

计算机编程



河北师范大学软件学院
Software College of Hebei Normal University





- 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)



- 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
- 算法

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

日常生活中的程序

银行

3. 将存折和取款单递给银行职员

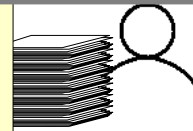
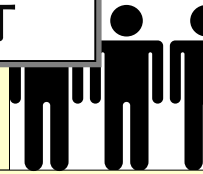
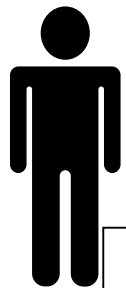
4. 银行职员办理取款事宜

5. 拿到钱并离开银行

2. 填写取款单并到相应窗口排队

1. 带上存折去银行

银行



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。

程序设计语言都有哪些呢

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

- 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。

0000,0000,000000001000

0000,0000,000000000001

0000,0001,000000001000



- 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



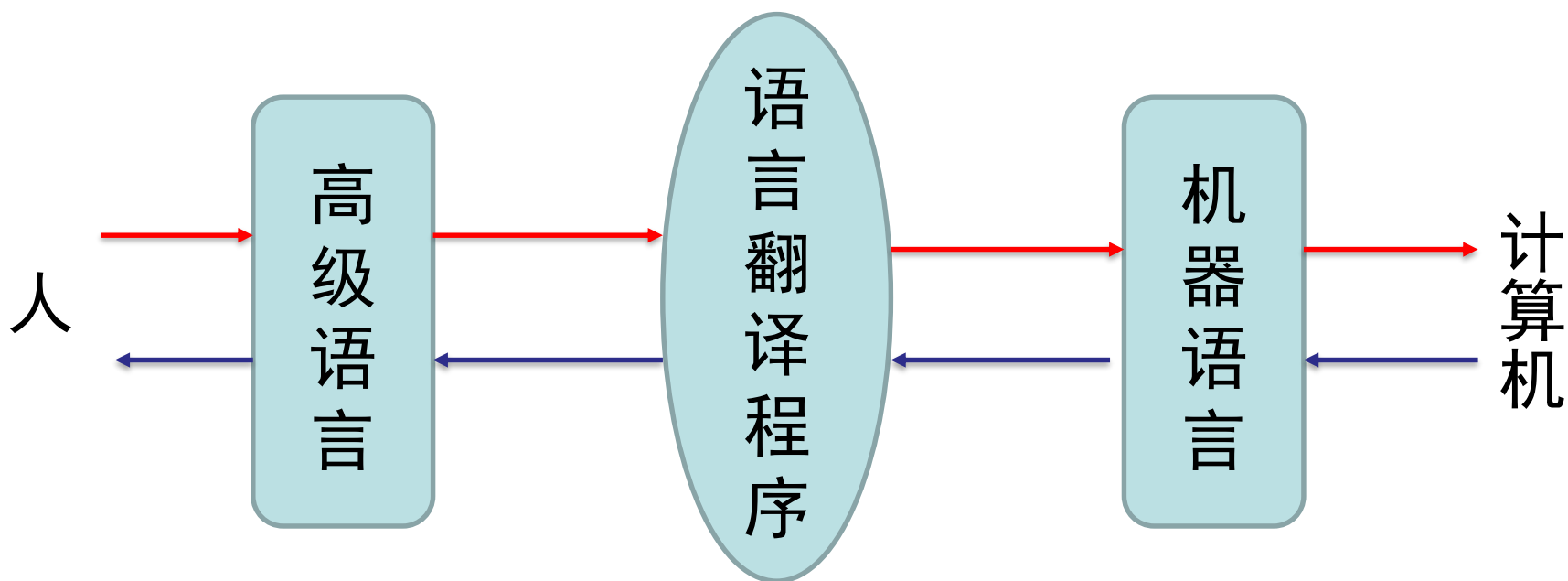
- 接近人们使用的自然语言，一条语句不仅仅是完成单一的机器指令操作，也可能是多项操作。
- 常用的高级语言有C、C++、Java等。

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

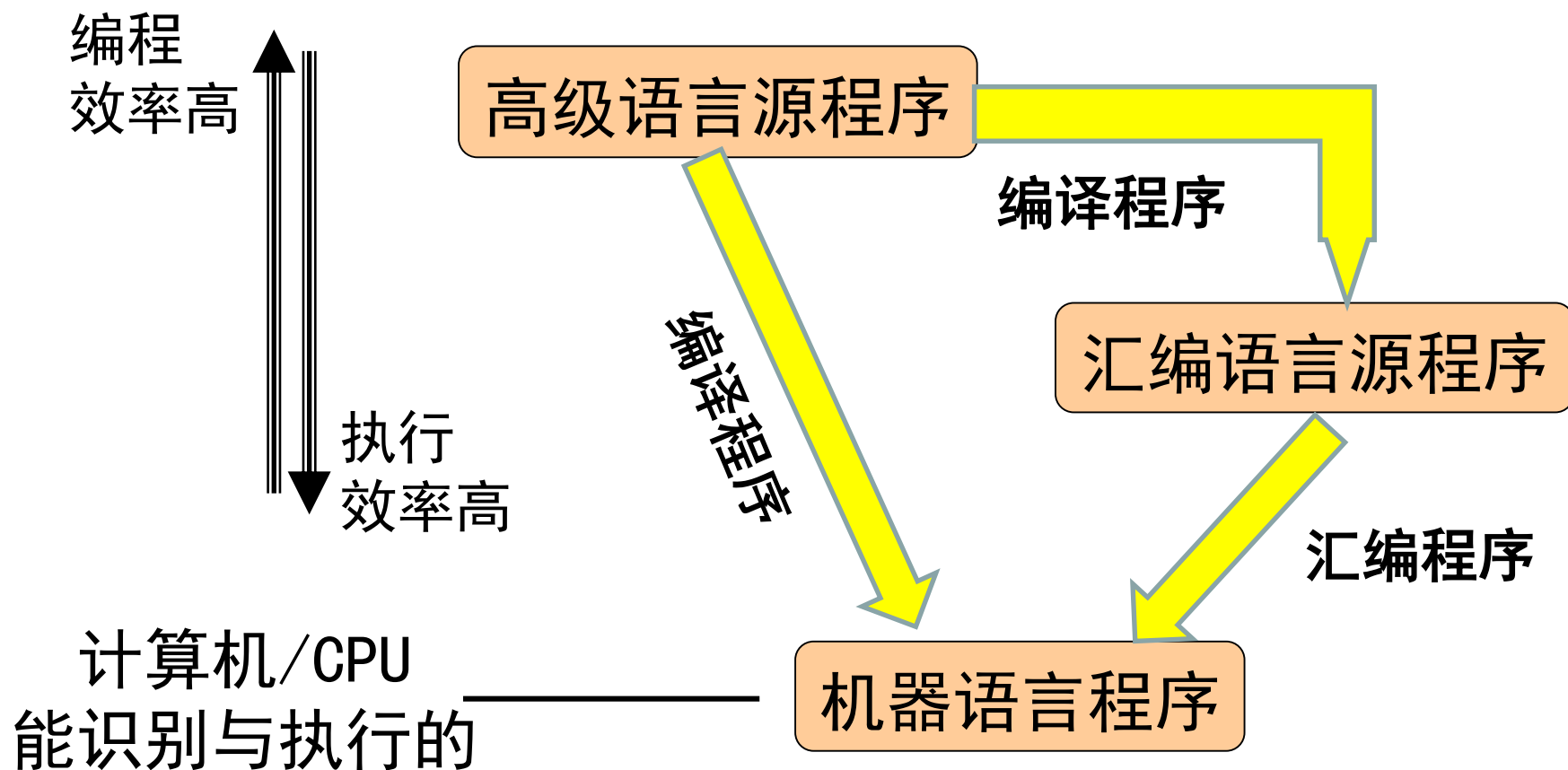


- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- Python
- 列举高级语言
- 算法

语言的层次



语言的层次





- 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

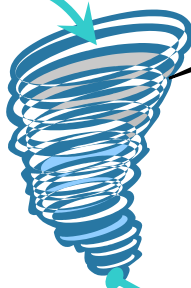


- 源程序
- 编译、解释
- 程序的执行



源程序

编译器



可执行程序





- 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
- 算法

高级语言的特点



- 高度封装, 与低级语言相对
- 使用一般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



- 字母表就是一个有穷字符集

$\Sigma = \{a-z, A-Z, 0-9, (,),$
 $[,], \rightarrow, ., !, \sim, +, -, *,$
 $/, \&, \%, <, >, =, ^,$
 $|, ?, , , ; \}$

- 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：P8
 - 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



- 算术表达式

- $1+2$

- $(1*2)+3$

- 布尔表达式

- $a \geq b$

- $1==2$

- $!(1 \leq 2)$

- $(1==1) \text{ or } (1==2)$

- $(1==1) \text{ and } (1==2)$

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
和函数调用语句`x=add (y, z)`

控制结构



顺序、分支、循环语句：

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

```
while(B<=C)  
    语句1
```

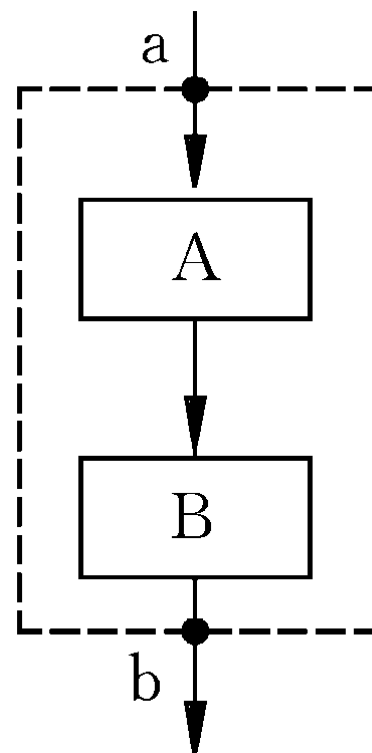
```
if(B<=C)  
    语句1  
else  
    语句2
```

```
for(int i=0;i<10;i++)  
    语句1  
或者  
for i in range (0, 10)  
    语句2
```

顺序结构

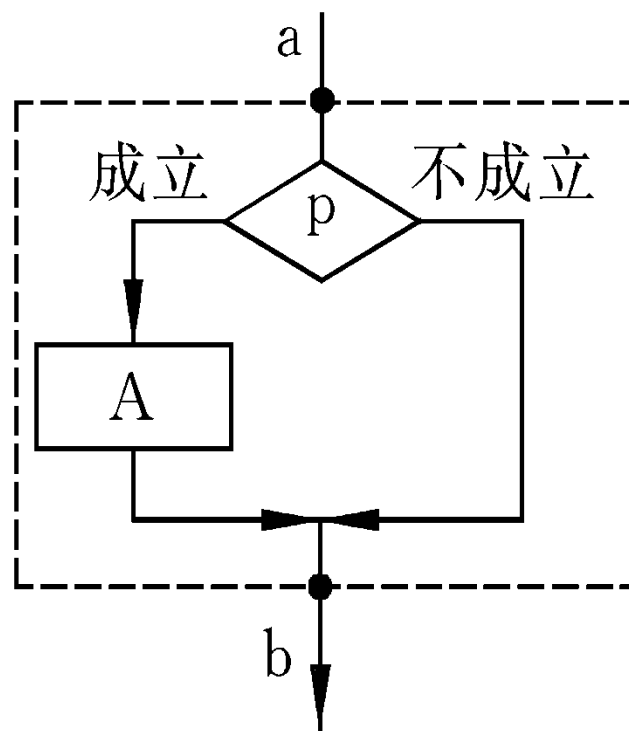
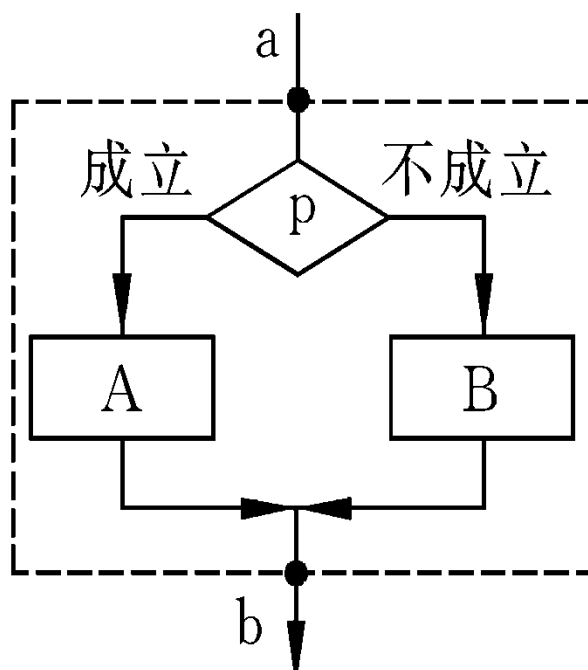


- 顺序结构是最简单的程序结构，也是最常用的程序结构，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。



选择结构

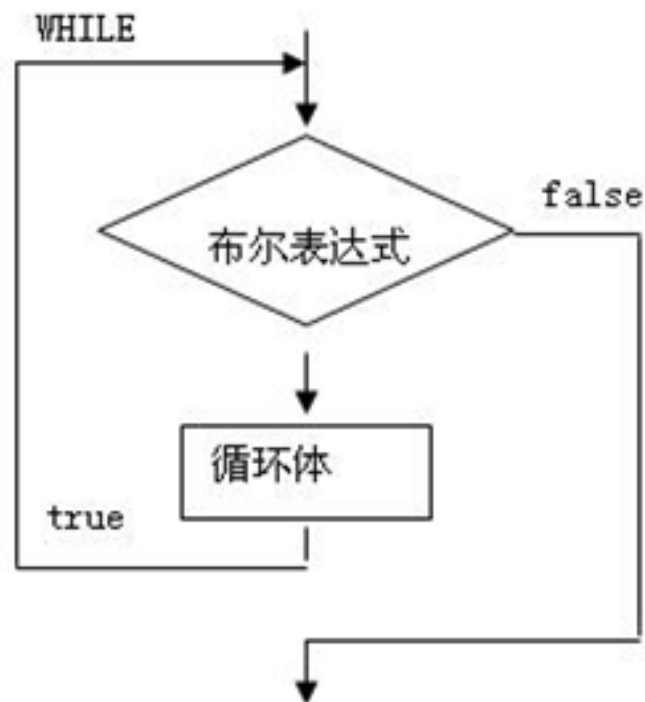
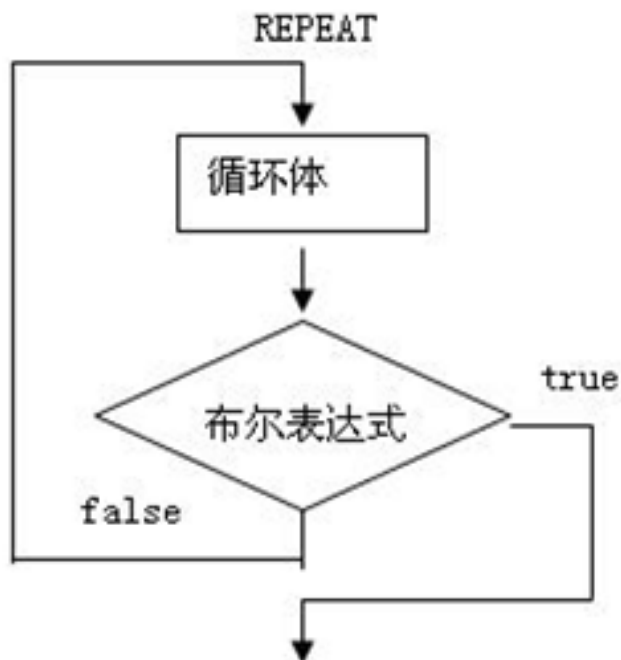
- 选择程序结构用于判断给定的条件，根据判断的结果判断某些条件，根据判断的结果来控制程序的流程。



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



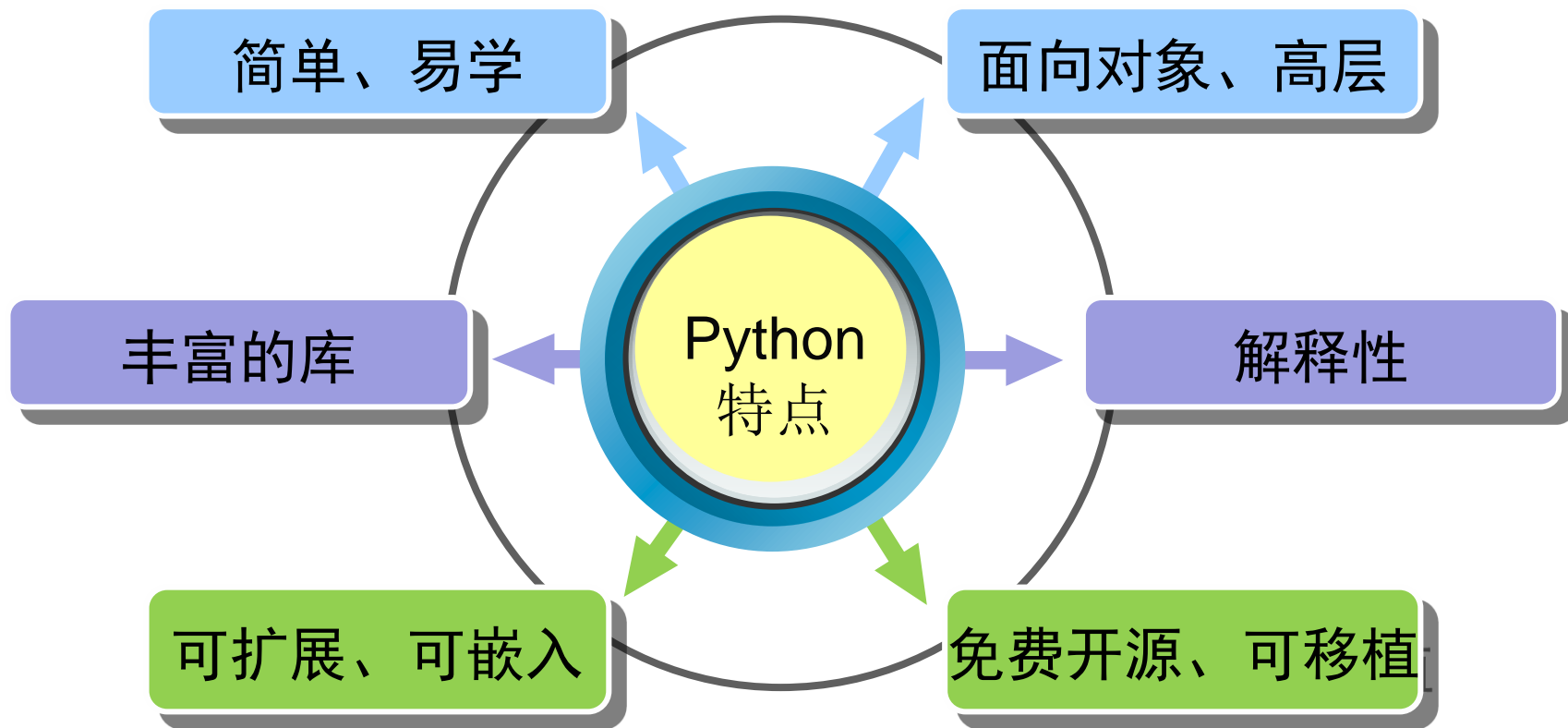


- 计算机编程的基本概念
- 语言的层次
- 初窥高级语言
- Python
- 列举高级语言
- 算法



- 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



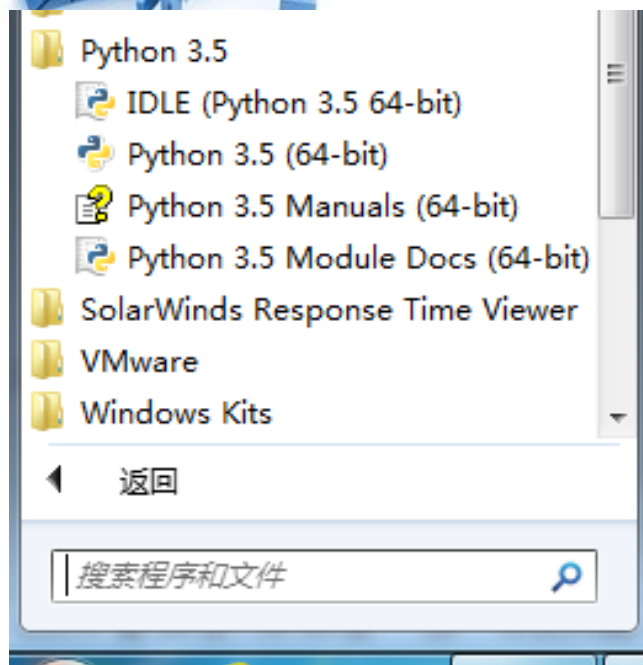
Windows中使用Python



- 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



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

引入模块

循环结构

选择结构

函数
调用

```
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>key try again')
        else:
            print('guss<key try again')
        i+=1
    else:
        print("game over")
        print("The key is:",key)
```

caishu()

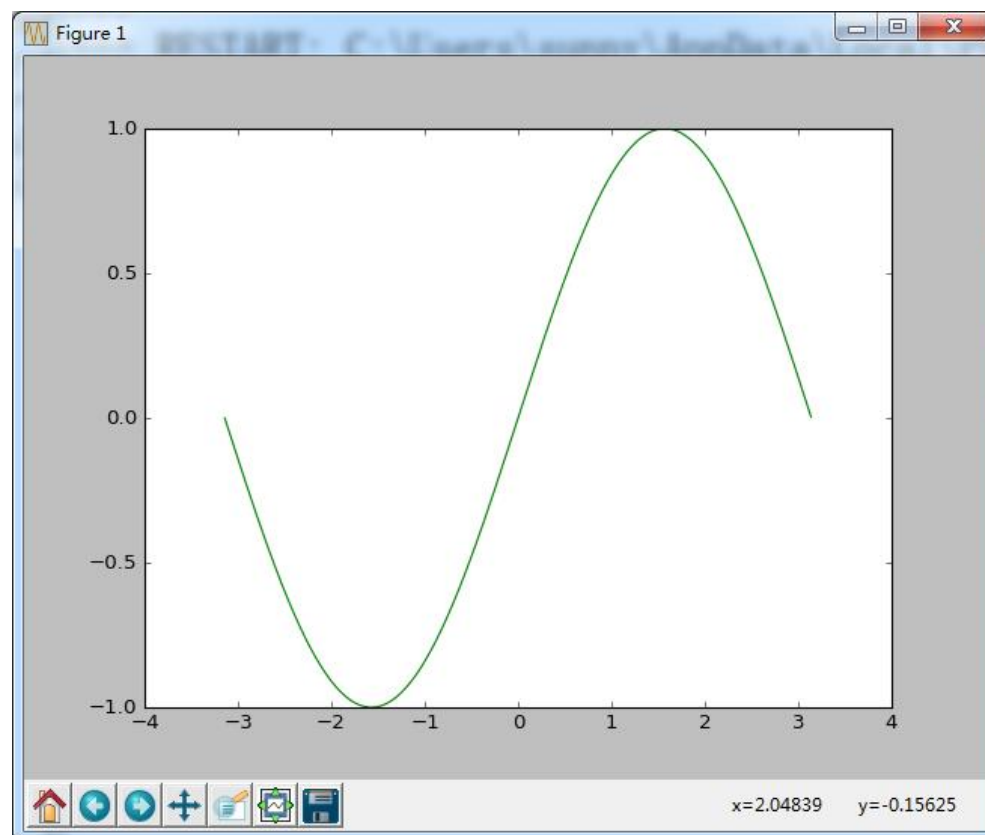
中学知识再现

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()
```

运行结果

函数调用





- Python初体验（P12）

- 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

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

```
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
```



- 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



- 数值类
- 布尔类型
- 列表
- 字符串类型

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)
```



- 数值类
- 布尔类型
- 列表
- 字符串类型

布尔类型



#<程序：布尔类型例子>

b=100<101

print(b)

布尔类型变量

布尔表达式

- 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



- 数值类
- 布尔类型
- 列表
- 字符串类型



- 有顺序编号的结构称为“**序列**”

- 列表

列表类似于数组

- 元组

不可修改的列表

- 字符串

- 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



- 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' ]]
```

区别于数组

- 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

#<程序：例子>

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


序列的通用操作



分片

分片结果中第一个元素的索引号

$\text{index2} - 1$: 最后一个元素的索引号

- `L[index1:index2]`
- `L[index1:]`
- `L[:index2]`
- `L[:]`
- `L[index1:index2:stride]`

步长

- 步长：默认为1
- 可以大于1，可以取负数，但不能为0

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

#<程序：例子>

```
L1=[1,1.3]
```

```
L2=["2","China",["I","am","another","list"]]
```

```
L =L1+L2
```

```
print(L)
```

乘 序列重复多次

列表专有方法



S=[1,2]

	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]/[1,2]
4	s.extend(t)	将列表t添加到列表s的末尾	['3','4']	[1,2,'3','4']
5	s.insert(i,x)	将数据x插入到s的第i号位置	0,'3'	['3',1,2]

列表专有方法



$S=[1,2]$

	函数	作用	参数	结果
6	<code>s.pop(i)</code>	将列表s第i号元素弹出并返回其值	1或无	[1]/2
7	<code>s.remove(x)</code>	删除列表s中第一个值为x的元素	1	[2]
8	<code>s.reverse()</code>	反转s中的所有元素	无	[2,1]



- 数值类
- 布尔类型
- 列表
- 字符串类型



- 字符串表示方式

- 单引号 eg: 'hello,world!'
- 双引号 eg: "hello world!"

注意：

双引号表示的字符串中有单引号



双引号表示的字符串中有双引号



单引号表示的字符串中有单引号



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)
```

键盘输入6,
程序报错

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

键盘输入6,
结果输出7



- 数值类
- 布尔类型
- 列表
- 字符串类型



- 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

```
#<程序： 基本赋值语句>  
x = 1; y = 2  
k = x + y  
print(k)
```

运行结果： 3

序列赋值



- 序列赋值形式

右侧值依次赋给
左侧变量

左侧的一系列变量 = 右侧的一系列值

```
#<程序：序列赋值语句>  
a,b=4,5  
print(b)
```

运行结果：5



- 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



```
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(88)
```

While语句-通用格式



- while语句

```
while(test1):  
    <语句块1>  
else:  
    <语句块2>
```

测试条件为真,
执行循环体

可选部分

While语句-continue



- while语句

```
while(test1):
```

```
<语句>
```

```
<语句>
```

```
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

```
while(test1):
```

```
<语句>
```

```
<语句>
```

```
break
```

- 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



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

立即结束本次循环，重新开始下一轮循环

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

立即跳出循环结构，执行while后面的语句

While语句-else



- 可选部分else语句

```
while(test1):
```

```
    <语句>
```

```
    <语句>
```

```
    break
```

```
else:
```

```
    <语句>
```

- 注意：遇到break和else

若执行了break语句，不再执行else

若没有执行break语句，test1为假时，执行else

例子：else语句



```
b=7
a=b//2
while a>1:
    if b%a==0:
        print("b is not prime")
        break
    a=a-1
else:
    print("b is prime")
|
```

•注意

- else一定和while里的break相结合考虑才有意义。



- for语句

遍历序列

```
for <target> in <object>:  
    <语句块1>  
else:  
    <语句块2>
```

- 注意：object中的每一个元素会依次赋给target



- 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)
 - *function*一词本身就有"功能"的含义
 - 我们已经熟悉的函数:
 - 自己编的函数,如常用的`main()`
 - Python内建函数,如`abs()`
 - Python库函数,如`math.sqrt()`
 - 对象的方法,如`win.close()`和`p.draw()`

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)
```


为什么需要函数



- 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

编程实例:生日歌



- 减少相同代码的重复

```
def main():  
    print (Happy birthday to you!)  
    print (Happy birthday to you!)  
    print (Happy birthday, dear Fred.)  
    print (Happy birthday to you!)
```



```
def happy():  
    print (Happy birthday to you!)  
def main():  
    happy()  
    happy()  
    print (Happy birthday, dear Fred.)  
    happy()
```

重复代码的缺点:
(1)费时费力
(2)代码一致性维护

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>)
```

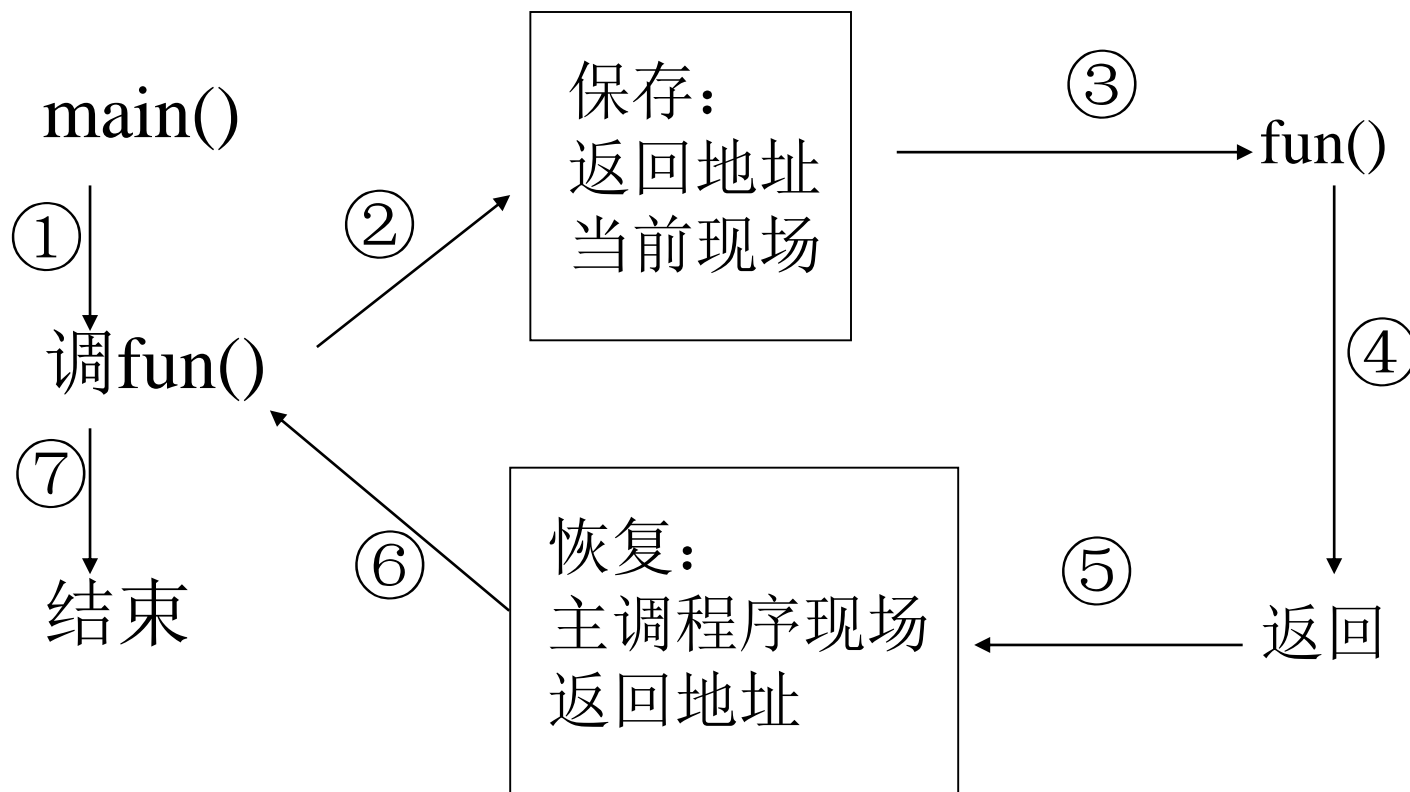
- 函数形参被赋值为实参

- 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

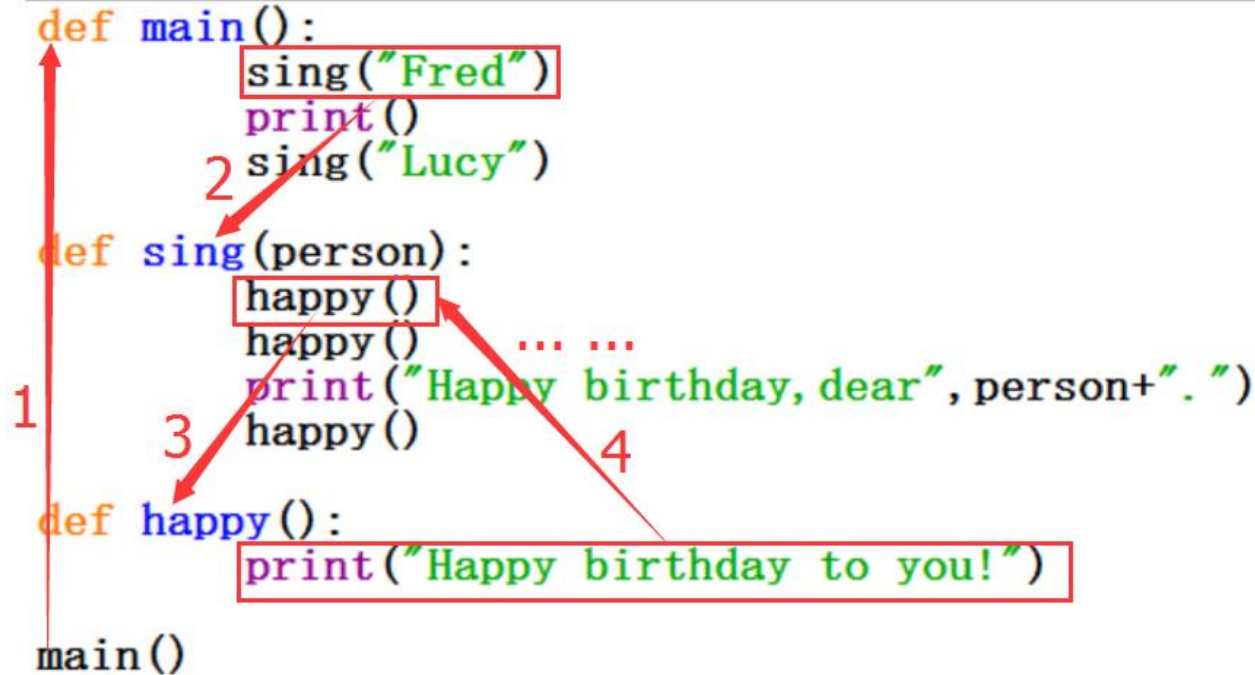
- 执行函数体

- 控制返回调用者 (调用点的下一条语句)

函数调用的执行过程细节



函数调用过程图解



- 调用过程?

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred.
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lucy.
Happy birthday to you!

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>):

.....

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

引入模块

内建函
数调用

缩进体
现逻辑

函数
调用

```
import random
def caishu():
    i=0
    key=random.randint(1, 10)
```

字符串

循环结构

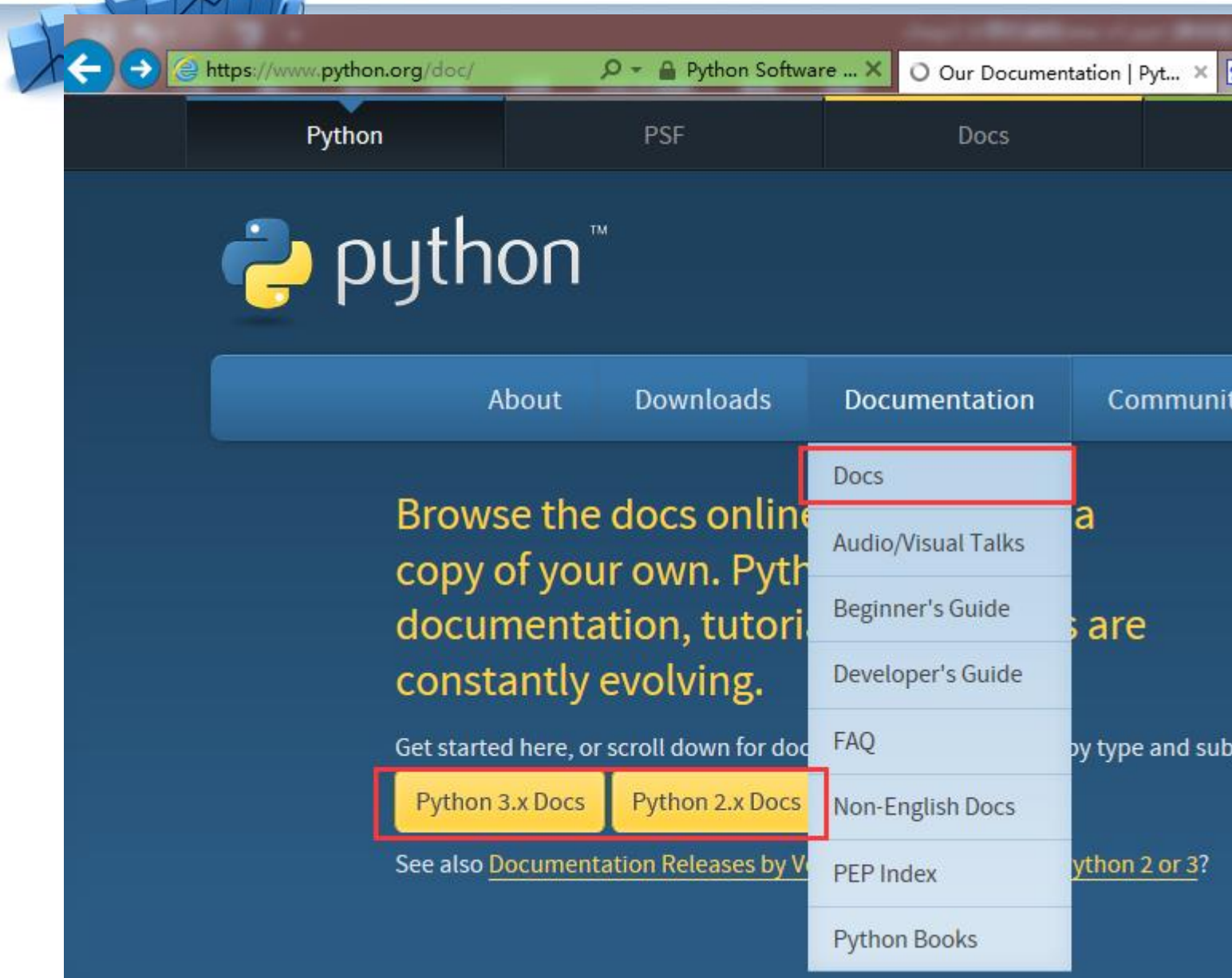
```
while i<5:
    guss=int(input("enter:"))
    if key==guss:
        print('good guess!')
        break
    elif guss>key:
        print('guss>key try again')
    else:
        print('guss<key try again')
    i+=1
else:
    print("game over")
    print("The key is:", key)
```

选择结构

内建函数
调用

caishu()

Python进阶学习



The screenshot shows the Python.org website with the documentation menu open. The browser address bar displays `https://www.python.org/doc/`. The website header includes the Python logo and navigation tabs for Python, PSF, and Docs. The main navigation bar contains links for About, Downloads, Documentation, and Community. The Documentation dropdown menu is open, showing options like Docs, Audio/Visual Talks, Beginner's Guide, Developer's Guide, FAQ, Non-English Docs, PEP Index, and Python Books. The 'Docs' option is highlighted with a red box. Below the main navigation bar, there is a section titled 'Browse the docs online' with a yellow background, followed by a section titled 'Get started here, or scroll down for doc' with a yellow background. Below this, there are two yellow buttons: 'Python 3.x Docs' and 'Python 2.x Docs', both of which are highlighted with red boxes. The text 'See also [Documentation Releases by Version](#)' is visible at the bottom of the highlighted section.

Python

PSF

Docs

python™

About Downloads Documentation Community

Docs

Audio/Visual Talks

Beginner's Guide

Developer's Guide

FAQ

Non-English Docs

PEP Index

Python Books

Browse the docs online

copy of your own. Python documentation, tutorials, and guides are constantly evolving.

Get started here, or scroll down for documentation

Python 3.x Docs Python 2.x Docs

See also [Documentation Releases by Version](#)



- 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
- 算法

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#

Web

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。



- 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#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。



- 用于网站开发和网页控制的编程语言，包括PHP，ASP，JSP和一些脚本语言JavaScript等等。



- 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



Jul 2017	Jul 2016	Change	Programming Language	Ratings	Change
1	1		Java	13.774%	-6.03%
2	2		C	7.321%	-4.92%
3	3		C++	5.576%	-0.73%
4	4		Python	3.543%	-0.62%
5	5		C#	3.518%	-0.40%
6	6		PHP	3.093%	-0.18%
7	8	▲	Visual Basic .NET	3.050%	+0.53%
8	7	▼	JavaScript	2.606%	-0.04%
9	12	▲	Delphi/Object Pascal	2.490%	+0.45%
10	55	▲	Go	2.363%	+2.20%
11	9	▼	Perl	2.334%	-0.09%
12	14	▲	Swift	2.253%	+0.29%
13	11	▼	Ruby	2.249%	+0.13%
14	10	▼	Assembly language	2.240%	-0.04%
15	17	▲	R	2.105%	+0.59%



- 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
- 算法

算法的重要性



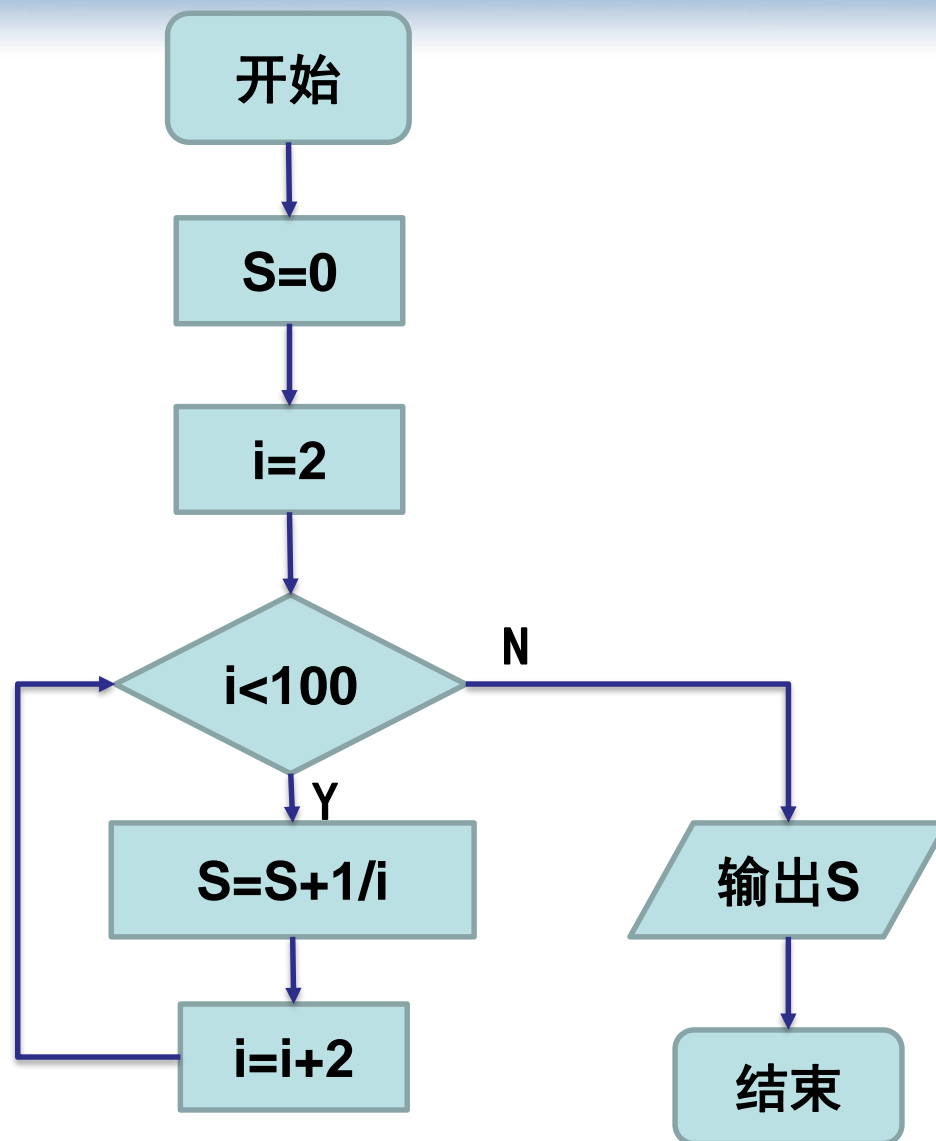
- **算法**：编程的第一步，有了好的算法，通过熟悉的语言来编写程序。
- 算法的**重要特征**
 - 有限定的运行步骤
 - 具有确定的执行步骤
 - 具有输入项
 - 输出项
 - 可行性

什么是算法



- **算法**：在有限的步骤内求解某一个问題所使用的一组定义明确的规则。
- 算法的**描述方法**
 - 伪代码（数据结构中较为普遍）
 - 流程图
 - 自然语言

流程图



算法实例：解平方根算法



- 算法一：趋近法
- 算法二：二分法
- 算法三：牛顿迭代法逼近近似解

趋近法



- `def square_root_1 () :`
- `i = 0`
- `c = 10`
- `g = 0`
- `for j range (0, c+1) :`
- `if (j*j>c and g==0) :`
- `g = j-1`
- `while (abs (g*g-c)>0.0001) :`
- `g += 0.00001`
- `i = i+1`
- `print ("%d:g = %.5f" %(i, g))`

二分法



```
• def square_root_2() :  
•     i = 0  
•     c = 10  
•     m_max = c  
•     m_min = 0  
•     g = (m_min+m_max)/2  
•     while (abs(g*g-c)>0.000000000001) :  
•         if (g*g<c) :  
•             m_min = g  
•         else :  
•             m_max = g  
•         g = (m_min+m_max)/2  
•         i = i + 1  
•     print ("%d:%.13f" % (i, g))
```

牛顿迭代法



- `def square_root_3() :`
- `i = 0`
- `c = 10`
- `g = c/2`
- `while (abs (g*g-c) > 0.0001) :`
- `g = (g + c/g) / 2`
- `i = i+1`
- `print ("%d: %.13f" % (i, g))`



- 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
- 算法

本章小结（续）

- 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

```
demo.py - C:\Users\sunny\AppData\Local\Programs\Python\Python35\demo.py (3.5.2)
File Edit Format Run Options Window Help

import random;

listAll = [];
lists = [[] for i in range(5)];

#初始游戏
def initGame():
    # -*- coding:UTF-8 -*-
    #! /usr/bin/python

import random

v = [[0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0]]

def display(v, sc):
    '''显示界面'''
    print(' {0:4}'.format(v[0][0]), end=' ')
    print(' {0:4}'.format(v[0][1]), end=' ')
    print(' {0:4}'.format(v[0][2]), end=' ')
    print(' {0:4}'.format(v[0][3]), end=' ')
    print('')
    print(' {0:4}'.format(v[1][0]), end=' ')
    print(' {0:4}'.format(v[1][1]), end=' ')
    print(' {0:4}'.format(v[1][2]), end=' ')
    print(' {0:4}'.format(v[1][3]), end=' ')
    print('')
    print(' {0:4}'.format(v[2][0]), end=' ')
    print(' {0:4}'.format(v[2][1]), end=' ')
    print(' {0:4}'.format(v[2][2]), end=' ')
    print(' {0:4}'.format(v[2][3]), end=' ')
    print('')
    print(' {0:4}'.format(v[3][0]), end=' ')
    print(' {0:4}'.format(v[3][1]), end=' ')
    print(' {0:4}'.format(v[3][2]), end=' ')
    print(' {0:4}'.format(v[3][3]), end=' ')
    print('')

operator:
```

Questions?

