



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Γραφικών Διεπαφών Με Χρήση Των Βιβλιοθηκών Tkinter Και PyQt

ΡΑΙΚΟΣ ΝΙΚΗΤΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Δαδαλιάρης Αντώνιος

Επίκουρος Καθηγητής

Λαμία 15 Μαρτίου 2024

1



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΝΑΠΤΥΞΗ ΓΡΑΦΙΚΩΝ ΔΙΕΠΑΦΩΝ ΜΕ ΧΡΗΣΗ ΤΩΝ
ΒΙΒΛΙΟΘΗΚΩΝ TKINTER ΚΑΙ PYQT

ΡΑΙΚΟΣ ΝΙΚΗΤΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΔΑΔΑΛΙΑΡΗΣ ΑΝΤΩΝΙΟΣ

ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ



UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

DEVELOPMENT OF GRAPHICAL INTERFACES USING THE TKINTER AND PYQT LIBRARIES

RAIKOS NIKITAS

FINAL THESIS

ADVISOR

DADALIARIS ADONIOS

PROFESSOR

Lamia 15/03/2024

5

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις ⁽¹⁾, που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάστηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.
2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.
3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια
4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 15/03/2024

Ο Δηλών
Ράικος Νικήτας

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας εργασίας αποτελεί η γνωριμία των βιβλιοθηκών **Tkinter** και **PyQt** της γλώσσας προγραμματισμού **Python** στην ανάπτυξη γραφικών διεπαφών(**GUI**). Αρχικά, υπάρχει μια γενική αναφορά της γλώσσας προγραμματισμού Python. Στη συνέχεια, παρουσιάζεται η ανάλυση της βιβλιοθήκης Tkinter και των εργαλείων που διαθέτει με σκοπό την ανάπτυξη γραφικών διεπαφών. Έπειτα, ακολουθεί η γνωριμία της βιβλιοθήκης PyQt και των εργαλείων που κατέχει καθώς και πως τα χρησιμοποιεί. Ακολουθεί μια σύγκριση των δύο βιβλιοθηκών στο πως και στο ποια εργαλεία χρησιμοποιεί η κάθε μία για την επίλυση συγκεκριμένων προβλημάτων.

ABSTRACT

The purpose of this paper is to introduce the Tkinter and PyQt libraries of the Python programming language in the development of graphical user interfaces (GUI). Initially, there is a general overview of the Python programming language. Then, an analysis of the Tkinter library and the tools it has for the purpose of GUI development is presented. Next, there is an introduction to the PyQt library and the tools it holds and how it uses them. This is followed by a comparison of the two libraries in how and which tools each uses to solve specific problems.

Πίνακας περιεχομένων

ΠΕΡΙΛΗΨΗ	8
ABSTRACT	10
ΚΕΦΑΛΑΙΟ 1 PYTHON	13
ΚΕΦΑΛΑΙΟ 2 TKINTER	23
WIDGETS:.....	23
Widget Label	24
Widget Button.....	24
Widget Entry	25
Widget Text.....	26
Widget Frame	27
Widget Checkbutton	28
Widget Radiobutton.....	29
Widget Listbox	29
Widget Scale	30
Widget Canvas	31
MODULES:.....	32
Tkinter.messagebox	32
Tkinter.filedialog	33
Tkinter.colorchooser	34
Tkinter.ttk	34
Tkinter.scrolledtext	35
Tkinter.simpledialog	36
LAYOUT MANAGEMENT AND GEOMETRY MANAGERS:	36
Layout management:	36
Geometry Manager:.....	37
ΚΕΦΑΛΑΙΟ 3 PYQT	41
WIDGETS:.....	41
Widget QPushButton	41
Widget QLabel	42
Widget QLineEdit	43

Widget QTextEdit.....	44
Widget QComboBox.....	45
Widget QRadioButton	47
Widget QSlider	48
Widget QProgressBar	49
Widget QTreeWidget	50
Widget QTableWidget.....	51
Widget QTabWidget.....	52
Widget QDialog.....	53
MODULES:	54
QtCore.....	54
QtGui.....	56
QtMultimedia	57
QtNetwork	58
QtSql	59
QtTest	60
ΚΕΦΑΛΑΙΟ 4 ΣΥΓΚΡΙΣΗ TKINTER ΚΑΙ PYQT	62
1. Ευκολία χρήσης και εκμάθησης.....	62
2. Χαρακτηριστικά και δυνατότητες	62
3. Εμφάνιση	63
4. Επιδόσεις	63
5. Κοινότητα και υποστήριξη	64
6. Αδειοδότηση.....	64
ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑ	64
ΒΙΒΛΙΟΓΡΑΦΙΑ	66

ΚΕΦΑΛΑΙΟ 1 PYTHON

Η **Python** αποτελεί μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού, ξεχωρίζει για τα μοναδικά χαρακτηριστικά και τη φιλοσοφία σχεδιασμού της, που την καθιστούν ευρέως εφαρμόσιμη σε διάφορους τομείς. Η **Python** είναι γλώσσα υψηλού επιπέδου, καθώς έχει σχεδιαστεί να είναι εύκολη στην ανάγνωση και τη συγγραφή. Η σύνταξή της είναι σαφής και εκφραστική, γεγονός που καθιστά τον κώδικα πιο κατανοητό και άμεσο. Επίσης, το συντακτικό της έχει σχεδιαστεί ώστε να είναι σαφές και ευανάγνωστο, γεγονός που την καθιστά εξαιρετική επιλογή τόσο για αρχάριους όσο και για έμπειρους προγραμματιστές. Η **Python** χρησιμοποιεί την εσοχή για να ορίσει τμήματα κώδικα. Η συνεπής εσοχή είναι πολλή σημαντική για τη σωστή εκτέλεση του κώδικα. Το κενό διάστημα (κενά και **tabs**) είναι σημαντικό στην **Python** χρησιμοποιείται για την εσοχή και είναι απαραίτητο για την αναγνωσιμότητα του κώδικα.

Σε αντίθεση με τις μεταγλωττισμένες γλώσσες, ο κώδικας της **Python** εκτελείται γραμμή προς γραμμή, γεγονός που διευκολύνει την εύρεση σφάλματος, αλλά μπορεί να

οδηγήσει σε πιο αργούς χρόνους εκτέλεσης για ορισμένους τύπους εφαρμογών. Οι πιο γνωστές εφαρμογές που συμβαίνει αυτό είναι αριθμητικοί και επιστημονικοί υπολογισμοί, επεξεργασία εικόνων, δημιουργία παιχνιδιών και γενικά ότι έχει να κάνει με εφαρμογές που υλοποιούνται σε πραγματικό χρόνο.

Η **Python** είναι δυναμικά τυποποιημένη, πράγμα που σημαίνει ότι δεν απαιτείται η ρητή δήλωση των τύπων των μεταβλητών. Ο τύπος συνάγεται κατά την εκτέλεση, γεγονός που καθιστά τον κώδικα πιο ευέλικτο, αλλά απαιτεί επίσης προσεκτικό έλεγχο για την αποφυγή σφαλμάτων που σχετίζονται με τον τύπο. Επιπλέον, η **Python** υποστηρίζει διάφορα παραδείγματα προγραμματισμού, συμπεριλαμβανομένων του διαδικαστικού, του αντικειμενοστραφούς και του λειτουργικού προγραμματισμού, επιτρέποντας ευελιξία στον τρόπο προσέγγισης και επίλυσης των προβλημάτων. Πιο συγκεκριμένα, η **python** υποστηρίζει με τον αντικειμενοστραφή προγραμματισμό(**OOP**) κλάσεις και αντικείμενα, επιτρέποντας την ενθυλάκωση, την κληρονομικότητα και τον πολυμορφισμό. Αυτό καθιστά την **Python** κατάλληλη για το σχεδιασμό πολύπλοκων συστημάτων λογισμικού. Η **Python** υποστηρίζει παραδείγματα λειτουργικού προγραμματισμού, συμπεριλαμβανομένων χαρακτηριστικών όπως οι συναρτήσεις λάμδα, οι συναρτήσεις χάρτη και φίλτρου και η κατανόηση λίστας. Τέλος, ο διαδικαστικός προγραμματισμός σε συνδυασμό με την **python** είναι ένα παραδοσιακό στυλ προγραμματισμού που επικεντρώνεται στη χρήση συναρτήσεων ως βασικών στοιχείων και στην αποφυγή των μεταβλητών και των αλλαγών κατάστασης.

Η τυπική βιβλιοθήκη της **Python** είναι εκτεταμένη, καλύπτοντας ένα ευρύ φάσμα προγραμματιστικών αναγκών, από υπηρεσίες ιστού έως τη δημιουργία διεπαφής χρήστη, από το χειρισμό αρχείων έως τη σειριοποίηση δεδομένων. Η **Python** διαθέτει ένα τεράστιο σύνολο τυποποιημένων βιβλιοθηκών πακέτων και πλαισίων τρίτων κατασκευαστών για την ανάπτυξη ιστοσελίδων (**Django**, **Flask**), την ανάλυση δεδομένων (**Pandas**, **NumPy**) και τη δημιουργία γραφικών διεπαφών χρήστη (**Tkinter**, **Pyqt**) που θα αναλυθεί παρακάτω.

Επίσης, η **Python** εξειδικεύεται σε εφαρμογές όπως είναι η ανάπτυξη ιστοσελίδων, η επιστήμη δεδομένων και μηχανική μάθηση, εκπαίδευση και οι εφαρμογές γραφικών διεπαφών.

Τέλος, η γλώσσα προγραμματισμού **Python** μπορεί να γραφτεί με διάφορους τρόπους. Πιο συγκεκριμένα:

1. **Διερμηνευτής Python:** Ο διερμηνέας της **Python** είναι ένα πρόγραμμα που διαβάζει και εκτελεί κώδικα **Python**. Είναι το βασικό συστατικό της γλώσσας προγραμματισμού **Python** και είναι υπεύθυνο για την ερμηνεία του κώδικα γραμμή προς γραμμή και την εκτέλεσή του. Η **Python** είναι μια διερμηνευόμενη γλώσσα, πράγμα που σημαίνει ότι ο πηγαίος κώδικας μεταφράζεται σε ενδιάμεσο κώδικα (**bytecode**) από τον διερμηνέα και εκτελείται άμεσα. Η **Python** είναι δυναμικά τυποποιημένη, που σημαίνει ότι δεν χρειάζεται να δηλώνετε ρητά τον τύπο δεδομένων των μεταβλητών. Ο διερμηνέας καθορίζει τον τύπο των μεταβλητών κατά τη διάρκεια της εκτέλεσης με βάση τις τιμές που τους έχουν ανατεθεί. Ο διερμηνέας **Python** μπορεί να χρησιμοποιηθεί διαδραστικά, επιτρέποντάς την

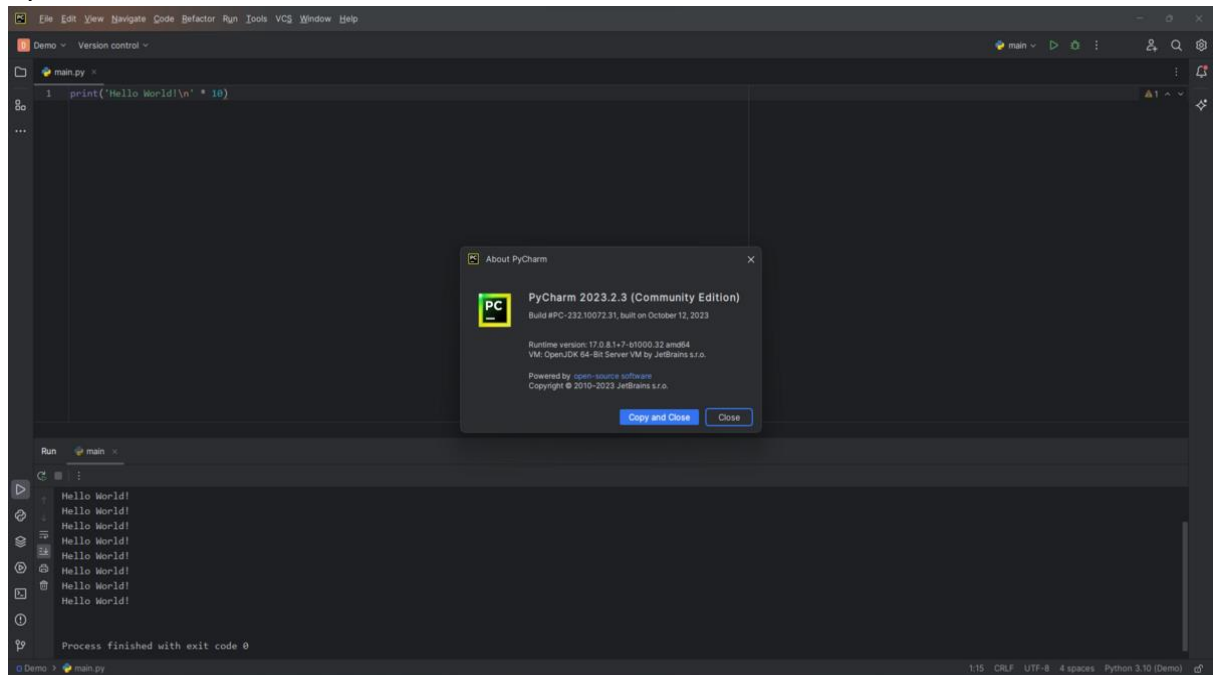
εισαγωγή εντολών **Python** και να προβληθούν άμεσα αποτελέσματα. Αυτό είναι χρήσιμο για γρήγορο πειραματισμό και δοκιμή αποσπασμάτων κώδικα. Η διαδραστική λειτουργία του διερμηνευτή **Python** ακολουθεί έναν βρόχο **REPL**, όπου διαβάζει την είσοδο από τον χρήστη, αξιολογεί την είσοδο ως κώδικα **Python**, εκτυπώνει τυχόν αποτελέσματα ή έξοδο και στη συνέχεια επιστρέφει στην αρχή για να περιμένει περισσότερη είσοδο. Εκτός από τη διαδραστική λειτουργία, ο διερμηνέας **Python** χρησιμοποιείται συνήθως για την εκτέλεση σεναρίων **Python** που είναι αποθηκευμένα σε αρχεία με επέκταση **.py**. Ο διερμηνέας **Python** συνοδεύεται από μια ολοκληρωμένη τυπική βιβλιοθήκη, η οποία περιλαμβάνει ενότητες και πακέτα που παρέχουν λειτουργίες για εργασίες όπως είσοδος/έξοδος αρχείων, δικτύωση, κανονικές εκφράσεις και πολλά άλλα. Αυτές οι ενότητες μπορούν να εισαχθούν και να χρησιμοποιηθούν στα προγράμματά σας **Python**. Δεδομένου ότι η **Python** είναι μια διερμηνευόμενη γλώσσα, ο κώδικας **Python** μπορεί να εκτελεστεί σε οποιαδήποτε πλατφόρμα για την οποία είναι διαθέσιμος διερμηνευτής **Python**. Αυτό καθιστά τα προγράμματα **Python** εξαιρετικά φορητά και διασφαλίζει ότι ο κώδικας που έχει γραφτεί σε μια πλατφόρμα μπορεί εύκολα να εκτελεστεί σε μια άλλη χωρίς τροποποίηση.

2. Επεξεργαστές κειμένου: Οι επεξεργαστές κειμένου είναι εφαρμογές λογισμικού που έχουν σχεδιαστεί για τη δημιουργία και την επεξεργασία απλών αρχείων κειμένου. Αποτελούν βασικά εργαλεία για τη συγγραφή κώδικα, συμπεριλαμβανομένης της **Python**. Οι επεξεργαστές κειμένου εργάζονται με αρχεία απλού κειμένου, τα οποία περιέχουν ακατέργαστο κείμενο χωρίς μορφοποίηση ή ειδικούς χαρακτήρες. Αυτή η απλότητα τους καθιστά ευέλικτους για τη συγγραφή κώδικα. Οι περισσότεροι σύγχρονοι επεξεργαστές κειμένου παρέχουν υπογράμμιση σύνταξης, μια λειτουργία που χρωματίζει διάφορα στοιχεία του κώδικα για να τον κάνει πιο ευανάγνωστο. Αυτό περιλαμβάνει την επισήμανση λέξεων-κλειδιών, συμβολοσειρών, σχολίων και άλλων συντακτικών στοιχείων, γεγονός που μπορεί να βοηθήσει τους προγραμματιστές να εντοπίσουν τα λάθη και να κατανοήσουν ευκολότερα τη δομή του κώδικα. Πολλοί επεξεργαστές κειμένου προσφέρουν λειτουργίες αυτόματης συμπλήρωσης ή πρότασης κώδικα, οι οποίες μπορούν να εξοικονομήσουν χρόνο και να μειώσουν τα λάθη πληκτρολόγησης, προτείνοντας συμπληρώσεις για ονόματα μεταβλητών, ονόματα συναρτήσεων και άλλα στοιχεία κώδικα. Οι επεξεργαστές κειμένου συχνά επιτρέπουν εκτεταμένη προσαρμογή μέσω πρόσθετων προγραμμάτων, επεκτάσεων ή ρυθμίσεων διαμόρφωσης. Αυτή η ευελιξία επιτρέπει στους προγραμματιστές να προσαρμόσουν τον επεξεργαστή στις συγκεκριμένες ανάγκες τους, προσθέτοντας λειτουργίες όπως ενσωμάτωση ελέγχου έκδοσης, διόρθωση κώδικα και εργαλεία διαχείρισης έργων. Οι επεξεργαστές κειμένου συνήθως ενσωματώνονται με συστήματα ελέγχου εκδόσεων όπως το **Git**, επιτρέποντας στους προγραμματιστές να διαχειρίζονται τις αλλαγές κώδικα, να παρακολουθούν τις αναθεωρήσεις και να συνεργάζονται με άλλους απευθείας μέσα από τον επεξεργαστή. Ενώ οι επεξεργαστές κειμένου παρέχουν βασικές λειτουργίες επεξεργασίας, ορισμένοι επεξεργαστές, ειδικά αυτοί που έχουν σχεδιαστεί για προγραμματισμό, όπως το **Visual Studio Code**, προσφέρουν

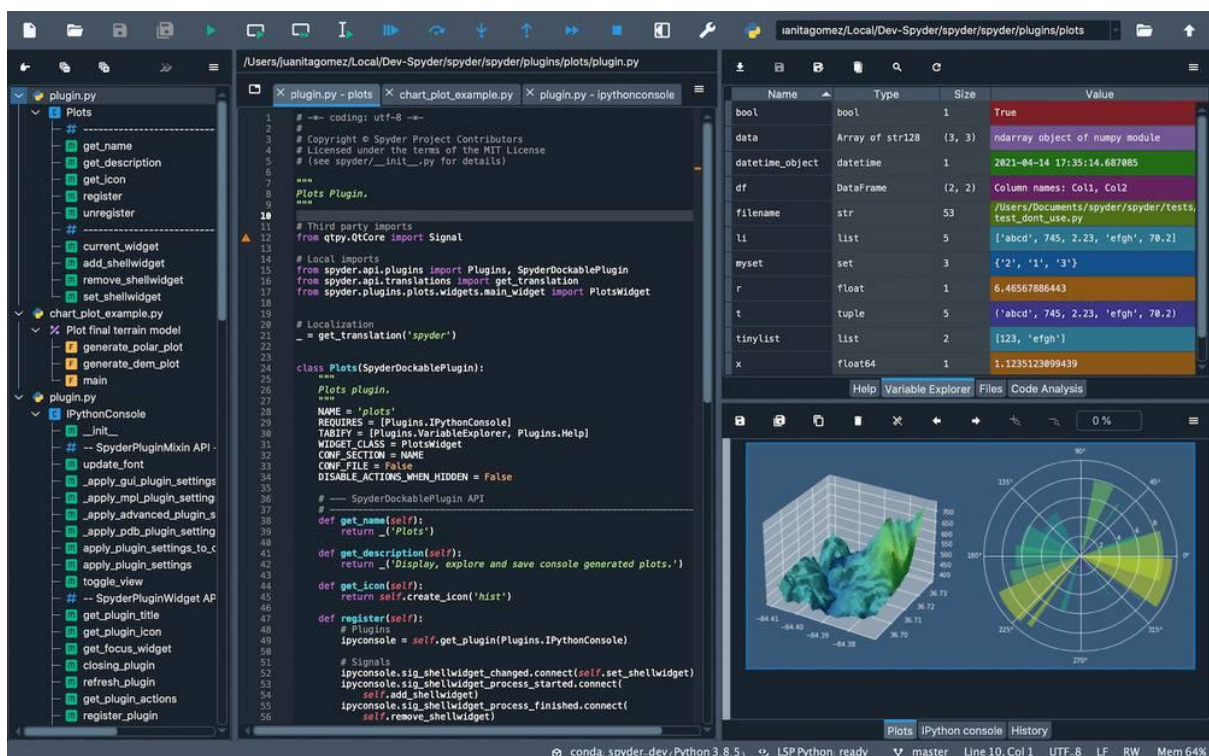
ολοκληρωμένες λειτουργίες που μοιάζουν με περιβάλλον ανάπτυξης (**IDE**), όπως αποσφαλμάτωση, πλοήγηση στον κώδικα και διαχείριση έργου. Δημοφιλείς επεξεργαστές κειμένου για τη συγγραφή κώδικα Python είναι οι **Sublime Text**, **Atom**, **Visual Studio Code** και **Notepad++**.

3. Ολοκληρωμένα περιβάλλοντα ανάπτυξης (**IDE**): Τα ολοκληρωμένα περιβάλλοντα ανάπτυξης (**IDE**) είναι ολοκληρωμένες εφαρμογές λογισμικού που παρέχουν στους προγραμματιστές ένα σύνολο εργαλείων και χαρακτηριστικών για τη διευκόλυνση ολόκληρης της διαδικασίας ανάπτυξης λογισμικού. Τα **IDEs** προσφέρουν προηγμένες δυνατότητες επεξεργασίας κώδικα, όπως επισήμανση συντακτικού, συμπλήρωση κώδικα και εργαλεία αναδιαμόρφωσης κώδικα. Αυτές οι δυνατότητες βοηθούν τους προγραμματιστές να γράφουν κώδικα πιο αποτελεσματικά και με μεγαλύτερη ακρίβεια. Τα **IDE** περιλαμβάνουν εργαλεία αποσφαλμάτωσης που επιτρέπουν στους προγραμματιστές να εντοπίζουν και να διορθώνουν τα σφάλματα στον κώδικά τους. Αυτό περιλαμβάνει χαρακτηριστικά όπως σημεία διακοπής, λίστες παρακολούθησης και ίχνη στοίβας που βοηθούν τους προγραμματιστές να παρακολουθούν τη ροή εκτέλεσης των προγραμμάτων τους. Τα **IDE** παρέχουν εργαλεία για την κατασκευή και τη μεταγλώττιση του κώδικα σε εκτελέσιμα δυαδικά αρχεία ή πακέτα. Ενσωματώνονται με συστήματα κατασκευής και μεταγλωττιστές για να αυτοματοποιήσουν τη διαδικασία κατασκευής και ανάπτυξης λογισμικού και με συστήματα ελέγχου εκδόσεων, όπως το **Git**, το **SVN** ή το **Mercurial**, επιτρέποντας στους προγραμματιστές να διαχειρίζονται αποθετήρια κώδικα, να παρακολουθούν τις αλλαγές και να συνεργάζονται με τα μέλη της ομάδας απευθείας μέσα από το **IDE**. Τα **IDE** περιλαμβάνουν λειτουργίες για τη διαχείριση έργων και την οργάνωση αρχείων κώδικα. Αυτό μπορεί να περιλαμβάνει υποστήριξη για πρότυπα έργων, εργαλεία πλοήγησης σε αρχεία και δυνατότητες αναζήτησης σε όλο το έργο, ώστε να βοηθούν τους προγραμματιστές να πλοηγούνται αποτελεσματικά σε μεγάλες βάσεις κώδικα. Επίσης, περιέχουν εργαλεία για την ανάλυση της ποιότητας του κώδικα, τον εντοπισμό πιθανών προβλημάτων και την πρόταση βελτιώσεων του κώδικα. Αυτό μπορεί να περιλαμβάνει χαρακτηριστικά όπως στατική ανάλυση κώδικα, μετρικές κώδικα και αυτοματοποιημένα εργαλεία αναδιαμόρφωσης κώδικα που βοηθούν τους προγραμματιστές να διατηρούν καθαρό και συντηρήσιμο κώδικα. Τα **IDE** μπορούν να ενσωματωθούν με εξωτερικά εργαλεία και βιβλιοθήκες για να επεκτείνουν τη λειτουργικότητά τους. Αυτό μπορεί να περιλαμβάνει ενσωμάτωση με πλαίσια δοκιμών, γεννήτριες κώδικα, εργαλεία τεκμηρίωσης και άλλα βοηθητικά προγράμματα ανάπτυξης για τον εξορθολογισμό της ροής εργασιών ανάπτυξης. Τα ολοκληρωμένα περιβάλλοντα ανάπτυξης είναι σχεδιασμένα για να υποστηρίζουν πολλαπλές γλώσσες προγραμματισμού και πλατφόρμες ανάπτυξης (π.χ. **PyCharm** για την ανάπτυξη **Python**). Τα δημοφιλή **IDE** για την ανάπτυξη **Python** περιλαμβάνουν το **PyCharm**, το **Spyder** και το **Visual Studio Code** με επεκτάσεις

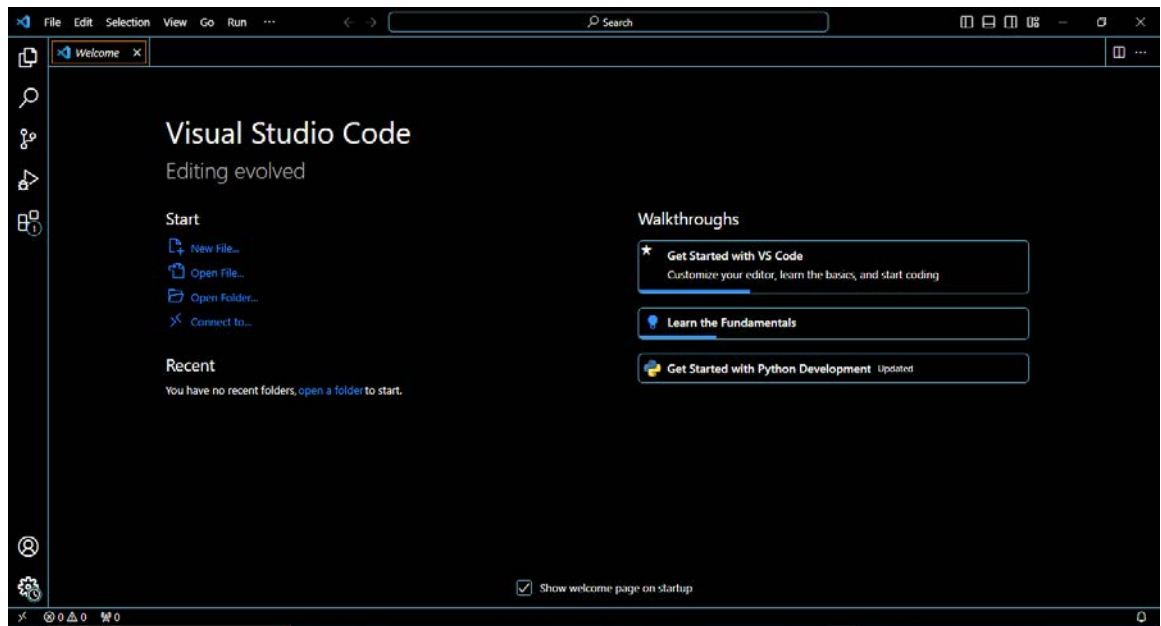
Python.



Παράδειγμα PyCharm, <https://en.wikipedia.org/wiki/PyCharm>

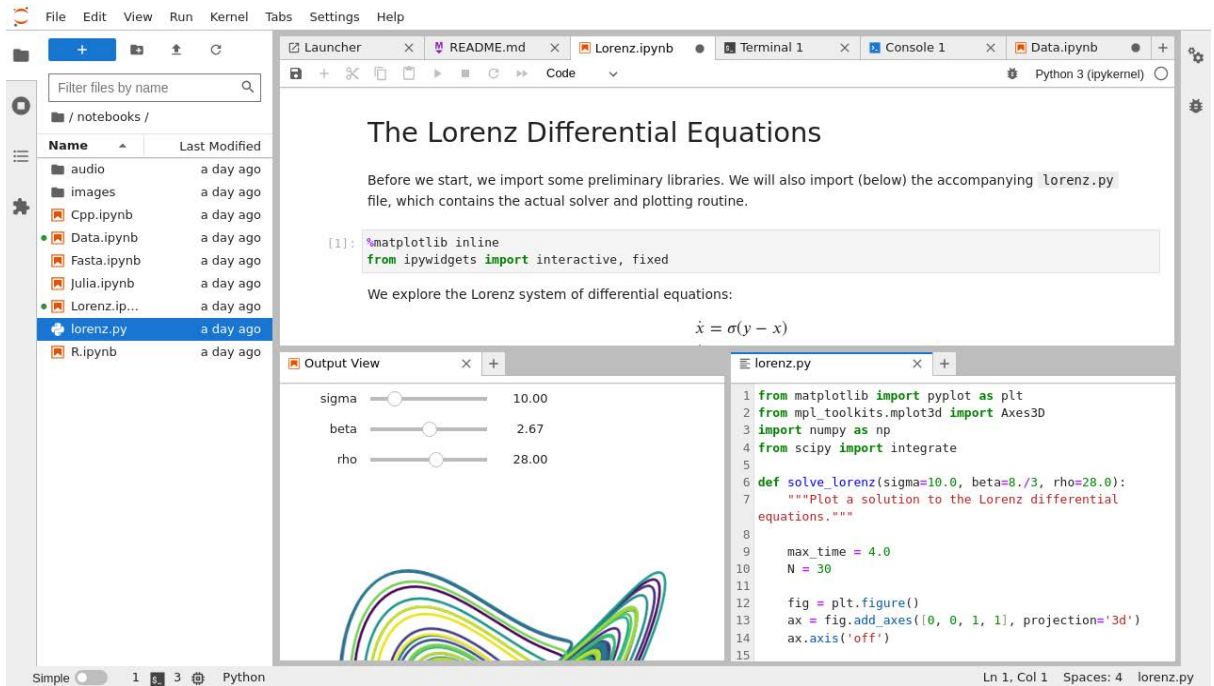


Παράδειγμα Spyder, <https://github.com/spyder-ide/spyder>



Παράδειγμα Virtual Code Studio

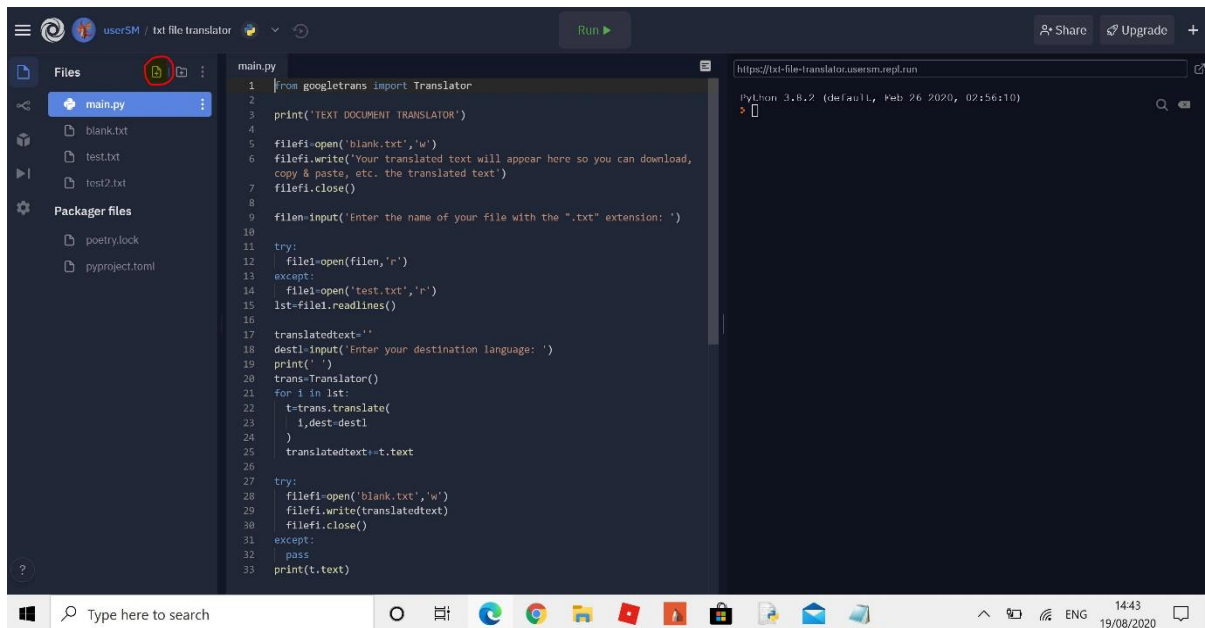
4. Σημειωματάρια **Jupyter**: Τα **Jupyter Notebooks** είναι ένα ισχυρό εργαλείο που χρησιμοποιείται για διαδραστικούς υπολογισμούς, ανάλυση δεδομένων, οπτικοποίηση και τεκμηρίωση. Τα **Jupyter Notebooks** παρέχουν ένα διαδραστικό περιβάλλον υπολογισμού όπου μπορεί να γραφτεί και να εκτελεστεί κώδικας σε μορφή κυψελών. Κάθε κελί μπορεί να περιέχει κώδικα, κείμενο **Markdown**, εξισώσεις (χρησιμοποιώντας σύνταξη **LaTeX**) ή ακόμη και στοιχεία πολυμέσων. Τα κελιά μπορούν να εκτελεστούν μεμονωμένα. Αυτό επιτρέπει μια ευέλικτη και επαναληπτική προσέγγιση στην ανάπτυξη κώδικα και την ανάλυση δεδομένων, καθώς μπορείτε να εκτελέσετε και να δοκιμάσετε τον κώδικα σε μικρά τμήματα χωρίς να επαναλάβετε ολόκληρο το σημειωματάριο. Τα σημειωματάρια **Jupyter** βασίζονται σε μια αρχιτεκτονική πελάτη-εξυπηρετητή, όπου η διεπαφή του σημειωματαρίου (πελάτης) επικοινωνεί με μια υπολογιστική μηχανή (πυρήνας) που εκτελεί τον κώδικα. Ο πυρήνας μπορεί να συσχετιστεί με διαφορετικές γλώσσες προγραμματισμού, με την **IPython** να είναι ο προεπιλεγμένος πυρήνας για την **Python**. Τα σημειωματάρια **Jupyter** υποστηρίζουν πλούσιες μορφές εξόδου, συμπεριλαμβανομένων διαδραστικών οπτικοποιήσεων, διαγραμμάτων, πινάκων και στοιχείων πολυμέσων. Τα σημειωματάρια **Jupyter** χρησιμοποιούνται ευρέως σε περιβάλλοντα εκπαίδευσης και κατάρτισης για τη διδασκαλία του προγραμματισμού, της επιστήμης των δεδομένων και της επιστημονικής πληροφορικής. Η διαδραστική και αφηγηματική προσέγγισή τους βοηθά τους εκπαιδευόμενους να ασχοληθούν με σύνθετες έννοιες και να πειραματιστούν με κώδικα σε πραγματικό χρόνο. Τα **Jupyter Notebooks** μπορούν να επεκταθούν και να προσαρμοστούν χρησιμοποιώντας ένα πλούσιο οικοσύστημα επεκτάσεων και **plugins**. Ενσωματώνονται επίσης με άλλες βιβλιοθήκες και εργαλεία ανάλυσης και οπτικοποίησης δεδομένων, όπως οι **pandas**, **NumPy**, **Matplotlib** και **scikit-learn**.



Παράδειγμα Jupyter Notebook, <https://docs.jupyter.org/en/latest/>

5. Διαδικτυακές πλατφόρμες: Οι διαδικτυακές πλατφόρμες για τη συγγραφή κώδικα παρέχουν διαδικτυακά περιβάλλοντα που μπορεί να γραφτεί, να εκτελεστεί και να μοιραστεί κώδικας χωρίς να χρειάζεται να εγκατασταθεί τοπικά οποιοδήποτε λογισμικό. Οι διαδικτυακές πλατφόρμες κωδικοποίησης είναι προσβάσιμες από οποιαδήποτε συσκευή με σύνδεση στο διαδίκτυο και πρόγραμμα περιήγησης στο διαδίκτυο, γεγονός που τις καθιστά βολικές για την κωδικοποίηση εν κινήσει ή σε συσκευές όπου η εγκατάσταση λογισμικού δεν είναι δυνατή ή επιτρεπτή. Δεδομένου ότι οι διαδικτυακές πλατφόρμες εκτελούνται εξ ολοκλήρου στο πρόγραμμα περιήγησης, δεν χρειάζεται να δημιουργηθεί περιβάλλον ανάπτυξης ή να εγκατασταθούν εξαρτήσεις. Αυτό μειώνει το εμπόδιο εισόδου για τους αρχάριους και επιτρέπει στους χρήστες να ξεκινήσουν αμέσως την κωδικοποίηση. Πολλές διαδικτυακές πλατφόρμες κωδικοποίησης υποστηρίζουν χαρακτηριστικά συνεργασίας, όπως επεξεργασία και σχολιασμό σε πραγματικό χρόνο, επιτρέποντας σε πολλούς χρήστες να εργάζονται ταυτόχρονα στην ίδια βάση κώδικα. Αυτό προάγει την ομαδική εργασία και διευκολύνει τις αναθεωρήσεις κώδικα και τον προγραμματισμό σε ζεύγη. Επίσης, κάποιες διαδικτυακές πλατφόρμες ενσωματώνονται με συστήματα ελέγχου εκδόσεων όπως το **Git**, επιτρέποντας στους χρήστες να διαχειρίζονται αποθετήρια κώδικα, να παρακολουθούν τις αλλαγές και να συνεργάζονται σε έργα χρησιμοποιώντας οικείες ροές εργασίας ελέγχου εκδόσεων. Οι διαδικτυακές πλατφόρμες κωδικοποίησης συχνά παρέχουν εκπαιδευτικούς πόρους, σεμινάρια και προκλήσεις κωδικοποίησης για να βοηθήσουν τους χρήστες να βελτιώσουν τις δεξιότητές τους στον προγραμματισμό. Μπορεί επίσης να προσφέρουν διαδραστικά περιβάλλοντα κωδικοποίησης όπου οι χρήστες μπορούν να πειραματιστούν με τον κώδικα και να λάβουν άμεση

ανατροφοδότηση. Ακόμα, η ασφάλεια αποτελεί κρίσιμη πτυχή των διαδικτυακών πλατφορμών κωδικοποίησης, καθώς οι χρήστες συχνά αλληλεπιδρούν με ευαίσθητα δεδομένα και κώδικα. Οι πάροχοι πλατφορμών εφαρμόζουν μέτρα όπως κρυπτογράφηση, ελέγχους πρόσβασης και ασφαλείς μηχανισμούς ελέγχου ταυτότητας για την προστασία των δεδομένων των χρηστών και τη διασφάλιση της ιδιωτικής ζωής. Παραδείγματα δημοφιλών διαδικτυακών πλατφορμών κωδικοποίησης περιλαμβάνουν τις **Repl.it**, **Codecademy**, **HackerRank**, **LeetCode** και **GitHub Codespaces**.



The screenshot shows a web-based Python IDE interface. On the left, a file explorer shows a project named 'txt file translator' with files 'main.py', 'blank.txt', 'test.txt', and 'test2.txt'. The 'main.py' file is open in the editor, displaying a Python script that uses the 'googletrans' library to translate text from one file to another. The script prompts the user for a file name and a destination language, then performs the translation and saves the result to 'blank.txt'. The right side of the interface shows the terminal output, which is currently empty. The top of the interface includes a 'Run' button and a 'Share' link. The bottom of the interface shows a Windows taskbar with various application icons and a system clock indicating 14:43 on 19/08/2020.

```
1 from googletrans import Translator
2
3 print('TEXT DOCUMENT TRANSLATOR')
4
5 file1=open('blank.txt','w')
6 file1.write('Your translated text will appear here so you can download,
7 copy & paste, etc. the translated text')
8 file1.close()
9
10 file1=input('Enter the name of your file with the ".txt" extension: ')
11
12 try:
13     file1=open(file1,'r')
14 except:
15     file1=open('test.txt','r')
16 lst=file1.readlines()
17
18 translatedtext=''
19 destl=input('Enter your destination language: ')
20 print(' ')
21 trans=Translator()
22 for i in lst:
23     t=trans.translate(
24         i,destl
25     )
26     translatedtext+=t.text
27
28 try:
29     file1=open('blank.txt','w')
30     file1.write(translatedtext)
31     file1.close()
32 except:
33     pass
34 print(t.text)
```

Παράδειγμα **Repl.it**, <https://replit.com/talk/learn/An-addition-guide-to-Replit-for-newcomers/50257>

```

test.rb
1 # Todo List Classes
2
3 class Person
4   def initialize(name, age)
5     @name = name
6     @age = age
7   end
8
9   def greet
10    puts "Hi, my name is #{@name}"
11  end
12 end
13
14 me = Person.new("Jon", 28)
15 me.greet
16
17 class List
18   def initialize
19     @all_tasks = []
20   end
21
22   def add_task(task)
23     @all_tasks << task
24   end
25
26   def show_all_tasks
27     @all_tasks
28   end
29
30   def save
31     list_file = File.open("todo_list.txt", "w")
32     @all_tasks.each do |task|
33

```

Παράδειγμα CodeAcademy, <https://github.com/jonsamp/codecademy-syntax>

Task Description

A binary number is represented as a series of 0s and 1s. In this challenge, the series will be in the form of a singly-linked list. Each node instance, a *LinkedListNode*, has a value, *data*, and a pointer to the next node, *next*. Given a reference to the head of a singly-linked list, convert the binary number represented to a decimal number.

Example

Linked list corresponding to the binary number $(010011)_2$ or $(19)_{10}$

Function Description

Complete the function *getNumber* in the editor below.

getNumber has the following parameter(s):

- binary*: reference to the head of a singly-linked list of binary digits

Returns:

- int: a (long integer)₁₀ representation of the binary number

Constraints

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

```

#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);

typedef struct SinglyLinkedListNode SinglyLinkedListNode;
typedef struct SinglyLinkedList SinglyLinkedList;

struct SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode* next;
};

struct SinglyLinkedList {
    SinglyLinkedListNode* head;
    SinglyLinkedListNode* tail;
};

SinglyLinkedListNode* create_singly_linked_list_node(int node_data) {
    SinglyLinkedListNode* node = malloc(sizeof(SinglyLinkedListNode));

```

Language selection dropdown: C, Clojure, C++, C++14, C#, Erlang, Go, Haskell, Java 7

Run Tests Run Code

Παράδειγμα HackerRank, <https://candidatesupport.hackerrank.com/hc/en-us/articles/4402914035091-Interview-Preparation-Checklist-for-Candidates>

The screenshot shows the LeetCode interface for problem 129. The left sidebar contains the problem description, difficulty (Medium), and statistics (5.5K votes, 96 comments). The main area displays the problem statement: "You are given the root of a binary tree containing digits from 0 to 9 only. Each root-to-leaf path in the tree represents a number. For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123. Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a 32-bit integer. A leaf node is a node with no children." The right sidebar shows a Python3 solution snippet:

```

1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def sumNumbers(self, root: Optional[TreeNode]) ->
9         int:

```

At the bottom, there are buttons for 'Console', 'Run', and 'Submit'.

Παράδειγμα LeetCode, <https://interviewguide.dev/blog/posts/neetcode-vs-leetcode/>

The screenshot shows the GitHub Codespace interface. The left sidebar displays the file explorer with a project structure including folders like '.devcontainer', '.github', and 'app', and files like 'emotion.rb' and 'interaction.rb'. The main editor area shows the content of 'emotion.rb', which includes a class definition for 'Emotion' with methods for initializing content, label, pronounceable_label, emoji_character, and platform_enum. The bottom panel shows the terminal output, which includes logs for starting and finishing various extensions and the Monaco editor.

Παράδειγμα GitHubCodeSpace, <https://nira.com/github-codespaces/>

ΚΕΦΑΛΑΙΟ 2 TKINTER

Η **Tkinter** αποτελεί μία βιβλιοθήκη της γλώσσας προγραμματισμού **Python** που χρησιμοποιείται για την δημιουργία γραφικών διεπαφών χρήστη(**GUI**). Οι γραφικές διεπαφές χρήστη επιτρέπουν στους χρήστες να αλληλεπιδρούν με το σύστημα χειριζόμενοι οπτικά στοιχεία, αντί να πληκτρολογούν εντολές. Η **Tkinter** είναι πρότυπο **GUI toolkit** για την **Python**. Αυτό σημαίνει ότι έρχεται προ εγκατεστημένη στις περισσότερες εκδόσεις της **Python**. Αυτό δε σημαίνει, βέβαια, ότι αποτελεί το καλύτερο ή το μόνο εργαλείο για την επίλυση προβλημάτων. Ως **GUI toolkit** αναφέρεται μία συλλογή λογισμικών, η οποία βοηθάει τους προγραμματιστές να δημιουργούν εφαρμογές χωρίς να τους απασχολεί το λειτουργικό σύστημα στο οποίο λειτουργεί η πλατφόρμα εμφάνισης, δηλαδή είτε αυτό είναι **Windows**, **MacOs** ή **Linux** εφόσον η μετατροπή γίνεται αυτόματα. Η **Tkinter** είναι χρήσιμη για εφαρμογές που το μέγεθος τους δεν είναι μεγάλο και χρειάζονται αμεσότητα και ταχύτητα. Πολλές φορές με τη χρήση της συναντάμε την γλώσσα προγραμματισμού **Tool Command Language(TCL)**, η οποία δημιουργήθηκε από τον **John Ousterhout** το 1988 και σχεδιάστηκε για την ευκολία, την απλότητα. Το κύριο χαρακτηριστικό της γνώριμα είναι η προσθήκη **widgets** σε άλλες γλώσσες. Ο συνδυασμός αυτών των δύο προσφέρει:

1. ευκολία χρήσης
2. Ταχύτητα
3. ευελιξία εφαρμογών

Η **Tkinter** περιλαμβάνει πολλές κλάσεις(**widgets**). Κάποια από τα βασικά και ευρέως χρησιμοποιημένα είναι τα:

- **Label**, για δημιουργία ετικέτας
- **Button**, για δημιουργία κουμπιού
- **Entry**, για δημιουργία κειμένου μίας γραμμής
- **Text**, για επεξεργασία κειμένου
- **Frame**, για την επεξεργασία της διάταξης
- **Checkbutton**, για δημιουργία κουμπιού με 2 επιλογές
- **Radiobutton**, για δημιουργία κουμπιού που καλεί συνάρτηση
- **Listbox**, για την εμφάνιση στοιχείων
- **Scale**, για την επιλογή αριθμού από καθορισμένο μέγεθος
- **Canvas**, για τη σχεδίαση γραφικών στοιχείων

WIDGETS:

Widget Label

Το **widget Label** χρησιμοποιείται για την εμφάνιση κειμένου ή εικόνων. Αρχικός σκοπός του είναι η εμφάνιση κειμένου, στο οποίο ο χρήστης δεν έχει την δυνατότητα της αλληλεπίδρασης. Επίσης, οι εντολές **bg** και **fg** είναι υπεύθυνες για το χρώμα του φόντου και του κειμένου αντίστοιχα. Με το **justify** ορίζεται η στοίχιση του κειμένου. Με **wrplength** ορίζεται το μέγιστο μήκος γραμμής.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο Tkinter :

```
1 import tkinter as tk
2
3 # Create the main window
4 root = tk.Tk()
5 root.title("Label Widget Example")
6
7 # Create a Label widget with customized appearance
8 custom_label = tk.Label(
9     root,
10    text="Customized Label",
11    font=("Arial", 16),
12    fg="blue",
13    bg="lightgray",
14    padx=20,
15    pady=10
16 )
17 custom_label.pack(pady=20)
18
19 # Start the Tkinter event loop
20 root.mainloop()
```

Εικόνα 2.1.α. Παράδειγμα χρήσης του Label widget

Widget Button

Το **widget Button** χρησιμοποιείται για τη δημιουργία ενός κουμπιού με δυνατότητα επιλογής από το χρήστη. Τα κουμπιά μπορούν να έχουν διάφορες καταστάσεις, όπως ενεργή, μη ενεργή και κανονική. Το **state** μπορούμε να το επιλέξουμε είτε για να ορίσουμε το κουμπί, είτε για να αλλάξουμε την κατάσταση του. Τα κουμπιά έχουν τη δυνατότητα να δεσμεύουν ενέργειες, όπως για παράδειγμα μια συνάρτηση να έχει την εντολή **<Button-1>** που αντιστοιχεί στο αριστερό κλικ του ποντικιού.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο Tkinter με ένα κουμπί (**Button**). Όταν ο χρήστης πατήσει το κουμπί, εκτελείται η συνάρτηση **BUTTON**:

```

1  import tkinter as tk
2
3  # Function to be called when the button is clicked
4  def button_click():
5      |   label.config(text="Button Clicked!")
6
7  # Create the main window
8  root = tk.Tk()
9  root.title("Button Widget Example")
10
11 # Create a Button widget
12 button = tk.Button(root, text="Click Me", command=button_click)
13 button.pack(pady=20)
14
15 # Create a Label to display the result
16 label = tk.Label(root, text="")
17 label.pack()
18
19 # Start the Tkinter event loop
20 root.mainloop()

```

Εικόνα 2.1.β. Παράδειγμα χρήσης του **Button widget**

WidgetEntry

Το **widget Entry** χρησιμοποιείται προκειμένου να δημιουργηθεί ένα πεδίο κειμένου μίας γραμμής στο οποίο ο χρήστης μπορεί να προσθέσει και να τροποποιήσει τα δεδομένα, κάποιες βασικές επιλογές του είναι

- το **width**, που καθορίζει το πλάτος του **Entry**
- το **show**, που αναλαμβάνει να κρύψει τους χαρακτήρες που χρησιμοποιήθηκαν με κάτι που προκαθορίζει ο χρήστης
- το **textvariable**, που συνδέει μια μεταβλητή **Stringvar**, η οποία είναι μία κλάση μεταβλητής που κρατάει την τιμή μίας μεταβλητής και δημιουργεί μια σύνδεση με το **widget** της **Tkinter** και όποιες αλλαγές συμβαίνουν ενημερώνονται αυτόματα

Κάποιες σημαντικές μέθοδοι της **Entry** είναι:

- το **get()**, με το οποίο επιστρέφεται η καταχώρηση ως συμβολοσειρά
- το **delete(start, end)**, με το οποίο διαγράφονται οι χαρακτήρες
- το **insert(index, string)**, με το οποίο εισάγονται οι χαρακτήρες.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter** που φαίνονται οι λειτουργίες όλων των παραπάνω:

```

1  import tkinter as tk
2
3  def get_entry_text():
4      entered_text = entry.get()
5      print("Entered text:", entered_text)
6
7  # Create the main window
8  root = tk.Tk()
9  root.title("Entry Widget Example")
10
11 # Create an Entry widget
12 entry = tk.Entry(root, width=30)
13
14 # Create a button to get the entry text
15 button = tk.Button(root, text="Get Text", command=get_entry_text)
16
17 # Pack the Entry widget and button
18 entry.pack(pady=10)
19 button.pack(pady=10)
20
21 # Start the Tkinter event loop
22 root.mainloop()

```

Εικόνα 2.1.γ. Παράδειγμα χρήσης του **Entry widget**

Widget Text

Το **widget Text** χρησιμοποιείται για την επεξεργασία και την εμφάνιση κειμένου, το οποίο ο χρήστης μπορεί να επεξεργαστεί. Κάποια από τα βασικά χαρακτηριστικά της είναι διάφορες γραμματοσειρές, χρώματα και στυλ. Ακόμα, έχει κάποιες βασικές εντολές, όπως **insert(index, text)** εισάγει το κείμενο,

get(a, b) ανακτά το κείμενο μεταξύ των δεικτών, **delete(start, end)** διαγράφει το κείμενο μεταξύ των δεικτών, **tag_add(tag, a, b)** συνδέει μια ετικέτα στην καθορισμένη περιοχή, **tag_config(tag, **options)** διαμορφώνει την εμφάνιση του κειμένου σύμφωνα με την ετικέτα, στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**

```

1 import tkinter as tk
2
3 def get_text_content():
4     text_content = text_widget.get("1.0", tk.END) # Retrieve all text from the beginning to the end
5     print("Text content:\n", text_content)
6
7 def clear_text():
8     text_widget.delete("1.0", tk.END) # Delete all text from the beginning to the end
9
10 # Create the main window
11 root = tk.Tk()
12 root.title("Text Widget Example")
13
14 # Create a Text widget
15 text_widget = tk.Text(root, wrap=tk.WORD, height=10, width=40)
16 text_widget.pack(pady=10)
17
18 # Create buttons to interact with the Text widget
19 get_text_button = tk.Button(root, text="Get Text", command=get_text_content)
20 clear_button = tk.Button(root, text="Clear Text", command=clear_text)
21
22 get_text_button.pack(pady=5)
23 clear_button.pack(pady=5)
24
25 # Start the Tkinter event loop
26 root.mainloop()
27

```

Εικόνα 2.1.δ. Παράδειγμα χρήσης του **Text widget**

WidgetFrame

Το **widget Frame** χρησιμοποιείται για να οργανώσει τη διάταξη και την ομαδοποίηση άλλων **widgets**. Κάποιες από τις βασικές του ιδιότητες είναι η δημιουργία ορίων. Αυτό γίνεται με την εντολή **bd(border)** και με το **relief**, που μας δείχνει τον τύπο του ορίου συνήθως είναι **flat**, **raised**, **sunken** και **ridge**. Η εντολή **background(bg)**, είναι υπεύθυνη για την αλλαγή του χρώματος του φόντου.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**

```

1 import tkinter as tk
2
3 # Create the main window
4 root = tk.Tk()
5 root.title("Tkinter Frame Example")
6
7 # Create a Frame
8 frame = tk.Frame(root)
9 frame.pack(padx=10, pady=10) # Use pack geometry manager to organize the frame
10
11 # Add widgets to the frame
12 label = tk.Label(frame, text="This is a label inside the frame")
13 label.pack()
14
15 button = tk.Button(frame, text="Click me!")
16 button.pack()
17
18 # Run the main loop
19 root.mainloop()
20

```

Εικόνα 2.1.ε. Παράδειγμα χρήσης του **Frame widget**

Widget Checkbutton

Το **Checkbutton** χρησιμοποιείται σε ρυθμίσεις προτιμήσεων και σε οποιαδήποτε άλλη κατάσταση στην οποία απαιτείται ο χρήστης να επιλέξει ανάμεσα σε δύο καταστάσεις (επιλεγμένο(**checked**) ή μη επιλεγμένο(**unchecked**)). Για το **Checkbutton** μπορεί να γίνει η χρήση μίας μεταβλητής με την οποία καθορίζεται η κατάσταση. Συνήθως έχουμε

- μια μεταβλητή **BooleanVar**, αν θέλουμε να έχουμε μια δυαδική κατάσταση(πχ **on/off**),
- μια **StringVar** ή μια **IntVar**, αν θέλουμε να συνδεθεί κάθε κατάσταση με την τιμή της συμβολοσειράς.

Επίσης, η εμφάνιση του μπορεί να καθοριστεί με ένα κείμενο, μια γραμματοσειρά και ένα φόντο. Απαραίτητες, συνήθως, είναι και οι επιλογές **padding** και **relief (padx, pady)** για καλύτερη οπτική παρουσία.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```
1 import tkinter as tk
2
3 def on_checkbox_click():
4     | selected_option.set("Selected" if checkbox_var.get() else "Not Selected")
5
6 # Create the main window
7 root = tk.Tk()
8 root.title("Checkbutton Example")
9
10 # Create a variable to store the state of the checkbox
11 checkbox_var = tk.BooleanVar()
12
13 # Set an initial value for the checkbox
14 checkbox_var.set(True)
15
16 # Create a Checkbutton widget
17 checkbox = tk.Checkbutton(root, text="Option", variable=checkbox_var, command=on_checkbox_click)
18
19 # Create a label to display the selected option
20 selected_option = tk.StringVar()
21 selected_option_label = tk.Label(root, textvariable=selected_option)
22
23 # Pack the Checkbutton and label into the main window
24 checkbox.pack(pady=10)
25 selected_option_label.pack(pady=10)
26
27 # Run the Tkinter event loop
28 root.mainloop()
29
```

Εικόνα 2.1.στ. Παράδειγμα χρήσης του **Checkbutton** widget

Widget Radiobutton

Το **Radiobutton widget** χρησιμοποιείται προκειμένου να δώσει στο χρήστη να κάνει μία επιλογή σε μια λίστα αμοιβαία αποκλειόμενων επιλογών, όπως για παράδειγμα η επιλογή μίας διαδρομής ή ενός φύλλου.

Το **Radiobutton** απαιτεί την χρήση μίας συνάρτησης επανάκλησης(**command**). Η επιλογή της τιμής(**value**) χρησιμοποιείται για τον καθορισμό ενός μοναδικού αναγνωριστικού για κάθε κουμπί επιλογής μέσα στην ομάδα, το **variable** είναι απαραίτητο καθώς συνδέει όλα τα κουμπιά μεταξύ τους και διασφαλίζει τη μοναδικότητα του καθενός. Ακόμα, η εμφάνιση του κουμπιού **Radiobutton** καθορίζεται με το ίδιο τρόπο όπως το **Checkbutton**.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```
1 import tkinter as tk
2 def on_radio_button_click():
3 |     selected_option.set(f"Selected option: {radio_var.get()}")
4
5 # Create the main window
6 root = tk.Tk()
7 root.title("Radiobutton Example")
8
9 # Create a variable to store the selected radio button value
10 radio_var = tk.StringVar()
11
12 # Set an initial value for the radio buttons (optional)
13 radio_var.set("Option 1")
14
15 # Create Radiobutton widgets with a common variable and command callback
16 radio_button1 = tk.Radiobutton(root, text="Option 1", variable=radio_var, value="Option 1", command=on_radio_button_click)
17 radio_button2 = tk.Radiobutton(root, text="Option 2", variable=radio_var, value="Option 2", command=on_radio_button_click)
18 radio_button3 = tk.Radiobutton(root, text="Option 3", variable=radio_var, value="Option 3", command=on_radio_button_click)
19
20 # Create a label to display the selected option
21 selected_option = tk.StringVar()
22 selected_option_label = tk.Label(root, textvariable=selected_option)
23
24 # Pack the Radiobuttons and label into the main window
25 radio_button1.pack(pady=5)
26 radio_button2.pack(pady=5)
27 radio_button3.pack(pady=5)
28 selected_option_label.pack(pady=10)
29 # Run the Tkinter event loop
30 root.mainloop()
```

Εικόνα 2.1.η. Παράδειγμα χρήσης του **Radiobuttonwidget**

Widget Listbox

Το **widgetListbox** χρησιμοποιείται για να εμφανιστεί μία λίστα στοιχείων, που μπορούν να είναι συμβολοσειρές, αριθμοί ή οποιοσδήποτε άλλος τύπος δεδομένων, από τα οποία ο χρήστης μπορεί να επιλέξει ένα ή περισσότερα. Ακόμα, ο χρήστης έχει τη δυνατότητα να δεσμεύσει συναρτήσεις προκειμένου να χειριστεί συμβάντα επιλογής, μπορεί να χρησιμοποιήσει το ευρετήριο για να αναζητήσει συγκεκριμένα στοιχεία ή να χρησιμοποιήσει τη γραμμή κύλισης αν τα στοιχεία δεν βρίσκονται στην ορατή του περιοχή,

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter** με την προσθήκη και τη διαγραφή στοιχείων από το **ListBox**:

```
1 import tkinter as tk
2 def get_selected_item():
3     selected_index = listbox.curselection()
4     if selected_index:
5         item = listbox.get(selected_index[0])
6         result_label.config(text=f"Selected item: {item}")
7
8 # Create the main window
9 root = tk.Tk()
10 root.title("ListBox Widget Example")
11
12 # Create a ListBox widget
13 listbox = tk.Listbox(root, selectmode=tk.SINGLE)
14 listbox.pack(pady=10)
15
16 # Add items to the ListBox
17 items = ["Item 1", "Item 2", "Item 3", "Item 4", "Item 5"]
18 for item in items:
19     listbox.insert(tk.END, item)
20
21 # Create a button to get the selected item
22 get_item_button = tk.Button(root, text="Get Selected Item", command=get_selected_item)
23 get_item_button.pack(pady=5)
24
25 # Create a label to display the result
26 result_label = tk.Label(root, text="")
27 result_label.pack(pady=10)
28
29 # Start the Tkinter event loop
30 root.mainloop()
```

Εικόνα 2.1.Θ. Παράδειγμα χρήσης του **ListBoxwidget**

Widget Scale

Το **widget Scale** χρησιμοποιείται προκειμένου ο χρήστης να επιλέξει μία τιμή μέσα σε ένα καθορισμένο εύρος τιμών, η παράμετρος από(**from**) αποτελεί την αρχή και η παράμετρος έως(**to**) αποτελεί το τέλος. Με το **orient** καθορίζεται ο προσανατολισμός του, με (**tk.HORIZONTAL**) για οριζόντιο ή σε (**tk.VERTICAL**) για κάθετο, με το **length** το μήκος του, με το **resolution** τα κλιμακωτά βήματα με τα οποία αλλάζει η τιμή του.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```

1  import tkinter as tk
2
3  # Function to display the selected value
4  def update_label(value):
5      |   label.config(text=f"Selected Value: {value}")
6
7  # Create the main window
8  root = tk.Tk()
9  root.title("Scale Widget Example")
10
11 # Create a Scale widget
12 scale = tk.Scale(root, from_=0, to=100, orient="horizontal", command=update_label)
13 scale.pack(pady=20)
14
15 # Create a label to display the selected value
16 label = tk.Label(root, text="Selected Value: 0")
17 label.pack()
18
19 # Start the Tkinter event loop
20 root.mainloop()

```

Εικόνα 2.1.ι. Παράδειγμα χρήσης του **Scale widget**

Widget Canvas

Το **widget Canvas** χρησιμοποιείται για να σχεδιάζονται σχήματα, εικόνες και άλλα γραφικά στοιχεία. Παρέχει μια περιοχή σχεδίασης που μπορούν να σχεδιαστούν και να εμφανιστούν γραφικά στοιχεία, όπως γραμμές, ορθογώνια, οβάλ, πολύγωνα και κείμενο, χρησιμοποιεί ένα σύστημα συντεταγμένων όπου το (0,0) αντιπροσωπεύει την επάνω αριστερή γωνία. Μπορείτε να καθορίσετε συντεταγμένες για να τοποθετήσετε και να χειριστείτε αντικείμενα στον καμβά.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```

1  import tkinter as tk
2
3  # Function to draw a rectangle on the canvas
4  def draw_rectangle():
5      canvas.create_rectangle(50, 50, 200, 150, fill="blue")
6
7  # Create the main window
8  root = tk.Tk()
9  root.title("Canvas Widget Example")
10
11 # Create a Canvas widget
12 canvas = tk.Canvas(root, width=300, height=200)
13 canvas.pack(pady=20)
14
15 # Create a button to draw a rectangle
16 draw_button = tk.Button(root, text="Draw Rectangle", command=draw_rectangle)
17 draw_button.pack()
18
19 # Start the Tkinter event loop
20 root.mainloop()

```

Εικόνα 2.1.κ. Παράδειγμα χρήσης του **Canvas widget**

MODULES:

Ακόμα, η **Tkinter** έχει διάφορες επιπρόσθετες ενότητες (**modules**) οι βασικότερες είναι:

- **Tkinter.messagebox**: δημιουργία μηνύματος
- **Tkinter.filedialog**: επιλογή αρχείου
- **Tkinter.colorchooser**: επιλογή χρώματος
- **Tkinter.ttk**: βελτίωση εμφάνισης σε εφαρμογές **Tkinter**
- **Tkinter.scrolledtext**: δημιουργία κειμένου με δυνατότητα κύλισης
- **Tkinter.simpledialog**: απλοποίηση διαλόγων

Tkinter.messagebox

Το **module Tkinter.messagebox** χρησιμοποιείται προκειμένου να αναδύονται πλαίσια από τα οποία εμφανίζονται πληροφορίες, προειδοποιήσεις, σφάλματα ή ζητούν άδεια από το χρήστη για μία ενέργεια. Με αυτό το **module** δημιουργούνται απλά ή **modal** (σημαίνει ότι είναι παράθυρο που θα ολοκληρωθεί η όποια ενέργεια είναι να γίνει και θα κλείσει πριν συνεχιστεί το κύριο πρόγραμμα) πλαίσια διαλόγου που εμφανίζουν διάφορα μηνύματα. Κάποιοι χαρακτηριστικοί τύποι μηνυμάτων είναι:

- **showinfo()**: ενημερωτικό μήνυμα.
- **showwarning()**: προειδοποιητικό μήνυμα.
- **showerror()**: μήνυμα σφάλματος
- **askquestion()**: ερώτηση με απαντήσεις(Ναι και Όχι)
- **askokcancel()**: μήνυμα επιβεβαίωσης(OK και Cancel)

- **askyesno()**: μήνυμα επιβεβαίωσης (**True** ή **False**)
- **askretrycancel()**: μήνυμα για επανάληψη ή ακύρωση

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```

1 import tkinter as tk
2 from tkinter import messagebox
3
4 # Create the main window
5 root = tk.Tk()
6 root.title("Message Box Example")
7
8 # Function to show a simple info message box
9 def show_info():
10 |     messagebox.showinfo("Information", "This is an information message!")
11
12 # Button to trigger the message box
13 info_button = tk.Button(root, text="Show Info", command=show_info)
14 info_button.pack(pady=20)
15
16 # Start the GUI event loop
17 root.mainloop()

```

Εικόνα 2.2.α. Παράδειγμα χρήσης του **Tkinter.messagebox** module

Tkinter.filedialog

Το **module Tkinter.filedialog** χρησιμοποιείται για να ζητείται από το χρήστη η επιλογή αρχείων ή καταλόγων, αυτό επιτυγχάνεται μέσω πλαισίων διαλόγου. Επίσης, αυτό το **module** είναι **modal**. Κάποια βασικά παράθυρα είναι :

- **askopenfilename()**: επιλογή ενός αρχείου
- **askopenfilenames()**: επιλογή πολλών αρχείων
- **asksaveasfilename()**: αποθήκευση ενός αρχείου
- **askdirectory()**: επιλογή ενός καταλόγου

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```

1  import tkinter as tk
2  from tkinter import filedialog
3
4  # Create the main window
5  root = tk.Tk()
6  root.title("File Dialog Example")
7
8  # Function to open a file dialog and print the selected file path
9  def open_file_dialog():
10     file_path = filedialog.askopenfilename(title="Select a File")
11     if file_path:
12         print(f"Selected file: {file_path}")
13
14 # Button to trigger the file dialog
15 open_button = tk.Button(root, text="Open File", command=open_file_dialog)
16 open_button.pack(pady=20)
17
18 # Start the GUI event loop
19 root.mainloop()

```

Εικόνα 2.2.β. Παράδειγμα χρήσης του **Tkinter.filedialog** module

Tkinter.colorchooser

Το module **Tkinter.colorchooser** χρησιμοποιείται προκειμένου ο χρήστης να μπορεί να επιλέξει χρώματα φόντου, χρώματα κειμένου ή χρώματα σχεδίασης σε εφαρμογές γραφικών. Η κύρια συνάρτηση αυτού του module είναι η **askcolour**, με την κλήση της ανοίγει ένα παράθυρο στο οποίο δίνετε η δυνατότητα στο χρήστη να επιλέξει το χρώμα που επιθυμεί το οποίο και επιστρέφεται. Επίσης, αυτό το module χαρακτηρίζεται από την απλότητα και την ευκολία που παρέχεται στο χρήστη.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```

1  import tkinter as tk
2  from tkinter import colorchooser
3
4  # Create the main window
5  root = tk.Tk()
6  root.withdraw() # Hide the main window
7
8  # Open a color chooser dialog
9  selected_color = colorchooser.askcolor(title="Choose a color")
10
11 print("Selected color (RGB, Hex):", selected_color)

```

Εικόνα 2.2.γ. Παράδειγμα χρήσης του **Tkinter.colorchooser** module

Tkinter.ttk

Το **Tkinter.ttk** χρησιμοποιείται για να δώσει μία αναβάθμιση στα είδη υπάρχοντα **widgets**, ώστε οι εφαρμογές που δημιουργούνται με τη βιβλιοθήκη **Tkinter** να έχουν μία καλύτερη οπτική εικόνα και χρηστικότητα. Τα βασικά χαρακτηριστικά αυτού του module είναι το **Themed Tk Widgets** με το οποίο επιτυγχάνεται η προσθήκη θεμάτων για να παρέχουν μια σύγχρονη και εγγενή εμφάνιση σε διαφορετικά λειτουργικά συστήματα. Επίσης, πέρα από τα βασικά **widgets** (**Label**, **Button**, **Entry**, **Frame**) εισάγονται και κάποια

άλλα το **Combobox**, συνδυασμός πεδίου κειμένου και αναπτυσσόμενης λίστας , η **Progressbar** , εμφάνιση της προόδου μιας εργασίας ,και το **Treeview**, εμφάνιση ιεραρχικών δεδομένων.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```
1 import tkinter as tk
2 from tkinter import ttk
3
4 root = tk.Tk()
5 root.title("ttk Example")
6
7 # Create a ttk button and place it
8 button = ttk.Button(root, text="Click Me")
9 button.pack()
10
11 root.mainloop()
```

Εικόνα 2.2.δ. Παράδειγμα χρήσης του Tkinter.ttk module

Tkinter.scrolledtext

Το **Tkinter.scrolledtext** αποτελεί μία αναβάθμιση του **widget Text**. Έχει ακριβώς τις ίδιες δυνατότητες με το **Text** αλλά ειδικεύεται σε περιπτώσεις μεγάλων κειμένων, όπως αρχεία καταγραφής ή ιστορικά συνομιλιών. Το βασικό χαρακτηριστικό του είναι η ενσωματωμένη γραμμή κύλισης , η οποία εμφανίζεται όταν το περιεχόμενο του κειμένου υπερβαίνει την ορατή περιοχή του **widget**. Επίσης, μπορεί να χειριστεί την προσθήκη άλλων **widgets** και διάφορες λειτουργίες χειρισμού και επεξεργασίας κειμένου.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```
1 import tkinter as tk
2 from tkinter import scrolledtext
3
4 # Create the main window
5 root = tk.Tk()
6
7 # Create a ScrolledText widget
8 scrolled_text = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=40, height=10)
9 scrolled_text.pack(fill=tk.BOTH, expand=True)
10
11 root.mainloop()
```

Εικόνα 2.2.ε. Παράδειγμα χρήσης του Tkinter.scrolledtext module

Tkinter.simpdialog

Το **Tkinter.simpdialog** χρησιμοποιείται για να προσφέρει στο χρήστη ένα μεγάλο εύρος συναρτήσεων και κλάσεων, με τα οποία δεν χρειάζεται η δημιουργία προσαρμοσμένων παραθύρων διαλόγου αλλά γίνεται άμεση εισαγωγή των δεδομένων που ζητούνται. Περιλαμβάνει 3 βασικές συναρτήσεις, **AskString**, για να εισάγει μια συμβολοσειρά, **AskInteger**, έναν ακέραιο αριθμό ή **AskFloat**, ένα **float**. Ακόμα, με την κλάση **Dialog** προσφέρει προκαθορισμένα παράθυρα διαλόγου, η εμφάνιση και η συμπεριφορά αυτών των διαλόγων μπορούν να προσαρμοστούν σε κάποιο βαθμό για βασικούς τύπους εισόδου και επιτρέπει τον πλήρη έλεγχο της διάταξης και της λειτουργικότητας του πλαισίου διαλόγου.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήσαμε ένα παράθυρο **Tkinter**:

```
1 import tkinter as tk
2 from tkinter import simpdialog
3
4 root = tk.Tk()
5 root.withdraw() # Hides the main window
6
7 # Asking the user for an integer
8 user_input = simpdialog.askinteger("Input", "Enter a number")
9
10 print("User entered:", user_input)
```

Εικόνα 2.2.στ. Παράδειγμα χρήσης του **Tkinter.simpdialogmodule**

LAYOUT MANAGEMENT AND GEOMETRY MANAGERS:

Layout management:

Η διαχείριση διάταξης (Layout management) στο Tkinter περιλαμβάνει τον καθορισμό του μεγέθους και της θέσης των widgets. Η ορθή διαχείριση διάταξης εξασφαλίζει ότι το γραφικό περιβάλλον είναι ευέλικτο σε διαφορετικά μεγέθη και αναλύσεις παραθύρων. Επίσης, βοηθάει το χρήστη στο οπτικό κομμάτι και είναι ιδιαίτερα οικείο προς αυτόν. Βασικές αρχές της διάταξης είναι:

Δομή της διεπαφής χρήστη: Η διαχείριση διάταξης αποτελείται από την οργάνωση των widgets και εικόνες με συνεπή και λογικό τρόπο

Δυνατότητα και ευελιξία: Η καλή διαχείριση διάταξης εγγυάται ότι το γραφικό περιβάλλον χρήστη είναι προσαρμόσιμο σε διαφορετικά μεγέθη και αναλύσεις οθόνης

Αισθητική: Η διαχείριση της διάταξης όταν είναι ορθά δομημένη στο οπτικό κομμάτι κάνει μία εφαρμογή πιο οικεία και απλή στο χρήστη

Αμεσότητα: Με την σωστή διαχείριση διάταξης επιτυγχάνεται η μείωση κόπου από τη πλευρά του χρήστη προκειμένου να την κατανοήσει και βελτιώνεται η συνολική του εμπειρία

Τεχνικές διαχείρισης διάταξης είναι:

Ακριβή θέση: Η τοποθέτηση των widgets γίνεται σε συγκεκριμένες συντεταγμένες σε ένα παράθυρο, το μέγεθος του παραθύρου δεν μπορεί να μεταβληθεί

Διάταξη ροής: Τα widgets είναι τοποθετημένα σε μία σειρά παρόμοια με αυτή του κειμένου, το μέγεθος του παραθύρου μπορεί να τροποποιηθεί

Διάταξη πλέγματος: Η οργάνωση των widgets σε μοτίβο πλέγματος δείχνει την ακριβή θέση, το μέγεθος του παραθύρου μεταβάλλεται

Geometry Manager:

Ο διαχειριστής γεωμετρίας(**Geometry Manager**) αποτελεί τον υπεύθυνο της χωρικής διάταξης(τοποθέτηση και μέγεθος) των **widgets** και με τη χρήση του δεν είναι απαραίτητο να καθοριστούν οι ακριβείς συντεταγμένες για κάθε **widget**. Υπάρχουν 3 διαχειριστές γεωμετρίας(**pack,grid,place**).

Pack: Το **Pack** βάζει αυτόματα τα **widgets**, είτε οριζόντια είτε κάθετα. Χρησιμοποιείται σε απλές και γραμμικές διατάξεις, τα **widgets** τοποθετούνται με τη σειρά που εισάγονται. Κάποιες ιδιότητες που έχει είναι το **side**, με το οποίο καθορίζεται από ποια πλευρά θα προστεθεί το **widget**(πάνω, κάτω, αριστερά, δεξιά), **fill**, με το οποίο καθορίζεται αν ένα **widget** θα χρειαστεί επιπλέον χώρο και αν αυτός ο χώρος θα βρίσκεται είτε στον οριζόντιο άξονα(**X**) είτε στον κατακόρυφο άξονα(**Y**) είτε και στους δύο, **expand**, όταν δεχθεί την τιμή **TRUE** δίνει τη δυνατότητα στο **widget** να αποκτήσει περισσότερο χώρο στο κύριο παράθυρο πέρα από το δηλωμένο μέγεθος του. Τα πλεονεκτήματα της χρήσης του **pack** είναι η απλότητα, πράγμα που σημαίνει ότι είναι εύχρηστο για τους αρχάριους ή για τη γρήγορη δημιουργία πρότυπων γραφικών διεπαφών και η ευελιξία, με την οποία μπορεί να χειριστεί μεγάλη πληθώρα διατάξεων.

Στην παρακάτω εικόνα ακολουθεί ένα παράδειγμα του **pack**:

```

1  import tkinter as tk
2
3  def on_submit():
4      entered_text = entry.get()
5      print("Entered Text:", entered_text)
6
7  # Create the main window
8  root = tk.Tk()
9  root.title("Pack Geometry Manager Example")
10
11 # Create a label widget
12 label = tk.Label(root, text="Enter something:")
13 label.pack(pady=10) # Add some vertical padding
14
15 # Create an entry widget
16 entry = tk.Entry(root)
17 entry.pack(pady=5) # Add some vertical padding
18
19 # Create a submit button
20 submit_button = tk.Button(root, text="Submit", command=on_submit)
21 submit_button.pack(pady=10) # Add some vertical padding
22
23 # Start the GUI event loop
24 root.mainloop()

```

Εικόνα 2.3.α. Παράδειγμα χρήσης του **pack geometry manager**

Grid: Το **Grid** τοποθετεί τα **widgets** σε μία διάταξη που μοιάζει με πίνακα και πλέγμα. Ο έλεγχος που πραγματοποιεί είναι αυστηρότερος σε σχέση με το **pack**, χρησιμοποιείται σε σύνθετες εφαρμογές γραφικών διεπαφών που απαιτείται ακριβή τοποθέτηση των **widgets**, όπως δημιουργία φορμών, αριθμομηχανών και άλλων εφαρμογών που απαιτούν μία πιο δομημένη διάταξη. Η διάταξη του γίνεται με έναν δισδιάστατο πίνακα που αποτελείται από γραμμές και στήλες. Κάθε **widget** μπορεί να τοποθετηθεί σε ένα συγκεκριμένο "κελί" αυτού του πίνακα. Τα **widgets** τοποθετούνται καθορίζοντας τους δείκτες των γραμμών και των στηλών τους. Η πάνω αριστερή γωνία του πλέγματος είναι (**row=0, column=0**). Με τις εντολές **rowspan** και **columnspan** δίνεται η δυνατότητα να χρησιμοποιούνται πολλές γραμμές και στήλες καθιστώντας τη διάταξη πιο ευέλικτη. Επίσης, με τις εντολές **rowconfigure** και **columnconfigure** μπορεί να προσαρμοστεί το μέγεθος κάθε κελιού, το οποίο στην αρχή λαμβάνει τη χωρητικότητα που μπορεί να λάβει το μεγαλύτερο **widget**. Τα πλεονεκτήματα χρήσης του είναι η ακρίβεια, που σημαίνει ακριβής έλεγχος της τοποθέτησης των **widgets**, ευελιξία, διαθέτει τη δυνατότητα υποστήριξης σε σύνθετες διατάξεις και ευκολία προσαρμογών, προσθήκη ή αφαίρεση ενός **widget** χωρίς να επηρεάζεται η υπόλοιπη διάταξη.

Στην παρακάτω εικόνα ακολουθεί ένα παράδειγμα του **grid**:

```

1  import tkinter as tk
2
3  # Create the main window
4  root = tk.Tk()
5  root.title("Grid Example")
6
7  # Create and position labels and entry widgets using grid
8  label1 = tk.Label(root, text="Label 1")
9  label1.grid(row=0, column=0)
10
11  entry1 = tk.Entry(root)
12  entry1.grid(row=0, column=1)
13
14  label2 = tk.Label(root, text="Label 2")
15  label2.grid(row=1, column=0)
16
17  entry2 = tk.Entry(root)
18  entry2.grid(row=1, column=1)
19
20 # Start the Tkinter event loop
21 root.mainloop()

```

Εικόνα 2.3.β. Παράδειγμα χρήσης του **grid geometry manager**

Place: Ο διαχειριστής γεωμετρίας θέσης (**Place**) στο **Tkinter** αποτελεί το ισχυρότερο εργαλείο παροχής ελέγχου ακριβούς τοποθέτησης των **widgets** μέσα σε ένα δοχείο. Σε αντίθεση με τους διαχειριστές **pack** και **grid**, οι οποίοι χειρίζονται την τοποθέτηση των **widget** αυτόματα με βάση ένα σύνολο κανόνων, το **place** καθορίζει το ακριβές πλάτος και ύψος των **widgets** δίνοντας τον απόλυτο έλεγχο του μεγέθους. Το **anchor** δίνει από ποιο σημείο του **widget** μετρούνται οι συντεταγμένες **x** και **y** (π.χ. **N**, **E**, **S**, **W**, **CENTER**, κ.λπ.). Χρησιμοποιείται κυρίως σε εφαρμογές όπως προσαρμοσμένες διατάξεις, επικάλυψη **widgets**. Τα πλεονεκτήματα χρήσης του είναι: ακρίβεια, καθώς προσφέρει έλεγχο με ακρίβεια εικονοστοιχείου για την θέση των **widgets** και ευελιξία, επειδή επιτρέπει την επικάλυψη **widgets** και προσαρμοσμένα μοτίβα διάταξης που δεν είναι δυνατά με το **pack** ή το **grid**. Στην παρακάτω εικόνα ακολουθεί ένα παράδειγμα του **Place**:

```
1 import tkinter as tk
2
3 # Create the main window
4 root = tk.Tk()
5 root.title("Place Geometry Example")
6 root.geometry("300x200") # Set the size of the window
7
8 # Create a label widget
9 label1 = tk.Label(root, text="Label at (50, 50)")
10 label1.place(x=50, y=50)
11
12 # Create a button widget
13 button1 = tk.Button(root, text="Button at (100, 100)")
14 button1.place(x=100, y=100)
15
16 # Start the Tkinter event loop
17 root.mainloop()
```

Εικόνα 2.3.γ. Παράδειγμα χρήσης του **place geometry manager**

ΚΕΦΑΛΑΙΟ 3 PYQT

Το **Pyqt** αποτελεί μία βιβλιοθήκη της γλώσσας προγραμματισμού **Python** που χρησιμοποιείται για τη δημιουργία εφαρμογών **Qt**, που επιτρέπει στους προγραμματιστές **Python** να δημιουργούν πλούσιες σε χαρακτηριστικά και διαπλατφορμικές εφαρμογές γραφικής διεπαφής χρήστη (**GUI**). Το **Pyqt** είναι ολοκληρωμένο πλαίσιο για την ανάπτυξη εφαρμογών **GUI** στην **Python**, συνδυάζοντας την ευκολία και την αναγνωσιμότητα της **Python** με τη δύναμη και την πολυπλοκότητα του πλαισίου **Qt**. Διατίθεται σε δύο μεγάλες εκδόσεις την - **PyQt5**, η οποία είναι συνδεδεμένη με την **Qt5**, και η νεότερη **PyQt6**, η οποία είναι συνδεδεμένη με την **Qt6**. Αυτές οι εκδόσεις αντικατοπτρίζουν τις δυνατότητες και τις ενημερώσεις του **Qt**. Οι εφαρμογές **PyQt** είναι εγγενώς διαπλατφορμικές, που σημαίνει ότι μπορούν να τρέξουν σε διάφορα λειτουργικά συστήματα, όπως τα **Windows**, το **macOS** και το **Linux**, χωρίς να απαιτούνται τροποποιήσεις για συγκεκριμένες πλατφόρμες. Το **Pyqt** είναι χρήσιμο για διαφορετικές εφαρμογές, όπως για παράδειγμα για εφαρμογές γραφείου, εμπορικό λογισμικό, επιστημονικές και τεχνικές εφαρμογές. Επίσης, χρειάζεται να γίνει η εγκατάσταση του ξεχωριστά καθώς δεν έρχεται προεγκατεστημένο σε κάποια μορφή της **Python**. Επιπλέον, παρέχει ένα ευρύ φάσμα από **widgets** που μπορούν να χρησιμοποιηθούν για τη δημιουργία εξελιγμένων και πλούσιων σε χαρακτηριστικά διεπαφών χρήστη. Τα βασικότερα **widgets** είναι:

- **QPushButton**: Κουμπί για να εμφανιστεί μία εικόνα, ένα κείμενο
- **QLabel**: Εμφάνιση κειμένου ή εικόνων ή φόρμες για την περιγραφή πεδίων εισαγωγής
- **QLineEdit**: Επεξεργασία μίας γραμμής κειμένου
- **QTextEdit**: Επεξεργασία κειμένου πολλών γραμμών
- **QComboBox**: Συνδυασμός μιας αναπτυσσόμενης λίστας και ενός επεξεργάσιμου πεδίου κειμένου
- **QCheckBox**: Εμφάνιση ενός πλαισίου ελέγχου με μια συνοδευτική ετικέτα
- **QRadioButton**: Επιλογή μέσα από ένα σύνολο
- **QSlider**: Επιλογή μίας τιμής ολισθαίνοντας μία λαβή σε μία διαδρομή
- **QProgressBar**: Εμφάνιση της προόδου μιας μακροχρόνιας εργασίας και
- **QTreeWidget**: Παρουσίαση στοιχείων σε μια ιεραρχική δεντρική δομή
- **QTableWidget**: Εμφάνιση και επεξεργασία δεδομένων σε μορφή πίνακα.
- **QTabWidget**: Διαχείριση πολλών widgets σε μια διεπαφή με καρτέλες
- **QDialog**: Υπεύθυνο για παράθυρα διαλόγου

WIDGETS:

Widget QPushButton

Το widget **QPushButton** αποτελεί ένα widget κουμπιού το οποίο συνδυάζει τις αρχές του σχεδιασμού με τις αρχές της ανάπτυξης γραφικών διεπαφών (**GUI**). Το widget αυτό επιδεικνύει τον προγραμματισμό με γνώμονα τα γεγονότα, τον

αντικειμενοστραφή σχεδιασμό, τους μηχανισμούς ανατροφοδότησης του χρήστη, της εκτίμησης προσβασιμότητας και την ενσωμάτωση στο ευρύτερο πλαίσιο μιας εφαρμογής GUI. Το **QPushButton** μπορεί να εμφανίζει απλό κείμενο, μια εικόνα ή και τα δύο και εκπέμπει το σήμα **clicked()** όταν πατηθεί. Το μέγεθος του, η εικόνα του, το χρώμα του, η γραμματοσειρά και το εικονίδιο του μπορούν να μεταβληθούν. Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
1  import sys
2  from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
3
4  def button_clicked():
5      print("Button clicked!")
6
7  app = QApplication(sys.argv)
8
9  # Create a main window widget
10 window = QWidget()
11 window.setWindowTitle('QPushButton Example')
12
13 # Create a QPushButton
14 button = QPushButton('Click Me')
15 button.clicked.connect(button_clicked) # Connect the clicked signal to a function
16
17 # Layout to hold the button
18 layout = QVBoxLayout()
19 layout.addWidget(button)
20 window.setLayout(layout)
21
22 # Show the window
23 window.show()
24
25 # Run the application
26 sys.exit(app.exec_())
```

Εικόνα 3.1.α Παράδειγμα χρήσης του **QPushButtonwidget**

Widget QLabel

Το **widget QLabel** είναι ένα ευέλικτο στοιχείο που χρησιμοποιείται για την εμφάνιση κειμένου και εικόνων σε μια εφαρμογή ανάπτυξης γραφικών διεπαφών(GUI). Πιο συγκεκριμένα το **QLabel** μπορεί να εμφανίζει μία μόνο γραμμή απλού κειμένου ή πολλαπλές γραμμές κειμένου. Υποστηρίζει επίσης τη μορφοποίηση πλούσιου κειμένου χρησιμοποιώντας ένα υποσύνολο της **HTML**, το οποίο επιτρέπει διάφορα στυλ και διατάξεις κειμένου. Η δυνατότητα εμφάνισης εικόνων της **QLabel**, συμβάλλει στην πολυτροπική πτυχή των **GUI**, συνδυάζοντας κείμενο και γραφικά για μια πλουσιότερη εμπειρία χρήστη. Επιπλέον, το **QLabel** περιλαμβάνει την προσβασιμότητα, όπως η κλιμάκωση κειμένου και η συμβατότητα με προγράμματα ανάγνωσης οθόνης, διασφαλίζοντας ότι το **GUI** είναι προσβάσιμο σε χρήστες με αναπηρίες. Όταν στο **QLabel**

ενσωματώνεται με το **Event-DrivenModel** της **PyQt** έχουμε χειρισμό σημάτων και χειρισμό συμβάντων και προσαρμοσμένες υποδοχές.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout

app = QApplication(sys.argv)

# Create the main window
window = QWidget()
window.setWindowTitle('QLabel Example')

# Create QLabel
label = QLabel('Hello, PyQt5!')
label.setAlignment(Qt.AlignCenter) # Align the text to the center

# Layout to hold the widgets
layout = QVBoxLayout()
layout.addWidget(label)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())
```

Εικόνα 3.1.β Παράδειγμα χρήσης του **QLabel widget**

Widget **QLineEdit**

Το **widget QLineEdit** στο **PyQt** χρησιμοποιείται για την επεξεργασία κειμένου μιας γραμμής, για φόρμες, ρυθμίσεις και εισαγωγή διαλόγων στις εφαρμογές ανάπτυξης γραφικών διεπαφών(**GUI**). Πιο συγκεκριμένα, το **QLineEdit** είναι υπεύθυνο για την εισαγωγή χρήστη και συλλογή δεδομένων, καθώς αποτελεί ένα μηχανισμό για την είσοδο του χρήστη και τη συλλογή δεδομένων, αντιπροσωπεύει ένα θεμελιώδες στοιχείο αλληλεπίδρασης, που οι χρήστες μπορούν να εισάγουν ή να επεξεργαστούν μία μόνο γραμμή κειμένου πληροφορίας. Επίσης, είναι υπεύθυνο για την επικύρωση και τον έλεγχο κειμένου, διασφαλίζει ότι η εισαγωγή συμμορφώνεται με συγκεκριμένες απαιτήσεις, όταν πρόκειται για φόρμες και εισαγωγή δεδομένων. Τέλος, σημαντικά του στοιχεία είναι η ευχρηστία του και η προσβασιμότητα του, σε κείμενο υπό μορφή **placeholder**, παρέχει στους χρήστες καθοδήγηση ή υποδείξεις σχετικά με την αναμενόμενη εισαγωγή, και οι λειτουργίες **echo**, όπως η απόκρυψη κωδικού πρόσβασης, εξυπηρετούν τις ανησυχίες για την ιδιωτικότητα και την ασφάλεια.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLineEdit, QVBoxLayout

def on_return_pressed():
    entered_text = line_edit.text()
    print("Entered Text:", entered_text)

app = QApplication(sys.argv)

# Create the main window
window = QWidget()
window.setWindowTitle('QLineEdit Example')

# Create QLineEdit
line_edit = QLineEdit()
line_edit.setPlaceholderText("Enter some text here and press enter")
line_edit.returnPressed.connect(on_return_pressed)

# Layout to hold the widgets
layout = QVBoxLayout()
layout.addWidget(line_edit)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.γ Παράδειγμα χρήσης του **QLineEdit** widget

Widget QTextEdit

Το widget **QTextEdit** στο **PyQt** χρησιμοποιείται για την εμφάνιση και την επεξεργασία τόσο απλού όσο και σύνθετου κειμένου, είναι ιδιαίτερα χρήσιμο σε εφαρμογές που απαιτείται η εισαγωγή ή η εμφάνιση κειμένου πολλών γραμμών, όπως επεξεργαστές κειμένου ή οποιαδήποτε εφαρμογή που απαιτεί μια περιοχή κειμένου. Το **QTextEdit** μπορεί να χειριστεί τη μορφή σύνθετου κειμένου, η οποία περιλαμβάνει διάφορες γραμματοσειρές, χρώματα, λίστες και εικόνες. Βασικά χαρακτηριστικά και ιδιότητες του είναι:

- Μορφοποίηση κειμένου: πολυάριθμες επιλογές μορφοποίησης
- Υποστήριξη HTML: μπορεί να εμφανίζει και να επεξεργάζεται κείμενο διαμορφωμένο με τη χρήση ενός υποσυνόλου της HTML, επιτρέποντας τη δημιουργία πλούσια διαμορφωμένων εγγράφων κειμένου.
- Κύλιση: Υποστηρίζεται αυτόματη κύλιση
- Λειτουργίες πρόχειρου: Υποστηρίζει τυπικές λειτουργίες πρόχειρου, όπως αποκοπή, αντιγραφή και επικόλληση
- Αναίρεση/επανάληψη: Διατίθενται ενσωματωμένες λειτουργίες αναίρεσης και επανάληψης

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QTextEdit, QVBoxLayout, QPushButton

def print_text():
    entered_text = text_edit.toPlainText()
    print("Entered Text:", entered_text)

app = QApplication(sys.argv)

# Create the main window
window = QWidget()
window.setWindowTitle('QTextEdit Example')

# Create QTextEdit
text_edit = QTextEdit()
text_edit.setPlaceholderText("Enter your text here...")

# Create QPushButton
button = QPushButton('Print Text')
button.clicked.connect(print_text) # Connect the clicked signal to print_text function

# Layout to hold the widgets
layout = QVBoxLayout()
layout.addWidget(text_edit)
layout.addWidget(button)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.δ Παράδειγμα χρήσης του **QTextEdit widget**

Widget QComboBox

Το **widget QComboBox** στο **PyQt** χρησιμοποιείται προκειμένου να δίνεται η δυνατότητα στους χρήστες να επιλέξουν από μια λίστα επιλογών ή να εισάγουν μια προσαρμοσμένη τιμή στη λίστα, εάν είναι επεξεργάσιμη και αποτελείται μία λίστα και ένα κουμπί, συνήθως χρησιμοποιείται από απλές φόρμες μέχρι σύνθετες εφαρμογές που βασίζονται σε δεδομένα. Τα βασικά χαρακτηριστικά του **QComboBox** είναι:

- διαχείριση στοιχείων: Μπορεί να γίνει προσθήκη ή αφαίρεση από το πλαίσιο συνδυασμού και κάθε στοιχείο μπορεί να αποτελεί είτε μια απλή συμβολοσειρά κειμένου είτε πιο σύνθετα **widgets**
- Χειρισμός επιλογής: Το **widget** εντοπίζει το επιλεγμένο στοιχείο και μπορεί να εκπέμπει σήματα όταν αλλάζει η επιλογή
- Προσαρμογή: Η εμφάνιση του κουμπιού και της αναπτυσσόμενης λίστας μπορεί να τροποποιηθεί

- Δέσμευση δεδομένων: Κάθε στοιχείο της λίστας μπορεί να χρησιμοποιηθεί με αυθαίρετα δεδομένα Python (χρησιμοποιώντας την παράμετρο **userData**), επιτρέποντας πιο σύνθετα σενάρια χειρισμού δεδομένων.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QComboBox, QVBoxLayout

def selection_changed(i):
    print(f"Selected item index: {i}; Value: {combo_box.itemText(i)}")

app = QApplication(sys.argv)

# Create the main window
window = QWidget()
window.setWindowTitle('QComboBox Example')

# Create QComboBox
combo_box = QComboBox()
combo_box.addItem("Option 1")
combo_box.addItem("Option 2")
combo_box.addItem("Option 3")
combo_box.currentIndexChanged.connect(selection_changed) # Connect signal

# Layout to hold the widgets
layout = QVBoxLayout()
layout.addWidget(combo_box)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())
```

Εικόνα 3.1.ε Παράδειγμα χρήσης του **QComboBox** widget

WidgetQCheckBox

Το widget **QCheckBox** στο **PyQt** χρησιμοποιείται για την αναπαράσταση μιας επιλογής που ο χρήστης μπορεί να ενεργοποιήσει ή να απενεργοποιήσει. Εμφανίζει ένα μικρό πλαίσιο που μπορεί να είναι είτε τσεκαρισμένο (επιλογή επιλεγμένη) είτε μη τσεκαρισμένο (επιλογή μη επιλεγμένη), σε εφαρμογές όπως φόρμες και διαλόγους ρυθμίσεων. Κάθε **checkbox** συνήθως συνοδεύεται από μια ετικέτα που υποδεικνύει τι αντιπροσωπεύει η επιλογή. Τα βασικά χαρακτηριστικά του **QCheckbox** είναι:

- Κατάσταση: Η κατάσταση ενός **QCheckBox** μπορεί να είναι : **unchecked**, **checked**, ή **partially checked**, το οποίο χρησιμοποιείται για να αναπαραστήσει μία απροσδιόριστη κατάσταση(όπως ιεραρχικές λίστες ελέγχου).

- Σήματα: Το πιο συχνά χρησιμοποιούμενο σήμα είναι το **stateChanged**, το οποίο εκπέμπεται κάθε φορά που το **checkbox** είναι επιλεγμένο ή μη επιλεγμένο.
- Προσαρμογή: Η εμφάνιση του πλαισίου ελέγχου μπορεί να τροποποιηθεί σε μέρη όπως το μέγεθος του πλαισίου, το χρώμα όταν είναι επιλεγμένο

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QCheckBox, QVBoxLayout

def checkbox_state_changed():
    if checkbox.isChecked():
        print("Checkbox is checked!")
    else:
        print("Checkbox is unchecked!")

app = QApplication(sys.argv)

# Create the main window
window = QWidget()
window.setWindowTitle('QCheckBox Example')

# Create QCheckBox
checkbox = QCheckBox('Check me')
checkbox.stateChanged.connect(checkbox_state_changed) # Connect signal

# Layout to hold the widget
layout = QVBoxLayout()
layout.addWidget(checkbox)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())
```

Εικόνα 3.1.στ Παράδειγμα χρήσης του **QCheckbox widget**

Widget QRadioButton

Το widget **QRadioButton** στην **PyQt** χρησιμοποιείται για την παρουσίαση επιλογών που ο χρήστης καλείται να επιλέξει μόνο μία από ένα σύνολο επιλογών. Τα **QRadiobuttons** ομαδοποιούνται όταν επιλέγεται ένα από αυτά, αυτόματα καταργεί κάθε προηγουμένως επιλεγμένο κουμπί στην ομάδα, αυτό αποτελεί ένα από τα βασικά χαρακτηριστικά του. Κάθε κουμπί επιλογής έχει μια ετικέτα δίπλα στο κυκλικό κουμπί που υποδεικνύει τι αντιπροσωπεύει η επιλογή. Επίσης, βασικά χαρακτηριστικά και ιδιότητες του **QRadiobutton** αποτελεί η κατάσταση του, η οποία μπορεί να είναι είτε ενεργοποιημένη είτε απενεργοποιημένη. Το **toggled** αποτελεί το σημαντικότερο σήμα και εκπέμπεται κάθε φορά που αλλάζει η επιλεγμένη κατάσταση του **QRadiobutton**. Η εμφάνιση του κουμπιού επιλογής μπορεί να τροποποιηθεί, μέγεθος, χρώμα.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QRadioButton, QVBoxLayout

def radio_button_selected():
    for radio_button in radio_buttons:
        if radio_button.isChecked():
            print(f"Selected: {radio_button.text()}")

app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('QRadioButton Example')
# Create a list to store radio buttons
radio_buttons = []
# Create and add radio buttons
for i in range(1, 4):
    radio_button = QRadioButton(f'Option {i}')
    radio_button.toggled.connect(radio_button_selected) # Connect signal
    radio_buttons.append(radio_button)
# Layout to hold the widgets
layout = QVBoxLayout()
for radio_button in radio_buttons:
    layout.addWidget(radio_button)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())
```

Εικόνα 3.1.η Παράδειγμα χρήσης του **QRadiobutton widget**

Widget QSlider

Το **widget QSlider** στο **PyQt** χρησιμοποιείται για την επιλογή μιας τιμής από ένα εύρος με τη συρόμενη λαβή κατά μήκος μιας διαδρομής, εμφανίζεται σε εφαρμογές όπως η ρύθμιση έντασης, φωτεινότητας, η επιλογή μιας τιμής ή ο ορισμός ενός εύρους. Το **QSlider** μπορεί να προσανατολιστεί οριζόντια ή κάθετα, ανάλογα με τη διάταξη και τις απαιτήσεις της εφαρμογής. Το εύρος του ολισθητήρα (ελάχιστη και μέγιστη τιμή) και η τρέχουσα τιμή μπορούν να οριστούν προγραμματιστικά. Η θέση του ρυθμιστικού αντικατοπτρίζει την τρέχουσα τιμή. Το **QSlider** μπορεί να εμφανίζει σημάδια τικ κατά μήκος της διαδρομής του ολισθητή, παρέχοντας οπτικές ενδείξεις της κλίμακας ή των διαιρέσεων του εύρους. Το σήμα **valueChanged** εκπέμπεται κάθε φορά που αλλάζει η τιμή του ολισθητήρα, είτε από την αλληλεπίδραση του χρήστη είτε προγραμματιστικά.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QSlider, QLabel, QVBoxLayout
from PyQt5.QtCore import Qt
def slider_value_changed():
    label.setText(f"Slider Value: {slider.value()}")

app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('QSlider Example')
# Create QSlider
slider = QSlider(Qt.Horizontal)
slider.setMinimum(0)
slider.setMaximum(100)
slider.valueChanged.connect(slider_value_changed) # Connect signal
# Create QLabel
label = QLabel("Slider Value: 0")
label.setAlignment(Qt.AlignCenter)
# Layout to hold the widgets
layout = QVBoxLayout()
layout.addWidget(slider)
layout.addWidget(label)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.θ Παράδειγμα χρήσης του **QSlider widget**

Widget QProgressBar

Το widget **QProgressBar** στο **PyQt** χρησιμοποιείται για να δώσει οπτική ανατροφοδότηση σχετικά με την πρόοδο μιας εργασίας που εκτελείται για μεγάλο χρονικό διάστημα, όπως μεταφορτώσεις αρχείων, λήψεις, υπολογισμοί ή οποιαδήποτε εργασία που έχει ένα καθορισμένο σημείο έναρξης και λήξης. Το **QProgressBar** εμφανίζει την πρόοδο μιας λειτουργίας ως μπάρα που γεμίζει από το σημείο έναρξης έως το σημείο λήξης. Η τρέχουσα τιμή προόδου μπορεί να ενημερωθεί προγραμματιστικά εντός αυτού του εύρους και η γραμμή προόδου μπορεί να είναι είτε οριζόντια είτε κατακόρυφη, ανάλογα με τη διάταξη και τις απαιτήσεις της εφαρμογής. Επίσης, η εμφάνισή του μπορεί να τροποποιηθεί επιτρέποντας την αλλαγή σε πτυχές όπως το χρώμα της μπάρας προόδου, το περίγραμμα.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QProgressBar, QVBoxLayout
from PyQt5.QtCore import QTimer
def update_progress_bar():
    current_value = progress_bar.value()
    max_value = progress_bar.maximum()
    if current_value < max_value:
        progress_bar.setValue(current_value + 1)
    else:
        timer.stop()
app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('Basic QProgressBar Example')
# Create QProgressBar
progress_bar = QProgressBar()
progress_bar.setMaximum(100)
# Set up a timer to update the progress bar
timer = QTimer()
timer.timeout.connect(update_progress_bar)
timer.start(100) # Update every 100 milliseconds
# Layout to hold the widget
layout = QVBoxLayout()
layout.addWidget(progress_bar)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.ι Παράδειγμα χρήσης του **QProgressBar widget**

Widget QTreeWidget

Το **widget QTreeWidget** στο **PyQt** χρησιμοποιείται για την εμφάνιση δεδομένων σε μια ιεραρχική δομή δέντρου, όπως εξερευνητές αρχείων, παράθυρα διαλόγου ρυθμίσεων ή οποιαδήποτε εφαρμογή απαιτεί μια δεντρική αναπαράσταση δεδομένων. Το **QTreeWidget** υποστηρίζει την επιλογή ενός ή πολλαπλών στοιχείων. Μπορεί να ρυθμιστεί ώστε να υποστηρίζει τη λειτουργία σύρσιμο, επιτρέποντας στους χρήστες να αναδιατάξουν στοιχεία μέσα στο δέντρο ή να τα μετακινήσουν μεταξύ διαφορετικών δέντρων ή widgets. Επίσης, το **QTreeWidget** παρέχει σήματα όπως **itemClicked**, **itemExpanded**, **itemCollapsed**, τα οποία μπορούν να χρησιμοποιηθούν για την ενεργοποίηση ενεργειών όταν ο χρήστης αλληλεπιδρά με το δέντρο. Οι στήλες του δέντρου μπορούν να αλλάξουν μέγεθος, να κρυφτούν ή να ταξινομηθούν, ενώ η εμφάνιση των στοιχείων και του ίδιου του δέντρου μπορεί να τροποποιηθεί.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QTreeWidget, QTreeWidgetItem, QVBoxLayout
app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('QTreeWidget Example')
# Create QTreeWidget
tree_widget = QTreeWidget()
tree_widget.setHeaderLabels(["Column 1", "Column 2"])
# Add items to QTreeWidget
root_item = QTreeWidgetItem(tree_widget, ["Root Item", "Root Data"])
child_item1 = QTreeWidgetItem(root_item, ["Child 1", "Data 1"])
child_item2 = QTreeWidgetItem(root_item, ["Child 2", "Data 2"])
# Expand the root item to show children
root_item.setExpanded(True)
# Layout to hold the widget
layout = QVBoxLayout()
layout.addWidget(tree_widget)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.κΠαράδειγμα χρήσης του **QTreeWidget**

Widget **QTableWidget**

Το widget **QTableWidget** στο **PyQt** χρησιμοποιείται για την εμφάνιση και τον χειρισμό δεδομένων σε μορφή πίνακα, είναι χρήσιμο σε εφαρμογές που απαιτείται μορφή πίνακα για την εμφάνιση συνόλου δεδομένων με πολλές στήλες και γραμμές. Κάθε κελί στον πίνακα(γραμμής ή στήλης) μπορεί να περιέχει στοιχεία όπως κείμενο, αριθμούς ή ακόμα και **widgets**. Το κελί σε ένα **QTableWidget** αναπαρίσταται από ένα **QTableWidgetItem**, το οποίο μπορεί να προσαρμοστεί ως προς το περιεχόμενο, τη μορφοποίηση και τη συμπεριφορά. Το **QTableWidget** επιτρέπει την προσθήκη, την αφαίρεση και την τροποποίηση γραμμών και στηλών. Επιτρέπει την επιλογή ενός, πολλών ή καθόλου στοιχείων. Επίσης, οι επικεφαλίδες γραμμών όσο και οι επικεφαλίδες στηλών μπορούν να προσαρμοστούν και να χρησιμοποιηθούν για την εμφάνιση σχετικών ετικετών για τα δεδομένα.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QTableWidgetItem, QVBoxLayout
app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('QTableWidget Example')
# Create QTableWidget
table_widget = QTableWidgetItem()
table_widget.setRowCount(3) # Set 3 rows
table_widget.setColumnCount(2) # Set 2 columns
# Set header labels
table_widget.setHorizontalHeaderLabels(["Column 1", "Column 2"])
# Add items to the table
table_widget.setItem(0, 0, QTableWidgetItem("Row 1, Col 1"))
table_widget.setItem(0, 1, QTableWidgetItem("Row 1, Col 2"))
table_widget.setItem(1, 0, QTableWidgetItem("Row 2, Col 1"))
table_widget.setItem(1, 1, QTableWidgetItem("Row 2, Col 2"))
table_widget.setItem(2, 0, QTableWidgetItem("Row 3, Col 1"))
table_widget.setItem(2, 1, QTableWidgetItem("Row 3, Col 2"))
# Layout to hold the widget
layout = QVBoxLayout()
layout.addWidget(table_widget)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1. Παράδειγμα χρήσης του **QTableWidget**

Widget QTabWidget

Το **QTabWidget** στο **PyQt** χρησιμοποιείται για την οργάνωση περιεχομένου σε καρτέλες. Το **QTabWidget** παρέχει μια διεπαφή με καρτέλες, όπου κάθε καρτέλα λειτουργεί ως υποδοχή για άλλα **widgets**. Αυτή η δομή είναι ιδανική για την οργάνωση περιεχομένου σε ξεχωριστές, εναλλασσόμενες προβολές μέσα στο ίδιο παράθυρο. Το **QTabWidget** επιτρέπει στους χρήστες να πλοηγούνται εύκολα μεταξύ διαφορετικών τμημάτων ή κατηγοριών περιεχομένου. Βασικά χαρακτηριστικά του αποτελούν:

- προσθήκη και αφαίρεση καρτελών: Οι καρτέλες μπορούν να προστίθενται ή να αφαιρούνται δυναμικά από το widget καρτελών
- προσαρμόσιμες καρτέλες: Κάθε καρτέλα μπορεί να προσαρμοστεί με τη δική της ετικέτα, εικονίδιο, παρέχοντας ευελιξία στον τρόπο παρουσίασης των πληροφοριών
- Χωροταξία καρτελών: Οι καρτέλες μπορούν να τοποθετηθούν στην κορυφή, το κάτω μέρος, την αριστερή ή τη δεξιά πλευρά της περιοχής περιεχομένου, προσφέροντας ευελιξία στο σχεδιασμό της διάταξης
- Χειρισμός συμβάντων: Το **QTabWidget** εκπέμπει σήματα όταν ο χρήστης αλλάζει καρτέλες, επιτρέποντας στην εφαρμογή να ανταποκρίνεται στις αλλαγές καρτελών

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QTabWidget, QLabel, QVBoxLayout
app = QApplication(sys.argv)
# Create the main window
window = QWidget()
window.setWindowTitle('QTabWidget Example')
# Create QTabWidget
tab_widget = QTabWidget()
# Create the first tab
first_tab = QWidget()
first_tab_layout = QVBoxLayout()
first_tab_layout.addWidget(QLabel("Content of the first tab"))
first_tab.setLayout(first_tab_layout)
tab_widget.addTab(first_tab, "Tab 1")
# Create the second tab
second_tab = QWidget()
second_tab_layout = QVBoxLayout()
second_tab_layout.addWidget(QLabel("Content of the second tab"))
second_tab.setLayout(second_tab_layout)
tab_widget.addTab(second_tab, "Tab 2")
# Layout to hold the widget
layout = QVBoxLayout()
layout.addWidget(tab_widget)
window.setLayout(layout)
# Show the window
window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.μ Παράδειγμα χρήσης του **QTabWidget**

Widget QDialog

Το widget **QDialog** στο **PyQt** χρησιμοποιείται για τη δημιουργία παραθύρων διαλόγου μέσα σε εφαρμογές. Το **QDialog** είναι ένας εξειδικευμένος τύπος παραθύρου, που χρησιμοποιείται για βραχυπρόθεσμες εργασίες και σύντομες επικοινωνίες με τον χρήστη. Τα παράθυρα διαλόγου χρησιμοποιούνται γενικά για να ζητούν πληροφορίες από τον χρήστη, να παρουσιάζουν πληροφορίες ή να ζητούν αποφάσεις. Τα παράθυρα διαλόγου μπορεί να είναι **modal**, ο χρήστης πρέπει να απαντήσει πριν επιστρέψει στη μητρική εφαρμογή, ή **modeless**, που επιτρέπει στο χρήστη να αλληλεπιδράσει με άλλα παράθυρα ενώ το παράθυρο διαλόγου είναι ανοιχτό. Το **QDialog** έχει σχεδιαστεί για να διευκολύνει την αλληλεπίδραση του χρήστη, αυτό γίνεται με κουμπιά και άλλα **widgets** όπως πεδία κειμένου, πλαίσια ελέγχου ή κουμπιά επιλογής για την είσοδο του χρήστη. Επιπλέον, μπορεί να διαμορφωθεί ώστε να επιστρέφει μια τιμή, η οποία είναι χρήσιμη για τον προσδιορισμό της αντίδρασης του χρήστη και μπορεί να διαμορφωθεί προς τη διάταξη, το περιεχόμενο και την εμφάνιση.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```

import sys
from PyQt5.QtWidgets import QApplication, QWidget, QDialog, QVBoxLayout, QPushButton
class CustomDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Custom Dialog")
        layout = QVBoxLayout()
        layout.addWidget(QPushButton("OK", self))
        self.setLayout(layout)
    def show_dialog():
        dialog = CustomDialog()
        dialog.exec_()
app = QApplication(sys.argv)
# Create the main window
main_window = QWidget()
main_window.setWindowTitle('QDialog Example')
main_window.setGeometry(100, 100, 200, 100)
# Create a button to open the dialog
open_dialog_button = QPushButton("Open Dialog", main_window)
open_dialog_button.clicked.connect(show_dialog)
main_window.show()
# Run the application
sys.exit(app.exec_())

```

Εικόνα 3.1.ν Παράδειγμα χρήσης του **QDialog Widget**

MODULES:

Ακόμα, η **PYQT** έχει διάφορες επιπρόσθετες ενότητες (**modules**) οι βασικότερες είναι:

- **QtCore**: Προσφέρει κλάσεις για τα βασικά συστατικά της εφαρμογής
- **QtGui**: Περιέχει κλάσεις για ενσωμάτωση παραθυρικού συστήματος
- **QtMultimedia**: Διευκολύνει το χειρισμό περιεχομένου πολυμέσων
- **QtNetwork**: Προσφέρει κλάσεις για τον προγραμματισμό δικτύων
- **QtSql**: Παρέχει κλάσεις για την ενσωμάτωση βάσεων δεδομένων με χρήση **SQL**
- **QtTest**: Περιέχει συναρτήσεις που διευκολύνουν τον έλεγχο των εφαρμογών

QtCore

Το **module QtCore** περιλαμβάνει βασικά συστατικά της εφαρμογής, υποδομή για το χειρισμό συμβάντων και διάφορα βοηθητικά προγράμματα που είναι καθοριστικά για τη δημιουργία λειτουργικών και αποδοτικών εφαρμογών **PyQt**. Το **QtCore** χρησιμοποιείται σχεδόν σε κάθε εφαρμογή **PyQt**. Παρέχει τις θεμελιώδεις βοηθητικές λειτουργίες που απαιτούνται τόσο για απλές όσο και για σύνθετες εφαρμογές. Τα βασικά χαρακτηριστικά είναι:

- Βασικά στοιχεία εφαρμογών: **QCoreApplication**: Παρέχει τον βασικό βρόχο συμβάντων και τα βασικά χαρακτηριστικά της εφαρμογής. Είναι μια βασική κλάση

για εφαρμογές χωρίς **GUI**. **QApplication**: Επεκτείνει την **QCoreApplication** και χρησιμοποιείται για εφαρμογές με **GUI**. Διαχειρίζεται τα βασικά στοιχεία της εφαρμογής **GUI**, όπως ο χειρισμός συμβάντων και το **styling** των **widget**

- Διαχείριση συμβάντων: σε όλες τις εφαρμογές PyQt, ο βρόχος συμβάντων (**QCoreApplication.exec_()**) διαχειρίζεται τα συμβάντα και την είσοδο του χρήστη
 - Ώρα και ημερομηνία: **QDateTime**, **QDate**, **QTime**: Κλάσεις για το χειρισμό δεδομένων ημερομηνίας και ώρας
 - Λειτουργίες αρχείων :**QFile**, **QDir**, **QIODevice**: Κλάσεις για το χειρισμό λειτουργιών αρχείων και καταλόγων
 - Νήματα και συγχρονισμός: **QThread**, **QThreadPool**: Αυτές οι κλάσεις παρέχουν εργαλεία για την υλοποίηση πολυνημάτων, επιτρέποντας στις εφαρμογές να εκτελούν λειτουργίες στο παρασκήνιο χωρίς να παγώνει το γραφικό περιβάλλον
 - Χρονοδιακόπτες: **QTimer**: Χρησιμοποιείται για τον προγραμματισμό γεγονότων που θα συμβούν σε μεταγενέστερο χρόνο ή σε τακτά χρονικά διαστήματα
 - Τύποι δεδομένων και βοηθητικά προγράμματα: **QString**, **QVariant**, **QList**, **QDict**: Κλάσεις και δομές δεδομένων για τη διαχείριση και τον χειρισμό δεδομένων.
- Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

```
import sys
from PyQt5.QtCore import QTimer, QDateTime
from PyQt5.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget

class AppDemo(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
    def initUI(self):
        self.setWindowTitle("QtCore QTimer Example")
        # Create a vertical layout
        layout = QVBoxLayout()
        # Create a label to display the time
        self.label = QLabel(self)
        layout.addWidget(self.label)
        # Update the label every second using QTimer
        timer = QTimer(self)
        timer.timeout.connect(self.showTime)
        timer.start(1000) # 1000 milliseconds = 1 second
        self.setLayout(layout)
    def showTime(self):
        # Get current time
        current_time = QDateTime.currentDateTime().toString()
        # Update the label
        self.label.setText(current_time)
if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = AppDemo()
    demo.show()
    sys.exit(app.exec_())
```

Εικόνα 3.2.α Παράδειγμα χρήσης του **QtCore Module**

QtGui

Το **module QtGui** είναι ένα βασικό συστατικό του πλαισίου **PyQt**, παρέχοντας μια συλλογή κλάσεων για στοιχεία γραφικής διεπαφής χρήστη (**GUI**) που δεν περιλαμβάνουν **widgets**. Το **QtGui** παίζει βασικό ρόλο στο χειρισμό των δυνατοτήτων γραφικών και απεικόνισης, στην ενσωμάτωση παραθυρικών συστημάτων. Βασικά χαρακτηριστικά του **QtGui** αποτελούν τα γραφικά και η ζωγραφική, η κλάση **QPainter** χρησιμοποιείται για τη σχεδίαση **2D** γραφικών, η κλάση **QPixmap** για την βελτιωμένη εμφάνιση εικόνων στην οθόνη, η **QImage** για την βέλτιστη επεξεργασία εικόνων εκτός οθόνης και η **QIcon** χρησιμοποιείται για την αναπαράσταση εικονιδίων. Οι κλάσεις **QColor**, **QPalette** είναι υπεύθυνες για τη διαχείριση χρωμάτων και παλετών στα γραφικά της εφαρμογής. Επίσης, στα βασικά χαρακτηριστικά του συμπεριλαμβάνεται η διαχείριση παραθύρων και ο χειρισμός εισόδου, οι κλάσεις **QWindow**, **QScreen** χρησιμοποιούνται στη διαχείριση παραθύρων και οθονών εφαρμογών και οι κλάσεις **QKeyEvent**, **QMouseEvent**, **QWheelEvent** αναπαριστούν διάφορες εισόδους χρήστη, όπως πατήματα πληκτρολογίου, κινήσεις ποντικιού, κλικ και συμβάντα κύλισης. Τέλος, οι γραμματοσειρές, το σύρσιμο και η γεωμετρία χρησιμοποιούν τις κλάσεις **QFont**, **QFontMetrics**, **QTextLayout** για το χειρισμό γραμματοσειρών και την απόδοση κειμένου, **QDrag**, **QClipboard** υποστήριξη για λειτουργίες **drag-and-drop** και διαχείριση **clipboard** και **QPoint**, **QRect**, **QTransform** για χειρισμό γεωμετρικών μετασχηματισμών.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
import sys
from PyQt5.QtGui import QPainter, QColor, QFont
from PyQt5.QtWidgets import QWidget, QApplication
class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
    def initUI(self):
        self.setGeometry(300, 300, 280, 170)
        self.setWindowTitle('QtGui Example')
        self.show()
    def paintEvent(self, event):
        qp = QPainter()
        qp.begin(self)
        self.drawText(event, qp)
        qp.end()
    def drawText(self, event, qp):
        qp.setPen(QColor(168, 34, 3))
        qp.setFont(QFont('Decorative', 10))
        qp.drawText(event.rect(), Qt.AlignCenter, 'Hello PyQt5')
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Εικόνα 3.2.β Παράδειγμα χρήσης του **QtGui Module**

QtMultimedia

Το **module QtMultimedia** χρησιμοποιείται σε εφαρμογές που απαιτούν τον χειρισμό περιεχομένου πολυμέσων, όπως ήχο, βίντεο, ραδιόφωνο και κάμερα. Τα βασικά χαρακτηριστικά αυτού του **module** αποτελούν:

1. Αναπαραγωγή ήχου και βίντεο: **QMediaPlayer** μια κλάση για την αναπαραγωγή αρχείων ήχου και βίντεο και **QMediaPlaylist** διαχειρίζεται μια λίστα περιεχομένου πολυμέσων
2. Εγγραφή ήχου: η κλάση **QAudioRecorder** παρέχει λειτουργικότητα για εγγραφή ήχου
3. Καταγραφή βίντεο και έλεγχος κάμερας: η κλάση **QCamera** η οποία ενώνεται με το υλικό της κάμερας του συστήματος, παρέχοντας λειτουργίες για τη λήψη φωτογραφιών και βίντεο και η κλάση **QCameraInfo** που παρέχει πληροφορίες σχετικά με τις διαθέσιμες κάμερες στο σύστημα
4. Εφέ και επεξεργασία ήχου: οι κλάσεις **QAudioProbe**, **QAudioBuffer** είναι υπεύθυνες για την ανάλυση και την επεξεργασία δεδομένων ήχου και η **QSoundEffect** για την αναπαραγωγή σύντομων ηχητικών εφέ
5. Λειτουργικότητα ραδιοφώνου: **QRadioTune** με αυτή την κλάση διαχειρίζεται λειτουργίες όπως η ένταση και ο συντονισμός του ραδιοφώνου

Στις παρακάτω εικόνες υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

Στην πρώτη είναι το κύριο παράθυρο:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QMediaPlayer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
    QMediaPlayer *player;
};

#endif // MAINWINDOW_H
```

Εικόνα 3.2.γ.1 Παράδειγμα χρήσης του **QtMultimedia Module**

Στη δεύτερη είναι το **project file** που γίνεται η κλήση το **QT**:

```
# Add multimedia widgets to the project
QT += multimedia widgets

# The rest of your .pro file configuration...
```

Εικόνα 3.2.γ.2 Παράδειγμα χρήσης του **QtMultimedia Module**

Στην τρίτη είναι το παράδειγμα με χρήση των άλλων 2:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    player(new QMediaPlayer(this))
{
    ui->setupUi(this);
    #Set the audio file
    player->setMedia(QUrl::fromLocalFile("/path/to/audiofile.mp3"));
    # Set initial volume
    player->setVolume(50);
    # Connect the play button
    connect(ui->playButton, &QPushButton::clicked, this, [this]() {
        player->play();
    });
    # Connect the stop button
    connect(ui->stopButton, &QPushButton::clicked, this, [this]() {
        player->stop();
    });
    # Optional: Connect a volume slider
    connect(ui->volumeSlider, &QSlider::valueChanged, this, [this](int value) {
        player->setVolume(value);
    });
}
MainWindow::~MainWindow()
{
    delete ui;
}
```

Εικόνα 3.2.γ.3 Παράδειγμα χρήσης του **QtMultimedia Module**

QtNetwork

Το **module QtNetwork** της **PyQt** προσφέρει ένα σύνολο κλάσεων για προγραμματισμό δικτύων που επιτρέπουν στις εφαρμογές να επικοινωνούν σε δίκτυα χρησιμοποιώντας τυποποιημένα πρωτόκολλα όπως το **HTTP** και το **TCP/IP**. Το **QtNetwork** χρησιμοποιείται σε διάφορες εφαρμογές **PyQt**, όπως για παράδειγμα εφαρμογές που απαιτούν κρυπτογραφημένη επικοινωνία χρησιμοποιώντας **SSL**, προγράμματα και εργαλεία δικτύου για παρακολούθηση, διάγνωση ή διαμόρφωση δικτύου, εφαρμογές πωλητή-πελάτη που επικοινωνούν μέσω **TCP/IP**. Τα βασικά χαρακτηριστικά του είναι:

- Υποδοχές **TCP/IP** και **UDP**: Οι κλάσεις **QTcpSocket** και **QUdpSocket** είναι υπεύθυνες για τον προγραμματισμό TCP και UDP **socket**, αντίστοιχα, επιτρέποντας στις εφαρμογές να στέλνουν και να λαμβάνουν δεδομένα μέσω του δικτύου χρησιμοποιώντας αυτά τα πρωτόκολλα
- Υποστήριξη **HTTP** υψηλού επιπέδου: Η κλάση **QNetworkAccessManager** χρησιμοποιείται για την αποστολή αιτημάτων δικτύου και τη λήψη απαντήσεων. Οι κλάσεις **QNetworkRequest**, **QNetworkReply** χρησιμοποιούνται σε συνδυασμό με την παραπάνω κλάση για τη διαχείριση αιτημάτων δικτύου.

- **SSL και ασφάλεια:** Η κλάση **QSslSocket** επεκτείνει το **QTcpSocket** με υποστήριξη για κρυπτογραφημένη επικοινωνία **SSL(SecureSockets Layer)**. Οι κλάσεις **QSslCertificate** και **QSslKey** χρησιμοποιούνται για το χειρισμό πιστοποιητικών και κλειδιών **SSL**.
- **Βοηθητικά προγράμματα δικτύου:** Οι κλάσεις **QHostAddress** και **QHostInfo** είναι χρήσιμες για την αναζήτηση **DNS**, δίνουν πληροφορίες για τις διευθύνσεις **IP** και ονόματα κεντρικών υπολογιστών. Η κλάση **QNetworkInterface** παρέχει λειτουργίες για την αναζήτηση και την αλληλεπίδραση με τις δικτυακές διεπαφές του συστήματος.
- **Πρωτόκολλα δικτύου και βοηθητικά προγράμματα:** Οι κλάσεις **QUrl** και **QUrlQuery** βοηθούν στην ανάλυση και κατασκευή διευθύνσεων **URL**. Η κλάση **QHttpMultiPart** είναι υπεύθυνη για τις αιτήσεις **HTTP** πολλαπλών μερών που χρησιμοποιούνται για μεταφόρτωση αρχείων.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο

Pyqt:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QUrl>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    networkManager(new QNetworkAccessManager(this))
{
    ui->setupUi(this);
    # Connect button signal to custom slot
    connect(ui->fetchButton, &QPushButton::clicked, this, &MainWindow::onFetchButtonClicked);
    # Connect the network reply finished signal to a slot
    connect(networkManager, &QNetworkAccessManager::finished, this, &MainWindow::onReplyFinished);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::onFetchButtonClicked()
{
    # Start an HTTP request
    QUrl url("http://example.com"); // Replace with your URL
    networkManager->get(QNetworkRequest(url));
}

void MainWindow::onReplyFinished(QNetworkReply *reply)
{
    if(reply->error() == QNetworkReply::NoError) {
        # Process the reply content
        QString content = reply->readAll();
        # Display content or do something with it
    }
    reply->deleteLater();
}
```

Εικόνα 3.2.δ Παράδειγμα χρήσης του **QtNetwork Module**

QtSql

Το **module QSql** της **Pyqt** αποτελεί ευέλικτο εργαλείο για την ανάπτυξη εφαρμογών που βασίζονται σε βάσεις δεδομένων, είτε αφορά τοπικές βάσεις δεδομένων είτε μεγαλύτερους διακομιστές βάσεων δεδομένων. Η κλάση **QSqlDatabase** χειρίζεται τη

σύνδεση με τη βάση δεδομένων, όπως **SQLite**, **MySQL**, **PostgreSQL**, συμπεριλαμβανομένου του ανοίγματος και του κλεισίματος συνδέσεων. Η κλάση **QSqlQuery** επιτρέπει την εκτέλεση ερωτημάτων SQL σε μια βάση δεδομένων και η κλάση **QSqlQueryModel** χρησιμοποιείται για την παροχή δεδομένων μόνο για ανάγνωση από ερωτήματα SQL. Για τη διαχείριση σφαλμάτων κατά τη λειτουργία της βάσης δεδομένων υπεύθυνη είναι η κλάση **QSqlError**. Επίσης, οι κλάσεις **QSqlDatabase.transaction()**, **QSqlDatabase.commit()**, **QSqlDatabase.rollback()** αποτελούν τις μεθόδους για τη διαχείριση των συναλλαγών, που επιτρέπουν μεγαλύτερο έλεγχο των λειτουργιών της βάσης δεδομένων και εξασφαλίζουν την ακεραιότητα των δεδομένων. Τέλος, η κλάση **QSqlTableModel** χρησιμοποιείται για ανάγνωση και εγγραφή σε έναν συγκεκριμένο πίνακα της βάσης δεδομένων.

Στην παρακάτω εικόνα υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    initializeDatabase();
    createTable();
    insertData();
    displayData();
}

MainWindow::~MainWindow()
{
    db.close();
    delete ui;
}

void MainWindow::initializeDatabase()
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(":memory:"); // Using in-memory database for example purposes
    if (!db.open()) {
        QMessageBox::critical(this, "Error", "Failed to open database");
    }
}

void MainWindow::createTable()
{
    QSqlQuery query;
    query.exec("CREATE TABLE people (id INTEGER PRIMARY KEY, name TEXT, age INTEGER);");
}

void MainWindow::insertData()
{
    QSqlQuery query;
    query.exec("INSERT INTO people (name, age) VALUES ('Alice', 30)");
    query.exec("INSERT INTO people (name, age) VALUES ('Bob', 25)");
}

void MainWindow::displayData()
{
    QSqlQuery query("SELECT name, age FROM people");
    while (query.next()) {
        QString name = query.value(0).toString();
        int age = query.value(1).toInt();
        qDebug() << name << age; // Replace with your display logic
    }
}
```

Εικόνα 3.2.ε Παράδειγμα χρήσης του **QtSql Module**

QtTest

Το **module QtTest** της **Pyqt** διαθέτει εργαλεία και βοηθητικά προγράμματα που διευκολύνουν τη διαδικασία συγγραφής δοκιμών μονάδας για εφαρμογές, διασφαλίζοντας ότι το γραφικό περιβάλλον συμπεριφέρεται όπως χρειάζεται. Το **QtTest** είναι ιδιαίτερα χρήσιμο σε στοιχεία **GUI**, όπως κουμπιά και άλλα **widgets** καθώς ελέγχει αν συμπεριφέρονται σωστά κάτω από διάφορες ενέργειες του χρήστη, στον αυτόματο έλεγχο που γίνεται στις αλλαγές στην βάση κώδικα και στην εξασφάλιση της αξιοπιστίας της

εφαρμογής, καθώς επαληθεύεται η άρτια λειτουργία της εκάστοτε εφαρμογής. Τα βασικά χαρακτηριστικά του **QtTest** είναι:

1. Δοκιμές εφαρμογών **PyQt**: Το **QtTest** παρέχει κλάσεις και συναρτήσεις που βοηθούν στη δημιουργία αυτοματοποιημένων δοκιμών ειδικά για εφαρμογές **PyQt**. Έτσι, επιτυγχάνεται η σωστή λειτουργικότητα και σταθερότητα των στοιχείων **GUI**.
2. Προσομοίωση της αλληλεπίδρασης του χρήστη: Το **QtTest** περιλαμβάνει τη δυνατότητα προσομοίωσης διάφορων ενεργειών του χρήστη κλικ του ποντικιού, πάτημα πλήκτρων και άλλα συμβάντα, αυτό επιτρέπει στον εκάστοτε προγραμματιστή να ελέγχει προγραμματιστικά τον τρόπο με τον οποίο τα widgets ανταποκρίνονται στην είσοδο του χρήστη.
3. Ενσωμάτωση με πλαίσια δοκιμών: Το **QtTest** έχει τη δυνατότητα να προσθέσει τυποποιημένα πλαίσια δοκιμών, όπως το unittest ή το pytest, έτσι θα καταφέρει οι δοκιμές που γίνονται να είναι πιο δομημένες και επεκτάσιμες.

Στις παρακάτω εικόνες υπάρχει ένα παράδειγμα που δημιουργήθηκε ένα παράθυρο **Pyqt**:
Στην πρώτη είναι το κύριο παράθυρο:

```
#include <QtTest>
#include "testcalculator.h"

QTEST_MAIN(TestCalculator)
```

Εικόνα 3.2.στ.1 Παράδειγμα χρήσης του **QtTest Module**

Στη δεύτερη είναι η συνάρτηση για το τεστ

```
#include "testcalculator.h"

void TestCalculator::testAdd()
{
    QCOMPARE(Calculator::add(1, 1), 2);
    QCOMPARE(Calculator::add(-1, 1), 0);
    QCOMPARE(Calculator::add(0, 0), 0);
}
```

Εικόνα 3.2.στ.2 Παράδειγμα χρήσης του **QtTest Module**

Στην τρίτη είναι η κλήση της συνάρτησης για την εκτέλεση του τεστ

```

#ifndef TESTCALCULATOR_H
#define TESTCALCULATOR_H

#include <QtTest>
#include "calculator.h"

class TestCalculator : public QObject
{
    Q_OBJECT

private slots:
    void testAdd();
};

#endif // TESTCALCULATOR_H

```

Εικόνα 3.2.στ.3 Παράδειγμα χρήσης του **QtTest Module**

ΚΕΦΑΛΑΙΟ 4 ΣΥΓΚΡΙΣΗ TKINTER ΚΑΙ PYQT

Η σύγκριση μεταξύ της **Tkinter** και της **Pyqt** φαίνεται παρακάτω:

1. Ευκολία χρήσης και εκμάθησης

- **Tkinter**: Αποτελεί καλύτερη επιλογή για τους αρχάριους. Το **Tkinter** έχει απλούστερο **API(Application Programming Interface)** και περιλαμβάνεται στις περισσότερες εγκαταστάσεις της **Python**, οπότε δεν χρειάζεται να εγκατασταθούν πρόσθετα πακέτα. Είναι μια καλή επιλογή για εφαρμογές μικρής κλίμακας και για όσους είναι αρχάριοι στην ανάπτυξη γραφικών διεπαφών.
- **PyQt**: Αποτελεί πιο σύνθετη στην εκμάθησή της σε σύγκριση με το **Tkinter**, όμως παρέχει πιο προηγμένες λειτουργίες και **widgets**. Το **PyQt** είναι καταλληλότερο για προγραμματιστές που απαιτούν ολοκληρωμένες λειτουργίες και είναι διατεθειμένοι να επενδύσουν περισσότερο χρόνο στην εκμάθηση.

Γενικά, η επιλογή μεταξύ αυτών των δύο εξαρτάται από το τρέχον επίπεδο δεξιοτήτων του χρήστη, την πολυπλοκότητα του έργου και τη μακροπρόθεσμη επεκτασιμότητα των δεξιοτήτων και των εργαλείων που εμπλέκονται.

2. Χαρακτηριστικά και δυνατότητες

- **Tkinter**: Προσφέρει βασικά **widgets** και εργαλεία που απαιτούνται για απλές εφαρμογές **GUI**. Είναι περιορισμένο όσον αφορά τα προηγμένα χαρακτηριστικά, όπως προσαρμοσμένα **widgets**, κινούμενα σχέδια και **styling**.

- **PyQt:** Παρέχει ένα ευρύ φάσμα προηγμένων **widgets** και χαρακτηριστικών, συμπεριλαμβανομένης της υποστήριξης για προσαρμοσμένα **widgets**, σύνθετες διατάξεις, κινούμενα σχέδια και ενσωμάτωση με περιεχόμενο ιστού. Είναι πιο κατάλληλο για την ανάπτυξη εξελιγμένων και σύγχρονων γραφικών διεπαφών.

Η επιλογή μεταξύ του **Tkinter** και του **PyQt** όσον αφορά τα χαρακτηριστικά και τις δυνατότητες εξαρτάται από τις απαιτήσεις του έργου και το επιθυμητό επίπεδο πολυπλοκότητας. Για βασικές εφαρμογές, η απλότητα του **Tkinter** μπορεί να είναι πλεονεκτήματα, ενώ για πιο προηγμένες, πλούσιες σε χαρακτηριστικά εφαρμογές, το **PyQt** είναι η καταλληλότερη επιλογή.

3. Εμφάνιση

- **Tkinter:** Η εμφάνιση των εφαρμογών **Tkinter** μπορεί να είναι κάπως ξεπερασμένη και μπορεί να μην ευθυγραμμίζεται πάντα με την εγγενή εμφάνιση του λειτουργικού συστήματος.
- **PyQt:** Προσφέρει εγγενή εμφάνιση σε διαφορετικές πλατφόρμες και επιτρέπει εκτεταμένη προσαρμογή της εμφάνισης του **GUI** με τη χρήση στυλ (παρόμοια με το **CSS**).

Η επιλογή μεταξύ του **Tkinter** και του **PyQt** όσον αφορά την εμφάνιση εξαρτάται από τη σημασία της αισθητικής της εφαρμογής και τις απαιτήσεις της εμπειρίας του χρήστη. Για απλές εφαρμογές εργαλείων και βοηθητικών εφαρμογών όπου η λειτουργικότητα είναι η προτεραιότητα, η απλότητα του **Tkinter** μπορεί να είναι αρκετή. Για εφαρμογές όπου η εμπλοκή του χρήστη, η σύγχρονη αισθητική και το **branding** είναι ζωτικής σημασίας, το **PyQt** προσφέρει πιο ισχυρές δυνατότητες.

4. Επιδόσεις

- **Tkinter:** Γενικά επαρκής για απλές εφαρμογές, αλλά ενδέχεται να δυσκολεύεται με τις επιδόσεις σε πιο σύνθετες ή απαιτητικές εφαρμογές.
- **PyQt:** Τείνει να προσφέρει καλύτερες επιδόσεις, ειδικά για εφαρμογές που απαιτούν το χειρισμό σύνθετων διατάξεων, κινούμενων σχεδίων ή την απόδοση περιεχομένου ιστού.

Συνοψίζοντας, η επιλογή μεταξύ του **Tkinter** και του **PyQt** όσον αφορά τις επιδόσεις θα πρέπει να βασίζεται στις απαιτήσεις της εφαρμογής. Εάν η προτεραιότητα είναι η ανάπτυξη μιας απλής και ελαφριάς εφαρμογής, το **Tkinter** είναι συχνά επαρκές. Για πιο σύνθετες, πλούσιες σε χαρακτηριστικά εφαρμογές που η ομαλή εμπειρία του χρήστη είναι ζωτικής σημασίας, η **PyQt** καθιστά καταλληλότερη επιλογή, παρά τις δυνητικά υψηλότερες απαιτήσεις της σε πόρους.

5. Κοινότητα και υποστήριξη

- **Tkinter**: Έχει μεγάλη κοινότητα λόγω της μακράς ιστορίας του και της ευκολίας χρήσης του για αρχάριους. Ωστόσο, ενδέχεται να στερείται προηγμένων πόρων ή παραδειγμάτων για σύνθετες εργασίες.
- **PyQt**: Ωφελείται από το γεγονός ότι αποτελεί μέρος της ευρύτερης κοινότητας **Qt**, η οποία περιλαμβάνει εμπορική υποστήριξη. Υπάρχουν εκτεταμένοι πόροι και τεκμηρίωση, ειδικά για πολύπλοκες λειτουργίες.

Όσον αφορά την κοινότητα και την υποστήριξη, η επιλογή μεταξύ του **Tkinter** και του **PyQt** εξαρτάται συχνά από τις συγκεκριμένες ανάγκες του προγραμματιστή ή του έργου. Για τους αρχάριους και τα έργα μικρότερης κλίμακας, η ευθύτητα του **Tkinter**, σε συνδυασμό με την υποστήριξη της κοινότητας της **Python**, το καθιστά προσιτό και εύκολο στην αρχή. Για πιο σύνθετες εφαρμογές και επαγγελματικά περιβάλλοντα ανάπτυξης, η ευρύτερη υποστήριξη της κοινότητας του **PyQt**, σε συνδυασμό με επαγγελματικούς πόρους και ένα πλούσιο οικοσύστημα εργαλείων, το καθιστά καταλληλότερη επιλογή.

6. Αδειοδότηση

- **Tkinter**: Έρχεται με την τυπική βιβλιοθήκη της **Python** και είναι ελεύθερο προς χρήση.
- **PyQt**: Η αδειοδότηση μπορεί να αποτελέσει προβληματισμό. Ενώ υπάρχει μια έκδοση με άδεια χρήσης **GPL (PyQt GPL)**, οι εμπορικές εφαρμογές μπορεί να απαιτούν άδεια χρήσης **PyQt** επί πληρωμή.

Συνοψίζοντας, η επιλογή μεταξύ του **Tkinter** και του **PyQt** όσον αφορά την αδειοδότηση θα πρέπει να βασίζεται στη φύση του έργου (ανοικτού κώδικα ή ιδιοκτησίας) και στις οικονομικές και νομικές εκτιμήσεις της ομάδας ή του οργανισμού ανάπτυξης. Τα έργα ανοικτού κώδικα μπορούν να χρησιμοποιούν ελεύθερα οποιοδήποτε από τα δύο πλαίσια υπό τις αντίστοιχες άδειες χρήσης τους. Αντίθετα, τα ιδιόκτητα έργα μπορούν να χρησιμοποιούν το **Tkinter** χωρίς ανησυχίες σχετικά με τα τέλη αδειοδότησης, ενώ το **PyQt** θα απαιτούσε εμπορική άδεια για μια τέτοια χρήση.

ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑ

Γενικά, η επιλογή μεταξύ του **Tkinter** και του **PyQt** εξαρτάται από διάφορους παράγοντες. Το **Tkinter** είναι μια τυπική εργαλειοθήκη **GUI (Graphical User Interface)** για την **Python**, με την οποία έρχεται πακέτο, δεν χρειάζεται να εγκατασταθούν πρόσθετες βιβλιοθήκες ή πακέτα. Το **Tkinter** είναι διαπλατφορμικό, είναι γνωστό για την απλότητα, την ταχύτητα και την ευκολία χρήσης του. Διαθέτει ένα απλό και διαισθητικό **API**, καθιστώντας το προσιτό για αρχάριους και ο κώδικας είναι συνήθως συνοπτικός. Επίσης, διαθέτει εκτεταμένη τεκμηρίωση και μεγάλη κοινότητα χρηστών, μπορεί εύκολα να ενσωματωθεί με άλλες βιβλιοθήκες και πλαίσια **Python**. Το **Tkinter** είναι ανοικτού κώδικα και δωρεάν στη χρήση, καθιστώντας το μια οικονομικά αποδοτική επιλογή. Από την άλλη η επιλογή του **PyQt** για εφαρμογές **GUI** έχει πολλά πλεονεκτήματα. Το **PyQt** είναι ένα σύνολο

συνδέσεων Python για το πλαίσιο εφαρμογών Qt και παρέχει ένα ισχυρό και πλούσιο σε δυνατότητες περιβάλλον για τη δημιουργία γραφικών διεπαφών χρήστη. Το PyQt προσφέρει ένα ευρύ φάσμα προηγμένων widgets και χαρακτηριστικών. Το PyQt συνοδεύεται από το Qt Designer, ένα εργαλείο οπτικού σχεδιασμού που επιτρέπει να σχεδιαστεί το GUI με drag and drop widgets. Οι εφαρμογές Qtέχουν επαγγελματική εμφάνιση. Το PyQt είναι διαπλατφορμικό. Το PyQt διευκολύνει την εκμάθηση και την αντιμετώπιση προβλημάτων. Επιπλέον, διαθέτει μια ισχυρή, ενεργή κοινότητα προγραμματιστών και είναι διαθέσιμο τόσο με εμπορικές άδειες όσο και με άδειες ανοικτού κώδικα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- 1) <https://docs.python.org/3/>
- 2) "Python Crash Course" by Eric Matthes (2019):
- 3) "Fluent Python" by Luciano Ramalho (2015):
- 4) <https://realpython.com/>
- 5) "Python and Tkinter Programming" by John E. Grayson(2000)
- 6) "Modern Tkinter for Busy Python Developers" by Mark Roseman(2012)
- 7) <https://docs.python.org/3/library/tkinter.html>
- 8) <https://tkdocs.com/tutorial/>
- 9) <https://www.geeksforgeeks.org/python-gui-tkinter/>
- 10) https://www.tutorialspoint.com/python/python_gui_programming.htm
- 11) <https://realpython.com/python-gui-tkinter/>
- 12) <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
- 13) <https://www.riverbankcomputing.com/static/Docs/PyQt6/>
- 14) <https://www.pythonguis.com/pyqt5-tutorial/>
- 15) <https://zetcode.com/pyqt6/>
- 16) <https://www.tutorialspoint.com/pyqt5/index.htm>
- 17) Herrmann, Michael. "Python and Qt: The Best Parts."
- 18) Summerfield, Mark. "Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming." ISBN: 978-0132354189
- 19) "PyQt6 By Example" by Nicholas J. Harbour (2021)
- 20) "Python GUI Programming with Tkinter" by Alan D. Moore (2018):
- 21) "Rapid GUI Programming with Python and Qt" by Mark Summerfield (2007)

