Запуск програм на Python з командного рядка

Командні рядки служать для запуску програм на Python з командної оболонки системи у наступному форматі:

```
python [параметр*]
    [файл_сценарію | -c команда | -m модуль | - ] [arg*]
```

У цьому форматі `python` позначає інтерпретатор мови Python, який виконується як за повністю зазначеним шляхом до каталогу, так і за словом `python`, зарезервованим у командній оболонці системи (наприклад, у змінній оточення `PATH`). Параметри командного рядка (*`параметр`*), що визначають режим роботи інтерпретатора Python, зазвичай вказуються перед ім'ям виконуваної програми, тоді як аргументи (*`arg`*), потрібні для виконання програми, — після її імені.

Параметри командного рядка у Python

Елементи *`параметр`* командного рядка слугують для позначення режиму роботи самого інтерпретатора Python. У версії Python 3.X допускаються наведені нижче параметри командного рядка, а їх відмінності у версії 2.X див. у частині "*Параметри командного рядка у версії Python 2.X*".

`-b`

Видати попередження про помилки під час виклику функції `str()` з об'єктом типу `bytes` або `bytearray`, але без аргументу, який позначає спосіб кодування символів, а також при порівнянні даних типу `bytes` або `bytearray` з даними типу `str`. За вказівки параметра **`-bb`** помилки видаються безпосередньо.

`-B`

Не записувати байт-код у файли з розширенням **`.pyc`** або **`.pyo`** під час імпорту.

`-d`

Активізувати виведення результатів налагодження синтаксичного аналізатора (призначене для розробників ядра інтерпретатора Python).

-E

Ігнорувати описані далі змінні оточення Python (наприклад, змінну `PYTHONPATH`).

-h

Вивести допоміжне повідомлення та вийти з програми.

-i

Увійти до діалогового режиму роботи після виконання сценарію. *Порада:* цей параметр зручний для налагодження програм після відмов. Див. також опис функції `pdb.pm()` у посібнику з бібліотек Python.

-O

Оптимізувати згенерований байт-код, створюючи та використовуючи файли з розширенням **`.pyc`** для зберігання байт-коду. На даний час цей параметр не надає помітних переваг з точки зору ефективності.

-OO

Цей параметр діє подібно до параметра **-O**, але при його вказівці видаляються також рядки документації з байт-коду.

-q

Не виводити повідомлення про версію та авторське право під час запуску програм у діалоговому режимі (починаючи з версії Python 3.2).

-s

Не вказувати локальний каталог користувача в шляху пошуку модулів у змінній `sys.path`.

-S

Не виконувати імпорт модулів з локального каталогу користувача під час ініціалізації.

-u

Примусово зробити стандартні потоки виведення даних (*`stdout`*) та помилок (*`stderr`*) небуферизованими та бінарними.

-v

Щоразу, коли ініціалізується модуль, виводити повідомлення з вказівкою місця, з якого він був завантажений. Для більш детального відображення цей параметр можна повторити.

-V

Вивести номер версії Python і вийти з програми (цей параметр може бути вказаний у вигляді **--version**).

-W *arg*

Цей параметр керує відображенням попереджень, де аргумент **arg** набуває вигляду *дія*:*повідомлення*:*категорія*:*модуль*:*номер_рядка*. Див. далі частини "*Каркас попереджень*" та "*Винятки категорій попереджень*", а також документацію модулю `warnings` у довідковому посібнику з бібліотеки Python (Library Reference), що доступний за адресою `http://www.python.org/doc/`.

-x

Пропустити перший рядок вихідного коду, що дозволяє використовувати форми директив `#!`*cmd*, що відрізняються від прийнятих в Unix.

-X *параметр*

Встановити *параметр* залежно від реалізації (починаючи з версії Python 3.2). Значення, що підтримуються і які приймає *параметр*, наведені в документації до конкретної реалізації.

Вказівка програм у командному рядку

Виконуваний код програми на Python і аргументи, що передаються їй, можуть бути вказані в командному рядку наступними способами.

файл_сценарію

Позначає ім'я файлу сценарію мовою Python, який має виконуватися як основний файл програми, що знаходиться на верхньому рівні її ієрархії (наприклад, за командою **python main.py** виконується код із файлу `main.py`). Ім'я файлу сценарію може бути вказано як за абсолютним, так і за відносним шляхом (з використанням "`.`"), і доступне в якості елемента `sys.argv[0]` списку

аргументів. У командних рядках на деяких платформах елемент **python** може бути опущений, якщо ці рядки починаються з імені файлу сценарію і не містять параметри, що визначають режим роботи самого інтерпретатора Python.

-c *команда*

Позначає (у вигляді символьного рядка) виконуваний код Python (наприклад, за командою **python -c "print('spam' * 8)"** у Python виконується операція виведення на друк). Значення `'-c'` встановлюється в елементі `sys.argv[0]` списку аргументів.

-m *модуль*

Виконує модуль у вигляді сценарію. Пошук *модуля* здійснюється за шляхом у змінній `sys.path`, а його виконання — у вигляді файлу, що знаходиться на верхньому рівні ієрархії (наприклад, за командою **python -m pdb s.py** модуль `pdb` налагодження програм на Python, що у каталозі стандартної бібліотеки, виконується з аргументом `s.py`). Повний шлях до модуля вказується в елементі `sys.argv[0]` списку аргументів.

-

Приймає команди Python зі стандартного потоку введення (за замовчуванням *stdin*). Входить у діалоговий режим роботи, якщо команди вводяться зі стандартного потоку введення `tty` (інтерактивного пристрою). Значення `'-'` вказується в елементі `sys.argv[0]` списку аргументів.

arg*

Позначає, що решта командного рядка передається файлу сценарію або команді і доступна в елементі `sys.argv[1:]` переліку аргументів.

Якщо жоден з параметрів *файл_сценарію*, *команда* або *модуль* не вказаний у командному рядку, інтерпретатор Python переходить у діалоговий режим роботи, вводячи команди зі стандартного потоку введення *stdin* і використовуючи для цієї мети бібліотеку *readline* з проекту GNU, за умови, що вона встановлена, а також задаючи значення `'-'` (пустий рядок) в елементі `sys.argv[0]` списку аргументів, якщо тільки інтерпретатор не викликається зі згаданим вище параметром **–**.

Окрім традиційних командних рядків, що вводяться за запрошенням системної командної оболонки, програми на Python можна запускати клацанням кнопкою миші на іменах їх файлів у графічному користувацькому інтерфейсі Провідника по файлах; викликаючи функції зі стандартної бібліотеки Python (наприклад, функцію `os.popen()`); а також вибираючи відповідні команди запуску з меню інтегрованих середовищ розробки (ICP) на кшталт IDLE, Komodo, Eclipse або NetBeans.

Параметри командного рядка у версії Python 2.X

У версії Python 2.X підтримується той самий формат командного рядка, що й у версії 3.X. Але в цій версії відсутня підтримка параметра **`-b`**, що пов'язано зі змінами у рядковому типі даних, а також параметрів **`-q`** та **`-X`**, введених у версії 3.X. У той же час підтримуються наведені нижче додаткові параметри, доступні у версіях 2.6 та 2.7, а можливо, і в попередніх версіях Python.

`-t` та **`-tt`**

Видають попередження про неузгоджене спільне вживання символів табуляції та пробілів при відступах. При використанні параметра **`-tt`** помилки видаються безпосередньо. У версії 3.X таке спільне вживання символів табуляції та пробілів завжди інтерпретується як синтаксичні помилки (додатково див. далі частину "*Правила синтаксису*").

`-Q`

Ці параметри **`-Qold`** (за замовчуванням), **`-Qwarn`**, **`-Qwarnall`** та **`-Qnew`** відносяться до нового режиму поділу у версії Python 3.X (див. далі частину "*Примітки щодо застосування операторів*").

`-3`

Видає попередження про будь-які ознаки несумісності з версією Python 3.X у вихідному коді, які неспроможний усунути інструментальний засіб *`2to3`* зі стандартної інсталяції Python.

`-R`

Активізує псевдовипадкове "зерно", щоб зробити непередбачуваними хеш-значення різних типів у проміжках між послідовними викликами інтерпретатора і тим самим запобігти атакам

типу відмови в обслуговуванні. Цей параметр з'явився у версії Python 2.6.8 і залишився заради сумісності у версії 3.X, починаючи з випуску 3.2.3. Однак починаючи з випуску 3.3 дана рандомізація активується за замовчуванням.

Змінні оточення Python

Змінні оточення (або так звані змінні *командної оболонки*) встановлюються на рівні системи. Вони доступні в програмах і застосовуються для їхнього глобального налаштування.

Операційні змінні

Нижче перераховані основні змінні оточення, що налаштовуються користувачем, які пов'язані з режимом роботи сценаріїв.

PYTHONPATH

Розширює вихідний шлях пошуку файлів модулів, які імпортуються. Формат значення цієї змінної такий самий, як і у значення, що встановлюється у змінній оточення `PATH` командної оболонки, а саме: імена шляхів до каталогів поділяються двокрапками (або точками з комою у Windows). Якщо змінну `PYTHONPATH` встановлено, то пошук імпортованих файлів чи каталогів модулів здійснюється в кожному каталозі, перерахованому в цій змінній зліва направо. Значення цієї змінної включається в змінну оточення `sys.path`, призначену для зберігання повного шляху пошуку модулів, що імпортуються, у крайніх зліва складових абсолютного шляху, після каталогу зі сценарієм і перед каталогами стандартних бібліотек. Див. далі опис змінної оточення `sys.path` у частинах "*Модуль `sys`*" та "*Оператор `import`*".

PYTHONSTARTUP

Якщо в цій змінній задано ім'я файлу, який можна прочитати, то команди Python з цього файлу виконуються до появи першого запрошення в діалоговому режимі роботи, що зручно для визначення інструментальних засобів, які часто використовуються.

PYTHONHOME

Якщо встановлено, це значення використовується як альтернативний каталог префіксів для бібліотечних модулів (`sys.prefix`, `sys.exec_prefix`). За замовчуванням пошук модулів здійснюється за шляхом `sys.prefix/lib`.

PYTHONCASEOK

Якщо ця змінна встановлена, то в інструкціях імпорту регістр в іменах файлів не враховується (нині підтримується лише у Windows та Mac OS X).

PYTHONIOENCODING

Цій змінній присвоюється символьний рядок формату *`найменування_кодування`*`[:`*`обробник_помилок`*`]` для заміни кодування в унікоді (і додатково — обробника помилок) при введенні тексту зі стандартного потоку *`stdin`* і його виведенні у стандартні потоки *`stdout`* і *`stderr`*. Таке налаштування може знадобитися в деяких командних оболонках для обробки тексту, який не може бути представлений в коді ASCII. Так, якщо виведення тексту виявиться невдалим, можна спробувати задати в цій змінній кодування UTF-8 (тобто значення `"utf8"`).

PYTHONHASHSEED

Якщо має значення `random`, в якості початкового значення для послідовності псевдовипадкових чисел для хешування об'єктів типу `str`, `bytes` і `datetime` використовується випадкове значення. Для отримання хеш-значень з точно передбачуваним значенням у цій змінній можна встановити ціле число в межах від **`0`** до **`4294967295`**. Така можливість підтримується у версіях Python 2.6.8 та 3.2.3.

PYTHONFAULTHANDLER

Якщо ця змінна встановлена, то задані обробники реєструються при запуску програми на Python для відображення вмісту оперативної пам'яті і трасування викликів функцій при фатальних помилках. Починаючи з версії Python 3.3 це рівнозначно вказівці параметра **`-X`** *`обробник_відмов`* у командному рядку.

Змінні, аналоги параметрів командного рядка в Python

Наведені нижче змінні оточення є аналогами деяких параметрів командного рядка в Python (див. частину "*Параметри командного рядка у Python*").

PYTHONDEBUG

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-d**.

PYTHONDONTWRITEBYTECODE

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-B**.

PYTHONINSPECT

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-i**.

PYTHONNOUSERSITE0

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-s**.

PYTHONOPTIMIZE

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-O**.

PYTHONUNBUFFERED

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-u**.

PYTHONVERBOSE

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-v**.

PYTHONWARNINGS

Якщо ця змінна не порожня, то вона діє аналогічно параметру **-W** з тим самим значенням. Ця змінна приймає також розділений комами символьний рядок, рівнозначний вказівці кількох параметрів **-W**. Така можливість є у версіях Python, починаючи з 2.7 та 3.2.

Запуск програм Python у Windows

Починаючи з версії Python 3.3 у Windows (і лише в цій операційній системі) встановлюється засіб запуску сценаріїв, який в попередніх версіях був доступний окремо. Цей засіб складається з виконуваних файлів `py.exe` (консольний варіант) та `pyw.exe` (віконний варіант), які можна викликати без налаштувань змінної оточення `PATH`. Ці фай-

ли реєструються для виконання сценаріїв Python через співставлення типів файлів і дозволяють обирати версії Python за допомогою:

- Unix-подібних директив `#!`, що вказуються на самому початку сценаріїв;
- аргументів командного рядка;
- параметрів за замовчуванням.

Директиви запуску файлів

Засіб запуску розпізнає рядки коду з директивами `#!` на самому початку файлів сценаріїв. У цих директивах вказуються версії Python в одній із наведених нижче форм, де **`*`** означає використання версії *за замовчуванням* (нині — версії **`2`**, якщо вона встановлена, що рівноцінно пропуску директиви `#!`); номера *основної* версії (наприклад, **`3`**) для запуску останньої встановленої версії або ж *повної* специфікації у формі *`мажорна_версія.міноpна_версія`* з додатковим суфіксом **`-32`**, що означає бажаний варіант установки 32-розрядної версії (наприклад, `3.1-32`).

```
#!/usr/bin/env python*
#!/usr/bin/python*
#!/usr/local/bin/python*
#!python*
```

Будь-які аргументи команд Python (`python.exe`) можуть бути задані в кінці рядка, а в Python версії 3.4 і пізніших можна звернутися до змінної оточення `PATH` для отримання рядків з директивами `#!`, що надають лише позначення `python` без явної вказівки номера версії.

Командні рядки для запуску

Засіб запуску може бути викликано з командних рядків на запрошення системної командної оболонки в наступній формі:

```
py [pyarg] [pythonarg*] script.py [scriptarg*]
```

У більш загальному випадку все, що може з'явитися в команді `python` після складової `python`, може також з'явитися після додаткового аргументу *`pyarg`* у команді `py` і бути дослівно передано засобу запуску програм на Python. До цього відносяться параметри **`-m`**, **`-c`** та

– з форм специфікації програм (див. частину "*Запуск програм на Python з командного рядка*").

Засіб запуску приймає наведені нижче форми свого додаткового, але не обов'язкового аргументу *pyarg*, що відображають складову * із рядка з директивою #! у кінці файлу сценарію.

```
-2          Запустити останню встановлену версію 2.X
-3          Запустити останню встановлену версію 3.X
-X.Y        Запустити вказану версію, де X дорівнює 2 або 3
-X.Y-32     Запустити вказану 32-розрядну версію
```

Якщо є і те, і те, то аргументам командного рядка надається перевага над значеннями в рядках з директивою #!. При відповідній установці рядки з директивою #! можуть застосовуватися і в ширшому контексті (наприклад, при виборі піктограм клацанням миші).

Змінні оточення для запуску

У засобі запуску розпізнаються також налаштування додаткових змінних оточення, які можуть бути використані для спеціального налаштування вибору версії у стандартних чи окремих випадках (наприклад, пропуск версії, її вказівка тільки в основній директиві #! або в аргументі командного рядка py), як показано нижче. Всі ці налаштування використовуються тільки у виконуваних файлах засобу запуску, але не при безпосередньому виклику команди *python*.

```
PY_PYTHON   Версія для стандартних випадків (інакше версія 2)
PY_PYTHON3  Версія для 3 окремих випадків (наприклад, 3.2)
PY_PYTHON2  Версія для 2 окремих випадків (наприклад, 2.6)
```

Вбудовані типи та оператори

Оператори та їх пріоритети

У табл. 1 перераховані оператори виразів Python. Оператори в нижніх комірках цієї таблиці мають більш високий пріоритет (тобто тіснішу прив'язку), коли вони застосовуються у виразах з різнотипними операторами без круглих дужок.

Таблиця 1. Оператори виразів Python 3 та їх пріоритети

Оператор	Опис
`yield X`	Повертає результат виконання функції-генератора (значення функції `send()`)
`lambda args1:X`	Створює анонімну функцію (повертає під час виклику значення *X*)
`X if Y else Z`	Тернарний оператор вибору (значення *X* обчислюється лише в тому випадку, якщо *Y* є істиною)
`X or Y`	Логічна операція АБО: значення *Y* обчислюється лише у разі, якщо *X* хибне
`X and Y`	Логічна операція І: значення *Y* обчислюється лише в тому випадку, якщо *X* істинне
`not X`	Логічне заперечення
`X in Y`, `X not in Y`	Членство: ітератори, множини
`X is Y`, `X is not Y`	Перевірки об'єктів на ідентичність
`X<Y`, `X<=Y`, `X>Y`, `X >=Y`	Порівняння величин, підмножин і надмножин окремих множин
`X==Y`, `X !=Y`	Оператори рівності
`X|Y`	Логічна операція порозрядного АБО, об'єднання множин
`X^Y`	Логічна операція порозрядного виключного АБО, різнойменність множин
`X&Y`	Логічна операція порозрядного І, перетин множин
`X<<Y`, `X >Y`	Зсув значення *X* вліво або вправо на кількість бітів, що дорівнює *Y*
`X+Y`, `X–Y`	Додавання/зчеплення, віднімання/різниця множин
`X*Y`, `X%Y`, `X/Y`, `X//Y`	Множення/повторення, отримання залишку від ділення/форматування, поділ, цілочисельне ділення
`–X`, `+X`	Унарна зміна знаку, тотожність
`~X`	Логічна операція порозрядного НІ (інверсія)
`X**Y`	Піднесення до ступеня
`X[i]`	Індексування (послідовностей, відображень та іншого)
`X[i:j:k]`	Нарізка (усі три межі необов'язкові)
`X(args2)`	Виклик (функції, методу, класу та іншого об'єкта, що викликається)
`X.attr`	Посилання на атрибут
`(....)`	Кортеж, вираз, вираз-генератор
`[....]`	Список, генератор списків
`{....}`	Словник, множина, генератор словників та множин

Атомарні члени та динамічна типізація

Елементи виразів **`X`**, **`Y`**, **`Z`**, **`i`**, **`j`** і **`k`** у табл. 1 можуть бути наступними.

- *Імена змінних*, що замінені останнім присвоєним їм значенням.
- *Літеральні вирази*, що визначаються далі у частині "*Конкретні вбудовані типи*".
- *Вкладені вирази*, що обираються з будь-якого рядка в табл. 1 (можливо, у круглих дужках).

Змінні в Python слідують *моделі динамічної типізації*, тобто вони не оголошуються, але створюються, коли їм присвоюють значення; також зберігають посилання на об'єкти у вигляді значень; можуть посилатися на будь-який тип об'єкта, а оскільки вони не мають значень за замовчуванням, значення мають присвоюватися їм до їх використання у виразах. У іменах змінних обов'язково враховується регістр (докладніше про це — у частині "*Правила іменування*"). Об'єкти, на які посилаються змінні, автоматично створюються та звільняються з оперативної пам'яті системою "збору сміття" Python, коли вони більше не потрібні. Для цього в реалізації CPython використовується підрахунок посилань.

Крім того, замінний елемент **`attr`** у табл. 1 має бути буквальним ім'ям атрибута (без лапок); елемент **`args1`** — списком формальних аргументів, що визначається далі в частині "*Оператор* `def`"; елемент **`args2`** — списком вхідних аргументів, що визначається далі в частині "*Оператор виразу*"; а літерал **`...`** кваліфікується як атомарний вираз (лише у версії 3.X). Синтаксис генераторів літералів та структур даних (кортежів, списків та множин), що вказуються в абстрактній формі у трьох останніх рядках табл. 1, визначається далі в частині "*Конкретні вбудовані типи*".

Примітки щодо застосування операторів

- Нерівність значень може бути записана як `X!=Y` або `X<>Y`, але тільки у версії Python 2.X. У версії ж Python 3.X останній із цих двох варіантів запису виключено як надмірний.
- Вираз у зворотних лапках `` `X` `` діє аналогічно виразу `repr(X)`, перетворюючи об'єкти в символьні рядки, які відображаються, проте лише у версії Python 2.X. У версії Python 3.X для цієї мети

використовуються більш зрозумілі й зручні для читання вбудовані функції `str()` і `repr()`.

- В обох версіях, Python 2.X і 3.X, вираз *цілочисельного ділення* *X*`//`*Y* завжди призводить до відкидання дробових залишків і повернення цілочисельного результату для цілих початкових значень.
- Вираз *X*`/`*Y* виконує *справжнє (істинне) ділення* у версії 3.X, завжди зберігаючи в результаті залишок у форматі з плаваючою точкою, а у версії 2.X — *класичне ділення*, відкидаючи залишок від ділення цілих значень, окрім тих випадків, коли у версії 2.X активізовано режим справжнього ділення з версії 3.X за допомогою оператора `from __future__ import division` або параметра **`-Qnew`** командного рядка в Python.
- Синтаксис `[....]` потрібен для позначення спискових літералів і виразів-генераторів. У останньому випадку виконується цикл, який мається на увазі, а результати обчислення виразу накопичуються в новому списку.
- Синтаксис `(....)` потрібен для позначення кортежів і виразів, а також виразів-генераторів — форми генераторів списків, що видає результати за вимогою замість побудови списку результатів. Круглі дужки іноді можуть опускатися в усіх трьох мовних конструкціях.
- Синтаксис **`{....}`** потрібен для позначення словникових літералів. У версіях Python 2.7 і 3.X використовуються також літерали множин, генератори словників та множин. У Python 2.6 і більш ранніх версіях для цієї мети слід використовувати функцію `set()` і оператори циклів.
- Оператор вибору `yield` та умовні оператори `if`/`else` доступні, починаючи з версії Python 2.5. Так, оператор `yield` повертає аргументи функції `send()` у генераторах, а умовні оператори `if`/`else` є короткою формою багаторядкового умовного оператора `if`. Оператор `yield` потрібно вказувати у круглих дужках, якщо він не є єдиним у правій частині оператора присвоювання.
- Оператори порівняння можна поєднувати в ланцюжки. Так, вираз *X*`<`*Y*`<`*Z* дає такий самий результат, як і вирази *X*`<`*Y* та *Y*`<`*Z*, але при об'єднанні в ланцюжок значення *Y* обчислюється лише один раз.

- Вираз нарізки `X[i:j:k]` є тотожним індексації за допомогою об'єкта нарізки: `X[slice(i,j,k)]`.
- У версії Python 2.X допускається порівняння різнотипних величин зі зведенням числових значень до загального типу та впорядкуванням інших типів даних відповідно до найменування типу. У версії Python 3.X порівняння різнотипних нечисельних величин, у тому числі й при сортуванні за проксі-об'єктом, не допускається, і призводить до появи винятків.
- Порівняння величин у словниках, починаючи з версії Python 3.X, також не підтримується, окрім перевірки на рівність. Тому одним із можливих варіантів заміни порівняння у версії 3.X може бути виклик функції `sorted(addict.items())`.
- У виразах із викликом функцій допускається вказувати позиційні та іменовані аргументи, а також довільні великі числа як аргументи. Синтаксис викликів функцій наведено далі у частинах "*Оператор виразу*" та "*Оператор* `def`".
- У версії Python 3.X допускається вказувати багатокрапку (буквально — як `...`, або за допомогою вбудованого імені `Ellipsis`) для позначення атомарних виразів у вихідному коді. Таке позначення може слугувати альтернативою оператору `pass` або об'єкту `None` в деяких видах контексту (наприклад, у тілі фіктивних функцій або ж при ініціалізації змінних незалежно від їх типу).
- У наступних версіях, починаючи з Python 3.5, *може* бути узагальненим (хоча повної впевненості в цьому поки немає) синтаксис виразів `*X` і `**X` зі знаком зірочки, що з'являються в літералах структур даних і генераторах, де подібним чином колекції розпаковуються в окремі елементи, як це зараз робиться під час виклику відповідних функцій. Докладніше про це — у частині "*Оператор присвоєння*".

Категорії операцій

У іменах методів типу `__X__`, що згадуються в цій частині, завершальні круглі дужки задля стислості опускаються. Загалом, усі вбудовані типи даних підтримують операції *порівняння* та *логічні* операції,