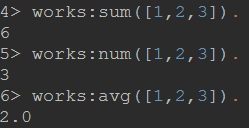
1. 求列表L（元素为整数）平均值

avg(L) ->

sum(L) -> sum(L, 0)**.**sum([H|T], Sum) -> sum(T, Sum + H);  
sum([], Sum) -> Sum**.**num([]) ->0;  
num([\_H|T]) ->1+num(T)**.**avg([]) ->0;  
avg(L)->  
 sum(L)/num(L)**.**

**运行效果：**



2. 求列表L（元素为整数）最小值

min(L) ->

min([H|T]) -> min(T, H)**.**min([H|T], Min) **when** H < Min -> min(T, H);  
min([\_|T], Min) -> min(T, Min);  
min([], Min) -> Min**.**

**运行效果：**

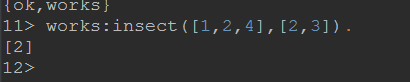


3. 求列表L1和L2的交集

insect(L1,L2) ->

insect(L1,L2) ->[X**||**X<-L1,Y<-L2,X=:=Y]**.**

**运行效果：**



4. 实现2个字符串拼接，不能使用++

concat(String1,String2) ->

concat(String1,String2)->  
 List=[String1|String2],  
 binary\_to\_list(list\_to\_binary(List))**.**

**运行效果：**



4. 进程字典、ETS、DETS的区别

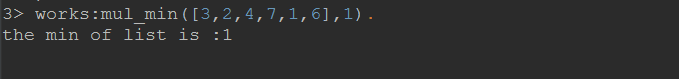
进程字典的数据为进程私有，无锁操作，而ETS和DETS存储的数据可供共享访问，具有读写锁，但DETS与ETS也有区别，ETS将数据存储在内存，而DETS则将数据存储在磁盘里面从而被持久化。

5. 实现多进程计算列表L的最小值，主进程开启3个子进程，主进程把L平均分为3部分，分别发消息给3个进程计算最小值，返回最小值给主进程把3个最小值比较得出L的最小值

mul\_min(L) ->

mul\_min(L,State)->  
  
 Size=length(L),  
%% 输入列表的元素个数需为3的倍数  
 L1=lists:sublist(L,1,Size **div** 3),  
 L2=lists:sublist(L,(Size **div** 3)+1,Size **div** 3),  
 L3=lists:sublist(L,(Size **div** 3)\*2+1,Size-length(L2)-length(L1)),  
  
 Pid1=spawn(**fun**()->loop\_min()**end**),  
 Pid2=spawn(**fun**()->loop\_min()**end**),  
 Pid3=spawn(**fun**()->loop\_min()**end**),  
%% 从当前进程(主进程)发送消息给子进程  
 Pid1 ! {self(),L1},  
 Pid2 ! {self(),L2},  
 Pid3 ! {self(),L3},  
  
 **receive** %% 主进程接收子进程的返回值  
  
 {Pid1,Min1} **when** State==1 ->put(min,Min1)  
 ,mul\_min(L,2)  
 ;  
  
 {Pid2,Min2} **when** State==2 ->  
 Temp2 = get(min),  
  
 **if** Temp2 > Min2 -> put(min,Min2);  
 true->get(min)  
 **end** ,mul\_min(L,3)  
 ;  
  
 {Pid3,Min3} **when** State==3 ->Min3,  
 Temp3 = get(min),  
 **if** Temp3 > Min3 -> put(min,Min3);  
 true->get(min)  
 **end**,  
 io:format("the min of list is :~p~n",[get(min)])  
%% 递归出口  
 ,mul\_min(L,0)  
 **end.**loop\_min()->  
  
  
 **receive** {From,L}->  
 Min=lists:min(L),  
 From ! {self(),Min},  
 loop\_min()  
 **end.**

**运行效果：**



6. 实现一个简单的聊天服务器（通过socket连接），支持多个客户端连接，某个客户端发送聊天信息后，其他客户端（包括自己），能看到聊天数据（格式：[客户端ID]：聊天信息）

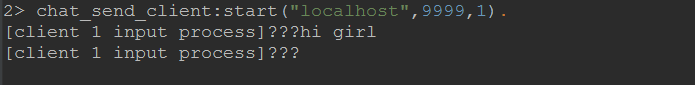
%%%-------------------------------------------------------------------  
%%% @author Administrator  
%%% @copyright (C) 2018, <COMPANY>  
%%% @doc  
%%% 监听Socket连接，并群发消息到所有的client  
%%% @end  
%%% Created : 20. 八月 2018 20:23  
%%%-------------------------------------------------------------------

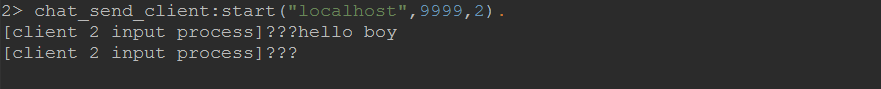
-module(chat\_server)**.**-author("Administrator")**.**-define(TCP\_OPTIONS,[list,{packet,0},{active,false},{reuseaddr,true}])**.**%% API  
-export([start/1])**.**start(ClientPort)->  
 register(client\_manager,spawn(**fun**()->manage\_clients([])**end**)),  
 {ok,LSocket}=gen\_tcp:listen(ClientPort,?TCP\_OPTIONS),  
 do\_accept(LSocket)**.**do\_accept(LSocket)->  
 {ok,Socket}=gen\_tcp:accept(LSocket),  
 spawn(**fun**()->handle\_client(Socket)**end**),  
 client\_manager ! {connect,Socket},  
 do\_accept(LSocket)**.**handle\_client(Socket)->  
 **case** gen\_tcp:recv(Socket, 0) **of** {ok,Data}->  
 client\_manager ! {data,Data},  
 handle\_client(Socket);  
 {error,closed}->  
 client\_manager !{disconnect,Socket}  
**end.**manage\_clients(Sockets)->  
  
 **receive** {connect,Socket}->  
 io:format("connect the socket~w~n",[Socket]),  
 SocketList=[Socket|Sockets],  
 manage\_clients(SocketList);  
  
 {disconnect,Socket}->  
 SocketList=lists:delete(Socket,Sockets),  
 manage\_clients(SocketList);  
  
 {data,Data}->  
 send\_data(Sockets,Data),  
 SocketList=Sockets,  
 manage\_clients(SocketList)  
  
 **end.**send\_data(Sockets,Data)->  
 SendData=**fun**(Socket)->  
 gen\_tcp:send(Socket,Data)  
 **end**,  
 lists:foreach(SendData,Sockets)**.**

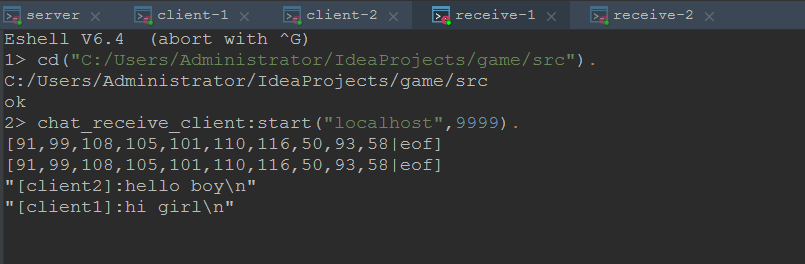
%%%-------------------------------------------------------------------  
%%% @author Administrator  
%%% @copyright (C) 2018, <COMPANY>  
%%% @doc  
%%% 用于启动指定client广播消息的进程  
%%% @end  
%%% Created : 21. 八月 2018 11:08  
%%%-------------------------------------------------------------------  
-module(chat\_send\_client)**.**-author("Administrator")**.**%% API  
-export([start/3])**.**-export([talk/2])**.**start(IP, Port,ClientId) ->  
 {ok, Socket} = gen\_tcp:connect(IP, Port, [binary, {packet, 4}]),  
 talk(Socket,ClientId)**.**talk(Socket,ClientId)->  
 Id=integer\_to\_list(ClientId),  
 io:format("[client "++Id++" input process]"),  
 Msg=io:get\_line("input："),  
 ok=gen\_tcp:send(Socket,term\_to\_binary("[client"++Id++"]:"++Msg)),  
 talk(Socket,ClientId)  
**.**

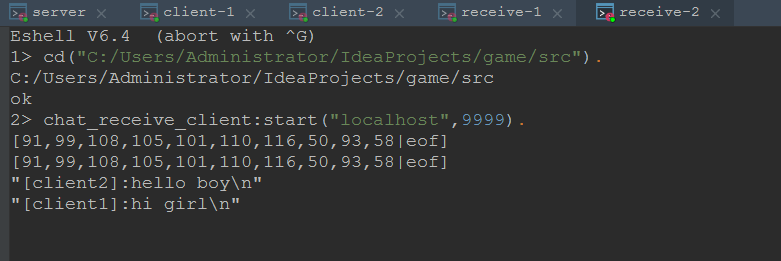
%%%-------------------------------------------------------------------  
%%% @author Administrator  
%%% @copyright (C) 2018, <COMPANY>  
%%% @doc  
%%% 客户端消息显式  
%%% @end  
%%% Created : 21. 八月 2018 11:14  
%%%-------------------------------------------------------------------  
-module(chat\_receive\_client)**.**-author("Administrator")**.**%% API  
-export([start/2])**.**start(Ip,Port)->  
 {ok,Socket}=gen\_tcp:connect(Ip,Port,[binary,{packet,4}]),  
 receive\_msg(Socket)**.**receive\_msg(Socket)->  
 **receive** {tcp,Socket,Bin}->  
 Msg=binary\_to\_term(Bin),  
 io:format("~p~n",[Msg]),  
 receive\_msg(Socket)  
 **end.**

**运行效果：**



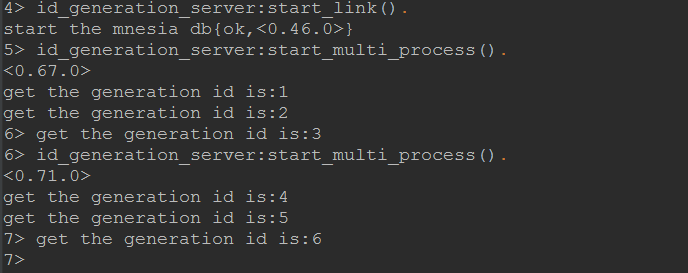






7. 实现一个唯一id进程（从1开始递增），开3个进程同时请求id进程，获取唯一id并打印出来，看有没有出现重复。

-module(id\_generation\_server)**.**-author("Administrator")**.**-behavior(gen\_server)**.**%%唯一id  
-record(ids,{idtype, ids})**.**%%ids表状态  
-record(state, {})**.**-export([init/1,handle\_call/3,handle\_cast/2,handle\_info/2,terminate/2,code\_change/3])**.**%% API  
-export([start\_multi\_process/0])**.**-export([get\_newId/1])**.**-export([start\_link/0])**.**%%同时开启多个进程  
start\_multi\_process()->  
 spawn(id\_generation\_server,get\_newId,[client]),  
 spawn(id\_generation\_server,get\_newId,[client]),  
 spawn(id\_generation\_server,get\_newId,[client])**.**%%注册生成自增id的进程  
start\_link()->  
 gen\_server:start({local,?MODULE},?MODULE,[],[])**.**get\_newId(IdType)->  
 mnesia:force\_load\_table(ids),  
 Id=gen\_server:call(?MODULE,{get\_newId,IdType}),  
 io:format("get the generation id is:~p~n",[Id])**.**init([]) ->  
 mnesia:start(),  
 io:format("start the mnesia db"),  
 mnesia:create\_schema([node()]),  
 **case** mnesia:create\_table(ids,[{type,ordered\_set},  
 {attributes,record\_info(fields,ids)},  
 {disc\_copies,[]}  
 ]) **of** {atomic,ok}->  
 {atomic,ok};  
 {error,Reason}->  
 io:format("create table error:~p~n",[Reason])  
 **end**,  
 {ok,#state{}}**.**handle\_cast(\_From,State)->  
 {noreply,State}**.**handle\_info(\_Info,State)->  
 {noreply,State}**.**terminate(\_From,State)->  
 {ok,State}**.**code\_change(\_OldVer,State,\_Ext)->  
 {ok,State}**.**%%generate new Id for given type  
handle\_call({get\_newId,IdType},\_From,State)->  
 F=**fun**()->  
 Result=mnesia:read(ids,IdType,write),  
 **case** Result **of** [S]->  
 Id=S#ids**.**ids,  
 NewColumn=S#ids{ids=Id+1},  
 mnesia:write(ids,NewColumn,write),  
 Id;  
 []->  
 NewColumn=#ids{idtype=IdType,ids=2},  
 mnesia:write(ids,NewColumn,write),  
 1  
 **end  
 end**,  
 **case** mnesia:transaction(F)**of** {atomic,Id}->  
 {atomic,Id};  
 {aborted,Reason}->  
 io:format("run transaction error ~1000.p ~n",[Reason]),  
 Id=0;  
 \_Els->  
 Id=1000  
 **end**,  
 {reply,Id,State}  
**.**



8.使用gen\_server实现 题目5

%%开启主进程  
start\_link()->  
 gen\_server:start({local,?MODULE},?MODULE,[],[])  
  
**.**mul\_min(L)->  
  
 Size=length(L),  
%% 输入列表的元素个数需为3的倍数  
 L1=lists:sublist(L,1,Size **div** 3),  
 L2=lists:sublist(L,(Size **div** 3)+1,Size **div** 3),  
 L3=lists:sublist(L,(Size **div** 3)\*2+1,Size-length(L2)-length(L1)),  
  
  
 %%使用该函数通过gen\_server启动多个进程  
%% Pid1=gen\_server:start\_link(?MODULE,[L1],[]),  
%% Pid2=gen\_server:start\_link(?MODULE,[L2],[]),  
%% Pid3=gen\_server:start\_link(?MODULE,[L3],[]),  
 Pid1=spawn(mul\_min\_gen\_server,loop\_min,[L1,1]),  
 Pid2=spawn(mul\_min\_gen\_server,loop\_min,[L2,2]),  
 Pid3=spawn(mul\_min\_gen\_server,loop\_min,[L3,3]),  
  
 io:format("pid : ~p~n",[Pid1]),  
 io:format("pid : ~p~n",[Pid2]),  
 io:format("pid : ~p~n",[Pid3])  
  
**.**%%发起计算最小值服务器的远程调用  
loop\_min(L,Index)->  
   
 Min=  
 gen\_server:call(?MODULE,{loop\_min,L,Index})  
 ,  
 io:format("the min: ~p~n",[Min])  
  
**.**%%the callback interface  
init([]) ->  
 io:format("start the master process ~n"),  
 {ok,ets:new(min\_table,[public,named\_table,ordered\_set])}**.**handle\_cast(\_From,State)->  
 {noreply,State}**.**handle\_info(\_Info,State)->  
 {noreply,State}**.**terminate(\_From,State)->  
 {ok,State}**.**code\_change(\_OldVer,State,\_Ext)->  
 {ok,State}**.**%%计算所给列表的最小值 -callback  
handle\_call({loop\_min,L,Index},\_From,State)->  
  
 Min=lists:min(L),  
 ets:insert(min\_table,{Index,Min}),  
 {reply,Min,State}**.**show()->  
  
 List=ets:tab2list(min\_table),  
 [{\_,Min1},{\_,Min2},{\_,Min3}] = List,  
 Result=lists:min([Min1,Min2,Min3]),  
 io:format("the min of list is : ~p~n",[Result])**.**

**运行效果：**

