**Лекція 1**

**Тема . Введення в програмування на мові Python**

План

1. Історія створення та особливості мови програмування Python
2. Основні типи даних
3. Pядки. Базові операції над рядками. Методи рядків.
4. Логічний (булевий) тип. Оператори порівняння. Логічні оператори.
5. Умовний оператор if
6. Списки. Кортежі. Словники. Множини
7. Цикли. Цикл з умовою (while)
8. Цикл обходу послідовності (for). Дострокове завершення циклу
9. **Історія створення та особливості мови програмування Python**

Комп’ютерна програма – набір інструкцій у вигляді слів, цифр, кодів, схем, символів, виражених у формі, придатній для зчитування комп’ютером, які приводять його у дію для досягнення певної мети або результату.

Мову програмування Python створено приблизно в 1991 році голландцем Гвідо ван Россумом. Свою назву – «Пайтон» (або «Пітон») – мова отримала від назви телесеріалу «Повітряний цирк Монті Пайтона», а не класу плазунів. Після того, як Гвідо ван Россум розробив мову, він виклав її в мережу Інтернет, до її поліпшення приєдналося співтовариство провідних програмістів .

Мова програмування Python активно вдосконалюється і в даний час, постійно виходять нові версії. Офіційний сайт мови: http://python.org.

*Особливості мови програмування Python* –це інтерпретована мовапрограмування: вихідний код частинами перетворюється в машинний у процесі його читання спеціальною програмою – інтерпретатором.

Крім того, мова програмування Python має зрозумілий синтаксис, читати код досить легко, тому що він має мінімум допоміжних елементів (дужок, розділових знаків), а правила мови змушують програмістів робити необхідні відступи. Слід зазначити, що добре систематизований та оформлений текст з невеликою кількістю відволікаючих елементів читати і розуміти значно легше.

Мова програмування Python – це повноцінна (універсальна) мова програмування, яка підтримує об’єктно-орієнтоване програмування.

**Області застосування Python**

Будучи вдало спроектованою мовою програмування, Python прекрасно підходить для вирішення щоденних реальних завдань. Він має широкий спектр застосувань: як інструмент управління іншими програмними компонентами і для реалізації самостійних програм.

Python має величезну кількість високоякісних вже готових модулів, які розповсюджуються безкоштовно, які ви можете використовувати в будь-якій частині програми. У модулі вже реалізовані багато потрібних вам деталей програми. Написання програми з використанням вже готових модулів можна порівняти з будівництвом збірного каркасного будинку: окремі деталі: фундамент, стіни, дах, комунікації вже зроблені до вас, вам потрібно тільки вибрати відповідні деталі і зібрати разом. Модулі підключаються за допомогою команди import, яка присутня на початку кожного прикладу.

В даний час Python застосовується в самих різних областях, таких як:

Системне адміністрування;

Написання графічних інтерфейсів користувача;

Веб програмування;

Програми для роботи з базами даних;

Ігри, штучний інтелект;

Програмування математичних і наукових обчислень.

**Інтерактивний режим і перша програма**

Після завантаження та установки Python відкриваємо IDLE (Integrated Development and Learning Environment – середовище розробки на мові Python, що поставляється разом з дистрибутивом). Запускаємо IDLE (спочатку запускається в інтерактивному режимі). Далі піде запрошення до введення (>>>).

Тепер можна починати писати першу програму. Традиційно, першою програмою буде «Hello, world». Щоб написати «Hello, world» на Python, достатньо всього одного рядка:

**>>> print ( 'Hello, World')**

Функція print виводить дані на екран.

Інтерпретатор виконує команди порядково: пишеш рядок, натискаєш <Enter>, інтерпретатор виконує її, спостерігаєш результат.

Для того, щоб створити нове вікно для написання скрипта, в інтерактивному режимі IDLE виберіть File → New File (або натисніть <Ctrl> + N). У вікні, спробуйте ввести наступний код:

name = i n p u t ( 'Як вас звати?')

p r i n t ( 'Привіт,', name)

Функція input зчитує дані, введені з клавіатури, і записує їх в змінну name.

Перший рядок друкує питання ( «Як вас звати?»), Очікує, поки ви надрукуєте що-небудь і натиснете <Enter> і зберігає введене значення в змінній name.

У другому рядку використовується функція print для виведення тексту на екран, в даному випадку для виведення "Привіт," і того, що зберігається в змінній name.

Для «складання» рядків можна також скористатися оператором +:

p r i n t ('Привіт,' + s t r (name) + '!').

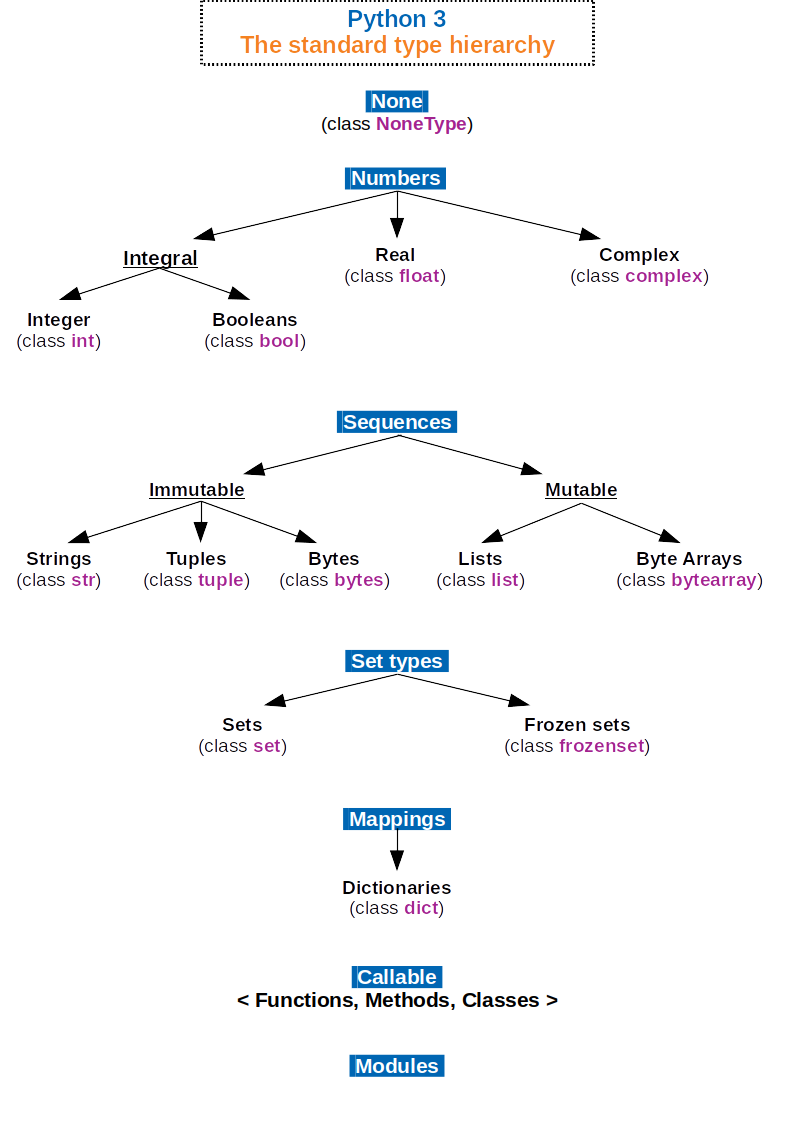
1. **Основні типи даних**

В Python є безліч вбудованих типів даних. Всі типи діляться на прості і складні. Змінні простих типів складаються з одного значення, до них відносяться числа (всього 3 типу: цілі, дійсні і комплексні) і логічні змінні. Змінні складових типів складаються з набору значень і, в свою чергу, діляться на незмінні, для яких не можна змінювати значення елементів, додавати або вилучати елементи, і змінювані, для яких все це можна робити. До незмінних типім відносяться строки і кортежі, до змінюваних - списки, словники і множини. Щоб отримати коректно працюючу програму, важливо розуміти, до якого типу належить змінна. В Python тип змінної не оголошується, а автоматично визначається при присвоєнні їй значення.

Python підтримує *динамічну типізацію*, тобто, тип змінної визначається лише під час виконання. З базових типів слід зазначити підтримку *цілих чисел* довільної довжини і *комплексних чисел*. Python має багату бібліотеку для роботи з *рядками*, зокрема, кодованими в юнікоді.

З колекцій Python підтримує *кортежі (tuples), списки (масиви), словники (асоціативні масиви)* і від версії 2.4, *множини*.

Система класів підтримує множинне успадкування і метапрограмування. Будь-який тип, включаючи базові, входить до системи класів, й за необхідності можливе успадкування навіть від базових типів.

1

**Числа. Арифметичні операції з числами. Модуль math**

Числа в Python діляться на:

• цілі,

• дійсні (з плаваючою точкою розміром в 64 біта),

• комплексні (з плаваючою точкою розміром в 128 біт).

Python підтримує динамічну типізацію, тобто тип змінної визначається лише під час виконання. Змінна може бути перевизначена, при цьому її тип зміниться:

>>> a = 2

>>> a

2

>>> a = 2.0

>>> a

2.0

>>> a = 2 + 3 j

>>> a

(2 + 3 j)

Інструкція *a = 2* створює числовий об'єкт-змінну з цілочисельним значенням *2* і привласнює йому ім'я *a*. Далі інструкція *a = 2.0* створює новий числовий об'єкт з дійсним значенням *2.0* і привласнює вже йому ім'я *a*.

А об'єкт зі значенням *2* видаляється з пам'яті автоматично збирачем сміття (англ. Garbage collector, спеціальний процес, періодично звільняє пам'ять, видаляючи об'єкти, які вже не будуть затребувані додатками), так як було втрачено останнє посилання на цей об'єкт. Потім інструкція *a = 2 + 3j* створює числовий об'єкт з комплексним значенням і все з тим же ім'ям, а об'єкт з дійсним значенням видаляється.

При операціях з числами існує загальне правило: результат буде того ж типу, що і операнди, якщо операнди різних типів, то результат буде приведений до найбільш загального з усіх наявних. Порядок спільності природний: цілі → дійсні → комплексні. Наступний приклад ілюструє це правило:

>>> a = 2

>>> b = 2 + 3 j

>>> a + b

(4 + 3 j)

Так само корисно запам'ятати, що для перевірки типу будь-якого значення і будь-якої змінної можна використовувати функцію type ():

>>> a = 5

>>> b = 17.1

>>> c = 4 + 2 j

>>> type (a); type (b); type (c)

< сlass 'int'>

< сlass 'float'>

<сlass 'complex'>

Можна явно перетворювати значення будь-якого типу за допомогою відповідних функцій int (), float () або complex ():

>>> a = 5

>>> i n t (a)

5

>>> f l o a t (a)

5.0

>>>

  сomplex (a)

(5 + 0 j)

Важливо пам'ятати, що комплексне число *не можна* перетворити за допомогою функцій int() і float() до цілого або дійсного. Функція int() відкидає дробову частину числа, а не округлює його:

>>> a = 4.3

>>> i n t (a)

4

>>> b = -4.3

>>> i n t (b)

-4

А тепер про підводні камені і відмінність третього Python від другого при роботі в скриптовому (текстовому) режимі. Напишемо в текстовому режимі в Python 3.x просту програму:

a = i n p u t ( 'Введіть перше число:')

b = i n p u t ( 'Введіть друге число:')

p r i n t (a + b)

Введіть перше число: 10

Введіть друге число: 4

104

Отримали зовсім не те, що очікували. Python версій 2.x підтримував автоматичне визначення типу змінної при введенні з клавіатури. Однак це часто викликало різні помилки або складності. Тому при проектуванні версії 3.0 було вирішено відмовитися від автоматичного визначення типу і завжди зчитувати дані як рядок. Тому, якщо ми хочемо дійсно скласти два числа, програму доведеться переписати:

a = i n t (i n p u t ('Введіть перше число:'))

b = i n t (i n p u t ('Введіть друге число:'))

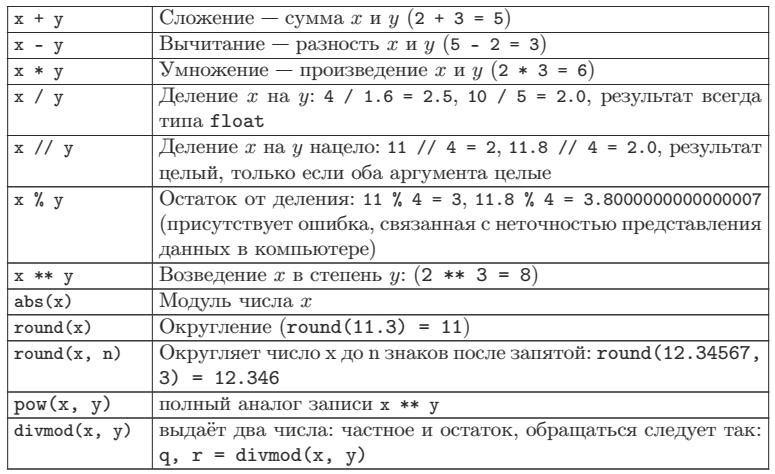
p r i n t (a + b)

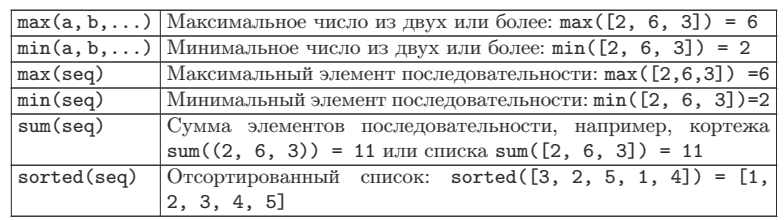
Введіть перше число: 10

Введіть друге число: 4

14

Python підтримує всі основні арифметичні оператори (див. Табл. 1.1).

Таблиця 1.1. Арифметичні операції з числами і вбудовані функції з одним і двома аргументами

Таблиця 1.2. Вбудовані функції c послідовностями або довільним числом аргументів

Те, що описані функції є вбудованими, означає, що вони доступні без всяких додаткових дій. Однак вбудованих функцій трохи, набагато більше функцій знаходиться в стандартній бібліотеці - наборі модулів, що поставляються завжди разом з інтерпретатором Python.

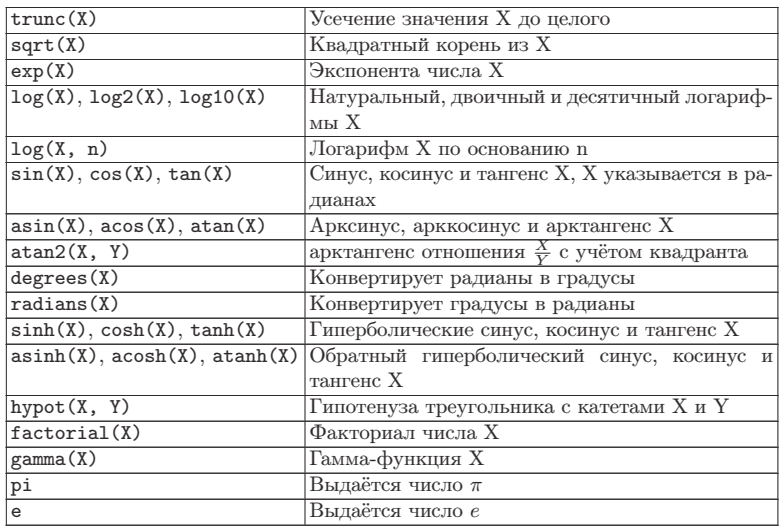
Завантаження модулів в Python здійснюється за допомогою оператора import:

import math

t = math .sin (math .pi / 6)

v = math .log (math .e)

p r i n t (t, v)

Таблиця 1.3. Найбільш вживані функції і константи модуля math

1. **Pядки. Базові операції над рядками. Методи рядків**

Рядки в Python - впорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути представлено в текстовій формі.

При цьому окремого символьного типу в Python немає, символ - це рядок довжини 1. Більш того, символи як елементи рядка теж є рядками. Робота з рядками в Python дуже зручна. Існує кілька варіантів написання рядків:

>>> S1 = 'Alice said: "Hi, Anne!" '

>>> S2 = "Anne answered: 'Hi, Alice'"

Рядки можна писати в потрійних лапках для запису багаторядкових блоків тексту:

>>> S = '''Це

довгий

рядок '''

>>> S

'Це \nдовгий \nрядок'

>>> p r i n t (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.

Тут доречно згадати про те, як в Python при написанні коду програми робити коментарі. Однорядкові коментарі починаються зі знака решітки «#», багаторядкові - починаються і закінчуються трьома подвійними лапками «"""».

Числа можуть бути перетворені в рядки за допомогою функції ***str ().*** Наприклад, *str (123)* дасть рядок '123'. Якщо рядок є послідовністю знаків-цифр, то вона може бути перетворена в ціле число в допомогою функції

*int (): int ( '123')* дасть в результаті число 123, а в дійсне за допомогою функції *float (): float ('12 .34 ')* дасть в результаті число 12.34. Для будь-якого символу можна дізнатися його номер (код символу) за допомогою функції ***ord ()*,** наприклад, *ord('s')* дасть результат 115. І навпаки, отримати символ по числовому коду можна за допомогою функції ***chr ()***, наприклад *chr (100 )* дасть результат 'd'.

**Базові операції над рядками**

Існують кілька різних підходів до операцій над рядками.

• *«Арифметичні операції».* Для рядків подібно числам визначені оператори складання + і множення \*. В результаті складання вміст двох рядків записується поспіль в новий рядок, наприклад:

>>> S1 = 'Py'

>>> S2 = 'thon'

>>> S3 = S1 + S2

>>> p r i n t (S3)

Python

Множення визначено для рядка і цілого позитивного числа, в результаті виходить новий рядок:

>>> S3 \* 4

'PythonPythonPythonPython'

>>> 2 \* S3

'PythonPython'

• *Функція len ()* вирахує довжину рядка, результат має цілочисельний тип. Наприклад, len ('Python') видасть 6.

• *Доступ за індексом*. Можна звернутися до будь-якого елементу (символу) рядки по його номеру, нумерація починається з 0 (перший елемент рядка S має номер 0, останній - len (S) -1):

>>> S = 'Python'

>>> S [0]

'P'

>>> S [-1]

'N'

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

При використанні індексів необхідно пам'ятати, що рядки в Python відносяться до категорії незмінних послідовностей: не можна поміняти значення того чи іншого символу, а можна лише створити новий рядок.

>>> S = 'abc'

>>> S [1] = 'w'

Traceback (most recent call last):

File "<pyshell # 68>", line 1, i n <module>

S [1] = 'w'

TypeError: 'str' object does not support item assignment

>>> S = S [0] + 'w' + S [2]

>>> S

'awc'

*• Зрізи* дозволяють скопіювати або використовувати у виразах частина рядка. Оператор вилучення зрізу з рядка виглядає так: S [n1: n2]. n1 - це індекс початку зрізу, а n2 - його закінчення, причому символ з номером n2 в зріз вже не входить! Якщо зазначений негативний індекс, це означає, що будь-який індекс -n аналогічний записи 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, afternoon, evening'

>>> Day [0: 7]

'Morning'

>>> Day [9: -9]

'Afternoon'

>>> Day [-7:]

'Evening'

Можна витягати символи не підряд, а через певну кількість. У такому випадку оператор індексування виглядає так: [n1: n2: n3]; n3 - це крок, через який здійснюється вибір елементів:

>>> flag = ' Morning Evening Morning '

>>> flag [:: 8]

'MEM'

Зверніть увагу, що в зрізі рядки s можуть бути пропущені і перший, і другий індекси одночасно: замість них підставляються 0 і len (s) відповідно.

• Оператор in дозволяє дізнатися, чи належить підрядок в рядку. Оператор повертає логічне значення: True, якщо елемент в складі рядка зустрічається і False, якщо немає:

>>> S = 'Python'

>>> SubS = 'th'

>>> SubS in S

True

• Функції min і max застосовні також і до рядків: min (S) визначає і виводить (повертає) символ з найменшим кодом - номером в кодової таблиці. наприклад:

>>> S = 'Python'

>>> m i n (S)

'P'

Функції max (S) повертає символ з найбільшим значенням (кодом)

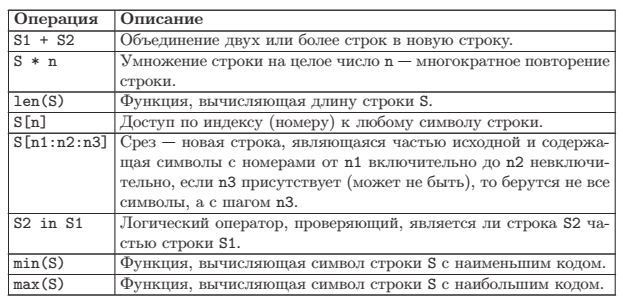
>>> S = 'Python'

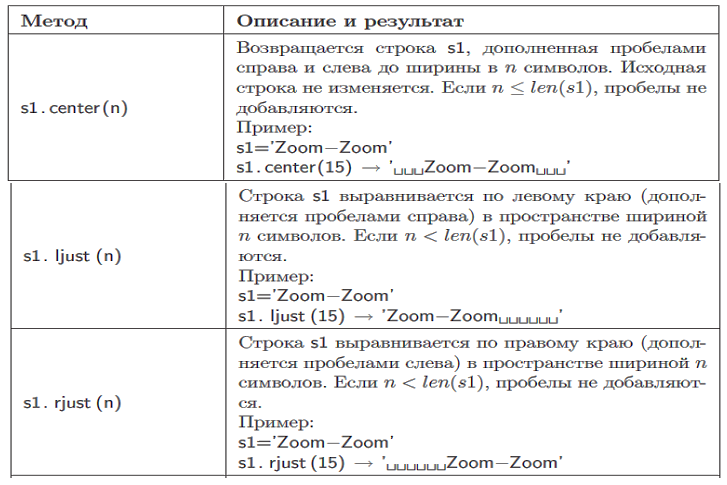
>>> m a x (S)

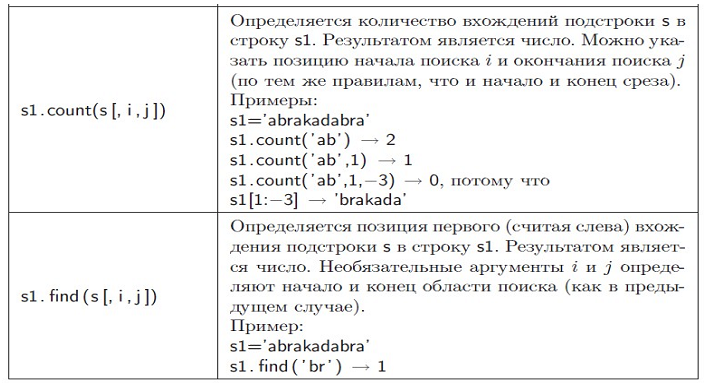
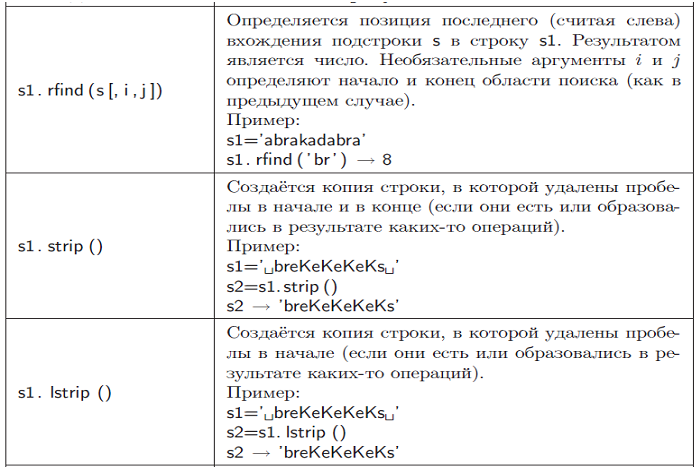
'Y'

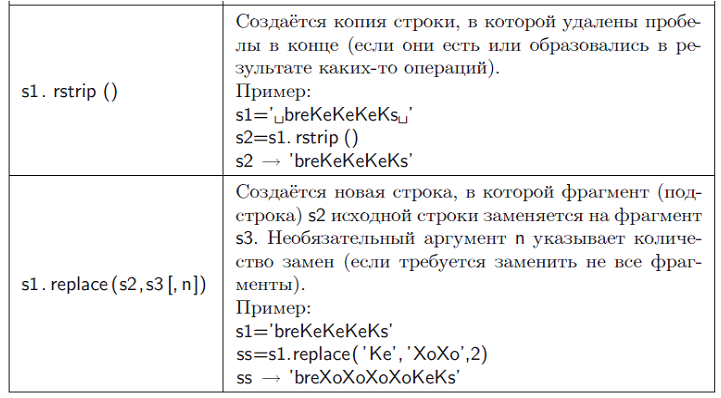
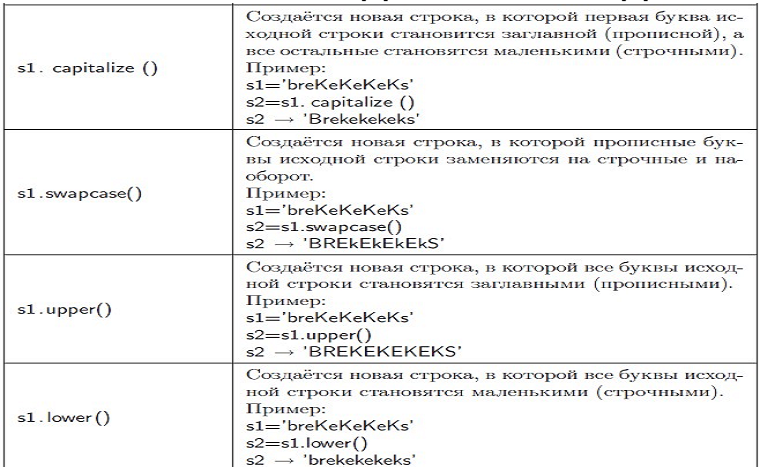
**Методи рядків**

Крім операторів, функцій і зрізів значна кількість операцій над рядками є у вигляді методів. Основна відмінність методів і функцій - синтаксична; так, більшість методів раніше були функціями стандартного модуля string, в якому тепер залишилися майже тільки різні константи. Зверніть увагу, як записуються методи об'єкта: об'єкт.метод (), наприклад, S.isdigit (). Методи - це по суті функції, у яких в якості першого аргументу виступає сам об'єкт, метод якого викликали. наприклад, виклик методу S.isdigit () видасть логічне значення: True, якщо всі символи рядка S і False інакше.

Таблиця 1.4. Базові операції над рядками

Таблиця 1.5. Базові методи для рядками





1. **Логічний (булевий) тип. Оператори порівняння. Логічні оператори**

Логічний (булевий) тип може приймати одне з двох значень True (істина) або False (брехня). У мові Python булевий тип даних позначається як bool, для приведення інших типів даних до булевого існує функція bool (), що працює за такими угодами:

• рядки: порожній рядок - брехня, непорожній рядок - істина.

• числа: нульове число - брехня, ненульове число (в тому числі і менше одиниці) - істина.

• функції - завжди істина.

Для роботи з алгеброю логіки в Python крім логічного типу даних передбачені оператори порівняння:

• «>» більше,

• «<» менше,

• «==» одно (одиночне «=» зарезервовано за оператором присвоювання),

• «! = »Не дорівнює,

• «> =» більше або дорівнює,

• «<=» менше або дорівнює.

Ось простенька програма, що обчислює різні логічні вирази:

x = 12 - 5

h1 = x == 4

h2 = x == 7

h3 = x! = 7

h4 = x! = 4

h5 = x> 5

h6 = x <5

p r i n t (h1, h2, h3, h4, h5, h6)

Її висновок:

False True False True True False

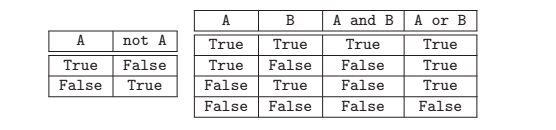
З логічних змінних і виразів можна будувати більш складні (складені) логічні вирази за допомогою логічних операторів: *not* (заперечення, логічне НЕ), *or* (логічне АБО) і *and* (логічне І):

• *x and y* - логічне **«І»** (множення). Приймає значення True (істина), тільки коли x = True і y = True. Приймає значення False (брехня), якщо хоча б одна з змінних дорівнює False, або обидві змінні False.

• *x or y* - логічне **«АБО»** (додавання). Приймає значення True (істина), якщо хоча б одна з змінних дорівнює True, або обидві змінні True. Приймає значення False (брехня), якщо x == y == False.

• *not x* - логічне **«НЕ»** (заперечення). Приймає значення True (істина), якщо x == False. Приймає значення False (брехня), якщо x == True.

Таблиця 1.6. Таблиці істинності логічних функцій для логічного типу.



>>> b o o l ([1, 2]) a n d b o o l ([1]) o r b o o l (13)

True

>>> b o o l ([1, 2]) a n d b o o l ([1]) a n d b o o l (13)

True

1. **Умовний оператор if**

В Python найпростіша форма умовного оператора має вигляд:

**i f** *<логічний вираз>:*

*<Дії, що виконуються, коли логічний вираз приймає значення True>*

У такій формі дії після двокрапки виконуються, якщо логічний вираз істинно. Якщо ж умова помилкова, програма нічого не робить і переходить до оператора, наступного за if. Коли потрібно виконати різні дії, якщо умова істинна і якщо вона помилкова, використовується наступна повна форма:

**i f** *<логічний вираз>:*

*<Дії, що виконуються, коли логічний вираз приймає*

*значення True>*

**e l s e:**

*<Дії, що виконуються, коли логічний вираз приймає значення False>*

Нарешті, якщо потрібно послідовно перевірити кілька умов, використовується форма з додатковим оператором **elif** (скорочення від else if):

**i f** *<логічний вираз>:*

*<Дії, що виконуються, якщо логічний вираз приймає значення True>*

**e l i f** *<другий логічний вираз>:*

*<Дії, що виконуються, якщо друге логічне вираз приймає значення True>*

**e l i f** *<третя логічне вираз>:*

*<Дії, що виконуються, якщо третя логічне вираз приймає значення True>*

. . .

**e l s e:**

*<Дії, що виконуються, якщо жодне з логічних виразів не приймає значення True>*

Зверніть увагу, що після двокрапки в конструкціях типу if, else, elif завжди йде блок, виділений відступом вправо. Більшість редакторів коду, в тому числі і IDLE, роблять цей відступ автоматично. Те, що виділено відступами, і є *тіло оператора*, а то, що до двокрапки, називається *заголовком*.

Наступна проста програма перевіряє, чи ділиться перше введене число на друге націло:

a = int (input ('Введіть перше число:'))

b = int (input ('Введіть друге число:'))

if a% b == 0:

print ("Yes")

e l s e:

print ("No")

**6. Списки. Кортежі. Словники. Множини**

Списки в мові програмування Python, як і рядки, є впорядкованими послідовностями значень. Однак, на відміну від рядків, списки складаються не з символів, а з різних об'єктів (значень, даних), і заключаються не в лапки, а в квадратні дужки [ ]. Об'єкти відокремлюються 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. Ось 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 рядків

['Київ', 'Львів', 104, 18] # змішаний список

[[1, 0, 0], [0, 1, 0], [0, 0, 1]] # список списків

Як і над рядками, над списками можна виконувати операції об’єднання і повторення:

>>> [6, 'Жовтня', 2015] + [16, 'грудня', 2015]

[6, 'Жовтня', 2015, 16, 'грудня', 2015]

>>> [2, 3, 4] \* 2

[2, 3, 4, 2, 3, 4]

За аналогією з символами (елементами) рядки можна отримувати доступ до елементів списку по їх індексах, складати їх, витягувати зрізи, вимірювати довжину списку, дізнаватися тип даних:

>>> list1 = ['P', 'y', 'th', 'o', 'n', 3.4]

>>> len (list1)

6

>>> list1 [0] + list1 [1]

'Py'

>>> list1 [0]

'P'

>>> list1 [0: 5]

['P', 'y', 'th', 'o', 'n']

>>> list1 [5:]

[3.4]

>>> type (list1)

<Сlass 'list'>

На відміну від рядків, списки - це змінні послідовності. Якщо уявити рядок як об'єкт в пам'яті, то коли над ним виконуються операції конкатенації і повторення, то цей рядок не змінюється, а в результаті операції створюється інший рядок в іншому місці пам'яті. У рядок не можна додати новий символ або видалити існуючий, не створивши при цьому нового рядка. Зі списком інша справа. При виконанні операцій нові списки можуть не створюватися, а буде змінюватися безпосередньо оригінал. Зі списків можна видаляти елементи, додавати нові. При цьому слід пам'ятати, багато що залежить від того, як ви розпоряджаєтеся змінними.

Символ в рядку змінити не можна, елемент списку - можна:

>>> mystr = 'Python'

>>> mylist = [ 'P', 'y', 't', 'h', 'o', 'n']

>>> mystr [1] = 'i'

Traceback (most recent call last):

File "<pyshell # 2>", line 1, i n <module>

mystr [1] = 'i'

TypeError: 'str' objet does n o t support item assignment

>>> mylist [1] = 'i'

>>> mylist

[ 'P', 'i', 't', 'h', 'o', 'n']

У списку можна замінити цілий зріз:

>>> mylist [: 3] = [ 'Y', 'e', ​​'s']

>>> mylist

[ 'Y', 'e', ​​'s', 'h', 'o', 'n']

Для списку можна створювати його копію:

>>> list1 = [ 'P', 'y', 't', 'h', 'o', 'n']

>>> list2 = list1 .copy () # Створення копії списку

>>> list2 [1] = 'i "

>>> list2, list1

([ 'P', 'i', 't "," h "," o', 'n'], [ 'P', 'y', 't "," h "," o', 'n '])

Список list2 змінився, а список list1 - ні. Для списку можна створити друге посилання на список. Увага! При створенні другого посилання дані не копіюються, просто ці дані тепер мають два імені, тому зміна list1 буде приводити до зміни list2:

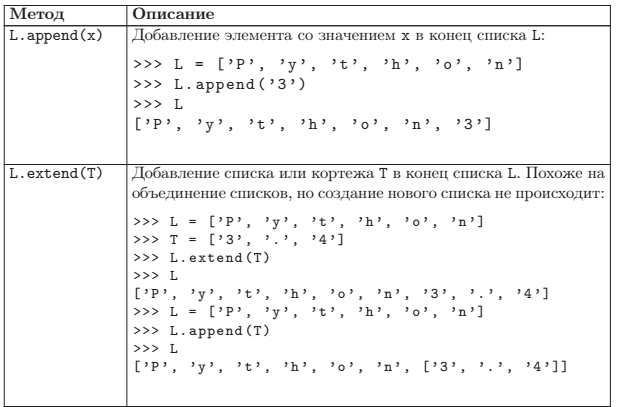
>>> list2 = list1 # Створення другого посилання, а не копії

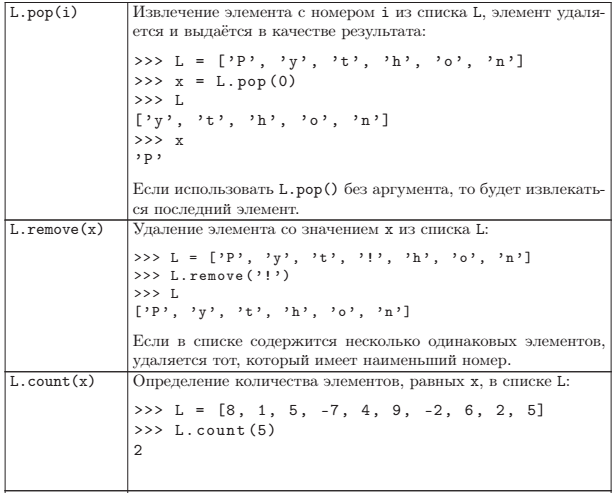
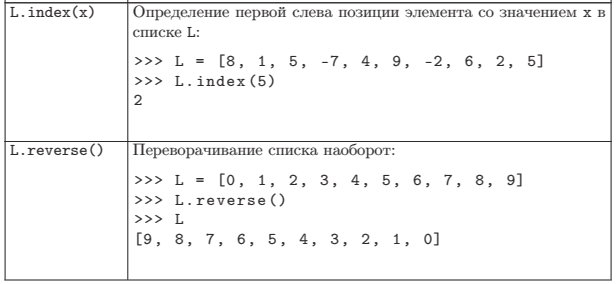
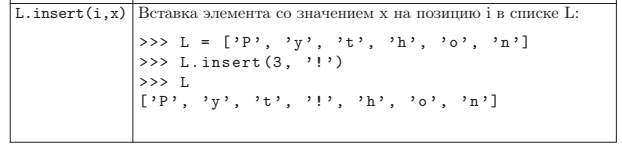
>>> list2 [1] = 'i'

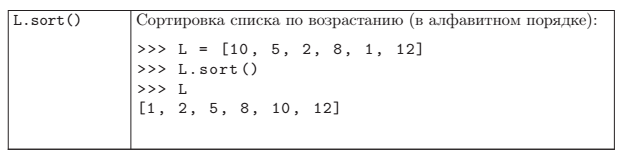
>>> list2, list1

(['P', 'i', 't', 'h', 'o', 'n'], ['P', 'i', 't', 'h', 'o', 'n '])

Змінилися обидва списки. Для створення копії передбачений більш простий синтаксис, ніж використання стандартного методу copy: достатньо взяти зріз списку від початку і до кінця: list3 = list1 [:] еквівалентно тому, що ми написали б list3 = list1.copy ().

Таблиця 1.7. Методи списку

 До списків застосовуть деякі стандартні функції, наприклад, знайома нам ***len***, до якої звертатися потрібно в такий спосіб: ***len(mylist)***. Функція ***sum*** підраховує суму елементів списку, якщо всі вони числового типу. Функція ***range*** дозволяє сформувати діапазон значень цілих чисел.

У найзагальнішому випадку ***range*** приймає 3 аргументу: *початок діапазону, кінець* (завжди береться НЕ включно) *і крок*. Зверніть увагу, що в Python 3.x ця функція не видає список, а виробляє спеціальний об'єкт-діапазон. Тому, щоб отримати список чисел, потрібно обов'язково явно перетворити результат за допомогою функції ***list***. Є ще функція ***sorted ()***, яка повертає новий список, відсортований за спаданням. Функції ***min*** і ***max*** знаходять максимальний і мінімальний елементи списку. Ось невелика програма з використанням цих функцій:

>>> A = list (range (0, 10, 1))

>>> A

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> sum (A)

45

>>> min (A)

0

>>> max (A)

9

В даному прикладі об'єкт A формується за допомогою ітератора range, а потім явно перетвориться до типу list.

>>> mylist1 = ['P', 'y', 't', 'h', 'o', 'n']

>>> mylist2 = [6, 10, 2015]

>>> 'y' i n mylist1

True

>>> 30 i n mylist2

False

>>> mylist2. append (['Програмування', 11.30])

>>> mylist2

[6, 10, 2015 року, ['Програмування', 11.3]]

>>> mylist2 [-1]

['Програмування', 11.3]

**Кортежі**

Список так само може бути незмінним, як і рядок, в цьому випадку він називається кортеж (tuple). Кортеж використовує менше пам'яті, ніж список. При оголошенні кортежу замість квадратних дужок використовуються круглі (хоча можна і зовсім без дужок). Кортеж не допускає змін, в нього не можна додати новий елемент, видалити або замінити існуючі елементи, але він може містити змінні об'єкти, наприклад, списки:

>>> ll = []

>>> A = (1, 2, 3, ll)

>>> A

(1, 2, 3, [])

>>> A [1] = 4

Traceback (most recent call last):

File "<pyshell # 22>", line 1, i n <module>

A [1] = 4

TypeError: 'tuple' objet does n o t support item assignment

>>> A [3]. append (3)

>>> p r i n t (A, ll)

(1, 2, 3, [3]) [3]

Видно, що пряма заміна елемента кортежу недопустима - викликає TypeError, так як тип tuple не підтримує зміну елементів, але якщо використовувати вбудований метод append у списку, що є елементом кортежу, цей список можна змінити.

Функція tuple () бере в якості аргументу рядок або список і перетворює його в кортеж, а функція list () переводить кортеж в список:

>>> B = l i s t (A)

>>> B

[1, 2, 3]

>>> C = t u p l e (B)

>>> C

(1, 2, 3)

Основна відмінність між кортежами і списками полягає в тому, що кортежі не можуть бути змінені. На практиці це означає, що у них немає методів, які б дозволили їх змінити: append (), extend (), insert (), remove (), pop (). Але можна взяти зріз від кортежу, так як при цьому створиться новий кортеж.

В Python можна використовувати кортежі, щоб привласнювати значення декільком змінним одразу:

>>> v = ('f', 5, True)

>>> (x, y, z) = v

>>> x

'F'

>>> y

5

>>> z

True

>>>

Кортежі в деяких випадках швидше, ніж списки. Але такі оптимізації в кожному конкретному випадку вимагають додаткових досліджень. Кортежі роблять код безпечніше в тому випадку, якщо у вас є «захищені від запису» дані, які не повинні змінюватися. Деякі кортежі можуть використовуватися в якості елементів множин і ключів словника (конкретно, кортежі, що містять незмінні значення, наприклад, рядки, числа та інші кортежі). Списки ніколи не можуть використовуватися в якості ключів словника, тому що списки – змінювані об'єкти.

**Словники**

Одним із складних типів даних поряд з рядками і списками в мові програмування Python є словники. Словник - це змінний (як список) невпорядкований (на відміну від рядків і списків) набір пар 'ключ: значення'. Словники виявляються дуже зручними об'єктами для зберігання даних і, по суті, є своєрідною заміною базі даних.

>>> animal = {'cat': 'кішка', 'dog': 'пес', 'bird': 'птах',

'Mouse': 'миша'}

>>> animal

{'Mouse': 'миша', 'cat': 'кішка', 'dog': 'пес', 'bird': 'птах'}

>>> t y p e (animal)

<Сlass 'dict'>

Зверніть увагу на фігурні дужки, саме з їх допомогою визначається словник. Такий тип даних в Python називається ***dict***. Якщо створити словник в інтерпретаторі Python (як і було зроблено), то після натискання <Enter> можна спостерігати, що послідовність виведення пар ‘ключ: значення ‘ не співпаде з

тим, як було введено. Справа в тому, що в словнику абсолютно не важливий порядок пар, і інтерпретатор виводить їх у випадковому порядку. Тоді як же отримати доступ до певного елемента, якщо індексація неможлива в принципі? В словнику доступ до значень здійснюється по ключам, які заключають в квадратні дужки (по аналогії з індексами рядків і списків):

>>> animal = { 'cat': 'кішка', 'dog': 'пес', 'bird': 'птах', 'mouse': 'миша'}

>>> animal [ 'cat']

'Кішка'

Словники, як і списки, є змінним типом даних: можна змінювати, додавати і видаляти елементи - пари 'ключ: значення'. Спочатку словник можна створити порожнім, наприклад, dic = {} і лише потім заповнити його елементами.

Додавання і зміна має однаковий синтаксис: словник [ключ] = значення. Ключ може бути, як уже існуючим (тоді відбувається зміна значення), так і новим (відбувається додавання елемента словника). Видалення елемента словника здійснюється за допомогою функції del (dic [key]) або методу

pop (key):

>>> dic = { 'cat': 'кішка', 'dog': 'пес', 'bird': 'птах', 'mouse': 'миша'}

>>> dic [ 'cat'] = 'кіт'

>>> dic

{ 'Mouse': 'миша', 'cat': 'кіт', 'dog': 'пес', 'bird': 'птах'}

>>> dic [ 'fox'] = 'лисиця'

>>> dic

{ 'Fox': 'лисиця', 'mouse': 'миша', 'cat': 'кіт', 'dog': 'пес', 'Bird': 'птах'}

>>> del (dic [ 'mouse'])

>>> dic

{ 'Fox': 'лисиця', 'cat': 'кіт', 'dog': 'пес', 'bird': 'птах'}

>>> dic.pop ( 'fox')

'Лисиця'

>>> dic

{ 'Bird': 'птах', 'cat': 'кіт', 'dog': 'пес'}

Тип даних ключів і значень словників не обов'язково повинен бути строковим:

>>> DicProg = {1: 'Pascal', 2: 'Python', 3: 'C', 4: 'Java'}

Словники - це широко використовуваний тип даних мови Python. Для роботи з ними існує ряд вбудованих методів і функцій. Метод keys () для словника повертає послідовність всіх використовуваних ключів в довільному порядку. Для визначення наявності певного ключа раніше був метод has\_key (), але у версії 3.0 замість нього є знайомий нам оператор in:

>>> DicProg .keys ()

dict\_keys ([1, 2, 3, 4])

>>> 1 i n DicProg

True

>>> 'Pascal' i n DicProg

False

**Множини**

[Множина](http://pythonguide.rozh2sch.org.ua/#%D0%9C%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0) схожа на словник, у якого відсутні значення, тобто має тільки ключі. Як і ключі словників, елементи множини повинні бути незмінними і хешованими. Це означає, що цілі числа, числа з плаваючою крапкою, рядки і кортежі можуть бути елементами множини, а списки, словники і самі множини - ні. Множина є неупорядкованою сукупністю об’єктів, в якій не може бути дублікатів.

Як і кортеж, множину можна створити на базі списку або інших послідовностей, елементи яких можна перебирати. Однак, на відміну від списків і кортежів, для множин **неважливий порядок елементів**. Множини часто використовуються для двох цілей: для видалення дублікатів елементів і для перевірки належності множині.

*Щоб створити множину, слід використовувати функцію****set()****або розмістити у фігурних дужках одне або кілька значень, розділених комами.*

>>> empty\_set = set()

>>> empty\_set set()

>>> numbers = {0, 1, 4, 5, 7, 9}

>>> numbers {0, 1, 4, 5, 7, 9}

Ви можете створити множину за допомогою функції set() зі списку, рядка, кортежу або словника, втративши всі значення, що повторюються. Для прикладу, поглянемо на рядок, який містить більш ніж одне входження деяких букв:

>>> set('office')

{'e', 'c', 'o', 'f', 'i'}

Множина містить тільки одне входження літери f, незважаючи на те, що у слові office є два входження цієї літери.

Для перевірки приналежності множині використовується операція in:

>>> numbers = {0, 1, 4, 5, 7, 9}

>>> 5 in numbers

True

>>> 30 in numbers

False

Множини Python підтримують класичні операції теорії множин, такі як **об’єднання** (|), **перетин** (&), **різницю** (-) і **виключаюче АБО** (ˆ).

Виконаємо деякі операції над множинами і зробимо пояснення:

>>> x = {0, 1, 4, 5, 7, 9}

>>> y = {1, 2, 3, 6, 7, 8, 9}

>>> x

{0, 1, 4, 5, 7, 9}

>>> y

{1, 2, 3, 6, 7, 8, 9}

>>> x.add(100)

>>> x

{0, 1, 4, 5, 100, 7, 9}

>>> x.remove(100)

>>> x

{0, 1, 4, 5, 7, 9}

>>> x | y

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

>>> x & y

{1, 9, 7}

>>> x - y

{0, 4, 5}

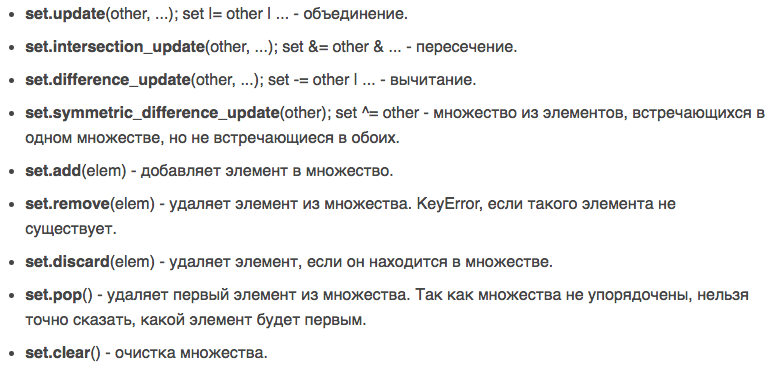
>>> y - x

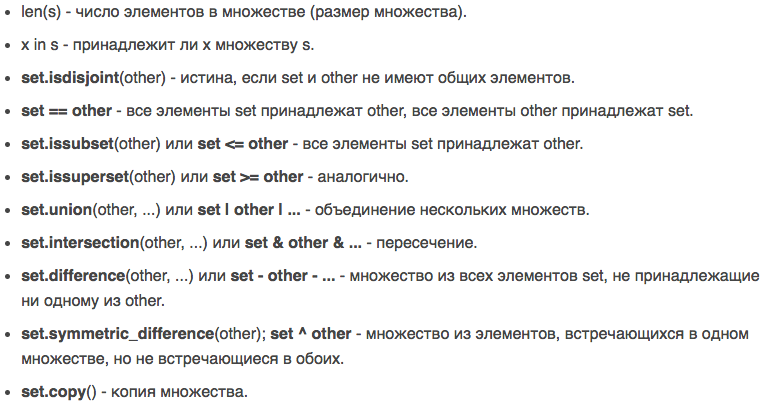
{8, 2, 3, 6}

>>> x ^ y

{0, 2, 3, 4, 5, 6, 8}

Для множин існує величезна кількість операцій, які наведені нижче





Множина **frozenset**

Оскільки множини є змінними і не хешуються, вони не можуть бути елементами інших множин.

Для вирішення цієї проблеми в Python існує ще один тип множин **frozenset**, який веде себе як множина, але не може змінюватися після створення. Виходить, що Frozenset це суміш множини і кортежу.

>>> off = set([0, 1, 2, 3, 4])

>>> on = frozenset(off)

>>> on

frozenset({0, 1, 2, 3, 4})

>>> on.add(50)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'frozenset' object has no attribute 'add'

>>> off.add(on)

>>> off

{0, 1, 2, 3, 4, frozenset({0, 1, 2, 3, 4})}

1. **Цикли. Цикл з умовою (while)**

Цикли - це інструкції, які виконують одну й ту ж послідовність дій багаторазово. Універсальним організатором циклу в мові програмування Python є цикл з умовою (конструкція while - поки логічний вираз повертає істину, виконувати певні операції). Конструкція while на мові Python може виглядати наступним чином:

a = початкове значення

while a оператор порівняння b:

дії

зміна a

дії

Ця схема сильно неповна, так як логічний вираз в заголовку циклу може бути більш складним, а змінюватися може змінна (або вираз) b.

Якщо умова ніколи не буде хибним, то не буде причин для зупинки циклу, і програма зациклиться. Найпростіший спосіб створити таку ситуацію:

w h i l e True:

p r i n t ( 'У попа була собака, він її любив.')

p r i n t ( 'Вона з'їла шматок м'яса - він її вбив.')

p r i n t ( 'Вирив ямку, закопав і на камені написав:')

Щоб такого не сталося в звичайній програмі, необхідно передбачити можливість виходу з циклу - хибність виразу в заголовку. Таким чином, змінюючи значення змінної в тілі циклу, можна довести логічний вираз до хибності. Цю змінену змінну, яка використовується в заголовку циклу while, зазвичай називають *лічильником*. Як і будь-якій змінній, їй можна давати довільні імена, однак дуже часто використовуються літери i і j.

Приклад використання циклу while: вивід перших n чисел Фібоначчі. Ряд Фібоначчі - ряд чисел, в якому кожне наступне число дорівнює сумі двох попередніх: 0, 1, 1, 2, 3, 5, 8, 13 і т. Д. Виведемо перші 10 чисел:

fib1 = 0

fib2 = 1

p r i n t (fib1)

p r i n t (fib2)

n = 10

i = 2

summa = 0

w h i l e i <= n:

summa = fib1 + fib2

p r i n t (summa)

fib1 = fib2

fib2 = summa

i = i + 1

1. **Цикл обходу послідовності (for). Дострокове завершення циклу**

Як правило, цикли for використовуються або для повторення будь-якої послідовності дій задане число раз, або для зміни значення змінної в циклі від деякого початкового значення до деякого кінцевого. Для повторення циклу задане n раз можна використовувати цикл for разом з функцією range:

for i in range (n):

тіло циклу

Як n може використовуватися числова константа, змінна або довільний арифметичний вираз (наприклад, 2 \*\* 10). Якщо значення n дорівнює нулю або від’ємне , то тіло циклу не виконається жодного разу.

Якщо задати цикл таким чином:

for i in range (a, b):

тіло циклу

то індексна змінна i буде приймати значення від a до b - 1 включно.

summa = 0

for i in range (1, n +1):

summa = summa + i

В мові програмування Python цикл for має найчастіше дещо інше застосування. Наприклад, список в Python відноситься до *ітерованих об'єктів*. Це означає, що його елементи можна обійти циклом for, причому змінна-лічильник буде на кожному кроці приймати значення чергового елемента циклу:

mylist = [12, 17.9, True, -8, False]

for j in mylist:

print (j)

Програма виведе всі елементи списку mylist в стовпчик:

12

17.9

True

-8

False

Наведений спосіб можна назвати обходом за значенням, оскільки автоматично створювана змінна j на кожному кроці приймає значення чергового елемента списку. Є ще один спосіб обійти список - за індексами, коли така ж змінна буде приймати номер чергового елемента:

mylist = [12, 17.9, True, -8, False]

f o r j in r a n g e (0, l e n (mylist), 1):

p r i n t (j)

Вивід буде зовсім інший:

0

1

2

3

4

Якщо написати:

mylist = [12, 17.9, True, -8, False]

for j in range (0, len (mylist), 1):

print (mylist [j])

то вивід буде такий же, як в першому прикладі.

За допомогою циклу for можна перебирати рядки, якщо не намагатися їх при цьому змінювати:

str1 = 'Привіт'

f o r i i n str1:

p r i n t (i, end = ' ')

Буде виведено:

П р и в і т

Тут можна бачити, що у функції print є параметр end. За замовчуванням end = '\ n' - символу нового рядка. У попередньому прикладі параметру end було присвоєно символ пробіл. Цикл for використовується і для роботи зі словниками:

dic = {'cat': 'кішка', 'dog': 'пес','Bird': 'птах', 'mouse': 'миша'}

for i in dic:

dic [i] = dic [i] + '\_ua'

print (dic)

Вивід програми:

{'Bird': 'птах\_ua ', 'cat': 'кішка\_ua ', 'mouse': 'миша\_ua ', 'dog': 'пес\_ua '}

**Дострокове завершення циклу**

Відзначимо два простих оператора *break* і *continue*, за допомогою яких можна управляти ходом виконання циклу. Оператор *break* перериває виконання циклу, управління передається операторам, наступним за оператором циклу. Оператор *continue* перериває виконання чергового кроку циклу і повертає управління в початок циклу, починаючи наступний крок.

for n in range (10):

if n% 2 == 0:

сontinue

if n == 7:

break

print (n)

Дана програма буде друкувати тільки непарні числа через спрацювання continue. Цикл припинить виконуватися, коли n стане дорівнює 7. У результаті висновок програми такий:

1

3

5