



# Python开发基础



# Python基础课程

0

认识Python与Python的基础知识

03

1

Python的控制流

50

2

Python的数据类型

82

3



## 1.1 认识python



# Python的优缺点

- 优点
  - 简单——Python是一种代表简单主义思想的语言。阅读一个良好的Python程序就感觉像是在读英语一样，尽管这个英语的要求非常严格！Python的这种伪代码本质是它最大的优点之一。它使你能够专注于解决问题而不是去搞明白语言本身。
  - 易学——就如同你即将看到的一样，Python极其容易上手。前面已经提到了，Python有极其简单的语法。
  - 免费、开源——Python是FLOSS（自由/开放源码软件）之一。简单地说，你可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。FLOSS是基于一个团体分享知识的概念。这是为什么Python如此优秀的原因之一——它是由一群希望看到一个更加优秀的Python的人创造并经常改进着的。
  - 高层语言——当你用Python语言编写程序的时候，你无需考虑诸如如何管理你的程序使用的内存一类的底层细节。
  - 可移植性——由于它的开源本质，Python已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。如果你小心地避免使用依赖于系统的特性，那么你的所有Python程序无需修改就可以在下述任何平台上面运行。这些平台包括Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE甚至还有PocketPC、Symbian以及Google基于linux开发的Android平台！

# Python的优缺点

- 解释性———这一点需要一些解释。一个用编译性语言比如C或C++写的程序可以从源文件（即C或C++语言）转换到一个你的计算机使用的语言（二进制代码，即0和1）。这个过程通过编译器和不同的标记、选项完成。当你运行你的程序的时候，连接/转载器软件把你的程序从硬盘复制到内存中并且运行。而Python语言写的程序不需要编译成二进制代码。你可以直接从源代码运行程序。在计算机内部，Python解释器把源代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。事实上，由于你不再需要担心如何编译程序，如何确保连接转载正确的库等等，所有这一切使得使用Python更加简单。由于你只需要把你的Python程序拷贝到另外一台计算机上，它就可以工作了，这也使得你的Python程序更加易于移植。
- 面向对象———Python既支持面向过程的编程也支持面向对象的编程。在“面向过程”的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在“面向对象”的语言中，程序是由数据和功能组合而成的对象构建起来的。与其他主要的语言如C++和Java相比，Python以一种非常强大又简单的方式实现面向对象编程。
- 可扩展性———如果你需要你的一段关键代码运行得更快或者希望某些算法不公开，你可以把你的部分程序用C或C++编写，然后在你的Python程序中使用它们。
- 丰富的库———Python标准库确实很庞大。它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI（图形用户界面）、Tk和其他与系统有关的操作。记住，只要安装了Python，所有这些功能都是可用的。这被称作Python的“功能齐全”理念。除了标准库以外，还有许多其他高质量的库，如wxPython、Twisted和Python图像库等等。
- 规范的代码———Python采用强制缩进的方式使得代码具有极佳的可读性。

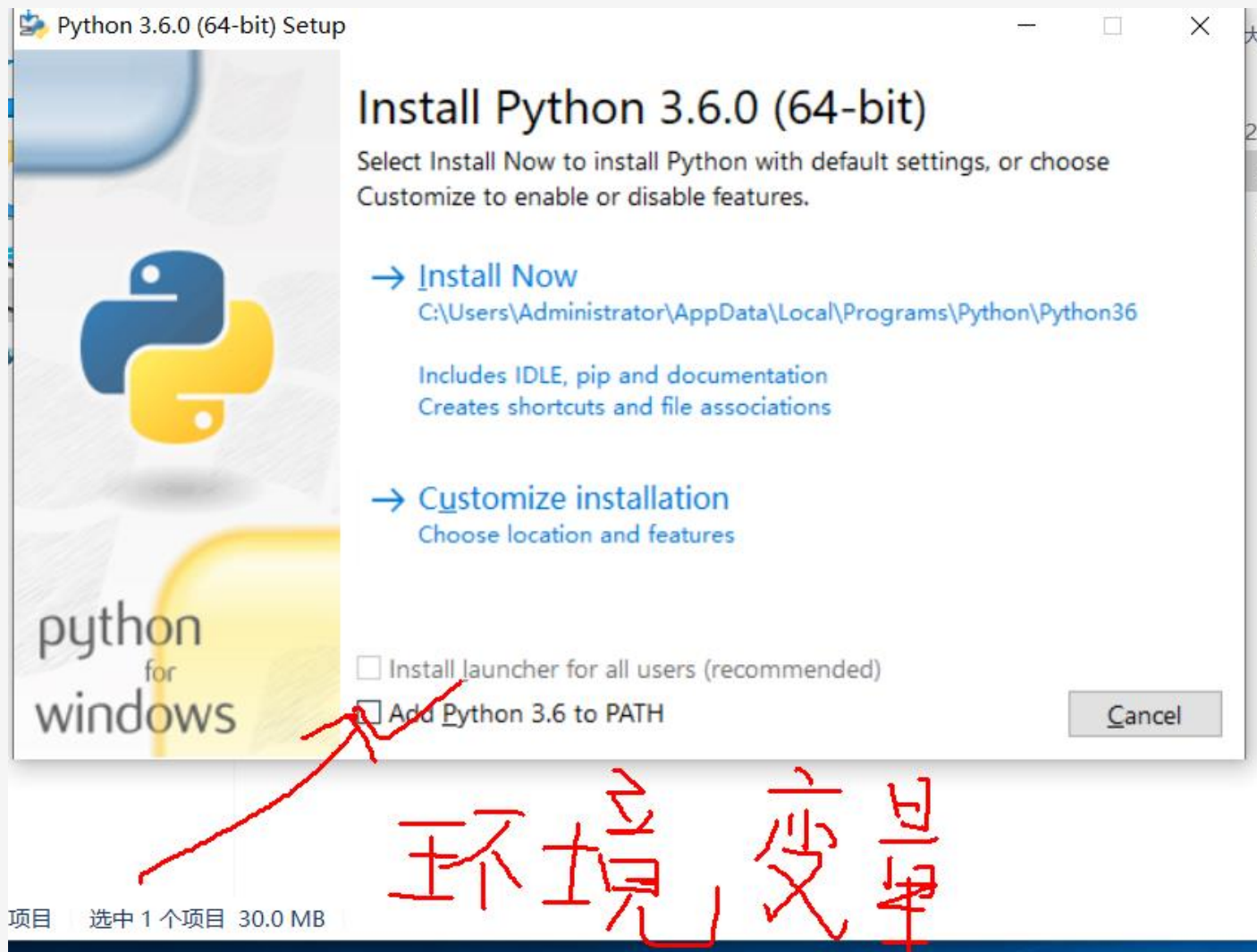
# Python的优缺点

- 缺点
  - 运行速度，有速度要求的话，用C++改写关键部分吧。
  - 国内市场较小（国内以python来做主要开发的，目前只有一些web2.0公司）。但时间推移，目前很多国内软件公司，尤其是游戏公司，也开始规模使用他。
  - 中文资料匮乏（好的python中文资料屈指可数）。托社区的福，有几本优秀的教材已经被翻译了，但入门级教材多，高级内容还是只能看英语版。
  - 构架选择太多（没有像C#这样的官方.net构架，也没有像ruby由于历史较短，构架开发的相对集中。Ruby on Rails 构架开发中小型web程序天下无敌）。不过这也从另一个侧面说明，python比较优秀，吸引的人才多，项目也多。

# Python的应用场景

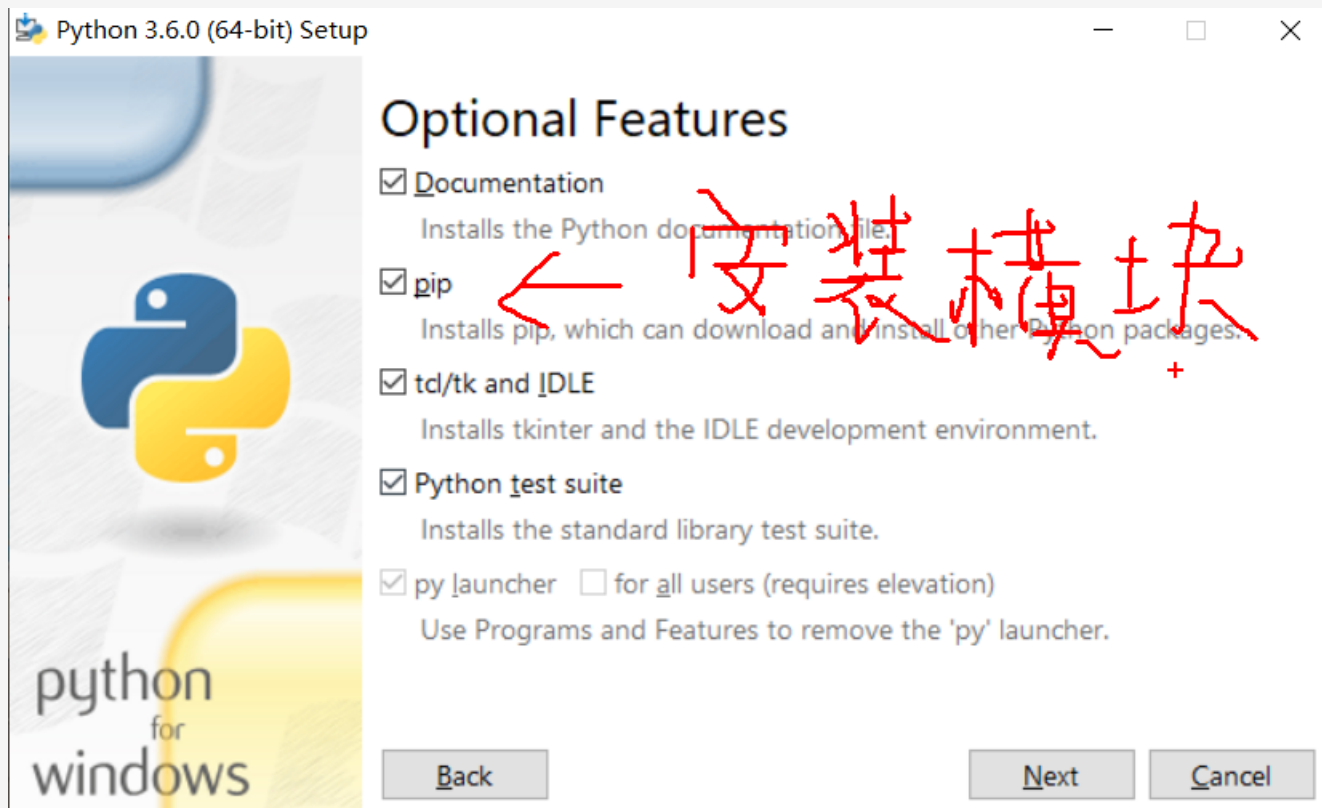
- Web应用开发
- 操作系统管理、服务器运维的自动化脚本
- 科学计算
- 桌面软件
- 服务器软件（网络软件）
- 游戏
- 构思实现，产品早期原型和迭代

# Python运行环境的安装





# Python运行环境的安装

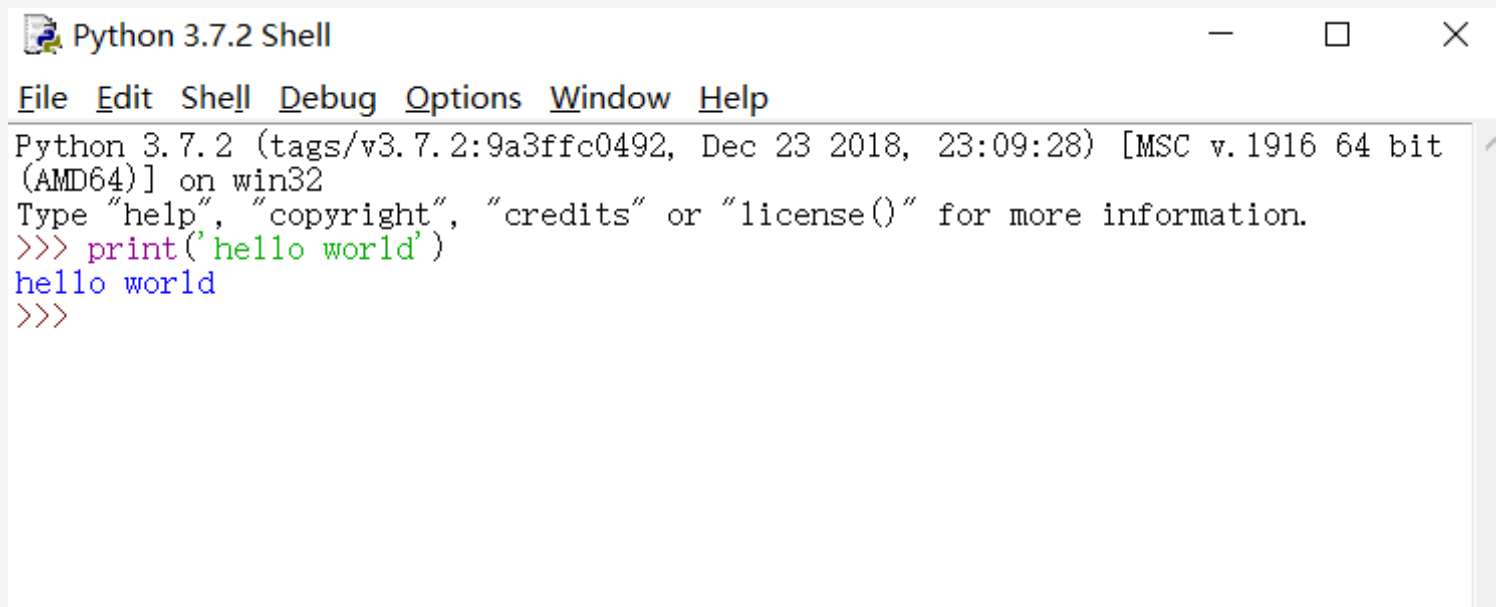




## 1.2 第一个Python程序



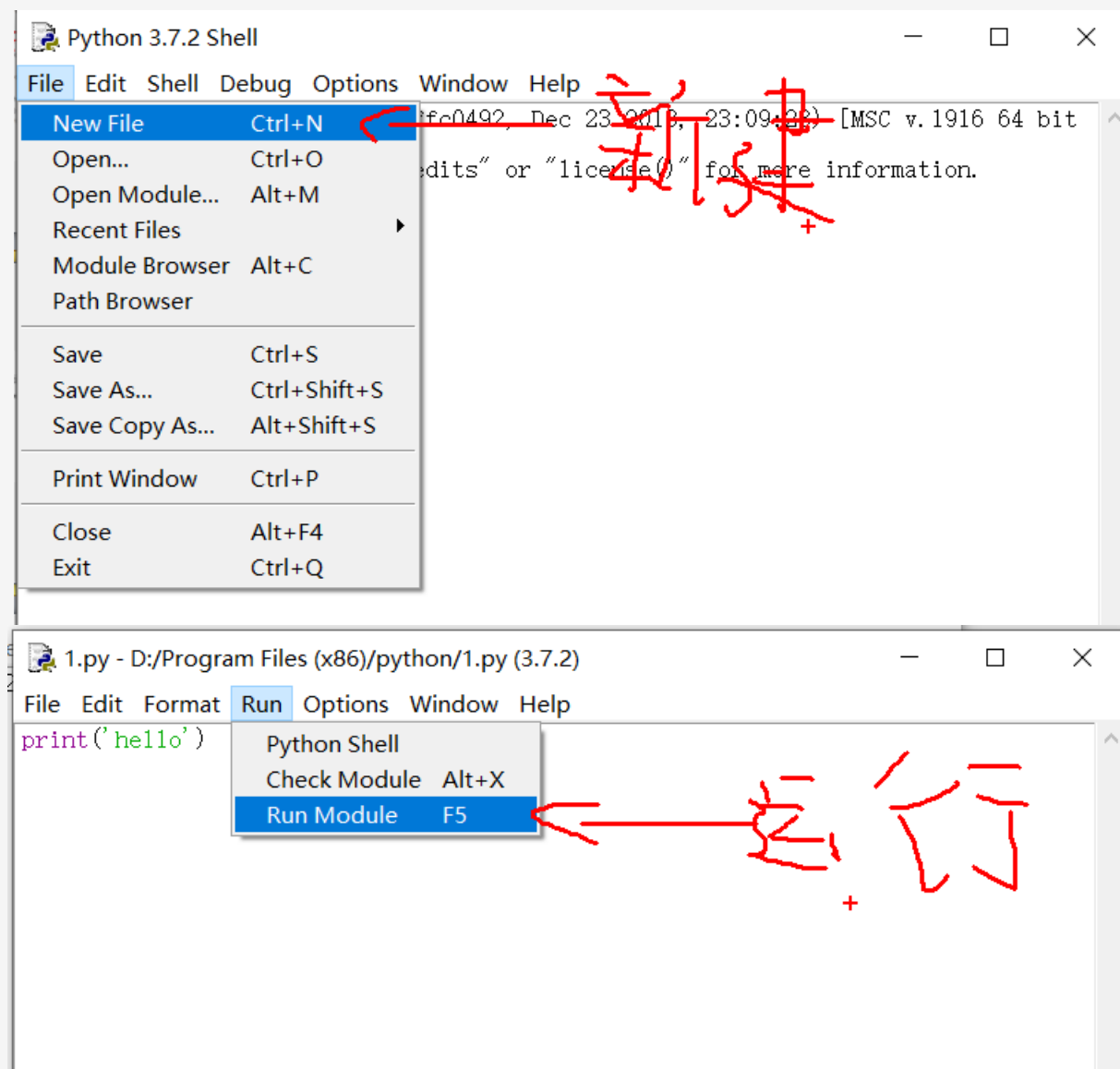
# Python的shell



The image shows a screenshot of a Windows application window titled "Python 3.7.2 Shell". The window has a standard menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the following content: "Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32", followed by "Type 'help', 'copyright', 'credits' or 'license()' for more information." Then, a prompt ">>>" is followed by the command "print('hello world')", which is executed, resulting in the output "hello world" on the next line, and another prompt ">>>" on the line below. The text is color-coded: the prompt is blue, the command is purple, and the output is blue.

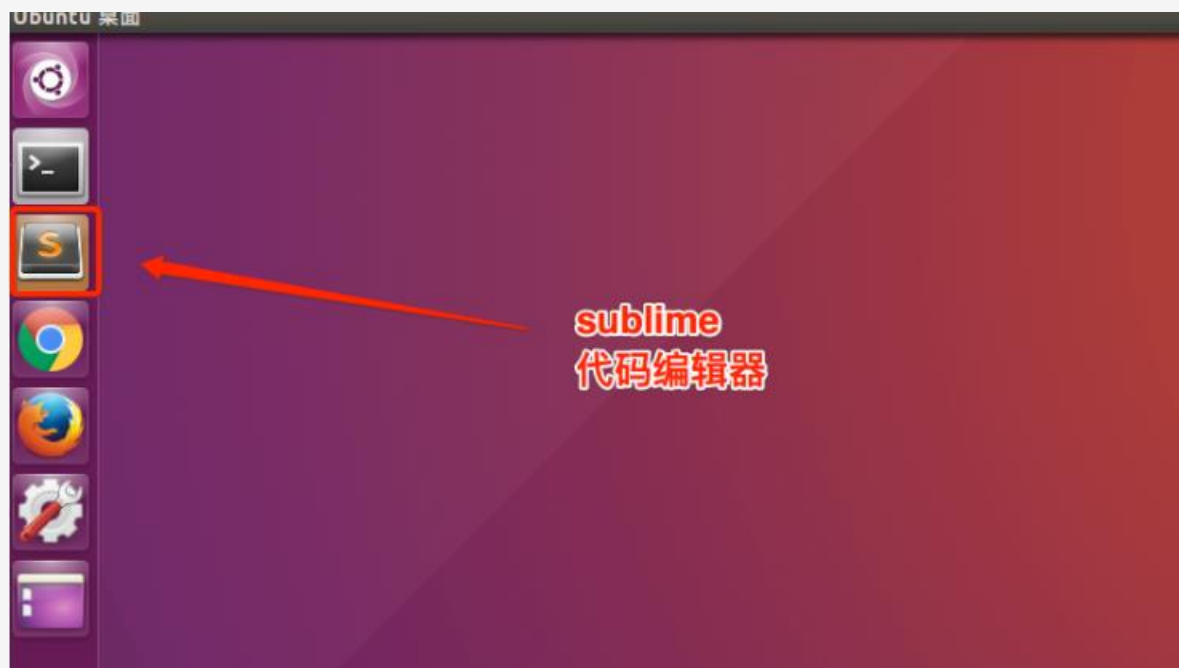
```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('hello world')
hello world
>>>
```

# Python的shell



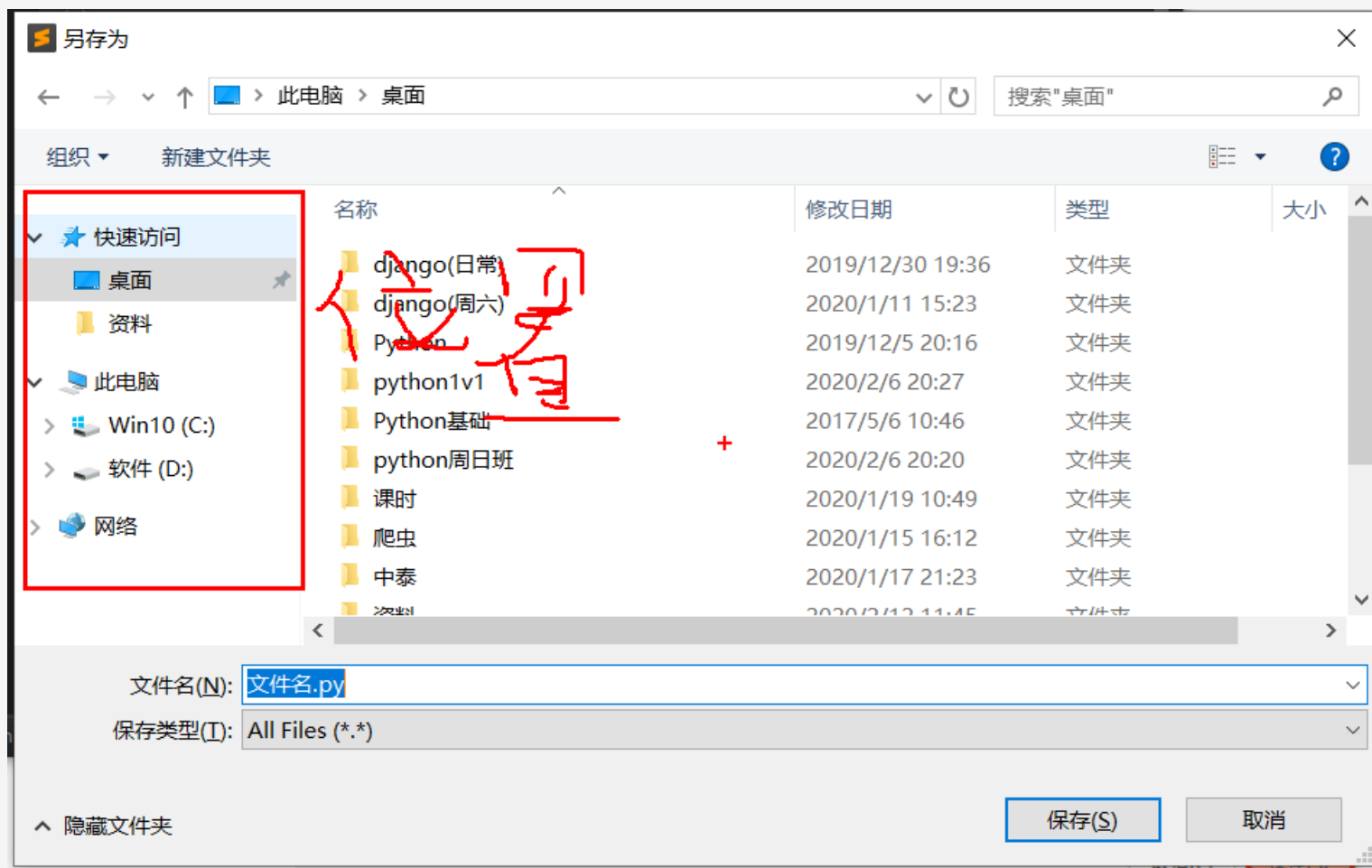
# Python的编辑器

- 编辑软件sublime



# Python的编辑器

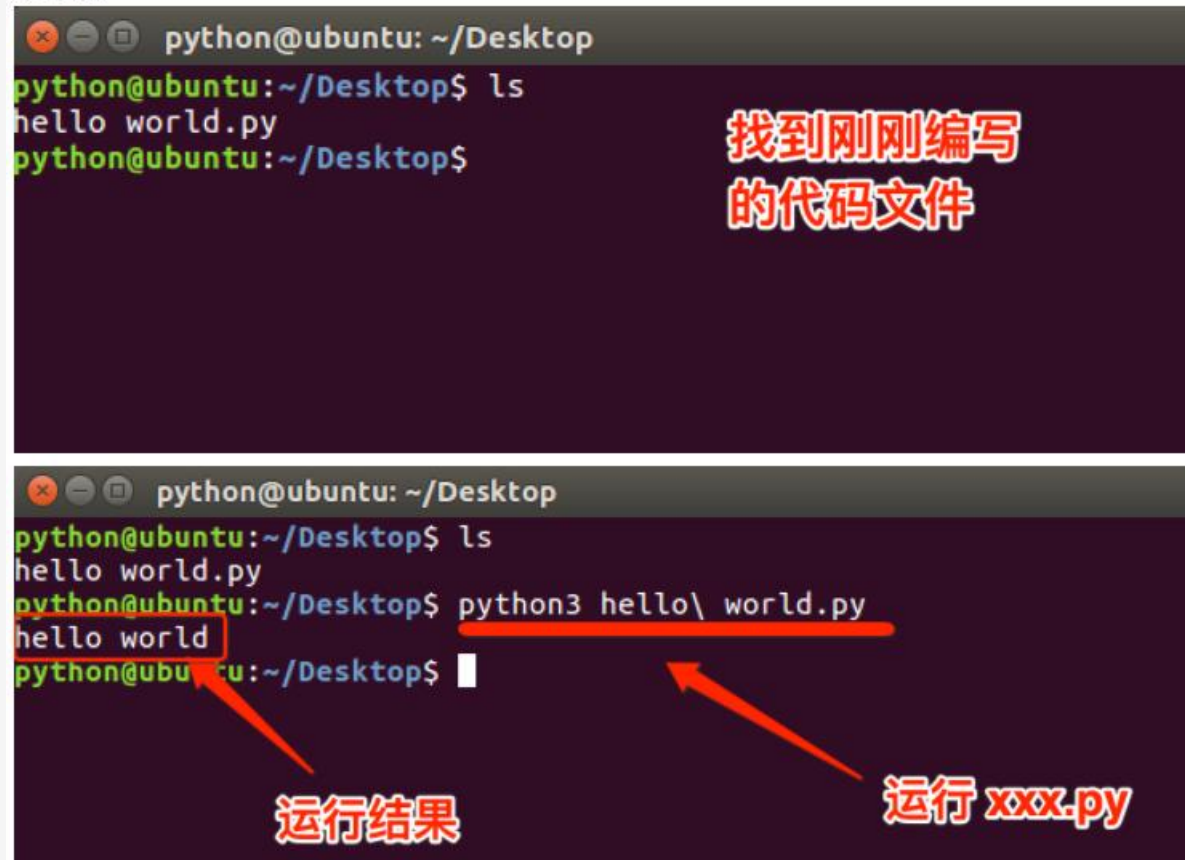
- 编辑软件sublime



# Python的编辑器

- 编辑软件sublime

运行程序



The image displays two terminal windows from a user named 'python' on an 'ubuntu' machine, located in the directory '~/Desktop'.

The top terminal window shows the command `ls` being executed, which lists the file `hello world.py`. A red annotation "找到刚刚编写的代码文件" (Found the code file just written) points to the output of the `ls` command.

The bottom terminal window shows the command `python3 hello\ world.py` being executed. The output of the command is `hello world`, which is highlighted with a red box. A red annotation "运行结果" (Execution result) points to this output. Another red annotation "运行 xxx.py" (Run xxx.py) points to the command being executed.

# 小结

- 对于编写python程序，上面有3种方法，那到实际开发中哪一种用的比较多呢？一般是用第2、3种，即保存在xxx.py文件中，这样可以直接下一次执行运行；而如果用第一种方法的话，每一次运行程序都需要重新进行输入，费时费力
- 练一练：
- 编写一个程序，输出itcast.cn





## 1.3 注释



- <1> 看以下程序示例（未使用注释）

```
2048.py
86 def is_gameover(self):
87     return not any(self.move_is_possible(move) for move in actions)
88
89 def draw(self, screen):
90     help_string1 = '(W)Up (S)Down (A)Left (D)Right'
91     help_string2 = '          (R)Restart (Q)Exit'
92     gameover_string = '          GAME OVER'
93     win_string = '          YOU WIN!'
94     def cast(string):
95         screen.addstr(string + '\n')
96
97     def draw_hor_separator():
98         line = '+' + ('+-----' * self.width + '+')[1:]
99         separator = defaultdict(lambda: line)
100         if not hasattr(draw_hor_separator, "counter"):
101             draw_hor_separator.counter = 0
102         cast(separator[draw_hor_separator.counter])
103         draw_hor_separator.counter += 1
104
105     def draw_row(row):
106         cast(''.join('|{: ^5} '.format(num) if num > 0 else '|' for num in row))
107
108     screen.clear()
109     cast('SCORE: ' + str(self.score))
110     if 0 != self.highscore:
111         cast('HGHSORE: ' + str(self.highscore))
112     for row in self.field:
113         draw_hor_separator()
114         draw_row(row)
115     draw_hor_separator()
```

- <2> 看以下程序示例（使用注释）

```
2048.py
158 def main(stdscr):
159     def init():
160         #重置游戏棋盘
161         game_field.reset()
162         return 'Game'
163
164     def not_game(state):
165         #画出 GameOver 或者 Win 的界面
166         game_field.draw(stdscr)
167         #读取用户输入得到action, 判断是重启游戏还是结束游戏
168         action = get_user_action(stdscr)
169         responses = defaultdict(lambda: state) #默认是当前状态, 没有行为就会一直在当前界面循环
170         responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态
171         return responses[action]
172
173     def game():
174         #画出当前棋盘状态
175         game_field.draw(stdscr)
176         #读取用户输入得到action
177         action = get_user_action(stdscr)
178
179         if action == 'Restart':
180             return 'Init'
181         if action == 'Exit':
182             return 'Exit'
183         if game_field.move(action): # move successful
184             if game_field.is_win():
185                 return 'Win'
186             if game_field.is_gameover():
187                 return 'Gameover'
188     return 'Game'
```

以#开头的就是注释

- **<3> 小总结（注释的作用）**
- 通过用自己熟悉的语言，在程序中对某些代码进行标注说明，这就是注释的作用，能够大大增强程序的可读性，除了用来表示解释说明外，还可以用来逻辑删除代码，比如一部分代码不想要运行时，可以注释起来。

- <1> 单行注释
- 以#开头，#右边的所有东西当做说明，而不是真正要执行的程序，起辅助说明作用

```
# 我是注释，可以在里写一些功能说明之类的哦  
print('hello world')
```

- <2> 多行注释
- '''
- 我是多行注释，可以写很多很多行的功能说明 这就是我牛X指出 哈哈哈哈哈。。。
- '''
- '''
- 下面的代码完成，
- 打印一首诗 名字叫做：
- 春江花月夜 作者，忘了
- '''



## 1.4 标识符和关键字



# 标识符和关键字

- <1>标示符
- 什么是标示符，看下图:



- 开发人员在程序中自定义的一些符号和名称
- 标示符是自己定义的,如变量名、函数名等



# 标识符和关键字

- <2>标示符的规则
- 标示符由字母、下划线和数字组成，且数字不能开头
- 思考：下面的标示符哪些是正确的，哪些不正确为什么

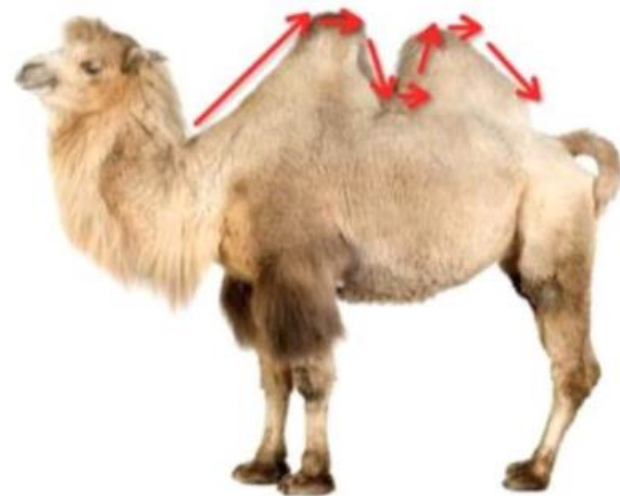
python中的标识符是区分大小写的

Andy  $\neq$  andy

```
fromNo12
from#12
my_Boolean
my-Boolean
Obj2
2ndObj
myInt
test1
Mike2jack
My_tExt
_test
test!32
haha(da)tt
int
jack_rose
jack&rose
GUI
G.U.I
```

# 标识符和关键字

- <3>命名规则
- 见名知意
- 起一个有意义的名字，尽量做到看一眼就知道是什么意思 (提高代码可读性) 比如: 名字 就定义为 `name`, 定义学生 用 `student`
- 驼峰命名法



如:

`userName`

`userLoginFlag`

小驼峰式命名法 ( lower camel case ) : 第一个单词以小写字母开始; 第二个单词的首字母大写, 例如: `myName`、`aDog`

大驼峰式命名法 ( upper camel case ) : 每一个单字的首字母都采用大写字母, 例如: `FirstName`、`LastName`

不过在程序员中还有一种命名法比较流行, 就是用下划线 "\_" 来连接所有的单词, 比如 `send_buf`

# 标识符和关键字

- <4>关键字
- 什么是关键字
- python一些具有特殊功能的标示符，这就是所谓的关键字
- 关键字，是python已经使用的了，所以不允许开发者自己定义和关键字相同的名字的标示符
- 查看关键字:

and	as	assert	break	class	continue	def	del
elif	else	except	exec	finally	for	from	global
if	in	import	is	lambda	not	or	pass
print	raise	return	try	while	with	yield	

可以通过以下命令进行查看当前系统中python的关键字

```
root@ubuntu: ~/ftp/share
root@ubuntu:~/ftp/share# python3
Python 3.5.2 (default, Jul 5 2016, 12:43:10)
[GCC 5.4.0 20160609] on linux
Type "help()" "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> 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']
>>>
```



## 1.5 变量及类型



- <1>变量的定义

- 在程序中，有时我们需要对2个数据进行求和，那么该怎样做呢？
- 大家类比一下现实生活中，比如去超市买东西，往往咱们需要一个菜篮子，用来进行存储物品，等到所有的物品都购买完成后，在收银台进行结账即可
- 如果在程序中，需要把2个数据，或者多个数据进行求和的话，那么就需要把这些数据先存储起来，然后把它们累加起来即可
- 在Python中，存储一个数据，需要一个叫做变量的东西，如下示例:

```
num1 = 100 #num1就是一个变量，就好一个小菜篮子
```

```
num2 = 87 #num2也是一个变量
```

```
result = num1 + num2 #把num1和num2这两个“菜篮子”中的数据进行累加，然后放到 result变量中
```

- 说明:所谓变量，可以理解为菜篮子，如果需要存储多个数据，最简单的方式是有多个变量，当然了也可以使用一个
- 程序就是用来处理数据的，而变量就是用来存储数据的

# 变量以及类型

- 想一想：我们应该让变量占用多大的空间，保存什么样的数据？
- <2>变量的类型
- 生活中的“类型”的例子：



# 变量以及类型

- 程序中:
- 为了更充分的利用内存空间以及更有效率的管理内存，变量是有不同的类型的，如下所示:



- 怎样知道一个变量的类型呢？
- 在python中，只要定义了一个变量，而且它有数据，那么它的类型就已经确定了，不需要咱们开发者主动的去说明它的类型，系统会自动辨别
- 可以使用`type(变量的名字)`，来查看变量的类型

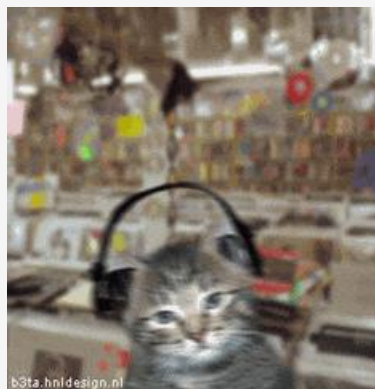
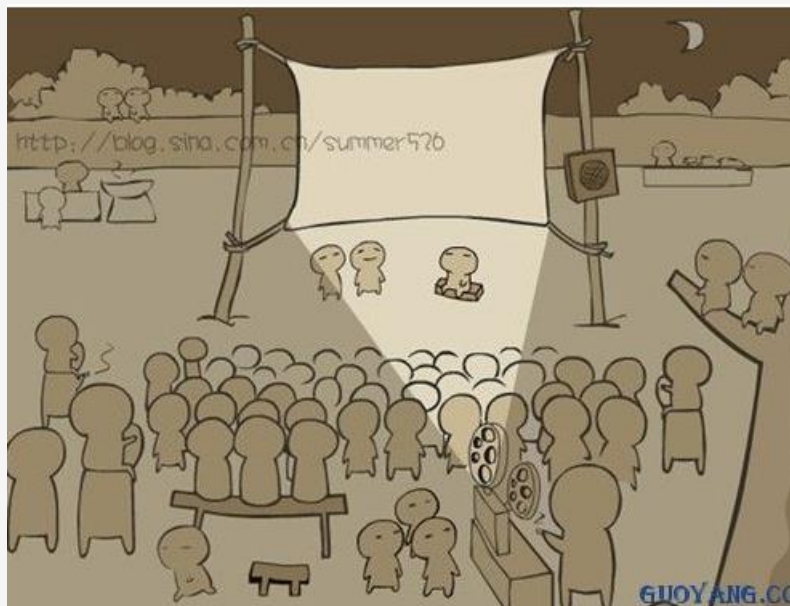




1.5 输出

# 输出

- 1. 普通的输出
- 生活中的“输出”



- 软件中的“输出”



- python中变量的输出

```
# 打印提示
print('hello world')
print('给我的卡---印度语，你好的意思')
```

- 2. 格式化输出
- <1>格式化操作的目的
- 比如有以下代码:

```
pirnt("我今年10岁")  
pirnt("我今年11岁")  
pirnt("我今年12岁")  
...
```

- 想一想:
- 在输出年龄的时候, 用了多次"我今年xx岁", 能否简化一下程序呢???
- 答:
- 字符串格式化

- 看如下代码:

```
age = 10
print("我今年%d岁"%age)

age += 1
print("我今年%d岁"%age)

age += 1
print("我今年%d岁"%age)

...
```

- 在程序中，看到了%这样的操作符，这就是Python中格式化输出。

- <3>常用的格式符号
- 下面是完整的，它可以与%符号使用列表:

格式符号	转换
%c	字符
%s	通过str() 字符串转换来格式化
%i	有符号十进制整数
%d	有符号十进制整数
%u	无符号十进制整数
%o	八进制整数
%x	十六进制整数 ( 小写字母 )
%X	十六进制整数 ( 大写字母 )
%e	索引符号 ( 小写'e' )
%E	索引符号 ( 大写'E' )
%f	浮点实数
%g	%f和 %e 的简写
%G	%f和 %E的简写

- **3. 换行输出**

- 在输出的时候，如果有\n那么，此时\n后的内容会在另外一行显示。

- **4.制表符输出**

- 在输出的时候，如果有\t那么，此时\t后的内容会在后面自动空格。

- **5. 练一练**

- 编写代码完成以下名片的显示

```
=====
姓名: dongGe
QQ:xxxxxxx
手机号:131xxxxxx
公司地址:北京市xxxx
=====
```



## 1.6 输入



- 1. python3版本中



- 咱们在银行ATM机器前取钱时，肯定需要输入密码，对不？
- 那么怎样才能让程序知道咱们刚刚输入的是什么呢？
- 大家应该知道了，如果要完成ATM机取钱这件事情，需要先从键盘中输入一个数据，然后用一个变量来保存，是不是很好理解啊

- **2.input()**
- input()接受表达式输入，并把表达式的结果赋值给等号左边的变量

```
>>> a = input()
123
>>> a
123
>>> type(a)
<type 'int'>
>>> a = input()
.
```



## 1.7 数据类型转换



- 常用的数据类型转换

- 举例

```
a = '100' # 此时a的类型是一个字符串，里面存放了100这3个字符
b = int(a) # 此时b的类型是整型，里面存放的是数字100

print("a=%d"%b)
```

函数	说明
int(x [,base ])	将x转换为一个整数
long(x [,base ])	将x转换为一个长整数
float(x )	将x转换到一个浮点数
complex(real [,imag ])	创建一个复数
str(x )	将对象 x 转换为字符串
repr(x )	将对象 x 转换为表达式字符串
eval(str )	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s )	将序列 s 转换为一个元组
list(s )	将序列 s 转换为一个列表
chr(x )	将一个整数转换为一个字符
unichr(x )	将一个整数转换为Unicode字符
ord(x )	将一个字符转换为它的整数值
hex(x )	将一个整数转换为一个十六进制字符串
oct(x )	将一个整数转换为一个八进制字符串



## 1.8 运算符



- python支持以下几种运算符
- 算术运算符

运算符	描述	实例
+	加	两个对象相加 a + b 输出结果 30
-	减	得到负数或是一个数减去另一个数 a - b 输出结果 -10
*	乘	两个数相乘或是返回一个被重复若干次的字符串 a * b 输出结果 200
/	除	x除以y b / a 输出结果 2
//	取整除	返回商的整数部分 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0
%	取余	返回除法的余数 b % a 输出结果 0
**	幂	返回x的y次幂 a**b 为10的20次方， 输出结果 100000000000000000000

```
>>> 9/2.0
4.5
>>> 9//2.0
4.0
```

- 赋值运算符

运算符	描述	实例
=	赋值运算符	把=号右边的结果给左边的变量 num=1+2*3 结果num的值为7

```
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

- 复合赋值运算符

运算符	描述	实例
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

- 比较(即关系)运算符

运算符	描述	示例
==	检查两个操作数的值是否相等，如果是则条件变为真。	如a=3,b=3则 ( a == b) 为 true.
!=	检查两个操作数的值是否相等，如果值不相等，则条件变为真。	如a=1,b=3则(a != b) 为 true.
<>	检查两个操作数的值是否相等，如果值不相等，则条件变为真。	如a=1,b=3则(a <> b) 为 true。这个类似于 != 运算符
>	检查左操作数的值是否大于右操作数的值，如果是，则条件成立。	如a=7,b=3则(a > b) 为 true.
<	检查左操作数的值是否小于右操作数的值，如果是，则条件成立。	如a=7,b=3则(a < b) 为 false.
>=	检查左操作数的值是否大于或等于右操作数的值，如果是，则条件成立。	如a=3,b=3则(a >= b) 为 true.
<=	检查左操作数的值是否小于或等于右操作数的值，如果是，则条件成立。	如a=3,b=3则(a <= b) 为 true.



- 逻辑运算符

运算符	逻辑表达式	描述	实例
and	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
or	x or y	布尔"或" - 如果 x 是 True，它返回 True，否则它返回 y 的计算值。	(a or b) 返回 10。
not	not x	布尔"非" - 如果 x 为 True，返回 False。如果 x 为 False，它返回 True。	not(a and b) 返回 False



## 2.1 判断语句



# 判断语句

- 判断语句介绍
- <1>生活中的判断场景
- 火车站安检



- 上网吧



- <2>开发中的判断场景
- 重要日期判断

```
if 今天是周六或者周日:  
    约妹子  
  
if 今天是情人节:  
    买玫瑰  
  
if 今天发工资:  
  
    先还信用卡的钱  
  
    if 有剩余:  
  
        又可以happy了，O(n_n)O哈哈~  
  
    else:  
  
        噢，no。。。还的等30天
```

- 小总结:
- 如果某些条件满足，才能做某件事情，而不满足时不允许做，这就是所谓的判断
- 不仅生活中有，在软件开发中“判断”功能也经常会被用到

- **if-else**
- **<1>if-else的使用格式**

```
if 条件:  
    满足条件时要做的事情1  
    满足条件时要做的事情2  
    满足条件时要做的事情3  
    ...(省略)...  
else:  
    不满足条件时要做的事情1  
    不满足条件时要做的事情2  
    不满足条件时要做的事情3  
    ...(省略)...
```

- **<2>练一练**
- 要求：从键盘输入刀子的长度，如果刀子长度没有超过10cm，则允许上火车，否则不允许上火车

- **Elif**

- 想一想:

- if能完成当xxx时做事情

- if-else能完成当xxx时做事情1，否则做事情2

- 如果有这样一种情况：当xxx1时做事情1，当xxx2时做事情2，当xxx3时做事情3，那该怎么实现呢？

- 答:

- elif

- **<1> elif的功能**

elif的使用格式如下:

```
if xxx1:  
    事情1  
elif xxx2:  
    事情2  
elif xxx3:  
    事情3
```

- 说明:
- 当xxx1满足时, 执行事情1, 然后整个if结束
- 当xxx1不满足时, 那么判断xxx2, 如果xxx2满足, 则执行事情2, 然后整个if结束
- 当xxx1不满足时, xxx2也不满足, 如果xxx3满足, 则执行事情3, 然后整个if结束

- **demo:** 输入分数，判断等级

```
score = 77

if score >= 90 and score <= 100:
    print('本次考试，等级为A')
elif score >= 80 and score < 90:
    print('本次考试，等级为B')
elif score >= 70 and score < 80:
    print('本次考试，等级为C')
elif score >= 60 and score < 70:
    print('本次考试，等级为D')
elif score >= 0 and score < 60:
    print('本次考试，等级为E')
```



- <2> 注意点

```
if 性别为男性:  
    输出男性的特征  
    ...  
elif 性别为女性:  
    输出女性的特征  
    ...  
else:  
    第三种性别的特征  
    ...
```

- 说明:
- 当“性别为男性”满足时，执行“输出男性的特征”的相关代码
- 当“性别为男性”不满足时，如果“性别为女性”满足，则执行“输出女性的特征”的相关代码
- 当“性别为男性”不满足，“性别为女性”也不满足，那么就默认执行else后面的代码，即“第三种性别的特征”相关代码

- **if嵌套**
- 通过学习if的基本用法，已经知道了
- 当需要满足条件去做事情的这种情况需要使用if
- 当满足条件时做事情A，不满足条件做事情B的这种情况使用if-else

想一想：

坐火车或者地铁的实际情况是：先进行安检如果安检通过才会判断是否有车票，或者是先检查是否有车票之后才会进行安检，即实际的情况某个判断是再另外一个判断成立的基础上进行的，这样的情况该怎么解决呢？

答：

if嵌套

- <1>if嵌套的格式

```
if 条件1:  
  
    满足条件1 做的事情1  
    满足条件1 做的事情2  
    ... (省略) ...  
  
    if 条件2:  
        满足条件2 做的事情1  
        满足条件2 做的事情2  
        ... (省略) ...
```

- 说明外层的if判断，也可以是if-else
- 内层的if判断，也可以是if-else
- 根据实际开发的情况，进行选择

- <2>if嵌套的应用
- demo:

```
chePiao = 1      # 用1代表有车票, 0代表没有车票
daoLenght = 9    # 刀子的长度, 单位为cm

if chePiao == 1:
    print("有车票, 可以进站")
    if daoLenght < 10:
        print("通过安检")
        print("终于可以见到Ta了, 美滋滋~~~")
    else:
        print("没有通过安检")
        print("刀子的长度超过规定, 等待警察处理...")
else:
    print("没有车票, 不能进站")
    print("亲爱的, 那就下次见了, 一票难求啊~~~~(>_<)~~~~")
```

结果1: chePiao = 1; daoLenght = 9

有车票, 可以进站  
通过安检  
终于可以见到Ta了, 美滋滋~~~

结果2: chePiao = 1; daoLenght = 20

有车票, 可以进站  
没有通过安检  
刀子的长度超过规定, 等待警察处理...

结果3: chePiao = 0; daoLenght = 9

没有车票, 不能进站  
亲爱的, 那就下次见了, 一票难求啊~~~~(>\_<)~~~~

结果4: chePiao = 0; daoLenght = 20

没有车票, 不能进站  
亲爱的, 那就下次见了, 一票难求啊~~~~(>\_<)~~~~

想一想: 为什么结果3和结果4相同???

- **<3>练一练**
  - 情节描述：上公交车，并且可以有座位坐下
  - 要求：输入公交卡当前的余额，只要超过2元，就可以上公交车；如果空座位的数量大于0，就可以坐下



## 2.2 循环语句while



# 循环语句-while

- 循环介绍
- <1>生活中的循环场景
- 跑道



- CF加特林



# 循环语句-while

- <2>软件开发中循环的使用场景
- 跟媳妇承认错误，说一万遍“媳妇儿，我错了”

```
print("媳妇儿，我错了")  
print("媳妇儿，我错了")  
print("媳妇儿，我错了")  
...(还有99997遍)...
```

- 使用循环语句一句话搞定

```
i = 0  
while i<10000:  
    print("媳妇儿，我错了")  
    i+=1
```



# 循环语句-while

- <3>小总结
  - 一般情况下，需要多次重复执行的代码，都可以用循环的方式来完成
  - 循环不是必须要使用的，但是为了提高代码的重复使用率，所以有经验的开发者都会采用循环

# 循环语句-while

- while循环应用
- 1. 计算1~100的累积和（包含1和100）
- 2. 计算1~100之间偶数的累积和（包含1和100）
- 参考代码如下：

```
#encoding=utf-8

i = 1
sum = 0
while i<=100:
    sum = sum + i
    i += 1

print("1~100的累积和为:%d"%sum)
```

```
#encoding=utf-8

i = 1
sum = 0
while i<=100:
    if i%2 == 0:
        sum = sum + i
    i+=1

print("1~100的累积和为:%d"%sum)
```



## 2.3 循环语句for



# 循环语句-for

- **for**循环
- 像while循环一样，for可以完成循环的功能。
- 在Python中 for循环可以遍历任何序列的项目，如一个列表或者一个字符串等。

- **for**循环的四种形式
- **NO1:**
- **for** 临时变量 in range(start, end):
- 循环满足条件时执行的代码
- **start:** 起始数 包含数字
- **end:** 结束数 不包含数字
- **NO2:**
- **for** 临时变量 in range(start, end, step):
- 循环满足条件时执行的代码
- **step:** 代表步长，每次的累计数量

# 循环语句-for

- **for**循环的四种形式
- **NO3:**
- for 临时变量 in range(num):
- 循环满足条件时执行的代码
- **NO4:**
- for 临时变量 in 列表或字符串:
- 循环满足条件时执行的代码

# 循环语句-for

- **for嵌套**
- 语法格式如下：
- `for 临时变量 in xxxx:`
- `for 临时变量2 in xxxx:`
- 循环语句
- **for嵌套应用一**
- 要求：打印如下图形：

```
*  
* *  
* * *  
* * * *  
* * * * *
```

# 循环语句-for

- for嵌套应用二：九九乘法表

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```





## 2.4 跳转语句



# 跳转语句-break

- break和continue
- 1. break
- <1> for循环
- 带有break的循环示例如下:

```
name = 'dongGe'

for x in name:
    print('----')
    if x == 'g':
        break
    print(x)
```

运行结果:

```
>>> name = 'dongGe'
>>>
>>> for x in name:
...     print('----')
...     if x == 'g':
...         break
...     print(x)
...
----
d
----
o
----
n
----
>>>
```

# 跳转语句-break

- <2> while循环
- 带有break的循环示例如下:

```
i = 0

while i<10:
    i = i+1
    print('----')
    if i==5:
        break
    print(i)
```

运行结果:

```
>>> i = 0
>>>
>>> while i<10:
...     i = i+1
...     print('----')
...     if i==5:
...         break
...     print(i)
...
----
1
----
2
----
3
----
4
----
>>> █
```

# 跳转语句-continue

- **2. continue**
- **<1> for循环**
- 带有continue的循环示例如下:

```
name = 'dongGe'

for x in name:
    print('----')
    if x == 'g':
        continue
    print(x)
```

运行结果:

```
>>> name = 'dongGe'
>>>
>>> for x in name:
...     print('----')
...     if x == 'g':
...         continue
...     print(x)
...
----
d
----
o
----
n
----
G
----
e
>>>
```

# 跳转语句-continue

- <2> while 循环
- 带有 continue 的循环示例如下:

```
i = 0

while i<10:
    i = i+1
    print('----')
    if i==5:
        continue
    print(i)
```

运行结果:

```
>>> i = 0
>>>
>>> while i<10:
...     i = i+1
...     print('----')
...     if i==5:
...         continue
...     print(i)
...
----
1
----
2
----
3
----
4
----
6
----
7
----
8
----
9
----
10
>>> █
```

- 小总结: continue 的作用: 用来结束本次循环, 紧接着执行下一次的循环

# 跳转语句-continue

- **3. 注意点**
- break/continue只能用在循环中，除此以外不能单独使用
- break/continue在嵌套循环中，只对最近的一层循环起作用



## 2.5 总结



# 跳转语句-continue

- 总结
- if往往用来对条件是否满足进行判断
- if有4中基本的使用方法:

## 1. 基本方法

```
if 条件:  
    满足时做的事情
```

## 2. 满足与否执行不同的事情

```
if 条件:  
    满足时做的事情  
  
else:  
    不满足时做的事情
```

## 3. 多个条件的判断

```
if 条件:  
    满足时做的事情  
  
elif 条件2:  
    满足条件2时做的事情  
  
elif 条件3:  
    满足条件3时做的事情  
  
else:  
    条件都不满足时做的事情
```

## 4. 嵌套

```
if 条件:  
    满足时做的事情  
  
    这里还可以放入其他任何形式的if判断语句
```



# 跳转语句-continue

- 总结
- while循环一般通过数值是否满足来确定循环的条件

```
i = 0
while i<10:
    print("hello")
    i+=1
```

- for循环一般是对能保存多个数据的变量，进行便利

```
name = 'dongGe'

for x in name:
    print(x)
```

- if、while、for等其他语句可以随意组合，这样往往就完成了复杂的功能



## 3.1 字符串介绍



- 字符串介绍

- 想一想：

当打来浏览器登录某些网站的时候，需要输入密码，浏览器把密码传送到服务器后，服务器会对密码进行验证，其验证过程是把之前保存的密码与本次传递过去的密码进行对比，如果相等，那么就认为密码正确，否则就认为不对；服务器既然想要存储这些密码可以用数据库（比如MySQL），当然为了简单起见，咱们可以先找个变量把密码存储起来即可；那么怎样存储带有字母的密码呢？

- 答：

字符串

- <1>python中字符串的格式
- 如下定义的变量a，存储的是数字类型的值
- `a = 100`
- 如下定义的变量b，存储的是字符串类型的值
- `b = "hello itcast.cn"`
- 或者
- `b = 'hello itcast.cn'`
- 小总结：
- 双引号或者单引号中的数据，就是字符串



## 3.2 字符串的使用



- 字符串输出
- Demo

```
name = 'xiaoming'  
position = '讲师'  
address = '北京市昌平区建材城西路金燕龙办公楼1层'  
  
print('-----')  
print("姓名：%s"%name)  
print("职位：%s"%position)  
print("公司地址：%s"%address)  
print('-----')
```

- 结果：

```
-----  
姓名： xiaoming  
职位： 讲师  
公司地址： 北京市昌平区建材城西路金燕龙办公楼1层  
-----
```

- 字符串输入
- 之前在学习input的时候，通过它能够完成从键盘获取数据，然后保存到指定的变量中；
- 注意：input获取的数据，都以字符串的方式进行保存，即使输入的是数字，那么也是以字符串方式保存

- 字符串的拼接

```
In [8]: a = "lao"
In [9]: b = "wang"
In [10]: c = "zhao"
In [11]: d = a+b
In [12]: d
Out[12]: 'laowang'

In [13]: A = 100
In [14]: B = 200
In [15]: C = A+B
In [16]: C
Out[16]: 300

In [17]: e = "===" + a + b + "==="
In [18]: e
Out[18]: '===laowang==='

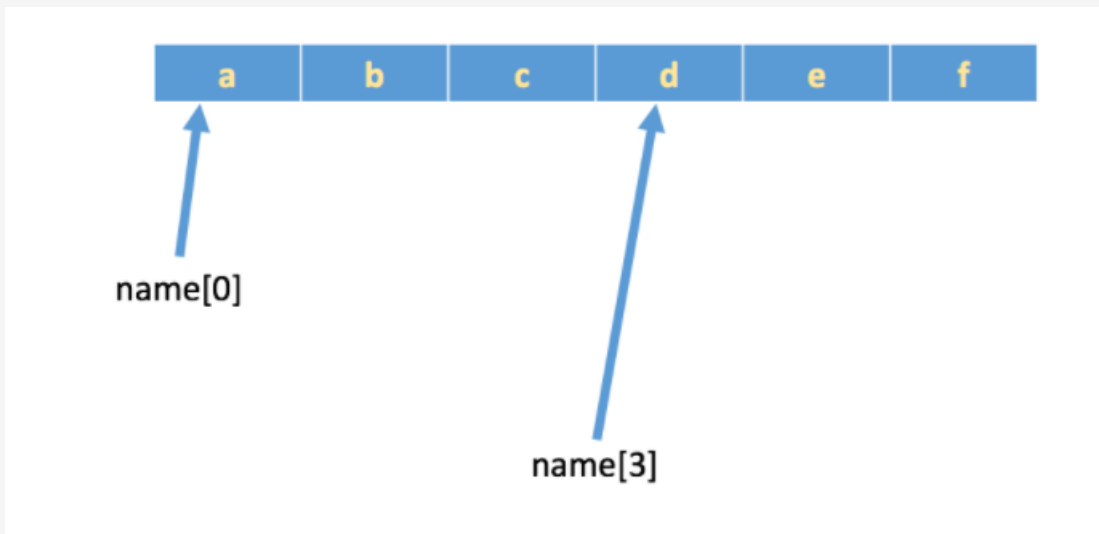
In [19]:
In [19]: f = "===%s==="%(a+b)
In [20]: f
Out[20]: '===laowang==='
```



- 下标和切片
- 1. 下标索引
- 所谓“下标”，就是编号，就好比超市中的存储柜的编号，通过这个编号就能找到相应的存储空间
- 生活中的 "下标"
- 超市储物柜



- 字符串中"下标"的使用
- 列表与元组支持下标索引好理解，字符串实际上就是字符的数组，所以也支持下标索引。
- 如果有字符串: `name = 'abcdef'`，在内存中的实际存储如下：



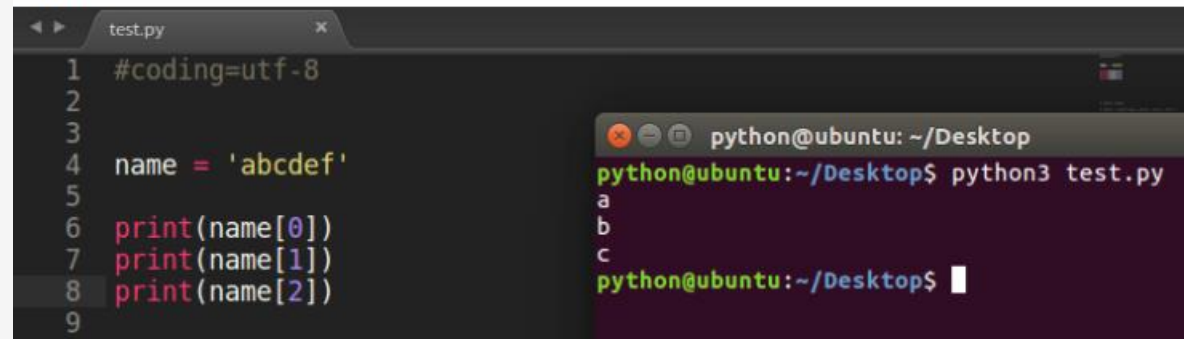
- 如果想取出部分字符，那么可以通过下标的方法，（注意python中下标从0开始）

- 如果想取出部分字符，那么可以通过下标的方法，（注意python中下标从0开始）

```
name = 'abcdef'

print(name[0])
print(name[1])
print(name[2])
```

运行结果:



The screenshot shows a code editor window titled 'test.py' with the following code:

```
1 #coding=utf-8
2
3
4 name = 'abcdef'
5
6 print(name[0])
7 print(name[1])
8 print(name[2])
9
```

Next to the code editor is a terminal window titled 'python@ubuntu: ~/Desktop'. It shows the command 'python3 test.py' being executed, with the output:

```
python@ubuntu:~/Desktop$ python3 test.py
a
b
c
python@ubuntu:~/Desktop$
```

- 2. 切片
- 切片是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。
- 切片的语法：[起始:结束:步长]
- 注意：选取的区间属于左闭右开型，即从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身）。
- 我们以字符串为例讲解。
- 如果取出一部分，则可以在中括号[]中，使用：

```
name = 'abcdef'

print(name[0:3]) # 取 下标0~2 的字符
```

- `name[:2]` 取从头开始带下标为1的字符

```
name = 'abcdef'

print(name[2:]) # 取 下标为2开始到最后的字符
```

- 其他使用如下:

```
>>> a = "abcdef"
>>> a[:3]
'abc'
>>> a[::2]
'ace'
>>> a[5:1:-2]
''
>>> a[1:5:2]
'bd'
>>> a[::-2]
'fdb'
>>> a[5:1:-2]
'fd'
```



## 3.3 字符串常用操作



- 如有字符串 `mystr = 'hello world itcast and itcastcpp'`，以下是常见的操作
- **<1>find**
- 检测 `str` 是否包含在 `mystr` 中，如果是返回开始的索引值，否则返回 -1

```
mystr.find(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast")
12
>>>
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast", 0, 10)
-1
>>>
```

- **<2>index**
- 跟find()方法一样，只不过如果str不在 mystr中会报一个异常。

```
mystr.index(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast",0,10)
-1
>>> mystr.index("itcast",0,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>
```

- **<3>count**
- 返回 str在start和end之间 在 mystr里面出现的次数

```
mystr.count(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.count("itcast")
2
>>>
```



- **<4>replace**
- 把 mystr 中的 str1 替换成 str2,如果 count 指定, 则替换不超过 count 次.

```
mystr.replace(str1, str2, mystr.count(str1))
```

```
>>> name="hello world ha ha"
>>> name.replace("ha", "Ha")
'hello world Ha Ha'
>>> name.replace("ha", "Ha", 1)
'hello world Ha ha'
>>> 
```

- **<5>split**
- 以 str 为分隔符切片 mystr, 如果 maxsplit有指定值, 则仅分隔 maxsplit 个子字符串

```
mystr.split(str=" ", 2)
```

```
>>> name="hello world ha ha"
>>> name.split(" ")
['hello', 'world', 'ha', 'ha']
>>> name.split(" ", 2)
['hello', 'world', 'ha ha']
>>> 
```

- **<6>startswith**
- 检查字符串是否是以 obj 开头, 是则返回 True, 否则返回 False

```
mystr.startswith(obj)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.startswith("hello")
True
>>> mystr.startswith("Hello")
False
>>>
```

- **<7>endswith**
- 检查字符串是否以obj结束, 如果是返回True, 否则返回 False.
- 个子字符串

```
mystr.endswith(obj)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.endswith('cpp')
True
>>> mystr.endswith('app')
False
>>>
```

# 字符串

- **<8>isalpha**
- 如果 mystr 所有字符都是字母 则返回 True,否则返回 False

```
mystr.isalpha()
```

```
>>> mystr = 'abc'
>>> mystr.isalpha()
True
>>> mystr = '123'
>>> mystr.isalpha()
False
>>> mystr = 'abc 123'
>>> mystr.isalpha()
False
>>>
```

- **<9>isdigit**
- 如果 mystr 只包含数字则返回 True 否则返回 False.

```
mystr.isdigit()
```

```
>>> mystr = 'abc'
>>> mystr.isdigit()
False
>>> mystr = '123'
>>> mystr.isdigit()
True
>>> mystr = 'abc123'
>>> mystr.isdigit()
False
>>>
```

# 字符串

- **<10>isalnum**
- 如果 mystr 所有字符都是字母或数字则返回 True, 否则返回 False
- **<11>isspace**
- 如果 mystr 只包含空格则返回 True 否则返回 False.

```
mystr.isalnum()
```

```
>>> mystr = '123'
>>> mystr.isalnum()
True
>>> mystr = 'abc'
>>> mystr.isalnum()
True
>>> mystr = 'abc123'
>>> mystr.isalnum()
True
>>> mystr = 'abc 123'
>>> mystr.isalnum()
False
>>>
```

```
mystr.isspace()
```

```
>>> mystr = 'abc123'
>>> mystr.isspace()
False
>>> mystr = ''
>>> mystr.isspace()
False
>>> mystr = ' '
>>> mystr.isspace()
True
>>> mystr = '   '
>>> mystr.isspace()
True
```

- **<12>lower**
- 转换 mystr 中所有大写字符为小写

```
mystr.lower()
```

```
>>> mystr = 'HELLO world itcast and itcastcpp'
>>> mystr.lower()
'hello world itcast and itcastcpp'
>>>
```

- **<13>upper**
- 转换 mystr 中的小写字母为大写

```
mystr.upper()
```

```
>>> mystr = 'HELLO world itcast and itcastcpp'
>>> mystr.upper()
'HELLO WORLD ITCAST AND ITCASTCPP'
>>>
```

- `<14>join`
- `mystr` 中每个字符后面插入`str`,构造出一个新的字符串

```
mystr.join(str)
```

```
>>> str = " "  
>>> li = ["my", "name", "is", "dongGe"]  
>>> str.join(li)  
'my name is dongGe'  
>>> str = "_"   
>>> str.join(li)  
'my_name_is_dongGe'  
>>>
```

```
mystr.splitlines()
```

```
>>> mystr="hello\nworld"
>>> print mystr
hello
world
>>> mystr.splitlines()
['hello', 'world']
>>>
```

- **<15>splitlines**
- 按照行分隔，返回一个包含各行作为元素的列表

- **<16>partition**
- 把mystr以str分割成三部分,str前, str和str后

```
mystr.partition(str)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.partition("itcast")
('hello world ', 'itcast', ' and itcastcpp')
>>>
```

- **<17>rpartition**
- 类似于 partition()函数,不过是从右边开始.

```
mystr.rpartition(str)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.partition("itcast")
('hello world ', 'itcast', ' and itcastcpp')
>>> mystr.rpartition("itcast")
('hello world itcast and ', 'itcast', 'cpp')
>>>
```



- **<18>rfind**
- 类似于 find()函数，不过是从右边开始查找.

```
mystr.rfind(str, start=0,end=len(mystr) )
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.rfind("itcast")
23
>>>
```

- **<19>rindex**
- 类似于 index(), 不过是从右边开始.

```
mystr.rindex( str, start=0,end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.rindex("IT")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>
```



## 3.4 列表介绍



- 列表介绍

想一想：

前面学习的字符串可以用来存储一串信息，那么想一想，怎样存储咱们班所有同学的名字呢？

定义100个变量，每个变量存放一个学生的姓名可行吗？有更好的办法吗？

答：

列表

- <1>列表的格式
- 变量A的类型为列表

```
namesList = ['xiaoWang', 'xiaoZhang', 'xiaoHua']
```

- 比C语言的数组强大的地方在于列表中的元素可以是不同类型的

```
testList = [1, 'a']
```

- <2>打印列表

demo:

```
namesList = ['xiaoWang','xiaoZhang','xiaoHua']  
print(namesList[0])  
print(namesList[1])  
print(namesList[2])
```

结果：

```
xiaoWang  
xiaoZhang  
xiaoHua
```

- <3>列表的下标

```
[50]: name = "laowang"
```

```
[51]: name[0]
```

```
[51]: 'l'
```

```
[52]: names
```

```
[52]: ['八戒', '沙僧', '老李', '老刘', '老赵', '悟空', '八戒', '老王', '沙僧', '老李']
```

```
[53]: names[0]
```

```
[53]: '八戒'
```

```
[54]: names[1]
```

```
[54]: '沙僧'
```

```
[55]: names[2]
```

```
[55]: '老李'
```

```
[56]: names[-1]
```

```
[56]: '老李'
```

```
[57]: names[2:5]
```

```
[57]: ['老李', '老刘', '老赵']
```

```
[58]:
```



## 3.5 列表的循环遍历



- 1. 使用for循环
- 为了更有效率的输出列表的每个数据，可以使用循环来完成
- demo:

```
namesList = ['xiaoWang','xiaoZhang','xiaoHua']  
for name in namesList:  
    print(name)
```

- 结果

```
xiaoWang  
xiaoZhang  
xiaoHua
```



- 2. 使用while循环
- 为了更有效率的输出列表的每个数据，可以使用循环来完成
- demo:

```
namesList = ['xiaoWang','xiaoZhang','xiaoHua']  
  
length = len(namesList)  
  
i = 0  
  
while i<length:  
    print(namesList[i])  
    i+=1
```

- 结果

```
xiaoWang  
xiaoZhang  
xiaoHua
```



## 3.6 列表的常用操作



- 列表的相关操作
- 列表中存放的数据是可以进行修改的，比如"增"、"删"、"改"
- <1>添加元素("增"append, extend, insert)
- **append**
- 通过append可以向列表添加元素
- demo:

```
#定义变量A，默认有3个元素
A = ['xiaoWang', 'xiaoZhang', 'xiaoHua']

print("-----添加之前，列表A的数据-----")
for tempName in A:
    print(tempName)

#提示、并添加元素
temp = input('请输入要添加的学生姓名:')
A.append(temp)

print("-----添加之后，列表A的数据-----")
for tempName in A:
    print(tempName)
```

结果:

```
-----添加之前，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
请输入要添加的学生姓名:dongGehahahaha
-----添加之后，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
dongGehahahaha
```

- **extend**
- 通过**extend**可以将另一个集合中的元素逐一添加到列表中

```
>>> a = [1, 2]
>>> b = [3, 4]
>>> a.append(b)
>>> a
[1, 2, [3, 4]]
>>> a.extend(b)
>>> a
[1, 2, [3, 4], 3, 4]
```

- **insert**
- **insert(index, object)** 在指定位置index前插入元素object

```
>>> a = [0, 1, 2]
>>> a.insert(1, 3)
>>> a
[0, 3, 1, 2]
```

- <2>修改元素("改")
- 修改元素的时候，要通过下标来确定要修改的是哪个元素，然后才能进行修改
- demo:

```
#定义变量A，默认有3个元素
A = ['xiaoWang','xiaoZhang','xiaoHua']

print("-----修改之前，列表A的数据-----")
for tempName in A:
    print(tempName)

#修改元素
A[1] = 'xiaoLu'

print("-----修改之后，列表A的数据-----")
for tempName in A:
    print(tempName)
```

结果:

```
-----修改之前，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
-----修改之后，列表A的数据-----
xiaoWang
xiaoLu
xiaoHua
```

- <3>查找元素("查"in, not in, index, count)
- 所谓的查找，就是看看指定的元素是否存在
- in, not in
- python中查找的常用方法为：
- in（存在）,如果存在那么结果为true，否则为false
- not in（不存在），如果不存在那么结果为true，否则false
- demo

```
#待查找的列表
nameList = ['xiaoWang','xiaoZhang','xiaoHua']

#获取用户要查找的名字
findName = input('请输入要查找的姓名:')

#查找是否存在
if findName in nameList:
    print('在字典中找到了相同的名字')
else:
    print('没有找到')
```

结果1:(找到)

请输入要查找的姓名:xiaoWang  
在字典中找到了相同的名字

结果2:(没有找到)

请输入要查找的姓名:xiaowanghaha  
没有找到

- **index, count**
- index和count与字符串中的用法相同

```
>>> a = ['a', 'b', 'c', 'a', 'b']
>>> a.index('a', 1, 3) # 注意是左闭右开区间
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'a' is not in list
>>> a.index('a', 1, 4)
3
>>> a.count('b')
2
>>> a.count('d')
0
```

- **<4>删除元素("删"del, pop, remove)**
- 类比现实生活中，如果某位同学调班了，那么就应该把这个条走后的学生的姓名删除掉；在开发中经常会用到删除这种功能。
- 列表元素的常用删除方法有：
- del: 根据下标进行删除
- pop: 删除最后一个元素
- remove: 根据元素的值进行删除
- demo:(del)

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

del movieName[2]

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```



- **demo:(del)**

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

del movieName[2]

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

```
----- 删除之前 -----
加勒比海盗
骇客帝国
第一滴血
指环王
霍比特人
速度与激情
----- 删除之后 -----
加勒比海盗
骇客帝国
指环王
霍比特人
速度与激情
```

- **demo:(pop)** pop也可以根据下标删除元素

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

movieName.pop()

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

----- 删除之前 -----

加勒比海盗

骇客帝国

第一滴血

指环王

霍比特人

速度与激情

----- 删除之后 -----

加勒比海盗

骇客帝国

第一滴血

指环王

霍比特人

- **demo:(remove)**

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

movieName.remove('指环王')

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

```
----- 删除之前 -----
加勒比海盗
骇客帝国
第一滴血
指环王
霍比特人
速度与激情
----- 删除之后 -----
加勒比海盗
骇客帝国
第一滴血
霍比特人
速度与激情
```

- <5>排序(sort, reverse)
- sort方法是将list按特定顺序重新排列，默认为由小到大，参数reverse=True可改为倒序，由大到小。
- reverse方法是将list逆置。

```
>>> a = [1, 4, 2, 3]
>>> a
[1, 4, 2, 3]
>>> a.reverse()
>>> a
[3, 2, 4, 1]
>>> a.sort()
>>> a
[1, 2, 3, 4]
>>> a.sort(reverse=True)
>>> a
[4, 3, 2, 1]
```



## 3.7 列表的嵌套



- 1. 列表嵌套
- 类似while循环的嵌套，列表也是支持嵌套的
- 一个列表中的元素又是一个列表，那么这就是列表的嵌套

```
schoolNames = [['北京大学', '清华大学'],  
                ['南开大学', '天津大学', '天津师范大学'],  
                ['山东大学', '中国海洋大学']]
```

- 2. 应用
- 一个学校，有3个办公室，现在有8位老师等待工位的分配，请编写程序，完成随机的分配

```
#encoding=utf-8

import random

# 定义一个列表用来保存3个办公室
offices = [[],[],[ ]]

# 定义一个列表用来存储8位老师的名字
names = ['A','B','C','D','E','F','G','H']

i = 0
for name in names:
    index = random.randint(0,2)
    offices[index].append(name)

i = 1
for tempNames in offices:
    print('办公室%d的人数为:%d'%(i,len(tempNames)))
    i+=1
    for name in tempNames:
        print("%s"%name,end='')
    print("\n")
    print("-"*20)
```

运行结果如下:

办公室 1 的人数为 :4  
ABCE

-----

办公室 2 的人数为 :3  
DGH

-----

办公室 3 的人数为 :1  
F

-----

## 3.8 元组



- 元组
- Python的元组与列表类似，不同之处在于元组的元素不能修改。元组使用小括号，列表使用方括号。

```
>>> aTuple = ('et',77,99.9)
>>> aTuple
('et',77,99.9)
```

- <1>访问元组

```
>>> tuple=('hello',100,3.14)
>>> tuple[0]
'hello'
>>> tuple[1]
100
>>> tuple[2]
3.14
>>> █
```

- <2>修改元组

```
[>>> tuple[2]=188
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> █
```

说明：**python**中不允许修改元组的数据，包括不能删除其中的元素。

- <3>元组的内置函数count, index
- index和count与字符串和列表中的用法相同

```
>>> a = ('a', 'b', 'c', 'a', 'b')
>>> a.index('a', 1, 3) # 注意是左闭右开区间
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: tuple.index(x): x not in tuple
>>> a.index('a', 1, 4)
3
>>> a.count('b')
2
>>> a.count('d')
0
```



## 3.9 字典介绍



- 列表介绍
- 学生信息列表，每个学生信息包括学号、姓名、年龄等，如何从中找到某个学生的信息？

```
>>> studens = [[1001, "王宝强", 24], [1002, "马蓉", 23], [1005, "宋喆", 24], ...]
```

- <1>生活中的字典



- <2>软件开发中的字典
- 变量info为字典类型:

```
info = {'name': '班长', 'id': 100, 'sex': 'f', 'address': '地球亚洲中国北京'}
```

- 说明:
- 字典和列表一样,也能够存储多个数据
- 列表中找到某个元素时,是根据下标进行的
- 字典中找到某个元素时,是根据'名字'(就是冒号:前面的那个值,例如上面代码中的'name'、'id'、'sex')
- 字典的每个元素由2部分组成,键:值。例如 'name': '班长', 'name' 为键, '班长' 为值

- <3>根据键访问值

```
info = {'name': '班长', 'id': 100, 'sex': 'f', 'address': '地球亚洲中国北京'}

print(info['name'])
print(info['address'])
```

- 结果:

```
班长
地球亚洲中国北京
```

- 若访问不存在的键，则会报错:

```
>>> info['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```



- 在我们不确定字典中是否存在某个键而又想获取其值时，可以使用`get`方法，还可以设置默认值：

```
>>> age = info.get('age')
>>> age # 'age'键不存在，所以age为None
>>> type(age)
<type 'NoneType'>
>>> age = info.get('age', 18) # 若info中不存在'age'这个键，就返回默认值18
>>> age
18
```



## 3.9 字典常用操作1



- <1>修改元素
- 字典的每个元素中的数据是可以修改的，只要通过key找到，即可修改

- demo:

```
info = {'name': '班长', 'id': 100, 'sex': 'f', 'address': '地球亚洲中国北京'}
```

```
newId = input('请输入新的学号')
```

```
info['id'] = int(newId)
```

```
print('修改之后的id为%d: %s' % info['id'])
```

- 结果:

```
请输入新的学号 88  
修改之后的 id为： 88
```

- <2>添加元素
- demo:访问不存在的元素

```
info = {'name':'班长', 'sex':'f', 'address':'地球亚洲中国北京'}  
  
print('id为:%d'%info['id'])
```

结果:

```
MacBook-Pro 01-python基础班-资料$ python test.py  
File "test.py", line 10  
    print 'id为:' info['id']  
                  ^  
SyntaxError: invalid syntax
```

- 如果在使用 变量名['键']=数据 时，这个“键”在字典中，不存在，那么就会新增这个元素

- demo:添加新的元素

```
info = {'name':'班长', 'sex':'f', 'address':'地球亚洲中国北京'}

# print('id为:%d'%info['id'])#程序会终端运行，因为访问了不存在的键

newId = input('请输入新的学号')

info['id'] = newId

print('添加之后的id为:%d'%info['id'])
```

- 结果:

```
请输入新的学号188
添加之后的id为: 188
```

- <3>删除元素
- 对字典进行删除操作，有以下几种：
- del
- pop
- demo:del删除指定的元素

```
info = {'name': '班长', 'sex': 'f', 'address': '地球亚洲中国北京'}

print('删除前,%s'%info['name'])

del info['name']

print('删除后,%s'%info['name'])
```

结果

```
MacBook-Pro 01-python基础班-资料$ python test.py
删除前, 班长
删除后,
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print '删除后,',info['name']
KeyError: 'name'
```

删除后不  
能访问

- pop: 根据key值删除数据
- dic.pop(key)



## 3.9 字典常用操作2





- **<1>len()**

测量字典中，键值对的个数

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> len(dict)  
2  
>>> █
```

- **<2>keys**

返回一个包含字典所有KEY的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.keys()  
['name', 'sex']  
>>> █
```

- **<3>values**

返回一个包含字典所有value的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.values()  
['zhangsan', 'm']  
>>> █
```

- **<4>items**

返回一个包含所有 ( 键，值 ) 元祖的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.items()  
[('name', 'zhangsan'), ('sex', 'm')]  
>>> █
```



## 3. 10 字典的遍历



- 遍历
- 通过for ... in ...:的语法结构，我们可以遍历字符串、列表、元组、字典等数据结构。
- 注意python语法的缩进

- 字符串遍历

```
>>> a_str = "hello itcast"
>>> for char in a_str:
...     print(char,end=' ')
...
h e l l o   i t c a s t
```

- 列表遍历

```
>>> a_list = [1, 2, 3, 4, 5]
>>> for num in a_list:
...     print(num,end=' ')
...
1 2 3 4 5
```

- 元组遍历

```
>>> a_tuple = (1, 2, 3, 4, 5)
>>> for num in a_tuple:
...     print(num,end=" ")
1 2 3 4 5
```

- 字典遍历
- <1> 遍历字典的key（键）
- <2> 遍历字典的value（值）
- <3> 遍历字典的项（元素）
- <4> 遍历字典的key-value（键值对）

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for key in dict.keys():
...     print key
...
name
sex
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for value in dict.values():
...     print value
...
zhangsan
m
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for item in dict.items():
...     print item
...
('name', 'zhangsan')
('sex', 'm')
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for key,value in dict.items():
...     print("key=%s,value=%s"%(key,value))
...
key=name,value=zhangsan
key=sex,value=m
>>>
```



## 3. 11 附加



- **python**内置函数
- Python包含了以下内置函数

序号	方法	描述
1	cmp(item1, item2)	比较两个值
2	len(item)	计算容器中元素个数
3	max(item)	返回容器中元素最大值
4	min(item)	返回容器中元素最小值
5	del(item)	删除变量

- 公共方法
- 运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	'Hi!' * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	复制	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典
not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典

- 多维列表/元祖访问的示例

```
>>> tuple1 = [(2,3),(4,5)]
>>> tuple1[0]
(2, 3)
>>> tuple1[0][0]
2
>>> tuple1[0][2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
>>> tuple1[0][1]
3
>>> tuple1[2][2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> tuple2 = tuple1+[(3)]
>>> tuple2
[(2, 3), (4, 5), 3]
>>> tuple2[2]
3
>>> tuple2[2][0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
```