

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни

“Крос-платформне програмування”

для студентів

спеціальностей 121 Інженерія програмного забезпечення,
122 Комп’ютерні науки та інформаційні технології
(всіх форм навчання)

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Крос-платформне програмування” для студентів спеціальностей 121 Інженерія програмного забезпечення, 122 Комп’ютерні науки та інформаційні технології (всіх форм навчання) / А. О. Олійник, О. О. Олійник, Є. М. Федорченко, О. І. Качан. – Запоріжжя: ЗНТУ, 2016. – 19 с.

Автори: О. О. Олійник, к.т.н., доцент
Є. М. Федорченко, ст. викладач

Рецензент: С. О. Субботін, д.т.н., професор

Відповідальний
за випуск: С. О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

Вступ	4
1 Лабораторна робота № 1 Знайомство з мовою програмування Ruby.....	5
1.1 Мета роботи	5
1.2 Основні теоретичні відомості.....	5
1.3 Завдання до роботи.....	6
1.4 Зміст звіту.....	6
1.5 Контрольні питання.....	6
2 Лабораторна робота № 2 Робота з масивами в мові Ruby	7
2.1 Мета роботи	7
2.2 Основні теоретичні відомості.....	7
2.3 Завдання до роботи.....	9
2.4 Зміст звіту.....	9
2.5 Контрольні питання.....	10
3 Лабораторна робота № 3 Об'єктно-орієнтоване програмування у Ruby.....	11
3.1 Мета роботи	11
3.2 Основні теоретичні відомості.....	11
3.3 Завдання до роботи.....	13
3.4 Зміст звіту.....	14
3.5 Контрольні питання.....	14
4 Лабораторна робота № 4 Жадібні алгоритми в Ruby	15
4.1 Мета роботи	15
4.2 Основні теоретичні відомості.....	15
4.3 Завдання до роботи.....	17
4.4 Зміст звіту.....	17
4.5 Контрольні питання.....	18
Література.....	19

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ крос-платформного програмування.

Відповідно до графіка студенти перед виконанням лабораторної або самостійної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку по кожній роботі студент здає викладачу цілком оформлений звіт, а також електронні версії матеріалів з текстами розроблених програм, файлами програм, що виконуються, файлами даних і текстом звіту.

Звіт має містити:

- титульний аркуш;
- мету, варіант і завдання роботи;
- лаконічний опис теоретичних відомостей;
- текст програми, що обов'язково містить коментарі;
- вхідні та вихідні дані програми;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 x 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок. Для набору тексту звіту використовують редактор MS Word 97: шрифт Times New Roman, 12 пунктів. Міжрядковий інтервал: полуторний – для тексту звіту, одинарний – для лістингів програм, таблиць і роздруківок даних.

Під час співбесіди студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів розробленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і здачі роботи студент повинний відповісти на контрольні питання, приведені наприкінці опису відповідної роботи. Загальний залік студент одержує після виконання і здачі останньої роботи.

1 ЛАБОРАТОРНА РОБОТА № 1

ЗНАЙОМСТВО З МОВОЮ ПРОГРАМУВАННЯ

RUBY

1.1 Мета роботи

1.1.1 Ознайомитись з побудовою програм на мові Ruby.

1.1.2 Виконати завдання.

1.2 Основні теоретичні відомості

Ruby — це інтерпретована, повністю об'єктно-орієнтована мова програмування зі строгою динамічною типізацією. Ruby поєднує в собі Perl-подібний синтаксис із об'єктно-орієнтованим підходом мови програмування Smalltalk. Також деякі риси запозичено із мов програмування Python, Lisp, Dylan та CLU.

Щоб створити найпростішу програму на Ruby необхідно створити текстовий файл з розширенням `.rb`, наприклад `hello-world.rb`, в якому розмістити наступний код

```
puts 'Hello world'
```

Після чого запустити цей файл:

```
$ ruby hello-world.rb
```

```
Hello world
```

Ви також можете запустити коротеньку "hello world" програму без створення текстового файлу, так званий one-liner:

```
$ ruby -e "puts 'Hello world'"
```

```
Hello world
```

Ще можна запустити цей код за допомогою `irb`, але вихід буде трохи незвичним, оскільки `puts` виведе "Hello world", але `irb` виведе також значення, що повертає `puts` – це значення `nil`.

```
$ irb --simple-prompt
```

```
>> puts "Hello world"
```

```
Hello world
```

```
=> nil
```

1.3 Завдання до роботи

1.3.1 Закачайте дистрибутив Ruby за цією адресою - <http://rubyinstaller.org/downloads/> , встановіть інтерпретатор Ruby на свій комп'ютер.

1.3.2 Встановіть інтегроване середовище розробки, обрати можете будь-яке, але найбільш бажаним є безплатне середовище NetBeans IDE.

1.3.3 Після встановлення Ruby, виконайте наведене вище завдання.

1.3.4 Оформити звіт з роботи.

1.3.5 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Короткі теоретичні відомості.

1.4.3 Текст розробленого програмного забезпечення з коментарями.

1.4.4 Результати роботи програмного забезпечення.

1.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

1.5 Контрольні питання

1.5.1 Що таке крос-платформне програмування?

1.5.2 Місце Ruby у крос-платформному програмуванні?

1.5.3 Особливості програмування у Ruby?

1.5.4 Переваги та недоліки Ruby?

1.5.5 Засоби запуску Ruby-програм?

2 ЛАБОРАТОРНА РОБОТА № 2 РОБОТА З МАСИВАМИ В МОВІ RUBY

2.1 Мета роботи

2.1.1 Ознайомитись з особливостями побудови і роботи з масивами в мові Ruby.

2.1.2 Реалізувати програму відповідно до завдання.

2.2 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 2.1 – 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]

Метод Inject

Загальний вигляд:

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

start – початкове 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`.

2.3 Завдання до роботи

2.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи.

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

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

2.3.4 Дано масив з 10 цілих чисел (будь-яких). Вивести на екран всі елементи масиву, які менше нуля і діляться на 3.

2.3.5 Дано масив з 10 цілих чисел (будь-яких). Вивести на екран суму всіх парних додатних елементів масиву.

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

2.3.7 Оформити звіт з роботи.

2.3.8 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Короткі теоретичні відомості.

2.4.3 Текст програми (із коментарями).

2.4.4 Результати тестування програм.

2.4.5 Результати тестування програм.

2.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

2.5 Контрольні питання

2.5.1 Який основний тип даних?

2.5.2 Яким чином можна створювати масиви у Ruby?

2.5.3 Основні методи роботи з масивами у Ruby?

3 ЛАБОРАТОРА РОБОТА № 3

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ У RUBY

3.1 Мета роботи

Засвоїти основи об'єктно-орієнтованого програмування у Ruby.

3.2 Основні теоретичні відомості

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

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 головний (head?): для директора повертає істину, для інших – хибність.

ПБ 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
    2012 - @birthday
  end
end

```

Оскільки, в більшості випадків метод head? повертає істину, 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
```

3.3 Завдання до роботи

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

3.3.2 Розробити ієрархію класів, використовуючи основні парадигми програмування, обравши відповідний варіант:

Варіант №1. Створити наступну ієрархію класів: транспорт, автомобіль, автобус, трамвай, тролейбус.

Варіант №2. Створити наступну ієрархію класів: птах, курка, лебідь, пінгвін, страус.

Варіант №3. Створити наступну ієрархію класів: тварина, корова, кит, дельфін, собака.

3.3.3 Оформити звіт з роботи.

3.3.4 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Короткі теоретичні відомості.

3.4.3 Текст розробленого програмного забезпечення з коментарями.

3.4.4 Результати роботи програмного забезпечення.

3.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

3.5 Контрольні питання

3.5.1 Які основні принципи об'єктно-орієнтованого програмування?

3.5.2 Що таке поліморфізм?

3.5.3 Що таке інкапсуляція?

3.5.4 Що таке спадкування?

3.5.5 Назвіть засоби Ruby для забезпечення об'єктно-орієнтованої парадигми програмування?

4 ЛАБОРАТОРНА РОБОТА № 4 ЖАДІБНІ АЛГОРИТМИ В RUBY

4.1 Мета роботи

Вивчити методи рішення задач за допомогою жадібних алгоритмів у Ruby

4.2 Основні теоретичні відомості

Жадібний алгоритм — простий і прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на поточному етапі даних, не турбуючись про можливі наслідки, сподіваючись врешті-решт отримати оптимальне рішення. Легкий в реалізації і часто дуже ефективний за часом виконання. Багато задач не можуть бути розв'язані з його допомогою.

Наприклад, використання жадібної стратегії для задачі комівояжера породжує наступний алгоритм: «На кожному етапі вибирати найближче з невідвіданих міст».

Зазвичай, жадібний алгоритм базується на п'яти принципах:

- Набір можливих варіантів, з яких робиться вибір;
- Функція вибору, за допомогою якої знаходиться найкращий варіант;
- Функція придатності, яка визначає придатність отриманого набору;
- Функція цілі, оцінює цінність рішення, не виражена явно;
- Функція розв'язку, яка вказує на те, що знайдене кінцеве рішення.

Жадібний алгоритм добре розв'язує деякі задачі, а інші — ні. Більшість задач, для яких він спрацьовує добре, мають дві властивості: по-перше, до них можливо застосувати Принцип жадібного вибору, по-друге, вони мають властивість Оптимальної підструктури.

Принцип жадібного вибору.

До оптимізаційної задачі можна застосувати принцип жадібного вибору, якщо послідовність локально оптимальних виборів дає глобально оптимальний розв'язок. В типовому випадку доведення оптимальності має таку схему: спочатку доводиться, що жадібний вибір на першому етапі не унеможливорює шлях до оптимального розв'язку: для всякого розв'язку є інше, узгоджене із жадібним і не гірше першого. Далі доводиться, що підзадача, що виникла після жадібного вибору на першому етапі, аналогічна початковій, і міркування закінчується за індукцією. Інакше кажучи, жадібний алгоритм ніколи не переглядає свої попередні вибори для здійснення наступного, на відміну від динамічного програмування.

Задача: В певній державі існують монети номіналом 1, 2, 5, 10, 20, 50 та 100 тугриків. Іван кожного місяця отримує зарплату N . Допоможіть начальнику Івана видати йому зарплату найменшою кількістю монет.

Процес вирішення задачі можна організувати як циклом (динамічне програмування), так і рекурсією.

Рекурсія:

- задамо монети як масив
- створимо рекурсивну функцію, яка видає одну монету, на вхід вона буде приймати набір монет і залишок, повертати – властивість монети, яку необхідно видати
- якщо залишок співпадає з номіналом одної з монет – її необхідно видати
- знаходимо максимальну монету, яку можемо видати Івану, видаємо її, запускаємо функцію зі зменшеним залишком.

```
def give_coin (coins, n)
  if coins.include?(n)
    puts n
  else
    max = coins.find_all{|elem| elem < n}. max
    puts max
    give_coin(coins, n - max)
  end
end
coins = [1, 2, 5, 10, 20, 50, 100]
n=398
```

give_coin(coin, n)

Динамічне програмування:

- розіб'ємо задачу на дві підзадачі: знаходження жадібного вибору та видача монет
- цикл має продовжуватись доти, доки Івану є що видавати
- Жадібний вибір: знаходимо максимальну монету, яку можемо видати Івану
- в спеціальну змінну зберігаємо розмір залишку
 coins = [1, 2, 5, 10, 20, 50, 100]
 n=48
 while (n != 0)
 max = coins.find_all{|elem| elem <= n}. max
 puts max
 n = n - max
 end

4.3 Завдання до роботи

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

4.3.2 Створити програму, що вирішує наступну задачу: комівояжеру необхідно відвідати декілька міст. Необхідно знайти найкоротший із можливих шляхів. Передбачається, що всі міста сполучені між собою прямими шляхами, кожне з кожним.

4.3.3 Виконати тестування.

4.3.4 Оформити звіт з роботи.

4.3.5 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Короткі теоретичні відомості.

4.4.3 Текст розробленого програмного забезпечення з коментарями.

4.4.4 Результати роботи програмного забезпечення.

4.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

4.5 Контрольні питання

4.5.1 Що таке жадібний алгоритм?

4.5.2 Особливості жадібних алгоритмів?

4.5.3 Що таке динамічне програмування?

ЛИТЕРАТУРА

1. Martin Fowler. Patterns of Enterprise Application Architecture. Addison Wesley Longman, Reading, MA, 2003.
2. Фаулер. Ч. Rails. Сборник рецептов. Питер, СПб, 2008.
3. Хант Э., Томас Д. Программист-прагматик. Питер, СПб, 2007.
4. Dion Almaer, Justin Gehtland, Ben Galbraith. Pragmatic AJAX: A Web 2.0 Primer. The Pragmatic Programmers, LLC, Raleigh, NC, Dallas, TX, 2006.
5. David Thomas, Chad Fowler, Andrew Hunt. Programming Ruby: The Pragmatic Programmers' Guide. The Pragmatic Programmers, LLC, Raleigh, NC, Dallas, TX, 2nd edition, 2005.
6. Томас Д., Хэнссон Д.Х. Гибкая разработка веб-приложений в среде Rails. – СПб. : Питер, 2008. – 716 с.