МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М. М. Козуля, Т. В. Козуля

ПОЧАТОК РОБОТИ З PYTHON І РОБОТА З ДАНИМИ

Лабораторний практикум
з навчальної дисципліни
«Основи програмування Python (дисципліна вибору 02)»
Частина перша
для студентів спеціальностей 122 «Комп'ютерні науки»
126 «Інформаційні системи та технології»

Харків
НТУ «ХПІ»
2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М. М. Козуля, Т. В. Козуля

ПОЧАТОК РОБОТИ З PYTHON І РОБОТА З ДАНИМИ

Лабораторний практикум
з навчальної дисципліни
«Основи програмування Python (дисципліна вибору 02)»
Частина перша
для студентів спеціальностей 122 «Комп'ютерні науки»
126 «Інформаційні системи та технології»

Рекомендовано
редакційно-видавничою
радою університету,
протокол №1 від 03.01.2022

Харків
НТУ «ХПІ»
2022

УДК 004.432.2
К 59

Рецензенти:
М.А. Гринченко, канд. техн. наук, доцент САУ
В. О. Дурєєв, канд. техн. наук, доцент АСБІТ (НУЦЗ України)

Козуля М. М., Козуля Т. В.

К 59 Початок роботи з Python і робота з даними: лабораторний практикум з навчальної дисципліни «Основи програмування Python (дисципліна вибору 02)» для студентів спеціальностей 122 «Комп'ютерні науки» та 126 «Інформаційні системи та технології». Харків: НТУ «ХПІ», 2022. 97 с.

ISBN

У першій частині лабораторного практикуму з дисципліни «Основи програмування Python» надані матеріали 3-х лабораторних робоіт з завданням до практичного виконання задач щодо набуття початкових знань з Python. У практикуму відзначені перші кроки роботи з даними для набуття студентами професійних умінь (компетенцій) та практичних навичок при роботі з мовою програмування Python. Наведено багато прикладів, які ілюструють реалізацію конкретних завдань.

Призначено для студентів, викладачів і користувачів, які вивчають основи програмування Python для практичного застосування при створенні додатків для розвязку задач у різних галузях науково-практичних досліджень.

Табл. 7. Іл. 26 Бібліогр. 17

УДК 004.432.2

ISBN

ЗМІСТ

ВСТУП

Популярність Python почала рости з 2010-го року. Сьогодні Python залишається одним із самих затребуваних мов програмування. За даними опитування Stack Overflow за 2020 рік, він займає четвертий рядок у рейтингу популярних технологій і розташовується на першому місці в списку мов, які прагли б вивчити користувачі ресурсу. Згідно з індексом співтовариства програмістів TIOBE, у травні 2021-го більш актуальною за Пітон відзначено тільки мову С.

Python розроблявся як розширювана скриптова мова для розподіленої ОС Amoeba, що пов'язана з розв'язком невеликих завдань. Надалі Python удосконалювався у напряму набуття позитивних властивостей і розширення можливостей, таких як:

- простота – зрозумілий синтаксис, код виглядає лаконічно, його легко читати й писати;
- кросплатформеність – програма буде працювати на будь-якій платформі, на якій установлений інтерпретатор для виконання коду;
- розвинене середовище – інформаційний простір на профільних форумах, у блогах;
- затребуваність на ринку – на Python майже повністю написаний Youtube і серверна частина Instagram, аналізують дані Spotify і Amazon, створена система автоматизації процесів WAS на його основі в NASA, часто використовують для стартапів;
- універсальність – затребувана мова для програмування і для наукових досліджень;
- спрощує роботу – велика кількість бібліотек і фреймворків.

Python підтримує структурне об'єктно-орієнтоване, функціональне, імперативне та аспектно-орієнтоване програмування.

За наявності деяких недоліків, а саме низька швидкість, скрипти компілюються щораз під час виконання коду, що призводить до зростання кількості тестів для виявлення помилок, необхідно відзначити головні сфери застосування Python: Веб-розробка; Графічний інтерфейс; Бази даних; Системне програмування; Складні обчислювальні процеси; Машинне навчання; Автоматизація процесів; Індустрія ігор.

Python уважається кращою мовою для таких активно зростаючих областей, як Великі дані й Машинне навчання. Наприклад, за допомогою

Python збирають інформацію про купівельну активність і знаходять нові шляхи розвитку брендів, автоматизують рутинні завдання, як збір усіх зображень із сайту, готовлять алгоритми машинного навчання, так, Netflix створив свій сервіс рекомендацій. Python допомагає в розвитку соціальних мереж. З ним можна створити бота або зібрати цільову аудиторію, написавши програму для парсинга.

Отже, актуальним є вивчення цієї мови програмування і підвищення навичок роботи з нею. Відповідно до сказаного вище лабораторний практикум до навчальної дисципліни «Основи програмування Python» поділений на три тематичні частини, де пропонується студентам опанувати основи Python за такими темами:

- Початок роботи з Python;
- Робота з умовними операторами, умовна конструкція if..else, if…elif…else. Модуль Math у Python;
- Робота з даними: списки, рядки, множини, словники;
- Робота з циклами;
- Робота з функціями, виключеннями та класами;
- Робота з вбудованими модулями;
- ООП (Object-Oriented programming);
- Робота з файлами;
- Робота з потоками. Тестування.

Кожній темі відповідають методичні вказівки, які оформлені як окреме видання, що відповідає певній тематичній частині курсу, а саме «Початок роботи з Python і робота з даними», «Робота з Python: функції, класи, вбудовані модулі», «Об'єктно-орієнтоване програмування в Python. Потоки. Тестування». Це обумовлено, по-перше, тим, що у результаті об'єднання лабораторних за тематикою курсу дотримується зв'язок з теоретичним лекційним матеріалом. По-друге, цей лабораторний комплекс будуть опановувати студенти чотирьох спеціальностей з різними вимогами до ступеню вивчення матеріалу різних тем. По-третє, оформлення методичних вказівок за різними темами сприяє більшій гнучкості навчального процесу, дозволяє викладачу у разі потреби створювати індивідуальні графіки роботи для студентів.

Лабораторний практикум кожної частини курсу складається з теоретичного матеріалу, завдань і вправ для закріплення лекційного матеріалу і отримання навичок програмування на Python сучасних версій.

ЛАБОРАТОРНА РОБОТА № 1
ІНСТАЛЯЦІЯ PYTHON ТА СЕРЕДОВИЩА РОЗРОБКИ

1 ЗАГАЛЬНІ ВІДОМОСТІ: ПОЧАТОК РОБОТИ В PYTHON

1.1 Запуск і перевірка працездатності Python

Зараз доступні дві версії Python: Python 2 і більш нова версія Python 3. Кожна мова програмування розвивається з появою нових ідей і технологій, тому розробки Python спрямовані зробити мову більш потужною і гнучкою. Багато змін мають другорядний характер і малопомітні на перший погляд, але в окремих випадках код, написаний на Python 2, некоректно працює в системах з встановленою підтримкою Python 3. Отже, якщо в вашій системі встановлені обидві версії чи ви ще встановили Python, то використовуйте Python 3.

У світі програмування здавна прийнято починати освоєння нової мови з програми, що виводить на екран повідомлення Hello world! Мовою Python програма Hello World складається всього з одного рядка:

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

Навіть така проста програма виконує цілком конкретну функцію, якщо вона запускається в вашій системі, то і будь-яка програма, яку ви напишете на Python, теж повинна запускатися нормально.

Python використовується та розповсюджується абсолютно безкоштовно. Для завантаження файлів інсталяції Python необхідно перейти на сайт https://www.python.org (рис. 1.1).

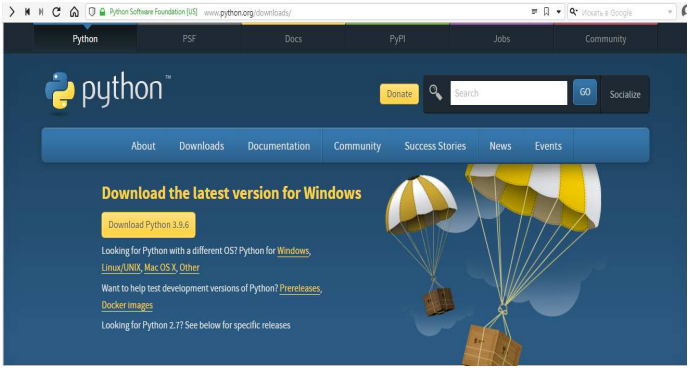
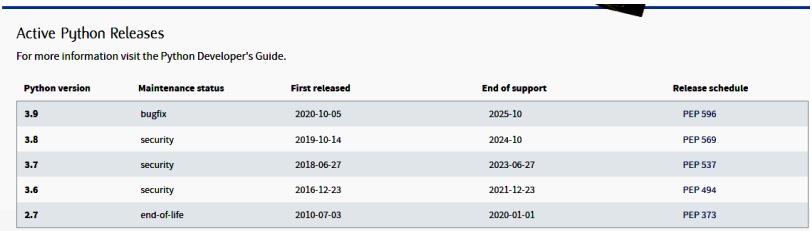


Рисунок 1.1 – Веб-сторінка Python

Існує декілька версій Python (рис. 1.2), але для вивчення Python як мови програмування та виконання лабораторних робіт необхідно завантажити та використовувати останню версію.



Active Python Releases

For more information visit the Python Developer's Guide.

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
3.6	security	2016-12-23	2021-12-23	PEP 494
2.7	end-of-life	2010-07-03	2020-01-01	PEP 373

Рисунок 1.2 – Версії Python

Після завантаження необхідним кроком є інсталяція .exe файла. Для цього обирають меню «Install Now» (рис. 1.3).

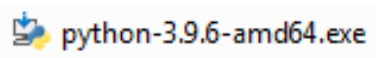


Рисунок 1.3 – Файл інсталяції Python

Необхідно почекати інсталяцію усіх файлів (рис. 1.4–1.5).

Рисунок 1.4 – Вікно початку інсталяції Python

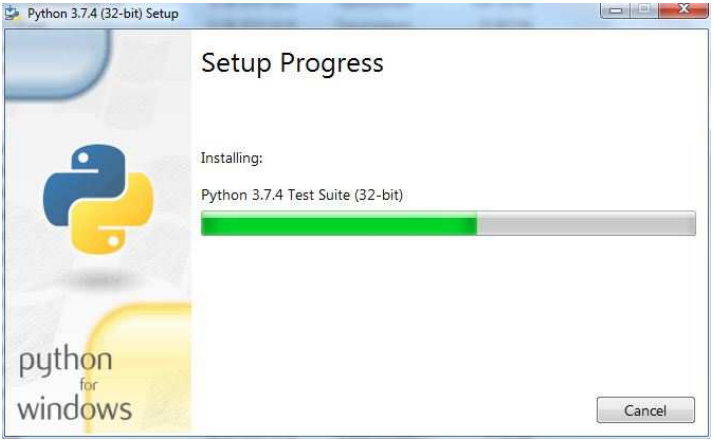


Рисунок 1.5 – Вікно процесу інсталяції Python

Закінчення інсталяції повідомляється завдяки появі вікна «Setup was succesfull» (рис. 1.6).

Рисунок 1.6 – Повідомлення про закінчення процесу інсталяції

У Python входить інтерпретатор, який виконується в термінальному вікні і дозволяє випробувати фрагменти коду Python без збереження і запуску усієї програми.

Для перевірки повноти та правильності інсталяції Python необхідно у Меню Пуск знайти теку з інстальованим Python та запустити інтерпретатор (рис. 1.7).

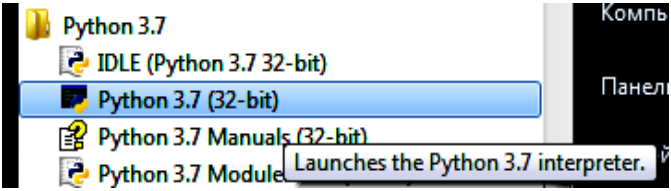


Рисунок 1.7 – Визначення Інтерпретатора Python

Інтерпретатор дозволяє переконатися у працездатності Python за результатами операції ініціалізації змінної та її виведення (рис. 1.8).

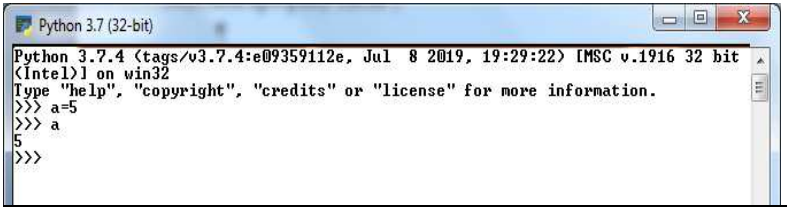


Рисунок 1.8 – Робота з інтерпретатором Python

1.2 Середовища розробки Python

Python безкоштовно використовують на будь-яких операційних системах, і він підтримується широкою спільнотою розробників, для нього є безліч безкоштовних бібліотек. Python підтримує всі способи розробки, включаючи веб-додатки, веб-служби, комп'ютерні програми, скрипти і наукові обчислення

Для роботи з Python використовують сторонні інтегровані середовища розробки (Integrated Development Environment, IDE), серед яких більш поширеними (десктоп і онлайн) вважаються такі:

1 **PyCharm** – інтегроване кросплатформне середовище розробки для програмування на мові Python, яка сумісна з Windows, MacOs, Linux. Завантаження Python здійснюється через веб-сторінку https://www.jetbrains.com/ru-ru/pycharm/(рис. 1.9).

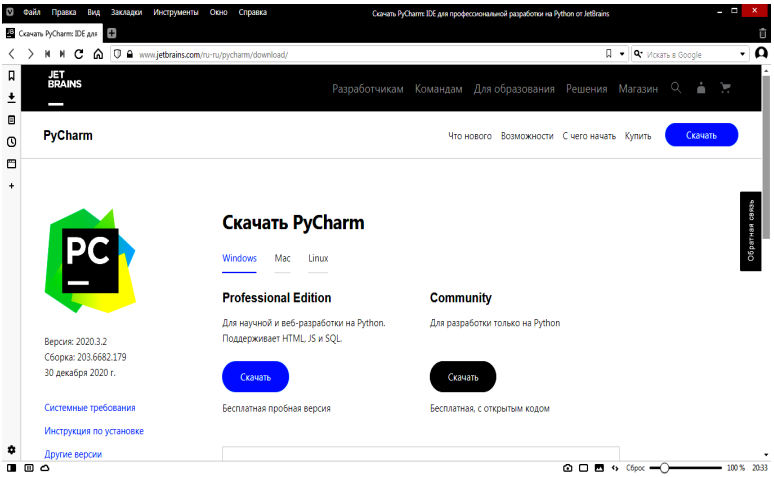


Рисунок 1.9 – Веб-сторінка завантаження PyCharm

2 **PyDev** – це Python IDE для Eclipse, яке може використовуватися в розробці Python, Jython та IronPython. Здійснюється завантаження при посиланні на веб-сторінку https://www.pydev.org. (рис. 1.10)

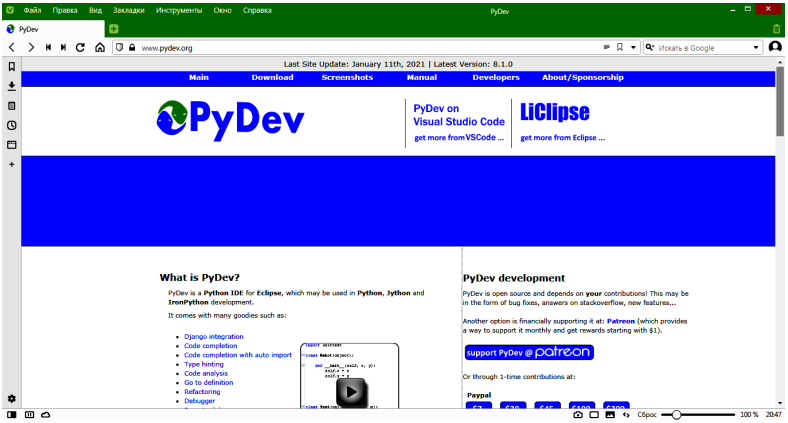


Рисунок 1.10 – Веб-сторінка завантаження PyDev

3 **PyScripter** – це Python IDE працює під Windows і поширюється безкоштовно, завантаження здійснюється з веб-сторінки htt-

ps://sourceforge.net/projects/pyscripter/ (рис. 1.11).

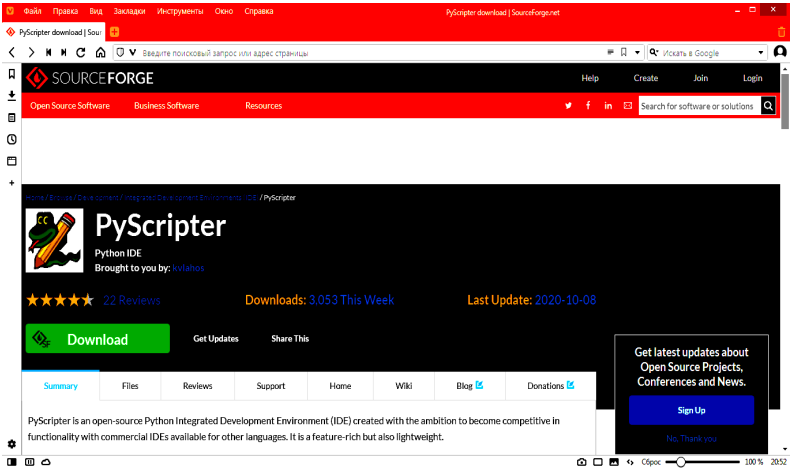


Рисунок 1.11 – Веб-сторінка завантаження PyScripter

4 **Python-IDLE** (Integrated Development and Learning Environment) – це інтегрована середа розробки та навчання на мові Python, створена за допомогою бібліотеки Tkinter. Інсталюється разом з Python (рис. 1.12). Завдяки Python-IDLE розширені можливості роботи з Python, наприклад, переглядаються всі інстальовані бібліотеки (рис. 1.13).

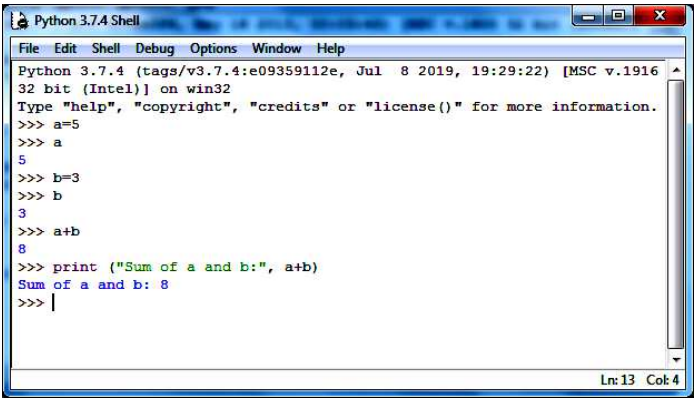


Рисунок 1.12 – Вікно Python-IDLE

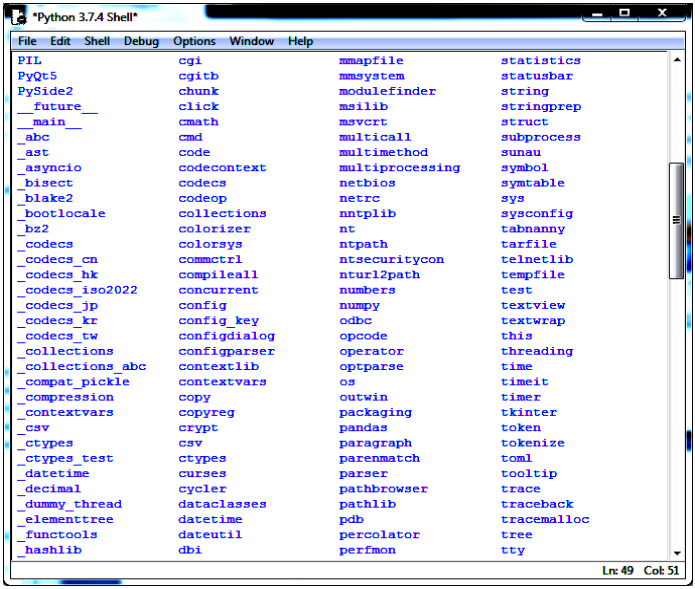


Рисунок 1.13 – Перелік інстальованих бібліотек

5 **CodeChef** – це співтовариство програмістів з усього світу. CodeChef був запущений в якості освітньої ініціативи в 2009 році індійською компанією-розробником програмного забезпечення Directi, сайт: https://www.codechef.com (рис. 1.14–1.15).

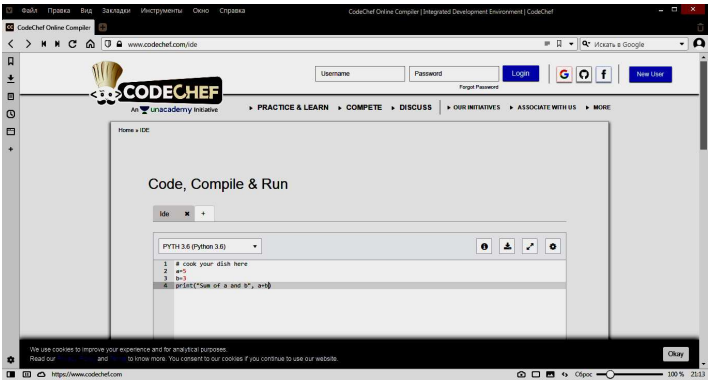


Рисунок 1.14 – Приклад роботи он-лайн компілятора CodeChef

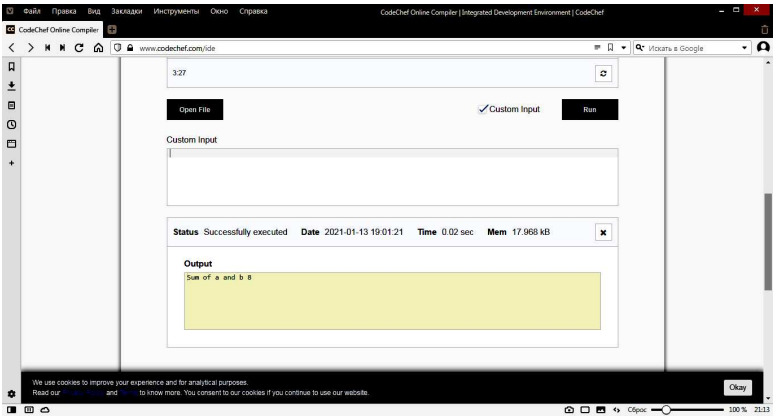


Рисунок 1.15 – Результати операцій в он-лайн компіляторі

6 **Python Fiddle** – це веб-IDE для роботи з Python, що є доступним через сайт: http://pythonfiddle.com (рис. 1.16).

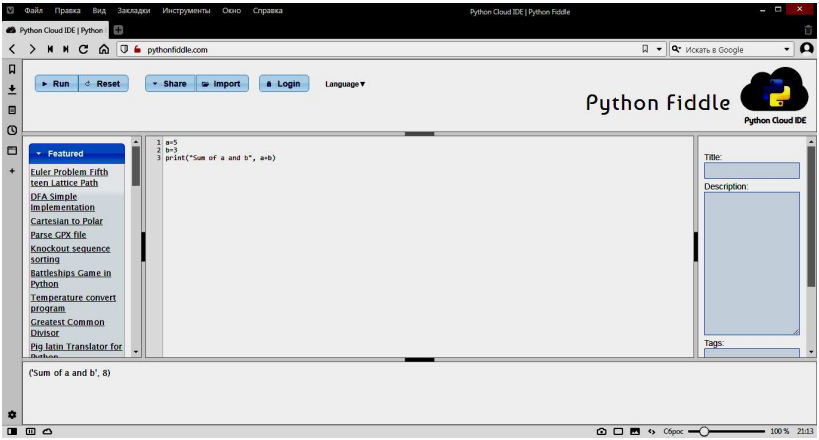


Рисунок 1.16 – Робочий простір веб-IDE Python Fiddle

7 **CodingGround tutorialspoint** – це безкоштовна он-лайн веб-IDE, де можна працювати з Python та з іншими мовами програмування. Для роботи з Python 3 необхідно скористатися таким сайтом: https://www.tutorialspoint.com/execute_python3_online.php (рис. 1.17).

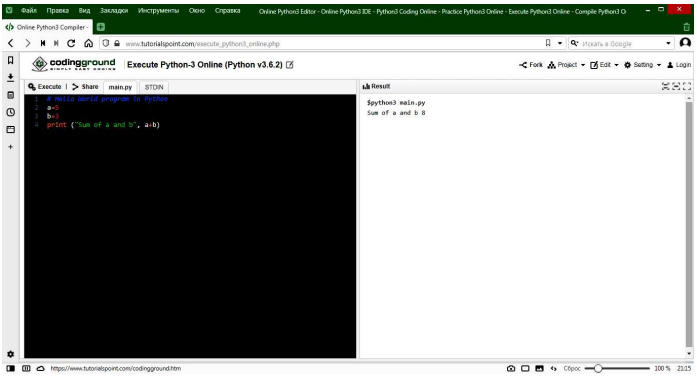


Рисунок 1.17 – Приклад роботи веб-IDE CodingGround

8 **OnlineGDB** – он-лайн IDE для роботи з Python. Цей компілятор підтримує й інші мови програмування, які можна вибрати з вкладеного списку, що на сайті https://www.onlinegdb.com/online_python_compiler (рис. 1.18).

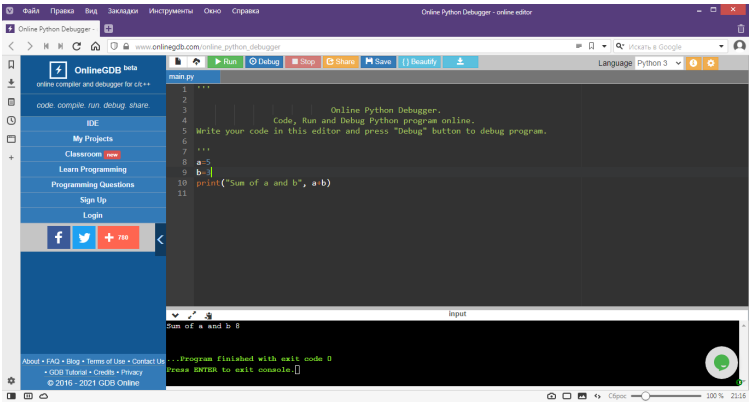


Рисунок 1.18 – Приклад роботи OnlineGDB

Окрім існуючих IDE для роботи з Python можна використовувати звичайний блокнот і зберігати файли з розширенням *.py. Запуск цих файлів відбувається через командну строку Windows (рис. 1.19–1.20). Недоліком використання блокноту є неможливість отримання необхідної інформації при написанні коду, автоматичного створення синтаксису.

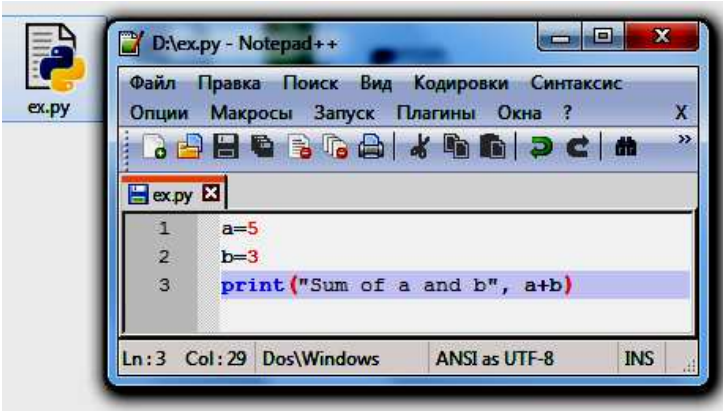


Рисунок 1.19 – Робота з Python у блокноті

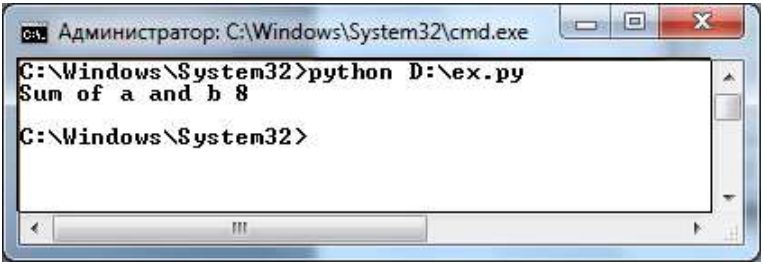


Рисунок 1.20 – Запуск коду через cmd

Рекомендацій щодо використання конкретного IDE немає.

Python є крос-платформних мовою програмування, таким чином, він працює у всіх основних операційних системах. Будь-яка програма написана на мові Python повинна виконуватися на будь-якому сучасному комп'ютері зі встановленою підтримкою Python.

2 РОБОТА В PYTHON: ТИПИ ДАНИХ, ВВЕДЕННЯ ТА ВИВЕДЕННЯ ДАНИХ

Типи даних, їх використання в Python

Кожне значення в Python відноситься до певного типу даних. Оскільки все в мові програмування Python є об'єктом, типи даних насправді є класами, а змінні є екземплярами (об'єктами) цих класів.

До основних типів даних відносять такі:

1) None – неозначене значення змінних;

2) Boolean Type – Логічні змінні;

3) Numerical type – Числа :

– int – ціле число;

– float – число з плаваючою точкою;

– complex – комплексне число;

4) Sequence Type – Списки:

– list – список;

– tuple – кортеж;

– range – діапазон;

5) Text sequence Type (str) – Рядки;

6) Binary Sequence Types – Бінарні списки:

– bytes – байти;

– bytearray – масиви байтів;

– memoryview – спеціальні об'єкти для доступу до внутрішніх даних об'єкта через protocol buffer;

7) Set Types – множина:

– set – множина;

– frozenset – незмінна множина;

8) Mapping Types – Словники:

– dict – словники.

Для роботи з даними у Python ініціалізація їх не потрібна. Для **введення** змінної і присвоєння значення необхідно зазначити ім'я змінної і надати значення. Якщо значення є рядком, то його беруть у лапки (“ ”, ‘ ’). Наприклад:

```
a=5
c=“Hello”
```

При використанні оператора присвоєння (=) в одній інструкції можна провести ініціалізацію декількох змінних:

```
a = b = c =3
a, b, c=1, 2, 3
```

Для **виведення даних** використовують вбудовану функцію `print()`. Для виводу рядків символів потрібно взяти в лапки дані або подвійні (“ ”), або одинарні (‘ ’).

Наприклад, для створення простої програми з виведення даних необхідно створити код, як зазначено на рисунку 2.1.

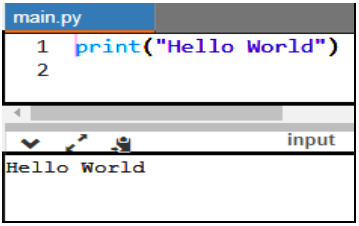


Рисунок 2.1 – Лістинг коду та результат виведення даних

Функція виведення даних `print()` має такий синтаксис:

```
print(*object, sep= ‘ ’, end= ‘\n’, file= sys.stdout,
                flush=False),
```

де `objects` – об’єкт, який необхідно вивести; * – знак, що вказує на наявність декілька об’єктів; `sep` – розділ об’єктів; значення за замовчуванням: пробіл; `end` – ставиться після всіх об’єктів; значення за замовчуванням: перехід на нову строку (‘\n’); `file` – очікується об’єкт з виводом методом `write (string)` (якщо значення не задано, то для вивода об’єктів використовують файл `sys.stdout`); `flush` – задано значення `True` і потік примусово скидається в файл; значення за замовчуванням: `False`.

Аргументи функції `print()` розділяються між собою комою. Якщо серед аргументів є вираз (в дужках), то спочатку буде виконано його дія, а потім виведено на екран результат функції (рис. 2.2).

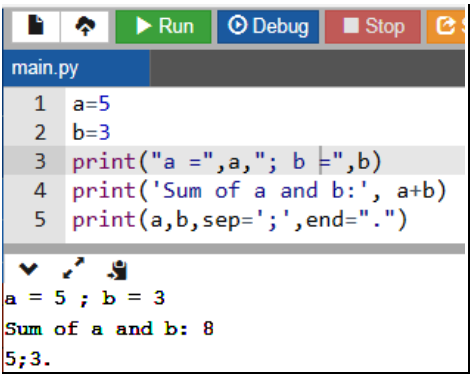


Рисунок 2.2 – Робота з функцією `print()`

Функцію `print()` використовують також для форматування рядків: старий стиль (С-стиль) і метод `format` (рис. 2.3).

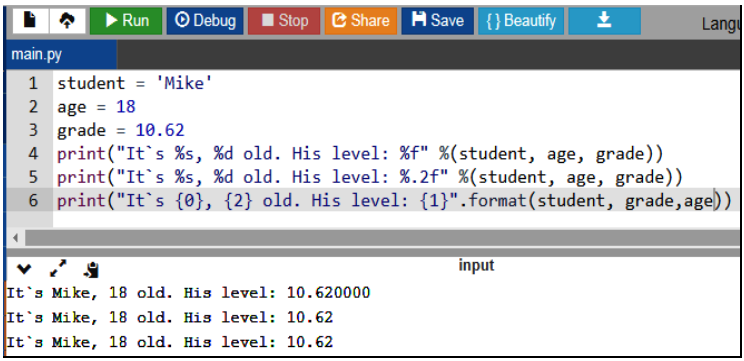


Рисунок 2.3 – Робота з форматуванням виводу

Згідно з лістингом на рисунку 2.3 на рядках 4 і 5 використано старий стиль форматування, де %s, %d, %f – рядок, ціле число, дійсне число, відповідно.

Якщо необхідно вивести декілька разів рядок/ціле число/дійсне число, то дублюється %s / %d / %f відповідно, а значення беруться з дужок у порядку, що задано (рис. 2.4).

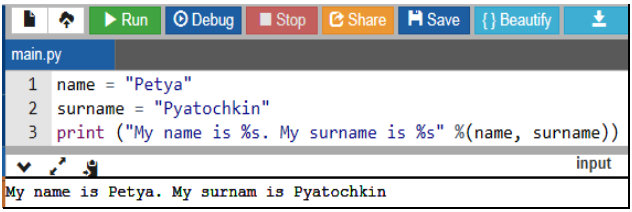


Рисунок 2.4 – Виведення рядків

Якщо необхідно задати кількість знаків після коми, необхідно перед буквою f поставити точку та число, яке і вказує кількість цифр (див. рис. 2.3 рядок 5).

Використання методу `format` вказано на рисунку 2.3 рядком 6. У фігурних дужках вказується номер даних (нумерація починається з 0), які беруться з методу `format()`. Номери в дужках можна не вказувати, але тоді необхідно слідкувати за порядком перерахування у методі.

Для вводу даних у Python надано вбудовану функцію `input()`, що має такий синтаксис:

```
input([prompt]),
```

де `prompt` – аргумент підказка, який записується у стандартний вивід без переходу на новий рядок, укладається в лапки або подвійні (“ ”), або одинарні (‘ ’).

Функція зчитує рядок із наданого вводу, перетворює його у відповідний рядок (рис. 2.5).

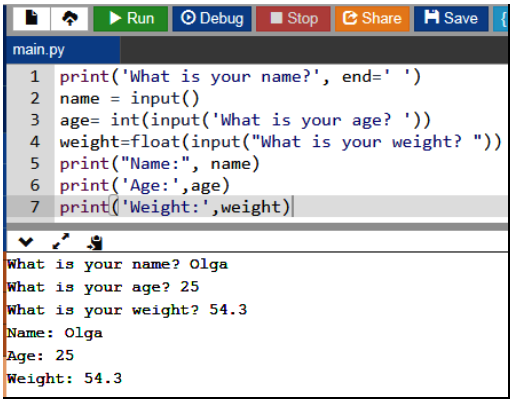


Рисунок 2.5 – Приклад роботи з функцією `input()`

Згідно з рисунком 2.5 введення даних проводиться за таким встановленим сценарієм:

– прямий виклик функції (рядок 2);

– застосування функції перетворення типів даних (рядки 3, 4).

За рисунком 2.5 є можливість використання виводу аргументу підказки (рядки 3, 4) та введення даних на тому ж рядку, що і підказки – результат у блоці виконання програми.

Для приведення до числового типу даних використовується функції `int()` та `float()` (рядок 3,5).

Прості операції з числовими змінними у Python стосуються таких простих арифметичних операцій:

\+ – додавання;

\- – віднімання;

* – множення;

** – зведення в ступінь;

/ – ділення;

// – цілечисельне ділення;

% – остача від ділення.

Інтерпретатор діє як простий калькулятор: при введенні виразу він виводить відповідне значення. Синтаксис виразу простий: оператори +, -, * і / працюють так само, як і в,наприклад, Pascal або C: круглі дужки (()) використовуються для групування.

Цілі числа як 2, 4, 20 мають тип `int`, числа із дробовою частиною такі, як 5.0, 1.6 – тип `float`. На рисунку 2.26 надано приклад роботи простих арифметичних операцій з числами.

```
>>> a = 5; b = 98
>>> c1 = b/a; c2 = b//a; c3 = b%a
>>> c1; c2; c3
19.6
19
3
```

```
1  print(3+2) #5
2  print(5-2) #3
3  print(3*2) #6
4  print(3**2) #9
5  print(9/2) #4.5
6  print(9//2) #4
7  print(11%3) #2
```

```
5
3
6
9
4.5
4
2
```

Рисунок 2.6 – Прості арифметичні операції

Формально в Python 3 `print` є функцією, тому круглі дужки обов'язкові.

Ділення (/) завжди повертає число із плаваючою крапкою (https://ru.wikibooks.org/wiki/Python/Учебник_Python_3.9). Для виконання `floor division` і одержання цілочисельного результату, відкинувши дробову частину, використовують оператор //; для розрахунків залишку від розподілу використовується %:

```
>>> 17 / 3  # Класичне ділення повертає число із плаваючою крапкою
5.666666666666667
>>>
>>> 17 // 3  # floor division відкидає дробову частину
>>> 17 % 3  # Оператор % повертає залишок від ділення
2
>>> 5 * 3 + 2  # результат * дільник + залишок
17
```

У Python для обчислення ступенів використовують оператор ** 1:

```
>>> 5 ** 2  # 5 у квадраті
25
>>> 2 ** 7  # 2 у ступеню 7
128
```

Символ рівності (=) використовується для присвоювання значення змінної. Результат цієї операції перед наступним запрошенням уведення не відображається:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Якщо змінна не «визначена», тобто їй не було привласнене значення, спроба її використання призведе до помилки:

```
>>> n  # Спробувати одержати доступ до невизначеної змінної
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Є повна підтримка чисел із плаваючою крапкою; оператори з операндами змішаного типу перетворять цілочисельний операнд у число із плаваючою крапкою:

```
>>> 4 * 3.75 - 1
14.0
```

В інтерактивному режимі останній виведений результат привласнюється змінній. Це означає, що при використанні Python у якості калькулятора, можна просто відновити обчислення, наприклад:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Ця змінна повинна розглядатися користувачем як доступна тільки для читання. Не варто призначати їй значення явне, бо це призводить б до створення незалежної локальної змінної з таким же іменем, яка маскує вбудовану змінну з її незвичайною поведінкою.

Крім `int` і `float`, Python підтримує інші типи чисел, такі як Decimal і Fraction. Python також має вбудовану підтримку комплексних чисел і використовує суфікс j або J для позначення уявної частини, наприклад, 3+5j.

Отже, Python підтримує всі основі арифметичні оператори, що стосуються операцій з числами (табл. 2.1).

Таблиця 2.1 – Арифметичні операції з числами

Операція	Зміст операції
`x + y`	Додавання –сума x и y (2 + 3 = 5)
`x - y`	Віднімання – різниця x и y (5 – 2 = 3)
`x * y`	Множення – добуток x и y (2 * 3 = 6)
`x / y`	Ділення x на y: 4 / 1.6 = 2.5, 10 / 5 = 2.0, результат завжди типа `float`
`x // y`	Ділення x на y без остачі: 11 // 4 = 2, 11.8 // 4 = 2.0, результат цілий, якщо тільки обидва аргументи цілі
`x % y`	Остача від ділення: 11 % 4 = 3, 11.8 % 4 = 3.8000000000000007 (присутня похибка, що пов'язана з неточністю подання даних в комп'ютері)
`x ** y`	Зведення x в ступінь y: (2 ** 3 = 8)

Перетворення типів даних:

- `int (x)` – перетворює x в ціле число;
- `float (x)` – перетворює x в число з плаваючою точкою;
- `str (x)` – перетворює x в строкове представлення
- `chr (x)` – перетворює ціле x в символ;
- `unichr (x)` – перетворює ціле x в символ Юнікоду;
- `ord (x)` – перетворює символ x до відповідного цілого числа;
- `hex (x)` – перетворює ціле x в шістнадцятковий рядок;
- `oct (x)` – перетворює ціле x в вісімковий рядок.

Приклад перетворення типів даних

Вираз	Результат виконання
`int ("62")`	62
`int (5.04)`	5
`int ("comp 486")`	Помилка
`float (62)`	62.0
`float ("62")`	62.0
`str (62)`	'62'
`str (5.04)`	'5.04'

Взяти до уваги, що операції додавання, віднімання, множення і зведення в ступінь видають відповідь типу `int`, якщо тільки обидва аргументи цілі, типу `float`, якщо один з аргументів дійсний, а інший цілий або

дійсний, і типу `complex`, якщо хоча б один аргумент комплексний. Операція зведення в ступінь також може видати комплексний результат при зведенні негативних чисел в ступінь крім випадку, коли ця ступінь ціла і непарна. Таким чином, зазначені оператори в Python виконують загальновживані правила перетворення типів.

Оператор ділення традиційно є «проблемним»: результат його роботи в різних мовах програмування визначається різними правилами. Для Python версії 3.x ділення розподіл «/» завжди дійсного типу.

Коментування коду розпочинається зі знаку '#' (див. рис. 2.6). Усе, що після знаку решітки, не відображається для користувача програми.

3 ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Завдання 1*:*

Розрахувати площу прямокутника. Під час реалізації програми необхідно враховувати запитання до користувача про введення необхідних даних – сторін *a* та *b*. Вивести результат у вигляді «*S*=».

Завдання 2*:*

Розробити програму, яка привітається з користувачем і знайомиться з ним – запитує про його ім'я *name*. На виході програма виводить речення з таким синтаксисом «Доброго дня, *name*!».

Реалізувати програму з застосуванням в двох методів форматування рядків – використовуючи С-стиль та `.format`.

Завдання 3*:*

Користувач вводить число *a*, а програма розраховує наступне та попереднє число. Результат виконання програми дві строки:

«Попереднє число *a* дорівнює *b*»
«Наступне число *a* дорівнює *b*»

Завдання 4*:*

Перед початком навчального року необхідно розподілити комп'ютерні класи з розрахунку 1 комп'ютер на два студенти. Програма повинна розрахувати скільки комп'ютерів необхідно для заданої кількості студентів у групі, враховуючи такий розподіл: на 3 студента необхідно мати 2 комп'ютера, на 5 студентів – 3 і т.д. На вхід подається кількість студентів, а на виході кількість комп'ютерів.

Завдання 5*:*

Зробити просте повідомлення: збережіть текстове повідомлення в змінній і виведіть його на екран.

Завдання 6*:*

Процитувати відомий вислів: знайдіть відоме висловлювання, яке вам сподобалося. Виведіть текст цитати з іменем автора. Результат повинен виглядати приблизно так, включаючи лапки:

```
Albert Einstein once said, "A person who never made a
mistake never tried anything new."
```

Завдання 7*:*

Відома цитата 2: повторити завдання 5 і 6 за умови збереження ім'я автора цитати в змінній `famous_person`. Скласти повідомлення й зберегти його в новій змінній з іменем `message`. Виведіть своє повідомлення.

Завдання 8*:*

Написати операції додавання, віднімання, множення й ділення, результатом яких є число 8. Укласти операції в команди `print`, щоб перевірити результат. Написати рядки коду, які виглядають приблизно так:

```
print(5 + 3)
```

Результатом повинні бути чотири рядки, у кожному з яких виводиться число.

Завдання 9*:*

Зберегти своє улюблене число в змінній *`my_fav_num`*. За допомогою змінної створити повідомлення для виводу цього числа. Виведіть повідомлення виду: «Моє улюблене число: *`my_fav_num`*».

Основні вимоги до виконання і здачі завдань із лабораторної роботи: обов'язкова присутність коментарів. При відсутності можливості написати коментар, тому що програми були занадто прості, додайте своє ім'я й поточну дату в початок коду. Додайте одну пропозицію з описом того, що робить програма.

ЛАБОРАТОРНА РОБОТА № 2
ПРОСТІ ОПЕРАЦІЇ ПРИ РОБОТІ В PYTHON: ОПЕРАТОРИ ПОРІВНЯННЯ

1 ЗАГАЛЬНІ ВІДОМОСТІ: ОПЕРАТОРИ ПОРІВНЯННЯ ПРИ РОБОТІ В PYTHON

1.1 Вимоги і особливості використання операторів - порівняння

Для порівняння двох величин у мові Python вводяться операції порівняння (відносини). Операції порівняння є бінарні, тобто вимагають двох операндів. Результатом будь-якої операції порівняння є логічне значення `True` або `False`. Значення `True` встановлюється, якщо операція порівняння виконується (істина). Значення `False` встановлюється, якщо операція порівняння не виконується (невірно, неправда) (рис. 1.1).

```
1   nil = 0
2   num = 0
3   max = 1
4   cap = "A"
5   low = "a"
6   print("Equality:", nil ,"==", num, nil==num)
7   print("Equality:", cap ,"==", low, cap==low)
8   print("Inequality:", nil ,"!=", max, nil!=max)
9   print("Greater:", nil ,">", max, nil>max)
10  print("Lesser:", nil ,"<", max, nil<max)
11  print("More or equal:", nil ,">=", num, nil>=num)
12  print("Less or equal:", nil ,"<=", num, nil<=num)
```

```
Equality: 0 == 0 True
Equality: A == a False
Inequality: 0 != 1 True
Greater: 0 > 1 False
Lesser: 0 < 1 True
More or equal: 0 >= 0 True
Less or equal: 0 <= 0 True
```

Рисунок 1.1 – Операції порівняння в Python

Надано перелік операцій порівняння в порядку їх пріоритету:

1) ==, ! = – оператори (операції) перевірки на рівність (найвищий пріоритет):

== – у точності так само;

! = – не дорівнює;

2) <, >, <=, > = – оператори порівняння, відповідно:

> – більше, умова істина, якщо те, що зліва від знака більше того, що справа;

< – менше, умова істина, якщо те, що зліва від знака менше того, що справа;

> = – більше або дорівнює;

<= – менше або дорівнює (рис. 1.2).

```
# Операції порівняння
5 < 6 # п’ять менше шести
a = 8
b = 3
c = a == b # c = False
c = a < b # c = False
c = a!=b # c = True
```

Рисунок 1.2 – Приклади програмного коду, який має операції порівняння

Приклади:

1.1.1 Більше або менше ніж x > y:

```
x > y
x < y
```

Ці оператори порівнюють два типи значень, вони менше й більше ніж оператори. Для чисел це просто порівняння числових значень, щоб побачити, що більше:

```
12 > 4
# True
12 < 4
# False
1 < 4
# True
```

Рядки будуть порівнюватися лексикографічно, що схоже на алфавітний порядок, але не зовсім те ж саме.

```
"alpha" < "beta"
# True
"gamma" > "beta"
# True
"gamma" < "OMEGA"
# False
```

У цих порівняннях малі літери вважаються «більше чому» верхній регістр, тому «gamma» < «OMEGA» є неправильним. Якби всі у верхньому регістрі, то мали результат за абеткою:

```
"GAMMA" < "OMEGA"
# True
```

Кожний тип визначає розрахунки з < і > оператором по-різному, так що необхідно досліджувати, що оператори означають із даним типом перед його використанням.

1.1.2 Не рівно

```
x != y
```

Якщо `x` і `y` не рівні, то це повертає `True`, а якщо ні, то повертає значення `False`:

```
12 != 1
# True
12 != '12'
# True
'12' != '12'
# False
```

1.1.3 Дорівнює

```
x == y
```

Цей вираз використовують, якщо `x` і `y` мають однакове значення й повертає результат як логічне значення. У цілому тип і значення повинні збігатися, тому `int 12` не те ж саме, що рядок '12':

```
12 == 12
# True
12 == 1
# False
'12' == '12'
# True
'spam' == 'spam'
# True
'spam' == 'spam '
# False
'12' == 12
# False
```

Зверніть увагу, що кожний тип при порівнянні повинен визначати функцію, яка буде використовуватися для оцінки, якщо два значення однакові. Для вбудованих типів ці функції поводяться так, як очікують, і просто оцінюють речі, ґрунтуючись на тому ж значенні. Однак користувацькі типи можна визначити тестуванням рівності, щоб вони не хотіли завжди повертати `True` або завжди повертати `False.`

1.2 Порівняння ланцюгів

```
x > y > z
x > y and y > z
```

Це стосується порівняння кілька елементів з декількома операторами порівняння за допомогою ланцюжка порівняння.

У мові Python є можливість створювати ланцюжки із декількох операцій порівняння. Отримують ланцюг операцій типу `a < b < c`, що неявно перетворюється в форму, в якій кожна операція надається звичайним чином, а між ними використано оператор `and`:

```
a < b and b < c.
```

Наприклад:

```
# Складові операції порівняння
a = 5
b = 7
c = a < b < 8 # c = True
c = a != b != 8 # c = True
c = a == b == 8 # c = False
```

Це дозволить оцінити `True`, тільки якщо обидва порівняння `True`.

Загальна форма

```
a OP b OP c OP d ...,
```

де `OP` – один з декількох операцій порівняння, які можна використовувати, а літери є довільні дійсні вирази.

Немає теоретичного обмеження на кількість елементів і операцій порівняння, якщо використовується правильний синтаксис:

```
1 > -1 < 2 > 0.5 < 100 != 24
```

Наведене вище повертає `True`, якщо кожне порівняння повертає `True`. Однак, використання складного ланцюжка не дуже добрий стиль. Хороший ланцюжок буде «спрямованим», не більше складним, чим

```
1 > x > -4 > y != 8
```

Побічні ефекти: як тільки одне порівняння повертає значення `False`, вираз відразу обчислює значення `False`, пропускаючи всі інші порівняння. Наприклад, вираз

```
exp в a > exp > b
```

буде оцінюватися тільки один раз. У той же час для виразу

```
a > exp and exp > b
```

`exp` буде розраховуватися два рази, якщо `a > exp` вірно.

Порівняння за виразом `is` vs` == `

Вважаються заплутаними оператори порівняння рівності `is i ==`. Так, `a == b` порівнює значення `i a b`; `is b` порівнює тотожності `i a b`.

Наприклад:

```
a = 'Python is fun!'
b = 'Python is fun!'
a == b # returns True
a is b # returns False

a = [1, 2, 3, 4, 5]
b = a      # b references a
a == b     # True
a is b     # True
b = a[:]   # b now references a copy of a
a == b     # True
a is b     # False [!!]
```

Взагалі, `is` можна розглядати як скорочення для `id(a) == id(b)` .

Існують особливості середовища порівняння, які ускладнюють ситуацію. Короткі рядки і невеликі цілі числа будуть повертати `True`, в порівнянні з `is`, із-за машини Python намагається використовувати менше пам'яті для однакових об'єктів `a = 'short'`:

```
b = 'short'
c = 5
d = 5
a is b # True
c is d # True
```

Більш довгі рядки і великі цілі числа будуть зберігатися окремо:

```
a = 'not so short'
b = 'not so short'
c = 1000
d = 1000
a is b # False
c is d # False
```

Щоб перевірити на `None` , необхідно використати `is` :

```
if myvar is not None:
    # not None
    pass
    if myvar is None:
        # None
        pass
```

Операція з `is` – це перевірка на «дозорних», унікальних об'єктах:

```
sentinel = object()
def myfunc(var=sentinel):
    if var is sentinel:
        # value wasn't provided
        pass
    else:
        # value was provided
        pass
```

Частіше операції порівняння зустрічаються в операторах, де відбувається перевірка умови (`if, while`) і розв'язок задачі залежить від виконання або невиконання деякої умови.

Команда `if` у мові Python дозволяє перевірити поточний стан програми й вибрати подальші дії залежно від результатів перевірки.

У кожній команді `if` центральне місце займає вираз, результатом якого є логічна істина `True` або логічна неправда `False`. Такий вираз називається *умовою*. Залежно від результату перевірки Python вирішує, чи повинен виконуватися код у команді `if`. Якщо результат умови дорівнює `True`, то Python виконує код, що випливає за командою `if`.

Перевірка рівності. У багатьох умовах поточне значення змінної порівнюється з конкретним значенням. Найпростіша умова перевіряє, чи дорівнює значення змінної конкретній величині:

```
❶ >>> car = 'bmw'
❷ >>> car == 'bmw'
  True
```

У рядку 1 змінній `car` надається значення `'bmw'`; операція виконується одним знаком `=`. Рядок 2 перевіряє, чи дорівнює значення `car` рядку `'bmw'`; для перевірки використовується подвійний знак рівності (`= =`).

Цей оператор повертає `True`, якщо значення ліворуч і праворуч від оператора рівні; якщо ж значення не збігаються, оператор повертає `False`. У прикладі значення збігаються, тому Python повертає `True.`

Якщо `car` ухвалює будь-яке інше значення замість `'bmw'`, перевірка повертає `False`:

```
❶ >>> car = 'audi'
❷ >>> car == 'bmw'
  False
```

Одиночний знак рівності виконує операцію; код 1 можна прочитати у формі «`Надати car значення 'audi'`». З іншого боку, подвійний знак

рівності (рядок 2) ставить запитання: «Значення car рівно 'bmw'?»

У мові Python перевірка рівності виконується з урахуванням регістру значень. Наприклад, не вважаються рівними два значення з різним регістром символів:

```
>>> car = 'Audi'
>>> car == 'audi'
False
```

Якщо регістр символів важливий, така поведінка приносить користь. За умови, що перевірка повинна виконуватися на рівні символів без обліку регістру, необхідно привести значення змінної до нижнього регістру перед виконанням порівняння:

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Умова повертає True незалежно від регістру символів 'Audi', тому що перевірка тепер виконується без обліку регістру. Функція lower() не змінює значення, яке завжди зберігалося в car, так що порівняння не відбивається на вихідній змінній:

```
❶ >>> car = 'Audi'
❷ >>> car.lower() == 'audi'
  True
❸ >>> car
  'Audi'
```

У точці 1 рядок 'Audi' зберігається в змінній car. У точці 2 значення car приводиться до нижнього регістру й порівнюється зі значенням рядка 'audi', записаним також у нижньому регістрі. Два рядки збігаються, тому Python повертає True. Висновок у точці 3 показує, що значення, яке зберігається в car, не змінилося в результаті перевірки.

1.3 Логічні оператори

За допомогою логічних операторів: not (логічне НІ), or (логічне АБО), and (логічне І) будують складні (складові) логічні вирази:

• x and y – логічне «І» (множення); ухвалює значення True (істина), тільки коли x = True і y = True; ухвалює значення False (неправда), якщо хоча б одна зі змінних рівна False, або обидві змінні False;

• x or y – логічне «АБО» (додавання); ухвалює значення True (істина), якщо хоча б одна зі змінних рівна True, або обидві змінні True; ухвалює значення False (неправда), якщо x = = y = = False;

• `not x` – логічне «НІ» (заперечення); ухвалює значення `True` (істина), якщо `x = = False`; ухвалює значення `False` (неправда), якщо `x = = True` (табл. 1.1).

Таблиця 1.1 – Таблиці істинності логічних функцій

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	False
False	False	False	False

Правила роботи логічних операторів задається за допомогою таблиць істинності, у яких вказується істинність складеного виразу відповідно до значень вихідних простих відносин між даними:

A	not A
`False`	`True`
`True`	`False`

Слід зазначити, що логічні операції в Python визначені для об'єктів будь-яких типів, але результати таких операцій для операторів `and` і `or` не завжди легко зрозумілі. Оператор `not` працює досить просто, він приводить аргумент до логічного значення за визначеними для нього правилами і видає завжди тільки логічне значення:

```
>>> [1, 2] and [1] and 13
13
>>> [1, 2] and [1] or 13
[1]
```

У прикладі всі три об'єкти: [1, 2], [1] і 13 будуть інтерпретуватися як істина, оскільки списки не порожні, а число не дорівнює нулю. Однак, у першому випадку в результаті обчислення виразу вийде число, а в другому – один зі списків. Отже, рекомендовано програмістам не використовувати логічні оператори у виразах з нелогічними об'єктами, або перетворювати ці об'єкти до логічного типу прямо за допомогою функції `bool()`:

```
>>> bool([1 , 2]) and bool ([1]) or bool (13)
True
>>> bool([1 , 2]) and bool ([1]) and bool (13)
True
```

Логічні оператори `and` і `or` в Python використовуються суттєво рідше, чим у інших мовах програмування, як Java, C/C++ або Pascal/Delphi, тому що в Python можна робити будь-які подвійні, потрійні та інші порівняння, наприклад, `a < x < b` еквівалентно `(x > a) and (x <b):`

```
1   a=True
2   b=False
3   print("And Logic:")
4   print(a and a)
5   print(a and b)
6   print(b and b)
7   print("OR Logic:")
8   print(a or a)
9   print(a or b)
10  print(b or b)
11  print("NOT Logic:")
12  print(not a)
13  print(not b)
```

```
And Logic:
True
False
False
OR Logic:
True
True
False
NOT Logic:
False
True
```

1.4 Команди `if` та умовна інструкції `if-elif-else`

Проста форма команди `if` складається з одного умови і одного дії:

```
if умова:
    дія
```

У першому рядку розміщується умова, а в блоці з відступом – практично будь-який дія. Якщо умова істинно, то Python виконує код в блоці після команди `if`, а якщо помилково, цей код ігнорується.

Відступи в командах `if`: якщо умова істина, то всі рядки з відступом після команди `if` виконуються; якщо помилково – весь блок з відступом ігнорується. Блок команди `if` може містити скільки завгодно рядків:

voting.py

```
age = 19
❶ if age >= 18:
❷     print("You are old enough to vote!")
```

```
age = 19
if age >= 18:
print("You are old enough to vote!")
    print("Have you registered to vote yet?")
```

Умова виконується, обидві команди `print` забезпечені відступом, тому виводяться обидва повідомлення:

```
You are old enough to vote!
Have you registered to vote yet?
```

Якщо значення `age` менше 18, програма нічого не виводить.

Команди `if-else`: при необхідності у програмі виконати одну дію в тому випадку, якщо умова істинно, і деякі інші дії, якщо воно помилкове. З синтаксисом `if-else` це можливо. Блок `if-else` у цілому схожий на команду `if` і додатково секція `else` визначає дію або набір дій, які виконуються при невдалій перевірці. Синтаксис інструкції `if-else` має вигляд

```
if test1:
    state1
else:
    state2
```

У кінці строки `if` і `else` ставиться двокрапка ":"Перед `state1, state2, … , state N` обов'язково повинно бути 4 пробіли або табуляція:

```
1  a=5
2  c=0
3  if a==5:
4      print("a=",a)
5  else:
6      print("No")
7      c=a
8  print("c=",c)
```

```
a= 5
c= 0
```

Тримісний вираз, що має конструкцію

```
if X:
    A=Y
else:
    A=Z
```

можна записати таким чином:

```
A=Y if X else Z
```

Для прикладу реалізації команди `if-else` розглядається повідомлення стосовно віку достатнього для голосування з позначенням умов при будь-якому іншому віці виведення іншого повідомлення:

```
age = 17
❶ if age >= 18:
print("You are old enough to vote!")
print("Have you registered to vote yet?")
❷ else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

Якщо умова 1 істинно, то виконується перший блок з командами `print`. Якщо ж умова помилкова, виконується блок `else` у точці 2. Так як значення `age` менше 18, умова виявляється помилковою, і виконується код в блоці `else`:

```
Sorry, you are too young to vote.
Please register to vote as soon as you turn 18!
```

Цей код працює, тому що існують лише дві можливі ситуації: вік або достатній для голосування, або недостатній. Структура `if-else` добре підходить для тих ситуацій, в яких Python завжди виконує тільки одне з двох можливих дій. У подібних простих ланцюжках `if-else` завжди виконується одна з двох можливих дій.

Ланцюжки `if-elif-else`: перевірка більше двох можливих ситуацій; Python виконує тільки один блок в ланцюжку `if-elif-else`. Усі умови перевіряються за порядком до тих пір, поки одна з них не дасть правильний результат. Далі виконується код за цією умовою, а всі інші перевірки Python пропускає. У багатьох реальних ситуаціях існують більше двох можливих результатів. Синтаксис інструкції `if-elif-else` має вигляд

```
if test1:
    state1
elif test2:
    state2
elif test3:
    state3
……
else:
    stateN
```

У кінці строки `if, elif` та `else` завжди ставиться двокрапка ":"

Перед `state1, state2, … , stateN` обов'язково повинно бути 4 пробіли або табуляція.

Наприклад, оплата атракціонів у парку, де передбачено різну плату за вхід для різних вікових груп:

- для відвідувачів молодше 4 років вхід безкоштовний;
- для відвідувачів від 4 до 18 років квиток коштує $ 5;
- для відвідувачів від 18 років і старше квиток коштує $ 10.

Для встановлення плати за вхід за використанням команди `if` наступний код визначає, до якої вікової категорії належить відвідувач, і виводить інформацію стосовно вартості квитка для нього:

amusement_park.py

```
age = 12

❶ if age < 4:
      print("Your admission cost is $0.")
❷ elif age < 18:
      print("Your admission cost is $5.")
❸ else:
      print("Your admission cost is $10.")
```

Умова `if` у точці 1 перевіряє, що вік відвідувача менше 4 років.

За умови істинно програма виводить відповідне повідомлення, і Python пропускає інші перевірки.

Рядок `elif` у точці 2 насправді є ще однією перевіркою `if`, яка виконується тільки в тому випадку, якщо попередня перевірка помилкова. За умови завдання вік відвідувача більше 4 років, тому необхідна ще перевірка. Отже, відвідувачеві менше 18 років, і програма виводить відповідне повідомлення, Python пропускає блок `else.`

При помилковості обох умов – `if` і `elif,` Python виконує код в блоці `else` у точці 3. У даному прикладі умова 1 дає помилковий результат, тому його блок не виконується. Друга умова виявляється істиною (12 менше 18), тому код буде виконаний. Висновок складається з одного повідомлення з ціною квитка:

```
Your admission cost is $5.
```

При будь-якому значенні віку більше 17 перші дві умови помилкові. У таких ситуаціях блок `else` виконується, і ціна квитка складе $ 10.

Замість виведення відомості з ціною квитка в блоках `if-elif-else`, застосовують більш компактне рішення: присвоїти ціну в ланцюжку `if-elif-else`, додати одну команду `print` після виконання ланцюжка:

```
age = 12

if age < 4:
❶     price = 0
elif age < 18:
❷     price = 5
else:
❸     price = 10

❹ print("Your admission cost is $" + str(price) + ".")
```

Рядки 1, 2 і 3 привласнюють значення `price` в залежності від значення `age`, як і в попередньому прикладі. Після присвоювання ціни в ланцюжку `if-elif-else` окрема команда `print` без відступу в 4 використовує це значення для виведення повідомлення з ціною квитка.

Цей приклад виводить той же результат, що і попередній, але ланцюжок `if-elif-else` має більш чітке завдання, упорядковане стосовно подання інформації. Відсутні зайві результати, виводиться повідомлення, що визначає ціну квитка. Поряд з підвищенням ефективності цього коду реалізується найкраща змінність. Для зміни тексту вихідного повідомлення редагують всього одну команду `print` – замість трьох різних команд.

Код може містити скільки завгодно блоків `elif`. Python не вимагає, щоб ланцюжок `if-elif` неодмінно завершувалася блоком `else`.

Блок `else` зручний, але іноді наочніше використовувати додаткову секцію `elif` для обробки конкретної умови:

```
age = 12

if age < 4:
price = 0
elif age < 18:
price = 5
elif age < 65:
price = 10
❶ elif age >= 65:
price = 5

print("Your admission cost is $" + str(price) + ".")
```

Блок `elif` у точці 1 призначає ціну $ 5, якщо вік відвідувача дорівнює 65 і вище. Зміст такого коду зрозуміліший, ніж у узагальненого блоку `else`. За такими змінами виконання кожного блоку можливо тільки при істинності конкретної умови.

Блок `else` «універсальний»: він обробляє всі умови, що не підходять ні під жодну конкретну перевірку `if` або `elif`, причому в цю категорію іноді можуть потрапити недійсні або навіть шкідливі дані. Якщо є завершальна конкретна умова, краще використати завершальний блок `elif` і опустити блок `else`. У цьому випадку є впевненість у тому, що ваш код буде виконуватися тільки в правильних умовах.

Ланцюжки `if-elif-else` ефективні тільки в тому випадку, якщо істинним має бути тільки одна умова. При необхідності перевірки умов слід застосовувати як серії простих команд `if` без блоків `elif`, `else`. Та-

ке рішення доречно, коли істинними можуть бути відразу декілька умов і доцільно відреагувати на всі наявні умови.

Так, наприклад, при замовленні піци додають два доповнення, що необхідно ввести до програми:

toppings.py

```
❶ requested_toppings = ['mushrooms', 'extra cheese']

❷ if 'mushrooms' in requested_toppings:
      print("Adding mushrooms.")
❸ if 'pepperoni' in requested_toppings:
      print("Adding pepperoni.")
❹ if 'extra cheese' in requested_toppings:
      print("Adding extra cheese.")

  print("\nFinished making your pizza!")
```

Обробка починається в точці 1 зі списку, що містить замовлені доповнення. Команди `if` у точці 2 і 3 перевіряють, чи включає замовлення конкретні додатки – наявність у виробі грибів і пеперонів. Якщо включає, то виводяться підтвердження.

Перевірка в точці 3 реалізована простою командою `if`, а не `elif` або `else`, бо умова буде перевірятися незалежно від того, чи було попереднє умова істинна чи хибна. Код у точці 4 перевіряє, чи була замовлена додаткова порція сиру, незалежно від результату перших двох перевірок. Ці три незалежні умови перевіряються при кожному виконанні програми.

Так як у цьому коді перевіряються всі можливі варіанти доповнень, то в замовлення будуть включені два доповнення з трьох:

```
Adding mushrooms.
Adding extra cheese.

Finished making your pizza!
```

При використанні в програмі блоку `if-elif-else` код працює неправильно, тому що він перериває роботу після виявлення першої справжньої умови, тобто мали б такий результат:

```
requested_toppings = ['mushrooms', 'extra
          cheese']

if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
elif 'pepperoni' in requested_toppings:
```

```
    print("Adding pepperoni.")
elif 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")
```

Перша ж перевірена умова для `'mushrooms'` виявляється істинною, тому значення `'extra cheese'` і `'pepperoni'` після цього не перевіряються. Отже, в ланцюжках `if-elif-else` після виявлення першої справжньої умови всі інші умови пропускаються. В результаті в піцу буде включено тільки перше з замовлених доповнень:

```
Adding mushrooms.
Finished making your pizza!
```

Таким чином, якщо необхідно в програмі виконання лише одного блоку коду, то використовують ланцюжок `if-elif-else`. Якщо ж треба врахувати всі доповнення, то повинні виконуватися кілька блоків і використовуватися серія незалежних команд `if`.

1.5 Пріоритет операцій у Python

Операції над об’єктами виконуються в конкретному порядку:

1. `**`.
2. `-x, +x`.
3. `* , / , // , %`.
4. `+, -`.
5. `<, <=, >, >=, !=, ==`.
6. `is, is not`.
7. `in, not in`.
8. `not, and, or`.

Зміни порядку відбуваються за рахунок використання скобок, наприклад: (5 + 2) * 3 чи (3 * 2)**3 (табл. 1.2).

Таблиця 1.2 – Пріоритет операцій у Python

Оператор	Назва	Пояснення
**	Зведення в ступінь	Повертає число `x`, зведене в ступінь `y`
+	Додавання	Підсумовує два об’єкти
-	Віднімання	Дає різницю двох чисел; якщо перший операнд відсутній, то він вважається рівним нулю

Кінець таблиці 1.2

Оператор	Назва	Пояснення
*	Множення	Дає результат множення двох чисел або повертає рядок, повторений задане число раз.
/	Ділення	Повертає частка від ділення x на y
//	Цілочисельне - ділення	Повертає неповне частка від ділення
%	Розподіл по модулю	Повертає залишок від ділення
<<	Зрушення вліво	Зрушує біти числа вліво на задану кількість позицій. Будь-яке число в пам'яті комп'ютера надано у вигляді бітів або двійкових чисел, тобто 0 і 1.
>>	Зрушення вправо	Зрушує біти числа вправо на задане число позицій.
&	Побітове І	Побітова операція І над числами
|	Побітове АБО	Побітова операція АБО над числами
^	Побітове ВИКЛЮЧНО АБО	Побітова операція ВИКЛЮЧНО АБО
~	Побітове НІ	Побітова операція НІ для числа x відповідає визначенню (x + 1)
<	Менше	Визначає, чи вірно, що x менше y. Усі оператори порівняння повертають True або False.
>	Більше	Визначає, чи вірно, що x більше y
<=	Менше або дорівнює	Визначає, чи вірно, що x менше або дорівнює y
>=	Більше чи рівно	Визначає, чи вірно, що x більше або дорівнює y
==	Так само	Перевіряє, чи однакові об'єкти
!=	Не дорівнює	Перевіряє, чи вірно, що об'єкти не рівні
not	Логічне НІ	Якщо x одно True, то оператор поверне False. Якщо ж x одно False, то отримаємо True.
and	Логічне І	x and y дає False, якщо x одно False, то в іншому випадку повертає значення y
or	Логічне АБО	Якщо x одно True, то в результаті отримаємо True, в іншому випадку отримаємо значення y

2 ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Завдання 1:

На вхід програми подається три числа. Визначити скільки серед

них однакових. На виході програма повинна вивести такі значення: 3 – усі числа співпадають, 2 – співпало 2 числа та 0 – усі числа різні.

Завдання 2:

Написати програму для пошуку найбільшого числа з двох введених. Вивести найбільше число.

Завдання 3:

Користувач вводить число від 1 до 7. Вивести назву дня відповідно до залежності 1 – понеділок, 2 – вівторок і т.д. Врахувати можливість вводу некоректних даних.

Завдання 4:

Користувач вводить будь-яке значення у програму. На виході програма повинна надати результат, що зазначений користувачем: «Це ціле число!», «Це дійсне число!», «Можливо введено текст!»

Примітка: вбудовані рядкові методи не можна використовувати.

Завдання 5:

На цей час Американська асоціація дієтологів визнає таку формулу підрахунку необхідної кількості кілокалорій, що є найбільш точною серед схожих рівнянь. Це формула Миффлина-Сан-Жеора, що має вигляд:

- для жінок: (10 × вага в кілограмах) + (6,25 × зріст у сантиметрах) - (5 × вік у літах) – 161;
- для чоловіків: (10 × вага в кілограмах) + (6,25 × ріст у сантиметрах) – (5 × вік у літах) + 5.

Формула враховує й фізичну активність, виходячи з якої до результату, що вийшов, додається коефіцієнт при таких умовах;

- якщо у вас немає фізичних навантажень і сидяча робота, то помножте отриманий результат на 1,2;
- якщо ви робите невеликі пробіжки або робите легку гімнастику 1–3 рази в тиждень, то помножте результат на 1,375;
- якщо ви займаєтеся спортом із середніми навантаженнями 3–5 раз на тиждень, то помножте кількість необхідних калорій на 1,55;
- якщо ви повноцінно тренуєтеся 6–7 раз на тиждень, то вам необхідно помножити результат на 1,725;
- якщо ваша робота пов’язана з фізичною працею, ви тренуєтеся 2 рази в день і включаєте в програму тренувань силові вправи, то ваш коефіцієнт корегування буде рівний 1,9.

Отримати результат у вигляді рекомендації вживаних кілокалорій відповідно до статі, наявних розмірів (вага, зріст, вік) і умов життєдіяль-

ності. *При здачі для отримання додаткових балів пропонується додатково внести корективи стосовно навантаження при наявних «негараздах» із дотримання норм ваги за умови (зріст – вага) = чи < фактичної ваги.*

Завдання 6:

Створити програму, де необхідно вивести позитивне твердження умови, коли число X є парним і менше 100.

Завдання 7:

Напишіть ланцюжок `if-elif-else` для визначення періоду життя людини. Надайте значення змінній `age` і виведіть результат за таких умов:

• якщо значення менше 2 – немовля;

• якщо значення більше або дорівнює 2, але менше 4 – малюк;

• якщо значення більше або дорівнює 4, але менше 13 – дитина;

• якщо значення більше або дорівнює 13, але менше 20 – підліток;

• якщо значення більше або дорівнює 20, але менше 65 – дорослий;

• якщо значення більше або дорівнює 65 – літня людина.

Завдання 8-10 виконайте відповідно до номеру варіантів, які вказані у дужках.

Завдання 8 (1, 4, 7, 10, 13, 16, 19):

Створіть змінну з ім'ям `alien_color` і надайте їй значення `'green'`, `'Yellow'` або `'red'`.

1. Напишіть команду `if` для перевірки того, що змінна містить значення `'green'`.

Якщо умова істинно, виведіть повідомлення про те, що ви тільки що заробили 5 очок.

2. Напишіть одну версію програми, в якій умова `if` виконується, і іншу версію, в якій вона не виконується. (У другій версії ніяке повідомлення виводитися не повинно.)

Завдання 9 (2, 5, 8, 11, 14, 17, 20):

Створіть змінну з ім'ям `alien_color` і надайте їй значення `'green'`, `'Yellow'` або `'red'`.

Виберіть колір і напишіть ланцюжок `if-else`.

1. Напишіть команду `if` для перевірки того, що змінна містить значення `'green'`.

Якщо умова істинно, виведіть повідомлення про те, що ви тільки що заробили 5 очок.

Якщо змінна містить будь-яке інше значення, виведіть повідомлення про те, що ви тільки що заробили 10 очок.

2. Напишіть одну версію програми, в якій виконується блок `if`, і іншу версію, в якій виконується блок `else`.

Завдання 10 (3, 6, 9, 12, 15, 18, 21):

Створіть змінну з ім'ям alien_color і надайте їй значення `'green'`, `'Yellow'` або `'red'`.

Перетворіть ланцюжок if-else з попереднього завдання в ланцюжок if-elif-else.

• Якщо змінна містить значення 'green', виведіть повідомлення про те, що гравець тільки що заробив 5 очок.

• Якщо змінна містить значення 'yellow', виведіть повідомлення про те, що гравець тільки що заробив 10 очок.

• Якщо змінна містить значення 'red', виведіть повідомлення про те, що гравець тільки що заробив 15 очок.

• Напишіть три версії програми і простежте за тим, щоб для кожного кольору виводилося відповідне повідомлення.

***Завдання 12**

1. Перевірка умов: напишіть послідовність умов. Виведіть опис кожної перевірки і ваш прогноз щодо її результату. Код повинен виглядати приблизно так:

```
car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')
```

Уважно перегляньте результати. Переконайтеся в тому, що ви розумієте, чому результат кожного рядка дорівнює `True` або `False`.

• Створити як мінімум 10 умов. Не менше 5 повинні давати результат `True`, а не менше 5 інших – результат `False`.

2. Написати умови і включити їх в `conditional_tests.py`. Програма повинна видавати один істинний і один помилковий результат для таких видів перевірок.

• Перевірка рівності і нерівності рядків.

• Перевірки з використанням функції `lower ().`

• Числові перевірки рівності і нерівності, умов «більше», «менше», «більше або дорівнює »,« менше або дорівнює ».

- Перевірки з ключовим словом `and` і `or`.
- Перевірка входження елемента в список.
- Перевірка відсутності елементу в списку.

Основні вимоги до виконання і здачі завдань із лабораторної роботи: при написанні коду використовувати тільки конструкцію `if…elif...else`; виконати завдання і надати результати роботи розроблених програм; у висновку визначити, що за функції забезпечують оператори порівняння при розв'язку конкретних завдань за умови використання (`if, while`) (підтвердити теоретичні знання з лекційного матеріалу), як то: *при розв'язку завдань* 1–4 *використано* `if` *для позначення …; при розробці програми для завдання* 5 *оператори (*`if, while`*) забезпечують ….*

ЛАБОРАТОРНА РОБОТА № 3
РОБОТА З ДАНИМИ: СПИСКИ, РЯДКИ, МНОЖИНИ, СЛОВНИКИ

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО ОПЕРАЦІЇ ЗІ СПИСКАМИ, РЯДКАМИ, МНОЖИНАМИ, СЛОВНИКАМИ

1.1 Особливості виконання основних операцій зі списками

Списки в мові програмування Python, як і рядки, є упорядкованими послідовностями значень. Однак, на відміну від рядків, списки містять не символи, а різні об'єкти (значення, дані), які включають не в лапки, а у квадратні дужки [ ]. Об'єкти відділяються один від одного за допомогою коми. Списки можуть складатися з різних об'єктів: чисел, рядків і навіть інших списків. В останньому випадку, списки називають вкладеними.

Отже, списки в Python є послідовностями , які здатні до мутації, тобто їх елементи можуть змінюватися. От деякі приклади списків:

```
[159 , 152 , 140 , 128 , 113] #список цілих чисел
[15.9 , 15.2 , 14.0 , 128. , 11.3] # список дійсних чисел
['Даша', 'Катя', 'Ксюша'] #список строк
['Саратов', 'Астраханская', 104 , 1 змішаний список
[[1 , 0, 0], [0, 1, 0], [0, 0, 1]] #список списків
```

Над списками виконують операції об'єднання і/та дублювання:

```
>>> [45, -12, 'april'] + [21, 48.5, 33]
 [45, -12, 'april', 21, 48.5, 33]
>>> [[0,0],[0,1],[1,1]] * 2
 [[0, 0], [0, 1], [1, 1], [0, 0], [0, 1], [1, 1]]
```

Доступ до об'єктів списку отримують за їх індексами. Доступним є отримання зрізу, вимірювання довжину списку:

```
>>> li = ['a','b','c','d','e','f']
>>> len(li)
6
>>> li[4]
'e'
>>> li[:3]
 ['a', 'b', 'c']
```

У списках видаляють елементи, додають нові відповідно до того, як оперують змінними. Допускаються ситуації, коли списки копіюють.

Наприклад, результат операції присвоюється інший змінній. При виконанні операцій над списками можна не створювати нові, а змінювати безпосередньо оригінал. Списки – це змінні послідовності. Зі списків можна видаляти елементи, додавати нові, змінювати існуючі:

```
>>> mylist = ['ab','ra','ka','da','bra']
>>> mylist[1] = 'ro'
>>> mylist
 ['ab', 'ro', 'ka', 'da', 'bra']
>>> mylist[0:2] = [10, 20]
>>> mylist
 [10, 20, 'ka', 'da', 'bra']
>>> del(mylist[0])
>>> mylist
 [20, 'ka', 'da', 'bra']
>>> mylist = mylist + [20, 30]
>>> mylist
 [20, 'ka', 'da', 'bra', 20, 30]
```

У списку можна замінити цілий зріз. При присвоєнні списку більше, ніж одній змінній відбудуться зміни не тільки у списку в одному місці, а й зміни його в інших місцях.

Значення списку копіюють в незалежний новий список за допомогою функції `copy()`; функції `list()`; розділенням списку за допомогою [:].

Наприклад, список присвоєно змінній `b`, списки `a, c, d` – це копії списку `b`:

```
>>> b = [1, 2, 3]
>>> a = b.copy()
>>> c = list(b)
>>> d = b [:]
```

Надані у списку `a, c, d` – це нові об'єкти зі своїми значеннями, що не пов'язані з оригінальним списком елементів [1, 2, 3], на який посилається змінна `b`. Зміни в `b` не впливають на копії `a, c, d`.

При створенні списку використовують два способи: вбудовану функцію `list();` за допомогою літерала `[]`.

Для звернення до елементу списку використовують ім'я списку та індекс у квадратних скобках. Нумерація починається з 0.

Створення списку, звернення до елементу та виведення типу елемента. Для створення списку укладають в квадратних дужках перелік елементів, розділених комами:

```
# Змінній x присвоюється список з трьох елементів
x = [1, 2, 3]
```

Не потрібно оголошувати список або заздалегідь фіксувати його розмір: створюється список, він присвоюється змінній. Список автоматично збільшується чи скорочується в міру необхідності.

Списки Python містять елементи різних типів; елементом списку може бути будь-який об'єкт Python. Наприклад, наданий список містить різнорідні елементи:

```
# Перший елемент – число, другий – рядок, третій – інший список.
x = [2, "two", [1, 2, 3]]
```

Основною вбудованою функцією списків є функція `len`, яка повертає кількість елементів в списку:

```
>>> x = [2, "two", [1, 2, 3]]
>>> len (x)
3
```

Зверніть увагу: функція `len` не враховує елементи внутрішнього вкладеного списку.

Для модифікації списків і для вилучення з них окремих елементів використовують синтаксис індексування. Індекс вказують в лівій частині оператора присвоювання:

```
>>> x = [1, 2, 3, 4]
>>> x [1] = "two"
>>> x
[1, 'two', 3, 4]
```

У цьому випадку використовується синтаксис сегментів. Команда виду `lista [index1: index2] = listb` замінює всі елементи `lista` між `index1 i index2` елементами з `listb.` Список `listb` може містити більше або менше елементів, ніж видаляється з `lista`; у цьому випадку довжина `lista` змінюється. Сегментне присвоювання використовується для виконання різних операцій, що виглядає таким чином:

```
>>> x = [1, 2, 3, 4]
>>> x [len (x):] = [5, 6, 7] ← приєднує список до кінця списку
>>> x
[1, 2, 3, 4, 5, 6, 7]
>>> x [: 0] = [-1, 0]        ← приєднує список до початку списку
>>> x
[-1, 0, 1, 2, 3, 4, 5, 6, 7]
```

```
>>> x [1: -1] = []                ← видаляє елементи зі списку
>>> x
[-1, 7]
```

Приєднання одного елемента до списку є часто використаною операцією, тому для неї був визначений спеціальний метод `append:`

```
>>> x = [1, 2, 3]
>>> x.append ( "four")
>>> x
[1, 2, 3, 'four']
```

При спробі приєднання одного списку до іншого список буде приєднаний як один елемент основного списку:

```
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7]
>>> x.append (y)
>>> x
[1, 2, 3, 4, [5, 6, 7]]
```

Для додавання елементів одного списку до іншого застосовують метод `extend`, схожий на метод `append`:

```
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7]
>>> x.extend (y)
>>> x
[1, 2, 3, 4, 5, 6, 7]
```

Новий елемент списку вставляється між двома існуючими елементами або в початок списку за спеціальним методом `insert`. Метод `insert` використовується як метод списків і отримує два додаткових аргументи. Перший додатковий аргумент визначає індекс позиції списку, в яку буде вставлений новий елемент, а другий – сам новий елемент:

```
>>> x = [1, 2, 3]
>>> x.insert (2, "hello")
>>> print (x)
[1, 2, 'hello', 3]
>>> x.insert (0, "start")
>>> print (x)
[ 'Start', 1, 2, 'hello', 3]
```

Метод `insert` інтерпретує індекси, але в більшості випадків простіше застосовувати виклик `list.insert (n, elem)` як вставку `elem` перед *n*-м елементом списку `list.`

Метод `insert` є зручним, його функції можна реалізувати за допомогою присвоювання сегментів – конструкція `list.insert (n, elem)` еквівалентна `list [n: n] = [elem]`, де `n` – не негативне число . Використання методу `insert` спрощує читання коду, підтримує негативні індекси:

```
>>> x = [1, 2, 3]
>>> x.insert (-1, "hello")
>>> print (x)
[1, 2, 'hello', 3]
```

Для видалення елементів списків або сегментів рекомендується використовувати команду `del`. Ця команда може бути замінена на операцію присвоювання сегментів, але вона краще запам'ятовується і легко з нею розпізнати об'єкти коду:

```
>>> x = [ 'a', 2, 'c', 7, 9, 11]
>>> del x [1]
>>> x
[ 'A', 'c', 7, 9, 11]
>>> del x [: 2]
>>> x
[7, 9, 11]
```

Загалом команда `del list [n]` еквівалентна `list [n: n + 1] = []`, а команда `del list [m: n]` робить те, що `list [m: n] = []`.

За методом `remove` здійснюється пошук першого екземпляра заданого значення в списку і видалення цього значення зі списку:

```
>>> x = [1, 2, 3, 4, 3, 5]
>>> x.remove (3)
>>> x
[1, 2, 4, 3, 5]
>>> x.remove (3)
>>> x
[1, 2, 4, 5]
>>> x.remove (3)
Traceback (innermost last):
File "<stdin>", line 1, in?
ValueError: list.remove (x): x not in list
```

Використовуючи засоби обробки винятків Python, можна перехопити помилку або ж запобігти саму проблему, перевіряючи наявність елементів в списку оператором `in` перед тим, як їх видалити.

Метод `reverse` – спеціалізований метод зміни списку для ефективної перестановки елементів списку в зворотному порядку «на місці»:

```
>>> x = [1, 3, 5, 6, 7]
>>> x.reverse ()
>>> x
[7, 6, 5, 3, 1]
```

Для доступу до будь-якого елементу списку враховують, що списки є упорядкованими наборами даних, тому слід повідомити Python позицію (індекс) потрібного елемента. Індекси приймають тільки значення цілого числа. При зверненні до елементу у списку, необхідно вказати ім'я списку, а потім індекс елемента в квадратних дужках.

Зважати, що індекси починаються з 0, а не з 1. Цей принцип зустрічається у більшості мов програмування, у мові програмування Python він пояснюється особливостями низького рівня реалізації операцій зі списками. Отже, перший елемент списку відповідає індексу 0. Другий елемент списку відповідає індексу 1 і т. д. Щоб звернутися до четвертого елементу списку, слід вказати елемент з індексом 3. Отже, зі списку можна отримати конкретне значення, вказавши його індекс.

Оскільки функція `len()` повертає кількість елементів списку, то можна отримати доступ до об'єктів списку за їх індексами, витягувати зрізи, вимірювати довжину списку:

```
>>> abetka = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> len(abetka)
7
>>> abetka[0]
'a'
>>> abetka[4]
'e'
>>> abetka[0:3]
['a', 'b', 'c']
>>> abetka[3:]
['d', 'e', 'f', 'g']
```

Для визначення індексу елемента у списку за його значенням, використовується функція index():

```
>>> abetka = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> abetka.index('c')
2
```

Функції та методи списків надано в таблиці 1.1.

Таблиця 1.1 – Характеристика функцій і методів списків

Метод	Основний зміст
`list.append(x)`	Додає елемент до кінця списку
`list.extend(L)`	Розширює список `list`, додає в кінець всі елементи L
`list.insert(i, x)`	Вставляє на *i*-й елемент значення `x`
`list.remove(x)`	Видаляє перший елемент в списку, який має значення `x`. `ValueError`, якщо такого елементу не існує
`list.pop([i])`	Видаляє *i*-й елемент та повертає його. Якщо індекс не вказано, видаляє останній елемент.
`list.index(x, [start [, end]])`	Повертає місцезнаходження першого елемента зі значенням `x` (при цьому пошук ведеться від `start` до `end`)
`list.count(x)`	Повертає кількість елементів зі значенням `x`
`list.sort([key=функция])`	Сортує список на основі функції
`list.reverse()`	Перевертає список
`list.copy()`	Поверхнева копія списку
`list.clear()`	Очищає список

1.2 Особливості виконання основних операцій з рядками

Рядки в Python є впорядковані послідовності символів, використовуються для зберігання і подання текстової інформації, що дозволяє за допомогою рядків працювати з усім, наданим у текстовій формі.

У Python, рядок – це послідовність символів Unicode. Юнікод був введений для включення кожного символу на всіх мовах і забезпечення однаковості в кодуванні:

```
>>> unicode_a ='\N{LATIN SMALL LETTER A}' Екранування за ім'ям у Юнікоді
>>> unicode_a
'a' ❶
>>> unicode_a_with_acute = '\N{LATIN SMALL LETTER A WITH ACUTE}'
>>> unicode_a_with_acute
'á'
>>> "\u00E1" Екранування за кодом символу c \u
'á'
>>>
```

Оскільки окремого символьного типу в Python немає, то будь-яка послідовність символів, укладена в лапки, в Python вважається рядком; рядки можуть бути укладені як в одиночні, так і в подвійні лапки:

```
"This is a string."
'This is also a string.'
```

Причина наявності двох варіантів в тому, щоб дозволити вставляти в літерали рядків символи лапок і апострофів. Рядки записують у потрійних лапках або апострофі. Головна перевага рядків у потрійних лапках – це можливість використовувати для запису багаторядкові блоки тексту:

```
>>> S = '''Це
довгий
рядок'''
>>> S
'Це\nдовгий\nрядок'
>>> print(S)
Це
довгий
рядок
```

Усередині такого рядка можлива присутність лапок і апострофів, головне, щоб не було трьох лапок поспіль. Символ '\ n' означає новий рядок на одну вниз (кнопка <Enter>) і розділяє рядки текстових файлів. У Windows прийнято використовувати 2 розділових символи поспіль: '\ r \ n', а в MacOSX тільки '\ r'. Сучасні редактори за виключенням «Блокнота» Windows справляються з файлами, записаними з використанням «чужих» роздільників.

Рядки розглядають як послідовності символів у контексті вилучення символів і підрядків, тобто для них використовуються індекси і синтаксис сегментів:

```
>>> 'spam eggs'  # Одинарні лапки
'spam eggs'
>>> 'doesn\'t'   # \' використовуються для екранування* одинарні лапки... екранування -> відображення одинарних або подвійних лапок, що стоять після \
"doesn't"
>>> "doesn't"  # ...або, як варіант, подвійні лапки
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
```

```
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

Зокрема, синтаксис сегментів часто застосовується для відсікання символу нового рядка в кінці рядка. Символ \ (зворотний слеш) використовується для екранування символів, тобто для призначення їм особливого значення. `\ N` – символ нового рядка, `\ t` – символ табуляції, \\ – один звичайний символ зворотного слеша, а `\ "`– звичайний символ подвійної лапки (такий символ рядки не завершує):

```
x = "\tThis string starts with a \"tab\"."
x = "This string contains a single backslash(\\)."
```

Однорядкові коментарі починаються зі знака решітки «#», багаторядкові починаються і закінчуються трьома подвійними лапками «" " "».

Для визначення символів у рядку використовують функцію `len` так само, як для визначення кількості елементів в списку:

```
>>> len ( "Goodbye")
7
```

Однак рядки не є списками символів. Саме принципова відмінність між рядками і символами полягає в тому, що на відміну від списків, рядки не можуть змінюватися. При спробі використовувати вираз виду `string.append ( 'c')` або `string [0] = 'H'` мають помилку. Для відсікання символу нового рядка створюється рядок, який є сегментом попереднього, а не прямою модифікацією попереднього рядка. Це базове обмеження Python, встановлене з міркувань ефективності.

В інтерактивному режимі вихідний рядок заносять в лапки, а спеціальні символи екрануються символами \. Іноді можуть виникнути відмінності від вихідного введення – лапки можуть бути змінені, але обидва рядки еквівалентні.

Рядок у подвійних лапках, якщо він містить одинарні лапки і не містить подвійних лапок, в іншому випадку рядок заключають в одинарні лапки. Функція `print()` надає висновок, що більш читається – опускає обрамляючі лапки та друкує екрановані і спеціальні символи:

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
```

```
"Isn't," they said.
>>> s = 'Перший рядок.\nДругий Рядок.'  # \n означає переведення рядка
>>> s  # Без функції print() \n включений у висновок
'Перший рядок.\nДругий Рялок.'
>>> print(s)  # З функцією print() \n здійснює перехід на новий рядок
Перший Рядок.
Другий Рядок
```

Якщо необхідно, щоб символи, що починаються з \ , інтерпретувалися як спеціальні, то використовують raw-рядки, додавши r перед першими лапками:

```
>>> print('C:\some\name')  # Тут \n є переведення рядка!
C:\some
ame
>>> print(r'C:\some\name')  # Символ r перед лапками
C:\some\name
```

Рядкові літерали розміщують на декількох рядках. Один із способів їх подання є використання потрійних лапок: "" "..." "" . Кінець рядків автоматично включається в рядок, цього уникають завдяки \ в кінці рядка:

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

що генерує такий висновок (варто зауважити, що початкове переведення рядка не включено):

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname
```

Єдина відмінність полягає в тому, що в рядках, взятих в одинарні лапки, не обов'язково екранувати символи ", а в рядках, взятих в подвійні лапки, – символи ':

```
x = "Don't need a backslash"
x = 'Can\'t get by without a backslash'
x = "Backslash your \" character!"
x = 'You can leave the " alone'
```

1.2.1 Основні базові операції над строками

1. *Арифметичні операції* для рядків подібні до роботи з числами. Вони визначені операторами *додавання* + і *множення* *. У результаті додавання вміст двох рядків записується підряд у новий рядок, наприклад:

```
>>> S1 = ’Py ’
>>> S2 = ’thon ’
>>> S3 = S1 + S2
>>> print(S3)
Python
```

Конкатенація – складання кілька рядків підряд:

```
>>>
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
'spameggs'
```

Множення визначене для рядка й цілого позитивного числа, у результаті виходить новий рядок, що повторює вихідну стільки раз, яке було значення числа, наприклад рядок S3 з попереднього прикладу:

```
>>> S3 * 4
' Pythonpythonpythonpython '
>>> 2 * S3
' Pythonpython'
```

2. *Дублювання рядка*:

```
>>>
>>> print('spam' * 3)
spamspamspam
```

3. *Функція* `len()` дозволяє обчислити довжину рядка, результат має цілочисельний тип. Наприклад, `len('Python')` видасть 6.

```
>>>
>>> len('spam')
4
```

4. *Доступ за індексом* дозволяє звернутися до елемента (символу) рядка за його номером: перший елемент рядка `S` має номер 0, останній `len(S)-1`. Дозволяється використовувати від’ємні індекси, у цьому випадку нумерація відбувається з кінця:

```
>>>
>>> S = 'spam'
>>> S[0]
```

```
's'
>>> S[2]
'a'
>>> S[-2]
'a'
```

що інтерпретується як правило: до негативних індексів завжди додається довжина рядка, останній елемент рядка позначають як -1:

```
>>> S = 'Python '
>>> S [0]
'P'
>>> S[ -1]
'n'
```

Звертання до символу з неіснуючим номером породжує помилку: «IndexError: string index out of range».

Рядки в Python відносяться до категорії незмінних послідовностей, тому при використанні індексів необхідно пам'ятати, що значення символу не можна поміняти, а можна створити новий рядок.

```
>>> S = 'Ура'
>>> S[1] = 'x'
Traceback ( most recent call last ):
File "<pyshell #68 >", line 1, in <module >
S[1] = 'x'
Typeerror : 'str ' obje
t does not support item assignment
>>> S = S [0]+ 'x'+S[2]
>>> S
'Yxa'
```

5. *Зрізи* дозволяють скопіювати або використовувати у виразах частину рядка. *Оператор вилучення зрізу* з рядка має вигляд S [n1: n2]. Індекс початку зрізу є n1, а його закінчення – n2, причому символ з номером n2 в зріз вже не входить. Якщо має місце негативний індекс, то будь-який індекс –n аналогічний запису len (s)–n. Якщо відсутній перший індекс, то зріз береться від початку до другого індексу; при відсутності другого індексу зріз береться від першого індексу до кінця рядка:

```
>>> Day = 'morning ,Ъ afternoon ,Ъ evening '
>>> Day [0:7]
'morning '
>>> Day [9: -9]
```

```
’afternoon ’
>>> Day [ -7:]
’evening ’
```

Оператор вилучення зрізу має вигляд `[X: Y]`, де `X` – початок зрізу, а `Y` – закінчення; символ з номером `Y` в зріз не входить. За замовчуванням перший індекс дорівнює 0, а другий – довжині рядка:

```
>>>
>>> s = 'spameggs'
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs'
```

Для витягування потрібного зрізу можна задати крок. Тоді оператор індексування виглядає так: `[n1: n2: n3]`, де `n3` – це крок, через який здійснюється вибір елементів:

```
>>> flag = 'Червоний Блакитний Білий'
>>> flag [:: 8]
'КГБ'
```

У зрізі рядки s можуть бути пропущені і перший, і другий індекси одночасно: замість них підставляються 0 і `len (s)`, відповідно.

```
>>>
>>> s[::-1]
'sggemaps'
>>> s[3:5:-1]
''
>>> s[2::2]
'aeg'
```

6. *Оператор in* дозволяє дізнатися, чи належить підрядок рядку. оператор повертає логічне значення: Якщо елемент в складі рядка зустрічається, то `True`, і `False`, якщо немає:

```
>>> S = 'Python'
>>> SubS = 'th'
>>> SubS in S
True
```

7. *Функції* min і max застосовують до рядків. Так, `min (s)` визначає і виводить (повертає) символ з найменшим кодом номером в кодової таблиці, наприклад:

```
>>> S = 'Python'
>>> min (S)
'P'
```

Функція `max (s)` повертає символ з найбільшим значенням (кодом), наприклад:

```
>>> S = 'Python'
>>> max (S)
'Y'
```

1.2.2 Методи рядків

Значна кількість операцій над рядками визначається у вигляді методів. Основна відмінність методів і функцій синтаксична. Так, більшість методів раніше були функціями стандартного модуля `string`, в якому тепер залишилися тільки різні константи. Записують методи об'єкта як `об'єкт.метод ()`. Наприклад, `S.isdigit ().`

Методи – це змістовно функції, у яких в якості першого аргументу виступає сам об'єкт, для якого вони викликали. Наприклад, виклик методу `S.isdigit ()` надає логічне значення `True`, якщо всі символи складають рядок `S`, і `False`, якщо інакше.

При виклику методів необхідно пам'ятати, що рядки в Python відносяться до категорії незмінних послідовностей, тобто всі функції і методи можуть лише створювати новий рядок. Тому всі строкові методи повертають новий рядок, який потім слід привласнити змінній:

```
>>>
>>> s = 'spam'
>>> s[1] = 'b'
Traceback (most recent call last):
  File "", line 1, in
    s[1] = 'b'
TypeError: 'str' object does not support item assignment
>>> s = s[0] + 'b' + s[2:]
>>> s
'sbam'
```

Є методи, які не потребують ніяких аргументів, бувають з одним аргументом, наприклад метод `S1.endswith (S2)` вимагає 1 аргумент – рядок – і перевіряє, чи закінчується рядок `S1` рядком `S2`. Є методи з двома

аргументами, наприклад `S1.replace (S2, S3)`, який замінює в заданому рядку `S1` підрядок `S2`, що міститься в ньому, новим підрядком `S3` і видає новий рядок, при цьому `S1` залишається незмінним. Більш повну інформацію про методи рядків можна отримати при введенні в інтерактивному режимі команди `help (str)` (табл. 1.2).

Таблиця 1.2 – Функції та методи рядків

Функція або метод	Зміст і результат
`S.find``(str, [start],[end])`	Пошук підрядка в рядку. Повертає номер першого входження або –1
`S.rfind``(str, [start],[end])`	Пошук підрядка в рядку. Повертає номер останнього входження або –1
`S.index``(str, [start],[end])`	Пошук підрядка в рядку. Повертає номер першого входження або викликає `ValueError`
`S.rindex``(str, [strt],[end])`	Пошук підрядка в рядку. Повертає номер останнього входження або викликає `ValueError`
`S.replace` (шаблон, заміна)	Заміна шаблону
`S.split`(символ)	Розбиття рядка за роздільником
`S.isdigit``()`	Чи складається рядок з цифр
`S.isalpha``()`	Чи складається рядок з букв
`S.isalnum``()`	Чи складається рядок з цифр або букв
`S.islower``()`	Чи складається рядок із символів у нижньому регістрі
`S.isupper``()`	Чи складається рядок із символів у верхньому регістрі
`S.isspace``()`	Чи складається рядок з не відображаються символів (пробіл, символ перекладу сторінки `( '\ f')`, "новий рядок" `( '\ n')`, "переклад каретки" `( '\ r')`, "горизонтальна табуляція" `( '\ t' )` і "вертикальна табуляція" `( '\ v')`)
`S.istitle``()`	Чи починаються слова в рядку з великої літери
`S.upper``()`	Перетворення рядка до верхнього регістру
`S.lower``()`	Перетворення рядка до нижнього регістру

Продовження таблиці 1.2

Функція або метод	Зміст і результат
S.startswith(str)	Чи починається рядок S з шаблону str
S.endswith(str)	Закінчується рядок S шаблоном str
S.join(список)	Збірка рядки зі списку з роздільником S
ord(символ)	Символ в його код ASCII
S.capitalize()	Перекладає перший символ рядка в верхній регістр, а всі інші в нижній
S.center(width, [fill])	Повертає відцентрувати рядок, по краям якого стоїть символ fill (пробіл за замовчуванням)
S.count(str, [start],[end])	Повертає кількість непересічних входжень підрядка в діапазоні [початок, кінець] (0 і довжина рядка за замовчуванням)
S.expandtabs([tabsize])	Повертає копію рядка, в якій всі символи табуляції замінюються одним або декількома пропусками, в залежності від поточного стовпця. Якщо tabsize не вказано, розмір табуляції вважається рівним 8 прогалин
S.lstrip([chars])	Видалення символів-пробілів на початку рядка
S.rstrip([chars])	Видалення символів пробілів у кінці рядка
S.strip([chars])	Видалення символів-пробілів на початку і в кінці рядка
S.partition(шаблон)	Повертає кортеж, що містить частину перед першим шаблоном, сам шаблон, і частина після шаблону. Якщо шаблон не знайдений, повертається кортеж, що містить сам рядок, а потім два порожніх рядки
S.rpartition(sep)	Повертає кортеж, що містить частину перед останнім шаблоном, сам шаблон, і частина після шаблону. Якщо шаблон не знайдений, то повертається кортеж, який містить два порожні рядки, а потім сам рядок
S.title()	Першу букву кожного слова переводить в верхній регістр, а всі інші в нижній

Кінець таблиці 1.2

Функція або метод	Зміст і результат
`S.zfill(width)`	Робить довжину рядки не меншою `width`, в разі потреби заповнюючи перші символи нулями
`S.ljust(width, fillchar=" ")`	Робить довжину рядки не меншою `width`, в разі потреби заповнюючи останні символи символом `fillchar`
`S.rjust(width, fillchar=" ")`	Робить довжину рядки не меншою `width`, в разі потреби заповнюючи перші символи символом `fillchar`
`S.format(*args, **kwargs)`	Форматування рядка

1.3 Особливості виконання основних операцій з множинами

Множини (`set`) в Python являють собою невпорядковану колекцію об'єктів, яка використовується в тому випадку, якщо є факт приналежності об'єкта до колекції і його унікальність. Елементи множини повинні бути незмінними і хешованими. Це означає, що цілі числа, числа з плаваючою точкою, рядки і кортежі можуть бути елементами множини, а списки, словники і самі множини – ні.

1.3.1 Операції з множинами

Множини мають ряд операцій, специфічних для множин поряд з операціями, які можна застосувати до колекцій взагалі, як `in`, `len` і перебір для циклів `for`. Множини можна створити викликом `set` для послідовності, наприклад, для списку (`1`). При перетворенні послідовності в множину дублікати виключаються (`2`). Після створення множини функцією `set` можна використати методи `add` і `remove` для зміни елементів множини. Ключове слово `in` використовується для перевірки приналежності об'єкта до множини (`5`). Оператор | (`6`) обчислює об'єднання двох множин, оператор & – їх перетин (`7`), а оператор ^ (`8`) – їх симетричну різницю (тобто елементи, що входять тільки в одне з двох множин).

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5} ❷
>>> x.add(6) ❸
```

```
>>> x
{1, 2, 3, 5, 6}
>>> x.remove(5) ❹
>>> x
{1, 2, 3, 6}
>>> 1 in x ❺
True
>>> 4 in x
False
>>> y = set([1, 7, 8, 9])
>>> x | y ❻
{1, 2, 3, 6, 7, 8, 9}
>>> x & y ❼
{1}
>>> x ^ y ❽
{2, 3, 6, 7, 8, 9}
>>>
```

Робота з множинами визначена офіційною документацією Python.

1.3.2 Фіксовані множини

Оскільки множини є змінними і хешованими, вони не можуть бути елементами інших множин. Для вирішення цієї проблеми в Python існує ще один тип множин `frozenset`, який веде себе як множина незмінності і хешованості. Ці фіксовані множини можуть бути елементами інших множин, наприклад

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> z = frozenset(x)
>>> z
frozenset({1, 2, 3, 5})
>>> z.add(6)
Traceback (most recent call last):
File "<pyshell#79>", line 1, in <module>
z.add(6)
AttributeError: 'frozenset' object has no attribute 'add'
>>> x.add(z)
>>> x
{1, 2, 3, 5, frozenset({1, 2, 3, 5})}
```

Множини є ітерованими колекціями, але вони невпорядковані і не містять елементи, що повторюються. Множини в Python – «контейнер», що містить елементи, які не повторюються у випадковому порядку:

```
>>>
>>> a = set()
>>> a
set()
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range(10)} # генератор множин
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {}  # А так неможна!
>>> type(a)
<class 'dict'>
```

Множини використовують для видалення однакових елементів:

```
>>>
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

З множинами знаходять об'єднання, перетини та інші зміни:

- len (s) – число елементів у множині (розмір множини);
- x in s – чи належить x множині s;
- set.isdisjoint (other) – істина, якщо set і other не мають спільних елементів;
- set == other – усі елементи set належать other, усі елементи other належать set;
- set.issubset (other) або set <= other – усі елементи set належать other;
- set.issuperset (other) або set> = other – аналогічно.
- set.union (other, ...) або set | other | ... – об'єднання декількох множин;
- set.intersection (other, ...) або set & other & ... – перетин;
- set.difference (other, ...) або set - other - ... – множина з усіх елементів set, які не належать жодному з other;
- set.symmetric_difference (other); set ^ other – множини з елементів, що зустрічаються в одній множині, але не зустрічаються в обох;

▪ set.copy () – копія множини.

Операцій, які безпосередньо змінюють множини, є такими:

• set.update (other, ...); set | = other | ... – об'єднання;

• set.intersection_update (other, ...); set & = other & ... – перетин;

• set.difference_update (other, ...); set - = other | ... – віднімання;

• set.symmetric_difference_update (other); set ^ = other – множини з елементів, що зустрічаються в одній множині, але відсутні в обох;

• set.add (elem) – додає елемент у множину;

• set.remove (elem) – видаляє елемент з множини; KeyError, якщо такого елемента не існує;

• set.discard (elem) – видаляє елемент, що знаходиться в множині;

• set.pop () – видаляє перший елемент з множини; однак, множини є невпорядкованими, то не можна сказати, який елемент буде першим;

• set.clear () – очищення множини.

Відмінність `set` від `frozenset` полягає в тому, що set – змінюваний тип даних, а `frozenset` – ні, як то ситуація з списками і кортежами:

```
>>>
>>> a = set('qwerty')
>>> b = frozenset('qwerty')
>>> a == b
True
>>> True
True
>>> type(a - b)
<class 'set'>
>>> type(a | b)
<class 'set'>
>>> a.add(1)
>>> b.add(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Отже, множини – це об'єкт, який зберігає колекцію даних таким же чином, що і математичні множини:

1) усі елементи в множині повинні бути унікальними – ніякі два елементи не можуть мати однакове значення;

2) множини не впорядковані, тобто елементи в множині не зберігаються в якомусь певному порядку;

3) елементи в множині можуть мати різні типи даних.

1.3.3 Створення множини

Для побудови множини необхідно викликати вбудовану функцію set, наприклад порожньої множини:

```
myset = set ()
```

Після виконання цієї інструкції змінна `myset` буде посилатися на порожню множину. Передача одного аргументу повинен бути об'єктом із елементами ітерацій, таких як список, кортеж чи рядкові значення. Окремі елементи об'єкта, переданого в якості аргументу, стають елементами множини. Наприклад в функцію `set` як аргумент передається список:

```
myset = set (( 'а', 'б', 'в'])
```

Після виконання цієї інструкції змінна `myset` посилається на множину, що містить елементи `'а', 'б' і 'в'`. Якщо в якості аргументу в функцію `set` передати рядкове значення, то кожен окремий символ в рядковому значенні стає членом множини:

```
myset = set ( 'абв')
```

Після виконання цієї інструкції змінна `myset` буде посилатися на множину, що містить елементи 'а', 'б' і 'в'.

Множини не можуть містити елементи, що повторюються. Якщо в функцію `set` передати аргумент, що містить повторювані елементи, то в множині з'явиться тільки один з цих повторюваних елементів:

```
myset = set ( 'аааабв')
```

Функцію `len` використовують для визначення кількості елементів у множині. Наприклад при інтерактивному сеансі мають такий результат:

```
1 >>> myset = set([1, 2, 3, 4, 5]) Enter
2 >>> len(myset) Enter
3 5
4 >>>
```

Множини є об'єктами, що здатні до змін за рахунок додавання і видалення елементів. **Для додавання елемента до множини** використовується метод `add()`. У наданому нижче прикладі інструкція в рядку 1 створює порожню множину та привласнює його змінній `myset`. Інструкції в рядках 2–4 додають у множину значення 1, 2 і 3. Рядок 5 показує вміст множини, що виводиться в рядку 6. Інструкція в рядку 7 намагається до-

дати в множину значення 2, яке вже є в множині. Метод не викличе змін, він просто не додасть елемент:

```
1 >>> myset = set() Enter
2 >>> myset.add(1) Enter
3 >>> myset.add(2) Enter
4 >>> myset.add(3) Enter
5 >>> myset Enter
6 {1, 2, 3}
7 >>> myset.add(2) Enter
8 >>> myset
9 {1, 2, 3}
```

У множину можна **додати відразу всю групу елементів** за допомогою методу `update ()`. При виклику методу update () як аргумент передається об'єкт, який містить елементи ітерацій, такі як список, кортеж, рядкове значення або іншу множину. Окремі елементи об'єкта, переданого в якості аргументу, стають елементами множини:

```
1 >>> myset = set([1, 2, 3]) Enter
2 >>> myset.update([4, 5, 6]) Enter
3 >>> myset Enter
4 {1, 2, 3, 4, 5, 6}
5 >>>
```

Інструкція в рядку 1 створює множину, що містить значення 1, 2 і 3. Рядок 2 додає значення 4, 5 і 6. Можливими є більш складні перетворення з множиною, наприклад:

```
1 >>> set1 = set([1, 2, 3]) Enter
2 >>> set2 = set([8, 9, 10]) Enter
3 >>> set1.update(set2) Enter
4 >>> set1
5 {1, 2, 3, 8, 9, 10}
6 >>> set2 Enter
7 {8, 9, 10}
8 >>>
```

Рядок 1 створює множину, що містить значення 1, 2 і 3, і привласнює його змінній set1. Рядок 2 створює множину, що містить значення 8, 9 і 10, і привласнює його змінній `set2`. Метод `set1. update ()` викликадено у рядку 3 викликає, передаючи `set2` як аргумент. У результаті елемент `set2` додається в `set1`. Зверніть увагу, що `set2` залишається без змін. У створювану множину включають числові і рядкові значення:

```
1 >>> myset = set([1, 2, 3]) Enter
2 >>> myset.update('abc') Enter
3 >>> myset Enter
4 {1, 2, 3, 'a', 'c', 'b'}
5 >>>
```

Інструкція в рядку 1 створює множину зі значеннями 1, 2, 3. Рядок 2 викликає метод `myset. Update ()` для передавання рядкового значення 'abc' в якості аргументу. В результаті кожен символ у цьому рядковому значенні додається як елемент в `myset`.

Елемент з множини можна видалити або методом `remove ()`, або методом `discard ()`. Видалений елемент передається в якості аргументу в один із зазначених методів, і цей елемент видаляється з множини. Між цими двома методами є тільки одна різниця, що визначена особливістю поведінки при видаленні елементу, не знайденого в множині. Метод `remove ()` викликає виняток `KeyError`, а метод `discard ()` виключення не викликає:

```
 1 >>> myset = set([1, 2, 3, 4, 5]) Enter
 2 >>> myset Enter
 3 {1, 2, 3, 4, 5}
 4 >>> myset.remove(1) Enter
 5 >>> myset Enter
 6 {2, 3, 4, 5}
 7 >>> myset.discard(5) Enter
 8 >>> myset Enter
 9 {2, 3, 4}
10 >>> myset.discard(99) Enter
11 >>> myset.remove(99) Enter
12 Traceback (most recent call last):
13    File "<pyshell#12>", line 1, in <module>
14      myset.remove(99)
15 KeyError: 99
16 >>>
```

Рядок 1 створює множину з елементами 1, 2, 3, 4 і 5. Рядок 2 показує множину, що виводиться в рядку 3. Рядок 4 викликає метод `remove ()`, щоб видалити з множини значення 1. За станом рядку 6, видно, що значення 1 більше в множині не існує. Рядок 7 викликає метод `discard()`, щоб видалити з множини значення 5. У рядку 9 показано відсутність значення 5 у результативній множині. Рядок 10 викликає метод `discard ()`, щоб видалити з множини значення 99, що відсутнє в ній. У зв'язку з цим

метод discard() не викликає. Рядок 11 викликає метод remove () для видалення з множини значення 99. У такому разі за цим методом використано виключення KeyError, як показано в рядках 12–15.

Усі елементи множини видаляють при виклику методу clear (). Це демонструє наведений інтерактивний сеанс:

```
1 >>> myset = set ((1, 2, 3, 4, 5]) [Enter]
2 >>> myset [Enter]
3 {1, 2, 3, 4, 5)
4 >>> myset. clear () [Enter]
5 >>> myset [Enter]
6 set ()
7 >>>
```

Інструкція в рядку 4 викликає метод clear() для спустошення множини. Зверніть увагу, коли в рядку 6 виводиться пуста множина, інтерпретатор показує set().

Застосування циклу for для послідовного обходу множини. Для послідовного перебору всіх елементів у множині цикл for використовується в такому загальному форматі:

```
for змінна in множини:
інструкція
інструкція
…
```

У даному форматі змінна – це ім'я змінної; множина – ім'я множини. Цей цикл робить одну ітерацію для кожного елемента в множині, під час якої змінній присвоюється елемент. Інтерактивний сеанс надає цей процес у такому вигляді:

```
1 >>> myset = set ([ 'a', 'b', 'c']) [Enter]
2 >>> for val in myset: [Enter]
3         print (val) [Enter] IEnterl
4
5 a
6 b
7 c
8 >>>
```

Рядки 2–3 містять цикл for, який виконує одну ітерацію для кожного елемента множини myset. Під час кожної ітерації циклу елемент множини отримує значення змінної, яке відповідно до рядка 3 друкується. Рядки 5–7 показують результати роботи циклу.

Застосування операторів in і not in для перевірки на приналежність значення множині. Оператор in використовується для визначення існування значення в множині:

```
1 >>> myset = set ([l, 2, 3]) [Enter]
2 >>> if 1 in myset: [Enter]
3 print ( 'Значення 1 знаходиться в множині. ‘) [Enter] [Enter]
4
5 Значення 1 знаходиться в множині.
6 >>>
```

Інструкція if в рядку 2 виявляє знаходження значення 1 в множині myset. Інструкція в рядку 3 показує повідомлення.

Для визначення відсутності елемента в множині використовується оператор not in:

```
1 >>> myset = set ([l, 2, 3]) [Enter]
2 >>> if 99 not in myset: [Enter]
3 print ( 'Значення 99 не перебуває в множині.') [Enter] [Enter]
4
5 Значення 99 не перебуває в множині.
6 >>>
```

Об’єднання множин: операція, результатом якої є множина, що містить елементи обох множин. Для об’єднання двох множин викликається метод union (), загальний формат виклику якого має вигляд:

```
множина1.union (множина2)
```

У цьому форматі множина1 і множина2 – це множини. Метод повертає множину, яка містить елементи множини1 і елементи множини2:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 = set ([3, 4, 5, 6]) [Enter]
3 >>> set3 = setl.union (set2) [Enter]
4 >>> set3 [Enter]
5 (1, 2, 3, 4, 5, 6)
6 >>>
```

Інструкція в рядку 3 викликає метод union () об’єкта setl, передаючи set2 як аргумент. У результаті мають множину, що включає всі елементи setl і елементи set2 без повторів. Отриманій множині присвоюється змінна set3. Для об’єднання двох множин використовують оператор 1, наприклад, оператор 1 з двома множинами:

```
Множина1 | множина2
```

У результаті мають множину, до якої входять елементи множиниl і елементи множини2:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 = set ([3, 4, 5, 6]) [Enter]
3 >>> set3 = setl 1 set2 [Enter]
4 >>> set3 [Enter]
5 (1, 2, 3, 4, 5, 6)
6 >>>
```

Перетин двох множин – це операція над множинами, що надає множину з елементами, які знаходяться в обох множинах. У Python для отримання перетину двох множин викликається метод `intersection ()`, що має такий загальний вигляд формату виклику:

```
множина1.intersection (множина2)
```

Даний метод повертає множину, що містить елементи, які знаходяться одночасно в множиніl і в множині2. Наприклад, інтерактивний сеанс цього методу:

```
1 >>> setl
2 >>> set2
3 >>> set3
set ([1, 2, 3, 4]) [Enter]
set ([3, 4, 5, 6]) [Entei]
setl.intersection (set2) [Enter]
4 >>> set3 [Efite]
5 {3, 4)
6 >>>
```

Інструкція в рядку 3 викликає метод `intersection ()` об'єкта `setl`, передаючи `set2` у якості аргументу. Цей метод повертає множину, до якої входять елементи, що знаходяться одночасно в `setl` і в `set2`. Отримана множина присвоюється змінній `set3`.

Для знаходження перетину двох множин використовують оператор, наприклад оператор `&` з двома множинами:

```
множина1 & множина2
```

Цей вираз повертає множину, що включає елементи, які є одночасно в множиніl і в множині2. Приклад інтерактивного сеансу цієї операції:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 = set ([3, 4, 5, 6]) [Enter]
3 >>> set3 = setl & set2 [Enter]
```

```
4 >>> set3 [Enter]
5 {3, 4)
6 >>>
```

Різниця множиниl і множини2 – це все елементи множиниl, що не входять у множину2. Для отримання різниці двох множин викликається метод `difference()`, загальний формат якого має такий вигляд:

```
множина1.difference (множина2)
```

Даний метод повертає множину, в яку входять усі елементи множиниl, що не входять в множину2. Інтерактивний сеанс таким чином демонструє цей метод:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 = set ([3, 4, 5, 6]) [Enter]
3 >>> set3 = setl.difference (set2) [Enter]
4 >>> set3 [Enter]
5 {1, 2)
6 >>>
```

Для знаходження різниці двох множин використовують також оператор, наприклад з двома множинами:

```
множина1 - множина2
```

Цей вираз повертає множину, в яку входять усі елементи множиниl, що не входять в множину2, ця операція визначається таким чином:

```
1 >>> setl set ([l, 2, 3, 4]) [Enter]
2 »> set2 = set ([3, 4, 5, 6]) [Enter]
490 Глава 9. Словники і множини
3 >>> set3 = setl - set2 [Enter]
4 >>> set3 [Enter]
5 {1, 2)
6 >>>
```

Симетрична різниця двох множин – це множина, яка містить елементи, які не належать одночасно обом вихідним множинам. Іншими словами, це елементи, які входять в одне з множин, але не входять в обидві множини одночасно.

У Python для отримання симетричної різниці двох множин викликається метод `synunetric_difference ()`, що має такий формат виклику:

```
множина1.synunetric_difference (множина2)
```

Даний метод повертає множину, що містить елементи або множиниl, або множини2, але не входять в обидві множини одночасно:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 = set ([3, 4, 5, 6]) [Enter]
3 >>> set3 = setl.synunetric_difference (set2) [Enter]
4 >>> set3 [Enter]
5 {1, 2, 5, 6)
6 >>>
```

Для знаходження симетричної різниці двох множин використовувати оператор ^, загальний формат вираження якого має вигляд:

```
множина1 ^ множина2
```

Таким чином отримують множину, до якої входять елементи або множини1, або множини2, але не входять в обидві множини одночасно:

```
1 >>> setl set ([1, 2, 3, 4]) [Enter]
2 >>> set2 set ([3, 4, 5, 6]) [Enter]
3 >>> set3 setl ^ set2 [Enter]
4 >>> set3 [Enter]
5 {1, 2, 5, 6)
6 >>>
```

Підмножини і над множини: якщо одна з двох розглянутих множин містить всі елементи іншої множини, як наприклад `setl` містить всі елементи `set2:`

```
setl set ([l, 2, 3, 4])
set2 = set ([2, 3])
```

то `set2` є підмножиною `setl`. Це означає, що `setl` є надмножиною `set2`.

У Python для визначення, чи є одна з множин підмножиною іншої, використовується метод `issubset ()`:

```
множина2.issubset (множина1)
```

Даний метод повертає `True`, якщо множина2 є підмножиною множини1. В іншому випадку він повертає `False`.

Для того щоб визначити, чи є одне з множин надмножиною іншої, викликається метод `issuperset ()`:

```
множина1.issuperset (множина2)
```

Тут множина1 і множина2 – це множини. Даний метод повертає `True`, якщо множина1 є надмножиною множини2, а інакше – `False`:

```
1 >>> setl = set ([l, 2, 3, 4]) [Enter]
2 >>> set2 = set ([2, 3]) [Enter]
3 >>> set2.issubset (setl)
```

```
4 True
5 >>> setl.issuperset (set2) [Enter]
6 True
7 >>>
```

Для визначення, чи є одне з множин підмножиною іншого, застосовується оператор <=.

Для визначення, чи є одне з множин надмножиною іншого, використовується оператор> =.

Загальний формат вираження з використанням оператора <= має такий вигляд з двома множинами:

```
множина2 <= множинаl
```

Цей вираз повертає `True`, якщо множина2 є підмножиною множиниl. В іншому випадку отримують `False`.

Загальний формат вираження з використанням оператора> = з двома множинами має такий вигляд:

```
множинаl> = множина2
```

Цей вираз повертає `True`, якщо множинаl є надмножиною множини2. В іншому випадку воно повертає `False`.

Наданий інтерактивний сеанс демонструє ці операції:

```
1 >>> setl = set ([l, 2, 3, 4]) [Enter]
2 >>> set2 = set ([2, 3]) [Enter]
3 >>> set2 <= setl [Enter]
4 True
5 >>> setl> = set2 [Enter]
6 True
7 >>> setl <= set2 [Enter]
8 False
```

Мова Python 3 підтримує генератори множин, синтаксис яких схожий на синтаксис генераторів списків, але вираз включають у фігурні дужки, а не у квадратні. Оскільки результатом є множина, то всі повторювані елементи будуть видалені:

```
≫ > {x for x in [1, 2, 1, 2, 1, 2, 3] }
(1, 2, 3}
```

Генератори множини можуть мати складну структуру.

Наприклад, складатися з кількох вкладених циклів `for` і (або) містити оператор розгалуження `if` після циклу.

1.4 Словники: поняття, особливості роботи з словниками

1.4.1 Словник у Python

Словник Python є невпорядкованою колекцією елементів. У той час як інші складові типи даних мають лише значення як елемент, словник має за елемент пару *ключ: значення*.

Словники в Python – невпорядковані колекції довільних об'єктів із доступом за ключем. Їх іноді ще називають *асоціативними масивами* чи *хеш-таблицями*.

Отже, словники – це набори об'єктів, доступ до яких здійснюється не за індексом, а за ключем. Елементи словника можуть містити об'єкти довільного типу даних і мати необмежену міру вкладеності. Елементи у словниках розташовуються у довільному порядку. Щоб отримати елемент, необхідно вказати ключ, який використовувався для збереження значення. Ключем є незмінний об'єкт, як число, рядок чи кортеж.

Словники відносяться до відображень, а не до послідовностей. З цієї причини до словників не застосовуються функції, призначені для роботи з послідовностями, операції вилучення зрізу, конкатенації, повторення та ін. Так само як і списки, словники відносяться до змінних типів даних, тобто можна не лише отримати значення за ключем, але змінити його.

Словник являє собою структуру даних, призначених для об'єднання взаємозалежної інформації. Операції зі словниками дозволяють моделювати різноманітні реальні об'єкти. *Python словник* – це об'єкт, який зберігає колекцію даних. Кожен елемент, що зберігається у словнику, має дві частини: ключ і значення. *Елементи словника* зазвичай визначають парою *ключ/значення*. Коли потрібно отримати зі словника певний об'єкт, використовується ключ, який пов'язаний із його значенням. Це подібно до пошуку слова в «Великому тлумачному словнику», де слова є ключами, а визначення – значеннями. Пари ключ/значення часто називаються відображеннями, тому що кожному ключу поставлено у відповідність значення, тобто кожен ключ як би відображається на відповідне значення.

Для звернення до значень у словниках використовують цілі числа, звані індексами. Вони визначають позицію списку, де знаходиться задане значення. Словники можуть використовувати цілі, рядки або інші об'єкти Python у якості ключа. Таким чином, списки та словники надають індексований доступ до довільних значень. Однак множини елементів, які можуть використовуватися як індекси словників, набагато більше множини значень, які можуть використовуватися як індекси списків. Механізм, який

використовується словниками для індексування, дуже відрізняється від аналогічного механізму списків.

Значення, що зберігаються у списку, неявно впорядковуються за своєю позицією, тому що індекси, що використовуються для звернення до них, є цілими числами. Для значень, що зберігаються у словнику, неявний порядок щодо один одного не визначений, тому що ключі не обмежуються числами. Якщо робота пов'язана зі словником і є потреба упорядкування елементів (порядок їх додавання), то використовують упорядкований словник субкласу, який можна імпортувати з модуля `collections`. Визначити порядок для елементів словника можна за допомогою іншої структури даних, де упорядкована інформація зберігається в явному вигляді, часто це список. У базових словників неявний (вбудований) порядок не визначено.

Словник дозволяє відобразити одну множину довільних об'єктів на іншу логічно пов'язану, але так само довільну множину об'єктів. Хорошим аналогом словників Python є реальні словники або алфавітні довідники, наприклад перекладач назв квітів з англійської на французьку:

```
>>> english_to_french = {} Створює пустий словник
>>> english_to_french['red'] = 'rouge' Зберігає три слова
>>> english_to_french['blue'] = 'bleu'
>>> english_to_french['green'] = 'vert'
>>> print("red is", english_to_french['red']) C Отримує
значення для ключа 'red'
red is rouge
```

Словник визначають також як серію пар «ключ-значення», розділених комами:

```
>>> english_to_french = {'red': 'rouge', 'blue': 'bleu',
'green': 'vert'}
```

Незважаючи на всі відмінності, операції зі словниками та списками часто виглядають однаково. Спочатку програма створює порожній словник майже так само, як порожній список:

```
>>> x = []
>>> y = {}
```

Перший рядок створює новий порожній список і надає змінній `x`. Другий рядок створює новий порожній словник і надає його змінній `y`.

Після того як словник буде створено, в ньому можна зберігати значення так, якби це був список. За допомогою фігурних дужок визначається словник. Такий тип даних у Python називається `dict`.

При створенні словника у інтерпретатор Python після натискання <Enter> можна спостерігати, що послідовність виведення пар 'ключ : значення' не співпадає з тим, як було запроваджено. Це тому, що у словнику абсолютно не важливий порядок пар, та інтерпретатор виводить їх у випадковому порядку. Тоді доступ до значень у словнику здійснюється за ключами, які укладаються у квадратні дужки (аналогічно роботі з індексами рядків і списків):

```
>>> animal ={ ’cat ’:’кішка’,’dog ’:’пес’,’bird ’:’птушка’,’mouse ’:’миша’}
>>> animal [’cat ’]
'кішка'
```

Пара «ключ-значення» являє собою дані, пов’язані один з одним. При зверненні до ключа Python поверне значення, пов’язане з цим ключем. Ключ відділяється від значення двокрапкою, а окремі пари поділяються комами. У словнику може зберігатися будь-яка кількість пар «ключ-значення». Найпростіший словник містить одну пару «ключ-значення», як наприклад у версії словника `alien_0`:

```
alien_0 = {'color': 'green'}
```

У цьому словнику зберігається як один фрагмент інформації про об’єкт `alien_0`, а саме – його колір. Рядок `'color'` є ключем у словнику; з цим ключем пов’язане значення `'green'`.

Кількість пар "ключ-значення" у словнику не обмежена. Наприклад, вихідний словник `alien_0` з двома парами «ключ-значення»:

```
alien_0 = {'color': 'green', 'points': 5}
```

Тепер програма може отримати значення, пов’язане з будь-яким ключем `alien_0: color` або `points`. Якщо передбачити зникнення об’єкта і визначити отримані бали як перевагу його відсутності, то можна скористатися кодом такого виду:

```
alien_0 = {'color': 'green', 'points': 5}
❶ new_points = alien_0['points']
❷ print("You just earned " + str(new_points) + " points!")
```

Після того як словник буде визначено, код 1 отримує значення, пов’язане з ключем `'points'`, зі словника. Потім це значення зберігається у змінній `new_points`. Рядок 2 перетворює ціле значення на рядок і виводить повідомлення з кількістю зароблених очок:

```
You just earned 5 points!
```

Словники, як і списки, є типом даних, що можна змінювати: додавати та видаляти елементи у вигляді пари `'ключ : значення'`. Починати треба зі створення порожнього словника, наприклад, `dic = {}` і лише потім заповнити його елементами.

Додавання та зміна має однаковий синтаксис:

`словник[ключ] = значення`.

Ключ може існувати, при зміні його значення відбувається поява нового, що визначається додаванням елемента словника.

Ключем можуть бути рядкові значення, цілі числа, значення з плаваючою точкою або кортежі. **Ключами не можуть бути списки або об'єкти інших типів, що мутують.**

Отже, як ключ словника використовують будь-який об'єкт Python, який є незмінним та хешованим.

У Python будь-який об'єкт, який може бути змінений «на місці», називається *змінним*. Списки змінюються за рахунок операцій додавання, змінювання та видалення їх елементів. Словники змінюються з тих самих причин. *Числа незмінні*. Якщо змінна `x` посилається на число 3, то після надання `x` значення 4 змінна буде посилатися на нове число 4, але число 3 при цьому не зміниться. Рядки теж незмінні. Запис `list[n]` повертає n елемент списку, `string[n]` повертає n символ рядка, а `list[n] = value` змінює n елемент рядка; проте `запис string[n] = character` в Python неприпустимий і породжує помилку, тому що рядки в Python незмінні.

Вимога незмінності та хешованості ключів означає, що списки не можуть бути ключами словників, але в багатьох випадках було б зручно працювати з ключами, що нагадують списки. Наприклад, доцільно зберігати інформацію про людину з ключем, що складається з імені та прізвища, якщо використати двоелементний список як ключ.

У Python ця проблема вирішується за рахунок кортежів, що змістовно і є незмінними списками. Кортежі створюються і використовуються так само, як списки, але після створення вони не можуть змінюватися.

Друге обмеження: ключі словників повинні бути хешовані, що виводить ситуацію за межу простої незмінності. Щоб бути хешованим, значення повинно мати хеш-код, що надається методом `__hash__`, який ніколи не змінюється протягом терміну життя значення. Це означає, що кортежі, що містять змінювані значення, не є хешованими, хоча самі кортежі формально залишаються незмінними. Тільки кортежі, які не містять змінюваних об'єктів, є хешованими і стають ключем у словниках.

У таблиці 1.3 показано, які з вбудованих типів Python є незмінними, хешованими та придатними для використання як ключі словника.

Таблиця 1.3 – Значення Python, придатні як ключі словника

Тип Python	Незмінний?	Хешований?	Ключ словника?
`int`	Так	Так	Так
`float`	Так	Так	Так
`boolean`	Так	Так	Так
`complex`	Так	Так	Так
`str`	Так	Так	Так
`bytes`	Так	Так	Так
`bytearray`	Немає	Немає	Немає
`list`	Немає	Немає	Немає
`tuple`	Так	Іноді	Іноді
`set`	Немає	Немає	Немає
`frozenset`	Так	Так	Так
`dictionary`	Немає	Немає	Немає

У таких областях, як наприклад, метеорологічне прогнозування, результати спостережень надані дуже великими матрицями. Кількість рядків і стовпців обчислюється тисячами, отже, матриця може містити мільйони елементів. Такі матриці зазвичай містять дуже багато нульових елементів. Для економії пам'яті такі матриці зберігаються у формі, де фактично знаходяться тільки ненульові елементи. У цьому випадку мова йде про *розріджені матриці*.

Розріджені матриці легко реалізуються як словники з індексами-кортежами і в програмі відображаються таким чином:

```
matrix = {(0, 0): 3, (0, 2): -2, (0, 3): 11,
          (1, 1): 9, (2, 1): 7, (3, 3): -5}
```

При зверненні до окремого елементу матриці із заданими номерами стовпця та рядка можна скористатися таким фрагментом коду:

```
if (rownum, colnum) in matrix:
   element = matrix[(rownum, colnum)]
else:
    element = 0
```

Більш ефективним способом вирішення цього завдання є застосування методу `get`, завдяки якому можна повернути `0`, якщо не знайдено ключ у словнику, інакше повертається значення, пов'язане з ключем, що запобігає окремому пошуку за словником.

Словники можуть використовуватися *як кеші* – структури даних, в яких зберігаються обчислені результати, щоб їх не доводилося перераховувати заново. Наприклад, потрібна функція з ім'ям `sole`, яка отримує в аргументах три цілих числа та повертає результат, має такий вигляд:

```
def sole(m, n, t):
    # . . . Дуже довгі обчислення. . .
    return(result)
```

Для уникнення повторного обчислення `sole` для певного набору аргументів, що дозволяє заощадити багато часу, можна використати словник з кортежами як ключі:

```
sole_cache = {}
def sole(m, n, t):
    if (m, n, t) in sole_cache:
        return sole_cache[(m, n, t)]
else:
    # . . . Дуже довгі обчислення . . .
    sole_cache[(m, n, t)] = result
    return result
```

Змінена функція `sole` використовує глобальну змінну для збереження попередніх результатів. Глобальна змінна є словником, ключами якого є кортежі, відповідні комбінаціям аргументів, що передавались `sole` у минулому. Щоразу, коли функція `sole` отримуватиме комбінацію аргументів, на яку результат вже було обчислено, вона поверне збережений результат замість того, щоб перераховувати його заново.

1.4.2 Створення словника

Словник створюється шляхом укладання його елементів у фігурні дужки ({ J }). Елемент складається з ключа, потім двокрапки, після якої йде значення. Елементи словника відокремлюються комами.

Створення словника відбувається кількома способами. *По-перше*, за допомогою літералу:

```
>>> d = {}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

По-друге, за допомогою функції `dict`:

```
>>> d = dict(short='dict', long='dictionary')
```

```
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

По-третє, за допомогою методу `fromkeys`

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'a': 100, 'b': 100}
```

По-четверте, за допомогою генераторів словників, які дуже подібні до генераторів списків:

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Наприклад, інструкція для визначення словника такого виду:

```
phonebook = {'Kpic': '555-1111', 'Кеті': '555-2222',
'Джоанна' :'555-3333'}
```

Ця інструкція створює словник і надає його змінній `phonebook` (телефонна книга). Словник містить такі запропоновані три елементи.

• Перший елемент – `'Kpic': '555-1111'`. У цьому елементі ключем є `'Kpic'`, а значенням – `'555-1111'`.

• Другий елемент – `'Кеті': '555-2222'`. У цьому елементі ключем є `'Кеті'`, а значенням – `'555-2222'`.

• Третій елемент - `'Джоанна': '555-3333'`. У цьому елементі ключем є `'Джоанна'`, а значенням – `'555-3333'`.

У цьому прикладі ключами та значеннями є рядкові об'єкти. Значення у словнику можуть бути об'єктами будь-якого типу, але ключі повинні бути об'єктами, що не змінюються.

Вкладення списку в словник може застосовуватися щоразу, коли з одним ключем пов'язано більше одного значення. Наприклад, при опитуванні студентів щодо вибору мов програмування відповіді зберігаються у списку і надається можливість вибрати одразу кілька улюблених мов. При переборі словника значення, пов'язане з кожною людиною, буде являти собою список мов замість однієї мови. У циклі `for` словника створюється цикл для перебору списку мов, пов'язаних із кожним учасником:

favorite_languages.py

```
❶ favorite_languages = {
      'jen': ['python', 'ruby'],
      'sarah': ['c'],
      'edward': ['ruby', 'go'],
      'phil': ['python', 'haskell'],
      }

❷ for name, languages in favorite_languages.items():
      print("\n" + name.title() + "'s favorite languages are:")
❸     for language in languages:
          print("\t" + language.title())
```

У точці 1 визначене значення пов'язане з кожним ім'ям, що становить список. У деяких учасників опитування лише одна улюблена мова програмування, в інших таких мов кілька. При переборі словника у точці 2 змінна з іменем `languages` використовується для зберігання кожного значення зі словника, тому кожне значення буде являти собою список. В основному циклі за елементами словника цикл 3 перебирає елементи списку улюблених мов кожного учасника. Отже, кожен учасник опитування може вказати скільки завгодно улюблених мов програмування:

```
Jen's favorite languages are:
Python
Ruby
Sarah's favorite languages are:
C
Phil's favorite languages are:
Python
Haskell
Edward's favorite languages are:
Ruby
Go
```

Словник може містити інший словник, але в таких випадках код швидко ускладнюється. Наприклад, якщо на сайті є кілька користувачів з унікальними іменами, використовують імена користувачів як ключі в словнику. Інформація про користувача при цьому зберігається у словнику, який використовується як значення, пов'язане з ім'ям. Наприклад, про користувача зберігається інформація: ім'я, прізвище та місце проживання.

Щоб отримати доступ до цієї інформації, треба перебрати імена користувачів та словник з інформацією, пов'язаною з кожним ім'ям:

many_users.py

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
        },
    }

❶ for username, user_info in users.items():
❷     print("\nUsername: " + username)
❸     full_name = user_info['first'] + " " + user_info['last']
      location = user_info['location']
❹     print("\tFull name: " + full_name.title())
      print("\tLocation: " + location.title())
```

У програмі визначається словник з ім'ям `users`, що містить два ключі: користувачів `'aeinstein'` та `'mcurie'`. Значення, пов'язане з кожним ключем, являє собою словник з ім'ям, прізвищем і місцем проживання користувача. У процесі перебору словника `users` у точці 1 Python зберігає кожен ключ в змінній `username`, а словник, пов'язаний з кожним ім'ям користувача, зберігається у змінній `user_info`. Усередині основного циклу у словнику виводиться Ім'я користувача 2.

У точці 3 починається робота із внутрішнім словником. Змінна `user_info`, що є у словнику з інформацією про користувача, містить три ключі: `'first'`, `'last'` та `'location'`. Кожен ключ використовується для побудови акуратно відформатованих даних з повним ім'ям та місцем проживання користувача, з таким висновком зведення відомої інформації про користувача 4:

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton
Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

1.4.3 Методи словників, операції та функції

Робота зі словниками пов'язана з такими операціями, як вбудовані функції, ключові слова, наприклад, цикли `for` і while, і додатково зі спеціальними методами словників.

До методів словників відносять таке:

- `dict.clear()` – очищає словник;
- `dict.copy()` – повертає копію словника;
- `classmethod dict.fromkeys(seq[, value])` – створює словник з ключами `seq` і значенням `value` (за замовчуванням `None`);
- `dict.get(key[, default])` – повертає значення ключа; при його відсутності не відзначає виняток, а повертає `default` (за замовчуванням `None`);
- `dict.items()` – повертає пари (ключ, значення);
- `dict.keys()` – повертає ключі у словнику;
- `dict.pop(key[, default])` – видаляє ключ та повертає значення; при його відсутності повертає `default` (за замовчуванням – виняток);
- `dict.popitem()` – видаляє та повертає пару (ключ, значення), якщо словник порожній, то викидає `KeyError`;
- `dict.setdefault(key[, default])` – повертає значення ключа, але якщо його немає, то не кидає виняток, а створює ключ зі значенням `default` (за замовчуванням `None`);
- `dict.update([other])` – оновлює словник, додаючи пари (ключ, значення) з other, існуючі ключі перезаписуються та повертає `None` (не новий словник!);
- `dict.values()` – повертає значення у словнику.

Метод `keys()` для словника повертає послідовність всіх використаних ключів у довільному порядку і для отримання ключів у словнику. Метод використовується для перебору вмісту словника у циклі `for`:

```
>>> list(english_to_french.keys())
['green', 'blue', 'red']
```

При необхідності відсортування ключів збережіть їх у списку змінної і відсортуйте отриманий список. Починаючи з Python 3.6, словники зберігають порядок створення ключів і повертають їх у цьому порядку.

Методом `values` отримують усі значення, що є у словнику:

```
>>> list(english_to_french.values())
 ['vert', 'bleu', 'rouge']
```

Метод `items` повертає всі ключі та пов'язані з ними значення у вигляді послідовності кортежів:

```
>>> list(english_to_french.items())
[('green', 'vert'), ('blue', 'bleu'), ('red', 'rouge')]
```

Як і метод `keys`, цей метод часто використовується в циклах для перебору вмісту словника.

Команда `del` використовується для видалення елементів (пари "ключ-значення") зі словника:

```
>>> list(english_to_french.items())
[('green', 'vert'), ('blue', 'bleu'), ('red', 'rouge')]
>>> del english_to_french['green']
>>> list(english_to_french.items())
 [('blue', 'bleu'), ('red', 'rouge')]
```

Методи `keys`, `values` та `items` повертають не списки, а уявлення (views); вони поводяться як послідовності, які динамічно оновлюються за змінами словника. У цих прикладах доводиться використовувати функцію `List` для того, щоб інтерпретувати їх як списки. В іншому випадку вони поводяться як послідовності, що дозволяє програмам перебирати їх у циклах `for`, використовувати `in` для перевірки належності тощо.

Уявлення, що повертається методом `keys` (у деяких випадках методом `items`), поводиться подібно множині з виконанням операцій об'єднання, перетину та різниці.

Присутність ключа у словнику можна перевірити оператором `in`, який повертає `True`, якщо у словнику є значення, асоційоване із заданим ключем, і `False` у іншому випадку:

```
>>> 'red' in english_to_french
True
>>> 'orange' in english_to_french
False
```

Функція `get` повертає значення, пов'язане з ключем, якщо цей ключ є у словнику, або повертає другий аргумент, якщо ключ відсутній:

```
>>> print(english_to_french.get('blue', 'No translation'))
bleu
>>> print(english_to_french.get('chartreuse', 'No
translation'))
No translation
```

Метод `setdefault` дозволяє безпечно отримати значення, пов'язане з ключем, і визначити у словнику його значення:

```
>>> print(english_to_french.setdefault('chartreuse', 'No
translation'))
No translation
```

Відмінність між `get` і `setdefault` полягає в тому, що після виклику setdefault у словнику з'являється ключ `'chartreuse'` зі значенням `'No translation'`.

Для отримання копії словника використовується метод `copy`:

```
>>> x = {0: 'zero', 1: 'one'}
>>> y = x.copy()
>>> y
{0: 'zero', 1: 'one'}
```

Метод `update` оновлює перший словник усіма парами «ключ-значення» з другого словника. Для ключів, загальних для обох словників, значення другого словника замінюють значенням першого словника:

```
>>> z = {1: 'One', 2: 'Two'}
>>> x = {0: 'zero', 1: 'one'}
>>> x.update(z)
>>> x
{0: 'zero', 1: 'One', 2: 'Two'}
```

Методи словників надають у розпорядження повний набір інструментів для роботи зі словниками (табл. 1.4)

Таблиця 1.4 – Найбільш важливі функції словників

Операція	Зміст операції	Приклад
`{}`	Створює порожній словник	`x = {}`
`len`	Повертає кількість елементів у словнику	`len(x)`
`keys`	Повертає уяву, що містить усі ключі в словнику	`x.keys()`
`values`	Повертає виставу, що містить усі значення в словнику	`x.values()`
`items`	Повертає виставу, що містить усі елементи в словнику	`x.items()`
`del`	Видаляє елемент зі словника	`del(x[key])`
`in`	Перевіряє присутність ключа в словнику	`'y' in x`
`get`	Повертає значення ключа або обране значення за замовчуванням	`x.get('y', None)`
`setdefault`	Повертає значення, якщо ключ присутній у словнику; а якщо ні, то асоціює із ключем задане значення за замовчуванням і повертає його	`x.setdefault ('y', None)`
`copy`	Створює поверхневу копію словника	`y = x.copy()`
`update`	Проводить злиття елементів двох словників	`x.update(z)`

Висновок

Списки й кортежі – структури послідовності, що втілюють ідею, елементів (наприклад, рядка).

Списки схожі на масиви інших мов програмування, але вони підтримують автоматичну зміну розмірів, синтаксис сегментів і різні допоміжні функції.

Кортежі схожі на списки, але вони не можуть змінюватися, тому вони витрачають менше пам'яті й можуть використовуватися в якості ключів словника.

Рядки Python підтримують потужні засоби обробки тексту, включаючи пошук і заміну, відсікання символів і зміна регістру.

Рядки є незмінними, тобто вони не можуть змінюватися «на місці». Операції, які на перший погляд змінюють рядки, у дійсності повертають копію зі змінами.

Модуль re містить ще більш потужні засоби роботи з рядками (регулярні вираження).

Множини являють собою ітеровані колекції, але вони не впорядковані й не можуть містити повторювані елементи.

Словники відрізняються особливостями роботи з інформацією, що зберігається в них, мають свої правила звертання до окремих елементів словника та заміни їх, перебору всієї інформації у словнику: пари «ключ-значення», ключі та значення словника.

Розглянуті можливості вкладення словників у список, вкладення списків у словники та вкладення інших словників.

Отже, словники – потужні структури даних, що використовуються для різних цілей навіть у внутрішній реалізації Python.

Ключі словників повинні бути незмінними, будь-який незмінний об'єкт є ключем словника. Використання ключів означає більш прямий доступ до колекцій даних з меншим обсягом коду, ніж інші рішення.

2 ЗАВДАННЯ ТА ВПРАВИ ДО ЛАБОРАТОРНОЇ РОБОТИ

Завдання 1:

Що поверне функція `len ()` для кожного з наступних списків:

[0]; []; [[1, 3, [4, 5], 6], 7]?

Завдання 2:

Використовуючи можливості функції `len()` і сегментах списків, надати будь-якого розміру список і отримати другу половину списку неві-

домого розміру. Проекспериментувати в сеансі Python і переконатися в тому, що ваше рішення працює.

Завдання 3:

Задати список з 10 елементів. Реалізувати на ньому його модифікацію з використанням методів приєднання до списку `append`, видалення з списку методу `remove`, метод `reverse` та метод `insert`.

Завдання 4:

Задати список з 15 елементів. Як перемістити три останніх елемента з кінця в початок списку без порушення їх вихідного порядку? Змінити розміри списку за допомогою команди `del list[n]` і еквівалентної їй `list[n:n+1] = [].`

Завдання 5:

• Скласти список своїх улюблених фруктів. Написати серію незалежних команд `if` для перевірки того, чи присутні фрукти в списку.

• Створити список трьох своїх улюблених фруктів і назвати його `favorite_fruits`.

• Написати п'ять команд `if.` Кожна команда повинна перевіряти, чи входить окремий тип фрукта в список. Якщо фрукт входить у список, то блок `if` повинен надати результат виду «`You really like` *`bananas`*`!».`

Завдання 6:

Дано списки:

```
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89];
b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13].
```

Необхідно повернути список, який складається зі спільних елементів для цих двох списків.

Завдання 7:

Створити словник, та вивести значення та ключі у вигляді впорядкованих списків.

Завдання 8:

Написати перевірку на те, чи є рядок паліндромом. Рядок вводиться користувачем. *Паліндром* – це слово або фраза, які однаково читаються зліва направо і справа наліво.

Завдання 9

Скласти список і кортеж з числами, що прийняті від користувача як послідовність чисел, розділених комою.

Завдання 10

Виведіть перший і останній елемент списку.

Завдання 11

Порахувати *n + nn + nnn* при заданому цілому числі `n`.

Завдання 12

Написати програму, яка приймає два списки та виводить усі елементи першого, яких немає в другому списку.

Завдання 13

Порахувати, скільки разів символ зустрічається в рядку. Символ і рядок вводяться користувачем.

Основні вимоги до виконання і здачі завдань із лабораторної роботи: використовувати можна тільки операції з типами даних та `if..else` конструкцію. Обов'язкова присутність коментарів. При відсутності можливості написати коментар, тому що програми були занадто прості, додайте своє ім'я й поточну дату в початок коду. Додайте одну пропозицію з описом того, що робить програма.

ВИМОГИ ДО ФОРМУВАННЯ ЗВІТУ

Роздрукований звіт з лабораторної роботи, що підтверджує виконання студентом завдань, повинен бути оформлений відповідно до вимог ДСТУ 3008:2015 «Звіти у сфері науки і техніки. Структура та правила оформлення».

Звіт з лабораторної роботи повинен мати обов'язково такі складові інформації про її виконання:

1) мета та завдання лабораторної роботи;
2) скриншоти або лістинг програмного коду;
3) скриншот результату роботи програми;
4) висновки з виконаної роботи.

Основні вимоги до оформлення паперового варіанту звіту:

– параметри сторінки: лівий відступ – 2 см; правий – 1 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14пт;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка – 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скриншоти, розміщені у звіті, є рисунками та повинні мати підписи й відповідну нумерацію.

Звіт з лабораторної роботи роздруковується на аркуші формату А4 з титульним аркушем. Роздрукований звіт здається студентом викладачеві у файлі після опитування та остаточного оцінювання роботи.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

Лабораторна робота №1

1. *Для підтвердження того, що ви навчилися взаємодіяти з інтерпретатором Python*, виконати на комп'ютері наведені завдання.

• Запустити інтерпретатор Python в інтерактивному режимі.

• Навпроти підказки `>>>` набрати наступну інструкцію, потім натисніть клавіші<Enter>:

```
print ('Перевірка інтерпретатора Python. ') [Enter]
```

• Після натискання клавіші <Enter> інтерпретатор виконати інструкцію. Якщо ви ввели все правильно, то ваш сеанс має виглядати так:

```
>>> print ('Перевірка інтерпретатора Python. ') [Enter]
Перевірка інтерпретатора Python.
>>>
```

• Якщо є повідомлення про помилку, ввести інструкцію знову і переконатися, що її набрано точно, як показано.

• Вийдіть із інтерпретатора Python: у Windows натисніть клавіші <Ctrl>+<Z>, а потім <Enter>. В інших операційних системах натисніть клавіші <Ctrl>+<D>.

2. Застосуйте знання про двійкову систему числення, щоб перетворити наступні десяткові числа у двійкові:

11
65
100
255

3. Перетворити двійкові числа до десяткових:

1101
1000
101011

4. Яка операція з цілими числами ніколи не дає як остаточний результат ціле число?

5. Чи відрізняються чимось варіанти оголошення цілого числа: `1230` або `int (1230)`?

6. Перерахувати всі типи даних, які потенційно можна перетворити на ціле число безпосередньо через функцію `int()`.

7. Що таке змінна?

8. Які з наведених імен змінних неприпустимі в Python і чому?

```
99bottles
july2009
theSalesFigureForFiscalYear
r&d
grade_report
```

9. Чи є імена змінних Sales та sales однаковими? Чому?

10. Чи допустима наведена нижче інструкція присвоєння? Якщо вона неприпустима, то чому?

```
72 = amount
```

11. Що покаже наведений нижче фрагмент коду?

```
val = 99
print ('Значення дорівнює', 'val')
```

12. Погляньте на наведені нижче інструкції присвоєння:

```
value1 = 99
value2 = 45.9
value3 = 7.0
value4 = 7
value5 = 'abc'
```

Який тип даних Python матимуть ці значення, коли на них посилатимуться змінні після виконання вказаних інструкцій?

Лабораторна робота №2

1. Що таке складовий булевий вираз?

2. Заповніть наведену таблицю комбінації істинності та помилковості значень, з'єднаних логічним оператором, обвівши "І" або "Н", щоб показати, чи є результатом такої комбінації істина або неправда.

Логічне вираження	Результат (обвести І чи Н	
Істина and неправда	І	Н
Істина and істина	І	Н
Неправда and істина	І	Н
Неправда and неправда	І	Н
Істина or неправда	І	Н
Істина or істина	І	Н
Неправда or істина	І	Н
Неправда or неправда	І	Н
not істина	І	Н
not неправда	І	Н

3. Символи >,< і== є операторами:

а) реляційними;

б) логічними;

в) умовними;

г) трикомпонентними.

4. Яка структура перевіряє умову і потім приймається один шлях, якщо умова істинна, або інший шлях, якщо умова хибна?

а) інструкція `if`;

б) структура ухвалення рішення з єдиним варіантом;

в) структура прийняття рішення із двома альтернативними варіантами;

г) послідовна.

5. Яка інструкція використовується для написання структури прийняття рішення з єдиним варіантом?

а) переходу за умовою;

б) якщо;

в) `if-else`;

г) виклику за умовою.

6. Яка інструкція використовується для написання структури прийняття рішення із двома альтернативними варіантами?

а) переходу за умовою;

б) якщо;

в) `if-else`;

г) виклику за умовою.

7. Якими є оператори `and`, `or` і `not`:

а) реляційні;

б) логічні;

в) умовні;

г) трикомпонентні.

8. Нижче наведений фрагмент коду містить кілька вкладених інструкцій `if-else`. На жаль, вони були написані без належного вирівнювання та виділення відступом. Перепишіть цей фрагмент і застосуйте відповідні правила вирівнювання та виділення відступом.

```
if score >= A score:
print ('Ваш рівень - A.')
else:
if score >= У score:
print ('Ваш рівень - B.')
```

```
else:
if score >= 3 score:
print ('Ваш рівень - C.')
else:
if score >= D score:
print('Ваш рівень - D.')
else:
print ('Ваш рівень - F.')
```

9. Припустимо, що дані змінні a = 2, в = 4 і c = 6. Обведіть "I" та "H" для кожного наведеної нижче умови, щоб показати, чи є значення істинним або хибним:

```
a == 4 or b > 2    I      H
6 <= c and a > 3   I      H
1 != ь and c != 3  I      H
a >= -1 or a <= ь  I      H
not (a > 2)        I      H
```

10. Що покаже наведений нижче фрагмент коду?

```
if 'z' <'a':
    print ( 'z менше a.')
else:
     print ( 'z не менше a.')
```

11. Що покаже наведений нижче фрагмент коду?

```
s1 = 'Нью-Йорк'
s2 = 'Бостон'
if s1 > s2:
    print(s2)
    print (s1)
else:
    print (s1)
    print (s2)
```

Лабораторна робота №3

1. Перерахуйте характеристики типу даних «список».
2. Як перевірити наявність елемента у списку?
3. Чим відрізняються методи `append()` та `extend()`?
4. Які параметри можна передавати під час зрізів списків?
5. Що станеться зі списком lst1 у першому та другому випадках? Поясніть результат. Випадок 1 – `IDE lst1 = [1, 2, 3, 14, 33, 1, 9] lst2 = [1, 2, 3, 14, 33, 1, 9] lst2.append(789)` Випадок 2 – `IDE lst1 = [1, 2, 3, 14, 33, 1, 9] lst2 = lst1 lst2.append(789).`
6. Які засоби створення списку з символів рядка ви знаєте?

7. Елементи множини є впорядкованими чи неупорядкованими?
8. Чи дозволяє множина зберігати повторювані елементи?
9. Як створити порожню множину?
10. Які елементи будуть зберігатися в множині `rnyset` після виконання наведеної нижче інструкції?

```
rnyset = set ( 'Юпітер')
```

11. Які елементи будуть зберігатися в множині rnyset після виконання наведеної нижче інструкції?

```
rnyset = set (25)
```

12. Які елементи будуть зберігатися в множині `rnyset` після виконання наведеної нижче інструкції?

```
rnyset = set ( '''' eee ююю яяя')
```

13. Які елементи будуть зберігатися в множині `rnyset` після виконання наведеної нижче інструкції?

```
rnyset = set ((1, 2, 2, 3, 4, 4, 4])
```

14. Які елементи будуть зберігатися в множині `rnyset` після виконання наведеної нижче інструкції?

```
rnyset = set ([ '''''', 'eee', 'ююю', 'яяя'])
```

15. Як визначається кількість елементів у множині?
16. Які елементи будуть зберігатися в множині `rnyset` після виконання наведеної нижче інструкції?

```
rnyset = set ((10, 9, 8])
rnyset.update ([1, 2, 3])
```

17. Які елементи будуть зберігатися в множині `myset` після виконання наведеної нижче інструкції?

```
myset = set ([10, 9, 8])
myset. update ( 'абв')
```

18. У чому різниця між методами `discard ()` і `remove ()`?
19. Як визначити, чи належить певний елемент множині?
20. Які елементи будуть зберігатися в множині `set3` після виконання наведеної нижче інструкції?

```
setl set ([lO, 20, 30])
set2 set ([100, 200, 300])
set3 setl.union (set2)
```

21. Які елементи будуть зберігатися в множині `set3` після виконання наведеної нижче інструкції?

```
setl set ([l, 2, 3, 4])
set2 set ([3, 4, 5, 6])
```

```
set3 setl.intersection (set2)
```

22. Які елементи будуть зберігатися в множині `set3` після виконання наведеної нижче інструкції?

```
setl set ([1, 2, 3, 4])
set2 set ([3, 4, 5, 6])
set3 setl.difference (set2)
```

23. Які елементи будуть зберігатися в множині `set3` після виконання наведеної нижче інструкції?

```
setl set ([1, 2, 3, 4])
set2 set ([3, 4, 5, 6])
set3 set2.difference (setl)
```

24. Які елементи будуть зберігатися в множині `set3` після виконання наведеної нижче інструкції?

```
setl set ([ 'а', '6', 'в'])
set2 set ([ '6', 'в', 'г'])
set3 setl.symmetric_difference (set2)
```

25. Погляньте на наведений нижче фрагмент коду:

```
setl set ([1, 2, 3, 4])
set2 set ([2, 3])
```

26. У якому випадку множина є підмножиною та надмножиною?

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

Основна література до навчальноїдисципліни

1. Shovic John and Simpson Alan. Python All-in-One. For Dummies , 2019. 720p
2. Bhasin H., Python Basics: A Self-Teaching Introduction. Mercury Learning and Information, 2019. 450p.
3. Irv Kalb, Learn to Program with Python 3: A Step-by-Step Guide to Programming 2nd ed. Edition .Apress, 2018. 371p.
4. Okken Brian. Python Testing with pytest: Simple, Rapid, Effective, and Scalable. 2017. 256p.
5. Lutz Mark, Learning Python, 5th Edition. O'Reilly Media, 2017. 1648p.
6. Python Tricks and Tips Magazine: 5th Edition 2021: Gain Insider Skills Kindle Edition – Mehedi Hzn Press Publications, 2021.164p.

Додаткова література

7. Lamy Jean-Baptiste Ontologies with Python. Apress, 2021
8. Reuven M. Lerner Python Workout 50 ten-minute exercises – Manning, 2020. 249p.
9. Robert C. Matthews Coding in Python: A Comprehensive Beginners Guide to Learn the Realms of Coding in Python. Independently published, 2020. 211p.
10. Bill Lubanovic, Introducing Python : Modern Computing in Simple Packages. O'Reilly Media, Inc, USA, 2020.500p.
11. Сысоева М.В., Сысоев И.В. Программирование для .нормальных. с нуля на языке Python: Учебник. В двух частях. Часть 1 / Ответственный редактор: В.Л.Черный : М.: Базальт СПО; МАКС Пресс. 2018. 176 с.

Інтернет ресурси

12. Beginner's Guide to Python [Електронний ресурс]. – Режим доступу: https://wiki.python.org/moin/BeginnersGuide.
13. The Python Standard Library [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/index.html
14. https://wiki.python.org/moin/BeginnersGuide
15. https://docs.python.org/3/library/index.html
16. https://docs.python.org/3/tutorial/index.html
17. https://programfiles.info/python/tipy-dannyh-v-python/

Навчальне видання

КОЗУЛЯ Марія Михайлівна
КОЗУЛЯ Тетяна Володимирівна

ПОЧАТОК РОБОТИ З PYTHON І РОБОТА З ДАНИМИ

Лабораторний практикум
з навчальної дисципліни
«Основи програмування Python (дисципліна вибору 02)»
Частина перша
для студентів спеціальностей
122 «Комп'ютерні науки»
126 «Інформаційні системи та технології»

Відповідальний за випуск Годлевський М.Д.
Роботу до видання рекомендував Гамаюн І.П.

В авторській редакції

План 2022 р., поз. 27

Підп. до друку __.__.202_. Гарнітура Times New Roman.
Ум. друк. арк. Ум. друк. арк. 3,1.

Самостійне електронне видання