

Python基础内容

## 第二章 内置对象

# 目录

---

- 初步理解面向对象
- 简单语法
- 用户输入和输出
- 数字和简单运算
- 字符串
- 列表
- 元组
- 字典
- 集合

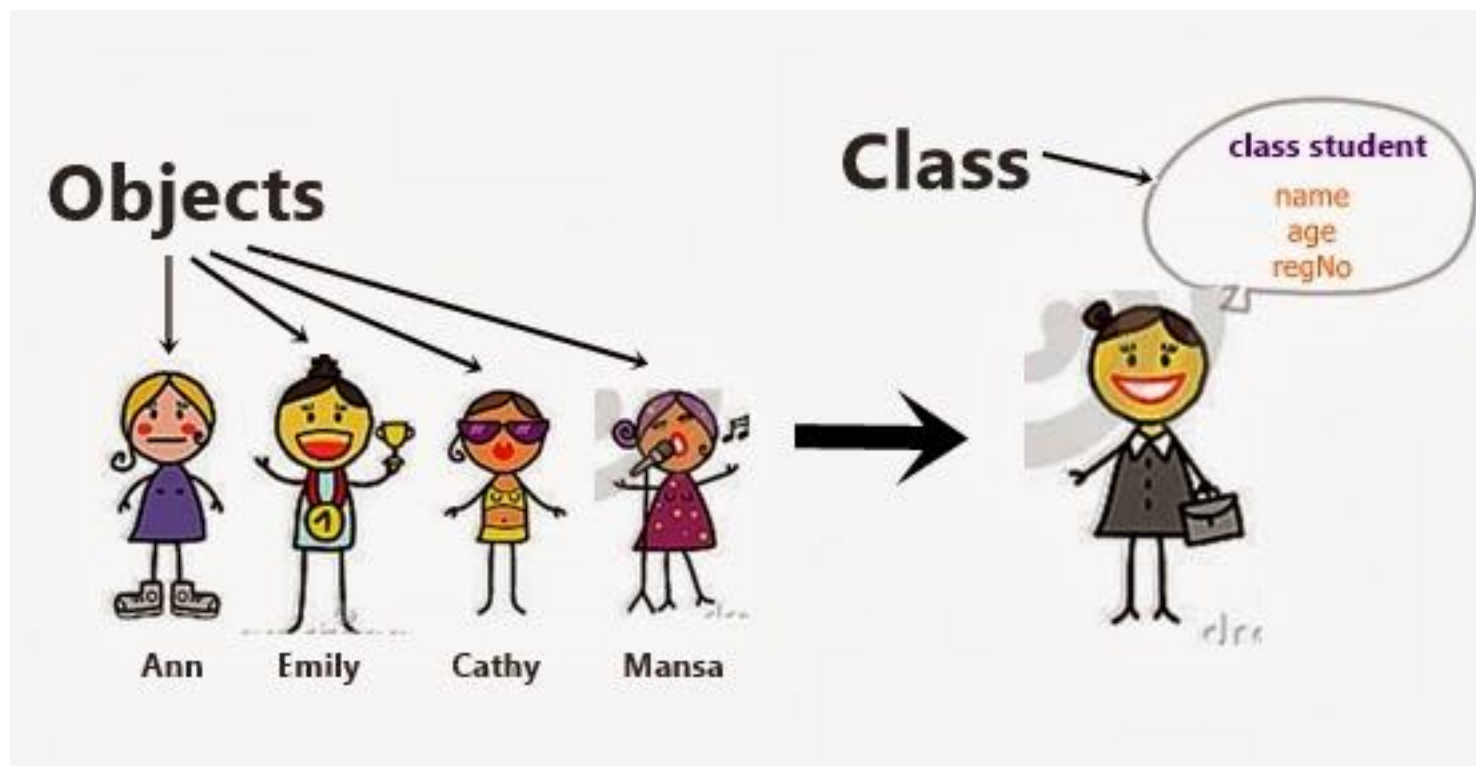
---

# 初步理解面向对象

对象概念，属性，方法

# 对象

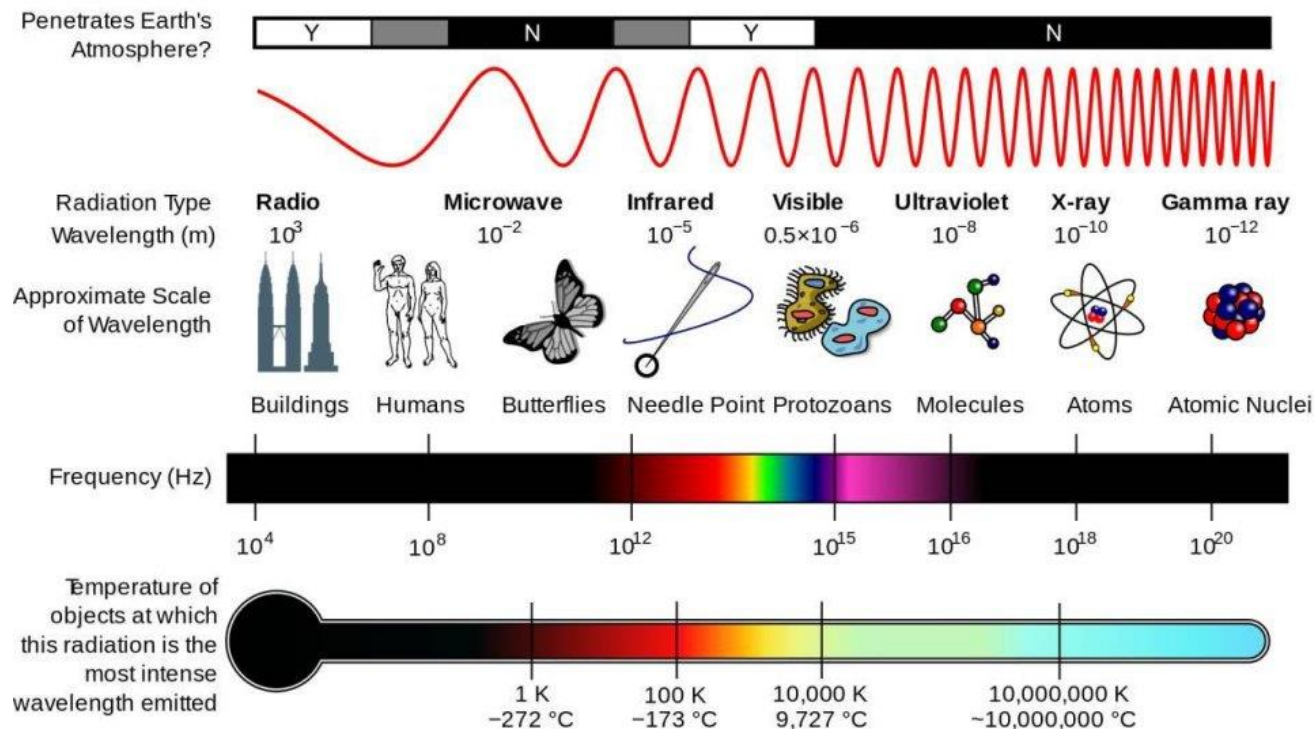
- 什么是对象



*Picture from website*

# 对象

- 组成物理世界的基本物质
  - 朴素的观点：金、木、水、火、土
  - 物理学：分子、原子、核子、夸克以及其它基本粒子

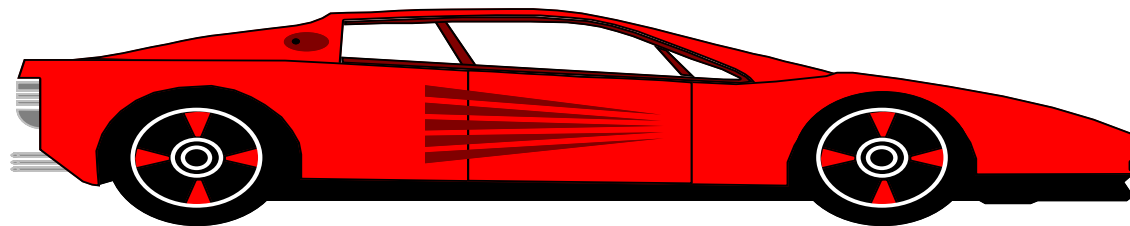
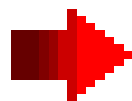


# 面向对象

—对象

何为对象？

自然实体  
(有形)



概念实体  
(无形)



生产计划

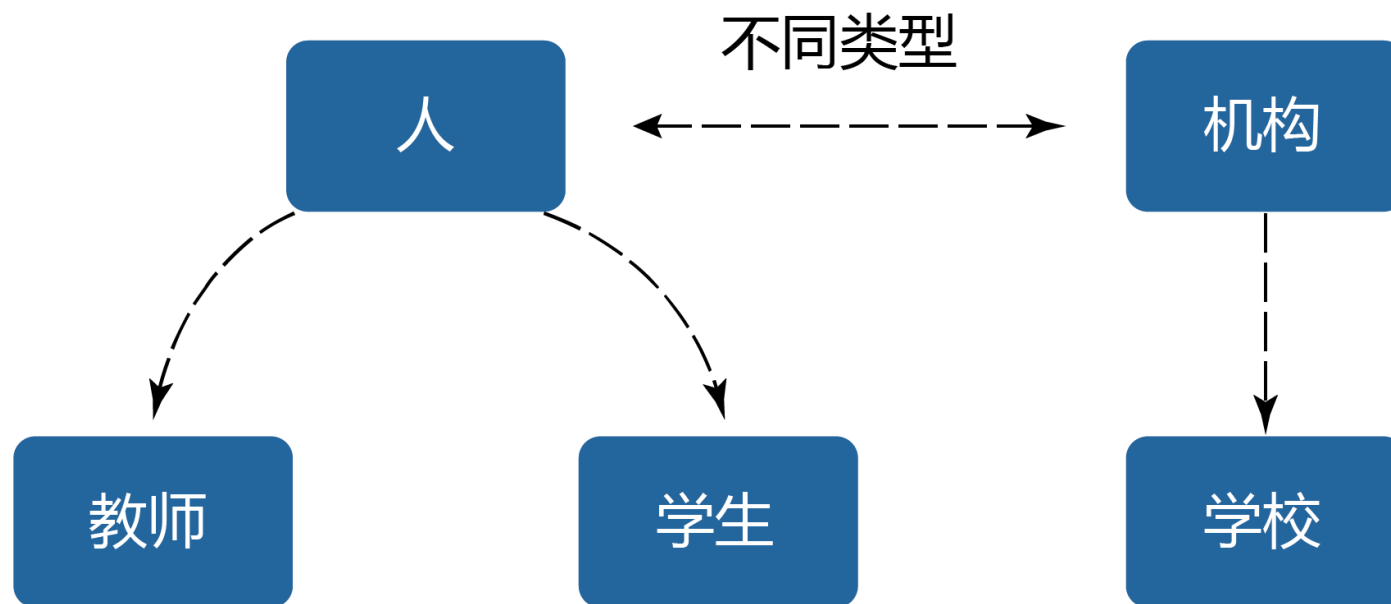
# 对象

---

- 对象分析：学生上学缴费过程
  - 对象1 → 学生：姓名、会支付、会说话、 .....
  - 对象2 → 财务教师：姓名、会说话、会收钱、 .....
  - 对象3 → 学校：地址和名称、收费标准、学生名录、教师名录、 .....

# 对象

---

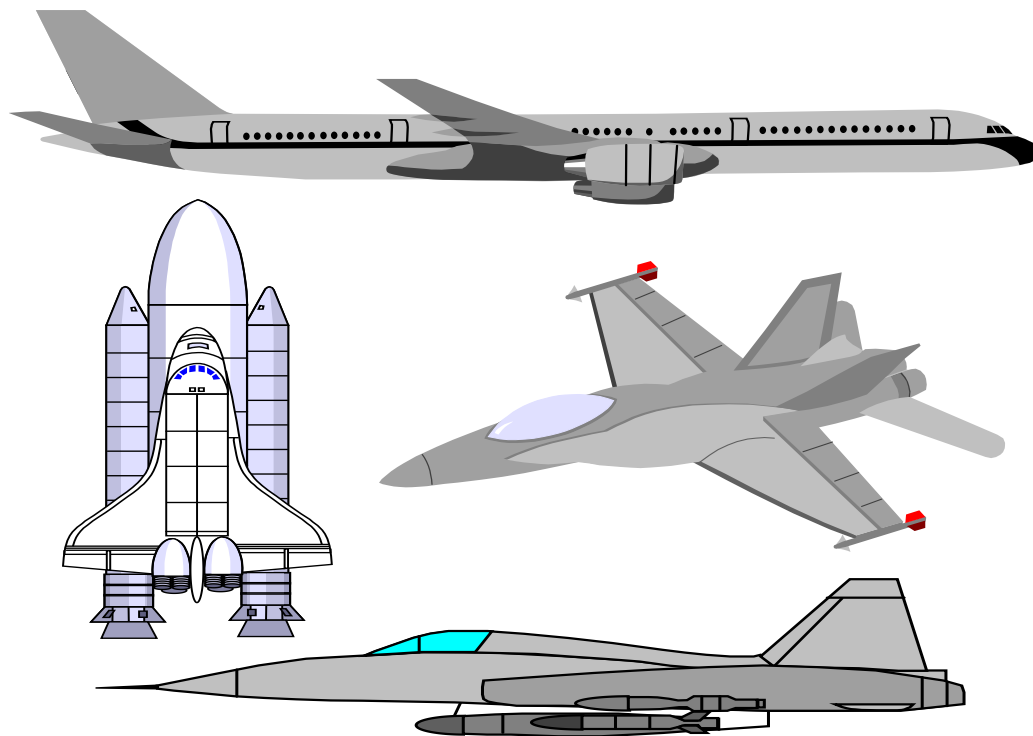
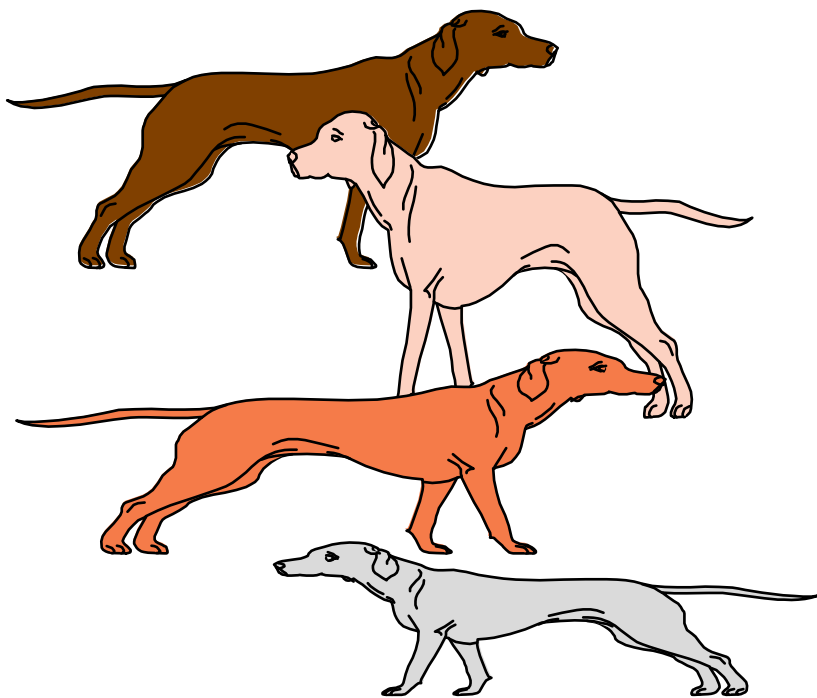




# 面向对象

两个重要概念—类

具有相同属性和操作的一组对象的集合。





# 类的示例

## Structure

Name  
Address  
Position  
Salary  
Start Date  
End Date

## Class Employee



## Behavior

Hire  
Fire  
Promote  
Increase Salary  
Retire

# 初步理解面向对象

---

- 对象
  - 属性：是什么（静态描述）
  - 方法：做什么（动态描述）

# 初步理解面向对象

---



赵继伟

- 一个人（球员）
  - 人的组成元素（属性）
    - 头、五官、四肢
    - 性别、血型、身高、名字
    - 所在球队、球衣号码
  - 人的行为（方法）
    - 吃喝
    - 跑步、跳跃
    - 运球、投篮、传球、抢断、传球



# 面向对象技术的基本观点

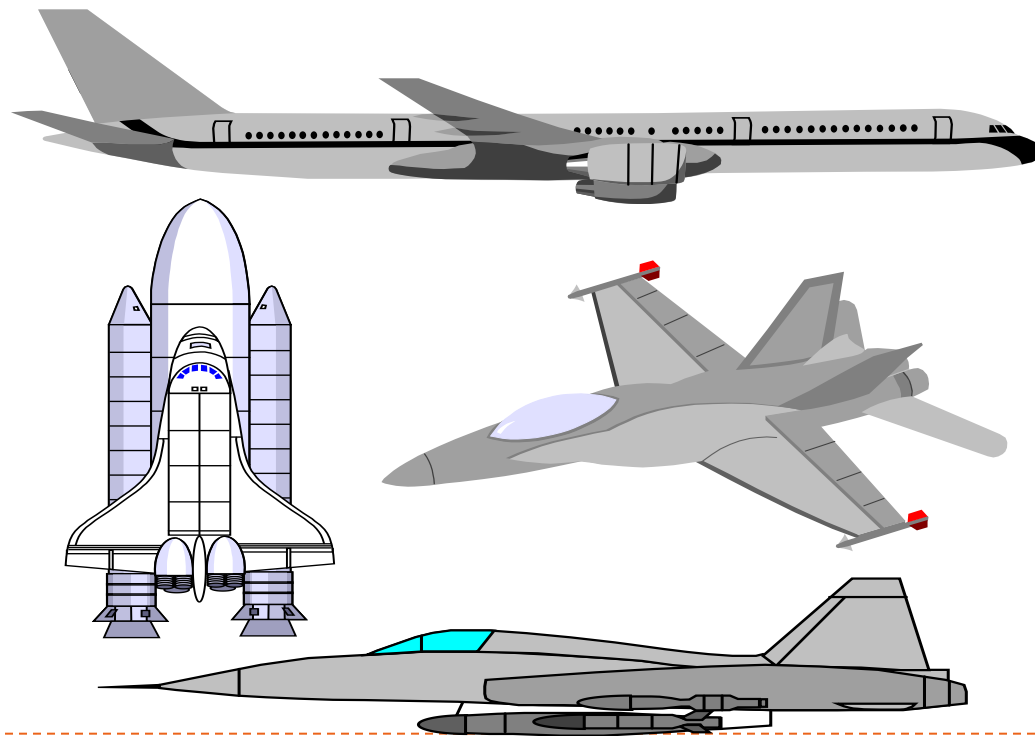
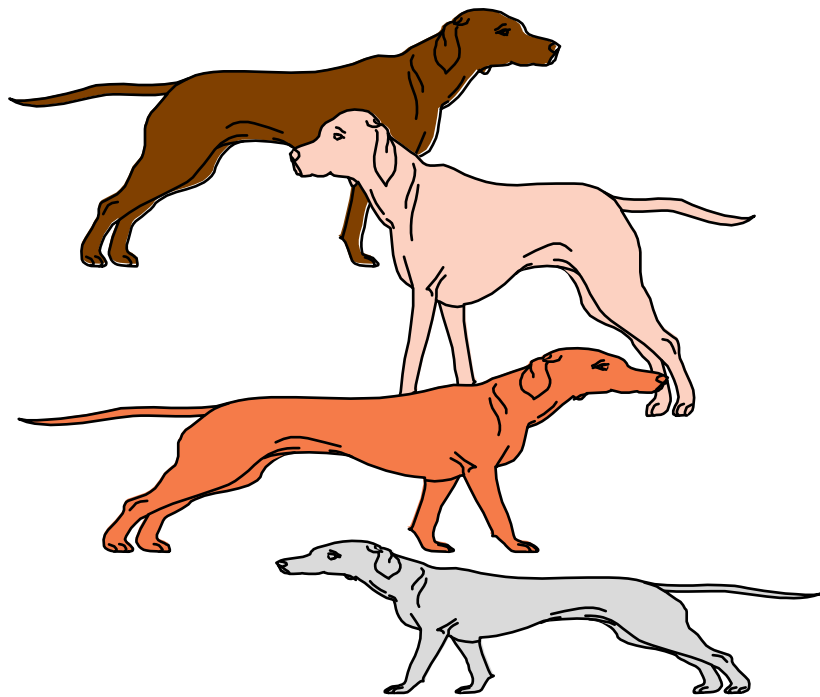
- ▶ 客观世界是由对象组成的。
- ▶ 具有相同的数据和相同的操作的对象可以归并为一类，对象是类的一个实例。类可以产生对象。
- ▶ 类可以派生子类，子类继承父类的特性。
- ▶ 对象之间通过消息传递相互联系。

软件工程师Codd和Yourdon认为：

**面向对象程序=对象+类+继承+通讯**

# 类

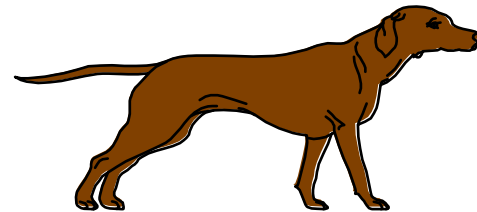
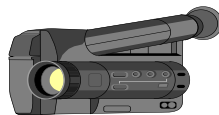
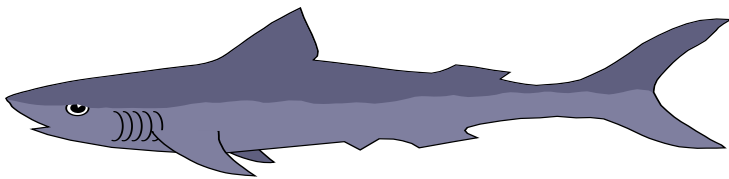
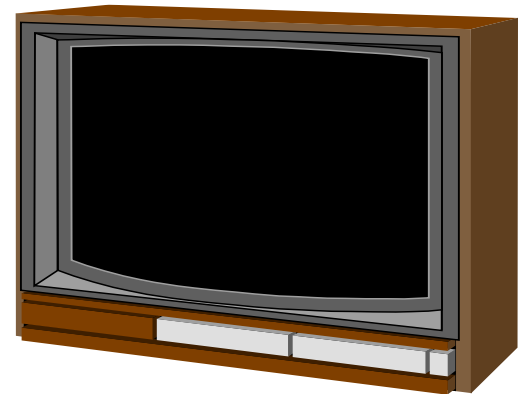
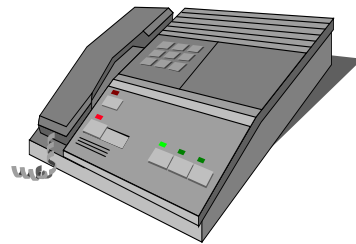
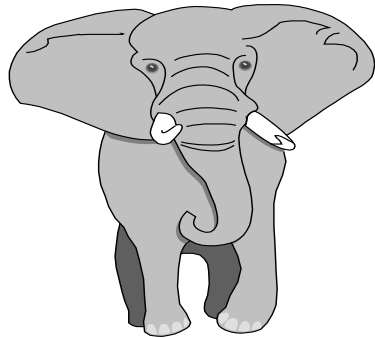
- ▶ 具有相同特征和操作的对象可以放到类中。
- ▶ 在下面图中可得到哪些类？





# 对象的类

► 可定义几个类?





# 属 性

- ▶ 属性是类的特征或特性
- ▶ 属性的值是某一特定对象的属性值
- ▶ 在类中属性名必须是唯一的
- ▶ 每一个类的实例都有为这个类定义的所有属性的值
- ▶ 例如：

银行帐户类属性

帐号  
银行名称  
拥有者  
金额

Mary的银行帐户属性值

12345678  
First National Bank  
Mary Smith  
\$1024.48



---

# 属性取决于视点

一辆汽车具有的属性：



从销售人员的角度

- ▶ 型号
- ▶ 价格
- ▶ 颜色
- ▶ 里程数

从维修人员的角度

- ▶ 马达类型
- ▶ 传动类型
- ▶ 维修记录



# 操作

- ▶ 对象的行为由为此对象定义的一系列操作决定的
- ▶ 操作访问或修改对象的属性值
- ▶ 一个类的所有实例都可使用在该类中定义的操作

# 操作取决于视点

一辆汽车具有的操作：

- ▶ 从销售人员的角度
  - ▶ 处理客户订单
  - ▶ 准备销售合同
  - ▶ 加入清单
  - ▶ 从清单中删除

- ▶ 从维修人员的角度
  - ▶ 测试刹车
  - ▶ 修理刹车
  - ▶ 转动轮胎
  - ▶ 检查马达速度



# 初步理解面向对象

---

- 面向对象的编程

将事物分解成各个对象。

对象由数据和功能组合而成；为了叙述某事物在解决问题中的步骤和行为；

在python一切皆对象（如一个函数，一个变量等）

各个对象各司其职，共同实现设计目标。

# 初步理解面向对象

---

- 面向过程的编程

分析出解决问题所需步骤；(大象冰箱)

再用函数将这些步骤一步步实现；

使用时依次调用。

---

# 简单语法

# 标识符

---

- 标识符由字母、数字、下划线组成
- 标识符可以包括英文、数字以及下划线，但不能以数字开头
- 标识符是区分大小写的
- 通常以下划线开头的标识符是有特殊意义的

---

- 以单下划线开头的代表不能直接访问的类属性

程序员使用名称前的单下划线，用于指定该名称属性为“私有”。这有点类似于惯例，为了使其他人（或你自己）使用这些代码时将会知道以“\_”开头的名称只供内部使用。正如Python文档中所述：

以下划线“\_”为前缀的名称（如\_spam）应该被视为API中非公开的部分（不管是函数、方法还是数据成员）。此时，应该将它们看作是一种实现细节，在修改它们时无需对外部通知。

代码“from <模块/包名> import \*”，那么以“\_”开头的名称都不会被导入，除非模块或包中的“\_\_all\_\_”列表显式地包含了它们（双\_）。

```
In [38]: dir(keyword)
Out[38]: ['__all__',
          '__builtins__',
          '__cached__',
          '__doc__',
          '__file__',
          '__loader__',
          '__name__',
          '__package__',
          '__spec__',
          'iskeyword',
          'kwlist']
```

\_\_all\_\_ 列表是一个用来存放可导出属性的字符串列表  
\_\_all\_\_ = ['变量1','变量2'..., '函数1','函数2'...]

用于模块导入时限制. 定义了\_\_all\_\_属性，则只有\_\_all\_\_内指定的属性、方法、类可被导入；若没定义，则导入模块内的所有公有属性，方法和类。



---

- 以双下划线开头的代表类的私有成员

只能被本类函数调用引用的成员，防止被其它地方修改。比如私人考试分数只能被此类成绩录入函数调用，其它类的其它成员比如Stu的Student成员无法碰到这个变量

为了避免与子类定义的名称冲突

- 以双下划线开头和结尾的代表python里特殊方法专用的标识

这种用法表示Python中特殊的方法名。一种惯例，对Python系统来说，这将确保不会与用户自定义的名称冲突。

# 变量

---

- 变量是用来存储一些之后可能会变化的值
- 以篮球运动员投篮为例：
  - 对一次投篮进行分析，那么我们就可以创建一个名称为 `shot_id` 的变量，表示这是某次投篮，并且将 1 值储存在变量 `shot_id` 中
  - 如果之后想要分析科比的另外一次投篮，比如投篮ID为 2 的投篮，只需要修改变量 `shot_id` 的赋值，将 `shot_id` 赋值为 2 即可

# 变量命名规则

---

- 变量名称由大小写字母、数字和下划线组成，且不以数字开头
- 对大小写敏感
- 不使用保留字（关键字）
- 单词之间用单个下划线连接
- 推荐使用有意义的单词

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

```
['False', 'None', 'True', 'and', 'as',  
'assert', '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']
```

```
In [37]: import keyword  
print(keyword.kwlist)
```

```
['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']
```

# 变量赋值

---

- 通过赋值运算符“=”变量名和想要赋予变量的值连接起来
- 同一变量可以反复赋值，而且可以是不同类型的变量，这也是Python语言称之为动态语言的原因

```
shot_id = 1
print (shot_id,type(shot_id))
shot_id = '1'
print (shot_id,type(shot_id))
```

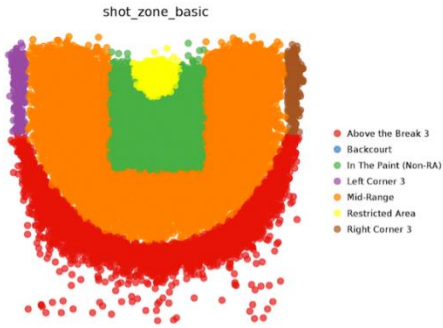
```
1 <class 'int'>
1 <class 'str'>
```

```
In [39]: shot_id = 1
          print (shot_id,type(shot_id))
          shot_id = '1'
          print (shot_id,type(shot_id))

1 <class 'int'>
1 <class 'str'>
```

变量名	含义	变量名	含义
shot_id	投篮ID	seconds_remaining	比赛剩余秒数
action_type	细分投篮类型	shot_distance	投篮离篮筐的距离
combined_shot_type	综合投篮类型，共六种： Jump Shot 跳投 Layup 上篮 Dunk 扣篮 Tip Shot 补篮 Hook Shot 勾手投篮 Bank Shot 擦板投篮	shot_made_flag	命中标记 0 未命中 1 命中
game_event_id	赛事ID	shot_type	得分类型 2PT Field Goal 2分球 3PT Field Goal 3分球
game_id	比赛ID	shot_zone_area	投篮位置（区域划分）
lat	投篮位置纬度	shot_zone_basic	投篮位置(基本)
loc_x	横向投篮位置（短边） $x$	shot_zone_range	投篮位置离框距离
loc_y	纵向投篮位置（长边） $y$	team_id	所在队伍ID
lon	投篮位置经度	team_name	所在队伍名字
minutes_remaining	比赛剩余分钟数	game_date	比赛日期
period	比赛第几节	matchup	对阵形势
playoffs	季后赛标识 0 常规赛 1 季后赛	opponent	对手
season	赛季		

变量及相关含义



# 常量

---

- 常量表示“不能变”的变量
- Python中是**没有常量的关键字的**，只是常常约定使用大写字母组合的变量名表示常量，也有不要对其进行“赋值”的提醒作用

PI = 3.14159265

# 常用的帮助方法

---

- 通过Python解释器获取帮助
- `help(object)` 帮助了解该对象的更多信息
- `dir(object)` 显示该对象的大部分相关属性名
- `object.__doc__` 显示其相对应的文档字符串(双下划线)

# help()

- `help()`函数是Python的一个内置函数,可帮助了解括号内对象的更多信息
- 函数原型: `help(object)`
- `help(math)`

```
import math  
help(math)
```

```
In [40]: import math  
         help(math)  
  
Help on built-in module math:  
  
NAME  
    math  
  
DESCRIPTION  
    This module provides access to the mathematical functions  
    defined by the C standard.  
  
FUNCTIONS  
    acos(x, /)  
        Return the arc cosine (measured in radians) of x.  
  
    acosh(x, /)  
        Return the inverse hyperbolic cosine of x.  
  
    asin(x, /)  
        Return the arc sine (measured in radians) of x.
```



# dir()

---

- **dir()**函数是Python的一个内置函数
- 函数原型：**dir(object)**
- 帮助我们获取该对象的大部分相关属性和方法

**dir(list)**

list列表

```
['__add__', '__class__', '__contains__', '__delattr__',  
 '__delitem__', '__dir__', '__doc__', '__eq__', '__format__',  
 '__ge__', '__getattribute__', '__getitem__', '__gt__',  
 '__hash__', '__iadd__', '__imul__', '__init__',  
 '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',  
 '__mul__', '__ne__', '__new__', '__reduce__',  
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',  
 '__setattr__', '__setitem__', '__sizeof__', '__str__',  
 '__subclasshook__', 'append', 'clear', 'copy', 'count',  
 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

# \_\_doc\_\_

---

- object.\_\_doc\_\_ 显示此对象从属的类型的构造函数的文档字符串
- 为模块、类、函数等添加说明性的文字，使程序易读易懂，更重要的是可以通过Python自带的标准方法将这些描述性文字信息输出
- 上面提到的自带的标准方法就是\_\_doc\_\_前后各两个下划线

```
import math  
math.__doc__
```

```
'This module is always available. It provides access to  
the\nmathematical functions defined by the C standard.'
```

- 为自定义的函数创建 \_\_doc\_\_ 的方法

```
>>> def func(): """Here's a doc string"""  
...  
>>> func.__doc__  
"Here's a doc string"
```

- 一般语言都是通过{}来标识代码块的，而在Python中，是通过缩进来识别代码块的
- 同一层次的语句必须有相同的缩进
- 缩进几个空格或者tab都是可以的，只要你保持一致就可以
- 为了别人阅读方便，常用tab或者4个空格

# 注释

---

- Python语言会通过注释符号识别出注释的部分，将它们当做纯文本，并在执行代码时跳过这些纯文本
- 在Python语言中，使用 # 进行行注释
- 如同在看书时做笔记一样，帮助理解代码
- 注释可提高程序的可维护性，撰写好的注释是好的程序员必备的技能

#这是在代码前的一个注释!

shot\_id = 1 # 这是在代码所在行的一个注释!

#这是在代码之后的一个注释!

# 注释

---

- 可以把 `''' '''` 这种多行字符串当作注释来使用

```
'''  
这是第一行注释!  
这是第二行注释!  
'''
```

```
''''  
这是第一行注释!  
这是第二行注释!  
''''
```

---

# 数字与简单运算

Python中的数字，内置函数，变量命名规则

- 数字(number)
  - 专门用于储存数值的数据类型，具有不可改变性。数据的不可改变性意味着：每改变一个数据的类型，计算机就分配内存空间以创建新的对象，解释器则基于数据的类型分配指定的内存，决定什么样的数据可以被储存在内存中。要“改变”不可改变的数据类型，只能通过创造新变量的间接方式。任何高级语言中都有数字类型的对象
- Python中的数字-Python支持4种不同的数值数据类型，如下表。

数据类型	说明	示例
int	整数	1、 10、 100
float	浮点数	15.20、 -21.9、 30.1+e18
complex	复数	3.14j、 45.j、 9322e-36j
bool	布尔值	0、 1



# 内置对象类型：数字

---

- `int`只有整数部分，即整型数。`int`仅表示广为使用的十进制整数，如果需要用到二进制数、八进制数和十六进制数，需要分别通过`bin`函数、`oct`函数和`hex`函数进行创建或转换。
- `float`是既有整数部分也有小数部分的数值类型，即浮点型数字。
- `complex`是由实部（`real`）和虚部（`imag`）组成的数值类型，即复数，复数的实部和虚部都是浮点数。
- `bool`表示布尔值，只有`True`（1）和`False`（0）两种取值。因为`bool`继承了`int`类型，所以`True`可以等价于数值1，`False`可以等价于数值0，`bool`值可以直接用于数学运算。

# 内置对象类型：数字

- 变量

- 给变量指定一个数值时，number对象就被创建，并在内存中分配储存空间。通过type函数可以判断number对象的类型

- 变量和对象之间建立了“引用”关系

变量引用对象

- 变量无类型，对象有类型

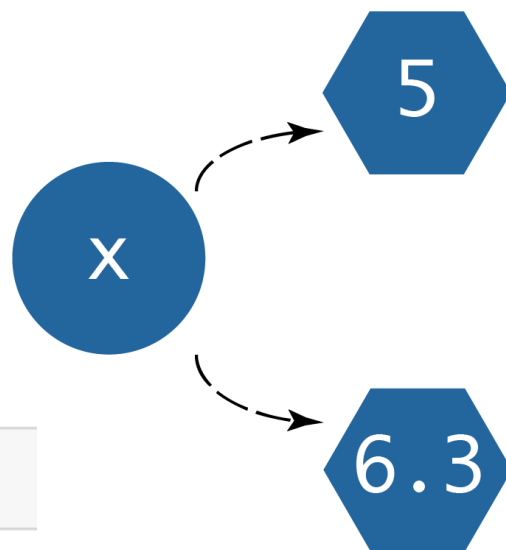
```
In [35]: x=5  
         type(x)
```

```
Out[35]: int
```

```
In [36]: x=6.3
```

```
In [34]: type(x)
```

```
Out[34]: float
```



# 整型

---

- Python语言的数据类型包括整型、浮点型、字符串、布尔型和空值
- 整型 (int)
  - 整型的取值为整数，有正有负，如 2, -666, 666 等。在篮球运动员投篮数据集集中，shot\_id、game\_event\_id、game\_id、loc\_x、loc\_y、minutes\_remaining、period、playoffs、seconds\_remaining、shot\_distance、shot\_made\_flag、team\_id都是整型变量

# 浮点型

---

- 浮点型 (**float**)

- 浮点型的取值为小数，当计算有精度要求时被使用,由于小数点可以在相应的二进制的不同位置浮动，故而称为浮点数
- 如 3.14, -6.66 等，但是如果是非常大或者非常小的浮点数，就需要使用科学计数法表示，用 e 代替 10

3.14e9

3140000000.0

```
In [31]: 3.14e9
```

```
Out[31]: 3140000000.0
```

# 内置对象类型：数字

---

- Python中的数字
- 简单运算
- 内置函数

# 内置对象类型：数字

---

- Python中的数字

- 整数型

```
type(3)
```

```
int
```

- 浮点数型

```
type(3.14)
```

```
float
```

# 内置对象类型：数字

---

- 简单运算

- 加减乘法

`a = 10`

`b = 2.5`

`a + b`

12.5

`a - b`

7.5

`a * b`

25.0

- 除法

`5 / 2`

2.5

`2 / 5`

0.4

`5 // 2` 整数除法

2

# 内置对象类型：数字

---

- 简单运算

- 取余运算

`5 % 2`

1

- 幂运算

`2 ** 3`

8



# 内置对象类型：数字

---

- 简单运算

- 运算顺序：先乘除后加减

$3.14 + 3.14 * 2 / 5 - 7$

-2.604

- 内置函数

`divmod(5, 2)` #`divmod(a,b)`返回一个包含商和余数的元组(`a // b`, `a % b`)。

(2, 1)

# 内置对象类型：数字

---

- 简单运算

- 浮点数运算

0.1 + 0.2

0.30000000000000004

```
In [55]: 0.1 + 0.2
```

```
Out[55]: 0.30000000000000004
```

0.1 + 0.1 + 0.1 - 0.3

5.551115123125783e-17

```
In [56]: 0.1 + 0.1 + 0.1 - 0.3
```

```
Out[56]: 5.551115123125783e-17
```

5.3-4.2

1.0999999999999996

```
[>>> print(5.3-4.2)
1.0999999999999996
>>>
```

# 内置对象类型：数字

---

- 简单运算

- 浮点数运算

## 为什么出现这种结果？

因为这是浮点型(float)数据，计算机对浮点数的表达本身是不精确的。

因为保存在计算机中的是二进制数，二进制对有些数字不能准确表达，只能非常接近这个数。

# 内置对象类型：数字

- 简单运算

- 浮点数运算

## 浮点数应该怎么办呢

把浮点数都同时精确到小数点某个位数来比较，比如同时精确到小数点后2位，后4位这样。

在Python中你可以使用`round()`这个函数来控制浮点数精确位数，`round()`函数是返回某个值四舍五入后的值。

### 实例

```
#!/usr/bin/python

print "round(80.23456, 2) : ", round(80.23456, 2)
print "round(100.000056, 3) : ", round(100.000056, 3)
print "round(-100.000056, 3) : ", round(-100.000056, 3)
```

以上实例运行后输出结果为：

```
round(80.23456, 2) : 80.23
round(100.000056, 3) : 100.0
round(-100.000056, 3) : -100.0
```

```
[>>> print(5.3-4.2)
1.0999999999999996
>>>
```

```
[>>> print(round((5.3-4.2), 2))
1.1
>>>
```

# 内置对象类型：数字

---

- 简单运算
  - 科学计数法

```
a = 1.2e4  
a
```

```
12000.0
```

```
In [57]: a = 1.2e4  
a
```

```
Out[57]: 12000.0
```

# 内置对象类型：数字

- 内置函数 (Built-in Functions)
  - 内置函数随Python解释器的运行而创建。可以随时调用这些函数，不需要定义
  - 内置函数网址：<https://docs.python.org/3/library/functions.html>
  - <https://www.runoob.com/python/python-built-in-functions.html>

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

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__()


# 内置对象类型：数字

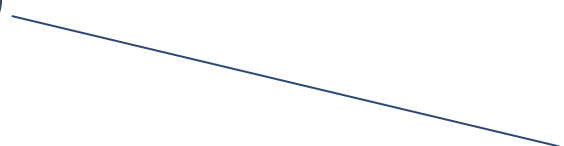
---

- 内置函数

- help()

- abs() ————— 返回数字的绝对值

- round()  返回浮点数四舍五入的值。  
如round(80.23456, 2)输出为  
小数点后两位80.23

- type()
  - id()  返回对象的唯一标识符，即  
内存地址

- ...

# 与运算相关的标准库

---

- math 模块
  - 提供了许多对浮点数的数学运算函数
- decimal 模块
  - 用于浮点数计算
  - 相比内置的二进制浮点数实现 float 这个类型有助于金融应用和其它需要精确十进制表达の場合
- fractions 模块
  - 可用于表达分数的模块



---

# 字符和字符串

字符编码，定义，转义符，基本操作

索引，切片

字符串方法

# 字符编码

- 键盘上的都是字符

- 简单的字符编码

- ASCII编码— American Standard Code for Information Interchange

美国信息互换标准代码基于拉丁字母的字符编码，共收录了 128 个字符  
用一个字节就可以存储（英语足够用了）

```
In [1]: #字符串编码  
ord("A")
```

```
Out[1]: 65
```

```
In [2]: #10进制向二进制转换  
bin(65)
```

```
Out[2]: '0b1000001'
```

```
In [3]: #一个字节最多可表示256个字符  
256*256
```

```
Out[3]: 65536
```

Bin (二进制)	Oct (八进制)	Dec (十进制)	Hex (十六进制)	缩写/字符	解释
0100 0000	0100	64	0x40	@	电子邮件符号
0100 0001	0101	65	0x41	A	大写字母A
0100 0010	0102	66	0x42	B	大写字母B

# 字符编码

- 键盘上的都是字符
- 简单的字符编码

- ASCII编码— American Standard Code for Information Interchange

美国信息互换标准代码基于拉丁字母的字符编码，共收录了 128 个字符  
用一个字节就可以存储（英语足够用了）

- 其它非英文编码：汉语中GB2312，滋生乱码
- UNICODE编码-->UTF-8（还有UTF-16 UTF-32）-实现方式

计算机科学领域里的一项业界标准，包括字符集、编码方案等。Unicode 是为了解决传统的字符编码方案的局限而产生的，它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求

```
In [4]: import sys
sys.getdefaultencoding()

Out[4]: 'utf-8'
```

```
In [1]: #字符串编码
ord("A")

Out[1]: 65
```

```
In [2]: #10进制向二进制转换
        bin(65)

Out[2]: '0b1000001'
```

```
In [3]: #一个字节最多可表示256个字符
256*256

Out[3]: 65536
```



Picture from:  
<http://archives.miloush.net/michkap/archive/2008/02/16/7735630.html>

# 字符编码

## Python中对两种utf-8格式的理解

- 1、python文件开头utf-8格式的理解
- 2、程序中读取文件时utf-8格式的理解

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

fr1 = open("goods_information", "r", encoding="utf-8")
print(fr1.read())
```

第二行代码中的utf-8代表对本文件aa.py的解码格式，第4行中的utf-8代表对读取文件goods\_information的解码格式。

另外：#!/usr/bin/python 是告诉操作系统执行这个脚本的时候，调用 /usr/bin 下的 python 解释器。#!/usr/bin/env python 这种用法是为了防止操作系统用户没有将 python 装在默认的 /usr/bin 路径里。当系统看到这一行的时候，首先会到 env 设置里查找 python 的安装路径，再调用对应路径下的解释器程序完成操作。

# 何为字符串

- 字符串（String），是由零个或多个字符组成的有限序列

- “Hello World”

- “”



这也是“串”！

Python中的字符串是存放着Unicode字符序列，用于表示文本的数据类型。str可以由任何字符构成，包括字母、数值、符号或标点符号以及它们的任意组合，如“Hello,word!”、“1+1”等。和number相同的是，Python中的str也是不可变的，无法直接修改str中的某一位字符。

创建一个str，除字符外，还要在字符序列的首尾加上引号。使用单引号（'）、双引号（"）是等效的，但需要保证str两端的引号类型相同。如果要指定一个多行的str，则需要使用三引号（'''）。

*Picture from website*

# 何为字符串

---

- 字符串的表示方式

- 1 使用单引号扩起来字符串

```
>>> 'my python lesson'      #以单引号将字符串扩起来
'my python lesson'
>>> a = 'my python lesson'
>>> print(a)
my python lesson
```


- 2 使用双引号将字符串扩起来

```
>>> "my python lesson"      #使用双引号将字符串扩起来
'my python lesson'
>>> a = "my python lesson"
>>> print(a)
my python lesson
```


# 何为字符串

- 字符串的表示方式

3 当想要输出单引号或者双引号时（将单引号，双引号作为普通字符输出），通过 \ 进行转义



```
>>> 'python \'escape'
"python 'escape"
>>> a = 'python \'escape'
>>> print(a)
python 'escape    #通过\反斜线将单引号进行转移，不在乎最外层的是单引号还是双引号，反正是中间是字符串，有\就将后面的单引号，双引号进行转义
>>>
>>> 'python \' escape'
'python " escape'
>>> a = 'python \' escape'
>>> print(a)
python " escape
>>>
>>> "python \' escape"
"python ' escape"
>>> a = "python \' escape"
>>> print(a)
python ' escape
>>>
>>> "python \' escape"
'python " escape'
>>> a = "python \' escape"
>>> print(a)
python " escape
```



# 何为字符串

- 字符串的表示方式



```
>>> "double quote"      #单引号中，使用双引号，直接将双引号输出
"double quote"
>>> a = "double quote"
>>> print(a)
double quote
>>>
>>> 'single quote'      #双引号中，使用单引号，将单引号输出
'single quote'
>>> a = 'single quote'
>>> print(a)
single quote
>>>
>>> ""double""         #双引号中直接输出双引号报错
SyntaxError: invalid syntax
>>> "\"double\""      #双引号中直接输出双引号报错，但是将其中的双引号通过反斜线进行转义就可以了
'double'
>>> a = "\"double\""
>>> print(a)
double
>>>
>>> 'single'           #单引号中输出单引号字符报错，加上转义字符就可以了。
SyntaxError: invalid syntax
>>> '\'single\''
'single'
>>> a = '\'single\''
>>> print(a)
single
>>> "My python', lession' #单引号中输出双引号可以，如有单引号，必须进行转义
SyntaxError: EOL while scanning string literal
>>> "My python\'', lession'
'My python\'', lession'
>>> a = "My python\'', lession'
>>> print(a)
My python', lession
```





# 何为字符串

---

- Python 三引号与多行注释相区别

• `s="hello`

`python "`

`" hello`

`python "`

```
In [66]: s=''' hello
python'''
```

```
In [67]: ''' hello
python'''
```

```
Out[67]: ' hello \npython'
```

(也是一种字符串，只不过运行时电脑不读取)

# 字符串基本操作

---

- 转义字符：字符串里常常存在一些如换行、制表符等有特殊含义的字符，这些字符称之为转义字符

- 比如 `\n` 表示换行，`\t` 表示制表符，Python还允许用 `r" "` 表示" " 内部的字符串默认不

**转义-保留原样** 注：`r`的全称是raw string，即原始字符串常量，可以让字符保持原来的意思。

```
print("kobe's team name is Los Angeles Lakers.")
print("kobe's team name is \t Los Angeles Lakers.")
print("kobe's team name is \n Los Angeles Lakers.")
print(r"kobe's team name is \n Los Angeles Lakers.")
```

```
kobe's team name is Los Angeles Lakers.
kobe's team name is      Los Angeles Lakers.
kobe's team name is
Los Angeles Lakers.
kobe's team name is \n Los Angeles Lakers.
```

# 转义符

➤ Python中常用的转义字符，如下表所示

转义字符	说明	转义字符	说明
\	续行符，用在行尾	\\	反斜杠符号
\'	单引号	\"	双引号
\a	响铃	\b	退格
\e	转义	\000	空
\n	换行	\v	纵向制表符
\t	横向制表符	\r	回车
\f	换页	\o	八进制数
\x	十六进制数	\other	其它字符以普通格式输出

# 字符串基本操作

- 字符串是序列

- 字符对位置敏感，有确切的顺序

```
>>> a='hello'
>>> b='world'
>>> c=a+b
>>> c
'helloworld'
```

- 基本操作

- +: 连接两个字符串

- \*: 重复元素

```
>>> a='hello'
>>> b=a*10
>>> b
'hellohellohellohellohellohellohellohellohellohello'
```

- len(): 序列的长度

- in: 判断元素是否属于序列

```
>>> b
'hellohellohellohellohellohellohellohellohellohello'
>>> len(b)
50
```

```
>>> a='hello'
>>> 'e' in a
True
```

# 索引

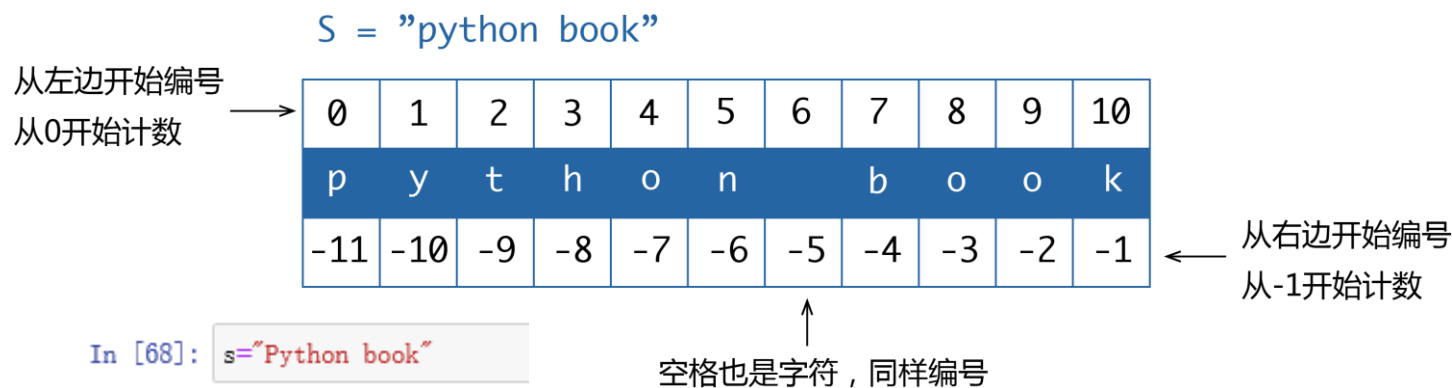
- 字符串中的字符排列有顺序(包含)
- str支持索引，索引一般按照为“变量[下标]”和“变量[头下标:尾下标]”两种格式，其中的“变量[下标]”格式能够索引单个字符，“变量[头下标:尾下标]”格式能够进行切片split（索引index连续一片元素）。

- **索引：用于标识字符顺序的整数**

- 从左边开始计数：0、1、2、.....
- 从右边开始计数：-1、-2、.....

- **字符与索引对应**

- 通过索引，可以得到相应的字符
- 使用.index()方法得到字符的索引



```
In [68]: s="Python book"
```

```
In [69]: s.index('P')
```

```
Out[69]: 0
```

```
In [70]: s.index('Py')
```

```
Out[70]: 0
```

```
In [71]: s[-11]
```

```
Out[71]: 'P'
```

# 索引—规则

---

- 下标为正数时，最小为0，表示首位，最大为总字符数减1，表示末位。
- 下标为负数时，下标最小为总字符数的相反数，表示首位。尾下标最大为-1，表示末位。
- 当进行切片时，索引从头下标位置字符开始，到尾下标位置前一位字符终止。
- 在一个索引式中，头下标与尾下标可以异号，但必须保证头下标字符位置在尾下标字符之前。
- 头下标留空，表示索引从首位开始；尾下标留空，表示索引到末位结束。

除一般的索引格式外，str还支持按步长索引，即指定步长后，每隔固定的步数索引一次字符，其格式为“变量[头下标：尾下标：步长]”；此外，通过“变量[::-1]”这一索引式可以将整个str反向排序。

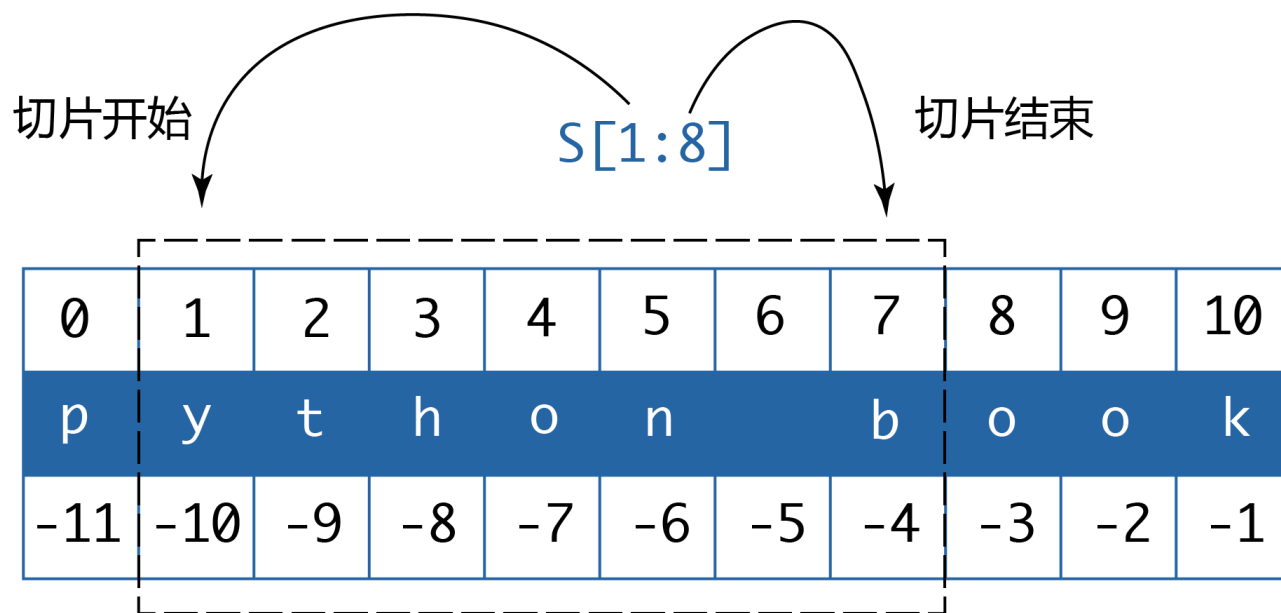
**S[index<sub>start</sub>:index<sub>stop</sub>:step]**

# 字符串切片

- 切片公式-获取字符的特定部分

$S[\text{index}_{\text{start}}:\text{index}_{\text{stop}}:\text{step}]$

- 当  $\text{step} = 1$  时，切片操作方法（默认为1）
- 左闭右开！



切片结果：“ython b”

```
In [12]: s[:8]
```

```
Out[12]: 'python b'
```

```
In [13]: s[1:]
```

```
Out[13]: 'ython book'
```

```
In [14]: s[:]
```

```
Out[14]: 'python book'
```

```
In [15]: s
```

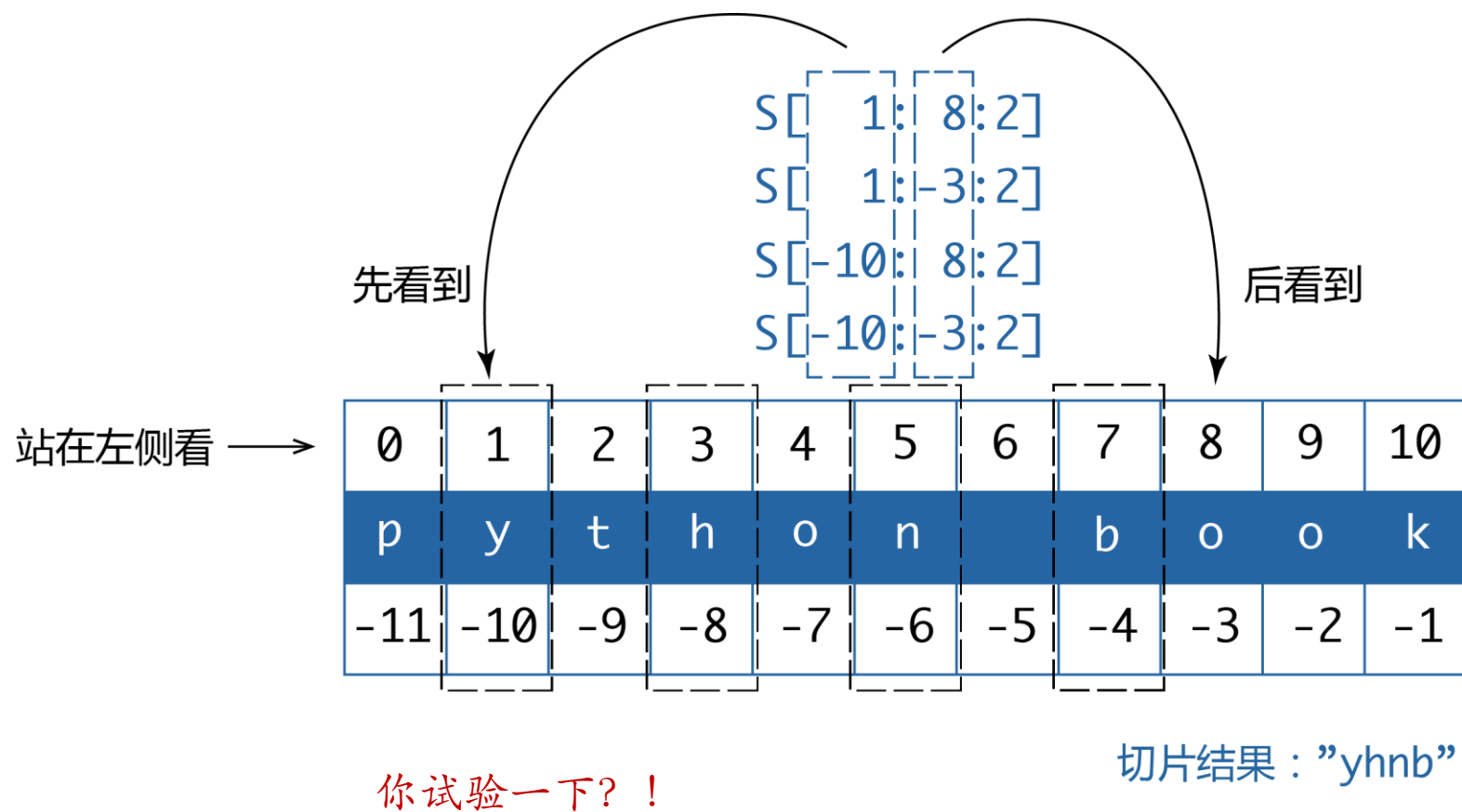
```
Out[15]: 'python book'
```

```
In [16]: s[:-3]
```

```
Out[16]: 'python b'
```

# 字符串切片

- 理解 step 为正数的含义和操作

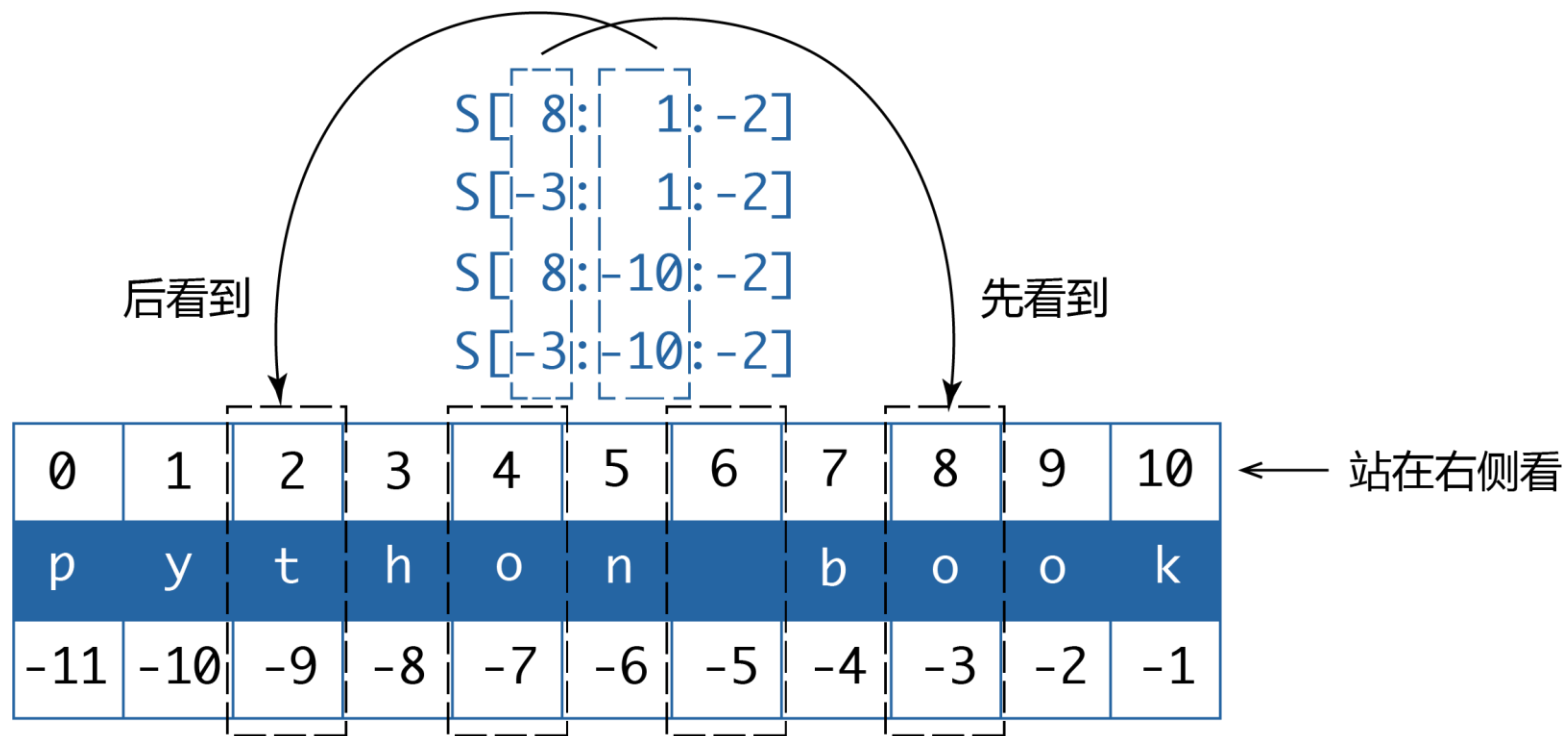




# 字符串切片

- 理解 step 为负数的含义和操作

- 左闭右开？

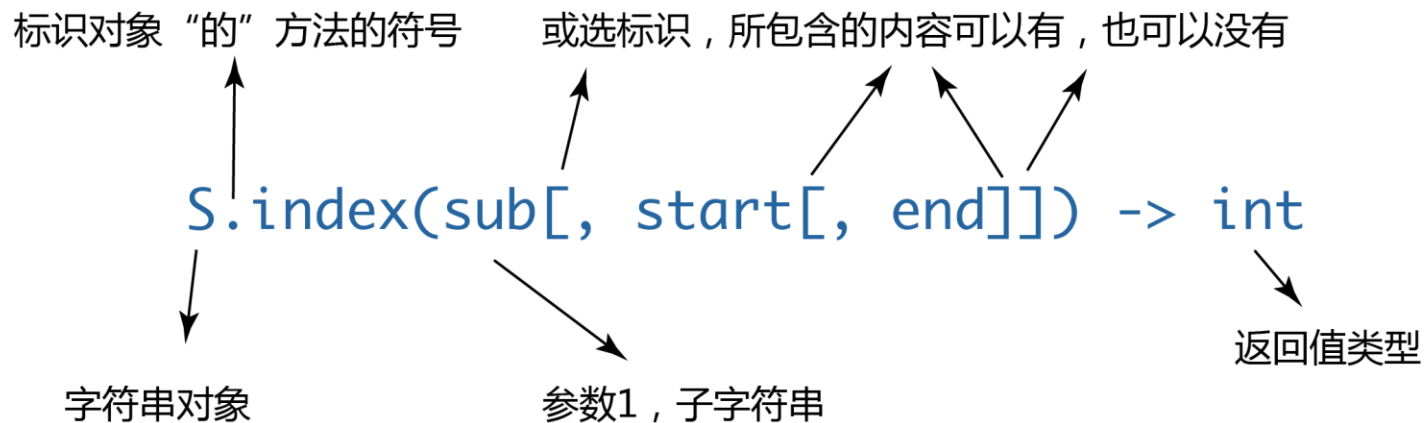


你试验一下？！

切片结果：“o ot”

# 字符串方法

- 字符串是对象类型，也是对象
- 方法调用方式



- 更多方法

- `dir()`函数查看方法列表
- `help()`函数查看方法文档

```
In [68]: s="Python book"
```

```
In [69]: s.index('P')
```

```
Out[69]: 0
```

```
In [70]: s.index('Py')
```

```
Out[70]: 0
```

# 字符串方法

---

```
>>> dir(str)
['_add_', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getat
tribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
', __len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
', __rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'e
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal',
', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rparti
tion', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfi
ll']
>>> help(str.isdigit)
Help on method_descriptor:

isdigit(self, /)
    Return True if the string is a digit string, False otherwise.

    A string is a digit string if all characters in the string are digits and there
    is at least one character in the string.
```

# 字符串方法

- 查询功能的str方法
- str的索引方式对Python中其他支持索引的数据类型都具有参考价值。

Python为str提供了极为丰富的方法，使str具备了极佳的可操作性。str的方法中部分是用于查询的，这些方法包括检查str中是否包含某个对象，是否只包含某个对象，某个对象是否在特定的位置，以及包含某个对象的个数。提供查询功能的str方法，如下表所示。

方法	说明	方法	说明
string.find()	检查str中是否包含某字符	string.isdecimal()	检查str中是否只包含十进制数字
string.rfind()	检查str中是否包含某字符，从右侧开始检索	string.isdigit()	检查str中是否只包含数字
string.index()	检查str中是否包含某字符	string.isnumeric()	检查str中是否只包含数字字符
string.rindex()	检查str中是否包含某字符，从右侧开始检索	string.isspace()	检查str中是否只包含空格
string.isalnum()	检查str中是否为全字母或数字	string.startswith()	检查str是否以指定对象开头
string.isalpha()	检查str中是否只包含字母	string.endswith()	检查str是否以指定字符结尾
string.islower()	检查str中是否只包含小写字母	string.isidentifier()	检查str是否标题化
string.isupper()	检查str中是否只包含大写字母	string.count()	计算指定字符出现的次数

```
[>>> a = "python top"
[>>> a.find("p")
0
[>>> a.find("top")
7
[>>> a.find("py")
0
[>>> a.find("o")
4
[>>> string.endswith("o")
False
[>>> a.rfind("o")
8
```

# 字符串方法

- 查询功能的str方法
- str的索引方式对Python中其他支持索引的数据类型都具有参考价值。

Python为str提供了极为丰富的方法，使str具备了极佳的可操作性。str的方法中部分是用于查询的，这些方法包括检查str中是否包含某个对象，是否只包含某个对象，某个对象是否在特定的位置，以及包含某个对象的个数。提供查询功能的str方法，如下表所示。

方法	说明	方法	说明
string.find()	检查str中是否包含某字符	string.isdecimal()	检查str中的是否只包含十进制数字
string.rfind()	检查str中是否包含某字符，从右侧开始检索	string.isdigit()	检查str中是否只包含数字
	右侧	string.isnumeric()	检查str中是否只包含数字字符
		string.isspace()	检查str中是否只包含空格
		string.startswith()	检查str是否是以指定对象开头
		string.endswith()	检查str是否以指定字符结尾
string.islower()	检查str中是否只包含小写字母	string.istitle()	检查str是否标题化
string.isupper()	检查str中是否只包含大写字母	string.count()	计算指定字符出现的次数

# 字符串方法

➤ 对str中的字母，进行大小写调整或替换，如下表所示。

方法	说明	方法	说明
string.strip()	删除str首尾空格	string.expandtabs( )	将str中的制表符转为空格
string.lstrip()	删除str左边的空格	string.upper()	转换str中的小写字母为大写
string.rstrip()	删除str末尾的空格	string.lower()	转换str中的大写字符为小写
string.ljust()	str左对齐，空格填充至指定长度	string.swapcase()	翻转str中的大小写
string.rjust()	str右对齐，空格填充至指定长度	string.capitalize()	将str中首字符大写
string.zfill()	右对齐指定长度的str，前面填充0	string.title()	返回标题化的str
string.center()	使str居中，用空格填充至指定长度	string.replace()	替换str中的字符

# 字符串方法

➤ str方法除提供了查询与改写功能外，还提供了一些其他功能，如下表所示。

方法	说明	方法	说明
string.encode()	以指定编码格式编码str	string.splitlines()	按行分隔，返回以各行作为元素的序列
string.decode()	以指定编码格式解码str	string.join()	以str为分隔合并序列为新的str
string.partition()	以指定分隔符分割str，仅分割一次，且包含分隔符	string.split()	以指定字符为分隔切片str
string.rpartition()	以指定分隔符分割str，右起	string.format()	格式化str

---

# 用户输入和输出

键盘输入，格式化输出



# 用户输入

---

- 大多数程序都旨在解决终端用户的问题，为此，需要从用户那里获取一些信息
- 函数()让程序暂停运行，等待用户的输入。input函数在用于交互式的信息键入，相当于一个容器，用户从键盘输入的信息先存放在容器中，再被变量引用。input函数可以接纳多种数据类型，包括number、str等基础变量及list、tuple、dict、set等高级变量。  
使用input函数时，可以在括号内添加str以提示输入
- 使用函数()时，Python将用户输入解读为字符串，此时需要使用函数int()将数字的字符串转换为数值表示

```
name = input("Please input your name:")
```

```
Please input your name:Li xiaodong
```

# 键盘输入

---

- 内置函数: input()
- 举例: 编写一段程序, 实现如下功能
  - 询问姓名和年龄;
  - 计算10年之后的年龄;
  - 打印出所输入的姓名和当前年龄、10年之后的年龄。

```
7  name = input("what is your name?")
8  age = input("How old are you?")
9
10 after_ten = int(age) + 10
11
12 print("-" * 20)
13 print("Your name is: ", name)
14 print("You are " + age + " years old.")
15 print("You will be " + str(after_ten) + " years old after ten years.")
```

# 用户输出

---

- 输出的最简单方法是使用 `print()`
- 可以通过用逗号分隔零个或多个表达式，`print`函数会依次打印每个`str`，遇到逗号则输出一个空格，因此输出的`str`是拼起来的；`print`函数也可以自动计算结果，运行“`print(number1+number2)`”语句，解释器会自动计算出相加的结果后输出。
- 举例：
  - 这个函数传递表达式转换为一个字符串，如下结果

```
print ("Python is really a great language,", "isn't it?")
```

```
Python is really a great language, isn't it?
```

# 格式化输出

- 格式化输出是计算机输出中的一个重要概念，主要针对str，其运行机制为：使用占位符在str中进行占位，再用数值或字符替换占位符，重组str后输出。这种输出方法主要是为了方便修改语句，减少编写代码的工作量，并且包含自动取位、转换进制等功能。Python中的格式化输出方法有两种，即“%+格式符”的方法和format函数方法。
- %+格式符字符串格式化时百分号后面有不同的格式符号，代表要转换的不同类型，具体的表示符号如下：

输出类型	格式符	作用	输出类型	格式符	作用
str	%s	字符串（采用str函数的显示）	整数	%b	二进制整数
	%r	字符串（采用repr函数的显示）		%d	十进制整数
	%c	单个字符		%i	十进制整数
float	%e	指数（基底写为e）		%o	八进制整数
	%E	指数（基底写为E）		%x	十六进制整数
	%f	浮点数			
	%F	浮点数			
	%g	指数（e）或浮点数（根据显示长度）			

---

```
1 >>> from datetime import datetime
2 >>> now = datetime.now()
3 >>> print(str(now))
4 2017-04-22 15:41:33.012917
5 >>> print(repr(now))
6 datetime.datetime(2017, 4, 22, 15, 41, 33, 12917)
```

通过 `str()` 的输出结果我们能很好地知道 `now` 实例的内容，但是却丢失了 `now` 实例的数据类型信息。而通过 `repr()` 的输出结果我们不仅能获得 `now` 实例的内容，还能知道 `now` 是 `datetime.datetime` 对象的实例。

因此 `str()` 与 `repr()` 的不同在于：

- `str()` 的输出追求可读性，输出格式要便于理解，适合用于输出内容到用户终端。
- `repr()` 的输出追求明确性，除了对象内容，还需要展示出对象的数据类型信息，适合开发和调试阶段使用。

# 格式化输出

- 使用.format()方法

- format函数是更为强大的格式化输出工具，format函数收集位置参数和关键字参数的任意集合，使用它们的值替换str中的占位符。该方法使用大括号（{}）作为特殊字符代替%，{}中可以不带参数、带数字编号或带关键字编号进行占位和替换，前两种属于位置替换方法，后一种属于关键字替换方法。
- format函数也支持格式符，如下表所示。

格式符	说明	格式符	说明
'c'	字符。打印前将整数转换成对应的Unicode字符串	'e'	幂符号。用科学计数法打印数字。用'e'表示幂
'b'	二进制。将数字以2为基数进行输出	'g'	一般格式。将数值以fixed-point格式输出，数值特别大时用幂形式打印
'o'	八进制。将数字以8为基数进行输出	'n'	数字。值为整数则等效于'd'，为float则等效于'g'。
'd'	十进制。将数字以10为基数进行输出	'%'	百分数。数值乘以100后以fixed-point('f')格式打印，值后有一个百分号
'x'	十六进制。将数字以16为基数进行输出，9以上的位数用小写字母		

# 格式化输出

- 使用`.format()`方法

占位符

参数，依次编号为 0 1

```
"I like {0} and {1}".format("python", "physics")
```

返回值："I like python and physics"

- 使用`%s, %d`占位

按顺序填入

```
"I like %s and %s" % ("python", "physics")
```

返回值："I like python and physics"

- `f`前缀的字符串

```
>>> account = '测试工程师小站'
>>> month = '30'
>>> f'我的微信公众号是：{account}，已经连续发文{int(month) * 5}天啦！'
'我的微信公众号是：测试工程师小站，已经连续发文150天啦！'
```

Python3.6新加特性，前缀`f`用来格式化字符串。可以看出`f`前缀可以更方便的格式化字符串,比`format()`方法可读性高且使用方便。而且加上`f`前缀后,支持在大括号内,运行Python表达式。

# 格式化输出format

---

```
>>>"{} {}".format("hello", "world")    # 不设置指定位置，按默认顺序  
'hello world'
```

```
"I like {0} and {1}".format("python", "physics")
```

```
'I like python and physics'
```

```
"I like {1} and {0}".format("python", "physics")
```

```
'I like physics and python'
```

```
"I like {1} and {0}. {0} is a programming language".format("python", "physics")
```

```
'I like physics and python. python is a programming language'
```



# 格式化输出format

```
In [30]: "She is {0:d} years old and {1:f} in height.".format(28, 1.68)
```

```
Out[30]: 'She is 28 years old and 1.680000 in height.'
```

```
In [31]: "She is {0:4d} years old and {1:.1f} in height.".format(28, 1.68)
```

```
Out[31]: 'She is    28 years old and 1.7 in height.'
```

```
In [32]: "She is {0:04d} years old and {1:06.1f} in height.".format(28, 1.68)
```

```
Out[32]: 'She is 0028 years old and 0001.7 in height.'
```

```
In [33]: "I like {subject} and {lang}.".format(lang="python", subject="physics")
```

```
Out[33]: 'I like physics and python'
```

1位浮点数

0填充4位,  
0填充6位, 1位  
小数

^, <, > 分别是居中、左对齐、右对齐, 后面带宽度, : 号后面带填充的字符, 只能是一个字符, 不指定则默认是用空格填充。

+ 表示在正数前显示 +, 负数前显示 -; (空格) 表示在正数前加空格

b、d、o、x 分别是二进制、十进制、八进制、十六进制。

此外我们可以使用大括号 {} 来转义大括号, 如下实例:

# 格式化输出format

数字	格式	输出	描述
3.1415926	{:.2f}	3.14	保留小数点后两位
3.1415926	{:+.2f}	+3.14	带符号保留小数点后两位
-1	{:+.2f}	-1.00	带符号保留小数点后两位
2.71828	{:.0f}	3	不带小数
5	{:0>2d}	05	数字补零 (填充左边, 宽度为2)
5	{:x<4d}	5xxx	数字补x (填充右边, 宽度为4)
10	{:x<4d}	10xx	数字补x (填充右边, 宽度为4)
1000000	{:,}	1,000,000	以逗号分隔的数字格式
0.25	{:.2%}	25.00%	百分比格式
1000000000	{:.2e}	1.00e+09	指数记法

**^**, **<**, **>** 分别是居中、左对齐、右对齐, 后面带宽度, **:** 号后面带填充的字符, 只能是一个字符, 不指定则默认是用空格填充。

**+** 表示在正数前显示 **+**, 负数前显示 **-**;  (空格) 表示在正数前加空格

b、d、o、x 分别是二进制、十进制、八进制、十六进制。

此外我们可以使用大括号 **{ }** 来转义大括号, 如下实例:

# 格式化输出%s %d 占位符

```
# 打印字符串
print("This is %s Python Test"%("Lily's"))

# 打印整数
print("%d*d=%d" %(1,5,5))

# 打印浮点数
print("%f"%12.2345)

# 打印浮点数（指定保留小数点位数）
print("%.2f"%12.2345)

# 指定占位符宽度（左对齐）
print("This is %s Python Test"%("Lily's"))
print("This is %-20s Python Test"%("Lily's"))

# 指定占位符（只能用0当占位符）
print("My Name is%10s,Height:%5f,Age:%3d"%("Lily",168.0,27))
print("My Name is%010s,Height:%010.2f,Age:%08d"%("Lily",168.0,27))
```

```
>>> print("my name is%10s,heigh:%5f,Age:%3d"%("Lily", 168.0,27))
my name is      Lily,high:168.000000,Age: 27
>>> print("my name is%010s,heigh:%010.2f,Age:%08d"%("Lily", 168.0,27))
my name is      Lily,high:0000168.00,Age:00000027
>>>
```

《Python基础》

## 1. 打印字符串

```
print ("His name is %s"%("Aviad"))
```

结果：His name is Aviad

## 2. 打印整数

```
print ("He is %d years old"%(25))
```

结果：He is 25 years old

## 3. 打印浮点数

```
print ("His height is %f m"%(1.83))
```

结果：His height is 1.830000 m

## 4. 打印浮点数（指定保留小数点位数）

```
print ("His height is %.2f m"%(1.83))
```

结果：His height is 1.83 m

## 5. 指定占位符宽度

```
print ("Name:%10s Age:%8d Height:%8.2f"%("Aviad",25,1.83))
```

结果：Name: Aviad Age: 25 Height: 1.83

## 6. 指定占位符宽度（左对齐）

```
print ("Name:%-10s Age:%-8d Height:%-8.2f"%("Aviad",25,1.83))
```

结果：Name:Aviad Age:25 Height:1.83

## 7. 指定占位符（只能用0当占位符？）

```
print ("Name:%-10s Age:%08d Height:%08.2f"%("Aviad",25,1.83))
```

结果：Name:Aviad Age:00000025 Height:00001.83

# 格式化输出f前缀的字符串

---

```
In [34]: name = "laoqi"
         age = 28
         f"My name is {name}, I am {age} years old."
```

```
Out[34]: 'My name is laoqi, I am 28 years old.'
```

```
In [35]: f"My name is {name.upper()}, I am {age + 30} years old."
```

```
Out[35]: 'My name is LAOQI, I am 58 years old.'
```

---

# 列表

创建，切片，基本操作和常用方法  
浅拷贝和深拷贝

# 列表

- 定义列表的基本方法

- list可以用方括号[]创建：空的方括号创建空的list；包含多个项的list可以在方括号中使用逗号分隔的项序列创建。
- list列表属于序列类数据，是包含0或多个对象引用的有序序列。由于list中所有的数据项都是对象引用，因此list可以存放任意数据类型的数据项，既可以是int、float、str等这种基础数据类型，也可以是list、tuple、dict等这一类的高级数据类型。list是Python中最通用的复合数据类型。列表是个筐，什么都能装（所有python中的对象都可以）

- 列表的索引和切片

- 列表是一个有序的序列结构
- 序列中的元素可以是不同的数据类型
- 索引和切片方法与字符串相同

```
In [7]: lst = [1, 2.2, "python", [], [1, 2, 3]]
```

```
In [8]: type(lst)
```

```
Out[8]: list
```

- 加、乘和检查成员等

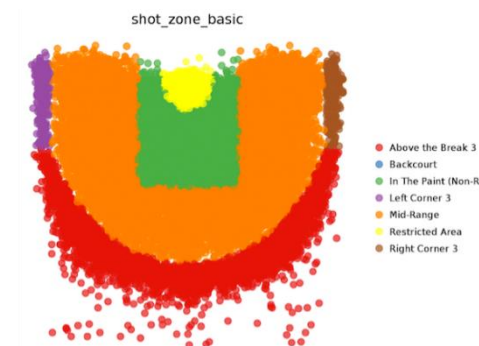
# 创建列表

- 将列表中的各元素用逗号分隔开，并用中括号将所有元素包裹起来

```
kobe_list = [2,'Jump Shot','Los Angeles Lakers']  
print(kobe_list)  
print(type(kobe_list))
```

```
[2, 'Jump Shot', 'Los Angeles Lakers']  
<class 'list'>
```

科比投篮数据集



# 列表切片

---

- 切片操作需要提供起始索引位置和最后索引位置，然后用冒号：将两者分开

列表名称[起始索引位置: 最后索引位置: 步长]

- 如果未输入步长，则默认步长为 1
- 切片操作返回一系列从起始索引位置开始到最后索引位置结束的数据元素
- 起始索引位置的值包含在返回结果中，而最后索引位置的值不包含在返回结果中

左闭右开

```
In [27]: lst = ["a", "b", "c", "d", "e"]
```

```
In [33]: lst[1:3]
```

```
Out[33]: ['b', 'c']
```

```
In [ ]: lst
```

```
In [34]: lst[-4: 3]
```

```
Out[34]: ['b', 'c']
```



# 列表切片

---

- 逆向切片：
  - 省略起始索引位置，表示从最开始进行切片
  - 当将两个索引都省略之后，将按原样复制一个列表
  - 如果想要将列表的**顺序颠倒(逆序)**，则可以使用`::-1`

```
In [27]: lst = ["a", "b", "c", "d", "e"]
```

```
In [35]: lst[: 3]
```

```
Out[35]: ['a', 'b', 'c']
```

```
In [ ]: lst[1:]
```

```
In [37]: lst[::2]
```

```
Out[37]: ['a', 'c', 'e']
```

```
In [36]: lst[::-1]
```

```
Out[36]: ['e', 'd', 'c', 'b', 'a']
```

# 基本操作和常用方法

## • 基本操作

- 序列的基本操作适用于列表

## • 常用方法

- 增加列表的元素：append、extend、insert
- 删除列表的元素：remove、pop、clear
- 元素排序：sort、sorted
- 元素翻转：reverse、reversed

都是原地修改！

```
[>>> l = ['c', 'o', 'o', 'k']
[>>> " ".join(l)
'c o o k'
[>>> "".join(l)
'cook'
[>>>
```

## 列表

`l = ['c', 'o', 'o', 'k', 1]` | 创建一个包含字符元素c、o、o、k和整数1的列表.

`list()` | 创建空列表，或将其他数据结构转换为列表.

`l[0]` | 返回列表的第一个元素，即字符c.

`l[-1]` | 返回列表的最后一个元素，即1.

`l[1:3]` | 列表切片，返回包含原列表的第二个元素和第三个元素的列表[ 'o', 'o' ].

`len(l)` | 返回列表的元素个数.

`l[::-1]` | 将列表进行逆序排列.

`l.reverse()` | 将列表进行逆序排列.

`l.insert(1, 'b')` | 在指定的索引位置插入 'b' .

`l.append()` | 在列表末尾添加元素.

`l.extend(L)` | 等价于“`l+L`”，将列表L中的元素依次添加到l的末尾.

`l.remove()` | 删除列表中的某个元素.

`l.pop()` | 等价于“`del l[]`”，删除列表中对应索引位置的元素.

`"".join(['c', 'o', 'o', 'k'])` | 将列表中的各个字符串元素用空格连接起来并转换为字符串，返回结果为c o o k.

“”不包含空格则为cook!

# 列表的基本操作

- 使用append()方法添加元素，该方法会在列表末尾位置添加数据元素（一次只添加一个元素）
- 使用remove()方法删除元素

```
kobe_list = [2,'Jump Shot','Los Angeles Lakers']
```

```
print(kobe_list)
```

```
kobe_list.append('POR')
```

```
print(kobe_list)
```

```
[2, 'Jump Shot', 'Los Angeles Lakers']
```

```
[2, 'Jump Shot', 'Los Angeles Lakers', 'POR']
```

```
kobe_list.remove(2)
```

```
print(kobe_list)
```

```
['Jump Shot', 'Los Angeles Lakers', 'POR']
```

```
In [2]: kobe_list = [2,'Jump Shot','Los Angeles Lakers']  
print(kobe_list)  
kobe_list.append('POR')  
print(kobe_list)
```

```
[2, 'Jump Shot', 'Los Angeles Lakers']  
[2, 'Jump Shot', 'Los Angeles Lakers', 'POR']
```

```
In [3]: kobe_list.remove(2)  
print(kobe_list)
```

```
['Jump Shot', 'Los Angeles Lakers', 'POR']
```

```
In [5]: kobe_list.remove('POR')  
print(kobe_list)
```

```
['Jump Shot', 'Los Angeles Lakers']
```

# 列表对象的增减

- `remove()`方法适用于知道要删除的值的值的情况，当我们不知道具体元素值，但是知道元素的索引位置时，我们可以使用 `del` 函数配合列表索引，删除索引位置的元素或者使用 `pop()`方法

```
kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']  
del kobe_list[-2]  
print(kobe_list)  
kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']  
kobe_list.pop(-2)  
print(kobe_list)
```

```
[2, 'Jump Shot', 'POR']  
[2, 'Jump Shot', 'POR']
```

```
In [7]: kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']  
del kobe_list[-2]  
print(kobe_list)  
kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']  
kobe_list.pop(-2)  
print(kobe_list)  
  
[2, 'Jump Shot', 'POR']  
[2, 'Jump Shot', 'POR']
```

# 列表对象的增减-20210916

- 通过 insert()方法在指定的索引位置前添加数据元素

```
print(kobe_list)
kobe_list.insert(1,'Los Angeles Lakers')
print(kobe_list)
```

```
[2, 'Jump Shot', 'POR']
[2, 'Los Angeles Lakers', 'Jump Shot', 'POR']
```

```
In [8]: print(kobe_list)
kobe_list.insert(1,'Los Angeles Lakers')
print(kobe_list)

[2, 'Jump Shot', 'POR']
[2, 'Los Angeles Lakers', 'Jump Shot', 'POR']
```

```
help(list.insert)
```

Help on method\_descriptor:

```
insert(self, index, object, /)
    Insert object before index.
```

```
In [17]: sample = ['1325', 'City Hospital', 0.67]
```

```
In [18]: sample.insert(-2, '2021/09/16')
```

```
In [19]: sample?
```

### 【正数索引】

```
1 x = [10, 34, 45, 64, 100]
2 x.insert(1,0) #索引位1之前插入0
3 x
```

结果为：[10, 0, 34, 45, 64, 100]

看起来，新插入的0恰好在位置1上。

### 【负数索引】

```
1 x = [10, 34, 45, 64, 100]
2 x.insert(-1,0) #在最后一个元素之前插入0
3 x
```

结果为：[10, 34, 45, 64, 0, 100]

新插入的0在最后一个元素之前，插入后为倒数第二个元素。

```
In [17]: sample = ['1325', 'City Hospital', 0.67]
```

```
In [18]: sample.insert(-2, '2021/09/16')
```

```
In [19]: sample?
```

```
Out[19]: ['1325', '2021/09/16', 'City Hospital', 0.67]
```

# 列表长度

---

- Python内置的用于判断列表长度的函数为 len()

```
kobe_list = ['Los Angeles Lakers','LAL @ POR','Jump Shot','POR','Left Side(L)','2000-10-31']  
print(len(kobe_list))
```

6

## 列表合并

```
Print( len([2, 3, 5] + [3, 6, 8]))
```

```
In [32]: print([2, 3, 5] + [3, 6, 8])
```

```
[2, 3, 5, 3, 6, 8]
```

```
In [34]: print(len([2, 3, 5] + [3, 6, 8]))
```

```
6
```

```
In [35]: print([2, 3, 5] + [3, 6, 8, 9])
```

```
[2, 3, 5, 3, 6, 8, 9]
```

```
In [36]: print(len([2, 3, 5] + [3, 6, 8, 9]))
```

```
7
```

# 列表长度

---

- 列表中的元素也可以是列表，这样可以将列表看成更高维的数组

```
nested_kobe_list = [['Los Angeles Lakers', 'LAL @ POR'], 'Jump Shot']  
print(nested_kobe_list[0][0])
```

```
Los Angeles Lakers
```

- 拆开很容易理解

```
team_info = ['Los Angeles Lakers', 'LAL @ POR']  
nested_kobe_list = [team_info, 'Jump Shot']  
print(nested_kobe_list[0][0])  
print(team_info[0])
```

```
Los Angeles Lakers  
Los Angeles Lakers
```



# 比较列表和字符串

- 列表是可变对象!! 字符串是不可变对象

- 列表 (l) 和字符串 (s) 之间的转化

- list(s), str(l)
- s.split(), "".join(l)

```
In [20]: a = "python"
          lst = list(a)
          lst
```

```
Out[20]: ['p', 'y', 't', 'h', 'o', 'n']
```

```
In [21]: s = "i love python"
          s_lst = s.split()
          s_lst
```

```
Out[21]: ['i', 'love', 'python']
```

```
In [22]: str(lst)
```

```
Out[22]: "['p', 'y', 't', 'h', 'o', 'n']"
```

```
In [23]: "".join(lst)
```

```
Out[23]: 'python'
```

## ””.join(l)

```
In [5]: lst=["C","O","O","K"]  
"".join(lst)
```

```
Out[5]: 'COOK'
```

```
In [6]: ':'.join(lst)
```

```
Out[6]: 'C:O:O:K'
```

```
In [7]: "".join(lst)
```

```
Out[7]: 'COOK'
```

```
In [8]: " ".join(lst)
```

```
Out[8]: 'C O O K'
```

```
In [9]: '-'.join(lst)
```

```
Out[9]: 'C-O-O-K'
```

```
In [10]: "-".join(lst)
```

```
Out[10]: 'C-O-O-K'
```

```
In [11]: "*".join(lst)
```

```
Out[11]: 'C*O*O*K'
```

### #对字符串进行操作

```
>>> seq2 = "hello good boy doido"
```

```
>>> print ':'.join(seq2)
```

```
h:e:l:l:o: :g:o:o:d: :b:o:y: :d:o:i:i:d:o
```

### #对元组进行操作

```
>>> seq3 = ('hello','good','boy','doido')
```

```
>>> print ':'.join(seq3)
```

```
hello:good:boy:doido
```

### #对字典进行操作

```
>>> print ':'.join(s)  
hello:good:boy:doido  
>>> s = {'hello':1,'good':2,'boy':3,'doido':4}  
>>> print ':'.join(s)  
hello:good:boy:doido  
>>>
```

### #合并目录

```
>>> import os
```

```
>>> os.path.join('/hello/','good/boy/','doido')
```

```
 '/hello/good/boy/doido'
```

# 浅拷贝和深拷贝

- 对对象赋值实际上是对对象的引用
- 当创建一个对象，然后把它赋给另一个变量的时候，Python并没有拷贝这个对象，而只是拷贝了这个对象的引用
- 直接赋值,传递对象的引用而已,原始列表改变，被赋值的b也会做相同的改变

```
alist = [1,2,3,['a','b']]  
b = alist  
print(b)  
alist.append(5)  
print(alist)  
print(b)
```

```
[1, 2, 3, ['a', 'b']]  
[1, 2, 3, ['a', 'b'], 5]  
[1, 2, 3, ['a', 'b'], 5]
```

```
In [7]: a=[1, 2, 3]  
  
In [8]: b=a  
  
In [9]: print(id(a), id(b))  
2397557259584 2397557259584  
  
In [10]: a=[4, 5, 6, 7]  
  
In [12]: print(id(a), id(b))  
2397556990464 2397557259584  
  
In [13]: print(b)  
[1, 2, 3]
```

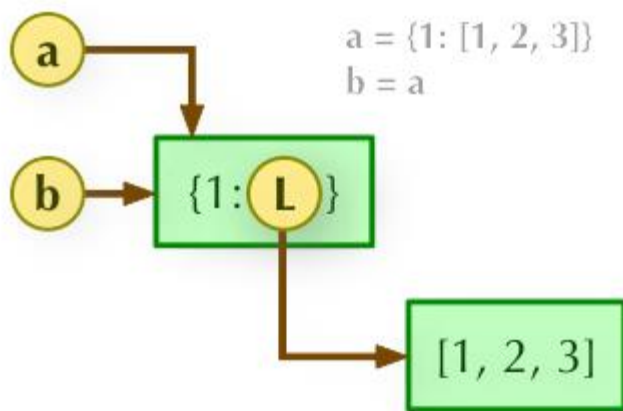
# 浅拷贝和深拷贝

浅拷贝和深拷贝的不同仅仅是对组合对象来说，所谓的组合对象就是包含了其它对象的对象，如列表，类实例。而对于数字、字符串以及其它“原子”类型，没有拷贝一说，产生的都是原对象的引用。

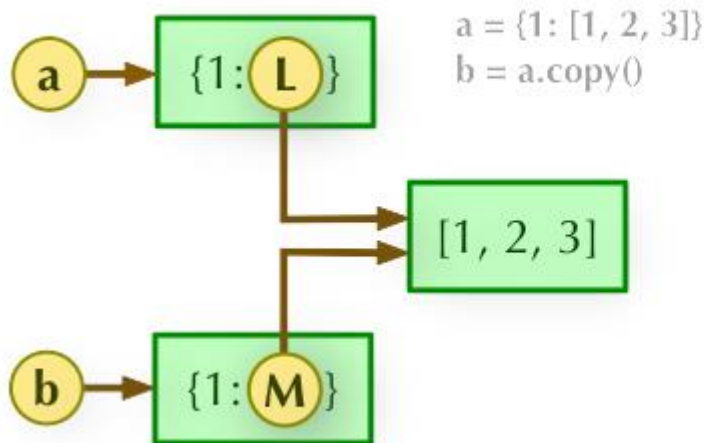
浅拷贝会创建新对象，其内容非原对象本身的引用，而是原对象内第一层对象的引用。  
(拷贝组合对象，不拷贝子对象)

- **直接赋值**：其实就是对象的引用（别名）。
- **浅拷贝(copy)**：拷贝父对象，不会拷贝对象的内部的子对象。
- **深拷贝(deepcopy)**：copy 模块的 deepcopy 方法，完全拷贝了父对象及其子对象。

1、**b = a**: 赋值引用，a 和 b 都指向同一个对象。

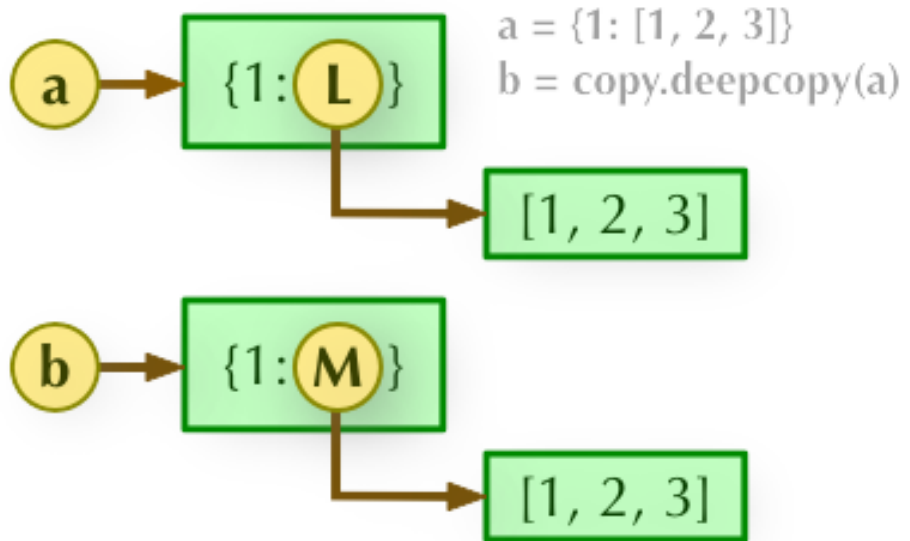


2、**b = a.copy()**: 浅拷贝，a 和 b 是一个独立的对象，但他们的子对象还是指向统一对象（是引用）。



---

`b = copy.deepcopy(a)`: 深度拷贝, a 和 b 完全拷贝了父对象及其子对象, 两者是完全独立的。



# 浅拷贝

- 浅拷贝，没有拷贝子对象内部，所以原始数据改变，子对象会改变

```
import copy
alist = [1,2,3,['a','b']]
c = copy.copy(alist)
print(alist)
print(c)  # 1浅拷贝
```

```
[1, 2, 3, ['a', 'b']]
[1, 2, 3, ['a', 'b']]
```

```
alist.append(5)
print(alist)
print(c)
```

```
[1, 2, 3, ['a', 'b'], 5]
[1, 2, 3, ['a', 'b']]
```

但是要注意的是，浅拷贝之所以称之为浅拷贝，是它仅仅只拷贝了一层，在列表中有一个嵌套的**list**，如果修改了它，情况就不一样了。

```
alist[3].append('boya')
print(alist)
print(c)  # 2浅拷贝，子对象不会被拷贝，会改变
```

```
[1, 2, 3, ['a', 'b', 'boya'], 5]
[1, 2, 3, ['a', 'b', 'boya']]
```

# 深拷贝

- 深拷贝，包含对象里面的自对象的拷贝，所以原始对象的改变不会造成深拷贝里任何子元素的改变

```
alist = [1,2,3,['a','b']]
d=copy.deepcopy(alist)
print(alist)
print(d) #始终没有改变
alist.append(5)
print(alist)
print(d) #始终没有改变
alist[3].append('boya')
print(alist)
print(d) #始终没有改变
```

```
[1, 2, 3, ['a', 'b']]
[1, 2, 3, ['a', 'b']]
[1, 2, 3, ['a', 'b'], 5]
[1, 2, 3, ['a', 'b']]
[1, 2, 3, ['a', 'b', 'boya'], 5]
[1, 2, 3, ['a', 'b']]
```

```
import copy
a = [[1, 2, 3],[0,0]]
b = copy.copy(a)           #浅拷贝得b
c = copy.deepcopy(a)       #深拷贝得c
print(id(a), id(b))        #a和b不同
>>> 60722504 60772424
```

## 实例

```
#!/usr/bin/python
# -*-coding:utf-8 -*-

import copy
a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a                #赋值，传对象的引用
c = copy.copy(a)     #对象拷贝，浅拷贝
d = copy.deepcopy(a) #对象拷贝，深拷贝

a.append(5)           #修改对象a
a[4].append('c')      #修改对象a中的['a', 'b']数组对象

print( 'a = ', a )
print( 'b = ', b )
print( 'c = ', c )
print( 'd = ', d )
```

以上实例执行输出结果为：

```
('a = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('b = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('c = ', [1, 2, 3, 4, ['a', 'b', 'c']])
('d = ', [1, 2, 3, 4, ['a', 'b']])
```



```
In [2]: import copy
list_1=[1, 2, 3, [1, 2]]
list_2=copy.copy(list_1)
print(list_2, '\n')
print(list_1 is list_2, '\n')
print(list_1[1] is list_2[1], '\n')
```

[1, 2, 3, [1, 2]]

False

True

```
In [3]: id(list_1)
```

Out[3]: 2660248629888

```
In [4]: id(list_2)
```

Out[4]: 2660248642176

```
In [5]: id(list_1[1])
```

Out[5]: 140710639253312

```
In [6]: id(list_2[1])
```

Out[6]: 140710639253312

```
In [8]: print(list_1[-1] is list_2[-1], '\n\n')
```

True

```
In [9]: id(list_1[-1])
```

Out[9]: 2660248641216

```
In [10]: id(list_2[-1])
```

Out[10]: 2660248641216

```
In [11]: print(id(list_1[0]), id(list_2[0]), '\n')
```

140710639253280 140710639253280

```
In [12]: list_1.append(4)
list_1[3].append(3)
print(list_1, '\n')
print(list_2, '\n')
```

[1, 2, 3, [1, 2, 3], 4]

[1, 2, 3, [1, 2, 3]]

---

Python中对象的赋值都是进行对象引用（内存地址）传递

使用`copy.copy()`，可以进行对象的浅拷贝，它复制了对象，但对于对象中的元素，依然使用原始的引用。

如果需要复制一个容器对象，以及它里面的所有元素（包含元素的子元素），可以使用`copy.deepcopy()`进行深拷贝

对于非容器类型（如数字、字符串、和其他'原子'类型的对象）没有被拷贝一说

如果元组变量只包含原子类型对象，则不能深拷贝。

# 列表方法

➤ Python为list提供了一些常用的内置方法，可以实现list的查询、增删和排序等功能，如下表所示。

方法	说明
list.index()	定位list中首个匹配项
list.count()	统计list中某个元素出现的次数
list.insert()	在list中指定位置插入元素
list.append()	追加元素至list末尾
list.extend()	将一个list扩展至另一list
list.pop()	按位置删除list中的元素
list.remove()	按对象删除list中的第一个匹配项
list.reverse()	反向排序list元素
list.sort()	对原list进行排序

---

# 元组

元组基本知识，元组VS列表

# 元组

- 定义元组的基本方法

- 元组(tuple)
  - 与list同属序列类数据，包含0个或多个对象引用的有序序列。与list不同的是，tuple是**不可更改**的数据类型。
  - 用圆括号()创建：空的圆括号创建空的tuple；包含一个或多个项的tuple可以使用逗号分隔开元素；如果tuple内只包含一个元素，需要在元素后加上逗号予以区分。有时，tuple必须被包含在圆括号中以避免语义二义性。例如，要将tuple(1,2,3)传递给一个函数，应该写成function((1,2,3))的形式，以免被识别成“1,2,3”这3个数字变量。
  - 支持索引，索引方式与str、list类似；与list相同的是，tuple可以进行连接、重复操作；与list不同的是，tuple中的元素**无法做增删操作，只能使用del函数删除整个tuple**。

- 元组是序列

- 索引和切片
- 基本操作

```
In [16]: tup1 = ('physics', 'chemistry', 2021, 913);  
          tup2 = (1, 2, 3, 4, 5, 6, 7 );  
          tup3=('Python',);
```

# 元组

---

- 元组 (**tuple**) 数据结构与列表类似，其中元素可以有不同的类型
- 元组中的元素是不可变的，即一旦初始化之后，就不能够再做修改（报错：元组对象不支持赋值）

```
kobe_tuple = (2, 'Jump Shot', 'Los Angeles Lakers', 'POR')  
kobe_tuple[0] = 3
```

```
-----  
TypeError  
Traceback (most recent call last) <ipython-input-3-70117284a3ad> in <module>()  
      1 kobe_tuple = (2, 'Jump Shot', 'Los Angeles Lakers', 'POR')  
----> 2 kobe_tuple[0] = 3  
TypeError: 'tuple' object does not support item assignment
```

# 元组

---

- 由于元组是不可变的，因此元组对没有append()、insert()和del这样的方法
- 那为何要使用元祖？
- 实际上，tuple的使用可以使得代码更安全，防止错误赋值导致重要对象的改变。

```
kobe_tuple.append(5)
```

```
File "<ipython-input-74-cece51e78192>", line 1
    kobe_tuple = (2, 'Jump Shot','Los Angeles Lakers' ,
'POR'])
```

^

```
SyntaxError: invalid syntax
```

# 元组的方法

➤ Python为tuple提供的内置方法较少，主要用于查询，如下表所示。

方法	说明
tuple.count()	计算某一元素在tuple中出现的次数
tuple.index()	找出某一元素在tuple中首次出现的位置

Python元组包含了以下内置函数

- 1、cmp(tuple1, tuple2)：比较两个元组元素。
- 2、len(tuple)：计算元组元素个数。
- 3、max(tuple)：返回元组中元素最大值。
- 4、min(tuple)：返回元组中元素最小值。
- 5、tuple(seq)：将列表转换为元组。



---

# 字典

创建字典，字典的基本操作  
字典的方法

# 字典 20180923

## • 创建字典

- dict可以用花括号{}创建：使用空的花括号创建空的dict；非空的花括号包含一个或多个逗号分隔的项，每个项包含一个键、一个冒号以及一个值。通过dict函数也可以创建dict：不带参数时返回一个空的dict；带一个映射类型参数时返回以该参数为基础的dict，当参数本身为dict时返回该参数的浅复制；也可以使用序列型参数，前提是序列中的每个项是包含两个对象的序列，第一个作为键，第二个作为值。

## • 映射关系

## • 创建字典的几种方法

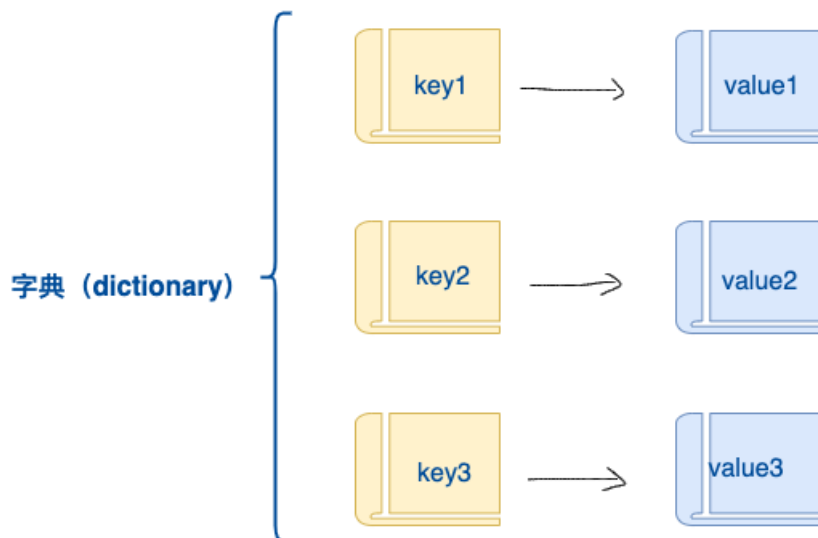
## • 字典的基本操作

- len(d)
- d[key]
- d[key] = value
- del d[key]
- key in d

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值 `key=>value` 对用冒号 `:` 分割，每个对之间用逗号`,`分割，整个字典包括在花括号 `{}` 中，格式如下所示：

```
d = {key1 : value1, key2 : value2, key3 : value3 }
```



键必须是唯一的，但值则不必。

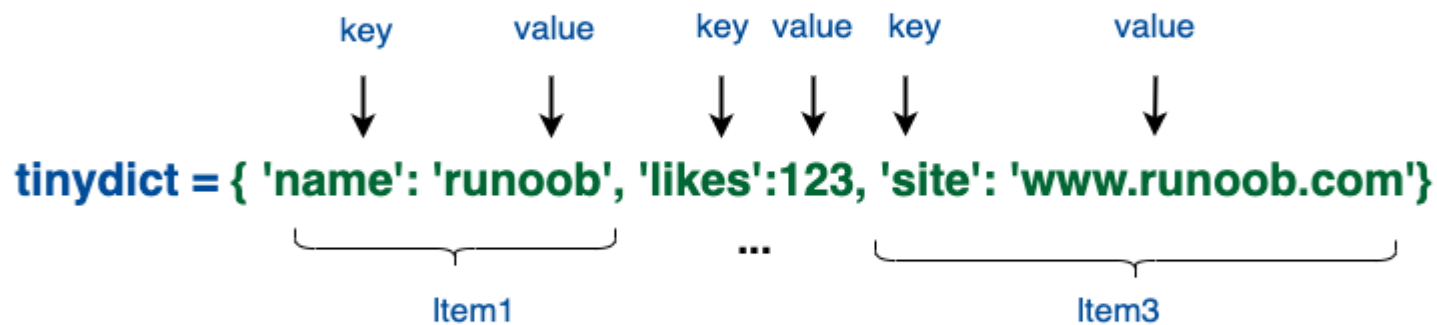
值可以取任何数据类型，但键必须是不可变的，如字符串，数字。

# 字典

- 字典（dict）在其他语言中被称作哈希映射（hash map）或者相关数组（associative arrays）
- 字典是一种大小可变的键值对集，其中的键（key）和值（value）都是Python对象

一个简单的字典实例：

```
dict = {'name': 'runoob', 'likes': 123, 'url': 'www.runoob.com'}
```



也可如此创建字典：

```
dict1 = { 'abc': 456 }  
dict2 = { 'abc': 123, 98.6: 37 }
```

# 字典的创建

---

- 字典创建使用大括号 {} 包含键值对，并用冒号 : 分隔键和值，形成键值对
- 不同键所对应的值可以相同，但是字典中的键必须是唯一的

```
kobe_dict = {2:'Jump Shot',1:'Jump Shot',3:'Jump Shot',4:'Jump Shot',5:'Driving Dunk Shot',6:'Jump Shot'}  
kobe_dict
```

```
{2: 'Jump Shot',  
 1: 'Jump Shot',  
 3: 'Jump Shot',  
 4: 'Jump Shot',  
 5: 'Driving Dunk Shot',  
 6: 'Jump Shot'}
```

# 字典的创建

---

- 利用for循环和zip()函数创建字典
- **zip()**函数用于将多个序列（列表、元组等） **中的元素配对**，产生一个如 [(列表1元素,列表2元素),(,)] 的新的元组列表
- **for**循环用于重复执行将值放入键中的操作

```
shot_id = [1,2,3]
shot_zone_area = ['Right Side(R)','Left Side(L)','Left Side Center(LC)']
kobe_dict = {}
for key,value in zip(shot_id,shot_zone_area):
    kobe_dict[key] = value
print(kobe_dict)
```

```
{1: 'Right Side(R)', 2: 'Left Side(L)', 3: 'Left Side Center(LC)'}
```

# 字典的方法

---

- 读取值的方法
- 视图对象
- 增加键值对
- 删除键值对

# 字典索引

- 字典的元素访问（以及插入、设置）方式与列表和元组一样
- 不同的是，列表和元组的索引号是按照顺序自动生成，而字典的索引号是键
- 字典中某值的索引还可以通过 `get()` 方法，如果字典不包含某个键，可以返回 `None`，或者自己指定的值

```
kobe_dict = {2:'Jump Shot',1:'Jump Shot',3:'Jump Shot',4:'Jump Shot',5:'Driving Dunk Shot',6:'Jump Shot'}
```

```
kobe_dict[5]
```

```
'Driving Dunk Shot'
```

```
print(kobe_dict.get(1))  
print(kobe_dict.get(6))  
print(kobe_dict.get(5,1))
```

```
Jump Shot  
Jump Shot  
Driving Dunk Shot
```

`dict.get(a,b)`

说明：`a`表示字典中的键，`b`为可选择参数，表示默认值，当没有从字典中找到键`a`时，返回默认值`b`，如果不写`b`，则没找到会返回`None`。

```
In [17]: My_dict={1:1,2:2,3:3}
```

```
In [21]: print(My_dict.get(4,7))
```

```
7
```

```
In [20]: print(My_dict.get(4))
```

```
None
```

# 字典的索引

---

- 通过 `in` 判断是否存在某个键，其语法跟在列表和元组中判断是否存在某个值是相同的

```
kobe_dict = {2:'Jump Shot',1:'Jump Shot',3:'Jump Shot',4:'Jump Shot',5:'Driving Dunk Shot',6:'Jump Shot'}
```

```
print(5 in kobe_dict)  
print(8 in kobe_dict)
```

```
True  
False
```



# 字典的索引

---

- 使用 `keys()` 方法或者 `values()` 方法

```
print('keys are', kobe_dict.keys())  
print('values are', kobe_dict.values())
```

```
keys are dict_keys([2, 1, 3, 4, 5, 6])  
values are dict_values(['Jump Shot', 'Jump Shot', 'Jump Shot', 'Jump Shot', 'Driving Dunk Shot', 'Jump Shot'])
```

# 字典的索引

---

- 当需要取出字典中的键值对用于下一步的分析，此时可以使用 `items()` 方法，该方法将返回所有键值对的视图，并将其保存在一个元组列表（列表中的元素为元组）中

```
kobe_dict.items()
```

```
dict_items([(2, 'Jump Shot'), (1, 'Jump Shot'), (3, 'Jump Shot'), (4, 'Jump Shot'),  
(5, 'Driving Dunk Shot'), (6, 'Jump Shot')])
```

`d.items()`可以得到`d`中的所有键值对元组的列表

```
In [57]: kobe_dict.items()
```

```
Out[57]: dict_items([(2, 'Jump Shot'), (1, 'Jump Shot'), (3, 'Jump Shot'), (4, 'Jump Shot'), (5, 'Driving Dunk Shot'), (6, 'Jump Shot')])
```

# 字典的索引

---

- 字典的删减有三种方法
- 使用 `del()` 函数依据键对单一元素或者整个字典进行删除(这里的一个元素即为一个键值对)
- 使用 `pop()` 方法依据键删除单一元素
- 使用 `clear()` 方法清空词典的所有元素

```
del kobe_dict[1]  
print(kobe_dict)
```

```
kobe_dict.pop(2)  
print(kobe_dict)
```

```
kobe_dict.clear()  
print(kobe_dict)
```

```
{2: 'Jump Shot', 3: 'Jump Shot', 4: 'Jump Shot', 5: 'Driving Dunk Shot', 6: 'Jump Shot'}
```

```
{3: 'Jump Shot', 4: 'Jump Shot', 5: 'Driving Dunk Shot', 6: 'Jump Shot'}
```

```
{}
```

# 字典的方法

Python为dict提供了丰富的内置方法，通过内置方法也可以实现查询、增删和创建，如下表所示。

方法	说明
<code>dict.items()</code>	返回dict的所有键值对
<code>dict.keys()</code>	返回dict的所有键
<code>dict.values()</code>	返回dict的所有值
<code>dict.get()</code>	以键查值，返回指定键的值
<code>dict.setdefault()</code>	以键查值，如果键不存在，将添加键
<code>dict.update()</code>	将dict的键值对更新到另一个dict中
<code>dict.copy()</code>	将一个dict的内容复制给另一dict
<code>dict.pop()</code>	删除指定键的值
<code>popitem()</code>	随机返回并删除dict中的一对键和值
<code>dict.clear()</code>	清空dict

---

# 集合

创建集合，集合的方法，不可变集合，集合的关系和运算

# 集合

---

- Python中有两种内置集合类型：set（可变集合）和frozenset（不可变集合）。set是引用零个或多个对象的无序组合，所引用的对象都是不可变的，所有内置的固定数据类型（如float、frozenset、int、str、tuple）都是不可变的。以下所指的集合都是set。
- set可以使用花括号{}或set函数创建。使用花括号{}创建集合时使用{}包裹一个或多个项，项与项间用","分割；空的set无法用{}创建。使用set函数创建set时，不带参数时返回空set；带一个参数时返回参数的浅复制；带多个参数时，则尝试将给定的对象转换为set。
- 创建集合
  - 可变集合
  - 互异、无序、确定
- 集合的方法
  - 增加元素
  - 删除元素
- 不可变的集合

# 集合的创建

---

- 集合的创建有两种方式：使用 `set()` 函数或`{}`
- 创建空集合，必须使用 `set()`，而不是`{}`，因为`{}`表示一个空的字典（默认）

`shot_type_set = set()` #通过`set()` 创建集合的实例！

```
shot_type_set = {'Driving Dunk Shot','Running Jump Shot','Jump Shot','Driving Layup shot','Layup Shot'}  
print(shot_type_set)
```

```
{'Layup Shot', 'Jump Shot', 'Driving Layup shot', 'Running Jump Shot', 'Driving Dunk Shot'}
```

# 关系和运算

---

- 元素与集合的关系

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                                # 这里演示的是去重功能
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                            # 快速判断元素是否在集合内
True
>>> 'crabgrass' in basket
False
```



# 关系和运算

- 集合与集合的关系

```
1 """
2     集合之间的关系：
3     1.两个集合是否相等：
4         可以使用运算符 == 或者 != 进行判断
5     2.一个集合是否是另一个集合的子集：
6         可以使用issubset() 来判断
7     3.一个集合是否是另一个集合的超集：
8         可以调用方法issuperset() 进行判断
9     4.两个集合是否有交集：
10        可以调用方法isdisjoint进行判断
11 """
12 set2 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
13 set3 = {9, 8, 7, 6, 4, 5, 1, 2, 3}
14 set4 = {1, 2, 3, 4, 5, 6}
15 print(set2 == set3)
16 print(set2 != set3)
17 print(set2.issubset(set3)) # set2是set3的子集
18 print(set4.issubset(set2)) # set4是set2的子集
19 print(set2.issuperset(set4)) # set2是set4的超集
```

a.issubset(b)	如果a的全部元素都包含于b，则为True
a.issuperset(b)	如果b的全部元素都包含于a，则为True
a.isdisjoint(b)	如果a和b没有公共元素，则为True

```
True
False
True
True
True
```

超集：如果一个集合S2中的每一个元素都在集合S1中，且集合S1中可能包含S2中没有的元素，则集合S1就是S2的一个超集，反过来，S2是S1的子集。 S1是S2的超集，若S1中一定有S2中没有的元素，则S1是S2的真超集，反过来S2是S1的真子集。

# 关系和运算

- 集合支持数学集合运算，如并、交、差以及对称差等

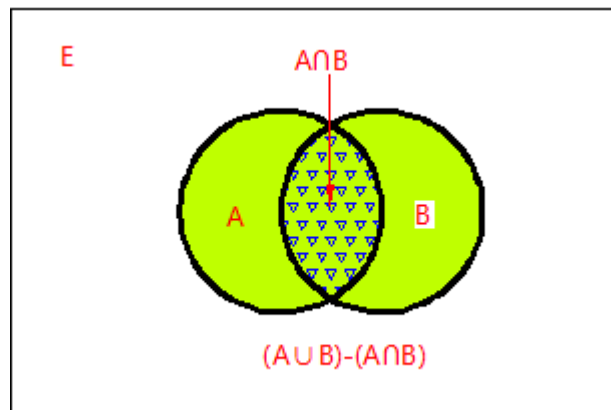
函数	其他表示法	说明
<code>a.union(b)</code>	<code>a b</code>	a和b中全部的唯一元素
<code>a.intersection(b)</code>	<code>a&amp;b</code>	a和b都有的元素
<code>a.difference(b)</code>	<code>a-b</code>	a中不属于b的元素
<code>a.symmetric_difference(b)</code>	<code>a^b</code>	a或b中不同时属于a和b的元素

# 关系和运算

- 集合支持数学集合运算，如并、交、差以及对称差等

```
shot_type_set1 = {'Jump Shot', 'Driving Dunk Shot'}
shot_type_set2 = {'Layup Shot', 'Jump Shot', 'Driving Layup shot'}
print('集合1:', shot_type_set1)
print('集合2:', shot_type_set2)
print('并集:', shot_type_set1 | shot_type_set2)
print('交集:', shot_type_set1 & shot_type_set2)
print('差集:', shot_type_set1 - shot_type_set2)
print('对称差(异或):', shot_type_set1 ^ shot_type_set2)
```

对称差



```
In [28]: shot_type_set1 = {'Jump Shot', 'Driving Dunk Shot'}
shot_type_set2 = {'Layup Shot', 'Jump Shot', 'Driving Layup shot'}
print('集合1:', shot_type_set1)
print('集合2:', shot_type_set2)
print('并集:', shot_type_set1 | shot_type_set2)
print('交集:', shot_type_set1 & shot_type_set2)
print('差集:', shot_type_set1 - shot_type_set2)
print('对称差(异或):', shot_type_set1 ^ shot_type_set2)

集合1: {'Driving Dunk Shot', 'Jump Shot'}
集合2: {'Driving Layup shot', 'Jump Shot', 'Layup Shot'}
并集: {'Driving Dunk Shot', 'Driving Layup shot', 'Layup Shot', 'Jump Shot'}
交集: {'Jump Shot'}
差集: {'Driving Dunk Shot'}
对称差(异或): {'Driving Dunk Shot', 'Driving Layup shot', 'Layup Shot'}
```

# 集合的方法

➤ set是可变的，但由于其中的项是无序的，因此没有索引的概念。set可变而无法索引，这使得它无法进行查询和修改元素的操作，但仍支持元素的增删，并可以清空和复制。set的常用操作基本都需要通过内置方法，如下表所示。

方法	说明
set.add()	将元素添加到set中，如已存在则不做操作
set.update()	通过更新添加元素，对象可以是其他高级变量
set.pop	随机set中的删除
set.remove()	删除指定元素
set.discard()	删除指定元素
set.clear()	清除set中所有元素
set.copy()	返回set的浅复制

# 不可变集合frozenset

集合中的元素可以变化，不过当想要固定其中的元素就不方便了。当需要一个可固定的集合时，就可以使用frozenset来返回，确保集合中的元素不可变更。一旦创建便不能更改，和set不同的是没有add，remove等改变集合的方法。

```
In [17]: t=frozenset('bookshop')
```

```
In [18]: print(t)
```

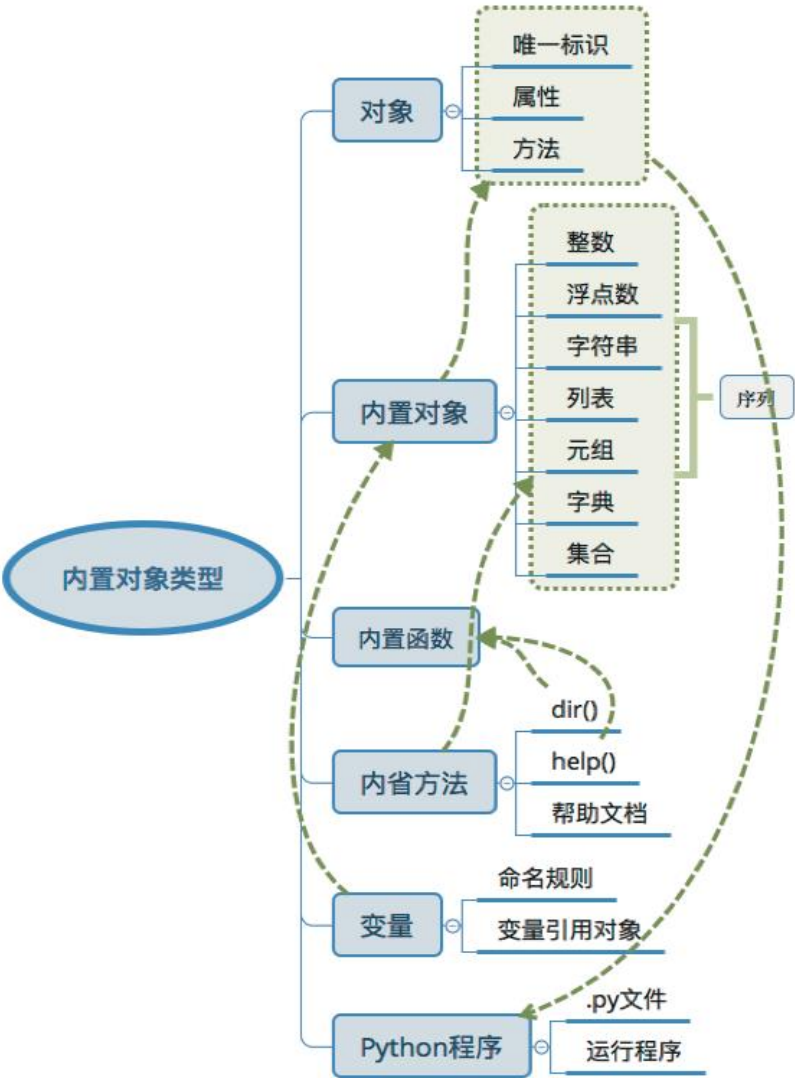
```
frozenset({'s', 'k', 'o', 'b', 'h', 'p'})
```

```
In [19]: w=t
```

```
In [20]: print(id(t),id(w))
```

```
2397557957120 2397557957120
```

# 内置对象类型全貌



# 总结

---

- 初步理解面向对象
- 输入和输出
- 数字
- 字符串
- 列表
- 元组
- 字典
- 集合