МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

І. Лазарович, М. Дутчак

МЕТОДИЧНІ ВКАЗІВКИ
ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «ПРОГРАМУВАННЯ МОВОЮ RUBY»

для студентів спеціальності
"Інженерія програмного забезпечення"

Івано-Франківськ
2023

УДК 004.415.2
ББК 32.973.4
Л17

Рекомендовано до друку вченою радою факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника, (протокол № 7 від 22.08.2023 р.)

Рецензенти:

Козленко М.І. – кандидат технічних наук, доцент кафедри кафедри інформаційних технологій Прикарпатського національного університету імені Василя Стефаника;

Ткачук В.М. – кандидат фізико-математичних наук, доцент кафедри інформаційних технологій факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника

І. Лазарович, М. Дутчак**.**

Л17 Методичні вказівки до виконання лабораторних робіт з курсу «Програмування мовою Ruby» для студентів спеціальності "Інженерія програмного забезпечення". Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2023. – 80 с.

Методичні вказівки містять теоретичні відомості до виконання робіт, варіанти завдань для виконання, вимоги до оформлення звітів та контрольні питання для самоперевірки. Теоретичні відомості містять короткий опис операторів, мовних конструкцій, методів і класів мови Ruby, що полегшує виконання роботи і засвоєння матеріалу. Тематика робіт відповідає робочій програмі з дисципліни «Програмування мовою Ruby» для студентів спеціальності «Інженерія програмного забезпечення». Методичні вказівки передбачають, що студенти мають мінімальні навики роботи із сучасними засобами Web-розробки, зокрема базові поняття мов HTML, CSS, JavaScript.

УДК 004.415.2
ББК 32.973.4

ЗМІСТ

ПЕРЕДМОВА

Методичні вказівки до лабораторних робіт з курсу "Програмування мовою Ruby" спрямовані на закріплення теоретичних знань і розвиток практичних навичок у програмуванні мовою Ruby та використанні її екосистеми.

Ruby — це сучасна, зручна та потужна мова програмування, яка відзначається простотою синтаксису та широкими можливостями для розробки різноманітних програмних продуктів, зокрема веб-додатків на основі фреймворку Ruby on Rails.

Методичні вказівки охоплюють як базові концепції програмування на Ruby, так і спеціалізовані теми, пов’язані з об’єктно-орієнтованим програмуванням, роботою з масивами, рядками, асоціативними структурами даних, а також з використанням фреймворку Ruby on Rails для створення веб-застосунків. Вказівки структуровані за принципом поступового ускладнення матеріалу, що дозволяє студентам послідовно нарощувати знання та досвід.

Кожна лабораторна робота містить:

- теоретичні відомості, що допоможуть студентам зрозуміти основні концепції та підготуватися до виконання практичних завдань;
- завдання до виконання, спрямовані на практичне застосування отриманих знань;
- рекомендовані інформаційні ресурси, які стануть у нагоді для поглибленого вивчення матеріалу та самостійного опрацювання.

Виконання запропонованих лабораторних робіт дозволить студентам оволодіти основами програмування мовою Ruby, розробляти ефективні програмні рішення, а також отримати базові навички створення сучасних веб-додатків.

Наведені методичні вказівки стануть корисними не лише для виконання лабораторних робіт, але й для самостійної роботи та підготовки до контрольних заходів.

ЛАБОРАТОРНА РОБОТА № 1

Тема: Встановлення середовища розробки і робота з масивами в мові Ruby

Мета роботи: Ознайомитись з особливостями побудови і роботи з масивами в мові Ruby.

Основні теоретичні відомості

Одним з базових типів змінних є масив. Формально масив – це ряд змінних, доступ до яких здійснюється за індексом. Кажучи простіше, масив – група змінних, що пронумеровані починаючи з нуля, що зберігаються в єдиній змінній. Масиви використовують при роботі з однотипними даними. Наприклад, дуже зручно зберігати в масиві інформацію про температуру повітря за останній місяць. Існують багатомірні масиви, в яких елементи масиву самі є масивами.

Для того щоб змінити масив в мові Ruby до нього необхідно застосувати певний метод. Метод – це те, що змінює змінну згідно до заданого правила. Для різних типів змінних можуть використовуватись різні методи.

Існує декілька способів створення масиву:

- звичайний

```
arr= [ 5.3,  4.2,  2.0 , -0.8, 1.5 ]
```

- ручний (зверніть увагу, що при додаванні елементу в кінець вказувати його ключ не обов’язково)

```
arr = Array.new
arr[0]=1
arr[1]=2
arr[]=3
…
```

- через діапазон

```
arr = (1..6).to_a
```

Щоб застосувати метод method до змінної var достатньо написати:

```
var.method
```

Зверніть увагу звичайний метод не змінює самої змінної. Якщо це необхідно слід присвоїти отримане значення іншій змінній, або тій же:

```
var = var.method
```

Нижче наведено таблицю основних методів роботи з масивами. Результат наведено виходячи з того, що для прикладу взято масив [6,3,5,1,2,4].

Таблиця 1.1 – Основні методи роботи з масивами

Метод	Опис	Результат
`arr.size`	кількість елементів	6
`arr.min`	мінімальний елемент	1
`arr.max`	максимальний елемент	6
`arr.sort`	сортування	[1,2,3,4,5,6]
`arr.reverse`	перевернути	[4,2,1,5,3,6]
`arr.sort.reverse`	відсортувати і перевернути	[6,5,4,3,2,1]
`arr.include?(6)`	чи є в масиві значення 6	true
`arr.empty?`	чи пустий масив	false
`arr.any?`	чи існує хоча б один елемент	true
`arr.delete(6)`	видалити елемент 6	[3,5,1,2,4]
`arr.delete_at(1)`	видалити елемент з ключем 1	[6,5,1,2,4]

Існує ряд методів для обробки всіх елементів масиву певним способом, їх у мові Ruby називають ітераторами. Це, зокрема, наступні методи:

Метод Inject

Загальний вигляд:

```
res = arr . inject ( start ) { | result , element | expression }
```

start – начальне значення змінної result;

result – змінна, в яку записується результат;

element – поточний елемент масиву;

expression – вираз для змінних result і element.

Метод Find

Загальний вигляд:

arr_res = arr.find_all { | element | condition }

element – поточний (той, що розглядається) елемент масиву (змінна змінюється з кожною ітерацією);

condition – логічний вираз або декілька виразів, пов'язаних логічними операціями (кон'юнкція, диз'юнкція, заперечення);

Метод вибирає з масиву arr всі елементи, що задовольняють умові condition і зберігає їх в змінній arr_res.

Метод Map

Загальний вигляд:

```
arr_res = arr.map { | element | expression }
```

element – поточний (той, що розглядається) елемент масиву (змінна змінюється з кожною ітерацією);

expression – вираз, що вказує на те, як саме необхідно змінити елементи масиву.

Метод проходить весь масив arr, змінюючи його елементи згідно до умови expression, результат записує в arr_res.

Завдання до роботи

1.2.1 Встановити середовище розробки (Ruby + RubyOnRails) [1,2]

1.2.2 Ознайомитися з основними теоретичними відомостями за темою роботи.

1.2.3 Використовуючи наведені в роботі методи для роботи з масивами, та зокрема ітератори виконати наступні завдання:

1.2.3.1 Напишіть програму, що за допомогою inject обчислює добуток елементів довільно взятого масиву.

1.2.3.2 Напишіть програму, що обчислює середнє арифметичне та середнє геометричне елементів масиву.

1.2.3.3 Дано масив температури за останні 10 днів в градусах Цельсія. Вивести температуру на кожен день по Фаренгейту і Кельвіну.

1.2.3.4 Виконайте індивідуальне завдання з таблиці 1.2 згідно свого варіанту.

1.2.4 Оформіть звіт з роботи.

Таблиця 1.2 – Індивідуальні завдання

№ вар.	Умова задачі
1	Знайти три найменші унікальні елементи масиву, відсортовані за зростанням.
2	Знайти суму квадратів всіх парних чисел у масиві.
3	Знайти середнє значення всіх непарних чисел, які більше 10.
4	Перевірити, чи є масив порожнім після видалення всіх нульових елементів.
5	Знайти всі елементи, які діляться на 3, відсортувати за спаданням та перевірити їхню наявність у масиві.
6	Знайти всі індекси елементів, що є більше середнього значення масиву.
7	Обчислити добуток всіх чисел, які не є нулями, та відсортувати їх.
8	Знайти мінімальне значення у масиві після видалення всіх чисел, що менші за 5.
9	Знайти всі парні числа, відсортувати їх за спаданням і обчислити суму.
10	Перевірити, чи містить масив хоча б одне парне число більше 10.
11	Знайти найбільше число, якщо масив містить більше 5 елементів, інакше повернути -1.
12	Видалити всі повторювані елементи, відсортувати і перевірити наявність 7.
13	Знайти всі непарні числа, які менші за середнє значення масиву.
14	Створити новий масив, що містить квадрати всіх чисел, відсортовані за спаданням.
15	Перевірити, чи всі елементи масиву більші за середнє значення.
16	Знайти всі індекси елементів, які є меншими за мінімальне значення масиву.
17	Обчислити суму елементів, що діляться на 4, відсортованих за зростанням.
18	Знайти всі унікальні елементи масиву, обчислити їхню суму і перевірити наявність 10.
19	Знайти найбільше число серед парних чисел і перевірити, чи є воно в масиві.
20	Видалити всі елементи з індексами 1, 3, 5 і перевірити, чи масив порожній.
21	Знайти всі парні числа, які менші за середнє значення масиву.
22	Обчислити добуток непарних чисел, відсортованих за спаданням.
23	Перевірити, чи масив містить хоча б одне від'ємне число і видалити його.
24	Знайти всі числа, що діляться на 5, підняти їх до квадрата і відсортувати.
25	Обчислити суму всіх парних чисел, що є більше середнього значення масиву.
26	Перевірити, чи масив містить лише унікальні елементи і відсортувати їх.
27	Знайти всі парні числа, які більші за мінімальне значення масиву.
28	Перевірити, чи всі елементи масиву є парними і видалити ті, що не парні.
29	Обчислити добуток всіх чисел, які діляться на 3, і перевірити наявність нуля.
30	Знайти всі унікальні числа, підняті до куба, відсортовані за спаданням.

Вимоги до звіту

Тема та мета роботи.

Скріншот, що демонструє версію Ruby і RubyOnRails.

Основна частина роботи, яку навести у наступному форматі:

- Сформульоване завдання 1;
- Лістинг виконання завдання 1 (чорним по білому);
- Результати виконання програми за завданням 1.
- Сформульоване завдання 2;

і т.д.

Висновки, що відображують результати виконання роботи та їх критичний аналіз.

Контрольні питання

1. Які основні етапи встановлення середовища розробки Ruby та Ruby on Rails?
2. Як функція inject працює в Ruby, і які типи операцій можна з її допомогою виконувати, призначення і обов'язковість аргументу?
3. Яким чином у Ruby можна перевірити, чи є число парним і додатнім?
4. Яким чином можна створювати масиви у Ruby?
5. Яка різниця між методами inject і each у Ruby?
6. Чим відрізняються методи select та find_all у Ruby, і в яких випадках кожен з них краще використовувати?
7. Як метод map відрізняється від методу each і коли варто використовувати один замість іншого?
8. Які відмінності між методами sort, sort_by та sort!, і як кожен з них змінює масив?
9. Як методи include? та any? можуть використовуватися для пошуку елементів у масиві, і які відмінності між ними?
10. Як можна використовувати методи map, select та inject разом для складної обробки масивів у Ruby?

Рекомендовані ресурси

1. Уроки Ruby on Rails (налаштування середовища та перший запуск проекту) URL: https://youtu.be/lBG5hcQlfAw?t=125
2. Installing Rubies URL: https://rvm.io/rubies/installing

ЛАБОРАТОРНА РОБОТА № 2

Тема: Багатомірні масиви в Ruby

Мета роботи: Ознайомитись з підходами до обробки багатомірних масивів у мові Ruby та вдосконалення навиків роботи із ітераторами.

Основні теоретичні відомості

Двовимірний у Ruby масив представляється як масив масивів:

```
matrix = [[1, 2, 3],          # перший рядок
          [4, 5, 6],          # другий рядок
          [7, 8, 9]]          # третій рядок
```

Доступ до елементів двовимірного масиву здійснюється за допомогою індексів:

```
element = matrix[1][2]    # результат буде 6
```

Можна також створити двовимірний масив динамічно, використовуючи ітерації:

```
rows = 3
cols = 3
matrix = Array.new(rows) { Array.new(cols) }
matrix.each_with_index do |row, i|
  row.each_with_index do |_, j|
    matrix[i][j] = i * cols + j + 1
  end
end
```

У цьому прикладі ми створили порожній двовимірний масив розміром 3x3 та заповнили його значеннями за допомогою вкладених циклів.

Основні методи для роботи з багатовимірними масивами:

Метод **transpose** міняє місцями рядки і стовпці двовимірного масиву. Це корисно для роботи зі стовпцями як з рядками:

```
transposed = matrix.transpose
```

Метод **flatten** розгортає багатовимірний масив в одновимірний. Це дозволяє застосовувати до нього методи, які працюють лише з одновимірними масивами:

```
flat_array = matrix.flatten
```

Метод **each_with_index** ітерує по елементах масиву разом з їх індексами, що дозволяє одночасно працювати з елементами і їх індексами:

```
matrix.each_with_index do |row, i|
  row.each_with_index do |element, j|
    puts "Element at (#{i}, #{j}) is #{element}"
  end
end
```

Метод **select** повертає новий масив з елементами, які відповідають умові блоку. Це зручно для фільтрації елементів у багатовимірному масиві:

```
positive_elements = matrix.flatten.select { |n| n > 0 }
```

Завдання до роботи

Із кожного блоку завдань виконати **два завдання**, які вибрати за своїм варіантом згідно списку у підгрупі по Ruby:

№Завдання_1_із_блоку_k= ((№студ_по_списку_підгрупи-1) % к-сть_завдань_у_блоці_k) + 1;
№Завдання_2_із_блоку_k = ((№студ_по_списку_підгрупи) % к-сть_завдань_у_блоці_k) + 1.

Наприклад: студент по списку у підгрупі Ruby під номером 7. Тоді його завдання із групи 1 такі:

№Завдання_1_із_блоку1 = ((7-1) % 4) +1 = 3

№Завдання_2_із_блоку1 = (7 % 4) + 1 = 4

Кожне завдання виконати хоча б **двома способами!** Для виконання завдань використовувати можливості ітераторів і методів мови Ruby, і уникати класичного використання циклів і умов

Блок 1. Завдання по роботі з одновимірними масивами.

1. Вивести індекси масиву в тому порядку, в якому відповідні їм елементи утворюють зростаючу послідовність.
2. У чисельному масиві знайти суму від’ємних елементів.

3. Знайти усі індекси, по яких розташовується максимальний елемент.
4. У масиві переставити в початок елементи, що стоять на парній позиції, а в кінець - що стоять на непарній.
5. Знайти усі елементи, більші за середнє арифметичне елементів.
6. До парних елементів додати перший елемент, а до непарних — останній. Перший і останній елемент не змінювати.
7. Замінити усі додатні елементи на значення мінімального.
8. Знайти добуток усіх парних елементів масиву.
9. Знайти кількість мінімальних елементів.
10. Вивести індекси елементів, менших свого лівого сусіда.

Блок 2. Завдання на двовимірні масиви

1. Поміняти перший і останній стовпець масиву місцями.
2. Упорядкувати N-й стовпець.
3. Упорядкувати рядки, що містять максимальний елемент.
4. Упорядкувати рядки, якщо вони відсортовані і перемішати, якщо вони невідсортовані.
5. Упорядкувати рядки масиву за значенням елементу головної діагоналі в кожному з рядків (у початковому масиві).
6. Знайти номери рядків, елементи яких впорядковані за збільшенням.
7. Знайти максимальний елемент для кожного стовпця, а потім отримати добуток цих елементів.
8. Знайти мінімум в двовимірному масиві.
9. Знайти добуток додатніх елементів.
10. Знайти суму додатніх елементів, більших за введену з клавіатури константу .
11. Обчислити суму і середнє арифметичне елементів головної діагоналі.
12. Знайти номери рядків, усі елементи яких - нулі.

Вимоги до звіту

1.1.1 Тема та мета роботи.

1.1.2 Скріншот, що демонструє версію Ruby і RubyOnRails.

1.1.3 Основна частина роботи, яку навести у наступному форматі:

- Сформульоване завдання 1;
- Лістинг виконання завдання 1 (чорним по білому);
- Результати виконання програми за завданням 1.
- Сформульоване завдання 2;

і т.д.

1.1.4 Висновки, що відображують результати виконання роботи та їх критичний аналіз.

Контрольні питання

1. Як за допомогою методу map можна змінити всі елементи двовимірного масиву?
2. Як скористатися методом flatten для перетворення багатовимірного масиву в одновимірний?
3. Як можна сортувати рядки двовимірного масиву за значеннями елементів у певному стовпці?
4. Як знайти максимальне і мінімальне значення в двовимірному масиві за допомогою методів max і min?
5. Як використовувати метод each_with_index для ітерації по елементах двовимірного масиву разом з їхніми індексами?
6. Як знайти суму і середнє арифметичне елементів головної діагоналі двовимірного масиву?
7. Як за допомогою методу select знайти всі додатні елементи в двовимірному масиві?
8. Яким чином можна поміняти місцями перший і останній стовпець двовимірного масиву?
9. Як реалізувати пошук рядків, елементи яких впорядковані за зростанням, у двовимірному масиві?

Рекомендовані ресурси

1. class Array. URL: https://docs.ruby-lang.org/en/3.2/Array.html
2. Ruby API. Array. URL: https://rubyapi.org/3.3/o/array

ЛАБОРАТОРНА РОБОТА № 3

Тема: Робота з асоціативними масивами

2.1 Основні теоретичні відомості

Hash в Ruby – це колекція пар ключ-значення, де кожен ключ унікальний. Це схоже на словник у інших мовах програмування. Хеш дозволяє ефективно зберігати і швидко знаходити значення за ключем. Створення хешу в Ruby може бути виконане кількома способами:

Порожній хеш:

```
empty_hash = {}
```

Хеш з початковими значеннями:

```
person = { "name" => "Alice", "age" => 30, "city" => "London" }
```

Хеш з символами як ключами (поширений спосіб):

```
person = { name: "Alice", age: 30, city: "London" }
```

Доступ до значення за ключем здійснюється за допомогою синтаксису квадратних дужок:

```
name = person[:name]  # повертає "Alice"
```

Якщо ключа не існує, повертається значення nil.

Додавання або оновлення елементів у хеші відбувається так само за допомогою квадратних дужок:

```
person[:job] = "Developer"  # додає нову пару ключ-значення
person[:age] = 31           # оновлює значення ключа "age"
```

Для видалення елементу з хешу можна використати метод delete:

```
person.delete(:city)  # видаляє пару з ключем :city
```

Для перевірки наявності ключа або значення в хеші використовуються методи key? і value?:

```
person.key?(:name)  # повертає true
person.value?("Alice")  # повертає true
```

Ітерація по елементах хешу може бути здійснена за допомогою методів each, each_key, each_value:

```
person.each do |key, value|
  puts "#{key}: #{value}"
end
```

Хеші можуть містити інші хеші в якості значень, що дозволяє створювати вкладені структури даних:

```
person = { name: "Alice", age: 30, address:
           { city: "London", zip: "E1 6AN" } }
```

Доступ до вкладених елементів здійснюється через послідовний доступ по ключах:

```
city = person[:address][:city]  # повертає "London"
```

Метод merge дозволяє об'єднати два хеші, повертаючи новий хеш, який містить пари з обох хешів. Якщо ключі співпадають, значення з другого хешу замінюють значення з першого:

```
hash1 = { a: 1, b: 2 }
hash2 = { b: 3, c: 4 }
result = hash1.merge(hash2)  # повертає { a: 1, b: 3, c: 4 }
```

Хеш можна перетворити в масив пар ключ-значення та назад:

```
array = person.to_a  # перетворює хеш в масив
hash_from_array = array.to_h  # перетворює масив назад в хеш
```

Можна задати значення за замовчуванням для хешу, яке буде повертатись, якщо шуканий ключ відсутній:

```
default_hash = Hash.new("default")
default_hash[:missing_key]  # повертає "default"
```

2.2 Завдання для виконання

Реалізувати задачу згідно варіанту. Всі завдання виконувати з хешем (окрім завдання 2). В завданні 3 передбачити вибір опції з клавіатури. При підготовці до

захисту роботи **продумати альтернативний варіант(и)** рішення кожного завдання.

Вимоги до звіту:

2.2.1 Тема та мета роботи.

2.2.2 Скріншот, що демонструє версію Ruby і RubyOnRails.

2.2.3 Основна частина роботи, яку навести у наступному форматі:

- Сформульоване завдання 1;
- Лістинг виконання завдання 1 (чорним по білому);
- Результати виконання програми за завданням 1.
- Сформульоване завдання 2;

і т.д.

2.2.4 Висновки, що відображують результати виконання роботи та їх критичний аналіз.

Варіанти завдань:

1. Інтернет-магазин

Завдання 1: Створіть хеш, який містить 10 товарів інтернет-магазину. Ключем має бути назва товару, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим, попередньо ввівши по чому сортувати (напр. по назві товару, по ціні і т.п.).

Завдання 4: Вивести 3 товари, які мають найкоротшу назву.

Завдання 5: Вивести 5 найдешевших товарів з ненульовою вартістю.

Завдання 6: Перевірити початковий масив на наявність товару, назва якого введена з клавіатури, вивести кількість цього товару.

Завдання 7: Обчислити вартість усіх товарів, які є на складі.

Завдання 8: Перевірити, чи є в базі товари, для яких не задана ціна (або рівна нулю). Якщо є такі, ввести ціну з клавіатури. Якщо ціна нульова, а кількість товару не нульова – ввести ціну з клавіатури. Застосувати ітератори.

2. Журнал оцінок студентів

Завдання 1: Створіть хеш, який містить 10 студентів. Ключем має бути ім'я студента, а значенням — його середній бал.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за середнім балом або іменем студента.

Завдання 4: Вивести 3 студентів, які мають найкоротші імена.

Завдання 5: Вивести 5 студентів з найнижчими середніми балами, які не мають нульовий бал.

Завдання 6: Перевірити початковий масив на наявність студента, ім'я якого введено з клавіатури, і вивести його середній бал.

Завдання 7: Обчислити загальний середній бал усіх студентів.

Завдання 8: Перевірити, чи є в базі студенти без заданого середнього балу (або з нульовим балом). Якщо є такі, ввести їхні бали з клавіатури.

3. Кінотеатр

Завдання 1: Створіть хеш, який містить 10 фільмів. Ключем має бути назва фільму, а значенням — його ціна квитка.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою фільму або ціною квитка.

Завдання 4: Вивести 3 фільми, які мають найкоротшу назву.

Завдання 5: Вивести 5 найдешевших квитків на фільми з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність фільму, назва якого введена з клавіатури, і вивести кількість доступних квитків.

Завдання 7: Обчислити загальний дохід від продажу квитків на всі фільми.

Завдання 8: Перевірити, чи є в базі фільми без заданої ціни квитка (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

4. Бібліотека

Завдання 1: Створіть хеш, який містить 10 книг у бібліотеці. Ключем має бути назва книги, а значенням — кількість доступних примірників.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою книги або кількістю примірників.

Завдання 4: Вивести 3 книги, які мають найкоротшу назву.

Завдання 5: Вивести 5 книг з найменшою кількістю примірників, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність книги, назва якої введена з клавіатури, і вивести кількість доступних примірників.

Завдання 7: Обчислити загальну кількість всіх книг у бібліотеці.

Завдання 8: Перевірити, чи є в базі книги без заданої кількості примірників (або з нульовою кількістю). Якщо є такі, ввести їхню кількість з клавіатури.

5. Система управління запасами на складі

Завдання 1: Створіть хеш, який містить 10 товарів на складі. Ключем має бути назва товару, а значенням — кількість на складі.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою товару або кількістю на складі.

Завдання 4: Вивести 3 товари, які мають найкоротшу назву.

Завдання 5: Вивести 5 товарів з найменшою кількістю на складі, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність товару, назва якого введена з клавіатури, і вивести його кількість на складі.

Завдання 7: Обчислити загальну кількість всіх товарів на складі.

Завдання 8: Перевірити, чи є в базі товари з нульовою кількістю на складі. Якщо є такі, ввести їхню кількість з клавіатури.

6. Управління проектами

Завдання 1: Створіть хеш, який містить 10 проектів. Ключем має бути назва проекту, а значенням — його статус (наприклад, "в процесі", "завершено").

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою проекту або статусом.

Завдання 4: Вивести 3 проекти, які мають найкоротшу назву.

Завдання 5: Вивести 5 проектів з найменшою кількістю завдань, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність проекту, назва якого введена з клавіатури, і вивести його статус.

Завдання 7: Обчислити загальну кількість завдань усіх проектів.

Завдання 8: Перевірити, чи є в базі проекти без заданого статусу (або з нульовим статусом). Якщо є такі, ввести їхні статуси з клавіатури.

7. Менеджмент персоналу

Завдання 1: Створіть хеш, який містить 10 працівників. Ключем має бути ім'я працівника, а значенням — його посада.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за іменем працівника або посадою.

Завдання 4: Вивести 3 працівники, які мають найкоротші імена.

Завдання 5: Вивести 5 працівників з найменшою зарплатою, які мають ненульову зарплату.

Завдання 6: Перевірити початковий масив на наявність працівника, ім'я якого введено з клавіатури, і вивести його посаду.

Завдання 7: Обчислити загальну суму зарплат усіх працівників.

Завдання 8: Перевірити, чи є в базі працівники без заданої зарплати (або з нульовою зарплатою). Якщо є такі, ввести їхні зарплати з клавіатури.

8. Система управління подіями

Завдання 1: Створіть хеш, який містить 10 подій. Ключем має бути назва події, а значенням — кількість доступних квитків.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою події або кількістю квитків.

Завдання 4: Вивести 3 події, які мають найкоротшу назву.

Завдання 5: Вивести 5 подій з найменшою кількістю квитків, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність події, назва якої введена з клавіатури, і вивести кількість доступних квитків.

Завдання 7: Обчислити загальну кількість всіх квитків на події.

Завдання 8: Перевірити, чи є в базі події з нульовою кількістю квитків. Якщо є такі, ввести їхню кількість з клавіатури.

9. Система бронювання готелю

Завдання 1: Створіть хеш, який містить 10 номерів готелю. Ключем має бути тип номера, а значенням — кількість доступних кімнат.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за типом номера або кількістю кімнат.

Завдання 4: Вивести 3 типи номерів з найкоротшими назвами.

Завдання 5: Вивести 5 типів номерів з найменшою кількістю кімнат, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність типу номера, назва якого введена з клавіатури, і вивести його кількість доступних кімнат.

Завдання 7: Обчислити загальну кількість усіх кімнат у готелі.

Завдання 8: Перевірити, чи є в базі типи номерів з нульовою кількістю кімнат. Якщо є такі, ввести їхню кількість з клавіатури.

10. Музичний магазин

Завдання 1: Створіть хеш, який містить 10 музичних альбомів. Ключем має бути назва альбому, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою альбому або ціною.

Завдання 4: Вивести 3 альбоми з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших альбомів з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність альбому, назва якого введена з клавіатури, і вивести його кількість на складі.

Завдання 7: Обчислити загальну вартість усіх альбомів у магазині.

Завдання 8: Перевірити, чи є в базі альбоми без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

11. Система управління бібліографічними даними

Завдання 1: Створіть хеш, який містить 10 бібліографічних записів. Ключем має бути назва книги, а значенням — її автор.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою книги або автором.

Завдання 4: Вивести 3 записи з найкоротшими назвами книг.

Завдання 5: Вивести 5 записів з найменшими назвами авторів, які мають ненульову довжину.

Завдання 6: Перевірити початковий масив на наявність книги, назва якої введена з клавіатури, і вивести її автора.

Завдання 7: Обчислити загальну кількість авторів у бібліографічних даних.

Завдання 8: Перевірити, чи є в базі записи без автора (або з порожнім полем). Якщо є такі, ввести їхніх авторів з клавіатури.

12. Система управління рецептами

Завдання 1: Створіть хеш, який містить 10 рецептів. Ключем має бути назва страви, а значенням — час приготування в хвилинах.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою страви або часом приготування.

Завдання 4: Вивести 3 рецепти з найкоротшими назвами страв.

Завдання 5: Вивести 5 найшвидших рецептів з ненульовим часом приготування.

Завдання 6: Перевірити початковий масив на наявність рецепту, назва якого введена з клавіатури, і вивести його час приготування.

Завдання 7: Обчислити загальний час приготування усіх рецептів.

Завдання 8: Перевірити, чи є в базі рецепти без заданого часу приготування (або з нульовим часом). Якщо є такі, ввести їхні часи з клавіатури.

13. Система управління транспортними засобами

Завдання 1: Створіть хеш, який містить 10 транспортних засобів. Ключем має бути модель автомобіля, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за моделлю автомобіля або ціною.

Завдання 4: Вивести 3 автомобілі з найкоротшими назвами моделей.

Завдання 5: Вивести 5 найдешевших автомобілів з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність автомобіля, модель якого введена з клавіатури, і вивести його кількість на складі.

Завдання 7: Обчислити загальну вартість усіх автомобілів на складі.

Завдання 8: Перевірити, чи є в базі автомобілі без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

14. Система управління бібліотекою фільмів

Завдання 1: Створіть хеш, який містить 10 фільмів у бібліотеці фільмів. Ключем має бути назва фільму, а значенням — його жанр.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою фільму або жанром.

Завдання 4: Вивести 3 фільми з найкоротшими назвами.

Завдання 5: Вивести 5 фільмів з найменшими назвами жанрів, які мають ненульову довжину.

Завдання 6: Перевірити початковий масив на наявність фільму, назва якого введена з клавіатури, і вивести його жанр.

Завдання 7: Обчислити загальну кількість жанрів у бібліотеці фільмів.

Завдання 8: Перевірити, чи є в базі фільми без жанру (або з порожнім жанром). Якщо є такі, ввести їхні жанри з клавіатури.

15. Система управління подорожами

Завдання 1: Створіть хеш, який містить 10 подорожей. Ключем має бути напрямок подорожі, а значенням — її ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за напрямком подорожі або ціною.

Завдання 4: Вивести 3 подорожі з найкоротшими назвами напрямків.

Завдання 5: Вивести 5 найдешевших подорожей з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність подорожі, напрямок якої введений з клавіатури, і вивести її кількість доступних місць.

Завдання 7: Обчислити загальну вартість усіх подорожей.

Завдання 8: Перевірити, чи є в базі подорожі без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

16. Система управління рестораном

Завдання 1: Створіть хеш, який містить 10 страв у ресторані. Ключем має бути назва страви, а значенням — її ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою страви або ціною.

Завдання 4: Вивести 3 страви з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших страв з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність страви, назва якої введена з клавіатури, і вивести її кількість у меню.

Завдання 7: Обчислити загальну вартість усіх страв у меню.

Завдання 8: Перевірити, чи є в базі страви без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

17. Система управління спортзалом

Завдання 1: Створіть хеш, який містить 10 тренувальних програм у спортзалі. Ключем має бути назва програми, а значенням — її ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою програми або ціною.

Завдання 4: Вивести 3 програми з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших програм з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність програми, назва якої введена з клавіатури, і вивести її кількість доступних місць.

Завдання 7: Обчислити загальну вартість усіх програм у спортзалі.

Завдання 8: Перевірити, чи є в базі програми без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

18. Система управління музичними інструментами

Завдання 1: Створіть хеш, який містить 10 музичних інструментів. Ключем має бути назва інструменту, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою інструменту або ціною.

Завдання 4: Вивести 3 інструменти з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших інструментів з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність інструменту, назва якого введена з клавіатури, і вивести його кількість на складі.

Завдання 7: Обчислити загальну вартість усіх інструментів у магазині.

Завдання 8: Перевірити, чи є в базі інструменти без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

19. Система управління курсами навчання

Завдання 1: Створіть хеш, який містить 10 курсів навчання. Ключем має бути назва курсу, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою курсу або ціною.

Завдання 4: Вивести 3 курси з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших курсів з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність курсу, назва якого введена з клавіатури, і вивести його кількість доступних місць.

Завдання 7: Обчислити загальну вартість усіх курсів.

Завдання 8: Перевірити, чи є в базі курси без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

20. Система управління подкастами

Завдання 1: Створіть хеш, який містить 10 подкастів. Ключем має бути назва подкасту, а значенням — його тривалість у хвилинах.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою подкасту або тривалістю.

Завдання 4: Вивести 3 подкасти з найкоротшими назвами.

Завдання 5: Вивести 5 подкастів з найменшою тривалістю, які мають ненульову тривалість.

Завдання 6: Перевірити початковий масив на наявність подкасту, назва якого введена з клавіатури, і вивести його тривалість.

Завдання 7: Обчислити загальну тривалість усіх подкастів.

Завдання 8: Перевірити, чи є в базі подкасти без заданої тривалості (або з нульовою тривалістю). Якщо є такі, ввести їхні тривалості з клавіатури.

21. Система управління бібліотекою журналів

Завдання 1: Створіть хеш, який містить 10 журналів у бібліотеці. Ключем має бути назва журналу, а значенням — його жанр.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою журналу або жанром.

Завдання 4: Вивести 3 журнали з найкоротшими назвами.

Завдання 5: Вивести 5 журналів з найменшим числом номерів, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність журналу, назва якого введена з клавіатури, і вивести його жанр.

Завдання 7: Обчислити загальну кількість журналів у бібліотеці.

Завдання 8: Перевірити, чи є в базі журнали без заданого жанру (або з порожнім жанром). Якщо є такі, ввести їхні жанри з клавіатури.

22. Система управління ресторанними меню

Завдання 1: Створіть хеш, який містить 10 страв у меню ресторану. Ключем має бути назва страви, а значенням — її ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою страви або ціною.

Завдання 4: Вивести 3 страви з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших страв з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність страви, назва якої введена з клавіатури, і вивести її ціну.

Завдання 7: Обчислити загальну вартість усіх страв у меню.

Завдання 8: Перевірити, чи є в базі страви без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

23. Система управління спортивними командами

Завдання 1: Створіть хеш, який містить 10 гравців у спортивній команді. Ключем має бути ім'я гравця, а значенням — його позиція.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за ім'ям гравця або позицією.

Завдання 4: Вивести 3 гравців з найкоротшими іменами.

Завдання 5: Вивести 5 гравців з найменшими рейтингами, які мають ненульовий рейтинг.

Завдання 6: Перевірити початковий масив на наявність гравця, ім'я якого введено з клавіатури, і вивести його позицію.

Завдання 7: Обчислити загальну кількість гравців у команді.

Завдання 8: Перевірити, чи є в базі гравці без заданої позиції (або з порожньою позицією). Якщо є такі, ввести їхні позиції з клавіатури.

24. Система управління курсами університету

Завдання 1: Створіть хеш, який містить 10 курсів університету. Ключем має бути назва курсу, а значенням — кількість кредитів.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою курсу або кількістю кредитів.

Завдання 4: Вивести 3 курси з найкоротшими назвами.

Завдання 5: Вивести 5 курсів з найменшою кількістю кредитів, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність курсу, назва якого введена з клавіатури, і вивести кількість його кредитів.

Завдання 7: Обчислити загальну кількість кредитів усіх курсів.

Завдання 8: Перевірити, чи є в базі курси без заданої кількості кредитів (або з нульовою кількістю). Якщо є такі, ввести їхні кредити з клавіатури.

25. Система управління транспортними маршрутами

Завдання 1: Створіть хеш, який містить 10 транспортних маршрутів. Ключем має бути назва маршруту, а значенням — його довжина в кілометрах.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою маршруту або довжиною.

Завдання 4: Вивести 3 маршрути з найкоротшими назвами.

Завдання 5: Вивести 5 маршрутів з найменшою довжиною, які мають ненульову довжину.

Завдання 6: Перевірити початковий масив на наявність маршруту, назва якого введена з клавіатури, і вивести його довжину.

Завдання 7: Обчислити загальну довжину усіх маршрутів.

Завдання 8: Перевірити, чи є в базі маршрути без заданої довжини (або з нульовою довжиною). Якщо є такі, ввести їхні довжини з клавіатури.

26. Система управління бібліотекою аудіокниг

Завдання 1: Створіть хеш, який містить 10 аудіокниг у бібліотеці. Ключем має бути назва аудіокниги, а значенням — її тривалість у хвилинах.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою аудіокниги або тривалістю.

Завдання 4: Вивести 3 аудіокниги з найкоротшими назвами.

Завдання 5: Вивести 5 аудіокниг з найменшою тривалістю, які мають ненульову тривалість.

Завдання 6: Перевірити початковий масив на наявність аудіокниги, назва якої введена з клавіатури, і вивести її тривалість.

Завдання 7: Обчислити загальну тривалість усіх аудіокниг у бібліотеці.

Завдання 8: Перевірити, чи є в базі аудіокниги без заданої тривалості (або з нульовою тривалістю). Якщо є такі, ввести їхні тривалості з клавіатури.

27. Система управління бібліотекою фотографій

Завдання 1: Створіть хеш, який містить 10 фотографій у бібліотеці. Ключем має бути назва фотографії, а значенням — її розмір у мегабайтах.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою фотографії або розміром.

Завдання 4: Вивести 3 фотографії з найкоротшими назвами.

Завдання 5: Вивести 5 фотографій з найменшим розміром, які мають ненульовий розмір.

Завдання 6: Перевірити початковий масив на наявність фотографії, назва якої введена з клавіатури, і вивести її розмір.

Завдання 7: Обчислити загальний розмір усіх фотографій у бібліотеці.

Завдання 8: Перевірити, чи є в базі фотографії без заданого розміру (або з нульовим розміром). Якщо є такі, ввести їхні розміри з клавіатури.

28. Система управління курсами онлайн-навчання

Завдання 1: Створіть хеш, який містить 10 онлайн-курсів. Ключем має бути назва курсу, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою курсу або ціною.

Завдання 4: Вивести 3 курси з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших курсів з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність курсу, назва якого введена з клавіатури, і вивести його ціну.

Завдання 7: Обчислити загальну вартість усіх курсів.

Завдання 8: Перевірити, чи є в базі курси без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

29. Система управління бібліотекою відеоігор

Завдання 1: Створіть хеш, який містить 10 відеоігор у бібліотеці. Ключем має бути назва гри, а значенням — її жанр.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою гри або жанром.

Завдання 4: Вивести 3 відеоігри з найкоротшими назвами.

Завдання 5: Вивести 5 відеоігор з найменшим числом гравців, які мають ненульову кількість.

Завдання 6: Перевірити початковий масив на наявність відеоігри, назва якої введена з клавіатури, і вивести її жанр.

Завдання 7: Обчислити загальну кількість жанрів у бібліотеці відеоігор.

Завдання 8: Перевірити, чи є в базі відеоігри без заданого жанру (або з порожнім жанром). Якщо є такі, ввести їхні жанри з клавіатури.

30. Система управління рецептурами косметики

Завдання 1: Створіть хеш, який містить 10 рецептур косметичних засобів. Ключем має бути назва засобу, а значенням — його ціна.

Завдання 2: Перетворити цей масив у індексний та вивести його.

Завдання 3: Вивести початковий масив відсортованим за назвою засобу або ціною.

Завдання 4: Вивести 3 косметичні засоби з найкоротшими назвами.

Завдання 5: Вивести 5 найдешевших рецептур з ненульовою ціною.

Завдання 6: Перевірити початковий масив на наявність рецептури, назва якої введена з клавіатури, і вивести її ціну.

Завдання 7: Обчислити загальну вартість усіх косметичних засобів.

Завдання 8: Перевірити, чи є в базі рецептури без заданої ціни (або з нульовою ціною). Якщо є такі, ввести їхні ціни з клавіатури.

2.3 Контрольні питання

1. Як створити хеш з ключами-символами і значеннями-рядками в Ruby? Наведіть приклад.
2. Яким чином у Ruby можна перевірити наявність ключа в хеші, використовуючи метод key? або його альтернативу?
3. Як у Ruby здійснюється доступ до значення хешу за ключем і що відбудеться, якщо ключа не існує?
4. Як можна додати нову пару ключ-значення до хешу і оновити існуюче значення за ключем?
5. Яким методом у Ruby видаляється елемент з хешу за ключем і яке значення при цьому повертається?
6. Як використати метод each для ітерації по всіх ключах і значеннях хешу? Наведіть приклад.
7. Як метод fetch відрізняється від звичайного доступу до значення в хеші і як задати значення за замовчуванням у випадку відсутності ключа?
8. Яким чином можна об'єднати два хеші в один, використовуючи метод merge? Що відбувається у випадку конфлікту ключів?
9. Як перетворити хеш у масив пар ключ-значення і навпаки? Наведіть приклади.
10. Як перевірити наявність значення в хеші, використовуючи метод value? або його альтернативу?
11. Як створити вкладений хеш у Ruby і як отримати доступ до його елементів? Наведіть приклад.

2.4 Рекомендовані ресурси

1. class Hash. URL: https://docs.ruby-lang.org/en/3.2/Hash.html
2. Ruby API. Hash. URL: https://rubyapi.org/3.3/o/hash

ЛАБОРАТОРНА РОБОТА № 4

Тема: Робота з рядками в мові Ruby

Мета роботи: Ознайомитись з підходами до обробки рядків у мові Ruby зокрема з використанням регулярних виразів.

4.1 Основні теоретичні відомості

У Ruby регулярні вирази (РВ) використовуються для пошуку, перевірки та маніпуляції текстовими даними. Вони є потужним інструментом для обробки рядків. Основні методи роботи з РВ:

Перевірка відповідності:

`=~` - повертає індекс першого входження або nil.

```
"hello" =~ /ll/ # => 2
"hello" =~ /z/  # => nil
```

`.match` – повертає об'єкт MatchData або nil.

```
result = "hello".match(/ll/) # => #<MatchData "ll">
result[0] # => "ll"
```

Пошук і заміна:

`.sub` - замінює перше входження.

```
"hello".sub(/l/, "r") # => "herlo"
```

`.gsub` - замінює всі входження.

```
"hello".gsub(/l/, "r") # => "herro"
```

Розбиття рядка:

`.split` - ділить рядок за шаблоном РВ.

```
"1,2,3".split(/,/) # => ["1", "2", "3"]
```

Перевірка наявності:

`.scan` – повертає масив усіх збігів.

```
"hello world".scan(/o/) # => ["o", "o"]
```

Фільтрація тексту:

`.select` разом із блоком для вибору елементів, що відповідають РВ.

```
["apple", "banana", "cherry"].select { |word| word =~ /a/ }
            # => ["apple", "banana"]
```

Якщо метод повертає об'єкт MatchData, він дозволяє доступ до:

- Знайденого тексту (match[0]).
- Груп захоплення (match[1], match[2], ...).

```
match = /(\d+)-(\w+)/.match("123-abc")
match[1] # => "123"
match[2] # => "abc"
```

4.2 Завдання до роботи

Використовуючи теоретичний матеріал по роботі з рядками реалізувати поставлені завдання. Парні варіанти виконують з завдання з парним номером, і відповідно непарні. Кожну із виконуваних задач реалізувати **двома** способами. Для всіх задач, окрім 1 і 2 використати регулярні вирази принаймні в одній із реалізацій. В кінці роботи навести висновки.

Основну частину роботи навести у наступному форматі:

• Сформульоване завдання 1;

• Лістинг виконання завдання 1 (чорним по білому);

• Результати виконання програми за завданням 1.

• Сформульоване завдання 2;

і т.д.

Завдання для виконання:

1. Створіть рядок слів, розділених пробілами. Вивести найдовше слово.
2. Створіть рядок, що містить кирилицю, латиницю і цифри. Вивести усі слова, довжина яких рівна середній.

3. Знайти в рядку перше цілком кириличне слово.
4. Даний текст (рядок з перенесеннями). Знайти усі слова, що містять лише три букви "о".
5. Знайти в тексті час у форматі "години:хвилини:секунди".
6. Знайти усі слова без букв що повторюються (наприклад, "Лісп" або "Ruby", але не "Паскаль" або "Java"),.не враховуйте цифри в словах.
7. Знайти в тексті слова, що містять дві прописні букви і виправити.
8. Знайти в тексті дати формату "день.місяць.рік".
9. Знайдіть дату, де день обмежений числом 31, а місяць 12. Рік обмежуйте чотиризначними числами.
10. Дано текст. Знайдіть усі URL адреси і виділіть з них тільки кореневий домен (наприклад, з https://en.wikibooks.org/wiki/Ruby_Programming зробіть https://en.wikibooks.org).

4.4 Контрольні питання

1. Що повертає оператор =~, якщо вхідний рядок не відповідає регулярному виразу?
2. Яку різницю має використання методів .sub і .gsub для роботи з рядками?
3. Які методи можна використовувати для отримання всіх збігів регулярного виразу у рядку?
4. Що таке об’єкт MatchData, і які дані він дозволяє отримати?
5. Як у регулярному виразі забезпечити нечутливість до регістру під час пошуку?
6. Яке призначення модифікаторів /m та /x у регулярних виразах?
7. Як розділити рядок на частини за певним шаблоном, використовуючи регулярний вираз?
8. У чому різниця між використанням оператора =~ і методу .match?
9. Як за допомогою регулярних виразів перевірити, чи містить рядок певний шаблон?

4.5 Рекомендовані ресурси

1. class RegExp. URL: https://docs.ruby-lang.org/en/3.2/Regexp.html

ЛАБОРАТОРНА РОБОТА № 5

Тема: Методи в мові Ruby

Мета роботи: Ознайомитись з різновидами методів в мові Ruby та навчитися їх використовувати.

5.1 Основні теоретичні відомості

Методи визначаються за допомогою ключового слова def. Після цього йде ім'я методу, список параметрів (необов'язковий) і тіло методу. Завершується метод ключовим словом end.

```
def greet(name)
  "Hello, #{name}!"
end
```

Методи викликаються за їхнім іменем. Якщо метод приймає параметри, вони передаються у круглих дужках (дужки необов'язкові, якщо це не впливає на читабельність).

```
greet("Alice") # => "Hello, Alice!"
```

Типи параметрів:

Обов'язкові параметри: Їх потрібно передати під час виклику.

```
def add(a, b)
  a + b
end
```

Параметри за замовчуванням: Їм можна призначити значення за замовчуванням.

```
def greet(name = "Guest")
  "Hello, #{name}!"
end
```

Іменовані параметри: Використовуються для передачі іменованих аргументів.

```
def greet(title:, name:)
  "Hello, #{title} #{name}!"
end
```

Змінна кількість параметрів: Метод може приймати будь-яку кількість аргументів.

```
def sum(*numbers)
      numbers.sum
end
```

Методи повертають останнє обчислене значення. Для явного повернення можна використовувати return.

```
def multiply(a, b)
  return a * b if a > 0 && b > 0
  "Invalid input"
end
```

Використання блоків

Методи можуть приймати блоки, які виконуються за допомогою yield.

```
def repeat(times)
  times.times { yield }
end

repeat(3) { puts "Hello!" }
```

Singleton-методи

Це методи, які належать конкретному об'єкту, а не класу.

```
obj = "hello"
def obj.shout
  upcase
end

obj.shout # => "HELLO"
```

Використання методів як аргументів

Методи можна передавати як аргументи за допомогою & або використовувати об’єкт Proc.

```
def execute_block(&block)
  block.call
end

execute_block { puts "Block called!" }
```

5.2 Завдання для виконання

Використовуючи теоретичний матеріал п.5.1 та лекції по роботі з методами реалізувати поставлені завдання. В кінці роботи навести висновки.

Основну частину звіту сформувати у такому вигляді:

- Сформульоване завдання 1;
- Лістинг виконання завдання 1 (чорним по білому);
- Результати виконання програми за завданням 1.
- Сформульоване завдання 2;

і т.д.

Завдання для виконання:

Розробити та протестувати методи наступних видів:

1. Метод із аргументами по замовчуванню, перевизначення методів
2. Метод з необов'язковими іменованими параметрами (наприклад:
   def greet(name:, age: nil, city: "Unknown"))
2. Логічні методи
3. Методи оператори і методи присвоювання
4. Методи із масивом в якості аргументу
5. Псевдонім методу
6. Методи-блоки із аргументом
7. yeld-методи з аргументом і без нього

8. Proc без аргументів і з аргументом (показати «неконтрольованість» аргументів, та особливості return)
9. lambda без аргументів і з аргументом (показати «контрольованість» аргументів, та особливості return)

Індивідуальна тематика завдань для розробки методів згідно варіантів:

1. Методи для роботи готелю
2. Методи для роботи кінотеатру
3. Методи для роботи ресторану
4. Методи для обробки замовлень в інтернет-магазині
5. Методи для керування бібліотекою
6. Методи для організації заходів (концертів, конференцій)
7. Методи для управління запасами на складі
8. Методи для обробки даних клієнтів в CRM-системі
9. Методи для обчислення податків в бухгалтерії
10. Методи для управління студентами в університеті
11. Методи для роботи з користувачами у соціальній мережі
12. Методи для роботи з книгами в електронній бібліотеці
13. Методи для обчислення заробітної плати співробітників компанії
14. Методи для обчислення витрат на поїздки та транспорт
15. Методи для ведення обліку студентів на курсах
16. Методи для керування персоналом у компанії
17. Методи для роботи з обліковими записами користувачів у банківській системі
18. Методи для ведення списку товарів в магазині
19. Методи для обробки даних про пацієнтів у медичній клініці
20. Методи для роботи з поштовими відправленнями в поштовій службі
21. Методи для планування та організації спортивних змагань
22. Методи для роботи зі студентами в освітній платформі
23. Методи для створення та управління проектами в проектному менеджменті

24. Методи для ведення архіву клієнтських звернень у службі підтримки
25. Методи для обробки даних про замовлення в системі доставки
26. Методи для управління виставковими залами або музеями
27. Методи для організації роботи на будівельному майданчику
28. Методи для керування завданнями у команді розробників програмного забезпечення
29. Методи для роботи з даними про тури в туристичному агентстві
30. Методи для розрахунку кредитних платежів у банківській системі

5.3 Контрольні питання

1. Що повертає метод у Ruby, якщо немає явного ключового слова return?
2. Як передати параметри до методу? У чому різниця між параметрами за замовчуванням і обов'язковими параметрами?
3. Що таке ключові параметри методу, і як вони використовуються?
4. Як створити метод, який приймає змінну кількість аргументів? Наведіть приклад.
5. У чому відмінність між методами public, private та protected? Коли слід використовувати кожен із них?
6. Що таке Proc у Ruby? Як створити та викликати об'єкт типу Proc?
7. У чому різниця між Proc і Lambda? Як вони обробляють return і перевіряють кількість аргументів?
8. Як передати блок до методу? Яка різниця між yield та явною передачею блоку через &block?
9. Що таке singleton-методи, і для чого вони використовуються? Як створити singleton-метод у Ruby?

5.4 Рекомендовані ресурси

1. Object. URL: https://rubyapi.org/3.3/o/object
2. class Proc. URL: https://docs.ruby-lang.org/en/3.2/Proc.html

ЛАБОРАТОРА РОБОТА № 6

Об’єктно-орієнтоване програмування в Ruby

6.1 Мета роботи

Засвоїти основи об’єктно-орієнтованого програмування у Ruby.

6.2 Основні теоретичні відомості

Мова програмування Ruby підтримує основні парадигми об’єктно-орієнтованого програмування: наслідування, поліморфізм, інкапсуляція.

Об'єктне — орієнтоване програмування — це програмування засноване на використанні об'єктів — екземплярів абстрактних типів (класах). Об'єкт в ООП — це програмна модель якогось реально існуючого об'єкту, наприклад студента або викладача, при цьому будучи об'єктами реального світу ми будемо екземплярами деякого типу (виду) — Людини Розумної. Об'єкти, що є екземплярами одного класу називаються спорідненими або братами, а клас від якого вони пішли, якщо провести аналогію, є шаблоном ДНК людини в якому описуються усі загальні моменти похідних об'єктів, проте кожен об'єкт може мати свої відмінності від іншого об'єкту того ж типу.

Усі люди однакові в тому, що мають дві ноги, два очі, проте у усіх людей розрізняється колір очей, відбитки пальців, ім'я, характер, особа. Не дивлячись на ці відмінності між людьми, ми все-таки відрізняємо інших людей, скажемо від макак або павіанів. Усе загальне в людях і загальний набір властивостей визначається в класі, а усе різне визначається в самих об'єктах.

ООП є значним кроком в перед, який дозволяє писати програмістові не просто код, а описувати реальну модель взаємодії об'єктів. Об'єктно-орієнтоване програмування підвищує читабельність коду за рахунок його систематизації і подібності з того, як людина мислить і представляє світ.

Три кити об'єктно-орієнтованого програмування: Інкапсуляція, Наслідування, Поліморфізм

Спершу створимо код, на якому ми розбиратимемо приклади, він складатиметься з двох класів: Kitten і Cat, де Kitten наслідується від класу Cat

Що таке Інкапсуляція?

Інкапсуляція — це менший кит об'єктно-орієнтованого програмування. Інкапсуляція полягає в тому, що властиві функціональному програмування дані і функції над ними перетворилися на властивості і методи об'єкту, які поміщені в об'єкт.

Доступ до цих властивостей і методів доступний тільки через спеціально визначувані інтерфейси. Ці інтерфейси називаються геттери і сетери. Геттери набувають значення, а сетери встановлюють.

Таким чином ховається внутрішній устрій об'єкту. Якщо ви заходите поїсти, вам не треба знати усі методи вашого організму по переварюванню їжі. Усе, що вам потрібне — це скористатися спеціальним інтерфейсом — ротом, передати туди їжу (встановити значення), при цьому внутрішні методи вашого організму запустяться самі, взаємодіючи з цим одним єдиним інтерфейсом (методом — аксесором, від access — доступ).

Ваші властивості, як екземпляра хомо сапієнса, це розміри елементів вашого тіла, колір шкіри, колір очей, родимки і веснянки, ваше ім'я і прізвище, ваш фізичний, духовний і психологічний стан. Ви хороший, здоровий, темноокий, блідолиций, дев'яносто кілограмовий і ста-вісімдесяти сантиметровий Іван — і усе, що тут перераховано — це ваші властивості.

Ваші методи — це ходьба, біг, вживання їжі, сон, читання, писання, програмування і так далі.

Властивості до яких можна отримати доступ називаються відкритими, відкриті вони не із-за якоїсь своєї особливості, а через те, що для них визначені методи — аксесори.

За допомогою наших об'єктів покажемо простий приклад інкапсуляції

```
class Cat
 def initialize(options={})
   @height = options[:height]
```

```
    @weight = options[:weight]
  end

  # Наш геттер для властивості ріст
  def height
    @height
  end

  # Наш геттер для властивості вага
  def weight
    @weight
  end

  # Встановлюємо сетер для властивості ріст
  def height=(height)
    @height = height
  end

  def weight=(weight)
    @weight = weight
  end
end

tom = Cat.new({height: 10, weight: 100})
p tom.height
tom.height = 25
p tom.height
```

Обов'язково поекспериментуйте з геттерами і сетерами. Подивіться, що відбувається якщо їх прибрати.

Метод *initialize* дозволяє задати властивості об'єкту при його створенні (при використанні методу new). Де властивості передаємо як хеш значень.

Можна використати спрощене створення геттерів і сетерів:

```
attr_reader :height, :weight # Для геттера
attr_writer :height, :weight # Для сеттера
attr_accessor:height,:weight #Для геттера и сеттера
```

Що таке Наслідування

Наслідування — наступний кит ООП, проте він дуже важливий. Наслідування полягає в передачі усієї структури одного класу в іншій, іншими словами це просто копіювання одного класу в іншій. Але тут ще один важливий момент, що наслідування дозволяє створювати додаткові властивості, не доступні класу від якого пішло наслідування.

Як котеня пішло від кішки, так і один клас може наслідувати від іншого.

Наслідування реалізується дуже просто, при визначенні класу Kitten досить дописати < Cat, що і вказує на наслідування класу Kitten від класу Cat.

Говорять, що Cat наслідує Kitten, а Kitten наслідує від Cat. Говорять, що Cat — супер клас або батьківський клас, а Kitten — підклас, субклас або дочірній клас.

Наслідування — дуже корисна річ, оскільки: надає можливість повторного використання коду, рятує від повторень в коді.

Покажемо як відбувається наслідування і поява нових властивостей у спадкоємця.

```
 class Kitten < Cat
 def initialize(options)
   @name = options[:name]
   super
 end

 def name
   @name
 end

 def name=(name)
   @name = name
 end
end

p k = Kitten.new({height: 1,weight: 10, name: 'Tom'})
```

Поліморфізм — усе змінюється!

Поліморфізм — найжирніший кит об'єктно-орієнтованого програмування, який тісно пов'язаний з китом спадкоємства і робить наслідування ще потужнішим. Поліморфізм (полі — багато, морфе — форма) полягає в тому, що наслідуючись в класі — нащадку (дочірньому класі або підкласі або субкласі або .) ви можете перевизначати успадковані властивості і методи. Іншими словами, властивості об'єктів з однаковою специфікацією(наслідуванням) можуть мати різну реалізацію.

Іншим об'єктам при взаємодії не важливо якого класу об'єкт(батько або спадкоємець) з яким вони мають взаємодію. Важливо, що він має той інтерфейс, який від нього чекають.

Також ключове слово **super** повертає властивості з батьківського класу. Ми самі визначаємо, які властивості потрібні спадкоємцеві, а які ні.

6.3 Приклад роботи з класами

Створити наступні класи: людина, учень, поганий учень, учитель, директор.

Кожна людина має: прізвище, ім’я, по батькові, рік народження. Наслідування визначено відповідно здоровому глузду (поганий учень – нащадок учня). Всі сутності мають наступні методи:

- підрахувати вік (getAges)
- звернутися за ім’ям (getName) за правилом: вчитель і директор – ім’я, по батькові; учень - ім’я; поганий учень – «поганий» + ім’я.
- булевий метод (head?): для директора повертає істину, для інших – хибність.

ПІБ має задаватися у конструкторі.

Після реалізації створити екземпляри кожного класу, та визвати для них методи getAges, getName, head?

Створимо клас людина:

```
class Person
attr_accessor: :first_name, :last_name, :middle_name, birthday
end
```

Приклад створення об’єкта:

```
p = Person.new(“Іванов”, “Іван”, “Іванович”, “1975”)
```

Нам необхідно щоб працював наступний конструктор:

```
class Person
attr_accessor: :first_name, :last_name, :middle_name, birthday
def initialize (fname, lname, mname, birthday)
   @first_name = fname
   @last_name = lname
   @middle_name = mname
   @birthday = birthday
```

```
  end
end
```

Визначимо метод для задання віку:

```
class Person
  …
 def age
  2016 - @birthday
 end
end
```

Оскільки, в більшості випадків метод head? повертає false, визначимо його в класі Людина, а за допомогою поліморфізму перевизначимо його в класі Директор.

```
class Person
  …
 def head?
  false
 end
end
```

Стандартне звертання до людини: імя, по батькові – отже визначимо його у класі Людина:

```
class Person
  …
 def name
  @first_name + " " + @middle_name
 end
end
```

Перевіримо клас Person, визвавши всі його методи:

```
p = Person.new("Іванов", "Іван", "Іванович", "1975")
puts p.name
puts p.age
puts p.head?
```

Оскільки клас Teacher нічим не відрізняється від класу Person, достатньо його перевизначити:

```
class Teacher < Person
end
```

Скористаємось поліморфізмом для перевизначення методу name класу Student:

```
class Student < Person
```

```
 def name
  @first_name
  end
 end
```

Для класу BadStudent також необхідно перевизначити цей метод:

```
class BadStudent < Person
 def name
  "Поганий" + @first_name
  end
 end
```

Далі необхідно перевизначити метод head? для класу Headmaster:

```
class Headmaster < Person
 def head?
  true
  end
 end
```

6.4 Завдання до роботи

6.4.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

6.4.2 Розробити ієрархію класів, використовуючи основні парадигми програмування (продемонструвати роботу із сеттерами/геттерами, успадкування, поліморфізм, інкапсуляцію, роботу з модулями, перевизначення методів (із зверненням до оригінальних методів), роботу із конструкторами, перевизначення синглтон-методів).

Варіанти завдань (при необхідності додати потрібні сутності):

1. Транспортні засоби: розробити ієрархію для різних видів транспорту (автомобіль, велосипед, поїзд, літак), додати особливі методи для кожного типу (наприклад, завести двигун).
2. Тварини: створити класи для різних видів тварин (ссавці, птахи, риби) з особливими методами для їх поведінки (полетіти, пірнути тощо).
3. Музичні інструменти: ієрархія для інструментів (струнні, духові, ударні), кожен тип має методи для гри певного типу звуку.

4. Спортивні дисципліни: розробити ієрархію для видів спорту (командні, індивідуальні), додати різні види спортивного інвентаря.
5. Ігри: створити класи для різних жанрів ігор (настільні, комп'ютерні, мобільні), методи для правил гри та рівнів.
6. Геометричні фігури: створити класи для фігур (коло, квадрат, трикутник) з методами для обчислення площі та периметру.
7. Побутова техніка: ієрархія для різних видів техніки (пральна машина, пилосос, мікрохвильовка), методи для увімкнення та встановлення режимів.
8. Електронні пристрої: розробити класи для пристроїв (телефон, планшет, ноутбук), додати методи для заряджання батареї.
9. Рослини: створити класи для різних видів рослин (дерева, квіти), методи для росту, цвітіння.
10. Меблі: створити ієрархію для меблів (стіл, стілець, ліжко), додати методи для складання і розкладання.
11. Професії: класи для різних професій (лікар, вчитель, інженер), методи для роботи та специфічних задач кожної професії.
12. Космічні об'єкти: розробити класи для об'єктів (планета, зірка, супутник), методи для обчислення орбіт, обертання.
13. Фінансові інструменти: ієрархія для різних інструментів (акції, облігації, криптовалюти), методи для розрахунку вартості.
14. Страви: ієрархія для видів страв (супи, десерти, гарніри), методи для приготування та подачі.
15. Користувачі соціальних мереж: класи для ролей користувачів (адміністратор, модератор, користувач), методи для створення, редагування та видалення постів.
16. Транспортні квитки: розробити класи для різних квитків (авіаквиток, залізничний квиток, автобусний квиток), методи для перевірки квитка та реєстрації.
17. Будівлі: класи для різних типів будівель (житлові, офісні, громадські), методи для обчислення поверховості, опалення.

18. Інтернет-магазини: класи для елементів магазину (користувачі, товари, замовлення), методи для покупки та обробки замовлення.
19. Підприємства: розробити ієрархію для різних типів підприємств (заводи, магазини), методи для роботи персоналу.
20. Кораблі: ієрархія для морських транспортних засобів (підводний човен, лайнер, баржа), методи для занурення та руху.
21. Домашні тварини: класи для різних тварин (собаки, коти), методи для догляду та годування.
22. Додатки на телефоні: розробити класи для додатків (соціальні мережі, банківські додатки, гри), методи для встановлення та оновлення.
23. Кухонні прилади: класи для приладів (блендер, кавоварка, мультиварка), методи для роботи кожного приладу.
24. Фермерські продукти: розробити ієрархію для продуктів (молочні, м'ясні, зернові), методи для зберігання та обробки.
25. Арт-об'єкти: створити класи для об'єктів (скульптури, картини, фотографії), методи для виставлення та зберігання.
26. Люди в команді: класи для ролей в команді (лідер, розробник, дизайнер), методи для командної роботи та координації.
27. Види товарів: ієрархія для товарів у магазині (продукти, одяг, електроніка), методи для обліку та продажу.
28. Парковий транспорт: розробити ієрархію для транспорту в парку (потяг, велосипед, електросамокат), методи для оренди.
29. Історичні пам'ятки: створити класи для пам'яток (замки, музеї, монументи), методи для екскурсій та обслуговування.
30. Навчальні предмети: ієрархія для предметів у школі (математика, фізика, література), методи для викладання та оцінювання.

6.5 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.3 Код розробленої програми.

3.4.4 Результати роботи програми.

3.4.5 Висновки, що відображають результати виконання роботи та їх критичний аналіз.

6.6 Контрольні питання

1. Як визначити та використовувати конструктор у класі?
2. Що таке інкапсуляція і як вона реалізується в ruby?
3. Як працюють сеттери та геттери в ruby?
4. У чому різниця між attr_reader, attr_writer та attr_accessor?
5. Як реалізувати успадкування класів у ruby?
6. Що таке поліморфізм і як його можна продемонструвати у ruby?
7. Як працює метод super і для чого він використовується?
8. Як перевизначити метод у дочірньому класі та звернутися до методу батьківського класу?
9. Що таке синглтон-метод? Як його створити для окремого об’єкта?
10. Що таке модуль у ruby?
11. Як підключити модуль до класу? Що таке включення (include) і розширення (extend) модуля?
12. Як в ruby організувати ієрархію класів для різних типів об'єктів?
13. Як в ruby визначити метод як приватний або захищений? У чому різниця між ними?
14. Які особливості має ключове слово self в ruby? Як воно використовується?
15. Як створити статичний метод у класі? У чому відмінність від звичайних методів?
16. Як в ruby можна створити константи і де їх краще використовувати?
17. Як викликати метод, якщо об'єкт класу зберігається в змінній?
18. Що таке перевантаження методів і як цього досягти в ruby?

Лабораторна робота № 7

Тема: створення простих веб-застосувань на основі Ruby on Rails

Мета роботи – засвоєння теоретичних відомостей про принципи проектування Model-View-Controller і отримання практичних навичок створення веб-застосування з використанням засобів Ruby on Rails, побудови простих форм.

7.1 Основні теоретичні відомості

Ruby on Rails — об'єктно-орієнтований програмний каркас (фреймворк) для створення веб-додатків, написаний на мові програмування Ruby.

Ідеологія RubyOnRails реалізує концепцію модель-представлення-контролер (Model-View-Controller, MVC). Прийнято вважати, що MVC була описана в 1979 р. Трюгве Рінскаугом (Trygve Reenskaug), що працював тоді над мовою програмування Smalltalk в Xerox PARC.

У цій концепції модель відповідальна за збереження стану застосування. У одних випадках стани є перехідними, єдине призначення яких – забезпечити взаємодію з користувачем, в інших випадках стани постійні і зберігаються поза застосуванням, наприклад в системі управління базами даних (СУБД).

Модель (*Model*) надає решті компонентів програми об'єктно-орієнтоване представлення даних (таких як каталог продуктів або список замовлень). Об'єкти моделі здійснюють завантаження і збереження даних в реляційній базі даних. Наприклад, не може бути надана знижка на замовлення менш ніж 1000грн., і модель повинна містити це обмеження. Таким чином, завдання моделі – забезпечити цілісність даних за допомогою застосування певних правил (обмежень).

Завдяки можливостям динамічної типізації в мові Ruby розробникові досить успадкувати свій клас моделі від базового класу ActiveRecord::Base. Ruby on Rails автоматично пов'язує класи моделі з таблицями в базі даних і створює атрибути об'єктів для відповідних полів таблиці..

Представлення (*View*) відповідальні за формування інтерфейсу користувача, заснованого на даних, що отримуються з моделі. Представлення створює інтерфейс користувача для відображення отриманих від контролера даних, також передає запити користувача на маніпуляцію даними в контролер (як правило, вид не змінює безпосередньо дані з моделі).

У Ruby on Rails представлення описується за допомогою шаблонів RHTML. Вони є файлами HTML з додатковими включеннями фрагментів коду Ruby (Embedded Ruby або ERb). Вивід, згенерований вбудованим кодом Ruby, включається в текст шаблону сторінки HTML, яка після цього повертається користувачеві. Представлення можуть використовувати фрагменти інших представлень і, у свою чергу, бути включеними в шаблон (layout) вищого рівня.

Наприклад, сайт магазину має список продуктів, що відображається на екрані монітора. Цей список буде отриманий через модель саме представленням, яке передасть його у відповідному форматі кінцевому користувачеві. Також представлення може пропонувати користувачеві різні способи введення даних, але воно ніколи не займається їх обробкою. Представлення завершує роботу, як тільки дані передані користувачеві.

Можлива ситуація, коли декілька представлень дозволяють отримати доступ до одних і тих же даних, але з різними методами. Наприклад, на сайті інтернет-магазину товари повинні відображатися по-різному, залежно від того, хто зайшов на сайт : користувач (покупець) для оформлення замовлення або адміністратор для редагування товарів.

Контролери (*Controller*) зв'язують окремі елементи застосування. Вони отримують події із зовнішнього світу (наприклад, команди користувача), взаємодіють з моделями і активують відповідне представлення для користувача. Контролер прочитує необхідні дані з моделі і готує їх для відображення, а також зберігає отримані від відображення дані в моделі.

Контролером в Ruby on Rails є клас, успадкований від ActionController::Base. Відкриті методи контролера є так званими діями (actions). Action часто відповідає

окремому представленню. Наприклад, по запиту користувача admin/list буде викликаний метод list класу AdminController і потім використаний вид list.rhtml.

На рисунку 7.1 представлена описана схема взаємодії, тобто концепція MVC.

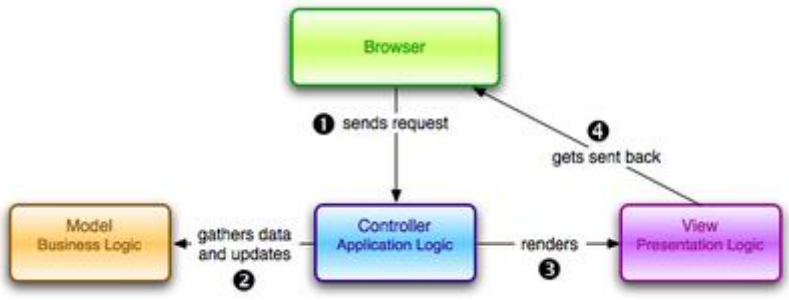


Рисунок 7.1 - Концепція MVC :

1 – браузер відправляє запит; 2 – контролер взаємодіє з моделлю; 3 – контролер підключає представлення; 4 – засобами відображення формується нова сторінка для браузеру

7.2 Завдання для виконання

7.2.1 Створення простої аплікації

Всі вказівки даної лабораторної роботи базуються на тому, що у вас є встановлено Ruby on Rails останньої версії, що повинно бути виконано в лабораторній №1

Розглянемо створення простого застосування (аплікації). Rail має засоби для автоматичної генерації базової структури застосування. Для того, щоб створити каркас застосування, необхідно в консолі перейти в директорію, в якій має бути розміщене застосування, і виконати команду

```
rails new Lab7RailsApp
```

В результаті буде створена директорія з ім’ям Lab7RailsApp, в якій формується структура застосування. При цьому в консоль видаються повідомлення. Далі наведені їх приклади.

Створення застосування :

```
create
```

Створення шаблону короткої інформації про застосування:

```
create README.rdoc
```

Створення файлу для системи зборки Rake (аналог команди **make** для Ruby) :

```
create Rakefile
```

Допоміжні файли:

```
create config.txt
create .gitignore
```

Файл, який містить опис необхідних застосуванню gem –пакетів :

```
create Gemfile
```

Створення основного каркаса – контролери, представлення, моделі, помічники (helper), шаблони типового відображення :

```
create     app
create     app/assets/images/rails.png
create     app/assets/javascripts/application.js
create     app/assets/stylesheets/application.css
create     app/controllers/application controller.rb
create     app/helpers/application helper.rb
create     app/mailers
create     app/models
create     app/views/layouts/application.html.erb
```

Створення конфігураційної частини застосування, яка включає маршрутизацію, налаштування запуску, локали, параметри підключення до баз даних :

```
create     config/routes.rb
create     config/application.rb
create     config/environment.rb
create     config/environments/development.rb
create     config/environments/production.rb
create     config/environments/test.rb
create     config/initializers/backtrace silencers.rb
create     config/initializers/inflections.rb
create     config/initializers/mime_types.rb
```

```
create     config/initializers/secret_token.rb
create     config/initializers/session_store.rb
create     config/initializers/wrap_parameters.rb
create     config/locales/en.yml
create     config/boot.rb
create     config/database.yml
```

Створення директорії для бази даних :

```
create db
create db/seeds.rb
```

Директорія для майбутньої документації:

```
create doc
create doc/README_FOR_APP
```

Створення директорій для бібліотек функцій :

```
create lib create lib/tasks create lib/assets
```

Директорія для журналів виконання застосування :

```
create log
```

Директорія для загальнодоступних статичних файлів:

```
create     public
create     public/4 04.html
create     public/422.html
create     public/500.html
create     public/favicon.ico
create     public/index.html
create     public/robots.txt
```

Директорія службових скриптів Rails :

```
create     script
create     script/rails
```

Директорія тестів застосування :

```
create     test/fixtures
create     test/functional
create     test/integration
create     test/unit
create     test/performance/browsing test.rb
create     test/test_helper.rb
```

Тимчасова директорія для службових цілей, вона ж для зберігання кеша :

```
create     tmp/cache
create     tmp/cache/assets
```

Директорії для розміщення додаткових модулів :

```
create     vendor/assets/javascripts
create     vendor/assets/stylesheets
create     vendor/plugins
```

Далі по команді **bundle install** автоматично запускається спеціальний менеджер пакетів bundle, який перевіряє склад наявних gem-пакетів і встановлює безпосередньо з вказаного у файлі Gemfile сайту усі необхідні пакети. Приведемо невеликий фрагмент цього виводу :

```
Using rake 13.0.6
Using concurrent-ruby 1.1.10
Using i18n 1.12.0
Using minitest 5.16.3
Using tzinfo 2.0.5
 ...
Using turbo-rails 1.1.1
Using web-console 4.2.0
Using webdrivers 5.0.0
 Bundle complete! 16 Gemfile dependencies, 75 gems now installed.
```

Таким чином, після виконання команд **rails new Lab7RailsApp** і **run bundle install** в мінімальному виді каркас застосування створений, а необхідні пакети встановлені. Перейдемо в директорію застосування і запустимо його командою **rails server** або псевдонім **rails s.** Отримаємо приблизно наступне повідомлення:

```
=> Booting Puma
=> Rails 7.0.4 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 5.6.5 (ruby 3.1.2-p20) ("Birdie's Version")
*  Min threads: 5
*  Max threads: 5
*  Environment: development
*          PID: 5091
* Listening on http://127.0.0.1:3000
* Listening on http://[::1]:3000
Use Ctrl-C to stop
```

де **Puma** – налагоджувальний веб-сервер, який вбудований в Rails. Його використовуватимемо всякий раз, коли захочемо побачити свою роботу у веб браузері, тобто для роботи в цілях відладки.

На відміну від веб-серверів, які служать для експлуатації застосувань, цей веб-сервер дозволяє вносити зміни в код застосування і бачити зміни без його перезапуску. У приведеному повідомленні також вказані версії Rails, Ruby і рядок http://127.0.0.1:3000. Цей рядок означає, що веб-сервер приєднався на фіктивний інтерфейс 127.0.0.1, який готовий приймати запити, що поступають з усіх наявних мережевих адаптерів, а значення :3000 – це ip-порт.

Відкриємо браузер, введемо адресу http://localhost:3000 і отримаємо сторінку, приведену на рисунку 7.3. Вона є сторінкою за умовчанням rails/templates/rails/welcome/index.html.erb, яка згенерована у момент створення застосування.

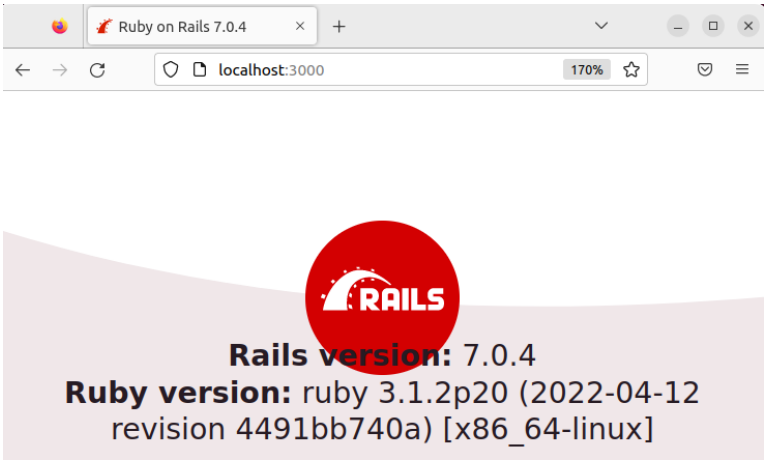


Рисунок 7.3. Сторінка http://localhost:3000

Після того, як браузер відобразить сторінку в консолі, в якій запущений сервер, побачимо приблизно наступний текст:

```
    Started GET "/" for 127.0.0.1 at 2022-10-26 20:45:34 +0300
    Processing by Rails::WelcomeController#index as HTML
      Rendering          /usr/share/rvm/gems/ruby-3.1.2/gems/railties-
7.0.4/lib/rails/templates/rails/welcome/index.html.erb
```

```
      Rendered           /usr/share/rvm/gems/ruby-3.1.2/gems/railties-
7.0.4/lib/rails/templates/rails/welcome/index.html.erb      (Duration:
8.6ms | Allocations: 526)
    Completed 200 OK in 39ms (Views: 14.4ms | ActiveRecord: 0.0ms |
Allocations: 3752)
```

З цього тексту виходить, що до сервера звернулися з локальної адреси 127.0.0.1 і запросили методом GET згенерований файл index.html.erb. В результаті виконання запиту було здійснено підключення до бази даних, вказаної в database.yml, а сторінка успішно повернена клієнтові (код 200 OK). Веб-сервер можна зупинити натисненням комбінації Ctrl-C.

7.2.3 Hello Rails!

Доброю традицією стало "`Hello World`" як приклад роботи будь-якої мови або движка. Що ж, давайте зробимо сторіночку, яка вітатиме нас. Ми знаємо, що застосування на RoR повинне складатися з трьох компонентів: Виду, Контролера і Моделі. Сказати привіт - це просто, досить передати браузеру HTML -код. Нескладно здогадатися, що HTML повинен міститися у Виді, проте RoR не дозволяє створювати види безпосередньо, оскільки Вид має бути безпосередньо асоційований з Контролером. Щоб згенерувати що-небудь в RoR, треба йти в командний рядок (ми вже знаємо, як це робиться) і перемістити в теку застосування. Тут ми і виконаємо:

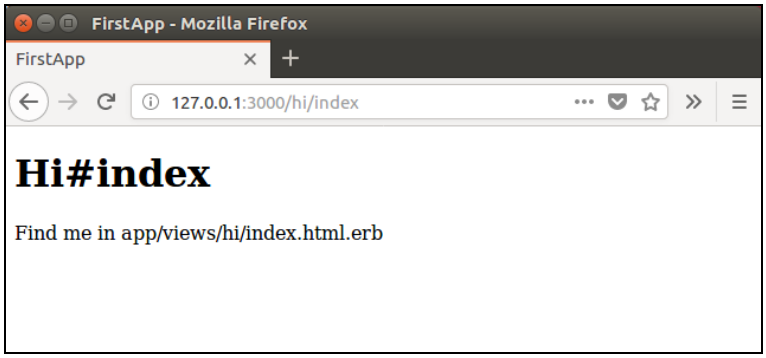
```
rails generate controller Hi index
```

`generate` - це Shell-скрипт generate в теці `script`, написаний на Ruby, тому запускати скрипт треба через інтерпретатор (для цього ми додали `rails` на початку команди). Так само ми передали скрипту два аргументи: `controller` визначає, що необхідно згенерувати код Контролера, в даному випадку він буде називатися `Hi` - це говорить другий аргумент. А `index` у кінці команди прив'яже до контролера Вид `index`, що, власне, нам і вимагалося. Трохи почекаємо, і ось що видасть нам скрипт:

```
ihor@ihor-virtual-machine:~/RubyPrj/Demo0$ rails generate controller Hi index
      create  app/controllers/hi_controller.rb
       route  get 'hi/index'
      invoke  erb
      create    app/views/hi
      create    app/views/hi/index.html.erb
      invoke  test_unit
      create    test/controllers/hi_controller_test.rb
      invoke  helper
      create    app/helpers/hi_helper.rb
      invoke    test_unit
ihor@ihor-virtual-machine:~/RubyPrj/Demo0$
```

Знаючи про структуру тек, ми вже можемо сказати, що були створені: тека `hi` у Видах, Контролер, шаблон для створення тестів, хелпер і файл Виду `index.html.erb` (формат `.html.erb` - це поєднання HTML і коду Ruby).

Тепер `index` став доступний для застосування. Давайте запустимо сервер і заглянемо в http://127.0.0.1: 3000/hi/index:



Небагатослівно, але сторіночка підказує нам, що файл знаходиться в `app/views/hi/index.html.erb`. Відкривши його, ви помітите, що в коді навіть немає базової структури HTML. Давайте трохи доповнимо файл і приведемо його в порядніший вид:

```
<html>
<head><title>Hi PNU Student!</title></head>
<body>
<h1>Hello!</h1>
<p>Це вітання прибуло з app/views/hi/index.html.erb</p>
</body>
</html>
```

Збережемо файл і відновимо сторіночку:

Напевно, усі ці генерації безлічі тек виглядають досить сумнівно для того, щоб створити звичайну HTML сторіночку. Для того, щоб почати використати силу RoR, попрацюємо з Контролером `app/controllers/hi_controller.rb`. Ось, що в ньому знаходиться:

```
class HiController < ApplicationController
def index
end

end
```

Ми з вами вже знаємо Ruby, тому кодом нас не налякаєш: оголошені клас `HiController` (помітимо, що назва класу була згенерована на основі імені Контролера — це RoR і ми ще говоритимемо чому і навіщо), який є нащадком класу `ApplicationController` і порожній метод `index`. Давате пофантазуємо з методом, задаючи в нім змінні і відображаючи їх у Виде. Для цього в RoR використовуються змінні екземпляра (ті самі, який починаються з `@`):

```
class HiController < ApplicationController
def index
     @stud = 'Student'
     @message = 'Моє повідомлення прийшло з Контролера'
end
end
```

Відповідно відредагуємо Вид, щоб показати змінні:

```
<html>
<head><title>Hi <%= @stud %>!</title></head>
<body>
<h1>Hello, <%= @stud %>!</h1>
<p>Це повідомлення прибуло з app/views/hi/index.html.erb</p>
<p><%= @message %></p>
</body>
</html>
```

І тут ми бачимо, як код Ruby впроваджується в HTML. Для виведення значення змінної в HTML -код ми використовуємо теги `<%= . %>` - цей так званий вираз (`expression`).

Для того, щоб отримати поточний час Ruby використовується метод `Time.now`. Досить вставити в HTML `<%= Time.now %>`, проте варто використати переваги MVC і робити розрахунки в Контролері. Спробуйте "рахувати" час в Контролері і відобразити його на сторінці Виду.

Логіка у Виді

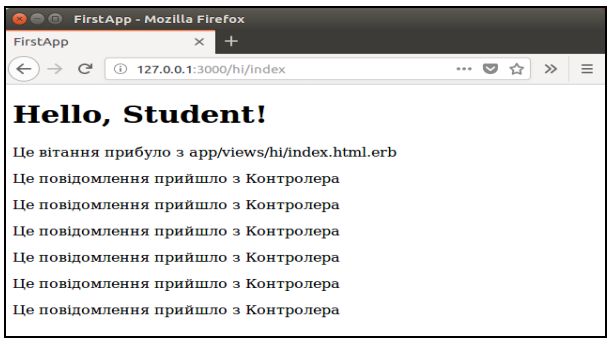
Ми також можемо додавати деяку логіку **у файли Виду**, наприклад, щоб створювати списки, ми можемо використати ітератори. Припустимо, ми хочемо вивести змінну `@message` п'ять разів в параграфах. Ми знаємо, як зробити це на чистому Ruby:

```
5.times do
puts "<p> #{@message} </p>"
end
```

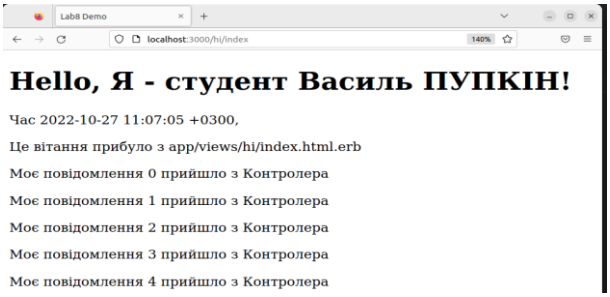
Залишилося тільки переробити код у формат `.erb`, виходить навіть дещо простіше:

```
<% 5.times do%>
<p><%= @message %></p>
<% end %>
```

`puts` замінили теги `<%= . %>`. А коли виведення коду нам не потрібне ми просто опускаємо = в тегах. Ось що у нас вийшло



Використовуючи вищенаведені вказівки реалізувати підрахунок повідомлень (у Виді), результат повинен бути такий (реалізувати це самостійно):



Як це працює?

Коли запускається код, RoR інтерпретує запит `127.0.0.1:3000/hi/index` як виклик Контролера `Hi`. У RoR є редагований список правил роутинга запитів, за умовчанням перша частина запиту — ім'я Контролера, друга — метод в ньому. Метод визначає необхідні базові змінні. На цьому робота Контролера закінчена і RoR передає дані у Вид. Як він знає, в який Вид треба передати дані? Працюють `conventions` — магія домовленостей про іменування.

4.2.3 Створення простого CRUD – додатку з Rails

У рамках знайомства з Rails використовуємо генератор типових контролерів, так звань Rails Scaffold (каркас). Scaffold — це вбудований генератор, сам по собі він нічого не генерує, але запускає інші генератори. Scaffold в Rails — це повний набір з моделі, міграції бази даних для цієї моделі, контролер для дії на неї, вьюхи для перегляду і поведінки з даними і тестовий набір для усього цього.

Виконаємо команду для запуску Scaffold -генератора:

```
rails generate scaffold User name:string email:string
```

У результаті отримаємо набір повідомлень. Далі наведені їх приклади.

Створення специфікації бази даних :

```
  invoke  active_record
  create    db/migrate/20221026181412_create_users.rb
```

Створення еквівалентної моделі :

```
create app/models/user.rb
```

Створення Unit -тестів застосування :

```
invoke     test unit
create     test/unit/user test.rb
create     test/fixtures/users.yml
```

Додавання нового маршруту :

```
invoke resource route
route resources: users
```

Створення нового контролера :

```
invoke scaffold controller
create app/controllers/users controller.rb
```

Генерація представлень :

```
invoke     erb
create     app/views/users
create     app/views/users/index.html.erb
create     app/views/users/edit.html.erb
create     app/views/users/show.html.erb
create     app/views/users/new.html.erb
create     app/views/users/_form.html.erb
```

Створення функціонального теста контролера :

```
invoke     test unit
create     test/functional/users controller test.rb
```

Створення помічника (helper) :

```
invoke     helper
create     app/helpers/users helper.rb
```

Створення тесту для помічника:

```
invoke     test unit
create     test/unit/helpers/users helper test.rb
```

Створення «корисностей»:

```
      jbuilder
      create      app/views/users/index.json.jbuilder
      create      app/views/users/show.json.jbuilder
      create      app/views/users/_user.json.jbuilder
```

Для запуску реального створення бази даних виконаємо команду

```
rake db:migrate
```

В результаті буде отримана база даних /db/development. sqlite3 (задана в конфігурації) по файлу міграції db/migrate/20221026181412_create_users.rb Вміст цього файлу :

```
class CreateUsers < ActiveRecord::Migration[7.0]
  def change
    create_table :users do |t|
      t.string :name
      t.string :email
      t.timestamps
    end
  end
end
```

Тут виділені рядки відповідають полям, які були вказані при запуску Scaffold-генератора. При цьому буде створена таблиця users, а кожен запис міститиме name, email і timestamp. Зверніть увагу на те, що в команді запуску Scaffold-генератора було використане ім'я User, а таблиця називається users. Відмінність в написанні цих імен полягає в прийнятій угоді про імена. У момент запуску Scaffold-генератора вказується ім'я сутності в однині, а ім'я контролера і ім'я таблиці – в множині.

Після створення бази даних можна запустити вебсервер командою **rails s.** Введемо у браузер адресу http://localhost:3000/users/, де users – ім'я створеного контролера, і отримаємо сторінку, приведену на Рисунку 7.4.

Натиснемо New User, і на екрані буде відображена форма додавання нового користувача з полями name, email, заданими при запуску Scaffold -генератора. Додамо по черзі декількох користувачів (Рисунки 7.4 і 7.5).

Повторно введемо у браузер адресу http://localhost:3000/users/ і отримаємо сторінку, представлену на рисунку 7.6.

Рисунок 7.4 - Сторінка http://localhost:3000/users/new

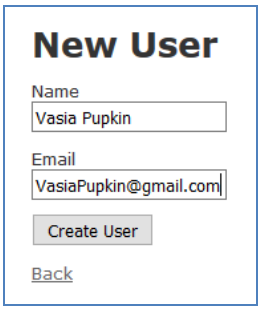


Рисунок 7.5 - Сторінка http://localhost:3000/users/1

Таким чином, після введення команди Scaffold -генератора можемо переконатися, що згенероване застосування дозволяє виконувати функції перегляду, додавання, редагування і видалення користувачів. При цьому слід звернути увагу на те, що увесь необхідний код був сформований автоматично.

Далі розглянемо інші генератори і на прикладі згенерованого застосування розберемо призначення компонентів Rails.

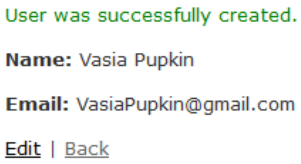


Рисунок 7.6 - Результат створення користувача

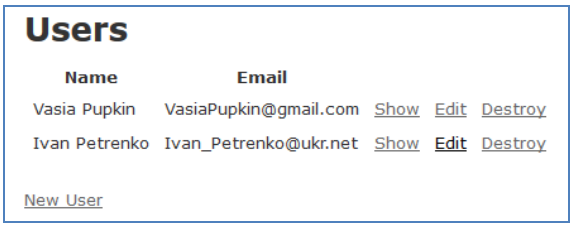


Рисунок 7.7 - Сторінка http://localhost:3000/users/

Внесемо доповнення в схему взаємодії компонентів по концепції MVC з урахуванням реального згенерованого застосування Rails (Рисунок 7.8).

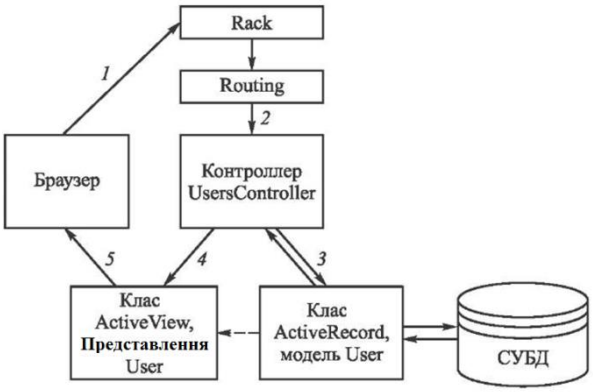


Рисунок 7.8. Схема взаємодії компонентів MVC на прикладі Rails -додатку:

1 – браузер відправляє запит http://localhost: 3000/users/1; 2 – маршрутизатор знаходить контоллер users; 3 – контролер UsersController взаємодіє з моделлю User; 4 – контролер підключає представлення User; 5 – засобами відображення формується нова сторінка для браузеру. Штриховою стрілкою позначена логічна передача даних. Взаємодія з СУБД являється опціональною

Принципових відмінностей на цій схемі від загальної концепції MVC мало. Rails-додаток контролює компонент з назвою Rack, який є посередником в передачі запитів користувачів від веб-сервера до веб-застосування. У документації на http://rack.rubyforge.org/doc наводяться наступні веб-сервери, з якими Rack може працювати:

```
Mongrel;    - WEBrick;      - SCGI;
EventedMongrel;       - Puma;    - LiteSpeed;
SwiftipliedMongrel;  - CGI;     - Thin.
```

З використанням компонента Rack побудовані декілька бібліотек для створення веб-застосувань :

```
Camping;    - Racktools::SimpleApplication;    - Vintage;
Coset;          - Ramaze; - Waves;
Halcyon;    - Ruby on Rails;    - Wee;
Mack;       - Rum;     ...
Maveric;    - Sinatra;
Merb;       - Sin;
```

Велика їх частина має досить обмежене застосування. У контексті Rails-додатку необхідно знати, що запит від Rack передається компоненту маршрутизації. Початковий код, який контролює маршрутизацію, розташований у

файлі config/routes.rb. Спочатку цей файл містить лише каркас для додавання маршрутів :

```
TestApp::Application.routes.draw do
…
end
```

Проте після завершення роботи Scaffold-генератора в нього був доданий рядок:

```
resources: users
```

де resources – створення стандартного набору REST-маршрутів, який створить 4 іменовані маршрути (у концепції CRUD – Create Read Update Destroy) і 7 дій : index, show, new, create, edit, update, destroy.

Маршрути виділяються з URL. Наприклад, адреси http://localhost: 3000/users/ і http://localhost:3000/users/new явно ведуть на маршруту UserController#index, UserController#new, а адреса http://localhost: 3000/users/1 – неявно на маршрут UserController#show, але з параметром :id.

Маршрути вказують на конкретні контролери, які повинні обробляти запити. В даному випадку існує єдиний контролер app/controllers/application_controller.rb. Після автоматичної генерації він містить наступний код:

```
class UsersController < ApplicationController
  before_action :set_user, only: [:show, :edit, :update, :destroy]

  # GET /users
  # GET /users.json
  def index
    @users = User.all
  end

  # GET /users/1
  # GET /users/1.json
  def show
  end

  # GET /users/new
  def new
    @user = User.new
  end
```

```
      # GET /users/1/edit
      def edit
      end

      # POST /users
      # POST /users.json
      def create
        @user = User.new(user_params)

        respond_to do |format|
          if @user.save
            format.html { redirect_to @user, notice: 'User was
successfully created.' }
            format.json { render :show, status: :created, location:
@user }
          else
            format.html { render :new }
            format.json { render json: @user.errors, status:
:unprocessable_entity }
          end
        end
      end

      # PATCH/PUT /users/1
      # PATCH/PUT /users/1.json
      def update
        respond_to do |format|
          if @user.update(user_params)
            format.html { redirect_to @user, notice: 'User was
successfully updated.' }
            format.json { render :show, status: :ok, location: @user
}
          else
            format.html { render :edit }
            format.json { render json: @user.errors, status:
:unprocessable_entity }
          end
        end
      end

      # DELETE /users/1
      # DELETE /users/1.json
      def destroy
        @user.destroy
        respond_to do |format|
          format.html { redirect_to users_url, notice: 'User was
successfully destroyed.' }
          format.json { head :no_content }
        end
      end
      private
        def set_user
          @user = User.find(params[:id])
```

```
    end

    def user_params
      params.require(:user).permit(:name, :email)
    end
end
```

Клас UsersController є нащадком Rails-класу ApplicationController.

Зверніть увагу на те, що в коментарях перед методами згенерованого класу UsersController вказані маршрути, по яких передбачається виклик цих методів. Розглянемо один з маршрутів і, відповідно, один метод контролера, наприклад, index:

```
def index
  @users = User.all
end
```

Тут перший рядок – маршрут отримання списку користувачів через клас User; цей клас є моделлю користувачів; по імені методу, що викликається .all можна припустити, що буде отриманий список усіх користувачів. Для подальшого використання цього списку введена змінна рівня екземпляра з ім'ям @users.

Файл моделі app/models/user.rb містить наступний код:

```
class User < ActiveRecord::Base
  attr_accessible :email, :name
end
```

З цього коду виходить, що клас User є нащадком Rails-класу ActiveRecord::Base. Більше того, в класі оголошений два атрибути:: email і :name, тому в екземплярах цього класу до них можна безпосередньо звернутися.

Повернемося до контролера і методу index, де метод respond_to призначений для надання можливості отримання відповіді від веб-сервера в різних форматах. Проте явно метод може і не використовуватися, якщо формат отримання єдиний. Тут же передбачений два формати: html і json. Для перевірки цього введемо у браузері адресу: **http://localhost:**3000/users.json. В результаті отримаємо відповідь у форматі json -типу :

```
    [{"id":1,"name":"Vasia
Pupkin","email":"VasiaPupkin@gmail.com","created_at":"2022-10-
```

```
10T06:13:34.085Z","updated_at":"2022-10-
10T06:13:34.085Z","url":"http://localhost:3000/users/1.json"},{"id":2
,"name":"Ivan
Petrenko","email":"Ivan_Petrenko@ukr.net","created_at":"2022-10-
10T06:18:48.797Z","updated_at":"2022-10-
10T06:18:48.797Z","url":"http://localhost:3000/users/2.json"}]
```

У разі format.html передбачається використання представлення index.html.erb. Зверніть увагу на те, що в приведеному раніше коді методу index запис "**#index.html.erb**" – усього лише коментар, що пояснює, звідки буде використаний представлення.

В процесі виконання Scaffold-генератора представлення були згенеровані для кожного маршруту, проте зараз потрібний лише файл app/views/users/index.html.erb. Цей файл містить наступний код:

```
<p style="color: green"><%= notice %></p>
<h1>Users</h1>
<div id="users">
  <% @users.each do |user| %>
    <%= render user %>
    <p>
      <%= link_to "Show this user", user %>
    </p>
  <% end %>
</div>
<%= link_to "New user", new_user_path %>
```

У коді є присутня HTML-розмітка, спеціальна розмітка у формі <% %> і розмітка <%= ... %>, яка містить виклики Ruby-коду і спеціальних методів типу link_to. Зверніть увагу на те, що для виведення списку користувачів формується HTML-таблиця. Проте число користувачів не відоме, тому в коді використаний цикл генерації рядків :

```
<% @users.each do |user| %>
  <%= render user %>
  <p>
    <%= link_to "Show this user", user %>
  </p>
<% end %>
```

Підстановка імені користувача і його електронної адреси здійснюється через автоматично згенерований парціальний (partials) шаблон _user.html.erb, який містить такий код:

```
<div id="<%= dom_id user %>">
  <p>
    <strong>Name:</strong>
    <%= user.name %>
  </p>

  <p>
    <strong>Email:</strong>
    <%= user.email %>
  </p>
</div>
```

Слід також мати на увазі, що цей шаблон не містить усієї необхідної HTML-розмітки. Шаблон відображення сторінок застосування був згенерований на найпершому етапі генерації Rails -додатку при виконанні Scaffold-генератора і зберігається у файлі app/views/layouts/application.html.erb. Його вміст представлений нижче:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Demo0</title>
    <meta   name="viewport"   content="width=device-width,initial-
scale=1">
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%=  stylesheet_link_tag  "application",  "data-turbo-track":
"reload" %>
    <%= javascript_importmap_tags %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Тут міститься зовнішнє обрамлення, а вставка конкретних шаблонів представлень виконується замість рядка <%= yield %>.

Завершимо дослідження Rails-додатку аналізом згенерованих тестів. Відмітимо, що за умовчанням в Rails генеруються тести вже застарілої системи тестування Unit. Для використання системи тестування rspec необхідно здійснити додаткові дії. У цих вказівках розглянемо Unit -тести.

Для запуску тестів досить виконати в директорії Rails-застосування команду **rake.** Файл Rakefile містить мету test в якості мети за умовчанням. Додатково вкажемо трасування викликів і запустимо **rake --trace**. Наведемо приклад отриманого повідомлення :

```
** Invoke default (first_time)
** Invoke test (first_time)
** Execute test
** Execute default
Run options: --seed 55155
# Running:
.......
Finished in 17.299634s, 0.4046 runs/s, 0.5202 assertions/s.
7 runs, 9 assertions, 0 failures, 0 errors, 0 skips
```

З цього повідомлення виходить, що було запущено 7 тестів, прийняті 9 тверджень, помилок не виявлено.

Велика частина тестів нічого не містить. Проте є згенерований функціональний тест у файлі \test\controllers\users_controller_test.rb, який містить наступний текст:

```
require 'test_helper'

class UsersControllerTest < ActionDispatch::IntegrationTest
  setup do
    @user = users(:one)
  end

  test "should get index" do
    get users_url
    assert_response :success
  end

  test "should get new" do
    get new_user_url
    assert_response :success
  end
```

```
      test "should create user" do
        assert_difference('User.count') do
          post users_url, params: { user: { email: @user.email, name:
@user.name } }
        end

        assert_redirected_to user_url(User.last)
      end

      test "should show user" do
        get user_url(@user)
        assert_response :success
      end

      test "should get edit" do
        get edit_user_url(@user)
        assert_response :success
      end

      test "should update user" do
        patch user_url(@user), params: { user: { email: @user.email,
name: @user.name } }
        assert_redirected_to user_url(@user)
      end

      test "should destroy user" do
        assert_difference('User.count', -1) do
          delete user_url(@user)
        end

        assert_redirected_to users_url
      end
    end
```

З приведеного тексту виходить, що передбачається емуляція GET-, POST-, DELETE-запитів протоколу HTTP за допомогою виклику відповідних методів, яким передаються URL, що дозволяє перевірити поведінку усіх маршрутів і методів згенерованого контролера.

7.3 Зміст звіту

Звіт повинен містити:

1. Скріншот із версією Ruby, Rails;
2. Скріншоти виконання завдання Hello Rails
3. Скріншоти виконання завдання зі створення CRUD-додатку;

4. Скріншоти введення даних та існуючих записів в БД;
5. Відповідь сервера у форматі json;
6. Скріншот запущеного сервера із 2-3 відповідями;
7. Зезультат виконання тесту.

7.4 Контрольні питання

1. В чому суть концепції MVC
2. Які компоненти потрібно встановити для роботи з Rails?
3. Як створити аплікацію в Rails, що при цьому відбуваєтсья?
4. Як перевірити запустити сервер і перевірити його роботу?
5. Які можливі коди відповіді сервера?
6. Що таке Rack, як використовувати?
7. Що таке концепція CRUD?
8. Що таке json?
9. Для чого використовується Scaffold–генератор?
10. Яким чином Rails генерує код html-сторінки, які зв’язки між файлами?
11. Як згенерувати тести для створеної аплікації?

7.5 Рекомендовані ресурси

1. Michael Hartl. Ruby on Rails Tutorial. URL: https://www.railstutorial.org/book
2. Full code for Michael Hartl's Ruby on Rails Tutorial. URL: https://github.com/raleighlittles/RubyOnRails-Tutorial

Лабораторна робота № 8

Тема: Робота з компонентами MVC-RoR проекту

Мета роботи – навчитися створювати веб-аплікації із хедерами, футерами та Bootstrap.

8.1 Завдання для виконання

В лабораторній роботі №7 було створено просту аплікацію для реєстрування користувачів. В цій роботі потрібно додати зручності для роботи з додатком і додати оформлення.

Послідовність виконання завдань:

1. Відкрити проект ЛР7;
2. Додати до проекту хедер і футер [1].
3. У футері розмістити контактну інформацію про розробника.
4. Приєднати до проекту фреймворк Bootstrap [2].
5. Вибрати один із готових шаблонів в розділі Navs & Tabs [3], розмістити його у хедері свого проекту.
6. До пунктів навігації приєднати такі дії "Про проект", "Створити користувача", "Показати всіх користувачів", "Видалити всіх користувачів". Дія "Про проект" повинна відображати завдання до цієї лабораторної роботи.
7. Видалити дії, які є на сторінках, і продубльовані в навбарі.
8. Змінити маршрутизацію, щоб за адресою http://localhost:3000 відкривалася сторінка "Про проект".

Звіт по цих завданнях потрібно оформити р**азом із завданням наступної лабораторної роботи!**

8.2 Рекомендовані ресурси:

1. Загальний принцип роботи сайту. Робота з VIEW. URL: https://www.youtube.com/watch?v=fCbMzGMCbII
2. HTML. Bootstrap. Assets. URL: https://www.youtube.com/watch?v=ueVf6HIWA_c
3. Bootstrap Navs and tabs. URL: https://getbootstrap.com/docs/5.2/components/navs-tabs/

Лабораторна робота № 9

Тема: Розширення функціональності RoR проекту

Мета роботи – Навчитися інтегрувати авторизацію за допомогою гему Devise, використовувати завантаження зображень через ActiveStorage та обмежити доступ до функцій відповідно до ролей користувачів.

9.1 Завдання для виконання

Доповнити проект, що виконувався в лаб7-8 наступними завданнями.

Опис завдань:

1. Налаштування автентифікації користувачів за допомогою Devise [1]
   - Підключити гем Devise до проекту.
   - Налаштувати існуючу модель User для роботи з Devise.
   - Додати поле name до моделі User та дозволити його використання у формі реєстрації та редагування профілю.
   - Реалізувати поділ користувачів на ролі (admin та user) через атрибут role.
   - Налаштувати перевірку ролей у моделі User (наприклад, метод admin?).
2. **Додати модель Post** із такими полями:
   - title:string — заголовок поста
   - content:text — зміст поста
   - user_id:integer — посилання на автора поста
   - image:string — поле для збереження зображення, яке користувач може прикріпити до поста.
3. **Налаштувати асоціацію між моделями User та Post:**
   - Один користувач може мати багато постів.
   - Кожен пост належить до одного користувача.
4. **Валідації:**
   - Для моделі User: ім'я та email мають бути обов'язковими. Email повинен бути унікальним.
   - Для моделі Post: заголовок має бути обов'язковим та містити мінімум 5 символів; зміст поста — мінімум 20 символів.

5. **Додати функціонал завантаження зображень до постів:**
   - Використати гем active_storage [2] для роботи з файлами.
   - Перевірити, щоб завантажувалися лише зображення (формати jpg, png).
6. **Інтерфейс:**
   - Забезпечити доступ до створення, редагування та видалення постів лише для авторизованих користувачів.
   - Надати адміністраторам можливість переглядати та видаляти всі пости.
   - Додати перевірку, щоб звичайний користувач міг редагувати або видаляти лише свої пости.
   - У хедері додати пункт навігації **"Показати всі пости"**, який веде на сторінку списку всіх постів.
   - Для кожного поста на сторінці списку відобразити: заголовок, автора (ім'я користувача), зображення (якщо є), та перші 100 символів змісту.
7. **Сторінка користувача:**
   - На сторінці користувача додати список усіх його постів із посиланнями на їх редагування.
8. **Сторінка "Про проект":**
   - Розширити опис завдання, додавши роз'яснення про реалізовані валідації та роботу з файлами.

Звіт по цих завданнях потрібно оформити р**азом** із звітом по завданнях 8ї лабораторної роботи!

Вимоги вмісту до звіту:

1. Навести всі завершені фрагменти коду, які створювалися відповідно до завдань, або ті які модифікувалися відносно коду, що згенерував фреймворк

2. Навести скріншоти, які показують функціонування авторизації, розділення ролей, валідації, асоціації, завантаження файлів

9.2 Рекомендовані ресурси

1. https://www.youtube.com/watch?v=VjjiMNHuEfA Уроки Ruby on Rails (Авторизація - Devise. Генерація файлів MVC, Скафолд.)

2. https://www.youtube.com/watch?v=gGXG0aInNJw Уроки Ruby on Rails (Робота з картинками. Active Store. Налаштування Amazon S3. Lorem Ipsum).

9.3 Контрольні запитання

1. Які моделі використовуються в проекті, і які зв’язки між ними реалізовані?
2. Як забезпечується створення, редагування та видалення постів у додатку?
3. Як перевіряється, що пост належить поточному користувачеві, перед редагуванням або видаленням?
4. Який гем використовується для автентифікації користувачів, і які основні методи він надає?
5. Як налаштувати Devise для роботи з існуючою моделлю User?
6. Як виглядає базова структура маршрутизації додатку, і які маршрути відповідають за функції постів та користувачів?
7. Що таке strong parameters, і чому вони важливі в Ruby on Rails?
8. Як використовується фреймворк Bootstrap для стилізації сторінок у вашому проекті?
9. Як забезпечити доступ до певних дій лише для авторизованих користувачів?
10. Які основні кроки для додавання функціональності завантаження зображень у Rails-додаток?
11. Як додати до сторінки можливість перегляду завантаженого зображення разом із даними поста?
12. Як у хедері проекту реалізована навігація між основними сторінками додатку?

Рекомендована література

1. Sam Ruby, David Bryant Copeland, Dave Thomas. Agile Web Development with Rails 7 — Pragmatic Bookshelf, 2023. — 474 p..

2. M. Hartl: Ruby on Rails Tutorial: Learn Web Development with Rails. Addison-Wesley Professional; 7th edition, 2023

3. O. Fernandez: The Rails™ 7 Way. Addison-Wesley Professional, 2023

4. Polished Ruby Programming: Build better software with more intuitive, maintainable, scalable, and high-performance Ruby code. Packt Publishing, 2021

Електронні ресурси

1. Web application development with Ruby on Rails. URL:https://www.railstutorial.org/ (дата звернення: 26.05.2023)
2. Ruby Programming Language - Full Course. URL: https://www.youtube.com/watch?v=t_ispmWmdjY (дата звернення: 26.05.2023).
3. Rubyist UA. URL:https://www.youtube.com/@rubyistua (дата звернення: 26.05.2023)
4. Ruby on Rails 6 for Beginners (43 Lessons). URL: https://www.youtube.com/playlist?list=PLm8ctt9NhMNV75T9WYIrA6m9I_uw7vS56 (дата звернення: 26.05.2023)
5. Ruby - Programming Language (35 Tutorials) URL: https://www.youtube.com/playlist?list=PLLAZ4kZ9dFpO90iMas70Tt4_wYjhLGkya (дата звернення: 26.05.2023)