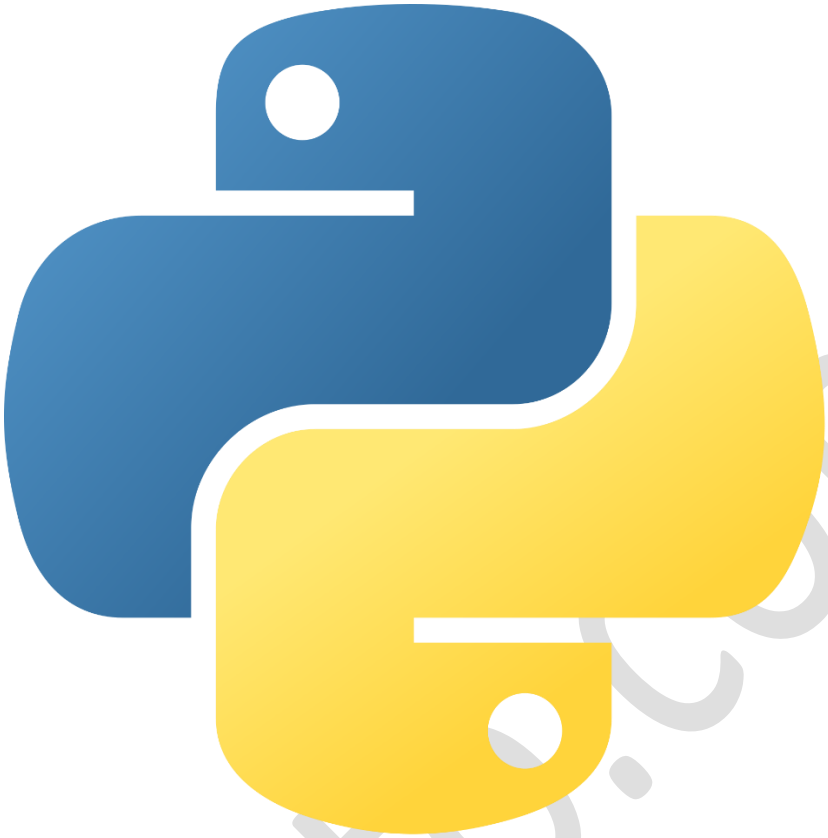

آموزش برنامه نویسی پایتون

پیج : learn_pythonn

آموزشگاه : فنی حرفه ای ویرا

مدرس : آرمین عالی

تاریخ : ۹ شهریور ۱۴۰۲

هشدار

تمامی محتوا و مباحث ارائه شده در این مطلب توسط

آموزشگاه فنی و حرفه ای ویرا ایجاد شده‌اند و تحت حق کپی‌رایت محافظت می‌شوند. هیچ‌گونه انتشار، کپی‌برداری یا توزیع محتوای این مطلب بدون موافقت کتبی از جناب آقای

میر مهدی میر تاج الدینی مجاز نمی‌باشد. تمامی حقوق مادی و معنوی این مطلب متعلق به vira-institute.ir می‌باشد.

سرفصل جزوه آموزشی پایتون:

۱. مقدمه

– معرفی پایتون و تاریخچه آن

۲. نصب پایتون و محیط توسعه

– نصب پایتون

– نصب ویژوال استودیو کد

۳. نصب کتابخانه‌ها در VS Code

– استفاده از مدیر پکیج pip

۴. شروع به کار با پایتون

– ایجاد فایل‌های .py

– مدیریت فایل‌ها در VS Code

۵. دستور `print`

– چاپ متن و متغییرها

۶. متغییرها

– تعریف و نام‌گذاری متغییرها

– نوع‌های مختلف داده

۷. داده‌های عددی و رشته‌ای

– داده‌های int، float **و** str

– اعمال جبری بر روی اعداد

۸. دستور `type`

– نوع داده‌ها را تشخیص دهی

۹. عملگرهای جبری و رشته‌ای

– عملیات +، –، * و /

– عملگرهای رشته‌ای

۱۰. کامنت‌ها

– نحوه نوشتن کامنت در کد

۱۱. عملگرهای توان و رادیکال وقسمت‌بندی و باقیمانده

– **، *، ۰.۵**، // و %**

۱۲. آشنایی با داده‌های list، boolean، dictionary و none

– تعریف و استفاده از انواع داده‌ها

۱۳. داده‌ی رشته‌ای (String)

– معرفی و خصوصیات رشته‌ها

– تعامل با کاراکترها و ایندکس‌ها

۱۴. String Interpolation (f"")

– ترکیب متغییرها با رشته‌ها

۱۵. اپراتورهای مقایسه و داده‌های بولین

– اپراتورهای ==، !=، >، <، >= و <=

– داده‌های بولین و استفاده در شرط‌ها

۱۶. اپراتورهای منطقی (and، or، not)

– اپراتورهای منطقی برای ایجاد شرط‌های پیچیده

۱۷. حلقه‌ها و مدیریت تکرار

– **حلقه‌های** for **و** while

۱۸. حلقه `for` **و تابع** `range()`

– **تکرار تعدادی بار با حلقه** `for`

– **استفاده از تابع** `range()` **برای تولید دنباله اعداد**

۱۹. حلقه `while` **و پروژه**

– **تکرار تا اجرای شرطی برآورده شود**

۲۰. مدیریت داده‌ها با List

– **تعریف و استفاده از لیست‌ها**

– **دسترسی به اعضای لیست با ایندکس**

۲۱. متدهای اضافه کردن به List

– **متدهای** append، extend **و** insert **برای اضافه کردن اعضا به لیست**

۲۲. متدهای حذف از List

– **متدهای** pop، remove **و** clear **برای حذف اعضا از لیست**

۲۳. مدیریت داده‌ها با Dictionary

– **تعریف و استفاده از دیکشنری‌ها**

– **دسترسی به اطلاعات با کلیدها**

۲۴. متدهای Dictionary

– **متدهای** keys، values **و** items **برای دسترسی به محتوای دیکشنری**

۲۵. متد update **در** Dictionary

– **به‌روزرسانی یا اضافه کردن زوج‌های کلید–مقدار در دیکشنری**

۲۶. List Comprehension **و** Dictionary Comprehension

– **تکنیک‌های سریع برای ایجاد لیست و دیکشن**

۲۷. **مقدمه به Tuple **و** Set******

– **معرفی** Tuple **و** Set

– **تفاوت‌ها و موارد استفاده**

۲۸. ** Tuple **در پایتون****

– **تعریف و مفهوم** Tuple

– **دسترسی به اعضا و تغییر آنها**

– **کاربردها و مزایای** Tuple

۲۹. ** Set **در پایتون****

– **تعریف و مفهوم** Set

– **ایجاد** Set **و عملیات مجموعه‌ای**

– **کاربردها و مزایای** Set

۳۰. **مقدمه به Lambda **در پایتون****

– **تعریف و کاربردهای** Lambda

۳۱. **استفاده از تابع Lambda******

– **تعریف و استفاده از توابع** Lambda

۳۲. **مقدمه به Map **و** Lambda******

– **تعریف** Map **و استفاده از توابع** Lambda

۳۳. **استفاده از تابع Map **با لیست****

– **کاربردها و مثال‌های استفاده از تابع** Map

۳۴. **مقدمه به Filter **و** Lambda******

– **تعریف** Filter **و استفاده از توابع** Lambda

۳۵. **استفاده از تابع Filter **با لیست****

– **کاربردها و مثال‌های استفاده از تابع** Filter

۳۶. **استفاده از توابع All **و** Any******

– **تعریف و کاربرد توابع** All **و** Any

۳۷. **مقدمه به توابع Sorted **و** Min **و** Max **و** Reversed******

– **معرفی و کاربردهای توابع** Sorted **و** Min **و** Max **و** Reversed

۳۸. **توابع Sorted **و** Min **و** Max **و** Reversed******

– **کاربردها و مثال‌های استفاده از توابع** Sorted **و** Min **و** Max **و** Reversed

۳۹. **استفاده از تابع Len******

– **معرفی و کاربردهای تابع** Len

۴۰. **استفاده از توابع Abs **و** Sum **و** Round******

– **معرفی و کاربردهای توابع** Abs **و** Sum **و** Round

۴۱. **مقدمه به توابع Zip******

– **تعریف و کاربرد توابع** Zip

۴۲. **استفاده از تابع Zip******

– **مثال‌های استفاده از تابع** Zip

۴۳. **مقدمه به Error Handling **در پایتون****

– **تعریف و کاربردهای** Error Handling

۴۴. **استفاده از دستورات Try **و** Except******

– **تعریف و مثال‌های استفاده از دستورات** Try **و** Except

۴۵. **استفاده از دستورات Finally **و** Else******

– **تعریف و مثال‌های استفاده از دستورات** Finally **و** Else

۴۶. **مقدمه به Module**ها در پایتون****

– **تعریف و کاربردهای** Module**ها**

۴۷. **استفاده از Module**های تعریف شده****

– **تعریف و استفاده از** Module**های تعریف شده**

۴۸. **استفاده از Module**های استاندارد****

– **استفاده از** Module**های استاندارد پایتون**

۴۹. **مقدمه به Object-Oriented Programming (OOP)******

– **تعریف و مفاهیم اولیه** OOP

۵۰. **کلاس‌ها و اشیاء در OOP******

– **تعریف و استفاده**

از کلاس‌ها و اشیاء در OOP

۵۱. **متدها و ویژگی‌ها در کلاس‌ها**

– **تعریف و استفاده از متدها و ویژگی‌ها در کلاس‌ها**

۵۲. **ساختگی‌ها (Constructors**) و** Destructor******

– **تعریف و استفاده از ساختگی‌ها و** Destructor **در کلاس‌ها**

۵۳. **مفهوم Inheritance **(ارث‌بری)****

– **تعریف و مفهوم** Inheritance

۵۴. **استفاده از Inheritance **(ارث‌بری)****

– **استفاده از** Inheritance **در** OOP

۵۵. **مفهوم Polymorphism (چندریختی)**

– **تعریف و مفهوم** Polymorphism

۵۶. **استفاده از Polymorphism (چندریختی)**

– **استفاده از** Polymorphism **در** OOP

۵۷. **مفهوم Encapsulation (محافظت از داده)**

– **تعریف و مفهوم** Encapsulation

۵۸. **استفاده از Encapsulation (محافظت از داده)*

– **استفاده از** Encapsulation **در** OOP

۵۹. **مقدمه به repr در پایتون**

– **تعریف و کاربردهای** repr **در پایتون**

۶۰. **استفاده از repr در پایتون**

– **مثال‌ها و استفاده از** repr

۶۱. **مقدمه به Getter و Setter و Properties**

– **تعریف و کاربرد** Getter **و** Setter **و** Properties

۶۲. **استفاده از Getter و Setter و Properties**

– **استفاده از** Getter **و** Setter **و** Properties **در** OOP

۶۳. **مقدمه به Iterable و Iterator**

– **تعریف و مفهوم** Iterable **و** Iterator

۶۴. **استفاده از Iterable و Iterator**

– **استفاده از** Iterable **و** Iterator **در پایتون**

۶۵. **متدهای iter و next و ایجاد Custom For**

– **تعریف و استفاده از متدهای** iter **و** next **و ایجاد** Custom For

۶۶. **Generatorها در پایتون****

– **معرفی و کاربردهای** Generator**ها**

۶۷. **استفاده از Generator**ها****

– **استفاده از** Generator**ها در پایتون**

۶۸. **مقدمه به توابع در پایتون**

– **تعریف و مفهوم توابع**

۶۹. **استفاده از توابع در پایتون**

– **استفاده از توابع در پایتون**

۷۰. **مقدمه به Decorator**ها****

– **تعریف و کاربردهای** Decorator**ها**

۷۱. **استفاده از Decorator**ها****

– **استفاده از** Decorator**ها در پایتون**

۷۲. **مقدمه به کار با API******

– **تعریف و مفهوم** API

۷۳. **استفاده از API**ها در پایتون****

– **استفاده از** API**ها در پایتون**

۷۴. **کار با دیتابیس در پایتون**

– **معرفی و کار با دیتابیس‌ها در پایتون**

۷۵. **مقدمه به GUI **و توسعه نرم‌افزار با** Tkinter******

– **تعریف و مفهوم** GUI **و** Tkinter

۷۶. **ساخت دکمه و لیبل و جعبه متنی با Tkinter**

– **ساخت و کار با دکمه، لیبل و جعبه متنی در** Tkinter

۷۷. **استفاده از ویجت Entry **در** Tkinter**

– **ساخت و استفاده از ویجت** Entry **در** Tkinter

۷۸. **پیاده‌سازی بازی X **و** O **با** Tkinter**

– **پیاده‌سازی بازی** X **و** O **با استفاده از** Tkinter

فصل ۱: مقدمه به زبان پایتون

۱.۱ معرفی پایتون:

زبان برنامه‌نویسی پایتون یکی از محبوب‌ترین و پرکاربردترین زبان‌های برنامه‌نویسی در دنیا است. این زبان به دلیل سادگی سینتکس، خوانایی بالا و قابلیت توسعه و نگهداری آسان کد، به ویژه در پروژه‌های کوچک تا متوسط وب، داده‌کاوی، هوش مصنوعی، علوم داده و توسعه نرم‌افزار مورد استفاده قرار می‌گیرد.

۱.۲ تاریخچه پایتون:

– پایتون برای اولین بار در اواخر دهه ۱۹۸۰ توسط "گوئیدو وان راسوم" (Guido van Rossum) در کشور هلند ابداع شد. نام "پایتون" به افتخار نام کمدی تلویزیونی "مونتی پایتون" انتخاب شد و نشان از نگرانی و طنزپردازی ایجاد کننده زبان دارد.

– نسخه اولیه پایتون در سال ۱۹۹۱ منتشر شد. این نسخه نسبت به زبان‌های دیگر کم‌ترین پیچیدگی را داشت و باعث تسهیل فرآیند برنامه‌نویسی شد.

– در سال‌های بعد، پایتون به سرعت شهرت و محبوبیت پیدا کرد. نسخه‌های جدید با افزودن ویژگی‌ها و بهبودهای متعدد منتشر شدند که منجر به افزایش استفاده و توسعه‌دهندگان پایتون شد.

– با انتشار نسخه ۳.۰ پایتون در سال ۲۰۰۸، تغییرات اساسی در زبان اعمال شدند که بهبودهای مهمی در عملکرد، قابلیت‌ها و سینتکس زبان به همراه داشت.

۱.۳ ویژگی‌های پایتون:

– **خوانایی بالا:** سینتکس ساده و قواعد خوانایی در پایتون باعث می‌شود که کدهای نوشته شده به راحتی قابل فهم باشند.

– **سهولت نگهداری:** پایتون از تورفتگی (Indentation) برای تعریف بلوک‌های کد استفاده می‌کند، که به صورت طبیعی باعث می‌شود کد منظم و قابل نگهداری تولید شود.

– **کامل و پویا:** پایتون دارای مجموعه‌ای از کتابخانه‌ها و ماژول‌های پیش‌فرض است که به برنامه‌نویسان کمک می‌کند تا به سرعت و با کمترین تلاش به اهداف خود برسند.

– **توسعه سریع:** پایتون به برنامه‌نویسان اجازه می‌دهد که به سرعت ایده‌های خود را پیاده‌سازی کنند و بازخورد دریافت کنند.

– **پورتابلیته:** پایتون بر روی اکثر سیستم‌عامل‌ها اجرا می‌شود و کدهای نوشته شده را می‌توان به سادگی به سیستم‌های دیگر منتقل کرد.

– **پشتیبانی از متغیرها و انواع داده‌های پیشرفته:** پایتون به برنامه‌نویسان امکان استفاده از انواع داده‌های پیچیده‌تر نظیر لیست‌ها، دیکشنری‌ها، تاپل‌ها و مجموعه‌ها را می‌دهد.

۱.۴ نتیجه‌گیری:

پایتون با ترکیبی از خوانایی بالا، قدرت، انعطاف‌پذیری و جامعه فعال برنامه‌نویسان، به یکی از محبوب‌ترین زبان‌های برنامه‌نویسی تبدیل شده است. این زبان به برنامه‌نویسان اجازه می‌دهد تا به راحتی پروژه‌های متنوعی را از وب تا هوش مصنوعی و علوم داده توسعه دهند.

فصل ۲: نصب پایتون و ویژوال استودیو کد

۲.۱ نصب پایتون:

برای نصب پایتون، شما می‌توانید از منابع مختلفی مانند Anaconda یا نصب مستقیم از مخازن رسمی استفاده کنید. در اینجا نحوه نصب پایتون با استفاده از مخازن رسمی آورده شده است:

۱. به وب‌سایت رسمی پایتون در آدرس https://www.python.org بروید.

۲. در بخش "Downloads"، نسخه‌ای از پایتون را که مورد نظر دارید، انتخاب کنید. توصیه می‌شود از آخرین نسخه پایتون استفاده کنید.

۳. برای سیستم‌عامل خود (مانند Windows، macOS یا Linux)، فایل نصب مناسب را دانلود کنید.

۴. پس از دانلود، فایل نصب را اجرا کنید و دستورهای نصب را دنبال کنید.

۵. در هنگام نصب، مطمئن شوید که گزینه "Add Python to PATH" انتخاب شده باشد. این گزینه به شما اجازه می‌دهد از هر مکانی در ترمینال یا دستور خط فرمان به پایتون دسترسی داشته باشید.

۶. پس از نصب موفقیت‌آمیز، می‌توانید با وارد کردن دستور "python" در ترمینال، ورژن نصب‌شده پایتون را بررسی کنید.

۲.۲ نصب ویژوال استودیو کد:

ویژوال استودیو کد (Visual Studio Code) یک محیط توسعه یکپارچه (IDE) کم‌حجم و قدرتمند است که به برنامه‌نویسان امکان توسعه و پیاده‌سازی پروژه‌ها را با آسانی می‌دهد. در اینجا نحوه نصب ویژوال استودیو کد آورده شده است:

۱. به وب‌سایت رسمی ویژوال استودیو کد در آدرس https://code.visualstudio.com بروید.

۲. دانلود نسخه متناسب با سیستم‌عامل خود (مانند Windows، macOS یا Linux) را انتخاب کنید.

۳. فایل نصب را دانلود کرده و اجرا کنید.

۴. دستورهای نصب را دنبال کرده و به تمامی گزینه‌ها پاسخ دهید. توصیه می‌شود تمامی پیش‌فرض‌ها را قبول کنید مگر اینکه دلیل خاصی برای تغییر آن‌ها داشته باشید.

۲.۳ نکته‌های نصب:

– **مسیر نصب: هنگام نصب پایتون و ویژوال استودیو کد، به مسیر نصب دقت کنید. معمولاً پیشنهاد می‌شود از مسیر پیش‌فرض استفاده کنید.**

– **نسخه متناسب: مطمئن شوید که نسخه‌های پایتون و ویژوال استودیو کد متناسب با سیستم‌عامل خود و همچنین با یکدیگر استفاده می‌شوند.**

– **تایید نصب ماژول‌ها: هنگام نصب پکیج‌ها و ماژول‌ها در پایتون با استفاده از ابزارهای مانند pip، مطمئن شوید که دستورات را در محیط مدیریت پکیج‌ها (مانند محیط مجازی) اجرا می‌کنید.**

۲.۴ نتیجه‌گیری:

نصب پایتون و ویژوال استودیو کد اولین قدم در توسعه پروژه‌های پایتونی شما است. با رعایت نکات نصب و استفاده از ابزارهای مناسب، به راحتی می‌توانید به توسعه و پیاده‌سازی پروژه‌های خود بپردازید.

فصل ۳: نصب کتابخانه‌ها در ویژوال استودیو کد

۳.۱ نصب کتابخانه‌ها با استفاده از ترمینال:

یکی از راه‌های رایج نصب کتابخانه‌ها در پایتون، استفاده از ترمینال و ابزار مدیریت پکیج pip است. در ویژوال استودیو کد نیز می‌توانید از ترمینال محیط توسعه خود استفاده کنید:

۱. باز کردن پروژه در ویژوال استودیو کد.

۲. اجرای ویژگی "Terminal" (ترمینال) در بالای پنجره.

۳. در ترمینال، دستور نصب کتابخانه‌ها با استفاده از pip را وارد کنید، به عنوان مثال:

```
pip install نام_کتابخانه
```

جایگزین "نام_کتابخانه" با نام ورودی کتابخانه مورد نظر شوید.

۴. پس از اجرای دستور، pip به طور خودکار کتابخانه‌ها را دانلود و نصب خواهد کرد.

۳.۲ نصب کتابخانه‌ها با استفاده از ویژوال استودیو کد:

ویژوال استودیو کد از امکاناتی برای نصب کتابخانه‌ها به طور مستقیم در محیط خود برخوردار است. این کار به شما امکان می‌دهد بدون نیاز به ترمینال، کتابخانه‌ها را نصب کرده و پروژه خود را به‌روز نگه دارید:

۱. باز کردن پروژه در ویژوال استودیو کد.

۲. باز کردن منوی "View" **و انتخاب گزینه** "Extensions" **(یا همان** Ctrl+Shift+X**).**

۳. جستجوی نام کتابخانه مورد نظر در باکس موجود و انتخاب آن از نتایج.

۴. در صفحه کتابخانه، گزینه "Install" **(نصب) را انتخاب کنید.**

۵. پس از نصب کتابخانه، می‌توانید از آن در پروژه‌های خود استفاده کنید.

۳.۳ نکته‌های نصب کتابخانه‌ها:

– توصیه می‌شود همواره از آخرین نسخه کتابخانه‌ها استفاده کنید تا از بهترین ویژگی‌ها و به‌روزرسانی‌ها بهره‌برداری کنید.

– برخی از کتابخانه‌ها به وابستگی‌های خاصی نیاز دارند. در این صورت، pip **به طور خودکار وابستگی‌ها را نیز نصب خواهد کرد.**

– برای اجتناب از تداخل وابستگی‌ها در پروژه‌های مختلف، استفاده از محیط‌های مجازی توصیه می‌شود.

۳.۴ نتیجه‌گیری:

نصب کتابخانه‌ها در ویژوال استودیو کد با استفاده از ترمینال یا ابزار Extensions **بسیار آسان است. با توجه به نیازهای پروژه‌های خود، می‌توانید کتابخانه‌های مختلف را به راحتی نصب کرده و از آن‌ها در توسعه پروژه‌های خود بهره‌برداری کنید.**

فصل ۴: ایجاد و مدیریت فایل‌های .py در ویژوال استودیو کد

۴.۱ ایجاد یک فایل .py:

یکی از ابتدایی‌ترین کارها در توسعه با پایتون، ایجاد فایل‌های .py است که در آنها کدهای برنامه‌نویسی خود را قرار می‌دهید:

۱. باز کردن پروژه در ویژوال استودیو کد.

۲. در منوی "File" گزینه "New File" (یا همان Ctrl+N) را انتخاب کنید.

۳. نام فایل را با پسوند .py تعیین کنید، به عنوان مثال: "my_script.py".

۴. کدهای مورد نظر خود را درون فایل .py وارد کنید.

۴.۲ ذخیره و مدیریت فایل‌ها:

ویژوال استودیو کد امکانات مدیریت فایل‌های پروژه را فراهم کرده است که به شما کمک می‌کند که به راحتی فایل‌های پروژه خود را مدیریت کنید:

– ذخیره فایل: برای ذخیره یک فایل باز، از گزینه "File" و سپس "Save" (یا همان Ctrl+S) استفاده کنید. این کار باعث ذخیره تغییرات اعمال‌شده در فایل می‌شود.

– **ذخیره فایل با نام:** برای ذخیره فایل با نام خاصی، ابتدا از منوی "File" و گزینه "Save As..." انتخاب کنید. سپس نام دلخواه و پسوند .py را وارد کرده و ذخیره کنید.

– **باز کردن فایل:** برای باز کردن یک فایل، از منوی "File" و گزینه "Open File" (یا همان Ctrl+O) استفاده کنید. فایل مورد نظر خود را انتخاب کنید.

– **نمایش فهرست فایل‌ها:** در پنجره سمت چپ ویژوال استودیو کد، فهرست فایل‌ها و پوشه‌های پروژه‌تان را مشاهده خواهید کرد. با کلیک بر روی فایل‌ها، آنها را باز کرده و ویرایش کنید.

۴.۳ نکته‌های مدیریت فایل‌ها:

– توصیه می‌شود فایل‌های پروژه خود را در پوشه‌های منظم و سازمان‌یافته قرار دهید تا بهترین نظم را حفظ کنید.

– ویژوال استودیو کد از ویژگی‌های "Source Control" نیز پشتیبانی می‌کند که به شما اجازه می‌دهد تغییرات در فایل‌های پروژه را ردیابی کنید.

۴.۴ نتیجه‌گیری:

ایجاد و مدیریت فایل‌های .py در ویژوال استودیو کد بسیار آسان است. با استفاده از این امکانات، می‌توانید کدهای برنامه‌نویسی خود را در فایل‌های جداگانه ذخیره کرده و به راحتی به مدیریت پروژه‌های پایتون بپردازید.

فصل ۵: دستور `print` در پایتون

۵.۱ معرفی دستور `print`:

دستور `print` یکی از اساسی‌ترین و پرکاربردترین دستورات در زبان برنامه‌نویسی پایتون است. این دستور برای نمایش متن، مقادیر متغیرها و اشیاء مختلف در خروجی ترمینال یا کنسول مورد استفاده قرار می‌گیرد.

۵.۲ نحوه استفاده از دستور `print`:

ساده‌ترین فرم دستور `print` به این صورت است:

```
print("متن یا مقدار متغیر")
```

در اینجا، شما متن یا مقدار متغیر را درون پرانتز قرار داده و با استفاده از این دستور، آن را در خروجی نمایش می‌دهید.

۵.۳ نمونه‌های استفاده از دستور `print`:

۱. نمایش متن ساده:

```
print("سلام، این یک نمونه پرینت است.")
```

۲. نمایش مقدار یک متغیر:

```
name = "علی"
print(name)
```

۳. ترکیب متن و مقدار متغیر:

```
age = 25
print("سن من", age, "سال است.")
```

۴. استفاده از ترکیب‌های متغیرها و متن:

```
x = 10
y = 5
print("جمع", x, "و", y, "برابر است با:", x + y)
```

****۵.۴ قالب‌بندی خروجی با دستور `print`:****

شما می‌توانید با استفاده از قالب‌بندی خاص، خروجی دستور `print` را زیباتر و بهتر مشخص کنید. این کار با استفاده از نمادهای خاص درون متن انجام می‌شود. به عنوان مثال:

```
name = "مریم"
age = 30
print("نام: {}, سن: {}".format(name, age))
```

یا با استفاده از فمیلی‌ر:

```
name = "مریم"
age = 30
print(f"نام: {name}, سن: {age}")
```

****۵.۵ نکته‌های استفاده از دستور `print`:****

– در دستور `print` می‌توانید تعدادی آرگومان (متن یا مقادیر متغیرها) جداگانه قرار دهید. آنها به طور خودکار با یک فاصله جدا می‌شوند.

– برای ایجاد فاصله یا تب در خروجی، از کاراکترهای خاصی مانند `\t` برای تب و `\n` برای خط جدید استفاده کنید.

– دستور `print` پیش‌فرضاً خودکار خط جدید را در انتهای خروجی اضافه می‌کند. اگر می‌خواهید این خط جدید را حذف یا تغییر دهید، از پارامتر `end` استفاده کنید.

```
print("متن", end=" ")
print("بدون خط جدید")
```

****۵.۶ نتیجه‌گیری:****

دستور `print` به عنوان یک ابزار اساسی در پایتون برای نمایش اطلاعات در خروجی استفاده می‌شود. با اجاد ترکیب‌های مختلف از متن و مقادیر متغیرها، می‌توانید خروجی‌های متنوعی در کنسول نمایش دهید و با استفاده از قالب‌بندی مناسب، خروجی را زیبا و قابل فهم کنید.

فصل ۶: متغیرها و نام‌گذاری در پایتون

۶.۱ مقدمه‌ای درباره متغیرها:

متغیرها در پایتون به شما اجازه می‌دهند تا داده‌ها و اطلاعات را در حافظه ذخیره کرده و با آن‌ها کار کنید. هر متغیر دارای یک نام منحصر به فرد است و مقدار مشخصی را می‌تواند نگه دارد.

۶.۲ نام‌گذاری متغیرها:

هنگام نام‌گذاری متغیرها در پایتون، باید توجه داشته باشید که:

– نام متغیر باید با یک حرف یا کاراکتر آندرسکور (_) شروع شود.

– نام متغیرها تا حد امکان باید معنی داشته باشد و توصیف کننده محتوای آن متغیر باشد.

– نام متغیرها حساس به بزرگی و کوچکی حروف است، بنابراین `myVar` و `myvar` دو متغیر متفاوت هستند.

– نام متغیرها نمی‌تواند شامل فاصله یا کاراکترهای خاصی مانند @، $، ٪ و ... باشد.

****۶.۳ انواع داده‌ها در پایتون:****

پایتون دارای انواع داده‌های مختلفی است که می‌توانید در متغیرها ذخیره کنید:

– اعداد صحیح (integer): مثلاً `x = ۵`

– اعداد اعشاری (float): مثلاً `pi = ۳,۱٤`

– رشته‌ها (string): مثلاً `name = "John"`

– لیست‌ها (list): مثلاً `numbers = [۱, ۲, ۳, ۴]`

– دیکشنری‌ها (dictionary): مثلاً `person = {"name": "Ali", "age": ۳۰}`

– تاپل‌ها (tuple): مثلاً `coordinates = (۱۰, ۲۰)`

– مجموعه‌ها (set): مثلاً `fruits = {"apple", "banana", "orange"}`

****۶.۴ نمونه‌های تعریف متغیر و استفاده از آن‌ها:****

```
# تعریف متغیرها
age = 25
name = "Ali"
grades = [18, 16, 20, 19]
person = {"first_name": "Mary", "last_name":
"Smith"}
coordinates = (5, 10)
fruits = {"apple", "banana", "orange"}

# استفاده از متغیرها
```

```
print(name)          # نمایش مقدار متغیر name
total_grades = sum(grades)  # محاسبه مجموع عناصر لیست grades و ذخیره در متغیر total_grades
print(person["first_name"],
person["last_name"])  # نمایش اطلاعات از دیکشنری person
```

۶.۵ نکته‌های مربوط به متغیرها:

– انواع داده‌های مختلف نیاز به مدیریت متفاوت دارند. مثلاً برای عملیات روی لیست‌ها و دیکشنری‌ها نیاز به روش‌ها و متدهای خاصی دارید.

– هنگام تعریف متغیرها، بهتر است نام‌های معنی‌دار و توصیفی انتخاب کنید تا کدتان خوانا و قابل فهم باشد.

– استفاده از توضیحات (comment) در

کد به شما کمک می‌کند تا منظور و عملکرد متغیرها را بهتر درک کنید و در آینده کدتان را بهبود دهید.

۶.۶ نتیجه‌گیری:

متغیرها ابزار اساسی برای ذخیره و مدیریت داده‌ها در پایتون هستند. با توجه به انواع مختلف داده‌ها و نیازهای پروژه‌های خود، متغیرهای مختلفی ایجاد کرده و از آن‌ها برای انجام محاسبات و عملیات مورد نیاز خود استفاده کنید.

فصل ۷: انواع داده‌ها در پایتون int :، float، complex، str**

۷,۱ انواع داده‌ها در پایتون:

در پایتون، انواع مختلفی از داده‌ها وجود دارند که برای ذخیره و مدیریت داده‌های مختلف به کار می‌روند. چندی از انواع اصلی داده‌ها هستند (strings) و رشته‌ها (complex numbers) ، اعداد مختلط (floats) ، اعداد اعشاری (integers) در پایتون شامل اعداد صحیح.

۷,۲ اعداد صحیح (integers):

اعداد صحیح به اعدادی اطلاق می‌شود که بدون مقدار اعشاری و قسمت تعداد صفری آنها هستند. به عنوان مثال: ۵، −۱۰، ۰.

۷٫۳ **اعداد اعشاری (floats):**

اعداد اعشاری شامل اعدادی با مقادیر اعشاری هستند. این اعداد با استفاده از نقطه به عنوان علامت اعشاری نمایش داده می‌شوند. به عنوان مثال: ۳.۱۴، −۰.۵، ۲.۰.

۷٫۴ **اعداد مختلط (complex numbers):**

اعداد مختلط از دو قسمت تشکیل شده‌اند: قسمت حقیقی و قسمت موهومی. این اعداد با استفاده از حروف مختلف به عنوان علامت مختلط j، زنمایش داده می‌شوند. به عنوان مثال: ۳+۲، −۱−۴j.

۷٫۵ **رشته‌ها (strings):**

رشته‌ها مجموعه‌ای از کاراکترها هستند که می‌توانند حروف، اعداد، نمادها و حتی کاراکترهای خاص مانند فاصله باشند. رشته‌ها با استفاده از علامت‌های " یا ' اطراف خود نمایش داده می‌شوند. به عنوان مثال: 'Python'، "Hello World!".

۷٫۶ **مثال‌هایی از تعریف و استفاده از انواع داده‌ها:

```
# اعداد صحیح
age = 25
count = -10
zero = 0

# اعداد اعشاری
pi = 3.14
temperature = -2.5
half = 0.5

# اعداد مختلط
z1 = 2 + 3j
z2 = -1 - 2j

# رشته‌ها
```

```
name = "John"
message = 'Hello, world!'
address = "123 Main St"
```

:نکته‌های مربوط به انواع داده‌ها ۷,۷

- در پایتون، نوع داده‌ها به طور خودکار تشخیص داده می‌شوند و شما نیازی به تعیین نوع داده به صورت صریح ندارید.

- می‌توانید از توابع تبدیل مانند `float()`، `int()` و ... برای تبدیل نوع داده‌ها به یکدیگر (برای مثال تبدیل float به int یا بالعکس) استفاده کنید.

- توجه داشته باشید که در عملیات‌های محاسباتی ممکن است نوع داده‌ها تاثیر داشته باشد. بنابراین باید به نوع داده‌ها در طراحی کد خود توجه داشته باشید.

:نتیجه‌گیری ۷,۸

در پایتون، انواع داده‌های مختلف از جمله اعداد صحیح، اعشاری، مختلط و رشته‌ها وجود دارند. با استف

اده از این انواع داده‌ها، می‌توانید انواع مختلف اطلاعات را ذخیره و در پروژه‌های خود مورد استفاده قرار دهید.

فصل ۸: دستور `type` و تشخیص نوع داده‌ها در پایتون

۸.۱ معرفی دستور `type`:

در پایتون، دستور `type` برای تشخیص نوع یک متغیر یا مقدار استفاده می‌شود. این دستور به شما امکان می‌دهد تا نوع داده‌ای که یک متغیر دارد را بدست آورید.

۸.۲ استفاده از دستور `type`:

برای استفاده از دستور `type`، متغیر یا مقدار مورد نظر را درون پرانتز قرار داده و نوع آن را بررسی می‌کنید:

```
x = 5
y = 3.14
name = "John"

print(type(x))     # نوع اعداد صحیح
print(type(y))     # نوع اعداد اعشاری
print(type(name)) # نوع رشته
```

****۸.۳ نمونه‌های تشخیص نوع داده‌ها:****

```
num = 10
pi = 3.14
message = "Hello"
complex_num = 2 + 3j

print(type(num))         # <class 'int'>
print(type(pi))          # <class 'float'>
print(type(message))     # <class 'str'>
print(type(complex_num)) # <class 'complex'>
```

****۸.۴ نکته‌های مربوط به دستور `type`:****

– دستور `type` به شما یک نوع داده از جنس `type` باز می‌گرداند. برای مشخص‌تر دیدن نوع داده، می‌توانید از تابع `str()` برای تبدیل آن به رشته استفاده کنید.

– از تایپ‌های مختلف برای تشخیص نوع داده استفاده می‌شود، به عنوان مثال: ``int`، `float`، `str`` و ...

****۸.۵ نتیجه‌گیری:****

دستور `type` به شما امکان تشخیص نوع داده‌های مختلف را در پایتون می‌دهد. این اطلاعات مفیدی است که می‌توانید از آن برای تأیید نوع داده‌ها و اطمینان از صحت انجام عملیات‌ها در برنامه‌های خود استفاده کنید.

فصل ۹: اعمال جبری در پایتون (+، −، *، /) و استفاده از دستور `print`

۹.۱ اعمال جبری:

در پایتون، شما می‌توانید اعمال جبری مختلفی مانند جمع (+)، تفریق (−)، ضرب (*) و تقسیم (/) را بر روی اعداد انجام دهید. این اعمال به شما امکان می‌دهند تا با اعداد کارهای مختلفی انجام دهید.

۹.۲ استفاده از دستور `print` برای نمایش نتایج:

برای نمایش نتایج اعمال جبری می‌توانید از دستور `print` استفاده کنید. با استفاده از این دستور، می‌توانید نتایج محاسبات خود را در خروجی نمایش دهید.

۹.۳ نمونه‌های اعمال جبری با دستور `print`:

```
# جمع
result1 = 10 + 5
print("جمع:", result1)

# تفریق
result2 = 20 - 8
print("تفریق:", result2)

# ضرب
result3 = 3 * 4
print("ضرب:", result3)

# تقسیم
result4 = 15 / 3
print("تقسیم:", result4)
```

۹.۴ نکته‌های مربوط به اعمال جبری و دستور `print`:

– شما می‌توانید اعمال جبری را بر روی اعداد صحیح و اعشاری انجام دهید.

– برای اعمال تقسیم، نتیجه به طور پیش‌فرض به نوع اعشاری تبدیل می‌شود.

– می‌توانید از پرانتزها برای مشخص کردن اولویت اعمال استفاده کنید.

۹.۵ نتیجه‌گیری:

اعمال جبری اساسی در پایتون شامل جمع، تفریق، ضرب و تقسیم است که با استفاده از این اعمال می‌توانید با اعداد مختلف کارهای محاسباتی را انجام دهید. با استفاده از دستور `print` نتایج این اعمال را در خروجی نمایش داده و برنامه‌های خود را بررسی کنید.

فصل ۱۰: کامنت کردن در پایتون و توضیحات مستند

۱۰.۱ کامنت‌ها در پایتون:

کامنت‌ها (توضیحات) در پایتون به شما امکان می‌دهند تا توضیحاتی را در کدهایتان اضافه کنید که برای انسان‌ها قابل فهم باشند، اما توسط ماشین‌ها نادیده گرفته می‌شوند. کامنت‌ها به شما اجازه می‌دهند که توضیحاتی را درباره عملکرد بخش‌های مختلف کد، توجیه‌نامه‌ها، راهنماها و غیره ارائه دهید.

۱۰.۲ نوع‌های کامنت در پایتون:

در پایتون، دو نوع کامنت وجود دارد:

– کامنت تک‌خطی: با استفاده از # کامنت‌های تک‌خطی را ایجاد می‌کنید. هرچیزی که پس از # بنویسید، به عنوان کامنت در نظر گرفته می‌شود.

۱۰.۳ نمونه‌های کامنت‌ها:

```
# این یک کامنت تک‌خطی است
print("Hello, World!")  # توضیح این کامنت به تابع print می‌دهد

# محاسبه میانگین دو عدد
num1 = 10
num2 = 20
average = (num1 + num2) / 2
print("میانگین:", average)
```

۱۰.۴ کامنت‌های چندخطی:

برای کامنت‌های چندخطی، شما می‌توانید از سه علامت نقل قول (تک یا دوتایی) قبل و بعد از متن کامنت استفاده کنید:

```
"""
این یک کامنت چندخطی است
در این قسمت می‌توانید توضیحات بیشتری را اضافه کنید
"""
print("Hello, World!")
```

۱۰.۵ توضیحات مستند (Docstrings)**:**

توضیحات مستند (یا داک استرینگ‌ها) توسط معمولاً برای توضیح کامل‌تر و مستندسازی توابع، کلاس‌ها و ماژول‌ها استفاده می‌شوند. این توضیحات با استفاده از سه علامت نقل قول تک یا دوتایی قرار می‌گیرند و به شما اجازه می‌دهند تا توضیحات دقیق‌تری درباره نحوه استفاده و ورودی‌ها و خروجی‌های توابع ارائه دهید.

۱۰.۶ نمونه‌های توضیحات مستند:

```
def calculate_average(numbers):
    """
    محاسبه میانگین اعداد داده‌شده.

    :param numbers: یک لیست از اعداد
    :type numbers: list
    :return: میانگین اعداد
    :rtype: float
    """
    total = sum(numbers)
    avg = total / len(numbers)
    return avg
```

۱۰.۷ نکته‌های مربوط به کامنت‌ها و توضیحات مستند:

– کامنت‌ها و توضیحات مستند برای افزایش خوانایی و قابلیت فهم کد بسیار مهم هستند.

– استفاده از کامنت‌ها به شما کمک می‌کند تا کد را توضیح دهید، اما توجه داشته باشید که کدتان باید به خودی خود قابل فهم باشد.

– در توضیحات مستند، می‌توانید از قالب‌های مشخصی برای توصیف ورودی‌ها، خروجی‌ها و نحوه استفاده از تابع یا کلاس استفاده کنید.

****۱۰.۸ نتیجه‌گیری:****

کامنت‌ها و توضیحات مستند در پایتون به شما اجازه می‌دهند تا توضیحاتی را درباره عملکرد کد، توابع، کلاس‌ها و ماژول‌هایتان ارائه دهید. این توضیحات مفیدی برای افزایش خوانایی و قابلیت فهم کدتان هستند و به انسان‌ها کمک می‌کنند تا به راحتی کد شما را تفسیر کنند.

فصل ۱۱: اعمال جبری پیشرفته در پایتون (توان، تقسیم صحیح و باقیمانده)

۱۱.۱ توان:

در پایتون، برای محاسبه توان از عملگر `**` استفاده می‌شود. این عملگر به شما امکان می‌دهد یک عدد را به توان یک عدد دیگر ارتفاع دهید.

۱۱.۲ تقسیم صحیح:

عملگر `//` برای انجام تقسیم صحیح بین دو عدد استفاده می‌شود. نتیجه این تقسیم به صورت کامل و بدون مقدار باقیمانده نمایش داده می‌شود.

۱۱.۳ باقیمانده:

عملگر `%` برای محاسبه باقیمانده تقسیم دو عدد به هم استفاده می‌شود. نتیجه این عملگر نشان می‌دهد که باقیمانده تقسیم دو عدد چقدر است.

۱۱.۴ نمونه‌های اعمال جبری پیشرفته:

```
# توان
power_result = 2 ** 3  # نتیجه: ۸

# تقسیم صحیح
integer_division_result = 17 // 3  # نتیجه: ۵

# باقیمانده
remainder_result = 17 % 3  # نتیجه: ۲
```

۱۱.۵ نکته‌های مربوط به اعمال جبری پیشرفته:

– برای محاسبه توان از عملگر `**` استفاده کنید، به عنوان مثال: `۲ ** ۳` معادل ۲ به توان ۳ است.

– تقسیم صحیح با استفاده از `//` نتیجه تقسیم دو عدد را به صورت صحیح نشان می‌دهد.

– باقیمانده تقسیم دو عدد با استفاده از `%` نشان می‌دهد که چقدر از تقسیم دو عدد باقی می‌ماند.

****۱۱.۶ نتیجه‌گیری:****

اعمال جبری پیشرفته مانند توان، تقسیم صحیح و باقیمانده در پایتون به شما امکان می‌دهند که محاسبات پیچیده‌تر را انجام داده و به تجزیه و تحلیل دقیق‌تر اعداد بپردازید. این عملگرها به شما ابزارهای قدرتمندی را برای انجام عملیات‌های پیچیده فراهم می‌کنند.

فصل ۱۲: معرفی انواع داده‌های پیشرفته در پایتون

۱۲.۱ لیست (List):

لیست‌ها مجموعه‌ای از اشیاء هستند که می‌توانند داده‌های مختلف از جمله اعداد، رشته‌ها و حتی لیست‌های دیگر را در خود نگه دارند. لیست‌ها با استفاده از `[]` تعریف می‌شوند.

```
numbers = [1, 2, 3, 4, 5]
names = ["Alice", "Bob", "Charlie"]
mixed_list = [1, "Hello", True]
```

۱۲.۲ بولی (Boolean):

بولی یک نوع داده است که دو مقدار True و False را می‌پذیرد. این نوع داده برای انجام مقایسه‌ها و شرط‌ها در برنامه‌نویسی بسیار مهم است.

```
is_active = True
is_authenticated = False
```

۱۲.۳ دیکشنری (Dictionary):

دیکشنری‌ها مجموعه‌ای از جفت‌های کلید–مقدار هستند که به شما امکان می‌دهند تا داده‌ها را با استفاده از کلیدها مدیریت کنید. دیکشنری‌ها با استفاده از `{}` تعریف می‌شوند.

```
person = {
    "name": "John",
    "age": 30,
    "is_student": False
}
```

۱۲.۴ مقدار None:

مقدار None به معنای عدم وجود داده یا مقدار تهی است. این مقدار معمولاً برای نمایش عدم وجود مقداری در یک متغیر استفاده می‌شود.

```
result = None
```

۱۲.۵ نکته‌های مربوط به انواع داده‌های پیشرفته:

– لیست‌ها، دیکشنری‌ها و دیگر نوع‌های داده‌های پیشرفته به شما امکان مدیریت و سازماندهی داده‌های پیچیده‌تر را می‌دهند.

– بولی معمولاً برای مقایسه‌ها و شرط‌ها در برنامه‌نویسی استفاده می‌شود.

– مقدار None به عنوان نماینده‌ای از مقدار تهی یا عدم وجود داده مفید است.

۱۲.۶ نتیجه‌گیری:

انواع داده‌های پیشرفته از جمله لیست‌ها، بولی‌ها، دیکشنری‌ها و None در پایتون به شما امکان می‌دهند تا با داده‌های پیچیده‌تر و موارد ویژه‌تر کار کنید و برنامه‌های کاملاً پویا و قابل تنظیم ایجاد کنید.

فصل ۱۳: رشته‌ها (Strings) در پایتون

۱۳.۱ معرفی رشته‌ها:

رشته‌ها یک نوع داده اساسی در پایتون هستند که برای ذخیره و کار با متن و مجموعه‌ای از کاراکترها استفاده می‌شوند.

۱۳.۲ ایجاد رشته:

شما می‌توانید رشته‌ها را با استفاده از علامت تکی یا دوتایی نقل قول ایجاد کنید:

```
single_quoted = 'Hello, world!'
double_quoted = "Python Programming"
```

۱۳.۳ کاراکترهای اختصاصی (Escape Characters):

برای اضافه کردن کاراکترهای خاص مانند خط جدید یا دابل کوتیشن داخل رشته‌ها، از کاراکترهای اختصاصی استفاده می‌شود. برخی از این کاراکترها عبارتند از `\n` برای خط جدید و `\"` برای دابل کوتیشن.

۱۳.۴ ترکیب رشته‌ها (String Concatenation):

شما می‌توانید دو یا چند رشته را با استفاده از عملگر `+` به هم اضافه کنید:

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
```

۱۳.۵ تکرار رشته (String Repetition):

با استفاده از عملگر `*` می‌توانید یک رشته را تکرار کنید:

```
stars = "*" * 10  # نتیجه: **********
```

۱۳.۶ دسترسی به کاراکترها (Indexing):

رشته‌ها به صورت اندیسی که از ۰ شروع می‌شود، اندیس‌گذاری می‌شوند. شما می‌توانید به کاراکترهای مشخصی با استفاده از اندیس دسترسی پیدا کنید:

```
message = "Hello"
first_char = message[0]  # نتیجه: 'H'
```

۱۳.۷ ترکیب متغیرها با رشته‌ها (String Formatting):

شما می‌توانید متغیرها را درون یک رشته ترکیب کنید با استفاده از فرمت‌بندی رشته:

```
name = "Alice"
age = 25
message = "My name is {} and I am {} years
old.".format(name, age)
```

۱۳.۸ تعداد کاراکترها و طول رشته (String Length):

با استفاده از تابع `len()` می‌توانید تعداد کاراکترها و طول یک رشته را به دست آورید:

```
text = "Hello, World!"
length = len(text)  # نتیجه: ۱۳
```

۱۳.۹ تکه‌های زیررشته (Substring):

با استفاده از اندیس‌ها می‌توانید تکه‌هایی از یک رشته را بازیابی کنید:

```python
message = "Python Programming"
substring = message[7:15]  # نتیجه: "Programming"
```

۱۳.۱۰ **نکته‌های مربوط به رشته‌ها:**

– رشته‌ها در پایتون اشیاء غیرقابل تغییر (immutable) هستند، یعنی پس از ایجاد، نمی‌توانید مقادیر داخل رشته را تغییر دهید.

– ترکیب و تکرار رشته‌ها باعث ایجاد رشته

ای جدید می‌شود.

– اندیس‌ها به صورت مثبت و منفی (شروع از انتهای رشته) قابل استفاده هستند.

– رشته‌ها را می‌توانید با توابع متداولی همچون `upper()` و `lower()` به بزرگی یا کوچکی تبدیل کنید.

۱۳.۱۱ **نتیجه‌گیری:**

رشته‌ها در پایتون اساسی‌ترین انواع داده برای نمایش متن هستند. با استفاده از اندیس‌گذاری، ترکیب، تکرار و توابع رشته‌ها، می‌توانید متن‌های مختلف را مدیریت و تغییر دهید. رشته‌ها در برنامه‌نویسی بسیار حیاتی هستند و در بسیاری از وظایف مختلف استفاده می‌شوند.

فصل ۱۴: تعبیر رشته (String Interpolation) با استفاده از f""

۱۴.۱ مقدمه‌ای درباره تعبیر رشته:

تعبیر رشته یا String Interpolation یک روش قدرتمند در پایتون برای ترکیب مقادیر متغیرها و عبارات با رشته‌ها است. این روش از ویژگی‌هایی مانند f-strings (رشته‌های f) برای ایجاد رشته‌ها با مقادیر متغیرها استفاده می‌کند.

۱۴.۲ استفاده از رشته‌های f:

در رشته‌های f، شما می‌توانید متغیرها و عبارات را با داخل کردن آنها داخل داخل زیررشته‌ها {} ترکیب کنید:

```
name = "Alice"
age = 30
message = f"My name is {name} and I am {age} years old."
```

۱۴.۳ استفاده از عبارات داخل رشته‌های f:

علاوه بر قرار دادن متغیرها، می‌توانید عبارات ریاضی و محاسباتی داخل زیررشته‌های {} قرار دهید:

```
x = 5
y = 10
result = f"The sum of {x} and {y} is {x + y}."
```

۱۴.۴ ترتیب و دقت در تعبیر رشته‌ها:

در رشته‌های f، ترتیب تعبیرها مهم است. ابتدا رشته ترجمه می‌شود و سپس مقادیر متغیرها درج می‌شوند. همچنین دقت کنید که نوع‌های مختلف متغیرها تبدیل و تطابق خواهند یافت.

۱۴.۵ نکته‌های مربوط به تعبیر رشته‌ها:**

– استفاده از f-strings **برای تعبیر رشته‌ها در پایتون به شما امکان می‌دهد تا به راحتی مقادیر متغیرها و عبارات را داخل رشته‌ها ترکیب کنید.**

– شما می‌توانید عبارات ریاضی و محاسباتی پیچیده را داخل زیررشته‌ها {} **قرار دهید.**

– ترتیب تعبیرها و دقت در نوع‌های متغیرها در استفاده از تعبیر رشته بسیار مهم است.

۱۴.۶ نتیجه‌گیری:**

تعبیر رشته با استفاده از f-strings **یک ابزار قدرتمند در پایتون است که به شما اجازه می‌دهد تا به راحتی مقادیر متغیرها و عبارات را با رشته‌ها ترکیب کنید. این ویژگی به خوانایی و قابلیت فهم بیشتر کد شما کمک می‌کند.**

فصل ۱۵: آشنایی اولیه با اندیس‌ها در رشته‌ها

۱۵.۱ مقدمه‌ای درباره اندیس‌ها:

اندیس‌ها در رشته‌ها به شما امکان می‌دهند تا به کاراکترهای مختلف داخل یک رشته دسترسی پیدا کنید. اندیس‌ها از ۰ شروع می‌شوند.

۱۵.۲ دسترسی به کاراکترها با اندیس‌ها:

با استفاده از اندیس‌ها می‌توانید به کاراکترهای مختلف داخل یک رشته دسترسی پیدا کنید:

```
text = "Hello, World!"
first_char = text[0]  # نتیجه: 'H'
second_char = text[1]  # نتیجه: 'e'
```

۱۵.۳ استفاده از اندیس‌های منفی:

شما می‌توانید از اندیس‌های منفی نیز برای دسترسی به کاراکترها از انتهای رشته به سمت ابتدا استفاده کنید:

```
last_char = text[-1]  # نتیجه: '!'
second_last_char = text[-2]  # نتیجه: 'd'
```

۱۵.۴ ترکیب اندیس‌ها برای دسترسی به تکه‌های رشته:

شما می‌توانید با استفاده از اندیس‌ها تکه‌های مختلفی از یک رشته را دسترسی پیدا کنید:

```
substring = text[7:12]  # نتیجه: 'World'
```

۱۵.۵ نکته‌های مربوط به اندیس‌ها در رشته‌ها:**

– اندیس‌ها از ۰ شروع می‌شوند، بنابراین اندیس اولیه رشته ۰ است.

– اندیس‌های منفی از آخرین کاراکتر شروع می‌شوند (–۱).

– با استفاده از اندیس‌ها می‌توانید به تکه‌های مختلف رشته دسترسی پیدا کنید.

۱۵.۶ نتیجه‌گیری:**

آشنایی با اندیس‌ها در رشته‌ها امکان دسترسی به کاراکترها و تکه‌های مختلف رشته را به شما می‌دهد. اندیس‌ها یک ابزار مهم در کار با متن در پایتون هستند و به شما امکان می‌دهند تا به صورت دقیق بر روی کاراکترها عملیات انجام دهید.

فصل ۱۶: تبدیل داده‌ها از یک نوع به نوع دیگر (Type Conversion)

۱۶.۱ معرفی تبدیل داده‌ها:**

در برنامه‌نویسی، گاهی اوقات نیاز دارید تا داده‌ها را از یک نوع به نوع دیگر تبدیل کنید. این عملیات به نام تبدیل داده‌ها یا Type Conversion **شناخته می‌شود.**

۱۶.۲ تبدیل به اعداد صحیح (Integer):**

با استفاده از تابع `int()` می‌توانید داده‌ها را به اعداد صحیح تبدیل کنید:

```
number = int("5")  # نتیجه: ۵
```

۱۶.۳ تبدیل به اعداد اعشاری (Float):**

با استفاده از تابع `float()` می‌توانید داده‌ها را به اعداد اعشاری تبدیل کنید:

```
decimal = float("3.14")  # نتیجه: ۳.۱۴
```

۱۶.۴ تبدیل به رشته (String):**

با استفاده از تابع `str()` می‌توانید داده‌ها را به رشته تبدیل کنید:

```
text = str(123)  # نتیجه: "۱۲۳"
```

۱۶.۵ نکته‌های مربوط به تبدیل داده‌ها:

– توجه داشته باشید که تبدیل داده‌ها ممکن است منجر به اطلاعات گمشده شود. مثلاً تبدیل عدد اعشاری به عدد صحیح تقریباً معنی ندارد.

– توجه کنید که تبدیل به رشته می‌تواند مخصوصاً در ترکیب با رشته‌های تعبیری مفید باشد.

۱۶.۶ نتیجه‌گیری:

تبدیل داده‌ها از یک نوع به نوع دیگر یک عملیات مهم در برنامه‌نویسی است. این عملیات به شما اجازه می‌دهد تا داده‌ها را بر اساس نیازهای خود به نوع‌های مختلف تبدیل کنید و از آنها در بخش‌های مختلف برنامه استفاده کنید.

فصل ۱۷: دستور ورودی (input) در پایتون

۱۷.۱ معرفی دستور input:

دستور `input` به شما امکان می‌دهد تا از کاربر ورودی دریافت کنید. این ورودی معمولاً توسط کاربر از صفحه کلید وارد می‌شود.

۱۷.۲ استفاده از دستور input:

برای استفاده از دستور `input`، کافیست آن را صدا بزنید و مقدار وارد شده توسط کاربر را دریافت کنید:

```
user_input = input("Please enter your name: ")
```

۱۷.۳ تبدیل ورودی به دیگر انواع داده:

وقتی کاربر ورودی را وارد می‌کند، این ورودی به عنوان یک رشته دریافت می‌شود. شما می‌توانید این رشته را به دیگر انواع داده تبدیل کنید:

```
age = input("Please enter your age: ")
age_int = int(age)  # تبدیل به عدد صحیح
```

۱۷.۴ نکته‌های مربوط به دستور input:

– دستور `input` باعث متوقف شدن اجرای برنامه می‌شود تا کاربر ورودی را وارد کند.

– ورودی به عنوان یک رشته دریافت می‌شود و برای استفاده‌های مختلف نیاز به تبدیل داده ممکن است داشته باشید.

– توجه داشته باشید که دستور `input` در برنامه‌های واقعی معمولاً با دستورات شرطی (مثل `if`) برای بررسی ورودی کاربر استفاده می‌شود.

۱۷.۵ نتیجه‌گیری:

استفاده از دستور `input` به شما امکان می‌دهد تا از کاربر ورودی دریافت کنید و از آن در برنامه‌های خود استفاده کنید. با توجه به این ورودی‌ها، می‌توانید برنامه‌های پویا و تعاملی طراحی کنید که با کاربر ارتباط دارند.

فصل ۱۸: دستور گرد کردن اعداد (round) در پایتون

۱۸.۱ معرفی دستور round:

دستور `round` به شما امکان می‌دهد اعداد اعشاری را به عدد صحیح گرد کنید یا تا تعداد معینی از اعشار را نمایش دهید.

۱۸.۲ گرد کردن به عدد صحیح:

با استفاده از دستور `round` می‌توانید یک عدد اعشاری را به نزدیک‌ترین عدد صحیح گرد کنید:

```
result = round(3.6)  # نتیجه: ٤
```

۱۸.۳ گرد کردن با تعیین تعداد اعشار:

شما می‌توانید با استفاده از دستور `round` عدد اعشاری را به تعداد مشخصی از اعشار گرد کنید:

```
pi = 3.14159265
rounded_pi = round(pi, 2)  # نتیجه: ٣,١٤
```

۱۸.۴ نکته‌های مربوط به دستور round:

– دستور `round` به شما اجازه می‌دهد اعداد اعشاری را به صورت گرد شده نمایش دهید یا به تعداد مشخصی از اعشار تغییر دهید.

– توجه داشته باشید که گرد کردن اعداد ممکن است منجر به تغییر ناخواسته مقدار دقیق اعداد شود.

۱۸.۵ نتیجه‌گیری:

استفاده از دستور `round` در پایتون به شما امکان می‌دهد اعداد اعشاری را به صورت گرد شده یا با تعداد معینی از اعشار نمایش دهید. این دستور می‌تواند در مواردی که نیاز به نمایش اعداد به شکل مناسب دارید بسیار مفید باشد.

فصل ۱۹: پروژه – تبدیل کیلومتر به مایل با استفاده از پایتون

۱۹.۱ مقدمه به پروژه:

در این پروژه، ما قصد داریم یک برنامه ساده با پایتون بنویسیم که کاربر از ورودی خود کیلومتر را دریافت کند و آن را به مایل تبدیل کند. همچنین می‌خواهیم نتیجه تبدیل را با دقت دو اعشاری نمایش دهیم.

۱۹.۲ ایجاد برنامه:

ابتدا از کاربر ورودی کیلومتر را دریافت می‌کنیم و سپس آن را به مایل تبدیل می‌کنیم. برای تبدیل از فرمول زیر استفاده می‌کنیم:

```
مایل= کیلومتر* 0.621371
```

۱۹.۳ پیاده‌سازی کد:

```
# دریافت ورودی از کاربر
kilometers = float(input("لطفا مسافت به کیلومتر وارد کنید: "))

# تبدیل کیلومتر به مایل
```

```
conversion_factor = 0.621371
miles = kilometers * conversion_factor

# نمایش نتیجه با دقت دو اعشاری
formatted_miles = round(miles, 2)
print(f"{kilometers} کیلومتر معادل {formatted_miles} مایل می‌شود.")
```

 # تبدیل کیلومتر به مایل

conversion_factor = ۰٫۶۲۱۳۷۱

miles = kilometers * conversion_factor

 # نمایش نتیجه با دقت دو اعشاری

formatted_miles = round(miles, ۲)

print(f"{kilometers} **کیلومتر معادل** {formatted_miles} **مایل می‌شود.")**

```

**\*\*۱۹.۴ نتیجه‌گیری:\*\***

**این پروژه به شما نشان می‌دهد چگونه با استفاده از پایتون یک برنامه ساده برای تبدیل واحدها ایجاد کنید. با تبدیل کیلومتر به مایل به عنوان مثال، می‌توانید مفهوم تبدیل واحدها و کار با عملیات ریاضی در پایتون را تجربه کنید.**
```

فصل ۲۰: گذاره‌های شرطی (Conditional Statements) به صورت ساده

۲۰.۱ معرفی گذاره‌های شرطی:

گذاره‌های شرطی در برنامه‌نویسی به شما امکان می‌دهند تا بسته به شرایط مختلف، قطعه‌ای از کد را اجرا یا نادیده بگیرید. این به شما امکان می‌دهد که برنامه‌هایی ایجاد کنید که به صورت پویا و تعاملی عمل می‌کنند.

۲۰.۲ استفاده از گذاره‌های if:

گذاره‌های `if` به شما اجازه می‌دهند تا یک بلوک کد را اجرا کنید اگر یک شرط مشخص برقرار باشد:

```
age = 18
if age >= 18:
    print("شما قادر به رای دادن هستید.")
```

۲۰.۳ استفاده از گذاره‌های if-else:

گذاره‌های `if-else` به شما اجازه می‌دهند تا بین دو بلوک کد انتخاب کنید. اگر شرط اول برقرار باشد، بلوک `if` اجرا می‌شود و در غیر این صورت بلوک `else` اجرا می‌شود:

```
temperature = 25
if temperature > 30:
    print("هوا گرم است.")
else:
    print("هوا خنک است.")
```

۲۰.۴ استفاده از گذاره‌های if-elif-else:

گذاره‌های `if-elif-else` به شما اجازه می‌دهند تا بین چندین شرط مختلف انتخاب کنید. اگر شرط اول برقرار نباشد، به شرط دوم (`elif`) می‌روید و در غیر این صورت بلوک `else` اجرا می‌شود:

```
score = 75
if score >= 90:
    print("عالی")
elif score >= 70:
    print("خوب")
else:
    print("ضعیف")
```

۲۰.۵ نکته‌های مربوط به گذاره‌های شرطی:

– گذاره‌های شرطی به شما امکان می‌دهند برنامه را بر اساس شرایط مختلف تنظیم کنید.

– می‌توانید از گذاره‌های if، if-else و if-elif-else برای ایجاد ساختار تصمیم‌گیری در برنامه‌های خود استفاده کنید.

– توجه داشته باشید که در هر گذاره تنها یک بلوک کد اجرا می‌شود (به شرطی که شرط برقرار باشد).

۲۰.۶ نتیجه‌گیری:

گذاره‌های شرطی در برنامه‌نویسی ابزار مهمی هستند که به شما امکان می‌دهند تا بر اساس شرایط مختلف برنامه‌های خود را تنظیم کنید. با استفاده از این گذاره‌ها می‌توانید برنامه‌های پویا و تعاملی ایجاد کنید.

فصل ۲۱: آشنایی با اپراتورهای مقایسه در پایتون

۲۱.۱ معرفی اپراتورهای مقایسه:

اپراتورهای مقایسه در پایتون برای مقایسه دو مقدار به کار می‌روند و نتیجه مقایسه را به صورت منطقی برمی‌گردانند.

۲۱.۲ اپراتور مساوی (==):

اپراتور `==` برای بررسی مساوی بودن دو مقدار به کار می‌رود:

```
x = 5
y = 7
result = x == y  # نتیجه: False
```

۲۱.۳ اپراتور نامساوی (!=):

اپراتور `!=` برای بررسی نامساوی بودن دو مقدار به کار می‌رود:

```
a = 10
b = 10
result = a != b  # نتیجه: False
```

۲۱.۴ اپراتور بزرگترین (>):

اپراتور `>` برای بررسی اینکه یک مقدار بزرگتر از مقدار دیگری است، به کار می‌رود:

```
num1 = 15
num2 = 10
result = num1 > num2  # نتیجه: True
```

۲۱.۵ اپراتور کوچکترین (<):

اپراتور `<` برای بررسی اینکه یک مقدار کوچکتر از مقدار دیگری است، به کار می‌رود:

```
value1 = 5
value2 = 8
result = value1 < value2  # نتیجه: True
```

۲۱.۶ اپراتور بزرگتر یا مساوی (>=) و کوچکتر یا مساوی (<=):

اپراتورهای `>=` و `<=` برای بررسی بزرگتر یا مساوی بودن یا کوچکتر یا مساوی بودن دو مقدار به کار می‌روند:

```
number1 = 7
number2 = 7
result1 = number1 >= number2  # نتیجه: True
result2 = number1 <= number2  # نتیجه: True
```

۲۱.۷ نکته‌های مربوط به اپراتورهای مقایسه:

– اپراتورهای مقایسه نتایج منطقی (True یا False) برمی‌گردانند.

– این اپراتورها برای مقایسه انواع داده‌های مختلف از جمله اعداد صحیح، اعشاری، رشته‌ها و بیشتر به کار می‌روند.

– از این اپراتورها به عنوان بخشی از گذاره‌های شرطی نیز استفاده می‌شود.

۲۱.۸ نتیجه‌گیری:

اپراتورهای مقایسه در پایتون به شما امکان می‌دهند تا دو مقدار را با یکدیگر مقایسه کرده و نتیجه مقایسه را به صورت منطقی دریافت کنید. این اپراتورها به شما امکان می‌دهند تا شرایط مختلفی را در برنامه‌های خود ایجاد کنید و عملیات تصمیم‌گیری را پیاده‌سازی کنید.

فصل ۲۲: داده‌ی بولین در پایتون و استفاده از آن در گذاره‌های شرطی

۲۲.۱ معرفی داده‌ی بولین:

داده‌ی بولین در پایتون دو مقدار ممکن دارد: True **یا** False**. این داده‌ها به شما امکان می‌دهند تا وضعیت منطقی یک شرط را نمایش دهید.**

۲۲.۲ استفاده از داده‌ی بولین در اپراتورهای مقایسه:

اپراتورهای مقایسه به عنوان نتیجه‌ای از مقایسه دو مقدار، مقدار بولین تولید می‌کنند. به عبارت دیگر، نتیجه مقایسه یک اظهارنامه منطقی است:

```
x = 5
y = 7
result = x < y  # نتیجه: True
```

۲۲.۳ استفاده از داده‌ی بولین در گذاره‌های شرطی (if):

در گذاره‌های شرطی، از داده‌ی بولین برای تصمیم‌گیری بر اساس یک شرط استفاده می‌شود. اگر شرط برقرار باشد (True)، بلوک کد مرتبط با `if` اجرا می‌شود، در غیر این صورت نادیده گرفته می‌شود:

```
age = 16
if age >= 18:
    print("شما مجاز به رای دادن هستید.")
else:
    print("شما مجاز به رای دادن نیستید.")
```

۲۲.۴ ماهیت غلظ در برخی اشاره‌ها به داده‌های بولین:

در برخی موارد، اشاره‌ها به داده‌های بولین می‌توانند غلظ و کارآمد باشند. به عنوان مثال، برای مقایسه دو رشته، می‌توانیم از داده‌های بولین برای بررسی تطابق استفاده کنیم:

```
name = "Alice"
is_name_alice = name == "Alice"  # نتیجه: True
```

۲۲.۵ نتیجه‌گیری:

داده‌ی بولین در پایتون با ارزش True یا False به شما امکان می‌دهد تا شرایط منطقی را در برنامه‌های خود مدیریت کنید. این داده‌ها به شما امکان می‌دهند تا عملیات‌های تصمیم‌گیری منطقی انجام دهید و بر اساس آنها اجرای بخش‌های مختلف برنامه را کنترل کنید.

فصل ۲۳: آشنایی با اپراتورهای منطقی در پایتون

۲۳.۱ معرفی اپراتورهای منطقی:

اپراتورهای منطقی در پایتون برای ایجاد عبارات منطقی با استفاده از داده‌های بولین (True یا False) به کار می‌روند. این اپراتورها امکان ترکیب دستورات منطقی را به شما می‌دهند.

۲۳.۲ اپراتور and:

اپراتور `and` برای ترکیب دو شرط منطقی استفاده می‌شود و نتیجه آن True خواهد بود اگر هر دو شرط برقرار باشند:

```
x = 5
y = 7
result = (x < y) and (x != y)  # نتیجه: True
```

۲۳.۳ اپراتور or:

اپراتور `or` برای ترکیب دو شرط منطقی استفاده می‌شود و نتیجه آن True خواهد بود اگر حداقل یکی از شرط‌ها برقرار باشد:

```
a = 10
b = 5
result = (a > b) or (a == b)  # نتیجه: True
```

۲۳.۴ اپراتور not:

اپراتور `not` برای نقض یک شرط منطقی استفاده می‌شود، به این معنا که اگر شرط برقرار باشد، `not` آن را به False تبدیل می‌کند و برعکس:

```
value = True
result = not value  # نتیجه: False
```

۲۳.۵ ترکیب اپراتورهای منطقی:

شما می‌توانید اپراتورهای منطقی را با هم ترکیب کنید تا عبارات منطقی پیچیده‌تری بسازید:

```
x = 5
y = 10
z = 3
result = (x < y) and (y > z) and not (x == z)  # نتیجه: True
```

۲۳.۶ نکته‌های مربوط به اپراتورهای منطقی:

– اپراتورهای منطقی می‌توانند با هم ترکیب شوند تا شرایط منطقی پیچیده‌تری را بسازند.

– ترتیب اجرای اپراتورها مشابه قوانین ریاضی است: ابتدا `not`، سپس `and` و در آخر `or` اجرا می‌شوند.

– از این اپراتورها معمولاً در گذاره‌های شرطی (if) برای ایجاد شرایط منطقی استفاده می‌شود.

****۲۳.۷ نتیجه‌گیری:****

اپراتورهای منطقی در پایتون به شما امکان می‌دهند تا شرایط منطقی پیچیده‌تری را در برنامه‌های خود ایجاد کنید. این اپراتورها به شما امکان می‌دهند تا اظهارنامه‌های منطقی را ترکیب و اجرای شرایط مختلف را کنترل کنید.

فصل ۲۴: پروژه – بازی سنگ کاغذ قیچی با استفاده از دانش‌های یادگرفته شده

****۲۴.۱ مقدمه به پروژه:****

در این پروژه، می‌خواهیم یک بازی ساده سنگ کاغذ قیچی بنویسیم که کاربر با کامپیوتر به مبارزه بپردازد. در این پروژه، ما از دانش‌هایی که تا الان یاد گرفته‌اید، استفاده می‌کنیم تا این بازی را بسازیم.

۲۴.۲ ایجاد بازی:

ابتدا کاربر ورودی خود را (سنگ، کاغذ یا قیچی) انتخاب می‌کند و سپس کامپیوتر به صورت تصادفی یکی از این گزینه‌ها را انتخاب می‌کند. سپس بر اساس قوانین بازی، نتیجه بازی مشخص می‌شود.

۲۴.۳ پیاده‌سازی کد:

```
import random

# انتخاب تصادفی گزینه توسط کامپیوتر
options = ["سنگ", "کاغذ", "قیچی"]
computer_choice = random.choice(options)

# دریافت گزینه از کاربر
user_choice = input("لطفاً یکی از گزینه‌های 'سنگ'، 'کاغذ' یا 'قیچی' را وارد کنید: ")

# نمایش انتخاب کامپیوتر و کاربر
print(f"شما: {user_choice}")
print(f"کامپیوتر: {computer_choice}")

# تعیین نتیجه بازی
if user_choice == computer_choice:
    result = "مساوی"
elif (user_choice == "سنگ" and computer_choice == "قیچی") or \
     (user_choice == "کاغذ" and computer_choice == "سنگ") or \
     (user_choice == "قیچی" and computer_choice == "کاغذ"):
    result = "شما برنده شدید!"
```

else:

result = "**کامپیوتر برنده شد.**"

 # نمایش نتیجه بازی

print(f"**نتیجه:** {result}")

۲۴.۴ نتیجه‌گیری:

این پروژه نشان می‌دهد که چگونه با استفاده از دانش‌های پایه‌ای یادگرفته‌شده، یک بازی ساده را ایجاد کنید. این پروژه به شما امکان می‌دهد تا تجربه ایجاد برنامه‌های تعاملی و مبارزه را تجربه کنید.

فصل ۲۵: پروژه – بازی دو نفره سنگ کاغذ قیچی با استفاده از دانش‌های یادگرفته شده

۲۵.۱ مقدمه به پروژه:

در این پروژه، می‌خواهیم بازی سنگ کاغذ قیچی را به گونه‌ای ارتقا دهیم که دو بازیکن بتوانند به صورت نوبتی در مقابل یکدیگر بازی کنند. این پروژه از دانش‌هایی که تا الان یاد گرفته‌اید، استفاده می‌کند.

۲۵.۲ ایجاد بازی دو نفره:

در این نسخه، دو بازیکن می‌توانند به تناوب گزینه‌های خود را (سنگ، کاغذ یا قیچی) انتخاب کنند و نتیجه بازی بر اساس گزینه‌های آن‌ها تعیین می‌شود.

۲۵.۳ پیاده‌سازی کد:

```
# تابع برای تعیین برنده بر اساس انتخاب دو بازیکن
def determine_winner(player1_choice, player2_choice):
    if player1_choice == player2_choice:
        return "مساوی"
    elif (player1_choice == "سنگ" and player2_choice == "قیچی") or \
         (player1_choice == "کاغذ" and player2_choice == "سنگ") or \
         (player1_choice == "قیچی" and player2_choice == "کاغذ"):
        return "بازیکن ۱ برنده شد!"
    else:
        return "بازیکن ۲ برنده شد!"

# بازی دو نفره
print("بازی دو نفره سنگ کاغذ قیچی")
print("-----------------------")

# ورود گزینه‌های بازیکن‌ها
player1_choice = input("بازیکن ۱، لطفاً یکی از گزینه‌های 'سنگ'، 'کاغذ' یا 'قیچی' را وارد کنید: ")
player2_choice = input("بازیکن ۲، لطفاً یکی از گزینه‌های 'سنگ'، 'کاغذ' یا 'قیچی' را وارد کنید: ")

# تعیین برنده
result = determine_winner(player1_choice, player2_choice)

# نمایش نتیجه بازی
print(f"نتیجه: {result}")
```

۲۵.۳ پیاده‌سازی کد:

```
# بازی دو نفره
print("بازی دو نفره سنگ کاغذ قیچی")
print("-----------------------")

# ورود گزینه‌های بازیکن‌ها
player1_choice = input("بازیکن ۱، لطفاً یکی از گزینه‌های 'سنگ'، 'کاغذ' یا 'قیچی' را وارد کنید: ")
player2_choice = input("بازیکن ۲، لطفاً یکی از گزینه‌های 'سنگ'، 'کاغذ' یا 'قیچی' را وارد کنید: ")

# تعیین نتیجه بازی
if player1_choice == player2_choice:
    result = "مساوی"
elif (player1_choice == "سنگ" and player2_choice == "قیچی") or \
     (player1_choice == "کاغذ" and player2_choice == "سنگ") or \
     (player1_choice == "قیچی" and player2_choice == "کاغذ"):
    result = "بازیکن ۱ برنده شد!"
else:
    result = "بازیکن ۲ برنده شد."

# نمایش نتیجه بازی
print(f"نتیجه: {result}")
```

۲۵.۴ نتیجه‌گیری:

این پروژه نشان می‌دهد که چگونه با استفاده از دانش‌های یادگرفته‌شده، بازی ساده‌ای را به گونه‌ای ارتقا دهیم که دو بازیکن بتوانند در مقابل یکدیگر بازی کنند. این پروژه به شما امکان می‌دهد تا تجربه ایجاد برنامه‌های تعاملی و چالش‌های جدیدتر را تجربه کنید.

فصل ۲۶: انواع حلقه‌ها در پایتون

۲۶.۱ حلقه while:

حلقه‌ی while **تا زمانی که شرط مشخص برقرار باشد، بلاواسطه دستورات داخل حلقه را اجرا می‌کند.**

```
while شرط:
    # دستوراتی که باید تکرار شوند
```

۲۶.۲ حلقه for:

حلقه‌ی for **برای اجرای یک دسته دستورات بر روی یک ترتیبی از مقادیر (مانند لیست یا رشته) استفاده می‌شود.**

```
for متغیر in ترتیب:
    # دستوراتی که باید بر روی هر مقدار اجرا شوند
```

۲۶.۳ حلقه for با range:

حلقه‌ی for **معمولاً با تابع** `range()` **برای ایجاد ترتیبی از اعداد صحیح استفاده می‌شود.**

```
for متغیر in range(شروع, پایان, گام):
    # دستوراتی که باید بر روی هر مقدار اجرا شوند
```

۲۶.۴ حلقه for در ترتیب‌های دیگر:

حلقه‌ی for **می‌تواند بر روی ترتیب‌های دیگری مانند لیست‌ها، رشته‌ها، تاپل‌ها و دیکشنری‌ها نیز اجرا شود.**

```
for مقدار in ترتیب:
    # دستوراتی که باید بر روی هر مقدار اجرا شوند
```

۲۶.۵ حلقه‌ی تودرتو (Nested Loop):

می‌توانید یک حلقه را داخل حلقه‌ی دیگری قرار دهید. این نوع حلقه‌ها را حلقه‌های تودرتو یا تعاملی می‌نامند.

۲۶.۶ توقف حلقه با break **و ادامه حلقه با** continue**:**

دستور `break` برای ترک حلقه به صورت کامل و دستور `continue` برای رفتن به تکرار بعدی در حلقه‌ی تکرار استفاده می‌شود.

۲۶.۷ نتیجه‌گیری:

حلقه‌ها ابزارهای قدرتمندی هستند که به شما امکان تکرار دستورات را بر اساس شرایط مختلف می‌دهند. حلقه‌ی while برای تکرار تا زمانی که یک شرط برقرار باشد و حلقه‌ی for برای تکرار بر روی یک ترتیب مشخص از مقادیر استفاده می‌شود.

فصل ۲۷: حلقه for در پایتون

۲۷.۱ مقدمه به حلقه for:

حلقه‌ی for در پایتون برای اجرای یک دسته دستورات بر روی ترتیبی از مقادیر، مانند لیست‌ها، رشته‌ها، تاپل‌ها و دیکشنری‌ها، استفاده می‌شود. این نوع حلقه برای تکرار کردن دستورات به ازای هر مقدار در ترتیب مشخص شده به کار می‌رود.

۲۷.۲ نحوه استفاده از حلقه for:

ساختار کلی حلقه for به صورت زیر است:

```
for متغیر in ترتیب:
    # دستوراتی که بر روی هر مقدار اجرا می‌شوند
```

در این ساختار:

– متغیر: یک متغیر که در هر مرحله از حلقه مقدار ترتیبی را که در حال اجرا است، به خود می‌گیرد.

– ترتیب: ترتیبی از مقادیر که بر روی آن‌ها دستورات اجرا می‌شود.

۲۷.۳ مثال‌های استفاده از حلقه for:

استفاده از حلقه for بر روی لیست:

```
fruits = ["سیب", "موز", "پرتقال", "خیار"]
for fruit in fruits:
    print(fruit)
```

استفاده از حلقه for بر روی رشته:

```
text = "Hello, World!"
for char in text:
```

```
    print(char)
```

****استفاده از حلقه for با range:****

```
for number in range(1, 6):  # ترتیب اعداد ۱ تا ۵
    print(number)
```

****استفاده از حلقه for بر روی دیکشنری:****

```
student_scores = {"آریا": 85, "مهراد": 92, "نیکا": 78}
for name, score in student_scores.items():
    print(f"{name}: {score}")
```

****۲۷.۴ نکته‌های مربوط به حلقه for:****

– متغیر در هر مرحله از حلقه مقدار ترتیبی را که در حال اجرا است، به خود می‌گیرد.

– ترتیب می‌تواند از لیست‌ها، رشته‌ها، تاپل‌ها و دیکشنری‌ها باشد.

– ترتیب‌ها می‌توانند مقادیر از یک مجموعه یا دنباله اعداد باشند.

– می‌توانید به هر متغیر دلخواهی نام بدهید و از آن در داخل حلقه استفاده کنید.

****۲۷.۵ نتیجه‌گیری:****

حلقه for به شما امکان می‌دهد تا به راحتی از ترتیب‌های مختلف، مانند لیست‌ها، رشته‌ها، تاپل‌ها و دیکشنری‌ها، برای تکرار دستورات استفاده کنید. این حلقه ابزار بسیار قدرتمندی است که برنامه‌های شما را کوتاه‌تر و خواناتر می‌کند.

فصل ۲۸: تابع range در پایتون

۲۸.۱ مقدمه به تابع range:

تابع `range()` در پایتون برای ایجاد یک ترتیب از اعداد صحیح به کار می‌رود. این ترتیب می‌تواند برای تعیین محدوده‌ی تکرارها در حلقه‌ها و تکرارهای دیگر استفاده شود.

۲۸.۲ نحوه استفاده از تابع range:

ساختار کلی تابع `range()` به صورت زیر است:

```
range(شروع, پایان, گام)
```

در این ساختار:

– `شروع`: مقدار شروع ترتیب (شامل می‌شود).

– `پایان`: مقدار پایان ترتیب (نهایی نمی‌شود).

– `گام`: مقداری که به طور پیش‌فرض ۱ است و نشان‌دهنده‌ی افزایش مقدار در هر مرحله است.

۲۸.۳ مثال‌های استفاده از تابع range:

ترتیب اعداد به تعداد مشخص:

```
for number in range(5):
    print(number)
```

ترتیب اعداد از یک عدد شروع تا یک عدد پایان:

```
for number in range(2, 7):
    print(number)
```

ترتیب اعداد با مقدار گام:

```
for number in range(1, 10, 2):
    print(number)
```

۲۸.۴ نکته‌های مربوط به تابع range:

– تابع `range()` یک محدوده از اعداد را ایجاد می‌کند که از شروع تا قبل از پایان می‌رسد.

– در صورتی که تنها یک مقدار دهید، این مقدار به عنوان پایان ترتیب در نظر گرفته می‌شود و شروع به طور پیش‌فرض از ۰ است.

– اگر دو مقدار دهید، اولی به عنوان شروع و دومی به عنوان پایان ترتیب در نظر گرفته می‌شود.

– اگر همه سه مقدار را تعیین کنید، سومی به عنوان گام افزایش مقدار است.

– ترتیب تولید شده تا قبل از مقدار پایان می‌رسد و مقدار پایان خود در ترتیب نمایش داده نمی‌شود.

۲۸.۵ نتیجه‌گیری:

تابع `range()` به شما امکان می‌دهد تا ترتیب‌های اعداد صحیح مختلف را ایجاد کرده و از آنها در حلقه‌ها و تکرارهای دیگر استفاده کنید. این تابع ابزار مهمی برای مدیریت محدوده‌ها و تکرارها در برنامه‌های پایتون است.

فصل ۲۹: حلقه while در پایتون

۲۹.۱ مقدمه به حلقه while:

حلقه‌ی while در پایتون به شما اجازه می‌دهد تا تا زمانی که یک شرط خاص برقرار باشد، یک دسته دستورات را تکرار کنید. این حلقه معمولاً زمانی استفاده می‌شود که تعداد تکرارها پیش‌فرض نیست و به شرط خاصی وابسته است.

۲۹.۲ نحوه استفاده از حلقه while:

ساختار کلی حلقه while به صورت زیر است:

```
while شرط:
    # دستوراتی که تا زمانی که شرط برقرار باشد تکرار می‌شوند
```

۲۹.۳ مثال‌های استفاده از حلقه while:

استفاده از حلقه while برای چاپ اعداد از ۱ تا ۵:

```
number = 1
while number <= 5:
    print(number)
    number += 1
```

استفاده از حلقه while با شرط ورودی:

```python
name = ""
while name != "exit":
    name = input("نام خود را وارد کنید یا 'exit' را برای خروج وارد کنید: ")
    print(f"سلام{name}")
```

۲۹.۴ نکته‌های مربوط به حلقه while:**

– در حلقه while، ابتدا شرط بررسی می‌شود و سپس دستورات داخل حلقه اجرا می‌شوند.

– اگر شرط از ابتدا نادرست باشد، دستورات داخل حلقه هیچ‌گاه اجرا نخواهند شد.

– شما باید دستوری داخل حلقه تعیین کنید که در هر مرحله از حلقه تغییری در شرایطی ایجاد کند تا در نهایت شرط غلط شود و حلقه متوقف شود. در غیر این صورت حلقه به طور بی‌پایان ادامه خواهد داشت.

۲۹.۵ نتیجه‌گیری:**

حلقه while به شما اجازه می‌دهد تا تا زمانی که یک شرط خاص برقرار باشد، یک دسته دستورات را تکرار کنید. این حلقه برای مواقعی که تعداد تکرارها پیش‌فرض نیست و به شرط خاصی وابسته است، مناسب است.

فصل ۳۰: لیست‌ها در پایتون

۳۰.۱ مقدمه به لیست‌ها:

لیست یکی از ساختارهای داده‌ای پرکاربرد در پایتون است که به شما اجازه می‌دهد مجموعه‌ای از مقادیر را در یک متغیر ذخیره کنید. لیست‌ها می‌توانند داده‌های مختلف را شامل شوند، از جمله اعداد صحیح، اعشاری، رشته‌ها، و حتی لیست‌های دیگر.

۳۰.۲ ایجاد لیست‌ها:

برای ایجاد یک لیست در پایتون، مقادیر موردنظر را داخل یک جفت پرانتز مربع قرار دهید و آنها را با کاما جدا کنید. به عنوان مثال:

```
numbers = [1, 2, 3, 4, 5]
fruits = ["سیب", "موز", "پرتقال"]
mixed_list = [10, "خوش آمدید", 3.14, True]
empty_list = []
```

۳۰.۳ دسترسی به عناصر لیست:

برای دسترسی به عناصر مختلف یک لیست، از اندیس‌ها استفاده می‌شود. اندیس‌ها در پایتون از صفر شروع می‌شوند. بنابراین، اولین عنصر لیست در اندیس ۰ قرار دارد.

```
numbers = [10, 20, 30, 40, 50]
first_number = numbers[0]  # عنصر اول (10)
```

```
second_number = numbers[1]  # عنصر دوم (20)
third_number = numbers[2]   # عنصر سوم (30)
```

۳۰.۴ تغییر عناصر لیست:

شما می‌توانید عناصر یک لیست را بازنویسی یا تغییر دهید. به طور معمول از اندیس‌ها برای تعیین عنصر موردنظر استفاده می‌شود.

```
fruits = ["سیب", "موز", "پرتقال"]
fruits[1] = "انگور"  # تغییر میوه دوم به "انگور"
```

۳۰.۵ افزودن عناصر به لیست:

برای افزودن عناصر به یک لیست، می‌توانید از دستور `append()` استفاده کنید.

```
fruits = ["سیب", "موز", "پرتقال"]
fruits.append("انگور")  # افزودن "انگور" به انتهای لیست
```

۳۰.۶ حذف عناصر از لیست:

برای حذف عناصر از یک لیست، می‌توانید از دستور `remove()` استفاده کنید.

```
fruits = ["سیب", "موز", "پرتقال"]
fruits.remove("موز")  # حذف "موز" از لیست
```

۳۰.۷ تعداد عناصر لیست:

برای شمردن تعداد عناصر یک لیست، از تابع `len()` استفاده می‌شود.

```
numbers = [10, 20, 30, 40, 50]
count = len(numbers)  # تعداد عناصر لیست (5)
```

۳۰.۸ لوپ در لیست:

با استفاده از حلقه `for`، می‌توانید تمام عناصر یک لیست را پیمایش کنید.

```
fruits = ["سیب", "موز", "پرتقال"]
for fruit in fruits:
    print(fruit)
```

۳۰.۹ نتیجه‌گیری:

لیست‌ها در پایتون به شما امکان می‌دهند تا مجموعه‌ای از مقادیر را در یک متغیر ذخیره کنید. شما می‌توانید به عناصر لیست دسترسی داشته باشید، آنها را تغییر دهید، عناصر جدید اضافه کنید و یا عناصر موجود را حذف کنید. لیست‌ها مفیدترین ساختار داده‌ای در پایتون هستند که برای مدیریت داده‌های مختلف استفاده می‌شوند.

فصل ۳۱: متدهای اضافه کردن به لیست در پایتون

۳۱.۱ مقدمه به متدهای اضافه کردن به لیست:

در پایتون، متدهای مختلفی برای اضافه کردن عناصر به لیست‌ها وجود دارند. این متدها به شما امکان می‌دهند عناصر جدید را به لیست‌ها اضافه کرده یا عناصر موجود را با عناصر دیگر ترکیب کنید.

۳۱.۲ متد `append()`:

متد `append()` به شما اجازه می‌دهد یک عنصر جدید به انتهای لیست اضافه کنید.

```
numbers = [1, 2, 3]
numbers.append(4)  # اضافه کردن عنصر ۴ به انتهای لیست
# اکنون لیست: [1, 2, 3, 4]
```

۳۱.۳ متد `extend()`:

متد `extend()` به شما اجازه می‌دهد عناصر یک لیست دیگر را به انتهای لیست اضافه کنید.

```
fruits = ["سیب", "موز"]
new_fruits = ["پرتقال", "انگور"]
fruits.extend(new_fruits)  # اضافه کردن لیست new_fruits به انتهای لیست fruits
# اکنون لیست: ["سیب", "موز", "پرتقال", "انگور"]
```

۳۱.۴ **متد `insert()`:**

متد `insert()` به شما اجازه می‌دهد یک عنصر جدید را در مکان معینی درون لیست قرار دهید. این متد دارای دو پارامتر است: اندیس مکان موردنظر و مقدار عنصر.

```
numbers = [1, 2, 3, 4]
numbers.insert(2, 2.5)  # درج عنصر ۲٫۵ در اندیس ۲
# اکنون لیست: [1, 2, 2.5, 3, 4]
```

۳۱.۵ نکته‌های مربوط به متدهای اضافه کردن به لیست:

– متد `append()` و `extend()` عناصر را به انتهای لیست اضافه می‌کنند و اندیس‌ها تغییر نمی‌کنند.

– متد `insert()` به شما اجازه می‌دهد یک عنصر جدید را در مکان دلخواه اضافه کنید و سایر عناصر بعدی اندیس‌ها به تعداد یکی افزایش می‌یابند.

۳۱.۶ نتیجه‌گیری:

متدهای `append()`، `extend()` و `insert()` به شما امکان می‌دهند عناصر جدید را به لیست‌ها اضافه کنید. این متدها به شما امکان می‌دهند لیست‌ها را تغییر داده و عناصر را به طور موردنظر درج کنید یا به لیست‌های دیگر اضافه کنید.

فصل ۳۲: متدهای حذف عناصر از لیست در پایتون

۳۲.۱ مقدمه به متدهای حذف عناصر از لیست:

در پایتون، برای حذف عناصر از لیست‌ها متدهای مختلفی وجود دارد. این متدها به شما امکان می‌دهند عناصر مشخصی را از لیست حذف کنید.

۳۲.۲ متد `pop()`:

متد `pop()` به شما اجازه می‌دهد عنصر موجود در اندیس معینی را حذف کنید و مقدار آن را برگردانید. اگر اندیس مشخص نشود، آخرین عنصر از لیست حذف می‌شود.

```
numbers = [10, 20, 30, 40, 50]
popped_number = numbers.pop(2)  # حذف و برگرداندن عنصر در اندیس ۲ (30)
# اکنون لیست: [10, 20, 40, 50]
```

۳۲.۳ متد `remove()`:

متد `remove()` به شما اجازه می‌دهد اولین عنصر با مقدار معین را از لیست حذف کنید. اگر چندین عنصر با مقدار مشابه وجود داشته باشد، فقط اولین عنصر حذف می‌شود.

```
fruits = ["سیب", "موز", "پرتقال", "موز"]
fruits.remove("موز")  # حذف اولین عنصر با مقدار "موز"
# اکنون لیست: ["سیب", "پرتقال", "موز"]
```

۳۲.۴ متد `clear()`:

متد `clear()` به شما اجازه می‌دهد تمام عناصر لیست را حذف کنید، به طوری که لیست خالی می‌شود.

```
numbers = [1, 2, 3, 4, 5]
numbers.clear()  # حذف همه عناصر از لیست
# اکنون لیست: []
```

۳۲.۵ نکته‌های مربوط به متدهای حذف عناصر از لیست:

– متد `pop()` عنصر موجود در اندیس مشخص را حذف می‌کند و مقدار آن را برگرداند.

– متد `remove()` اولین عنصر با مقدار مشخص را حذف می‌کند.

– متد `clear()` تمام عناصر لیست را حذف کرده و لیست را خالی می‌کند.

۳۲.۶ نتیجه‌گیری:

متدهای `pop()`، `remove` و `clear()` به شما امکان می‌دهند عناصر مشخصی را از لیست‌ها حذف کنید. این متدها به شما امکان می‌دهند لیست‌ها را تغییر داده و عناصر موجود را حذف کنید.

فصل ۳۳: دیکشنری‌ها در پایتون

۳۳.۱ مقدمه به دیکشنری‌ها:

دیکشنری یکی از ساختارهای داده‌ای پرکاربرد در پایتون است که به شما اجازه می‌دهد مقادیر را با استفاده از کلیدها ذخیره کنید. در دیکشنری، هر کلید متناظر با یک مقدار قرار می‌گیرد. این ساختار به شما امکان می‌دهد به سرعت به مقادیر بر اساس کلیدها دسترسی پیدا کنید.

۳۳.۲ ایجاد دیکشنری:

برای ایجاد یک دیکشنری در پایتون، مقادیر موردنظر را با کلیدها متناظر قرار دهید و آنها را با کاما جدا کنید. از دو علامت گیومه که داخل آنها کلیدها قرار دارند استفاده کنید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}
```

۳۳.۳ دسترسی به مقادیر دیکشنری:

برای دسترسی به مقادیر دیکشنری، می‌توانید از کلید متناظر استفاده کنید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

name = student["نام"]  # مقدار متناظر با کلید "نام" (علی)
age = student["سن"]    # مقدار متناظر با کلید "سن" (25)
```

۳۳.۴ تغییر مقادیر دیکشنری:

شما می‌توانید مقادیر متناظر با کلیدها را بازنویسی یا تغییر دهید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

student["سن"] = 26      # تغییر مقدار متناظر با کلید "سن" به ۲۶
```

```
student["رشته"] = "فیزیک"   # تغییر مقدار متناظر با کلید "رشته" به "فیزیک"
```

۳۳.۵ افزودن و حذف مقادیر دیکشنری:**

شما می‌توانید مقادیر جدید را با کلیدهای جدید اضافه کنید یا مقادیر متناظر با کلیدها را حذف کنید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

student["شماره دانشجویی"] = "123456"   # افزودن کلید و مقدار جدید به دیکشنری
del student["سن"]   # حذف مقدار متناظر با کلید "سن"
```

۳۳.۶ تعداد کلیدها و مقادیر دیکشنری:**

برای شمردن تعداد کلیدها یا مقادیر دیکشنری، از تابع `len()` استفاده کنید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

num_keys = len(student)          # تعداد کلیدها (3)
num_values = len(student.values())   # تعداد مقادیر (3)
```

۳۳.۷ لوپ در دیکشنری:**

با استفاده از حلقه `for`، می‌توانید تمام کلیدها یا مقادیر دیکشنری را پیمایش کنید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}
```

```
# نمایش همه کلیدها
for key in student:
    print(key)

# نمایش همه مقادیر
for value in student.values():
    print(value)
```

۳۳.۸ **نکته‌های مربوط به دیکشنری‌ها:**

– دیکشنری‌ها از ساختارهای متغیر غیر ترتیبی هستند، بنابراین ترتیب کلیدها و مقادیر در دیکشنری مهم نیست.

– هر کلید در یک دیکشنری تنها یک بار می‌تواند وجود داشته باشد.

– مقادیر دیکشنری می‌توانند داده‌های هر نوعی از جمله اعداد، رشته‌ها، لیست‌ها و دیکشنری‌های دیگر باشند.

فصل ۳۴: متدهای کاربردی دیکشنری در پایتون

۳۴.۱ **مقدمه به متدهای دیکشنری:**

در پایتون، دیکشنری‌ها تعدادی متد مفید برای انجام عملیات مختلف روی دیکشنری‌ها دارند. این متدها به شما امکان می‌دهند عملیات مرتبط با کلیدها و مقادیر دیکشنری را انجام دهید.

۳۴.۲ متد `keys()`:

متد `keys()` به شما لیستی از تمام کلیدها در دیکشنری را بر می‌گرداند.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

keys_list = student.keys()  # بازگرداندن لیست کلیدها
print(keys_list)  # ["رشته","سن","نام"]
```

۳۴.۳ متد `values()`:

متد `values()` به شما لیستی از تمام مقادیر در دیکشنری را بر می‌گرداند.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

values_list = student.values()  # بازگرداندن لیست مقادیر
print(values_list)  # ["ریاضی",۲۵,"علی"]
```

۳۴.۴ متد `items()`:

متد `items()` به شما لیستی از تماج مرتب‌شده از تاجوج کلیدها و مقادیر در دیکشنری را بر می‌گرداند.

```
student = {
    "نام": "علی",
    "سن": 25,
```

```
    "رشته": "ریاضی"
}

items_list = student.items()  # بازگرداندن لیست تاجوج کلیدها و مقادیر
print(items_list)  # [("نام", "علی"), ("سن", ۲۵), ("رشته", "ریاضی")]
```

۳۴.۵ متد `get()`:

متد `get()` به شما امکان می‌دهد مقدار متناظر با یک کلید را بر اساس نام کلید به شکل ایمنتر و با امکان برگشت به یک مقدار پیش‌فرض در صورت عدم وجود کلید بدست آورید.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

name = student.get("نام")  # بازگرداندن مقدار متناظر با کلید "نام" (علی)
invalid_key = student.get("شهر")  # بازگرداندن None به عنوان کلید نامعتبر
default_value = student.get("مقطع", "نامشخص")  # بازگرداندن مقدار پیش‌فرض در صورت عدم وجود کلید ("نامشخص")
```

۳۴.۶ متد `pop()`:

متد `pop()` به شما امکان می‌دهد مقدار متناظر با یک کلید را حذف و برگشت دهید. اگر کلید وجود نداشته باشد، یک مقدار پیش‌فرض قابل تعیین برگشت داده می‌شود.

```python
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "ریاضی"
}

age = student.pop("سن")  # حذف و بازگرداندن مقدار متناظر با کلید "سن" (25)
invalid_key = student.pop("شهر", "نامشخص")  # بازگرداندن مقدار پیش‌فرض در صورت عدم وجود کلید ("نامشخص")
```

۳۴.۷ نتیجه‌گیری:

متدهای `keys()`، `values()`، `items()`، `get` و `pop()` به شما امکان می‌دهند اطلاعات مرتبط با کلیدها و مقادیر در دیکشنری‌ها را بازیابی، مدیریت و حذف کنید. این متدها به شما ابزارهای قدرتمندی برای کار با دیکشنری‌ها ارائه می‌دهند.

فصل ۳۵: متد `update()` در دیکشنری

۳۵.۱ **مقدمه به متد `update()` در دیکشنری:**

متد `update()` به شما اجازه می‌دهد که یک دیکشنری با مقادیر یک دیکشنری دیگر را ترکیب کنید یا مقادیر جدید را به دیکشنری اضافه کنید. این متد به شما امکان می‌دهد که دیکشنری‌ها را به صورت پویا به روز رسانی کنید.

۳۵.۲ **استفاده از متد `update()` برای افزودن مقادیر جدید:**

شما می‌توانید با استفاده از متد `update()` مقادیر جدید به یک دیکشنری اضافه کنید. اگر کلیدها در دیکشنری اصلی و دیکشنری جدید تداعی کنند، مقدارهای دیکشنری اصلی با مقادیر دیکشنری جدید به‌روز خواهند شد.

```
student = {
    "نام": "علی",
    "سن": 25
}

new_info = {
    "رشته": "ریاضی",
    "شماره دانشجویی": "123456"
}

student.update(new_info)  # افزودن مقادیر جدید به دیکشنری
# حاصل: {"نام": "علی", "سن": ۲۵, "رشته": "ریاضی", "شماره دانشجویی": "۱۲۳٤٥٦"}
```

۳۵.۳ **استفاده از متد `update()` برای به‌روزرسانی مقادیر:**

در صورتی که کلیدهای موجود در دیکشنری اصلی با کلیدهای دیکشنری جدید تداعی کنند، مقدارهای دیکشنری اصلی با مقادیر دیکشنری جدید به‌روز خواهند شد.

```
student = {
    "نام": "علی",
    "سن": 25,
    "رشته": "فیزیک"
}

new_info = {
    "سن": 26,
    "رشته": "ریاضی"
}

student.update(new_info)  # به‌روزرسانی مقادیر موجود با مقادیر جدید
# حاصل: {"نام": "علی", "سن": ۲۶, "رشته": "ریاضی"}
```

۳۵.۴ **نکته‌های مربوط به متد `update()` در دیکشنری:**

– متد `update()` به شما اجازه می‌دهد دو دیکشنری را با هم ترکیب کرده یا مقادیر جدید را به دیکشنری اصلی اضافه کنید.

– اگر کلیدها در دو دیکشنری تداعی کنند، مقادیر دیکشنری اصلی با مقادیر دیکشنری جدید به‌روز خواهند شد.

– اگر کلیدها در دیکشنری جدید تداعی نکنند، کلیدها و مقادیر مرتبط با آنها به دیکشنری اصلی اضافه می‌شوند.

۳۵.۵ **نتیجه‌گیری:**

متد `update()` به شما اجازه می‌دهد دو دیکشنری را با یکدیگر ترکیب کرده و مقادیر جدید را به دیکشنری اصلی اضافه کنید. این متد به شما امکان می‌دهد داده‌های دیکشنری را به‌روز کرده و ترتیب و مقادیر کلیدها را تغییر دهید.

فصل ۳۶: متدهای کاربردی لیست در پایتون

۳۶.۱ مقدمه به متدهای لیست:

لیست‌ها یکی از ساختارهای داده‌ای پرکاربرد در پایتون هستند. لیست‌ها به شما امکان می‌دهند مجموعه‌ای از مقادیر را با نگه‌داشتن ترتیب و ایندکس‌ها ذخیره کنید. در اینجا متدهایی که بر روی لیست‌ها قابل اعمال هستند، توضیح داده خواهند شد.

۳۶.۲ متد `append()`:

متد `append()` به شما امکان می‌دهد یک مقدار را به انتهای لیست اضافه کنید.

```
numbers = [1, 2, 3]
numbers.append(4)  # افزودن مقدار ۴ به انتهای لیست
# حاصل: [1, 2, 3, 4]
```

۳۶.۳ متد `extend()`:

متد `extend()` به شما امکان می‌دهد لیست دیگری را به انتهای لیست فعلی اضافه کنید.

```
numbers = [1, 2, 3]
more_numbers = [4, 5, 6]
numbers.extend(more_numbers)  # اضافه کردن اعضای لیست دیگر به انتهای لیست اصلی
# حاصل: [1, 2, 3, 4, 5, 6]
```

۳۶.۴ متد `insert()`:

متد `insert()` به شما امکان می‌دهد یک مقدار را در مکان مشخصی در لیست اضافه کنید. شما می‌توانید موقعیت اضافه کردن مقدار را با استفاده از ایندکس مشخص کنید.

```
numbers = [1, 2, 3]
numbers.insert(1, 5)  # اضافه کردن مقدار ۵ در اندیس ۱ (در مکان دوم) لیست
# حاصل: [1, 5, 2, 3]
```

۳۶.۵ متد `remove()`:

متد `remove()` به شما امکان می‌دهد یک مقدار مشخص را از لیست حذف کنید. این متد اولین مقدار با مقدار مشخص شده که با آن تطابق دارد، را حذف می‌کند.

```
numbers = [1, 2, 3, 2, 4]
numbers.remove(2)  # حذف اولین مقدار ۲ از لیست
# حاصل: [1, 3, 2, 4]
```

۳۶.۶ متد `pop()`:

متد `pop()` به شما امکان می‌دهد مقدار موجود در اندیس مشخصی را حذف و برگشت دهید. اگر اندیس مشخص نشود، آخرین مقدار لیست حذف و برگشت داده می‌شود.

```
numbers = [1, 2, 3]
popped_value = numbers.pop(1)  # حذف و بازگرداندن مقدار در اندیس ۱ (در مکان دوم) لیست
# حاصل: [۱, ۳], مقدار بازگشتی: ۲
```

۳۶.۷ متد `clear()`:

متد `clear()` به شما امکان می‌دهد تمام مقادیر لیست را حذف کنید و لیست را خالی کنید.

```
numbers = [1, 2, 3]
numbers.clear()  # حذف تمام مقادیر لیست (خالی کردن لیست)
# حاصل: []
```

۳۶.۸ متد `index()`:

متد `index()` به شما امکان می‌دهد اندیس اولین مقدار مشخص را در لیست پیدا کنید.

```
numbers = [1, 2, 3, 2, 4]
index = numbers.index(2)  # پیدا کردن اندیس اولین مقدار ۲ در لیست
# حاصل: index = 1
```

۳۶.۹ متد `count()`:

متد `count()` به شما امکان می‌دهد تعداد تکرار یک مقدار مشخص را در لیست بشمارید.

```
numbers = [1, 2, 3, 2, 4]
count = numbers.count(2)  # شمارش تعداد تکرار مقدار ۲ در لیست
# حاصل: count = 2
```

۳۶.۱۰ متد `sort()`:

متد `sort()` به شما امکان می‌دهد مقادیر لیست را به ترتیب صعودی (کوچکترین به بزرگترین) مرتب کنید.

```
numbers = [3, 1, 4, 2]
numbers.sort()  # مرتب‌سازی مقادیر به ترتیب صعودی
# حاصل: [1, 2, 3, 4]
```

۳۶.۱۱ متد `reverse()`:

متد `reverse()` به شما امکان می‌دهد مقادیر لیست را به ترتیب نزولی (بزرگترین به کوچکترین) مرتب کنید.

```
numbers = [3, 1, 4, 2]
numbers.reverse()  # معکوس کردن ترتیب مقادیر لیست
# حاصل: [2, 4, 1, 3]
```

۳۶.۱۲ نتیجه‌گیری:

متدهای `append()`، `extend()`، `insert()`، `remove()`، `pop()`، `clear()`، `index()`، `count()`، `sort` و `reverse()` به شما امکان می‌دهند با لیست‌ها به طور موثر کار کنید، مقادیر را مدیریت کنید و تغییرات را اعمال کنید. این متدها به شما ابزارهای قدرتمندی را برای کار با لیست‌ها در پایتون ارائه می‌دهند.

فصل ۳۷: متدهای `fromkeys()` و `setdefault()` در دیکشنری

۳۷.۱ مقدمه به متدهای `fromkeys()` و `setdefault()` در دیکشنری:

متدهای `fromkeys()` و `setdefault()` دو ابزار کاربردی در پایتون برای کار با دیکشنری‌ها هستند. این متدها به شما امکان می‌دهند کلیدها و مقادیر دیکشنری‌ها را به راحتی مدیریت کنید.

۳۷.۲ متد `fromkeys()`:

متد `fromkeys()` به شما امکان می‌دهد یک دیکشنری جدید با کلیدهای مشخص و مقدار پیش‌فرض بسازید.

```
keys = ["نام", "سن", "رشته"]
default_value = "نامشخص"
student = dict.fromkeys(keys, default_value)
# حاصل: {"نام": "نامشخص", "سن": "نامشخص", "رشته": "نامشخص"}
```

۳۷.۳ متد `setdefault()`:

متد `setdefault()` به شما امکان می‌دهد مقدار متناظر با یک کلید را در دیکشنری بیابید. اگر کلید وجود داشته باشد، مقدار متناظر با آن کلید برگشت داده می‌شود. در غیر این صورت، مقدار پیش‌فرض تعیین شده به عنوان مقدار متناظر با کلید قرار داده می‌شود و در دیکشنری ذخیره می‌شود.

```
student = {
    "نام": "علی",
    "سن": 25
}

name = student.setdefault("نام", "نامشخص")  # بازگرداندن مقدار متناظر با کلید "نام" (علی)
city = student.setdefault("شهر", "نامشخص")  # اضافه کردن کلید و مقدار پیش‌فرض به دیکشنری
# حاصل: {"نام": "علی", "سن": ۲۵, "شهر": "نامشخص"}
```

۳۷.۴ نکته‌های مربوط به متدهای `fromkeys()` و `setdefault()` در دیکشنری:

– متد `fromkeys()` به شما امکان می‌دهد یک دیکشنری جدید با کلیدهای مشخص و مقدار پیش‌فرض بسازید.

– متد `setdefault()` به شما امکان می‌دهد مقدار متناظر با یک کلید را در دیکشنری بیابید. اگر کلید وجود داشته باشد، مقدار متناظر با آن کلید برگشت داده می‌شود، در غیر این صورت کلید و مقدار پیش‌فرض به دیکشنری اضافه می‌شوند.

۳۷.۵ نتیجه‌گیری:

متدهای `fromkeys()` و `setdefault()` به شما امکان می‌دهند به راحتی دیکشنری‌ها را ایجاد و مقادیر متناظر با کلیدها را مدیریت کنید. این متدها به شما ابزارهای کاربردی را برای کار با دیکشنری‌ها در پایتون ارائه می‌دهند.

فصل ۳۸: List Comprehension و Dictionary Comprehension

۳۸.۱ مقدمه به List Comprehension و Dictionary Comprehension:

List Comprehension و Dictionary Comprehension **دو تکنیک قدرتمند در پایتون هستند که به شما امکان می‌دهند به راحتی لیست‌ها و دیکشنری‌ها را با استفاده از یک خط کد ایجاد کنید. این تکنیک‌ها به شما اجازه می‌دهند کد را کوتاه‌تر و خواناتر نوشته و فرآیند ایجاد داده‌ها را تسریع می‌کنند.**

۳۸.۲ List Comprehension:

List Comprehension **به شما امکان می‌دهد یک لیست جدید با استفاده از قوانین و مقادیر موجود در یک لیست موجود ایجاد کنید.**

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x ** 2 for x in numbers]  # ایجاد لیست مربع اعداد موجود در لیست اصلی
# حاصل: [1, 4, 9, 16, 25]
```

۳۸.۳ Dictionary Comprehension:

Dictionary Comprehension **به شما امکان می‌دهد یک دیکشنری جدید با استفاده از قوانین و مقادیر موجود در یک دیکشنری موجود ایجاد کنید.**

```
students = {"علی": 25, "محمد": 30, "سارا": 28}
age_group = {name: "جوان" if age < 30 else "بزرگسال" for name, age in students.items()}
# ایجاد دیکشنری بر اساس رده‌بندی سن دانش‌آموزان
# حاصل: {"علی": "جوان", "محمد": "بزرگسال", "سارا": "جوان"}
```

۳۸.۴ نکته‌های مربوط به Comprehension:

– List Comprehension و Dictionary Comprehension **باعث می‌شوند کد شما کوتاه‌تر و خواناتر شود.**

– شما می‌توانید شرط‌ها و عبارات شرطی را در داخل تکنیک‌های Comprehension **استفاده کنید.**

– این تکنیک‌ها برای ایجاد داده‌های کوچک و ساده بسیار مناسب هستند، اما برای داده‌های پیچیده‌تر ممکن است قابلیت خوانایی را کاهش دهند.

۳۸.۵ نتیجه‌گیری:

List Comprehension و Dictionary Comprehension **ابزارهای کاربردی هستند که به شما اجازه می‌دهند به سادگی لیست‌ها و دیکشنری‌ها را ایجاد کنید و کد را کوتاه‌تر و خواناتر نوشته و برنامه‌هایتان را بهبود بخشید. با استفاده از این تکنیک‌ها می‌توانید به صورت مؤثرتر و البته زیباتر با داده‌ها کار کنید.**

فصل ۳۹: مرور کلی

۳۹.۱ مقدمه به مرور کلی:

در این فصل، به مرور کلی از مواردی که در این جزوه آموخته‌ایم، خواهیم پرداخت. مروری اجمالی از مفاهیم و تکنیک‌هایی که در طول جزوه آموخته‌ایم را ارائه می‌دهیم.

۳۹.۲ محتوای فصل:

- **آشنایی با پایتون:** در فصل اول، با مفهوم پایتون و تاریخچه آن آشنا شدیم.

- **نصب پایتون و ویژوال استودیو کد:** در فصل دوم، نحوه نصب پایتون و ویژوال استودیو کد را مرور کردیم.

- **نصب کتابخانه‌ها:** در فصل سوم، نحوه نصب کتابخانه‌ها در محیط ویژوال استودیو کد را یادگرفتیم.

- **ایجاد و مدیریت فایل‌ها:** در فصل چهارم، با ایجاد فایل‌های پایتونی و مدیریت آنها در ویژوال استودیو کد آشنا شدیم.

- **دستور `print`:** در فصل پنجم، دستور `print` برای چاپ اطلاعات در پایتون را مرور کردیم.

- **متغیرها:** در فصل ششم، مفهوم متغیرها و نحوه استفاده از آنها را آموختیم.

- **انواع داده‌ها:** در فصل هفتم، انواع داده‌های اصلی در پایتون از جمله اعداد صحیح، اعشاری، مختلط و رشته‌ها را بررسی کردیم.

– **دستور `type`:** در فصل هشتم، دستور `type` برای تشخیص نوع یک متغیر را مورد بررسی قرار دادیم.

– **عملیات جبری:** در فصل نهم، عملیات جمع، تفریق، ضرب و تقسیم در پایتون را مرور کردیم.

– **کامنت‌ها:** در فصل دهم، نحوه نوشتن کامنت‌ها در کد پایتون را آموختیم.

– **عملیات توان و تقسیم صحیح:** در فصل یازدهم، عملیات توان، توان با اندیس دهی اعشاری، تقسیم صحیح و باقی‌مانده را بررسی کردیم.

– **معرفی داده‌های ``list`، `boolean`، ``dictionary`` و `None`:** در فصل دوازدهم، این داده‌ها را معرفی کردیم.

– **رشته‌ها (Strings):** در فصل سیزدهم، داده‌های رشته‌ای را به تفصیل مرور کردیم.

– **رشته‌های ترکیبی (String Interpolation):** در فصل چهاردهم، نحوه جایگذاری متغیرها در رشته‌ها با استفاده از رشته‌های ترکیبی (f-strings) را آموختیم.

– **آشنایی با ایندکس‌ها در رشته‌ها:** در فصل پانزدهم، با نحوه استفاده از ایندکس‌ها در رشته‌ها آشنا شدیم.

– **تبدیل داده‌ها:** در فصل شانزدهم، نحوه تبدیل داده‌ها از یک نوع به نوع دیگر را مرور کردیم.

– **دستور `input`:** در فصل هفدهم، نحوه استفاده از دستور `input` برای دریافت ورود ی از کاربر را یادگرفتیم.

– **دستور `round`:** در فصل هجدهم، دستور `round` برای گرد کردن اعداد اعشاری را مورد بررسی قرار دادیم.

– **پروژه تبدیل کیلومتر به مایل:** در فصل نوزدهم، یک پروژه عملی برای تبدیل واحد اندازه‌گیری کیلومتر به مایل اجرا کردیم.

– **گذارهای شرطی:** در فصل بیستم، مفهوم و استفاده از گذارهای شرطی (if-else) را مورد بررسی قرار دادیم.

– **اپراتورهای مقایسه:** در فصل بیست و یکم، اپراتورهای مقایسه مانند `==`، `!=`، `<=`، `>=`، `>` و `<` را مرور کردیم.

– **داده‌های بولین (Boolean):** در فصل بیست و دوم، داده‌های بولین و نحوه استفاده از آن‌ها در گذارهای شرطی را بررسی کردیم.

– **اپراتورهای منطقی (``and``، `or`، `not``):** در فصل بیست و سوم، اپراتورهای منطقی ``and``، ``or`` و `not` را آموختیم.

– **پروژه بازی سنگ، کاغذ، قیچی:** در فصل بیست و چهارم، یک پروژه ساده برای بازی سنگ، کاغذ، قیچی بدون استفاده از توابع تعریف شده اجرا کردیم.

– **بازی دو نفره سنگ، کاغذ، قیچی:** در فصل بیست و پنجم، بازی سنگ، کاغذ، قیچی را برای دو نفره پیاده‌سازی کردیم.

– **انواع حلقه‌ها:** در فصل بیست و ششم، انواع حلقه‌ها از جمله ``for``، ``while`` را معرفی کردیم.

– **حلقه `for`:** در فصل بیست و هفتم، حلقه `for` و نحوه استفاده از آن برای تکرار اعضای لیست‌ها و دیکشنری‌ها را آموختیم.

– **تابع `range`:** در فصل بیست و هشتم، تابع `range` برای ایجاد دنباله‌های عددی در حلقه‌ها را مورد بررسی قرار دادیم.

– **حلقه `while`:** در فصل بیست و نهم، حلقه `while` و کاربردهای آن را مورد بررسی قرار دادیم.

– **پروژه بازی سنگ، کاغذ، قیچی با حلقه `while`:** در فصل سی ام، بازی سنگ، کاغذ، قیچی را با استفاده از حلقه `while` پیاده‌سازی کردیم.

– **List Comprehension و Dictionary Comprehension:** در فصل سی و هشتم، تکنیک‌های List Comprehension و Dictionary Comprehension را برای ایجاد لیست‌ها و دیکشنری‌ها با کد کوتاه‌تر مورد بررسی قرار دادیم.

۳۹.۳ نکته‌های مهم:

– پایتون یک زبان برنامه‌نویسی قدرتمند و انعطاف‌پذیر است که به شما امکان می‌دهد برنامه‌های متنوعی را پیاده‌سازی کنید.

– در طول این جزوه، اصول اولیه پایتون، نحوه نصب و تنظیم محیط توسعه، مفاهیم اساسی مانند متغیرها و انواع داده، عملیات ریاضی، گذاره‌های شرطی، حلقه‌ها و تکنیک‌های پیشرفته نظیر Comprehension آموخته شد.

– با پایان این جزوه، شما باید توانایی نوشتن کدهای ساده تا متوسط در زبان پایتون را داشته باشید و بتوانید پروژه‌های کوچک را پیاده‌سازی کنید.

۳۹.۴ نتیجه‌گیری:

این جزوه به شما کمک می‌کند تا اساس‌های زبان برنامه‌نویسی پایتون را فراگیری کنید و با تکنیک‌ها و مفاهیم اصلی

این زبان آشنا شوید. این اطلاعات می‌توانند پایه‌ای قوی برای کسب دانش بیشتر در زمینه برنامه‌نویسی با پایتون باشند و شما را به سمت توانایی‌های پیشرفته‌تر هدایت کنند.

فصل ۴۰: توابع در پایتون (Defining Functions)

۴۰.۱ مقدمه به توابع:

توابع یکی از مفاهیم مهم در برنامه‌نویسی هستند. یک تابع یک بلاک کد است که یک نام دارد و می‌توانید آن را در هر جایی از برنامه‌ی خود فراخوانی کنید. توابع به شما این امکان را می‌دهند که کد را بازیابی‌پذیرتر و قابل مدیریت‌تر کنید.

۴۰.۲ تعریف تابع:

در پایتون، تعریف تابع با استفاده از کلمه کلیدی `def` آغاز می‌شود. یک تابع معمولاً نام دارد که برای فراخوانی آن استفاده می‌شود. می‌توانید ورودی‌ها یا پارامترهای تابع را تعیین کنید و نتایجی را با استفاده از `return` از تابع برگردانید.

نمونه تعریف تابع:

```
def greet(name):
    """این تابع با استفاده از نام فرد ورودی، پیامی خوش‌آمدگویی چاپ می‌کند."""
    print(f"سلام، {name}!")

# فراخوانی تابع
greet("آرمین")
```

۴۰.۳ ورودی‌ها و پارامترها:

توابع می‌توانند ورودی‌ها یا پارامترهای خود را دریافت کنند. این پارامترها اطلاعاتی هستند که تابع برای اجرا نیاز دارد. شما می‌توانید پارامترهای اجباری و یا پارامترهای پیش‌فرض برای یک تابع تعریف کنید.

****نمونه تابع با پارامترها:****

```
def add(a, b):
    """این تابع دو عدد را جمع می‌کند و نتیجه را بر می‌گرداند."""
    result = a + b
    return result

# فراخوانی تابع با ارسال پارامترها
sum_result = add(5, 3)
print(f"جمع: {sum_result}")
```

****۴۰.۴ مقادیر پیش‌فرض:****

می‌توانید برای پارامترهای تابع مقادیر پیش‌فرض تعیین کنید. این مقادیر توسط تابع استفاده می‌شوند اگر مقدار مشخصی برای آن پارامتر ارسال نشود.

****نمونه تابع با مقدار پیش‌فرض:****

```
def power(base, exponent=2):
    """این تابع توان می‌گیرد و به توان می‌رساند، اگر توان مشخص نشود به توان ۲ می‌رساند."""
    result = base ** exponent
    return result

# فراخوانی تابع با مقادیر مختلف
result1 = power(3)      # توان ۲
result2 = power(2, 4)   # توان ٤
```

****۴۰.۵ بازگرداندن مقادیر:****

تواب

ع می‌توانند مقادیری را با استفاده از دستور `return` به تابع فراخوانده بازگردانند. این مقادیر می‌توانند از نوع‌های مختلفی مانند عدد، رشته، لیست و حتی دیکشنری باشند.

نمونه تابع با بازگرداندن مقدار:

```
def get_full_name(first_name, last_name):
    """این تابع نام کامل را به صورت رشته بازگردانی می‌کند."""
    full_name = f"{first_name} {last_name}"
    return full_name

# فراخوانی تابع و بازگرداندن مقدار
full_name = get_full_name("محمد", "رضایی")
```

۴۰.۶ نتیجه‌گیری:

در این فصل، با تعریف توابع در پایتون آشنا شدیم. توابع به شما امکان می‌دهند تا کد خود را سازماندهی کنید و از قابلیت بازیابی کد بهره‌برید. همچنین نحوه تعریف پارامترها، مقادیر پیش‌فرض و بازگرداندن مقادیر را نیز مورد بررسی قرار دادیم.

با یادگیری توابع، می‌توانید کد خود را به راحتی قابل مدیریت‌تر و خواناتر کنید و از تکرار مفرط کد جلوگیری کنید.

فصل ۴۱: Nested Lists (لیست‌های تو در تو) در پایتون

۴۱.۱ مقدمه به Nested Lists:

لیست‌های تو در تو (Nested Lists) به شما امکان می‌دهند داده‌ها را در ساختارهای چند لایه‌ای ذخیره کنید. این ساختار به شما امکان می‌دهد تا داده‌ها را به صورت سلسله مراتبی وارد کنید و در مورد دسترسی به داده‌ها به صورت مرتب‌تری فکر کنید.

۴۱.۲ ساختار Nested Lists:

لیست‌های تو در تو در واقعیت به تعداد نامحدودی لایه تو در تو می‌توانند داشته باشند. این ساختار به صورت زیر تعریف می‌شود:

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

در این مثال، ما یک لیست اصلی داریم که سه لیست دیگر را در خود دارد. این سه لیست دیگر به عنوان عناصر اصلی لیست اصلی ظاهر می‌شوند.

۴۱.۳ دسترسی به عناصر لیست‌های تو در تو:

برای دسترسی به عناصر لیست‌های تو در تو، می‌توانید از عملگرهای فهرست استفاده کنید. به عنوان مثال، اگر بخواهیم به عنصر دوم در لیست دوم از مثال بالا دسترسی پیدا کنیم:

```
value = nested_list[1][1]  # عنصر دوم در لیست دوم
```

۴۱.۴ مثال‌های دیگر Nested Lists:

لیست‌های تو در تو می‌توانند در موارد مختلف مفید باشند. به عنوان مثال، می‌توانید یک لیست از دانش‌آموزان و نمراتشان داشته باشید:

```
students = [["علی", 18], ["محمد", 20], ["سارا", 17]]
```

یا حتی یک جدول اطلاعات فروش در یک فروشگاه:

```
sales_data = [
    ["محصول", "تعداد", "قیمت واحد"],
    ["لپ‌تاپ", 10, 1200],
    ["موبایل", 20, 500],
    ["تبلت", 5, 300],
]
```

۴۱.۵ نتیجه‌گیری:

لیست‌های تو در تو (Nested Lists) یکی از ابزارهای قدرتمند در پایتون هستند که به شما امکان می‌دهند داده‌ها را به صورت سلسله مراتبی ذخیره کنید. این ساختار به شما امکان می‌دهد که در مورد دسترسی به داده‌ها و ایجاد ساختارهای پیچیده‌تر به صورت مؤثر عمل کنید.

فصل ۴۲: Tuple در پایتون

۴۲.۱ مقدمه به Tuple:

Tuple یکی از داده‌ساختارهای مهم در پایتون است که برخلاف لیست‌ها (List) غیرقابل تغییر (Immutable) هستند. این به معنای این است که یک بار ایجاد شده، نمی‌توانید مقادیر داخل یک Tuple را تغییر دهید.

۴۲.۲ ایجاد Tuple:

برای ایجاد یک Tuple، از پرانتز () استفاده می‌شود. مقادیر داخل Tuple با کاما (,) جدا می‌شوند.

نمونه ایجاد Tuple:

```
my_tuple = (1, 2, 3, 4, 5)
```

۴۲.۳ دسترسی به عناصر Tuple:

شما می‌توانید به عناصر Tuple با استفاده از ایندکس‌ها دسترسی پیدا کنید، مانند لیست‌ها.

نمونه دسترسی به عناصر Tuple:

```
my_tuple = (1, 2, 3, 4, 5)
element = my_tuple[2]  # عنصر با ایندکس ۲ (شماره ۳) که برابر با ۳ است.
```

۴۲.۴ Tuple به عنوان یک داده‌ساختار Immutable:

یکی از ویژگی‌های مهم Tuple این است که غیرقابل تغییر هستند. این به معنای این است که پس از ایجاد یک Tuple، نمی‌توانید مقادیر آن را تغییر دهید. این ویژگی می‌تواند در مواردی مفید باشد که نیاز به ایجاد داده‌ساختاری ثابت دارید.

نمونه Tuple غیرقابل تغییر:

```
my_tuple = (1, 2, 3)
# این خطا را ایجاد می‌کند: TypeError: 'tuple' object does not support item assignment
my_tuple[0] = 10
```

۴۲.۵ Tuple بازگشتی (Returned Tuples):

توابع در پایتون می‌توانند Tupleها را بازگشت دهند. این به شما امکان می‌دهد تا چندین مقدار را از یک تابع بازگردانید و از آن‌ها در جایی دیگر استفاده کنید.

نمونه تابع بازگشتی که یک Tuple بر می‌گرداند:

```
def get_name_and_age():
    name = "علی"
    age = 25
    return name, age  # یک Tuple بازگشت داده می‌شود

# فراخوانی تابع و بازگرداندن Tuple
result = get_name_and_age()
name, age = result  # عناصر Tuple به متغیرها منتقل می‌شوند
```

۴۲.۶ تاپل‌های بی‌نام (Unnamed Tuples):

می‌توانید از تاپل‌ها به عنوان داده‌ساختارهای بی‌نام استفاده کنید. این به معنای این است که به جای استفاده از متغیرها برای دسترسی به عناصر تاپل، می‌توانید به صورت مستقیم به عناصر با ایندکس‌ها دسترسی پیدا کنید.

نمونه تاپل بی‌نام:

```
point = (3, 4)
x = point[0]  # عنصر اول Tuple
y = point[1]  # عنصر دوم Tuple
```

۴۲.۷ نتیجه‌گیری:

در این فصل، با داده‌ساختار Tuple **در پایتون آشنا شدیم.** Tuple**ها به شما امکان می‌دهند تا مقادیر را در یک داده‌ساختار غیرقابل تغییر ذخیره کنید و از آن‌ها به عنوان متغیرها یا به عنوان مقادیر بازگشتی در توابع استفاده کنید. این داده‌ساختار بسیار مفید واقع می‌شود در مواردی که نیاز به ایجاد داده‌های ثابت و غیرقابل تغییر دارید.**

فصل ۴۳: Set در پایتون

۴۳.۱ مقدمه به Set:

Set **یک داده‌ساختار در پایتون است که به شما اجازه می‌دهد مجموعه‌ای از عناصر منحصر به فرد را ذخیره کنید.** Set**ها بسیار شبیه به لیست‌ها هستند با این تفاوت که هر عنصر در یک** Set **فقط یک بار ظاهر می‌شود و ترتیب عناصر در** Set **اهمیتی ندارد.**

۴۳.۲ ایجاد Set:

برای ایجاد یک Set**، از دستور** `set()` **استفاده می‌کنید. عناصر داخل** Set **با استفاده از علامت تیره کوتاه** `{}` **جدا می‌شوند.**

نمونه ایجاد Set:

```
my_set = set([1, 2, 3, 4, 5])
```

یا به شکل کوتاهتر:

```
my_set = {1, 2, 3, 4, 5}
```

۴۳.۳ عملیات مشترک بر روی Set:

Setها انواع عملیات مشترک ریاضی مانند اجتماع، تفاضل و تقاطع را پشتیبانی می‌کنند.

نمونه عملیات مشترک بر روی Set:

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

union_set = set1 | set2  # اجتماع مجموعه‌ها
intersection_set = set1 & set2  # تقاطع مجموعه‌ها
difference_set = set1 - set2  # تفاضل مجموعه‌ها
```

۴۳.۴ Set به عنوان داده‌ساختار Mutable:

Setها متغیرند، به این معنا که شما می‌توانید عناصر را اضافه یا حذف کنید.

نمونه اضافه کردن عناصر به Set:

```
my_set = {1, 2, 3}
my_set.add(4)  # اضافه کردن عنصر ۴ به Set
```

نمونه حذف عناصر از Set:

```
my_set = {1, 2, 3}
my_set.remove(2)  # حذف عنصر ۲ از Set
```

۴۳.۵ نتیجه‌گیری:

در این فصل، با داده‌ساختار Set **در پایتون آشنا شدیم.** Set**ها به شما امکان می‌دهند مجموعه‌هایی از عناصر منحصر به فرد را ذخیره کنید و عملیات‌های مشترک بر روی مجموعه‌ها را انجام دهید. این داده‌ساختار مفید است برای حفظ عناصر منحصر به فرد و انجام محاسبات مجموعه‌ای در پایتون.**

فصل ۴۴: Lambda **در پایتون**

۴۴.۱ مقدمه به Lambda**:**

Lambda **یک مکانیزم کوچک و مفید در پایتون است که به شما اجازه می‌دهد تا یک تابع کوتاه و بدون نام (**Anonymous Function**) ایجاد کنید.** Lambda**ها معمولاً برای انجام عملیات‌های ساده و کوتاه مورد استفاده قرار می‌گیرند.**

۴۴.۲ ساختار یک Lambda**:**

یک Lambda **در پایتون به شکل زیر ساختار دارد:**

```
lambda arguments: expression
```

– `arguments`: **پارامترهای ورودی تابع.**

– `expression`: **عبارتی که تابع انجام می‌دهد و نتیجه آن برگردانده می‌شود.**

****نمونه یک Lambda ساده:****

```
square = lambda x: x ** 2
result = square(5)  # نتیجه: ۲۵
```

****۴۴.۳ کاربردهای متداول Lambda:****

– ****تابع‌های Lambda (Sort**: معمولاً در تابع‌های مرتب‌سازی مثل `sorted()` استفاده می‌شوند.**

****نمونه استفاده از Lambda برای مرتب‌سازی بر اساس اعداد دومین:****

```
students = [("علی", 25), ("محمد", 30), ("سارا", 28)]
sorted_students = sorted(students, key=lambda student: student[1])
# مرتب‌سازی دانش‌آموزان بر اساس سن
# نتیجه: [("علی", ۲۵), ("سارا", ۲۸), ("محمد", ۳۰)]
```

– ****تابع‌های Lambda (Filter و Map**: معمولاً به عنوان تابع‌های پارامتری برای توابع `filter()` و `map()` مفید هستند.**

****نمونه استفاده از Lambda در `filter()` و `map()`:****

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(lambda x: x % 2 == 0, numbers)  # انتخاب اعداد زوج
squared_numbers = map(lambda x: x ** 2, numbers)  # مربع اعداد
```

****۴۴.۴ محدودیت‌های Lambda:****

– **Lambda‌ها معمولاً برای توابع کوتاه و ساده استفاده می‌شوند و نباید بسیار پیچیده باشند.**

– از Lambda **برای تعریف توابع با بیش از یک عبارت نباید استفاده کنید.**

۴۴.۵ نتیجه‌گیری:

یک مکانیزم کوتاه و کاربردی در پایتون است که به شما اجازه می‌دهد تا توابع کوچک و بدون نام را به راحتی ایجاد کنید. Lambda**ها معمولاً برای عملیات‌های ساده مورد استفاده قرار می‌گیرند و به شما امکان می‌دهند کد خود را کوتاهتر و خواناتر کنید.**

فصل ۴۵: Map **و استفاده از** Lambda **در آن**

۴۵.۱ مقدمه به Map**:**

تابع `map()` **یک تابع سازگار (یا تابع** Lambda**) را برای هر عنصر یک** iterable **اعمال می‌کند و یک** iterable **جدید با نتایج به دست آمده از تابع را ایجاد می‌کند. این ابزار بسیار کارآمد است زمانی که شما می‌خواهید یک عملیات خاص را بر روی همه عناصر یک لیست یا دیگر** iterable**ها انجام دهید.**

۴۵.۲ ساختار تابع map:

ساختار تابع `map()` به صورت زیر است:

```
map(function, iterable)
```

– `function`: **تابعی که بر روی هر عنصر از** iterable **اعمال می‌شود.**

– `iterable`: **لیست، تاپل، یا هر** iterable **دیگری که بر روی آن تابع** `function` **اعمال می‌شود.**

۴۵.۳ استفاده از Lambda در map:

می‌توانید یک تابع Lambda **به عنوان تابع** `function` **در** `map()` **استفاده کنید تا به سادگی عملیات مورد نظر خود را روی همه عناصر** iterable **انجام دهید.**

نمونه استفاده از map با Lambda:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x ** 2, numbers)  # مربع اعداد
```

۴۵.۴ تبدیل نتیجه به لیست:

نتیجه تابع `map()` **از نوع** iterable **است. برای مشاهده نتایج به عنوان یک لیست، می‌توانید آن را به لیست تبدیل کنید.**

نمونه تبدیل نتیجه map به لیست:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x ** 2, numbers)  # مربع اعداد
```

۴۵.۵ نتیجه‌گیری:

در این فصل، با استفاده از تابع `map()` **در پایتون و استفاده از توابع** Lambda **برای اعمال توابع مختلف بر روی اعداد یک** iterable **آشنا شدیم. تابع** `map()` **به شما امکان می‌دهد تا به سادگی تغییرات مختلفی را بر روی اعداد یا عناصر دیگر یک** iterable **اعمال کنید و نتایج را به دست آورید.**

فصل ۴۶: Filter، All و Any در پایتون

۴۶.۱ مقدمه به Filter:

تابع `filter()` **یک تابع فیلتر کننده را بر روی یک** iterable **اعمال می‌کند و عناصری را که شرط تابع را برآورده می‌کنند را به عنوان یک** iterable **جدید بر می‌گرداند. این ابزار بسیار کارآمد است زمانی که شما می‌خواهید عناصری را از یک لیست یا دیگر** iterableها **بر اساس شرایط خاصی انتخاب کنید.**

۴۶.۲ ساختار تابع filter:

ساختار تابع `filter()` **به صورت زیر است:**

```
filter(function, iterable)
```

– `function`: **تابعی که به عنوان فیلتر بر روی عناصر** iterable **اعمال می‌شود.**

– `iterable`: **لیست، تاپل، یا هر** iterable **دیگری که بر روی آن تابع** `function` **اعمال می‌شود.**

نمونه استفاده از filter:

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(lambda x: x % 2 == 0, numbers)  # انتخاب اعداد زوج
```

۴۶.۳ تبدیل نتیجه به لیست:

نتیجه تابع `filter()` **از نوع** iterable **است. برای مشاهده نتایج به عنوان یک لیست، می‌توانید آن را به لیست تبدیل کنید.**

نمونه تبدیل نتیجه filter به لیست:

```
even_numbers_list = list(even_numbers)  # تبدیل به لیست
# نتیجه: [2, 4, 6]
```

۴۶.۴ تابع All:

تابع `all()` بررسی می‌کند که آیا تمام عناصر یک iterable شرط داده شده را برآورده می‌کنند یا خیر. اگر همه عناصر شرط را برآورده کنند، `all()` مقدار True بر می‌گرداند؛ در غیر این صورت، مقدار False بر می‌گرداند.

****نمونه استفاده از تابع all:****

```
numbers = [2, 4, 6, 8, 10]
are_even = all(x % 2 == 0 for x in numbers)  # آیا همه اعداد زوج هستند؟
# نتیجه: True
```

****۴۶.۵ تابع Any:****

تابع `any()` بررسی می‌کند که آیا حداقل یکی از عناصر یک iterable شرط داده شده را برآورده می‌کند یا خیر. اگر حداقل یکی از عناصر شرط را برآورده کند، `any()` مقدار True بر می‌گرداند؛ در غیر این صورت، مقدار False بر می‌گرداند.

****نمونه استفاده از تابع any:****

```
numbers = [1, 3, 5, 7, 8]
has_even = any(x % 2 == 0 for x in numbers)  # آیا حداقل یکی از اعداد زوج است؟
# نتیجه: True
```

****۴۶.۶ نتیجه‌گیری:****

در این فصل، با توابع `all()`، `any()` و `filter()` در پایتون آشنا شدیم. این توابع برای انتخاب عناصر با شرایط خاص و ارزیابی شرایط مورد نظر برای عناصر iterable بسیار مفید هستند. همچنین با استفاده از توابع `sorted()`، `min()`، `max()`، `reversed()`، `len()`، `abs()`، `sum()` و `round()` آشنا شدیم که عملیات‌های متداول روی داده‌ها را انجام می‌دهند.

فصل ۴۷: توابع Sorted، Min، Max، Reversed، Len، Abs، Sum

۴۷.۱ تابع Sorted:

تابع `sorted()` یک iterable را مرتب می‌کند و نسخه‌ی جدیدی از آن باز می‌گرداند. این تابع به شما امکان می‌دهد یک iterable را بر اساس مقادیر آن یا به تعیین شده ایجاد کنید.

نمونه استفاده از تابع sorted:

```
numbers = [5, 1, 3, 2, 4]
sorted_numbers = sorted(numbers)  # مرتب‌سازی لیست اعداد
# نتیجه: [1, 2, 3, 4, 5]
```

۴۷.۲ تابع Min:

تابع `min()` کمترین عنصر یک iterable را باز می‌گرداند. این تابع برای یافتن مقدار کمترین عنصر یک لیست یا تاپل مفید است.

نمونه استفاده از تابع min:

```
numbers = [5, 1, 3, 2, 4]
min_number = min(numbers)  # یافتن کمترین عدد
# نتیجه: ۱
```

۴۷.۳ تابع Max:

تابع `max()` بزرگترین عنصر یک iterable را باز می‌گرداند. این تابع برای یافتن مقدار بزرگترین عنصر یک لیست یا تاپل مفید است.

نمونه استفاده از تابع max:

```
numbers = [5, 1, 3, 2, 4]
max_number = max(numbers)  # یافتن بزرگترین عدد
# نتیجه: ۵
```

۴۷.۴ تابع Reversed:

تابع `reversed()` یک iterable را معکوس می‌کند و نسخه‌ی معکوس آن را باز می‌گرداند. این تابع برای بازگرداندن ترتیب عناصر یک لیست یا تاپل به صورت معکوس مفید است.

****نمونه استفاده از تابع reversed:****

```
numbers = [1, 2, 3, 4, 5]
reversed_numbers = list(reversed(numbers))  # معکوس کردن ترتیب اعداد
# نتیجه: [5, 4, 3, 2, 1]
```

۴۷.۵ تابع Len:

تابع `len()` تعداد عناصر یک iterable را باز می‌گرداند. این تابع برای دریافت تعداد عناصر یک لیست، تاپل یا رشته مفید است.

****نمونه استفاده از تابع len:****

```
numbers = [1, 2, 3, 4, 5]
length = len(numbers)  # تعداد عناصر لیست
# نتیجه: ۵
```

۴۷.۶ تابع Abs:

تابع `abs()` مقدار مطلق یک عدد را باز می‌گرداند. این تابع برای گرفتن مقدار مطلق یک عدد مفید است.

****نمونه استفاده از تابع abs:****

```
number = -5
absolute_value = abs(number)  # مقدار مطلق عدد
# نتیجه: ۵
```

۴۷.۷ تابع Sum:

تابع `sum()` مجموع تمام اعداد یک iterable عددی را باز می‌گرداند. این تابع برای محاسبه مجموع اعداد یک لیست یا تاپل مفید است.

****نمونه استفاده از تابع sum:****

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

```
total = sum(numbers)  # مجموع اعداد لیست
# نتیجه: ۱۵
```

۴۷.۸ تابع Round:

تابع `round()` عدد اعشاری را به نزدیکترین عدد صحیح یا به تعداد اعشار مشخص شده گرد می‌کند. این تابع برای گرد کردن اعداد اعشاری به صورت مورد نیاز مفید است.

نمونه استفاده از تابع round:

```
number = 3.14159
rounded_number = round(number, 2)  # گرد کردن عدد به ۲ اعشار
# نتیجه: ۳.۱۴
```

۴۷.۹ نتیجه‌گیری:

در این فصل، با استفاده از توابع `sorted()`، `max()`، `min()`، `reversed()`، `len()`، `abs()`، `sum()`، و `round()` آشنا شدیم که عملیات‌های متداول روی داده‌ها را انجام می‌دهند. این توابع برای مدیریت و تحلیل داده‌ها در پایتون بسیار مفید هستند.

فصل ۴۸: تابع Zip در زبان Python

۴۸.۱ مقدمه به تابع Zip:

تابع `zip()` در زبان Python یک ابزار کارآمد برای ترکیب (zip) دو یا چند iterable با هم است. این تابع به شما امکان می‌دهد اطلاعات از چندین لیست، تاپل یا iterable دیگر را در کنار هم گروه‌بندی کرده و یک iterable جدید از تاپل‌ها ایجاد کنید.

۴۸.۲ استفاده از تابع Zip:

برای استفاده از تابع `zip()`، شما باید دو یا چند iterable را به عنوان ورودی به آن ارائه دهید. تابع `zip()` تاپل‌هایی از عناصر متناظر از هر iterable ایجاد می‌کند. تعداد تاپل‌ها بر اساس iterable کوتاه‌تر مشخص می‌شود.

نمونه استفاده از تابع zip:

```
names = ["علی", "محمد", "سارا"]
scores = [85, 92, 78]

zipped_data = zip(names, scores)  # ترکیب لیست نام‌ها و امتیازها
result = list(zipped_data)
# نتیجه: [('علی', ۸۵), ('محمد', ۹۲), ('سارا', ۷۸)]
```

۴۸.۳ انتقال داده‌های Zip به لیست‌ها:

بعضی مواقع، شما ممکن است بخواهید داده‌هایی که توسط تابع `zip()` ایجاد شده است را به لیست‌ها یا دیگر iterable تبدیل کنید. برای انجام این کار، شما می‌توانید از تابع `list()` بر روی نتیجه تابع `zip()` استفاده کنید.

نمونه انتقال داده‌های Zip به لیست‌ها:

```
names = ["علی", "محمد", "سارا"]
scores = [85, 92, 78]
```

```
zipped_data = zip(names, scores)  # ترکیب لیست نام‌ها و امتیازها
result = list(zipped_data)  # تبدیل به لیست
# نتیجه: [('علی', ۸۵), ('محمد', ۹۲), ('سارا', ۷۸)]
```

۴۸.۴ **نکته‌های مربوط به تابع** Zip**:**

– تابع `zip()` **برای ترکیب داده‌های دو یا چند** iterable **با هم بسیار مفید است.**

– تعداد تاپل‌های ایجاد شده توسط `zip()` **بر اساس** iterable **کوتاه‌تر مشخص می‌شود.**

– شما می‌توانید از توابع مانند `list()` **برای تبدیل نتایج به لیست‌ها یا دیگر** iterable **استفاده کنید.**

۴۸.۵ **نتیجه‌گیری:**

در این فصل، با تابع `zip()` **آشنا شدیم که به شما اجازه می‌دهد داده‌های دو یا چند** iterable **را به هم متصل کنید و تاپل‌هایی از داده‌های متناظر ایجاد کنید. این تابع مفید است زمانی که می‌خواهید اطلاعات متناظر را از چندین منبع به صورت همزمان استخراج کنید.**

فصل ۴۹: Error Handling در زبان Python

۴۹.۱ مقدمه به Error Handling:

در برنامه‌نویسی، همیشه ممکن است با خطاها و استثناءها روبه‌رو شوید. Error Handling به شما امکان می‌دهد با این خطاها مقابله کرده و برنامه خود را به درستی اجرا کنید. در زبان Python، شما می‌توانید از ساختارهایی مانند `try`، `except`، `finally` برای Error Handling استفاده کنید.

۴۹.۲ استفاده از try و except:

ساختار `try` و `except` به شما اجازه می‌دهد کدی را در داخل `try` اجرا کنید و در صورتی که خطایی اتفاق بیافتد، بلافاصله به بخش `except` منتقل شوید تا با خطا برخورد کنید و برنامه را به درستی ادامه دهید.

نمونه استفاده از try و except:

```
try:
    number = int(input("لطفاً یک عدد وارد کنید: "))
    result = 10 / number
    print("نتیجه: ", result)
except ZeroDivisionError:
    print("خطا: تقسیم بر صفر مجاز نیست.")
except ValueError:
    print("خطا: ورودی باید یک عدد باشد.")
```

۴۹.۳ استفاده از finally:

ساختار `finally` اجازه می‌دهد کدی را اجرا کنید که برای هر حالت (با یا بدون خطا) اجرا شود. این بخش به عنوان یک مکان مناسب برای انجام تمیزکاری‌هایی مانند بستن فایل‌ها یا اتصال به پایگاه داده استفاده می‌شود.

نمونه استفاده از finally:

```
file = None
try:
    file = open("example.txt", "r")
    data = file.read()
    print(data)
except FileNotFoundError:
    print("خطا: فایل مورد نظر یافت نشد.")
finally:
    if file is not None:
        file.close()
```

۴۹.۴ استفاده از Exception:

شما می‌توانید از `except Exception` **برای مدیریت همه انواع خطاها استفاده کنید. این به شما اجازه می‌دهد تمام خطاهای عمومی را در یک بلاک** `except` **مدیریت کنید.**

نمونه استفاده از except Exception:

```
try:
    result = 10 / 0
except Exception as e:
    print("خطا: ", e)
```

۴۹.۵ نکته‌های مربوط به Error Handling:

– Error Handling **به شما امکان می‌دهد با خطاها مقابله کرده و برنامه خود را به درستی اجرا کنید.**

– شما می‌توانید از ساختارهای `try`, `except`, `finally` برای Error Handling در Python استفاده کنید.

– می‌توانید از `except Exception` برای مدیریت همه انواع خطاها استفاده کنید.

****۴۹.۶ نتیجه‌گیری:****

در این فصل، با مفهوم Error Handling و استفاده از ساختارهای `try`, `except`, `finally` برای مدیریت خطاها در زبان Python آشنا شدیم. این مفاهیم بسیار مهمی در برنامه‌نویسی هستند و به شما امکان می‌دهند برنامه‌های خود را ایمن‌تر و پایدارتر کنید.

فصل ۵۰: Module ها در زبان Python

****۵۰.۱ مقدمه به Module ها:****

در زبان Python، یک Module یک فایل Python است که توابع، متغیرها و کلاس‌های مرتبط را در خود جای داده و اجازه می‌دهد که کد شما به صورت منظم و مدیریت‌پذیر سازمان‌دهی شود. هر برنامه Python می‌تواند از ماژول‌های دیگری که در همان پروژه یا از طریق پکیج‌های مختلف در دسترس هستند، استفاده کند.

****۵۰.۲ ایجاد Module:****

شما می‌توانید یک Module ایجاد کنید توسط ایجاد یک فایل Python با پسوند `.py` و تعریف توابع و متغیرهای مرتبط در آن.

****مثال Module با نام `my_module.py`:****

```
# تعریف تابع در ماژول
def greet(name):
    return f"سلام{name}"

# تعریف متغیر در ماژول
PI = 3.14159
```

****۵۰.۳ Import Module:****

برای استفاده از یک Module در برنامه‌ی Python، شما باید آن را وارد کنید (import) با استفاده از دستور `import`.

مثال Import Module:

```
# وارد کردن ماژول
import my_module

# استفاده از تابع ماژول
greeting = my_module.greet("علی")
print(greeting)  # خروجی: سلام علی

# استفاده از متغیر ماژول
print(my_module.PI)  # خروجی: ۳,۱۴۱۵۹
```

۵۰.۴ استفاده از Aliases:

شما می‌توانید از نام‌های مخفف (aliases) برای Moduleها یا توابعی که در آنها تعریف شده‌اند، استفاده کنید. این کار باعث کاهش طول کد و بهبود خوانایی آن می‌شود.

مثال استفاده از Aliases:

```
import my_module as mm  # تعریف نام مخفف برای ماژول

# استفاده از نام مخفف برای تابع ماژول
greeting = mm.greet("علی")
print(greeting)  # خروجی: سلام علی
```

۵۰.۵ استفاده از from...import:

شما می‌توانید یک یا چند تابع یا متغیر خاص را از یک ماژول به صورت مستقیم وارد کنید.

مثال استفاده از from...import:

```
from my_module import greet, PI  # وارد کردن تابع و متغیر خاص از ماژول

# استفاده از تابع و متغیر وارد شده
greeting = greet("علی")
print(greeting)  # خروجی: سلام علی
print(PI)  # خروجی: ۳,۱۴۱۵۹
```

۵۰.۶ استفاده از Built-in Modules:

پایتون دارای ماژول‌های تعداد زیادی Built-in است که قبلاً تعریف شده‌اند و می‌توانید از آنها برای انجام کارهای مختلف استفاده کنید. برای مثال، ماژول `math` ابزارهای ریاضی مفیدی ارائه می‌دهد.

مثال استفاده از Built-in Module `math`:

```
import math

result = math.sqrt(25)  # محاسبه جذر
print(result)  # خروجی: ۵.۰
```

۵۰.۷ نکته‌های مربوط به Module:

– Moduleها به شما اجازه می‌دهند کدتان را سازمان‌دهی و مدیریت کنید.

– شما می‌توانید Moduleها را با استفاده از دستور `import` وارد کنید و از توابع و متغیرهای آنها استفاده کنید.

– از توابع `from...import` می‌توانید توابع و متغیرهای خاص را به صورت مستقیم وار

د کنید.

– Python دارای ماژول‌های Built-in بسیاری است که ابزارهای مفیدی را ارائه می‌دهند.

۵۰.۸ نتیجه‌گیری:

در این فصل، مفهوم Moduleها در زبان Python را آموختیم و یادگرفتیم چگونه Moduleها را ایجاد کرده و در برنامه‌های خود وارد کنیم. همچنین با استفاده از Built-in Modules و نحوه استفاده از توابع و متغیرهای آنها آشنا شدیم. Moduleها ابزار مهمی در سازماندهی کد و افزایش قابلیت انعطاف‌پذیری برنامه‌ها هستند.

فصل ۵۱: شی گرایی (Object-Oriented Programming) در زبان Python

۵۱.۱ مقدمه به شی گرایی:

شی گرایی (OOP) یک پارادایم برنامه‌نویسی است که به شما امکان می‌دهد کد خود را با استفاده از اشیاء (Objects) و کلاس‌ها (Classes) سازماندهی کنید. در OOP، اشیاء مجموعه‌ای از داده‌ها و توابع هستند که با هم مرتبطند و به صورت مستقل از کد دیگر قابل استفاده‌اند.

۵۱.۲ کلاس‌ها و اشیاء:

– کلاس (Class): یک کلاس تعریف می‌کند که چگونه یک شیء باید ساخته شود. کلاس‌ها می‌توانند ویژگی‌ها (attributes) و متدها (methods) را تعریف کنند. به عبارت دیگر، کلاس یک الگوی ساخت (blueprint) برای ایجاد اشیاء است.

مثال تعریف کلاس:

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def drive(self):
        print(f"{self.make} {self.model} is driving.")
```

– **شیء (Object): یک شیء یک نمونه از یک کلاس است که با استفاده از الگوی ساخت کلاس ایجاد می‌شود. اشیاء دارای ویژگی‌ها و متدها هستند که از کلاس تعریف شده‌اند.**

مثال ایجاد اشیاء:

```
car1 = Car("Toyota", "Camry")
car2 = Car("Honda", "Civic")
```

۵۱.۳ ویژگی‌ها و متدها:

– **ویژگی (Attribute): ویژگی‌ها داده‌هایی هستند که به شیء مرتبط می‌شوند. به عبارت دیگر، ویژگی‌ها ویژگی‌های داخلی یک شیء هستند که اطلاعات را نگهداری می‌کنند.**

مثال تعریف ویژگی‌ها:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

– **متد (Method): متدها عملکردهایی هستند که بر روی اشیاء اعمال می‌شوند. آنها می‌توانند به ویژگی‌ها دسترسی داشته باشند و عملیات‌هایی روی آنها انجام دهند.**

مثال تعریف متدها:

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius
```

۵۱.۴ مفهوم ارث‌بری (Inheritance):

ارث‌بری به شما اجازه می‌دهد یک کلاس جدید را از یک کلاس موجود (پدر یا والد) ایجاد کنید و ویژگی‌ها و متدهای کلاس پدر را به کلاس جدید ارث بری کنید. این ویژگی به شما امکان می‌دهد کد را باز استفاده کرده و سازمان‌دهی بهتری برای کلاس‌های مرتبط داشته باشید.

مثال ارث‌بری:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass  # متد خالی

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"
```

۵۱.۵ مفهوم پلی‌مورفیسم (Polymorphism):

پلی‌مورفیسم به شما امکان می‌دهد توابع یا متدهای مشابه را برای اشیاء مختلف با استفاده از تعریف‌های مختلف پیاده‌سازی کنید. به این ترتیب، شما می‌توانید توابعی را بسازید که بر روی اشیاء مختلف عمل کنند.

مثال پلی‌مورفیسم:

```
def animal_sound(animal):
    return animal.speak()

dog = Dog("Buddy")
cat = Cat("Whiskers")

print(animal_sound(dog))  # خروجی: Buddy says Woof!
print(animal_sound(cat))  # خروجی: Whiskers says Meow!
```

۵۱.۶ نکته‌های مربوط به OOP:

– OOP به شما امکان می‌دهد کد را سازماندهی کنید و از قابلیت بازاستفاده و

انعطاف‌پذیری بیشتری بهره‌ببرید.

– کلاس‌ها و اشیاء به صورت اصولی برای مدیریت کد و داده‌ها در برنامه‌ها استفاده می‌شوند.

– مفاهیم ارث‌بری و پلی‌مورفیسم به شما کمک می‌کنند کد را بهبود ببخشید و کلاس‌های مشابه را بازاستفاده کنید.

۵۱.۷ نتیجه‌گیری:

شی گرایی یک مفهوم مهم در برنامه‌نویسی است که به شما امکان می‌دهد کد خود را به صورت سازماندهی‌شده تر و انعطاف‌پذیرتری ایجاد کنید. با استفاده از کلاس‌ها و اشیاء، می‌توانید برنامه‌هایی با کیفیت بهتر بنویسید و قابلیت بازاستفاده را افزایش دهید.

فصل ۵۲: بررسی `repr` در پایتون

۵۲.۱ مقدمه به `repr`:

در زبان پایتون، تابع `repr` یک تابع ویژه است که برای نمایش رشته‌ای قابل تفسیر (توصیف‌کننده) از یک شیء استفاده می‌شود. `repr` به ما اجازه می‌دهد یک نمایش مفهومی از یک شیء ایجاد کنیم تا در ارتباط با متغیرها و شیءها بتوانیم اطلاعات بیشتری کسب کنیم.

۵۲.۲ تعریف `repr` در یک کلاس:

برای استفاده از `repr` در یک کلاس، شما باید تابع `__repr__` را در کلاس خود تعریف کنید. این تابع باید یک رشته برگرداند که توصیفی از شیء را ایجاد کند.

مثال تعریف `__repr__`:

```
class Person:
    def __init__(self, name, age):
```

```
        self.name = name
        self.age = age

    def __repr__(self):
        return f"Person(name='{self.name}', age={self.age})"
```

۵۲.۳ **استفاده از `repr`:**

با تعریف `__repr__` در کلاس، می‌توانیم از تابع `repr` برای نمایش اطلاعات یک شیء استفاده کنیم. این اطلاعات برای دیباگ و تست‌ها بسیار مفید هستند.

مثال استفاده از `repr`:

```
person = Person("Alice", 30)
print(repr(person))  # خروجی: Person(name='Alice', age=30)
```

۵۲.۴ **نکته‌های مرتبط با `repr`:**

– `repr` باید یک رشته باشد که توسط پایتون تفسیر شود.

– معمولاً `repr` باید اطلاعات کافی را برای بازسازی یک شیء فراهم کند.

– توجه داشته باشید که `repr` معمولاً باید مفهومی و توصیف‌کننده باشد و نباید برای نمایش یک فرمت ظاهری بصری استفاده شود.

۵۲.۵ **نتیجه‌گیری:**

تابع `repr` در پایتون یک ابزار قدرتمند برای نمایش مفهومی و توصیف‌کننده اطلاعات یک شیء است. با تعریف `__repr__` در کلاس‌های خود، می‌توانید اطلاعات مفهومی از شیء را نمایش دهید که برای دیباگ، تست‌ها و تعامل با کد دیگران بسیار مفید است.

فصل ۵۳: بررسی Getter و Setter و Properties در پایتون

۵۳.۱ مقدمه به Getter و Setter و Properties:

در برنامه‌نویسی شیءگرا، گترها (Getter) و سترها (Setter) ابزارهایی هستند که به شما امکان می‌دهند دسترسی به و تغییر مقادیر یک ویژگی (متغیر عضو کلاس) را کنترل کنید. از آنجایی که پایتون اجازه دسترسی مستقیم به ویژگی‌های کلاس را می‌دهد، گترها و سترها به صورت غیرمستقیم مفید هستند.

۵۳.۲ گترها (Getter) در پایتون:

یک گتر (Getter) یک متد است که به شما امکان می‌دهد مقدار یک ویژگی را بخوانید. نام یک گتر معمولاً با `_get` آغاز می‌شود.

مثال تعریف گتر:

```
class Circle:
    def __init__(self, radius):
        self._radius = radius  # ویژگی حفاظت‌شده

    def get_radius(self):
        return self._radius

    def area(self):
        return 3.14 * self._radius ** 2
```

۵۳.۳ سترها (Setter) در پایتون:

یک ستر (Setter) یک متد است که به شما امکان می‌دهد مقدار یک ویژگی را تغییر دهید. نام یک ستر معمولاً با `_set` آغاز می‌شود.

مثال تعریف ستر:

```
class Circle:
    def __init__(self, radius):
        self._radius = radius  # ویژگی حفاظت‌شده

    def get_radius(self):
        return self._radius

    def set_radius(self, radius):
        if radius >= 0:
            self._radius = radius
```

```
    def area(self):
        return 3.14 * self._radius ** 2
```

۵۳.۴ **استفاده از Properties در پایتون:**

Properties **(ویژگی‌ها) در پایتون از ترکیب گترها و سترها برای تعریف ویژگی‌های کلاس استفاده می‌کنند. این ویژگی‌ها به شما اجازه می‌دهند ویژگی‌ها را مثل ویژگی‌های عادی مورد استفاده قرار دهید.**

مثال استفاده از Properties:

```
class Circle:
    def __init__(self, radius):
        self._radius = radius  # ویژگی حفاظت‌شده

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        if value >= 0:
            self._radius = value

    def area(self):
        return 3.14 * self._radius ** 2
```

۵۳.۵ **نکته‌های مرتبط با Getter و Setter و Properties:**

– استفاده از گترها و سترها به شما امکان می‌دهد برای دسترسی و تغییر مقادیر ویژگی‌ها کنترل بیشتری داشته باشید.

– توجه داشته باشید که نام ویژگی‌های شما معمولاً با یک زیرخط (_) شروع می‌شود تا نشان دهد که ویژگی‌ها به طور مستقیم دسترسی نباید داشته باشند.

– استفاده از Properties **به شما امکان می‌دهد ویژگی‌هایی را تعریف کنید که مثل ویژگی‌های عادی به نظر بیایند و از گترها و سترها به عنوان روش‌های دسترسی به آن‌ها استفاده کنید.**

۵۳.۶ نتیجه‌گیری:

در این فصل، مفاهیم Getter **و** Setter **و** Properties **در پایتون را بررسی کردیم. این ابزارها به شما اجازه می‌دهند کنترل بیشتری بر روی دسترسی و تغییر مقادیر ویژگی‌ها در کلاس‌های خود داشته باشید.** Properties **به شما امکان می‌دهند ویژگی‌هایی را تعریف کنید که مثل ویژگی‌های عادی به نظر بیایند و با استفاده از گترها و سترها به آن‌ها دسترسی کنید.**

فصل ۵۴: بررسی Iterator و Iterable در پایتون و تفاوت آنها

۵۴.۱ مقدمه به Iterator و Iterable:

Iterator **و** Iterable **دو مفهوم کلیدی در زبان پایتون هستند که به شما اجازه می‌دهند برای اجتیاح و پیمایش داده‌ها در یک ساختار خاص استفاده کنید. این دو مفهوم مهم برای فهم بهتر نحوه کار با داده‌ها در پایتون هستند.**

۵۴.۲ Iterable:

یک Iterable **(قابل تکرار) هر چیزی است که می‌توانید بر روی آن حلقه بزنید (**iterable **شود) و به ترتیب اعضای آن را پیمایش کنید. لیست‌ها، رشته‌ها، دیکشنری‌ها و سایر ساختارهای داده در پایتون** Iterable **هستند.**

مثال Iterable:

```
my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)
```

۵۴.۳ Iterator:

یک Iterator **(تکرارکننده) یک شیء است که برای پیمایش اعضای یک** Iterable **به کار می‌رود.** Iterator **باید دارای دو متد باشد:** `__iter__` **که خود** Iterator **را باز می‌گرداند و** `__next__` **که مقدار بعدی در** Iterable **را باز می‌گرداند.**

مثال Iterator:

```
my_list = [1, 2, 3, 4, 5]
my_iterator = iter(my_list)
print(next(my_iterator))  # حاصل: ۱
print(next(my_iterator))  # حاصل: ۲
```

۵۴.۴ تفاوت بین Iterable و Iterator:

– Iterable **یک مجموعه از داده‌ها را نشان می‌دهد و قابلیت پیمایش دارد.**

– Iterator **وظیفه پیمایش یک** Iterable **را دارد و اعضای آن را به ترتیب باز می‌گرداند.**

– Iterable **به وسیله تابع** `iter()` **به** Iterator **تبدیل می‌شود.**

– Iterator **با استفاده از تابع** `next()` **اعضای** Iterable **را باز می‌گرداند و در صورتی که به انتهای** Iterable **برسد، خطا ایجاد می‌کند.**

۵۴.۵ نتیجه‌گیری:

Iterable **و** Iterator **دو مفهوم مهم در پایتون هستند که به شما امکان می‌دهند داده‌ها را به ترتیب پیمایش کنید.** Iterable **یک مجموعه از داده‌ها را نشان می‌دهد و** Iterator **وظیفه پیمایش این داده‌ها را دارد. با درک تفاوت بین این دو مفهوم، می‌توانید به بهترین شکل ممکن از پتانسیل پیمایش داده‌ها در پایتون استفاده کنید.**

فصل ۵۵: بررسی متدهای `iter` و `next` و ایجاد دستی حلقه `for`

۵۵.۱ مقدمه به متدهای `iter` و `next`:

متدهای `iter` و `next` **دویی اساسی در زبان پایتون هستند که برای ایجاد و پیمایش** Iteratorها **(تکرارکننده‌ها) به کار می‌روند. با استفاده از این متدها، شما می‌توانید داده‌ها را به ترتیب پیمایش کنید.**

۵۵.۲ متد `iter`:

متد `iter` **یک متد خاص است که در یک** Iterable **(قابل تکرار) تعریف می‌شود. این متد یک** Iterator **(تکرارکننده) ایجاد می‌کند که به شما امکان می‌دهد بر روی اعضای** Iterable **حلقه بزنید.**

مثال استفاده از متد `iter`:

```
my_list = [1, 2, 3, 4, 5]
my_iterator = iter(my_list)
```

۵۵.۳ متد `next`:

متد `next` **یک متد خاص است که در یک** Iterator **(تکرارکننده) تعریف می‌شود. این متد مقدار بعدی در** Iterator **را باز می‌گرداند. اگر به انتهای** Iterator **برسید، یک استثنا** `StopIteration` **ایجاد می‌شود.**

مثال استفاده از متد `next`:

```
my_list = [1, 2, 3, 4, 5]
my_iterator = iter(my_list)
next_value = next(my_iterator)
```

۵۵.۴ ایجاد دستی حلقه `for`:

شما می‌توانید یک دستی حلقه `for` **ایجاد کنید با استفاده از متدهای** `iter` **و** `next`**. این کار به شما امکان می‌دهد به صورت دستی بر روی اعضای یک** Iterable **حلقه بزنید.**

مثال ایجاد دستی حلقه `for`:

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

```python
my_iterator = iter(my_list)

while True:
    try:
        item = next(my_iterator)
        print(item)
    except StopIteration:
        break
```

۵۵.۵ نکته‌های مرتبط با متدهای `iter` و `next`:

– متدهای `iter` و `next` برای ایجاد و پیمایش Iteratorها استفاده می‌شوند.

– با استفاده از این متدها، می‌توانید به ترتیب اعضای یک Iterable را پیمایش کنید.

– استفاده از دستی حلقه `for` با متدهای `iter` و `next` به شما امکان می‌دهد به صورت دستی بر روی اعضای یک Iterable حلقه بزنید.

۵۵.۶ نتیجه‌گیری:

در این فصل، متدهای `iter` و `next` را برای ایجاد و پیمایش Iteratorها (تکرارکننده‌ها) در پایتون بررسی کردیم.

فصل ۵۶: بررسی Generators در پایتون

۵۶.۱ مقدمه به Generators:

Generators (مولدها) یک اساس مهم در زبان پایتون هستند که به شما امکان می‌دهند دنباله‌هایی از مقادیر را تولید کنید و از آنها در طول زمان به صورت تاخیری (lazy) استفاده کنید. این مفهوم مفید برای پردازش داده‌های بزرگ یا تولید دنباله‌های بی‌پایان است.

۵۶.۲ تولید Generators:

یک Generator با استفاده از تابع‌ها و عبارات مانند `yield` تعریف می‌شود. هنگامی که تابع یک عبارت `yield` اجرا می‌شود، مقداری تولید می‌شود و اجرای تابع به همان نقطه‌ای که `yield` واقع شده است متوقف می‌شود. از این تابع می‌توانید به عنوان یک Generator استفاده کنید.

مثال تعریف یک Generator:

```
def simple_generator():
    yield 1
    yield 2
    yield 3

gen = simple_generator()
```

۵۶.۳ استفاده از Generators:

Generators به شما امکان می‌دهند داده‌ها را به صورت تاخیری تولید کنید و به صورت پیمایشی از آنها استفاده کنید. برای این کار می‌توانید از حلقه `for` یا تابع `next()` استفاده کنید.

مثال استفاده از یک Generator:

```
def simple_generator():
    yield 1
    yield 2
    yield 3

gen = simple_generator()

for item in gen:
```

```
    print(item)
```

۵۶.۴ مزایای Generators:**

– مصرف حافظه بهینه: Generators **فقط یک مقدار را در هر لحظه در حافظه نگه می‌دارند، بنابراین برای پردازش داده‌های بزرگ مفیدند.**

– تولید داده به صورت تاخیری: Generators **به شما امکان می‌دهند داده‌ها را به صورت تاخیری تولید کنید و از آنها به مرور زمان استفاده کنید.**

– پایداری در طول زمان: Generators **می‌توانند دنباله‌های بی‌پایان از داده‌ها را تولید کنند.**

۵۶.۵ نتیجه‌گیری:

Generators **ابزارهای مفیدی در پایتون هستند که به شما امکان می‌دهند داده‌ها را به صورت تاخیری تولید کنید و از آنها به صورت پیمایشی استفاده کنید. این مفهوم مفید برای پردازش داده‌های بزرگ یا تولید دنباله‌های بی‌پایان است و به بهبود کارایی برنامه‌های شما کمک می‌کند.**

فصل ۵۷: انواع استفاده از Functions در پایتون

۵۷.۱ مقدمه به انواع استفاده از Functions:

در پایتون، Functions (توابع) به عنوان بلاک‌هایی از کد تعریف می‌شوند که یک وظیفه خاص را انجام می‌دهند. در این فصل، انواع مختلفی از استفاده از Functions را بررسی خواهیم کرد.

۵۷.۲ تعریف و استفاده از Functions:

تعریف یک Function در پایتون با استفاده از کلمه کلیدی `def` انجام می‌شود. Functions می‌توانند با یک تعداد دلخواه از ورودی‌ها تعریف شوند و مقدار خروجی دلخواهی را تولید کنند.

مثال تعریف و استفاده از یک Function:

```
def add(x, y):
    result = x + y
    return result

sum = add(3, 4)  # فراخوانی تابع و ذخیره نتیجه در متغیر sum
```

۵۷.۳ استفاده از Functions به عنوان متغیرها:

در پایتون، Functions نیز می‌توانند به عنوان متغیرها مورد استفاده قرار گیرند. این به شما امکان می‌دهد توابع را به عنوان ورودی به توابع دیگر منتقل کرده و از آنها به عنوان خروجی استفاده کنید.

مثال استفاده از Function به عنوان متغیر:

```
def add(x, y):
    result = x + y
    return result

def subtract(x, y):
    result = x - y
    return result
```

```
operation = add  # تابع add را به عنوان یک متغیر معرفی می‌کنیم
result = operation(3, 4)  # فراخوانی تابع انتخابی (می‌تواند add یا subtract باشد)
```

۵۷.۴ تعریف توابع داخلی (Nested Functions):

در پایتون، شما می‌توانید توابع را در داخل توابع دیگر تعریف کنید. این توابع داخلی به عنوان توابع محلی در دسترس هستند و می‌توانند به متغیرها و توابع دیگر در داخل تابع اصلی دسترسی داشته باشند.

مثال تعریف توابع داخلی:

```
def outer_function(x):
    def inner_function(y):
        return y * 2
    result = inner_function(x)
    return result
```

۵۷.۵ استفاده از Functions به عنوان ورودی:

در پایتون، می‌توانید یک Function را به عنوان ورودی به یک Function دیگر منتقل کنید. این به شما امکان می‌دهد توابع را پارامتری واحدتر و دارای قابلیت استفاده چندگانه کنید.

مثال استفاده از Function به عنوان ورودی:

```
def apply_operation(x, operation):
    result = operation(x)
    return result

def double(x):
    return x * 2

result = apply_operation(3, double)  # اعمال تابع double بر روی عدد ۳
```

۵۷.۶ استفاده از Functions به عنوان خروجی:

شما می‌توانید یک Function را به عنوان خروجی از یک Function دیگر بازگردانید. این مفهوم به شما امکان می‌دهد توابع را به عنوان نتیجه‌ای از توابع دیگر تولید کنید.

****مثال استفاده از Function به عنوان خروجی:****

```
def create_multiplier(factor):
    def multiplier(x):
        return x * factor
    return multiplier

double = create_multiplier(2)  # تولید یک تابع جدید (تابع double)
result = double(3)  # اعمال تابع double بر روی عدد ۳
```

****۵۷.۷ نتیجه‌گیری:****

در این فصل، ما انواع مختلفی از استفاده از Functions در پایتون را بررسی کردیم. از تعریف و استفاده از توابع تا ارسال توابع به عنوان ورودی و بازگرداندن توابع به عنوان خروجی، Functions یک اساس مهم در برنامه‌نویسی پایتون هستند و به شما امکان می‌دهند کد خود را به صورت مؤسسه‌ای و قابل توسعه نوشته و مدیریت کنید.

فصل ۵۸: بررسی Decorators در زبان Python

۵۸.۱ مقدمه به Decorators:

در پایتون، Decorators یکی از ابزارهای قدرتمندی هستند که به شما امکان می‌دهند عملکرد یک تابع را تغییر دهید یا به آن ویژگی‌های اضافی اعمال کنید. Decorators به شما امکان می‌دهند کد را بهبود بخشیده و بازسازی کنید بدون اینکه تغییرات مستقیم در تابع اصلی اعمال شود.

۵۸.۲ تعریف Decorator:

Decorator یک تابع است که یک تابع دیگر را به عنوان ورودی می‌گیرد و تغییراتی در عملکرد آن ایجاد می‌کند. Decorator معمولاً با استفاده از تابع‌ها، توابع داخلی و توابع بالامجموعه تعریف می‌شوند.

مثال تعریف Decorator:

```
def my_decorator(func):
    def wrapper():
        print("قبل از اجرای تابع")
        func()
        print("بعد از اجرای تابع")
    return wrapper

@my_decorator
def say_hello():
    print("سلام")

say_hello()
```

۵۸.۳ استفاده از Decorators:

برای استفاده از یک Decorator**، شما می‌توانید آن را با علامت** `@` **به عنوان یک تگ بر روی تابع مورد نظر قرار دهید. این کار** Decorator **را به تابع مورد نظر اعمال می‌کند.**

۵۸.۴ Decorators داخلی و استفاده چندگانه:

شما می‌توانید Decorators **داخلی (یعنی** Decorators **که خود** Decorator **هستند) و چندین** Decorator **را بر روی یک تابع اعمال کنید.**

مثال استفاده از چندین Decorator:

```
def uppercase_decorator(func):
    def wrapper():
        result = func()
        return result.upper()
    return wrapper

def exclamation_decorator(func):
    def wrapper():
        result = func()
        return result + "!"
    return wrapper

@exclamation_decorator
@uppercase_decorator
def say_hello():
    return "سلام"

greeting = say_hello()
```

۵۸.۵ استفاده از Decorators برای پارامترهای توابع:

شما می‌توانید Decorator **را برای توابع با پارامترهای ورودی نیز استفاده کنید.**

مثال استفاده از Decorator برای توابع با پارامترها:

```
def multiply_decorator(factor):
    def real_decorator(func):
        def wrapper(*args, **kwargs):
```

```
            result = func(*args, **kwargs)
            return result * factor
        return wrapper
    return real_decorator

@multiply_decorator(2)
def multiply_by_two(number):
    return number

result = multiply_by_two(5)
```

۵۸.۶ **نتیجه‌گیری:**

Decorators یک ویژگی قدرتمند در پایتون هستند که به شما امکان می‌دهند عملکرد توابع را بهبود بخشیده و بازسازی کنید. این ویژگی به شما اجازه می‌دهد تا تغییرات مرتبط با لجستیک، نمایش و ویژگی‌های دیگر را به توابع اعمال کنید بدون اینکه به تعریف تابع اصلی دست بزنید. Decorators یک ابزار مهم در برنامه‌نویسی پایتون هستند که برای ایجاد کد قابل خواندن، تمیز و قابل توسعه بسیار مفید هستند.

فصل ۵۹: بررسی API و نحوه کار با آن در زبان Python

۵۹.۱ مقدمه به API:

مخفف عبارت API "Application Programming Interface" **است و یک مجموعه از قوانین و دستورات است که برای ارتباط بین برنامه‌ها یا برنامه‌نویسان مورد استفاده قرار می‌گیرد.** APIها **به برنامه‌ها امکان می‌دهند با یکدیگر ارتباط برقرار کنند و داده‌ها و عملکردها را به اشتراک بگذارند.**

۵۹.۲ انواع API:

در دنیای برنامه‌نویسی، دو نوع اصلی API **وجود دارد:**

۱. API وب (Web APIs): این نوع APIها **برای ارتباط بین برنامه‌ها از طریق اینترنت استفاده می‌شوند. آنها به برنامه‌ها امکان می‌دهند با استفاده از** HTTP **درخواست‌ها را ارسال کرده و داده‌ها را دریافت کنند. مثال‌هایی از** API **وب شامل** APIهای **رسانه‌های اجتماعی،** APIهای **پرداخت آنلاین و** APIهای **ابری می‌شوند.**

۲. API محلی (Local APIs): این نوع APIها **برای ارتباط بین برنامه‌ها در یک سیستم محلی یا روی یک دستگاه استفاده می‌شوند. آنها به برنامه‌ها امکان می‌دهند تا با استفاده از توابع و دستورات دیگر برنامه‌ها اطلاعات و عملکردها را به اشتراک بگذارند.**

۵۹.۳ نحوه کار با API در Python:

برای کار با API در زبان Python، شما می‌توانید از کتابخانه‌های آماده مانند `requests` استفاده کنید. این کتابخانه‌ها به شما امکان می‌دهند درخواست‌های HTTP به API ارسال کرده و داده‌ها را دریافت کنید.

نمونه کد ارسال درخواست GET به یک API:

```python
import requests

# URL API مورد نظر
url = 'https://api.example.com/data'

# ارسال درخواست GET به API
response = requests.get(url)

# بررسی وضعیت درخواست
if response.status_code == 200:
    data = response.json()  # تبدیل داده‌های دریافتی به فرمت JSON
    print(data)
else:
    print('خطا در درخواست: ', response.status_code)
```

۵۹.۴ تعیین نوع درخواست:

APIها معمولاً درخواست‌های مختلفی را پشتیبانی می‌کنند، از جمله GET (دریافت داده)، POST (ارسال داده)، PUT (به‌روزرسانی داده) و DELETE (حذف داده). شما باید نوع درخواست مناسب را بر اساس نیاز خود انتخاب کنید.

نمونه کد ارسال درخواست POST به یک API:

```
import requests

# URL API مورد نظر
url = 'https://api.example.com/data'

# داده‌هایی که می‌خواهید ارسال کنید
data_to_send = {'key1': 'value1', 'key2': 'value2'}

# ارسال درخواست POST به API
response = requests.post(url, data=data_to_send)

# بررسی وضعیت درخواست
if response.status_code == 201:
    print('داده با موفقیت ارسال شد.')
else:
    print('خطا در درخواست: ', response.status_code)
```

۵۹.۵ نمونه‌های کاربردی API:

– استفاده از API وب برای دریافت اطلاعات از وب‌سایت‌ها معتبر (مانند API توییتر برای دریافت توییت‌ها).

– اتصال به سرویس‌های آنلاین (مثل API پرداخت آنلاین برای انجام تراکنش‌های مالی).

– ارسال درخواست‌های API به سرورهای دیگر برای دریافت یا ارسال داده‌ها (مانند APIهای مربوط به ابر).

– اتصال به دستگاه‌های محلی یا شبکه‌های دیگر با استفاده از API محلی (مثل API دستگاه‌های IoT).

۵۹.۶ نتیجه‌گیری:

APIها یک ابزار قدرتمند در

برنامه‌نویسی هستند که به شما امکان می‌دهند با دیگر برنامه‌ها، سرویس‌ها و دستگاه‌ها ارتباط برقرار کنید و داده‌ها و عملکردهای مختلف را به اشتراک بگذارید. با استفاده از کتابخانه‌های مناسب در زبان Python، شما می‌توانید به راحتی با APIها کار کنید و داده‌های مورد نیاز خود را دریافت یا ارسال کنید.

فصل ۶۰: کار با پایگاه داده‌ها در پایتون

۶۰.۱ مقدمه به کار با پایگاه داده‌ها:

کار با پایگاه داده‌ها یکی از جزئیات مهم در توسعه نرم‌افزارها است. پایگاه داده‌ها به برنامه‌ها امکان می‌دهند داده‌ها را ذخیره، بازیابی و مدیریت کنند. در این فصل، ما به بررسی چگونگی کار با پایگاه داده‌ها در زبان برنامه‌نویسی پایتون می‌پردازیم.

۶۰.۲ انواع پایگاه داده‌ها:

در زمینه پایگاه داده‌ها، دو نوع اصلی وجود دارد:

۱. **پایگاه داده‌های رابطه‌ای (Relational Databases): این نوع پایگاه داده‌ها از جداول و روابط بین داده‌ها استفاده می‌کنند. مثال‌هایی از پایگاه داده‌های رابطه‌ای شامل MySQL، PostgreSQL و SQLite می‌شوند.**

۲. **پایگاه داده‌های NoSQL: این نوع پایگاه داده‌ها از ساختارهای مختلفی برای ذخیره داده‌ها استفاده می‌کنند و اغلب برای مواردی که نیاز به ذخیره سازی داده‌های نامنظم یا بزرگ دارند مناسب هستند. مثال‌هایی از پایگاه داده‌های NoSQL شامل MongoDB و Cassandra می‌شوند.**

۶۰.۳ کار با پایگاه داده‌ها در پایتون:

در پایتون، شما می‌توانید از کتابخانه‌های مختلفی برای کار با پایگاه داده‌ها استفاده کنید. برخی از کتابخانه‌های معروف در این زمینه عبارتند از:

– SQLite: یک پایگاه داده رابطه‌ای کوچک و تحت‌وب ساده که به صورت داخلی در پایتون معرفی شده است.**

– MySQL و PostgreSQL: دو پایگاه داده رابطه‌ای پرکاربرد که می‌توانید با استفاده از کتابخانه‌های مختلفی در پایتون به آنها متصل شوید.**

– MongoDB: یک پایگاه داده NoSQL که برای ذخیره داده‌های ساختاری و نامنظم استفاده می‌شود.**

۶۰.۴ نمونه کاربردهای کار با پایگاه داده‌ها:

– ذخیره و بازیابی داده‌ها: از پایگاه داده‌ها برای ذخیره و بازیابی اطلاعات مانند اطلاعات کاربران، محصولات و سفارشات در یک فروشگاه آنلاین استفاده می‌شود.**

– سیستم‌های مدیریت محتوا (CMS): CMS**ها مانند WordPress از پایگاه داده‌ها برای ذخیره محتوا و تنظیمات استفاده می‌کنند.**

– تحلیل داده: در تحلیل داده، اطلاعات از پایگاه داده‌ها بازیابی و تجزیه و تحلیل می‌شوند تا الگوها و اطلاعات جدیدی به دست آید.**

۶۰.۵ نتیجه‌گیری:

کار با پایگاه داده‌ها یکی از جزئیات مهم در توسعه نرم‌افزارهاست. در این فصل، به مقدمه‌ای از کار با پایگاه داده‌ها در پایتون پرداختیم و انواع پایگاه داده‌ها و کتابخانه‌های مربوط به آنها را بررسی کردیم.

فصل ۶۱: بررسی ابتدایی GUI و ورود به Tkinter

۶۱.۱ مقدمه به GUI در پایتون:

GUI یا رابط کاربری گرافیکی، یکی از اصلی‌ترین اجزای برنامه‌های کاربردی مدرن است. با استفاده از GUI، کاربران می‌توانند با برنامه‌ها تعامل داشته و عملیات مختلف را به صورت گرافیکی انجام دهند. Tkinter یکی از کتابخانه‌های معروف برای ایجاد GUI در پایتون است.

۶۱.۲ شروع با Tkinter:

برای شروع کار با Tkinter در پایتون، شما نیاز به وارد کردن این کتابخانه دارید:

```
import tkinter as tk
```

سپس یک پنجره اصلی (Main Window) ایجاد می‌کنید:

```
root = tk.Tk()
```

۶۱.۳ ایجاد المان‌های GUI:

در Tkinter، شما می‌توانید انواع مختلفی از المان‌های GUI ایجاد کنید. برخی از این المان‌ها عبارتند از:

- Label: **برچسب متنی که می‌تواند متن یا تصویر را نمایش دهد.**
- Button: **دکمه‌ای که کاربر می‌تواند کلیک کند.**
- Entry: **جعبه متن برای ورود اطلاعات توسط کاربر.**
- Text: **یک جعبه متن بزرگ برای نمایش و ویرایش متن چند خطه.**
- Canvas: **یک صفحه نقاشی برای ترسیم اشیاء گرافیکی.**

۶۱.۴ تنظیم ویژگی‌ها و رفتار المان‌ها:

بعد از ایجاد المان‌های GUI، شما می‌توانید ویژگی‌ها و رفتارهای آنها را تنظیم کنید. برای مثال، می‌توانید متن یک Label تغییر دهید یا به یک دکمه عملیاتی (Command) اختصاص دهید.

۶۱.۵ تنظیم طراحی و چیدمان:

شما می‌توانید المان‌های GUI را با استفاده از مدیران چیدمان (Layout Managers) در Tkinter به صورت دقیق در پنجره خود چیده و طراحی کنید. مدیران چیدمان معمول شامل `pack()`, `grid`, و `place()` هستند.

۶۱.۶ برنامه‌های GUI ساده:

به عنوان مثال ساده، می‌توانید یک برنامه‌ی تستی GUI بسازید که یک Label و یک دکمه داشته باشد و هنگام کلیک بر روی دکمه، متن Label تغییر کند.

۶۱.۷ نتیجه‌گیری:

در این فصل، با مفاهیم ابتدایی GUI و ورود به Tkinter در پایتون آشنا شدیم. Tkinter یک کتابخانه قدرتمند برای ایجاد رابط کاربری گرافیکی در پایتون است و می‌تواند به شما کمک کند برنامه‌های تعاملی و جذاب بسازید. در فصول آینده، ما به بررسی عمیق‌تر و پیشرفته‌تر از Tkinter و ایجاد برنامه‌های GUI پیچیده‌تر می‌پردازیم.

فصل ۶۲: بررسی دقیق Tkinter و ساخت دکمه، لیبل و جعبه متنی

۶۲.۱ مقدمه به Tkinter:

Tkinter یک کتابخانه استاندارد برای ایجاد رابط کاربری گرافیکی (GUI) در پایتون است. در این فصل، ما به بررسی دقیق‌تر Tkinter و ایجاد سه المان مهم GUI یعنی دکمه (Button)، لیبل (Label) و جعبه متنی (Entry) می‌پردازیم.

۶۲.۲ ساخت یک پنجره جدید:

برای شروع کار با Tkinter، شما باید یک پنجره اصلی ایجاد کنید. این پنجره معمولاً به عنوان پنجره اصلی یا پنجره اصلی تعریف می‌شود.

```
import tkinter as tk

root = tk.Tk()  # ایجاد پنجره اصلی
```

۶۲.۳ ساخت دکمه:

برای ساخت یک دکمه در Tkinter، از کلاس `Button` استفاده می‌کنیم.

```
button = tk.Button(root, text="کلیک کنید")  # ایجاد یک دکمه با متن "کلیک کنید"
button.pack()  # نمایش دکمه در پنجره
```

۶۲.۴ ساخت لیبل:

لیبل برای نمایش متن یا تصویر در پنجره استفاده می‌شود.

```
label = tk.Label(root, text="سلام، دنیا!")  # ایجاد یک لیبل با متن "سلام، دنیا!"
label.pack()  # نمایش لیبل در پنجره
```

۶۲.۵ ساخت جعبه متنی:

جعبه متنی یک المان است که به کاربر اجازه می‌دهد متن ورودی را وارد کند.

```
entry = tk.Entry(root)  # ایجاد یک جعبه متنی خالی
entry.pack()  # نمایش جعبه متنی در پنجره
```

۶۲.۶ نتیجه‌گیری:

در این فصل، با ساخت سه المان اساسی GUI در Tkinter یعنی دکمه، لیبل و جعبه متنی آشنا شدیم. این المان‌ها به شما امکان می‌دهند اطلاعات را نمایش دهید و از کاربر دریافت کنید. در فصول آتی، ما به بررسی المان‌های دیگر Tkinter و ایجاد برنامه‌های GUI پیچیده‌تر می‌پردازیم.

فصل ۶۳: بررسی ویجت Entry و کار با آن در Tkinter

۶۳.۱ ویجت Entry:

ویجت Entry در Tkinter یک جعبه متنی (Text Box) است که به کاربر اجازه می‌دهد متن ورودی را وارد کند. این ویجت بسیار مفید برای دریافت ورودی از کاربر در برنامه‌های GUI است.

۶۳.۲ ایجاد ویجت Entry:

برای ایجاد یک ویجت Entry، از کلاس `Entry` استفاده می‌کنیم و آن را در پنجره اصلی یا فریم مورد نظر ایجاد می‌کنیم.

```
import tkinter as tk
```

```
root = tk.Tk()  # ایجاد پنجره اصلی

entry = tk.Entry(root)  # ایجاد یک ویجت Entry خالی
entry.pack()  # نمایش ویجت Entry در پنجره
```

****۶۳.۳ دریافت مقدار ورودی:****

برای دریافت مقدار ورودی کاربر از ویجت Entry**، می‌توانیم از متد** `get()` **استفاده کنیم.**

```
value = entry.get()  # دریافت مقدار وارد شده توسط کاربر
print("مقدار وارد شده:", value)
```

****۶۳.۴ تنظیم مقدار اولیه:****

می‌توانیم مقدار اولیه ویجت Entry **را با استفاده از متد** `insert()` **تنظیم کنیم.**

```
entry.insert(0, "مقدار اولیه")  # تنظیم مقدار اولیه برای ویجت Entry
```

****۶۳.۵ پاک کردن ورودی:****

برای پاک کردن محتوای ویجت Entry**، می‌توانیم از متد** `delete()` **استفاده کنیم.**

```
entry.delete(0, tk.END)  # پاک کردن محتوای ویجت Entry
```

****۶۳.۶ نتیجه‌گیری:****

ویجت Entry **در** Tkinter **ابزاری قدرتمند برای دریافت و نمایش متن ورودی از کاربر است. با استفاده از متدهای** `get()`، `insert()`، **و** `delete()` **می‌توانید با ویجت** Entry **کار کنید و اطلاعات مورد نیاز را از کاربر دریافت کنید و یا نمایش دهید. این ویجت معمولاً در ایجاد فرم‌ها و ورود اطلاعات به برنامه‌های** GUI **استفاده می‌شود.**

فصل ۶۴: پیاده‌سازی بازی X و O با Tkinter

۶۴.۱ مقدمه به بازی X و O:

در این فصل، ما یک نسخه ساده از بازی X و O (یا Tic-Tac-Toe) را با استفاده از کتابخانه Tkinter در پایتون پیاده‌سازی می‌کنیم. این بازی معمولاً بین دو نفر انجام می‌شود و هدف آن ایجاد یک ترتیب از سه علامت یکسان (X یا O) در یک خط افقی، عمودی یا مورب است.

۶۴.۲ ایجاد پنجره بازی:

ابتدا یک پنجره بازی Tkinter **ایجاد می‌کنیم و سه خانه با ابعاد ۳**x۳ **برای بازی ایجاد می‌کنیم.**

```
import tkinter as tk

root = tk.Tk()  # ایجاد پنجره بازی

# ایجاد سه خانه بازی
button1 = tk.Button(root, text=" ", font=("Helvetica", 20), height=2,
width=5)
button2 = tk.Button(root, text=" ", font=("Helvetica", 20), height=2,
width=5)
button3 = tk.Button(root, text=" ", font=("Helvetica", 20), height=2,
width=5)

button1.grid(row=0, column=0)
button2.grid(row=0, column=1)
button3.grid(row=0, column=2)
```


****۶۴.۳ اضافه کردن علامت‌ها:****

سپس، ما علامت‌های X **و** O **را به بازی اضافه می‌کنیم و وقتی کاربر روی یک خانه کلیک می‌کند، علامت مربوطه در آن خانه قرار می‌گیرد.**

```
# اضافه کردن علامت‌ها به بازی
turn = "X"  # نوبت بازیکن X

def place_sign(row, col):
    global turn
    if turn == "X":
```

```
            buttons[row][col].config(text="X")
            turn = "O"
        else:
            buttons[row][col].config(text="O")
            turn = "X"

# تعریف تابع برای هر خانه
def on_click(row, col):
    if buttons[row][col]["text"] == " ":
        place_sign(row, col)

# اضافه کردن تابع برای هر خانه
button1.config(command=lambda: on_click(0, 0))
button2.config(command=lambda: on_click(0, 1))
button3.config(command=lambda: on_click(0, 2))
```

۶۴.۴ نتیجه‌گیری:

در این فصل، ما یک نسخه ساده از بازی X و O با استفاده از کتابخانه Tkinter در پایتون پیاده‌سازی کردیم. این بازی ابتدایی و ساده بود، اما شما می‌توانید آن را بهبود دهید و ویژگی‌های بیشتری به آن اضافه کنید، مانند بررسی برنده شدن یا تساوی شدن بازی.

فصل ۶۵: مرور کلی و تاکید بر نکات مهم

۶۵.۱ مرور مفاهیم مهم:

در این فصل، ما مفاهیم مهمی را که در طول این دوره آموختیم مرور می‌کنیم. این مفاهیم شامل اصول پایه‌ای برنامه‌نویسی در پایتون، متغیرها، داده‌ها و نوع‌های داده مختلف، عبارات شرطی و حلقه‌ها، توابع و ماژول‌ها، کتابخانه‌های مفید، کار با کتابخانه‌های گرافیکی مانند Tkinter و مفاهیم پیشرفته‌تر مانند مفهوم شی گرایی.

۶۵.۲ تاکید بر نکات مهم:**

– حتماً به تاریخچه پایتون و مزیت‌های آن در برنامه‌نویسی توجه کنید.

– مراعات کنید که پایتون را بصورت صحیح نصب کنید و از محیط توسعه مناسبی استفاده کنید.

– نحوه مدیریت فایل‌ها و پروژه‌ها در ویژوال استودیو کد را یاد بگیرید.

– با مفاهیم کلی داده‌ها و عملیات بر روی آن‌ها مانند جمع و تفریق آشنا شوید.

– عبارات شرطی و حلقه‌ها را به دقت بیاموزید تا کنترل بهتری برنامه‌هایتان داشته باشید.

– توانمندی‌های توابع را درک کنید و نحوه تعریف و فراخوانی توابع را مسلط شوید.

– با مفاهیم مهم مانند لیست‌ها، دیکشنری‌ها، تاپل‌ها و مجموعه‌ها آشنا شوید.

– کتابخانه‌های پایتون را برای افزایش بهره‌وری برنامه‌هایتان بهره ببرید.

– در مورد توابع و کتابخانه‌های گرافیکی مانند Tkinter مهارت‌های خود را توسعه دهید.

– به مفاهیم پیشرفته‌تر مانند شی گرایی و پردازش استثنائات توجه کنید.

۶۵.۳ پیام انگیزشی:**

برنامه‌نویسی یک مسیر هیجان‌انگیز است که همچنان با یادگیری و ارتقاء مهارت‌ها ادامه پیدا می‌کند. هیچ وقت ترس از اشتباه کردن نداشته باشید، زیرا هر اشتباهی یک درس جدید است. پیشرفت خود را با مطالعه، تمرین و پروژه‌های عملی ایجاد کنید. همیشه به سوالات خود جواب دهید و از منابع آموزشی بهره ببرید. با تلاش و پشتکار، می‌توانید به یک برنامه‌نویس حرفه‌ای تبدیل شوید و پروژه‌های جذابی را ایجاد کنید. آغاز کنید، مسیرتان را انتخاب کنید و به سوی موفقیت حرکت کنید.