Прикладне програмування

ТЕМА 5. ЗАГАЛЬНА БУДОВА ГЛОБАЛЬНИХ МЕРЕЖ

Глобальні мережі

Сучасні інформаційні технології неможливо уявити без використання глобальних комп'ютерних мереж.

Мережі настільки увійшли у наше повсякденне життя, що зараз будь-яка серйозна програма обов'язково здійснює взаємодію у мережі.

Ми розглянемо загальні принципи побудови глобальних мереж та засоби побудови програм, які обмінюються даними у мережі.

Коли кажуть про глобальні мережі, перш за все, мають на увазі Інтернет, який виник як мережа для військових потреб, але зараз використовується у різноманітних застосуваннях: від отримання інформації та спілкування до віддаленого керування пристроями та навіть побутовою технікою.

Мережні протоколи

Взаємодія програм у мережі здійснюється за допомогою протоколів.

Протокол – це набір правил взаємодії між різними сутностями.

Протоколи були відомі і до існування глобальних мереж.

Зокрема, всі знають про існування дипломатичного протоколу.

Правила поведінки у світі також можна назвати протоколом.

Наприклад, правила поведінки за столом та використання ножа та виделки.

Мережні протоколи.2

Мережний протокол – це домовленість щодо структурування даних, які надсилаються між двома або більше сторонами у мережі.

Мережні протоколи утворюють так званий стек протоколів.

У цьому стеку на перших рівнях діють низькорівневі протоколи, починаючи з фізичного зв’язку між комп’ютерами, а на останніх рівнях, - протоколи що забезпечують взаємодію окремих застосувань.

Класичною є семирівнева модель протоколів OSI (open systems interconnection) та стек протоколів OSI.

Мережні протоколи.3

У моделі OSI виділяють 7 рівнів:

Рівень	Назва
7	Прикладний (Application)
6	Представлення (Presentation)
5	Сеансовий (Session)
4	Транспортний (Transport)
3	Мережний (Network)
2	Канальний (Data link)
1	Фізичний (Physical)

Кожний вищий рівень базується на нижчому рівні та визначає цілий ряд окремих протоколів.

Протокол TCP/IP

Надзвичайно важливу ролі у побудові сучасних глобальних мереж мають два протоколи:

- протокол мережного рівня IP (Internet Protocol)
- протокол транспортного рівня TCP (Transmission Control Protocol).

IP з’єднує між собою різні мережі, засновані, можливо, на різних фізичних принципах. IP об’єднує дані у пакети та відправляє по мережі.

TCP забезпечує надійність та правильний порядок передачі/приймання пакетів даних.

Ці протоколи постійно використовують разом, тому їх назви часто об’єднують у TCP/IP.

TCP/IP базуються на протоколах нижчого рівня (канального, фізичного).

Над ними також побудовані протоколи прикладного рівня: HTTP, FTP, BitTorrent тощо.

Адреси та порти

Кожний пристрій, під’єднаний до мережі TCP/IP, має власну адресу.

Адреса задається числом або ім’ям пристрою у мережі.

Довжина числа, яке визначає адресу, залежить від версії протоколу IP.

У більш звичній версії IPv4 адреса складається з 4 байтів (32 біт).

У написанні адреси значення окремих байтів записуються у десятковій системі числення та розділяються крапками. Наприклад:

192.168.0.3

У новішій версії протоколу IPv6 адреса складається з 16 байтів (128 біт).

У написанні адреси значення окремих байтів записуються у системі числення за основою 16, а двохбайтні частини адреси розділяються двокрапками. Наприклад:

2001:0DB8:0000:ABCD:0000:0000:0000:1234

Адреси та порти.2

Необхідність переходу від IPv4 до IPv6 викликана тим, що з появою великої кількості пристроїв, які приєднано до мережі, 32 біт просто не вистачає на всі адреси.

Надалі ми будемо розглядати роботу у IPv4.

Існує одна виділена адреса, яка позначає поточний комп’ютер: 127.0.0.1.

У деяких мовах програмування до поточного комп’ютера можна також звернутись з використанням символічного імені localhost.

Адреси та порти.3

Однак адреси комп’ютера недостатньо, щоб зв’язатись з ним у мережі.

Необхідно знати ще номер порту.

Номер порту – це число від 0 до 65535.

З портом може бути пов’язана програма, що виконується на даному комп’ютері та здатна з’єднуватись з іншими комп’ютерами через цей порт.

Перші 1024 порти зарезервовані для стандартних програм, інші можуть використовуватись власними або зовнішніми нестандартними програмами.

Наприклад, порт 80 використовується Інтернет браузерами.

Сокети

Для розуміння порядку обміну даними у мережі важливу роль грає поняття сокету.

Сокет – це програмний інтерфейс обміну даними між двом програмами на різних або одному комп’ютері.

Слово сокет – це калька з англійського socket (гніздо або розетка).

Щоб створити сокет, програма задає пару (<адреса>, <порт>) та готується до обміну даними через сокет.

Сервери та клієнти

Коли дві програми взаємодіють у мережі, серед них виділяють сервер та клієнта.

Сервер – це програма, яка постійно завантажена та готова приймати та віддавати дані клієнтам.

Сервер надає певний сервіс – послугу, яку отримують програми-клієнти.

Клієнт – це програма, яка звертається до сервера за отриманням згаданої послуги.

Сервер у мережі утворює сокет, задаючи пару (<адреса>, <порт>), та «слухає» (listens) порт, поки не надійде запит від клієнта.

Один сервер може обслуговувати багато клієнтів одночасно та/або послідовно.

Реалізація програмування у мережі у Python

Стандартна бібліотека Python містить багато модулів, які підтримують програмування у мережі.

Ми розглянемо тільки деякі з них.

Зазвичай, ім’я модуля, що реалізує певний протокол, співпадає з ім’ям протоколу (наприклад, http), або починається з імені протоколу, за яким йде ‘lib’, наприклад, ftplib.

Модуль socket

Модуль socket підтримує низькорівневе мережне програмування серверів та клієнтів.

Цей модуль містить, зокрема, клас socket, об’єктами якого є сокети.

Щоб створити новий сокет, треба викликати

```
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
```

- де socket.AF_INET означає так зване адресне сімейство інтернет (на противагу сокетам Unix), а socket.SOCK_STREAM – використання протоколу TCP/IP.

Ця функція повертає об’єкт класу socket, який потім використовується для з’єднання.

Модуль socket.2

Деякі методи класу socket зібрані у таблиці нижче

Метод	Опис
s.bind((host, port))	Зв'язати сервер з комп'ютером host та портом port
s.connect((host, port))	З'єднатися (клієнту) з сервером
s.listen(n)	Очікувати серверу на з'єднання з клієнтом (максимально у черзі може бути до n клієнтів)
conn, addr = s.accept()	Отримати серверу параметри з'єднання з клієнтом (conn – новий об’єкт класу socket для обміну даними, addr – адреса клієнта)
data = conn.recv(size)	Отримати від об’єкта conn дані data (рядок байтів) розміром не більше size
conn.sendall(data)	Відправити об’єкту conn усі дані з data (рядок байтів)
conn.close()	Закрити з'єднання conn

Модуль socket.3

Стандартна послідовність дій при створенні сервера за допомогою модуля socket:

```
s = socket(AF_INET, SOCK_STREAM)# створити сокет сервера
s.bind((host, port))       # Зв'язати з комп'ютером host та портом port
s.listen(n)                # Очікувати  серверу на з'єднання з клієнтом
while True:
    # нескінченний цикл сервера (обробка підключень клієнтів)
    conn, addr = s.accept()     # Отримати параметри з'єднання
    ...
    while True:
        # цикл комунікації з 1 клієнтом
        ...
        conn.recv()/conn.send() # отримати/передати дані
    conn.close()                # закрити клієнтський сокет
s.close()                       # закрити серверний сокет
```

Модуль socket.4

Стандартна послідовність дій при створенні клієнта за допомогою модуля socket виглядає так:

```
s = socket(AF_INET, SOCK_STREAM)# створити сокет
s.connect((HOST, PORT))    # з'єднатися з сервером.

while True:

    # цикл комунікації з сервером

    ...

    s.recv()/s.send()   # отримати/передати дані

s.close()              # закрити клієнтський сокет
```

Рядки байтів

Дані, якими обмінюються комп’ютери у мережі, з точки зору Python є рядками байтів.

Костанта типу «рядок байтів» позначається так само, як константа типу рядок, але безпосередньо перед нею ставиться літера b.

Наприклад, b’metro’.

Рядок байтів можна отримати зі звичайного рядка або списку цілих чисел (<256) за допомогою функції bytes. Наприклад:

```
bytes(string, encoding='utf-8')
```

```
bytes([1,2,38])
```

Тут encoding – це стандарт кодування звичайного рядка.

Отримати звичайний рядок за рядком байтів можна за допомогою функції str, наприклад:

```
str(bt, encoding='utf-8')
```

Запуск серверів та клієнтів

Запуск серверів та клієнтів може здійснюватися на одному або різних комп’ютерах.

Але навіть якщо сервер та клієнт запускаються на одному комп’ютері, вони повинні запускатися різними копіями інтерпретатора Python.

В якості адреси комп’ютера (host) для сервера часто вказують порожній рядок ‘’, що означає «будь-яка адреса».

Тобто, сервер може створити сокет на тому комп’ютері, де він буде запущений, не зважаючи на ім’я або адресу комп’ютера у мережі.

В якості порту сервера вказують довільне число, більше або рівне 1024.

Запуск серверів та клієнтів.2

Якщо клієнт запускається на тому ж комп’ютері, що й сервер, - в якості адреси (host) вказують ‘localhost’ або ‘127.0.0.1’.

Якщо сервер запущено на віддаленому комп’ютері, клієнт повинен вказати адресу цього віддаленого комп’ютера.

При цьому, клієнт повинен мати мережний доступ до цього віддаленого комп’ютера (мати на ньому аккаунт).

Приклад: Сервер та клієнт паліндромів. Версія 1

Скласти програми, які реалізують сервер, що перевіряє, чи є рядок паліндромом, а також клієнта, який вводить рядки, та перевіряє, чи є вони паліндромами, використовуючи сервер.

Сервер використовує модуль socket.

Він обробляє з’єднання від одного клієнта, який може передати на вхід декілька рядків.

Для кожного рядка сервер здійснює перевірку, чи є цей рядок паліндромом, використовуючи функцію test_palindrome.

Після перевірки сервер відправляє клієнту рядок байтів, що відповідає бульовому значенню True (рядок довжини 1) або False (рядок довжини 0).

До рядка-відповіді додається ‘\n’.

Клієнт встановлює з’єднання з сервером, вводить рядки, поки не буде введено порожній рядок, що означає завершення з’єднання.

Для кожного рядка показує відповідь сервера: чи є цей рядок паліндромом.

Використання файлоподібних об’єктів для обміну даними

Для обміну даними у мережі Python дозволяє використовувати у сокетах файлоподібні об’єкти.

У цьому випадку обмін даним спрощується та нагадує читання та запис у файл.

Для створення файлоподібних об’єктів (далі – просто файлів) застосовують метод makefile об’єкту класу socket.

Як правило, створюють два файли: один для читання та один для запису.

Параметри makefile такі ж, як і у стандартної функції відкриття файлу open. Наприклад, після

```
inp = conn.makefile('rb', 0)
```

```
out = conn.makefile('wb', 0)
```

- створюються файли inp та out для читання та запису даних відповідно.

Другий параметр makefile, який дорівнює 0, означає відсутність буферизації файлу.

Використання файлоподібних об’єктів для обміну даними.2

Після утворення файлів для того, щоб передати або прийняти дані через сокет, можна просто використати стандартні методи для файлів: readline, write тощо.

Однак при використанні файлів треба не забувати ставити в кінці кожного рядка даних перехід на наступний рядок (b‘\n’).

Закрити обидва файлоподібних об’єкти можна методом shutdown:

```
conn.shutdown(2)
```

Приклад: Сервер та клієнт паліндромів. Версія 2

Скласти програми, які реалізують сервер, що перевіряє, чи є рядок паліндромом, а також клієнта, який вводить рядки, та перевіряє, чи є вони паліндромами, використовуючи сервер.

У версії 2 сервер та клієнт використовують файли для обміну даними.

У модулі, що містить сервер, описано клас PalindromeServer.

Конструктор цього класу __init__ створює сокет, зв’язує його з адресою та портом та чекає на з’єднання клієнтів.

Обробляє з’єднання клієнтів метод run.

На відміну від версії 1, сервер версії 2 може обробляти з’єднання багатьох клієнтів, він не завершує роботу після від’єднання першого клієнта.

Читання та передача даних здійснюється за допомогою readline та write відповідно.

Клієнт версії 2 майже не відрізняється від клієнта версії 1, окрім того, що для обміну даними, як і сервер, використовує файли.

Модуль socketserver

У модулі socketserver зібрано засоби побудови серверів більш високого рівня, ніж у модулі socket.

Клас TCPServer реалізує сервер за заданою адресою та портом.

Для обробки запитів від клієнтів на обмін даними необхідно описати власний клас обробки запитів на ім’я, наприклад, RequestHandler.

Цей клас повинен бути нащадком класу BaseRequestHandler або StreamRequestHandler, які описані у socketserver.

У ньому треба перевизначиати метод handle().

Метод handle викликається тоді, коли надійшов запит від клієнта, тобто, коли клієнт з’єднався з сервером.

Ще один метод, який можна перевизначити, - це метод finish(), який викликається після завершення обробки запиту методом handle.

Модуль socketserver.2

StreamRequestHandler відрізняється від BaseRequestHandler тим, що обмінюється даними за допомогою файлоподібних об’єктів.

Для цього використовуються поля rfile та wfile, відповідно, для читання (приймання) та запису (передачі) даних.

Для старту сервера треба викликати метод serve_forever():

```
socketserver.TCPServer((HOST, PORT),
        RequestHandler).serve_forever()
```

Приклад: Сервер паліндромів. Версія 3

Скласти програми, які реалізують сервер, що перевіряє, чи є рядок паліндромом, а також клієнта, який вводить рядки, та перевіряє, чи є вони паліндромами, використовуючи сервер.

У версії 3 сервер написаний на базі класу TCPServer з модуля socketserver.

Клас RequestHandler обробляє запити клієнтів, перевіряє, чи є рядок паліндромом та надсилає відповідь.

Кожний клієнт може надсилати декілька рядків.

Кількість клієнтів не обмежена, але вони обслуговуються послідовно.

Для звернення до сервера версії 3 використовується клієнт паліндромів версії 2.

Багатопроцесні та багатопотокові сервери

До цього часу ми розглядали тільки такі сервери, які обслуговують одночасно не більше одного клієнта.

Звичайно, у реальному житті потрібні сервери, що обслуговують багато клієнтів паралельно.

Такий режим роботи сервера може бути досягнений за допомогою багатопроцесності або багатопотоковості.

Багатопроцесні та багатопотокові сервери.2

Модуль socketserver містить реалізацію багатопроцесних та багатопотокових серверів.

Ця реалізація базується на класах домішках: ForkingMixIn та ThreadingMixIn, які надають серверам поведінку, що необхідна для паралельного виконання.

Відповідні класи серверів називаються ForkingTCPServer та ThreadingTCPServer.

Їх використання не відрізняється від використання простого класу TCPServer.

Звичайно, треба мати на увазі, що запити від кожного клієнта будуть оброблятись у власному процесі (потоці).

Приклад: гра у відгадування слів у мережі

У темі «рекурсивні структури даних» ми розглядали в якості прикладу гру у відгадування слів.

Умови цієї гри такі.

По колу розташовані гравці (відгадувачі), яким презентують слово для відгадування.

Всі літери цього слова спочатку закриті (замінені зірочками, ‘*’).

Гравці вступають у гру по порядку.

Кожен гравець може назвати літеру або слово.

Приклад: гра у відгадування слів у мережі.2

Якщо гравець називає літеру, а цієї літери, у слові немає, - хід переходить до наступного гравця. Якщо ж така літера у слові є, то всі входження цієї літери у слові відкриваються, а гравцю нараховуються стільки балів, скільки є входжень названої літери у слово. Якщо всі літери слова відкриті, - гравець стає переможцем.

Якщо гравець називає слово і це слово не дорівнює заданому, то всі бали гравця анулюються, а хід переходить до наступного гравця. Якщо ж слово названо правильно, - гравець отримує стільки балів, скільки є у слові невідгаданих літер, та стає переможцем.

Переможець отримує премію: стільки балів, скільки літер було у слові.

Необхідно написати сервер та клієнта для реалізації гри у відгадування слів у мережі.

Гра у відгадування слів у мережі. Реалізація

Для реалізації гри нам знадобиться описати власний протокол прикладного рівня.

Цей протокол буде визначати послідовність обміну даними між сервером та клієнтами.

Загальний процес гри виглядає так.

Сервер очікує, поки до нього підключиться достатня кількість клієнтів для початку гри.

Кожен клієнт при підключенні повідомляє своє ім’я.

Сервер повідомляє усіх клієнтів щодо того, хто підключився до гри.

Коли всі клієнти приєдналися, сервер повідомляє про початок гри зображує слово, заповнене зірочками ‘*’, та надає хід першому клієнту.

Отримавши хід, клієнт вводить з клавіатури літеру або слово та передає серверу.

Сервер показує всім клієнтам, яку літеру або слово було введено, аналізує дані та надає хід тому ж клієнту, або передає хід наступному.

Гра у відгадування слів у мережі. Реалізація.2

Окрім того, після кожного ходу сервер зображує поточний стан слова для відгадування.

Коли гру завершено, сервер повідомляє клієнтів про результати гри та набрані бали а також відправляє команду про завершення гри.

Клієнти від’єднуються від сервера, а сервер знову очікує на підключення клієнтів для нової гри.

В якості сервера виберемо багатопотоковий сервер ThreadingTCPServer, оскільки нам буде потрібно обмінюватись даними між клієнтами.

Гра у відгадування слів у мережі. Протокол

Протокол гри у відгадування слів у мережі складається з команд сервера, команд клієнта та повідомлень сервера клієнтам. Команди задаються словами, перший символ команди – коса риска ‘/’.

Якщо команда має параметри, вони відділяються від команди пропуском.

Команди сервера:

- /turn – надати хід
- /quit – завершити гру

Команди сервера не мають параметрів.

Гра у відгадування слів у мережі. Протокол.2

Команди клієнта:

- /letter - літера
- /word - слово

Команди клієнта мають один параметр: власне введена літера або слово.

Повідомлення сервера – це будь-який текст, що не є командою.

Клієнт повинен просто вивести цей текст.

Гра у відгадування слів у мережі. Сервер

У модулі, що реалізує сервер гри у відгадування слів, описано класи WordGuessServer, RequestHandler, NetGuesser та ClientError.

WordGuessServer – це клас, який наслідує від класу ThreadingTCPServer.

Використовується для ініціалізації сервера а також для збереження та модифікації спільних даних різних потоків.

Клас має поля:

- self.glist - кільцевий список гравців (відгадувачів) типу NetGuesser
- self.num_guessers - поточна кількість гравців
- self.num_to_start - кількість гравців для початку гри
- self.game_on - чи йде гра
- self.word - слово для відгадування
- self.guessed - слово, заповнене '*'

Гра у відгадування слів у мережі. Сервер.2

RequestHandler – клас обробки запитів клієнтів, наслідує від StreamRequestHandler. Виконує основну роботу та буде описаний далі.

NetGuesser – клас, що описує відгадувача. Наслідує від класу Guesser, описаного у темі «Рекурсивні структури даних». Додатково має поле self.wfile, яке зберігає файл для передачі даних відповідному клієнту. Також у класі реалізовано метод __str__ для перетворення об’єкту у рядок.

ClientError – клас виключення, що повідомляє про помилкові дані від клієнта.

Гра у відгадування слів у мережі. Сервер. Клас RequestHandler

Клас RequestHandler містить поля

- self.name – ім’я гравця. Значення цього поля встановлюється у методі addGuesser
- self.server – об’єкт сервера. Успадковується від класу StreamRequestHandler, надає доступ до спільних даних.
- self.wfile - зберігає файл для передачі даних поточному клієнту. Успадковується від класу StreamRequestHandler.

Клас містить методи handle, finish, processInput, addGuesser, startGame, endgame, nextTurn, letterCommand, wordCommand, _processResults, broadcast, privateMessage, _readline, _ensureNewline, _parseCommand.

Гра у відгадування слів у мережі. Сервер. Клас RequestHandler.2

Метод handle обробляє запити клієнта, що підключився до сервера.

- Спочатку він розраховує отримати ім’я клієнта, додає це ім’я до списку відгадувачів, перевіряє, чи треба почати гру та починає її. Потім обробляє дані від клієнта.

Метод finish видаляє клієнта, що завершив роботу з списку відгадувачів та, якщо список порожній, - завершує гру.

Метод processInput викликається з методу handle. Читає рядок тексту від клієнта та обробляє отриману команду. Клієнт може надсилати серверу тільки команди.

Метод addGuesser викликається з методу handle. Додає гравця у список гравців та збільшує кількість гравців, що приєднались до гри. Також повідомляє всім про приєднання нового гравця.

Гра у відгадування слів у мережі. Сервер. Клас RequestHandler.3

Метод startGame починає гру: вибирає та запам’ятовує слово для відгадування, повідомляє усім про початок гри а також надсилає слово, заповнене зірочками.

Метод endgame будує та надсилає усім клієнтам рядок з результатами гри а також команду завершення гри.

Метод nextTurn надає хід гравцю, який є поточним у списку гравців (надсилає команду \turn). Повідомляє усім про те, чий зараз хід.

Метод letterCommand обробляє команду \letter від клієнта. Повідомляє усім, яку літеру названо. Аналізує, чи входить літера у слово та скільки раз. Змінює зірочки у слові для відгадування на літеру у позиціях входження. Нараховує бали та перевіряє, чи відгадано слово.

Метод wordCommand обробляє команду \word від клієнта. Повідомляє усім, яке слово названо. Аналізує, чи дорівнює назване слово слову для відгадування. Якщо так, то змінює усі зірочки у слові для відгадування. Нараховує бали.

Гра у відгадування слів у мережі. Сервер. Клас RequestHandler.4

Метод _processResults обробляє результати команд \letter та \word. Якщо бали зароблено, повідомляє кількість зароблених балів. Якщо слово повністю відгадано, завершує гру викликом endGame. Якщо літера або слово неправильні, готує передачу ходу наступному гравцю. Надає наступний хід.

Метод broadcast надсилає повідомлення всім клієнтам, які приєднані до сервера.

Метод privateMessage надсилає повідомлення тільки поточному клієнту.

Метод _readline читає рядок даних від клієнта.

Метод _ensureNewline запевняє, що рядок завершується символом '\n'

Метод _parseCommand намагається розібрати рядок як команду. Окремо отримує аргументи команди, якщо вони є. Повертає метод, що відповідає цій команді (з ім’ям <команда>Command) та аргументи.

Гра у відгадування слів у мережі. Клієнт

У модулі, що реалізує клієнт гри у відгадування слів, описано класи WordGuessClient та ServerError.

WordGuessClient – це клас, який виконує основну роботу та буде описаний далі.

ServerError – клас виключення, що повідомляє про помилкові дані від сервера.

Гра у відгадування слів у мережі. Клієнт. Клас WordGuessClient

Клас WordGuessClient містить поля

- self.socket – об’єкт сокет клієнта
- self.input – файл для отримання даних від сервера
- self.output – файл для передачі даних серверу

Клас містить методи __init__, run, sendMessage, processInput, turnCommand, quitCommand, _parseCommand, _readline.

Гра у відгадування слів у мережі. Клієнт. Клас WordGuessClient.2

Конструктор __init__ здійснює з'єднання з сервером та передає йому ім'я гравця. Також створює файли для отримання та передачі даних.

Метод run у циклі обробляє отримані від сервера дані.

Метод sendMessage надсилає один рядок серверу.

Метод turnCommand обробляє команду /turn. Запитує введення літери або слова та передає на сервер відповідну команду з аргументом.

Метод quitCommand завершує роботу клієнта, закриває файли та з'єднання.

Методи processInput, _parseCommand та _readline аналогічні або ідентичні відповідним методам сервера.

Резюме

Ми розглянули:

1. Мережні протоколи
2. Протокол TCP/IP
3. Адреси та порти.
4. Сокети. Сервери та клієнти
5. Реалізація програмування у мережі у Python. Модуль socket
6. Рядки байтів
7. Запуск серверів та клієнтів
8. Використання файлоподібних об’єктів для обміну даними
9. Модуль socketserver
10. Багатопроцесні та багатопотокові сервери

Де прочитати

1. Обвінцев О.В. Об’єктно-орієнтоване програмування. Курс на основі Python. Матеріали лекцій. – К., Основа, 2017
2. Peter Norton, Alex Samuel, David Aitel та інші - Beginning Python
3. Wesley J. Chun - Core Python Programming - 2001
4. Magnus Lie Hetland - Beginning Python from Novice to Professional, 2nd ed – 2008
5. Mark Lutz - Programming Python. 4th Edition - 2011
6. Прохоренок Н.А. - Python 3 и PyQt. Разработка приложений – 2012
7. Mark Pilgrim - Dive into Python, Version 5.4 - 2004
8. Jim Knowlton - Python Create Modify Reuse – 2008
9. Noah Gift, Jeremy M. Jones Python for Unix and Linux System Administration