PYTHON 介绍

PYTHON 安装

安装 PYTHON2.X 还是 3.X?

2 与 3 的区别？

简单总结： *Python 2.x is legacy, Python 3.x is the present and future of the language*
Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release. The 2.x branch will see no new major releases after that. 3.x is under active development and has already seen over five years of stable releases, including version 3.3 in 2012, 3.4 in 2014, and 3.5 in 2015. This means that all recent standard library improvements, for example, are only available by default in Python 3.x.

Guido van Rossum (the original creator of the Python language) decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. The most drastic improvement is the better Unicode support (with all text strings being Unicode by default) as well as saner bytes/Unicode separation.

Besides, several aspects of the core language (such as print and exec being statements, integers using floor division) have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language, and old cruft has been removed (for example, all classes are now new-style, "range()" returns a memory efficient iterable, not a list as in 2.x).

详细区别：

PRINT IS A FUNCTION

The `print` statement has been replaced with a `print()` function, with keyword arguments to replace most of the special syntax of the old `print` statement (**PEP 3105**). Examples:

```
Old: print "The answer is", 2*2
New: print("The answer is", 2*2)

Old: print x,           # Trailing comma suppresses newline
New: print(x, end=" ")  # Appends a space instead of a newline

Old: print              # Prints a newline
New: print()            # You must call the function!

Old: print >>sys.stderr, "fatal error"
New: print("fatal error", file=sys.stderr)

Old: print (x, y)       # prints repr((x, y))
New: print((x, y))      # Not the same as print(x, y)!
```

You can also customize the separator between items, e.g.:

```
print("There are <", 2**32, "> possibilities!", sep="")
```

which produces:

```
There are <4294967296> possibilities!
```

ALL IS UNICODE NOW

从此不用再为编码错误而烦躁啦

我擦，还可以这么玩？ (A, *REST, B) = RANGE(5)

PEP 3132: Extended Iterable Unpacking. You can now write things like `a, b, *rest = some_sequence`. And even `*rest, a = stuff`. The `rest` object is always a (possibly empty) list; the right-hand side may be any iterable. Example:

```
(a, *rest, b) = range(5)
```

This sets *a* to `0`, *b* to `4`, and *rest* to `[1, 2, 3]`.

LIBRARY CHANGES

Old Name	New Name
_winreg	winreg
ConfigParser	configparser
copy_reg	copyreg
Queue	queue
SocketServer	socketserver
markupbase	_markupbase
repr	reprlib
test.test_support	test.support

还有谁敢不支持 PYTHON3？

One popular module that don't yet support Python 3 is Twisted (for networking and other applications). Most actively maintained libraries have people working on 3.x support. For some libraries, it's more of a priority than others: Twisted, for example, is mostly focused on production servers, where supporting older versions of Python is important, let alone supporting a new version that includes major changes to the language. (Twisted is a prime example of a major package where porting to 3.x is far from trivial.)

总结：哪些 PYTHON2 的语法在 3 里变了？

1. 1 / 2 终于等于0.5了
2. print “hello World” 变成了 print(“hello world!”)
3. raw_input 没了
4. class Foo: 写法不能用了, 只能 class Foo(object)

安装 PYTHON3.X

windows：

```
1、下载安装包
    https://www.python.org/downloads/
2、安装
    默认安装路径：C:\python27
3、配置环境变量
    【右键计算机】--》【属性】--》【高级系统设置】--》【高级】--》【环境变量】--》【在第二个内
    如：原来的值;C:\python27，切记前面有分号
```

linux：

我想就不用说了吧！

HELLO WORLD

打印一条输出

```
# python3   #进入 Python 交互器模式

Python 3.4.3 (default, Oct 14 2015, 20:28:29)

[GCC 4.8.4] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> print("Welcome to Oldboy,Looking forward to become outstanding after 6 months!")

Welcome to Oldboy,Looking forward to become outstanding after 6 months!
```

缩进

Python uses indentation for blocks, instead of curly braces. Both tabs and spaces are supported, but the standard indentation requires standard Python code to use four spaces. For example:

```
x = 1

if x == 1:

    # indented four spaces
    print "x is 1."
```

脚本执行

```
python3 hello.py

或者

$ vim hello.py

#!/usr/bin/env python   #必须声明用什么解释器来执行此脚本

print(“hello world!”)


$ chmod +x hello.py  #加上可执行权限

$ ./hello.py
```

.PYC 是个什么鬼？

Most probably you will have read somewhere that the Python language is an interpreted programming or a script language. The truth is: Python is both an interpreted and a compiled language. But calling Python a compiled language would be misleading. (At the end of this chapter, you will find the definitions for Compilers and Interpreters, if you are not familiar with the concepts!) People would assume that the compiler translates the Python code into machine language. Python code is translated into intermediate code, which has to be executed by a virtual machine, known as the PVM, the Python virtual machine. This is a similar approach to the one taken by Java. There is even a way of translating Python programs into Java byte code for the Java Virtual Machine (JVM). This can be achieved with Jython.

The question is, do I have to compile my Python scripts to make them faster or how can I compile them? The answer is easy: Normally, you don't need to do anything and you shouldn't bother, because "Python" is doing the thinking for you, i.e. it takes the necessary steps automatically.

For whatever reason you want to compile a python program manually? No problem. It can be done with the module py_compile, either using the interpreter shell

```
>>> import py_compile

>>> py_compile.compile('my_first_simple_script.py')

>>>
```

or using the following command at the shell prompt

```
python -m py_compile my_first_simple_script.py
```

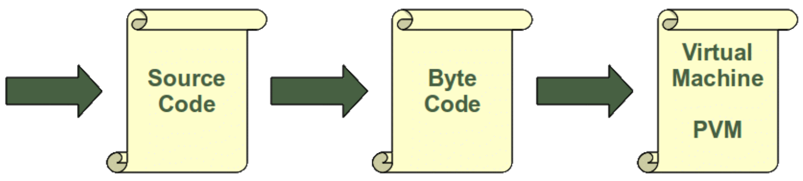
Either way, you may notice two things: First, there will be a new subdirectory "__pycache__", if it hasn't already existed. You will find a file "my_first_simple_script.cpython-34.pyc" in this subdirectory. This is the compiled version of our file in byte code.

You can also automatically compile all Python files using the compileall module. You can do it from the shell prompt by running compileall.py and providing the path of the directory containing the Python files to compile:

```
monty@python:~/python$ python -m compileall .

Listing . ...
```

But as we have said, you don't have to and shouldn't bother about compiling Python code. The compilation is hidden from the user for a good reason. Some newbies to Python wonder sometimes where these ominous files with the .pyc suffix might come from. If Python has write-access for the directory where the Python program resides, it will store the compiled byte code in a file that ends with a .pyc suffix. If Python has no write access, the program will work anyway. The byte code will be produced but discarded when the program exits.
Whenever a Python program is called, Python will check, if there exists a compiled version with the .pyc suffix. This file has to be newer than the file with the .py suffix. If such a file exists, Python will load the byte code, which will speed up the start up time of the script. If there exists no byte code version, Python will create the byte code before it starts the execution of the program. Execution of a Python program means execution of the byte code on the Python Virtual Machine (PVM).



Every time a Python script is executed, byte code is created. If a Python script is imported as a module, the byte code will be stored in the corresponding .pyc file.
So the following will not create a byte code file:

```
monty@python:~/python$ python my_first_simple_script.py
My first simple Python script!
monty@python:~/python$
```

The import in the following session will create a byte code file with the name "easy_to_write.pyc":

```
monty@python:~/tmp$ ls
my_first_simple_script.py

monty@python:~/tmp$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import my_first_simple_script
My first simple Python script!
>>> exit()
monty@python:~/tmp$ ls
my_first_simple_script.py  my_first_simple_script.pyc
monty@python:~/tmp$
```

Compiler

Definition: A compiler is a computer program that transforms (translates) source code of a programming language (the source language) into another computer language (the target language). In most cases compilers are used to transform source code into executable program, i.e. they translate code from high-level programming languages into low (or lower) level languages, mostly assembly or machine code.

Interpreter

Definition: An interpreter is a computer program that executes instructions written in a programming language. It can either

- execute the source code directly or
- translates the source code in a first step into a more efficient representation and executes this code

变量

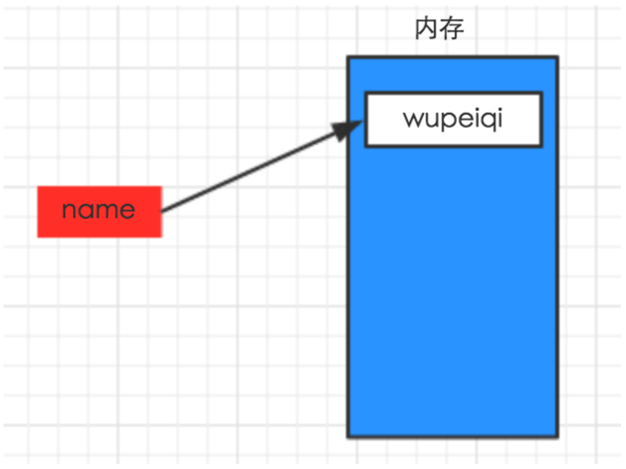
1、声明变量

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

name = "wupeiqi"
```

上述代码声明了一个变量，变量名为： name，变量 name 的值为："wupeiqi"

变量的作用：昵称，其代指内存里某个地址中保存的内容



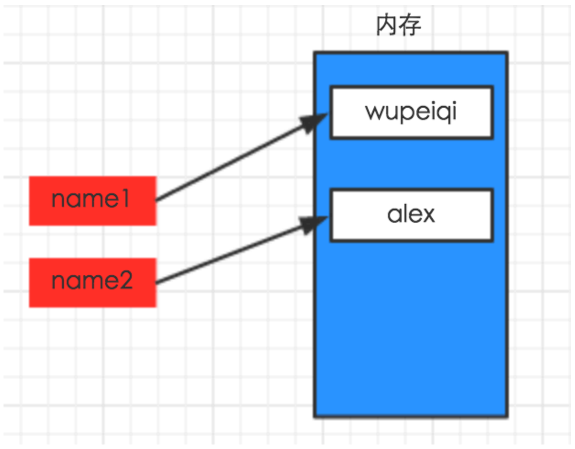
变量定义的规则：

- 变量名只能是 字母、数字或下划线的任意组合
- 变量名的第一个字符不能是数字
- 以下关键字不能声明为变量名
  ['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield']

2、变量的赋值

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

name1 = "wupeiqi"
name2 = "alex"
```

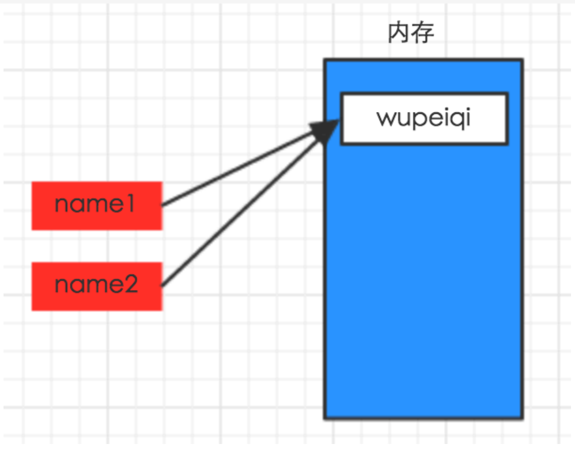


```
#!/usr/bin/env python

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

name1 = "wupeiqi"

name2 = name1
```

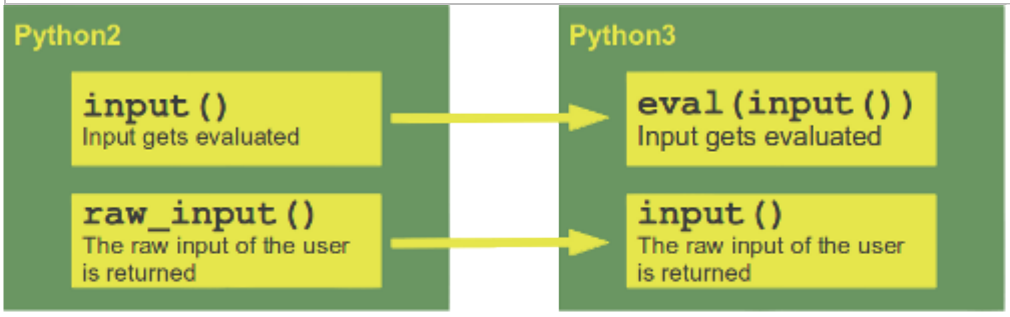


用户交互

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

# 将用户输入的内容赋值给 name 变量
name = input("请输入用户名：")  #no more raw_input

# 打印输入的内容
print name
```



输入密码时，如果想要不可见，需要利用 getpass 模块中的 getpass 方法，即：

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

import getpass

# 将用户输入的内容赋值给 name 变量
pwd = getpass.getpass("请输入密码：")

# 打印输入的内容
print pwd
```

条件语句和缩进

需求一、用户登陆验证

```
# 提示输入用户名和密码

# 验证用户名和密码
#       如果错误，则输出用户名或密码错误
#       如果成功，则输出 欢迎，XXX!
```

```
#!/usr/bin/env python
# -*- coding: encoding -*-

import getpass


name = raw_input('请输入用户名：')
pwd = getpass.getpass('请输入密码：')

if name == "alex" and pwd == "cmd":
    print "欢迎，alex！"
else:
    print "用户名和密码错误"
```

需求二、根据用户输入内容输出其权限

```
# 根据用户输入内容打印其权限

# alex --> 超级管理员
# eric --> 普通管理员
# tony --> 业务主管
# 其他 --> 普通用户
```

```
#!/usr/bin/env python
# -*- coding: encoding -*-

name = raw_input('请输入用户名：')

if name == "alex"：
    print "超级管理员"
```

```
elif name == "eric":
    print "普通管理员"
elif name == "tony":
    print "业务主管"
else:
    print "普通用户"
```

*详细关于 if 条件表达式的介绍：http://www.python-course.eu/python3_conditional_statements.php

初识数据类型

1、数字

2 是一个整数的例子。
长整数 不过是大一些的整数。
3.23 和 52.3E-4 是浮点数的例子。E 标记表示 10 的幂。在这里，52.3E-4 表示 52.3 * 10-4。

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。

2、布尔值

真或假

1 或 0

3、字符串

"hello world"

万恶的字符串拼接：

python 中的字符串在 C 语言中体现为是一个字符数组，每次创建字符串时候需要在内存中开辟一块连续的空，并且一旦需要修改字符串的话，就需要再次开辟空间，万恶的+号每出现一次就会在内从中重新开辟一块空间。

字符串格式化

```
name = "alex"
print "i am %s " % name

#输出: i am alex
```

PS: 字符串是 %s;整数 %d;浮点数%f

字符串常用功能：

- 移除空白
- 分割
- 长度
- 索引
- 切片

4、列表

创建列表：

```
name_list = ['alex', 'seven', 'eric']
或
name_list = list(['alex', 'seven', 'eric'])
```

基本操作：

- 索引
- 切片
- 追加
- 删除
- 长度
- 切片
- 循环
- 包含
-

5、元组

- 创建元组：

```
ages = (11, 22, 33, 44, 55)
或
ages = tuple((11, 22, 33, 44, 55))
```

- 基本操作：
- 索引
- 切片
- 循环
- 长度
- 包含

6、字典（无序）

- 创建字典：

```
person = {"name": "mr.wu", 'age': 18}
或
person = dict({"name": "mr.wu", 'age': 18})
```

- 常用操作：
- 索引
- 新增
- 删除
- 键、值、键值对
- 循环
- 长度
- PS：循环，range，continue 和 break

运算符

算术运算符

以下假设变量 a 为 10，变量 b 为 20：

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
%	取模 - 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
//	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

比较运算符

运算符	描述	实例
==	等于 - 比较对象是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象是否不相等	(a != b) 返回 true.
<>	不等于 - 比较两个对象是否不相等；(Python3 中已废弃)	(a <> b) 返回 true。这个运算符类似 != 。
>	大于 - 返回 x 是否大于 y	(a > b) 返回 False。
<	小于 - 返回 x 是否小于 y。所有比较运算符返回 1 表示真，返回 0 表示假。这分别与特殊的变量 True 和 False 等价。注意，这些变量名的大写。	(a < b) 返回 true。

>=	大于等于 - 返回 x 是否大于等于 y。	(a >= b) 返回 False。
<=	小于等于 - 返回 x 是否小于等于 y。	(a <= b) 返回 true。

赋值运算符

以下假设变量 a 为 10，变量 b 为 20：

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	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

位运算符

按位运算符是把数字看作二进制来进行计算的。Python 中的按位运算法则如下：

运算符	描述	实例
&	按位与运算符	(a & b) 输出结果 12 ，二进制解释： 0000 1100
|	按位或运算符	(a | b) 输出结果 61 ，二进制解释： 0011 1101
^	按位异或运算符	(a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~	按位取反运算符	(~a) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。
<<	左移动运算符	a << 2 输出结果 240 ，二进制解释： 1111 0000
>>	右移动运算符	a >> 2 输出结果 15 ，二进制解释： 0000 1111

以下实例演示了 Python 所有位运算符的操作：

```
#!/usr/bin/python

a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0

c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c = a | b;        # 61 = 0011 1101
print "Line 2 - Value of c is ", c
```

```
c = a ^ b;        # 49 = 0011 0001
print "Line 3 - Value of c is ", c

c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c

c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;       # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

以上实例输出结果：

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

逻辑运算符

运算符	描述	实例
and	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 true。
or	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 true。
not	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。

```
#!/usr/bin/python

a = 10
b = 20
c = 0

if ( a and b ):
   print "Line 1 - a and b are true"
else:
   print "Line 1 - Either a is not true or b is not true"

if ( a or b ):
   print "Line 2 - Either a is true or b is true or both are true"
else:
   print "Line 2 - Neither a is true nor b is true"


a = 0
```

```
if ( a and b ):
   print "Line 3 - a and b are true"
else:
   print "Line 3 - Either a is not true or b is not true"

if ( a or b ):
   print "Line 4 - Either a is true or b is true or both are true"
else:
   print "Line 4 - Neither a is true nor b is true"

if not( a and b ):
   print "Line 5 - Either a is not true or b is  not true or both are not true"
else:
   print "Line 5 - a and b are true"
```

成员运算符

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False。	x 在 y 序列中 , 如果 x 在 y 序列中返回 True。
not in	如果在指定的序列中没有找到值返回 True，否则返回 False。	x 不在 y 序列中 , 如果 x 不在 y 序列中返回 True。

以下实例演示了 Python 所有成员运算符的操作：

```
#!/usr/bin/python

a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"

if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"

a = 2
if ( a in list ):
   print "Line 3 - a is available in the given list"
else:
   print "Line 3 - a is not available in the given list"
```

身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	x is y, 如果 id(x) 等于 id(y) , **is** 返回结果 1
is not	is not 是判断两个标识符是不是引用自不同对象	x is not y, 如果 id(x) 不等于 id(y). **is not** 返回结果 1

字符编码

python 解释器在加载 .py 文件中的代码时，会对内容进行编码（默认 ascill）

ASCII（American Standard Code for Information Interchange，美国标准信息交换代码）是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语和其他西欧语言，其最多只能用 8 位来表示（一个字节），即：2**8 = 256，所以，ASCII 码最多只能表示 256 个符号。

0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	∆	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	“	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	”	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	‘	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	’	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	˚
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	ﬁ	254	˛
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	ﬂ	255	ˇ

显然 ASCII 码无法将世界上的各种文字和符号全部表示，所以，就需要新出一种可以代表所有字符和符号的编码，即：Unicode

Unicode（统一码、万国码、单一码）是一种在计算机上使用的字符编码。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，规定所有的字符和符号最少由 16 位二进制来表示（2 个字

节），即：2 **16 = 65536,

注：此处说的的是最少 2 个字节，可能更多

UTF-8，是对 Unicode 编码的压缩和优化，他不再使用最少使用 2 个字节，而是将所有的字符和符号进行分类：ascii 码中的内容用 1 个字节保存、欧洲的字符用 2 个字节保存，东亚的字符用 3 个字节保存...

所以，python 解释器在加载 .py 文件中的代码时，会对内容进行编码（默认 ascill），如果是如下代码的话：

报错：ascii 码无法表示中文

```
#!/usr/bin/env python

print "你好，世界"
```

改正：应该显示的告诉 python 解释器，用什么编码来执行源代码，即：

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

print "你好，世界"
print "有没有觉得 Alex 长的好帅！ "
```

循环

There are two types of loops in Python, for and while.

THE "FOR" LOOP

For loops iterate over a given sequence. Here is an example:

```
primes = [2, 3, 5, 7]

for prime in primes:

    print prime
```

Execute Code

For loops can iterate over a sequence of numbers using the "range" and "xrange" functions. The difference between range and xrange is that the range function returns a new list with numbers of that specified range, whereas xrange returns an iterator, which is more efficient. (Python 3 uses the range function, which acts like xrange). Note that the xrange function is zero based.

```
# Prints out the numbers 0,1,2,3,4

for x in xrange(5): # or range(5)

    print x



# Prints out 3,4,5

for x in xrange(3, 6): # or range(3, 6)
```

```
    print x


# Prints out 3,5,7

for x in xrange(3, 8, 2): # or range(3, 8, 2)

    print x
```

Execute Code

"WHILE" LOOPS

While loops repeat as long as a certain boolean condition is met. For example:

```
# Prints out 0,1,2,3,4


count = 0

while count < 5:

    print count

    count += 1  # This is the same as count = count + 1
```

Execute Code

"BREAK" AND "CONTINUE" STATEMENTS

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement. A few examples:

```
# Prints out 0,1,2,3,4


count = 0

while True:

    print count

    count += 1

    if count >= 5:

        break


# Prints out only odd numbers - 1,3,5,7,9

for x in xrange(10):

    # Check if x is even

    if x % 2 == 0:

        continue

    print x
```

*详细关于循环的文档: http://www.python-course.eu/python3_loops.php

文件操作初识

打开文件：

```
file_obj = file("文件路径","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：

```
# 一次性加载所有内容到内存
obj.read()

# 一次性加载所有内容到内存，并根据行分割成字符串
obj.readlines()

# 每次仅读取一行数据
for line in obj:
    print line
```

写文件的内容：

```
obj.write('内容')
```

关闭文件句柄：

```
obj.close()
```

本节作业

作业一：编写登陆接口

- 输入用户名密码
- 认证成功后显示欢迎信息
- 输错三次后锁定

作业二：多级菜单

- 三级菜单
- 可依次选择进入各子菜单
- 所需新知识点：列表、字典

作业要求

1. Readme 文件， 告诉别人如何使用你的程序(必须)
2. 代码加注释， 让别人可以轻松读懂你的代码(必须)
3. 目录结构要符合规范,每天单独一个目录， 如 Day1, Day2, Day3…(必须)
4. 流程图， 帮自己理清思路、帮别人更容易了解你的代码设计逻辑(必须)
5. blog， 写好 blog, 让更多的人知道你，关注你(必须)
6. 开个 github 吧（强烈建议）