型注解

普通写法

```
def f(a, b):
    return a + b
print(f(1,2))
```

type hint写法

```
def f(a: int, b: int) -> int:
    return a + b
print(f(1,2))
```

python面向对象编程简介

类的定义

```
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性,私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    def speak(self):
        print("%s 说: 我 %d 岁。" %(self.name,self.age))
```

对象的定义和使用

```
p=people('wang',30,50)
```

```
p.speak()
```

```
wang 说: 我 30 岁。
```