

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Л. О. Краснов, К. Ю. Дергачов, С. В. Багінський

**ОСНОВИ ПОБУДОВИ СУЧАСНИХ МОБІЛЬНИХ СИСТЕМ
ТЕХНІЧНОГО ЗОРУ
Частина 1
КОМП'ЮТЕРИ І ЗАСОБИ ПРОГРАМУВАННЯ**

Навчальний посібник

Харків «ХАІ» 2018

УДК 004.942(075.8)
К78

Рецензенти: канд. техн. наук В. О. Кочура,
канд. техн. наук С. М. Флерко

Краснов, Л. О.

К78 Основи побудови сучасних мобільних систем технічного зору [Текст] : навч. посіб. Ч. 1. Комп'ютери і засоби програмування / Л. О. Краснов, К. Ю. Дергачов, С. В. Багінський. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2018. – 104 с.

ISBN 978-966-662-580-2

Подано матеріали з теоретичного й практичного вивчення методів побудови сучасних систем технічного зору. Розглянуто питання застосування мови програмування Python спільно з бібліотекою для оброблення зображень OpenCV на базі мікрокомп'ютерів Raspberry Pi з метою вирішення завдань оброблення даних і керування процесом відеоспостереження. Наведено приклади практичної роботи з реальними зображеннями й даними відеокамер.

Для студентів, що навчаються за напрямками підготовки «Авіоніка», «Аеронавігація» і «Системна інженерія» спеціальностей «Системи управління літальними апаратами і комплексами», «Комп'ютеризовані системи управління і автоматика», «Аеронавігаційні системи і системи аеронавігаційного обслуговування».

Іл. 74. Табл. 6. Бібліогр.: 17 назв

УДК 004.942(075.8)

© Краснов Л. О., Дергачов К. Ю., Багінський С. В., 2018
© Національний аерокосмічний
університет ім. М. Є. Жуковського
«Харківський авіаційний інститут», 2018

ISBN 978-966-662-580-2

ВСТУП

Створення високоефективних систем керування безпілотними літальними апаратами, автомобілями, роботами та іншими видами транспорту супроводжується бурхливим упровадженням інноваційних систем технічного зору. Це дає змогу вирішувати комплекс питань щодо виявлення рухомих об'єктів, оцінювання параметрів їх руху та створення інтелектуальних алгоритмів раціонального керування в області локальної і глобальної навігації транспортних засобів. З'являється можливість удосконалення методів і моделей керування різномірною транспортною інфраструктурою на основі використання технологій штучного інтелекту й технічного зору. Це насамперед стосується керування динамічними транспортними системами як військового, так і подвійного призначення.

Інтерес до цієї проблеми постійно підвищується, а актуальність проведення нових наукових досліджень і створення нових інженерних розробок в області технічного зору неухильно зростає.

Центральним завданням при розробленні й проектуванні мобільних інтелектуальних систем технічного зору є правильне обґрунтування вибору технічних засобів і програмного забезпечення, що дає можливість реалізувати проект на високому науковому й технічному рівні.

Для цього необхідно підібрати мікрокомп'ютер, який може ефективно вирішувати не лише завдання оброблення відеоданих, але й генерувати команди керування рухом транспортної платформи, а також положенням закріпленої на ній відеокамери. Окрім цього мікрокомп'ютер повинен задовольняти й додаткові вимоги: компактність, мале енергоспоживання й незалежне живлення, можливість підключення до пристроїв периферії стандартними засобами, передавання даних за допомогою проводових і безпроводових каналів зв'язку, робота з сучасними операційними системами і підтримкою мов програмування, що забезпечують вирішення поставлених завдань простими й досяжними методами. Найважливішим питанням є також оптимальний вибір мови програмування, яка є функціональною для проведення математичних обчислень, має простий синтаксис, взаємодіє з такими мовами, як C, C++, Matlab, і бібліотеками оброблення зображень openCV. Усі ці й багато інших питань викладено в пропонованій роботі.

Через обмеженість обсягу публікації у ній відображено далеко не всі аспекти проблем, що вивчаються. Сподіваємося, що Вам вдасться самотійно здолати труднощі вивчення й застосувати набуті навички в повсякденній практиці дослідження й проектування систем керування, орієнтованих на використання візуальної інформації та даних відеоспостережень.

1. ВИКОРИСТАННЯ МІКРОКОМП'ЮТЕРІВ У МОБІЛЬНИХ СИСТЕМАХ ТЕХНІЧНОГО ЗОРУ

1.1. Вибір моделі одноплатового мікрокомп'ютера

Сьогодні для розробників проектів, пов'язаних зі створенням мобільних систем технічного зору різного призначення, доступною є велика кількість мікрокомп'ютерів від різних виробників.

Найбільш популярними є моделі, реалізовані на базі платформ **Lego**, **Intel Galileo**, **Android IOIO OTG**, **Arduino** і **Raspberry Pi**. Проведемо порівняльний аналіз їх основних технічних можливостей і характеристик.

Платформа Lego, що є найбільш простою, призначена для створення з базових блоків як простих, так і складних роботів. Програмування здійснюється за допомогою графічної мови в середовищі **LABVIEW**, мов **ROBOTC** (розширення мови **C**) і **Java**. Основна перевага – простота вивчення. Як недоліки слід зазначити досить високу вартість (основний модуль – майже 180 дол., датчики – від 25 до 50 дол.), а також відсутність можливості прямого доступу до програми для створення й підключення власних пристроїв.

Платформа Android IOIO OTG – це плата, що розширює функціонал **android**-смартфона для потреб робототехніки. Її можна підключати як до смартфона, так і до **ПК**. Основні переваги – використання обчислювальних потужностей сучасних смартфонів, наявність стандартної шини **GPIO** (general purpose input/output) з підтримкою протоколів **UART**, **SPI**, **I²C** і звичний для **android**-програмістів спосіб програмування. Недоліки – неможливість автономного використання без з'єднання з комп'ютером або **android**-смартфоном і досить обмежене коло користувачів.

Платформа Arduino – одна з найпоширеніших на цей час платформ для груп користувачів різного рівня підготовки. Пропонується декілька різновидів базових плат, рознімів, що різняться кількістю, тактовою частотою й розрядністю процесора, розміром оперативної пам'яті й параметрами живлення. Програмується як за допомогою графічної мови (**Scratch**), так і за допомогою трохи модифікованої мови **C**.

Основні переваги:

- відносно недорога у використанні платформа (оригінальні базові плати коштують від 12 дол.);
- можливість використання як на початковому, так і на професіональному рівні;
- наявність стандартних шин **GPIO** з підтримкою протоколів **UART**, **SPI**, **I²C**;
- велика кількість сумісних з цією платформою датчиків і плат для потреб робототехніки;
- можливість створення власних плат для розширення цієї платформи.

Основні недоліки полягають у такому:

- мала потужність процесора (максимум 84 МГц);
- нестача пам'яті (максимум 256 Кб);
- відсутність можливості використовувати стандартну комп'ютерну периферію (клавіатуру, мишку, «флешки», веб-камери, зовнішні жорсткі диски); для підключення потребуються плати розширення, які необхідно придбати окремо.

Платформа Intel Galileo. Одна з наймолодших на цей час платформ, але, як видно з назви, просувається одним з ІТ-гігантів в області сучасних процесорів, так що незабаром можна очікувати зростання популярності. Сумісна з платами **Arduino** і може працювати з **Arduino IDE**.

Основні переваги:

- архітектура **x86**;
- досить потужний процесор – 400 МГц;
- 256 Мб оперативної пам'яті;
- можливість установлення **ОС Linux** для роботи на самій платі;
- стандартні шини **GPIO** з підтримкою протоколів **UART, SPI, I²C**;
- наявність портів **USB**-клієнт і **USB**-хост;
- наявність розніму **Ethernet**;
- рознім для карт **MICRO-SD**;
- рознім **MINI-PCI Express**;
- послідовний порт **RS-232**.

До недоліків слід віднести високу вартість (від 70 дол.) і обмежене коло користувачів.

Платформа Raspberry Pi на цей час представлена лінійкою моделей, що мають різну апаратну реалізацію за різною ціною. Нижче наведено порівняльну таблицю технічних характеристик різних моделей (табл. 1.1).

Операційні системи, що застосовуються для **Raspberry Pi: Linux, Android, RISC OS**. Для **Linux** використовуються такі **ОС**:

- **Raspbian** (рекомендована розробником як пріоритетна);
- **Arch Linux ARM**;
- **Moebius**;
- **Raspbmc**;
- **Xbian**.

Програмування **Raspberry Pi** здійснюють на мовах високого рівня (**C, Java, Python, PHP, C#, Ruby, Perl, Scratch, Lazarus / Free Pascal, BASIC**). Програмування можна здійснювати у двох основних режимах – автономно на **Raspberry Pi** і шляхом об'єднання звичайного комп'ютера з **Raspberry Pi**.

Окрім того, доступним є використання готових вузлів (наприклад, від **Arduino**) і створення власних електронних компонент з використанням безпайних макетних плат. **Raspberry Pi** підтримує промислові інтерфейси **SPI, I²C, UART** через порти **GPIO**.

Таблиця 1.1

Варіанти виконання			
Raspberry Pi	Модель А	Модель В	Модель В+
Chip	Broadcom BSM2835 SoC full HD multimedia applications processor	Broadcom BSM2835 SoC full HD multimedia applications processor	Broadcom BSM2835 SoC full HD multimedia applications processor
RAM	256 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz
Storage	SD card	SD Card	MicroSD
USB 2.0	1 x USB Ports	2 x USB Ports	4 x USB Ports
Power Drav/Voltage	600 mA up to 1.2 A @ 5 V	750 mA up to 1.2 A @ 5 V	600 mA up to 1.8 A @ 5 V
GPIO	26	26	40

Таким чином, проаналізувавши зазначені платформи, можна зробити висновок, що під час проектування сучасних систем технічного зору й робототехнічних систем доцільно орієнтуватися на використання платформи **Raspberry Pi**. Це дає можливість не лише забезпечити необхідні технічні характеристики, але й закласти ресурси для розроблення перспективних проектів.

Ураховуючи результати порівняльного аналізу й висновки, зроблені на його основі, надалі розглядатимемо використання мікрокомп'ютера в мобільних системах технічного зору на базі **Raspberry Pi**.

1.2. Загальні відомості про одноплатовий комп'ютер Raspberry Pi

Raspberry Pi – одноплатовий комп'ютер, тобто різні частини комп'ютера, які зазвичай розташовані на окремих платах, тут – на одній. До того ж, ця плата має відносно невеликий розмір – приблизно 8,5х5,5 см.

У назві продукту об'єднано **Raspberry** – малина і **Pi** – число π . Зображення малини стало логотипом проекту (рис. 1.1).

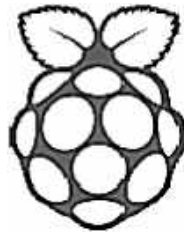


Рис. 1.1. Логотип проекту Raspberry Pi

Raspberry Pi – недорога платформа, рекомендована ціна якої становить усього 35 дол., а для новітньої версії A+ ще менше – 20 дол.

Комерційне постачання цього одноплатового комп'ютера почалося порівняно недавно – на початку 2012 р., а сьогодні це – найбільш популярна платформа у своїй області, продано вже більше 3,5 млн екземплярів **Raspberry Pi**.

Raspberry Pi часто використовується як мозок робота, домашній сервер або просто комп'ютер. Спочатку проект **Raspberry Pi** створювався як освітній для вивчення основ електроніки. На ньому базується безліч комп'ютерних класів. Проте сьогодні його призначення вийшло далеко за межі освітніх проектів.

Конструктивне виконання Raspberry Pi. Зовнішній вигляд плати комп'ютера зображено на рис. 1.2.

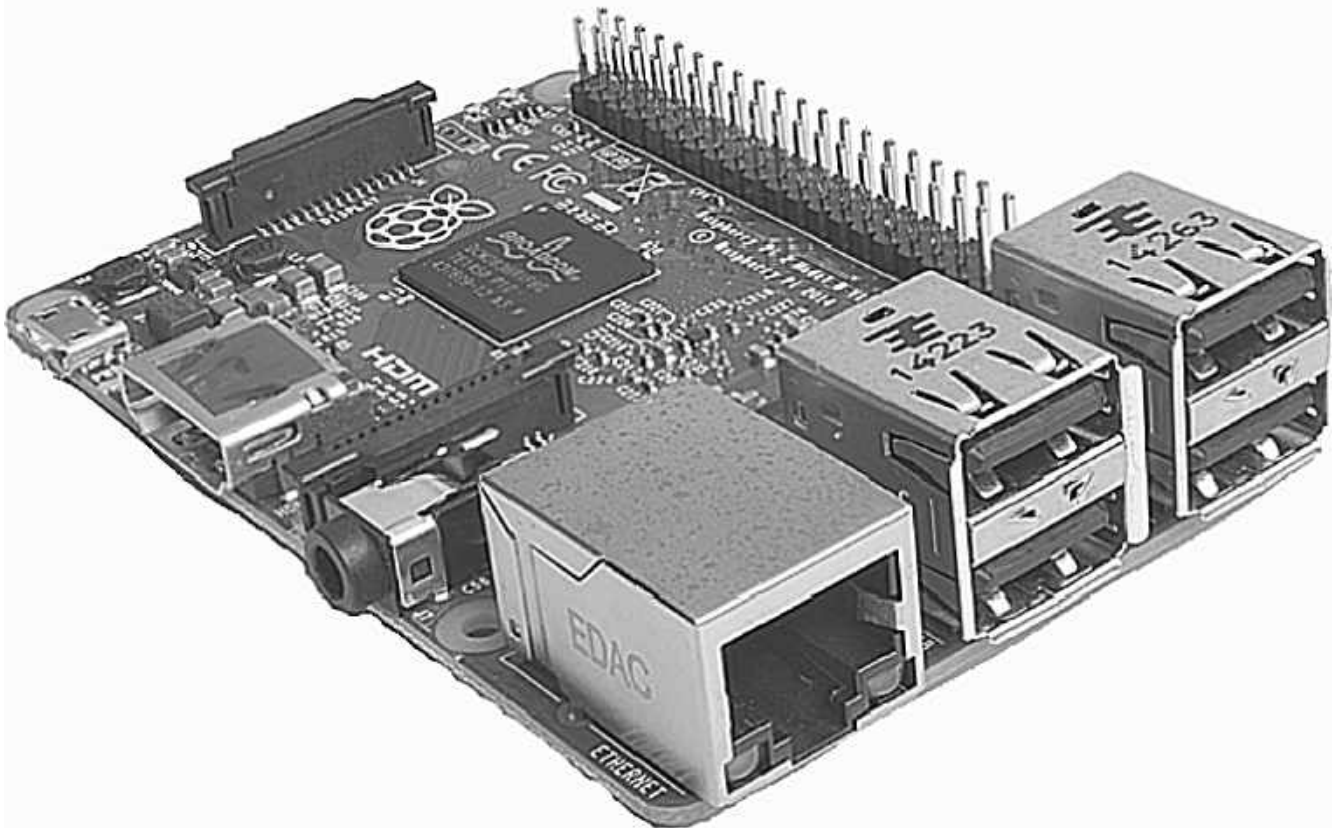


Рис. 1.2. Плата **Raspberry Pi 2** (модель B)

На рис. 1.3 показано фотографію плати **Raspberry Pi**, розміщеної на долоні людини. Це яскраве свідцтво її малогабаритності.

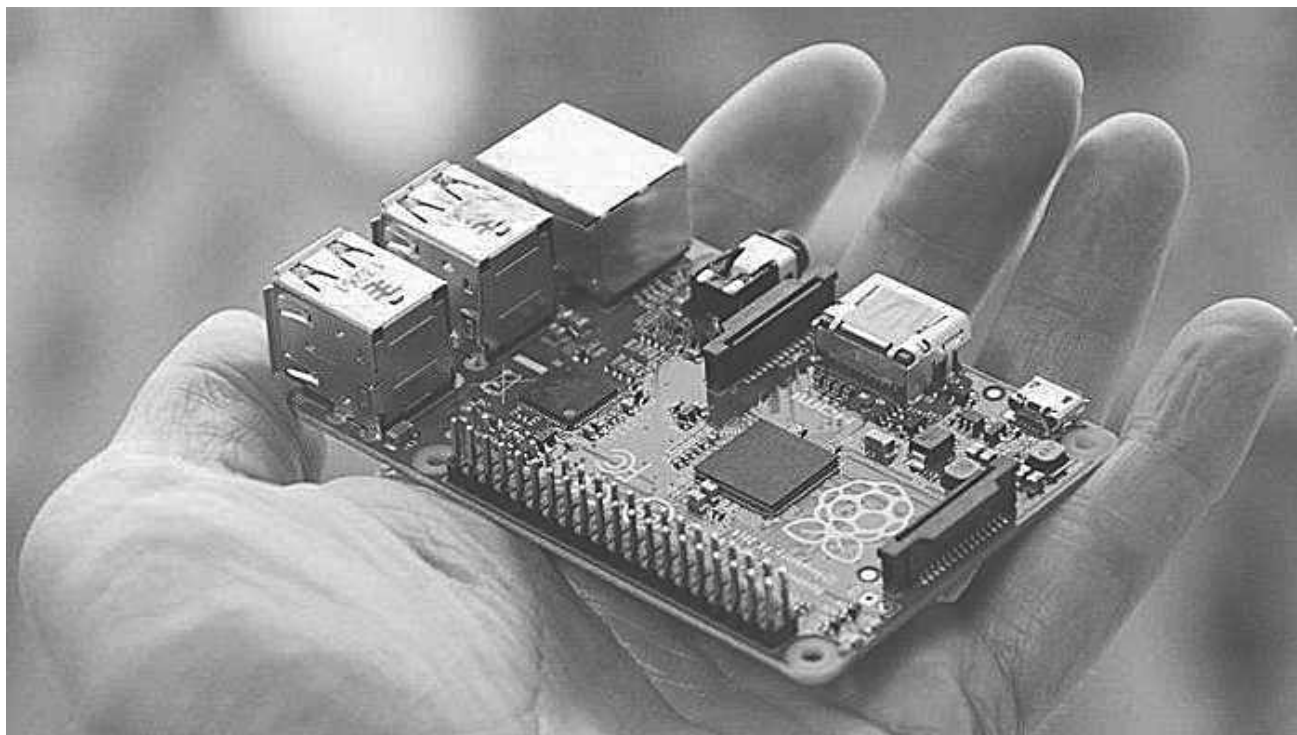


Рис. 1.3. Розміри комп'ютера **Raspberry Pi**

Один із варіантів підключення **Raspberry Pi** до зовнішніх пристроїв зображено на рис. 1.4.

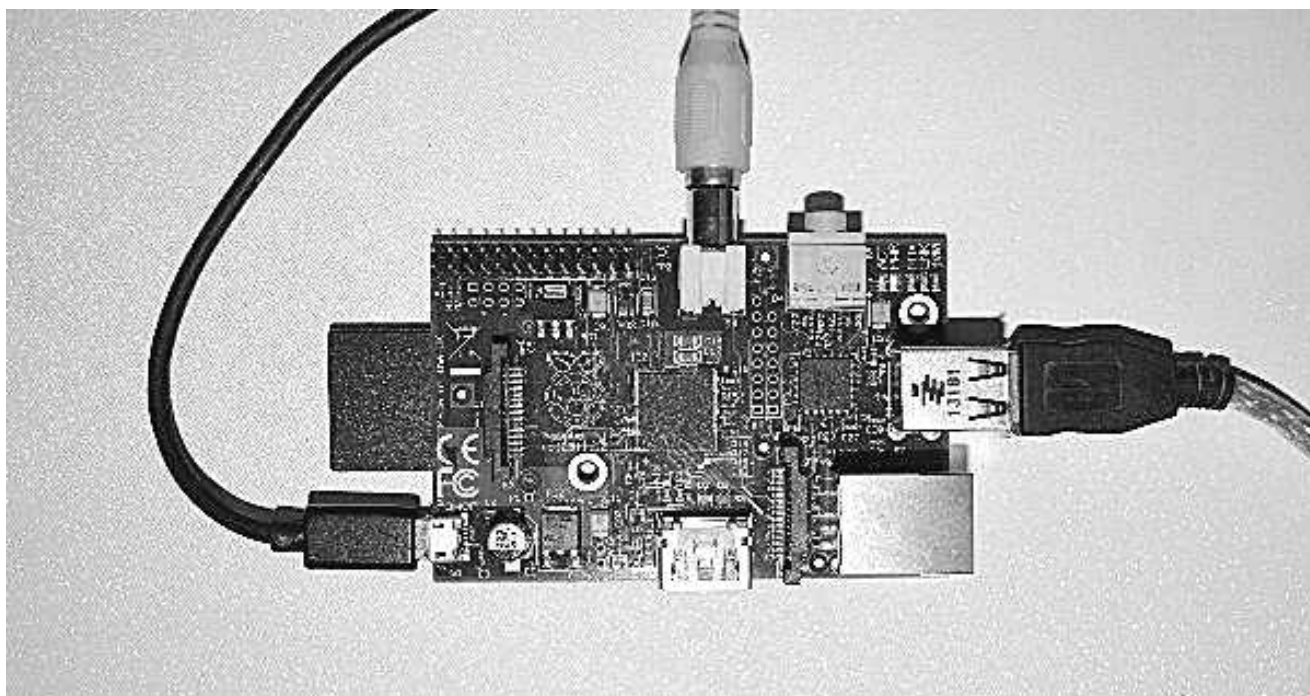


Рис. 1.4. Варіант підключення до зовнішніх пристроїв

На рис. 1.5. показано колісний робот на платформі **Raspberry Pi 2**, оснащений відеокамерою як засобом технічного зору.

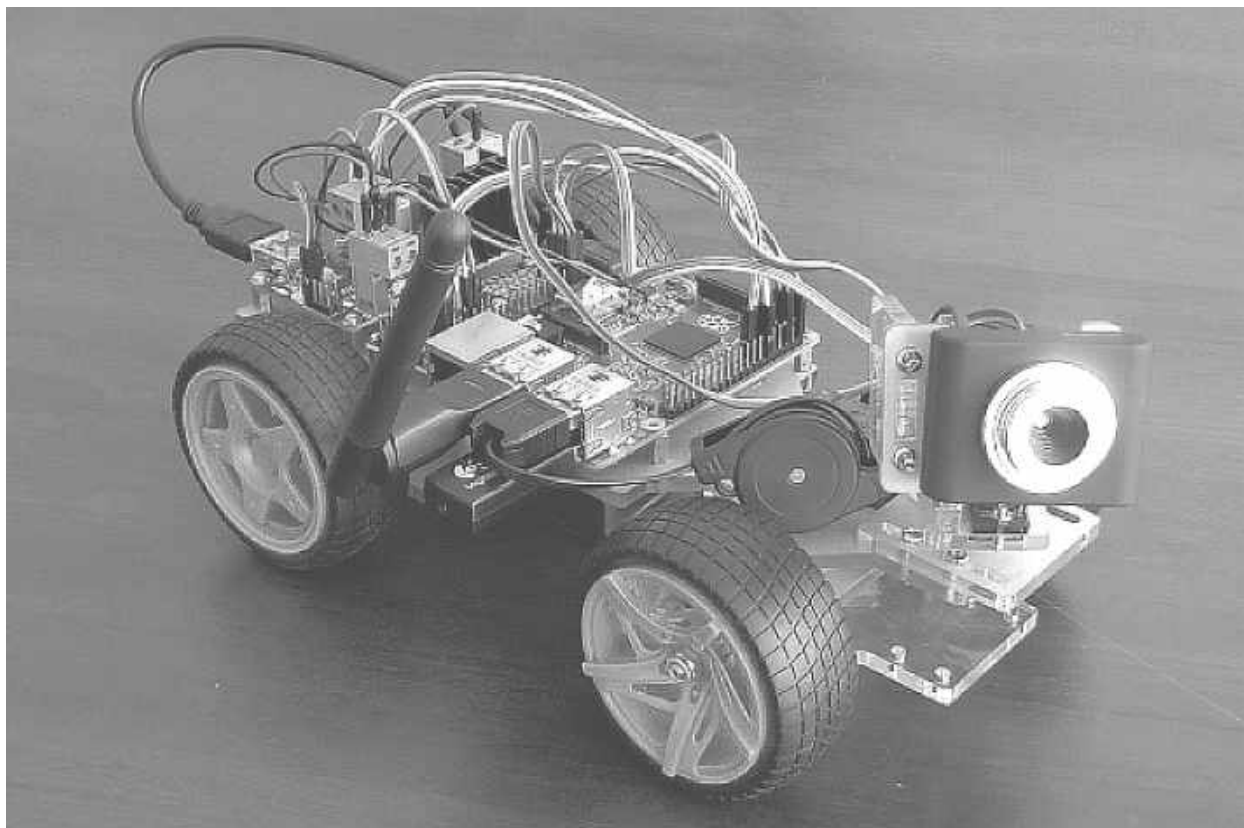


Рис. 1.5. Колісний робот на платформі **Raspberry Pi 2**

Тут і далі користуватимемося технічними характеристиками платформи **Raspberry Pi 2** (модель **B**), наведеними в табл. 1.2.

Таблиця 1.2

Процесор (CPU)	Broadcom BCM2836 ARM Cortex-A7 Quad Core 900 MHz
Кількість ядер процесора	4
RAM	1 Gb
Пам'ять	MicroSD
Мережні можливості	Ethernet 10/100
Відеовивід	HDMI
USB-порти	4
Аудіовивід	3,5 Jack
GPIO	40 pin
Живлення	5 V, 2 A
Додаткові можливості	2 діоди
Габарити, маса	85,6 x 53,98 x 17мм, 45 г

За необхідності можна використовувати додаткові аксесуари. Зокрема, пропонується великий набір металевих і пластикових корпусів для плати **Raspberry Pi 2**, один з яких показано на рис. 1.6.



Рис. 1.6. Пластиковий корпус для **Raspberry Pi 2**

На рис. 1.7 зображено всі розніми для зовнішнього підключення й детально описано їх призначення.

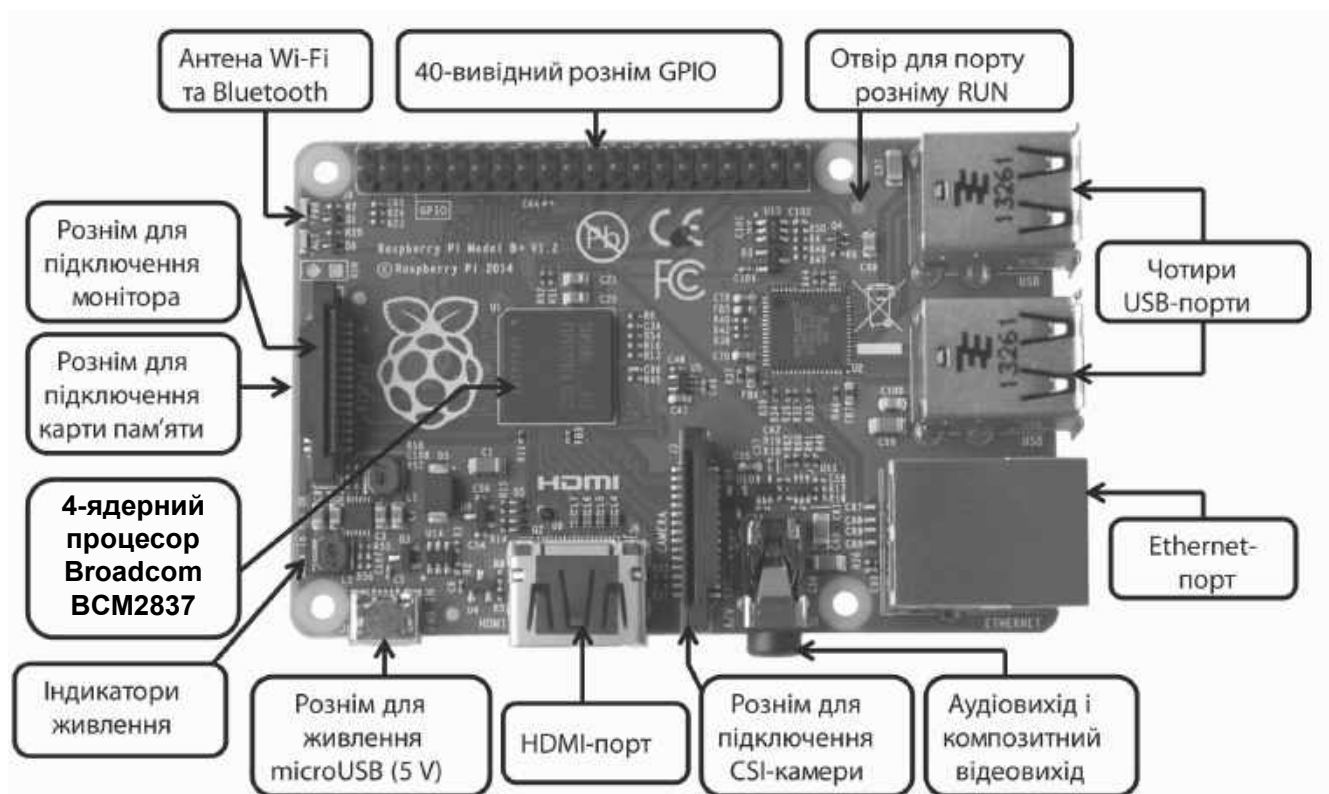


Рис. 1.7. Схема призначення рознімів для підключення зовнішніх пристроїв

Оскільки порти **GPIO** (general purpose input/output) на **Raspberry Pi** не підписані, приводимо роздрук цоколівки цього розніму (рис. 1.8). Це зав-

жди необхідно пам'ятати при створенні проектів з керування зовнішніми пристроями.

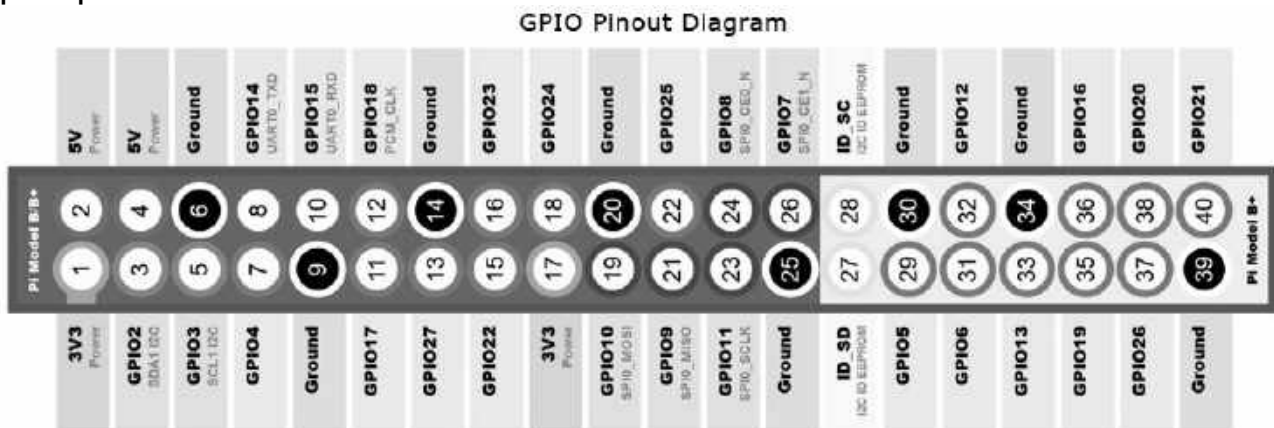


Рис. 1.8. Цоколівка розніму **GPIO** для **Raspberry Pi** (модель **2B+**)

Опишемо технічні характеристики **Raspberry Pi** (модель **2B+**) детальніше. Плата має чотириядерний процесор **ARM Cortex-A7** з частотою 900 МГц, збільшену оперативну пам'ять – 1 Гб. На платі встановлено чотири **USB**-порти, що дає змогу підключити до **Raspberry Pi** ще більше пристроїв. Інтерфейс **GPIO** має 40 пін, перші 26 контактів є ідентичними контактам моделі **A**, що забезпечує 100-відсоткову зворотну сумісність проектів. У пристрої використовується слот для карти пам'яті **MicroSD** для зберігання інформації й завантаження операційних систем, так само є **HDMI**-рознім і два діоди для виведення системної інформації. Сенсацією нової "малинки" стала підтримка Windows 10, однак поки що лише в консольному її вигляді. Платформа **Raspberry Pi 2B+** є придатною для вирішення завдань створення систем технічного зору, до цієї моделі існує безліч зручної периферії.

Необхідна комплектність. Для успішної роботи з **Raspberry Pi** окрім самої плати потребуються такі аксесуари:

- SD-карта для завантаження операційної системи; виробник рекомендує використовувати карту від 8 до 32 Гб;
- монітор або телевізор з рознімами **HDMI**, **DVI** або **RCA** (лише для моделей **A** і **B**) і відповідно кабель **HDMI-HDMI**, **HDMI-DVI** або **RCA-RCA**, також можна використовувати **HDMI-VGA**-перетворювач;
- **USB**-клавіатура;
- **USB**-мишка;
- кабель живлення або акумулятор **microUSB**.

1.3. Операційні системи для Raspberry Pi і програми NOOBS

Виробник рекомендує шість операційних систем (за посиланнями – офіційні сайти систем):

- **Raspbian**, що базується на **Debian Wheezy**;
- **Pidora**, що ґрунтується на **Fedora Remix**;
- **Arch Linux**;

- **Risc OS**;
- Arm-версія **OPENELEC**, **XMBC**-медіацентру;
- **RASPBMC**, **XMBC**-медіацентр.

Із перелічених **ОС** виробником рекомендується використовувати **Raspbian**. Дотримуватимемося цієї рекомендації. **Raspbian** та останні перелічені операційні системи включено до складу **NOOBS**. Найпростіший спосіб установити **ОС** на **Raspberry Pi** – використовувати **NOOBS** – програму, що містить дистрибутиви перелічених вище операційних систем і є зрозумілою навіть для новачка. Програма **NOOBS** розробляється **Raspberry Pi Foundation**, і скачати її можна безкоштовно на офіційному сайті за посиланням <https://www.raspberrypi.org/downloads/>. Процедuru скачування показано на рис. 1.9.

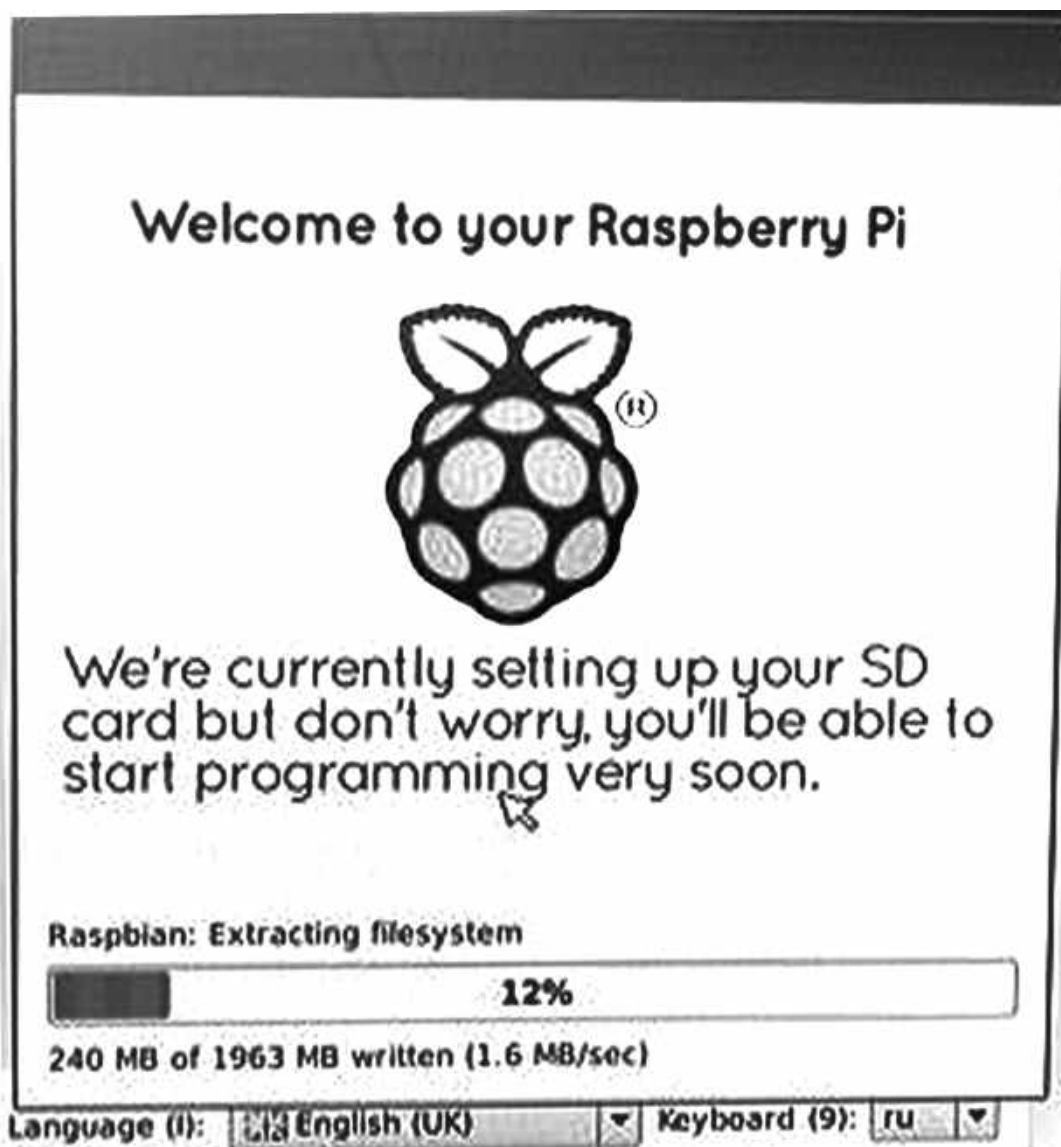


Рис. 1.9. Процедура скачування операційної системи для Raspberry Pi

Нагадаємо, що як постійна пам'ять для **Raspberry Pi** використовується **SD**-карта. На **SD**-карті зберігатиметься у тому числі й операційна система.

1.4. Установлення операційної системи Raspbian для комп'ютера Raspberry Pi

Оскільки плата комп'ютера **Raspberry Pi** не має встановленої операційної системи, її необхідно відразу встановити.

Існують три способи встановлення **ОС** на **Raspberry Pi**:

1. Покупка **SD**-карті з уже встановленою **Raspbian** або **NOOBS**. Особливого сенсу в цьому немає. Хоча за ціною така карта вийде не набагато дорожчою за звичайну **SD**-карту того ж розміру, але її доведеться чекати (якщо замовляти поштою).
2. Скачування програми **NOOBS** на **SD**-карту й установлення **ОС Raspbian** з неї.
3. Монтування зображення **ОС Raspbian** прямо на **SD**-карту. У цьому випадку можна буде приступати до використання відразу після її підключення.

Розглянемо порядок установлення **ОС** покроково.

Як зазначалось раніше, уся документація й інформація для початкової роботи з **Raspberry Pi 2** знаходиться на офіційному сайті <https://www.raspberrypi.org>. Проведемо установлення операційної системи **Raspbian** для комп'ютера **Raspberry Pi 2**.

Крок 1. Необхідно вибрати й скачати операційну систему (<https://www.raspberrypi.org/downloads/>). Скачуватимемо **Raspbian** – основну операційну систему для **Raspberry Pi**. На рис. 1.10 зображено вікно **DOWNLOADS** головної сторінки офіційного сайту на «звичайному» комп'ютері, а не на **Raspberry Pi 2**.

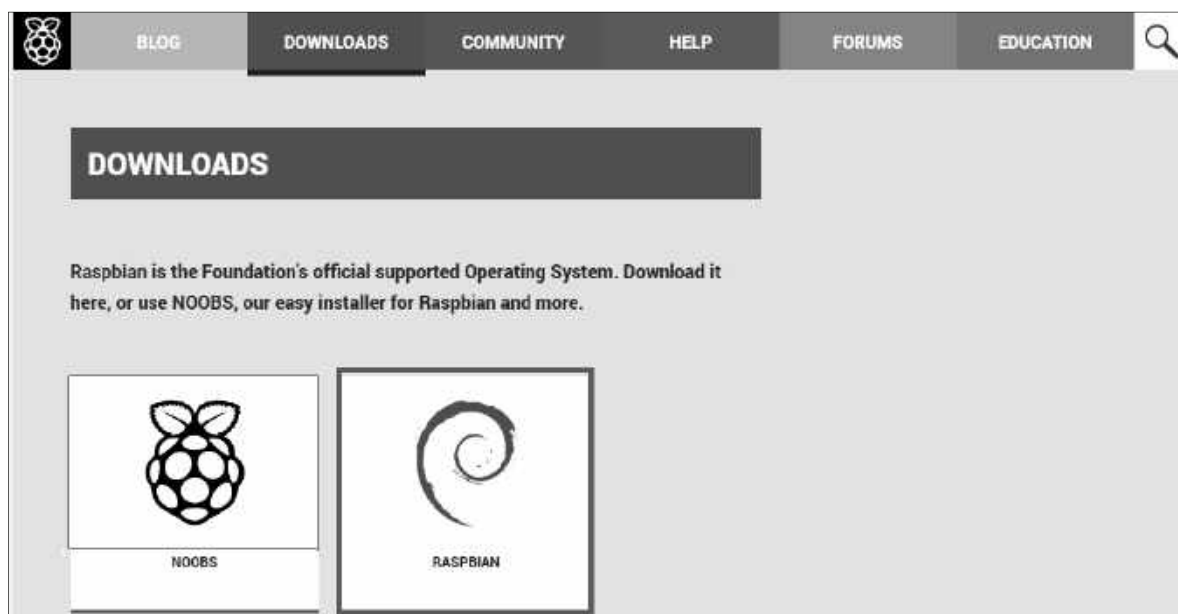


Рис. 1.10. Вікно скачування операційної системи **Raspbian**

Крок 2. Завантажений образ операційної системи необхідно записати на карту пам'яті **SD/microSD**. Вставляємо **SD**-карту в комп'ютер (не в

Raspberry Pi, а в «звичайний») і форматуємо її (виробник рекомендує **SDformatter**, але можна використовувати будь-які інші засоби, у тому числі ті, що належать до **Windows**-програми); при форматуванні вказуємо файловою систему **Fat32**. Це можна зробити за допомогою утиліти **Win32diskimager.exe** (<http://sourceforge.net/projects/win32diskimager/>).

Для цього запускаємо **Win32diskimager**, у полі **Device** вибираємо карту пам'яті, на яку записується образ. Потім клацанням лівої кнопки мишки на піктограмі з синьою текою (рис. 1.11) вказуємо шлях до образу **ОС**.

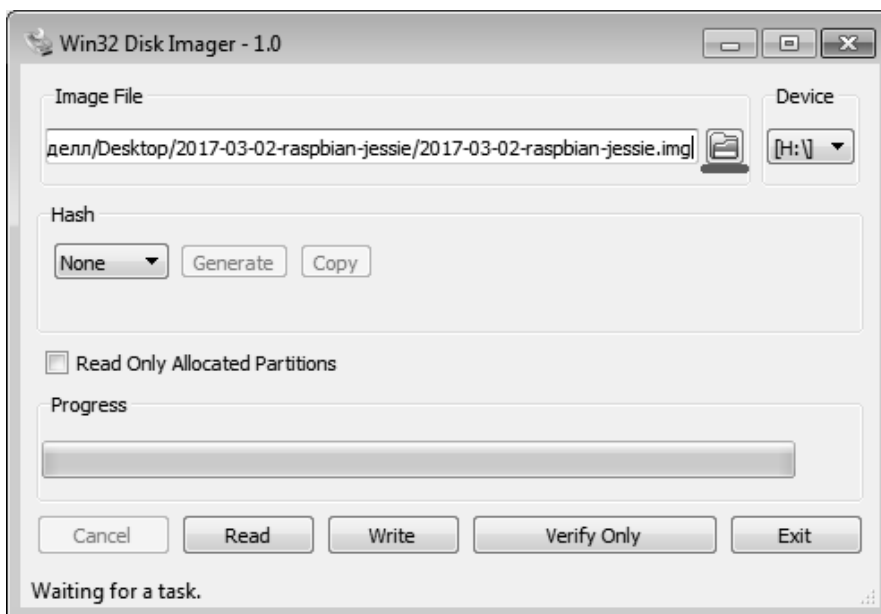


Рис. 1.11. Вибір карти пам'яті і шлях до образу **ОС**

Потім (рис. 1.12) записуємо образ **ОС** за допомогою кнопки **Write (2)**. Далі, використовуючи **Win32diskimager**, можна зробити резервне копіювання умісту **SD**-карти натисненням кнопки **Read (1)**.

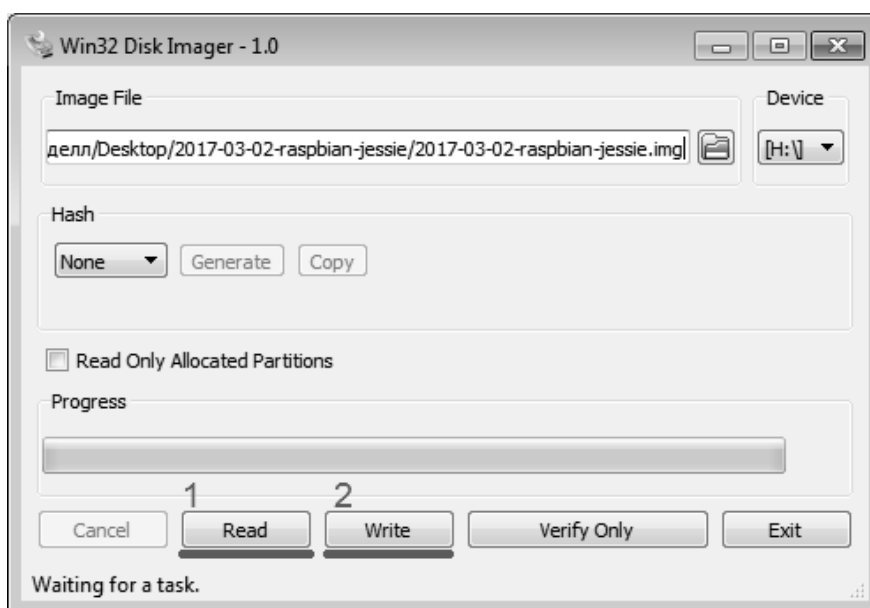


Рис. 1.12. Записування й резервне копіювання **SD**-карти

Крок 3. Вставляємо карту пам'яті в **Raspberry Pi** і підключаємо мікро-комп'ютер до джерела живлення через **microUSB**-вхід. Джерело живлення повинне видавати постійний струм напругою 5 В і силою мінімум 1 А (але краще більше, наприклад, 2 А). У всякому разі, можна підключити **Raspberry Pi** до **USB**-порту комп'ютера або ноутбука. Також необхідно підключити дисплей і **USB**-клавіатуру. За їх відсутності комп'ютером **Raspberry Pi** можна керувати віддалено в мережі за протоколом **SSH**. При правильному встановленні операційної системи можна виконати перше підключення **Raspberry Pi**. Якщо всі дії було виконано коректно, то з'явиться завантажувальний екран (рис. 1.13) після встановлення операційної системи **Raspbian**.



```

[ 3.286397] [c01c1f90] (ext4_lookup) from [c014c018] (lookup_real+0x30/0x5c)
[ 3.297203] [c014c018] (lookup_real) from [c014cbb0] (__lookup_hash+0x41/0x4c)
[ 3.308355] [c014cbb8] (__lookup_hash) from [c014ec38] (lookup_slow+0x48/0xb4)
[ 3.319425] [c014ec30] (lookup_slow) from [c014f60c] (path_lookupat+0x6e0/0x738)
[ 3.330678] [c014f60c] (path_lookupat) from [c014ff98] (filename_lookup.isra.46+0x30/0x70)
[ 3.342818] [c014ff98] (filename_lookup.isra.46) from [c0152130] (user_path_at_empty+0x64/0x8c)
[ 3.355400] [c0152130] (user_path_at_empty) from [c015217c] (user_path_at+0x24/0x2c)
[ 3.367063] [c015217c] (user_path_at) from [c0141f18] (SyS_faccessat+0xa0/0x1d8)
[ 3.378378] [c0141f18] (SyS_faccessat) from [c0142070] (SyS_access+0x28/0x2c)
[ 3.389433] [c0142070] (SyS_access) from [c0000ebc0] (ret_fast_syscall+0x0/0x48)
[ 3.400611] Code: e7934004 e3540000 0a00004c e5963014 (e794e003)
[ 3.410367] ---[ end trace de0305cb0d5102c5 ]---
[ 3.413518] usb 1-1: New USB device found, idVendor=0424, idProduct=9514
[ 3.413525] usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
[ 3.414105] hub 1-1:1.0: USB hub found
[ 3.414178] hub 1-1:1.0: 5 ports detected
[ 3.454547] Kernel panic - not syncing: Attempted to kill init! exitcode=0x0000000b
[ 3.454547]
[ 3.470682] CPU0: stopping
[ 3.476842] CPU: 0 PID: 0 Comm: swapper/0 Tainted: G      D      3.18.7-v7+ #755
[ 3.487908] [c0016d14] (unwind_backtrace) from [c0012c40] (show_stack+0x20/0x24)
[ 3.499161] [c0012c40] (show_stack) from [c0052efc0] (dump_stack+0x90/0xd0)
[ 3.509881] [c0052efc0] (dump_stack) from [c001509c] (handle_IPI+0x234/0x268)
[ 3.520759] [c001509c] (handle_IPI) from [c0000618] (do_IPI+0x10/0x1c)
[ 3.531122] [c0000618] (do_IPI) from [c005319b4] (__irq_svc+0x34/0x14c)
[ 3.541471] Exception stack(0xb07cbf08 to 0xb07cbf50)
[ 3.550003] b700: 007e9ccc 00000000 ffffffff 00000000 b07ca020 b07e8dd4

```

Рис. 1.13. Завантажувальний екран **OC Raspbian**

Операційна система містить стандартні програми (**Python 2.7, 3.6, Java, C+**) для математичних обчислень (**Wolfram, Mathematica**), редагування й перегляду текстових документів (**Openoffice**).

1.5. Первинні налаштування після встановлення ОС Raspbian

Первинні налаштування після встановлення **ОС** можна провести двома способами:

1. Налаштування режимів через вікно конфігурації шляхом виклику їх у терміналі командою «**Sudo raspi-config**» (рис. 1.14, 1.15).
2. Налаштування в меню «**Configuration**» (рис. 1.16).



Рис. 1.14. Вікно конфігурації **OC**

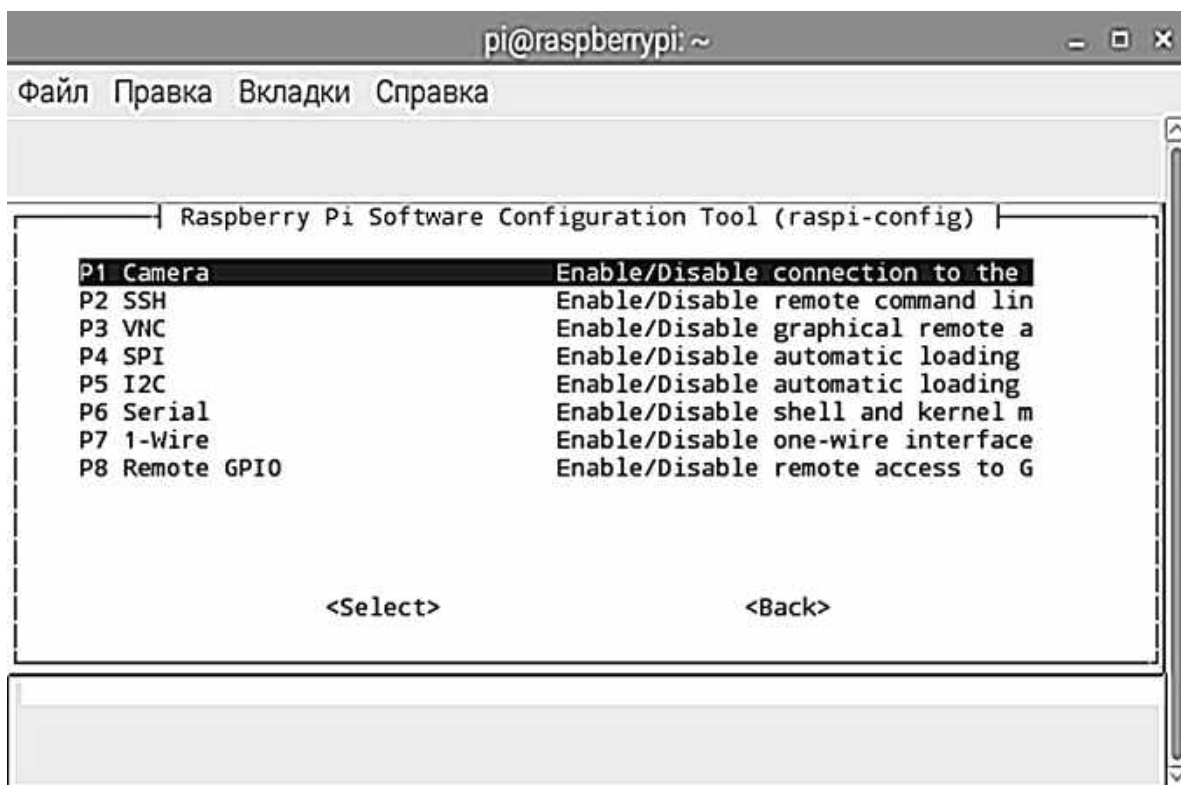


Рис. 1.15. Вікно конфігурації **OC** детальніше



Рис. 1.16. Налаштування в меню «**Configuration**»

Здійснюємо потрібні нам налаштування, такі, як підключення **web**-камери, **SSH**, **GPIO**, **I²C**, **SPI**. Розберемо всі пункти меню налаштування по черзі (для вибору пунктів використовуються клавіші зі стрілками вгору/вниз, для вибору пункту меню – **Enter**):

- **Expand filesystem** – розширення основного розділу на всю карту пам'яті. Просто натискаємо **Enter**.
- **Change User Password** – установлення пароля для користувача "**pi**" (за умовчанням пароль "**raspberrypi**"). При введенні пароля символи, що вводяться, не відображаються, пароль необхідно ввести двічі (другий раз для підтвердження коректності введення).
- **Enable Boot to Desktop** – завантажувати/не завантажувати графічний інтерфейс. Зазвичай відключається, оскільки графічний інтерфейс у будь-який момент можна запустити, і в основному робота з **Raspberry Pi** відбувається через консоль. Після вибору "**Enable Boot to Desktop**" з'явиться додаткове діалогове вікно: перший рядок – не завантажувати графічний інтерфейс; другий – завантажувати.
- **Internationalisation Options** – вибір мови й місцезростаювання. Після вибору з'явиться діалогове вікно з трьома пунктами:
 - **Change Locale** – вибір мови. Вибираємо "**ru_ru.UTF-8 Utf-8**" (переглядаємо до потрібного пункту за допомогою стрілки, а потім

вибираємо потрібний пункт клавішею "пропуск", після натиснення кнопки **Tab** курсор перейде на **Ok**, що підтверджуємо натисненням кнопки **Enter**. Далі в діалоговому вікні вибираємо **ru_ru.UTF-8**, переходимо до **Ok** клавішею **Tab** і вибір підтверджуємо клавішею **Enter**;

- **Change Timezone** – вибір часового поясу. Вибираємо зі списку, після виділення потрібного міста натискаємо кнопку **Tab** для переведення курсора на **Ok**;
- **Change Keyboard Layout** – налаштування клавіатури (можна залишити без змін).
- **Enable Camera** – підключення підтримки камери. Цей пункт виконуємо, якщо необхідно використовувати відеокамеру.
- **Add to Rastrack** – реєстрація пристрою в загальній базі. Зазвичай необхідність у цьому не виникає.
- **Overclock** – прискорення процесора. За умовчанням установлена частота дорівнює 700 МГц, її можна збільшувати до 1000 МГц (доступними є проміжні режими). Без необхідності не слід збільшувати частоту, краще просто пропустити пункт – це завжди можна буде зробити згодом.
- **Advanced Options** – додаткові налаштування. Після вибору цього рядка відкриється діалогове вікно з такими пунктами:
 - **Overscan** – налаштування «стислої розгортки» (зазвичай необхідно в'язкове);
 - **Hostname** – мережне ім'я **Raspberry Pi** (за умовчанням "**raspberrypi**") можна залишити без змін);
 - **Memory Split** – виділення пам'яті для роботи графічного процесора (якщо збираєтеся працювати в консолі, достатньо 16 Мб, для роботи в графічній оболонці краще використовувати 64 Мб);
 - **SSH** – віддалений доступ за **SSH** (для зручного використання й подальшого налаштування слід підключити **Enable**);
 - **Update** – оновлення операційної системи (потребується доступ до мережі інтернету).
 - **About raspi-config** – загальна інформація про це меню налаштувань (пропускаємо).

Клавішею **Tab** вибираємо команду **Finish**, пристрій перевантажиться і можна починати роботу. Після перезавантаження буде запитано логін – вводимо **pi**, після буде запитано пароль **piip** (якщо ви не задали свій, то за умовчанням це буде пароль **raspberry**). Після цих дій **Raspberry Pi** буде готовий до роботи.

Основні команди **Raspberry Pi** прописуються у вікні «**Terminal**» – так званому **командному вікні**. На рис. 1.17 добре видно, де знаходиться це вікно (кнопку підкреслено червоною лінією).



Рис. 1.17. Вибір вікна «Terminal»

Покажемо приклад вікна «**Terminal**» з прописаною командою й підкресленим полем, у яке потрібно вписувати наступну команду (рис. 1.18). Потрібні рядки підкреслено червоними лініями.

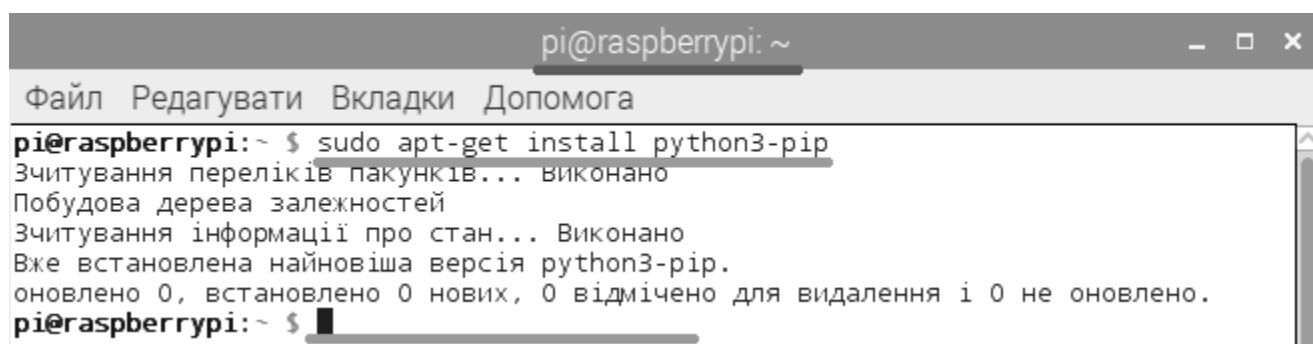


Рис. 1.18. Робота у вікні «Terminal»

Подано список основних команд для керування Raspberry:

- **top** – запуск диспетчера завдань, уже встановленого в **Raspbian**;
- **sudo raspi-config** – запуск первинного меню налаштувань;
- **sudo passwd root** – створення пароля для користувача **root**;
- **startx** – запуск графічної оболонки;
- **sudo halt** – виключення **Raspberry Pi**;
- **logout** – вихід із системи;
- **sudo reboot** – перезавантаження **Raspberry Pi**;
- **cd** – перехід у необхідну директорию, наприклад, для переходу в директорию **/etc/network/** – "**cd /etc/network/**";
- **pwd** – шлях до поточної директорії;
- **dir** – зміст поточної директорії;
- **mkdir** – створення директорії, наприклад: «**Mkdir /home/pitest/**» – створення директорії «**pitest**»;
- **rmdir** – видалення директорії, наприклад: «**Mdir /home/pitest/**» – видалення директорії «**pitest**»;
- **cat** – відкриття файла для читання, наприклад: «**Cat /etc/network/interfaces**» покаже зміст файла «**interfaces**»;

- **nano** – відкриття файлу для редагування, наприклад: «**Nano /etc/network/interfaces**» відкриє для редагування файл «**interfaces**»;
- **ifconfig** – відображення поточної конфігурації мережі;
- **df** – виведення в консолі вільного й використовуваного дискового простору для всіх розділів файлової системи;
- **clear** – очищення екрана термінала;
- **Ctrl+ins** (поєднання клавіш) – копіювання виділеного (тексту);
- **Shift+ins** (поєднання клавіш) – вставлення з буфера (тексту);
- **sudo** – виконання команд з правами **root**-користувача, наприклад, це актуально, якщо ви зайшли під ім'ям користувача **pi** і хочете з консолі відредагувати який-небудь системний файл, – введіть «**sudo nano шлях_до_файла**»;
- **Ctrl+C** (поєднання клавіш) – зупинення поточної дії/вихід з консольного додатка;
- **sudo apt-get update** – оновлення списку доступних пакетів;
- **sudo apt-get upgrade** – оновлення встановлених пакетів;
- **sudo apt-get install** – установлення необхідного пакета (наприклад, для установлення консольного браузера **Links** уводимо «**sudo apt-get install links**»).

На рис. 1.19 показано налаштування, які слід обов'язково підключити для вирішення завдань технічного зору.

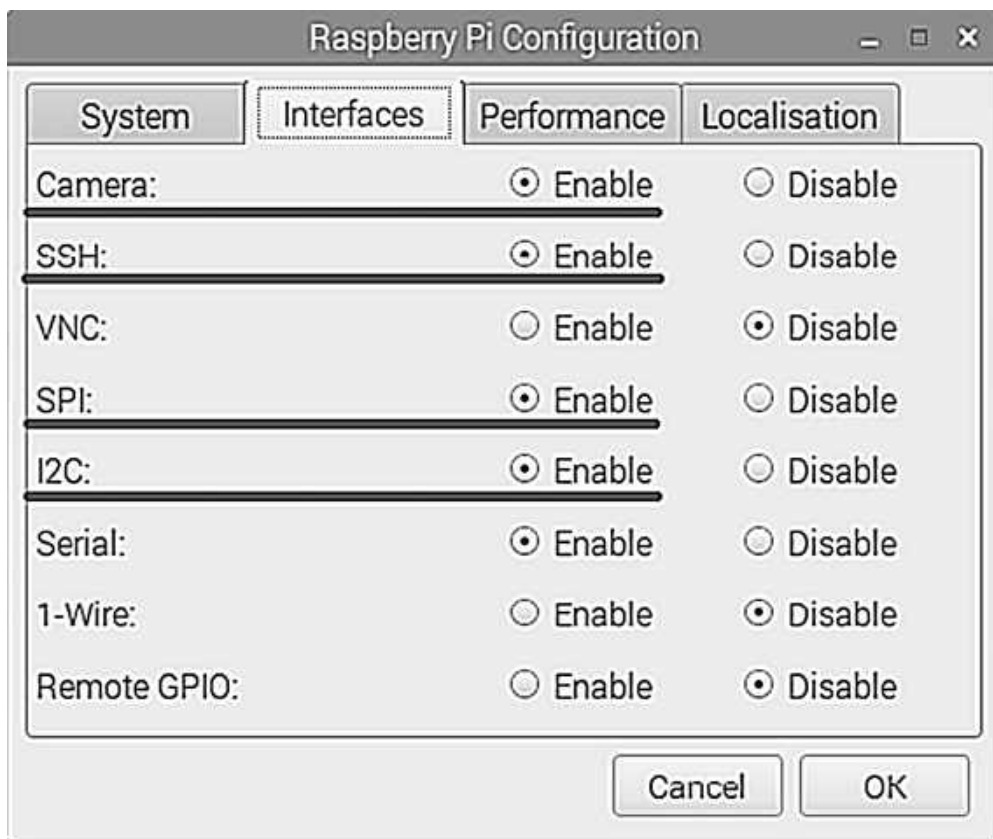


Рис. 1.19. Обов'язкові налаштування для систем технічного зору

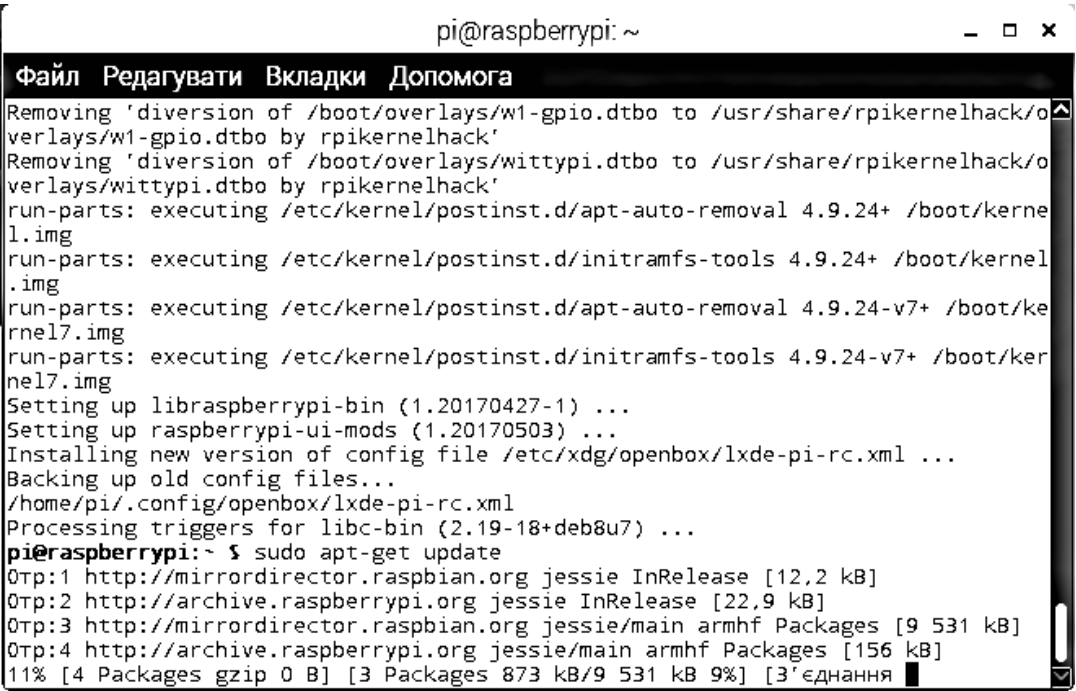
GPIO (General Purpose Input Output) – це низькорівневий інтерфейс уведення-виведення прямого керування. На платі **Raspberry** цей інтерфейс у вигляді гребінки з 40 штирками знаходиться в кутку поряд з відеовиходом, тобто через цей інтерфейс **Raspberry** може слухати і давати команди будь-якому зовнішньому пристрою, наприклад роботу.

SSH (англ. Secure Shell – *безпечна оболонка*) – мережний протокол прикладного рівня, що дає змогу виконувати віддалене керування операційною системою і тунелювання **TCP**-з'єднань (наприклад, для передавання файлів). Знадобиться для подальшого передавання файлів.

SPI (англ. Serial Peripheral Interface, SPI bus – *послідовний периферійний інтерфейс, шина SPI*) – послідовний синхронний стандарт передавання даних у режимі повного дуплекса, призначений для забезпечення простого й недорогого високошвидкісного сполучення мікроконтролерів і периферії. **SPI** також інколи називають чотирипровідним (англ. four-wire) інтерфейсом.

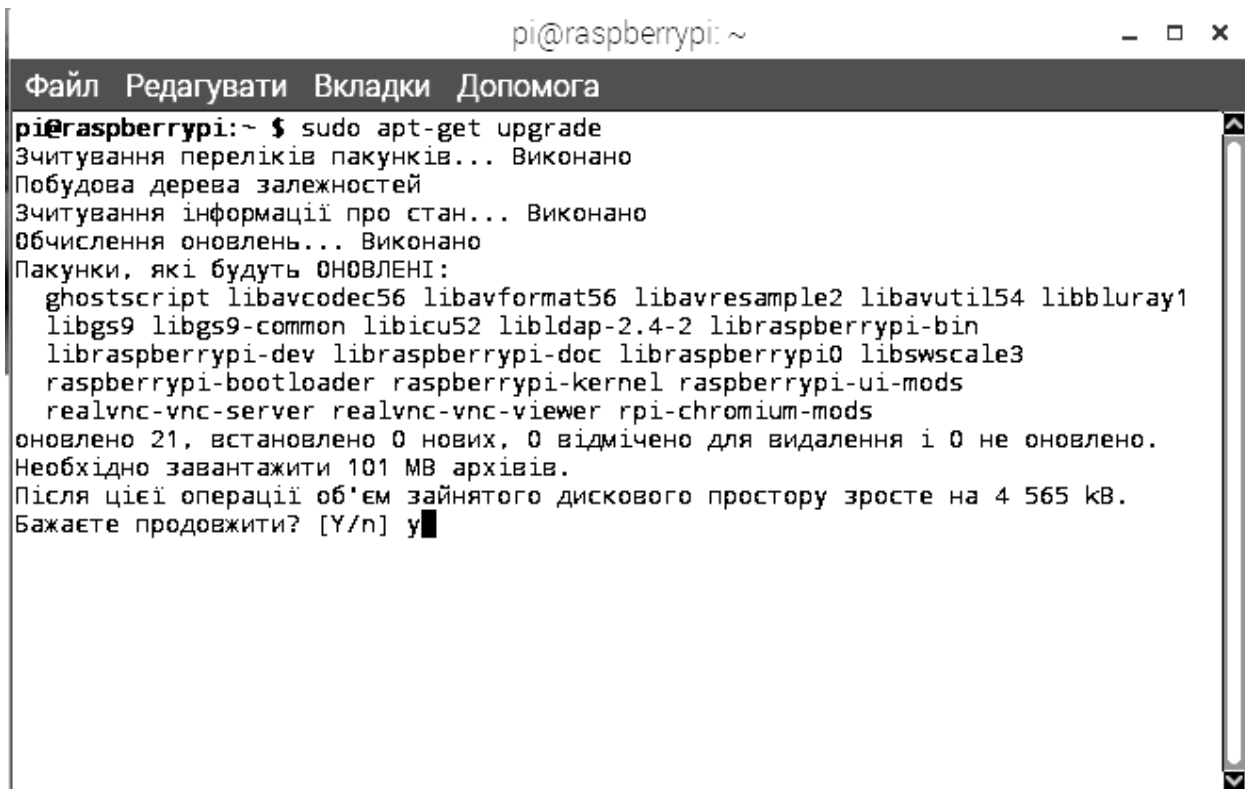
I²C (IIC, англ. Inter-integrated Circuit) – послідовна асиметрична шина для зв'язку між інтегральними схемами всередині електронних приладів. Має дві двонапрямлені лінії зв'язку (**SDA** і **SCL**) для з'єднання низькошвидкісних периферійних компонентів з процесорами й мікроконтролерами (наприклад, на материнських платах, у вбудованих системах, у мобільних телефонах).

Слід також перевірити оновлення системи і, якщо необхідно, виконати його. У терміналі за допомогою команди «**Sudo apt-get update**» скачують нові елементи системи або ті, яких не вистачає, а з допомогою команди «**Sudo apt-get upgrade**» встановлюють раніше скачані пакети (рис. 1.20–1.22).



```
pi@raspberrypi: ~
Файл Редагувати Вкладки Допомога
Removing 'diversion of /boot/overlays/w1-gpio.dtbo to /usr/share/rpikernelhack/o
verlays/w1-gpio.dtbo by rpikernelhack'
Removing 'diversion of /boot/overlays/wittydtbo to /usr/share/rpikernelhack/o
verlays/wittydtbo by rpikernelhack'
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.9.24+ /boot/kerne
l7.img
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.9.24+ /boot/kernel
7.img
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.9.24-v7+ /boot/ke
rnel7.img
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.9.24-v7+ /boot/ker
nel7.img
Setting up libraspberrypi-bin (1.20170427-1) ...
Setting up raspberrypi-ui-mods (1.20170503) ...
Installing new version of config file /etc/xdg/openbox/lxde-pi-rc.xml ...
Backing up old config files...
/home/pi/.config/openbox/lxde-pi-rc.xml
Processing triggers for libc-bin (2.19-18+deb8u7) ...
pi@raspberrypi:~$ sudo apt-get update
0tp:1 http://mirrordirector.raspbian.org jessie InRelease [12,2 kB]
0tp:2 http://archive.raspberrypi.org jessie InRelease [22,9 kB]
0tp:3 http://mirrordirector.raspbian.org jessie/main armhf Packages [9 531 kB]
0tp:4 http://archive.raspberrypi.org jessie/main armhf Packages [156 kB]
11% [4 Packages gzip 0 B] [3 Packages 873 kB/9 531 kB 9%] [З'єднання]
```

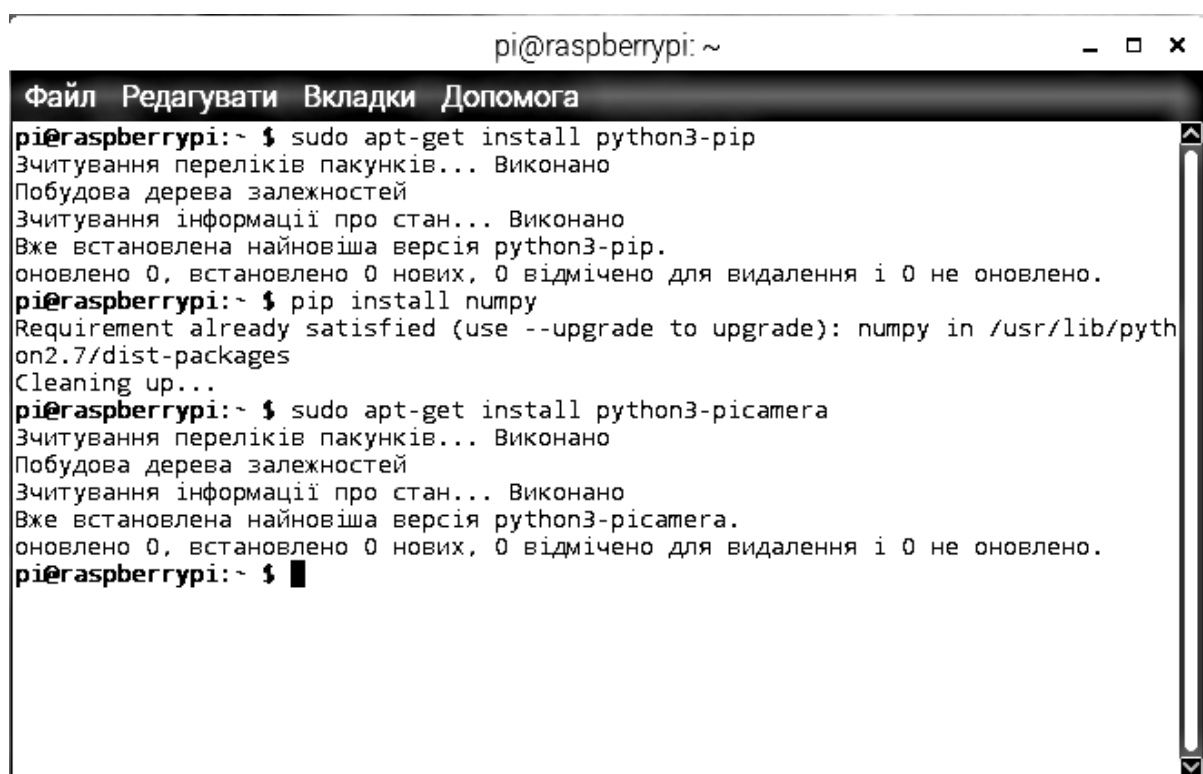
Рис. 1.20. Оновлення системи



```
pi@raspberrypi: ~  
Файл Редагувати Вкладки Допомога  
pi@raspberrypi:~ $ sudo apt-get upgrade  
Зчитування переліків пакунків... Виконано  
Побудова дерева залежностей  
Зчитування інформації про стан... Виконано  
Обчислення оновлень... Виконано  
Пакунки, які будуть ОНОВЛЕНІ:  
ghostscript libavcodec56 libavformat56 libavresample2 libavutil54 libbluray1  
libgs9 libgs9-common libicu52 libldap-2.4-2 libraspberrypi-bin  
libraspberrypi-dev libraspberrypi-doc libraspberrypi0 libswscale3  
raspberrypi-bootloader raspberrypi-kernel raspberrypi-ui-mods  
realvnc-vnc-server realvnc-vnc-viewer rpi-chromium-mods  
оновлено 21, встановлено 0 нових, 0 відмічено для видалення і 0 не оновлено.  
Необхідно завантажити 101 MB архівів.  
Після цієї операції об'єм зайнятого дискового простору зросте на 4 565 kB.  
Бажаєте продовжити? [Y/n] y
```

Рис. 1.21. Підтвердження встановлення оновлень ОС

За допомогою команди «***Sudo apt-get install***» можна встановити пакети, необхідні для роботи в середовищі програмування ***Python***, – ***numpy***, ***pip***, ***pi-camera*** та ін.



```
pi@raspberrypi: ~  
Файл Редагувати Вкладки Допомога  
pi@raspberrypi:~ $ sudo apt-get install python3-pip  
Зчитування переліків пакунків... Виконано  
Побудова дерева залежностей  
Зчитування інформації про стан... Виконано  
Вже встановлена найновіша версія python3-pip.  
оновлено 0, встановлено 0 нових, 0 відмічено для видалення і 0 не оновлено.  
pi@raspberrypi:~ $ pip install numpy  
Requirement already satisfied (use --upgrade to upgrade): numpy in /usr/lib/python2.7/dist-packages  
Cleaning up...  
pi@raspberrypi:~ $ sudo apt-get install python3-picamera  
Зчитування переліків пакунків... Виконано  
Побудова дерева залежностей  
Зчитування інформації про стан... Виконано  
Вже встановлена найновіша версія python3-picamera.  
оновлено 0, встановлено 0 нових, 0 відмічено для видалення і 0 не оновлено.  
pi@raspberrypi:~ $
```

Рис. 1.22. Установлення пакетів для роботи з ***Python***

2. ЗАСОБИ ПРОГРАМУВАННЯ МОБІЛЬНИХ СИСТЕМ ТЕХНІЧНОГО ЗОРУ

2.1. Вибір мови програмування

При реалізації проектів зі створення сучасних мобільних систем технічного зору найважливішим питанням є оптимальний вибір мови програмування. Як зазначалося раніше, при використанні платформи **Raspberry Pi** з **ОС Raspbian** існує можливість вибору декількох мов програмування. Це мови **C, Java, Python, PHP, C#, Ruby, Perl, Scratch, Lazarus / Free Pascal, BASIC**, кожна з яких має свої переваги й недоліки. Проте, специфічна постановка завдання (практична реалізація систем технічного зору) визначає вибір на користь мови **Python**. Це зумовлено багатьма причинами, які слід розглянути детальніше.

Мова програмування **Python** має простий і зрозумілий синтаксис і тому легко вчиться й добре підходить для виконання проекту. Мова близька до **Matlab** і є придатною для програмування математичних обчислень. До того ж **Python** уміє працювати з такими мовами, як **Fortran, C** і **C++**, що широко використовуються в наукових та інженерних розрахунках.

Python як сучасна об'єктно-орієнтована мова найчастіше використовується для програмування **GPIO** на **Raspberry Pi** і входить до складу операційної системи **Raspbian**.

До **Python** можна підключити велику кількість спеціалізованих бібліотек, що розширюють можливості вирішення потрібних застосувань. Наприклад, бібліотека **Python Imaging Library (Pillow)** є дуже ефективною при обробленні й аналізі зображень, а це – першочергове завдання систем технічного зору.

Особливо важливим є те, що у в'язці з **Python** можна використовувати **OpenCV** – бібліотеку алгоритмів комп'ютерного зору, оброблення зображень і числових алгоритмів загального призначення з відкритим кодом.

Бібліотека **OpenCV** надає більше 2500 алгоритмів, як класичних, так і таких, що відображають останні досягнення в області комп'ютерного зору й систем машинного навчання. Бібліотека може застосовуватися для розпізнавання об'єктів на фото й відео (наприклад, розпізнавання осіб і фігур людей, тексту й т. ін.), відслідковування руху об'єктів і камери, класифікації дій на відео, перетворення зображень, формування 3D-простору із зображення від стереокамер, пошуку на зображенні об'єктів, схожих на поданий набір елементів, використання методів машинного навчання, розставляння маркерів.

Крім того, більшість продуктів і модулів, написаних на мові **Python**, розповсюджуються безкоштовно, у тому числі й усі ті інструменти, які ми будемо використовувати.

2.2. Установлення програми-інтерпретатора Python

Для запуску програм на мові **Python** необхідно є програма-інтерпретатор (віртуальна машина **Python**). Ця програма приховує від

python-програміста всі особливості операційної системи, тому, написавши програму на **Python** в системі **Windows**, її можна запустити, наприклад, у **GNU/Linux** й отримати такий же результат.

Скачати і встановити інтерпретатор **Python** можна безкоштовно з офіційного сайту <http://python.org>. Для роботи нам знадобиться інтерпретатор **Python** версії 3 або вище.

2.2.1. Установлення Python 3.6.1 для операційної системи Windows 7

Для успішного початку роботи необхідно скачати з офіційного сайту <https://www.python.org/downloads/> версію програми 3.6.1. На рис. 2.1 показано головне вікно цього сайту. Виберіть пункт меню **Downloads**, а в списку версій – пункт **Python 3.6.1**. Тут і далі потрібні пункти меню підкреслено червоними лініями.

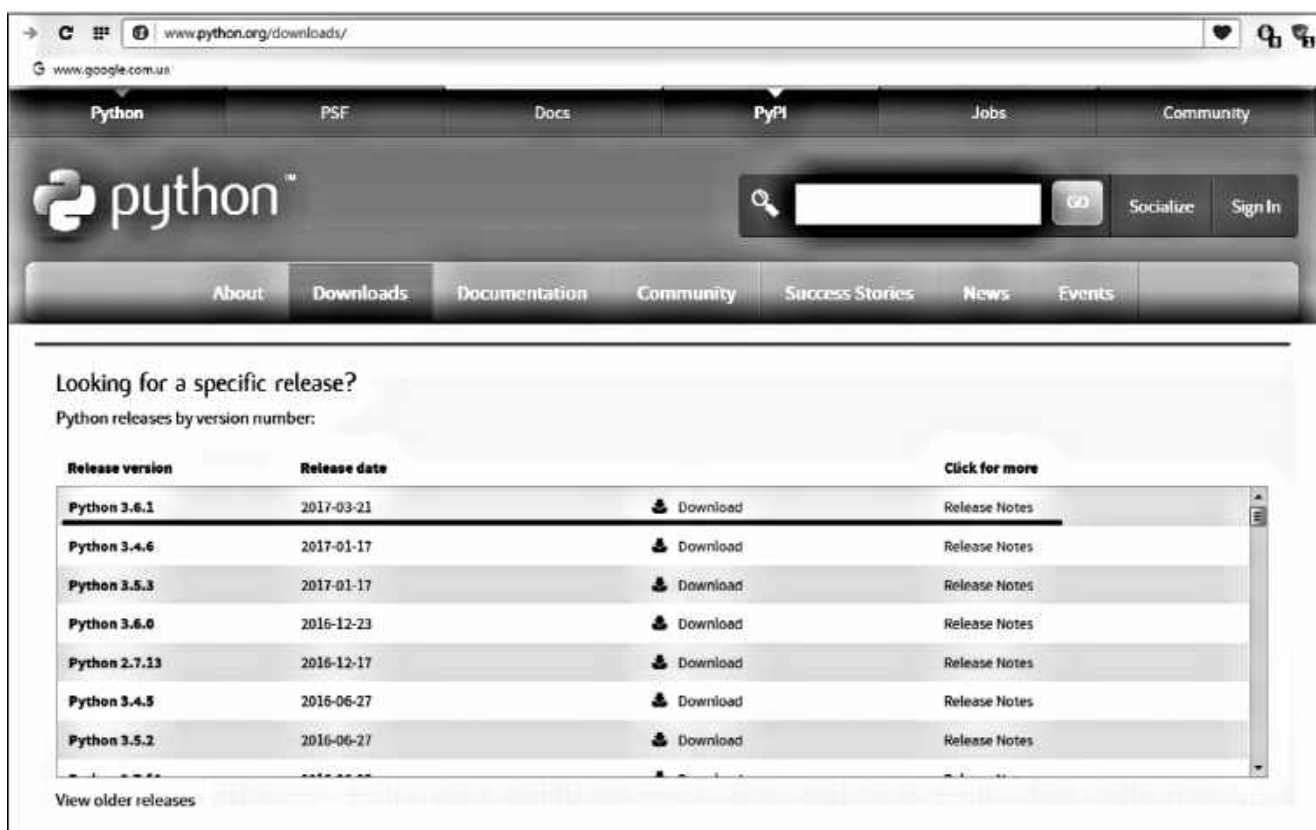


Рис. 2.1. Головне вікно сайту <https://www.python.org>

Після вибору версії **Python 3.6.1** необхідно натиснути кнопку **Download**. У вікні **Files** (рис. 2.2), що з'явилося, виберіть операційну систему, що використовується на вашому комп'ютері. Її розрядність повинна відповідати вашій версії **Windows**. У нашому прикладі це **Windows x86 executable installer**. Цей пункт меню також підкреслено червоною лінією.

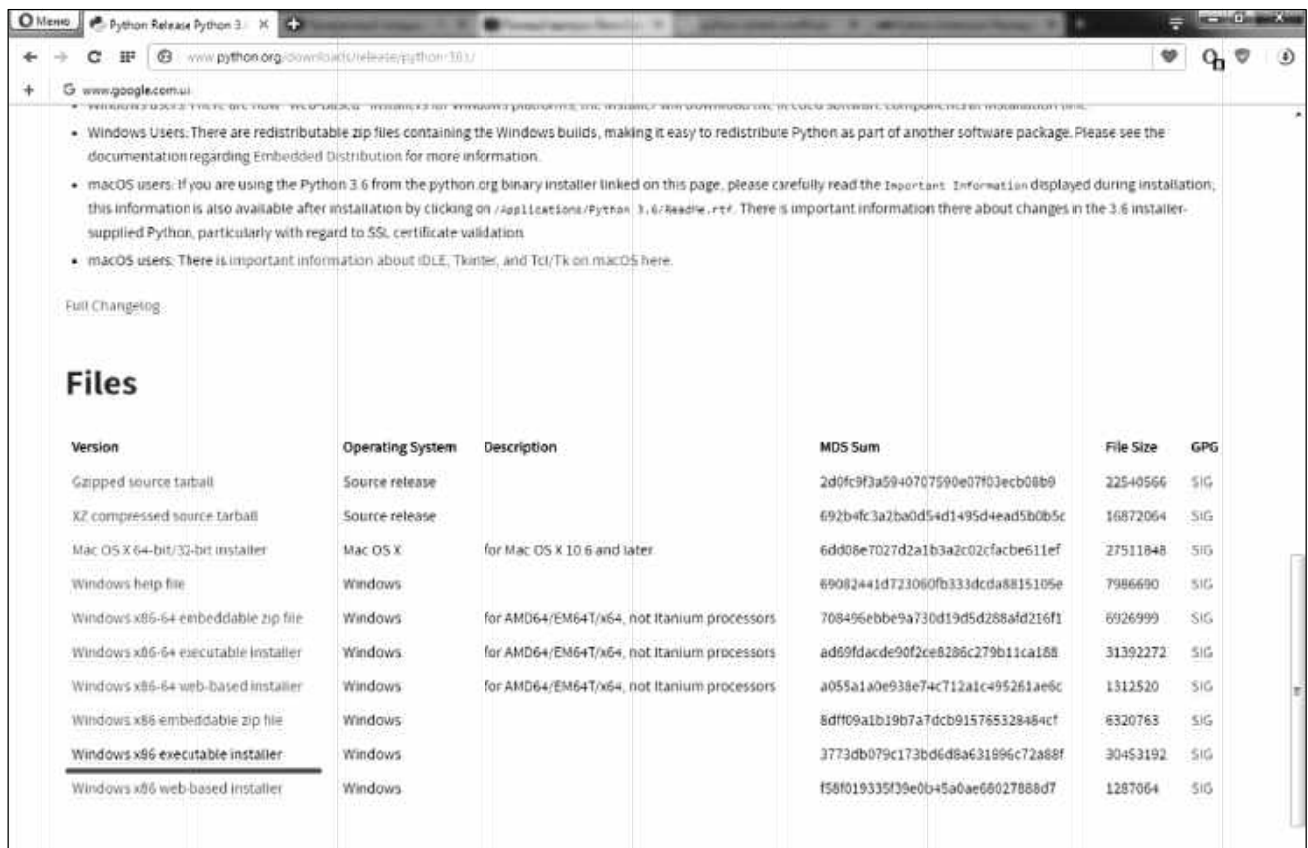


Рис. 2.2. Вікно вибору версії **Windows**

Після скачування установника запустіть файл `python-3.6.1.exe` і дотримуйтесь інструкцій з установлення, як показано далі (рис. 2.3, 2.4).

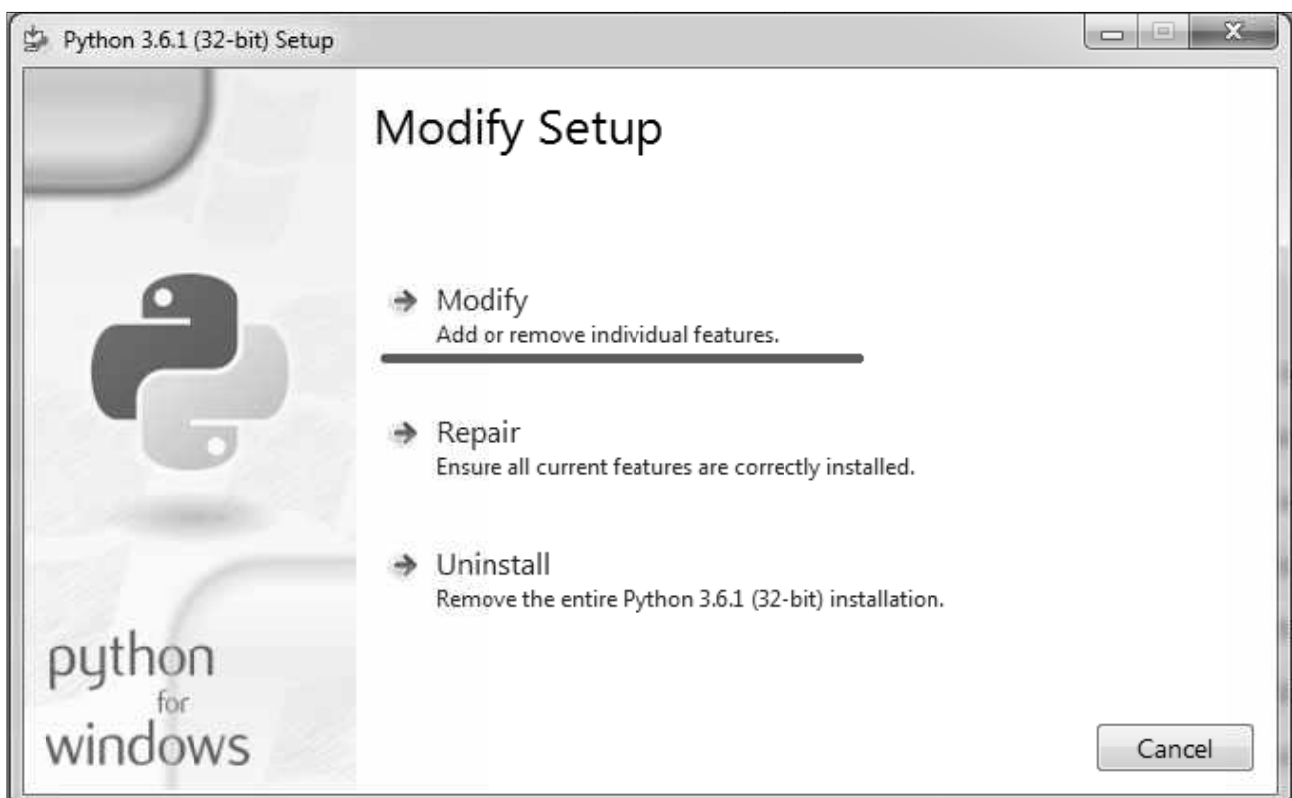


Рис. 2.3. Перше вікно встановлення інтерпретатора **Python**

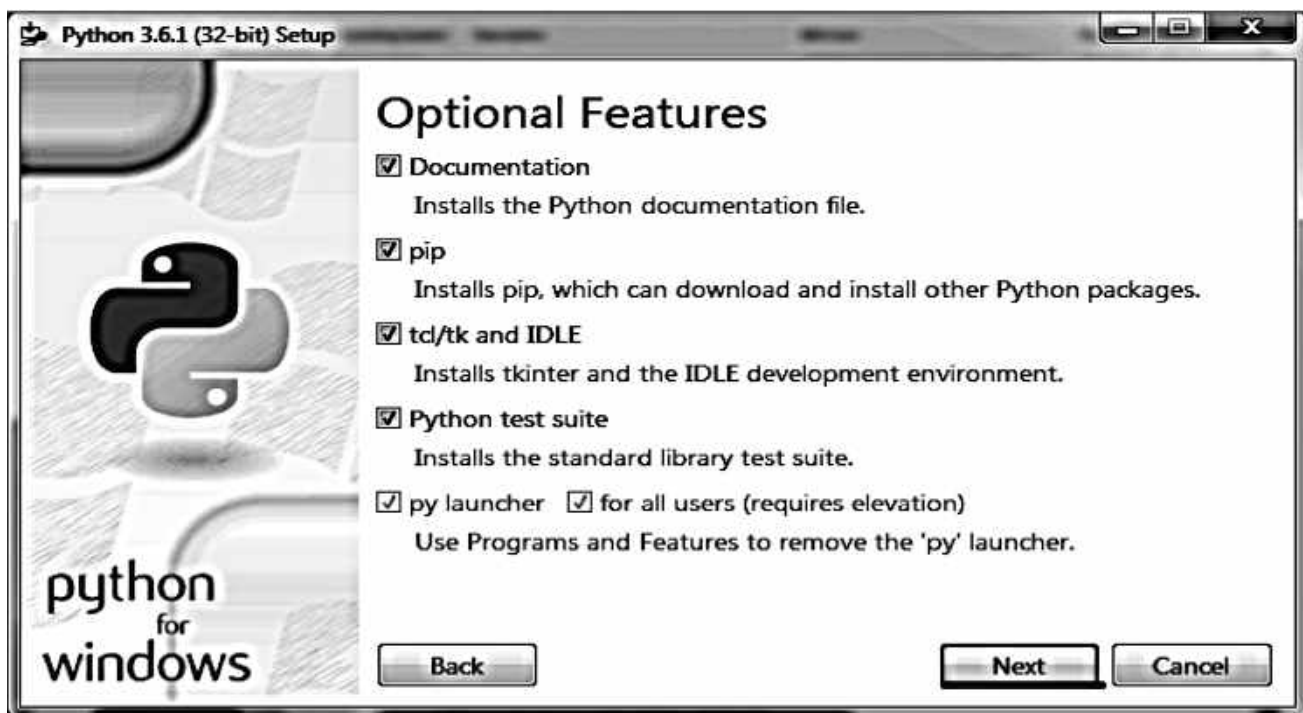


Рис. 2.4. Друге вікно встановлення інтерпретатора **Python**

У вікні **Advanced Options** виберіть опції, необхідні для роботи із програмою (рис. 2.5).

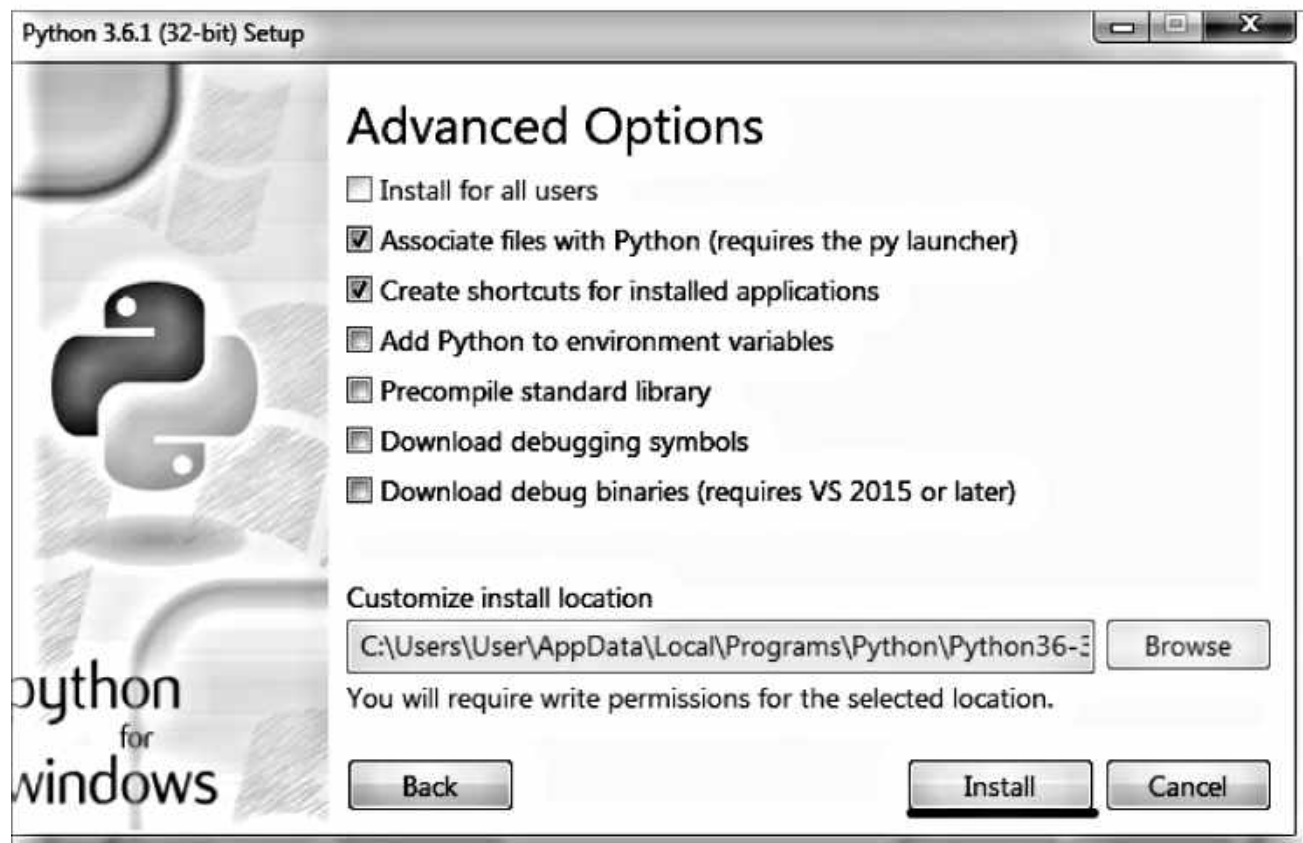


Рис. 2.5. Вікно вибору опцій

Після встановлення програми запусить інтерактивне графічне середовище **IDLE (Python 3.6.1 64-bit).exe** і дочекайтеся появи запрошення для введення команд `>>>` (рис. 2.6).



Рис. 2.6. Вікно інтерпретатора

2.2.2. Особливості встановлення програми Python 3.6.1 для ОС Raspbian на комп'ютер Raspberry Pi

Зручність роботи з програмою **Python** полягає в тому, що є можливість роботи з операційною системою **Windows** на персональному комп'ютері і з операційною системою **Raspbian** на мікрокомп'ютері **Raspberry Pi**. Код **Python** однаково успішно використовується в обох випадках.

Програма **Python** на комп'ютер **Raspberry** установлюється разом з операційною системою (див. підрозд. 1.4, 1.5). Особливість використання **Python** на комп'ютері **Raspberry Pi** полягає в тому, що при вирішенні завдань синтезу систем технічного зору необхідним є використання деяких бібліотек **Python**, які мало застосовуються в звичайних розрахунках, – **numpy**, **PIL**, **pi-camera** та ін. Детально процедури підключення цих бібліотек опишемо в підрозд. 2.7.

2.3. Програмування в інтерактивному режимі

Інтелектуальний калькулятор. На початку навчання програму **Python** можна використовувати як звичайний інтерактивний калькулятор. В інтерактивному режимі **IDLE** знайдемо значення для різних математичних виразів. Набравши вираз, натисніть клавішу **Enter** для завершення введення й виведення результату на екран:

```
>>> 5 + 7
12
>>> 3 + 11
14
>>> 2 - 7
-5
>>> _ +9
4
>>>
```

Нижнім підкресленням у попередньому прикладі позначено останній отриманий результат.

Якщо з якої-небудь причини зробити помилку під час введення команди, то **Python** повідомить про це:

```
>>> a
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
a
NameError: name 'a' is not defined
>>>
```

Математичні оператори, доступні для роботи з числами в **Python**, наведено в табл. 2.1.

Таблиця 2.1

Оператор	Опис
+	Додавання
-	Віднімання
/	Ділення (у результаті – дійсне число)
//	Ділення з округленням униз
**	Піднесення до степеня
%	Остача від ділення

Зазвичай одна команда **Python** вводиться на окремому рядку після знака запрошення '>>>'. Але можна набирати декілька команд в одному рядку, тоді їх потрібно розділяти символом «;» (крапка з комою):

```
>>>x=5;y=10; x+y # Три інструкції на одному рядку
15
```

Текст після символу # є коментарем. Коментарі призначені для вставлення пояснень у текст програми, і інтерпретатор повністю їх ігнорує. У мові **Python** є лише однорядковий коментар, який починається символом #.

Рядки виводяться функцією `print(...)`:

```
>>> print("Hello world!")
Hello world!
```

Рядок – це набір символів, узятих у подвійні або одинарні лапки. Можна просто ввести рядок і натискувати клавішу **Enter** для отримання результату:

```
>>>"Привіт, світ!"
'Привіт, світ!'
```

Результуючий рядок відображатиметься в лапках. Цього не станеться, якщо виводити рядок за допомогою функції `print (...)`.

Ураховуючи можливість отримати результат одразу після введення команди, вікно **Python** можна використовувати як багатофункціональний калькулятор:

```
>>> 12*45+30
570
```

Результат обчислення останньої інструкції зберігається в змінній '_' (одне підкреслення). Це дає змогу виконувати подальші розрахунки без уведення попереднього результату. Замість нього достатньо ввести символ підкреслення:

```
>>> 134*4
536
>>> _*6
3216
>>> _/4
804.0
```

Перевести курсор в одну з верхніх секцій інтерпретатора неможливо. Для повторення однієї з попередніх команд можна використовувати клавішу <↑> (або <↓>), натискаючи її потрібну кількість разів. У поточному командному рядку по черзі відображатимуться попередні (наступні) команди. Коли з'явиться потрібна команда, можна її відредагувати й натиснути клавішу **Enter**.

Для математичних операцій у виразах використовуються такі позначення:

- * – множення (34*89 – 34 помножити на 89).
- ** – піднесення до степеня (2**3 – 2 у третьому степені). Степінь може бути дробом:

```
>>>3**0.5
1.7320508075688772
```

Піднесення цілого числа до цілого степеня дає ціле число, якщо показник степеня ≥ 0 , і число з плаваючою крапкою, якщо < 0 .

- / – ділення (15/6 – 15 поділити на 6 з результатом 2.5). Ділення цілих чисел завжди дає результат з плаваючою крапкою, навіть якщо вони діляться без остачі.
- // – цілочислове ділення (15//6 поверне 2).
- % – операція обчислення остачі від ділення:

```
>>> 5%2
1
>>> 5.4%2.1 # остача від ділення
1.20000000000000002
```

- () – круглі дужки, що вказують порядок обчислень: (6+9)/4 – сума (6+9), поділена на 4. Не використовуйте квадратні або фігурні дужки для вказання порядку обчислень. Крім того, круглі дужки використовуються для вказання аргументів функцій, а також для створення кортежів. Кортежем є будь-яка послідовність елементів, розділених комами, узяті в круглі дужки.
- [] – квадратні дужки, що позначають список ([1,2,3,4] – список з чотирьох чисел). До елементів списку можна звертатися за індексом. Нумерація індексів починається з нуля:

```
>>> x=[10,20,30,40]
>>> x[2]
30
>>> x[0]
10
```

- Для запам'ятовування результатів обчислень зручно використовувати змінні. Для визначення змінної застосовується знак рівності =. Зліва від знака рівності стоїть ім'я змінної, а справа – її значення або вираз для обчислення. Одного значення можна надати відразу декільком змінним:

```
>>>x = y = z = 0
```

Ім'я змінної (ідентифікатор) не повинне збігатися з іменами ключових слів (операторів і функцій) **Python**. Ім'я має починатися з букви, може міс-

тити букви, цифри й символ підкреслення. Неприпустимо включати до складу імен змінних пропуски і спеціальні знаки. Регістр букв урахується при іменуванні змінних. При спробі використовувати змінну, якій не надано жодного значення, генерується помилка.

- Підкресленням визначається змінна, що є результатом останнього обчислення. Ця змінна використовується лише для читання, і їй не можна надати значення явно.
- `=` – перевірка на рівність (`a==5` повертає `True`, якщо `a` дорівнює 5).
- `!=` – перевірка на нерівність (`a!=5` повертає `True`, якщо `a` не дорівнює 5).
- `<`, `<=`, `>`, `>=` – позначення нерівності. Можна використовувати звичайні математичні порівняння, наприклад, `1 < x < 2`:

```
>>>x=3.4
>>>1<x<5
True
>>>1<x<3
False
```

Набрана формула зазвичай не закінчується жодним символом. Якщо надавалося значення, то результат операції відразу не відображується:

```
>>> z=1<x<8
>>> z
True
```

З логічними значеннями `True` і `False` можна виконувати логічні операції `and`, `or`, `not`:

```
>>> True and False
False
>>> not(True or False)
False
```

Логічні (булеві) значення `True` і `False` можна використовувати в арифметичних виразах, у яких вони інтерпретуються як 1 і 0:

```
>>> a=4
>>> b=True
>>> a+b
5
>>> (3+True)**0.5 # извлечение квадратного корня из 4
2.0
```

Припустимо, існує такий вираз:

```
>>> (3*(4>3)+(5!=6))**0.5 # добування квадратного кореня із
4
2.0
```

Проте пам'ятайте, що арифметичні операції мають пріоритет перед логічними. Порівняйте попередній приклад з таким:

```
>>> (3*(4>3)+5!=6)**0.5 # добування квадратного кореня із
1
1.0
```

У мові **Python** вбудованих функцій небагато. Більшість треба імпортувати (завантажувати зі спеціальних файлів – модулів). Елементарні математичні функції, такі, як `sin`, `cos`, `log` та ін., зібрано в модулі `math`. Щоб функції стали доступними, модуль слід імпортувати в робочий простір виконавчої системи командою `import ім'я_модуля`. Потім для виклику функції потрібно використовувати команду `ім'я_модуля.ім'я_функції(...)`:

```
>>> import math >>> math.exp(1.0)
2.718281828459045
```

Використовуючи функцію `dir (ім'я_модуля)`, можна визначити імена функцій, які є доступними з цього модуля:

```
>>> dir(math)

['__doc__', '__loader__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', '...', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Імена з двома підкресленнями є внутрішніми, усі інші – це функції, а також константи, такі, як `pi` і `e`:

```
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
>>> math.log(math.e)
1.0
```

Можна (і рекомендується) імпортувати лише ті функції, які збираєтеся використовувати. Для цього потрібно вказати їх імена в команді `from ім'я_модуля import ім'я_функції, ім'я_функції...`.

У цьому випадку перед ім'ям функції (або константи) ім'я модуля вказувати не потрібно, наприклад:

```
>>> from math import sin, pi
>>> pi
3.141592653589793
>>> sin(pi/2)
>>> 1.0
```

Для імпорту всіх функцій модуля можна використовувати команду

```
>>>from math import *
```

При імпорті функції з модуля їй можна надати іншого ім'я. У цьому випадку одночасно з додаванням функції створюється її нове ім'я, наприклад:

```
>>> from math import factorial as f
>>> f(5)
120
```

2.4. Зберігання програми в окремому файлі

В інтерактивному режимі не можна вносити змін до виразу, який раніше було виконано. Доводиться повторно набирати вираз і його запускати.

У разі великих програм зручно використовувати окремі файли з розширенням `.py`.

У меню **IDLE** виберемо **File** → **New File**. З'явиться вікно текстового редактора, у якому можна набирати команди на мові **Python**.

Наберемо такий код:

```
a=5
print(a)
print(a+5)
```

У меню редактора виберемо **Save As** і збережемо файл у вільну директорію, указавши ім'я `myprog1.py`. Щоб виконати програму в меню редактора, виберемо **Run** → **Run Module** (або натиснемо <F5>). Результат роботи програми відображатиметься в інтерактивному режимі (рис. 2.7).

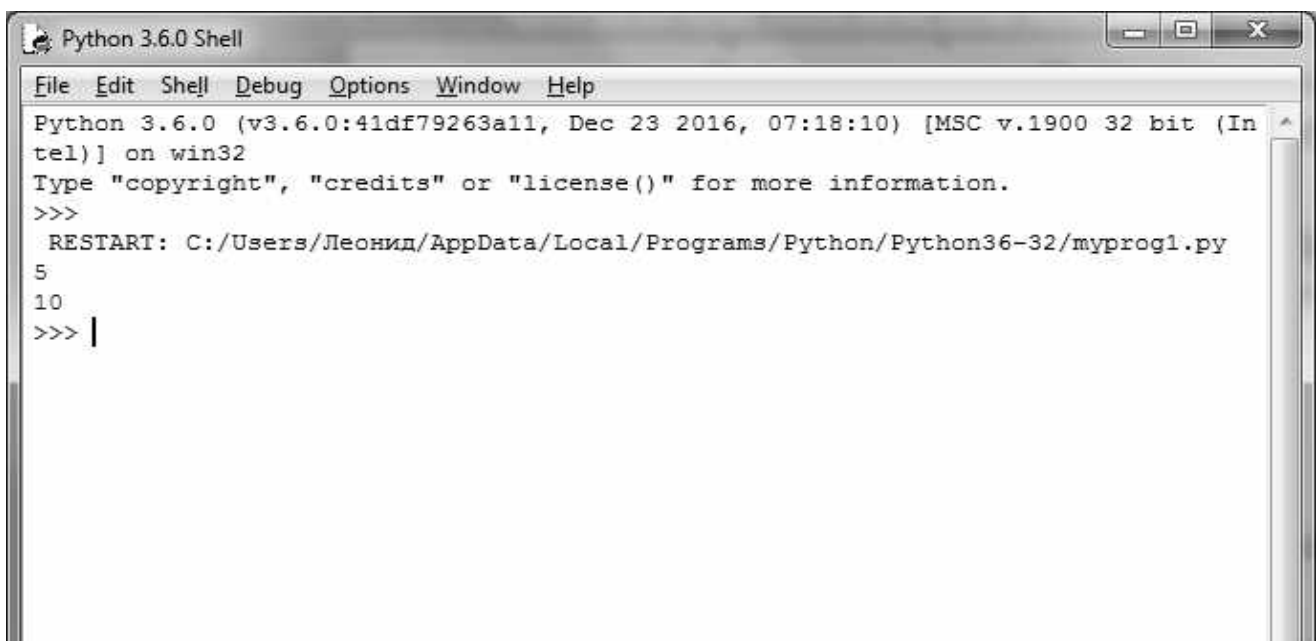


Рис. 2.7. Виведення результатів розрахунку в інтерактивному режимі

2.5. Змінні в мові Python

Розглянемо вираз $y=x+3*6$, де y і x є змінні, які можуть містити значення числового типу. На мові **Python** обчислити значення при x , що дорівнює 1, можна таким чином:

```
>>> x = 1
>>> y = x + 3*6
>>> y
19
>>>
```

У виразі не можна використовувати змінну, якщо раніше їй не було надано значення – для **Python** такі змінні є невизначеними.

Значення змінної y можна побачити, якщо в інтерактивному режимі набрати її ім'я.

Імена змінних створює програміст, але є декілька обмежень, пов'язаних з найменуванням. Як імена змінних не можна використовувати ключові слова, які для мови **Python** мають певний сенс (ці слова підсвічуються в **IDLE** оранжевим кольором) (табл. 2.2).

Таблиця 2.2

№ п/п	Імена змінних, створюваних програмістом, які заборонено до використання					
1	and	as	assert	break	class	continue
2	def	del	elif	else	except	exec
3	finally	for	from	global	if	import
4	in	is	lambda	nonlocal	not	or
5	pass	raise	return	try	while	with
6	yield	True	False	None		

2.6. Модулі

Викладемо основи модульної системи **Python** та імпорту компонентів. Імпорт модулів – це основа програмної архітектури в **Python**. Модулі виконують як мінімум три важливі функції:

- Повторне використання коду. Такий код може бути завантажений багато разів у багатьох місцях.

- Керування адресним простором. Модуль – це високорівнева організація програм, це пакет імен, який звільняє вас від конфліктів. Кожний об'єкт «проживає» свій цикл усередині свого модуля, тому модуль – це засіб для групування системних компонентів.
- Глобалізація сервісів і даних. Для реалізації об'єкта, який використовується в багатьох місцях, достатньо написати один модуль, який буде імпортований.

Програма **Python** дає змогу помістити класи, функції або дані в окремий файл і використовувати їх в інших програмах. Такий файл називають модулем. Об'єкти з модуля можуть бути імпортовані в інші модулі. Ім'я файла створюється шляхом додавання до імені модуля розширення `.py`. При імпорті модуля інтерпретатор шукає файл з ім'ям `my_module.py` спочатку в поточному каталозі, потім в каталогах, указаних у змінній оточення `PYTHONPATH`, потім в залежних від платформи шляхах за умовчанням, а також у спеціальних файлах з розширенням `'.pth'`, які містяться в стандартних каталогах. Програміст може внести зміни до `PYTHONPATH` і в `'.pth'`, додавши туди свій шлях. Каталоги, у яких здійснюється пошук, можна побачити в змінній `sys.path`.

Великі програми зазвичай складаються зі стартового файла – файла верхнього рівня, і набору файлів-модулів. Головний файл контролює програми. Водночас модуль – це не лише фізичний файл, колекція компонентів. У цьому сенсі модуль – це простір імен, `namespace`, і всі імена всередині модуля ще називають атрибутами, такими, наприклад, як функції і змінні.

2.7. Підключення модулів і бібліотек

2.7.1. Підключення модулів і бібліотек Python для Windows

Для встановлення модулів і бібліотек при використанні операційної системи **Windows** насамперед необхідно перевірити розрядність системи – **x64** чи **x86**. Для цього потрібно пройти по шляху **Панель керування** → **Система і безпека** → **Система** і в цьому вікні визначити тип системи (рис. 2.8). У цьому ж вікні клацніть лівою кнопкою мишки на закладці «Додаткові параметри системи» (рис. 2.9). Відкриється нове вікно «Властивості системи» (рис. 2.10, а), у якому у вкладці «Додатково» виберіть вкладку «Змінні середовища» та активуйте її лівою кнопкою мишки. У вікні «Змінні середовища» (рис. 2.10, б) у розділі «**Системні змінні**» впишіть у змінному середовищі «**Path**» шлях до системної теки, у якій знаходиться тека **Script** програми **Python**, наприклад:

```
«C:\Users\User\AppData\Local\Programs\Python\Python36-32\Scripts».
```

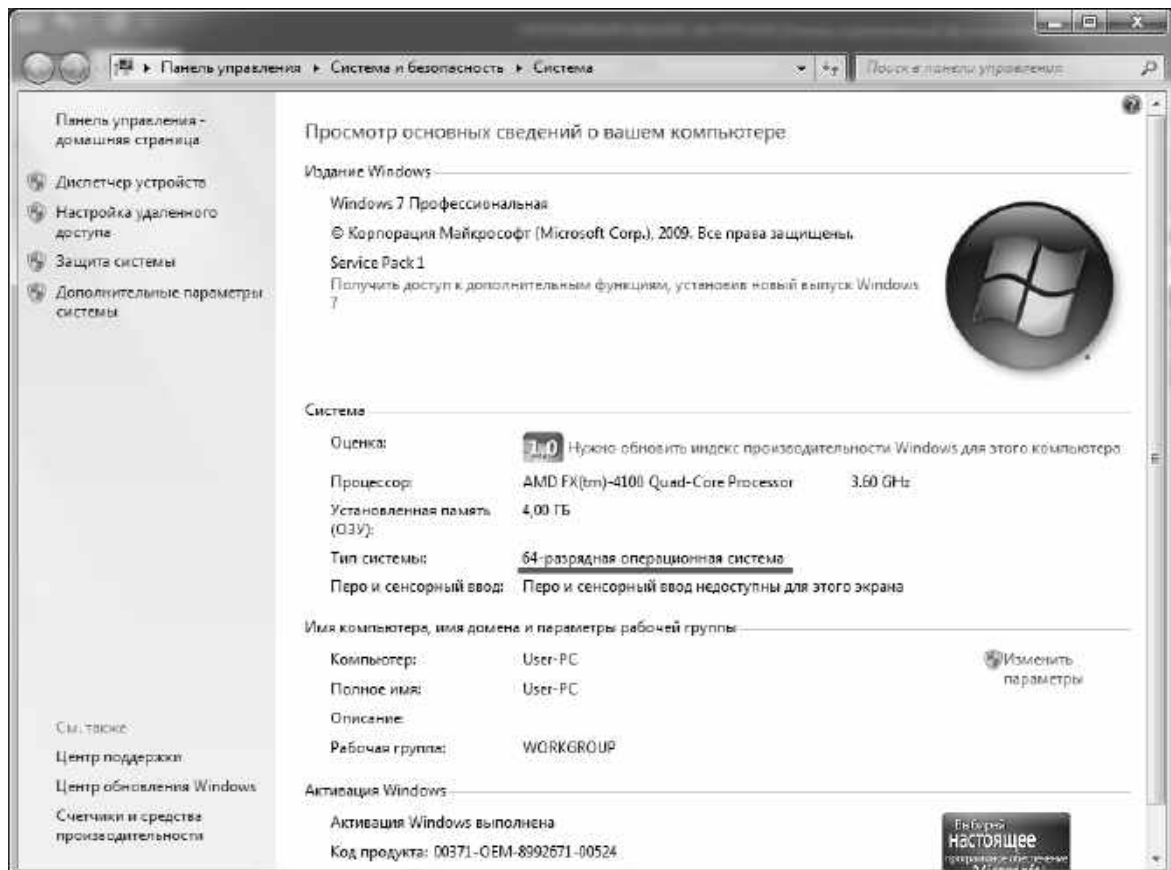


Рис. 2.8. Проверка розрядности операционной системы

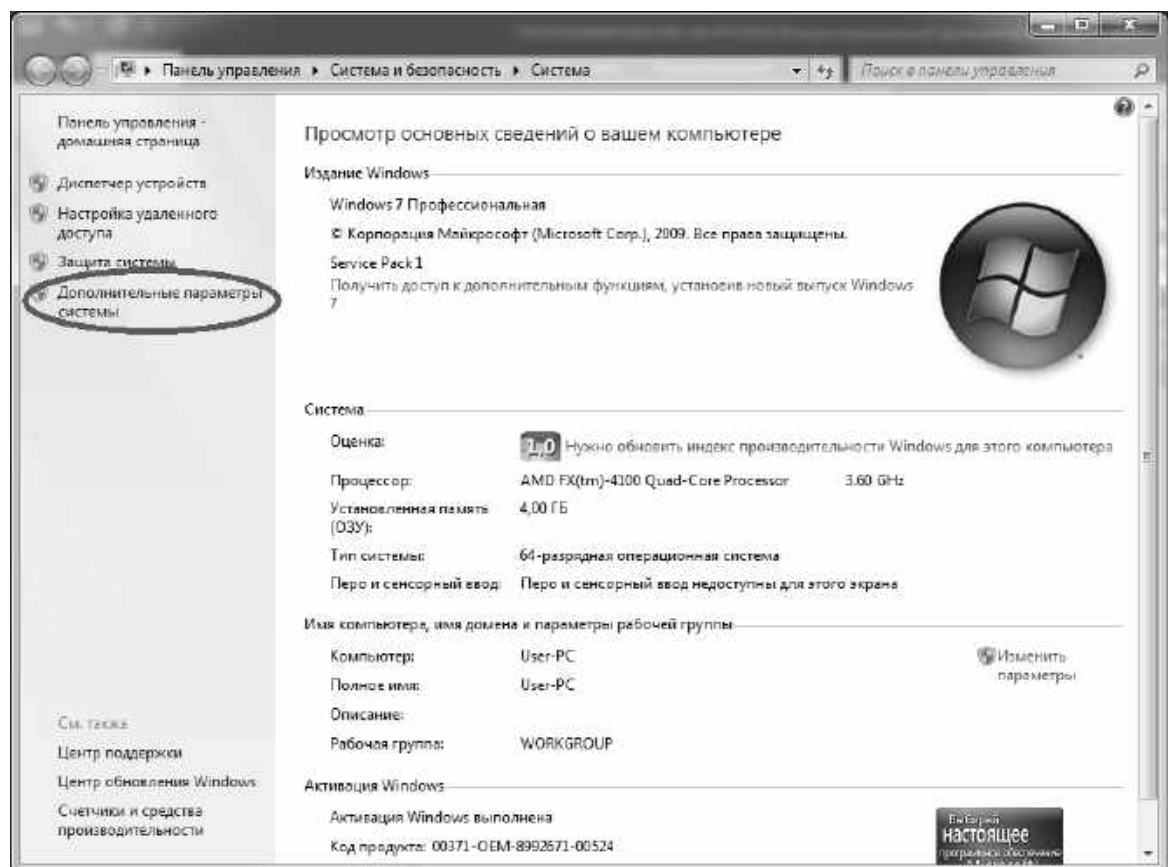
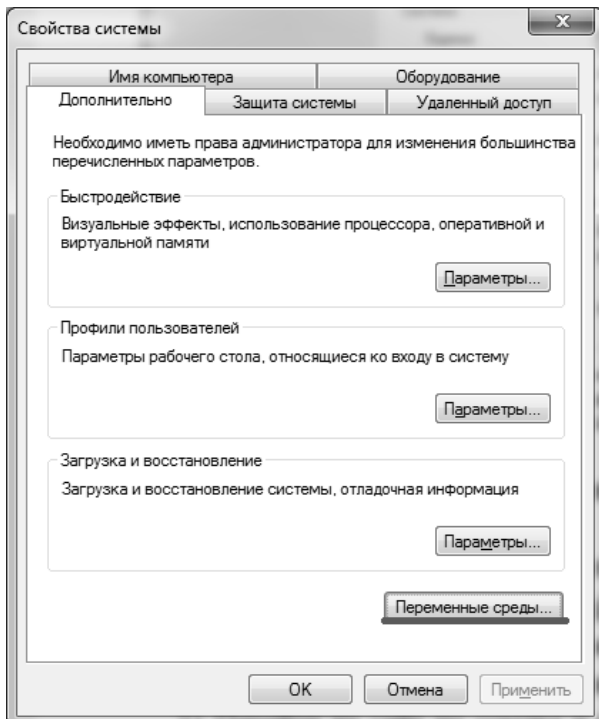
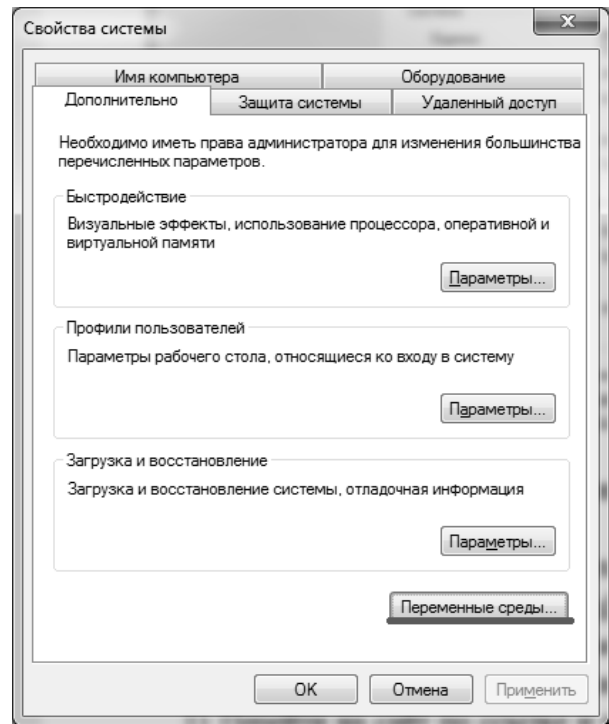


Рис. 2.9. Выбор дополнительных параметров системы



а



б

Рис. 2.10. Установлення системних змінних

Після цієї перевірки й установлень потрібно скачати бібліотеки згідно з розрядністю системи.

Бібліотеки для **Python 3.6** знаходяться у вільному доступі на сайті <http://www.lfd.uci.edu/~gohlke/pythonlibs/>. Потрібну бібліотеку необхідно скачати з цього сайту.

Для прикладу пройдемо весь шлях установлення бібліотеки «**Pillow**». Для цього необхідно перейти на сайт за посиланням і знайти потрібну бібліотеку. Для зручності можна використовувати комбінацію клавіш **Ctrl+F** і у вікні (рис. 2.11), що з'явилася, уписати назву бібліотеки.

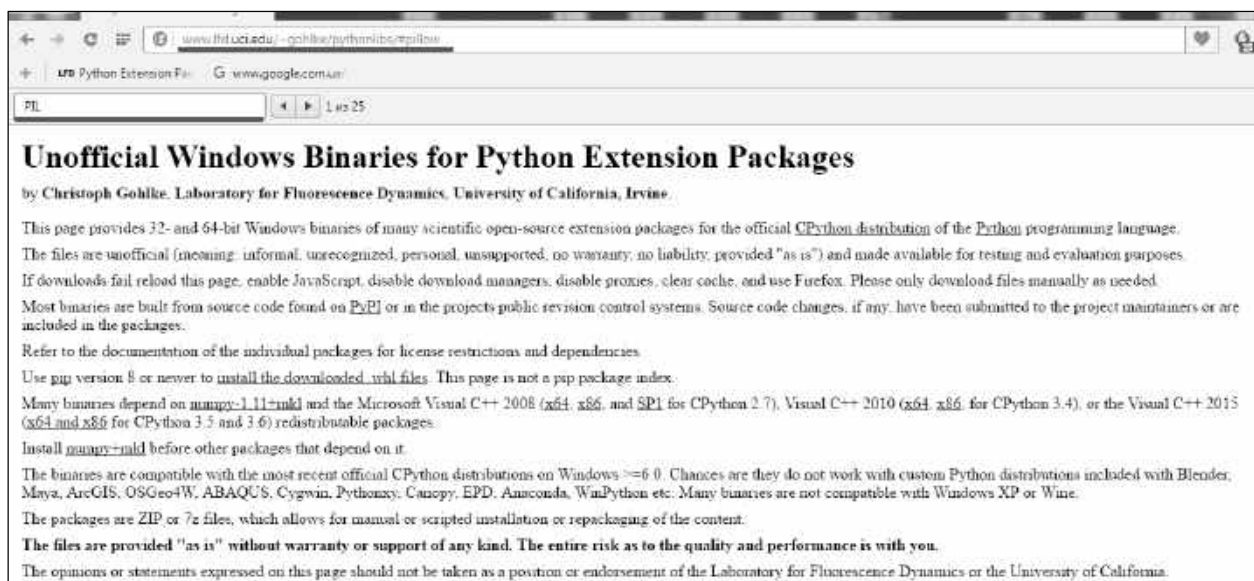


Рис. 2.11. Пошук потрібної бібліотеки на сайті

Список потрібних бібліотек розташовується в нижній частині головної сторінки сайту (рис. 2.12).

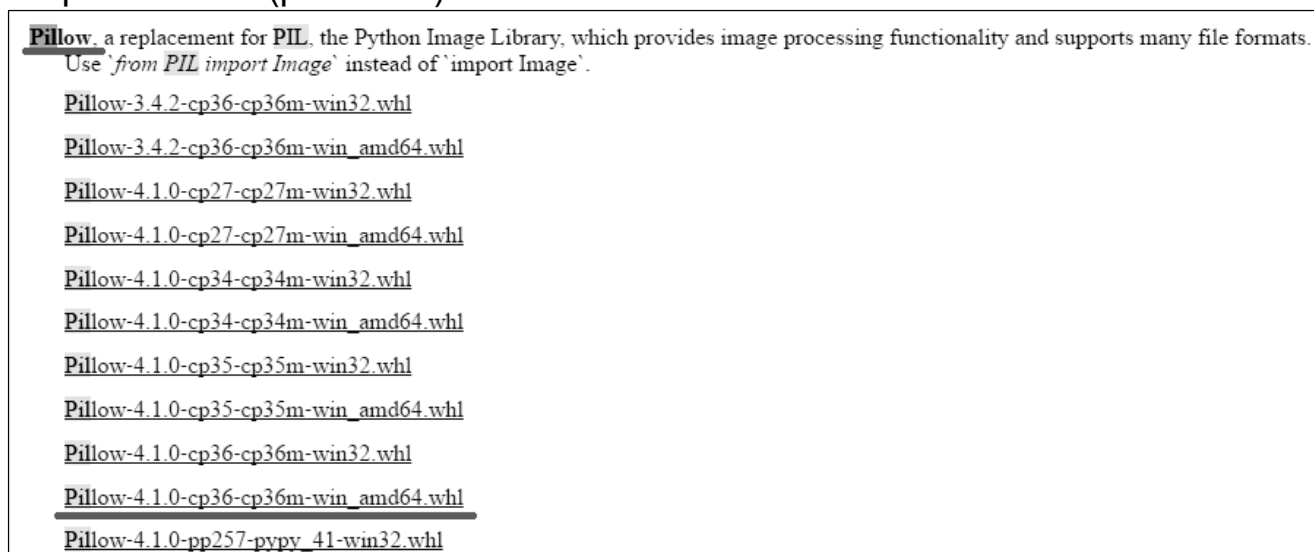


Рис. 2.12. Вибір потрібної бібліотеки

Скачаємо вибрану бібліотеку, урахувавши розрядність операційної системи і раніше встановлену версію **Python**. Далі необхідно зайти в системну теку «**Python**», куди раніше встановлювалася програма, наприклад, указавши такий шлях: «C:\users\user\appdata\Local\Programs\python\python36-32». У командному рядку введемо команду «cmd» і натиснемо кнопку **Enter** (рис. 2.13).

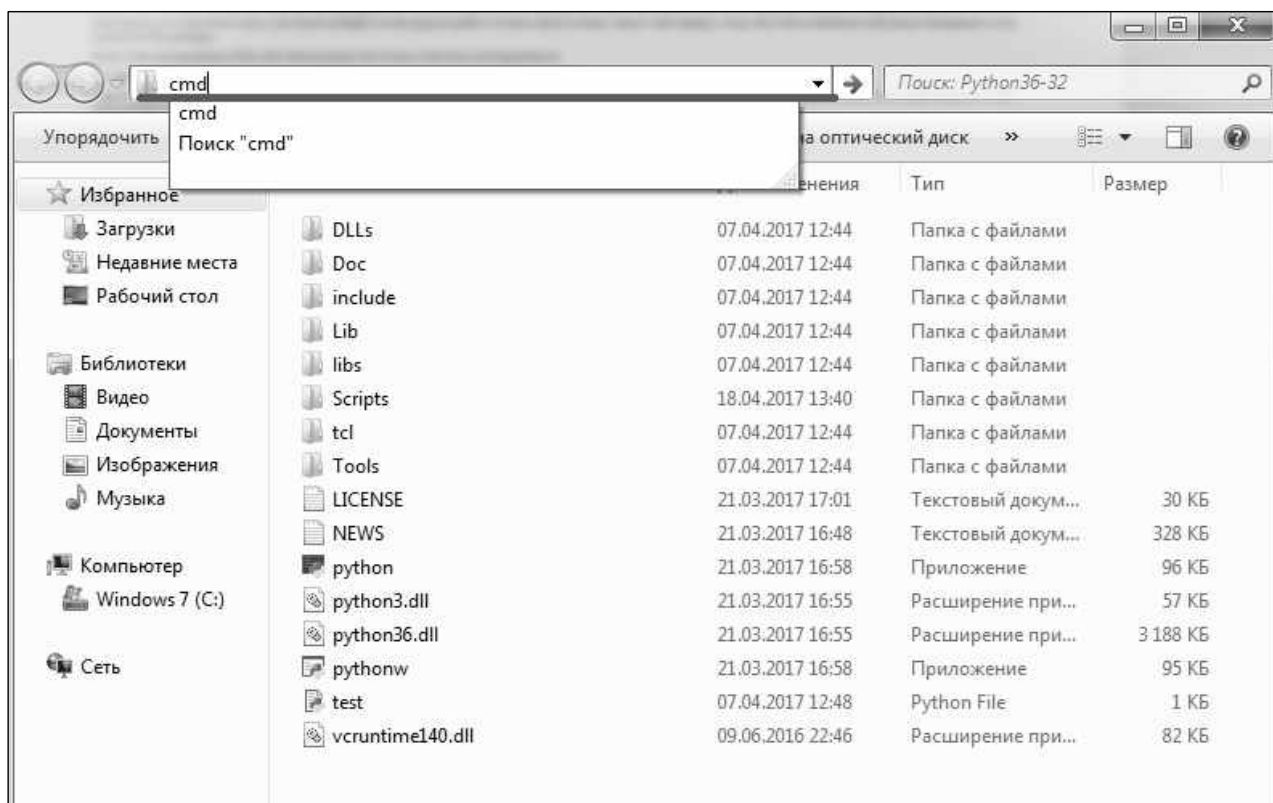


Рис. 2.13. Розділи бібліотеки

У вікні адміністратора (рис. 2.14) за допомогою вбудованого пакета **PIP** додамо раніше скачану бібліотеку **pillow** командою «**install**» і підтвердимо натисненням кнопки **Enter**.

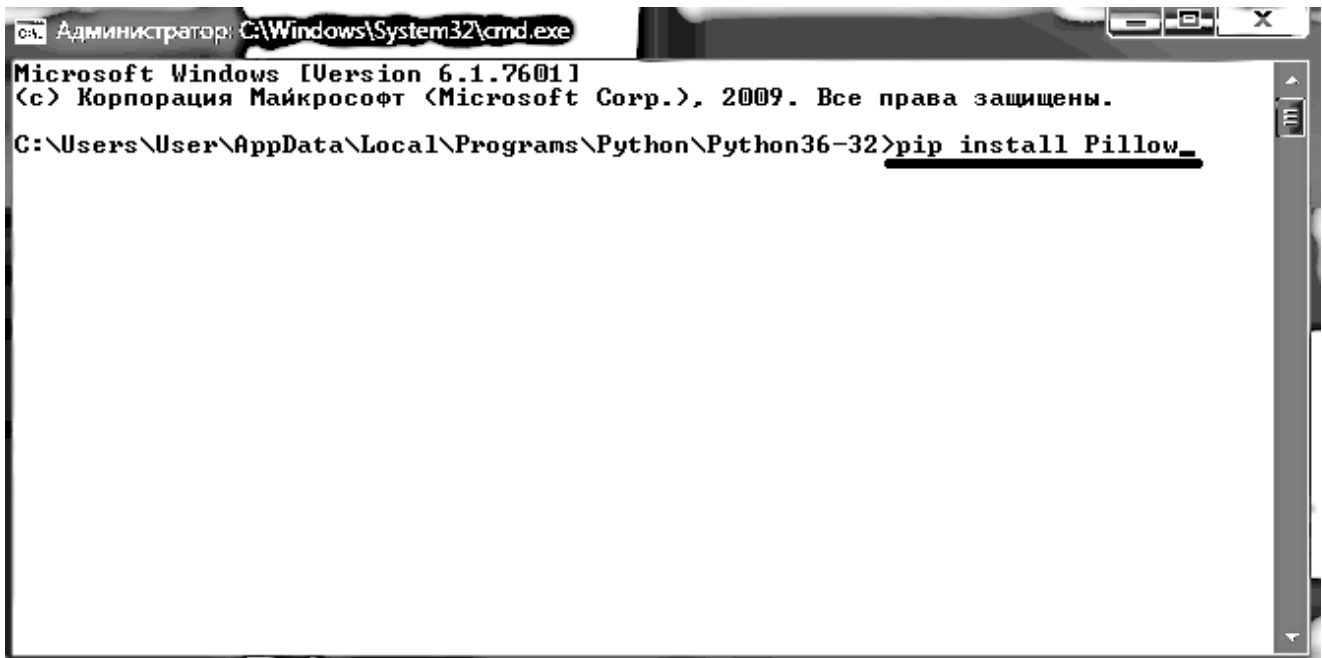


Рис. 2.14. Інсталяція бібліотеки

Після цих дій бібліотека встановиться, і її можна буде використовувати в своїй роботі. Інші бібліотеки, наприклад бібліотеку **Matplotlib**, установлюють аналогічним чином.

2.7.2. Підключення модулів і бібліотек Python для платформи Raspberry Pi

За допомогою команди «**Sudo apt-get install**» можна встановити пакети, необхідні для роботи в середовищі програмування **Python**, – **numpy**, **pip**, **pi-camera** та ін.

1. Установимо **pip** для інтерпретатора **Python 3.6**. Слід взяти до уваги, що за допомогою команди «**Sudo apt-get install python3-pip**» встановлюється **PYPI (Python Package Index** – каталог пакетів **Python**). «**pip**» – це система керування пакетами, яка використовується для встановлення програмних пакетів, написаних на мові **Python**, і керування ними.
2. Слід звернути увагу на версію **Python** для подальшого встановлення бібліотек. Для **Python** версії 2 використовується команда «**python-pip**», а для **Python** версії 3 – команда «**python3.2-pip**».
3. Команда **pip-3.2** встановлює модулі для **Python 3**, а **pip** – для **Python 2**.
4. Основні команди «**pip**»:
 - **pip help** – допомога щодо доступних команд;
 - **pip install package_name** – встановлення пакета(ів);

- `pip uninstall package_name` – видалення пакета(ів);
 - «`pip list`» – список установлених пакетів;
 - `pip show package_name` – інформація про встановлений пакет;
 - `pip search` – пошук пакетів за ім'ям;
 - `pip - proxy user:passwd@proxy.server:port` – використання з проксі;
 - `pip install -U` – оновлення пакета(ов);
 - `pip install --force-reinstall` – при оновленні переустановлення пакета, навіть якщо він останньої версії.
5. Команда «***Pip-3.2 install numpy***» установлює бібліотеку ***Numpy*** для ***Python 3***, як показано на рис. 2.15.

```

pi@raspberrypi: ~
Файл Редагувати Вкладки Допомога
pi@raspberrypi:~ $ sudo apt-get install python3-pip
Зчитування переліків пакунків... Виконано
Побудова дерева залежностей
Зчитування інформації про стан... Виконано
Вже встановлена найновіша версія python3-pip.
оновлено 0, встановлено 0 нових, 0 відмічено для видалення і 0 не оновлено.
pi@raspberrypi:~ $ pip install numpy
Requirement already satisfied (use --upgrade to upgrade): numpy in /usr/lib/python2.7/dist-packages
Cleaning up...
pi@raspberrypi:~ $ sudo apt-get install python3-picamera
Зчитування переліків пакунків... Виконано
Побудова дерева залежностей
Зчитування інформації про стан... Виконано
Вже встановлена найновіша версія python3-picamera.
оновлено 0, встановлено 0 нових, 0 відмічено для видалення і 0 не оновлено.
pi@raspberrypi:~ $ █

```

Рис. 2.15. Вікно встановлення бібліотек

2.7.3. Імпорт модуля

Перший варіант імпорту. Якщо запустити в каталозі, у якому знаходиться певний модуль (наприклад, `my_module.py`), інтерпретатор

```
>>> python
```

і потім зробити імпорт модуля

```
>>> import my module
```

то дістанемо доступ до всіх функцій, визначених у модулі:

```
>>> my_module.func1()
>>> my_module.func2()
. . .
```

Для коротшого запису можна створити локальну змінну:

```
>>> f1 = module.func1
```

Другий варіант імпорту – узяття безпосередньо імені без імені модуля:

```
>>> from my_module import func1, func2
>>> func()
```

Третій варіант імпорту – підключення всіх імен, визначених в модулі:

```
>>> from my_module import *
>>> func1()
```

Для запобігання конфлікту імен можна створити аліас:

```
>>> from my_module import open as my_open
```

Щоб уникнути непорозумінь, оператор `import` застосовують переважно без `from` у тих випадках, коли один і той же модуль використовується в декількох місцях.

Оскільки модуль завантажується один раз, для його повторного завантаження можна використовувати функцію `reload()`.

Кожний модуль має власний простір імен, що є глобальною зоною видимості для всіх функцій, визначених у ньому. Для того щоб змінні цього модуля не входили в конфлікт з іншими глобальними іменами або іншими модулями, необхідно використовувати префікс: `_ім'я_модуля_`. `_ім'я_змінної_`.

Модулі можуть імпортувати інші модулі. Зазвичай інструкцію `import` розташовують на початку модуля або програми.

2.7.4. Стандартні модулі

Програма **Python** розповсюджується разом з бібліотекою стандартних модулів. Бібліотека містить більше 200 модулів, які виконують платформено-залежну підтримку таких завдань: інтерфейс до операційної системи, керування об'єктами, пошук, мережа + інтернет, **GUI** і т. д. Повний список стандартних модулів можна побачити на сайті <http://docs.python.org/library/>.

Частину модулів вбудовано в інтерпретатор за умовчанням із забезпеченням доступу до операцій. Їх вбудовано або з міркувань ефективності, або для забезпечення доступу до примітивів операційної системи (наприклад, модуль `sys`).

Змінна `sys.path`, що містить список рядків з іменами каталогів, у яких відбувається пошук модулів, ініціалізувалася зі значення змінної оточення `PYTHONPATH` і вбудованого значення за умовчанням. Можна додати шлях:

```
>>> import sys
>>> sys.path.append(/home/my/lib/python)
```

Для з'ясування імен, визначених у модулі, можна використовувати вбудовану функцію `dir()`, яка повертає відсортований список рядків:

```
>>> dir(sys)
['_displayhook_', '_doc_', 'egginsert_', '_excepthook_', '_
name_',
. . .
'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version
info']
```

2.7.5. Перегляд установлених модулів

Бібліотека ***Python Standard Library*** містить безліч модулів, які забезпечують доступ до функціональних можливостей системи або надають стандартні рішення. ***Python Standard Library*** є частиною кожного встановлення ***Python***.

Щоб переконатися в тому, що ці модулі є доступними, відкрийте середовище розробки ***Python 3*** і запустіть інтерпретатор ***Python*** у командному рядку:

```
Python
```

Тепер можна запустити оператор `import`, щоб переконатися, що необхідний модуль готовий до роботи:

```
>>> import math
```

Оскільки модуль `math` є вбудованим, інтерпретатор не поверне жодного виводу. Це означає, що не потрібно нічого робити, щоб запустити цей модуль.

Запустіть `import` і вкажіть ім'я модуля, який не поставляється за умовчанням, наприклад `matplotlib`:

```
>>> import matplotlib
```

Якщо модуль не встановлено, то команда видасть помилку:

```
ImportError: No module named 'matplotlib'
```

Тепер потрібно відключити інтерпретатор ***Python*** (клавіші ***CTRL + D***) і встановити `matplotlib` за допомогою `pip`:

```
pip install matplotlib
```

2.7.6. Імпорт функцій модулів

Щоб одержати доступ до функцій модуля, його потрібно імпортувати за допомогою оператора `import`.

Вираз для оператора `import` складається з ключового слова `import` та імені модуля. У файлі **Python** цей вираз розташовується у верхній частині коду.

Наприклад, щоб імпортувати в програму `my_rand_int.py` модуль `random`, який генерує випадкові числа, необхідно відкрити файл і вставити команду

```
>> import random
```

Модуль, що імпортується, стає доступним у поточній програмі як окремий простір імен. Це означає, що посилатися на функцію потрібно в точковій нотації (`[module].[function]`). Для прикладу розглянемо на практиці модуль `random`, який працюватиме як функція в таких варіантах:

- `random.randint()` – викликає функцію, яка повертає випадкове число;
- `random.randrange()` – викликає функцію, яка повертає випадковий елемент із заданої послідовності.

2.8. Використання модуля `math`

Модуль ***math*** – один із найважливіших в **Python**. Цей модуль надає великий набір функцій для роботи з числами і дає змогу виконати такі обчислення:

- `math.ceil(X)` – округлення до найближчого більшого числа;
- `math.copysign(X, Y)` – повернення числа, що має такий же модуль, що й число `X`, а знак – як у числа `Y`;
- `math.fabs(X)` – визначення модуля `X`;
- `math.factorial(X)` – обчислення факторіала числа `X`;
- `math.floor(X)` – округлення вниз;
- `math.fmod(X, Y)` – остача від ділення `X` на `Y`;
- `math.frexp(X)` – повернення мантиси й експоненти числа;
- `math.ldexp(X, I)` – $X * 2^I$. Функція, обернена до функції `math.frexp()`;
- `math.fsum` (послідовність) – складання всіх членів послідовності; еквівалент вбудованої функції `sum()`, але функція `math.fsum()` є точнішою для чисел з плаваючою крапкою;
- `math.isfinite(X)` – визначення, чи є `X` числом;
- `math.isinf(X)` – визначення, чи є `X` нескінченністю;

- **math.isnan**(X) – визначення, чи є X NAN (Not a Number – не число);
- **math.modf**(X) – повертання дробу й цілої частини числа X; обидва числа мають той же знак, що й X;
- **math.trunc**(X) – усічення значення X до цілого;
- **math.exp**(X) – розрахунок e^X ;
- **math.expm1**(X) – $e^X - 1$. При $X \rightarrow 0$ точніше, ніж **math.exp**(X) - 1;
- **math.log**(X, [base]) – логарифм X за основою base. Якщо base не вказано, обчислюється натуральний логарифм;
- **math.log1p**(X) – натуральний логарифм $(1 + X)$. При $X \rightarrow 0$ точніше, ніж **math.log**(1+X);
- **math.log10**(X) – логарифм X за основою 10;
- **math.log2**(X) – логарифм X за основою 2;
- **math.pow**(X, Y) – X^Y ;
- **math.sqrt**(X) – квадратний корінь з X;
- **math.acos**(X) – арккосинус X. У радіанах;
- **math.asin**(X) – арксинус X. У радіанах;
- **math.atan**(X) – арктангенс X. У радіанах;
- **math.atan2**(Y, X) – арктангенс Y/X. У радіанах. З урахуванням чверті, у якій знаходиться точка (X, Y);
- **math.cos**(X) – косинус X (X вказується в радіанах);
- **math.sin**(X) – синус X (X вказується в радіанах);
- **math.tan**(X) – тангенс X (X вказується в радіанах);
- **math.hypot**(X, Y) – обчислення гіпотенузи трикутника з катетами X і Y (**math.sqrt**(x * x + y * y));
- **math.degrees**(X) – конвертування радіанів у градуси;
- **math.radians**(X) – конвертування градусів у радіани;
- **math.cosh**(X) – гіперболічний косинус;
- **math.sinh**(X) – гіперболічний синус;
- **math.tanh**(X) – гіперболічний тангенс;
- **math.acosh**(X) – обернений гіперболічний косинус;

- `math.asinh(X)` – обернений гіперболічний синус;
- `math.atanh(X)` – обернений гіперболічний тангенс;
- `math.erf(X)` – функція помилок;
- `math.erfc(X)` – додаткова функція помилок
(`1 - math.erf(X)`);
- `math.gamma(X)` – гамма-функція X ;
- `math.lgamma(X)` – натуральний логарифм гамма-функції X ;
- `math.pi` – $\pi = 3,1415926\dots$
- `math.e` – $e = 2,718281\dots$

Наведемо декілька простих прикладів обчислення (програмний код з використанням модуля ***math***):

```
>>> import math
>>> # Обчислення десяткового логарифма
>>> x=100
>>> y=math.log10(X)
>>> # Обчислення косинуса
>>> x=math.pi/2
>>> y=math.cos(x)
>>> print(y)
6.123233995736766e-17
>>> # Добування квадратного кореня
>>> x=64
>>> y=math.sqrt(x)
>>> print (y)
8.0
>>> # Піднесення числа X до степеня числа Y
>>> x=5
>>> y=3
>>> z=math.pow(x,y)
>>> print (z)
125.0
```

2.9. Формування масивів і матриць

Для формування великих багатовимірних масивів і матриць використовується бібліотека `numpy` разом з бібліотеками високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами. Особливості підключення бібліотек детально описано вище, і підключення цієї бібліотеки великих труднощів не викликає.

Основним об'єктом `numpy` є однорідний багатовимірний масив (у `numpy` він має назву `numpy.ndarray`). Це багатовимірний масив елементів (зазвичай чисел) одного типу.

Наведемо найбільш важливі атрибути об'єктів `ndarray`:

- **`ndarray.ndim`** – кількість вимірів (частіше їх називають "осі") масиву.
- **`ndarray.shape`** – розміри масиву, його форма. Це кортеж натуральних чисел, що вказує довжину масиву по кожній осі. Для матриці з n рядків і m стовпів `shape` буде (n, m) . Кількість елементів кортежу `shape` дорівнює `ndim`.
- **`ndarray.size`** – кількість елементів масиву, яка дорівнює добутку всіх елементів атрибута `shape`.
- **`ndarray.dtype`** – об'єкт, що описує типи елементів масиву. Можна визначити `dtype`, використовуючи стандартні типи даних **Python**. **`Numpy`** тут дає безліч можливостей, як вбудованих, наприклад: `bool_, character, int8, int16, int32, int64, float8, float16, float32, float64, complex64`, `object_`, так і можливість визначити власні типи даних, у тому числі й складені.
- **`ndarray.itemsize`** – розмір кожного елемента масиву в байтах.
- **`ndarray.data`** – буфер, що містить фактичні елементи масиву. Зазвичай не потрібно використовувати цей атрибут, оскільки звертатися до елементів масиву найпростіше за допомогою індексів.

Створення масивів. У модулі `numpy` існує багато способів створити масив. Один з найбільш простих способів – створити масив зі звичайних списків або кортежів **Python**, використовуючи функцію `numpy.array()` (`array` – функція, що створює об'єкт типу `ndarray`):

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> type(a)
<class 'numpy.ndarray'>
```

Функція `array()` трансформує послідовності, вкладені в багатовимірні масиви. Тип елементів масиву залежить від типу елементів вихідної послідовності.

```
>>> import numpy as np
>>> b = np.array([[1.5, 2, 3], [4, 5, 6]])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

Функція `array()` – не єдина функція для створення масивів. Зазвичай елементи масиву спочатку є невідомими, а масив, у якому вони зберігатимуться, уже потрібен. Тому є декілька функцій для того, щоб створювати масиви з якимось вихідним умістом (за умовчанням тип створюваного масиву – `float64`).

Функція `zeros()` створює масив з нулів, а функція `ones()` – масив з одиниць. Обидві функції приймають кортеж з розмірами і аргумент `dtype`:

```
>>> import numpy as np
>>> np.zeros((3, 5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> np.ones((2, 2, 2))
array([[[ 1.,  1.],
        [ 1.,  1.]],
       [[ 1.,  1.],
        [ 1.,  1.]])])
```

Функція `eye()` створює одиничну діагональну матрицю (двовимірний масив):

```
>>> import numpy as np
>>> np.eye(5)
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

Функція `print(...)` з ядра **Python** друкує масив у вигляді списку, наприклад:

```
>>> a=np.array([0.0,2.5,5.2])
>>> print(a)
[0. 2.5 5.2]
```

Перебирати елементи масиву можна в циклі, наприклад, так:

```
>>> for i in a:
        print(i)
0.0
2.5
5.2
```

Наведені приклади формування масивів і матриць не вичерпують усього різноманіття використовуваних для цього способів. Читачеві пропонується ознайомитися з ними самостійно.

2.10. Матричні обчислення

Операції між масивом і скаляром. Операції $+$, $-$, $*$, $/$ між масивом і числом означають збільшення (віднімання) числа до кожного елемента масиву, множення (ділення) на число кожного елемента масиву. Наприклад, збільшення числа до елементів масиву і множення елемента масиву на якесь число:

```
>>> a=np.array([1.0,2.0,3.0])
>>> a+10
array([11., 12., 13.])
>>> print(a*5)
[ 5. 10. 15.]
```

Можна піднести до «числового» степеня кожний елемент масиву з попереднього прикладу:

```
>>> a**2
array([ 1.,  4.,  9.])
```

Операції між двома масивами. З масивами можна виконувати поелементні арифметичні операції. Розмірність масивів має бути однаковою або один з масивів має бути одноелементним. У другому випадку операція виконується, як між масивом і скаляром. Якщо операнди мають різного типу елементи, то результат зводиться до «старшого» типу.

Наприклад, можна скласти поелементно два масиви:

```
>>> a=np.array([1.0,2.0,3.0])
>>> b=np.array([10,20,30])
>>> print(a+b)
[11. 22. 33.]
```

Поелементне множення двох масивів не є скалярним (або матричним) добутком:

```
>>> a=np.array([1.0,2.0,3.0])
>>> b=np.array([10,20,30])
>>> print(a*b)
[10. 40. 90.]
```

Можна поділити поелементно два масиви:

```
>>> a=np.array([1.0,2.0,3.0])
>>> b=np.array([10,20,30])
>>> print(a/b)
[ 0.1  0.1  0.1]
```

Припустимо, що є складання масивів різної розмірності. Кожний елемент першого масиву складається з відповідними елементами внутрішніх масивів другого доданка. Це відбувається так само, як і складання скаляра з масивом:

```
>>> a=np.array([1,2,3])
>>> b=np.array([ [1,2,3] , [4,5,6]])
>>> a+b
array([[ 2.,  4.,  6.],
       [ 5.,  7.,  9.]])
```

Векторні й матричні операції над масивами. Одновимірні і двовимірні масиви інтерпретуються як вектори й матриці, якщо вони використовуються як аргументи функцій, які обчислюють скалярний або векторний добуток. Наведемо приклад їх перетворень з використанням класів `array` і `matrix` (зазначимо, що для перемноження матриць за правилами лінійної алгебри використовується метод `dot`):

```
>>> a = numpy.array ([[1, 2, 3], [4, 5, 6]], "f")
>>> print a
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
>>> b = numpy.array ([[0, 1, 2], [4, -1, 1]], "f")
>>> print b
[[ 0.  1.  2.]
 [ 4. -1.  1.]]
# Поелементне перемноження
>>> c1 = a * b
>>> print c1
[[ 0.  2.  6.]
 [ 16. -5.  6.]]
# Транспонування матриці
>>> bt = b.T
>>> print bt
[[ 0.  4.]
 [ 1. -1.]
 [ 2.  1.]]
# Перемноження матриць
>>> c2 = numpy.dot(a, bt)
>>> print c2
[[ 8.  5.]
 [ 17. 17.]]
```

2.11. Особливості використання операторів циклу в мові Python

Стисло викладемо основні особливості використання операторів циклу в мові **Python**. Це допоможе раціонально організувати багато процедур при написанні коду програм. Цикл `for` у **Python** є потужним інструментом, що дає змогу послідовно виконувати різні операції на наборах значень: арифметичні прогресії, значення масивів, букви в рядку.

Розглянемо для прикладу роботу оператора циклу `for` з арифметичною прогресією.

Нехай `range(N)` – послідовність чисел від 0 до $(N - 1)$ включно, зручна для обходу індексів масиву. Якщо $N \leq 0$, то послідовність буде порожньою, а `for` не виконається жодного разу. Наведемо приклад коду:

```
for i in range(5):  
    print(i)
```

Цей код при виконанні виведе числа від 0 до 4 включно. Зверніть увагу на те, що цикл `for` сам визначає змінну циклу, окремо це робити не потрібно:

```
0  
1  
2  
3  
4
```

Тепер розглянемо `range(a, b)` – послідовність чисел від a до $(b - 1)$ включно. Якщо $b \leq a$, то послідовність буде порожньою, а `for` не виконається жодного разу:

```
for i in range(14, 17):  
    print(i)
```

У цьому випадку при виконанні коду виведеться послідовність

```
14  
15  
16
```

Нехай `range(a, b, step)` є послідовністю чисел від a до b з кроком `step`. Число b у послідовність не включається. Якщо $a == b$ або знаки кроку `step` і вирази $(b - a)$ не збігаються, то послідовність буде порожньою, тіло `for` не виконається:

```
for i in range(1, 9, 2):  
    print(i)
```

При виконанні циклу виведеться послідовність

```
1  
3  
5  
7
```

Наведемо ще один приклад:

```
for i in range(3, -4, -2):  
    print(i)  
3  
1  
-1  
-3
```

Після завершення циклу `for` у змінній циклу зберігається значення, яке було в ній на останній ітерації. Для `range(10)` це буде число 9.

Зверніть увагу на те, що нове значення змінної циклу кожного разу береться з `range`, тобто навіть якщо змінити змінну циклу, на наступній ітерації вона все одно матиме наступне значення з послідовності, неначе вона не змінювалася зовсім.

Оператор break перериває цикл `for` або `while`, наприклад:

```
for i in range(1, 20):  
    print('i =', i)  
    if i % 6 == 0:  
        break  
print('Final i =', i) # виводить 6
```

Такий цикл, як і очікується, перерветься за `break`:

```
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
Final i = 6
```

Якщо оператор `break` вкладений відразу в декілька циклів, то він перериває найбільш вкладений із них.

Цикл while. У мові **Python** також використовується цикл `while`, заголовок якого містить деяку умову, що перевіряється перед початком ітерації, потім виконується тіло циклу. Якщо умова не виконається при початковій перевірці, то тіло циклу не виконається жодного разу. Слід звернути увагу на те, що належність інструкцій до тіла циклу визначається відступами. Напишемо за допомогою `while` аналог наступного циклу `for`:

```
for i in range(2, 8):  
    print(i)
```

Цей аналог матиме такий вигляд:

```
i = 2
while i < 8:
    print(i)
    i = i + 1
```

Очевидно, що `for` – простіше, коротше, і в ньому важче забути зробити приріст змінної. Крім того, такий `for` працюватиме швидше, ніж `while`, оскільки `range` написано на **Ci**, а операції `i < 8` і `i = i + 1` виконуються на **Python**. Замість `i = i + 1` можна поставити `i += 1`, оскільки оператора `++` у мові **Python** немає. Наведемо особливості використання умов заголовка в циклі `while`:

```
while умова:
    pass # "Порожня інструкція", у цьому випадку порож-
не тіло циклу
```

За цієї умови цикл не виконується.

Цикли з умовою (`do..while`) поста, що перевіряє умову після виконання тіла циклу, в **Python** не використовуються. Коли такий цикл потрібний, то зазвичай його імітують з допомогою нескінченного циклу й виходу за умовою через `break`:

```
while True:
    тіло_циклу
    if умова_завершення:
        break
```

Логічні вирази. `bool` – логічний тип, значення якого (`True`, `False`) повертають оператори порівняння (`<`, `>`, `<=`, `>=`, `==`, `!=`), а також оператори `in`, `not in`, `is`, `not is`. Результати цих операцій зазвичай і використовують в умовах циклу `while` та оператора `if`:

```
a = int(input("Введіть a: "))
b = int(input("Введіть b: "))
if a == b:
    print("Числа однакові")
else:
    print("Числа різні")
```

2.12. Використання бібліотеки **Matplotlib**

Бібліотека **Matplotlib** – це бібліотека двовимірної графіки для мови програмування **Python**, за допомогою якої можна створювати високоякісні рисунки різних форматів. **Matplotlib** є модулем-пакетом для **Python**.

Підключення Matplotlib. Пакет **Matplotlib** забезпечує `matlab`-подібний набір команд для побудови графіків. Він підключається такою інструкцією:

```
from pylab import *
```

Зазначимо, що при цьому також завантажується пакет `numpy`.

Побудова двовимірних графіків. Наведемо приклад побудови простих графіків. Для цього достатньо таких команд:

```
>>> from pylab import *  
>>> y=sin(linspace(0,10,100))  
>>> plot(y)  
[<matplotlib.lines.Line2D object at 0x0000000005211A58>]
```

Графік, проте, саме створюється, про що свідчить відповідь системи (у квадратних дужках), але нікуди не виводиться. Щоб його побачити, необхідно додати команду `show()`, яка виведе на екран вікно з графіком (рис. 2.16).

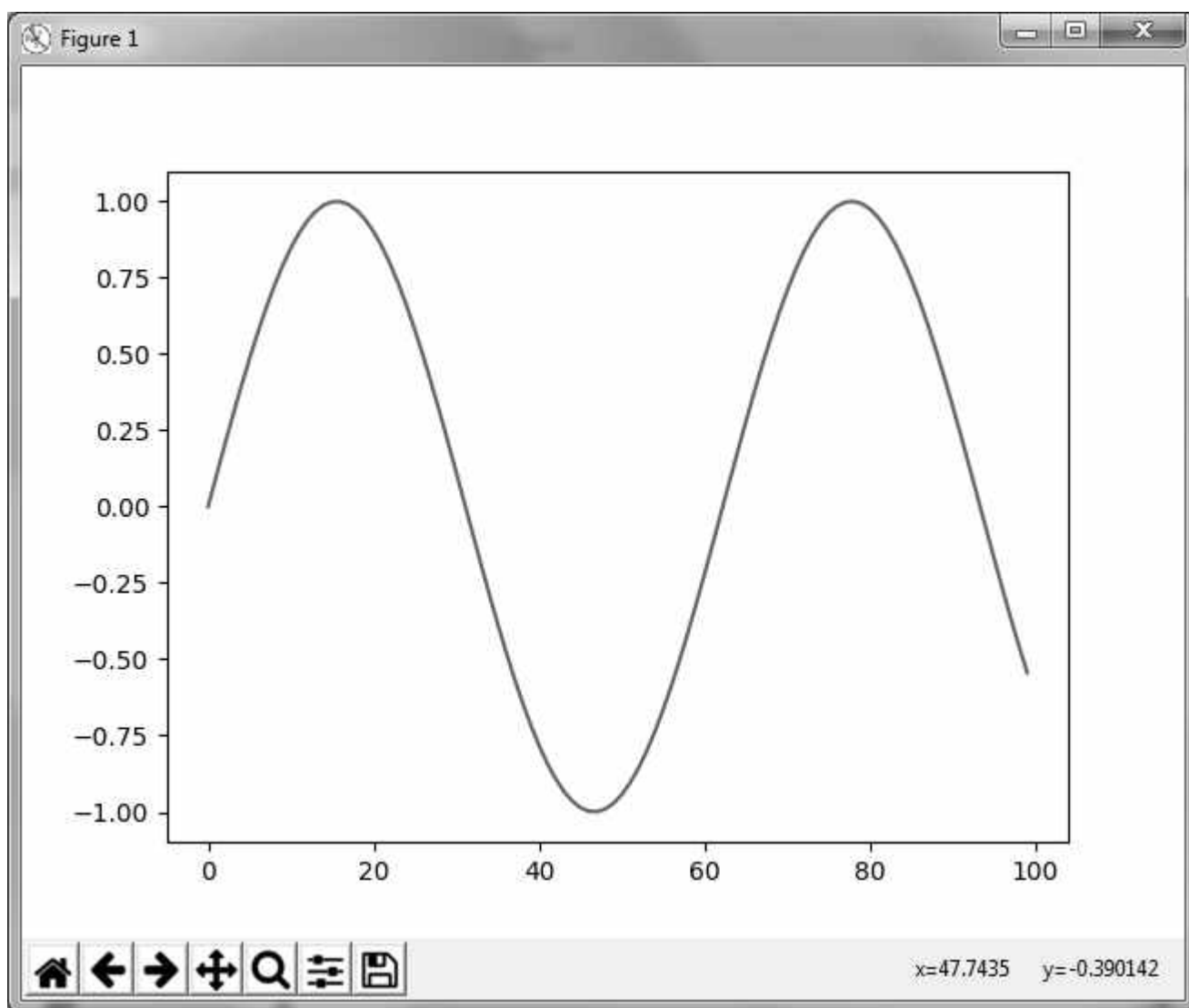


Рис. 2.16. Приклад побудови графіка

У графічному вікні зліва внизу є панель керування, яка складається з декількох кнопок. Пояснимо їх призначення, переміщаючись зліва направо. Перша кнопка, на якій зображено будинок, повертає перегляд графіка до вигляду, який був створений при відкритті вікна. Друга й третя кнопки зі

стрілками дають змогу переміщатися між видами, які можуть мати, наприклад, різний масштаб, зсув і т. д. Четверта кнопка (з хрестом) має два можливі режими. Вибравши цю кнопку, а потім, затиснувши ліву клавішу миші, можна переміщати графік у межах вікна. Натиснувши цю кнопку, а потім праву клавішу миші, можна змінювати масштаб зображення. П'ята кнопка дає можливість збільшувати або зменшувати вибрану зону графічного вікна. Натиснення шостої кнопки приводить до виклику панелі налаштувань графічного вікна. Сьома кнопка дає змогу зберегти рисунок у графічний файл у форматі `png` (за умовчанням) або в інших відомих форматах (`gif`, `jpeg` та ін.).

Як незалежна змінна в наведеному вище прикладі використовувався індекс масиву `0:100`. Проте можна задати її і явно:

```
>> from pylab import *
>> x=linspace(-10,10,50)
>> y=sin(x)/x
>> plot(x,y)
>> show()
```

Результат цієї процедури показано на рис. 2.17.

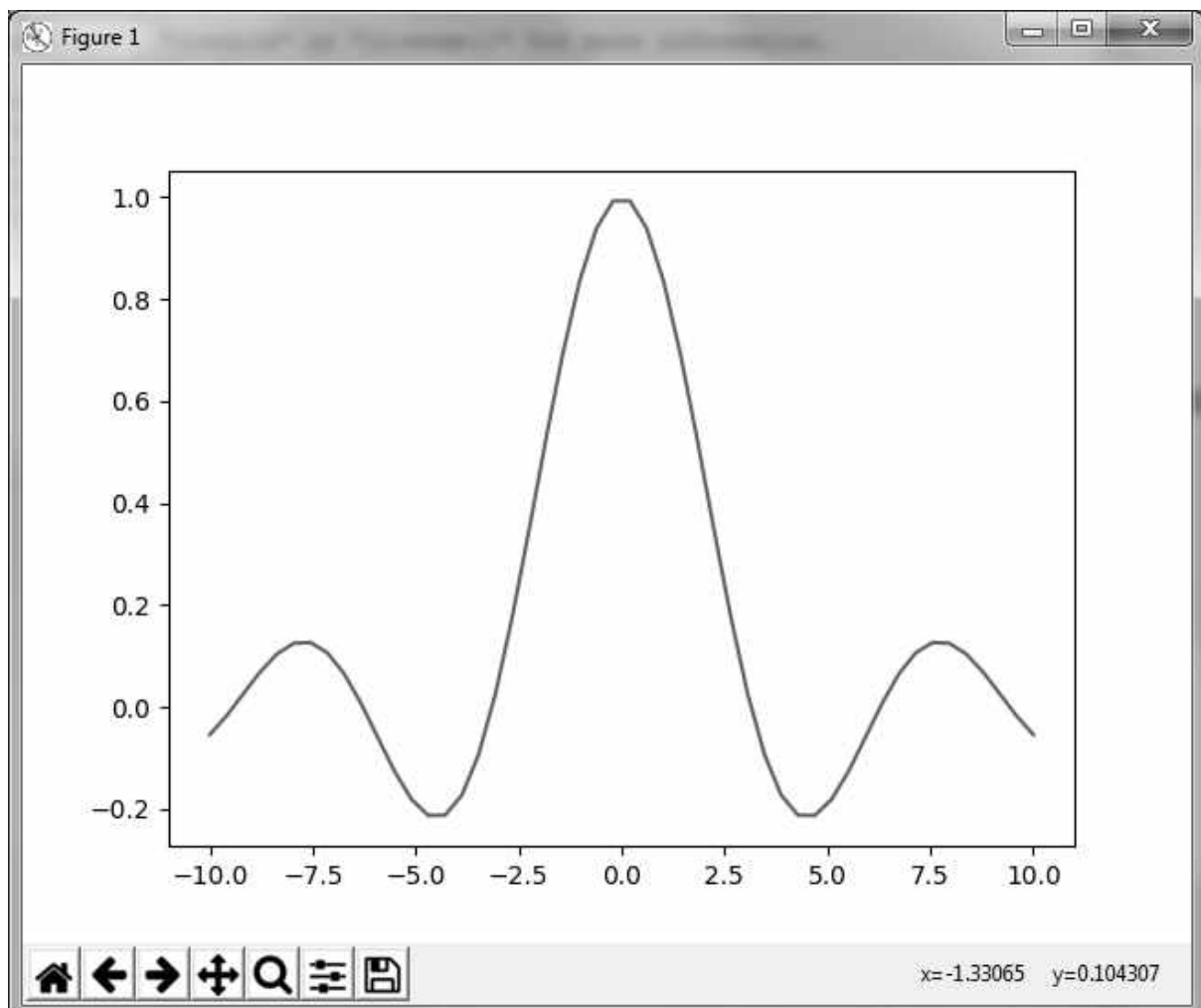


Рис. 2.17. Побудова графіка при заданні незалежної змінної в явному вигляді

Вигляд і колір графіка можна задавати єдиним параметром – рядком, що містить указівки:

- кольори (одна з букв `rgbycmkw`);
- вигляд лінії (`- -- -. :`);
- маркер (один з символів `.,o?v<>s+xDd1234hHp|_`).

Наведемо приклад (рис. 2.18) побудови графіка таким способом:

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,y,'mD--')
>>> show()
```

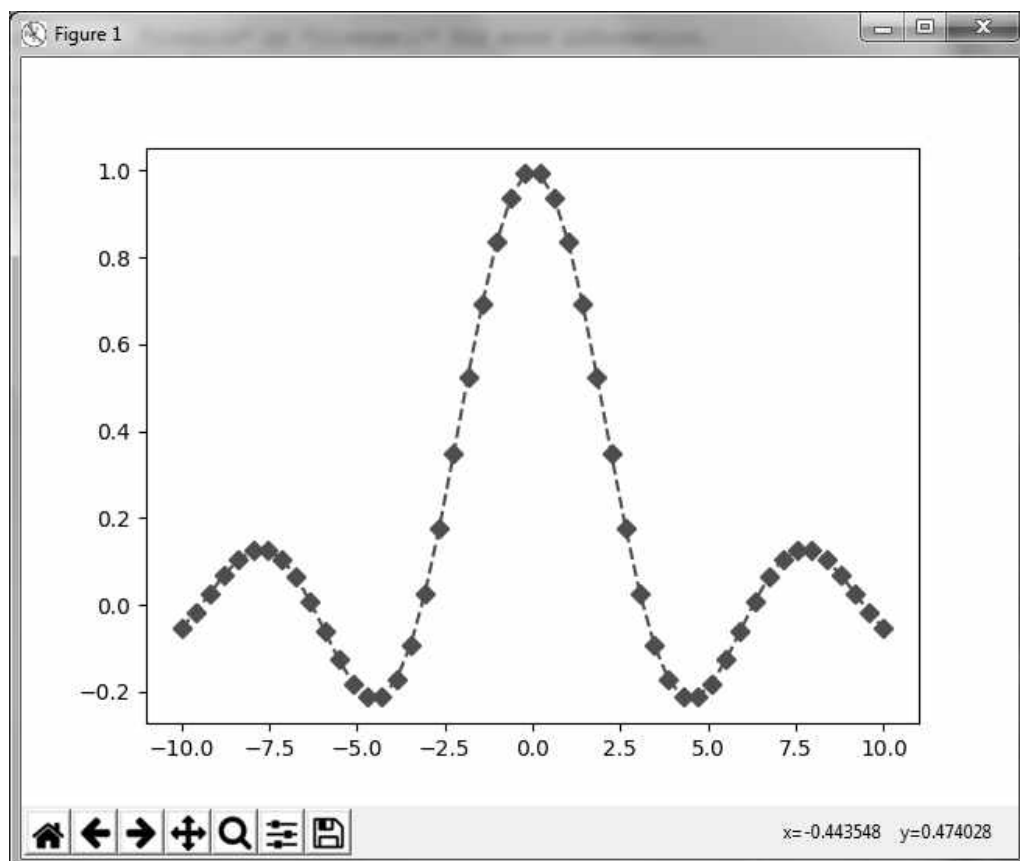


Рис. 2.18. Задання вигляду й кольору графіків

Детальне керування параметрами графіка забезпечується спеціальними аргументами:

- `c = <колір лінії>;`
- `ls = '<тип лінії>'` (те саме `- -- -. :`);
- `lw = <товщина лінії>;`
- `marker = '<тип маркера>'` (те саме `.,o^v<>s+xDd1234hHp|_`);
- `mec = <колір межі маркера>;`
- `mfc = <колір маркера>;`
- `mew = <товщина межі маркера>;`
- `ms = <розмір маркера>.`

Приклад детального керування параметрами графіка (рис. 2.19):

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> y=sin(x)/x
>>> plot(x,y,c='#7F007F',lw=3,marker='D',mec='#FF0000',\
mfc='#FFFF00',mew=2,ms=10)
[<matplotlib.lines.Line2D object at 0x0000000005100B38>]
>>> show()
```

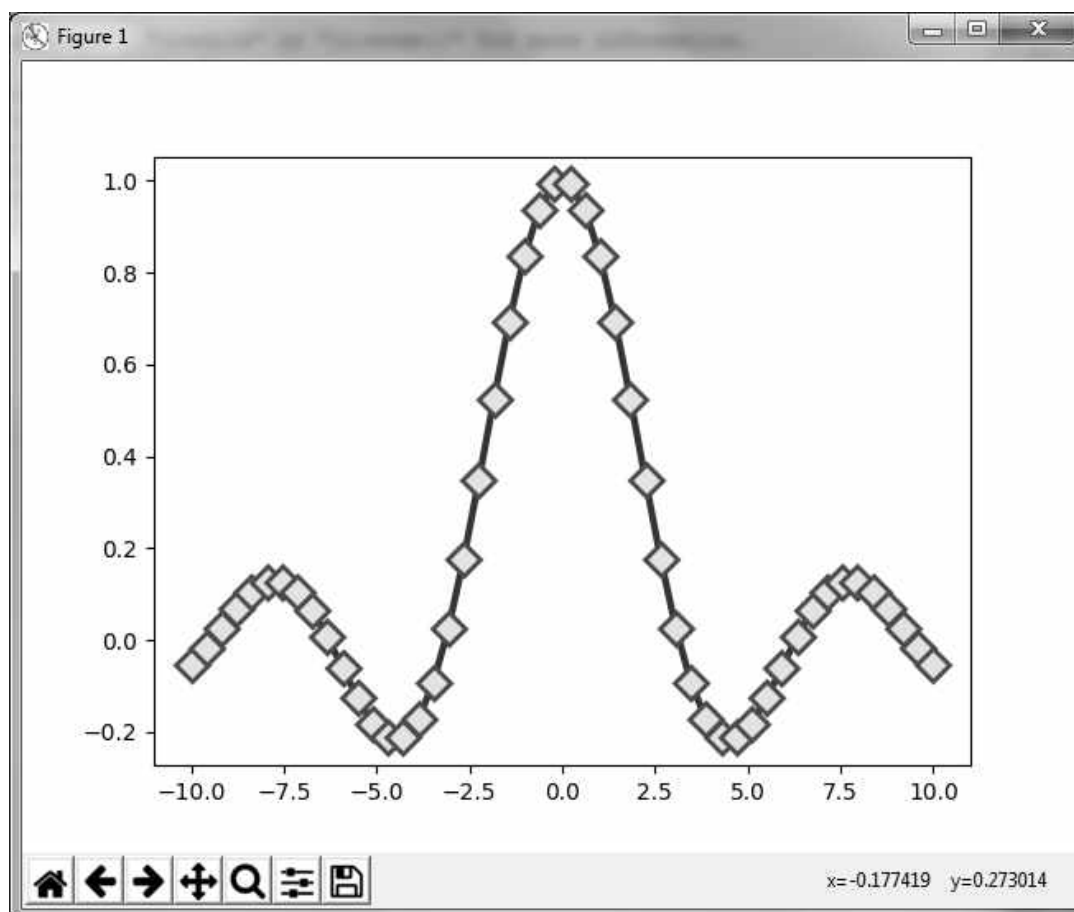


Рис. 2.19. Детальне керування параметрами графіка

Команди `xlim(xmin, xmax)` і `ylim(ymin, ymax)` задають масштаб по осях `x` і `y` (рис. 2.20):

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> y=sin(x)/x
>>> plot(x,y,'mD--')
[<matplotlib.lines.Line2D object at 0x0000000005020B38>]
>>> xlim(-1,5)
(-1, 5)
>>> ylim(0,1.1)
(0, 1.1)
>>> show()
```

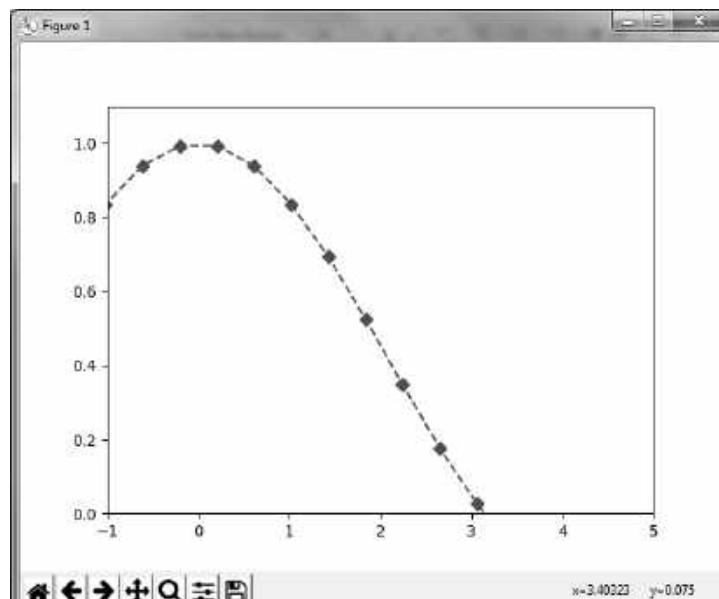


Рис. 2.20. Керування масштабами графіка

Для побудови декілька кривих на одному графіку використовують команди `plot`, кожна з яких додає свою криву (рис. 2.21):

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x0000000005101668>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x00000000051013C8>]
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x0000000001E7FF28>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> show()
```

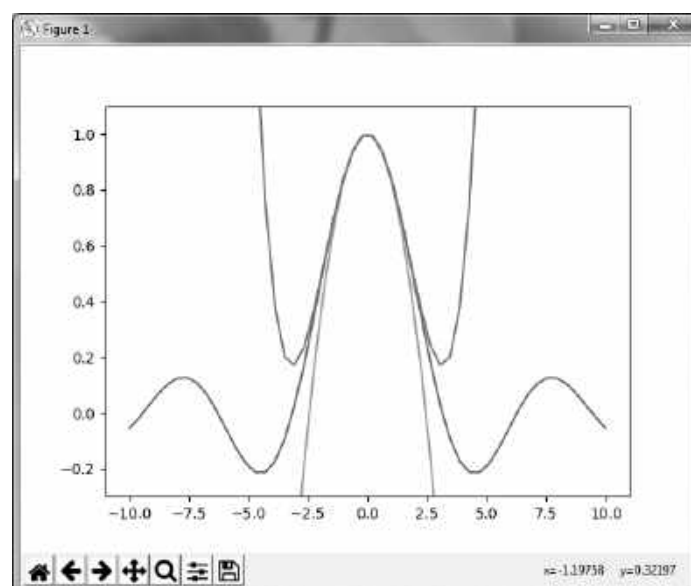


Рис. 2.21. Поеднання на графіку декілька кривих

Для `show()` розміщення декількох графіків в одному вікні застосовують команду `subplot`. Наведемо приклад її використання (рис. 2.22):

```
>>> from pylab import *
>>> x=linspace(0,10,50)
>>> subplot(221)
<matplotlib.axes._subplots.AxesSubplot object at
0x0000000004FF16A0>
>>> plot(x,sin(x))
[<matplotlib.lines.Line2D object at 0x0000000005020668>]
>>> subplot(222)
<matplotlib.axes._subplots.AxesSubplot object at
0x0000000005013C88>
>>> plot(x,cos(x))
[<matplotlib.lines.Line2D object at 0x0000000005020160>]
>>> subplot(223)
<matplotlib.axes._subplots.AxesSubplot object at
0x00000000051A4940>
>>> plot(x,sin(x)**2)
[<matplotlib.lines.Line2D object at 0x00000000051AD470>]
>>> subplot(224)
<matplotlib.axes._subplots.AxesSubplot object at
0x0000000005205470>
>>> plot(x,cos(x)**2)
[<matplotlib.lines.Line2D object at 0x0000000005205E10>]
>>> show()
```

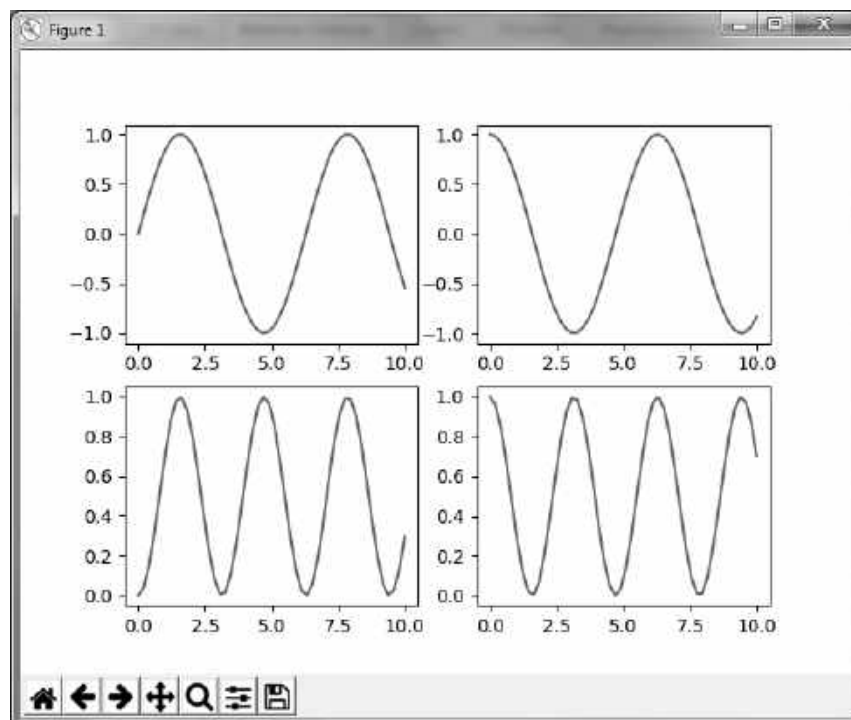


Рис. 2.22. Розміщення графіків в одному вікні

Команда `axes(<координати>, axisbg=<цвет фону>)` (другий аргумент необов'язковий) створює вкладений графік (рис. 2.23):

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x0000000005101780>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x0000000005101828>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> sp=axes([0.62,0.6,0.25,0.22],axisbg='#BFFFFFF')
>>> sp=axes([0.62,0.6,0.25,0.22],axisbg='#BFFFFFF')
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x000000000758C128>]
>>> show()
```

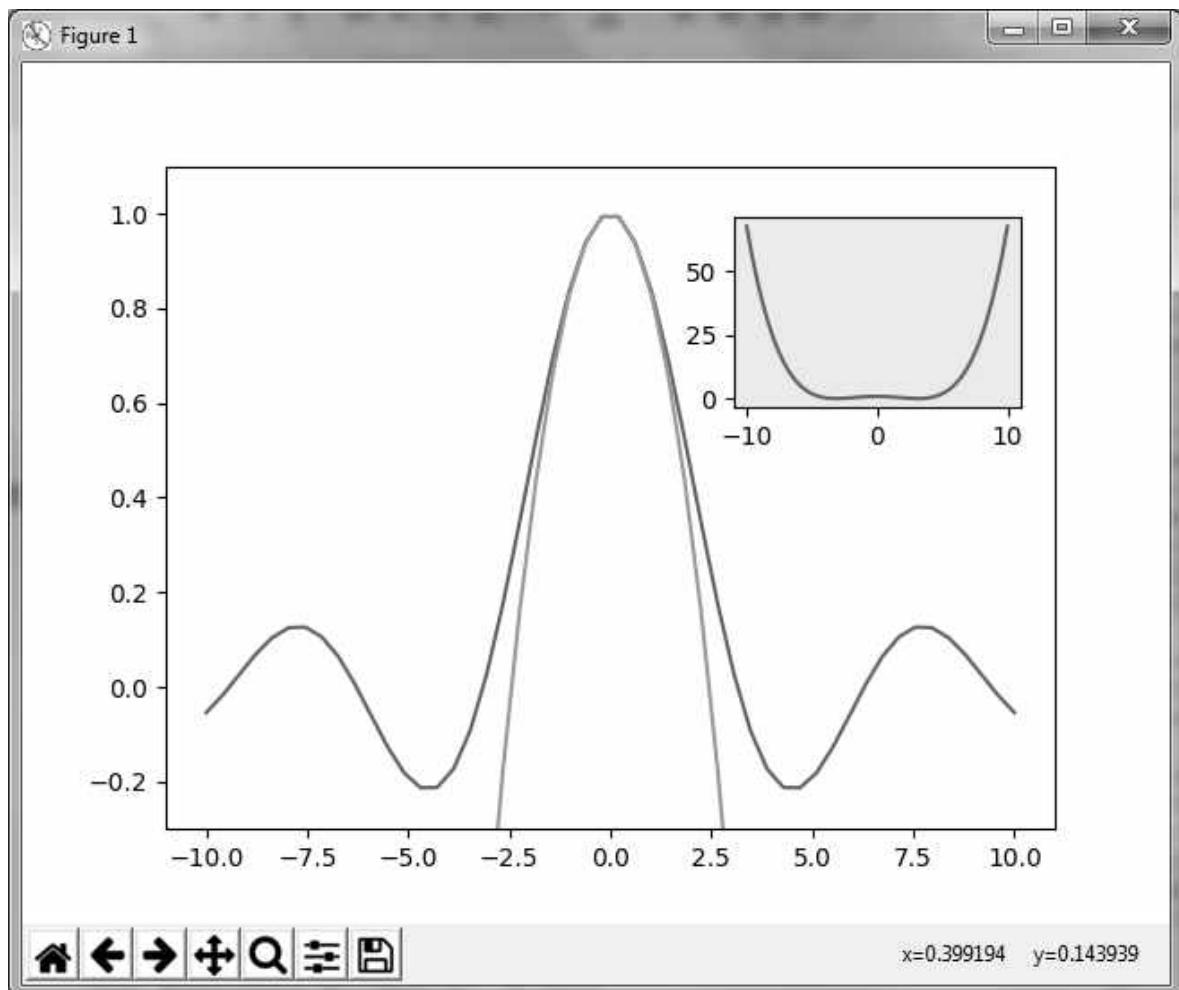


Рис. 2.23. Вкладення графіків

Усі графіки для однієї публікації зручно формувати одним скриптом. Для переходу від одного графіка до іншого використовується команда `figure()` (рис. 2.24):

```

>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x0000000005021780>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x0000000003013A90>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> figure()
<matplotlib.figure.Figure object at 0x0000000001DAF940>
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x00000000053CB0B8>]
>>> show()

```

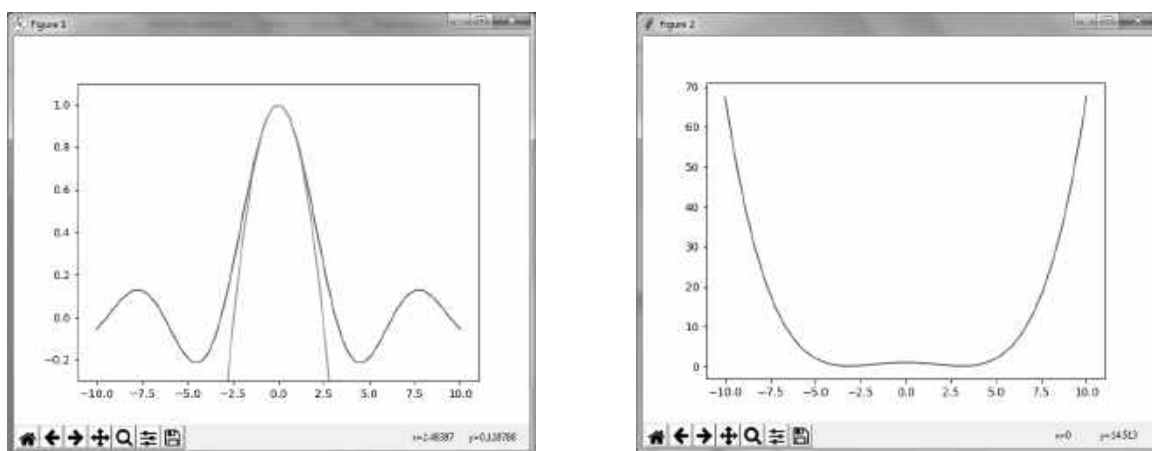


Рис. 2.24. Публікація всіх графіків одним скриптом

Команда `title` створює загальні заголовки до графіків і підграфіків (рис. 2.25):

```

>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x00000000051016A0>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x0000000005101B38>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> title('Taylor expansion')
<matplotlib.text.Text object at 0x000000000542C6A0>
>>> sp=axes([0.62,0.6,0.25,0.22],axisbg='#BFFFFFF')
>>> sp=axes([0.62,0.6,0.25,0.22],axisbg='#BFFFFFF')
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x000000000545F470>]
>>> title('Taylor 5')
<matplotlib.text.Text object at 0x0000000005166390>
>>> show()

```

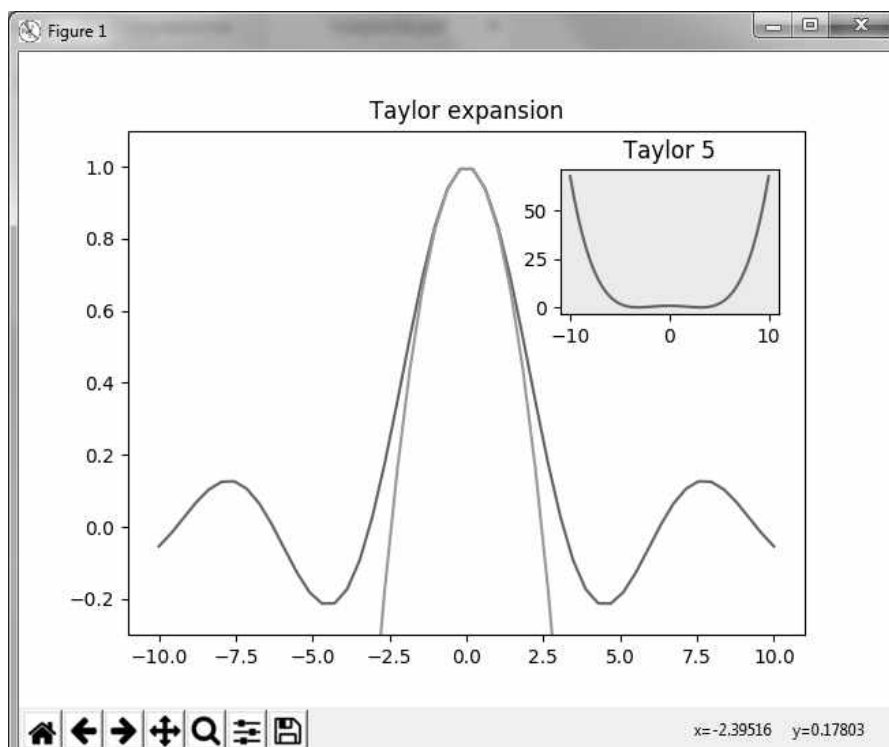


Рис. 2.25. Заголовки графіків

Команда `legend` створює описи до графіків (рис. 2.26):

```
import math
# Імпортуємо один з пакетів Matplotlib
import pylab
# Імпортуємо пакет з допоміжними функціями
from matplotlib import mlab
# Рисуватимемо графік цієї функції
def func (x):
    """
    sinc (x)
    """
    if x == 0:
        return 1.0
    return math.sin (x) / x
# Інтервал зміни змінної по осі X
xmin = -20.0
xmax = 20.0
# Крок між точками
dx = 0.01
# Створимо список координат по осі X на відрізку [-xmin;
xmax], включаючи кінці
xlist = mlab.frange (xmin, xmax, dx)
# Обчислимо значення функції в заданих точках
ylist1 = [func (x) for x in xlist]
```

```

ylist2 = [func (x * 0.2) for x in xlist]
# Нарисуємо два одновимірні графіки
pylab.plot (xlist, ylist1, "b-")
pylab.plot (xlist, ylist2, "g--")
# Додамо легенду.
# Першому графіку відповідатиме напис "f(x)",
# А другому - "f(0.2 * x)"
pylab.legend (("f(x)", "f(0.2 * x)"))
# Покажемо вікно з нарисованим графіком
pylab.show()

```

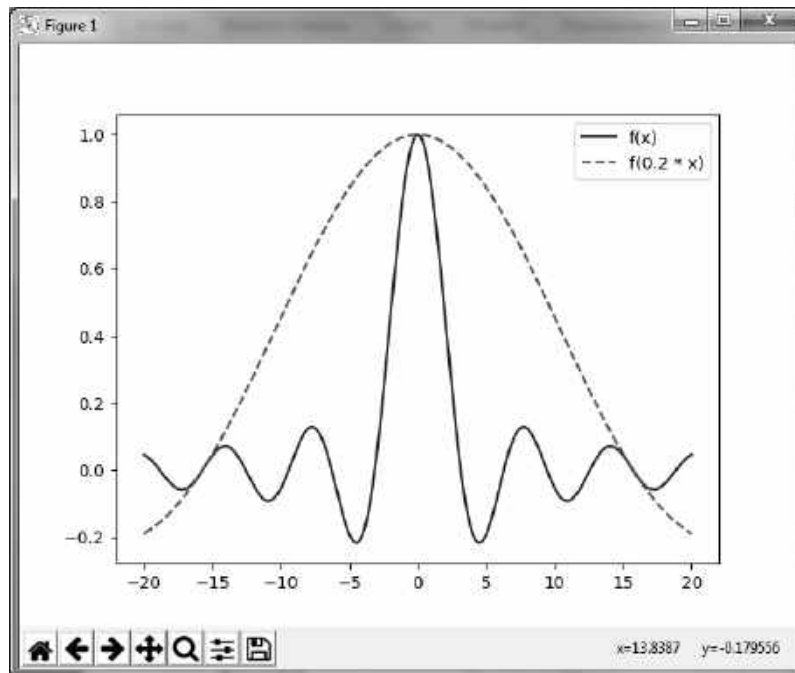


Рис. 2.26. Використання команди legend

Команди xlabel і ylabel створюють написи по осях (рис. 2.27,а):

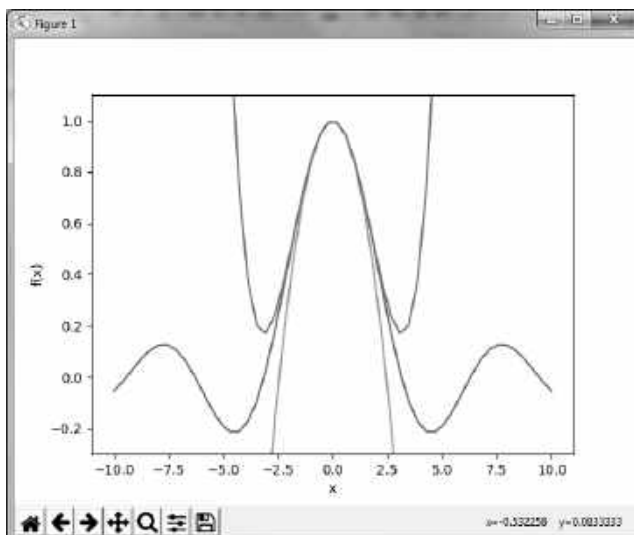
```

>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x00000000035F16A0>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x00000000035F1B38>]
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x00000000026FFF28>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> xlabel('x')
<matplotlib.text.Text object at 0x00000000035E2748>
>>> ylabel('f(x)')
<matplotlib.text.Text object at 0x00000000035F8DD8>
>>> show()

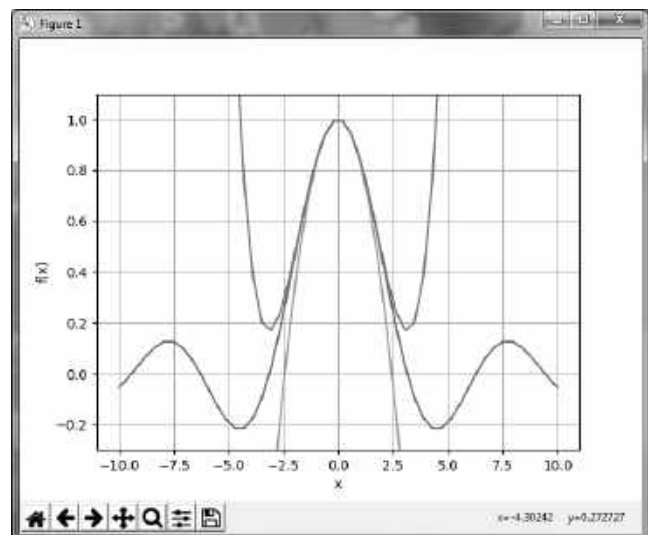
```

Команда `grid` рисує сітку на графіку (рис. 2.27, б):

```
>>> from pylab import *
>>> x=linspace(-10,10,50)
>>> plot(x,sin(x)/x)
[<matplotlib.lines.Line2D object at 0x0000000005021780>]
>>> plot(x,1-x**2/6)
[<matplotlib.lines.Line2D object at 0x00000000030D3AC8>]
>>> plot(x,1-x**2/6+x**4/120)
[<matplotlib.lines.Line2D object at 0x000000000021BFF28>]
>>> ylim(-0.3,1.1)
(-0.3, 1.1)
>>> xlabel('x')
<matplotlib.text.Text object at 0x000000000050127F0>
>>> ylabel('f(x)')
<matplotlib.text.Text object at 0x00000000005027E80>
>>> grid()
>>> show()
```



а



б

Рис. 2.27. Нанесення сітки на графіки

Команда `polar(theta,r)` рисує графік у полярних координатах. Останні необов'язкові параметри такі самі, що й команди `plot` (рис. 2.28):

```
>>> from pylab import *
>>> x=linspace(-2*pi,2*pi,100)
>>> y=sin(x)/x
>>> polar(x, y, c='blue', lw=2)
[<matplotlib.lines.Line2D object at 0x000000000054C6CC0>]
>>> #grid()
>>> show()
```

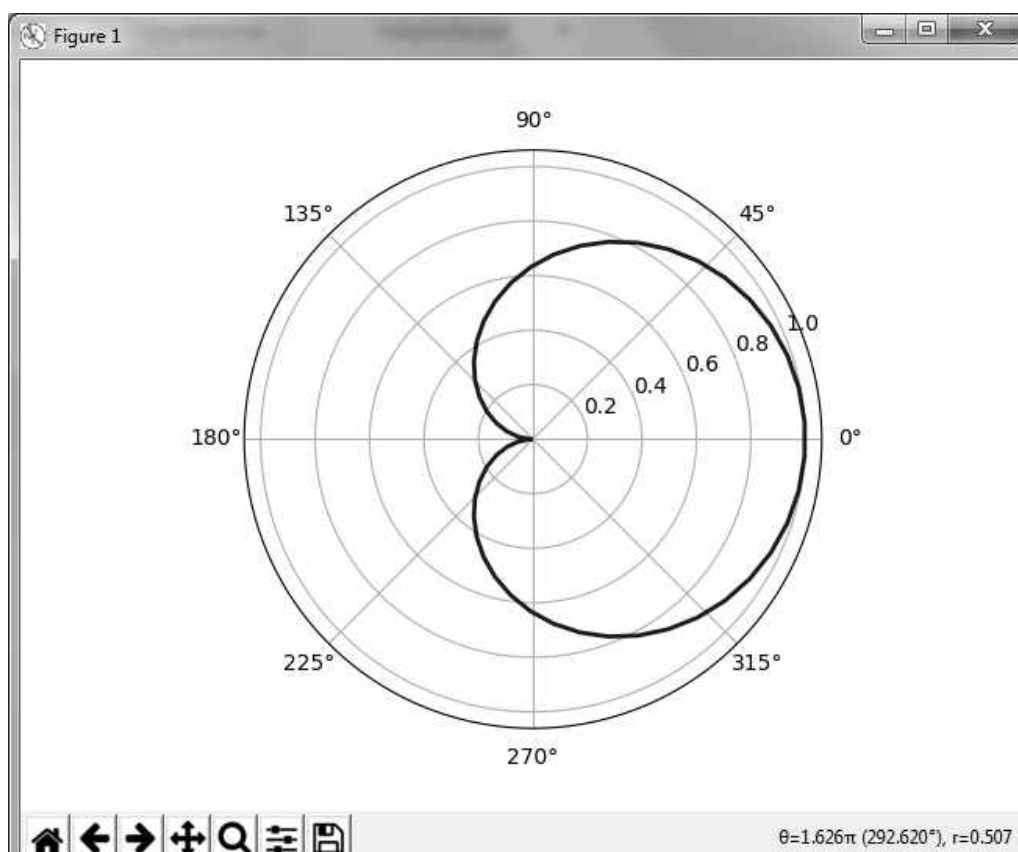


Рис. 2.28. Графіки в полярних координатах

Побудова тривимірних графіків. Усі класи для роботи з тривимірними графіками містяться в пакеті `mpl_toolkits.mplot3d`, з якого їх потрібно імпортувати. Крім того, необхідною є математична бібліотека `numpy`, оскільки в деяких методах класів рисуння тривимірних графіків як параметри використовуються екземпляри класу `numpy.array`. Бібліотека `Numpy` дає змогу значно зменшити кількість рядків коду.

Для того щоб нарисувати тривимірний графік, необхідно створити тривимірні осі. Для цього потрібно використовувати екземпляр класу `mpl_toolkits.mplot3d.Axes3D`, конструктор якого задає як мінімум один параметр – екземпляр класу `matplotlib.figure.Figure`, який, у свою чергу, можна створити викликом `pylab.figure()`. У конструкторі класу `matplotlib.figure.Figure` є й інші необов'язкові параметри.

Спочатку нарисуємо вихідні тривимірні осі без формування графіка (рис. 2.29). Для цього створимо такий код:

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
fig = pylab.figure()
Axes3D(fig)
<mpl_toolkits.mplot3d.axes3d.Axes3D object at
0x000000000522B1D0>
pylab.show()
```

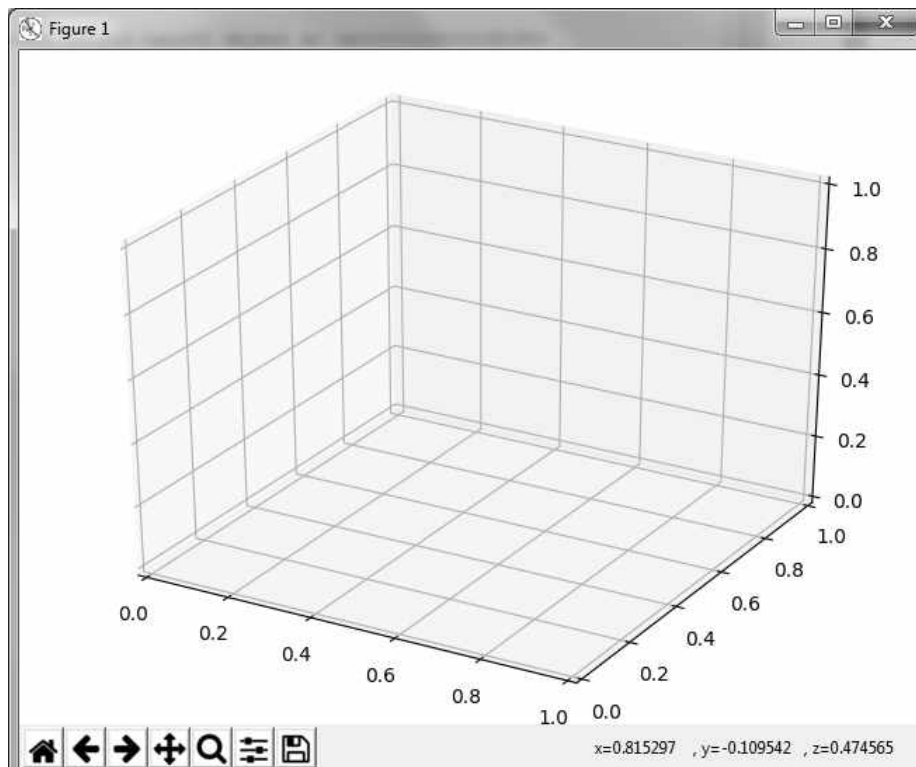


Рис. 2.29. Формування осей для тривимірних графіків

Для побудови графіка використовуємо функцію від двох координат:

$$f(x, y) = \frac{\sin(x) \sin(y)}{xy}.$$

Спочатку потрібно підготувати дані для рисування. Для цього знадобляться три двовимірні матриці: матриці X і Y зберігатимуть координати сітки точок, у яких обчислюватиметься наведена вище функція, а матриця Z зберігатиме значення цієї функції у відповідній точці.

При створенні тривимірного графіка на еквідистантній сітці для створення матриць, які зберігатимуть координати, використовують функцію `meshgrid()` з бібліотеки `numpy`. Ця функція формує двовимірні матриці сіток за одновимірними масивами:

```
>>> X,Y=numpy.meshgrid([1,2,3],[4,5,6,7])
>>> X
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
>>> Y
array([[4, 4, 4],
       [5, 5, 5],
       [6, 6, 6],
       [7, 7, 7]])
```

Для розділення процесу підготовки даних для рисування (створення сітки і розрахунок функції) сформуємо окрему функцію:

```
Def makeData ():  
    # Будуємо сітку в інтервалі від -10 до 10 з кроком 0.1  
    по обох координатах  
    x = numpy.arange (-10, 10, 0.1)  
    y = numpy.arange (-10, 10, 0.1)  
  
    # Створюємо двовимірну матрицю-сітку  
    xgrid, ygrid = numpy.meshgrid(x, y)  
  
    # У вузлах розраховуємо значення функції  
    zgrid=numpy.sin(xgrid)*numpy.sin(ygrid)/(xgrid*ygrid)  
    return xgrid, ygrid, zgrid
```

Ця функція повертає три двовимірні матриці: x , y , z . Координати x і y лежать в інтервалі від -10 до 10 з кроком 0.1. Використовуємо її для створення графіка.

Щоб дані відображувалися, достатньо викликати метод `plot_surface()` екземпляра класу `Axes3d`, у який потрібно передати отримані за допомогою функції `makeData()` двовимірні матриці. Остаточ-но код побудови графіка має такий вигляд:

```
Import pylab  
From mpl_toolkits.mplot3d import Axes3D  
Import numpy  
  
def makeData ():  
    x = numpy.arange (-10, 10, 0.1)  
    y = numpy.arange (-10, 10, 0.1)  
    xgrid, ygrid = numpy.meshgrid(x, y)  
  
    zgrid=numpy.sin(xgrid)*numpy.sin(ygrid)/(xgrid* ygrid)  
    return xgrid, ygrid, zgrid  
  
x, y, z = makeData()  
  
fig = pylab.figure()  
axes = Axes3D(fig)  
  
axes.plot_surface(x, y, z)  
  
pylab.show()
```

Графік функції зображено на рис. 2.30.

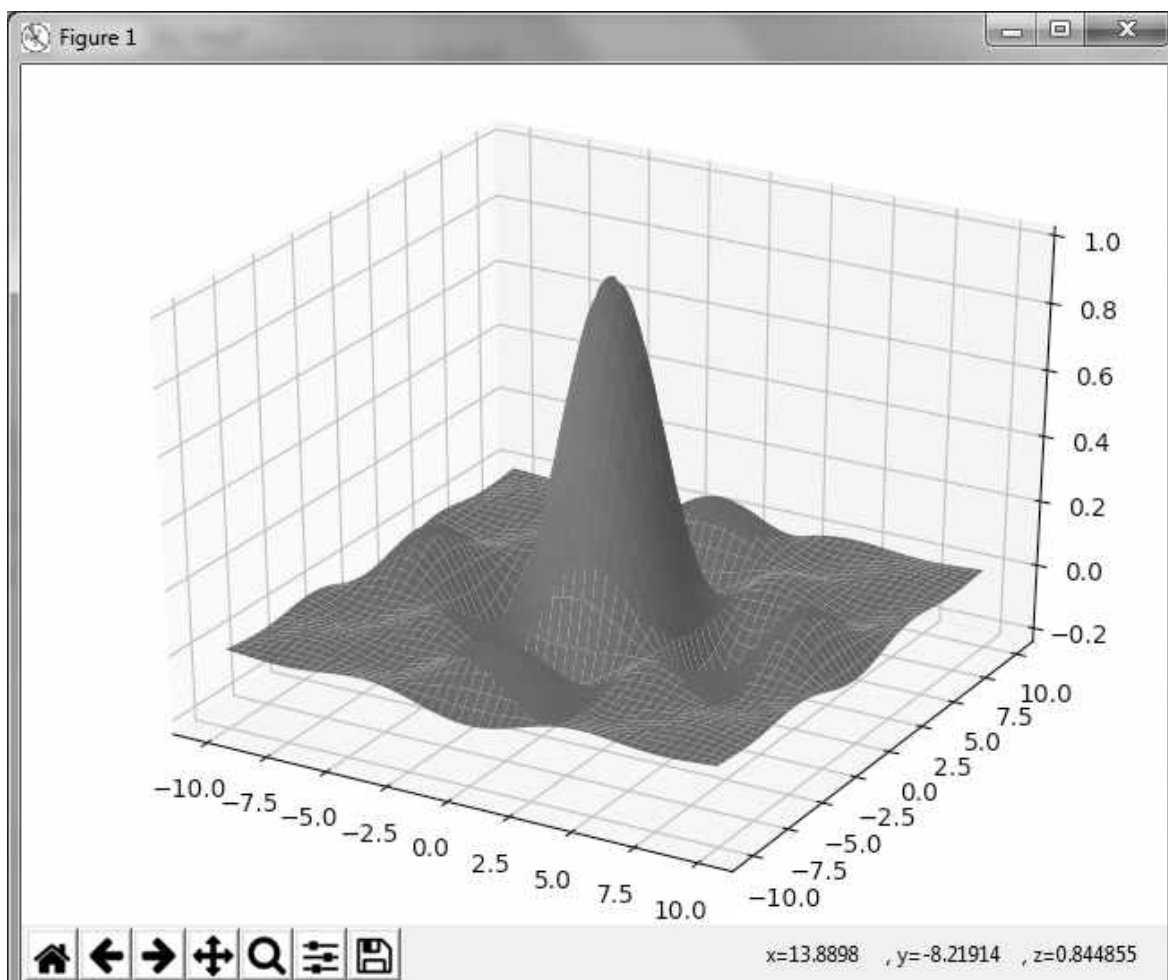


Рис. 2.30. Приклад тривимірного графіка

Змінення зовнішнього вигляду графіка здійснюється шляхом використання додаткових параметрів методу `Axes3d.plot_surface(X,Y,Z,*args,**kwargs)`:

- `X`, `Y` і `Z` – параметри, які задають сітку і значення функції у вузлах;
- `rstride` і `cstride` задають крок виведення графіка (чим менше крок, тим точніше відображується функція, але тим довше відбувається рисування);
- `color` задає колір графіка;
- `cmap` задає градієнт кольорів, щоб колір вічка графіка залежав від значення функції в цій області.

Розглянемо деякі методи керування параметрами відображення графіка, наведеного в попередньому прикладі. За допомогою методу `plot_surface()` змінимо крок сітки. Для цього внесемо змінення в рядок вихідного коду, додаючи значення параметрів `rstride` і `cstride`:

```
axes.plot_surface(x,y,z,rstride=20,cstride=20
```

Унаслідок цих змінень отримано більший крок сітки на графіку, що добре видно при порівнянні двох графіків на рис. 2.30 і 2.31.

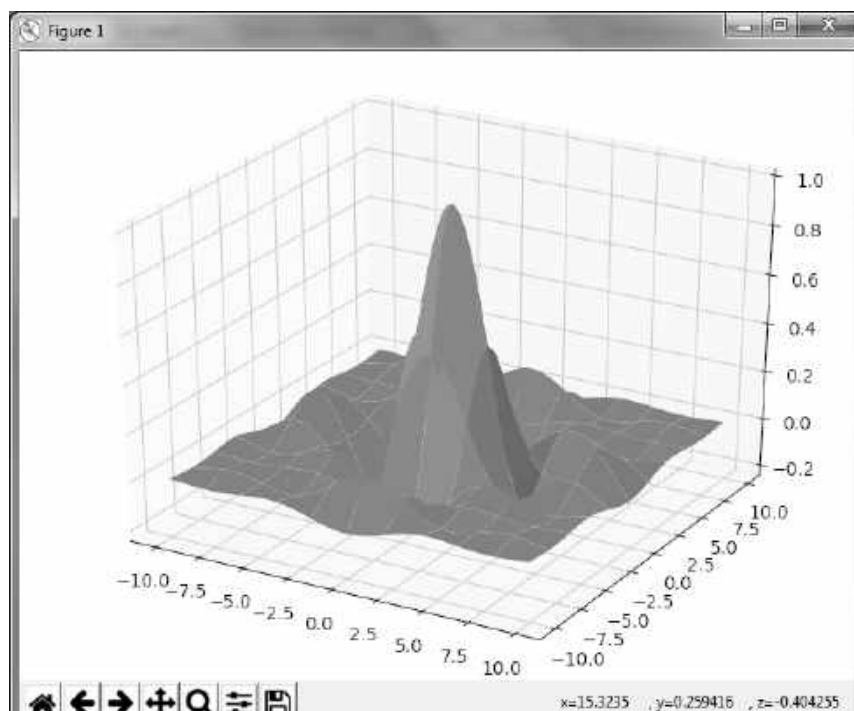


Рис. 2.31. Керування параметрами відображення

Далі розглянемо змінення кольору поверхні за допомогою параметра `color`. Цей параметр є рядком, який описує колір і може задаватися різними способами.

Колір визначається словом для відповідного кольору або його початковою літерою:

- 'b' або 'blue';
- 'g' або 'green';
- 'r' або 'red';
- 'c' або 'cyan';
- 'm' або 'magenta';
- 'y' або 'yellow';
- 'k' або 'black';
- 'w' або 'white'.

Наведемо приклад змінення кольору поверхні фігури на графіку. Для цього внесемо змінення в рядок вихідного коду, і колір поверхні фігури на графіку зміниться на жовтий:

```
axes.plot_surface(x, y, z, color='yellow')
```

Колірні карти використовуються, якщо потрібно вказати, у які кольори повинні забарвлюватися ділянки тривимірної поверхні залежно від значення z у цій області (задання колірного градієнта). Щоб при виведенні графіка використовувався градієнт як значення параметра `cmap` (від `colormap` – *колірна карта*), необхідно передати екземпляр класу `matplotlib.colors.Colormap`.

Пакет **Matplotlib** містить безліч колірних карт:

Spectral, copper, RdYlGn, Set2, summer, spring, Accent, OrRd, RdBu, autumn, Set1, PuBu, Set3, gist_rainbow, pink, binary, winter, jet, BuPu, Dark2, prism, Oranges, gist_yarg, BuGn, hot, PiYG, YlOrBr, Reds, spectral, RdPu, Greens, gist_ncar, PRGn, gist_heat, YlGnBu, RdYlBu, Paired, flag, hsv, BrBG, Purples, cool, Pastel2, gray, Pastell1, gist_stern, GnBu, YlGn, Greys, RdGy, YlOrRd, PuOr, PuRd, gist_gray, Blues, PuBuGn, gist_earth, bone.

Наведемо приклад використання карти `cm.jet`. Тоді код побудови графіка (рис. 2.32) матиме такий вигляд:

```
Import pylab
From mpl_toolkits.mplot3d import Axes3D
From matplotlib.colors import LinearSegmentedColormap
From matplotlib import cm
Import numpy
Def makeData ():
    x = numpy.arange (-10,10,0.1)
    y = numpy.arange (-10,10,0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin(xgrid) * numpy.sin(ygrid) / (xgrid *
ygrid)
    return xgrid, ygrid, zgrid
x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=4, cstride=4, cmap =
cm.jet)
pylab.show()
```

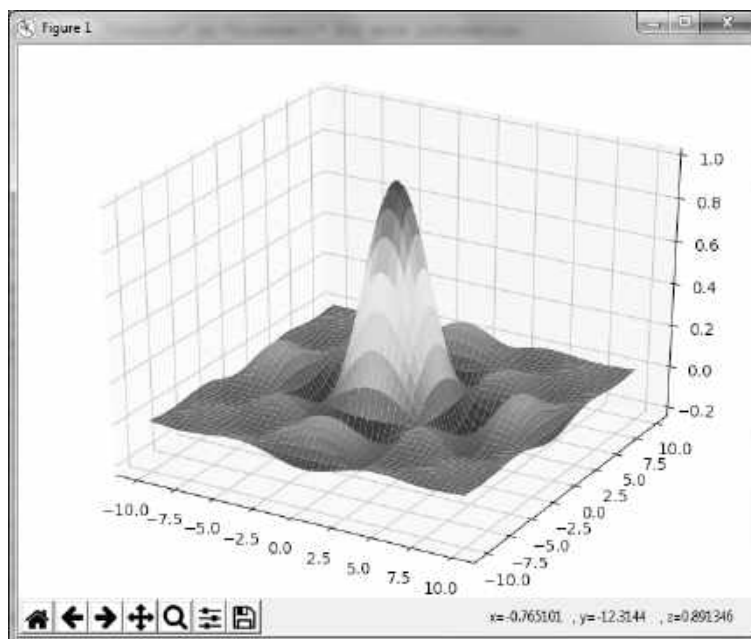


Рис. 2.32. Приклад використання колірних карт

У цьому розділі розглянуто лише основні питання побудови й оформлення графіків функцій за результатами розрахунків. Розширені можливості подання графічної інформації читач може засвоїти самостійно.

3. СТВОРЕННЯ ІНТЕРАКТИВНИХ ІНТЕРФЕЙСІВ

У різних завданнях оброблення інформації для ефективного використання програмного забезпечення завжди необхідно формувати додатки з інтерактивним графічним інтерфейсом **GUI** (Graphical User Interface), призначеним для користувача. При створенні таких програм важливими є не лише самі алгоритми оброблення даних, але й розроблення для користувача програми зручного інтерфейсу, за допомогою якого легко керувати поведінкою додатка.

Взаємодію з програмою зазвичай здійснюють за допомогою набору різних меню і кнопок для введення інформації в спеціальні поля, а також для формування керувальних команд. На базі цих елементів, які називають віджетами (від англ. widget – *штучка*), зазвичай формують **GUI**.

У **Python** віджети зберігаються в спеціальній бібліотеці – `tkinter`. Імпорт віджетів у робочу програму (скрипт) дає можливість ефективно створювати графічний інтерфейс. Зазначимо, що при імпорті `tkinter` для версій **Python 3** ім'я модуля слід писати з малої літери.

Послідовність кроків при створенні графічного додатка має низку особливостей. Програма повинна виконувати своє основне призначення, бути зручною для користувача, адекватно реагувати на його дії. Розглянемо етапи програмування, необхідні для ефективного формування графічного інтерфейсу **GUI**:

1. Імпорт бібліотеки.
2. Створення головного вікна.
3. Створення віджетів.
4. Установлення властивостей віджетів.
5. Визначення подій.
6. Визначення обробників подій.
7. Розташування віджетів у головному вікні.
8. Відображення головного вікна.

3.1. Основні етапи роботи з віджетами при створенні інтерфейсу

Імпорт бібліотеки. Імпорт модуля `tkinter` в **Python**, як і будь-якого іншого модуля, виконується з допомогою команд `import tkinter` і `from tkinter import *`. Друга команда є переважною, оскільки не треба кожного разу вказувати ім'я модуля при зверненні до об'єктів, які в ньому містяться. Ще раз звертаємо вашу увагу на те, що у версії **Python 3** ім'я мо-

дуля пишеться з малої літери (`tkinter`), а в раніших версіях використовувалася велика (`Tkinter`). Перший рядок програми записується так:

```
from tkinter import *
```

Створення головного вікна. У сучасних операційних системах будь-який додаток, призначений для користувача, міститься у вікні, яке можна назвати головним, оскільки в ньому розташовуються всі останні віджети. Об'єкт-вікно верхнього рівня створюється при зверненні до класу `Tk` модуля `tkinter`. Змінну, зв'язану з об'єктом-вікном, називають `root`. Другий рядок коду записується так:

```
root = Tk()
```

Створення віджетів. Для прикладу сформуємо вікно, у якому розташовуватиметься всього одна кнопка. Кнопка створюється при зверненні до класу `Button` модуля `tkinter`. Об'єкт-кнопка зв'язується з якою-небудь змінною. У класі `Button` (як і в усіх останніх класах, за винятком `Tk`) є обов'язковий параметр – об'єкт, якому кнопка належить (кнопка не може "бути нічийною"). Поки маємо єдине вікно (`root`), воно й буде аргументом, що передається до класу при створенні об'єкта-кнопки:

```
but = Button(root)
```

Установлення властивостей віджетів. Кнопка має багато властивостей: розмір, колір фону й напису та ін. Для прикладу встановимо всього одну властивість – текст напису (`text`):

```
but["text"] = "Печать"
```

Визначення подій і їх обробників. Графічний інтерфейс зазвичай відслідковує різні події, пов'язані з клавіатурою й мишею, що відбуваються при використанні того або іншого віджета. У `Tk` події описуються у вигляді текстового рядка – шаблону події, що складається з трьох елементів (модифікатора, типу події і деталізації події). Наведемо список цих подій у табл. 3.1.

Таблиця 3.1

Тип події	Зміст події
Activate	Активізація вікна
ButtonPress	Натиснення кнопки мишки
ButtonRelease	Відтискання кнопки мишки
Deactivate	Деактивація вікна
Destroy	Закриття вікна

Тип події	Зміст події
Enter	Входження курсора в межі віджета
FocusIn	Одержання фокусу вікном
FocusOut	Втрата фокусу вікном
KeyPress	Натиснення клавіші на клавіатурі
KeyRelease	Відтискання клавіші на клавіатурі
Leave	Вихід курсора за межі віджета
Motion	Рух миші в межах віджета
MouseWheel	Прокрутка коліщатка миші
Reparent	Змінення породного вікна
Visibility	Змінення видимості вікна

Для цього прикладу необхідно визначити порядок дій – що робитиме кнопка і в який момент часу? Передбачимо, що завдання кнопки – вивести яке-небудь повідомлення в потік виводу з використанням функції `print`. Вона повинна виконати це при натисненні на неї лівою кнопкою мишки.

Дії, які відбуваються при тій або іншій події, можуть бути досить складними. Тому зазвичай їх оформляють у вигляді функції, а потім викликають її, якщо потрібно. У цьому прикладі друк на екран оформимо у вигляді функції `printer`:

```
def printer(event):
    print ("Виведення на друк 'Аналіз даних'")
```

Слід пам'ятати, що функцію необхідно розміщувати на початку коду. Параметр `event` – це яка-небудь подія. Подія натиснення лівою кнопкою миші має такий вигляд: `<Button-1>`. Потрібно зв'язати цю подію з обробником (функцією `printer`) з допомогою методу `bind`.

Синтаксис організації скріплення події з обробником має такий вигляд:

```
but.bind("<Button-1>",printer)
```

Розміщення віджетів. Для будь-якого програмного додатка віджети потрібно розмістити так, щоб інтерфейс був простим та ергономічним і не потребував додаткових дій користувача. Найпростіший спосіб – це використання методу `pack`:

```
but.pack()
```

Цей рядок коду є необхідним, оскільки при його відсутності кнопка у вікні інтерфейсу не відображатиметься, не зважаючи на те, що вона є в програмі.

Головне вікно відображається внаслідок виклику спеціального методу `mainloop`:

```
root.mainloop()
```

Цей рядок коду завжди розміщують у кінці створюваного скрипту.

З урахуванням прийнятих правил код програми для цього прикладу має такий вигляд:

```
from tkinter import *
def printer(event):
    print ("Аналіз даних")
root = Tk()
but = Button(root)
but["text"] = "Друк"
but.bind("<Button-1>",printer)
but.pack()
root.mainloop()
```

Результати роботи цієї програми показано на рис. 3.1.

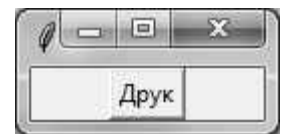
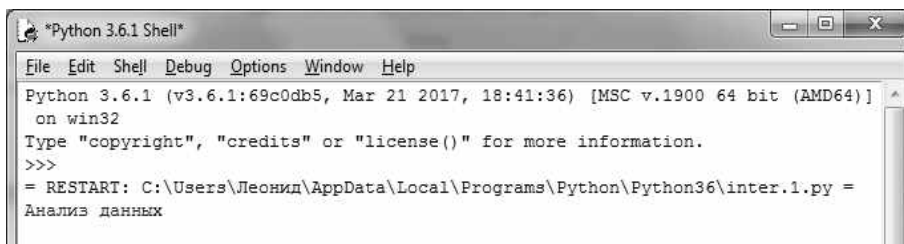


Рис. 3.1. Результат роботи в командному вікні при натисненні кнопки «Друк»

При програмуванні графічного інтерфейсу користувача найефективнішим є об'єктно-орієнтований підхід, тому програму краще оформляти з використанням класів. У цьому прикладі можна використовувати клас:

```
from tkinter import *
class But_print:
    def __init__(self):
        self.but = Button(root)
        self.but["text"] = "Друк"
        self.but.bind("<Button-1>",self.printer)
        self.but.pack()
    def printer(self,event):
        print ("Виведення на друк 'Аналіз даних'")
root = Tk()
obj = But_print()
root.mainloop()
```

3.2. Графічні об'єкти та їх властивості

Розглянемо основні графічні об'єкти (віджети), що містяться в бібліотеці Tkinter, – кнопки, поля для введення, мітки, прапорці, перемикачі й списки. Кожний клас віджетів має певні властивості, значення яких можна задавати при їх створенні, а також програмувати їх зміну при діях користувача і внаслідок виконання програми. Наведемо основні класи віджетів, що використовуються при створенні графічних інтерфейсів:

- `Button` (кнопка) – проста кнопка для виклику деяких дій (виконання певної команди);
- `Canvas` (рисунок) – основа для виведення графічних примітивів;
- `Checkbutton` (прапорець) – кнопка, яка вміє перемикатися між двома станами при натисненні на неї;
- `Entry` (поле введення) – горизонтальне поле, у яке можна ввести рядок тексту;
- `Frame` (рамка) – віджет, у якому містяться інші візуальні компоненти;
- `Label` (напис) – віджет, який може показувати текст або графічне зображення;
- `Listbox` (список) – прямокутна рамка зі списком, з якого користувач може виділити один або декілька елементів;
- `Menu` (меню) – елемент, за допомогою якого можна створювати спливні (`popup`) і випадні (`pulldown`) меню;
- `Menubutton` (кнопка-меню) – кнопка з випадним меню;
- `Message` (повідомлення) – повідомлення, що є аналогічним напису, але дає змогу згорнути довгі рядки й змінювати розмір на вимогу менеджера розташування;
- `Radiobutton` (селекторна кнопка) – кнопка для подання одного з альтернативних значень (такі кнопки зазвичай діють у групі, при натисненні однієї із них кнопка групи, вибрана раніше, «відскакує»);
- `Scale` (шкала) – шкала для задання числового значення шляхом переміщення повзуна в певному діапазоні;
- `Scrollbar` (смуга прокручування) – смуга для відображення величини прокручування в інших віджетах, яка може бути як вертикальною, так і горизонтальною;
- `Text` (форматований текст) – прямокутний віджет, який дає змогу редагувати й формувати текст з використанням різних стилів, вводити в текст рисунки і навіть вікна;
- `Toplevel` (вікно верхнього рівня), – окреме вікно, яке містить інші віджети.

Розглянемо детальніше синтаксис і методи створення віджетів, що найчастіше використовуються, а також їх основні властивості.

Кнопки (Button). Об'єкт-кнопка створюється викликом класу `Button` модуля `tkinter`. При цьому обов'язковим аргументом є лише породний віджет (наприклад, вікно верхнього рівня). Інші властивості можуть вказуватися при створенні кнопки або задаватися (змінюватися) пізніше. Об'єкт-кнопка має такий синтаксис:

```
змінна=button(породний_віджет[властивість=значення . .])
```

Покажемо на прикладі деякі властивості кнопок:

```
from tkinter import *
root = Tk()
but = Button(root,
              text="Це кнопка", #напис на кнопці
              width=30,height=5, #ширина і висота
              bg="blue",fg="red") #колір фону і напису
but.pack()
root.mainloop()
```

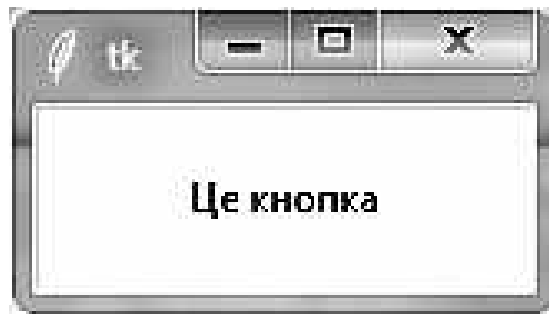


Рис. 3.2. Приклад керування властивостями кнопок при створенні об'єкта-кнопки

Тут `bg` і `fg` – скорочення від `background` (фон) і `foreground` (передній план). Ширина й висота вимірюються в знаках (кількість символів).

Мітки (Label) (або написи) – це достатньо прості віджети, що містять рядок (або декілька рядків) тексту і призначені в основному для інформування користувача:

```
lab = Label(root, text="Це мітка! \n 3 двох рядків.",
            font="Arial 18")
```

Однорядкове текстове поле (Entry) створюється викликом класу `Entry` модуля `tkinter`. У це поле користувач може ввести лише один рядок тексту:

```
ent = Entry(root,width=20,bd=3)
```

Тут `bd` – скорочення від `borderwidth` (ширина межі).

Створимо віджет для однорядкового текстового поля. Вигляд цього вікна показано на рис. 3.3.

```

from tkinter import *
root = Tk ()
ent = Entry(root,width=50,bd=5)
ent.pack()
root.mainloop()

```

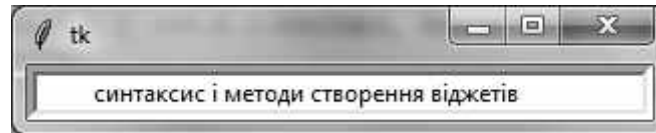


Рис. 3.3. Віджет Entry для однорядкового текстового поля

Багаторядкове текстове поле (Text) призначене для надання користувачеві можливості введення не одного рядка тексту, а значно більшої кількості рядків (рис. 3.4):

```

from tkinter import *
root = Tk()
tex = Text(root,width=40,
            font="Verdana 12",
            wrap=WORD)
tex.pack()
root.mainloop()

```

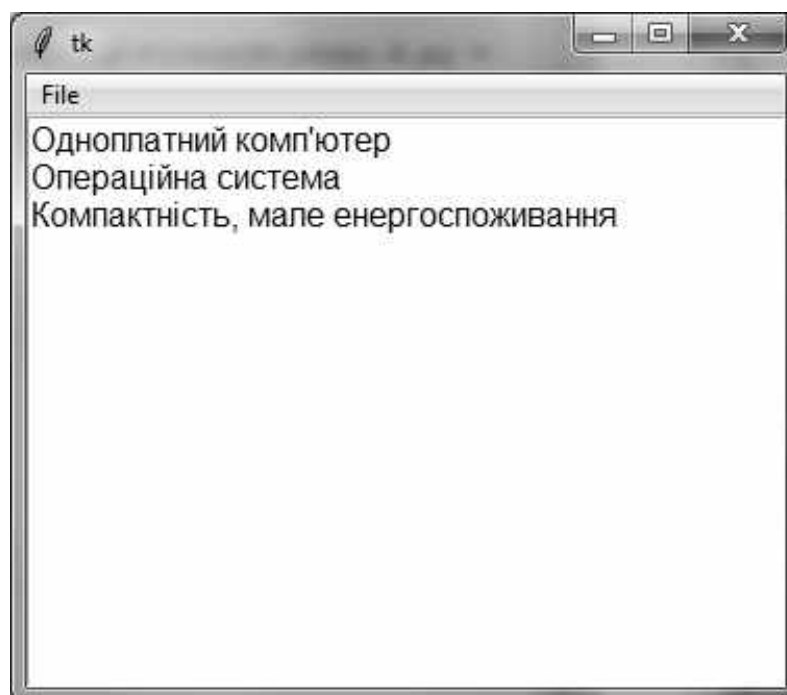


Рис. 3.4. Текстове поле (Text)

Остання властивість (`wrap`) залежно від свого значення дає можливість переносити текст, що вводиться користувачем або по символах, або по словах, або взагалі не переносити, поки користувач не натискуватиме Enter.

Радіокнопки (*Radiobutton*). Об'єкти-радіокнопки ніколи не використовують поодиночі. Їх використовують групами, при цьому в одній групі може бути «включена» лише одна кнопка:

```
from tkinter import *
root = Tk()
var=IntVar()
var.set(1)
rad0 = Radiobutton(root,text="Перша",
                    variable=var,value=0)
rad1 = Radiobutton(root,text="Друга",
                    variable=var,value=1)
rad2 = Radiobutton(root,text="Третя",
                    variable=var,value=2)
rad0.pack()
rad1.pack()
rad2.pack()
root.mainloop()
```

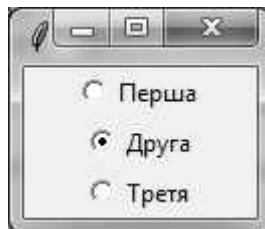


Рис. 3.5. Радіокнопка (*Radiobutton*) на три положення

Кожна група визначає значення однієї змінної, тобто якщо в прикладі буде вибрано радіокнопку `rad2`, то значенням змінної `var` буде 2. Спочатку також потрібно встановити значення змінної (вираз `var.set(1)` задає для змінної `var` значення 1).

Прапорці (*Checkbutton*). Об'єкт `checkbutton` призначений для вибору невзаємовиключних пунктів у вікні (у групі можна активувати один, два або більше прапорців або не активувати жодного). На відміну від радіокнопок, значення кожного прапорця прив'язується до своєї змінної, значення якої визначаються опціями `onvalue` (увімкнено) і `offvalue` (вимкнено) в описі прапорця. Результати роботи коду, наведеного нижче, показано на рис. 3.6:

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

```

c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root, text="Перший прапорець",
                    variable=c1, onvalue=1, offvalue=0)
che2 = Checkbutton(root, text="Другий прапорець",
                    variable=c2, onvalue=2, offvalue=0)
che1.pack()
che2.pack()
root.mainloop()

```

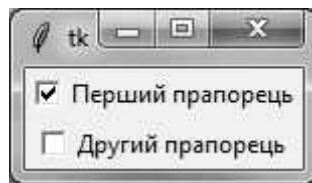


Рис. 3.6. Прапорці (Checkbutton) на два положення

Списки (Listbox). Виклик класу `Listbox` створює об'єкт, у якому користувач може вибрати один або декілька пунктів залежно від значення опції `selectmode`. У прикладі, наведеному нижче, значення `SINGLE` дає змогу вибирати лише один пункт зі списку (рис. 3.7):

```

from tkinter import *
root = Tk()
r = ['Linux', 'Python', 'Tk', 'Tkinter']
lis = Listbox(root, selectmode=SINGLE, height=4)
for i in r:
    lis.insert(END, i)
lis.pack()
root.mainloop()

```

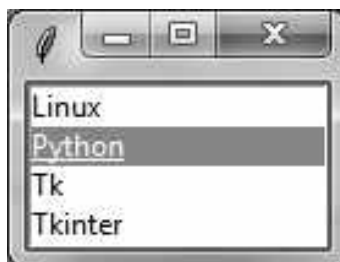


Рис. 3.7. Вибір одного пункту зі списку

Спочатку список (`Listbox`) є порожнім. За допомогою циклу `for` до нього додаються пункти зі списку (тип даних) `r`. Додавання відбувається за допомогою спеціального методу класу `Listbox` – `insert`. Цей метод приймає два параметри: куди додати і що додати.

Рамка (Frame) – інструмент для розміщення віджетів у вигляді груп усередині вікна, а також оформлення інтерфейсу:

```
from tkinter import *
root = Tk()
fra1 = Frame(root,width=500,height=100,bg="darkred")
fra2 = Frame(root,width=300,height=200,bg="green",bd=20)
fra3 = Frame(root,width=500,height=150,bg="darkblue")
fra1.pack()
fra2.pack()
fra3.pack()
root.mainloop()
```

У цьому прикладі (рис. 3.8) створюються три фрейми різного розміру. Властивість `bd` (скорочення від `boderwidth`) визначає відстані від краю рамки до віджетів, що в ній містяться (якщо вони є).

На фреймах також можна розміщувати віджети так, як на основному вікні (`root`). Тут текстове поле знаходиться на рамці `fra2`:

```
ent1 = Entry(fra2,width=20)
ent1.pack()
ent1 = Entry(fra2,width=20)
ent1.pack()
```

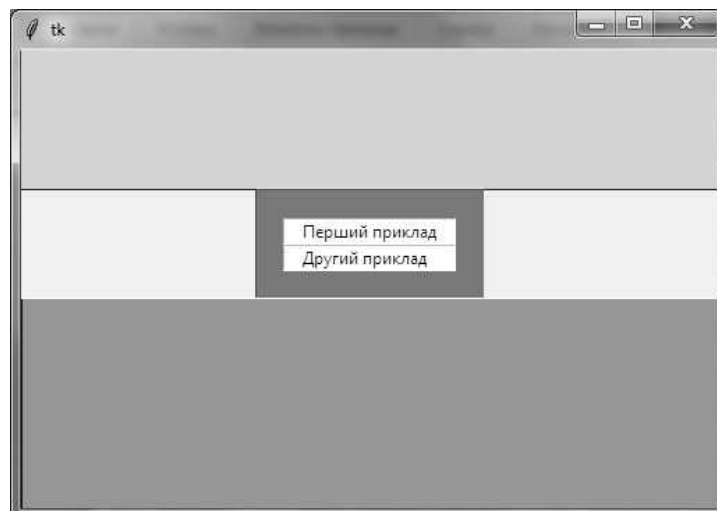


Рис. 3.8. Створення трьох різних фреймів

Шкала (Scale). Призначення шкали – це надання користувачеві вибору якогось значення з певного діапазону. Зовні шкала є горизонтальною або вертикальною смугою з розміткою, по якій можна пересувати повзун, здійснюючи тим самим вибір значення (рис. 3.9):

```
from tkinter import *
root = Tk()
fra3 = Frame(root,width=500,height=150,bg="darkblue")
sca1 = Scale(fra3,orient=HORIZONTAL,length=300,
```

```

        from_=0,to=100,tickinterval=10,resolution=5)
sca2 = Scale(root,orient=VERTICAL,length=400,
        from_=1,to=2,tickinterval=0.1,resolution=0.1)
fra3.pack()
sca1.pack()
sca2.pack()
root.mainloop()

```

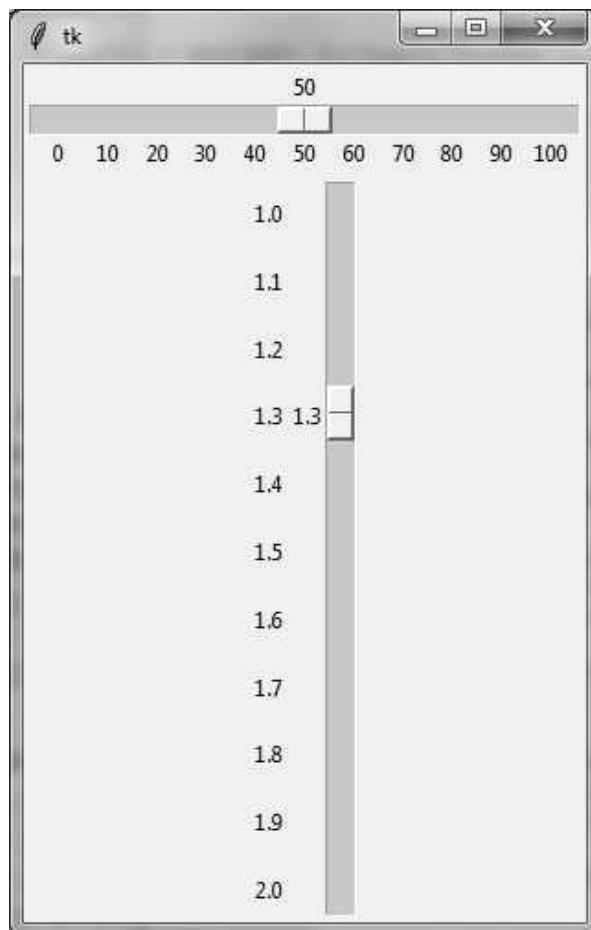


Рис. 3.9. Горизонтальна й вертикальна шкали (Scale)

Властивості шкали:

- `orient` – напрямок шкали;
- `length` – довжина шкали в пікселях;
- `from_` і `to` – значення, з якого шкала починається і яким закінчується (тобто діапазон значень);
- `tickinterval` – інтервал, через який відображуються мітки для шкали;
- `resolution` – мінімальна довжина відрізання, на яку користувач може пересунути повзун.

Смуга прокручування (Scrollbar). Цей віджет дає можливість прокручувати вміст іншого віджета (наприклад, текстового поля або списку). Прокручувати можна як по горизонталі, так і по вертикалі:

```

from tkinter import *
root = Tk()
tx = Text(root,width=40,height=3,font='14')
scr = Scrollbar(root,command=tx.yview)
tx.configure(yscrollcommand=scr.set)
tx.grid(row=0,column=0)
scr.grid(row=0,column=1)
root.mainloop()

```

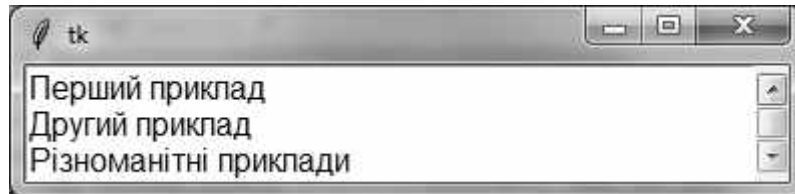


Рис. 3.10. Смуга прокручування (Scrollbar)

У цьому прикладі (рис. 3.10) спочатку створюється текстове поле (tx), потім смуга прокручування (scr), яка прив'язується за допомогою опції `command` до поля tx по вертикальній осі (`yview`). Далі поле tx змінюється (конфігурується) за допомогою методу `configure`: установлюється значення опції `yscrollcommand`. Тут використовується метод `grid`, що є іншим способом розташування віджета на вікні.

Вікно верхнього рівня (Toplevel). За допомогою класу `Toplevel` створюються дочірні вікна, на яких також можуть розташовуватися віджети. Слід зазначити, що при закритті головного вікна (або породного) вікно `Toplevel` також закривається. З іншого боку, закриття дочірнього вікна не приводить до закриття головного (рис. 3.11):

```

from tkinter import *
root = Tk()
win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")
win.title("Дочірнє вікно")
win.minsize(width=400,height=200)
root.mainloop()

```



Рис. 3.11. Вікно верхнього рівня й дочірнє вікно

Метод `title` визначає заголовок вікна. Метод `minsize` конфігурує мінімальний розмір вікна (є метод `maxsize`, що визначає максимальний розмір вікна). Якщо значення аргументів `minsize` буде таким самим, як у `maxsize`, то користувач не зможе змінювати розміри вікна.

3.3. Програмування подій у Tkinter

Тепер розглянемо, як елементам інтерфейсу додати функціональність, тобто можливості здійснювати з їх допомогою ті або інші дії.

Більшість консольних додатків виконуються при мінімальних зовнішніх діях. На відміну від них графічні додатки реагують на зовнішні події (клацання кнопкою мишки, натиснення клавіш на клавіатурі, змінення віджетів) і потім виконують закладену дію. Цей принцип роботи передбачає таку схему налаштування функціональності **GUI**: на віджет надходить зовнішня дія, за результатами якої виконується відповідна функція (дія). Зовнішню дію на графічний компонент називають подією. Подій – досить багато. Для простоти викладення використовуємо всього два види подій: клацання лівою кнопкою миші `()` і натиснення клавіші `Enter` `()`.

Одним зі способів скріплення віджета, події і функції (того, що має відбуватися після події) є використання методу `bind`. Синтаксис скріплення зображено на рис. 3.12.

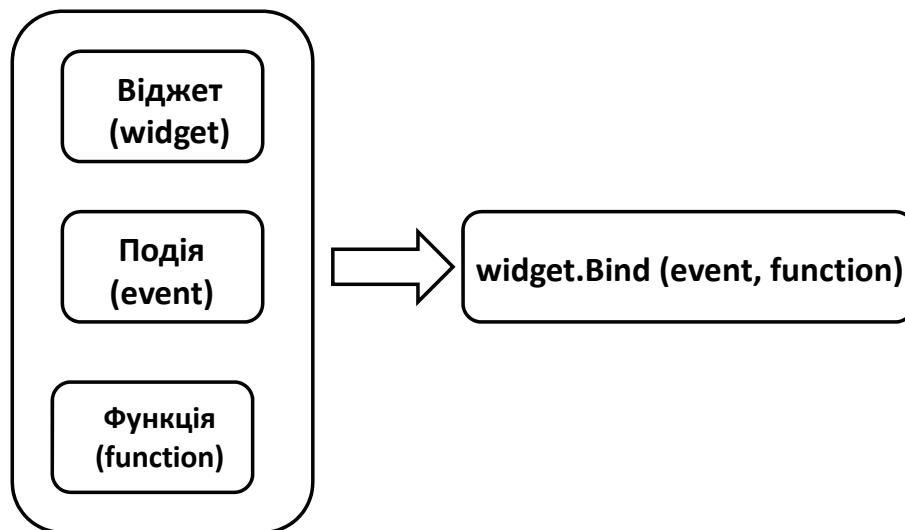


Рис. 3.12. Додавання функціональності графічному елементу за допомогою методу `Bind`

Наведемо приклад:

```
li = ["red", "green"]
def color(event):
    fra.configure(bg=li[0])
    li[0], li[1] = li[1], li[0]
def outgo(event):
    root.destroy()
from tkinter import *
```

```

root = Tk()
fra = Frame(root,width=200,height=100)
but = Button(root,text="Вихід")
fra.pack()
but.pack()
root.bind("<Return>",color)
but.bind("<Button-1>",outgo)
root.mainloop()

```

У цьому прикладі створюються два віджети: фрейм і кнопка (рис. 3.13).

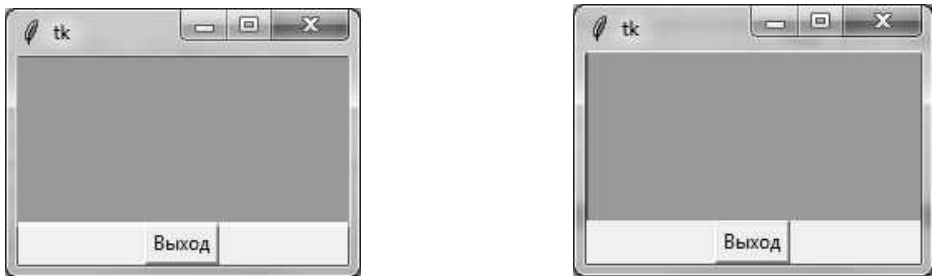


Рис. 3.13. Поєднання фрейма і кнопки

Додаток реагує на дві події: натиснення клавіші `Enter` у межах головного вікна і клацання лівою кнопкою миші по кнопці `but`. У першому випадку викликається функція `color`, у другому – `outgo`. Функція `color` змінює колір фону (`bg`) фрейма (`fra`) за допомогою методу `configure`, який призначений для змінення значення властивостей віджетів у процесі виконання скрипту. Як значення опції `bg` підставляється перший елемент списку. Потім у списку два елементи міняються місцями, щоб при наступному натисненні `Enter` колір фрейма знову змінився. У функції `outgo` викликається метод `destroy` відносно головного вікна. Цей метод призначений для «руйнування» віджета (вікно закриється).

Типи подій. Виділяють три основні типи подій, які виникають унаслідок такого:

- клацання кнопками мишки;
- натиснення клавіш на клавіатурі;
- змінення інших графічних об'єктів.

Класифікуємо ці події детальніше в табл. 3.2.

Таблица 3.2

Типи подій		
Мишка	Клавіатура	Вікна
Клацання кнопками мишки	Натиснення клавіші	Уведення даних
Уведення й виведення курсора за межі віджета	Натиснення комбінації клавіш	Змінення розмірів вікна
Рух мишки	Відтискання клавіші	Прокручування

Спосіб запису. При виклику методу `bind` подія передається як перший аргумент:

```
widget.bind(event, function)
```

Назва події береться в лапки, а також у знаки `< i >`. Подія описується за допомогою зарезервованих послідовностей ключових слів.

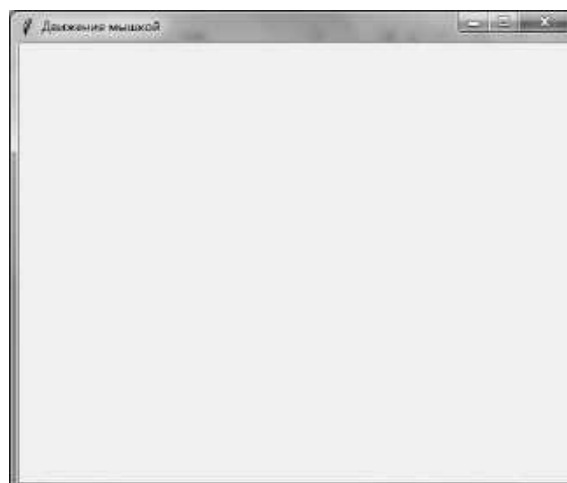
Події, що виконуються мишкою:

- `<Button-1>` – клацання лівою кнопкою мишки;
- `<Button-2>` – клацання середньою кнопкою мишки;
- `<Button-3>` – клацання правою кнопкою мишки;
- `<Double-button-1>` – подвійний клік лівою кнопкою мишки;
- `<Motion>` – рух мишки і т. д.

Наведемо приклад коду:

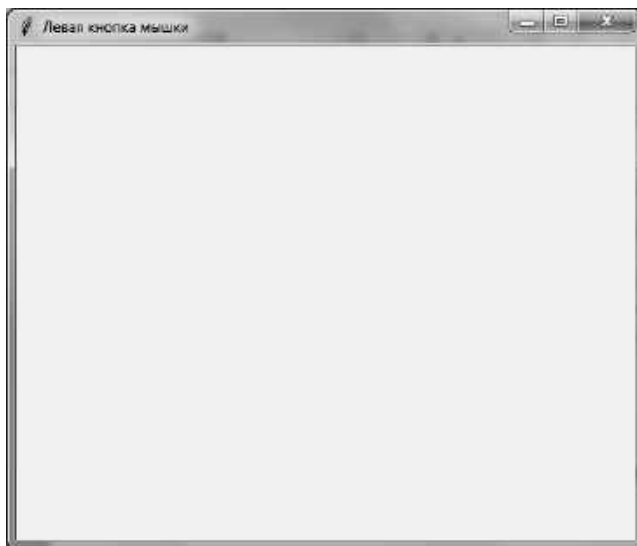
```
from tkinter import *
def b1(event):
    root.title("Ліва кнопка мишки")
def b3(event):
    root.title("Права кнопка мишки")
def move(event):
    root.title("Рух мишки")
root = Tk()
root.minsize(width = 500, height=400)
root.bind('<Button-1>',b1)
root.bind('<Button-3>',b3)
root.bind('<Motion>',move)
root.mainloop()
```

У цій програмі змінюється напис у заголовку головного вікна залежно від того, чи рухається мишка (рис. 3.14, а), чи клацають лівою (рис. 3.14, б) або правою (рис. 3.14, в) кнопкою мишки.

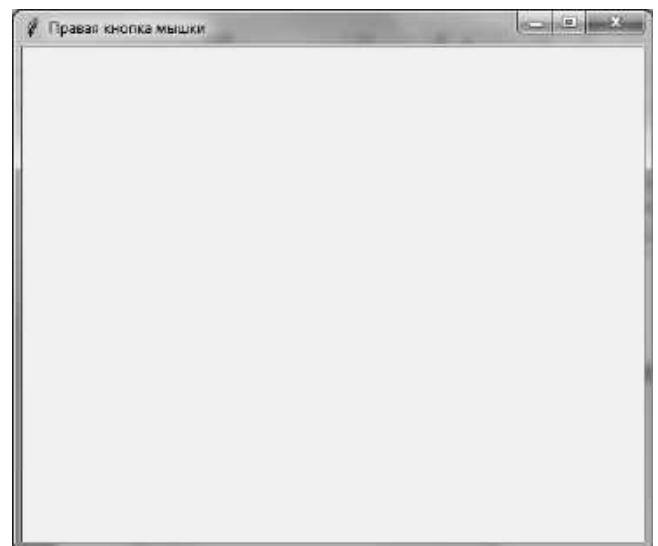


а

Рис. 3.14. Зміна найменування вікна при русі мишки



б



в

Рис. 3.14. Закінчення

Події, що виконуються за допомогою клавіатури:

- Буквені клавіші можна записувати без кутових дужок (наприклад, 'L').
- Для неалфавітних клавіш існують спеціальні зарезервовані слова:
 - * <Return> – натиснення клавіші Enter;
 - * <space> – пропуск і т. д.
- Поєднання клавіш пишуться через тире, наприклад: * <Control-shift> – одночасне натиснення клавіш Ctrl і Shift.

Наведемо приклад:

```
from tkinter import *
def exit_(event):
    root.destroy()
def caption(event):
    t = ent.get()
    lbl.configure(text = t)
root = Tk()
ent = Entry(root, width = 40)
lbl = Label(root, width = 80)
ent.pack()
lbl.pack()
ent.bind('<Return>',caption)
root.bind('<Control-z>',exit_)
root.mainloop()
```

У цьому прикладі при натисненні клавіші Enter у межах текстового рядка (ent) викликається функція caption, яка поміщає символи з текстового рядка (ent) у мітку (lbl). Унаслідок натиснення комбінації клавіш Ctrl + z головне вікно закривається (рис. 3.15).

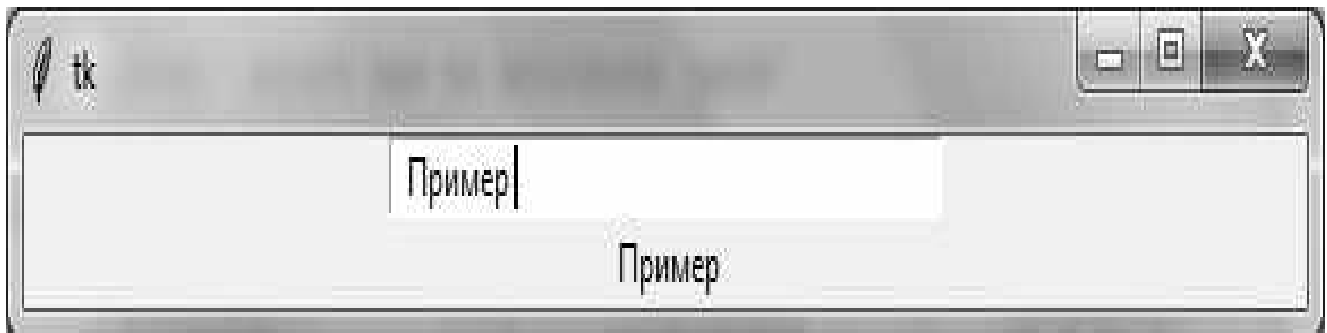


Рис. 3.15. Перенесення символів текстового рядка в мітку lbl

3.4. Змінні Tkinter

Бібліотека **Tkinter** містить спеціальні класи, об'єкти яких виконують роль змінних для зберігання значень стану різних віджетів. Зі зміненням значення такої змінної змінюється і властивість віджета і, навпаки, зі зміненням властивості віджета змінюється значення асоційованої змінної.

Використовуються такі класи Tkinter, що призначені для оброблення даних різних типів:

- `Stringvar()` – рядків;
- `Intvar()` – цілих чисел;
- `Doublevar()` – дробових чисел;
- `Booleanvar()` – булевих значень (`true` і `false`).

Наведемо приклад. При створенні групи радіокнопок (див. рис. 3.5) раніше використовувалася змінна-об'єкт типу `Intvar()` :

```
var=IntVar()
var.set(1)
rad0 = Radiobutton(root,text="Перша",variable=var,value=0)
rad1 = Radiobutton(root,text="Друга",variable=var,value=1)
rad2 = Radiobutton(root,text="Третя",variable=var,value=2)
```

Тут створюється об'єкт класу `Intvar` і зв'язується із змінною `var`. За допомогою методу `set` установлюється початкове значення, яке дорівнює 1. Три радіокнопки належать до однієї групи: про це свідчить однакове значення опції `variable`, призначеної для зв'язування змінної Tkinter з радіокнопкою. Опція `value` визначає значення, яке буде передано змінною, якщо ця кнопка буде в стані «включено». Якщо в процесі виконання скрипту значення змінної `var` буде змінено, то це відображається на групі кнопок. Наприклад, це робиться в другому рядку коду: включена кнопка `rad1`.

Метод `set` дає змогу встановлювати значення змінних. Метод `get`, навпаки, дає змогу отримувати значення для подальшого їх використання:

```

def display(event):
    v = var.get()
    if v == 0:
        print ("Включена перша кнопка")
    elif v == 1:
        print ("Включена друга кнопка")
    elif v == 2:
        print ("Включена третя кнопка")
but = Button(root,text="Одержати значення")
but.bind('<Button-1>',display)

```

Унаслідок виклику функції `display` у змінну `v` «записується» значення, зв'язане у цій момент зі змінною `var`. Щоб отримати значення змінної `var`, використовується метод `get` (другий рядок коду).

Окрім опції `variable`, що зв'язує віджет зі змінною-об'єктом Tkinter (`Intvar`, `Stringvar` та ін.), багато віджетів мають опцію `textvariable`, яка визначає текст-вміст або текст-напис віджета. Не зважаючи на те, що «текстова властивість» може бути встановлена для віджета і змінена в процесі виконання коду без використання асоційованих змінних, інколи такий спосіб змінення є зручнішим (рис. 3.16):

```

from tkinter import *
root = Tk()
v = StringVar()
ent1 = Entry (root, textvariable =
v,bg="black",fg="white")
ent2 = Entry(root, textvariable = v)
ent1.pack()
ent2.pack()
root.mainloop()

```

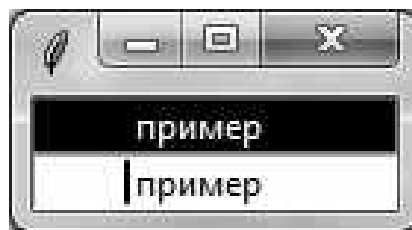


Рис. 3.16. Використання опції `textvariable`

У цьому прикладі вміст одного текстового поля відразу відображується в іншому, оскільки обидва поля зв'язані з однією й тією ж змінною `v`.

3.5. Об'єкт Меню (Menu) у GUI

Меню – це об'єкт, який є практично у всіх додатках, призначених для користувача. Знаходиться меню під рядком заголовка і має вигляд випадних списків під словами. Кожний такий список може містити інший список,

вкладений у нього. Кожний пункт списку є командою, що запускає яку-небудь дію або відкриває діалогове вікно.

Розглянемо приклад програми створення меню в Tkinter (рис. 3.17):

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

m = Menu(root) # створюється об'єкт Меню на головному
вікні
root.config(menu=m) # вікно конфігурується з указівкою ме-
ню для нього

fm = Menu(m) # створюється пункт меню з розміщенням на ос-
новному меню (m)
m.add_cascade(label="File", menu=fm) # пункт розташовується
на основному меню (m)
fm.add_command(label="Open...") # формується список команд
пункту меню
fm.add_command(label="New")
fm.add_command(label="Save...")
fm.add_command(label="Exit")

hm = Menu(m) # другий пункт меню
m.add_cascade(label="Help", menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")

root.mainloop()
```

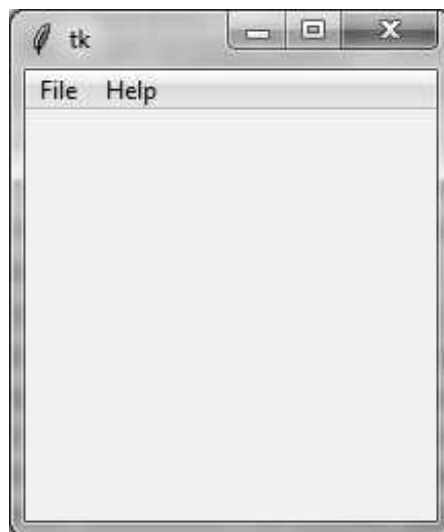


Рис. 3.17. Приклад створення випадного меню

Метод `add_cascade` додає новий пункт у меню, який указується як значення опції `menu`.

Метод `add_command` додає нову команду в пункт меню. Одна з опцій цього методу (у попередньому прикладі її поки немає) – `command` – зв'язує цю команду з функцією-обробником.

Можна створити вкладене меню. Для цього створюється ще одне меню й за допомогою `add_cascade` прив'язується до породного пункту:

```
nfm = Menu(fm)
fm.add_cascade(label="Import", menu=nfm)
nfm.add_command(label="Image")
    nfm.add_command(label="Text")
```

Прив'язка функцій до меню. Кожна команда меню зазвичай має бути зв'язана зі своєю функцією, що виконує ті або інші дії. Зв'язок установлюється за допомогою опції `command` методу `add_command`. Функція-обробник до цього має бути визначена.

Для прикладу наведемо виправлені рядки додавання команд «About», «New» і «Exit», а також функції, що викликаються, коли користувач клацає лівою кнопкою мишки по відповідних пунктах підміню:

```
def new_win():
    win = Toplevel(root)

def close_win():
    root.destroy()

def about():
    win = Toplevel(root)
    lab = Label(win, text="Це просто програма-тест \n меню в Tkinter")
    lab.pack()

...
fm.add_command(label="New", command=new_win)
...
fm.add_command(label="Exit", command=close_win)

...
hm.add_command(label="About", command=about)
```

Створимо додаток з меню, що містить два пункти: `Color` і `Size` (рис. 3.18). Пункт `Color` повинен містити три команди (`Red`, `Green` і `Blue`), що змінюють колір рамки на головному вікні. Пункт `Size` повинен містити дві команди (`300x300` і `400x200`), що змінюють розмір рамки:

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

```

def colorR():
    fra.config(bg="Red")
def colorG():
    fra.config(bg="Green")
def colorB():
    fra.config(bg="Blue")
def square():
    fra.config(width=500)
    fra.config(height=500)
def rectangle():
    fra.config(width=700)
    fra.config(height=400)

fra = Frame(root,width=300,height=100,bg="Black")
fra.pack()

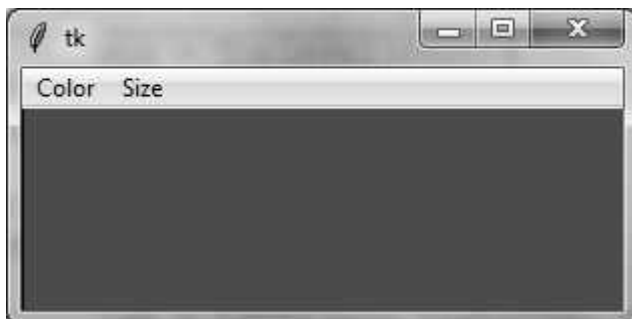
m = Menu(root)
root.config(menu=m)

cm = Menu(m)
m.add_cascade(label="Color",menu=cm)
cm.add_command(label="Red",command=colorR)
cm.add_command(label="Green",command=colorG)
cm.add_command(label="Blue",command=colorB)

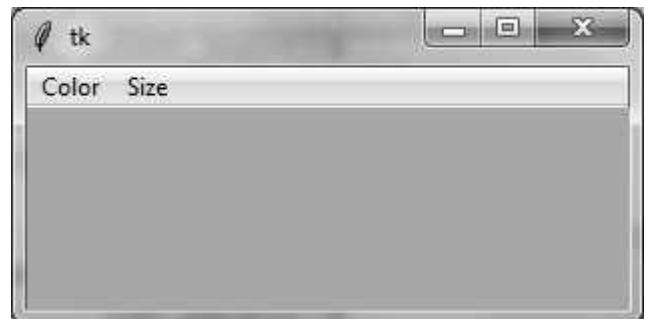
sm = Menu(m)
m.add_cascade(label="Size",menu=sm)
sm.add_command(label="500x500",command=square)
sm.add_command(label="700x400",command=rectangle)

root.mainloop()

```

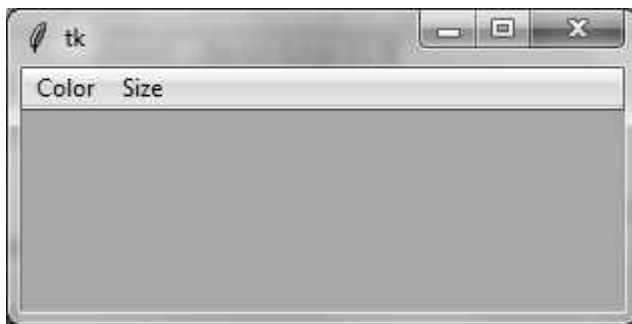


a



б

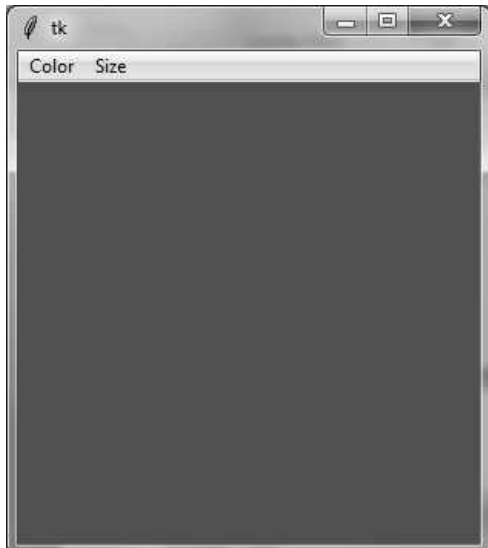
Рис. 3.18. Додаток з меню Color і Size



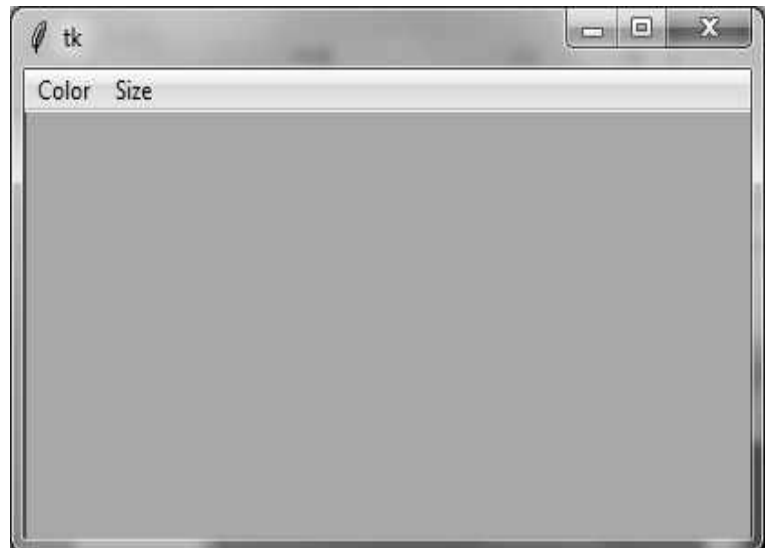
б



з



д



е

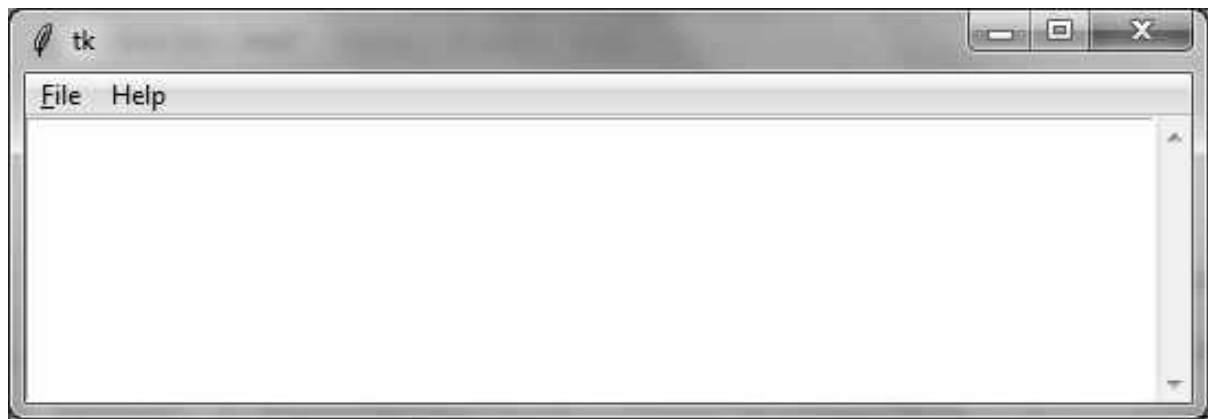
Рис. 3.18. Закінчення

3.6. Діалогові вікна в Tkinter

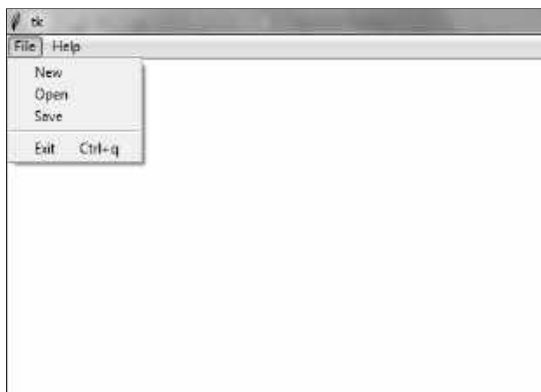
Діалогові вікна як елементи графічного інтерфейсу призначені для виведення повідомлень для користувача, отримання від нього якої-небудь інформації, а також керування. Слід зазначити, що існує велика кількість різних діалогових вікон. Щоб уникнути збільшення обсягу публікації, у цьому підрозділі наведемо лише характерні приклади.

Розглянемо, як запрограмувати за допомогою **Tkinter** виклик діалогових вікон відкриття й зберігання файлів і роботу з ними на прикладі текстового редактора, який має працювати з візуальними елементами **Text**, **Scrollbar**, **Menu** і діалоговими вікнами відкриття й зберігання файлів.

Вигляд вихідного вікна програми показано на рис. 3.19, а. У робочій частині вводиться текст, який за допомогою меню **File** можна зберегти у файл або завантажити з файлу. Меню **File** зображено на рис. 3.19, б. Пункт меню **File** $\Rightarrow$ **Save** зберігає текст у файл, а команда **File** $\Rightarrow$ **Open** завантажує текстовий файл у вікно програми. Пункт меню **File** $\Rightarrow$ **New** очищає вікно редактора, а команда **Exit** завершує роботу програми. Меню **Help** $\Rightarrow$ **About** (рис. 3.19, в) виводить вікно повідомлення.



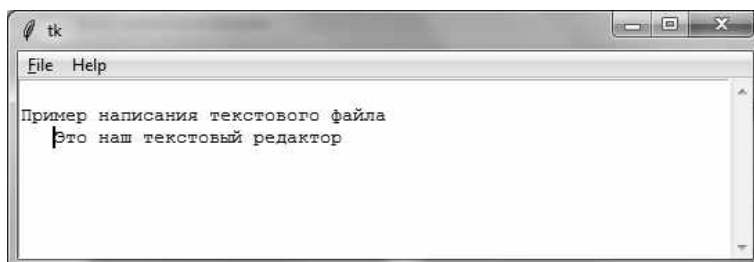
а



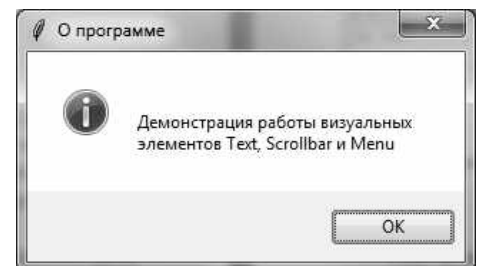
б



в



г



д

3.19. Вигляд вікон текстового редактора

Побудову коду програми розглядатимемо й пояснюватимемо по частинах і виділятимемо рядки коментарів.

Починається код імпортуванням усіх функцій модуля *tkinter* і трьох додаткових функцій *askopenfile*, *asksaveasfile* і *showinfo* з його підмодулів, які активують вікна зберігання й відкриття редагованих файлів (рис. 3.20).

Потім формуються код функцій і меню, що викликаються командами. Функція *about()*, що викликається з пункту *Help* $\Rightarrow$ *About*, створює вікно повідомлень *messagebox.showinfo(...)*, описане раніше. Функції *openfile()* і *savefile()* починаються з виклику методів *filedialog.askopenfile()* і *filedialog.asksaveasfile()*, які викликають діалогові вікна відкриття і зберігання файлів.

`asksaveasfile()` повертають посилання на файлові об'єкти `fopen` і `fsave`. Вони мають методи читання `str = fopen.read()` і записування `fsave.write(str)` файлів. При використанні у діалогових вікнах кнопки «Відміна» методи `askopenfile()` і `asksaveasfile()` повертають `None`, що і має бути проаналізоване до того, як в коді відбуватиметься читання або записування файла. Функція `fopen.read()` повертає рядок, який містить прочитаний з файла текст, а функція `fsave.write(str)` як аргумент приймає рядок, який буде записаний у текстовий файл. Цей рядок вставляється у віджет `Text` у коді функції `openfile()` і читається з нього в коді функції `savefile()`. Ці інструкції, що використовуються у функціях меню, описуються таким кодом:

```
from tkinter import *
from tkinter.filedialog import askopenfile, asksaveasfile
from tkinter.messagebox import showinfo
def about():
    msg="""
    Демонстрація роботи візуальних
    елементів Text, Scrollbar і Menu"""
    showinfo('Про програму',msg)
def newfile(): # очищення вікна без запиту про збереження
    text.delete(0.0, END)

def openfile():
    fopen=askopenfile( mode='r', defaultextension=".txt",
        filetypes =(("Text files", "*.txt"),("All files",
        "**.*")) )
    if fopen==None: return
    str=fopen.read()
    text.delete(0.0, END)
    text.insert(END,str)

def savefile():
    fsave=asksaveasfile( mode='w',defaultextension=".txt",
        filetypes =(("Text files", "*.txt"),("All files",
        "**.*")) )
    if fsave==None: return
    str=text.get(0.0,END)
    fsave.write(str)
    fsave.close()
```

Розглянемо далі віджети `Text` і `Scrollbar`, розташовані у вікні програми. Елемент `Text` дає змогу користувачеві ввести будь-яку кількість рядків тексту. Опишемо деякі його можливості. Властивість `wrap` керує відображенням довгих рядків. Наприклад, при використанні опції

`wrap= WORD` текст усередині віджета переноситиметься у вигляді слів. При значенні опції за умовчанням `wrap = char` рядок, досягаючи правого краю, переноситиметься (розриватиметься) на будь-якому символі. Для змінення шрифту призначена опція `font`, яку можна застосовувати до уривків тексту. Такі уривки, які називають тегами, конструюються методом `tag_add()` об'єкта `Text`. Наприклад, інструкція `text.tag_add('first','3.0','4.11')` створює тег з ім'ям `'first'`, який починається в третьому рядку першим символом (`'3.0'`) і закінчується в четвертому рядку дванадцятим символом (`'4.11'`). Перший аргумент `'first'` методу `tag_add` – ім'я тега. Далі йдуть індекси початкового і кінцевого символів тексту, які вказують область, до якої тег прикріплюється. Місце вставки записується у вигляді рядка `'line.col'`, де `line` – це рядок, а `col` – стовпець. При цьому нумерація рядків починається з одиниці, а стовпців – з нуля. Наприклад, перший символ другого рядка має індекс `'2.0'`, а останній символ п'ятого рядка – `'5.end'`. Різні області тексту можуть бути помічені однаковим тегом (загальним ім'ям).

Зміна властивостей поміченого тега виконується методом `tag_config(ім'я_тега[,опції])`. Наприклад, інструкція `text.tag_config('first',foreground='red',font=('times',14))` установлює в елементі `Text` в області, відміченій тегом `«first»`, колір фону і шрифт тексту.

Для читання тексту з елемента `Text` зазвичай використовується метод `get(startindex[,endindex])`. Його аргументи є рядками вигляду `«line.col»`, що подають індекси положення початку і кінця тексту, який читається. Значення `«end»` другого аргумента вказує на те, що текст читається до кінця. Метод повертає текстовий рядок.

Метод `insert(„line.col“,string)` вставляє рядок `string` у вказану позицію. Метод `delete(startindex[,endindex])` видаляє частину тексту. Наприклад, інструкція `text.delete(0.0,end)`, використана нами у функції меню `newfile()`, видаляє з віджета `text` весь текст.

Щоб в елементі `Text` прокручувати текст, потрібно використовувати інший віджет – `Scrollbar` (смуга прокручування), який дає можливість користувачеві "прокрутити" інший віджет (текстове поле, список або полотно (`Canvas`)). Для прив'язки вертикальної смуги прокручування до текстового поля виконуються дві дії:

- подання елемента `Text` змінюється, цей елемент інформує **`Scrollbar`** (смугу), в елементі **`Text`** встановлюється опція `yscrollcommand= scrollbar.set`;
- в елементі **`Scrollbar`** установлюється опція `command=text.yview` (коло візуального прокручування) з використанням методу `set()`. На-

впаки, коли користувач керує елементом *Scrollbar*, викликається метод `yview()` віджета *Text*.

Додавання горизонтальної смуги прокручування виконується аналогічно, лише використовуються опція `xscrollcommand` і метод `xview`.

Інструкції щодо створення вікна програми описуються таким кодом:

```
root = Tk()
text = Text(root, height=3, width=40)
text.pack(side='left', fill='both', expand='yes')
scrollbar = Scrollbar(root, command=text.yview)
text.configure(yscrollcommand=scrollbar.set)
scrollbar.pack(side='right', fill=Y)
```

Після проектування основного вікна створюємо рядок меню. Для цього призначений візуальний елемент *Menu*. Він використовується для створення меню головного вікна, випадних і спливних меню.

Меню головного вікна знаходиться під рядком заголовка. Воно складається з випадних меню, які містять списки пунктів меню, а також інші вкладені випадні меню. Кожний пункт меню є командою, що виконує яку-небудь дію.

Для створення об'єкта меню верхнього рівня використовуємо інструкцію `menubar = menu(root)`. Аргументом конструктора *Menu* є ідентифікатор `root` породного вікна. Конструктор створює об'єкт `menubar`, який інструкцією `root.config(menu=menubar)` додається в головне вікно `root`. Потім, використовуючи той же конструктор *Menu*, створюємо об'єкт випадного меню `filemenu`. Тепер функції *Menu* як перший аргумент передаються на ідентифікатор меню головного вікна, а опція `tearoff=0` відключає відображення пунктирної лінії на початку списку пунктів меню.

```
filemenu = Menu(menubar, tearoff=0)
```

Ця інструкція створює об'єкт меню, який перетворюється на випадне меню після його додавання в рядок меню `menubar` методом `menubar.add_cascade(...)`. Каскадне меню, що додається, указується в опції `menu=filemenu`. У цьому прикладі інструкція створення випадного меню *File* має такий вигляд:

```
menubar.add_cascade(label="File", underline=0, menu=filemenu)
```

Опція `label` задає назву меню, а опція `underline` вказує в ньому номер символу, який буде підкреслений при натисненні клавіші ALT (клавіші доступу). Подальші інструкції `filemenu.add_command(...)` додають команди у випадне меню *File*. Опція `command` методу `add_command` зв'язує команди меню з функціями-обробниками. Опція `accelerator` додає в пункт меню рядок, у якому зазвичай указується швидка комбінація

клавіш. Додається лише рядок, а саму функцію, що викликається цією комбінацією клавіш, слід підключити окремою інструкцією, використовуючи метод `bind()` об'єкта `root` вікна верхнього рівня.

Інструкція `filemenu.add_separator()` додає розмежувальну лінію у випадному меню **File**.

Інше випадне меню **Help** створюється аналогічно.

Таким чином, для створення рядка меню необхідно створити екземпляр класу **Menu**, а потім використовувати його як значення атрибута `menu` головного вікна. Після цього можна створювати інші меню й підключати їх в рядок меню, використовуючи метод `add_cascade()`. Рядки меню описуються таким кодом:

```
menubar = Menu(root)
root.config(menu=menubar)
```

Фрагмент коду описує меню `file`:

```
filemenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="File", underline=0,
menu=filemenu)
filemenu.add_command(label="New", underline=0, command=newfi
le)
filemenu.add_command(label="Open",
underline=0, command=openfile)
filemenu.add_command(label="Save",
underline=0, command=savefile)
filemenu.add_separator()
filemenu.add_command(label="Exit", underline=1,
accelertor
="Ctrl+q", command=root.destroy )
```

Код меню `Help` має вигляд:

```
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="About...", command=about)
menubar.add_cascade(label="Help", menu=helpmenu)
```

Сформуємо тепер загальний програмний код поставленого завдання – синтезу алгоритму діалогових вікон відкриття й зберігання файлів і роботи з ними на прикладі текстового редактора:

```
from tkinter import *
from tkinter.filedialog import askopenfile, asksaveasfile
from tkinter.messagebox import showinfo
def about():
    msg="""
    Демонстрація роботи візуальних
```

```

        елементів Text, Scrollbar і Menu"""
        showinfo('Про програму',msg)
def newfile(): # очищення вікна без запиту про збереження
    text.delete(0.0, END)

def openfile():
    fopen=askopenfile( mode='r', defaulttextextension=".txt",
        filetypes =(("Text files", "*.txt"),("All files",
        "*.*")) )
    if fopen==None: return
    str=fopen.read()
    text.delete(0.0, END)
    text.insert(END,str)

def savefile():
    fsave=asksaveasfile( mode='w',defaulttextextension=".txt",
        filetypes =(("Text files", "*.txt"),("All files",
        "*.*")) )
    if fsave==None: return
    str=text.get(0.0,END)
    fsave.write(str)
    fsave.close()

# Створення вікна програми
root = Tk()
text = Text(root, height=3, width=40)
text.pack(side='left', fill='both', expand='yes')
scrollbar = Scrollbar(root, command=text.yview)
text.configure(yscrollcommand=scrollbar.set)
scrollbar.pack(side='right', fill=Y)

# Рядок меню
menubar = Menu(root)
root.config(menu=menubar)

# Меню File
filemenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="File", underline=0,
menu=filemenu)
filemenu.add_command(label="New",underline=0,command=newfi
le)
filemenu.add_command(label="Open", under-
line=0,command=openfile)
filemenu.add_command(label="Save", under-
line=0,command=savefile)

```

```

filemenu.add_separator()
filemenu.add_command(label="Exit",underline=1,
                      accelerator="Ctrl+q", com-
mand=root.destroy )

# Меню Help
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="About...", command=about)
menubar.add_cascade(label="Help", menu=helpmenu)

# -----
root.bind("<Control-q>", lambda event: root.destroy())
root.mainloop()

```

Наведений приклад наочно демонструє можливості побудови за допомогою бібліотеки **Tkinter** розвинених інтерактивних інтерфейсів для роботи з різного роду програмними додатками. Проте модуль `tkinter` дає можливість створювати додатки з віконним інтерфейсом, що містить векторну графіку. Зокрема, використання графічних функцій віджета `Canvas` дає змогу будувати графіки функцій і вставляти в інтерфейс потрібні зображення. Проте можливості модуля `matplotlib` є значно досконалішими, і бажано поєднувати віконні додатки `tkinter` з графічними функціями `matplotlib`. Цю можливість реалізовано за допомогою «складеного» полотна `FigureCanvasTkagg` – спеціального класу, який успадковує багато методів віджета `Canvas` і додає можливості використання графічних функцій модуля `matplotlib`.

Оскільки ці матеріали виходять за межі посібника, рекомендуємо вивчити їх практичну реалізацію самостійно.

ВИСНОВКИ

Цим навчальним посібником ми починаємо серію публікацій з проблеми створення мобільних систем технічного зору, в основному орієнтованих на використання в робототехніці, системах керування безпілотними літальними апаратами, автомобілями та іншими транспортними засобами.

Під час проектування подібного роду систем виникає велика кількість питань стосовно виявлення рухомих об'єктів, оцінювання параметрів їх руху і створення алгоритмів раціонального керування в області навігації транспортних засобів за відеоспостереженням.

При цьому насамперед розробникам проектів необхідно вирішити ключові питання вибору мікрокомп'ютера і засобів програмування, оскільки це надалі визначить архітектуру апаратно-програмних засобів, ефективність їх використання, можливості оброблення інформації в реальному масштабі часу й перспективи подальшого поліпшення технічних характеристик цих систем.

Зважаючи на велику кількість питань з проектування мобільних систем технічного зору, автори визнали доцільним поділити матеріал навчального посібника на окремі частини. У першій частині розглянуто завдання з вибору мікрокомп'ютера і засобів програмування. Причому вдалося відобразити лише основні його аспекти. Уважаємо, що це не обмежить можливості поглибленого вивчення проблеми.

У наступній частині цієї роботи планується дослідити питання оброблення зображень і відеоданих у системах технічного зору з використанням різних алгоритмів цифрового оброблення, розроблених як самостійно, так і з допомогою бібліотек оброблення зображень **OpenCV**.

Надалі планується опублікувати цикл лабораторних робіт, які поза сумнівом, допоможуть закріпити отримані знання і будуть корисними для самостійного проектування систем технічного зору.

БІБЛІОГРАФІЧНИЙ СПИСОК

Гридин, В. Н. Адаптивные системы технического зрения / В. Н. Гридин, В. С. Титов, М. Н. Труфанов. – СПб. : Наука, 2009. – 442 с.

Лутц, М. Программирование на Python. Т. 1 / М. Лутц. – СПб. : Символ-Плюс, 2011. – 992 с.

Магда, Ю. С. Raspberry Pi. Руководство по настройке и применению / Ю. С. Магда. – СПб. : ДМК Пресс, 2014. – 188 с.

Методы компьютерной обработки изображений / под ред. В. А. Сойфера. – М. : Физматлит, 2003. – 784 с.

Обработка и анализ изображений в задачах машинного зрения : курс лекций и практ. занятий / Ю. В. Визильтер и др. – М. : Физматкнига, 2010. – 672 с.

Официальный сайт документации для работы с компьютером Raspberry Pi [Электронный ресурс]. – Режим доступа: <https://www.raspberrypi.org>.

Официальный сайт языка Python [Электронный ресурс]. – Режим доступа: <http://python.org/>.

Официальный сайт разработчиков библиотеки OpenCV [Электронный ресурс]. – Режим доступа: <http://opencv.org>.

Прохоренок, Н. А. Python. Самое необходимое / Н. А. Прохоренок. – СПб. : БХВ-Петербург, 2011. – 416 с.

Прэтт, У. Цифровая обработка изображений / У. Прэтт. – М. : Мир, 1982. – 791 с.

Саммерфилд, М. Программирование на Python 3. Подробное руководство. – СПб. : Символ-Плюс, 2009. – 608 с.

Сергиенко, А. Б. Цифровая обработка сигналов / А. Б. Сергиенко. – СПб. : Питер, 2005. – 587 с.

Техническое зрение роботов / В. И. Мошкин и др. – М. : Машиностроение, 1990. – 272 с.

Хахаев, И. А. Практикум по алгоритмизации и программированию на Python / И. А. Хахаев. – М. : Альт Линукс, 2010. – 126 с.

Хуанг, Т. С. Быстрые алгоритмы в цифровой обработке изображений / Т. С. Хуанг, Дж.-О. Эклунд, Г. Дж. Нуссбаумер / под ред. Т. С. Хуанга. – М. : Радио и связь, 1984. – 224 с.

Федоров, Д. Ю. Основы программирования на примере языка Python : учеб. пособие / Д. Ю. Федоров. – СПб. : СПбГЭУ, 2016. – 176 с.

Форсайт, Дэвид А. Компьютерное зрение. Современный подход / Дэвид А. Форсайт, Жан Понс. – М. : Вильямс, 2004. – 928 с.

ЗМІСТ

ВСТУП	3
1. ВИКОРИСТАННЯ МІКРОКОМП'ЮТЕРІВ У МОБІЛЬНИХ СИСТЕМАХ	
ТЕХНІЧНОГО ЗОРУ	4
1.1. Вибір моделі одноплатового мікрокомп'ютера.....	4
1.2. Загальні відомості про одноплатовий комп'ютер Raspberry Pi	6
1.3. Операційні системи для Raspberry Pi і програми NOOBS.....	11
1.4. Установлення операційної системи Raspbian для комп'ютера Raspberry Pi	13
1.5. Первинні налаштування після встановлення ОС Raspbian	15
2. ЗАСОБИ ПРОГРАМУВАННЯ МОБІЛЬНИХ СИСТЕМ ТЕХНІЧНОГО ЗОРУ	23
2.1. Вибір мови програмування	23
2.2. Установлення програми-інтерпретатора Python.....	23
2.2.1. Установлення Python 3.6.1 для операційної системи Windows 7	24
2.2.2. Особливості встановлення програми Python 3.6.1 для ОС Raspbian на комп'ютер Raspberry Pi.....	27
2.3. Програмування в інтерактивному режимі.....	28
2.4. Зберігання програми в окремому файлі	33
2.5. Змінні в мові Python	34
2.6. Модулі	34
2.7. Підключення модулів і бібліотек	35
2.7.1. Підключення модулів і бібліотек Python для Windows.....	35
2.7.2. Підключення модулів і бібліотек Python для платформи Raspberry Pi	39
2.7.3. Імпорт модуля.....	40
2.7.4. Стандартні модулі	41
2.7.5. Перегляд установлених модулів	42
2.7.6. Імпорт функцій модулів	43

2.8. Використання модуля math	43
2.9. Формування масивів і матриць	45
2.10. Матричні обчислення	48
2.11. Особливості використання операторів циклу в мові Python	50
2.12. Використання бібліотеки Matplotlib	52
3. СТВОРЕННЯ ІНТЕРАКТИВНИХ ІНТЕРФЕЙСІВ	70
3.1. Основні етапи роботи з віджетами при створенні інтерфейсу	70
3.2. Графічні об'єкти та їх властивості	74
3.3. Програмування подій у Tkinter	82
3.4. Змінні Tkinter	86
3.5. Об'єкт Меню (Menu) у GUI	87
3.6. Діалогові вікна в Tkinter	91
ВИСНОВКИ	100
БІБЛІОГРАФІЧНИЙ СПИСОК	101

Навчальне видання

**Краснов Леонід Олександрович
Дергачов Костянтин Юрійович
Багінський Сергій Вікторович**

**ОСНОВИ ПОБУДОВИ СУЧАСНИХ МОБІЛЬНИХ СИСТЕМ
ТЕХНІЧНОГО ЗОРУ
Частина 1
КОМП'ЮТЕРИ І ЗАСОБИ ПРОГРАМУВАННЯ**

Редактор Т. О. Іващенко

Зв. план, 2018

Підписано до друку 01.02.2018

Формат 60x84 1/16. Папір офс. № 2. Офс. друк

Ум. друк. арк. 5,7. Обл.-вид. арк. 6,5. Наклад 100 пр.

Замовлення 16. Ціна вільна

Видавець і виготовлювач

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

61070, Харків-70, вул. Чкалова, 17

<http://www.khai.edu>

Видавничий центр «ХАІ»

61070, Харків-70, вул. Чкалова, 17

izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.02.2001