

- python 解释器的作用
- 数值类型
  - python 动态类型
  - 类型转换
- 标识符（变量、类、方法）命名规则--内容、大小写、关键字
  - 运算符
  - 变量命名规范
  - 字符串
  - 列表 []
  - 元组 tuple ()
  - 字典 dictionary {}
  - 字典练习
  - 集合---Set
- 函数
  - 参数类型
  - 匿名函数--lambda
  - return
  - 可更改（mutable）对象--list , dict / 不可更改(immutable) 对象--strings , tuples , numbers
- 模块--Module
  - globals() / locals()
  - reload()
- 面向对象
  - 函数属性
  - 函数内置类属性

## python 解释器的作用

将 Python 代码翻译成计算机认识的 0 / 1 并提交计算机执行。在解释器环境内可以一行行执行代码。也可用解释器程序，执行'.py'文件。

## 数值类型

---

类型	描述	说明
数字 (Number)	支持 <ul style="list-style-type: none"> <li>• 整数 (int)</li> <li>• 浮点数 (float)</li> <li>• 复数 (complex)</li> <li>• 布尔 (bool)</li> </ul>	整数 (int) , 如: 10、-10
		浮点数 (float) , 如: 13.14、-13.14
		复数 (complex) , 如: 4+3j, 以j结尾表示复数
		布尔 (bool) 表达现实生活中的逻辑, 即真和假, True表示真, False表示假。 True本质上是一个数字记作1, False记作0
字符串 (String)	描述文本的一种数据类型	字符串 (string) 由任意数量的字符组成
列表 (List)	有序的可变序列	Python中使用最频繁的数据类型, 可有序记录一堆数据
元组 (Tuple)	有序的不可变序列	可有序记录一堆不可变的Python数据集合
集合 (Set)	无序不重复集合	可无序记录一堆不重复的Python数据集合
字典 (Dictionary)	无序Key-Value集合	可无序记录一堆Key-Value型的Python数据集合

## python 动态类型

类型是属于对象的，而不是变量

变量没有类型，变量存储的数据有类型

## 类型转换

## 标识符（变量、类、方法）命名规则--内容、大小写、关键字

- 内容限定
- 大小写敏感
- 不可使用关键字 False None True and not or as assert 断言 async await break class continue while def except 包含捕获异常后的操作代码块，和 try / finally 结合使用。 for from import global if in is lambda 定义匿名函数 nonlocal 标识外部作用域的变量 pass 空的类，方法或函数的占位符 raise 异常抛出操作 return with 简化python 语句 yield 用于从函数依次返回值

## 运算符

- 成员运算符 IN / NOT IN
- 身份运算符 IS / IS NOT

# 变量命名规范

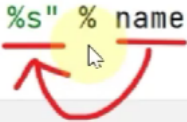
- \_
- 英文字符全小写

## 字符串

- 格式化 / 模板字符串

我们可以通过如下语法，完成字符串和变量的快速拼接。

```
name = "黑马程序员"
message = "学IT就来 %s" % name
print(message)
```



```
test x
D:\dev\Python\Python3.10.4\python.
学IT就来 黑马程序员
```

其中的，%s

- % 表示：我要占位
- s 表示：将变量变成字符串放入占位的地方

所以，综合起来的意思就是：我先占个位置，等一会有个变量过来，我把它变成字符串放到占位的位置

```
class_num = 22
salary1 = 32400
salary2 = 12500
message = "python第%s班，学员最高工资%s" % (class_num, salary2)
print(message)
```

通过 f"内容{变量}" 的格式 不管类型 / 不做精度控制

```
name = "传智播客"
set_up_year = 2006
stock_price = 19.99
# f: format
print(f"我是{name}，我成立于：{set_up_year}年，我今天的股价是：{stock_price}")
```

在无需使用变量进行数据存储的时候，可以直接格式化表达式，简化代码

```
print("字符串在python中的类型是：%s" % type('字符串'))
```

- 控制精度 `--m.n m` 控制宽度，要求数字（很少使用），设置宽度小于数字自身，不生效。`n`控制小数点精度，要求数字，会进行小数的四舍五入。

```
float1 = 33.334  
print("float1 is %8S.3f" % float1)
```

### 1. 精度控制的语法是：

`m.n`的形式控制，如`%5d`、`%5.2f`、`%.2f`

`m`和`.n`均可省略

### 2. 如果`m`比数字本身宽度还小，会发生什么事？

`m`不生效

### 3. `.n`会对小数部分做精度限制，同时：？

会对小数部分做四舍五入

- 字符串内建函数

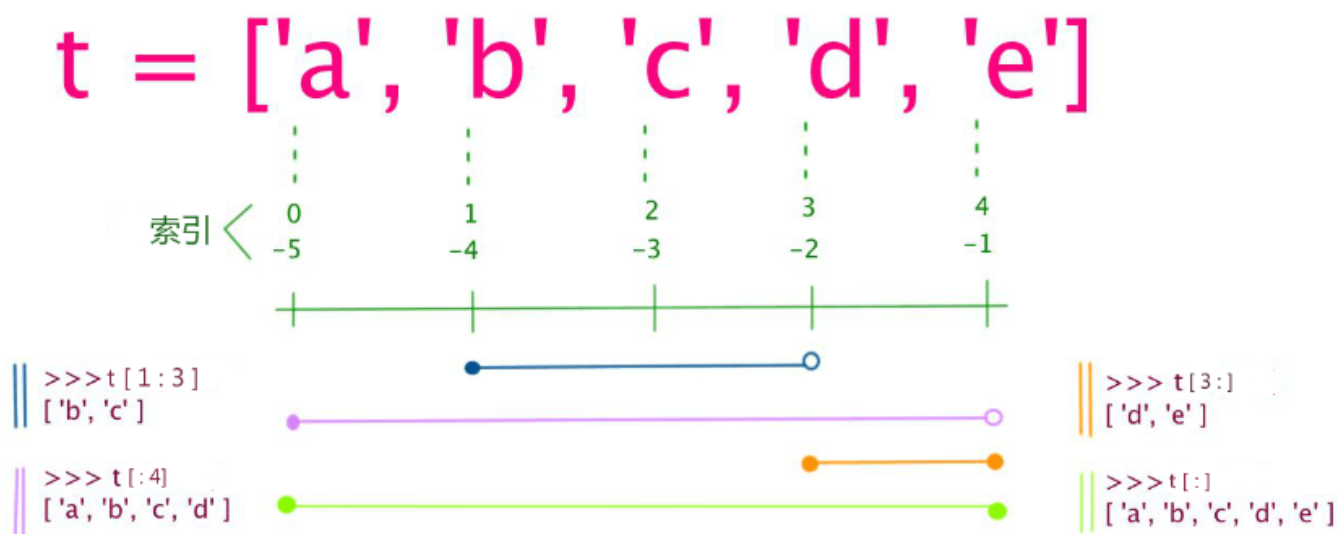
```
string.find(str , beg , end = len(string))
```

```
# 检查str是否包含在string中，不包含返回 -1
```

```
string.join(sequence)
```

```
# 将序列中的元素按string 连接成一个新的字符串
```

## 列表 []



- 更新 / 删除

```
list.append('str')  
# 更新添加  
  
del list[]  
# 删除列表元素
```

- 列表函数 / 方法 函数:

```
cmp(list1 , list2)  
len(list)  
max(list)  
min(list)  
list(seq) # 将元组转换为列表
```

```
#!/usr/bin/python

list1, list2 = [123, 'xyz'], [456, 'abc']

print cmp(list1, list2);
print cmp(list2, list1);
list3 = list2 + [786];
print cmp(list2, list3)
```

以上实例输出结果如下：

```
-1
1
-1
```

方法：

```
list.append()
list.count(obj) # 统计某个元素在列表中的出现次数
list.extend(seq) # 扩展列表
list.index(obj) # 第一个匹配项的索引
list.insert(index , obj)
list.pop(index)
list.remove(obj) # 移除列表中某个值的第一个匹配项
list.reverse()
list.sort()
```

- del / pop(index) / remove(obj)区别 del list[] 可删除任意位置元素并返回；  
remove(obj); pop(index);

## 元组 tuple ()

用 () 标识，不能二次赋值，相当于只读列表。

```
tuple[index] # 元组索引
tuple(seq) # 将列表转换为元组
```

# 字典 dictionary {}

无序的对象集合 和列表区别： 通过键来存取，而非偏移存取。

```
dict.clear() # 删除字典元素
dict.copy() # 字典的浅复制
dict.fromkeys(seq[ , val]) # 创建一个新字典，以序列seq中元素作为字典的键，val为字典所有键对应的初始值
dict.get(key , default = None) # 返回指定键的值
dict.has_key(key) # python3不支持
dict.items() # 以列表返回可遍历的（键， 值）元组数组
dict.keys()
dict.update(dict2) # 把字典dict2的键/值对更新到dict里
dict.values()
pop(key[ , default])
popitem() # 返回并删除字典中最后一对键值对
```

## 字典练习

```

emp_dic = {
    "王力宏": {
        "部门": "科技部",
        "工资": "3000",
        "级别": "1"
    },
    "周杰伦": {
        "部门": "市场部",
        "工资": "3400",
        "级别": "3"
    },
    "张学友": {
        "部门": "营销部",
        "工资": "3500",
        "级别": "2"
    },
    "周润发": {
        "部门": "科技部",
        "工资": "5000",
        "级别": "1"
    }
}

for name in emp_dic:
    if emp_dic[name]["级别"] == "1":
        emp_dic[name]["工资"] = str(int(emp_dic[name]["工资"]) + 1000)

for a in emp_dic:
    print(a, emp_dic[a])

```

```

王力宏 {'部门': '科技部', '工资': '4000', '级别': '1'}
周杰伦 {'部门': '市场部', '工资': '3400', '级别': '3'}
张学友 {'部门': '营销部', '工资': '3500', '级别': '2'}
周润发 {'部门': '科技部', '工资': '6000', '级别': '1'}

```

## 集合---Set



# 函数

## 参数类型

- 必备参数
- 关键字参数 允许函数调用时参数顺序和声明时不一致（自动匹配）：

```
def printme(str):  
    print(str)  
  
print(str = "Germany")
```

- 默认参数

```
def printme(age = 30 , name):  
    print(age , name)  
  
printme(nam = 'Nikki')
```

- 不定长参数 *\*val\**

```
def printme(arg1 , *val):  
    for i in val:  
        print(i)  
  
print(1 , 2 , 3)  
# 2  
# 3
```

## 匿名函数--lambda

```
sum = lambda arg1 , arg2:arg1 + arg2  
  
print sum(1 , 3)  
# 4
```

# return

```
def sum(arg1, arg2):  
    # 返回2个参数的和."  
    total = arg1 + arg2  
    print(total)  
  
# 调用sum函数  
print(sum(10, 20 ))
```

即函数运算的结果

结果: None

```
def sum(arg1, arg2):  
    # 返回2个参数的和."  
    total = arg1 + arg2  
    print(total)  
    return total  
  
# 调用sum函数  
print(sum(10, 20 ))
```

结果: 30

## 可更改 (mutable) 对象--list , dict / 不可更改 (immutable) 对象--strings , tuples , numbers

- a=5 后再赋值 a=10, 这里实际是新生成一个 int 值对象 10, 再让 a 指向它, 而 5 被丢弃, 不是改变 a 的值, 相当于新生成了 a。
- 可变类型: 变量赋值 la=[1,2,3,4] 后再赋值 la[2]=5 则是将 list la 的第三个元素值更改, 本身 la 没有动, 只是其内部的一部分值被修改了。

### python 函数的参数传递:

- 不可变类型: 类似 c++ 的值传递, 如 整数、字符串、元组。如 fun(a), 传递的只是 a 的值, 没有影响 a 对象本身。比如在 fun(a)内部修改 a 的值, 只是修改另一个复制的对象, 不会影响 a 本身。
- 可变类型: 类似 c++ 的引用传递, 如 列表, 字典。如 fun(la), 则是将 la 真正的传过去, 修改后 fun 外部的 la 也会受影响

# 模块--Module

---

## globals() / locals()

## reload()

## 面向对象

---

**self** 相对于 其他编程语言的 **this** 指针，**self** 通常作为类方法的第一个参数，用于访问实例变量、实例方法和类方法。

```
class Employee:
    '所有员工的基类'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name, ", Salary: ", self.salary
```

*init()*: 类的构造函数或初始化方法。

## 函数属性

- getattr(obj, name[, default])--访问对象属性
- hasattr(obj, name)--检查是否存在一个属性
- setattr(obj, name, value)--设置一个属性，若不存在，创建一个新属性
- delattr(obj, name)--删除属性

# 函数内置类属性

- *dict* : 类属性--包含一个字典， 由类的数据属性组成
- *doc* : 类的文档字符串
- *\_name* : 类名
-