# Python基础

## 为什么学Python

1 开源免费

开源自由，可以从Internet免费获得，可以随产品分发，甚至可以出售。

可移植的

2 简单易学

使用优美的语法，贴近人类的表达习惯，使你编写程序更容易。

易于使用

简单易学

核心语言相当简单，有经验的程序员通过几天（甚至几个小时）的学习就可以写出不错的python代码。

Python的类库齐全并且使用简洁，如果要实现同样的功能，Python 10行代码可以解决，C可能就需要100行甚至更多.

开发效率高，通常实线同样的功能，python的代码量只有C++或Java代码的五分之一到三分之一，并且可以立即执行，不需要编译链接等步骤，进一步提升开发效率。

3 功能强大

一个大型的标准库支持许多常见编程任务,诸如连接到Web服务器、 正则表达式全文检索、阅读及修改文件。

可以应用于众多领域，如：系统维护，数据库编程，用户图形接口，网络服务，图像处理，快速原型，数值计算和科学计算，人工智能，大数据，机器人等等。目前业内几乎所有大中型互联网企业都在使用Python，如：Youtube、豆瓣、知乎、Google、Yahoo!、Facebook、NASA、百度、腾讯、汽车之家、美团等。互联网公司广泛使用Python来做的事一般有：自动化运维、自动化测试、大数据分析、爬虫、Web 等。

可移植的

可混合的

缺点： Python与C或C++相比在速度上稍显逊色。

不适合使用Python的场景：编写操作系统，通常用C和汇编；编写手机应用；编写3D游戏，最好用C或C++。

## Python安装

第一步：下载Python安装包

在Python的官网 www.python.org 中找到最新版本的Python安装包，点击进行下载，请注意，当你的电脑是32位的机器，请选择32位的安装包，如果是64位的，请选择64位的安装包；

第二步：安装

A.双击下载好的安装包，弹出如下界面：

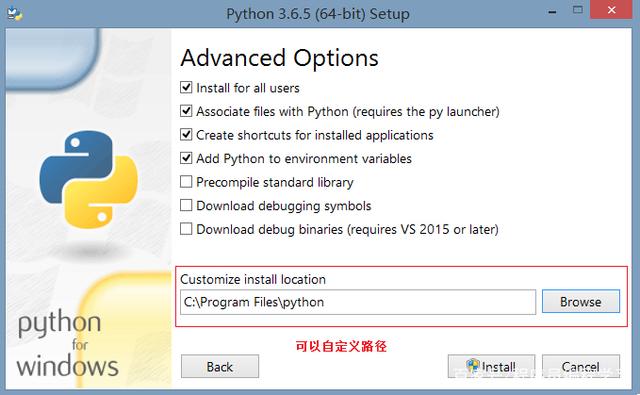


这里要注意的是，将python加入到windows的环境变量中，如果忘记打勾，则需要手工加到环境变量中；在这里我选择的是自定义安装，点击“自定义安装”进行下一步操作；

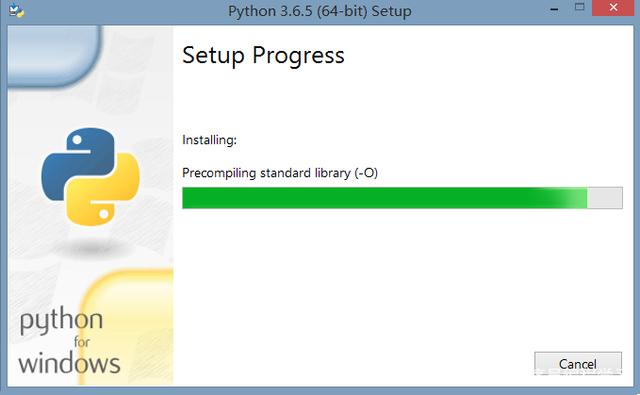
B.进入到下一步后，选择需要安装的组件，然后点击下一步:



C.在这里可以自定义路径选择安装：



D.点击下一步后，就开始真正安装了：



E:安装完成后，会有一个安装成功的提示界面：



第三步：测试

python安装好之后，我们要检测一下是否安装成功，用系统管理员打开命令行工具cmd，输入“python -V”,然后敲回车，如果出现如下界面，则表示我们安装成功了；



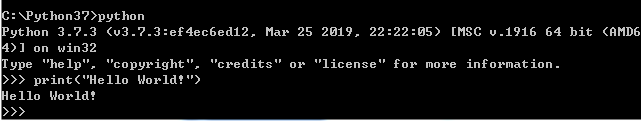
Linux系统通常已经预装了python，也可以通过相应的包管理器安装，比如yum，apt都可以。

## 程序运行

### 交互命令行

python

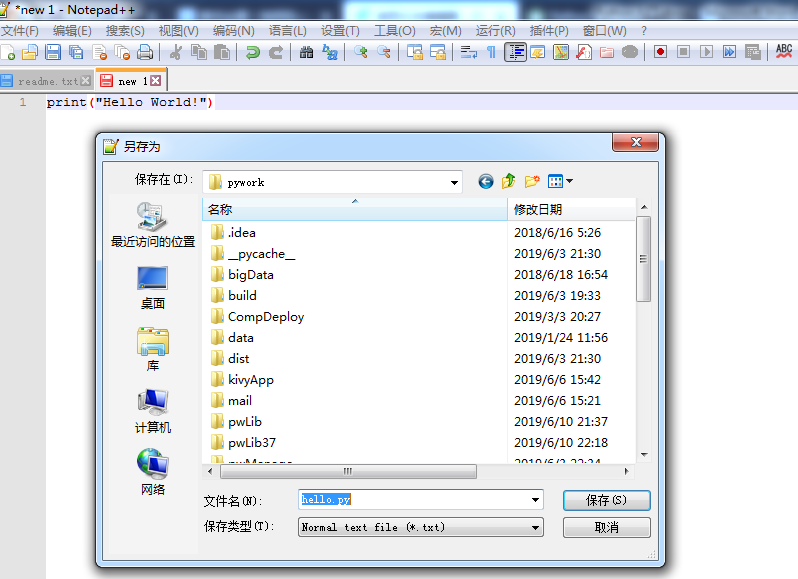
>>>print 'Hello world!'

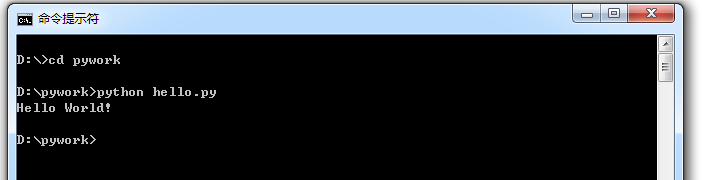


### 作为Python模块文件

保存为文本文件

python hello.py





### 作为Unix/Linux文件型脚本

脚本第一行#!解释器路径，赋予可执行权限，即可在命令行直接输入文件名执行

## 语法基础

程序的基本组成

注释

缩进

变量

表达式

输入

输出

分支

循环

示例：

*# -\*- coding:utf-8 -\*-*

*name = input("输入你的名字:")*

*if len(name) == 0:*

*print("Hello world!")*

*else:*

*print("Hello {}".format(name))*

### 注释

代码中的说明信息，计算机不执行

注释方法：

单行注释以#开头

多行注释以'''开头和结尾

''' This is comment'''

### 缩进

Python语言不使用java/c语言中的{} ; ，主要靠缩进表示代码的层次关系

通常一个缩进=4个空格

### 变量

变量是用于引用对象的标识符

1.变量的主要作用，就是用来存储信息，然后在计算机程序中使用这些信息。

2.通过给变量赋值，将数据与一些能够描述的名字连接起来。

3.变量的值，最终是存储在内存中。

**赋值**

语法：变量名 = 对象（数值，表达式等）

pi = 3.14

radius = 4

area = pi \* radius\*\*2

print(area)

增量赋值运算符

x+=2

x-=2

x\*=2

x/=2

x%=2

x\*\*=2

同步赋值

同时给多个变量赋值（先运算右侧N个表达式，然后同时将表达式结果赋给左侧）

<变量1>,......,<变量N> = <表达式1>,......,<表达式N>

perimeter, area = 2\*pi\*radius, pi\*radius\*\*2

用同步赋值交换对象的值：

x, y = y, x 等价于 t = x

x = y

y = t

**变量命名规则（函数/模块）**

任意长度

包含数字、字母、下划线

首字母必须是字母或下划线

大小写敏感

标识符不能是关键字

Python关键字（3.7版本）

False await else import pass

None break except in raise

True class finally is return

and continue for lambda try

as def from nonlocal while

assert del global not with

async elif if or yield

### 表达式

程序中产生或计算新数据值的一行代码

Python语言的关键字或操作符可产生符合语法的表达式

不改变缩进的前提下，适当增加空格可提高程序可读性。

例如：

*name = 'Python'*

*PI = 3.14*

*PI\*5\*\*2*

### 输入输出函数

标准（键盘）输入

input()函数

input([prompt]) prompt提示语，可以没有

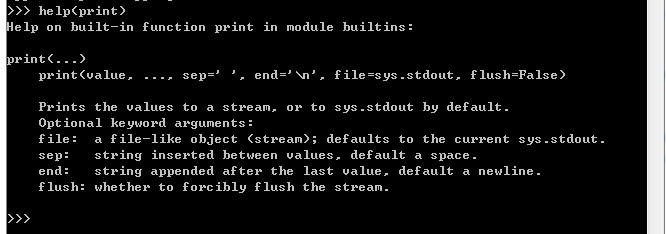
name = input("please input your name:")

标准（控制台）输出

print()函数

print()

print("HelloWorld!")



### 分支语句

控制程序运行，根据判断条件选择程序执行路径。

if <表达式1成立>:

<表达式组1>

elif <表达式2成立>:

<表达式组2>

......

elif <条件N成立>:

<表达式组N>

else:

<表达式组N+1>

### 循环语句

控制程序运行，根据判断条件或计数条件确定一段程序的运行次数

**for循环**

通用序列迭代器，可遍历任何有序的序列对象内的元素。可用于字符串、列表、元组、其他内置可迭代对象以及我们通过类创建的新对象。

for <target> in <object>:

<表达式组1>

else:

<表达式组2>

*sum = 0*

*for i in range(5):*

*print(i\*\*2)*

*sum += i\*\*2*

*else:*

*print(sum)*

**while循环**

只要顶端测试一直计算得到真值，就会重复执行一个语句快。当顶端测试为假时，控制权传递给while块后的语句。就是循环主体在顶端测试为真时会重复执行，而如果测试一开始就是假，主体就不会执行。

格式如下:

while <测试条件>:

<语句组1>

else:

<语句组2>

**break、continue、pass和循环else**

上面介绍了Python中的两种循环结构for和while，下面介绍下几个于循环相关的简单语句：break、continue、 pass和循环else。

break语句的作用是跳出最近所在的循环（跳过整个循环语句）。

continue语句的作用是跳到最近所在循环的开头处（来到循环的首行）。

pass语句什么也不做，只是空占位语句。当语法需要，并且卖油任何实用的语句可写时，就用这个。

循环else块只有当循环正常离开时才会执行（也就是没有碰到break语句）。

循环else分句是Python语言特有的，c和java中都没有这种语法。

使用break和continue的循环通常是这样的：

while <循环测试条件>：

if <测试条件1>: break #符合条件就结束循环，跳过else语句块

if <测试条件2>: continue #符合条件就跳过语句组1，回到while测试循环条件

<语句组1>

else :

<语句组2> #如果while循环结束，并未执行break语句，执行这段

## 数据类型

### Python中的基本对象类型

#### 数字

int（整型）

　　在32位机器上，整数的位数为32位，取值范围为-2\*\*31～2\*\*31-1，即-2147483648～2147483647

　　在64位系统上，整数的位数为64位，取值范围为-2\*\*63～2\*\*63-1，即-9223372036854775808～9223372036854775807

long（长整型）

　　跟C语言不同，Python的长整数没有指定位宽，即：Python没有限制长整数数值的大小，但实际上由于机器内存有限，我们使用的长整数数值不可能无限大。

　　注意，自从Python2.2起，如果整数发生溢出，Python会自动将整数数据转换为长整数，所以如今在长整数数据后面不加字母L也不会导致严重后果了。

float（浮点型）

　　浮点数用来处理实数，即带有小数的数字。类似于C语言中的double类型，占8个字节（64位），其中52位表示底，11位表示指数，剩下的一位表示符号。

complex（复数）

　　复数由实数部分和虚数部分组成，一般形式为x＋yj，其中的x是复数的实数部分，y是复数的虚数部分，这里的x和y都是实数。

#### 布尔值

　　真或假

　　1 或 0

#### 字符串、列表、元组、

#### 字典（无序）

### 为何区分对象类型

1 不同类型对象运算规则不同

2 不同类型对象在计算机内表示方式不同

3 为何区分整数与浮点数

浮点数表达能力更强

浮点数有精度损失

CPU有专门的浮点运算部件

>>> print(3.3==1.1\*3)

False

### 类型转换

int('123') = 123

str(123) = '123'

float('123') = 123.

bool(123) = True

注：type函数可查看对象类型

dir(9)

(9).to\_bytes(8,sys.byteorder)

## 运算符与表达式

算数运算 + - \* / % \*\* //

比较运算 < > <= >= == !=

赋值运算 =

逻辑运算 and or not

成员运算 in

### 算数运算



### 比较运算

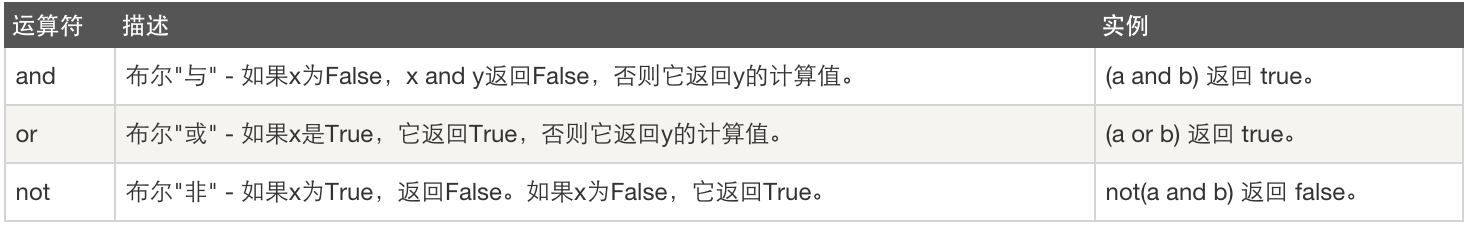
注:python3中<>已经废弃：



### 赋值运算



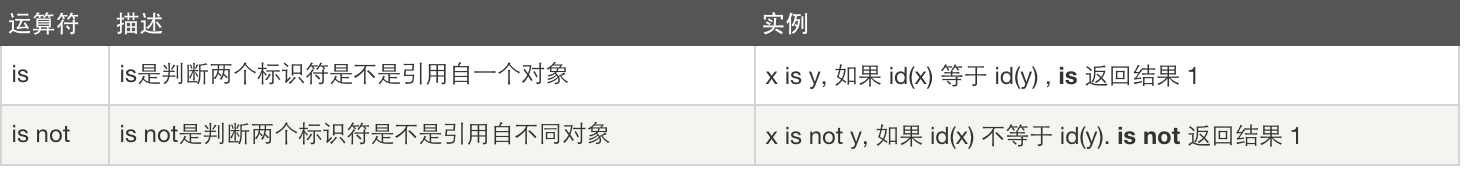
### 逻辑运算



### 成员运算



### 身份运算



### 位运算



### 运算符优先级



### 自动类型转换

参与运算的两个对象类型相同，则结果类型不变

参与运算的两个对象类型不同，则按如下规则自动类型转换

bool -> int -> float -> complex

1.0+3 = 4.0

True + 6.0 = 7.0

### 三元运算

result = 值1 if 条件 else 值2

如果条件为真：result = 值1  
如果条件为假：result = 值2

## 函数

参考 Python语言程序设计 北京交通大学 鲁凌云

定义：完成特定功能的语句组，可以给这组单位取一个名字，作为一个单位使用。

函数优点

实现代码复用，提高开发效率，减少重复编码。

简化程序结构，将复杂的编程问题分解为若干简单的子问题，每个子问题由函数来解决。 使用函数可以使程序更容易阅读；使调试测试更容易。

信息封装与隐藏。

### 函数定义

def funcname(params):

xxx

xxx

return xxx

函数名、参数、函数体

### 匿名函数

f = lambda 参数1, 参数2, ... :表达式

f = lambda x,y,z : x+y+z

g = lambda x: lambda y : x+y

### 带默认参数值的匿名函数

f = lambda x,y=3 : x\*\*y

f(2,2) = 4

f(2) = 8

### 函数调用

funcname()

### 参数传递

1 按照位置传参数

2 按照关键字传参数

3 按照默认值传参数

函数定义时如果指定默认值参数，这类参数必须放在无默认值参数的后面。

def sum(a,b,c=10)

return a+b+c

sum1 = sum(4,5,6)

sum2 = sum(a=4,c=8,b=6)

可变数量的参数定义

定义函数时，如果参数数量不确定，可以定义可变数量的参数。

参数名前面加一个'\*'表示参数是以形参名为标志符的元组，元组中的元素个数可以是零个、一个或多个。

参数名前加两个'\*'表示参数数据类型是字典，其中参数名为“key”，参数值为“value”

定义函数时，可以混合使用多种参数传递方式，要遵循如下规则：

1 关键字参数放在位置参数后面

2 元组参数放在关键字参数后面

3 字典参数放在元组参数后面

调用函数时，首先按位置顺序传递参数，其次按关键字传递参数。多余的非关键字参数传递给元组，多余的关键字参数传递给字典。

### 函数返回值

函数定义中，return语句是可选项，主要功能是返回函数执行结果。

返回的数据类型除常见的数值，字符串，还可以是布尔值、列表对象等。

当有多条return语句时，执行第一条return语句就会退出函数，不再执行return后面的语句。

如果函数没有return语句，自动返回None，表示没有返回值。

return语句可以返回多个值，此时实际上返回的是一个元组对象。

### 变量作用域

函数内部创建的对象只能在函数内部访问，这种变量称为局部变量。

函数外部定义的变量可以被程序的任何部分访问，这种变量称为全局变量。

Python3中全局变量在函数内访问时需要用global指明是访问全局变量。

### 内建函数

|  |  | **Built-in Functions** |  |  |
| --- | --- | --- | --- | --- |
| [abs()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "abs) | [delattr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "delattr) | [hash()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "hash) | [memoryview()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-memoryview) | [set()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-set) |
| [all()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "all) | [dict()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-dict) | [help()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "help) | [min()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "min) | [setattr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "setattr) |
| [any()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "any) | [dir()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "dir) | [hex()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "hex) | [next()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "next) | [slice()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "slice) |
| [ascii()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "ascii) | [divmod()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "divmod) | [id()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "id) | [object()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "object) | [sorted()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "sorted) |
| [bin()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "bin) | [enumerate()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "enumerate) | [input()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "input) | [oct()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "oct) | [staticmethod()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "staticmethod) |
| [bool()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "bool) | [eval()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "eval) | [int()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "int) | [open()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "open) | [str()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-str) |
| [breakpoint()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "breakpoint) | [exec()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "exec) | [isinstance()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "isinstance) | [ord()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "ord) | [sum()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "sum) |
| [bytearray()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-bytearray) | [filter()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "filter) | [issubclass()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "issubclass) | [pow()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "pow) | [super()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "super) |
| [bytes()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-bytes) | [float()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "float) | [iter()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "iter) | [print()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "print) | [tuple()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-tuple) |
| [callable()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "callable) | [format()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "format) | [len()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "len) | [property()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "property) | [type()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "type) |
| [chr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "chr) | [frozenset()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-frozenset) | [list()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-list) | [range()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "func-range) | [vars()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "vars) |
| [classmethod()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "classmethod) | [getattr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "getattr) | [locals()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "locals) | [repr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "repr) | [zip()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "zip) |
| [compile()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "compile) | [globals()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "globals) | [map()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "map) | [reversed()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "reversed) | [\_\_import\_\_()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "__import__) |
| [complex()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "complex) | [hasattr()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "hasattr) | [max()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "max) | [round()](mk:@MSITStore:C:\Python37\Doc\python373.chm::/library/functions.html" \l "round) |  |

### 常用内建函数

dir函数

返回由给定模块、类、实例，或其他类型的所有成员组成的列表，在交互式python解释器下很有用，也可以用在其他地方。

help函数

apply函数--

eval函数

open函数

len函数

sum函数

sorted函数

filter函数

list函数

dict函数

max函数

map函数

resersed函数

input函数

print函数

## 函数库(模块)

模块，实现一定功能的Python脚本集合

引入

import math

察看模块内容

dir(math)

察看帮助

help(math.floor)

### 函数库的引用方式

第一种

import <库名>

如果要使用库中的函数：

<库名>.<函数名>

例如：

import math

a = math.ceil(3.4)

还可以在引入时给函数库取一个简短的名字，方便调用

import <库名> as <简称>

调用：

<简称>.<函数名>

第二种

from <库名> import <函数名>

from <库名> import \*

调用时不需要<库名>，直接使用<函数名>

from math import ceil

a = ceil(3.6)

### 常用函数库

#### 内建函数和异常

定义内建函数(例如 len,int, range ...)的 \_ \_builtin\_ \_ 模块, 以及定义所有内建异常的 exceptions模块.

Python 在启动时导入这两个模块, 使任何程序都能够使用它们.

#### 操作系统接口模块

Python 有许多使用了 POSIX 标准 API 和标准 C 语言库的模块. 它们为底层

操作系统提供了平台独立的接口.

这类的模块包括: 提供文件和进程处理功能的 os 模块; 提供平台独立的文件

名处理 (分拆目录名, 文件名, 后缀等)的 os.path 模块; 以及时间日期处理

相关的 time/datetime 模块.

#### 类型支持模块

标准库里有许多用于支持内建类型操作的库. string 模块实现了常用的字符串处理. math 模块提供了数学计算操作和常量(pi, e 都属于这类常量), cmath 模块为复数提供了和 math 一样的功能.Decimal模块实现了小数运算。

#### 正则表达式

re 模块为 Python 提供了正则表达式支持. 正则表达式是用于匹配字符串或特定子字符串的有特定语法的字符串模式.

#### 语言支持模块

sys 模块可以让你访问解释器相关参数,比如模块搜索路径,解释器版本号等.operator 模块提供了和内建操作符作用相同的函数. copy 模块允许你复制对象, Python 2.0 新加入的 gc 模块提供了对垃圾收集的相关控制功能.

os

os.path

系统操作

sys 模块可以让你访问解释器相关参数,比如模块搜索路径,解释器版本号等.

re 模块为 Python 提供了正则表达式支持. 正则表达式是用于匹配字符串或特

定子字符串的有特定语法的字符串模式.

string 模块实现了常用的字符串处理.

math 模块提供了数学计算操作和常量(pi, e 都属于这类常量),

cmath 模块为复数提供了和 math 一样的功能.

time

shutil

getopt

random

随机数函数库，获取0到1之间的随机数，获取随机整数，随机获取序列中的对象。

md5

sha

zlib

## 文件操作

文件是存储在硬盘、CD、DVD或其他存储介质上的数据集合。

待处理的数据或处理完的数据结果都可以保存在文件中，方便我们随时使用。

文件可以保存的数据形式多种多样，比如文本、图片、音频、视频、可执行程序等。

### 打开文件

　　file\_obj = file("文件名","模式")

file\_obj = open("文件名","模式")

打开文件的模式有：

r，以只读方式打开文件

w，打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

a，打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

w+，打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。

### 读取文件的内容

Python提供三种读取文件内容的方法：read()，readline()，readlines()

如果给定参数，read方法返回参数指定的读取字节数，否则返回全部内容。

# 一次性加载所有内容到内存

file\_obj.read()

# 读取512字符

file\_obj.read(512)

# 一次性加载所有内容到内存，并根据行分割成字符串

file\_obj.readlines()

# 每次仅读取一行数据

for line in file\_obj:

　　print(line)

|  |  |
| --- | --- |
|  |  |

### 写文件的内容

写入文件时用write()方法，有两种写入方式，'w'和'a'。

file\_obj.write('内容')

### 关闭文件句柄

file\_obj.close()

writelines()方法与readlines()方法对应，向文件中写入一个列表，列表中的每个元素就是文件的每一行内容。

### 操作Excel

主要用到xlrd和xlwt这两个库，xlrd是读Excel，xlwt是写Excel的库

库安装：

pip install xlrd

pip install xlwt

参考高级语言程序设计(Python)CAP 哈工大 车万鹏

## 字符串

字符串(String)是一个字符的序列

### 三种定义方式:

单引号括起来

s = 'hello world'

双引号括起来

s = "hello world"

三引号括起来("""或''')

s = """this is a test

hello world."""

### 基本运算

长度（len()函数）

拼接（+）

重复（\*）

成员运算符（in）

示例：

计算字符串中元音字母数目

def vowels\_count(s):

ret = 0

for item in s:

if item in 'aeiouAEIOU'：

ret+=1

return ret

### 索引与切片

字符串是不可变的

### 字符串方法

replace

find

split

### 字符串比较

按顺序依次比较每个位置上的字符，如果相同就比较下一个，如果不同大小关系就由这两个字符的大小关系决定

### 字符串格式化format方法

{}的格式 {field name,align width.precision type}

print("PI is {}".format(math.pi))

print("PI is {:4.2f}".format(math.pi))

## 列表与元组

内建数据结构，用来存储一系列元素，列表与元组的区别是，元组是不可变对象，不能修改，列表可以修改。

元组：

t = (1,2,3,4,5)

列表

l = [1,2,3,4,5]

列表和元组与字符串一样，都是有序序列，所以列表和元组与字符串有很多相同点：

1 索引运算[]

2 切片[:]

3 拼接和重复 + \*

4 成员运算 in

5 长度 len

6 循环迭代 for

与字符串只能包含字符不同，列表和元组可以包含多种类型的对象。

### 列表的方法

append() #追加

extend() #扩充

pop() #删除

remove() #删除

sort() #排序

reverse() #逆序

### 列表赋值

a = [1,2,3,4]

b = a

b[1] = 100

b = [:]

b[1] = 100

### 嵌套列表

x = [[1,2,3,[4,5,6],[7,8,9]]]

len(x) = ?

len(x[1]) = ?

### 列表解析、推导

列表解析：由原列表创建新列表的简单方法

[表达式 for 变量 in 列表 if 条件]

例如生成x平方的序列：

ist = []

for x in range(1,10)

ist.append(x\*\*2)

print(ist)

列表解析：

ist = [x\*\*2 for x in range(1,10)]

使用列表解析、推导求输入数字x的因数和

例如输入6 ， 因数和为1+2+3+6 = 12

sum([i for i in range(1,x+1) if x%i == 0])

## 字典与集合

字典：一系列“键-值（key-value）”对

通过“键”查找对应的“值”

电话本，字典

java中的Map (HashTable,HashMap)

### 创建字典

使用{}创建字典

使用:指明 键：值 对

myDict = {'小李':87977111,'李明':13598881112,'小丁':98761221}

字典中的键必须是不可变的且不可重复，值可以是任意类型。

### 访问字典

使用[]运算符，键作为索引

print(myDict['小李'])

增加新的键值对：

myDict['小王']=17621231232

### 字典运算符和方法

len

key in myDict

for key in myDict

myDict.items() #全部键值对

myDict.keys() #全部键

myDict.values() #全部值

myDict.clean() #清空字典

示例：

计算一个字符串中每个字母出现的次数

count = {}

for i in 'hello world':

if i in count:

count[i] += 1

else:

count[i] = 1

### 集合（Set）

无序不重复元素集

和字典类似，但是只有“键”没有“值”

创建

x = set()

x = {key1,key2,...}

添加和删除

x.add()

x.remove()

集合的运算符：

- 差集

& 交集

| 并集

!= 不等于

== 等于

in 成员

for key in set 枚举

## pip介绍

# Python进阶

## 虚拟环境

## 面向对象

Python从设计之初就已经是一门面向对象的语言，正因为如此，在Python中创建一个类和对象是很容易的。本章节我们将详细介绍Python的面向对象编程。

如果你以前没有接触过面向对象的编程语言，那你可能需要先了解一些面向对象语言的一些基本特征，在头脑里头形成一个基本的面向对象的概念，这样有助于你更容易的学习Python的面向对象编程。

接下来我们先来简单的了解下面向对象的一些基本特征。

### 面向对象技术简介

* **类(Class):** 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
* **类变量：**类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
* **数据成员：**类变量或者实例变量, 用于处理类及其实例对象的相关的数据。
* **方法重写：**如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。
* **局部变量：**定义在方法中的变量，只作用于当前实例的类。
* **实例变量：**在类的声明中，属性是用变量来表示的。这种变量就称为实例变量，是在类声明的内部但是在类的其他成员方法之外声明的。
* **继承：**即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。
* **实例化：**创建一个类的实例，类的具体对象。
* **方法：**类中定义的函数。
* **对象：**通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

### 创建类

使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾:

class ClassName:

'类的帮助信息' #类文档字符串

class\_suite #类体

类的帮助信息可以通过ClassName.\_\_doc\_\_查看。

class\_suite 由类成员，方法，数据属性组成。

### 实例

## 多线程

多线程类似于同时执行多个不同程序，多线程运行有如下优点：

* 使用线程可以把占据长时间的程序中的任务放到后台去处理。
* 用户界面可以更加吸引人，这样比如用户点击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度
* 程序的运行速度可能加快
* 在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下我们可以释放一些珍贵的资源如内存占用等等。

线程在执行过程中与进程还是有区别的。每个独立的进程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

每个线程都有他自己的一组CPU寄存器，称为线程的上下文，该上下文反映了线程上次运行该线程的CPU寄存器的状态。

指令指针和堆栈指针寄存器是线程上下文中两个最重要的寄存器，线程总是在进程得到上下文中运行的，这些地址都用于标志拥有线程的进程地址空间中的内存。

* 线程可以被抢占（中断）。
* 在其他线程正在运行时，线程可以暂时搁置（也称为睡眠） -- 这就是线程的退让。

### 开始学习Python线程

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用thread模块中的start\_new\_thread()函数来产生新线程。语法如下:

thread.start\_new\_thread ( function, args[, kwargs] )

参数说明:

* function - 线程函数。
* args - 传递给线程函数的参数,他必须是个tuple类型。
* kwargs - 可选参数。

## 正则表达式

## GUI编程(Tkinter)

Python 提供了多个图形开发界面的库，几个常用 Python GUI 库如下：

* **Tkinter：** Tkinter 模块(Tk 接口)是 Python 的标准 Tk GUI 工具包的接口 .Tk 和 Tkinter 可以在大多数的 Unix 平台下使用,同样可以应用在 Windows 和 Macintosh 系统里。Tk8.0 的后续版本可以实现本地窗口风格,并良好地运行在绝大多数平台中。
* **wxPython：**wxPython 是一款开源软件，是 Python 语言的一套优秀的 GUI 图形库，允许 Python 程序员很方便的创建完整的、功能健全的 GUI 用户界面。
* **Jython：**Jython 程序可以和 Java 无缝集成。除了一些标准模块，Jython 使用 Java 的模块。Jython 几乎拥有标准的Python 中不依赖于 C 语言的全部模块。比如，Jython 的用户界面将使用 Swing，AWT或者 SWT。Jython 可以被动态或静态地编译成 Java 字节码。

### Tkinter 编程

Tkinter 是 Python 的标准 GUI 库。Python 使用 Tkinter 可以快速的创建 GUI 应用程序。

由于 Tkinter 是内置到 python 的安装包中、只要安装好 Python 之后就能 import Tkinter 库、而且 IDLE 也是用 Tkinter 编写而成、对于简单的图形界面 Tkinter 还是能应付自如。

**注意**：Python3.x 版本使用的库名为 tkinter,即首写字母 T 为小写。

import tkinter

创建一个GUI程序

* 1、导入 Tkinter 模块
* 2、创建控件
* 3、指定这个控件的 master， 即这个控件属于哪一个
* 4、告诉 GM(geometry manager) 有一个控件产生了。

### 实例

#!/usr/bin/python # -\*- coding: UTF-8 -\*- import Tkinter top = Tkinter.Tk() # 进入消息循环 top.mainloop()

以上代码执行结果如下图:

## 数据库操作

## Web开发

### danjo框架介绍

### Flask框架介绍

## 网络爬虫

## numpy,

## pandas

## Matplotlib

图形pillow

# Python高级

# 附录

## Python IDE

### PyCharm

PyCharm 是由 JetBrains 打造的一款 Python IDE。

PyCharm 具备一般 **Python IDE** 的功能，比如：调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。

另外，PyCharm 还提供了一些很好的功能用于 Django 开发，同时支持 Google App Engine，更酷的是，PyCharm 支持 IronPython。

PyCharm 官方下载地址：<http://www.jetbrains.com/pycharm/download/>

### Sublime Text

Sublime Text 具有漂亮的用户界面和强大的功能，例如代码缩略图，Python 的插件，代码段等。还可自定义键绑定，菜单和工具栏。

Sublime Text 的主要功能包括：拼写检查，书签，完整的 Python API ， Goto 功能，即时项目切换，多选择，多窗口等等。

Sublime Text 是一个跨平台的编辑器，同时支持 Windows、Linux、Mac OS X等操作系统。

### Eclipse+Pydev

#### 1、安装Eclipse

Eclipse可以在它的官方网站[Eclipse.org](http://eclipse.org/)找到并下载，通常我们可以选择适合自己的Eclipse版本，比如Eclipse Classic。下载完成后解压到到你想安装的目录中即可。

当然在执行Eclipse之前，你必须确认安装了Java运行环境,即必须安装JRE或JDK，你可以到（<http://www.java.com/en/download/manual.jsp>）找到JRE下载并安装。

#### 2、安装Pydev

运行Eclipse之后，选择help-->Install new Software，如下图所示。

### [vscode](https://code.visualstudio.com/)

安装插件简单方便，可中文，打开速度快，缺点：相对pycharm，提示没那么好

### [atom](https://atom.io/)

安装插件简单方便，不常用