**《课程综合实践Ⅱ》实验报告二**

**区块链原理初识及数字币模拟器操作**

**姓名\_\_\_ 沈云昊 专业 电子信息工程**

**学号 3220104126 联系方式 2766636730@qq.com**

## 一、实验目的

1. 在ubuntu系统上配置python环境。

2. 学习scp等文件远程拷贝命令，在宿主机和虚拟机之间传输文件。

3. 了解区块链原理，部署并操作simchain数字币模拟器。

## 二、实验内容及要求

**准备工作：为了方便后续的学习和实践，建议同学们自带电脑，在自己电脑上进行实验内容，并完成实验报告。**

**实验步骤（仔细阅读，按照步骤完成实验）**

**（本次实验所需的静态资源文件可见学在浙大上实验二的说明，包含simchain数字币模拟器）**

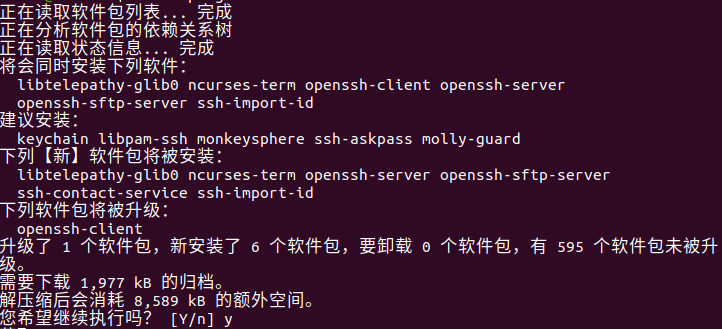
### SCP远程文件传输

scp命令是基于ssh登录进行安全的远程文件拷贝命令，可用于主机和虚拟

机之间的文件复制，因此我们首先需要在ubuntu中安装ssh工具，输入以下命令：

sudo apt-get update

sudo apt-get install ssh-contact-service



下载完成后，我们需要开放22号端口作为ssh服务器的默认运行端口，输入以下代码：

sudo ufw disable ##关闭防火墙

sudo ufw allow 22 ##打开22号端口

scp命令格式如下：

scp -r [要传送的目录] [虚拟机用户名]@[虚拟机ip]:[要接收的目录]

为了看到虚拟机的ip地址，我们首先需要安装net工具，

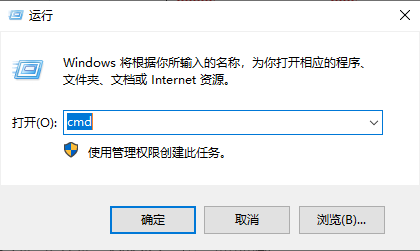
sudo apt-get install net-tools

安装完成后，输入命令，查看IP：

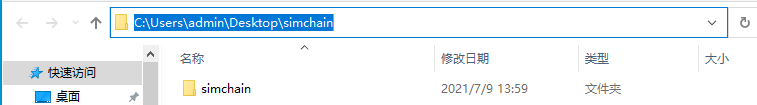
ifconfig

其中，ens33项中的inet即为ip地址

得到虚拟机ip信息后，我们在主机上解压学在浙大上下载的simchain压缩包，并按windows+R快捷键呼出运行框，在框中输入cmd打开控制台。



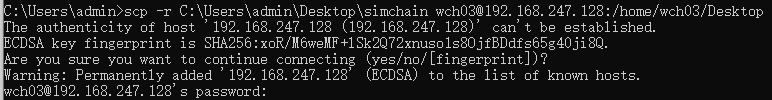
在cmd中输入scp命令将文件传输到虚拟机内，例如，



即命令为：

scp -r C:\Users\admin\Desktop\simchain wch03@192.168.247.128:/home/wch03

意为把在主机上的simchain文件夹传送到ip地址为192.168.247.128的wch03虚拟机内的/home/wch03文件夹中，



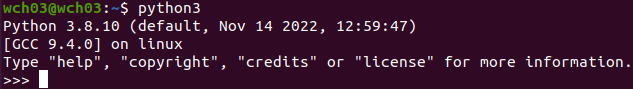
是否继续连接输入yes后回车，输入虚拟机密码后回车即可(为保护安全性，输入内容不会在cmd内显示)

### Python环境配置

ubuntu系统安装后，应已经配置了python环境，在虚拟机中打开终端，输入命令

python3

应能跳出如图python3命令行



在python3命令行中输入命令退出

exit()

为了便于python程序的包管理，我们仍需要安装pip作为python的包管理工具，输入命令

sudo apt install python3-pip

输入命令查看是否安装成功

pip -V

### Simchain数字币模拟器配置及基础操作

将simchain文件夹转到虚拟机中后，在虚拟机内进行如下操作：

打开终端，使用cd命令进入到刚刚传输的simchain目录内，使用ls命令查看该目录下包含的文件(需进到包含setup.py文件的那层目录)，输入命令，

pip install simchain

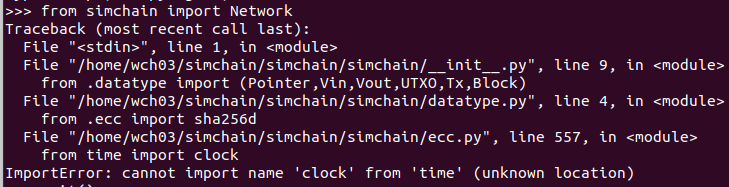
python3 setup.py install

Simchain配置完成后，我们正式开始操作：

Python3 ##进入python终端

>>>from simchain import Network,Peer ##从simchain导入Network,Peer结构

此时会报错



根据报错提示，我们发现报错的python文件为ecc.py，行数为557行，先退出python环境

>>>exit()

根据报错的文件路径提示，使用gedit文本编辑器打开该文件，输入命令（此处路径需替换为你自己的）

gedit /home/wch03/simchain/simchain/simchain/ecc.py

找到557行，将clock替换为process\_time即可，如图所示：



替换完成后保存。

重新进行刚才的流程，

Python3

>>>from simchain import Network,Peer

接下来我们将模拟一种类似比特币的情形(请截图保存在实验记录中)

1. **构建一个名为net的区块链网络，并假设网络中只有一位数字货币用户，并给自己写了一笔10000分的交易。**

在python环境中输入命令

>>>net = Network(nop = 1,von = 10000)

为确保网络正确构建，输入代码查看网络具体信息，

>>> net.nop ## 查询网络内节点数量，即用户数

>>> net.peers ## 列出所有节点坐标信息，坐标可类比于显示比特币网络中的IP地址

为方便操作，我们将第一位用户命名为Alice，输入命令

>>> Alice = net.peers[0]

输入命令查看Alice的相关身份信息，

>>> Alice.sk ## 查询Alice的私钥

>>> Alice.pk ## 查询Alice的公钥

>>> Alice.addr ## 查询Alice的地址

其中，公私钥和地址是Alice用户在该区块链网络中的身份证明，存在一一对应关系，即一个私钥对应一个公钥，一个公钥对应一个地址。

接下来，我们输入命令查看Alice账户相关信息，

>>> Alice.get\_balance() ## 查看Alice账户余额

可以看到正好是创建区块链网络时为他赋的初值10000，

随后我们输入命令查看Alice的UTXO集，

所谓UTXO(Unspent Transaction Output)，直译为未花费交易输出，在区块链中，每笔比特币交易均存在输入和输出，其结构如图所示：



根据比特币所属方的不同，别人付给你称为“交易输入”，你收到的称为“交易输出”，而你收到的还未付出给别人的即称为未花费交易输出（UTXO），可简单看作为一种类似于支票的支付单元，图中id为48edf…的交易输出内索引0号就是一个UTXO，其归属于地址为1AkB…的节点。

>>> Alice. get\_utxo() ## 查看Alice账户UTXO集

从打印输出的结果中可以看到UTXO的结构由两部分组成：

1. vout中的to\_addr表示该UTXO的使用权归地址为该值的用户所有(此处与刚刚我们查询的Alice的地址一致)。
2. pointer是一个指针，指向该UTXO所在的交易编号及在交易输出中的索引，而这笔交易实际已存在于该区块链的第一个区块中，输入命令

>>> Alice.blockchain[0].txs ## 查看第一个区块中存储的所有交易

可以看到区块中存储的和pointer指向的交易id是相同的。

1. **目前，区块链内只有一位用户，为了让整个系统正常运作，我们添加一位新的用户，并将其命名为Bob。**

输入命令

>>> net.add\_peer()

>>> Bob = net.peers[1]

随后请按（1）中的方式，查看该用户的公钥、私钥、地址和账户余额，应完全与用户Alice相异，然后我们查看同一区块链网络内的两者的UTXO集和区块链是否相同，输入命令

>>> Alice.blockchain == Bob.blockchain

>>> Alice.utxo\_set == Bob.utxo\_set

应输出True，原因在于区块链系统的正常运行是以全网数据保持一致为前提的，于是，当下线过的存储节点再次上线或者新的存储节点加入到网络中时，这些节点必须要向网络中的其他存储节点发送请求来同步本地存储的区块链数据。

需要注意的是，这里的UTXO集一致与我们之前通过get\_utxo()方法获取到的个人的UTXO集有区别，这里指的是区块链网络记录的所有UTXO保持一致。

1. **我们模拟一种现实比特币交易的情况，为了吸引更多的用户加入这个区块链中，Alice决定向Bob转一笔钱让他帮忙宣传。**
2. **创建交易**

输入命令，

>>> Alice.create\_transaction(Bob.addr,100)

可以看到create\_transaction需要两个最基本的参数，接收转账方的地址和转账金额，此时我们查看当前的交易的“交易输出”，输入命令：

>>> Alice.current\_tx.tx\_out

可以看到产生了两个交易输出，一个归属于接收转账方Bob且金额100，另一个归属于发送转账方且金额为9890，

此时我们再用步骤（1）（2）中的命令，查看两位用户的余额，但发现仍未发生改变，说明交易仍未结束。

1. **交易广播**

为了让交易继续正常进行，Alice需要将该交易广播到所有区块链节点，输入命令，

>>> Alice.broadcast\_transaction()



在被其余节点确认后，我们再次输入命令查看两位用户的余额，发现已经被更新，此时我们发现Alice账户被多扣除了10分，且10分的交易费消失了，整个系统的收支看似已经不平衡了，为了确认转账是否真正被实现了，我们输入命令来获取Alice还未确认的UTXO

>>> Alice.get\_unconfirmed\_utxo()

并请同学们以此方式获取Bob还未确认的UTXO，账户余额已经改变，但还有未确认的UTXO的原因在于：

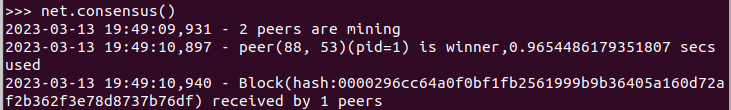
对一个区块链网络而言，最重要的是数据的一致性，因此当创建的有效交易广播到节点时，节点会将其存放到自己的交易池中，每个节点都有自己的独立交易池，由于P2P网络的特殊性，同一时刻每个节点的交易池可能不一样。为了保证区块链数据的一致，需要达成一个分布式共识，即在所有节点中选择一个“特殊”节点，将该节点的交易池打包成最新区块。为了找到这个“特殊”节点，出现了各种共识算法，在每一轮共识发起时，网络中的共识节点将开始解题竞赛，第一个得到答案的将获得打包区块的权利，其交易池中的交易将被写入区块链，其他节点按照他的区块更新自己的区块链数据，从而保证了全网数据的一致。

为了鼓励所有节点解题，区块链网络存在激励机制，即获胜的节点将获得系统给予的固定金额的货币和交易费作为奖励，这也是之前提到的多扣除的10分的去向。直到共识完成，即区块被成功打包，交易被成功写入区块链后，两位用户的未确认UTXO才会消失，账户的余额也才会被真正改写。

1. **共识**

为了让交易最终完成，我们输入命令让整个区块链网络就这次交易达成共识：

>>> net.consensus()



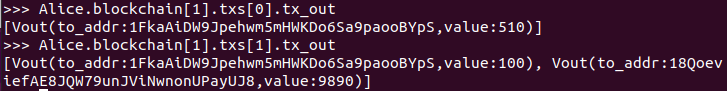
分析图中的结果，有1个节点在解题竞赛中获胜，成功的打包了包含本次交易的区块。

根据winner的pid(0为Alice,1为Bob)，此时我们再次查看获胜节点的账户余额， 发现余额多了510分(500是系统给予的固定金额，10是交易费)。

随后，我们再次输入命令查看Alice和Bob的未确认UTXO，应显示为空，输入命令查看区块链高度，

>>>Alice.get\_height()

发现区块高度增加了1，说明新的记载着转账交易的区块已被挂在链上，进一步的我们查看新区块上记载的交易的输出，如下图



发现记录了两笔交易的交易输出，一笔是奖励给挖出该区块的用户的交易输出，一笔是转账进行后双方持有的交易输出，使用权分别属于to\_addr指向的用户。

如此，通过创建交易、交易广播、共识、区块打包、广播区块、最后添加到区块链等步骤，区块链网络得以正常运行。

### Simchain数字币模拟器——钱包和密钥概念

我们重新构建一个12个节点的区块链网络，并为每个节点赋100000分的初始资金，将节点0命名为Alice，节点1命名为Bob，输入以下命令：

>>> from simchain import Network

>>> net = Network() ##使用默认构造方法即为12节点100000分

>>> Alice = net.peers[0]

>>> Bob = net.peers[1]

在数字货币的区块链网络中，每个用户即每个节点都有一个虚拟钱包属性(wallet)，钱包中存储的是节点的私钥、公钥和地址，在上文中已经提到，公私钥和地址存在一一对应关系，但同一个用户可以拥有多个这样的公钥-私钥-地址对，并将其存放在wallet中，是一对多关系。

为了验证这一点，模拟器中，我们输入命令

>>> Alice.wallet.nok ## 获取Alice钱包内的钥匙对数量

>>> Alice.wallet.keys[0].sk.to\_bytes() ##获取第一个密钥对的私钥字节串编码

>>> Alice.wallet.keys[0].pk.to\_bytes() ##获取第一个密钥对的公钥字节串编码

>>> Alice.wallet.addrs[0] ##获取第一个密钥对的对应地址

用这种方式，请同学们尝试获取Bob的钱包信息。

为了验证用户与钥匙对之间的一对多关系，我们输入命令，

>>> Alice.wallet.generate\_keys() ## 为Alice生成新的钥匙对

请同学们输入相应命令查看新生成的公私钥的字节串编码以及地址。

此时，用户Alice的钱包里存在两对密钥和两个地址，接下来，我们将来理解密钥对的用途，以Bob对Alice拥有的第一个钱包地址进行转账为例，输入命令，

>>> Bob.create\_transaction(Alice.wallet.addrs[0],100)

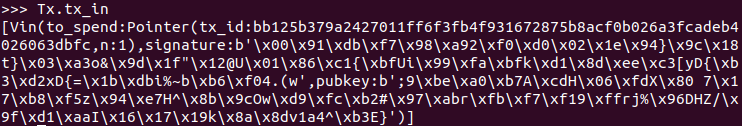
为方便之后的操作，获取Bob最新创建的交易，并将其命名为Tx,输入命令

>>> Tx = Bob.txs[-1]



之前提到交易由输入输出两部分组成，其中，交易输出像是本次交易对双方余额产生的最终结果，而交易输入可类比于一种体现交易可靠性的支付凭证，主要用于让别的节点相信本次交易的可靠性，现在我们查看交易输入，输入命令

>>> Tx.tx\_in



如图所示，发现交易输入在该模拟器中由三部分信息组成，对三部分信息

说明如下：

1. **to\_spend**：和UTXO中的pointer一致，是一个指针对象，指向要使用的UTXO所在交易编号及在交易中的索引，其实是表明了我现在要花的这笔UTXO的来源。
2. **signature**: 数字签名，交易发起方用自己的私钥对数据进行加密后的字节流
3. **pubkey**:交易发起方的公钥

事实上，在之前提到的交易广播这步中，广播的就是这种交易输入，输入命令，

>>> Bob. broadcast\_transaction()

其他节点会利用收到的signature和pubkey去验证这笔交易是否确实是由Bob发起，至于验证过程涉及大量密码学知识，同学们只需知道数字签名和公钥在区块链网络中能唯一确定交易有效性即可，为了验证这一点，我们控制变量做接下来的实验：

1. **不修改本次交易的交易输入，交由Alice做有效性判断**

输入命令,

>>> Alice.verify\_transaction(Tx)

结果显示有效。

1. **修改本次交易的交易输入中的数字签名(Signature)项，交由Alice做有效性判断**

输入以下命令

>>> from simchain import Vin ##引入Vin模块，便于伪造Vin

>>> vin = Tx.tx\_in[0] ##保存原本交易的真实Vin

>>> vin1 = Vin(vin.to\_spend,b'0'\*64,vin.pubkey)

##伪造新的vin，保持to\_spend和pubkey项不变，将数字签名伪造为64位0.

>>> Tx.tx\_in[0] = vin1 ##将伪造的vin1写入交易中

>>> Alice.verify\_transaction(Tx)

结果显示无效，并给出数字签名不匹配的报错且交易编号也相应改变.

下面请同学们参考b中的代码，自行完成下面两个实验的验证：

1. **修改本次交易的交易输入中的公钥(pubkey)项，交由Alice做有效性判断**
2. **修改本次交易的交易输入中的UTXO来源(to\_spend)项，交由Alice做有效性判断**

通过以上实验，我们不难发现只有当**数字签名、公钥、to\_spend完全匹配**才能被证明为是有效交易。

换句话说，如果要创建能被其他节点接收的有效交易，创建者必须使用**自己的UTXO**，即UTXO指向的地址属于创建者，而且数字签名、公钥必须与该地址相匹配，数字签名又是由与公钥一一对应的私钥“签署”，其实体现了**私钥-公钥-地址的绝对匹配机制**对交易有效性的保护。

**作业上交内容与事项：**

1．实验报告文档：按照要求完成实验并将关键步骤实验结果进行截图记录。注意文档工整；

2．程序代码：如果有程序代码或其他相关材料，可汇总压缩所有文件并上传压缩包。注意文件命名；

**本期实验作业上交期限：**

请在实验设置的截止日期内提交实验报告，若逾期提交，成绩会被适当打折。

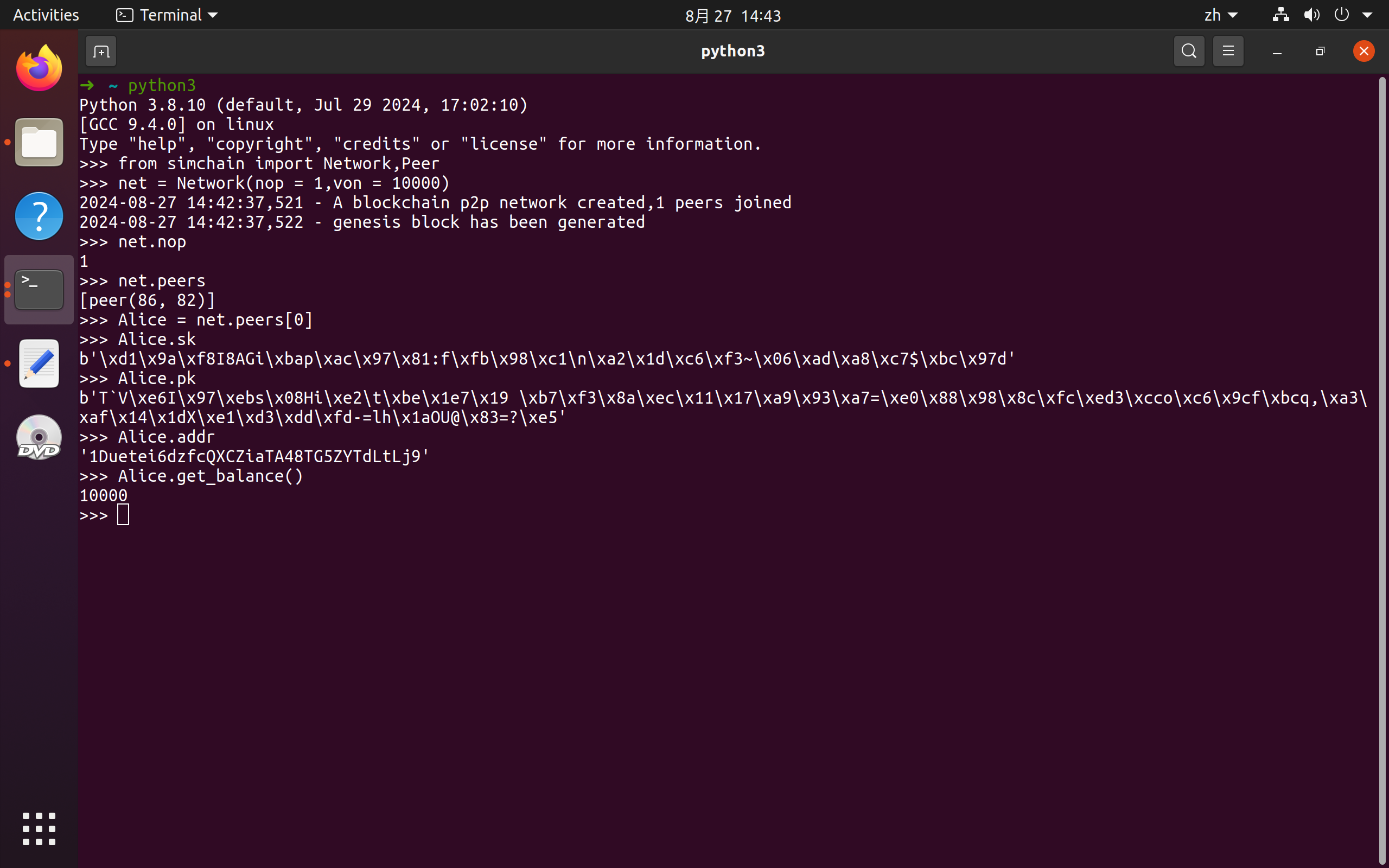
**本次作业上交内容：**

* 实验报告文档(.docx)

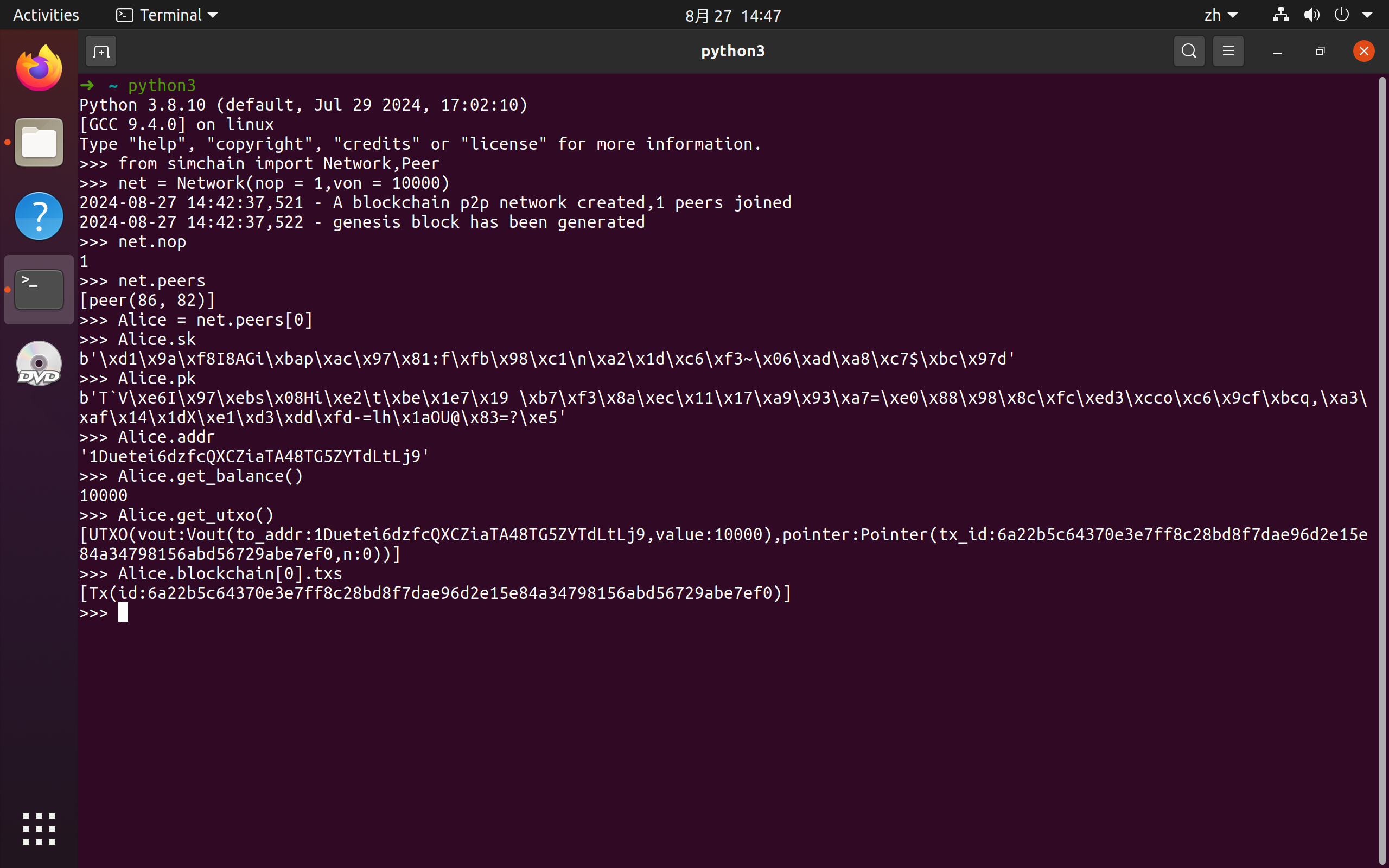
## 三、实验感受及记录

* 1. **实验感受（本次实验遇到的问题、主要收获等内容）**
  2. **实验记录（实验过程中关键步骤截图记录及文字描述）**

1. 构建一个名为net的区块链网络，并假设网络中只有一位数字货币用户，并给自己写了一笔10000分的交易。

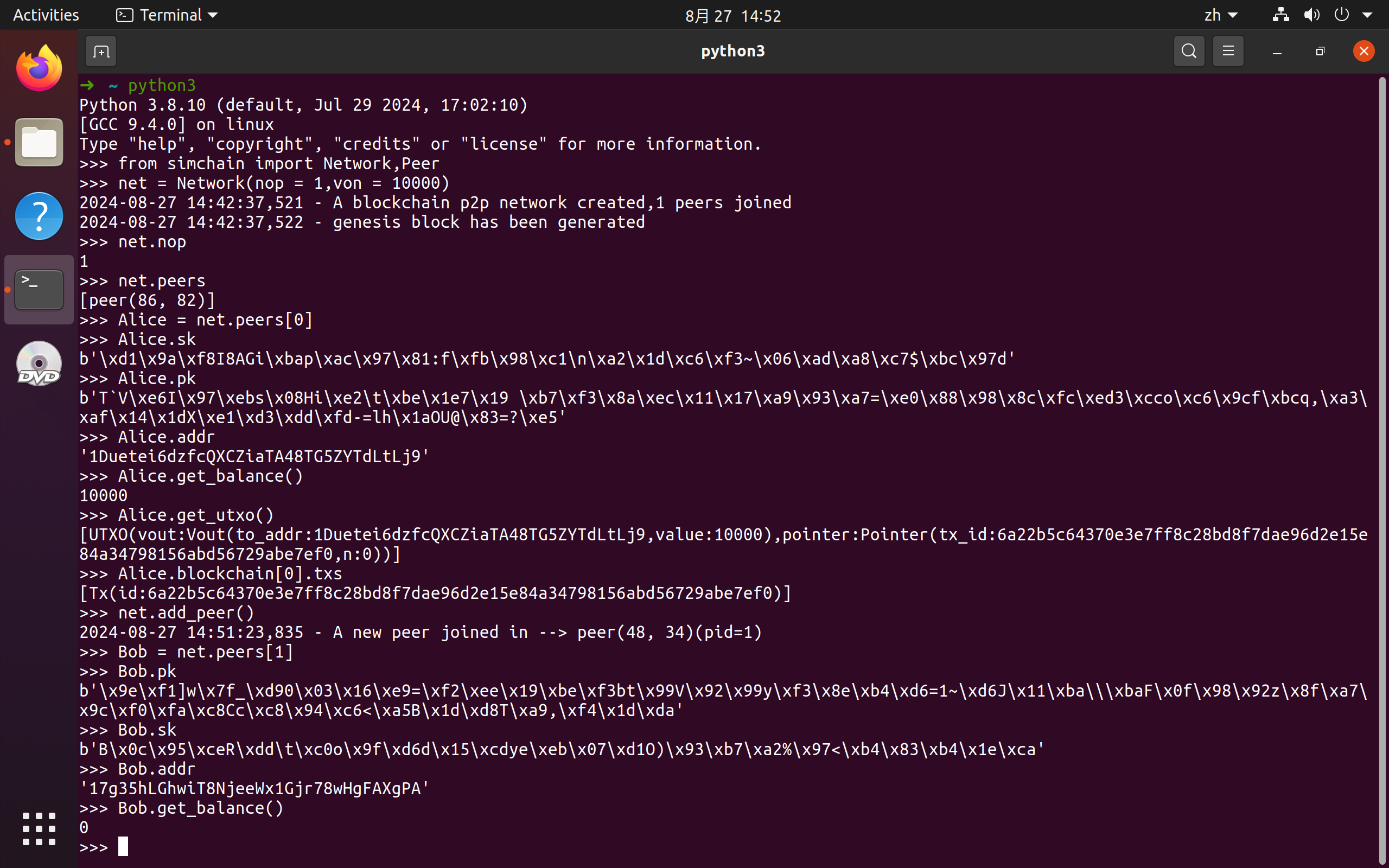


由上图可知，成功构建了一个区块链网络，并设置了一位用户，上图为用户的部分信息。

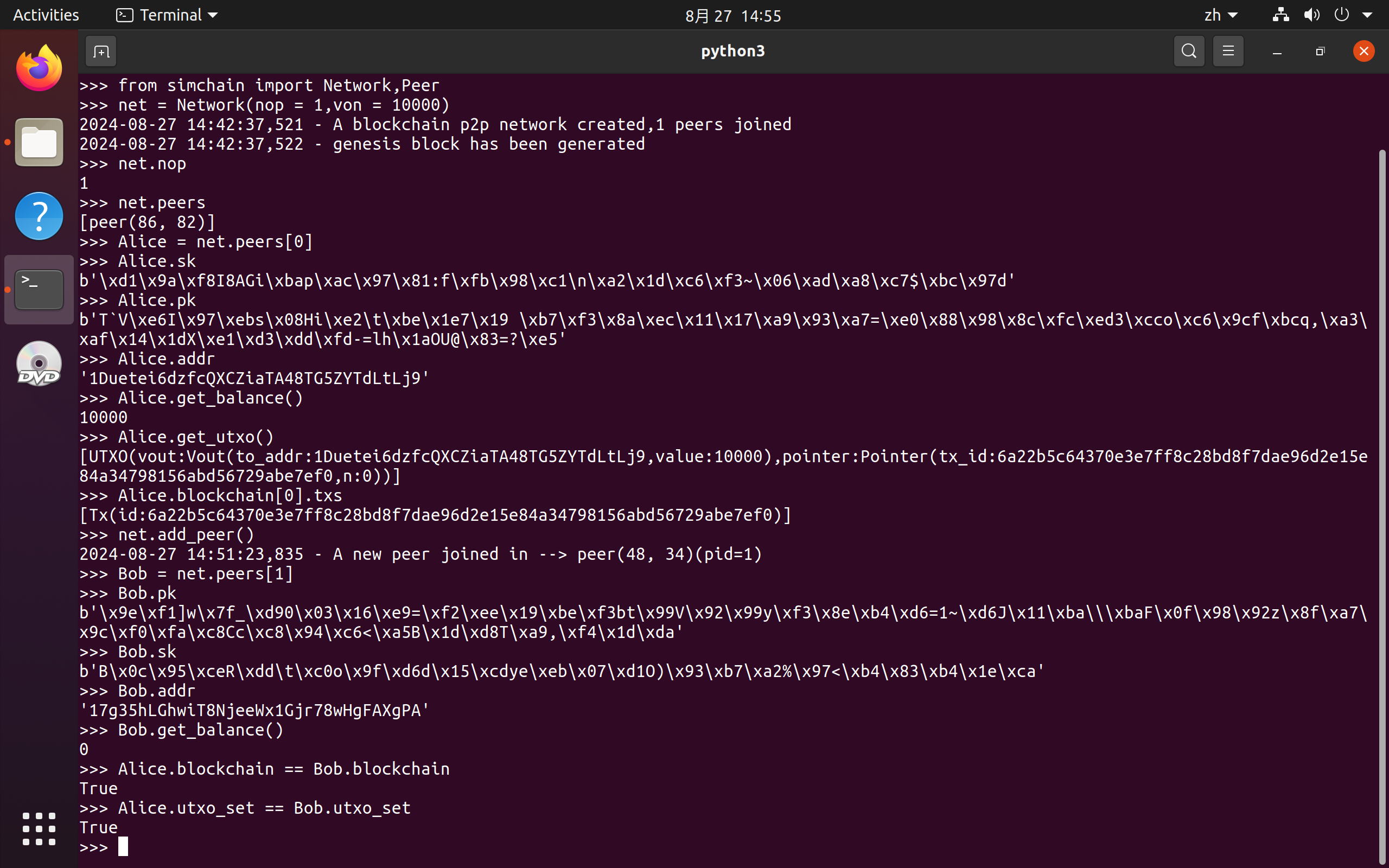


上图为Alice用户的UTXO块的信息，to\_addr部分与上图的地址相同，pointer指向的交易信息id符合存储的交易信息id。

2.目前，区块链内只有一位用户，为了让整个系统正常运作，我们添加一位新的用户，并将其命名为Bob。

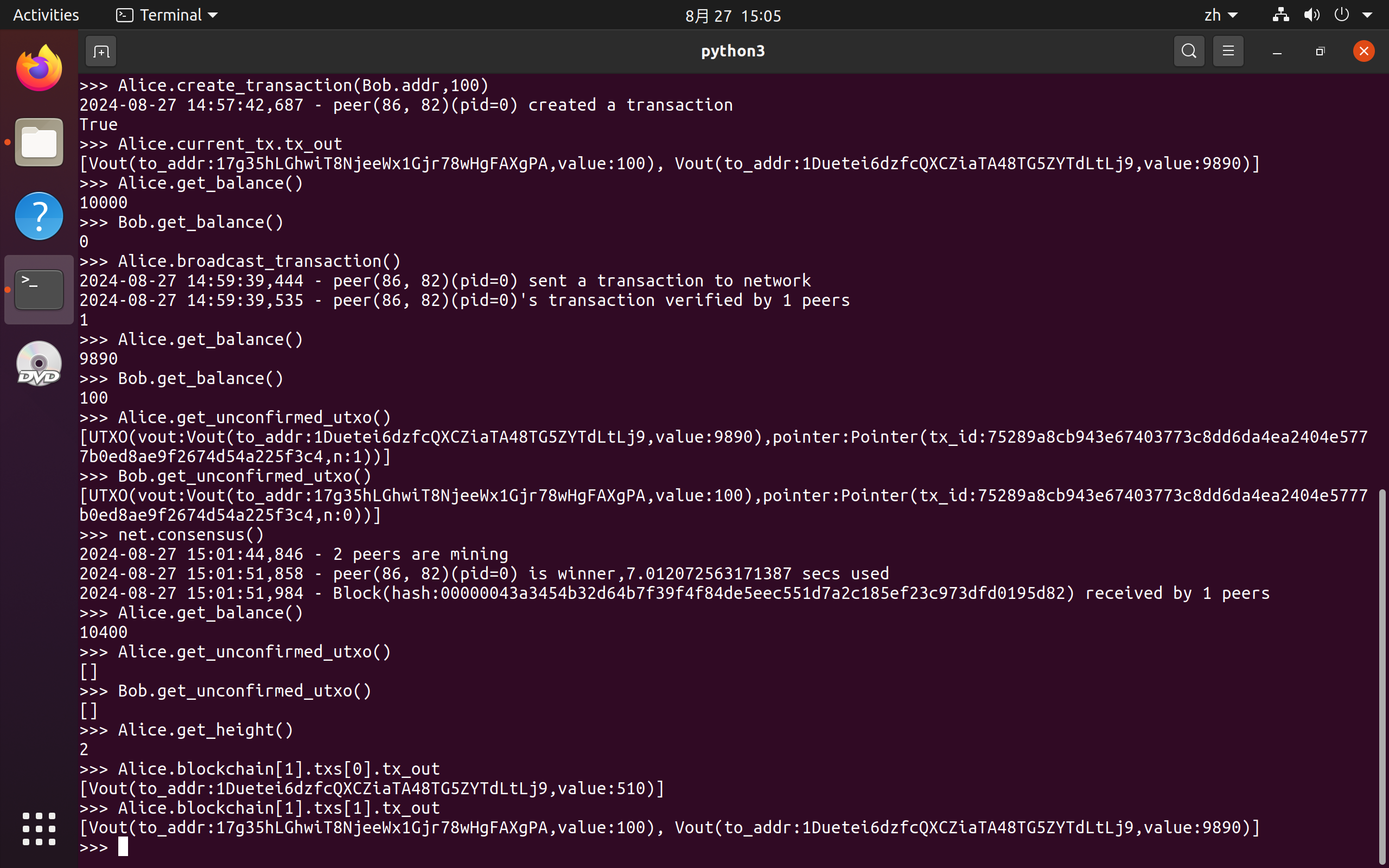


上图中成功添加了新用户Bob，依次打印出了私钥、公钥、地址和账户余额。



同时，同一区块链网络内的两者的UTXO集和区块链相同。

3.我们模拟一种现实比特币交易的情况，为了吸引更多的用户加入这个区块链中，Alice决定向Bob转一笔钱让他帮忙宣传。



上图是一笔交易的信息，Alice向Bob转账100，一开始二者的余额都不发生变化，广播后余额才变化。区块链网络确定utxo后，Alice获得打包的资格，所以余额增加了奖励500与手续费10，区块链的高度变为2。

4.Simchain数字币模拟器——钱包和密钥概念