Software Validation, Verification and Testing

Term Project Report

Ahmet Oral

GitHub Repo

Objective:

Test automation with Selenium Web Driver

Website:

Döviz.com

Tasks:

1. Define a virtual balance account in your program including dollar, Euro, TL and gold sub accounts.

2. Initial accounts include: TL: 100000 TL Dollar: 0 USD Euro: 0 EUR Gold: 0 grams

3. Check döviz.com web page for these values and note the buying prices for them.

4. Then at every 1 hour your program should check the buying prices and selling prices of each.

5. Below algorithm will be executed at every one hour: İf one of the Exchange type (say dollar) buying price is %1 lower than the previous noted rate and the Turkish lira account has more than 1000 TLs, then buy 10 dollars and update the Exchange rate. If one of the Exchange type (say dollar) selling price is %0.5 higher than the previous noted rate and the dollar account has more than or equal to 5 dollars, then sell 5 dollars and update the Exchange rate.

6. Calculate hourly balance by using lately noted Exchange rates.

7. At the end of the day if your balance is more than 100.000 TLs, than print out "YOU WIN!", else print out "YOU'RE LOSER!"

Tools Used:

Python 3.8 – IDE Pycharm 2020.2.5 – Selenium Web Driver – Google Firebase

Code Explanation:

Step 1,2

- Define a virtual balance account in your program including dollar, Euro, TL and gold sub accounts.

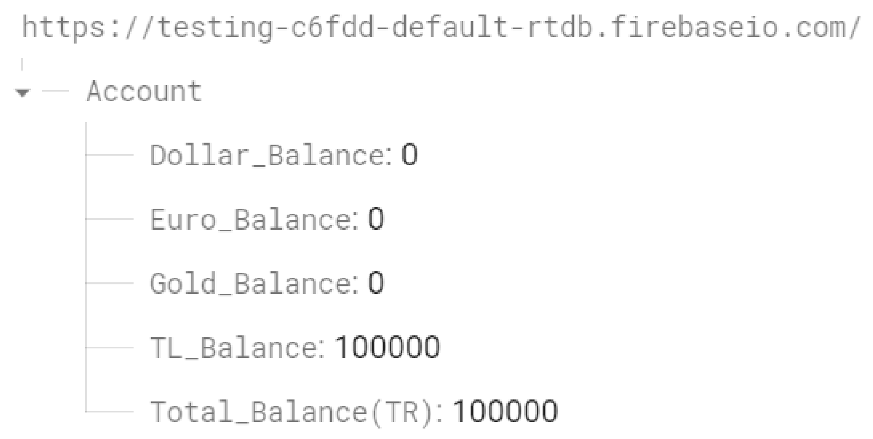
- Initial accounts include: TL: 100000 TL Dollar: 0 USD Euro: 0 EUR Gold: 0 grams

For defining a virtual balance account, we used Google Firebase Database. Code below shows defining the parameters and adding this data to the Firebase Database. We create 4 different balance account that includes Tl, Dollar, Euro and Gold. We also have a parameter called Total Balance(TR), which now equals to amount of TL we have, in the future it will be calculated using current exchange rates and amount of assets in the account.

```
# Initializing the Database
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()

# Creating dummy bank account with given parameters and uploading it to the database
data = {"TL_Balance": 100000, "Dollar_Balance": 0, "Euro_Balance": 0,
"Gold_Balance": 0,
        "Total_Balance(TR)": 100000}
db.child("Account").set(data)
```

Here is the Firebase Database after creating the balance account:



```
https://testing-c6fdd-default-rtdb.firebaseio.com/
Account
    Dollar_Balance: 0
    Euro_Balance: 0
    Gold_Balance: 0
    TL_Balance: 100000
    Total_Balance(TR): 100000
```

Step 3

- Check döviz.com web page for these values and note the buying prices for them.

The code below is how we created our web driver:

```
# Creating webdriver with given options
options = Options()
options.add_argument("--window-size=1920,1080")
driver = webdriver.Chrome(options=options)
driver.get("https://www.döviz.com/")
```

After opening our website, we need the Xpath variables for getting these values. From the code below, you can see that we found the Xpath for all of them and assigned these paths to relating variables.

```
# Xpath for prices of Dollar, Euro and Gold
dollarBuyPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[1]/td[3]"
dollarSellPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[1]/td[4]"

EuroBuyPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[2]/td[3]"
EuroSellPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[2]/td[4]"

GoldBuyPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[3]/td[3]"
GoldSellPriceXpath = "/html/body/main/div/div/div/table/tbody/tr[3]/td[4]"
```

Because the algorithm will be in a loop that checks for the values every once in a defined time arrival, we put these values in a list so we can easily operate on them.

```
# Putting Xpath's to a list so we can scan data using for loop
pricePaths = [dollarBuyPriceXpath, dollarSellPriceXpath, EuroBuyPriceXpath,
EuroSellPriceXpath, GoldBuyPriceXpath,
              GoldSellPriceXpath]
```

After that we need to create a way to separate previous values from the current values so we can check the price difference. To do that we created 2 empty list, one will be created every time we call ‘get_prices()’ function which will store current values, and the other will only be defined once which will store the previous values.

```
# Creating two empty lists for storing current and previous prices of exchanges
previousPrices = []
currentPrices = []

# Function to get all of the values from website with given Xpath's
def get_prices():
    # Because we will keep updating prices, we need to clear the list before
initialization
    currentPrices.clear()
    for i in pricePaths:
        currentPrices.append(driver.find_element(by=By.XPATH, value=i).text)

# Getting previous prices (which is required only at first iteration)
for i in pricePaths:
    previousPrices.append(driver.find_element(by=By.XPATH, value=i).text)
```

Step 4,5,6

- Check the buying prices and selling prices of each exchange rate every hour.

- Implement the buying and selling algorithm.

- Calculate hourly balance by using lately noted Exchange rates.

Here we created our main loop that will be executing all of the operations defined in steps 4,5 and 6.

```python
# Main loop that checks prices every hour and buys or sells according to price changes.
# Then updates the bank account (database) with current exchange prices and completed transactions
# timeCheck is for keeping track of the number of loops
timeCheck = 0
while True:
    timeCheck += 1
    # Refresh the page before executing commands (comment this while testing)
    # driver.refresh()

    # Get current exchange rates
    get_prices()

    # Convert string values to float in each list
    currentPrices = [float(i) for i in currentPrices]
    previousPrices = [float(i) for i in previousPrices]

    # Dictionary to get and store account details from database
    balances = {
        "TL_Balance": db.child("Account").get().val()["TL_Balance"],
        "Dollar_Balance": db.child("Account").get().val()["Dollar_Balance"],
        "Euro_Balance": db.child("Account").get().val()["Euro_Balance"],
        "Gold_Balance": db.child("Account").get().val()["Gold_Balance"],
        "Total_Balance(TR)": db.child("Account").get().val()["Total_Balance(TR)"]
    }

    # Buy 10 Unit (Dollar, Euro or Gold) if buying price is %1 lower than the previous noted rate
    # ...and the Turkish lira account has more than 1000 TLs
    #
-----------------------------------------------------------------------------------------------
    # Sell 5 Unit (Dollar, Euro or Gold) if selling price is %0.5 higher than the previous noted rate
    # ...and the Unit account has more than or equal to 5 dollars
    #
-----------------------------------------------------------------------------------------------
    # Than update account balances according to current exchange rates and completed transactions

    # For Dollar Account
    dollarAccountCheck()
    # For Euro Account
    euroAccountCheck()
    # For Gold Account
    goldAccountCheck()

    # Calculating Total TR balance with new exchange rates according bought or sold units
    # ... and update the bank account (database)
    calculateTotalBalance()

    # Update previous price list
    previousPrices = currentPrices.copy()

    # Wait for 1 hour (3600 seconds)
    time.sleep(5)

    # If loop has run 24 times
    if timeCheck == 24:
        break
```

Each line of code is detailly explained in the code, but basically what it does is:

Calling ‘get_prices()’ function to get current exchange rates. Than convert both price lists to float so we can operate over the elements in these lists. Than we create a dictionary for keeping track of balances.

In this dictionary we get the values directly from database. This way we can be sure that the values in the dictionary are always up to date.

The algorithm described in the step 5 is shown below. There are 3 of these functions which stands for Dollar, Euro and Gold. The one below is for the Dollar rate. It gets the current exchange rate and current account balances than operates over the conditions of:

- Buy 10 Unit (Dollar, Euro or Gold) if buying price is %1 lower than the previous noted rate and the Turkish lira account has more than 1000 TLs #

- Sell 5 Unit (Dollar, Euro or Gold) if selling price is %0.5 higher than the previous noted rate and the Unit account has more than or equal to 5 dollars

```
def dollarAccountCheck():
    if (currentPrices[0] < (previousPrices[0] - previousPrices[0] / 100)) and
(balances["TL_Balance"] > 1000):
        balances["TL_Balance"] = balances["TL_Balance"] - 10 * currentPrices[0]
        balances["Dollar_Balance"] = balances["Dollar_Balance"] + 10
        print("Bought 10 Dollar for: ", currentPrices[0], " each.")
    if currentPrices[1] > (previousPrices[1] + previousPrices[1] / 200) and
(balances["Dollar_Balance"] >= 5):
        balances["TL_Balance"] = balances["TL_Balance"] + 5 * currentPrices[1]
        balances["Dollar_Balance"] = balances["Dollar_Balance"] - 5
        print("Sold 5 Dollar for: ", currentPrices[1], " each.")
```

After we call these functions to perform our operations we move on to step 6, which is calculating total balance with respect to current exchange rates. Here is the function we created for this step:

```
def calculateTotalBalance():
    # Calculating Total TR balance with new exchange rates according bought or sold
units
    balances["Total_Balance(TR)"] = balances["TL_Balance"] +
balances["Dollar_Balance"] * currentPrices[0] + balances[
        "Euro_Balance"] * currentPrices[2] + balances["Gold_Balance"] *
currentPrices[4]
    # Update bank account (database)
    db.child("Account").set(balances)
```

It updates the values in the dictionary by multiplying the amount of a unit and it’s exchange rate. Than takes the sum to calculate total TL balance. After setting up the dictionary, it updates the bank account by pushing these values to the database.

Step 7

- At the end of the day if your balance is more than 100.000 TLs, than print out “YOU WIN!”, else print out “YOU’RE LOSER!”

```
    # If loop has run 24 times
    if timeCheck == 24:
        break

# If  balance is more than 100.000 TLs, than print out “YOU WIN!”, else print out
“YOU’RE LOSER!”
if balances["Total_Balance(TR)"] > 100000:
    print("YOU WIN!")
else:
    print("YOU’RE LOSER!")
```

After our loop has run 24 times, we break out of the loop and check our balance. If balance is more than 100k Tl’s which means we made profit, we print out YOU WIN!, if we lost money, we print out YOU’RE LOSER!

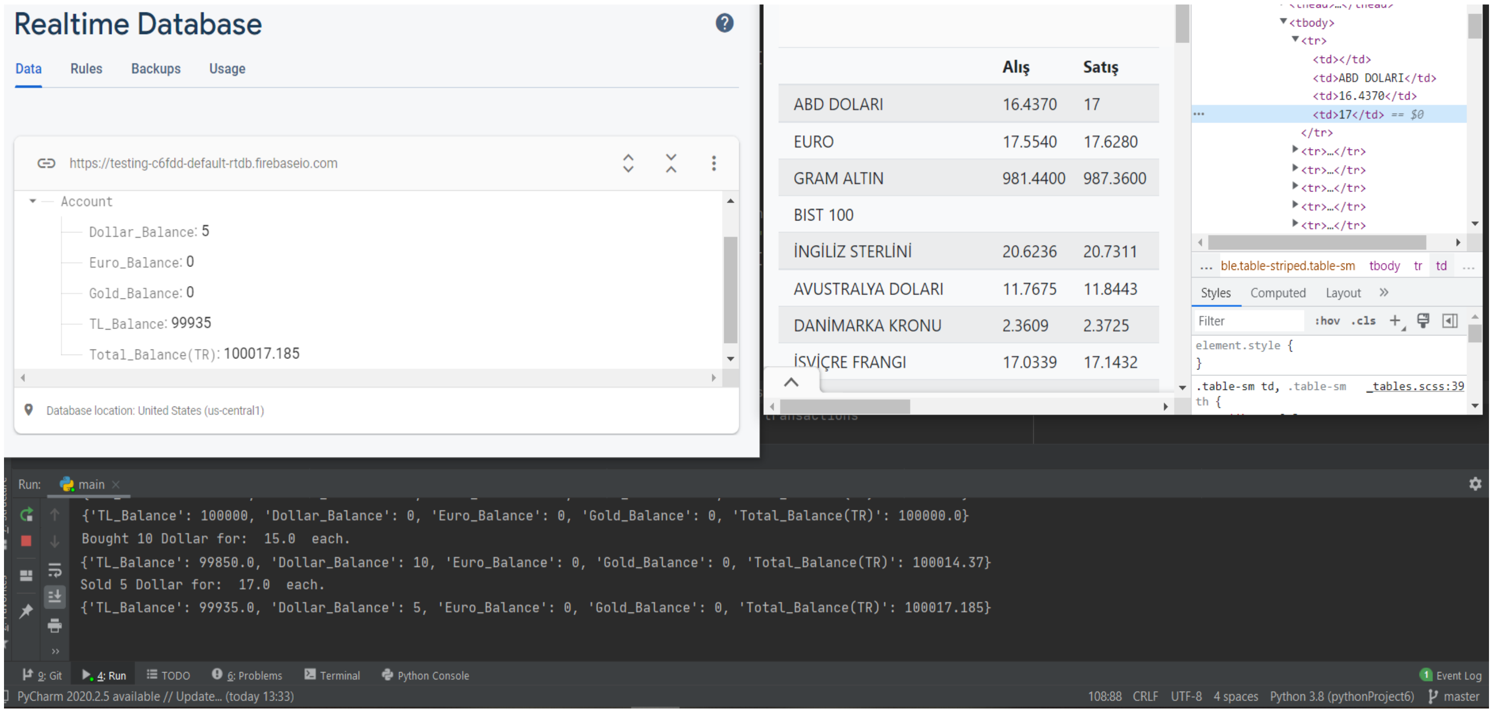
Code Running Example – *Dollar falls to 15, then increases to 17*

Screenshot below show what happens when we change:

- Dollar’s buy rate from 16.43 to 15. (Bought 10 dollars)

- Changing it back to 16.43 (Waited for a price change because conditions aren’t met)

- Than changing it’s sell rate to 17 (Sold 5 dollars)



Realtime Database

https://testing-c6fdd-default-rtdb.firebaseio.com

```
Account
    Dollar_Balance: 5
    Euro_Balance: 0
    Gold_Balance: 0
    TL_Balance: 99935
    Total_Balance(TR): 100017.185
```

Database location: United States (us-central1)

	Alış	Satış
ABD DOLARI	16.4370	17
EURO	17.5540	17.6280
GRAM ALTIN	981.4400	987.3600
BIST 100		
İNGİLİZ STERLİNİ	20.6236	20.7311
AVUSTRALYA DOLARI	11.7675	11.8443
DANİMARKA KRONU	2.3609	2.3725
İSVİÇRE FRANGI	17.0339	17.1432

```
<tbody>
  <tr>
    <td></td>
    <td>ABD DOLARI</td>
    <td>16.4370</td>
    <td>17</td> == $0
  </tr>
  <tr>…</tr>
  <tr>…</tr>
  <tr>…</tr>
  <tr>…</tr>
  <tr>…</tr>
```

```
element.style {
}
.table-sm td, .table-sm th {          _tables.scss:39
```

```
{'TL_Balance': 100000, 'Dollar_Balance': 0, 'Euro_Balance': 0, 'Gold_Balance': 0, 'Total_Balance(TR)': 100000.0}
Bought 10 Dollar for:  15.0  each.
{'TL_Balance': 99850.0, 'Dollar_Balance': 10, 'Euro_Balance': 0, 'Gold_Balance': 0, 'Total_Balance(TR)': 100014.37}
Sold 5 Dollar for:  17.0  each.
{'TL_Balance': 99935.0, 'Dollar_Balance': 5, 'Euro_Balance': 0, 'Gold_Balance': 0, 'Total_Balance(TR)': 100017.185}
```