# 1 SQL

## 二表查询

SELECT \* FROM PRODUCT;

SELECT \* FROM CATEGORY;

SELECT \* FROM PRODUCT,CATEGORY WHERE CNO=CID;

SELECT \* FROM PRODUCT P INNER JOIN CATEGORY C ON P.CNO=C.CID;

SELECT \* FROM PRODUCT P LEFT OUTER JOIN CATEGORY C ON P.CNO=C.CID;

SELECT \* FROM PRODUCT LIMIT 0,3;

SELECT \* FROM PRODUCT LIMIT 3,3;

SELECT \* FROM PRODUCT WHERE CNO=(SELECT CID FROM CATEGORY WHERE CNAME='手机数码');

SELECT P.PNAME,C.CNAME FROM PRODUCT P LEFT OUTER JOIN CATEGORY C ON P.CNO=C.CID;--左连接

SELECT PNAME,(SELECT CNAME FROM CATEGORY C WHERE P.CNO=C.CID) AS'商品分类名称' FROM PRODUCT P;

SELECT \* FROM COMPANY WHERE NAME LIKE 'KI%' ;

SELECT \* FROM COMPANY WHERE AGE=25 OR AGE=27 ;

SELECT \* FROM COMPANY WHERE AGE > (SELECT AGE FROM COMPANY WHERE SALARY > 65000);

SELECT AGE FROM COMPANY WHERE EXISTS (SELECT AGE FROM COMPANY WHERE SALARY > 65000);

SELECT \* FROM PRODUCT P LEFT OUTER JOIN CATEGORY C ON P.CNO=C.CID UNION SELECT \* FROM PRODUCT RIGHT OUTER JOIN CATEGORY ON PRODUCT.CNO=CATEGORY.CID;

SELECT \* FROM PRODUCT P JOIN CATEGORY C ON P.CNO=C.CID

SELECT \* FROM PRODUCT P LEFT JOIN CATEGORY C ON P.CNO=C.CID

## 三表查询

CREATE TABLE AB(

ABID INT PRIMARY KEY NOT NULL,

XCV CHAR(50) NOT NULL,

ZXC INT NOT NULL

);

SELECT PID,PNAME,PRICE FROM PRODUCT P JOIN CATEGORY C ON P.CNO=C.CID JOIN AB ON C.CDESC=AB.ABID ；

SELECT \* FROM PRODUCT P JOIN CATEGORY C ON P.CNO=C.CID JOIN AB ON PRICE>100 AND XCV LIKE 'S%'--|| PID = 2)--|| ABID > 10--|| XCV LIKE 'S%';

SELECT \* FROM PRODUCT P JOIN PRODUCT R ON (P.PRICE=R.CNO);--同一个表查找

SELECT P.PNAME,R.PNAME AS ‘上级’ FROM PRODUCT P LEFT JOIN PRODUCT R ON (P.PRICE=R.CNO);--同一个表查找

SELECT P.PNAME,R.PNAME AS 上级 FROM PRODUCT P LEFT JOIN PRODUCT R ON (P.PRICE=R.CNO) WHERE P.PRICE<99;--SQL99比SQL92简单，条理清晰

SELECT P.PNAME,R.PNAME AS 上级,CATEGORY.CDESC FROM PRODUCT P LEFT JOIN PRODUCT R ON (P.PRICE=R.CNO) JOIN CATEGORY ON CATEGORY.CDESC BETWEEN 10 AND 11 WHERE P.PRICE<99;--三个表查询

SELECT AB.ABID,AB.XCV,P.PNAME,R.PNAME AS 上级,CATEGORY.CDESC FROM PRODUCT P LEFT JOIN PRODUCT R ON (P.PRICE=R.CNO) JOIN CATEGORY ON CATEGORY.CDESC BETWEEN 10 AND 11 JOIN AB ON AB.ABID BETWEEN 12 AND 13 WHERE P.PRICE<99;--四个个表查询

# C++



静态变量可以起到全局变量的作用



全局变量extern

|  |  |
| --- | --- |
| [continue 语句](https://www.runoob.com/cplusplus/cpp-continue-statement.html" \o "C++ 中的 continue 语句) | 引起循环跳过主体的剩余部分，立即重新开始测试条件。 |

for( ; ; )

{ printf("This loop will run forever.\n"); }

当条件表达式不存在时，它被假设为真。您也可以设置一个初始值和增量表达式，但是一般情况下，C++ 程序员偏向于使用 for(;;) 结构来表示一个无限循环。

**注意：**您可以按 Ctrl + C 键终止一个无限循环。

我们已经在前面的章节中讲解了 **[条件运算符 ? :](https://www.runoob.com/cplusplus/cpp-conditional-operator.html)**，可以用来替代 **if...else** 语句。它的一般形式如下：

Exp1 ? Exp2 : Exp3;

其中，Exp1、Exp2 和 Exp3 是表达式。请注意，冒号的使用和位置。

? 表达式的值是由 Exp1 决定的。如果 Exp1 为真，则计算 Exp2 的值，结果即为整个 ? 表达式的值。如果 Exp1 为假，则计算 Exp3 的值，结果即为整个 ? 表达式的值。

在 C++ 中要声明一个数组，需要指定元素的类型和元素的数量，如下所示：

type arrayName [ arraySize ];

这叫做一维数组。**arraySize** 必须是一个大于零的整数常量，**type** 可以是任意有效的 C++ 数据类型。

大括号 { } 之间的值的数目不能大于我们在数组声明时在方括号 [ ] 中指定的元素数目。

如果您省略掉了数组的大小，数组的大小则为初始化时元素的个数。因此，如果：

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

您将创建一个数组，它与前一个实例中所创建的数组是完全相同的。下面是一个为数组中某个元素赋值的实例：

balance[4] = 50.0;

#include <iostream>

using namespace std;

#include <iomanip>

using std::setw;

int main()

{

int n[10]; // n 是一个包含 10 个整数的数组

// 初始化数组元素

for (int i = 0; i < 10; i++)

{

n[i] = i + 100; // 设置元素 i 为 i + 100

}

cout << "Element" << setw(13) << "Value" << endl;

// 输出数组中每个元素的值

for (int j = 0; j < 10; j++)

{

cout << setw(7) << j << setw(13) << n[j] << endl;

}

return 0;

}

例如，下面的声明创建了一个三维 5 . 10 . 4 整型数组：

int threedim[5][10][4];

多维数组可以通过在括号内为每行指定值来进行初始化。下面是一个带有 3 行 4 列的数组。

int a[3][4] = {

{0, 1, 2, 3} , /\* 初始化索引号为 0 的行 \*/

{4, 5, 6, 7} , /\* 初始化索引号为 1 的行 \*/

{8, 9, 10, 11} /\* 初始化索引号为 2 的行 \*/};

内部嵌套的括号是可选的，下面的初始化与上面是等同的：

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

#include <iostream>

using namespace std;

int main()

{

// 一个带有 5 行 2 列的数组

int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8} };

// 输出数组中每个元素的值

for (int i = 0; i < 5; i++)

for (int j = 0; j < 2; j++)

{

cout << "a[" << i << "][" << j << "]: ";

cout << a[i][j] << endl;

}

return 0;

}

数组元素可以通过数组名称加索引进行访问。元素的索引是放在方括号内，跟在数组名称的后边。例如：

double salary = balance[9];

**runoobAarray** 是一个指向 &runoobAarray[0] 的指针，即数组 runoobAarray 的第一个元素的地址。因此，下面的程序片段把 **p** 赋值为 **runoobAarray** 的第一个元素的地址：

double \*p;double runoobAarray[10];

p = runoobAarray;

数组名是指向数组中第一个元素的常量指针。

使用数组名作为常量指针是合法的，反之亦然。因此，\*(runoobAarray + 4) 是一种访问 runoobAarray[4] 数据的合法方式。

一旦您把第一个元素的地址存储在 p 中，您就可以使用 \*p、\*(p+1)、\*(p+2) 等来访问数组元素。

#include <iostream>

using namespace std;

int main()

{

// 带有 5 个元素的双精度浮点型数组

double runoobAarray[5] = { 1000.0, 2.0, 3.4, 17.0, 50.0 };

double\* p;

p = runoobAarray;

// 输出数组中每个元素的值

cout << "使用指针的数组值 " << endl;

for (int i = 0; i < 5; i++)

{

cout << "\*(p + " << i << ") : ";

cout << \*(p + i) << endl;

}

cout << "使用 runoobAarray 作为地址的数组值 " << endl;

for (int i = 0; i < 5; i++)

{

cout << "\*(runoobAarray + " << i << ") : ";

cout << \*(runoobAarray + i) << endl;

}

cout << "P地址："<<p;

return 0;

}

形式参数是一个指针：

void myFunction(int \*param){...}

形式参数是一个已定义大小的数组：

void myFunction(int param[10]){...}

形式参数是一个未定义大小的数组：

void myFunction(int param[]){...}

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

// 要生成和返回随机数的函数

int\* getRandom()

{

static int r[10];

// 设置种子

srand((unsigned)time(NULL));

for (int i = 0; i < 10; ++i)

{

r[i] = rand();

cout << r[i] << endl;

}

return r;

}

// 要调用上面定义函数的主函数

int main()

{

// 一个指向整数的指针

int\* p;

p = getRandom();

for (int i = 0; i < 10; i++)

{

cout << "\*(p + " << i << ") : ";

cout << \*(p + i) << endl;

}

return 0;

}

### 字符串

#include <iostream>

using namespace std;

int main()

{

char site[7] = { 'R', 'U', 'N', 'O', 'O', 'B', '\0' };

cout << "菜鸟教程: ";

cout << site << endl;

return 0;

}

### 字符串连接

#include <iostream>

using namespace std;

int main()

{

string str1 = "runoob";

string str2 = "google";

string str = str1 + str2;

cout << str << endl;

return 0;

}

|  |  |
| --- | --- |
| **序号** | **函数 & 目的** |
| 1 | **strcpy(s1, s2);** 复制字符串 s2 到字符串 s1。 |
| 2 | **strcat(s1, s2);** 连接字符串 s2 到字符串 s1 的末尾。连接字符串也可以用 **+** 号，例如:  string str1 = "runoob";string str2 = "google";string str = str1 + str2; |
| 3 | **strlen(s1);** 返回字符串 s1 的长度。 |
| 4 | **strcmp(s1, s2);** 如果 s1 和 s2 是相同的，则返回 0；如果 s1<s2 则返回值小于 0；如果 s1>s2 则返回值大于 0。 |
| 5 | **strchr(s1, ch);** 返回一个指针，指向字符串 s1 中字符 ch 的第一次出现的位置。 |
| 6 | **strstr(s1, s2);** 返回一个指针，指向字符串 s1 中字符串 s2 的第一次出现的位置。 |

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

char str1[13] = "runoob";

char str2[13] = "google";

char str3[13];

int len;

// 复制 str1 到 str3

strcpy(str3, str1);

cout << "strcpy( str3, str1) : " << str3 << endl;

// 连接 str1 和 str2

strcat(str1, str2);

cout << "strcat( str1, str2): " << str1 << endl;

// 连接后，str1 的总长度

len = strlen(str1);

cout << "strlen(str1) : " << len << endl;

return 0;

}

#include <iostream>

#include <string>

using namespace std;

int main()

{

string str1 = "runoob";

string str2 = "google";

string str3;

int len;

// 复制 str1 到 str3

str3 = str1;

cout << "str3 : " << str3 << endl;

// 连接 str1 和 str2

str3 = str1 + str2;

cout << "str1 + str2 : " << str3 << endl;

// 连接后，str3 的总长度

len = str3.size();

cout << "str3.size() : " << len << endl;

return 0;

}

### 静态变量和动态变量

静态变量，就是在定义的时候，有static 修饰的变量，形式为  
static TYPE var\_name = init\_value;  
而动态变量，形式为  
TYPE var\_name = init\_value;  
即没有static 修饰。其中的=init\_value均可省略。  
区分定义在函数外的全局变量，和函数内的局部变量，作用域，生命周期，及无显式初始化时的初始值，均有区别。  
1 动态全局变量：  
作用域为整个项目，即最终编译成可执行文件的所有文件中均可以使用动态全局变量。  
生命周期为从程序运行到程序退出，即贯穿整个运行时间。  
无显式初始化时默认初始化值为0。  
  
2 静态全局变量：  
作用域为当前文件，从定义/声明位置到文件结尾。  
生命周期为从程序运行到程序退出，即贯穿整个运行时间。  
无显式初始化时默认初始化值为0。  
  
3 动态局部变量：  
作用域为当前函数，从定义位置，到其所在的{}的结束位置。  
生命周期为从函数调用到函数退出。  
无显式初始化时默认初始化值为随机值。  
  
4 静态局部变量：  
作用域为当前函数，从定义位置，到其所在的{}的结束位置。  
生命周期为从程序运行到程序退出，即贯穿整个运行时间，当下次函数调用时，静态局部变量不会被再次初始化，而是沿用上次函数退出时的值。  
无显式初始化时默认初始化值为0。

### 指针

#include <iostream>

using namespace std;

int main()

{

int var = 20; // 实际变量的声明

int\* ip; // 指针变量的声明

ip = &var; // 在指针变量中存储 var 的地址

cout << "Value of var variable: ";

cout << var << endl;

// 输出在指针变量中存储的地址

cout << "Address stored in ip variable: ";

cout << ip << endl;

// 访问指针中地址的值

cout << "Value of \*ip variable: ";

cout << \*ip << endl;

return 0;

}

假设 **ptr** 是一个指向地址 1000 的整型指针，是一个 32 位的整数，让我们对该指针执行下列的算术运算：

ptr++

在执行完上述的运算之后，**ptr** 将指向位置 1004，因为 ptr 每增加一次，它都将指向下一个整数位置，即当前位置往后移 4 个字节。这个运算会在不影响内存位置中实际值的情况下，移动指针到下一个内存位置。如果 **ptr** 指向一个地址为 1000 的字符，上面的运算会导致指针指向位置 1001，因为下一个字符位置是在 1001。

#include <iostream>

using namespace std;

const int MAX = 3;

int main()

{

int var[MAX] = { 10, 100, 200 };

int\* ptr;

// 指针中的数组地址

ptr = var;

for (int i = 0; i < MAX; i++)

{

cout << "Address of var[" << i << "] = ";

cout << ptr << endl;

cout << "Value of var[" << i << "] = ";

cout << \*ptr << endl;

// 移动到下一个位置

ptr++;

}

return 0;

}

递减一个指针

同样地，对指针进行递减运算，即把值减去其数据类型的字节数，如下所示：

实例

#include <iostream>

using namespace std;

const int MAX = 3;

int main()

{

int var[MAX] = { 10, 100, 200 };

int\* ptr;

// 指针中最后一个元素的地址

ptr = &var[MAX - 1];

for (int i = MAX; i > 0; i--)

{

cout << "Address of var[" << i << "] = ";

cout << ptr << endl;

cout << "Value of var[" << i << "] = ";

cout << \*ptr << endl;

// 移动到下一个位置

ptr--;

}

return 0;

}

试想变量名称是变量附属在内存位置中的标签，您可以把引用当成是变量附属在内存位置中的第二个标签。因此，您可以通过原始变量名称或引用来访问变量的内容。例如：

int i = 17;

我们可以为 i 声明引用变量，如下所示：

int& r = i;double& s = d;

#include <iostream>

using namespace std;

int main()

{

// 声明简单的变量

int i;

double d;

// 声明引用变量

int& r = i;

double& s = d;

i = 5;

cout << "Value of i : " << i << endl;

cout << "Value of i reference : " << r << endl;

d = 11.7;

cout << "Value of d : " << d << endl;

cout << "Value of d reference : " << s << endl;

return 0;

}

我们已经讨论了如何使用指针来实现引用调用函数。下面的实例使用了引用来实现引用调用函数。

#include <iostream>

using namespace std;

// 函数声明

void swap(int& x, int& y);

int main()

{

// 局部变量声明

int a = 100;

int b = 200;

cout << "交换前，a 的值：" << a << endl;

cout << "交换前，b 的值：" << b << endl;

/\* 调用函数来交换值 \*/

swap(a, b);

cout << "交换后，a 的值：" << a << endl;

cout << "交换后，b 的值：" << b << endl;

return 0;

}

// 函数定义

void swap(int& x, int& y)

{

int temp;

temp = x; /\* 保存地址 x 的值 \*/

x = y; /\* 把 y 赋值给 x \*/

y = temp; /\* 把 x 赋值给 y \*/

return;

}

通过使用引用来替代指针，会使 C++ 程序更容易阅读和维护。C++ 函数可以返回一个引用，方式与返回一个指针类似。

当函数返回一个引用时，则返回一个指向返回值的隐式指针。这样，函数就可以放在赋值语句的左边。例如，请看下面这个简单的程序：

#include <iostream>

using namespace std;

double vals[] = { 10.1, 12.6, 33.1, 24.1, 50.0 };

double& setValues(int i) {

double& ref = vals[i];

return ref; // 返回第 i 个元素的引用，ref 是一个引用变量，ref 引用 vals[i]，最后再返回 shit。

}

// 要调用上面定义函数的主函数

int main()

{

cout << "改变前的值" << endl;

for (int i = 0; i < 5; i++)

{

cout << "vals[" << i << "] = ";

cout << vals[i] << endl;

}

setValues(1) = 20.23; // 改变第 2 个元素

setValues(3) = 70.8; // 改变第 4 个元素

cout << "改变后的值" << endl;

for (int i = 0; i < 5; i++)

{

cout << "vals[" << i << "] = ";

cout << vals[i] << endl;

}

return 0;

}

当返回一个引用时，要注意被引用的对象不能超出作用域。所以返回一个对局部变量的引用是不合法的，但是，可以返回一个对静态变量的引用。

int& func() {

int q;

//! return q; // 在编译时发生错误

static int x;

return x; // 安全，x 在函数作用域外依然是有效的

}

### 时间

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

// 基于当前系统的当前日期/时间

time\_t now = time(0);

// 把 now 转换为字符串形式

char\* dt = ctime(&now);

cout << "本地日期和时间：" << dt << endl;

// 把 now 转换为 tm 结构

tm\* gmtm = gmtime(&now);

dt = asctime(gmtm);

cout << "UTC 日期和时间：" << dt << endl;

}

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

// 基于当前系统的当前日期/时间

time\_t now = time(0);

cout << "1970 到目前经过秒数:" << now << endl;

tm\* ltm = localtime(&now);

// 输出 tm 结构的各个组成部分

cout << "年: " << 1900 + ltm->tm\_year << endl;

cout << "月: " << 1 + ltm->tm\_mon << endl;

cout << "日: " << ltm->tm\_mday << endl;

cout << "时间: " << ltm->tm\_hour << ":";

cout << ltm->tm\_min << ":";

cout << ltm->tm\_sec << endl;

}

### GIT 配置

<https://blog.csdn.net/huangqqdy/article/details/83032408>

git安装好去GitHub上注册一个账号，注册好后，点击桌面上的Git Bash快捷图标，我们要用账号进行环境配置啦啦

# 配置用户名

git config --global user.name "username" //（ "username"是自己的账户名，）

# 配置邮箱

git config --global user.email "username@email.com" //("username@email.com"注册账号时用的邮箱)

以上命令执行结束后，可用 git config --global --list 命令查看配置是否OK

生成ssh

继续刚才的操作，在命令框中输入以下命令，

ssh-keygen -t rsa

1

然后连敲三次回车键，结束后去系统盘目录下（一般在 C:\Users\你的用户名.ssh）(mac: /Users/用户/.ssh）查看是否有。ssh文件夹生成，此文件夹中以下两个文件

4. 将ssh文件夹中的公钥（ id\_rsa.pub）添加到GitHub管理平台中，在GitHub的个人账户的设置中找到如下界面

title随便起一个，将公钥（ id\_rsa.pub）文件中内容复制粘贴到key中，然后点击Ass SSH key就好啦

5. 测试一下配置是否成功，在Git Bush命令框（就是刚才配置账号和邮箱的命令框）中继续输入以下命令，回车

ssh -T git@github.com

1

要是看见下面的这句话就说明配置好啦

在命令行 输入 git init  然后回车就好了

如果远程仓库信息有误，则删除本地仓库配置，并且设置相关地址

git remote rm origingit remote add origin XXXX

touch+文件名，直接新建一个文件

git pull origin master拉取

退出log按键q

head 命令可用于查看文件的开头部分的内容，有一个常用的参数 **-n** 用于显示行数，默认为 10，即显示 10 行的内容

# PYTHON3

list =["a", "b", "c"]

print (list)

print ('hello');print ('runoob');

word = 'word'

sentence = "这是一个句子。"

paragraph = """这是一个段落。

包含了多个语句"""

#!/usr/bin/python

counter = 100 # 赋值整型变量

miles = 1000.0 # 浮点型

name = "John" # 字符串

print (counter)

print (miles)

print (name)

Python有五个标准的数据类型：

* Numbers（数字）
* String（字符串）
* List（列表）
* Tuple（元组）
* Dictionary（字典）

列表

s = 'abcdef'

s[1:5]

'bcde'

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

str = 'Hello World!'

print str # 输出完整字符串

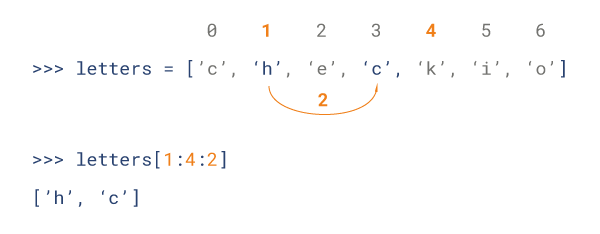
print str[0] # 输出字符串中的第一个字符

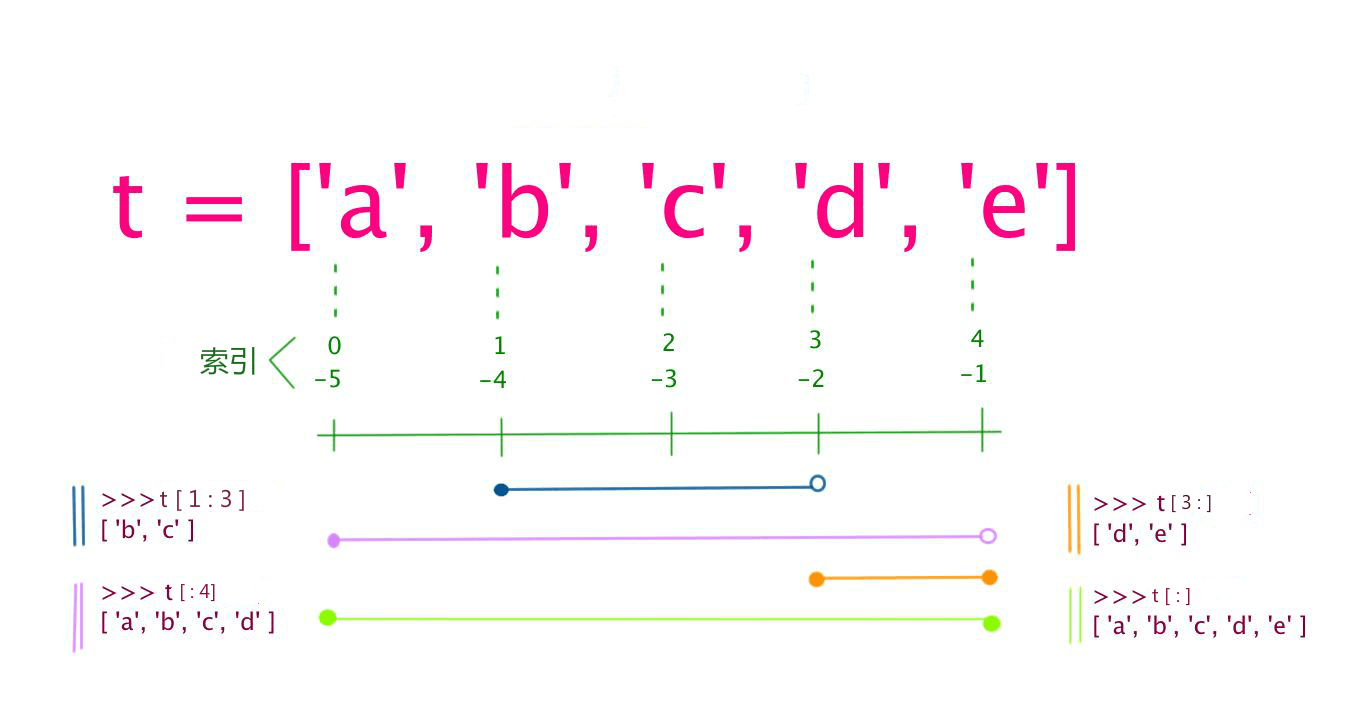
print str[2:5] # 输出字符串中第三个至第六个之间的字符串

print str[2:] # 输出从第三个字符开始的字符串

print str \* 2 # 输出字符串两次

print str + "TEST" # 输出连接的字符串





#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print list # 输出完整列表

print list[0] # 输出列表的第一个元素

print list[1:3] # 输出第二个至第三个元素

print list[2:] # 输出从第三个开始至列表末尾的所有元素

print tinylist \* 2 # 输出列表两次

print list + tinylist # 打印组合的列表

元组

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

tuple = ( 'runoob', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print tuple # 输出完整元组

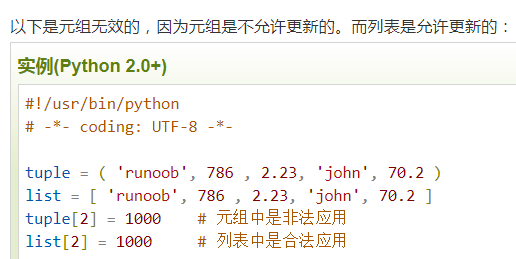
print tuple[0] # 输出元组的第一个元素

print tuple[1:3] # 输出第二个至第四个（不包含）的元素

print tuple[2:] # 输出从第三个开始至列表末尾的所有元素

print tinytuple \* 2 # 输出元组两次

print tuple + tinytuple # 打印组合的元组



字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象集合，字典是无序的对象集合。

两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'runoob','code':6734, 'dept': 'sales'}

print dict['one'] # 输出键为'one' 的值

print dict[2] # 输出键为 2 的值

print tinydict # 输出完整的字典

print tinydict.keys() # 输出所有键

print tinydict.values() # 输出所有值

|  |  |
| --- | --- |
| [int(x [,base])](https://www.runoob.com/python/python-func-int.html" \t "https://www.runoob.com/python/_blank) | 将x转换为一个整数 |
| [long(x [,base] )](https://www.runoob.com/python/python-func-long.html" \t "https://www.runoob.com/python/_blank) | 将x转换为一个长整数 |
| [float(x)](https://www.runoob.com/python/python-func-float.html" \t "https://www.runoob.com/python/_blank) | 将x转换到一个浮点数 |
| [complex(real [,imag])](https://www.runoob.com/python/python-func-complex.html" \t "https://www.runoob.com/python/_blank) | 创建一个复数 |
| [str(x)](https://www.runoob.com/python/python-func-str.html" \t "https://www.runoob.com/python/_blank) | 将对象 x 转换为字符串 |
| [repr(x)](https://www.runoob.com/python/python-func-repr.html" \t "https://www.runoob.com/python/_blank) | 将对象 x 转换为表达式字符串 |
| [eval(str)](https://www.runoob.com/python/python-func-eval.html" \t "https://www.runoob.com/python/_blank) | 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)](https://www.runoob.com/python/att-tuple-tuple.html" \t "https://www.runoob.com/python/_blank) | 将序列 s 转换为一个元组 |
| [list(s)](https://www.runoob.com/python/att-list-list.html" \t "https://www.runoob.com/python/_blank) | 将序列 s 转换为一个列表 |
| [set(s)](https://www.runoob.com/python/python-func-set.html" \t "https://www.runoob.com/python/_blank) | 转换为可变集合 |
| [dict(d)](https://www.runoob.com/python/python-func-dict.html" \t "https://www.runoob.com/python/_blank) | 创建一个字典。d 必须是一个序列 (key,value)元组。 |
| [frozenset(s)](https://www.runoob.com/python/python-func-frozenset.html" \t "https://www.runoob.com/python/_blank) | 转换为不可变集合 |
| [chr(x)](https://www.runoob.com/python/python-func-chr.html" \t "https://www.runoob.com/python/_blank) | 将一个整数转换为一个字符 |
| [unichr(x)](https://www.runoob.com/python/python-func-unichr.html" \t "https://www.runoob.com/python/_blank) | 将一个整数转换为Unicode字符 |
| [ord(x)](https://www.runoob.com/python/python-func-ord.html" \t "https://www.runoob.com/python/_blank) | 将一个字符转换为它的整数值 |
| [hex(x)](https://www.runoob.com/python/python-func-hex.html" \t "https://www.runoob.com/python/_blank) | 将一个整数转换为一个十六进制字符串 |
| [oct(x)](https://www.runoob.com/python/python-func-oct.html" \t "https://www.runoob.com/python/_blank) | 将一个整数转换为一个八进制字符串 |

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

a = 1

b = 2

c = 0

c = a / float(b)

print "1 - c 的值为：", c

|  |  |  |
| --- | --- | --- |
| /= | 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 |
| & | 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0 | (a & b) 输出结果 12 ，二进制解释： 0000 1100 |
| | | 按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。 | (a | b) 输出结果 61 ，二进制解释： 0011 1101 |
| ^ | 按位异或运算符：当两对应的二进位相异时，结果为1 | (a ^ b) 输出结果 49 ，二进制解释： 0011 0001 |

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

a = 10

b = 20

if a and b :

print "1 - 变量 a 和 b 都为 true"

else:

print "1 - 变量 a 和 b 有一个不为 true"

if a or b :

print "2 - 变量 a 和 b 都为 true，或其中一个变量为 true"

else:

print "2 - 变量 a 和 b 都不为 true"

# 修改变量 a 的值

a = 0

if a and b :

print "3 - 变量 a 和 b 都为 true"

else:

print "3 - 变量 a 和 b 有一个不为 true"

if a or b :

print "4 - 变量 a 和 b 都为 true，或其中一个变量为 true"

else:

print "4 - 变量 a 和 b 都不为 true"

if not( a and b ):

print "5 - 变量 a 和 b 都为 false，或其中一个变量为 false"

else:

print "5 - 变量 a 和 b 都为 true"

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

a = 10

b = 20

list = [1, 2, 3, 4, 5 ];

if ( a in list ):

print "1 - 变量 a 在给定的列表中 list 中"

else:

print "1 - 变量 a 不在给定的列表中 list 中"

if ( b not in list ):

print "2 - 变量 b 不在给定的列表中 list 中"

else:

print "2 - 变量 b 在给定的列表中 list 中"

# 修改变量 a 的值

a = 2

if ( a in list ):

print "3 - 变量 a 在给定的列表中 list 中"

else:

print "3 - 变量 a 不在给定的列表中 list 中"

|  |  |  |
| --- | --- | --- |
| **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**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False |
| 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** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。 |

*is 与 == 区别：*

*is 用于判断两个变量引用对象是否为同一个(同一块内存空间)， == 用于判断引用变量的值是否相等。*

a = [1, 2, 3]

b = a

b is a

True

b == a

True

b = a[:]

b is a

False

b == a

True

Python运算符优先级

以下表格列出了从最高到最低优先级的所有运算符：

|  |  |
| --- | --- |
| **运算符** | **描述** |
| \*\* | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| \* / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ | | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += \*= \*\*= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| not and or | 逻辑运算符 |

### Else if

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# 例2：elif用法

num = 5

if num == 3: # 判断num的值

print 'boss'

elif num == 2:

print 'user'

elif num == 1:

print 'worker'

elif num < 0: # 值小于零时输出

print 'error'

else:

print 'roadman' # 条件均不成立时输出

### 循环

#!/usr/bin/python

# continue 和 break 用法

i = 1

while i < 10:

i += 1

if i%2 > 0: # 非双数时跳过输出

continue

print i # 输出双数2、4、6、8、10

i = 1

while 1: # 循环条件为1必定成立

print i # 输出1~10

i += 1

if i > 10: # 当i大于10时跳出循环

Break

### Python 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。

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

for letter in 'Python': # 第一个实例

print '当前字母 :', letter

fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # 第二个实例

print '当前水果 :', fruit

print "Good bye!"

#### 另外一种执行循环的遍历方式是通过索引，如下实例：

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

fruits = ['banana', 'apple', 'mango']

for index in range(len(fruits)):

print '当前水果 :', fruits[index]

以上实例我们使用了内置函数 len() 和 range(),函数 len() 返回列表的长度，即元素的个数。 range返回一个序列的数。

### 输出3个数

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

for num in range(10,20): # 迭代 10 到 20 之间的数字

for i in range(2,num): # 根据因子迭代

if num%i == 0: # 确定第一个因子

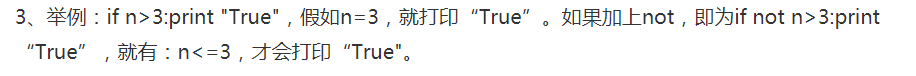
j=num/i # 计算第二个因子

print '%d 等于 %d \* %d' % (num,i,j)

break # 跳出当前循环

else: # 循环的 else 部分

print num, '是一个质数'



### 找1-100素数

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

i = 2

while(i < 100):

j = 2

while(j <= (i/j)):#第三步判断是否达到除数上限值

if not(i%j): break #第一步排除2的倍数

j = j + 1 #第二步增加除数排除更大基数

if (j > i/j) : print i, " 是素数"#除数达到上限，是素数

i = i + 1

print "Good bye!"