ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

Υλοποίηση εφαρμογής σε γλώσσα Python, με χρήση αντικειμενοστραφών τεχνικών για τη διαχείριση της πελατειακής βάσης ενός καταστήματος τραπέζης

Συγγραφέας
Φωτόπουλος Παναγιώτης
Αριθμός Μητρώου: 71446003

Επιβλέπων
Δρ. Χρήστος Δρόσος
Επίκουρος Καθηγητής

Αθήνα, Μάρτιος, 2024

UNIVERSITY OF WEST ATTICA
SCHOOL OF ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING

Implementation of an application in Python, using object-oriented techniques for the management of the customer base of a bank branch

Author
Fotopoulos Panagiotis
Registration Number: 71446003

Supervisor
Dr. Christos Drosos
Assistant Professor

Athens, March, 2024

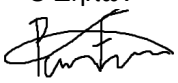
Δήλωση Συγγραφέα Διπλωματικής Εργασίας

Ο κάτωθι υπογεγραμμένος Φωτόπουλος Παναγιώτης ,με αριθμό μητρώου 71446003, φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Βιομηχανικής Σχεδίασης και Παραγωγής, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών



Μέλη Εξεταστικής Επιτροπής

Η διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

α/α	Όνομα / Επώνυμο	Ψηφιακή Υπογραφή
1	ΔΡΟΣΟΣ ΧΡΗΣΤΟΣ Επίκουρος Καθηγητής	Digitally signed by CHRISTOS DROSOS Date: 2024.07.16 19:40:26 +03'00'
2	ΛΑΣΚΑΡΗΣ ΝΙΚΟΛΑΟΣ	Nikolaos Laskaris Digitally signed by Nikolaos Laskaris Date: 2024.07.19 13:10:21 +03'00'
3	ΠΑΠΑΚΙΤΣΟ ΕΥΑΓΓΕΛΟΣ	Evangelos Papakitsos Digitally signed by Evangelos Papakitsos Date: 2024.07.17 10:23:59 +03'00'

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας κύριο Χρίστο Δρόσο για την επιλογή και παραχώρηση ενός ενδιαφέροντος αλλά και πολύ δημιουργικού θέματος, καθώς και για την υποστήριξή του κατά τη διάρκεια της συγγραφής της.

Περίληψη

Στο πλαίσιο αυτής της διπλωματική εργασία εξετάζονται οι θεμελιώδεις έννοιες του προγραμματισμού όπως η αντικειμενοστράφια, η κληρονομικότητα και ο πολυμορφισμός. Το έργο παρέχει μια λεπτομερή ανάλυση της δομής ενός προγράμματος, εξετάζοντας τη σχέση μεταξύ της κύριας γονικής κλάσης και των τριών υποκλάσεών της. Επίσης, διερευνάται ο τρόπος με τον οποίο χρησιμοποιούνται οι τύποι δεδομένων στον κώδικα. Επιπλέον, σε αυτήν την διπλωματική εργασία εξετάζονται διάφορες βιβλιοθήκες και frameworks,και πως αυτά τα εξειδικεύουν την εκάστοτε γλώσσα προγραμματισμού. Απώτερος σκοπός είναι να προσφέρει μια σαφή κατανόηση των παραπάνω καθώς και την σωστή αλλά και πρακτική εφαρμογή τους.

Λέξεις κλειδιά:

#Αντικειμενοστραφής προγραμματισμός

#Κληρονομικότητα

#Ανάπτυξη Python

#Τραπεζικό Λογισμικό

Abstract

In the context of this thesis, the fundamental concepts of programming such as object-oriented programming, inheritance, and polymorphism are examined. The project provides a detailed analysis of a program's structure, examining the relationship between the main parent class and its three subclasses. Additionally, the way data types are used in the code is investigated. Furthermore, the thesis examines various libraries and frameworks and how these specialize in each programming language. The goal is to offer a clear understanding of the above concepts as well as their correct and practical application.

Keywords

#Object-oriented programming

#Inheritance

#Python Development

#Banking Software

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

1 Εισαγωγή

Στο πλαίσιο αυτής της εργασίας, εξετάζεται η ανάπτυξη ενός συστήματος τραπεζικής λογικής με χρήση της γλώσσας προγραμματισμού Python. Το σύστημα αυτό προορίζεται να παρέχει βασικές λειτουργίες τραπεζικών λογαριασμών, όπως κατάθεση, ανάληψη κεφαλαίου και διαχείριση λογαριασμών διαφορετικών τύπων.

Η εργασία διατυπώνεται σε δύο κύρια μέρη. Αρχικά, περιγράφεται η δομή του συστήματος, συμπεριλαμβανομένης την δημιουργία χρήστη ,την αποθήκευση των στοιχείων του, την κλάσης Account (Λογαριασμού) καθώς και τις υποκλάσεις του αντικειμένου, όπως οι Λογαριασμοί : Checking Accounts, Savings Accounts και DepositGuard Accounts . Εξηγείται η λογική πίσω από την ανάπτυξη αυτών των κλάσεων και πώς αλληλοεπιδρούν με τον χρήστη.

Στη συνέχεια, παρουσιάζεται η μεθοδολογία που ακολουθήθηκε για την υλοποίηση του συστήματος. Αυτό επιτεύχθηκε με τις κατάλληλες γνώσεις ανάπτυξης λογισμικού και εφαρμογές των αρχών του αντικειμενοστραφούς προγραμματισμού και της κληρονομικότητας.

2 Θεμελιώδη Κώδικα Python

2.1 Κλάση(Class)

Στην Python, η έννοια της "κλάσης" αναφέρεται σε ένα πρότυπο ή ένα "σχέδιο" αντικειμένου, το οποίο ορίζει τα χαρακτηριστικά και τη συμπεριφορά του [1]. Μια κλάση είναι ένας τύπος δεδομένων που ορίζει τις ιδιότητες και τις λειτουργίες που μπορεί να έχει το αντικείμενο.

Παράδειγμα: Μπορούμε να πάρουμε για παράδειγμα έναν ορισμό κλάσης "Αυτοκίνητο". Η κλάση αυτή θα ορίσει τις ιδιότητες (όπως μάρκα, μοντέλο, χρώμα) και τις λειτουργίες (όπως εκκίνηση, πέδηση, επιτάχυνση) ενός αυτοκινήτου.

2.1.1 Γονική Κλάση (Parent Class)

Η γονική κλάση σε ένα πρόγραμμα είναι η κλάση από την οποία κληρονομούνται τα χαρακτηριστικά και οι μέθοδοι από άλλες κλάσεις, γνωστές ως υποκλάσεις [1]. Η γονική κλάση παρέχει τη βάση για τις υποκλάσεις, και οι υποκλάσεις αντίστοιχα μπορούν να τις επεκτείνουν αναλόγως των αναγκών λειτουργείας τους.

2.1.2 Υποκλάση (Subclass)

Στην Python, οι υποκλάσεις είναι κλάσεις που κληρονομούν χαρακτηριστικά και μεθόδους από μια γονική κλάση. Μπορούμε να χρησιμοποιήσουμε τις υποκλάσεις για να ορίσουμε πιο συγκεκριμένους τύπους αντικειμένων που κληρονομούν τις ιδιότητες και τη συμπεριφορά των γονικών τους κλάσεων, επεκτείνοντας η τροποποιώντας τις ήδη υπάρχουσες λειτουργίες [1].

2.2 Κληρονομικότητα (Inheritance)

Η κληρονομικότητα είναι μια από τις θεμελιώδεις έννοιες στον προγραμματισμό. Ειδικότερα, στην Python αναφέρεται στη δυνατότητα μιας κλάσης να αποκτά τα χαρακτηριστικά και τη συμπεριφορά μιας άλλης κλάσης [2]. Αυτό σημαίνει ότι η νέα κλάση μπορεί να χρησιμοποιεί τα χαρακτηριστικά και τις μεθόδους μιας υπάρχουσας κλάσης χωρίς να χρειάζεται να ξανά κατασκευαστούν.

Στην Python, η κληρονομικότητα υλοποιείται χρησιμοποιώντας τον μηχανισμό των κλάσεων. Μια υποκλάση μπορεί να κληρονομήσει τα χαρακτηριστικά και τις μεθόδους μιας γονικής κλάσης, επεκτείνοντας τη λειτουργικότητά της ή προσθέτοντας νέα χαρακτηριστικά και μεθόδους.

Η κληρονομικότητα επιτρέπει την αναδιάταξη και την επαναχρησιμοποίηση κώδικα, καθώς μπορούμε να δημιουργούμε νέες κλάσεις βασισμένες στις υπάρχουσες και να προσθέτουμε λειτουργικότητα χωρίς να ξανά γράφουμε τον κώδικα που έχει ήδη γραφτεί σε προηγούμενες κλάσεις.

2.3 Πολυμορφισμός (Polymorphism)

Ο πολυμορφισμός στην Python αναφέρεται στην ικανότητα των αντικειμένων να παρουσιάζουν διαφορετική συμπεριφορά ανάλογα με την κατάσταση ή την κλάση από την οποία κλήθηκαν. Με άλλα λόγια, διαφορετικά αντικείμενα μπορούν να αντιδρούν διαφορετικά σε κοινές εντολές [3].

Ένα κλασικό παράδειγμα είναι η κλάση "Shape" που έχει μια μέθοδο "area", αλλά υποκλάσεις της όπως "Circle" και "Rectangle" υλοποιούν τη μέθοδο αυτή με διαφορετικό τρόπο, ανάλογα με τον τύπο του σχήματος. Όταν καλείται η μέθοδος "area" για ένα αντικείμενο "Circle", υπολογίζεται το εμβαδό του κύκλου, ενώ όταν καλείται για ένα αντικείμενο "Rectangle", υπολογίζεται το εμβαδόν του ορθογωνίου.

Ο πολυμορφισμός μας επιτρέπει να γράψουμε πιο ευέλικτο και αποτελεσματικό κώδικα, καθώς μπορούμε να χειριστούμε διαφορετικούς τύπους αντικειμένων με τον ίδιο τρόπο, χωρίς να χρειάζεται να γνωρίζουμε εκ των προτέρων τον ακριβή τύπο κάθε αντικειμένου.

2.4 Αντικειμενοστραφής-προγραμματισμός (Object-Oriented Programming OOP)

Αντικειμενοστραφής Προγραμματισμός αναφέρεται σε μια μεθοδολογία ανάπτυξης λογισμικού όπου ο κώδικας οργανώνεται γύρω από αντικείμενα και τις αλληλεπιδράσεις μεταξύ τους. Τα αντικείμενα αυτά αναπαριστούν συνήθως συγκεκριμένες οντότητες με συγκεκριμένες ιδιότητες και λειτουργίες [4]. Η Python, όπως και πολλές άλλες γλώσσες προγραμματισμού, υποστηρίζει πλήρως την OOP.

Υπενθυμίζοντας κάποιες από τις βασικές αρχές του Αντικειμενοστραφή Προγραμματισμού στην Python:

1. Κλάσεις και Αντικείμενα: Στην Python, η κλάση είναι μια αποθήκη δεδομένων και λειτουργιών που περιγράφει έναν τύπο αντικειμένου. Ένα αντικείμενο είναι μια συγκεκριμένη εμφάνιση μιας κλάσης, με συγκεκριμένες τιμές για συγκεκριμένα δεδομένα.
2. Κληρονομικότητα: Η Python υποστηρίζει την κληρονομικότητα, η οποία επιτρέπει σε μια κλάση να κληρονομεί τις ιδιότητες και τις μεθόδους μιας άλλης κλάσης. Αυτό βοηθά στην αποφυγή της επανάληψης κώδικα και στην οργάνωση του.

Ένα παράδειγμα οργάνωσης κώδικα βασισμένο σε Αντικειμενοστραφή Προγραμματισμό θα μπορούσε να είναι μια κλάση Car, η οποία έχει ιδιότητες την μάρκα, το μοντέλο και την χρονολογία, καθώς και μεθόδους για την εκκίνηση του αυτοκινήτου, την επιτάχυνση και το φρενάρισμα. Μπορεί επίσης να υπάρχουν υποκλάσεις όπως ElectricCar ή GasolineCar, που κληρονομούν τις ιδιότητες και τις μεθόδους της βασικής κλάσης Car και προσθέτουν επιπλέον λειτουργικότητα που είναι μοναδική για αυτούς τους τύπους αυτοκινήτων.

3 Βασικά Χαρακτηριστικά Λογισμικού

3.1 Συναρτήσεις(Functions)

Στην Python οι συναρτήσεις είναι ένα μπλοκ κώδικα που εκτελείται με σκοπό την επίτευξη μιας συγκεκριμένης εργασίας .Μπορούν να χρησιμοποιηθούν επανειλημμένα κατά την διάρκεια λειτουργίας του προγράμματος και να κληρονομηθούν από τις υποκλάσεις εάν και μόνο αν οι παραπάνω βρίσκονται μέσα στην γονική κλάση, εδώ αναφέρονται μερικά πλεονεκτήματα χρήσης συναρτήσεων [1].

- Επαναχρησιμοποίηση κώδικα: Η δημιουργία συναρτήσεις προσφέρει την δυνατότητα επαναχρησιμοποίησης του ίδιου κώδικα πολλές φορές σε διαφορετικά μέρη του προγράμματος.
- Οργάνωση και διάσπαση κώδικα: Οι συναρτήσεις βοηθούν στην διάσπαση μεγάλων προγραμμάτων σε μικρότερα κομμάτια κώδικα γεγονός που κάνει τον κώδικα ποιο δομοστοιχειωτό, επιτυγχάνοντας καλύτερο επίπεδο τόσο στην ανάγνωση όσο και στην συντήρηση του

Οι συναρτήσεις στην Python γράφονται με την μορφή def <*function_name*> (self, . . .).

- Το def είναι η λέξη-κλειδί που χρησιμοποιείται για να οριστεί μια συνάρτηση στην Python [1].
- Το *function_name* είναι το όνομα της συνάρτησης και πρέπει να είναι μια συνεχομένη λέξη χωρίς κενά αλλιώς να συνδέετε με κάτω παύλα, συνηθίζεται πολύ το Capitalize (def TestFuction) τέλος είναι σημαντικό το όνομα με το οποίο βαφτίζεται μια συναρτήσει να περιγράφει και την λειτουργεία της [1].

    - Το def __init__(self, . . .) είναι ένας ειδικός τύπος μεθόδου ο οποίος ονομάζεται constructor (κατασκευαστής). Η παραπάνω μέθοδος καλείτε κάθε φορά που δημιουργείται ένα νέο αντικείμενο μιας κλάσης αυτοματοποιημένα και χρησιμοποιείται για την αρχικοποίηση των χαρακτηριστικών του αντικειμένου [1].

- Αρχικοποίηση των χαρακτηριστικών του αντικειμένου: Η κυριά λειτουργεία του __init__ είναι να θέτει αρχικές τιμές στα χαρακτηριστικά του αντικειμένου

```
class Account():
    def __init__(self):
        self.withdrawals_times = 0
        self.days_count = 0
        self.ck_account = CheckingAccount
        self.name = 1
```

Εικόνα 1: Αρχικοποίηση μέσω init στην Parent Class.

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")
```

Εικόνα 2: Αρχικοποίηση Μεταβλητής Τύπου String.

- Κλήση κατά την δημιουργία αντικειμένων: Όταν δημιουργείται ένα νέο αντικείμενο στην κλάση, η Python καλεί αυτόματα την μέθοδο __init__.

- super().__init__(): η έκφραση super().__init__() χρησιμοποιείται σε υποκλάσεις (subclasses) που κληρωνόμουν από τις γονικές κλάσεις(Parent Classes) και λειτουργικά είναι ο τρόπος με τον οποίον καλείς την __init__ της γονικής κλάσης.
  Είναι ένα από τα βασικότερα χαρακτηρίστηκα που προσφέρει η κληρονομικότητα καθώς διασφαλίζεις την αποτελεσματική και σωστή χρήση των χαρακτηριστικών και των μεθόδων της γονική κλάσης μέσα στις υποκλάσεις, διατηρώντας την αλυσίδα αρχικοποίησης και την επαναχρησιμοποίηση του ήδη υπάρχων κώδικα.

```
class CheckingAccount(Account):
    def __init__(self, customer_id, account_type, action, default_value):
        super().__init__()
        self.customer_id = customer_id
        self.account_type = account_type
```

Εικόνα 3: Το Subclass CheckingAccount μέσω της super().__init__().Κληρονομεί τα Data του Parent Class.

- super().withdraw()/super().returning() : η συνάρτηση super().withdraw ()/returning() είναι ένα παράδειγμα κληρονομικότητας και σημαίνει ότι καλείται η μέθοδο

withdraw()/returning() της γονικής κλάσης σε μια υπόκλιση κάνοντας φανερά αντιληπτό την αποτροπή επανάληψης ήδη υπάρχον κώδικα στο πρόγραμμα.

```
def withdraw(self):
    super().withdraw()
    super().update_total_balance()
    self.action = ''
    super().returning()
```

Εικόνα 4: Καλείτε η super().withdraw().

Χωρίς όμως να σημαίνει αυτό ότι εξαλείφει την επιλογή της ανάπτυξης του ήδη υπάρχον κώδικα μόνο σε μια συγκεκριμένη υπόκλιση.

```
def withdraw(self):
    if datetime.now().day == 1 and (
            self.last_withdraw_date is None or self.last_withdraw_date.month != datetime.now().month):
        self.withdrawls_times = 0
    elif self.withdrawls_times < 3:
        print(f"Withdrawal of {self.amount} € successful.")
        self.last_withdraw_date = datetime.now()
        super().withdraw()
```

Εικόνα 5: Καλείτε η super withdraw πρηοσθέντας extra configurations.

Στις παραπάνω φωτογραφίες απεικονίζονται δυο παραδείγματα. Παρατηρείται ότι χρησιμοποιείται η ιδιά μέθοδο (withdraw) σε δυο διαφορετικές υποκλίσεις. Αξίζει να σημειωθεί εδώ ότι στην πρώτη φωτογραφία η μέθοδος καλύπτει ακριβός τις ανάγκες της υποκλάσεις αλλά στην δεύτερη για λογούς εκτέλεσης του έργου χρησιμοποιείται ένας μετρητής ο οποίος να περιορίζει την διαδικασία εάν έχουν πραγματοποιηθεί πάνω από 3 αναλήψεις μέσα σε έναν μηνά.

```
print('Withdrawal limit reached. Withdrawal functionality is locked.')
```

Εικόνα 6: Print message.

- Παρενθέσεις ( ): Οι παρενθέσεις είναι τα σύμβολα που περιβάλουν τις παραμέτρους της κάθε συνάρτησης.
- Παράμετροι : Οι παράμετροι είναι οι είσοδοι που δέχεται η συνάρτηση όταν καλείται. Ορίζονται στις παρενθέσεις και χρησιμοποιούνται για να μεταφορά δεδομένων μέσα στην συνάρτηση
    - Self: Στην Python, όταν ορίζεις μια μέθοδο μέσα σε μια κλάση η πρώτη παράμετρος της μεθόδου είναι πάντα το “self”. Το self είναι ένας συμβατικός τρόπος αναφοράς στο ίδιο το αντικείμενο της κλάσης όταν βρίσκεσαι μέσα σε μια μέθοδο

```python
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")
```

Εικόνα 7: Δηλώσει παραμέτρων.

Στο παραπάνω παράδειγμα μέσω της εντολής self.brand = brand και self.model = model ουσιαστικά αρχικοποιούνται οι ιδιότητες του αντικειμένου car.

3.2 Δομές Ελέγχου(control structures)

Οι δομές ελέγχου στον προγραμματισμό αναφέρονται στις δομές που επιτρέπουν τον έλεγχο της ροής εκτέλεσης του προγράμματος. Αυτές οι δομές επιτρέπουν τη λήψη αποφάσεων βάσει κάποιων συνθηκών και την επανάληψη ενός τμήματος κώδικα όσο κάποια συνθήκη είναι αληθής. Οι βασικές δομές ελέγχου περιλαμβάνουν:

1. Δομή if-else: Χρησιμοποιείται για τη λήψη αποφάσεων βάσει της αληθής ή ψευδής τιμής μιας συνθήκης. Αν η συνθήκη είναι αληθής, εκτελείται μια συγκεκριμένη πράξη, διαφορετικά εκτελούνται άλλες εντολές.
2. Δομή while: Χρησιμοποιείται για την επανάληψη ενός τμήματος κώδικα όσο μια συνθήκη είναι αληθής(ή ψευδής αναλόγως τις ανάγκες του κώδικα). Η συνθήκη αυτή ελέγχεται πριν από κάθε εκτέλεση του τμήματος κώδικα.
3. Δομή for: Χρησιμοποιείται για την επανάληψη ενός τμήματος κώδικα για έναν ορισμένο αριθμό φορών ή για κάθε στοιχείο σε μια ακολουθία.

Αυτές οι δομές είναι κρίσιμες για τον έλεγχο της ροής ενός προγράμματος και τη δημιουργία πιο σύνθετων λειτουργιών.

3.3 Βιβλιοθήκες Framework και Attributes

3.3.1 Βιβλιοθηκες (Libraries)

Οι βιβλιοθήκες (libraries) στον προγραμματισμό είναι πολύτιμα και δίνουν πάρα πολλές εναλλακτικές λύσεις στον προγραμματιστή

Οι βιβλιοθήκες ποιο αναλυτικά:

1. Ορισμός:

- Οι βιβλιοθήκες είναι σύνολα προγραμματιστικού κώδικα που περιλαμβάνουν συναρτήσεις, κλάσεις, και/ή μεθόδους που επιλύουν συγκεκριμένα προβλήματα ή παρέχουν λειτουργικότητα που μπορεί να είναι χρήσιμη σε πολλαπλά έργα.

2. Χρήση:
   - Οι προγραμματιστές μπορούν να εισάγουν μια βιβλιοθήκη στον κώδικά τους χρησιμοποιώντας τη δήλωση import.
   - Έπειτα από την εισαγωγή, μπορούν να καλέσουν τις συναρτήσεις και να χρησιμοποιήσουν τις κλάσεις που παρέχονται από τη βιβλιοθήκη στον κώδικά τους.
3. Παραδείγματα:
   - Numpy: Μια βιβλιοθήκη στην Python για τον χειρισμό πινάκων και την εκτέλεση αριθμητικών υπολογισμών.
   - Pandas: Παρέχει δομές δεδομένων και λειτουργίες για τον ανάλυση και τον καθαρισμό δεδομένων σε Python.
   - Matplotlib: Μια βιβλιοθήκη για τη δημιουργία διαγραμμάτων και γραφικών παραστάσεων για τα δεδομένα στην Python.

3.3.2 Frameworks

Τα frameworks στην Python είναι σύνολα βιβλιοθηκών και εργαλείων που παρέχουν έτοιμες δομές για την ανάπτυξη λογισμικού. Τα frameworks παρέχουν ένα πλαίσιο εργασίας για την επίλυση συγκεκριμένων προβλημάτων ή για την υλοποίηση συγκεκριμένων λειτουργιών, με σκοπό την απλούστευση της διαδικασίας ανάπτυξης λογισμικού και την επιτάχυνση του προγραμματισμού.
Στην Python, υπάρχουν πολλά δημοφιλή frameworks που καλύπτουν διάφορους τομείς, όπως:

- η διαδικτυακή ανάπτυξη (π.χ. Django, Flask),
- η επιστημονική υπολογιστική (π.χ. NumPy, SciPy),
- η μηχανική μάθηση (π.χ. TensorFlow, PyTorch),
- η ανάπτυξη λογισμικού (π.χ. Twisted, PyQT) και πολλά άλλα.

Κάθε framework προσφέρει ένα σύνολο εργαλείων, δομών δεδομένων και βελτιστοποιημένων αλγορίθμων που βοηθούν τους προγραμματιστές να επιτύχουν τους στόχους τους με μεγαλύτερη αποτελεσματικότητα και ταχύτητα.

3.3.3 Attribute

Στην Python, το "attribute" αναφέρεται σε ένα χαρακτηριστικό ενός αντικειμένου. Μπορεί να είναι μια μεταβλητή που είναι συνδεδεμένη με το αντικείμενο ή μια μέθοδος που μπορεί να κληθεί από αυτό. Τα attributes επιτρέπουν την πρόσβαση στις ιδιότητες και τη συμπεριφορά ενός αντικειμένου.

Για παράδειγμα , στην συγκεκριμένη εργασία γίνετε αναφορά στην δημιουργία λογαριασμού. Κάποια από τα attributes που χρησιμοποιήθηκαν είναι το name δηλαδή το όνομα του χρήστη, το tax_id μια προσομοίωση του φορολογικού του αριθμού και το customer_id ένας μοναδικός κωδικός για τον κάθε χρήστη.

Τα attributes μπορεί να είναι διαθέσιμα για ανάγνωση ή ενημέρωση ανάλογα με τον τρόπο με τον οποίο έχουν καθοριστεί.

3.4 Τύποι Δεδομένων(Data Type)

Οι τύποι δεδομένων σε μια γλώσσα προγραμματισμού ορίζουν τις κατηγορίες δεδομένων που μπορούν να αναπαραστήσουν τιμές και να εκτελέσουν συγκεκριμένες λειτουργίες πάνω σε αυτές. Στην Python, υπάρχουν διάφοροι τύποι δεδομένων που καλύπτουν μια ευρεία γκάμα πληροφοριών [2].

3.4.1 Ακέραιοι αριθμοί (integers, -int):

Αναπαριστούν ακέραιους αριθμούς, όπως τον αριθμό 1 ή τον αρνητικό αριθμό -5. Χρησιμοποιούνται για απλές μαθηματικές πράξεις και για αναπαράσταση δεδομένων που δεν έχουν δεκαδικά [3].

3.4.2 Αλφαριθμητικές ακολουθίες (strings, -str):

Χρησιμοποιούνται για την αποθήκευση κειμένου. Μπορούν να περιέχουν οποιοδήποτε σύμβολο, συμπεριλαμβανομένων γραμμάτων, αριθμών και ειδικών χαρακτήρων. Για παράδειγμα, `"Hello, World!"` είναι μια αλφαριθμητική ακολουθία [4].

3.4.3 Δεκαδικοί αριθμοί (floats):

Αναπαριστούν αριθμούς με δεκαδικά ψηφία. Χρησιμοποιούνται για ακριβείς υπολογισμούς που απαιτούν δεκαδική ακρίβεια, όπως οι χρηματιστηριακές συναλλαγές ή οι επιστημονικοί υπολογισμοί [5].

3.4.4 Λίστες (lists):

Αναπαριστούν μια συλλογή από δεδομένα, τα οποία μπορούν να είναι οποιουδήποτε τύπου. Χρησιμοποιούνται για την οργάνωση και τη διαχείριση συναφών δεδομένων [6].

3.4.5 Πλειάδες (Tuples):

Οι πλειάδες είναι παρόμοιες με τις λίστες αλλά είναι μη μεταβλητές, δηλαδή δεν μπορούν να τροποποιηθούν μετά τη δημιουργία τους [7].

3.4.6 Λεξικά (dictionaries):

Αναπαριστούν μια συλλογή από ζεύγη "κλειδιού-τιμής" (Pair Value:Key). Κάθε στοιχείο στο λεξικό αντιστοιχεί σε ένα μοναδικό κλειδί και μια τιμή που συσχετίζεται με αυτό το κλειδί. Τα λεξικά χρησιμοποιούνται για την αναπαράσταση δεδομένων σε μορφή πίνακα, όπου η πρόσβαση σε μια τιμή γίνεται με βάση το κλειδί και όχι τη θέση του στον πίνακα. Αυτό τα καθιστά ιδιαίτερα χρήσιμα για την αποθήκευση και την ανάκτηση δεδομένων με βάση κάποιο συγκεκριμένο αναγνωριστικό [8].

-Παράδειγμα: `{'name': 'John', 'age': 30, 'city': 'New York'}` είναι ένα λεξικό που περιέχει πληροφορίες για ένα άτομο.

4 Έλεγχος Ροής (Control Flow)

4.1 Ροή (Flow)

Η ροή στον προγραμματισμό πρεσβεύει την ακολουθία με την οποία οι εντολές εκτελούνται σε ένα πρόγραμμα. Η ροή αυτή επηρεάζεται από διάφορες δομές ελέγχου όπως βρόγχους συνθήκες, και συναρτήσεις [9]. Επίσης, μιλώντας για ροή και έχοντας ήδη κατανοήσει την λειτουργεία των συναρτήσεων είναι σημαντικό να αναφερθεί ότι μέσα στις συναρτήσεις κρύβονται δεδομένα και ουσιαστικά μπορεί κάποιος να αναγνωρίσει την πορεία τους μέσα στον κώδικα. Τι αντίδραση έχει δηλαδή στην λειτουργία το προγράμματος η κάθε δράση του χρήστη.

4.2 Συνθήκη If, elif, else

Η συνθήκη «if statement» όπως λέει και η λέξη είναι μια εντολή η οποία ελέγχει εάν μια συνθήκη είναι αληθής ή όχι (Boolean True or False) [9]. Είναι μια από τις πιο βασικές μεθόδους ελέγχου ορθότητας: αν η συνθήκη είναι αληθής, τότε εκτελείται το κομμάτι κώδικα που βρίσκεται κάτω από την συνθήκη "if". Συνήθως τα if statements ακολουθούνται από το elif ή else σε περίπτωση που η πρώτη συνθήκη είναι ψευδής και αυτό σημαίνει ότι είναι μια διαδικασία βιωματική οπού σε κάθε βήμα ελέγχεται η ορθότητα της συνθήκης μέχρις ότου να εκτελεστεί το κομμάτι κώδικα ή να βγει από αυτό.

Βασικές συνθήκες [10]:

- Σύγκριση δεδομένων:
  - == : Ισότητα
  - != : Ανισότητα
  - > : Μεγαλύτερο από
  - < : Μικρότερο από
  - >= : Μεγαλύτερο ή ίσο
  - <= : Μικρότερο ή ίσο
  - Is : η ταυτότητα του αντικείμενου
  - Is not: δεν είναι αυτή η ταυτότητα του αντικειμένου

```
12
13    if x == 1 :
14        print ('one')
15        elif x== 2:
16            print ('two')
17        else:
18            print('the number is greater than "X"')
```

Εικόνα 8: If Elif Else παράδειγμα.

- Λογικοί Τελεστές [10]
  - And : Επιστρέφει True αν και οι δυο συνθήκες είναι αληθής
  - Or : Επιστρέφει True αν μια από τις δυο συνθήκες είναι αληθής
  - Not : Επιστρέφει True αν καμία από τις δυο συνθήκες δεν είναι αληθής

```
5<6 and 7<8
True
5<6 and 7>8
False
not 5<6
False
```

Εικόνα 9: Συγκριτικοί Τελεστές.

4.3 Βρόγχος (Loop)

Οι επαναλήψεις είναι ένα θεμελιώδεις κομμάτι στον προγραμματισμό καθώς πραγματοποιείται μια συνεχής εκτέλεσης ενός κομματιού κώδικα με σκοπό να επιτευχθεί μια συγκεκριμένη εργασία. Οι επαναλήψεις εκτελούνται μέχρις ότου μια συνθήκη γίνει ψευδής ή μέχρι να εξαντληθεί μια λίστα ή άλλη μορφή δεδομένων [11].

Οι επαναλήψεις επιτυγχάνονται με την χρήση λέξεων – κλειδιών, όπως για παράδειγμα if, for, while, try και expect.

4.3.1 Βρόγχος For

Η εντολή for είναι από τις ποιο ισχυρές επαναλήψεις και χρησιμοποιείτε μέχρι να εκπληρωθεί μια συνθήκη. Η συγκεκριμένη διαδικασία χρησιμοποιείτε κύριος στην επανάληψη στις δομές συλλογών (όπως λίστα, πλειάδα, λεξικό, σύνολο κλπ.)

Η εντολή for είναι πολύ ισχυρή και ευέλικτη, καθώς μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές περιπτώσεις για την επανάληψη και την επεξεργασία στοιχείων από συλλογές δεδομένων.

Χαρακτηριστική είναι η επανάληψη για τα λεξικά, καθώς μπορεί να πραγματοποιηθεί επανάληψη για περισσότερες μεταβλητές [11].

```
PairOfStudentsAndGrades = {
    'Alice': 85,
    'Bob': 92,
    'Charlie': 78
}

for student, grade in PairOfStudentsAndGrades.items():
    print(f'Student: {student}, Grade: {grade}')
```

Εικόνα 10: Επανάληψη For μέσα σε ένα λεξικό.

4.3.2 Βρόγχος While

Η εντολή while διασφαλίζει την συνεχή επανάληψη σε ένα κομμάτι κώδικα μέχρις ότου η συνθήκη να γίνει αληθείς. Στη συγκεκριμένη δήλωση εφαρμόζονται πολύ τα κριτήρια αποδοχής (acceptance criteria) όπως για παράδειγμα μια μεταβλητής πρέπει να λαμβάνει μόνο τιμές yes/no και να μη δέχεται άλλες καταχωρήσεις.

```
while decision.lower() != 'yes' and decision.lower() != 'no':
    decision = input("Do you want to do another transaction? Yes/No:")
```

Εικόνα 11: Acceptance Criteria Yes/No.

```
Do you want to do another transaction? Yes/No:vai
Do you want to do another transaction? Yes/No:yes
--|--------------------|
1.|   Checking Account  |
--|--------------------|
2.|   Savings Account   |
```

Εικόνα 12 Σε περίπτωση που δεν καλύπτοντες οι ανάγκες ρωτάει πάλι αλλιώς προχωράει στην επόμενη συνάρτηση.

Σημαντική διαφορά μεταξύ for και while είναι ότι η while loop είναι αγνώστου αριθμού επανάληψη, ενώ η for είναι συγκεκριμένου. Τέλος στην επανάληψη while, εάν υπάρχει κάποιο λάθος στον κώδικα και στα κριτήρια της επανάληψης παρατηρείται και ο ατρεμών βρόγχος [11].

```python
age = 28

# the test condition is always True
while age > 19:
    print('Infinite Loop')
```

Εικόνα 13 Μια συνθήκη που ισχύει συνέχεια οπότε δεν βγαίνει ποτέ από το while.

4.3.3 Βρόγχος Try Expect Else Finally

Στην Python υπάρχουν 2 ήδη σφαλμάτων, τα συντακτικά και οι εξαιρέσεις. Τα συντακτικά λάθη θα οδηγήσουν στην διακοπή εκτέλεσης του κώδικα ενώ οι εξαιρέσεις έρχονται όταν κάποιες εσωτερικές διαδικασίες αλλάζουν την κανονική ροή του κώδικα [12]. Όπως για παράδειγμα το tax_id

```
while self.tax_id <= 0:
    try:
        self.tax_id = int(input("Hello, please input a Tax ID: "))
        if self.tax_id <= 0:
            print("Your Tax ID must be positive")
    except ValueError:
        print("Invalid input. Please enter an integer.")
        continue
    else:
        self.customer_id = str(uuid.uuid4())
        account_details = {
            'name': self.name.title(),
            'tax_id': self.tax_id,
            'Total Balance': 0,
            'accounts': {
                'Checking Account': self.default_value,
                'Savings Account': self.default_value,
                'DepositGuard Account': self.default_value,
                'Business Account': self.default_value
            }
        }
```

Εικόνα 14: Παράδειγμα για το Try Except Else.

Στο συγκεκριμένο παράδειγμα το tax_id πρέπει να είναι αριθμός και μόνο αριθμός. Για τον σκοπό αυτόν χρησιμοποιείτε η δήλωση Try and Except.

Πιο αναλυτικά:

- Exception error
  - IOError: εάν το αρχείο δεν μπορεί να ανοίξει.
  - KeyboardInterrupt: όταν ο χρήστης πατήσει λάθος πλήκτρο.
  - ValueError: Όταν η συνάρτηση λαμβάνει λάθος όρισμα (πιο κοινό error).
  - ImportError: Όταν δεν μπορεί να βρει το import – module
- Try:
  - Αρχικά εκτελείται ο κώδικας στο Try. Εάν προκύψει Exception error (σφάλμα) ο κώδικας συνεχίζει να τρέχει.
- Else:
  - Αν δεν προκύψει καμία εξαίρεση, ο κώδικας μέσα στο κομμάτι else θα εκτελεστεί.
- Finally:
  - Ο κώδικας σε αυτό το κομμάτι θα εκτελεστεί ανεξαρτήτως των παραπάνω.

4.4 Επανάληψη μέσα σε επανάληψη (Nested loop)

Στον προγραμματισμό, βρόγχοι μέσα στους βρόγχους είναι αρκετά χρήσιμοι για επανάληψη σε περίπλοκες δομές δεδομένων με περισσότερες από μια διαστάσεις όπως λίστες λιστών η πίνακες.

Η σειρά με την οποία εκτελούνται οι εντολές είναι [11]:

- Ο εξωτερικός βρόγχος ξεκινάει και εκτελεί την πρώτη του επανάληψη.
- Μέσα στην πρώτη επανάληψη του εξωτερικού βρόγχου, **ο** εσωτερικός βρόγχος ξεκινάει και εκτελεί ολόκληρο τον κύκλο του.
- Αφού ολοκληρωθούν όλες οι επαναλήψεις του εσωτερικού βρόγχου, η εκτέλεση επιστρέφει στον εξωτερικό βρόγχο, ο οποίος προχωράει στη δεύτερη επανάληψή του.
- Η διαδικασία επαναλαμβάνεται: για κάθε επανάληψη του εξωτερικού βρόγχου, ο εσωτερικός βρόγχος εκτελεί όλες τις επαναλήψεις του από την αρχή.

```
for i in range(1, 4):  # Εξωτερικός βρόχος
    for j in range(1, 4):  # Εσωτερικός βρόχος
        print(f"i = {i}, j = {j}")
```

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

Εικόνα 15: Επανάληψη σε πολυδιάστατο αντικείμενο.

Στο παράδειγμα αυτό, ο εξωτερικός βρόγχος εκτελείτε 3 φόρες ενώ ο εσωτερικός 3 για κάθε μια του εξωτερικού: σύνολο 9.

4.5 Break/Continue/Pass

Μέσω των δυο δηλώσεων Break και Continue ελέγχεται η ροή εκτέλεσης μέσα στις επαναλήψεις for και while [9].

4.5.1 Break

Το Break χρησιμοποιείται για την πρόωρη διακοπή μιας επανάληψης [9]. Δηλαδή όταν εκτελεστεί η δήλωση Break τότε ο έλεγχος εξέρχεται από τον βρόγχο που βρίσκεται και συνεχίζει με την επόμενη εντολή στον επόμενο βρόγχο.

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```
```
1
2
3
4
```

Εικόνα 16: Η επανάληψη σταματάει λόγο του break στο 5 αντί του 10.

Στο συγκεκριμενο παραδειγμα γινεται print το i για τους αριθμους 1 έως 10, και όταν φτασει στο 5 βγαίνει από την επαναληψη οποτε γινεται print το 1,2,3,4.

4.5.2 Continue

Το Continue χρησιμοποιείτε για την παράκαμψη της υπόλοιπης εκτέλεσης του βρόγχου για την τρέχουσα επανάληψη εάν η συνθήκη για παράδειγμα εκπληρώνετε [9].

```
1  for i in range(1, 10):
2      if i == 5 or i == 7:
3          continue
4      print(i)
```
```
1
2
3
4
6
8
9
```

Εικόνα 17: Μέσω του Continue δεν γίνετε η επανάληψη στο 5 και στο 7.

Στο συγκεκριμενο παραδειγμα γινετε print το i για τους αριθμου 1 εως 10 και όταν φτασει στο 5 και στο 7 τοτε παρακαμπτει το print και συνεχειζει την επαναληψη.

4.5.3 Pass

Το pass εξασφαλίζει ότι δεν γίνεται τίποτα. Λειτουργεί σαν το προσχέδιο του κώδικα και καταχωρείται στο τέλος κάθε συνάρτησης ή ακόμα και κλάσης. Στην Python η δήλωση Pass χρησιμοποιείται όταν μια συνάρτηση ή κλάση απαιτείται συντακτικά αλλά δεν χεριάζετε να εκτελεστεί καμία εντολή ή δεν υπάρχει καθόλου κώδικας μέσα σε αυτήν [9].

```
def IneedThisFunctionForLater():
    pass
```

Εικόνα 18: "Placeholder" το Pass.

4.6 Print/return/ f-strings

Είναι κάποιες από τις βασικές εντολές επικοινωνίας μεταξύ προγράμματος και χρήστη [13].

Πιο αναλυτικά:

4.6.1 Print

Η εντολή Print χρησιμοποιείται για την άμεση επικοινωνία μεταξύ κονσόλας και χρήστη [13] . Είναι ένας τρόπος μεταφοράς πληροφορίας προς τον χρήστη.

```
while self.action.lower() not in ['login', 'register', 'withdraw', 'deposit', 'return', 'quit', 'yes', 'no']:
    self.action = input('Welcome to Python Bank!\nWould you like to login or register?')
```

Εικόνα 19: Acceptance Criteria και Print.

```
Welcome to Python Bank!
Would you like to login or register?
```

Εικόνα 20: Το τι βλέπει ο χρήστης.

4.6.2 Return

Η εντολή Return χρησιμοποιείται για να επιστρέφει ένα αποτέλεσμα από μια συνάρτηση καθώς δίνει και την δυνατότητα και σε άλλες συναρτήσεις του προγράμματος να χρησιμοποιήσουν αυτήν την μεταβλητή [14].

```
def calculate_sum(a, b):
    return a + b

result = calculate_sum( a: 3,  b: 5)
print(f"The sum is: {result}")
```

Εικόνα 21: Το return δεν κάνει Print αλλά επιστρέφει την τιμή της συναρτάς.

```
The sum is: 8
```

Εικόνα 22: Κάνει Print το αποτέλεσμα του Return.

4.6.3 F strings

Τα f-string είναι ένας εναλλακτικός τρόπος με τον οποίο συνδικάζονται οι εντολές Print και Return, δημιουργώντας δυναμικές συμβολοσειρές [15]. Είναι ιδιαίτερα χρήσιμα καθώς εμφανίζει πληροφορίες στον χρήστη με ευανάγνωστο και ευχάριστο τρόπο. Εν ολίγης καταχωρείται η μεταβλητή μέσα στη

συμβολοσειρά, κάνοντας ένα στατικό Print σε κάτι πιο δυναμικό, αναλόγως τα στοιχεία ή τα δεδομένα που καταχωρεί ο χρήστης.

```
print ('Hello, my name is Fotopoulos Panagiotis i am 27 years old .And this is my Thesis')
```

Εικόνα 23: Στατικό Print.

```
first_name = 'Panagiotis'
second_name =  'Fotopoulos'
my_age = 27
print(f"Hello, my name is {first_name} {second_name} iam  {my_age} years old. And this is my Thesis ")
```

Εικόνα 24: Δυναμικά F-strings.

```
Hello, my name is Panagiotis Fotopoulos iam  27 years old. And this is my Thesis
```

Εικόνα 25 Το Print του F-string.

5 Βασικές δομές που χρησιμοποιήθηκαν στα πλαίσια της εφαρμογής

5.1 Python Γιατί;

Η Python ξεχωρίζει ως μία από τις πιο ευρέως διαδεδομένες γλώσσες προγραμματισμού για πολλούς λόγους. Επέλεξα αυτή τη γλώσσα για το έργο μου διότι προσφέρει ευκολία στην κατανόηση θεμελιωδών αρχών όπως ο αντικειμενοστραφής προγραμματισμός και η κληρονομικότητα.

Η δομή της Python είναι εξαιρετικά απλή, και κατανοώντας τα παραπάνω μπορεί κανείς εύκολα να επεκτείνει τις γνώσεις είτε εμβαθύνοντας πολύ στην συγκεκριμένη γλώσσα είτε μαθαίνοντας και άλλες γλώσσες προγραμματισμού. Οι έννοιες που μαθαίνει κάποιος με τη χρήση της Python είναι θεμελιώδεις, όχι μόνο για την Python, αλλά γενικότερα για τον προγραμματισμό, και μπορούν να εφαρμοστούν εύκολα και σε άλλα περιβάλλοντα και γλώσσες.

Όλες οι γλώσσες προγραμματισμού μπορούν να κάνουν την ίδια εφαρμογή να λειτουργεί το ίδιο αποτελεσματικά, ωστόσο ο λόγος που υπάρχει πληθώρα γλωσσών είναι ότι η κάθε μια εξειδικεύεται σε κάτι συγκεκριμένο και αυτό επιτυγχάνεται με την δημιουργία και χρήση βιβλιοθηκών και Frameworks. Η Python διαθέτει μια εκτενή συλλογή από βιβλιοθήκες και frameworks κατακτώντας την κορυφή ιδιαίτερα στον χώρο της επιστήμης, εστιάζοντας στην ανάλυση δεδομένων τα οποία είναι βασικά χαρακτηριστικά για τους τομείς της τεχνητής νοημοσύνης και της μηχανικής μάθησης.

Αυτές οι ιδιαιτερότητες καθιστούν την Python μία από τις πιο επιτυχημένες γλώσσες προγραμματισμού σήμερα, καθώς προσφέρει ένα ευέλικτο, δυναμικό και εύχρηστο περιβάλλον για την ανάπτυξη λογισμικού και την ανάλυση δεδομένων.

5.2 Βιβλιοθήκες

5.2.1 Τι είναι βιβλιοθήκη και βασικά παραδείγματα

Οι βιβλιοθήκες στην Python είναι σύνολα προκαθορισμένου κώδικα που παρέχουν διάφορες λειτουργίες και εργαλεία για να διευκολύνουν την ανάπτυξη λογισμικού. Αυτές οι βιβλιοθήκες περιλαμβάνουν χρήσιμες συναρτήσεις, κλάσεις και μεθόδους που επιτρέπουν στους προγραμματιστές να εκτελέσουν συγκεκριμένες εργασίες χωρίς να χρειάζεται να γράψουν τον κώδικα από την αρχή [16].

Οι βιβλιοθήκες της Python καλύπτουν πολλούς τομείς, όπως η επεξεργασία δεδομένων, ο υπολογισμός, η ανάπτυξη web εφαρμογών, η επιστημονική υπολογιστική, η τεχνητή νοημοσύνη και πολλά άλλα. Οι βιβλιοθήκες αυτές συχνά είναι διαθέσιμες μέσω του Python Package Index (PyPI), ενός κεντρικού αποθετηρίου λογισμικού για τη γλώσσα Python, όπου οι προγραμματιστές μπορούν να αναζητήσουν, να εγκαταστήσουν και να διανέμουν πακέτα λογισμικού.

.

5.2.2 Βιβλιοθήκες που χρησιμοποιήθηκαν με σκοπό την υλοποίηση του έργου και γιατί

Οι βιβλιοθήκες που χρησιμοποιήθηκαν με σκοπό την υλοποίηση του έργου είναι οι εξής

- ✓ **UUID**(Universally Unique Identifiers)**:** Είναι μια βιβλιοθήκη η οποία παρέχει τη δυνατότητα δημιουργίας μοναδικών αλφαριθμητικών ακολουθιών δίνοντας την σιγουριά της ασφαλείας καθώς οι πιθανότητες για να για να παράγει ίδιο id αν η ακολουθία ήταν μόνο αριθμοί φτάνει το 1 στα 10^{32}, αλλά αν προσθέσουμε και τα γράμματα του αγγλικού αλφαβήτου οι πιθανότητες να παράγει ίδιο ID αγγίζουν το 1 στα 36^{32}. [17]

  Ο λόγος που χρησιμοποιήθηκε αυτή η βιβλιοθήκη στον κώδικα είναι για την δημιουργία μιας μοναδικής ακολουθίας – κωδικός με σκοπό την προσομοίωση της τραπεζικής κάρτας.

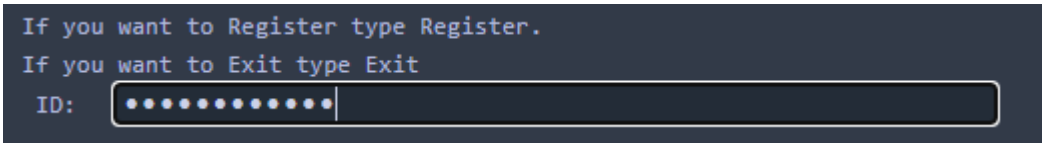
```
Your unique ID is 48b716da-8e24-4620-99ea-2a9e1d88b611
```

Εικόνα 26: UUID Generator.

- ✓ **GetPass**: η βιβλιοθήκη getpass χρησιμοποιείται κατά κύριο λόγο όταν υπάρχει αλληλεπίδραση μεταξύ χρήστη και προγράμματος (inputs) καλύπτοντας τις εισαγωγές με σύμβολα και

εξαλείφοντας τον κίνδυνο για τυχών διαρροές δίνοντας ασφάλεια και προστασία στα προσωπικά δεδομένα [18].

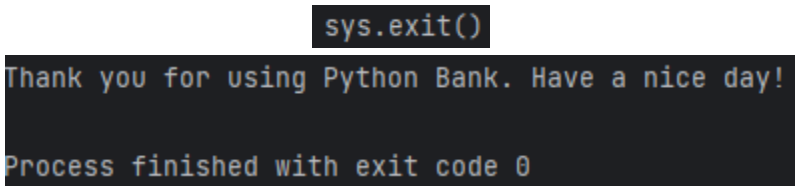
Η βιβλιοθήκη αυτή χρησιμοποιήθηκε στον κώδικα κατά τη διαδικασία της σύνδεσης και έχει ως σκοπό την αντικατάσταση του μοναδικού κωδικού κάθε χρήστη με τις τέλειες κρύβοντας την ταυτότητά του.



Εικόνα 27: Αποτέλεσμα χρήσης βιβλιοθήκης GetPass.

- ✓ **Sys**(system): η βιβλιοθήκη Sys στην Python δίνει την δυνατότητα της εκτέλεσης λειτουργιών που σχετίζονται με το σύστημα [19].

  Στο συγκεκριμένο πρόγραμμα η λειτουργία sys.exit()τερματίζει το πρόγραμμα.

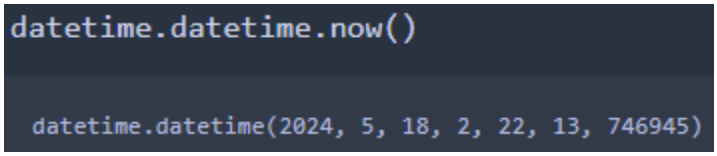


Εικόνα 28: Η συνάρτηση System Exit και το μήνυμα για τον χρήστη.

«Process finished with exit code 0» σημαίνει ότι ο κώδικας εκτελέστηκε χωρίς να επιστρέψει κάποιο πρόβλημα ή κάποια δυσλειτουργική μέθοδος να διακόψει την διαδικασία και την ροή του κώδικα.

- ✓ **Time & Datetime** είναι μια βιβλιοθήκη η οποία διαθέτει μεθόδους και κλάσεις που παρέχουν πρόσβαση και διαμόρφωση ημερομηνίας και ώρας [20].

Η εφαρμογή της συγκεκριμένης βιβλιοθήκης επιτρέπει το μέτρημα των ημερών μέσα στον μηνά ώστε με την κατάλληλη χρήση κώδικα να επιτυγχάνονται διάφορες διαδικασίες (όπως reset ενός μετρητή κάθε πρώτη του μηνά).



Εικόνα 29:Datetime Στήν παραπάνω φωτογραφεία διατυπώνετε ένα παράδειγμα (Year/month/day / hour minutes seconds milliseconds).

- ✓ **CSV**(Coma Separated Values “,”) Η βιβλιοθήκη αυτή δίνει προσβάσεις σε λειτουργίες CSV καθώς αυτά είναι εύχρηστα για αποθήκευση και μελλοντική χρήση [21].

Στο συγκεκριμένο πρόγραμμα χρησιμοποιείται ένα dictionary(λεξικό) με σκοπό την επίτευξη μιας υποτυπώδους database. Η συγκεκριμένη βιβλιοθήκη χρησιμοποιείται ώστε να μετατρέψει το παρακάτω database σε ένα csv αρχείο το οποίο είναι ιδανικό για την αρχειοθέτηση πληροφοριών.

```
database

{'90074d66-a6dc-48d3-b0c3-8e38c37b49f8': {'name': 'Panos',
  'tax_id': 1234,
  'Total Balance': 7000,
  'accounts': {'Checking Account': 1000,
   'Savings Account': 2000,
   'DepositGuard Account': 3000,
   'Business Account': 1000}}}
```

Εικόνα 30: Προσομοίωση Database.

Εδώ εμφανίζεται η δομή της βάσης δεδομένων. Παρατηρείται ότι ακολουθεί δομή dictionary, δομείται δηλαδή από ζευγάρια (keys:values). Στο παραπάνω, το UUID είναι το key και τα περεταίρω στοιχεία του χρήστη αποτελούν το value του key UUID(ID), στην συνέχεια μέσω της βιβλιοθήκης csv και της δημιουργία μιας μεθόδου θα επιτευχθεί η μετατροπή της βάσης δεδομένων σε csv, όπως θα δούμε παρακάτω.

```
1 uuid,name,tax_id,Total Balance,accounts
2 90074d66-a6dc-48d3-b0c3-8e38c37b49f8,Panos,1234,7000,"{'Checking Account': 1000, 'Savings Account': 2000,
  'DepositGuard Account': 3000, 'Business Account': 1000}"
3
```

Εικόνα 31: CSV αρχείο βάση του παραπάνω Database.

- ✓ **Ast** (Abstract Syntax Trees):Μια βιβλιοθήκη με την οποία η Python μετατρέπει μια συμβολοσειρά (string) σε αναπαράσταση λεξικού (dictionary) [22].

5.3 Κλάσεις

Account: Η βασική κλάση που αντιπροσωπεύει τους λογαριασμούς των πελατών.

- CheckingAccount: Υποκλάση της Account που αντιπροσωπεύει τους τραπεζικούς λογαριασμούς όψεως.
- Savings Account: Υποκλάση της Account που αντιπροσωπεύει τους τραπεζικούς λογαριασμούς αποταμίευσης.
- DepositGuardAccount: Υποκλάση της Account που αντιπροσωπεύει τους λογαριασμούς προστασίας καταθέσεων.

5.4 Σημαντικές Μεθόδοι:

- __init__(): Η μέθοδος κατασκευής για κάθε κλάση, όπου ορίζονται οι μεταβλητές και οι αρχικές τους τιμές.
- Super().__init__(): είναι μια μέθοδος που χρησιμοποιείται συνήθως στην Python για να κληθεί η αρχικοποίηση (initializer) της γονικής κλάσης (superclass) από την υποκλάση (subclass).
- Initialize subclass instances(): Δημιουργία των subclass.
- Ask_user_for_action(): Ζητά από τον χρήστη να επιλέξει μια δράση, όπως σύνδεση ή εγγραφή.
- Register(): Επιτρέπει σε νέους χρήστες να εγγραφούν στο σύστημα.
- Login(): Επιτρέπει σε υπάρχοντες χρήστες να συνδεθούν στο σύστημα.
- Quit(): Τερματίζει την εφαρμογή και αποθηκεύει τα δεδομένα σε ένα αρχείο CSV.
- Choosingaccount(): Ζητά από τον χρήστη να επιλέξει έναν τύπο λογαριασμού.
- Show_account_info(): Εμφανίζει πληροφορίες λογαριασμών για τον τρέχοντα χρήστη.
- Ask_user_for_account_action(): Ζητά από τον χρήστη να επιλέξει μια δράση για έναν συγκεκριμένο τύπο λογαριασμού.
- Deposit(): Επιτρέπει στον χρήστη να καταθέσει χρήματα σε έναν λογαριασμό.
- Withdraw(): Επιτρέπει στον χρήστη να αποσύρει χρήματα από έναν λογαριασμό, αν είναι δυνατό.
- Returning (): Ερωτά τον χρήστη αν επιθυμεί να πραγματοποιήσει άλλη ενέργεια ή να τερματίσει την εφαρμογή.
- Update_total_balance (): Υπολογίζει το συνολικό υπόλοιπο ενός χρήστη.
- Day_check(): Ελέγχει αν είναι η πρώτη μέρα του μήνα για την εφαρμογή ορισμένων ενεργειών.
- Update total balance(): Ανανέωση του συνολικού υπολοίπου.
- Print with dots(): Εκτυπώνει Loading… όταν ο χρηστής επιλέγει να γυρίσει πίσω.
- Get amount Input(): Αλληλοεπιδρά με τον χρήστη για την καταχώριση κεφαλαίου.
- Csv to nested dict1(): Σπάει την δομή της database, δίνοντας την δυνατότητα για να εγγραφεί σε csv αρχείο και το αντίστροφο.

Η γενική αρχιτεκτονική του κώδικα που παρουσιάζεται βασίζεται σε μια δομή αντικειμενοστραφούς προγραμματισμού (OOP). Αυτό σημαίνει ότι ο κώδικας οργανώνεται γύρω από αντικείμενα που αντιπροσωπεύουν διάφορες έννοιες και λειτουργίες του τραπεζικού συστήματος.

Κάθε λειτουργία ή λειτουργικότητα αντιπροσωπεύεται από μια κλάση, η οποία μπορεί να περιλαμβάνει μεθόδους για την εκτέλεση συγκεκριμένων εργασιών. Επιπλέον, η κληρονομικότητα χρησιμοποιείται εκτενώς για την αποφυγή διπλού κώδικα και την οργάνωσή του σε ιεραρχικές δομές.

Αναλυτικά, η δομή του κώδικα έχει ως εξής :

5.4.1 def__Init__()

Account (Λογαριασμός):Το __init__ είναι μια ειδική μέθοδος (constructor) στην Python που χρησιμοποιείται για την αρχικοποίηση των νέων αντικειμένων

```
class Account():
    def __init__(self):
        self.withdrawals_times = 0
        self.days_count = 0
        self.default_value = 0
        self.customer_id = 0
        self.name = 1
        self.tax_id = 0
        self.total_account_balance = 0
        self.ck_account = CheckingAccount
        self.sv_account = SavingsAccount
        self.dg_account = DepositGuardAccount
        self.bs_account = BusinessAccount
        self.account_type = ''
        self.action = ''
        self.amount = 0
        self.account_type_num = ''
```

Εικόνα 32: Αρχικοποίηση δεδομένων def __init__().

5.4.2 Initialize subclass instances

Εδώ επιτυγχάνετε η Δημιουργία και αρχικοποίηση υποκλίσεων μέσω της μεθόδου initialize_subclass_instances. Κάθε υποκλάση που δημιουργείται αντιστοιχεί και σε ένα τύπο τραπεζικού λογαριασμού με διαφορετικές λειτουργίες και χαρακτηριστικά.

```
def initialize_subclass_instances(self, customer_id, account_type, action, default_value, days_count,
                                  withdrawals_times):
    self.ck_account = CheckingAccount(customer_id, account_type, action, default_value)
    self.sv_account = SavingsAccount(customer_id, account_type, action, default_value, days_count, withdrawals_times)
    self.dg_account = DepositGuardAccount(customer_id, account_type, action, default_value)
    self.bs_account = BusinessAccount(customer_id, account_type, action, default_value, days_count)
```

Εικόνα 33:Subclass initialization.

5.4.3 Ask user for action

Δημιουργία λογικής στον κώδικα με σκοπό την επίτευξη βασικών λειτουργιών: Η μέθοδος ask_user_for_action είναι η μέθοδος η οποία όταν λαμβάνει εντολές από τον χειρίστη το μετατρέπει στην αντίστοιχη ενέργεια.

```
def ask_user_for_action(self):
    while self.action.lower() not in ['login', 'register', 'withdraw', 'deposit', 'return', 'quit', 'yes', 'no']:
        self.action = input('Welcome to Python Bank!\nWould you like to login or register?')
    if self.action.lower() == "login":
        self.login()
    elif self.action.lower() == "register":
        self.register()
    elif self.action.lower() == ["withdraw"]:
        self.withdraw()
    elif self.action.lower() == ["deposit", 'yes']:
        self.deposit()
    elif self.action.lower() == ['return', 'no']:
        self.returning()
    elif self.action.lower() == 'quit':
        self.quit()
```

Εικόνα 34: Η συνάρτηση Ask User For Action Logic 1.

Στο συγκεκριμένο σημείο χρησιμοποιείται και το not in σε μια λίστα . Αυτό δίνει περιορισμένες επιλογές προς χρήση και κάθε φόρα που ο χρήστης δίνει μια απάντηση η οποία δεν είναι μέσα σε αυτήν την λίστα τότε του ζητά νέα καταχώρηση. Η αλληλεπίδραση χρήστη-προγράμματος γίνεται κατά κύριο λόγο μέσω της χρήσης του input η οποία επιτυγχάνει την παραπάνω διαδικασία.

```
Welcome to Python Bank!
Would you like to login or register?
```

Εικόνα 35 Interacting with user asking for login or register.

5.4.4 Register

Το register πραγματοποίει την Δημιουργία λογαριασμού: η μέθοδος έχει ως σκοπό να αντλήσει βασικές πληροφορίες του χρήστη όπως όνομα και φορολογικό του αριθμό, στην συνέχεια δημιουργεί μια βάση δεδομένων (dictionary Json style) οπού γεφυρώνονται τα στοιχεία με τα αντίστοιχα πεδία στην βάση δεδομένων.

```python
def register(self):
    while True:
        self.name = input("Provide a username or type 'back' in order to login!: ")
        if self.name.lower() == 'back':
            self.back()
            break
        elif not self.name.isalpha():
            print("Username can only contain alphabetic characters!")
        else:
            break
    while self.tax_id <= 0:
        try:
            self.tax_id = int(input("Hello, please input a Tax ID: "))
            if self.tax_id <= 0:
                print("Your Tax ID must be positive")
        except ValueError:
            print("Invalid input. Please enter an integer.")
            continue
        else:
            self.customer_id = str(uuid.uuid4())
            account_details = {
                'name': self.name.title(),
                'tax_id': self.tax_id,
                'Total Balance': 0,
                'accounts': {
                    'Checking Account': self.default_value,
                    'Savings Account': self.default_value,
                    ' DepositGuard Account': self.default_value,
                    'Business Account': self.default_value
                }
            }
```

Εικόνα 36: Η συνάρτηση Register.

5.4.5 Login

Η συγκεκριμένη λειτουργεία προσφέρει την επιλογή στον χρήστη για σύνδεση στο τραπεζικό σύστημα σε περίπτωση που έχει ήδη εγγραφή πριν. Κάνοντας την κατάλληλη αντιστοίχιση των στοιχείων του πελάτη από το csv αρχείο στα στοιχεία της database.

```
def login(self):
    while True:
        self.user_id = str(getpass.getpass('Hello, Please enter your user ID.\nIf you want to Register t
        if self.user_id in database:
            self.customer_id = self.user_id
            self.initialize_sublcass_instances(self.customer_id,self.account_type,self.action,self.defau
            print(f"Welcome Mr/Mrs {database[self.customer_id]['name']}, we are happy to see you again!'
            self.show_account_info(False)
            self.choosingaccount()
            break
```

Εικόνα 37 Η συνάρτηση Login.

5.4.6 Quit

Η λειτουργεία αυτή εκτελείται όταν ο χρήστης αποφασίσει να τερματίσει το πρόγραμμα. Μέσω της for δημιουργείτε μια επανάληψη με την οποία τα στοιχεία της database εγγράφονται στο csv αρχείο με όνομα «OfficialDB.csv».

```
def quit(self):
    print("Thank you for using Python Bank. Have a nice day!")
    fieldnames = ['uuid','name','tax_id','Total Balance','accounts']
    with open('OfficialDB.csv', 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for random_number, inner_dict in database.items():
            row = {'uuid': random_number}
            row.update(inner_dict)
            writer.writerow(row)
            self.action = ''
    sys.exit
```

Εικόνα 38: Η συνάρτηση Quit

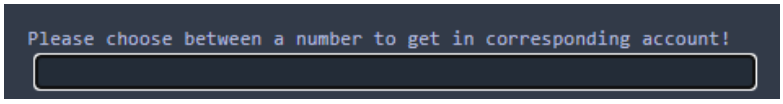
```
uuid,name,tax_id,Total Balance,accounts
bae07227-7f3e-4753-8740-4664f6e7f578,Panos,1234,10000,"{'Checking Account': 1000, 'Savings Account': 2000,
'DepositGuard Account': 3000, 'Business Account': 4000}"
7cbd1729-050f-4c1f-acf4-cf3cecf0eda1,Nikos,12345,11110,"{'Checking Account': 1111, 'Savings Account': 2222,
'DepositGuard Account': 3333, 'Business Account': 4444}"
```

Εικόνα 39 Το αποτέλεσμα της συνάρτησης Quit.

5.4.7 Choosingaccount

Επιλογή λογαριασμού: Η μέθοδος def choosingaccount(self) προβάλει τους λογαριασμούς – subclasses του τραπεζικού συστήματος στον χρήστη αναμένοντας την επιλογή του χρήστη.

```
def choosingaccount(self):
    self.account_type_num = ''
    while self.account_type_num not in ['1', '2', '3', '4']:
        print('--|--------------------|')
        print('1.|   Checking Account  |')
        print('--|--------------------|')
        print('2.|   Savings Account   |')
        print('--|--------------------|')
        print('3.| DepositGuard Account|')
        print('--|--------------------|')
        print('4.|   Business Account  |')
        print('--|--------------------|')
        self.account_type_num = (input('Please choose between a number to get in corresponding account! '))
        self.ask_user_for_account_action()
```



Εικόνα 40:ChoosingAccount

5.4.8 Show account info

Η συνάρτηση def show_account_info(self) εμφανίζει πληροφορίες για τον λογαριασμό ενός χρήστη. Εκτυπώνει είτε τα βασικά στοιχεία εισόδου (όπως ο αριθμός του χρήστη και το όνομά του), είτε αναλυτικές πληροφορίες για τον λογαριασμό του, συμπεριλαμβανομένων του ονόματός του, του φορολογικού του αριθμού και του συνολικού υπολοίπου του λογαριασμού, καθώς και το κεφάλαιο για κάθε τύπο λογαριασμού που διαθέτει.

```python
def show_account_info(self, register):
    print('Account Details')
    if register:
        print(f'User ID: {self.customer_id} ----- You will need this to login!')
    else:
        print(self.customer_id)
        print(f"Name : {database[self.customer_id]['name']}")
        print(f"Tax ID: {database[self.customer_id]['tax_id']}")
        print(f"Your Total Account Balance is : {database[self.customer_id]['Total Balance']}" + '€')
        print(
            f"• Checking's Account Balance is : {database[self.customer_id]['accounts']['Checking Account']}" + '€')
        print(f"• Saving's Account Balance is : {database[self.customer_id]['accounts']['Savings Account']}" + '€')
        print(
            f"• DepositGuard's Account Balance is : {database[self.customer_id]['accounts']['Checking Account']}" + '€')
        print(
            f"• Business's Account Balance is : {database[self.customer_id]['accounts']['Business Account']}" + '€')
```

Εικόνα 41: Account Information

5.4.9 Ask user for account action

Η συνάρτηση def ask_user_for_account_action(self) είναι μια από τις ποιο σημαντικές συναρτήσεις στον παρόν κώδικα καθώς αποτελεί την λογική του προγράμματος με την οποία γίνονται όλες οι ενέργειες στο τραπεζικό σύστημα καθώς ζητά από τον χρήστη να επιλέξει με μια ενέργεια στον τρέχοντα λογαριασμό του. Ανάλογα με τον τύπο του λογαριασμού που έχει επιλέξει ο χρήστης, ο κώδικας αντιστοιχίζει το subclass, και του ζητείται να εκτελέσει μια από τις παρακάτω ενέργειες: κατάθεση, ανάληψη, επιστροφή στην προηγούμενη επιλογή λογαριασμού ή διακοπή της διαδικασίας. Αν η επιλογή του χρήστη δεν είναι έγκυρη, τότε του ζητείται να επιλέξει μια έγκυρη ενέργεια.

```
def ask_user_for_account_action(self):
    if self.account_type_num == "1":
        self.account_type = ('Checking Account')
        print('You are Currently in your Checking Account')
        self.action = ''
        self.action = input(
            'Do you want to deposit or withdraw. Press "back" if u willing to choose another account or "quit" to stop the process?')
        if self.action.lower() == "deposit":
            CheckingAccount(self.customer_id, self.account_type, self.action, self.default_value).deposit()
        elif self.action.lower() == 'withdraw':
            CheckingAccount(self.customer_id, self.account_type, self.action, self.default_value).withdraw()
        elif self.action.lower() == 'back':
            self.back()
        elif self.action.lower() == 'quit':
            self.quit()
        elif self.action.lower() != 'deposit' or self.action.lower() != 'withdraw' or self.action != 'back' or self.action != 'quit':
            print('Please enter a valid action!')
            self.ask_user_for_account_action()
    elif self.account_type_num == "2":
        self.account_type = ('Savings Account')
        print('Savings Account')
        self.action = ''
        self.action = input(
            'Do you want to deposit or withdraw. Press "back" if u willing to choose another account or "quit" to stop the process?')
        if self.action.lower() == "deposit":
            SavingsAccount(self.customer_id, self.account_type, self.action, self.default_value, self.days_count,
                           self.withdrawls_times).deposit()
        elif self.action.lower() == 'withdraw':
            SavingsAccount(self.customer_id, self.account_type, self.action, self.default_value, self.days_count,
                           self.withdrawls_times).withdraw()
        elif self.action.lower() == 'back':
            self.back()
        elif self.action.lower() != 'deposit' or self.action.lower() != 'withdraw' or self.action != 'back' or self.action != 'quit':
            print('Please enter a valid action!')
            self.ask_user_for_account_action()
    elif self.account_type_num == '3':
        self.account_type = ('DepositGuard Account')
        print('DepositGuard Account')
        self.action = ''
        self.action = input(
            'DepositGuard Account only allows deposits, not withdrawals.\nDo you want to continue? Yes/No')
        if self.action.lower() == 'yes':
            DepositGuardAccount(self.customer_id, self.account_type, self.action, self.default_value).deposit()
        elif self.action == 'no':
            self.back()
        else:
            print('Please enter a valid action!')
            self.ask_user_for_account_action()
    elif self.account_type_num == '4':
        self.account_type = ('Business Account')
        print('Business Account')
        self.action = ''
        self.action = input(
            'Do you want to deposit or withdraw. Press "back" if u willing to choose another account or "quit" to stop the process?')
        if self.action.lower() == "deposit":
            BusinessAccount(self.customer_id, self.account_type, self.action, self.default_value,
                            self.days_count).deposit()
        elif self.action.lower() == 'withdraw':
            BusinessAccount(self.customer_id, self.account_type, self.action, self.default_value,
                            self.days_count).withdraw()
        elif self.action.lower() == 'back':
            self.back()
        elif self.action.lower() == 'quit':
            self.quit()
        elif self.action.lower() != 'deposit' or self.action.lower() != 'withdraw' or self.action != 'back' or self.action != 'quit':
            print('Please enter a valid action!')
            self.ask_user_for_account_action()
    else:
        print('Please choose a valid account!')
```

Εικόνα 42:Η συνάρτηση Ask User For Action Logic 2.

5.4.10 Deposit

Η λειτουργεία deposit είναι μια από τις βασικές λειτουργίες του κώδικα όπου πραγματοποιείται η κατάθεση κεφαλαίου στον αντίστοιχο λογαριασμό

```python
def deposit(self):
    self.get_amount_input()
    old_subclass_account_balance = 0
    if self.amount == 0:
        self.print_with_dots()
    else:
        print('Deposit Accepted')
        old_subclass_account_balance = database[self.customer_id]['accounts'].get(self.account_type, se
        new_subclass_account_balance = old_subclass_account_balance + self.amount
        database[self.customer_id]['accounts'][self.account_type] = new_subclass_account_balance
        print(f'Your Current Balance on {self.account_type} is ' + "%.2f" % new_subclass_account_balance
        self.update_total_balance()
```

Εικόνα 43: Η συνάρτηση Deposit (Parent Class).

5.4.11 Withdrawal

Η λειτουργεία withdrawal είναι η δεύτερη από τις ποιο βασικές λειτουργίες του κώδικα η οποία επιτρέπει την ανάληψη κεφαλαίου στον αντίστοιχο λογαριασμό.

```python
def withdraw(self):
    self.get_amount_input()
    account_balance = database[self.customer_id]['accounts'].get(self.account_type, 0)
    if self.amount == 0:
        self.print_with_dots()
    elif self.amount <= account_balance:
        new_balance = account_balance - self.amount
        database[self.customer_id]['accounts'][self.account_type] = new_balance
        print('Withdraw Accepted')
        self.update_total_balance()
        print('Your total balance is ' + "%.2f" % database[self.customer_id]['Total Balance'] + '€')
        print(f'Your {self.account_type} balance is ' + "%.2f" % new_balance + '€')
    else:
        print('Insufficient funds. Unable to withdraw.')
```

Εικόνα 44: Η συνάρτηση Withdraw (Parent Class)

5.4.12 Returning

Αλληλοεπιδρά με τον χρήστη και τον ρωτάει αν θέλει να κάνει κάποια άλλη ενέργεια. Η συγκεκριμένη συνάρτηση λαμβάνει χώρα όταν ο χρήστης μεταφερθεί σε κάποιο από τα subclass account ρωτώντας

τον αν θέλει να κάνει κατάθεση η ανάληψη καθώς και μετά από κάθε συναλλαγή σε περίπτωση που θέλει να πραγματοποιήσει κάποια ακόμα.

```
def returning(self):
    decision = ''
    while decision.lower() != 'yes' and decision.lower() != 'no':
           decision = input("Do you want to do another transaction? Yes/No:")
    else:
        if decision.lower() == 'yes':
            self.account_type = ''
            self.account_type_num = 0
            self.choosingaccount()
        elif decision.lower() == 'no':
            self.quit()
```

Εικόνα 45: Η συνάρτηση Returning.

5.4.13 Update total balance

Η συγκεκριμένη συνάρτηση χρησιμοποιείται για την ενημέρωση το συνολικού υπολοίπου του χρήστη και καλείται μετά από κάθε παραμετροποίηση.

```
def update_total_balance(self):
    total_balance = 0
    for account_type, balance in database[self.customer_id]['accounts'].items():
        total_balance += balance
        database[self.customer_id]['Total Balance'] = total_balance
```

Εικόνα 46: Η συνάρτηση update total balance.

5.4.14 Print with dots

Μια συνάρτηση που βελτιώνει την εμπειρία του χρήστη, δείχνοντάς του ότι το αίτημά του είναι υπό διεργασία.

```python
def print_with_dots(message='Retuning',num_dots = 3):
    print('Loading', end='', flush=True)
    for _ in range(num_dots):
        time.sleep(0.4)
        print('.', end='', flush=True)
    print()
```

Εικόνα 47: Η συνάρτηση Print with dots.

Έχει ως σκοπό την ποιο φιλική προς τον χρήστη αναπαράσταση της επιλογής 0 η αλλιώς «returning».

```
Or '0' to go back! 0
Loading...
```

Εικόνα 48: Το αποτέλεσμα της συνάρτησης Returning.

5.4.15 Get amount Input

Είναι άλλη μια από τις ποιο χρήσιμες συνάρτησής. Μέσω της συγκεκριμένης ενέργειας, όταν ο χρήστης αλληλεπιδρά με το πρόγραμμα για το deposit και withdrawal, τα funds περνάνε μια εξέταση αν καλύπτουν τις κατάλληλες προδιάγραφες. Δηλαδή αν είναι αριθμοί (integer) και όχι κάποιου αλλού ιδίους data.

```python
def get_amount_input(self):
    while True:
        try:
            self.amount = int(input("Please enter a valid amount:\nOr '0' to go back! "))
            if self.amount < 0:
                print("Amount needs to be a positive integer.\nOr '0' to go back!")
            else:
                break
        except ValueError:
            print('Invalid input. Please enter an integer.')
```

Εικόνα 49: Η συνάρτηση Get Amount Input.

```
def deposit(self):
    self.get_amount_input()
    old_subclass_account_balance = 0
    if self.amount == 0:
        self.print_with_dots()
    else:
        print('Deposit Accepted')
        old_subclass_account_balance = database[self.customer_id]['accounts'].get(self.account_type, se
        new_subclass_account_balance = old_subclass_account_balance + self.amount
        database[self.customer_id]['accounts'][self.account_type] = new_subclass_account_balance
        print(f'Your Current Balance on {self.account_type} is ' + "%.2f" % new_subclass_account_balanc
        self.update_total_balance()

def withdraw(self):
    self.get_amount_input()
    account_balance = database[self.customer_id]['accounts'].get(self.account_type, 0)
    if self.amount == 0:
        self.print_with_dots()
    elif self.amount <= account_balance:
        new_balance = account_balance - self.amount
        database[self.customer_id]['accounts'][self.account_type] = new_balance
        print('Withdraw Accepted')
        self.update_total_balance()
        print('Your total balance is ' + "%.2f" % database[self.customer_id]['Total Balance'] + '€')
        print(f'Your {self.account_type} balance is ' + "%.2f" % new_balance + '€')
    else:
        print('Insufficient funds. Unable to withdraw.')
```

Εικόνα 50: Acceptance Criteria.

5.4.16 Csv to nested dict

Η συγκεκριμένη συνάρτηση καλείται κατά τη διαδικασία του Login, καθώς διαβάζει το csv αρχείο και μέσω μιας for αντιστοιχεί τα στοιχεία του csv με τα καταλληλά του database – dictionary.

```python
import csv
import ast

def csv_to_nested_dict(csv_file):
    database = {}
    with open(csv_file, 'r', newline='') as file:
        reader = csv.DictReader(file)
        for row in reader:
            tax_id = int(row['tax_id'])
            total_balance = int(row['Total Balance'])

            accounts_str = row['accounts']
            accounts_dict = ast.literal_eval(accounts_str)

            accounts_int = {account: int(amount) for account, amount in accounts_dict.items()}

            uuid = row['uuid']
            database[uuid] = {
                'name': row['name'].title(),
                'tax_id': tax_id,
                'Total Balance': total_balance,
                'accounts': accounts_int
            }
    return database
```

Εικόνα 51: Η συνάρτηση CSV To από Dict.

```
database

{}
```

Εικόνα 52: Άδειο Database από database = {}

```
database

{'bae07227-7f3e-4753-8740-4664f6e7f578': {'name': 'Panos',
  'tax_id': 1234,
  'Total Balance': 10000,
  'accounts': {'Checking Account': 1000,
   'Savings Account': 2000,
   'DepositGuard Account': 3000,
   'Business Account': 4000}},
 '7cbd1729-050f-4c1f-acf4-cf3cecf0eda1': {'name': 'Nikos',
  'tax_id': 12345,
  'Total Balance': 11110,
  'accounts': {'Checking Account': 1111,
   'Savings Account': 2222,
   'DepositGuard Account': 3333,
   'Business Account': 4444}}}
```

Εικόνα 53: Η συνάρτηση CSV To Nested Dict γεμίζει το παραπάνω άδειο database

5.4.17 CheckingAccount

CheckingAccount (Λογαριασμός Όψεως): Υποκλάση της κλάσης Account που αντιπροσωπεύει έναν τραπεζικό λογαριασμό όψεως. Περιλαμβάνει επιπλέον λειτουργίες που είναι συγκεκριμένες για αυτόν τον τύπο λογαριασμού. Προσμοιάζει έναν απλό καθημερινό λογαριασμό τραπέζης ο οποίος χρησιμοποιείται για τις καθημερινές συναλλαγές του χρήστη.

```
class CheckingAccount(Account):
    def __init__(self, customer_id, account_type, action, default_value):
        super().__init__()
        self.customer_id = customer_id
        self.account_type = account_type
        self.action = action
        self.default_value = default_value
        self.subclass_account_balance = database[self.customer_id]['accounts'].get(self.account_type,
                                                                                    self.default_value)

    1 usage
    def deposit(self):
        super().deposit()
        self.action = ''
        super().returning()

    1 usage
    def withdraw(self):
        super().withdraw()
        super().update_total_balance()
        self.action = ''
        super().returning()
```

Εικόνα 54: Checking Account(Account) Subclass

5.4.18 SavingsAccount

SavingsAccount (Λογαριασμός Ταμιευτηρίο): Υποκλάση της κλάσης Account που αντιπροσωπεύει έναν τραπεζικό λογαριασμό αποταμίευσης. Περιλαμβάνει επιπλέον λειτουργίες και χαρακτηριστική του λειτουργεία είναι ο φραγμός των αναλήψεων από τρεις φόρες και πάνω.

```
def withdraw(self):
    if datetime.now().day == 1 and (self.last_withdraw_date is None or self.last_withdraw_date.month !=
        self.withdrawls_times = 0
    elif self.withdrawls_times < 3:
        self.last_withdraw_date = datetime.now()
        super().withdraw()
        if self.amount !=0:
            self.withdrawls_times += 1
            print (f' This is your: {self.withdrawls_times} withdrawl!')
            super().update_total_balance()
            self.action = ""
            super().returning()
        else:
            super().update_total_balance()
            self.action = ""
            super().returning()
    else:
        print('Withdrawal limit reached. Withdrawal functionality is locked.')
        self. returning()
```

Εικόνα 55 Subclass Savings Accounts Withdraw Function

Στο συγκεκριμένο κομμάτι κώδικα ανακαλείτε η super().withdraw() δηλαδή κληρονομεί την συνάρτηση της γωνιακής κλάσης withdraw χωρίς να χριάζεται η επανάληψη κώδικα. Επίσης, στο συγκεκριμένο σημείο καταχωρείται και ένας μετρητής self.withdrawls_times ο οποίος είναι υπεύθυνος για τον φραγμό των συναλλαγών. Τέλος, μέσω της last_withdraw_date, κάθε πρώτη του μήνα επιτυγχάνετε μια επανεκκίνηση στην μεταβλητή withdrawl_times.

```
def calculate_the_presentage_of_last_day(self):
    value = database[self.customer_id]['accounts'][account_type]
    last_day_value =value + value*10/100
    last_day_value = database[self.customer_id]['accounts'][account_type]
```

Εικόνα 56: Extra function only for Subclass Savings Account.

Ο λογαριασμός SavingsAccount έχει επιτόκιο 10%. Μια παραπάνω λειτουργεία που παρατηρείται μόνο στην συγκεκριμένη υποκλάση, προστατεύοντας το κεφάλαιο από τις επιδράσεις του πληθωρισμού,

5.4.19 DepositGuardAccount

DepositGuardAccount (Λογαριασμός Προστασίας Καταθέσεων): Υποκλάση της κλάσης Account που αντιπροσωπεύει έναν τραπεζικό λογαριασμό προστασίας καταθέσεων. Παρέχει λειτουργίες για αποθήκευση και προστασία των καταθέσεων του χρήστη καθώς απαγορεύει την οποιαδήποτε προσπάθεια για ανάληψη. Ο μονός τρόπος ανάληψης είναι από κάποιο κατάστημα ακόμα μεγαλύτερη προστασία κεφαλαίων.

```
if self.account_type_num == '3':
  self.account_type = ('DepositGuard Account')
  print('DepositGuard Account')
  self.action = ''
  self.action = input('DepositGuard Account only allows deposits, not withdrawals.\nDo you want to continue?
  if self.action.lower() == 'yes':
      DepositGuardAccount(self.customer_id,self.account_type,self.action,self.default_value).deposit()
  elif self.action == 'no':
      self.back()
```

Εικόνα 57:Acceptance criteria for DepositGuardAccount. Valid Actions is only Deposit.

6 Συμπεράσματα και πεδία μελλοντικής έρευνας

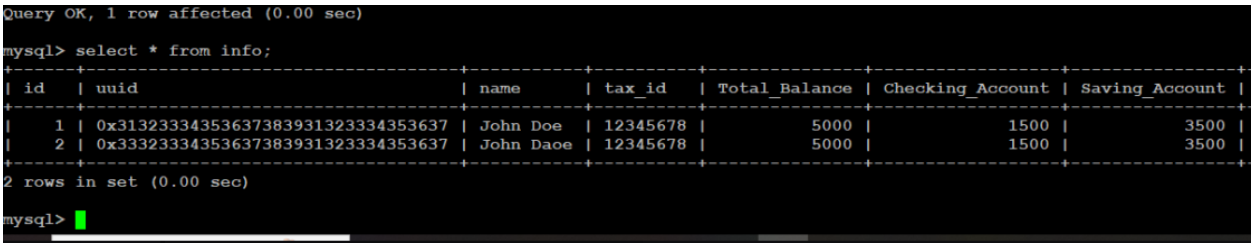
Μέσω του συγκεκριμένου έργου γίνονται αντιληπτά στον αναγνώστη τι σημαίνουν κάποιες από τις θεμελιώδεις αρχές του προγραμματισμού: αντικειμενοστράφια, κληρονομικότητα και πολυμορφισμός. Ακόμα γίνετε φανερή η ολοκληρωμένη δομή ενός προγράμματος, η δομή της βασικής γονικής κλάσης καθώς και των τριών υποκλάσεων της μεταξύ τους σχέσης. Επίσης, γίνεται αντιληπτό το πως διαφοροποιούνται τα δεδομένα και το τι χρήση έχει ο κάθε τύπος δεδομένων στον εκάστοτε κώδικα. Τέλος με την δημιουργία λογισμικού έχει την δυνατότητα ο αναγνώστης να δει όλα τα παραπάνω σε πράξη, να αντιληφθεί την ροή της πληροφορίας μέσα στον κώδικα, καθώς και το πως η μία συνάρτηση συνδέεται με την άλλη, αλλά και το τελικό αποτέλεσμά: τη λογική της δημιουργίας του προγράμματος.

6.1.1 Πιθανές Προσθήκες και βελτιστοποιήσεις

Κατά την διάρκεια ανάπτυξης της εφαρμογής, προέκυψαν και επιπλέον ιδέες για την προσθήκη περαιτέρω λειτουργιών:

- Αρχιτεκτονική: Η αρχιτεκτονική στον κώδικα αποτελεί ξεχωριστό κεφάλαιο. Παρατηρείται κατά κύριο λόγο σε εταιρίες και αφορά τους προγραμματιστές και τον τρόπο εγγραφής κώδικα. Έχει ως σκοπό ο κώδικας να γράφεται πανομοιότυπα ώστε να είναι ευανάγνωστος από όλους τους προγραμματιστές ανεξαρτήτως του αρχικού συγγραφέα [23]. Στο συγκεκριμένο σημείο και στην συγκεκριμένη εφαρμογή ωστόσο, μέσω της αρχιτεκτονικής θα μπορούσε να βελτιστοποιηθεί η δομή του κώδικα, δίνοντας τη δυνατότητα για καλύτερη διαχείριση, συντήρηση αλλά και την καλύτερη επίδοσή του.
- Βάση Δεδομένων (Database): Στη συγκεκριμένη εργασία χρησιμοποιείτε csv αρχείο με σκοπό την προσομοίωση μιας βάσης δεδομένων το οποίο καλύπτει πλήρως κάθε απότιση για την σωστή λειτουργεία του προγράμματος [24]. Ωστόσο, θα μπορούσε να μεταφερθεί σε μία διαφορετική

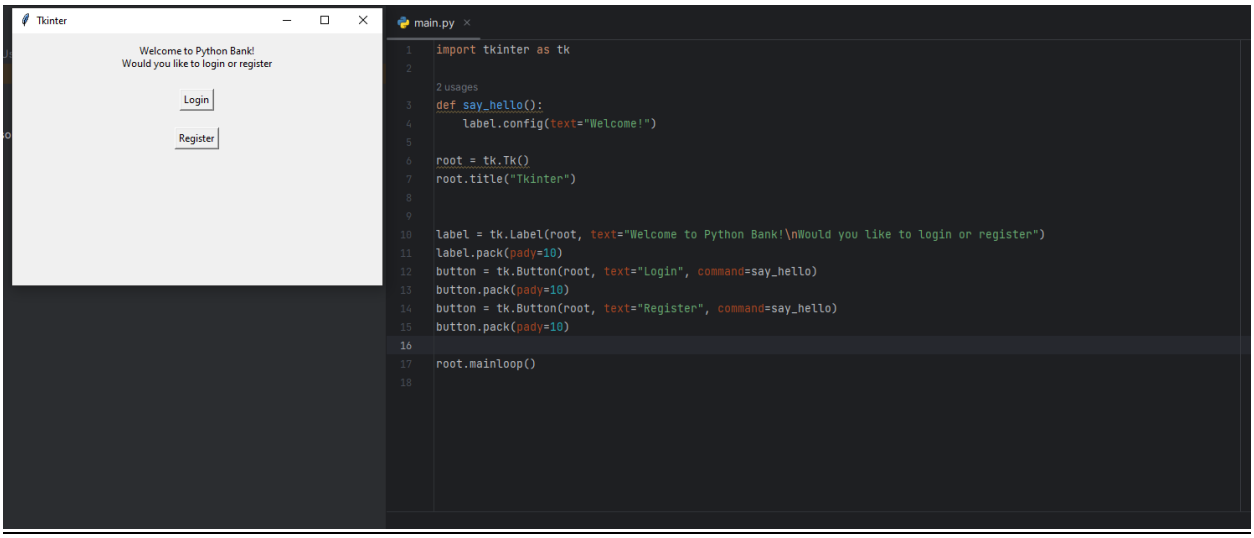
βάση δεδομένων όπου ένας διαχειριστής με γνώσεις MySQL θα έκανε το πρόγραμμα να προσήκει σε συστήματα σύγχρονης τραπεζικής.



Εικόνα 58: Υποτυπώδες Database για την παραπάνω εφαρμογή.

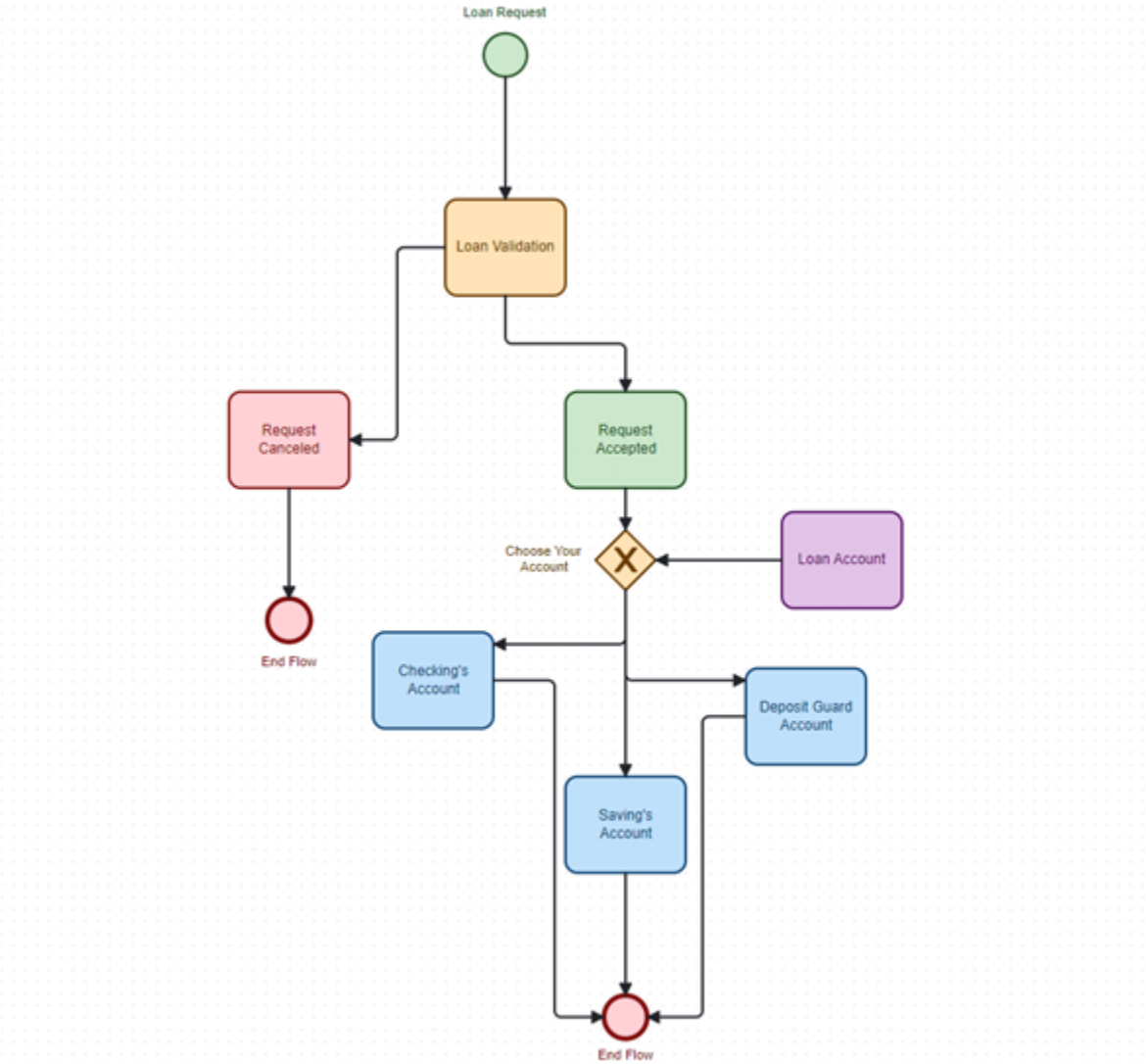
- Εταιρικός Λογαριασμός (Business Account): Ένας επιπλέον λογαριασμός που θα μπορούσε να προστεθεί στον κώδικα είναι ο εταιρικός. Τα χρηματοπιστωτικά ιδρύματα υποχρεούνται νομικά να διαθέτουν διαφορετικούς λογαριασμούς για μη φυσικά πρόσωπα. Αυτοί οι λογαριασμοί δίνουν σε εταιρίες την δυνατότητα συναλλαγών με το κοινό, επεξεργασίας πληρωμών καθώς και ενδεχόμενες πιστωτικές υπηρεσίες και δάνεια από την τράπεζα. Παράλληλα, ένας εταιρικός λογαριασμός προσφέρει πρόσβαση διαφορετικών επίπεδων σε υπαλλήλους οι οποίοι μπορούν να πραγματοποιούν συναλλαγές εκ μέρους της επιχείρησης. Η συγκείμενη λειτουργία θα μπορούσε να υλοποιηθεί εφαρμόζοντας στους εταιρικούς λογαριασμούς ευνοϊκότερα επιτόκιο για μελλοντικά δάνεια.
- Δάνεια: Ακόμα θα μπορούσε να προστεθεί ένας λογαριασμός για δάνεια (Loan_Account) ο οποίος θα καταχωρεί το ζητούμενο κεφάλαιο στον λογαριασμό ταμιευτήριου με επιτόκιο λόγου χάρη 1% του τρέχοντος ποσού στο τέλος κάθε μηνά.
- User Interface (UI): Το περιβάλλον χρήστη στην σημερινή εποχή είναι ένα κομβικής σημασίας. Οι χρήστες πλέον παρατηρείτε ότι διαθέτουν όλο και λιγότερο χρόνο κατά την περιήγηση τους σε μια εφαρμογή. Αυτό κάνει το συγκεκριμένο κομμάτι πολύ ενδιαφέρον καθώς πέρα από δημιουργικό θεωρείται και μεγάλη πρόκληση. Ωστόσο, κάτι τέτοιο δεν παρέχεται στη

συγκεκριμένη εφαρμογή καθώς είναι εκτός των ζητουμένων της εργασίας αλλά είναι μια ιδέα που θα έχει ενδιαφέρον να υλοποιηθεί στο μέλλον [25].



Εικόνα 59: User Interface

- Flow Chart: Τα flows είναι ένας τρόπος απεικόνισης μιας διαδικασίας που απαρτίζεται συνήθως από αρκετές και κατά κύριο λόγο ξεχωριστές υπό-εργασίες [26]. Τα flow charts επίσης είναι ένας από τους τρόπους με τον οποίον θα μπορούσαν όλες οι λειτουργίες να συνεργαστούν μεταξύ τους πετυχαίνοντας το ζητούμενο αποτέλεσμα. Στο συγκεκριμένο έργο τα flow charts θα μπορούσαν να συνδυαστούν με τα δάνεια καθώς μέσω του πάρα πάνω η αίτηση θα μπορούσε να περάσει της επικύρωση ή όχι του δανείου και στην συνέχεια μέσω του κατάλληλου task να καταχωρηθεί στον λογαριασμό ταμιευτηρίου



Εικόνα 60 Flow for Loan

7 Βιβλιογραφία

[1] Python Software Foundation, «Built-in Functions,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/functions.html. [Πρόσβαση 21 06 2024].

[2] Python Software Foundation., «Data Types,» Python, 21 01 2024b. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/datatypes.html. [Πρόσβαση 21 06 2024].

[3] G. v. Rossum, «What's New In Python 3.0,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/whatsnew/3.0.html#integers. [Πρόσβαση 21 06 2024].

[4] Python Software Foundation., «string — Common string operations,» 21 01 2024c. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/string.html. [Πρόσβαση 21 06 2024].

[5] Python Land, «Python Float: Working With Floating-Point Numbers,» 09 01 2024. [Ηλεκτρονικό]. Available: https://python.land/python-data-types/python-float#google_vignette. [Πρόσβαση 21 06 2024].

[6] Python Software Foundation., «Data Structures,» 21 01 2024d. [Ηλεκτρονικό]. Available: https://docs.python.org/3/tutorial/datastructures.html. [Πρόσβαση 21 06 2024].

[7] L. P. Ramos, «Python's tuple Data Type: A Deep Dive With Examples,» [Ηλεκτρονικό]. Available: https://realpython.com/python-tuple/.

[8] S. John, «Dictionaries in Python,» Real Python, 06 08 2018. [Ηλεκτρονικό]. Available: https://realpython.com/python-dicts/. [Πρόσβαση 21 06 2024].

[9] Python Software Foundation, «More Control Flow Tools,» 21 01 2024d. [Ηλεκτρονικό]. Available: https://docs.python.org/3/tutorial/controlflow.html. [Πρόσβαση 21 06 2024].

[10] Python Software Foundation, «Built-in Types,» 21 01 2024d. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/stdtypes.html. [Πρόσβαση 21 06 2024].

[11] GeeksforGeeks, «Loops in Python – For, While and Nested Loops,» 20 01 2024. [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/loops-in-python/. [Πρόσβαση 21 06 2024].

[12] Python Software Foundation, «Built-in Exceptions,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/exceptions.html. [Πρόσβαση 21 06 2024].

[13] Z. Bartosz, «Your Guide to the Python print() Function,» 12 8 2019. [Ηλεκτρονικό]. Available: https://realpython.com/python-print/. [Πρόσβαση 21 06 2024].

[14] Python Software Foundation, «Simple statements,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/reference/simple_stmts.html#grammar-token-return_stmt. [Πρόσβαση 21 06 2024].

[15] Python Software Foundation, «Input and Output,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/tutorial/inputoutput.html. [Πρόσβαση 21 06 2024].

[16] Python Software Foundation, «The Python Standard Library,» 21 01 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/index.html. [Πρόσβαση 21 06 2024].

[17] Python Software Foundation, «uuid — UUID objects according to RFC 4122,» 6 7 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/uuid.html. [Πρόσβαση 6 7 2024].

[18] Python Software Foundation., «getpass — Portable password input,» 05 07 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/getpass.html. [Πρόσβαση 06 07 2024].

[19] Python Software Foundation., «sys — System-specific parameters and functions,» 05 07 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/sys.html. [Πρόσβαση 06 07 2024].

[20] Python Software Foundation., «datetime — Basic date and time types,» 05 07 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/datetime.html. [Πρόσβαση 06 07 2024].

[21] Python Software Foundation, «csv — CSV File Reading and Writing,» 06 07 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/csv.html. [Πρόσβαση 06 07 2024].

[22] Python Software Foundation., «ast — Abstract Syntax Trees,» 05 07 2024. [Ηλεκτρονικό]. Available: https://docs.python.org/3/library/ast.html. [Πρόσβαση 06 07 2024].

[23] F. Martin, «Software Architecture Guide,» 1 08 2019. [Ηλεκτρονικό]. Available: https://martinfowler.com/architecture/. [Πρόσβαση 21 06 2024].

[24] MySQL, «MySQL,» 2024. [Ηλεκτρονικό]. Available: https://www.mysql.com/. [Πρόσβαση 21 06 2024].

[25] App Architecture, «user interface (UI),» 2024. [Ηλεκτρονικό]. Available: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI. [Πρόσβαση 21 06 2024].

[26] smartdraw, With SmartDraw, You Can Create Many Different Types of Diagrams, Charts, and Visuals, 2024. [Ηλεκτρονικό]. Available: https://www.smartdraw.com/flowchart/. [Πρόσβαση 21 06 2024].