УДК 004.4:378.147
DOI: 10.31652/2412-1142-2021-60-143-157

Бак Сергій Миколайович
доктор фізико-математичних наук, професор кафедри математики та інформатики
Вінницького державного педагогічного університету імені Михайла Коцюбинського, м. Вінниця, Україна
ORCID ID: 0000-0003-1508-2144
sergiy.bak@gmail.com

Ковтонюк Галина Миколаївна
кандидат педагогічних наук, доцент кафедри математики та інформатики
Вінницького державного педагогічного університету імені Михайла Коцюбинського, м. Вінниця, Україна
ORCID ID: 0000-0002-3352-0358
galyna.kovtonyuk@gmail.com

ОСОБЛИВОСТІ СТВОРЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА ПІД ЧАС ВИВЧЕННЯ ПРОГРАМУВАННЯ МОВОЮ PYTHON МАЙБУТНІМИ ВЧИТЕЛЯМИ МАТЕМАТИКИ

Анотація. Стаття присвячена методичним аспектам вивчення особливостей створення користувацького графічного інтерфейсу під час вивчення програмування (на прикладі мови Python), що є необхідним для формування інформатичної компетентності майбутніх учителів математики. Зокрема, у статті продемонстровано авторський методичний підхід до вивчення даної теми, який передбачає комплексну теоретичну і практичну підготовку. Теоретична підготовка забезпечується на лекційних заняттях і передбачає, перш за все, засвоєння сутності поняття «графічний інтерфейс користувача». Для створення графічного інтерфейсу користувача студентам пропонується використання модуля Tkinter, який входить у стандартну бібліотеку Python і має досить потужні графічні можливості. Цей модуль має стандартний набір об'єктів (візуальних елементів керування або віджетів), за допомогою яких створюється графічний інтерфейс. Далі розкривається сутність понять: клас, властивість і метод. Наводиться алгоритм створення графічного інтерфейсу користувача. Вивчаються основні віджети (Button, Label, Entry, Text, Radiobutton, Checkbutton, Listbox, Menu тощо) та відповідні їм властивості і методи. Наводяться приклади програм з цими віджетами та результати їх виконання. Розглядаються можливості створення діалогових вікон. Звертається особлива увага на питання розташування віджетів у вікні. Для цього використовуються так звані менеджери розташування. Вивчаються три основні менеджери розташування: pack(), place(), grid(). Наводяться приклади їх застосування. В кінці цієї теми вивчається питання, яке стосується опрацювання подій. З'ясовується сутність поняття події і наводиться їх класифікація. Описано способи створення обробників подій та їх зв'язування з самими подіями. Наведено відповідні приклади. Практична підготовка студентів з даної теми здійснюється на практичних і лабораторних заняттях. Останні передбачають наявність індивідуальних завдань. Такий комплексний підхід, який поєднує теоретичну і практичну підготовку може сприяти якісному засвоєнню знань, набуттю практичних вмінь і навичок, здатності формулювати і розв'язувати практичні задачі у професійній діяльності.

Ключові слова: підготовка вчителів математики, інформатична компетентність, програмування, Python, графічний інтерфейс користувача, візуальні елементи керування.

1. ВСТУП

Постановка проблеми. Сучасна вища школа, на відміну від традиційної, представляється як «школа компетентностей». Основна особливість компетентностей, у порівнянні зі знаннями та вміннями, полягає у діяльності майбутнього фахівця, оскільки оволодіння професією ототожнюється із усвідомленням того, які саме задачі потрібно навчитися розв'язувати, і, головне, здатністю розв'язувати такі задачі у професійній діяльності. Безумовно, інформатична компетентність є однією із таких компетентностей у майбутніх учителів математики. Вона дозволяє не тільки розв'язувати задачі за допомогою відомих прикладних програм, але й для розв'язування цих задач проектувати власні програми. [1]

Більше того, реалії української школи такі, що є великий попит на вчителів інформатики, яких вкрай не вистачає в школах. Не секрет, що кращі випускники з дипломом вчителя інформатики переважно не працюють в школах, а обирають більш оплачувані професії, зокрема, в сфері ІТ. Разом з тим нинішнє законодавство дозволяє працювати фахівцям, які не мають педагогічної освіти, але знають і можуть викладати свій предмет на високому рівні. Тому вчителем інформатики може працювати, наприклад, випускник технічного ЗВО, або ж вчитель з дипломом з іншої предметної спеціальності. В цьому розрізі вчитель математики, який є фахівцем з інформатики, є чудовим виходом з ситуації, що склалася.

Аналіз останніх досліджень. Проблемам підготовки майбутніх учителів з алгоритмізації і програмування та технологіям формування в них інформатичної компетентності присвячені праці В. Бикова, В. Величка, Р. Гуревича, М. Жалдака, В. Жукової, Н. Морзе, М. Рагуліної, С. Ракова, Ю. Рамського, В. Осадчого, О. Семеніхіної та ін.

Окремі аспекти вивчення алгоритмізації і програмування у ЗВО досліджували А. Гуржій, М. Жалдак, В. Клочко, Е. Кузнєцов, М. Львов, О. Миленький, В. Монахов, Н. Морзе, В. Петрушин, С. Раков, Ю. Рамський, Д. Румянцев, С. Семеріков, О. Співаковський, О. Спірін, Н. Шаховська та ін.

Проведений аналіз результатів досліджень дозволяє зробити висновок, що в системі підготовки майбутніх учителів є низка невирішених проблем та різних аспектів, які потребують більш детального вивчення. Зокрема, це стосується підготовки майбутніх учителів математики та формування в них інформатичної компетентності під час вивчення програмування.

Мета статті: розкрити деякі методичні аспекти вивчення особливостей створення користувацького графічного інтерфейсу в додатках під час вивчення програмування мовою Python майбутніми вчителями математики.

2. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Як вже було зазначено, для формування інформатичної компетентності у майбутніх учителів математики необхідним є вивчення програмування, яке традиційно входить до навчальних програм з інформатики. Звісно, його вивченню передує вивчення алгоритмізації, яка є фундаментом для програмування. Під час вивчення програмування студенти обов'язково вивчають алфавіт і синтаксис мови (однієї або декількох), типи даних та операторів, різні парадигми програмування тощо.

Однією із важливих і цікавих для студентів тем є «Графічний інтерфейс користувача. Візуальні елементи керування в Python», оскільки більшість сучасних додатків мають графічний інтерфейс користувача і лише виняткові програми, які передбачають взаємодію з людиною, залишаються консольними. Зрозуміло, що до вивчення цієї теми студенти писали консольні програми. Зауважимо, що дана тема входить до діючих навчальних програм з інформатики для студентів спеціальностей 111 Математика (освітня програма «Комп'ютерна математика») та 014.04 Середня освіта (Математика) (освітня програма «Середня освіта. Математика, інформатика»), затверджених Вченою радою Вінницького державного педагогічного університету імені Михайла Коцюбинського. Варто також зазначити, що у вказаних програмах для вивчення програмування дозволяється вільний вибір мови програмування. Саме тому для вивчення програмування, провівши детальне дослідження та взявши до уваги переваги і недоліки, наш вибір було зроблено на користь Python [7]. Зауважимо, що зараз є багато різноманітних посібників, в яких доступно і достатньо повно викладені основи програмування мовою Python (див., наприклад, [3–6]).

Варто зазначити, що вивчення створення графічного інтерфейсу користувача під час вивчення програмування, є обов'язковим не тільки в програмах з інформатики для майбутніх вчителів математики, але й невід'ємним компонентом шкільних навчальних програм з інформатики (див. [5]).

На початку вивчення цієї теми студенти знайомляться (або пригадують) з поняттям

інтерфейсу користувача. Нагадаємо, що під *інтерфейсом користувача* розуміють засіб зручної взаємодії користувача з програмою, інформаційною системою. Користувач має довільний доступ (за допомогою клавіатури або миші) до всіх видимих екранних об'єктів – елементів інтерфейсу. Зокрема, *текстовий інтерфейс користувача* (англ. ***T**ext **U**ser **I**nterface* або **TUI**) – спосіб взаємодії користувача з комп'ютером з використанням текстового (буквено-цифрового) режиму дисплея або аналогічних, наприклад, командного рядка. Програми з текстовим інтерфейсом називаються *консольними*. А от *графічний інтерфейс користувача* (англ. ***G**raphical **U**ser **I**nterface* або **GUI**) – тип інтерфейсу, який дозволяє користувачеві взаємодіяти з комп'ютерною програмою або електронним пристроєм через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів.

Зауважимо, що виклад лекційних матеріалів обов'язково супроводжується наочною демонстрацією прикладів, рисунків, схем тощо. Зокрема, студентам демонструються готові програми, які під час лекції запускаються на виконання в інтегрованому середовищі розробки IDLE.

Далі студенти знайомляться з поняттям графічної бібліотеки. Мова Python забезпечує роботу з різними графічними бібліотеками або модулями (Tkinter, PyQt, wxPython, PyGTK та інші), за допомогою яких можна створювати програми з графічним інтерфейсом користувача. Ми пропонуємо студентам для вивчення модуль *Tkinter* (від англ. **T**ool**k**it for **I**nterfaces), який входить у стандартну бібліотеку Python і має досить потужні графічні можливості. Ця бібліотека має стандартний набір об'єктів (візуальних елементів керування), за допомогою яких власне і створюється графічний інтерфейс. *Візуальні елементи керування* або *віджети* (від англ. *widget – window gadget*) – графічні об'єкти, розташовані у вікні програми для показу або введення даних, виконання дій або полегшення роботи: текстові поля, списки, перемикачі, кнопки, прапорці тощо. Кожному об'єкту в бібліотеці відповідає свій *клас*, що має *властивості* та *методи*. *Властивості* – це певні параметри, характеристики об'єкта, значенння яких описують його зовнішній вигляд чи стан. *Метод* – це фрагмент програмного коду, який вбудований в об'єкт та передбачає виконання деяких дій з ним. Тобто *методи* – це міні-програми, що пов'язані з об'єктами.

Для повного розуміння цих понять студентам наводяться очевидні приклади з життя. Наприклад, *клас* – Машини, *об'єкти:* Lamborghini, Ferrari, Electric, *властивості*: колір, об'єм двигуна, потужність, тип коробки передач, *методи*: їхати, зупинитися, ввімкнути/вимкнути сигналізацію тощо. Для виклику (виконання) методу об'єкта використовують такий синтаксис:

Імя_об'єкта.Метод()

Для прикладу метод *pack()* в модулі *tkinter* виконує розміщення об'єкта у вікні.

Далі студенти з'ясовують, що головне вікно додатку створюється за допомогою класу *Tk,* який є базовим класом в *Tkinter*. При створенні об'єкта цього класу буде завантажено інтерпретатор *tcl/tk* і створено основне вікно програми. Для використання можливостей графічного інтерфейсу необхідно в програмі передбачити розміщення у вікні необхідних графічних об'єктів, а також дії, які мають виконуватися в процесі взаємодії користувача з цими об'єктами. Всі віджети в програмі створюються за однаковим принципом, а саме шляхом виклику конструктора відповідного класу, який має таку загальну структуру:

<ім'я змінної>=<назва класу>[(параметри)]

Наприклад, створити кнопку можна за допомогою такої інструкції:

Далі ми знайомимо студентів з алгоритмом створення графічного інтерфейсу користувача, який передбачає дотримання такої послідовності дій:

1. Імпортувати модуль Tkinter і створити головне вікно.

Інструкція імпортування модуля *import tkinter* або *from tkinter import**. Для створення вікна використовуємо клас *Tk*. Інструкція оголошення змінних цього класу має такий вигляд:

<ім'я змінної>=Tk( )

Змінну пов'язану з об'єктом – вікно, зазвичай називають *root* (рис. 1).

```
from tkinter import*
root=Tk()
root.mainloop()
```

або

```
import tkinter
root=tkinter.Tk()
root.mainloop()
```

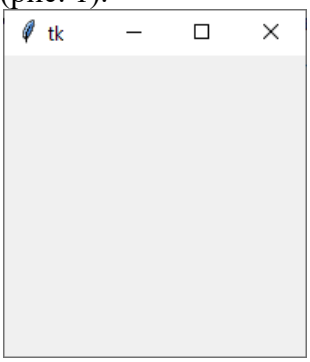


Рис. 1. Створення головного вікна

Через змінну *root* можна керувати атрибутами (властивостями) вікна. Для цього використовують такі *методи* (рис. 2):

- *title*() – можна змінити заголовок вікна. За замовчуванням заголовок вікна tk;
- *geometry*() – розмір вікна;
- *resizable*(0,0) – фіксований розмір вікна (width, height)

```
from tkinter import*
root=Tk()
root.geometry("400x400+300+300")
root.resizable(False,False)
root.title("Графічний інтерфейс користувача")
root.iconbitmap("as.ico")
mainloop()
```

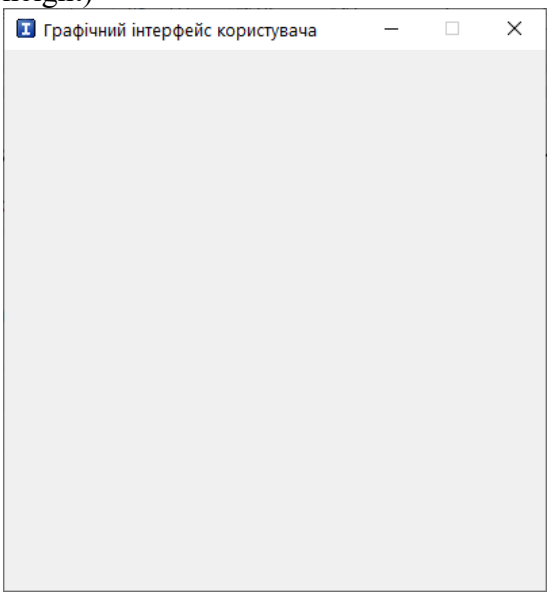


Рис. 2. Створення голвного вікна із використанням методів

2. Створити віджети та визначити їх властивості. Будь-який віджет має бути пов'язаний із відповідним ім'ям змінної.

3. Визначити події і порядок їх створення. Наприклад, які дії необхідно виконати під час натиснення кнопки.

4. У головному вікні розмістити віджети, використовуючи менеджер *(пакувальник, менеджер геометрії)* розташування віджетів у вікні: pack(), grid(), place().

5. Відобразити головне вікно на екрані. Вікно відображається за допомогою методу **mainloop**(). За допомогою інструкції:

root.mainloop()

Дана інструкція має бути останньою в програмі.

Після цього студенти ознайомлюються з візуальними елементи керування (віджетами) та їх властивостями. До основних типів віджетів в *tkinter* належать: Button, Label, Entry, Text, Listbox Checkbutton, Radiobutton, Combobox, Frame, Scrollbar, TopLevel, Scale, Progressbar.

Першим ми розглядаємо віджет **Button** (*кнопка*). *Кнопки* призначені для керування програмою. Кнопці відповідає клас **Button.** Структура конструктора створення кнопки має вигляд:

<змінна>=Button(<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

Перший параметр визначає посилання на об'єкт, у якому створюється кнопка. Його ще називають *батьківським* параметром. Інші параметри визначають властивості кнопки та їх значення. Ці параметри оголошуються за такою структурою:

<властивість=значення>

До основні *властивостей* кнопки належать:

- **text** – текст, який буде відображатися на кнопці;
- **bg/ background –** фоновий колір кнопки;
- **width, height –** ширина і висота кнопки;
- **fg/ foreground** – колір тектсу на кнопці;
- **font** – шрифт, наприклад, font=("Verdana", 13, "bold");
- **bd** – товщина межі (за замовчуванням 2);
- **justify** – вирівнювання тексту. Значення LEFT вирівнювання тексту по лівому краю, CENTER – по центру, RIGHT – по правому краю;
- **padx** – відступ від межі кнопки до її тексту зправа і зліва;
- **pady** – відступ від межі кнопки до її тексту зверху і знизу;
- **textvariable** – втановлює прив'язку до елемента StringVar.

Далі наводимо приклад створення кнопки (рис. 3).

```
from tkinter import*
root = Tk()
root.title("Віджет кнопка")
button1=Button(root,text="Кнопка1")
button2=Button(root,
               text="Кнопка2",
               width=20,height=2,
               bg="blue",fg="red",
               font="Arial 14")
button1.pack()
button2.pack()
root.mainloop()
```

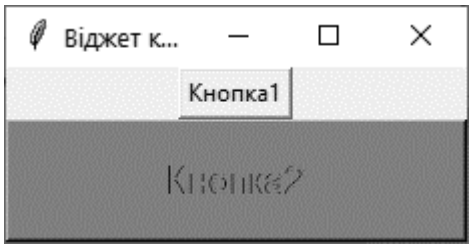


Рис. 3. Створення кнопок

Наступний віджет – **Label** (*мітка*). *Мітки* (написи) використовують для розміщення необхідної інформації, зокрема, для інформування користувача про результати виконання програмою дій або про дії які необхідно виконати. Мітки можуть містити один або декілька рядків. Мітці відповідає клас **Label**. Структура конструктора створення мітки має вигляд:

<змінна>= Label (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

До основних *властивостей* мітки належать:

- **anchor** – позиціювання тексту;
- **bg/background** – фоновий колір;
- **bd/border-width** – товщина межі мітки;
- **fg/foreground** – колір тексту;
- **font** – шрифт тексту;
- **height, width –** висота, ширина;
- **justify** – вирівнювання тексту. Значення LEFT вирівнювання тексту по лівому краю, CENTER – по центру, RIGHT – по правому краю;

- **text** – встановлює текст мітки;
- **textvariable** – втановлює прив'язку до елемента StringVar.

Далі наводимо приклад створення мітки (рис. 4).

```
from tkinter import*
root = Tk()
root.title("Віджет мітка")
label1=Label(root,text="Hello Python",
                  width=10,height=2,
                  bg="white",fg="red",
                  font="Arial 14")
Text = "Програми з графічним інтерфейсом\nмають розширення -.p
label2 = Label(text=Text, justify=CENTER)
label1.pack()
label2.pack()
root.mainloop()
```

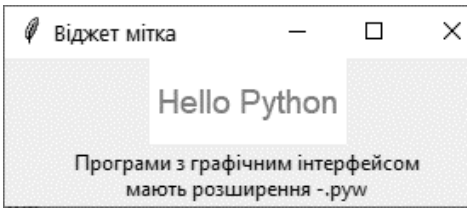


Рис. 4. Створення міток

Після цього розгядається віджет **Entry** (*однорядкове текстове поле*). Такі поля використовуються для введення користувачем тексту в один рядок. Таким полям відповідає клас **Entry.** Структура конструктора створення однорядкового текстового поля має вигляд:

<змінна>= Entry (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

До основних *властивостей* Entry належать:

- **bg/background** – фоновий колір;
- **bd/border-width** – товщина межі текстового поля;
- **fg/foreground** – колір тексту;
- **font** – шрифт тексту;
- **height, width –** висота, ширина;
- **justify** – вирівнювання тексту. Значення LEFT вирівнювання тексту по лівому краю, CENTER – по центру, RIGHT – по правому краю;
- **text** – встановлює текст в полі;
- **textvariable** – втановлює прив'язку до елемента StringVar.

Далі наводимо приклад створення однорядкового текстового поля (рис. 5).

```
from tkinter import *
root = Tk()
edit1 = Entry(root,width=18,bd=5).pack()
root.mainloop()
```

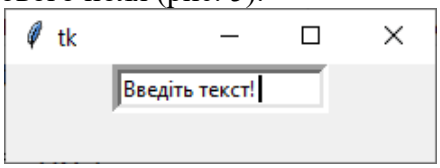


Рис. 5. Створення однорядкового текстового поля введення

До основних *методів* Entry належать:

- **get() –** дозволяє отримати значення, що знаходиться в текстовому полі;
- **insert(index, str) –** виводить в текстове поле рядок, починаючи зі знакомісця з номером index;
- **delete(first, last) –** вилучає символи, починаючи зі знакомісця з номером first до знакомісця з номером last (нумерація починається з 0). Щоб вилучити увесь текст, в якості другого параметра потрібно указати END. Бажано перед викликом методу insert() записувати виклик методу delete(), тобто перед виведенням очищувати текстове поле.

Наступний віджет – **Text** (*багаторядкове текстове поле*). Це поле також використовується для введення тексту користувачем. Йому відповідає клас **Text.** Структура конструктора створення багаторядкового текстового поля має вигляд:

<змінна>= Text (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

До основних *властивостей* поля Text належать:

- **bg/background** – фоновий колір;
- **bd/border-width**– товщина межі текстового поля;

- **fg/foreground** – колір тексту;
- **font** – шрифт тексту;
- **height, width –** висота, ширина;
- **justify** – вирівнювання тексту. Значення LEFT вирівнювання тексту по лівому краю, CENTER – по центру, RIGHT – по правому краю;
- **text** – встановлює текст в полі;
- **wrap** – визначає, що слова не будуть розриватися в процесі перенесення на новий рядок.

Далі наводимо приклад створення багаторядкового текстового поля (рис. 6).

```
from tkinter import *
root = Tk()
root.geometry("200x200")
text1=Text(root,width=20,height=7,font="Calibri 14",wrap=WORD)
text1.pack()
root.mainloop()
```

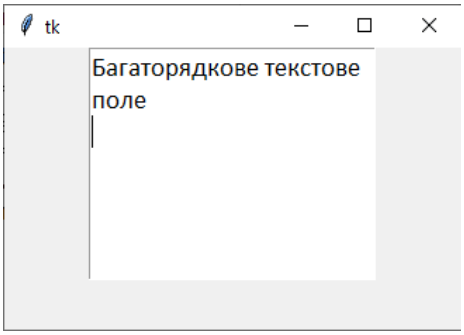


Рис. 6. Створення багаторядкового текстового поля введення

До основних *методів* поля Text належать:

▪ <ім’я змінної тестового поля>.**get**(початок, кінець) – *повертає фрагмент тексту від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром параметром (нумерація рядків починається з 1, а символів в рядку починається з 0);*

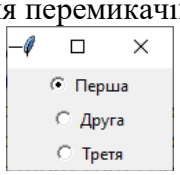
▪ < ім’я змінної текстового поля >.**insert**(позиція, текст) – *вставляє текст в поле перед символом індекс якого вказаний, як парметр позиції (нумерація починається з 0);*

▪ < ім’я змінної текстового поля >.**delete**(початок, кінець) – *видаляє фрагмент тексту від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром параметром (нумерація рядків починається з 1, а символів в рядку починається з 0).*

Далі вивчається віджет **Radiobutton** *(радіокнопка* або *перемикач).* Радіокнопки призначені для вибору одного з кількох запропонованих варіантів. Радіокнопок обов’язково має бути декілька, у кожен окремий момент можна увімкнути лише одну радіокнопку. Цьому віджету відповідає клас **Radiobutton.** Структура конструктора створення перемикачів має вигляд:

<змінна>= Radiobutton (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

Після цього наводимо приклад створення перемикачів (рис. 7).




```
from tkinter import*
root=Tk()
var = IntVar()              # створюємо змінну, яка приймає цілі значення
var.set(1)                  # встановлюємо перше значення для створеної змінної
rad1 = Radiobutton(root, text="Перша", variable=var, value=1)    # створюємо перемикач зі значенням 1
rad2 = Radiobutton(root, text="Друга", variable=var, value=2)    # створюємо перемикач зі значенням 2
rad3 = Radiobutton(root, text="Третя", variable=var, value=3)    # створюємо перемикач зі значенням 3

# розміщуємо перемикачі у вікні
rad1.pack()
rad2.pack()
rad3.pack()
```

Рис. 7. Створення перемикачів

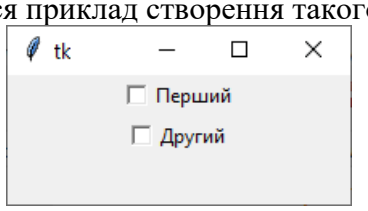
За допомогою команди **var=IntVar()** створюється об’єкт **var**, який виконує роль змінної. За допомогою методу **set**() цій змінній можна надати початкове значення. Початкове значення змінної за допомогою методу **set**() має значення 1. Властивість **variable** зв’язує змінну з радіокнопкою, а властивість **value** визначає значення, яке буде передано змінній, якщо цю кнопку увімкнути. *Tkinter* не може використовувати будь-яку змінну для зберігання станів віджетів. Для цих цілей передбачені спеціальні *класи-змінні* пакета *tkinter*:

- **BooleanVar** (приймає булеві значення);
- **IntVar** (приймає цілі значення);
- **DoubleVar** (приймає дробові та цілі значення);
- **StringVar** (приймає рядкові значення).

Наступний, дещо схожий до попереднього, віджет – **Checkbutton** (прапорець). Прапорців, як і радіокнопок, може бути декілька. Але, на відміну від радіокнопок, кілька прапорців можуть бути одночасно ввімкненими. Цьому віджету відповідає клас **Checkbutton.** Структура конструктора створення перемикачів має вигляд:

<змінна>= Checkbutton (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

Далі знову ж таки наводиться приклад створення такого віджета (рис. 8).




```
from tkinter import *
root = Tk()
var1 = IntVar()                          # створення змінної цілого типу для першого прапорця
check_b1 = Checkbutton(root,
                       text="Перший",
                       variable=var1,    # значення змінної першого прапорця
                       onvalue=1,        # значення при включеному прапорці
                       offvalue=0)       # значення при вимкненому прапорці
check_b1.pack()

var2 = IntVar()                          # створення змінної цілого типу для другого прапорця
check_b2 = Checkbutton(root,
                       text="Другий",
                       variable=var2,    # значення змінної другого прапорця
                       onvalue=1,        # значення при включеному прапорці
                       offvalue=0)       # значення при вимкненому прапорці
check_b2.pack()
root.mainloop()
```

Рис. 8. Створення перемикачів

Властивість **variable** зв’язує змінну з прапорцем, а властивість **value** визначає значення, яке буде передано змінній, якщо цю кнопку увімкнути. За допомогою опції **onvalue** встановлюється значення, яке приймає пов'язана змінна при включеному прапорці. За допомогою властивості **offvalue** – при вимкненому.

Після цього вивчається віджет **Listbox** (*список*). Це віджет, який представляє собою список, з елементів якого користувач може вибирати один або кілька пунктів. Цьому віджету відповідає клас **Listbox.** Структура конструктора створення списку має вигляд:

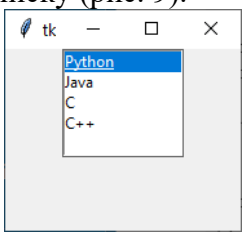
<змінна>= Listbox (<Параметр_1>,< Параметр_2>,…,< Параметр_n>)

До основних *властивостей* списку належать:

- **bg/background** – фоновий колір списку;
- **bd/border-width** – товщина межі текстового поля списку;
- **fg/foreground** – колір тексту;
- **font** – шрифт тексту;
- **height, width –** висота, ширина списку;
- **text** – заголовок списку;

- **selectmode** – визначає скільки елементів можна вибрати і перетягування миші впливає на вибір: SINGLE – можна вибрати лише один рядок, і не можна перетягувати курсор миші; EXTENDED – можна вибрати будь-яку суміжну групу рядків одночасно, натиснувши на перший рядок і перетягнувши на останній рядок.

Далі наводиться приклад створення списку (рис. 9).




```
from tkinter import *
root = Tk()
list1 = ["Python", "Java", "C", "C++"]                          # задаємо елементи, які повинні

listbox1 = Listbox(root, height=5, width=15, selectmode=EXTENDED)    # створюємо віджет – список

for i in list1:                                                 # додаємо елементи у віджет
    listbox1.insert(END, i)
listbox1.pack()
root.mainloop()
```

Рис. 9. Створення списку

До основних методів *Listbox* належать:

•<ім'я змінної списку>.**get**(початок, кінець) – повертає фрагмент списку від рядка, позиція якого визначається першим параметром, до рядка, позиція якого визначена другим параметром;

•<ім'я змінної списку>.**insert**(позиція, назва_елемента) – вставляє елемент в список після даної позиції, визначеної першим параметром (нумерація починається з 0);

•<ім'я змінної списку>.**delete**(початок, кінець) – видаляє зі списку всі елементи.

Ще одним важливим віджетом є **Menu** (*меню*). Цьому віджету відповідає клас **Menu.** Меню може складатися з декількох пунктів, які у свою чергу можуть складатися з підпунктів. Для того, щоб додати пункт в головне меню призначений метод **add_cascade**. А для додавання підпункту (команди) призначений метод **add_command**. При цьому використовується такий синтаксис:

<змінна-посилання на головне вікно>. add_cascade (label="Пункт", menu=<змінна-посилання на основне меню>)

<змінна-посилання на основне меню>. add_command (label="<Команда>")

Для прикладу було створено меню, яке складається з пунктів: Файл, Правка, Довідка з відповідними підпунктами (див. рис. 10).

```
from tkinter import *
root=Tk()
root.title("Menubar")
root.geometry('300x200+500+200')
#створення об'єкта меню Menu у головному вікні
mainmenu=Menu(root)
root.config(menu=mainmenu)
# Файл
mainmenu1=Menu(mainmenu)
mainmenu.add_cascade(label="Файл",menu=mainmenu1)
mainmenu1.add_command(label="Новий файл      Ctrl+N")
mainmenu1.add_command(label="Зберегти          Ctrl+S")
mainmenu1.add_separator()
mainmenu1.add_command(label="Закрити            Alt+F4")
#Правка
mainmenu2=Menu(mainmenu)
mainmenu.add_cascade(label="Правка",menu=mainmenu2)
mainmenu2.add_command(label="Вирізати")
mainmenu2.add_command(label="Копіювати")
#Довідка
mainmenu3=Menu(mainmenu)
mainmenu.add_cascade(label="Довідка",menu=mainmenu3)
mainmenu3.add_command(label="Help")
mainmenu3.add_command(label="Про програму")
root.mainloop()
```

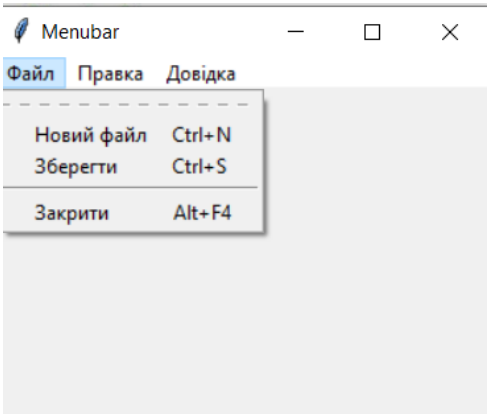


Рис. 10. Приклад створення меню

Для створення графічного інтерфейсу користувача часто є потреба в діалогових вікнах. **Діалогові вікна** призначені для виведення повідомлень користувачу й отримання від нього відповідей та для керування об'єктами. Для їх створення слід, окрім модуля **tkinter**, імпортувати модуль **tkinter.filedialog**. Тут є різні функції, які дозволяють створювати різні типи діалогових вікон. Наприклад, вікно для *відкриття файлів* створюють за допомогою функції **askopenfilename**(), а для *збереження файлів* за допомогою – **asksaveasfilename**() (рис. 11).

```
from tkinter import *
from tkinter.filedialog import *
import fileinput

root = Tk()
root.title("Filedialog")
root.geometry('300x200+500+200')

# Функція відкриття файлу
def Open_file():
    open_file=askopenfilename()  # Вікно відкриття файлу
    for i in fileinput.input(open_file):
        text.insert(END,i)

# Функція збереження файлу
def Save_file():
    save_file=asksaveasfilename() # Вікно збереження файлу
    readtext=text.get(1.0,END)
    open_writeF=open(save_file,"w")
    open_writeF.write(readtext)
    open_writeF.close()
```

Рис. 11. Функції створення діалогових вікон для відкриття і збереження файлу

Іноді виникає необхідність вивести певне повідомлення про роботу програми в діалоговому вікні, не створюючи спеціального віджета. Пакет *tkinter* містить модуль **messagebox**, який надає доступ до вікон повідомлень. Модуль **messagebox** потрібно імпортувати додатково:

from tkinter import messagebox

Для того, щоб згенерувати вікно повідомлення, потрібно для об'єкта **messagebox** викликати функції **showinfo, showwarning, showerror, askquestion, askyesno, askretrycancel.** Від обраної функції залежить вигляд піктограми у вікні повідомлення. Синтаксис виклику вікна повідомлення такий:

messagebox. функція (заголовок, текст_повідомлення)

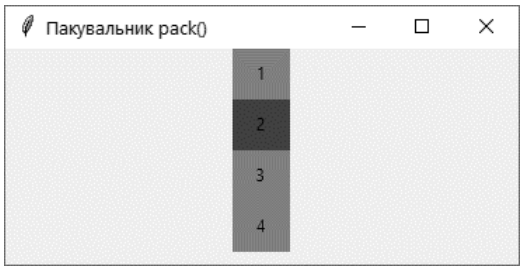
Далі, під час викладу даної теми, ми зупиняємося на питанні розташування віджетів у вікні. Це важливе питання, тому що від інтуїтивності інтерфейсу багато в чому залежить зручність використання програми. Зауважуємо, що для вирішення цієї проблеми використовується **менеджер розташування** (*пакувальник, менеджер геометрії*) –спеціальний механізм (метод), який розміщує віджети у вікні. Розрізняють такі менеджери розташування: метод **pack**(), метод **place**(), метод **grid().** Якщо до елементу інтерфейсу не застосувати будь-який з менеджерів геометрії, то він не буде відображений у вікні. В одному вікні не можна комбінувати різні менеджери.

Як правило пакувальник **pack()** використовують для розміщення віджетів один за одним (зліва направо або зверху вниз). При використанні цього пакувальника за допомогою властивості **side** можна вказати до якої сторони батьківського віджета він повинен примикати. Розглянемо параметри пакувальника **pack():**

- параметр **side**, який приймає значення: **TOP** (за замовчуванням) – розміщує віджет зверху, **BOTTOM** – розміщує віджет знизу, **LEFT** – розміщує віджет ліворуч, **RIGHT** – розміщує віджет праворуч;
- параметр **fill**, який заставляє віджет заповнювати весь доступний простір у вказаному напрямі по одній із осей: **X** – заповнює простір по горизонталі, **Y** – заповнює простір по вертикалі
- параметр **anchor** (якір), який може приймати такі значення: **N** – north (північ), **S** – south (південь), **W** – west (захід), **E** – east (схід). Значення можна комбінувати: SE, NW тощо.

Для прикладу було створено чотири різнокольорові мітки та змінювалось їхнє розташування (рис. 12).

```
from tkinter import*
root = Tk()
root.title("Пакувальник pack()")
label1 = Label(text="1", width=5, height=2, bg='red')
label2 = Label(text="2", width=5, height=2, bg='green')
label3 = Label(text="3", width=5, height=2, bg='blue')
label4 = Label(text="4", width=5, height=2, bg='yellow')
label1.pack() # параметри відсутні
label2.pack()
label3.pack()
label4.pack()
root.mainloop()
```




```
label1.pack(side='bottom')
label2.pack(side='right')
label3.pack(side='left')
label4.pack(side='top')
```


```
label1.pack(fill=X)
label2.pack(fill=X)
label3.pack(fill=X)
label4.pack(fill=X)
```

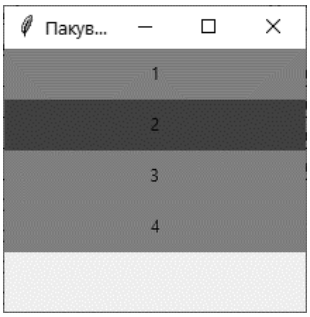


Рис. 12. Приклади розміщення міток у вікні за допомогою пакувалька pack()

Наступний пакувальник – **grid()**. З англійської **grid** перекладається як "сітка". Тож віджети цей пакувальник розміщує у таблиці. Вікно розділяється на рядки та стовпці і кожна комірка (клітинка) в отриманій таблиці може містити віджет. Адреса кожної комірки складається з номера рядка і номера стовпця. Нумерація починається з нуля. Комірки можна об'єднувати як по вертикалі, так і по горизонталі. Пакувальник **grid()** є найбільш гнучким менеджером геометрії в tkinter. Розміщення віджета в тій чи іншій комірці задається через аргументи **row** і **column**, яким присвоюються номер рядка і стовпця відповідно. Щоб об'єднати комірки по горизонталі, використовується атрибут **columnspan**, якому присвоюється кількість поєднуваних комірок. Опція **rowspan** об'єднує комірки по вертикалі. Розглянемо параметри пакувальника **grid()**:

- **row –** номер рядка, в який розміщується віджет;
- **column** – номер стовпця, в який розміщується віджет;
- **columnspan** – кількість стовпців, які займає віджет;
- **rowspan** – кількість рядків, які займає віджет;
- **padx/pady** – розмір зовнішньої межі по горизонталі та вертикалі;
- **ipadx/ipady** – розмір внутрішньої межі по горизонталі і вертикалі (різниця між **pad** і **ipad** в тому, що при вказівці **pad** розширюється вільний простір, а при **ipad** розширюється віджет);
- **sticky** – визначає до якої межі «приклеїти» віджет, може приймати такі значення: 'n' – north (північ), 's' – south (південь), 'w' – west (захід), 'e' – east (схід);

- **in_** – явна вказівка в який батьківський віджет потрібно розмістити.

Після цього розглядається приклад (рис. 13).

```
from tkinter import *
root = Tk()
label1 = Label(root, text="Введіть прізвище:")
label1.grid(row=0, column=0)
label2 = Label(root, text="Введіть ім'я:")
label2.grid(row=1, column=0)
edit1 = Entry(root)
edit1.grid(row=0, column=1)
edit2 = Entry(root)
edit2.grid(row=1, column=1)
root.mainloop()
```

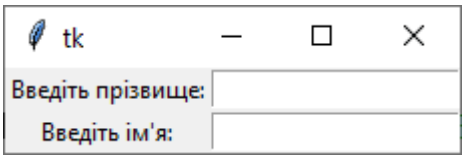


Рис.13. Приклад використання пакувалька grid()

І нарешті розглядається останній пакувальник – **place()**. Він є простим пакувальником, що дозволяє розміщувати віджет у фіксованому місці з фіксованим розміром. При використанні цього пакувальника необхідно вказувати координати кожного віджета. Цей пакувальник, хоч і здається незручним, надає повну свободу в розміщенні віджетів на вікні. Методом **place**() віджету вказується його положення або в абсолютних значеннях (в пікселях), або в частках батьківського вікна, тобто відносно. Також абсолютно і відносно можна задавати розмір самого віджета. До параметрів **place()** належать:

- **anchor** (якір) – визначає частину віджета, для якої задаються координати. Приймає значення N, NE, E, SE, SW, W, NW або CENTER. За замовчуванням NW (верхній лівий кут);
- **relwidth/relheight** – відносні ширина і висота віджета. Визначають розмір віджета відносно батьківського віджета;
- **relx/rely** – визначаєть відносну позицію в батьківському віджеті. Кординати (0;0) – у лівого верхнього кута, (1;1) – у правого нижнього кута;
- **width/height** – абсолютні ширина і висота віджета;
- **x/y** – абсолютні координати (в пікселях) розміщення віджета.

Для прикладу було створено дві мітки синього та жовтого кольору і задано їх розташування. Для першої вказано абсолютні координати, а для другої – відносні (рис. 14).

```
from tkinter import *

root = Tk()
root.geometry('200x200')
root.resizable(0,0)
label1 = Label(bg='blue', width=7, height=3)
label1.place(x=0, y=0)

label2 = Label(bg='yellow', width=7, height=3)
label2.place(relx=0.7, rely=0.7)

root.mainloop()
```

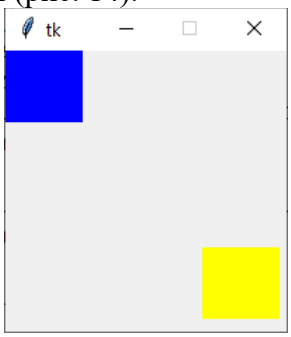


Рис.14. Приклад використання пакувалька place()

І на звершення цієї теми студенти вивчають питання, яке стосується *опрацювання подій*, оскільки додатки з графічним інтерфейсом користувача подійно-орієнтовані. Це означає, що та чи інша частина програмного коду починає виконуватися лише за умови виконання тієї чи іншої події. Зауважимо, що *подією* називається зовнішня дія на віджет. Кожна подія приводить до виконання методу об’єкта або описаної користувачем функції.

Розглядаються події, які застосовуються найчастіше. Їх можна розділити на групи:

1. **Події, що виникають у результаті дії на мишу**. Назви подій цього типу беруть у лапки та знаки <>. Наприклад,

- **<’Button-1’>** – клацання лівою кнопкою миші;

- **<'Button-3'>** – клацання правою кнопкою миші;
- **<'Motion'>** – рух миші.

2. Події, що виникають внаслідок дії на клавіші. Клавіші букв записують у лапках, наприклад, 'W'. Для неалфавітних клавіш існують спеціальні зарезервовані слова. Наприклад, натискання клавіши **Enter** позначається <**Return**>, а клавіша «*пробіл*» – <**space**>. Для клавіш, які натискаються одночасно, теж існують спеціальні позначення. *Наприклад*, натискання клавіш **Ctl+Shift** позначається **'<Control-Shift>'**, а натискання клавіш **Ctrl+z** – **'<Control-z>'**.

3. Події, що виникають внаслідок зміни властивостей інших об'єктів.

Опрацювання подій часто реалізують у вигляді функцій, які викликаються під час виникнення певної події. Функція має таку структуру:

def <назва функції>(<назва події>)
<тіло функції>

Функція створюється, як правило, на початку програмного коду. Функція, пов'язана з певною подією, називається *обробником події*. Щоб можна було виконати функцію опрацювання події, необхідно зв'язати її із самою подією. Зв'язування реалізується за допомогою методу **bind**().

< Ім'я віджета>.**bind**('<Ім'я події>',Ім'я функції)

Наприклад, віджет – кнопка, подія – натиснення на кнопку лівою клавішею миші, дія — зміна кольору фону вікна (рис. 15).

```
from tkinter import*
def button_click(event):
    root['bg'] ='green'
root=Tk()
root.geometry('200x100')
button1=Button(root,text='Колір форми',width=10,font='Arial 16')
button1.bind("<Button-1>",button_click)
button1.pack()
root.mainloop()
```

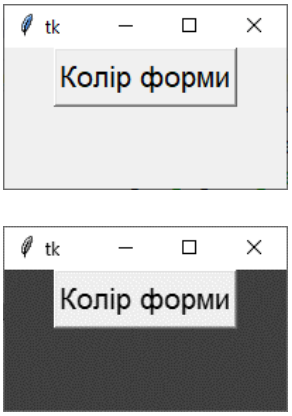


Рис. 15. Приклад створення обробника події натиску лівої кнопки миші

З іншого боку, зв'язування функції з подією можна здійснити без використання методу **bind**(). Наприклад, щоб прикріпити до віджета **Button** обробник події, потрібно при створенні цього об'єкта в переліку атрибутів указати параметр **command** і присвоїти йому посилання на метод, який буде виконуватися при натисканні:

```
from tkinter import*
def button_click():
    root['bg'] ='green'
root=Tk()
root.geometry('200x100')
button1=Button(root,text='Колір форми',width=10,font='arial 16', command = button_click)
button1.pack()
root.mainloop()
```

При використанні методу **bind**() може бути вказана будь-яка подія, а під час запису властивості **command** кнопки – тільки клік лівої кнопки миші.

Зауважимо, що практична підготовка студентів з даної теми здійснювалась на практичних і лабораторних заняттях. Зокрема, відповідно до робочих програм з інформатики студенти спеціальностей 111 Математика та 014.04 Середня освіта (Математика) виконували такі лабораторні роботи:

- Python. Модуль tkinter. Графічні об'єкти (віджети) та їх властивості;
- Python. Модуль tkinter. Опрацювання подій;
- Python. Модуль tkinter. Графічні примітиви;

– Python. Модуль tkinter. Створення меню;
– Python. Модуль tkinter. Діалогові вікна.

Кожна лабораторна робота складається з теоретичних відомостей з прикладами, індивідуальних завдань та контрольних запитань.

Зазначимо, що для студентів математичних спеціальностей Вінницького державного педагогічного університету нами розроблені лекції, тексти лабораторних і практичних робіт з інформатики. Їх можна знайти на персональному сайті, який описано в статті [2].

3. ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Таким чином, в статті продемонстровано методичні особливості створення графічного інтерфейсу користувача під час вивчення програмування (на прикладі мови Python), яке є необхідним для формування інформатичної компетентності майбутніх учителів математики. Тільки комплексний підхід, який поєднує теоретичну підготовку (передбачає детальний виклад теоретичного матеріалу із наочними прикладами) на лекційних заняттях та практичну підготовку на практичних і лабораторних заняттях (з індивідуальними завданнями), може сприяти якісному засвоєнню знань, набуттю практичних вмінь і навичок, здатності формулювати і розв’язувати практичні задачі у професійній діяльності.

Подальші дослідження можуть бути спрямовані на створення лабораторного практикуму з основ програмування мовою Python, який можна запропонувати майбутнім учителям математики для організації самостійної навчально-пізнавальної діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1.] Ковтонюк Г. М. До питання формування інформатичної компетентності майбутніх учителів фізико-математичних дисциплін. *Нова педагогічна думка*. 2017. Том 91, № 3. С. 49-51.
[2.] Ковтонюк Г. М. Персональний сайт викладача як ефективний засіб організації самостійної пізнавальної діяльності майбутніх учителів фізико-математичних дисциплін. *Фізико-математична освіта*. 2017. Вип. 14, № 4. С. 205-208.
[3.] Креневич А. П. Python у прикладах і задачах. Ч. 1. Структурне програмування: навч. посіб. Київ: ВПЦ «Київський університет», 2017. 206 с.
[4.] Креневич А. П. Python у прикладах і задачах. Ч. 2. Об’єктно-орієнтоване програмування: навч. посіб. Київ: ВПЦ «Київський університет», 2020. 152 с.
[5.] Навчальні програми для 10-11 класів. URL: https://mon.gov.ua/ua/osvita/zagalna-serednya-osvita/navchalni-programi/navchalni-programi-dlya-10-11-klasiv (дата звернення: 01.09.2021).
[6.] Руденко В. Д., Жугастров О. О. Основи алгоритмізації і програмування мовою Python. Харків: Вид-во «Ранок», 2019. 192 с.
[7.] Панченко О., Ковтонюк Г. До питання вивчення основ програмування мовою Python майбутніми вчителями математики. Матеріали ІІ Всеукраїнської науково-практичної Інтернет-конференції «Математика та інформатика у вищій школі: виклики сучасності» (Вінниця, 15-16 травня 2019 р.) [Електронне наукове видання] : збірник матеріалів. Вінниця, 2019. С. 122-125.

FEATURES OF CREATING A GRAPHICAL USER INTERFACE DURING THE STUDY OF PYTHON PROGRAMMING BY FUTURE TEACHERS OF MATHEMATICS

Bak Serhii Mykolaiovych
Doctor of Physical and Mathematical Sciences, Professor at the Department of Mathematics and Informatics, Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University,
Vinnytsia, Ukraine
ORCID ID: 0000-0003-1508-2144
sergiy.bak@gmail.com

Kovtoniuk Halyna Mykolaivna
Candidate of Pedagogical Sciences, Associate Professor at the Department of Mathematics and Informatics, Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University,
Vinnytsia, Ukraine
ORCID ID: 0000-0002-3352-0358
galyna.kovtonyuk@gmail.com

Abstract. The article is devoted to the methodological aspects of studying the features of creating a graphical user interface during the study of programming (on the example of Python), which is necessary for the formation of informatical competence of future teachers of mathematics. In particular, the article demonstrates the author's methodological approach to the study of this topic, which provides comprehensive theoretical and practical training. Theoretical training is provided in lectures and involves, above all, mastering the essence of the concept of "graphical user interface". To create a graphical user interface, students are offered to use the Tkinter module, which is part of the standard Python library and has a very powerful graphical capabilities. This module has a standard set of objects (visual controls or widgets) that create a graphical interface. Next reveals the essence of the concepts: class, property and method. The algorithm for creating a graphical user interface is given. The main widgets (Button, Label, Entry, Text, Radiobutton, Checkbutton, Listbox, Menu, etc.) and their corresponding properties and methods are studied. Examples of programs with these widgets are given. Possibilities of creation of dialog windows are considered. Special attention is paid to the location of widgets in the window. For this, so-called location managers are used. Three main location managers are studied: pack (), place (), grid (). Examples of their application are given. At the end of this topic, the question concerning the treatment of events is studied. The essence of the concept of event is clarified and their classification is given. Describes how to create event handlers and associate them with the events themselves. Relevant examples are given. Practical training of students on this topic is carried out in practical and laboratory classes. The latter provide for the presence of individual tasks. Such a comprehensive approach, which combines theoretical and practical training, can contribute to the quality of knowledge acquisition, the acquisition of practical skills, the ability to formulate and solve practical problems in professional activities.

Keywords: training of teachers of mathematics, informatical competence, programming, Python, graphical user interface, widgets.

References (TRANSLATED AND TRANSLITERATED)

[1.] Kovtoniuk H. M. On the question of the formation of the informatical competence of future teachers of physical and mathematical disciplines. *Nova pedahohichna dumka*. 2017. Vol. 91, № 3. P. 49-51. (in Ukrainian)

[2.] Kovtoniuk H. M. Personal website of a teacher as an effective means of organizing independent cognitive activity of future teachers of physical and mathematical disciplines. *Physical and Mathematical Education*. 2017. Vol. 14, № 4. P. 205-208. (in Ukrainian)

[3.] Krenevych A. P. Python in examples and problems. Part 1. Structural programming: textbook. Kyiv: VPTs «Kyivskyi universytet», 2017. 206 p. (in Ukrainian)

[4.] Krenevych A. P. Python in examples and problems. Part 2. Object-oriented programming: textbook. Kyiv: VPTs «Kyivskyi universytet», 2020. 152 p. (in Ukrainian)

[5.] Study programs for grades 10-11. URL: https://mon.gov.ua/ua/osvita/zagalna-serednya-osvita/navchalni-programi/navchalni-programi-dlya-10-11-klasiv. (in Ukrainian)

[6.] Rudenko V. D., Zhuhastrov O. O. Fundamentals of algorithmization and programming in Python. Kharkiv: «Ranok», 2019. 192 p. (in Ukrainian)

[7.] Panchenko O., Kovtoniuk H. To the question of learning the basics of programming in Python by future teachers of mathematics. Mathematics and Informatics in Higher Education: Challenges of the Present : Materials of II Ukr. Scien. Confer. (Vinnytsia, 2019 May 15-16). Vinnytsia, 2019. P. 122-125. (in Ukrainian)