MINISTRY OF EDUCATION AND SCIENCE OF THE KYRGYZ REPUBLIC

KYRGYZ-GERMAN INSTITUTE OF APPLIED INFORMATICS

**COURSEWORK**

Programming Languages 1

on the subject: «Mobile phone repair services»

Supervisor:A.Kibekbaev(Senior Lecturer)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

English Language Adviser:M.K. Karagulova(Senior Lecturer)\_\_\_\_\_\_\_\_\_\_

English Language Adviser:A.M. Jeenalieva(Senior Lecturer)\_\_\_\_\_\_\_\_\_\_

English Language Adviser:A.S. Asanova(Senior Lecturer)\_\_\_\_\_\_\_\_\_\_\_\_\_

Monitoring Adviser:K.B.Turganov(Senior Lecturer)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Completed by WIN-1-21: Asylbekova A.A.\_\_\_\_\_\_\_\_\_\_\_

Imaraliev B.A.\_\_\_\_\_\_\_\_\_\_\_\_\_

Maasaliev B.J.\_\_\_\_\_\_\_\_\_\_\_\_\_

Bishkek 2021

**Content**

[**Introduction** 3](#_Toc91688784)

[**Charter laws and rights:** 4](#_Toc91688785)

[**Team Roles:** 5](#_Toc91688786)

[**Software Requirements Specification** 6](#_Toc91688787)

[**Architectural Representation** 7](#_Toc91688788)

[**Main Part:** 9](#_Toc91688789)

[**Customer account** 13](#_Toc91688790)

[**Repairer account** 20](#_Toc91688791)

[**Account of a worker** 27](#_Toc91688792)

[**Account of a supplier** 34](#_Toc91688793)

[**Conclusion** 38](#_Toc91688794)

[**References:** 39](#_Toc91688795)

# **Introduction**

**Abstract:** The smartphone repair market is very promising. And the list of competitors is huge. Studying the local market, we found out that in many workshops in the country the level of customer service is extremely low: repair workers break deadlines, overestimate the final cost, use low-quality spare parts. And also in many services, there is no automated system. The main difference in our program is the high level of service. We were required to write a program that would facilitate and improve the work of the workshop. The program should keep records (give information) about the warehouse, employees, customers, and orders. We were required to write a program that would facilitate and improve the work of the workshop.

**Subject**: Phone repair service is the most popular service of all digital electronics repair

service centers. The workshop offers customers phone repair services.

**Goals:** The main goal of our course work is to create a convenient and effective program and database for the mobile service "Mobile phone repair services".

**Objectives:** Automation is an opportunity to speed up the processes of the service and to maintain them thoroughly. Our program gives you the opportunity to:

1. log in or create an account;
2. select from the menu based on the type of user account;
3. give your gadget for repair;
4. check the status of readiness of the order;
5. order the necessary spare;
6. view the lists of parts for maintenance, etc.

**Relevance:**  This topic is very relevant nowadays as cell phones are a necessary part of our life. After all in order to fulfill our daily goals and tasks, we need to be always in touch and have faithful assistant in hand. Any phone can fail and this requires mobile phone repair services. And for large services it is necessary to automate work, since automation makes it possible to reduce human resources and greatly reduce the costs of your work.

**Target group:** The users of the program are a repairman, an employee, a supplier and a client. Each user must have their own account to the system by login and password.

Depending on the account category, the corresponding menu opens to perform further actions.

# **Charter laws and rights:**

1. To receive, on request, limited medical leave for health reasons or for another valid reason.
2. The right to privacy and the non-disclosure of private information, provided that it is not prejudicial to the work.
3. Voluntarily physically attend our meetings in University.
4. Political, religious agitation within our team is prohibited.
5. It is prohibited to leave the team without raising the issue for general discussion.

# **Team Roles:**

Asylbekova Astra:

* Customer’s account developer
* Employee’s account developer
* Documentation
* Presentation

Imaraliev Baysal:

* Supplier’s account developer
* Repairman’s account developer
* Documentation
* Presentation

Maasaliev Bektur:

* Main menu developer
* Automation part developer
* Documentation
* Presentation

# **Software Requirements Specification**

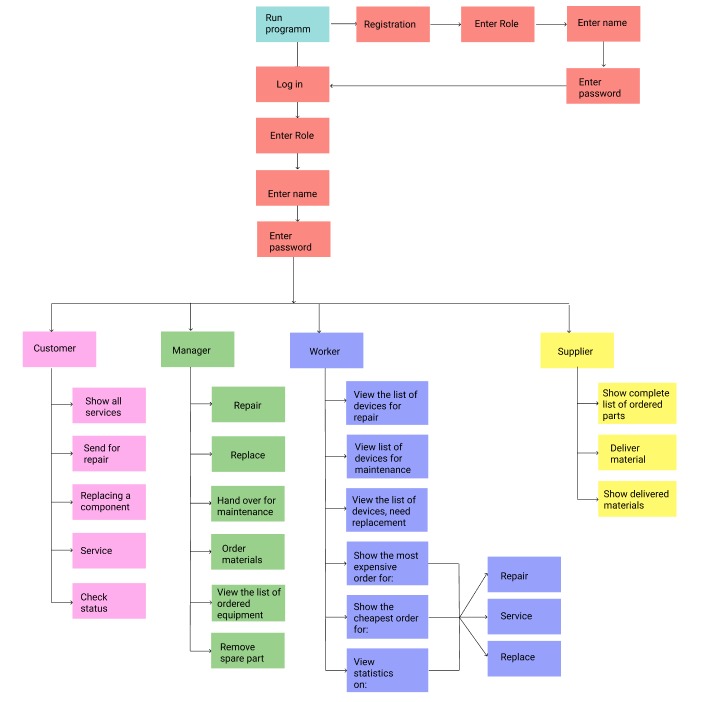
**Used tools**

*Program language:* Python 3.8

*IDE:* PyCharm

*Database:* txt files

# **Architectural Representation**



Pic.1.A Map-diagram

This diagram of our project shows the functions that the program performs. After launching the program, which is the very beginning of the system, it asks the user to log in or register. In case of registration, after creating your own account, the User returns to the entry point, where he needs to enter his data: account, login and password. Depending on the information entered into the system, the corresponding functions are opened:

1. Customer look at all services, give them for repair, give them for replacement of components, give them for maintenance, check the order readiness status by the serial number provided to him in the previous functions

2. Repairman: repairs, replace, hand over for maintenance, order the necessary materials, view the list of ordered equipment, remove the spare part from the list;

3. Worker: view the list of devices for repair, view the list of devices for maintenance, view the list of devices for replacement, show the expensive order, show the cheapest order, view statistics;

4. Supplier: show the complete list of ordered parts, deliver material, show delivered materials.

# **Main Part:**

*import* autorization  
*import* user  
*import* repair  
*import* work  
*import* supply  
  
*def* main():  
 result = autorization.menu\_start()  
 *if*(result[0]=="customer"):  
 user.menu\_start(result[1])  
 *if*(result[0]=="repairer"):  
 repair.menu\_start(result[1])  
 *if*(result[0]=="worker"):  
 work.menu\_start(result[1])  
 *if*(result[0]=="supplier"):  
 supply.menu\_start(result[1])  
   
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()

Pic.2. Main menu

Here’s given the main file of our program. All python files used during the process are imported here. Also, main file include Login and Registration. Everything starts from here.

Context file:

*def* get\_items\_by\_login(*file\_name*, *serial*, *sep*=",", *index*=-1):  
 result = ''  
 *with* open(*file\_name*, encoding='utf-8') *as* file:  
 *for* line *in* file.readlines():  
 *if*(line.split(*sep*)[*index*].replace("\n","") == *serial*):  
 print(line.split(*sep*)[*index*])  
 result = line.split(*sep*)[*index*].replace("\n","")  
 *break* print(result)  
  
 *return* result  
  
*def* write(*file\_name*, *param\_list*, *sep*=","):  
 param\_list = list(map(str,*param\_list*))  
 *with* open(*file\_name*, "a", encoding='utf-8') *as* file:  
 file.write(*sep*.join(*param\_list*)+"\n")  
  
*def* from\_to(*file\_need*,*file\_end*,*line*):  
 delete\_line(*file\_need*, *line*)  
 write(*file\_end*, [*line*])  
  
*def* delete\_line(*file*, *line*):  
 lines=[]  
 *with* open(*file*, encoding='utf-8') *as* f:  
 lines = f.readlines()  
 *with* open(*file*, "w",encoding='utf-8') *as* f:  
 *for* row *in* lines:  
 *if*(row.strip("\n") != *line*.strip("\n")):  
 f.write(row)  
  
*def* get\_lines(*file*):  
 *with* open(*file*, encoding='utf-8') *as* f:  
 *return* f.readlines()   
  
*def* select\_only(*file\_name*, *index*, *sep*=","):  
 *return*[ select.split(*sep*)[*index*]  
 *for* select *in* get\_lines(*file\_name*)   
 ]

Pic.3. Context file

This file contains functions that are used in other files.

Automation:

*import* hashlib  
  
*def* get\_password\_hash(*password*):  
 *return* hashlib.sha256(*password*.encode('utf-8')).hexdigest()

Pic.4. Hash function

As stated above, there are only 4 accounts that are stored in a separate file "users.txt ". In our project, we have installed the "hashlib" module, which implements a common interface for many different secure hashing algorithms. The module includes secure hashing algorithms, such as FIPS SHA1, SHA224, SHA256, SHA384 and SHA512. We used the hashlib.sha256() function, which creates hash objects for user passwords during registration. The hash is then translated into a string using the method .hexdigest.

*def* auth(*login*, *password*, *account\_type*, *file\_lines*):  
 *for* line *in file\_lines*:  
 line = line.split()  
 *if*(line[0]==*login and* line[1]==get\_password\_hash(*password*) *and* line[2]==*account\_type*.lower()):  
 *return True  
 return False  
  
def* menu\_start():  
 lines = []  
 account\_types = ["customer", "repairer", "worker", "supplier"]  
  
 enter = input("1. Login \ 2. Registration : ")  
  
 line = '----------------------------'  
  
 *if* enter == '2':  
 register()  
  
 *with* open("users.txt", encoding='utf-8') *as* file:  
 lines = file.readlines()  
  
 *while True*:  
 print()  
 print("To run the program, please enter your account type: >>>")  
 print(line)  
 print("1.Customer")  
 print(line)  
 print("2.Repairer")  
 print(line)  
 print("3.Worker")  
 print(line)  
 print("4.Supplier")  
 print(line)  
   
 account\_type = input("Enter your account type: ")  
   
 *if* account\_type == '1':  
 acc\_type = 'customer'  
 *elif* account\_type == '2':  
 acc\_type = 'repairer'  
 *elif* account\_type == '3':  
 acc\_type = 'worker'  
 *elif* account\_type == '4':  
 acc\_type = 'supplier'  
 *else*:  
 print("Sorry, but we did not find this type of account, please repeat")  
 *continue* login = input("Enter your login:")  
 password = input("Enter your password:")  
  
 result = auth(login, password, acc\_type, lines)

Pic.5. Part of automation

First of all, the system asks the User whether it wants to log in or register.



Pic.6. Run Programm

When selecting 1(Pic.6.):

Entrance (Pic.5.):

You need to enter your user type. If you enter incorrect information, the system will ask you to re-enter your user type, it will continue to ask for re-entry until the user enters the correct information. After that, you will need to enter your username and password.

If the username and password match, then you will successfully log in, otherwise the system will display: "Invalid username or password, please try again" and the system will start working from the beginning, you will return to the account type selection point.

*def* register():  
 account\_types = ["customer", "repairer", "worker", "supplier"]  
 line = '----------------------------'  
 print("1.Customer")  
 print(line)  
 print("2.Repairer")  
 print(line)  
 print("3.Worker")  
 print(line)  
 print("4.Supplier")  
 print(line)  
 account\_type = input("Enter your account type:")  
  
 acc\_type = ''  
 *if* account\_type == '1':  
 acc\_type = 'customer'  
 *elif* account\_type == '2':  
 acc\_type = 'repairer'  
 *elif* account\_type == '3':  
 acc\_type = 'worker'  
 *elif* account\_type == '4':  
 acc\_type = 'supplier'  
 *else*:  
 print("Sorry, but we did not find this type of account, please repeat")  
 *return* login = input("Enter your login:")  
 password = input("Enter your password:")  
  
 *with* open("users.txt", "a", encoding='utf-8') *as* file:  
 file.write(f"{login.lower()} {hashlib.sha256(password.encode('utf-8')).hexdigest()} {acc\_type.lower()} \n")  
  
 *with* open("users-without-hash.txt", "a", encoding='utf-8') *as* file:  
 file.write(f"{login.lower()} {password} {acc\_type.lower()} \n")  
  
  
  
*if* (\_\_name\_\_ == "\_\_main\_\_"):  
 register()

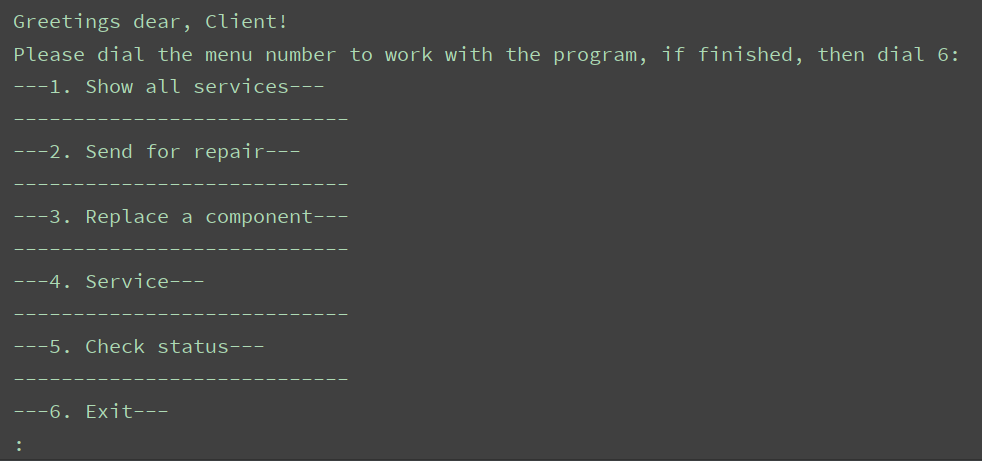
Pic.7. Register

When selecting :

Registration :

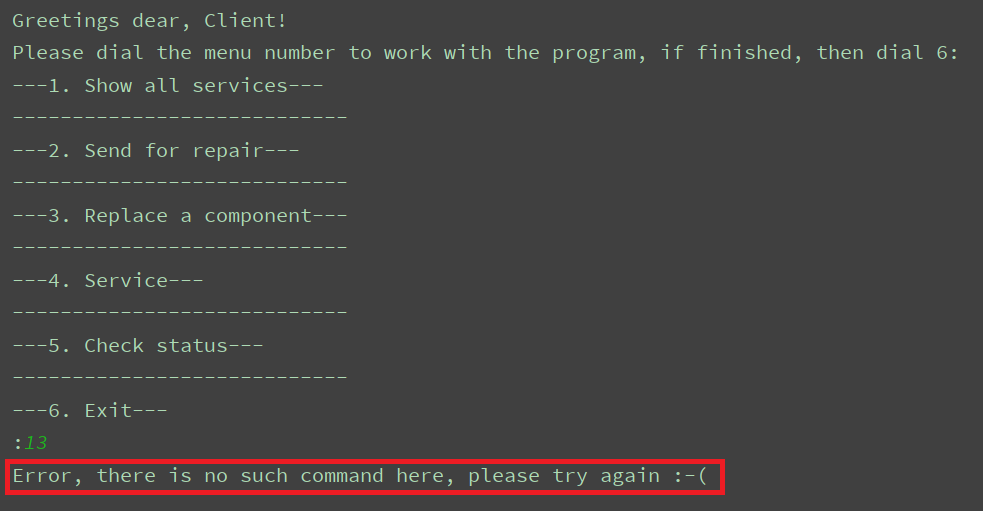
When registering, the User must enter a category from the accounts listed to him, or the system will output: "Sorry, but we did not find this type of account, please repeat" and ask to re-enter the account type. Then the login and password are requested by which he will log in in the future. The account type , User login and its hashed password are stored in the file "users.txt ".

# **Customer account**



Pic.8. Main menu of customer’s account.

A customer will be thrown to the main menu after authorization. The main menu of that account consist of 6 commands (functions). The user should enter a number from 1 to 6 to start the process. Unless the client will follow the conditions, he/she will be stuck in a loop.



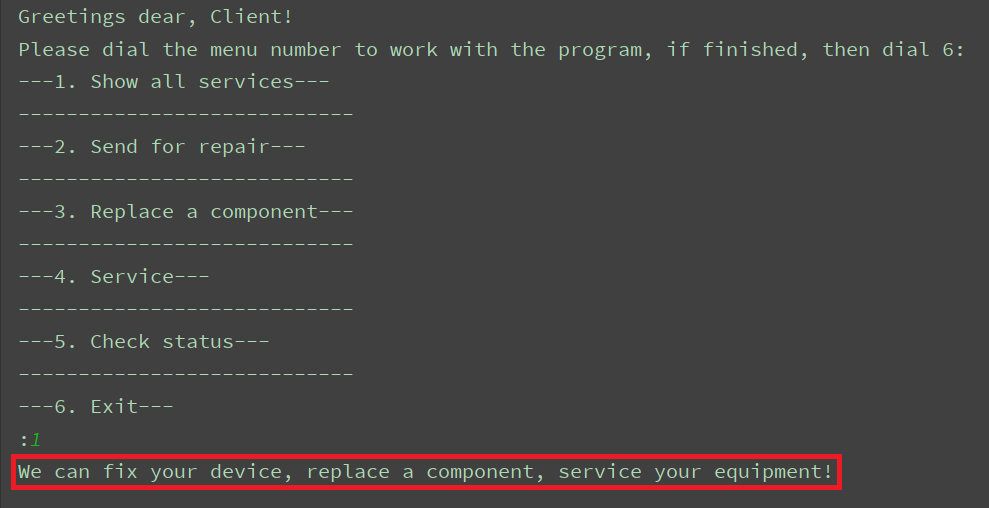
Pic.9. Command (function) selection error.

As we have noticed earlier, the main menu includes 6 commands: 1) Show all services,

2) send for repair, 3) replace a component, 4) service, 5) check status, 6) exit. Next we are going to parse all used functions and files below.

First we are going to show how it works, and then we’ll proceed to parsing of the code.(Pic.2.)

1. Show all services



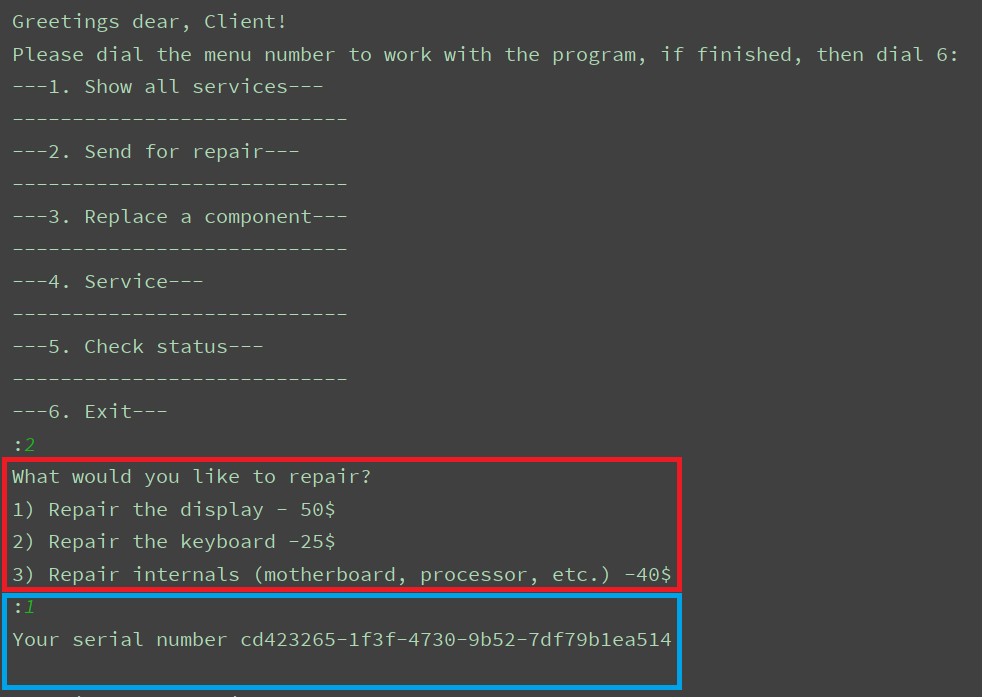
Pic.10. First command in the main menu

First command just shows all services the user can get in here.

**def show\_all\_services():** print**("We can fix your device, replace a component, service your equipment!")**

Pic.11. The function which shows all services.

This simple function just prints a sentence given in brackets.

1. Send for repair

Pic.12. Second command in the main menu

Second command is responsible for sending and repairing of the customer’s device.

The user should first choose a spare he/she wants to repair, and then he/she will be given a serial number. Serial numbers act like id’s.

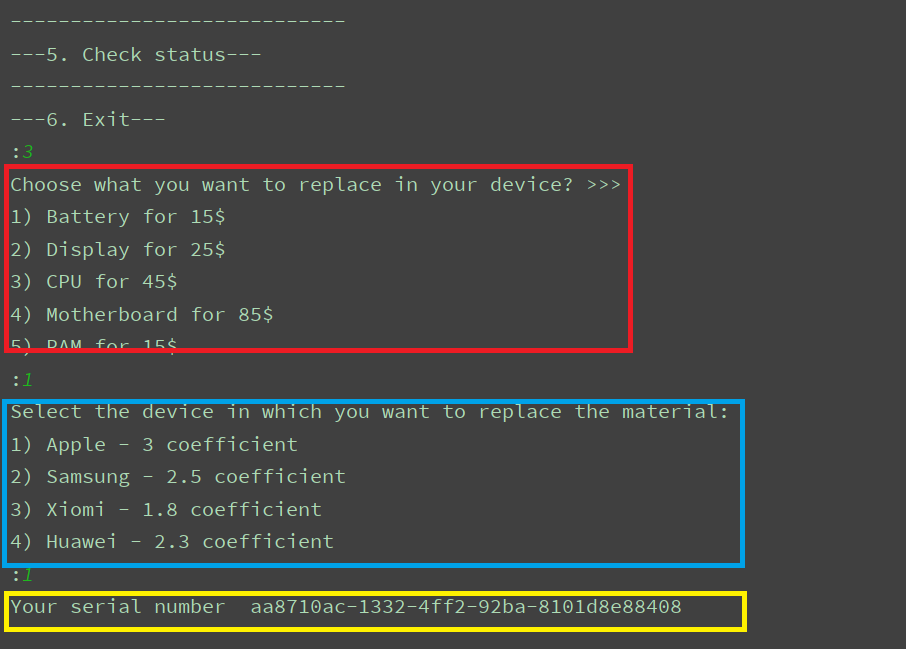
**def give\_to\_repair():** print**("What would you like to repair?")** print**("1) Repair the display - 50$")** print**("2) Repair the keyboard -25$")** print**("3) Repair internals (motherboard, processor, etc.) -40$")** choose **=** int**(**input**(":"))  
  
 if (**choose **>** 3 **and** choose **<** 1**):** print**("There is no such service")  
 return** services **= [  
 ("Repair the display"**, 50**)**,  
 **("Fix the keyboard"**, 25**)**,  
 **("Fix the insides "**, 40**)  
 ]** id **=** uuid.uuid4**()** service **=** services**[**choose**-**1**][**0**]** price **=** services**[**choose **-** 1**][**1**]** context.write**("repair-needed.txt"**, **[**service, price, login, id**])**

print**("Your serial number"**, id**)**

Pic.13. The function that is responsible for the second command in the main menu.

A function first prints a couple of sentences to make the user understand what to do next. The customer should choose one of given variants (enter a number from 1 to 3). Unless he/she enters an appropriate number, he/she will be thrown to the beginning. Uuid is used to create a unique serial number. Next, the functions writes received data which are variables: service, price, login, id and writes it inside a txt-file called ‘repair-needed.txt’. At the of the process the function prints the customer’s serial number. As you can see at the screenshot above, def give\_to\_repair() first writes the customer’s task, price that was paid for the task, customers login and then his/her serial number. As were noticed earlier, customer’s serial number is unique and it will play an important role in searching of tasks etc.

1. Replace a component



Pic.14. The third command in the main menu

With the help of the third command in the main menu the customer can choose a spare he/she wants to replace in his/her device. After the customer chooses a spare part, he/she should choose an appropriate model. At the end, the user will be given a serial number to use in future to check state of his/her device.

**def change():** print**("Choose what you want to replace in your device? >>>")** services **= [  
 ("Battery"**, 15**)**,  
 **("Display"**, 25**)**,  
 **("CPU"**,45**)**,  
 **("Motherboard"**, 85**)**,  
 **("RAM"**, 15**)  
 ]  
  
 for** i,v **in** enumerate**(**services**):** print**(f"{**i**+**1**})"**,v**[**0**]**,**f"for {**v**[**1**]}$")** choose **=** int**(**input**(":"))  
  
 if (**choose **>** 5 **and** choose **<** 1**):** print**("There is no such service")  
 return** id **=** str**(**uuid.uuid4**())** service **=** services**[**choose**-**1**][**0**]** price **=** services**[**choose**-**1**][**1**]** print**("Select the device in which you want to replace the material:")** print**("1) Apple - 3 coefficient")** print**("2) Samsung - 2.5 coefficient")** print**("3) Xiomi - 1.8 coefficient")** print**("4) Huawei - 2.3 coefficient")** choose **=** int**(**input**(":"))** models **= [("Apple"**,3**)**, **("Samsung"**, 2.5**)**, **("Xiomi"**, 1.8**)**, **("Huawei"**, 2.3**)]  
  
 if (**choose **>** 4 **and** choose **<** 1**):** print**("There is no such service")  
 return** model **=** models**[**choose**-**1**][**0**]** price **=** int**(**models**[**choose**-**1**][**1**]\***price**)** context.write**("change\_need.txt"**, **[**model, service, price, login, id**])** print**("Your serial number "**, id**)**

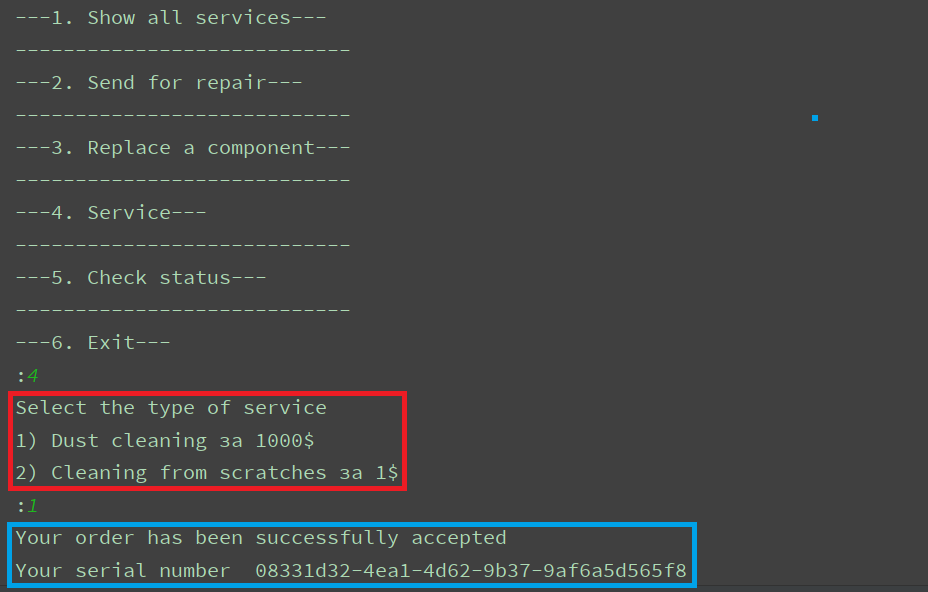
Pic.15. The function, responsible for the third command (def change ())

As you can see, the function prints the value of a list, ’services’, with the help of ‘for’- loop. Next step is choosing a model. It was made with the help of a list, too. Variable ‘id’ is used for serial number. All chosen data were saved in variables: model, service, login, id.The function writes received data in txt-file, ‘change-need.txt’.

As we noticed earlier, the function, def change (), writes first the model of a device, then a spare that need to be replaced, name of the customer and a serial number of that task

Soon, in the fifth command, you will understand why is the serial number is important. In short, a unique serial number simplifies the process of search.

4) Service



Pic.16. The command for services

The fourth command in the main menu is responsible for service. A customer can choose a type of service and then he/she will be given a serial number to know the state of his/her task.

Here’s the function behind:

**def service():** print**("Select the type of service")** services **= [("Dust cleaning"**, 1000**)**,**("Cleaning from scratches"**, 1**)]  
 for** i,serv **in** enumerate**(**services**):** print**(f"{**i**+**1**}) {**serv**[**0**]} за {**serv**[**1**]}$")** service **=** int**(**input**(":"))** print**("Your order has been successfully accepted")** id **=** str**(**uuid.uuid4**())  
 if(**service **>** 2 **and** service **<**1**):** print**("There is no such service")  
 return** context.write**("service-need.txt"**, **[**services**[**service **-** 1**][**0**]**, services**[**service **-** 1**][**1**]**, login,id**])** print**("Your serial number "**, id**)**

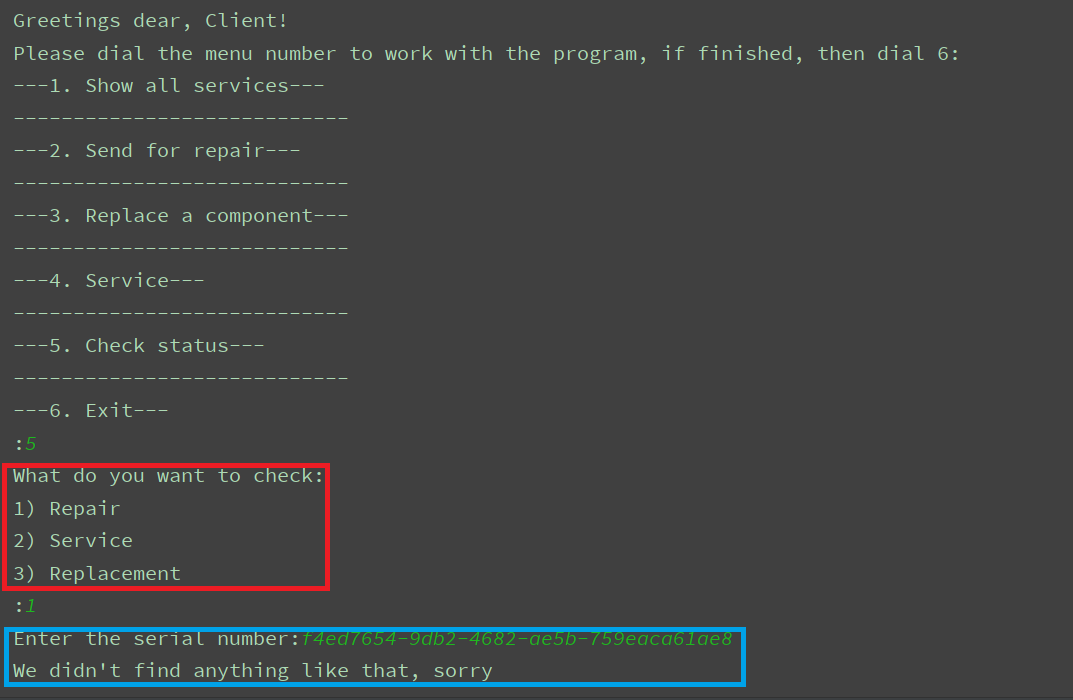
Pic.17. The function responsible for the fourth command

The work of this part of code is very simple. Def service() includes a variable, ‘services’, with services and their prices. The function uses ‘for’ loop to output the value of ‘services’. The customer should enter an appropriate number to start the work of our program, unless he/she will be stuck in a loop. All received data, the function writes in a txt-file, ‘service-need.txt’.

Each line if that txt-file is a service task. Each line consist of a name of the task, price of that task, customer’s name and his serial number.

The serial name will be used to know if the task is completed or not.

5) Check status



Pic.18. Check status

As were noticed before, a serial number plays an important role in a search. The fifth command was made to check the task’s status. The first thing the user should do is choose a type of a task that he/she has given before.

There’re given 2 options of status: Ready and Not yeat

Also, there’s given one variant of answer, if the serial number is incorrect.(An example given above.)

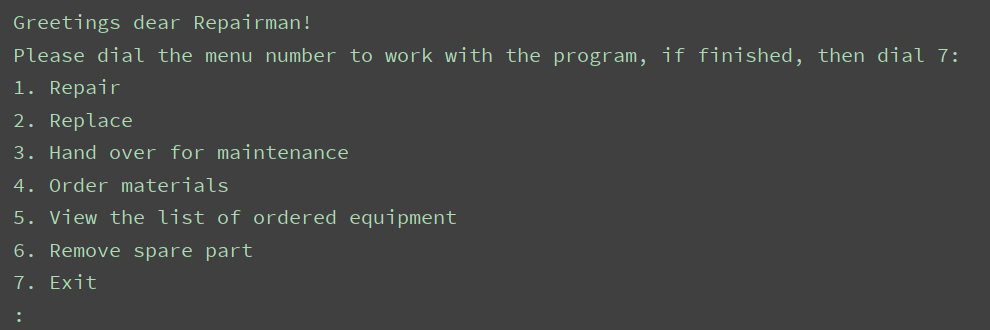
**def model\_in\_file(**serial, file\_needed, file\_end,index**):  
  
 for** row **in** context.get\_items\_by\_login**(**file\_needed, login**):  
 if(**row**[**index**]==**serial**):  
 return "Your technique is not ready yet"  
   
 for** row **in** context.get\_items\_by\_login**(**file\_end, login**):  
 if(**row**[**index**]==**serial**):  
 return "Your technique is ready!"  
  
 return "We didn't find anything like that, sorry"  
   
def check\_status():** print**("What do you want to check:")** print**("1) Repair")** print**("2) Service")** print**("3) Replacement")** section **=** int**(**input**(":"))** serail **=** input**("Enter the serial number:")  
 if(**section**<**4 **and** section**>**0**):  
 if(**section**==**1**):** print**(**model\_in\_file**(**serail, **"repair-needed.txt"**, **"repaired.txt"**,index **=** 3**))  
 if(**section**==**2**):** print**(**model\_in\_file**(**serail, **"service-need.txt"**, **"serviced.txt"**,index**=**3**))  
 if(**section**==**3**):** print**(**model\_in\_file**(**serail, **"change\_need.txt"**, **"changed.txt"**,index**=** 4**))**

Pic.19. Two functions that are responsible for status checking

def check\_status is the function that checks the status of given tasks. At first it needs on of three variants of answer: 1) Repair, 2) Service, 3) Replacement.

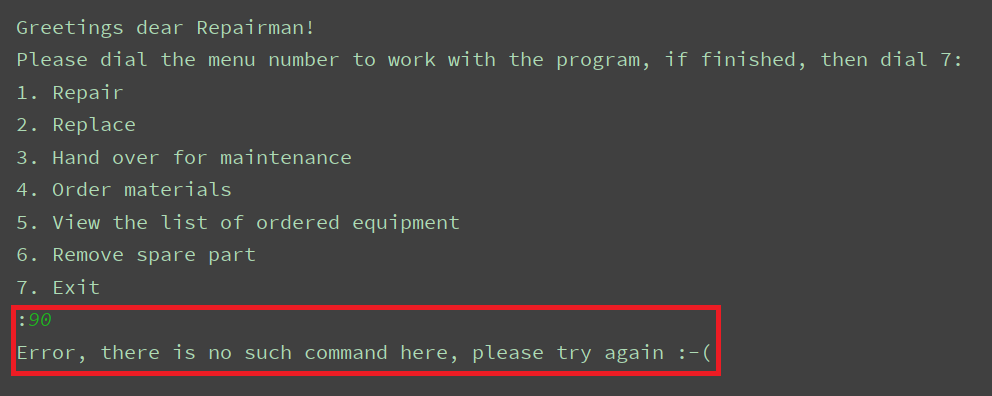
Depending on customer’s anwers a function, def model\_in\_file, will read and check an appropriate txt-file.

# **Repairer account**

****

Pic.20. Main menu of the repairer’s account

A repairer will be thrown to the main menu after authorization. It consists of 6 commands (functions). The repairer should enter a number from 1 to 7 to start the process. Unless he will follow the conditions, he’ll get an error will be stuck in a loop .



Pic.21. Entering error in the main menu

As we have noticed earlier, the main menu includes 6 commands: 1) Repair, 2) replace, 3) hand over for maintenance, 4) order materials, 5) view the list of ordered equipment, 6) remove spare part, 7) exit.

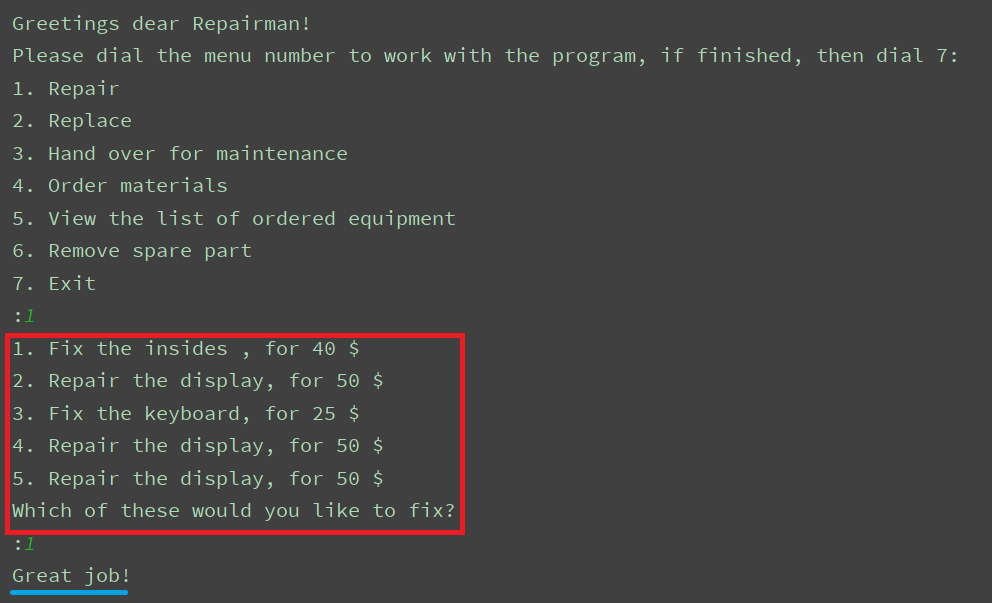
Here’s given the description of each command:

1. Repair – repair a device from a given list of orders
2. Replace – replace a spare part of a device given from the list
3. Hand over for maintenance – make an appropriate service
4. Order materials – add items and their quantity to the list of orders
5. View the list of ordered materials the repairer has ordered before
6. Remove spare part – remove an item from the list of orders
7. Exit – exit and enter the Login/Registration part

Account of the repairer works close with the account of a supplier.

Next we are going to parse all used functions and files below. First we are going to show how it works, and then we’ll proceed to parsing of the code.

1. Repair

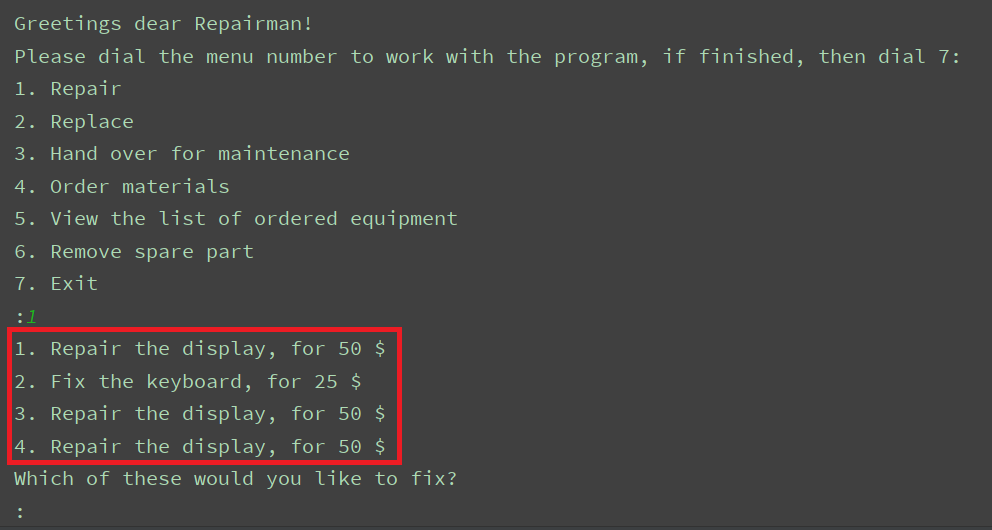


Pic.22. First command in the main menu

The first command shows the list of devices that need to be repaired and price for each task.

The repairer should enter a number depending on the amount of tasks. For example: On the screenshot above we have 5 tasks – he should enter a number from 1 to 5.

After he chooses a task, that task will be noticed as completed and will be removed from the list of tasks.



Pic.23. List of tasks

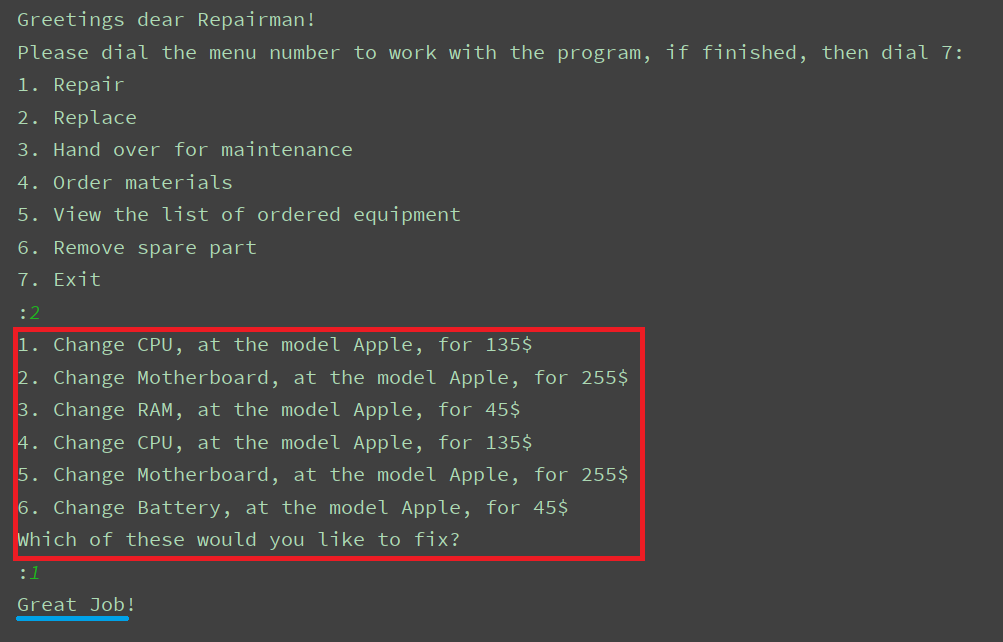
**def repair():** lines **=** context.get\_lines**("repair-needed.txt")  
  
 for** i,line **in** enumerate**(**lines**):** price **=** line.strip**("\n")**.split**(",")[**1**]** row **=** line.strip**("\n")**.split**(",")[**0**]** print**(f"{**i**+**1**}. {**row**}, for {**price**} $")** print**("Which of these would you like to fix?")** index **=** int**(**input**(":"))** context.from\_to**("repair-needed.txt"**, **"repaired.txt"**, lines**[**index **-** 1**])** print**("Great job!")**

Pic.24. Function for the first command

That function reads all lines in a txt-file, called ‘repair-need.txt’ and prints the whole value of it. It uses for - loop to out put the tasks as lines. The method ‘input’ is used to choose the task depending on its index. After that the chosen line will we removed from ‘repair-need.txt’ to ‘repaired.txt’ and a message will be printed.

As you can see account of the repairer works with two more account: customer and supplier. The repairer has access to txt-files related to customer and supplier.

1. Replace



Pic.25. The second command in the main menu

The repairer will be given a list of devices that need a replacement of a spare part and price of each task. As you can see, each task consists of a name of it, model of that device and the price were paid. The repairer should enter a number depending on the amount of tasks. For example: On the screenshot above we have 6 tasks – he should enter a number from 1 to 6.

After he chooses a task, that task will be noticed as completed and will be removed from the list of tasks.

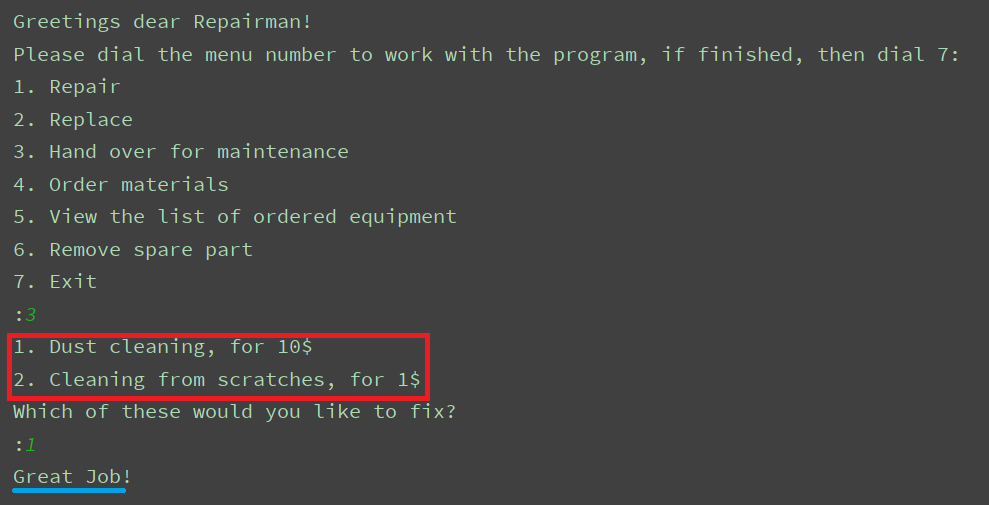
**def change():** lines **=** context.get\_lines**("change\_need.txt")  
  
 for** i,line **in** enumerate**(**lines**):** row **=** line.strip**("\n")**.split**(",")** print**(f"{**i**+**1**}. Change {**row**[**1**]}, at the model {**row**[**0**]}, for {**row**[**2**]}$")** print**("Which of these would you like to fix?")** index **=** int**(**input**(":"))** context.from\_to**("change\_need.txt"**, **"changed.txt"**, lines**[**index **-** 1**])** print**("Great Job!")**

Pic.26. The function for devices need a replacement of a spare

That function is responsible for the work of the second command in the main menu. It reads the whole value of a txt-file, ‘change-need.txt’ and prints it. With the help of for-loop the value of the txt-file will be printed as lines. The method ‘input’ is used to choose the task depending on its index.

After that the chosen line will we removed from ‘change-need.txt’ to ‘changed.txt’ and a message will be printed.

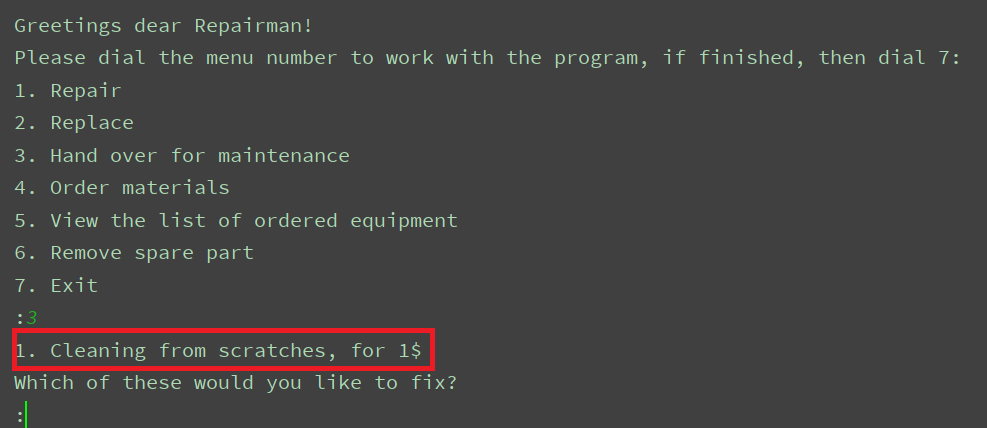
1. Hand over for maintenance



Pic.27. The third command in the main menu

With the help of the third command the repairer can see the list of devices handed over for maintenance and price, paid for the task. First, the repair will be given a list of devices handed over for maintenance, the name of service that customer need, and the price.

The repairer should enter a number depending on the amount of tasks. For example: On the screenshot above we have 2 tasks – he should enter a number from 1 to 2. After he chooses a task, that task will be noticed as completed and will be removed from the list of tasks.



Pic.28. The list of tasks for maintenance

**def service():** lines **=** context.get\_lines**("service-need.txt")  
  
 for** i,line **in** enumerate**(**lines**):** row **=** line.strip**("\n")**.split**(",")** print**(f"{**i**+**1**}. {**row**[**0**]}, for {**row**[**1**]}$")** print**("Which of these would you like to fix?")** index **=** int**(**input**(":"))** context.from\_to**("service-need.txt"**, **"serviced.txt"**, lines**[**index **-** 1**])** print**("Great Job!")**

Pic.29. The function, responsible for the third command

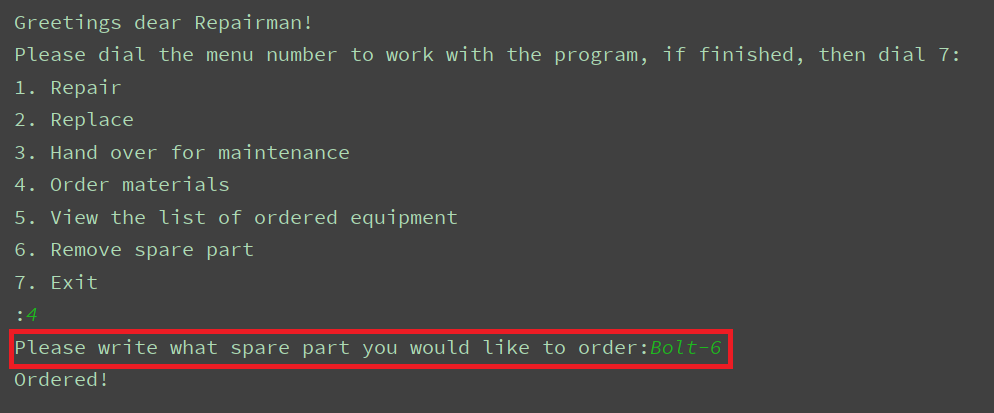
The function reads the whole value of a txt-file, ‘service-need.txt’ and prints it.

With the help of for-loop the value of the txt-file will be printed as lines.

The method ‘input’ is used to choose the task depending on its index.

After that the chosen line will we removed from ‘service-need.txt’ to ‘serviced.txt’ and a message will be printed.

1. Order materials



Pic.30. Order materials

This command helps the repairer to order materials.

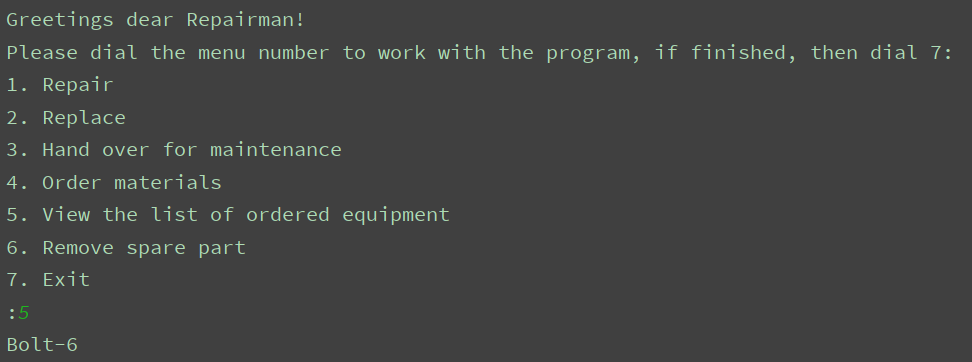
The repairer should type the name of an item he’d likes to order and his order will be added to the list of orders in the account of the supplier.

**def order\_sparepart():** order **=** input**("Please write what spare part you would like to order:")** context.write**("DetailOrdered.txt"**, **[**order, login**])** print**("Ordered!")**

Pic.31. The function for orders

The function gets the order, that were typed by the user and adds it a txt-file, ‘DetailOrdered.txt’.

1. View the list of ordered equipment



Pic.32. Work of the fifth command

The fifth command just shows the list of ordered by the repairer materials.

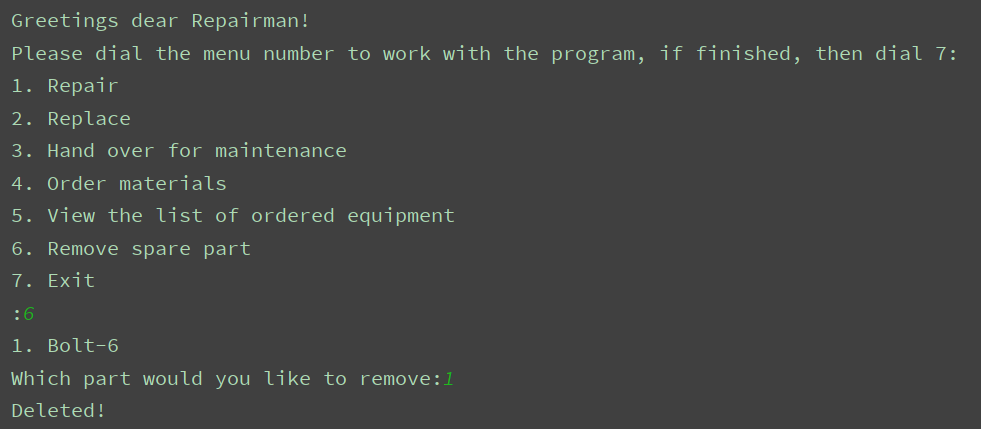
**def show\_sparepart():  
 for** sparepart **in** context.get\_items\_with\_login**("DetailOrdered.txt"**, login**):** print**(**sparepart**[**0**])**

Pic.33. The function, responsible for the fifth command

This function reads a txt-file and with the help of for-loop, prints the value of ‘DetailOrdered.txt’

1. Remove spare part

This command removes a chosen item from the list of ordered materials. The user should just enter an appropriate number to remove.



Pic.34. Work of the sixth command

**def delete\_sparepart():** lines **=** context.get\_items\_with\_login**("DetailOrdered.txt"**, login**)  
 for** i,sparepart **in** enumerate**(**lines**):** print**(f"{**i**+**1**}. {**sparepart**[**0**]}")** index **=** int**(**input**("Which part would you like to remove:"))** context.delete\_line**("DetailOrdered.txt"**, **","**.join**(**lines**[**index **-** 1**]))** print**("Deleted!")**

Pic.35. The function to delete an item from the list

That function reads the whole value of the txt-file and with the help of for-loop prints it.

The method ‘input’ is used to choose the task depending on its index.

After that the chosen line will we removed from ‘DetailOrdered.txt’.

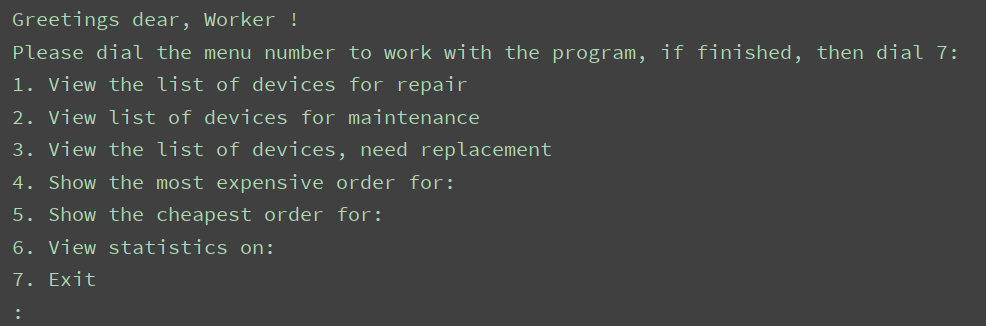
1. Exit

**if(**command **==** 7**):** main.main**()  
 break**

Pic.36. The part of code used to exit

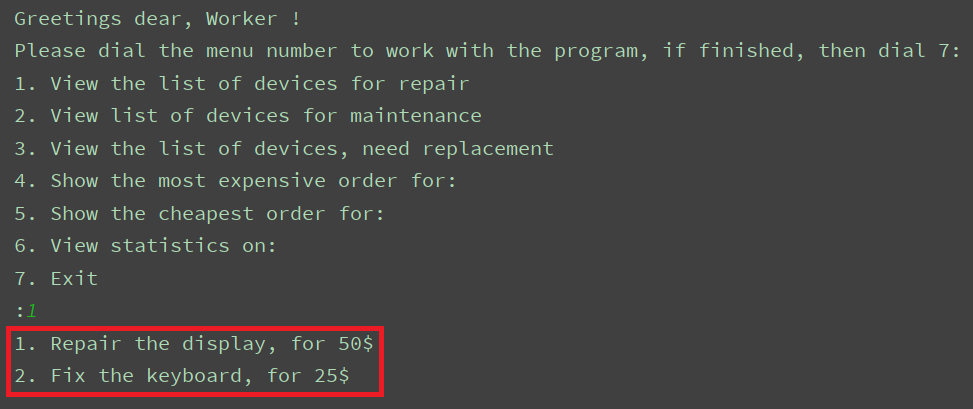
# **Account of a worker**

We have logged into the worker’s account. Below we may notice a dramatic growth of sections. The menu bar suggests choosing one section from the 7 given(Sc.8.).



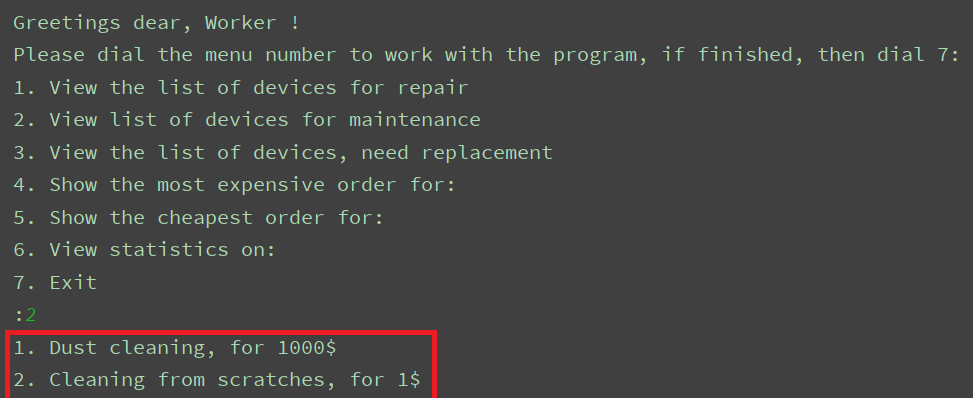
Pic.37. Worker’s menu

1. View the list of repair technicians - Show the entire list of devices handed over for repair with the name of a task and price that was paid.



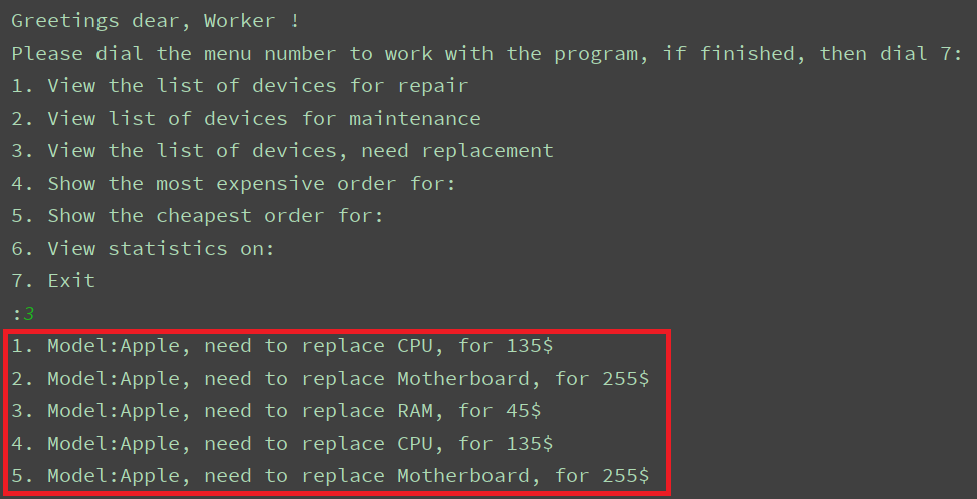
Pic.38. Repair list

1. View the list of devices for maintenance - Show the entirelist of devices handed over for maintenance(service) with the name of am aid and price that was paid.



Pic.39. Devices for maintence

1. View the list of devices, need replacement - Show the entire list of devices, that were left due to replacement of spare parts, with the device’s model, name of a spare need to be replaced and the price that was paid.



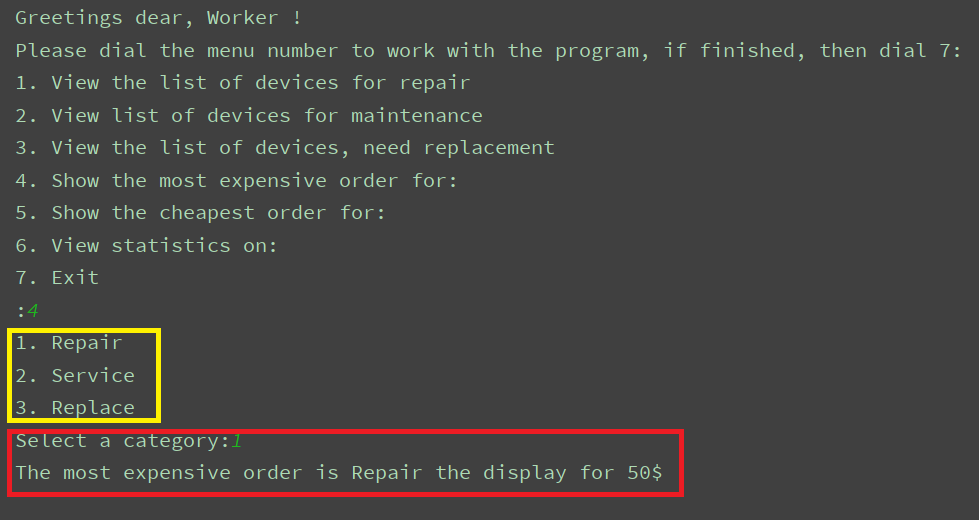
Pic.40. The third command

1. Show the most expensive order for - Show the expensive order for:

a. Repair b. Service c. Replace

The fourth command shows the most expensive order for one category with the

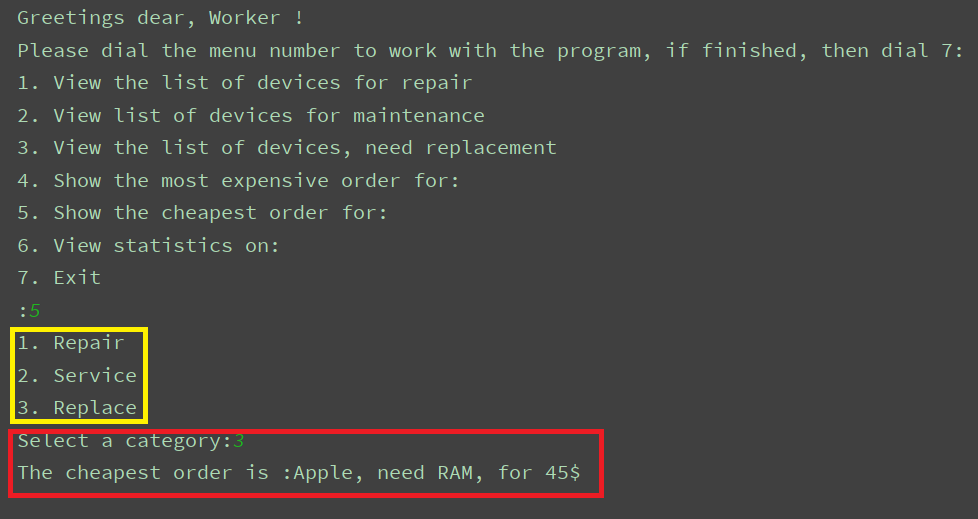
name of that task and the price that was paid.



Pic.41. The fourth command

1. Show the cheapest order for - Show the cheapest order for:

a. Repair b. Service c. Replace

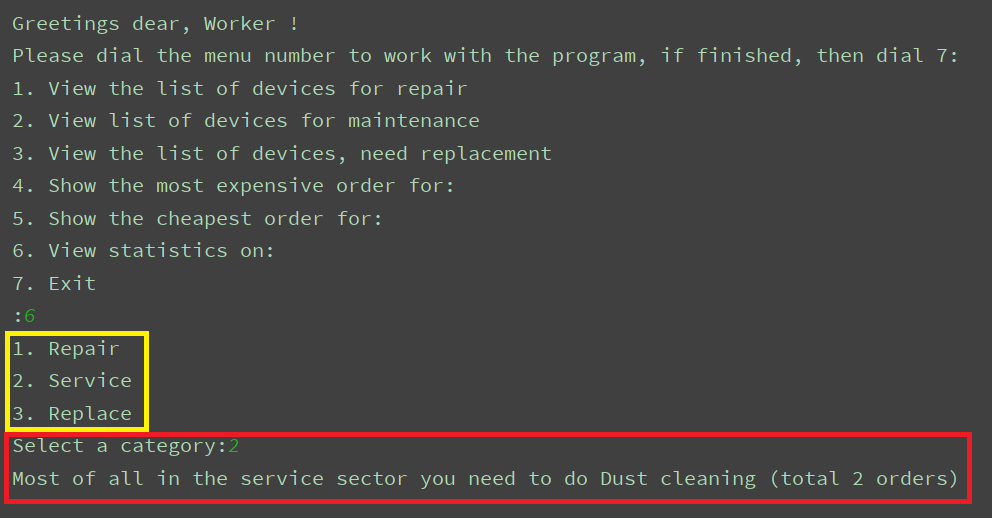


Pic.42. The fifth command

1. View statistics on – Show statistics of one category the the user will choose:

a. Repair b. Service c. Replace

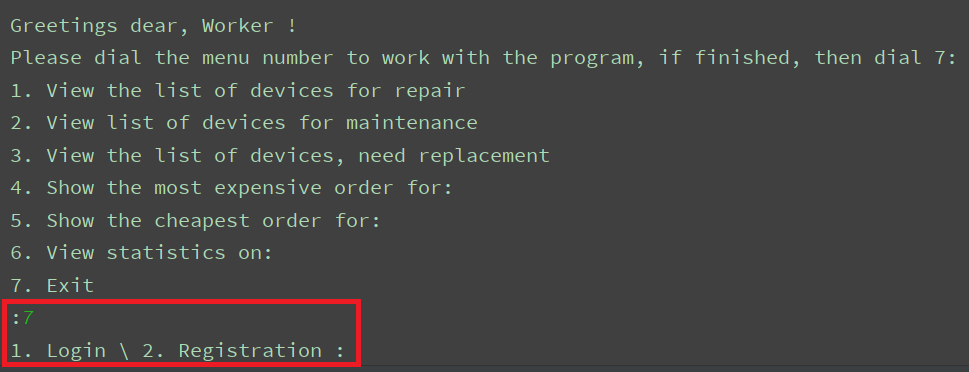
That command shows the most frequent order in chosen category.



Pic.43. The sixth command

1. Exit – Command to end the process

After that command the user will return to the menu of Login and Registration.



Pic.44. Exit

Code review (Account of a Worker)

**def menu\_start(**login\_par**):** commands **= [**show\_repair, show\_service, show\_change, show\_most\_expensive, show\_cheapest, show\_stats**]  
 global** login  
 login **=** login\_par  
 **while True:** print**()** print**("Greetings dear, Worker !")** print**("Please dial the menu number to work with the program, if finished, then dial 7:")** print**("1. View the list of devices for repair")** print**("2. View list of devices for maintenance")** print**("3. View the list of devices, need replacement")** print**("4. Show the most expensive order for:")** print**("5. Show the cheapest order for:")** print**("6. View statistics on:")** print**("7. Exit")** command **=** int**(**input**(":"))  
  
 if(**command**>**7 **or** command**<**1**):** print**("Error, there is no such command here, please try again :-(")  
 continue  
 if(**command **==** 7**):** main.main**()  
 break** commands**[**command**-**1**]()**

Pic.45. The worker’s menu

Here’s the main menu of an account. The user enters any number from 1 to 7. The list, named ‘commands’, contains all functions, that will participate in the process.

command **=** int**(**input**(":"))  
  
if(**command**>**7 **or** command**<**1**):** print**("Error, there is no such command here, please try again :-(")  
 continue  
if(**command **==** 7**):** main.main**()  
 break**commands**[**command**-**1**]()**

Pic.46. Part of code to stop process

After the user enters the number, this peace of code will begin the process.

commands**[**command**-**1**]()**

Pic.47. Indexing

Everything depends on the indexes of the list, ’commands’. Since indexing starts with ‘0’, we subtract ‘1’ from the value of ‘command.

Here we can see all functions, work behind the scenes.

1) def show\_repair()

**def show\_repair():** lines **=** context.get\_lines**("repair-needed.txt")  
  
 for** i,line **in** enumerate**(**lines**):** price **=** line.strip**("\n")**.split**(",")[**1**]** row **=** line.strip**("\n")**.split**(",")** print**(f"{**i**+**1**}. {**row**[**0**]}, for {**price**}$")**

Pic.48. The function to show the list of devices

That peace of code is responsible for the first command in the main menu. (View the list of devices for repair)

The function reads and outputs the value of the file, which contains data related to devices that were handed over for repair. It prints tasks and their prices. Here you can see the content of ‘repair-needed.txt’.

2) def show\_service()

**def show\_service():** lines **=** context.get\_lines**("service-need.txt")  
  
 for** i,line **in** enumerate**(**lines**):** row **=** line.strip**("\n")**.split**(",")** print**(f"{**i**+**1**}. {**row**[**0**]}, for {**row**[**1**]}$")**

Pic.49. The function to show the list of services

This part of code is in charge of the second command in the main menu (View the list of devices for maintenance)

That function just reads, splits and outputs the content of txt-file called “service-needed.txt” .

3) def show\_change()

**def show\_change():** lines **=** context.get\_lines**("change\_need.txt")  
  
 for** i,line **in** enumerate**(**lines**):** row **=** line.strip**("\n")**.split**(",")** print**(f"{**i**+**1**}. Model:{**row**[**0**]}, need to replace {**row**[**1**]}, for {**row**[**2**]}$")**

Pic.50. The function to show devices, need change

This function is responsible for the third command in the main menu. As other functions, def show\_change(), keeps the content of txt.file,”change\_need.txt”, in the variable called ‘lines’.

The function reads, splits and outputs the value of the variable. Here you can the content of our txt.file.

4)def show\_most\_expensive()

**def show\_most\_expensive():** print**("1. Repair")** print**("2. Service")** print**("3. Replace")** index **=** int**(**input**("Select a category:"))  
  
 if(**index**==**1**):** line **=** max**(**context.get\_lines**("repair-needed.txt")**, key**=lambda** x**:** cost**(**x, 1**))**.split**(",")** print**(f"The most expensive order is {**line**[**0**]} for {**line**[**1**]}$")  
 if(**index**==**2**):** line **=** max**(**context.get\_lines**("service-need.txt")**, key**=lambda** x**:** cost**(**x, 1**))**.split**(",")** print**(f"The most expensive order is {**line**[**0**]} for {**line**[**1**]}$")  
 if(**index**==**3**):** line **=** max**(**context.get\_lines**("change\_need.txt")**, key**=lambda** x**:** cost**(**x, 2**))**.split**(",")** print**(f"The most expensive order is :{**line**[**0**]}, need {**line**[**1**]}, for {**line**[**2**]}$")**

Pic.51. Most expensive order

Here the user chooses one of the three categories:

1) Repair, 2) Service, 3) Replace.

After this action the function will read, split the content, and then output the most expensive task contained in txt-file. Choose of txt-file depends on category, the user will choose.

5) def show\_cheapest()

**def show\_cheapest():** print**("1. Repair")** print**("2. Service")** print**("3. Replace")** index **=** int**(**input**("Select a category:"))  
  
 if(**index**==**1**):** line **=** min**(**context.get\_lines**("repair-needed.txt")**, key**=lambda** x**:** cost**(**x, 1**))**.split**(",")** print**(f"The cheapest order is {**line**[**0**]} for {**line**[**1**]}$")  
 if(**index**==**2**):** line **=** min**(**context.get\_lines**("service-need.txt")**, key**=lambda** x**:** cost**(**x, 1**))**.split**(",")** print**(f"The cheapest order is {**line**[**0**]} for {**line**[**1**]}$")  
 if(**index**==**3**):** line **=** min**(**context.get\_lines**("change\_need.txt")**, key**=lambda** x**:** cost**(**x, 2**))**.split**(",")** print**(f"The cheapest order is :{**line**[**0**]}, need {**line**[**1**]}, for {**line**[**2**]}$")**

Pic.52. Most cheapeast order

This function is responsible for the fifth command (Show the cheapest order for:)

Here the user chooses one of the three categories:

1) Repair, 2) Service, 3) Replace.

After this action the function will read, split the content, and then output the cheapest task contained in txt-file. Choose of txt-file depends on category, the user will choose.

The result of it will be: the name of the task and it’s price

6) def show\_stats()

**def show\_stats():** print**("1. Repair")** print**("2. Service")** print**("3. Replace")** index **=** int**(**input**("Select a category:"))  
  
 if(**index**==**1**):** a **= []** a **+=** context.select\_only**("repair-needed.txt"**, 0**)** a **+=** context.select\_only**("repaired.txt"**, 0**)** count **=** Counter**(**a**)**.most\_common**()[**0**]** print**("Most of all in the repair sector you need to do"**, count**[**0**]**, **f"(total {**count**[**1**]} orders)")  
  
 if(**index**==**2**):** a **= []** a **+=** context.select\_only**("service-need.txt"**, 0**)** a **+=** context.select\_only**("serviced.txt"**, 0**)** count **=** Counter**(**a**)**.most\_common**()[**0**]** print**("Most of all in the service sector you need to do"**, count**[**0**]**, **f"(total {**count**[**1**]} orders)")  
  
 if(**index**==**3**):** a **= []** a **+=** context.select\_only**("change\_need.txt"**,0**)** a **+=** context.select\_only**("changed.txt"**,0**)** count **=** Counter**(**a**)**.most\_common**()[**0**]** print**("Most part replacements required model"**, count**[**0**]**, **f"(total**

**{**count**[**1**]} orders)")**

Pic.53. Statistic

The user should choose one of the three categories. After that, our function reads and finds the most frequent task given in a specific txt-file.

**def select\_only(**file\_name, index, sep**=","):  
 return[** select.split**(**sep**)[**index**]  
 for** select **in** get\_lines**(**file\_name**)**

Pic.54. Select only

Also, we use def select-only () for selection inside of a specific txt-file.

7)

**if(**command **==** 7**):** main.main**()  
 break**

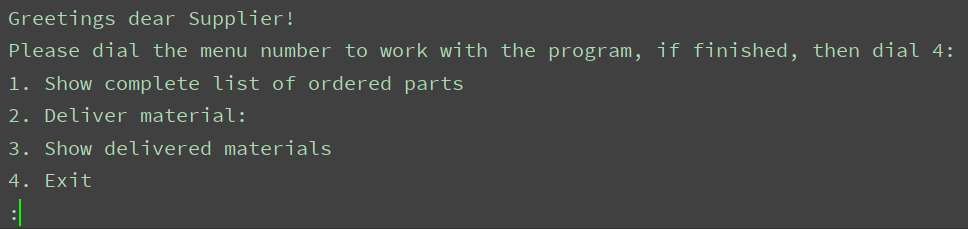
Pic.55. Break programm

If command is “7” process will end and the user will be thrown to the Login/Registration-menu.

# **Account of a supplier**

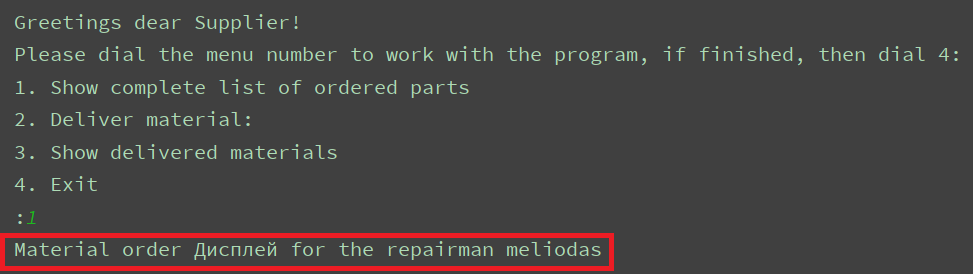
Here we have logged into a supplier’s account. The supplier can choose one of the four given commands in the main menu. He should enter a number from 1 to 4.

If the entered number is not equal to 1-4, he will be stuck in a loop, until he enters the right number.



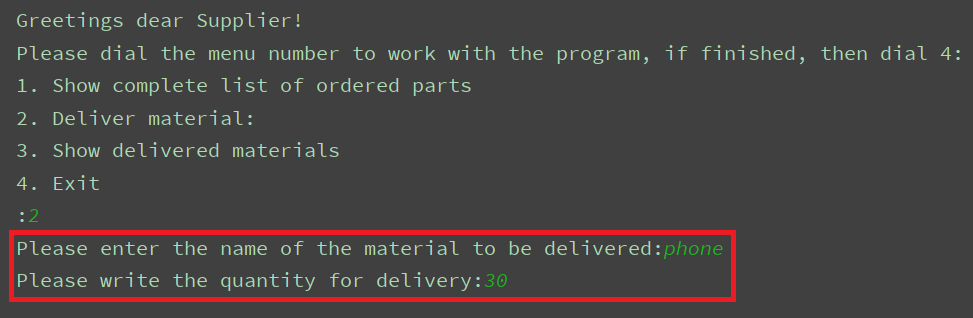
Pic.55. Supplier’s menu

1. Show complete list of ordered parts – Show the entire list of completed orders with the name of an item and the name of the customer or worker.



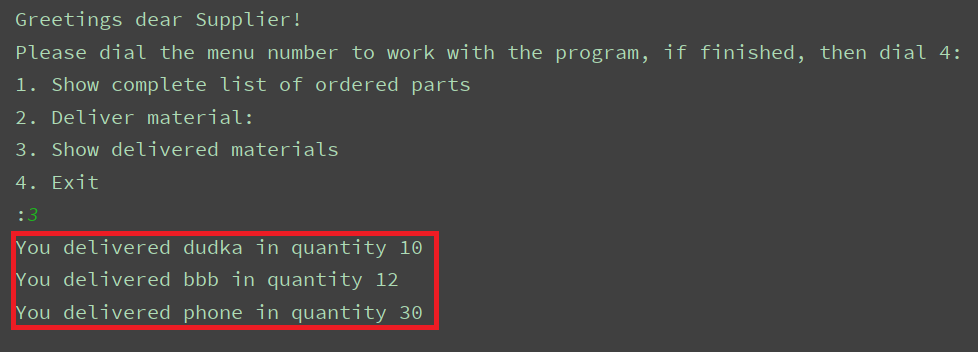
Pic.55. Show orders

1. Deliver material – The supplier adds an item to deliver and it’s quantity.



Pic.56. Deliver material

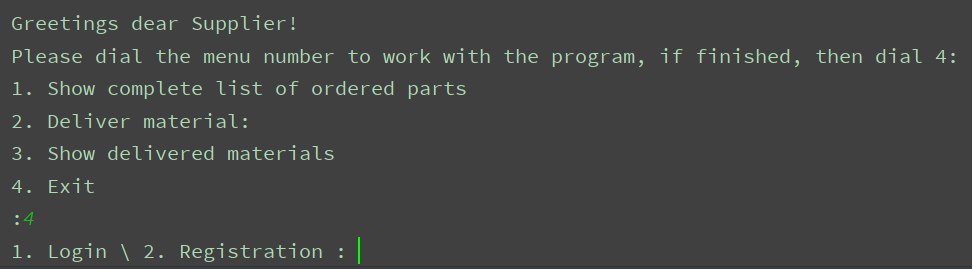
1. Show delivered materials – Show the entire list of delivered things and their quantity.



Pic.57. Delivered materials

1. Exit – Command to end the process

After that command the user will return to the menu of Login and Registration.



Pic.58. Exit from programm

Code review (Account of a Supplier)

**def menu\_start(**login\_par**):** commands **= [**show\_details, deliver\_detail, show\_delivered**]  
 global** login  
 login **=** login\_par  
 **while True:** print**()** print**("Greetings dear Supplier!")** print**("Please dial the menu number to work with the program, if finished, then dial 4:")** print**("1. Show complete list of ordered parts")** print**("2. Deliver material:")** print**("3. Show delivered materials")** print**("4. Exit")** command **=** int**(**input**(":"))  
  
 if(**command**>**4 **or** command**<**1**):** print**("Error, there is no such command here, please try again :-(")  
 continue  
 if(**command **==** 4**):** main.main**()  
 break** commands**[**command**-**1**]()**

Pic.59. Begin

Here we can see the main menu of Supplier’s account.

commands**[**command**-**1**]()**

Pic.60. Commands

The list ‘commands’ contains all functions that belong to Supplier’s account. The user should enter a number from 1 to 4. A variable, called ‘command’ is responsible for the starting of process. If all conditions are == True, that piece of code will start the program.

Next we can see whole code (used functions) and used files in the process.

1) def show\_details()

**def show\_details():  
 for** line **in** context.get\_lines**("DetailOrdered.txt"):** row **=** line.split**(",")** print**("Material order"**,row**[**0**]**,**"for the repairman"**, row**[**1**])**

Pic.61. Show delivered materials

This function is responsible for the first command in the main menu. It shows the entire list of completed orders with the name of an item and the name of the customer or worker.

Firstly, the function gets the value of a txt-file called ‘DetailOrdered.txt’.

2) def deliver\_material()

**def deliver\_detail():** a **=** input**("Please enter the name of the material to be delivered:")** b **=** input**("Please write the quantity for delivery:")** context.write**("delivered\_materials.txt"**, **[**a, b, login**])**

Pic.62. Materials for deliver

This function is responsible for the second command in the main menu. With it’s help the supplier adds an item to deliver and item’s quantity.

We have two variables here: ‘a’ and ‘b’. ‘A’ is for writing of the name of needed spare and ‘B’ is for writing the quantity of it.

3) def show\_delivered()

**def show\_delivered():  
 for** row **in** context.get\_items\_by\_login**("delivered\_materials.txt"**, login**):** print**("You delivered"**,row**[**0**]**,**"in quantity"**, row**[**1**])**

Pic.63. Delivered materials

This function is responsible for the third command, which is showing the entire list of delivered things and their quantity.

**def get\_items\_by\_login(**file\_name, login, sep**=","**, index**=-**1**):** result **= []  
 with** open**(**file\_name, encoding**='utf-8') as** file**:  
 for** line **in** file.readlines**():  
 if(**line.split**(**sep**)[**index**]**.replace**("\n"**,**"")==**login**):** result.append**(**line.split**(**sep**))  
 return** result

Pic.64. def get\_items\_by\_login()

With the help of cycle ‘for’ and function ‘get\_items\_by\_login’, which is in the file “context.py“ we get the value from txt-file, delivered\_materials.txt, and print(Sc.37.).

4) exit

**if(**command **==** 4**):** main.main**()  
 break**

Pic.65. Exit programm

If command == 4 process will end and the user will be thrown to the Login/Registration-menu.

# **Conclusion**

By doing our course work we achieved our goals.

Our goals were:

→ To create a program to facilitate process of work of people, whose job relates to repairing of gadgets.

→ To automate the work of program

Summing up the results of our work done, I want to tell you what we have achieved during this time. Our goal - to automate the phone repair service system has been fulfilled. While working on the task, we learned a lot, we fulfilled all the goals using the knowledge gained in the lessons, we also searched for additional information on the Internet, asked for help from teachers. While working on the project, our group encountered the difficulty of generating unique identifiers. To do this, we studied a library named UUID and used its .uuid4 function. Also, after we finished writing the code, one of the Vendor's functions was working incorrectly, it took us a lot of time to understand the reason and fix it. These difficulties have shown the importance of teamwork and we succeeded in our task. Our program works and can complete all of its functions that were given in the technical requirement. Also, we made some additional functions to improve the functionality of program. Despite the fact, there was not so much time, we made a demo version of a program, that can really be used in future if it will be made more dynamic. Thanks to the project, we have expanded our knowledge.

We learned how to:

1. work with different files in Python;

2. work with different types of functions in Python

3. work with hash-codes to make a better Log in/Log out etc.

4. work with UUID library

5. work with team

# **References:**

1. INAI.kg Lectures
2. StackOverFlow - <https://ru.stackoverflow.com/>
3. GitHub - <https://github.com/search?q=python+files&type=issues>
4. CodeRoad - <https://coderoad.ru/search/?q=%D0%92%D0%BA%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0+%D0%BE%D1%87%D0%B5%D1%80%D1%87%D0%B5%D0%BD%D0%B0+python+3+.txt+%D1%87%D1%82%D0%B5%D0%BD%D0%B8%D0%B5+%D1%84%D0%B0%D0%B9%D0%BB%D0%B0>