Типове данни

Както споменахме в предишната тема, всичко в Ruby е обект и всичко си има клас.

```
h = {"the answer to everything" => 42, :linux => "fun     for coders."}
puts "Stringy string McString!".class
puts 1.class
puts nil.class
puts h.class
puts :symbol.class
```

Резултат:

```
String
Fixnum
NilClass
Hash
Symbol
```

Всеки обект има метода `class`, който връща неговия клас.

Константи

Ще започнем с константите, защото са прости.
1.Константите започват с главни букви. `Constant` е константа, докато `constant` не е.
2. Можем да променяме стойността на константа.

Символи

Всеки път, когато пишем стрингове, Ruby създава нов обект, без значение дали те са идентични или не се създ. Всяка инстанция се третира като нов обект. Можем да имаме два еднакви стринга и Ruby няма да разпознае, че те са едно и също нещо.

```
irb> "live long and prosper".object_id
=> -507772268
irb> "live long and prosper".object_id
=> -507776538
```

ID то на върнатите обекти е различно. В такъв случай казваме, че обектите от такъв тип са immutable.

От друга страна, понякога не искаме подобно поведение(било то поради многото заета памет или други причини.) Точно за тази цел са създадени символите.

```
irb> :my_symbol.object_id
=> 150808
irb> :my_symbol.object_id
=> 150808
```

Символите се означават с двуеточие точно в началото на името - `:symbol_name`

Хешове

Хешовете са като речници. Имаме ключ по който намираме някаква стойност. Най - добрият начин да илюстрираме това е чрез бърза демонстрация:

1. `hash = { :leia => "Princess from Alderaan", :han => "Rebel without a cause", :luke => "Farmboy turned Jedi"}`
2. `puts hash[:leia]`
3. `puts hash[:han]`
4. `puts hash[:luke]`

Резултат:

```
Princess from Alderaan
Rebel without a cause
Farmboy turned Jedi
```

Можем да го напишем и по следния начин:

1. `hash = { :leia => "Princess from Alderaan", :han => "Rebel without a cause", :luke => "Farmboy turned Jedi"}`
2. `hash.each do |key, value|`
3. `    puts value`
4. `end`

Използваме цикъл, за да обходим всеки елемент от хеша добавяме ключа в `key`, а стойността в `value`.

Резултат:

```
Princess of Alderaan
Rebel without a cause
Farmboy turned Jedi
```

Ако искаме да махнем Luke, може да направим следното:

```
1. hash.delete(:luke)
```

Има и друг начин:

```
1. hash.delete_if {|key, value| value.downcase.match("farmboy")}
```

Тук итерираме през всички key-value двойки и изтриваме тези, за които блокът връща истина.

Можем да измерим дължината на шеша чрез `hash.length`

Масиви

Масивите много приличат на хешове с изключение на това, че ключовете винаги са последователни числа и винаги започват от 0.

Ето два начина за създаване на масив:

```
1. array1 = ["hello", "this", "is", "an", "array!"]

2. array2 = []

3. array2 << "This"   # index 0

4. array2 << "is"     # index 1

5. array2 << "also"   # index 2

6. array2 << "an"     # index 3

7. array2 << "array!" # index 4
```

Операторът << добавя елементите в масива. Това можем да го направим и като използваме `array2[4] = "array!"`

Изстриването става по следния начин:

```
1. string = array2.pop
```

Можем да преместим всички елементи от един масов в друг:

```
1. array1 << array2.pop until array2.empty?
```

Масивите могат да бъдат изваждани и добавяни един към друг.

```
1. array3 = array1 - array2

2. array4 = array1 + array2
```

Можем да превърнем целия масив в String:

```
1. string = array2.join(" ")
```

Стрингове

В Ruby има мощни вградени функции са работа със стрингове.

Можем да ги умножаваме с число:

```
  "Danger, Will Robinson!" * 5
```

Резултат:

```
  Danger, Will Robinson!Danger, Will Robinson!Danger, Will Robinson!Danger, Will
Robinson!Danger, Will Robinson!
```

Можем да ги сравняваме:

```
  "a" < "b"
```

Резултат:

```
  true
```

Конкатенацията е както при повечето езици: добавянето на символа “+” между два стринга ще върне като резултат стринг който има същата стойност като първоначалните два, поставени един до друг.

1. `"Hi, this is " + "a concatenated string!"`

Резултат:

```
Hi, this is a concatenated string!
```

Числа (Integer и Float)

Операциите с числа са подобни на тези при повечето езици за програмиране.