计算机导论

第三章 Python语言及应用

引子

- 探索黑匣子 —— 从一个程序谈起
  - 普通的计算机使用者
  - 计算机专业学生
- 计算机系统的层次
  - 硬件：电子器件，支撑
  - 软件：使用者的创造性，智能
    (操作系统)

本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- 列举高级语言
- Python

本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- 列举高级语言
- Python

计算机编程的基本概念

什么是程序?
程序能干什么?

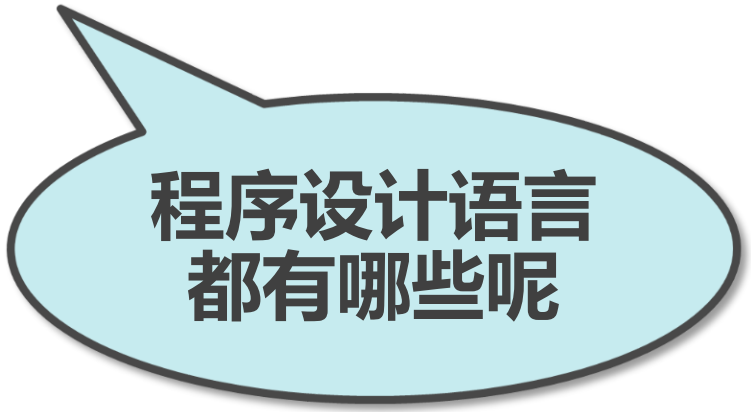
计算机程序

计算机语言


BANK
3. 将存折和取款单递给银行职员
5. 拿到钱并离开银行
4. 银行职员办理取款事宜
1. 带上存折去银行
$
2. 填写取款单并到相应窗口排队

计算机编程基本概念

- 程序（program）：为解决某一问题而设计的一系列指令，能被计算机识别和执行。
- 程序设计语言：用于书写计算机程序的语言。人与计算机打交道时交流信息的一类媒介和工具，由语句（statement）组成。



计算机编程基本概念

- 计算机程序设计语言经过一个从低级到高级的发展过程。
- 语言处理程序
  - 发展过程经历了机器语言、汇编语言和高级语言三个层次。

计算机编程基本概念

- 机器语言：（machine language）计算机直接使用的二进制形式的程序语言或机器代码。
- 汇编语言：（assembler language）一种面向机器的用符号表示的低级程序设计语言。相当于机器指令的助记符号，与机器语言很接近。
- 高级语言：（high - level language）是易为人们所理解的完全符号化的程序设计语言。

机器语言

- 机器语言
  - 以二进制代码表示指令集合、CPU直接能识别和执行的语言。
  - 优点是占用内存少、执行速度快；缺点是不易阅读和记忆、编程查错困难等。

```
0000,0000,0000000001000
0000,0000,0000000000001
0000,0001,0000000001000
```

汇编语言

- 汇编语言
  - 用助记符表示机器指令中操作码和操作地址的语言
  - 汇编语言也是面向机器的语言，与机器语言相比较为直观、易理解和记忆，但通用性不强。
  - 常用的汇编语言有80X86汇编、80C51汇编、ARM汇编等。

```
MOV AX,100H
AND AX,0FFH
```

高级语言

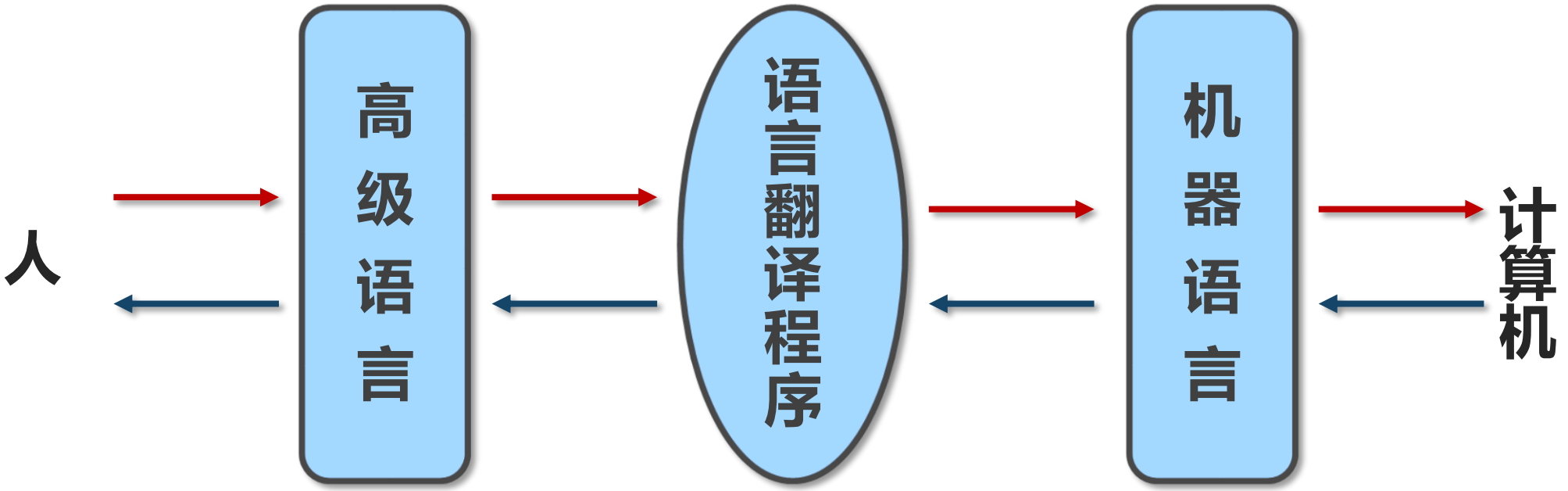
- 高级语言
  - 接近人们使用的自然语言，一条语句不仅仅是完成单一的机器指令操作，也可能是多项操作。
  - 常用的高级语言有C、C++、Java等。

```
int main(void){
    int first=2, second=3, sum;
    sum = first + second;
    printf("sum = %d", sum);
    return 0;
}
```

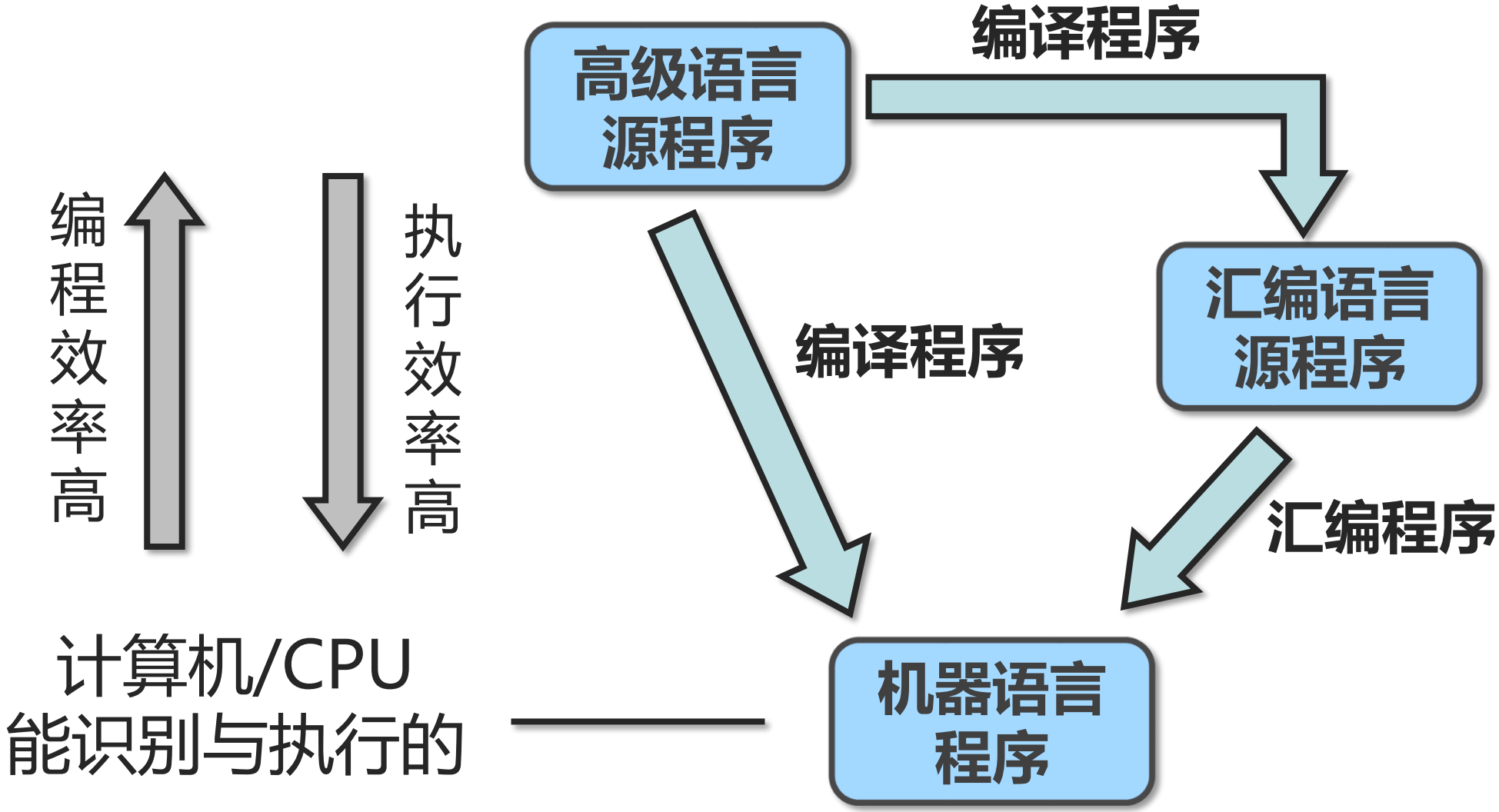
本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- 列举高级语言
- Python

语言的层次



语言的层次

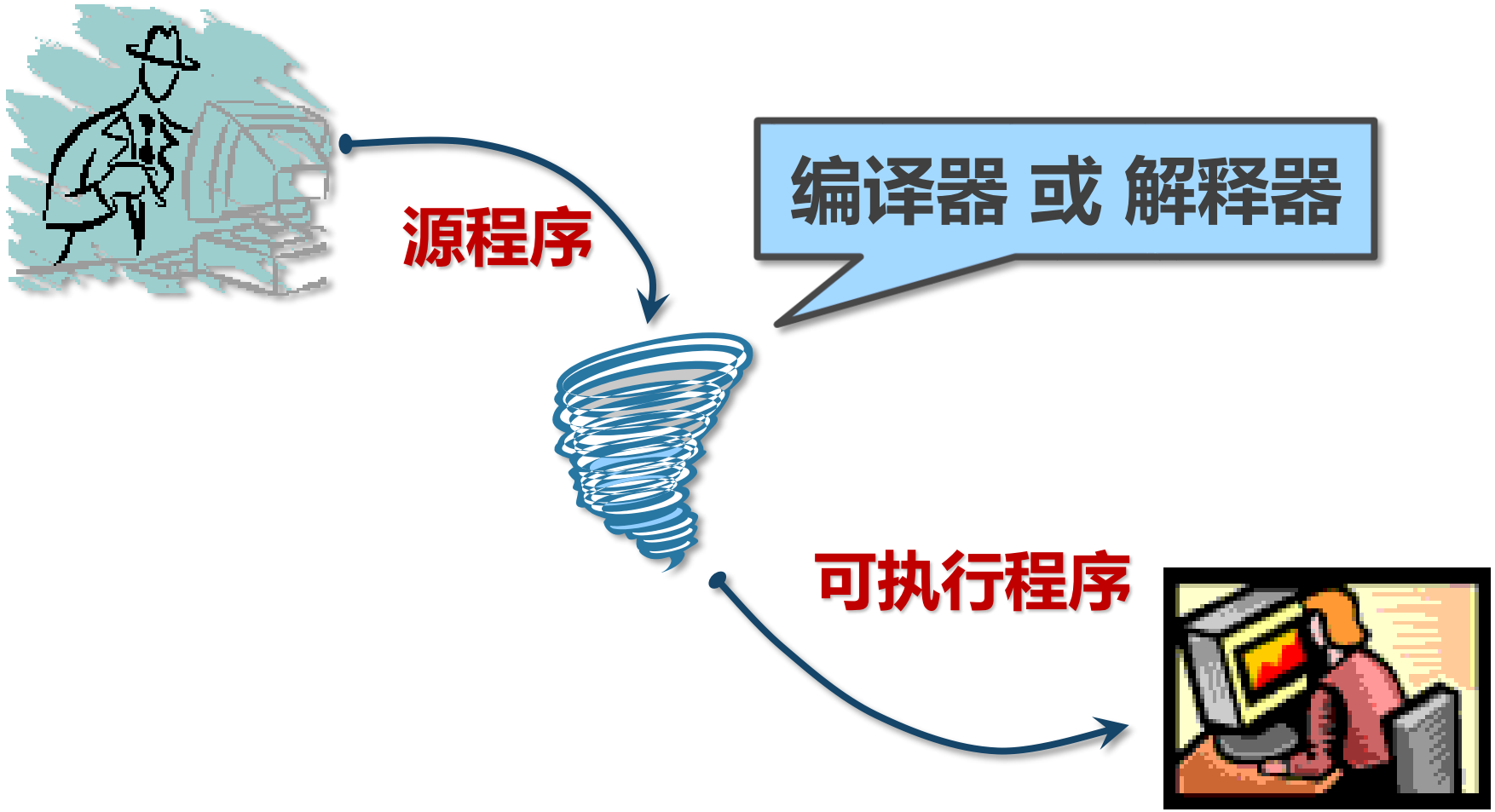

编程效率高
执行效率高
高级语言源程序
编译程序
汇编语言源程序
编译程序
汇编程序
机器语言程序
计算机/CPU能识别与执行的

编译系统

- 由于计算机只能识别和执行机器语言，高级语言编写的程序仍然不能直接被计算机识别，必须经过 "翻译" 才能被执行。
- 这种翻译方式有两种：编译、解释。
- 负责这种翻译工作的程序称语言处理程序：编译程序、解释程序，它们均是系统程序。

计算机软件

- 源程序
- 编译、解释
- 程序的执行



编译类语言

- 编译类语言：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，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才能执行，只有目标文件而没有源代码，修改很不方便。

解释类语言

- 解释类语言：执行方式类似于日常生活中的 "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，但这种方式比较灵活，可以动态地调整、修改应用程序。

本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- Python
- 列举高级语言

高级语言的特点

- 高度封装，与低级语言相对
- 使用人易于接受的文字来表示
- 不是特指的某一种语言，而是包括很多种
- 与计算机的硬件和指令系统无关
- 执行速度慢

高级语言的语法

- 不同的高级语言有不同的编写格式和语句分割符号，计算机按照语句分割符号识别每一条语句。
- 语言的语法是指这样一组规则，用它可产生一个程序。
  - 词法规则
  - 语法规则

词法规则

- 字母表就是一个有穷字符集

  Σ = { a–z 、 A–Z 、 0–9 、 (、)、 [、]、 .、 !、 ~、
  +、 -、 *、 /、 &、 %、 <、 >、 =、 ^、 |、 ?、 ,、 ; }

- 词法规则是指单词符号的形成规则

  –字母、下划线打头的字母、数字和下划线构成的符号串。
  如：a1、ave 、_day

语法规则

- 语法规则规定了如何从单词符号形成更大的结构，换言之，语法规则是语法单位的形成规则。
- 最常用的三种语句：
  - 表达式语句
  - 函数调用语句
  - 控制语句

高级语言的数据类型

- 一个值的集合以及定义在这个值集上的一组操作
- 数据类型
  - 分得多大空间
  - 表示多大范围
  - 能做何种运算
- 例如
  - 整型 数值100，变量x
  - 布尔型 True或者False …

高级语言的表达式语句

- 表达式语句由表达式组成。
  - 表达式：操作数 + 运算符
- 表达式形成规则：
  - 表达式由数字、运算符、数字分组符号（括号）、变量等组成的有意义的序列，并且能够求得数值。
  - 执行表达式语句就是计算表达式的值。

```
y+z
```

不完整表达式

```
x=y+(z+3)
```

完整表达式

表达式例子

- 算术表达式

```
1+2
(1*2)+3
```

- 布尔表达式

```
a>=b
1==2
!(1<=2)
(1==1) or (1==2)
(1==1) and (1==2)
```

函数调用语句

- 函数调用语句由函数名和函数的实际参数所组成。
- 例如:

　　已有函数 add(y, z)，功能是两个参数求和函数调用语句： x=add(y, z) 。

控制结构

- 顺序语句：

```
a = 3
a = a + 1
```

- 分支语句：

```
if(B<=C)
    语句1
else
    语句2
```

- 循环语句：

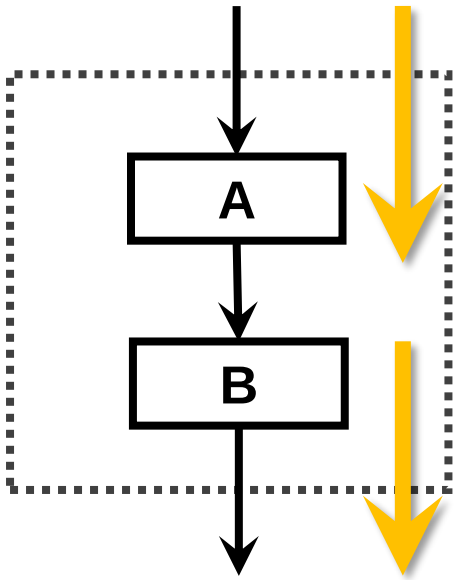
```
while(B<=C)
    语句1
```

```
for(int i=0; i<10; i++)
    语句1
```

```
for i in range (0, 10)
    语句2
```

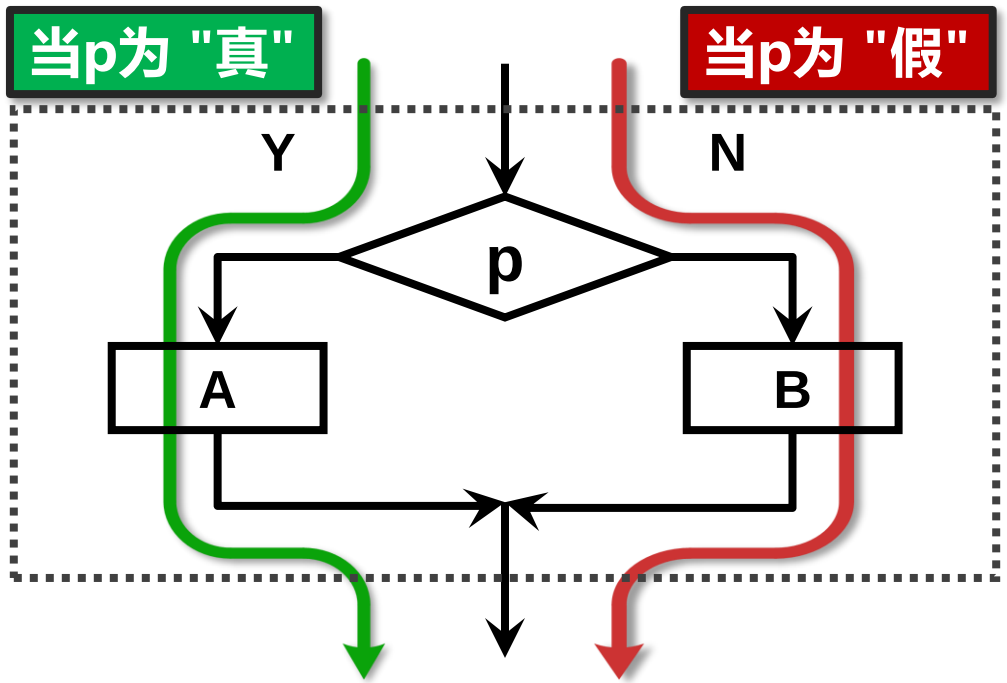
顺序结构

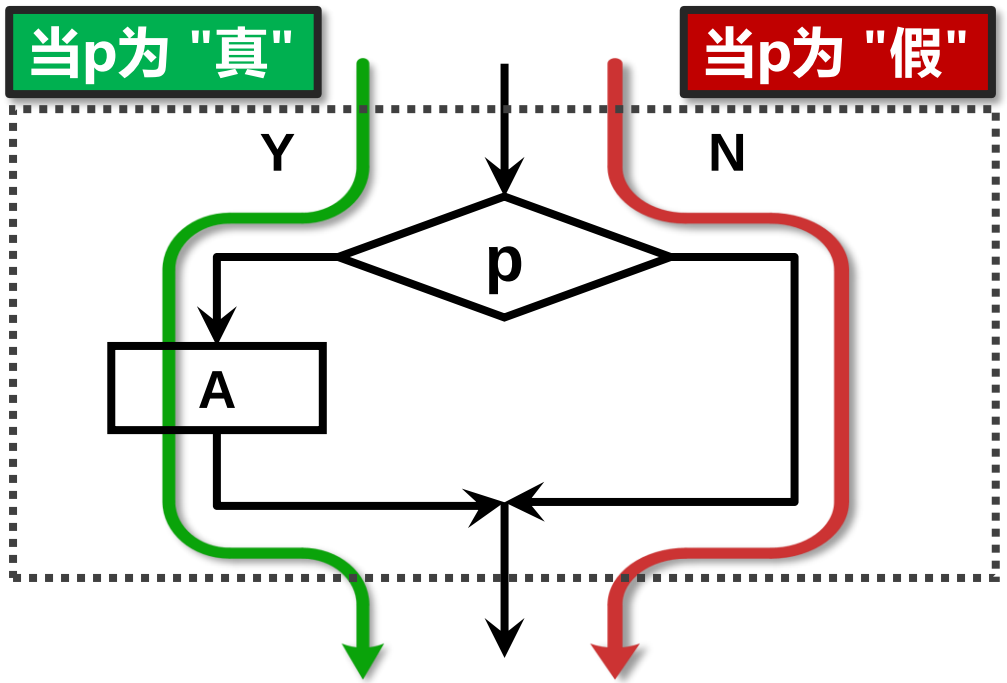
- 顺序结构是最简单的程序结构，也是最常用的程序结构，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。



选择结构

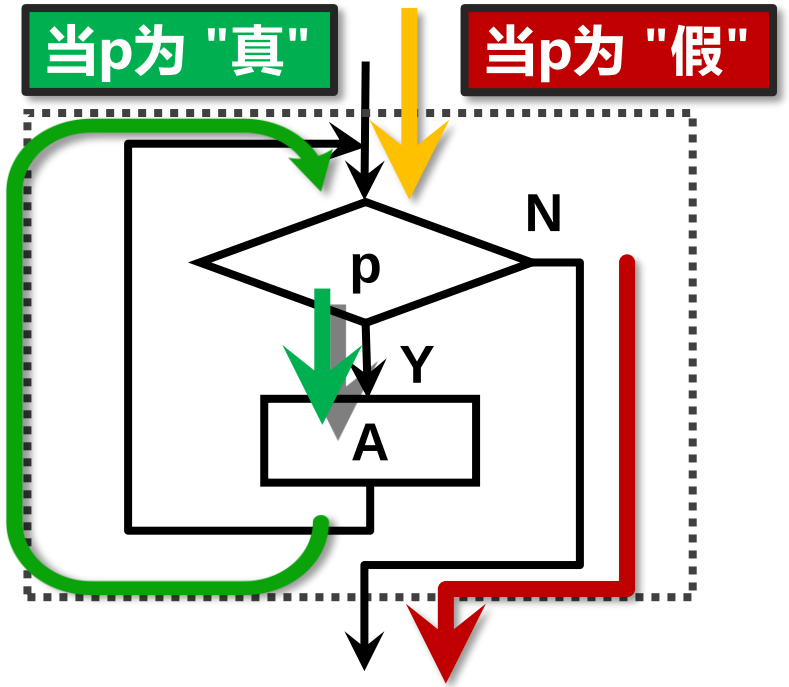
- 选择程序结构用于判断给定的条件，根据判断的结果判断某些条件，根据判断的结果来控制程序的流程。

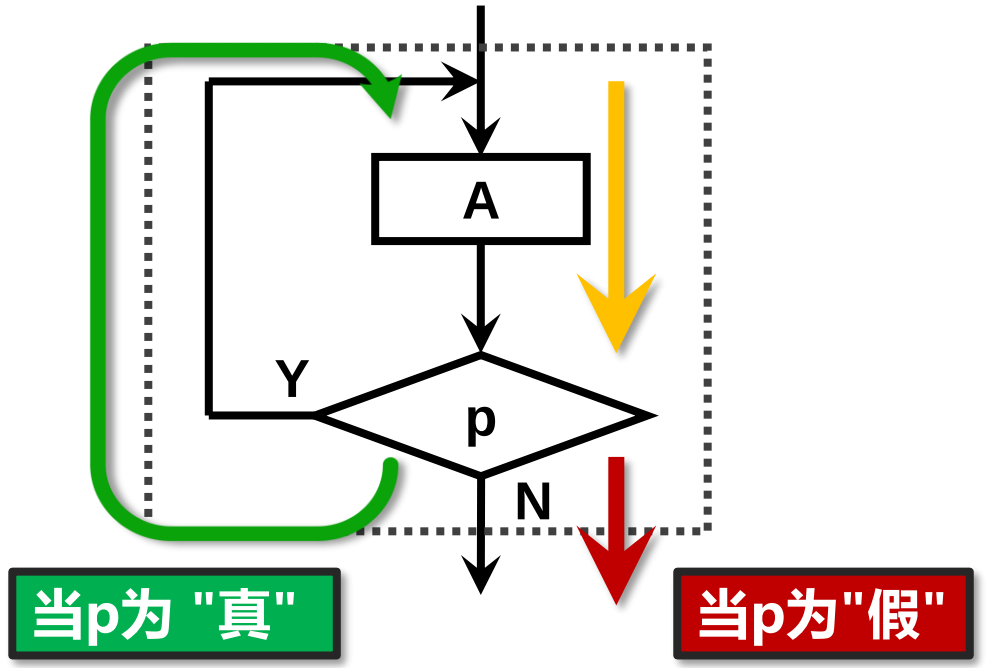




循环结构

- 循环结构可以减少源程序重复书写的工作量，用来描述重复执行某段算法的问题，这是程序设计中最能发挥计算机特长的程序结构 。





本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- 列举高级语言
- Python

常用的程序设计语言

- 目前有各种高级程序设计语言，其中以下几种应用非常广泛。

FORTRAN	PASCAL
C语言	C++
Java	Python
C#	...

高级语言时代（1954—1995）

- 随着世界上第一个高级语言FORTRAN的出现，新的编程语言开始不断涌现出来。各有特色，各有优势，随着时间的检验，一些流行至今，一些则逐渐消失。
- 1957年世界上第一个高级语言FORTRAN 开发成功。
- FORTRAN取的是FORmula TRANslator两个单词前几个字母拼成的，意思是公式翻译语言 。

被遗忘的PASCAL

- 1967年Niklaus Wirth开始开发PASCAL语言，1971年完成。
- 主要特点有：严格的结构化形式；丰富完备的数据类型；运行效率高；查错能力强，可以被方便地用于描述各种算法与数据结构有益于培养良好的程序设计风格和习惯。
- PASCAL是一个重要的里程碑结构化程序设计概念的语言。

C语言

- 语言简洁紧凑、使用灵活方便。
- 运算符丰富。
- 数据类型丰富。
- C是结构式语言。
- 语法限制不太严格、程序设计自由度大。
- 允许直接访问物理地址，可直接对硬件进行操作。
- 程序执行效率高。
- 适用范围大，可移植性好。

C++的特点

- 保持了与C语言的兼容性。
- 支持面向过程的程序设计。
- 具有程序效率高、灵活性强的特点。
- 具有通用性和可移植性。
- 具有丰富的数据类型和运算符，并提供了强大的库函数。
- 具有面向对象的特性，C++支持抽象性、封装性、继承性和多态性。

C#

- C# 充分借鉴了 C 和 Java 的语言，甚至照搬了 C 的部分语法几乎集中了所有关于软件开发和软件工程研究的最新成果。面向对象、类型安全、组件技术、自动内存管理、跨平台异常处理、版本控制、代码安全管理 …
- C# 程序需要 .NET 运行库作为基础 。

Java

- Java语言是一个完全面向对象的语言，并且对软件工程技术有很强的支持。
- Java语言的设计集中于对象及其接口，它提供了简单的类机制以及动态的接口模型。
- 对象中封装了它的状态变量以及相应的方法，实现了模块化和信息隐藏。
- 类提供了一类对象的原型，并且通过继承机制，子类可以使用父类所提供的方法，实现了代码的复用。

总结

- Java霸占了企业级应用市场，一部分移动开发（J2ME）和Web开发。—— 使用者排名第一
- C和C++是嵌入式开发和系统给开发的利器。操作系统、驱动程序、各种游戏大都是他们的开发的——地位不可替代
- ……
- 其他：Ruby，JSP，JavaScript，PHP 等等也占据了一定的市场。

TIOBE编程语言排行榜

Aug 2018	Aug 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.881%	+3.92%
2	2		C	14.966%	+8.49%
3	3		C++	7.471%	+1.92%
4	5	↑	Python	6.992%	+3.30%
5	6	↑	Visual Basic .NET	4.762%	+2.19%
6	4	↓	C#	3.541%	-0.65%
7	7		PHP	2.925%	+0.63%
8	8		JavaScript	2.411%	+0.31%
9	-	⇈	SQL	2.316%	+2.32%
10	14	⇈	Assembly language	1.409%	-0.40%
11	11		Swift	1.384%	-0.44%
12	12		Delphi/Object Pascal	1.372%	-0.45%
13	17	⇈	MATLAB	1.366%	-0.25%
14	18	⇈	Objective-C	1.358%	-0.15%
15	10	⇊	Ruby	1.182%	-0.78%

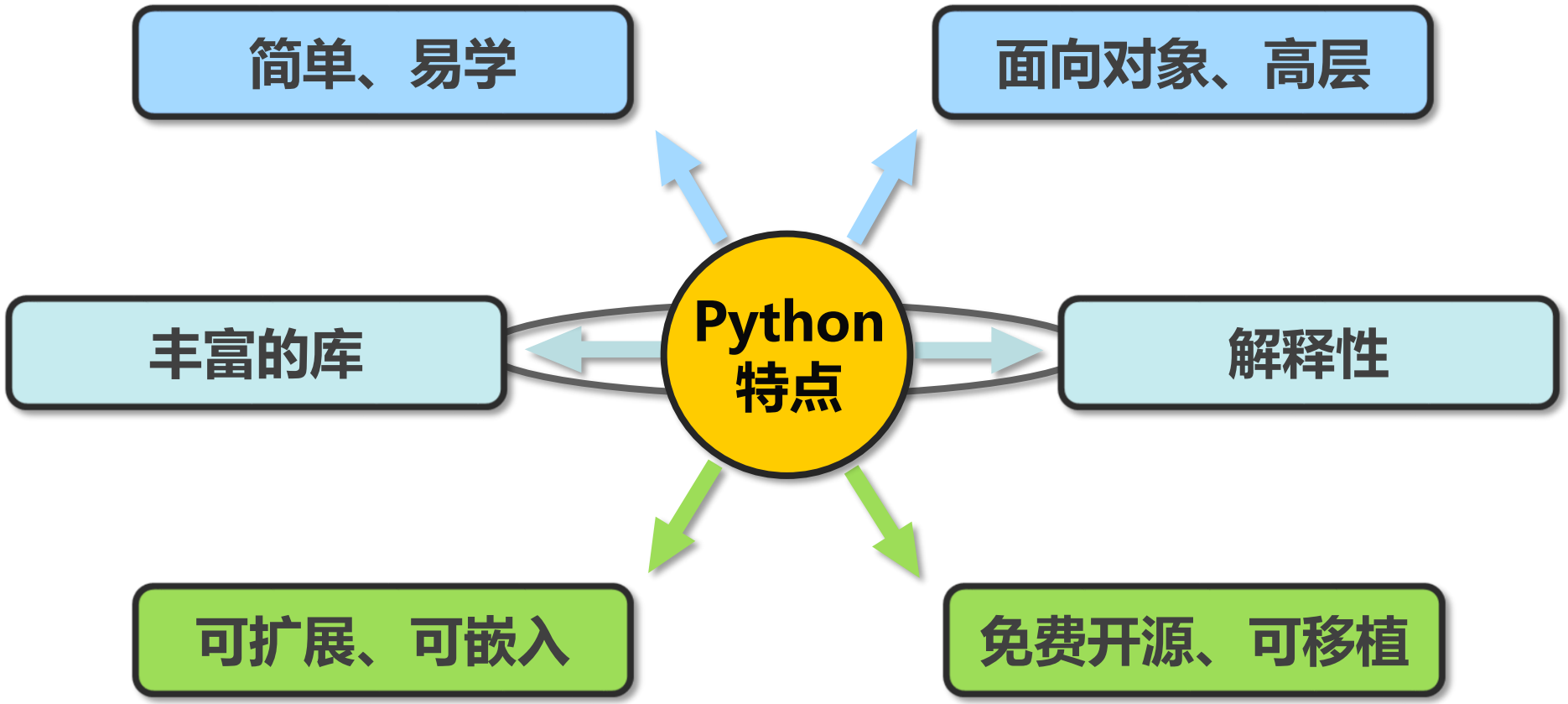
https://www.tiobe.com/tiobe-index/

本章内容

- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- 列举高级语言
- Python

- Python
  - Python简介
  - Python的内置数据类型
  - Python赋值语句
  - Python控制结构
  - Python函数调用

为什么要学Python



Python发展

- Python发展
  - 1991年，第一个Python编译器诞生

```
Python 1.0 1994/01
Python 2.0 2000/10
Python 2.4 2004/11
Python 2.5 2006/09
Python 2.6 2008/10
Python 2.7 2010/07
```

```
Python 3.1 2009/06
Python 3.2 2011/02
Python 3.3 2012/09
Python 3.4 2014/03
Python 3.5 2015/09
Python 3.6 2016/12
Python 3.7 2018/06
```

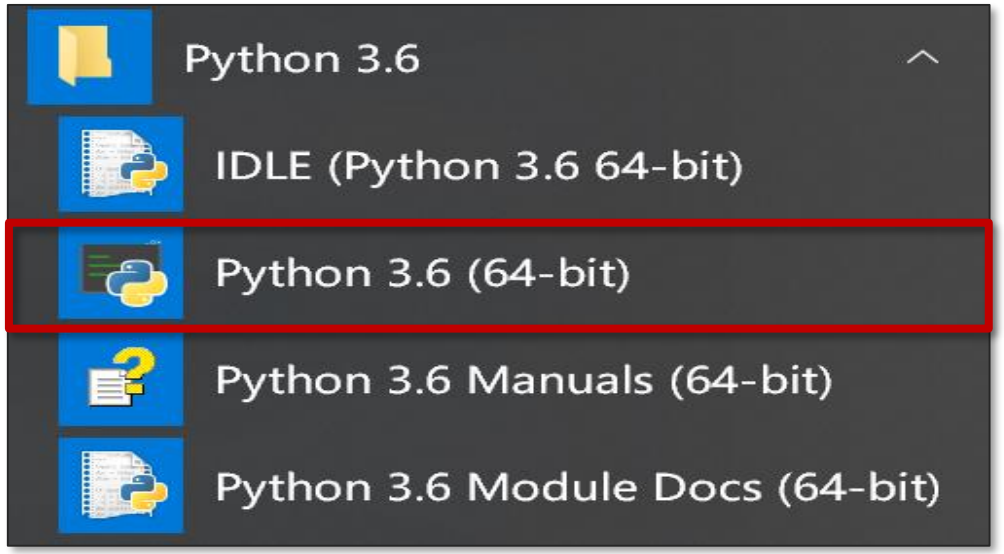
吉多·范罗苏姆
Guido van Rossum

  - Python2.7将在2020年停止支持，并且不会在发布2.8版本

Windows中使用Python

- Windows中要使用Python进行程序开发，必须先安装Python运行环境。
- 官网地址：https://www.python.org/

小例子：照猫画虎




```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

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

```
Hello World!
```

```
x=2
y=1
print(x*x+y*y)
```

```
5
```

```
def F(x, y):
    return(x*x+y*y)
F(2,1)
```

```
5
```

小游戏：结构分解

函数定义

```
import random
def caishu():
    i = 0
    key = random.randint(1,10)
    while i < 5:
        guss = int(input("enter:"))
        if key == guss:
            print("good guess!")
            break
        elif guss > key:
            print("guss>ken try again")
        else: print("guss<key try again")
            i += 1
    else:
        print("game over")
        print("The key is:", key)
caishu()
```

循环结构

选择结构

函数调用

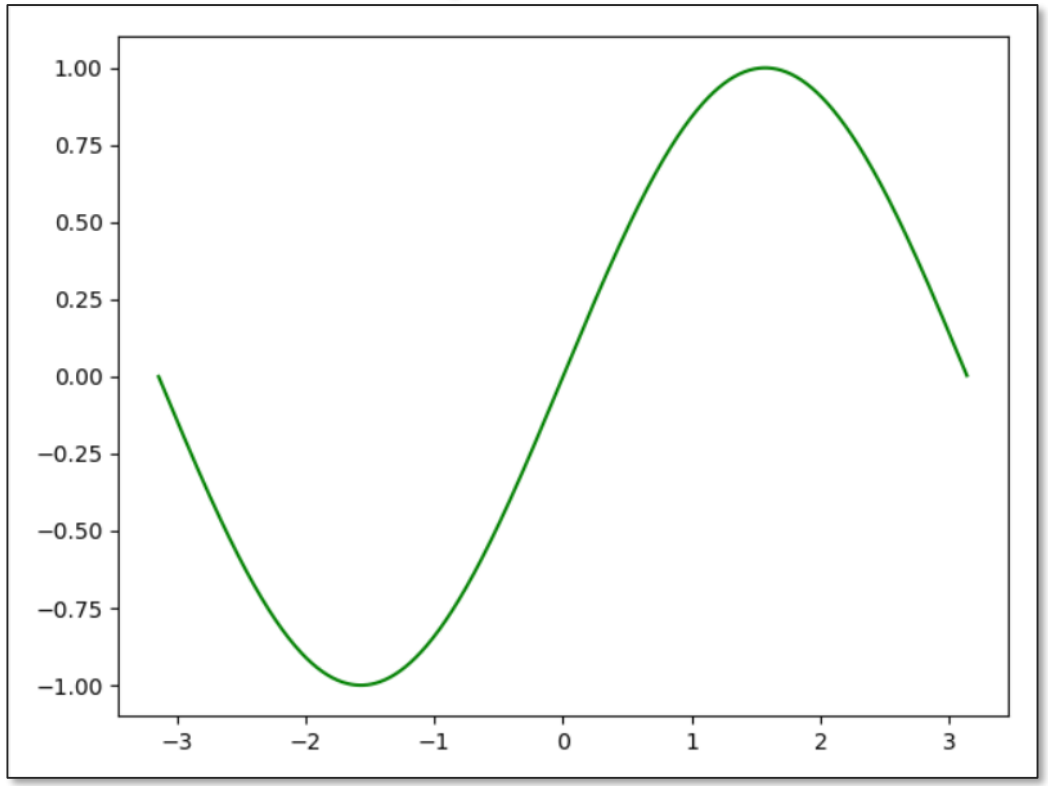
中学知识再现

引入库

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi, np.pi, 0.01)
y = np.sin(x)
plt.plot(x, y, 'g')
plt.show()
```

函数调用

运行结果



初识Python

- Python初体验
  - 数据类型
    - 整型类型、浮点类型、布尔类型…
  - 表达式
    - 算术表达式、逻辑表达式
  - 三种控制语句
    - 顺序结构
    - 循环结构for和while
    - 分支结构if

```
a = 5
a = a + 1
print(a)
```

```
b = 100 < 101
print(b)
```

```
for i in range(1, 5):
    print(i)
```

```
i = 1
while i < 5:
    print(i)
    i = i + 1
```

```
i = 10
j = 11
if i < j:
    print("i<j")
else:
    print("i>=j")
```

- Python
  - Python简介
  - Python数据类型
  - Python赋值语句
  - Python控制结构
  - Python函数调用

- Python
  - Python数据类型
    - 数值类
    - 布尔类型
    - 列表
    - 字符串类型
    - 字典

整数类型

- 通常整数表示的范围-2 147 483 648到2 147 483 647
- Python3.0以后的版本无范围限制

  例如：0, 100, -100

  012 (八进制的10)

  0x14 (十六进制的20)

- 操作符：+，-，*，%，（），/，//，**

浮点类型

- 浮点型：如 5.0，1.6，200.985 等有小数部分的数值
- 操作符：跟整型类似 +，-，*，（），/，//，**

生成随机数

- 在Python中要产生随机数，首先要引入random模块。
- 产生10-20的随机浮点数

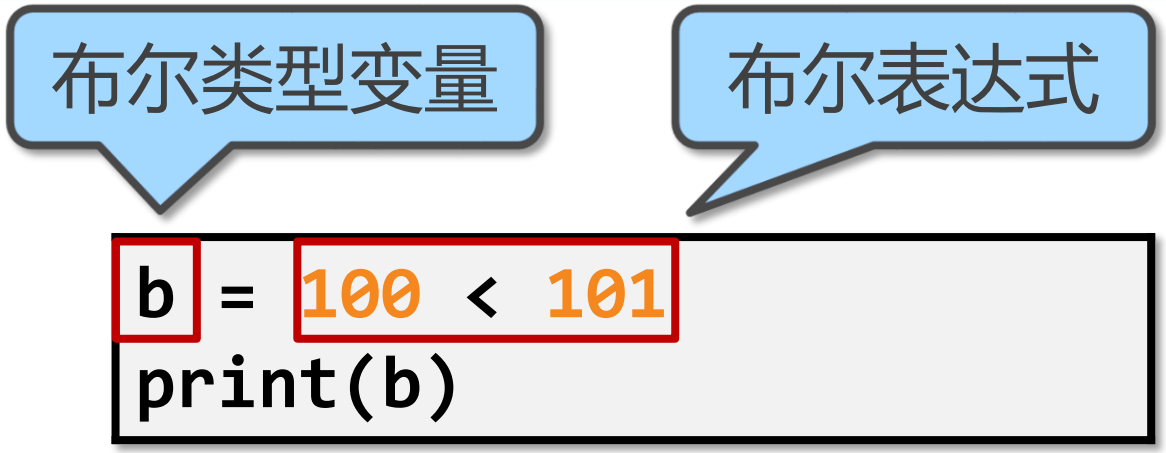
```
import random
f = random.uniform(10, 20)
print(f)
```

- 产生10-20的随机整数

```
import random
f = random.randint(10, 20)
print(f)
```

- Python
  - Python数据类型
    - 数值类
    - 布尔类型
    - 列表
    - 字符串类型
    - 字典

布尔类型



- 布尔型变量有两种可能的值：True False
- 布尔比较运算：<，>，<=，>=，==，!=
- 布尔逻辑运算：not，and，or

- Python
  - Python数据类型
    - 数值类
    - 布尔类型
    - 列表
    - 字符串类型
    - 字典

序列

- 有顺序编号的结构称为 "序列"
  - 列表
  - 元组
  - 字符串

不可修改的列表

- 列表的声明形式
  - L = [ ] (空列表)
  - L = [1, 3, 5]
- 列表中的元素以 "," 相隔

列表（list）

- 列表中的元素类型可以是不一样的

```
L = [1, 1.3, "2", "China", ["I", "am", "another", "list"]]
```

- 优点：给编程者带来许多便利
- 注意：对列表元素进行操作时需注意元素类型

序列的通用操作

- 索引
  - 序列中所有元素都有索引号
  - 索引号为正数时，从0开始递增
  - 索引号为负数时，表示从序列的最后一个元素开始计数

```
L = [1, 1.3, "2", "China", ["I", "am", "another", "list"]]
print(L[1])
```

```
1.3
```

序列的通用操作

- 分片
  - L[index1:index2]
  - L[index1:]
  - L[:index2]
  - L[:]
  - L[index1:index2:stride]
  - 步长：默认为1
  - 可以大于1，可以取负数，但不能为0

分片结果中第一个元素的索引号

index2 - 1：最后一个元素的索引号

步长

序列的通用操作

- 加：序列相加表示连接

进行连接的两个序列
必须具有相同的类型

```
L1 = [5, 1.3, "2"]
L2 = ["China", ["I", "am", "another", "list"]]
L = L1 + L2
print(L)
```

```
[1, 1.3, '2', 'China', ['I', 'am', 'another', 'list']]
```

- 乘：序列重复多次

```
L = ["I", "love", "Python"]
print(L*2)
```

```
['I', 'love', 'Python', 'I', 'love', 'Python']
```

列表方法

- 列表方法：（当 s = [1, 2] 时）

	函数	作用	参数	结果
1	s.append(x)	将一个数据添加到列表s的末尾	'3'	[1, 2, '3']
2	s.clear()	删除列表s的所有元素	无	[ ]
3	s.copy()	返回与s内容一样的列表	无	[1,2]
4	s.count(x)	统计x元素在列表中出现的次数	2	1
5	s.extend(t)	将列表t添加到列表s的末尾	['3', '4']	[1, 2, '3', '4']

列表方法

- 列表方法：（当 s = [1, 2] 时）

	函数	作用	参数	结果
6	s.insert(i, x)	将数据x插入到s的第i号位置	0, '3'	['3', 1, 2]
7	s.pop(i)	将列表s第i号元素弹出并返回其值	0 或 无	1 或 2
8	s.remove(x)	删除列表s中第一个值为x的元素	1	[2]
9	s.reverse()	反转s中的所有元素	无	[2, 1]

内建函数

- Python 提供了求长度、最大值和最小值的内建函数，分别为 len( )、max( )、min( ) 。

```
nums = [300, 200, 100, 500, 400]
# 数列长度
print(len(nums))
# 数列最大值
print(max(nums))
# 数列最小值
print(min(nums))
```

```
5
500
100
```

元组（tuple）

- 元组与数列类似，不同之处在于元组的元素不能修改
- 元组的创建：使用逗号分隔的一些值，会自动创建为元组

通常会使用括号将值括起来

```
nums = (1, 3, 2, 4)
print(nums)
```

```
(1, 3, 2, 4)
```

- Python
  - Python数据类型
    - 数值类
    - 布尔类型
    - 列表
    - 字符串类型
    - 字典

字符串类型

- 字符串表示方式
  - 单引号 例: 'hello world!'
  - 双引号 例: "hello world!"
- 注意:
  - `''`或`""`本身只是一种表示方式，不是字符串的一部分
  - 使用`''`表示字符串时，其中可以直接出现`"`
  - 使用`""`表示字符串时，其中可以直接出现`'`
  - 字符串内既包含`'`又包含`"`时，可使用`\`进行转义

字符串类型的部分应用

- 字符串主要用在输入和输出，input( ) 函数为输入函数。
  - Python 的 input( ) 函数返回字符串类型
- 字符串类型与数值型相互转化
  - 数值型转化成字符串类型 str( )
  - 字符串类型转化成数值型 int( ) float( )

```
s = input("Enter:")
s = s + 1
print(s)
```

```
TypeError: must be str, not int
```

键盘输入6
程序报错

```
s = int(input("Enter:"))
s = s + 1
print(s)
```

```
7
```

键盘输入6
结果输出7

字符串的格式化

"亲爱的 xx 你好！你 xx 月的话费是 xx，余额 xx 元。"

- 在开发时经常遇到如上字符串，其中 xx 部分内容是根据变量变化的，Python中提供了一种使用 % 实现的格式化字符串方法。

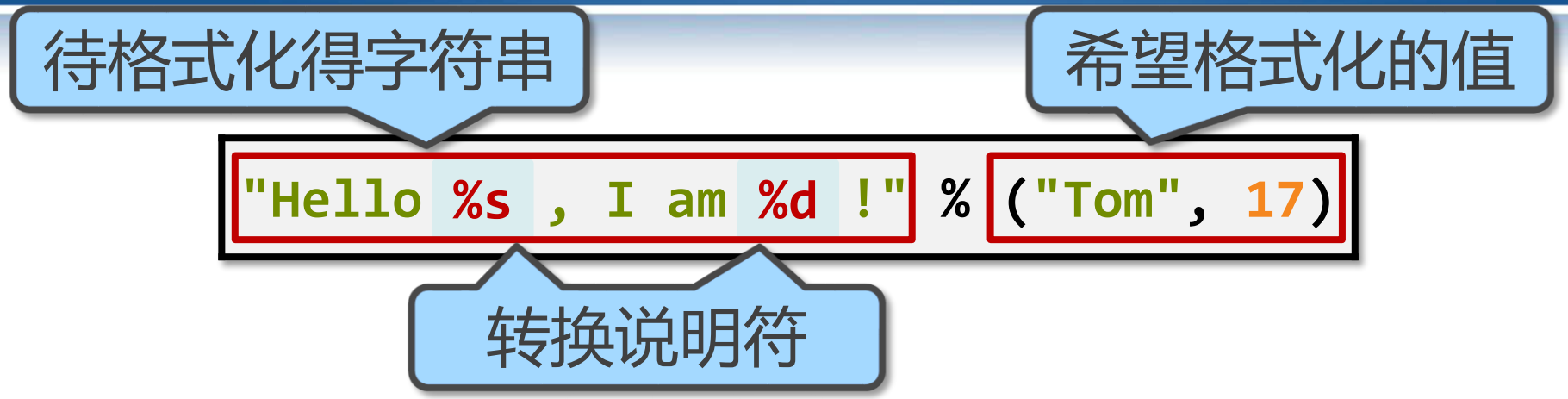
```
print("My name is %s!" % "Tom")
```

```
My name is Tom!
```

```
print("Hello %s, I am %d!" % ("Tom", 17))
```

```
Hello Tom, I am 17!
```

字符串的格式化



- 常见转换说明符
  - %s ： 字符串
  - %d ： 整数
  - %f ： 浮点数
- 注意:
  - 不确定用什么时，`%s` 永远起作用，会把任何类型转换为字符串
  - 字符串中有`%`字符时，用`%%`转义来表示`%`。

字符串专有方法

- 字符串专有方法：（当 s = "Just do IT" 时）

	函数	作用	参数	结果
1	s.find(str)	字符串中是否包含子字符串，并返回位置索引	"do"	5
2	s.strip(c)	移除字符串头尾指定字符	'T'	"Just do I"
3	s.lower()	将字符串的大写字母转小写字母	无	"just do it"
4	s.upper()	将字符串的小写字母转大写字母	无	"JUST DO IT"
5	s.swapcase()	将字符串大小写字母进行转换	无	"jUST DO it"

字符串专有方法

- 字符串专有方法：（当 s = "Just do IT" 时）

	函数	作用	参数	结果
6	s.replace(old, new)	将字符串中的旧字符串替换为新字符串	"IT","it"	"Just do it"
7	s.split(st)	将字符串根据分隔符st拆分成数列	" "	["Just", "do", "IT"]

- 字符串专有方法：（当 s = "+" 时）

	函数	作用	参数	结果
8	s.join(sq)	使用字符串将序列中的元素连接成一个新字符串	["1", "2", "3"]	"1+2+3"

- Python
  - Python的内置数据类型
    - 数值类
    - 布尔类型
    - 列表
    - 字符串类型
    - 字典

字典（dict）

- Python内置了字典：dict的支持，dict全称dictionary，在其它语言中也称为map，使用 键-值（key-value）存储，具有极快的查找速度

"键"与"值"使用：隔开

每组"键-值"使用，隔开

整个字典使用使用 {} 括起来

```
scores = { "Zhangsan": 95 , "Lisi": 75, "Wangwu": 85 }
print(scores["Zhangsan"])
scores["Zhangsan"] = 96
print(scores["Zhangsan"])
```

直接使用"键"获取对应值

```
95
96
```

字典基本操作

"键"不存在时完成添加操作

```
scores = {"Zhangsan": 95, "Lisi": 75}
print(scores)
scores["Zhangsan"] = 96
print(scores)
scores["Zhaoliu"] = 69
print(scores)
scores.pop("Lisi")
print(scores)
```

"键"已存在完成修改操作

删除字典中的"键-值"项

```
{'Zhangsan': 95, 'Lisi': 75}
{'Zhangsan': 96, 'Lisi': 75}
{'Zhangsan': 96, 'Lisi': 75, 'Zhaoliu': 69}
{'Zhangsan': 96, 'Zhaoliu': 69}
```

- Python
  - Python简介
  - Python的内置数据类型
  - Python赋值语句
  - Python控制结构
  - Python函数调用

基本赋值语句

- Python 中创建一个变量不需要声明其类型
- 基本赋值语句： 变量=值

```
x = 1
y = 2
k = x + y
print(k)
```

```
3
```

序列赋值

- 序列赋值形式

  左侧的一系列变量 = 右侧的一系列值

右侧值依次赋给左侧变量

```
a, b = 4, 5
print(b)
```

```
5
```

- Python
  - Python简介
  - Python的内置数据类型
  - Python赋值语句
  - Python控制结构
  - Python函数调用

if语句

- if语句

```
if <test1>:
      <语句块1>
elif <test2>:
      <语句块2>
elif <test3>:
      <语句块3>
.
.
.
else:
      <语句块n>
```

首先，进行条件测试

与test1同层次的多个选择

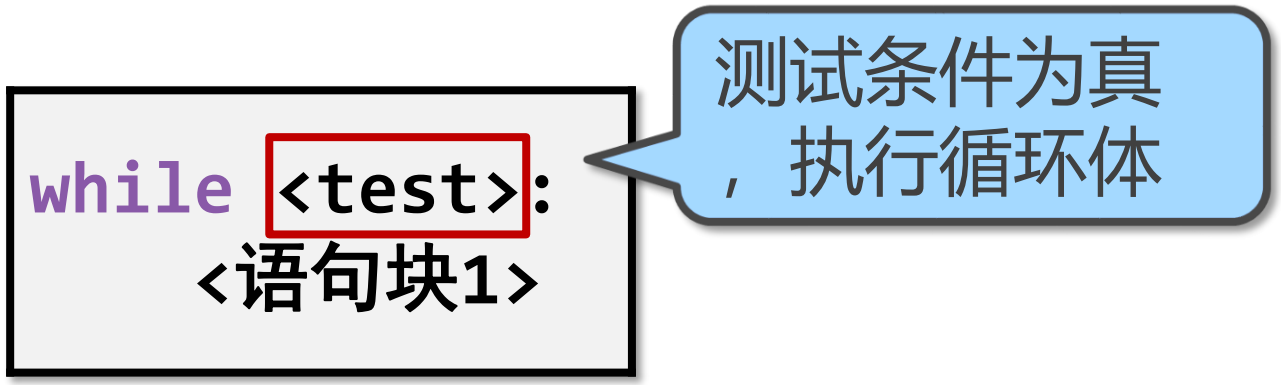
前面的测试均为假，执行else

例子：if

```
def if_test(score):
    if score >= 90:
        print("Excellent")
    elif score >= 80:
        print("Very Good")
    elif score >= 70:
        print("Good")
    elif score >= 60:
        print("Pass")
    else:
        print("Fail")
if_test(100)
```

While语句-通用格式

- while语句



While语句-continue

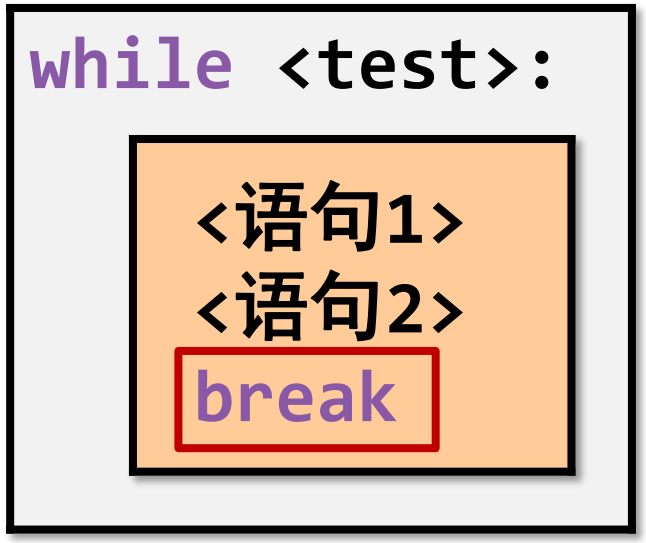
- while语句

```
while <test>:
    <语句1>
    <语句2>
    continue
```

- 注意：遇到continue，结束本次循环，重新开始下一轮循环

While语句-break

- while语句



- 注意：遇到break，结束整个循环，执行循环之后的语句

例子：continue和break

立即结束本次循环，
重新开始下一轮循环

```
x = 10
while x > 0:
    if x % 3 == 0:
        x = x-1
        continue
    print(2*x, end=" ")
    x = x-1
```

```
20 16 14 10 8 4 2
```

立即跳出循环结构，
执行while后面的语句

```
x = 10
while x > 0:
    if x % 6 == 0:
        break
    print(2*x, end=" ")
    x = x-1
```

```
20 18 16 14
```

for语句

- for语句

遍历序列

```
for <target> in <object>:
    <语句块1>
```

- 注意：object 中的每一个元素会依次赋给 target

- Python
  - Python简介
  - Python的内置数据类型
  - Python赋值语句
  - Python控制结构
  - Python函数调用

什么是函数

- 函数是一种程序构件，是构成大程序的小功能部件(子程序)
- 函数的好处
  - 编程更容易把握，大程序分解成小功能部件
  - 代码重用,避免重复相同/相似代码，提高开发效率，更易维护
  - 程序更可读,更易理解
  - 代码简洁美观

函数的定义和使用

- 先定义 (define)
- 再通过函数名调用 (call)
- 调用时传递参数
- 调用执行的是函数体 (语句序列)
- 调用产生返回值
- 函数定义可置于程序中任何地方，但必须在调用之前

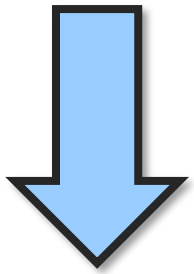
```
def func(x):
    y = x * x
    return y

a = func(2)
```

编程实例：生日歌

- 减少相同代码的重复

```
print("Happy birthday to you!")
print("Happy birthday to you!")
print("Happy birthday, dear Fred.")
print("Happy birthday to you!")
```

重复代码的缺点:
(1)费时费力
(2)代码一致性维护



```
def happy():
    print("Happy birthday to you!")

happy()
happy()
print("Happy birthday, dear Fred.")
happy()
```

函数调用过程

- 函数定义

```
def <函数名>(<形参列表>):
    <函数体>
```

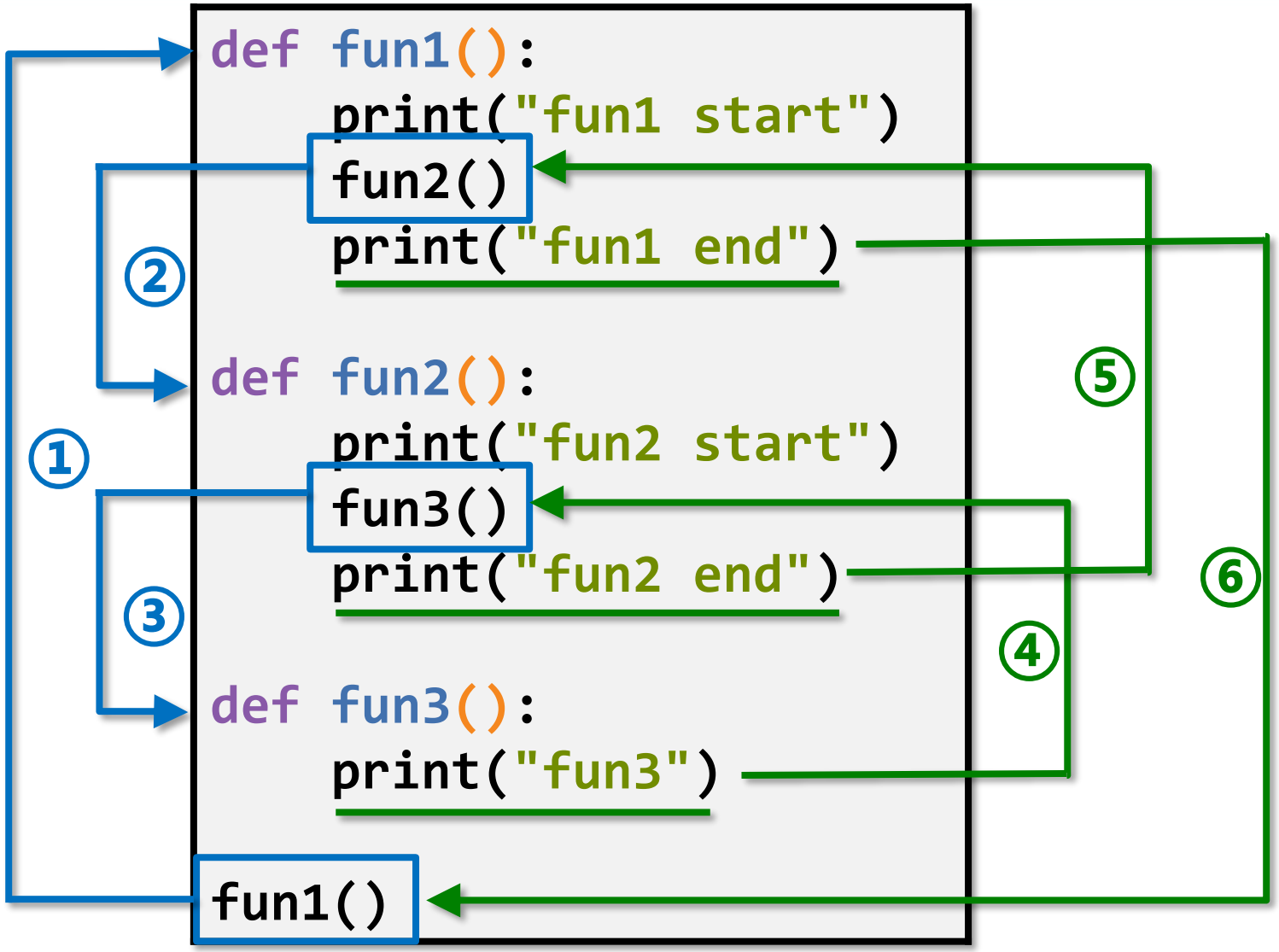
- 函数调用

```
<函数名>(<实参列表>)
```

  - 函数形参被赋值为实参
    - 按位置对应，或按名（形参=实参），即位置和个数均一一对应
    - 实参可以是字面值，也可以是已赋值的变量
  - 执行函数体
  - 控制返回调用者（调用点的下一条语句）

函数调用过程图解

- 调用过程?



带返回值的函数

- 函数与调用者之间的沟通：
  - 通过参数从调用者输入值
  - 通过返回值向调用者输出值
- 定义

```
def <函数名>(<形参列表>):
    ...
    return <表达式列表>
```

  - return计算各表达式，将结果返回调用者，退出函数
  - 没有return的函数其实也返回一个值：None

再见小游戏

函数定义

缩进体现逻辑

函数调用

字符串

循环结构

选择结构

```
import random
def caishu():
    i = 0
    key = random.randint(1,10)
    while i < 5:
        guss = int(input("enter:"))
        if key == guss:
            print("good guess!")
            break
        elif guss > key:
            print("guss>ken try again")
        else: print("guss<key try again")
            i += 1
    else:
        print("game over")
        print("The key is:", key)
caishu()
```

Python进阶学习


python™
About
Downloads
Documentation
Docs
Audio/Visual Talks
Beginner's Guide
Developer's Guide
FAQ
Non-English Docs
PEP Index
Python Books
Browse the docs online or d
your own.
Python's documentation, tutorials, and guides are
Get started here, or scroll down for documentatio
Python 3.x Docs
Python 2.x Docs
See also Documentation Releases by Version

练习

- 统计英文文章中出现频率最高的单词

```
import re
import operator
ss = open("harry.txt").read()
listoftokens = re.split(r"[\s\"\.\t\,!0-9]", ss)
dic = {}
for s in listoftokens:
    s = s.lower()
    if len(s) > 3:
        if s in dic:
            dic[s] = dic[s] + 1
        else:
            dic[s] = 1
t = sorted(dic.items(), key=operator.itemgetter(1), reverse=True)
print(t)
```

练习

- 用猜牌例子体会
  - 语言是工具
  - 算法是灵魂
- 用2048例子体会
  - 学无止境

```
============================
 178  130   57   66  220
  26  210   16   83  178
  97  239  170   68  149
 Please remember one num
============================
[178, 130, 57, 66, 220]
Is it in? (y or n)
Y
[26, 83, 130, 220, 170]
Is it in? (y or n)
Y
```

```
============================
    Welcome to 2 0 4 8
----------------------------
  Input: W(Up)   S(Down)
         A(Left) D(Right)
============================
Total score: [0]
   2   4   0   2
   0   0   2   2
   0   0   2   4
   0   0   0   2
operator:
```

Questions?