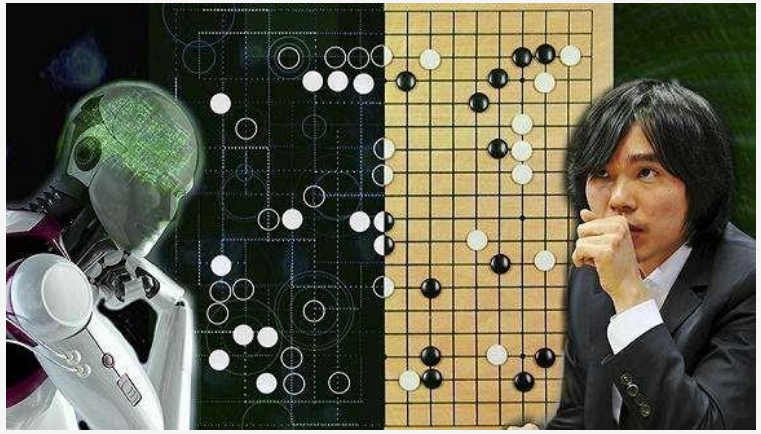
人工智能导论 & Python基础

人工智能与机器学习

- 机器学习(Machine Learning, ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科
- 计算器学习专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，并重新组织已有的知识结构使之不断改善自身 的性能
- 机器学习是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎
- 机器学习是计算机科学家想让计算机像人一样思考，所研发出来的计算机理论
- 诞生于上个世纪60年代，在最近的十几年发展非常迅速

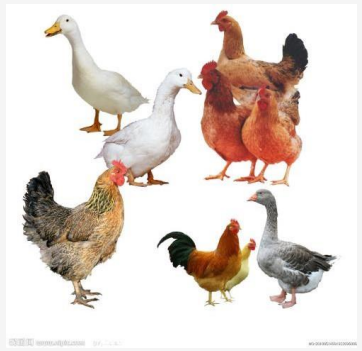
机器学习最常见应用

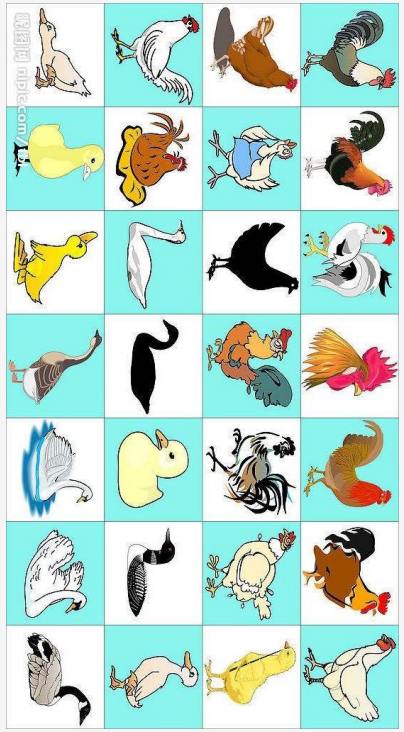


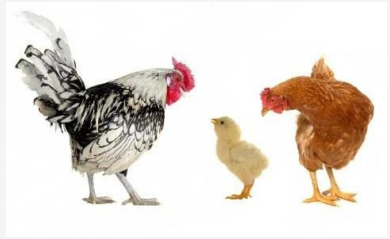
机器学习的方式

监督学习

无监督学习







无监督学习 vs 监督学习

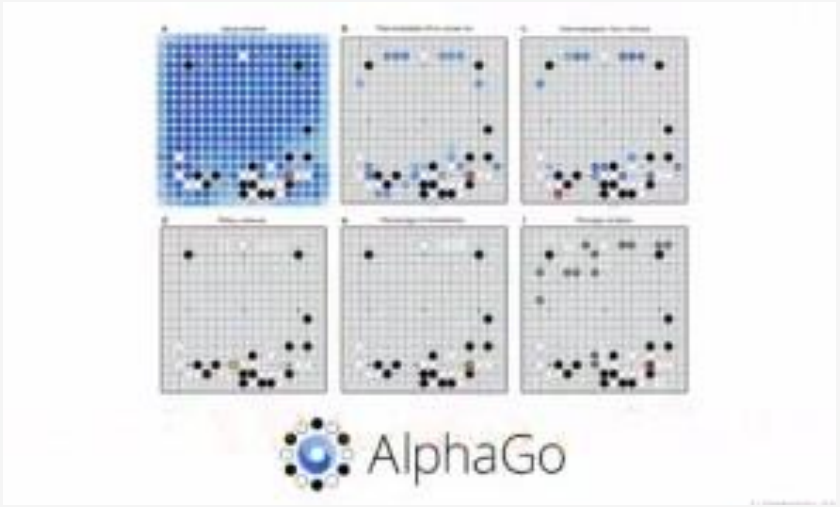
分类标记

編號	性別	年齡	婚姻	家庭人數	購買RV房車
A0001	Male	45	未婚	1	是
A0002	Male	52	已婚	7	是
A0003	Female	38	已婚	5	是
A0004	Male	25	已婚	5	否
A0005	Female	48	已婚	4	是
A0006	Male	32	未婚	3	是
A0007	Female	65	已婚	4	否
A0008	Male	33	已婚	3	是
A0009	Male	45	已婚	4	是
A0010	Female	52	未婚	1	是
A0011	Male	38	未婚	1	否
…	…	…	…	…	…
Z0099	Male	22	未婚	4	是

监督学习能实现，为什么还要研究无监督学习？

- 缺乏足够的先验知识的领域，难以人工标注类别
- 进行人工类别标注的成本太高

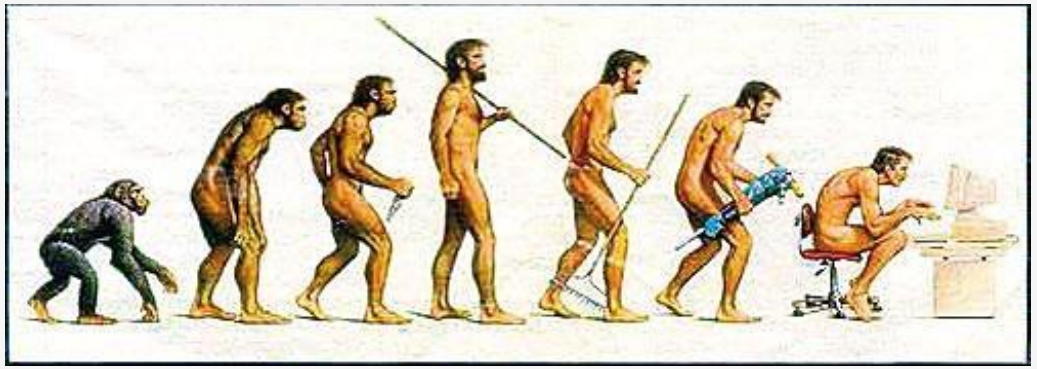
强化学习



命中

未命中

遗传算法



优胜劣汰

适者生存

Python基础课程

0 0 0 2 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

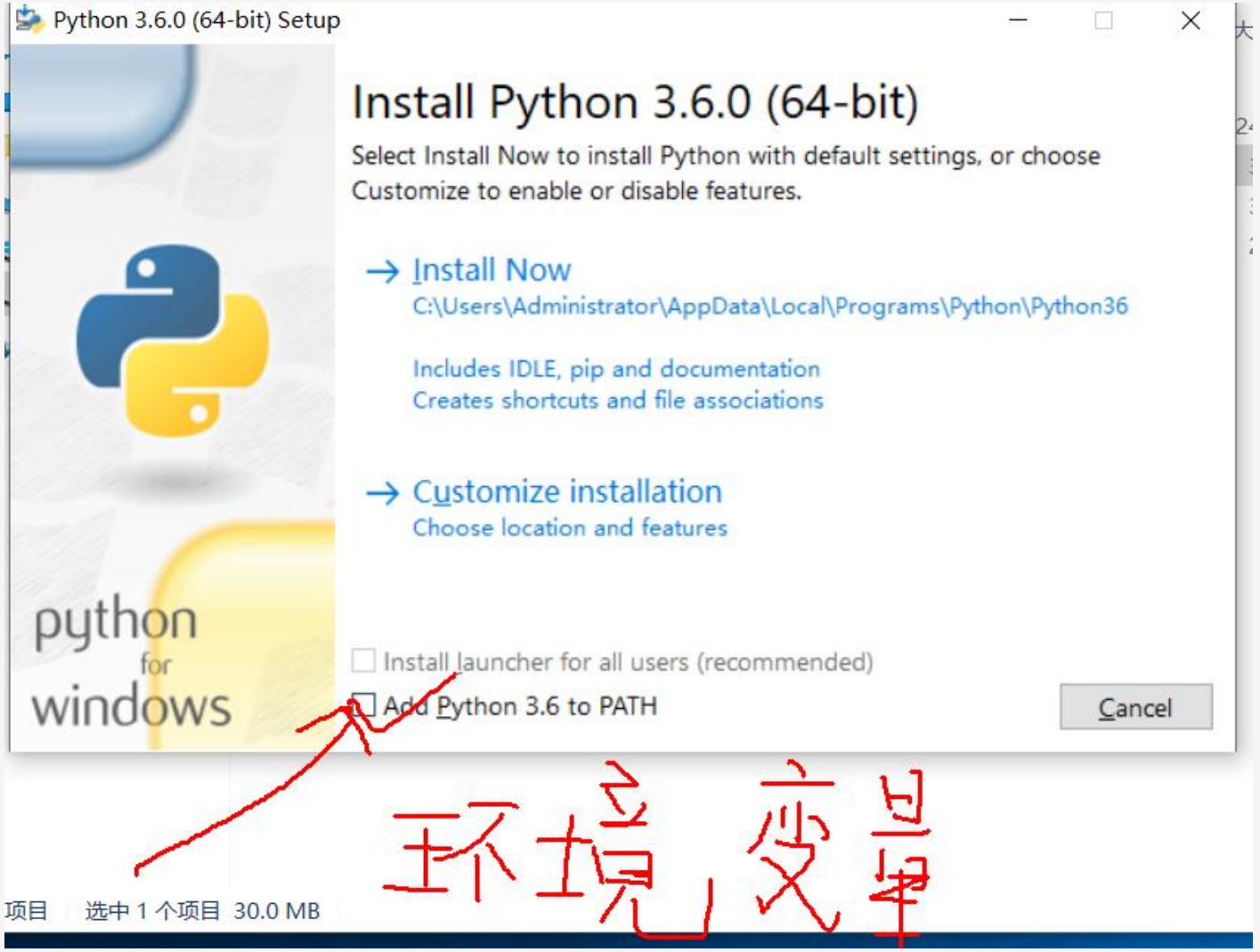
- **缺点**

- 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。

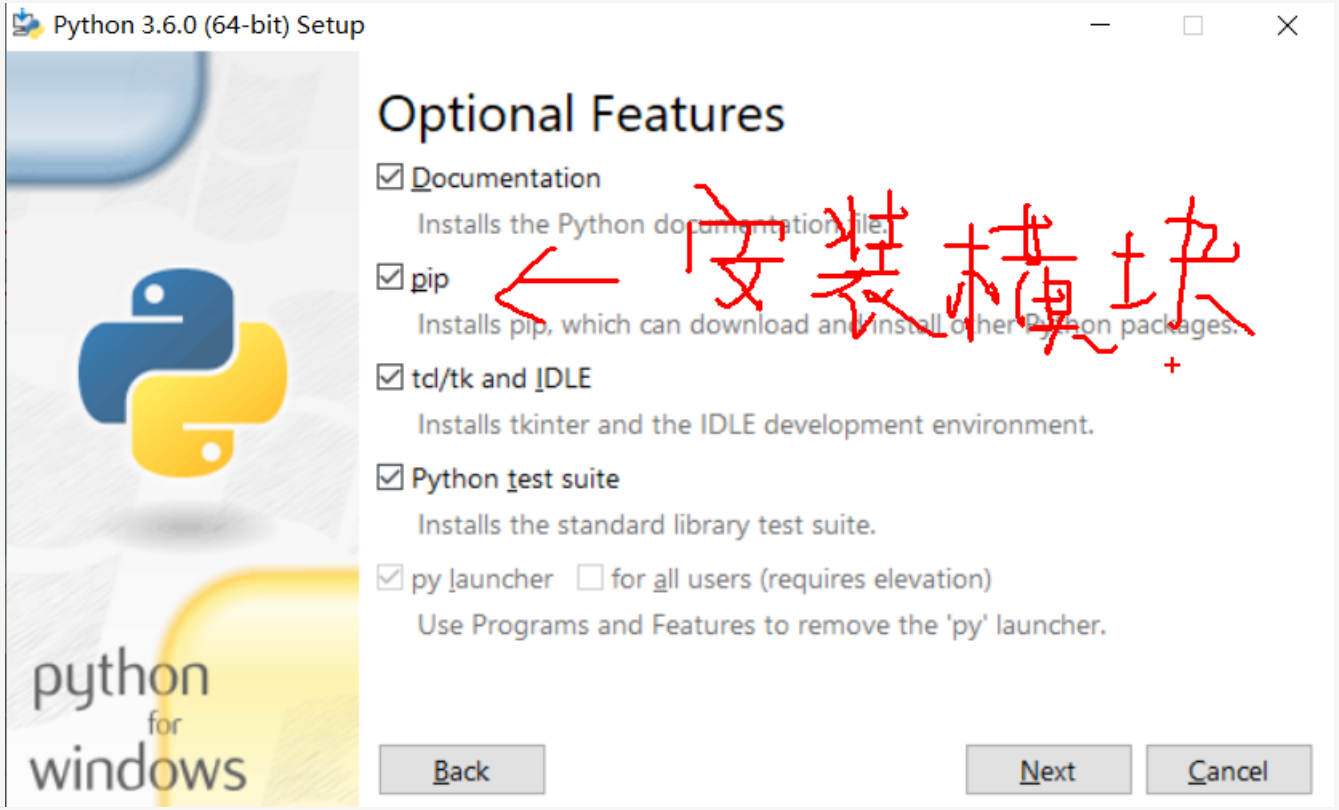
Python的应用场景

- 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

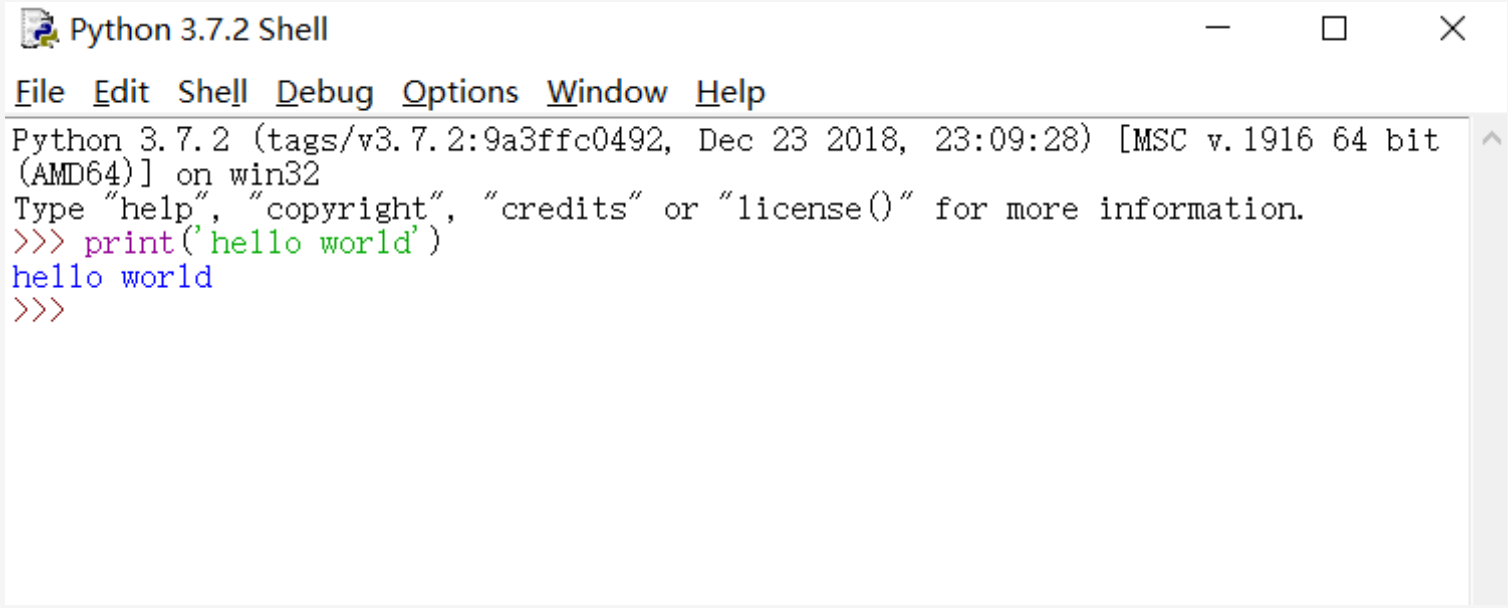


Python运行环境的安装

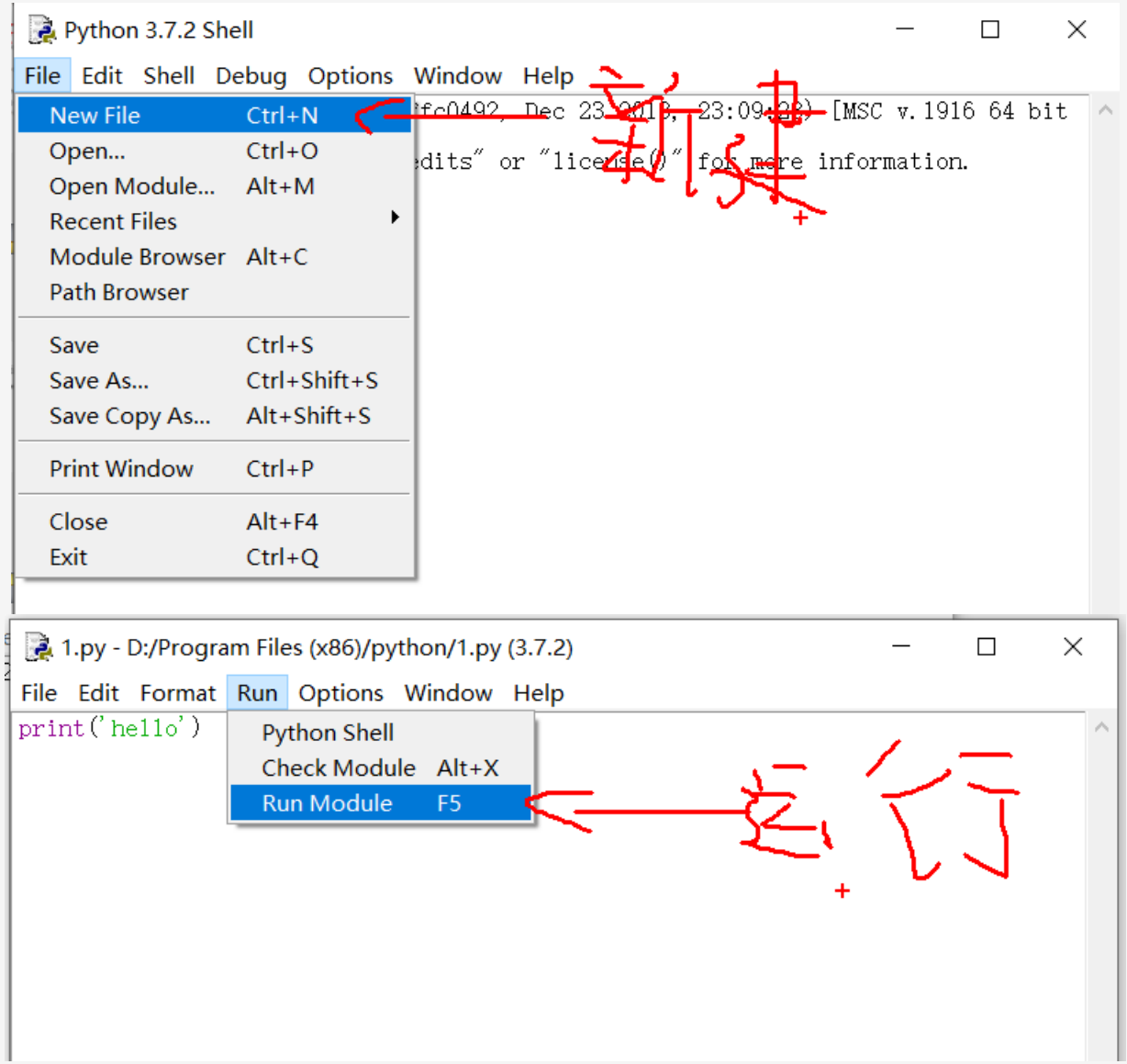


1.2 第一个Python程序

Python的shell

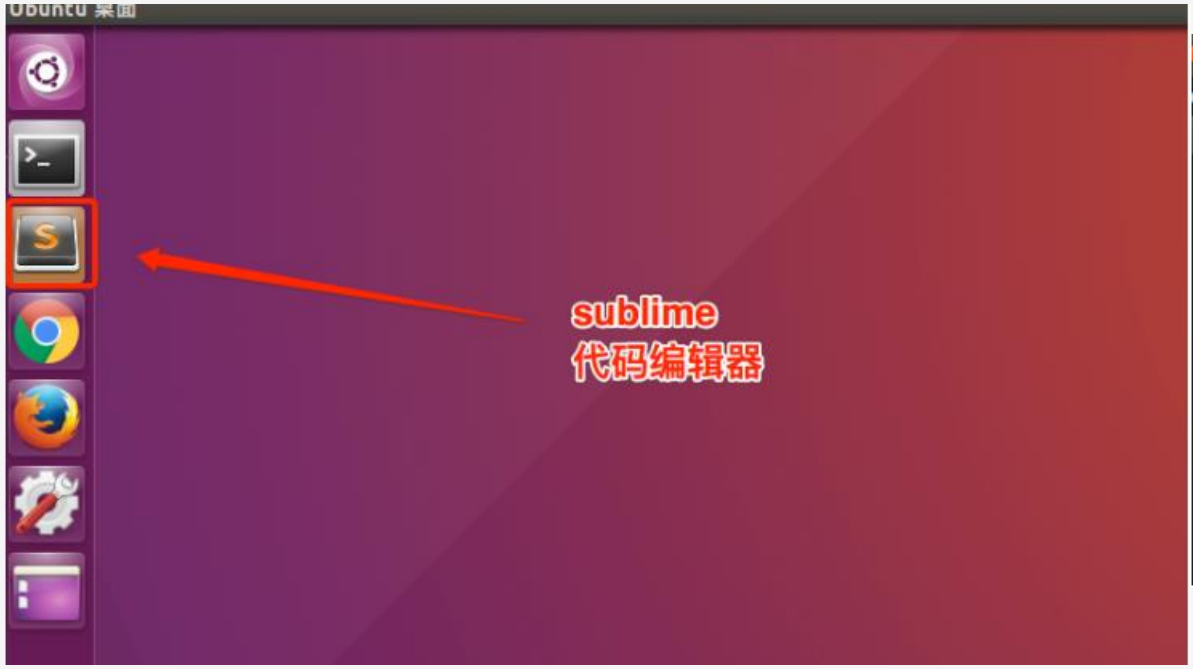


Python的shell



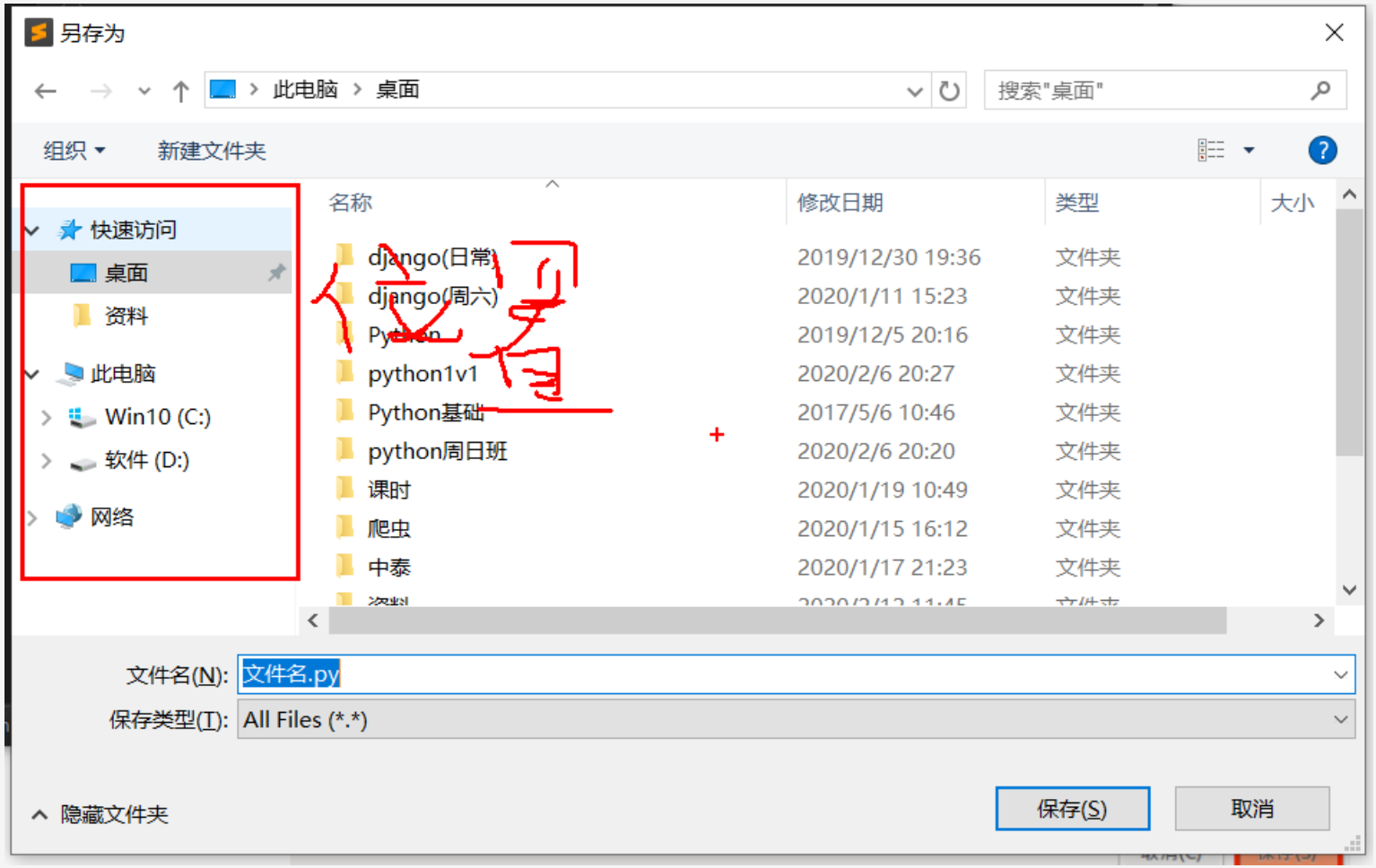
Python的编辑器

- 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

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

注释的引入

- **<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('HGHSCORE: ' + str(self.highscore))
112         for row in self.field:
113             draw_hor_separator()
114             draw_row(row)
115         draw_hor_separator()
```

注释的引入

- **<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

注释的引入

- **<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。

注释的分类

- **<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')
```

注释的分类

- **<2> 多行注释**

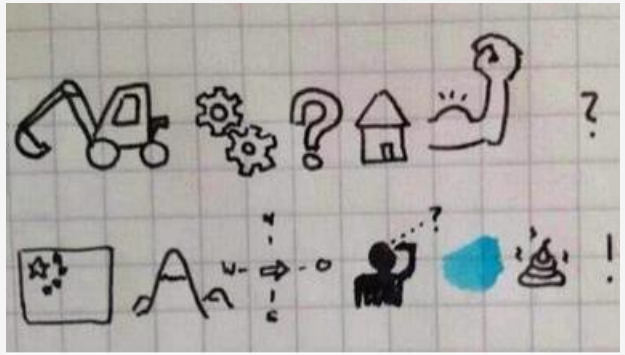
- '''
- 我是多行注释，可以写很多很多行的功能说明 这就是我牛X指出 哈哈哈。。。
  '''
- '''
- 下面的代码完成，
- 打印一首诗 名字叫做：
- 春江花月夜 作者，忘了
- '''

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

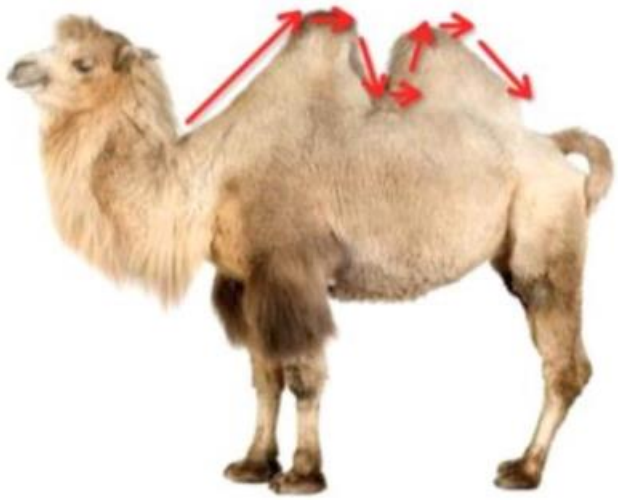
Andy ≠ andy

```
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



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

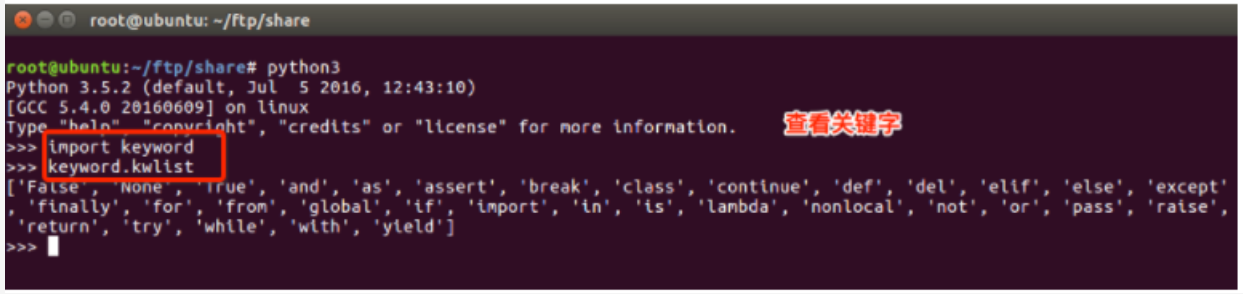
- **<4>关键字**

- 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



1.5 变量及类型

变量以及类型

- **<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？

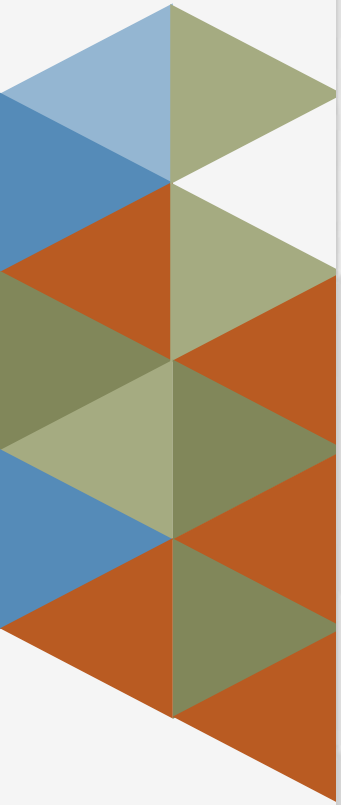
- **<2>变量的类型**
- 生活中的“类型”的例子:

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:

变量以及类型

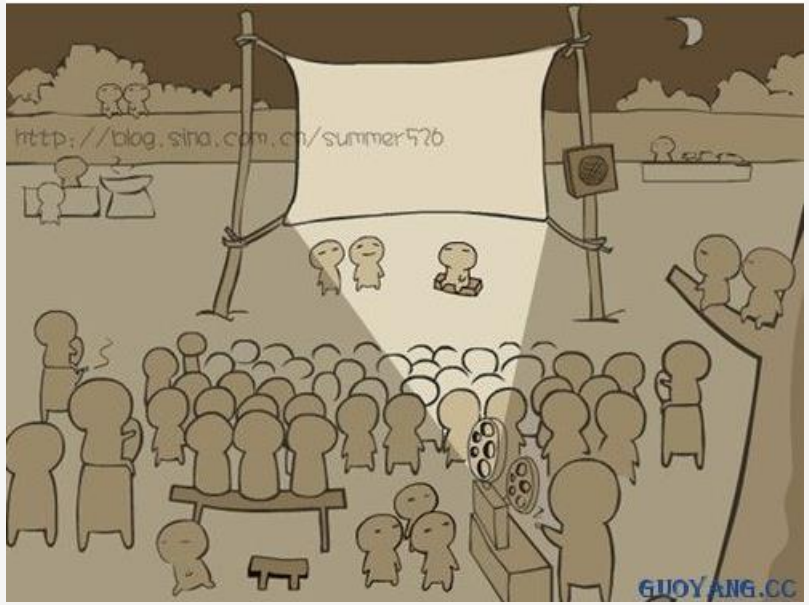
- 怎样知道一个变量的类型呢？
- 在python中，只要定义了一个变量，而且它有数据，那么它的类型就已经确定了，不需要咱们开发者主动的去说明它的类型，系统会自动辨别
- 可以使用type(变量的名字)，来查看变量的类型



1.5 输出

输出

- **1. 普通的输出**
- 生活中的“输出”



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')
```

输出

- **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

输出

- 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。

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

输出

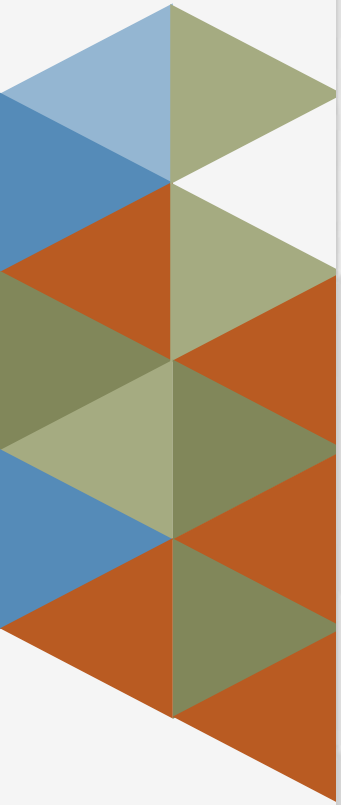
- **3. 换行输出**

- 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

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

输入

- **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

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

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
```

运算符

- 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

运算符

- 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.

运算符

- 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

判断语句

- **<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”功能也经常会用到

判断语句

- **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

判断语句

- **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

判断语句

- **<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

判断语句

- **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')
```

判断语句

- **<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，那么久默认执行else后面的代码，即“第三种性别的特征”相关代码

判断语句

- **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，这样的情况该怎样解决呢？

答：

> if嵌套

判断语句

- **<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

判断语句

- **<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

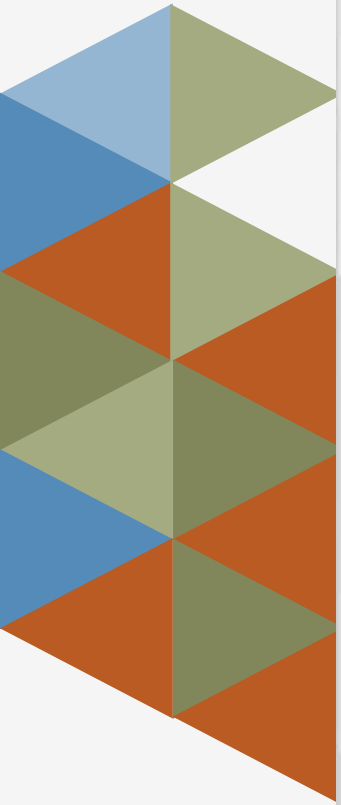
```
没有车票，不能进站
亲爱的，那就下次见了，一票难求啊~~~~(>_<)~~~~
```

想一想:为什么结果3和结果4相同？？？

判断语句

- **<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

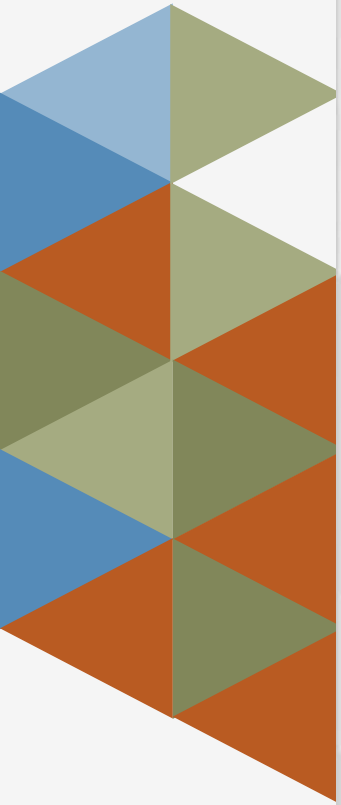
- **while循环应用**

- **1. 计算1~100的累积和（包含1和100）**

```
#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)
```

- **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:
    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。

循环语句-for

- **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

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

2.5 总结