آموزش زبان برنامه نویسی Ruby

محمدرضا حقیری

۸ اسفند ۱۳۹۳

فهرست مطالب

۱ مقدمه

زبان روبی ، یک زبان اسکریپتی، شی گرا، تابعی و مدرن است. این زبان توسط یوکیهیرو ماتسوموتو [۱] ایجاد شده و هم اکنون توسعه دهندگانی از سراسر جهان، آن را توسعه میدهند. این زبان، نحو [۲] ساده ای داشته، و می توان در عرض چند روز آن را به خوبی فرا گرفت. این زبان، برای توسعه برنامه های کاربردی دسکتاپ، وب و حتی نوشتن کتابخانه و سرویس های مختلف، کاربرد دارد. شما میتوانید به سادگی و با استفاده از این کتاب، این زبان را بیاموزید.

۱.۱ این کتاب برای چه کسانی است؟

به اختصار، باید گفت همه علاقمندان به یادگیری این زبان ساده و جذاب. جواب طولانی تر، اینست که این کتاب، برای کسانی است که واقعا میخواهند به سراغ روبی بروند، و چون روبی مرجع فارسی مطمئنی ندارد، پشیمان می شوند. پس اگر از هر دو دسته هستید، توصیه میکنم این کتاب را از دست ندهید.

۲.۱ آیا میتوانیم این کتاب را به اشتراک بگذاریم؟

۳.۱ چگونه این کتاب را بخوانیم؟

برای خواندن این کتاب، دو راه دارید، یا صرفا مطالعه روزنامه وار انجام دهید تا به آشنایی اجمالی با نحو زبان روبی برسید، یا اینکه تک تک نرم افزارها و مراحل گفته شده در کتاب را مرحله به مرحله انجام دهید، تا کاملا به روبی مسلط شوید.

---

[۱] Yukihiro Matsumoto
[۲] Syntax

۲ پایه ها و مقدمات یادگیری روبی

در این فصل، چه فراخواهید گرفت؟

- **نصب روبی در سیستم عامل**
- **آشنایی با محیط اینتراکتیو روبی**
- **ایجاد چند مثال ساده با روبی**
- **اجرای اسکریپت نوشته شده در روبی**

این موارد، اولین قدم های شما در یادگیری زبان روبی هستند. در فصول بعدی، با ساختارها و کدهای بیشتری آشنا خواهید شد، و طبیعتا آن زمان است که میتوانید از روبی، استفاده کاربردی کنید.

۱.۲ نصب روبی

برای نصب روبی، شما ابتدا باید نام و نسخه سیستم عامل خود را بدانید. در این کتاب، فرض بر آنست که شما از گنو/لینوکس و توزیع اوبونتو استفاده میکنید. [۳]

۱.۱.۲ وبسایت روبی

شما میتوانید با مراجعه به وبسایت روبی [۴] **نسخه مورد نظر روبی را دانلود نموده، سپس آن را کامپایل کنید.** [۵]

۲.۱.۲ نصب از مخازن

برای استفاده از مخازن اوبونتو، کافیست تا یک پنجره ترمینال باز کنید، و سپس دستورات زیر را اجرا کنید :

```
sudo apt-get install ruby
```

همچنین اگر مفسر [۶] **دیگری در نظر دارید، میتوانید در صورت موجود بودن در مخازن، آن را نصب کنید.**

---

[۳] **تفاوتی میانی استفاده از روبی در ویندوز و لینوکس نیست، طبیعتا میتوانید همین دستوراتی که در این کتابچه موجودند را به کار ببرید**

[۴] http://ruby-lang.org

[۵] **وقتی روبی را از سورس کامپایل میکنید، باید پیشاپیش هر چه پیش نیاز آن است را دستی نصب کنید. چنانچه دانش کافی در زمینه کامپایل ندارید، توصیه میکنم از مخازن استفاده کنید.**

[۶] Interpreter

۲.۲ استفاده از محیط تعاملی

محیط تعاملی [۷] روبی، به شما این امکان را میدهد تا بدون نیاز به نوشتن کد در یک ویرایشگر متن، آن را مستقیما در ترمینال تایپ نموده، سپس نتیجه برنامه را در ترمینال ببینید. برای دسترسی به محیط تعاملی، یک پنجره ترمینال باز کرده و مانند دستور زیر عمل کنید :

```
prp-e@prp-e ~ $ irb
```

پس از اجرای دستور فوق، باید یک پرامپت [۸] به این شکل مشاهده کنید :

```
irb(main):001:0>
```

درون این محیط، میتوانید دستورات مورد نظر را اجرا و تست کنید.

۳.۲ چند دستور در محیط تعاملی

مثال معروفی به نام «سلام دنیا» در تمامی زبان های برنامه نویسی وجود دارد. بیایید این مثال، اولین مثال ما نیز در زبان روبی باشد. اکنون، پنجره ترمینال را باز کرده، و همانند شکل زیر دستور را درون محیط تعاملی روبی، وارد کنید :

```
print "Hello, World\n"
```

در صورتی که کد را درست وارد کرده باشید، باید نتیجه به این شکل گرفته باشید :

```
irb(main):001:0> print "Hello, World\n"
Hello, World
=> nil
irb(main):002:0>
```

اما نکته اینجاست، در زبان روبی، اصولا نیازی به دستوری برای ایجاد خط جدید نداریم. دستور مناسبتری وجود دارد که جایگزین print است و می توان به کمک آن، به خوبی داده ها را آنگونه که میخواهیم، چاپ کنیم. برای انجام کار عملی، دستورات زیر را در محیط تعاملی اجرا کنید :

```
puts "Hello, World"
```

اکنون باید نتیجه ای مشابه این دریافت کنید :

```
irb(main):002:0> puts "Hello, World"
Hello, World
=> nil
irb(main):003:0>
```

دستور puts جایگزین خوبی برای print است. علی الخصوص وقتی نتیجه مورد نظر، تنها یک خط باشد. اگرچه در آینده، از هر دو دستور استفاده خواهیم کرد.

---

[۷] Interactive
[۸] Prompt

۴.۲ نوشتن اسکریپت در یک فایل

روبی یک زبان اسکریپتی است [۹] پس میتوانیم یک اسکریپت بنویسیم و با استفاده از مفسر، آن را اجرا کنیم. یا حتی می توانیم به آن یک دسترسی اجرایی بدهیم، سپس مستقیما اجرایش کنیم.

۱.۴.۲ انتخاب ویرایشگر

شما برای آن که بتوانید کد خود را درون یک فایل روبی بنویسید، طبیعتا به یک ویرایشگر متن [۱۰] نیاز دارید. ویرایشگر های متن که همراه سیستم عامل ها نصب می شوند، گزینه های خوبی هستند. اگر میخواهید بدانید من از چه استفاده میکنم، از **nano** به عنوان ویرایگشر تحت ترمینال، و از **gedit** به عنوان ویرایشگر گرافیکی استفاده میکنم. [۱۱]

۲.۴.۲ پسوند فایل های روبی

سورس کدهای روبی، عموما با پسوند های .rb و یا .ruby ساخته می شوند. توجه کنید که کامپیوتر به پسوند ها کاری ندارد، و این پسوند ها صرفا برای این ساخته شده اند که انسان بتواند با استفاده از آنها، نوع فایل را تشخیص دهد.

۳.۴.۲ نوشتن و اجرای اسکریپت از طریق مفسر

مفسر روبی، از طریق خط فرمان با نام **ruby** در دسترس است، پس میتوانیم به سادگی یک قطعه کد بنویسیم، آن را روی دسکتاپ قرار داده و سپس دستورات مورد نظر را درونش وارد کنیم. اکنون، یک فایل خالی روی دسکتاپ اجرا کنید و این دستور را درونش بنویسید :

```
puts "Hello, World"
```

با فرض آن که این برنامه، روی دسکتاپ شما با نام **HelloWorld.rb** ذخیره شده است، آن را اینگونه اجرا میکنیم :

```
prp-e@prp-e ~ $ ruby ~/Desktop/HelloWorld.rb
```

سپس باید روی ترمینال چنین نتیجه ای را مشاهده کنیم :

```
prp-e@prp-e ~ $ ruby ~/Desktop/HelloWorld.rb
Hello, World
prp-e@prp-e ~ $
```

این اسکریپت، از طریق مفسر روبی، اجرا شد. در قسمت بعدی، از طریق اجازه ها [۱۲] اسکریپت را اجرا خواهیم نمود.

---

[۹] زبان های اسکریپتی، خط به خط اجرا می شوند. برخلاف زبانهای کامپایل شده، که ابتدا کل برنامه به یک زبان قابل فهم برای سیستم عامل یا ماشین ترجمه می شود، سپس اجرا میگردد.

[۱۰] Text Editor

[۱۱] در نظر داشته باشید چنانچه میزکار **MATE** را نصب دارید، ادیتور **gedit** با نام **pluma** در دسترس است.

[۱۲] Permissions

۴.۴.۲ اجرای اسکریپت با استفاده از اجازه ها

این بار، از خاصیت اجازه دسترسی در یونیکس، استفاده خواهیم کرد و سپس اسکریپت قسمت قبل را مستقلا اجرا خواهیم کرد. اکنون، نام HelloWorld.rb را به HelloWorld تغییر دهید و سپس با مفسر اجرایش کنید :

```
prp-e@prp-e ~ $ ruby ~/Desktop/HelloWorld
Hello, World
prp-e@prp-e ~ $
```

نتیجه تفاوتی نکرد، نباید هم بکند. مفسر دستورات خودش را تشخیص میدهد. حالا میخواهیم مفسر را درون خود برنامه جای دهیم!
کد برنامه را به این شکل ویرایش کنید :

```
#!/usr/bin/ruby

puts "Hello, World"
```

اکنون، مفسر را به اسکریپت خود شناسانده ایم. اکنون وقت آن است که اجازه اجرا به سورس کد خود بدهیم. اکنون فقط کافیست تا این دستور را در ترمینال اجرا کنید :

```
prp-e@prp-e ~ $ chmod +x Desktop/HelloWorld
```

سپس میتوان این کد را مستقیما اجرا کرد. برای اجرای مستقیم کد کافیست دستور زیر را اجرا کنید :

```
prp-e@prp-e ~ $ Desktop/HelloWorld
```

نتیجه دریافتی روی ترمینال باید به این شکل باشد :

```
prp-e@prp-e ~ $ Desktop/HelloWorld
Hello, World
prp-e@prp-e ~ $
```

۵.۲ جمع بندی

در اینجا، باید یک جمع بندی نهایی از مباحثی که در فصل مطرح شدند، داشته باشیم. اول اینکه، فراگیری روبی بسیار آسان است، ولی باید بدانید کجا و چگونه، چه دستوری را به کار ببرید. دوم، باید بدانید که چه موقع محیط تعاملی، و چه موقع محیط ویرایشگر را برای نوشتن برنامه انتخاب کنید. در نهایت هم همیشه حواستان به چگونگی نوشتن کد در ویرایشگر باشد، چرا که کوچک ترین اشتباهی، میتواند مانع از اجرای درست برنامه شما باشد.

۶.۲ تمرینات

همانگونه که در فروم اوبونتو قول داده بودم [۱۳] ، در پایان هر فصل، تعدادی تمرین خواهم گنجاند. با انجام این تمرینات، به سادگی میتوانید دانش و توانایی خود در زمینه روبی را بسنجید.

۱. **برنامه ای بنویسید که نام خودتان را چاپ کند**

۲. **برنامه ای بنویسید که نام دانش آموزان یک کلاس ۱۵ نفره را چاپ کند، و آن را در قالب یک اسکریپت با اجازه اجرا، اجرا کنید**

[۱۳] **انجمن های فارسی اوبونتو، در آدرس forum.ubuntu.ir در دسترس هستند. قول این کتاب را در تاپیکی با مضمون آموزش روبی، در آن انجمن داده بودم.**

۳ انواع داده ها و متغیرها، ثابت ها و عملگرها

در هر زبان برنامه نویسی، چندین نوع داده مختلف وجود دارد. استفاده از انواع مختلف داده، کار شخص برنامه نویس و حتی خود برنامه را آسان میکند. چرا که اینگونه، ماشین میفهمد که باید چگونه عملیاتی را روی داده انجام دهد. در این فصل، قصد داریم چندین نوع داده و همچنین تعریف آنها به عنوان متغیر را در روبی بررسی کنیم.

۱.۳ در این فصل چه فرا خواهید گرفت؟

- **شناسایی انواع مختلف داده ها**
- **تعریف انواع مختلف داده در متغیر**
- **تبدیل انواع داده**

۲.۳ داده های عددی

به طور کل، این نوع داده ها، اولین داده هایی هستند که در آموزش هر زبان برنامه نویسی، با آنها کار خواهیم داشت. چرا که گاهی با مقایسه و یا کم و زیاد کردن اعداد، کارهای خاصی در روند یک برنامه انجام میشود که میتواند به سرعت اجرای برنامه، کمک کند.

۱.۲.۳ اعداد صحیح

اعداد صحیح [۱۴] ، نوعی از اعداد هستند که جزء اعشاری ندارند. دامنه این اعداد در طبیعت از منفی بی نهایت تا مثبت بی نهایت ادامه دارد، با این تفاوت از اعداد حقیقی، که این دسته از اعداد، گسسته هستند. برای مثال در فاصله ۲ تا ۳ در مجموعه اعداد صحیح، هیچ عدد دیگری قرار نمیگیرد، در حالی که در سایر گونه های اعداد، اینگونه نیست. اگرچه اینجا بحث، بحث ریاضیات نیست، ولی بهتر بود توضیح ساده ریاضی اعداد صحیح را در اینجا ذکر میکردم. حالا نوبت آن است تا در محیط تعاملی روبی، چند عدد صحیح را تایپ کنید. برای تایپ کردن کافیست با دستور irb وارد محیط تعاملی شده و اعداد زیر را تایپ کنید :

```
2
3
7
```

باید چنین خروجی هم دریافت کنید :

```
irb(main):001:0> 2
=> 2
irb(main):002:0> 3
=> 3
```

[۱۴] Integer

```
irb(main):003:0> 7
=> 7
irb(main):004:0>
```

البته میتوانید هر کدام از اعداد را به دلخواه، منفی کنید. مهم اینست که در این قسمت، شما با نوع داده صحیح آشنا شدید.

۲.۲.۳ اعداد ممیز شناور

اعداد ممیز شناور [۱۵] ، نوعی از اعداد هستند که بخش اعشاری دارند. در واقع، این اعداد در برنامه نویسی، همان اعداد حقیقی هستند. اگرچه ، اعداد گویا، اصم و گنگ را نیز میتوان انواع دیگری از داده حساب نمود، اما در برنامه نویسی، همه اعداد غیر صحیح در مبنای ده را به نوعی میتوان ممیز شناور به حساب آورد.
برای تعریف این اعداد، کافیست به سادگی یک ممیز و یک صفر به اعداد معرفی شده در قسمت بالاتر، اضافه نمایید :

```
2.0
3.0
7.0
```

و شاهد چنین خروجی باشید :

```
2irb(main):005:0> 2.0
=> 2.0
irb(main):006:0> 3.0
=> 3.0
irb(main):007:0> 7.0
=> 7.0
irb(main):008:0>
```

همانگونه که ملاحظه میفرمایید، مفسر ممیز را لحاظ کرده است. پس طبیعتا بین ۲.۰ و ۲ تفاوت قائل می شود. اکنون با دو نوع اصلی و عمده داده های عددی آشنا شده اید، بیاییم با دو نوع داده عددی دیگر نیز آشنا شویم.

۳.۲.۳ اعداد باینری

اعداد دودویی یا باینری [۱۶] اعدادی هستند که از مبنای ده به مبنای دو برده شده اند. [۱۷] این اعداد در روبی با 0b شروع می شوند. شما اکنون میتوانید این اعداد را در محیط تعاملی وارد کنید :

[۱۵] Floating Point
[۱۶] Binary
[۱۷] تنها ارقام ۰ و ۱ میتوانند درون مقادیر باینری قرار بگیرند

```
0b1
0b10
0b11
0b101
```

و چنین نتیجه ای را دریافت کنید :

```
irb(main):001:0> 0b1
=> 1
irb(main):002:0> 0b10
=> 2
irb(main):003:0> 0b11
=> 3
irb(main):004:0> 0b101
=> 5
irb(main):005:0>
```

بعد ها با این نوع داده، بیشتر آشنا خواهید شد.

۴.۲.۳ اعداد اوکتال

اعداد اوکتال [۱۸] اعدادی هستند که از مبنای ده، به مبنای هشت برده شده اند. روبی، زبانی است که قابلیت تشخیص مبنای هشت و شانزده را دارد. برای این که این اعداد را به محیط تعاملی روبی بدهیم، کافیست با دستور ورود آنها، و رنجی از اعداد که پشتیبانی میکنند، آشنا باشیم. [۱۹] اعداد اوکتال در روبی، با صفر آغاز می شوند [۲۰] پس اکنون کافیست محیط تعاملی روبی را باز کرده و اعداد زیر را وارد کنید :

```
012
02
013457
```

خب اکنون باید نتیجه زیر را شاهد باشید :

```
irb(main):001:0> 012
=> 10
irb(main):002:0> 02
```

[۱۸] Octal

[۱۹] اعداد اوکتال یا مبنای هشت، اعدادی هستند که از ارقام صفر تا هفت تشکیل شده اند

[۲۰] این یک قرار داد است که این اعداد، با صفر آغاز شوند.

```
=> 2
irb(main):003:0> 013457
=> 5935
irb(main):004:0>
```

احتمالا بعد از دیدن این نتایج تعجب کرده اید، و پیش خودتان حدس هایی زده اید. باید بگویم درست حدس زده اید! روبی این اعداد را به مبنای ده برده و برای شما، نشان داده است. روبی نوع دیگری از داده های در مبنای غیر ده نیز دارد که در قسمت بعدی با آن آشنا خواهید شد.

۵.۲.۳ اعداد هگزادسیمال

اعداد هگزادسیمال [۲۱] در مبنای ۱۶ هستند. این اعداد، در علوم کامپیوتر بسیار پر استفاده اند و در دروسی مانند مبانی برنامه نویسی و ساختمان داده، بسیار به آنها پرداخته می شود. [۲۲] برای تعریف این اعداد در روبی، عدد مورد نظر را پس از صفر و x در محیط تعاملی وارد میکنیم. برای امتحان، اعداد زیر را در محیط تعاملی وارد کنید :

```
0x12
0x2
0x13457
```

سپس باید شاهد چنین نتیجه ای باشید :

```
irb(main):004:0> 0x12
=> 18
irb(main):005:0> 0x2
=> 2
irb(main):006:0> 0x13457
=> 78935
irb(main):007:0>
```

این بار هم شاهد این بودید که مفسر روبی، اعداد را از مبنای شانزده به ده برده و به شما نمایش داده است. در ادامه، با انواع دیگری از متغیر ها آشنا خواهید شد، که دیگر عددی نیستند.

۳.۳ متغیرهای بولین

متغیر های بولین [۲۳] ، متغیر هایی هستند که تنها دو مقدار درست (true) و یا غلط (false) دریافت میکنند.

---

[۲۱] Hexadecimal

[۲۲] در اینجا، ارقام ۰ تا ۹ را به علاوه ارقام A ، B ، C ، D و E داریم که نمایانگر ۱۰ تا ۱۵ هستند.

[۲۳] Boolean

۴.۳ متغیر های رشته ای

یک بار دیگر، برنامه زیر را چک کنید :

```
puts "Hello, World"
```

اگر دقت کنید، خواهید دید که عبارت Hello, World را درون علامت نقل قول قرار داده ایم. این نوع داده را، رشته [۲۴] میگویند. هر رشته، از بهم پیوستن چندین کاراکتر [۲۵] تشکیل می شود. یک رشته، میتواند شامل چند خط باشد، یا شامل یک کلمه باشد. در کل، قانون و قاعده خاصی برای تعریف مقادیر درون یک متغیر رشته ای، وجود ندارد.

۵.۳ آرایه ها و هش ها

آرایه ها [۲۶] و هش ها [۲۷] دو نوع متغیر مجموعه ای هستند. آرایه، مجموعه ای از متغیر های یک نوع خاص است، و هش مانند یک دیکشنری عمل می کند. با عملکرد این دو نوع متغیر، در آینده آشنا خواهید شد.

۶.۳ تعریف متغیرها

بر خلاف زبان هایی مانند C و C++ که ابتدا نوع، سپس نام و سپس مقدار متغیر تعریف میگردد، در روبی تنها نام متغیر کافیست تا به آن مقدار داده شود. دستورات زیر، چند متغیر عددی و رشته ای را در روبی تعریف میکنند :

```
name = "Muhammadreza"
a = "2"
b = 3
```

متغیر های **name** و **a** از نوع رشته ای هستند (به علامت نقل قول توجه کنید) ، و متغیر **b** از نوع عدد صحیح. نکته اینجاست که متغیرها، امکان تبدیل به یک دیگر را نیز دارند.

۱.۶.۳ تبدیل نوع متغیر ها

متغیر **a** از نوع رشته ای، و متغیر **b** از نوع عدد صحیح است. برای تبدیل **a** به عدد صحیح، و برای تبدیل **b** به ممیز شناور، از دستورات زیر استفاده میکنیم :

```
a.to_i
b.to_f
```

و برای ذخیره نتایج، کافیست متغیر جدیدی تعریف کرده، و دستورات را درونش بریزید. اگرچه، تغییر نوع های دیگری نیز وجود دارد که در ادامه به آنها خواهیم پرداخت.

---

[۲۴] String
[۲۵] Character
[۲۶] Array
[۲۷] Hash

۷.۳ عملگرها

عملگرها، در ریاضی و همچنین برنامه نویسی از اهمیت ویژه ای برخوردارند. شما اگر بخواهید روی متغیرها عملیاتی انجام دهید، قطعا نیاز خواهید داشت تا از عملگرها استفاده کنید. در روبی، چهار عمل اصلی جمع، تفریق، ضرب و تقسیم توسط چهار عملگر / ,* ,- ,+ انجام می شوند. همچنین، برای توان رسانی نیز از عملگر ** استفاده می شود. برای آزمایش عملگرها، کافیست دستورات زیر را در محیط تعاملی وارد کنید :

```
a + b
a - b
a * b
a / b
a ** b
```

و سپس باید شاهد چنین خروجی باشید :

```
irb(main):003:0> a + b
=> 5.0
irb(main):004:0> a - b
=> -1.0
irb(main):005:0> a * b
=> 6.0
irb(main):006:0> a / b
=> 0.6666666666666666
irb(main):007:0> a ** b
=> 8.0
irb(main):008:0>
```

نکته ای که متوجه آن شده اید، اینست که اگر یکی از متغیرها از نوع ممیزشناور باشد، خود مفسر تبدیل نوع را انجام میدهد.

۸.۳ ثابت ها

ثابت ها نیز همانند متغیرها، میتوانند از نوع عدد صحیح، ممیزشناور، متن و ... باشند. با این تفاوت که متغیر ها را میتواند مقدارشان را تغییر داد ولی ثابت ها را نه. تفاوت تعریف ثابت با متغیر، این است که نام ثابت، همیشه با حرف بزرگ شروع می شود، برای آزمایش، مقدار زیر را به عنوان یک ثابت به محیط تعاملی بدهید :

```
CONST_INT = 2
CONST_FLOAT = 3.0
CONST_STRING = "Hello, World"
```

اکنون سعی کنید مقدار یکی از این ثابت ها را تغییر دهید، طبیعتا با مشکل روبرو خواهید شد. چرا که از دید مفسر، این ها یک بار برای همیشه تعریف شده اند و نمیتوان آنها را تغییر داد.

۹.۳ جمع بندی

در این فصل، آموختید که هر نوع داده ای کجا و چگونه به کار میرود. مهم است که بدانید کجا و چطور، از چه نوع داده ای استفاده کنید. مثلا موقعی که خرده داشتن نتیجه، برایتان مهم نیست، منطقی تر است حتی اگر ورودی ممیز شناور باشد، شما خروجی را به عدد صحیح تبدیل کنید. اما اگر جایی قرار است مثلا تا n رقم اعشار محاسبه شود، منطقی تر آن است که حتی ورودی های صحیح، در خروجی به صورت ممیز شناور نمایش داده شوند. برخی از انواع داده نیز معرفی شدند، اما مثالی از آنها زده نشد. این نوع داده ها، هر کدام خودشان یک فصل جداگانه میطلبیدند، پس نگران نباشید، این ها را در ادامه فراخواهید گرفت.

۱۰.۳ تمرینات

۱. **برنامه ای بنویسید که اعداد یک تا ده را به توان ۲ رسانده و در خروجی نشان دهد**

۲. **برنامه ای بنویسید که نشان دهد شما چند روز عمر کرده اید، و خروجی را به صورت عدد صحیح نشان دهد**

۳. **برنامه ای بنویسید که از یک ثابت PI و یک متغیر radius استفاده کرده، سپس مساحت دایره ای را حساب کند**

۴ آرایه ها و هش ها

در فصل پیش، اشاره کوچکی به این دو نوع داده ای شد. در واقع، این دو نوع داده ای، در کنار چندتایی ها [۲۸] نوع داده هایی هستند که با مجموعه ها سرو کار دارند. در این فصل، این نوع داده ها را بررسی کرده، سپس متدهایی که در روبی برای آنها در نظر گرفته است را بررسی میکنیم.

۱.۴ در این فصل چه فراخواهید گرفت؟

- تعریف یک آرایه
- تعریف یک هش
- استفاده از چند تایی ها
- استفاده از اعضای یک آرایه به عنوان متغیر
- استفاده از متدهای در نظر گرفته شده، برای مرتب کردن، کوتاه کردن، برعکس کردن و جایگزین کردن یک عضو در آرایه و هش.

۲.۴ آرایه ها

آرایه ها، مجموعه ای از چندین مقدار می باشند. این مقادیر، میتوانند عدد صحیح، ممیز شناور و رشته و کاراکتر باشند. گاهی نیز برای مرتب شدن متغیرهای مورد نیاز، یک سری از مقادیر مورد نیاز را داخل یک آرایه قرار میدهیم، و برای استفاده از آن مقدار، خانه ای از آرایه که آن مقدار درونش قرار داده شده را فراخوانی میکنیم. برای تعریف یک آرایه، دستور زیر را در محیط تعاملی وارد کنید :

```
a = [1, 2, 3, 4, 5]
```

دستور فوق، میگوید یک آرایه که اعداد ۱ تا ۵ درونش قرار دارند ایجاد شود. تا اینجا، آرایه را ایجاد کردیم. حال نوبت متدهاست که یکی یکی، روی آرایه ایجاد شده اعمال میکنیم تا کار با آرایه را فرا بگیریم. [۲۹]

۱.۲.۴ متد length

واژه length در زبان انگلیسی، معنای «طول» میدهد. با اجرای این متد، طول آرایه به دست می آید. برای اجرای دستور، در محیط تعاملی به این شکل وارد کنید :

```
a.length
```

نتیجه دریافتی باید به شکل زیر باشد :

---

[۲۸] Tuple

[۲۹] در اینجا همه متدها آورده نشده است، و دلیل این امر، آنست که اگر میخواستیم تک تک متدهای آرایه ها را اینجا بیاوریم، مطلب بسیار طولانی و حوصله سر بر می شود.

```
irb(main):002:0> a.length
=> 5
irb(main):003:0>
```

مفسر اینجا به شما می گوید که پنج عضو درون این آرایه قرار دارند.

۲.۲.۴ متد reverse

این متد، آرایه را برعکس میکند. برای آنکه بدانید چگونه کار میکند، دستور زیر را درون محیط تعاملی وارد کنید :

```
a.reverse
```

نتیجه باید به شکل زیر باشد :

```
irb(main):003:0> a.reverse
=> [5, 4, 3, 2, 1]
irb(main):004:0>
```

استفاده از این متد هم زمانی که یک آرایه که از کوچک به بزرگ (یا برعکس) مرتب شده باشد را نیاز داریم، کار آمد است.

۳.۲.۴ مرتب کردن آرایه

اگر به زبانهایی مانند C آشنا باشید، میدانید که برای مرتب کردن آرایه ها، نیاز دارید تا از روش هایی مانند حلقه های تکرار و ... استفاده کنید. اما روبی، اینگونه نیست.
قبل از اینکه بخواهیم آرایه را مرتب کنیم، دقت کنید که آرایه داده شده در قسمت های قبل، مرتب بود. حالا آن را به هم میریزیم :

```
a = [1, 3, 2, 4, 5]
```

حالا تنها کافیست دستور زیر را اجرا کنید، تا آرایه مرتب گردد :

```
a.sort
```

نتیجه دریافتی باید به شکل زیر باشد :

```
irb(main):005:0> a.sort
=> [1, 2, 3, 4, 5]
irb(main):006:0>
```

اکنون، با دستور زیر، آرایه را از بزرگ به کوچک مرتب میکنیم :

```
a.sort.reverse
```

و نتیجه ما چنین خواهد بود :

```
irb(main):007:0> a.sort.reverse
=> [5, 4, 3, 2, 1]
irb(main):008:0>
```

تا اینجا، کار با آرایه را یاد گرفتید و با چندین متد آشنا شدید. زین پس، به سراغ این می رویم که بعضی خواص آرایه را بررسی کنیم.

۴.۲.۴ نمایه اعضای آرایه

نمایه [۳۰] به عددی گفته می شود که نماینده اعضای آرایه است. نمایه یا ایندکس، از صفر شروع می شود (یعنی نخستین عضو نمایه صفر میگیرد) و تا طول آرایه، ادامه می یابد. در واقع برای a ، که در بالا تعریف کردیم، اعداد ۰ تا ۴ هستند که نمایه های اعضا به شمار می آیند. برای امتحان کردن این موضوع، دستور زیر را اجرا کنید :

```
a[0]
```

و نتیجه باید چنین باشد :

```
irb(main):009:0> a[0]
=> 1
irb(main):010:0>
```

و اگر این یکی دستور را اجرا کنید :

```
a[4]
```

نتیجه باید این چنین باشد :

```
irb(main):010:0> a[4]
=> 5
irb(main):011:0>
```

همچنین، اگر از اعداد منفی استفاده کنید، از آخر به اول مقادیر برگردانده میشوند، یعنی عضو -۱ در این آرایه، همان عضو ۴ است.

[۳۰] Index

۵.۲.۴ اضافه کردن اعضا به آرایه

برای اضافه کردن اعضا به آرایه، چندین راه حل وجود دارد. ساده ترین راه آن، این است که اعضایی که لازم داریم را در قالب یک آرایه جدید، به آرایه پیشین اضافه نماییم. به این شکل:

```
a += [6, 7, 8]
```

و اکنون، طول آرایه نیز عوض شده است. شما تا عدد ۸ را به آرایه افزوده اید، و طبیعتا اگر **length** را اجرا کنید، نتیجه عدد ۸ است. همچنین، میتوانید با استفاده از اندیس (= نمایه) مناسب، نیز عضو مورد نظر را اضافه کنید. ما اکنون میخواهیم عدد ۹ را نیز به آرایه خود اضافه کنیم و چون میدانیم اندیس گذاری آرایه، از صفر شروع می شود، این دستور را اجرا میکنیم :

```
a[8] = 9
```

اکنون نوبت چاپ آرایه است. برای چاپ آرایه هم دستور زیر را اجرا میکنیم :

```
puts a
```

و چنین نتیجه ای دریافت خواهیم کرد :

```
irb(main):014:0> puts a
1
2
3
4
5
6
7
8
9
=> nil
irb(main):015:0>
```

چون **puts** برای چاپ خط جدید نیز به کار میرود، اینجا میگوید اولین عضو را چاپ کن، برو خط بعدی، دومی را چاپ کن، برو خط بعدی و الی آخر. به این شکل، هر کدام از اعضا درون یک خط جداگانه چاپ می شوند. برای چاپ یک عضو مشخص هم، تنها کافیست تا از اندیس مناسب استفاده کنید.

۶.۲.۴ جایگزین کردن اعضای آرایه

برای جایگزین کردن یک عضو، باید از اندیس آن استفاده کنید. مثلا میخواهیم عدد ۱ را با صفر جایگزین کنیم، میدانیم که ۱، اندیس صفر گرفته است پس این دستور را اجرا می کنیم :

```
a[0] = 0
```

اینگونه، عضو مورد نظر ما، با یک مقدار جدید جایگزین می‌شود . راه حل دیگری نیز وجود دارد که بتوانیم به صورت دسته ای، مقادیر را جایگزین کنیم. برای جایگزین کردن دسته ای، کافیست این گونه عمل کنیم :

```
a[0, 3] = 0, 1, 2
```

و با این دستور، سه عضو اول آرایه، تغییر میکنند. حالا دوباره دستور **puts** را اجرا میکنیم :

```
irb(main):019:0> puts a
0
1
2
4
5
6
7
8
9
=> nil
irb(main):020:0>
```

اکنون دیدید که مواردی که میخواستیم، به خوبی اجرا شدند. حالا نوبت آن است که نکاتی در مورد آرایه ها بدانید، و بعد برویم سراغ چندتایی ها و بعد هم هش ها.

۷.۲.۴ نکات مهم

- یادتان باشد که آرایه ها، تغییر پذیرند، میتوانید هر گاه که خواستید، اعضای آن را تغییر دهید.
- طول آرایه ها نیز تغییر پذیر است، یعنی میتوانید اعضایی که نیاز است را به آرایه اضافه کنید یا از آن کم کنید
- متدهایی که برای آرایه ها در نظر گرفته شده، برای رشته ها هم قابل استفاده است.

۳.۴ چندتایی ها

اگر پایتون کار کرده باشید، با یک شکل عمومی از چندتایی ها آشنا هستید. اما در روبی، چند تایی به آن شکل نداریم. در روبی، چند تایی ها مجموعه ای از نام ها و یک آرایه را دریافت میکنند، سپس تعدادی از اعضای آرایه (یا همه آن ها) را، به متغیرها نسبت میدهد. برای مثال، از آرایه اولیه مان یک چند تایی درست میکنیم :

```
(b, c, d, e, f) = a
```

و در صورت چاپ کردن هر کدام از متغیرهای تعریف شده، عضو نظیر در آرایه چاپ میگردد. در کل، همین قدر در مورد چندتایی ها بدانید، کافیست. اکنون بیایید به سراغ هش برویم.

۴.۴ هش ها

هش ها، رفتاری مانند دیکشنری دارند. این نوع داده ها، یک کلید [۳۱] و یک مقدار [۳۲] دریافت میکنند. در واقع، به ازای هر کلیدی، یک خروجی یکتا تولید میکنند. کاربرد هش ها، در ایجاد لیست ها، لغت نامه ها و ... است. هش ها هم همانند آرایه ها، یک سری متد خاص دارند که در این قسمت با آنها آشنا می شوید.

۱.۴.۴ ایجاد یک هش جدید

ساده ترین نوع هش، آنست که یک سری لغت را به زبان های مختلف بخواهیم ترجمه کنیم. اینجا میخواهیم واژه «سلام» را به زبان های انگلیسی، اسپانیایی، اسپرانتو و پرتغالی درون یک هش قرار دهیم :

```
salam = { :en => "Hello" , :es => "Hola", :eo => "Saluton", :pt => "Ola"}
```

بدین گونه، ما یک هش تازه ایجاد کرده ایم، و لغات مورد نظر را درونش قرار داده ایم. اکنون، برای نمایش لغت انگلیسی، کافیست کد زیر را اجرا کنیم :

```
salam[:en]
```

و سپس خروجی مناسبی دریافت خواهید کرد.

۲.۴.۴ ویرایش هش ها

هش ها هم تغییر پذیرند، منتها با تفاوتی بزرگ، که نمیتوان آنها را مانند آرایه ها ویرایش کرد. فرض کنیم شما نیاز دارید تا واژه را به زبان ایتالیایی هم اضافه کنید. اکنون باید کلید ایتالیایی را با مقدار مناسب، استفاده کنید، به این شکل :

```
salam[:it] = "Ciao"
```

و مقدار مورد نظر شما، به هش اضافه می شود.

---

[۳۱] Key
[۳۲] Value

۳.۴.۴ مرتب کردن هش

هش ها نیز مرتب می شوند. منتها، بر اساس کلید ها. اگر کلید ها عددی باشند، از کوچک به بزرگ، و اگر حرف باشند (همانند هش **salam** که ما ساختیم) به ترتیب حروف الفبا، مرتب می شوند. برای مرتب کردن هش همان دستور **sort** به کار می رود. دقت کنید که البته، هش ها بعد از مرتب شدن به آرایه تبدیل می شوند.

۴.۴.۴ چاپ کردن اعضای یک هش

چاپ کردن اعضای هش، بیشتر به ساختار حلقه های تکرار شبیه است، ولی باز با این حال در این قسمت اشاره کوچکی به آن میکنیم، برای چاپ کردن اعضای هش، از متد **each** استفاده میکنیم :

```
salam.each{|x, y| puts x + ": " y}
```

و نتیجه خروجی باید این چنین باشد :

```
irb(main):025:0> salam.each{|x, y| puts x + ": " + y}
en: Hello
es: Hola
eo: Saluton
it: Ciao
=> {"en"=>"Hello", "es"=>"Hola", "eo"=>"Saluton", "it"=>"Ciao"}
irb(main):026:0>
```

البته توجه کنید که کلید ها به این شکل تغییر کردند :

```
{"en"=>"Hello", "es"=>"Hola", "eo"=>"Saluton", "it"=>"Ciao"}
```

و به رشته تبدیل شدند، تا در چاپ کردن آنها مشکلی پیش نیاید.

۵.۴ جمع بندی

در این فصل، با آرایه و هش آشنا شدید. همچنین یاد گرفتید که مفهوم چندتایی در روبی کاملا با پایتون متفاوت است، و بنابراین برای یادگیری روبی، قطعا نیاز است تا در آموخته های پیشین خود، تجدید نظر کنید. همچنین یاد گرفتید که هش ها کجا کاربرد دارند (و بعد ها نیز بیشتر با کاربرد آنها آشنا خواهید شد) و همچنین در نکاتی که برای آرایه ها بیان شد، اشاره شد که رشته ها خواص مشابه آرایه ها دارند. [۳۳] در نهایت هم با مرتب کردن، و سایر متد های پرکاربرد درون روبی آشنا شدید.

[۳۳] فراموش نکنید که یک رشته، خود آرایه ای از کاراکتر هاست.

۶.۴ تمرینات

۱. برنامه ای بنویسید که شامل یک آرایه از حروف انگلیسی باشد، سپس توسط یک چندتایی، یک کلمه ایجاد کنید.

۲. برنامه ای بنویسید که لغات پر کاربرد انگلیسی را شامل شود، و کلید گذاری را توسط اعداد انجام دهید.

۳. برنامه ای بنویسید که نام دانش آموزان یک کلاس را درون یک هش شامل حروف الفبا باشد، و هر کلید به یک آرایه از اسامی نسبت داده شود.

۵ ساختارهای کنترلی

در این فصل، قصد داریم وارد مباحثی شویم، که شما را به «برنامه نویس» شدن نزدیک میکند. این فصل، بدون شک مهم ترین فصل این کتاب است و چنانچه این فصل را جدی نگیرید، با مشکلات بسیار جدی روبرو خواهید شد!پس این فصل را جدی بگیرید و تمام جزئیات اشاره شده را مو به مو اجرا کنید.

در این فصل، قرار است با ساختارهای کنترلی آشنا شویم. ساختارهای کنترلی و دستورات شرطی، مهم ترین بخش یک زبان برنامه نویسی به شمار می روند، میپرسید چرا؟ زیرا تمام کنترلی که باید روی ورودی و خروجی ها انجام شود، در این ساختار ها انجام می شود. در فصل بعدی خواهید دید چگونه با این ساختارها، یک تابع بازگشتی می توان نوشت و می توان تا چه حد، در زمان نوشتن برنامه طولانی، صرفه جویی نمود.

۱.۵ در این فصل چه فرا خواهید گرفت؟

- آشنایی با ساختارهای کنترلی و دستور if
- آشنایی با elsif و قرار دادن چند شرط برای برنامه
- استفاده از else
- استفاده از case به جای elsif

۲.۵ آشنایی با دستور if

دستور if در اکثر زبان های برنامه نویسی وجود دارد، پس اگر پیش زمینه ای از برنامه نویسی داشته باشید، میدانید که if چه میکند. if یک مقدار را به عنوان شرط دریافت میکند، و سپس اگر شرط برقرار بود، دستور درون بلاک را اجرا میکند. در غیر این صورت، برنامه اجرا نمی شود. شکل کلی دستور if در زبان روبی به این شکل است :

```
if CONDITION
STATEMENT
end
```

برای مثال، شما می توانید یک متغیر را بررسی کنید که در چه حیطه ای از اعداد قرار گرفته :

```
if n == 2
puts "True"
end
```

ما به متغیر n مقدار ۲ داده و در محیط تعامی دستورات فوق را اجرا میکنیم :

```
irb(main):001:0> n = 2
=> 2
irb(main):002:0> if n == 2
irb(main):003:1>    puts "True"
irb(main):004:1> end
True
=> nil
irb(main):005:0>
```

و این نیز نتیجه اجرای دستور if است. حال میخواهیم مقدار n را به ۳ تغییر داده، سپس همین دستور را اجرا کنیم، با این تفاوت که میخواهیم این بار که مقدار با ۲ متفاوت است، یک ارور دریافت کنیم. باید چه کنیم؟ قسمت بعدی جواب شماست!

۳.۵ آشنایی با دستور else

برنامه فوق را دیدیم. اکنون میخواهیم با else آشنا شده، و برنامه قسمت قبلی را تکمیل کنیم. لازم به ذکر است که else تمام کننده if است، گرچه if را به تنهایی می توان استفاده کرد، ولی برای نوشتن یک برنامه که کاربرپسند [۳۴] باشد، بهتر است که اگر ورودی نامناسب دریافت شد، یک خطا تولید کند و به کاربر بگوید ورودی مناسبی به برنامه نداده است. شکل کلی دستور else مانند مثال زیر است :

```
if CONDITION
STATEMENT
else
STATEMENT
end
```

پس بیایید برنامه فوق را دوباره بنویسیم، با این فرض که این بار مقدار n به ۳ تغییر داده شده و نیازمند تولید ارور هستیم.

```
if n == 2
puts "True"
else
puts "False"
end
```

پس از اجرا در محیط تعاملی، نیاز داریم تا خروجی را بررسی کنیم :

```
irb(main):005:0> n = 3
=> 3
irb(main):006:0> if n == 2
```

[۳۴] User Friendly

```
irb(main):007:1>    puts "True"
irb(main):008:1> else
irb(main):009:1*    puts "False"
irb(main):010:1> end
False
=> nil
irb(main):011:0>
```

همانطور که ملاحظه کردید، این بار خروجی ما کلمه **False** است. حالا میخواهیم برنامه را طوری تنظیم کنیم که هم ۲ و هم ۳ را به عنوان خروجی مناسب بپذیرد. در بخش بعدی این مورد را بررسی خواهیم کرد.

۴.۵ آشنایی با elsif و به کارگیری همزمان چند شرط

برنامه ای که از ابتدای فصل نوشتیم را یک بار دیگر چک کنید، بار اول به متغیر **n** مقدار ۲ دادیم، و خروجی **if** را بررسی کردیم، سپس رفتیم به سراغ استفاده از **else** و آنجا، دیدیم که چگونه می توان یک خطای مناسب تولید کرد. این بار میخواهیم سه عدد ۲، ۳ و ۴ را به عنوان ورودی مناسب قبول کنیم، و در صورتی که ورودی نامناسب بود، یک ارور را چاپ کنیم. راه حل استفاده از دستور **elsif** است. شکل کلی دستور **elsif** این است :

```
if CONDITION0
STATEMENT
elsif CONDITION1
STATEMENT
elsif CONDITION2
STATEMENT
.
.
.
else
STATEMENT
end
```

دقت کنید که در تعداد **elsif** ها مختارید و هرچند تا که بخواهید را می توانید به کار ببرید، اما معمولا موقعی که تعداد کمی شرط داریم، از **elsif** استفاده میکنیم و برای تعداد شروط بیشتر، از **case** استفاده میکنیم که در بخش بعدی مورد بررسی قرار میگیرد.

حالا بیایید برنامه را دوباره بنویسیم. این بار مقدار **n** را تغییر نمی دهیم و برنامه را به این شکل مینویسیم :

```
if n == 2
puts "True, #{n} is standard input"
```

```
elsif n == 3
puts "True, #{n} is standard input"
elsif n == 4
puts "True, #{n} is standard input"
else
puts "False, #{n} is not in range of standard inputs"
end
```

و اکنون با اجرای این برنامه در محیط تعاملی، خواهیم داشت :

```
irb(main):013:0> if n == 2
irb(main):014:1>  puts "True, #{n} is standard input"
irb(main):015:1> elsif n == 3
irb(main):016:1>  puts "True, #{n} is standard input"
irb(main):017:1> elsif n == 4
irb(main):018:1>  puts "True, #{n} is standard input"
irb(main):019:1> else
irb(main):020:1* puts "False, #{n} is not in range of standard inputs"
irb(main):021:1> end
True, 3 is standard input
=> nil
irb(main):022:0>
```

اکنون با ساختار elsif هم آشنا شدیم. اما گاهی پیش می آید که میخواهیم شروط بیشتری را برای یک متغیر بررسی کنیم و آنگاه مجبوریم از case بهره بجوییم. این دستور هم در بخش بعدی مورد بررسی قرار می دهیم.

۵.۵ استفاده از case برای بررسی چندین شرط

گاهی اوقات بقدری شروط ما زیادند که نمی توان از elsif استفاده کرد. پس بنابراین بهتر است یک راه حل بهتر بیابیم. دستور case اینجا به کمک ما می آید. یکی از ویژگی های خوب دستور case اینست که به ما اجازه میدهد چندین شرط را بررسی کرده و در صورت مناسب نبودن مقدار ورودی، یک ارور مناسب چاپ کنیم. شکل کلی case در روبی این چنین است :

```
case VARIABLE
when CONDITION1
STATEMENT
when CONDITION2
STATEMENT
.
.
```

.
else
STATEMENT

همان برنامه فوق را در نظر بگیرید. برای این که با نوع داده ای جدیدی هم آشنا شوید، این بار میگوییم اگر n بین ۱ تا ۵ بود یک پیغام، اگر بین ۵ تا ۱۰ بود پیغام دیگر، و اگر خارج از محدوده بود یک ارور چاپ شود. ما اینجا از نوع داده ای رنج استفاده میکنیم. یک رنج مثلا از یک تا ده به این شکل تعریف می گردد

```
1..10
```

و اگر میخواهید ماکزیمم مقدار، جزئی از خود رنج نباشد :

```
1...10
```

اکنون، یک بار دیگر الگوریتم را بررسی کنید. گفتیم که مقدار n بین ۱ تا ۵ و بار دیگر، در بازه ۵ تا ۱۰ بررسی شود. اکنون کد ما به این شکل خواهد بود :

```
case
when 1..5
puts "True, #{n} is in range"
when 5..10
puts "True, #{n} is in range"
else
puts "False, #{n} is not in range"
end
```

این برنامه را در محیط تعاملی اجرا میکنیم و طبیعتا باید چنین نتیجه ای بگیریم :

```
irb(main):028:0> case n
irb(main):029:1> when 1..5
irb(main):030:1>  puts "True, #{n} is in range"
irb(main):031:1> when 5..10
irb(main):032:1>  puts "True, #{n} is in range"
irb(main):033:1> else
irb(main):034:1*  puts "False, #{n} is not in
range"
irb(main):035:1> end
True, 3 is in range
=> nil
irb(main):036:0>
```

و این برنامه، همان چیزی بود که از ابتدا نیاز داشتیم. حال میتوانید با elsif و case هم برنامه نویسی کنید و قطعا برنامه ای که این اصول درش رعایت شده باشد، برنامه بهتر و کاربردی تری است.

۶.۵ جمع بندی

در این فصل، با ساختارهای کنترلی آشنا شدید. دانستید که چگونه، با استفاده از if و else یک شرط ساده را بررسی کنید. همچنین دانستید که برای بررسی چندین شرط، باید از elsif استفاده کرد و چنانچه شروط زیاد و یا خاص باشند، case هم به کمک شما می آید. در نهایت باید گفت که این فصل، مهم ترین فصل کتاب بود و فصول بعدی در تکمیل این فصل هستند.

۷.۵ تمرینات

۱. **برنامه ای بنویسید که سن شما را دریافت کرده، در خروجی پیامی چاپ کند که سن شما قانونی است یا نه.**

۲. **در تکمیل برنامه سوال قبل، برنامه را طوری بنویسید که اگر سن شما قانونی نبود، بگوید چند سال تا سن قانونی باقی است.**

۳. **برنامه ای بنویسید که نمرات امتحانی را دریافت کرده، اگر نمره زیر ۱۰ بود عبارت Failed و اگر بالای ۱۰ بود عبارت Passed را چاپ کند.**

۴. **برنامه ای بنویسید که معدل ترم شما را دریافت کرده، چنانچه زیر ۱۲ بود عبارت Failed و چنانچه بالای ۱۲ بود عبارت Passed را چاپ کند، همچنین اگر معدل بالای ۱۷ بود عبارت** Allowed to choose 24 units **را نمایش دهد.**

۶ حلقه های تکرار

در فصول پیش، مفاهیمی که برای کار با روبی لازم بود را فرا گرفتید. اکنون نوبتی هم باشد، نوبت حلقه های تکرار است. بدون شک هر زبان برنامه نویسی ساخت یافته ای، یا حلقه های تکرار را در خود جای داده یا اینکه راهی برای پیاده سازی آنها، برای برنامه نویس فراهم کرده است.حلقه های تکرار، میتوانند در هنگامی که نیاز است در خروجی، تعداد زیادی متغیر را چاپ کنیم، هنگامی که در ورودی چندین اسم دریافت کنیم و ... کاربرد فراوانی دارند. روبی هم یک زبان کامل و مدرن است، طبیعی است تا حلقه های تکرار در روبی را فرا نگیرید، نمیتوانید برنامه های مورد نظرتان را به خوبی پیاده کنید.

۱.۶ در این فصل چه فراخواهید گرفت؟

- **استفاده از حلقه while**
- **استفاده از حلقه for**
- **استفاده از حلقه until**

۲.۶ حلقه while

این حلقه، پرکاربرد ترین حلقه در روبی است. در واقع، این حلقه همان عملکرد while در سایر زبان ها را دارد. عملکرد کلی این حلقه، به این شکل است که یک شرط درست را دریافت کرده، تا زمانی برنامه را ادامه میدهد که شرط غلط شود. پیاده سازی این حلقه در روبی بسیار آسان است. فقط کافیست متغیری که در نظر دارید را از قبل تعریف نموده، سپس شرطی ایجاد کنید و بعد مقدار متغیر را درون حلقه دستخوش تغییر کنید. بیایید با یک دیگر اعداد صفر تا ده را روی صفحه چاپ کنیم :

```
n = 0
while n <= 10
puts n
n += 1
end
```

پس از اجرای این کد در محیط تعاملی چنین نتیجه ای را شاهد هستیم :

```
irb(main):004:0> n = 0
=> 0
irb(main):005:0> while n <= 10
irb(main):006:1>    puts n
irb(main):007:1>    n += 1
irb(main):008:1> end
0
```

```
1
2
3
4
5
6
7
8
9
10
=> nil
irb(main):009:0>
```

ابتدا به مفسر، گفته می شود مقدار صفر را برای n لحاظ کند. سپس ، آن را چاپ کرده، یک واحد به آن بیافزاید. حلقه چک میکند که مقدار n با شرط همخوانی دارد یا نه. وقتی مقدار n به ۱۰ میرسد، حلقه قطع می شود، چرا که مقدار بعدی با شرط ما همخوانی ندارد.

۱.۲.۶ مثال کاربردی تر از حلقه while

این مثال، مثال بهتری است، چرا که تا حدود زیادی، عملیاتی است. فرض کنیم شما معلمی هستید که ۱۵ دانش آموز دارد. حالا میخواهید نمرات دانش آموزان را وارد یک آرایه کنید. بیایید کدی بنویسیم که این کار را برای ما انجام دهد :

```
n = 1
scores = []
while n <= 15
print "Enter student #{n}'s score: "
score = gets.chomp
scores[n] = score.to_i
n += 1
end
```

بیاید این قطعه کد را درون محیط تعاملی، اجرا کنیم، و نتیجه را ببینیم :

```
irb(main):013:0> while n <= 15
irb(main):014:1>  print "Enter student #{n}'s score: "
irb(main):015:1>  score = gets.chomp
irb(main):016:1>  scores[n] = score.to_i
irb(main):017:1>  n += 1
irb(main):018:1> end
Enter student 1's score: 20
```

```
Enter student 2's score: 18
Enter student 3's score: 14
Enter student 4's score: 15
Enter student 5's score: 18
Enter student 6's score: 18.5
Enter student 7's score: 20
Enter student 8's score: 12.25
Enter student 9's score: 14
Enter student 10's score: 20
Enter student 11's score: 12
Enter student 12's score: 10
Enter student 13's score: 8.5
Enter student 14's score: 14
Enter student 15's score: 20
=> nil
irb(main):019:0>
```

همانطور که مشخص است، ابتدا یک prompt به کاربر نشان داده می شود، و سپس یکی یکی نمرات دانش آموزان خواسته می شود، و نمرات به عضویت یک آرایه در می آیند. در این بخش :

```
irb(main):015:1>  score = gets.chomp
irb(main):016:1>  scores[n] = score.to_i
```

چون عملیات gets ورودی را به صورت رشته دریافت میکند، آن را به عدد صحیح تبدیل کرده و درون آرایه ریخته ایم (توجه کنید که نمراتی مانند ۱۲/۲۵ و ... هم به عدد صحیح تبدیل شده اند!).

۲.۲.۶ حلقه بی نهایت

حلقه ای که شرط همیشه درست دریافت کند، تا بی نهایت ادامه دارد و تنها با کلید های ترکیبی Ctrl + C میتوان متوقف کرد، یا دستوراتی تعریف کرد که با دریافت یک ورودی خاص، حلقه را متوقف کنند. مثلا، حلقه ای که مجموع تعداد نامعلومی عدد را حساب میکند :

```
while n != 0
 n = gets.chomp
 n = n.to_i
 n += m
end
```

برای مثال، حلقه بالا میگوید تا موقعی که متغیر n مخالف صفر است حلقه ادامه یابد، و چون متغیر درون خود حلقه به عنوان ورودی استفاده شده است، وقتی کاربر مقدار صفر را وارد کند حلقه

تمام می شود. مثال بالا، به قدر کافی کاربردی نیست. بیایید رابط خط فرمان [۳۵] را با روبی بازسازی کنیم :

```
while true
 print ">> "
 cmd = gets.chomp
 system cmd
end
```

اکنون برنامه را اجرا کرده و چندین دستور مربوط به خط فرمان سیستم عامل لینوکس را اجرا میکنم :

```
irb(main):001:0> while true
irb(main):002:1>  print ">> "
irb(main):003:1>  cmd = gets.chomp
irb(main):004:1>  system cmd
irb(main):005:1> end
>> uname -srm
Linux 3.13.0-37-generic x86_64
>> uptime
 20:43:41 up 4 days, 23:30,  2 users,  load average: 1.18, 1.06, 0.97
>> echo 'Hello, World'
Hello, World
>> uname
Linux
>> sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
            [command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
            prompt] [-u user] [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
            prompt] [-u user] file ...
>> exit
>> exit
>> ^CIRB::Abort: abort then interrupt!
from (irb):3:in `call'
from (irb):3:in `gets'
```

[۳۵] Command Line Interface a.k.a CLI

```
from (irb):3:in `gets'
from (irb):3
from /usr/bin/irb:12:in `<main>'
irb(main):006:0>
```

همانگونه که دیدید ، با استفاده از تابع کتابخانه ای **system** تعدادی ورودی را درون خط فرمان لینوکس اجرا نمودیم. حالا نوبتی هم باشد، نوبت توضیح کد است. این کد، یک حلقه بی نهایت را نشان میدهد، که یک **prompt** چاپ میکند و از کاربر ورودی میخواهد، کاربر دستوری را تایپ میکند که به شکل یک رشته، به داخل **cmd** رفته و سپس **cmd** را به عنوان ورودی به **system** میدهد. این دستور در آخر با **Ctrl-C** متوقف شده است.این برنامه میتواند تمرین خوبی برای طراحی ابزارهای کاربردی باشد.

۳.۶ حلقه until

این حلقه، بر خلاف حلقه **while** ، یک شرط غلط را به عنوان ورودی دریافت کرده، سپس حلقه را تا جایی ادامه میدهد که شرط درست شود. برای این که کدی با این حلقه بنویسیم، بیایید از آرایه ای به نام **scores** که در بخش قبل ایجاد کردیم، را با استفاده از این حلقه، چاپ کنیم.

```
n = 1
until n == scores.length
 puts scores[n]
 n += 1
end
```

نتیجه اجرای این کد :

```
irb(main):008:0> n = 1
=> 1
irb(main):009:0> until n == scores.length
irb(main):010:1>  puts scores[n]
irb(main):011:1>  n += 1
irb(main):012:1> end
20
18
14
15
18
18
20
12
14
```

```
20
12
10
8
14
20
=> nil
irb(main):013:0>
```

همانگونه که مشاهده میکنید، تمامی اعضا چاپ شده اند و اعضایی که خرده داشتند، به عدد صحیح تبدیل شده اند. تنها تفاوت **while** و **until** در این است که **while** یک شرط درست را تا غلط شدن ادامه میدهد، و **until** یک شرط غلط را تا درست شدن.

۱.۳.۶ حلقه بی نهایت با until

این حلقه، برخلاف **while** مقادیر **true** را نمیتواند پذیرا باشد. پس سولوشن مناسب برای ایجاد حلقه بی نهایت و نوشتن برنامه خط فرمان استفاده از مقدار **false** است. پس به این شکل برنامه را باز نویسی میکنیم :

```
until false
 print ">> "
 cmd = gets.chomp
 system cmd
end
```

اجرای این برنامه نیز بر عهده خودتان، به عنوان یک تمرین خوب.

۴.۶ حلقه for

اگر با زبانی مانند C آشنا باشید، قطعا میدانید که حلقه **for** در آن زبانها، به نوعی یک **while** تعمیم یافته است. اما در روبی این طور نیست. در روبی، حلقه **for** برای کار با داده های از نوع رنج و ... به کار می رود. بیایید چاپ اعداد صفر تا ده را با استفاده از حلقه **for** بررسی کنیم.

```
for i in 0..10
 puts i
end
```

نتیجه اجرای این دستور چنین است :

```
irb(main):016:0> for i in 1..10
irb(main):017:1>  puts i
irb(main):018:1> end
```

```
1
2
3
4
5
6
7
8
9
10
=> 1..10
irb(main):019:0>
```

اصول کار حلقه for در روبی این است که متغیری که به عنوان ورودی دریافت کرده است (اینجا i) را یکی یکی با اعضای آرایه یا رنج ورودی برابر قرار میدهد.

۱.۴.۶ استفاده از each به جای for

یک بار دیگر، بخش آرایه ها و هش ها را مطالعه کنید. آنجا با استفاده از دستور each ، اعضای یک هش را چاپ نمودیم. اکنون، اینجا دوباره آن را بررسی میکنیم. برای امتحان دوباره کد زیر را وارد کنید :

```
dic = {1 => "One", 2 => "Two", 3 => "Three", 4 => "Four", 5 => "Five"}
dic.each{|x, y| puts "#{x} : #{y}"}
```

حاصل اجرای دستور فوق در محیط تعاملی باید چنین باشد :

```
irb(main):027:0> dic = {1 => "One" , 2 => "Two", 3 => "Three", 4 => "Four", 5 => "Five"}
=> {1=>"One", 2=>"Two", 3=>"Three", 4=>"Four", 5=>"Five"}
irb(main):028:0> dic.each{|x, y| puts "#{x} : #{y}"}
1 : One
2 : Two
3 : Three
4 : Four
5 : Five
=> {1=>"One", 2=>"Two", 3=>"Three", 4=>"Four", 5=>"Five"}
irb(main):029:0>
```

در واقع دستور each نوع خیلی سفارشی شده for برای آرایه ها و هش ها به شمار می آید.

۵.۶ جمع بندی

در این فصل، با مباحث پیچیده و پیشرفته ای روبرو شدید. این فصل مقدمه بسیار خوبی برای فصول بعدی است که قرار است در مورد توابع و کلاس ها صحبت شود. در واقع با استفاده از این فصل، شما میتوانید برنامه هایی بنویسید که کاربردی تر از پیش هستند. در این فصل، دیدید که چقدر ساده می توانید با یک حلقه بی نهایت، عملیات خط فرمان لینوکس را شبیه سازی کنید، یا لیستی از نمرات دانش آموزان خود تهیه کنید. با استفاده از نکات این فصل، درک مفاهیم فصول بعدی برایتان بسیار آسان تر خواهد بود.

۶.۶ تمرینات

۱. **برنامه ای بنویسید که در یک آرایه، نام دانش آموزان و در آرایه دیگر نمرات آنان را دریافت کند. سپس برنامه ای بنویسید که نام و نمرات آنان را با یک تب (۴ فاصله) فاصله از هم چاپ کند.**

۲. **برنامه ای بنویسید که یک کلید دریافت کرده، آن را با خط بعدی که معنای کلمه کلیدی است، در یک هش قرار دهد. سپس آنها را لغت نامه وار چاپ کند.**

۳. **حروف نام خود را به صورت وارونه چاپ کنید.**

۷ توابع

در فصول قبل، با تمامی مبانی برنامه نویسی در روبی آشنا شدیم. اکنون، نیاز به این داریم تا یاد بگیریم چگونه، برنامه خود را به چند قسمت تقسیم کنیم. تقسیم کردن برنامه به چند برنامه کوچک تر، خود کمک بزرگی است تا بازبینی در کدها، بعدها آسان تر شود. در این فصل، با مفهوم تابع آشنا خواهید شد. همچنین، انواع مختلفی از توابع را خواهید نوشت و سپس با کمک توابع، برنامه های مختلفی را میتوانید بنویسید.

۱.۷ در این فصل چه فراخواهید گرفت؟

- **مفهوم تابع**
- **ایجاد تابع خالی**
- **ایجاد تابع ساده**
- **ایجاد تابع بازگشتی**
- **استفاده از توابع تو در تو**

۲.۷ مفهوم تابع

تابع [۳۶] قسمتی از برنامه است که رفتار جعبه سیاه [۳۷] گونه دارد. در واقع، یک متغیر به عنوان ورودی به تابع داده می شود، سپس دور از چشم کاربر عملیاتی روی تابع انجام شده، برنامه مقداری را به کاربر بر میگرداند. برنامه نویس، خودش میداند درون تابعش چه میگذرد، اما کسی که برنامه را اجرا میکند خبری ندارد. بنابراین، میگوییم تابع همانند جعبه سیاهی برای ورودی ها می ماند.

۳.۷ تعریف تابع در روبی

برای تعریف یک تابع در روبی به این شکل عمل میکنیم:

```
def FunctionName(arguments)
...
end
```

۴.۷ تابع خالی

تابع خالی، یا تابع تهی [۳۸] ساده ترین نوع توابع هستند. این توابع،با استفاده از دستوراتی مانند puts یا print کار کرده و مقداری را بر نمی گردانند. برای مثال تابع زیر یک نمونه تابع خالی است :

[۳۶] Function
[۳۷] Black Box
[۳۸] Void

```
def PrintName()
name = gets
puts name
end
```

پس از اجرای قطعه کد بالا در محیط تعاملی، چنین خروجی را شاهد هستیم :

```
irb(main):001:0> def PrintName()
irb(main):002:1>   name = gets
irb(main):003:1>   puts name
irb(main):004:1> end
=> nil
irb(main):005:0> PrintName()
Muhammadreza
Muhammadreza
=> nil
irb(main):006:0>
```

همانطور که دیدید، با صدا زدن [۳۹] یک تابع، آن تابع اجرا شد. این توابع کاربردشان دریافت ورودی و یا ارسال خروجی است و معمولا با توابع دیگر، به صورت ترکیبی استفاده می شوند.

۵.۷ تابع ساده

تابع ساده، یک تابع است که مقداری را برای ما بر میگرداند. برگشت مقادیر توسط دستور return انجام می شود. کد زیر، ساده ترین حالت یک تابع است که همان آرگومان دریافتی را باز میگرداند :

```
def SampleFunction(i)
return i
end
```

برای این که مقدار این تابع توسط یک برنامه برای کاربر چاپ شود، از دستوراتی مانند puts یا print استفاده می کنیم. نمونه زیر، اجرای کد بالاست :

```
irb(main):006:0> def SampleFunction(i)
irb(main):007:1>     return i
irb(main):008:1> end
=> nil
irb(main):009:0> puts SampleFunction(2)
2
=> nil
irb(main):010:0>
```

[۳۹] call

همچنین، مقدار برگردانده شده، می تواند آرگومان دریافتی نباشد. معمولا تعدادی آرگومان توسط کاربر به تابع داده می شود، ممکن است درون تابع، متغیری ایجاد شده باشد که ماحصل کار روی آرگومان ها درون آن قرار گرفته باشند. بدین ترتیب، میتوان ترتیبی داد که آن متغیر بازگردانده شود. اکنون تصور کنید میخواهیم برنامه ای بنویسیم که یک عدد را به تابعی بدهیم، سپس آن تابع از صفر، تا آن عدد را برای ما چاپ کند. تابع ما چنین می شود :

```
def Range(j)
i = 0
while i <= j
      puts i
      i += 1
end
return i
end
```

این تابع، به این شکل اجرا می شود :

```
irb(main):010:0> def Range(j)
irb(main):011:1>  i = 0
irb(main):012:1>  while i <= j
irb(main):013:2>    puts i
irb(main):014:2>    i += 1
irb(main):015:2>  end
irb(main):016:1>  return i
irb(main):017:1> end
=> nil
irb(main):018:0> puts Range(2)
0
1
2
3
=> nil
irb(main):019:0>
```

همانگونه که میبینید، تابع ما مقدار ۳ را برگردانده، حال آنکه ما مقدار ۲ را به تابع داده بودیم. دستور reutrn ، به این شکل عمل میکند که آخرین مقدار لحاظ شده توسط برنامه برای یک متغیر را بر میگرداند. البته توجه کنید که اگر بجای علامت «کوچکتر مساوی» از علامت «کوچکتر» استفاده کنید، مقدار درست بر میگردد.

۶.۷ تابع بازگشتی

تابع بازگشتی، نه تنها مبحث مهمی در برنامه نویسی است، بلکه مبحث بسیار مهمی در ریاضیات نیز هست. توابعی همچون فاکتوریل یا فیبوناچی، توابع بازگشتی هستند. در تابع بازگشتی، «با علم به آن که خروجی چند مقدار خاص را میدانیم، سایر خروجی ها را تعیین میکنیم». یکی از ساده ترین مثال ها، برنامه ایجاد خطوط خالی است. برای ایجاد یک خط خالی، از دستور puts میتوان استفاده برد. اما اگر ۵ خط خالی نیاز باشد چه؟ بهتر است ابتدا کد زیر را در محیط تعاملی امتحان کنیم :

```
def nLines(n)
    if n > 0
       puts
       nLines(n-1)
    end
end
```

قبل از اجرای تابع، بیایید عملکرد آن را بررسی کنیم. تابع مقدار n را به عنوان ورودی میگیرد، سپس چک میکند که بزرگتر از صفر است یا نه. سپس، یک خط خالی ایجاد میکند. در حافظه کامپیوتر، یکی از مقدار n کم میشود، سپس دوباره n چک می شود و الی آخر. بدین شکل تا موقعی که n مقدارش با صفر برابر گردد، تابع ادامه دارد. حال نتیجه اجرای دستور برای چاپ ۳ خط خالی :

```
irb(main):019:0> def nLines(n)
irb(main):020:1>     if n > 0
irb(main):021:2>        puts
irb(main):022:2>        nLines(n-1)
irb(main):023:2>     end
irb(main):024:1> end
=> nil
irb(main):025:0> nLines(3)



=> nil
irb(main):026:0>
```

یکی از کاربردهای تابع بازگشتی، در این زمینه است. گاهی هم بازسازی فرمولهایی که در ریاضیات استفاده میکنیم، میتوانند موضوعات جالبی برای نوشتن برنامه با استفاده از تابع بازگشتی باشند. بیایید با هم تابع فاکتوریل را بررسی کنیم. اولین چیزی که باید بدانیم، اینست که تابع فاکتوریل، فقط برای اعداد طبیعی صدق میکند، پس فرمول کلی تابع فاکتوریل برای اعداد طبیعی بدین شکل خواهد بود :

$$f(0) = 1$$
$$f(n) = n \times (n-1)!$$

حالا بیایید برای تابع خود، یک تابع در روبی بنویسیم. کد تابع ما به این شکل خواهد بود :

```
def factorial(n)
    if n == 0
       return 1
    else
       return n * factorial(n-1)
    end
end
```

این کد هم همانند قبلی، آنقدر مقدار n را چک میکند تا به صفر برسد. نتیجه اجرای همین تابع برای چند عدد :

```
irb(main):026:0> def factorial(n)
irb(main):027:1>     if n == 0
irb(main):028:2>        return 1
irb(main):029:2>     else
irb(main):030:2*        return n * factorial(n-1)
irb(main):031:2>     end
irb(main):032:1> end
=> nil
irb(main):033:0> factorial(0)
=> 1
irb(main):034:0> factorial(12)
=> 479001600
irb(main):035:0> factorial(4)
=> 24
irb(main):036:0>
```

و به همین شکل، می توانید توابع بازگشتی دیگری نیز بنویسید.

۷.۷ توابع تو در تو

یکی از خواص زبان روبی، اینست که میتوانید چندین تابع را درون یکدیگر ایجاد کنید، و سپس هر موقع لازم شد، یکی را به دلخواه خود صدا بزنید. در واقع یک تابع، می تواند درون یک تابع دیگر باشد و در عین حال، تابع دیگری را نیز درون خودش جای دهد. این قابلیت، در اکثر زبان های اسکریپتی وجود دارد، اما روبی یکی از بهترین پیاده سازی های این روش را ارائه کرده است. برای درک بهتر از توابع تو در تو، دستور System.out.println که در زبان جاوا به کار میرود را به این شکل بازسازی میکنیم :

```
def system
    def out
        def println(str)
```

```
            puts str
        end
    end
end
```

توجه کنید که نام تابع باید با حروف کوچک شروع گردد، به عبارتی نام تابع نمی تواند ثابت باشد.

اکنون این برنامه را اجرا کرده و شاهد چنین نتیجه ای خواهیم بود :

```
irb(main):001:0> def system
irb(main):002:1>     def out
irb(main):003:2>         def println(str)
irb(main):004:3>             puts str
irb(main):005:3>         end
irb(main):006:2>      end
irb(main):007:1> end
=> nil
irb(main):008:0> system.out.println("Hello, World")
Hello, World
=> nil
irb(main):009:0>
```

توابع داخلی تر، با نقطه از تابع اصلی جدا میگردند، به این شکل شما میتوانید یک تابع بزرگ بنویسید که خودش از چند تابع کوچکتر تشکیل شده باشد، سپس هر کجا نیاز به آن تابع شد، آن را با بخش های درونش صدا بزنید.

۸.۷ جمع بندی

در این فصل، کار با توابع را فراگرفتید. هر تابعی، به نحوی میتواند به بهتر شدن برنامه شما، کمک کند. این فصل، مقدمه ای شده است برای فصل بعدی، در فصل بعدی از شی گرایی سخن به میان می آید و چنانچه قادر به کار با توابع نباشید، یادگیری شی گرایی به شدت برایتان سخت خواهد شد. در دنیای مدرن، بیشترین وابستگی برنامه نویسی و ابزارهای موجود به توابع است. امروزه کمتر زبانی را پیدا میکنید که از توابع پشتیبانی نکند و مجبور باشید همانند قدیم، مفهوم تابع را خودتان به یک زبان استفاده کنید. پس این فصل، فصل بسیار مهمی است. در این فصل همچنان چندین نوع مختلف تابع را بررسی کردیم، این چند نوع انواعی هستند که در روبی بیشترین کاربرد را دارند. با جست و جوی درست در منابع مربوط به روبی، شما قادر خواهید بود تا با انواع دیگری از توابع هم آشنا شوید.

۹.۷ تمرینات

۱. **برنامه ای بنویسید که شامل یک تابع باشد، تابع آرایه ای را به عنوان ورودی دریافت کرده، آن را مرتب کرده و اعضایش را نمایش دهد.**

۲. برنامه ای بنویسید که اعضای یک آرایه ده عضوی را از کاربر دریافت کرده، آن را مرتب کرده و امکان جست و جوی مقدار درون آرایه هم باشد. این برنامه را در قالب سه تابع بنویسید.

۳. دنباله فیبوناچی را به تابع تبدیل کنید.

۴. چندین دستور زبان جاوا را به صورت تابع تو در تو در آورده، در قالب یک برنامه روبی عرضه کنید.

۸ کلاس ها و شی گرایی

از ابتدای کتاب، مفاهیم زیادی را فراگرفته اید که در فصول بعدی، مورد استفاده بوده اند. اینجا، قرار است با مفهوم شی گرایی آشنا شویم. در واقع اینجا قرار است مزرعه بزرگی که از اول کتاب ایجاد شده است، درو کنیم. اکنون برای آنکه ثمره تلاش خود از اول کتاب را ببینید، میتوانید تمرینات را حل کنید و ببینید چقدر در کد نویسی روبی، حرفه ای شده اید. اما اکنون، میخواهیم با مفاهیم تازه آشنا شویم. شی گرایی، مفهومی بود که پس از ساخت یافتگی برای برنامه نویسی مطرح شد. در این فصل، مفاهیم و طرز پیاده سازی آن در روبی، به تفصیل توضیح داده خواهد شد.

۱.۸ در این فصل چه فراخواهید گرفت؟

- **آشنایی با شی گرایی**
- **آشنایی با خواص شی گرایی**
- **آشنایی به انتزاع**
- **آشنایی با وراثت**
- **آشنایی با چند شکلی**
- **آشنایی با کپسوله سازی**
- **پیاده سازی کلاس ها و اشیاء**
- **شی گرایی در روبی**

۲.۸ آشنایی با شی گرایی

اول باید ببینیم در برنامه نویسی شی گرا با چه چیزی سر و کار داریم؟ خواص زبانهایی همچون روبی و یا پایتون ایجاب میکند که شی گرایی در این زبان ها را فرا بگیرید. در روبی، اصولا هر چه می بینید، یک شی از یک کلاس محسوب می شود. برای مثال، میتوانید با متد class ببینید نوع داده ای یک متغیر چیست :

```
a = 2
a.class
```

و با اجرای دستورات فوق در محیط تعاملی چنین نتیجه ای را شاهد هستیم :

```
irb(main):001:0> a = 2
=> 2
irb(main):002:0> a.class
=> Fixnum
irb(main):003:0>
```

پس به این شکل، میتوان دریافت که روبی، همه چیز را شیء می بیند. اما شی گرایی به معنای ایجاد اشیاء جدید را در این فصل، تجربه خواهید کرد.

۳.۸ آشنایی با مفهوم انتزاع

نوع داده انتزاعی [۴۰] نوعی از خواص ساختاری است که برای چندین شی مشترک است. برای مثال، میگوییم گربه، و نژاد و رنگ و وزن و رنگ چشم آن را مشخص نمی کنیم، اما ویژگی های مشترکی برای همه گربه ها قائل هستیم. مثلا، همه گربه ها میو میو میکنند و یا گوشتخوار هستند. و «گوشتخوار بودن» و «میو میو کردن» را از ویژگی همه گربه ها میدانیم، نه یک نژاد خاص. در صورتی که نژاد، رنگ و ... می تواند از گربه ای به گربه دیگر، کاملا متفاوت باشد.

۴.۸ آشنایی با وراثت

وراثت [۴۱] یعنی یک کلاس کلی داریم، و کلاسی جدید ایجاد میکنیم که ویژگی های آن را نیز داراست. مثلا، گربه ایرانی، نوعی از گربه است که در کنار ویژگی های خودش، ویژگی های کلاس والد [۴۲] را نیز دارد. به کلاسی که بخشی از ویژگی هایش را از کلاس دیگر به ارث برده است، کلاس فرزند [۴۳] می گویند.

۵.۸ آشنایی با چندشکلی

چندشکلی یا چندریختی [۴۴] به این گفته می شود که یک شی را، میتوان به چندین شکل مختلف نشان داد. برای مثال، یک کلاس کلی حیوان وجود دارد، و گربه هم نوعی حیوان است. زنبور و گاو و سگ هم حیوان هستند و هرکدام شکلی از کلاس «حیوان» به شمار می آیند. پس وقتی میگوییم «گربه» ، حتما منظورمان «حیوان» است، ولی وقتی میگوییم «حیوان»، میتوانیم به گربه، زنبور، گاو و یا سگ اشاره کرده باشیم.

۶.۸ آشنایی با کپسوله سازی

کپسوله سازی [۴۵] در علم رایانه، به معنای آن است که بعضی فرآیند ها طوری باشند که کاربر، فقط طرز کار آنها را ببیند، و نتواند تغییری در آن ایجاد کند.این ویژگی، این امکان را میدهد که چندین فرآیند در کنار هم بدون مشکل کار کنند، و کاربر نتواند آن ها را منوط به چگونگی انجام کار کند. در مسائل امنیتی، برای مثال زمانی که قرار است برنامه اطلاعاتی را درون یک دیتابیس بنویسد، این امکان می تواند بسیار مفید باشد.

---

[۴۰] Abstract Data Type
[۴۱] Inheritance
[۴۲] Parent
[۴۳] Child
[۴۴] Polymorphism
[۴۵] Encapsulation

۷.۸ پیاده سازی کلاس ها و اشیاء

پیاده سازی کلاس ها و اشیا، در هر زبان با زبان دیگر، تفاوت دارد. در واقع باید بدانیم که دو آیتم کلیدی در هر کلاس می توان تعریف نمود :

۱. متد ها

۲. متغیر نمونه

متد [۴۶] تابعی است که در دل یک کلاس قرار گرفته، با فرا خوانی اجرا می گردد. یک متد میتواند **public** یا عمومی باشد ، یا **private** باشد، در اصطلاح کپسوله شده باشد.
متغیر نمونه [۴۷] هم متغیری است که درون کلاس تعریف شده و میتوان بدون فراخوانی دوباره درون متدها از آن استفاده نمود.

۸.۸ شی گرایی در روبی

قبل از آنکه کلاس ها را فرا بگیرید، بیاید با مفهوم ماژول آشنا شویم. ماژول ها کلاس نیستند، بلکه نوعی ساختار [۴۸] به شمار می آیند. ماژول ها را میتوان معادل **struct** در زبانهایی مانند **C** به شمار آورد، با این تفاوت که ماژول ها میتوانند تابع عضو [۴۹] داشته باشند.
پیاده سازی ماژول به طور کلی به این شکل است :

```
 module NAME
        ....
end
```

و البته نام ماژول ها، مانند توابع و متغیر ها میتواند هرگونه که میخواهید، باشد. کاربرد ماژول ها، در این است که میتوانید با صدا زدن آنها درون کلاسها، تعدادی متد اضافه کنید. برای پیاده سازی یک ماژول ساده میتوانید چنین چیزی را درون محیط تعاملی بنویسید :

```
module A
   def a()
     puts "A is called"
   end
end
```

اکنون میتوانیم به پیاده سازی کلاس ها بپردازیم.

[۴۶] Method
[۴۷] Instance Variable
[۴۸] Structure
[۴۹] Member Function

۹.۸ پیاده سازی کلاس ها در روبی

کلاس ها در روبی به سادگی تعریف می شوند. شما میتوانید یک کلاس خالی ایجاد کنید، یا کلاسی ایجاد کنید که چندین متد داشته باشد، و یا متد هایش وارداتی از ماژول ها باشند. حتی امکان وارد کردن کلاس های دیگر درون یک کلاس نیز وجود دارد. شکل کلی کلاس ها به این شکل است :

```
class NAME
   ...
end
```

نام کلاس ها حتما باید همچون ثابت ها، با حروف بزرگ انگلیسی آغاز شود. بیایید برای نمونه، کلاس خوش آمد گو را ایجاد کنیم. پس محیط تعاملی را باز کرده و این چنین بنویسید :

```
class Greeter
   def say_hi(name)
       puts "Hello, #{name}"
   end
end
```

این کلاس، یک کلاس ساده است. متغیر عمومی ندارد و تنها یک متد را داراست. اکنون این کلاس را درون محیط تعاملی ایجاد می نماییم:

```
irb(main):006:0> class Greeter
irb(main):007:1>   def say_hi(name)
irb(main):008:2>       puts "Hello, #{name}"
irb(main):009:2>   end
irb(main):010:1> end
=> nil
irb(main):011:0>
```

کلاس، پایه و بنیان ایجاد یک شی است. اکنون یک شی جدید ایجاد میکنیم :

```
g = Greeter.new
```

و در محیط تعاملی وارد میکنیم :

```
irb(main):011:0> g = Greeter.new
=> #<Greeter:0x0000000219ab48>
irb(main):012:0>
```

همانگونه که دیدید، شی g ایجاد شده و اکنون میتوانیم متد تعریف شده را فراخوانی کنیم :

```
g.say_hi("Muhammadreza")
```

اکنون نتیجه اجرای همین دستور را درون محیط تعاملی مشاهده میکنیم :

```
irb(main):012:0> g.say_hi("Muhammadreza")
Hello, Muhammadreza
=> nil
irb(main):013:0>
```

همانگونه که ملاحظه کردید، متغیر ما موقع فراخوانی متد به آن داده شده است. اما ما میخواهیم همان لحظه که شی ایجاد می شود، نامش هم دریافت شود. پس نیاز داریم تا از تابع initialize استفاده کنیم. پس بیایید کلاس خوش آمد گو را دوباره بنویسیم. این بار با متغیر عمومی :

```
class Greeter
      def initialize(name)
      @name = name
      end
      def say_hi
          puts "Hello, #{@name}"
      end
end
```

پس از ایجاد آن در محیط تعاملی، شی g را اینگونه تعریف میکنیم :

```
g = Greeter.new("Muhammadreza")
```

و به این شکل در محیط تعاملی، نتیجه می گیریم :

```
irb(main):021:0> g = Greeter.new("Muhammadreza")
=> #<Greeter:0x0000000203fcd0 @name="Muhammadreza">
irb(main):022:0>
```

اکنون با فراخوانی متد نوشته شده در محیط تعاملی چنین نتیجه ای را شاهد هستیم :

```
irb(main):022:0> g.say_hi
Hello, Muhammadreza
=> nil
irb(main):023:0>
```

و دیگر لازم نیست مقداری را به عنوان آرگومان تابع، وارد کنیم.

۱.۹.۸ استفاده از ماژول ها درون کلاس

استفاده از ماژولها درون کلاس بسیار ساده است. کافیست ماژول A که نوشتیم را به این شکل درون کلاس جدید صدا کنیم :

```
class B
   include A
end
```

و سپس تابع a را از ماژول A که اکنون درون کلاس B است صدا میزنیم و چنین نتیجه ای می گیریم :

```
irb(main):040:0> b = B.new
=> #<B:0x000000022a65a0>
irb(main):041:0> b.a()
a is called
=> nil
irb(main):042:0>
```

۲.۹.۸ مفاهیم شی گرایی به صورت عملی

تا اینجا، با ساخت کلاس ها آشنا شدیم. پس بیایید کار عملی را آغاز کرده، مفاهیم تئوری را پیاده کنیم. مفاهیمی که در این قسمت گفته می شوند، مفاهیمی هستند که تئوری آن ها ابتدای فصل گفته شد.

۳.۹.۸ انتزاع

انتزاع [۵۰] را با همان مثال حیوان، میتوان ساخت. پس یک کلاس خالی ایجاد میکنیم که کلاس حیوان است.

```
class Animal
end
```

اکنون ما کلاس کلی حیوانات را داریم. آیا به نظر شما، گربه ها شکلی از حیوانات نیستند؟

۴.۹.۸ چندریختی و ارث بری

اکنون، کلاس گربه را ایجاد میکنیم که شکلی از حیوان به شمار می آید :

```
class Cat < Animal
end
```

[۵۰] Abstraction

در واقع، وقتی میگوییم «گربه» شکلی از «حیوان» است، ویژگی های «حیوانی» را نیز از کلاس «حیوان» به ارث برده است. علامت < برای انتقال ویژگی های یک کلاس به کلاس دیگر به کار میرود. [۵۱] پس تا اینجا، با مفهوم چندریختی و وراثت نیز آشنا شدیم. برویم سراغ کپسوله سازی.

۵.۹.۸ کپسوله سازی

همانگونه که گفته شد، کپسوله سازی برای مخفی سازی فرآیند از دید کاربر استفاده می شود. برای کپسوله کردن یک متد، باید آن را **private** کنیم. و برای دسترسی به آن، آن را درون یک متد دیگر فراخوانی میکنیم :

```
class Example
def a()
end
private :a
def b()
a()
end
end
```

و با اجرای متد **b** از کد بالا، کاربر میتواند ببیند که متد **a** چه میکند، اما در حقیقت نمی تواند در روند آن تغییری ایجاد کند.

۱۰.۸ کاربرد ها

کاربرد شی گرایی برای آن است، که روند برنامه نویسی و ساخت نرم افزارها، ساده تر شود. در واقع در زبانی مانند C++ این تکنیک پیاده شد تا این امر را محقق سازد و امروزه تقریبا همه کسانی که پایتون، روبی، جاوا، سی شارپ و ... را کار میکنند، میدانند که در این زبانها، موفقیت در گرو آن است که شی گرایی را خوب بدانند و پیاده کنند. برای مثال در فرمورک ریلز [۵۲] همه چیز در کلاسها اتفاق میفتد و شما نیاز دارید تا این بخش را به خوبی بدانید.

۱۱.۸ سخن آخر

تا این جای کتاب، هر چه از روبی در ذهن نگارنده نقش بسته بود، گفته شد. در واقع اگر افرادی که تمایل به یادگیری روبی دارند، وجود نداشتند، چنین ایده ای، یعنی ایجاد کتابی که در آن به آموزش روبی پرداخته شده باشد، در ذهن نگارنده نقش نمی بست. در این فصل، شی گرایی را یاد گرفتید. در واقع، سعی در این بود تا یادگیری شی گرایی، بیشتر با برانگیختن کنجکاوی کاربر توام باشد، بنابراین مثال های کمتری در این قسمت زده شد و بخش زیادی از مثال ها نیز اجرا نشدند، تا کاربر خودش آنها را ایجاد کند. امیدوارم با خواندن این کتاب، و یاد دادن روبی به دوستان خود، سهمی در گسترش دانش آزاد داشته باشید.

---

[۵۱] با علامت کوچکتر که از عملگرهای منطقی است اشتباه نشود.

[۵۲] Ruby on Rails

۱۲.۸ تمرینات

۱. برنامه ای بنویسید که در آن یک کلاس کاربر وجود داشته باشد، و سپس کاربر توسط صفحه کلید اطلاعاتی چون نام و نام کاربری را وارد کرده، سپس توسط تابعی در خروجی چاپ شود.

۲. یک کلاس کلی فیلم ایجاد کرده، سپس کلاس های ژانر های مختلفی مانند کمدی، ترسناک و اکشن ایجاد کنید و فیلم های مورد علاقه خود را توسط آنها دسته بندی کنید.