

---

# Програмування

---

## ТЕМА 20. ГРАФІЧНИЙ ІНТЕРФЕЙС

# Графічний інтерфейс

---

**Інтерфейс** – це засоби та правила зв'язку між двома сутностями.

Існують інтерфейси різноманітних апаратних засобів, інтерфейси для зв'язування між собою програм тощо.

Але найчастіше під словом «інтерфейс» мають на увазі засоби та правила зв'язку між користувачем та програмою.

У таких випадках кажуть про «інтерфейс користувача».

Інтерфейс користувача пройшов еволюційний шлях, який продовжується й зараз.

Доволі довго домінуючим був командний інтерфейс або інтерфейс командного рядка.

Цей інтерфейс передбачає подання команд користувачем та виконання цих команд програмою.

Команда - це рядок, що й обумовило назву командного інтерфейсу.

# Графічний інтерфейс.2

---

Із зростанням потужності комп'ютерів з'явилась можливість запропонувати замість командного графічний інтерфейс.

**Графічний інтерфейс користувача** (Graphical User Interface або GUI) передбачає спілкування користувача з програмою за допомогою графічних примітивів та дій над цими примітивами з використанням клавіатури та миші.

У сучасних мобільних пристроях замість миші використовують пальці рук.

З точки зору користувача, графічний інтерфейс є більш «дружнім», ніж командний.

Продовжуються також дослідження та розробки нових видів інтерфейсу: голосового, зорового тощо.

# Програмування, що керується подіями

---

Написання програм, що використовують графічний інтерфейс, відрізняється від написання програм з командним інтерфейсом.

Усі попередні розглянуті нами програми мали командний інтерфейс та були побудовані наступним чином: введення даних – обробка – виведення результатів.

Тобто, виконання відбувається послідовно. Такі програми ще називають консольними.

Програми, що мають графічний інтерфейс, побудовані інакше.

Спочатку в них створюються елементи графічного інтерфейсу, а потім починається умовно нескінченний головний цикл, у якому програма чекає на дії користувача.

Кожна дія користувача з елементом графічного інтерфейсу (наприклад, натиснення кнопки, введення тексту, вибір пункту меню) ініціює подію. Тому виконання програм з графічним інтерфейсом суттєво відрізняється від програм з командним інтерфейсом

# Програмування, що керується подіями.2

---

Предбачити послідовність дій користувача з графічним інтерфейсом найчастіше неможливо, та й непотрібно.

Тому виконання програм з графічним інтерфейсом суттєво відрізняється від програм з командним інтерфейсом.

Для обробки подій, що виникають в результаті дій користувача, програміст пише функцію обробки події.

Таким чином, програма з графічним інтерфейсом містить частину ініціалізації (створення елементів інтерфейсу) та набір функцій обробки подій

# Програмування, що керується подіями.3

---

Головний цикл є умовно нескінченним, тому що він все ж закінчується коли користувач закриває вікно на екрані.

Якщо послідовні програми можна порівняти із забігом, то програми, що керуються подіями, - зі стрибками у ширину.

Дійсно, коли буде викликана та чи інша функція обробки події, спрогнозувати неможливо.

Тому всі функції повинні бути незалежними, але, як правило, мають обмінюватись інформацією.

Важливо також, щоб функція обробки події не перебирала на себе управління надовго, оскільки у цьому випадку, графічний інтерфейс перестає реагувати на дії користувача та немов би «зависає»

# Графічні бібліотеки у Python

---

Для Python написано декілька бібліотек, що підтримують роботу з графікою.

tkinter – кросплатформений графічний пакет на базі відомої бібліотеки Tk (Tcl).

Tk може використовуватись разом з різними мовами програмування (Perl, Ruby, PHP, Common Lisp, Tcl), в тому числі, й Python.

Tkinter дозволяє як будувати графічний інтерфейс, так і зображувати графіку на екрані. До речі, розглянута нами раніше бібліотека turtle побудована на базі tkinter.

Однією з переваг tkinter є те, що цей пакет включений у стандартну поставку Python.

PyQt – адаптація відомої бібліотеки Qt до Python.

Qt може працювати у операційних системах Windows, Mac OS X, Unix та Linux. Qt відома, у тому числі, як основа для інтерфейсу KDE операційних систем Linux.

У порівнянні з tkinter, PyQt є більш складною, але й більш функціональною бібліотекою.

# Графічні бібліотеки у Python.2

---

wxPython – пакет для використання у Python графічної бібліотеки wxWidgets.

Так само, як і PyQt, wxPython доступний на різних платформах та надає широкий спектр можливостей з побудови графічного інтерфейсу.

wxPython має декілька реалізацій спеціальних програм-дизайнерів графічного інтерфейсу.

У порівнянні з tkinter, wxPython, як і PyQt, є складнішим.

PyGTK – графічна бібліотека, що поєднує Python та GTK – основу для відомого проекту Gnome інтерфейсу для Linux-систем.

Розробку PyGTK припинено у 2011 році, та замінено іншою бібліотекою: GObject.

Наведений перелік графічних бібліотек є далеко не повним та коротко описує лише найбільш розповсюджені засоби побудови графічного інтерфейсу у Python.

У цій темі ми більш докладно розглянемо використання tkinter.



# Початок роботи з tkinter.

## ОСНОВНІ ПОНЯТТЯ

---

Для того, щоб почати використовувати tkinter, слід імпортувати цей модуль командою

```
import tkinter
```

або

```
from tkinter import *
```

Графічний інтерфейс tkinter складається з вікон, які у термінах tkinter називають віджетами.

**Віджет** – це елемент графічного інтерфейсу, такий, як кнопка, вікно тексту, надпис або список.

Під віджетом мають на увазі не тільки сам елемент інтерфейсу, але й програмну компоненту, яка підтримує його функціонування.

# Початок роботи з tkinter.

## Основні поняття.2

---

tkinter містить більше десятка віджетів, кожен з яких – це окремий клас.

Отже, як і будь-який клас, віджет має властивості та методи.

Але більша частина властивостей віджетів доступна у вигляді елементів словника, де ключ – ім'я властивості, а значення – значення властивості.

Віджети поділяються на звичайні віджети та контейнери.

Контейнери можуть містити інші віджети.

# Основні віджети

Віджет	Опис
Button	Кнопка команд
Canvas	Полотно, дозволяє зображувати фігури, текст та готові зображення
Checkbutton	Кнопка вибору або «прапорець», що може бути вибраним або не вибраним
Entry	Поле введення тексту з одного рядка
Frame	Рамка, може містити інші віджети, контейнер
Label	Надпис, містить статичний текст або зображення
LabelFrame	Рамка з заголовком, може містити інші віджети, контейнер
Listbox	Список, складається з елементів, які можуть бути

# Основні віджети.2

Віджет	Опис
Menubutton	Містить меню (що випадає або спливає)
Message	Повідомлення, те ж саме, що й Label, але розміщує текст у декількох рядках
Radiobutton	Набір «радіокнопок», з яких тільки одна може бути натиснута у кожен момент часу
Scale	Повзунок, дає можливість позначити числове значення на шкалі
Scrollbar	Лінійка прокрутки для перегляду вмісту списків та інших віджетів
Text	Багаторядкове вікно тексту
Toplevel	Окреме вікно, яке може містити віджети, контейнер

# Кроки виконання програми, яка використовує tkinter

---

Типова програма, що використовує tkinter, проходить такі кроки:

1. Створення головного вікна
2. Створення віджетів та ініціалізація даних.
3. Зв'язування подій з функціями обробки.
4. Розміщення віджетів у вікні (вікнах)
5. Запуск головного циклу

Створити головне вікно – це створити об'єкт класу Tk, наприклад:

**top = Tk()**

Створення віджетів – це створення об'єктів відповідних класів, встановлення їх властивостей.

# Кроки виконання програми, яка використовує tkinter.2

---

Зв'язування подій з функціями обробки встановлює, яка функція буде викликана при виникненні тієї або іншої події.

Достатньо зв'язати з функціями обробки тільки ті події, у яких треба щось змінити, оскільки стандартну реакцію на події забезпечують функції самого tkinter.

Розміщення віджетів у вікні визначає, у якій позиції буде відображено той чи інший віджет.

Цю задачу виконують менеджери розміщення (layout managers), з якими ми познайомимось пізніше

# Кроки виконання програми, яка використовує tkinter.3

---

Запуск головного циклу виконується методом `mainloop`:

## **top.mainloop()**

Після запуску головного циклу програма очікує на дії користувача.

Програма буде продовжена (завершена) після закриття головного вікна.

Щоб завершити головний цикл, треба викликати метод `quit` головного вікна:

## **top.quit()**

# Ієрархія вкладень віджетів

---

Кожен віджет розміщується або у вікні верхнього рівня, або у контейнері (наприклад, у рамці).

Контейнери також можуть розміщуватись у інших контейнерах.

Таким чином, з точки зору розміщення та вкладення віджети утворюють ієрархію.

Не треба її плутати з ієрархією класів, оскільки один віджет не є нащадком іншого.

Вікно (клас), у якому розміщується віджет, вказують першим параметром конструктора будь-якого віджета.

Відповідна властивість, значення якої можна прочитати, називається `master`.



# Надпис (Label)

---

Щоб створити віджет Label, треба викликати конструктор відповідного класу, наприклад:

**lc = Label(top, text = "abc")**

Ключовий параметр text визначає рядок, що буде відображено у надпису.

# Приклад: введення початкових даних та обчислення результату функції (Версія 1)

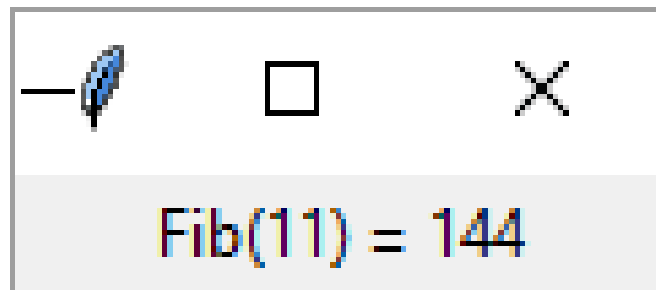
---

Зробити графічний інтерфейс для введення значення параметру  $n$  та обчислення  $n$ -го числа Фібоначчі.

Версія 1 програми тільки виводить результат виклику функції `fib` у надписі (Label).

Початкові дані вводяться, як звичайно.

Розміщення віджету Label у вікні виконується методом `pack`.



# Кнопка команд (Button) та поле введення (Entry)

---

Щоб створити кнопку команд, треба викликати конструктор класу Button.

Ключовий параметр text визначає рядок, що буде відображено на кнопці, а ключовий параметр command задає ім'я функції обробки події, яку буде викликано, коли кнопку буде натиснуто.

```
bcalc = Button(top, text='Обчислити',  
               command=calc)
```

Для створення поля введення треба викликати конструктор класу Entry, наприклад:

```
ein = Entry(top)
```

Отримати значення рядка, який введено у полі, можна за допомогою метода get():

```
s = ein.get()
```

# Встановлення відображення графічних елементів

---

tkinter дозволяє встановлювати власне відображення графічних елементів: шрифт, колір тексту та колір фону. Для встановлення треба вказати при створенні відповідного віджета ключові параметри:

- fg – колір тексту
- bg – колір фону
- font - шрифт

Кольори тексту та фону задають рядками – англійськими назвами кольорів.

Колір також можна задати рядком у форматі за основою 16.

Наприклад блакитний колір можна задати рядком «cyan» або рядком "#007F7F".

# Встановлення відображення графічних елементів.2

---

Шрифт встановлюють кортежем з трьома полями: (<назва>, <розмір>, <написання>), де назва – це ім'я шрифту, написання – чи є шрифт напівгрубим або нахиленим.

Відповідні позначення:

- 'bold' – напівгрубий шрифт;
- 'italic' – нахилений шрифт;
- 'bold italic' – напівгрубий нахилений;
- 'normal' – нормальний, можна не вказувати.

# Встановлення відображення графічних елементів.3

---

Так, кортеж:

**('arial', 16, 'bold')**

означає напівгрубий шрифт Arial розміром 16.

Наприклад, створити поле введення з зеленим фоном, червоним нахиленим шрифтом розміром 16 можна так:

**ein = Entry(top, fg = 'red', bg='green', font=('arial', 16, 'italic'))**

Для виділення віджета у вікні можна вказати «рельєф» ключовим параметром relief разом з параметром, що встановлює ширину границі – bd.

Можливими значеннями relief є FLAT (без виділення), SUNKEN (заглиблений), RAISED (піднятий), GROOVE (з вдавненою рамкою), RIDGE (з піднятою рамкою) або SOLID (з рамкою, зображеною іншим кольором). Значення за угодою - FLAT.

Наприклад:

**ledit = Label(top, bd=1, relief=SUNKEN)**

# Модифікація параметрів графічних елементів у динаміці

---

Під час роботи програми з графічним інтерфейсом часто виникає потреба змінити параметри деякого елемента як реакцію на подію, що виникла.

Наприклад, змінити текст надпису результату функції після обчислення функції для нового значення аргументу.

Для зміни будь-яких параметрів, що передаються як ключові параметри при створення віджетів, використовують метод `configure` (або `config`).

У цьому методі вказують нові значення потрібних параметрів так само, як це робиться під час створення нового віджета

# Модифікація параметрів графічних елементів у динаміці.2

---

Наприклад:

**`Irez.configure(text = result)`**

Оскільки параметри зберігаються у словнику, для їх зміни можна також використати відоме позначення для словників – квадратні дужки [ ].

У даному випадку:

**`Irez['text'] = result`**

Відмінність використання методу `configure` від нотації словників полягає у тому, що у методі за один виклик можна змінити значення декількох параметрів



# Приклад: введення початкових даних та обчислення результату функції (Версія 2)

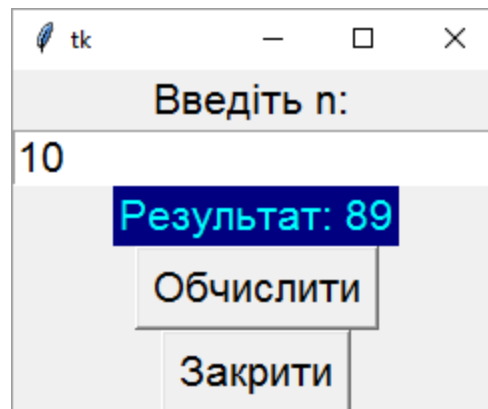
---

Зробити графічний інтерфейс для введення значення параметру  $n$  та обчислення  $n$ -го числа Фібоначчі.

Версія 2 програми вводить значення  $n$  у полі введення та виводить результат виклику функції `fib` у надписі (Label).

Обчислення здійснюється після натиснення кнопки «Обчислити».

Кнопка «Закрити» завершує роботу програми. Також, визначається розмір шрифту та кольори надпису.



# Менеджери розміщення

---

**Менеджери розміщення** – це програмні компоненти у складі `tkinter`, які визначають розміщення графічних елементів у їх контейнерах.

`tkinter` має три різних менеджери розміщення:

- `pack` – пакувальник, розміщує елементи за їх положенням відносно інших елементів;
- `grid` – таблиця, розміщує елементи як у таблиці, за вказаними рядком та стовпчиком;
- `place` – розміщення по заданих позиціях.

# Менеджер розміщення pack

---

Менеджер розміщення pack дає можливість вказати відносне місце для кожного графічного елементу (віджета).

Основні параметри, якими можна керувати:

- side – сторона, біля якої треба розмістити елемент
- fill – чи заповнює елемент вільне місце по осях X, Y
- expand – чи розширюється елемент після зміни розміру вікна

Можливі значення параметру side: TOP (верх), BOTTOM (низ), LEFT (ліва сторона), RIGHT (права сторона). Значення за угодою – TOP.

Можливі значення параметру fill: X (заповнення по осі x), Y (заповнення по осі y), BOTH (заповнення по осях x та y), NONE (немає заповнення). Значення за угодою – NONE.

Можливі значення параметру expand: 1 або '1' або YES (елемент розширюється), 0 або '0' або NO (елемент не розширюється). Значення за угодою – 0.

# Менеджер розміщення pack.2

---

Для розміщення використовується метод pack, наприклад:

**fbut.pack(side=LEFT, fill=X, expand='1')**

Менеджер розміщення pack розміщує елементи послідовно у порядку викликів методу pack.

Кожний наступний елемент розміщується у відповідності з заданими параметрами та з урахуванням місця, що залишилось після розміщення попередніх елементів

# Менеджер розміщення pack.3

---

Початкові розміри контейнеру обчислюються, виходячи з місця, яке потрібно для розміщення всіх елементів.

При зменшенні розмірів вікна менеджер розміщення першими зменшує (закриває) ті елементи, що розміщено останніми.

Це треба враховувати при визначенні порядку розміщення елементів.

Окрім основних можна задавати також додаткові параметри для керування пакуванням.

Так параметри  $radx$  та  $radu$  визначають відступ даного віджета від інших відповідно по осях  $x$  та  $y$ .

# Рамка (Frame)

---

Рамка виконує роль універсального контейнеру для інших віджетів та містить, як правило, декілька графічних елементів.

Рамка, частіше за все, не є видимою у вікні, оскільки її колір фону співпадає з кольором фону вікна.

Рамки часто використовують разом з менеджером розміщення pack для завдання потрібного зовнішнього вигляду інтерфейсу.

Щоб створити рамку, треба викликати конструктор класу Frame.

**fbut = Frame(top)**

# Створення та пакування елементів однією командою

---

Деякі віджети нам потрібні лише для того, щоб їх створити та розмістити у вікні.

У подальшому ми не будемо до них звертатись у програмі.

Це може бути характерно для надписів, кнопок та інших елементів.

У цьому випадку ми можемо не присвоювати значення об'єкта якійсь змінній, а відразу створювати та пакувати елемент.

Наприклад:

```
Label(fininput, text='Введіть n: ', font=('arial',  
16)).pack(side=LEFT)
```

# Приклад: введення початкових даних та обчислення результату функції (Версія 3)

---

Зробити графічний інтерфейс для введення значення параметру  $n$  та обчислення  $n$ -го числа Фібоначчі.

Версія 3 програми вводить значення  $n$  у полі введення та виводить результат виклику функції `fib` у надписі (Label).

Обчислення здійснюється після натиснення кнопки «Обчислити».

Кнопка «Закрити» завершує роботу програми.

Також, визначається розмір шрифту та кольори надпису.

У версії 3 здійснюється розміщення надпису з запрошенням введення та поля введення у одному рядку.

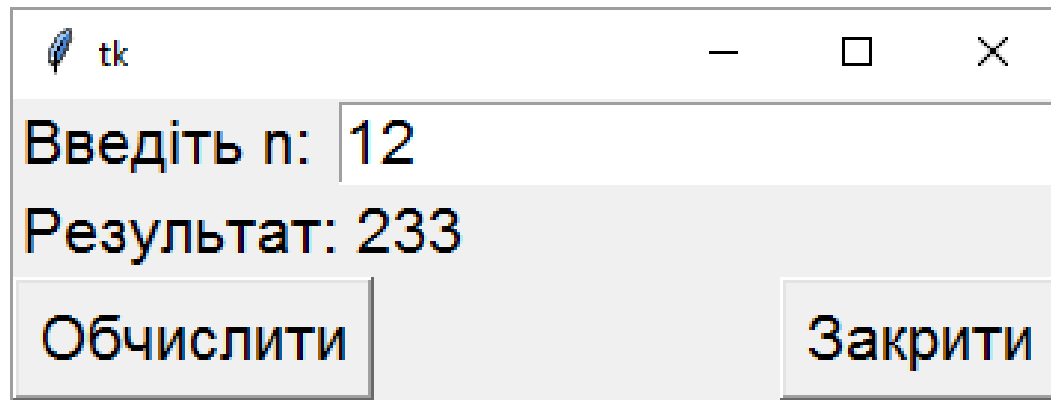
Надпис з результатом розміщується у наступному рядку та вирівнюється по лівому краю.

Обидві кнопки також розміщуються у одному рядку по лівій та правій стороні.



## Приклад: введення початкових даних та обчислення результату функції (Версія 3).2

---



# Прив'язка подій до функцій обробки

---

Ми вже зустрілись з призначенням функції обробки події для події натиснення кнопки команд.

Але існує ще велика кількість подій, на які можна реагувати при роботі з графічним інтерфейсом у tkinter.

Для того, щоб прив'язати функцію обробки до події у деякому віджеті, використовують метод `bind`.

Наприклад, наступний виклик `bind` прив'язує у віджеті `top` (та його вкладених віджетах) функцію `calc` до події натиснення клавіші `Enter` на клавіатурі.

**`top.bind('<Return>', calc)`**

На відміну від функції обробки команди, функція обробки події, яку прив'язано за допомогою `bind`, повинна мати один параметр.

Коли відповідна подія виникає у віджеті, ця функція викликається та їй передається в якості параметру об'єкт `event`, що містить характеристики події.

# Прив'язка подій до функцій обробки.2

Позначення події	Опис
<Button-1>	Натиснення лівої клавіші миші. Button 2 – середня клавіша (якщо є), а Button 3 – права клавіша.
<B1-Motion>	Миша переміщується з натиснутою лівою клавішою. Використовується для перенесення об'єктів.
<ButtonRelease-1>	Ліва клавіша миші відпущена.
<Double-Button-1>	Ліва клавіша миші натиснута двічі.
<Enter>	Курсор миші зайшов у область віджета.
<Leave>	Курсор миші вийшов з області віджета.
<FocusIn>	Віджет отримав фокус (сприймає натиснення клавіш як події)
<FocusOut>	Фокус перейшов від віджета

# Прив'язка подій до функцій обробки.3

Позначення події	Опис
<Return>	Користувач натиснув клавішу Enter. Окрім цієї клавіші фіксується також натиснення інших клавіш: BackSpace (видалення останнього символу), Tab (табуляція), Shift_L (клавіша Shift), Control_L (клавіша Control), Alt_L (клавіша Alt), Caps_Lock (верхній регістр), Escape (клавіша Esc), Prior (Page Up), Next (Page Down), End, Home, Left (стрілка вліво), Up (стрілка вгору), Right (стрілка вправо), Down (стрілка вниз) та інші
<Key>	Користувач натиснув будь-яку клавішу. Код відповідного символу передається у функцію обробки як атрибут char об'єкту event. Для клавіш керування передається порожній рядок.
a	Користувач натиснув клавішу "а". Аналогічні події існують і для інших клавіш, що друкуються.
<Configure>	Віджет змінив розміри або місцезнаходження. Нові розміри передаються у функцію обробки як атрибути width, height об'єкту event.
<Destroy>	Віджет знищується та його вікно закривається.

# Прив'язка подій до функцій обробки.4

Атрибути об'єкту event також наведені у таблиці

Атрибут	Опис
<b>widget</b>	Віджет, що згенерував подію
<b>x, y</b>	Поточна позиція курсора миші у пікселях
<b>x_root, y_root</b>	Поточна позиція курсора миші у пікселях відносно лівого верхнього кута екрану
<b>char</b>	Символ, що натиснуто на клавіатурі (для подій, пов'язаних з клавіатурою)
<b>width, height</b>	Новий розмір віджета у пікселях (для події Configure).

## Приклад: введення початкових даних та обчислення результату функції (Версія 4)

---

Зробити графічний інтерфейс для введення значення параметру  $n$  та обчислення  $n$ -го числа Фібоначчі.

У версії 4 кнопки розміщуються з відступом у 5 пікселів.

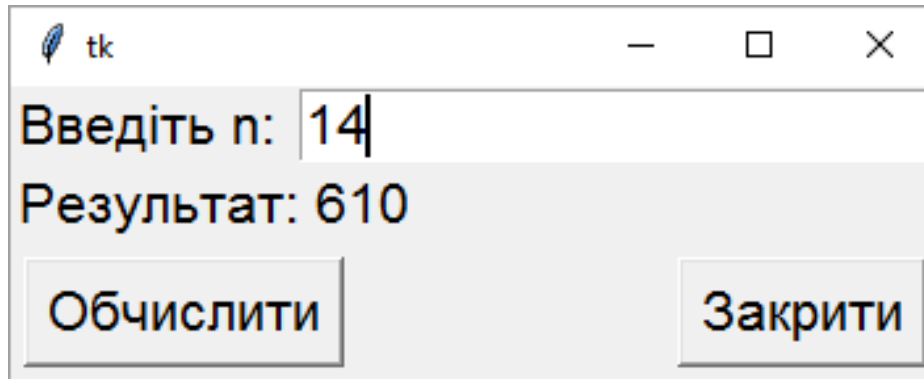
Також зв'язано натиснення клавіш та дії з вікном: натиснення клавіші Enter рівносильно натисненню кнопки «Обчислити», а натиснення клавіші Esc – натисненню кнопки «Закрити».

Окрім цього, поле введення номера числа Фібоначчі `ein` відразу отримує фокус за допомогою методу `focus()`.

**ein.focus()**

Приклад: введення початкових даних та обчислення результату функції (Версія 4).2

---



tk

Введіть n: 14

Результат: 610

Обчислити Закрити

# Графічний інтерфейс та об'єктно-орієнтоване програмування

---

При побудові графічних інтерфейсів з використанням tkinter часто використовують об'єктно-орієнтований підхід.

Сам tkinter надає об'єктно-орієнтований інтерфейс, усі віджети є класами.

Але використання об'єктно-орієнтованого програмування не обмежується тільки створенням стандартних класів та викликом методів tkinter.

Коли розробляють графічний інтерфейс, часто для кожного окремого вікна створюють власний клас.



# Графічний інтерфейс та об'єктно-орієнтоване програмування 2

---

Такий підхід обумовлено декількома факторами.

- По-перше, кожне вікно є окремою сутністю, що має свої властивості та поведінку, тобто повністю відповідає означенню об'єкту.
- По-друге, створення класу для вікна полегшує обмін інформацією між окремими віджетами та між вікном та зовнішнім світом. Зокрема, через поля та методи класу.
- По-третє, гарно спроектований клас може бути повторно використаний у інших програмах без його зміни.

Тому надалі ми будемо описувати клас для кожного окремого вікна.

# Графічний інтерфейс та об'єктно-орієнтоване програмування.3

---

Зв'язок власних класів графічного інтерфейсу та віджетів `tkinter` може бути оформлений двома способами: шляхом включення об'єктів `tkinter` в якості полів власних класів або шляхом наслідування власних класів від віджетів `tkinter`.

Кожний спосіб має плюси та мінуси.

Включення об'єктів в якості полів простіше та дозволяє включати одразу декілька віджетів у один клас.

З іншого боку, наслідування дозволяє будувати власні віджети з особливими характеристиками як нащадки віджетів `tkinter`, хоче цей спосіб є складнішим через необхідність дотримуватись правил виклику методів батьківського класу та обережності при перевизначенні

# Графічний інтерфейс та об'єктно-орієнтоване програмування.4

---

Якщо для вікна графічного інтерфейсу описано клас, віджети повинні створюватись у конструкторі цього класу.

Окрім конструктора типовий клас містить методи для обробки подій та методи для обміну інформацією між вікном та зовнішнім світом.

Слід також розділяти у конструкторі програмний код, пов'язаний зі створенням віджетів та інший програмний код, щоб не перевантажувати конструктор.

Це досягається описом окремих внутрішніх методів для створення (та можливо й для розміщення) віджетів, які називають, наприклад, `_make_widgets`

# Менеджер розміщення grid

---

Менеджер розміщення grid представляє вікно як таблицю.

У клітинках цієї таблиці менеджер розміщує графічні елементи.

Загальна кількість рядків та стовпчиків таблиці визначається після завершення розміщення усіх елементів.

Для розміщення елемента у цьому менеджері використовують метод grid:

**bcancel.grid(row=0,column=1)**

Ключовий параметр row задає рядок (починаючи з 0), а ключовий параметр column, - стовпчик (починаючи з 0).

# Менеджер розміщення grid.2

---

Для вирівнювання елемента у клітинці таблиці використовують ключовий параметр `sticky` (липнути).

Значеннями цього параметру є кортежі, що включають від 1 до 4 полів.

Поля вказують сторони світу, до яких треба наближувати елемент:

N – north – північ – верхня сторона

S – south – південь – нижня сторона

W – west – захід – ліва сторона

E – east – схід – права сторона

# Менеджер розміщення grid.3

---

Наприклад, `sticky=(N, W)` означає, що відповідний елемент треба розмістити у лівому верхньому куті клітинки таблиці.

Виклик методу

**`bcancel.grid(row=0, column=1, sticky=(E), padx=5, pady=5)`**

розміщує `bcancel` у першому рядку та другому стовпчику з вирівнюванням праворуч та відступами по 5 пікселів.

Для зміни параметрів розміщення рядка або стовпчика таблиці використовують методи `rowconfigure` та `columnconfigure`.

# Менеджер розміщення grid.4

---

Один з ключових параметрів розміщення, який треба міняти для того, щоб віджети масштабувались разом зі змінами розмірів вікна, є параметр `weight` (вага).

Наприклад, виклики

**`self.fedit.columnconfigure(0, weight=1)`**

**`self.fedit.columnconfigure(1, weight=2)`**

призводять до того, що при розширенні вікна другий стовпчик розширюється вдвічі швидше за перший.

В одному вікні не можна одночасно використовувати два різних менеджери розміщення.

Тобто, треба використовувати або `pack` або `grid`.

# Змінні tkinter

---

Для обміну інформацією між віджетами та зовнішнім світом tkinter використовує спеціальні «змінні».

Слово змінні тут взято в лапки, оскільки це не звичайні змінні, а об'єкти класів, що описані у tkinter.

Змінні можуть мати різні типи. Відповідні класи називаються StringVar (рядок), IntVar (цілий), DoubleVar (дійсний) та BooleanVar (бульовий).

Після створення змінної (виклику конструктора), наприклад, `s = StringVar()`, цю змінну можна зв'язати з деяким віджетом.



# Змінні tkinter.2

---

Зв'язана змінна відслідковує усі модифікації, що відбуваються. Отримати значення змінної tkinter можна за допомогою методу `get()`, наприклад:

## **s.get()**

Якщо ж змінити значення змінної tkinter методом `set()`, це нове значення відобразиться у зв'язаному віджеті. Виклик:

## **s.set(value)**

Зв'язування відбувається під час створення віджета. Для поля введення таке зв'язування задається ключовим параметром `textvariable`:

**entry = Entry(top, textvariable = s)**

# Приклад: Конструктор рецептів

---

Скласти програму з графічним інтерфейсом для конструювання та збереження рецептів страв за їх інгредієнтами.

# Клас DictEditor

---

Клас DictEditor створює графічний інтерфейс для редагування даних довільного словника.

Для кожного елемента словника створюється пара віджетів: надпис, у який записується ключ елемента, та поле введення, у яке записується значення елемента.

Це значення може бути змінено користувачем.

DictEditor використовує менеджер grid для розміщення елементів.

Клас має такі поля:

- `self.master` - вікно, у якому розміщується вікно редагування.
- `self.dct` - словник, що редагується
- `self.has_buttons` - чи є власні кнопки у вікна редагування
- `self.vars` - словник з текстовими змінними tkinter для зв'язування з полями введення
- `self.labels` - словник з надписами
- `self.entries` - словник з полями введення

# Клас DictEditor.2

---

Якщо у вікна редагування є власні кнопки, то додаються кнопки «Ok» та «Відмінити», які завершують редагування відповідно із збереженням та відміною результатів редагування.

Результати зберігаються у словнику `self.vars`.

Клас має конструктор `__init__`, який викликає внутрішній метод `_make_widgets` для створення віджетів.

Метод `_make_widgets`, у свою чергу, викликає внутрішні методи `_make_entries` для створення надписів та полів введення а також `_layout_entries` для розміщення створених надписів та полів введення.

Методи `ok_handler` та `cancel_handler` обробляють натиснення кнопок «Ok» та «Відмінити» відповідно.

Метод `get` повертає відредагований словник.

Цей метод, як правило, викликається після завершення редагування.

# Список (Listbox) та лінійка прокрутки (Scrollbar)

---

Віджет список відображає список рядків, з яких можна вибрати один або декілька рядків.

Для створення віджета список треба викликати конструктор класу Listbox, наприклад:

**lbs = Listbox(top)**

Необов'язкові ключові параметри width та height задають ширину та висоту списку у символах та рядках.

Для вставки рядка у віджет список використовують метод insert, наприклад:

**lbs.insert(0, item)**

Перший параметр – це індекс у списку, перед яким треба вставити новий елемент item.

# Список (Listbox) та лінійка прокрутки (Scrollbar).2

---

Якщо треба вставити новий елемент у кінець списку, в якості першого параметру слід використовувати END.

Для видалення елементів списку віджет має метод delete.

Наприклад, щоб очистити весь список, треба викликати

**lbs.delete(0, END)**

# Список (Listbox) та лінійка прокрутки (Scrollbar).3

---

Щоб отримати елемент списку з заданим індексом `idx`, використовують метод `get`:

**`item = lbs.get(idx)`**

Індекс вибраного елемента можна отримати за допомогою методу `curselection`:

**`idx = lbs.curselection()`**

Для того, щоб обробити подію вибору елемента списку, як правило, використовують зв'язування функції обробки з подвійним натисненням лівої клавіші миші:

**`lbs.bind('<Double-1>', sel_handler)`**

- де `sel_handler` – функція обробки події вибору елемента списку.

# Список (Listbox) та лінійка прокрутки (Scrollbar).4

---

Елементів у списку може бути більше, ніж вміщує вікно списку.

У цьому випадку для перегляду всіх елементів потрібна лінійка прокрутки (Scrollbar).

Створення такої лінійки та її зв'язування зі списком виконується так:

**svert = Scrollbar(top)**

**lbs = Listbox(top, yscrollcommand=svert.set)**

**svert.config(command=lbs.yview)**

- де svert – лінійка прокрутки, lbs – список, top – вікно, у якому розміщуються список та лінійка прокрутки.



# Стандартні вікна повідомлень

---

tkinter містить декілька стандартних вікон з повідомленнями.

Таке вікно відкривається нагорі та програма чекає натиснення однієї з доступних у вікні кнопок.

Які саме кнопки є доступними, залежать від типу вікна.

Для використання стандартних повідомлень треба імпортувати модуль `tkinter.messagebox`:

**`from tkinter.messagebox import *`**

Цей модуль містить функції, що відкривають вікна стандартних повідомлень:

- `showinfo` – показати інформаційне повідомлення
- `showwarning` – показати попередження
- `showerror` – показати повідомлення про помилку
- `askyesno` – запитати та отримати відповідь: так чи ні
- `askokcancel` - запитати та отримати відповідь: ок чи відмінити

# Стандартні вікна повідомлень.2

---

Усі функції вимагають 2 параметри: заголовок вікна та текст повідомлення.

Якщо у вікні повідомлення більше однієї доступної кнопки, можна проаналізувати результат функції, щоб визначити, яка саме кнопка була натиснута.

Так, `askyesno` та `askokcancel` повертають бульовий результат (`True`, якщо натиснуто «Так» або «Ok»).

# Діалоги

---

Стандартні повідомлення є прикладом діалогів.

**Діалогом** у графічному інтерфейсі називають відмінне від головного незалежне вікно верхнього рівня.

Якщо від відкриття до закриття такого вікна інші вікна програми не є доступними, такий діалог називають **модальним**, інакше – **немодальним**.

Щоб створити діалогове вікно, у tkinter використовують клас `TopLevel`:

**`dialog = Toplevel()`**

Закрити вікно діалогу можна методом `destroy`:

**`dialog.destroy()`**

# Діалоги.2

---

Робота з немодальними діалогами більше нічим не відрізняється від роботи з головним вікном.

Що ж стосується модальних діалогів, то після створення діалогового вікна та його віджетів треба викликати ще декілька методів TopLevel:

**# перевести фокус у область вікна**

**dialog.focus\_set()**

**# перехопити всі події графічного інтерфейсу**

**dialog.grab\_set()**

**# очікувати знищення діалогового вікна**

**dialog.wait\_window()**

# Клас вибору елементів списку у інший список DoubleList

---

Клас вибору елементів списку у інший список DoubleList призначено для перегляду списку елементів та вибору декількох елементів у другий список.

Клас має такі поля:

- `self.top` - вікно верхнього рівня у якому розміщено елементи
- `self._list_sel` - список вибраних елементів
- `self._list_all` - список усіх елементів
- `self._cancel` - чи було натиснуто "Відмінити"
- `self._l_all` - віджет список для всіх елементів
- `self._l_sel` - віджет список для вибраних

# Клас вибору елементів списку у інший список DoubleList.2

---

Клас має конструктор `__init__`, який викликає внутрішній метод `_make_widgets` для створення віджетів.

Внутрішній метод `_fill_list` заповнює список елементами

Методи `_right_handler` та `_left_handler` обробляють вибір елемента у першому або другому списках (подвійне натиснення клавіші миші) або натиснення на кнопки переміщення елементів

Під час зображення списку можливо, що клавішу буде натиснуто, коли у списку ще немає елементів.

У цьому випадку виникне виключення `TclError`, яке ми пропускаємо.

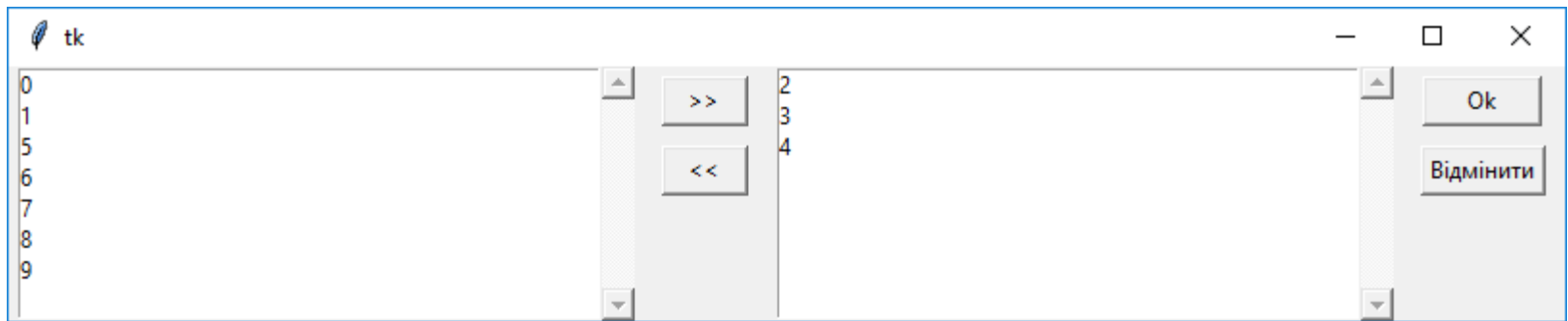
Про інші можливі помилки видається стандартне повідомлення.

Методи `ok_handler` та `cancel_handler` обробляють натиснення кнопок «Ok» та «Відмінити» відповідно

# Клас вибору елементів списку у інший список DoubleList.3

Метод `get` повертає список вибраних елементів або `None`.

Цей метод, як правило, викликається після завершення вибору.



# Клас ReceiptsGUI

---

Клас ReceiptsGUI реалізує інтерфейс головного вікна створення/модифікації рецептів.

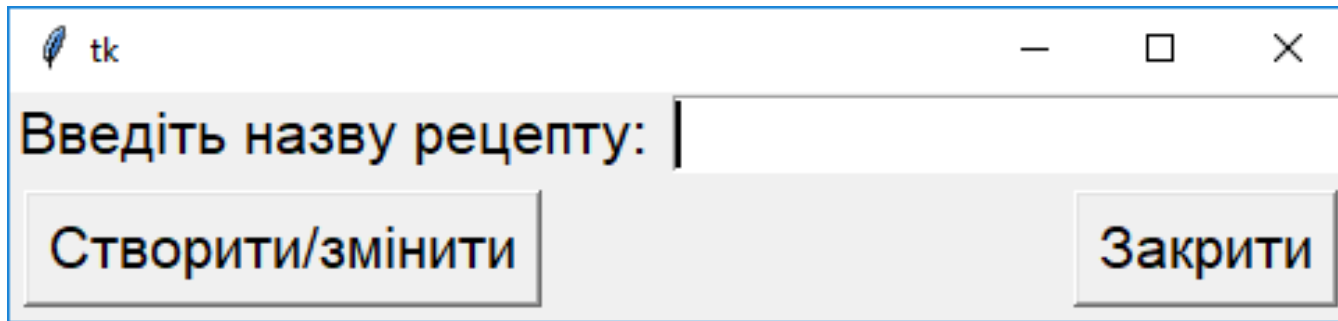
Робота цього класу багато в чому повторює попередні приклади для чисел Фібоначчі.

Різниця полягає у реалізації методу `_create_modify_receipt`



# Клас ReceiptsGUI.2

---



# Вікно тексту (Text)

---

Вікно тексту дозволяє показувати та редагувати текст, що складається з багатьох рядків.

Для створення віджета необхідно використати конструктор

**txt = Text(top)**

Необов'язковий ключовий параметр `wrap` вказує, чи буде текст переноситись у вікні тексту по рядках.

Вміст віджета представляється як рядок Python, який розбитий на окремі рядки у вікні символами `'\n'` по аналогії з вмістом текстових файлів.

# Вікно тексту (Text).2

---

Користувач за угодою може змінювати текст, набираючи оновлення з клавіатури.

Методи роботи з текстом містять параметри, що визначають положення у тексті або початок та кінець частини тексту.

Таке положення можна задавати одним з трьох способів:

- індексами рядка та стовпчика
- мітками (marks)
- ярликами (tags)

# Вікно тексту (Text).3

---

Кожний індекс – це рядок 'm.n', де m – номер рядка у вікні (починається з 1), n – номер стовпчика у рядку (починається з 0).

Таким чином, індекс '1.0' задає початкову позицію у тексті.

Спеціальний індекс END позначає кінець тексту.

Мітки позначають місця у тексті.

На відміну від індексів, мітка змінює абсолютну позицію, якщо текст змінюється.

Але мітка залишається між тими символами, між якими вона була встановлена.

# Вікно тексту (Text).4

---

Назва мітки – це рядок. Є спеціальна мітка INSERT, що позначає місце, у яке будуть вставлятись символи при наборі з клавіатури.

Ярлики позначають частину тексту. Назва ярлика - це також рядок.

Ярлик задається початковою та кінцевою позицією.

Є спеціальний ярлик SEL, що позначає вибрану частину тексту. SEL\_FIRST та SEL\_LAST – це індекси початку та кінця вибраної частини

# Вікно тексту (Text).5

---

Для вставки у текст використовують метод insert. Наприклад,

**txt.insert('1.0', s)**

вставляє рядок s у початок тексту.

Видалення частини тексту здійснюють методом delete, наприклад:

**txt.delete('1.0', END)**

Отримати частину тексту можна методом get. Наприклад отримати перших 10 символів:

**txt.get('1.0','1.9')**

# Вікно тексту (Text).6

---

Встановити мітку у тексті

**txt.mark\_set('mymark', '2.3')**

Забрати раніше встановлену мітку

**txt.mark\_unset('mymark')**

Встановити ярлик

**txt.tag\_add('mytag', '2.0', 'mymark')**

Забрати раніше встановлений ярлик з усього тексту

**txt.tag\_remove('mytag', '1.0', END)**

# Вікно тексту (Text).7

---

Отримати текст, виділений ярликом

```
s = txt.get('mytag.first', 'mytag.last')
```

Частина тексту, виділена ярликом, може мати власний шрифт, колір тексту та колір фону, наприклад:

```
txt.tag_config('mytag', background='navy',  
                foreground='gray',  
                font=('arial', 14, 'bold'))
```



# Вікно тексту (Text).8

---

Вікно тексту, як і список, треба поєднати з лініями прокрутки.

При цьому, якщо текст не переноситься (параметр `wrap='none'`), потрібна як вертикальна, так і горизонтальна лінійки прокрутки.

Поєднання здійснюється аналогічно спискам:

**`svert.config(command=tst.yview)`**

**`shor.config(command=tst.xview)`**

**`tst.config(yscrollcommand=svert.set,  
xscrollcommand=shor.set)`**

- де `svert`, `shor` – вертикальна та горизонтальна лінійки прокрутки.

# Меню (Menu)

---

Віджет меню дозволяє будувати меню, що випадають.

Меню, що випадає (pull down), розміщується у верхній частині вікна.

Кожний пункт меню розкриває список підпунктів.

Кожний підпункт є командою або містить власний список підпунктів тощо.

Щоб створити список пунктів меню, використовують конструктор

**menubar = Menu(top)**

Щоб додати список підпунктів, знову використовують конструктор, а потім додають пункти за допомогою методу `add_command`.

Ключовий параметр `command` – це функція обробки, що викликається при натисненні на пункт меню.

Весь список меню додається методом `add_cascade`

# Меню (Menu).2

---

Наприклад, створити просте меню, що містить 1 пункт «Файл» з підпунктами «Відкрити» та «Вихід» можна так:

```
menubar = Menu(top)

# створити меню, що випадає, та додати до головного меню
# tearoff=0 означає, що меню не може бути "відірване"
# та переміщуватись у окремому вікні

filemenu = Menu(menubar, tearoff=0)

filemenu.add_command(label="Відкрити", command=openfile)

filemenu.add_separator()

filemenu.add_command(label="Вихід", command=top.quit)

menubar.add_cascade(label="Файл", menu=filemenu)

top.config(menu=menubar)
```

Метод `add_separator` вставляє лінію розділу у список меню, завдання ключового параметра меню у `top.config` розміщує меню, що випадає, у вікні.

Кнопка вибору (Checkbutton), радіокнопка (Radiobutton) та рамка з заголовком (LabelFrame)

---

Кнопка вибору або «прапорець» призначена для встановлення або зняття деякого режиму.

Кнопки вибору є незалежними.

Для створення кнопки вибору треба використати конструктор

```
chk = Checkbutton(top, text='Напівгрубий',  
                    variable=boldvar)
```

## Кнопка вибору (Checkbutton), радіокнопка (Radiobutton) та рамка з заголовком (LabelFrame).2

---

Радіокнопка називається так тому, що колись у магнітолах був ряд кнопок, з яких натиснутою могла бути тільки одна. Натиснення якоїсь кнопки відстрілювало всі інші кнопки.

Радіокнопки у графічному інтерфейсі утворюють групи.

Кожна група дозволяє вибрати одну з декількох альтернатив.

Як правило, група радіокнопок міститься у рамці з заголовком, який роз'яснює зміст цієї групи.

Кожна радіокнопка з групи створюється конструктором

```
rb = Radiobutton(fsize, text='10',  
variable=sizevar, value=10)
```

Кнопка вибору (Checkbutton), радіокнопка (Radiobutton) та рамка з заголовком (LabelFrame).3

---

Ключовий параметр `variable` – це змінна `tkinter`, пов'язана з усіма радіокнопками групи.

Параметр `value` вказує значення, яке буде присвоєно цій змінній, якщо буде вибрано дану радіокнопку (і навпаки, що цю кнопку треба позначити, як вибрану, якщо змінна набуде значення `value`).

Кнопка вибору (Checkbutton), радіокнопка (Radiobutton) та рамка з заголовком (LabelFrame).3

---

Рамка з заголовком практично не відрізняється від звичайної рамки.

Тільки при створенні вказують рядок, який буде заголовком:

**fsize = LabelFrame(top, text='Розмір шрифту')**

# Стандартні діалоги

---

Раніше ми вже розглядали стандартні повідомлення у tkinter.

Але стандартні діалоги не обмежуються повідомленнями.

tkinter містить декілька файлових діалогів та діалог вибору кольору.

Щоб використовувати ці діалоги, треба імпортувати відповідні функції з модулів

```
from tkinter.filedialog import askopenfilename
```

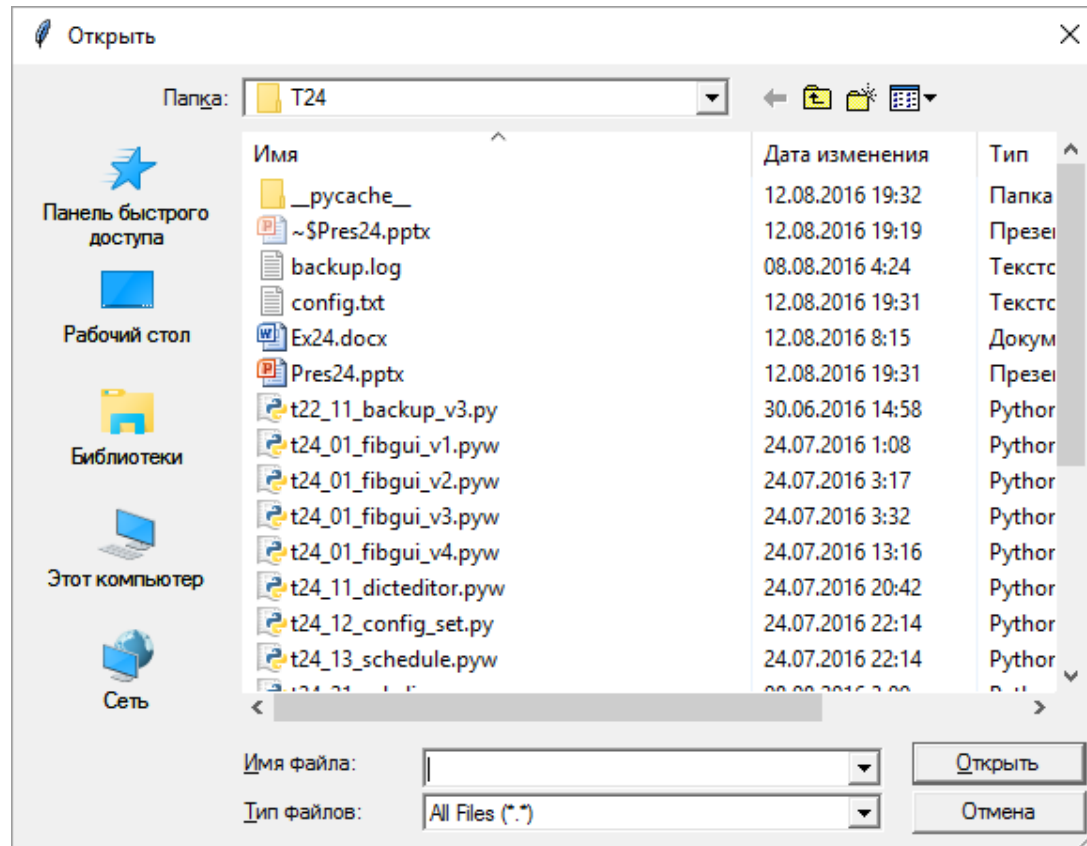
```
from tkinter.colorchooser import askcolor
```

Функція askopenfilename запускає стандартний діалог відкриття файлу та повертає ім'я файлу, якщо файл вибрано, або порожній рядок, якщо файл не вибрано.

```
filename = askopenfilename()
```



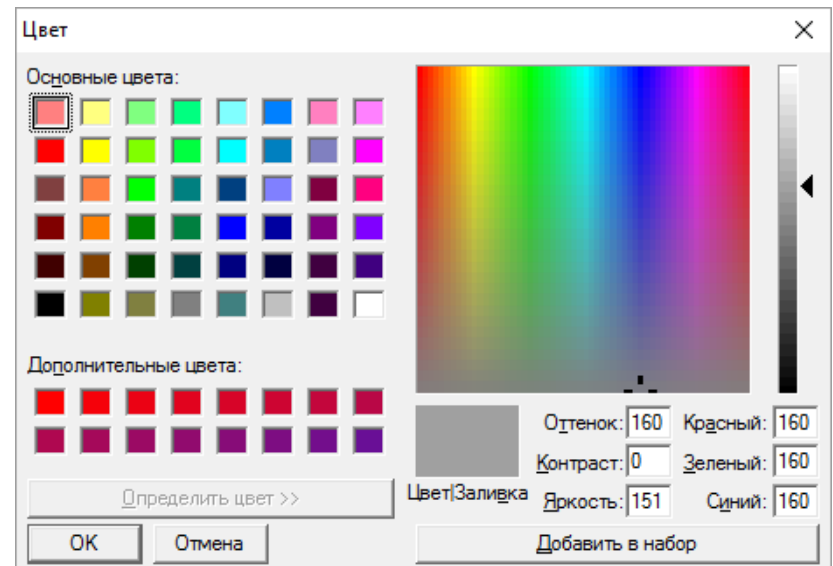
# Стандартні діалоги.2



# Стандартні діалоги.3

Функція `askcolor` запускає стандартний діалог вибору кольору та повертає вибраний колір у двох форматах: кортеж (R, G, B) та рядок у форматі за основою 16. Якщо колір не вибрано, то повертає `None` двічі.

**`triple, hexstr = askcolor()`**



# Приклад: Перегляд текстових файлів

---

Розробити програму для перегляду текстових файлів.

Надати можливість вибирати розмір та написання шрифту а також колір тексту та колір фону.

Для розв'язання задачі опишемо клас `TextViewer`, який надає інтерфейс головного вікна, та діалог з вибору розміру та написання шрифту – клас `FontOpts`.

# Клас TextViewer

---

Клас TextViewer створює графічний інтерфейс для перегляду текстових файлів.

Цей інтерфейс включає меню з введенням файлу (меню Файл) та вибором розмірів та написання шрифту а також кольорів тексту та фону (меню Опції), вікно тексту, у яке виводиться текст файлу.

Клас має поля:

- `self.top` - вікно верхнього рівня у якому розміщено елементи
- `self.filename` - ім'я файлу, що переглядається
- `self.content` - вміст файлу, що переглядається
- `self.text` – вікно тексту
- `self.menubar` – меню

# Клас TextViewer.2

```
D:/obv/Lect_Python2/T24/config.txt
Файл  Опції
BackupDirectory = "D:\Temp\Backup"
Interval = 0.01
Directories = "C:/obv/Lect_Python2/T22 C:\obv\Lect_Python2\T21 C:\obv\Lect_Pytho"
```

# Клас TextViewer.3

---

Конструктор `__init__` викликає внутрішній метод `_make_widgets` для створення елементів інтерфейсу а також внутрішній метод `_fileopen`, який відкриває та читає файл з ім'ям `self.filename`.

Метод `_make_widgets` містить зв'язування подій натискання на клавішу, щоб унеможливити зміну файлу у вікні тексту:

**`self.text.bind('<Key>', lambda e: "break")`**

Тобто, у відповідь на будь-яку клавішу повертається рядок «break», який перериває обробку події від клавіатури у tk.

`_make_widgets` також викликає внутрішній метод `_settext` для вставки тексту з `self.content` у вікно тексту.

# Клас TextViewer.4

---

Методи fgcolor та bgcolor обробляють вибір пунктів меню встановлення кольорів.

Ці методи викликають внутрішній метод \_setcolor, який ініціює стандартний діалог вибору кольору та встановлює у вікні тексту вибраний колір.

Метод setfont обробляє вибір пункту меню встановлення написання шрифту.

Для цього він запускає відповідний діалог (створює об'єкт класу FontOpts)

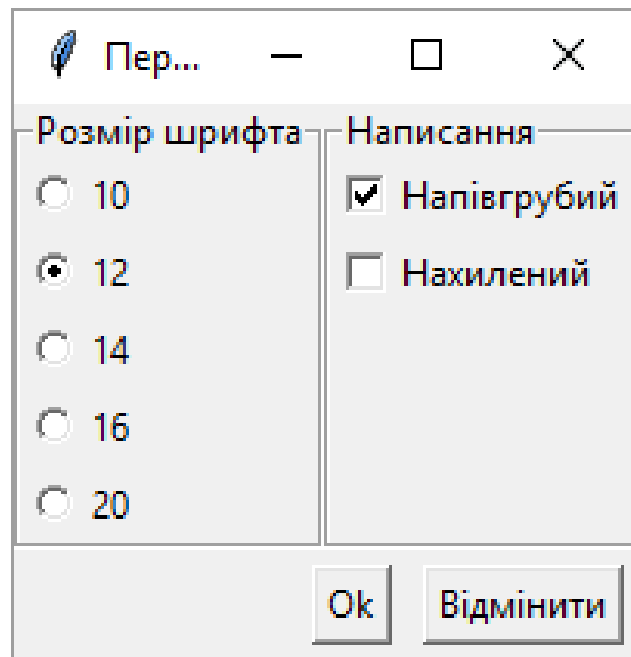
Метод openfile обробляє вибір пункту меню «Відкрити...».

Цей метод ініціює стандартний діалог вибору імені файлу, відкриває файл за допомогою методу \_fileopen та змінює текст у вікні тексту за допомогою \_settext.

# Клас FontOpts

Клас FontOpts призначено для вибору розміру шрифту та написання шрифту.

Графічний інтерфейс містить кнопки вибору та радіокнопки.





# Клас FontOpts.2

---

Клас має поля:

- `self.top` - вікно верхнього рівня у якому розміщено елементи
- `self.cancel` - чи було натиснуто кнопку "Відмінити"
- `self.sizevar` - змінна, пов'язана з радіокнопками
- `self.boldvar` - змінна, пов'язана з 'Напівгрубий'
- `self.italicvar` - змінна, пов'язана з 'Нахилений'

Конструктор `__init__` викликає внутрішній метод `_make_widgets` для створення елементів інтерфейсу.

Методи `ok_handler` та `cancel_handler` обробляють натиснення кнопок «Ok» та «Відмінити» відповідно.

Метод `get` повертає два результати: вибраний розмір шрифту (ціле число) та вибране написання (рядок).

Якщо натиснуто кнопку «Відмінити», то повертає `(None, None)`.

Цей метод, як правило, викликається після завершення вибору.

# Полотно (Canvas)

Полотно (Canvas) призначене для зображення ліній, фігур, тексту, фотографій та вкладених віджетів. Для створення віджета Canvas треба застосувати конструктор

```
canv = Canvas(top, width=200, height=150)
```

- де width та height – ширина та висота віджета у пікселях.

Зображення ліній, фігур, тексту здійснюється методами:

```
id = canv.create_line(fromX, fromY, toX, toY)      # лінія  
id = canv.create_oval(fromX, fromY, toX, toY)      # овал  
id = canv.create_arc( fromX, fromY, toX, toY, start=0, extent=45,  
style=chord)    # дуга  
id = canv.create_rectangle(fromX, fromY,  
                           toX, toY)              # прямокутник  
id = canv.create_image(x1, y1, image=photo) # зображення  
id = canv.create_text(x1, y1, text='Текст')      # текст  
id = canv.create_window(x1, y1, window=widget)   # віджет
```

# Полотно (Canvas).2

---

`fromX`, `fromY`, `toX`, `toY` – координати початку та завершення лінії або фігури.

`x1`, `y1` – координати розміщення зображення, тексту або вкладеного віджету.

За угодою `x1`, `y1` – координати центру.

Щоб зробити їх координатами лівого верхнього кута, треба використати параметр `anchor=NW`.

# Полотно (Canvas).3

---

Усе, що зображується на полотні, є об'єктом.

При створення tkinter повертає номер об'єкту – `id`, за яким можна звертатись до цього об'єкту для зміни його характеристик або видалення.

Інший спосіб звернення до окремого об'єкту на полотні – це ярлик.

Ярлик можна присвоїти при створенні об'єкту, вказавши параметр `tags=e`, де `e` – це один рядок або кортеж з рядків.

Один ярлик може бути присвоєний декільком об'єктам, після чого стає можливим виконувати одну операцію для всіх об'єктів, що мають однаковий ярлик.

# Полотно (Canvas).4

---

Для видалення об'єкту застосовують метод

## **canv.delete(id\_tag)**

- де id\_tag – номер або ярлик.

Для зміни характеристик об'єкту застосовують метод itemconfigure (або itemconfig)

## **canv.itemconfigure(id\_tag, \*\*options)**

- де options – ключові параметри, що встановлюють характеристики об'єкта. Наприклад, для фігур це може бути колір заповнення (fill) та колір границі (outline).

Для переміщення об'єкту застосовують метод

## **canv.move(id\_tag, dx, dy)**

- де dx, dy – відстань по x та y у пікселях, на яку треба перемістити об'єкт на полотні.

# Полотно (Canvas).5

---

Якщо полотно повинно мати розмір більше, ніж вікно на екрані, треба додавати лінійки прокрутки.

Для додавання лінійок прокрутки спочатку потрібно вказати загальний розмір полотна, встановивши ключовий параметр `scrollregion`, наприклад

**`canv.config(scrollregion=(0, 0, 300, 1000))`**

Після цього зв'язування полотна з лініями прокрутки здійснюється таким самим чином, як і для вікна тексту.

# Анімація

---

Анімація у tkinter може бути реалізована декількома способами.

Ми розглянемо один з них – за допомогою методу `after`. Цей метод є у різних віджетів, але нас буде цікавити `canvas`.

Виклик

**`canvas.after(mils, fun, *params)`**

означає, що через `mils` мілісекунд буде викликана функція `fun` і їй будуть передані параметри `params`.

# Анімація.2

---

Таким чином, якщо виклик `after` розмістити у функції `fun`, то кожні `milis` мілісекінд ця функція буде викликатись повторно, та зможе відпрацьовувати ефекти анімації.

Треба зазначити, що виклик `after` не зупиняє програму.

Якщо треба, щоб програма очікувала завершення анімації, слід використовувати один з методів `wait`, наприклад, `wait_variable`:

## **`canv.wait_variable(tk_var)`**

- де `tk_var` – змінна `tkinter`. `tkinter` буде очікувати у місці виклику `wait_variable`, доки не буде встановлене істинне значення `tk_var`.



# Приклад: гра у Lines

---

Треба реалізувати гру у Lines.

Правила гри у Lines (Лінійки, Кульки) полягають у наступному.

Є ігрове поле розміром 9х9 клітинок.

На полі можуть розміщуватись кульки 6 різних кольорів.

Гравець може переміщувати кульку з поточної до іншої позиції, якщо між двома позиціями є шлях.

Шлях складається з сусідніх порожніх клітинок по горизонталі та/або вертикалі.

Якщо гравець збирає 5 або більше сусідніх кульок однакового кольору по горизонталі, вертикалі або діагоналі, ці кульки знімаються з поля, а гравцю нараховують бали.

На кожному кроці комп'ютер розміщує у 3 випадкових порожніх позиціях 3 кульки випадкових кольорів.

# Приклад: гра у Lines.2

---

Якщо 5 або більше кульок знімаються, гравець отримує право на бонусний хід.

Гра закінчується, коли ігрове поле повністю заповнюється кульками.

Задача – набрати якомога більше балів до закінчення гри.

Для розв'язання задачі опишемо клас GridCanvas – клас, що зображує поле розміром  $m \times n$  клітинок та дозволяє розміщувати у клітинці фігуру, зображення або текст.

Також опишемо класи Lines, який містить методи для підтримки гри, та LinesGUI, який буде графічний інтерфейс та веде гру.

# Клас GridCanvas

---

Клас GridCanvas зображує поле розміром  $m \times n$  клітинок та є нащадком Canvas.

Таким чином, GridCanvas має всі поля та методи Canvas.

GridCanvas перевизначає конструктор `__init__`.

GridCanvas містить таблицю `self.grid` розміром  $m \times n$ , кожен елемент якої, - це зв'язаний об'єкт.

Зв'язаним об'єктом може бути фігура, зображення або текст.

# Клас GridCanvas.2

---

Клас має поля:

- `self.rows` - кількість рядків поля
- `self.cols` - кількість стовпчиків поля
- `self.selection_handler` - функція, що буде викликатись при виборі клітинки поля
- `self.bordercolor` - колір границі між клітинками
- `self.evenbg` - колір заповнення клітинок з парними номерами (якщо відрізняється для парних та непарних номерів). Перша клітинка має номер 0
- `self.highlightbg` - колір заповнення вибраної клітинки
- `self.ratio` - відсоток заповнення площі клітинки зв'язаним об'єктом
- `self.cellwidth` - ширина клітинки
- `self.cellheight` - висота клітинки
- `self.grid` - матриця, що складається зі зв'язаних об'єктів для всіх клітинок. Якщо до клітинки не прив'язано об'єкт, то значення відповідного елемента - `None`.
- `self.moved` - змінна `tkinter` для контролю завершення переміщення об'єкту

# Клас GridCanvas.3

---

Конструктор `__init__` викликає конструктор батьківського класу, задає початкові значення полів, будує таблицю зв'язаних об'єктів та викликає внутрішній метод `_drawgrid`.

Метод `_drawgrid` зображує поле як сукупність прямокутників.

Кожен прямокутник отримує свій ярлик, який повертає внутрішній метод `_tagstr`

Метод `create_bound` створює та зображує зв'язаний об'єкт класу нащадка `BoundObject`.

Метод `delete_bound` видаляє зв'язаний об'єкт.

Метод `move_bound` переміщує зв'язаний об'єкт з однієї клітинки до іншої.

Якщо параметр `slow=True`, то об'єкт переміщується повільно з використанням анімації.

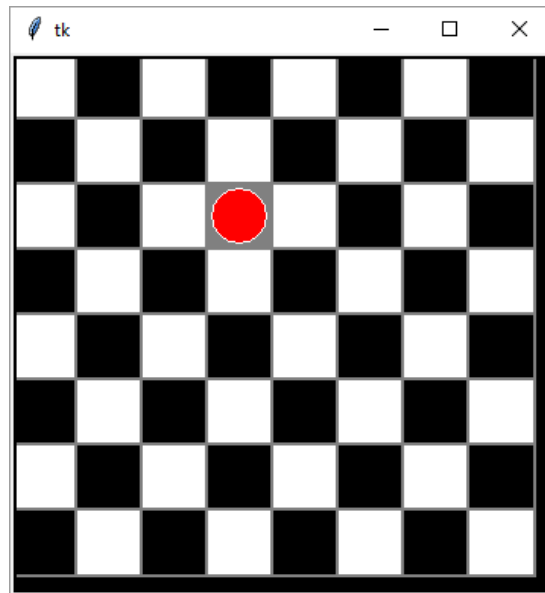
Для повільного переміщення використовується внутрішній метод `_movestep`.

# Клас GridCanvas.4

---

Методи `select_cell` та `deselect_cell` підсвічують кольором `highlightbg` вибрану клітинку або знімають підсвітку раніше вибраної клітинки.

Метод `on_click` обробляє натиснення лівої клавіші миші та викликає функцію `self.selection_handler`.



# Класи BoundObj та BoundOval

---

Клас BoundObj – це абстрактний клас зв'язаного з полем таблиці об'єкту.

Він має конструктор та абстрактні методи move – перемістити об'єкт, delete – видалити об'єкт, coords – повернути поточні координати об'єкту на полотні. Внутрішній метод \_get\_rect повертає прямокутник, у якому розміщується зв'язаний об'єкт

Клас BoundOval – це клас відображення овалу - нащадок BoundObj.

Він також має конструктор та методи move, delete, coords.

# Клас Lines

---

Клас Lines містить функціональність, яка підтримує гру у lines.

Зокрема, клас має поля:

- `self.cl` - кольори нових кульок
- `self.tries` - список можливих переходів у сусідні клітинки
- `self.empty_list` - список координат порожніх клітинок поля

Клас містить конструктор, що встановлює початкові значення полів, а також методи `clear`, `get_spheres` та `get_path`.

Метод `clear` знаходить 5 або більше кульок однакового кольору у горизонталях, вертикалях та діагоналях та повертає їх координати у списку списків.

Метод також обчислює, скільки треба додати до рахунку гравця.

Для пошуку кульок, які можна почистити, будується список, що складається зі списків усіх горизонталей, вертикалей та діагоналей.

Потім у кожному списку шукаємо 5 або більше підряд однакових кульок.

Якщо знайшли, додаємо до результату



# Клас Lines.2

---

Метод `get_spheres` серед порожніх клітинок знаходить та повертає місця для нових кульок.

Також отримує кольори нових кульок.

Використовує внутрішній метод `_set_empty` для побудови списку координат усіх порожніх клітинок поля.

Метод `get_path` перевіряє, чи є шлях між двома клітинками.

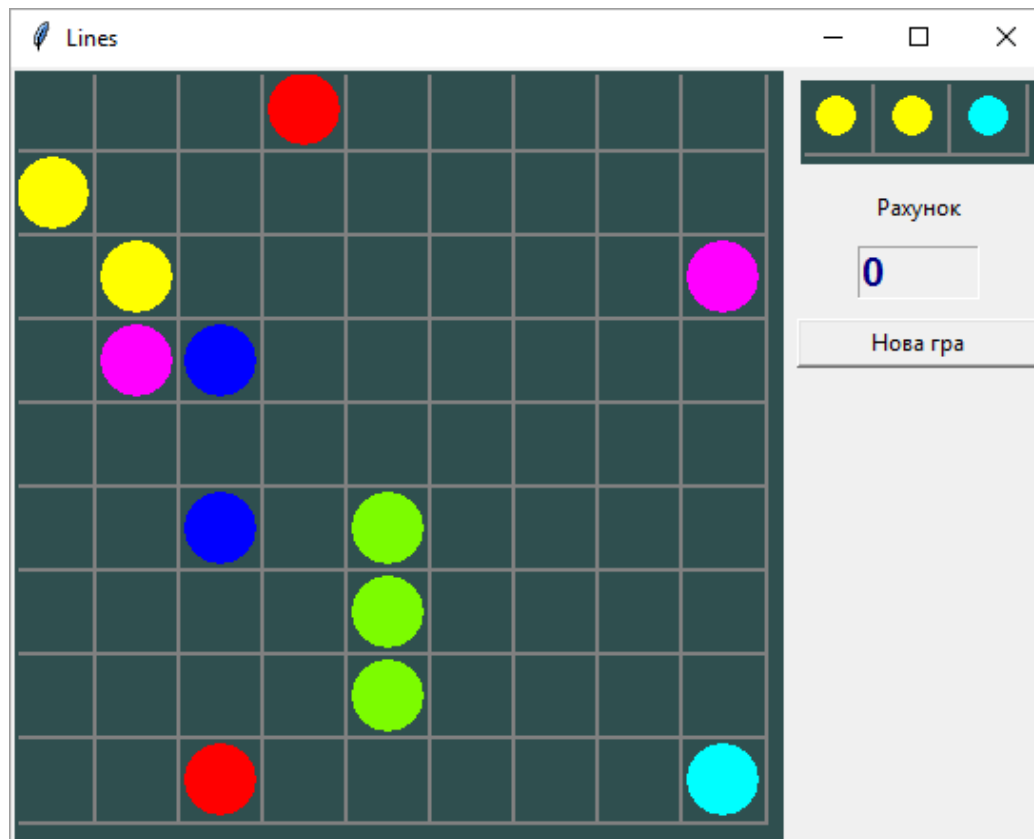
Якщо є, то повертає цей шлях.

`get_path` використовує внутрішній метод `_path_recursive`, який і виконує усю роботу.

Алгоритм `_path_recursive` аналогічний алгоритму пошуку туру коня з теми «Множини».

# Клас LinesGUI

Клас LinesGUI будує графічний інтерфейс Lines та веде гру.



# Клас LinesGUI.2

---

Клас має поля:

- `self.top` - вікно верхнього рівня у якому розміщено елементи
- `self.rows` - кількість рядків
- `self.cols` - кількість стовпчиків
- `self.empty` - кількість порожніх клітинок на полі
- `self.lines` - об'єкт класу `Lines` - містить методи, що підтримують гру
- `self.state` - стан: хід комп'ютера, хід користувача або вибрано клітинку для переміщення
- `self.selrow` - рядок вибраної клітинки
- `self.selcol` - стовпчик вибраної клітинки
- `self.gc` - об'єкт класу `GridCanvas` - поле з клітинками
- `self.little_gc` - поле з 3 клітинок для показу наступних кольорів
- `self.score` - ціла змінна `tkinter` для збереження та відображення рахунку

# Клас LinesGUI.3

---

Конструктор `__init__` встановлює початкові значення полів, викликає внутрішній метод `_make_widgets` для побудови елементів інтерфейсу а також внутрішній метод `_start_game`, який виконує дії, необхідні для початку нової гри.

Метод `move_path` здійснює повільне переміщення кульки вздовж шляху `path`.

Для переміщення використовує відповідний метод `GridCanvas`.

Метод `clear` видаляє з клітинок поля, що треба очистити, зв'язані об'єкти та змінює рахунок гри.

Список клітинок, які треба очистити, повертає метод `clear` з класу `Lines`.

Метод `show_next_colors` показує на маленькому полі з 3 клітинок кольори кульок, що будуть розміщені на наступному кроці.

# Клас LinesGUI.4

---

Основну роботу з ведення гри та підтримки ігрової логіки виконують методи `computer_move` та `sel_handler`.

Гра може знаходитись в одному з 3 станів:

- «хід гравця»

- «гравець вибрав клітинку»

- «хід комп'ютера».

Перехід між станами здійснюють ці два методи.

Метод `computer_move` виконує хід комп'ютера: розміщує нові кульки на полі та перевіряє, чи не закінчено гру (чи є ще порожні клітинки).

# Клас LinesGUI.5

---

Метод `sel_handler` викликається з класу `GridCanvas` для обробки події вибору гравцем клітинки.

Якщо поточний стан - «гравець вибрав клітинку», то , якщо клітинка порожня, це означає, що зараз вибрано клітинку, у яку треба перемістити кульку.

- Тому шукаємо шлях та, якщо знаходимо, то переміщуємо кульку та пробуємо очистити.
- Якщо вдалося очистити кульки, то хід залишається у гравця, інакше переходить до комп'ютера.
- Якщо у стані «гравець вибрав клітинку» нова вибрана клітинка не порожня, треба змінити вибрану клітинку на поточну без зміни стану.

Якщо ж поточний стан - «хід гравця» та клітинка не порожня, то вибираємо її та змінюємо стан на «гравець вибрав клітинку».

Метод `newgame_handler` обробляє натиснення кнопки «Нова гра». Він очищує поле, вибір клітинки та викликає внутрішній метод `_start_game`.

# Резюме

---

Ми розглянули:

1. Графічний інтерфейс
2. Програмування, що керується подіями
3. Графічні бібліотеки у Python
4. Початок роботи з tkinter. Основні поняття.
5. Основні віджети
6. Кроки виконання програми, яка використовує tkinter
7. Ієрархія вкладень віджетів
8. Надпис (Label), кнопка команд (Button) та поле введення (Entry)
9. Встановлення відображення графічних елементів
10. Модифікація параметрів графічних елементів

# Резюме.2

---

11. Менеджери розміщення. Менеджер розміщення pack
12. Рамка (Frame). Створення та пакування елементів однією командою
13. Прив'язка подій до функцій обробки
14. Графічний інтерфейс та об'єктно-орієнтоване програмування
15. Менеджер розміщення grid
16. Змінні tkinter
17. Список (Listbox) та лінійка прокрутки (Scrollbar)
18. Стандартні вікна повідомлень. Діалоги
19. Вікно тексту (Text). Меню (Menu)
20. Кнопка вибору (Checkbutton), радіокнопка (Radiobutton) та рамка з заголовком (LabelFrame)
21. Стандартні діалоги
22. Полотно (Canvas). Анімація



# Де прочитати

---

1. Обвінцев О.В. Об'єктно-орієнтоване програмування. Курс на основі Python. Матеріали лекцій. – К., Основа, 2017
2. Wesley J. Chun - Core Python Programming - 2001
3. Magnus Lie Hetland - Beginning Python from Novice to Professional, 2nd ed – 2008
4. Harwani B. M. - Introduction to Python Programming and Developing GUI Applications with PyQt – 2012
5. Mark Lutz - Programming Python. 4<sup>th</sup> Edition - 2011
6. Прохоренко Н.А. - Python 3 и PyQt. Разработка приложений – 2012
7. Марк Саммерфилд, Программирование на Python 3. Подробное руководство. - Символ-Плюс, 2009.
8. Марк Саммерфилд - Python на практике. ДМК - 2014
9. Paul Gries and Others - Practical Programming - An Introduction to Computer Science Using The Python 3 - 2nd Edition – 2013
10. Kent D. Lee - Python Programming Fundamentals (2nd edition) (Undergraduate Topics in Computer Science) – 2014
11. [http://www.python-course.eu/python\\_tkinter.php](http://www.python-course.eu/python_tkinter.php)