

**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FLUMINENSE**
Campus Campos-Centro

Secretaria de Educação
Profissional e Tecnológica

Ministério
da Educação



CURSO BACHAREL EM SISTEMAS DE INFORMAÇÃO

Guilherme Luiz Lessa

Wesley Lubanco de Souza Gomes

PROGRAMAÇÃO DISTRIBUÍDA: observações sobre a implementação na plataforma linux, distribuição ubuntu 12.04, usando Pyro4, na linguagem python para implementação do sistema bancário

Relatório apresentado ao Instituto Federal de
Educação, Ciência e Tecnologia Fluminense,
Campus Campos Centro, como parte dos requisitos
para aprovação na disciplina Programação Paralela e
Distribuída.

Orientador: Prof^a. M. Sc. Maria Alciléia Alves Rocha

Campos dos Goytacazes/RJ

2014

Sumário

CAPÍTULO 1: INTRODUÇÃO

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

CAPÍTULO 3: IMPLEMENTAÇÃO

CAPÍTULO 4: Análise dos Resultados

CAPÍTULO 5: Referências

CAPÍTULO 1: INTRODUÇÃO

Com o objetivo de solidificar os conhecimentos teóricos abordado na disciplina de Programação Paralela e Distribuída, foram propostos cinco temas envolvendo um miniprojeto específico para desenvolvimento de um sistema distribuído. O tema escolhido pelo grupo foi “consulta negócio bancário”.

A proposta foi criar um pequeno sistema para executar chamadas remotas, por meio de uma rede de computadores, de um dado cliente a um servidor. O trabalho aqui documentado irá apresentar a tecnologia utilizada e a maneira como o sistema proposto foi desenvolvido.

Concomitante a proposta de trabalho e a problemática envolvida, as soluções em teoria já eram conhecidas e também alguns experimentos práticos de implementação de *Sockets*. Sendo assim, para tornar real a implementação deste trabalho, foi feita uma pesquisa com fim à análise da melhor forma para fazer com que duas máquinas pudessem se identificar em uma rede de computadores e a melhor ferramenta e sua utilização. As escolhas foram: Linguagem de programação *Python* e seu *Framework Pyro*.

Python é uma linguagem alto nível, multiplataforma, livre e de código aberto, Orientada a Objetos, tipagem dinâmica e forte, sintaxe intuitiva, documentação e agilidade. Quanto ao *Framework Pyro* pois possui uma poderosa e avançada tecnologia de objetos distribuídos escrita inteiramente em *Python*, facilidade de uso e reuso, lembra RMI, pequeno e leve e gratuito (Licença *MIT*).

Nosso intuito foi desenvolver um sistema comunicando-se por meio da rede possibilitando a chamada de objetos remotos(em outra máquina). O cliente instancia um objeto remotamente e por meio dele realizar as rotinas desejadas. O cliente poderá acessar sua conta e executar tarefas como consulta, saque e depósitos. As tarefas de desenvolvimento seguiram o modelo agilista. Pequenas partes de código-fonte criadas ou alteradas *commitadas* para o github o mais rápido possível possibilitando o rápido entendimento de qualquer pessoa que possa contribuir com futuras melhorias para o sistema.

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

Um sistema distribuído consiste de uma interligação entre computadores com características semelhantes ou não que através da rede, utilizando-se do *ip*, que é o protocolo que identifica uma máquina exclusivamente na rede, podendo trocar informações entre si e realizar trabalhos em paralelo. Podemos entender de forma mais simples, do que se trata um sistema distribuído ao observarmos a citação adiante. Você sabe que tem um sistema distribuído quando a falha de um computador do qual você nunca ouviu falar faz com que você pare completamente de trabalhar (Leslie Lamport). Sendo assim, podemos notar que ao interligarmos as máquinas, geramos um dependência entre elas por meio de uma programação que faz com que duas máquinas em pontos distintos de uma rede de computadores se comuniquem.

No caso deste trabalho, utilizamos da programação distribuída para realizar a comunicação entre duas máquinas, que trabalham como cliente e servidor. No caso do servidor, teremos um programa mais específico que irá defini-lo como um servidor na rede. Isto se materializa no momento em que definimos que o protocolo de internet, *ip* da máquina que contém o programa servidor é a *URI* do servidor na rede.

Para que esta comunicação fosse possível, utilizamos um *Framework* denominada *Pyro*. Esta biblioteca é adicionada a linguagem de programação *Python*, que já está pré-configurada no sistema operacional, ubuntu versão 12.04 (Cliente) e 12.01 (Servidor), também escolhido para o desenvolvimento deste trabalho. O *Pyro* possibilita que os objetos definidos no programa possam ser compartilhados pela rede. Ele consegue tornar o fazer com que o *ip*, possa se tornar um objeto de rede e este ainda é definido como uma *URI*, que é utilizada pelo cliente.

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. (<http://pt.wikipedia.org/wiki/Python>). Conforme citado no trecho acima, podemos ter uma visão resumida do que é a linguagem de programação *python*. Trata-se basicamente de uma linguagem simples para o usuário mas por trás de toda simplicidade temos interpretadores que fazem as devidas associações necessárias para a compilação do programa. Através dela, por ser de alto nível, podemos criar códigos de programas com uma codificação mais próxima da linguagem natural, no caso, o idioma é o inglês.

O *pyro* é um *Framework* que lhe permite criar aplicações na qual os objetos podem se comunicar entre si por meio de uma rede de computadores que demanda de mínima programação para tal comunicação. Para isso, basta utilizar a programação *python* normalmente com chamadas de métodos e objetos em outro computador. O *pyro* foi desenvolvido utilizando *python* puro sendo assim, consegue funcionar e diversas versões *python*. Altamente portátil, funciona em qualquer ambiente TCP/IP, Servidor de Nomes. Possui objetos móveis, que transferem automaticamente *bytecode Python* necessário, caso cliente e/ou servidor não possuir, suporta todos os tipos de variáveis *Python*, propaga exceções para o cliente simulando a ocorrência localmente. Suporta Proxies Dinâmicos, criação e remoção dinâmica de objetos *Pyro* pelo servidor, reconexão automática caso a rede falhe, invocação rápida one-way, compactação, criptografia(SSL).

O gerenciador de banco de dados utilizamos o *MongoDB*, banco de dados não relacional Orientado a documentos, possui alta performance, Software Livre e Schema Livre. Facilita o trabalho sendo desnecessária a criação prévia de tabelas e seus campos, o mesmo podendo serem feitos no próprio código do sistema, graças a biblioteca *Pymongo* também utilizada. Esta biblioteca possibilita a conexão do *Python* com *MongoDB*.

A instalação do *Pyro* é simples, bastando apenas seguir o tutorial que encontra-se disponível no seguinte endereço eletrônico: <https://pythonhosted.org/Pyro4/install.html>.

CAPÍTULO 3: IMPLEMENTAÇÃO

O sistema desenvolvido neste trabalho, foi elaborado com dois códigos fontes. Um para a máquina Cliente e o outro para máquina do Servidor, ambos em *Python*. O código do Servidor basicamente instancia um ou mais objetos da Classe “Pessoa” registrando no Pyro Naming Service e Cliente consulta o NS(Naming Service) pela localização desses objetos recebendo a URI passada (URI gerada pelo servidor), cria Proxies remotos atuando como a Classe Pessoa verdadeira podendo invocar seus métodos. Segue abaixo os códigos-fonte:

cliente.py

```
#coding:utf-8
import Pyro4
import collections
# Mensagem na tela para entrar com a URI, use a URI impressa no servidor
uri = raw_input("Entre com a uri: ").strip()
p = Pyro4.Proxy(uri) #instancia que cria proxies para os objetos da URImotos
opcao=True

while opcao:
    #Menu
    print
    """*****
*****"""
    print "|Opções\t| 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar | 4 -> Imprimir Extrato |"
    print
    """*****
*****"""
    option=int(raw_input("Opção: "))

    if option==0:
        print "Saindo"
        opcao = False

    elif option==1:
        n = raw_input("Nome: ")
        s = raw_input("Senha: ")
        print p.imprimir(n, s) #Proxie atuando com classe Pessoa, podendo invocar os métodos nos
objetos remotos

    elif option==2:
        n = raw_input("Nome: ")
        s = raw_input("Senha: ")
        v = input("Valor a ser depositado: ")
        print p.depositar(n, s, v)#Proxie atuando com classe Pessoa, podendo invocar os métodos nos
objetos remotos
        print p.imprimir(n, s)#Proxie atuando com classe Pessoa, podendo invocar os métodos nos
objetos remotos

    elif option==3:
        n = raw_input("Nome: ")
        s = raw_input("Senha: ")
```

```

v = input("Valor do saque: ")
print p.sacar(n, s, v)#Proxie atuando com classe Pessoa, podendo invocar os métodos nos objetos
remotos
print p.imprimir(n, s)#Proxie atuando com classe Pessoa, podendo invocar os métodos nos
objetos remotos

elif option==4:
    n = raw_input("Nome: ")
    s = raw_input("Senha: ")
    print
    """*****"""
    print "|Data Movim.   \t| Histórico\t| Valor \t| Saldo   \t|"
    for x in p.gerar_extrato(n, s):
        print "|",x['Data'],"\t| ",x['tipo'],"\t| ",x['valor']," \t| ",x['saldo_atual'],"\t| "

print"*****"""

```

Servidor.py

```

#coding:utf8
import Pyro4
from pymongo import Connection
import time

class Pessoa(object):

    @classmethod
    def autentica(self, n, s):
        con = Connection('localhost')
        db = con['Banco']
        self.db = db
        if(db.pessoa.find({"name": n, "senha": s}).count() > 0 ):
            users = db.pessoa.find({"name": n})
            self.p = users[0]
            return users[0]
        else:
            return False

    @classmethod
    def autentica_deposito(self, n, s, val): #to do
        con = Connection('localhost')
        db = con['Banco']
        self.db = db
        if(db.pessoa.find({"name": n, "senha": s}).count() > 0 ): #verifica se existe
            users = db.pessoa.find({"name": n}) #resultado json em user
            p = users[0] #linha encontrada passada para p. p = {"Chave":"valor"...}
            idobj = p['_id']
            val = p['conta']['saldo'] + val #acréscimo do valor depositado a saldo
            db.pessoa.update( { "_id" : idobj}, { "$set":{ "conta" : { "saldo": val}}} ) #atualização do
            valor no banco
            return users[0]

```

```
else:  
    return False
```

```
@classmethod  
def autentica_saque(self, n, s, val): #to do  
    con = Connection('localhost')  
    db = con['Banco']  
    self.db = db  
    if(db.pessoa.find({"name": n, "senha": s}).count() > 0 ): #verifica se existe  
        users = db.pessoa.find({"name": n}) #resultado json em user  
        p = users[0] #linha encontrada passada para p. p = {"Chave":"valor"...}  
        idobj = p['_id']  
        val = p['conta']['saldo'] - val #acréscimo do valor depositado a saldo  
        db.pessoa.update( { "_id" : idobj }, { "$set": { "conta" : { "saldo": val } } }) #atualização do  
valor no banco  
        return users[0]  
    else:  
        return False
```

```
@classmethod  
def registrar_log(self, tipo, valor, n, s):  
    con = Connection('localhost')  
    db = con['Banco']  
    if(db.pessoa.find({"name": n, "senha": s}).count() > 0 ): #verifica se existe  
        users = db.pessoa.find({"name": n}) #resultado json em user  
        p = users[0] #linha encontrada passada para p. p = {"Chave":"valor"...}  
        idobj = p['_id']  
        data = time.strftime("%Y-%m-%d %H:%M:%S")  
        if(tipo == "Deposito"):  
            saldo_atual = p['conta']['saldo'] + valor  
        else:  
            saldo_atual = p['conta']['saldo'] - valor  
        db.log_transacoes.insert( { "id_cliente": idobj, "tipo": tipo, "valor": valor, "Data" : data,  
"saldo_atual" : saldo_atual } )  
        return "!"  
    else:  
        return "!Falha de registro de extrato"
```

```
def __init__(self):  
    self.erro = "Login e/ou Senha incorreto(s)"
```

```
def imprimir(self, n, s):  
    if Pessoa.autentica(n, s) != False:  
        return self.p['conta']['saldo'] #retorno do saldo  
    else : return self.erro
```

```
def depositar(self, n, s, valor):  
    Pessoa.registrar_log("Deposito", valor, n, s)  
    if Pessoa.autentica_deposito(n, s, valor) != False:  
        return "Deposito Efetuado com sucesso"
```

```
def sacar(self, n, s, valor):
```

```
Pessoa.registrar_log("Saque", valor, n, s)
if Pessoa.autentica_saque(n, s, valor) != False:
    return "Saque efetuado com sucesso"
```

```
def gerar_extrato(self, n, s):
    con = Connection('localhost')
    db = con['Banco']
    self.db = db
    if Pessoa.autentica(n, s) != False:
        id_cli = self.p['_id'] #retorno do saldo
        users = db.log_transacoes.find({"id_cliente": id_cli})
        b = []
        for a in users:
            b.append(a)
        return b
    else : return self.erro
```

----- Código padrão encontrado na documentação da biblioteca -----

```
daemon = Pyro4.Daemon('192.168.43.5', 5000) #instancia que contém a lógica do lado do servidor
e distribui os métodos remotos e as chamadas recebidas para os objetos apropriados. A instancia é
contruída passando o IP e a porta para execução do sistema.
uri = daemon.register(Pessoa())#Registra um objeto Pyro com o identificador fornecido. O objeto é
agora conhecido apenas dentro deste daemon, não está automaticamente disponível em um servidor
de nomes. Este método retorna uma URI para o objeto registrado.
print "uri=",uri
daemon.requestLoop()#loop para atender as solicitações de entrada.
```

```
Pyro4.Daemon.serveSimple( # inicia um servidor Daemon com a classe Pessoa
{
    Pessoa(): None
},
ns=False, verbose=True)
```

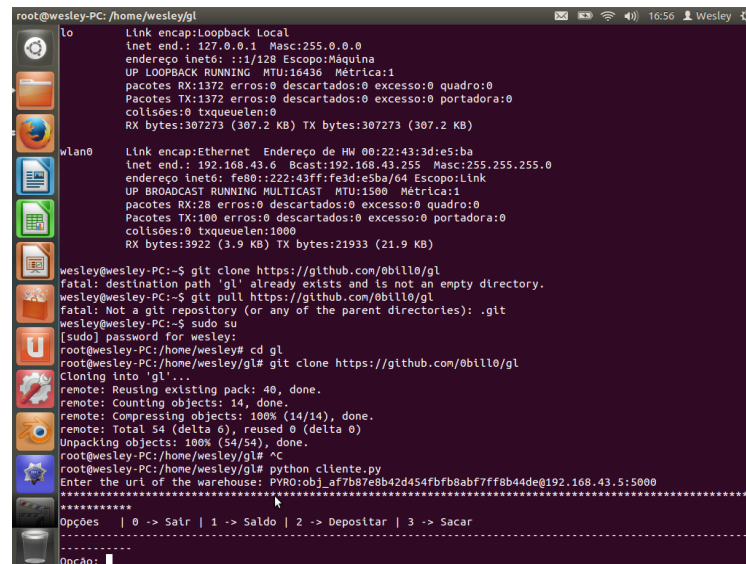
O sistema possui apenas dois requisitos, Instalação do *Python* e *Pyro4* (ambos livres e gratuitos). Após a instalação, basta adicionar algumas linhas ao código-fonte `servidor.py` para configuração de comunicação em rede (IP e porta), conforme segue abaxaio.

```
daemon = Pyro4.Daemon('<IP>', <porta>)
```

Dessa forma, podemos ver que ao executarmos o programa principal, este atuará como um servidor, mostrando sua *uri*, que é seu identificar único.

4- Análise dos Resultados

Todos os nossos testes foram feitos com máquina sendo cliente sistema operacional Linux-Ubuntu 12.04 e outra sendo o servidor com sistema operacional Linux-Ubuntu 12.10 , CPU Intel® Core™ i3-3110M CPU @ 2.40GHz × 4 e Memória de 5,7 GiB. Com os testes feitos ficou evidente que a linguagem *Python* somada ao framework *Pyro4* e a biblioteca para acesso ao banco de dados não-relacional *MongoDB*, tiveram uma surpreendente performance. Não houve alterações consideráveis no nível de processamento no cliente (como já era esperado por nós) e também no servidor, mesmo após inserção de uma quantidade de dados um pouco maior que a inicial, algumas dezenas de linhas nas tabelas do banco. Abaixo as figuras que comprovam a afirmação acima:

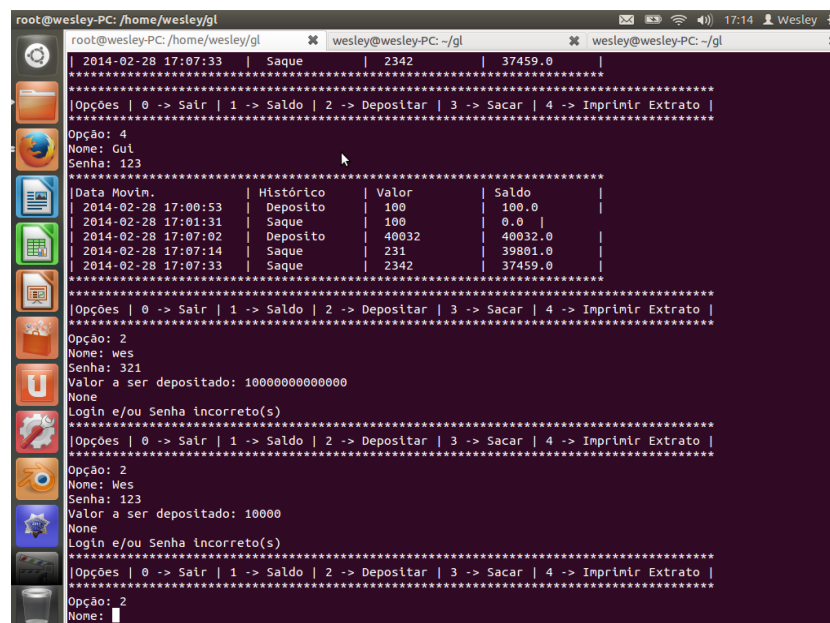


```
root@wesley-PC: /home/wesley/gl
lo      Link encap:Loopback Local
        inet end.: 127.0.0.1  Masc:255.0.0.0
        endereço inet6: ::1/128 Escopo:Máquina
        UP LOOPBACK RUNNING  MTU:16436  Métrica:1
        pacotes RX:1372 erros:0 descartados:0 excesso:0 quadro:0
        Pacotes TX:1372 erros:0 descartados:0 excesso:0 portadora:0
        colisões:0 txqueuelen:0
        RX bytes:307273 (307.2 KB) TX bytes:307273 (307.2 KB)

wlan0   Link encap:Ethernet Endereço de HW 00:22:43:d5:ba
        inet end.: 192.168.43.0 Bcast:192.168.43.255 Masc:255.255.0
        endereço inet6: fe80::222:43ff:fe3d:e5ba/64 Escopo:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Métrica:1
        pacotes RX:28 erros:0 descartados:0 excesso:0 quadro:0
        Pacotes TX:100 erros:0 descartados:0 excesso:0 portadora:0
        colisões:0 txqueuelen:1000
        RX bytes:3922 (3.9 KB) TX bytes:21933 (21.9 KB)

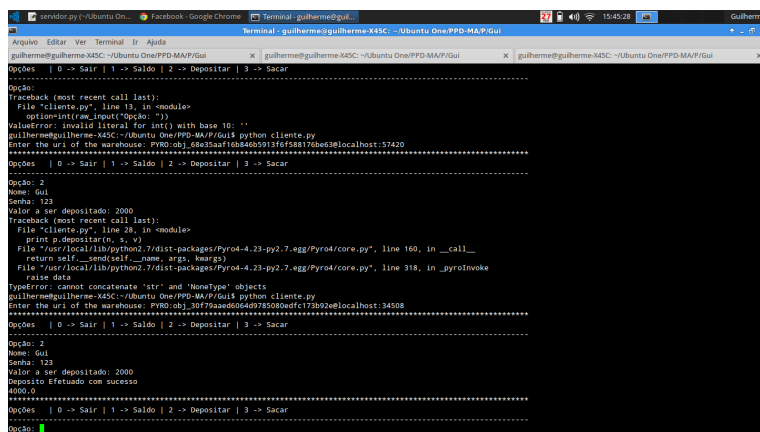
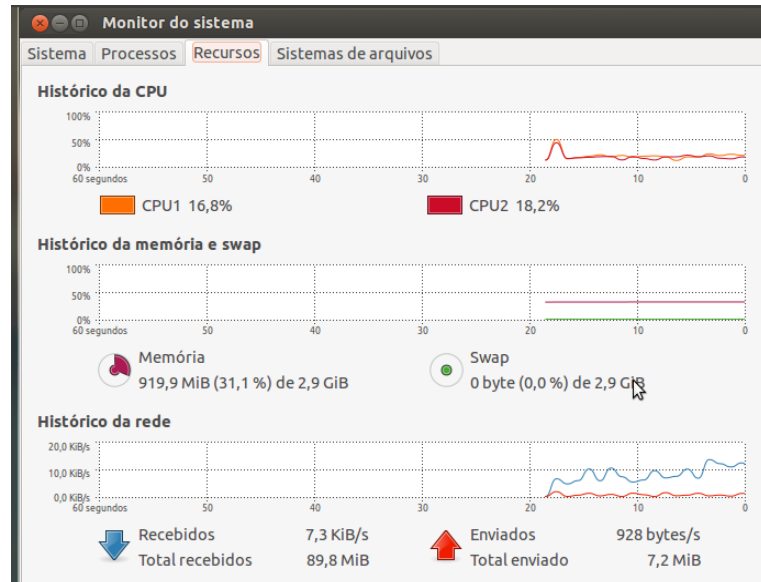
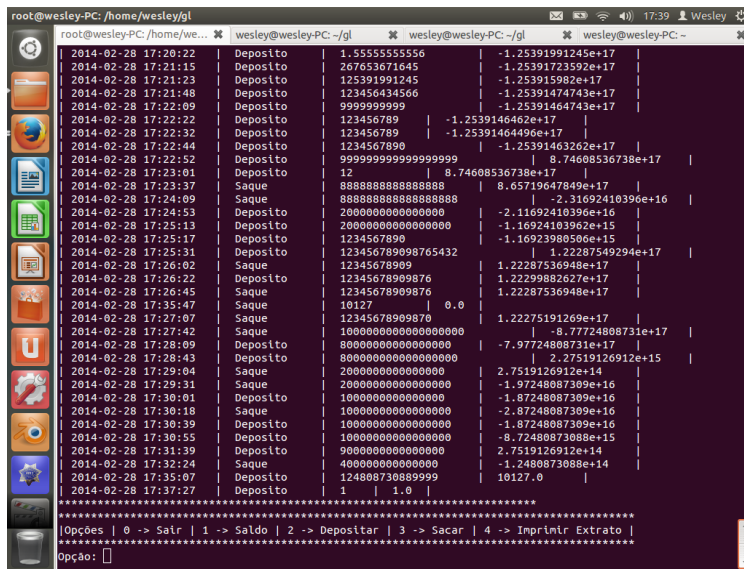
wesley@wesley-PC:~$ git clone https://github.com/0billo/gl
fatal: destination path 'gl' already exists and is not an empty directory.
wesley@wesley-PC:~$ git pull https://github.com/0billo/gl
fatal: Not a git repository (or any of the parent directories): .git
wesley@wesley-PC:~$ sudo su
[sudo] password for wesley:
root@wesley-PC: /home/wesley# cd gl
root@wesley-PC: /home/wesley/gl# git clone https://github.com/0billo/gl
Cloning into 'gl'...
remote: Reusing existing pack: 40, done.
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 54 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (54/54), done.
root@wesley-PC: /home/wesley/gl# ^C
root@wesley-PC: /home/wesley/gl# python cliente.py
Enter the url of the warehouse: PYRO:obj_af7b87eb42d454f8b4bf7ff8b44de@192.168.43.5:5000
*****
Opções | 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar
*****
Opção: 1
```

Cliente - Primeiro acesso remoto ao sistema.



```
root@wesley-PC: /home/wesley/gl
root@wesley-PC: /home/wesley/gl  wesley@wesley-PC: ~/gl  wesley@wesley-PC: ~/gl
| 2014-02-28 17:07:33 | Saque | 2342 | 37459.0 |
*****
|Opções | 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar | 4 -> Imprimir Extrato |
*****
Opção: 4
Nome: Gui
Senha: 123
*****
|Data Movim. | Histórico | Valor | Saldo |
| 2014-02-28 17:00:53 | Deposito | 100 | 100.0 |
| 2014-02-28 17:01:31 | Saque | 100 | 0.0 |
| 2014-02-28 17:07:02 | Deposito | 40032 | 40032.0 |
| 2014-02-28 17:07:14 | Saque | 231 | 39801.0 |
| 2014-02-28 17:07:33 | Saque | 2342 | 37459.0 |
*****
|Opções | 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar | 4 -> Imprimir Extrato |
*****
Opção: 2
Nome: Wes
Senha: 123
Valor a ser depositado: 100000000000000
None
Login e/ou Senha incorreto(s)
*****
|Opções | 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar | 4 -> Imprimir Extrato |
*****
Opção: 2
Nome: Wes
Senha: 123
Valor a ser depositado: 10000
None
Login e/ou Senha incorreto(s)
*****
|Opções | 0 -> Sair | 1 -> Saldo | 2 -> Depositar | 3 -> Sacar | 4 -> Imprimir Extrato |
*****
Opção: 2
Nome: 1
```

Cliente – Inserção de senha incorreta



5 – Referências

<http://www.slideshare.net/hyun/shlideshare>

Python disponível em: < <http://pt.wikipedia.org/wiki/Python> > .Acesso em: 24 fev. 2014.

<http://www.profissionaisti.com.br/2009/01/10-motivos-para-voce-aprender-a-programar-em-python/>

Comunidade ubuntu Brasil disponível em:<ubuntu-br.org>. Acesso em 24 fev. 2014.

Pyro introdução disponível em:<<http://pythonhosted.org/Pyro4/>>. Acesso em 24 fev.2014

Pyro, criando um cliente disponível em:

<<http://pythonhosted.org/Pyro4/clientcode.html>> Acesso em 24 fev. 2014

Pyro, criando um servidor disponível

em:<<http://pythonhosted.org/Pyro4/servercode.html>>

Pyro, definindo o nome do

servidor:<<http://pythonhosted.org/Pyro4/nameserver.html>> Acesso em 24 fev. 2014