

# Computer Science Notes

---

- **Linux基本命令与操作**
  - 1. 终端基本使用命令
    - 1.1. 基本命令
    - 1.2. 代码细节的补充
  - 2. vi&vim编辑器操作
    - 2.1. 命令模式的基本命令
    - 2.2. shift +::末行模式命令
  - 3. c&py可光标下移n行执行文件生成与运行
    - 3.1. c的基本操作
    - 3.2. 更换c语言标准(若需要c99标准运行)
    - 3.3. py交互模式
  - 4. 系统运行级别 init
    - 4.1. 开机默认进入系统界面
    - 4.2. 关机权限
  - 5. 一些命令补充以及根目录文件作用
    - 5.1. 命令补充
    - 5.2. 根目录作用
  - 6. 解压与打包文件
- **python程序设计**
  - 1. python语言基础(2,3,4,5章为人工智能应用案例)
    - 1.1. 基础语法
    - 1.2. 关系运算符&逻辑运算符
    - 1.3. 标准键盘输入
    - 1.4. 选择结构(e.g. if语句)
    - 1.5. 循环结构(while and for)
    - 1.6. \*\*\*\*函数\*\*\*\*
    - 1.7. 字符串基础
    - 1.8. 字符串方法
    - 1.9. 列表与元组
    - 1.10. 字典与集合
  - 2. 数据处理(废案)
  - 3. 数据有关工具(废案)
  - 4. turtle 绘图(动画)(废案)
  - 5. 文件快速处理
  - 6. os模块
    - 6.1. 目录基本操作相关
    - 6.2. 文件目录方法(path模块)
    - 6.3. shutil模块(os的扩充)&zipfile模块
  - 7. 数据处理(numpy&matplotlib)
    - 7.1. 初等函数图像
    - 7.2. 画布图像(axes)
    - 7.3. 随机函数(注解)
    - 7.4. numpy模块

- 7.4.1.array(创建普通数组)
  - 7.4.2.arange、linspace and logspace(区间数组&等差等比数列)
  - 7.4.3.numpy数组属性
  - 7.5.数据的切片与索引
- **c语言程序设计**
  - 1.c语言标准库的补充以及注意事项
    - 1.1.字符串处理注解
    - 1.2.数据类型占用容量
    - 1.3.dev使用c99(期末考试看)
  - 2.函数
    - 2.1.变量&形式参数
    - 2.2.静态变量
    - 2.3.递归
  - 3.指针
    - 3.1.definition
    - 3.2.const 指针 & 指针数组
    - 3.3.多级指针
    - 3.4.指针数组与数组指针
    - 3.5.指针函数与数组指针
  - 4.结构体、联合体和枚举
    - 4.1.结构体's definition
    - 4.2.结构体数组与指针
    - 4.3.联合体(又称共用体)
    - 4.4.枚举
    - 4.5.typedef关键字
  - 5.c语言底层特性
    - 5.1.预处理文件
- **matlab 科学计算(信号与系统)**
- **单片机应用技术(stm32)**

## Linux基本命令与操作

### 1. 终端基本使用命令

#### 1.1.基本命令

ls 参数 目录地址 pwd: Print Working Directory 打印当前目录绝对路径 ls:list, 列出命令, 用于列出指定工作目录下内容(ls 参数 目录地址)

cp 复制 cp [参数] [文件名] [目的路径] -a 复制目录, 保留链接、文本数学, 并复制目录下的所有内容 -d 复制时保留链接(类似快捷方式) -f 覆盖已经存在的目标文件而无提示 -i 覆盖目标文件前提醒。回答y 时目标文件将被覆盖 -p 除复制文件外, 还把修改时间和访问权限也复制到新文件中 -r 若给出的源文件是一个目录文件, 将该目录下所有的子目录和文件一同复制 -l 不复制文件, 只生成链接(快捷方式)

mv 移动或改名(待补充) mv 原名 新名 mv 文件 对应目录地址

touch [文件名.后缀]

rm 删除 (待补充)

chown [选项] 所有者:所属组 文件名 修改文件所有者 chgrp 改变文件, 目录(-R含目录下文件)所在的组  
chmod 修改权限

## 1.2.代码细节的补充

对于ls输出而言:

-a 列出目录下的所有文件, 包括以 . 开头的隐含文件。-l 列出文件的详细信息 -r z-a反过来排序 -t 时间排序文件 -A 把所有.和.的文件去掉, 显示剩下的文件 -F一样打出所有文件, 然后用符合后缀表示文件类型 -R列出所有子目录文件 (-a和-A区别: -a显示了-A不显示的..文件)

白色: 表示普通文件 蓝色: 表示目录 绿色: 表示可执行文件 红色: 表示压缩文件 浅蓝色: 链接文件 红色闪烁: 表示链接的文件有问题 黄色: 表示设备文件灰色: 表示其他文件

## 2. vi&vim编译器操作

### 2.1.命令模式的基本命令

h l k j 移动光标

n+ 光标下移n行 n- 光标上移n行

0 (数字零) 光标移至当前行的行首 G 光标移至文档的最末行行首 gg 光标移至文档的首行行首 ndd 删除当前行及其后n-1行 nyy 复制当前行及其后n-1行

p, P p :粘贴在光标下一行 P :粘贴在光标上一行 u 撤销上一步操作

:set nu 设置文件的行号 :set nonu 取消文件的行号 打入n 再打入shift+g 将光标移动到第n行 /关键字+回车 查找关键字, 输入 n 查找下一个

### 2.2. shift + : :末行模式命令

:w 保存文件 :wq 存盘退出 :q 直接退出 :q! 不保存文件而直接退出

## 3. c&py可光标下移n行执行文件生成与运行

### 3.1. c的基本操作

gcc -o hello hello.c ./hello -o表示可自定义生成的output文件名为hello, 否则默认为啊a.out ./运行可执行程序

### 3.2. 更换c语言标准(若需要c99标准运行)

gcc -std=c99 -o hello hello.c

### 3.3. py交互模式

命令行输入python进入py交互模式 quit () 退出交互模式 输入python hello.py即可运行py文件

## 4. 系统运行级别 init

## 4.1.开机默认进入系统界面

- a) # 0 : 关机, 系统默认运行级别不能设为0, 否则不能正常启动 b) # 1 : 单用户模式, root权限, 用于系统维护, 禁止远程登陆 (找回丢失密码) c) # 2 : 多用户状态没有网络服务
- d) # 3 : 多用户状态有网络服务, 登陆后进入控制台命令行模式 e) # 4 : 系统未使用保留给用户 f) # 5 : 图形界面 g) # 6 : 系统正常关闭并重启, 默认运行级别不能设为6, 否则不能正常启动

常见的运行级别是3和5, 要修改默认的运行级别可修改文件'/etc/inittab'中的'id:3:initdefault'这一行的数字 (inittab为init配置文件)

快速切换界面 `sudo init [运行级别]`

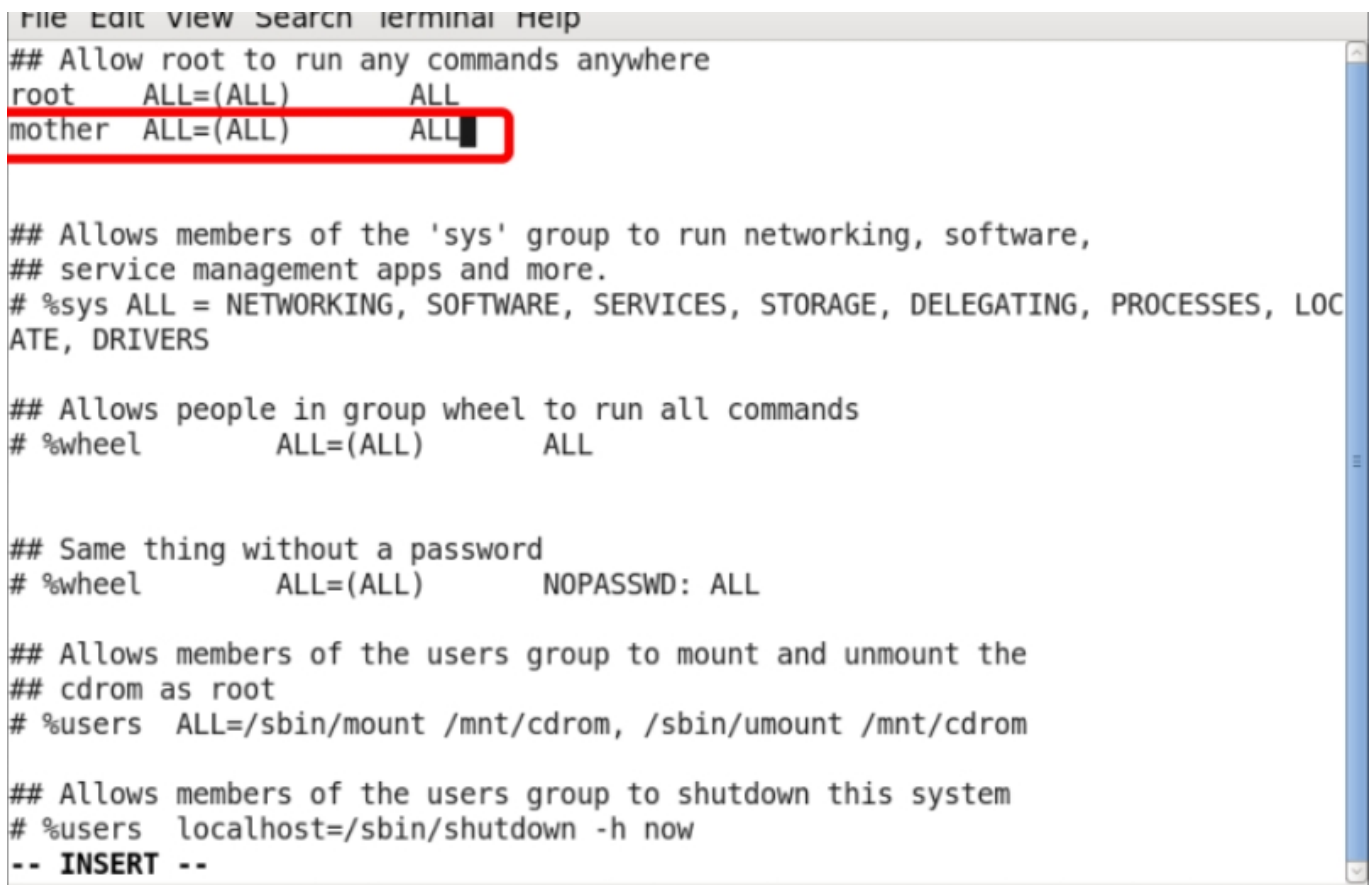
## 4.2.关机权限

shutdown 关机命令(需要管理员权限root) shutdown 参数 时间[正整数分钟] 广播 -r 关机后重新开机 -k 并不会真的关机, 只是将警告讯息传给所有使用者 -h 关机后停机 -n 不用正常程序关机,用强迫的方式杀掉所有执行中的程序后自动关机

su 切换用户 su [用户名] exit 退出root

sudo [命令行] 以超级用户身份执行命令

赋予某用户root权限 `sudo visudo` 搜索Allow, 复写一份与root同样的句子(如图所示)



```
File Edit View Search Terminal Help
## Allow root to run any commands anywhere
root    ALL=(ALL)    ALL
mother  ALL=(ALL)    ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys  ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOC
ATE, DRIVERS

## Allows people in group wheel to run all commands
# %wheel  ALL=(ALL)    ALL

## Same thing without a password
# %wheel  ALL=(ALL)    NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now
-- INSERT --
```

## 5. 一些命令补充以及根目录文件作用

### 5.1.命令补充

`vi /etc/init` +Tab 将显示目录中的类似名

`groupadd [组名]` 新建组 `su [用户名]` 切换其他用户

`useradd [用户名] -g[组名] -d[家目录地址] -u [id号码]` `id [用户名]` 查询用户 `id passwd [用户名]` 修改用户密码 (需要管理员权限) `userdel [用户名]` 删除用户

`vi /etc/passwd` 查看用户列表

```
[0765cjb@localhost Desktop]$ sudo useradd Sat
[sudo] password for 0765cjb:
[0765cjb@localhost Desktop]$ id Sat
uid=502(Sat) gid=503(Sat) groups=503(Sat)
[0765cjb@localhost Desktop]$ sudo passwd Sat
Changing password for user Sat.
New password:
BAD PASSWORD: it does not contain enough DIFFERENT characters
BAD PASSWORD: is too simple
Retype new password:
passwd: all authentication tokens updated successfully.
[0765cjb@localhost Desktop]$ sudo userdel Sat
[0765cjb@localhost Desktop]$
```

`usermod [用户] -g [移入组]`

## 5.2.根目录作用

`./bin` 重要的二进制 (binary) 应用程序,包含二进制文件, 系统的所有用户使用的命令都在这个目录下。  
`./etc` 配置文件、启动脚本等(etc)包含所有程序所需的配置文件, 也包含了用于启动/停止单个程序的启动和关闭shell脚本 脚本和相关文件, 可以一下子执行多个命令实现某些功能 `./home` 本地用户主 (home) 目录所有用户用home目录来存储他们的个人档案 类似win的document

`./usr` 包含绝大部分所有用户(users)都能访问的应用程序和文件包含二进制文件, 库文件。文档和二级程序的源代码 就是放软件的地方 (以及软件相关的数据) `./opt` 提供一个供可选的(optional)应用程序安装目录包含从各个厂商的附加应用程序, 附加的应用程序应该安装在/opt或者/opt的子目录下 驱动和接口? `./proc` 特殊的动态目录, 用以维护系统信息和状态, 包括当前运行中进程 (processes) 信息。包含系统进程的相关信息, 是一个虚拟的文件系统, 包含有关正在运行的进程的信息, 系统资源以文本信息形式存在

`./root` root (root) 用户主文件夹, 读作“slash-root”管理员的home `./sbin` 重要的系统二进制 (systembinaries)文件也是包含的二进制可执行文件。在这个目录下的linux命令通常都是由系统管理员使用的, 对系统进行维护 管理员版本的命令集 `./dev` 设备 (device)文件包含设备文件, 包括终端设备, USB或连接到系统的任何设备 集线器? `./mnt` 挂载 (mounted)文件系统临时安装目录, 系统管理员可以挂载文件系统 类似光盘iso?

`./boot` 启动 (boot) 配置文件,包含引导加载程序相关的文件 引导程序? 黑苹果也要做efi引导进入系统 `./lib`系统库 (libraries)文件包含支持位于/bin和/sbin下的二进制文件的库文件。bin和sbin的大集合 `./tmp` 临时(temporary)文件包含系统和用户创建的临时文件。当系统重启时, 这个目录下的文件将都被删除慢存储的闪存文件

`./var` 经常变化的(variable)文件, 诸如日志或数据库等。代表变量文件。在这个目录下可以找到内容可能增长的文件

`./lost+found` 在根 (/) 目录下提供一个遗失+查找(lost+found) 系统.必须在root用户下才可以查看当前目录下的内容。

6.解压与打包文件

`tar [选项] xxx.tar [打包文件内容]` 打包 `tar -cvf xxx.tar [目标文件]` 解包 `tar -xvf xxx.tar -C [目标目录]` 追加 `tar -rvf xxx.tar [目标文件]` 打包并压缩 `tar -czvf xxx.tar.gz [目标文件]` 解包 `tar -xvf xxx.tar.gz -C [目标文件]`

`-c` 创建新的归档文件 `-x` 提取归档文件的内容 `-t` 列出归档文件的内容 `-r` 追加文件到归档文件 `-z` 通过gzip进行压缩/解压缩 `-v` 显示详细信息 `-f` 指定归档文件的名称 `-p` 保留原始文件的权限

所有参数选项后都需+`f`,参数连用，如`-xvf`

python程序设计

名称	举例
字符串	'Joker'
整数	2
浮点数	2.0
变量	animal_names
列表	['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
字典	{'cats': 2, 'dogs': 5, 'horses': 1, 'snakes': 0}

举例	用途
<code>type('Joker')</code>	返回 'Joker' 的对象类型
<code>dir('Joker')</code>	返回一个列表，给出对象 'Joker' 可以做的所有事情 (方法和属性)
<code>help('Joker'.strip)</code>	返回给定方法 (在本例中是 <code>strip</code> ) 的说明文档，以便我们更好地了解如何使用它

1.python语言基础(2,3,4,5章为人工智能应用案例)

1.1.基础语法

`print(type('nmsl'))`输出数据类型<class 'str'> 不同数据类型的加减乘除。其输出不同，如'1'+'2'='12',1+1=2

引入模块的方式:`import module_name`,for instance`import math` 查看模块内容:`dir(module_name)`,for instance`dir(math)` 查看帮助:`help(math.sin)`,即查看math模块的sin功能

1.2.关系运算符&逻辑运算符



关系运算符	含义	举例
==	等于 ( equal )	10 == 20 is false
!= , <>	不等于 ( not equal )	10 != 20 is true
>	大于 ( greater )	10 > 20 is false
<	小于 ( less )	10 < 20 is true
>=	大于等于 ( greater or equal )	10 >= 20 is false
<=	小于等于 ( less or equal )	10 <= 20 is true

e.g.

```
a=(10==2)
print(a)
```

输出False

与或非:

and(与) e.g. True and False == False or(或) e.g. True or False == True not(非) e.g. not True == False

运算符优先级:

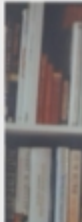
- ❖ 括号 : ( )
- ❖ 一元运算 : + , -
- ❖ 幂次 : \*\*
- ❖ 算术运算 : \* , / , % , //
- ❖ 算术运算 : + , -
- ❖ 比较运算 : == , != , <> <= >=
- ❖ 逻辑非 : not
- ❖ 逻辑与 : and
- ❖ 逻辑或 : or
- ❖ 赋值运算 : = , \*= , /= , += , -= , %= , //=

规则1 :

自上而下

括号最高

逻辑最低



```
a=1+2*2 #6 or 5
print(a) #输出为5, 乘法优先级比加法高, 一元运算与算数运算不同
```

### 1.3.标准键盘输入

`raw_input`函数: 可读取键盘输入, 将所有输入作为字符串看待(此为python2.X的函数, 新版为)  
e.g.`raw_input([prompt])`,`[prompt]`是提示符

python3.X修正版为

```
a=float(input("nmsl\n")) #标准形式
print(a)
```

常用转义符:

转义字符	描述
<code>\</code> (在行尾时)	续行符
<code>\\</code>	反斜杠符号
<code>'\'</code>	单引号
<code>'\"'</code>	双引号
<code>\a</code>	响铃
<code>\b</code>	退格(Backspace)
<code>\000</code>	空
<code>\n</code>	换行
<code>\v</code>	纵向制表符
<code>\t</code>	横向制表符
<code>\r</code>	回车
<code>\f</code>	换页
<code>\oyy</code>	八进制数, yy 代表的字符, 例如: <code>\o12</code> 代表换行, 其中 o 是字母, 不是数字 0。
<code>\xyy</code>	十六进制数, yy代表的字符, 例如: <code>\x0a</code> 代表换行
<code>\other</code>	其它的字符以普通格式输出

end语句的用法:

```
print('张三',end=': ')
print('今天天气很好')           #输出 张三: 今天天气很好
```

end表示print输出的语句以什么结束。这里就是输出张三后接冒号: , 然后接"今天天气很好"。

format语句结合使用:



```
name = '张三'
score = 100
print('{}的成绩是{}'.format(name,score)) #输出 张三的成绩是100
```

这里的{}对应format语句里的变量，几个{}就是几个变量，输出时直接读取变量的值。

## 1.4.选择结构(e.g. if语句)

if的标准形式:

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

e.g.

```
a=float(input("nmsl\n"))
if a>10:
    print(a)
elif a==10:
    print(a+10)
else: print("nmsl")
```

elif相当于else: if

## 1.5.循环结构(while and for)

```
while 判断条件(condition):
    执行语句(statements).....
```

e.g.

```
a=0
while a<5:
    print("nmsl")
    a+=1
```

break结束当前循环体 continue结束当次循环

```
a=0
while a<5:
    print("nmsl")
    a+=1
    if a==3:
        break    #输出结果变为只循环三次
```

```
for variable in object:
    <statements>
else:
    <statements>
```

依次遍历对象object中的每一个元素(object一般是一个数列),并将其赋值给变量variable,然后执行循环体语句.

range函数,range(start,stop,step),step可空,默认为1, 形成一个等差数列

e.g.

```
range(2,10)->[2,3,4,5,6,7,8,9] 最后一位数不会取到 range(2,10,3)->[2,5,8] range(10,2,-1)->
[10,9,8,7,6,5,4,3]
```

for一般应用于已知循环范围的情况 while则用于不知循环何时终止的情况

## 1.6.\*\*\*\*函数\*\*\*\*

函数定义:

```
def print_sum(start,stop):    #关键字 函数名(参数):
    result=0

    for i in range(start,stop+1):    #函数体&函数语句
        result+=i
    print('sum is',result)
```

函数体使用print\_sum(start,stop)直接调用即可

e.g.(此为一等差数列求和函数)

```
def print_sum(start,stop):    #关键字 函数名(参数):
    result=0
    for i in range(start,stop+1):
        result+=i
    print('sum is',result)
print_sum(1, 10)
```

函数内为局部变量，不会改变整个程序所使用的变量(全局变量),这点与c语言一致(变量名相同也互不影响)

**递归函数** 与c语言的递归调用类同，这里以阶乘的函数举例:

```
def f(n):
    if n==0 or n==1:
        return 1
    else:
        return p*p(n-1)
```

1.7.字符串基础

字符串长度len()函数

```
a='nmsl'
print(len(a))
#输出4
```

字符串的加法(+)是拼接的，乘法(\*)为重复的

成员运算符in函数,返回True or False

```
a='nmsl'
print(len(a))
#输出4
b=('a' in a)
print(b)    #此条输出False
```

for可循环枚举每个字符

```
a='nmsl'
for char in a:
    print(char)    #当然也可以用作判断字符对错
```

字符串索引(字符串本身是个char型的数组,c语言理解):

- ❖ 字符串中每个字符都有一个索引值（下标）
- ❖ 索引从0（前向）或-1（后向）开始

forward index	0	1	2	3	4	5	6	7	8	9	10
	h	e	l	l	o		w	o	r	l	d
backward index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

切片概念(选择字符串的子序列):

```
a='Hello world!'
print(a[6:10])    #输出'worl'
```

[start:end],start和end可以空其一,前者空代表从0序开始, 后者空代表到末序结束

计数参数:

```
a='Hello world!'
print(a[::-1])    #输出'!dlrow olleH'
```

-1位上的称conutBy, 默认为1, -1时可获得逆字符串, 2表示隔一个字符获取一个字符, 输出为hlowrd, 3表示隔两个字符获取一个字符, 输出为Hlw1

字符串是不可改变的, 诸如a[1]='m'的代码不会生效, 只会报错

## 1.8.字符串方法

.replace(old,new)语句:new替代字符串中所有的old

```
a='Hello world!'
a=a.replace('l','k')
print(a)    #输出'Hekko workd! '
#也可以使用print(a.replace('l','k'))输出而不改变a
```

.find('字符'):输出'字符'第一次出现的下标

.split():将字符串切割成若干子串(在空格位置)

```
a='nmsl ssc'
print(a.split())
```

## 1.9.列表与元组

## ❖ 列表内容是可变的

- `my_list[0] = 'a'`
- `my_list[0 : 2] = [1.2, 3, 5.6]`
- `my_list.append()`, `my_list.extend()` #追加元素
- `my_list.insert()` #任意位置插入元素
- `my_list.pop()`, `my_list.remove()` #删除元素
- `my_list.sort()` #排序
- `my_list.reverse()` #逆序

```
list1 = ['Google', 'Runoob', 1997, 2000]
list2 = [1, 2, 3, 4, 5]
list3 = ["a", "b", "c", "d"]
list4 = ['red', 'green', 'blue', 'yellow', 'white', 'black']
```

**追加元素:**append是将输入对象添加到列表末尾；而extend则是将输入对象的可迭代对象添加到列表末尾。

e.g.

```
my_list = ['abc', '123']
my_list.extend('456') # 字符串
my_list.extend(['alice', 18]) # 列表
my_list.extend({'alice': 18, 'amy': 22}) # 字典（默认key）
print('extend输出结果:', my_list)

my_list = ['abc', '123']
my_list.append('456') # 字符串
my_list.append(['alice', 18]) # 列表
my_list.append({'alice': 18, 'amy': 22}) # 字典（默认key）
print('append输出结果:', my_list)
```

```
extend输出结果: ['abc', '123', '4', '5', '6', 'alice', 18, 'alice', 'amy']
append输出结果: ['abc', '123', '456', ['alice', 18], {'alice': 18, 'amy': 22}]
```

**任意位置插入元素:**`list.insert(index, obj)`

index: 对象obj需要插入的索引位置。 obj: 要插入列表中的对象。 e.g.

```
list=['n',6,888,'nmsl']
list.insert(2,'nb') 插入第三位,之后整体后置
print(list)
```

### 删除元素: `.remove`, `.pop` and `del`

`remove(item)`方法是直接对可变序中的元素进行检索删除, 返回的是删除后的列表,不返回删除值 (返回 `None`) -> 删除第一个被检索到的item, 其余不管

`pop(index)`方法是对可变序列中元素下标进行检索删除, 返回删除值, 效果就是删除`list[index]`

`del(list[index])`方法是对可变序列中元素下边进行检索删除, 不返回删除值

### 排序: `list.sort(cmp=None, key=None, reverse=False)`

`cmp` -- 可选参数, 如果指定了该参数会使用该参数的方法进行排序。 `key` -- 主要是用来进行比较的元素, 只有一个参数, 具体的函数的参数就是取自于可迭代对象中, 指定可迭代对象中的一个元素来进行排序。 `reverse` -- 排序规则, `reverse = True` 降序, `reverse = False` 升序 (默认) 。

### 逆序:

`.sort(reverse=True)`与`.reverse`效果一致

### 查找方法 `index()`方法语法:

```
list.index(x, start, end)
```

参数 `x`-- 查找的对象。 `start`-- 可选, 查找的起始位置。 `end`-- 可选, 查找的结束位置。

`len(list)` 列表元素个数 `max(list)` 返回列表元素最大值 `min(list)` 返回列表元素最小值 `list(seq)` 将元组转换为列表

### 元组略

## 1.10.字典与集合

definition:`dict={'a':1,'b':2,'c':3}` 访问字典:`dict['a']`,输出2 增长字典对:`dict['d']=4`

`len(dict)`可输出字典中键-对的数量 `e in dict`查询'e'是否为字典中的键

```
dict={'a':1,'b':2,'c':3}
k=('d' in dict)
print(k)    #输出False
```

### 更多方法:

`dict.items()` -> 全部的键-值对 (`print(dict.items())`)将以列表形式输出键-值对)

`dict.keys()` -> 全部的键 (输出同理)

`dict.values()` -> 全部的值 (输出同理)



dict.clear() -> 清空字典

```
dict={'a':1,'b':2,'c':3}
dict.clear()
print(dict)    #输出空字典 '{}'
```

案例:for循环遍历字典时,用for i in dict里的i['name']表示遍历的字典中的其中一部分

```
result = {
    'city_num': 4,
    'city_list': [
        {'city_name': '广州', 'city_area': 7435, 'population': 1270.08},
        {'city_name': '深圳', 'city_area': 2050, 'population': 1035.79},
        {'city_name': '厦门', 'city_area': 1569, 'population': 353.13},
        {'city_name': '青岛', 'city_area': 11026, 'population': 871.51},
    ]
}
n=0
a=input("请输入要查找的城市:")
for i in result['city_list']:
    if i['city_name']==a:
        print("要查找的城市是",i['city_name'],i['city_area'],i['population'])
        n=1
if n==0:
    print("没有这个城市")
```

and

# 设置一个列表sList, 用于存储工资的数据。列表sList中存储7个员工的工资数据,  
# 每个员工的工资数据保存在一个字典中; 每个字典均包括员工编号 ('number')、  
# 姓名 ('name')、基本工资 ('salary') 和津贴 ('subsidy')。  
# 根据程序中的以下数据, 编程实现:  
# (1) 查询: 输入要查询的员工编号, 若有该编号则打印输出对应的姓名、总工资, 否则输出“无此员工!”。  
# (2) 计算: 最高工资、最低工资 ('salary'), 及其对应的员工姓名。

```
sList=[{'number':'001', 'name':'张大刚','salary':5200,'subsidy':2600},
        {'number':'002', 'name':'包慧慧','salary':6850,'subsidy':3500},
        {'number':'003', 'name':'沈谦','salary':3980,'subsidy':1800},
        {'number':'004', 'name':'谭小琴','salary':4770,'subsidy':2800},
        {'number':'005', 'name':'罗飞虎','salary':6100,'subsidy':3100},
        {'number':'006', 'name':'肖浩彤','salary':5600,'subsidy':2600},
        {'number':'007', 'name':'葛翠珊','salary':4330,'subsidy':2100},
    ]
```

```

# (1) 查询：输入要查询的员工编号，若有该编号则打印输出对应的姓名、总工资，否则输出“无此员工！”
a=input("员工编号:")
n=0
for i in sList:
    if a==i['number']:
        print(i['name'],i['salary'],i['subsidy'])
        n=1

if n==0:print('无此员工')

# (2) 计算：最高工资、最低工资 ('salary')，及其对应的员工姓名
all1=0
all2=0
i=1
tmp=0
for n in sList:
    if i==1:
        all1=n['salary']
        all2=n['salary']
        i=2

    elif i==2:
        if all1<n['salary']:
            all1=n['salary']
            if all2>n['salary']:
                all2=n['salary']
        elif all1>n['salary']:
            if all2 > n['salary']:
                all2 = n['salary']

print("最高工资:",all1,'\n')
print("最低工资:",all2)

```

需要搜索的字词非字典的‘键’时，不可使用‘a’ in dict这样的语法，这样的的语法是用于判断字典中是否存在‘a’这个键的。

**集合:**

## 2.数据处理(废案)

```

from decimal import getcontext, Decimal  #(1)
getcontext().prec = 1  #(2)
Decimal(0.1) + Decimal(0.2)  #(3)

```

(1)从 decimal 模块中导入 getcontext 和 Decimal。(2)将舍入精度设定为一位小数。decimal 模块将大部分的舍入和精度设置保存在默认的上下文 (context) 中。本行代码将上下文的精度改成只保留一位小数。(3)对两个小数 (一个值为 0.1，一个值为 0.2) 求和。

列表e.g. `x=['a','bb','ccc']` and `[a,bb,ccc]` (前者为字符串列表, 后者为变量列表)

字典e.g. `x={'a':1,'bb':2,'ccc':3}`

```
cat_names = ['Walter', 'Ra']      #(1)
dog_names = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
horse_names = ['Mr. Ed']
animal_names = {
    'cats': cat_names,             #(2)
    'dogs': dog_names,
    'horses': horse_names
}
```

(1)这一行代码将变量 `cat_names` 定义为猫咪名字的列表(字符串列表)。 (2)这一行代码使用变量 `cat_names`, 传递猫咪名字的列表作为字典中键 `'cats'` 对应的值。

`x.strip()`:去除多余空格 `x.upper()`:转换为大写字母

**字符串&列表加法** e.g. `'This is ' + 'awesome.'` `['Joker', 'Simon', 'Ellie'] + ['Lishka', 'Turtle']`

**列表元素增减** `x.append('元素')`:增加元素 `x.remove('元素')`:删除元素

### 字典方法

`x={} x['新元素']=1`: 增加新元素并赋值 `x.keys()`:返回字典所有键(元素名)

`if y=x[a],print(y)=a`所代表的值

## 3.数据有关工具(废案)

**type 返回数据类型** `type('20011')`: str `type(20011)`: int

**dir 返回一个内置方法与属性的列表&x.split** `dir(1)`:返回关于此类型可以参与的运算 如add,abs(加法、绝对值等)

`x='cat,dog,horse'`:存在一个内有逗号分隔的字符串 `x='cat,dog,horse'.split(',')`:以','分割形成列表 (`x.split('分隔符')`)

`x.reverse` `x.sort` (待补充)

## 4.turtle 绘图(动画)(废案)

```
import turtle
turtle.forward('距离') :动画箭头默认向右
turtle.circle('圆半径')
turtle.done() :运动结束, 界面不关闭
```

五角星e.g.1.

```
import turtle

side_num = 36 #角数, 此为36角星
side_length = 400
side_angle = 180 - 180/side_num
turtle.pensize(2) #笔尺寸
turtle.speed(10) #笔速

for side in range(side_num): #range(10)默认为:0,1,2...8,9
    if side % 3 == 0:
        side_color = 'blue' #if改变笔刷颜色
    elif side % 3 == 1:
        side_color = 'red'
    else:
        side_color = 'green'

    turtle.color(side_color)

    turtle.forward(side_length)

    turtle.left(side_angle)

turtle.done()
```

`turtle.left(角度):向左偏转'角度'`

五角星e.g.2.

```
import turtle

i = 0
side_num = 5
side_lengt = 200
side_angle = 180 - 180/side_num

turtle.pencolor('blue')
turtle.pensize(5)
#turtle.speed(5)
turtle.speed('slow')

while i < side_num: #while起到类似c语言的for的作用
    turtle.forward(side_lengt)
    turtle.left(side_angle)
    i += 1

turtle.done()
```

多角五彩星e.g.

```
import turtle

fill_color= input('请输入角星的填充颜色(gold,yellow,pink)?')
side_num = int(input('想画几角星? '))
side_length = int(input('想画角星的边长是多少? '))
side_angle = 180 - 180/side_num
turtle.pensize(2)
turtle.speed(10)

turtle.begin_fill()
for side in range(side_num):
    if side % 3 == 0:
        side_color = 'blue'
    elif side % 3 == 1:
        side_color = 'red'
    else:
        side_color = 'green'
    turtle.color(side_color,fill_color)
    turtle.forward(side_length)
    turtle.left(side_angle)
turtle.end_fill()

turtle.done()
```

## 5.文件快速处理

```
a = len('hello') #判断字符长度
print(a)
b = len('深圳职业技术学院')
print(b)
c = len(['赵','钱','孙','李']) #判断列表元素个数
print(c)

d = str(123.5)
print(d + '6') #字符串加法 'yuan'+ 'shen'='yuanshen'

a = 'python'.endswith('on') #判断末尾是否为'on'，是则回True，否则回False
print(a)
'''

import os
a = os.getcwd()
print(a)

b = os.mkdir('new')
print(b)
```

## 6.os模块

**os.sep** : 主要用于系统路径的分隔符 :

Windows系统通过是“\\”，Linux类系统如Ubuntu的分隔符是“/”，而苹果Mac OS系统中是

**os.name** : 指示你正在使用的工作平台。

比如对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'。

**os.getenv(环境变量名称)** : 读取环境变量

**os.getcwd()** : 获取当前的路径。

<https://blog.csdn.net/xxlovesht>

```
\
nt
C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program F
E:\??\??\??\??\??\??\day7\day7
```

<https://blog.csdn.net/xxlovesht>

## 6.1.目录基本操作相关

### 目录操作 - 增删改查

**os.listdir()** : 返回指定目录下的所有文件和目录名

**os.mkdir()** : 创建一个目录。只创建一个目录文件。

**os.rmdir()** : 删除一个空目录。若目录中有文件则无法删除。

**os.makedirs(dirname)** : 可以生成多层递归目录。如果目录全部存在，则创建目录失败。

**os.removedirs(dirname)** : 可以删除多层递归的空目录，若目录中有文件则无法删除。

**os.chdir()** : 改变当前目录，到指定目录中。

**os.rename()** : 重命名目录名或者文件名。重命名后的文件名已存在，则重命名失败。

**os.unlink(path)** 删除文件(path -- 删除的文件路径),其与**remove(path)**作用相同**shutil模块中的rmtree()**可以递归彻底删除非空文件夹

**os.replace(src, dst, \*, src\_dir\_fd=None, dst\_dir\_fd=None)** 重命名文件夹与目录(与**os.rename()**同用处) src -- 源文件或目录，如果该目录文件不存在会引发错误 **FileNotFoundError**。 dst -- 重命名后的文件或目录，如果已存在，会直接替换。 src\_dir\_fd -- 相对目录描述符的路径 dst\_dir\_fd -- 相对目录描述符的路径。

以上操作一般需预设一个关于路径的字符串 **dir** (对目录操作而言):

```
import os
b='C:\\Users\\曹建邦\\Desktop\\vscode'
a=os.listdir(b)
print(a)
```

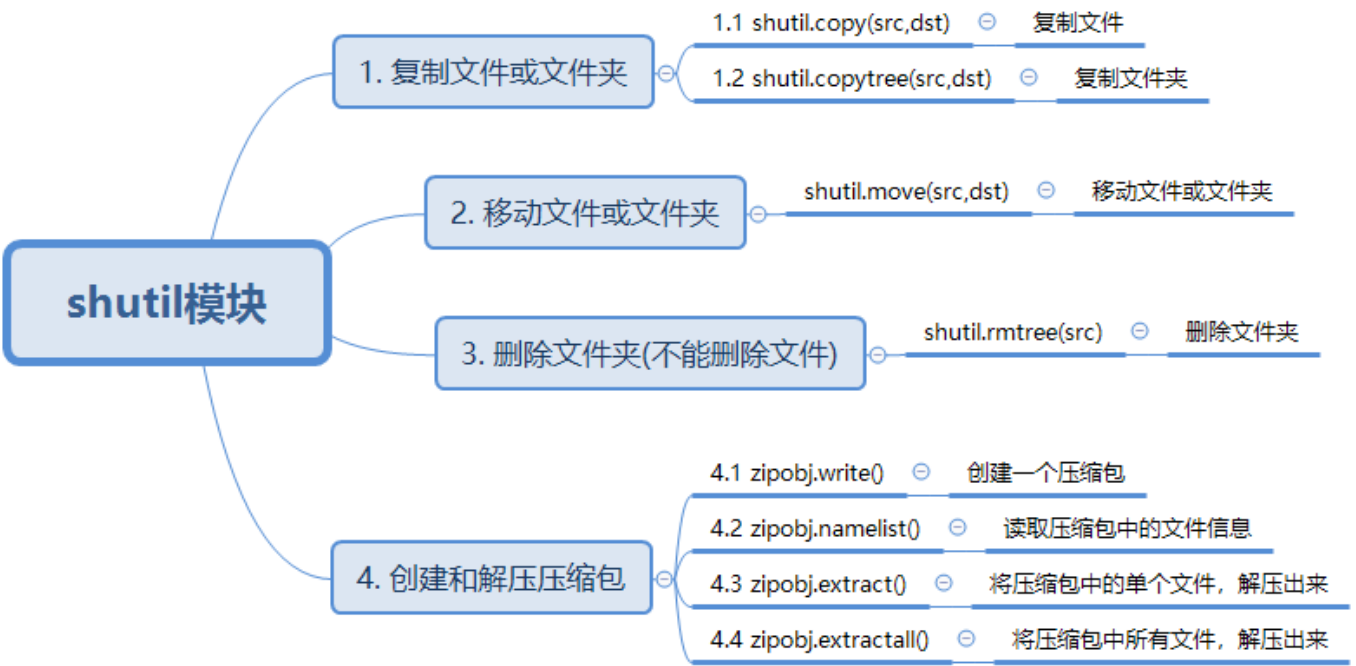


`chdir('path')`,其中`path`即`dir`，其为关于路径的字符串，其中路径的间隔符是`//`

for `os.rename(src, dst)`: `src` -- 要修改的文件或目录名 `dst` -- 修改后的文件或目录名

6.2.文件目录方法(path模块)

6.3.shutil模块(os的扩充)&zipfile模块



`src`表示源文件，`dst`表示目标文件夹 e.g.

```
import shutil
import os
src='C:\\Users\\曹建邦\\Desktop\\pycharm\\.venv\\114.py'
dst='C:\\Users\\曹建邦\\Desktop'
shutil.copy(src,dst)
```

压缩包:

```
import zipfile
import os
file_list = os.listdir(os.getcwd())
# 将上述所有文件, 进行打包, 使用“w”
with zipfile.ZipFile(r"我创建的压缩包.zip", "w") as zipobj:
    for file in file_list:
        zipobj.write(file)
```

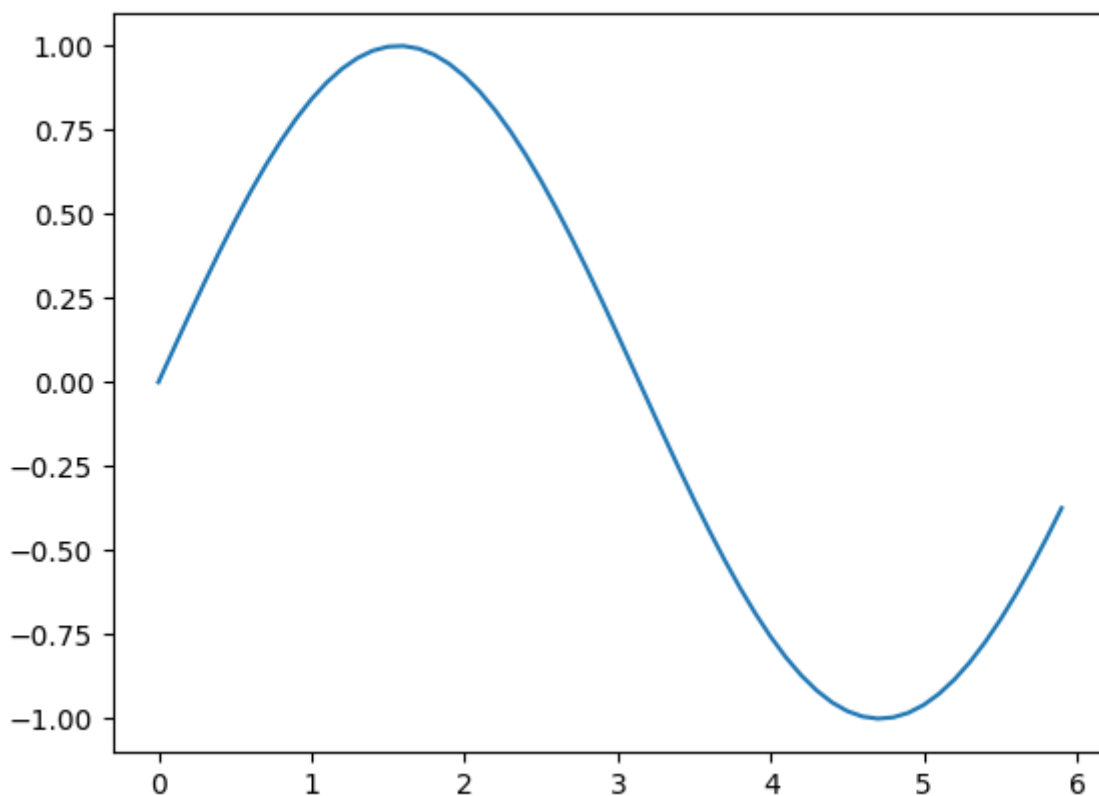
7.数据处理(numpy&matplotlib)

numpy&matplotlib模块(from...import \*)

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

x = np.arange(0, 6, 0.1)    #0, 0.1, 0.2, ... , 5.9, 6.0
y = np.sin(x)

plt.plot(x,y)
plt.show()
```



### 一般性前缀:

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

**import相关注解** `import` 模块 :导入一个模块 `from...import` :导入了一个模块中的一个函数 `from...import *` :导入一个模块的所有函数 e.g.(1)情况下调用`sin()`函数需使用`np.sin()`,(3)or(2)则使用`sin()`

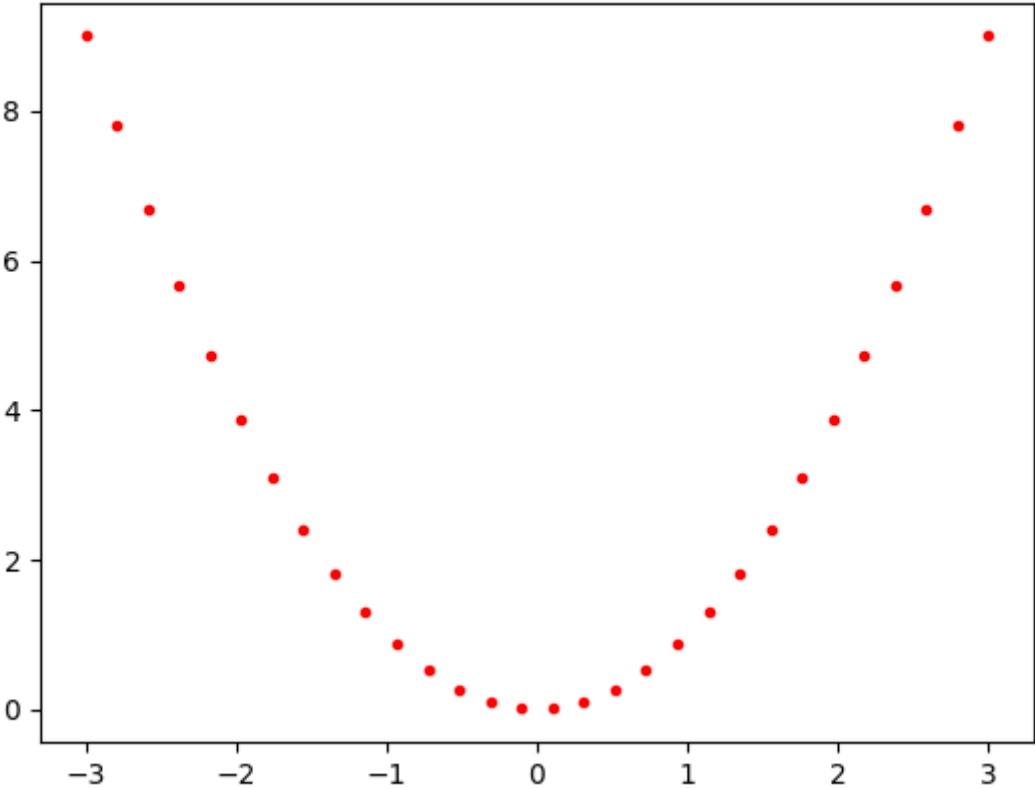
### 7.1.初等函数图像

```
import numpy as np
a = np.array([1,2,3]) #一行一列数组
print (a)
```

`np.linspace(start = 0, stop = 100, num = 5)`:一般作为`plt.plot`的显示空间,num为start-stop之间的点数

特殊线条:

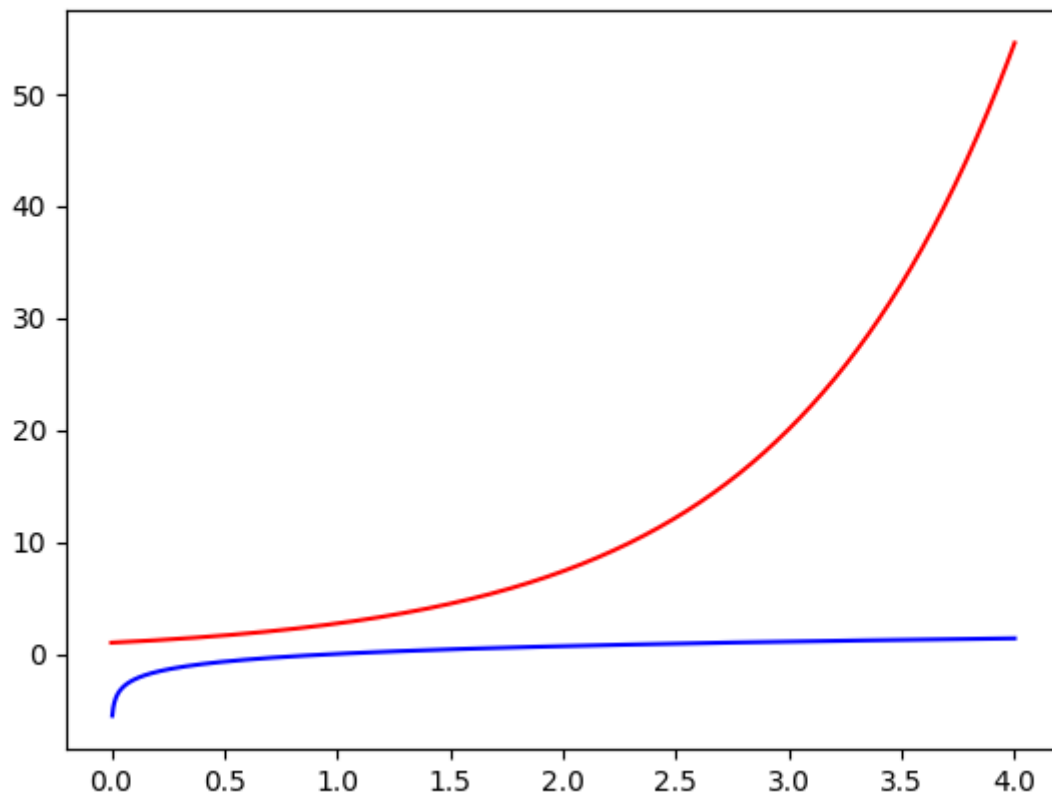
```
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y, 'r.')
show()
```



符号	'-', 'o', '^', 'v', '<', '>', 's', '+', 'x', 'D', 'd', '1', '2', '3', '4', 'h', 'H', 'p', 'l', '_'
颜色	b(蓝色), g(绿色), r(红色), c(青色), m(品红), y(黄色), k(黑色), w(白色)

多重曲线与异色案例

```
from pylab import *  
x = linspace(0, 4, 1000)  
y1 = log(x)  
y2 = exp(x)  
plot(x, y1, 'b-')  
plot(x, y2, 'r-')  
show()
```



可补

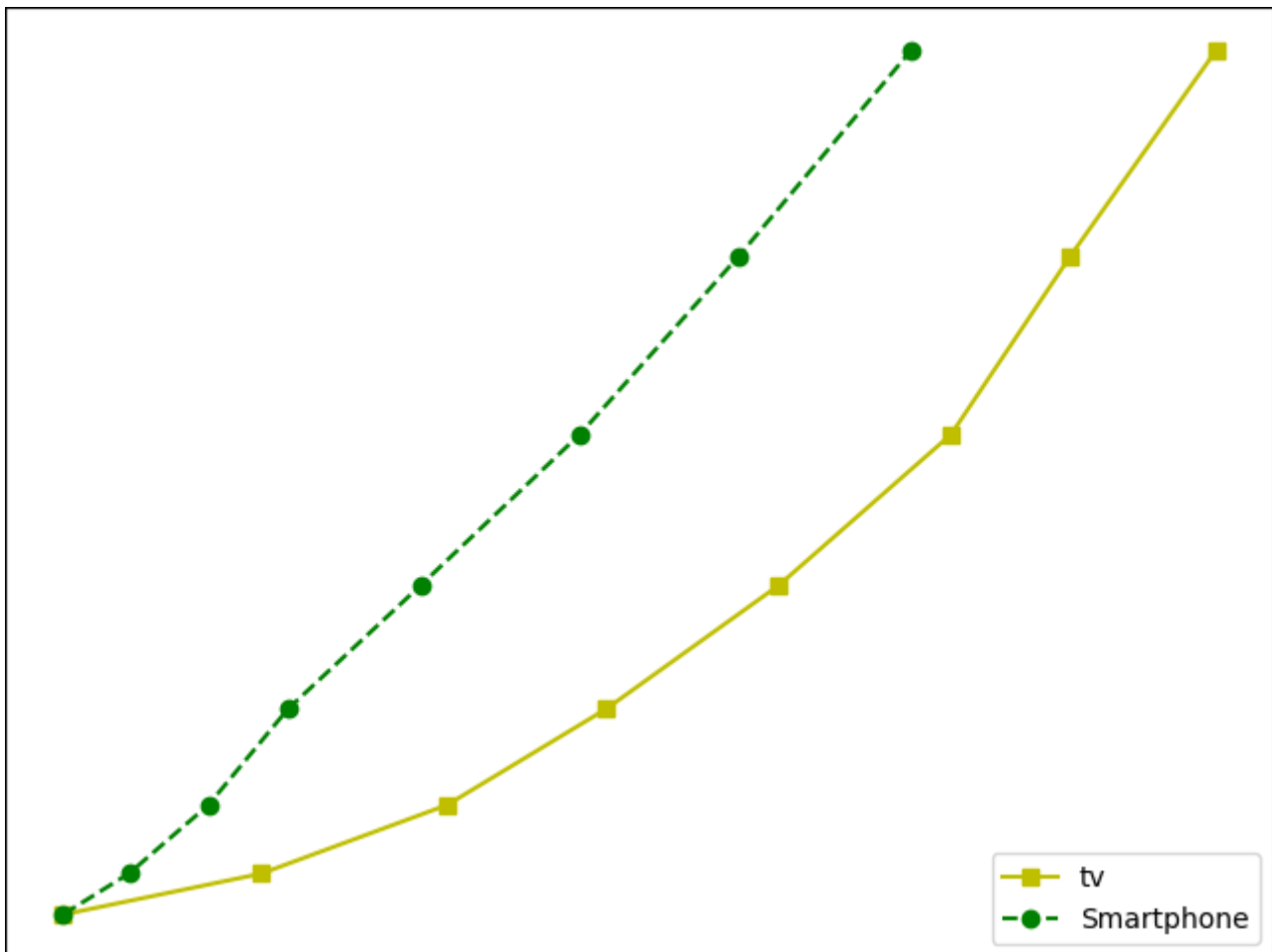
充:

```
xlabel('x')  
ylabel('y')  
title('log(x) and exp(x)')
```

## 7.2.画布图像(axes)

参数	说明
figsize	指定画布的大小, (宽度,高度), 单位为英寸。
dpi	指定绘图对象的分辨率, 即每英寸多少个像素, 默认值为80。
facecolor	背景颜色。
dgecolor	边框颜色。
frameon	是否显示边框。

```
import matplotlib.pyplot as plt
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
#使用简写的形式color/标记符/线型
l1 = ax.plot(x1,y,'ys-')
l2 = ax.plot(x2,y,'go--')
ax.legend(labels = ('tv', 'Smartphone'), loc = 'lower right') # legend placed at
lower right
ax.set_title("Advertisement effect on sales")
ax.set_xlabel('medium')
ax.set_ylabel('sales')
plt.show()
```



## 散点图

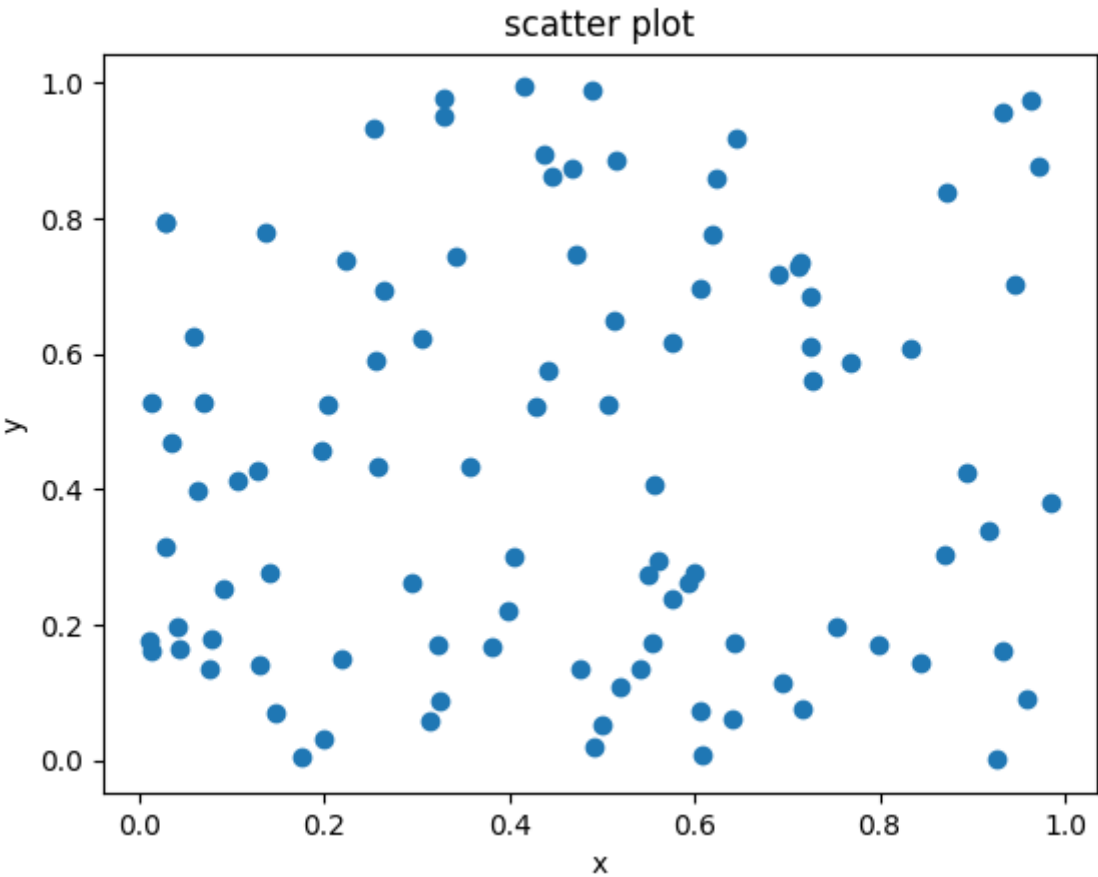
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(100)
y = np.random.rand(100)

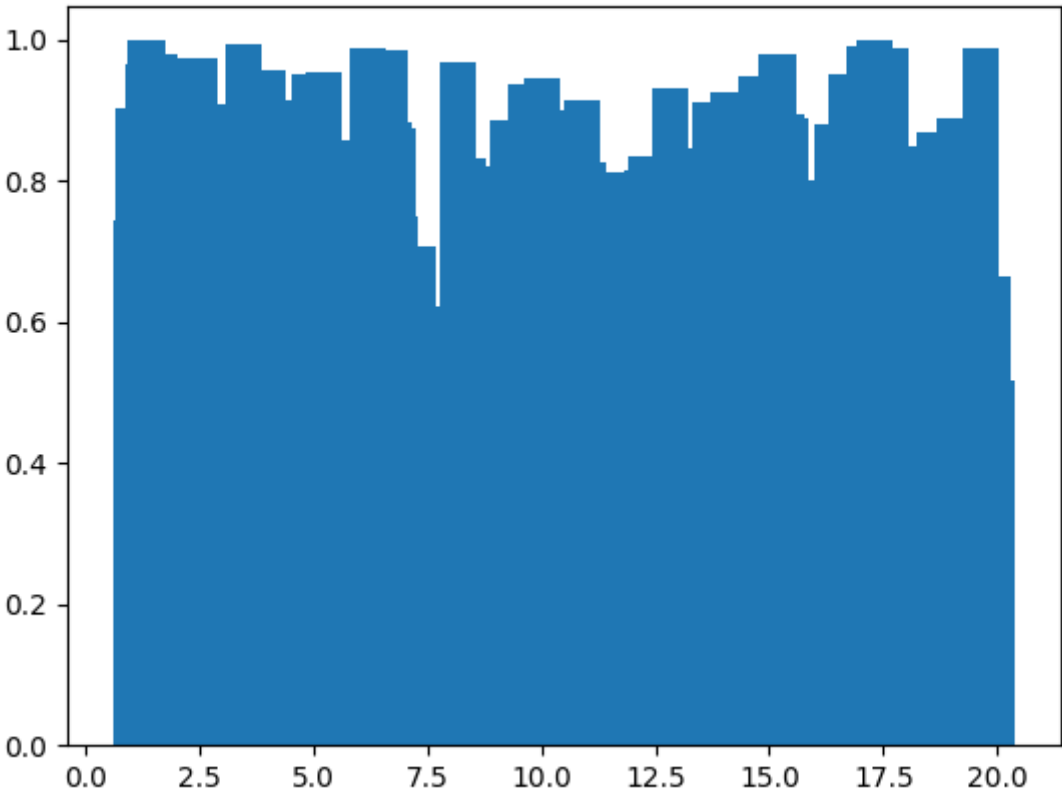
fig, ax = plt.subplots()
ax.scatter(x, y)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('scatter plot')

plt.show()
```

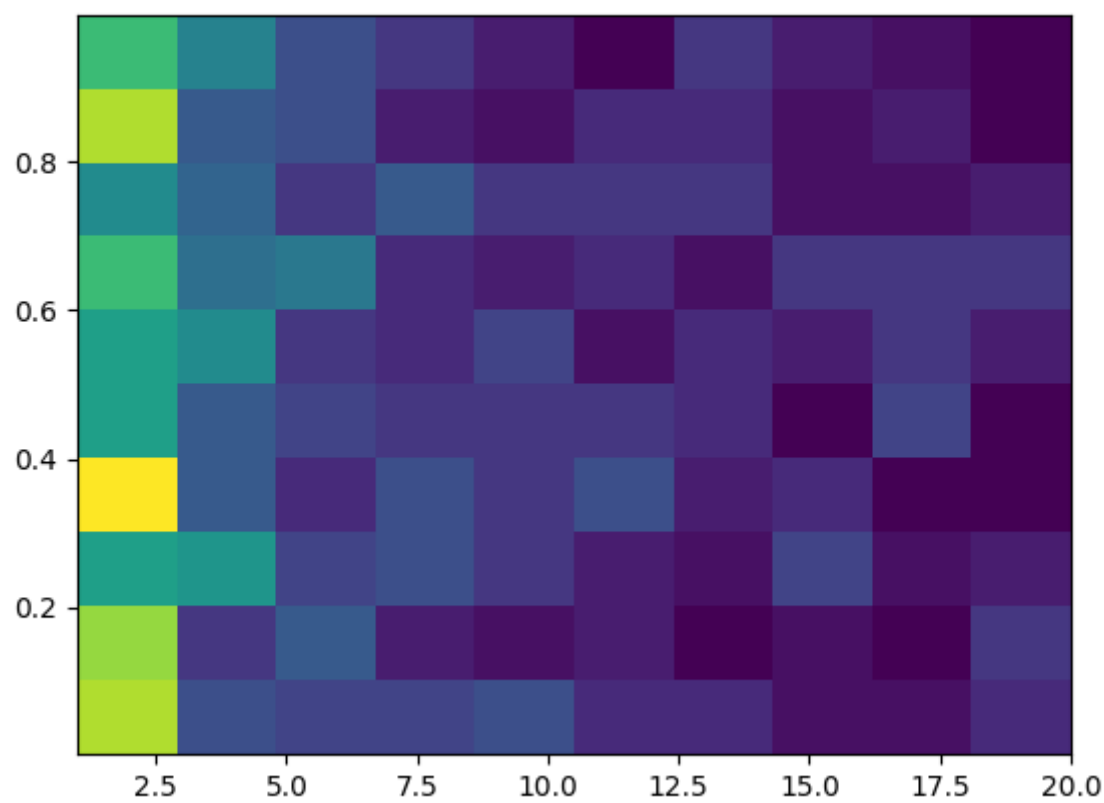




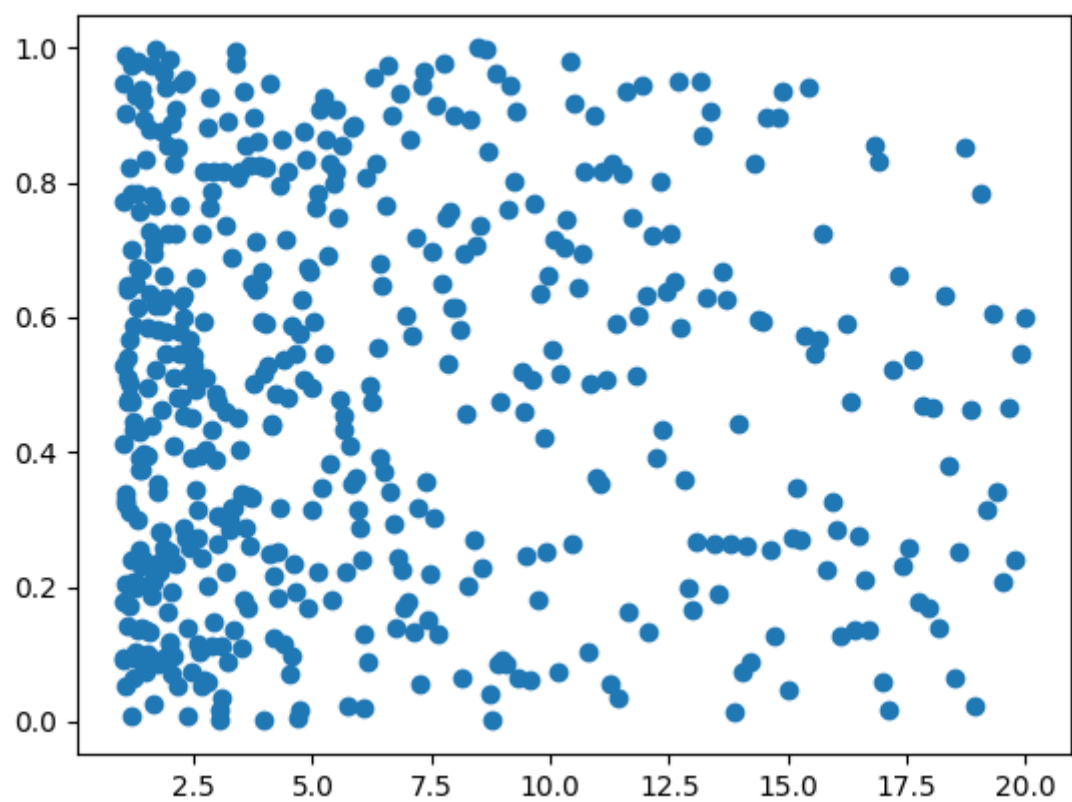
图形类型 (1) `bar(x,y)` 条图



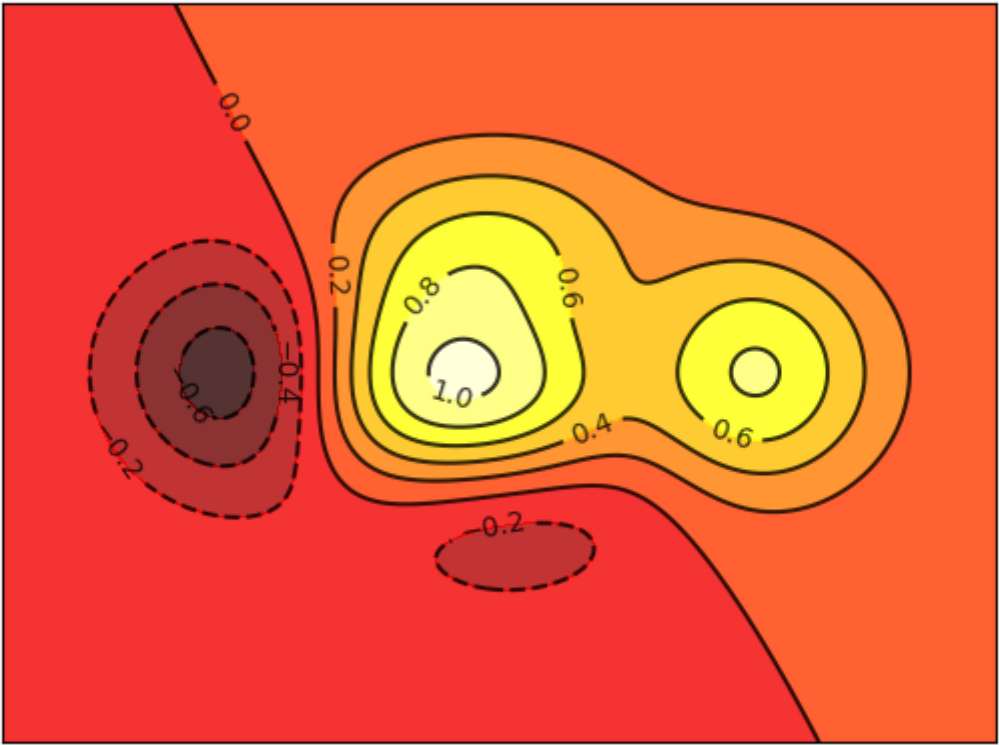
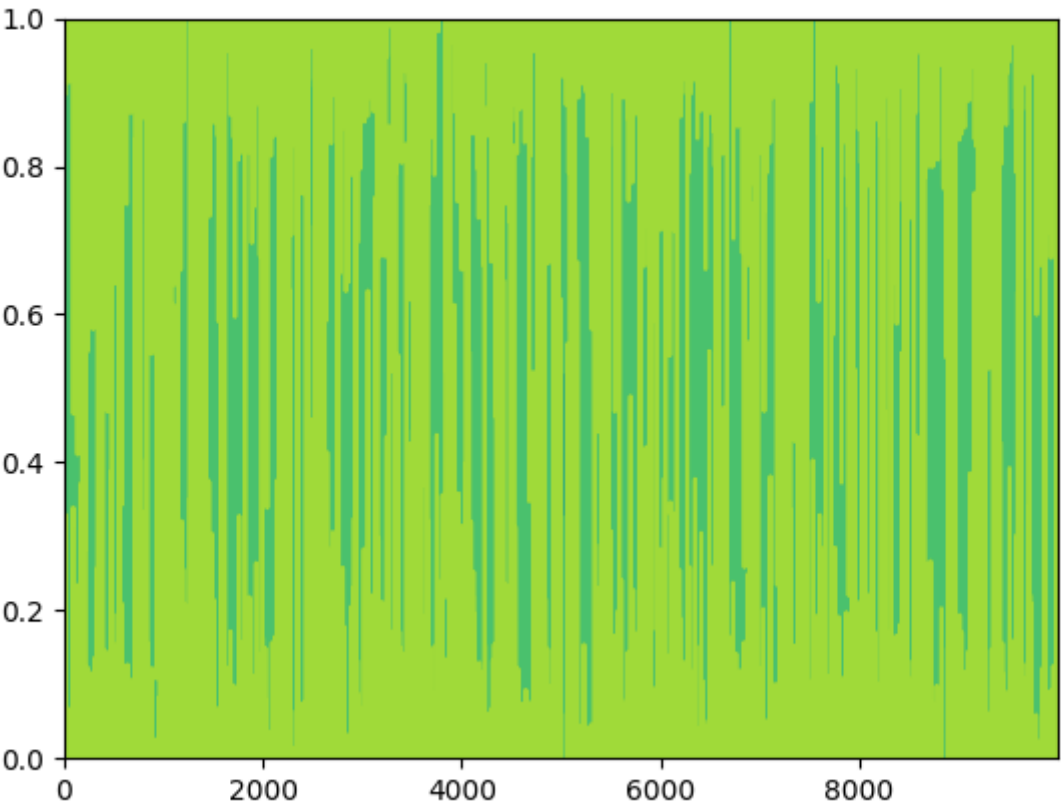
(2)`plt.hist2d(x,y)`色图



(3)`plt.scatter(x,y)`散点图



(4)`plt.contour([x,y])`要求双数据



7.3.随机函数(注解)

- (1) `random.random()`: 返回随机生成的一个浮点数, 范围在[0,1)之间
- (2) `random.uniform(a, b)`: 返回随机生成的一个浮点数, 范围在[a, b)之间
- (3) `np.random.rand(d0, d1, ..., dn)`: 返回一个或一组浮点数, 范围在[0, 1)之间
- (4) `np.random.normal(loc=a, scale=b, size=())`: 返回满足条件为均值=a, 标准差=b的正态分布 (高斯分布) 的概率密度随机数
- (5) `np.random.randn(d0, d1, ... dn)`: 返回标准正态分布(均值=0, 标准差=1)的概率密度随机数
- (6) `np.random.standard_normal(size=())`: 返回标准正态分布(均值=0, 标准差=1)的概率密度随机数
- (7) `random.sample(k)`: 从总体序列或集合中随机选取k个唯一的元素
- (8) `np.random.randint(a, b, size=(), dtype=int)`: 返回在范围在[a, b)中的随机整数 (含有重复值)
- (9) `random.randrange(a, b, step=c)`: 在指定范围内, 在指定的基数和步长值形成的集合中获取1个随机数值
- (10) `random.choice(x)`: 从指定的序列x中随机获取一个数据
- (11) `random.shuffle(x)`: 将一个列表x中的元素打乱, 随机排序 (俗称: 洗牌)
- (12) `random.seed()`: 设定随机种子

7.4.numpy模块

7.4.1.array(创建普通数组)

创建数组(元组亦可用):

```
numpy.array(object,dtype=None,copy=True,order=None,subok=False,ndmin=0)
```

序号	参数	描述说明
1	object	表示一个数组序列。
2	dtype	可选参数, 通过它可以更改数组的数据类型
3	copy	可选参数, 当数据源是ndarray时表示数组能否被复制, 默认是 True。
4	order	可选参数, 以哪种内存布局创建数组, 有 3 个可选值, 分别是 C(行序列)/F(列序列)/A(默认)。
5	ndmin	可选参数, 用于指定数组的维度。
6	subok	可选参数, 类型为bool值, 默认False。为True, 使用object的内部数据类型; False: 使用object数组的数据类型。

```
e.g.numpy.array([1,2,3],dtype=`float`)
```

**copy的注意事项** `copy`参数影响`a=np.array([1,2,3])`能否`b=a` `b=a`的情况下, 二者的内存地址相同, 其一改变另者亦改, 但若`a=np.array([1,2,3])`,`b`也`b=np.array(a)`,二者则为互不关联的数据,`b=a.copy()`有相同作

用

```
import numpy as np
a=np.array([1,2,3],ndmin=2)
b=a.copy()
print(a,'\t',b)
print(id(a),id(b))
```

迭代对象:`np.array(range(10))`将生成一个0-9的数组 生成器:`np.array([i**2 for i in range(10)])`将输出一个 $f(x)=x^2,x=0-9$ 的整数的离散数组

e.g.10以内偶数的数列 `np.array([i for i in range(10) if i%2 == 0])`

如果数组内有多种元素类型，输出的每个个体将变为内存占用最大的数据类型 def.二维数组:`np.array([1,2,3],[4,5,6])`

aka.`dtype`参数:若浮点型数组设置为整型，输出所有元素将只取元素的整数部分 and.`id(a)`可查看a的内存地址

对于`ndmin` `a=np.array([1,2,3],ndmin=2)`强行令a为二维数组 e.g.

```
import numpy as np
a=np.array([1,2,3],ndmin=2)
print(a)      #输出 [[1 2 3]]
print(a.ndmin)  #输出2
```

对于`subok` a为一个矩阵类型(非数组类型),`a1=np.array(a,subok=True)`,a1输出为原类型(复制副本又保持原类型),`a2=np.array(a)`,a2输出为数组类型.

7.4.2.arange、linspace and logspace(区间数组&等差等比数列)

区间数列: `numpy.arange(start,stop,step,dtype)`浮点导致的误差可用整型后期处理

序号	参数	描述说明
1	start	起始值，默认为0
2	stop	终止值（不包含）
3	step	步长，默认为1
4	dtype	返回ndarray的数据类型，如果没有提供，则会使用输入数据的类型。

**等差数列:** `numpy.linspace(start,stop,num=50,endpoint=True,retstep=False,dtype=None)`

序号	参数	描述说明
1	start	必填项，序列的起始值，
2	stop	必填项，序列的终止值，如果endpoint为true，该值包含于数列中
3	num	要生成的等步长的样本数量，默认为50
4	endpoint	该值为 true 时，数列中包含stop值，反之不包含，默认是True。
5	retstep	如果为 True 时，生成的数组中会显示间距，反之不显示。
6	dtype	ndarray 的数据类型

e.g.

```
import numpy as np
a=np.linspace(0,10,11)
print(a) #输出 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.] 浮点型
```

**retstep=True的案例:**

```
import numpy as np
a=np.linspace(0,10,10,retstep=True)
print(a)
```

输出:(间距有误差)

```
(array([ 0.          ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.          ]),
 np.float64(1.1111111111111112))
```

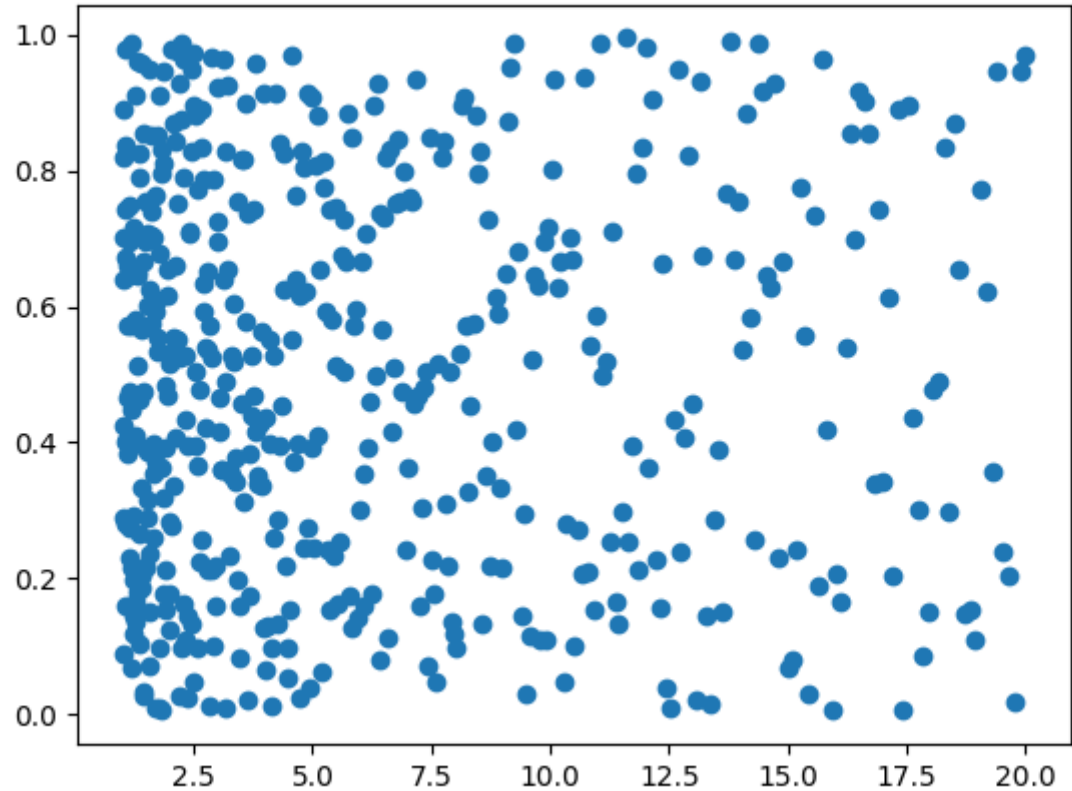


等比数列: `np.logspace(start,stop,num=50,endpoint=True,base=10.0,dtype=None)`

序号	参数	描述说明
1	start	必填项，序列的起始值，
2	stop	必填项，序列的终止值，如果endpoint为true，该值包含于数列中
3	num	要生成的等步长的样本数量，默认为50
4	endpoint	该值为 true 时，数列中包含stop值，反之不包含，默认是True。
5	base	对数 log 的底数
6	dtype	ndarray 的数据类型

e.g.

```
import matplotlib.pyplot as plt
x=np.logspace(0,1,500,base=20)
y=np.random.rand(500)
plt.scatter(x,y)
plt.show()
```



全零数组&全一数组: `numpy.zeros(shape,dtype=float,order='C')`

序号	参数	描述说明
1	shape	数组形状
2	dtype	数据类型, 可选

e.g.`np.zeros([2,2])` 输出两行两列的数组(全零数组默认输出浮点型) `np.zeros_like(a)` 函数可返回与a同形状和类型的全零数组

`numpy.ones()`使用参数与`numpy.zeros()`相同,且存在相同的`np.ones_like(a)`函数

7.4.3.numpy数组属性

属性	说明
<code>ndarray.ndim</code>	秩, 即轴的数量或维度的数量
<code>ndarray.shape</code>	数组的维度, 对于矩阵, n 行 m 列
<code>ndarray.size</code>	数组元素的总个数, 相当于 <code>.shape</code> 中 <code>n*m</code> 的值
<code>ndarray.dtype</code>	<code>ndarray</code> 对象的元素类型
<code>ndarray.itemsize</code>	<code>ndarray</code> 对象中每个元素的大小, 以字节为单位

维度调整函数:`.reshape`(返回调整维度后的副本, 而不改变原ndarray) e.g.

```
a=np.arange(20).reshape([4,5]) #改变的维数行*列元素数应当等于原生元素数, 否则error
```

调整维度:`.resize(a,new_shape)`与`.reshape`同理, 但若新数组大于原始数组, 则新数组将填充啊的重复副本; attention:此行为与`a.resize(new_shape,refcheck=False)`不同, 后者用零而不是重复的a填充(后者代码是直接对a操作)

```
import numpy as np
import matplotlib.pyplot as plt
a=np.array([1,2,3,4,5,6,7,8,9,10])
b=np.array([0])

aa=np.resize(a,[6,6])
a.resize((6,6),refcheck=False)
print(aa)
print(a)
```

输出:

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10  1  2]
 [ 3  4  5  6  7  8]
 [ 9 10  1  2  3  4]
 [ 5  6  7  8  9 10]
 [ 1  2  3  4  5  6]]
[[ 1  2  3  4  5  6]
 [ 7  8  9 10  0  0]
 [ 0  0  0  0  0  0]
 [ 0  0  0  0  0  0]
 [ 0  0  0  0  0  0]
 [ 0  0  0  0  0  0]]
```

## 7.5.数据的切片与索引

```
a=np.arange(10)
b=a[2:7:2]    #从索引2到索引7, 间隔为2
```

### 多维数组操作

# c语言程序设计

## 1.c语言标准库的补充以及注意事项

### 1.1.字符串处理注解

```
#include <string.h> //需加头文件
```

**strlen():测试字符串长度** str -- 要计算长度的字符串。strlen(str)=字符串长度

**strcpy():复制字符串** strcpy(dest,src) dest -- 指向用于存储复制内容的目标数组。 src -- 要复制的字符串。

**strcat():连接字符串** strcat(dest,src) dest -- 指向目标数组, 该数组包含了一个 C 字符串, 且足够容纳追加后的字符串。 src -- 指向要追加的字符串, 该字符串不会覆盖目标字符串。 e.g.

```
#include <stdio.h>
#include <string.h>

int main ()
{
    char src[50], dest[50];

    strcpy(src, "This is source");
```

```

strcpy(dest, "This is destination");

strcat(dest, src);

printf("最终的目标字符串:  |%s|", dest);

return(0);
}

```

**strcmp():比较字符串** `n=strcmp(str1, str2)` `n`=返回值 如果返回值小于 0, 则表示 `str1` 小于 `str2`。如果返回值大于 0, 则表示 `str1` 大于 `str2`。如果返回值等于 0, 则表示 `str1` 等于 `str2`。

**scanf("%[^\n]", str)正则用法** 1 ^表示"非", `[^\n]`表示读入换行字符就结束读入。这个是scanf的正则用法, 我们都知道scanf不能接收空格符, 一接受到空格就结束读入, 所以不能像gets()等函数一样接受一行字符串, 但是使用`%[^\n]`就可以读取一行, 直到碰到`\n`才结束读入。2 表示该输入项读入后不赋予任何变量, 即`scanf("%[^\n] %c")`表示跳过一行字符串。其中`%c`可以把`\n`吸收掉, 防止影响后续输入。

### strchr 用法

```

#include<iostream>
#include<cstring>
using namespace std;
int main()
{
    char str[] = "I love you.cn";
    char ch = '.';
    char *point;

    point = strchr(str, ch);

    printf("|%c| 之后的字符串是 -  |%s|\n", ch, point);
}

```

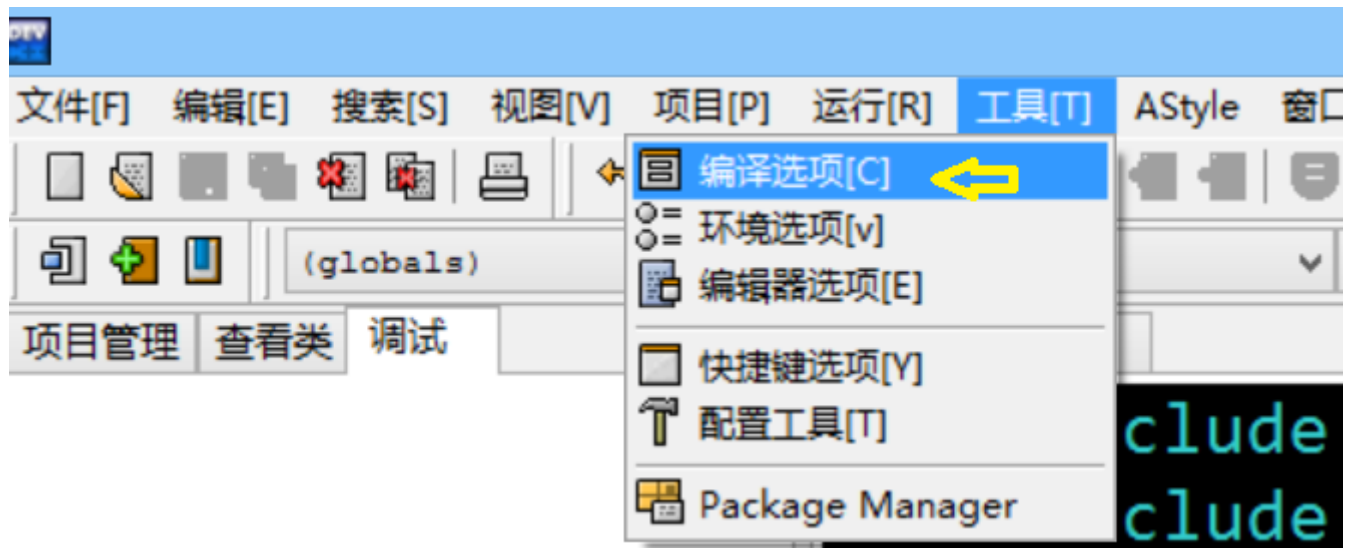
## 1.2.数据类型占用容量

`sizeof(int)`可计算int类型的字节占用(int占用是4)

## 1.3.dev使用c99(期末考试看)

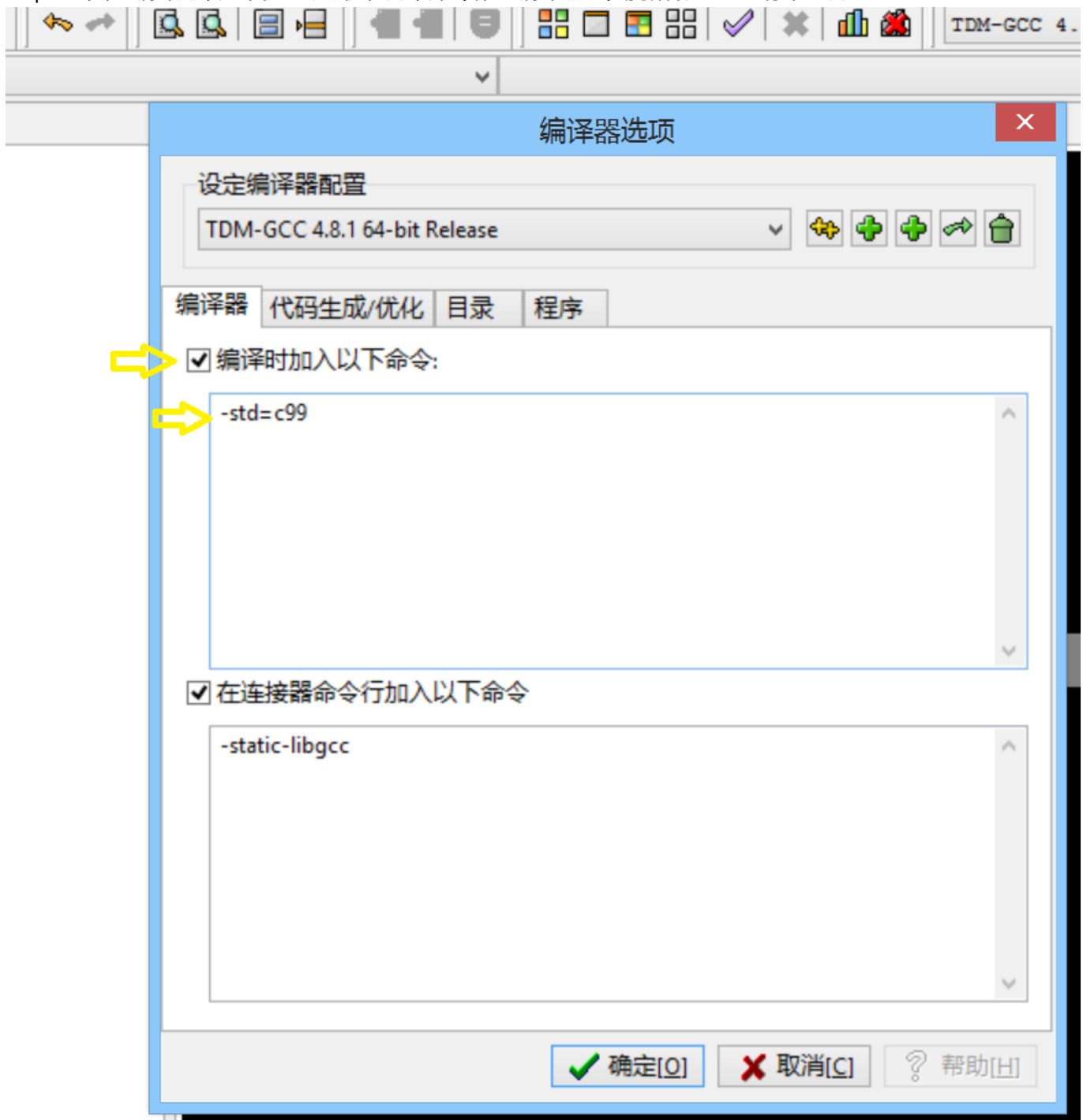
适用于DEV-C++报Error如: [Error] 'for' loop initial declarations are only allowed in C99 mode [Note] use option `-std=c99` or `-std=gnu99` to compile your code

1.



2. step1 工具->编译选项->编译器选项卡中,在"编译时加入以下命令"复选框前打钩,里面输入命令 `-std=c99` (与GCC不同,这里c99中的字母c是小写)

step2 工具->编译选项->代码生成/优化选项卡中,在C编译器->支持所有ANSI C标准上选NO



## 2.函数

### 2.1.变量&形式参数

全局变量 全体函数外 int

```
#include <stdio.h>

int a=10;
int main(){
    printf("%d",a);
}
```

形式参数：函数的局部变量 e.g. `void test(int,int)`

```
#include <stdio.h>
int sum(int,int);

int main()
{
    int a=10;
    int b=100;
    int c=sum(a,b);
    printf("%d",c);
}

int sum(int a,int b){
    return a+b; ///返回一个值给主函数
}
```

输出为110

单一数值在外函数内的改变不影响主函数的原数值，但数组可以做到在外函数修改的同时，主函数内的数组值也改变。 `void test(int arr[])` and `test(arr)`; 数组是整体传递。

## 2.2.静态变量

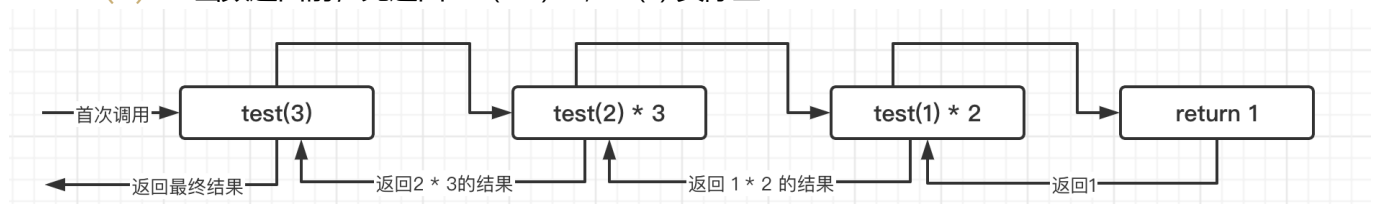
外函数内包含 `static int a`，该静态变量不会在函数结束时销毁其值，反而是保存其值，这个可以用作计算函数被调用的次数。

## 2.3.递归

计算阶乘例(使用递归)

```
int test(int n){
    if(n==1) return 1;
    return test(n-1)*n;
}
```

`a=test(5)` test函数返回前，先返回 `test(n-1)*n`, `test(5)` 实际上 =  $5 * 4 * 3 * 2 * 1$



**计算 `test(n)` 前需计算 `test(n-1)`**

## 3.指针

### 3.1.definition

```
#include <stdio.h>
int main(){
int a = 10;
///指针类型需要与变量的类型相同，且后面需要添加一个*符号（注意这里不是乘法运算）表示是对
于类型的指针

int * p = &a; ///这里的&并不是进行按位与运算，而是取地址操作，也就是拿到变量a的地址
printf("a在内存中的地址为: %p", p); ///地址使用%p表示
}
```

`printf("%p %d",p,*p);`地址为%p输出，\*p在指针变量前添加一个\*号（间接运算符，也可以叫做解引用运算符）来获取对应地址存储的值。

取对应地址存储的值，可以直接更改其数值使其参与运算。

e.g. `void test(int * a,int * b)`int类型的指针,此外函数内的运算将直接改变主函数的a、b值  
`test(&a,&b);`, &取地址操作。

初始化指针值，以防其乱指地址 `int * a = NULL;`

### 3.2.const 指针 & 指针数组

`const int * p =&a;`这样的指针不允许改变地址值，但可以改变其指向的地址，如 `*p=10` 将报错，`p=&b` 将改变指针指向地址。

```
#include <stdio.h>
int main(){
char str[] = "Hello World!";
char * p = str; ///把数组作为地址赋值给指针变量p
printf("%c",*p); ///将打印字符串第一个字符
printf("%c",p[1]);///打印e
printf("%c %c %c",*p,*p+1,*p+2)); ///p+1移动整个char类型对应大小
}
```

0x00000000

0xFFFFFFFF



一段4G的内存空间

一个长度为3的int类型数组在内存中占据 4个字节x3 的空间  
此空间是一块连续的空间，首元素起始地址是：0x00FAFAFA

类似的是 `int * p=str`, `*(p+1)` 越过一个int数值的空间到达下一个int数值空间，\*p本身就是数组第一位int数组的地址。

`*(p+i) <=> str[i]` ///实际上就是可以相互转换的



```

*p //数组的第一个元素
p //数组的第一个元素的地址
p == str //肯定是真，因为都是数组首元素地址
*str //因为str就是首元素的地址，所以这里对地址加*就代表第一个元素，使用的是指针表示法
&str[0] //这里得到的实际上还是首元素的地址
*(p + 1) //代表第二个元素
p + 1 //第二个元素的内存地址
*p + 1 //注意*的优先级比+要高，所以这里代表的是首元素的值+1，得到字符'K'

```

对于二维数组: `int arr[][3]={ {1,2,3},{4,5,6}};`  for instance:

```

int main(){
    int arr[][3]={ {1,2,3},{4,5,6}};
    int * p=arr[0];    ///实际上就是二维数组的首元素
    printf("%d=%d",arr[1][1],*(p+4));    ///前者是第二个数组的第二个数值，后者一样，也
    就是内存中连续地址中的第五个
}

```

### 3.3.多级指针



这个指针变量存放的是另一个指针变量在内存中的地址

指向变量a的指针本身也是个变量,因此可用\*\*创建一个指针的指针(二级指针)

```

int a=20;
int * p=&a;
int ** pp=&p;
int *** ppp=&pp;    ///指向指针的指针的指针

```

### 3.4.指针数组与数组指针

`int * arr[3]={&a,&b,&c};` 为存放指针的数组(整型指针类型的数组), `*arr[0]=999` 将999赋值到a上; `[]` 运算符的优先级更高, 所以这里先通过`[0]`取出地址, 然后再使用`*`将值赋值到对应的地址上.

`int (*p)[3]` 为数组指针, 表示指向整个数组, `()` 将值的提取提前, `int (*p)[3]=&arr` 直接对整个数组取一次地址(不过本质还是提取数组首元素的地址)

- `p` 代表整个数组的地址
- `*p` 表示所指向数组中首元素的地址
- `*p+i` 表示所指向数组中第 `i` 个 (0开始) 元素的地址 (实际上这里的 `*p` 就是指向首元素的指针)
- `*(p + i)` 就是取对应地址上的值了

e.g.

```
#include <stdio.h>
int main(){
int arr[3] = {111, 222, 333};
int (*p)[3] = &arr; ///直接对整个数组再取一次地址
printf("%d, %d, %d", *(*p+0), *(*p+1), *(*p+2)); ///要获取数组中的每个元素, 稍微有点麻烦
///此输出即111,222,333
}
```

对于二维数组`int arr[][3] = {{111, 222, 333}, {444, 555, 666}};`,`int (*p)[3]=arr;`即可对整个二维数组取地址。

比如现在我们要访问第一个数组的第二个元素, 根据上面`p`各种情况下的意义:

```
printf("%d", *(*p+1)); //因为上面直接指向的就是第一个数组, 所以想要获取第一个元素和之前是
```

那么要是我们现在想要获取第二个数组中的最后一个元素呢?

```
printf("%d", *(*p+1)+2);
```

首先`*(p+1)`为一个整体, 表示第二个数组 (因为是数组指针, 所以这里+1一次性跳一个数组的长度), 然后再到外层+2表示数组中的第三个元素, 最后再取地址, 就是第二个数组的第三个元素了。再译:`*(p+1)`越过整个一维数组的内存, 进入第二个数组, 其+2, 表示第二个数组的第三个元素

当然也可以使用数组表示法:

```
printf("%d", p[1][2]);
```

这两用着是同一个东西.

### 3.5.指针函数与数组指针

我们的函数可以返回一个指针类型的结果, 这种函数我们就称为指针函数。

```
#include <stdio.h>
int * test(int * a){    ///函数的返回值类型是int *指针类型的
return a;
}
int main(){
int a = 10;
int * p = test(&a);    ///使用指针去接受函数的返回值
printf("%d", *p);
```

```
printf("%d", *test(&a)); ///当然也可以直接把间接运算符在函数调用前面表示直接对返回的地址取地址上的值
}
```

函数指针，实际上指针除了指向一个变量之外，也可以指向一个函数，当然函数指针本身 还是一个指针，所以依然是用变量表示，但是它代表的是一个函数的地址（编译时系统会为函数代码分配一段存储空间，这段存储空间的首地址称为这个函数的地址）

```
#include <stdio.h>
int sum(int a, int b) {
    return a + b;
}
int main(){
    int (*p)(int, int) = sum;
    printf("%p", p);
}
```

函数指针标准样式类型: (\* 指针变量名称)(函数参数...) 注意一定要把\*和指针变量名称括起来，不然优先级不够。

函数指针也可以用作调用函数，实现函数回调的功能。

```
#include <stdio.h>
int sum(int a, int b) {
    return a + b;
}

int main(){
    int (*p)(int, int) = sum;
    int result = (*p)(1, 2); //就像我们正常使用函数那样，(*p)表示这个函数，后面依然是在小括号里面填上实参
    int result = p(1, 2); //当然也可以直接写函数指针变量名称，效果一样（咋感觉就是给函数换了个名呢）
    printf("%d", result);
}
```

```
#include <stdio.h>
int sum(int (*p)(int, int), int a, int b){
    return p(a, b);
}
int sumImpl(int a, int b){ ///这个函数实现了a + b
    return a + b;
}

int main(){
    int (*p)(int, int) = sumImpl; ///拿到实现那个函数的地址
```

```
printf("%d", sum(p, 10, 20));
}
```

`p(a,b)`即`sumImpl`函数, `sum(p,10,20)`返回一个`sumImpl`函数的相关值;

## 4.结构体、联合体和枚举

### 4.1.结构体's definition

```
struct Student { //使用 (struct关键字 + 结构体类型名称) 来声明结构体类型, 这种类型是我
们
自己创建的 (同样也可以作为函数的参数、返回值之类的)
int id; //结构体中可以包含多个不同类型的数据, 这些数据共同组成了整个结构体类型 (当然结构
体内部也能包含结构体类型的变量)
int age;
char * name; //用户名可以用指针指向一个字符串, 也可以用char数组来存, 如果是指针的话, 那
么数据不会存在结构体中, 只会存放字符串的地址, 但是如果是数组的话, 数据会存放在结构体中
};
```

```
int main() {
struct Student { //也可以以局部形式存在
}; ///结构体最后分号前为结构变量, 可以不填
}
```

一个作用实例:

```
#include <stdio.h>

struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
} book = {"C 语言", "RUNOOB", "编程语言", 123456};

int main()
{
    printf("title : %s\nauthor: %s\nsubject: %s\nbook_id: %d\n", book.title,
book.author, book.subject, book.book_id);
}
```

```
#include <stdio.h>
#include <string.h>
struct Student{
```

```

    int years;
    char * name;
    float num;
};

int main(){
    char a[4];
    scanf("%s",&a);
    struct Student id={14,a,14};
    printf("%d %s %f",id.years,id.name,id.num);

    struct Student id2={24,a,14};
    printf(" %d %s %f",id2.years,id2.name,id2.num);
}

```

不同名称有不同的结构体数据.id2.years=13此类可以直接修改结构体内数据。

## 4.2.结构体数组与指针

```

#include <stdio.h>
struct Student {
    int id;
    int age;
    char * name;
};
int main() {
    struct Student arr[3] = {{1, 18, "小明"}, //声明一个结构体类型的数组，其实和基本类型声明数组是一样的
    {2, 17, "小红"}, //多个结构体数据用逗号隔开
    {3, 18, "小刚"}};
}

```

printf("%s",arr[1].name);即输出"小红"

### 结构体指针:

```

int main() {
    struct Student student = {1, 18, "小明"};
    struct Student * p = &student;
    printf("%s", (*p).name); //由于.运算符优先级更高，所以需要先使用*p得到地址上的值，然后再去访问对应数据
}

```

or

```

printf("%s", p->name); //使用 -> 运算符来快速将指针所指结构体的对应数据取出

```

e.g.

```
void test(struct Student * student){ //这里使用指针，那么现在就可以指向外部的结构体了
student->age = 19; //别忘了指针怎么访问结构体内部数据的
}
int main() {
struct Student student = {1, 18, "小明"};
test(&student); //传递结构体的地址过去
printf("%d", student.age);
}
```

### 4.3.联合体(又称共用体)

definition:

```
union Object { //定义一个联合体类型唯一不同的就是前面的union了
int a;
char b;
float c;
};
```

联合体与结构体的区别在于。联合体所有的变量共用一个空间 e.g.

```
#include "stdio.h"
#include <string.h>
struct student{
    int id;
    int age;
    char * name;
};
int main()
{
    union kk{
        int a;
        double b;
        char name[100];
    }nmsl;
    strcpy(nmsl.name,"dauwgefiuasdfjcasdf");

    printf("%d\n\n %lf\n\n %s \n\n",nmsl.a,nmsl.b,nmsl.name);

    nmsl.a=100;
    printf("%d\n\n %lf\n\n %s \n\n",nmsl.a,nmsl.b,nmsl.name);
}
```

观测现象用代码，最后的输出发现有100、乱码和d（d的ASCII码为100），因此联合体可以用作ASCII码查表用（一个应用例）

#### 4.4.枚举

```
/**
 * 比如现在我们设计：
 * 1 = 低档位
 * 2 = 中档位
 * 3 = 高档位
 */
enum status {low = 1, middle = 2, high = 3}; //enum 枚举类型名称 {枚举 = 初始值, 枚举...}
```

使用示范:

```
enum status {low = 1, middle = 2, high = 3};
int main() {
    enum status a = low; //和之前一样，直接定义即可，类型为enum + 枚举名称，后面是变量名称，值可以直接写对应的枚举
    printf("%d", a);
}
```

枚举量不设置初始值，则默认赋值0, 1, 2..... if在变量中间设置初始值，比如enum status {low, middle = 114514, high};则low=0, middle=114514, high=114515.

#### 4.5.typedef关键字

```
typedef int lbwnb; //食用方式：typedef 类型名称 自定义类型别名
```

e.g.

```
typedef const char * String; //const char * 我们就起个名称为String表示字符串
int main() {
    String str = "Hello World!"; //是不是有Java那味了
    printf(str);
}
```

### 5.c语言底层特性

#### 5.1.预处理文件

## matlab 科学计算(信号与系统)

# 单片机应用技术(stm32)

---