به نام دانش و نه هیچ چیز دیگر

« تجزیه و تحلیل داده‌ها با پایتون »

نویسنده: فابیو نلی

مترجم: منصور تقی‌زاده

تحلیل داده‌ها و علم داده با زبان برنامه‌نویسی پایتون

تقدیم به کریستوفر هیچنز

کریستوفر هیچنز:

زیبایی علم و دانش آن است که به شدت از
منزلت افسون خرافات می‌کاهد

فهرست

مقدمه

درباره نویسنده

فابیو نلی یک متخصص علم IT در پارک علمی IRBM، یک مرکز تحقیقاتی خصوصی در پوتزیا، رم (ایتالیا) است. او سال‌ها مشاور کامپیوتر در IBM، EDS، مرک شارپ و دوهمه، همراه با چندین بانک و شرکت بیمه بوده است. او مدرک شیمی آلی دارد و نیز دارای سال‌ها تجربه در سیستم‌های فن‌آوری اطلاعات و اتوماسیون کاربردی در علوم زندگی (متخصص فن‌آوری در بکمن کولتر ایتالیا و اسپانیا) است. او در حال حاضر در حال توسعه برنامه‌های جاوا است که پایگاه‌های اطلاعاتی اوراکل را با ابزار علمی، تولید داده‌ها و برنامه‌های کاربردی وب سرور و تحلیل نتایج به صورت بلادرنگ به محققان متصل می‌کند. علاوه بر این، او هماهنگ‌کننده جامعه مکنیسو است.

درباره نشریه فنی

شوخام سینگ تامار یک مهندس داده در پریدیکت است. او در بنگلور هند زندگی و کار می‌کند. در آخر هفته‌ها او داوطلب کار بر روی پروژه‌های علوم داده برای سازمان‌های غیر دولتی و سازمان‌های اجتماعی با بخش بانگلور می‌شود. او در وبلاگ خود در مورد پایتون، علم داده و یادگیری ماشین می‌نویسد:

shubhamtomar.me

سپاس‌گزاری (نویسنده)

من می‌خواهم از دوستانم، به خصوص آلبرتو، دانیل، روبرتو و الکس به خاطر کنار آمدن با من و ارائه پشتیبانی اخلاقی مورد نیاز در طول یک سال مشکلات و پریشانی تشکر کنم. با تشکر از مادرم.

سپاس‌گزاری (مترجم)

با سپاس بی‌نهایت از بهترین دوستم که گرچه اکنون در قید حیات نیست اما یاد او تا ابد با من خواهد بود.

فصل یک: مقدمه‌ای بر تحلیل داده

با این بخش شما شروع به برداشتن اولین قدم در دنیای تجزیه و تحلیل داده‌ها می‌کنید و تمام مفاهیم و فرایندهایی که این رشته را تشکیل می‌دهند را به طور مفصل می‌بینید. مفاهیم مورد بحث در این فصل، زمینه مفیدی برای فصل‌های بعدی خواهند بود، که در آن این مفاهیم و روش‌ها به شکل کد پایتون، از طریق استفاده از چندین کتابخانه که در بسیاری از بخش‌ها مورد بحث قرار خواهند گرفت، به کار گرفته خواهند شد.

تحلیل داده

در دنیایی که به طور فزاینده‌ای حول فن‌آوری اطلاعات متمرکز شده‌است، حجم عظیمی از داده‌ها هر روز تولید و ذخیره می‌شوند. اغلب این داده‌ها از سیستم‌های تشخیص خودکار، حسگرها و ابزارهای علمی می‌آیند، و یا هر بار که با بانک سر و کار دارید یا خرید می‌کنید، زمانی که در وبلاگ‌های مختلف ضبط می‌کنید و یا حتی زمانی که در شبکه‌های اجتماعی چیزی پست می‌کنید، آن‌ها را به صورت روزانه و ناخود آگاه تولید می‌کنید. اما داده‌ها چه هستند؟ داده‌ها در واقع اطلاعات نیستند، حداقل از نظر شکل. در جریان بدون فرمِ بایت‌ها، در نگاه اول درک ماهیت، تعداد یا زمان گزارش آن‌ها مشکل است. اطلاعات در واقع نتیجه پردازش است که با در نظر گرفتن مجموعه خاصی از داده‌ها، نتایجی را استخراج می‌کند که می‌تواند به روش‌های مختلف مورد استفاده قرار گیرد. این فرآیند استخراج اطلاعات از داده‌های خام تحلیل دقیق داده‌ها است. هدف از تجزیه و تحلیل داده‌ها دقیقا استخراج اطلاعاتی است که به راحتی قابل استنتاج نیستند اما وقتی درک شوند منجر به امکان انجام مطالعات بر روی مکانیزم‌های سیستم‌هایی می‌شود که آن‌ها را تولید کرده‌اند، بنابراین امکان پیش‌بینی واکنش‌های احتمالی این سیستم‌ها و تکامل آن‌ها در زمان را فراهم می‌کند. با شروع از یک رویکرد روشمند ساده در حفاظت از داده‌ها، تحلیل داده‌ها به یک رشته واقعی تبدیل شده‌است که منجر به توسعه روش‌های واقعی تولید مدل‌ها می‌شود. این مدل در واقع «تبدیل به شکل ریاضی یک سیستم مورد مطالعه» است. هنگامی که یک شکل ریاضی یا منطقی وجود دارد که قادر است پاسخ‌های سیستم را تحت سطوح مختلف دقت توصیف کند، می‌توانید پیش‌بینی‌هایی در مورد توسعه آن یا پاسخ به ورودی‌های خاص انجام دهید.

بنابراین هدف تحلیل داده‌ها مدل نیست، بلکه خوبیِ قدرتِ پیش‌بینی آن است. قدرت پیش‌بینی یک مدل نه تنها به کیفیت تکنیک‌های مدل‌سازی بستگی دارد بلکه به توانایی انتخاب مجموعه داده مناسب برای ساخت کل تحلیل داده‌ها نیز بستگی دارد. بنابراین جستجو برای داده‌ها، استخراج آن‌ها و آماده‌سازی متعاقب آن‌ها، در حالی که فعالیت‌های اولیه یک تحلیل را نشان می‌دهند، به دلیل اهمیت آن‌ها در موفقیت نتایج، به خود تحلیل داده‌ها نیز تعلق دارند. تاکنون در مورد داده‌ها، نحوه کار و پردازش آن‌ها از طریق روش‌های محاسبه صحبت کرده‌ایم. به موازات تمام مراحل پردازش تجزیه و تحلیل داده‌ها، روش‌های مختلفی از تجسم داده‌ها ایجاد شده‌است. در واقع، برای درک داده‌ها، هم به صورت انفرادی و هم از لحاظ نقشی که در کل مجموعه داده‌ها ایفا می‌کنند، هیچ سیستمی بهتر از توسعه تکنیک‌های نمایش گرافیکی قادر به تغییر اطلاعات، گاهی اوقات به طور ضمنی، در اَشکال وجود ندارد که به شما کمک می‌کند به راحتی معنای آن‌ها را درک کنید. در طول سال‌ها بسیاری از حالت‌های نمایش برای حالت‌های مختلف نمایشِ داده توسعه‌یافته اند: نمودارها. در پایان تحلیل داده‌ها، شما یک مدل و مجموعه‌ای از نمایش‌های گرافیکی خواهید داشت و سپس قادر به پیش‌بینی پاسخ‌های سیستم تحت مطالعه خواهید بود؛ پس از آن به مرحله (فاز) آزمایش خواهید رفت. این مدل با استفاده از مجموعه دیگری از داده‌ها که ما آن‌ها را پاسخ سیستم می‌دانیم، تست خواهد شد. با این حال، این داده‌ها برای تعریف مدل پیش‌بینی مورد استفاده قرار نمی‌گیرند. بسته به توانایی مدل برای تکرار پاسخ‌های مشاهده‌شده واقعی، شما یک محاسبه خطا و دانش از اعتبار مدل و محدودیت‌های عملیاتی آن خواهید داشت.

این نتایج را می توان با هر مدل دیگری مقایسه کرد تا بفهمیم که آیا مدل ایجاد شده جدید کارآمدتر از مدل‌های موجود است. زمانی که این را ارزیابی کردید، می‌توانید به مرحله آخر تحلیل داده‌ها - استقرار - برسید. این شامل اجرای نتایج تولید شده توسط تجزیه و تحلیل داده‌ها است، یعنی اجرای تصمیماتی که باید براساس پیش‌بینی‌های ایجاد شده توسط مدل اتخاذ شوند و خطراتی که چنین تصمیمی نیز پیش‌بینی خواهد شد. تحلیل داده‌ها رشته‌ای است که برای بسیاری از فعالیت‌های حرفه‌ای مناسب است. بنابراین، آگاهی از این که چه چیزی است و چگونه می توان آن را به عمل درآورد با تحکیم تصمیمات اتخاذ شده مرتبط خواهد بود. این به ما اجازه می‌دهد تا فرضیه‌ها را آزمایش کنیم و عمیقاً سیستم‌های تحلیل‌شده را درک کنیم.

قلمروهای دانش تحلیل داده

تحلیل داده‌ها اساسا یک رشته مناسب برای مطالعه مسایل است که ممکن است در زمینه‌های مختلفی از کاربردها رخ دهد. علاوه بر این، در فرآیندهای تحلیل داده‌ها ابزارها و روش‌های زیادی دارید که به دانش خوب محاسبات و مفاهیم ریاضی و آماری نیاز دارند. بنابراین یک تحلیل گر داده خوب باید قادر به حرکت و عمل در بسیاری از زمینه‌های انضباطی مختلف باشد. بسیاری از این رشته‌ها اساس روش‌های تحلیل داده‌ها هستند، و مهارت در آن‌ها تقریبا ضروری است. دانش رشته‌های دیگر بسته به حوزه کاربرد و مطالعه پروژه تجزیه و تحلیل داده‌های خاصی که می‌خواهید انجام دهید ضروری است، و به طور کلی، تجربه کافی در این حوزه‌ها می‌تواند به شما کمک کند تا بهتر مسائل و نوع داده‌های مورد نیاز برای شروع با تجزیه و تحلیل را درک کنید.

اغلب، با توجه به مشکلات عمده تجزیه و تحلیل داده‌ها، داشتن یک تیم میان رشته‌ای از متخصصان متشکل از اعضایی که همگی قادر به مشارکت به بهترین روش ممکن در حوزه‌های صلاحیت مربوطه خود هستند، ضروری است. با توجه به مسایل کوچک‌تر، یک تحلیل گر خوب باید قادر به شناسایی مشکلاتی باشد که در طول تحلیل داده‌ها به وجود می‌آیند، تحقیق کند که برای حل این مشکل به چه رشته‌ها و مهارت‌هایی نیاز است، این رشته‌ها را مطالعه کند و شاید حتی از افراد آگاه این بخش بخواهد. به طور خلاصه، تحلیلگر باید بتواند بداند که چگونه نه تنها به دنبال داده‌ها بگردد، بلکه اطلاعات مربوط به نحوه برخورد با آن‌ها را نیز بداند.

دانش کامپیوتر

دانش علوم کامپیوتر یک نیاز اساسی برای هر تحلیلگر داده است. تنها کسی که دانش و تجربه خوبی در علوم کامپیوتر دارد می‌تواند به طور موثر ابزارهای لازم برای تحلیل داده‌ها را مدیریت کند. در واقع تمام مراحل مربوط به تجزیه و تحلیل داده‌ها شامل استفاده از تکنولوژی کامپیوتر به عنوان نرم‌افزار محاسبه (مانند IDL، Matlab و غیره) و زبان‌های برنامه‌نویسی (مانند سی‌پلاس‌پلاس، جاوا، پایتون) است. حجم زیاد داده‌های در دسترس امروز به لطف فن‌آوری اطلاعات نیازمند مهارت‌های خاصی است تا بتوان تا حد امکان به طور موثر مدیریت شود. در واقع، تحقیق و استخراج داده نیازمند شکل‌های مختلف دانش است. داده‌ها در فایل‌ها یا جداول پایگاه داده با فرمت های خاص ذخیره می‌شوند. پرونده‌های XML، Json، یا به سادگی XLS یا CSV در حال حاضر قالب‌های رایج برای ذخیره‌سازی و جمع‌آوری داده‌ها هستند و بسیاری از برنامه‌های کاربردی نیز خواندن و مدیریت داده‌های ذخیره‌شده بر روی آن‌ها را ممکن می‌سازند.

برای استخراج داده‌های موجود در یک پایگاه‌داده، همه چیز خیلی فوری نیست، بلکه شما باید زبان پرس و جوی SQL را بشناسید و یا از نرم‌افزاری که به طور خاص برای استخراج داده‌ها از یک پایگاه‌داده توسعه‌یافته، استفاده کنید. علاوه بر این، برای برخی از انواع خاص پژوهش داده‌ها، داده‌ها در یک فرمت از پیش تعیین‌شده و صریح موجود نیستند، بلکه در فایل‌های متنی (اسناد، فایل‌های لاگ) یا در صفحات وب، به صورت نمودارها، سنجش‌ها، تعداد بازدیدکنندگان، یا جداول HTML نشان‌داده شده‌اند که نیاز به تخصص فنی خاص برای تجزیه و استخراج نهایی این داده‌ها است.

بنابراین، دانش فن‌آوری اطلاعات برای دانستن چگونگی استفاده از ابزارهای مختلفی که توسط علوم کامپیوتر معاصر در دسترس قرار می‌گیرند، مانند کاربردها و زبان‌های برنامه‌نویسی، ضروری است. این ابزارها به نوبه خود برای انجام تجزیه و تحلیل داده‌ها و تجسم داده‌ها مورد نیاز هستند. هدف این کتاب به طور دقیق فراهم کردن تمام دانش لازم، تا آنجا که ممکن است، با توجه به توسعه روش‌های تجزیه و تحلیل داده‌ها با استفاده از پایتون به عنوان یک زبان برنامه‌نویسی و کتابخانه‌های تخصصی است که سهم تعیین‌کننده‌ای در عملکرد تمام مراحل تشکیل‌دهنده تجزیه و تحلیل داده‌ها، از پژوهش داده‌ها گرفته تا داده کاوی، تا انتشار نتایج مدل پیش‌بینی دارد.

ریاضیات و آمار

همانطور که در سراسر کتاب خواهید دید، تجزیه و تحلیل داده‌ها به ریاضی زیادی نیاز دارد، که می‌تواند در طول درمان (treatment) و پردازش داده‌ها بسیار پیچیده باشد. بنابراین حداقل برای درک آنچه انجام می‌دهید، صلاحیت در همه این‌ها لازم است. آشنایی با مفاهیم آماری اصلی نیز ضروری است زیرا تمام روش‌هایی که در تحلیل و تفسیر داده‌ها به کار می‌روند براساس این مفاهیم هستند. همانطور که می‌توانید بگویید علم کامپیوتر به شما ابزار تجزیه و تحلیل داده‌ها را می‌دهد، بنابراین شما می‌توانید بگویید که آمار مفاهیمی را فراهم می‌کند که پایه و اساس تجزیه و تحلیل داده‌ها را تشکیل می‌دهند. بسیاری از آن‌ها ابزارها و روش‌هایی هستند که این رشته برای تحلیلگر فراهم می‌کند، و دانش خوب در مورد چگونگی استفاده بهینه از آن‌ها نیازمند سال‌ها تجربه است. برخی از رایج‌ترین تکنیک‌های آماری مورد استفاده در تجزیه و تحلیل داده‌ها عبارتند از:

- روش‌های بایسیان (Bayesian)
- بسرفت (regression)

◄ دسته بندی (clustering)

با رسیدگی به این موارد، شما کشف خواهید کرد که چگونه ریاضیات و آمار ارتباط نزدیکی با یکدیگر دارند، اما به لطف کتابخانه‌های پایتون ویژه‌ای که در این کتاب پوشش داده شده‌اند، شما توانایی مدیریت آن‌ها را خواهید داشت.

یادگیری ماشین (Machine Learning) و هوش مصنوعی

یکی از پیشرفته‌ترین ابزارهایی که در تحلیل داده‌ها قرار می‌گیرد یادگیری ماشین است. در واقع، با وجود تجسم داده‌ها و تکنیک‌هایی مانند خوشه‌بندی و رگرسیون، که تا حد زیادی به ما کمک می‌کند تا اطلاعاتی در مورد مجموعه داده‌های خود پیدا کنیم، در طول این مرحله از تحقیق، شما اغلب ترجیح می‌دهید از روش‌های خاصی استفاده کنید که در الگوهای جستجو در مجموعه داده‌ها بسیار تخصصی هستند. یادگیری ماشین رشته‌ای است که از مجموعه کاملی از رویه‌ها و الگوریتم ها استفاده می‌کند که داده‌ها را به منظور شناسایی الگوها، خوشه‌ها، یا روندها تجزیه و تحلیل می‌کنند و سپس اطلاعات مفیدی را برای تجزیه و تحلیل داده‌ها به روش کاملاً خودکار استخراج می‌کنند. این رشته به طور فزاینده‌ای در حال تبدیل شدن به ابزار اساسی تجزیه و تحلیل داده‌ها است و بنابراین دانش آن، حداقل به طور کلی، از اهمیت اساسی برای تحلیلگر داده برخوردار است.

زمینه‌های حرفه‌ای کاربرد

نکته بسیار مهم دیگر نیز قلمروی صلاحیتِ جایی است که داده‌ها می‌آیند (زیست‌شناسی، فیزیک، امور مالی، آزمایش مواد، آمار جمعیت و غیره). در واقع، اگر چه تحلیلگر آمادگی ویژه‌ای در زمینه آمار دارد، او همچنین باید قادر به کاوش در زمینه کاربرد و یا مستند کردن منبع داده‌ها، با هدف درک و فهم بهتر مکانیزم‌های تولید داده باشد. در واقع، داده‌ها یک رشته یا عدد ساده نیستند، بلکه یک عبارت هستند از اندازه‌گیری هر یک از پارامترهای مشاهده شده. بنابراین، درک بهتر زمینه کاربردی که داده‌ها از آن می‌آیند می‌تواند تفسیر آن‌ها را بهبود بخشد. با این حال، اغلب اوقات، این کار برای یک تحلیلگر داده، حتی تحلیل گر با بهترین نیات، بسیار پرهزینه است و بنابراین خوب است که مشاور یا چهره‌های کلیدی پیدا کنید که بتوانید سوالات درست را برای آن‌ها مطرح کنید.

فهمیدن طبیعت داده

هدف مطالعه تحلیل داده‌ها، اساساً داده‌ها است. داده‌ها بازیگران کلیدی در تمام فرآیندهای تحلیل داده‌ها خواهند بود. آن‌ها مواد خام را تشکیل می‌دهند که باید پردازش شوند و به لطف پردازش و تحلیل آن‌ها، استخراج انواع اطلاعات به منظور افزایش سطح دانش سیستم تحت مطالعه، امکان پذیر است، یعنی، اطلاعاتی که از آن داده شده‌است.

وقتی داده، اطلاعات می‌شود

داده‌ها وقایع ثبت‌شده در جهان هستند. هر چیزی که بتوان اندازه‌گیری یا حتی طبقه‌بندی کرد می‌تواند به داده تبدیل شود. پس از جمع‌آوری، این داده‌ها می‌توانند هم برای درک ماهیت رویدادها و هم برای پیش‌بینی و یا حداقل برای گرفتن تصمیمات آگاهانه، مورد مطالعه و تحلیل قرار گیرند.

وقتی اطلاعات، دانش می‌شود

شما می‌توانید از دانش زمانی صحبت کنید که اطلاعات به مجموعه‌ای از قوانین تبدیل می‌شوند که به شما کمک می‌کنند تا مکانیسم‌های خاصی را بهتر درک کنید و در نتیجه در مورد تکامل برخی رویدادها پیش‌بینی بکنید.

انواع داده

داده‌ها می‌توانند به دو دسته مجزا تقسیم شوند:

- ◄ قاطع
  - ✓ اسمی
  - ✓ ترتیبی
- ◄ عددی
  - ✓ گسسته
  - ✓ پیوسته

داده‌های قطعی ارزش‌ها یا مشاهداتی هستند که می‌توانند به گروه‌ها یا دسته‌ها تقسیم شوند. دو نوع ارزش مطلق وجود دارد: اسمی و ترتیبی. یک متغیر اسمی هیچ نظم ذاتی ندارد که در دسته خود شناسایی شود. یک متغیر ترتیبی در عوض یک ترتیب از پیش تعیین‌شده دارد.

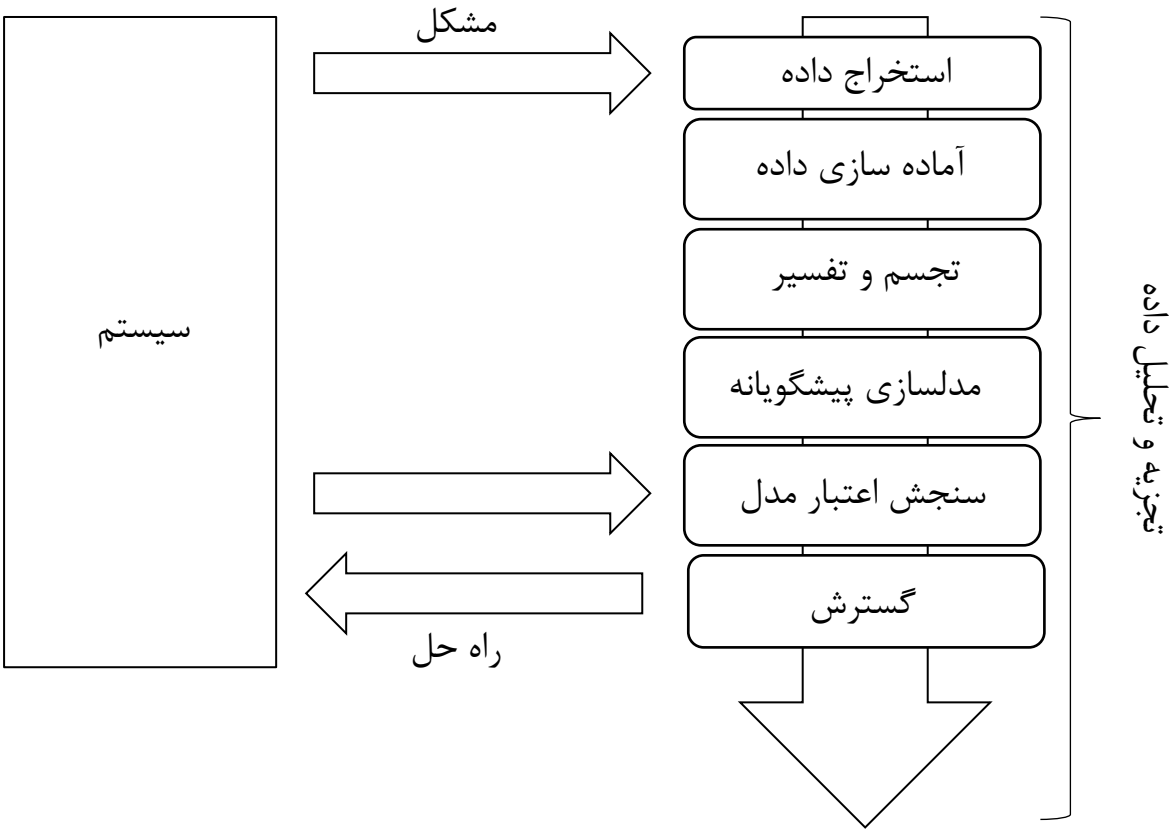
داده‌های عددی مقادیر یا مشاهداتی هستند که از اندازه‌گیری می‌آیند. دو نوع مقدار عددی متفاوت وجود دارد: اعداد گسسته و پیوسته. مقادیر گسسته مقادیری هستند که می‌توانند شمرده شوند و از یکدیگر مجزا و مجزا هستند. از طرف دیگر مقادیر پیوسته مقادیری هستند که توسط اندازه‌گیری‌ها یا مشاهداتی که هر مقدار را در یک محدوده تعریف‌شده فرض می‌کنند، تولید می‌شوند.

فرایند تجزیه و تحلیل داده

تجزیه و تحلیل داده‌ها می‌تواند به عنوان یک فرآیند متشکل از چندین مرحله توصیف شود که در آن داده‌های خام به منظور تجسم داده‌ها تبدیل و پردازش می‌شوند و می‌توانند به لطف یک مدل ریاضی براساس داده‌های جمع‌آوری‌شده پیش‌بینی کنند. سپس، تحلیل داده‌ها چیزی بیش از یک توالی مراحل نیست، که هر کدام نقشی کلیدی در مراحل بعدی ایفا می‌کنند. بنابراین، تجزیه و تحلیل داده‌ها تقریبا به صورت یک زنجیره فرآیند متشکل از مراحل زیر طرح‌ریزی می‌شود:

- تعریف مساله
- استخراج داده‌ها
- پاک‌سازی
- انتقال داده
- اکتشاف داده
- مدلسازی پیشگویانه
- سنجش اعتبار (تست کردن) مدل
- تجسم و تفسیر نتایج
- گسترش

شکل زیر یک نمایش شماتیک از تمام فرآیندهای درگیر در تجزیه و تحلیل داده‌ها است.



تعریف مشکل

فرآیند تجزیه و تحلیل داده‌ها در واقع خیلی قبل از جمع‌آوری داده‌های خام شروع می‌شود. در حقیقت، تجزیه و تحلیل داده‌ها همیشه با یک مشکل که باید حل شود، آغاز می‌گردد. مشکل تنها زمانی تعریف می‌شود که شما به خوبی بر روی سیستمی که می‌خواهید مطالعه کنید تمرکز داشته باشید: این ممکن است یک مکانیسم، یک برنامه، یا به طور کلی یک فرآیند باشد. به طور کلی این مطالعه می‌تواند به منظور درک بهتر عملکرد آن باشد، اما به طور خاص این مطالعه برای درک اصول رفتار آن به منظور توانایی پیش‌بینی یا انتخاب کردن، طراحی خواهد شد (که به عنوان یک انتخاب آگاهانه تعریف می‌شود). مرحله تعریف و سندسازی (خروجی‌های) مشکل علمی یا کسب و کار هر دو به منظور تمرکز دقیق بر روی رسیدن به نتایج مهم هستند. در واقع، یک مطالعه جامع و کامل از سیستم گاهی اوقات پیچیده است و شما همیشه اطلاعات کافی برای شروع ندارید. بنابراین تعریف مشکل و به‌ویژه برنامه‌ریزی آن می‌تواند دستورالعمل‌های پیروی از کل پروژه را تعیین کند.

وقتی مشکل تعریف و سندسازی شد، می‌توانید به سراغ برنامه‌ریزی یک پروژه تحلیل داده بروید. برنامه‌ریزی برای درک اینکه کدام متخصصان و منابع برای برآورده کردن الزامات انجام پروژه تا حد امکان موثر هستند، لازم است. بنابراین باید مسائل مربوط به حل این مشکل را در نظر بگیرید. شما به دنبال متخصصان حوزه‌های مختلف مورد علاقه خواهید بود و در نهایت نرم‌افزار مورد نیاز برای انجام تجزیه و تحلیل داده‌ها را نصب خواهید کرد. بنابراین، در طول فاز برنامه‌ریزی، انتخاب یک تیم کارآمد انجام می‌شود. به طور کلی، این تیم‌ها باید بین رشته‌ای باشند تا مشکل را با نگاه کردن به داده‌ها از دیدگاه‌های مختلف حل کنند. بنابراین، انتخاب یک تیم خوب قطعاً یکی از عوامل کلیدی موفقیت در تحلیل داده‌ها است.

استخراج داده

هنگامی که مساله تعریف شد، گام اول به دست آوردن داده‌ها به منظور انجام تحلیل است. داده‌ها باید با هدف اصلی ساخت مدل پیش‌بینی انتخاب شوند و بنابراین انتخاب آن‌ها برای موفقیت تحلیل نیز حیاتی است. داده‌های نمونه جمع‌آوری‌شده باید تا حد ممکن دنیای واقعی را منعکس کنند، یعنی، چگونه سیستم به محرک‌های دنیای واقعی پاسخ می‌دهد. در واقع، حتی با استفاده از مجموعه داده‌های عظیم داده‌های خام، اغلب، اگر آن‌ها به طور مناسب جمع‌آوری نشوند، ممکن است شرایط نادرست یا نامتعادل را در مقایسه با داده‌های واقعی نشان دهند.

بنابراین، انتخاب ضعیف داده‌ها، یا حتی انجام تحلیل بر روی مجموعه داده‌هایی که کاملا نماینده سیستم نیستند، منجر به مدل‌هایی خواهد شد که از سیستم تحت مطالعه دور خواهند شد. جستجو و بازیابی داده‌ها اغلب به شکلی از شهود نیاز دارد که فراتر از تحقیق فنی و استخراج دادهٔ صرف است. همچنین به درک دقیقی از ماهیت داده‌ها و شکل آن‌ها نیاز دارد، که تنها تجربه و دانش خوبی در زمینه کاربرد مشکل ارائه می‌دهد. صرف‌نظر از کمیت و کیفیت داده‌های مورد نیاز، مساله دیگر جستجو و انتخاب صحیح **منابع داده‌ای** است.

اگر محیط استودیو یک آزمایشگاه (فنی یا علمی) باشد و داده‌های تولید شده تجربی باشند، در این صورت منبع داده به راحتی قابل‌شناسایی است. در این حالت، مشکلات تنها مربوط به راه‌اندازی آزمایشی خواهند بود. اما امکان تجزیه و تحلیل داده‌ها برای تولید مجدد سیستم‌هایی که در آن‌ها داده‌ها به شیوه‌ای کاملا تجربی در هر زمینه از کاربرد جمع‌آوری می‌شوند، وجود ندارد. بسیاری از زمینه‌های کاربردی نیازمند جستجو برای داده‌های دنیای اطراف هستند، اغلب به داده‌های تجربی خارجی متکی هستند، یا حتی بیشتر آن‌ها را از طریق مصاحبه یا نظرسنجی جمع‌آوری می‌کنند.

بنابراین در این موارد، جستجو برای یک منبع داده خوب که قادر به ارائه تمام اطلاعات مورد نیاز برای تحلیل داده‌ها باشد می‌تواند بسیار چالش برانگیز باشد. اغلب بازیابی داده‌ها از منابع اطلاعاتی چندگانه برای تکمیل هر گونه نقص، شناسایی هر گونه اختلاف و عمومی کردن مجموعه داده‌های ما تا حد امکان ضروری است.

زمانی که می‌خواهید داده‌ها را به دست آورید، یک مکان خوب برای شروع فقط وب است. اما بیشتر داده‌های روی وب را می توان به سختی به‌دست آورد؛ در حقیقت، همه داده‌ها در یک فایل یا پایگاه داده موجود نیستند، بلکه می‌توانند کم و بیش محتوای ضمنی درون صفحات HTML در قالب‌های مختلف باشند. برای این منظور، یک روش به نام **وب اسکریپتینگ**، که امکان جمع‌آوری داده‌ها را از طریق شناسایی وقوع خاص تگ‌های HTML در صفحات وب فراهم می‌کند، توسعه داده شده‌است. نرم‌افزاری وجود دارد که به طور خاص برای این منظور طراحی شده‌است و زمانی که یک رویداد یافت می‌شود داده‌های مورد نظر را استخراج می‌کند. زمانی که جستجو کامل شد، لیستی از داده‌ها را آماده خواهید کرد تا در معرض آنالیز قرار گیرند.

آماده سازی داده ها

در میان تمام مراحل درگیر در تحلیل داده‌ها، آماده‌سازی داده‌ها، با وجود مشکلات کم‌تر، در واقع چیزی است که نیاز به منابع بیشتر و زمان بیشتری برای تکمیل دارد. داده‌های جمع‌آوری‌شده اغلب از منابع داده‌ای مختلف جمع‌آوری می‌شوند که هر کدام از آن‌ها داده‌ها را با یک شکل و نمایش متفاوت در خود دارند. بنابراین، تمام این داده‌ها باید برای فرآیند تحلیل داده‌ها آماده شوند. آماده‌سازی داده‌ها در ارتباط با به دست آوردن، تمیز کردن، نرمال سازی و تبدیل داده‌ها به یک مجموعه داده بهینه است، به عبارت دیگر در یک فرمت آماده، به طور معمول جدولی، مناسب برای روش‌های تحلیلی که در طول مرحله طراحی برنامه‌ریزی شده‌اند. بسیاری از مشکلات هستند که باید از آن‌ها اجتناب کرد، مانند مقادیر نامعتبر، مبهم، یا از دست رفته، رشته‌های تکراری، یا داده‌های خارج از محدوده.

تجسم و تفسیر داده

بررسی داده‌ها لزوما جستجو برای داده‌ها در یک ارایه گرافیکی یا آماری به منظور یافتن الگوها، اتصالات و روابط در داده‌ها است. تجسم داده‌ها بهترین ابزار برای برجسته کردن الگوهای ممکن است. در سال‌های اخیر، تجسم داده‌ها به حدی توسعه‌یافته است که خود، به یک رشته واقعی تبدیل شده‌است.

در واقع، تکنولوژی‌های متعددی منحصراً برای نمایش داده‌ها استفاده می‌شوند و به همان اندازه انواع نمایش کاربردی برای استخراج بهترین اطلاعات ممکن از یک مجموعه داده هستند. اکتشاف داده شامل بررسی اولیه داده‌ها است که برای درک نوع اطلاعاتی که جمع‌آوری شده‌اند و فهم منظور آن‌ها مهم است. در ترکیب با اطلاعات به‌دست‌آمده در طول تعریف مساله ، این طبقه‌بندی تعیین خواهد کرد که کدام روش تجزیه و تحلیل داده‌ها برای رسیدن به تعریف مدل مناسب‌تر خواهد بود. به طور کلی، این مرحله علاوه بر مطالعه دقیق نمودارها از طریق داده‌های بصری، ممکن است شامل یک یا چند فعالیت زیر باشد:

- خلاصه کردن داده‌ها
- گروه‌بندی داده‌ها
- بررسی رابطه بین ویژگی‌های مختلف
- شناسایی الگوها و گرایش‌ها
- ساخت مدل‌های رگرسیون
- ساخت مدل‌های طبقه‌بندی

به طور کلی، تجزیه و تحلیل داده‌ها نیازمند فرآیندهای خلاصه‌سازی عبارات در مورد داده‌هایی است که باید مطالعه شوند. خلاصه کردن فرآیندی است که طی آن داده‌ها بدون از دست دادن اطلاعات مهم مهیای تفسیر می‌شوند. خوشه‌بندی یک روش تجزیه و تحلیل داده‌ها است که برای یافتن گروه‌های متحد شده با ویژگی‌های مشترک (گروه‌بندی) به کار می‌رود. مرحله مهم دیگر تحلیل بر شناسایی روابط، روندها و بی‌قاعدگی‌ها در داده‌ها تمرکز می‌کند. به منظور یافتن این نوع اطلاعات، فرد اغلب باید به ابزارها متوسل شود و نیز یک دور دیگر از تجزیه و تحلیل داده‌ها را انجام دهد، این بار در خود تجسم داده‌ها. روش‌های دیگر داده کاوی مانند درخت‌های تصمیم‌گیری و قوانین ارتباط به طور خودکار حقایق یا قوانین مهم را از داده‌ها استخراج می‌کنند. این روش‌ها می‌توانند به طور موازی با تجسم داده‌ها برای یافتن اطلاعات در مورد روابط بین داده‌ها استفاده شوند.

مدلسازی پیشگویانه

مدل‌سازی پیشگویانه فرآیندی است که در تحلیل داده‌ها برای ایجاد یا انتخاب یک مدل آماری مناسب برای پیش‌بینی احتمال یک نتیجه استفاده می‌شود. بعد از بررسی داده‌ها شما تمام اطلاعات مورد نیاز

برای توسعه مدل ریاضی را دارید که رابطه بین داده‌ها را کدگذاری می‌کند. این مدل‌ها برای درک سیستم مورد مطالعه، مفید هستند و به روشی خاص برای دو هدف اصلی مورد استفاده قرار می‌گیرند. اولی پیش‌بینی در مورد مقادیر داده تولید شده توسط سیستم است؛ در این صورت با مدل‌های رگرسیونی سر و کار خواهیم داشت. دومی دسته‌بندی محصولات داده‌ای جدید است و در این مورد شما از مدل‌های طبقه‌بندی یا مدل‌های خوشه‌بندی استفاده خواهید کرد. در واقع، امکان تقسیم مدل‌ها با توجه به نوع نتیجه‌ای که تولید می‌کنند وجود دارد:

- مدل‌های طبقه‌بندی: اگر نتیجه به‌دست‌آمده از نوع مدل قطعی باشد.
- مدل‌های رگرسیون: اگر نتیجه به‌دست‌آمده از نوع مدل عددی باشد.
- مدل‌های خوشه‌بندی: اگر نتیجه به‌دست‌آمده از نوع مدل توصیفی باشد.

روش‌های ساده برای تولید این مدل‌ها شامل تکنیک‌هایی مانند رگرسیون خطی، رگرسیون لجستیک، درخت‌های طبقه‌بندی و رگرسیون و نزدیک‌ترین همسایگان k است. اما روش‌های تحلیل متعدد هستند، و هر کدام ویژگی‌های خاصی دارند که آن را برای برخی از انواع داده‌ها و تحلیل‌ها عالی می‌سازند. هر یک از این روش‌ها یک مدل خاص را تولید می‌کنند و سپس انتخاب آن‌ها برای ماهیت مدل محصول مناسب است. برخی از این مدل‌ها مقادیری متناظر با سیستم واقعی را فراهم می‌کنند و همچنین با توجه به ساختار خود برخی از ویژگی‌های سیستم تحت مطالعه را به روشی ساده و روشن توضیح می‌دهند. مدل‌های دیگر نیز پیش‌بینی‌های خوبی ارائه خواهند داد، اما ساختار آن‌ها چیزی بیش از یک «جعبه سیاه» با توانایی محدود برای توضیح برخی ویژگی‌های سیستم نخواهد بود.

سنجش اعتبار مدل

اعتبارسنجی مدل، یعنی مرحله آزمون، مرحله مهمی است که به شما اجازه می‌دهد تا مدل ساخته‌شده براساس داده‌های اولیه را تایید کنید. این مهم است زیرا به شما اجازه می‌دهد تا اعتبار داده‌های تولید شده توسط مدل را از طریق مقایسه مستقیم آن‌ها با سیستم واقعی ارزیابی کنید. اما این بار، شما از مجموعه داده‌های اولیه که کل تحلیل بر روی آن‌ها انجام شده‌است، بیرون می‌آیید.

به طور کلی، زمانی که از آن‌ها برای ساخت مدل استفاده می‌کنید و زمانی که از آن‌ها برای تایید مدل استفاده می‌کنید، به داده‌ها به عنوان مجموعهٔ اعتبار سنجی نگاه خواهید کرد.

بنابراین، با مقایسه داده‌های تولید شده توسط مدل با داده‌های تولید شده توسط سیستم شما قادر به ارزیابی خطا خواهید بود و با استفاده از مجموعه داده‌های تست مختلف، می‌توانید حدود اعتبار مدل تولید شده را تخمین بزنید. در واقع مقادیر به درستی پیش‌بینی‌شده می‌تواند تنها در محدوده خاصی معتبر باشد، یا سطوح متفاوتی از تطبیق بسته به محدوده مقادیر در نظر گرفته‌شده داشته باشد. این فرآیند نه تنها به شما این امکان را می‌دهد که به صورت عددی اثربخشی مدل را ارزیابی کنید بلکه آن را با هر مدل موجود دیگری مقایسه کنید. تکنیک‌های مختلفی در این زمینه وجود دارد؛ از مشهورترین آن‌ها، اعتبارسنجی متقابل است. این تکنیک براساس تقسیم مجموعه اعتبار سنجی به بخش‌های مختلف است. هر یک از این بخش‌ها به نوبه خود به عنوان مجموعه «اعتبار» و هر بخش دیگری به عنوان مجموعه «آموزش» استفاده خواهند شد. در این روش تکراری، شما یک مدل به طور فزاینده‌ای کامل خواهید داشت.

گسترش

این گام نهایی فرآیند تحلیل است، که هدف آن ارایه نتایج، یعنی نتایج تحلیل است. در فرآیند آماده‌سازی، در محیط کسب و کار، تحلیل به نفع مشتری است که آن را سفارش داده‌است. در محیط‌های فنی یا علمی، نتیجه به صورت راه‌حل‌های طراحی یا انتشارات علمی ترجمه می‌شود. به عبارت دیگر، «گسترش» اساساً شامل عملی کردن نتایج به‌دست‌آمده از تحلیل داده‌ها می‌باشد. چندین روش برای گسترش نتایج تحلیل داده یا داده کاوی وجود دارد. به طور معمول، گسترش تحلیلگر داده شامل نوشتن یک گزارش برای مدیریت یا برای مشتری است که درخواست تجزیه و تحلیل کرده‌است. این سند به طور مفهومی، نتایج به‌دست‌آمده از تجزیه و تحلیل داده‌ها را توصیف خواهد کرد. گزارش باید به سمت مدیرانی هدایت شود که سپس قادر به تصمیم‌گیری هستند. سپس، آن‌ها واقعاً نتایج تحلیل را عملی خواهند کرد. در مستندات ارائه‌شده توسط تحلیلگر، هر یک از این چهار موضوع به طور کلی به تفصیل مورد بحث قرار خواهند گرفت:

- نتایج تجزیه و تحلیل
- گسترش تصمیم‌گیری
- تحلیل ریسک
- اندازه‌گیری تاثیر کسب و کار

وقتی نتایج پروژه شامل ایجاد مدل‌های پیش‌بینی باشد، این مدل‌ها می‌توانند به عنوان یک برنامه مستقل به‌کار گرفته شوند یا می‌توانند در نرم‌افزار دیگری ادغام شوند.

تجزیه و تحلیل داده‌های کمی و کیفی

تجزیه و تحلیل داده‌ها فرآیندی است که کاملاً بر روی داده‌ها تمرکز دارد و بسته به ماهیت داده‌ها، امکان ایجاد برخی تمایزات وجود دارد. هنگامی که داده‌های تحلیل‌شده دارای یک ساختار قطعی یا عددی هستند، شما در مورد تحلیل کمی صحبت می‌کنید، اما زمانی که با ارزش‌هایی سر و کار دارید که از طریق توصیفات در زبان طبیعی بیان می‌شوند، در مورد آنالیز کیفی صحبت می‌کنید. دقیقا به خاطر ماهیت متفاوت داده‌های پردازش‌شده توسط دو نوع تحلیل، شما می‌توانید برخی تفاوت‌های بین آن‌ها را مشاهده کنید.

تجزیه و تحلیل کمی باید با داده‌هایی انجام شود که یک نظم منطقی در درون خود دارند، یا می‌تواند به نوعی طبقه‌بندی شود. این امر منجر به شکل‌گیری ساختارها در داده‌ها می‌شود. ترتیب، طبقه‌بندی و ساختارها به نوبه خود اطلاعات بیشتری فراهم می‌کنند و امکان پردازش بیشتر داده‌ها را به روش ریاضی سخت‌تر فراهم می‌نمایند. این امر منجر به تولید مدل‌ها می‌شود که می‌تواند پیش‌بینی‌های کمی ارایه دهد، در نتیجه به تحلیلگر داده این امکان را می‌دهد تا نتایج عینی تری به‌دست آورد.

در عوض تحلیل کیفی باید با داده‌هایی سروکار داشته باشد که عموما ساختار ندارند، حداقل یکی از آن‌ها مشهود نیست و ماهیت آن‌ها نه عددی است و نه قطعی. برای مثال، داده‌های مطالعه کیفی می‌توانند شامل داده‌های متنی، تصویری یا صوتی نوشته شده باشند. بنابراین این نوع تجزیه و تحلیل باید بر پایه متدلوژی باشد که اغلب فاقد عمومیت است تا اطلاعاتی را استخراج کند که به طور کلی منجر به مدل‌هایی می‌شود که قادر به ارایه پیش‌بینی‌های کیفی هستند، با این نتیجه که نتایجی که تحلیلگر داده می‌تواند به آن‌ها برسد ممکن است شامل تفاسیر ذهنی نیز باشد. از سوی دیگر، تحلیل کیفی می‌تواند سیستم‌های پیچیده‌تر را کشف کند و نتیجه‌گیری کند که با یک رویکرد صرفا ریاضی ممکن نیست. اغلب این نوع تحلیل شامل مطالعه سیستم‌هایی مانند پدیده‌های اجتماعی یا ساختارهای پیچیده است که به آسانی قابل‌اندازه‌گیری نیستند.

جدولی که در ادامه می‌بینید تفاوت بین دو نوع تحلیل را نشان می‌دهد:

تحلیل کیفی	تحلیل کمی
داده متنی، تصویری و صوتی	داده‌های عددی و قطعی
پیشگویی‌های کیفی	پیشگویی‌های کمی
تفاسیر ذهنی‌تر	تفاسیر عینی‌تر

داده‌های آزاد (باز)

در حمایت از افزایش تقاضا برای داده‌ها، تعداد زیادی از منابع داده در اینترنت در دسترس هستند. این منابع اطلاعاتی، اطلاعات را آزادانه برای هر کسی که نیاز دارد فراهم می‌کنند و آن‌ها را داده‌های باز می‌نامند.

- DataHub (http://datahub.io/dataset)
- World Health Organization (http://www.who.int/research/en/)
- Data.gov (http://data.gov)
- European Union Open Data Portal (http://open-data.europa.eu/en/data/)
- Amazon Web Service public datasets (http://aws.amazon.com/datasets)
- Facebook Graph (http://developers.facebook.com/docs/graph-api)
- Healthdata.gov (http://www.healthdata.gov)
- Google Trends (http://www.google.com/trends/explore)
- Google Finance (https://www.google.com/finance)
- Google Books Ngrams (http://storage.googleapis.com/books/ngrams/books/datasetsv2.html)
- Machine Learning Repository (http://archive.ics.uci.edu/ml/)

در این راستا، برای دریافت یک ایده از منابع داده باز در دسترس به صورت آنلاین، می‌توانید به نمودار ابری LOD (http://lod-cloud.net) نگاه کنید، که تمام اتصالات لینک داده بین چندین منبع داده باز را نشان می‌دهد که در حال حاضر در شبکه موجود هستند.

پایتون و تحلیل داده

استدلال اصلی این کتاب توسعه تمام مفاهیم تجزیه و تحلیل داده‌ها از طریق برخورد با آن‌ها بر حسب پایتون است. پایتون یک زبان برنامه‌نویسی است که به دلیل تعداد زیاد کتابخانه‌هایی که مجموعه‌ای کامل از ابزارها را برای تحلیل و دستکاری داده‌ها فراهم می‌کنند، به طور گسترده در محافل علمی مورد استفاده قرار می‌گیرد.

در مقایسه با دیگر زبان‌های برنامه‌نویسی که به طور کلی برای تجزیه و تحلیل داده‌ها استفاده می‌شوند، مانند R و Matlab، پایتون نه تنها یک پلتفرم برای پردازش داده‌ها فراهم می‌کند بلکه ویژگی‌هایی دارد که آن را در مقایسه با زبان‌های دیگر و برنامه‌های تخصصی منحصر به فرد می‌سازد. توسعه روز افزون تعداد کتابخانه‌های پشتیبانی، اجرای الگوریتم های متدلوژی‌های خلاقانه‌تر و توانایی ارتباط با دیگر زبان‌های برنامه‌نویسی (C و فرترن) پایتون را در میان نوع خود منحصر به فرد ساخته‌است. علاوه بر این، پایتون نه تنها برای تجزیه و تحلیل داده‌ها تخصص دارد، بلکه برنامه‌های کاربردی بسیار دیگری نیز دارد، مانند برنامه‌نویسی عمومی، نوشتار، ارتباط با پایگاه‌های داده و اخیراً توسعه وب، به لطف چارچوب‌های وب مانند جانگو. بنابراین توسعه پروژه‌های تحلیل داده‌ای که کاملاً با وب سرور سازگار هستند و امکان ادغام آن‌ها در وب وجود دارد، امکان پذیر است. بنابراین، برای کسانی که می‌خواهند تجزیه و تحلیل داده‌ها را انجام دهند، پایتون، با تمام پکیج‌های خود، می‌تواند بهترین انتخاب برای آینده در نظر گرفته شود.

نتیجه

در این بخش مشاهده کردید که تحلیل داده‌ها و به طور خاص فرایندهای مختلفی که شامل آن هستند، چیست. همچنین، شما متوجه نقش داده‌ها در ساخت یک مدل پیش‌بینی و چگونگی انتخاب دقیق آن‌ها براساس یک تحلیل دقیقِ داده‌ها شدید.

تا فصل بعد، این دیدگاه نسبت به پایتون و ابزارهایی که برای انجام تجزیه و تحلیل داده‌ها فراهم می‌کند را در نظر داشته باشید.

فصل دو: مقدمه‌ای بر دنیای پایتون

زبان پایتون و دنیای اطراف آن، توسط مفسران، ابزارها، ویراستاران، کتابخانه‌ها و غیره ساخته شده‌است. این دنیای پایتون در سال‌های اخیر گسترش زیادی پیدا کرده و توسعه دهندگان آن را غنی و غنی‌تر کرده‌اند. پایتون برای کسی که برای اولین بار بدان نزدیک می‌شود گاهی اوقات می‌تواند پیچیده و تا حدی گمراه‌کننده باشد. بنابراین اگر برای اولین بار در حال نزدیک شدن به برنامه‌نویسی در پایتون هستید، ممکن است در میان این همه انتخاب احساس گم شدن بکنید، به خصوص دربارهٔ جایی که باید شروع کنید.

در این بخش نگاهی کلی به جهان پایتون می‌اندازیم. ابتدا شما توصیفی از زبان پایتون و ویژگی‌های آن خواهید داشت که آن را منحصر به فرد ساخته‌است. خواهید دید که از کجا شروع کنید، یک مفسر چیست، و چگونه شروع به نوشتن اولین خطوط کد در پایتون کنید. سپس شکل‌های پیشرفته تری از نوشتار تعاملی را با توجه به پوسته‌هایی مانند IPython و دفترچه یادداشت IPython بررسی می‌کنید.

پایتون - زبان برنامه‌نویسی

پایتون یک زبان برنامه‌نویسی است که توسط گویدو ون روسوم در سال ۱۹۹۱ ایجاد شده‌است. این زبان می‌تواند با یک سری صفات مشخص شود:

- تفسیر شده
- پرتابل (قابل حمل)
- شی‌گرا
- دارای تاثیر بر یکدیگر
- متن - باز
- درک و استفاده آسان

پایتون یک زبان برنامه‌نویسی تفسیر شده‌است که شبه مدون است. زمانی که کد برنامه را نوشته‌اید، برای اجرای آن نیاز به مترجم دارید.

مفسر برنامه‌ای است که بر روی هر ماشینی نصب می‌شود که وظیفه تفسیر کد مبدا و اجرای آن را دارد. بنابراین بر خلاف زبان C، C++ و جاوا زمان کامپایل وجود ندارد. پایتون یک زبان برنامه‌نویسی بسیار قابل‌حمل است. تصمیم به استفاده از مترجم شفاهی به عنوان واسط خواندن و اجرای کد دارای مزیت کلیدی است: قابلیت حمل. در واقع، شما می‌توانید پایتون را بر روی هر پلتفرم موجود نصب کنید (لینوکس، ویندوز، مک). پایتون همچنین برای این جنبه به عنوان زبان برنامه‌نویسی برای بسیاری از دستگاه‌های کوچک مانند Raspberry Pi و میکروکنترلرها انتخاب شد.

پایتون یک زبان برنامه‌نویسی شی گرا است. در واقع به شما این امکان را می‌دهد که کلاس‌هایی از اشیا را مشخص کنید و وراثت آن‌ها را اجرا کنید. اما بر خلاف C++ و جاوا هیچ سازنده یا تخریب‌کننده‌ای وجود ندارد. پایتون همچنین به شما اجازه می‌دهد تا برای مدیریت استثناها، سازه‌های خاصی را در کد خود پیاده‌سازی کنید. با این حال ساختار زبان به قدری انعطاف‌پذیر است که امکان برنامه‌ریزی با رویکردهای جایگزین را نسبت به ساختار هدف گرا فراهم می‌کند.

پایتون یک زبان برنامه‌نویسی تعاملی است. به لطف این واقعیت که پایتون از یک مترجم برای اجرا استفاده می‌کند، این زبان می‌تواند جنبه‌های بسیار متفاوتی را بسته به زمینه‌ای که در آن استفاده می‌شود، در نظر بگیرد. در واقع، شما می‌توانید کدی را که از تعداد زیادی خط تشکیل شده‌است بنویسید، شبیه به آنچه که ما در زبان‌های C++ یا جاوا انجام می‌دهیم و سپس برنامه را راه‌اندازی کنید، یا می‌توانید بلافاصله یک خط فرمان وارد کنید و آن را اجرا کنید، فوراً نتایج دستور را بگیرید و بسته به آن‌ها می‌توانید تصمیم بگیرید که خط بعدی فرمان چه خواهد بود. این حالت بسیار تعاملی برای اجرای کد، پایتون را به یک محیط محاسباتی کاملاً شبیه به Matlab تبدیل می‌کند. این ویژگی پایتون است که موفقیت این زبان برنامه‌نویسی را در جامعه علمی به ارمغان آورد.

پایتون یک زبان برنامه‌نویسی است که می‌تواند interfaced باشد. در واقع، این زبان برنامه‌نویسی می‌تواند با کد نوشته شده در زبان‌های برنامه‌نویسی دیگر مانند C، C++ و فرترن تعامل داشته باشد. در واقع، به لطف این جنبه، پایتون توانست آنچه را که شاید تنها نقطه ضعف است (سرعت اجرا) جبران کند. ماهیت پایتون، به عنوان یک زبان برنامه‌نویسی بسیار پویا، گاهی اوقات می‌تواند منجر به اجرای برنامه‌ها تا ۱۰۰ برابر کندتر از برنامه‌های استاتیک متناظر گردآوری‌شده با زبان‌های دیگر شود؛ بنابراین راه‌حل این نوع از مشکلاتِ عملکرد، رابط پایتون برای تدوین کدهای زبان‌های دیگر با استفاده از آن به گونه‌ای است که گویی زبان خودش است.

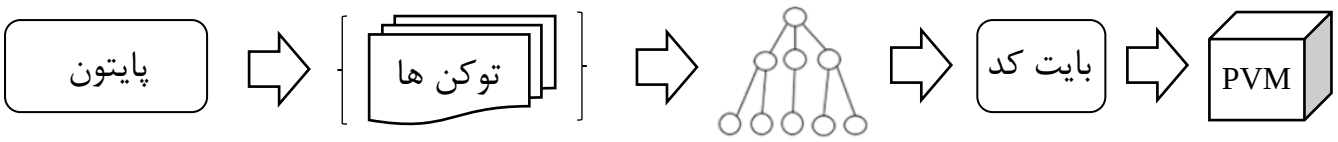
پایتون یک زبان برنامه‌نویسی متن باز است. سی‌پایتون، که مرجع پیاده‌سازی زبان پایتون است، کاملا آزاد و متن باز می‌باشد. علاوه بر این، هر ماژول یا کتابخانه در شبکه، متن باز است و کد آن‌ها به صورت آنلاین در دسترس است. هر ماه، یک جامعه توسعه دهنده گسترده پیشرفت‌هایی را برای غنی‌تر و کارآمدتر کردن این زبان و همه کتابخانه‌های آن به ارمغان می‌آورد. سی‌پایتون توسط بنیاد غیر انتفاعی نرم‌افزار پایتون اداره می‌شود که در سال ۲۰۰۱ ایجاد شد و وظیفه ترویج، حفاظت و توسعه زبان برنامه‌نویسی پایتون را بر عهده دارد.

در نهایت، پایتون یک زبان ساده برای استفاده و یادگیری است. این جنبه شاید از همه مهم‌تر باشد زیرا مستقیم‌ترین جنبه است که یک توسعه دهنده، حتی یک تازه‌کار، با آن مواجه است. درک شهودی بالا و سهولت خواندن کد پایتون اغلب منجر به «هم‌دردی» برای این زبان برنامه‌نویسی می‌شود و در نتیجه انتخاب بیشتر تازه واردان در برنامه‌نویسی است. با این حال، سادگی آن به معنای محدودیت نیست، زیرا پایتون زبانی است که در هر زمینه‌ای از محاسبات گسترش می‌یابد. علاوه بر این، پایتون همه این کارها را به سادگی در مقایسه با زبان‌های برنامه‌نویسی موجود مانند سی‌پلاس‌پلاس، جاوا و فرترن انجام می‌دهد که ماهیت آن‌ها بسیار پیچیده است.

پایتون – مفسر

همان طور که در بخش‌های قبلی توضیح داده شد، هر بار که فرمان پایتون را اجرا می‌کنید، مترجم پایتون شروع به کار می‌کند، که با یک پیام <<< مشخص می‌شود. مفسر پایتون به سادگی برنامه‌ای است که فرمان‌هایی را که به سرعت منتقل می‌شوند می‌خواند و تفسیر می‌کند. شما دیدید که مترجم می‌تواند یک دستور واحد را در یک زمان یا کل فایل‌های کد پایتون بپذیرد. با این حال روشی که با آن این کار را انجام می‌دهد همیشه یکسان است.

هر بار که کلید ورود را فشار می‌دهید، مترجم شروع به اسکن کد نوشته شده (یک ردیف یا فایل کامل کد) با نشان (توکن کردن - tokenization) می‌کند. این نشانه‌ها بخش‌هایی از متن هستند که مترجم آن‌ها را در ساختار درختی مرتب می‌کند. درخت به‌دست‌آمده ساختار منطقی برنامه است که سپس به بایت کد تبدیل می‌شود. زنجیره پردازش با بایت کد پایان می‌یابد که توسط ماشین مجازی پایتون (PVM) اجرا خواهد شد.



شما می‌توانید مستندات بسیار خوبی در مورد این موضوع در لینک زیر مشاهده نمایید:

https://www.ics.uci.edu/~pattis/ICS-31/lectures/tokens.pdf

مترجم استاندارد پایتون به عنوان سیتون گزارش شده‌است چون کاملا در C نوشته شده‌است. مناطق (areas) دیگری مانند جیتون (Jython) وجود دارند که با استفاده از زبان‌های برنامه‌نویسی دیگر مانند جاوا توسعه یافته‌اند؛ آیرون‌پایتون، در C# (و فقط برای ویندوز) و و پای‌پای (PyPy)، به طور کامل در پایتون توسعه یافته‌اند.

سیتون (Cython)

پروژه سیتون براساس ایجاد یک کامپایلر است که کد پایتون را به کد معادل C ترجمه می‌کند. سپس این کد در محیط سیتون در اجرا می‌شود. این نوع سیستم کامپایل امکان معرفی معانی C را در کد پایتون فراهم کرده‌است تا آن را حتی موثرتر کند. این سیستم منجر به ادغام دو دنیای زبان برنامه‌نویسی با تولد سیتون شده‌است که می‌تواند یک زبان برنامه‌نویسی جدید در نظر گرفته شود. شما می‌توانید مستندات زیادی در مورد آن به صورت آنلاین پیدا کنید؛ پیشنهاد می‌کنم به این لینک مراجعه نمایید:

http://docs.cython.org

جیتون (Jython)

به موازات سیتون، نسخه‌ای به نام جیتون در جاوا ساخته و تدوین شده‌است. جیتون توسط جیم هوگونین در سال ۱۹۹۷ ایجاد شد. جیتون نسخه‌ای از پیاده‌سازی زبان برنامه‌نویسی پایتون در جاوا است؛ همچنین با استفاده از کلاس‌های جاوا به جای پیمانه‌های پایتون برای پیاده‌سازی پسوندها و بسته‌های پایتون مشخص می‌شود.

پای‌پای

مفسر پای‌پای یک کامپایلر JIT (Just-in-time) است که کد پایتون را در زمان اجرا به طور مستقیم به کد ماشین تبدیل می‌کند. این انتخاب برای سرعت بخشیدن به اجرای پایتون صورت گرفت. با این حال،

این انتخاب منجر به استفاده از زیر مجموعه کوچکی از دستورها پایتون شده‌است، که به عنوان Rpyton تعریف شده‌است. برای کسب اطلاعات بیشتر در این زمینه می‌توانید به وب‌سایت رسمی مراجعه کنید:

http://pypy.org

پایتون ۲ و پایتون ۳

جامعه پایتون هنوز در حال انتقال از مفسران سری ۲ به سری ۳ است. در حقیقت، اکنون شما دو نسخه از پایتون را پیدا خواهید کرد که به صورت موازی استفاده می‌شوند (نسخه ۲.۷ و نسخه ۳.۴). این نوع ابهام می‌تواند سردرگمی زیادی ایجاد کند، به خصوص از نظر انتخاب کدام نسخه برای استفاده و تفاوت بین این دو نسخه. یک سوال که قطعا باید بپرسید این است که: «چرا نسخه ۲ هنوز با وجود انتشار نسخه ۳ ارائه می‌شود؟»

هنگامی که گویدو ون روسوم (خالق زبان پایتون) تصمیم گرفت تغییرات مهمی در زبان پایتون به وجود آورد، به‌زودی دریافت که این تغییرات پایتون جدید را با بسیاری از قوانین موجود ناسازگار خواهد کرد. بنابراین او تصمیم گرفت با نسخه جدیدی از پایتون به نام پایتون ۳.۰ شروع کند. مساله سازگاری و ایجاد مقادیر زیادی از کدهای غیرقابل‌استفاده در شبکه، باعث اتخاذ تصمیم حفظ یک نسخه سازگار یعنی ۲.۷ گردید. پایتون ۳.۰ اولین بار در سال ۲۰۰۸ ظاهر شد، در حالی که نسخه ۲.۷ در سال ۲۰۱۰ منتشر شد با این وعده که انتشار بزرگ آن دنبال نشود و در حال حاضر نسخه فعلی ۳.۴ (۲۰۱۴) است. در این کتاب ما به پایتون ۲ اشاره خواهیم کرد؛ با این حال، با چند استثنا، هیچ مشکلی با پایتون ۳ وجود ندارد.

نصب پایتون

برای توسعه برنامه‌ها در پایتون باید آن را بر روی سیستم‌عامل خود نصب کنید. برخلاف توزیع‌های ویندوز، لینوکس و Mac OS X باید نسخه از پیش نصب‌شده پایتون را در خود داشته باشند. اگر نمی‌خواهید، یا می‌خواهید آن را با نسخه دیگری جایگزین کنید، می‌توانید به راحتی آن را نصب کنید. نصب پایتون بسته به سیستم‌عامل متفاوت است؛ با این حال، این عمل نسبتا ساده است.

در سیستم‌های Debian-Ubuntu Linux

apt-get install python

در Red Hat، سیستم‌های لینوکس فدورا که با پکیج‌های rpm کار می‌کنند

yum install python

اگر سیستم‌عامل شما ویندوز یا Mac OS X است می‌توانید به سایت رسمی پایتون (http://www.python.org) بروید و نسخه‌ای را که ترجیح می‌دهید دانلود کنید. پکیج‌های موجود در این مورد به صورت خودکار نصب می‌شوند.

با این حال، امروزه توزیع‌هایی وجود دارند که همراه با مفسر پایتون، تعدادی ابزار را فراهم می‌کنند که مدیریت و نصب پایتون، تمام کتابخانه‌ها و برنامه‌های کاربردی مرتبط را آسان‌تر می‌سازند. من قویاً توصیه می‌کنم که یکی از توزیع‌های موجود آنلاین را انتخاب کنید.

توزیع‌های پایتون

با توجه به موفقیت زبان برنامه‌نویسی پیتون، در طول سال‌ها، ابزارهای موجود در پکیج، که برای برآورده کردن بسیاری از کارکردهای مختلف توسعه‌یافته اند، به چنان تعداد زیادی تبدیل شده‌اند که عملاً مدیریت همه آن‌ها به صورت دستی غیرممکن شده‌است. در این راستا، بسیاری از توزیع‌های پایتون که امکان مدیریت کارآمد صدها پکیج پایتون را فراهم می‌کنند، در حال حاضر در دسترس هستند. در واقع، به جای دانلود جداگانه مفسر، که تنها کتابخانه‌های استاندارد در درون خود دارد و سپس نیاز به نصب جداگانه همه کتابخانه‌های اضافی می‌باشد، نصب توزیع پایتون بسیار آسان‌تر است. قلب و مرکزیت این توزیع‌ها منیجرهای پکیج‌ها هستند که چیزی بیش از برنامه‌هایی نیستند که به طور خودکار بسته‌های پایتون را مدیریت، نصب، ارتقا، پیکربندی و حذف می‌کنند که بخشی از توزیع هستند. عملکرد آن‌ها بسیار مفید است، زیرا کاربر به سادگی یک پکیج خاص را درخواست می‌کند و مدیر پکیج، معمولاً از طریق اینترنت، عملیات را با تجزیه و تحلیل نسخه لازم، در کنار همه وابستگی‌ها با هر پکیج دیگر انجام می‌دهد و در صورت عدم وجود آن را دانلود می‌کند.

آناکوندا (Anaconda)

آناکوندا یک توزیع رایگان از بسته‌های پایتون است که با تجزیه و تحلیل پیوسته منتشر گشته است.

(https://store.continuum.io/cshop/anaconda/)

این توزیع از سیستم‌عامل لینوکس، ویندوز و مک OSX پشتیبانی می‌کند. آناکوندا، علاوه بر ارائه آخرین بسته‌های منتشر شده در جهان پایتون، اغلب ابزارهایی که برای ایجاد یک محیط توسعه برای زبان برنامه‌نویسی پایتون نیاز دارید را در بر می‌گیرد.

در واقع، وقتی توزیع آناکوندا را بر روی سیستم خود نصب می‌کنید، این فرصت را دارید که از ابزارها و برنامه‌های توضیح داده‌شده در این فصل استفاده کنید، بدون این که نگران این باشید که باید هر کدام از آن‌ها را به طور جداگانه نصب و مدیریت کنید. این توزیع پایه شامل Spyder به عنوان IDE، آی‌پایتون QtConnet و Notebook است. مدیریت کل توزیع آناکوندا با برنامه‌ای به نام کوندا انجام می‌شود. این مدیر بسته‌بندی و مدیر محیط توزیع آناکوندا است که همه بسته‌ها و نسخه‌های آن‌ها را مدیریت می‌کند.

conda install <package name>

یکی از جالب‌ترین جنبه‌های این توزیع، توانایی مدیریت محیط‌های توسعه چندگانه است، که هر کدام نسخه خود از پایتون را دارند. در واقع، وقتی آناکوندا را نصب می‌کنید، نسخه ۲.۷ پایتون به طور پیش‌فرض نصب می‌شود. همه بسته‌های نصب‌شده سپس به آن نسخه رجوع خواهند کرد. این یک مشکل نیست، زیرا آناکوندا امکان کار هم‌زمان و مستقل با دیگر نسخه‌های پایتون را با ایجاد یک محیط جدید فراهم می‌کند. برای مثال، شما می‌توانید یک محیط براساس پایتون ۳.۴ ایجاد کنید.

conda create -n py34 python=3.4 anaconda

این کار یک محیط جدید آناکوندا با تمام بسته‌های مربوط به نسخه پایتون ۳.۴ ایجاد خواهد کرد. این نصب به هیچ وجه محیط ساخته‌شده با پایتون ۲.۷ را تحت‌تاثیر قرار نخواهد داد. شما می‌توانید محیط جدید را با وارد کردن دستور زیر فعال نمایید.

source activate py34

روی ویندوز:

activate py34

C:\Users\Fabio>activate py34

Activating environment "py34"...

[py34] C:\Users\Fabio>

شما می‌توانید هر تعداد نسخه از پایتون را که می‌خواهید بسازید؛ شما فقط باید پارامتر عبور داده‌شده با گزینه پایتون در فرمان conda را تغییر دهید. وقتی می‌خواهید برای کار با نسخه اصلی پایتون برگردید باید از دستور زیر استفاده کنید:

source deactivate

روی ویندوز:

```
[py34] C:\Users\Fabio>deactivate
Deactivating environment "py34"
C:\Users\Fabio>
```

Enthought Canopy

توزیع دیگری نیز بسیار شبیه به آناکوندا وجود دارد و آن توزیع Canopy ارائه‌شده توسط Enthought است، شرکتی که در سال ۲۰۰۱ تاسیس شد و به ویژه برای پروژه SciPy بسیار معروف است. این توزیع از سیستم‌های لینوکس، ویندوز و مک او ایکس پشتیبانی می‌کند و شامل حجم زیادی از پکیج‌ها، ابزارها و برنامه‌های کاربردی است که توسط مدیر بسته مدیریت می‌شوند. مدیر پکیج Canopy بر خلاف کوندا کاملاً گرافیکی است. متاسفانه، تنها نسخه اصلی این توزیع، یعنی کانوپی اکسپرس، آزاد است؛ علاوه بر بسته‌ای که به طور معمول توزیع می‌شود، شامل آی‌پایتون و یک کپی از Canopy نیز می‌شود که ویژگی خاصی دارد که در IDE های دیگر وجود ندارد. این نرم‌افزار به منظور استفاده از این محیط به عنوان پنجره‌ای برای تست و اشکال‌زدایی کد، آی‌پایتون را تعبیه کرده‌است.

پایتون (x,y)

پایتون (x,y) یک توزیع رایگان است که فقط روی ویندوز کار می‌کند و از آدرس زیر قابل دانلود می‌باشد:

http://code.google.com/p/pythonxy/

این توزیع، Spyder را به عنوان IDE دارد.

استفاده از پایتون

پایتون زبانی غنی اما ساده و در عین حال بسیار انعطاف‌پذیر است. این روش امکان گسترش فعالیت‌های توسعه‌ای شما را در بسیاری از حوزه‌های کاری (تحلیل داده‌ها، علم‌گرایی، واسط‌های گرافیکی و غیره) فراهم می‌آورد. دقیقا به همین دلیل، امکان استفاده از پایتون می‌تواند زمینه‌های بسیار مختلفی را، اغلب با توجه به سلیقه و توانایی توسعه دهنده، در بر بگیرد. این بخش روش‌های مختلف استفاده از پایتون در دورهٔ کتاب را نشان می‌دهد؛ با توجه به موضوعات مختلفی که در فصل‌های گوناگون مورد بحث قرار می‌گیرند، این رویکردهای مختلف به طور خاص مورد استفاده قرار خواهند گرفت چرا که آن‌ها برای کار مناسب‌تر خواهند بود.

پوسته پایتون (Shell)

ساده‌ترین راه برای رسیدن به جهان پایتون باز کردن یک نشست (Session) روی پوسته (Shell) پایتون، یک خط فرمان اجرایی ترمینال است. در حقیقت، می‌توانید یک خط فرمان را در یک زمان وارد کرده و بلافاصله عملیات آن را تست کنید. این حالت ماهیت مفسر را روشن می‌سازد که اساس عملیات پایتون را تشکیل می‌دهد. در واقع مترجم قادر است فرمان را در یک زمان بخواند و وضعیت متغیرهای مشخص‌شده در خطوط قبلی را حفظ کند، رفتاری مشابه رفتار متلب و دیگر نرم‌افزارهای محاسبه. این روش برای آن روش‌ها برای اولین بار با زبان پایتون بسیار مناسب است. شما این توانایی را دارید که فرمان را هر بار بدون نیاز به نوشتن، ویرایش و اجرای یک برنامه کامل که گاهی اوقات از تعداد زیادی خط کد تشکیل شده‌است، تست کنید. این حالت برای تست و اشکال‌زدایی یک خط در یک زمان، یا به سادگی برای انجام محاسبات استفاده می‌شود. برای شروع یک نشست (Session) روی ترمینال، به سادگی در سطر فرمان بنویسید:

```
>>> python
Python 2.7.8 (default, Jul 2 2014, 15:12:11) [MSC v.1500 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

حالا پوسته (Shell) پایتون فعال است و مفسر آماده دریافت دستورها در پایتون است. با وارد کردن ساده‌ترین فرمان‌ها شروع کنید؛ به این کد کلاسیک توجه نمایید:

```
>>> print "Hello World!"
Hello World!
```

اجرای کد برنامه سراسری

آشناترین راه برای هر برنامه‌نویس نوشتن کل کد برنامه و سپس اجرای آن از پایانه است. ابتدا یک برنامه با استفاده از یک ویرایشگر متن ساده بنویسید؛ می‌توانید به عنوان مثال از کدی که در ادامه آمده استفاده کرده و آن را به نام MyFirstProgram.py ذخیره کنید.

به این نکته توجه داشته باشید که پسوند فایل‌های پایتون py می‌باشد.

```
myname = raw_input (What is your name? ")
print "Hi " + myname + ", I'm glad to say: Hello world!"
```

حالا شما اولین برنامه خود را در پایتون نوشته‌اید و می‌توانید آن را مستقیماً از خط فرمان با فراخوانی فرمان پایتون و سپس نام فایل حاوی کد برنامه اجرا کنید.

```
python myFirstProgram.py
What is your name? Fabio Nelli
Hi Fabio Nelli, I'm glad to say: Hello world!
```

اجرای کد با استفاده از یک IDE

یک رویکرد جامع‌تر از رویکردهای قبلی استفاده از یک محیط توسعه یکپارچه (یا IDE) است. این ویرایشگرها یک نرم‌افزار پیچیده واقعی هستند که محیط کاری را برای توسعه کد پایتون فراهم می‌کنند. آن‌ها در ابزارهایی غنی هستند که زندگی را برای توسعه دهندگان، به ویژه هنگامی که اشکال‌زدایی انجام می‌شود، آسان‌تر می‌کنند. در بخش‌های زیر به طور مفصل خواهید دید که چه IDE هایی در حال حاضر موجود هستند.

تعامل با پایتون

آخرین رویکرد، و به نظر من، شاید خلاقانه‌ترین رویکرد، رویکرد تعاملی است. در واقع، علاوه بر سه رویکرد قبلی، که آن‌هایی هستند که برای بهتر یا بدتر شدن توسط توسعه دهندگان زبان‌های برنامه‌نویسی دیگر استفاده می‌شوند، این رویکرد فرصت تعامل مستقیم با کد پایتون را فراهم می‌کند. در این راستا، جهان پایتون با معرفی «آی‌پایتون» به شدت غنی شده‌است. آی‌پایتون یک ابزار بسیار قدرتمند است که به طور خاص برای رفع نیازهای تعامل بین مفسر پایتون و توسعه دهنده طراحی شده‌است، که تحت این رویکرد نقش تحلیلگر، مهندس یا محقق را بر عهده می‌گیرد. در بخش بعدی، آی‌پایتون و ویژگی‌های آن با جزییات بیشتری توضیح داده خواهد شد.

نوشتن کد پایتون

در بخش قبلی دیدید که چگونه یک برنامه ساده بنویسید که در آن رشته "Hello World" چاپ شود. اکنون در این بخش شما یک مرور کوتاه از اصول اولیه زبان پایتون خواهید داشت تا فقط با مهم‌ترین جنبه‌های اساسی آشنا شوید.

این بخش برای آموزش شما با برنامه‌نویسی در پایتون، یا برای نشان دادن قواعد نحوی زبان برنامه‌نویسی نیست، بلکه برای ارائه یک مرور سریع از برخی اصول اساسی پایتون لازم برای ادامه با موضوعات این کتاب است. اگر از قبل زبان پایتون را می‌دانید، می‌توانید با اطمینان از این بخش مقدماتی صرف‌نظر کنید. در عوض اگر با برنامه‌نویسی آشنا نیستید و درک موضوعات برایتان دشوار است، من به شما توصیه می‌کنم که به اسناد آنلاین، آموزش‌ها و دوره‌های مختلف مراجعه کنید.

محاسبه

شما قبلا دیده‌اید که عملکرد ()print برای چاپ تقریبا هر چیزی به کار می‌رود. پایتون علاوه بر این که یک ابزار چاپ است، یک ماشین‌حساب بزرگ نیز هست. یک نشست (Session) در پوسته (Shell) پایتون شروع کنید و این عملیات ریاضی را انجام دهید:

```
>>> ۱ + ۲
۳
>>> (۱,۰٤٥ * ۳)/٤
۰,۷۸۳۷٥
>>> ٤ ** ۲
۱٦
>>> ((٤ + ٥j) * (۲ + ۳j))
(-۷+۲۲j)
>>> ٤ < (۲*۳)
True
```

پایتون قادر است بسیاری از انواع داده‌ها شامل اعداد و شرایط پیچیده با مقادیر بولی را محاسبه کند. همانطور که از محاسبات بالا می‌بینید، مفسر پایتون مستقیما نتیجه محاسبات را بدون نیاز به استفاده از تابع ()print برمی گرداند. همین امر در مورد ارزش‌های درون متغیرها نیز صدق می‌کند. کافی است که متغیر را صدا کنیم تا محتوای آن را ببینیم.

```
>>> a = ۱۲ * ۳,٤
>>> a
۴۰.۸
```

وارد کردن (import) کتابخانه‌ها و عملکردهای جدید

دیدید که قدرت پایتون با توانایی گسترش عملکرد خود به وسیله وارد کردن پکیج‌های متعدد و ماژول‌های موجود مشخص می‌شود. برای وارد کردن کامل یک ماژول باید از دستور import استفاده کنید.

```
>>> import math
```

به این ترتیب تمام توابع موجود در بسته ریاضی در نشست (Session) پایتون شما موجود هستند تا بتوانید آن‌ها را به طور مستقیم صدا بزنید. بنابراین وقتی یک نشست (Session) پایتون را آغاز می‌کنید، مجموعه استانداردی از توابع موجود را گسترش داده‌اید. این توابع با عبارت زیر فراخوانی می‌شوند:

```
library_name. function_name ()
```

برای مثال، اکنون می‌توانید سینوس مقدار موجود در متغیر a را محاسبه کنید.

```
>>> math.sin(a)
```

همانطور که می‌بینید این تابع همراه با نام کتابخانه فراخوانی می‌شود. گاهی ممکن است از این اصطلاح استفاده کنید:

```
>>> from math import *
```

حتی اگر این کار به درستی جواب دهد، باید به عنوان یک تمرین خوب از آن اجتناب کنید. در اصل نوشتن یک مفهوم به این روش شامل وارد کردن تمام وظایف بدون نیاز به تعریف کتابخانه‌ای است که به آن تعلق دارند.

```
>>> sin(a)
۰.۰۴۰۶۹۳۲۵۷۳۴۹۸۶۴۸۵۶
```

این نوع import ها در واقع می‌تواند منجر به خطاهای بسیار بزرگ شود، به خصوص اگر تعداد کتابخانه‌های وارد شده رو به افزایش باشد. در حقیقت، بعید نیست که کتابخانه‌های مختلف عملکردهایی با یک نام داشته باشند، و وارد کردن همه آن‌ها منجر به برتری همه کارکردها با همان نام قبلی شود. بنابراین رفتار برنامه می‌تواند خطاهای متعدد یا بدتر، رفتار غیر عادی ایجاد کند. این روش برای import به طور کلی تنها برای تعداد محدودی از توابع استفاده می‌شود، یعنی، توابعی که به شدت برای عملکرد برنامه ضروری هستند، در نتیجه از وارد کردن کل کتابخانه در زمانی که کاملا غیر ضروری است، اجتناب می‌شود.

```
>>> from math import sin
```

ساختمان داده

در مثال‌های قبلی دیدید که چگونه از متغیرهای ساده حاوی یک مقدار واحد استفاده کنید. پایتون تعدادی از ساختارهای اطلاعاتی بسیار مفید را فراهم می‌کند. این ساختارهای داده‌ای قادر هستند چندین داده را به طور همزمان و گاهی اوقات حتی از انواع مختلف در خود جای دهند. ساختارهای مختلف داده ارایه‌شده بسته به این که چگونه داده‌های آن‌ها به طور داخلی ساختار بندی شده‌اند، به صورت متفاوتی تعریف می‌شوند.

- لیست
- مجموعه
- رشته
- تاپل
- دیکشنری
- deque
- heap

این تنها بخش کوچکی از تمام ساختارهای داده‌ای است که می‌تواند با پایتون ساخته شود. در میان تمام این داده‌ها، دیکشنری‌ها و لیست‌ها بیش‌ترین کاربرد را دارند. نوع دیکشنری، که به عنوان dicts نیز تعریف می‌شود، یک ساختار داده است که در آن هر مقدار خاص با یک برچسب خاص به نام کلید (key) در ارتباط است. داده‌های جمع‌آوری‌شده در یک دیکشنری هیچ ترتیب داخلی ندارند بلکه فقط تعاریف جفت کلید / مقدار هستند.

```
>>> dict = {'name':'William', 'age':25, 'city':'London'}
```

اگر می‌خواهید به یک مقدار خاص در دیکشنری دسترسی داشته باشید باید نام کلید مربوطه را نشان دهید.

```
>>> dict["name"]
'William'
```

اگر می‌خواهید جفت مقادیر را در یک دیکشنری تکرار کنید، باید از ساختار "for-in" استفاده کنید. این امر از طریق استفاده از عملکرد ()items امکان پذیر است.

```
>>> for key, value in dict. items ():
... print (key, value)
...
name William
city London
age 25
```

لیست یک ساختار داده‌ای می‌باشد که شامل تعدادی شی به منظور تشکیل یک توالی است که عناصر را می‌توان به آن اضافه و حذف کرد. هر آیتم با یک عدد متناظر با ترتیب توالی به نام Index مشخص می‌شود.

```
>>> list = [1,2,3,4]
>>> list
[1, 2, 3, 4]
```

اگر می‌خواهید به هر یک از عناصر دسترسی داشته باشید، کافی است که شاخص را در براکت های مربعی مشخص کنید (اولین آیتم در لیست دارای ۰ به عنوان index است)، در حالی که اگر شما بخشی از لیست (یا یک توالی) را در نظر بگیرید، مشخص کردن دامنه با شاخص‌های i و j مطابق با کمترین و بیشترین قسمت کافی است.

```
>>> list [۲]
۳
>>> list [۱:۳]
[۲, ۳]
```

در عوض اگر از شاخص‌های (index) منفی استفاده می‌کنید، به این معنی است که شما آخرین مورد در لیست را در نظر می‌گیرید و به تدریج به سمت اولین حرکت می‌کنید.

```
>>> list [-۱]
۴
```

به منظور انجام اسکن عناصر یک لیست می‌توانید از ساختار for-in استفاده کنید.

```
>>> items = [۱,۲,۳,٤,٥]
>>> for item in items:
... item + ۱
...
۲
۳
۴
۵
۶
```

برنامه‌نویسی عملکردی (فقط برای پایتون ۳.۴)

حلقه for - in نشان‌داده‌شده در مثال قبلی بسیار شبیه به آن‌هایی است که در زبان‌های برنامه‌نویسی دیگر یافت می‌شوند. اما در واقع، اگر می‌خواهید یک توسعه دهنده «پایتون» باشید، باید از استفاده از حلقه‌های صریح اجتناب کنید. پایتون روش‌های جایگزین دیگری را ارایه می‌دهد، که تکنیک‌های برنامه‌نویسی مانند برنامه‌نویسی عملکردی را مشخص می‌کند.

ابزاری که پایتون برای توسعه برنامه‌نویسی عملکردی فراهم می‌کند شامل یک سری از توابع است:

- map(function,list)
- filter(function, list)
- reduce(function, list)
- lambda
- list comprehension

حلقه for که به تازگی دیدید یک هدف خاص دارد، که اعمال یک عملیات بر روی هر آیتم و سپس جمع‌آوری نتیجه است. این کار می‌تواند با تابع ()map انجام شود.

```
>>> items = [۱,۲,۳,٤,٥]
>>> def inc(x): return x+۱
...
>>> list (map (inc, items))
[۲, ۳, ٤, ٥, ٦]
```

در مثال قبلی، ابتدا تابعی را تعریف کرده‌اید که عملیات را بر روی هر عنصر انجام می‌دهد، و سپس آن را به عنوان اولین استدلال به ()map انتقال داده‌اید. پایتون به شما اجازه می‌دهد تا تابع را به طور مستقیم در آرگومان اول با استفاده از lambda به عنوان یک تابع تعریف کنید. این امر تا حد زیادی کد را کاهش می‌دهد و ساختار قبلی را در یک خط کد کامل می‌کند.

```
>>> list (map ((lambda x: x+۱), items))
[۲, ۳, ٤, ٥, ٦]
```

دو تابع دیگر که به روشی مشابه عمل می‌کنند، ()filter و ()reduce هستند. تابع ()filter عناصر فهرستی را استخراج می‌کند که برای آن تابع درست بر می‌گردد. در عوض تابع ()reduce همه عناصر لیست را برای ایجاد یک نتیجه واحد در نظر می‌گیرد. برای استفاده از ()reduce شما باید ابزارهای عملکرد ماژول را وارد کنید.

```
>>> list (filter ((lambda x: x < ٤), items))
[۱, ۲, ۳]
>>> from functools import reduce
>>> reduce ((lambda x, y: x/y), items)
۰.۰۰۸۳۳۳۳۳۳۳۳۳۳۳۳
```

هر دوی این توابع انواع دیگری از استفاده از حلقه را اجرا می‌کنند. آن‌ها قصد دارند این چرخه‌ها و عملکرد آن‌ها را جایگزین کنند، که می‌تواند به طور متناوب با عملکرد ساده فراخوانی بیان شود. این همان چیزی است که برنامه‌نویسی عملکردی را تشکیل می‌دهد. مفهوم نهایی برنامه‌نویسی تابعی، درک لیست است. از این مفهوم برای ساختن لیست‌ها به روشی بسیار ساده و طبیعی استفاده می‌شود که به آن‌ها به شیوه‌ای مشابه نحوه توصیف ریاضی‌دانان از مجموعه داده‌ها اشاره دارد. مقادیر دنباله از طریق یک تابع یا عملیات خاص تعریف می‌شوند.

```
>>> S = [x**۲ for x in range (۵)]
>>> S
[۰, ۱, ٤, ۹, ١٦]
```

فاصله گذاری (Indentation)

یک ویژگی برای کسانی است که از زبان‌های برنامه‌نویسی دیگر می‌آیند و نقش مهمی بازی می‌کند. در حالی که شما قبلا فاصله‌ها را به دلایل صرفا زیبایی‌شناسی مدیریت می‌کردید، که کد را تا حدی قابل خواندن‌تر می‌سازد، در پایتون نقش اساسی در اجرای کد فرض می‌شود، و آن را به بلوک‌های منطقی تقسیم می‌کند. در واقع، در حالی که در جاوا، سی و سی‌پلاس‌پلاس هر خط فرمان کد توسط یک ";" از خط بعدی جدا می‌شود؛ در پایتون شما نباید هیچ نمادی را که آن‌ها را از هم جدا می‌کند مشخص کنید. این نقش‌ها در پایتون از طریق فاصله‌گذاری اداره می‌شوند؛ یعنی بسته به نقطه شروع خط کد، مفسر آن خط را متعلق به بلوک منطقی می‌داند یا خیر.

```
>>> a = ٤
>>> if a > ۳:
... if a < ۵:
... print ("I'm four")
... else:
... print ("I'm a little number")
...
I'm four
>>> if a > ۳:
... if a < ۵:
... print ("I'm four")
... else:
... print ("I'm a big number")
...
I'm four
```

در این مثال می‌توانید ببینید که بسته به این که فرمان دیگر چگونه نامگذاری شده‌است، شرایط دو معنای متفاوت را در نظر می‌گیرند (که توسط برنامه نویس در خود رشته‌ها مشخص شده‌است).

آی‌پایتون

آی‌پایتون یک توسعه از پایتون است که شامل تعدادی ابزار است: پوسته آی‌پایتون، یک پوسته تعاملی قدرتمند است که منجر به یک پایانه (ترمینال) بسیار پیشرفته پایتون می‌شود؛ یک QtCon انحصاری که ترکیبی از یک پوسته و یک GUI است که به این طریق امکان نمایش گرافیک در داخل کنسول به جای پنجره‌های مجزا را می‌دهد و در نهایت IPython Notebook یک رابط وب است که به شما اجازه می‌دهد متن، کد اجرایی، گرافیک و فرمول را در یک نمایش واحد ترکیب کنید.

پوسته (Shell) آی‌پایتون

این پوسته (Shell) ظاهرا شبیه یک اجرای نشست (Session) پایتون از یک خط فرمان است، اما در واقع، ویژگی‌های بسیار دیگری را فراهم می‌کند که این پوسته (Shell) را قدرتمندتر و توانمندتر از نوع کلاسیک می‌سازد. برای راه‌اندازی این پوسته (Shell) فقط آی‌پایتون را در خط فرمان تایپ کنید.

```
> ipython
Python 2.7.8 (default, Jul 2 2014, 15:12:11) [M
Type "copyright", "credits", or "license" for more information.
IPython 2.4.1 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]:
```

همانطور که می‌بینید، یک prompt خاص با مقدار in [1] ظاهر می‌شود. این بدان معنی است که اولین خطِ ورودی است. در واقع، آی‌پیتون یک سیستم از prompt های عددی (شاخص گذاری شده) را با ذخیره‌سازی ورودی و خروجی ارایه می‌دهد.

```
In [1]: print "Hello World!"
Hello World!
In [2]: 3/2
Out [2]: 1
In [3]: 5.0/2
Out [3]: 2.5
In [4]:
```

همین امر در مورد مقادیر خروجی که با مقدار Out[1]، Out[2] و ... نشان داده می‌شوند نیز صدق می‌کند. آی‌پایتون تمام ورودی‌هایی را که شما برای جای دادن در متغیر وارد می‌کنید، ذخیره می‌کند. در واقع، تمام ورودی‌های وارد شده به عنوان فیلدهای داخل یک لیست به نام In وارد شدند.

```
In [4]: In
Out [4]: ['', u'print "Hello World!"', u'3/2', u'5،0/2', u'_i2', u'In']
```

شاخص‌های (index) عناصر لیست دقیقا مقادیری هستند که در هر مرحله ظاهر می‌شوند. بنابراین، برای دسترسی به یک خط ورودیِ منفرد می‌توانید به سادگی آن مقدار را مشخص کنید.

```
In [5]: In [3]
Out [5]: u'5،0/2'
```

حتی برای خروجی هم می‌توانید از آن استفاده کنید.

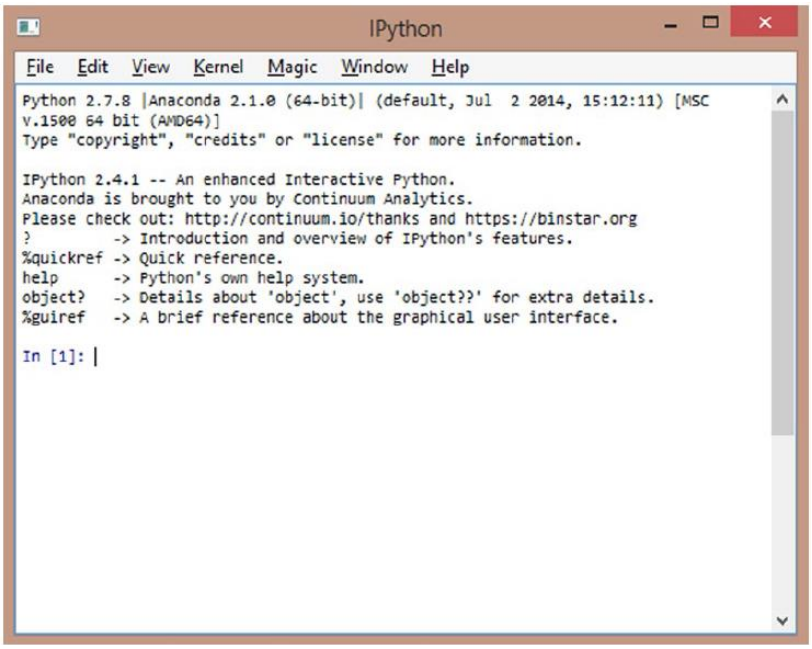
```
{2: 1,
3: 2.5,
4: ['',
u'print "Hello World!"',
u'3/2',
u'5.0/2',
u'_i2',
u'In',
u’In [3]',
u'Out'],
5: u'5.0/2'}
```

IPython Qt-Console

برای راه‌اندازی این برنامه از command line باید فرمان زیر را وارد کنید:

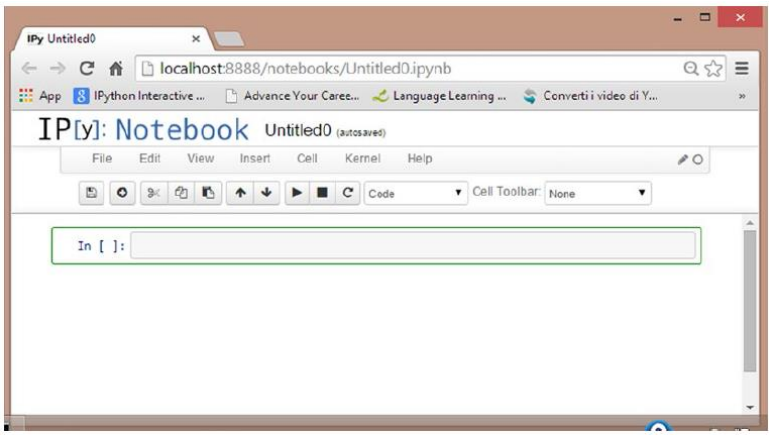
ipython qtconsole

این برنامه شامل یک GUI است که در آن شما تمام قابلیت‌های موجود در پوسته (Shell) آی‌پایتون را دارید. شکل زیر را مشاهده کنید.



IPython Notebook

IPython Notebook آخرین تکامل این محیط تعاملی است. در واقع، با IPython Notebook ، شما می‌توانید کد اجرایی، متن، فرمول‌ها، تصاویر و انیمیشن‌ها را در یک سند وب واحد ادغام کنید، که برای اهداف بسیاری مانند ارایه، آموزش، اشکال‌زدایی و ... مفید است.



پروژهٔ ژوپیتر

آی‌پایتون پروژه‌ای است که در زمان‌های اخیر رشد چشمگیری داشته‌است، و با انتشار آی‌پایتون ۳.۰، همه چیز به سمت یک پروژه جدید به نام Jupyter پیش می‌رود (https://jupyter.org). آی‌پایتون همچنان به عنوان پوسته پایتون و به عنوان هسته ژوپیتر وجود خواهد داشت، اما Notebook و دیگر اجزای ندانم‌گرایِ زبانیِ متعلق به پروژه آی‌پایتون برای تشکیل پروژه جدید ژوپیتر حرکت خواهند کرد.

PyPI - پکیج ضمیمه پایتون

پکیج ضمیمه پایتون (PyPI) یک مخزن نرم‌افزاری است که شامل تمام نرم‌افزارهای مورد نیاز برای برنامه‌نویسی در پایتون است. مخزن محتوا مستقیما توسط توسعه دهندگان پکیج‌های فردی مدیریت می‌شود که با به روز رسانی مخزن با آخرین نسخه‌های کتابخانه‌های منتشر شده‌شان سر و کار دارند. برای مشاهده یک لیست از پکیج‌های موجود در مخزن باید به دیدن صفحه رسمی PyPI با این لینک بروید:

https://pypi.python.org/pypi

در کنار مدیریت این پکیج‌ها، شما می‌توانید از برنامه کاربردی pip استفاده کنید که مدیر پکیج‌های PyPI است. با راه‌اندازی آن از خط فرمان، شما می‌توانید همه پکیج‌ها را به تنهایی مدیریت کنید و تصمیم بگیرید که بسته نصب یا حذف شود و یا ارتقا یابد. pip چک خواهد کرد که آیا بسته قبلا نصب شده‌است، و آیا لازم است به روز شود، تا وابستگی‌های آن کنترل شود، یعنی، برای ارزیابی اینکه آیا بسته‌های دیگر لازم هستند یا خیر؛ علاوه بر این، دانلود و نصب آن‌ها را مدیریت می‌کند.

```
$ pip install <<package_name>>
$ pip search <<package_name>>
$ pip show <<package_name>>
```

```
$ pip unistall <<package_name>>
```

با توجه به نصب، اگر شما پایتون +۳.۴ (منتشر شده در مارس ۲۰۱۴) و پایتون +۲.۷.۹ (منتشر شده در دسامبر ۲۰۱۴) را دارید، نرم‌افزار pip در حال حاضر در این نسخه‌های پایتون گنجانده شده‌است. اگر هنوز از نسخه قدیمی پایتون استفاده می‌کنید، باید pip را بر روی سیستم‌تان نصب کنید. نصب pip روی سیستم‌تان به سیستم‌عامل شما بستگی دارد.

روی Linux Debian-Ubuntu:

```
$ sudo apt-get install python-pip
```

روی Linux Fedora

```
$ sudo yum install python-pip
```

روی ویندوز

سایت *www.pip-installer.org/en/latest/installing.html* را ببینید و get-pip.py را دانلود نمایید. وقتی پرونده دانلود شد، فرمان را اجرا کنید.

```
python get-pip.py
```

به این ترتیب، شما مدیر (manager) پکیج را نصب خواهید کرد. فراموش نکنید که آدرس زیر را به متغیر محیطی (environment variable) PATH بیفزایید:

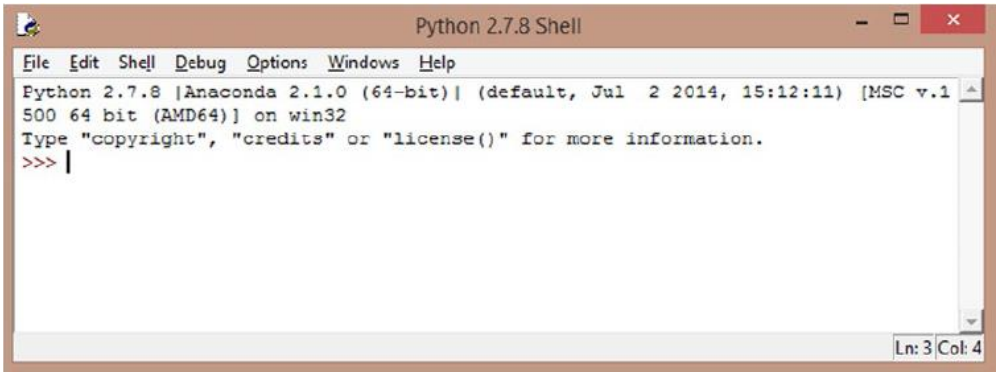
```
C:\Python2.X\Scripts
```

IDE های پایتون

اگر چه بیشتر توسعه‌دهندگان پایتون برای پیاده‌سازی کد خود به طور مستقیم از پوسته یا Shell (پایتون یا آی‌پایتون) استفاده می‌کنند، برخی از IDE ها (محیط‌های توسعه تعاملی) نیز در دسترس هستند. در واقع، علاوه بر یک ویرایشگر متن، این ویرایشگرهای گرافیکی مجموعه‌ای از ابزارها را فراهم می‌کنند که در طول تهیه پیش‌نویس کد بسیار مفید هستند. برای مثال، تکمیل خودکار کد، مشاهده مستندات مربوط به دستورها، اشکال‌زدایی، و breakpoint ها تنها برخی از ابزارهایی هستند که این نوع برنامه می‌تواند فراهم کند.

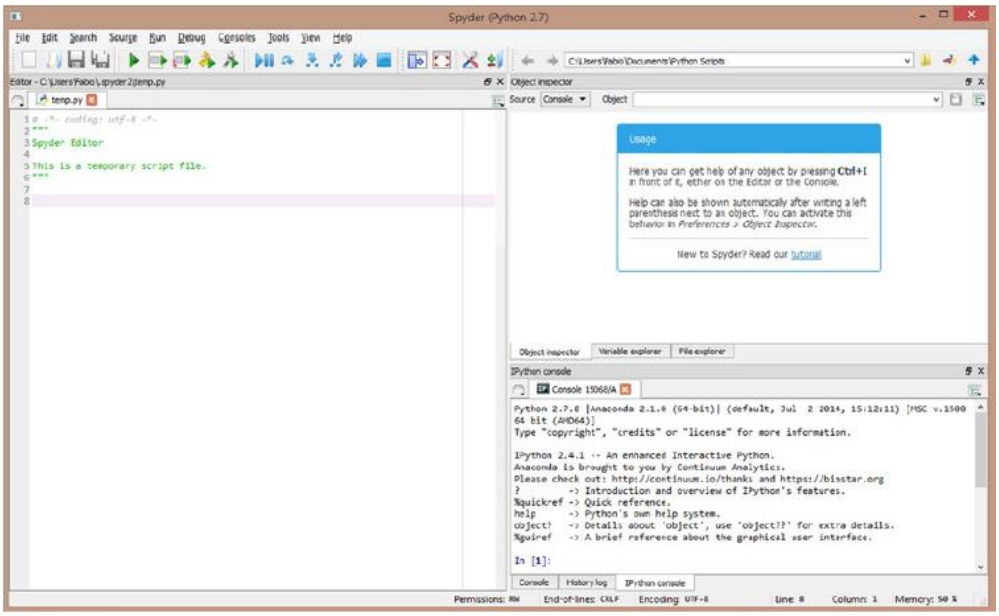
IDLE (محیط توسعه یکپارچه)

IDLE یک محیط برنامه‌نویسی بسیار ساده است که به طور خاص برای توسعه در پایتون ایجاد شده‌است. این محیط فراگیر رسمی در انتشار استاندارد پایتون گنجانده شده‌است، بنابراین در توزیع استاندارد پایتون تعبیه شده‌است (شکل زیر را ببینید). IDLE یک نرم‌افزار است که به طور کامل در پایتون اجرا می‌شود.



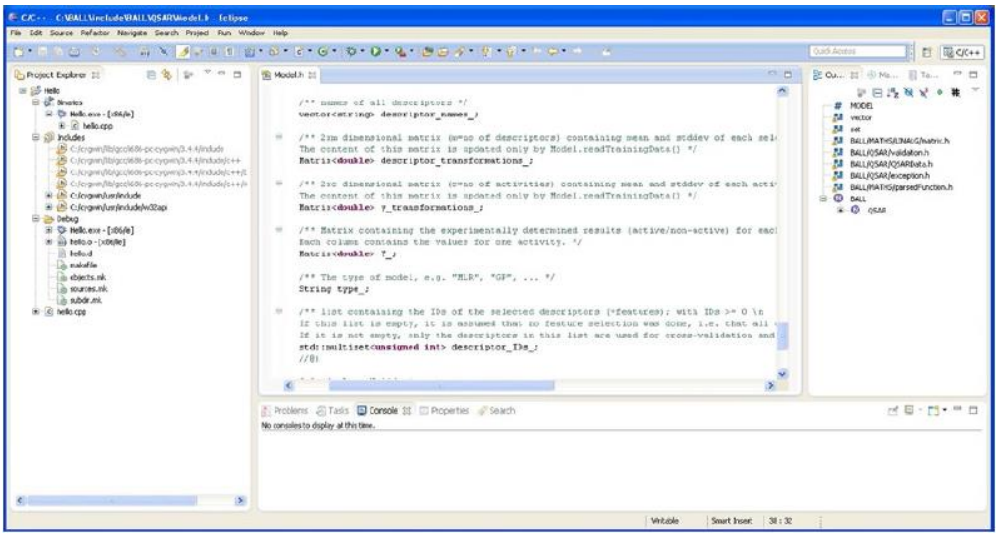
Spyder

Spyder (محیط توسعه علمی پایتون) یک محیط فراگیر است که ویژگی‌های مشابهی با IDE متلب دارد (شکل زیر را ببینید). ویرایشگر متن با ابزارهای برجسته کردن نحو (سینتکس) و تجزیه و تحلیل کد غنی شده‌است. همچنین با استفاده از این محیط شما می‌توانید ویجت های آماده برای استفاده را در برنامه‌های گرافیکی خود ادغام کنید.



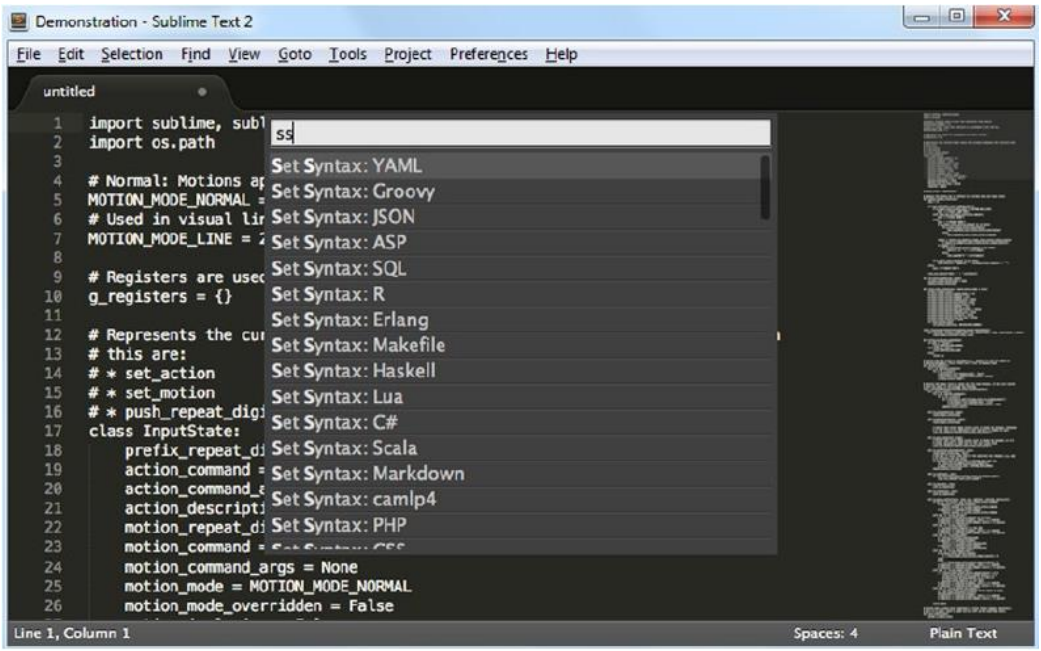
Eclipse (PyDev)

کسانی که با زبان‌های برنامه نویسی دیگری به توسعه دهندگی پرداخته‌اند، قطعاً Eclipse را می‌شناسند. یک نسخه Eclipse برای توسعه در پایتون به لطف نصب یک پلاگین اضافی به نام pyDev وجود دارد.



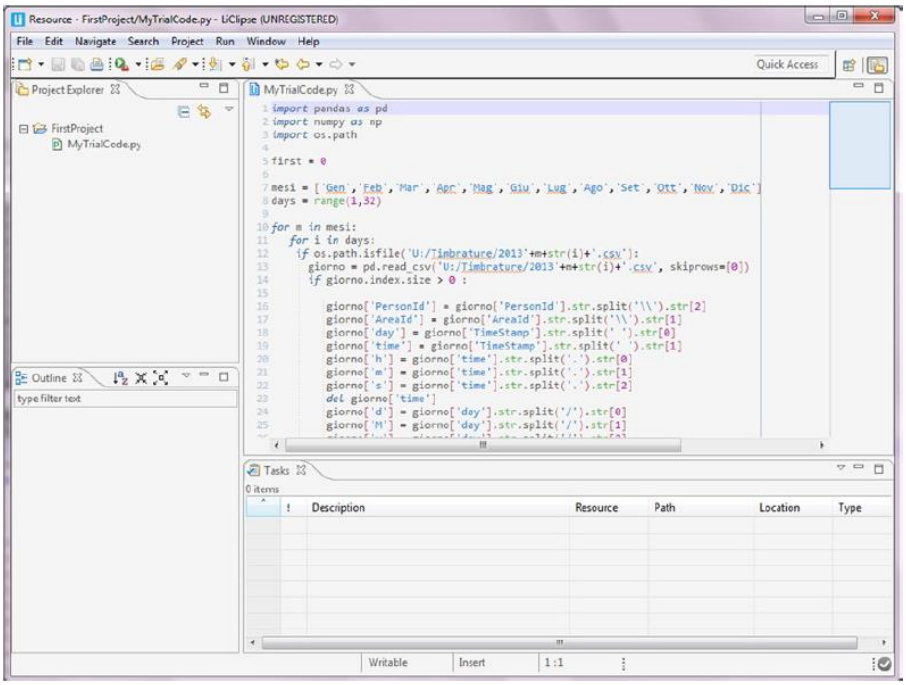
Sublime

این ویرایشگر متن یکی از محیط‌های ارجح برای برنامه نویسان پایتون است (شکل را ببینید). در واقع، چندین پلاگین برای این برنامه موجود است که اجرای پایتون را آسان و لذت بخش می‌کند.



Liclipse

Liclipse، مشابه با Spyder، یک محیط توسعه است که به طور خاص برای زبان پایتون طراحی شده‌است (شکل زیر را ببینید). اساسا کاملا شبیه به محیط Eclipse است اما به طور کامل برای استفاده خاص از پایتون، بدون نصب پلاگین هایی مانند PyDev، سازگار شده‌است. بنابراین نصب و چیدمان آن بسیار ساده‌تر از Eclipse است.



NinjaIDE

NinjaIDE (مخفف «Not Just Another **IDE**» به معنای نه فقط یک محیط دیگر) با نامی مشخص می‌شود که «مخفف بازگشتی» است و یک محیط ویژه برای زبان پایتون است. این یک برنامه بسیار جدید است که تلاش‌های بسیاری از توسعه دهندگان بر روی آن متمرکز شده‌اند. از آن‌جا که این محیط در حال حاضر بسیار امیدوار کننده است، به احتمال زیاد در سال‌های آینده، این IDE منبع بسیاری از شگفتی‌ها خواهد بود.

Komodo IDE

Komodo یک محیط بسیار قدرتمند سرشار از ابزارها است که آن را به یک محیط توسعه کامل و حرفه‌ای تبدیل می‌کند. این نرم‌افزار که در C++ نوشته شده‌است، یک IDE است که با بسیاری از زبان‌های برنامه‌نویسی از جمله پایتون سازگار می‌باشد.

SciPy

SciPy مجموعه‌ای از کتابخانه‌های پایتون است که برای محاسبات علمی تخصص دارد. در میان کتابخانه‌هایی که بخشی از SciPy هستند، برخی از آن‌ها در بخش‌های زیر مورد بحث قرار خواهند گرفت: NumPy، Matplotlib و Pandas.

NumPy

این کتابخانه، که نامش به معنی *پایتون عددی* است، در واقع هسته بسیاری از کتابخانه‌های دیگر پایتون را تشکیل می‌دهد که از آن سرچشمه گرفته‌اند. در واقع، کتابخانه اصلی محاسبات علمی در پایتون است زیرا ساختارهای داده‌ای و توابع با عملکرد بالا را فراهم می‌کند که پکیج اصلی پایتون نمی‌تواند فراهم کند. در واقع، همانطور که در ادامه این کتاب خواهید دید، NumPy یک ساختار داده خاص را تعریف می‌کند که یک آرایه N - بعدی تعریف‌شده به عنوان ndarray است.

دانش این کتابخانه در واقع از نظر محاسبات عددی ضروری است زیرا استفاده صحیح از آن می‌تواند تا حد زیادی عملکرد محاسبات را تحت‌تاثیر قرار دهد. در سراسر کتاب، این کتابخانه به دلیل ویژگی‌های منحصر به فرد خود تقریبا در همه جا حاضر خواهد بود، بنابراین بحث در مورد آن در یک فصل ضروری به نظر می‌رسد. این بسته ویژگی‌هایی را فراهم می‌کند که به پایتون استاندارد اضافه خواهد شد:

◀ ndarray: آرایه‌ای چند بعدی بسیار سریع‌تر و کارآمدتر از آرایه‌هایی است که توسط بسته ساده پایتون فراهم شده‌است.

◀ محاسبات اِلمان‌کاوانه: مجموعه‌ای از توابع برای انجام این نوع محاسبات با آرایه‌ها و عملیات ریاضی بین آرایه‌ها.

◀ مجموعه داده‌های خواندن - نوشتن: مجموعه‌ای از ابزارها برای خواندن و نوشتن داده‌های ذخیره‌شده در Hard Disk.

◀ یکپارچگی با زبان‌های دیگر مانند C، C++ و فورترن: مجموعه‌ای از ابزارها برای ادغام کد توسعه‌یافته با این زبان‌های برنامه‌نویسی.

Pandas

این پکیج ساختارها و توابع داده‌ای پیچیده‌ای را فراهم می‌کند که به طور خاص برای آسان، سریع و موثر کردن کار بر روی آن‌ها طراحی شده‌اند. این پکیج هسته تحلیل داده‌ها با پایتون است. بنابراین، مطالعه و استفاده از آن دلیل اصلی کار شما در سراسر کتاب خواهد بود. آگاهی از Pandas در حد جزییات، به خصوص هنگامی که برای تحلیل داده‌ها به کار می‌رود، یک هدف اساسی این کتاب است. مفهوم اصلی این پکیج قالب دیتا (DataFrame) است، یک ساختار داده جدولی دو بعدی با برچسب‌های سطر و ستون. Pandas ویژگی‌های عملکرد بالای کتابخانه NumPy را با هم ترکیب می‌کند تا آن‌ها را برای دستکاری داده‌ها در صفحات گسترده یا پایگاه‌های داده رابطه‌ای (پایگاه‌داده SQL) به کار ببرد. در واقع، با استفاده از نمایه‌سازی پیچیده، انجام عملیات زیادی بر روی این نوع از ساختارهای داده مانند تغییر شکل، برش، تجمع و انتخاب زیر مجموعه‌ها آسان خواهد بود.

Matplotlib

این پکیج یک کتابخانه پایتون است که در حال حاضر برای تولید طرح‌ها و دیگر تجسم داده‌ها به صورت ۲D بسیار محبوب است. از آن‌جا که تجزیه و تحلیل داده‌ها به ابزارهای تجسم نیاز دارد، این کتابخانه‌ای است که برای این منظور مناسب‌تر است.

نتیجه

در این فصل تمام جنبه‌های اساسی توصیف جهان پایتون نشان‌داده شد. زبان برنامه‌نویسی پایتون در مفاهیم اولیه خود با مثال‌های کوتاه معرفی می‌شود و جنبه‌های نوآورانه ای که معرفی می‌کند و به ویژه این که چگونه در مقایسه با زبان‌های برنامه‌نویسی دیگر برجسته است را توضیح می‌دهد. علاوه بر این، روش‌های مختلفی برای استفاده از پایتون در سطوح مختلف ارایه شده‌است. ابتدا شما مشاهده کردید که چگونه از یک مفسر خط فرمان ساده استفاده کنید، سپس مجموعه‌ای از رابط‌های کاربری گرافیکی ساده نشان داده می‌شوند تا زمانی که به محیط‌های توسعه پیچیده مانند Spyder و NinjaIDE برسید. حتی پروژه بسیار نوآورانه آی‌پایتون ارایه شد، که امکان توسعه تعاملی کد پایتون، به ویژه در مورد Ipython Notebook را نشان می‌داد. علاوه بر این، ماهیت ماژولار پایتون با توانایی گسترش مجموعه توابع اساسی استاندارد ارایه‌شده توسط پایتون با کتابخانه‌های خارجی برجسته شد. در این راستا، مخزن آنلاین PyPI همراه با دیگر توزیع‌های پایتون مانند آناکوندا و Enthought Canopy معرفی گردید.

فصل سه: کتابخانه NumPy

NumPy یک پکیج پایه برای محاسبات علمی با پایتون و به خصوص تجزیه و تحلیل داده‌ها است. در اصل، این کتابخانه پایه مقدار زیادی از پکیج‌های علمی و ریاضی پایتون است و در میان آن‌ها، همانطور که در ادامه کتاب خواهید دید، کتابخانه Pandas قرار دارد. این کتابخانه، که کاملا برای تحلیل داده‌ها تخصص دارد، به طور کامل با استفاده از مفاهیم معرفی‌شده توسط NumPy ایجاد شده‌است. ابزارهای داخلی ارایه‌شده توسط کتابخانه پایتون استاندارد می‌تواند برای بسیاری از محاسبات در تحلیل داده‌ها بسیار ساده یا ناکافی باشد.

بنابراین، دانش کتابخانه NumPy یک پیش‌نیاز برای روبه‌رو شدن با تمام بسته‌های علمی پایتون و به ویژه استفاده و درک بیشتر در مورد کتابخانه پاندا و چگونگی بیرون آمدن از آن است. کتابخانه Pandas موضوع اصلی در فصل بعدی خواهد بود. اگر با این کتابخانه آشنا هستید، می‌توانید مستقیما به بخش بعدی بروید؛ در غیر این صورت می‌توانید این فصل را به عنوان راهی برای بازبینی مفاهیم پایه و یا بازیابی آشنایی با آن با ارائه مثال‌های توصیف‌شده در نظر بگیرید.

NumPy: تاریخچه‌ای مختصر

در ابتدای زبان پایتون، توسعه دهندگان نیاز به محاسبات عددی را احساس کردند، به خصوص زمانی که این زبان توسط جامعه علمی مورد توجه قرار گرفت. اولین تلاش Numeric بود، که توسط جیم هوگونین در سال ۱۹۹۵ توسعه داده شد، که پس از آن یک پکیج جایگزین به نام Numarray قرار گرفت. هر دو پکیج برای محاسبه آرایه‌ها اختصاص داده شده بودند و هر کدام از آن‌ها بسته به حالتی که مورد استفاده قرار گرفته بود، نقاط قوتی داشت. این ابهام منجر به ایده یکی کردن دو پکیج شد و بنابراین تراویس اولیفنت شروع به توسعه کتابخانه NumPy کرد. اولین انتشار آن (v ۱.۰) در سال ۲۰۰۶ رخ داد. از آن به بعد، ثابت شده‌است که NumPy کتابخانه الحاقی پایتون برای محاسبات علمی است و در حال حاضر به طور گسترده برای محاسبه آرایه‌های چندبعدی و آرایه‌های بزرگ استفاده می‌شود. علاوه بر این، NumPy دارای طیفی از توابع است که به شما این امکان را می‌دهد تا عملیات روی آرایه‌ها را به روشی بسیار کارآمد انجام داده و محاسبات ریاضی سطح بالا را انجام دهید. در حال حاضرNumPy منبع - باز است و تحت مجوز BSD قرار دارد.

همکاران زیادی وجود دارند که با حمایت آن‌ها پتانسیل این کتابخانه افزایش یافته است.

نصب NumPy

به طور کلی، این ماژول به عنوان یک پکیج پایه در بیشتر توزیع‌های پایتون موجود است؛ با این حال، اگر نه، می‌توانید بعداً آن را نصب کنید.

روی Linux (Ubuntu and Debian)

```
sudo apt-get install python-numpy
```

روی Linux (Fedora)

```
sudo yum install numpy scipy
```

روی ویندوز و آناکوندا

```
conda install numpy
```

بعد از نصب NumPy، درون نشست (Session) پایتون خود بنویسید:

```
import numpy as np
```

Ndarray: قلب کتابخانه

تمام کتابخانه NumPy بر اساس Ndarray (مخفف آرایهٔ N بعدی) ساخته شده است. این شی یک آرایه همگن چندبعدی با تعدادی موارد از پیش تعیین‌شده است: همگن به این دلیل که تقریباً تمام موارد درون آن از لحاظ نوع و اندازه یکسان هستند. در حقیقت، نوع داده‌ای به وسیله یک شی NumPy دیگر به نام dtype (نوع داده) مشخص می‌شود؛ هر ndarray فقط با یک نوع dtype مرتبط است. تعداد ابعاد و اقلام موجود در یک آرایه با شکل آن تعریف می شود، یک تاپل از N عدد مثبت صحیح که اندازه هر بعد را مشخص می کند. ابعاد به عنوان محور و تعداد محورها به عنوان مرتبه (rank) تعریف می شوند. علاوه بر این، یکی دیگر از ویژگی های آرایه های NumPy این است که اندازه آن‌ها ثابت است، یعنی یک بار که آن‌ها را تعریف کردید اندازهٔ زمان ایجاد، بدون تغییر باقی می ماند. این رفتار با لیست های پایتون متفاوت است که اندازه آن‌ها می تواند رشد کرده یا کوچک گردد. برای تعریف ndarray جدید، ساده ترین راه استفاده از تابع () array است؛ ایجاد یک لیست پایتون که شامل عناصری است که به عنوان آرگومان در آن گنجانده می شوند.

```
>>> a = np. array ([1, 2, 3])
>>> a
array ([1, 2, 3])
```

برای بررسی نوع (type) داریم:

```
>>> type(a)
<type 'numpy. ndarray'>
```

برای بررسی dtype مربوط به ndarray از یک attribute (صفت) به نام dtype استفاده می‌کنیم:

```
>>> a. dtype
dtype('int32')
```

آرایهٔ ایجاد شده یک محور (axis) دارد و لذا مرتبه (rank) آن برابر با ۱ است، در حالی که شکل آن باید (۳,۱) باشد. برای به دست آوردن این مقادیر از آرایهٔ مذکور باید از attribute استفاده کرد؛ برای به دست آوردن محورها از attribute به نام ndim، برای به دست آوردن طول آرایه از attribute به نام size و برای به دست آوردن شکل از attribute به نام shape استفاده می‌گردد.

```
>>> a. ndim
۱
>>> a. size
۳
>>> a. shape
(۳L,)
```

آنچه دیدید ساده‌ترین مورد از آرایه‌ای یک بعدی است. استفاده از آرایه می‌تواند به راحتی به حالت‌هایی با ابعاد مختلف گسترش یابد. برای مثال، اگر یک آرایه دو بعدی دو در دو را تعریف کنید:

```
>>> b = np. array ([[1.3, 2.4], [0.3, 4.1]])
>>> b. dtype
dtype('float64')
>>> b. ndim
```

۲

```
>>> b. size
```

۴

```
>>> b. shape
(2L, 2L)
```

این آرایه دارای مرتبه (rank) ۲ است، چرا که دو محور دارد و هر کدام از آن‌ها طولشان ۲ است. attribute مهم دیگر، itemsize است که می‌تواند با اشیا ndarray استفاده شود. این attribute اندازه هر آیتم آرایه را به صورت بایتی تعریف می‌کند و داده، حائلِ شامل عنصرهای واقعی آرایه است. این attribute دوم هنوز به طور کلی مورد استفاده قرار نگرفته است، زیرا برای دسترسی به داده‌های درون آرایه شما از مکانیزم نمایه‌سازی (indexing) استفاده خواهید کرد که در بخش‌های بعدی خواهید دید.

```
>>> b.itemsize
8
>>> b.data
<read-write buffer for 0x0000000002D34DF0, size 32, offset 0 at 0x0000000002D5FEA0>
```

ایجاد یک آرایه

برای ایجاد یک آرایه جدید می‌توانید مسیرهای مختلفی را دنبال کنید. رایج‌ترین آن‌ها چیزی است که در بخش قبلی از طریق یک لیست یا ترتیب لیست‌ها به عنوان آرگومان برای تابع array() دیدید.

```
>>> c = np.array([[1, 2, 3],[4, 5, 6]])
>>> c
array([[1, 2, 3],
       [4, 5, 6]])
```

تابع array() علاوه بر لیست‌ها می‌تواند تاپل‌ها و توالی تاپل‌ها را نیز بپذیرد.

```
>>> d = np.array(((1, 2, 3),(4, 5, 6)))
>>> d
array([[1, 2, 3],
       [4, 5, 6]])
```

همچنین، توالی‌هایی از تاپل‌ها و لیست‌ها که به هم مرتبط هستند نیز تفاوتی ایجاد نمی‌کنند.

```
>>> e = np.array([(1, 2, 3), [4, 5, 6], (7, 8, 9)])
>>> e
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

انواع داده

تا کنون تنها از مقادیر عددی به عنوان عدد صحیح ساده و اعشاری استفاده کرده‌اید، اما آرایه‌های NumPy طوری طراحی شده‌اند که شامل طیف گسترده‌ای از انواع داده‌ها باشند. برای مثال، می‌توانید از داده از نوع رشته استفاده کنید:

```
>>> g = np.array([['a', 'b'],['c', 'd']])
>>> g
array([['a', 'b'],
           ['c', 'd']],
      dtype='|S1')
>>> g.dtype
dtype('S1')
>>> g.dtype.name
'string8'
```

جدول انواع داده پشتیبانی شده توسط NumPy

نوع داده	توضیح
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C size_t; normally either int32 or int64)
int8	Byte (–128 to 127)
int16	Integer (–32768 to 32767)
int32	Integer (–2147483648 to 2147483647)
int64	Integer (–9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half precision float: sign bit, 5-bit exponent, 10-bit mantissa
float32	Single precision float: sign bit, 8-bit exponent, 23-bit mantissa
float64	Double precision float: sign bit, 11-bit exponent, 52-bit mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

گزینهٔ dtype

تابع array() یک آرگومان منفرد را نمی‌پذیرد. شما دیدید که هر شی ndarray با یک شی dtype مرتبط است که به طور منحصر به فرد نوع داده‌هایی که هر آیتم را در آرایه اشغال خواهند کرد را تعریف می‌کند. به طور پیش‌فرض، تابع array() قادر است مناسب‌ترین نوع را با توجه به مقادیر موجود در توالی لیست‌ها یا نمونه‌های به کار رفته مرتبط سازد. در واقع شما می‌توانید به صراحت dtype را با استفاده از گزینه dtype به عنوان آرگومان تابع تعریف کنید. برای مثال اگر می‌خواهید آرایه‌ای با مقادیر پیچیده تعریف کنید می‌توانید از گزینه dtype به صورت زیر استفاده کنید:

```
>>> f = np.array([[1, 2, 3],[4, 5, 6]], dtype=complex)
>>> f
array([[ 1.+0.j,  2.+0.j,  3.+0.j],
       [ 4.+0.j,  5.+0.j,  6.+0.j]])
```

ایجاد ذاتی یک آرایه

کتابخانه NumPy مجموعه‌ای از توابع را ارایه می‌دهد که ndarray ها را با یک محتوای اولیه تولید می‌کنند که با مقادیر متفاوتی بسته به تابع ایجاد شده‌اند. در طول این بخش و نیز در طول این کتاب، متوجه خواهید شد که این ویژگی‌ها بسیار مفید خواهند بود. در واقع، آن‌ها به یک خط کد اجازه تولید مقادیر زیادی داده را می‌دهند. برای مثال تابع zeros() یک آرایه کامل از صفرها با ابعاد تعریف‌شده توسط آرگومانِ شکل ایجاد می‌کند. برای مثال، برای ایجاد یک آرایه دو بعدی ۳ در ۳:

```
>>> np.zeros((3, 3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

در حالی که عملکرد ones() آرایه‌ای پر از یک‌ها را به روشی بسیار مشابه ایجاد می‌کند.

```
>>> np.ones((3, 3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

به طور پیش‌فرض، این دو تابع، آرایه‌هایی با نوع داده float ۶۴ ایجاد کرده‌اند. یک attribute که به طور خاص مفید خواهد بود arrange() است. این تابع، آرایه‌هایی را با توالی‌های عددی تولید می‌کند که بسته به آرگومان‌های ورودی به قوانین خاصی پاسخ می‌دهند.

برای مثال، اگر بخواهید یک دنباله از مقادیر بین ۰ و ۱۰ ایجاد کنید، تنها یک آرگومان به تابع منتقل خواهد شد، که مقداری است که می‌خواهید با آن دنباله را به پایان برسانید.

```
>>> np.arange(0, 10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

اگر به جای شروع از صفر می‌خواهید از مقدار دیگری شروع کنید، به سادگی دو آرگومان را مشخص کنید: اولی مقدار شروع و دومی مقدار نهایی است.

```
>>> np.arange(4, 10)
array([4, 5, 6, 7, 8, 9])
```

همچنین تولید یک توالی از مقادیر با فواصل دقیق بین آن‌ها امکان پذیر است. اگر آرگومان سوم تابع arrange() مشخص شود، فاصله بین یک مقدار و آرگومان بعدی در توالی مقادیر را نشان می‌دهد.

```
>>> np.arange(0, 12, 3)
array([0, 3, 6, 9])
```

علاوه بر این، این آرگومان سوم نیز می‌تواند یک float (عدد اعشاری) باشد.

```
>>> np.arange(0, 6, 0.6)
array([ 0. ,  0.6,  1.2,  1.8,  2.4,  3. ,  3.6,  4.2,  4.8,  5.4])
```

اما تاکنون شما تنها آرایه‌های یک بعدی ایجاد کرده‌اید. برای تولید آرایه‌های دو بعدی هنوز هم می‌توانید از تابع arrange() استفاده کنید اما در ترکیب با تابع reshape(). این تابع یک آرایه خطی را در بخش‌های مختلف به روشی که توسط آرگومان شکل مشخص شده‌است، تقسیم می‌کند.

```
>>> np.arange(0, 12).reshape(3, 4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

تابع دیگری که بسیار شبیه به arrange() است، linspace() است. این تابع همچنان به عنوان دو آرگومان اول خود مقادیر اولیه و نهایی دنباله را در نظر می‌گیرد، اما آرگومان سوم به جای مشخص کردن فاصله بین یک عنصر و عنصر بعدی، تعداد عناصری که می‌خواهیم فاصله بین آن‌ها تقسیم شود را تعریف می‌کند.

```
>>> np.linspace(0,10,5)
array([  0. ,   2.5,   5. ,   7.5,  10. ])
```

در نهایت، روش دیگر برای به دست آوردن آرایه‌هایی که در حال حاضر حاوی مقادیر هستند، پر کردن آن‌ها با مقادیر تصادفی است.

این امر با استفاده از تابع ()random از ماژول numpy.random امکان پذیر است. این تابع آرایه‌ای با عناصر زیادی که در آرگومان مشخص شده‌اند را تولید می‌کند.

```
>>> np.random.random(3)
array([ 0.78610272,  0.90630642,  0.80007102])
```

اعداد به‌دست‌آمده در هر run متفاوت خواهند بود. برای ایجاد یک آرایه چند بعدی به سادگی اندازه آرایه را به عنوان یک آرگومان در نظر بگیرید.

```
>>> np.random.random((3,3))
array([[ 0.07878569,  0.7176506 ,  0.05662501],
       [ 0.82919021,  0.80349121,  0.30254079],
       [ 0.93347404,  0.65868278,  0.37379618]])
```

عملیات پایه

تا کنون دیده‌اید که چگونه یک آرایه جدید numpy ایجاد کنید و چگونه آیتم‌ها در آن تعریف می‌شوند. اکنون زمان آن است که ببینید چگونه عملیات مختلف را بر روی آرایه‌ها اعمال کنیم.

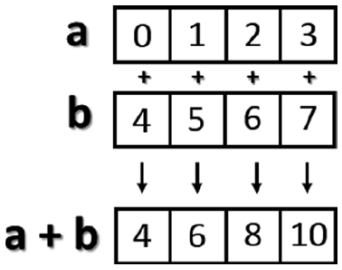
عملگرهای ریاضی

اولین عملیاتی که بر روی آرایه انجام می‌دهید کاربرد عملگرهای حسابی است. مشخص‌ترین آن‌ها جمع یا ضرب یک آرایه با یک اسکالر است.

```
>>> a = np.arange(4)
>>> a
array([0, 1, 2, 3])

>>> a+4
array([4, 5, 6, 7])
>>> a*2
array([0, 2, 4, 6])
```

این عملگرها همچنین می‌توانند بین دو آرایه استفاده شوند. در numpy این عملیات از نظر اِلمان عاقلانه هستند یعنی عملگرها تنها بین عناصر متناظر اعمال می‌شوند، یعنی که همان موقعیت را اشغال می‌کنند، به طوری که در پایان به عنوان نتیجه یک آرایه جدید حاوی نتایج در همان مکان عملوند ها وجود خواهد داشت (شکل را ببینید).




```
>>> b = np.arange(4,8)
>>> b
array([4, 5, 6, 7])
>>> a + b
array([ 4,  6,  8, 10])
>>> a - b
array([-4, -4, -4, -4])
>>> a * b
array([ 0,  5, 12, 21])
```

علاوه بر این، این عملگرها نیز برای توابع در دسترس هستند، به شرطی که مقدار بازگشتی یک آرایه numpy باشد. برای مثال، شما می‌توانید آرایه را در سینوس یا ریشه دوم عناصر آرایه b ضرب کنید.

```
>>> a * np.sin(b)
array([-0.        , -0.95892427, -0.558831  ,  1.9709598 ])
>>> a * np.sqrt(b)
array([ 0.        ,  2.23606798,  4.89897949,  7.93725393])
```

در حال حرکت به سمت حالت چند بعدی، حتی در اینجا عملگرهای حسابی به عملیات جزء به جزء (element-wise) ادامه می‌دهند.

```
>>> A = np.arange(0, 9).reshape(3, 3)
>>> A
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> B = np.ones((3, 3))
>>> B
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> A * B
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.],
       [ 6.,  7.,  8.]])
```

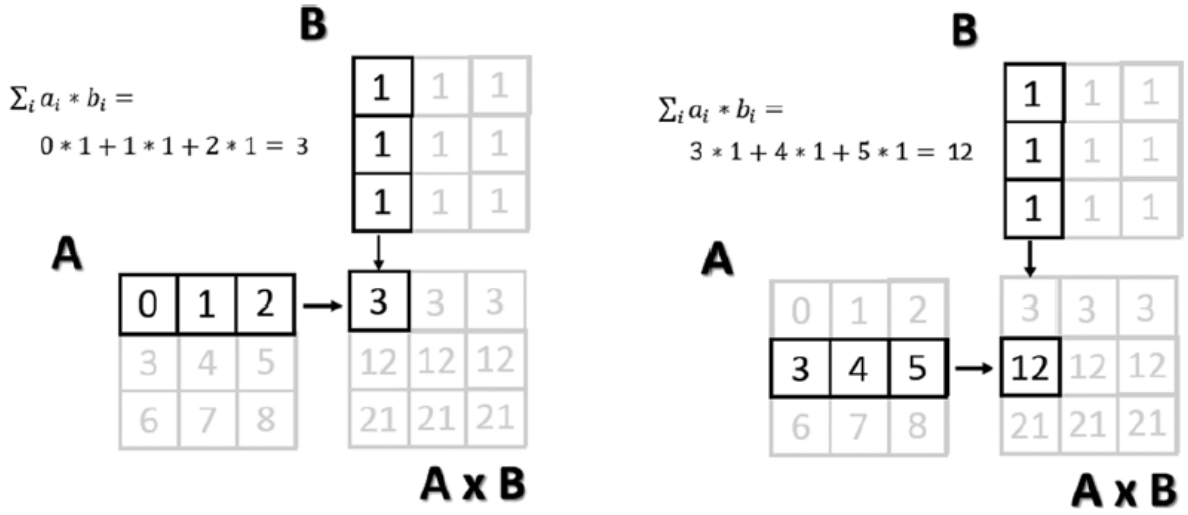
محصول ماتریس

انتخاب عنصر عملیاتی یک جنبه خاص از کتابخانه numpy است.

در واقع در بسیاری از ابزارهای دیگر برای تجزیه و تحلیل داده‌ها، وقتی عملگر به دو ماتریس اعمال می‌شود به عنوان محصول ماتریس شناخته می‌شود. در عوض این نوع محصول با استفاده از تابع dot() نشان داده می‌شود. این عملیات جزء به جزء نیست.

```
>>> np.dot(A,B)
array([[ 3.,  3.,  3.],
       [ 12.,  12.,  12.],
       [ 21.,  21.,  21.]])
```

نتیجه در هر موقعیت مجموع محصولات بین هر عنصر از ردیف متناظر ماتریس اول با عنصر متناظر ستون متناظر ماتریس دوم است. با این حال، شکل زیر فرآیند انجام‌شده در طول محصول ماتریس را نشان می‌دهد (تنها برای دو عنصر اجرا می‌شود).



یک راه دیگر برای نوشتن حاصل‌ضرب ماتریس دیدن تابع dot() به عنوان تابعِ شیِ یکی از دو ماتریس است.

```
>>> A.dot(B)
array([[ 3.,  3.,  3.],
       [ 12.,  12.,  12.],
       [ 21.,  21.,  21.]])
```

می‌خواهم اضافه کنم که چون محصول ماتریس یک عمل جابه‌جایی نیست، پس ترتیب عملوندها مهم است. در واقع A * B برابر با B * A نیست.

```
>>> np.dot(B,A)
array([[ 9.,  12.,  15.],
       [ 9.,  12.,  15.],
       [ 9.,  12.,  15.]])
```

عملگرهای افزایش و کاهش

در واقع، چنین عملگری در پایتون وجود ندارد زیرا هیچ عملگر ++ یا -- وجود ندارد. برای افزایش یا کاهش مقادیر باید از عملگرهایی مانند += یا -= استفاده کنید. این عملگرها متفاوت از آن‌هایی هستند که قبلاً دیدیم، با این تفاوت که به جای ایجاد یک آرایه جدید با نتایج، آن‌ها مجدداً نتایج را به همان آرایه تخصیص خواهند داد.

```
>>> a = np.arange(4)
>>> a
array([0, 1, 2, 3])
>>> a += 1
>>> a
array([1, 2, 3, 4])
>>> a -= 1
>>> a
array([0, 1, 2, 3])
```

توجه کنید:

```
array([0, 1, 2, 3])
>>> a += 4
>>> a
array([4, 5, 6, 7])
>>> a *= 2
>>> a
array([ 8, 10, 12, 14])
```

توابع جهانی (ufunc)

یک تابع جهانی که به طور کلی ufunc نامیده می شود، تابعی است که از یک عنصر به یک عنصر کار می کند. ufunc تابعی است که به صورت جداگانه بر روی تک تک عناصر آرایه ورودی عمل می کند و نتیجه متناظر را در یک آرایه خروجی جدید ایجاد می‌نماید. بسیاری از عملیات ریاضی و مثلثاتی وجود دارد که این تعریف را برآورده می کند برای مثال محاسبه ریشه مربع با sqrt() ، لگاریتم با log() یا سینوس با sin().

```
>>> a = np.arange(1, 5)
>>> a
array([1, 2, 3, 4])
>>> np.sqrt(a)
array([ 1.        ,  1.41421356,  1.73205081,  2.        ])
>>> np.log(a)
array([ 0.        ,  0.69314718,  1.09861229,  1.38629436])
>>> np.sin(a)
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

بسیاری از توابع قبلاً در کتابخانه NumPy پیاده سازی شده اند.

توابع جمع

توابع جمع آن دسته از توابع هستند که عملیاتی را روی مجموعه ای از مقادیر انجام می دهند، مثلاً یک آرایه و یک نتیجه واحد تولید می کنند. بنابراین مجموع تمام عناصر موجود در یک آرایه یک تابع جمع است. اغلب توابع از این نوع در داخل کلاس ndarray اجرا می شود.

```
>>> a = np.array([3.3, 4.5, 1.2, 5.7, 0.3])
>>> a.sum()
15.0
>>> a.min()
0.29999999999999999
>>> a.max()
5.7000000000000002
>>> a.mean()
3.0
>>> a.std()
2.0079840636817816
```

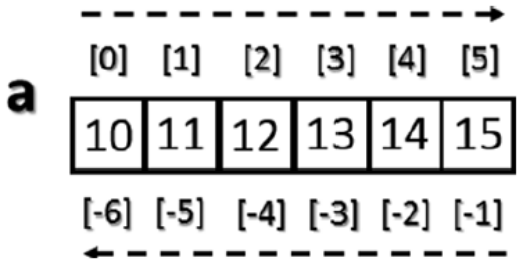
نمایه‌سازی، قطعه‌برداری و تکرار

در بخش‌های قبلی مشاهده کردید که چگونه یک آرایه ایجاد کنید و چگونه بر روی آن عملیات انجام دهید. در این بخش خواهید دید که چگونه این اشیا را دستکاری کنید، چگونه برخی از عناصر را از طریق شاخص‌ها و مقاطع انتخاب کنید، تا دیدگاه‌های ارزش‌های موجود در آن‌ها را به دست آورید یا وظایف را به منظور تغییر ارزش آن‌ها به جا آورید. در نهایت، خواهید دید که چگونه می‌توانید تکرارها (Iterating) را در آرایه‌ها پیاده کنید.

نمایه سازی

نمایه‌سازی آرایه‌ای همیشه به استفاده از براکت های مربعی برای فهرست کردن عناصر آرایه اشاره دارد تا بتوان آن را به صورت جداگانه برای کاربردهای مختلف مانند استخراج یک مقدار (value)، انتخاب آیتم‌ها و یا حتی تخصیص یک مقدار جدید ارجاع داد.

وقتی یک آرایه جدید ایجاد می‌کنید، یک شاخص (index) مقیاس (Scale) مناسب نیز به طور خودکار ایجاد می‌شود (شکل را ببینید).



برای دسترسی به یک عنصر منفرد از یک آرایه می‌توانید به شاخص (index) آن مراجعه کنید.

```
>>> a = np.arange(10, 16)
>>> a
array([10, 11, 12, 13, 14, 15])
>>> a[4]
14
```

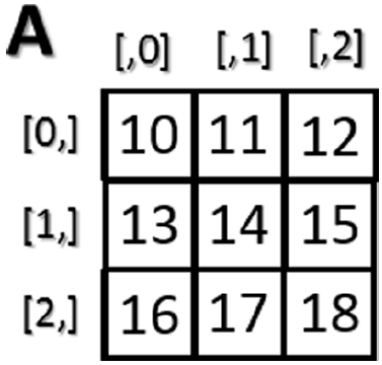
آرایه‌های NumPy شاخص‌های منفی را نیز می‌پذیرند. این شاخص‌ها توالی افزایشی یکسانی از ۰ تا ۱ - و ۲ - و ... دارند، اما در عمل شروع به ارجاع عنصر نهایی می‌کنند تا به تدریج به سمت عنصر اولیه حرکت کنند، که همان عنصر با مقدار شاخص منفی بیشتر خواهد بود.

```
>>> a[-1]
15
>>> a[-6]
10
```

برای انتخاب همزمان چندین آیتم، می‌توانید مجموعه‌ای از شاخص‌ها را به داخل براکت های مربعی (همان []) انتقال دهید.

```
>>> a[[1, 3, 4]]
array([11, 13, 14])
```

با حرکت به سمت حالت دو بعدی یعنی ماتریس‌ها، آن‌ها به صورت آرایه‌های مستطیلی متشکل از سطر و ستون (تعریف‌شده توسط دو محور) نمایش داده می‌شوند که در آن محور ۰ با ردیف‌ها و محور ۱ با ستون‌ها نمایش داده می‌شود. بنابراین شاخص گذاری (indexing) در این حالت با یک جفت از مقادیر نشان داده می‌شود: اولین مقدار شاخص (index) ردیف و دومین شاخص (index) ستون است. بنابراین، اگر می‌خواهید به مقادیر دسترسی داشته باشید یا عناصر درون ماتریس را انتخاب کنید، همچنان از براکت های مربعی استفاده خواهید کرد، اما این بار مقادیر دو [شاخص ستون، شاخص سطر] هستند (شکل را مشاهده نمایید).




```
>>> A = np.arange(10, 19).reshape((3, 3))
>>> A
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

بنابراین اگر می‌خواهید عنصر ستون سوم را در ردیف دوم حذف کنید، باید جفت [۱,۲] را وارد کنید.

```
>>> A[1, 2]
15
```

قطعه‌برداری

قطعه‌برداری عملیاتی است که به شما اجازه می‌دهد بخش‌هایی از یک آرایه را برای تولید بخش‌های جدید استخراج کنید. در حالی که با استفاده از پایتون، آرایه‌هایی که با برش دادن کپی می‌شوند فهرست می‌شوند، در NumPy آرایه‌ها بر روی همان بافر اصلی نمایش داده می‌شوند. بسته به بخشی از آرایه‌ای که می‌خواهید استخراج (یا مشاهده) کنید باید از نحو (سینتکس) slice استفاده کنید؛ به این معنی که شما از ترتیب اعداد جدا شده توسط کولون (":") در داخل براکت های مربعی استفاده خواهید کرد.

اگر می‌خواهید بخشی از آرایه را استخراج کنید، برای مثال یک برش که از عنصر دوم به عنصر ششم می‌رود، پس باید شاخص عنصر شروع را که ۱ است و شاخص عنصر نهایی را که ۵ است، جدا شده با ":" درج کنید.

```
>>> a = np.arange(10, 16)
>>> a
array([10, 11, 12, 13, 14, 15])
>>> a[1:5]
array([11, 12, 13, 14])
```

حال اگر می‌خواهید از بخش قبلی یک آیتم را استخراج کنید، از تعداد مشخصی از موارد زیر صرف‌نظر کنید، موارد بعدی را استخراج کنید و دوباره بپرید ... می‌توانید از عدد سومی استفاده کنید که شکاف موجود در توالی عناصر بین یک عنصر و عنصر بعدی را تعریف می‌کند. به عنوان مثال، با مقدار ۲، آرایه عناصر را به صورت متناوب خواهد گرفت.

```
>>> a[1:5:2]
array([11, 13])
```

برای درک بهتر نحو (سینتکس) برش، باید به مواردی نیز توجه کنید که در آن‌ها از مقادیر عددی صریح استفاده نمی‌کنید. اگر عدد اول را حذف کنید، آنگاه به طور ضمنی NumPy این عدد را به صورت ۰ تفسیر می‌کند (یعنی عنصر اولیه آرایه)؛ اگر عدد دوم را حذف کنید، این به عنوان شاخص بیشینه آرایه تفسیر می‌شود؛ و اگر عدد آخر را حذف کنید، این عدد به صورت ۱ تفسیر می‌شود و سپس همه عناصر بدون فاصله در نظر گرفته می‌شوند.

```
>>> a[::2]
array([10, 12, 14])
>>> a[:5:2]
array([10, 12, 14])
>>> a[:5:]
array([10, 11, 12, 13, 14])
```

با توجه به حالت آرایه دو بعدی، نحو (سینتکس) برش هنوز هم به کار می‌رود اما به طور جداگانه هم برای ردیف‌ها و هم برای ستون‌ها تعریف می‌شود.

```
>>> A = np.arange(10, 19).reshape((3, 3))
>>> A
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
>>> A[0,:]
array([10, 11, 12])
```

همانطور که در شاخص دوم می‌بینید، اگر شما تنها ":" را بدون هیچ عددی به کار ببرید، تمام ستون‌ها را انتخاب خواهید کرد. در عوض، اگر می‌خواهید تمام مقادیر ستون اول را استخراج کنید، باید معکوس آن را بنویسید.

```
>>> A[:,0]
array([10, 13, 16])
```

اگر می‌خواهید یک ماتریس کوچک‌تر را استخراج کنید، باید به صراحت تمام فواصل را با شاخص‌هایی که آن‌ها را تعریف می‌کنند تعریف کنید.

```
>>> A[0:2, 0:2]
array([[10, 11],
       [13, 14]])
```

اگر شاخص‌های ردیف‌ها یا ستون‌هایی که باید استخراج شوند متصل نباشند، می‌توانید یک آرایه از شاخص‌ها را مشخص کنید.

```
>>> A[[0,2], 0:2]
array([[10, 11],
       [16, 17]])
```

تکرار یک آرایه

در پایتون، تکرار آیتم‌های یک آرایه واقعا ساده است؛ فقط باید از for استفاده کنید.

```
>>> for i in a:
...      print i
...
10
11
12
13
14
15
```

البته، حتی در این‌جا، با حرکت به سمت حالت دو بعدی، می‌توانید به استفاده از راه‌حل دو حلقه تودرتو با ساختار for فکر کنید. حلقه اول ردیف‌های آرایه را و حلقه دوم ستون‌ها را اسکن می‌کند. در واقع، اگر حلقه را به یک ماتریس اعمال کنید، متوجه خواهید شد که همیشه یک اسکن را مطابق با محور اول انجام خواهد داد.

```
>>> for row in A:
...      print row
...
[10 11 12]
[13 14 15]
[16 17 18]
```

اگر می‌خواهید یک عنصرِ تکرار توسط اِلمان ایجاد کنید، ممکن است از ساختار زیر استفاده کنید، با استفاده از حلقه در A.flat.

```
>>> for item in A.flat:
...       print item
...
10
11
12
13
14
15
16
17
18
```

با این حال، با وجود تمام این‌ها، NumPy یک راه‌حل جایگزین و ظریف‌تر از حلقه for به ما ارائه می‌دهد. به طور کلی، شما باید یک تکرار را برای اِعمال یک تابع روی سطرها یا ستون‌ها یا روی یک آیتم خاص به کار ببرید. اگر می‌خواهید یک تابع کلی راه‌اندازی کنید که مقدار محاسبه‌شده برای هر ستون یا در هر ردیف را باز می‌گرداند، یک روش بهینه برای ایجاد این امر وجود دارد که برای مدیریت تکرار کاملاً نو خواهد بود: تابع apply_along_axis().

این تابع سه آرگومان دارد: تابع مجموع، محوری که تکرار را روی آن اعمال می‌کند، و در نهایت آرایه. اگر گزینه محور برابر با صفر باشد، تکرار عناصر را ستون به ستون ارزیابی می‌کند، در حالی که اگر محور برابر با ۱ باشد، تکرار ستون عناصر را ردیف به ردیف ارزیابی می‌کند. برای مثال، می‌توانید میانگین مقادیر را برای ستون به ستون و سپس ردیف محاسبه کنید.

```
>>> np.apply_along_axis(np.mean, axis=0, arr=A)
array([ 13.,  14.,  15.])
>>> np.apply_along_axis(np.mean, axis=1, arr=A)
array([ 11.,  14.,  17.])
```

در مورد قبلی، شما از تابعی استفاده کرده‌اید که در حال حاضر در کتابخانه NumPy تعریف شده‌است، اما هیچ چیز شما را از تعریف توابع به خودی خود باز نمی‌دارد. همچنین از یک تابع مجموع استفاده کردید. با این حال، هیچ چیز ما را از استفاده از یک ufunc منع نمی‌کند. در این حالت، با استفاده از تکرار هم در ستون و هم در سطر، نتیجه نهایی یکسان است. در واقع، استفاده از یک ufunc نحوه اجرای یک تکرار عنصر به عنصر است.

```
>>> def foo(x):
...     return x/2
...
>>> np.apply_along_axis(foo, axis=1, arr=A)
array([[5, 5, 6],
       [6, 7, 7],
       [8, 8, 9]])
>>> np.apply_along_axis(foo, axis=0, arr=A)
array([[5, 5, 6],
       [6, 7, 7],
       [8, 8, 9]])
```

همانطور که می‌بینید، تابع ufunc مقدار هر عنصر از آرایه ورودی را بدون توجه به این‌که تکرار توسط سطر یا ستون انجام می‌شود، نصف می‌کند.

شرایط و آرایه‌های بولی

تا کنون از نمایه‌سازی و برش برای انتخاب یا استخراج زیرمجموعه‌ای از آرایه استفاده کرده‌اید. این روش‌ها از شاخص‌ها به شکل عددی استفاده می‌کنند. یک روش جایگزین برای انجام استخراج انتخابی عناصر در یک آرایه استفاده از شرایط و عملگرهای بولی است.

جزئیات این روش جایگزین را ببینید. به عنوان مثال، فرض کنید می‌خواهید همه مقادیر کم‌تر از ۰.۵ را در یک ماتریس ۴x۴ حاوی اعداد تصادفی بین ۰ و ۱ انتخاب کنید.

```
>>> A = np.random.random((4, 4))
>>> A
array([[ 0.03536295,  0.0035115 ,  0.54742404,  0.68960999],
       [ 0.21264709,  0.17121982,  0.81090212,  0.43408927],
       [ 0.77116263,  0.04523647,  0.84632378,  0.54450749],
       [ 0.86964585,  0.6470581 ,  0.42582897,  0.22286282]])
```

هنگامی که یک ماتریس از اعداد تصادفی تعریف شود، اگر شما یک شرطِ عملگر را اعمال کنید، به این معنی که همانطور که گفتیم عملگر بزرگ‌تر است، شما به عنوان یک مقدار بازگشتی آرایه بولی حاوی مقادیر حقیقی در موقعیت‌های که در آن شرط ارضا می‌شود دریافت خواهید کرد، یعنی، تمام موقعیت‌هایی که در آن مقادیر کم‌تر از ۰.۵ هستند.

```
>>> A < 0.5
array([[ True,  True, False, False],
       [ True,  True, False,  True],
       [False,  True, False, False],
       [False, False,  True,  True]], dtype=bool)
```

آرایه‌های بولی به صورت ضمنی برای انتخاب بخش‌های آرایه استفاده می‌شوند. در واقع، با قرار دادن شرایط قبلی به طور مستقیم در داخل براکت های مربعی، شما تمام عنصرهای کوچک‌تر از ۰.۵ را برای به دست آوردن یک آرایه جدید استخراج خواهید کرد.

```
>>> A[A < 0.5]
array([ 0.03536295,  0.0035115 ,  0.21264709,  0.17121982,  0.43408927,
        0.04523647,  0.42582897,  0.22286282])
```

دستکاری شکل

در طول ایجاد یک آرایه دو بعدی مشاهده کردید که چگونه می‌توان به لطف تابع ()reshape، یک آرایه یک بعدی را به یک ماتریس تبدیل کرد.

```
>>> a = np.random.random(12)
>>> a
array([ 0.77841574,  0.39654203,  0.38188665,  0.26704305,  0.27519705,
        0.78115866,  0.96019214,  0.59328414,  0.52008642,  0.10862692,
        0.41894881,  0.73581471])
>>> A = a.reshape(3, 4)
>>> A
array([[ 0.77841574,  0.39654203,  0.38188665,  0.26704305],
       [ 0.27519705,  0.78115866,  0.96019214,  0.59328414],
       [ 0.52008642,  0.10862692,  0.41894881,  0.73581471]])
```

تابع ()reshape یک آرایه جدید بر می‌گرداند و بنابراین برای ایجاد اشیا جدید مفید است. با این حال، اگر می‌خواهید شی را با تغییر شکل اصلاح کنید، باید یک چند تایی حاوی ابعاد جدید را مستقیماً به صفتِ (attribute) شکل آن اختصاص دهید.

```
>>> a.shape = (3, 4)
>>> a
array([[ 0.77841574,  0.39654203,  0.38188665,  0.26704305],
       [ 0.27519705,  0.78115866,  0.96019214,  0.59328414],
       [ 0.52008642,  0.10862692,  0.41894881,  0.73581471]])
```

همانطور که می‌بینید، این بار این آرایه شروعِ تغییر شکل است و هیچ شی بازگشتی وجود ندارد. عملیات معکوس برای تبدیل یک آرایه دو بعدی به یک آرایه یک بعدی از طریق تابع ravel() امکان پذیر است.

```
>>> a = a.ravel()
array([ 0.77841574,  0.39654203,  0.38188665,  0.26704305,  0.27519705,
        0.78115866,  0.96019214,  0.59328414,  0.52008642,  0.10862692,
        0.41894881,  0.73581471])
```

یا حتی در این جا مستقیماً روی صفتِ (attribute) شکل خود آرایه عمل می‌کند.

```
>>> a.shape = (12)
>>> a
array([ 0.77841574,  0.39654203,  0.38188665,  0.26704305,  0.27519705,
        0.78115866,  0.96019214,  0.59328414,  0.52008642,  0.10862692,
        0.41894881,  0.73581471])
```

عملیات مهم دیگر جابه‌جایی یک ماتریس است که وارونگی ستون‌ها با ردیف‌ها است. NumPy این ویژگی را با تابع () transpose نشان می‌دهد.

```
>>> A.transpose()
array([[ 0.77841574,  0.27519705,  0.52008642],
       [ 0.39654203,  0.78115866,  0.10862692],
       [ 0.38188665,  0.96019214,  0.41894881],
       [ 0.26704305,  0.59328414,  0.73581471]])
```

دستکاری آرایه

اغلب نیاز به ایجاد آرایه‌ای با استفاده از آرایه‌های ایجاد شده قبلی دارید. در این بخش خواهید دید که چگونه می توان آرایه‌های جدیدی را با اتصال یا تقسیم آرایه‌های که قبلاً تعریف شده‌اند، ایجاد کرد.

پیوند زدن آرایه‌ها

شما می‌توانید آرایه‌های چندگانه را با هم ادغام کنید تا آرایه جدیدی را تشکیل دهید که شامل همه آن‌ها باشد. NumPy از مفهوم انباشت استفاده می‌کند و تعدادی تابع را در این زمینه فراهم می‌نماید. به عنوان مثال، شما می‌توانید پشته عمودی را با تابع vstack() اجرا کنید، که آرایه دوم را به عنوان ردیف‌های جدید آرایه اول ترکیب می‌کند. در این حالت شما یک آرایه را در جهت عمودی رشد می‌دهید. در مقابل، تابع hstack() پشته افقی را انجام می‌دهد؛ به این معنی که آرایه دوم به ستون‌های آرایه اول اضافه شده‌است.

```
>>> A = np.ones((3, 3))
>>> B = np.zeros((3, 3))
>>> np.vstack((A, B))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.hstack((A,B))
array([[ 1.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.,  0.]])
```

دو تابع دیگر که کار انباشت بین آرایه‌های چندگانه را انجام می‌دهند عبارتند از: column_stack() و row_stack(). این توابع متفاوت از دو تابع قبلی عمل می‌کنند. به طور کلی این توابع با آرایه‌های تک‌بعدی به کار می‌روند که به صورت ستون یا ردیف روی هم چیده شده‌اند تا یک آرایه دو بعدی جدید را تشکیل دهند.

```
>>> a = np.array([0, 1, 2])
>>> b = np.array([3, 4, 5])
>>> c = np.array([6, 7, 8])
>>> np.column_stack((a, b, c))
array([[0, 3, 6],
       [1, 4, 7],
       [2, 5, 8]])
>>> np.row_stack((a, b, c))
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

از هم جدا کردن (Splitting) آرایه‌ها

مشاهده کردید که چگونه می توان چندین آرایه را از طریق عملیات انباشت به یکدیگر متصل کرد. حالا عملیات مخالف آن را خواهید دید، یعنی این که یک آرایه را به چند بخش تقسیم کنید. در NumPy، برای انجام این کار از Splitting استفاده خواهید کرد. در این‌جا نیز یک مجموعه از توابع دارید که هم به‌صورت افقی با تابع hsplit() و هم به‌صورت عمودی با تابع vsplit() کار می‌کنند.

```
>>> A = np.arange(16).reshape((4, 4))
>>> A
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

بنابراین، اگر بخواهید آرایه را به صورت افقی تقسیم کنید، به این معنی که پهنای آرایه به دو بخش تقسیم شود، یک ماتریس ۴x۴ به دو ماتریس ۲x۴ تقسیم می‌شود.

```
>>> [B,C] = np.hsplit(A, 2)
>>> B
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
>>> C
array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])
```

در عوض، اگر بخواهید آرایه را به صورت عمودی تقسیم کنید، به این معنی که ارتفاع آرایه به دو بخش تقسیم شود، یک ماتریس ۴x۴ به دو ماتریس ۴x۲ تقسیم می‌شود.

```
>>> [B,C] = np.vsplit(A, 2)
>>> B
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
>>> C
array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

یک دستور پیچیده‌تر، تابع split() است که به شما اجازه می‌دهد آرایه را به بخش‌های نامتقارن تقسیم کنید. علاوه بر این، با استفاده از آرایه به عنوان یک آرگومان، باید شاخص‌های قسمت‌هایی که باید تقسیم شوند را نیز مشخص کنید. اگر شما از گزینهٔ محور = ۱ استفاده کنید، آنگاه نمایه‌ها مربوط به ستون‌ها خواهند بود. در عوض اگر گزینهٔ محور = ۰ باشد، به عنوان شاخص ردیف در نظر گرفته می‌شود. برای مثال، اگر می‌خواهید ماتریس را به سه بخش تقسیم کنید، اولی شامل ستون اول، دومی شامل ستون دوم و سوم، و سومی شامل ستون آخر خواهد بود، پس باید سه شاخص را به روش زیر مشخص کنید.

```
>>> [A1,A2,A3] = np.split(A,[1,3],axis=1)
>>> A1
array([[ 0],
       [ 4],
       [ 8],
       [12]])
>>> A2
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]])
>>> A3
array([[ 3],
       [ 7],
       [11],
       [15]])
```

می‌توانید عملی مشابه را با ردیف انجام دهید.

```
>>> [A1,A2,A3] = np.split(A,[1,3],axis=0)
>>> A1
array([[0, 1, 2, 3]])
>>> A2
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> A3
array([[12, 13, 14, 15]])
```

این ویژگی همچنین شامل عملکرد توابع vsplit() و hsplit() است.

مفاهیم عمومی

این بخش مفاهیم کلی مربوط به کتابخانه NumPy را توصیف می‌کند. تفاوت بین کپی‌ها و دیدگاه‌ها (views) به خصوص زمانی که مقادیر را برمی گردانند، نشان داده خواهد شد. همچنین مکانیزم پخش که به طور ضمنی در بسیاری از تراکنش‌ها با توابع NumPy رخ می‌دهد، در این بخش پوشش داده خواهد شد.

کپی‌ها یا دیدگاه‌های اشیا

همان طور که ممکن است در مورد NumPy متوجه شده باشید، به خصوص زمانی که عملیات و یا دستکاری را بر روی آرایه انجام می‌دهید، می‌توانید یک مقدار بازگشتی را هم به عنوان یک کپی یا هم یک دیدگاه (view) از آرایه داشته باشید. در NumPy ، همه وظایف، کپی آرایه‌ها و یا هیچ عنصری که درون آن‌ها وجود دارد را تولید نمی‌کنند.

```
>>> a = np.array([1, 2, 3, 4])
>>> b = a
>>> b
array([1, 2, 3, 4])
>>> a[2] = 0
>>> b
array([1, 2, 0, 4])
```

اگر آرایه a را به آرایه b اختصاص دهید، در اصل یک کپی انجام نمی‌دهید بلکه b فقط روش دیگری برای فراخوانی آرایه a است. در واقع با تغییر دادن مقدار عنصر سوم مقدار سومِ b را هم تغییر می‌دهید. زمانی که شما به برش یک آرایه را انجام می‌پردازید، در واقع شی بازگشتی تنها یک دیدگاه (view) از آرایه اصلی است.

```
>>> c = a[0:2]
>>> c
array([1, 2])
>>> a[0] = 0
>>> c
array([0, 2])
```

همانطور که می‌بینید، حتی با برش دادن، شما در واقع به یک شی اشاره می‌کنید. اگر می‌خواهید یک کپی کامل و آرایه متمایز ایجاد کنید، می‌توانید از تابع ()copy استفاده کنید.

```
>>> a = np.array([1, 2, 3, 4])
>>> c = a.copy()
>>> c
array([1, 2, 3, 4])
>>> a[0] = 0
>>> c
array([1, 2, 3, 4])
```

در این حالت، حتی با تغییر موارد در آرایه a، آرایه c بدون تغییر باقی می‌ماند.

بردارسازی

«بردارسازی» مفهومی است که همراه با «پخش» پایه و اساس پیاده‌سازی داخلی NumPy است. بردارسازی عدم وجود حلقه صریح در طول توسعه کد است. این حلقه‌ها در واقع نمی‌توانند حذف شوند، اما به صورت داخلی پیاده‌سازی می‌شوند و سپس با سازه‌های دیگر در کد جایگزین می‌شوند. استفاده از بردارسازی منجر به یک کد دقیق‌تر و خواناتر می‌شود و اصطلاحاً ظاهر کد بیشتر «پایتونیک» خواهد شد. در واقع، به لطف بردارسازی بسیاری از عملیات شکل یک عبارت ریاضی به خود می‌گیرند، به عنوان مثال، NumPy به شما اجازه می‌دهد تا ضرب دو آرایه را همانطور که نشان‌داده‌شده بیان کنید: a*b

یا حتی دو ماتریس: A*B

در زبان‌های دیگر، چنین عملیاتی با تعداد زیادی حلقه تودرتوی for بیان می‌شود. برای مثال، عملیات اول به روش زیر بیان می‌شود:

```
for (i = 0; i < rows; i++){
  c[i] = a[i]*b[i];
}
```

در حالی که محصول ماتریس‌ها به صورت زیر بیان می‌شود:

```
for( i=0; i < rows; i++){
   for(j=0; j < columns; j++){
      c[i][j] = a[i][j]*b[i][j];
   }
}
```

با این وجود، واضح است که با استفاده از NumPy کد خواناتر است و به ویژه به شیوه‌ای ریاضی بیان می‌شود.

پخش

پخش عملیاتی است که به عملگر یا تابع اجازه می‌دهد تا بر روی دو یا چند آرایه عمل کند حتی اگر این آرایه‌هاً دقیقا شکل یکسانی را نداشته باشند. در واقع، همه ابعاد با یکدیگر سازگار نیستند تا تحت پخش باشند اما باید قوانین خاصی را رعایت کنند. دیدید که با استفاده از NumPy، می‌توانید آرایه‌های چند بعدی را از طریق شکلی که چند تایی است و نشان‌دهنده طول عناصر برای هر بعد است، طبقه‌بندی کنید. بنابراین، دو آرایه ممکن است در معرض پخش باشند هنگامی که تمام ابعاد آن‌ها سازگار باشد، یعنی طول هر بعد باید بین دو آرایه برابر باشد یا یکی از آن‌ها باید برابر با ۱ باشد. اگر دو آرایه با هم سازگار نباشند، یک استثنا دریافت خواهید کرد بدین مضمون که دو آرایه همساز (compatible) نمی‌باشند.

```
>>> A = np.arange(16).reshape(4, 4)
>>> b = np.arange(4)
>>> A
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> b
array([0, 1, 2, 3])
```

در این حالت دو آرایه دریافت می‌کنید: ۴×۴

۴

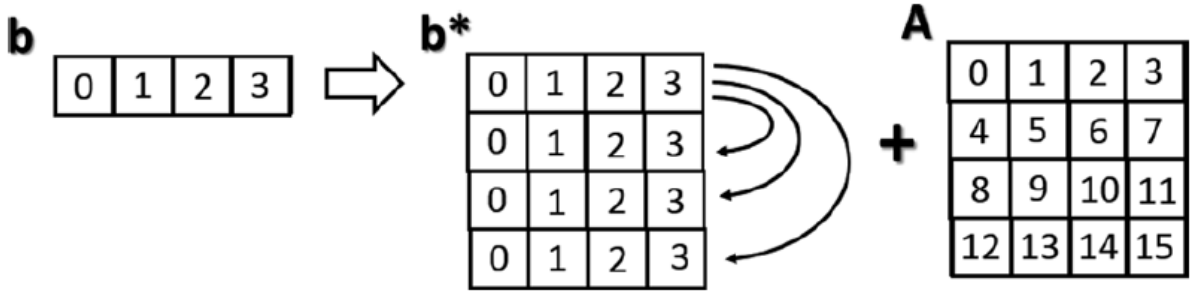
پخش دارای دو قانون است. قانون اول اضافه کردن یک ۱ به هر بعد گم‌شده است. اگر قوانین سازگاری در حال حاضر برآورده شوند، شما می‌توانید پخش برنامه را اعمال کنید و به قانون دوم بروید.

۴×۴

۴×۱

قانون سازگاری برآورده می‌شود. پس می‌توانید به سمت قانون دوم پخش حرکت کنید. این قانون توضیح می‌دهد که چگونه اندازه کوچک‌ترین آرایه را گسترش دهیم به طوری که با اندازه بزرگ‌ترین آرایه یکسان باشد، به طوری که تابع عنصر - محور یا عملگر قابل‌اجرا باشد.

قانون دوم فرض می‌کند که عناصر از دست رفته (اندازه، طول ۱) با نمونه‌های مقادیر موجود در اندازه‌های گسترده پر شده‌اند (شکل زیر را ببینید).



اکنون که این دو آرایه ابعاد یکسانی دارند، مقادیر درون آن‌ها ممکن است با هم جمع شوند.

```
>>> A + b
array([[ 0,  2,  4,  6],
       [ 4,  6,  8, 10],
       [ 8, 10, 12, 14],
       [12, 14, 16, 18]])
```

در این مورد با یک حالت ساده مواجه هستید که در آن یکی از دو آرایه کوچک‌تر از دیگری است. ممکن است حالات پیچیده تری وجود داشته باشد که در آن دو آرایه شکل‌های مختلفی دارند اما هر کدام از آن‌ها تنها در برخی ابعاد کوچک‌تر از دیگری هستند.

```
>>> m = np.arange(6).reshape(3, 1, 2)
>>> n = np.arange(6).reshape(3, 2, 1)
>>> m
array([[[0, 1]],

       [[2, 3]],

       [[4, 5]]])
>>> n
array([[[0],
        [1]],

       [[2],
        [3]],

       [[4],
        [5]]])
```

حتی در این مورد، با تجزیه و تحلیل اشکال دو آرایه، می‌توانید ببینید که آن‌ها سازگار هستند و در نتیجه قوانین پخش می‌تواند اعمال شود.

```
3 x 1 x 2
3 x 2 x 1
```

در این حالت هر دو تحت گسترش ابعاد (پخش) قرار می‌گیرند.

```
m* = [[[0,1],          n* = [[[0,0],
      [0,1]],                [1,1]],
     [[2,3],                [[2,2],
      [2,3]],                [3,3]],
     [[4,5],                [[4,4],
      [4,5]]]                [5,5]]]
```

و سپس می‌توانید به عنوان مثال عملگر جمع را بین دو آرایه به کار ببرید.

```
>>> m + n
array([[[ 0,  1],
        [ 1,  2]],

       [[ 4,  5],
        [ 5,  6]],

       [[ 8,  9],
        [ 9, 10]]])
```

آرایه‌های ساخت‌یافته

تاکنون در مثال‌های مختلف در بخش‌های قبلی، آرایه‌های تک بعدی و دو بعدی را مشاهده کردید. در واقع، NumPy امکان ایجاد آرایه‌هایی را فراهم می‌کند که نه تنها از نظر اندازه، بلکه از نظر خود ساختار، آرایه‌هایی با ساختار دقیق هستند. این نوع آرایه شامل ساختار یا رکوردها به جای اقلام تکی است. برای مثال، شما می‌توانید یک آرایه ساده از ساختارها را به عنوان اقلام ایجاد کنید. به لطف گزینه نوع dtype، شما می‌توانید فهرستی از ویژگی‌های جدا شده با ویرگول را برای نشان دادن عناصری که ساختار را تشکیل می‌دهند، همراه با نوع و ترتیب داده آن مشخص کنید.

```
bytes                b1
int                  i1, i2, i4, i8
unsigned ints        u1, u2, u4, u8
floats               f2, f4, f8
complex              c8, c16
fixed length strings a<n>
```

به عنوان مثال، اگر بخواهید ساختاری شامل یک عدد صحیح، یک رشته کاراکتری با طول ۶ و یک مقدار بولی را مشخص کنید، سه نوع داده در گزینه dtype با ترتیب درست را با استفاده از ویژگی‌های متناظر مشخص خواهید کرد.

```
>>> structured = np.array([(1, 'First', 0.5, 1+2j),(2, 'Second', 1.3, 2-2j),
(3, 'Third', 0.8, 1+3j)],dtype=('i2, a6, f4, c8'))
>>> structured
array([(1, 'First', 0.5, (1+2j)),
       (2, 'Second', 1.2999999523162842, (2-2j)),
       (3, 'Third', 0.800000011920929, (1+3j))],
      dtype=[('f0', '<i2'), ('f1', 'S6'), ('f2', '<f4'), ('f3', '<c8')])
```

همچنین می‌توانید از نوع داده‌ها استفاده کنید که به صراحت ۸int، ۸uint، ۱۶flat و غیره را مشخص می‌کند.

```
>>> structured = np.array([(1, 'First', 0.5, 1+2j),(2, 'Second', 1.3,2-2j),
(3, 'Third', 0.8, 1+3j)],dtype=('
int16, a6, float32, complex64'))
>>> structured
array([(1, 'First', 0.5, (1+2j)),
       (2, 'Second', 1.2999999523162842, (2-2j)),
       (3, 'Third', 0.800000011920929, (1+3j))],
      dtype=[('f0', '<i2'), ('f1', 'S6'), ('f2', '<f4'), ('f3', '<c8')])
```

با این حال، هر دو مورد نتیجه یکسانی دارند. در داخل آرایه شما یک توالی dtype را می‌بینید که شامل نام هر مورد از ساختار با نوع داده‌های مربوطه است. با نوشتن شاخص مرجع مناسب، ردیف مربوطه حاوی ساختار را به دست می‌آورید.

```
>>> structured[1]
(2, 'Second', 1.2999999523162842, (2-2j))
```

نام‌هایی که به طور خودکار به هر یک از مواردِ ساختار اختصاص داده می‌شوند را می توان به عنوان نام‌های ستون‌های آرایه در نظر گرفت و سپس با استفاده از آن‌ها به عنوان یک شاخص ساختار یافته، می‌توانید به تمام عناصر یک نوع، یا همان «ستون» مراجعه کنید.

```
>>> structured['f1']
array(['First', 'Second', 'Third'],
      dtype='|S6')
```

همانطور که دیدید، نام‌ها به طور خودکار با f (که مخفف فیلد است) و عدد صحیح تصاعدی که موقعیت در دنباله را نشان می‌دهد، اختصاص داده می‌شوند.

در واقع، مشخص کردن نام‌ها با چیزی با معنی تر مفیدتر خواهد بود. این امکان پذیر است و شما می‌توانید آن را در زمان اعلام ردیف انجام دهید:

```
>>> structured = np.array([(1,'First',0.5,1+2j),(2,'Second',1.3,2-2j),(3,'Third',0.8,1+3j)],
dtype=[('id','i2'),('position','a6'),('value','f4'),('complex','c8')])
>>> structured
array([(1, 'First', 0.5, (1+2j)),
       (2, 'Second', 1.2999999523162842, (2-2j)),
       (3, 'Third', 0.800000011920929, (1+3j))],
      dtype=[('id', '<i2'), ('position', 'S6'), ('value', '<f4'), ('complex', '<c8')])
```

یا در زمان بعدی، نمونه‌های نام‌های اختصاص‌یافته به ویژگی dtype آرایه ساختار یافته را بازتعریف می‌کنیم.

```
>>> structured. dtype. names = ('id','order','value','complex')
```

حالا می‌توانید از نام‌های معنی‌دار برای انواع مختلف زمینه‌ها استفاده کنید:

```
>>> structured['order']
array(['First', 'Second', 'Third'],
      dtype='|S6')
```

خواندن و نوشتن داده‌های آرایه بر روی پرونده‌ها (فایل‌ها)

یک جنبه بسیار مهم از NumPy که هنوز در نظر گرفته نشده است، خواندن داده‌های موجود در یک فایل است. این روش بسیار مفید است، به خصوص زمانی که شما باید با مقادیر زیادی از داده‌های جمع‌آوری‌شده در آرایه سر و کار داشته باشید. این یک عملیات بسیار رایج در تجزیه و تحلیل داده‌ها است، زیرا اندازه مجموعه داده‌ها که باید آنالیز شود تقریبا همیشه بزرگ است و بنابراین مدیریت رونویسی و خواندن بعدی داده‌ها توسط کامپیوتر به صورت دستی یا از یک نشست (Session) محاسبه به نشست دیگر قابل توصیه یا حتی ممکن نیست.

در واقع، در این زمینه مجموعه‌ای از توابع ارایه می‌شود که به تحلیلگر داده اجازه می‌دهد تا نتایج محاسبات خود را در یک متن یا فایل دودویی ذخیره کند. به طور مشابه، NumPy اجازه خواندن و تبدیل داده‌های نوشته شده درون یک فایل به یک آرایه را می‌دهد.

بارگذاری و ذخیره داده‌ها در پرونده‌های دودویی

با توجه به ذخیره و سپس بازیابی داده‌های ذخیره‌شده در فرمت دودویی، NumPy یک جفت از توابع به نام save() و load() را فراهم می‌کند.

به عنوان مثال، هنگامی که شما یک آرایه برای ذخیره دارید، حاوی نتایج پردازش خود در طول تحلیل داده، به سادگی تابع save() را صدا می‌زنید و به عنوان آرگومان نام فایل را مشخص می‌کنید.

```
>>> data
array([[ 0.86466285,  0.76943895,  0.22678279],
       [ 0.12452825,  0.54751384,  0.06499123],
       [ 0.06216566,  0.85045125,  0.92093862],
       [ 0.58401239,  0.93455057,  0.28972379]])
>>> np.save('saved_data',data)
```

هنگام نیاز به بازیابی داده‌های ذخیره شده درون یک فایل npy. باشد از تابع load() استفاده می‌شود.

```
>>> loaded_data = np.load('saved_data.npy')
>>> loaded_data
array([[ 0.86466285,  0.76943895,  0.22678279],
       [ 0.12452825,  0.54751384,  0.06499123],
       [ 0.06216566,  0.85045125,  0.92093862],
       [ 0.58401239,  0.93455057,  0.28972379]])
```

خواندن پرونده دارای داده‌های جدولی

بسیاری از اوقات، داده‌هایی که می‌خواهید بخوانید یا ذخیره کنید، در قالب textural هستند (برای مثال در فرمت TXT یا CSV). به طور کلی شما تصمیم می‌گیرید که اطلاعات را به جای دودویی در این فرمت ذخیره کنید، زیرا اگر با NumPy یا هر برنامه دیگری کار کنید، می‌توانید به طور مستقل به فایل‌ها دسترسی پیدا کنید. به عنوان مثال حالت مجموعه‌ای از داده‌ها در فرمت CSV (مقادیر جدا شده) را در نظر بگیرید، که در آن داده‌ها به شکل جدولی جمع‌آوری می‌شوند و در آن تمام مقادیر با کاما از یکدیگر جدا می‌گردند.

```
id,value1,value2,value3
1,123,1.4,23
2,110,0.5,18
3,164,2.1,19
```

برای این‌که بتوانید داده‌های خود را در یک فایل متنی بخوانید و آن‌ها را در یک آرایه قرار دهید، NumPy یک تابع به نام genfromtxt() فراهم می‌کند.

به طور معمول، این تابع سه آرگومان دارد، از جمله نام فایل حاوی داده‌ها، مشخصه‌ای که یک مقدار را از دیگری جدا می‌کند که در این مورد یک پاراگراف است و این‌که آیا شامل سرستون‌ها است یا نه.

```
>>> data = np.genfromtxt('data.csv', delimiter=',', names=True)
>>> data
array([(1.0, 123.0, 1.4, 23.0), (2.0, 110.0, 0.5, 18.0),
       (3.0, 164.0, 2.1, 19.0)],
      dtype=[('id', '<f8'), ('value1', '<f8'), ('value2', '<f8'), ('value3', '<f8')])
```

همانطور که می‌توانیم از نتایج ببینیم، شما یک آرایه ساختاری به دست می‌آورید که در آن عناوین ستون به نام‌های این حوزه تبدیل شده‌اند. این تابع به طور ضمنی دو حلقه را اجرا می‌کند: اول یک خط را در یک زمان می‌خواند و دوم مقادیر موجود در آن را جدا و تبدیل می‌کند و عناصر متوالی را که به طور خاص ایجاد شده‌اند جای‌گذاری می‌نماید. یک جنبه مثبت این ویژگی این است که اگر در داخل فایل تعدادی داده گم‌شده وجود داشته باشد، تابع می‌تواند آن‌ها را کنترل کند.

```
id,value1,value2,value3
1,123,1.4,23
2,110,,18
3,,2.1,19
```

با راه‌اندازی این فرمان‌ها، می‌توانید ببینید که چگونه تابع genfromtxt() جای خالی فایل با مقادیر nan را می‌گیرد.

```
>>> data2 = np.genfromtxt('data2.csv', delimiter=',', names=True)
>>> data2
array([(1.0, 123.0, 1.4, 23.0), (2.0, 110.0, nan, 18.0),
       (3.0, nan, 2.1, 19.0)],
      dtype=[('id', '<f8'), ('value1', '<f8'), ('value2', '<f8'), ('value3', '<f8')])
```

در پایین ردیف، شما می‌توانید سر ستون‌های موجود در فایل را پیدا کنید. این سرستون‌ها می‌توانند برچسب‌هایی در نظر گرفته شوند که به عنوان شاخصی برای استخراج مقادیر توسط ستون عمل می‌کنند:

```
>>> data2['id']
array([ 1.,  2.,  3.])
```

در عوض، با استفاده از شاخص‌های عددی به روش کلاسیک شما داده‌های مربوط به ردیف‌ها را استخراج خواهید کرد.

```
>>> data2[0]
(1.0, 123.0, 1.4, 23.0)
```

نتیجه

در این بخش، شما تمام جنبه‌های اصلی کتابخانه NumPy را دیدید و از طریق یک سری مثال‌ها با برخی از ویژگی‌هایی که پایهٔ بسیاری از جنبه‌های دیگر کتاب را تشکیل می‌دهند آشنا شدید. در واقع، بسیاری از این مفاهیم از دیگر کتابخانه‌های علمی و محاسباتی گرفته خواهد شد که تخصصی‌تر هستند، اما براساس این کتابخانه ساخته و توسعه داده شده‌اند. دیدید که چگونه به لطف ndarray می‌توانید قابلیت‌های پایتون را گسترش دهید و آن را به یک زبان مناسب برای محاسبات علمی و به روشی خاص برای تجزیه و تحلیل داده‌ها تبدیل کنید.

بنابراین دانستن NumPy برای هر کسی که می‌خواهد در دنیای تجزیه و تحلیل داده‌ها کاوش کند، حیاتی است.

در فصل بعد، شروع به معرفی یک کتابخانه جدید خواهیم کرد، pandas، که براساس NumPy ساختاربندی شده‌است.

فصل چهار: کتابخانه pandas

با این بخش می‌توانید در نهایت به قلب این کتاب برسید: کتابخانه pandas. این کتابخانه فوق‌العاده پایتون یک ابزار کامل برای هر کسی است که می‌خواهد تجزیه و تحلیل داده‌ها را با استفاده از پایتون به عنوان یک زبان برنامه‌نویسی تمرین کند. ابتدا جنبه‌های اساسی این کتابخانه و نحوه نصب آن را بر روی سیستم خود پیدا خواهید کرد و در نهایت با دو ساختار داده به نام series و DataFrame آشنا خواهید شد. در این بخش با یک مجموعه ابتدایی از توابع که توسط کتابخانه pandas فراهم شده‌است کار می‌کنید تا کارهای رایج در پردازش داده را انجام دهید. آشنایی با این عملیات یک مساله کلیدی خواهد بود. به همین دلیل است که تکرار این فصل برای شما بسیار مهم است تا زمانی که با تمام محتوای آن آشنا شوید.

علاوه بر این، با یک سری از مثال‌ها مفاهیم جدیدی را یاد می‌گیرید که توسط کتابخانه pandas معرفی شده‌اند: نمایه‌سازی ساختارهای داده‌ای آن.

Pandas: کتابخانه تجزیه و تحلیل داده‌ها در پایتون

pandas یک کتابخانه عمومی - خصوصی برای تحلیل داده‌های بسیار تخصصی است. در حال حاضر این نقطه مرجع است که تمام متخصصان با استفاده از زبان پایتون نیاز به مطالعه و تحلیل مجموعه داده‌ها برای اهداف آماری تحلیل و تصمیم‌گیری دارند. این کتابخانه از سال ۲۰۰۸ توسط وس مک کینی طراحی و توسعه یافت. بعدا در سال ۲۰۱۲، سین چانگ، یکی از همکارانش، به این توسعه اضافه شد. آن‌ها با هم یکی از پراستفاده ترین کتابخانه‌های جامعه پایتون را ایجاد کردند.

pandas از نیاز به داشتن یک کتابخانه خاص برای تجزیه و تحلیل داده‌ها ناشی می‌شود که در ساده‌ترین حالت ممکن تمام ابزارهای پردازش داده‌ها، استخراج داده‌ها و دستکاری داده‌ها را فراهم می‌کند.

این پکیج پایتون براساس کتابخانه NumPy طراحی شده‌است. می‌توان گفت که این انتخاب برای موفقیت و گسترش سریع pandas حیاتی بود.

در واقع، این انتخاب نه تنها این کتابخانه را با بیشتر ماژول‌های دیگر سازگار می‌سازد، بلکه از کیفیت بالای عملکرد در محاسبه ماژول NumPy استفاده می‌کند. انتخاب اساسی دیگر طراحی ساختارهای داده

ویژه برای تحلیل داده‌ها بوده‌است. در واقع، به جای استفاده از ساختارهای داده موجود ساخته‌شده در پایتون یا توسط کتابخانه‌های دیگر، دو ساختار داده جدید ایجاد شده‌است. این ساختارهای داده برای کار با داده‌های رابطه‌ای یا برچسب گذاری شده طراحی شده‌اند، بنابراین به شما اجازه می‌دهند تا داده‌ها را با ویژگی‌های مشابه آنچه برای پایگاه‌های داده رابطه‌ای SQL و صفحات گسترده اکسل طراحی شده‌اند، مدیریت کنید. در سراسر این کتاب، در واقع، ما یک سری عملیات اساسی برای تحلیل داده‌ها خواهیم دید که به طور معمول بر روی جداول پایگاه‌داده یا صفحات گسترده استفاده می‌شوند. در واقع pandas مجموعه گسترده‌ای از توابع و روش‌هایی را ارایه می‌دهد که به شما اجازه اجرا می‌دهند و در بسیاری از موارد حتی به بهترین شکل، این عملیات را انجام می‌دهند. در واقع pandas مجموعه گسترده‌ای از توابع و روش‌هایی را ارایه می‌دهد که به شما اجازه اجرا می‌دهند و در بسیاری از موارد حتی به بهترین شکل، به این عملیات می‌پردازند. بنابراین هدف اصلی pandas فراهم کردن تمام بلوک‌های ساختمانی برای هر کسی که به دنیای تجزیه و تحلیل داده‌ها نزدیک می‌شود، می‌باشد.

نصب

ساده‌ترین و عمومی‌ترین راه برای نصب کتابخانه پاندا استفاده از یک راه‌حل از پیش بسته‌بندی‌شده، یعنی نصب آن از طریق توزیع آناکوندا یا Enthought است.

نصب از آناکوندا

برای کسانی که استفاده از توزیع آناکوندا را انتخاب کرده‌اند، مدیریت نصب بسیار ساده است. ابتدا باید ببینیم که آیا ماژول pandas نصب شده‌است و کدام نسخه. برای انجام این کار، فرمان زیر را در پایانه تایپ کنید:

command. conda list pandas

از آنجایی که من در حال حاضر ماژول نصب‌شده روی کامپیوتر شخصی (ویندوز) را دارم، نتیجه زیر را به دست می‌آورم:

```
# packages in environment at C:\Users\Fabio\Anaconda:
#
pandas                    0.14.1                   np19py27_0
```

اگر در مورد شما این طور نیست، باید ماژول pandas را نصب کنید.

```
conda install pandas
```

آناکوندا فورا همه وابستگی‌ها را بررسی می‌کند، نصب ماژول‌های دیگر را مدیریت می‌کند، بدون این که شما نگران باشید. اگر می‌خواهید پکیج خود را به نسخه جدیدتری ارتقا دهید، این فرمان بسیار ساده را به کار ببرید:

conda update pandas

این سیستم نسخه pandas و نسخه تمام ماژول‌های وابسته به آن را بررسی کرده و هر گونه به روز رسانی را پیشنهاد می‌کند و سپس از شما می‌پرسد که آیا می‌خواهید ادامه دهید یا نه.

```
Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment C:\Users\Fabio\Anaconda:
```

پکیج‌های زیر دانلود خواهند شد:

```
package                    |            build
---------------------------|-----------------------------------
pytz-2014.9                |           py27_0         169 KB
requests-2.5.3             |           py27_0         588 KB
six-1.9.0                  |           py27_0          16 KB
conda-3.9.1                |           py27_0         206 KB
pip-6.0.8                  |           py27_0         1.6 MB
scipy-0.15.1               |       np19py27_0        71.3 MB
pandas-0.15.2              |       np19py27_0         4.3 MB
------------------------------------------------------------
                                       Total:        78.1 MB
```

پکیج‌های زیر به روز رسانی خواهند شد:

```
  conda:    3.9.0-py27_0      --> 3.9.1-py27_0
  pandas:   0.14.1-np19py27_0 --> 0.15.2-np19py27_0
  pip:      1.5.6-py27_0      --> 6.0.8-py27_0
  pytz:     2014.7-py27_0     --> 2014.9-py27_0
  requests: 2.5.1-py27_0      --> 2.5.3-py27_0
  scipy:    0.14.0-np19py27_0 --> 0.15.1-np19py27_0
  six:      1.8.0-py27_0      --> 1.9.0-py27_0

  Proceed ([y]/n)?
```

بعد از فشار دادن y، Anaconda شروع به دانلود تمام ماژول‌های به روز شده از شبکه خواهد کرد. بنابراین وقتی این مرحله را انجام می‌دهید لازم است که کامپیوتر به شبکه متصل شود.

```
Fetching packages ...
scipy-0.15.1-n 100% |##############################| Time: 0:01:11   1.05 MB/s
Extracting packages ...
[      COMPLETE      ] |##################################################| 100%
Unlinking packages ...
[      COMPLETE      ] |##################################################| 100%
Linking packages ...
[      COMPLETE      ] |##################################################| 100%
```

نصب از PyPI

Pandas می‌تواند با PyPI نیز نصب گردد:

```
pip install pandas
```

نصب روی لینوکس

اگر روی یک توزیع لینوکس کار می‌کنید و انتخاب می‌کنید که از هیچ کدام از این توزیع‌های از پیش بسته‌بندی‌شده استفاده نکنید، می‌توانید pandas را مانند هر پکیج دیگری نصب کنید.

```
On Debian and Ubuntu distributions:

sudo apt-get install python-pandas
```

روی OpenSuse و Fedora فرمان زیر را وارد کنید:

```
zypper in python-pandas
```

نصب از منبع

اگر می‌خواهید ماژول pandas را با شروع از کد منبع جمع‌آوری کنید، می‌توانید آنچه را که نیاز دارید در گیت‌هاب به لینکِ http://github.com/pydata /pandas پیدا کنید.

```
git clone git://github.com/pydata/pandas.git
cd pandas
python setup.py install
```

اطمینان حاصل کنید که Cython را در زمان کامپایل نصب کرده‌اید.

برای اطلاعات بیشتر لطفاً مستندات موجود در وب، از جمله صفحه رسمی را بخوانید.

(http://pandas.pydata.org/pandasdocs/stable/install.html)

یک مخزن ماژول برای ویندوز

برای کسانی که بر روی ویندوز کار می‌کنند و ترجیح می‌دهند پکیج‌های خود را مدیریت کنند تا همیشه بیش‌ترین ماژول‌های به‌روز را داشته باشند، همچنین منبعی در اینترنت وجود دارد که در آن شما می‌توانید بسیاری از ماژول‌های شخص ثالث را دانلود کنید: مخزن پکیج‌های افزونه پایتون برای ویندوز (متعلق به Christoph Gohlke). هر ماژول با فرمت WHL (wheel) در هر دو شکل ۳۲ بیت و ۶۴ بیتی عرضه می‌شود. برای نصب هر ماژول باید از برنامه pip استفاده کنید.

```
pip install SomePackege-1.0.whl
```

در انتخاب ماژول، دقت کنید که نسخه صحیح را مطابق با نسخه پایتونی که روی آن کار می‌کنید، انتخاب کنید. علاوه بر این، در حالی که NumPy نیازی به نصب بسته‌های دیگر ندارد، برعکس، pandas وابستگی‌های زیادی دارد. ترتیب نصب مهم نیست. عیب این روش این است که شما باید بسته‌ها را به صورت جداگانه و بدون هیچ مدیر بسته‌ای نصب کنید که به شما در مدیریت نسخه‌نویسی و وابستگی بین بسته‌های مختلف کمک کند. مزیت آن تسلط بیشتر بر ماژول‌ها و نسخه‌های آن‌ها است، بنابراین شما بدون توجه به انتخاب‌های توزیع‌ها مانند آناکوندا، دارای بیش‌ترین ماژول‌های فعلی ممکن هستید.

نصب pandas را تست کنید

کتابخانه pandas همچنین توانایی اجرا را پس از نصب آزمایشی برای بررسی قابلیت اجرای کنترل‌های داخلی خود فراهم می‌کند (مستندات رسمی بیان می‌کنند که این تست ۹۷ ٪ از کل کدهای داخلی را پوشش می‌دهد). ابتدا مطمئن شوید که ماژول nose را در توزیع پایتون نصب کرده‌اید (نوار کناری Nose را ببینید). اگر این کار را انجام دهید، می‌توانید تست را با وارد کردن دستور زیر آغاز کنید:

```
nosetests pandas
```

این تست چند دقیقه طول می‌کشد تا کار خود را انجام دهد و در پایان لیستی از مشکلات مواجه شده را نشان می‌دهد.

◀ این ماژول برای تست کد پایتون در طول مراحل ساخت یک پروژه یا به طور خاص یک ماژول پایتون طراحی شده‌است. این ماژول قابلیت‌های ماژول unittest را گسترش می‌دهد: ماژول پایتون درگیر آزمایش کد می‌شود، اما کدگذاری آن را بسیار ساده‌تر و ساده‌تر می‌کند. پیشنهاد می‌شود که این گفتار را مطالعه فرمایید: (http://pythontesting.net/framework/nose/noseintroduction/)

شروع کار با pandas

با توجه به ماهیت موضوع مورد بحث در این فصل، با تمرکز بر توضیح ساختار داده‌ها و توابع / روش‌های به کار رفته در آن، نیازی به نوشتن لیست‌های بزرگ یا خطوط نیست. بنابراین رویکردی که من برای این فصل احساس کردم باز کردن یک پوسته (Shell) پایتون و تایپ کردن یک به یک دستورها است. به این ترتیب، خواننده این فرصت را دارد تا با کارکردهای فردی و ساختارهای داده آشنا شود که به تدریج در این فصل توضیح داده شد.

علاوه بر این، داده‌ها و توابع تعریف‌شده در مثال‌های مختلف در موارد زیر معتبر باقی می‌مانند، در نتیجه خواننده باید هر زمان را تعریف کند. در پایان هر مثال از خواننده دعوت می‌شود تا فرمان‌های مختلف را تکرار کند، آن‌ها را در صورت لزوم اصلاح کند و کنترل کند که چگونه مقادیر درون ساختارهای داده در طول عملیات تغییر می‌کنند. این رویکرد برای آشنا شدن با موضوعات مختلف مطرح‌شده در این فصل عالی است و به خواننده این فرصت را می‌دهد تا آزادانه با مطالب تعامل داشته باشد و در نهایت کار تنها با فرایند خودکار نوشتن و اجرا به پایان نرسد.

توجه: در این بخش به این نکته توجه کنید که فرض می‌شود شما به طور کلی با پایتون و اعداد آشنا هستید. اگر مشکلی دارید می‌توانید فصل‌های ۲ و ۳ این کتاب را بخوانید.

اول، نشستی (Session) در پوسته (Shell) پایتون باز کنید و سپس کتابخانه pandas را وارد کنید. روش کلی برای وارد کردن ماژول pandas به شرح زیر است:

```
>>> import pandas as pd
>>> import numpy as np
```

بنابراین، در این فصل هر بار که pd و np را می‌بینید، به یک شی یا روشِ مراجعه به این دو کتابخانه اشاره خواهید کرد، حتی اگر اغلب برای وارد کردن ماژول pandas به این روش وسوسه شوید:

```
>>> from pandas import *
```

بنابراین، دیگر نیازی به ارجاع تابع، شی یا روش با pd ندارید. این روش به طور کلی توسط جامعه پایتون به عنوان یک روش خوب در نظر گرفته نمی‌شود.

مقدمه‌ای بر ساختارهای داده pandas

قلب pandas تنها دو ساختار داده اصلی است که تمام تراکنش ها، که به طور کلی در طول تجزیه و تحلیل داده‌ها ایجاد می‌شوند، بر روی آن‌ها متمرکز شده‌اند: series و DataFrame.

series، همانطور که خواهید دید، ساختار داده طراحی‌شده برای تطبیق توالی داده‌های یک بعدی را تشکیل می‌دهد، در حالی که فریم دیتابیس، یک ساختار داده پیچیده‌تر، برای شامل کردن مواردی با ابعاد مختلف طراحی شده‌است. اگر چه این ساختارهای داده راه‌حل جهانی برای همه مشکلات نیستند، اما یک ابزار معتبر و قوی را برای بیشتر کاربردها فراهم می‌کنند. در واقع، با سادگی‌شان، درک و استفاده از آن‌ها بسیار ساده است. علاوه بر این، بسیاری از ساختارهای داده پیچیده‌تر را می توان در این دو مورد ساده ردیابی کرد. با این حال، ویژگی‌های آن‌ها براساس یک ویژگی خاص، ادغام در ساختار اشیا شاخص و برچسب‌ها می‌باشد. خواهید دید که این فاکتور منجر به قابلیت دستکاری بالای این ساختارهای داده خواهد شد.

Series

series هدف کتابخانه pandas است که برای نمایش ساختارهای داده یک بعدی، مشابه آرایه اما با برخی ویژگی‌های اضافی طراحی شده‌است. ساختار داخلی آن ساده است (شکل را ببینید) و از دو آرایه مرتبط با یکدیگر تشکیل شده‌است. آرایه اصلی دارای این هدف است که داده‌ها (هر نوع داده‌های NumPy) را نگه دارد که در آن هر عنصر با یک برچسب مرتبط است، که درون آرایه دیگر قرار دارد و شاخص نامیده می‌شود.

Series	
index	value
0	12
1	-4
2	7
3	9

اعلام یک series

برای ایجاد Series همانطور که در شکل مشخص شد، سازندهٔ ()Series را به عنوان آرگومان یک آرایه شامل مقادیری که باید در آن گنجانده شود، فراخوانی کنید.

```
>>> s = pd.Series([12,-4,7,9])
>>> s
0    12
1    -4
2     7
3     9
dtype: int64
```

همان‌طور که می‌توانید از خروجی Series ببینید، در سمت چپ مقادیر در نمایه (index) وجود دارند، که یک Series از برچسب است و در سمت راست مقادیر متناظر.

اگر هیچ شاخصی را در طول تعریف سری مشخص نکنید، به طور پیش‌فرض pandas مقادیر عددی رو به افزایش را از صفر به عنوان برچسب اختصاص خواهد داد. در این حالت برچسب‌ها متناظر با شاخص‌های (موقعیت در آرایه) عناصر درون شی Series هستند. با این حال، اغلب ایجاد یک Series با استفاده از برچسب‌های معنی‌دار به منظور تشخیص و شناسایی هر یک از اقلام صرف‌نظر از ترتیبی که در Series وارد می‌شوند، ترجیح داده می‌شود. بنابراین در این مورد، در طول فراخوانیِ سازنده، لازم است که گزینه شاخص را که مجموعه‌ای از رشته‌ها حاوی برچسب‌ها را تعیین می‌کند، در نظر بگیریم.

```
>>> s = pd.Series([12,-4,7,9], index=['a','b','c','d'])
>>> s
a    12
b    -4
c     7
d     9
dtype: int64
```

اگر می‌خواهید به صورت جداگانه دو آرایه را ببینید که این ساختار داده را تشکیل می‌دهند، می‌توانید دو ویژگی Series را به این صورت نامگذاری کنید: شاخص و مقادیر.

```
>>> s.values
array([12, -4,  7,  9], dtype=int64)
>>> s.index
Index([u'a', u'b', u'c', u'd'], dtype='object')
```

انتخاب (select) کردن عناصر درونی

در مورد عناصر جداگانه، می توانید آن‌ها را به عنوان یک آرایه معمولی NumPy، با مشخص کردن کلید انتخاب کنید.

```
>>> s[2]
7
```

یا می‌توانید برچسب مربوط به موقعیت شاخص را مشخص کنید.

```
>>> s['b']
-4
```

به همین ترتیب شما می‌توانید چند مورد را در یک آرایه عددی انتخاب کنید و موارد زیر را مشخص کنید:

```
>>> s[0:2]
a    12
b    -4
dtype: int64
```

یا حتی در این حالت، از برچسب‌های متناظر استفاده کنید، اما لیست برچسب‌ها در یک آرایه را مشخص کنید.

```
>>> s[['b','c']]
b   -4
c    7
dtype: int64
```

تخصیص مقادیر به عناصر

حالا که درک می‌کنید چگونه عنصرهای فردی را انتخاب کنید، همچنین می‌دانید که چگونه مقادیر جدیدی را به آن‌ها اختصاص دهید. در واقع، شما می‌توانید مقدار را با شاخص یا برچسب انتخاب کنید.

```
>>> s[1] = 0
>>> s
a    12
b     0
c     7
d     9
dtype: int64
>>> s['b'] = 1
>>> s
a    12
b     1
c     7
d     9
dtype: int64
```

تعریف Series از آرایه‌های NumPy و سری‌های دیگر

شما می‌توانید سری جدیدی را تعریف کنید که با آرایه‌های نو یا سری‌های موجود شروع می‌شود.

```
>>> arr = np.array([1,2,3,4])
>>> s3 = pd.Series(arr)
>>> s3
0    1
1    2
2    3
3    4
dtype: int32

>>> s4 = pd.Series(s)
>>> s4
a    12
b     4
c     7
d     9
dtype: int64
```

با این حال، هنگام انجام این کار، همیشه باید به خاطر داشته باشید که مقادیر موجود در آرایه NumPy یا سری اصلی کپی نمی‌شوند، بلکه توسط مرجع منتقل می‌شوند. به این معنی که شی به صورت پویا درون شی سری جدید وارد می‌شود. اگر تغییر کند، برای مثال عنصر داخلی آن در مقدار تغییر کند، آن تغییرات در شی سری جدید نیز وجود خواهد داشت.

```
>>> s3
0    1
1    2
2    3
3    4
dtype: int32

>>> arr[2] = -2
>>> s3
0    1
1    2
2   -2
3    4
dtype: int32
```

همانطور که در این مثال می‌بینیم، با تغییر عنصر سوم آرایه arr، عنصر مربوطه در سری s۳ را نیز اصلاح کردیم.

فیلتر کردن مقادیر

به لطف انتخاب کتابخانه NumPy به عنوان پایه توسعه کتابخانه پاندا و در نتیجه ساختارهای داده‌ای آن،

بسیاری از عملیات قابل‌اجرا برای آرایه‌های NumPy به این سری بسط داده می‌شوند. یکی از این موارد، فیلتر کردن مقادیر موجود در ساختار داده‌ها از طریق شرایط است. برای مثال، اگر بخواهید بدانید کدام یک از عناصر درون مجموعه مقدار بیش از ۸ دارند، موارد زیر را می‌نویسید:

```
>>> s[s > 8]
a    12
d     9
dtype: int64
```

عملیات و توابع ریاضی

عملیات دیگر مانند اوپراتورها (+، -، ×، /) یا توابع ریاضی که برای آرایه NumPy قابل‌اجرا هستند را می‌توان به سری اشیا بسط داد. در مورد عملگرها می‌توانید به سادگی عبارت حسابی را بنویسید.

```
>>> s / 2
a    6.0
b   -2.0
c    3.5
d    4.5
dtype: float64
```

با این حال، با توجه به توابع ریاضی NumPy، شما باید تابع ارجاع شده به np و نمونه سری به عنوان آرگومان را مشخص کنید.

```
>>> np.log(s)
a    2.484907
b         NaN
c    1.945910
d    2.197225
dtype: float64
```

ارزیابی مقادیر

اغلب در یک سری ممکن است مقادیر تکراری وجود داشته باشد و پس از آن ممکن است لازم باشد اطلاعاتی در مورد این‌که نمونه‌ها شامل چیست، تکراری‌ها را بشمارید و اینکه آیا یک مقدار در سری وجود دارد یا خیر.

در این رابطه، یک سری را اعلام کنید که در آن مقادیر تکراری زیادی وجود دارد.

```
>>> serd = pd.Series([1,0,2,1,2,3], index=['white','white','blue','green','green','yellow'])
>>> serd
white     1
white     0
blue      2
green     1
green     2
yellow    3
dtype: int64
```

برای دانستن تمام مقادیر موجود در سری به‌جز تکراری‌ها، می‌توانید از تابع ( )unique استفاده کنید. مقدار بازگشتی آرایه‌ای است که شامل مقادیر منحصر به فرد در سری است اما لزوماً مرتب نیست.

```
>>> serd.unique()
array([1, 0, 2, 3], dtype=int64)
```

یک تابع ( )unique مشابه تابع ( )value_counts است که نه تنها مقادیر منحصر به فرد را باز می‌گرداند بلکه پیشامدها را در یک سری محاسبه می‌کند.

```
>>> serd.value_counts()
2    2
1    2
3    1
0    1
dtype: int64
```

در نهایت، ()isin یک تابع است که عضویت را ارزیابی می‌کند، به این معنی که با داشتن فهرستی از مقادیر، این تابع به شما اجازه می‌دهد که بدانید آیا این مقادیر در ساختار داده‌ها وجود دارند یا خیر. مقادیر بولی که بازگردانده می‌شوند می‌توانند در طول فیلترینگ داده‌ها در یک سری یا در یک ستون از DataFrame بسیار مفید باشند.

```
>>> serd.isin([0,3])
white     False
white      True
blue      False
green     False
green     False
yellow     True
dtype: bool
>>> serd[serd.isin([0,3])]
white     0
yellow    3
dtype: int64
```

مقادیر NaN

همان طور که در مورد قبلی می‌بینید ما سعی کردیم لگاریتم یک عدد منفی را اجرا کنیم و در نتیجه آن را دریافت کردیم. این مقدار خاص NaN (نه یک عدد) در ساختار داده pandas برای نشان دادن حضور یک میدان خالی استفاده می‌شود یا به صورت عددی قابل تعریف نیست. به طور کلی، این مقادیر یک مشکل هستند و باید به طریقی مدیریت شوند، به خصوص در طول تجزیه و تحلیل داده‌ها. این داده‌ها به ویژه زمانی تولید می‌شوند که استخراج داده‌ها از یک منبع مشکل ایجاد می‌کند، یا حتی زمانی که منبع یک داده گم‌شده است. علاوه بر این همانطور که قبلا هم مشاهده کردید، مقادیر NaN را می توان در موارد خاصی مانند محاسبه لگاریتم مقادیر منفی یا استثنا در طول اجرای برخی محاسبات یا توابع نیز تولید کرد.

با این حال، pandas علی‌رغم ماهیت مساله‌دار خود، به طور واضح این مقدار را در یک ساختار داده، مانند سری، تعریف و اضافه می‌کند. هرگاه بخواهیم یک مقدار گم‌شده را تعریف کنیم داخل آرایه شامل مقادیر np.NaN را وارد می‌کنیم.

```
>>> s2 = pd.Series([5,-3,np.NaN,14])
>>> s2
0     5
1    -3
2   NaN
3    14
```

توابع isnull() و notnull() برای شناسایی شاخص‌ها بدون مقدار بسیار مفید هستند.

```
>>> s2.isnull()
0    False
1    False
2     True
3    False
dtype: bool
>>> s2.notnull()
0     True
1     True
2    False
3     True
dtype: bool
```

در واقع، این توابع دو سری را با مقادیر بولی برمی گردانند که شامل مقادیر «درست» و «غلط» بسته به اینکه آیتم یک مقدار NaN باشد یا کم‌تر، است. تابع ()isnull در سری مقادیر "درست" را باز می‌گرداند؛ برعکس، ()notnull عدد صحیح را اگر NaN نباشد باز می‌گرداند. این توابع برای قرار گرفتن در داخل فیلترینگ برای ایجاد یک شرط مفید هستند.

```
>>> s2[s2.notnull()]
0     5
1    -3
3    14
dtype: float64
>>> s2[s2.isnull()]
2   NaN
dtype: float64
```

سری‌ها به عنوان دیکشنری‌ها

یک راه دیگر برای دیدن یک سری، این است که آن را به عنوان یک شی dic (دیکشنری) در نظر بگیریم. این شباهت همچنین در طول تعریف یک سری اشیا مورد استفاده قرار می‌گیرد. در اصل، شما می‌توانید یک سری از یک فرمان که قبلاً تعریف شده بود را ایجاد کنید.

```
>>> mydict = {'red': 2000, 'blue': 1000, 'yellow': 500, 'orange': 1000}
>>> myseries = pd.Series(mydict)
blue      1000
orange    1000
red       2000
yellow     500
dtype: int64
```

همانطور که از این مثال می‌بینید، ردیف شاخص با مقادیر کلیدها پر می‌شود در حالی که داده با مقادیر متناظر پر می‌شود. همچنین می‌توانید شاخص‌های آرایه را به صورت جداگانه تعریف کنید. در این مورد، کنترل تناظر بین کلیدهای فرمان و برچسب‌های ردیف شاخص‌ها اجرا خواهد شد. در صورت عدم تطابق، pandas مقدار NaN را اضافه خواهد کرد.

```
>>> colors = ['red','yellow','orange','blue','green']
>>> myseries = pd.Series(mydict, index=colors)
red       2000
yellow     500
orange    1000
blue      1000
green      NaN
dtype: float64
```

عملیات‌ها میان Series

مشاهده کردیم که چگونه عملیات ریاضی بین سری و مقادیر اسکالر انجام دهیم. همین امر با اجرای عملیات بین دو سری ممکن است، اما در این مورد حتی برچسب‌ها نیز وارد عمل می‌شوند. در واقع یکی از پتانسیل‌های بزرگ این نوع از ساختار داده‌ها، توانایی سری‌ها برای هم تراز کردن داده‌های متفاوت بین آن‌ها از طریق شناسایی برچسب مربوطه است. در مثال زیر شما مجموع دو سری را دارید که تنها برخی عناصر مشترک با برچسب دارند.

```
>>> mydict2 = {'red':400,'yellow':1000,'black':700}
>>> myseries2 = pd.Series(mydict2)
>>> myseries + myseries2
black      NaN
blue       NaN
orange     NaN
green      NaN
red       2400
yellow    1500
dtype: float64
```

شما یک سری شی جدید به دست می‌آورید که در آن فقط آیتم‌هایی با یک برچسب اضافه می‌شوند. همه برچسبی که در یکی از این دو سری وجود دارد هنوز هم به نتیجه اضافه می‌شوند اما دارای مقدار NaN هستند.

DataFrame

DataFrame یک ساختار داده جدولی بسیار شبیه به صفحه گسترده است (آشناترین آن‌ها صفحات گسترده اکسل هستند). این ساختار داده برای گسترش حالت سری به ابعاد چندگانه طراحی شده‌است. در واقع، DataFrame متشکل از یک مجموعه مرتب از ستون‌ها است که هر کدام می‌تواند حاوی مقدار انواع مختلف باشد (عددی، رشته‌ای، بولی، و ...).

DataFrame			
	columns		
index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

بر خلاف سری که دارای آرایه **شاخص** حاوی برچسب مربوط به هر عنصر است، در مورد DataFrame، دو آرایه **شاخص** وجود دارد. اولی، مرتبط با خطوط، دارای توابع بسیار مشابهی با آرایه شاخص در سری است. در واقع، هر برچسب با تمام مقادیر موجود در ردیف مرتبط است. آرایه دوم به جای آن شامل یک سری برچسب است که هر کدام مرتبط با یک ستون خاص هستند. یک DataFrame همچنین می‌تواند به عنوان یک فرمان سری درک شود، که در آن کلیدها نام‌های ستون و مقادیر، سری هستند که ستون‌های DataFrame را تشکیل می‌دهند. علاوه بر این، تمام عناصر هر سری با توجه به آرایه‌ای از برچسب‌ها به نام نمایه (index) طراحی می‌شوند.

تعریف یک DataFrame

رایج‌ترین روش برای ایجاد یک DataFrame جدید، انتقال دقیق یک شی به سازنده DataFrame() است. این شیِ دیکشنری شامل کلیدی برای هر ستون است که ما می‌خواهیم آن را تعریف کنیم.

```
>>> data = {'color' : ['blue','green','yellow','red','white'],
                      'object' : ['ball','pen','pencil','paper','mug'],
                      'price' : [1.2,1.0,0.6,0.9,1.7]}

frame = pd.DataFrame(data)
>>> frame
    color  object price
0    blue    ball   1.2
1   green     pen   1.0
2  yellow  pencil   0.6
3     red   paper   0.9
4   white     mug   1.7
```

اگر شیِ دیکشنری که می‌خواهیم از آن یک DataFrame ایجاد کنیم حاوی داده‌های بیشتری نسبت به آنچه که ما می‌خواهیم باشد، می‌توانید یک انتخاب ایجاد کنید. در سازنده DataFrame()، شما می‌توانید با استفاده از گزینه ستون‌ها توالی ستون‌ها را مشخص کنید. ستون‌ها به ترتیب توالی ایجاد می‌شوند بدون توجه به این که چگونه در شیِ دیکشنری گنجانده شده‌اند.

```
>>> frame2 = pd.DataFrame(data, columns=['object','price'])
>>> frame2
   object price
0    ball   1.2
1     pen   1.0
2  pencil   0.6
3   paper   0.9
4     mug   1.7
```

حتی برای اشیای DataFrame، اگر برچسب‌ها به طور واضح در آرایه نمایه مشخص نشده باشند، pandas به طور خودکار یک توالی عددی را با شروع از ۰ تخصیص می‌دهد. از یک DataFrame، شما باید از گزینه شاخص استفاده کنید و آن را به آرایه‌ای که حاوی برچسب‌ها است اختصاص دهید.

```
>>> frame2 = pd.DataFrame(data, index=['one','two','three','four','five'])
>>> frame2
       color  object  price
one     blue    ball    1.2
two    green     pen    1.0
three  yellow  pencil    0.6
four     red   paper    0.9
five   white     mug    1.7
```

اکنون که ما دو شاخص و ستون گزینه‌های جدید را معرفی کرده‌ایم، تصور یک روش جایگزین برای تعریف یک DataFrame آسان است. به جای استفاده از یک شیِ دیکشنری، شما می‌توانید در داخل سازنده سه آرگومان را به ترتیب زیر تعریف کنید: یک ماتریس داده، سپس یک آرایه شامل برچسب‌های اختصاص‌یافته به گزینهٔ شاخص و در نهایت یک آرایه شامل اسامی ستون‌های اختصاص‌یافته به گزینهٔ ستون. در بسیاری از مثال‌ها برای ایجاد ماتریسی از مقادیر می‌توانید از np.arange(16).reshape((4,4)) استفاده کنید که یک ماتریس چهار در چهار با اعداد افزایشی از صفر تا ۱۵ تولید می‌کند.

```
>>> frame3 = pd.DataFrame(np.arange(16).reshape((4,4)),
...                       index=['red','blue','yellow','white'],
...                       columns=['ball','pen','pencil','paper'])
>>> frame3
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white     12   13      14     15
```

عناصر انتخابی

اول، اگر بخواهیم نام تمام ستون‌های یک DataFrame را بدانیم برای مشخص کردن مشخصه (index) ستون‌ها در نمونه شیِ DataFrame کافی است.

```
>>> frame.columns
Index([u'colors', u'object', u'price'], dtype='object')
```

به طور مشابه، برای به دست آوردن فهرست شاخص‌ها، شما باید صفت (attribute) شاخص (index) را تعیین نمایید.

```
>>> frame.index
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

با توجه به مقادیر موجود در ساختار داده‌ها، شما می‌توانید کل مجموعه داده‌ها را با استفاده از صفت (attribute) **مقادیر** (values) به دست آورید.

```
>>> frame.values
array([['blue', 'ball', 1.2],
       ['green', 'pen', 1.0],
       ['yellow', 'pencil', 3.3],
       ['red', 'paper', 0.9],
       ['white', 'mug', 1.7]], dtype=object)
```

یا اگر می‌خواهید فقط محتویات یک ستون را انتخاب کنید، می‌توانید نام ستون را بنویسید.

```
>>> frame['price']
0    1.2
1    1.0
2    0.6
3    0.9
4    1.7
Name: price, dtype: float64
```

همانطور که می‌بینید، مقدار بازگردانده شده یک شیِ سری است. راه دیگر استفاده از نام ستون به عنوان یک صفت (attribute) از نمونه DataFrame است.

```
>>> frame.price
0    1.2
1    1.0
2    0.6
3    0.9
4    1.7
Name: price, dtype: float64
```

با توجه به ردیف‌های درون یک DataFrame، امکان استفاده از صفت (attribute) ix با مقدار شاخص (index) ردیفی که می‌خواهید استخراج کنید، وجود دارد.

```
>>> frame.ix[2]
color     yellow
object    pencil
price        0.6
Name: 2, dtype: object
```

شی بازگشتی یک سری است که در آن نام ستون‌ها به برچسب شاخص آرایه تبدیل‌شده در حالی که مقادیر به داده‌های سری تبدیل شده‌اند. برای انتخاب چندین ردیف شما آرایه‌ای را با توالی ردیف‌ها برای درج مشخص می‌کنید:

```
>>> frame.ix[[2,4]]
    color  object  price
2  yellow  pencil    0.6
4   white     mug    1.7
```

اگر نیاز به استخراج بخشی از یک DataFrame دارید، با انتخاب خطوطی که می‌خواهید استخراج کنید، می‌توانید از اعداد مرجع شاخص‌ها استفاده کنید. در واقع شما می‌توانید یک ردیف را به عنوان بخشی از یک چارچوب داده در نظر بگیرید که شاخص ردیف به عنوان مقدار مبدا (در ۰ بعدی) و خط بالای آن به عنوان مقدار دوم (در مقدار بعدی) باشد.

```
>>> frame[0:1]
  color object  price
0  blue   ball    1.2
```

همانطور که می‌بینید، مقدار بازگشت یک شیِ DataFrame حاوی یک ردیف واحد است. اگر بیش از یک خط می‌خواهید، باید محدوده انتخاب را گسترش دهید.

```
>>> frame[1:3]
    color  object  price
1   green     pen    1.0
2  yellow  pencil    0.6
```

در نهایت، اگر چیزی که می‌خواهید به دست آورید یک مقدار واحد در یک DataFrame باشد، ابتدا از نام ستون و سپس شاخص یا برچسب سطر استفاده می‌کنید.

```
>>> frame['object'][3]
'paper'
```

تخصیص مقادیر

هنگامی که درک کردید چگونه به عناصر مختلفی که یک DataFrame را تشکیل می‌دهند، دسترسی پیدا کنید، تنها از منطق مشابهی برای اضافه کردن یا تغییر مقادیر در آن استفاده کنید. به عنوان مثال، شما قبلاً دیدید که در ساختار DataFrame یک آرایه از شاخص‌ها با صفتِ index تعیین می‌شود و ردیف حاوی نام ستون‌ها با ویژگی ستون مشخص می‌شود.

خوب، شما همچنین می‌توانید با استفاده از صفتِ name یک برچسب به این دو زیر ساختار برای شناسایی آن‌ها اختصاص دهید.

```
>>> frame.index.name = 'id'; frame.columns.name = 'item'
>>> frame
item   color  object  price
id
0       blue    ball    1.2
1      green     pen    1.0
2     yellow  pencil    3.3
3        red   paper    0.9
4      white     mug    1.7
```

یکی از بهترین ویژگی‌های ساختار داده pandas انعطاف‌پذیری بالای آن‌هاست. در واقع شما همیشه می‌توانید در هر سطحی برای تغییر ساختار داده‌های داخلی مداخله کنید. برای مثال، یک عملیات بسیار معمول اضافه کردن یک ستون جدید است. شما می‌توانید این کار را تنها با اختصاص یک مقدار به نمونه DataFrame انجام دهید که یک نام ستون جدید را مشخص می‌کند.

```
>>> frame['new'] = 12
>>> frame
   colors  object price  new
0    blue    ball   1.2   12
1   green     pen   1.0   12
2  yellow  pencil   0.6   12
3     red   paper   0.9   12
4   white     mug   1.7   12
```

با این حال، اگر می‌خواهید محتویات یک ستون را به روز رسانی کنید، باید از یک آرایه استفاده کنید.

```
>>> frame['new'] = [3.0,1.3,2.2,0.8,1.1]
>>> frame
    color  object  price  new
0    blue    ball    1.2  3.0
1   green     pen    1.0  1.3
2  yellow  pencil    0.6  2.2
3     red   paper    0.9  0.8
4   white     mug    1.7  1.1
```

اگر می‌خواهید یک ستون کامل را به روز رسانی کنید، می‌توانید از یک روش مشابه پیروی کنید، برای مثال با استفاده از تابع np.arange() برای به روز رسانی مقادیر یک ستون با توالی از پیش تعیین‌شده.

```
>>> ser = pd.Series(np.arange(5))
>>> ser
0    0
1    1
2    2
3    3
4    4
dtype: int32
>>> frame['new'] = ser
>>> frame
    color  object  price  new
0    blue    ball    1.2    0
1   green     pen    1.0    1
2  yellow  pencil    0.6    2
3     red   paper    0.9    3
4   white     mug    1.7    4
```

در نهایت، برای تغییر یک مقدار واحد، به سادگی آیتم را انتخاب کرده و مقدار جدید را به آن بدهید.

```
>>> frame['price'][2] = 3.3
```

عضویت یک مقدار (value)

قبلاً تابع ()isin را در سری اعمال کرده‌اید تا به عضویت مجموعه‌ای از مقادیر بپردازید. خوب، این ویژگی همچنین در مورد اهداف DataFrame قابل‌اجرا است.

```
>>> frame.isin([1.0,'pen'])
   color object  price
0  False  False  False
1  False   True   True
2  False  False  False
3  False  False  False
4  False  False  False
```

یک DataFrame را دریافت می‌کنید که تنها شامل مقادیر بولی است، که در آن درست تنها مقادیری را دارد که با عضویت مطابقت دارند. اگر مقدار را به عنوان یک شرط برگشت دهید، یک DataFrame جدید خواهید داشت که تنها حاوی مقادیری است که شرط را برآورده می‌کنند.

```
>>> frame[frame.isin([1.0,'pen'])]
  color object  price
0   NaN    NaN    NaN
1   NaN    pen      1
2   NaN    NaN    NaN
3   NaN    NaN    NaN
4   NaN    NaN    NaN
```

حذف یک ستون

اگر می‌خواهید یک ستون کامل را با تمام محتویات آن حذف کنید، از دستور del استفاده کنید.

```
>>> del frame['new']
>>> frame
   colors  object price
0    blue    ball   1.2
1   green     pen   1.0
2  yellow  pencil   0.6
3     red   paper   0.9
4   white     mug   1.7
```

فیلترینگ (Filtering)

حتی برای یک DataFrame می‌توانید فیلترینگ را از طریق کاربرد شرایط خاص اعمال کنید، برای مثال اگر می‌خواهید همه مقادیر کوچک‌تر از یک عدد خاص، برای مثال ۱۲ را به دست آورید.

```
>>> frame[frame < 12]
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white    NaN  NaN     NaN    NaN
```

شما به عنوان شیِ بازگشتی یک DataFrame حاوی مقادیر کم‌تر از ۱۲، با حفظ موقعیت اصلی آن‌ها دریافت خواهید کرد. همه موارد دیگر با NaN جایگزین خواهند شد.

DataFrame از دیکشنری تو در تو

یک ساختار داده بسیار رایج در پایتون دیکشنری تودرتو است که به صورت زیر نشان داده می‌شود:

```
nestdict = { 'red': { 2012: 22, 2013: 33 },
                    'white': { 2011: 13, 2012: 22; 2013: 16},
                    'blue': {2011: 17, 2012: 27; 2013: 18}}
```

این ساختار داده، زمانی که به طور مستقیم به عنوان یک آرگومان برای سازنده DataFrame رد می‌شود، توسط pandas تفسیر می‌شود تا کلیدهای خارجی را به عنوان نام‌های ستون و کلیدهای داخلی را به عنوان برچسب‌های شاخص‌ها در نظر بگیرند. در طول تفسیر ساختار تودرتو، ممکن است که همه زمینه‌ها تطابق موفقی پیدا نکنند. pandas با اضافه کردن مقدار از دست رفته، این ناهماهنگی را جبران خواهد کرد.

```
>>> nestdict = {'red':{2012: 22, 2013: 33},
...             'white':{2011: 13, 2012: 22, 2013: 16},
...             'blue': {2011: 17, 2012: 27, 2013: 18}}
>>> frame2 = pd.DataFrame(nestdict)
>>> frame2
      blue  red  white
2011    17  NaN     13
2012    27   22     22
2013    18   33     16
```

فراگذاری DataFrame

عملیاتی که ممکن است در هنگام برخورد با ساختارهای داده جدولی مورد نیاز باشد فراگذاری است (یعنی ستون‌ها به سطرها و سطرها به ستون‌ها تبدیل می‌شوند). pandas به شما اجازه می‌دهد این کار را به روشی بسیار ساده انجام دهید. شما می‌توانید با اضافه کردن صفت (attribute) T به برنامه کاربردی آن، فراگذاری داده را به دست آورید.

```
>>> frame2.T
       2011  2012  2013
blue     17    27    18
red     NaN    22    33
white    13    22    16
```

اشیا شاخص (index)

حالا که می‌دانید Series و DataFrame چه هستند و چگونه ساخته شده‌اند، قطعاً می‌توانید ویژگی‌های این ساختارهای داده را درک کنید. در واقع، اکثر ویژگی‌های عالی آن‌ها در تجزیه و تحلیل داده‌ها ناشی از حضور یک شی شاخص است که به طور کامل در این ساختارهای داده ادغام شده‌است.

اشیا شاخص مسئول برچسب‌های روی محورها و متاداده های دیگر به عنوان نام محورها هستند. قبلاً دیدید که یک آرایه حاوی برچسب به یک شی نمایه تبدیل شده‌است: شما باید گزینه شاخص (index) در داخل سازنده (constructor) را مشخص کنید.

```
>>> ser = pd.Series([5,0,3,8,4], index=['red','blue','yellow','white','green'])
>>> ser.index
Index([u'red', u'blue', u'yellow', u'white', u'green'], dtype='object')
```

بر خلاف تمام عناصر دیگر در ساختار داده pandas (Series و DataFrame)، اشیا شاخص اشیا تغییرناپذیر هستند. به محض اعلام، این موارد را نمی توان تغییر داد. این امر اشتراک گذاری ایمن بین ساختارهای مختلف داده را تضمین می‌کند.

هر شیِ شاخص تعدادی روش (متد) و ویژگی (property) دارد به خصوص زمانی که شما نیاز به دانستن مقادیر دارید.

متدها روی شاخص (index)

چند روش خاص برای شاخص‌ها وجود دارد تا اطلاعاتی در مورد شاخص از ساختار داده‌ها به دست آورند. برای مثال، ()idmin و ()idmax دو تابع هستند که به ترتیب باز می‌گردند، شاخص با کم‌ترین و بیشترین مقدار.

```
>>> ser.idxmin()
'red'
>>> ser.idxmax()
'green'
```

نمایه با برچسب‌های تکراری

تا کنون، تمام مواردی را که در آن‌ها شاخص‌ها در یک ساختار داده واحد دارای برچسب منحصر به فرد بودند، مشاهده کرده‌اید. اگرچه بسیاری از توابع برای اجرا به این شرط نیاز دارند، برای ساختار داده‌های pandas این شرط الزامی نیست. به عنوان مثال، یک سری با چند برچسب تکراری تعریف می‌شود.

```
>>> serd = pd.Series(range(6), index=['white','white','blue','green','green','yellow'])
>>> serd
white     0
white     1
blue      2
green     3
green     4
yellow    5
dtype: int64
```

با توجه به انتخاب عناصر در یک ساختار داده، اگر در تطابق با یک برچسب مقادیر بیشتری وجود داشته باشد، شما یک سری به جای یک عنصر واحد به دست خواهید آورد.

```
>>> serd['white']
white    0
white    1
dtype: int64
```

همین منطق برای DataFrame ها با شاخص‌های تکراری که چارچوب داده‌ها را برمی گردانند نیز به کار می‌رود. در مورد ساختارهای داده با اندازه کوچک، شناسایی هر شاخص تکراری آسان است، اما اگر ساختار به تدریج بزرگ‌تر شود، این کار دشوار می‌شود.

درست در این رابطه، pandas ویژگی منحصر به فرد متعلق به اشیا شاخص را برای شما فراهم می‌کند. این ویژگی به شما خواهد گفت که آیا شاخص‌هایی با برچسب‌های تکراری در داخل داده ساختار وجود دارند (هم Series و هم DataFrame).

```
>>> serd.index.is_unique
False
>>> frame.index.is_unique
True
```

سایر کارکردها در نمایه‌ها (indexes)

در مقایسه با ساختارهای داده‌ای که معمولاً با پایتون مورد استفاده قرار می‌گیرند، دیدید که pandas و همچنین استفاده از کیفیت عملکرد بالا که توسط آرایه‌های NumPy ارایه می‌شود، برای ادغام شاخص‌ها در آن‌ها انتخاب شده‌است. این انتخاب تا حدودی موفق به اثبات رسیده‌است. در واقع، با وجود انعطاف‌پذیری بسیار زیاد ارائه‌شده توسط ساختارهای پویایی که در حال حاضر وجود دارند، توانایی استفاده از مرجع داخلی به ساختار، مانند آنچه که توسط برچسب‌ها ارائه شده‌است، به کسانی که باید عملیات را به روشی بسیار ساده‌تر و مستقیم‌تر انجام دهند، اجازه می‌دهد تا یک سری عملیات را انجام دهند.

در این بخش شما تعدادی از ویژگی‌های اساسی را که از این مکانیسم شاخص‌ها بهره می‌برند به طور مفصل تحلیل خواهید کرد.

◄ Reindexing (نمایه‌سازی مجدد)

◄ Dropping

◄ Alignment

Reindexing

قبلاً گفته شد که وقتی محتویات یک ساختار داده اعلام می‌شود، شیِ شاخص (index) نمی‌تواند تغییر کند. درست است، اما با اجرای یک نمایه‌سازی مجدد می‌توانید بر این مشکل نیز غلبه کنید.

در واقع، بدست آوردن یک ساختار داده جدید از یک ساختار داده موجود امکان پذیر است که در آن قوانین نمایه‌سازی می‌تواند دوباره تعریف شود.

```
>>> ser = pd.Series([2,5,7,4], index=['one','two','three','four'])
>>> ser
one      2
two      5
three    7
four     4
dtype: int64
```

به منظور ایجاد نمایه‌سازی مجدد این سری، pandas به شما یک تابع ( )reindex می‌دهد. این تابع یک شیِ سری جدید با مقادیر سری قبلی که با توجه به توالی جدید برچسب‌ها مرتب شده‌اند، ایجاد می‌کند. بنابراین در طول این عملیات نمایه‌سازی مجدد، امکان تغییر ترتیب توالی شاخص‌ها، حذف برخی از آن‌ها یا اضافه کردن شاخص‌های جدید وجود دارد. در مورد یک برچسب جدید، pandas مقدار NaN را به عنوان مقدار متناظر اضافه می‌کند.

```
>>> ser.reindex(['three','four','five','one'])
three     7
four      4
five    NaN
one       2
dtype: float64
```

همانطور که می‌بینید، ترتیب برچسب‌ها به طور کامل دوباره مرتب شده‌است. مقدار مربوط به برچسب «دو» حذف شده‌است و یک برچسب جدید «پنج» در این سری وجود دارد. با این حال، برای اندازه‌گیری نمایه‌سازی مجدد، تعریف لیست تمام برچسب‌ها می‌تواند دشوار باشد، به خصوص برای یک چارچوب داده بزرگ. بنابراین می‌توانید از روشی استفاده کنید که به شما این امکان را می‌دهد که مقادیر را به صورت خودکار پر کرده یا درون یابی کنید. برای درک بهتر عملکرد این حالت از نمایه‌سازی مجدد خودکار، سری زیر را تعریف کنید.

```
>>> ser3 = pd.Series([1,5,6,3],index=[0,3,5,6])
>>> ser3
0    1
3    5
5    6
6    3
dtype: int64
```

همانطور که در این مثال می‌بینید، ستون فهرست دنباله کاملی از اعداد نیست؛ در واقع مقادیر گم‌شده (۱، ۲ و ۴) وجود دارد. یک نیاز معمول، انجام درون یابی به منظور به دست آوردن توالی کامل اعداد است. برای رسیدن به این هدف، از نمایه‌سازی مجدد با تنظیم گزینهٔ متد روی ffill بهره خواهید برد. علاوه بر این، باید محدوده‌ای از مقادیر را برای شاخص‌ها تعیین کنید.

در این حالت، برای مشخص کردن مجموعه‌ای از مقادیر بین ۰ و ۵، می‌توانید از range(6) به عنوان آرگومان استفاده کنید.

```
>>> ser3.reindex(range(6),method='ffill')
0    1
1    1
2    1
3    5
4    5
5    6
dtype: int64
```

همانطور که از نتایج می‌بینید، شاخص‌هایی که در سری اصلی وجود نداشتند اضافه شدند. با درون یابی، آن‌هایی که کم‌ترین شاخص را در سری اصلی دارند به عنوان مقادیر اختصاص داده می‌شوند. در واقع شاخص‌های ۱ و ۲ دارای مقدار ۱ هستند که به شاخص ۰ تعلق دارد. اگر مایلید این مقدار شاخص در طول درج معین گردد باید از متد bfill() استفاده کنید.

```
>>> ser3.reindex(range(6),method='bfill')
0    1
1    5
2    5
3    5
4    6
5    6
dtype: int64
```

در این مورد مقدار اختصاص‌یافته به شاخص‌های ۱ و ۲ مقدار ۵ است که به شاخص ۳ تعلق دارد.

با گسترش مفاهیم نمایه‌سازی مجدد از سری به DataFrame، شما می‌توانید نه تنها برای نمایه‌ها (ردیف‌ها)، بلکه برای ستون‌ها، یا حتی هر دو، بازآرایی داشته باشید. همانطور که قبلا ذکر شد، اضافه کردن یک ستون یا شاخص جدید ممکن است، اما در مورد مقادیر از دست رفته در ساختار داده اصلی، pandas مقادیر NaN را به آن‌ها اضافه می‌کند.

```
>>> frame.reindex(range(5), method='ffill',columns=['colors','price','new','object'])
   colors price  new  object
0    blue   1.2  NaN    ballpand
1   green   1.0  NaN     pen
2  yellow   0.6  NaN  pencil
3     red   0.9  NaN   paper
4   white   1.7  NaN     mug
```

Dropping

عملیات دیگری که به اشیا شاخص مربوط است Dropping است. حذف یک سطر یا ستون، با توجه به برچسب‌های به کار رفته برای نشان دادن شاخص‌ها و نام‌های ستون، ساده می‌شود. همچنین در این مورد، pandas یک تابع خاص برای این عملیات فراهم می‌کند: drop(). این روش یک شیِ جدید را بدون مواردی که می‌خواهید حذف کنید، برمی گرداند. برای مثال حالتی را در نظر بگیرید که می‌خواهیم یک آیتم را از یک سری حذف کنیم. برای انجام این کار، سری‌ای با چهار عنصر و چهار برچسب مشخص تعریف کنید.

```
>>> ser = Series(np.arange(4.), index=['red','blue','yellow','white'])
>>> ser
red       0
blue      1
yellow    2
white     3
dtype: float64
```

حالا، برای مثال، شما می‌خواهید آیتم مربوط به برچسب «زرد» (yellow) را حذف کنید. به سادگی برچسب را به عنوان آرگومان تابع drop() مشخص کنید تا حذف شود.

```
>>> ser.drop('yellow')
red      0
blue     1
white    3
dtype: float64
```

برای حذف موارد بیشتر، تنها آرایه‌ای را با برچسب‌های مربوطه به کار گیرید.

```
>>> ser.drop(['blue','white'])
red       0
yellow    2
dtype: float64
```

در عوض با توجه به DataFrame، مقادیر می‌توانند با مراجعه به برچسب‌های هر دو محور حذف شوند. به عنوان مثال DataFrame زیر را اعلام کنید.

```
>>> frame = pd.DataFrame(np.arange(16).reshape((4,4)),
...                      index=['red','blue','yellow','white'],
...                      columns=['ball','pen','pencil','paper'])
>>> frame
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white     12   13      14     15
```

برای حذف ردیف‌ها، فقط از شاخص‌های ردیف‌ها بهره بگیرید.

```
>>> frame.drop(['blue','yellow'])
       ball  pen  pencil  paper
red       0    1       2      3
white    12   13      14     15
```

برای حذف ستون‌ها، همیشه باید شاخص‌های ستون‌ها را مشخص کنید، اما باید محوری را تعیین نمایید که مولفه‌ها از آن حذف شوند و این می‌تواند با استفاده از گزینه محور (axis) انجام شود. بنابراین برای اشاره به نام‌های ستون باید ***محور = ۱*** را مشخص کنید.

```
>>> frame.drop(['pen','pencil'],axis=1)
        ball  paper
red        0      3
blue       4      7
yellow     8     11
white     12     15
```

ترکیب ریاضی و داده

شاید قدرتمندترین ویژگی شامل شاخص‌ها در یک ساختار داده، این باشد که pandas قادر به انجام هم‌ترازی نمایه‌ها از دو ساختار داده متفاوت است. این به ویژه زمانی درست است که شما یک عمل حساب را بین آن‌ها انجام می‌دهید. در واقع، در طول این عملیات، نه تنها ممکن است شاخص‌های بین دو ساختار در یک ترتیب متفاوت باشند، بلکه ممکن است تنها در یکی از دو ساختار وجود داشته باشند. همانطور که از مثال‌های زیر می‌توانید ببینید، pandas در تنظیم شاخص‌ها در طول این عملیات بسیار قدرتمند است. به عنوان مثال، شما می‌توانید شروع به در نظر گرفتن دو سری کنید که در آن‌ها به ترتیب دو ردیف برچسب تعریف شده‌اند و کاملاً با یکدیگر تطابق ندارند.

```
>>> s1 = pd.Series([3,2,5,1],['white','yellow','green','blue'])
>>> s2 = pd.Series([1,4,7,2,1],['white','yellow','black','blue','brown'])
```

اکنون در میان انواع عملیات حساب، جمع کردن ساده را در نظر بگیرید. همانطور که می‌بینید، برخی از برچسب‌ها در هر دو سری وجود دارند، در حالی که برچسب‌های دیگر تنها در یکی از این دو سری وجود دارند. خوب، وقتی برچسب‌ها در هر دو اپراتور وجود دارند، مقادیر آن‌ها اضافه خواهد شد، در حالی که در مورد مخالف، آن‌ها نیز در نتیجه (سری جدید) اما با مقدار NaN نشان داده خواهند شد.

```
>>> s1 + s2
black    NaN
blue       3
brown    NaN
green    NaN
white      4
yellow     6
dtype: float64
```

در مورد DataFrameها، اگر چه ممکن است پیچیده‌تر به نظر برسد، هم‌ترازی از یک اصل پیروی می‌کند، اما هم برای ردیف‌ها و هم برای ستون‌ها انجام می‌شود.

```
>>> frame1 = pd.DataFrame(np.arange(16).reshape((4,4)),
...                       index=['red','blue','yellow','white'],
...                       columns=['ball','pen','pencil','paper'])
>>> frame2 = pd.DataFrame(np.arange(12).reshape((4,3)),
...                       index=['blue','green','white','yellow'],
...                       columns=['mug','pen','ball'])
>>> frame1
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white     12   13      14     15
>>> frame2
        mug  pen  ball
blue      0    1     2
green     3    4     5
white     6    7     8
yellow    9   10    11
>>> frame1 + frame2
        ball  mug  paper  pen  pencil
blue       6  NaN    NaN    6     NaN
green    NaN  NaN    NaN  NaN     NaN
red      NaN  NaN    NaN  NaN     NaN
white     20  NaN    NaN   20     NaN
yellow    19  NaN    NaN   19     NaN
```

عملیات بین ساختارهای داده

حالا که با ساختـارهای داده مانند سری و DataFrame آشنا شدید و دیدید که چـگونـه عملیات ابتدایی

مختلف می‌تواند بر روی آن‌ها انجام شود، زمان آن است که به عملیات‌های شامل دو یا چند مورد از این ساختارها بروید. برای مثال، در بخش قبلی دیدیم که چگونه عملگرهای حسابی بین دو شی به‌کار می‌روند. اکنون در این بخش عملیاتی را که می‌تواند بین دو ساختار داده انجام شود عمیق‌تر بررسی خواهید کرد.

روش‌های محاسباتی انعطاف‌پذیر

شما به تازگی دیدید که چگونه از عملگرهای ریاضی به طور مستقیم بر روی ساختارهای داده pandas استفاده کنید. عملیات مشابه را می‌توان با استفاده از روش‌های مناسب به نام روش‌های محاسباتی انعطاف‌پذیر نیز انجام داد.

- add()
- sub()
- div()
- mul()

برای این که این توابع را فراخوانی کنید، باید از یک ویژگی متفاوت از آنچه که برای کار با عملگرهای ریاضی استفاده می‌کنید، بهره بگیرید. به عنوان مثال، به جای نوشتن یک مجموع بین دو فریم (فریم ۱ + فریم ۲)، شما باید از فرمت زیر استفاده کنید:

```
>>> frame1.add(frame2)
        ball  mug  paper  pen  pencil
blue       6  NaN    NaN    6     NaN
green    NaN  NaN    NaN  NaN     NaN
red      NaN  NaN    NaN  NaN     NaN
white     20  NaN    NaN   20     NaN
yellow    19  NaN    NaN   19     NaN
```

همان‌طور که می‌توانید ببینید نتایج مشابه همان چیزی هستند که شما از اپراتور جمع «+» استفاده می‌کنید. همچنین می‌توانید توجه داشته باشید که اگر شاخص‌ها و نام‌های ستون از یک مجموعه به مجموعه دیگر تفاوت زیادی داشته باشند، با یک DataFrame جدید پر از مقادیر NaN مواجه خواهید شد.

```
>>> frame = pd.DataFrame(np.arange(16).reshape((4,4)),
...                      index=['red','blue','yellow','white'],
...                      columns=['ball','pen','pencil','paper'])
```

```
>>> frame
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white     12   13      14     15
>>> ser = pd.Series(np.arange(4), index=['ball','pen','pencil','paper'])
>>> ser
ball      0
pen       1
pencil    2
paper     3
dtype: int32
```

با رجوع به عملگرها، pandas به شما این امکان را می‌دهد که حتی بین ساختارهای مختلف به عنوان مثال، یک DataFrame و یک سری عملیات انجام دهید. برای مثال، ما این دو ساختار را به روش زیر تعریف می‌کنیم.

```
>>> frame = pd.DataFrame(np.arange(16).reshape((4,4)),
...                      index=['red','blue','yellow','white'],
...                      columns=['ball','pen','pencil','paper'])
>>> frame
        ball  pen  pencil  paper
red        0    1       2      3
blue       4    5       6      7
yellow     8    9      10     11
white     12   13      14     15
>>> ser = pd.Series(np.arange(4), index=['ball','pen','pencil','paper'])
>>> ser
ball      0
pen       1
pencil    2
paper     3
dtype: int32
```

دو ساختار داده‌ای که به تازگی تعریف شده‌اند به طور خاص ایجاد شده‌اند به طوری که شاخص‌های سری با نام ستون‌های DataFrame مطابقت دارند. به این ترتیب، می‌توانید یک عمل مستقیم انجام دهید.

```
>>> frame - ser
        ball  pen  pencil  paper
red        0    0       0      0
blue       4    4       4      4
yellow     8    8       8      8
white     12   12      12     12
```

همانطور که می‌بینید، عناصر سری از مقادیر DataFrame متناظر با همان شاخص بر روی ستون کم می‌شوند. مقدار برای تمام مقادیر ستون صرف‌نظر از شاخص آن‌ها کم می‌شود.

اگر یک شاخص در یکی از دو ساختار داده وجود نداشته باشد، نتیجه یک ستون جدید با این شاخص خواهد بود که همه عناصر آن NaN خواهند بود.

```
>>> ser['mug'] = 9
>>> ser
ball      0
pen       1
pencil    2
paper     3
mug       9
dtype: int64
>>> frame - ser
       ball  mug  paper  pen  pencil
red       0  NaN      0    0       0
blue      4  NaN      4    4       4
yellow    8  NaN      8    8       8
white    12  NaN     12   12      12
```

کاربرد تابع و نگاشت

این بخش توابع کتابخانه‌ای pandas را پوشش می‌دهد.

توابع با عنصر

کتابخانه pandas بر اساس NumPy ساخته می‌شود و لذا بسیاری از ویژگی‌های آن را گسترش می‌دهد تا آن‌ها را با ساختارهای داده جدید به صورت سری و DataFrame تطبیق دهد. در میان آن‌ها توابع کلی به نام ufunc وجود دارند؛ این دسته از توابع خاص هستند زیرا توسط عنصر در ساختار داده‌ها عمل می‌کنند.

```
>>> frame = pd.DataFrame(np.arange(16).reshape((4,4)),
...                      index=['red','blue','yellow','white'],
...                      columns=['ball','pen','pencil','paper'])
>>> frame
       ball  pen  pencil  paper
red       0    1       2      3
blue      4    5       6      7
yellow    8    9      10     11
white    12   13      14     15
```

به عنوان مثال، شما می‌توانید ریشه دوم هر مقدار را در DataFrame، با استفاده از np.sqrt() محاسبه کنید.

```
>>> np.sqrt(frame)
            ball       pen    pencil     paper
red     0.000000  1.000000  1.414214  1.732051
blue    2.000000  2.236068  2.449490  2.645751
yellow  2.828427  3.000000  3.162278  3.316625
white   3.464102  3.605551  3.741657  3.872983
```