

主流编程语言介绍

May 2017	May 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.639%	-6.32%
2	2		C	7.002%	-6.22%
3	3		C++	4.751%	-1.95%
4	5	⬆	Python	3.548%	-0.24%
5	4	⬇	C#	3.457%	-1.02%
6	10	⬆	Visual Basic .NET	3.391%	+1.07%
7	7		JavaScript	3.071%	+0.73%
8	12	⬆	Assembly language	2.859%	+0.98%
9	6	⬇	PHP	2.693%	-0.30%
10	9	⬇	Perl	2.602%	+0.28%
11	8	⬇	Ruby	2.429%	+0.09%
12	13	⬆	Visual Basic	2.347%	+0.52%
13	15	⬆	Swift	2.274%	+0.68%
14	16	⬆	R	2.192%	+0.86%
15	14	⬇	Objective-C	2.101%	+0.50%
16	42	⬆	Go	2.080%	+1.83%
17	18	⬆	MATLAB	2.063%	+0.78%
18	11	⬇	Delphi/Object Pascal	2.038%	+0.03%
19	19		PL/SQL	1.676%	+0.47%
20	22	⬆	Scratch	1.668%	+0.74%

世界上的编程语言有600多种，但真正大家主流在使用的最多二三十种，不同的语言有自己的特点和擅长领域，随着计算机的不断发展，新语言在不断诞生，也同时有很多老旧的语言慢慢无人用了。有个权威的语言排名网站，可以看到主流的编程语言是哪些

C语言

C语言是一种计算机程序设计语言，它既具有高级语言的特点，又具有汇编语言的特点。它由美国贝尔研究所的D.M.Ritchie于1972年推出，1978年后，C语言已先后被移植到大、中、小及微型机上，它可以作为工作系统设计语言，编写系统应用程序，也可以作为应用程序设计语言，编写不依赖计算机硬件的应用程序。它的应用范围广泛，具备很强的数据处理能力，不仅仅是在软件开发上，而且各类科研都需要用到C语言，适于编写系统软件，三维，二维图形和动画，具体应用比如单片机以及嵌入式系统开发。

适合操作系统底层应用程序、驱动程序、对运行速度要求较高的软件开发

缺点：开发效率低，学习成本高

C++语言

C++是C语言的继承的扩展，它既可以进行C语言的过程化程序设计，又可以进行以抽象数据类型为特点的基于对象的设计，还可以进行以继承和多态为特点的面向对象的设计。C++擅长面向对象程序设计的同时，还可以进行基于过程的设计，因而C++就适应的问题规模而论，大小由之。

C++不仅拥有计算机高效运行的实用性特征，同时还致力于提高大规模程序的编程质量与程序设计语言的问题描述能力。

运行速度快， 适合游戏开发、服务器高性能后台软件、网络编程、图形处理软件、芯片仿真、机器人等

缺点：开发效率高于C, 低于JAVA / Python

JAVA语言

Java是第一个跨平台的面向对象的程序设计语言，是由Sun Microsystems公司于1995年5月推出，Java 技术具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于个人PC、数据中心、游戏控制台、科学超级计算机、移动电话和互联网，同时拥有全球最大的开发者专业社群。在全球云计算和移动互联网的产业环境下，Java更具备了显著优势和广阔前景。

全球使用最广泛的语言，丰富、成熟的生态圈，应用领域极为广泛

缺点：代码臃肿、各种库太多了、设计复杂

PHP语言

PHP（外文名:PHP: Hypertext Preprocessor，中文名：“超文本预处理器”）是一种通用开源脚本语言。语法吸收了C语言、Java和Perl的特点，利于学习，使用广泛，主要适用于Web开发领域

广泛用于网站开发

缺点：应用领域相对单一，运行速度慢

GO语言

Go是从2007年末由谷歌开发，并于2009年11月开源，在2012年早些时候发布了Go 1稳定版本。

Go 的主要特点是开发效率高，并发性好，号称是可以接近C的运行速度，接近Python的开发效率！
未来必成大器！

主要用于高性能并发程序开发，著名的Docker容器就是基于GO开发的

缺点：生态圈还不够成熟，应用领域相对单一

Ruby语言

Ruby是一个与Python类似的解释性语言，开发效率高，学习成本低，著名的ruby on rails web框架在国外有很多人用

主要用于WEB开发，虽然其不只能做WEB开发

缺点：起步晚，它能干的Python都能干，且可以干的更好，所以感觉势能都被Python抢走了。



Python 21天入门

Alex Li

本节内容

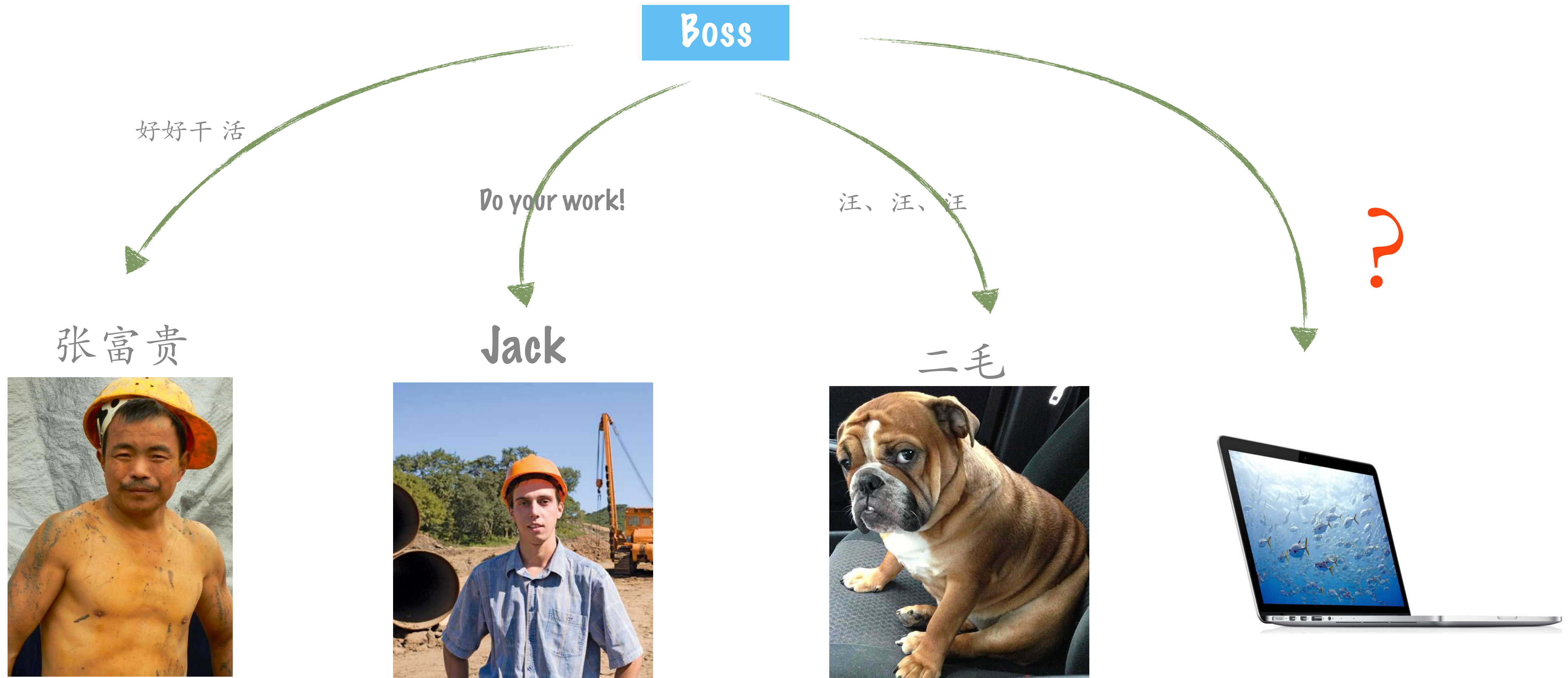
- ▶ 编程语言介绍
- ▶ Python发展史
- ▶ 环境安装
- ▶ 第一个Python程序

什么是编程？为什么要编程？

- ▶ 编程 是个动词，编程==写代码，
- ▶ 写代码为了什么？
 - ▶ 为了让计算机帮你搞事情，比如，马化腾想跟别人聊天，于是写了个聊天软件，这个软件就是一堆代码的集合，这些代码是什么？这些代码是计算机能理解的语言。



如何编程？



计算机可以理解什么语言呢？

```
>>:open "清白之年.mp3"  
>>:start play
```



效率低

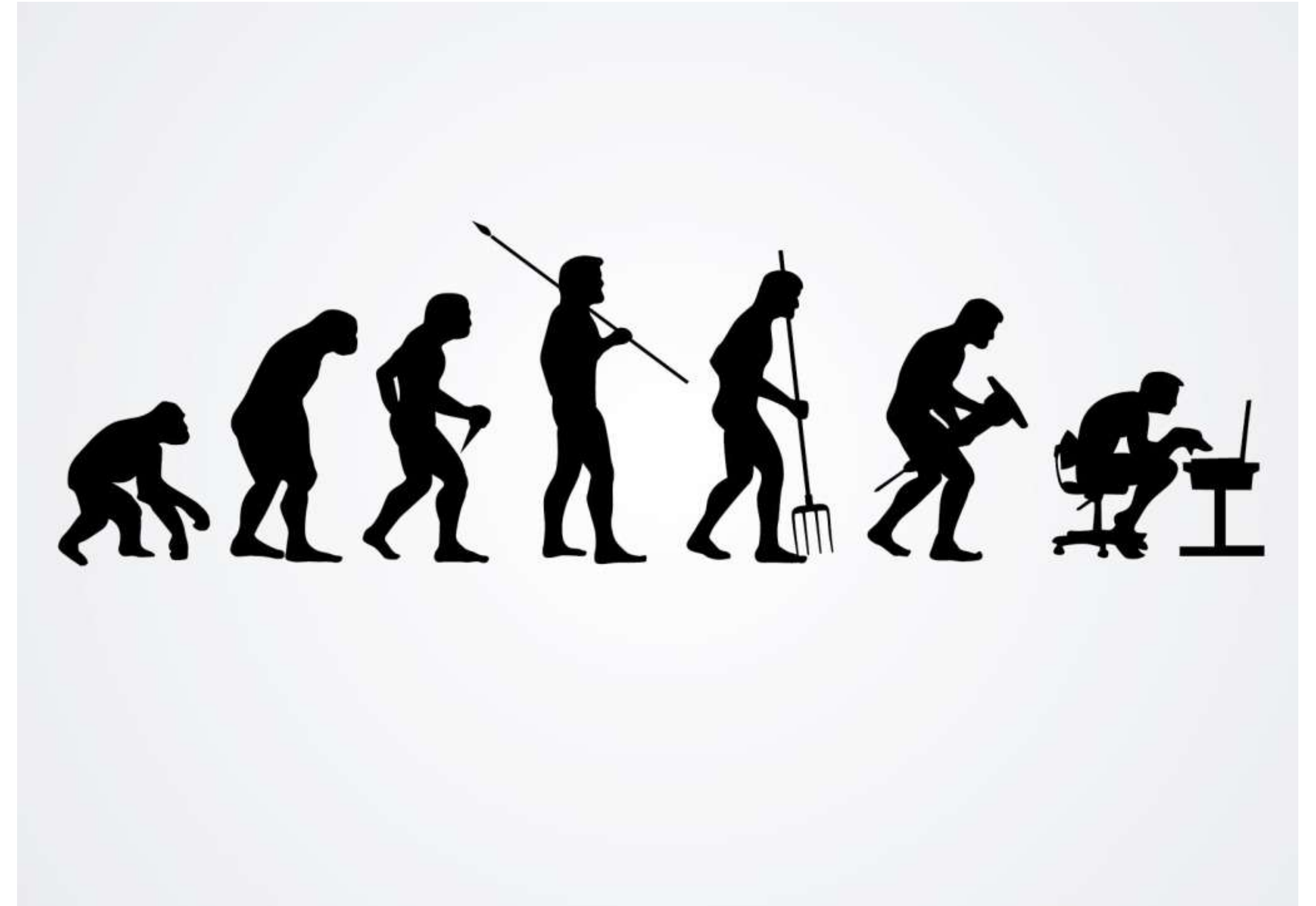


编程语言进化

机器语言

汇编语言

高级语言



机器语言

由于计算机内部只能接受二进制代码，因此，用二进制代码0和1描述的指令称为机器指令，全部机器指令的集合构成计算机的机器语言。

机器语言属于低级语言。

编出的程序全是些0和1的指令代码，直观性差，还容易出错。除了计算机生产厂家的专业人员外，绝大多数的程序员已经不再去学习机器语言了。

机器语言是微处理器理解和使用的，尽管机器语言好像是很复杂的，然而它是有规律的。

存在着多至100000种机器语言的指令。这意味着不能把这些种类全部列出来

以下是一些示例：

指令部份的示例

0000 代表 加载 (LOAD)

0001 代表 存储 (STORE)

...

暂存器部份的示例

0000 代表暂存器 A

0001 代表暂存器 B

...

存储器部份的示例

000000000000 代表地址为 0 的存储器

000000000001 代表地址为 1 的存储器

000000010000 代表地址为 16 的存储器

100000000000 代表地址为 2^{11} 的存储器

集成示例

0000,0000,000000010000 代表 LOAD A, 16

0000,0001,000000000001 代表 LOAD B, 1

0001,0001,000000010000 代表 STORE B, 16

0001,0001,000000000001 代表 STORE B, 1[1]

汇编语言

汇编语言的实质和机器语言是相同的，都是直接对硬件操作，只不过指令采用了英文缩写的标识符，更容易识别和记忆。

它同样需要编程者将每一步具体的操作用命令的形式写出来。汇编程序的每一句指令只能对应实际操作过程中的一个很细微的动作。例如移动、自增，因此汇编源程序一般比较冗长、复杂、容易出错，而且使用汇编语言编程需要有更多的计算机专业知识。

但汇编语言的优点也是显而易见的，用汇编语言所能完成的操作不是一般高级语言所能够实现的，而且源程序经汇编生成的可执行文件不仅比较小，而且执行速度很快。

```
; hello.asm
section .data          ; 数据段声明
    msg db "Hello, world!", 0xA    ; 要输出的字符串
    len equ $ - msg          ; 字符串长度
section .text          ; 代码段声明
global _start          ; 指定入口函数
_start:                ; 在屏幕上显示一个字符串
    mov edx, len        ; 参数三：字符串长度
    mov ecx, msg        ; 参数二：要显示的字符串
    mov ebx, 1          ; 参数一：文件描述符(stdout)
    mov eax, 4          ; 系统调用号(sys_write)
    int 0x80            ; 调用内核功能
                        ; 退出程序
    mov ebx, 0          ; 参数一：退出代码
    mov eax, 1          ; 系统调用号(sys_exit)
    int 0x80            ; 调用内核功能
```

F**k, being a coder is even harder than president!
妈的，做程序员感觉比当总统还难呀！



高级语言

高级语言是大多数编程者的选择。和汇编语言相比，它不但将许多相关的机器指令合成为单条指令，并且去掉了与具体操作有关但与完成工作无关的细节，例如使用堆栈、寄存器等，这样就大大简化了程序中的指令。同时，由于省略了很多细节，编程者也就不需要有太多的专业知识。

高级语言主要是相对于汇编语言而言，它并不是特指某一种具体的语言，而是包括了很多编程语言，如C\C++，JAVA，PHP,Python, GO，C#等都属于高级语言

相比机器和汇编语言，高级语言对开发人员更友好，在开发效率上大大提高！

```
27 print("Hello World!")
28
29 name = "Alex Li"
30
31 print("Hello, my name is",name )
32
33
```

高级语言分类

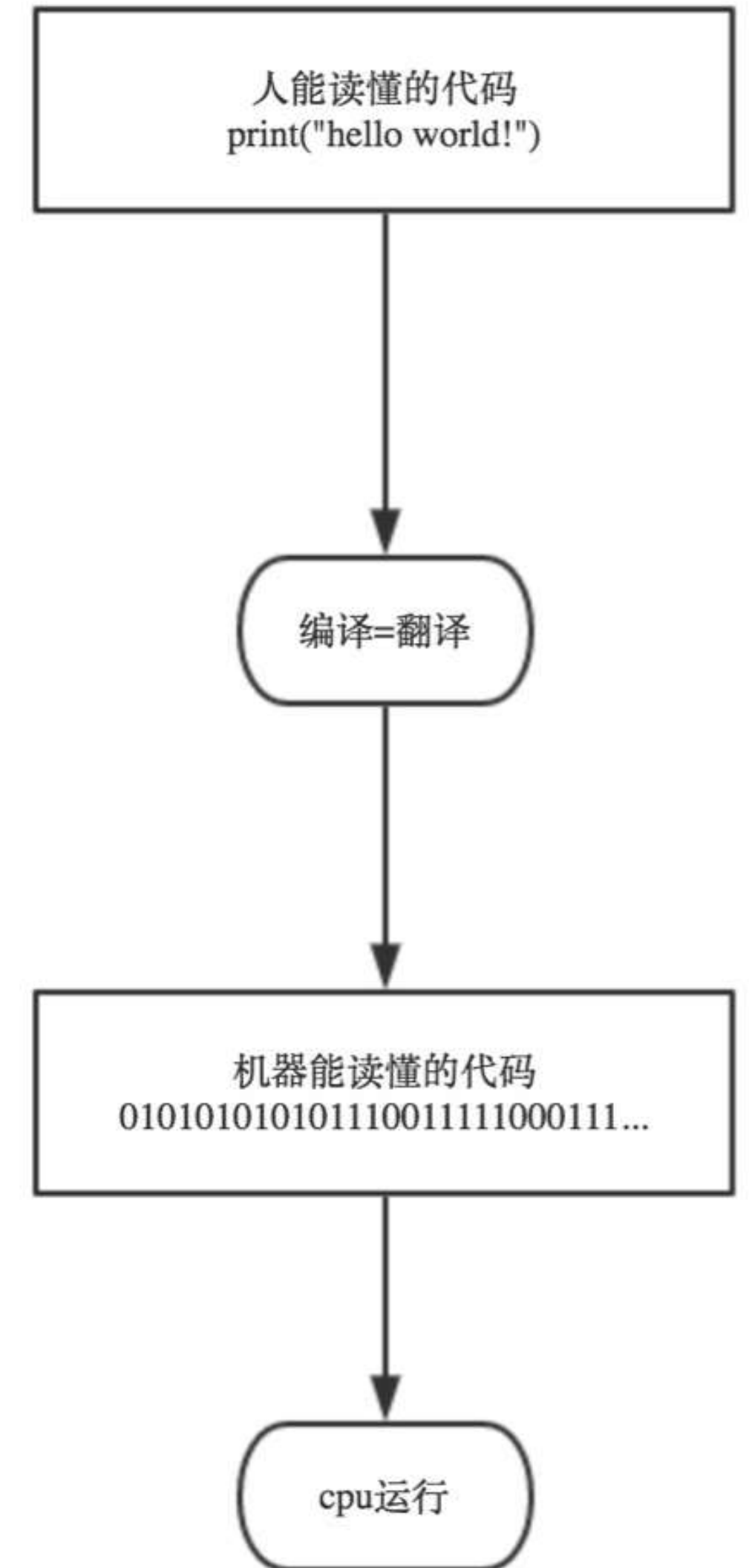
高级语言所编制的程序不能直接被计算机识别，必须经过转换才能被执行！

按转换方式可将它们分为两类：

编译类：

编译是指在应用源程序执行之前，就将程序源代码“翻译”成目标代码（机器语言），因此其目标程序可以脱离其语言环境独立执行(编译后生成的可执行文件，是cpu可以理解的2进制的机器码组成的)，使用比较方便、效率较高。但应用程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件（*.obj，也就是OBJ文件）才能执行，只有目标文件而没有源代码，修改很不方便。

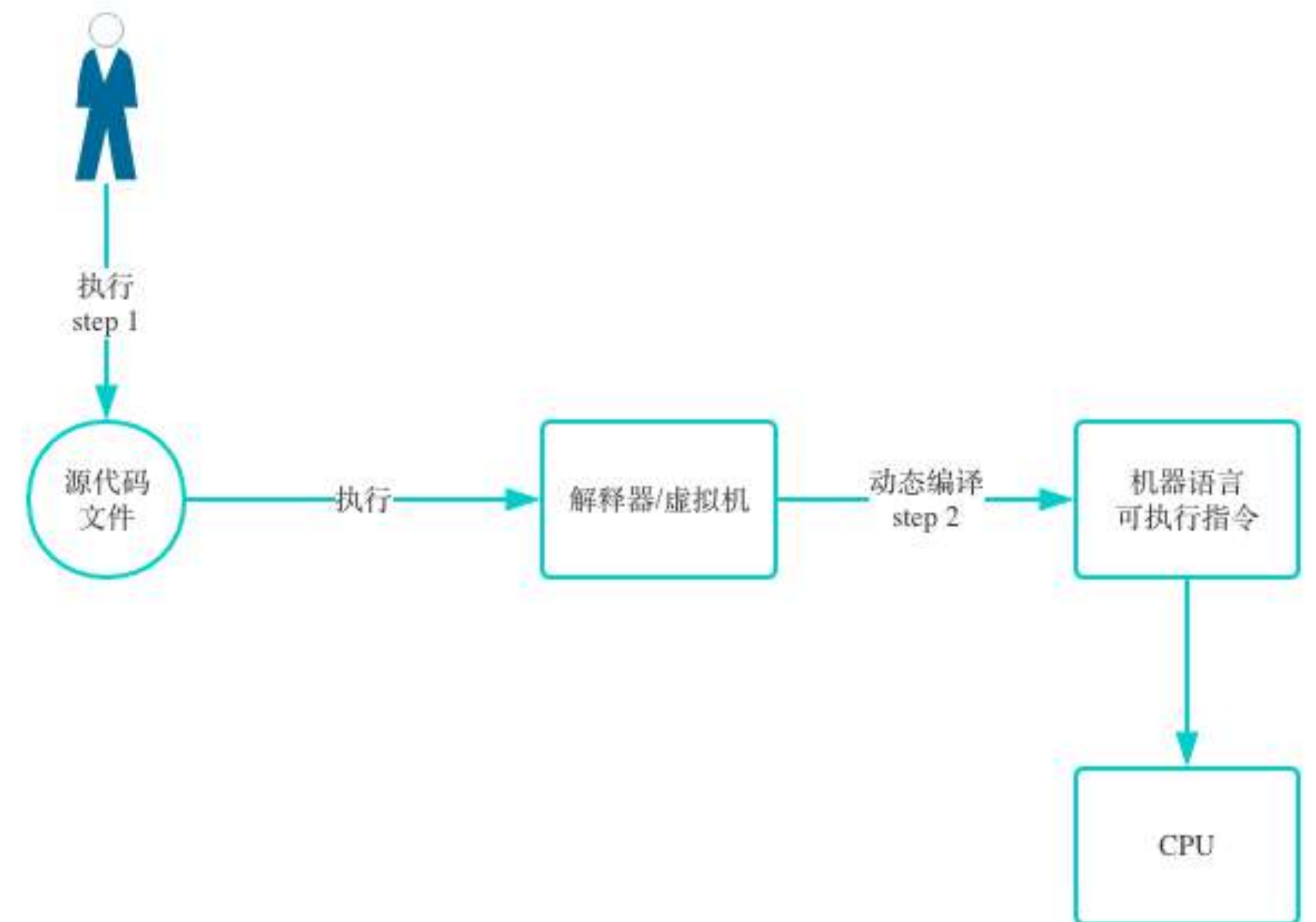
编译后程序运行时不需要重新翻译，直接使用编译的结果就行了。程序执行效率高，依赖编译器，跨平台性差些。如C、C++、Delphi等



高级语言-解释型

解释类：

执行方式类似于我们日常生活中的“同声翻译”，应用程序源代码一边由相应语言的解释器“翻译”成目标代码（机器语言），一边执行，因此效率比较低，而且不能生成可独立执行的可执行文件，应用程序不能脱离其解释器(想运行，必须先装上解释器，就像跟老外说话，必须有翻译在场)，但这种方式比较灵活，可以动态地调整、修改应用程序。如 Python、Java、PHP、Ruby等语言。



编译VS解释型

编译型：

1. 把源代码编译成机器语言的可执行程序
2. 执行 可执行程序文件

优点：

1. 程序执行时，不再需要源代码，不依赖语言环境，因为执行的是机器码文件
2. 执行速度快，因为你的程序代码已经翻译成了是计算机可以直接理解的机器语言，

缺点：

1. 每次修改了源代码，需要重新编译，生成机器码文件
2. 跨平台性不好，不同操作系统，调用底层的机器指令不同，需为不同平台生成不同的机器码文件

解释型：

1. 用户调用解释器，执行源代码文件
2. 解释器把源代码文件边解释成机器指令，边交给cpu执行

优点：

1. 天生跨平台， 因为解释器已经做好了对不同平台的交互处理，用户写的源代码不需再考虑平台差异性，可谓，一份源代码，所有平台都可直接执行
2. 随时修改，立刻见效，改完源代码后，直接运行看效果

缺点：

1. 运行效率低，所有的代码均需经过解释器边 解释边执行 ， 速度比编译型慢很多
2. 代码是明文

小节

机器语言

优点是最底层，速度最快，缺点是最复杂，开发效率最低

汇编语言

优点是比较底层，速度最快，缺点是复杂，开发效率最低

高级语言

编译型语言执行速度快，不依赖语言环境运行，跨平台差

解释型跨平台好，一份代码，到处使用，缺点是执行速度慢，依赖解释器运行



Python发展史



吉多·范罗苏姆 (Guido van Rossum)

1989年的圣诞节期间，Guido开始写Python语言的编译器。Python这个名字，来自Guido所挚爱的电视剧Monty Python's Flying Circus。他希望这个新的叫做Python的语言，能符合他的理想：创造一种C和shell之间，功能全面，易学易用，可拓展的语言。

2005年加入谷歌至2012年，2013年加入Dropbox 直到现在

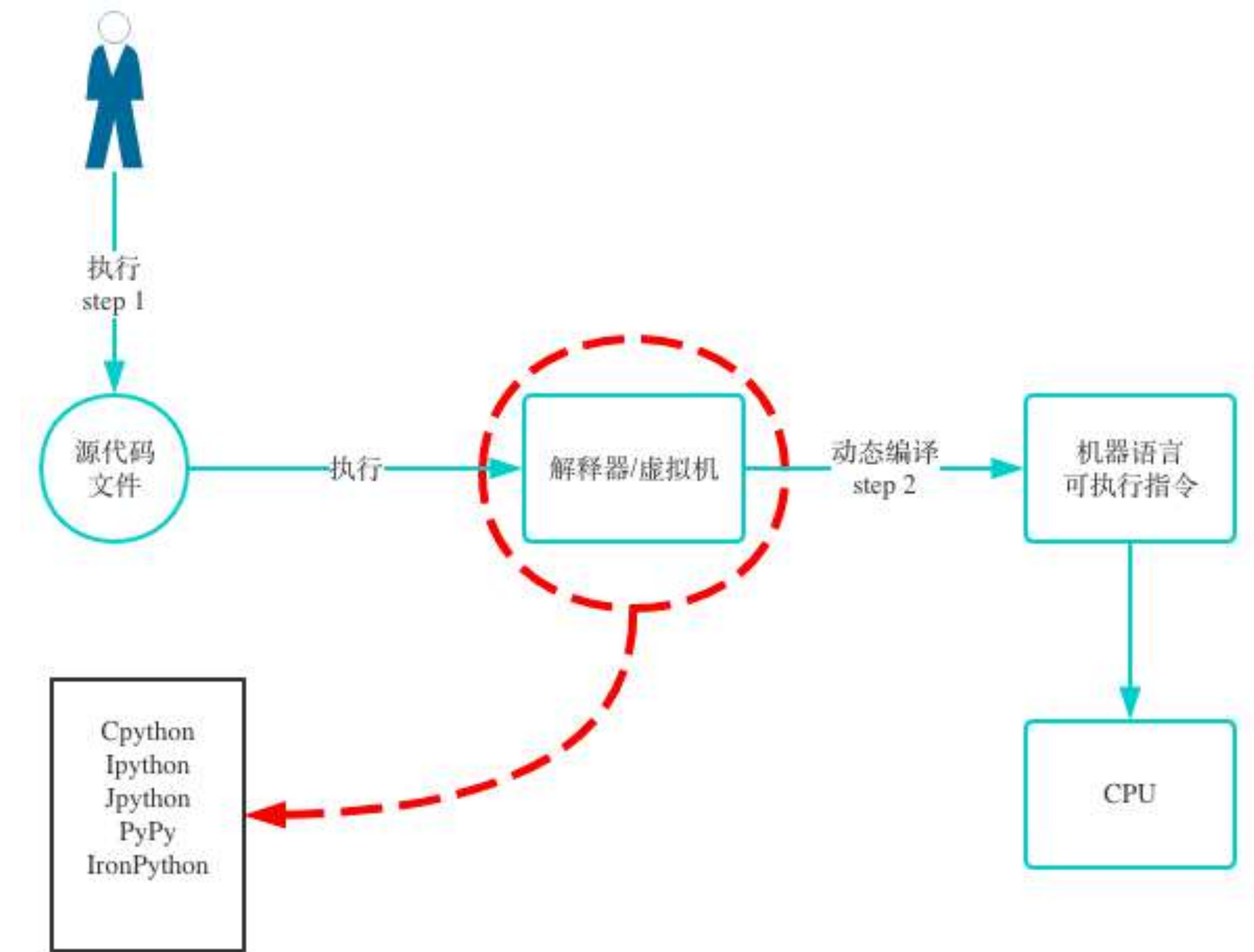
依然掌握着Python发展的核心方向，被称为仁慈的独裁者

Python发展史

- ★ 1989年,3.Guido开始写Python语言的编译器。
- ★ 1990年,3第一个Python编译器诞生。它是用C语言实现的，并能够调用C语言的库文件。从一出生，Python已经具有了：类，函数，异常处理，包含表和词典在内的核心数据类型，以及模块为基础的拓展系统。
- ★ Python 3.4 – May 16, 2014 frameworks, Zope 1 was released in 1999
- ★ Python 1.0 – January 1994 增加了lambda, Python 2.7 would be supported until 2020, and reaffirmed that there would be
- ★ Python 2.0 – October 16, 2000 加入了内存回收机制, Python 3.4 was as soon as possible
- ★ Python 2.4 – September 30, 2004, 同年目前最流行的WEB框架Django 诞生
- ★ Python 2.6 – September 19, 2006 发布python3.6.0版
- ★ Python 2.6 – October 1, 2008+
- ★ Python 3.0 – December 3, 2008
- ★ Python 2.7 – July 3, 2010

Python解释器种类

我们现在知道了Python是一门解释型语言，代码想运行，必须通过解释器执行，Python的解释器本身也可以看作是个程序（翻译官司是哪国人不重要），这个程序是什么语言开发的呢？答案是好几种语言？what？因为Python有好几种解释器，分别基于不同语言开发，每个解释器特点不同，但都能正常运行我们的Python代码



Python解释器种类

CPython

当我们从Python官方网站下载并安装好Python 2.7后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。

CPython是使用最广且被的Python解释器。教程的所有代码也都在CPython下执行。

IPython

IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。好比很多国产浏览器虽然外观不同，但内核其实都是调用了IE。

PyPy

PyPy是另一个Python解释器，它的目标是执行速度。PyPy采用JIT技术，对Python代码进行动态编译（注意不是解释），所以可以显著提高Python代码的执行速度。

Jython

Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。

IronPython

IronPython和Jython类似，只不过IronPython是运行在微软.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码

Python 2.x or 3.x ?

In summary : 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.

目前虽然业内很多企业还在大量使用Python 2.6 or 2.7，因为旧项目几十万甚至上百万行的代码想快速升级到3.0不是件容易的事，但是大家在开发新项目时几乎都会使用3.x。

另外Python 3 确实比2.x做了很多的改进，直观点来讲，就像从XP升级到Win7的感觉一样，棒棒的。

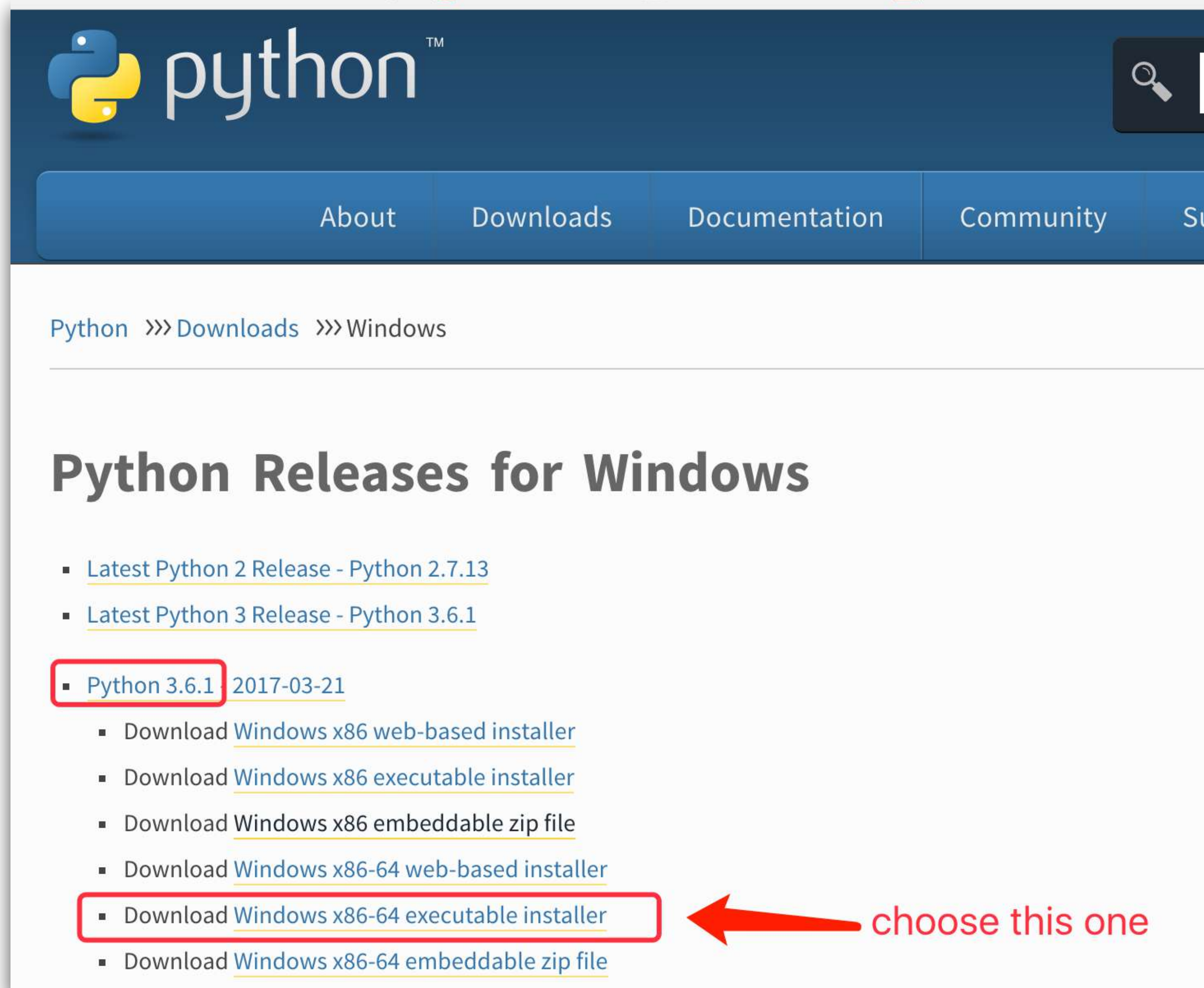
Python 环境安装

Windows安装

打开官网 <https://www.python.org/downloads/windows/> 下载中心

测试安装是否成功

windows --> 运行 --> 输入cmd，然后回车，弹出cmd程序，输入python，如果能进入交互环境，代表安装成功。



python™

About Downloads Documentation Community

Python >>> Downloads >>> Windows

Python Releases for Windows

- [Latest Python 2 Release - Python 2.7.13](#)
- [Latest Python 3 Release - Python 3.6.1](#)
- **Python 3.6.1** 2017-03-21
 - Download [Windows x86 web-based installer](#)
 - Download [Windows x86 executable installer](#)
 - Download [Windows x86 embeddable zip file](#)
 - Download [Windows x86-64 web-based installer](#)
 - **Download [Windows x86-64 executable installer](#)**
 - Download [Windows x86-64 embeddable zip file](#)

choose this one

用notepad++创建一个文件，输入以下代码

```
print("Hello World!")
```

```
print("Python好简单呀，我要学好挣大钱！")
```

保存为HelloWorld.py，注意.py后缀名的作用

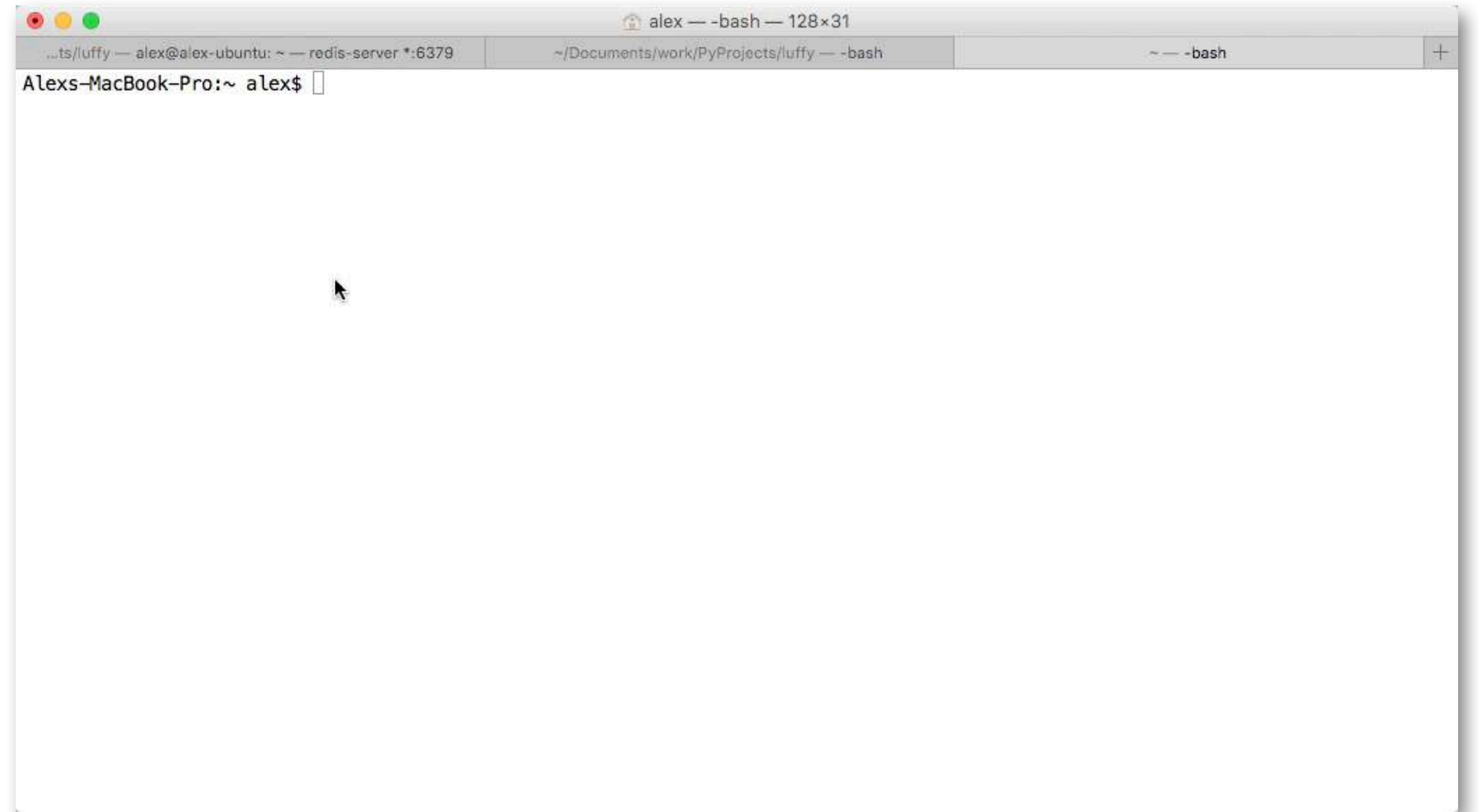
进入cmd命令行，执行python HelloWorld.py, 看结果

*注意文件名前面加python 的原因是要把代码交给python解释器去解释执行

Hello World!

交互器执行

Python交互器是主要用来对代码进行调试用的



5分钟精通各种语言

C++

```
#include <iostream>
int main(void)
{
    std::cout<<"Hello world";
}
```

C

```
#include <stdio.h>
int main(void)
{
    printf("\nhello world!");
    return 0;
}
```

JAVA

```
public class HelloWorld{
    // 程序的入口
    public static void main(String args[]){
        // 向控制台输出信息
        System.out.println("Hello World!");
    }
}
```

PHP

```
<?php
    echo "hello world!";
?>
```

Ruby

```
puts "Hello world."
```

GO

```
package main

import "fmt"

func main(){
    fmt.Printf("Hello World!\n God Bless You!");
}
```

变量

计算机的主要作用之一是进行运算，用python进行数值运算非常容易，跟我们平常用计算器一样简单

```
(venv) Alexs-MacBook-Pro:luffy alex$ python
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 4*2/6 + 1
2.3333333333333333
>>>
>>> 100 / (2*5)
10.0
```

背景提要:

姗姗前男友花钱如流水，为了帮助他理财，我们决定为他出个月底消费报表，报表中有2个重要数据,当月总花费和分类汇总费用，即总共花了多少钱吃饭、买衣服等。

变量

现在要求你用程序 把 每个消费分类统计 和总消费依次计算并打印出来，你怎么做呢？

	A	B	C	D	E	
1		吃饭	买衣服	交通	精神消费	
2	1月1日	10	20	6	300	
3	1月2日	15		6		
4	1月3日	7		6		
12	1月11日	7		6		
13	1月12日	4		6		
14	1月13日	7		6		
15	1月14日	3		6	300	
26	1月25日	7		6		
27	1月26日	4		6		
28	1月27日	7		6	400	
29	1月28日	3		6		
30	1月29日	5		6		
31	1月30日	20		6		
32	1月31日	4		6	200	
33	Total	265	20	186	1200	
34						

	MEMO NO.
	DATE
吃饭 = 10 + 15 + 7 + 7 + 4 + 7 + ... = 265	
买衣服 = 20 = 20	
交通 = 6 + 6 + 6 + ... + 6 = 186	
大消费	
精神消费 = 300 + 300 + 400 + 200 = 1200	
总消费 = 265 + 20 + 186 + 1200 = 1671	

程序实现

你发现没有？你在最后在算总消费的时候直接用的是之前已经算好的中间结果，为什么这么做？都知道这样是为了避免重新再算一遍所有的数据。那在程序中呢？

```
>>> print('eat',10+15+7+4+7+3)
eat 46
>>> print('cloth',20)
cloth 20
>>> print('traffic',6+6+6+6+6)
traffic 30
>>> print('精神',300+300+400+200)
精神 1200
>>>
>>>
>>> print('总消费', 46+20+30+1200)
总消费 1296
```

程序实现

我的亲，你这么写是有问题的，啥问题？你最后算总消费的时候 是人肉 把之前算出来的分类结果 填进去的， 但是我们把程序写在脚本里运行时， 你肯定不会预先知道吃饭、交通、买衣服3个分类的结果的，这个结果是动态算出来的，那你如何把这3个动态结果做为总消费运算的数据源呢？

答案简单， 直接把每个分类结果先起个名字存下来，然后计算总消费的时候，只需要把之前存下来的几个名字调用 一下就可以啦！

```
>>> eat = 10+15+7+4+7+3
>>> cloth = 20
>>> traffic = 6+6+6+6+6
>>> 精神=300+300+200+400
>>>
>>> total = eat + cloth + traffic + 精神
>>> print('总消息',total)
总消息 1296
```

eat,cloth,traffic,精神,total这几个名字的作用，就是把程序运算的中间结果临时存到内存里，以备后面的代码继续调用，这几个名字的学名就叫做“变量”

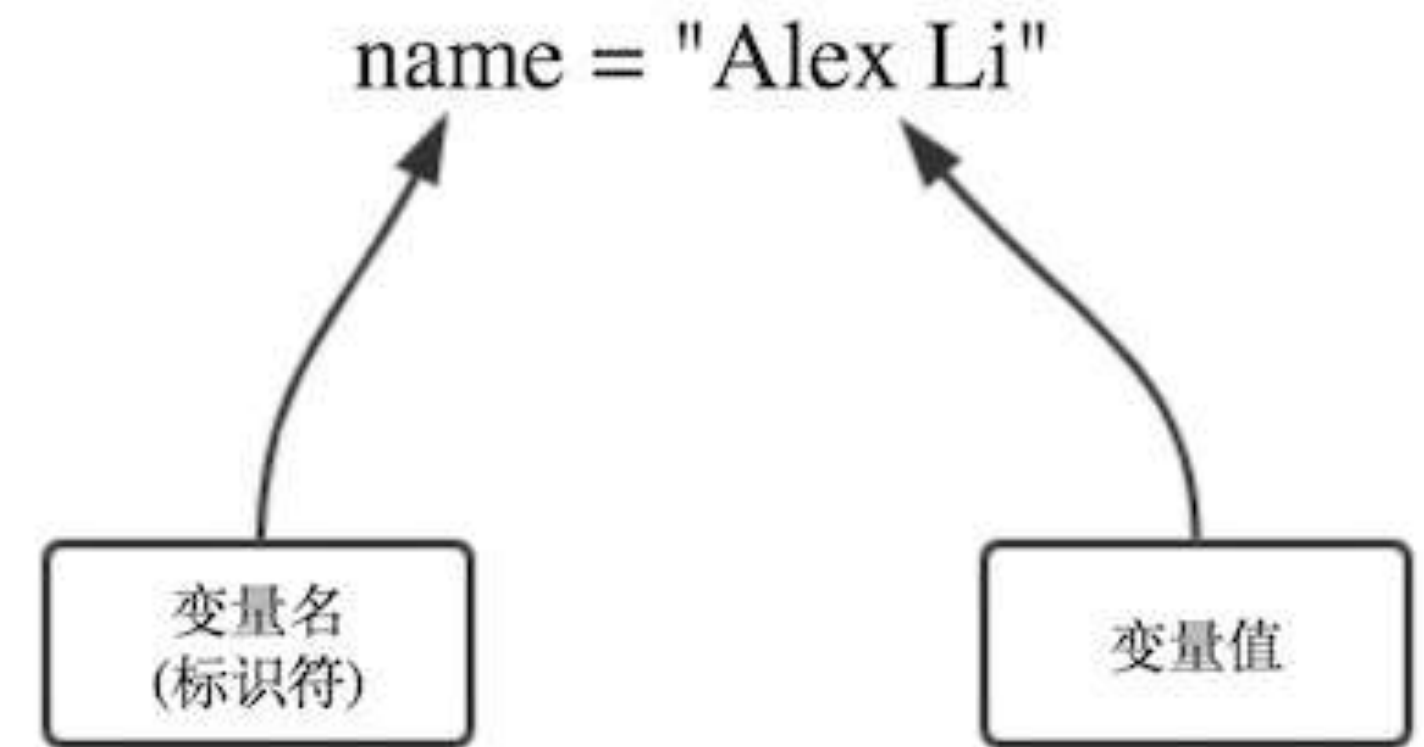
变量的作用

Variables are used to store information to be referenced and manipulated in a computer program. They also provide a way of labelling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of variables as containers that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

变量定义规范

声明变量

```
name = "Alex Li"
```



变量定义规则

- 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']

变量命名习惯

驼峰体

```
AgeOfOldboy = 56  
NumberOfStudents = 80
```

下划线

```
age_of_oldboy = 56  
number_of_students = 80
```

官方推荐用哪种?

定义变量的Low方式

变量名为中文、拼音

变量名过长

变量名词不达意

```
>>> 老男孩的年龄=56
>>> oldboy_nianling = 56
>>> the_name_of_oldboy_girlfriend = 'Lisa'
>>>
>>> name1 = 'oldboy'
>>> name2= '北京市昌平区'
```

常量

常量即指不变的量，如pai 3.141592653..., 或在程序运行过程中不会改变的量

在Python中没有一个专门的语法代表常量，程序员约定俗成用变量名全部大写代表常量

```
AGE_OF_OLDBOY = 56
```

在c语言中有专门的常量定义语法，const int count = 60;一旦定义为常量，更改即会报错

读取用户输入

```
name = input("What is your name?")  
  
print("Hello " + name )
```

执行脚本就会发现，程序会等待你输入姓名后再往下继续走。

可以让用户输入多个信息，如下

```
name = input("What is your name?")  
age = input("How old are you?")  
hometown = input("Where is your hometown?")  
  
print("Hello ", name , "your are ", age , "years old, you came from", hometown)
```


注释

代码注释用#

代码注释原则:

不用全部加注释，只需要在自己觉得重要或不好理解的部分加注释即可
注释可以用中文或英文，但绝对不要拼音噢

```
class HttpResponseBase(six.Iterator):
    """
    An HTTP response base class with dictionary-accessed headers.

    This class doesn't handle content. It should not be used directly.
    Use the HttpResponse and StreamingHttpResponse subclasses instead.
    """

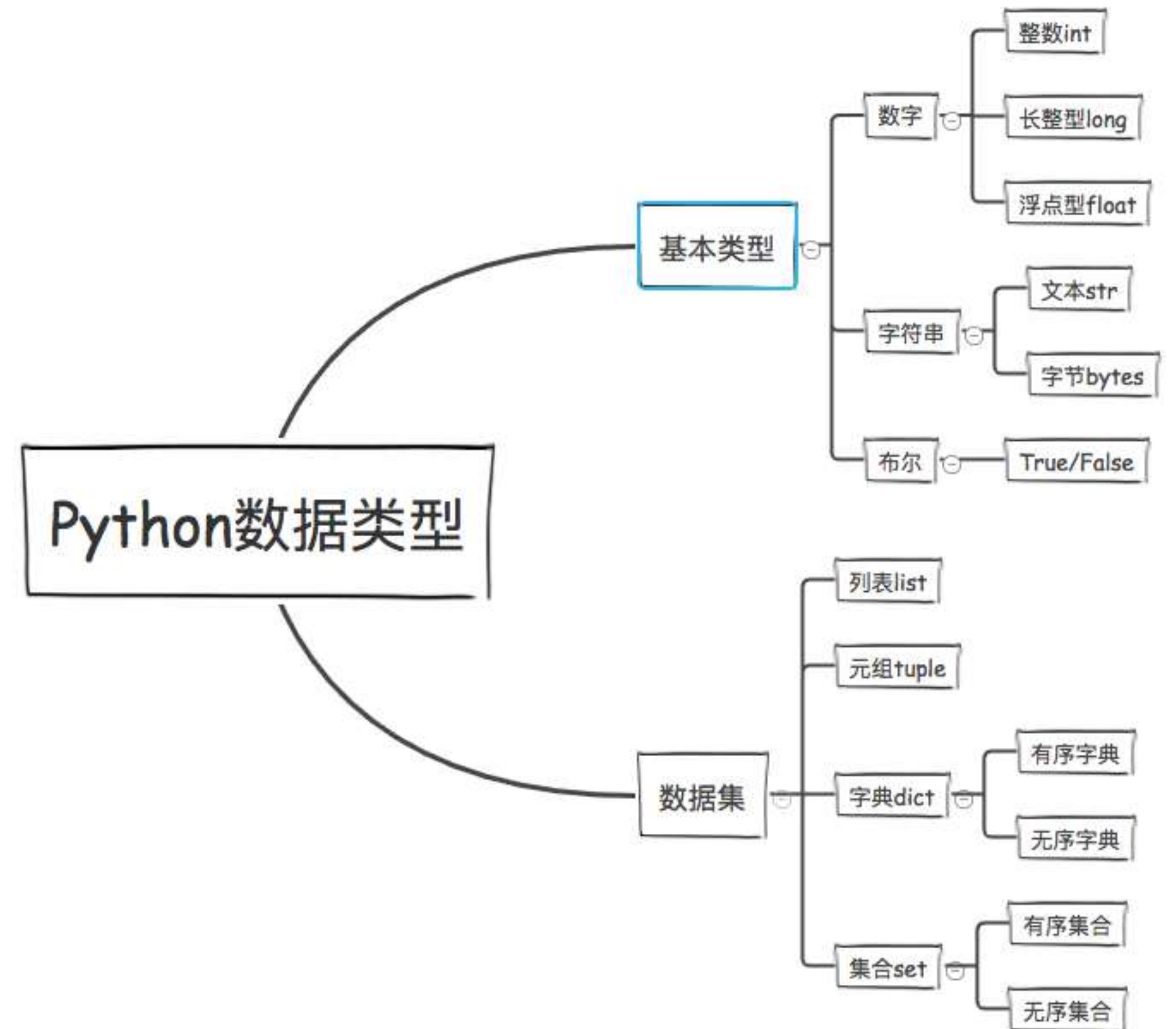
    status_code = 200

    def __init__(self, content_type=None, status=None, reason=None, charset=None):
        # _headers is a mapping of the lower-case name to the original case of
        # the header (required for working with legacy systems) and the header
        # value. Both the name of the header and its value are ASCII strings.
        self._headers = {}
        self._closable_objects = []
        # This parameter is set by the handler. It's necessary to preserve the
        # historical behavior of request_finished.
        self._handler_class = None
        self.cookies = SimpleCookie()
        self.closed = False
        if status is not None:
            self.status_code = status
        self._reason_phrase = reason
        self._charset = charset
        if content_type is None:
            content_type = '%s; charset=%s' % (settings.DEFAULT_CONTENT_TYPE,
                                                self.charset)
        self['Content-Type'] = content_type
```

数据类型

什么是数据类型？

我们人类可以很容易的分清数字与字符的区别，但是计算机并不能呀，计算机虽然很强大，但从某种角度上看又很傻，除非你明确的告诉它，1是数字，“汉”是文字，否则它是分不清1和‘汉’的区别的，因此，在每个编程语言里都会有一个叫数据类型的东东，其实就是对常用的各种数据类型进行了明确的划分，你想让计算机进行数值运算，你就传数字给它，你想让他处理文字，就传字符串类型给他。Python中常用的数据类型有哪些呢



数据类型-数字类型

int（整型）

- 在32位机器上，整数的位数为32位，取值范围为 $-2^{31} \sim 2^{31}-1$ ，即 $-2147483648 \sim 214748364$
- 在64位系统上，整数的位数为64位，取值范围为 $-2^{63} \sim 2^{63}-1$ ，即 $-9223372036854775808 \sim 9223372036854775807$

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。

```
>>> a= 2**64
>>> type(a) #type()是查看数据类型的方法
<type 'long'>
>>> b = 2**60
>>> type(b)
<type 'int'>
```

- 注意在Python3里不再有long类型了，全都是int
- 除了int和long之外， 其实还有float浮点型，复数型，但今天先不讲啦

数据类型-字符串

在Python中,加了引号的字符都被认为是字符串!

```
>>> name = "Alex Li" #双引号
>>> age = "22"        #只要加引号就是字符串
>>> age2 = 22          #int
>>>
>>> msg = '''My name is Alex, I am 22 years old!''' #我擦, 3个引号也可以
>>>
>>> hometown = 'ShanDong' #单引号也可以
```

那单引号、双引号、多引号有什么区别呢?

单双引号本有任何区别, 只有下面这种情况 你需要考虑单双的配合

```
msg = "My name is Alex , I'm 22 years old!"
```

多引号什么作用呢?

作用就是多行字符串必须用多引号

```
msg = '''
今天我想写首小诗,
歌颂我的同桌,
你看他那乌黑的短发,
好像一只炸毛鸡。
'''
```

```
print(msg)
```

字符串拼接

数字可以进行加减乘除等运算，字符串呢？让我大声告诉你也能！what ?是的，但只能进行"相加"和"相乘"运算。

```
>>> name
'Alex Li'
>>> age
'22'
>>>
>>> name + age  #相加其实就是简单拼接
'Alex Li22'
>>>
>>> name * 10 #相乘其实就是复制自己多少次，再拼接在一起
'Alex LiAlex LiAlex LiAlex LiAlex LiAlex LiAlex LiAlex LiAlex Li'
```

注意，字符串的拼接只能是双方都是字符串，不能跟数字或其它类型拼接单双引号

```
>>> type(name),type(age2)
(<type 'str'>, <type 'int'>)
>>>
>>> name
'Alex Li'
>>> age2
22
>>> name + age2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects #错误提示数字 和 字符 不能拼接
```

数据类型-布尔型

布尔类型很简单，就两个值，一个True(真)，一个False(假)，主要用于逻辑判断！

你是不是不明白？

```
>>> a=3
>>> b=5
>>>
>>> a > b #不成立就是False,即假
False
>>>
>>> a < b #成立就是True, 即真
True
```

```
if a > b
    print(a is bigger than b )

else
    print(a is smaller than b )
```

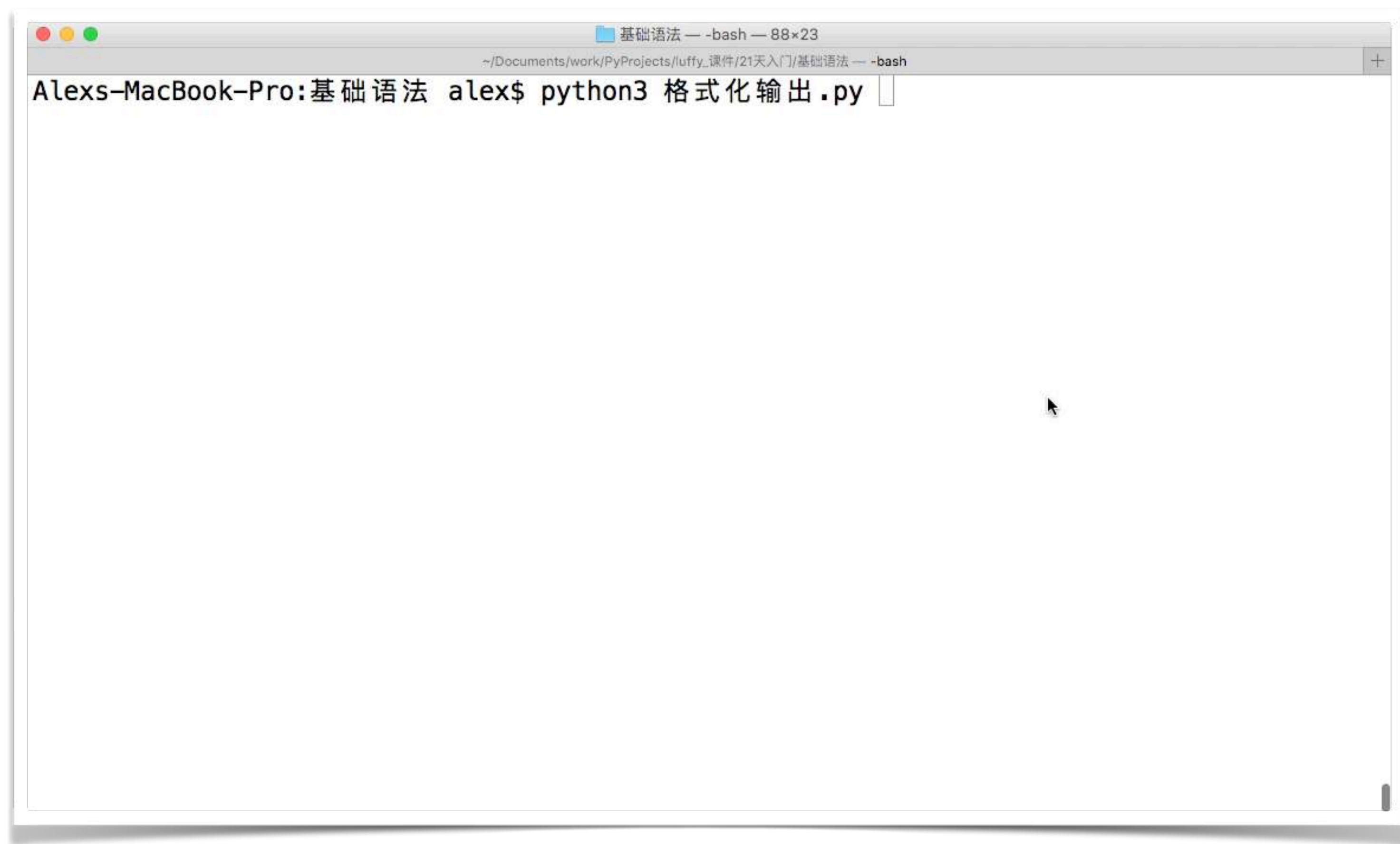
上面是伪代码，但是不是意味着， 计算机就可以根据判断结果不同，来执行不同的动作啦？

计算机为什么要描述这种条件呢？

因为接下来就可以根据条件结果来干不同的事情啦呀！

格式化输出

如何实现这个小程序？



运算符

计算机可以进行的运算有很多种，可不只加减乘除这么简单，运算按种类可分为算数运算、比较运算、逻辑运算、赋值运算、成员运算、身份运算、位运算，今天我们暂只学习算数运算、比较运算、逻辑运算、赋值运算

算数运算

以下假设变量：a=10，b=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

运算符

比较运算

以下假设变量：a=10，b=20

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	(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。

运算符

赋值运算

以下假设变量：a=10，b=20

运算符	描述	实例
=	简单的赋值运算符	<code>c = a + b</code> 将 <code>a + b</code> 的运算结果赋值为 <code>c</code>
+=	加法赋值运算符	<code>c += a</code> 等效于 <code>c = c + a</code>
-=	减法赋值运算符	<code>c -= a</code> 等效于 <code>c = c - a</code>
*=	乘法赋值运算符	<code>c *= a</code> 等效于 <code>c = c * a</code>
/=	除法赋值运算符	<code>c /= a</code> 等效于 <code>c = c / a</code>
%=	取模赋值运算符	<code>c %= a</code> 等效于 <code>c = c % a</code>
**=	幂赋值运算符	<code>c **= a</code> 等效于 <code>c = c ** a</code>
//=	取整除赋值运算符	<code>c //= a</code> 等效于 <code>c = c // a</code>

运算符

逻辑运算

运算符	描述
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。
or	布尔"或" - 如果x是True，它返回True，否则它返回y的计算值。
not	布尔"非" - 如果x为True，返回False。如果x为False，它返回True。

```
>>> a
10
>>> b
20
>>>
>>> a == 10 and b == 20
True
>>> a == 10 and b == 21
False
>>>
>>> a == 10 or b == 21
True
>>> a == 11 or b == 21
False
>>> a == 11 or b == 20
True
>>>
>>> a == 11 or b == 21
False
>>> not a == 11 or b == 21
True
```

流程控制

单分支

`if` 条件:

满足条件后要执行的代码

练习:

- 输入姓名、性别, 判断如果是女生, 打印, 我喜欢女生!
- 输入姓名、性别, 判断如果是女生且年龄小于28岁, 打印, 我喜欢女生!

```
if else.py x
if AgeOfOldboy ...
1  # *_coding:utf-8_*
2
3
4  AgeOfOldboy = 56
5
6  if AgeOfOldboy > 50 :
7      print("Too old, time to retire..")
```

Annotations:

- 顿号: 是语法格式 (points to the colon in line 6)
- 4个空格 (points to the indentation in line 7)
- if条件成立后要执行的代码 (points to the print statement in line 7)

Run if else

```
/usr/local/bin/python3.5 "/Users/alex/Documents/work/PyProjects/python基础/outline大纲/chapter1/if
Too old, time to retire..
Process finished with exit code 0
```

结果

流程控制

双分支

```
if 条件:
    满足条件执行代码
else:
    if条件不满足就走这段
```

```
AgeOfOldboy = 48

if AgeOfOldboy > 50 :
    print("Too old, time to retire..")
else:
    print("还能折腾几年!")
```

练习：

- 输入姓名、性别，判断如果是女生，打印我喜欢女生，否则，打印 一起来搞基！
- 输入姓名、性别、年龄， 判断如果是女生且年龄小于28岁，打印我喜欢女生，否则，打印姐弟恋也很好奥！
- 输入姓名、性别、年龄， 判断如果是女生且年龄小于28岁，打印我喜欢女生，否则，打印姐弟恋也很好奥！如果是男生，打印一起来搞基！

流程控制

多分支

```
if 条件:
    满足条件执行代码
elif 条件:
    上面的条件不满足就走这个
elif 条件:
    上面的条件不满足就走这个
elif 条件:
    上面的条件不满足就走这个
else:
    上面所有的条件不满足就走这段
```

```
age_of_oldboy = 48

guess = int(input(">>:"))

if guess > age_of_oldboy :
    print("猜的太大了, 往小里试试...")

elif guess < age_of_oldboy :
    print("猜的太小了, 往大里试试...")

else:
    print("恭喜你, 猜对了...")
```

练习:

- 写个猜年龄的游戏吧

流程控制

再来个匹配成绩的小程序吧，成绩有ABCDE5个等级，与分数的对应关系如下

A	90-100
B	80-89
C	60-79
D	40-59
E	0-39

要求用户输入0-100的数字后，你能正确打印他的对应成绩

猜年龄游戏升级版，允许用户最多猜3次！

```
age = 26
user_guess = int(input("your guess:"))
if user_guess == age :
    print("恭喜你答对了，可以抱得傻姑娘回家！")
elif user_guess < age :
    print("try bigger")
else :
    print("try smaller")
```

即使是小白的你，也觉得的太low了是不是，以后要修改功能还得修改3次。

因此记住，写重复的代码是程序员最不耻的行为。

那么如何做到不用写重复代码又能让程序重复一段代码多次呢？ 循环语句就派上用场啦

while 循环

语法

```
while 条件:  
    执行代码...
```

写个让程序从0打印到100的程序，每循环一次，+1+

```
count = 0  
while count <= 100 : #只要count<=100就不断执行下面的代码  
    print("loop ", count )  
    count +=1 #每执行一次，就把count+1，要不然就变成死循环啦，因为count一直是0
```

练习：

如果我想实现打印1到100的偶数怎么办呢？

循环打印1-100，第50次不打印值，第60-80次，打印对应值的平方。

Dead Loop

有一种循环叫死循环，一经触发，就运行个天荒地老、海枯石烂。

while 是只要后边条件成立(也就是条件结果为真)就一直执行，怎么让条件一直成立呢？

```
count = 0
while True: #True本身就是真呀

    print("你是风儿我是沙,缠缠绵绵到天涯...",count)
    count +=1
```


循环终止语句

如果在循环的过程中，因为某些原因，你不想继续循环了，怎么把它中止掉呢？这就用到break 或 continue 语句

- break用于完全结束一个循环，跳出循环体执行循环后面的语句
- continue和break有点类似，区别在于continue只是终止本次循环，接着还执行后面的循环，break则完全终止循环

```
count = 0
while count <= 100 : #只要count<=100就不断执行下面的代码
    print("loop ", count)
    if count == 5:
        break
    count +=1 #每执行一次，就把count+1，要不然就变成死循环啦，因

print("-----out of while loop -----")
```

输出

```
loop 0
loop 1
loop 2
loop 3
loop 4
loop 5
-----out of while loop -----
```

continue 语句

```
count = 0
while count <= 100 :
    count += 1
    if count > 5 and count < 95: #只要count在6-94之间，就不走下面的print
        continue
    print("loop ", count)

print("-----out of while loop -----")
```

输出

```
loop 1
loop 2
loop 3
loop 4
loop 5
loop 95
loop 96
loop 97
loop 98
loop 99
loop 100
loop 101
-----out of while loop -----
```

如果在循环的过程中，因为某些原因，你不想继续循环了，怎么把它中止掉呢？这就用到break 或 continue 语句

- break用于完全结束一个循环，跳出循环体执行循环后面的语句
- continue和break有点类似，区别在于continue只是终止本次循环，接着还执行后面的循环，break则完全终止循环

练习：

- 优化猜年龄游戏，允许用户最多猜3次，中间猜对了，直接跳出循环
- 优化猜年龄游戏，允许用户最多猜3次，猜了3次后，再问是否还想玩，如果用户选y, 刚再允许猜3次，以次往复....



while ... else 玩法

与其它语言else 一般只与if 搭配不同，在Python 中还有个while ...else 语句

while 后面的else 作用是指，当while 循环正常执行完，中间没有被break 中止的话，就会执行else后面的语句

```
count = 0
while count <= 5 :
    count += 1
    print("Loop",count)

else:
    print("循环正常执行完啦")
print("-----out of while loop -----")
```

输出

```
Loop 1
Loop 2
Loop 3
Loop 4
Loop 5
Loop 6
循环正常执行完啦
-----out of while loop -----
```

开发工具IDE

代码自动补全


语法错误提醒

代码调试

性能测试


web开发框架支持

git/svn支持



Version: 2017.2.2
Build: 172.3757.67
Released: August 24, 2017

[System requirements](#)
[Installation Instructions](#)
[Previous versions](#)



Use the ISUPPORTDJANGO promo code when [buying new PyCharm Professional Edition annual subscription](#) to get 30% discount. All money goes to the Django Software Foundation to drive the future of Django.

Download PyCharm

[Windows](#) [macOS](#) [Linux](#)

Professional

Full-featured IDE for Python & Web development

[DOWNLOAD](#)

Free trial

Community

Lightweight IDE for Python & Scientific development

[DOWNLOAD](#)

Free, open-source

