21. Elementy języka Ruby

1 Instrukcja — testy jednostkowe

Testy jednostkowe (ang. *unit tests*) to metoda testowania napisanego kodu poprzez weryfikację działania niedużych jednostek programu (metod, klas, zmiennych itd.). Testy jednostkowe sprawdzają oczekiwane wyjście (zwracaną wartość, typ obiektu, wyrzucony wyjątek) z wyjściem uzyskanym po wykonaniu testowanego fragmentu kodu.

W praktyce stworzenie testów jednostkowych sprowadza się do użycia biblioteki udostępniającej *asercje* (ang. *assertions*), np. `JUnit` w Javie, i napisania z ich wykorzystaniem krótkich testów. Na przykład, poniższa asercja — będąca dwuargumentową metodą — sprawdza, czy wartość 6 jest wynikiem odpowiedniego wywołania metody `add_items`.

```
assert_equal(6, add_items(1,2,3))
```

Testy jednostkowe zawarte są w plikach *test_task_xx.rb*, w których (w komentarzu) zostały zamieszczone również treści zadań. Rozwiązania zadań należy umieszczać w pliku o nazwie *task_xx.rb*, umiejscowionym w tym samym katalogu, co testy jednostkowe. Zadanie oznaczone numerem *00* zostało już rozwiązane. Uruchomienie testu:

```
ruby test_task_00.rb
```

Jeżeli przynajmniej jeden test się nie powiedzie, oznacza to, że rozwiązanie zaimplementowane w pliku `task_xx.rb` jest niepoprawne. Na podstawie uzyskanej informacji o błędzie, należy je poprawić i uruchomić test jeszcze raz. Test zakończony powodzeniem:

```
# Running tests:
..
Finished tests in 0.001175s, 1701.4442 tests/s, 4253.6105 assertions/s.
2 tests, 5 assertions, 0 failures, 0 errors, 0 skips
```

Zadanie 01 Uruchom przykład z zadania 00 i rozwiąż zadania 01–05. Przydatne mogą okazać się metody: `String#to_f`, `String#split`, `String#sub`, `String#sub`, `Array#size`.

2 Bloki

Za iteracje po kolekcjach elementów w języku Ruby odpowiadają metody `#each` i tym podobne (np. `Integer#times`). Przyjmują one jako argument *blok* (ang. *block*) z odpowiednimi argumentami:

```
tab = [5, 10, 15]
tab.each { |e| puts e*2 }
```

Zadanie 02 Rozwiąż zadania 07 i 08. Zwróć uwagę na metody: `Array#each`, `Array#each_with_index`, `Array#each_index`, `Hash#each`, `Hash#each_key`, `Hash#each_value`.

Blok jest również obiektem, który można zdefiniować wcześniej. Można go rozumieć jako anonimową funkcję:

```
two_times = lambda{ |e| puts e*2 }  # zdefiniowanie bloku
two_times.call(5)                   # blok mozna wywolac

tab.each(&two_times)                # przekazanie bloku do #each
                                    # uwaga: znak & jest wymagany!
```

3 Elementy programowania funkcyjnego

Język Ruby udostępnia trzy podstawowe funkcje (*map*, *filter*, *fold*) wywodzące się z paradygmatu programowania funkcyjnego, jednak realizuje je w sposób obiektowy za pomocą bloków. Funkcje te nazywane są *funkcjami wyższego poziomu* (ang. *higher-order functions*), gdyż przyjmują jako argument inną funkcję (w języku Ruby blok).

3.1 Metoda `map`

Metoda `map` (alias: `collect`) jest metodą, która wykonuje podany blok na każdym elemencie tablicy, zwracając nową tablicę. Wywołanie funkcji `map`:

```
tab = [1, 2, 3]
tab.map{ |e| e*2 }      # wynikiem jest [2, 4, 6]
```

odpowiada poniższemu fragmentowi kodu:

```
tab = [1, 2, 3]
tab_sqr = []
tab.each{ |e| tab_sqr << e*2 }
tab_sqr                 # wynikiem jest [2, 4, 6]
```

Zadanie 03 Wykonaj zadania 10–11.

3.2 Metoda `select`

Metoda `select` (alias: `filter`) jest metodą, zwracającą z podanej tablicy tylko te elementy, dla których podany warunek (funkcja zwracająca wartość boolowską) jest prawdziwy. Wywołanie funkcji `select`:

```
tab = [1, 2, 3, 4]
tab.select{ |e| e%2 == 0 }  # wynikiem jest [2, 4]
```

odpowiada poniższemu fragmentowi kodu:

```ruby
tab = [1, 2, 3, 4]
tab_even = []
tab.each do |e|
  if e%2 == 0
    tab_even << e
  end
end
tab_even                    # wynikiem jest [2, 4]
```

Zadanie 04 Wykonaj zadania 13–14.

3.3 Metoda `reduce`

Metoda `reduce` (alias: `inject`, inaczej: *fold left*, *aggregate*, *accumulate* itp.) jest metodą, która iterując po elementach kolekcji agreguje je w jednej zmiennej:

```ruby
tab = [1, 2, 3]
tab.reduce(0){ |sum,e| sum + e }  # wynikiem jest 6
```

odpowiada poniższemu fragmentowi kodu:

```ruby
tab = [1, 2, 3]
sum = 0
tab.each{ |e| sum = sum + e }
sum                               # wynikiem jest 6
```

Zadanie 05 Wykonaj zadania 16–17.

Zadanie 06 Wykonaj pozostałe zadania: 06, 09, 12, 15, 18. Przydatne mogą się okazać obiekt `Range` oraz metody `Array#max`, `Array#min`, `Array#include?`, `Array#values`, `Array#join`, `Range#to_a`.